SunFounder Kepler Kit for Raspberry Pi Pico W

Release 1.0

www.sunfounder.com

Aug 13, 2024

CONTENTS

1 Raspberry Pi Pico W **3**
- 1.1 Features . . . 4
- 1.2 Pico's Pins . . . 5

2 What is Included in This Kit **9**
- 2.1 Breadboard . . . 10
- 2.2 Jumper Wires . . . 11
- 2.3 Resistor . . . 12
- 2.4 Transistor . . . 15
- 2.5 Capacitor . . . 18
- 2.6 Diode . . . 19
- 2.7 Li-po Charger Module . . . 21
- 2.8 74HC595 . . . 23
- 2.9 TA6586 - Motor Driver Chip . . . 25
- 2.10 LED . . . 27
- 2.11 RGB LED . . . 28
- 2.12 LED Bar Graph . . . 31
- 2.13 7-segment Display . . . 33
- 2.14 4-Digit 7-Segment Display . . . 35
- 2.15 LED Dot Matrix . . . 37
- 2.16 I2C LCD1602 . . . 39
- 2.17 WS2812 RGB 8 LEDs Strip . . . 42
- 2.18 Buzzer . . . 43
- 2.19 DC Motor . . . 45
- 2.20 Servo . . . 47
- 2.21 DC Water Pump . . . 49
- 2.22 Relay . . . 50
- 2.23 Button . . . 52
- 2.24 Micro Switch . . . 54
- 2.25 Slide Switch . . . 56
- 2.26 Potentiometer . . . 57
- 2.27 Infrared Receiver . . . 59
- 2.28 Joystick Module . . . 61
- 2.29 4x4 Keypad . . . 63
- 2.30 MPR121 Module . . . 64
- 2.31 MFRC522 Module . . . 66
- 2.32 Photoresistor . . . 67
- 2.33 Thermistor . . . 68
- 2.34 Tilt Switch . . . 70
- 2.35 Reed Switch . . . 72

2.36 PIR Motion Sensor Module . . . 73
2.37 Water Level Sensor Module . . . 75
2.38 Ultrasonic Module . . . 76
2.39 DHT11 Humiture Sensor . . . 78
2.40 MPU6050 Module . . . 80

3 Electronic Circuit **83**
3.1 Hello, Breadboard! . . . 85
3.2 Beware of short circuits . . . 87
3.3 Direction of the circuit . . . 87
3.4 Protection of the circuit . . . 88

4 For MicroPython User **91**
4.1 1.1 Introduction of MicroPython . . . 91
4.2 1.2 Install Thonny IDE . . . 93
4.3 1.3 Install MicroPython on Your Pico . . . 95
4.4 1.4 Upload the Libraries to Pico . . . 99
4.5 1.5 Quick Guide on Thonny . . . 101
4.6 1.6 (Optional) MicroPython Basic Syntax . . . 110
4.7 2.1 Hello, LED! . . . 141
4.8 2.2 Display the Level . . . 147
4.9 2.3 Fading LED . . . 151
4.10 2.4 Colorful Light . . . 155
4.11 2.5 Reading Button Value . . . 160
4.12 2.6 Tilt It . . . 163
4.13 2.7 Toggle Left and Right . . . 166
4.14 2.8 Press Gently . . . 169
4.15 2.9 Feel the Magnetism . . . 172
4.16 2.10 Detect Human Movement . . . 175
4.17 2.11 Turn the Knob . . . 180
4.18 2.12 Feel the Light . . . 184
4.19 2.13 Thermometer . . . 187
4.20 2.14 Feel the Water Level . . . 191
4.21 2.15 Two Kinds of Transistors . . . 195
4.22 2.16 Control Another Circuit . . . 199
4.23 3.1 Beep . . . 204
4.24 3.2 Custom Tone . . . 207
4.25 3.3 RGB LED Strip . . . 211
4.26 3.4 Liquid Crystal Display . . . 214
4.27 3.5 Small Fan . . . 218
4.28 3.6 Pumping . . . 221
4.29 3.7 Swinging Servo . . . 224
4.30 4.1 Toggle the Joystick . . . 228
4.31 4.2 4x4 Keypad . . . 231
4.32 4.3 Electrode Keyboard . . . 237
4.33 5.1 Microchip - 74HC595 . . . 240
4.34 5.2 Number Display . . . 243
4.35 5.3 Time Counter . . . 248
4.36 5.4 8x8 Pixel Graphics . . . 253
4.37 6.1 Measuring Distance . . . 259
4.38 6.2 Temperature - Humidity . . . 264
4.39 6.3 6-axis Motion Tracking . . . 268
4.40 6.4 IR Remote Control . . . 271
4.41 6.5 Radio Frequency Identification . . . 277

4.42 7.1 Light Theremin . . . 281
4.43 7.2 Room Temperature Meter . . . 284
4.44 7.3 Alarm Siren Lamp . . . 287
4.45 7.4 Passenger Counter . . . 292
4.46 7.5 GAME - 10 Second . . . 296
4.47 7.6 Traffic Light . . . 301
4.48 7.7 Guess Number . . . 306
4.49 7.8 RFID Music Player . . . 312
4.50 7.9 Fruit Piano . . . 318
4.51 7.10 Reversing Aid . . . 323
4.52 7.11 Somatosensory Controller . . . 328
4.53 7.12 Digital Bubble Level . . . 333

5 IoT Projects **339**
5.1 1. Access to the Network . . . 339
5.2 2. Follow the @CheerLights . . . 348
5.3 3. Security System via @IFTTT . . . 352
5.4 4. Real-time Weather From @OpenWeatherMap . . . 365
5.5 5. Cloud Calling System with @MQTT . . . 374
5.6 6. Cloud Player with @MQTT . . . 380
5.7 7. Setup a Web Server . . . 386
5.8 8. Building Web APP with @Anvil . . . 394
5.9 9. Play in @SunFounder Controller . . . 414
5.10 10. Plant Monitor in @SunFounder Controller . . . 423

6 For Arduino User **427**
6.1 1.1 Install Arduino IDE(Important) . . . 427
6.2 1.2 Introduce of Arduino IDE . . . 433
6.3 1.3 Setting Up the Raspberry Pi Pico W (Important) . . . 435
6.4 1.4 Install libraries (Important) . . . 442
6.5 2.1 - Hello, LED! . . . 444
6.6 2.2 - Display the Level . . . 447
6.7 2.3 - Fading LED . . . 451
6.8 2.4 - Colorful Light . . . 455
6.9 2.5 - Reading Button Value . . . 459
6.10 2.6 - Tilt It . . . 463
6.11 2.7 - Toggle Left and Right . . . 466
6.12 2.8 - Press Gently . . . 469
6.13 2.9 - Feel the Magnetism . . . 471
6.14 2.10 - Detect Human Movement . . . 474
6.15 2.11 - Turn the Knob . . . 477
6.16 2.12 - Feel the Light . . . 482
6.17 2.13 - Thermometer . . . 484
6.18 2.14 - Feel the Water Level . . . 488
6.19 2.15 - Two Kinds of Transistors . . . 491
6.20 2.16 - Control Another Circuit . . . 495
6.21 3.1 - Beep . . . 499
6.22 3.2 - Custom Tone . . . 502
6.23 3.3 WS2812 RGB Strip . . . 505
6.24 3.4 - Liquid Crystal Display . . . 509
6.25 3.5 - Small Fan . . . 513
6.26 3.6 - Pumping . . . 516
6.27 3.7 - Swinging Servo . . . 519
6.28 4.1 - Toggle the Joystick . . . 523

6.29 4.2 4x4 Keypad . . . 526
6.30 4.3 - Electrode Keyboard . . . 529
6.31 5.1 Microchip - 74HC595 . . . 532
6.32 5.2 - Number Display . . . 535
6.33 5.3 - Time Counter . . . 539
6.34 5.4 - 8x8 Pixel Graphics . . . 543
6.35 6.1 - Measuring Distance . . . 549
6.36 6.2 - Temperature - Humidity . . . 552
6.37 6.3 - 6-axis Motion Tracking . . . 556
6.38 6.4 - IR Remote Control . . . 559
6.39 6.5 - Radio Frequency Identification . . . 562

7 For Piper Make **567**
7.1 1.1 Set up the Pico . . . 567
7.2 1.2 Quick Guide on Piper Make . . . 571
7.3 1.3 How to Save or Import Code? . . . 578
7.4 2.1 Blink LED . . . 581
7.5 2.2 Button . . . 584
7.6 2.3 Service Bell . . . 587
7.7 2.4 Rainbow Light . . . 590
7.8 2.5 Drum Kit . . . 593
7.9 2.6 Smart Water Tank . . . 596
7.10 2.7 Swing Servo . . . 601
7.11 2.8 Light Intensity Display . . . 604
7.12 2.9 Lucky Cat . . . 607
7.13 2.10 Flowing LEDs . . . 611
7.14 2.11 Reversing System . . . 619
7.15 2.12 Smart Fan . . . 624
7.16 2.13 Reaction Game . . . 628

8 Video Courses **637**
8.1 lesson 1: Write Your First Program for Absolute Beginners . . . 637
8.2 lesson 2: Understanding and Using Breadboards . . . 638
8.3 lesson 3: Understanding and Using Binary Numbers . . . 639
8.4 lesson 4: Create a Binary Counter Using the Pico W . . . 639
8.5 lesson 5: Reading Analog Voltages Using a Potentiometer . . . 640
8.6 lesson 6: Understanding If Statements in MicroPython . . . 641
8.7 lesson 7: Controlling 3 LED with a Potentiometer in Micropython . . . 641
8.8 lesson 8: Compound Conditionals and If Statements in MicroPython . . . 642
8.9 lesson 9: Getting Analog Output Using PWM (Pulse Width Modulation) . . . 643
8.10 lesson 10: Controlling a Dimmable LED with a Potentiometer . . . 644
8.11 lesson 11: Understanding and Controlling an RGB LED in MicroPython . . . 644
8.12 lesson 12: Create a Dimmable LED in Micropython . . . 645
8.13 lesson 13: User Specified RGB LED Colors Using Micropython . . . 646
8.14 lesson 14: Understanding Variable Types and Arrays in Micropython . . . 646
8.15 lesson 15: For Loops in Micropython . . . 647
8.16 lesson 16: Sequence of Colors in RGB LED With Micropython . . . 648
8.17 lesson 17: Buttons and Pull Up Resistors in Micropython . . . 649
8.18 lesson 18: Push Button Control of LEDs in Micropython . . . 650
8.19 lesson 19: Button Control of RGB LED in Micropython . . . 651
8.20 lesson 20: Using the DHT11 Temperature and Humidity Sensor in MicroPython . . . 652
8.21 lesson 21: Temperature and Humidity Measurements with Toggle Switch . . . 653
8.22 lesson 22: Using an LCD Display with the Pico W . . . 654
8.23 lesson 23: Temperature and Humidity Sensor with LCD Display . . . 655

8.24 lesson 24: Power Portable Projects with LiPo Rechargeable Battery . . . 656
8.25 lesson 25: Getting Started with OLED 1306 in Micropython . . . 657
8.26 lesson 26: Drawing a Circle on the OLED 1306 Display . . . 658
8.27 lesson 27: Creating Lissajous Patterns on an OLED Display . . . 659
8.28 lesson 28: Connecting the Raspberry Pi Pico W to WiFi . . . 660
8.29 lesson 29: Simple Client Server Project to Control RGB LED . . . 661
8.30 lesson 30: Project to Connect Your Raspberry Pi Pico W to the Internet . . . 662
8.31 lesson 31: Sensorless Remote Weather Station Project . . . 663
8.32 lesson 32: Mobile Weather Station Project . . . 664
8.33 lesson 33 : Display the HSV Color Wheel on an RGB LED . . . 665
8.34 lesson 34 : Convert HSV to RGB in Micropython . . . 666
8.35 lesson 35 : Remote Weather Station with RGB LED Temperature Indicator . . . 667
8.36 lesson 36 : Control a Servo With MicroPython . . . 668
8.37 lesson 37 : Control a Servo With a Potentiometer in MicroPython . . . 669
8.38 lesson 38 : Learn Trigonometry Fundamentals in One Session (EASY) . . . 670
8.39 lesson 39 : Practical Trigonometry for Easy Problem Solving . . . 671
8.40 lesson 40 : Measuring Acceleration with the MPU6050 Six Axis IMU . . . 672
8.41 lesson 41 : Inferring Tilt From a 3 Axis Accelerometer . . . 673
8.42 lesson 42 : Measuring Tilt With an MPU6050 Accelerometer . . . 674
8.43 lesson 43 : Measure Pitch and Roll Using a 3 Axis Accelerometer . . . 675
8.44 lesson 44 : Measure Simultaneous 0 G on All Three Axis of MPU6050 . . . 675
8.45 lesson 45 : Calculating Height from a Dropped Object in Freefall . . . 676
8.46 lesson 46 : Build a 2 Axis Tilt Meter with Display Using the MPU6050 . . . 677
8.47 lesson 47 : Improving Sensor Data With a Low Pass Filter . . . 678
8.48 lesson 48 : Measuring Rotation Using the Gyros on the MPU6050 . . . 679
8.49 lesson 49 : Improving IMU Performance with a Complimentary Filter . . . 680
8.50 lesson 50 : Removing Long Term Steady State Error from Sensor Data . . . 681
8.51 lesson 51 : Ultimate Pitch and Roll Gadget Using the MPU6050 . . . 682
8.52 lesson 52 : Controlling Neopixels in MicroPython . . . 683
8.53 lesson 53 : Animating NeoPixel Strips Using MicroPython . . . 683
8.54 lesson 54 : Animated Dynamic Rainbow on NeoPixel in MicroPython . . . 684
8.55 lesson 55 : Create a Dynamic Rainbow With Neopixels in MicroPython . . . 685
8.56 lesson 56 : Using a Joystick With MicroPython . . . 686
8.57 lesson 57 : Calibrating a Joystick in MicroPython . . . 687
8.58 lesson 58 : Determine Angular Position of a Joystick in MicroPython . . . 687
8.59 lesson 59 : Controlling a Servo with a Joystick . . . 688
8.60 lesson 60 : Control NeoPixel Colors with a Joystick in MicroPython . . . 689
8.61 lesson 61 : How to Write Modular Code With Micropython Functions . . . 689
8.62 lesson 62 : Create Functions to Average and Sort Grades in Micropythons . . . 690
8.63 lesson 63 : Object Oriented Programming in MicroPython with Classes and Methods . . . 691
8.64 lesson 64 : Object Oriented Programming Example in MicroPython with LEDs . . . 691
8.65 lesson 65 : Create a Servo Class and Method in MicroPython . . . 692
8.66 lesson 66 : Create Your Own Libraries in Micropython . . . 693
8.67 lesson 67 : Use Both Cores on Your Pi Pico with MicroPython . . . 694
8.68 lesson 68 : MicroPython Multicore Threading Example with LEDs and Servo . . . 695
8.69 lesson 69 : Cleanly Exit MicroPython Threads When Program Terminates . . . 695
8.70 lesson 70 : Example of Cleanly Exiting Dual Core Program in MicroPython . . . 696
8.71 lesson 71 : Allow Thread to Complete Task Before Termination . . . 697
8.72 lesson 72 : Understanding Using Dictionaries in MicroPython . . . 698
8.73 lesson 73 : Control a RGB LED Using a Dictionary in MicroPython . . . 699
8.74 lesson 74 : Create a MicroPython Class for Controlling RGB LED . . . 700

9 FAQ **701**
9.1 Arduino . . . 701

9.2 MicroPython 701
9.3 Piper Make 702

10 Copyright Notice **703**

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

Thanks for choosing our .

Note: This document is available in the following languages.

-
-
-

Please click on the respective links to access the document in your preferred language.

Thank you for choosing the SunFounder Kepler Kit

This is an learning kit based on the Raspberry Pi Pico W.

Raspberry Pi Pico W adds on-board single-band 2.4GHz wireless interfaces (802.11n) using the Infineon CYW4343 while retaining the Pico form factor, so in addition to the basic GPIO function, it can also connect to the network so we can use it for some IoT projects. For example, using IFTTT for a security system, building a cloud player and a cloud service bell system using MQTT and so on.

This kit contains a variety of components, including displays, sounds, drivers, controllers and sensors, which allows you to gain a comprehensive understanding of electronic devices.

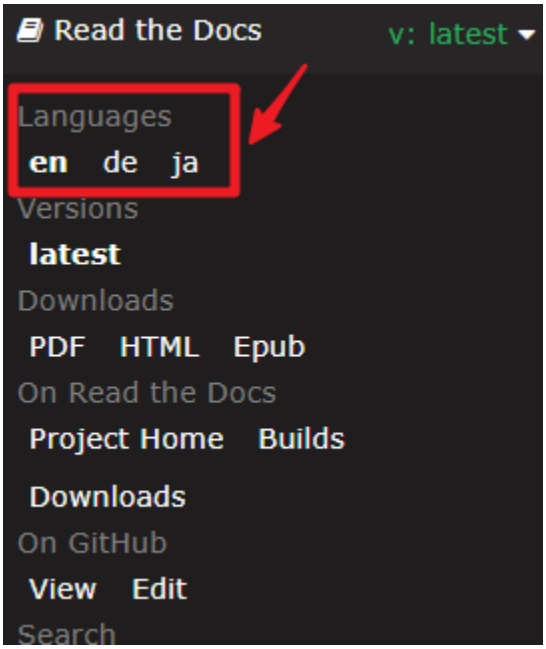
Additionally, we offer three programming languages. MicroPython, C/C++ (Arduino) and Piper Make are all available. To help you get started quickly, each language has targeted and interesting projects, so you can pick one that meets your needs.

Please feel free to e-mail us if you are interested in learning other projects we don't have. We will update our online tutorials as soon as possible.

Here is the email: service@sunfounder.com.

About the display language

This document is available in other languages as well. To switch the display language, kindly click on the **Read the Docs** icon located in the lower left corner of the page.



Source Code

- `SunFounder Kepler Kit`
- Or check out the code at Kepler Kit - GitHub

Content

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

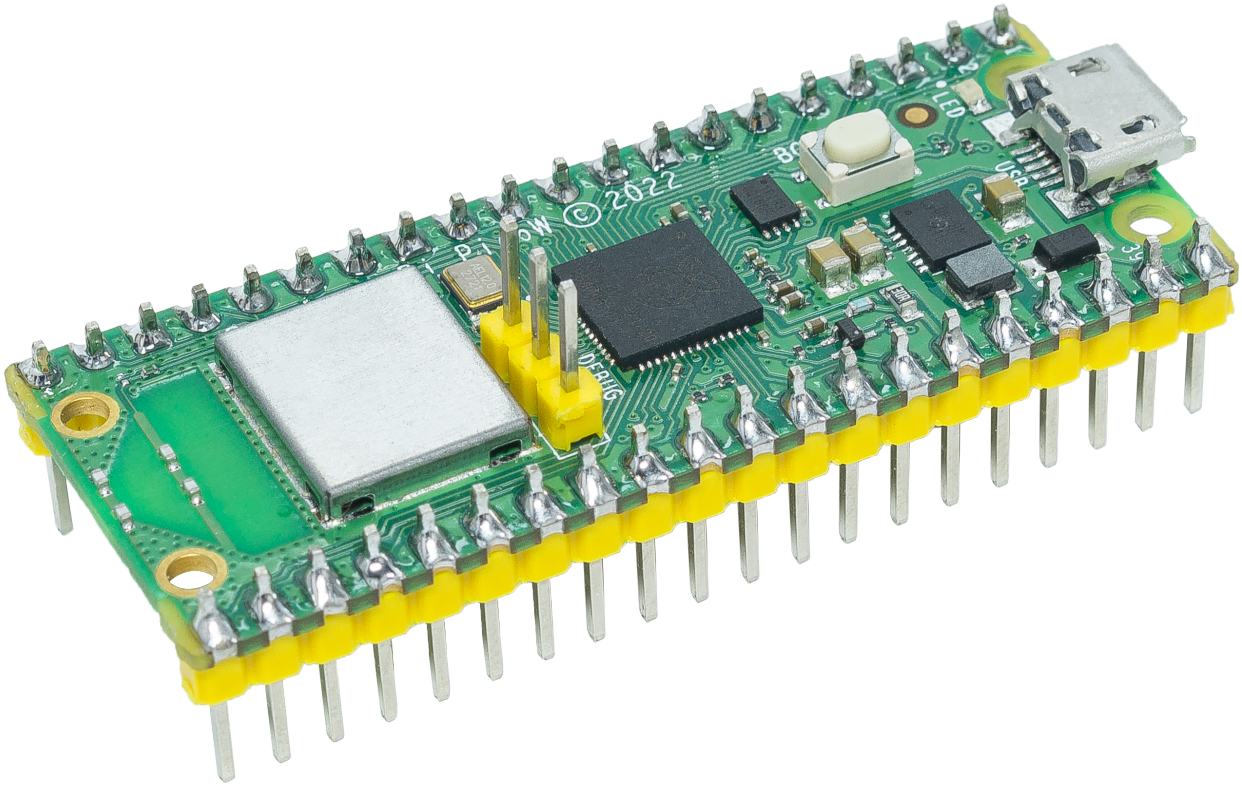
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

CHAPTER

ONE

RASPBERRY PI PICO W



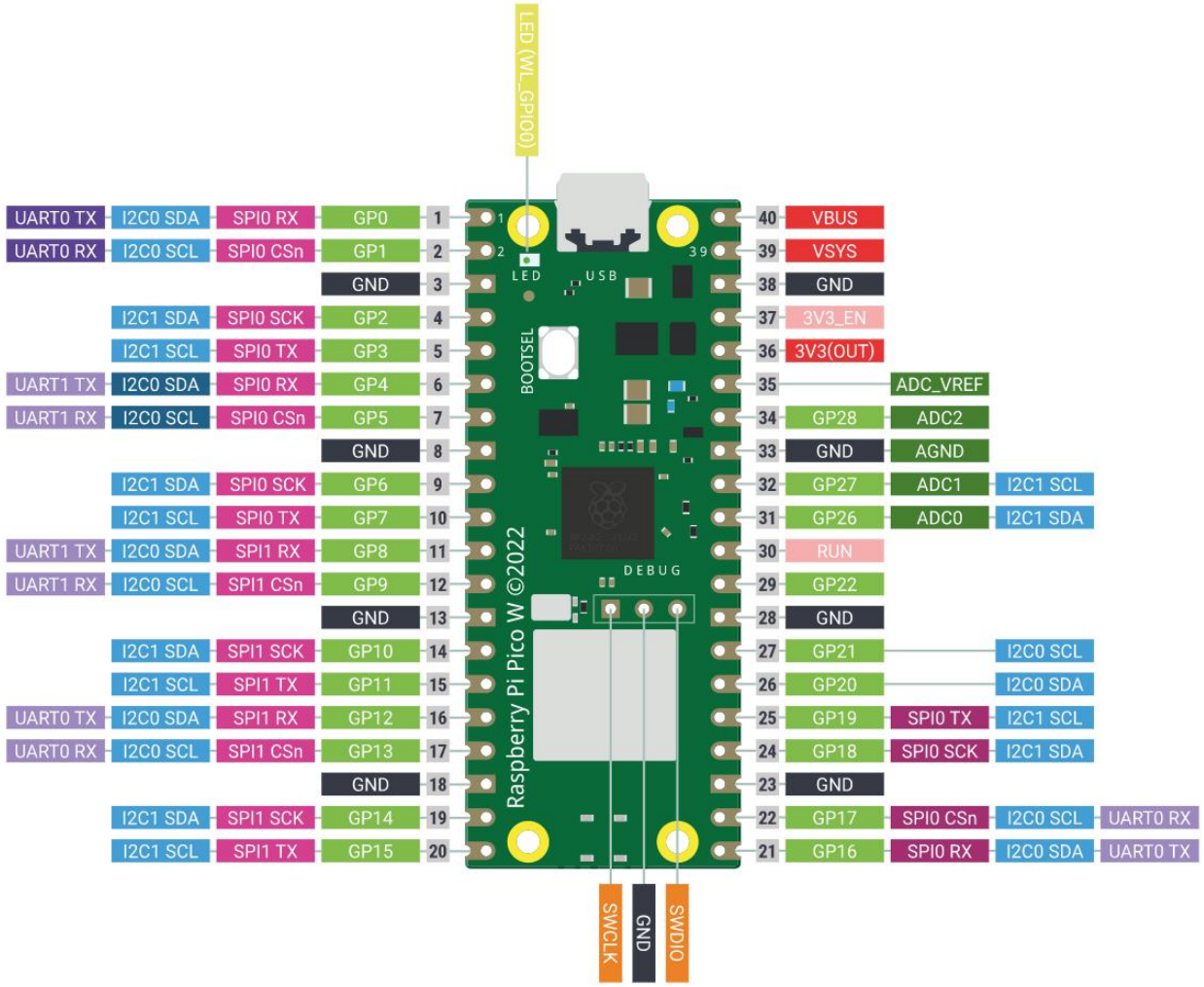
Raspberry Pi Pico W brings wireless connectivity to the best-selling Raspberry Pi Pico product line. Built around our RP2040 silicon platform, Pico products bring our signature values of high performance, low cost, and ease of use to the microcontroller space.

Raspberry Pi Pico W offers 2.4GHz 802.11 b/g/n wireless LAN support, with an on-board antenna, and modular compliance certification. It is able to operate in both station and access-point modes. Full access to network functionality is available to both C and MicroPython developers. Raspberry Pi Pico W pairs RP2040 with 2MB of flash memory, and a power supply chip supporting input voltages from 1.8–5.5V. It provides 26 GPIO pins, three of which can function as analogue inputs, on 0.1”-pitch through-hole pads with castellated edges. Raspberry Pi Pico W is available as an individual unit, or in 480-unit reels for automated assembly

1.1 Features

- 21 mm x 51 mm form factor
- RP2040 microcontroller chip designed by Raspberry Pi in the UK
- Dual-core Arm Cortex-M0+ processor, flexible clock running up to 133 MHz
- 264kB on-chip SRAM
- 2MB on-board QSPI flash
- 2.4GHz 802.11n wireless LAN
- 26 multifunction GPIO pins, including 3 analogue inputs
- 2 x UART, 2 x SPI controllers, 2 x I2C controllers, 16 x PWM channels
- 1 x USB 1.1 controller and PHY, with host and device support
- 8 x Programmable I/O (PIO) state machines for custom peripheral support
- Supported input power 1.8-5.5V DC
- Operating temperature -20°C to +70°C
- Castellated module allows soldering direct to carrier boards
- Drag-and-drop programming using mass storage over USB
- Low-power sleep and dormant modes
- Accurate on-chip clock
- Temperature sensor
- Accelerated integer and floating-point libraries on-chip

1.2 Pico's Pins



Name	Description	Function
GP0-GP28	General-purpose input/output pins	Act as either input or output and have no fixed purpose of their own
GND	0 volts ground	Several GND pins around Pico W to make wiring easier.
RUN	Enables or diables your Pico	Start and stop your Pico W from another microcontroller.
GPxx_ADCx	General-purpose input/output or analog input	Used as an analog input as well as a digital input or output – but not both at the same time.
ADC_VREF	Analog-to-digital converter (ADC) voltage reference	A special input pin which sets a reference voltage for any analog inputs.
AGND	Analog-to-digital converter (ADC) 0 volts ground	A special ground connection for use with the ADC_VREF pin.
3V3(O)	3.3 volts power	A source of 3.3V power, the same voltage your Pico W runs at internally, generated from the VSYS input.
3v3(E)	Enables or disables the power	Switch on or off the 3V3(O) power, can also switches your Pico W off.
VSYS	2-5 volts power	A pin directly connected to your Pico's internal power supply, which cannot be switched off without also switching Pico W off.
VBUS	5 volts power	A source of 5 V power taken from your Pico's micro USB port, and used to power hardware which needs more than 3.3 V.

The best place to find everything you need to get started with your Raspberry Pi Pico W is here

Or you can click on the links below:

- Raspberry Pi Pico W product brief
- Raspberry Pi Pico W datasheet
- Getting started with Raspberry Pi Pico: C/C++ development
- Raspberry Pi Pico C/C++ SDK
- API-level Doxygen documentation for the Raspberry Pi Pico C/C++ SDK
- Raspberry Pi Pico Python SDK
- Raspberry Pi RP2040 datasheet
- Hardware design with RP2040
- Raspberry Pi Pico W design files
- Raspberry Pi Pico W STEP file

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

CHAPTER

TWO

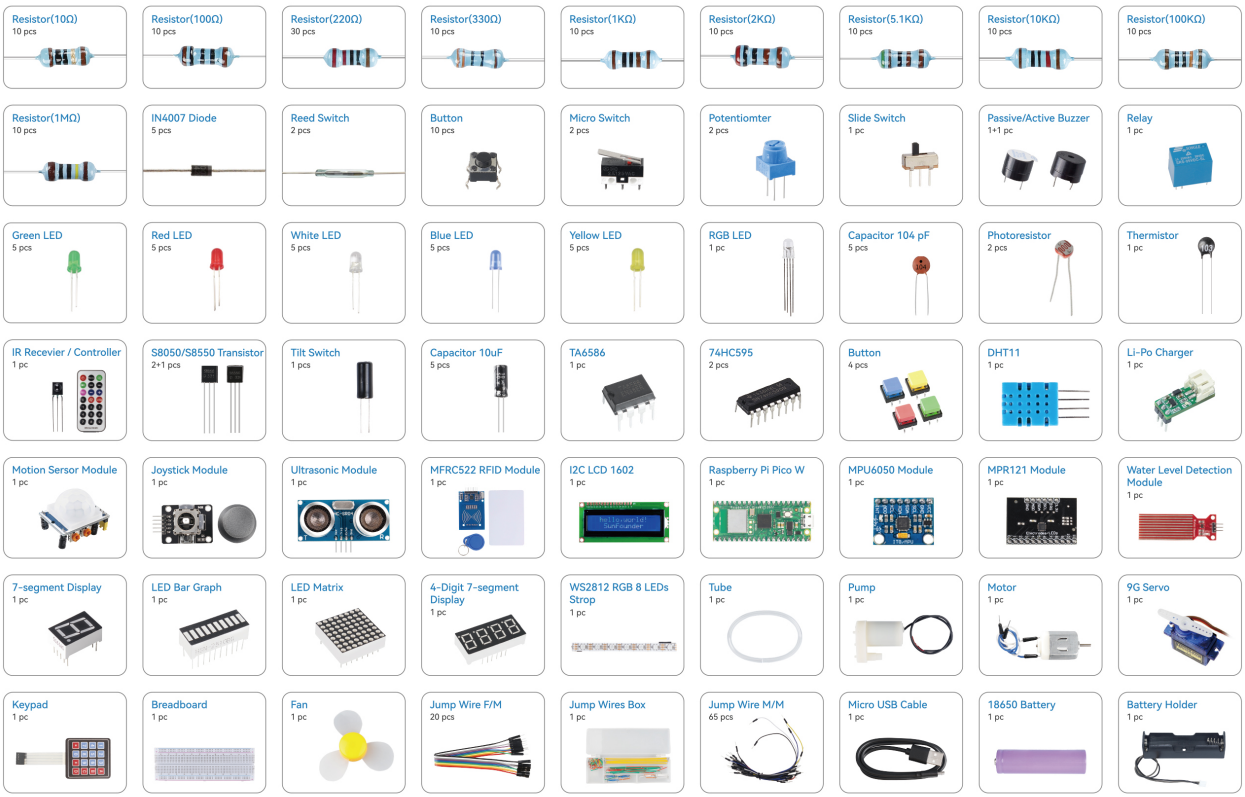
WHAT IS INCLUDED IN THIS KIT

The following is a list of this kit so you can check the contents of the kit once you receive it.

There are some components in the kit that are very small and look the same, and staff may miss or send them by mistake when packing the kit. You are welcome to send us the name of the missing or incorrect component if you find one.

Here is the email: service@sunfounder.com.

Packing List



Online Tutorials: https://kepler-kit.rtfd.io

Basic

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

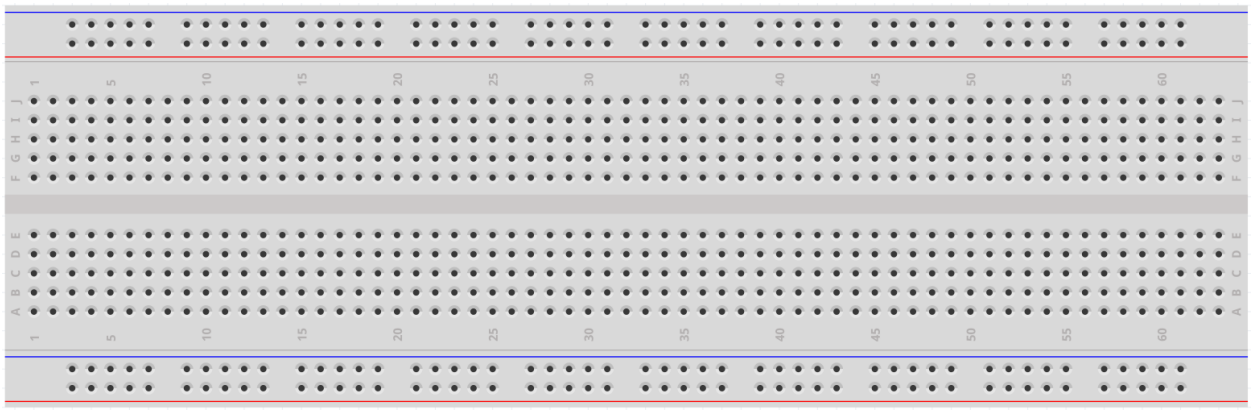
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.

- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

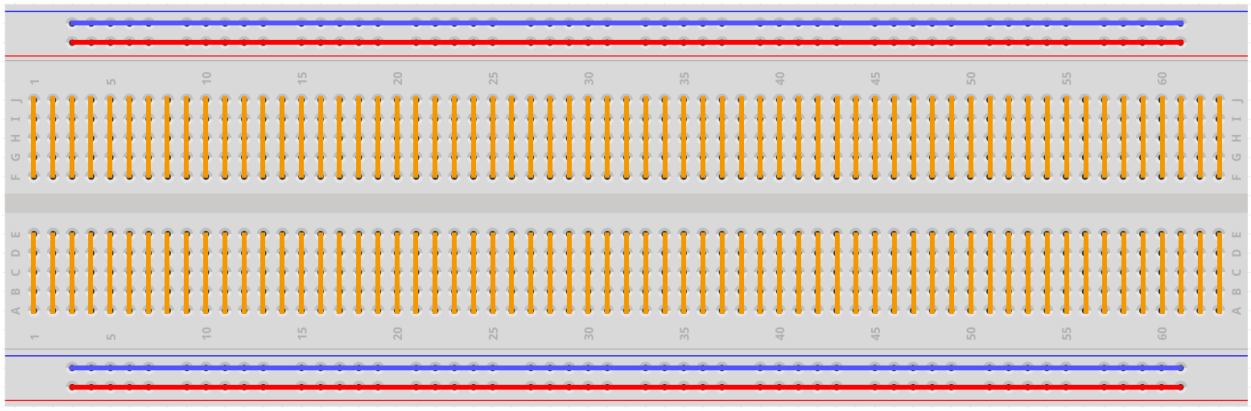
2.1 Breadboard



A breadboard is a construction base for prototyping of electronics. Originally the word referred to a literal bread board, a polished piece of wood used for slicing bread. In the 1970s the solderless breadboard (a.k.a. plugboard, a terminal array board) became available and nowadays the term “breadboard” is commonly used to refer to these.

It is used to build and test circuits quickly before finishing any circuit design. And it has many holes into which components mentioned above can be inserted like ICs and resistors as well as jumper wires. The breadboard allows you to plug in and remove components easily.

The picture shows the internal structure of a breadboard. Although these holes on the breadboard appear to be independent of each other, they are actually connected to each other through metal strips internally.



If you want to know more about breadboard, refer to: How to Use a Breadboard - Science Buddies

Example

- *Hello, Breadboard!*

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.2 Jumper Wires

Wires that connect two terminals are called jumper wires. There are various kinds of jumper wires. Here we focus on those used in breadboard. Among others, they are used to transfer electrical signals from anywhere on the breadboard to the input/output pins of a microcontroller.

Jump wires are fitted by inserting their "end connectors" into the slots provided in the breadboard, beneath whose surface there are a few sets of parallel plates that connect the slots in groups of rows or columns depending on the area. The "end connectors" are inserted into the breadboard, without soldering, in the particular slots that need to be connected in the specific prototype.

There are three types of jumper wire: Female-to-Female, Male-to-Male, and Male-to-Female. The reason we call it Male-to-Female is because it has the outstanding tip in one end as well as a sunk female end. Male-to-Male means both side are male and Female-to-Female means both ends are female.



Note:

- More than one type of them may be used in a project.
- The color of the jump wires is different but it doesn't mean their function is different accordingly; it's just designed so to better identify the connection between each circuit.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.3 Resistor



Resistor is an electronic element that can limit the branch current. A fixed resistor is a kind of resistor whose resistance cannot be changed, while that of a potentiometer or a variable resistor can be adjusted.

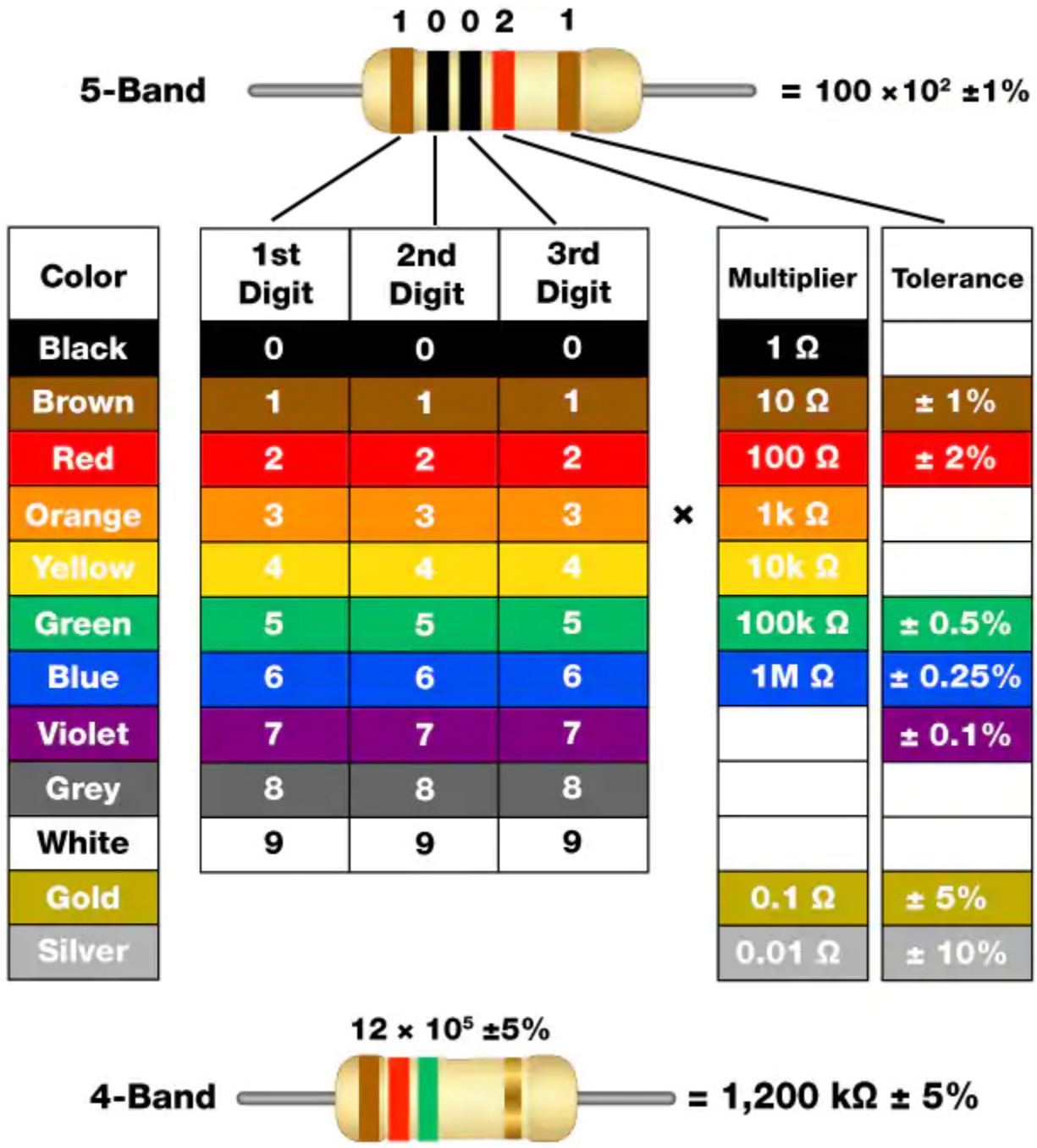
Two generally used circuit symbols for resistor. Normally, the resistance is marked on it. So if you see these symbols in a circuit, it stands for a resistor.





is the unit of resistance and the larger units include K, M, etc. Their relationship can be shown as follows: 1 M=1000 K, 1 K = 1000 . Normally, the value of resistance is marked on it.

When using a resistor, we need to know its resistance first. Here are two methods: you can observe the bands on the resistor, or use a multimeter to measure the resistance. You are recommended to use the first method as it is more convenient and faster.



As shown in the card, each color stands for a number.

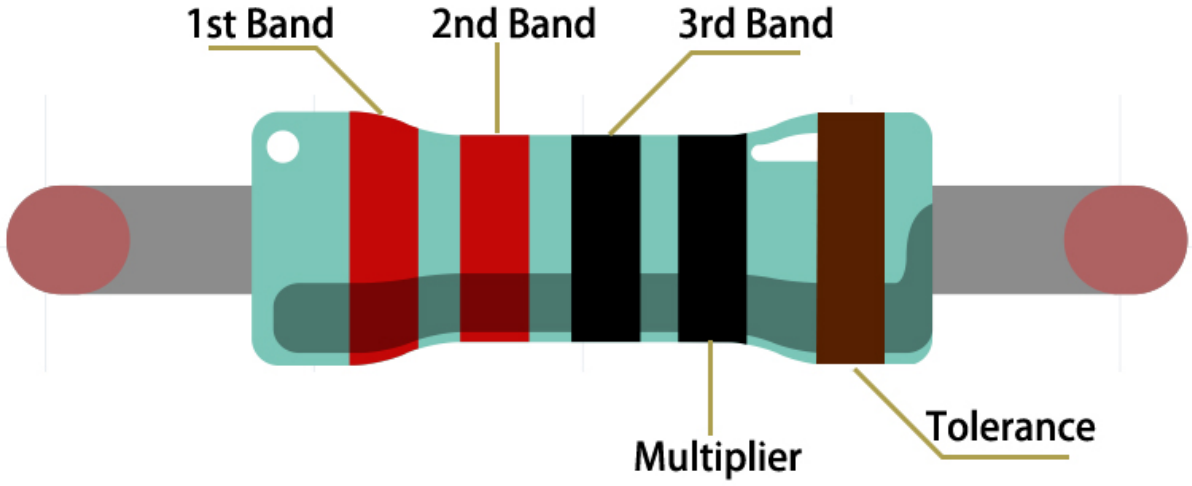
Black	Brown	Red	Orange	Yellow	Green	Blue	Violet	Grey	White	Gold	Silver
0	1	2	3	4	5	6	7	8	9	0.1	0.01

The 4- and 5-band resistors are frequently used, on which there are 4 and 5 chromatic bands.

Normally, when you get a resistor, you may find it hard to decide which end to start for reading the color. The tip is that the gap between the 4th and 5th band will be comparatively larger.

Therefore, you can observe the gap between the two chromatic bands at one end of the resistor; if it's larger than any other band gaps, then you can read from the opposite side.

Let's see how to read the resistance value of a 5-band resistor as shown below.



So for this resistor, the resistance should be read from left to right. The value should be in this format: 1st Band 2nd Band 3rd Band x 10^Multiplier () and the permissible error is ±Tolerance%. So the resistance value of this resistor is 2(red) 2(red) 0(black) x 10^0(black) = 220 , and the permissible error is ± 1% (brown).

Table 1: Common resistor color band

Resistor	Color Band
10	brown black black silver brown
100	brown black black black brown
220	red red black black brown
330	orange orange black black brown
1k	brown black black brown brown
2k	red black black brown brown
5.1k	green brown black brown brown
10k	brown black black red brown
100k	brown black black orange brown
1M	brown black black green brown

You can learn more about resistor from Wiki: Resistor - Wikipedia.

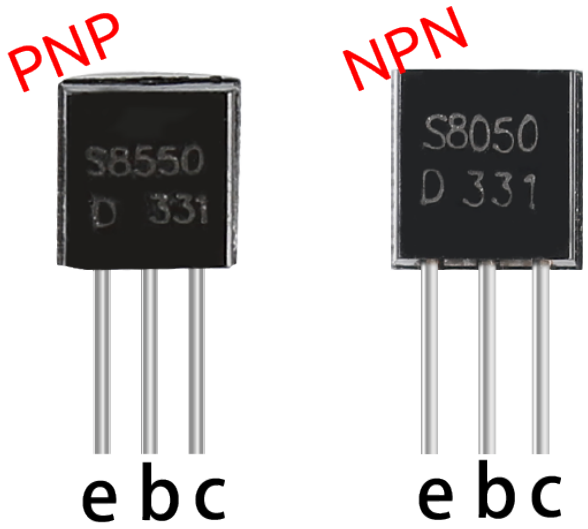
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.4 Transistor



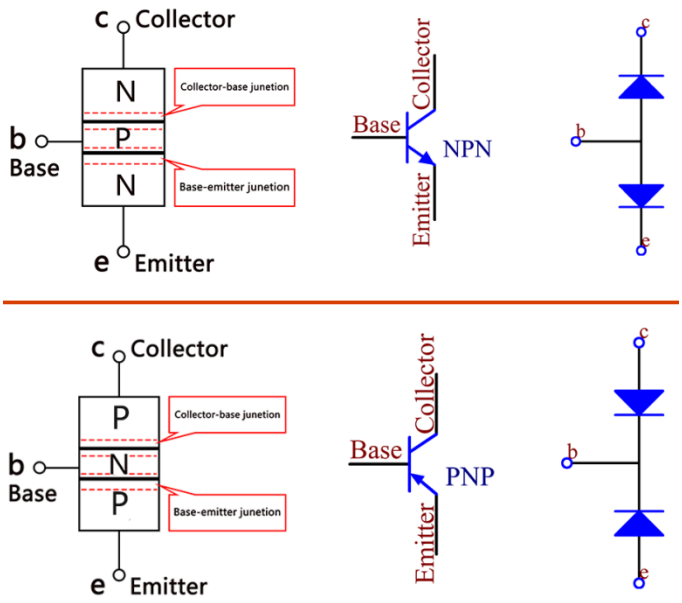
Transistor is a semiconductor device that controls current by current. It functions by amplifying weak signal to larger amplitude signal and is also used for non-contact switch.

A transistor is a three-layer structure composed of P-type and N-type semiconductors. They form the three regions internally. The thinner in the middle is the base region; the other two are both N-type or P-type ones – the smaller region with intense majority carriers is the emitter region, when the other one is the collector region. This composition enables the transistor to be an amplifier. From these three regions, three poles are generated respectively, which are base (b), emitter (e), and collector (c). They form two P-N junctions, namely, the emitter junction and collection junction. The direction of the arrow in the transistor circuit symbol indicates that of the emitter junction.

- P–N junction - Wikipedia

Based on the semiconductor type, transistors can be divided into two groups, the NPN and PNP ones. From the abbreviation, we can tell that the former is made of two N-type semiconductors and one P-type and that the latter is the opposite. See the figure below.

Note: s8550 is PNP transistor and the s8050 is the NPN one, They look very similar, and we need to check carefully to see their labels.



When a High level signal goes through an NPN transistor, it is energized. But a PNP one needs a Low level signal to manage it. Both types of transistor are frequently used for contactless switches, just like in this experiment.

- S8050 Transistor Datasheet
- S8550 Transistor Datasheet

Put the label side facing us and the pins facing down. The pins from left to right are emitter(e), base(b), and collector(c).



Note:

- The base is the gate controller device for the larger electrical supply.
- In the NPN transistor, the collector is the larger electrical supply and the emitter is the outlet for that supply, the PNP transistor is just the opposite.

Example

- *2.15 Two Kinds of Transistors* (For MicroPython User)

- *2.16 Control Another Circuit* (For MicroPython User)
- *3.1 Beep* (For MicroPython User)
- *3.2 Custom Tone* (For MicroPython User)
- *7.1 Light Theremin* (For MicroPython User)
- *7.3 Alarm Siren Lamp* (For MicroPython User)
- *7.8 RFID Music Player* (For MicroPython User)
- *7.9 Fruit Piano* (For MicroPython User)
- *7.10 Reversing Aid* (For MicroPython User)
- *3.1 - Beep* (For Arduino User)
- *3.2 - Custom Tone* (For Arduino User)
- *2.15 - Two Kinds of Transistors* (For Arduino User)
- *2.16 - Control Another Circuit* (For Arduino User)
- *2.3 Service Bell* (For Piper Make User)
- *2.11 Reversing System* (For Piper Make User)
- *2.13 Reaction Game* (For Piper Make User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.5 Capacitor



Capacitance, refers to the amount of charge storage under a given potential difference, denoted as C, and the international unit is farad (F). Generally speaking, electric charges move under force in an electric field. When there is a medium between conductors, the movement of electric charges is hindered and the electric charges accumulate on the conductors, resulting in accumulation of electric charges.

The amount of stored electric charges is called capacitance. Because capacitors are one of the most widely used electronic components in electronic equipment, they are widely used in direct current isolation, coupling, bypass, filtering, tuning loops, energy conversion, and control circuits. Capacitors are divided into electrolytic capacitors, solid capacitors, etc.

According to material characteristics, capacitors can be divided into: aluminum electrolytic capacitors, film capacitors, tantalum capacitors, ceramic capacitors, super capacitors, etc.

In this kit, ceramic capacitors and electrolytic capacitors are used.

- Ceramic Capacitor - Wikipedia
- Electrolytic Capacitor - Wikipedia

There are 103 or 104 label on the ceramic capacitors, which represent the capacitance value, 103=10x10^3pF, 104=10x10^4pF

Unit Conversion

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

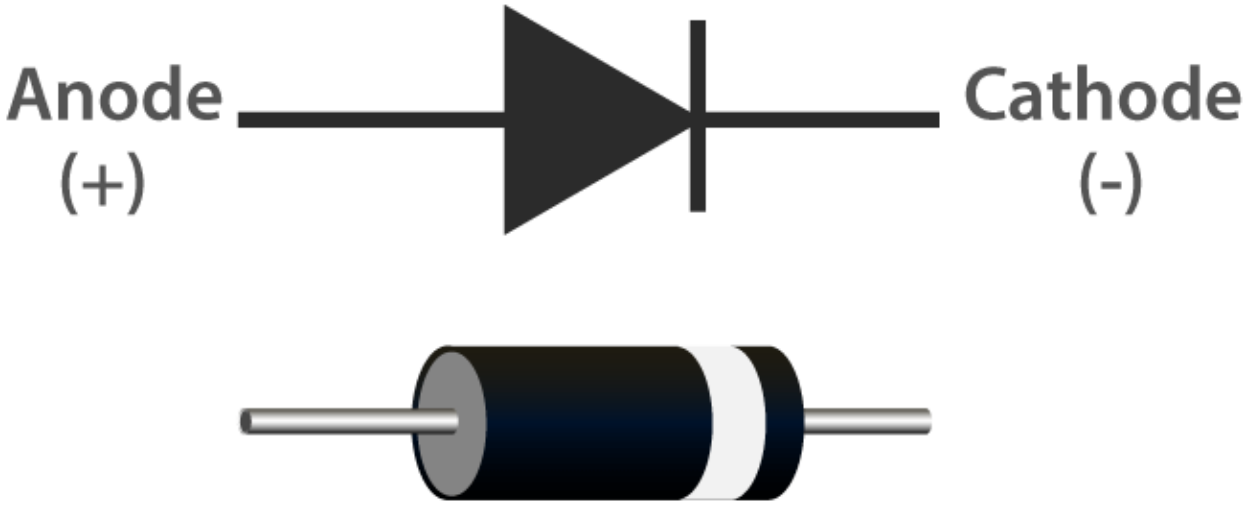
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.6 Diode



A diode is an electronic component with two electrodes. It allows current to flow in only one direction, which is often called the "Rectifying" function. Thus, a diode can be thought of as an electronic version of a check valve.

The two terminals of a diode are polarized, with the positive end called anode and the negative end called cathode. The cathode is usually made of silver or has a color band. Controlling the direction of current flow is one of the key features of diodes — the current in a diode flows from anode to cathode. The behavior of a diode is similar to the behavior of a check valve. One of the most important characteristics of a diode is the non-linear current voltage. If higher voltage is connected to the anode, then current flows from anode to cathode, and the process is known as forward bias. However, if the higher voltage is connected to the cathode, then the diode does not conduct electricity, and the process is called reverse bias.

Because of its unidirectional conductivity, the diode is used in almost all electronic circuits of some complexity. It was one of the first semiconductor devices to be created, and its applications are widespread.

However in reality diodes do not exhibit such perfect on and off directionality, but rather more complex non-linear electronic characteristics - which are determined by the specific type of diode technology.

A diode is a p-n junction formed by a p-type semiconductor and an n-type semiconductor, with a space charge layer formed on both sides at its interface and a self-built electric field, which is in electrical equilibrium when no applied voltage is present because the diffusion current due to the difference in carrier concentration between the two sides of the p-n junction and the drift current due to the self-built electric field are equal. When the forward voltage bias is generated, the mutual suppression of the external electric field and the self-built electric field increases the diffusion current of the carriers causing the forward current (that is, the reason for the conductivity). When the reverse voltage bias is generated, the external electric field and the self-built electric field are further strengthened to form a reverse saturation current I0 in a certain reverse voltage range independent of the value of the reverse bias voltage (which is the reason for non-conductivity). When the applied reverse voltage is high to a certain extent, the electric field strength in the p-n junction space charge layer reaches a critical value to produce a multiplication process of carriers, generating a large number of electron-hole pairs, resulting in a large value of the reverse breakdown current, called the diode breakdown phenomenon.

1. Forward Characteristic

When the external forward voltage is applied, in the beginning of the forward characteristic, the forward voltage is very small, not enough to overcome the blocking effect of the electric field in the p-n junction, the forward current is almost zero, this section is called the dead zone. This forward voltage that does not allow the diode to conduct is called the deadband voltage. When the forward voltage is greater than the deadband voltage, p-n junction electric field is overcome, the diode forward conduction, the current increases with the voltage and rises rapidly. In the normal use

of the current range, the terminal voltage of the diode during conduction remains almost constant, this voltage is called the forward voltage of the diode.

2. Reverse Characteristic

When the applied reverse voltage, and does not exceed a certain range, the current through the diode is a few carriers drifting movement formed by the reverse current. As the reverse current is very small, the diode is in the cutoff state. This reverse current is also known as reverse saturation current or leakage current, and is greatly influenced by temperature.

3. Breakdown

When the applied reverse voltage exceeds a certain value, the reverse current will suddenly increase, a phenomenon known as electrical breakdown. The critical voltage that causes electrical breakdown is called the reverse breakdown voltage, the diode will loses its unidirectional conductivity at the time of electrical breakdown. Therefore, the use of the diode should be avoided when the applied reverse voltage is too high.

Early diodes consisted of "Cat's Whisker" Crystals and Vacuum tubes (also called "Thermionic Valves"). Most of today's most common diodes use semiconductor materials such as silicon or germanium.

- P–N junction - Wikipedia
- Diode - Wikipedia

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.7 Li-po Charger Module



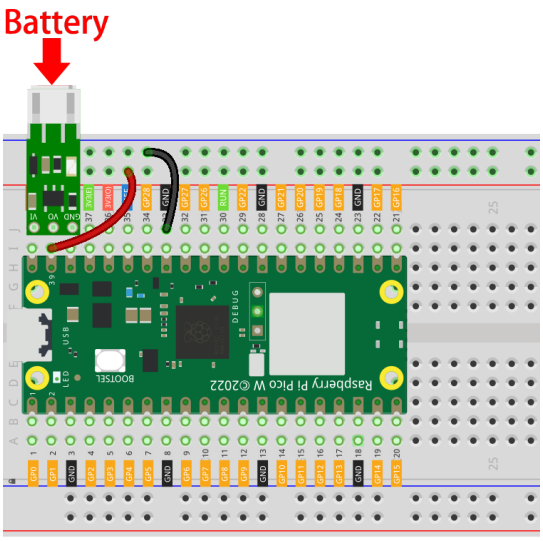
This is a Li-po charger module designed for Raspberry Pi Pico/Pico H/Pico W. Just plug it and the Pico into the breadboard as shown below, and then connect the battery at the other end and you are ready to use.

When you plug in the Pico W with a USB cable connected to a computer or socket, the indicator light on the Li-po Charger module lights up, representing the battery will be charged at the same time. When you unplug the USB cable, the Pico W will be powered by the battery, so you can keep your project running.

Note: For some computers with poor performance, sometimes if you plug in your Pico W to your computer with this charging module attached, it may cause the computer not to recognize your Pico W.

The reason is that after plugging in, while charging the battery, the USB port voltage is pulled down, resulting in the Pico W power supply is insufficient to be recognized by the computer.

In this case, you need to pull out the Li-Po charging module and then plug in the Pico W again.

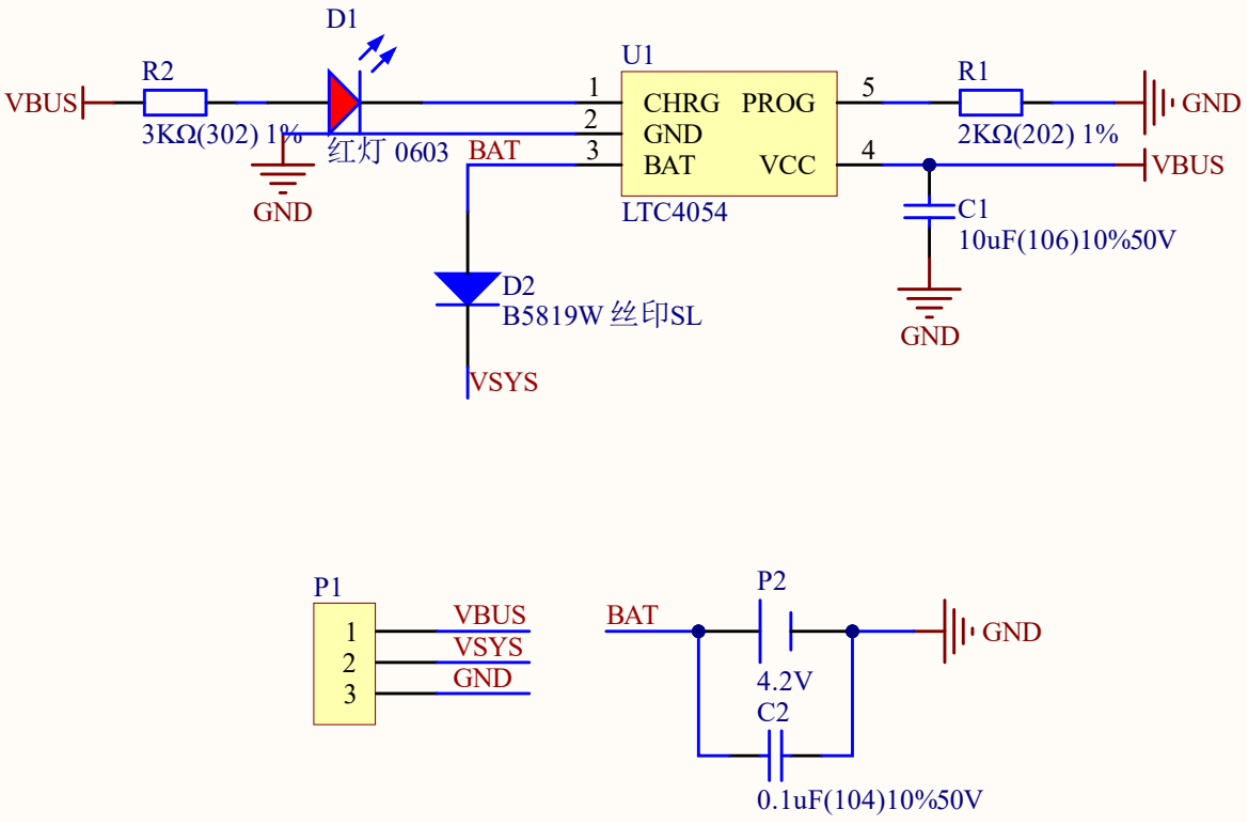


Features

- Input voltage: 5V

- Output voltage: 3.3V
- Size: 20mmx7mm
- Interface model: PH2.0
- There is a matching 1A battery holder as well as an 800mAh 18650 used together.

Schematic



Chip

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

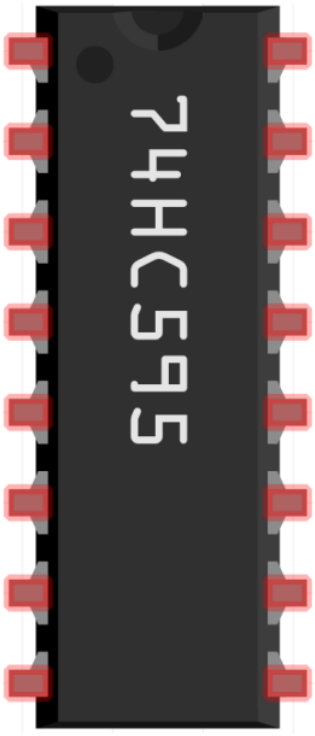
Ready to explore and create with us? Click [] and join today!

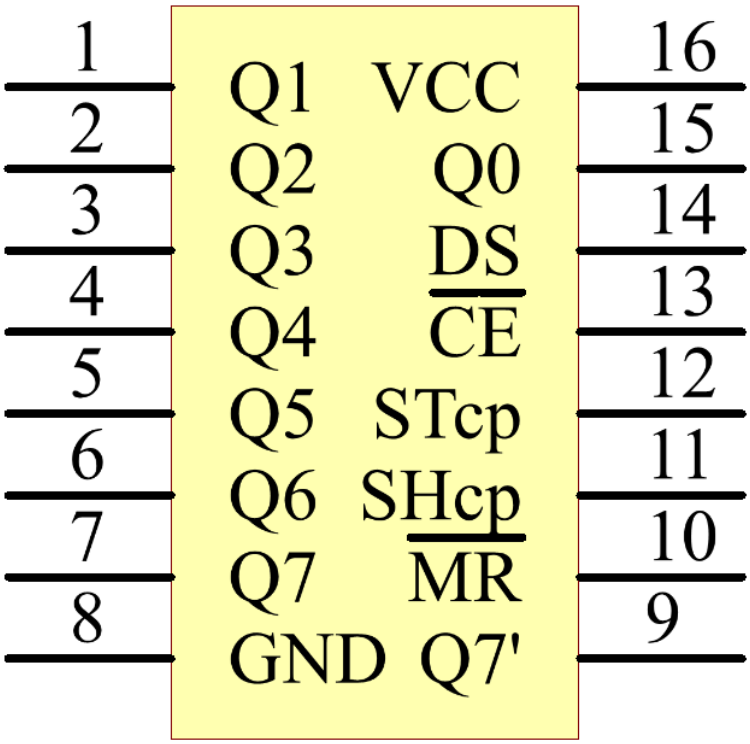
2.8 74HC595



The 74HC595 consists of an 8bit shift register and a storage register with threestate parallel outputs. It converts serial input into parallel output so you can save IO ports of an MCU.

- When MR (pin10) is high level and OE (pin13) is low level, data is input in the rising edge of SHcp and goes to the memory register through the rising edge of SHcp.
- If the two clocks are connected together, the shift register is always one pulse earlier than the memory register.
- There is a serial shift input pin (Ds), a serial output pin (Q) and an asynchronous reset button (low level) in the memory register.
- The memory register outputs a Bus with a parallel 8-bit and in three states.
- When OE is enabled (low level), the data in memory register is output to the bus(Q0 ~ Q7).
- 74HC595 Datasheet





Pins of 74HC595 and their functions:

- **Q0-Q7**: 8-bit parallel data output pins, able to control 8 LEDs or 8 pins of 7-segment display directly.
- **Q7'**: Series output pin, connected to DS of another 74HC595 to connect multiple 74HC595s in series
- **MR**: Reset pin, active at low level;
- **SHcp**: Time sequence input of shift register. On the rising edge, the data in shift register moves successively one bit, i.e. data in Q1 moves to Q2, and so forth. While on the falling edge, the data in shift register remain unchanged.

- **STcp**: Time sequence input of storage register. On the rising edge, data in the shift register moves into memory register.
- **CE**: Output enable pin, active at low level.
- **DS**: Serial data input pin
- **VCC**: Positive supply voltage.
- **GND**: Ground.

Example

- *5.1 Microchip - 74HC595* (For MicroPython User)
- *5.2 Number Display* (For MicroPython User)
- *5.3 Time Counter* (For MicroPython User)
- *5.4 8x8 Pixel Graphics* (For MicroPython User)
- *7.4 Passenger Counter* (For MicroPython User)
- *7.5 GAME - 10 Second* (For MicroPython User)
- *7.6 Traffic Light* (For MicroPython User)
- *7.12 Digital Bubble Level* (For MicroPython User)
- *5.1 Microchip - 74HC595* (For Arduino User)
- *5.2 - Number Display* (For Arduino User)
- *5.3 - Time Counter* (For Arduino User)
- *5.4 - 8x8 Pixel Graphics* (For Arduino User)

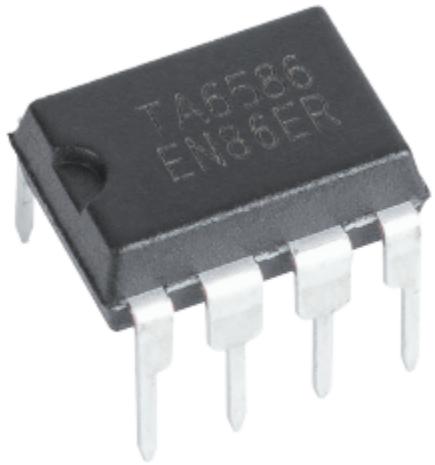
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.9 TA6586 - Motor Driver Chip



TA6586 is a monolithic IC designed for driving bi-directional DC motor. It has two pins of logic inputs for controlling the direction, forward and backward. The circuit feature good anti-interference performance, small standby current and low output saturation pressure drop. It has a built-in clamp diode to reverse the impact of the release of inductive load current, making it in the drive relays, DC motors, stepper motors or control the use of switching power safe and reliable. TA6586 is suitable for toy vehicles, remote-controlled aircraft motor drive, automatic valve motor, electromagnetic lock drive, precision instruments and other circuits.

Features

- Low stand-by current: 2uA
- Wide supply voltage range
- Built-in Brake Function
- Thermal Shutdown protection
- Over Current Limit and Short Circuit Protect Function
- DIP8 Pb-Free package.

Pin Function

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

Input Truth Table

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

Example

- *3.5 Small Fan* (For MicroPython User)
- *3.5 - Small Fan* (For Arduino User)
- *3.6 Pumping* (For MicroPython User)
- *3.6 - Pumping* (For Arduino User)
- *2.12 Smart Fan* (For Piper Make User)

Display

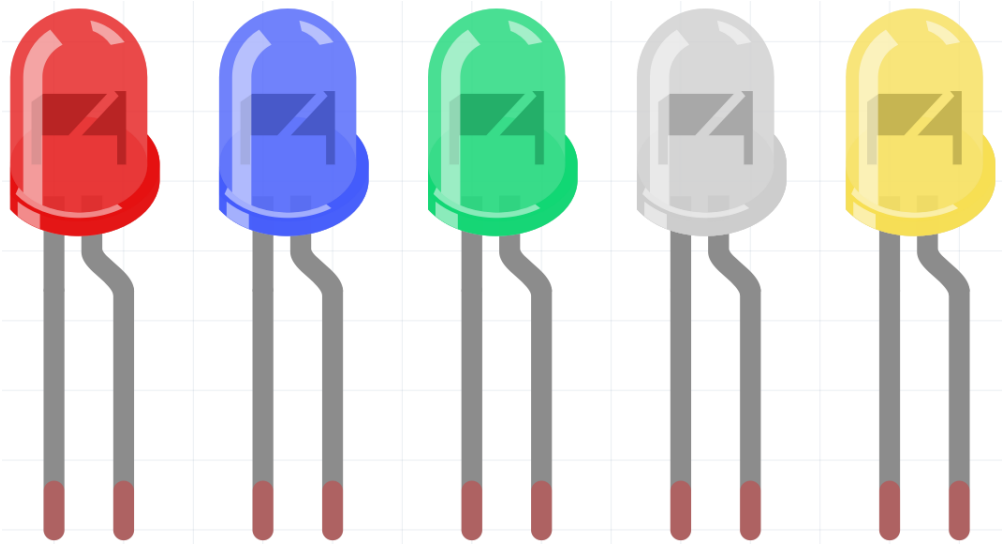
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

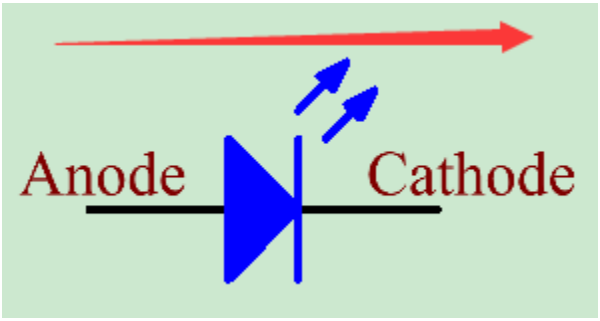
Ready to explore and create with us? Click [] and join today!

2.10 LED



Semiconductor light-emitting diode is a type of component which can turn electric energy into light energy via PN junctions. By wavelength, it can be categorized into laser diode, infrared light-emitting diode and visible light-emitting diode which is usually known as light-emitting diode (LED).

Diode has unidirectional conductivity, so the current flow will be as the arrow indicates in figure circuit symbol. You can only provide the anode with a positive power and the cathode with a negative. Thus the LED will light up.



An LED has two pins. The longer one is the anode, and shorter one, the cathode. Pay attention not to connect them inversely. There is fixed forward voltage drop in the LED, so it cannot be connected with the circuit directly because the supply voltage can outweigh this drop and cause the LED to be burnt. The forward voltage of the red, yellow, and green LED is 1.8 V and that of the white one is 2.6 V. Most LEDs can withstand a maximum current of 20 mA, so we need to connect a current limiting resistor in series.

The formula of the resistance value is as follows:

$$R = (Vsupply - VD)/I$$

R stands for the resistance value of the current limiting resistor, **Vsupply** for voltage supply, **VD** for voltage drop and **I** for the working current of the LED.

Here is the detailed introduction for the LED: LED - Wikipedia.

Example

- *2.1 Hello, LED!* (For MicroPython User)
- *2.3 Fading LED* (For MicroPython User)

- *7.3 Alarm Siren Lamp* (For MicroPython User)
- *7.6 Traffic Light* (For MicroPython User)
- *7.10 Reversing Aid* (For MicroPython User)
- *2.1 - Hello, LED!* (For Arduino User)
- *2.3 - Fading LED* (For Arduino User)
- *2.1 Blink LED* (For Piper Make User)
- *2.2 Button* (For Piper Make User)
- *2.3 Service Bell* (For Piper Make User)
- *2.11 Reversing System* (For Piper Make User)
- *2.13 Reaction Game* (For Piper Make User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

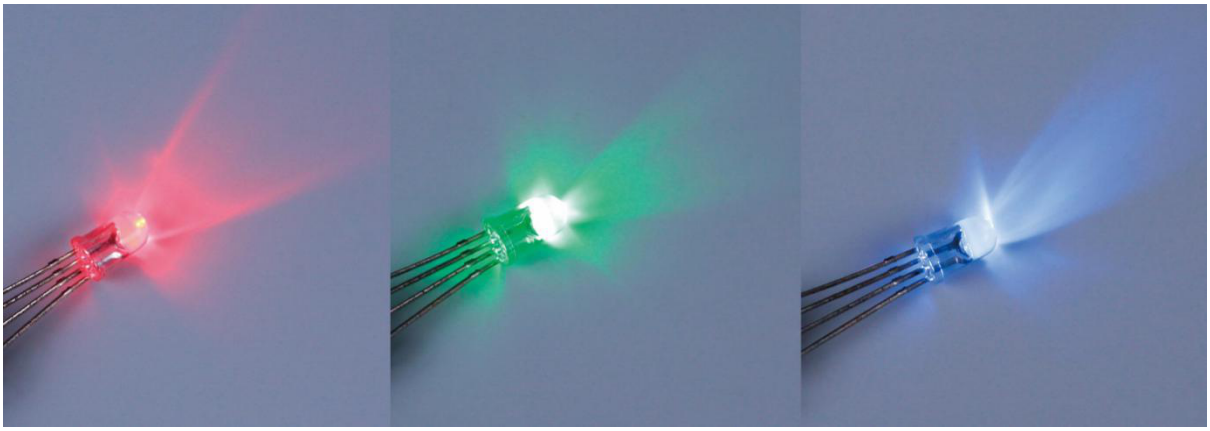
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.11 RGB LED

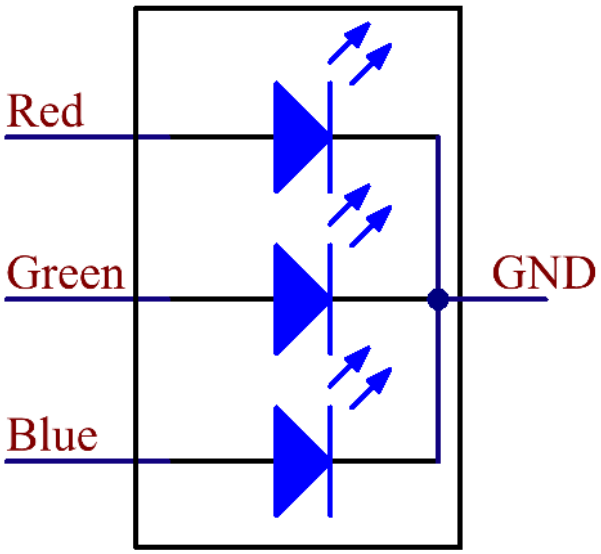


RGB LEDs emit light in various colors. An RGB LED packages three LEDs of red, green, and blue into a transparent or semitransparent plastic shell. It can display various colors by changing the input voltage of the three pins and superimpose them, which, according to statistics, can create 16,777,216 different colors.

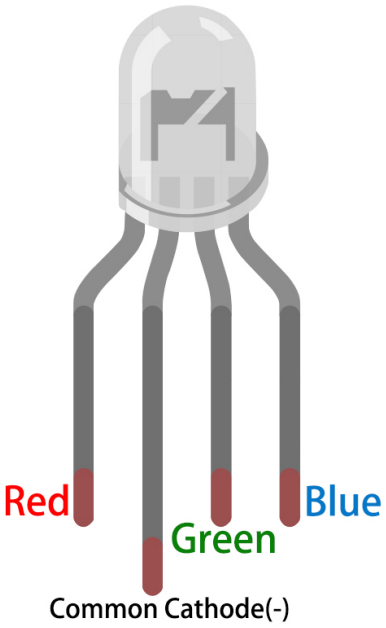


RGB LEDs can be categorized into common anode and common cathode ones. In this kit, the latter is used. The **common cathode**, or CC, means to connect the cathodes of the three LEDs. After you connect it with GND and plug in the three pins, the LED will flash the corresponding color.

Its circuit symbol is shown as figure.



An RGB LED has 4 pins: the longest pin is the common cathode pin, which is usually connected to GND, the left pin next to the longest pin is Red, and the 2 pins on the right are Green and Blue.



Features

- Color: Tri-Color (Red/Green/Blue)
- Common Cathode
- 5mm Clear Round Lens
- Forward Voltage: Red: DC 2.0 - 2.2V; Blue&Green: DC 3.0 - 3.2V (IF=20mA)
- 0.06 Watts DIP RGB LED
- Luminance Brighter Up To +20%
- Viewing Angle: 30°

Example

- *2.4 Colorful Light* (For MicroPython User)
- *7.9 Fruit Piano* (For MicroPython User)
- *2.4 - Colorful Light* (For Arduino User)
- *2.4 Rainbow Light* (For Piper Make User)

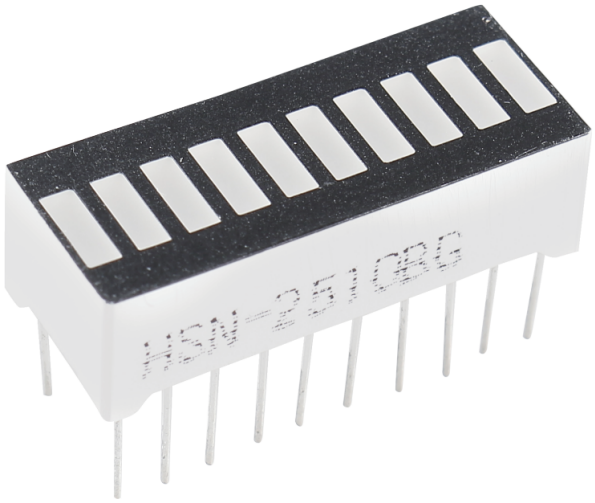
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

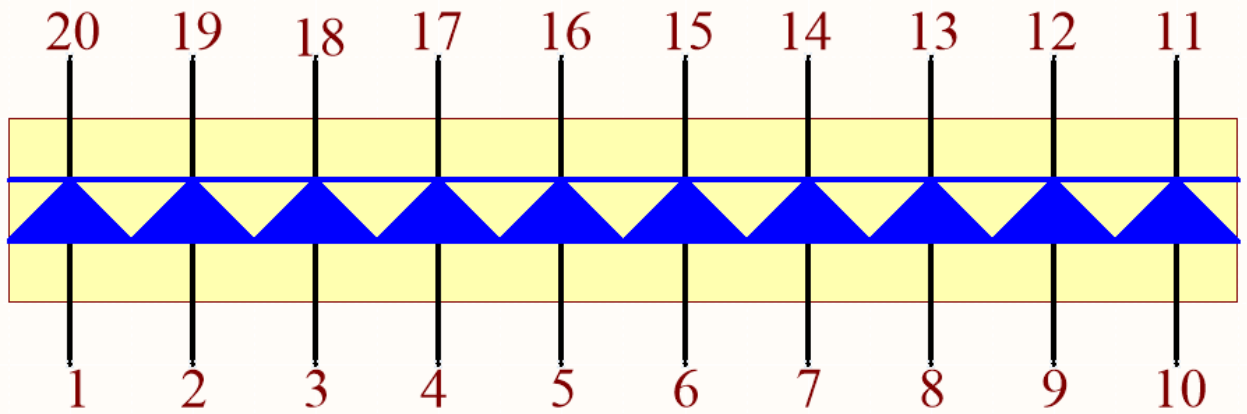
2.12 LED Bar Graph



LED Bar Graph is an LED array, which is used to connect with electronic circuit or microcontroller. It's easy to connect LED bar graph with the circuit like as connecting 10 individual LEDs with 10 output pins. Generally we can use the LED bar graph as a Battery level Indicator, Audio equipments, and Industrial Control panels. There are many other applications of LED bar graphs.

The following is the internal schematic diagram of LED Bar Graph. Generally speaking, the side with the label is the anode and the other side is the cathode.





Example

- *2.2 Display the Level* (For MicroPython User)
- *2.2 - Display the Level* (For Arduino User)
- *2.8 Light Intensity Display* (For Piper Make User)

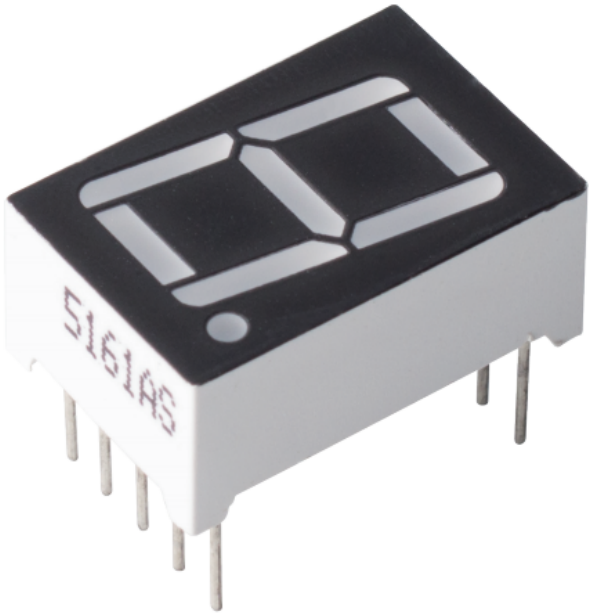
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

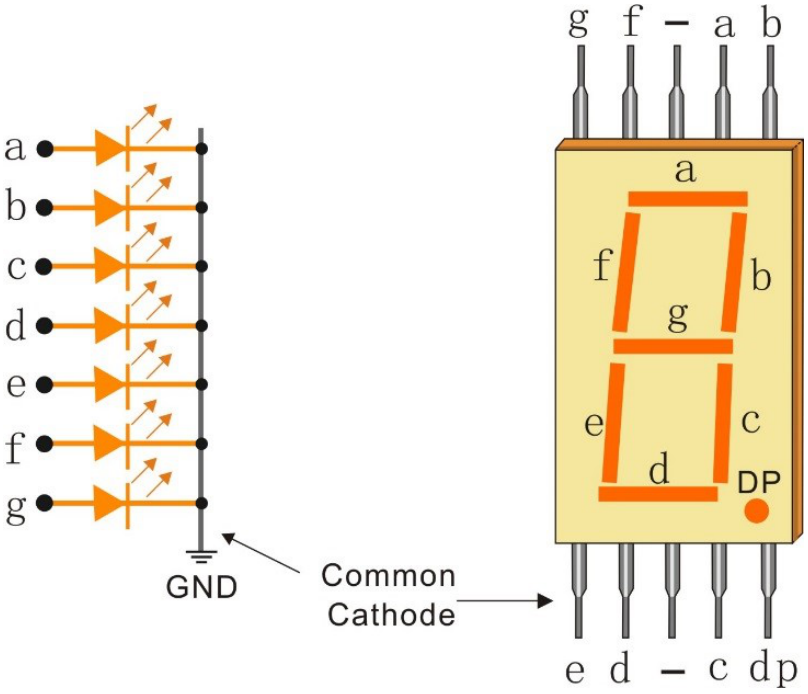
2.13 7-segment Display



A 7-segment display is an 8-shaped component which packages 7 LEDs. Each LED is called a segment - when energized, one segment forms part of a numeral to be displayed.

There are two types of pin connection: Common Cathode (CC) and Common Anode (CA). As the name suggests, a CC display has all the cathodes of the 7 LEDs connected when a CA display has all the anodes of the 7 segments connected.

In this kit, we use the Common Cathode 7-segment display, here is the electronic symbol.



Each of the LEDs in the display is given a positional segment with one of its connection pins led out from the rectangular plastic package. These LED pins are labeled from “a” through to “g” representing each individual LED. The other LED

pins are connected together forming a common pin. So by forward biasing the appropriate pins of the LED segments in a particular order, some segments will brighten and others stay dim, thus showing the corresponding character on the display.

- Seven-segment Display - Wikipedia

Display Codes

To help you get to know how 7-segment displays(Common Cathode) display Numbers, we have drawn the following table. Numbers are the number 0-F displayed on the 7-segment display; (DP) GFEDCBA refers to the corresponding LED set to 0 or 1, For example, 00111111 means that DP and G are set to 0, while others are set to 1. Therefore, the number 0 is displayed on the 7-segment display, while HEX Code corresponds to hexadecimal number.

Table 2: Glyph Code

Numbers	Binary Code	Hex Code
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

Example

- *5.2 Number Display* (For MicroPython User)
- *5.2 - Number Display* (For Arduino User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

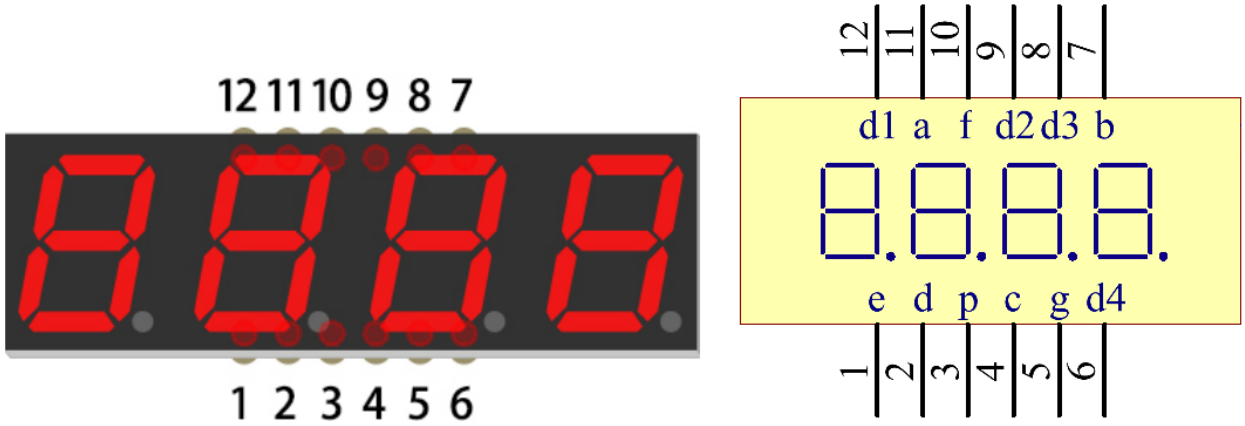
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

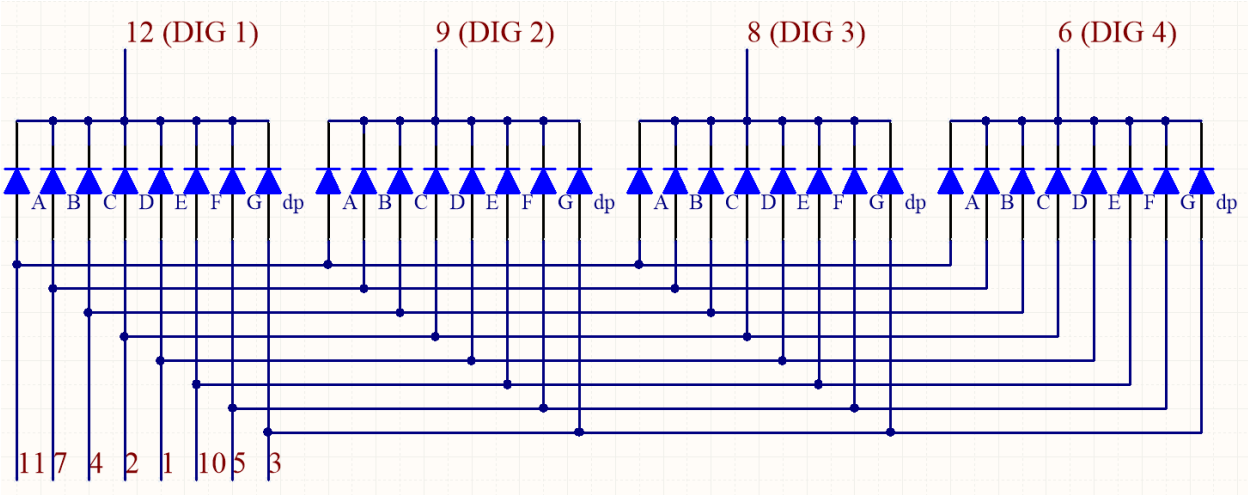
2.14 4-Digit 7-Segment Display

4-Digit 7-segment display consists of four 7- segment displays working together.



The 4-digtal 7-segment display works independently. It uses the principle of human visual persistence to quickly display the characters of each 7-segment in a loop to form continuous strings.

For example, when "1234" is displayed on the display, "1" is displayed on the first 7-segment, and "234" is not displayed. After a period of time, the second 7-segment shows "2", the 1st 3th 4th of 7-segment does not show, and so on, the four digital display show in turn. This process is very short (typically 5ms), and because of the optical afterglow effect and the principle of visual residue, we can see four characters at the same time.



Display Codes

To help you get to know how 7-segment displays(Common Cathode) display Numbers, we have drawn the following table. Numbers are the number 0-F displayed on the 7-segment display; (DP) GFEDCBA refers to the corresponding LED set to 0 or 1, For example, 00111111 means that DP and G are set to 0, while others are set to 1. Therefore, the number 0 is displayed on the 7-segment display, while HEX Code corresponds to hexadecimal number.

Table 3: Glyph Code

Numbers	Binary Code	Hex Code
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

Example

- *5.3 Time Counter* (For MicroPython User)
- *7.4 Passenger Counter* (For MicroPython User)
- *7.5 GAME - 10 Second* (For MicroPython User)
- *7.6 Traffic Light* (For MicroPython User)
- *5.3 - Time Counter* (For Arduino User)

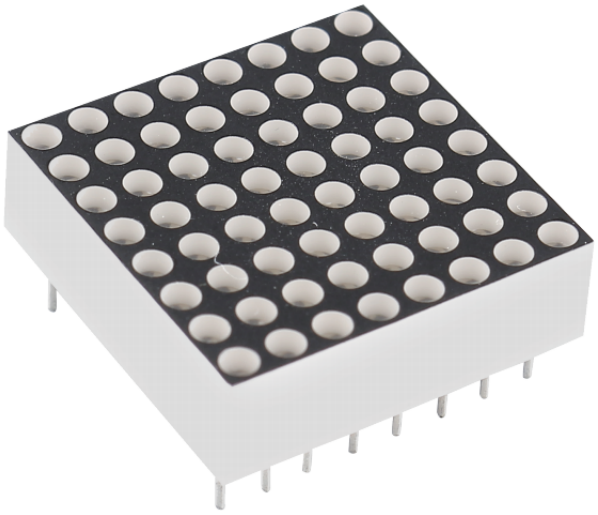
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

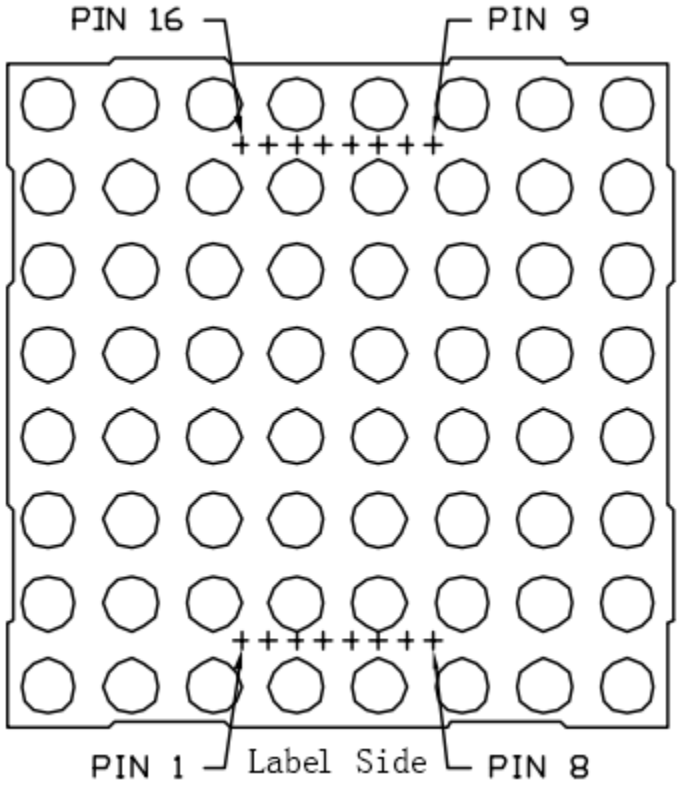
2.15 LED Dot Matrix



Generally, LED dot matrix can be categorized into two types: common cathode (CC) and common anode (CA). They look much alike, but internally the difference lies. You can tell by test. A CA one is used in this kit. You can see 788BS labeled at the side.

See the figure below. The pins are arranged at the two ends at the back. Take the label side for reference: pins on this end are pin 1-8, and oh the other are pin 9-16.

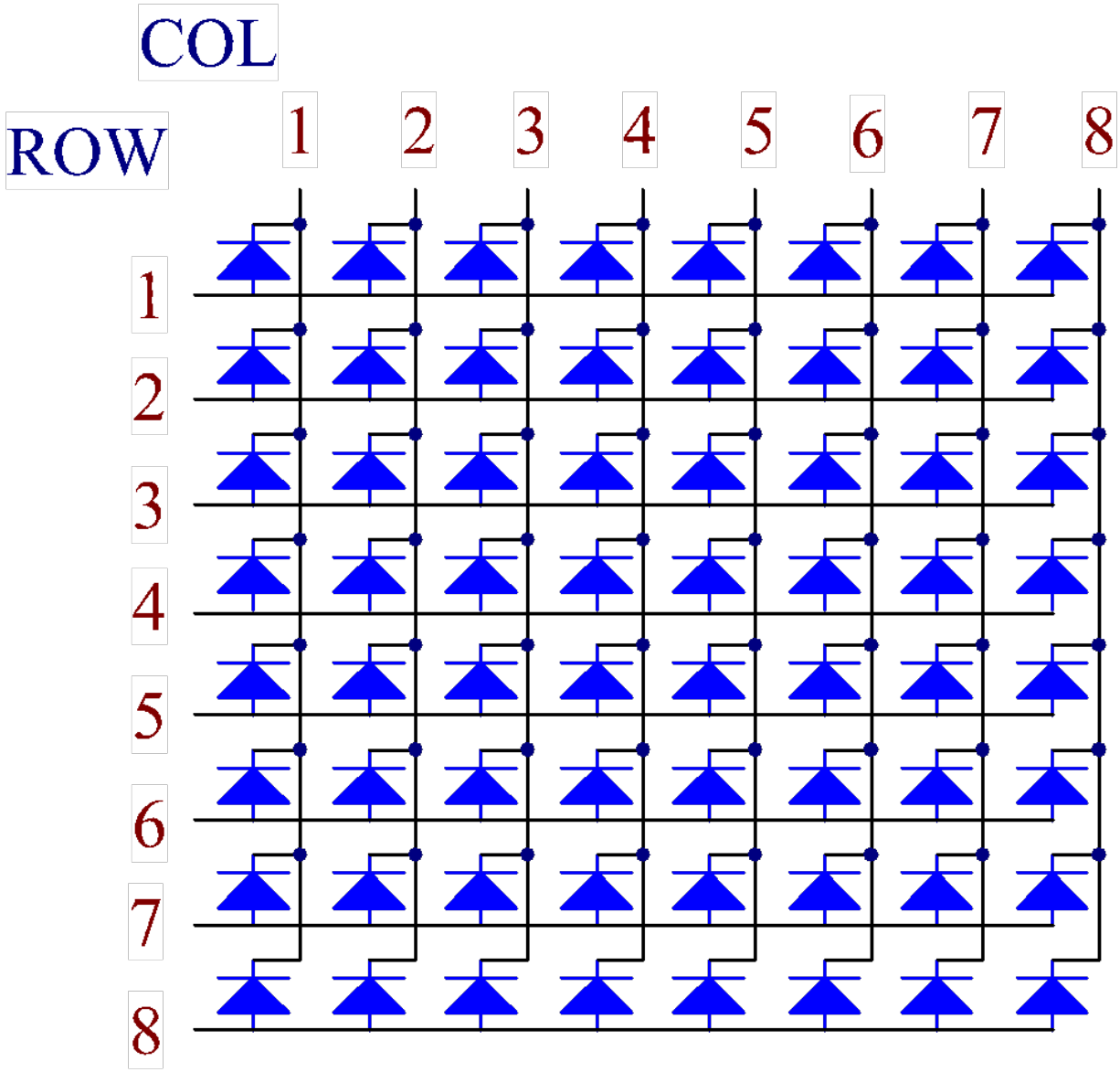
The external view:



Below the figures show their internal structure. You can see in a CA LED dot matrix, ROW represents the anode of the

LED, and COL is cathode; it's contrary for a CC one. One thing in common: for both types, pin 13, 3, 4, 10, 6, 11, 15, and 16 are all COL, when pin 9, 14, 8, 12, 1, 7, 2, and 5 are all ROW. If you want to turn on the first LED at the top left corner, for a CA LED dot matrix, just set pin 9 as High and pin 13 as Low, and for a CC one, set pin 13 as High and pin 9 as Low. If you want to light up the whole first column, for CA, set pin 13 as Low and ROW 9, 14, 8, 12, 1, 7, 2, and 5 as High, when for CC, set pin 13 as High and ROW 9, 14, 8, 12, 1, 7, 2, and 5 as Low. Consider the following figures for better understanding.

The internal view:



Pin numbering corresponding to the above rows and columns:

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

In addition, two 74HC595 chips are used here. One is to control the rows of the LED dot matrix while the other, the columns.

Example

- *5.4 8x8 Pixel Graphics* (For MicroPython User)
- *7.12 Digital Bubble Level* (For MicroPython User)
- *5.4 - 8x8 Pixel Graphics* (For Arduino User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

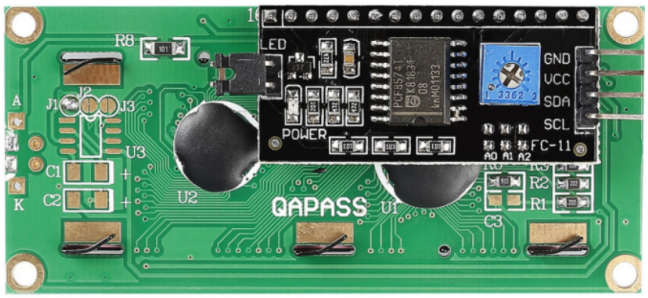
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.16 I2C LCD1602





- **GND**: Ground
- **VCC**: Voltage supply, 5V.
- **SDA**: Serial data line. Connect to VCC through a pullup resistor.
- **SCL**: Serial clock line. Connect to VCC through a pullup resistor.

As we all know, though LCD and some other displays greatly enrich the man-machine interaction, they share a common weakness. When they are connected to a controller, multiple IOs will be occupied of the controller which has no so many outer ports. Also it restricts other functions of the controller.

Therefore, LCD1602 with an I2C module is developed to solve the problem. The I2C module has a built-in PCF8574 I2C chip that converts I2C serial data to parallel data for the LCD display.

- PCF8574 Datasheet

I2C Address

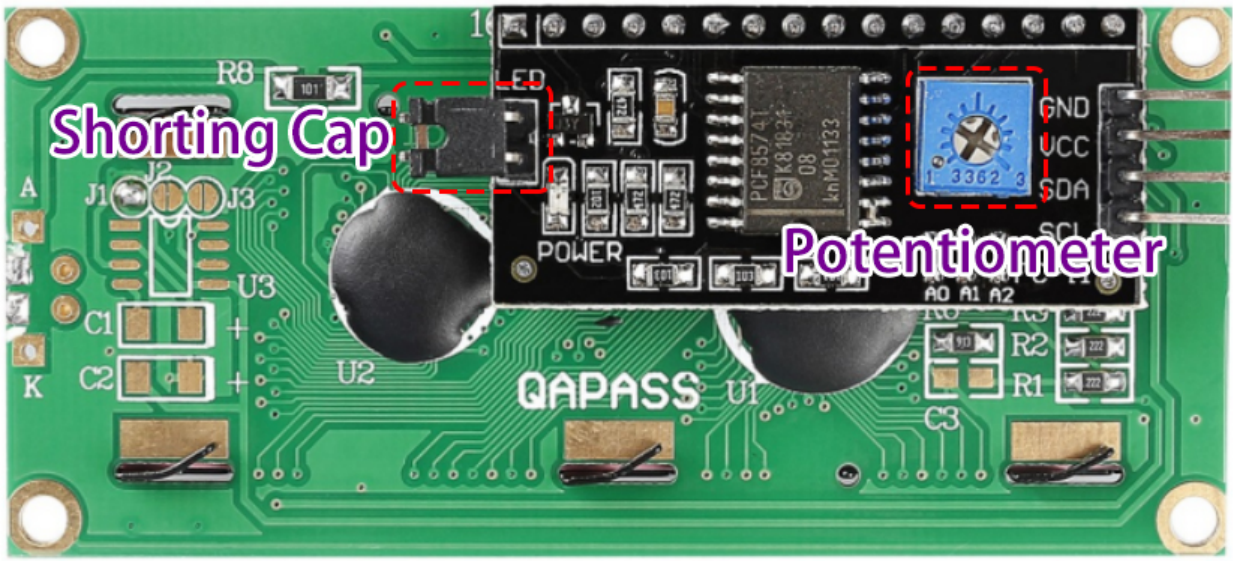
The default address is basically 0x27, in a few cases it may be 0x3F.

Taking the default address of 0x27 as an example, the device address can be modified by shorting the A0/A1/A2 pads; in the default state, A0/A1/A2 is 1, and if the pad is shorted, A0/A1/A2 is 0.



Backlight/Contrast

Backlight can be enabled by jumper cap, unplugg the jumper cap to disable the backlight. The blue potentiometer on the back is used to adjust the contrast (the ratio of brightness between the brightest white and the darkest black).



- **Shorting Cap**: Backlight can be enabled by this cap, unplugg this cap to disable the backlight.
- **Potentiometer**: It is used to adjust the contrast (the clarity of the displayed text), which is increased in the clockwise direction and decreased in the counterclockwise direction.

Example

- *3.4 Liquid Crystal Display* (For MicroPython User)
- *7.2 Room Temperature Meter* (For MicroPython User)
- *7.7 Guess Number* (For MicroPython User)
- *3.4 - Liquid Crystal Display* (For Arduino User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.17 WS2812 RGB 8 LEDs Strip



The WS2812 RGB 8 LEDs Strip is composed of 8 RGB LEDs. Only one pin is required to control all the LEDs. Each RGB LED has a WS2812 chip, which can be controlled independently. It can realize 256-level brightness display and complete true color display of 16,777,216 colors. At the same time, the pixel contains an intelligent digital interface data latch signal shaping amplifier drive circuit, and a signal shaping circuit is built in to effectively ensure the color height of the pixel point light Consistent.

It is flexible, can be docked, bent, and cut at will, and the back is equipped with adhesive tape, which can be fixed on the uneven surface at will, and can be installed in a narrow space.

Features

- Work Voltage: DC5V
- IC: One IC drives one RGB LED
- Consumption: 0.3w each LED
- Working Temperature: -15-50
- Color: Full color RGB
- RGB Type: 5050RGB(Built-in IC WS2812B)
- Light Strip Thickness: 2mm
- Each LED can be controlled individually

WS2812B Introdction

- WS2812B Datasheet

WS2812B is a intelligent control LED light source that the control circuit and RGB chip are integrated in a package of 5050 components. It internal include intelligent digital port data latch and signal reshaping ampli fication drive circuit. Also include a precision internal oscillator and a 12V voltage programmable constant curr e-nt control part, effectively ensuring the pixel point light color height consistent.

The data transfer protocol use single NZR communication mode. After the pixel power-on reset, the DIN port receive data from controller, the first pixel collect initial 24bit data then sent to the internal data latch, the other data which reshaping by the internal signal reshaping amplification circuit sent to the next cascade pixel through the DO port. After transmission for each pixel, the signal to reduce 24bit. pixel adopt auto resha -ping transmit technology, making the pixel cascade number is not limited the signal transmission, only depend on the speed of signal transmission.

LED with low driving voltage, environmental protection and energy saving, high brightness, scattering angl e is large, good consistency, low power, long life and other advantages. The control chip integrated in LED above becoming more simple circuit, small volume, convenient installation.

Example

- *3.3 RGB LED Strip* (For MicroPython User)
- *7.8 RFID Music Player* (For MicroPython User)
- *3.3 WS2812 RGB Strip* (For Arduino User)
- *2.10 Flowing LEDs* (For Piper Make User)

Sound

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.18 Buzzer

As a type of electronic buzzer with an integrated structure, buzzers, which are supplied by DC power, are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic devices, telephones, timers and other electronic products or voice devices.

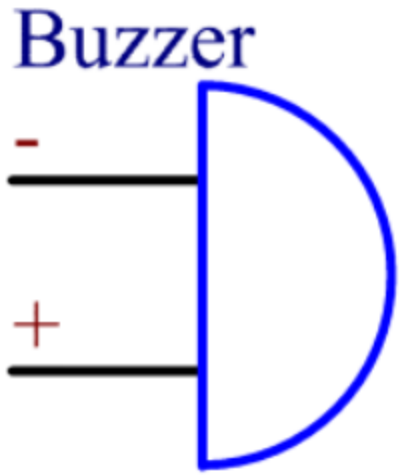
Buzzers can be categorized as active and passive ones (see the following picture). Turn the buzzer so that its pins are facing up, and the buzzer with a green circuit board is a passive buzzer, while the one enclosed with a black tape is an active one.



The difference between an active buzzer and a passive buzzer:

An active buzzer has a built-in oscillating source, so it will make sounds when electrified. But a passive buzzer does not have such source, so it will not beep if DC signals are used; instead, you need to use square waves whose frequency is between 2K and 5K to drive it. The active buzzer is often more expensive than the passive one because of multiple built-in oscillating circuits.

The following is the electrical symbol of a buzzer. It has two pins with positive and negative poles. With a + in the surface represents the anode and the other is the cathode.



You can check the pins of the buzzer, the longer one is the anode and the shorter one is the cathode. Please don't mix them up when connecting, otherwise the buzzer will not make sound.

Buzzer - Wikipedia

Example

- *3.1 Beep* (For MicroPython User)
- *3.2 Custom Tone* (For MicroPython User)
- *7.1 Light Theremin* (For MicroPython User)
- *7.3 Alarm Siren Lamp* (For MicroPython User)
- *7.8 RFID Music Player* (For MicroPython User)
- *7.9 Fruit Piano* (For MicroPython User)
- *7.10 Reversing Aid* (For MicroPython User)
- *3.1 - Beep* (For Arduino User)
- *3.2 - Custom Tone* (For Arduino User)
- *2.3 Service Bell* (For Piper Make User)
- *2.11 Reversing System* (For Piper Make User)
- *2.13 Reaction Game* (For Piper Make User)

Actuators

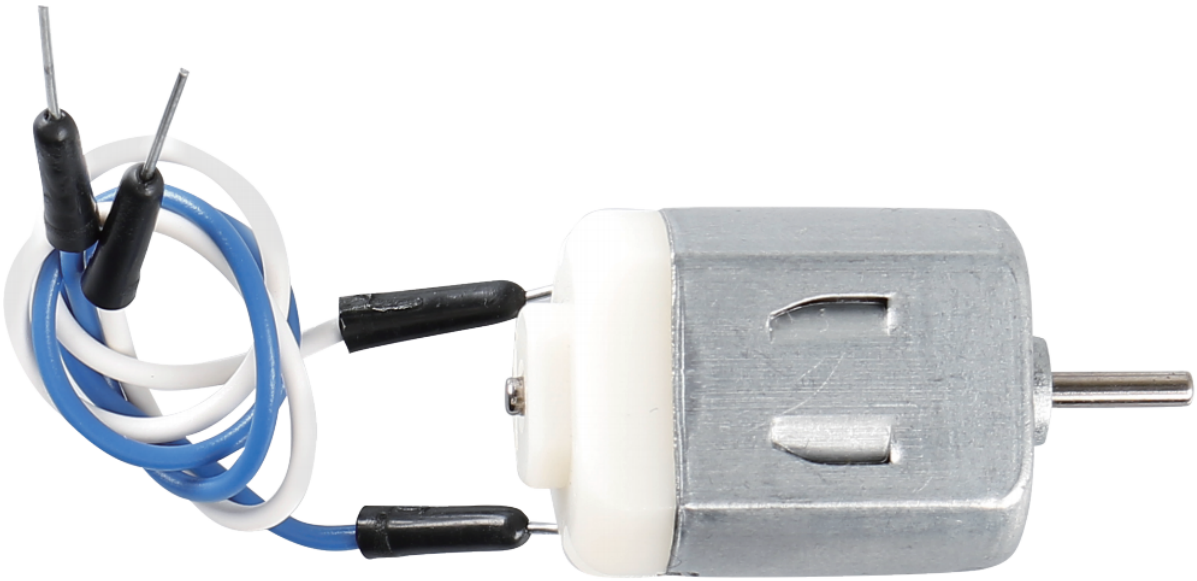
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.19 DC Motor

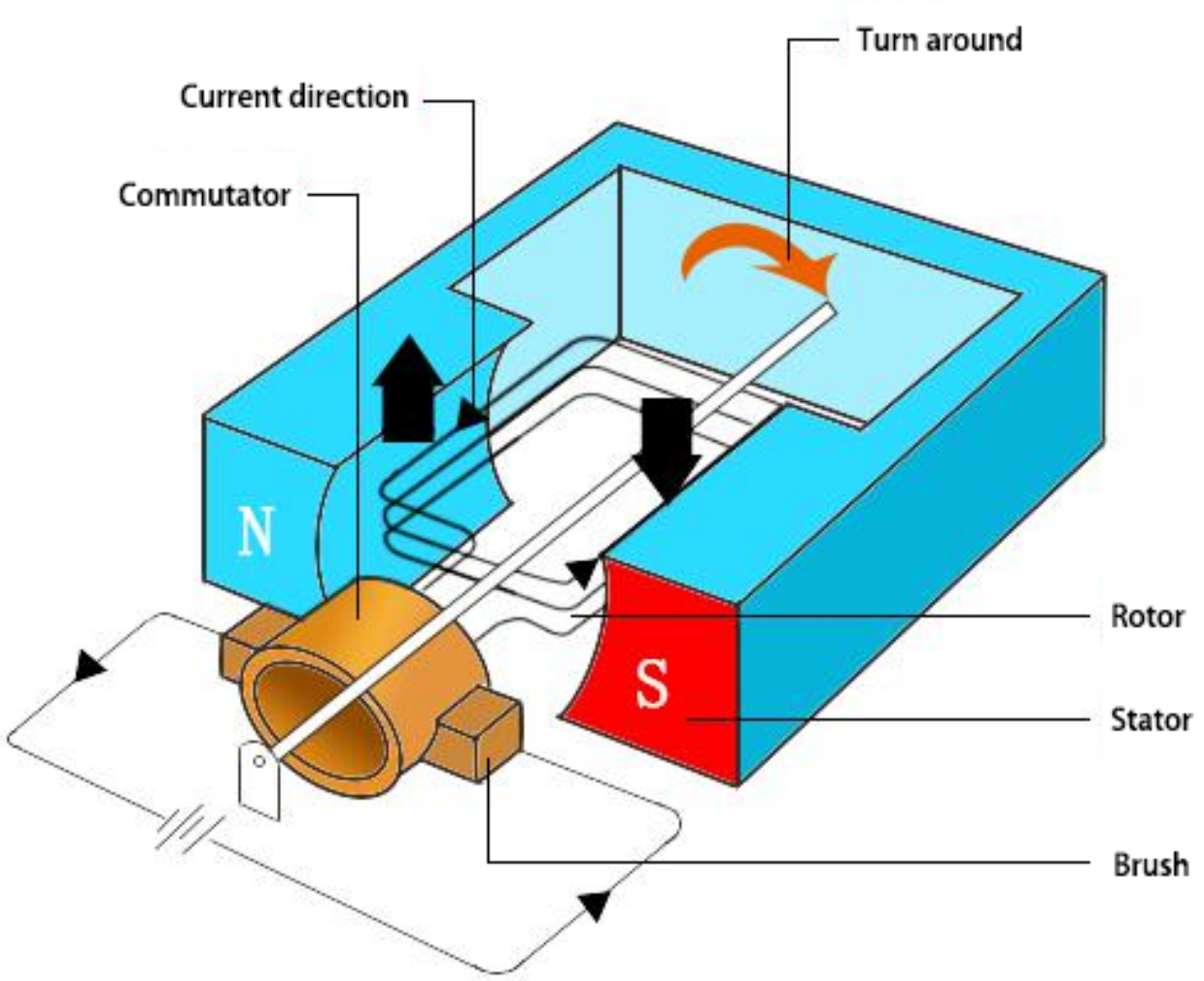


This is a 3V DC motor. When you give a high level and a low level to each of the 2 terminals, it will rotate.

- **Size**: 25*20*15MM
- **Operation Voltage**: 1-6V
- **Free-run Current** (3V): 70m
- **A Free-run Speed** (3V): 13000RPM
- **Stall Current** (3V): 800mA
- **Shaft Diameter**: 2mm

Direct current (DC) motor is a continuous actuator that converts electrical energy into mechanical energy. DC motors make rotary pumps, fans, compressors, impellers, and other devices work by producing continuous angular rotation.

A DC motor consists of two parts, the fixed part of the motor called the **stator** and the internal part of the motor called the **rotor** (or **armature** of a DC motor) that rotates to produce motion. The key to generating motion is to position the armature within the magnetic field of the permanent magnet (whose field extends from the north pole to the south pole). The interaction of the magnetic field and the moving charged particles (the current-carrying wire generates the magnetic field) produces the torque that rotates the armature.



Current flows from the positive terminal of the battery through the circuit, through the copper brushes to the commutator, and then to the armature. But because of the two gaps in the commutator, this flow reverses halfway through each complete rotation. This continuous reversal essentially converts the DC power from the battery to AC, allowing the armature to experience torque in the right direction at the right time to maintain rotation.

- DC Motor - MagLab
- Fleming's left-hand rule for motors - Wikipedia

Example

- *3.5 Small Fan* (For MicroPython User)
- *3.5 - Small Fan* (For Arduino User)
- *2.12 Smart Fan* (For Piper Make User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

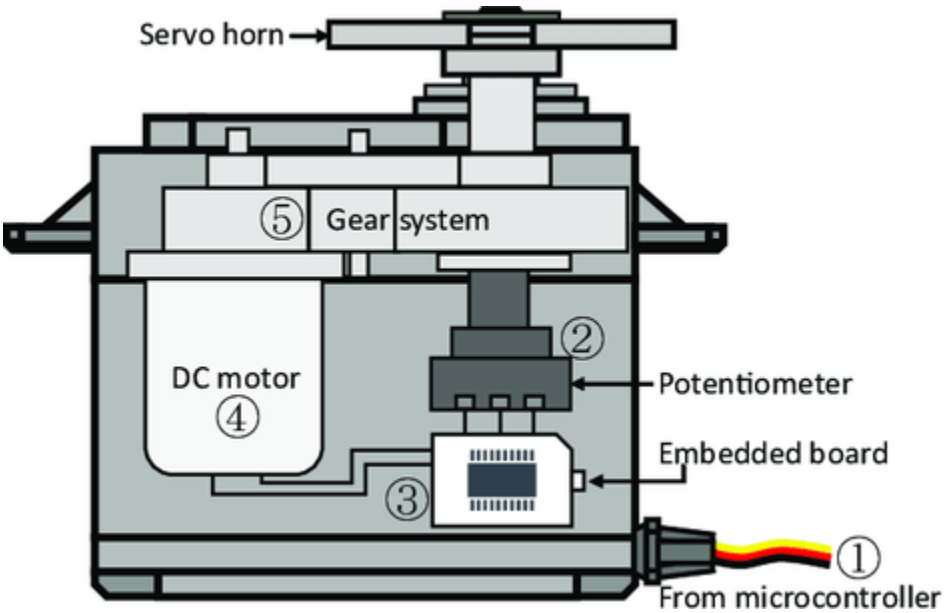
Ready to explore and create with us? Click [] and join today!

2.20 Servo



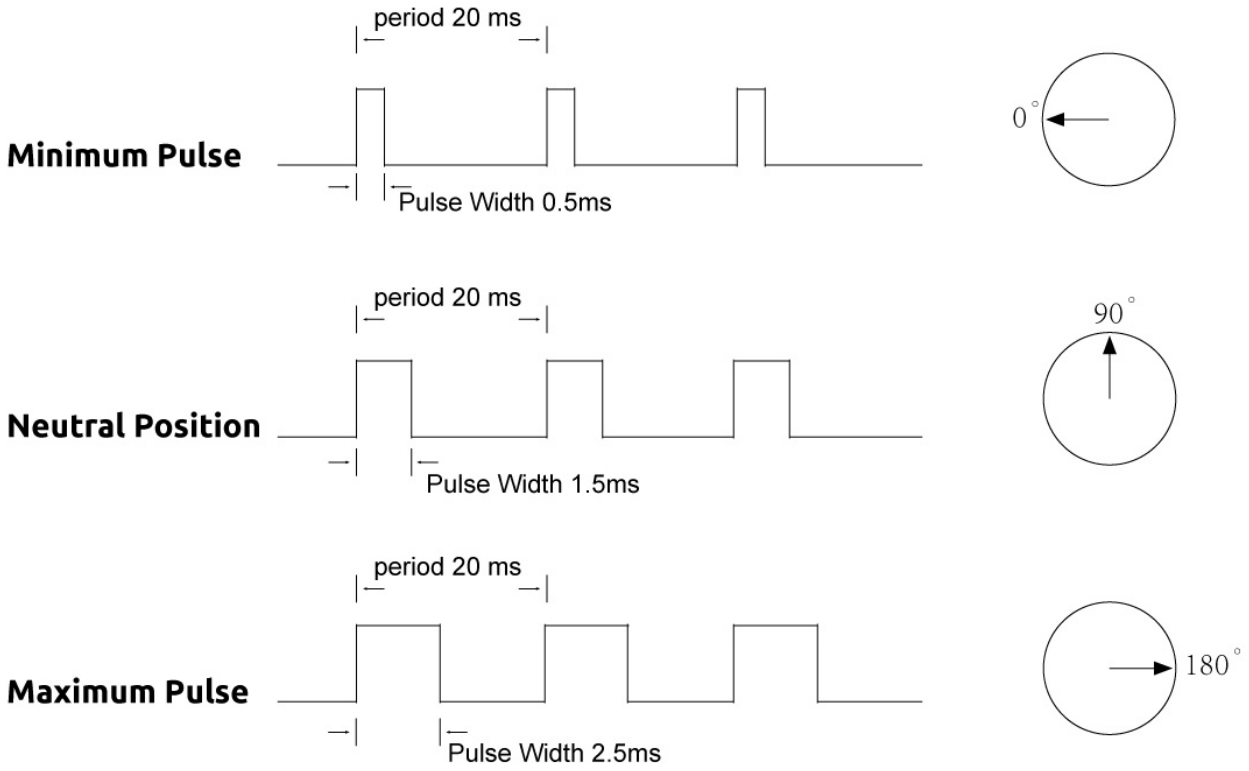
A servo is generally composed of the following parts: case, shaft, gear system, potentiometer, DC motor, and embedded board.

It works like this: The microcontroller sends out PWM signals to the servo, and then the embedded board in the servo receives the signals through the signal pin and controls the motor inside to turn. As a result, the motor drives the gear system and then motivates the shaft after deceleration. The shaft and potentiometer of the servo are connected together. When the shaft rotates, it drives the potentiometer, so the potentiometer outputs a voltage signal to the embedded board. Then the board determines the direction and speed of rotation based on the current position, so it can stop exactly at the right position as defined and hold there.



The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse width Modulation. The servo expects to see a pulse every 20 ms. The length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90 degree position (neutral position). When a pulse is sent to a servo that is less than 1.5 ms, the servo rotates to a position and holds its output shaft some number of degrees counterclockwise from the neutral point. When the pulse is wider than 1.5 ms the opposite occurs. The minimal width

and the maximum width of pulse that will command the servo to turn to a valid position are functions of each servo. Generally the minimum pulse will be about 0.5 ms wide and the maximum pulse will be 2.5 ms wide.



Example

- *3.7 Swinging Servo* (For MicroPython User)
- *7.11 Somatosensory Controller* (For MicroPython User)
- *3.7 - Swinging Servo* (For Arduino User)
- *2.6 Smart Water Tank* (For Piper Make User)
- *2.7 Swing Servo* (For Piper Make User)
- *2.9 Lucky Cat* (For Piper Make User)

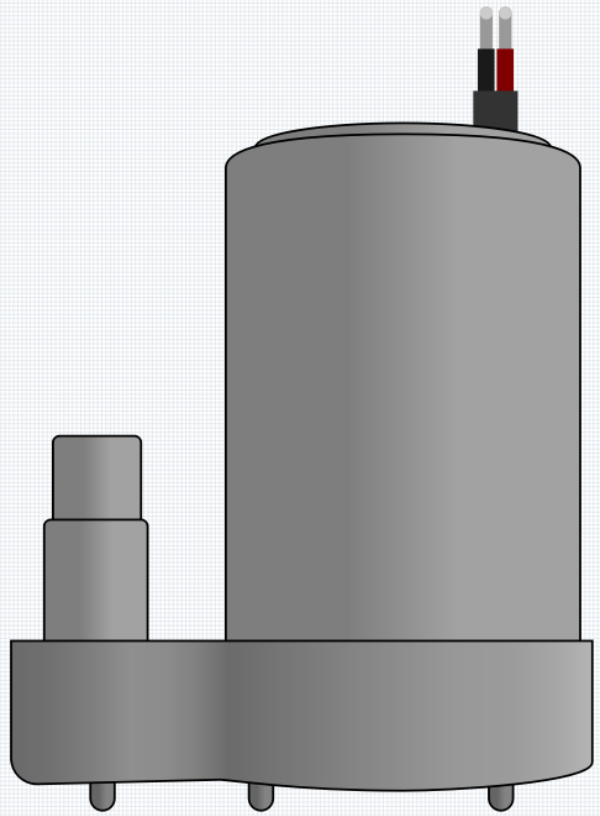
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.21 DC Water Pump



This pump essentially functions as a DC motor, operating at a voltage of 3V and a current of 100mA. Upon powering, the pump draws water in from the bottom of its plastic casing and expels it from the outlet pipe. It must always be kept immersed in water to function properly. Reversing the polarity won't turn it into a water intake device; it will only pump water out!

It's highly suitable for beginners to create a fountain or plant watering project using this submersible pump, as it is incredibly user-friendly!

Features

- **Voltage Scope**: DC 3 ~ 4.5V
- **Operating Current**: 120 ~ 180mA
- **Power**: 0.36 ~ 0.91W
- **Max Water Head**: 0.35 ~ 0.55M
- **Max Flow Rate**: 80 ~ 100 L/H
- **Continuous Working Life**: 100 hours
- **Water Fing Grade**: IP68
- **Driving Mode**: DC, Magnetic Driving
- **Material**: Engineering Plastic
- **Outlet Outside Diameter**: 7.8 mm
- **Outlet Inside Diameter**: 6.5 mm

- It is a submersible pump and should be used that way. It tends to heat too much that there's a risk of overheating if you turn it on unsubmerged.
- It comes with a 25cm male wire, allowing for easy insertion into a breadboard.

Example

- *3.6 Pumping* (For MicroPython User)
- *3.6 - Pumping* (For Arduino User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.22 Relay



As we may know, relay is a device which is used to provide connection between two or more points or devices in response to the input signal applied. In other words, relays provide isolation between the controller and the device as devices may work on AC as well as on DC. However, they receive signals from a microcontroller which works on DC hence requiring a relay to bridge the gap. Relay is extremely useful when you need to control a large amount of current or voltage with small electrical signal.

There are 5 parts in every relay:

Electromagnet - It consists of an iron core wounded by coil of wires. When electricity is passed through, it becomes magnetic. Therefore, it is called electromagnet.

Armature - The movable magnetic strip is known as armature. When current flows through them, the coil is it energized thus producing a magnetic field which is used to make or break the normally open (N/O) or normally close (N/C) points. And the armature can be moved with direct current (DC) as well as alternating current (AC).

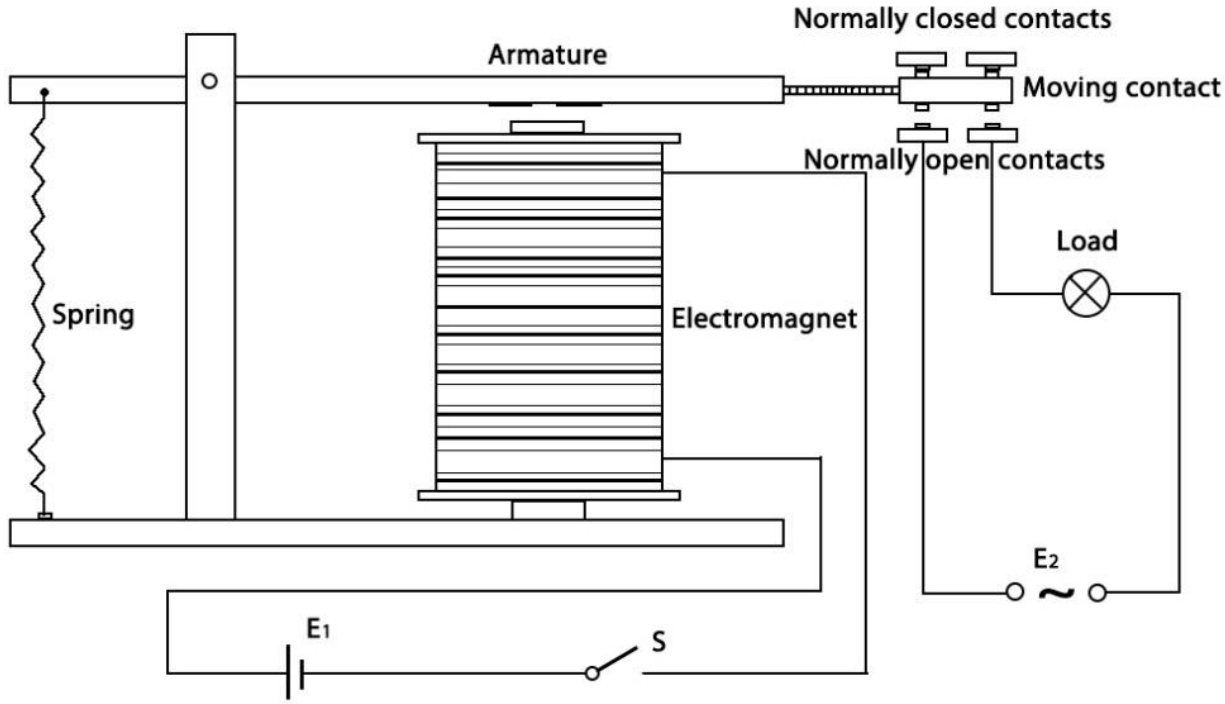
Spring - When no currents flow through the coil on the electromagnet, the spring pulls the armature away so the circuit cannot be completed.

Set of electrical **contacts** - There are two contact points:

- Normally open - connected when the relay is activated, and disconnected when it is inactive.
- Normally close - not connected when the relay is activated, and connected when it is inactive.

Molded frame - Relays are covered with plastic for protection.

The working principle of relay is simple. When power is supplied to the relay, currents start flowing through the control coil; as a result, the electromagnet starts energizing. Then the armature is attracted to the coil, pulling down the moving contact together thus connecting with the normally open contacts. So the circuit with the load is energized. Then breaking the circuit would a similar case, as the moving contact will be pulled up to the normally closed contacts under the force of the spring. In this way, the switching on and off of the relay can control the state of a load circuit.



- Relay - Wikipedia

Example

- *2.16 Control Another Circuit* (For MicroPython User)
- *2.16 - Control Another Circuit* (For Arduino User)

Controller

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

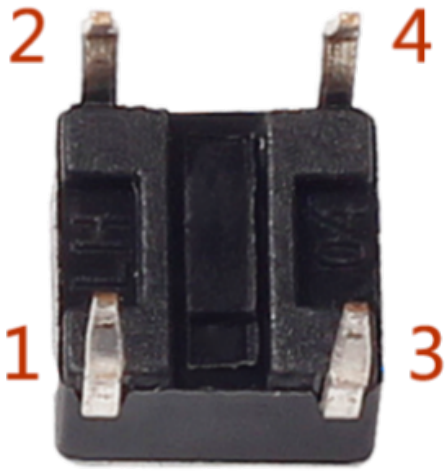
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

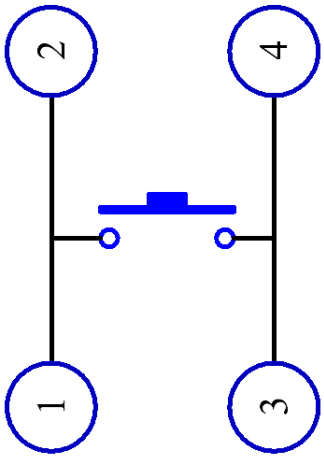
2.23 Button

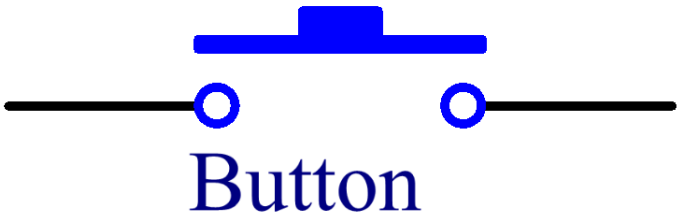




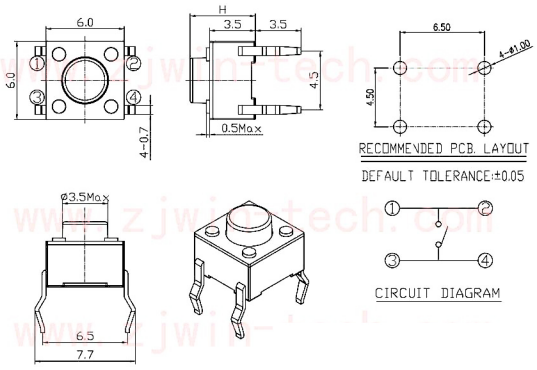
Buttons are a common component used to control electronic devices. They are usually used as switches to connect or break circuits. Although buttons come in a variety of sizes and shapes, the one used here is a 6mm mini-button as shown in the following pictures. Pin 1 is connected to pin 2 and pin 3 to pin 4. So you just need to connect either of pin 1 and pin 2 to pin 3 or pin 4.

The following is the internal structure of a button. The symbol on the right below is usually used to represent a button in circuits.





Since the pin 1 is connected to pin 2, and pin 3 to pin 4, when the button is pressed, the 4 pins are connected, thus closing the circuit.



Example

- *2.5 Reading Button Value* (For MicroPython User)
- *2.5 - Reading Button Value* (For Arduino User)
- *2.2 Button* (For Piper Make User)
- *2.4 Rainbow Light* (For Piper Make User)
- *2.5 Drum Kit* (For Piper Make User)
- *2.13 Reaction Game* (For Piper Make User)

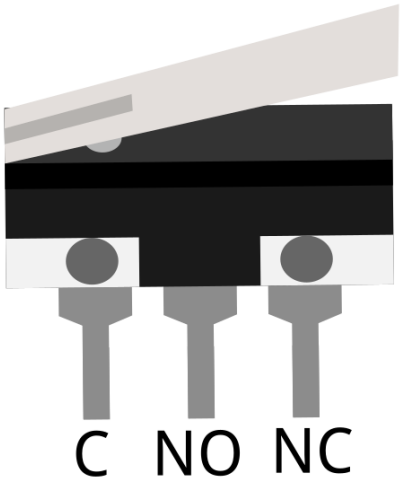
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

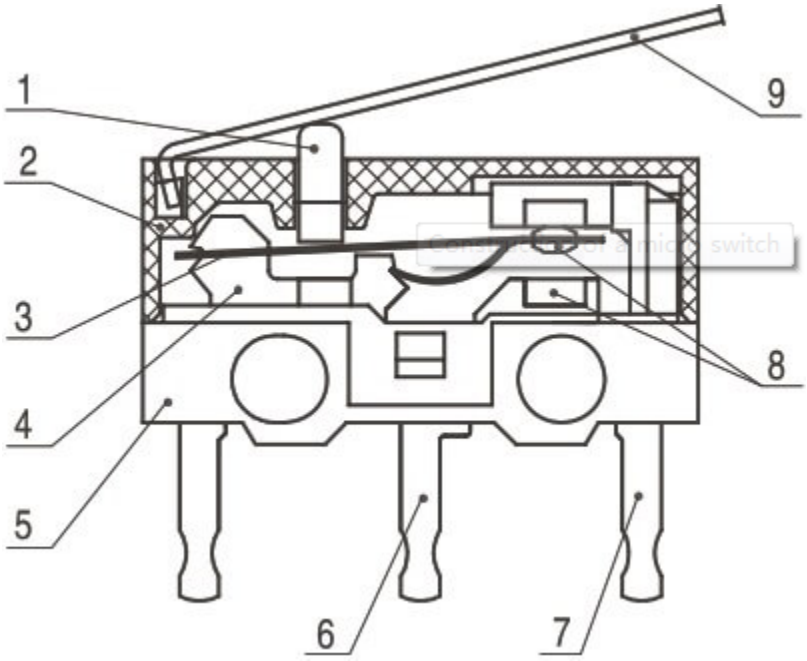
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.24 Micro Switch



The construction of a micro switch is really simple. The main parts of the switch are:



- 1.Plunger (Actuator)
- 2.Cover
- 3.Moving piece
- 4.Support
- 5.Case
- 6.NO terminal: normally open
- 7.NC terminal: normally closed
- 8.Contact

- 9.Moving arm

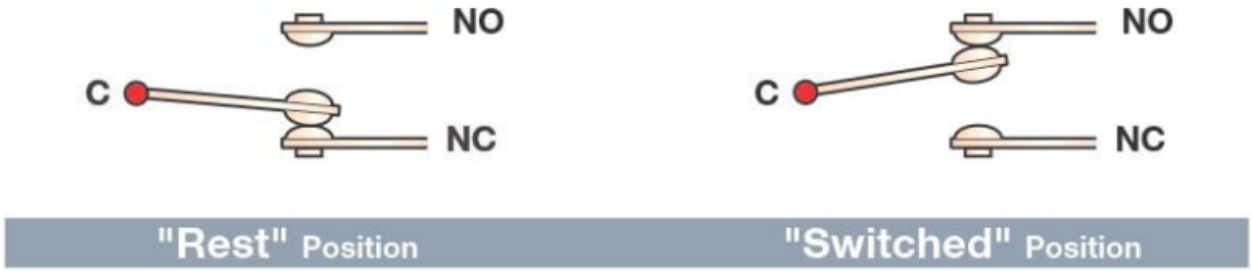
After a micro switch makes physical contact with an object, its contacts change position. The basic working principle is as follows.

When the plunger is in the released or rest position.

- The normally closed circuit can carry current.
- The normally open circuit is electrically insulated.

When the plunger is depressed or switched.

- The normally closed circuit is open.
- The normally open circuit is closed.



Example

- *2.8 Press Gently* (For MicroPython User)
- *2.8 - Press Gently* (For Arduino User)
- *2.3 Service Bell* (For Piper Make User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

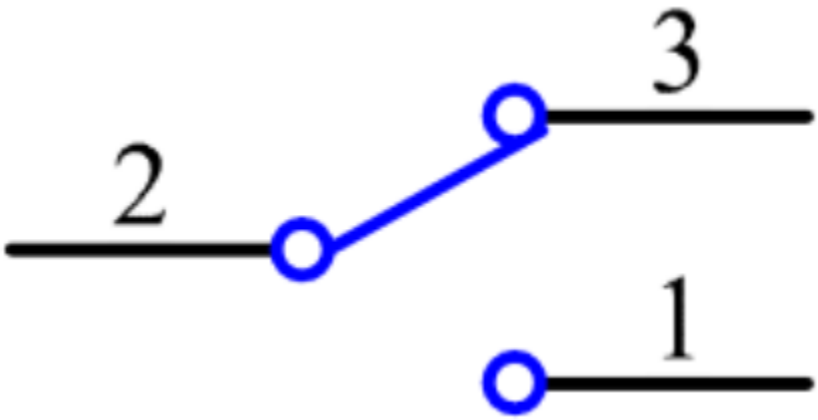
2.25 Slide Switch



A slide switch, just as its name implies, is to slide the switch bar to connect or break the circuit, and further switch circuits. The common-used types are SPDT, SPTT, DPDT, DPTT etc. The slide switch is commonly used in low-voltage circuit. It has the features of flexibility and stability, and applies in electric instruments and electric toys widely. How it works: Set the middle pin as the fixed one. When you pull the slide to the left, the two pins on the left are connected; when you pull it to the right, the two pins on the right are connected. Thus, it works as a switch connecting or disconnecting circuits. See the figure below:



The circuit symbol of the slide switch is shown as below. The pin2 in the figure refers to the middle pin.



Example

- *2.7 Toggle Left and Right* (For MicroPython User)
- *7.3 Alarm Siren Lamp* (For MicroPython User)
- *2.7 - Toggle Left and Right* (For Arduino User)
- *2.5 Drum Kit* (For Piper Make User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook!

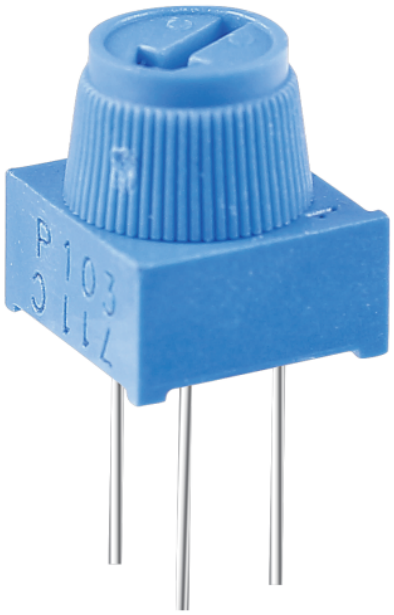
Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.26 Potentiometer

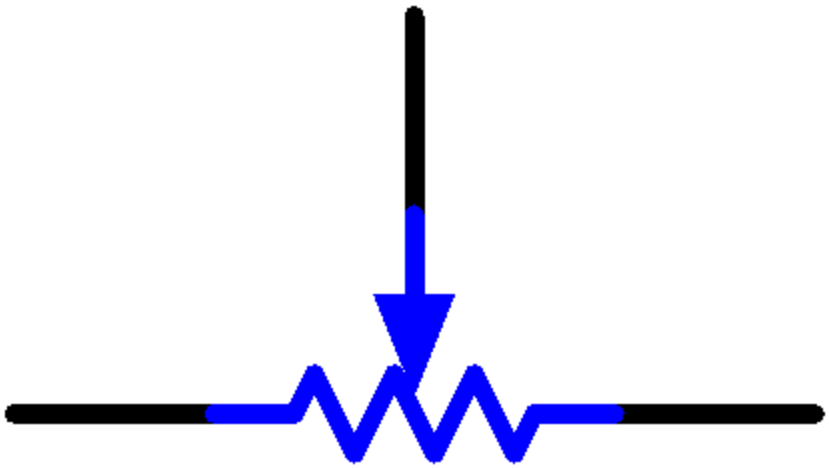


Potentiometer is also a resistance component with 3 terminals and its resistance value can be adjusted according to some regular variation.

Potentiometers come in various shapes, sizes, and values, but they all have the following things in common:

- They have three terminals (or connection points).
- They have a knob, screw, or slider that can be moved to vary the resistance between the middle terminal and either one of the outer terminals.
- The resistance between the middle terminal and either one of the outer terminals varies from 0 to the maximum resistance of the pot as the knob, screw, or slider is moved.

Here is the circuit symbol of potentiometer.



The functions of the potentiometer in the circuit are as follows:

1. Serving as a voltage divider

   Potentiometer is a continuously adjustable resistor. When you adjust the shaft or sliding handle of the potentiometer, the movable contact will slide on the resistor. At this point, a voltage can be output depending on the voltage applied onto the potentiometer and the angle the movable arm has rotated to or the travel it has made.

2. Serving as a rheostat

   When the potentiometer is used as a rheostat, connect the middle pin and one of the other 2 pins in the circuit. Thus you can get a smoothly and continuously changed resistance value within the travel of the moving contact.

3. Serving as a current controller

   When the potentiometer acts as a current controller, the sliding contact terminal must be connected as one of the output terminals.

If you want to know more about potentiometer, refer to: Potentiometer - Wikipedia

Example

- *2.11 Turn the Knob* (For MicroPython User)
- *2.11 - Turn the Knob* (For Arduino User)
- *2.7 Swing Servo* (For Piper Make User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.27 Infrared Receiver

2.27.1 IR Receiver



- S: Signal output
- +: VCC
- -: GND

An infrared-receiver is a component which receives infrared signals and can independently receive infrared rays and output signals compatible with TTL level. It is similar with a normal plastic-packaged transistor in size and is suitable for all kinds of infrared remote control and infrared transmission.

Infrared, or IR, communication is a popular, low-cost, easy-to-use wireless communication technology. Infrared light has a slightly longer wavelength than visible light, so it is imperceptible to the human eye - ideal for wireless communication. A common modulation scheme for infrared communication is 38KHz modulation.

- Adopted HX1838 IR Receiver Sensor, high sensitivity
- Can be used for remote control
- Power Supply: 3.3~5V
- Interface: Digital
- Modulate Frequency: 38Khz

2.27.2 Remote Control



This is a Mini thin infrared wireless remote control with 21 function buttons and a transmitting distance of up to 8 meters, which is suitable for operating a wide range of devices in a kid's room.

- Size: 85x39x6mm
- Remote control range: 8-10m
- Battery: 3V button type lithium manganese battery
- Infrared carrier frequency: 38KHz
- Surface paste material: 0.125mm PET
- Effective life: more than 20,000 times

Example

- *6.4 IR Remote Control* (For MicroPython User)
- *6.4 - IR Remote Control* (For Arduino User)

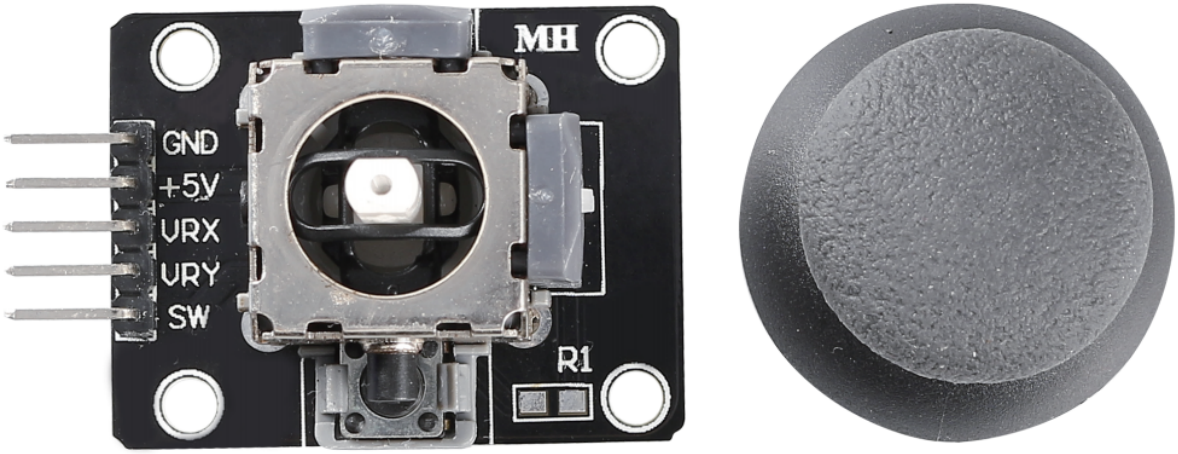
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.28 Joystick Module

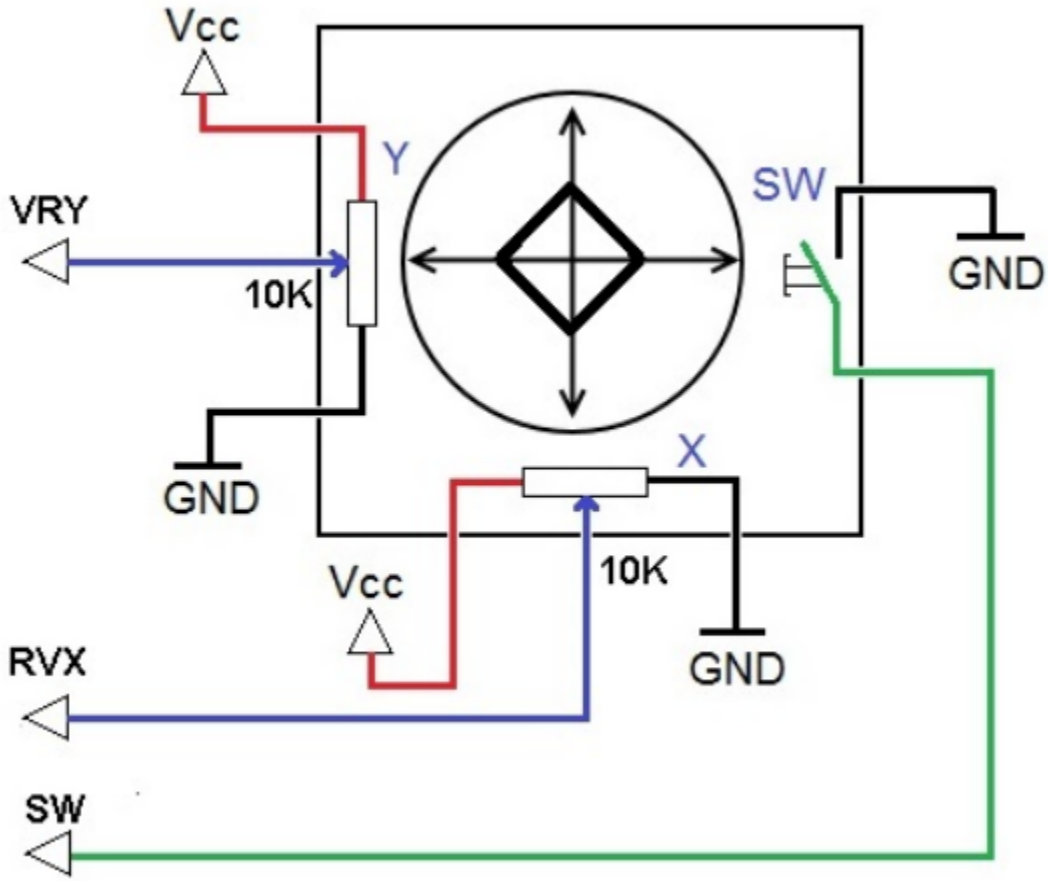


The basic idea of a joystick is to translate the movement of a stick into electronic information that a computer can process.

In order to communicate a full range of motion to the computer, a joystick needs to measure the stick's position on two axes – the X-axis (left to right) and the Y-axis (up and down). Just as in basic geometry, the X-Y coordinates pinpoint the stick's position exactly.

To determine the location of the stick, the joystick control system simply monitors the position of each shaft. The conventional analog joystick design does this with two potentiometers, or variable resistors.

The joystick also has a digital input that is actuated when the joystick is pressed down.



- Joystick - Wikipedia

Example

- *4.1 Toggle the Joystick* (For MicroPython User)
- *4.1 - Toggle the Joystick* (For Arduino User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

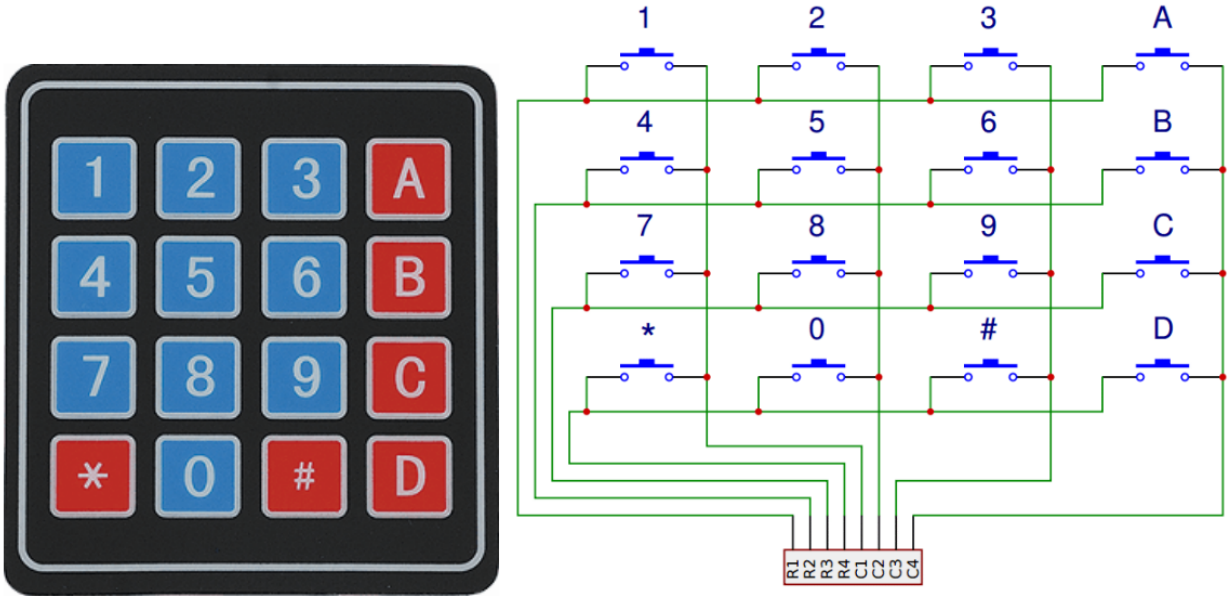
Ready to explore and create with us? Click [] and join today!

2.29 4x4 Keypad

Microcontroller system, if the use of more keys such as electronic code lock, telephone keypad, etc. generally have at least 12 to 16 keys, usually using a matrix keyboard.

Matrix keypad is also called row keypad, it is a keypad with four I/O lines as row lines and four I/O lines as column lines. One key is set at each intersection of the row and column lines. Thus the number of keys on the keyboard is 4*4. This row and column keyboard structure can effectively improve the utilization of I/O ports in a microcontroller system.

Their contacts are accessed via a header suitable for connection with a ribbon cable or insertion into a printed circuit board. In some keypads, each button connects with a separate contact in the header, while all the buttons share a common ground.



More often, the buttons are matrix encoded, meaning that each of them bridges a unique pair of conductors in a matrix. This configuration is suitable for polling by a microcontroller, which can be programmed to send an output pulse to each of the four horizontal wires in turn. During each pulse, it checks the remaining four vertical wires in sequence, to determine which one, if any, is carrying a signal. Pullup or pulldown resistors should be added to the input wires to prevent the inputs of the microcontroller from behaving unpredictably when no signal is present.

- Keypad - Wikipedia

Example

- *4.2 4x4 Keypad* (For MicroPython User)
- *7.7 Guess Number* (For MicroPython User)
- *4.2 4x4 Keypad* (For Arduino User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

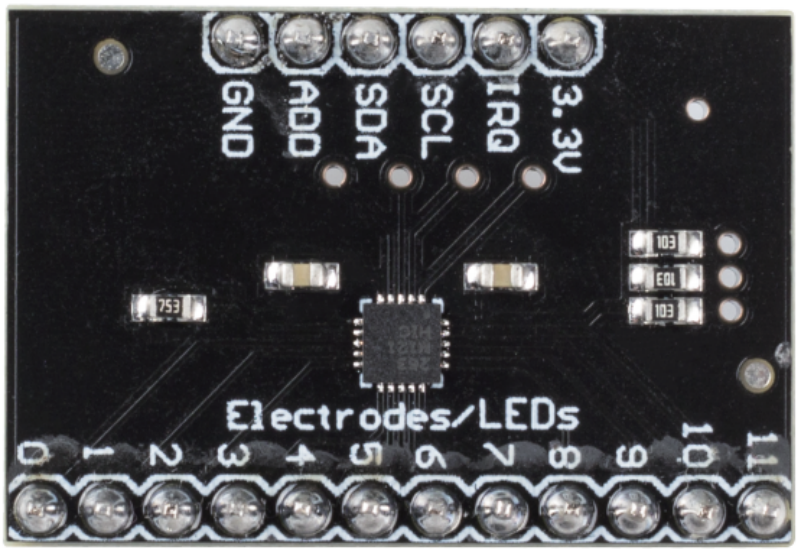
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.30 MPR121 Module



- **3.3V**: Power supply
- **IRQ**: Open Collector Interrupt Output Pin, active low
- **SCL**: I2C Clock
- **SDA**: I2C Data
- **ADD**: I2C Address Select Input Pin. Connect the ADDR pin to the VSS, VDD, SDA or SCL line, the resulting I2C addresses are 0x5A, 0x5B, 0x5C and 0x5D respectively
- **GND**: Ground
- **0~11**: Electrode 0~11, electrode is a touch sensor. Typically, electrodes can just be some piece of metal, or a wire. But some times depending on the length of our wire, or the material the electrode is on, it can make triggering the sensor difficult. For this reason, the MPR121 allows you to configure what is needed to trigger and untrigger an electrode.

MPR121 OVERVIEW

The MPR121 is the second generation capacitive touch sensor controller after the initial release of the MPR03x series devices. The MPR121 features increased internal intelligence, some of the major additions include an increased electrode count, a hardware configurable I2C address, an expanded filtering system with debounce, and completely independent electrodes with auto-configuration built in. The device also features a 13th simulated sensing channel dedicated for near proximity detection using the multiplexed sensing inputs.

- MPR121 Datasheet

Features

- **Low power operation**
  - 1.71 V to 3.6 V supply operation
  - 29 A supply current at 16 ms sampling interval period
  - 3 A Stop mode current
- **12 capacitance sensing inputs**
  - 8 inputs are multifunctional for LED driver and GPIO
- **Complete touch detection**
  - Auto-configuration for each sensing input
  - Auto-calibration for each sensing input
  - Touch/release threshold and debounce for touch detection
- I2C interface, with Interrupt output
- 3 mm x 3 mm x 0.65 mm 20 lead QFN package
- -40°C to +85°C operating temperature range

Example

- *4.3 Electrode Keyboard* (For MicroPython User)
- *7.9 Fruit Piano* (For MicroPython User)
- *4.3 - Electrode Keyboard* (For Arduino User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.31 MFRC522 Module



MFRC522 is a kind of integrated read and write card chip. It is commonly used in the radio at 13.56MHz. Launched by the NXP Company, it is a low-voltage, low-cost, and small-sized non-contact card chip, a best choice of intelligent instrument and portable handheld device.

The MF RC522 uses advanced modulation and demodulation concept which fully presented in all types of 13.56MHz passive contactless communication methods and protocols. In addition, it supports rapid CRYPTO1 encryption algorithm to verify MIFARE products. MFRC522 also supports MIFARE series of high-speed non-contact communication, with a two-way data transmission rate up to 424kbit/s. As a new member of the 13.56MHz highly integrated reader card series, MF RC522 is much similar to the existing MF RC500 and MF RC530 but there also exists great differences. It communicates with the host machine via the serial manner which needs less wiring. You can choose between SPI, I2C and serial UART mode (similar to RS232), which helps reduce the connection, save PCB board space (smaller size), and reduce cost.

- MFRC522 Data sheet

Example

- *6.5 Radio Frequency Identification* (For MicroPython User)
- *7.8 RFID Music Player* (For MicroPython User)
- *6.5 - Radio Frequency Identification* (For Arduino User)

Sensor

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

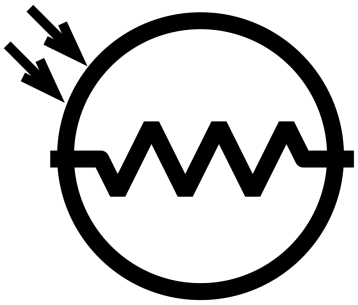
2.32 Photoresistor



A photoresistor or photocell is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity; in other words, it exhibits photo conductivity.

A photoresistor can be applied in light-sensitive detector circuits and light-activated and dark-activated switching circuits acting as a resistance semiconductor. In the dark, a photoresistor can have a resistance as high as several megaohms (M), while in the light, a photoresistor can have a resistance as low as a few hundred ohms.

Here is the electronic symbol of photoresistor.



- Photoresistor - Wikipedia

Example

- *2.12 Feel the Light* (For MicroPython User)
- *7.1 Light Theremin* (For MicroPython User)
- *2.12 - Feel the Light* (For Arduino User)
- *2.8 Light Intensity Display* (For Piper Make User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

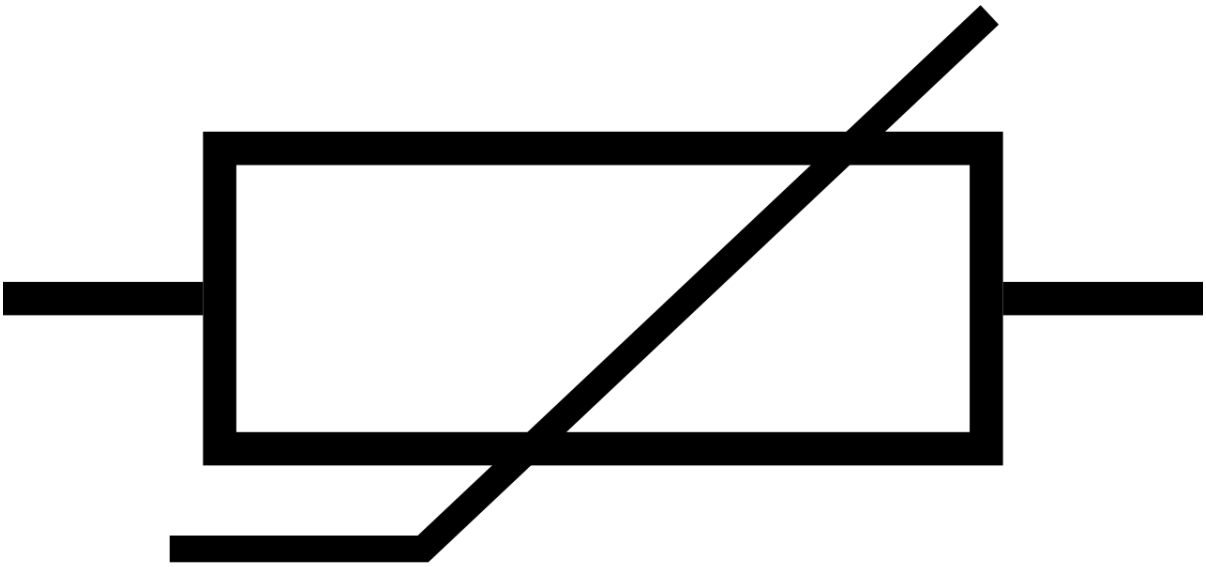
2.33 Thermistor



A thermistor is a type of resistor whose resistance is strongly dependent on temperature, more so than in standard resistors. The word is a combination of thermal and resistor. Thermistors are widely used as inrush current limiters, temperature sensors (negative temperature coefficient or NTC type typically), self-resetting overcurrent protectors, and self-regulating heating elements (positive temperature coefficient or PTC type typically).

- Thermistor - Wikipedia

Here is the electronic symbol of thermistor.



Thermistors are of two opposite fundamental types:

- With NTC thermistors, resistance decreases as temperature rises usually due to an increase in conduction electrons bumped up by thermal agitation from valency band. An NTC is commonly used as a temperature sensor, or in series with a circuit as an inrush current limiter.
- With PTC thermistors, resistance increases as temperature rises usually due to increased thermal lattice agitations particularly those of impurities and imperfections. PTC thermistors are commonly installed in series with a circuit, and used to protect against overcurrent conditions, as resettable fuses.

In this kit we use an NTC one. Each thermistor has a normal resistance. Here it is 10k ohm, which is measured under 25 degree Celsius.

Here is the relation between the resistance and temperature:

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

- **RT** is the resistance of the NTC thermistor when the temperature is TK.
- **RN** is the resistance of the NTC thermistor under the rated temperature TN. Here, the numerical value of RN is 10k.
- **TK** is a Kelvin temperature and the unit is K. Here, the numerical value of TK is 273.15 + degree Celsius.
- **TN** is a rated Kelvin temperature; the unit is K too. Here, the numerical value of TN is 273.15+25.
- And **B(beta)**, the material constant of NTC thermistor, is also called heat sensitivity index with a numerical value 3950.
- **exp** is the abbreviation of exponential, and the base number e is a natural number and equals 2.7 approximately.

Convert this formula TK=1/(ln(RT/RN)/B+1/TN) to get Kelvin temperature that minus 273.15 equals degree Celsius.

This relation is an empirical formula. It is accurate only when the temperature and resistance are within the effective range.

Example

- *2.13 Thermometer* (For MicroPython User)
- *7.2 Room Temperature Meter* (For MicroPython User)
- *2.13 - Thermometer* (For Arduino User)

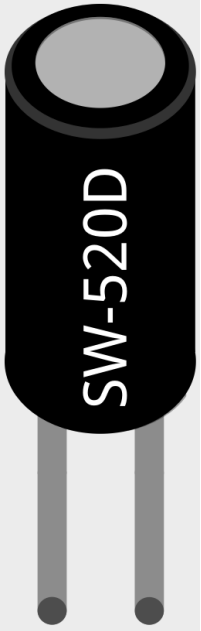
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

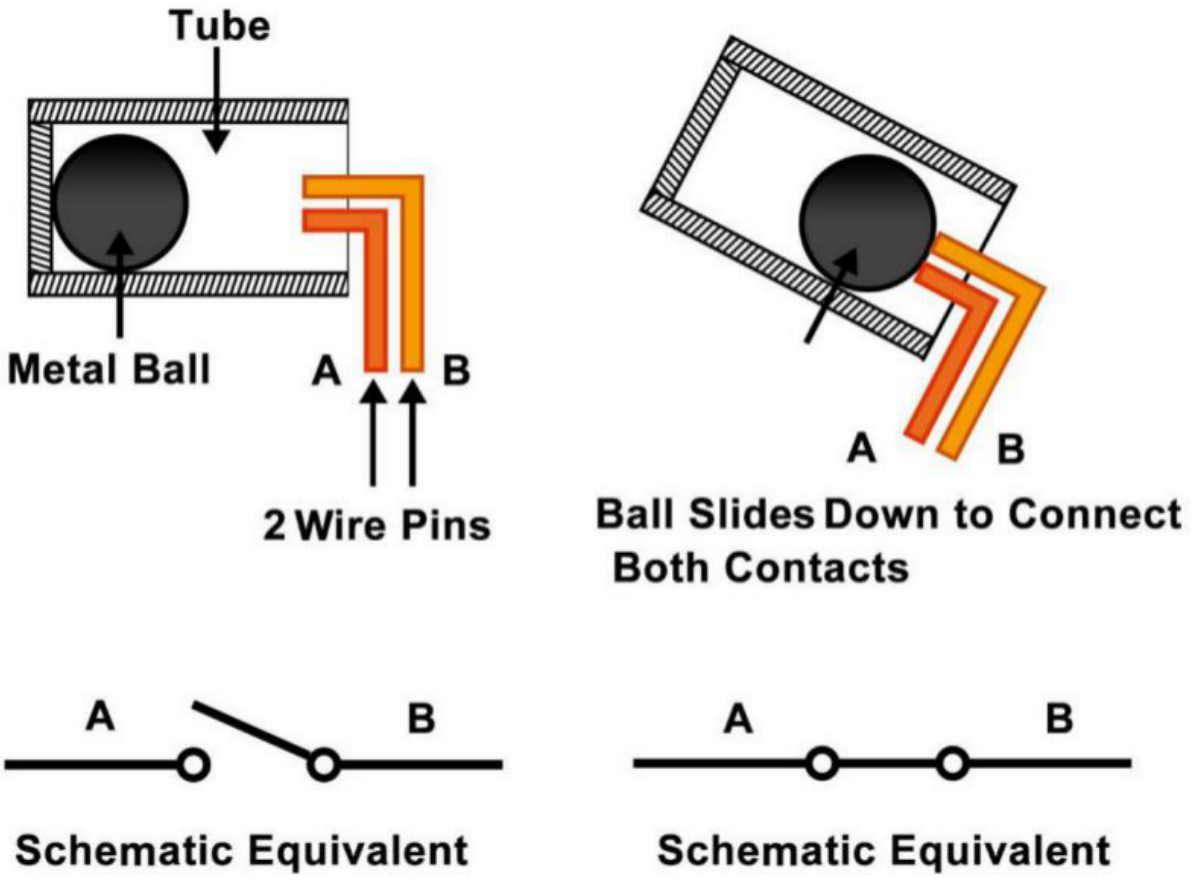
Ready to explore and create with us? Click [] and join today!

2.34 Tilt Switch



The tilt switch used here is a ball one with a metal ball inside. It is used to detect inclinations of a small angle.

The principle is very simple. When the switch is tilted in a certain angle, the ball inside rolls down and touches the two contacts connected to the pins outside, thus triggering circuits. Otherwise the ball will stay away from the contacts, thus breaking the circuits.



- SW520D Tilt Switch Datasheet

Example

- *2.6 Tilt It* (For MicroPython User)
- *7.5 GAME - 10 Second* (For MicroPython User)
- *2.6 - Tilt It* (For Arduino User)
- *2.10 Flowing LEDs* (For Piper Make User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.35 Reed Switch

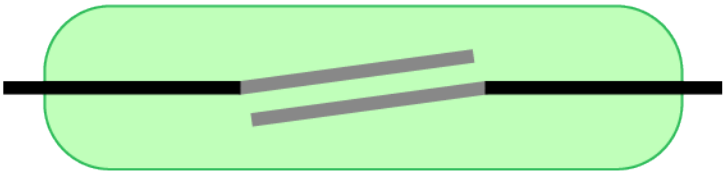


The Reed Switch is an electrical switch that operates by means of an applied magnetic field. It was invented by Walter B. Ellwood of Bell Telephone Laboratories in 1936 and patented in the United States on June 27, 1940, under patent number 2264746 .

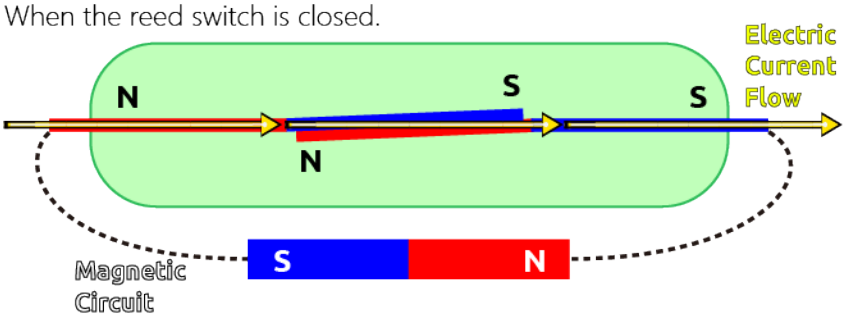
The principle of operation of a reed switch is very simple. Two reeds (usually made of iron and nickel, two metals) that overlap at the end points are sealed in a glass tube, with the two reeds overlapping and separated by a small gap (only about a few microns). The glass tube is filled with a high purity inert gas (such as nitrogen), and some reed switches are made to have a vacuum inside to enhance their high voltage performance.

The reed acts as a magnetic flux conductor. The two reeds are not in contact when not yet in operation; when passing through a magnetic field generated by a permanent magnet or electromagnetic coil, the applied magnetic field causes the two reeds to have different polarities near their endpoints, and when the magnetic force exceeds the spring force of the reeds themselves, the two reeds will be drawn together to conduct the circuit; when the magnetic field weakens or disappears, the reeds are released due to their own elasticity, and the contact surfaces will separate to open the circuit.

When the reed switch is open.



When the reed switch is closed.



- Reed Switch - Wikipedia

Example

- *2.9 Feel the Magnetism* (For MicroPython User)
- *2.9 - Feel the Magnetism* (For Arduino User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

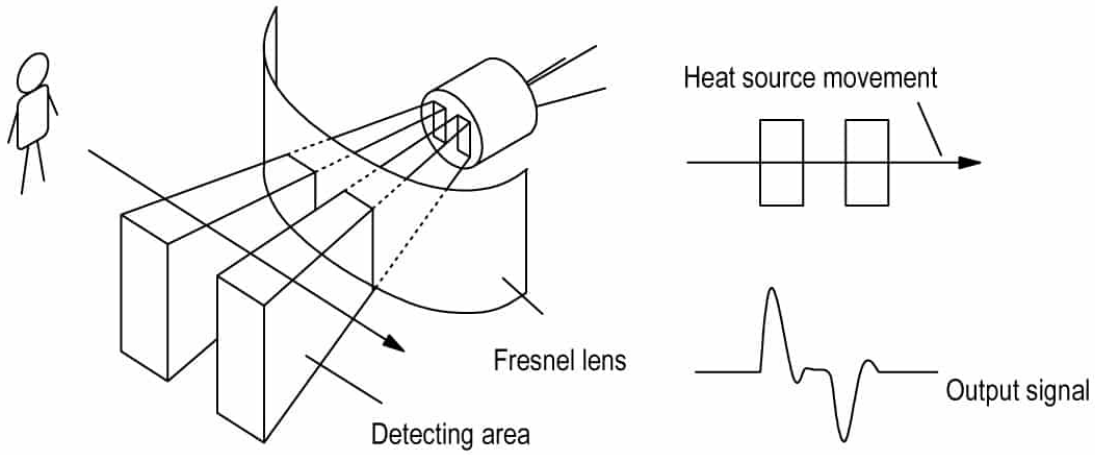
Ready to explore and create with us? Click [] and join today!

2.36 PIR Motion Sensor Module

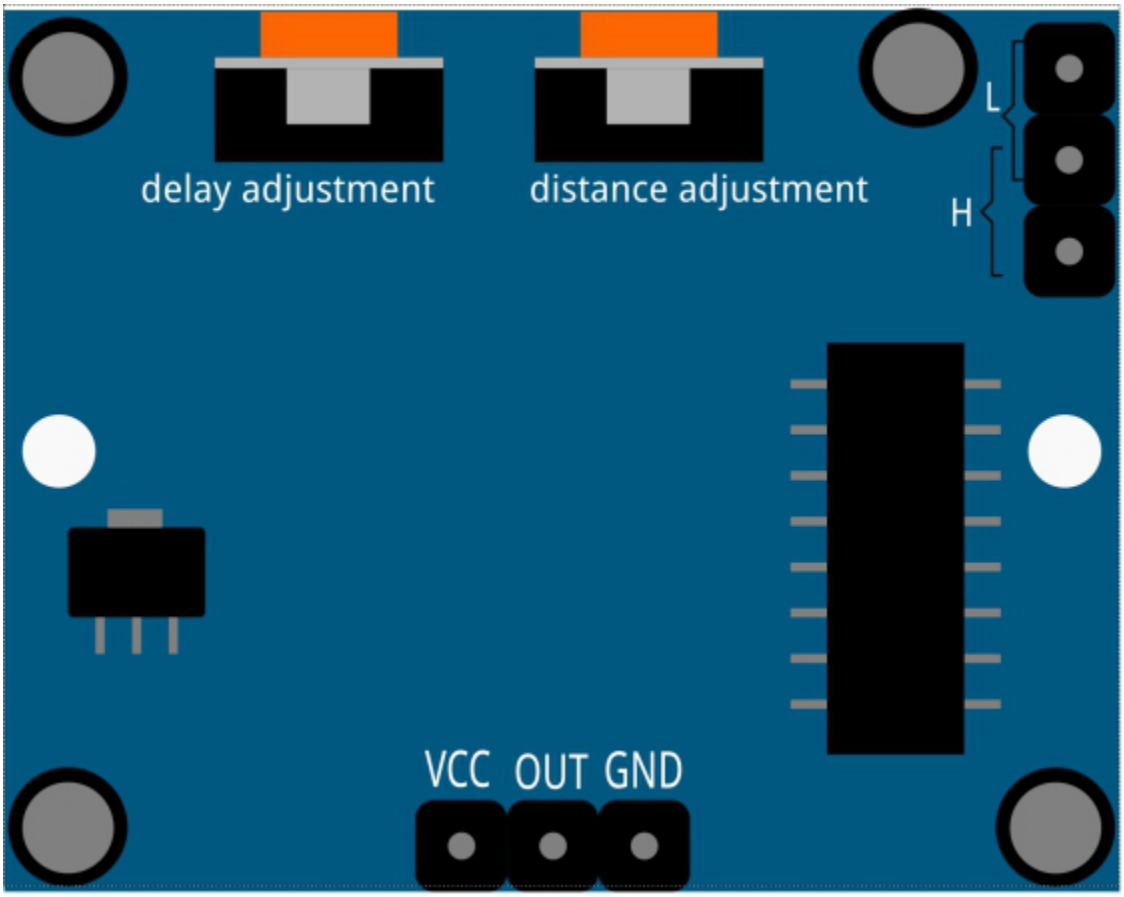


The PIR sensor detects infrared heat radiation that can be used to detect the presence of organisms that emit infrared heat radiation.

The PIR sensor is split into two slots that are connected to a differential amplifier. Whenever a stationary object is in front of the sensor, the two slots receive the same amount of radiation and the output is zero. Whenever a moving object is in front of the sensor, one of the slots receives more radiation than the other , which makes the output fluctuate high or low. This change in output voltage is a result of detection of motion.



After the sensing module is wired, there is a one-minute initialization. During the initialization, module will output for 0~3 times at intervals. Then the module will be in the standby mode. Please keep the interference of light source and other sources away from the surface of the module so as to avoid the misoperation caused by the interfering signal. Even you'd better use the module without too much wind, because the wind can also interfere with the sensor.



Distance Adjustment

Turning the knob of the distance adjustment potentiometer clockwise, the range of sensing distance increases, and the maximum sensing distance range is about 0-7 meters. If turn it anticlockwise, the range of sensing distance is reduced, and the minimum sensing distance range is about 0-3 meters.

Delay adjustment

Rotate the knob of the delay adjustment potentiometer clockwise, you can also see the sensing delay increasing. The maximum of the sensing delay can reach up to 300s. On the contrary, if rotate it anticlockwise, you can shorten the delay with a minimum of 5s.

Two Trigger Modes

Choosing different modes by using the jumper cap.

- **H**: Repeatable trigger mode, after sensing the human body, the module outputs high level. During the subsequent delay period, if somebody enters the sensing range,the output will keep being the high level.
- **L**: Non-repeatable trigger mode, outputs high level when it senses the human body. After the delay, the output will change from high level into low level automatically.

Example

- *2.10 Detect Human Movement* (For MicroPython User)

- *7.4 Passenger Counter* (For MicroPython User)
- *2.10 - Detect Human Movement* (For Arduino User)
- *2.9 Lucky Cat* (For Piper Make User)

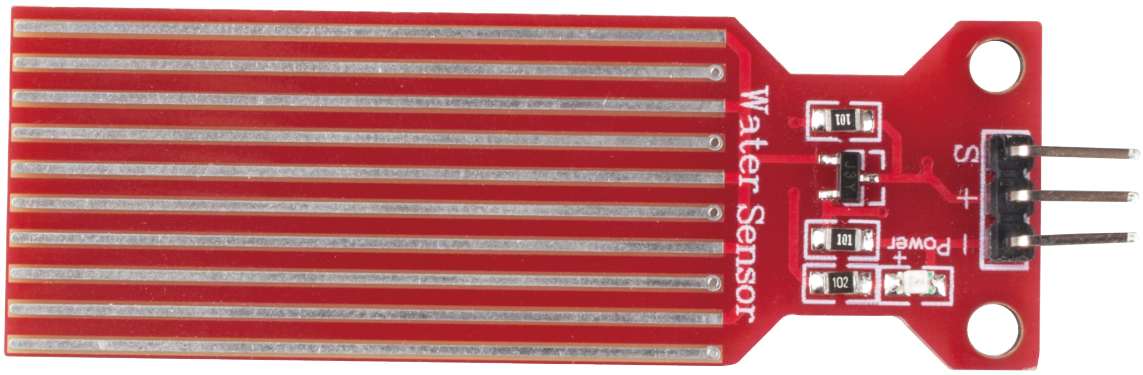
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.37 Water Level Sensor Module



The water level sensor transmits the sensed water level signal to the controller, and the computer in the controller compares the measured water level signal with the set signal to derive the deviation, and then issues “on” and “off” commands to the feedwater electric valve according to the nature of the deviation to ensure that the vessel reaches the set water level.

The water level sensor has ten exposed copper traces, five for the Power traces and five for the Sensor traces, which are crossed and bridged by water when flooded. The circuit board has a power LED that lights up when the board is energized.

The combination of these traces acts like a variable resistor, changing the resistance value according to the water level. To be more precise, the more water the sensor is immersed in, the better the conductivity and the lower the resistance. Conversely, the less conductive it is, the higher the resistance. Next, the sensor will process the output signal voltage which will be sent to the microcontroller, thus helping us to determine the water level.

Warning: The sensor cannot be fully submerged in water, please only leave the part where the ten traces are located in contact with water. In addition, energizing the sensor in a humid environment will speed up the corrosion of the probe and cut the life of the sensor, so we recommend that you only supply power when taking readings.

Example

- *2.14 Feel the Water Level* (For MicroPython User)
- *2.14 - Feel the Water Level* (For Arduino User)
- *2.6 Smart Water Tank* (For Piper Make User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.38 Ultrasonic Module



- **TRIG**: Trigger Pulse Input
- **ECHO**: Echo Pulse Output
- **GND**: Ground
- **VCC**: 5V Supply

This is the HC-SR04 ultrasonic distance sensor, providing non-contact measurement from 2 cm to 400 cm with a range accuracy of up to 3 mm. Included on the module is an ultrasonic transmitter, a receiver and a control circuit.

You only need to connect 4 pins: VCC (power), Trig (trigger), Echo (receive) and GND (ground) to make it easy to use for your measurement projects.

Features

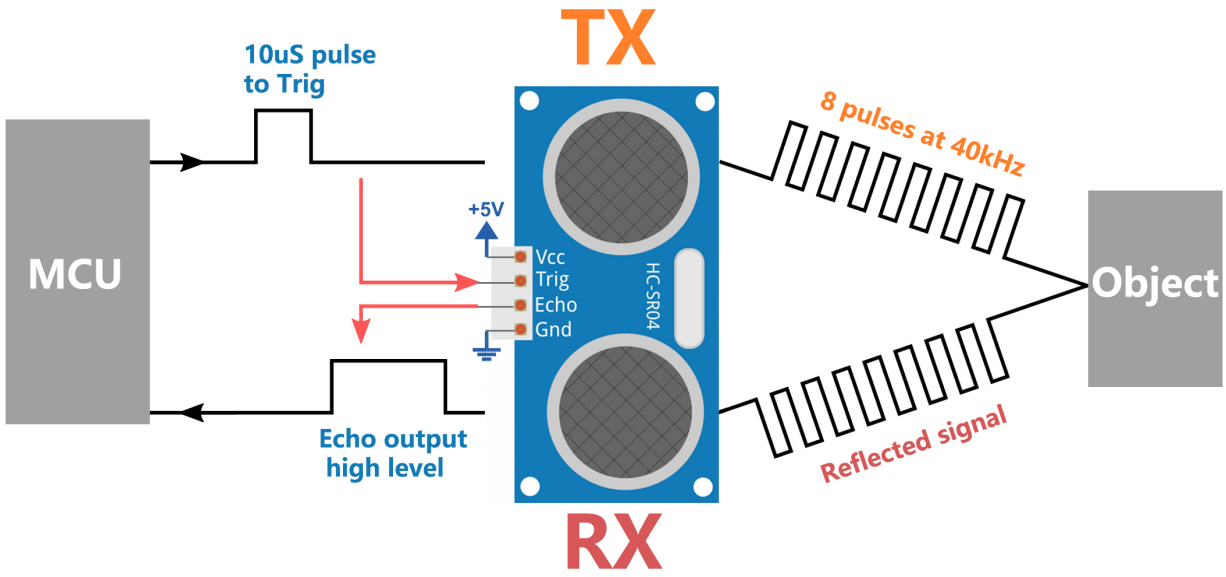
- Working Voltage: DC5V
- Working Current: 16mA

- Working Frequency: 40Hz
- Max Range: 500cm
- Min Range: 2cm
- Trigger Input Signal: 10uS TTL pulse
- Echo Output Signal: Input TTL lever signal and the range in proportion
- Connector: XH2.54-4P
- Dimension: 46x20.5x15 mm

Principle

The basic principles are as follows:

- Using IO trigger for at least 10us high level signal.
- The module sends an 8 cycle burst of ultrasound at 40 kHz and detects whether a pulse signal is received.
- Echo will output a high level if a signal is returned; the duration of the high level is the time from emission to return.
- Distance = (high level time x velocity of sound (340M/S)) / 2



Formula:

- us / 58 = centimeters distance
- us / 148 = inch distance
- distance = high level time x velocity (340M/S) / 2

Note: This module should not be connected under power up, if necessary, let the module's GND be connected first. Otherwise, it will affect the work of the module.

The area of the object to be measured should be at least 0.5 square meters and as flat as possible. Otherwise, it will affect results.

Example

- *6.1 Measuring Distance* (For MicroPython User)
- *7.10 Reversing Aid* (For MicroPython User)
- *6.1 - Measuring Distance* (For Arduino User)
- *2.11 Reversing System* (For Piper Make User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

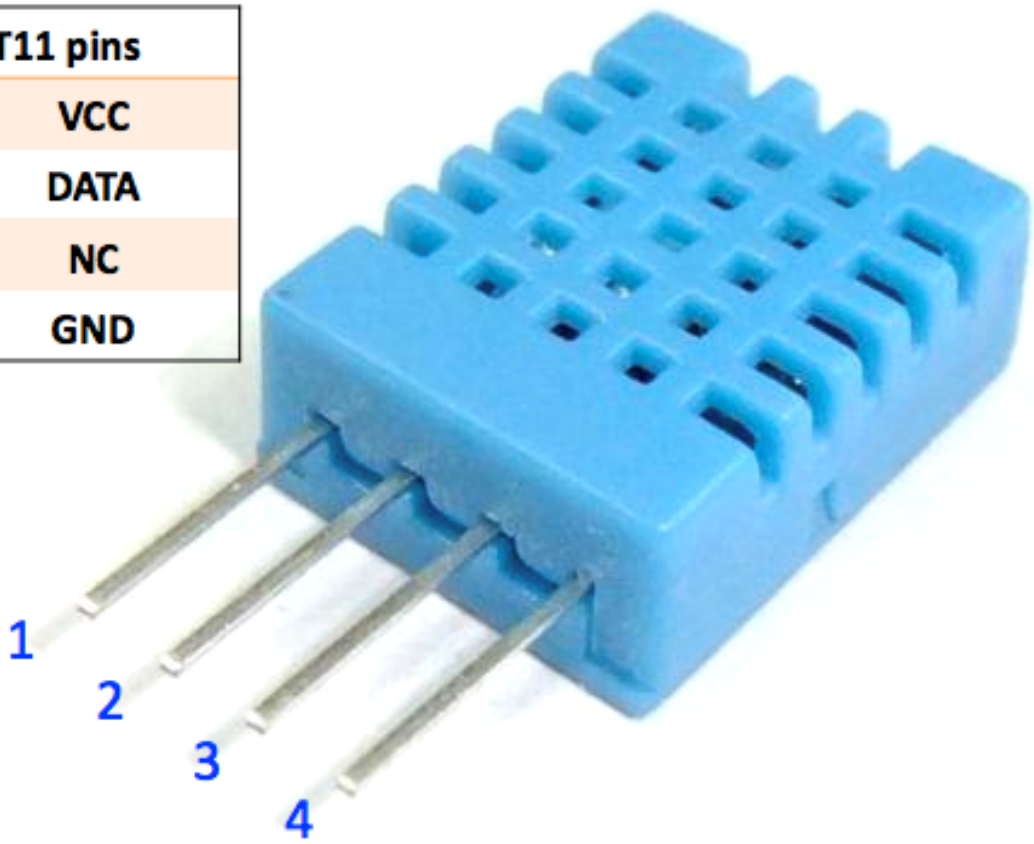
2.39 DHT11 Humiture Sensor

The digital temperature and humidity sensor DHT11 is a composite sensor that contains a calibrated digital signal output of temperature and humidity. The technology of a dedicated digital modules collection and the temperature and humidity sensing technology are applied to ensure that the product has high reliability and excellent long-term stability.

The sensor includes a resistive sense of wet component and an NTC temperature measurement device, and is connected with a high-performance 8-bit microcontroller.

Only three pins are available for use: VCC, GND, and DATA. The communication process begins with the DATA line sending start signals to DHT11, and DHT11 receives the signals and returns an answer signal. Then the host receives the answer signal and begins to receive 40-bit humiture data (8-bit humidity integer + 8-bit humidity decimal + 8-bit temperature integer + 8-bit temperature decimal + 8-bit checksum).

DHT11 pins	
1	VCC
2	DATA
3	NC
4	GND



Features

1. Humidity measurement range: 20 - 90%RH
2. Temperature measurement range: 0 - 60℃
3. Output digital signals indicating temperature and humidity
4. Working voltage:DC 5V; PCB size: 2.0 x 2.0 cm
5. Humidity measurement accuracy: ±5%RH
6. Temperature measurement accuracy: ±2℃

- DHT11 Datasheet

Example

- *6.2 Temperature - Humidity* (For MicroPython User)
- *6.2 - Temperature - Humidity* (For Arduino User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

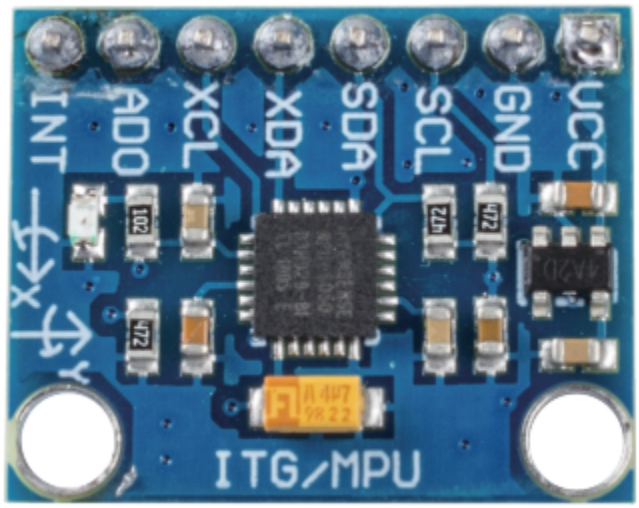
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

2.40 MPU6050 Module

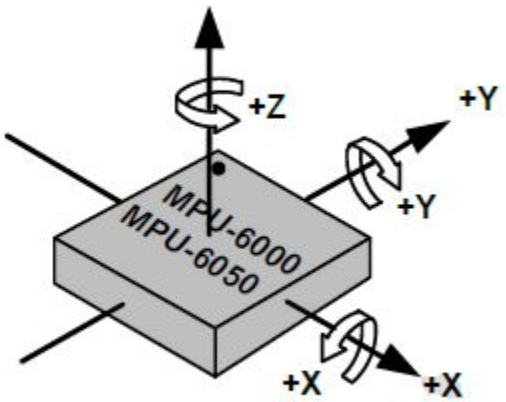
MPU6050



The MPU-6050 is a 6-axis(combines 3-axis Gyroscope, 3-axis Accelerometer) motion tracking devices.

Its three coordinate systems are defined as follows:

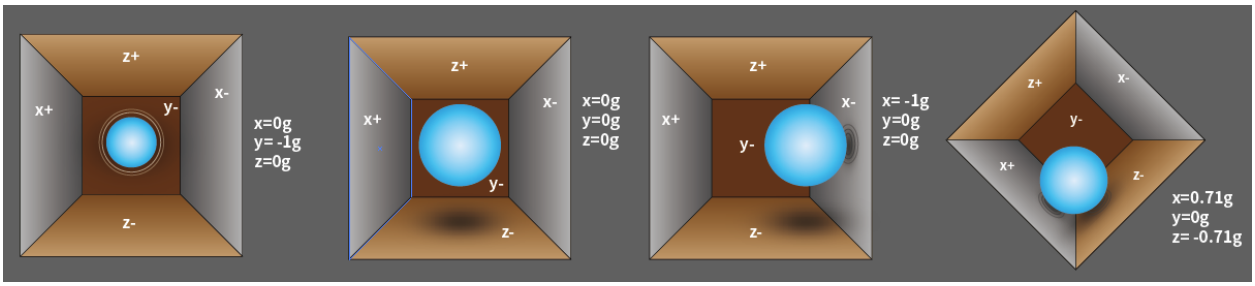
Put MPU6050 flat on the table, assure that the face with label is upward and a dot on this surface is on the top left corner. Then the upright direction upward is the z-axis of the chip. The direction from left to right is regarded as the X-axis. Accordingly the direction from back to front is defined as the Y-axis.



3-axis Accelerometer

The accelerometer works on the principle of piezo electric effect, the ability of certain materials to generate an electric charge in response to applied mechanical stress.

Here, imagine a cuboidal box, having a small ball inside it, like in the picture above. The walls of this box are made with piezo electric crystals. Whenever you tilt the box, the ball is forced to move in the direction of the inclination, due to gravity. The wall with which the ball collides, creates tiny piezo electric currents. There are totally, three pairs of opposite walls in a cuboid. Each pair corresponds to an axis in 3D space: X, Y and Z axes. Depending on the current produced from the piezo electric walls, we can determine the direction of inclination and its magnitude.



We can use the MPU6050 to detect its acceleration on each coordinate axis (in the stationary desktop state, the Z-axis acceleration is 1 gravity unit, and the X and Y axes are 0). If it is tilted or in a weightless/overweight condition, the corresponding reading will change.

There are four kinds of measuring ranges that can be selected programmatically: +/-2g, +/-4g, +/-8g, and +/-16g (2g by default) corresponding to each precision. Values range from -32768 to 32767.

The reading of accelerometer is converted to an acceleration value by mapping the reading from the reading range to the measuring range.

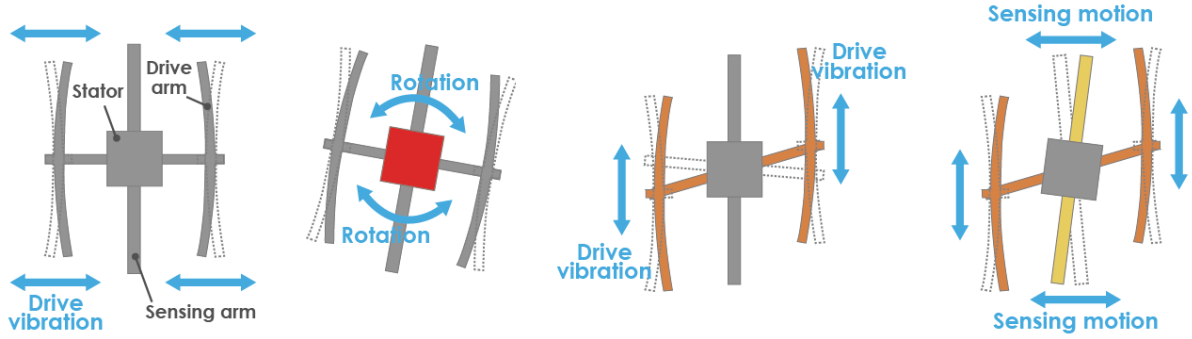
Acceleration = (Accelerometer axis raw data / 65536 * full scale Acceleration range) g

Take the X-axis as an example, when Accelerometer X axis raw data is 16384 and the range is selected as +/-2g:

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

3-axis Gyroscope

Gyroscopes work on the principle of Coriolis acceleration. Imagine that there is a fork like structure, that is in constant back and forth motion. It is held in place using piezo electric crystals. Whenever, you try to tilt this arrangement, the crystals experience a force in the direction of inclination. This is caused as a result of the inertia of the moving fork. The crystals thus produce a current in consensus with the piezo electric effect, and this current is amplified.



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.

The Gyroscope also has four kinds of measuring ranges: +/- 250, +/- 500, +/- 1000, +/- 2000. The calculation method and Acceleration are basically consistent.

The formula for converting the reading into angular velocity is as follows:

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

The X axis, for example, the Accelerometer X axis raw data is 16384 and ranges + / - 250°/ s:

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

Example

- *6.3 6-axis Motion Tracking* (For MicroPython User)
- *7.11 Somatosensory Controller* (For MicroPython User)
- *7.12 Digital Bubble Level* (For MicroPython User)
- *6.3 - 6-axis Motion Tracking* (For Arduino User)

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

CHAPTER

THREE

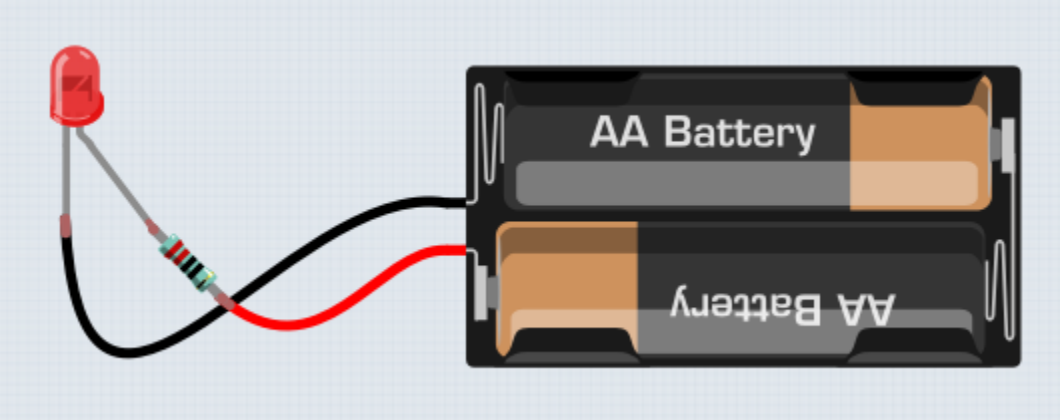
ELECTRONIC CIRCUIT

There are many things you use every day that are powered by electricity, such as the lights in your home and the computer you are reading this on.

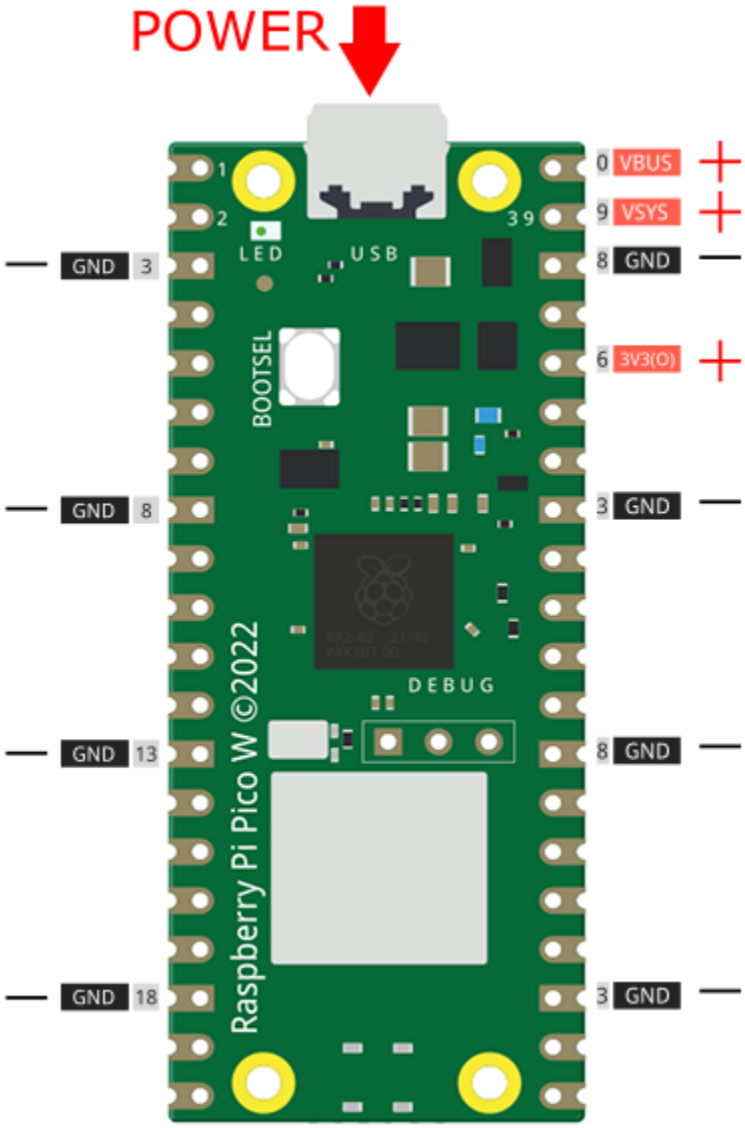
In order to use electricity, you must create an electrical circuit. An electric circuit consists of metal wires and electrical and electronic components.

Circuits require power from somewhere. In your home, most appliances (e.g., TVs, lights) are powered by wall outlets. But many smaller, portable circuits (e.g., electronic toys, cell phones) are powered by batteries. A battery has two terminals, one of which is called the positive terminal and is marked with a plus sign (+). Negative terminals are symbolized by minus signs (-), but are not usually printed on batteries.

For current to flow, a conductive path must connect the positive terminal of the battery to the negative terminal, which is referred to as a closed circuit(If it is disconnected, it is called an open circuit.). Electric current will flow through appliances such as lamps to make them work (e.g., light up).

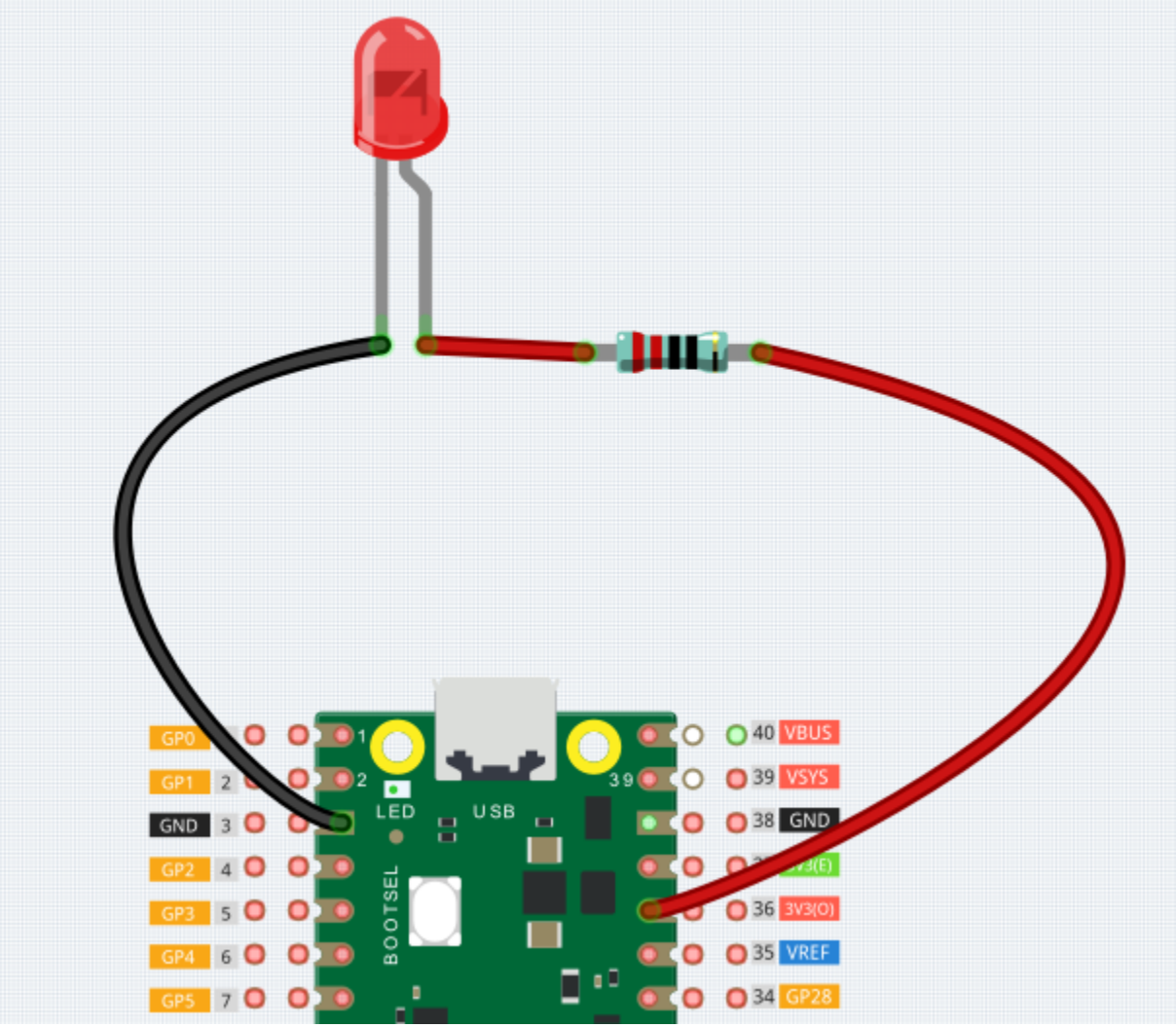


A Pico W has some power output pins (positive) and some ground pins (negative). You can use these pins as the positive and negative sides of the power supply by plugging the Pico W into a power source.



With electricity, you can create works with light, sound, and motion. You can light up an LED by connecting the long pin to the positive terminal and the short pin to the negative terminal. The LED will break down very quickly if you do this, so you need to add a 220* resistor inside the circuit to protect it.

The circuit they form is shown below.



You may have questions this time: how do I build this circuit? Hold the wires by hand, or tape the pins and wires?

In this situation, solderless breadboards will be your strongest allies.

3.1 Hello, Breadboard!

A breadboard is a rectangular plastic plate with a bunch of small holes. These holes allow us to easily insert electronic components and build electronic circuits. Breadboards do not permanently fix electronic components, so we can easily repair a circuit and start over if something goes wrong.

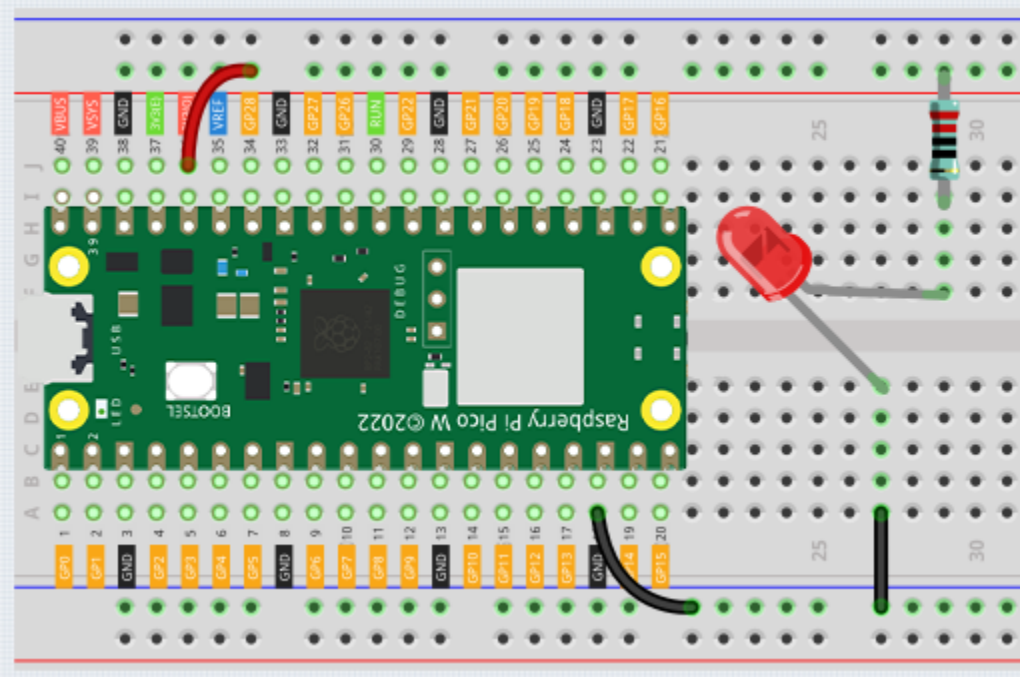
Note: There is no need for special tools to use breadboards. However, many electronic components are very small, and a pair of tweezers can help us to pick up small parts better.

On the Internet, we can find a lot of information about breadboards.

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

Here are some things you should know about breadboards.

1. Each half-row group (such as column A-E in row 1 or column F-J in row 3) is connected. Therefore, if an electrical signal flows in from A1, it can flow out from B1, C1, D1, E1, but not from F1 or A2.
2. In most cases, both sides of the breadboard are used as power buses, and the holes in each column (about 50 holes) are connected together. As a general rule, positive power supplies are connected to the holes near the red wire, and negative power supplies are connected to the holes near the blue wire.
3. In a circuit, current flows from the positive pole to the negative pole after passing through the load. In this case, a short circuit may occur.



Let us follow the direction of the current to build the circuit!

1. In this circuit, we use the 3V3 pin of the Pico W board to power the LED. Use a male-to-male (M2M) jumper wire to connect it to the red power bus.
2. To protect the LED, the current must pass through a 220 ohm resistor. Connect one end (either end) of the resistor to the red power bus, and the other end to the free row of the breadboard (row 24 in my circuit).

> **Note:** The color ring of the 220 ohm resistor is red, red, black, black and brown.

3. If you pick up the LED, you will see that one of its leads is longer than the other. Connect the longer lead to the same row as the resistor, and the shorter lead to the same row across the middle gap on the breadboard.

> **Note:** The longer lead is the anode, which represents the positive side of the circuit; the shorter lead is the cathode, which represents the negative side.
>
> The anode needs to be connected to the GPIO pin through a resistor; the cathode needs to be connected to the GND pin.

4. Using a male-to-male (M2M) jumper wire, connect the LED short pin to the breadboard's negative power bus.
5. Connect the GND pin of Pico W to the negative power bus using a jumper.

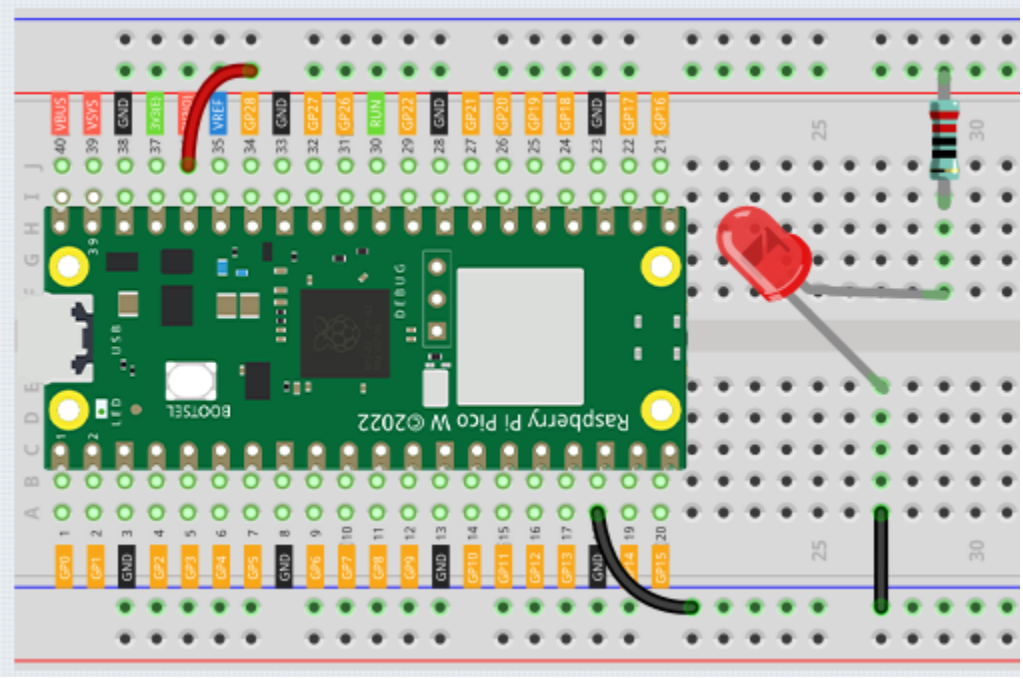
3.2 Beware of short circuits

Short circuits can occur when two components that shouldn't be connected are "accidentally" connected. This kit includes resistors, transistors, capacitors, LEDs, etc. that have long metal pins that can bump into each other and cause a short. Some circuits are simply prevented from functioning properly when a short occurs. Occasionally, a short circuit can damage components permanently, especially between the power supply and the ground bus, causing the circuit to get very hot, melting the plastic on the breadboard and even burning the components!

Therefore, always make sure that the pins of all the electronics on the breadboard are not touching each other.

3.3 Direction of the circuit

There is an orientation to circuits, and the orientation plays a significant role in certain electronic components. There are some devices with polarity, which means they must be connected correctly based on their positive and negative poles. Circuits built with the wrong orientation will not function properly.



If you reverse the LED in this simple circuit that we built earlier, you will find that it no longer works.

In contrast, some devices have no direction, such as the resistors in this circuit, so you can try inverting them without affecting the LEDs' normal operation.

Most components and modules with labels such as "+", "-", "GND", "VCC" or have pins of different lengths must be connected to the circuit in a specific way.

3.4 Protection of the circuit

Current is the rate at which electrons flow past a point in a complete electrical circuit. At its most basic, current = flow. An ampere (AM-pir), or amp, is the international unit used for measuring current. It expresses the quantity of electrons (sometimes called "electrical charge") flowing past a point in a circuit over a given time.

The driving force (voltage) behind the flow of current is called voltage and is measured in volts (V).

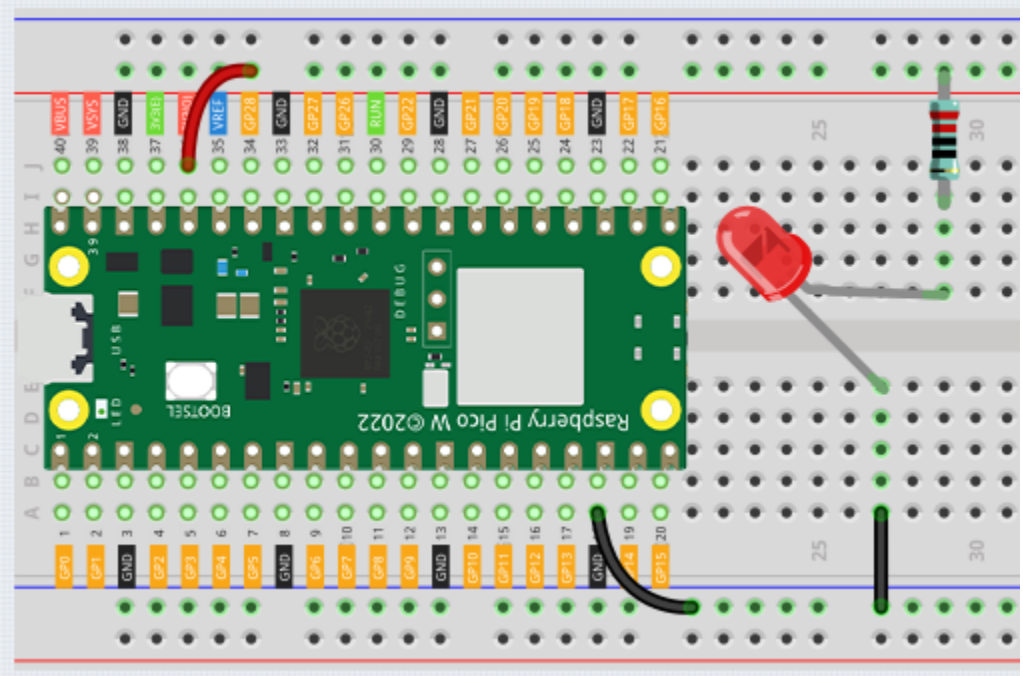
Resistance (R) is the property of the material that restricts the flow of current, and it is measured in ohms ().

According to Ohm's law (as long as the temperature remains constant), current, voltage, and resistance are proportional. A circuit's current is proportional to its voltage and inversely proportional to its resistance.

Therefore, current (I) = voltage (V) / resistance (R).

- Ohm's law - Wikipedia

About Ohm's law we can do a simple experiment.



By changing the wire connecting 3V3 to 5V (i.e. VBUS, the 40th pin of Pico W), the LED will become brighter. If you change the resistor from 220ohm to 1000ohm (color ring: brown, black, black, brown and brown), you will notice that the LED becomes dimmer than before. The larger the resistor, the dimmer the LED.

Note: For an introduction to resistors and how to calculate resistance values, see *Resistor*.

Most packaged modules only require access to the proper voltage (usually 3.3V or 5V), such as ultrasonic module.

However, in your self-built circuits, you need to be aware of the supply voltage and resistor usage for electrical devices.

As an example, LEDs usually consume 20mA of current, and their voltage drop is about 1.8V. According to Ohm's law, if we use 5V power supply, we need to connect a minimum of 160ohm ((5-1.8)/20mA) resistor in order not to burn out the LED.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook!

Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

CHAPTER

FOUR

FOR MICROPYTHON USER

In this section, you will learn the history of MicroPython, how to install MicroPython in Pico W, the basic syntax, and a dozen interesting and practical projects to help you learn MicroPython quickly.

We recommend that you read the chapters in order.

1. Get Started

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.1 1.1 Introduction of MicroPython

MicroPython is a software implementation of a programming language largely compatible with Python 3, written in C, that is optimized to run on a microcontroller.

MicroPython consists of a Python compiler to bytecode and a runtime interpreter of that bytecode. The user is presented with an interactive prompt (the REPL) to execute supported commands immediately. Included are a selection of core Python libraries; MicroPython includes modules which give the programmer access to low-level hardware.

- Reference: MicroPython - Wikipedia

4.1.1 The Story Starts Here

Things changed in 2013 when Damien George launched a crowdfunding campaign (Kickstarter).

Damien was an undergraduate student at Cambridge University and an avid robotics programmer. He wanted to reduce the world of Python from a gigabyte machine to a kilobyte. His Kickstarter campaign was to support his development while he turned his proof of concept into a finished implementation.

MicroPython is supported by a diverse Pythonista community that has a keen interest in seeing the project succeed.

Apart from testing and supporting the code base, the developers provided tutorials, code libraries, and hardware porting, so Damien was able to focus on other aspects of the project.

- Reference: realpython

4.1.2 Why MicroPython

Although the original Kickstarter campaign released MicroPython as a development board "pyboard" with STM32F4, MicroPython supports many ARM-based product architectures. The mainline supported ports are ARM Cortex-M (many STM32 boards, TI CC3200/WiPy, Teensy boards, Nordic nRF series, SAMD21 and SAMD51), ESP8266, ESP32, 16bit PIC, Unix, Windows, Zephyr and JavaScript. Second, MicroPython allows for fast feedback. This is because you can use REPL to enter commands interactively and get responses. You can even tweak code and run it immediately instead of traversing the code-compile-upload-execute cycle.

While Python has the same advantages, for some Microcontroller boards like the Raspberry Pi Pico, they are small, simple and have little memory to run the Python language at all. That's why MicroPython has evolved, keeping the main Python features and adding a bunch of new ones to work with these Microcontroller boards.

Next you will learn to install MicroPython into the Raspberry Pi Pico.

- Reference: MicroPython - Wikipedia
- Reference: realpython

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

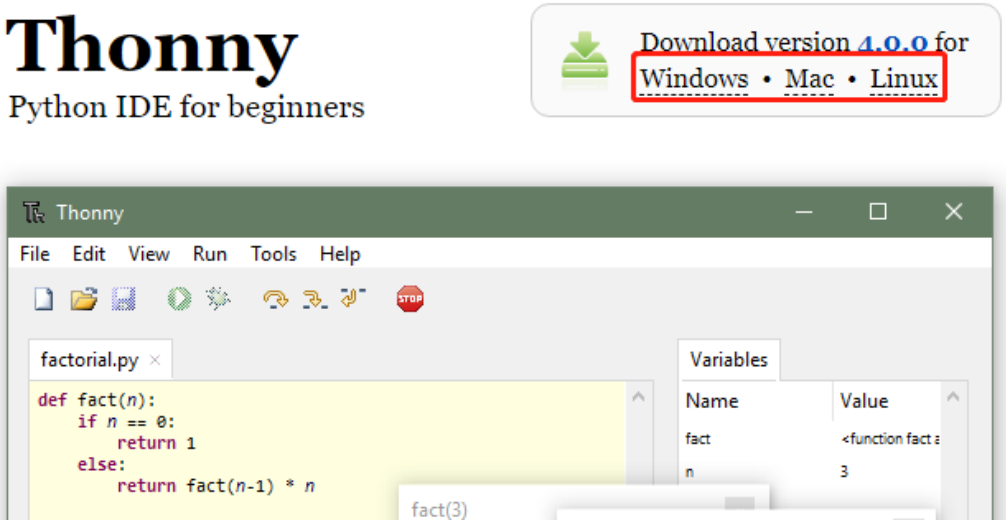
4.2 1.2 Install Thonny IDE

Before you can start to program Pico with MicroPython, you need an integrated development environment (IDE), here we recommend Thonny. Thonny comes with Python 3.7 built in, just one simple installer is needed and you're ready to learn programming.

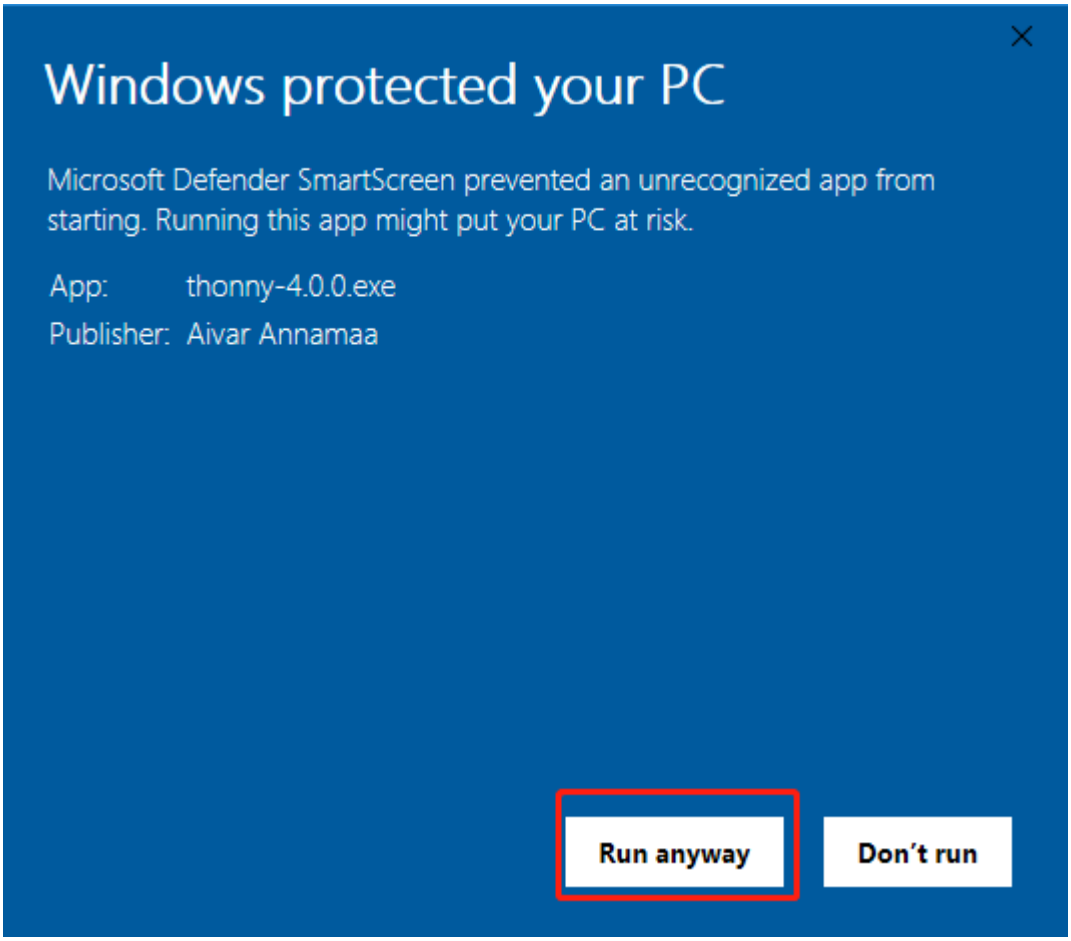
Note: Since the Raspberry Pi Pico interpreter only works with Thonny version 3.3.3 or later, you can skip this chapter if you have it; otherwise, please update or install it.

---

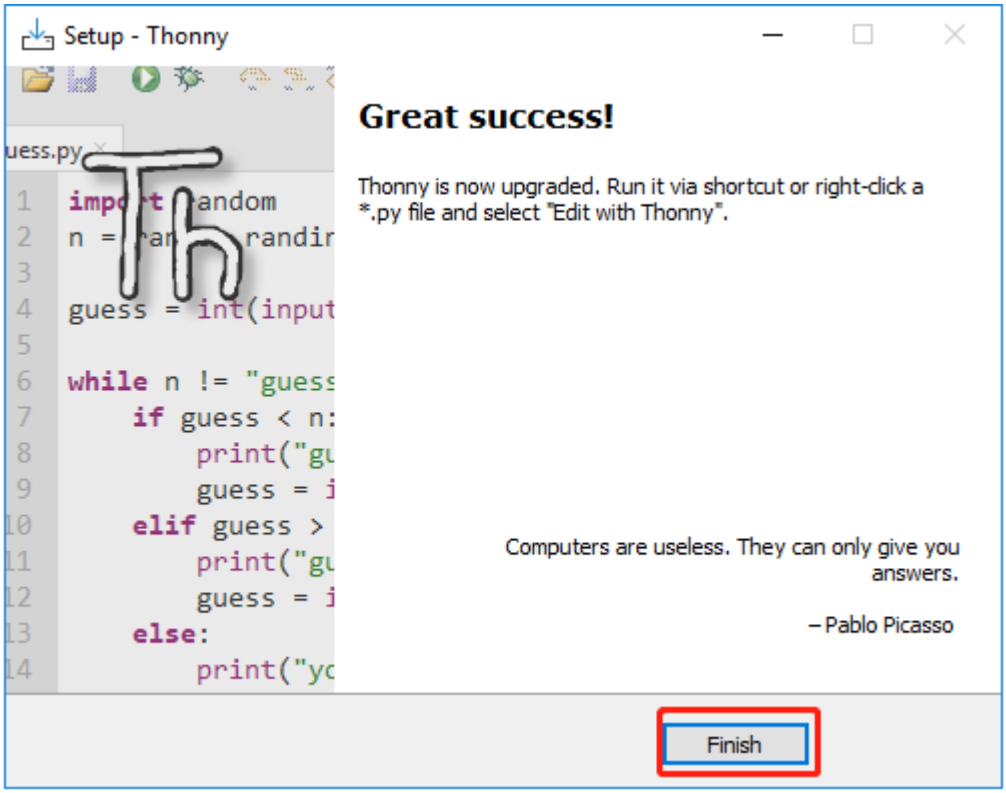
1. You can download it by visiting the website. Once open the page, you will see a light gray box in the upper right corner, click on the link that applies to your operating system.



2. The installers have been signed with a new certificate which hasn't built up its reputation yet. You may need to click through your browser warning (e.g. choose "Keep" instead of "Discard" in Chrome) and Windows Defender warning (**More info Run anyway**).



3. Next, click **Next** and **Install** to finish installing Thonny.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

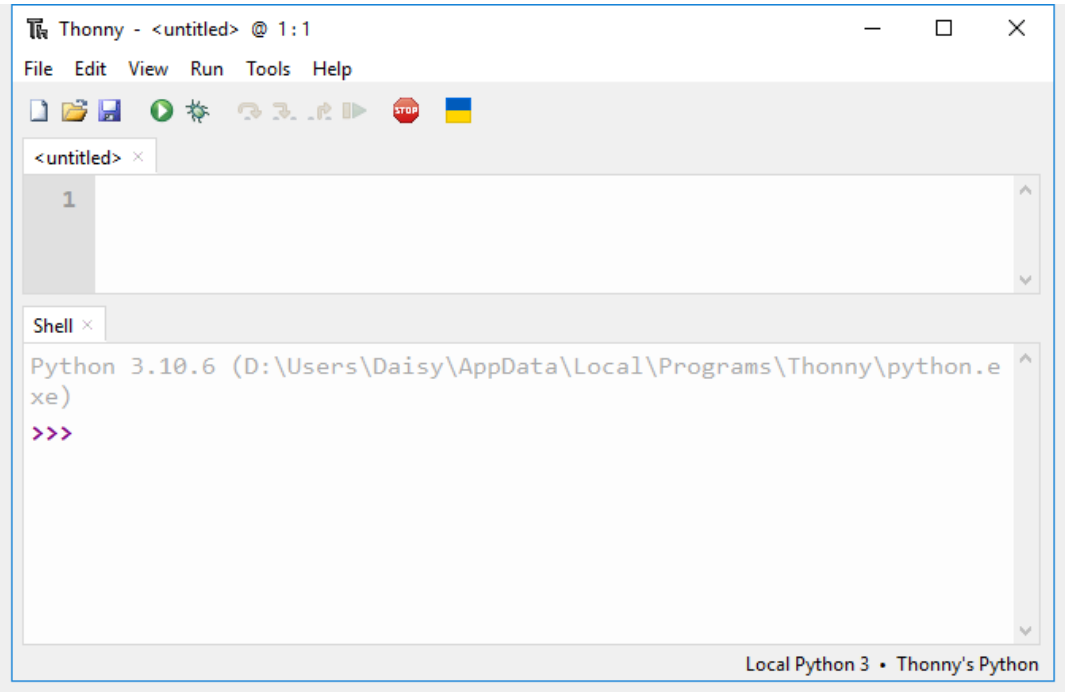
Ready to explore and create with us? Click [] and join today!

4.3 1.3 Install MicroPython on Your Pico

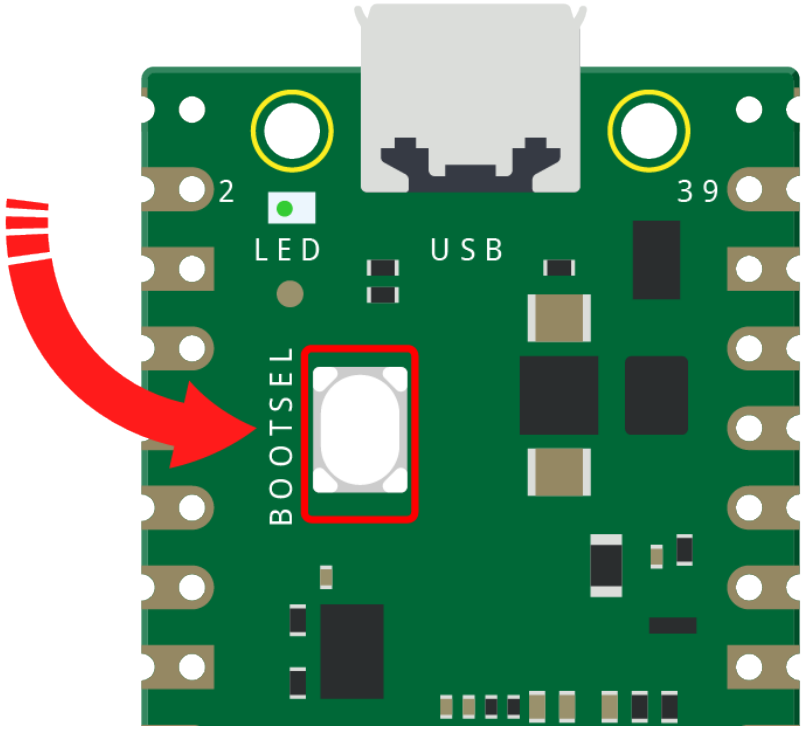
Now come to install MicroPython into Raspberry Pi Pico, Thonny IDE provides a very convenient way for you to install it with one click.

Note: If you do not wish to upgrade Thonny, you can use the Raspberry Pi official by dragging and dropping an `rp2_pico_xxxx.uf2` file into Raspberry Pi Pico.

1. Open Thonny IDE.

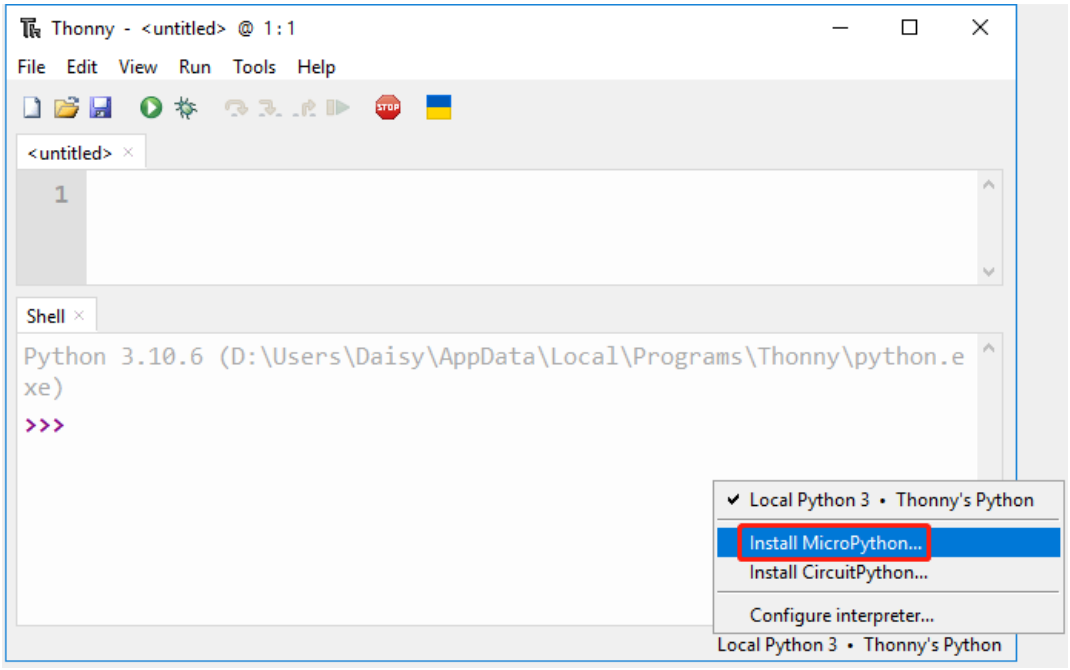


2. Press and hold the **BOOTSEL** button and then connect the Pico to computer via a Micro USB cable. Release the **BOOTSEL** button after your Pico is mount as a Mass Storage Device called **RPI-RP2**.

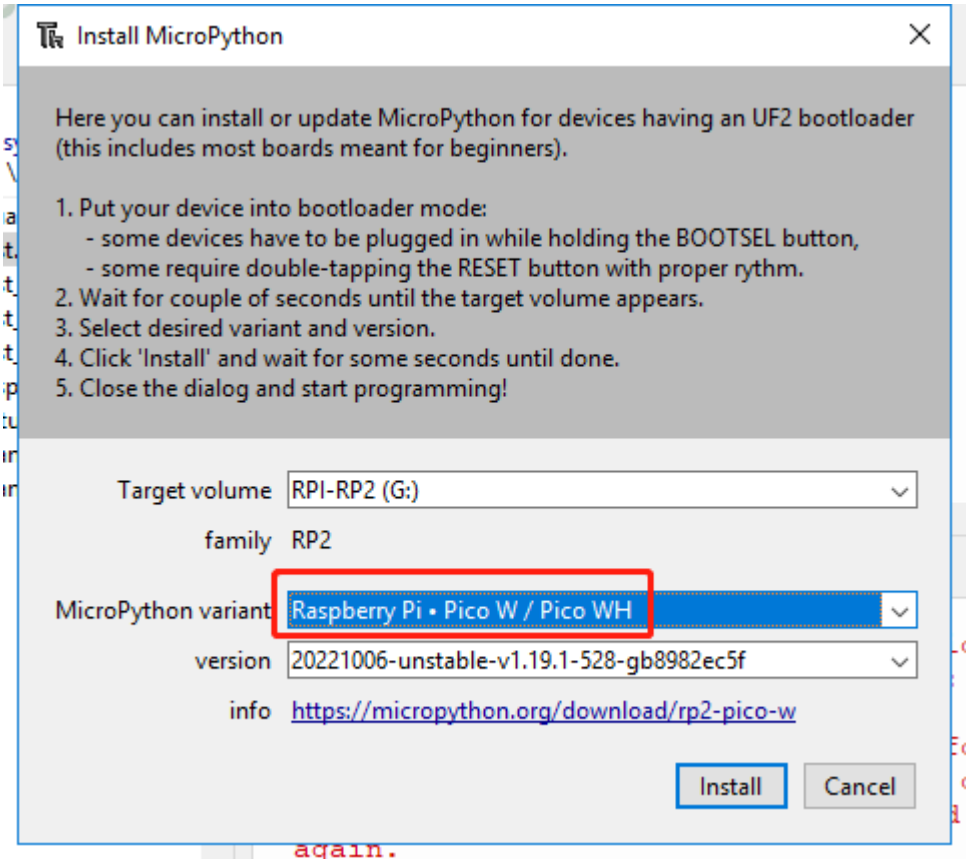


3. In the bottom right corner, click the interpreter selection button and select **Install Micropython**.

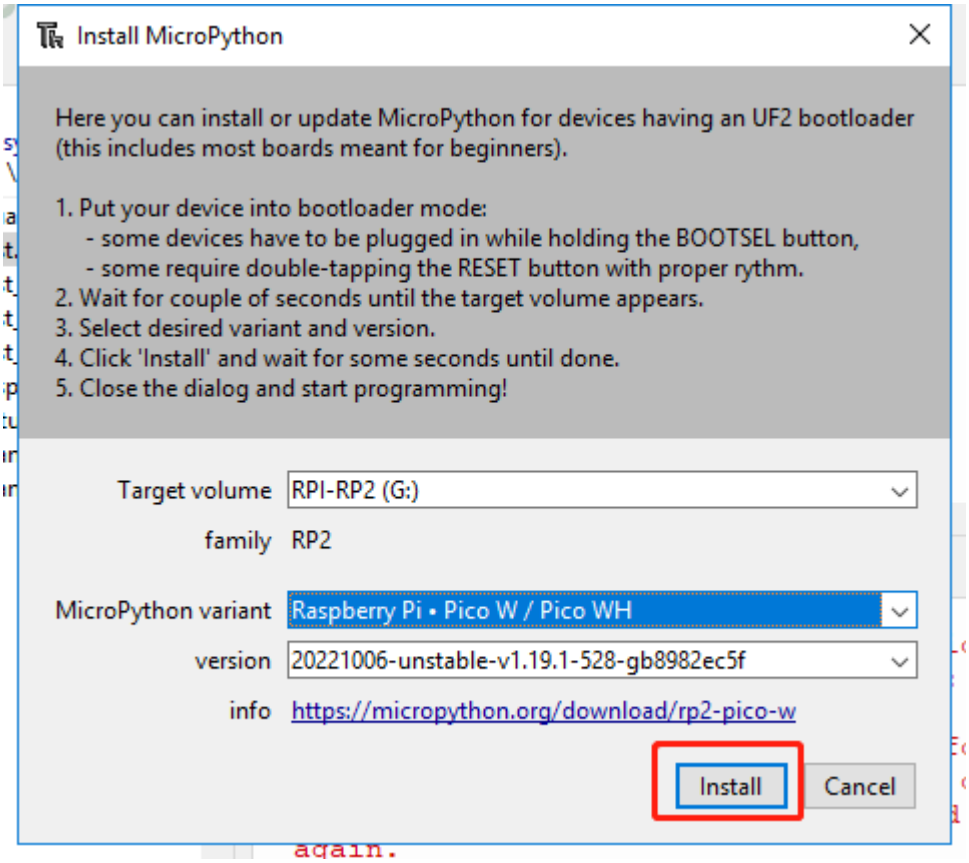
> **Note:** If your Thonny does not have this option, please update to the latest version.



4. In the **Target volume**, the volume of the Pico you just plugged in will automatically appear, and in the **Micropython variant**, select **Raspberry Pi.Pico/Pico H**.



5. Click the **Install** button, wait for the installation to complete and then close this page.



Congratulations, now your Raspberry Pi Pico is ready to go.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

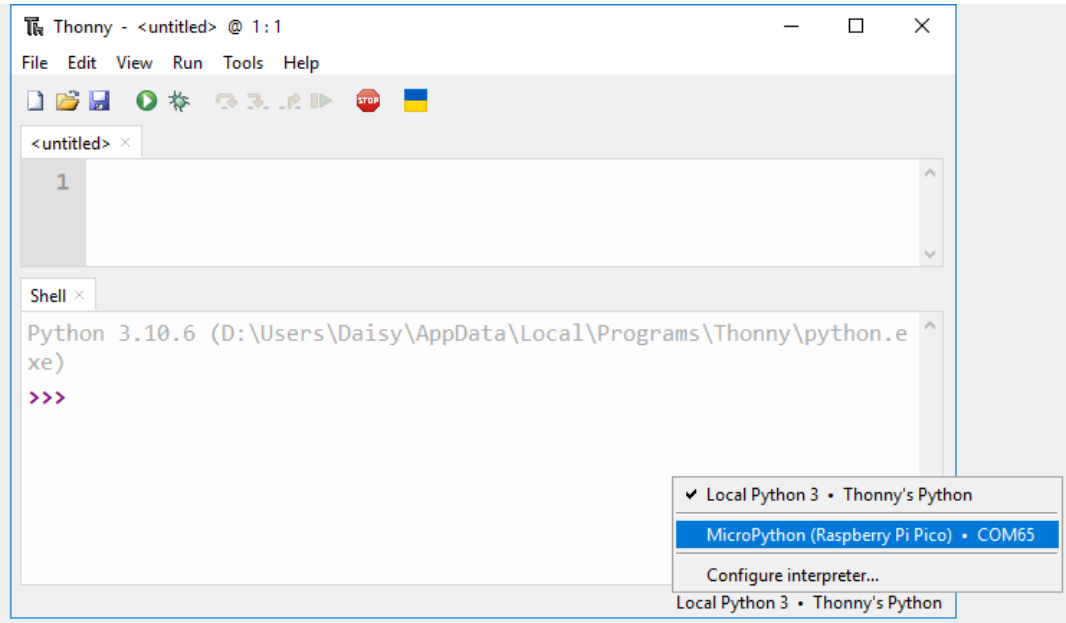
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

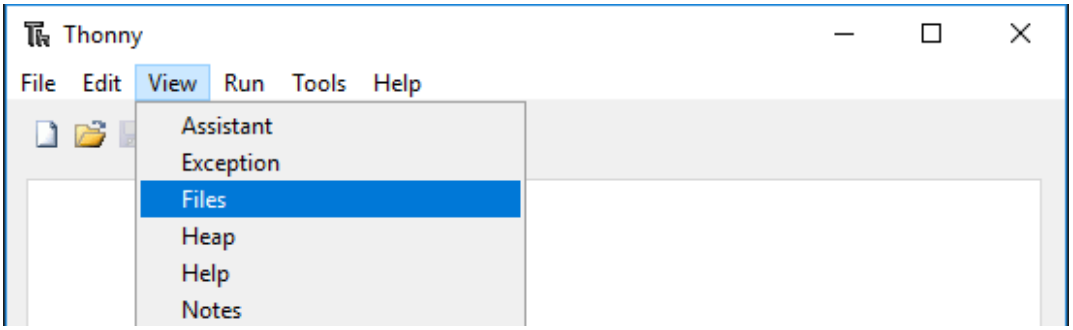
4.4 1.4 Upload the Libraries to Pico

In some projects, you will need additional libraries. So here we upload these libraries to Raspberry Pi Pico W first, and then we can run the code directly later.

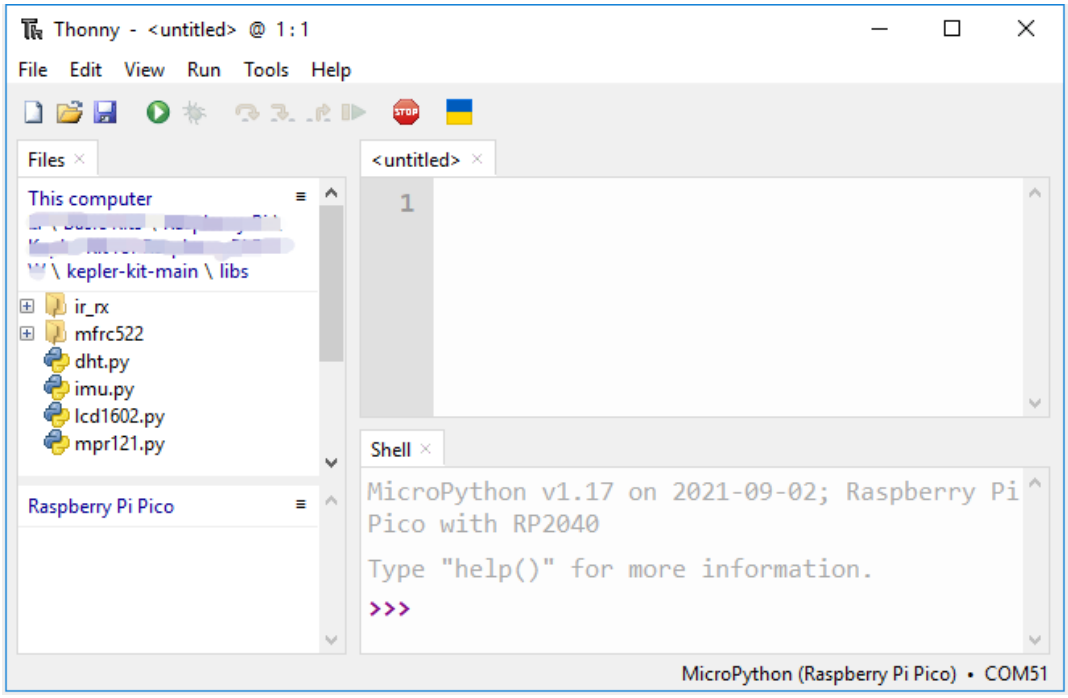
1. Download the relevant code from the link below.
    - `SunFounder Kepler Kit`
2. Open Thonny IDE and plug the Pico into your computer with a micro USB cable and click on the "MicroPython (Raspberry Pi Pico).COMXX" interpreter in the bottom right corner.



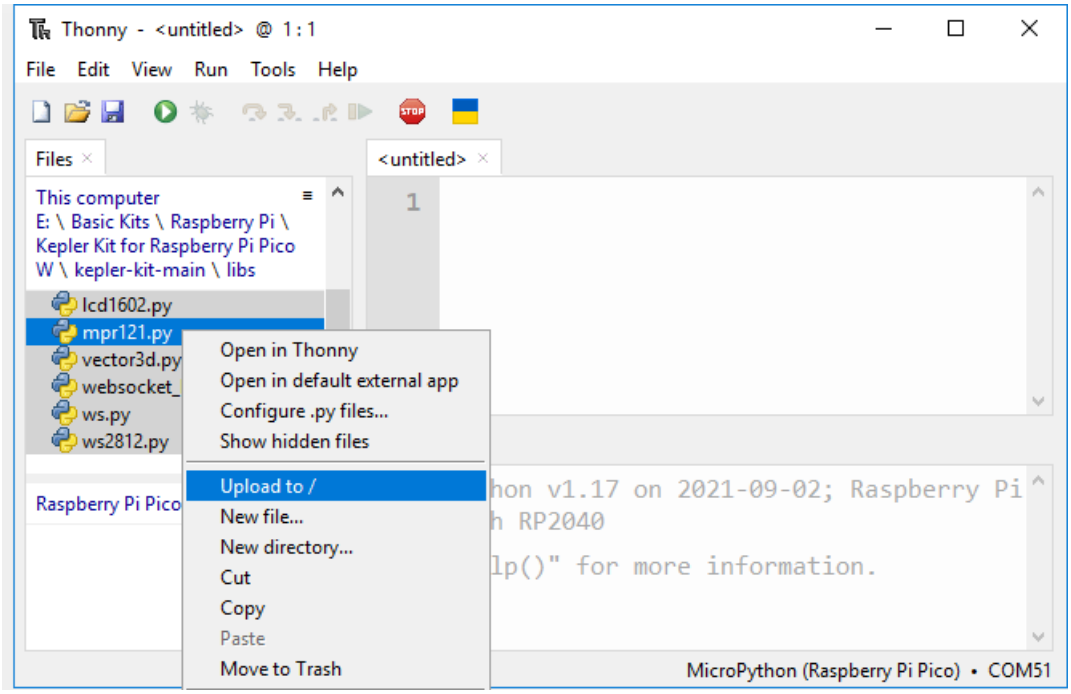
3. In the top navigation bar, click **View** -> **Files**.



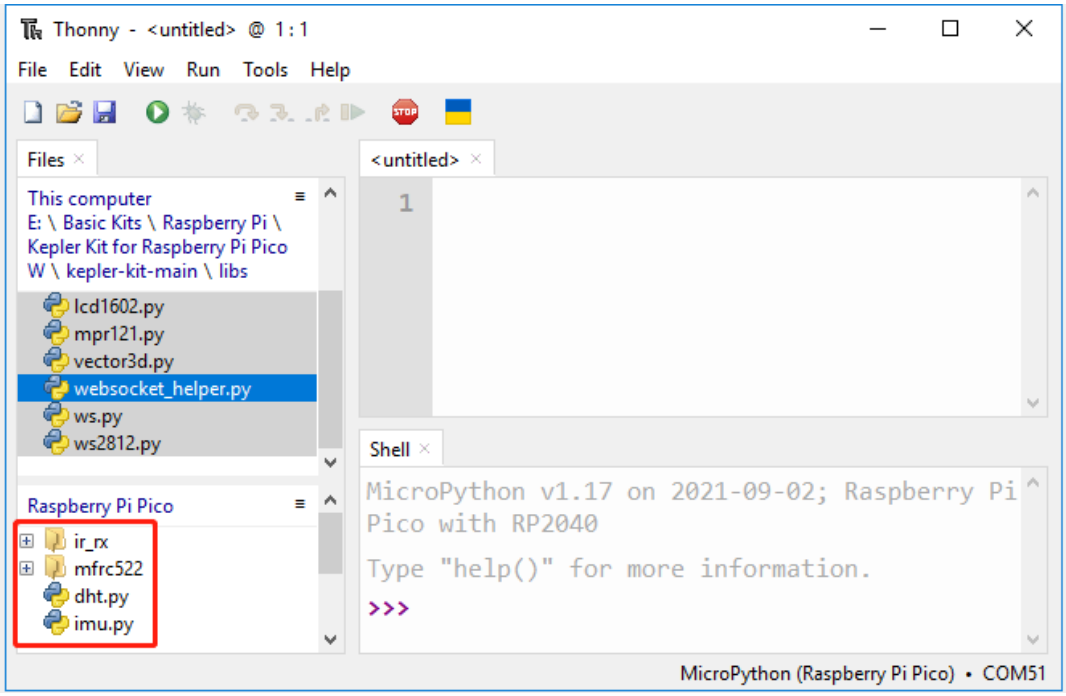
4. Switch the path to the folder where you downloaded the code package before, and then go to the `kepler-kit-main/libs` folder.



5. Select all the files or folders in the `libs/` folder, right-click and click **Upload to**, it will take a while to upload.



6. Now you will see the files you just uploaded inside your drive `Raspberry Pi Pico`.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.5 1.5 Quick Guide on Thonny

4.5.1 Open and Run Code Directly

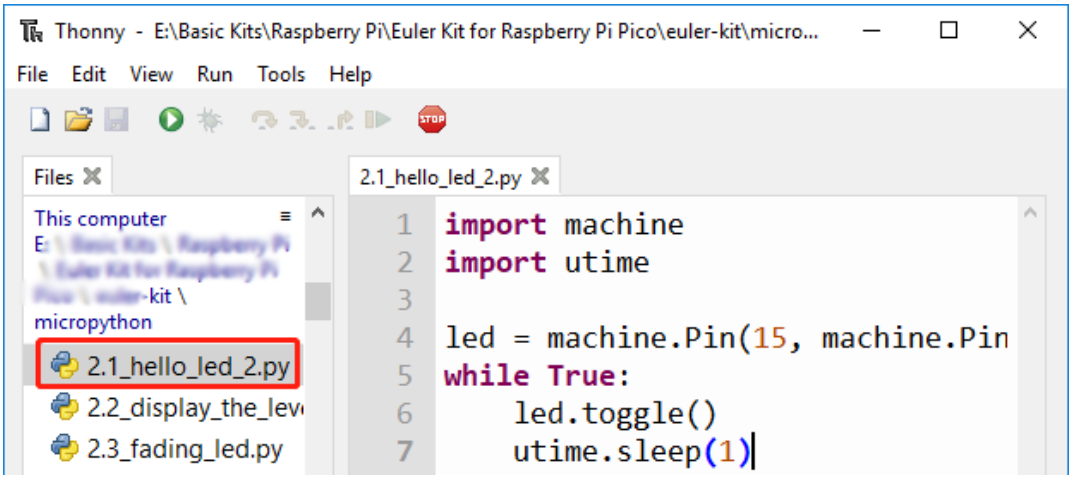
The code section in the projects tells you exactly which code is used, so double-click on the `.py` file with the serial number in the `kepler-kit-main/micropython/` path to open it.

However, you must first download the package and upload the library, as described in *1.4 Upload the Libraries to Pico*.

1. Open code.

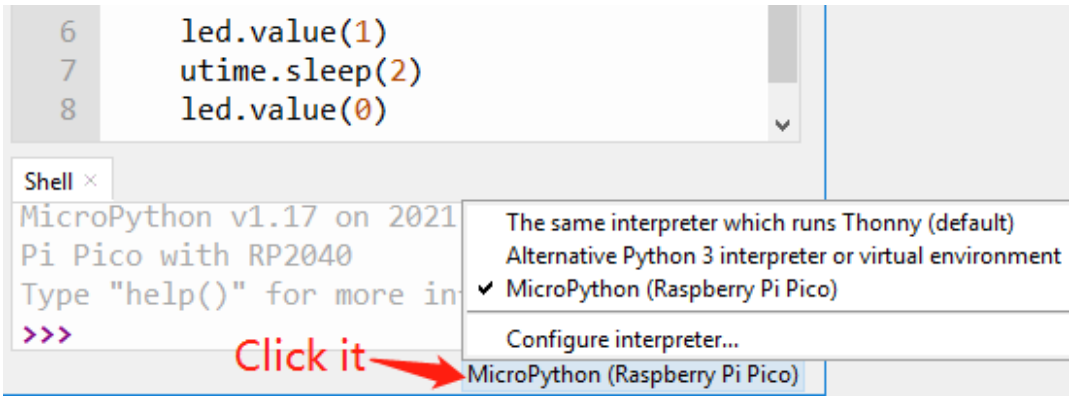
   For example, `2.1_hello_led.py`.

   If you double click on it, a new window will open on the right. You can open more than one code at the same time.



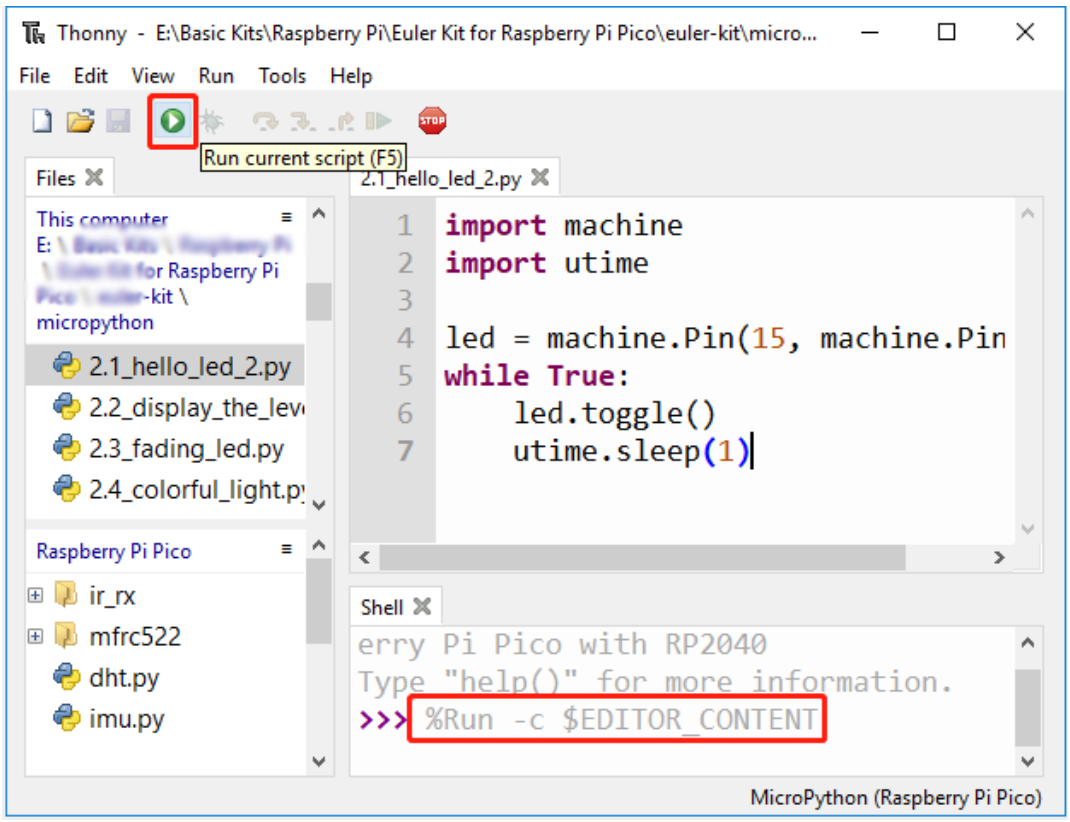
2. Select correct interpreter

   Use a micro USB cable to connect the Pico W to your computer and select the "MicroPython (Raspberry Pi Pico)" interpreter.



3. Run the code

   To run the script, click the **Run current script** button or press F5.



If the code contains any information that needs to be printed, it will appear in the Shell; otherwise, only the following information will appear.

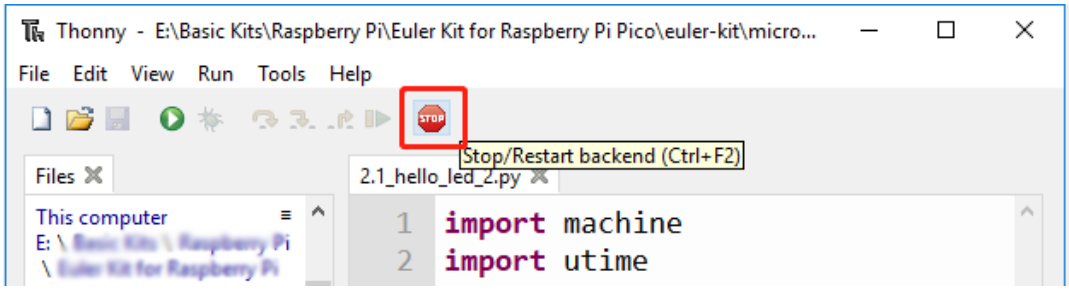
Click **View** -> **Edit** to open the Shell window if it doesn't appear on your Thonny.

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

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

- The first line shows the version of MicroPython, the date, and your device information.
- The second line prompts you to enter "help()" to get some help.
- The third line is a command from Thonny telling the MicroPython interpreter on your Pico W to run the contents of the script area - "EDITOR_CONTENT".
- If there is any message after the third line, it is usually a message that you tell MicroPython to print, or an error message for the code.

4. Stop running

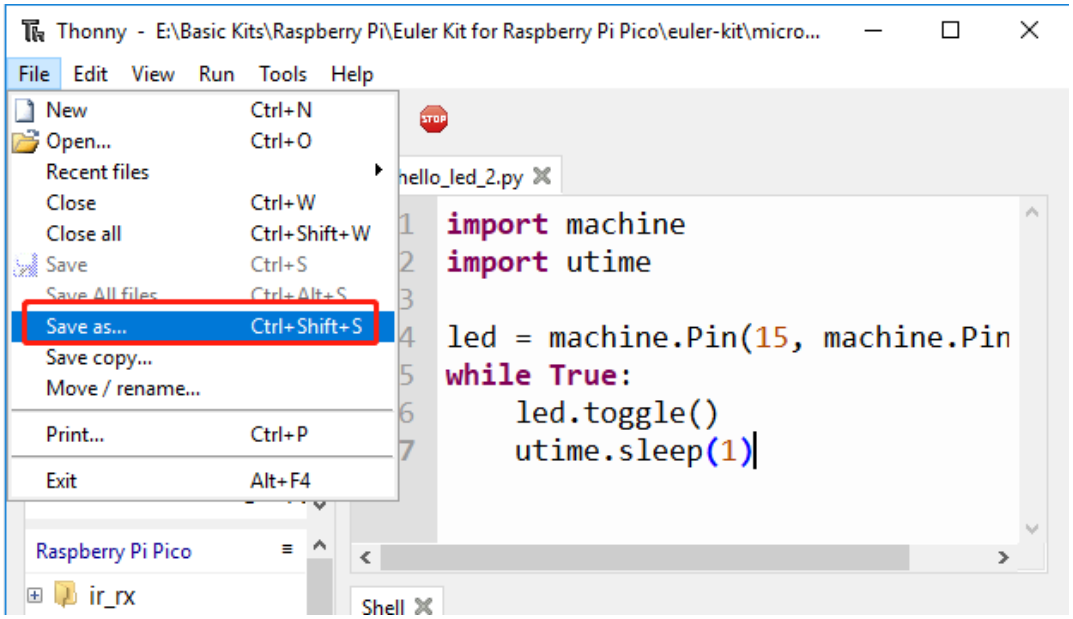


To stop the running code, click the **Stop/Restart backend** button. The **%RUN -c $EDITOR_CONTENT** command will disappear after stopping.

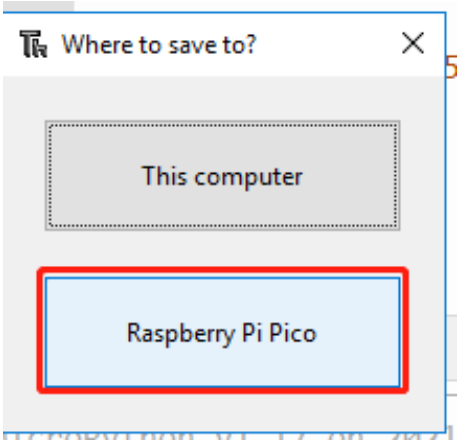
5. Save or save as

   You can save changes made to the open example by pressing **Ctrl+S** or clicking the **Save** button on Thonny.

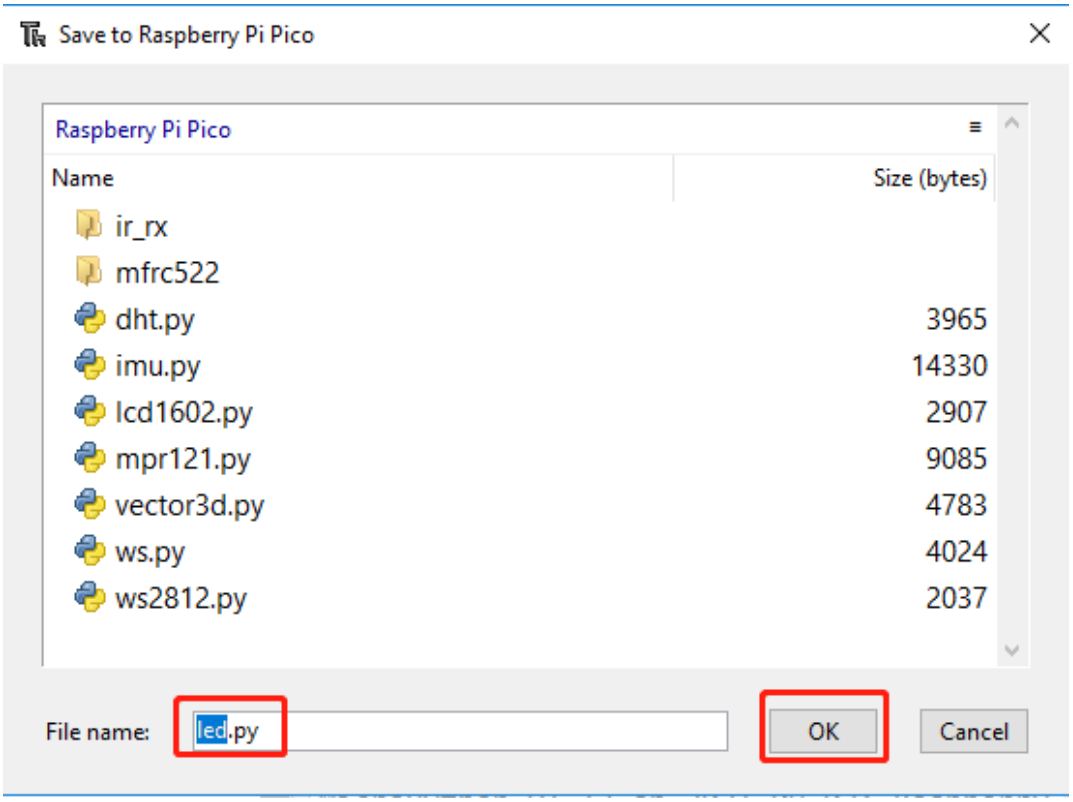
   The code can be saved as a separate file within the Raspberry Pi Pico W by clicking on **File** -> **Save As**.



Select **Raspberry Pi Pico**.



Then click **OK** after entering the file name and extension **.py**. On the Raspberry Pi Pico W drive, you will see your saved file.



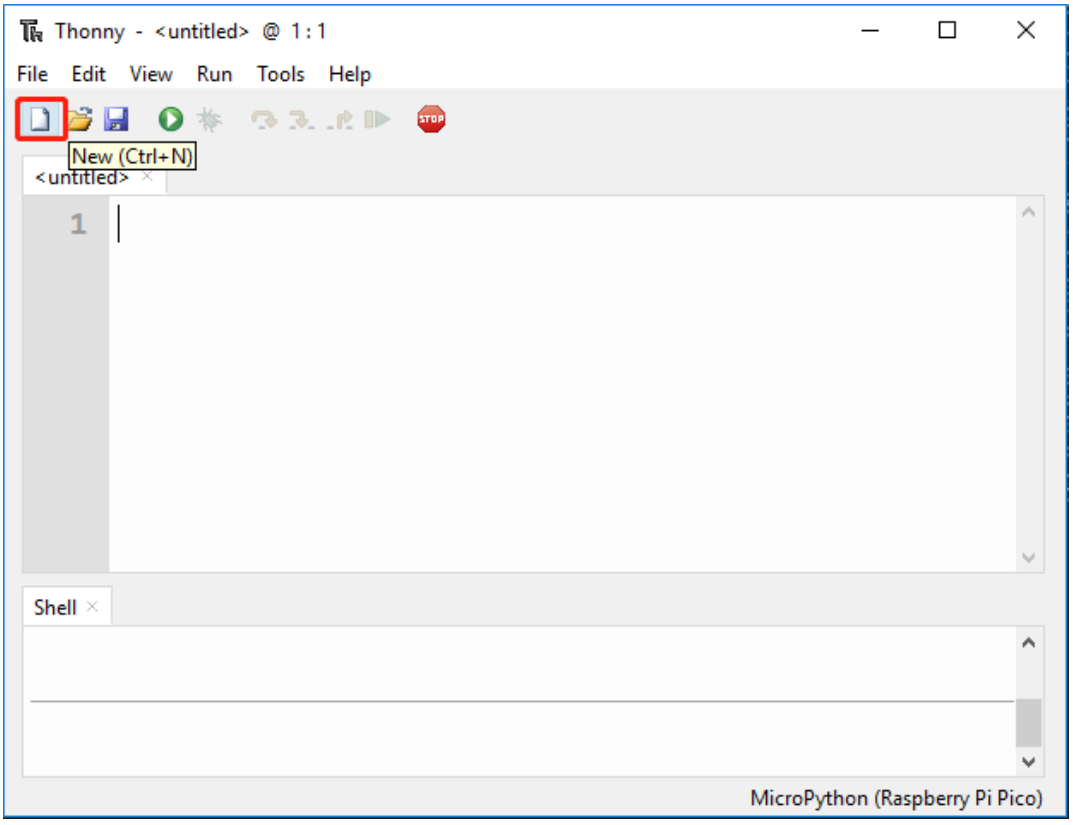
Note: Regardless of what name you give your code, it's best to describe what type of code it is, and not give it a meaningless name like `abc.py`. When you save the code as `main.py`, it will run automatically when the power is turned on.

4.5.2 Create File and Run it

The code is shown directly in the code section. You can copy it to Thonny and run it as follows.

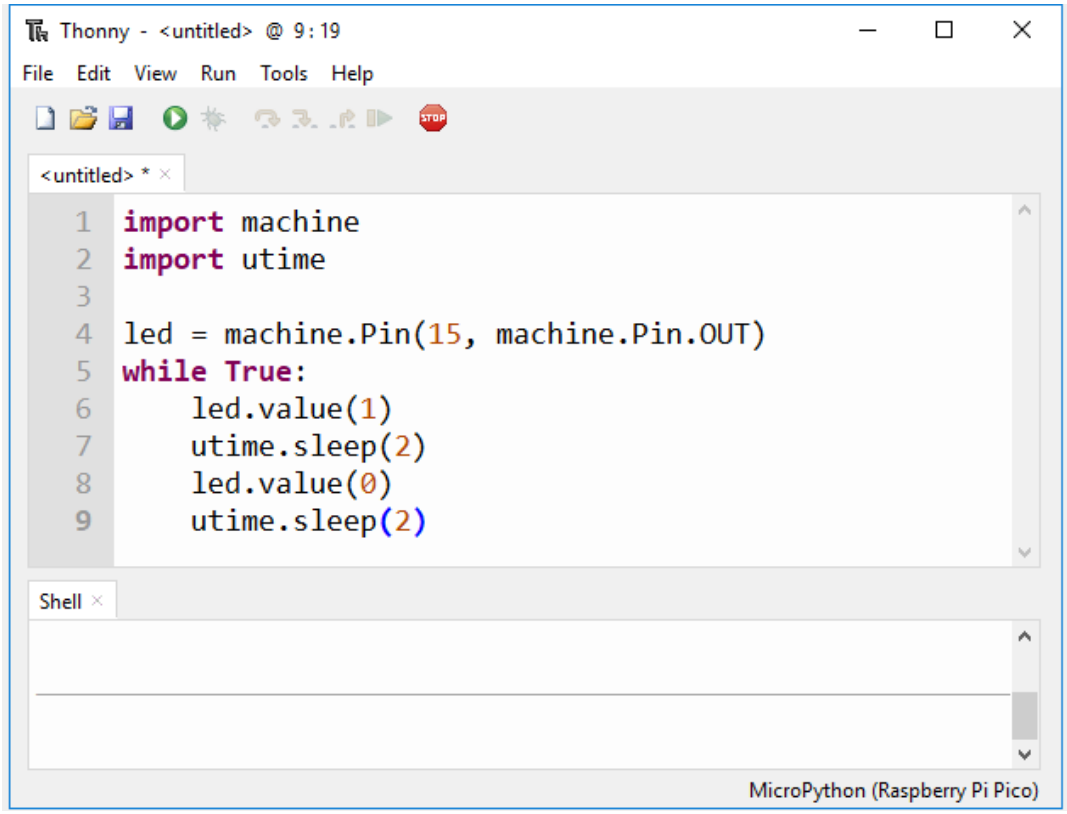
1. Create a new file

   Open Thonny IDE, click **New** button to create a new blank file.



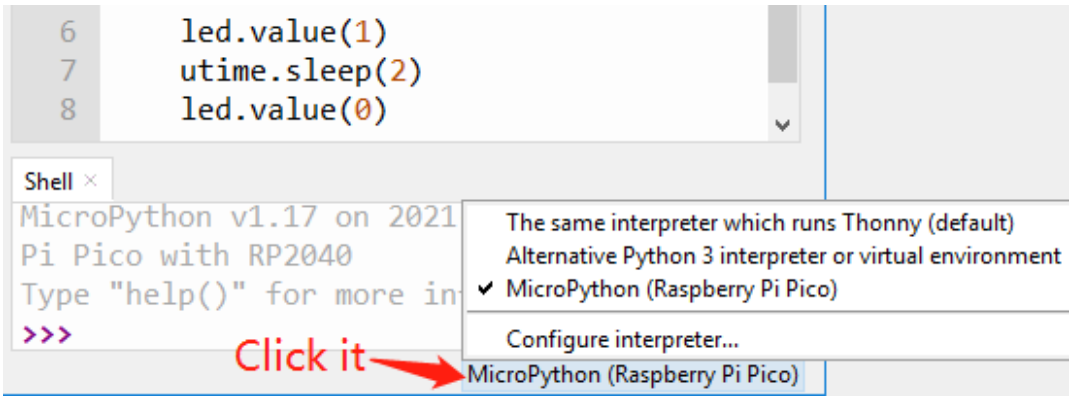
2. Copy code

   Copy the code from the project to the Thonny IDE.



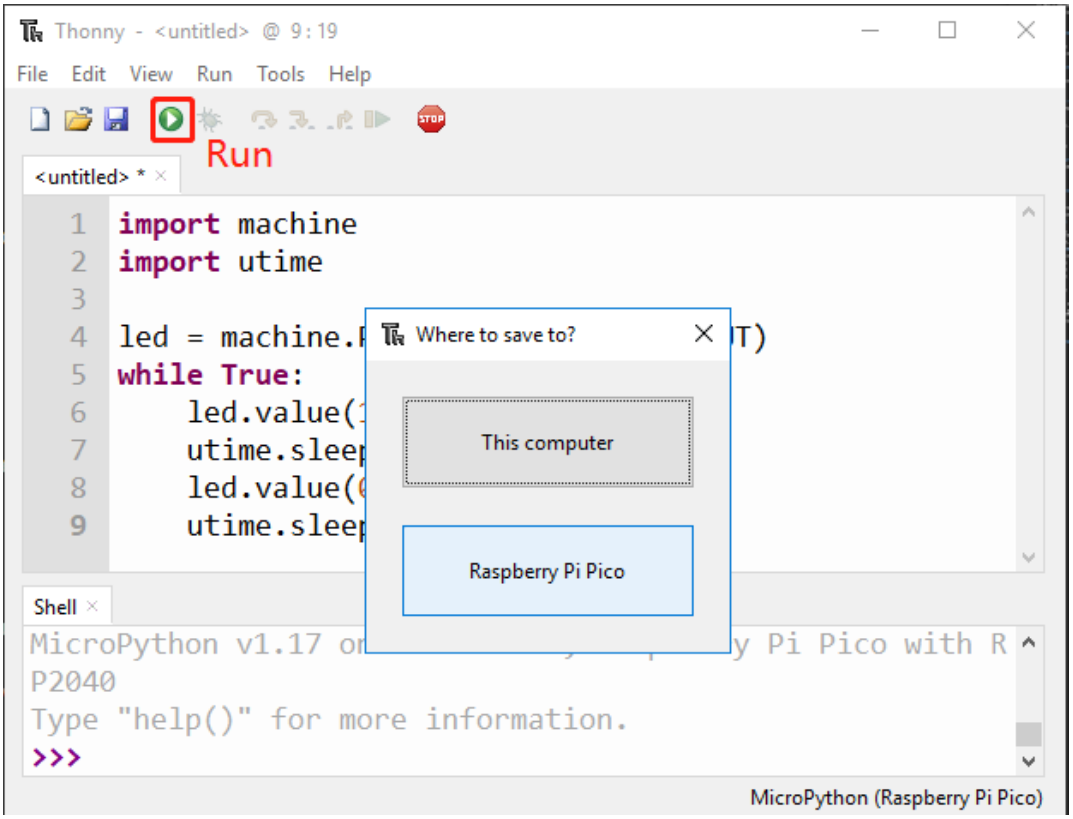
3. Select correct interpreter

   Plug the Pico W into your computer with a micro USB cable and select the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.



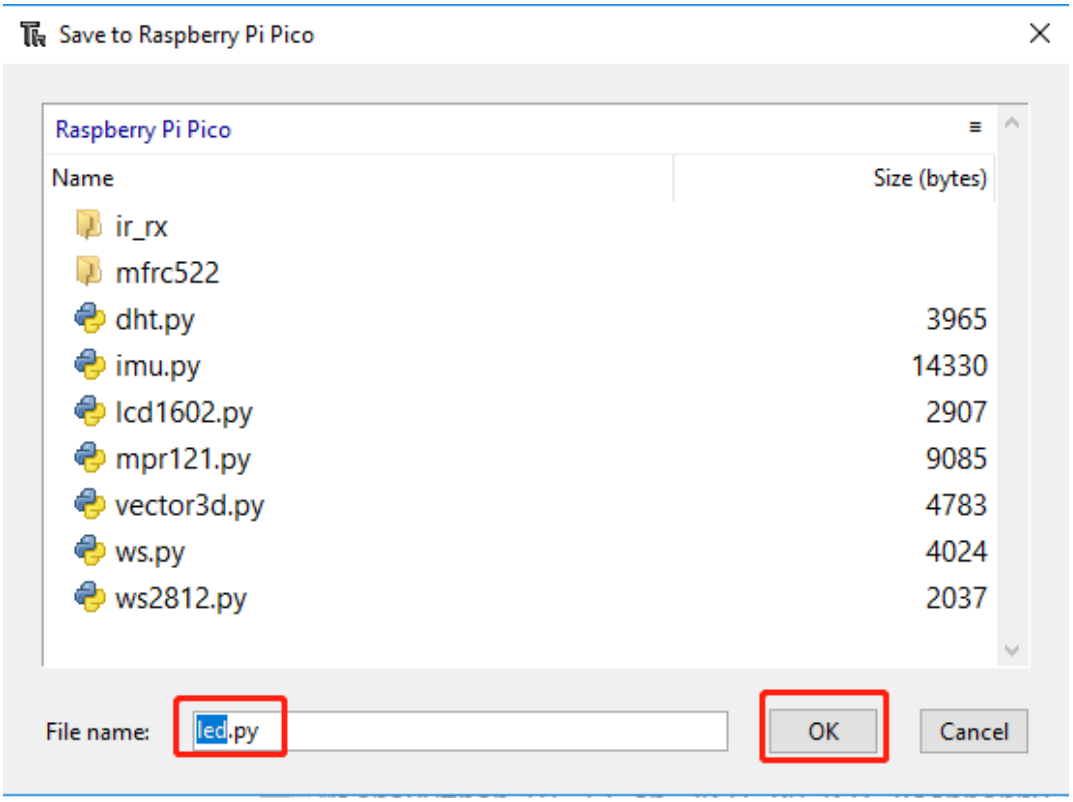
4. Run and save the code

   You need click **Run Current Script** or simply press F5 to run it. If your code has not been saved, a window will pop up asking to save to **This computer** or **Raspberry Pi Pico**.



Note: Thonny saves your program on the Raspberry Pi Pico W hen you tell him to, so if you unplug the Pico W and plug it into someone else's computer, your program remains intact.

Click OK after selecting the location, naming the file and adding the extension **.py**.



Note: Regardless of what name you give your code, it's best to describe what type of code it is, and not give it a meaningless name like `abc.py`. When you save the code as `main.py`, it will run automatically when the power is turned on.

Once your program is saved, it will run automatically and you will see the following information in the Shell area.

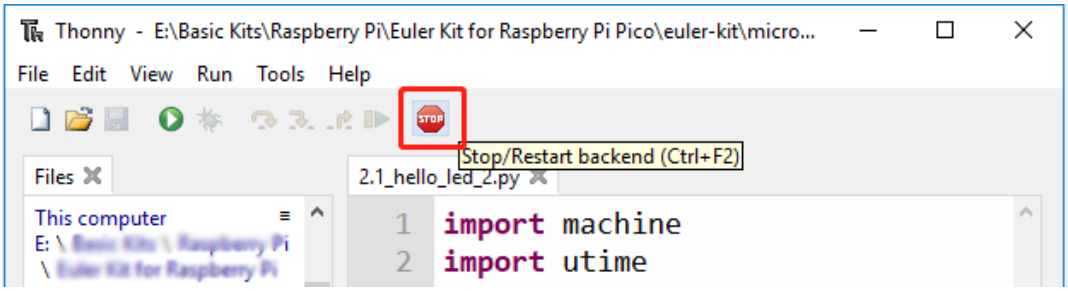
Click **View** -> **Edit** to open the Shell window if it does not appear on your Thonny.

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

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

- The first line shows the version of MicroPython, the date, and your device information.
- The second line prompts you to enter "help()" to get some help.
- The third line is a command from Thonny telling the MicroPython interpreter on your Pico W to run the contents of the script area - "EDITOR_CONTENT".
- If there is any message after the third line, it is usually a message that you tell MicroPython to print, or an error message for the code.

5. Stop running



To stop the running code, click the **Stop/Restart backend** button. The **%RUN -c $EDITOR_CONTENT** command will disappear after stopping.

6. Open file

   Here are two ways to open a saved code file.

   - The first method is to click the open icon on the Thonny toolbar, just like when you save a program, you will be asked if you want to open it from **this computer** or **Raspberry Pi Pico**, for example, click **Raspberry Pi Pico** and you will see a list of all the programs you have saved on the Pico W.
   - The second is to open the file preview directly by clicking **View**-> **File**-> and then double-clicking on the corresponding `.py` file to open it.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.6 1.6 (Optional) MicroPython Basic Syntax

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.6.1 Indentation

Indentation refers to the spaces at the beginning of a code line. Like standard Python programs, MicroPython programs usually run from top to bottom: It traverses each line in turn, runs it in the interpreter, and then continues to the next line, Just like you type them line by line in the Shell. A program that just browses the instruction list line by line is not very smart, though – so MicroPython, just like Python, has its own method to control the sequence of its program execution: indentation.

You must put at least one space before print(), otherwise an error message "Invalid syntax" will appear. It is usually recommended to standardise spaces by pressing the Tab key uniformly.

```
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
```

You must use the same number of spaces in the same block of code, or Python will give you an error.

```
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
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.6.2 Comments

The comments in the code help us understand the code, make the entire code more readable and comment out part of the code during testing, so that this part of the code does not run.

Single-line Comment

Single-line comments in MicroPython begin with #, and the following text is considered a comment until the end of the line. Comments can be placed before or after the code.

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

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

Comments are not necessarily text used to explain the code. You can also comment out part of the code to prevent micropython from running the code.

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

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

Multi-line comment

If you want to comment on multiple lines, you can use multiple # signs.

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

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

Or, you can use multi-line strings instead of expected.

Since MicroPython ignores string literals that are not assigned to variables, you can add multiple lines of strings (triple quotes) to the code and put comments in them:

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

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

As long as the string is not assigned to a variable, MicroPython will ignore it after reading the code and treat it as if you made a multi-line comment.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.6.3 Print()

The `print()` function prints the specified message to the screen, or other standard output device. The message can be a string, or any other object, the object will be converted into a string before written to the screen.

Print multiple objects:

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

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

Print tuples:

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

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

Print two messages and specify the separator:

```
print("Hello", "how are you?", sep="---")
```

```
>>> %Run -c $EDITOR_CONTENT
Hello---how are you?
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.6.4 Variables

Variables are containers used to store data values.

Creating a variable is very simple. You only need to name it and assign it a value. You don't need to specify the data type of the variable when assigning it, because the variable is a reference, and it accesses objects of different data types through assignment.

Naming variables must follow the following rules:

- Variable names can only contain numbers, letters, and underscores
- The first character of the variable name must be a letter or underscore
- Variable names are case sensitive

Create Variable

There is no command for declaring variables in MicroPython. Variables are created when you assign a value to it for the first time. It does not need to use any specific type declaration, and you can even change the type after setting the variable.

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

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

Casting

If you want to specify the data type for the variable, you can do it by casting.

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

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

Get the Type

You can get the data type of a variable with the *type()* function.

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

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

Single or Double Quotes?

In MicroPython, single quotes or double quotes can be used to define string variables.

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

Case-Sensitive

Variable names are case-sensitive.

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

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

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.6.5 If Else

Decision making is required when we want to execute a code only if a certain condition is satisfied.

if

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

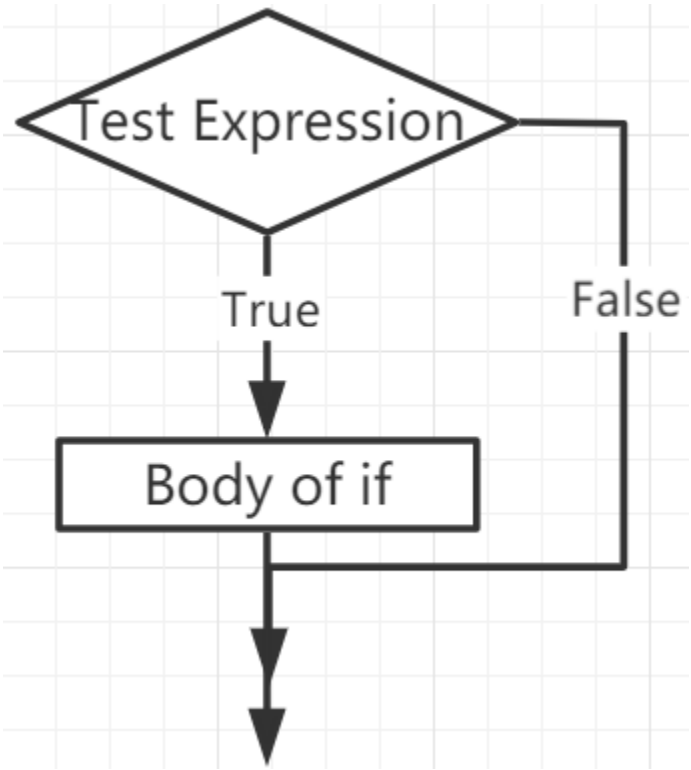
Here, the program evaluates the `test expression` and executes the `statement` only when the `test expression` is True.

If `test expression` is False, then `statement(s)` will not be executed.

In MicroPython, indentation means the body of the `if` statement. The body starts with an indentation and ends with the first unindented line.

Python interprets non-zero values as "True". None and 0 are interpreted as "False".

if Statement Flowchart



Example

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

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

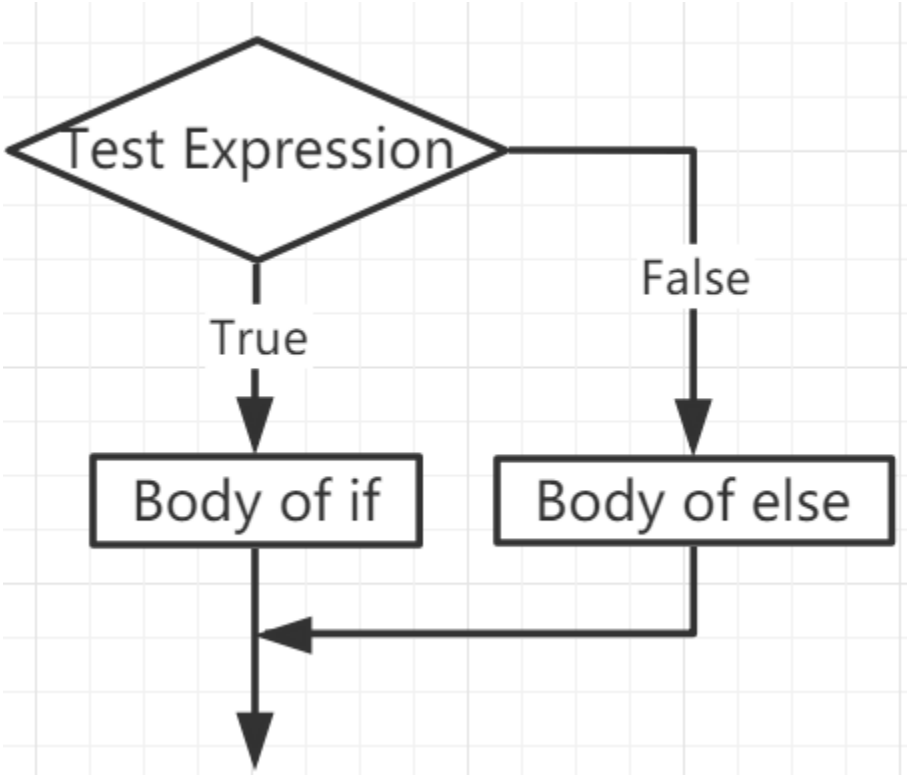
if...else

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

The `if..else` statement evaluates `test expression` and will execute the body of `if` only when the test condition is `True`.

If the condition is `False`, the body of `else` is executed. Indentation is used to separate the blocks.

if...else Statement Flowchart



Example

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

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

if...elif...else

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

`Elif` is short for `else if`. It allows us to check multiple expressions.

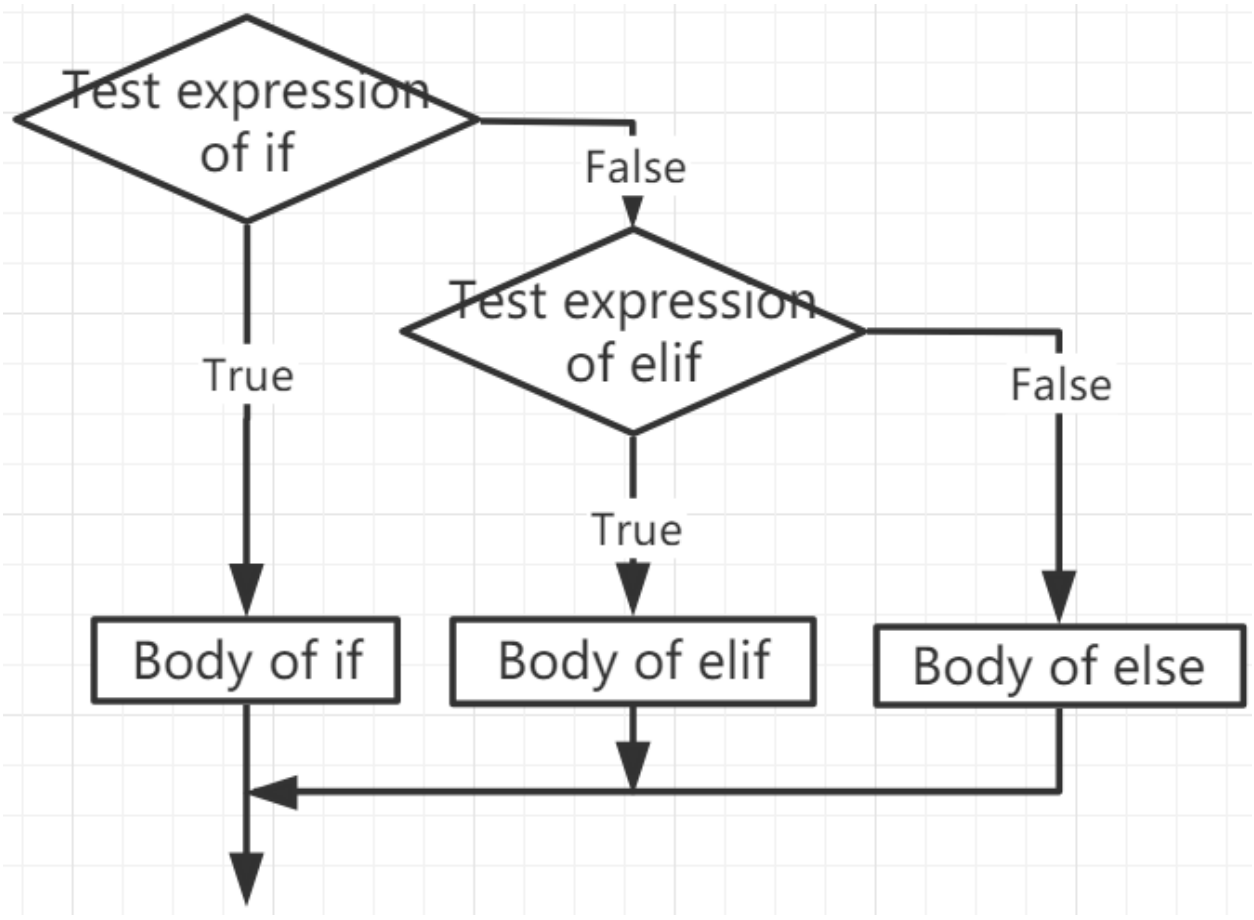
If the condition of the `if` is False, the condition of the next elif block is checked, and so on.

If all conditions are `False`, the body of `else` is executed.

Only one of several `if...elif...else` blocks is executed according to the conditions.

The `if` block can only have one `else` block. But it can have multiple `elif` blocks.

if...elif...else Statement Flowchart



Example

```
x = 10
y = 9

if x > y:
```

(continues on next page)

(continued from previous page)

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

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

Nested if

We can embed an if statement into another if statement, and then call it a nested if statement.

Example

```
x = 67

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

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

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.6.6 While Loops

The `while` statement is used to execute a program in a loop, that is, to execute a program in a loop under certain conditions to handle the same task that needs to be processed repeatedly.

Its basic form is:

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

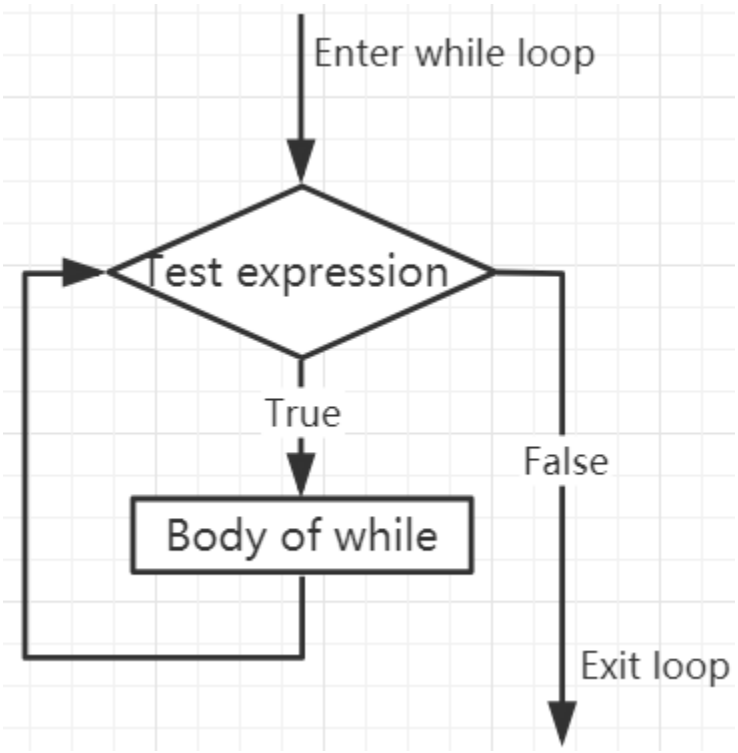
In the `while` loop, first check the `test expression`. Only when `test expression` evaluates to `True`, enter the body of the while. After one iteration, check the `test expression` again. This process continues until `test expression` evaluates to `False`.

In MicroPython, the body of the `while` loop is determined by indentation.

The body starts with an indentation and ends with the first unindented line.

Python interprets any non-zero value as `True`. None and 0 are interpreted as `False`.

while Loop Flowchart




```
x = 10

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

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

(continues on next page)

(continued from previous page)

```
7
6
5
4
3
2
1
```

Break Statement

With the break statement we can stop the loop even if the while condition is true:

```
x = 10

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

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

While Loop with Else

Like the `if` loop, the `while` loop can also have an optional `else` block.

If the condition in the `while` loop is evaluated as `False`, the `else` part is executed.

```
x = 10

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

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

(continues on next page)

(continued from previous page)

```
2
1
Game Over
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.6.7 For Loops

The `for` loop can traverse any sequence of items, such as a list or a string.

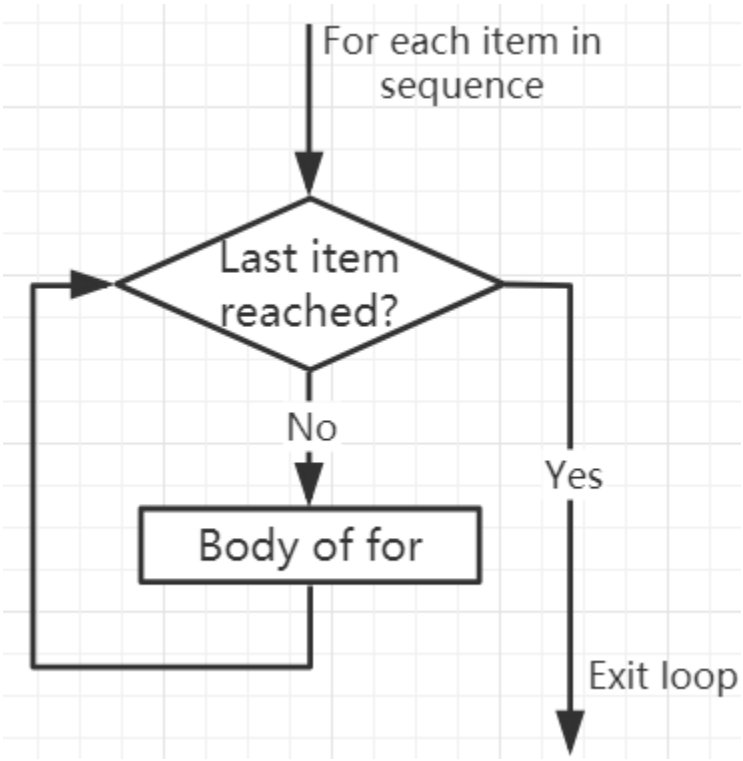
The syntax format of for loop is as follows:

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

Here, `val` is a variable that gets the value of the item in the sequence in each iteration.

The loop continues until we reach the last item in the sequence. Use indentation to separate the body of the `for` loop from the rest of the code.

Flowchart of for Loop




```
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
```

The break Statement

With the break statement we can stop the loop before it has looped through all the items:

```
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
```

The continue Statement

With the `continue` statement we can stop the current iteration of the loop, and continue with the next:

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

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

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

The range() function

We can use the range() function to generate a sequence of numbers. range(6) will produce numbers between 0 and 5 (6 numbers).

We can also define start, stop and step size as range(start, stop, step_size). If not provided, step_size defaults to 1.

In a sense of range, the object is "lazy" because when we create the object, it does not generate every number it "contains". However, this is not an iterator because it supports in, len and `__getitem__` operations.

This function will not store all values in memory; it will be inefficient. So it will remember the start, stop, step size and generate the next number during the journey.

To force this function to output all items, we can use the function list().

```
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]
```

We can use `range()` in a `for` loop to iterate over a sequence of numbers. It can be combined with the len() function to use the index to traverse the sequence.

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

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

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

Else in For Loop

The `for` loop can also have an optional `else` block. If the items in the sequence used for the loop are exhausted, the `else` part is executed.

The `break` keyword can be used to stop the `for` loop. In this case, the `else` part will be ignored.

Therefore, if no interruption occurs, the `else` part of the `for` loop will run.

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

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

The else block will NOT be executed if the loop is stopped by a break statement.

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

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

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.6.8 Functions

In MicroPython, a function is a group of related statements that perform a specific task.

Functions help break our program into smaller modular blocks. As our plan becomes larger and larger, functions make it more organized and manageable.

In addition, it avoids duplication and makes the code reusable.

Create a Function

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

- A function is defined using the `def` keyword
- A function name to uniquely identify the function. Function naming is the same as variable naming, and both follow the following rules.
  - Can only contain numbers, letters, and underscores.
  - The first character must be a letter or underscore.
  - Case sensitive.
- Parameters (arguments) through which we pass values to a function. They are optional.
- The colon (:) marks the end of the function header.
- Optional docstring, used to describe the function of the function, we usually use triple quotes so that the docstring can be expanded to multiple lines.
- One or more valid Micropython statements that make up the function body. Statements must have the same indentation level (usually 4 spaces).
- Each function needs at least one statement, but if for some reason there is a function that does not contain any statement, please put in the pass statement to avoid errors.
- An optional `return` statement to return a value from the function.

Calling a Function

To call a function, add parentheses after the function name.

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

my_function()
```

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

The return Statement

The return statement is used to exit a function and return to the place where it was called.

Syntax of return

```
return [expression_list]
```

The statement can contain an expression that is evaluated and returns a value. If there is no expression in the statement, or the `return` statement itself does not exist in the function, the function will return a `None` object.

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

print(my_function())
```

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

Here, `None` is the return value, because the `return` statement is not used.

Arguments

Information can be passed to the function as arguments.

Specify arguments in parentheses after the function name. You can add as many arguments as you need, just separate them with commas.

```
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!
```

Number of Arguments

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 parameters, you have to call the function with 2 arguments, not more, and not less.

```
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!")
```

Here, the function welcome() has 2 parameters.

Since we called this function with two arguments, the function runs smoothly without any errors.

If it is called with a different number of arguments, the interpreter will display an error message.

The following is the call to this function, which contains one and one no arguments and their respective error messages.

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

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

```
welcome()No arguments
```

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

Default Arguments

In MicroPython, we can use the assignment operator (=) to provide a default value for the parameter.

If we call the function without argument, it uses the default value.

```
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!
```

In this function, the parameter `name` has no default value and is required (mandatory) during the call.

On the other hand, the default value of the parameter `msg` is "Welcome to China!". Therefore, it is optional during the call. If a value is provided, it will overwrite the default value.

Any number of arguments in the function can have a default value. However, once there is a default argument, all arguments on its right must also have default values.

This means that non-default arguments cannot follow default arguments.

For example, if we define the above function header as:

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

We will receive the following error message:

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

Keyword Arguments

When we call a function with certain values, these values will be assigned to arguments based on their position.

For example, in the above function welcome(), when we called it as welcome("Lily", "Welcome to China"), the value "Lily" gets assigned to the `name` and similarly "Welcome to China" to parameter `msg`.

MicroPython allows calling functions with keyword arguments. When we call the function in this way, the order (position) of the arguments can be changed.

```
# 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!")
```

As we can see, we can mix positional arguments and keyword arguments during function calls. But we must remember that the keyword arguments must come after the positional arguments.

Having a positional argument after a keyword argument will result in an error.

For example, if the function call as follows:

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

Will result in an error:

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

Arbitrary Arguments

Sometimes, if you do not know the number of arguments that will be passed to the function in advance.

In the function definition, we can add an asterisk (*) before the parameter name.

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

welcome("Lily","John","Wendy")
```

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

Here, we have called the function with multiple arguments. These arguments are packed into a tuple before being passed into the function.

Inside the function, we use a for loop to retrieve all the arguments.

Recursion

In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.

This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

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

rec_func(6)
```

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

In this example, rec_func() is a function that we have defined to call itself ("recursion"). We use the `i` variable as the data, and it will decrement (-1) every time we recurse. When the condition is not greater than 0 (that is, 0), the recursion ends.

For new developers, it may take some time to determine how it works, and the best way to test it is to test and modify it.

Advantages of Recursion

- Recursive functions make the code look clean and elegant.
- A complex task can be broken down into simpler sub-problems using recursion.
- Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion

- Sometimes the logic behind recursion is hard to follow through.
- Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
- Recursive functions are hard to debug.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.6.9 Data Types

Built-in Data Types

MicroPython has the following data types:

- Text Type: str
- Numeric Types: int, float, complex
- Sequence Types: list, tuple, range
- Mapping Type: dict
- Set Types: set, frozenset
- Boolean Type: bool
- Binary Types: bytes, bytearray, memoryview

Getting the Data Type

You can get the data type of any object by using the `type()` function:

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

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

Setting the Data Type

MicroPython does not need to set the data type specifically, it has been determined when you assign a value to the variable.

```
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'>
>>>
```

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

Example	Date Type
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

You can print some of them to see the result.

```
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
>>>
```

Type Conversion

You can convert from one type to another with the int(), float(), and complex() methods: Casting in python is therefore done using constructor functions:

- int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

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

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

Note: You cannot convert complex numbers into another number type.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.6.10 Operators

Operators are used to perform operations on variables and values.

- *Arithmetic Operators*
- *Assignment operators*
- *Comparison Operators*
- *Logical Operators*
- *Identity Operators*
- *Membership Operators*
- *Bitwise Operators*

Arithmetic Operators

You can use arithmetic operators to do some common mathematical operations.

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponentiation
//	Floor division

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

Assignment operators

Assignment operators can used to assign values to variables.

Operator	Example	Same As
=	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
>>>
```

Comparison Operators

Comparison operators are used to compare two values.

Operator	Name
==	Equal
!=	Not equal
<	Less than
>	Greater than
>=	Greater than or equal to
<=	Less than or equal to

```
a = 6
b = 8

print(a>b)
```

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

Return **False**, beause the **a** is less than the **b**.

Logical Operators

Logical operators are used to combine conditional statements.

Operator	Description
and	Returns True if both statements are true
or	Returns True if one of the statements is true
not	Reverse the result, returns False if the result is true

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

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

Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location.

Operator	Description
is	Returns True if both variables are the same object
is not	Returns True if both variables are not the same object

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

print(a is c)
# returns True because z is the same object as x

print(a is b)
# returns False because x is not the same object as y, even if they have the same content

print(a == b)
# returns True because x is equal to y
```

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

Membership Operators

Membership operators are used to test if a sequence is presented in an object.

Operator	Description
`in`	Returns True if a sequence with the specified value is present in the object
`not in`	Returns True if a sequence with the specified value is not present in the object

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

print("welcome" in a)
```

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

Bitwise Operators

Bitwise operators are used to compare (binary) numbers.

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

```
num = 2

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

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

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.6.11 Lists

Lists are used to store multiple items in a single variable, and are created using square brackets:

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

List items are changeable, ordered, and allow duplicate values. The list items are indexed, with the first item having index [0], the second item having index [1], and so on.

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

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

A list can contain different data types:

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

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

List Length

To determine how many items are in the list, use the len() function.

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

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

Check List items

Print the second item of the list:

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

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

Print the last one item of the list:

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

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

Print the second, third item:

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

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

Change List Items

Change the second, third item:

```
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]
```

Change the second value by replacing it with two values:

```
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]
```

Add List Items

Using the append() method to add an item:

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

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

Insert an item as the second position:

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

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

Remove List Items

The remove() method removes the specified item.

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

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

The pop() method removes the specified index. If you do not specify the index, the pop() method removes the last item.

```
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]
```

The `del` keyword also removes the specified index:

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

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

The clear() method empties the list. The list still remains, but it has no content.

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

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

2. Output & Input

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.7 2.1 Hello, LED!

Just as printing "Hello, world!" is the first step in learning to program, using a program to drive an LED is the traditional introduction to learning physical programming.

- *LED*

Required Components

In this project, we need the following components.

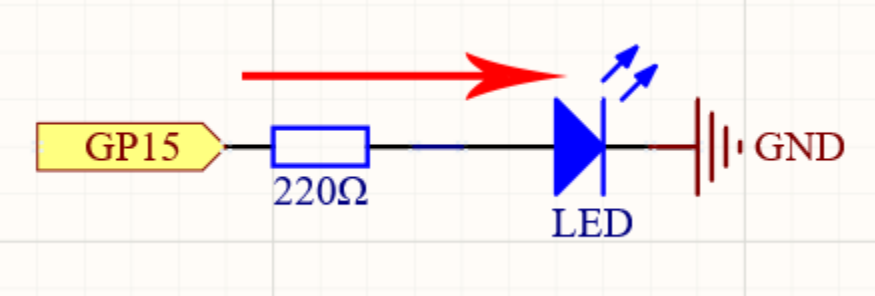
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

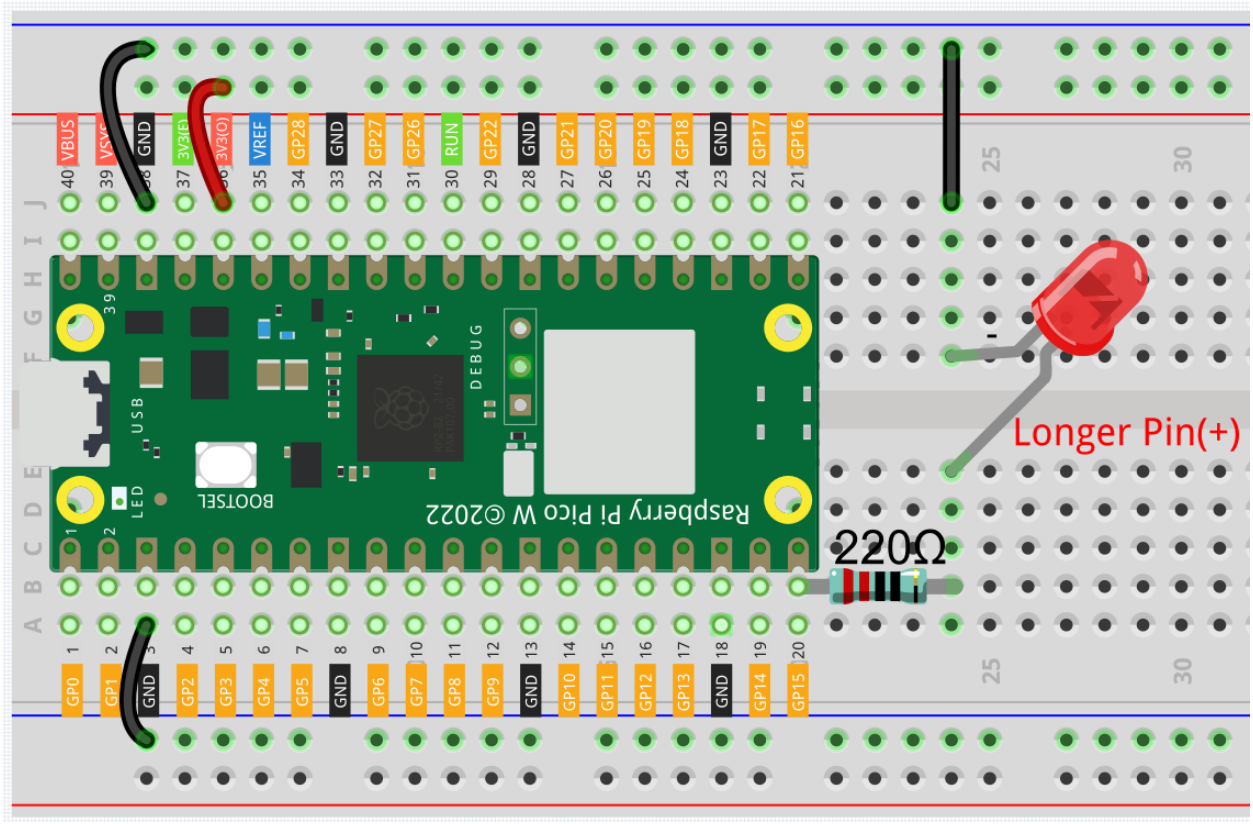
SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(220)	
6	*LED*	1	

Schematic



This circuit works on a simple principle, and the current direction is shown in the figure. The LED will light up after the 220ohm current limiting resistor when GP15 outputs high level (3.3v). The LED will turn off when GP15 outputs low level (0v).

Wiring



To build the circuit, let's follow the current's direction!

1. The LED is powered by the GP15 pin of the Pico W board, and the circuit begins here.
2. To protect the LED, the current must pass through a 220 ohm resistor. One end of the resistor should be inserted into the same row as the Pico W GP15 pin (row 20 in my circuit), and the other end should be inserted into the free row of the breadboard (row 24).

   Note: The color ring of the 220 ohm resistor is red, red, black, black and brown.

3. If you pick up the LED, you will see that one of its leads is longer than the other. Connect the longer lead to the same row as the resistor, and the shorter lead to the same row across the middle gap on the breadboard.

   Note: The longer lead is the anode, which represents the positive side of the circuit; the shorter lead is the cathode, which represents the negative side.

   The anode needs to be connected to the GPIO pin through a resistor; the cathode needs to be connected to the GND pin.

4. Using a male-to-male (M2M) jumper wire, connect the LED short pin to the breadboard's negative power bus.
5. Connect the GND pin of Pico W to the negative power bus using a jumper.

Code

Note:

- Open the `2.1_hello_led.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)
```

After the code runs, you will see the LED blinking.

How it works?

The machine library is required to use GPIO.

```
import machine
```

The library contains all the instructions needed to communicate between MicroPython and Pico W. In the absence of this line of code, we will not be able to control any GPIOs.

The next thing to notice is this line:

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

The object `led` is defined here. Technically, it can be any name, such as x, y, banana, Michael_Jackson, or any character. To ensure that the program is easy to read, it is best to use a name that describes the purpose.

In the second part of this line (the part after the equal sign), we call the Pin function found in the `machine` library. It is used to tell Pico's GPIO pins what to do. A `Pin` function has two parameters: the first (15) represents the pin to set; The second parameter (machine.Pin.OUT) specifies that the pin should be output rather than input.

The above code has "set" the pin, but it will not light up the LED. To do this, we also need to "use" the pin.

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

The GP15 pin has been set up previously and named `led`. The function of this statement is to set the value of `led` to 1 to turn the LED on.

All in all, to use GPIO, these steps are necessary:

- **import machine library**: This is necessary, and it is only executed once.
- **Set GPIO**: Before using, each pin should be set.
- **Use**: Change the working state of the pin by assigning a value to it.

If we follow the above steps to write an example, then you will get code like this:

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

Run it and you will be able to light up the LED.

Next, we try to add the "extinguished" statement:

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

Based on the code line, this program will turn on the LED first, then turn it off. But when you use it, you will find that this is not the case. There is no light coming from the LED. This is due to the very rapid execution speed between the two lines, much faster than the human eye can react. When the LED lights up, we don't perceive the light instantly. This can be fixed by slowing down the program.

The second line of the program should contain the following statement:

```
import utime
```

Similarly to `machine`, the `utime` library is imported here, which handles all things time-related. The delays we need to use are included in this. Add a delay statement between `led.value(1)` and `led.value(0)` and let them be separated by 2 seconds.

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

This is how the code should look now. We will see that the LED turns on first, then turns off when we run it:

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

Finally, we should make the LED blink. Create a loop, rewrite the program, and it will be what you saw at the beginning of this chapter.

```
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 Loops*

Learn More

There will usually be an API (Application Programming Interface) file associated with the library. It contains all the information necessary to use this library, including detailed descriptions of functions, classes, return types, parameter types, etc.

In this article, we used MicroPython's `machine` and `utime` libraries, we can find more ways to use them here.

- machine.Pin
- utime

Please read the API file to understand this example of making the LED blink!

Note:

- Open the `2.1_hello_led_2.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
import machine
import utime

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

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.8 2.2 Display the Level

The first project is simply to make the LED blink. For this project, let's use the LED Bar Graph, which contains 10 LEDs in a plastic enclosure, generally used to display power or volume levels.



- *LED Bar Graph*

Required Components

In this project, we need the following components.

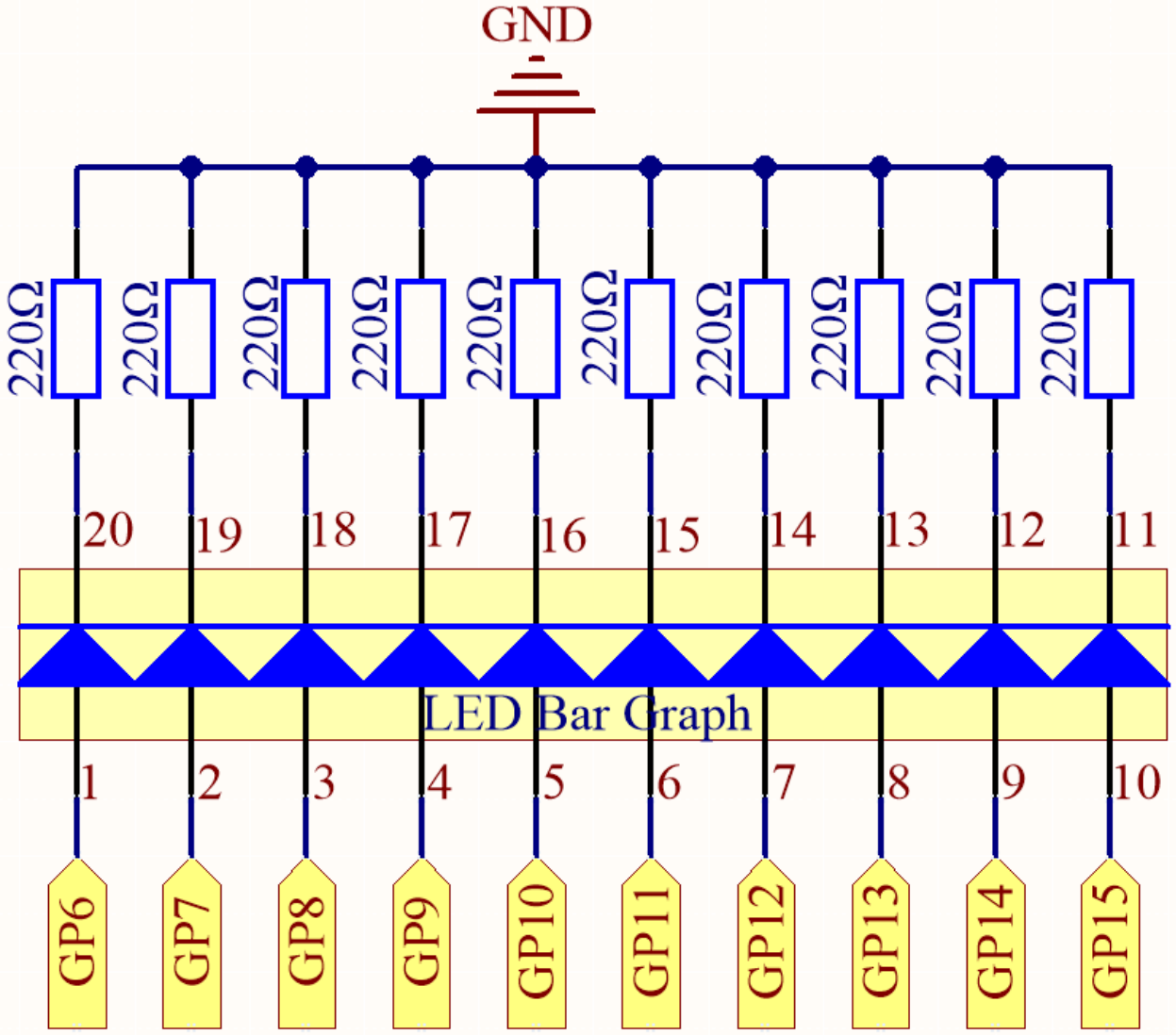
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

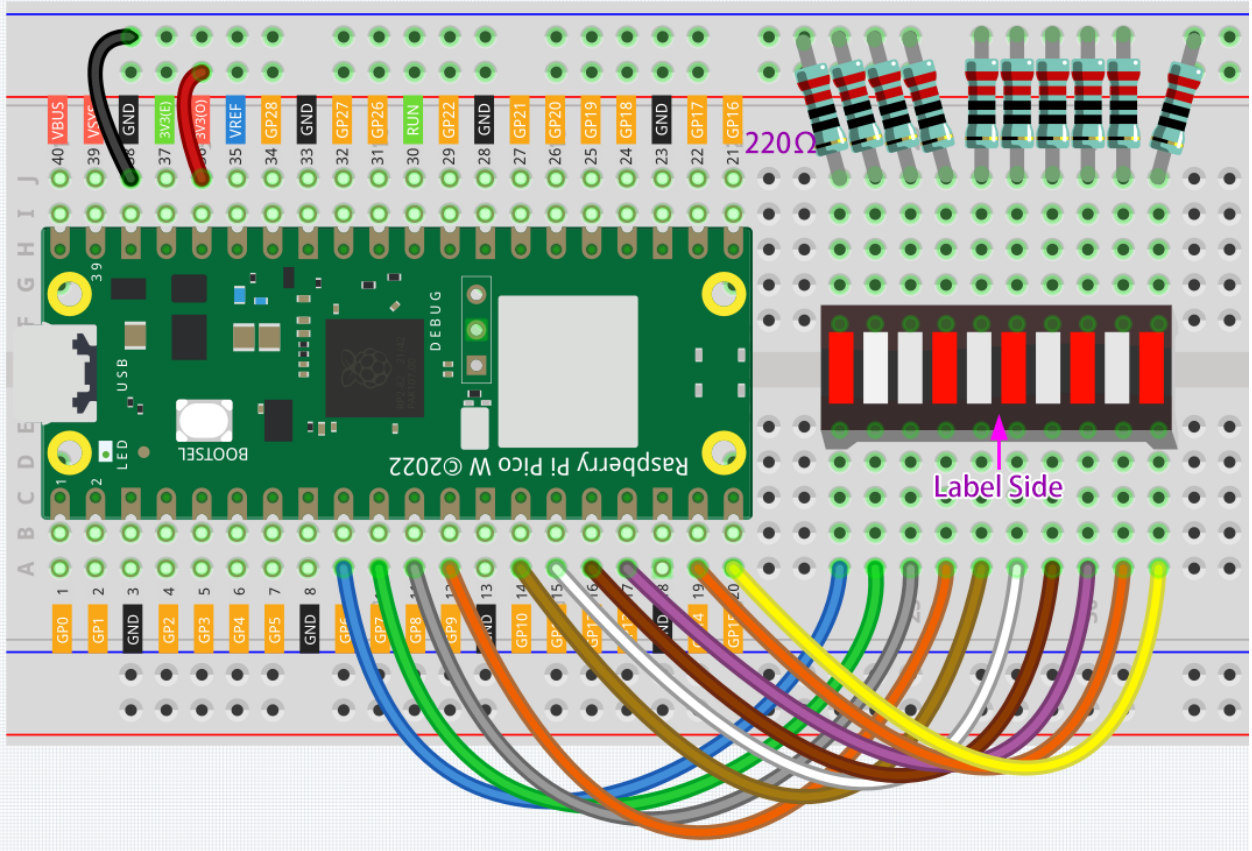
SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	10(220)	
6	*LED Bar Graph*	1	

Schematic



In the LED Bar Graph, there are 10 LEDs, each of which can be controlled individually. Each LED's anode is connected to GP6*GP15, and its cathode to a 220ohm resistor, and then to GND.

Wiring



Code

Note:

- Open the `2.2_display_the_level.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)
```

On the LED Bar Graph, you'll see LEDs lighting up and then turning off in sequence when the program is running.

How it works?

The LED Bar consists of ten LEDs that are controlled by ten pins, which means that we must define these pins. The process would be too tedious if we defined them one by one. So, here we use `Lists`.

Note: Python lists are one of the most versatile data types that allow us to work with multiple elements at once, and created by placing elements inside square brackets [], separated by commas.

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

A list `pin` is defined by this line of code, which contains the ten elements `6,7,8,9,10,11,12,13,14,15`. We can use the index operator [] to access an item in a list. In Python, indices start at 0. So, a list having 10 elements will have an index from 0 to 9. Using this list as an example, `pin[0]` is `6` and `pin[4]` is `10`.

Next, declare an empty list `led` that will be used to define ten LED objects.

```
led = []
```

Due to the length of the list, which is 0, direct operations on the array, such as printing led[0]**, won't work. There are new items we need to add.

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

As a result of this `append()` method, the list `led` has its first item, of length 1, and `led[0]` becomes a valid element despite its current value of `None` (which stands for null).

Our next step is to define `led[0]`, the LED connected to pin 6, as the first LED object.

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

The first LED object has now been defined.

As you can see, we have created the ten pin numbers as a list **pin**, which we can substitute into this line to make it easier to do bulk operations.

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

Use a `for` statement to have all 10 pins execute the above statement.

```
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)
```

- *Lists*
- *For Loops*

Use another `for` loop to make the ten LEDs on the LED Bar switch states one by one.

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

The code is finished by putting the above piece of code in a while loop.

```
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)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.9 2.3 Fading LED

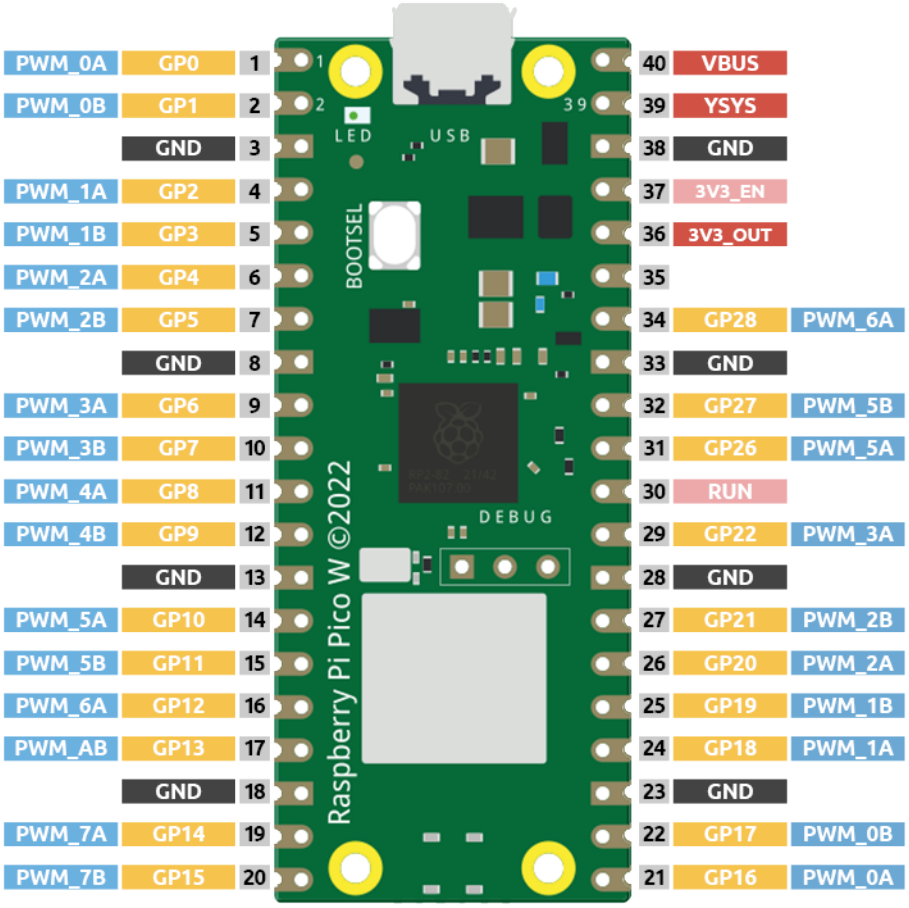
As of now, we have only used two output signals: high level and low level (also called ON and OFF), which is called digital output. However, in actual use, many devices do not simply ON/OFF to work, for example, adjusting the speed of the motor, adjusting the brightness of the desk lamp, and so on. To achieve this goal, a slider that adjusts resistance was used in the past, but it is unreliable and inefficient. Therefore, Pulse width modulation (PWM) has emerged as a feasible solution to such complex problems.

A pulse is a digital output that contains a high level and a low level. The pulse width of these pins can be adjusted by changing the ON/OFF speed.

When we are in a short period of time (like 20ms, which is most people's visual retention period), let the LED turn on, turn off, and turn on again, we won't see it has been turned off, but the brightness of the light will be slightly weaker. During this period, the more time the LED is on, the brighter it becomes. In other words, in the cycle, the wider the pulse, the greater the "electric signal strength" output by the microcontroller. This is how PWM controls LED brightness (or motor speed).

- Pulse-width modulation - Wikipedia

There are some points to pay attention to when Pico W uses PWM. Let's take a look at this picture.



Pico W supports PWM on each GPIO pin, but there are actually 16 independent PWM outputs (instead of 30), distributed between GP0 to GP15 on the left, and the right GPIO's PWM output is identical to the left.

It is important to avoid setting the same PWM channel for different purposes during programming. For example, GP0 and GP16 are both PWM_0A.

Let's try to achieve the faded LED effect after understanding this knowledge.

- *LED*

Required Components

In this project, we need the following components.

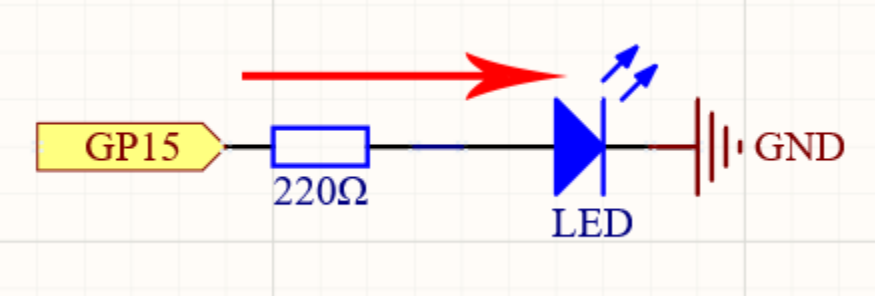
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

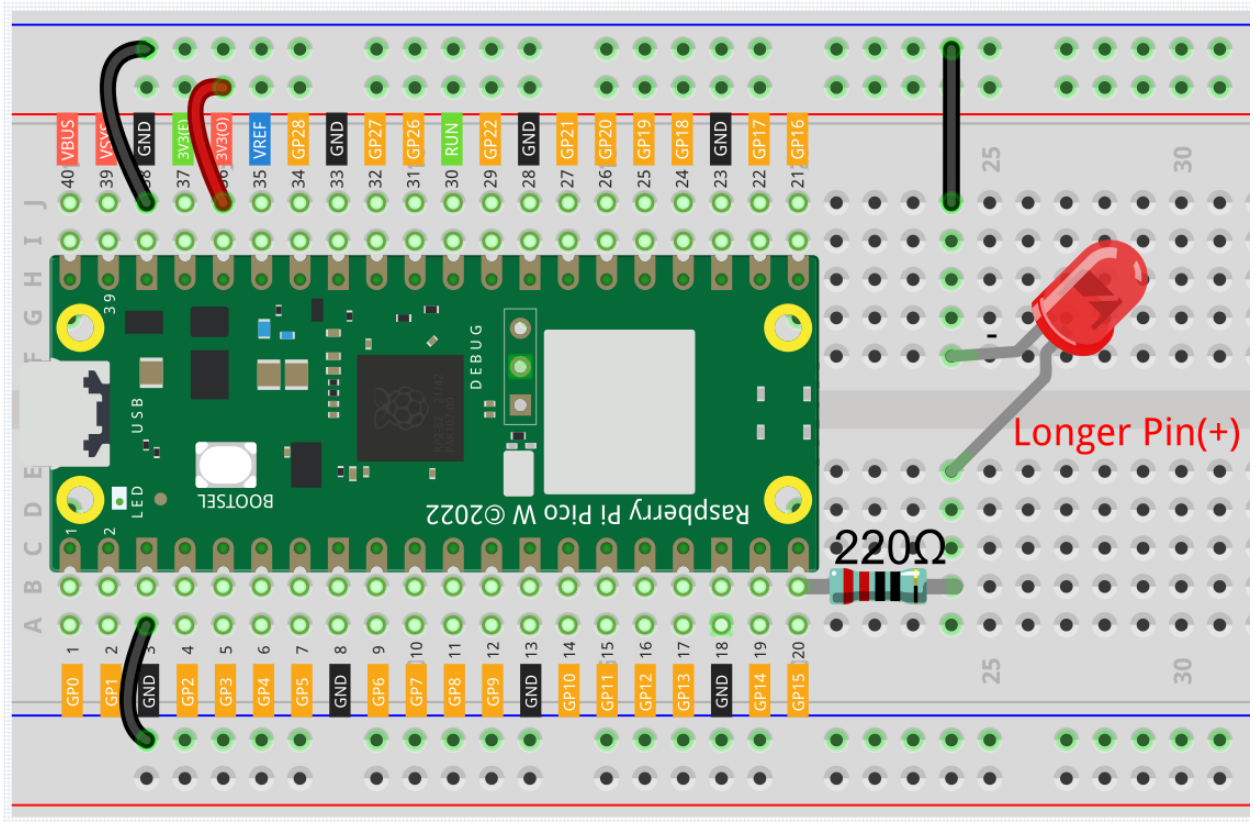
SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(220)	
6	*LED*	1	

Schematic



This project is the same circuit as the first project *2.1 Hello, LED!*, but the signal type is different. The first project is to output digital high and low levels (0&1) directly from GP15 to make the LEDs light up or turn off, this project is to output PWM signal from GP15 to control the brightness of the LED.

Wiring



Code

Note:

- Open the `2.3_fading_led.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)
```

The LED will gradually become brighter as the code runs.

How it works?

Here, we change the brightness of the LED by changing the duty cycle of the GP15's PWM output. Let's take a look at these lines.

```
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))` sets the GP15 pin as PWM output.
- The line `led.freq(1000)` is used to set the PWM frequency, here it is set to 1000Hz, which means 1ms (1/1000) is a cycle.
- The `led.duty_u16()` line is used to set the duty cycle, which is a 16-bit interger(2^16=65536). A 0 indicates 0% duty cycle, which means each cycle has 0% time to output a high level, i.e., all pulses are turned off. The value 65535 indicates a duty cycle of 100%, which means the whole pulse is turned on, and the result is '1'. When it is 32768, it will turn on half a pulse, so the LED will be half as bright when fully on.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.10 2.4 Colorful Light

As we know, light can be superimposed. For example, mix blue light and green light give cyan light, red light and green light give yellow light. This is called "The additive method of color mixing".

- Additive color - Wikipedia

Based on this method, we can use the three primary colors to mix the visible light of any color according to different specific gravity. For example, orange can be produced by more red and less green.

In this chapter, we will use RGB LED to explore the mystery of additive color mixing!

RGB LED is equivalent to encapsulating Red LED, Green LED, Blue LED under one lamp cap, and the three LEDs share one cathode pin. Since the electric signal is provided for each anode pin, the light of the corresponding color can be displayed. By changing the electrical signal intensity of each anode, it can be made to produce various colors.

- *RGB LED*

Required Components

In this project, we need the following components.

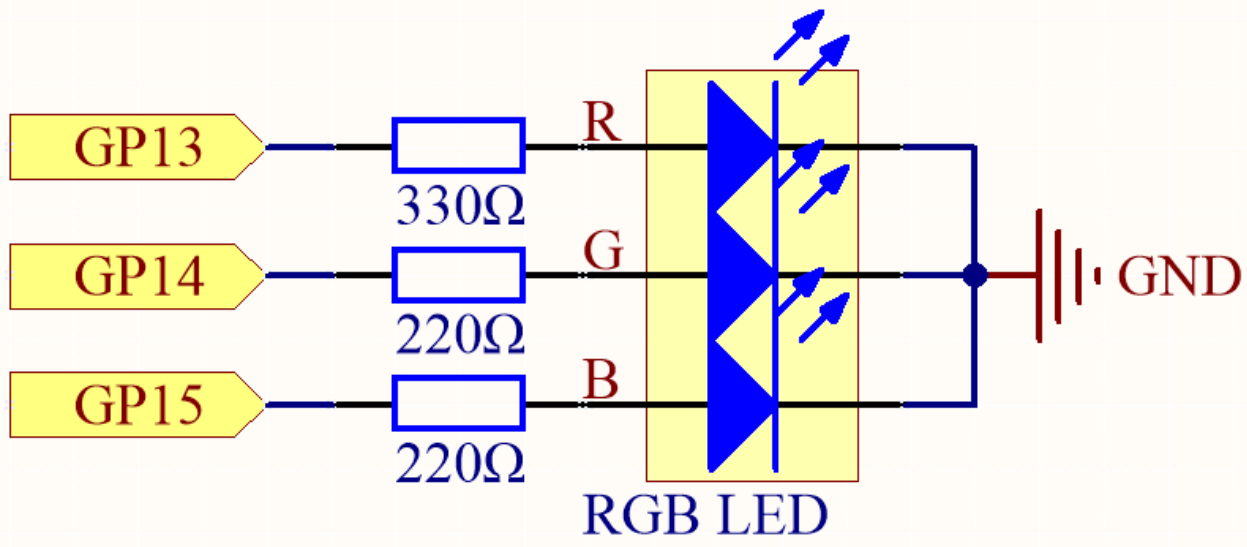
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

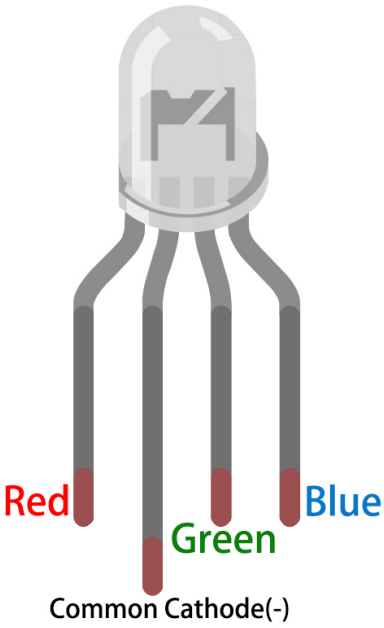
SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	3(1-330, 2-220)	
6	*RGB LED*	1	

Schematic

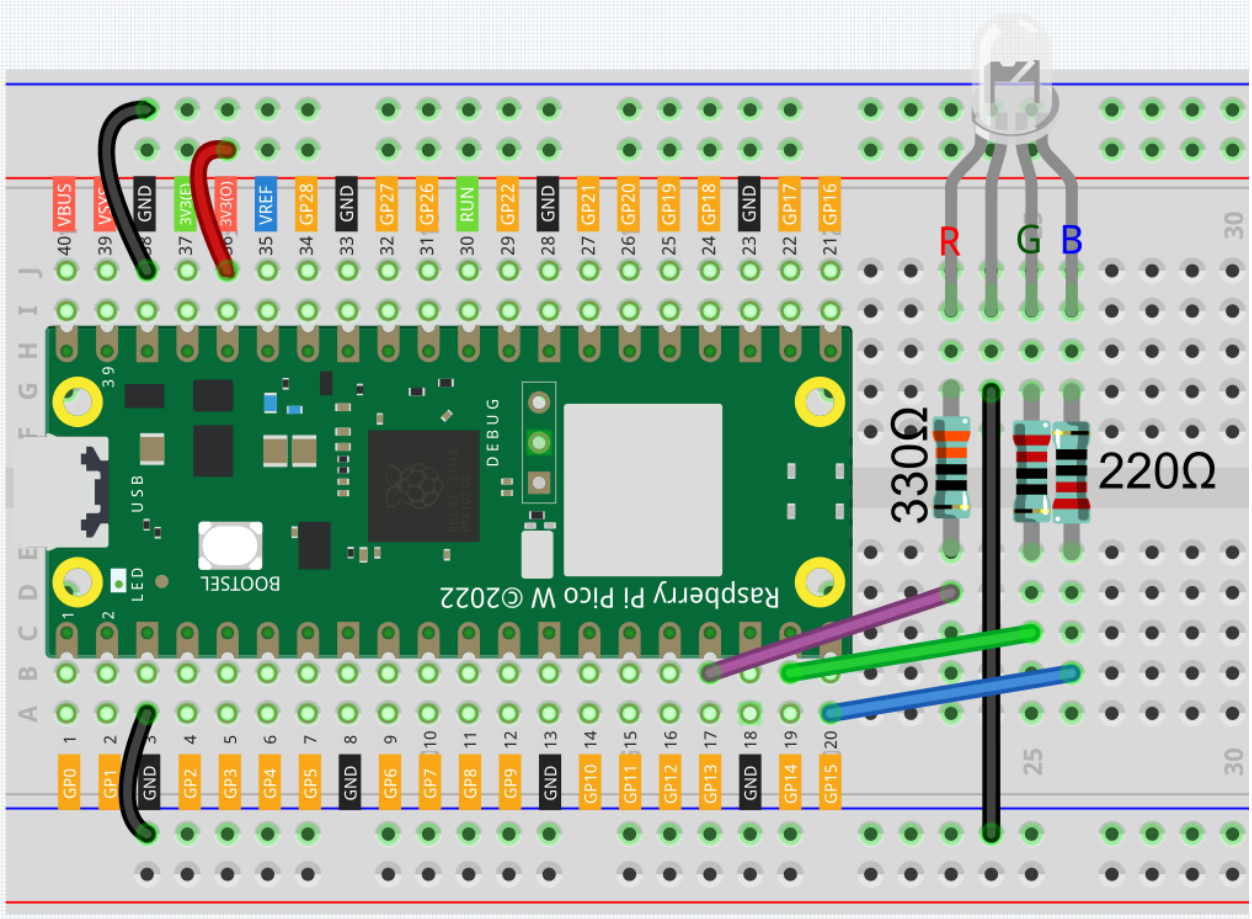


The PWM pins GP13, GP14 and GP15 control the Red, Green and Blue pins of the RGB LED respectively, and connect the common cathode pin to GND. This allows the RGB LED to display a specific color by superimposing light on these pins with different PWM values.

Wiring



The RGB LED has 4 pins: the long pin is the common cathode pin, which is usually connected to GND; the left pin next to the longest pin is Red; and the two pins on the right are Green and Blue.



Code

Note:

- Open the `2.4_colorful_light.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

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

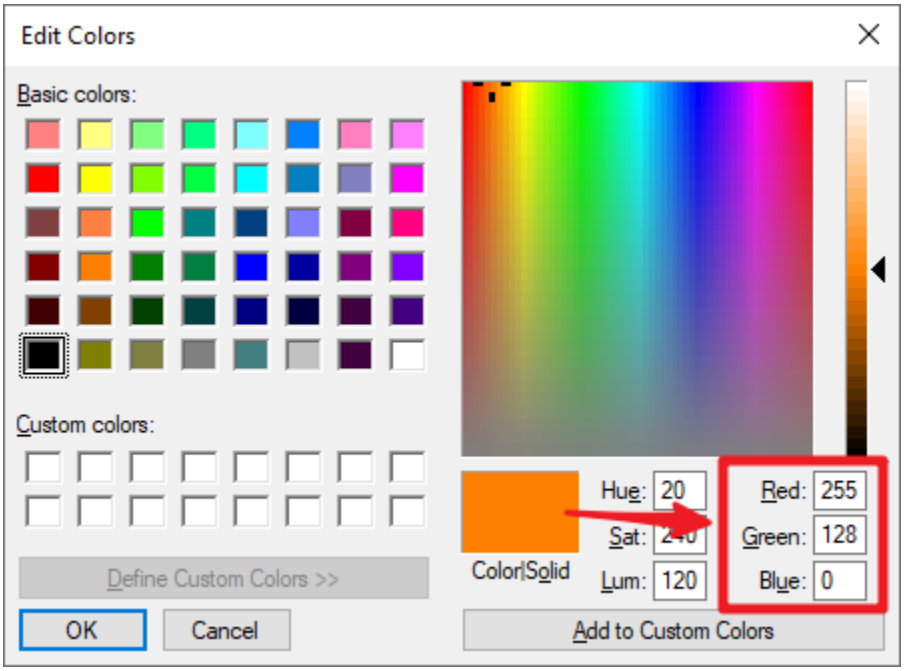
def interval_mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

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)
```

Here, we can choose our favorite color in drawing software (such as paint) and display it with RGB LED.



Write the RGB value into `color_set()`, you will be able to see the RGB light up the colors you want.

How it works?

To allow the three primary colors to work together, we defined a `color_set()` function.

At present, pixels in computer hardware usually use 24-bit representations. Each primary color is divided into 8 bits, and the color value range is 0 to 255. There are 256 possible combinations of each of the three primary colors (don't forget to count 0!), so 256 x 256 x 256 = 16,777,216 colors. The `color_set()` function also uses 24-bit notation, so we can choose a color more easily.

And since the value range of `duty_u16()` is 0~65535 (instead of 0 to 255) when the output signals to RGB LED through PWM, we have defined `color_to_duty()` and `interval_mapping ()` function to map the color values to the duty values.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.11 2.5 Reading Button Value

These pins have both input and output functions, as indicated by their name GPIO (General-purpose input/output). Previously, we used the output function; in this chapter, we will use the input function to input the button value.

- *Button*

Required Components

In this project, we need the following components.

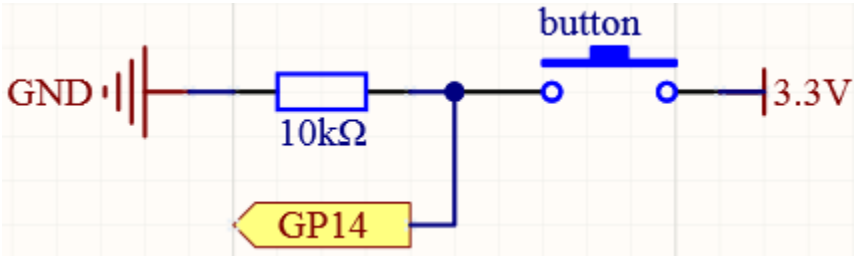
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

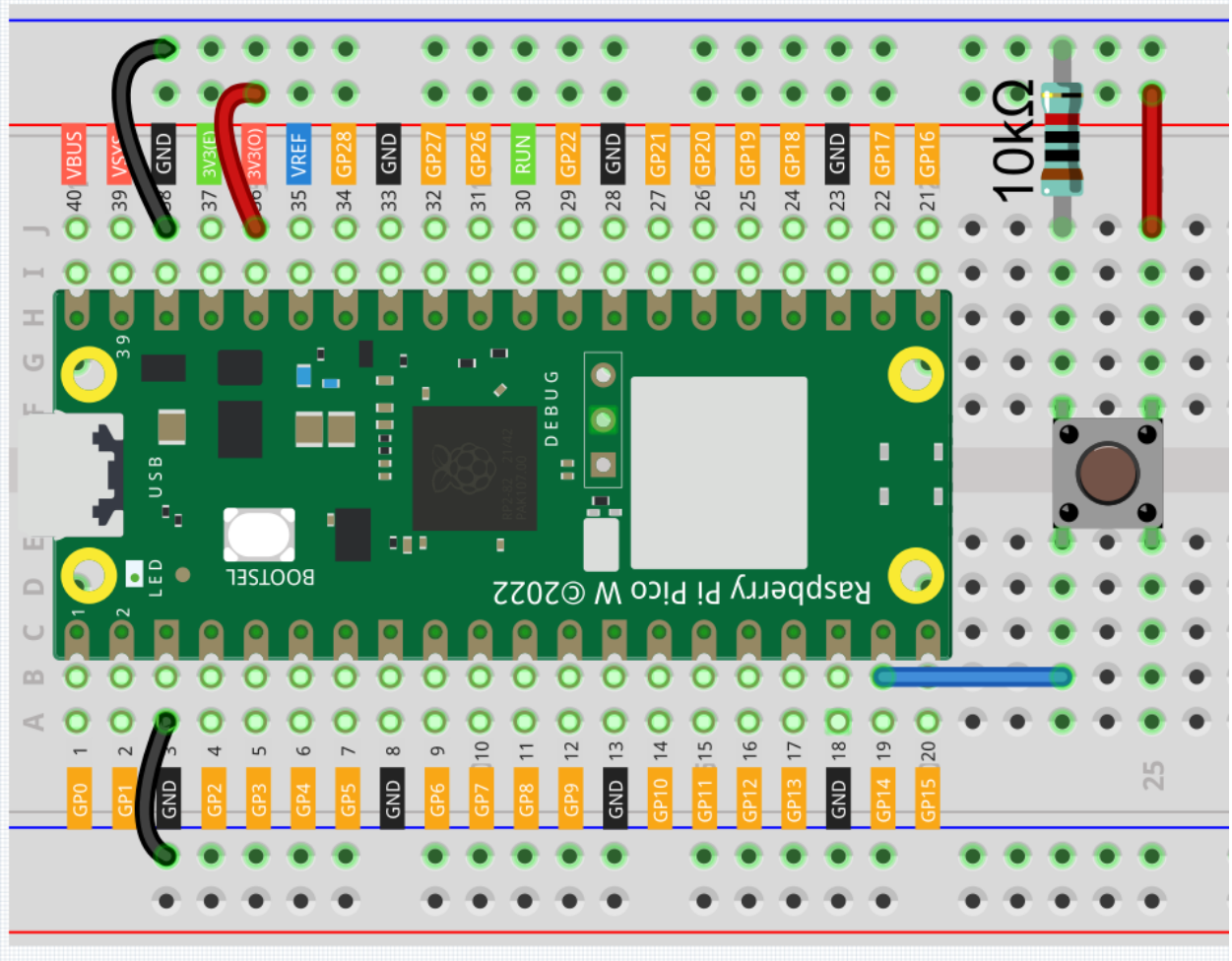
SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Button*	1	

Schematic



As long as one side of the button pin is connected to 3.3v, and the other side pin is connected to GP14, then when the button is pressed, GP14 will be high. However, when the button is not pressed, GP14 is in a suspended state and may be high or low. In order to get a stable low level when the button is not pressed, GP14 needs to be reconnected to GND through a 10K pull-down resistor.

Wiring



Note: A four-pin button is shaped like an H. Its left two pins or right two pins are connected, which means that when it crosses the central gap, it connects two half rows with the same row number. (For example, in my circuit, E23 and F23 are already connected, as are E25 and F25).

Until the button is pressed, the left and right pins are independent of each other and current cannot flow from one side to the other.

Code

Note:

- Open the `2.5_read_button_value.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
import machine
import utime
button = machine.Pin(14, machine.Pin.IN)
```

(continues on next page)

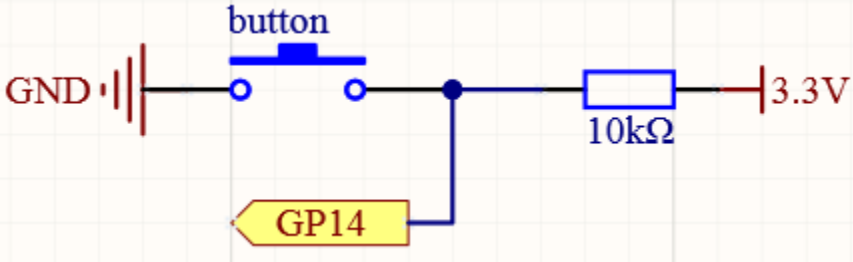
(continued from previous page)

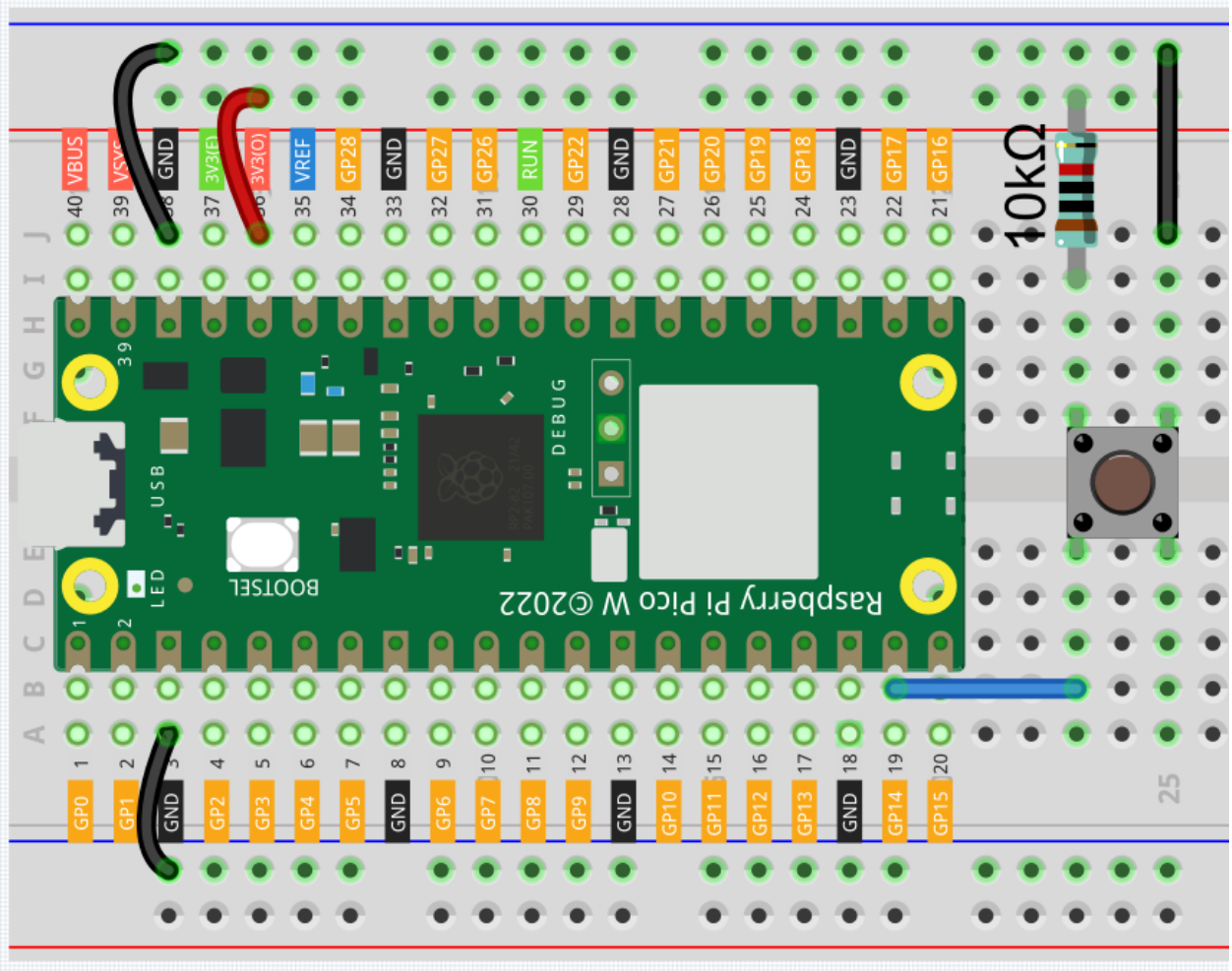
```
while True:
    if button.value() == 1:
        print("You pressed the button!")
        utime.sleep(1)
```

As soon as the code runs, the shell prints "You pressed the button!"

Pull-up Working Mode

The next part is the wiring and code when you use the button in the pull-up mode.





The only difference you will see with the pull-down mode is that the 10K resistor is connected to 3.3V and the button is connected to GND, so that when the button is pressed, GP14 will get a low level, which is the opposite of the value obtained in pull-down mode. So just change this code to `if button.value() == 0:`.

Also see the reference here:

- machine.Pin

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.12 2.6 Tilt It



The tilt switch is a 2-pin device with a metal ball in the middle. When the switch is upright, the two pins are connected; when it is tilted, the two pins are disconnected.

Required Components

In this project, we need the following components.

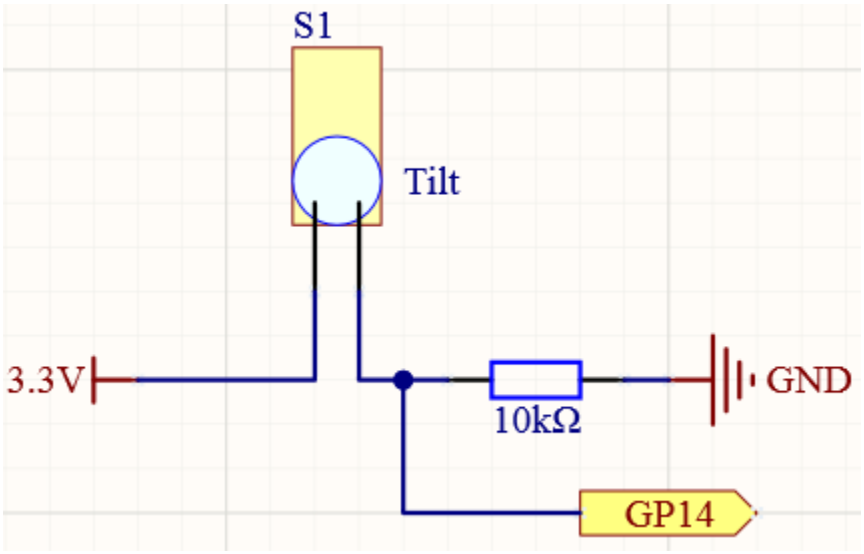
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Tilt Switch*	1	

Schematic

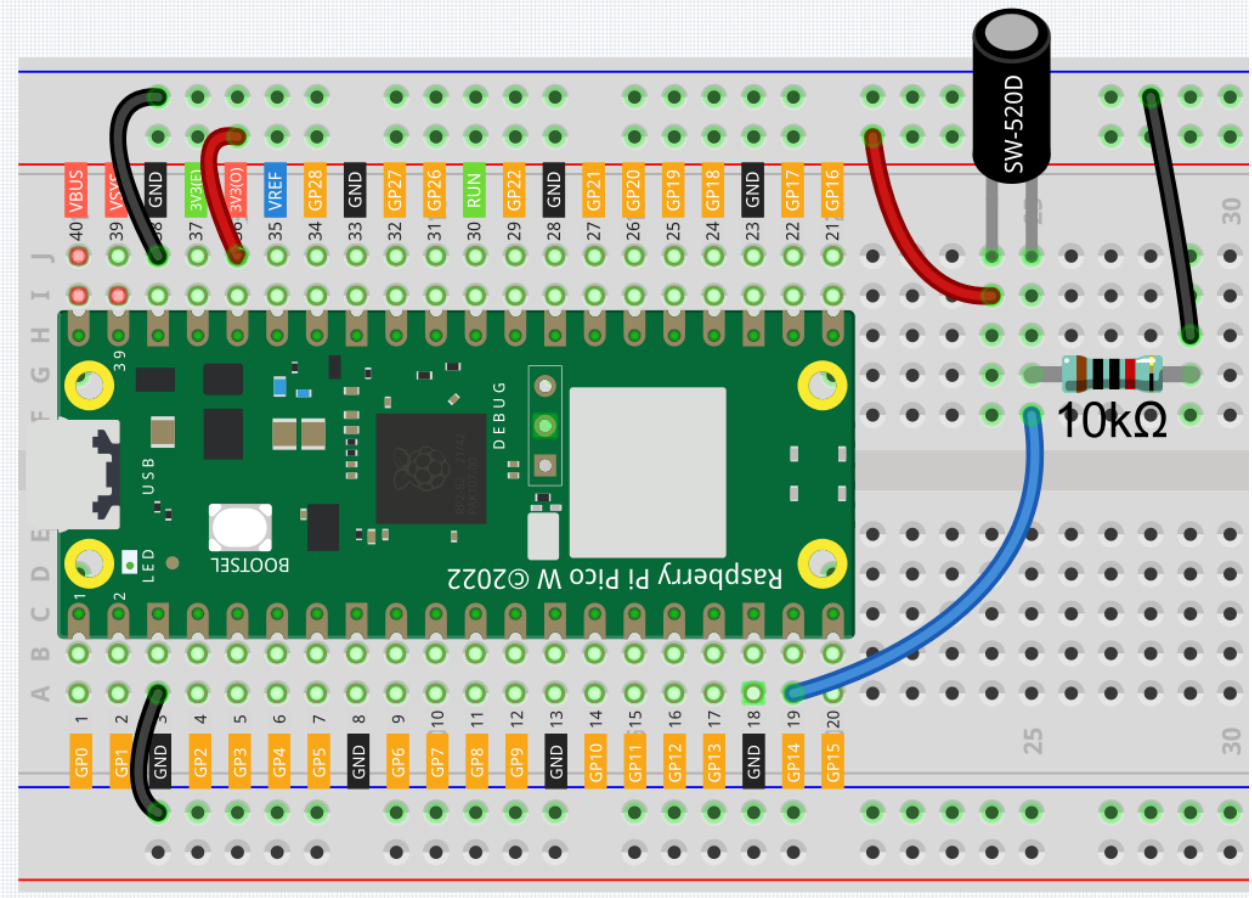


When you put it upright, GP14 will get high; after tilting it, GP14 will get low.

The purpose of the 10K resistor is to keep the GP14 in a stable low state when the tilt switch is in a tilted state.

- *Tilt Switch*

Wiring



Code

Note:

- Open the `2.6_tilt_switch.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

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

After the program runs, when you tilt the breadboard (tilt switch), "The switch works!" will appear in the shell.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.13 2.7 Toggle Left and Right



The slide switch is a 3-pin device, with pin 2 (middle) being the common pin. When the switch is toggled to the left, the left two pins are connected together, and when toggled to the right, the right two pins are connected together.

Required Components

In this project, we need the following components.

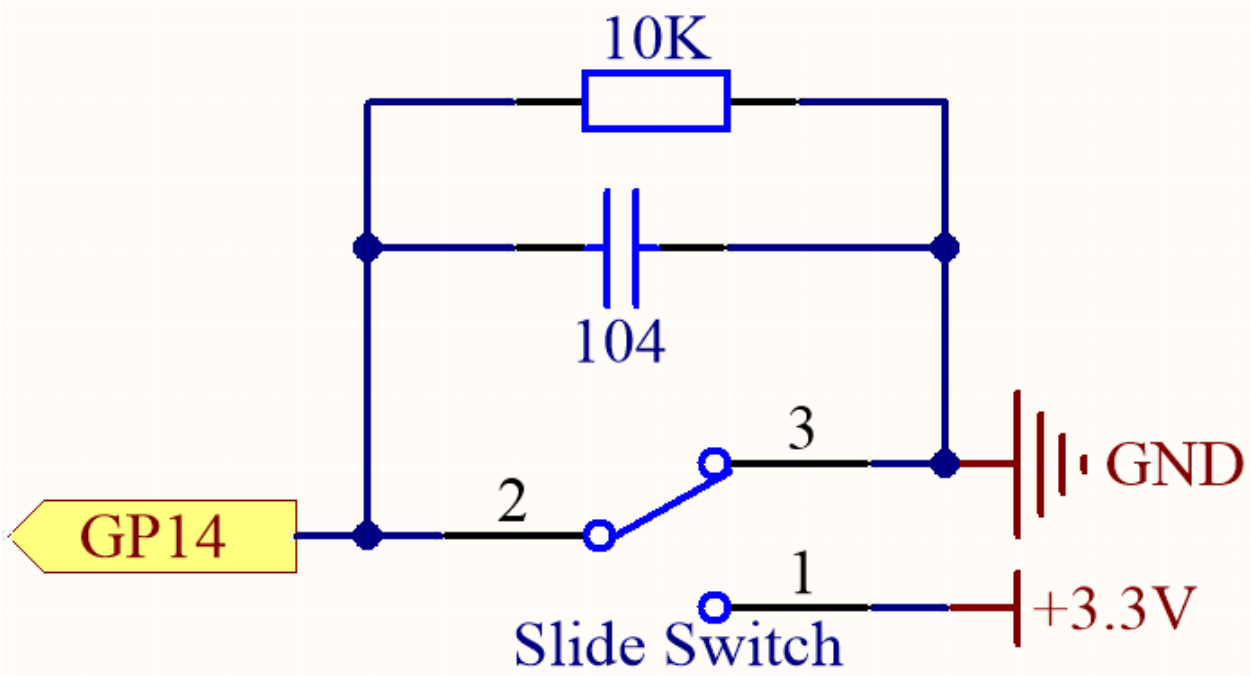
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Capacitor*	1(104)	
7	*Slide Switch*	1	

Schematic



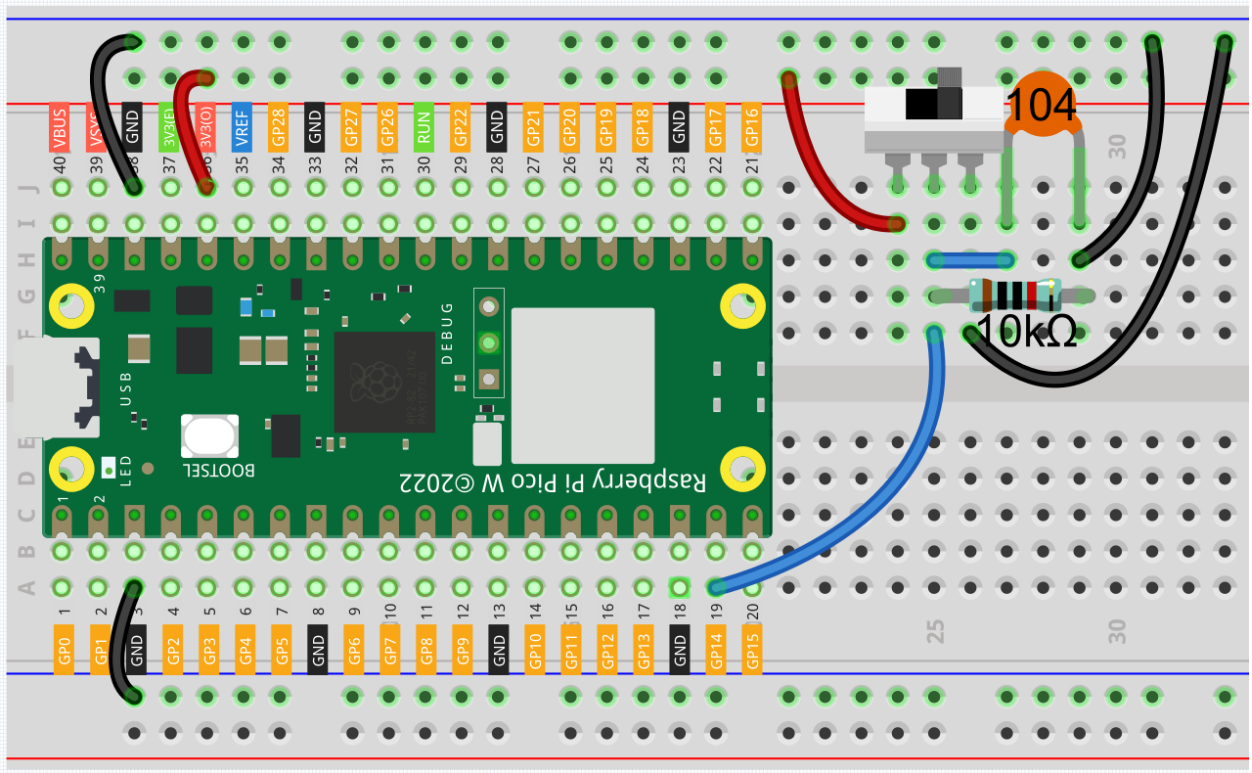
GP14 will get a different level, when you toggle the slide switch to the right or left.

The purpose of the 10K resistor is to keep the GP14 low during toggling (not toggling to the far left and not toggling to the far right).

The 104 ceramic capacitor is used here to eliminate jitter.

- *Slide Switch*
- *Capacitor*

Wiring



Code

Note:

- Open the `2.7_slide_switch.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

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

After the program runs, when you toggle the slide switch to the right, "The switch works!" will appear in the shell.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

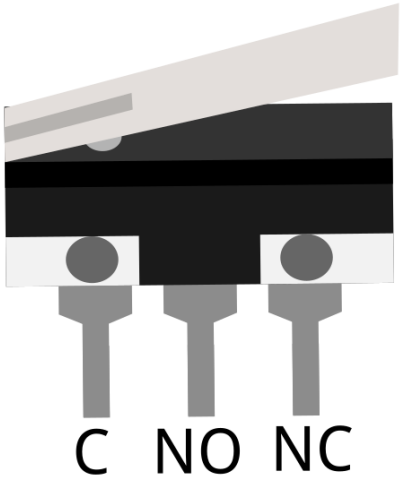
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.14 2.8 Press Gently



Micro Switch is also a 3-pin device, the sequence of the 3 pins are C (common pin), NO (normally open) and NC (normally closed) .

When the micro switch is not pressed, 1 (C) and 3 (NC) are connected together, when pressed 1 (C) and 2 (NO) are connected together.

- *Micro Switch*

Required Components

In this project, we need the following components.

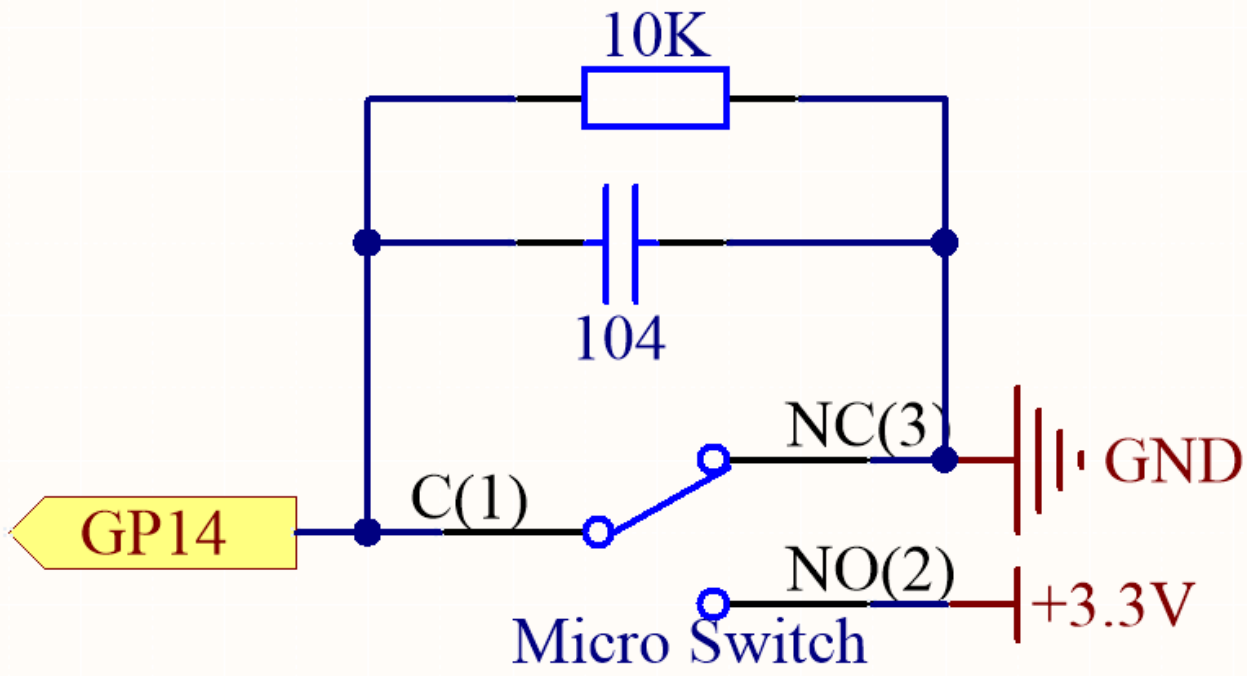
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Capacitor*	1(104)	
7	*Micro Switch*	1	

Schematic

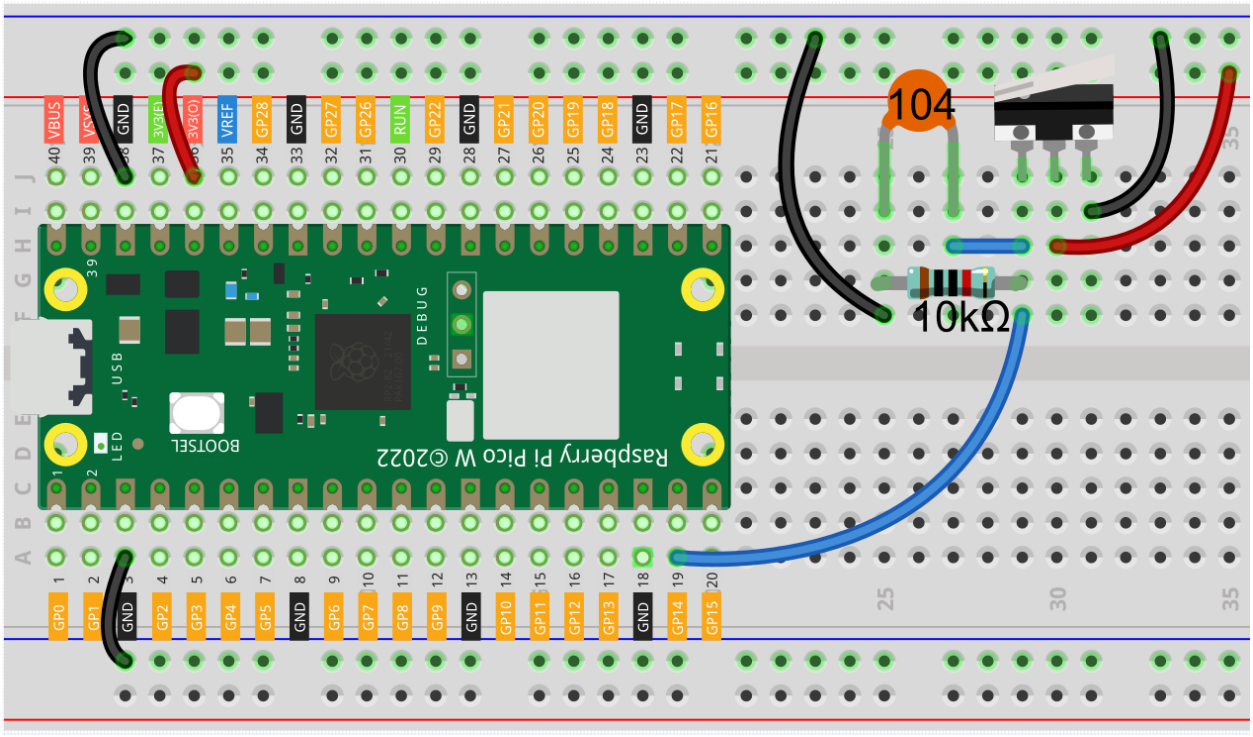


By default, GP14 is low and when pressed, GP14 is high.

The purpose of the 10K resistor is to keep the GP14 low during pressing.

The 104 ceramic capacitor is used here to eliminate jitter.

Wiring



Code

Note:

- Open the `2.8_micro_switch.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

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

After the program runs, when you toggle the slide switch to the right, "The switch works!" will appear in the shell.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.15 2.9 Feel the Magnetism

The most common type of reed switch contains a pair of magnetizable, flexible, metal reeds whose end portions are separated by a small gap when the switch is open.

A magnetic field from an electromagnet or a permanent magnet will cause the reeds to attract each other, thus completing an electrical circuit. The spring force of the reeds causes them to separate, and open the circuit, when the magnetic field ceases.

A common example of a reed switch application is to detect the opening of a door or windows, for a security alarm.

- *Reed Switch*

Required Components

In this project, we need the following components.

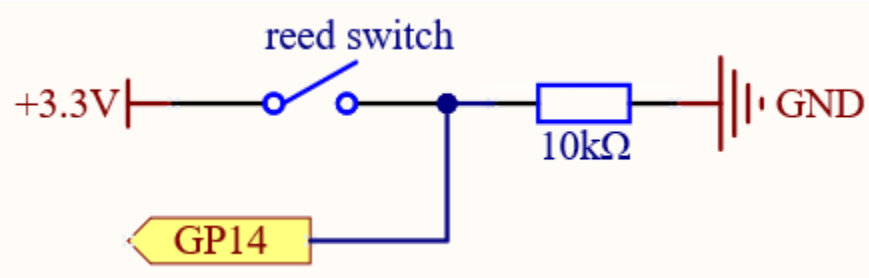
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUANTITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Reed Switch*	1	

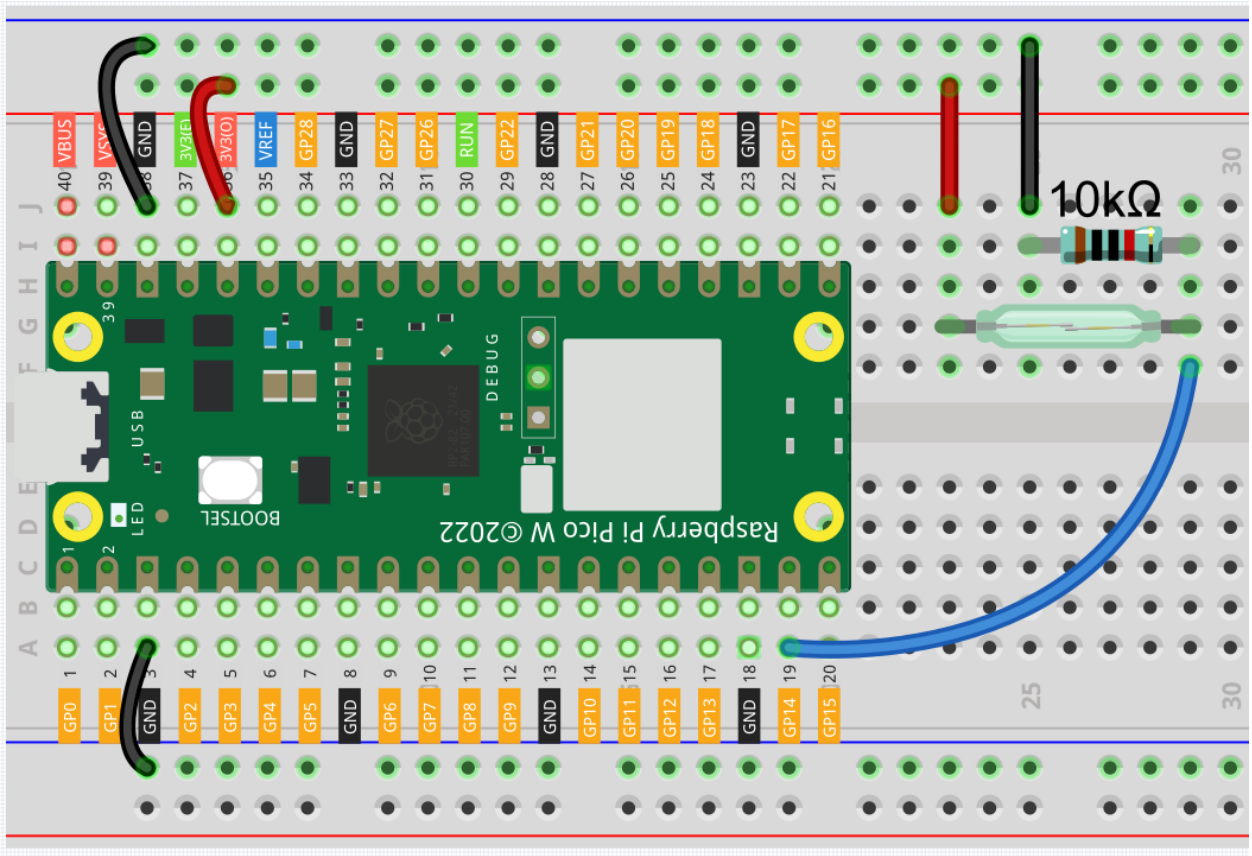
Schematic



By default, GP14 is low; and will go high when the magnet is near the reed switch.

The purpose of the 10K resistor is to keep the GP14 at a steady low level when no magnet is near.

Wiring



Code

Note:

- Open the `2.9_feel_the_magnetism.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)
```

When the code is run, GP14 goes high when a magnet is near the reed switch, otherwise it goes low. Just like the button in the *2.5 Reading Button Value* chapter.

Learn More

This time, we tried a flexible way of using switches: interrupt requests, or IRQs.: interrupt requests, or IRQs.

For example, you are reading a book page by page, as if a program is executing a thread. At this time, someone came to you to ask a question and interrupted your reading. Then the person is executing the interrupt request: asking you to stop what you are doing, answer his questions, and then let you return to reading the book after the end.

MicroPython interrupt request also works in the same way, it allows certain operations to interrupt the main program.

Note:

- Open the `2.9_feel_the_magnetism_irq.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
import machine
import utime

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

def detected(pin):
    print("Magnet!")

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

Here first a callback function `detected(pin)` is defined, called the interrupt handler. It will be executed when an interrupt request is triggered. Then, an interrupt request is set up in the main program, which contains two parts: the `trigger` and the `handler`.

In this program, `trigger` is `IRQ_RISING`, which indicates that the value of the pin is raised from low to high (i.e., button press).

`handler` is `detected` , the callback function we defined before.

- machine.Pin.irq - Micropython Docs

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.16 2.10 Detect Human Movement

Passive infrared sensor (PIR sensor) is a common sensor that can measure infrared (IR) light emitted by objects in its field of view. Simply put, it will receive infrared radiation emitted from the body, thereby detecting the movement of people and other animals. More specifically, it tells the main control board that someone has entered your room.

PIR Motion Sensor Module

Required Components

In this project, we need the following components.

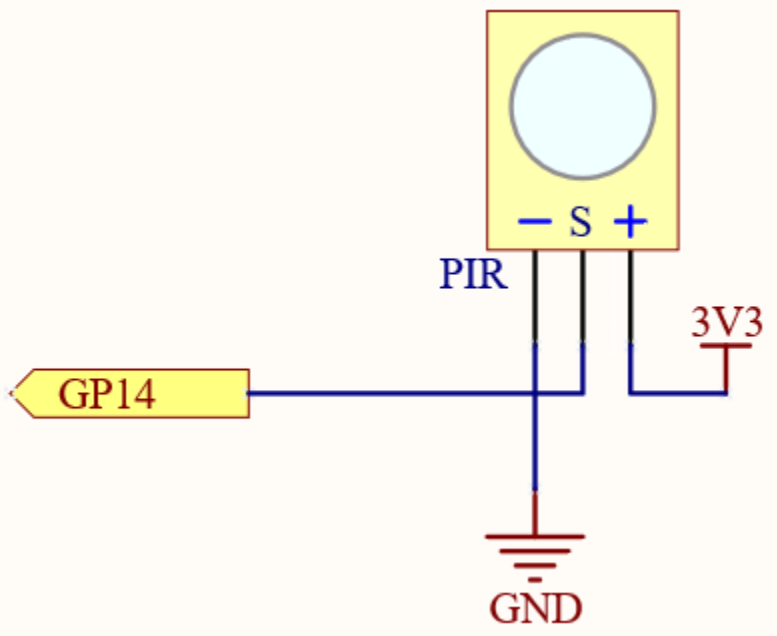
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*PIR Motion Sensor Module*	1	

Schematic

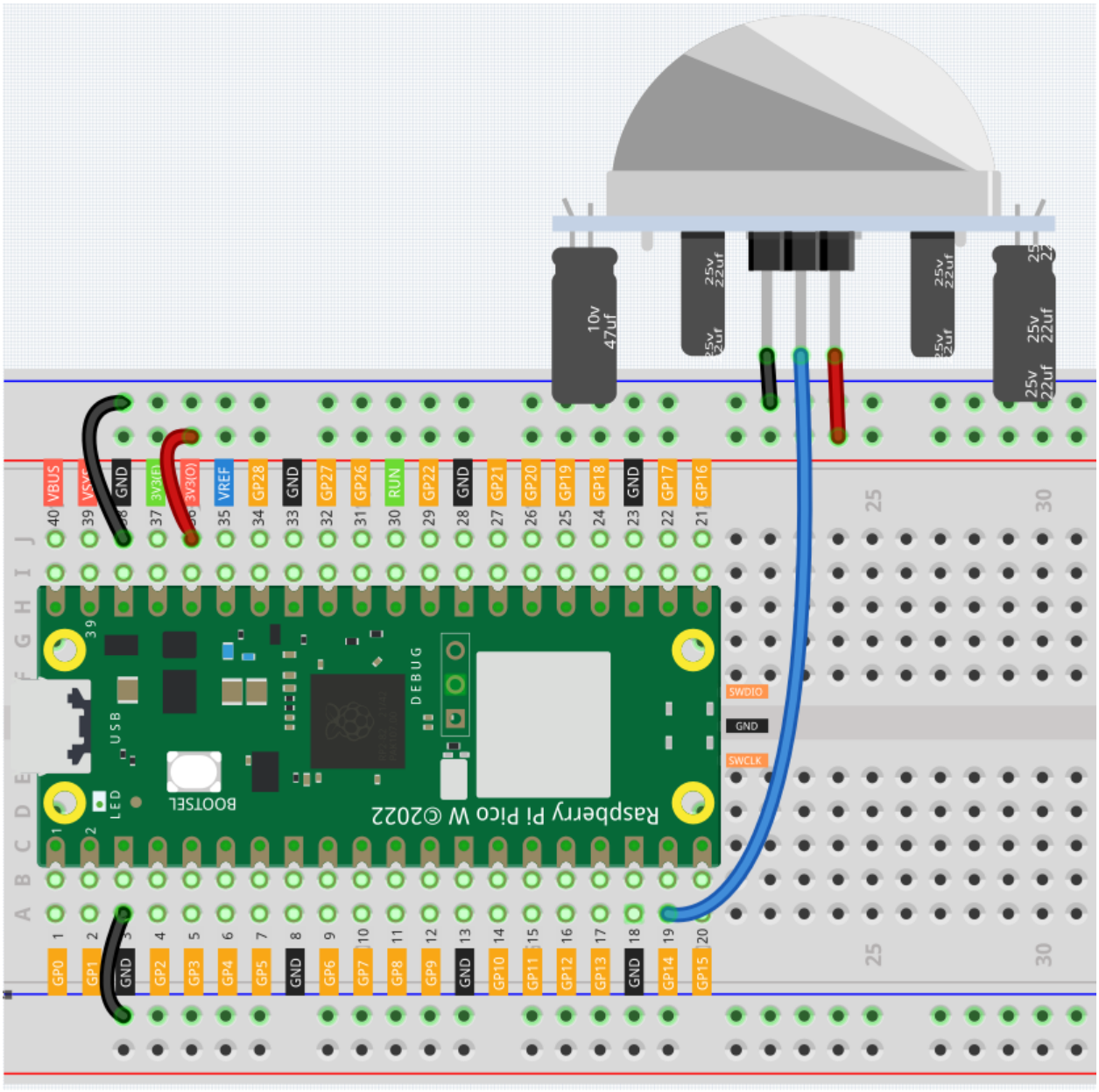


When the PIR module detects someone passing by, GP14 will be high, otherwise it will be low.

Note: The PIR module has two potentiometers: one adjusts sensitivity, the other adjusts detection distance. To make the PIR module work better, you need to turn both of them counterclockwise to the end.



Wiring



Code

Note:

- Open the `2.10_detect_human_movement.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
import machine
import utime
```

(continues on next page)

(continued from previous page)

```
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)
```

After the program runs, if the PIR module detects someone nearby, the Shell will print out "Somebody here!"

Learn More

PIR is a very sensitive sensor. In order to adapt it to the environment of use, it needs to be adjusted. Let the side with the 2 potentiometers facing you, turn both potentiometers counterclockwise to the end and insert the jumper cap on the pin with L and the middle pin.

Note:

- Open the `2.10_pir_adjustment.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
import machine
import utime

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

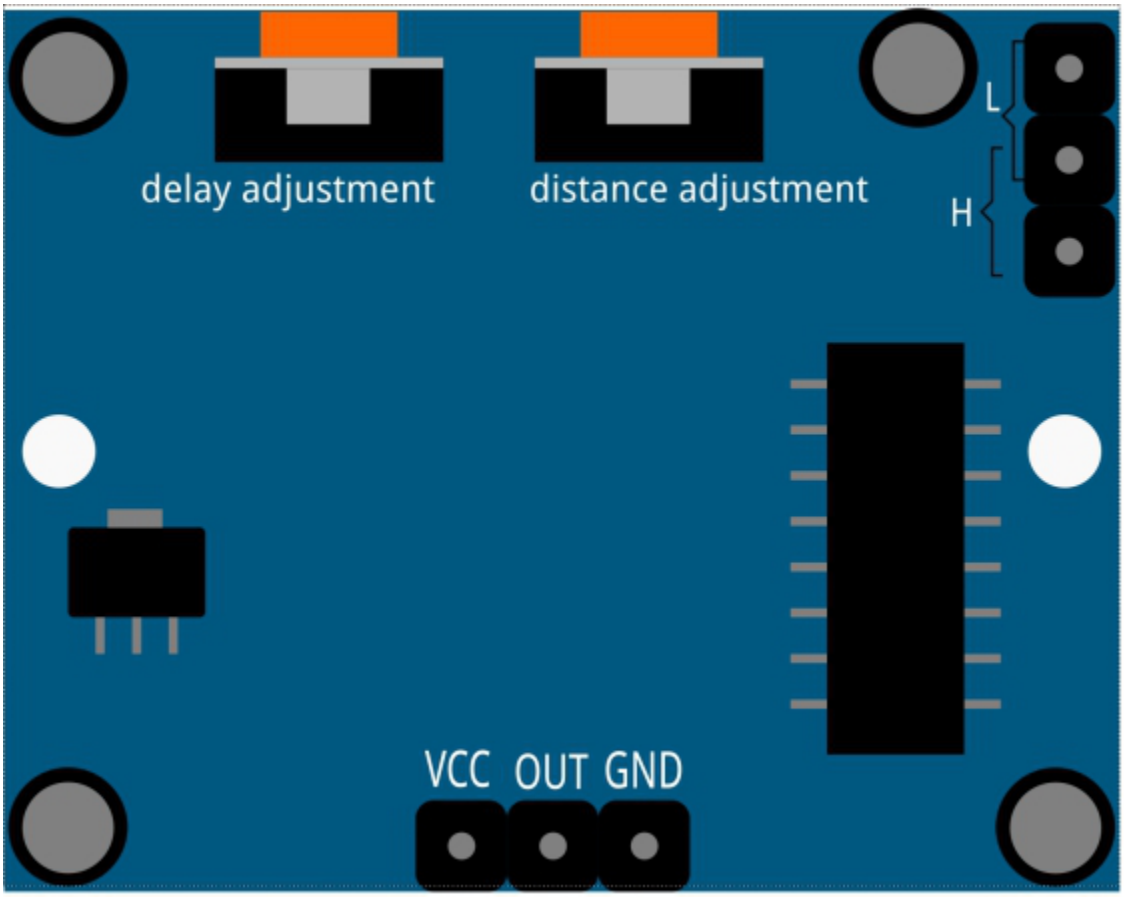
global timer_delay
timer_delay = utime.ticks_ms()
print("start")

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()
    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)
```

Let us analyze its adjustment method along with the experimental results.



1. Trigger Mode

   Let's take a look at the pins with jumper cap at the corner. It allows PIR to enter Repeatable trigger mode or Non-repeatable trigger mode

   At present, our jumper cap connects the middle Pin and L Pin, which makes the PIR in non-repeatable trigger mode. In this mode, when the PIR detects the movement of the organism, it will send a high-level signal for about 2.8 seconds to the main control board. We can see in the printed data that the duration of work will always be around 2800ms.

   Next, we modify the position of the lower jumper cap and connect it to the middle Pin and H Pin to make the PIR in repeatable trigger mode. In this mode, when the PIR detects the movement of the organism (note that it is movement, not static in front of the sensor), as long as the organism keeps moving within the detection range, the PIR will continue to send a high-level signal to the main control board. We can see in the printed data that the duration of work is an uncertain value.

2. Delay Adjustment

   The potentiometer on the left is used to adjust the interval between two jobs.

   At present, we screw it counterclockwise to the end, which makes the PIR need to enter a sleep time of about 5 seconds after finishing sending the high level work. During this time, the PIR will no longer detect the infrared radiation in the target area. We can see in the printed data that the dormancy duration is always no less than 5000ms.

   If we turn the potentiometer clockwise, the sleep time will also increase. When it is turned clockwise to the end, the sleep time will be as high as 300s.

3. Distance Adjustment

The centered potentiometer is used to adjust the sensing distance range of the PIR.

Turn the knob of the distance adjustment potentiometer **clockwise** to increase the sensing distance range, and the maximum sensing distance range is about 0-7 meters. If it rotates **counterclockwise**, the sensing distance range is reduced, and the minimum sensing distance range is about 0-3 meters.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

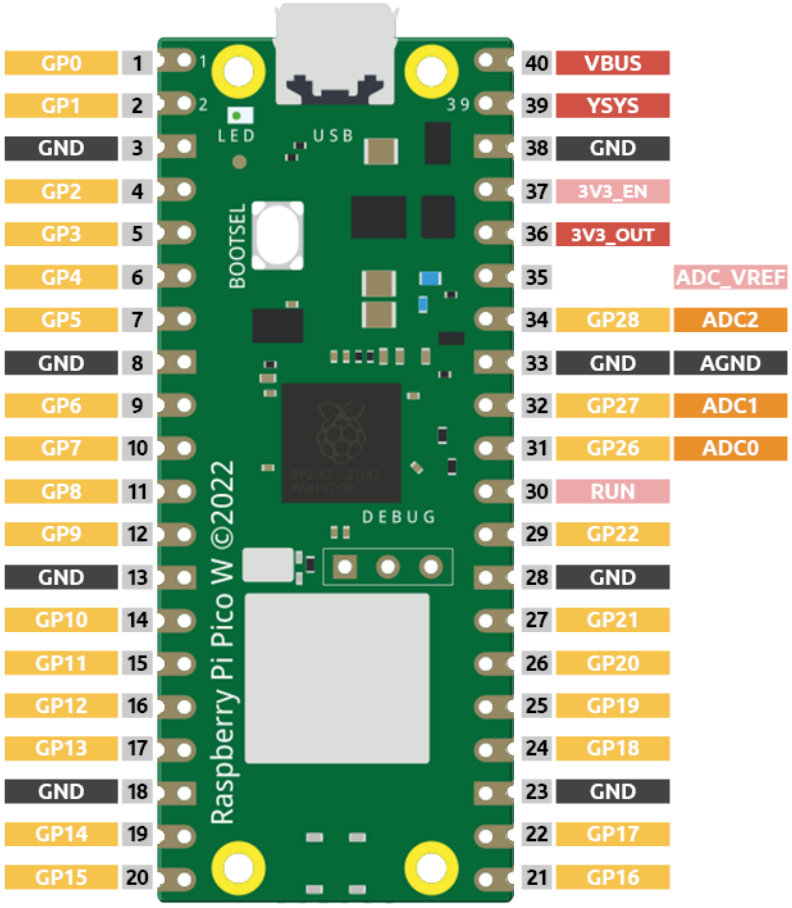
Ready to explore and create with us? Click [] and join today!

4.17 2.11 Turn the Knob

In the previous projects, we have used the digital input on the Pico W. For example, a button can change the pin from low level (off) to high level (on). This is a binary working state.

However, Pico W can receive another type of input signal: analog input. It can be in any state from fully closed to fully open, and has a range of possible values. The analog input allows the microcontroller to sense the light intensity, sound intensity, temperature, humidity, etc. of the physical world.

Usually, a microcontroller needs an additional hardware to implement analog input-the analogue-to-digital converter (ADC). But Pico W itself has a built-in ADC for us to use directly.



Pico W has three GPIO pins that can use analog input, GP26, GP27, GP28. That is, analog channels 0, 1, and 2. In addition, there is a fourth analog channel, which is connected to the built-in temperature sensor and will not be introduced here.

In this project, we try to read the analog value of potentiometer.

- *Potentiometer*

Required Components

In this project, we need the following components.

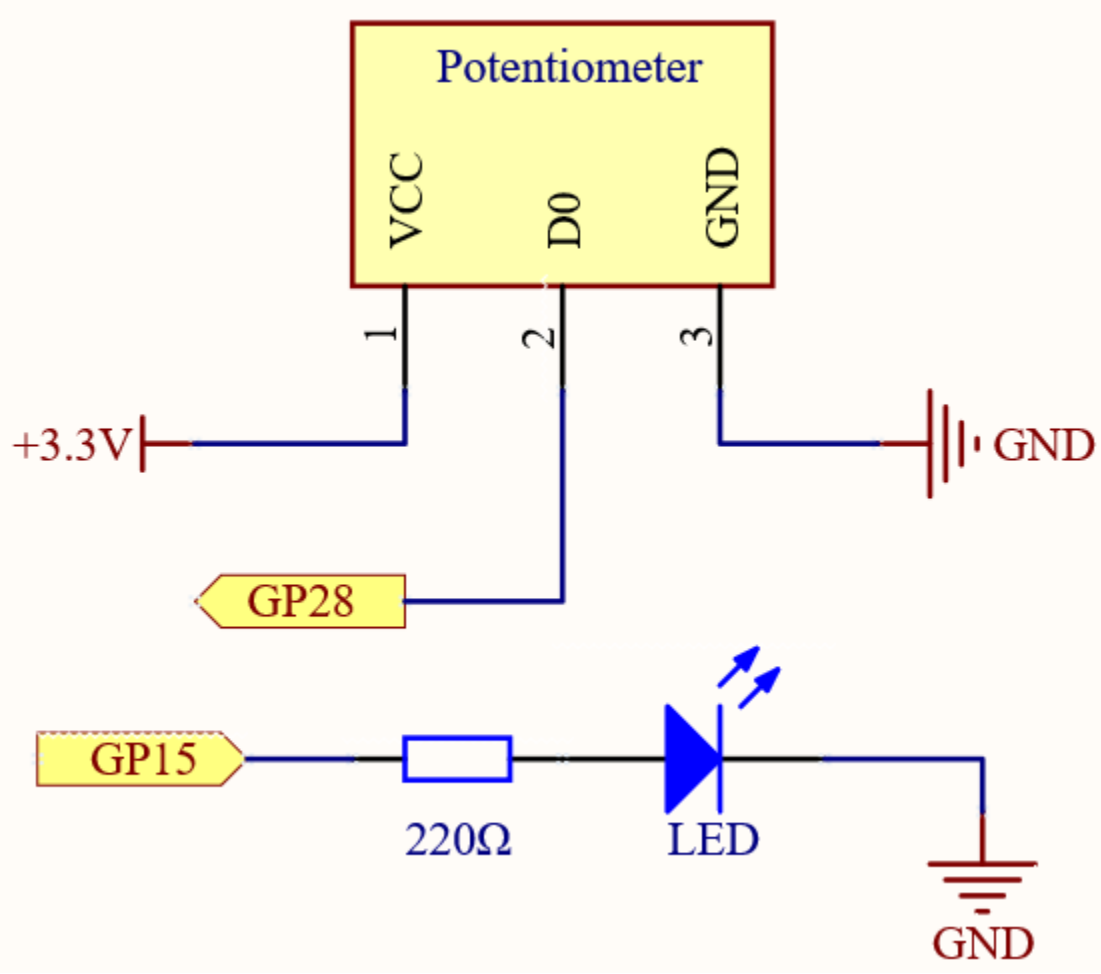
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(220)	
6	*LED*	1	
7	*Potentiometer*	1	

Schematic



The potentiometer is an analog device and when you turn it in 2 different directions.

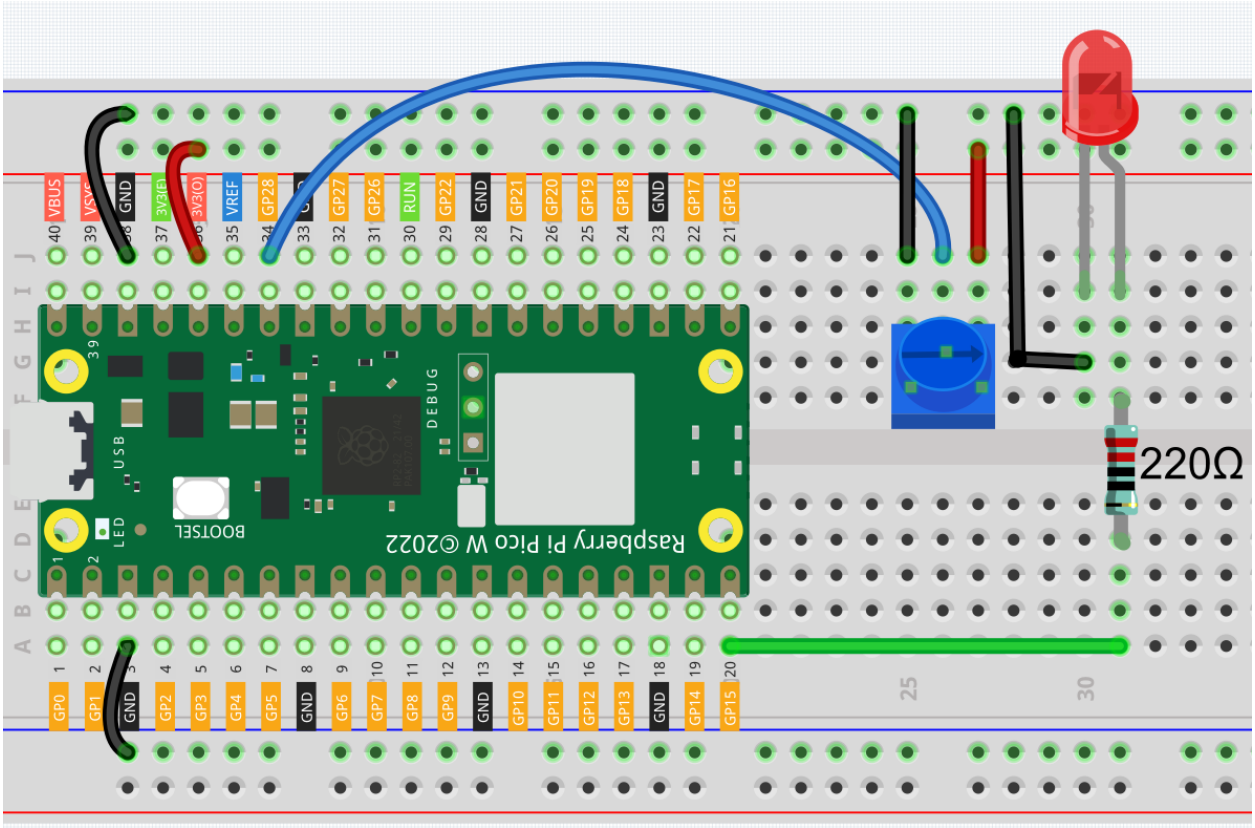
Connect the middle pin of the potentiometer to the analog pin GP28. The Raspberry Pi Pico W wcontains a multi-channel, 16-bit analog-to-digital converter. This means that it maps the input voltage between 0 and the operating voltage (3.3V) to an integer value between 0 and 65535, so the GP28 value ranges from 0 to 65535.

The calculation formula is shown below.

(Vp/3.3V) x 65535 = Ap

Then program the value of GP28 (potentiometer) as the PWM value of GP15 (LED). This way you will find that by rotating the potentiometer, the brightness of the LED will change at the same time.

Wiring



Code

Note:

- Open the `2.11_turn_the_knob.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
import machine
import utime

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

while True:
    value=potentiometer.read_u16()
```

(continues on next page)

(continued from previous page)

```
    print(value)
    led.duty_u16(value)
    utime.sleep_ms(200)
```

When the program is running, we can see the analog value currently read by the GP28 pin in the shell. Turn the knob, and the value will change from 0 to 65535. At the same time, the brightness of the LED will increase as the analog value increases.

How it works?

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

Access the ADC associated with a source identified by id. In this example it is GP28.

```
potentiometer.read_u16()
```

Take an analog reading and return an integer in the range 0-65535. The return value represents the raw reading taken by the ADC, scaled such that the minimum value is 0 and the maximum value is 65535.

- machine.ADC - MicroPython Docs

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.18 2.12 Feel the Light

The photoresistor is a typical device for analog inputs and it is used in a very similar way to a potentiometer. Its resistance value depends on the intensity of the light, the stronger the irradiated light, the smaller its resistance value; conversely, it increases.

- *Photoresistor*

Required Components

In this project, we need the following components.

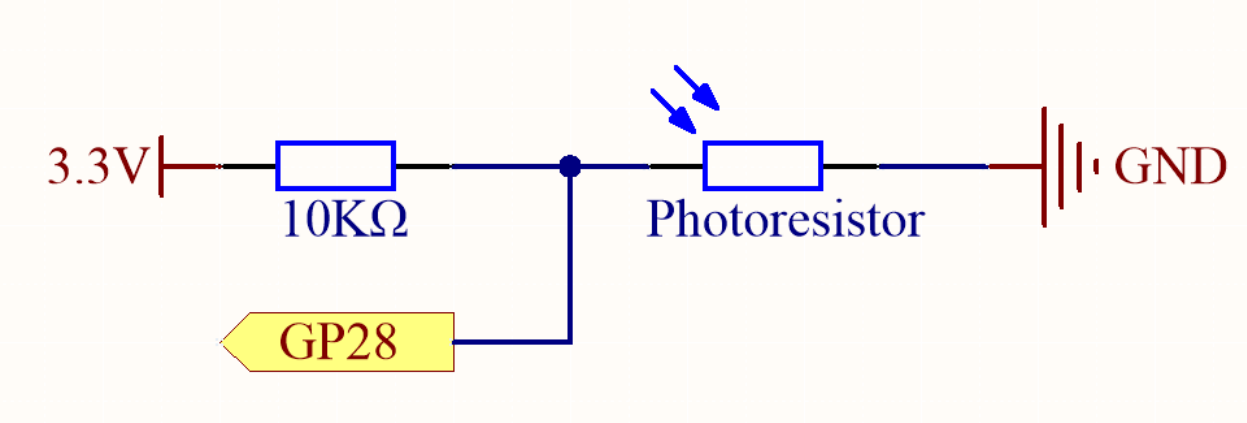
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Photoresistor*	1	

Schematic



In this circuit, the 10K resistor and the photoresistor are connected in series, and the current passing through them is the same. The 10K resistor acts as a protection, and the GP28 reads the value after the voltage conversion of the photoresistor.

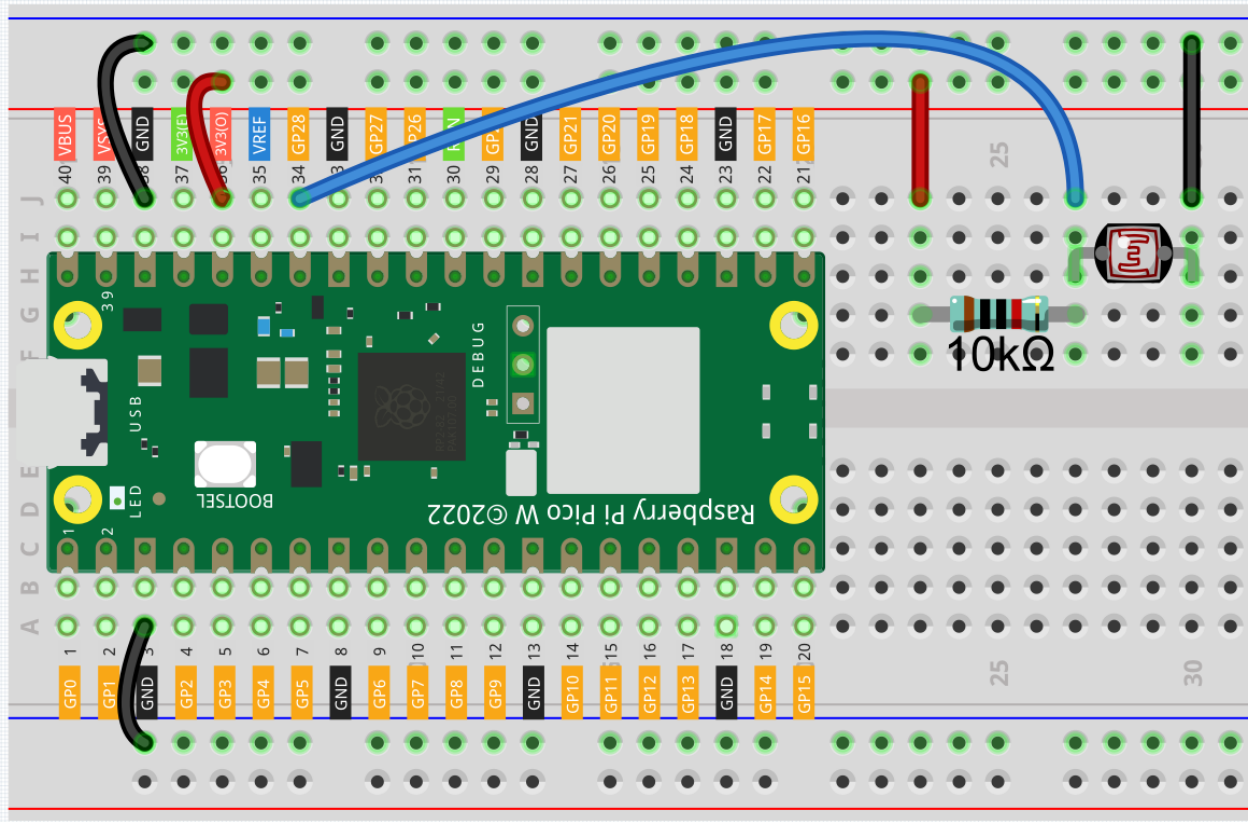
When the light is enhanced, the resistance of the photoresistor decreases, then its voltage decreases, so the value from GP28 will decrease; if the light is strong enough, the resistance of the photoresistor will be close to 0, and the value of GP28 will be close to 0. At this time, the 10K resistor plays a protective role, so that 3.3V and GND are not connected together, resulting in a short circuit.

If you place the photoresistor in a dark situation, the value of GP28 will increase. In a dark enough situation, the resistance of the photoresistor will be infinite, and its voltage will be close to 3.3v (the 10K resistor is negligible), and the value of GP28 will be close to the maximum value of 65535.

The calculation formula is shown below.

(Vp/3.3V) x 65535 = Ap

Wiring



Code

Note:

- Open the `2.12_feel_the_light.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
import machine
import utime

photoresistor = machine.ADC(28)

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

After the program runs, the Shell prints out the photoresistor values. You can shine a flashlight on it or cover it up with your hand to see how the value will change.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.19 2.13 Thermometer

A thermometer is a device that measures temperature or a temperature gradient (the degree of hotness or coldness of an object). A thermometer has two important elements: (1) a temperature sensor (e.g. the bulb of a mercury-in-glass thermometer or the pyrometric sensor in an infrared thermometer) in which some change occurs with a change in temperature; and (2) some means of converting this change into a numerical value (e.g. the visible scale that is marked on a mercury-in-glass thermometer or the digital readout on an infrared model). Thermometers are widely used in technology and industry to monitor processes, in meteorology, in medicine, and in scientific research.

A thermistor is a type of temperature sensor whose resistance is strongly dependent on temperature, and it has two types: Negative Temperature Coefficient (NTC) and Positive Temperature Coefficient (PTC), also known as NTC and PTC. The resistance of PTC thermistor increases with temperature, while the condition of NTC is opposite to the former.

In this experiment we use an **NTC thermistor** to make a thermometer.

- *Thermistor*

Required Components

In this project, we need the following components.

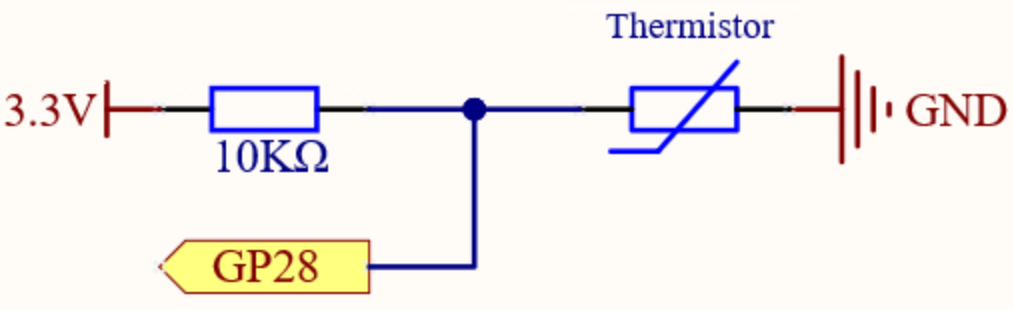
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUANTITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Thermistor*	1	

Schematic



In this circuit, the 10K resistor and the thermistor are connected in series, and the current passing through them is the same. The 10K resistor acts as a protection, and the GP28 reads the value after the voltage conversion of the thermistor.

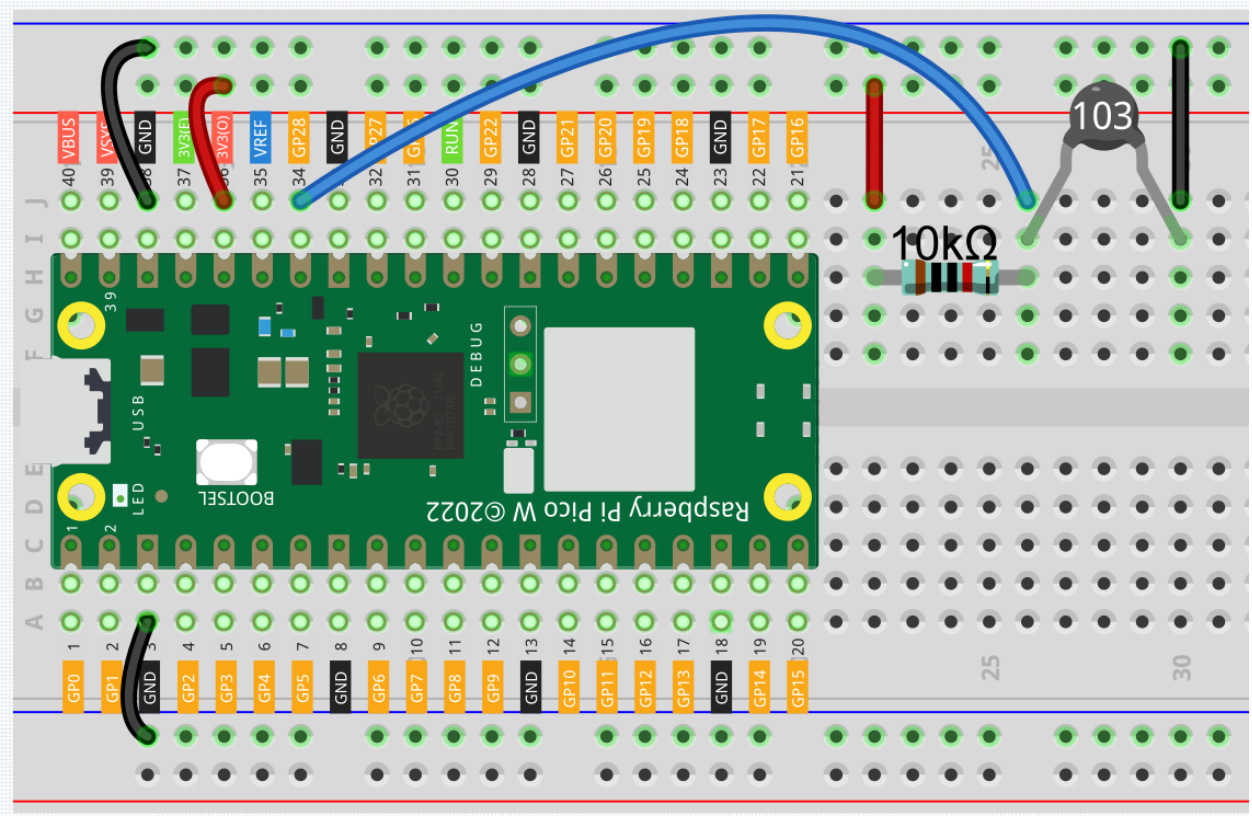
When the temperature increases, the resistance value of NTC thermistor decreases, then its voltage decreases, so the value from GP28 will decrease; If the temperature is high enough, the resistance of the thermistor will be close to 0, and the value of GP28 will be close to 0. At this time, the 10K resistor plays a protective role, so that 3.3V and GND are not connected together, resulting in a short circuit.

When the temperature drops, the value of GP28 will increase. When the temperature is low enough, the resistance of the thermistor will be infinite, and its voltage will be close to 3.3v (the 10K resistor is negligible), and the value of GP28 will be close to the maximum value of 65535.

The calculation formula is shown below.

(Vp/3.3V) x 65535 = Ap

Wiring



Note:

- The thermistor is black and marked 103.
- The color ring of the 10K ohm resistor is red, black, black, red and brown.

Code

Note:

- Open the `2.13_thermometer.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
import machine
import utime
import math

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)
```

After the program runs, the Shell will print out the Celsius and Fahrenheit temperatures.

How it works?

Each thermistor has a normal resistance. Here it is 10k ohm, which is measured under 25 degree Celsius.

When the temperature gets higher, the resistance of the thermistor decreases. Then the voltage data is converted to digital quantities by the A/D adapter.

The temperature in Celsius or Fahrenheit is output via programming.

```
import math
```

There is a numerics library which declares a set of functions to compute common mathematical operations and transformations.

- math

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

This function is used to read the value of the thermistor.

```
Vr = 3.3 * float(temperature_value) / 65535
Rt = 10000 * Vr / (3.3 - Vr)
temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
```

(continues on next page)

(continued from previous page)

```
Cel = temp - 273.15
Fah = Cel * 1.8 + 32
print ('Celsius: %.2f C  Fahrenheit: %.2f F' % (Cel, Fah))
utime.sleep_ms(200)
```

These calculations convert the thermistor values into centigrade degree and Fahrenheit degree.

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

In the two lines of code above, the voltage is first calculated using the read analoge value, and then get Rt (the resistance of the thermistor).

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

Note: Here is the relation between the resistance and temperature:

RT =RN expB(1/TK – 1/TN)

- RT is the resistance of the NTC thermistor when the temperature is TK.
- RN is the resistance of the NTC thermistor under the rated temperature TN. Here, the numerical value of RN is 10k.
- TK is a Kelvin temperature and the unit is K. Here, the numerical value of TK is 273.15 + degree Celsius.
- TN is a rated Kelvin temperature; the unit is K too. Here, the numerical value of TN is 273.15+25.
- And B(beta), the material constant of NTC thermistor, is also called heat sensitivity index with a numerical value 3950.
- exp is the abbreviation of exponential, and the base number e is a natural number and equals 2.7 approximately.

Convert this formula TK=1/(ln(RT/RN)/B+1/TN) to get Kelvin temperature that minus 273.15 equals degree Celsius.

This relation is an empirical formula. It is accurate only when the temperature and resistance are within the effective range.

This code refers to plugging Rt into the formula TK=1/(ln(RT/RN)/B+1/TN) to get Kelvin temperature.

```
temp = temp - 273.15
```

Convert Kelvin temperature into centigrade degree.

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

Convert the centigrade degree into Fahrenheit degree.

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

Print centigrade degree, Fahrenheit degree and their units in the shell.

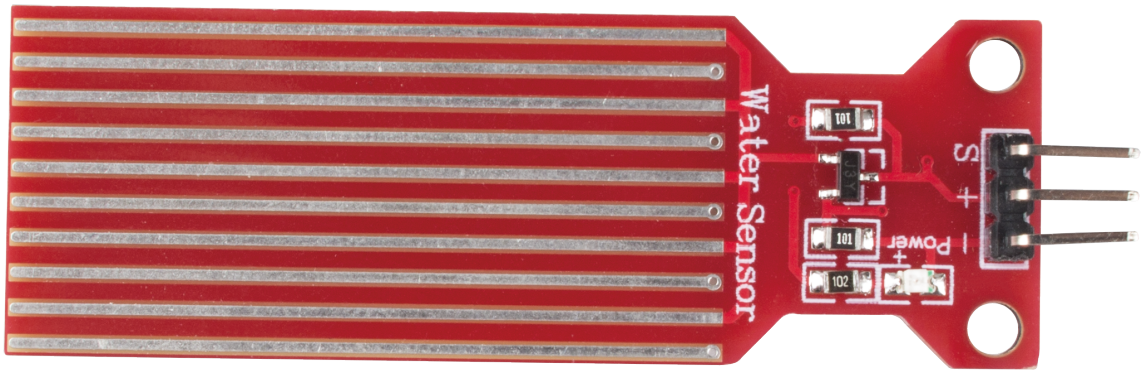
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.20 2.14 Feel the Water Level



Water sensor is designed for water detection, which can be widely used in sensing rainfall, water level, and even liquid leakage.

It measures the water level by having a series of exposed parallel wire traces to measure the size of the water drops/volume. The water volume is easily converted to an analog signal, and the output analog value can be read directly by the main control board to achieve the water level alarm effect.

Warning: The sensor cannot be fully submerged in water, please only leave the part where the ten Traces are located in contact with water. Also, energizing the sensor in a humid environment will accelerate the corrosion of the probe and reduce the life of the sensor, so it is recommended that you only supply power when taking readings.

- *Water Level Sensor Module*

Required Components

In this project, we need the following components.

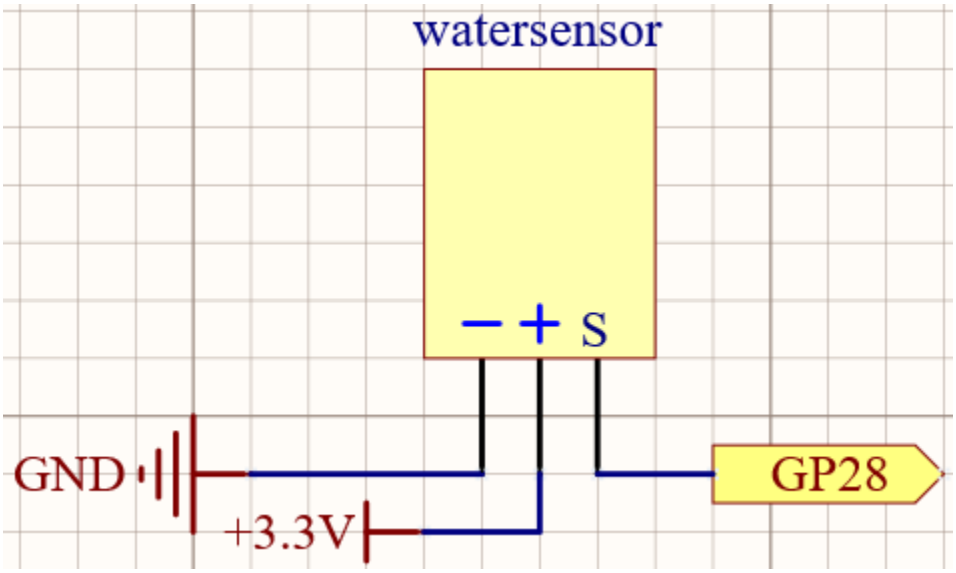
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

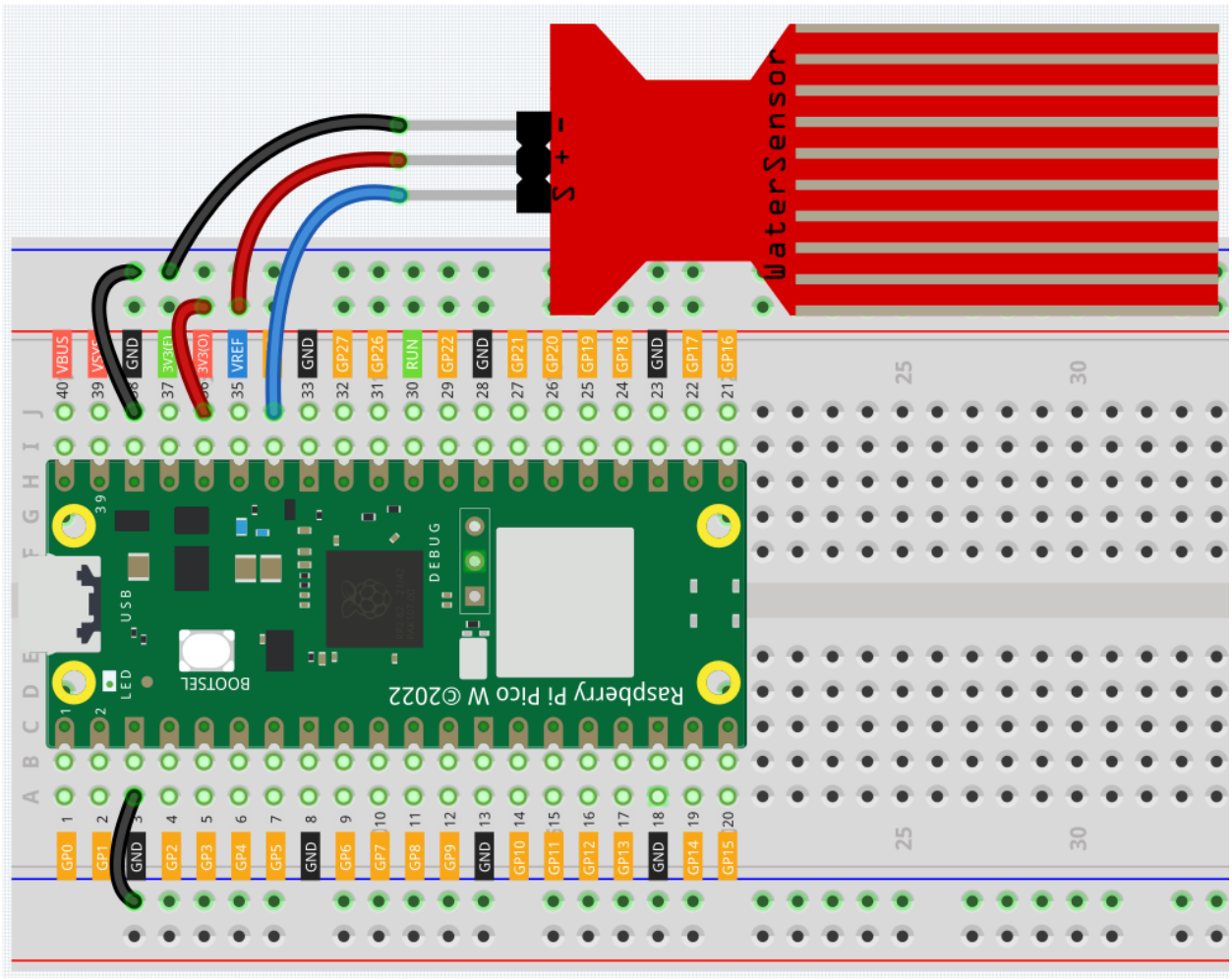
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Water Level Sensor Module*	1	

Schematic



Wiring



Code

Note:

- Open the `2.14_feel_the_water_level.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
import machine
import utime

sensor = machine.ADC(28)

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

After the program is run, submerge the Water Sensor module slowly into the water, and as the depth increases, the Shell

will print a larger value.

Learn More

There is a way to use the analog input module as a digital module.

First, take a reading of the Water Sensor in a dry environment first, record it, and use it as a threshold value. Then, complete the programming and re-read the reading of the water sensor. When the reading of the water sensor deviates significantly from the reading in a dry environment, it is exposed to liquid. In other words, by placing this device near a water pipe, it can detect if a water pipe is leaking.

Note:

- Open the `2.14_water_level_threshold.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
import machine
import utime

sensor = machine.ADC(28)
threshold = 30000 #This value needs to be modified with the environment.

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

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

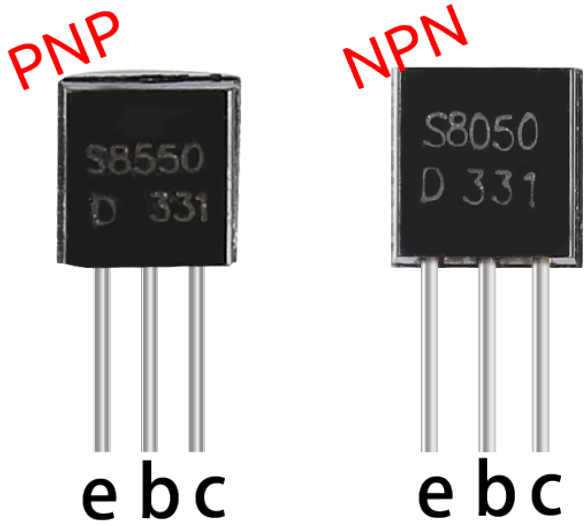
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.21 2.15 Two Kinds of Transistors

This kit is equipped with two types of transistors, S8550 and S8050, the former is PNP and the latter is NPN. They look very similar, and we need to check carefully to see their labels. When a High level signal goes through an NPN transistor, it is energized. But a PNP one needs a Low level signal to manage it. Both types of transistor are frequently used for contactless switches, just like in this experiment.



Let's use LED and button to understand how to use transistor!

Transistor

Required Components

In this project, we need the following components.

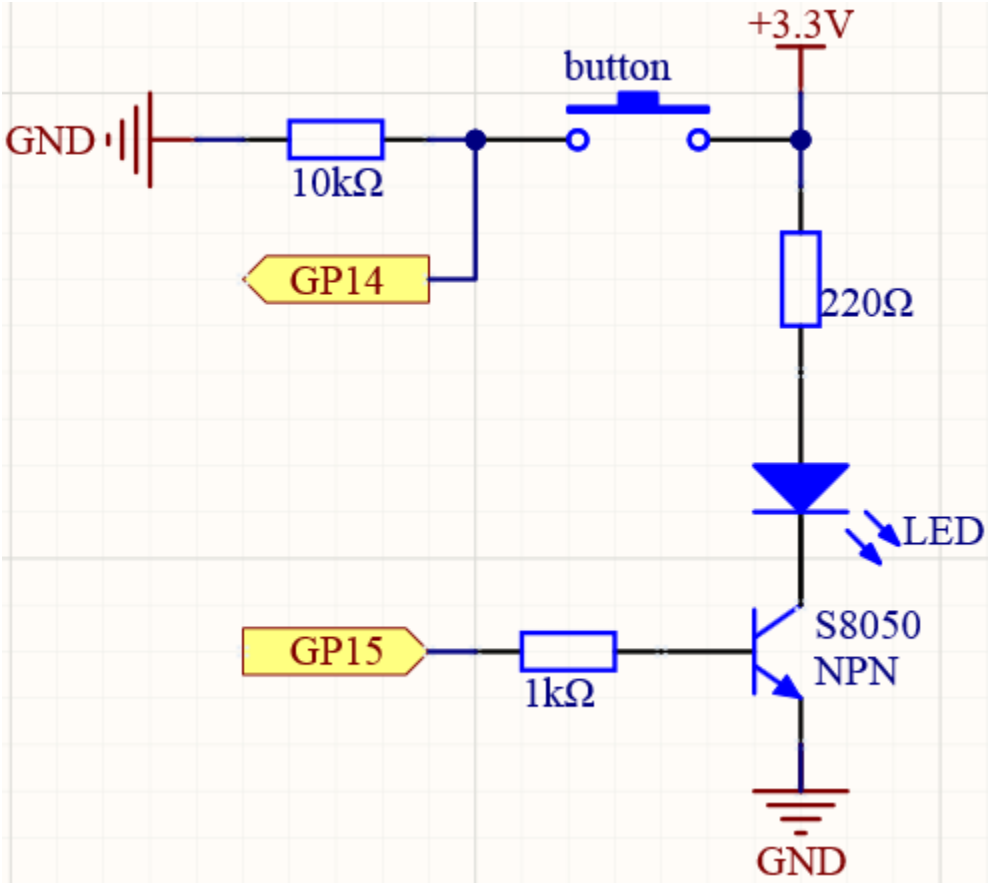
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

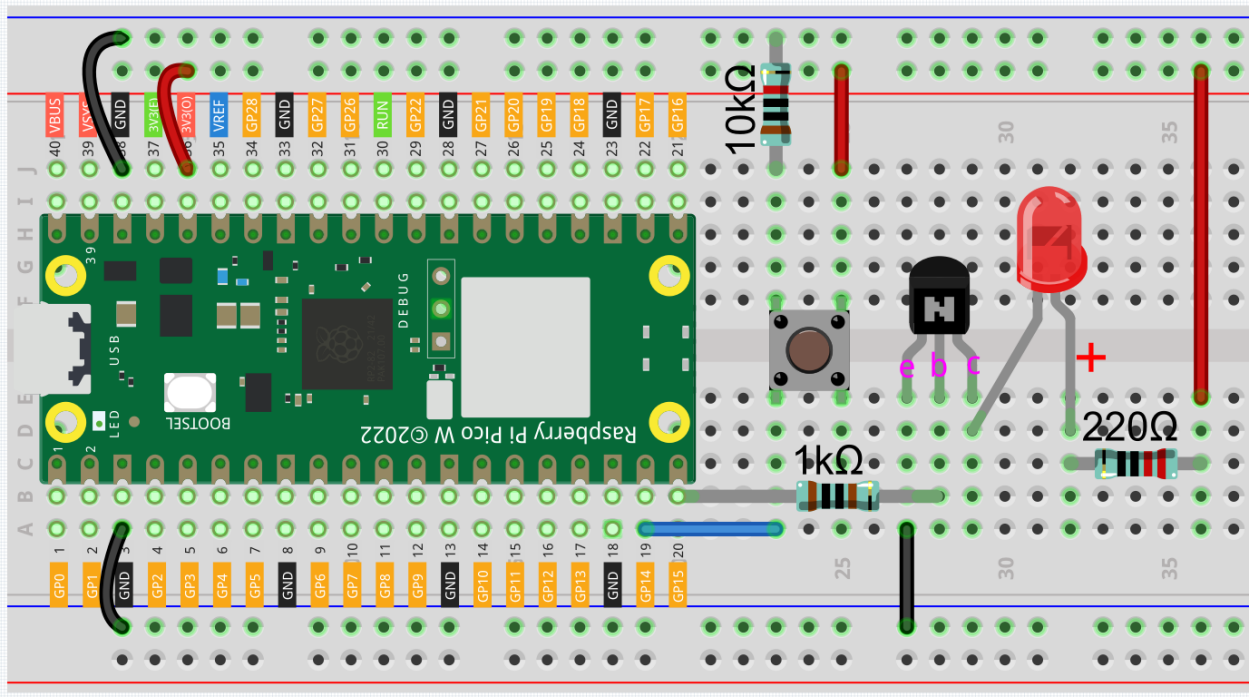
SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	3(220, 1K, 10K)	
6	*LED*	1	
7	*Button*	1	
8	*Transistor*	1(S8050/S8550)	

Way to connect NPN (S8050) transistor

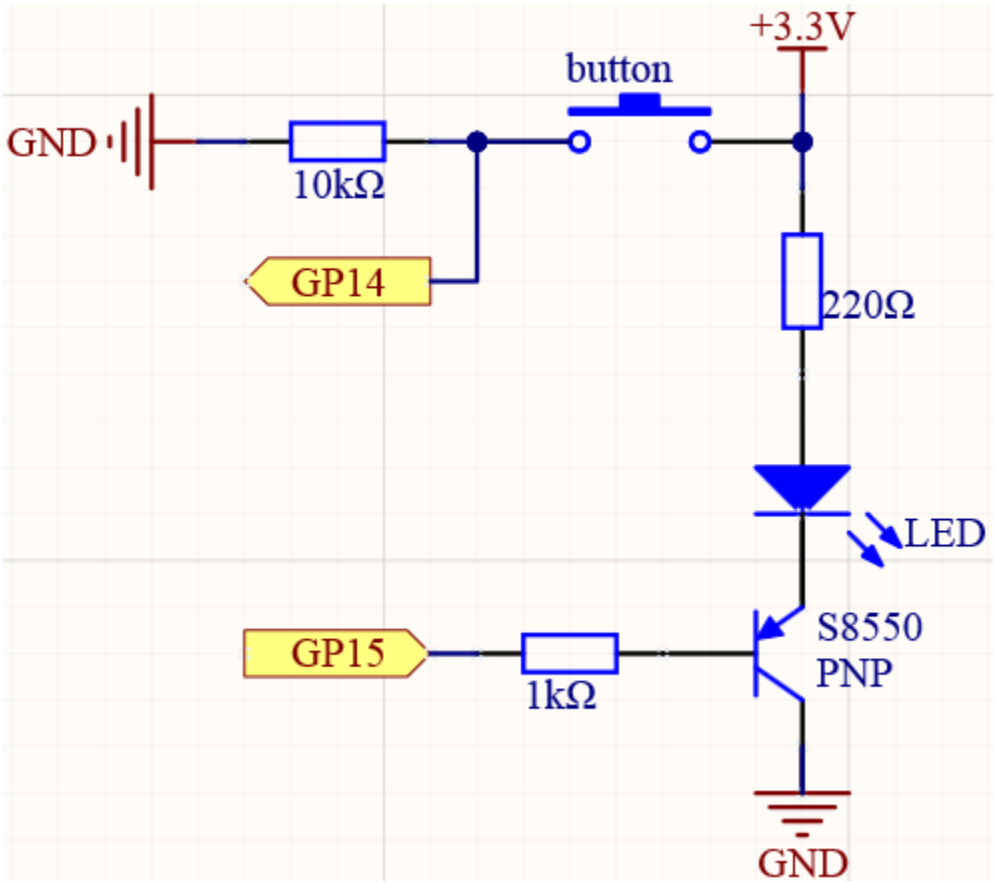


In this circuit, when the button is pressed, GP14 is high.

By programming GP15 to output high, after a 1k current limiting resistor (to protect the transistor), the S8050 (NPN transistor) is allowed to conduct, thus allowing the LED to light up.



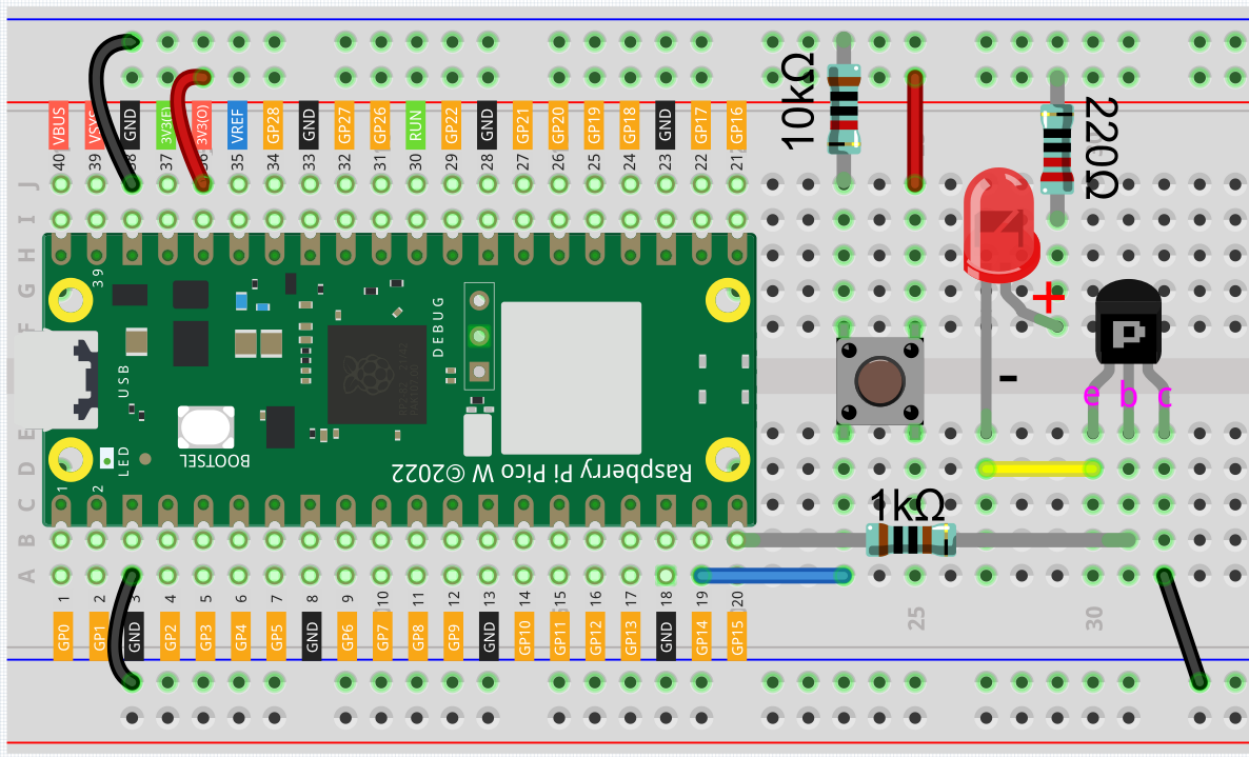
Way to connect PNP(S8550) transistor



In this circuit, GP14 is low by the default and will change to high when the button is pressed.

By programming GP15 to output **low**, after a 1k current limiting resistor (to protect the transistor), the S8550 (PNP transistor) is allowed to conduct, thus allowing the LED to light up.

The only difference you will notice between this circuit and the previous one is that in the previous circuit the cathode of the LED is connected to the **collector** of the **S8050 (NPN transistor)**, while this one is connected to the **emitter** of the **S8550 (PNP transistor)**.



Code

Note:

- Open the `2.15_transistor.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)
```

Two kinds of transistors can be controlled with the same code. When we press the button, Pico W will send a high-level signal to the transistor; when we release it, it will send a low-level signal. We can see that diametrically opposite

phenomena have occurred in the two circuits.

- The circuit using the S8050 (NPN transistor) will light up when the button is pressed, which means it is receiving a high-level conduction circuit;
- The circuit that uses the S8550 (PNP transistor) will light up when it is released, which means it is receiving a low-level conduction circuit.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.22 2.16 Control Another Circuit

In our daily life, we can press the switch to light up or turn off the lamp. But what if you want to control the lamp with Pico W so that it can turn off automatically after ten minutes?

A relay can help you accomplish this idea.

A relay is actually a special kind of switch that is controlled by one side of the circuit (usually a low-voltage circuit) and used to control the other side of the circuit (usually a high-voltage circuit). This makes it practical to modify our home appliances to be controlled by a program, to become smart devices, or even to access the Internet.

Warning: Modification of electrical appliances comes with great danger, do not try it lightly, please do it under the guidance of professionals.

- *Relay*

Here we only use a simple circuit powered by a breadboard power module as an example to show how to control it using relay.

- cpn_power_module

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

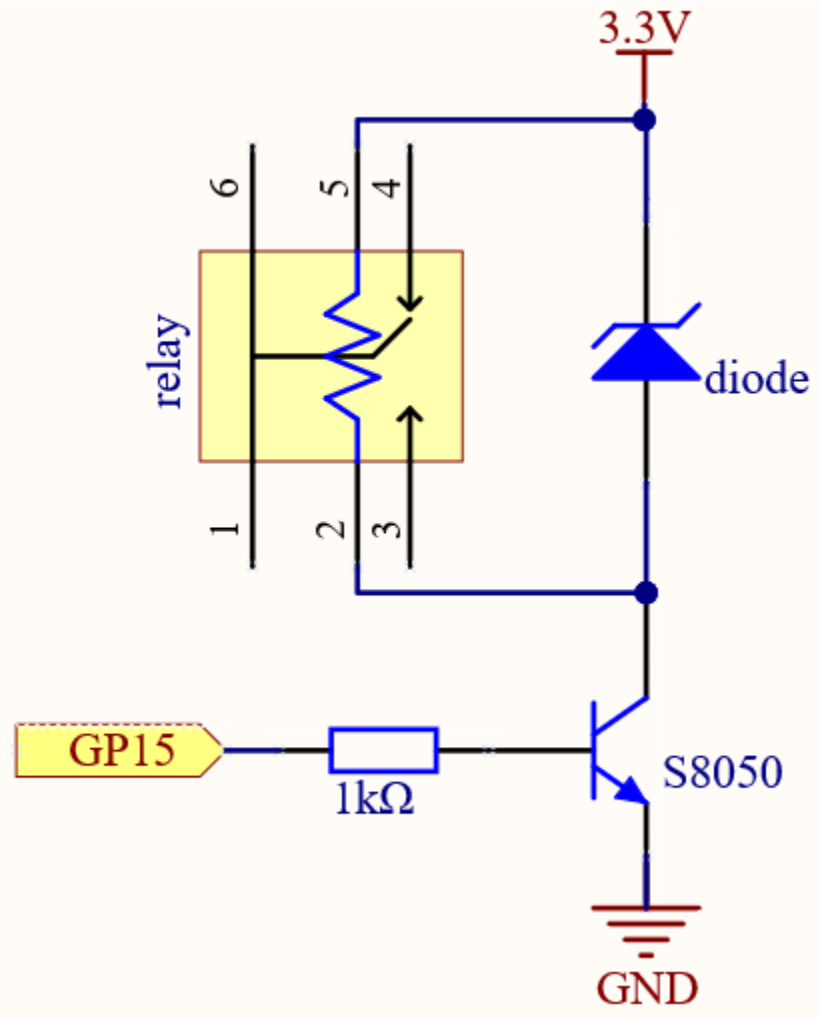
Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

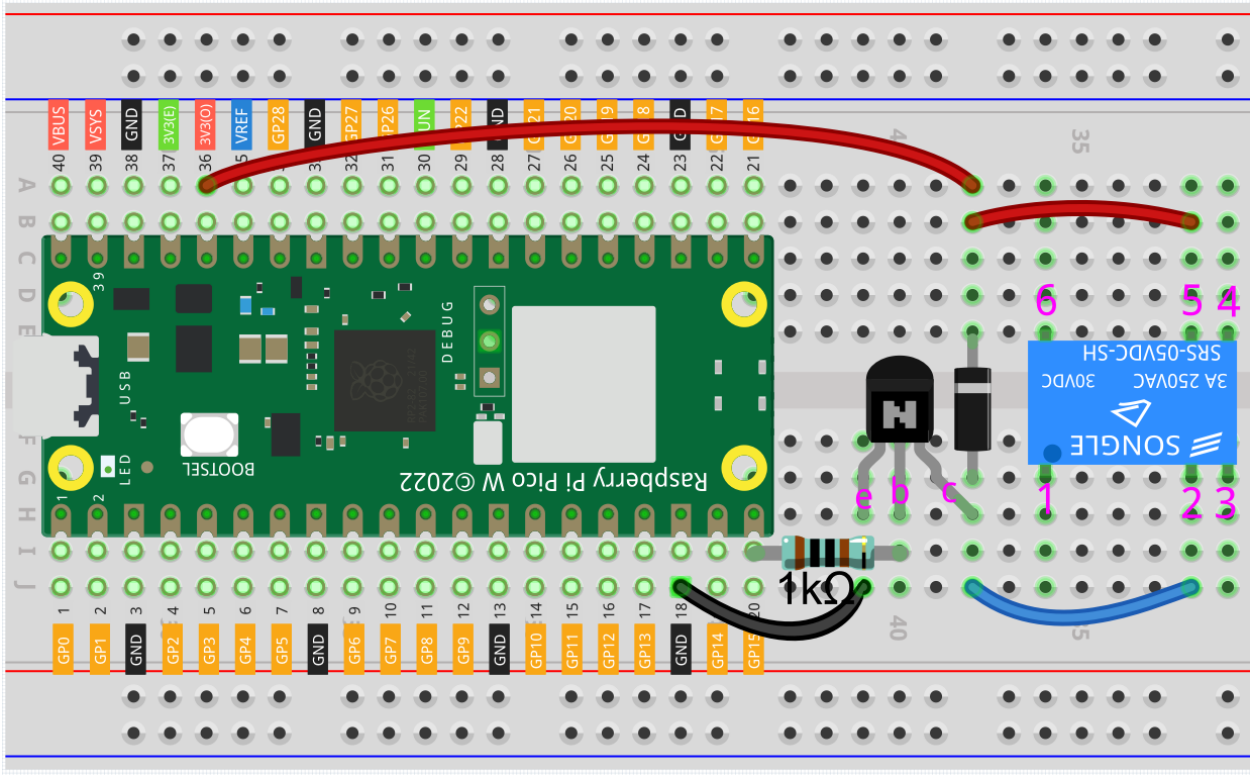
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Transistor*	1(S8050)	
6	*Diode*	1	
7	*Relay*	1	

Wiring

First, build a low-voltage circuit for controlling a relay. Driving the relay requires a high current, so a transistor is needed, and here we use the S8050.





A diode (continuity diode) is used here to protect the circuit. The cathode is the end with the silver ribbon connected to the power supply, and the anode is connected to the transistor.

When the voltage input changes from High (5V) to Low (0V), the transistor changes from saturation (amplification, saturation, and cutoff) to cutoff, and there is suddenly no way for current to flow through the coil.

At this point, if this freewheeling diode does not exist, the coil will produce a self-induced electric potential at both ends that is several times higher than the supply voltage, and this voltage plus the voltage from the transistor power supply is enough to burn it.

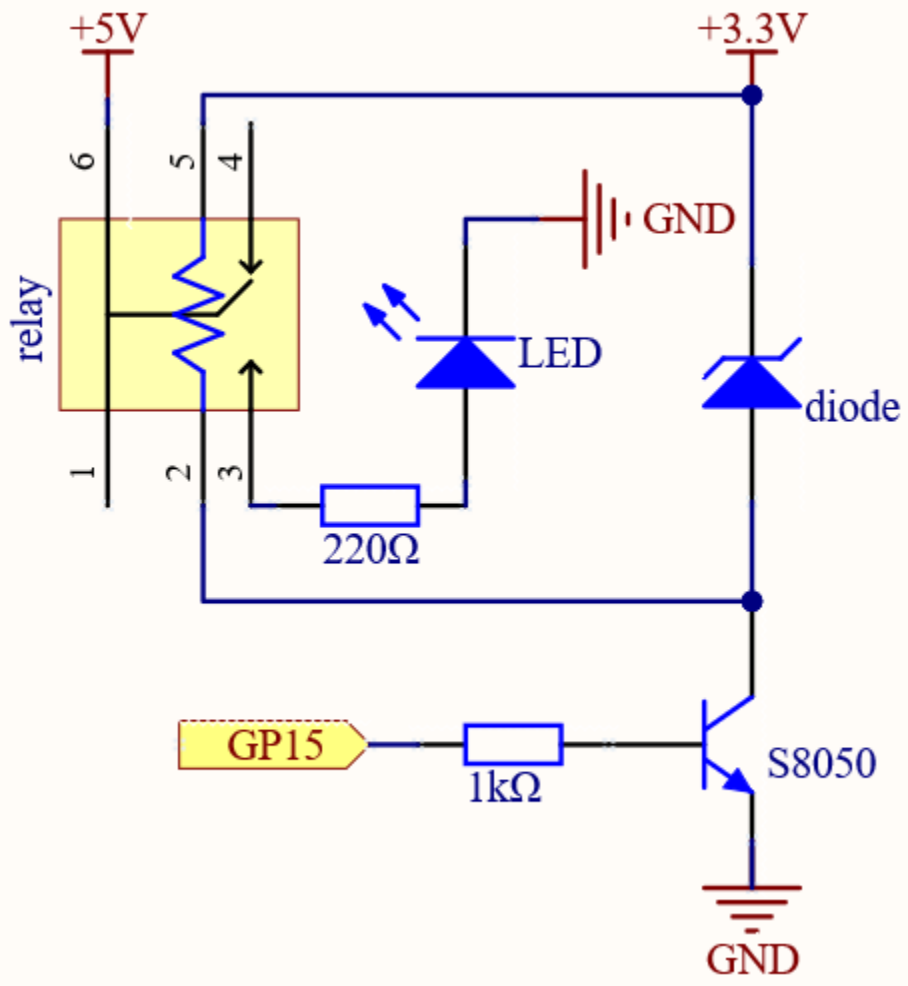
After adding the diode, the coil and the diode instantly form a new circuit powered by the energy stored in the coil to discharge, thus avoiding the excessive voltage will damage devices such as transistors on the circuit.

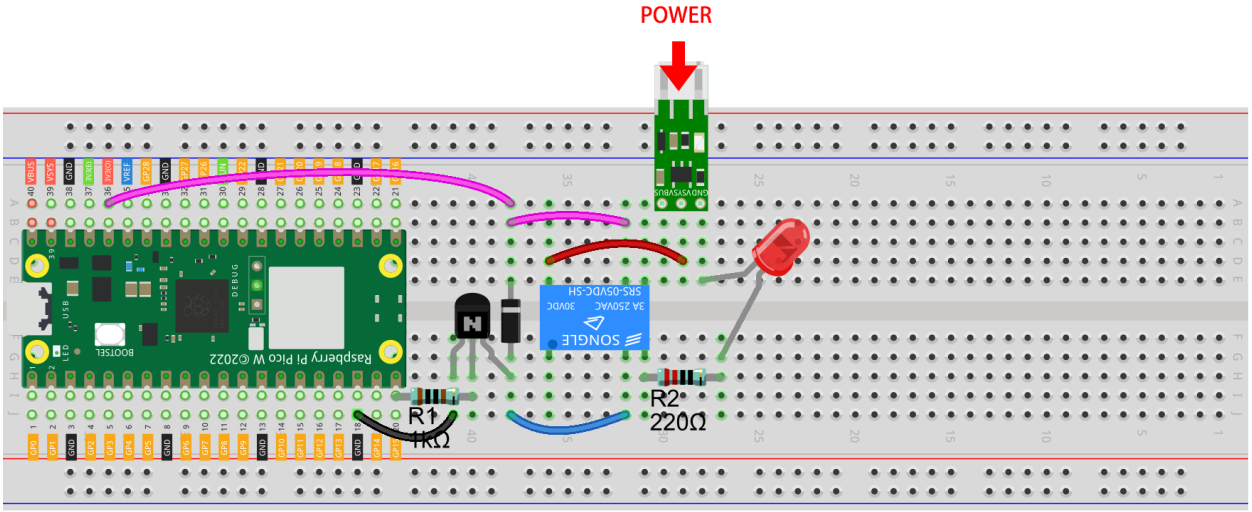
- *Diode*
- Flyback Diode - Wikipedia

At this point the program is ready to run, and after running you will hear the “tik tok” sound, which is the sound of the contactor coil inside the relay sucking and breaking.

Then we connect the two ends of the load circuit to pins 3 and 6 of the relay respectively.

..(Take the simple circuit powered by the breadboard power module described in the previous article as an example.)





At this point, the relay will be able to control the load circuit on and off.

Code

Note:

- Open the `2.16_control_another_circuit.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)
```

When the code is run, the relay will switch the operating state of the controlled circuit every two seconds. You can manually comment out one of the lines to further clarify the correspondence between the relay circuit and the load circuit.

Learn More

Pin 3 of the relay is normally open and only turns on when the contactor coil is operating; pin 4 is normally closed and turns on when the contactor coil is energized. Pin 1 is connected to pin 6 and is the common terminal of the load circuit.

By switching one end of the load circuit from pin 3 to pin 4, you will be able to get exactly the opposite operating state.

3. Sound & Display & Movement

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.23 3.1 Beep

The active buzzer is a typical digital output device that is as easy to use as lighting up an LED!

- *Buzzer*

Required Components

In this project, we need the following components.

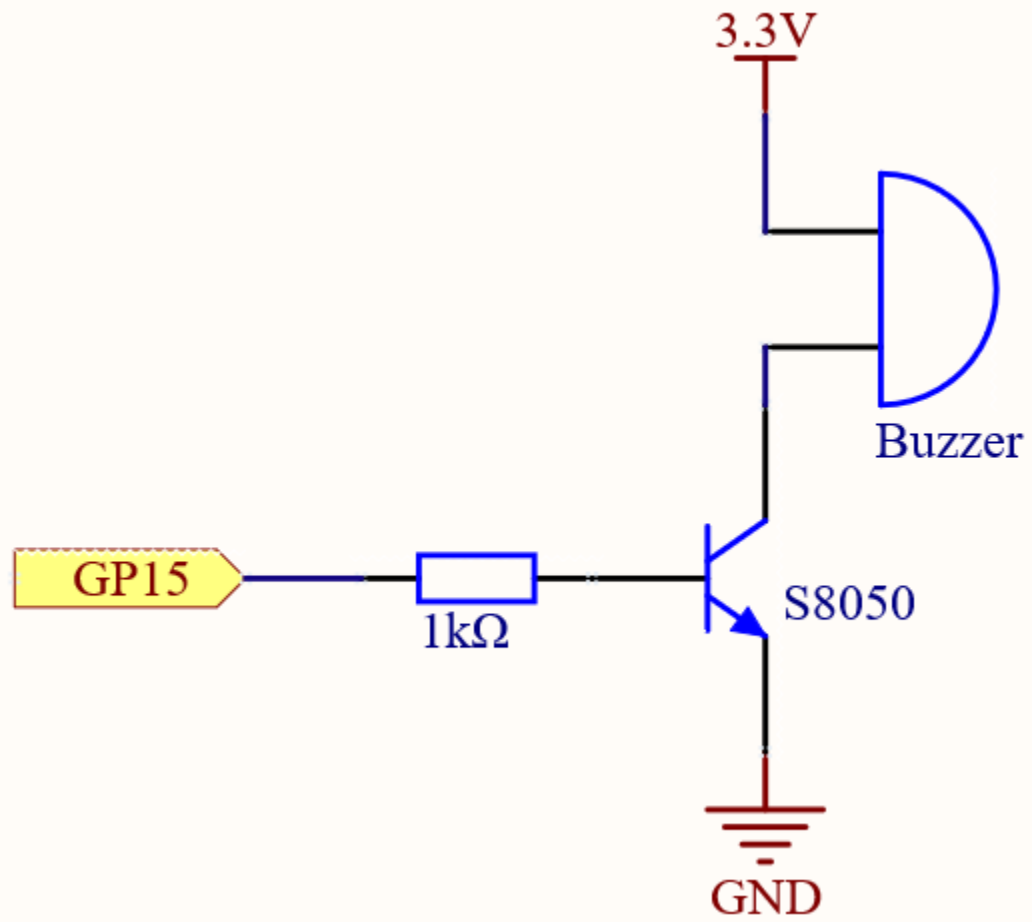
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Transistor*	1(S8050)	
6	*Resistor*	1(1K)	
7	Active *Buzzer*	1	

Schematic



When the GP15 output is high, after the 1K current limiting resistor (to protect the transistor), the S8050 (NPN transistor) will conduct, so that the buzzer will sound.

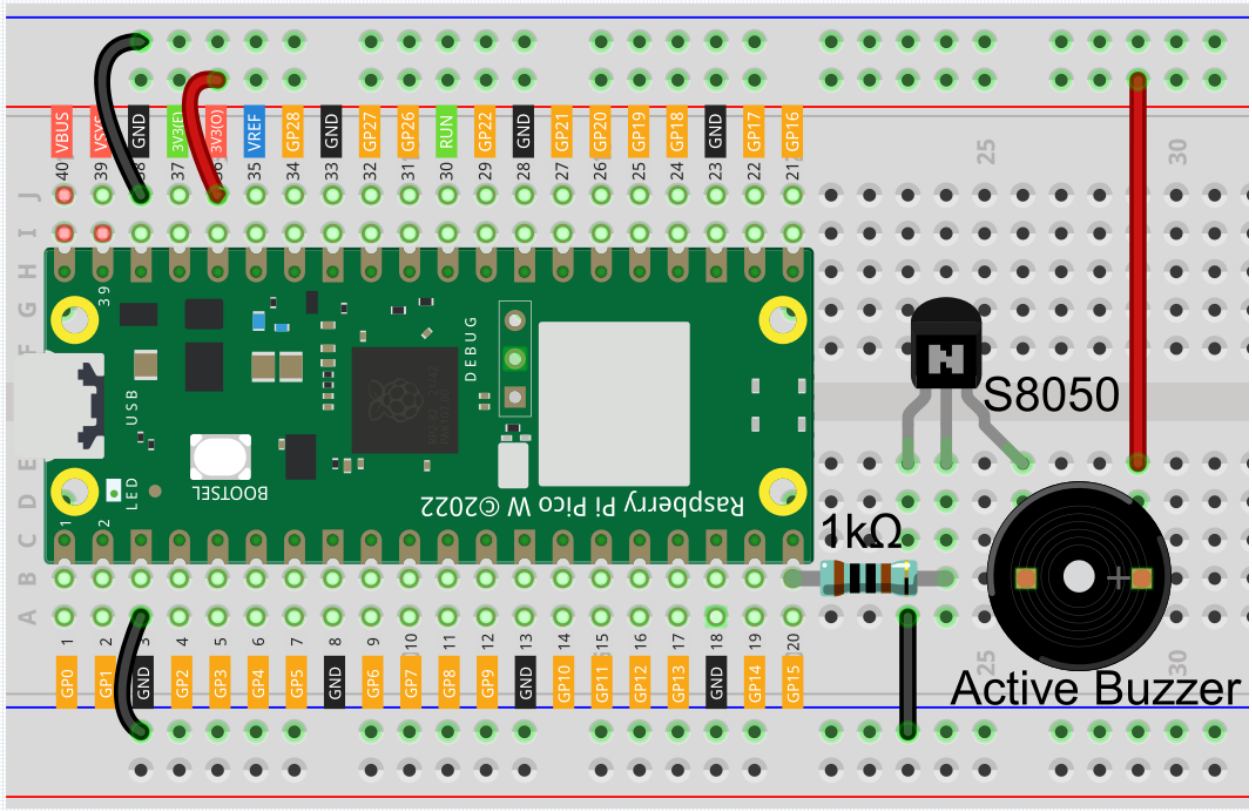
The role of S8050 (NPN transistor) is to amplify the current and make the buzzer sound louder. In fact, you can also connect the buzzer directly to GP15, but you will find that the buzzer sound is smaller.

Wiring

Two types of buzzers are included in the kit. We need to use active buzzer. Turn them around, the sealed back (not the exposed PCB) is the one we want.



The buzzer needs to use a transistor when working, here we use S8050 (NPN Transistor).



Code

Note:

- Open the `3.1_beep.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)
```

After the code runs, you will hear a beep every second.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.24 3.2 Custom Tone

We have used active buzzer in the previous project, this time we will use passive buzzer.

Like the active buzzer, the passive buzzer also uses the phenomenon of electromagnetic induction to work. The difference is that a passive buzzer does not have oscillating source, so it will not beep if DC signals are used. But this allows the passive buzzer to adjust its own oscillation frequency and can emit different notes such as "doh, re, mi, fa, sol, la, ti".

Let the passive buzzer emit a melody!

- *Buzzer*

Required Components

In this project, we need the following components.

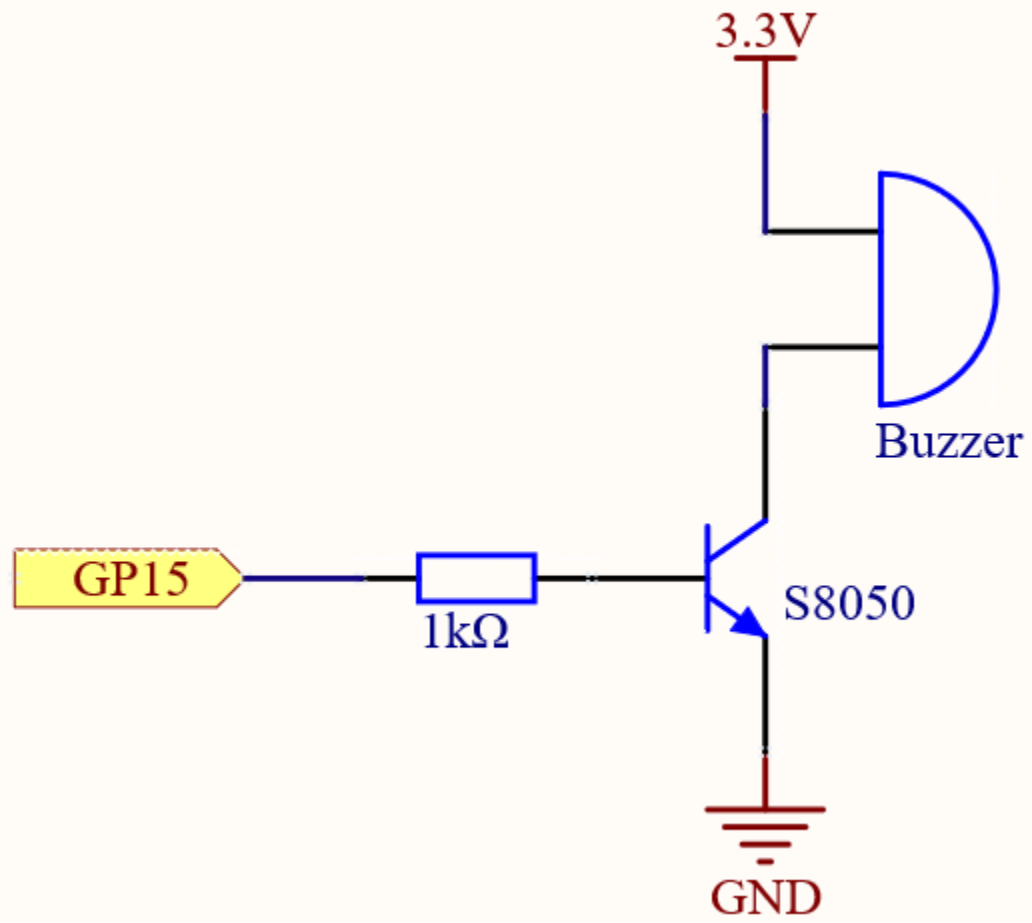
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUANTITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Transistor*	1(S8050)	
6	*Resistor*	1(1K)	
7	Passive *Buzzer*	1	

Schematic



When the GP15 output is high, after the 1K current limiting resistor (to protect the transistor), the S8050 (NPN transistor) will conduct, so that the buzzer will sound.

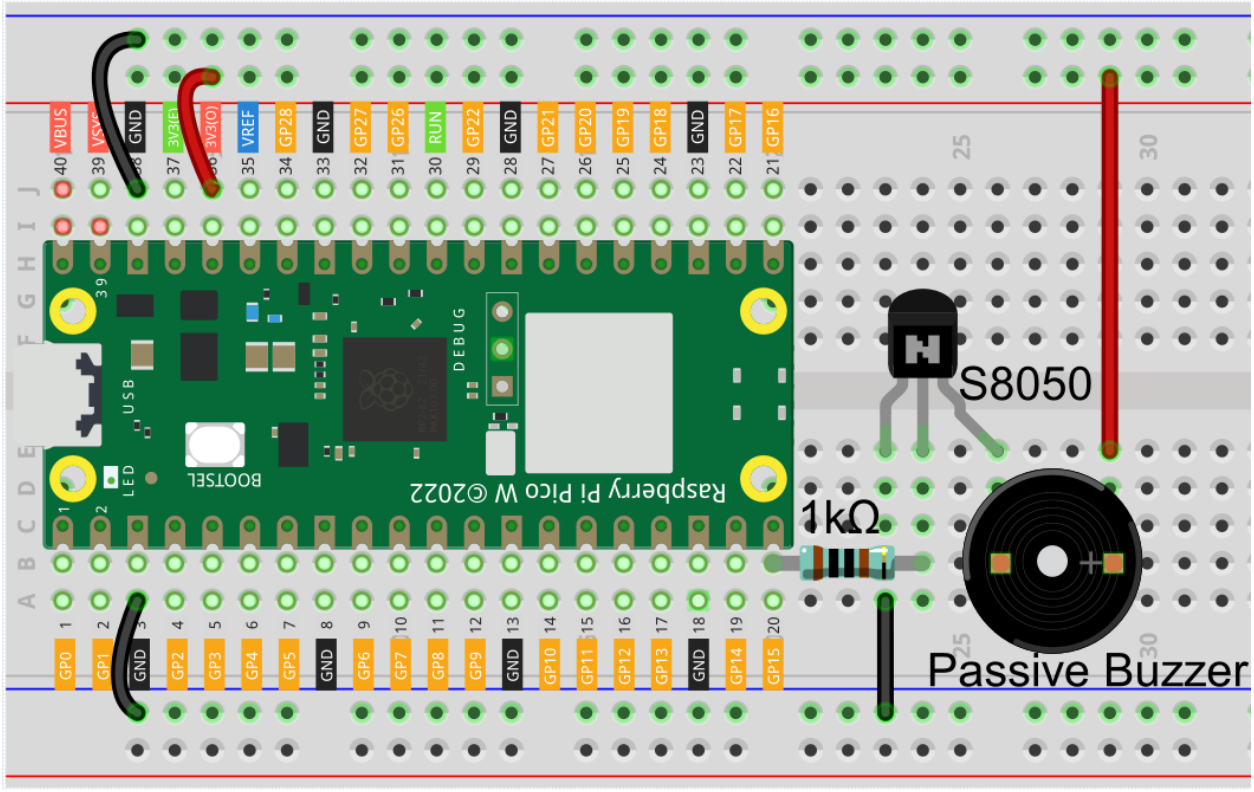
The role of S8050 (NPN transistor) is to amplify the current and make the buzzer sound louder. In fact, you can also connect the buzzer directly to GP15, but you will find that the buzzer sound is smaller.

Wiring



Two buzzers are included in the kit, we use a passive buzzer (one with an exposed PCB on the back).

The buzzer needs a transistor to work, here we use S8050.



Code

Note:

- Open the `3.2_custom_tone.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)
```

How it works?

If the passive buzzer given a digital signal, it can only keep pushing the diaphragm without producing sound.

Therefore, we use the `tone()` function to generate the PWM signal to make the passive buzzer sound.

This function has three parameters:

- **pin**, the GPIO pin that controls the buzzer.
- **frequency**, the pitch of the buzzer is determined by the frequency, the higher the frequency, the higher the pitch.
- **Duration**, the duration of the tone.

We use the `duty_u16()` function to set the duty cycle to 30000(about 50%). It can be other numbers, and it only needs to generate a discontinuous electrical signal to oscillate.

Learn More

We can simulate the specific tone according to the fundamental frequency of the piano, so as to play a complete piece of music.

- Piano key frequencies - Wikipedia

Note:

- Open the `3.2_custom_tone_2.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.

- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.25 3.3 RGB LED Strip

WS2812 is a intelligent control LED light source that the control circuit and RGB chip are integrated in a package of 5050 components. It internal include intelligent digital port data latch and signal reshaping amplification drive circuit. Also include a precision internal oscillator and a programmable constant current control part, effectively ensuring the pixel point light color height consistent.

The data transfer protocol use single NZR communication mode. After the pixel power-on reset, the DIN port receive data from controller, the first pixel collect initial 24bit data then sent to the internal data latch, the other data which reshaping by the internal signal reshaping amplification circuit sent to the next cascade pixel through the DO port. After transmission for each pixel, the signal to reduce 24bit. pixel adopt auto reshaping transmit technology, making the pixel cascade number is not limited the signal transmission, only depend on the speed of signal transmission.

- *WS2812 RGB 8 LEDs Strip*

Required Components

In this project, we need the following components.

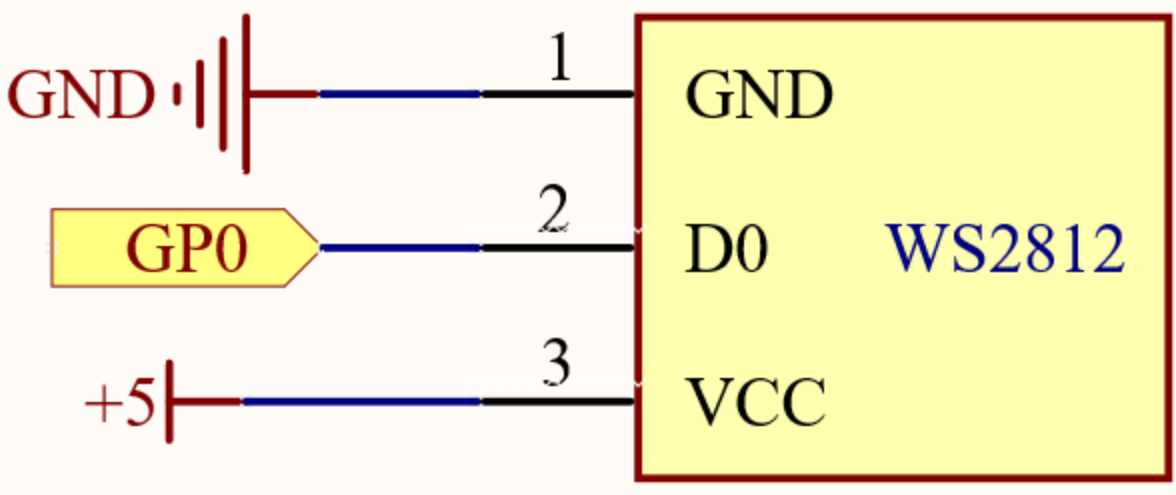
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

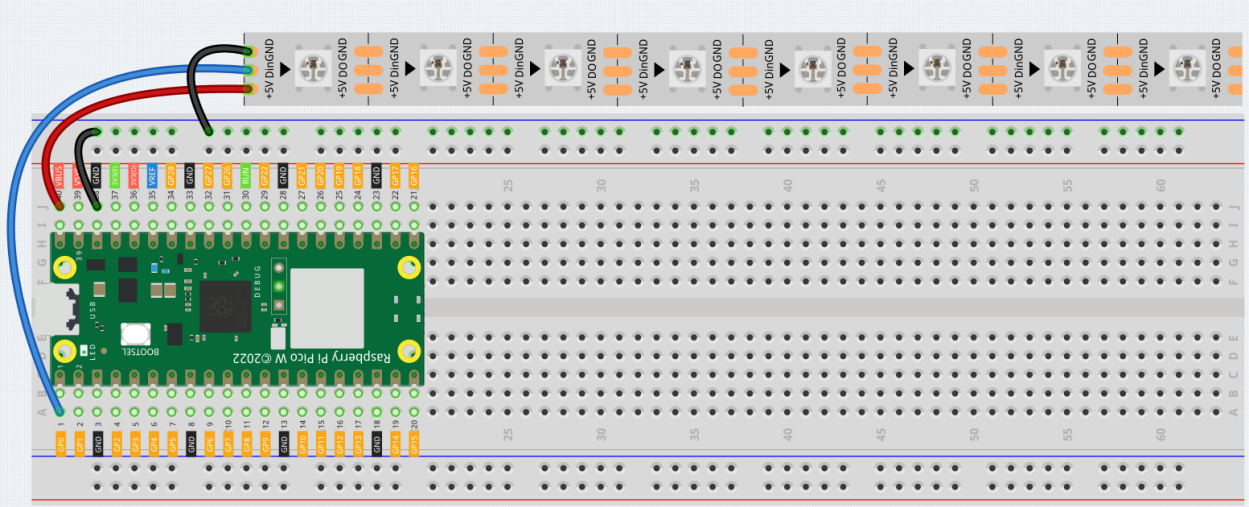
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*WS2812 RGB 8 LEDs Strip*	1	

Schematic



Wiring



> **Warning:** One thing you need to pay attention to is current.
>
> Although the LED Strip with any number of LEDs can be used in Pico W, the power of its VBUS pin is limited. Here, we will use eight LEDs, which are safe. But if you want to use more LEDs, you need to add a separate power supply.

Code

Note:

- Open the `3.3_rgb_led_strip.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the library called `ws2812.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *1.4 Upload the Libraries to Pico*.

```
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()
```

Let's select some favorite colors and display them on the RGB LED Strip!

How it works?

In the ws2812 library, we have integrated related functions into the WS2812 class.

You can use the RGB LED Strip with the following statement.

```
from ws2812 import WS2812
```

Declare a WS2812 type object, named "ws", it is connected to "pin", there are "number" RGB LEDs on the WS2812 strip.

```
ws = WS2812(pin,number)
```

ws is an array object, each element corresponds to one RGB LED on the WS2812 strip, for example, ws[0] is the first one, ws[7] is the eighth.

We can assign color values to each RGB LED, these values must be 24-bit color (represented with six hexadecimal digits) or list of 3 8-bit RGB.

For example, the red value is "0xFF0000" or "[255,0,0]".

```
ws[i] = color value
```

Then use this statement to write the color for the LED Strip and light it up.

```
ws.write()
```

You can also directly use the following statement to make all LEDs light up the same color.

```
ws.write_all(color value)
```

Learn More

We can randomly generate colors and make a colorful flowing light.

Note:

- Open the `3.3_rgb_led_strip_2.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.

- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)

while True:
    flowing_light()
    print(ws[0])
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

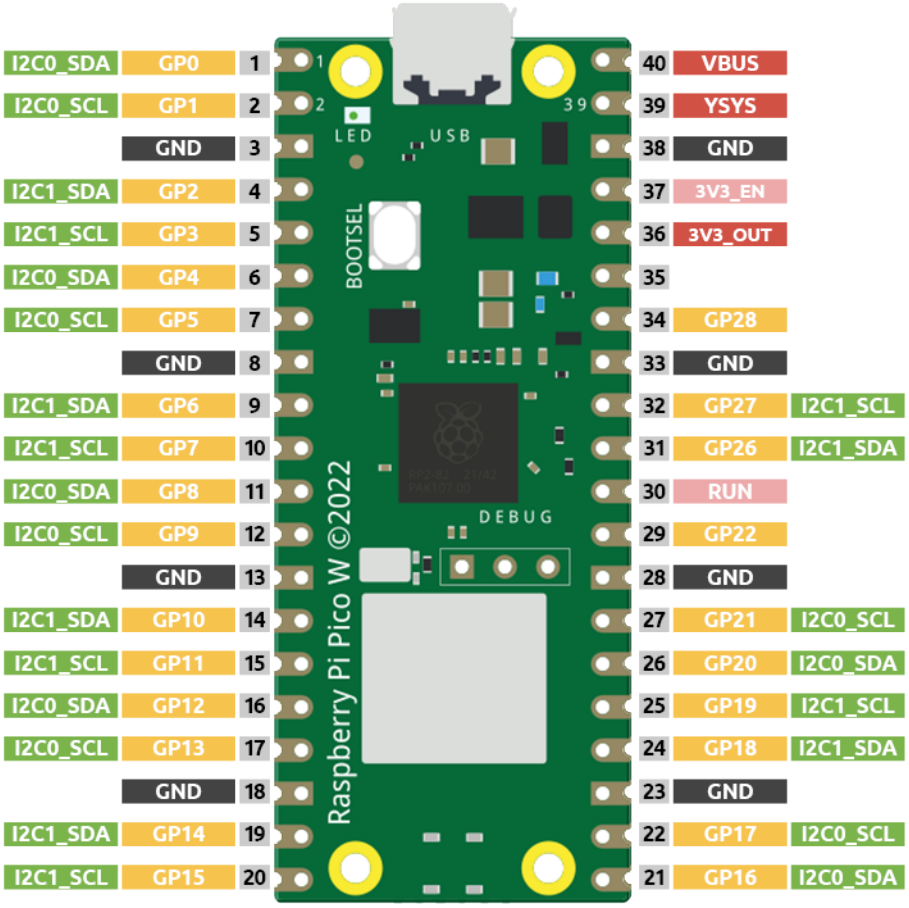
Ready to explore and create with us? Click [] and join today!

4.26 3.4 Liquid Crystal Display

LCD1602 is a character type liquid crystal display, which can display 32 (16*2) characters at the same time.

As we all know, though LCD and some other displays greatly enrich the man-machine interaction, they share a common weakness. When they are connected to a controller, multiple IOs will be occupied of the controller which has no so many outer ports. Also it restricts other functions of the controller. Therefore, LCD1602 with an I2C bus is developed to solve the problem.

- *I2C LCD1602*
- Inter-Integrated Circuit - Wikipedia



Here we will use the I2C0 interface to control the LCD1602 and display text.

Required Components

In this project, we need the following components.

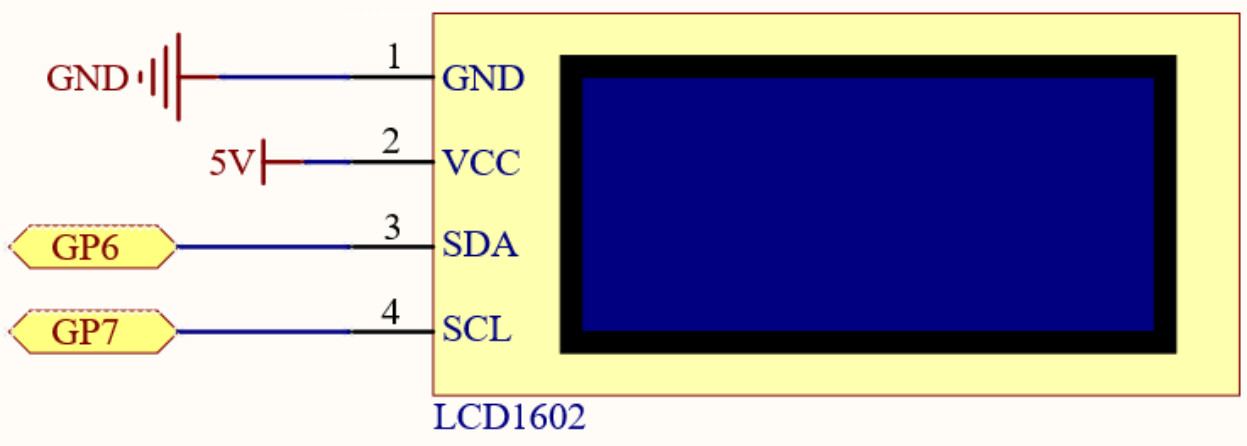
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

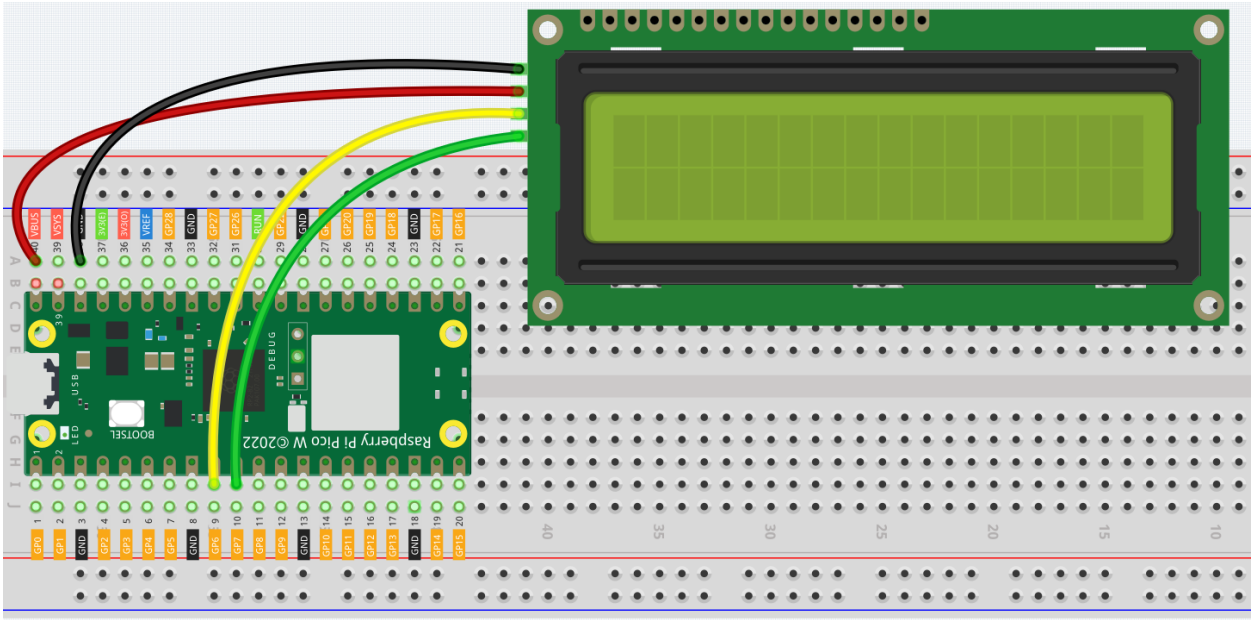
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*I2C LCD1602*	1	

Schematic



Wiring



Code

Note:

- Open the `3.4_liquid_crystal_display.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the library called `lcd1602.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *1.4 Upload the Libraries to Pico*.

```
from lcd1602 import LCD
import utime
```

(continues on next page)

(continued from previous page)

```
lcd = LCD()
string = " Hello!\n"
lcd.message(string)
utime.sleep(2)
string = "    Sunfounder!"
lcd.message(string)
utime.sleep(2)
lcd.clear()
```

After the program runs, you will be able to see two lines of text appear on the LCD in turn, and then disappear.

Note: When the code is running, if the screen is blank, you can turn the potentiometer on the back to increase the contrast.

How it works?

In the lcd1602 library, we integrate the relevant functions of lcd1602 into the LCD class.

Import lcd1602 library

```
from lcd1602 import LCD
```

Declare an object of the LCD class and name it lcd.

```
lcd = LCD()
```

This statement will display the text on the LCD. It should be noted that the argument must be a string type. If we want to pass an integer or float, we must use the forced conversion statement `str()`.

```
lcd.message(string)
```

If you call this statement multiple times, lcd will superimpose the texts. This requires the use of the following statement to clear the display.

```
lcd.clear()
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.27 3.5 Small Fan

Now we use the TA6586 to drive the DC motor to make it rotate clockwise and counterclockwise. Since the DC motor requires a relatively large current, for safety reasons, here we use a power module to supply power to the motor.

- *DC Motor*
- *TA6586 - Motor Driver Chip*
- cpn_power_module

Required Components

In this project, we need the following components.

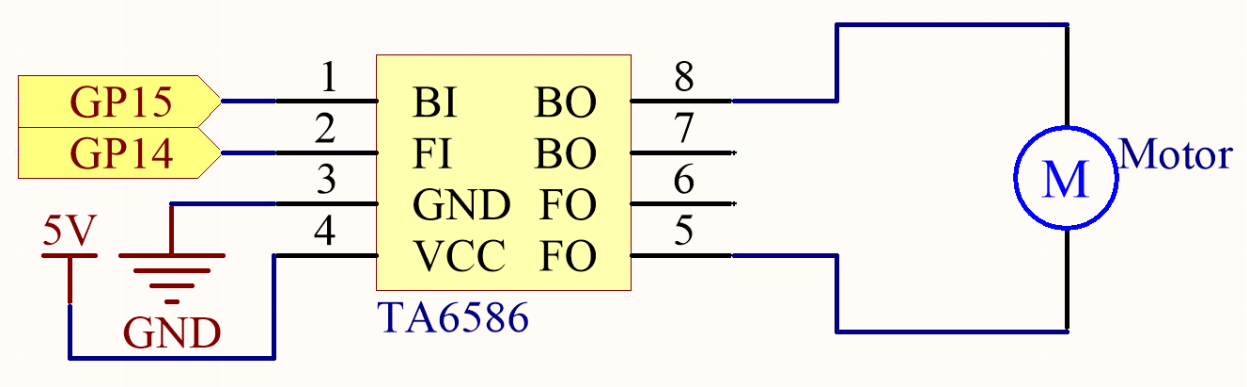
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*TA6586 - Motor Driver Chip*	1	
6	*DC Motor*	1	
7	*Li-po Charger Module*	1	
8	18650 Battery	1	
9	Battery Holder	1	

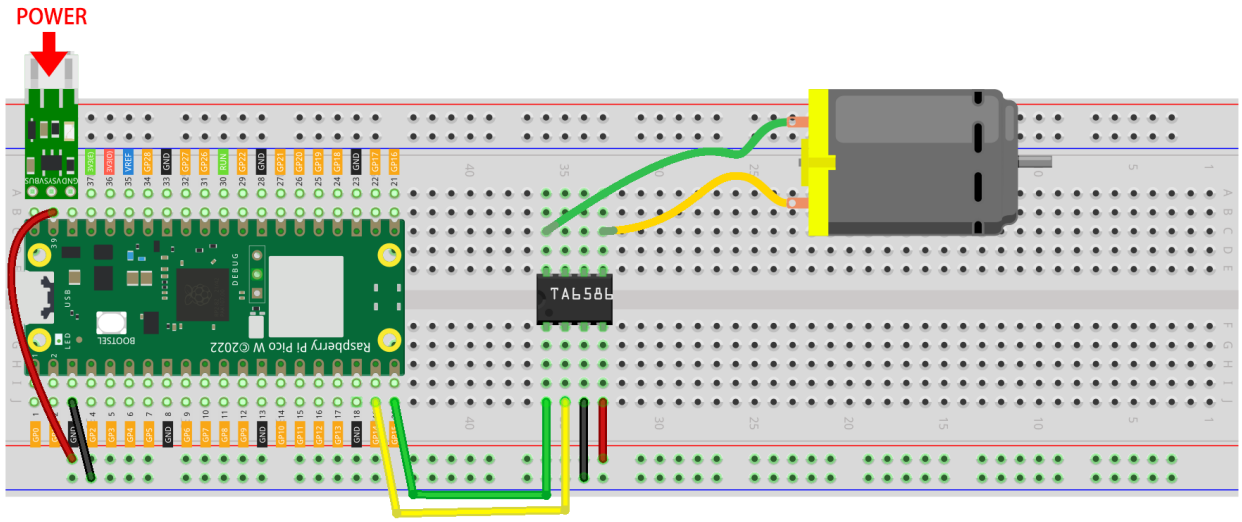
Schematic



Wiring

Note:

- Since DC motors require a high current, we use a Li-po Charger module to power the motor here for safety reasons.
- Make sure your Li-po Charger Module is connected as shown in the diagram. Otherwise, a short circuit will likely damage your battery and circuitry.



Code

Note:

- Open the `3.5_small_fan.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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():
    motor1A.low()
    motor2A.high()

def stopMotor():
    motor1A.low()
```

(continues on next page)

(continued from previous page)

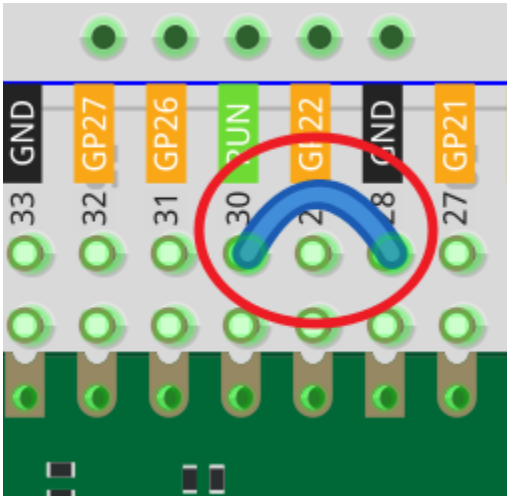
```
    motor2A.low()

while True:
    clockwise()
    utime.sleep(1)
    stopMotor()
    utime.sleep(1)
    anticlockwise()
    utime.sleep(1)
    stopMotor()
    utime.sleep(1)
```

Once the program is running, the motor will rotate back and forth in a regular pattern.

Note:

- If the motor is still spinning after you click the Stop button, you need to reset the **RUN** pin on the Pico W with a wire to GND at this time, and then unplug this wire to run the code again.
- This is because the motor is operating with too much current, which may cause the Pico W to disconnect from the computer.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.28 3.6 Pumping

Small centrifugal pumps are suitable for projects with automatic plant watering. It can also be used to make tiny smart water features.

Its power component is an electric motor, driven in exactly the same way as a normal motor.

- *DC Water Pump*
- *DC Motor*
- *TA6586 - Motor Driver Chip*
- cpn_power_module

Note:

1. Connect the tube to the motor outlet, submerge the pump in water, and then power it on.
2. You need to make sure that the water level is always higher than the motor. Idling may damage the motor due to heat generation and will also generate noise.
3. If you are watering plants, you need to avoid soil being drawn in, as this can clog the pump.
4. If water does not come out of the tube, there may be residual water in the tube blocking the air flow and needs to be drained first.

Required Components

In this project, we need the following components.

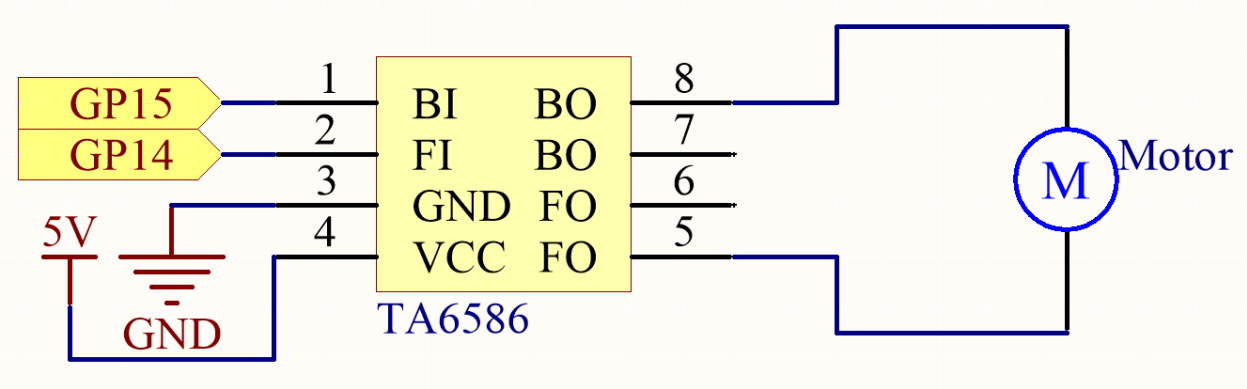
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*TA6586 - Motor Driver Chip*	1	
6	*Li-po Charger Module*	1	
7	18650 Battery	1	
8	Battery Holder	1	
9	*DC Water Pump*	1	

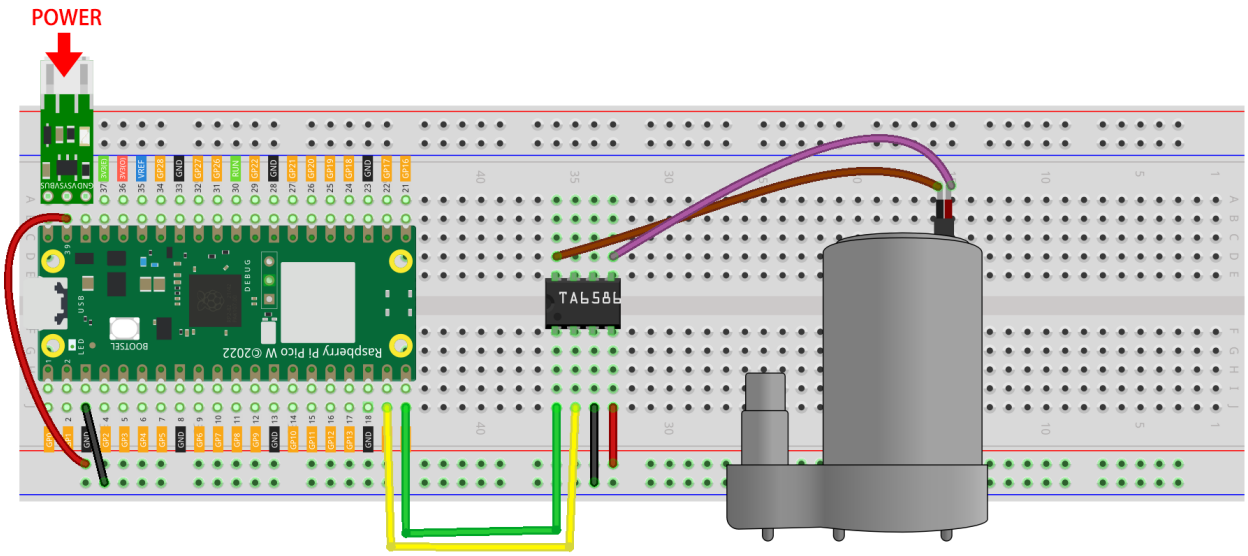
Schematic



Wiring

Note:

- Since pump require a high current, we use a Li-po Charger module to power the motor here for safety reasons.
- Make sure your Li-po Charger Module is connected as shown in the diagram. Otherwise, a short circuit will likely damage your battery and circuitry.



Code

Note:

- Open the `3.6_pumping.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
import machine
import utime
```

(continues on next page)

(continued from previous page)

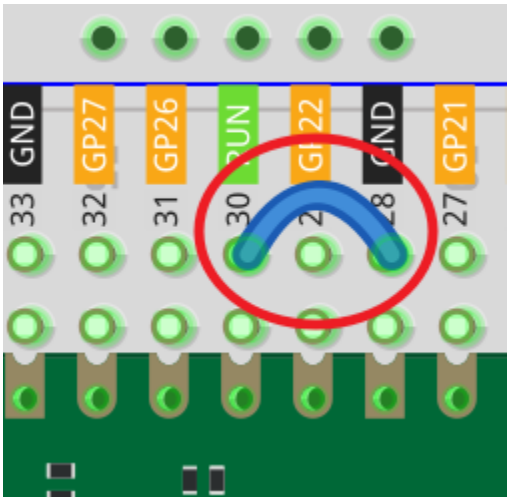
```
motor1A = machine.Pin(14, machine.Pin.OUT)
motor2A = machine.Pin(15, machine.Pin.OUT)

while True:
    motor1A.high()
    motor2A.low()
```

After the code is run, the pump starts working and you will see water flowing out of the tube at the same time.

Note:

- If the motor is still spinning after you click the Stop button, you need to reset the **RUN** pin on the Pico W with a wire to GND at this time, and then unplug this wire to run the code again.
- This is because the motor is operating with too much current, which may cause the Pico W to disconnect from the computer.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.29 3.7 Swinging Servo

In this kit, in addition to LED and passive buzzer, there is also a device controlled by PWM signal, Servo.

Servo is a position (angle) servo device, which is suitable for those control systems that require constant angle changes and can be maintained. It has been widely used in high-end remote control toys, such as airplanes, submarine models, and remote control robots.

Now, try to make the servo sway!

- *Servo*

Required Components

In this project, we need the following components.

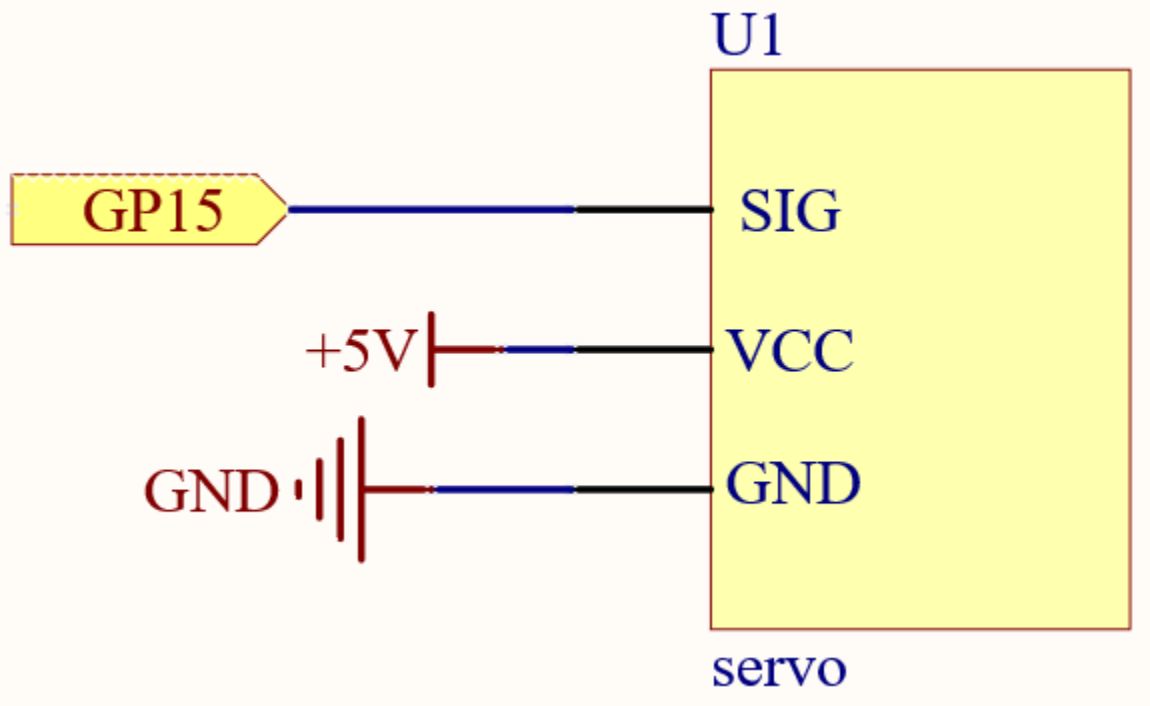
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

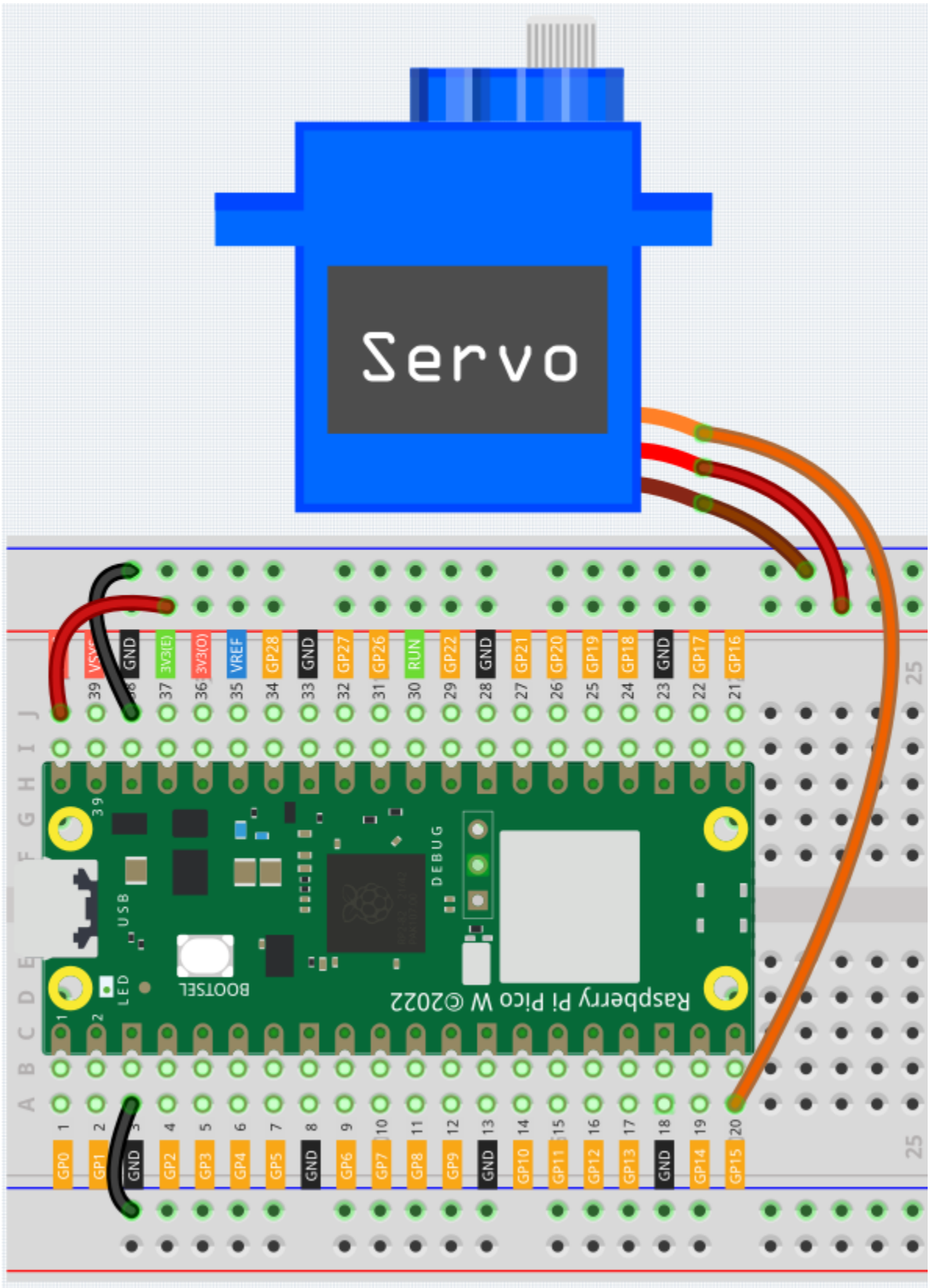
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Servo*	1	

Schematic



Wiring



- Orange wire is signal and connected to GP15.

- Red wire is VCC and connected to VBUS(5V).
- Brown wire is GND and connected to GND.

Code

Note:

- Open the `3.7_swinging_servo.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)
```

When the program is running, we can see the Servo Arm swinging back and forth from 0° to 180°.

The program will always run because of the `while True` loop, we need to press the Stop button to end the program.

How it works?

We defined the `servo_write()` function to make the servo run.

This function has two parameters:

- `pin`, the GPIO pin that controls the servo.
- `Angle`, the angle of the shaft output.

In this function, `interval_mapping()` is called to map the angle range 0 ~ 180 to the pulse width range 0.5 ~ 2.5ms.

```
pulse_width=interval_mapping(angle, 0, 180, 0.5,2.5)
```

Why is it 0.5~2.5? This is determined by the working mode of the Servo.

Servo

Next, convert the pulse width from period to duty. Since `duty_u16()` cannot have decimals when used (the value cannot be a float type), we used `int()` to force the duty to be converted to an int type.

```
duty=int(interval_mapping(pulse_width, 0, 20, 0,65535))
```

Finally, write the duty value into `duty_u16()`.

4. Controller

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.30 4.1 Toggle the Joystick

If you play a lot of video games, then you should be very familiar with the Joystick. It is usually used to move the character around, rotate the screen, etc.

The principle behind Joystick's ability to allow the computer to read our actions is very simple. It can be thought of as consisting of two potentiometers that are perpendicular to each other. These two potentiometers measure the analog value of the joystick vertically and horizontally, resulting in a value (x,y) in a planar right-angle coordinate system.

The joystick of this kit also has a digital input, which is activated when the joystick is pressed.

- *Joystick Module*

Required Components

In this project, we need the following components.

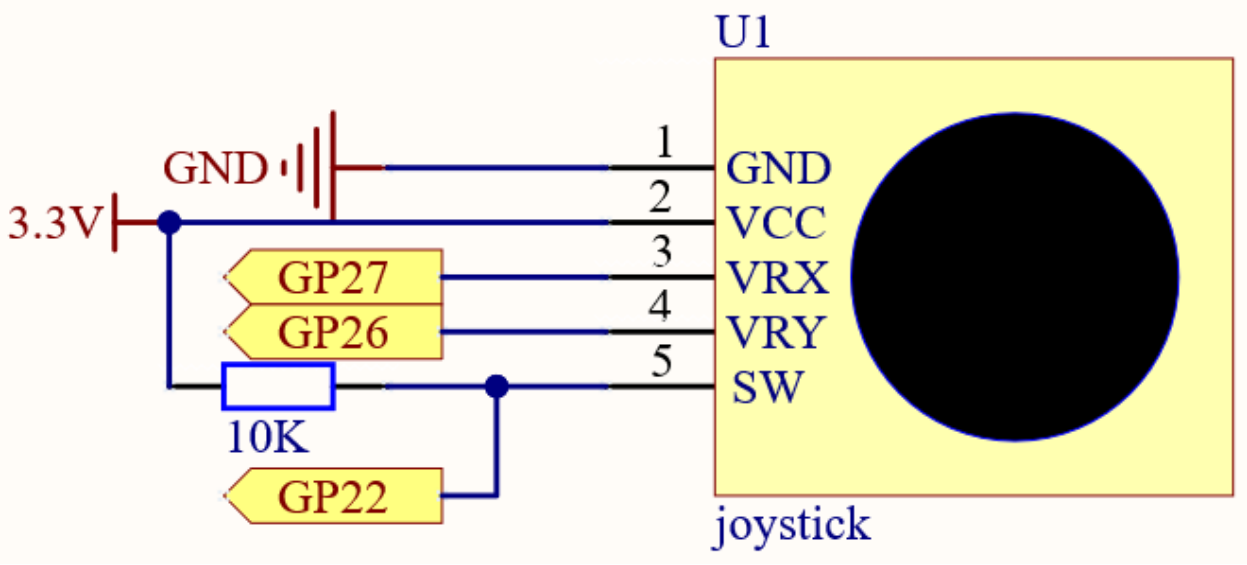
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

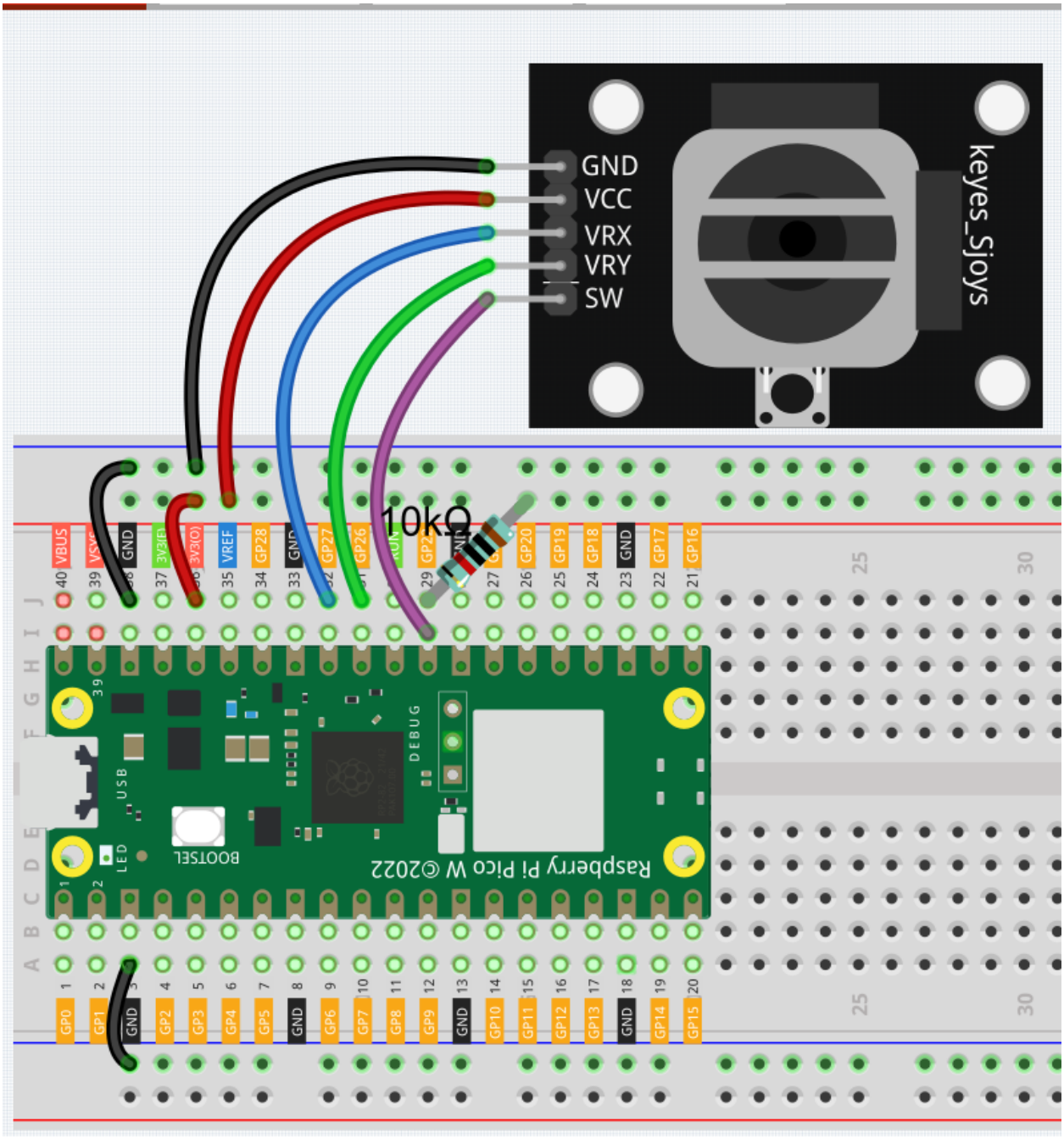
SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Joystick Module*	1	

Schematic



The SW pin is connected to a 10K pull-up resistor, the reason is to be able to get a stable high level on the SW pin (Z axis) when the joystick is not pressed; otherwise the SW is in a suspended state and the output value may vary between 0/1.

Wiring



Code

Note:

- Open the `4.1_toggle_the_joystick.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)
```

After the program runs, the Shell prints out the x,y,z values of joystick.

- The x-axis and y-axis values are analog values that vary from 0 to 65535.
- The Z-axis is a digital value with a status of 1 or 0.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.31 4.2 4x4 Keypad

The 4x4 keyboard, also known as the matrix keyboard, is a matrix of 16 keys excluded in a single panel.

The keypad can be found on devices that mainly require digital input, such as calculators, TV remote controls, push-button phones, vending machines, ATMs, combination locks, and digital door locks.

In this project, we will learn how to determine which key is pressed and get the related key value.

- *4x4 Keypad*
- E.161 - Wikipedia

Required Components

In this project, we need the following components.

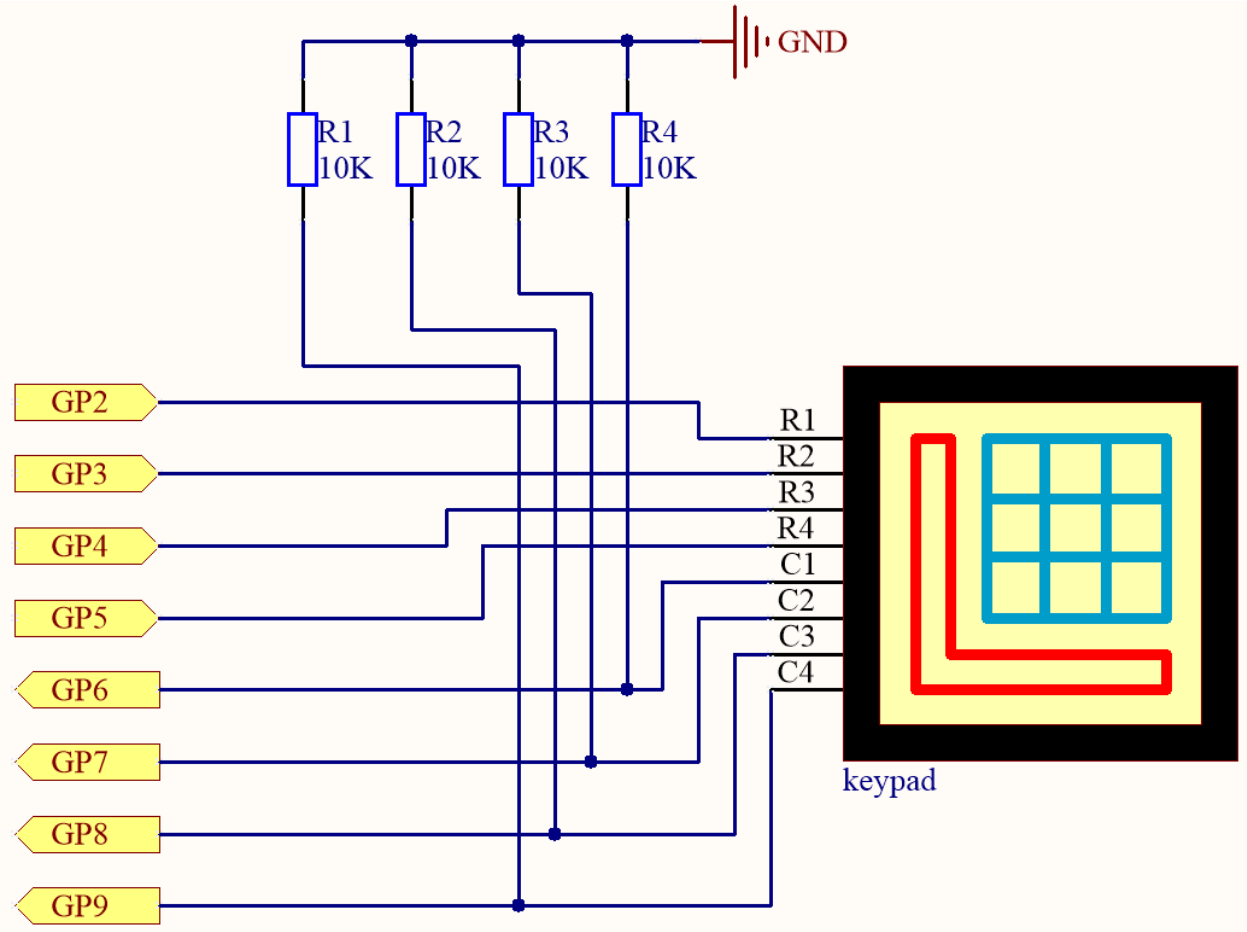
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	4(10K)	
6	*4x4 Keypad*	1	

Schematic



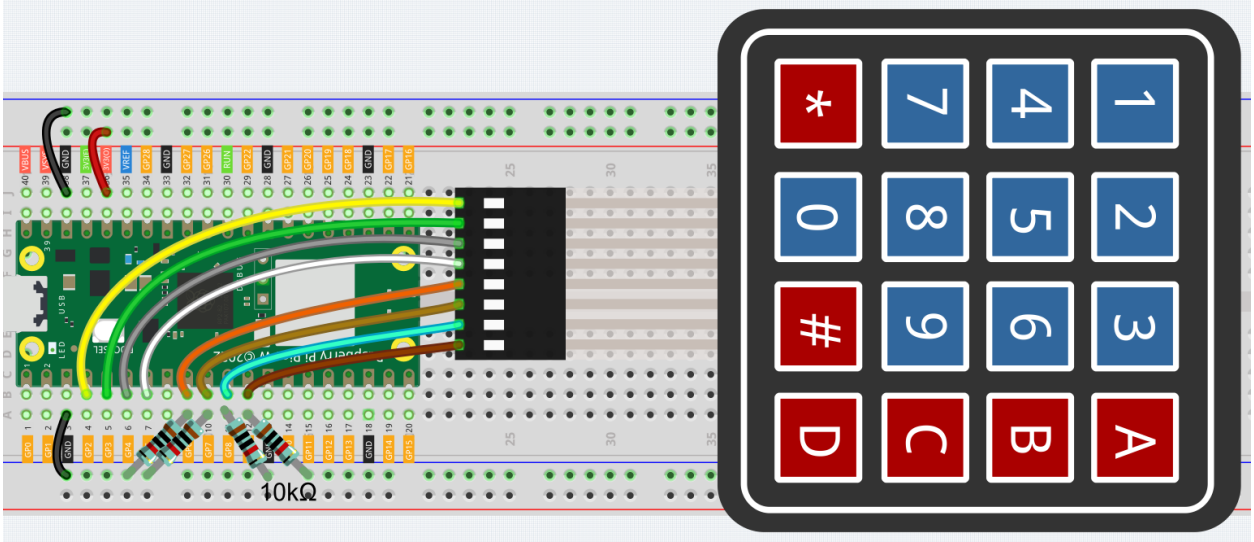
4 pull-down resistors are connected to each of the columns of the matrix keyboard, so that G6 ~ G9 get a stable low

level when the keys are not pressed.

The rows of the keyboard (G2 ~ G5) are programmed to go high; if one of G6 ~ G9 is read high, then we know which key is pressed.

For example, if G6 is read high, then numeric key 1 is pressed; this is because the control pins of numeric key 1 are G2 and G6, when numeric key 1 is pressed, G2 and G6 will be connected together and G6 is also high.

Wiring



To make the wiring easier, in the above diagram, the column row of the matrix keyboard and the 10K resistors are inserted into the holes where G6 ~ G9 are located at the same time.

Code

Note:

- Open the `4.2_4x4_keypad.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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]
col = []
for i in range(4):
```

(continues on next page)

(continued from previous page)

```
    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)
```

After the program runs, the Shell will print out the keys you pressed on the Keypad.

How it works

```
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]
col = []
for i in range(4):
    col.append(None)
    col[i] = machine.Pin(pin[i], machine.Pin.IN)
```

Declare each key of the matrix keyboard to the array `characters[]` and define the pins on each row and column.

```
last_key = None
while True:
    current_key = readKey()
    if current_key == last_key:
```

(continues on next page)

(continued from previous page)

```
        continue
    last_key = current_key
    if current_key != None:
        print(current_key)
    time.sleep(0.1)
```

This is the part of the main function that reads and prints the button value.

The function `readKey()` will read the state of every button.

The statement `if current_key != None` and `if current_key == last_key` is used to judge whether a key is pressed and the state of the pressed button. (If you press '3' when you press '1', the judgement is tenable.)

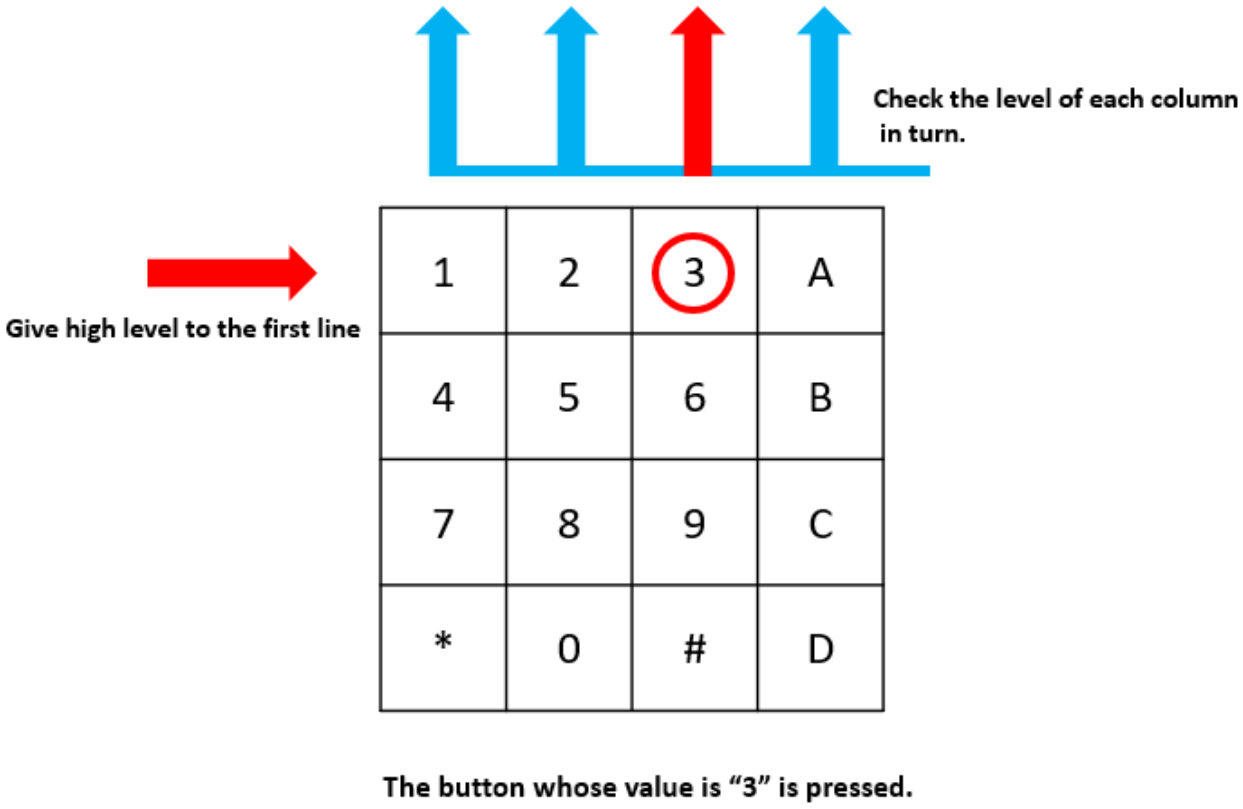
Prints the value of the currently pressed key when the condition is tenable.

The statement `last_key = current_key` assigns the state of each judgment to an array `last_key` to facilitate the next round of conditional judgment.

```
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
```

This function assigns a high level to each row in turn, and when the button is pressed, the column in which the key is located gets a high level. After the two-layer loop is judged, the value of the button whose state is 1 is stored in the array `key` .

If you press the key '3':



`row[0]` is written in high level, and `col[2]` gets high level.

`col[0]`, `col[1]`, `col[3]` get low level.

There are four states:0, 0, 1, 0; and we write '3' into `pressed_keys`.

When `row[1]` , `row[2]` , `row[3]` are written into high level, `col[0]` ~ `col[4]` get low level.

The loop stopped, there returns key = '3'.

If you press the buttons '1' and '3', there will return key = ['1','3'].

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.32 4.3 Electrode Keyboard

The MPR121 is a good choice when you want to add a large number of touch switches to your project. It has electrodes that can be extended with conductors. If you connect the electrodes to a banana, you can turn the banana into a touch switch.

- *MPR121 Module*

Required Components

In this project, we need the following components.

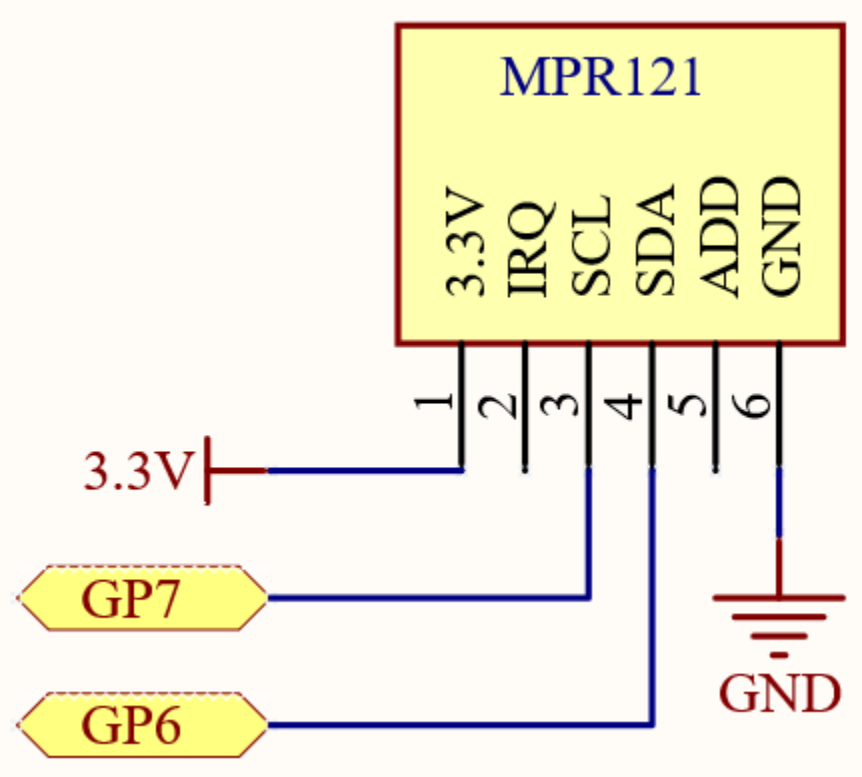
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

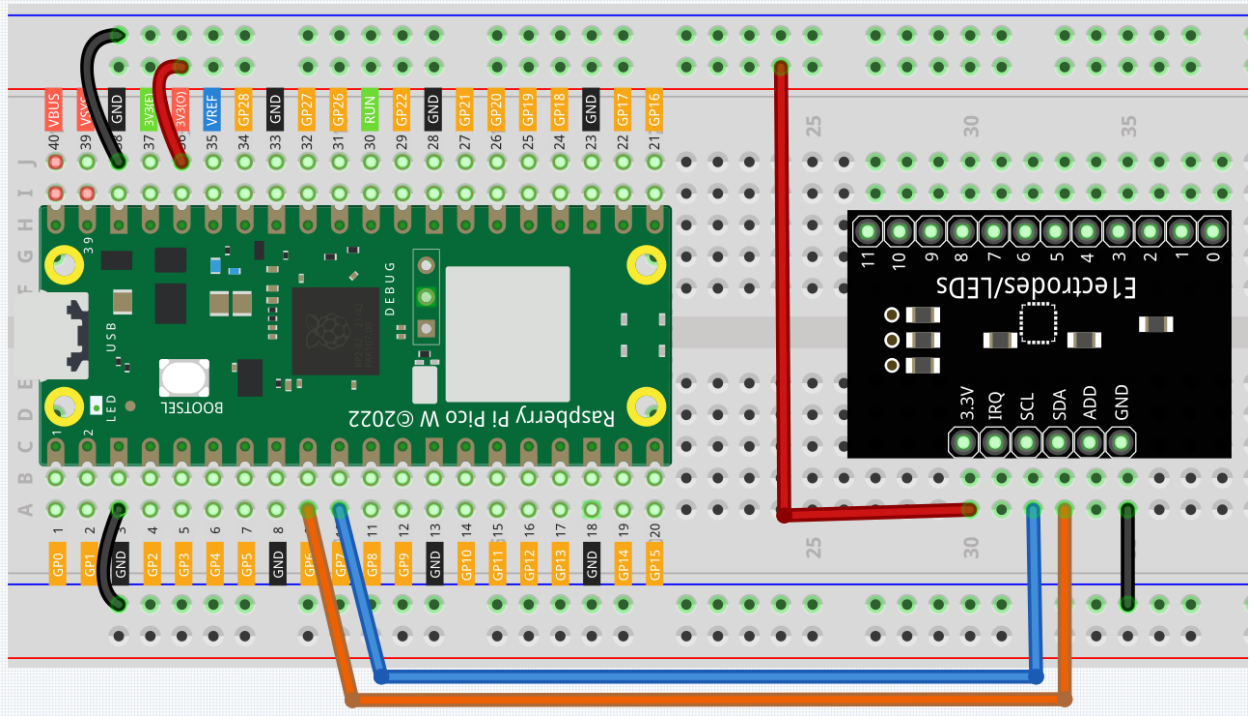
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*MPR121 Module*	1	

Schematic



Wiring



Code

Note:

- Open the `4.3_electrode_keyboard.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the library called `mpr121.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *1.4 Upload the Libraries to Pico*.

```
from mpr121 import MPR121
from machine import Pin, I2C
import time

i2c = I2C(1, sda=Pin(6), scl=Pin(7))
mpr = MPR121(i2c)

# check all keys
while True:
    value = mpr.get_all_states()
    if len(value) != 0:
        print(value)
    time.sleep_ms(100)
```

After the program runs, you can touch the twelve electrodes on the MPR121 with your hand and the touched electrodes will be printed out.

You can extend the electrodes to connect other conductors such as fruit, wire, foil, etc. This will give you more ways to trigger these electrodes.

How it works?

In the mpr121 library, we have integrated the functionality into the `MPR121` class.

```
from mpr121 import MPR121
```

MPR121 is an I2C module that requires a set of I2C pins to be defined to initialize the `MPR121` object. At this point the state of the module's electrodes will be recorded as initial values. If the electrodes are extended, the example needs to be rerun to reset the initial values.

```
from machine import Pin, I2C
i2c = I2C(1, sda=Pin(6), scl=Pin(7))
mpr = MPR121(i2c)
```

- Inter-Integrated Circuit - Wikipedia

Then use `mpr.get_all_states()` to read if the electrodes are triggered. If electrodes 2 and 3 are triggered, the value `[2, 3]` will be generated.

```
while True:
    value = mpr.get_all_states()
    if len(value) ! = 0:
        print(value)
    time.sleep_ms(100)
```

You can also use `mpr.is_touched(electrode)` to detect a specific electrode. When triggered, it returns `True`, otherwise it returns `False`.

```
while True:
    value = mpr.is_touched(0)
    print(value)
    time.sleep_ms(100)
```

5. Microchip

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.33 5.1 Microchip - 74HC595

Integrated circuit (integrated circuit) is a kind of miniature electronic device or component, which is represented by the letter "IC" in the circuit.

A certain process is used to interconnect the transistors, resistors, capacitors, inductors and other components and wiring required in a circuit, fabricate on a small or several small semiconductor wafers or dielectric substrates, and then package them in a package , it has become a micro-structure with the required circuit functions; all of the components have been structured as a whole, making electronic components a big step towards micro-miniaturization, low power consumption, intelligence and high reliability.

The inventors of integrated circuits are Jack Kilby (integrated circuits based on germanium (Ge)) and Robert Norton Noyce (integrated circuits based on silicon (Si)).

This kit is equipped with an IC, 74HC595, which can greatly save the use of GPIO pins. Specifically, it can replace 8 pins for digital signal output by writing an 8-bit binary number.

- Binary number - Wikipedia
- *74HC595*

Required Components

In this project, we need the following components.

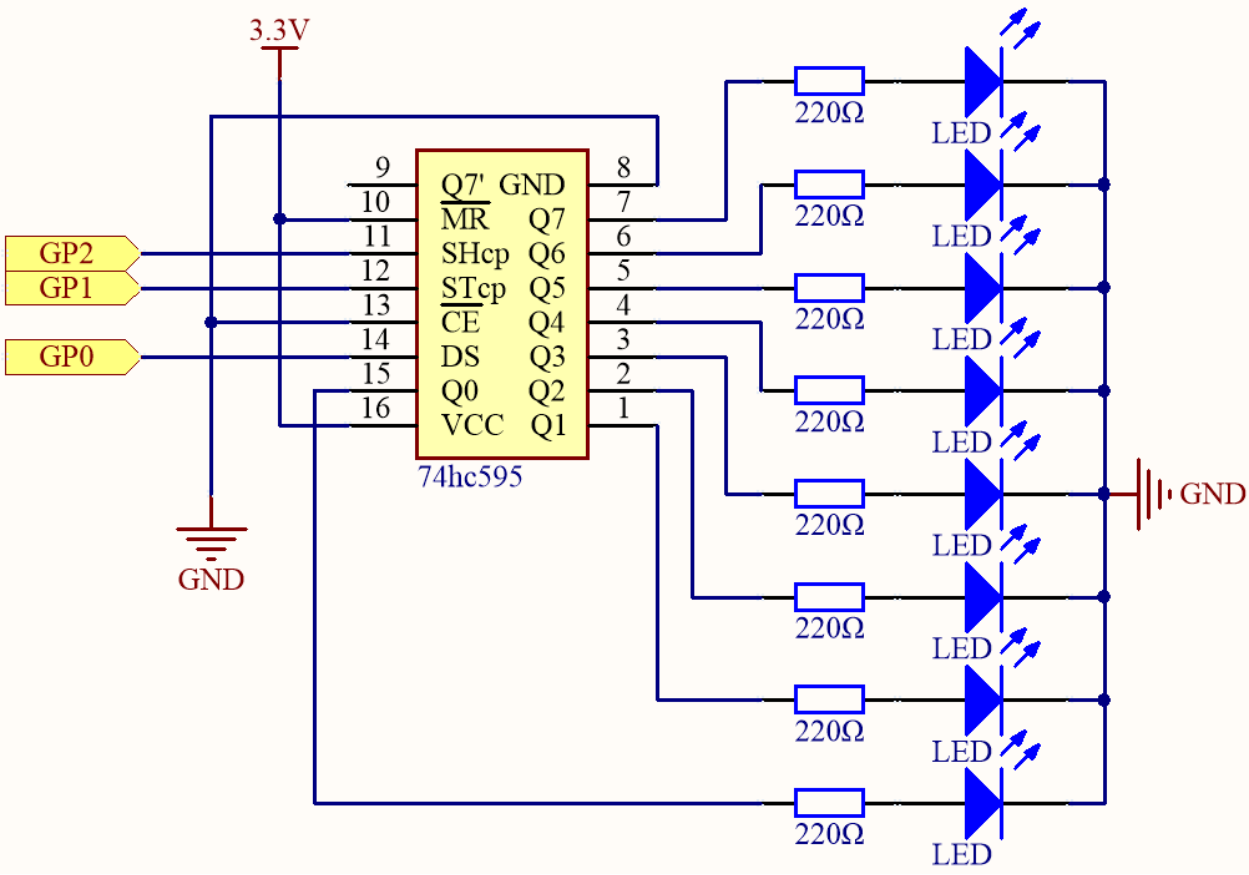
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	8(220)	
6	*LED*	8	
7	*74HC595*	1	

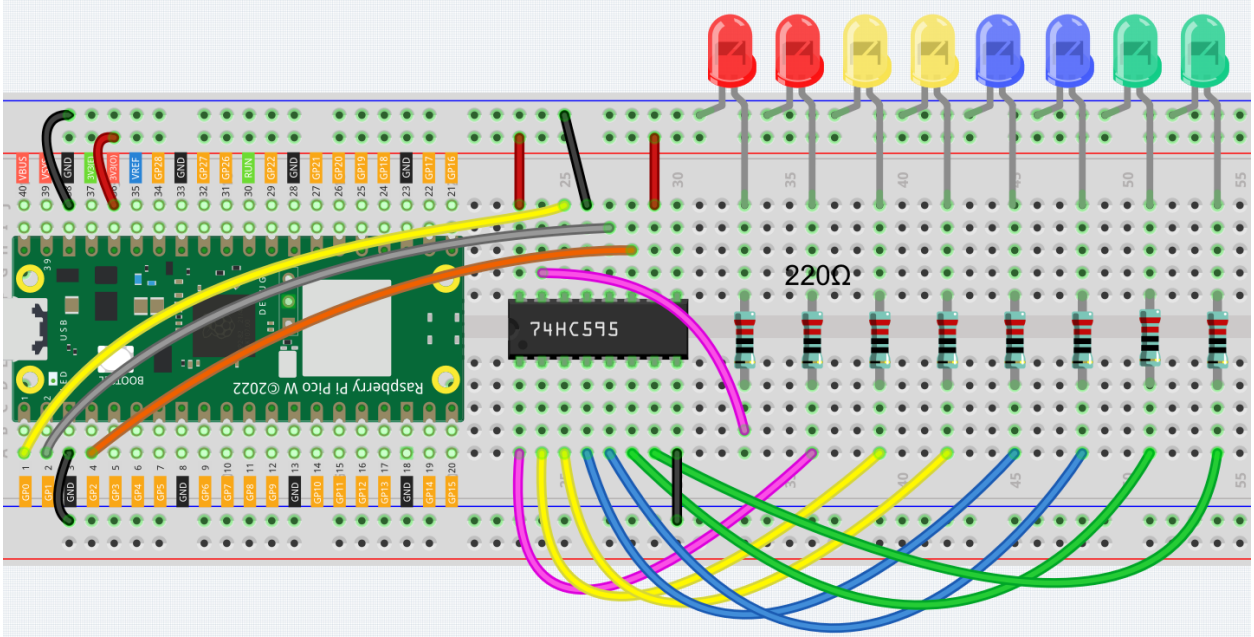
Schematic



- When MR (pin10) is high level and OE (pin13) is low level, data is input in the rising edge of SHcp and goes to the memory register through the rising edge of SHcp.
- If the two clocks are connected together, the shift register is always one pulse earlier than the memory register.
- There is a serial shift input pin (Ds), a serial output pin (Q) and an asynchronous reset button (low level) in the memory register.
- The memory register outputs a Bus with a parallel 8-bit and in three states.

- When OE is enabled (low level), the data in memory register is output to the bus(Q0 ~ Q7).

Wiring



Code

Note:

- Open the `5.1_microchip_74hc595.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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):
        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)
```

(continues on next page)

(continued from previous page)

```
    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)
```

When the program is running, num will be written into the 74HC595 chip as an eight-bit binary number to control the on and off of the 8 LEDs. We can see the current value of num in the shell.

How it works?

hc595_shift() will make 74HC595 output 8 digital signals. It outputs the last bit of the binary number to Q0, and the output of the first bit to Q7. In other words, writing the binary number "00000001" will make Q0 output high level and Q1~Q7 output low level.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

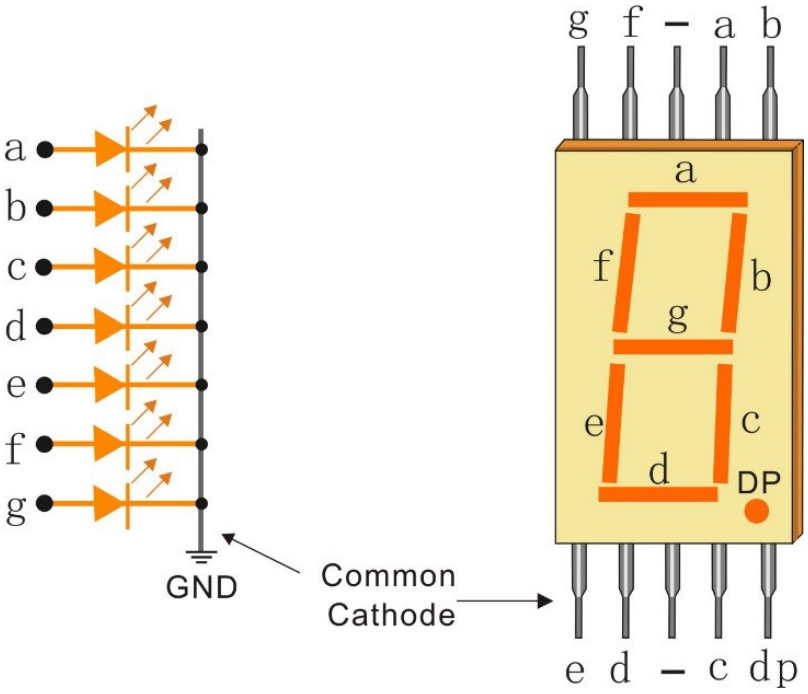
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.34 5.2 Number Display

7 Segment Display can be seen everywhere in life. For example, on an air conditioner, it can be used to display temperature; on a traffic indicator, it can be used to display a timer.

The 7 Segment Display is essentially a device packaged by 8 LEDs, of which 7 strip-shaped LEDs form an "8" shape, and there is a slightly smaller dotted LED as a decimal point. These LEDs are marked as a, b, c, d, e, f, g, and dp. They have their own anode pins and share cathodes. Their pin locations are shown in the figure below.



This means that it needs to be controlled by 8 digital signals at the same time to fully work and the 74HC595 can do this.

- *7-segment Display*

Required Components

In this project, we need the following components.

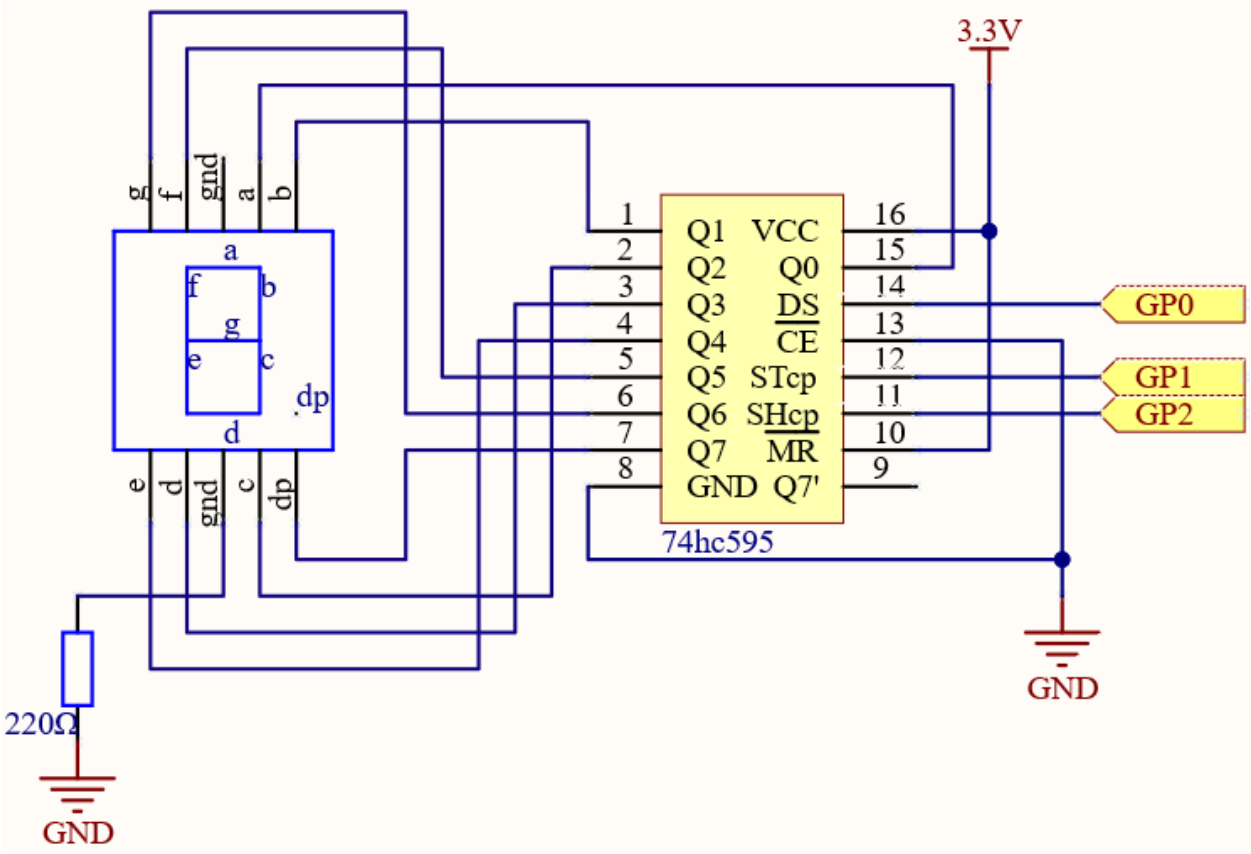
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(220)	
6	*7-segment Display*	1	
7	*74HC595*	1	

Schematic

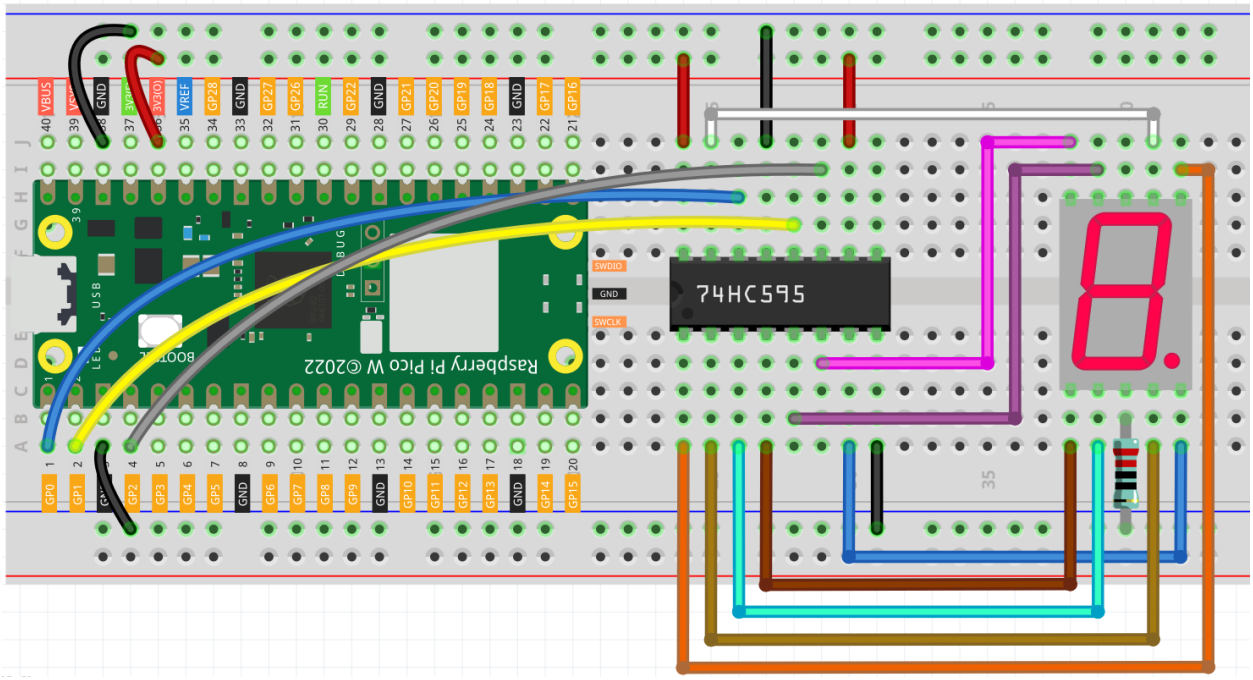


Here the wiring principle is basically the same as *5.1 Microchip - 74HC595*, the only difference is that Q0-Q7 are connected to the a ~ g pins of the 7 Segment Display.

Table 1: Wiring

74HC595	*LED* Segment Display
Q0	a
Q1	b
Q2	c
Q3	d
Q4	e
Q5	f
Q6	g
Q7	dp

Wiring



Code

Note:

- Open the `5.2_number_display.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)
```

(continues on next page)

(continued from previous page)

```
    time.sleep_ms(5)
    rclk.high()
    time.sleep_ms(5)

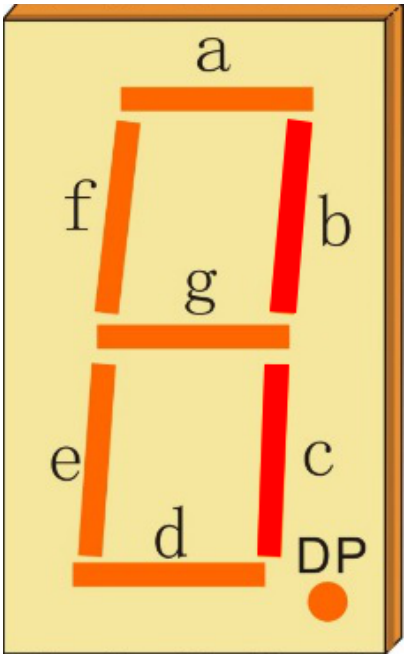
while True:
    for num in range(10):
        hc595_shift(SEGCODE[num])
        time.sleep_ms(500)
```

When the program is running, you will be able to see the LED Segment Display display 0~9 in sequence.

How it works?

`hc595_shift()` will make 74HC595 output 8 digital signals. It outputs the last bit of the binary number to Q0, and the output of the first bit to Q7. In other words, writing the binary number "00000001" will make Q0 output high level and Q1~Q7 output low level.

Suppose that the 7-segment Display display the number "1", we need to write a high level for b, c, and write a low level for a, d, e, f, g, and dg.



That is, the binary number "00000110" needs to be written. For readability, we will use hexadecimal notation as "0x06".

- Hexadecimal
- BinaryHex Converter

Similarly, we can also make the LED Segment Display display other numbers in the same way. The following table shows the codes corresponding to these numbers.

Table 2: Glyph Code

Numbers	Binary Code	Hex Code
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

Write these codes into `hc595_shift()` to make the LED Segment Display display the corresponding numbers.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.35 5.3 Time Counter

4-digit 7-segment display consists of four 7-segment displays working together.

The 4-digtal 7-segment display works independently. It uses the principle of human visual persistence to quickly display the characters of each 7-segment in a loop to form continuous strings.

For example, when "1234" is displayed on the display, "1" is displayed on the first 7-segment, and "234" is not displayed. After a period of time, the second 7-segment shows "2", the 1st 3th 4th of 7-segment does not show, and so on, the four digital display show in turn. This process is very short (typically 5ms), and because of the optical afterglow effect and the principle of visual residue, we can see four characters at the same time.

Required Components

In this project, we need the following components.

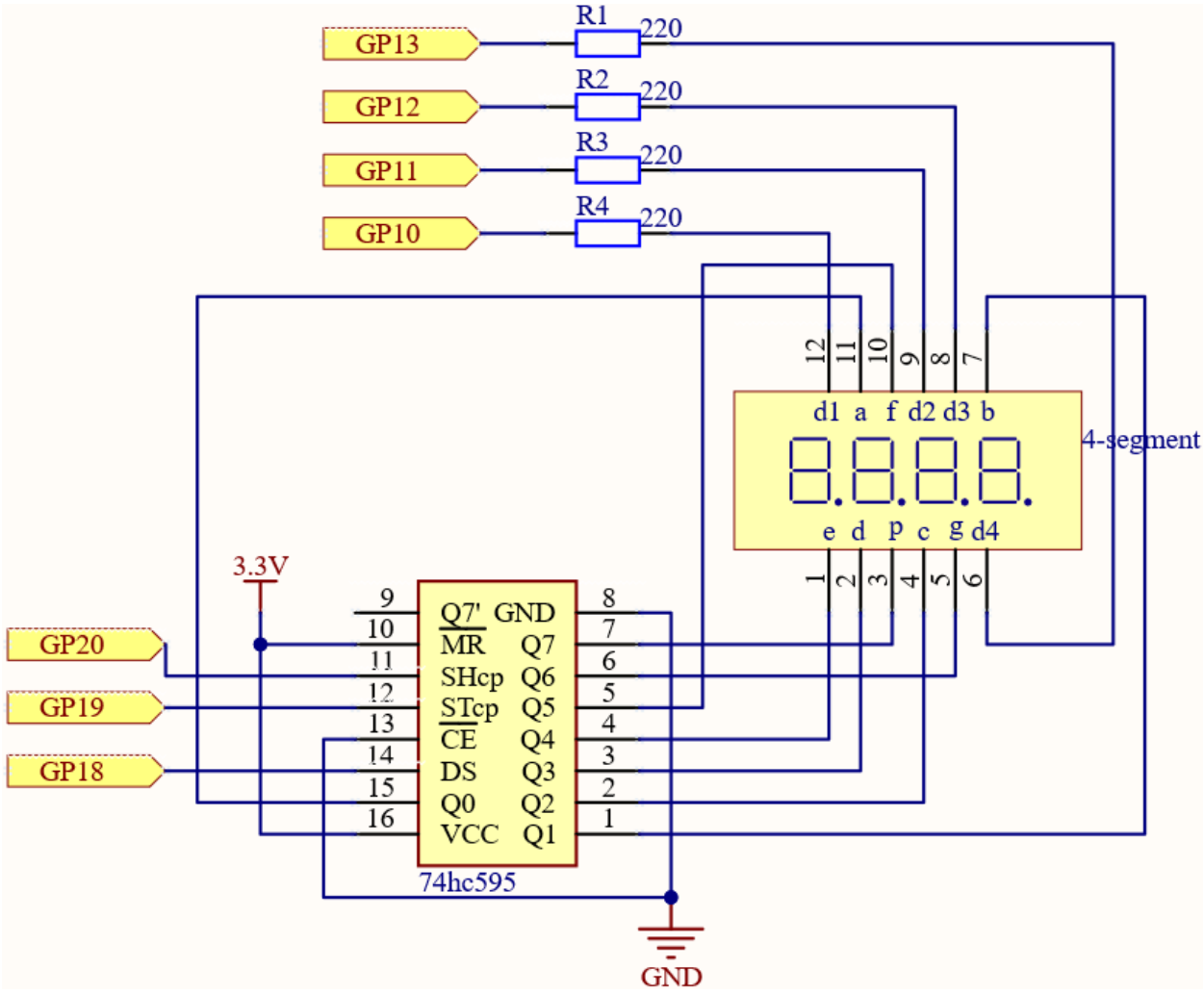
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	4(220)	
6	*4-Digit 7-Segment Display*	1	
7	*74HC595*	1	

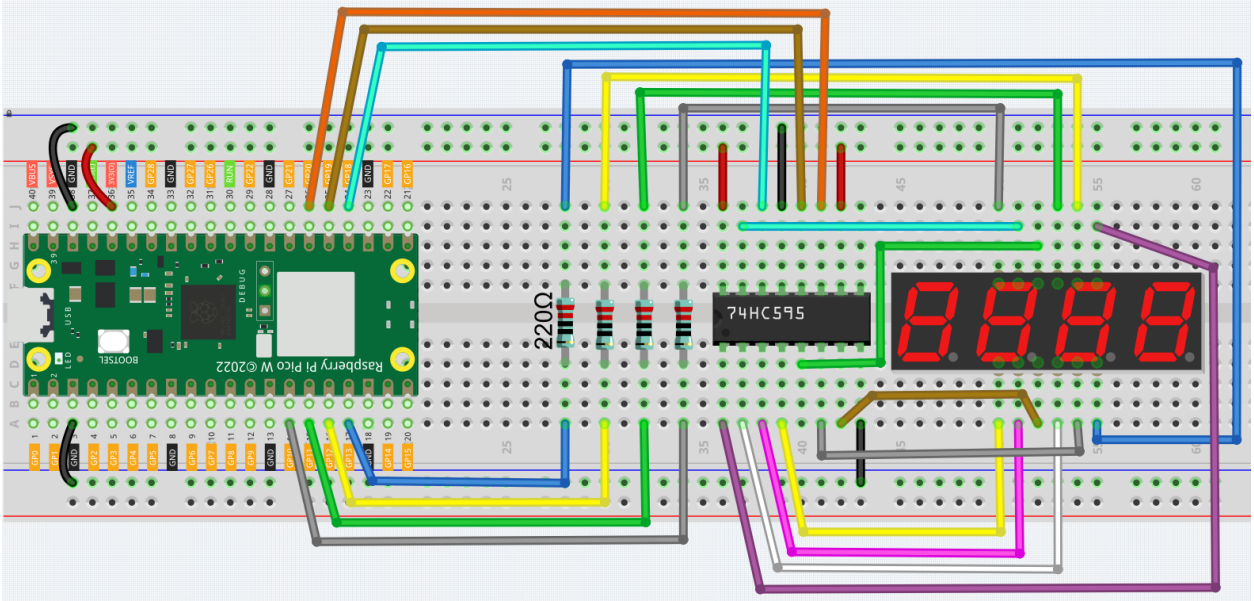
Schematic



Here the wiring principle is basically the same as *5.1 Microchip - 74HC595*, the only difference is that Q0-Q7 are connected to the a ~ g pins of the 4-digit 7-segment display.

Then G10 ~ G13 will select which 7-segment display to work.

Wiring



Code

Note:

- Open the `5.3_time_counter.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)

timerStart=time.ticks_ms()

def timer1():
    return int((time.ticks_ms()-timerStart)/1000)

def pickDigit(digit):
```

(continues on next page)

(continued from previous page)

```
    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])

    pickDigit(2)
    hc595_shift(SEGCODE[count%1000//100])

    pickDigit(3)
    hc595_shift(SEGCODE[count%10000//1000])
```

After the program is run, you will see the 4-digit 7-segment display become a counter and the number increases by 1 per second.

How it works?

Writing signals to each 7-segment display is done in the same way as *5.2 Number Display*, using the `hc595_shift()` function. The core point of the 4-digit 7-segment display is to selectively activate each 7-segment display. The code associated with this is as follows.

```
placePin = []
pin = [13,12,11,10]
for i in range(4):
    placePin.append(None)
    placePin[i] = machine.Pin(pin[i], machine.Pin.OUT)
```

(continues on next page)

(continued from previous page)

```
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)
```

Here, four pins (GP10, GP11, GP12, GP13) are used to control each bit of the 4-digit 7-segment display individually. When the state of these pins is `0`, the corresponding 7-segment display is active; when the state is 1, the opposite is true.

Here the `pickDigit(digit)` function is used to unable all four digits and then enable a particular digit individually. After that, `hc595_shift()` is used to write the corresponding 8 bits code for the 7-segment display.

The 4-digit 7-segment display needs to be continuously activated in turn so that we can see it display four digits, which means that the main program cannot easily add code that would affect the timing. However, we need to add a timing function to this example, and if we add a `sleep(1)`, we will know that it has four digits. we will see through the illusion of 4-digit 7-segment display working at the same time, exposing the fact that only one 7-segment display is illuminated at a time. Then, using the `time.ticks_ms()` function in the `time` library is an excellent way to do this.

```
import time

timerStart=time.ticks_ms()

def timer1():
    return int((time.ticks_ms()-timerStart)/1000)

while True:
    count = timer1()
```

The `time.ticks_ms()` function gets a (non-explicit) time, and we record the first time value we get as `timerStart`. Subsequently, when the time is needed, the `time.ticks_ms()` function is called again, and the value is subtracted from `timerStart` to get how long the program has been running (in milliseconds).

Finally, convert and output this time value to the 4-digit 7-segment display and you're done.

- Time - MicroPython Docs

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.36 5.4 8x8 Pixel Graphics

LED matrix is a low-resolution dot-matrix display. it uses an array of light-emitting diodes as pixels for patterned displays.

They are bright enough to be visible in outdoor sunlight, and you can see them on some stores, billboards, signs, and variable message displays (such as those on public transit vehicles).

Used in this kit is an 8x8 dot matrix with 16 pins. Their anodes are connected in rows and their cathodes are connected in columns (at the circuit level), which together control these 64 LEDs.

To light the first LED, you should provide a high level for Row1 and a low level for Col1. To light the second LED, it should provide a high level for Row1, a low level for Col2, and so on. By controlling the current through each pair of rows and columns, each LED can be controlled individually to display characters or pictures.

- *LED Dot Matrix*
- *74HC595*

Required Components

In this project, we need the following components.

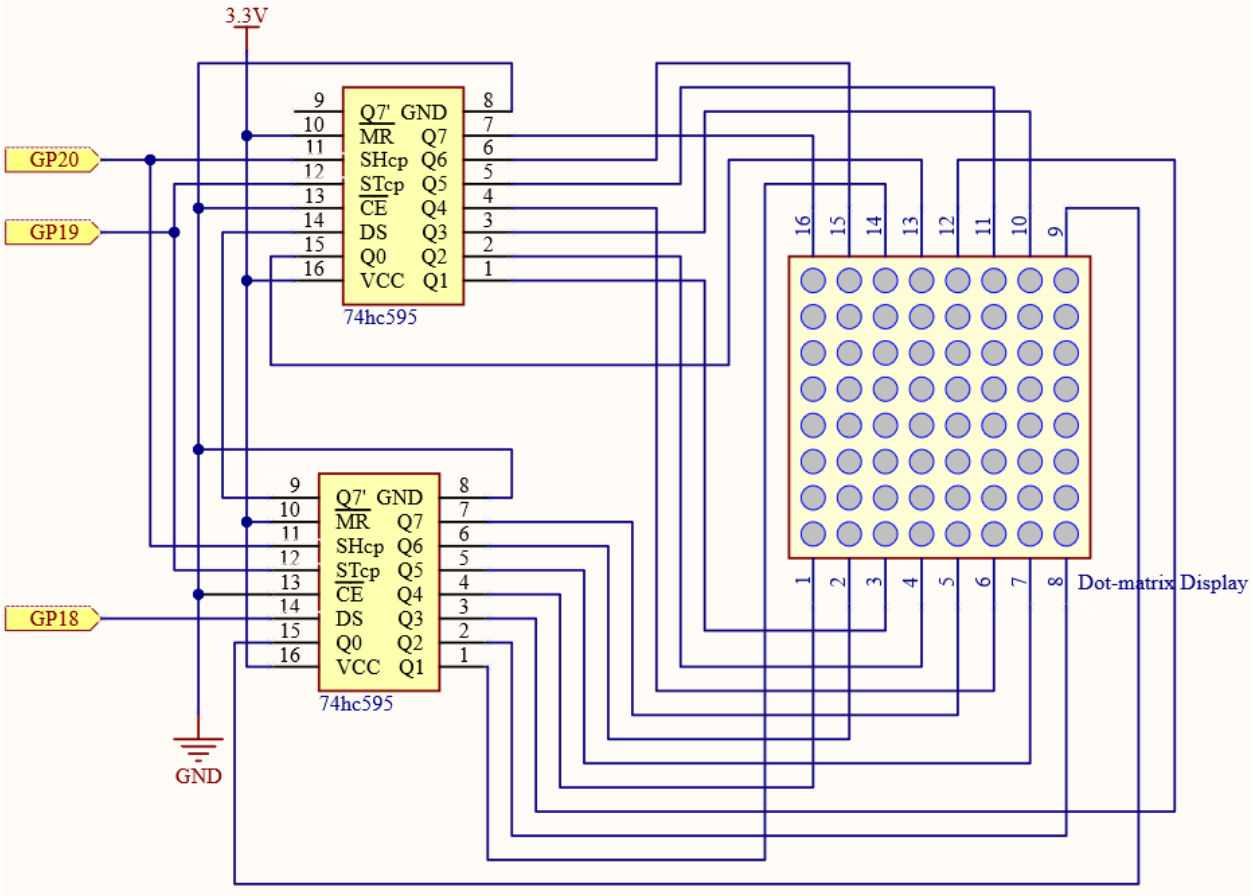
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*LED Dot Matrix*	1	
6	*74HC595*	2	

Schematic



The 8x8 dot matrix is controlled by two 74HC595 chips, one controlling the rows and one controlling the columns, while these two chips share G18~G20, which can greatly save the I/O ports of the Pico W board.

Pico W needs to output a 16-bit binary number at a time, the first 8 bits are given to the 74HC595 which controls the rows, and the last 8 bits are given to the 75HC595 which controls the columns, so that the dot matrix can display a specific pattern.

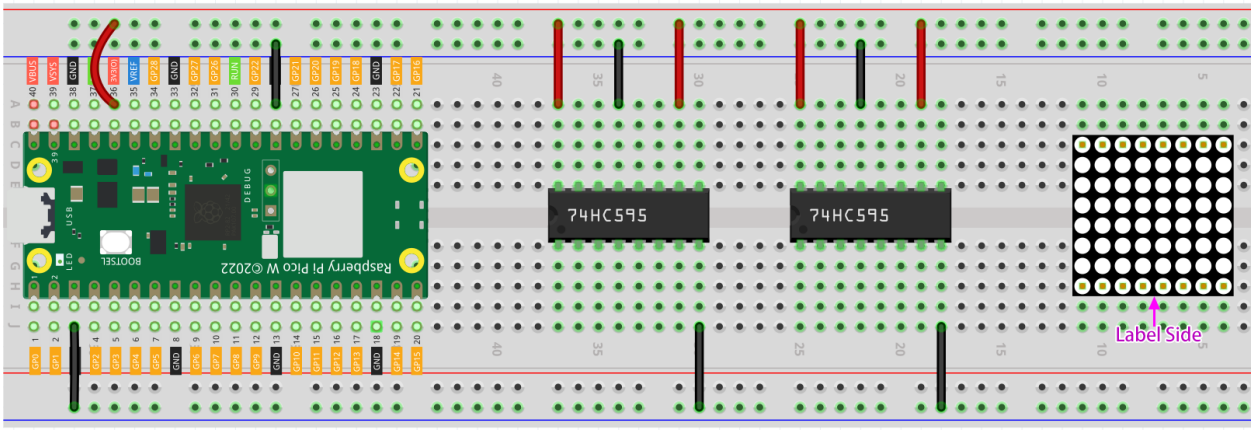
Q7': Series output pin, connected to DS of another 74HC595 to connect multiple 74HC595s in series.

Wiring

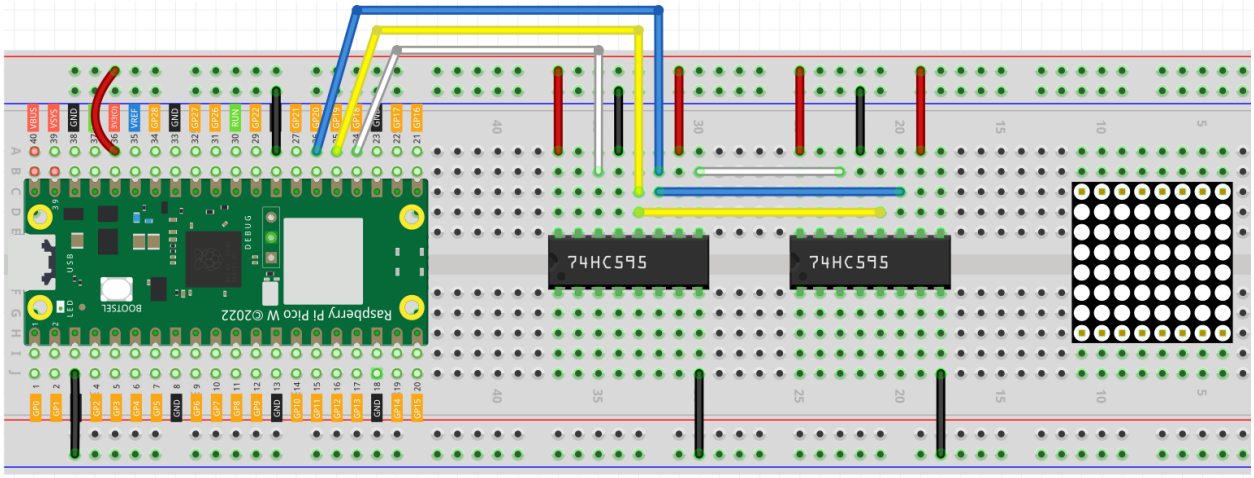
Build the circuit. Since the wiring is complicated, let's make it step by step.

Step 1: First, insert the Pico W, the LED dot matrix and two 74HC595 chips into breadboard. Connect the 3.3V and GND of the Pico W to holes on the two sides of the board, then hook up pin16 and 10 of the two 74HC595 chips to VCC, pin 13 and pin 8 to GND.

Note: In the Fritzing image above, the side with label is at the bottom.

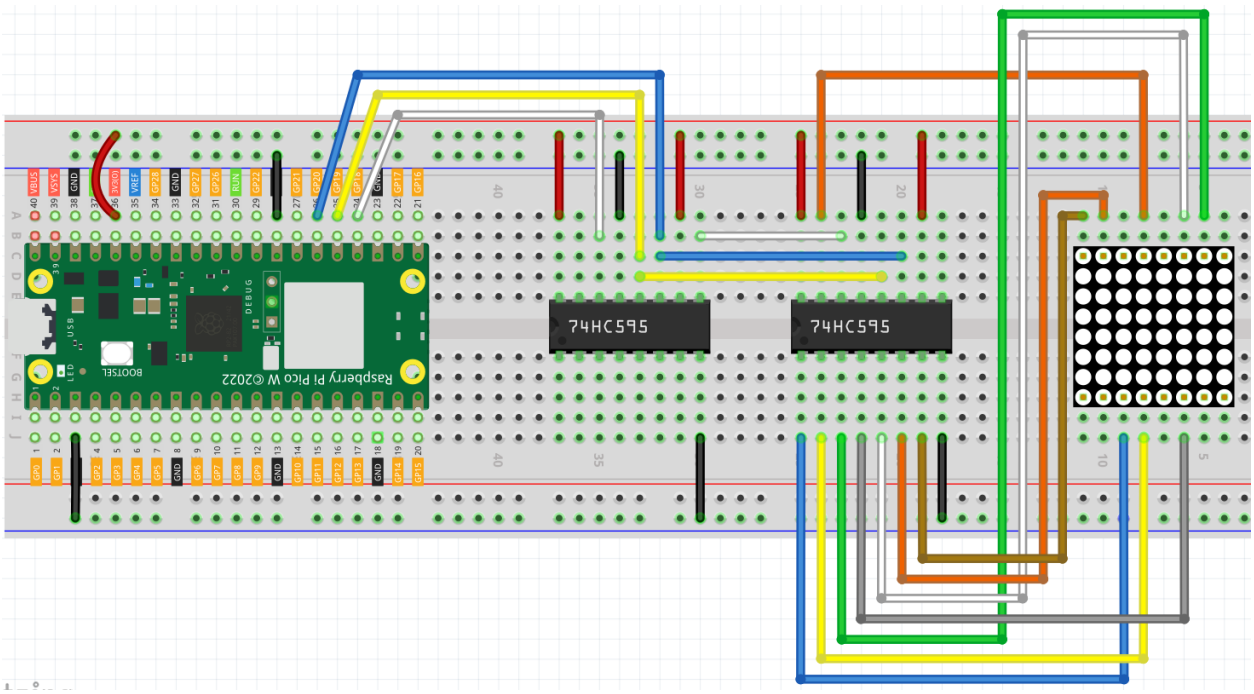


Step 2: Connect pin 11 of the two 74HC595 together, and then to GP20; then pin 12 of the two chips, and to GP19; next, pin 14 of the 74HC595 on the left side to GP18 and pin 9 to pin 14 of the second 74HC595.



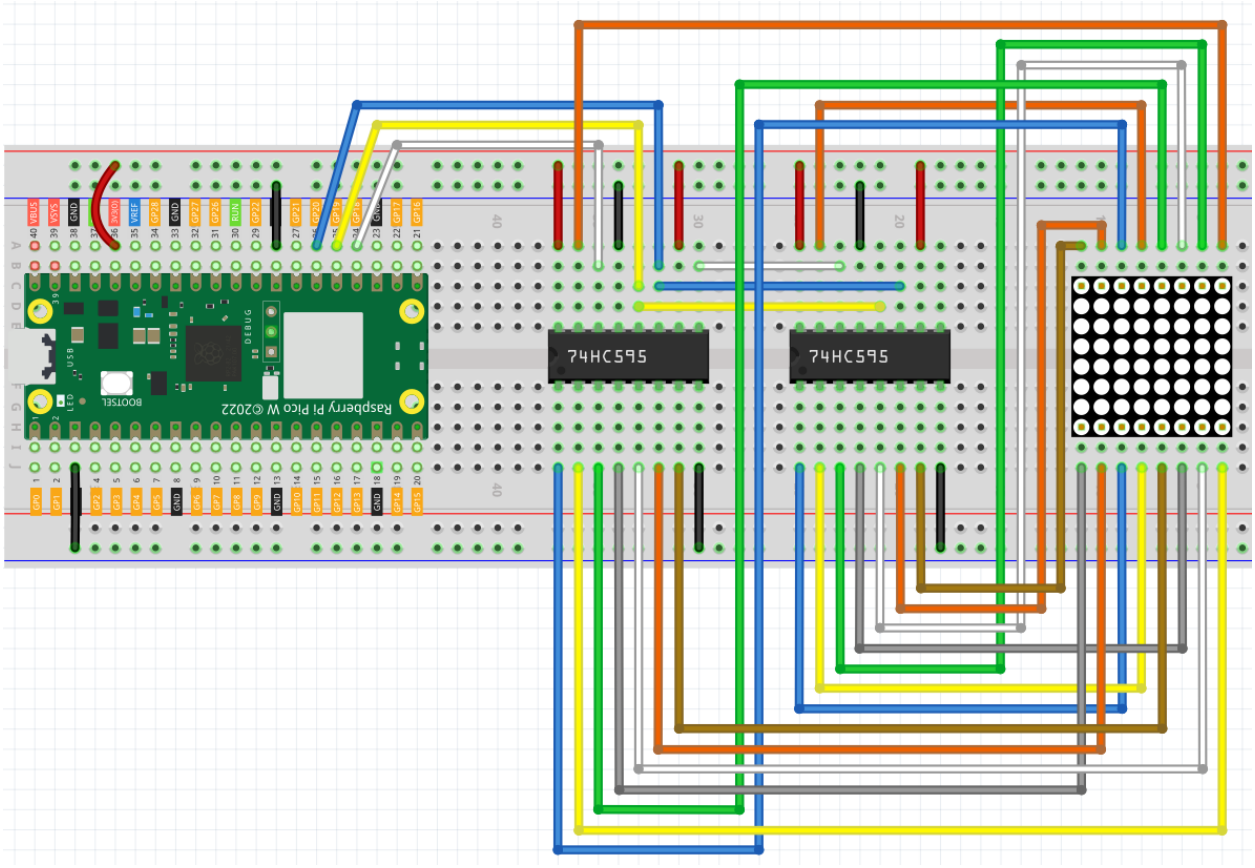
Step 3: The 74HC595 on the right side is to control columns of the LED dot matrix. See the table below for the mapping. Therefore, Q0-Q7 pins of the 74HC595 are mapped with pin 13, 3, 4, 10, 6, 11, 15, and 16 respectively.

74HC595	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
LED Dot Matrix	13	3	4	10	6	11	15	16



Step 4: Now connect the ROWs of the LED dot matrix. The 74HC595 on the left controls ROW of the LED dot matrix. See the table below for the mapping. We can see, Q0-Q7 of the 74HC595 on the left are mapped with pin 9, 14, 8, 12, 1, 7, 2, and 5 respectively.

74HC595	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
LED Dot Matrix	9	14	8	12	1	7	2	5



Code

Note:

- Open the `5.4_8x8_pixel_graphics.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)


glyph = [0xFF,0xBB,0xD7,0xEF,0xD7,0xBB,0xFF,0xFF]

# Shift the data to 74HC595
def hc595_in(dat):
    for bit in range(7,-1, -1):
        srclk.low()
        time.sleep_us(30)
```

(continues on next page)

(continued from previous page)

```
        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()
```

Once the program is running, you will see a **x** graphic displayed on the 8x8 dot matrix.

How it works?

Here we use two 74HC595s to provide signals for the rows and columns of the dot matrix. The method of providing signals is the same as `hc595_shift(dat)` in the previous chapters, but the difference is that here we need to write a 16-bit binary number at a time. So we split `hc595_shift(dat)` into two functions `hc595_in(dat)` and `hc595_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()
```

Then, call `hc595_in(dat)` twice in the main loop, write two 8-bit binary numbers and then call `hc595_out()` so that a pattern can be displayed.

However, since the LEDs in the dot matrix use common poles, controlling multiple rows/multiple columns at the same time will interfere with each other (e.g., if you light up (1,1) and (2,2) at the same time, (1,2) and (2,1) will inevitably be lit up together). Therefore, it is necessary to activate one column (or one row) at a time, cycle 8 times, and use the residual image principle to make the human eye merge 8 patterns, so as to get a pair of patterns containing 8x8 amount of information.

```
while True:
    for i in range(0,8):
        hc595_in(glyph[i])
        hc595_in(0x80>>i)
        hc595_out()
```

In this example, the main function nests a `for` loop, and when `i` is 1, only the first line is activated (the chip in the control line gets the value `0x80`) and the image of the first line is written. When `i` is 2, the second line is activated (the chip of the control line gets the value `0x40`) and the image of the second line is written. And so on, completing 8 outputs.

Incidentally, like the 4-digit 7-segment display, it has to maintain the refresh rate to prevent flickering by the human eye, so the extra `sleep()` in the main loop should be avoided as much as possible.

Learn More

Try replacing `glyph` with the following array and see what comes up!

```
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]
```

Or, you can try drawing your own graphics.

6. Advanced

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.37 6.1 Measuring Distance

The ultrasonic sensor module works on the principle of sonar and radar systems for determining the distance to an object.

- *Ultrasonic Module*

Required Components

In this project, we need the following components.

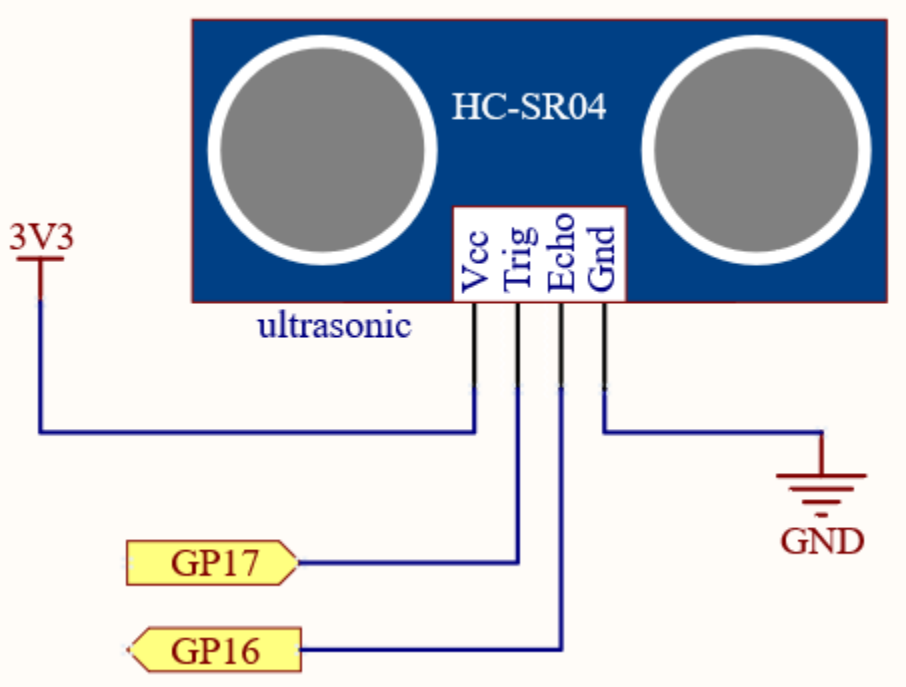
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

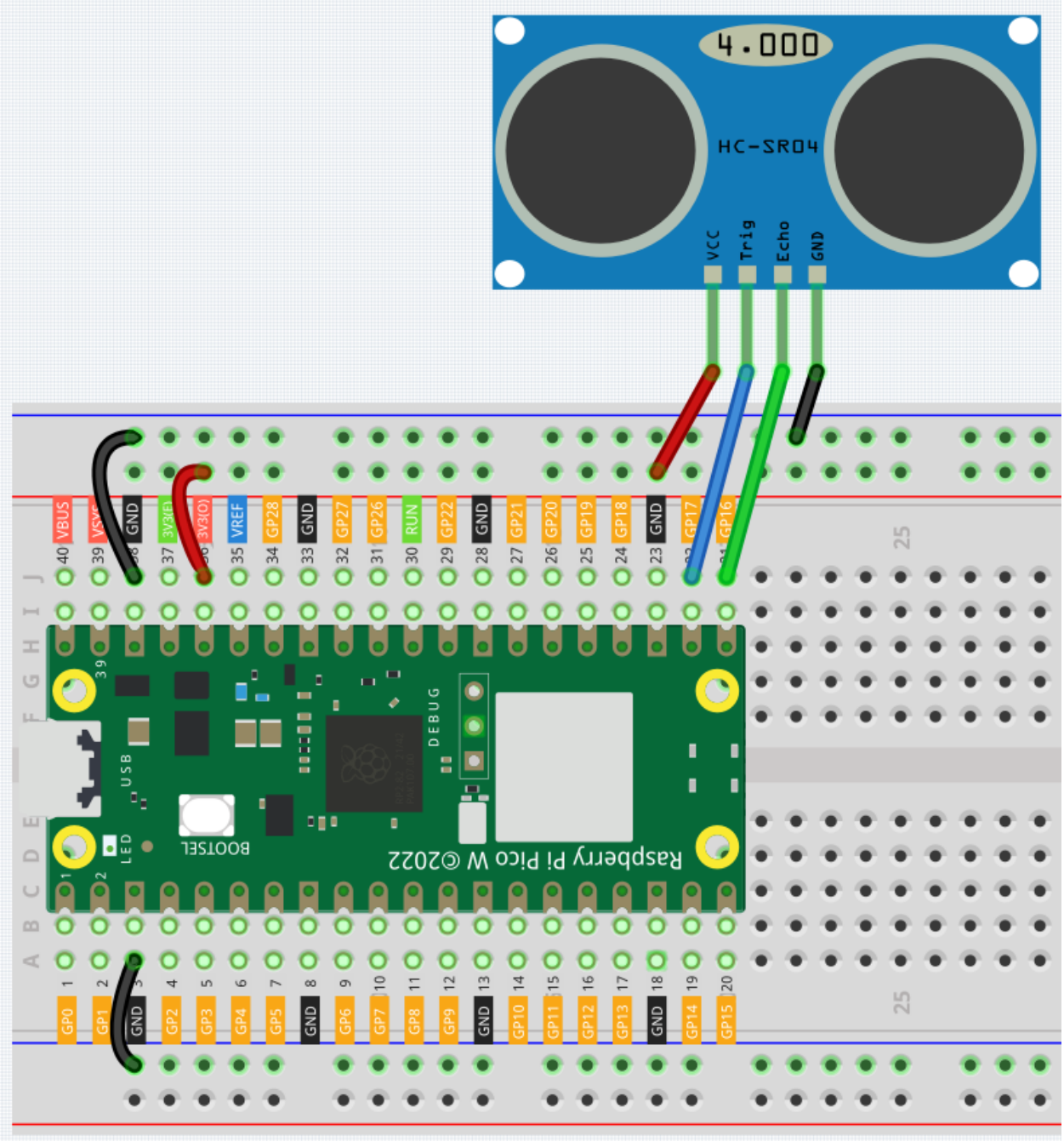
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Ultrasonic Module*	1	

Schematic



Wiring



Code

Note:

- Open the `6.1_measuring_distance.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)
```

Once the program is running, the Shell will print out the distance of the ultrasonic sensor from the obstacle ahead.

How it works?

Ultrasonic sensors produce high frequency sound waves (ultrasonic waves) emitted by the transmitting probe. When this ultrasonic wave hits an object, it is reflected as an echo, which is detected by the receiving probe. By calculating the time from transmission to reception, the distance can be calculated. Based on this principle, the function `distance()` can be derived.

```
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
```

- Among them, the first few lines are used to transmit a 10us ultrasonic wave.

```
TRIG.low()
time.sleep_us(2)
```

(continues on next page)

(continued from previous page)

```
TRIG.high()
time.sleep_us(10)
TRIG.low()
```

- Then, the program is paused and the current time is recorded when the ultrasonic wave has been emitted.

```
while not ECHO.value():
    pass
time1 = time.ticks_us()
```

- Subsequently, the program is suspended again. After the echo is received, the current time is recorded once again.

```
while ECHO.value():
    pass
time2 = time.ticks_us()
```

- Finally, based on the time difference between the two recordings, the speed of sound (340m/s) is multiplied by the time to obtain double the distance between the ultrasonic module and the obstacle (i.e., one round trip of the ultrasonic waves from the module to the obstacle). Converting the units to centimeters gives us the return value we need.

```
during = time.ticks_diff(time2,time1)
return during * 340 / 2 / 10000
```

Note that the ultrasonic sensor will pause the program when it is working, which may cause some lagging when writing complex projects.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

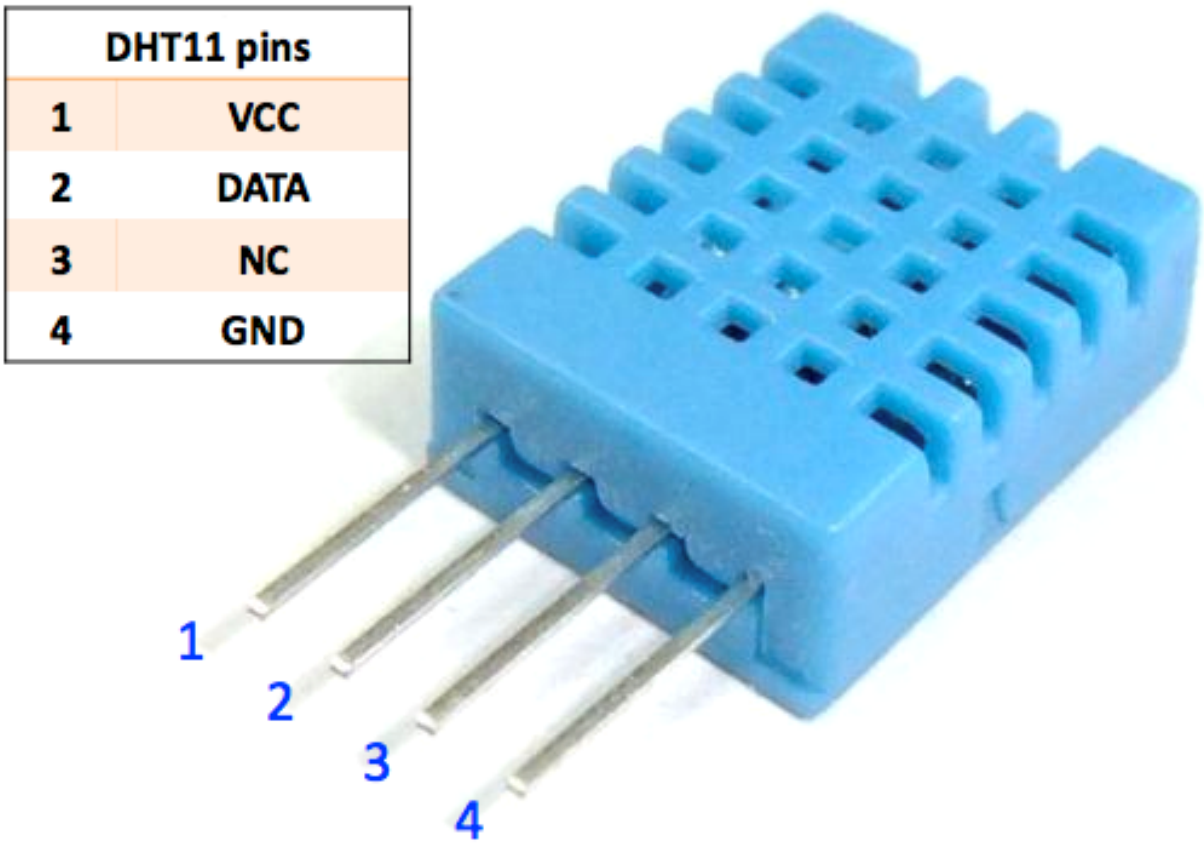
Ready to explore and create with us? Click [] and join today!

4.38 6.2 Temperature - Humidity

Humidity and temperature are closely related from the physical quantity itself to the actual people's life. The temperature and humidity of human environment will directly affect the thermoregulatory function and heat transfer effect of human body. It will further affect the thinking activity and mental state, thus affecting the efficiency of our study and work.

Temperature is one of the seven basic physical quantities in the International System of Units, which is used to measure the degree of hot and cold of an object. Celsius is one of the more widely used temperature scales in the world, expressed by the symbol "℃".

Humidity is the concentration of water vapor present in the air. The relative humidity of air is commonly used in life and is expressed in %RH. Relative humidity is closely related to temperature. For a certain volume of sealed gas, the higher the temperature, the lower the relative humidity, and the lower the temperature, the higher the relative humidity.



A basic digital temperature and humidity sensor, the **DHT11**, is provided in this kit. It uses a capacitive humidity sensor and thermistor to measure the surrounding air and outputs a digital signal on the data pins (no analog input pins are required).

- *DHT11 Humiture Sensor*

Required Components

In this project, we need the following components.

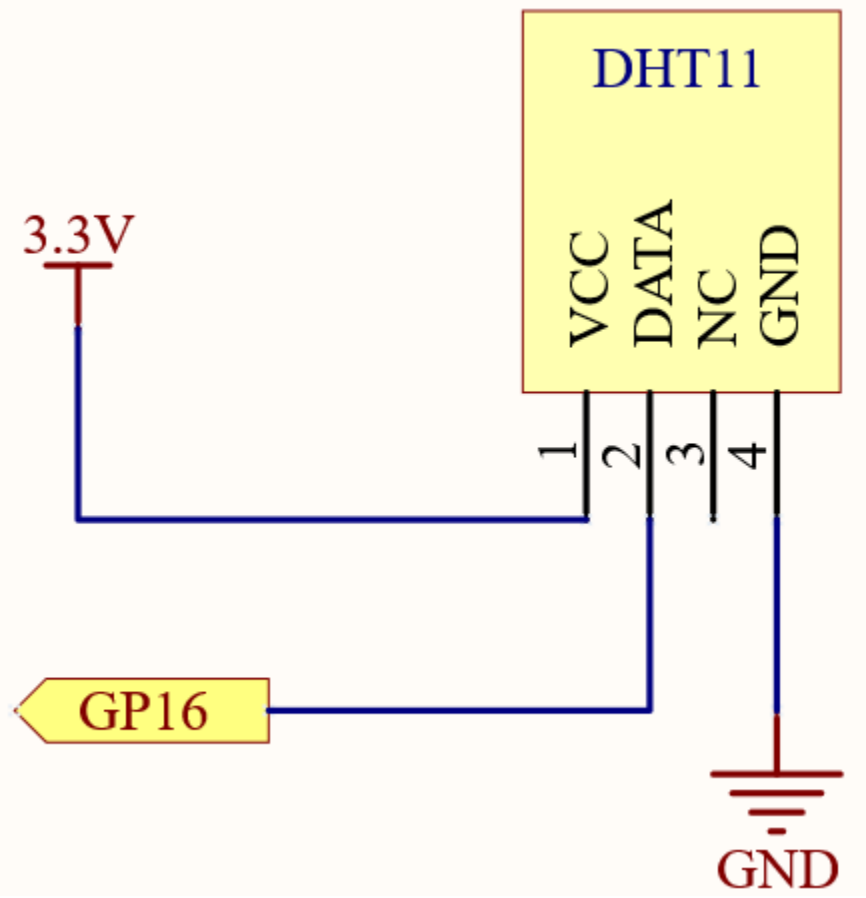
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

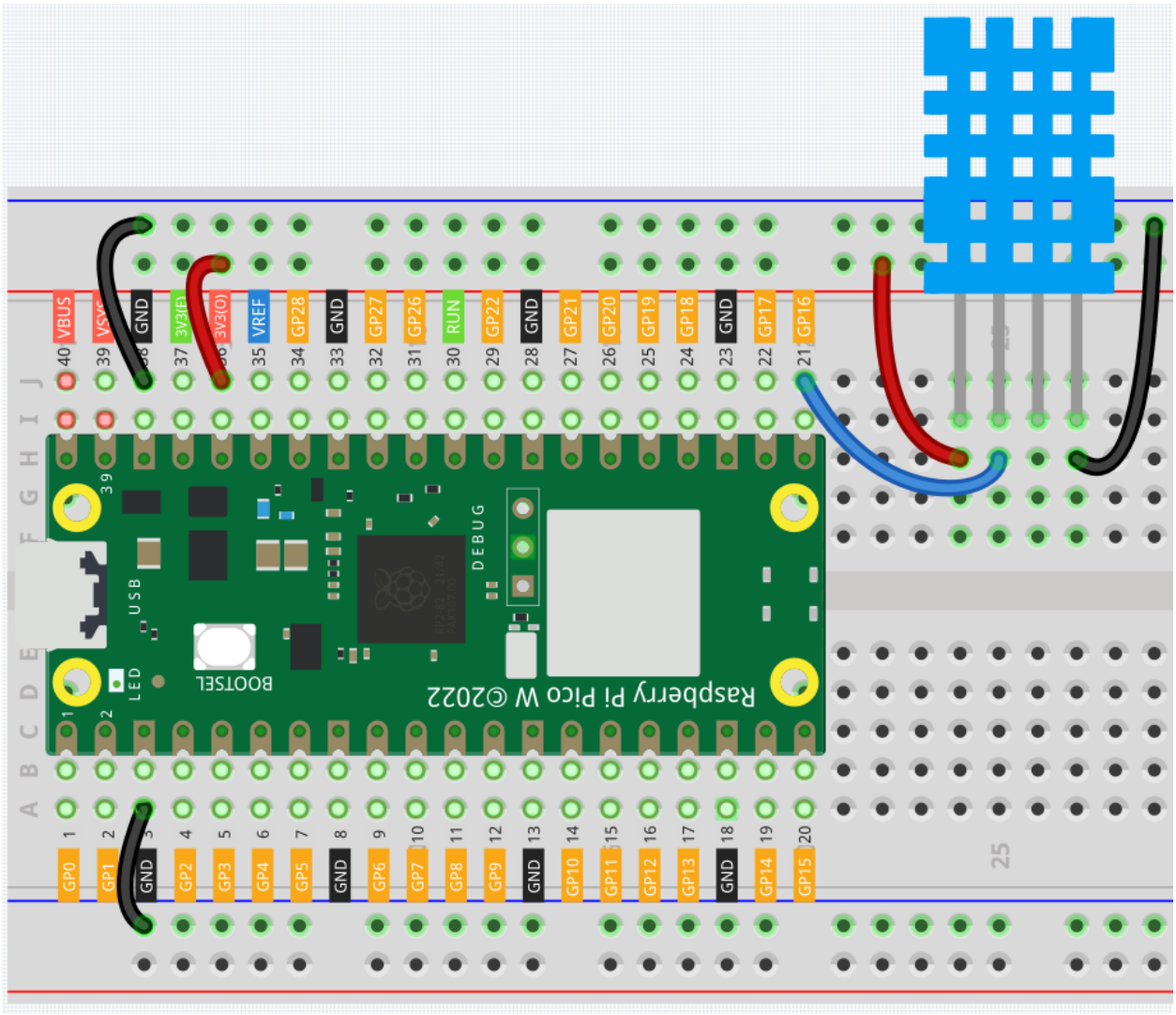
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*DHT11 Humiture Sensor*	1	

Schematic



Wiring



Code

Note:

- Open the `6.2_temperature_humidity.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the library called `dht.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *1.4 Upload the Libraries to Pico*.

```
from machine import Pin, I2C
import utime as time
from dht import DHT11, InvalidPulseCount

pin = Pin(16, Pin.IN, Pin.PULL_UP)
```

(continues on next page)

(continued from previous page)

```
sensor = DHT11(pin)
time.sleep(5)  # initial delay

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 ...')
```

After the code is run, you will see the Shell continuously print out the temperature and humidity, and as the program runs steadily, these two values will become more and more accurate.

How it works?

In the dht library, we have integrated the relevant functionality into the `DHT11` class.

```
from dht import DHT11, InvalidPulseCount
```

Initialize the `DHT11` object. This device only needs a digital input to be used.

```
pin = Pin(16, Pin.IN, Pin.PULL_UP)
sensor = DHT11(pin)
```

Use `sensor.measure()` to read the current temperature and humidity, which will be stored in `sensor.temperature`, `sensor.humidity`. They are then printed out. Finally the DHT11 sampling rate is 1HZ, a `time.sleep(1)` is needed in the loop.

```
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 ...')
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.

- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.39 6.3 6-axis Motion Tracking

The MPU-6050 is a 6-axis(combines 3-axis Gyroscope, 3-axis Accelerometer) motion tracking devices.

An accelerometer is a tool that measures proper acceleration.For example, an accelerometer at rest on the surface of the Earth will measure an acceleration due to Earth's gravity, straight upwards[3] (by definition) of g 9.81 m/s2.

Accelerometers have many uses in industry and science. For example: inertial navigation systems for aircraft and missiles, for keeping images on tablets and digital cameras vertical, etc.

Gyroscopes are used to measure orientation and angular velocity of a device or maintenance. Applications of gyroscopes include anti-rollover and airbag systems for automobiles, motion sensing systems for smart devices, attitude stabilization systems for drones, and more.

- *MPU6050 Module*

Required Components

In this project, we need the following components.

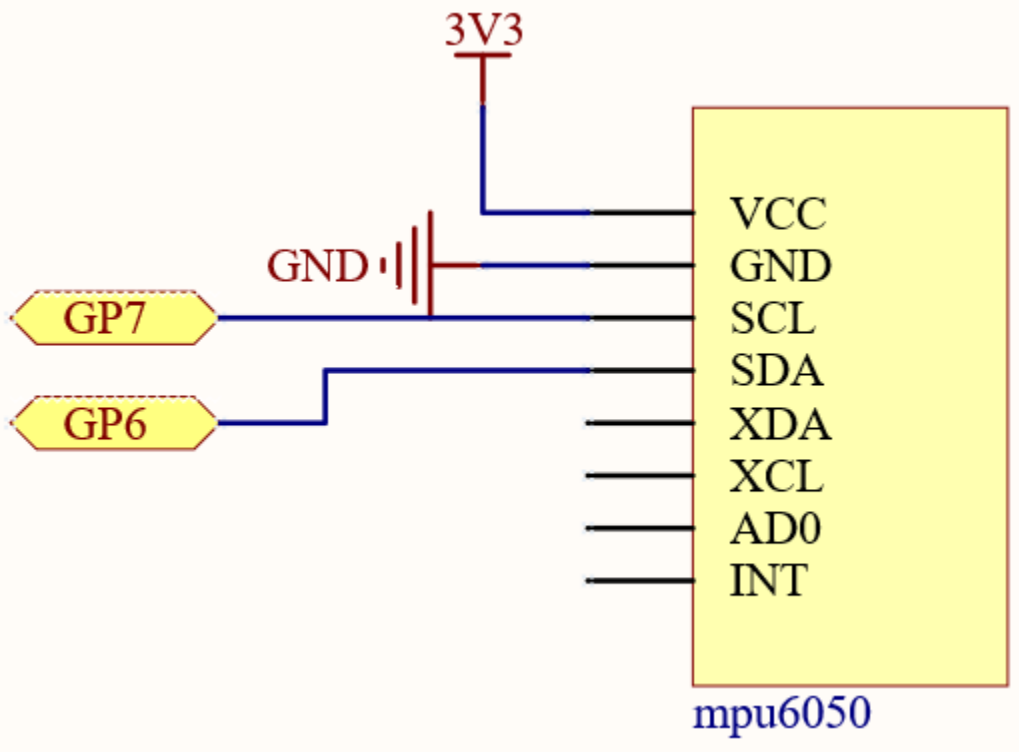
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

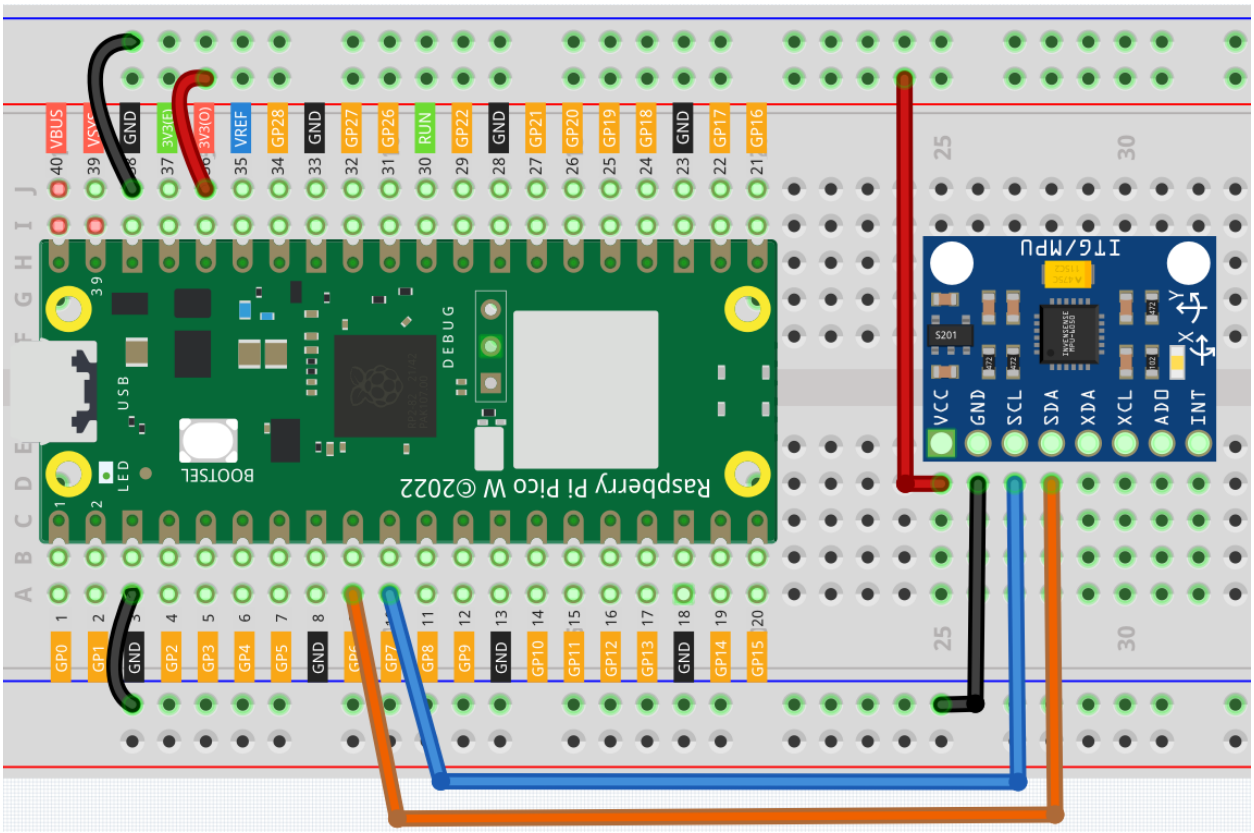
You can also buy them separately from the links below.

SN	COMPONENT	QUANTITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*MPU6050 Module*	1	

Schematic


3V3
GND
GP7
GP6
VCC
GND
SCL
SDA
XDA
XCL
AD0
INT
mpu6050

Wiring


VCC
GND
SCL
SDA
XDA
XCL
AD0
INT
ITG/MPU
Raspberry Pi Pico W ©2022
BOOTSEL
DEBUG
USB
LED

Code

Note:

- Open the `6.3_6axis_motion_tracking.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the `imu.py` and `vector3d.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *1.4 Upload the Libraries to Pico*.

```
from imu import MPU6050
from machine import I2C, Pin
import time

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)
```

After running the program, you can see the 3-axis accelerometer values and 3-axis gyroscope values cycling through the output. At this point you rotate the MPU6050 at random, and these values will appear to change accordingly. To make it easier to see the changes, you can comment out one of the print lines and concentrate on another set of data.

The unit of acceleration value is m/s^2, and the unit of gyroscope value is °/s.

How it works?

In the imu library, we have integrated the relevant functions into the `MPU6050` class. MPU6050 is an I2C module and requires a set of I2C pins to be defined for initialization.

```
from imu import MPU6050
from machine import I2C, Pin

i2c = I2C(1, sda=Pin(6), scl=Pin(7), freq=400000)
mpu = MPU6050(i2c)
```

Subsequently, you will be able to get real-time acceleration and angular velocity values in `mpu.accel.x`, `mpu.accel.y`, `mpu.accel.z`, `mpu.gyro.x`, `mpu.gyro.y`, `mpu.gyro.z`.

```
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)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.40 6.4 IR Remote Control

In consumer electronics, remote controls are used to operate devices such as televisions and DVD players. In some cases, remote controls allow people to operate devices that are out of their reach, such as central air conditioners.

IR Receiver is a component with photocell that is tuned to receive to infrared light. It is almost always used for remote control detection - every TV and DVD player has one of these in the front to receive for the IR signal from the clicker. Inside the remote control is a matching IR LED, which emits IR pulses to tell the TV to turn on, off or change channels.

- *Infrared Receiver*

Required Components

In this project, we need the following components.

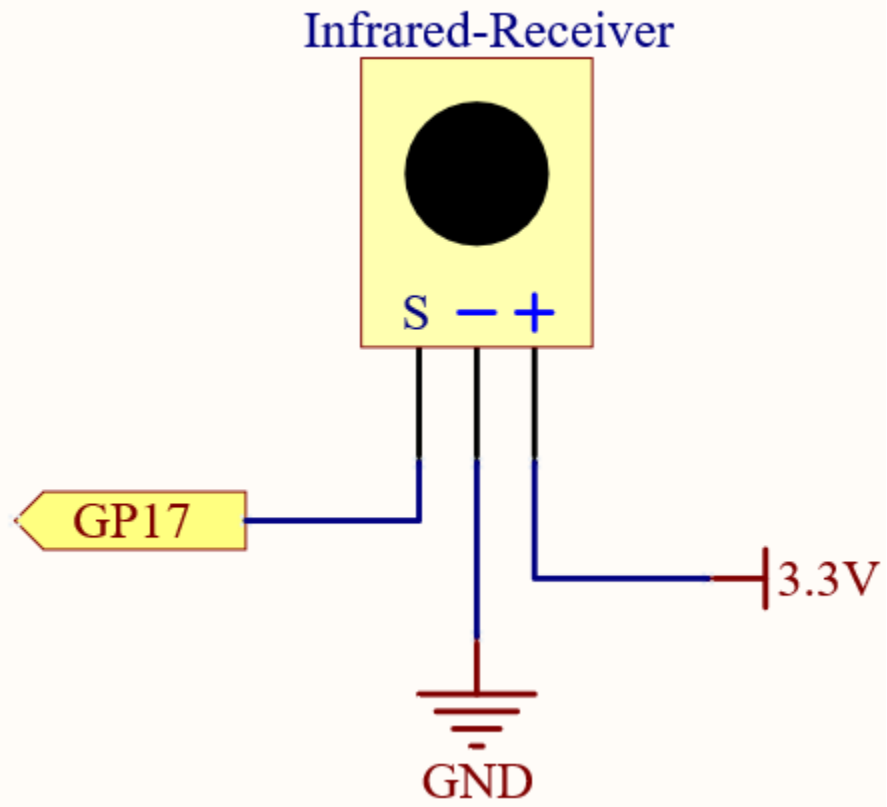
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

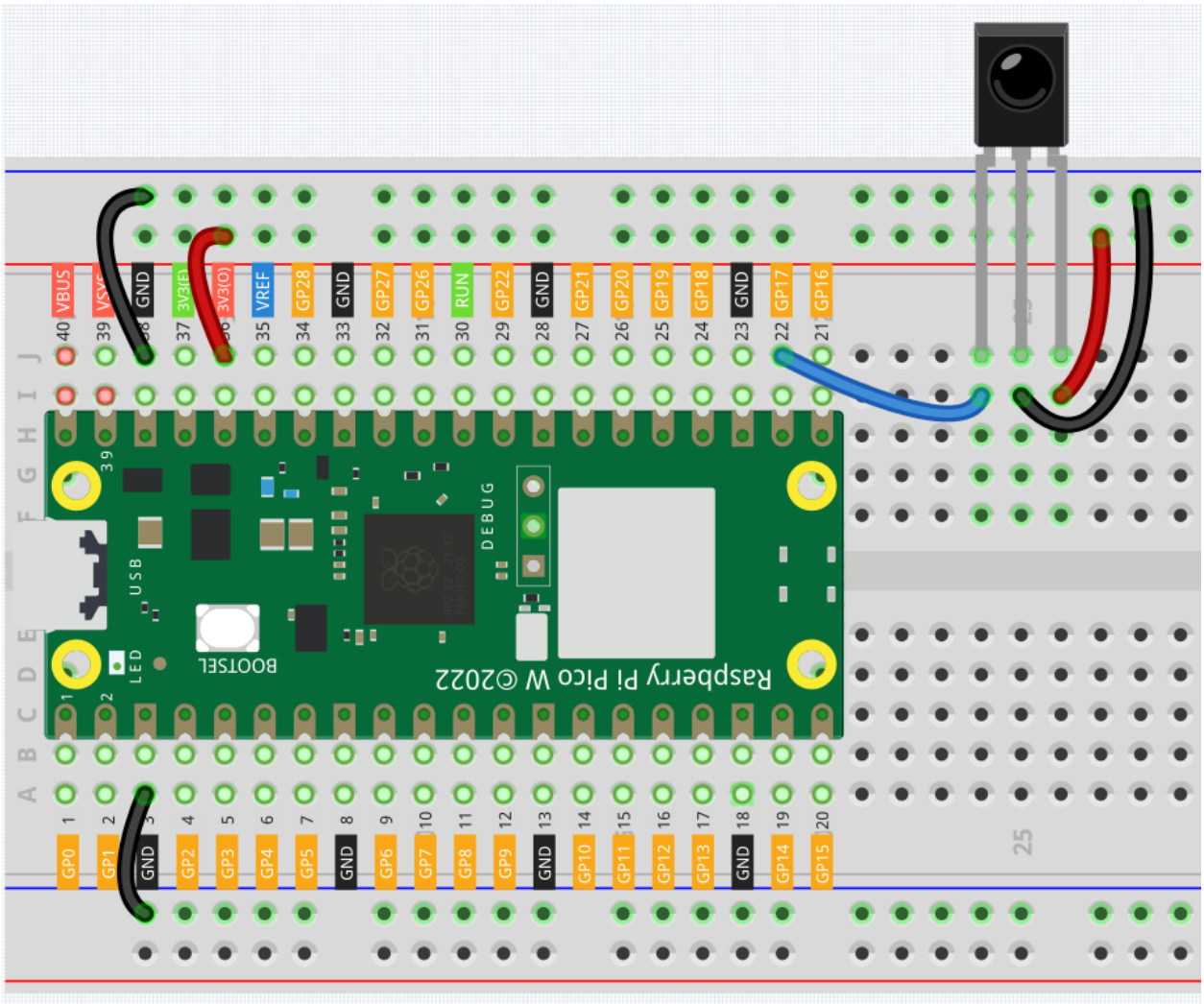
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Infrared Receiver*	1	

Schematic



Wiring



Code

Note:

- Open the `6.4_ir_remote_control.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the libraries in `ir_rx` folder, please check if it has been uploaded to Pico, for a detailed tutorial refer to *1.4 Upload the Libraries to Pico*.

```
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)
```

(continues on next page)

(continued from previous page)

```
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:
        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))
```

(continues on next page)

(continued from previous page)

```
ir = NEC_8(pin_ir, callback)  # Instantiate receiver
ir.error_function(print_error)  # Show debug information

try:
    while True:
        pass
except KeyboardInterrupt:
    ir.close()
```

The new remote control has a plastic piece at the end to isolate the battery inside. You need to pull out this plastic piece to power up the remote when you are using it. Once the program is running, when you press the remote control, the Shell will print out the key you pressed.

How it works?

This program looks slightly complicated, but it actually does the basic functions of the IR receiver with just a few lines.

```
import time
from machine import Pin, freq
from ir_rx.nec import NEC_8

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
```

Here an `ir` object is instantiated, which reads the signals acquired by the IR receiver at any time.

The result will be recorded in `data` of the callback function.

- Callback Function - Wikipedia

If the IR receiver receives duplicate values (e.g. pressing a key and not releasing it), then data < 0 and this data needs to be filtered.

Otherwise data would be a usable value, but some unspeakable code, and the `decodeKeyValue(data)` function is used to decode it.

```
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:
```

(continues on next page)

(continued from previous page)

```
        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:
        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"
```

If we press key **1**, the IR receiver outputs a value like `0x0C`, which needs to be decoded to correspond to the specific key.

Next are some debug functions. They are important, but not related to the effect we need to achieve, so we just put them in the program.

```
from ir_rx.print_error import print_error

ir.error_function(print_error) # Show debug information
```

Finally, we use an empty loop as the main program. And use try-except to make the program exit with the `ir` object terminated.

```
try:
    while True:
        pass
except KeyboardInterrupt:
    ir.close()
```

- Try Statement - Python Docs

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.41 6.5 Radio Frequency Identification

Radio Frequency Identification (RFID) is a technology that uses wireless communication between an object (or tag) and an interrogating device (or reader) to track and identify it. The tag's transmission range is limited to several meters. Readers and tags do not necessarily require a line of sight.

An integrated circuit (IC) and an antenna are usually present on most tags. As well as storing information, the microchip manages communication with the reader via radio frequency (RF). In passive tags, there is no independent energy source and they rely on an external electromagnetic signal from the reader for power. An active tag is powered by an independent energy source, such as a battery. As a result, they may be more powerful in terms of processing, transmission, and range.

- *MFRC522 Module*

Required Components

In this project, we need the following components.

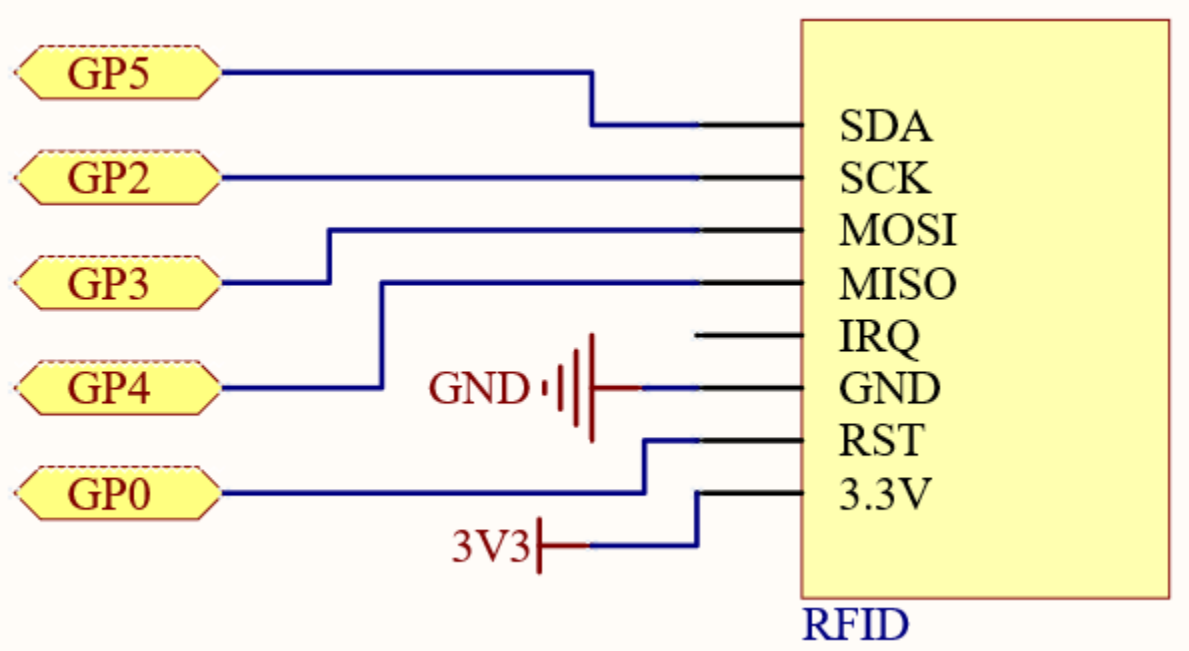
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

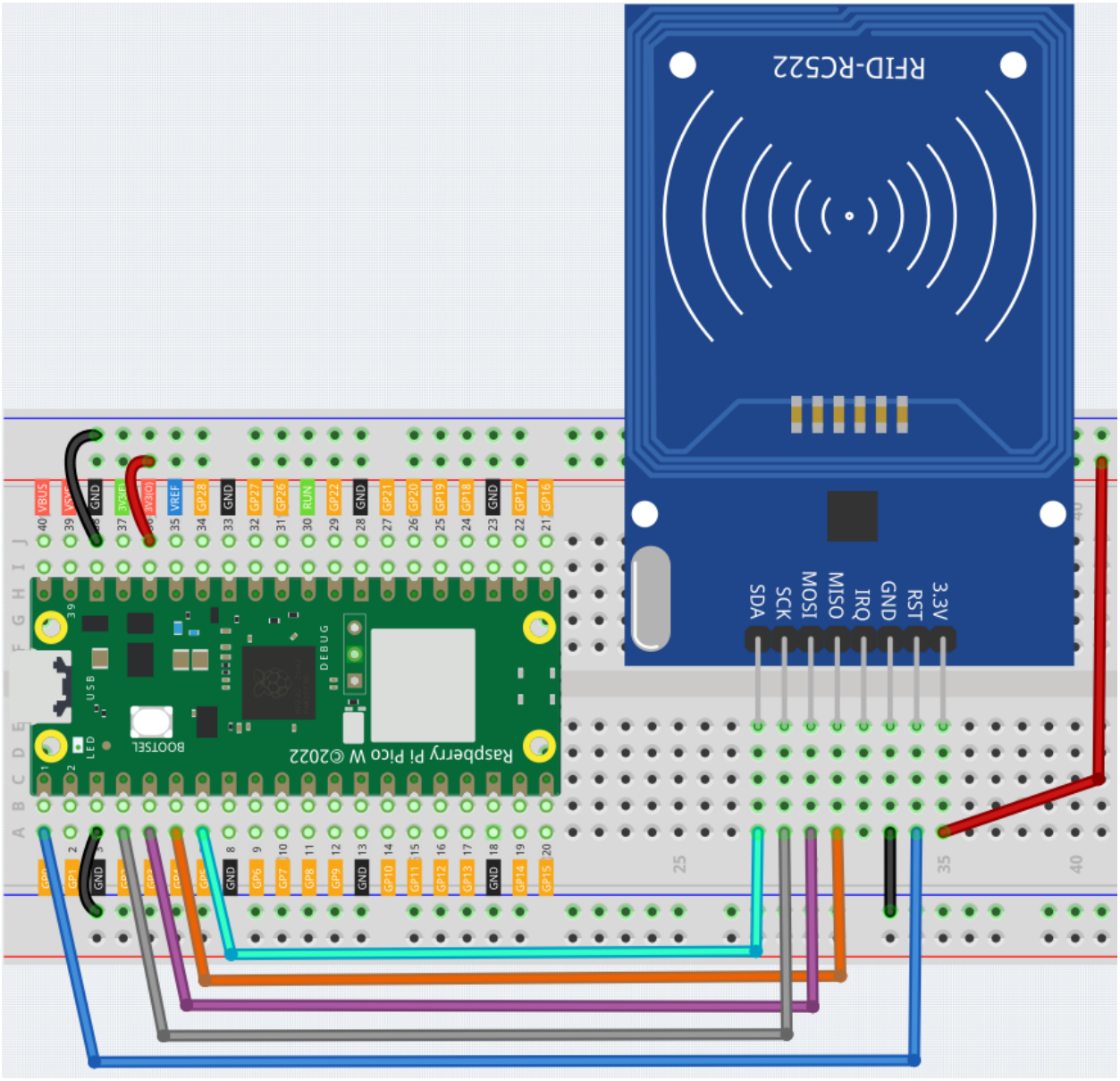
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*MFRC522 Module*	1	

Schematic



Wiring



Code

Here you need to use the libraries in `mfrc522` folder, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *1.4 Upload the Libraries to Pico*.

The main function is divided into two:

- `6.5_rfid_write.py`: Used to write information to the card (or key).
- `6.5_rfid_read.py`: used to read the information in the card (or key)

Open the `6.5_rfid_write.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.

After running you will be able to type message in the shell and then put the card (or key) close to the MFRC522 module to write the message in.

```
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()
```

Open the `6.5_rfid_read.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.

After running, you will be able to read the message stored in the card (or key).

```
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()
```

How it works?

```
from mfrc522 import SimpleMFRC522

reader = SimpleMFRC522(spi_id=0,sck=2,miso=4,mosi=3,cs=5,rst=0)
```

Instantiate `SimpleMFRC522()` class.

```
id, text = reader.read()
```

This function is used to read card data. If the reading is successful, id and text will be returned.

```
id, text = reader.write("text")
```

This function is used to write information to the card, press **Enter** key to finish writing. `text` is the information to be written to the card.

7. Funny Project

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.42 7.1 Light Theremin

Theremin is an electronic musical instrument that does not require physical contact. Based on the position of the player's hand, it produces different tones.

Its controlling section is usually made up of two metal antennas that sense the position of the thereminist's hands and control oscillators with one hand and volume with the other. The electric signals from the theremin are amplified and sent to a loudspeaker.

We cannot reproduce the same instrument through Pico W, but we can use photoresistor and passive buzzer to achieve similar gameplay.

- Theremin - Wikipedia

Required Components

In this project, we need the following components.

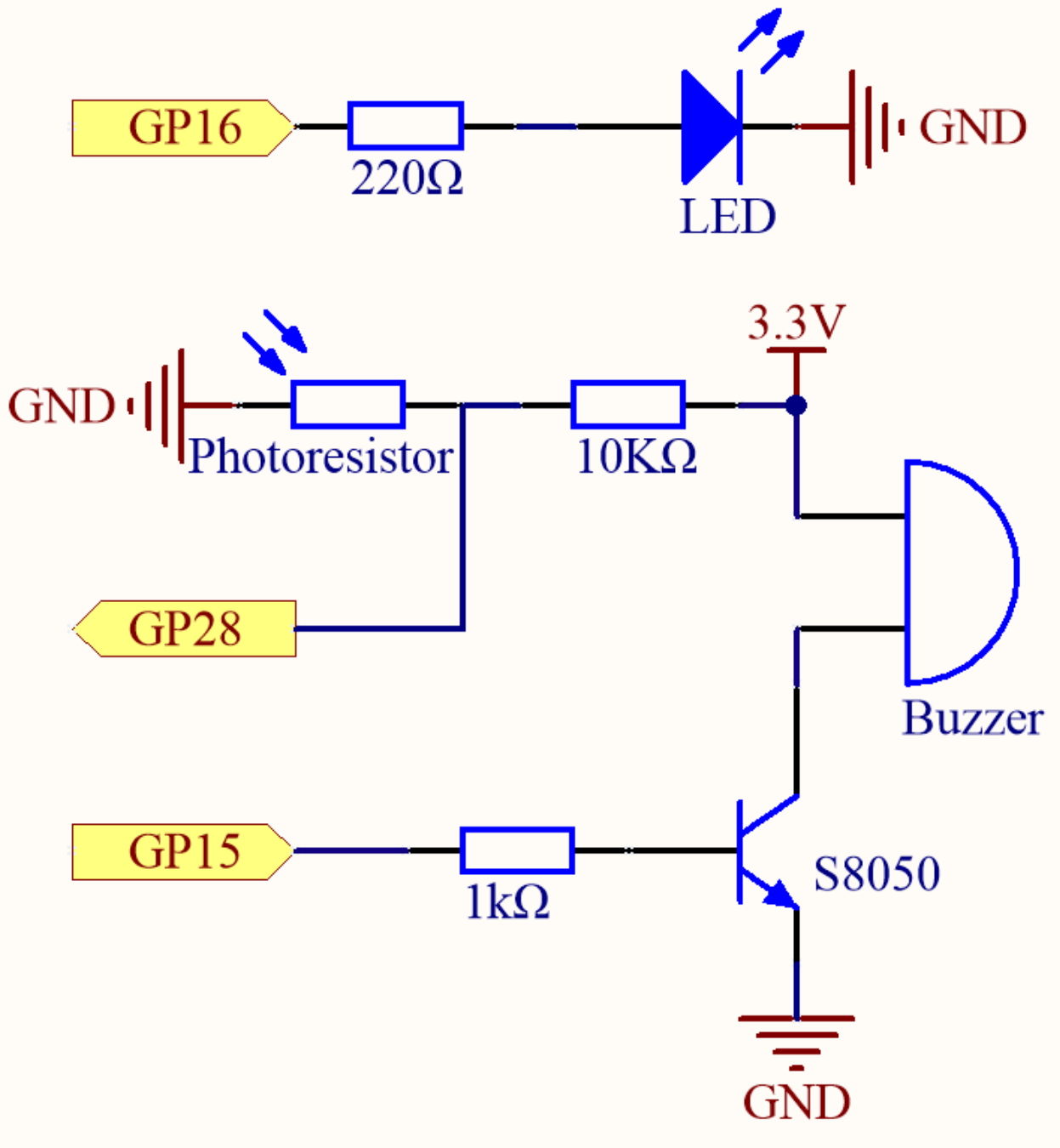
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*LED*	1	
6	*Transistor*	1(S8050)	
7	*Resistor*	3(1K, 220, 10K)	
8	Active *Buzzer*	1	
9	*Photoresistor*	1	

Schematic

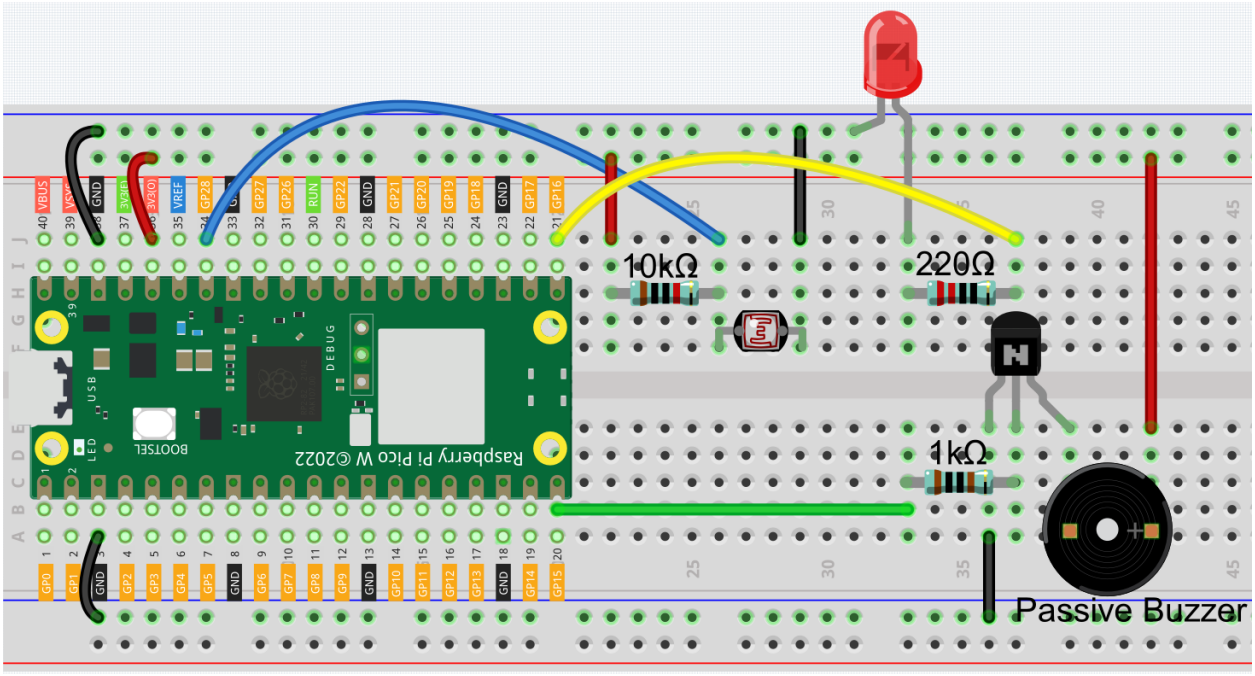


Before starting the project, wave your hand up and down over the photoresistor to calibrate the range of light intensity. The LED connected in GP16 is used to indicate the debugging time, and the LED is lit to indicate the start of debugging and off to indicate the end of debugging.

When GP15 outputs high level, S8050 (NPN transistor) conducts and the passive buzzer starts to sound.

When the light is stronger, GP28's value is smaller; vice versa, it is larger when the light is weaker. By programming the value of the photoresistor to affect the frequency of the passive buzzer, a photosensitive device can be simulated.

Wiring



Code

Note:

- Open the `7.1_light_theremin.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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

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)

# calibrate the photoresistor max & min values.
timer_init_start = utime.ticks_ms()
```

(continues on next page)

(continued from previous page)

```
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)

# play
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)
```

As soon as the program runs, the LED will light up, and we will have five seconds to calibrate the photoresistor's detection range.

This is due to the different light environments we may have when we use it (e.g., different light intensities at noon and dusk), as well as our hands' height above the photoresistor. You need to set the maximum and minimum height of your hand from the photoresistor, which is also the height at which you play the instrument.

After five seconds, the LED will turn off, at which point we can wave our hands over the photoresistor and play.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.43 7.2 Room Temperature Meter

Using a thermistor and an I2C LCD1602, we can create a room temperature meter.

This project is very simple, it is based on *2.13 Thermometer* with I2C LCD1602 to display the temperature.

Required Components

In this project, we need the following components.

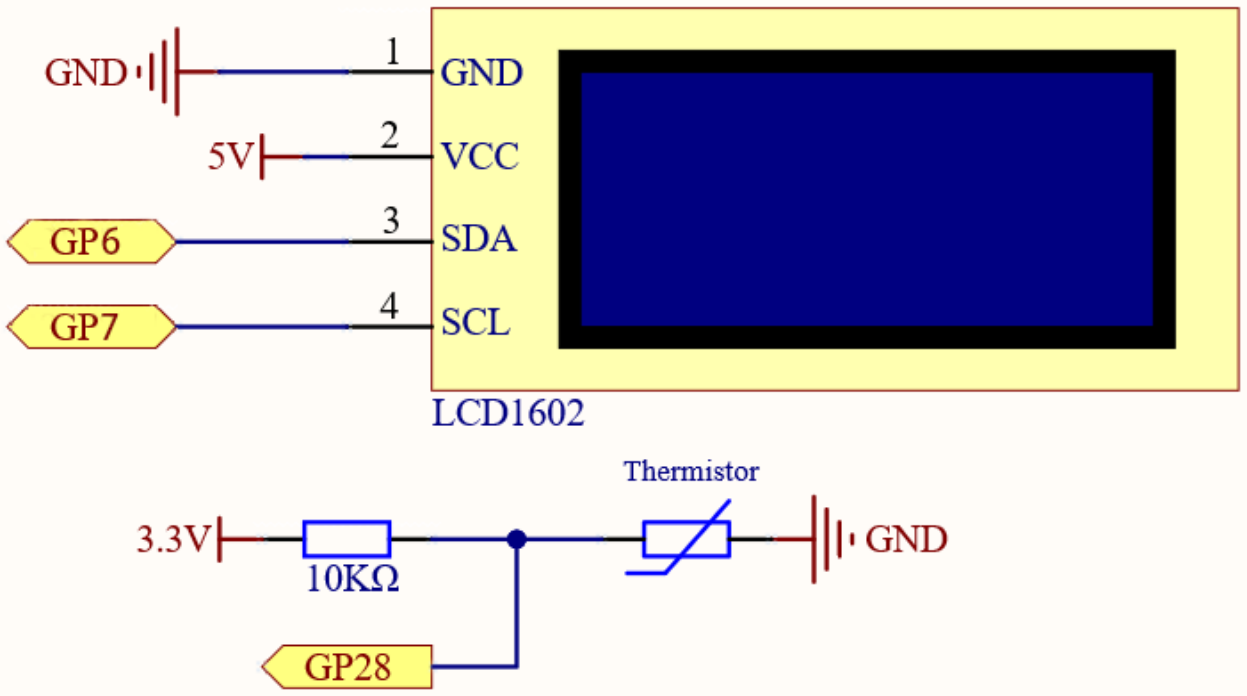
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

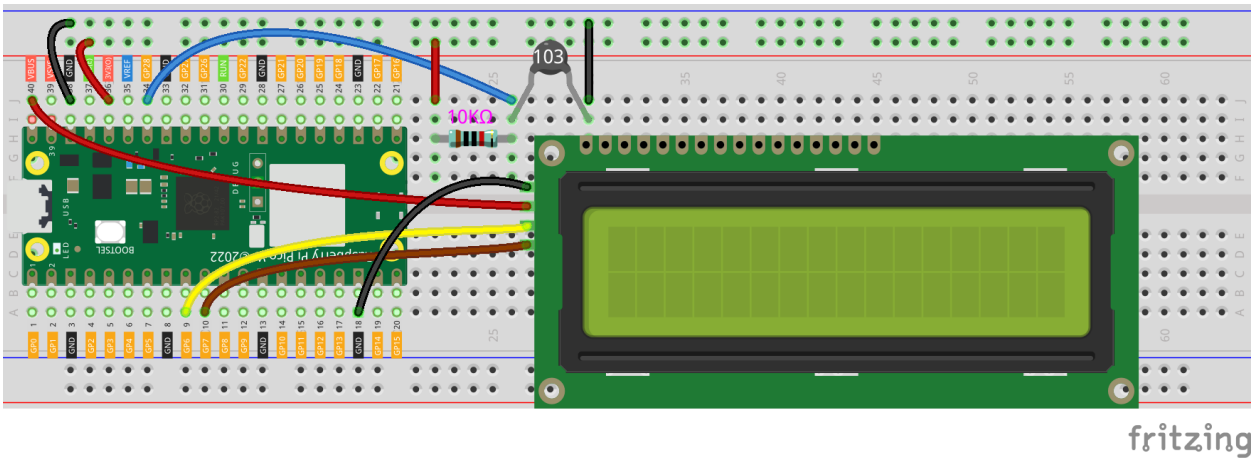
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Thermistor*	1	
7	*I2C LCD1602*	1	

Schematic



Wiring



Code

Note:

- Open the `7.2_room_temperature_meter.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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()
```

The LCD will display the temperature value in the current environment after the program runs.

Note: If the code and wiring are fine, but the LCD still does not display content, you can turn the potentiometer on the back to increase the contrast.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

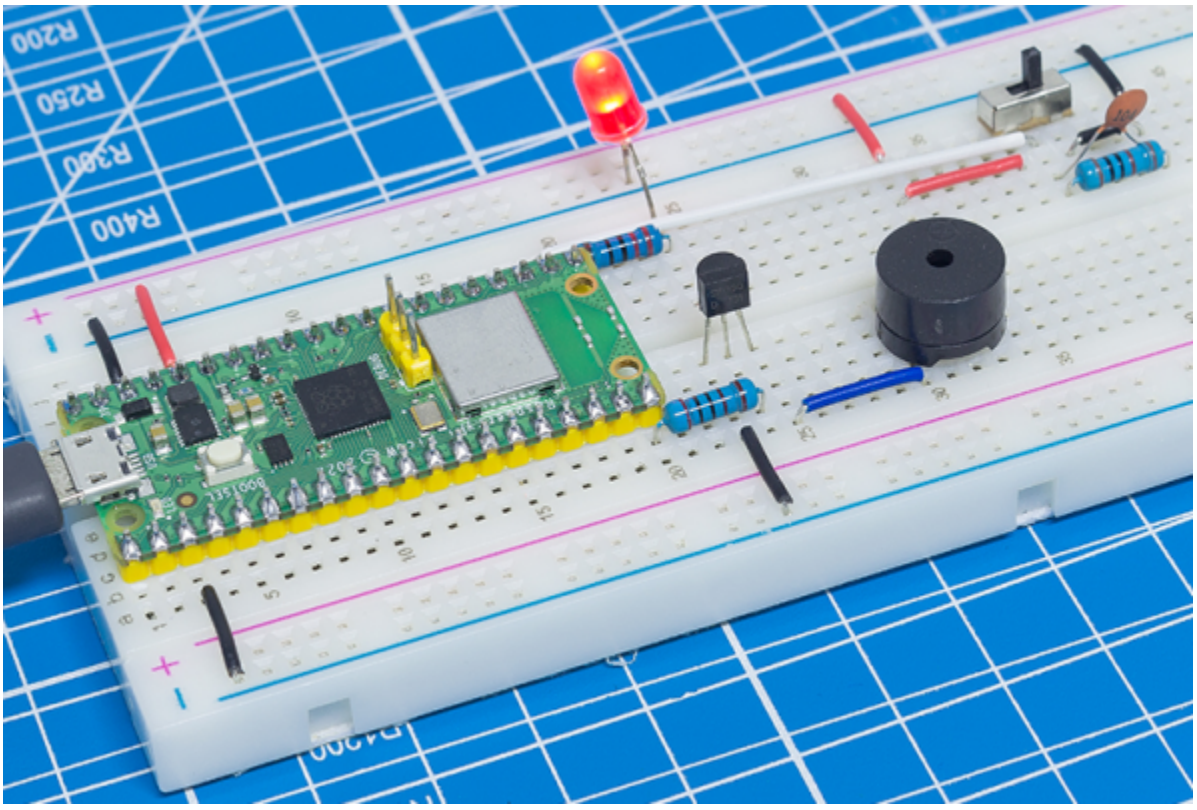
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.44 7.3 Alarm Siren Lamp

Police lights are often visible in real life (or in movies). Usually, it is used to maintain traffic, serve as a warning device, and serve as an important safety prop for officers, emergency vehicles, fire trucks, and engineering vehicles. When you see its lights or hear its sound, you must be careful, which means you (or those around you) may be in danger.

An LED and buzzer are used here to create a small warning light, which is activated by a slide switch.



Required Components

In this project, we need the following components.

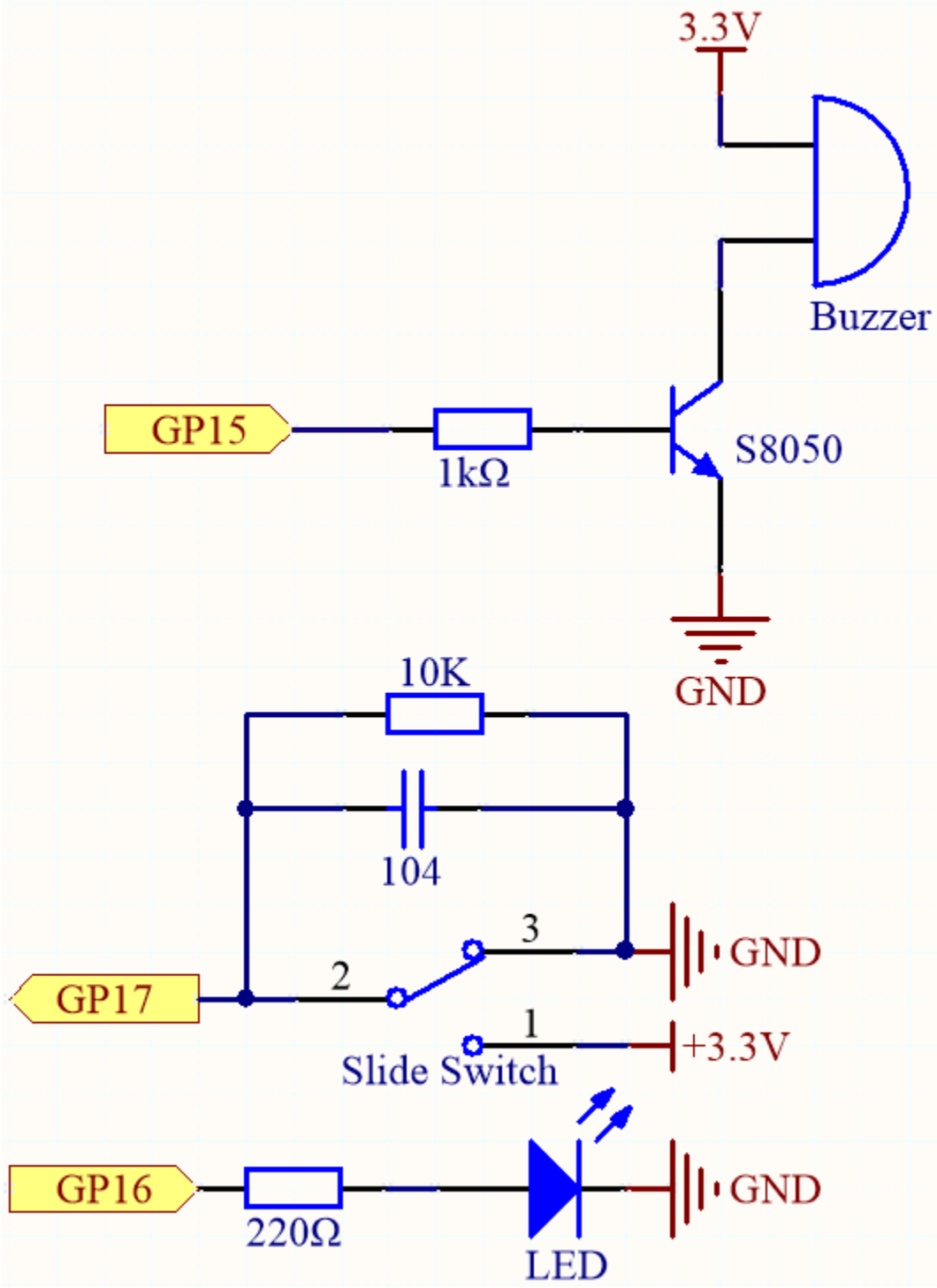
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

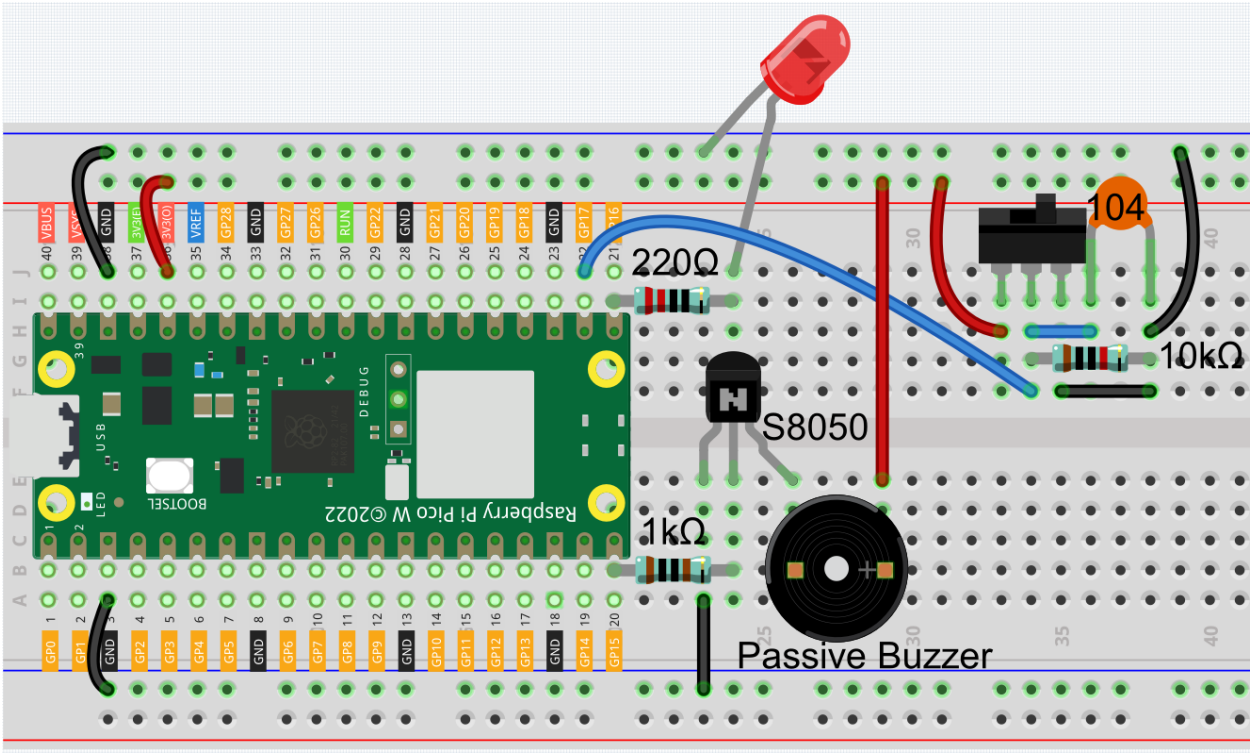
SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*LED*	1	
6	*Transistor*	1(S8050)	
7	*Resistor*	3(1K, 220, 10K)	
8	Passive *Buzzer*	1	
9	*Capacitor*	1(104)	
10	*Slide Switch*	1	

Schematic



- GP17 is connected to the middle pin of the slider, along with a 10K resistor and a capacitor (filter) in parallel to GND, which allows the slider to output a steady high or low level when toggled to the left or right.
- As soon as GP15 is high, the NPN transistor conducts, causing the passive buzzer to start sounding. This passive buzzer is programmed to gradually increase in frequency to produce a siren sound.
- An LED is connected to GP16 and is programmed to periodically change its brightness in order to simulate a siren.

Wiring



Code

Note:

- Open the `7.3_alarm_siren_lamp.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
import machine
import time


buzzer = machine.PWM(machine.Pin(15))
led = machine.PWM(machine.Pin(16))
led.freq(1000)

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)
```

(continues on next page)

(continued from previous page)

```
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)
```

Once the program is running, toggle the slide switch to the left (yours may be to the right, depending on how your slide switch is wired) and the buzzer will emit a progressive warning tone and the LED will change its brightness accordingly; toggle the slide switch to the right and the buzzer and LED will stop working.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.45 7.4 Passenger Counter

For large shopping malls, shopping centers, chain stores, airports, stations, museums, and public places such as exhibition halls, passenger traffic is an indispensable data.

In airports and stations, for example, the number of people needs to be strictly controlled to ensure safety and smooth flow. It is also possible to know when there are more visitors in shopping centers and chain stores, how many orders each user can generate, etc. As a result, we can analyze people's consumption habits and increase turnover.

Passenger counters can help people understand the operation of these public places and organize their operations efficiently.

A simple passenger counter is created using a PIR sensor and a 4-digit 7-segment display.

Required Components

In this project, we need the following components.

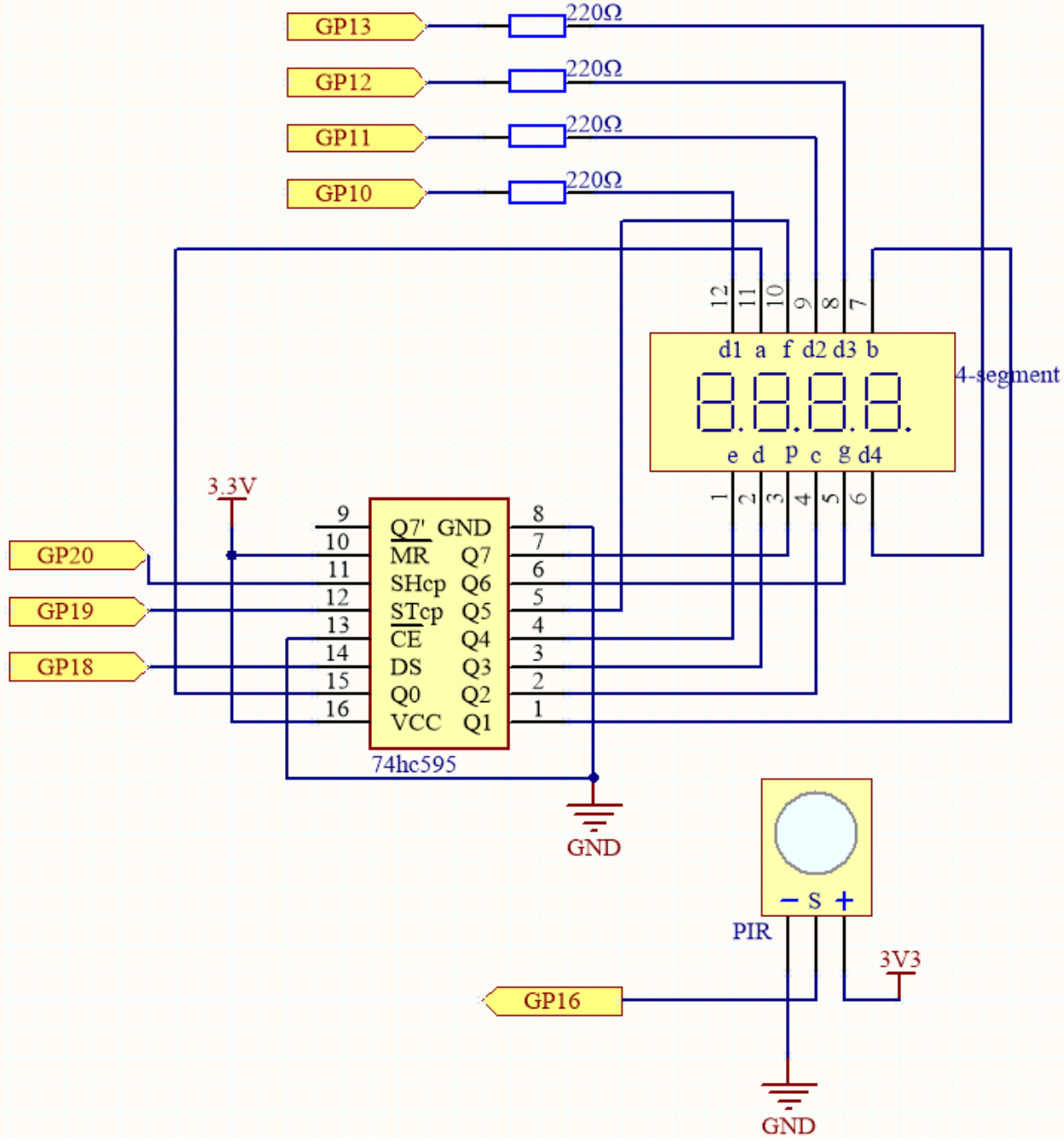
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUANTITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	4(220)	
6	*4-Digit 7-Segment Display*	1	
7	*74HC595*	1	
8	*PIR Motion Sensor Module*	1	

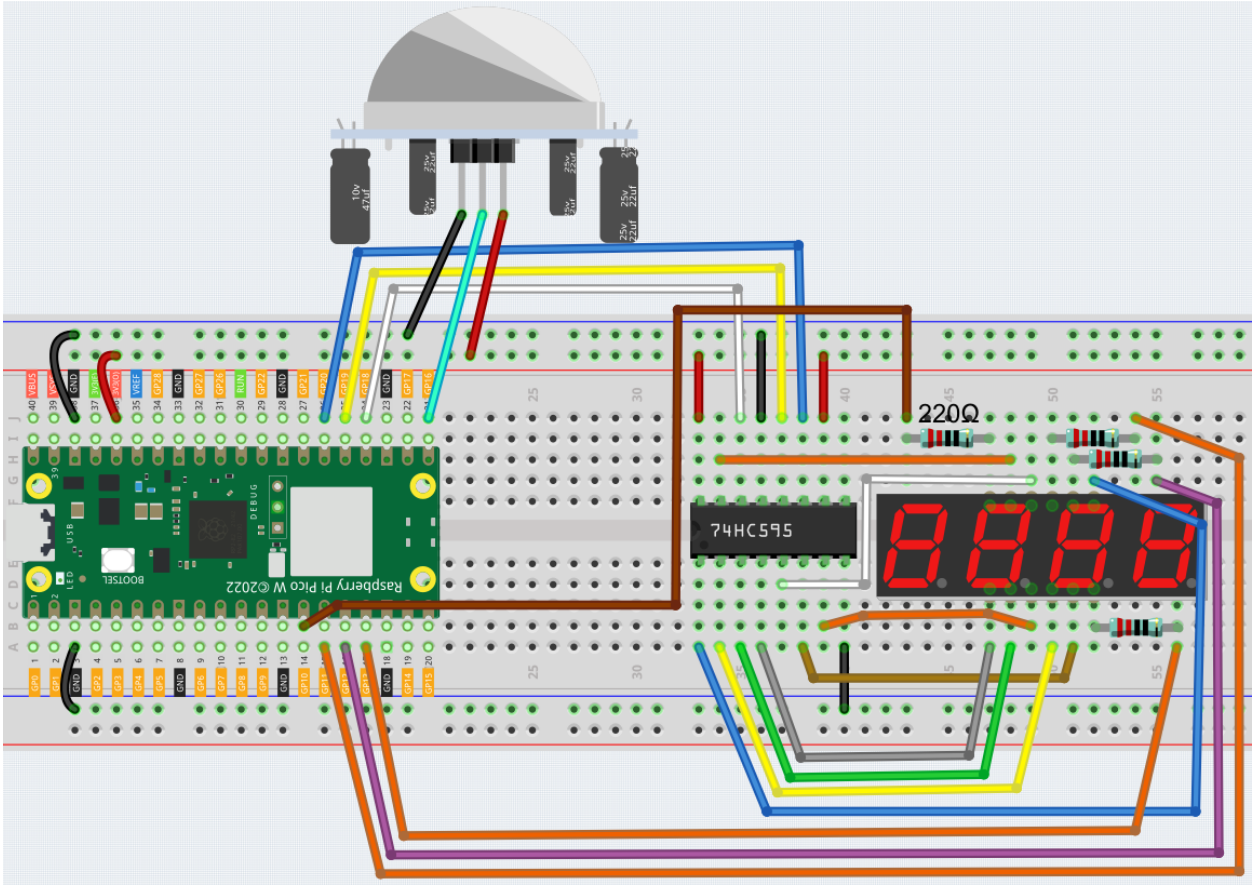
Schematic



- This circuit is based on the *5.3 Time Counter* with the addition of a PIR module.
- The PIR will send a high signal of about 2.8s long when someone passes by.
- The PIR module has two potentiometers: one adjusts sensitivity, the other adjusts detection distance. To make the PIR module work better, you need to turn both of them counterclockwise to the end.



Wiring



Code

Note:

- Open the `7.4_passenger_counter.py` file under the path of `kepler-kit-main/micropython` or copy this

code into Thonny, then click "Run Current Script" or simply press F5 to run it.

- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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()
    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
```

(continues on next page)

(continued from previous page)

```
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])
```

When the code is run, the number on the 4-digit 7-segment display will be added by one if someone passes in front of the PIR module.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.46 7.5 GAME - 10 Second

To challenge your concentration, follow me next to make a game device. Make a magic wand by connecting the tilt switch with a stick. When you shake the wand, the 4-digit segment display will start counting, and when you shake it again, it will stop counting. In order to win, you must keep the displayed count at **10.00**. You can play the game with your friends to see who is the time wizard.

Required Components

In this project, we need the following components.

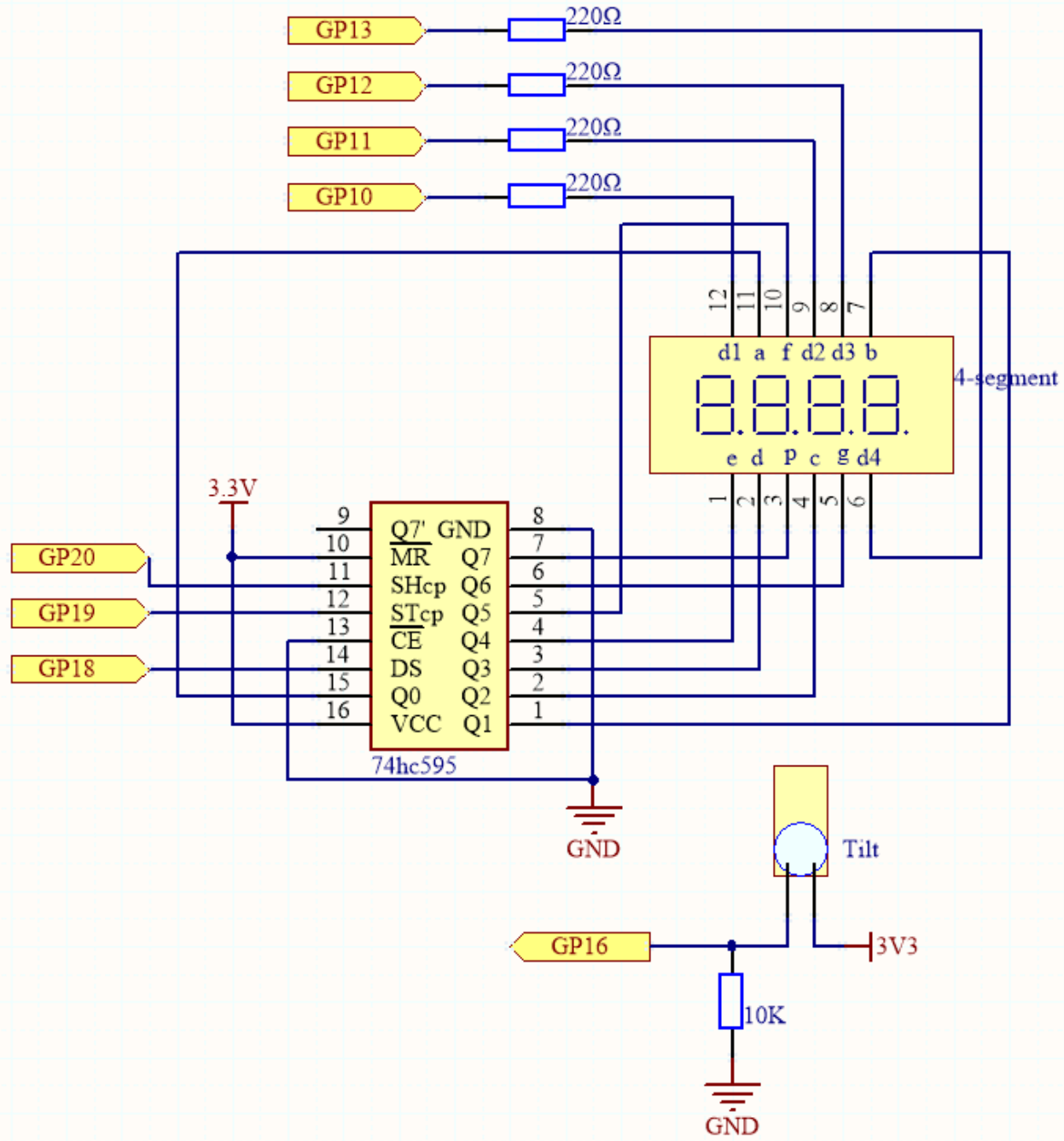
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

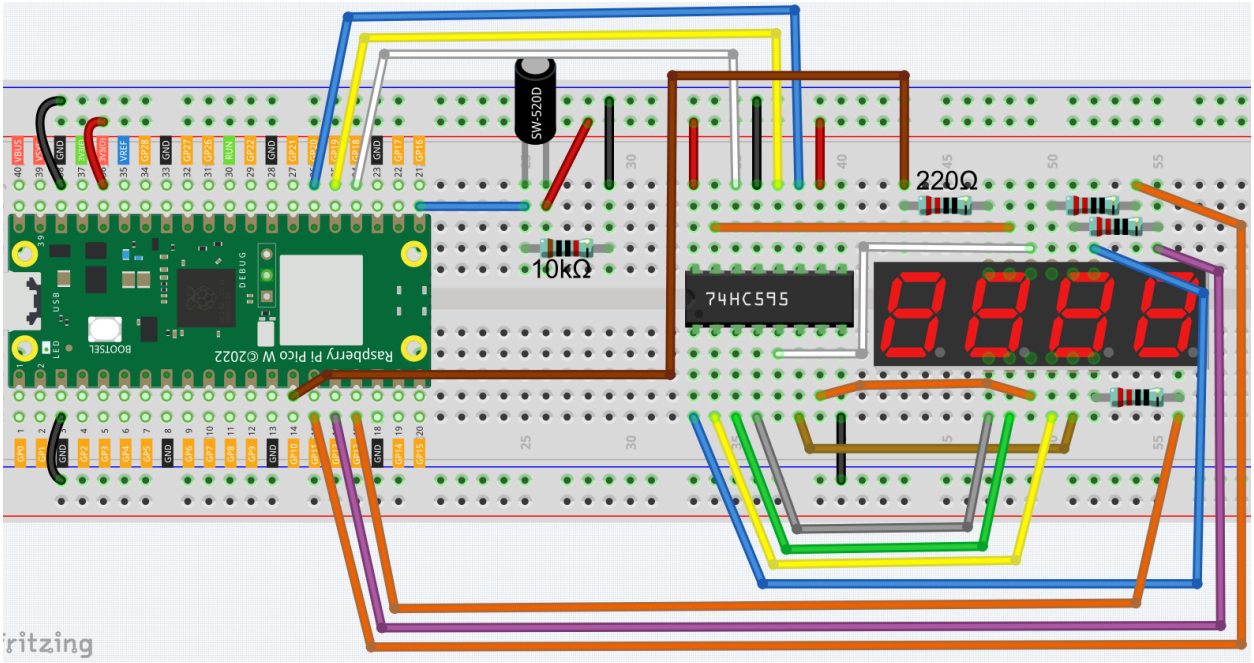
SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	5(4-220, 1-10K)	
6	*4-Digit 7-Segment Display*	1	
7	*74HC595*	1	
8	*Tilt Switch*	1	

Schematic



- This circuit is based on *5.3 Time Counter* with the addition of a tilt switch.
- GP16 is high when the tilt switch is upright; low when tilted.

Wiring



Code

Note:

- Open the `7.5_game_10_second.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)

def pickDigit(digit):
    for i in range(4):
        placePin[i].value(1)
    placePin[digit].value(0)

def clearDisplay():
```

(continues on next page)

(continued from previous page)

```
    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])


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)
```

The 4-digit 7-segment display will begin counting when you shake the wand, and will stop counting when you shake it

again. You win if you manage to keep the displayed count at 10.00. The game will continue after one more shake.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.47 7.6 Traffic Light

Traffic Light is a signal device located at roadway intersections, crosswalks and other locations to control the flow of traffic.

Traffic signals are standardized by the Vienna Convention on Road Signs and Signals. Provides users with the right-of-way by alternating LEDs in three standard colors.

- **Red light**: Traffic should stop if it sees a flashing red light, equivalent to a stop sign.
- **Yellow light**: A warning signal is about to turn red. Yellow lights are interpreted differently in different countries (regions).
- **Green light**: Allows traffic to move in the indicated direction.

In this project, we will use three colors of LEDs to implement traffic light changes and a 4-digit 7-segment display to show the time of each traffic state.

Required Components

In this project, we need the following components.

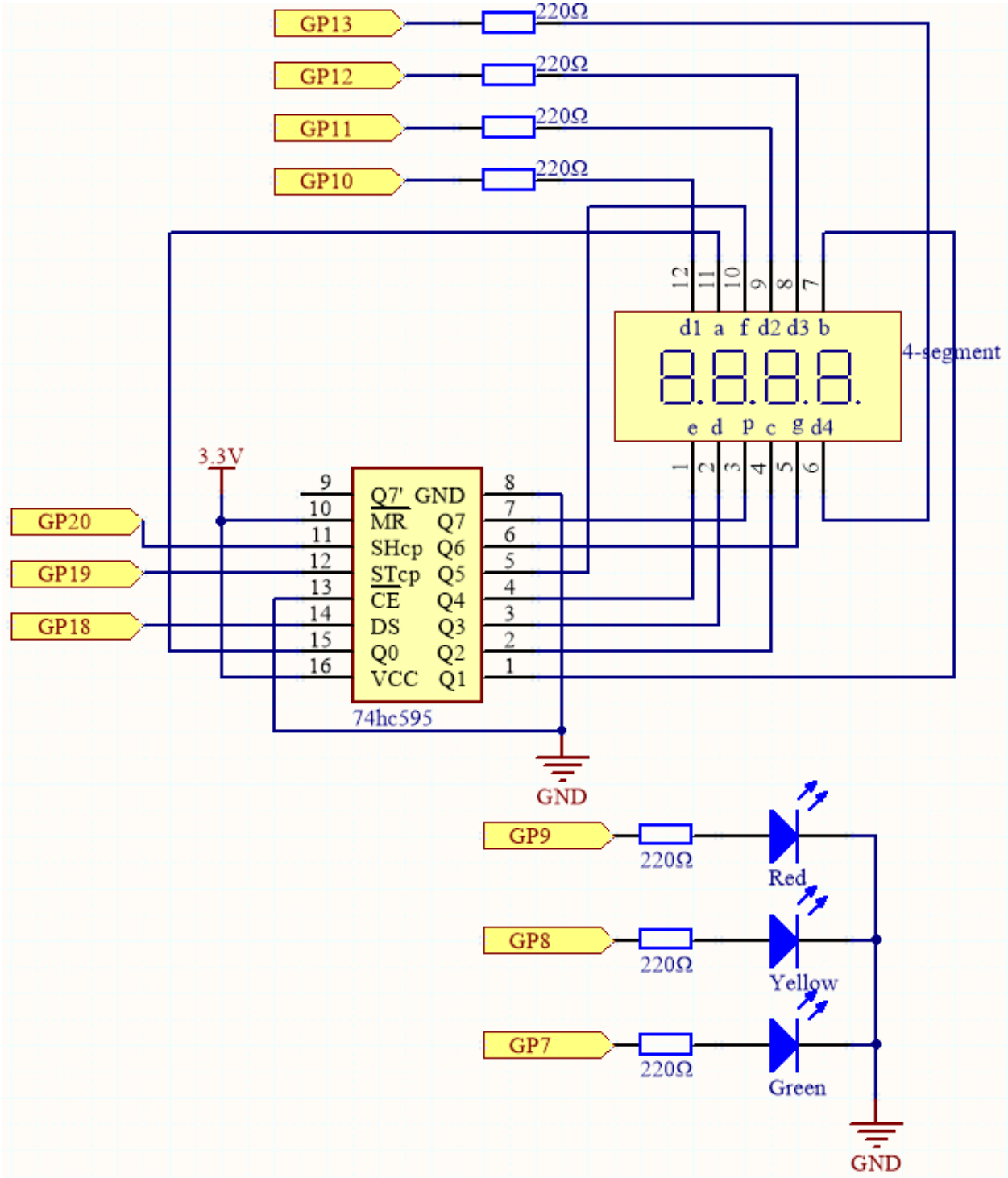
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

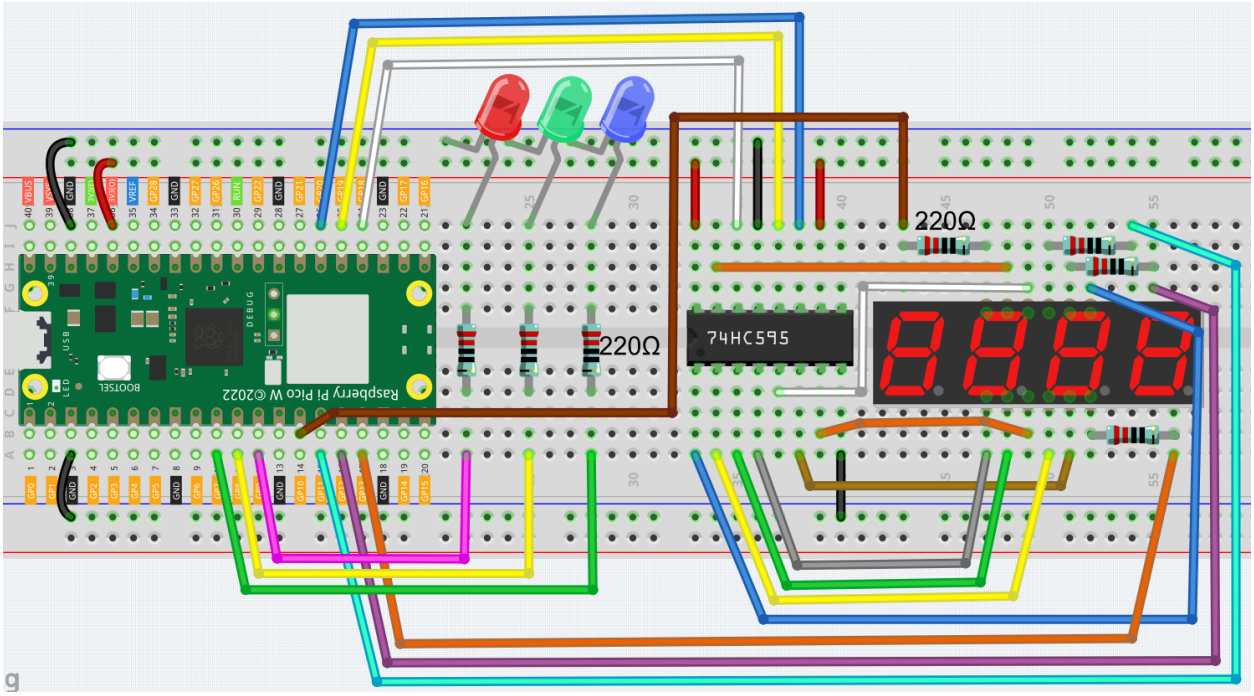
SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	7(220)	
6	*4-Digit 7-Segment Display*	1	
7	*74HC595*	1	
8	*LED*	1	

Schematic



- This circuit is based on the *5.3 Time Counter* with the addition of 3 LEDs.
- The 3 red, yellow and green LEDs are connected to GP7~GP9 respectively.

Wiring



Code

Note:

- Open the `7.6_traffic_light.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
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)
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)
```

(continues on next page)

(continued from previous page)

```
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)
    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
```

(continues on next page)

(continued from previous page)

```
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)
```

When the code runs, the green LED stays on for 30 seconds, the yellow LED stays on for 5 seconds, and the green LED stays on for 30 seconds.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

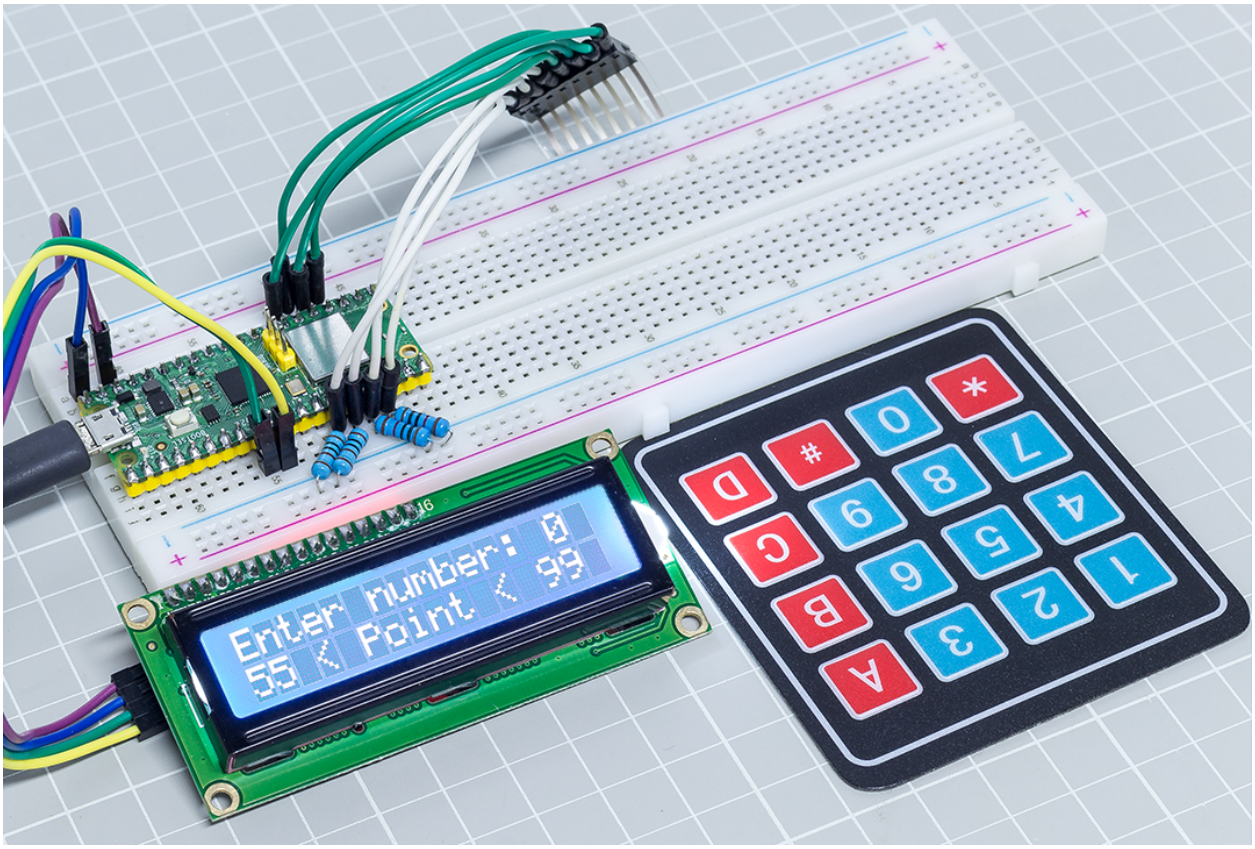
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.48 7.7 Guess Number

Guessing Numbers is a fun party game where you and your friends input numbers (0-99). With each input of the number, the range will shrink until a player answers the riddle correctly. Then the player is defeated and punished.

As an example, if the lucky number is 51, which the players cannot see, and the player 1 inputs 50, the prompt changes to 50 - 99; if the player 2 inputs 70, the range changes to 50 - 70; if the player 3 inputs 51, the player is unlucky. In this case, numbers are inputted through the keypad, and outcomes are displayed on a LCD screen.



Required Components

In this project, we need the following components.

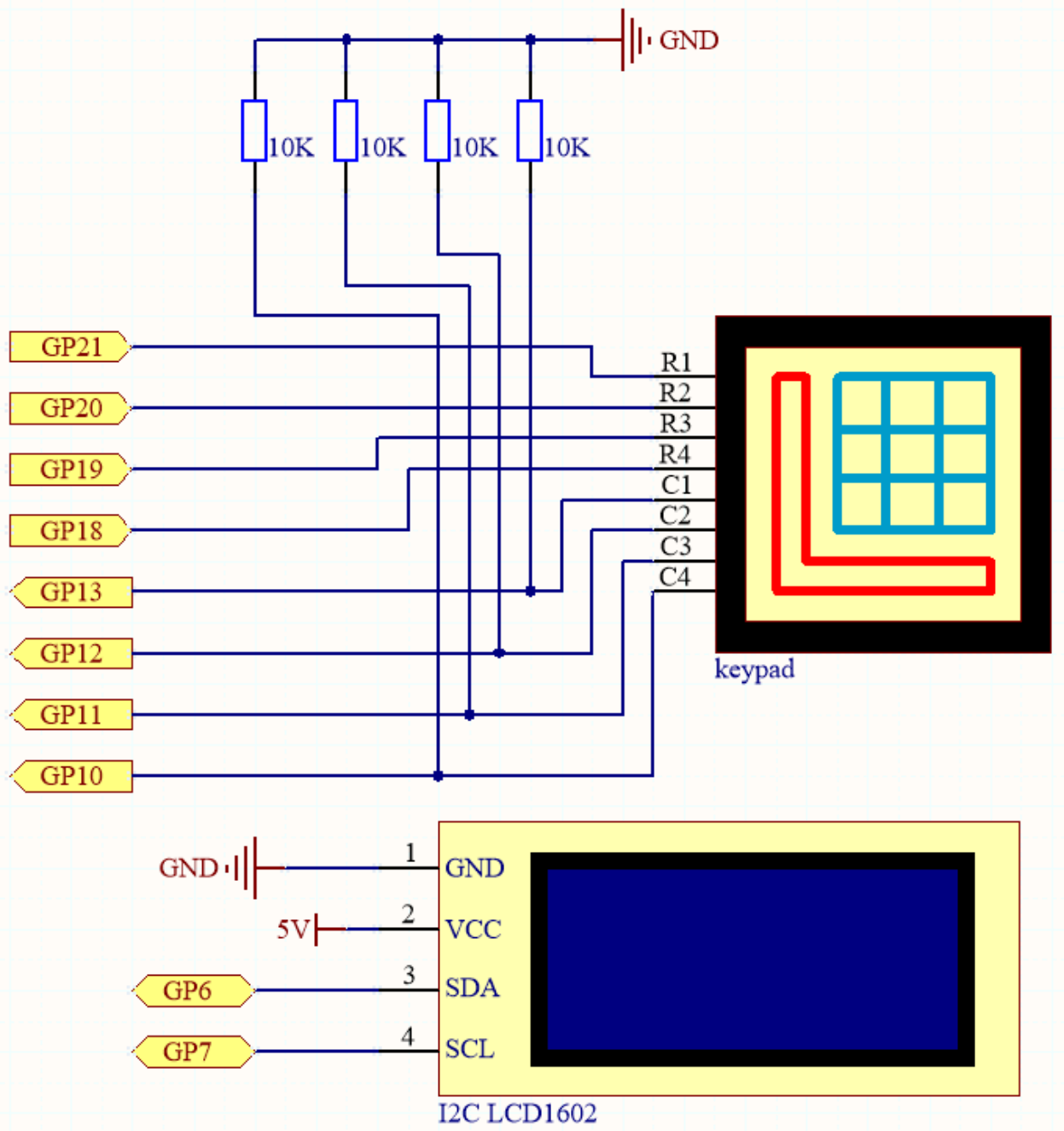
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

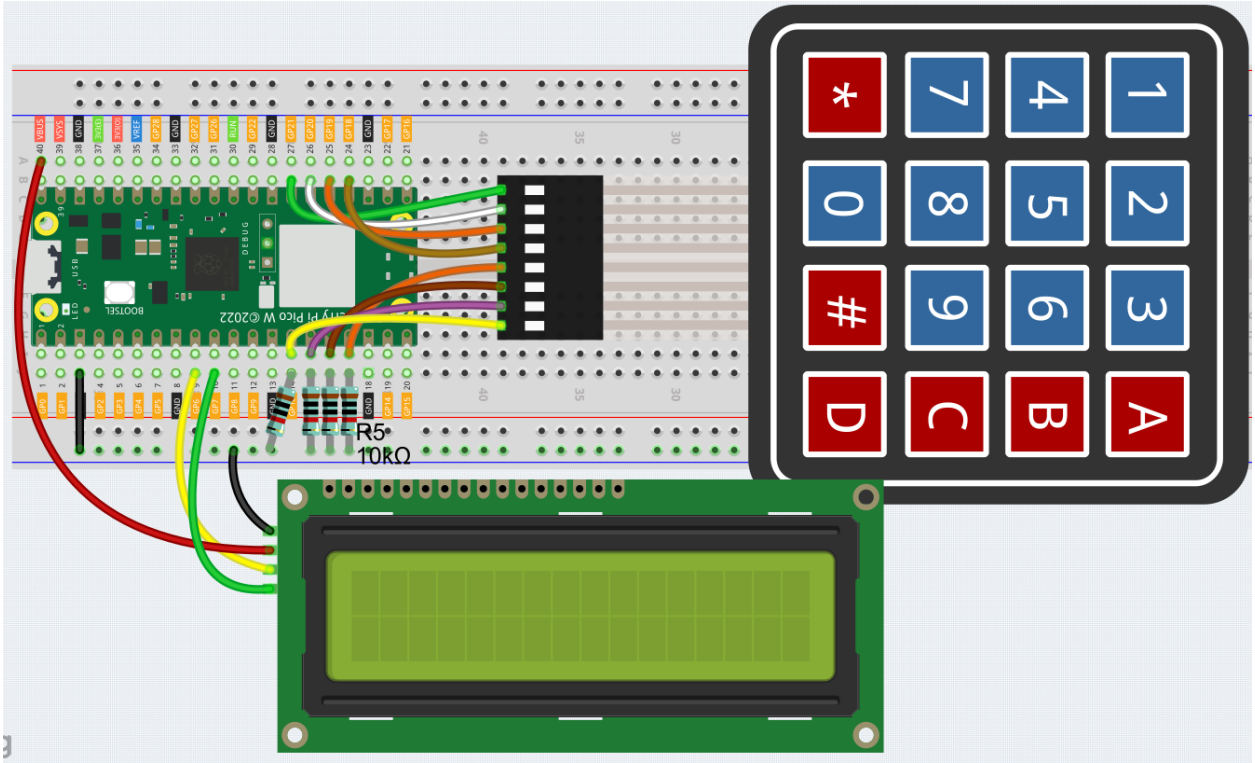
SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	4(10K)	
6	*4x4 Keypad*	1	
7	*I2C LCD1602*	1	

Schematic



This circuit is based on *4.2 4x4 Keypad* with the addition of an I2C LCD1602 to display the pressed keys.

Wiring



To make the wiring easier, in the above diagram, the column row of the matrix keyboard and the 10K resistors are inserted into the holes where G10 ~ G13 are located at the same time.

Code

Note:

- Open the `7.7_game_guess_number.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
from lcd1602 import LCD
import machine
import time
import urandom


# keypad function
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)
```

(continues on next page)

(continued from previous page)

```
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


# 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:
```

(continues on next page)

(continued from previous page)

```
            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
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)
```

- After the code runs, press `A` to start the game. A random number `point` is produced but not displayed on the LCD, and what you need to do is to guess it.
- The number you have typed appears at the end of the first line till the final calculation is finished. (Press `D` to start the comparation.)
- The number range of `point` is displayed on the second line. And you must type the number within the range.
- When you type a number, the range narrows; if you got the lucky number luckily or unluckily, there will appear `GAME OVER!`.

Note: If the code and wiring are fine, but the LCD still does not display content, you can turn the potentiometer on the back to increase the contrast.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

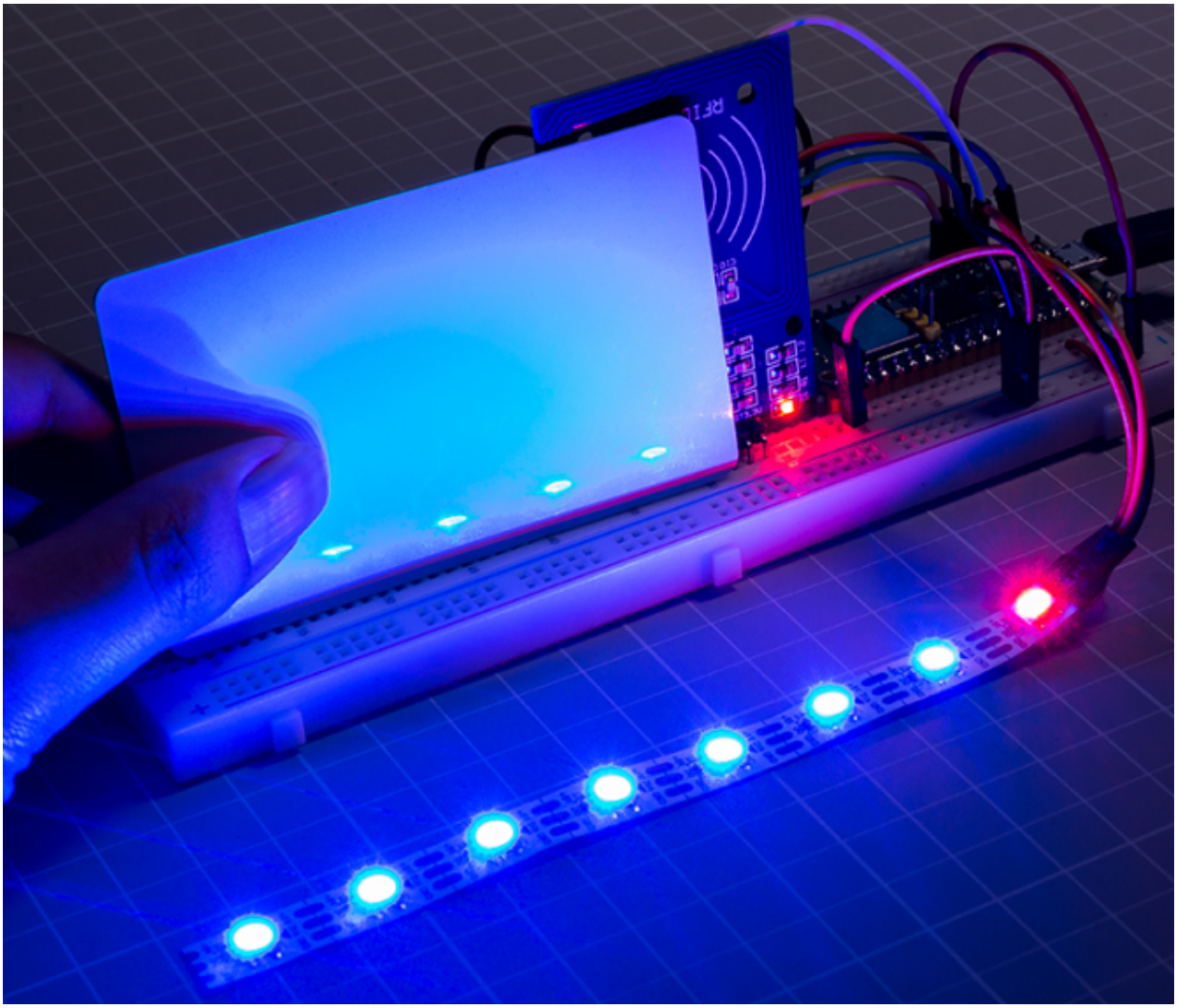
Ready to explore and create with us? Click [] and join today!

4.49 7.8 RFID Music Player

Through our previous project, *6.5 Radio Frequency Identification*, we learned that the MFRC522 module allows us to write up to 48 letters of information to the card (or key), including both the key and identity information, as well as the music score.

As an example, if you write `EEFGGFEDCCDEEDD EEFGGFEDCCDEDCC`, the buzzer will play the music when the card (or key) is read again. It can also be equipped with an WS2812 to display amazing effects.

You can find more sheet music on the Internet, or even write your own music, put them into the card (or key), and share them with your friends!



Required Components

In this project, we need the following components.

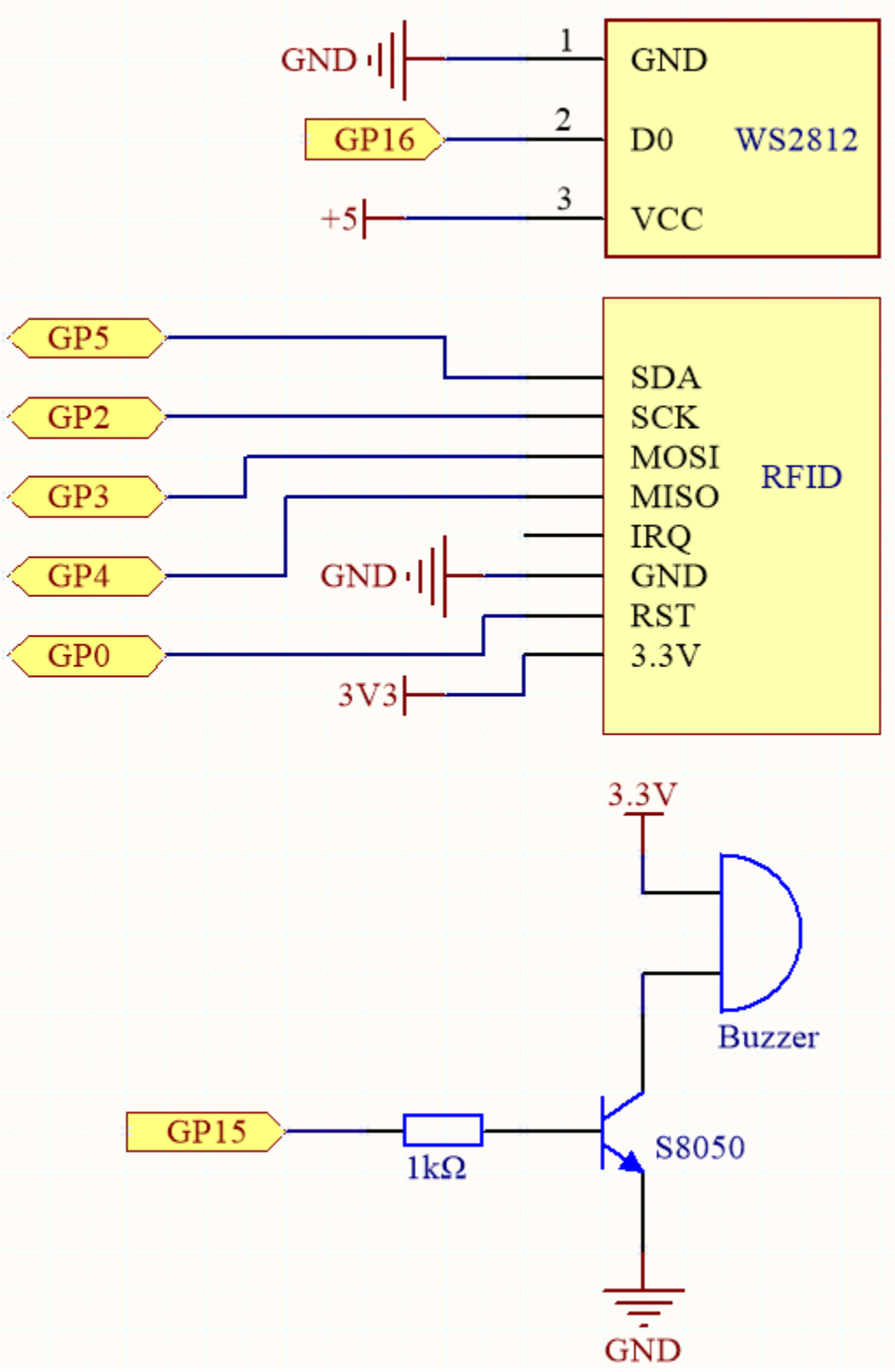
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

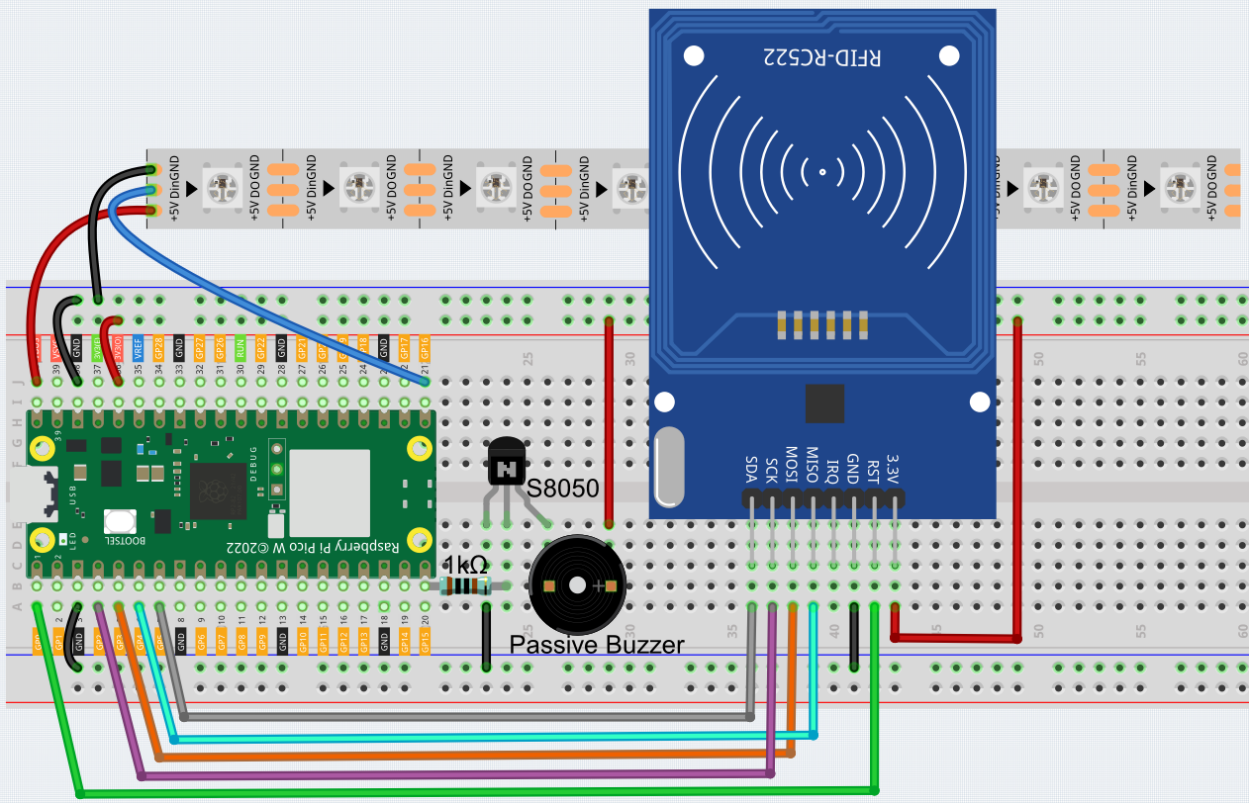
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Transistor*	1(S8050)	
6	*Resistor*	1(1K)	
7	Passive *Buzzer*	1	
8	*MFRC522 Module*	1	
9	*WS2812 RGB 8 LEDs Strip*	1	

Schematic


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

Wiring



Code

1. Open the `6.5_rfid_write.py` file under the path of `kepler-kit-main/micropython`, then click "Run Current Script" or simply press F5 to run it.
2. After running, type `EEFGGFEDCCDEEDD EEFGGFEDCCDEDCC` in the Shell and then put the card (or key) close to the MFRC522 module, this way an Ode an Joy score is stored in.
3. Open the `7.8_rfid_music_player.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.

```
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)

# buzzer
NOTE_C4 = 262
NOTE_D4 = 294
NOTE_E4 = 330
NOTE_F4 = 349
NOTE_G4 = 392
```

(continues on next page)

(continued from previous page)

```
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)


# lightup
def lumi(index):
    for i in range(8):
        ws[i] = 0x0000FF
    ws[index] = 0xFF0000 # int(urandom.uniform(0, 0xFFFFFF))
    ws.write()


# encode text to index
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]
        string=new_str

# read card
def read():
    print("Reading...Please place the card...")
    id, text = reader.read()
    print("ID: %s\nText: %s" % (id,text))
    take_text(text)

read()
```

4. By putting the card (or key) close to the MFRC522 module again, the buzzer will play the music stored on the card (or key), and the RGB strip will light up in a random color.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

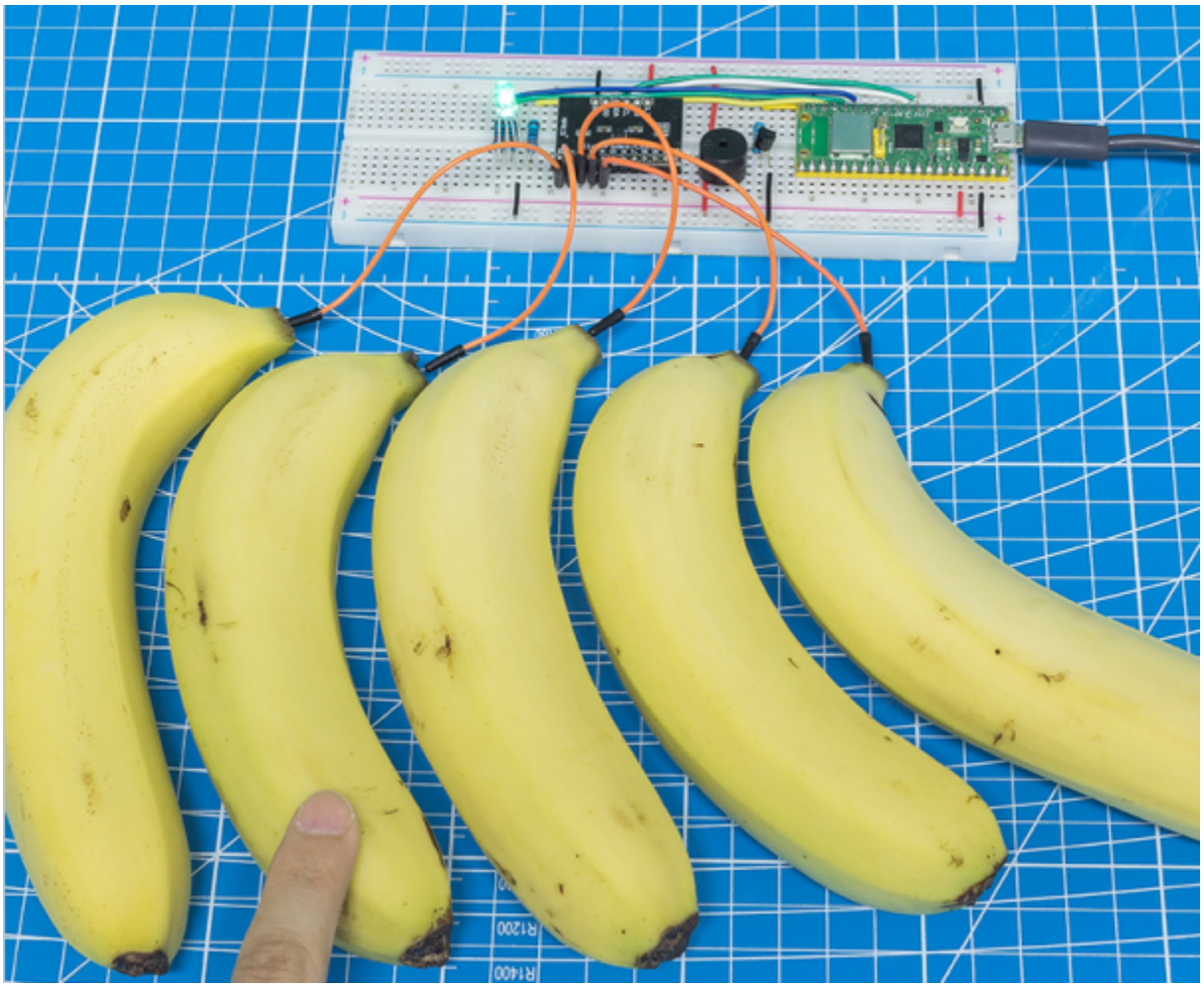
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.50 7.9 Fruit Piano

Electrical conductivity is found in many metal objects, as well as in the human body and fruits. This property can be used to create a fun little project: a fruit piano. In other words, we turn fruits into keyboards that can play music just by touching them.



Required Components

In this project, we need the following components.

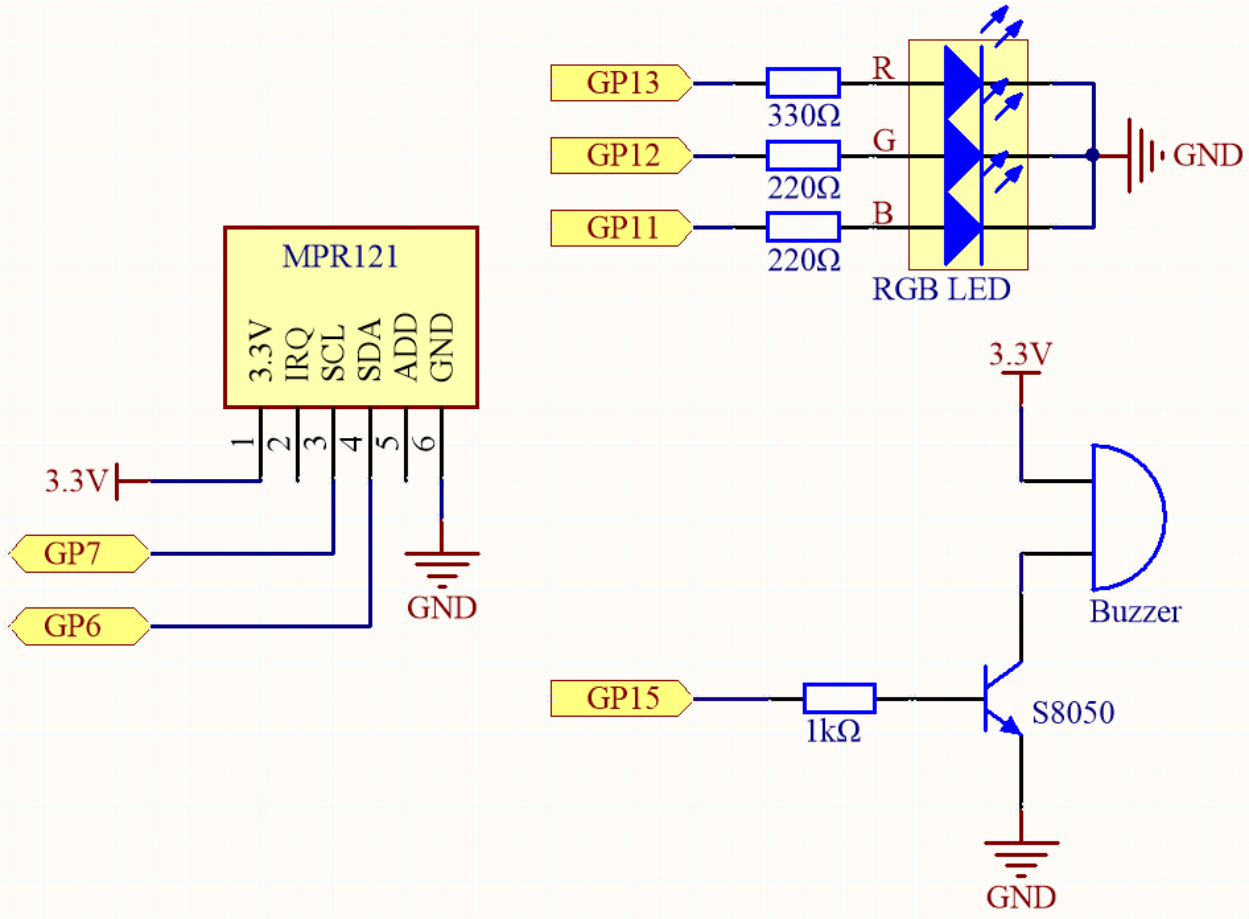
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Transistor*	1(S8050)	
6	*Resistor*	4(1-1K, 1-330, 2-220)	
7	Passive *Buzzer*	1	
8	*RGB LED*	1	
9	*MPR121 Module*	1	

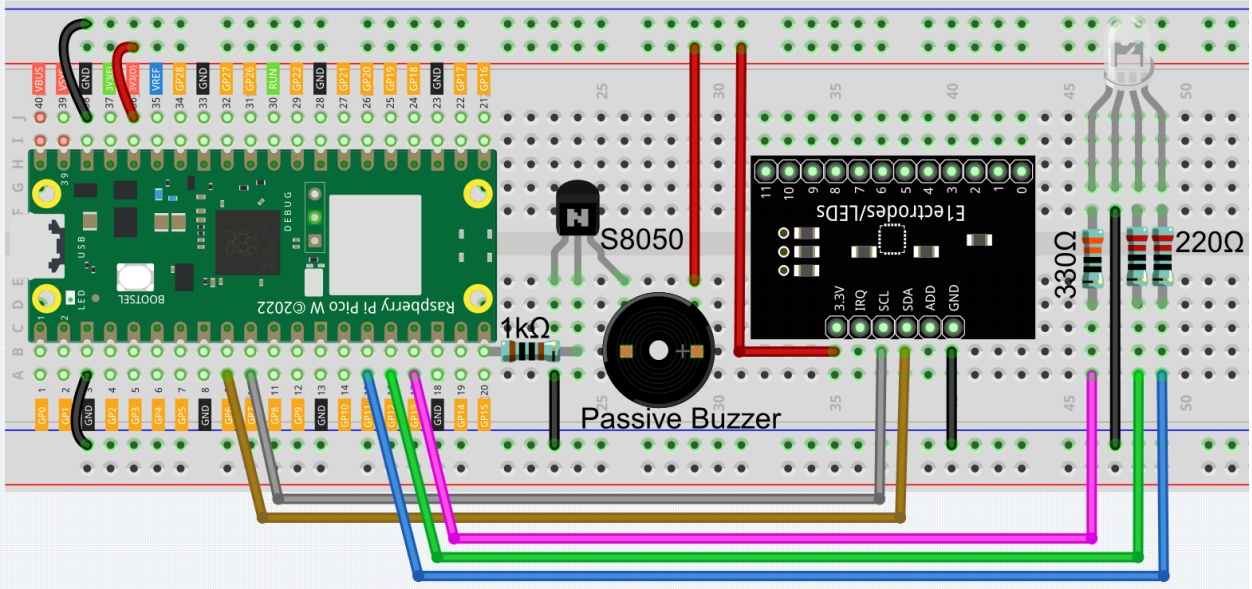
Schematic



To turn the fruit into a piano key, you still need to connect the electrodes on the MPR121 to the fruit (e.g. into the banana handle).

In the beginning, MPR121 will initialize and each electrode will get a value based on the current charge; when a conductor (such as a human body) touches an electrode, the charge will shift and rebalance. As a result, the electrode's value is different from its initial value, telling the main control board that it has been touched. During this process, ensure that the wiring of each electrode is stable so that its charge is balanced when initializing.

Wiring



Code

Note:

- Open the `7.9_fruit_piano.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the library called `mpr121.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *1.4 Upload the Libraries to Pico*.

```
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)


# buzzer
NOTE_A3 = 220
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
```

(continues on next page)

(continued from previous page)

```
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)))


def dark():
    red.duty_u16(0)
    green.duty_u16(0)
    blue.duty_u16(0)

# main project
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:
```

(continues on next page)

(continued from previous page)

```
        noTone(buzzer)
        dark()
    lastState = currentState
```

Please do not touch the fruit before the program runs to avoid getting a non-correct reference during initialization. After the program runs, touch the fruit gently, the buzzer will sound the corresponding tone and the RGB light will flash once randomly.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.51 7.10 Reversing Aid

This project uses an LED, a buzzer and an ultrasonic module to create a reversing assist system. We can put it on a remote control car to simulate the the actual process of reversing a car into a garage.

Required Components

In this project, we need the following components.

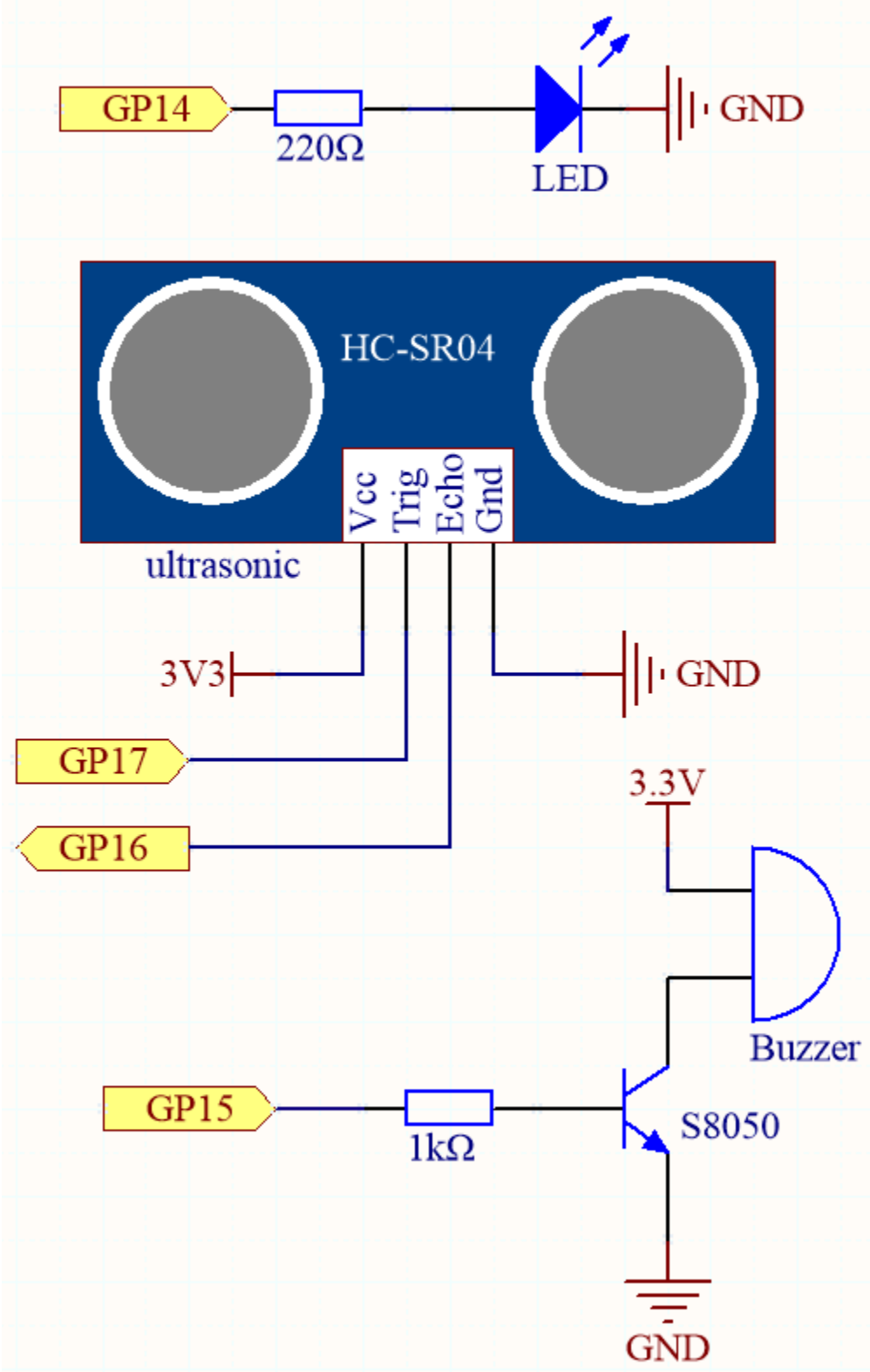
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

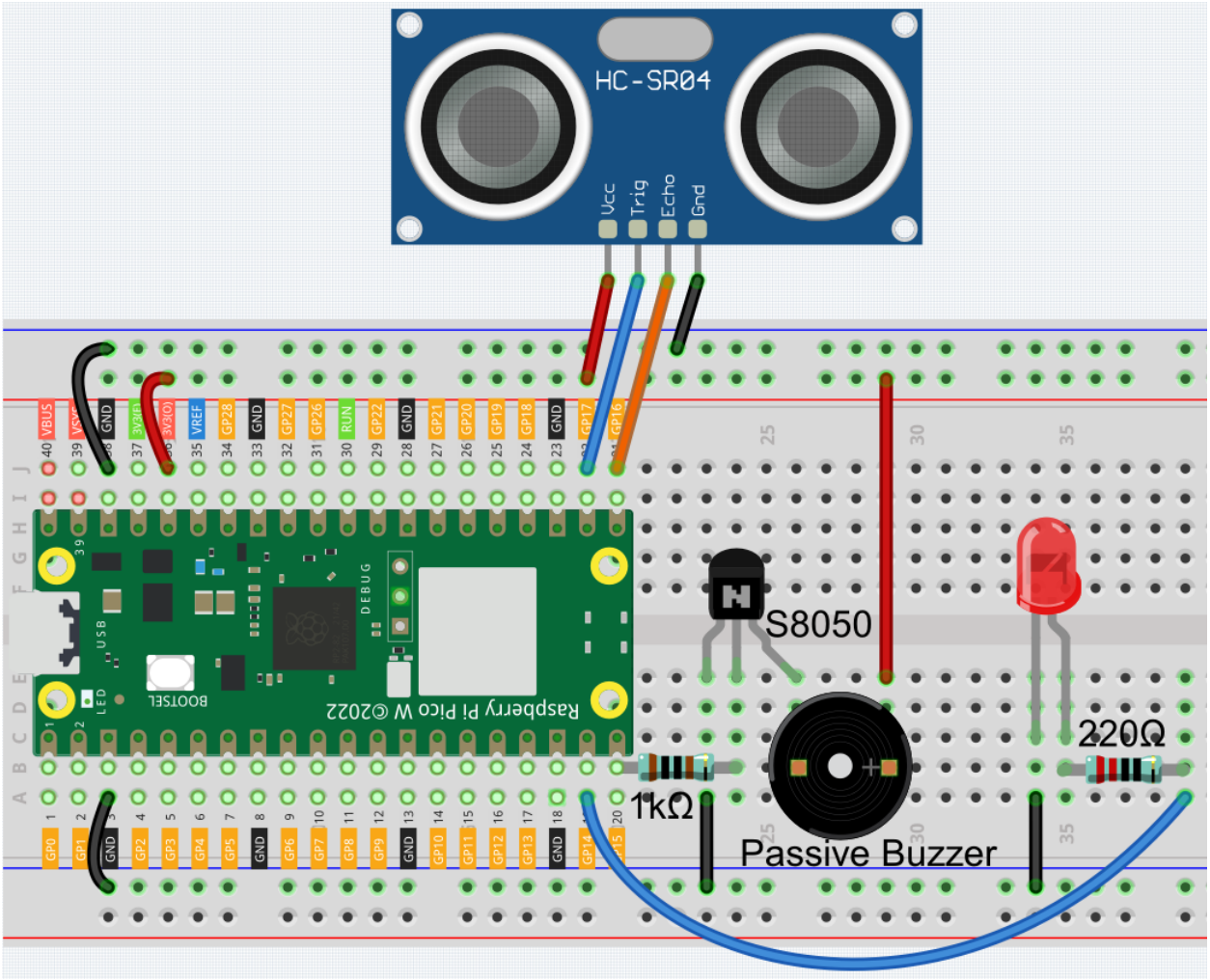
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Transistor*	1(S8050)	
6	*Resistor*	2(1K, 220)	
7	Passive *Buzzer*	1	
8	*LED*	1	
9	*Ultrasonic Module*	1	

Schematic


GP14
220Ω
LED
GND
HC-SR04
Vcc
Trig
Echo
Gnd
ultrasonic
3V3
GND
GP17
GP16
3.3V
Buzzer
GP15
1kΩ
S8050
GND

Wiring



Code

Note:

- Open the `7.10_reversing_aid.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

```
import machine
import time
import _thread


buzzer = machine.Pin(15, machine.Pin.OUT)
led = machine.Pin(14, machine.Pin.OUT)

TRIG = machine.Pin(17,machine.Pin.OUT)
ECHO = machine.Pin(16,machine.Pin.IN)
```

(continues on next page)

(continued from previous page)

```
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() # For timeout, re-read distance
    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)
    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
```

(continues on next page)

(continued from previous page)

```
    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
```

- As soon as the program runs, the ultrasonic sensor will continuously read the distance to the obstacle in front of you, and you will be able to see the exact distance value on the shell.
- The LED and buzzer will change the frequency of blinking and beeping depending on the distance value, thus indicating the approach of the obstacle.
- The *6.1 Measuring Distance* article mentioned that when the ultrasonic sensor works, the program will be paused.
- To avoid interfering with the LED or buzzer timing, we created a separate thread for ranging in this example.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.52 7.11 Somatosensory Controller

If you watch a lot of robot movies, you've probably seen images like this. The protagonist turned his wrist and the giant robot followed; the protagonist shakes his fist, and the robot follows, which is very cool.

The use of this technology is already common in universities and research institutes, and the arrival of 5G will greatly expand its application areas. "Surgical robot da Vinci" remote surgery medical is a typical example.

A robotic system of this type is typically composed of two modules: a human motion capture module and a robotic arm actuation module (some application scenarios also include a data communication module).

The MPU6050 is used here to implement human motion capture (by mounting it on a glove) and the servo is used to represent robotic arm motion.

Required Components

In this project, we need the following components.

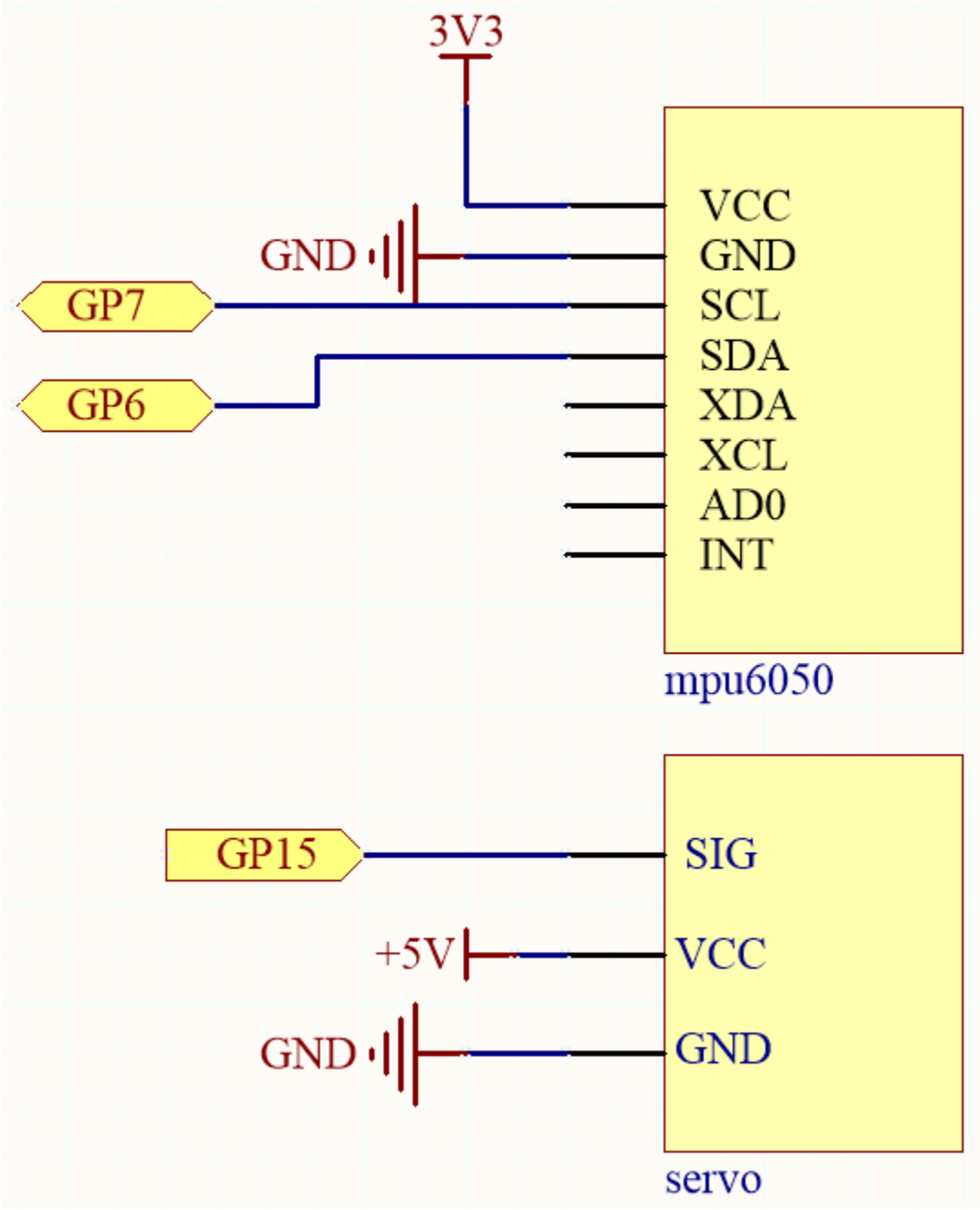
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*MPU6050 Module*	1	
6	*Servo*	1	

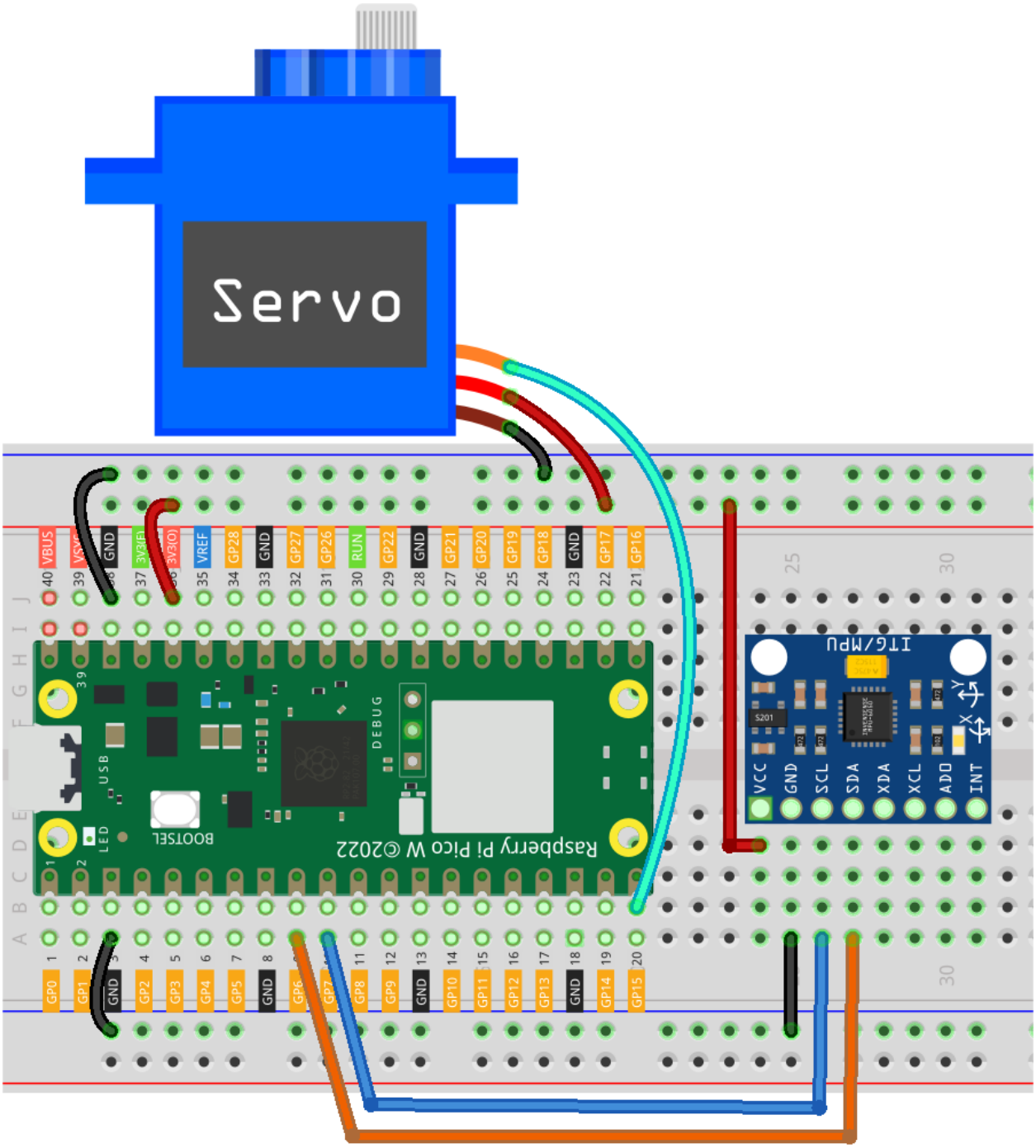
Schematic



The MPU6050 calculates the attitude angle based on the acceleration values in each direction.

The program will control the servo to make the corresponding deflection angle as the attitude angle changes.

Wiring



Code

Note:

- Open the `7.11_somatosensory_controller.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.

- Here you need to use the `imu.py` and `vector3d.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *1.4 Upload the Libraries to Pico*.

```
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):
    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))
```

As soon as the program runs, the servo will turn left and right as you tilt the MPU6050 (or turn your wrist if it is mounted on a glove).

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook!

Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

4.53 7.12 Digital Bubble Level

A bubble Level, is an instrument designed to indicate whether a surface is horizontal (level) or vertical (plumb). There are different types of spirit levels used by carpenters, stonemasons, bricklayers, other building trades workers, surveyors, millwrights, and other metalworkers, as well as in some photographic and videographic work.

Here we make a digital bubble level using MPU6050 and 8x8 LED matrix. When you deflect the MPU6050, the bubble on the LED matrix will also be deflected.

Required Components

In this project, we need the following components.

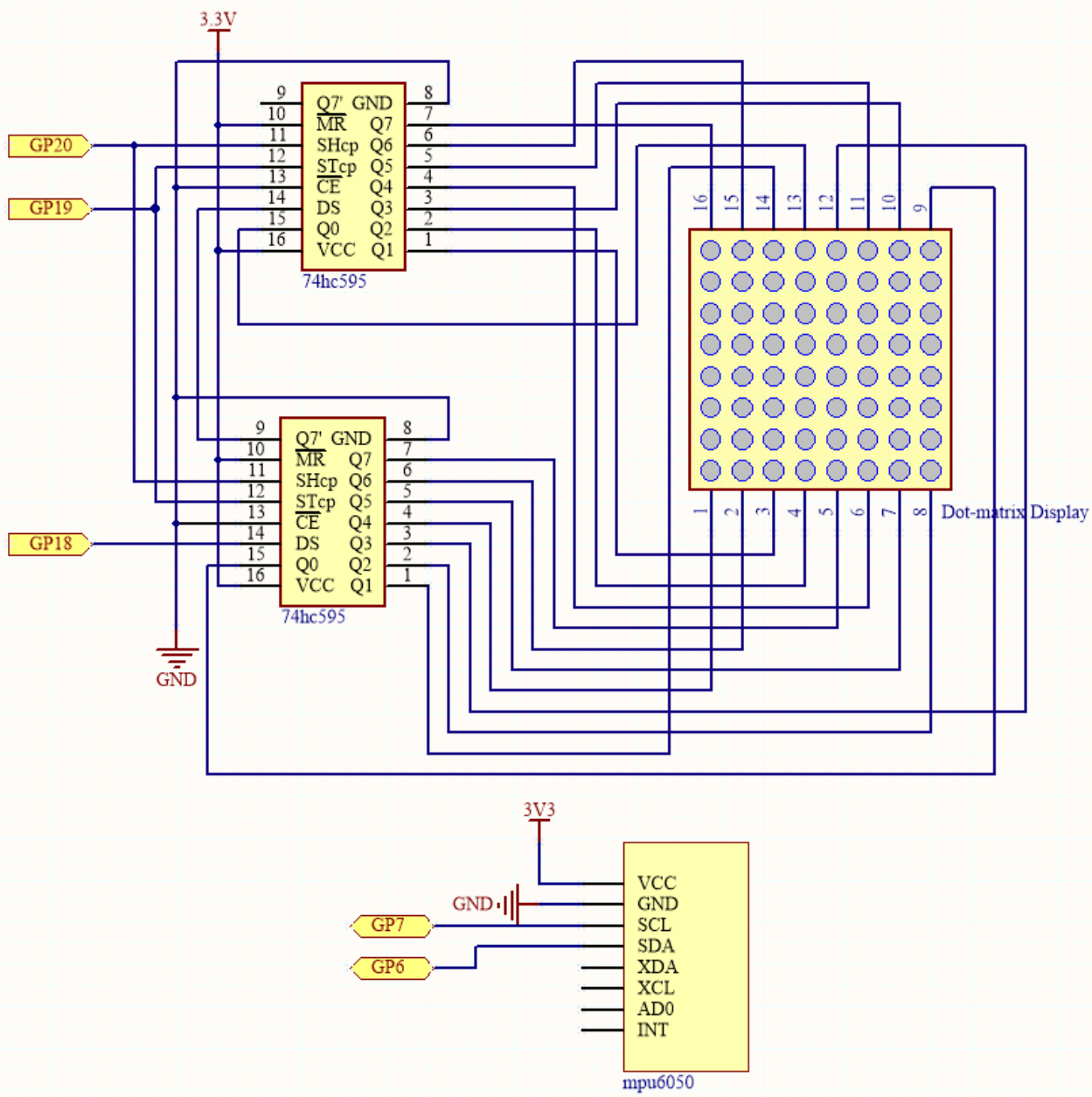
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*LED Dot Matrix*	1	
6	*74HC595*	2	
7	*MPU6050 Module*	1	

Schematic

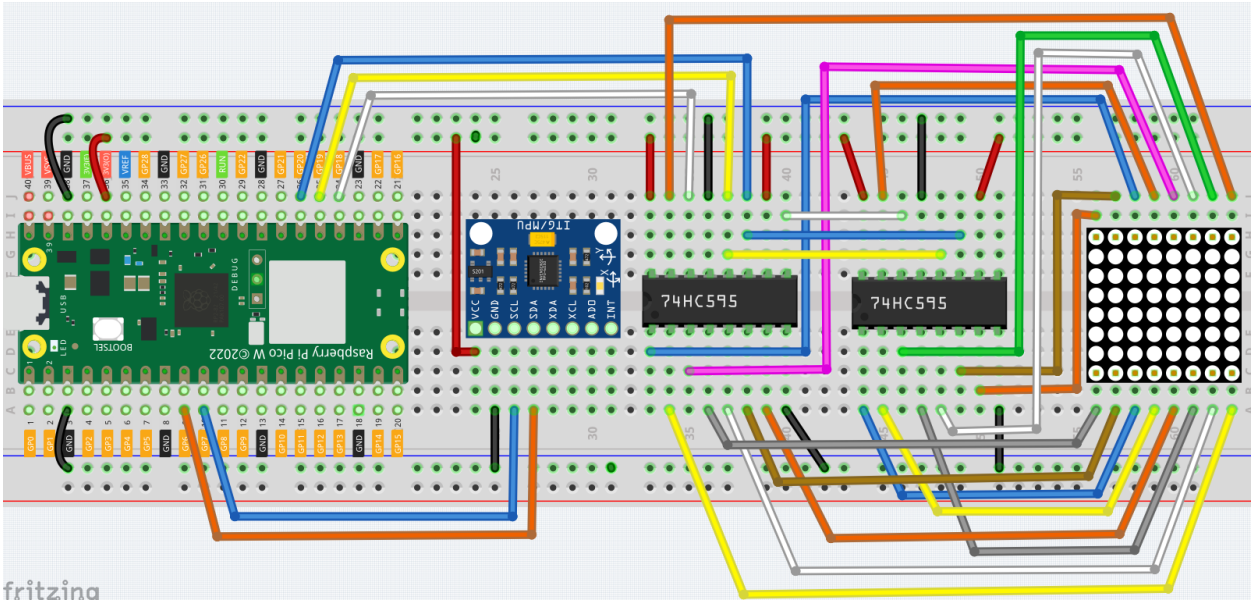


The MPU6050 takes the acceleration values in each direction and calculates the attitude angle.

As a result, the program draws a 2x2 dot on the dot matrix based on data from the two 74HC595 chips.

As the attitude angle changes, the program sends different data to the 74HC595 chips, and the position of the dot changes, creating a bubble effect.

Wiring



Code

Note:

- Open the `7.12_digital_bubble_level.py` file under the path of `kepler-kit-main/micropython` or copy this code into Thonny, then click "Run Current Script" or simply press F5 to run it.
- Don't forget to click on the "MicroPython (Raspberry Pi Pico)" interpreter in the bottom right corner.
- For detailed tutorials, please refer to *Open and Run Code Directly*.
- Here you need to use the `imu.py` and `vector3d.py`, please check if it has been uploaded to Pico W, for a detailed tutorial refer to *1.4 Upload the Libraries to Pico*.

```
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)

# 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):
```

(continues on next page)

(continued from previous page)

```
    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])
        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():
```

(continues on next page)

(continued from previous page)

```
    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:
    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
```

Once you have run the program, place the breadboard on a level surface. A dot will appear in the center of the LED matrix (if it isn't in the center, the MPU6050 may not be level). When you deflect the breadboard, the dot will move in the direction you deflected.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

CHAPTER

FIVE

IOT PROJECTS

This section will walk you through how to get the Pico W connected to the network and complete some interesting IoT projects.

But the first step before moving on to this section is to complete **1. Get Started** inside *For MicroPython User* to complete the installation of the Thonny IDE, install the Micropython firmware for the Raspberry Pi Pico W and upload the libraries.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

5.1 1. Access to the Network

Raspberry Pi Pico W is very similar to Raspberry Pi Pico, with the same GPIO, microUSB port, and size. The only difference is the addition of the CYW43439 2.4-GHz Wi-Fi chip from Infineon. Now let's see how it connects to our Wi-Fi network.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	

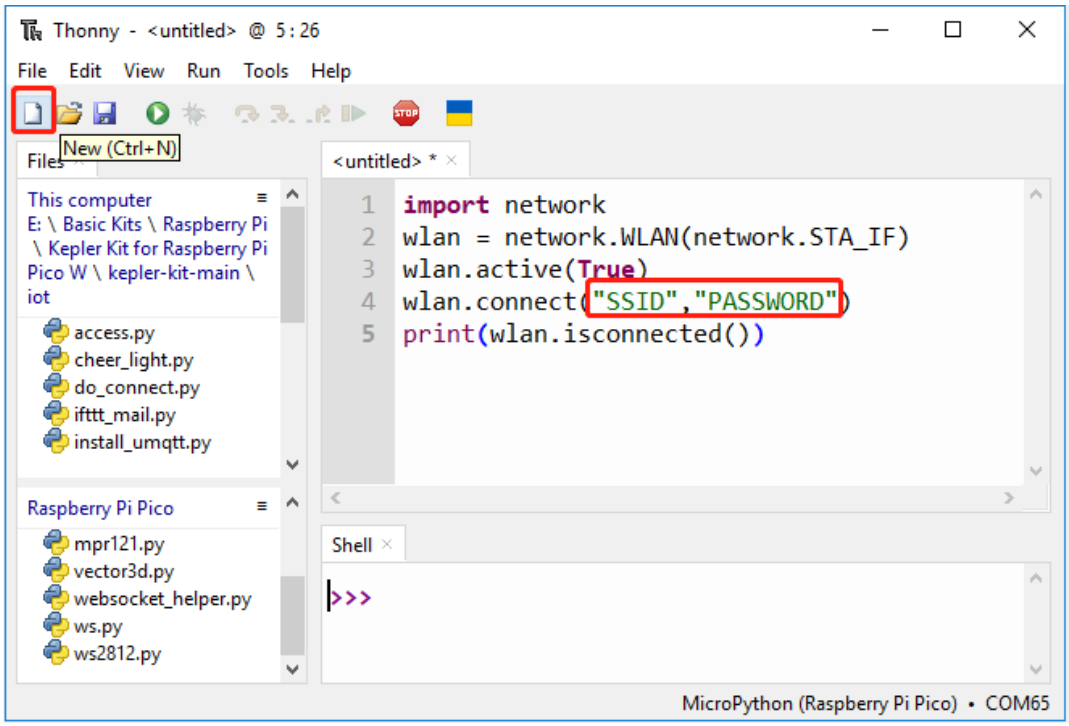
5.1.1 1. Connect to the Internet

With just five lines of MicroPython, our Raspberry Pi Pico W is happily connected to the Internet.

These 5 lines of code can be run directly from the Shell, pressing the `Enter` after you typing. Or refer to the following method and create a new `.py` file to run it.

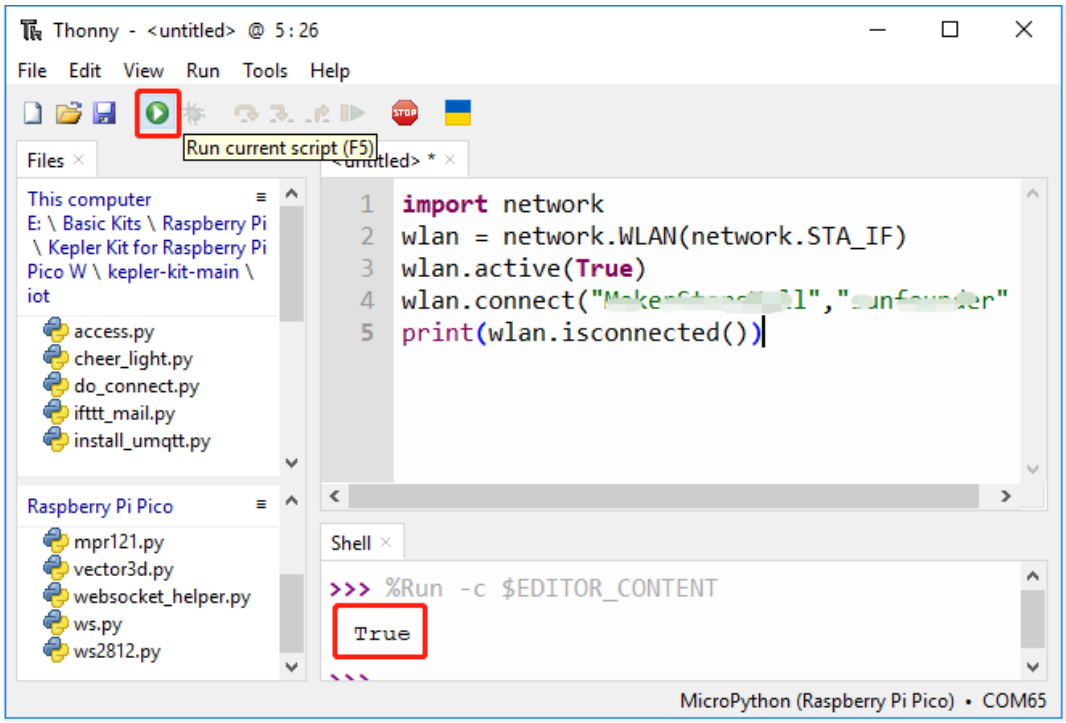
```
import network
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect("SSID","PASSWORD")
print(wlan.isconnected())
```

1. Create a new script by clicking the **New** button on Thonny, and copy and paste the above code, changing the `SSID` and `PASSWORD` to your own.



2. To run the script, click the **Run current script** button or press F5. If the connection is successful, `true` will be printed.

> **Note:** Make sure the Raspberry Pi Pico W is connected to computer via a USB cable, then click on the bottom right corner to select MicroPython (Raspberry Pi Pico).COMXxx as the interpreter.



5.1.2 2. Timeout Judgment and Show IP

Considering some poor network conditions, let's add some timeout judgment to the code.

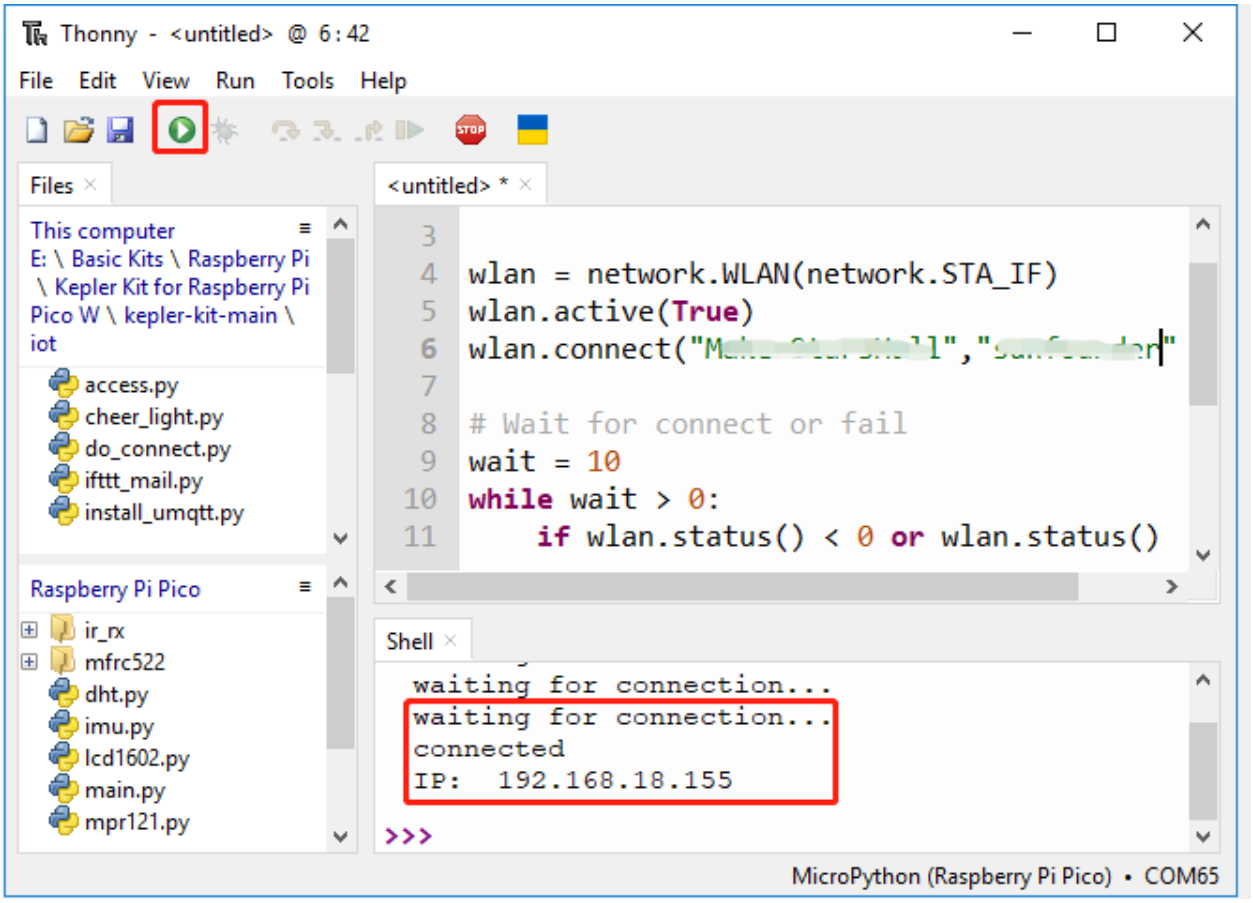
If the connection is successful, the IP of Pico W will be displayed after copying and running the script.

```
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)

# Handle connection error
if wlan.status() != 3:
    raise RuntimeError('wifi connection failed')
else:
    print('connected')
    print('IP: ', wlan.ifconfig()[0])
```



- `wlan.status()` function: Returns the current status of the wireless connection, the return value is shown in the table below.

Status	Value	Description
STAT_IDLE	0	no connection and no activity,
STAT_CONNECTING	1	connecting in progress,
STAT_WRONG_PASSWORD	-3	failed due to incorrect password,
STAT_NO_AP_FOUND	-2	failed because no access point replied,
STAT_CONNECT_FAIL	-1	failed due to other problems,
STAT_GOT_IP	3	connection successful.

- `wlan.ifconfig()` function: Obtains IP addresses, subnet masks, gateways, and DNS servers. This method will return a 4-tuple containing the above information when called directly. In this case, we only print the IP address.
- class WLAN – MicroPython Docs

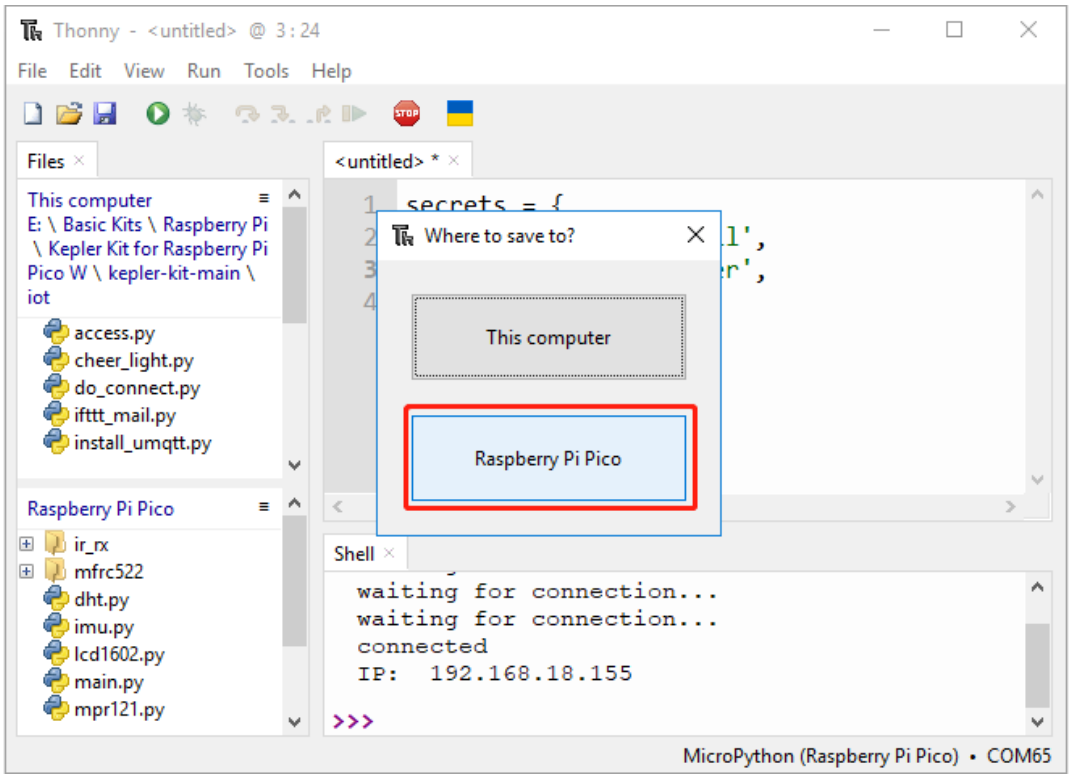
5.1.3 3. Save private information to `secrets.py`

When you share your Pico W project, you should not want others to see your Wi-Fi password or API key. For good security, we can create a `secrets.py` file to store your private information.

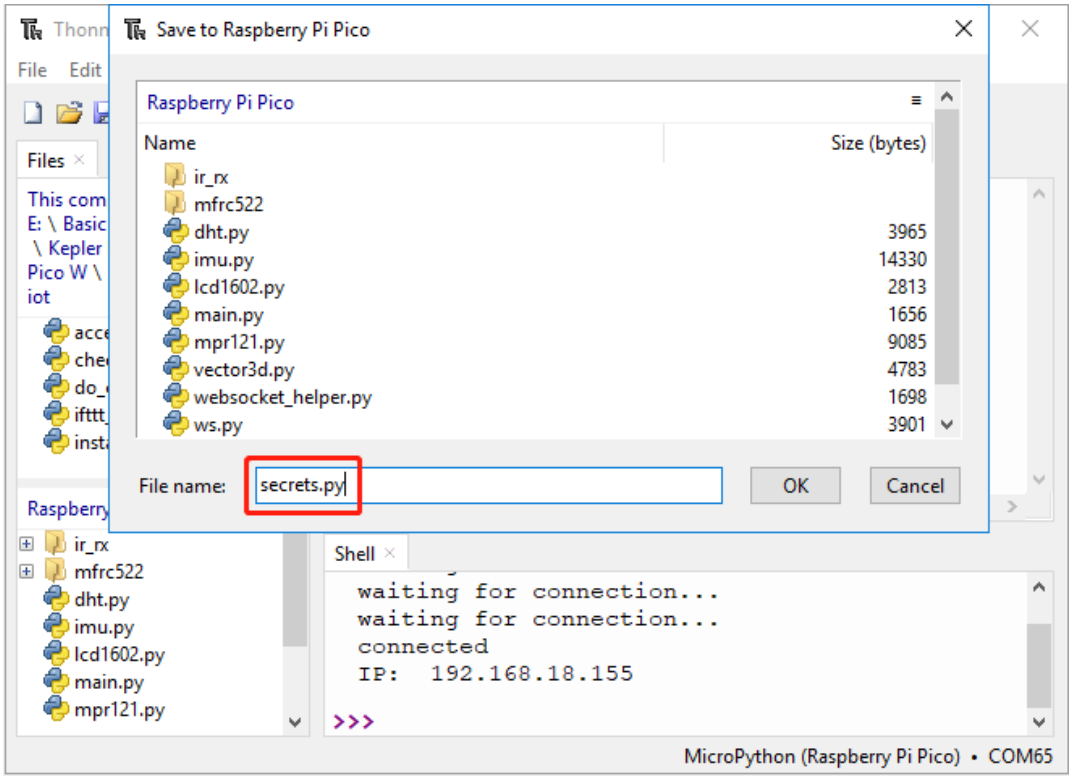
1. Copy the following code into a new script file on Thonny. Note to change `SSID` and `PASSWORD` to yours.

```
secrets = {
'ssid': 'SSID',
'password': 'PASSWORD',
}
```

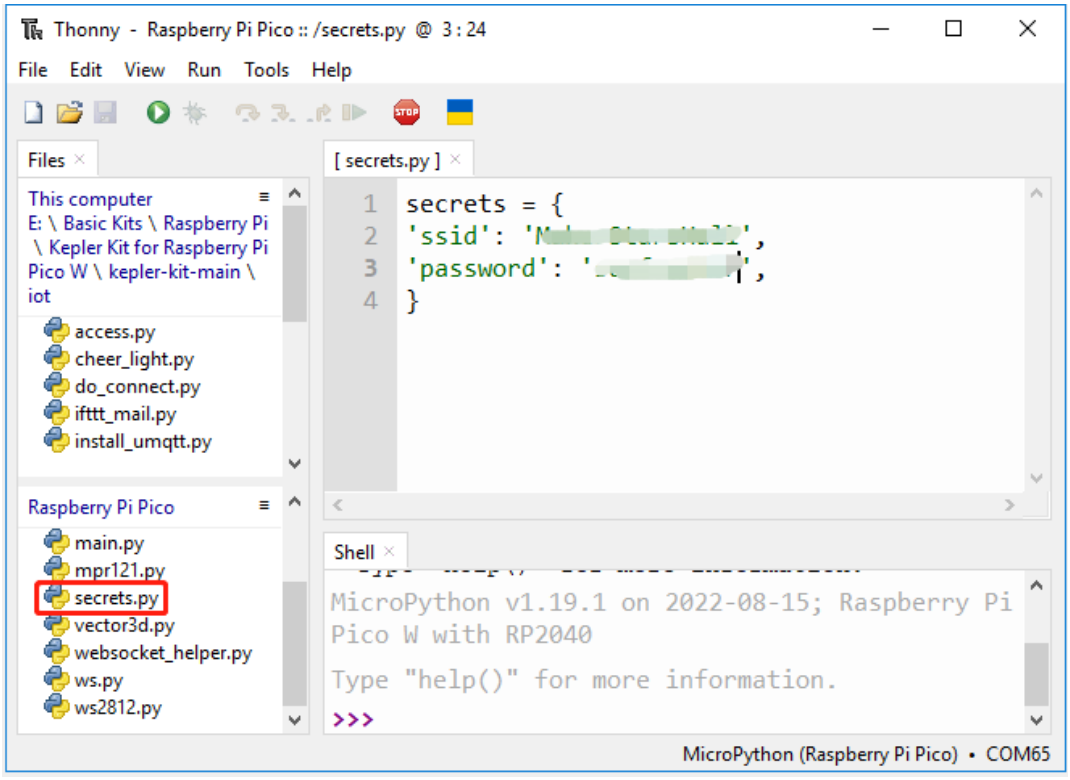
2. Choose Raspberry Pi Pico in the pop-up window that appears when you click the Save button or press `Ctrl`+S.



3. Set the name to `secrets.py`.



4. Now you can see this script in your Raspberry Pi Pico W.



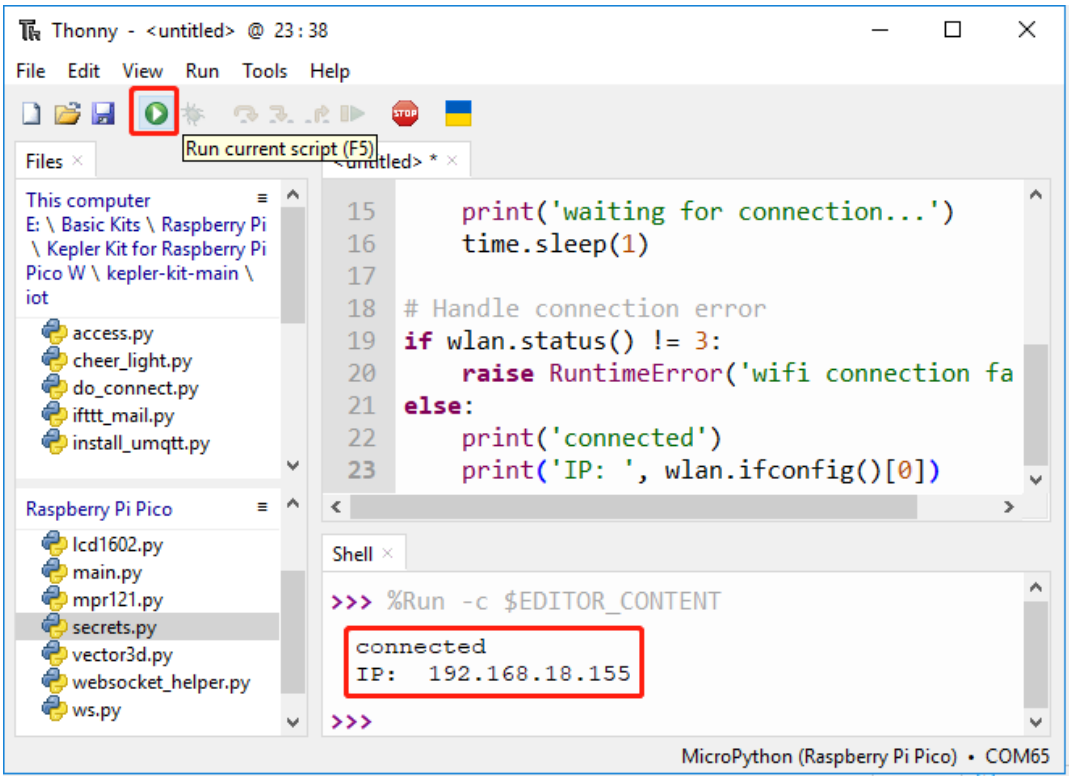
5. In other scripts, you can call it as follows. If you run it, you will see a successful Wi-Fi connection. The `secrets.py` file is imported as a library, so we need not worry about leakage of information.

```
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)

# Handle connection error
if wlan.status() != 3:
    raise RuntimeError('wifi connection failed')
else:
    print('connected')
    print('IP: ', wlan.ifconfig()[0])
```



5.1.4 4. Connect to the Internet via `do_connect.py`

Considering that each of our next projects will require a network connection, why don't we create a new `do_connect.py` file and write the relevant functions into it for reuse, which can greatly streamline the code of complex projects.

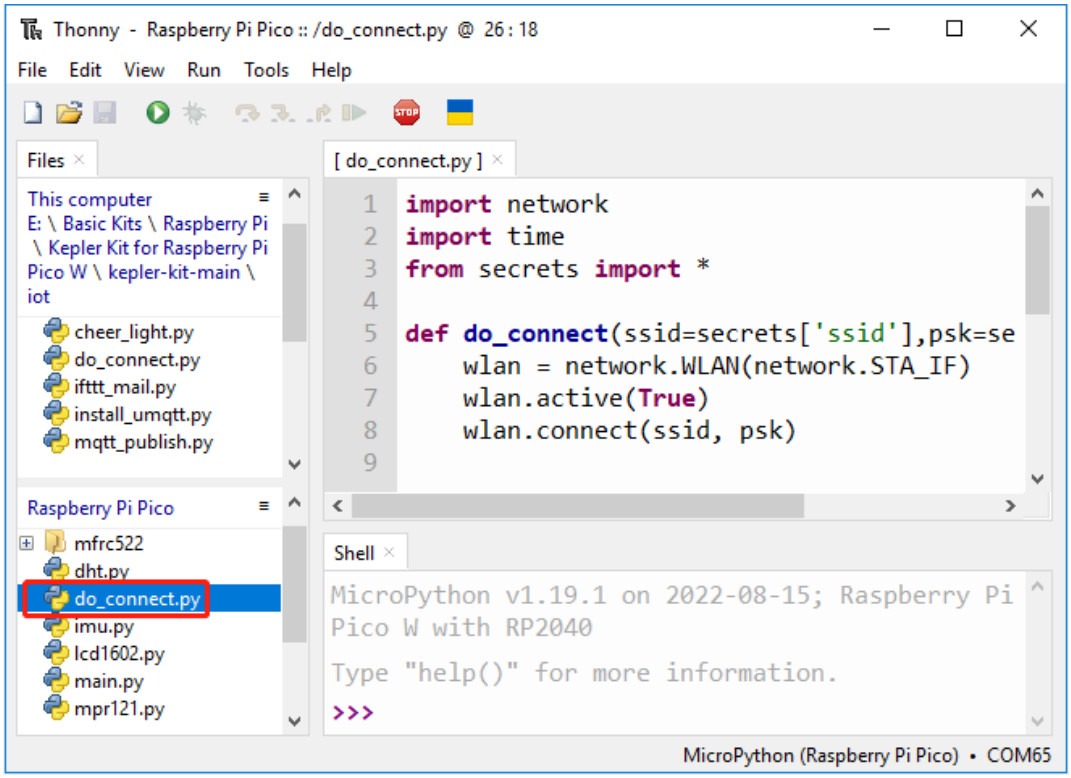
1. Copy the following code into a new script file, and save it to Raspberry Pi Pico as `do_connect.py`.

```
import network
import time
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. Calling it in other scripts by the following will allow the Raspberry Pi Pico W to connect to the network.

```
from do_connect import *
do_connect()
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

5.2 2. Follow the @CheerLights

This is a romantic project, join the LED color-changing community, which allows LEDs all over the world to change colors simultaneously.

You can place it in a corner of your office to remind yourself that you are not alone.

You can tweeting @cheerlights and including the color name in the tweet. This will change the LEDs around the world to the color you specify.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

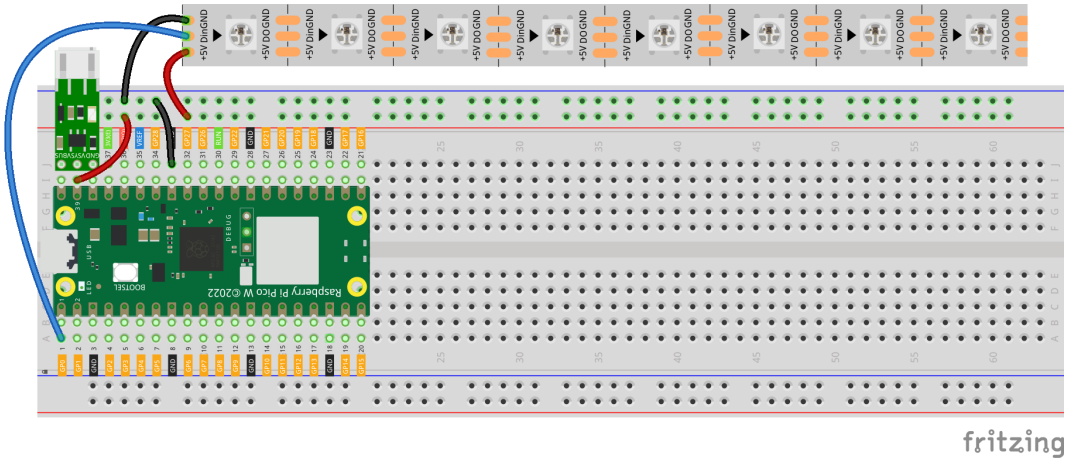
SN	COMPONENT	QUANTITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*WS2812 RGB 8 LEDs Strip*	1	
6	*Li-po Charger Module*	1	
7	18650 Battery	1	
8	Battery Holder	1	

Steps

1. Build the circuit.

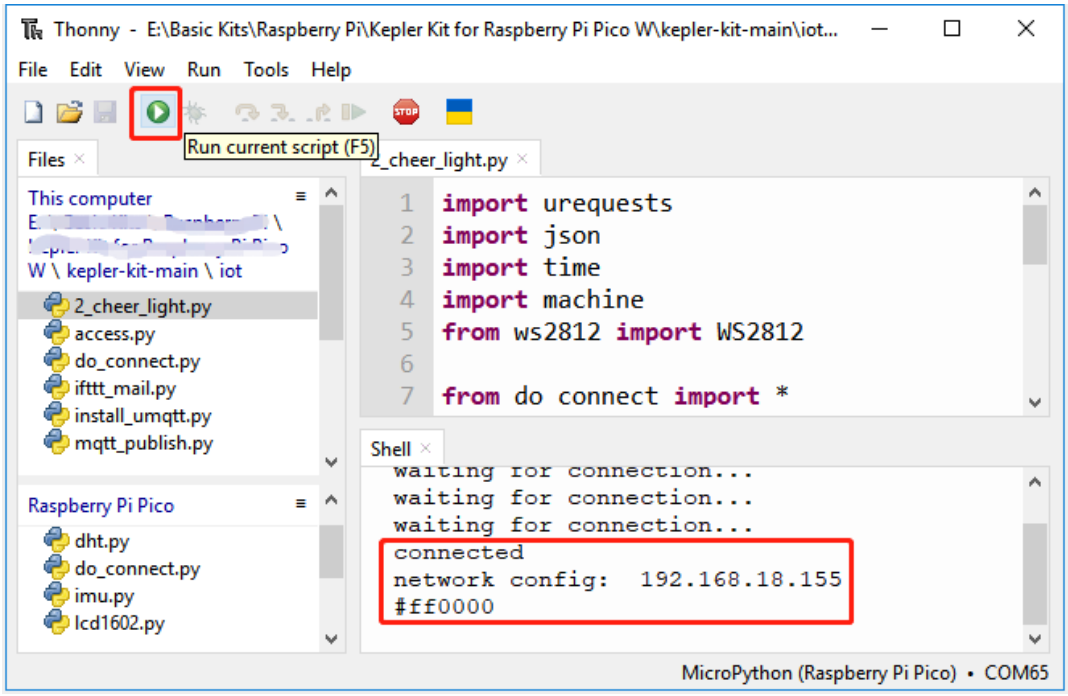
   The Li-po Charger Module used here will power your circuit so you can disconnect the USB cable and take your project somewhere else to play!

   Warning: Make sure your Li-po Charger Module is connected as shown in the diagram. Otherwise, a short circuit will likely damage your battery and circuitry.

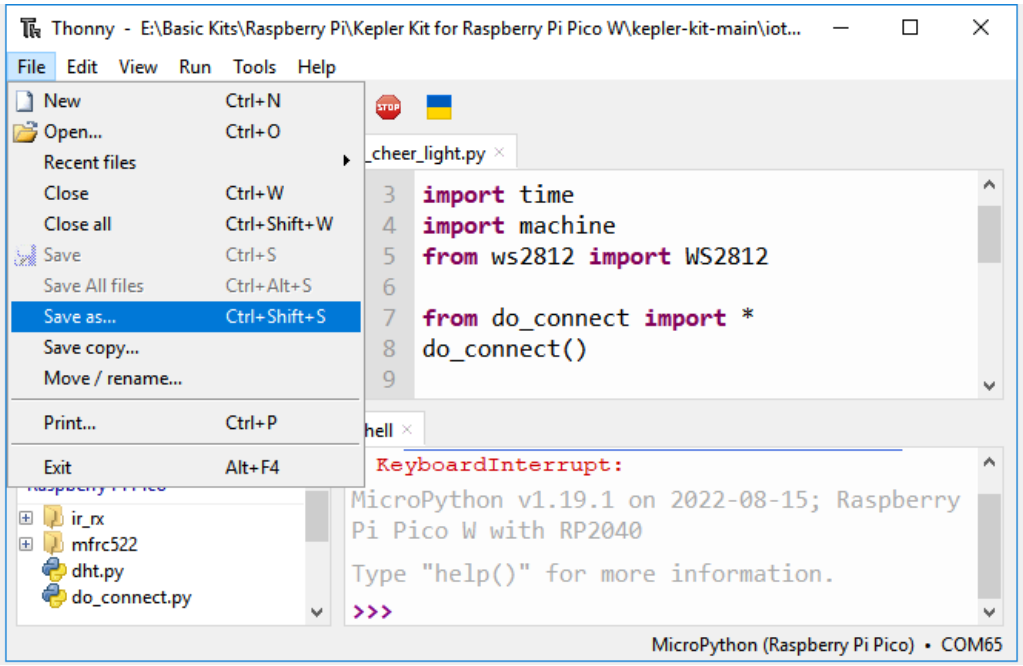


2. Switch the path to the folder where you downloaded the code package before and open the `2_cheer_light.py` file under the path of `kepler-kit-main/iot`.
3. To run the script, click the **Run current script** button or press F5, then you will see the connected prompt, the IP and the color (0xff0000 is red) in the Shell.

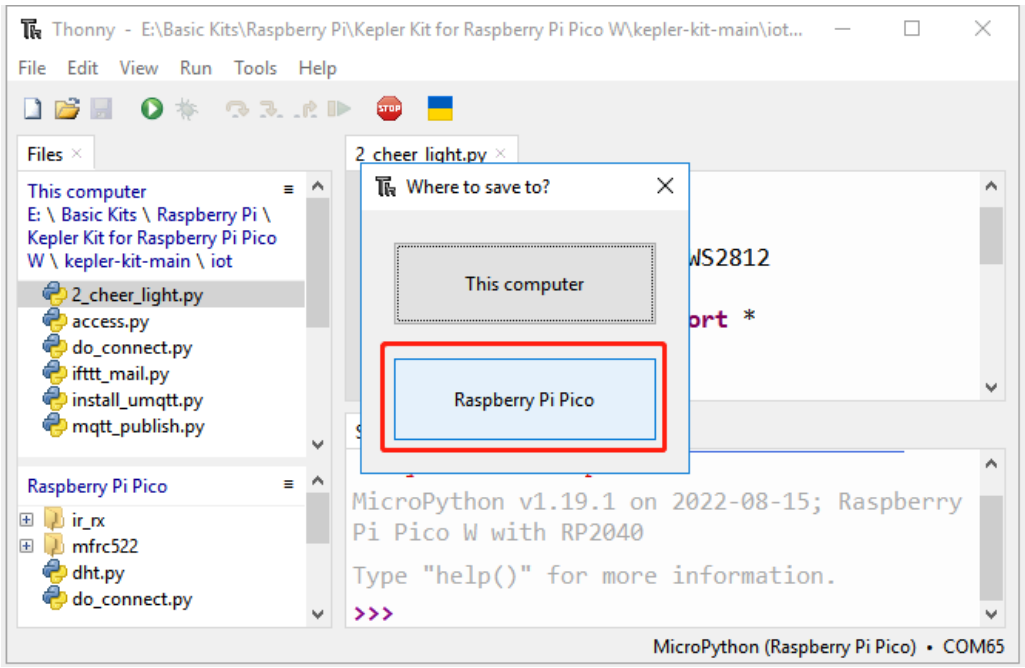
Note: Before running the code, you need to make sure you have `do_connect.py` and `secrets.py` scripts in your Pico W, if not please refer to *1. Access to the Network* to create them.



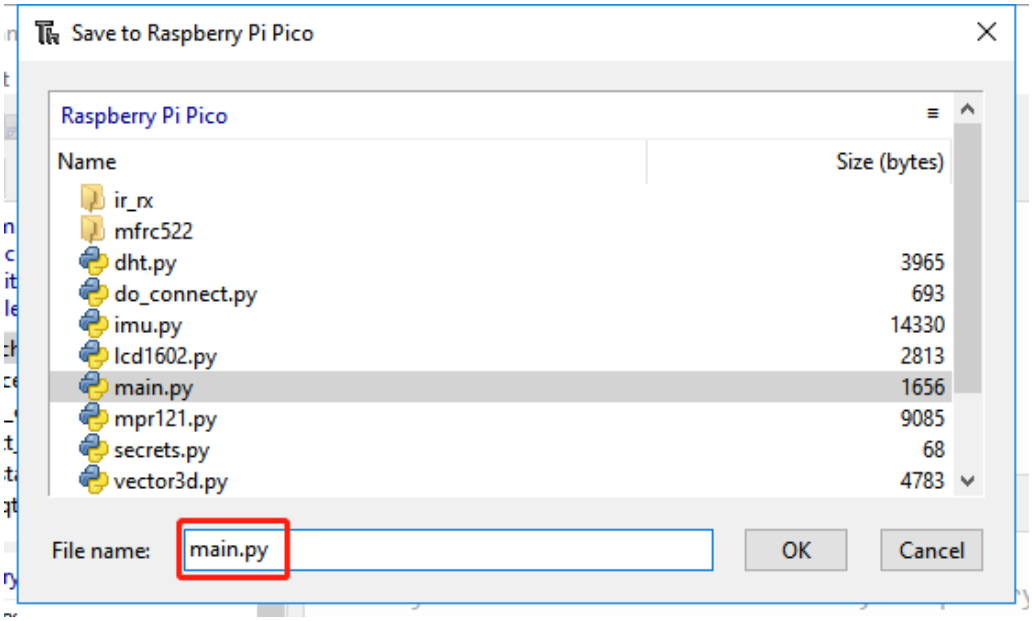
4. After the script runs, the WS2812 RGB strip will show a color, sometimes the color will change.
5. If you want to run this script on boot, you need to save it to the Raspberry Pi Pico W as `main.py`, as follows.
    - Stop the script from running and click **File** -> **Save as**.



- Select **Raspberry Pi Pico** in the popup window that appears.



- Set the file name to `main.py`. A prompt will appear if the same file already exists on your Pico W.



- You can now unplug the USB cable and use the Li-po Charger Module to power the Raspberry Pi Pico W. Put it in a corner and it will work automatically.

How it works?

The Raspberry Pi Pico W needs to be connected to the Internet, as described in *1. Access to the Network*. For this project, just use it.

```
from do_connect import *
do_connect()
```

Set WS2812 RGB strip, please refer to *3.3 RGB LED Strip* for its usage details.

```
import machine
from ws2812 import WS2812
ws = WS2812(machine.Pin(18), 8)
```

Now, we need a way to get the color of @CheerLights. There is a back-end system that takes the color changes from Twitter and posts them in JSON format to the URL: http://api.thingspeak.com/channels/1417/field/2/last.json.

If you open this URL directly in your browser, you will see something like the following. All we need is the `field2` data, which is a hexadecimal color-coded string.

```
{"created_at":"2022-08-16T06:12:44Z","entry_id":870488,"field2":"#ff00ff"}
```

We need to use the `urequests` module to get this data and the `json` module to convert this character into a python dictionary. The following code gets the latest @CheerLights color from the url and returns a color value that can be used by WS2812.

```
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'])
```

(continues on next page)

(continued from previous page)

```
            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
```

Finally, use a loop to make the ws2812 work once per minute.

```
while True:
    colour = get_colour()
    if colour is not None:
        ws.write_all(colour)
    time.sleep(60)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

5.3 3. Security System via @IFTTT

With this project, we make a security device that uses a PIR Sensor to detect when a burglar or stray animal breaks into your home. You will receive an email alert if this is the case.

Webhook will be used as the most basic service. A POST request is sent to IFTTT's service from Raspberry Pi Pico W. Using IFTTT, we will create an Applet to intercept the webhook and send an email.

1. Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

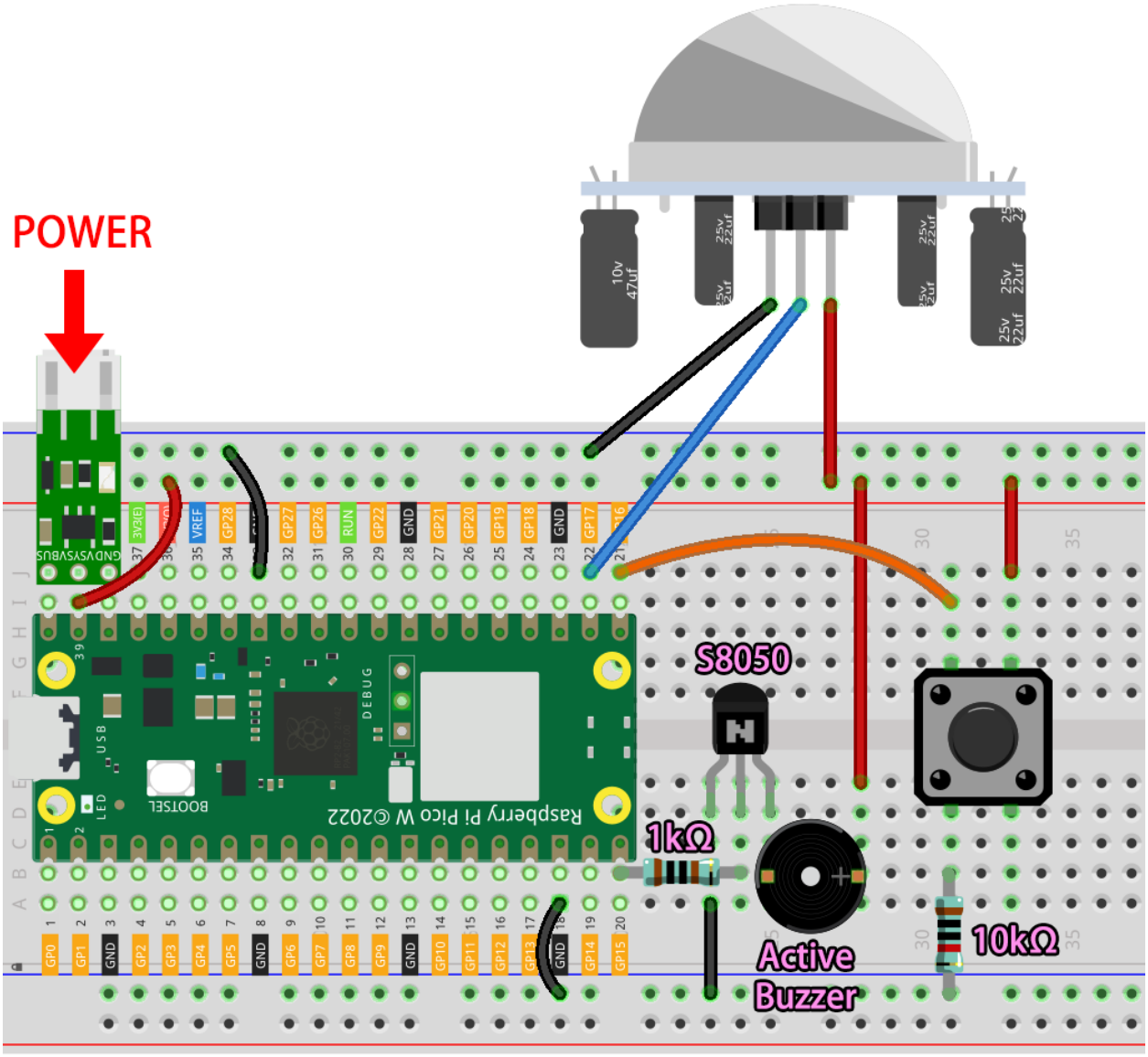
Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Transistor*	1(S8050)	
6	*Resistor*	2(1K, 10K)	
7	*Button*	1	
8	Active *Buzzer*	1	
9	*Li-po Charger Module*	1	
10	18650 Battery	1	
11	Battery Holder	1	
12	*PIR Motion Sensor Module*	1	

2. Build the Circuit

Warning: Make sure your Li-po Charger Module is connected as shown in the diagram. Otherwise, a short circuit will likely damage your battery and circuitry.



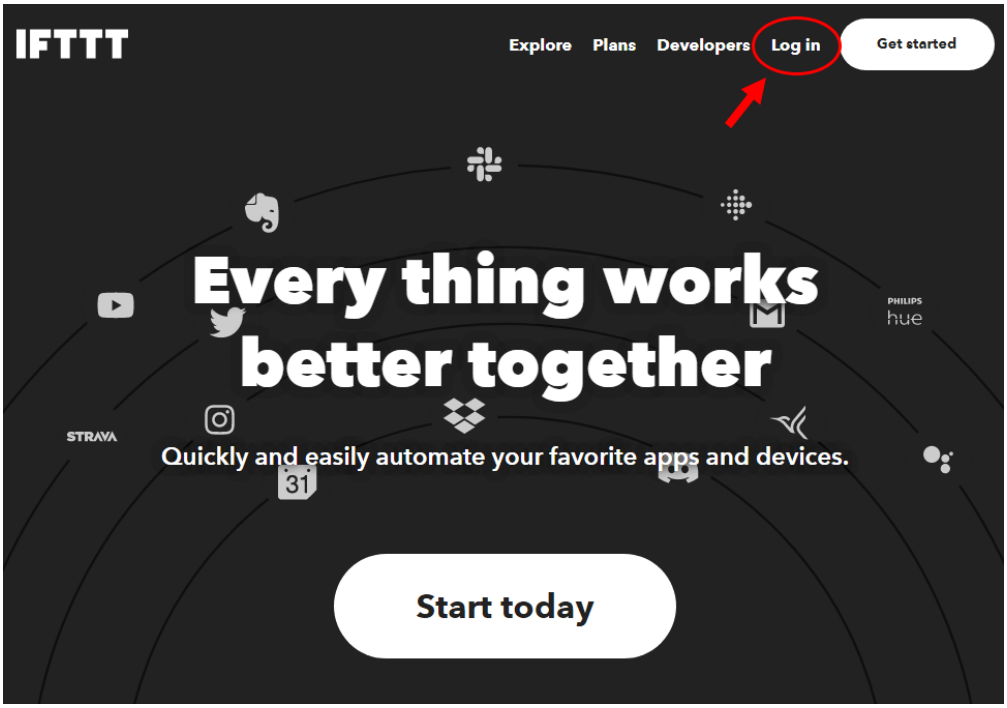
3. SET IFTTT

IFTTT is a free service that provides a variety of ways to connect different data services together.

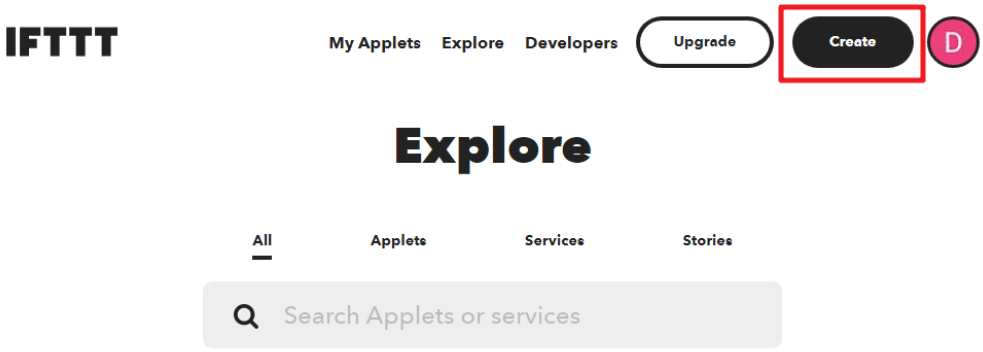
We'll create an Applet that reacts to a webhook (custom URL) that sends data to IFTTT. IFTTT will then send an email for you.

Please follow the steps below on IFTTT.

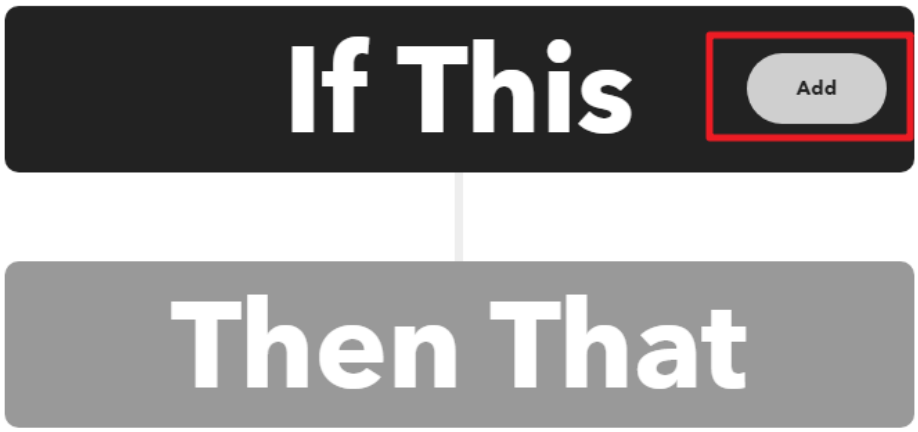
1. Visit to log in or create an account.



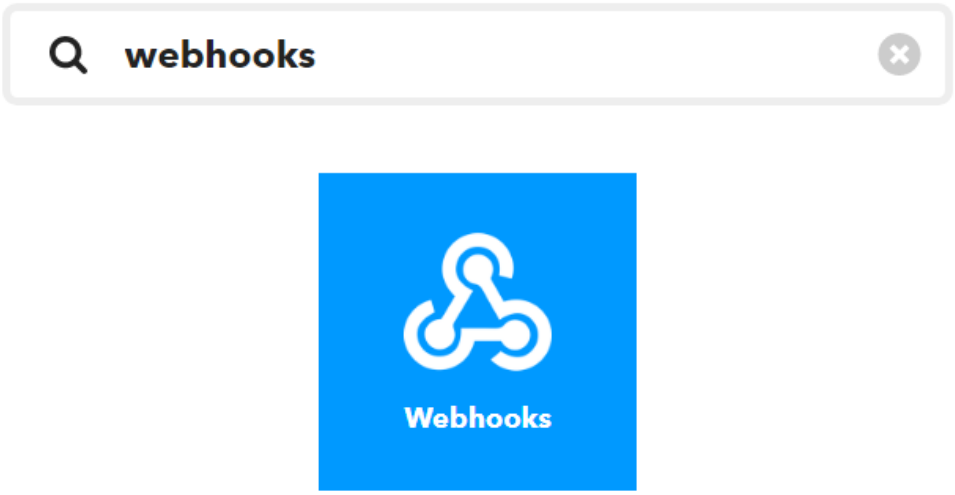
2. Click on **Create**.



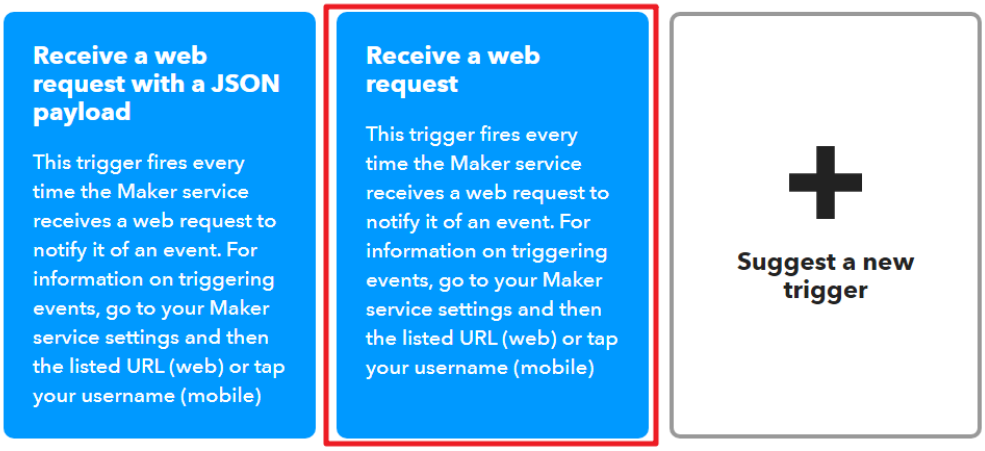
3. Add an **If This** event.



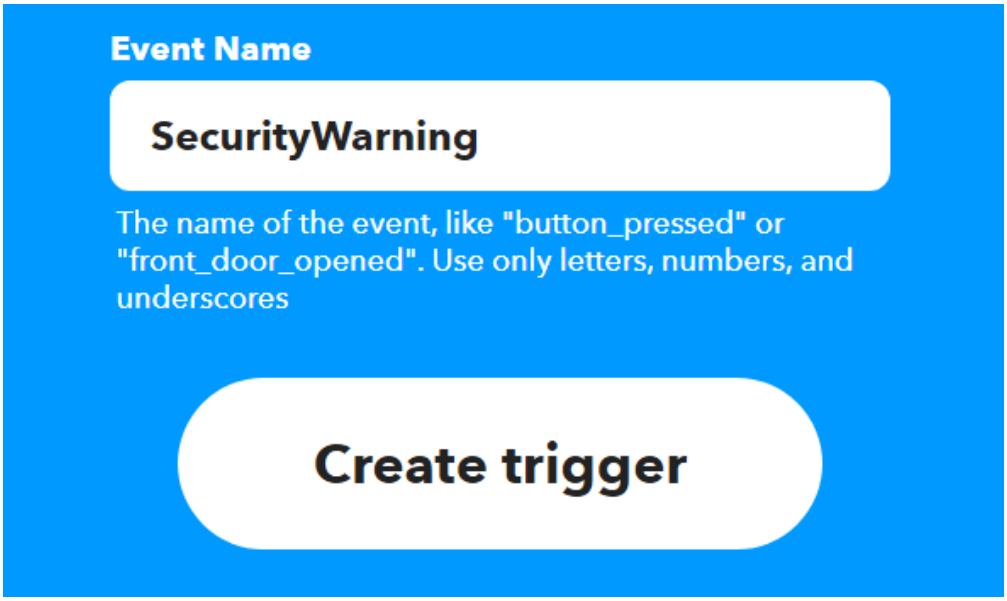
4. Search for **Webhooks**.



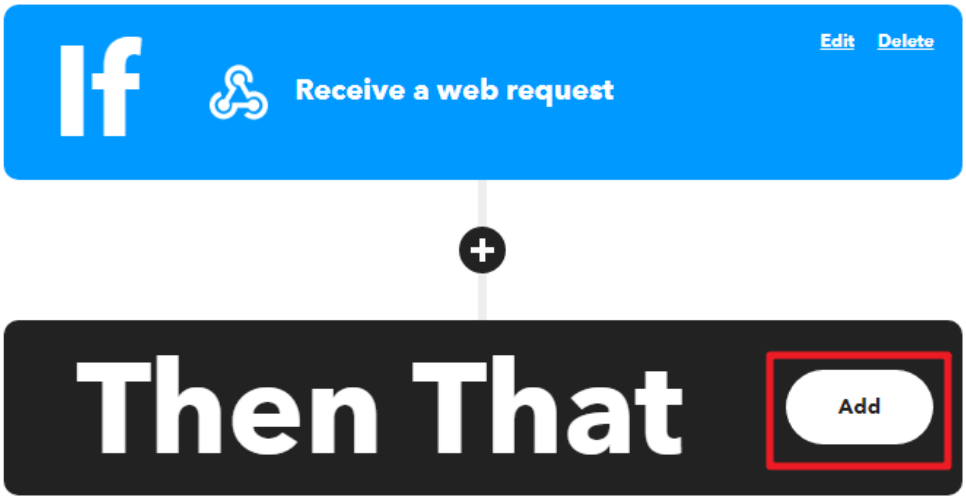
5. Tap **Receive a web request**.



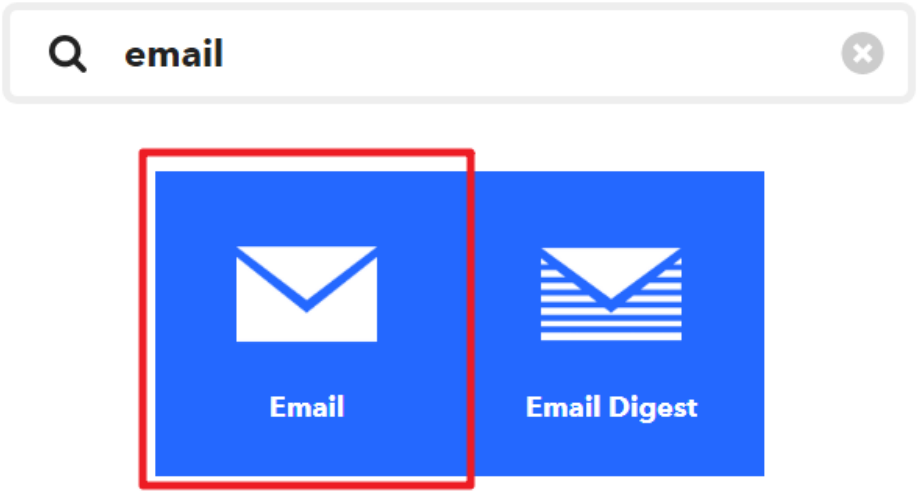
6. Fill in the event name (e.g. SecurityWarning), and click **Create trigger**.



7. Add a **Then That** event.



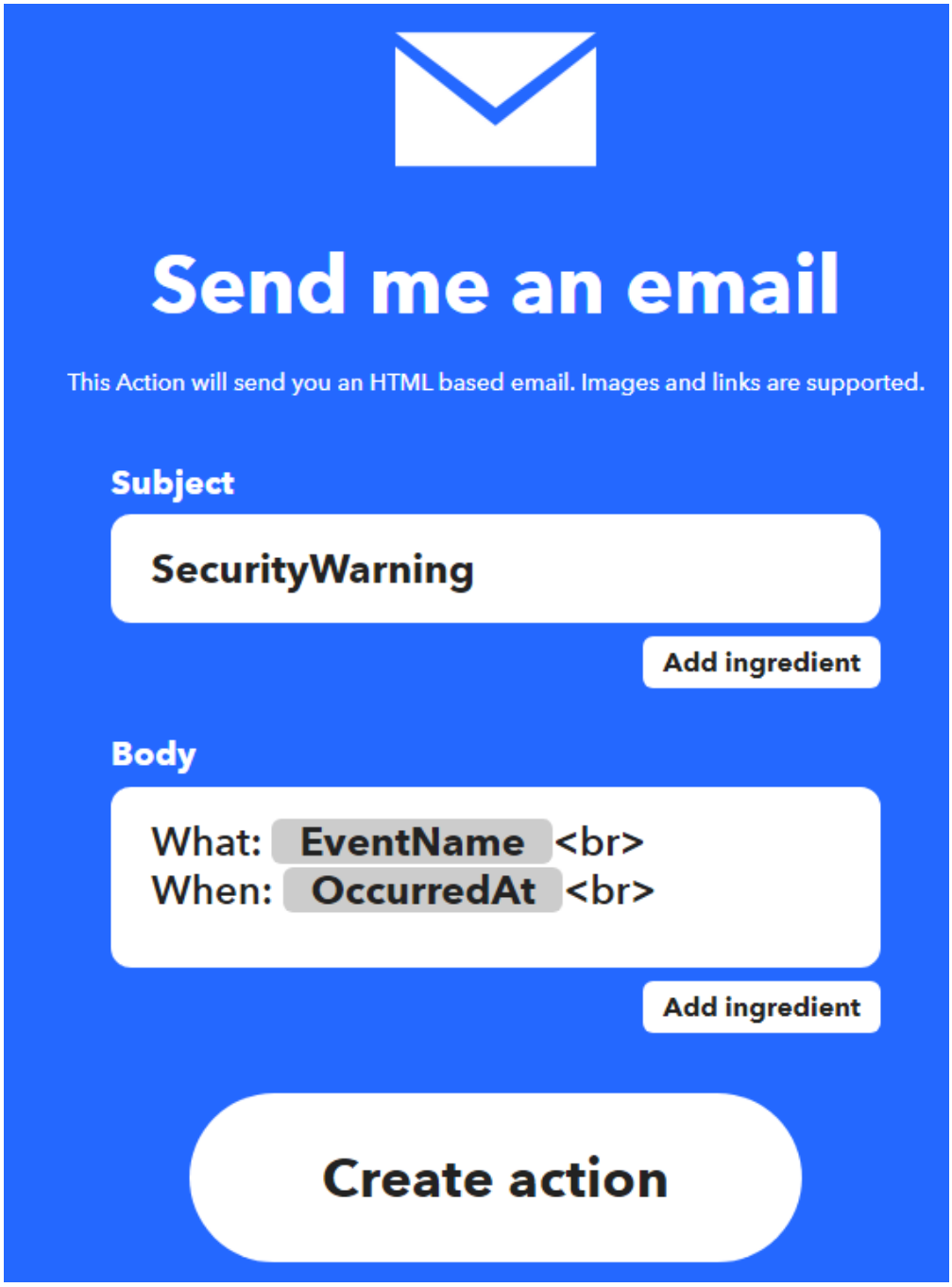
8. Search for Email.



9. Click **Send me an email**.



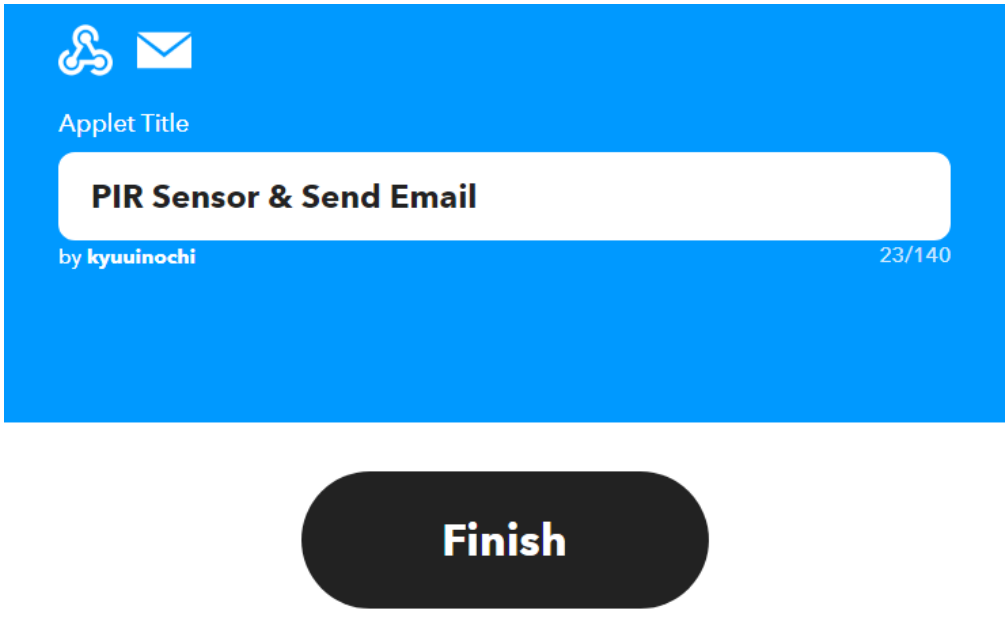
10. Fill in **Subject** and **Body**, then click on **Create action**.



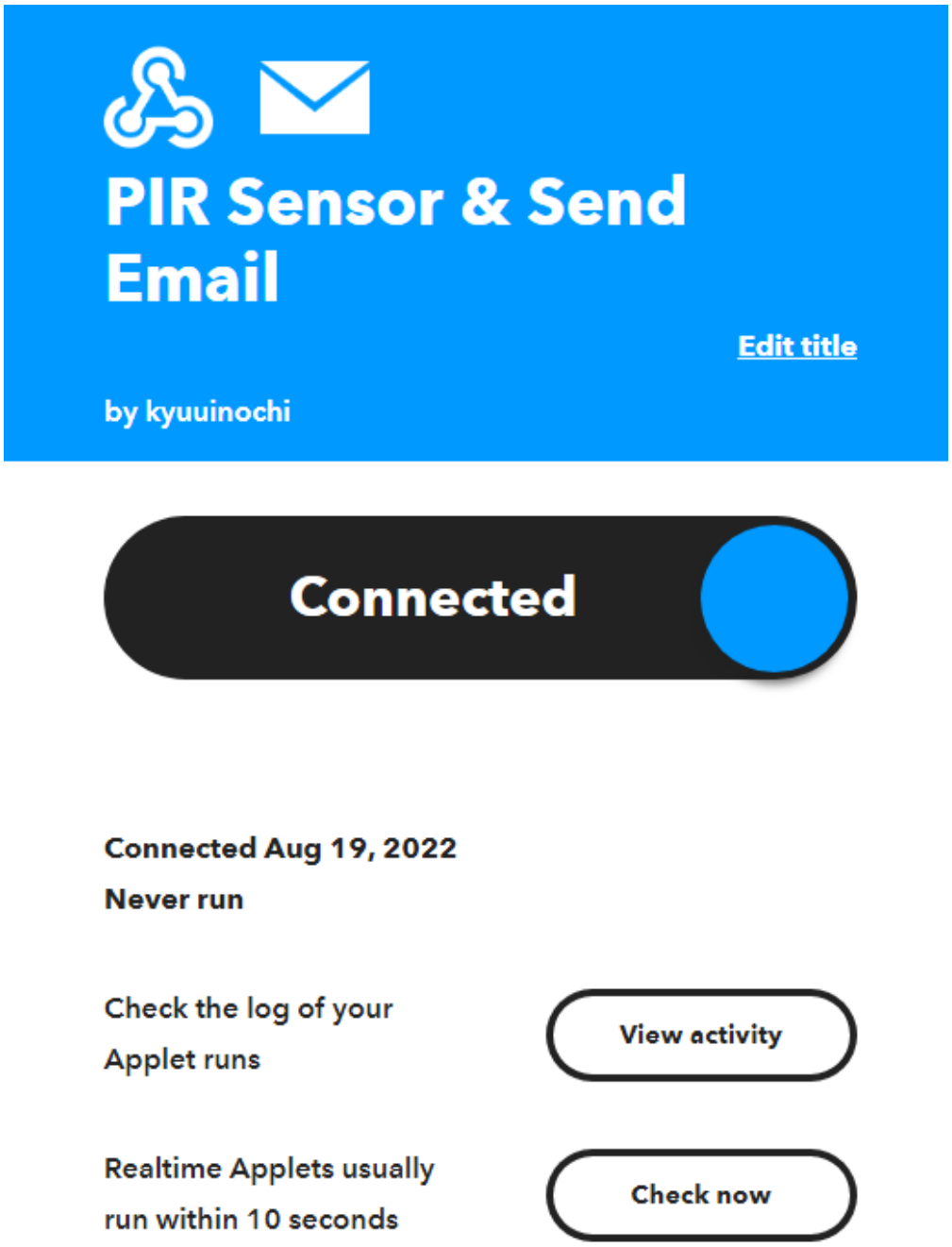
11. Click **Continue** to complete the configuration.



12. Modify the title name and you're finished.

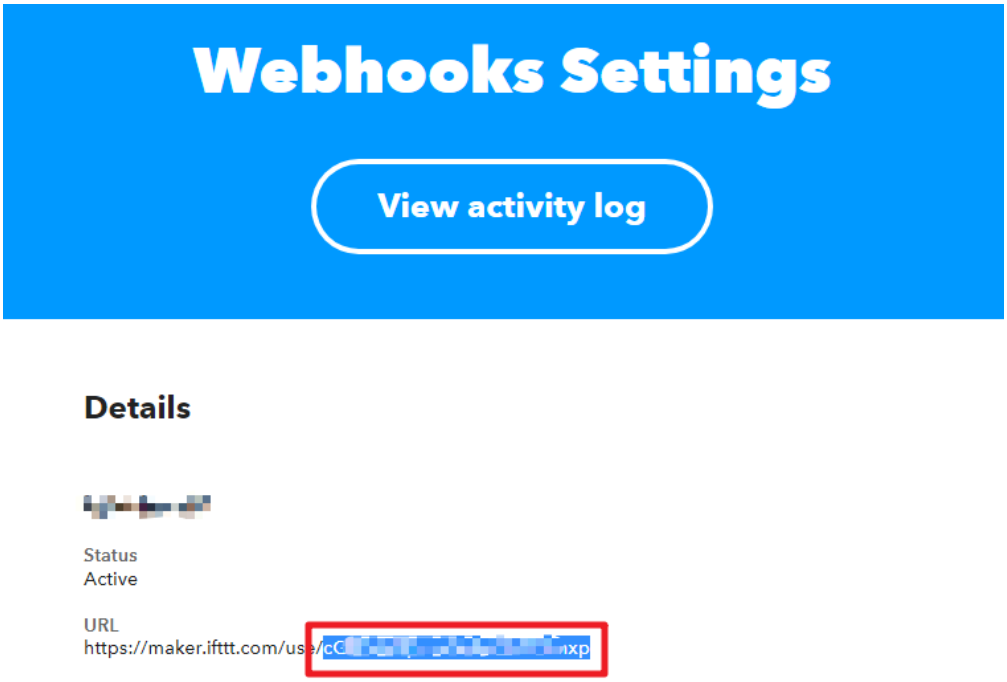


13. Now you will be automatically redirected to the Applet details page, where you can see that the Applet is currently connected and you can toggle the switch to start/close it.

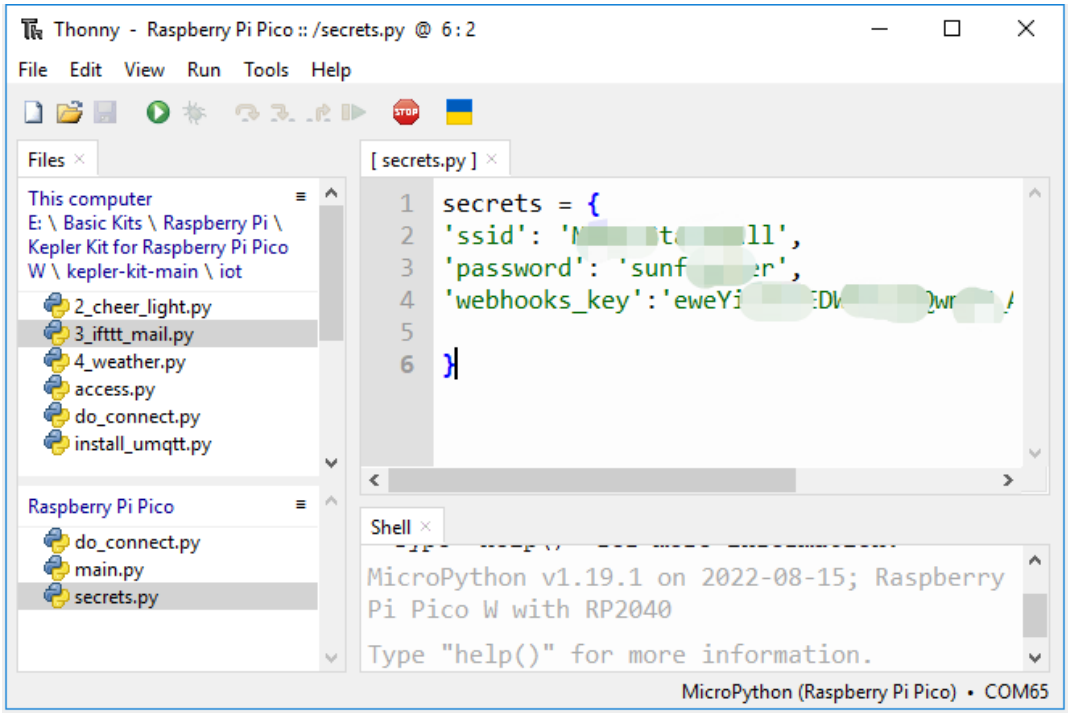


4. Run the Script

1. Now that we have created the IFTTT Applet, but we also need the API Key which can be obtained from the to allow Pico W to access IFTTT.



2. Copy it to the `secrets.py` script in Raspberry Pi Pico W.



Note: If you don't have `do_connect.py` and `secrets.py` scripts in your Pico W, please refer to *1. Access to the Network* to create them.

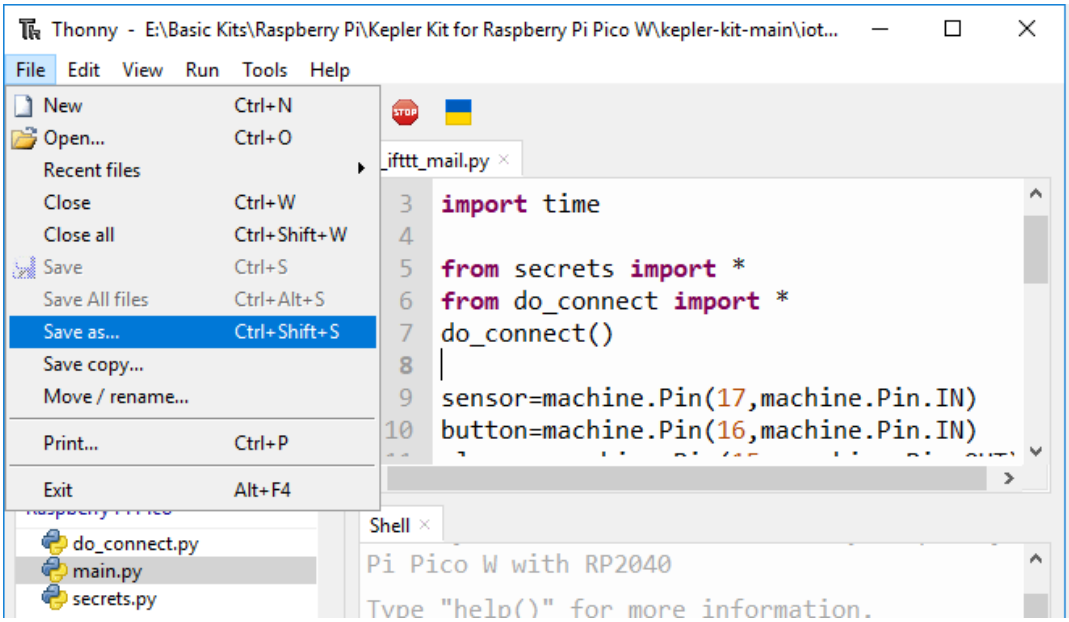
```
secrets = {
```

(continues on next page)

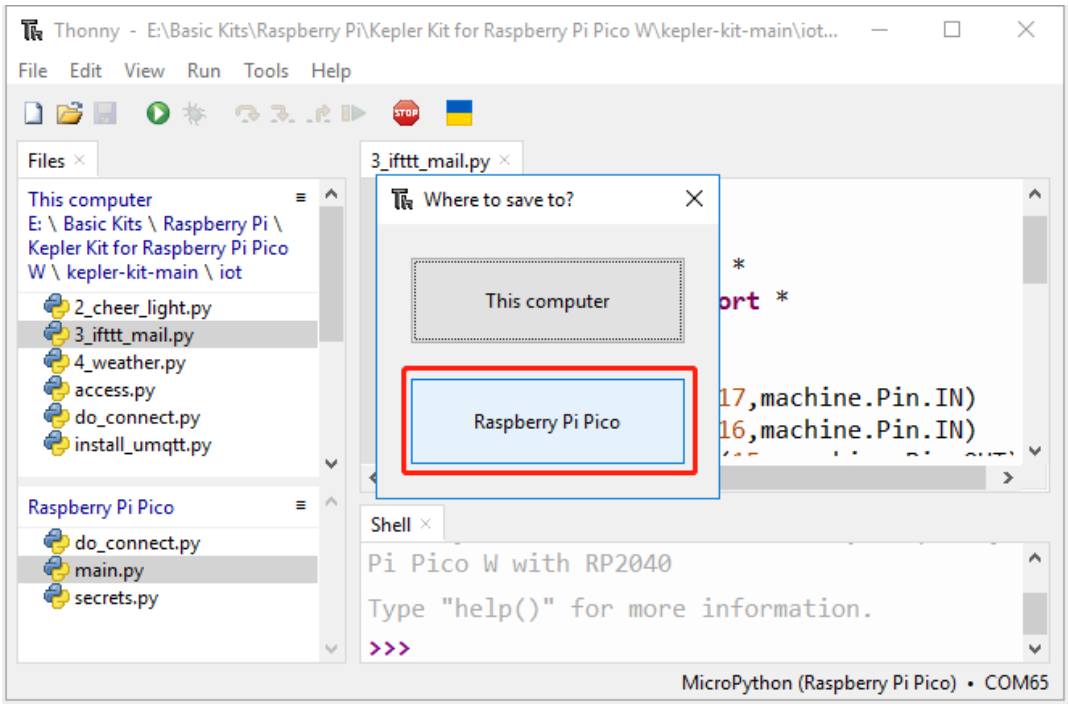
(continued from previous page)

```
'ssid': 'SSID',
'password': 'PASSWORD',
'webhooks_key':'WEBHOOKS_API_KEY'
}
```

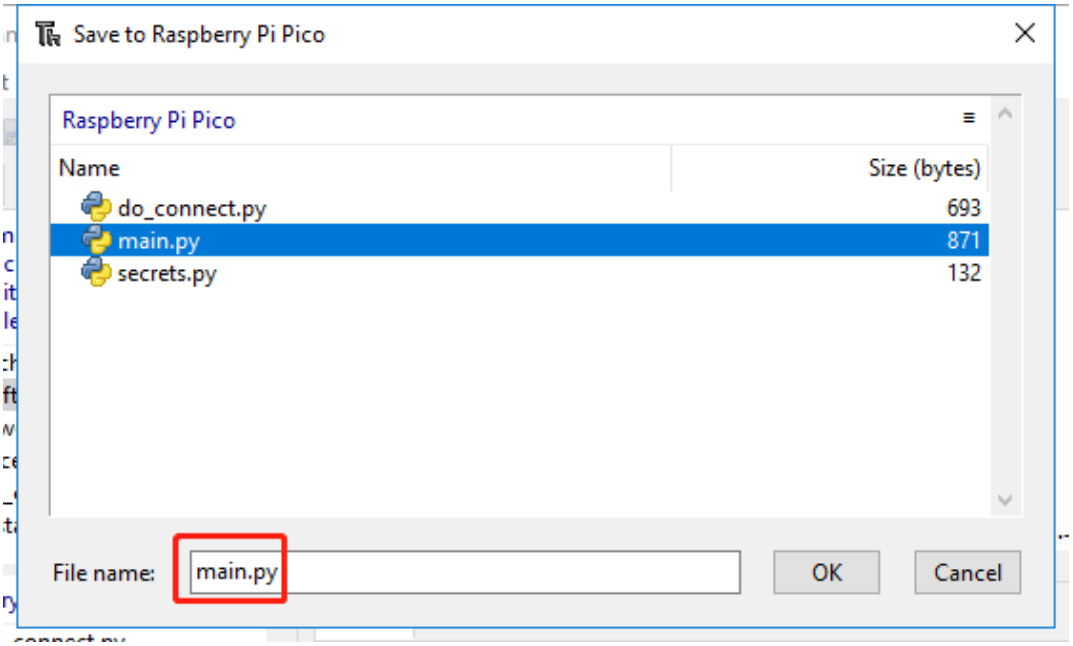
3. Open the `3_ifttt_mail.py` file under the path of `kepler-kit-main/iot`, then click **File** -> **Save as** or press `Ctrl+Shift+S`.



4. Select **Raspberry Pi Pico** in the popup window that appears.



5. Set the file name to `main.py`. A prompt will appear if the same file already exists on your Pico W.



6. You can now unplug the USB cable and use the Li-po Charger Module to power the Raspberry Pi Pico W. You will hear a buzzer tick when the script is running. The buzzer will keep sounding if the PIR module detects a person/creature passing by, and an email alert will be sent to you. The script can be restarted by pressing the button.

How it works?

The Raspberry Pi Pico W needs to be connected to the Internet, as described in *1. Access to the Network*. For this project, just use it.

```
from do_connect import *
do_connect()
```

Reads data from the PIR module and calls the `motion_detected()` function if it detects someone passing by (data from 0 to 1).

```
import machine

sensor=machine.Pin(17,machine.Pin.IN)

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

Next the Pico W sends data to IFTTT. As you can see, the `message` that you send to IFTTT is a url string. IFTTT identifies the sender by `secrets['webhooks_key']`, the triggered event is identified by `event`. So, make sure they are correct.

```
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)
```

(continues on next page)

(continued from previous page)

```
    print(message)
    global warn_flag
    warn_flag=True
    sensor.irq(handler=None)
```

When `motion_detected()` is called, the variable `warn_flag` is set to `True`, causing the buzzer to keep going.

```
while True:
    if warn_flag==True:
        alarm.toggle()
        time.sleep_ms(50)
```

The purpose of the button here is to restart the script.

```
button=machine.Pin(16,machine.Pin.IN)

def reset_device(pin):
    machine.reset()

button.irq(trigger=machine.Pin.IRQ_RISING, handler=reset_device)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

5.4 4. Real-time Weather From @OpenWeatherMap

This project involves making a smart clock that prints the weather in your city along with the time on the LCD.

1. Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

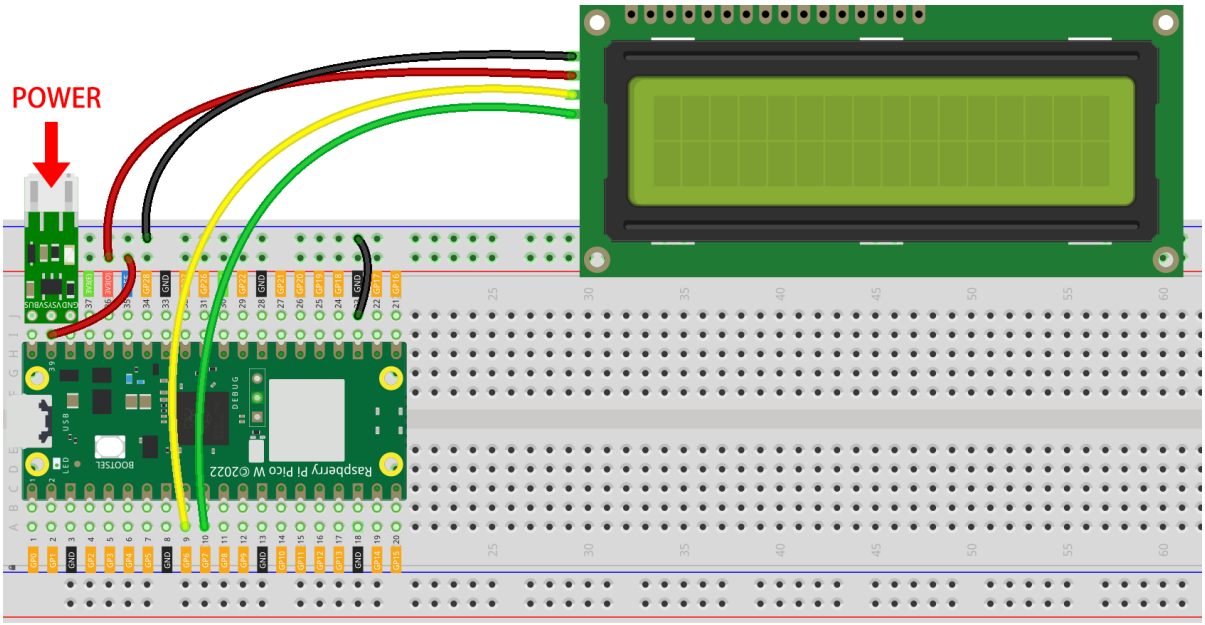
Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*I2C LCD1602*	1	
6	*Li-po Charger Module*	1	
7	18650 Battery	1	
8	Battery Holder	1	

2. Build the Circuit

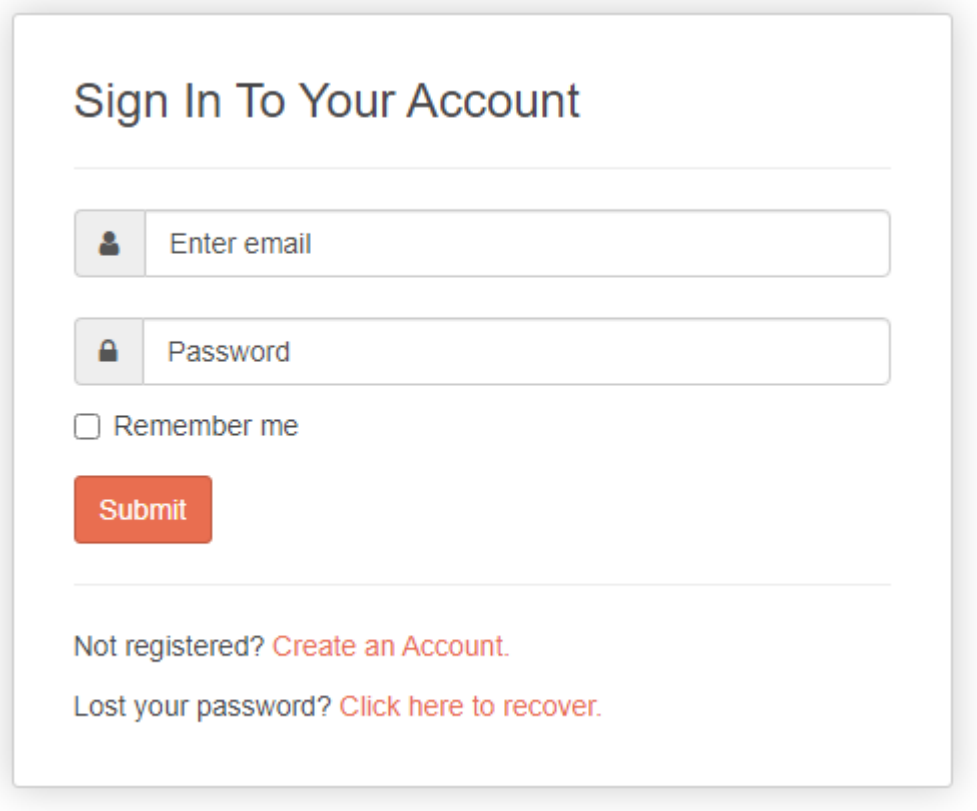
> **Warning:** Make sure your Li-po Charger Module is connected as shown in the diagram. Otherwise, a short circuit will likely damage your battery and circuitry.



3. Get OpenWeather API keys

is an online service, owned by OpenWeather Ltd, that provides global weather data via API, including current weather data, forecasts, nowcasts and historical weather data for any geographical location.

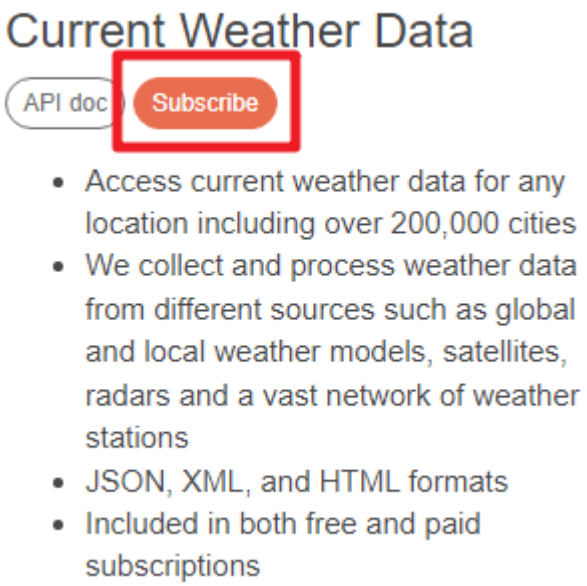
1. Visit to log in/create an account.



2. Click into the API page from the navigation bar.



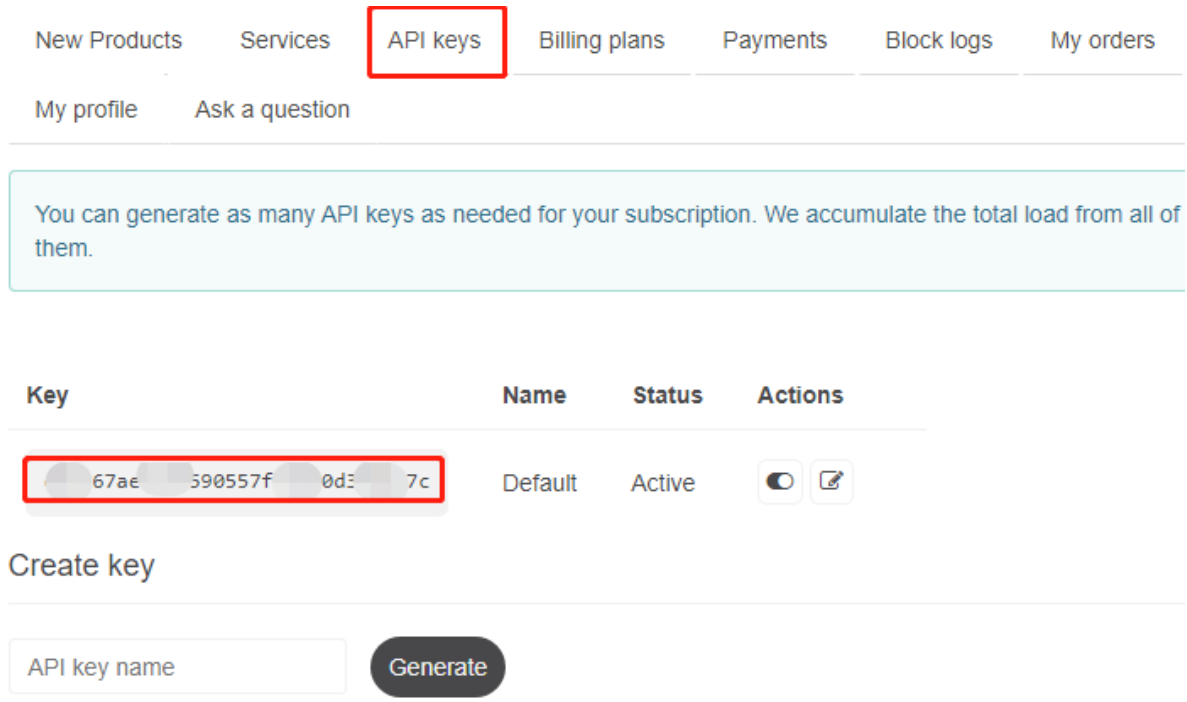
3. Find **Current Weather Data** and click Subscribe.



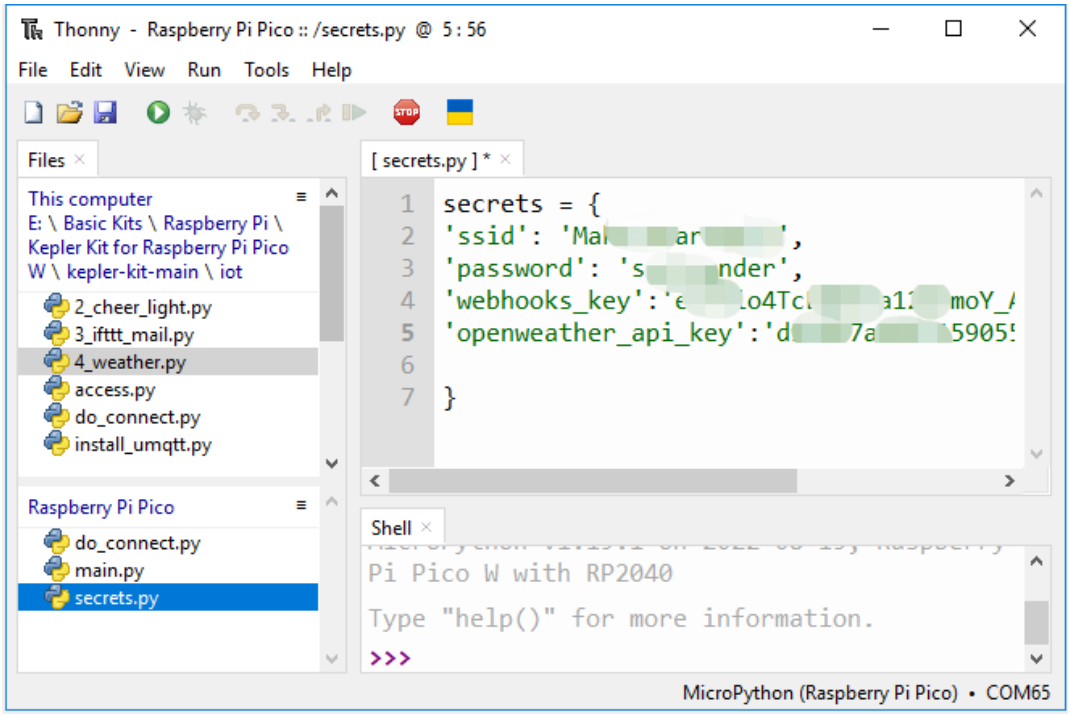
4. Under **Current weather and forecasts collection**, subscribe to the appropriate service. In our project, Free is good enough.



5. Copy the Key from the **API keys** page.



6. Copy it to the `secrets.py` script in Raspberry Pi Pico W.

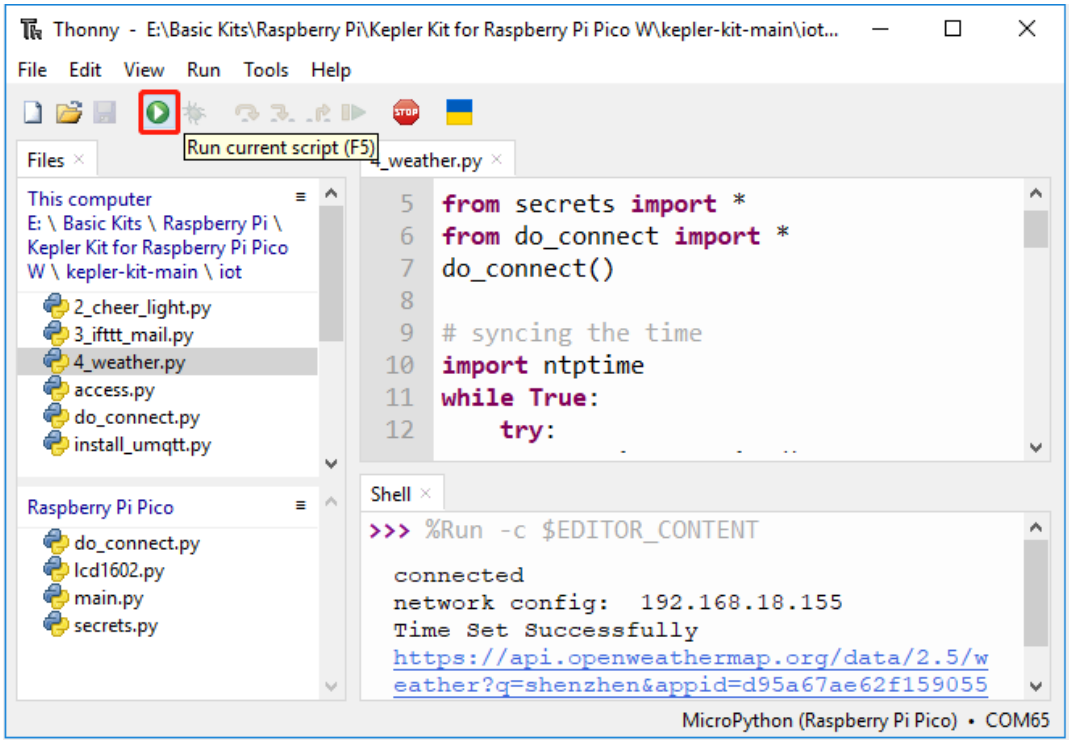


Note: If you don't have `do_connect.py` and `secrets.py` scripts in your Pico W, please refer to *1. Access to the Network* to create them.

```
secrets = {
'ssid': 'SSID',
'password': 'PASSWORD',
'webhooks_key':'WEBHOOKS_API_KEY',
'openweather_api_key':'OPENWEATHERMAP_API_KEY'
}
```

4. Run the Script

1. Open the `4_weather.py` file under the path of `kepler-kit-main/iot`, click the **Run current script** button or press F5 to run it.



2. After the script runs, you will see the time and weather information of your location on the I2C LCD1602.

> **Note:** When the code is running, if the screen is blank, you can turn the potentiometer on the back of the module to increase the contrast.

3. If you want this script to be able to boot up, you can save it to the Raspberry Pi Pico W as `main.py`.

How it works?

The Raspberry Pi Pico W needs to be connected to the Internet, as described in *1. Access to the Network*. For this project, just use it.

```
from do_connect import *
do_connect()
```

After connecting to the Internet, these few lines of code will help synchronize your Pico W to Greenwich Mean Time.

```
import ntptime
while True:
    try:
        ntptime.settime()
        print('Time Set Successfully')
        break
    except OSError:
        print('Time Setting...')
        continue
```

Initialize your LCD, please refer to *3.4 Liquid Crystal Display* for its usage details.

```
from lcd1602 import LCD
lcd=LCD()
lcd.clear()
string = 'Loading...'
lcd.message(string)
```

We need to select the unit for some weather data (e.g. temperature, wind speed) before we get the weather data. In this case, the unit is `metric`.

```
# Open Weather
TEMPERATURE_UNITS = {
   "standard": "K",
   "metric": "°C",
   "imperial": "°F",
}

SPEED_UNITS = {
   "standard": "m/s",
   "metric": "m/s",
   "imperial": "mph",
}

units = "metric"
```

Next, this function gets the weather data from `openweathermap.org`. We will post a URL message to it with your city, API keys, and a set unit. As a result, you will receive a `JSON` file with weather data.

```
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()
```

If you print out this set of raw data, you will be able to see information similar to that shown below.

```
weather data example:
{
    'timezone': 28800,
    'sys': {
        'type': 2,
```

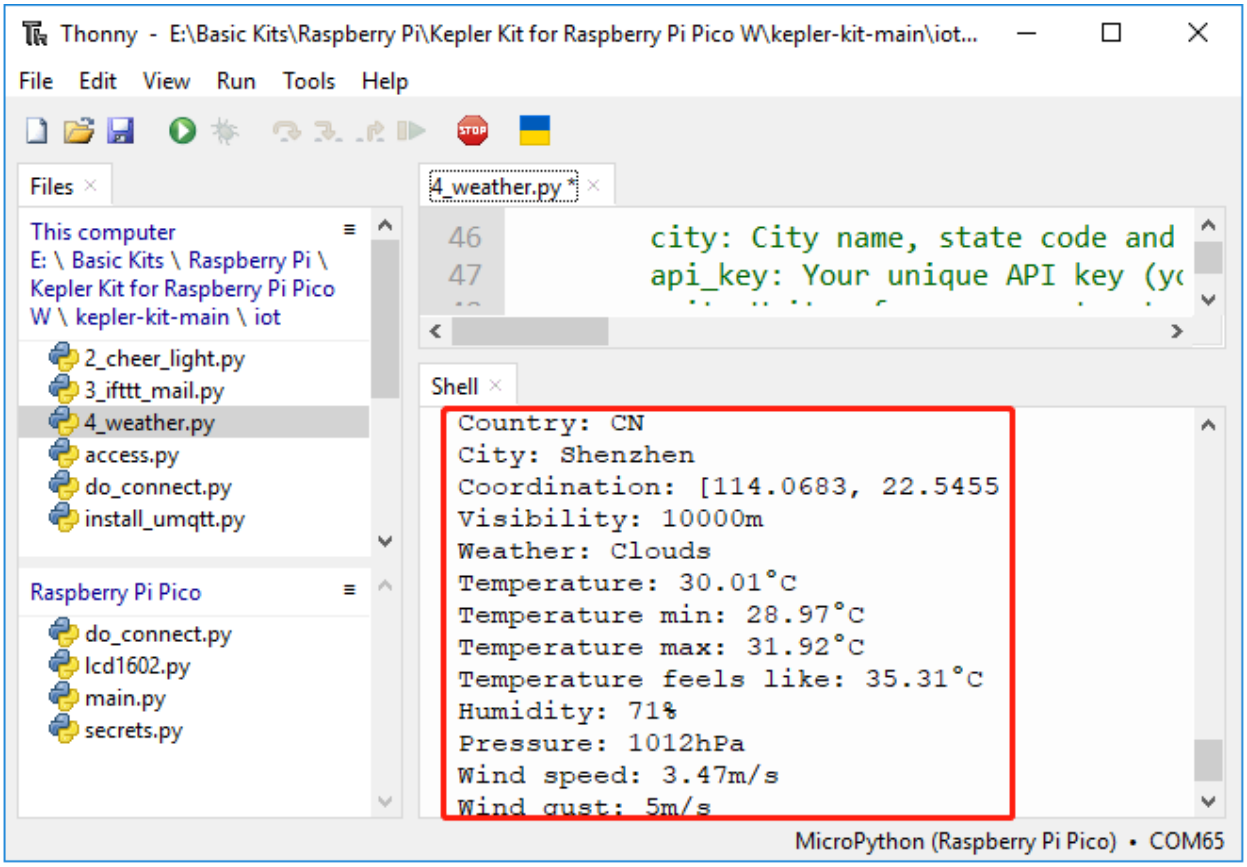
(continues on next page)

(continued from previous page)

```
        'sunrise': 1659650200,
        'country': 'CN',
        'id': 2031340,
        'sunset': 1659697371
    },
    'base': 'stations',
    'main': {
        '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
    }
}
```

We used the `print_weather(weather_data)` function to convert these raw data into easy-to-read data format and print them.

But this function is not called, and you can uncomment this line in `while True` as needed.




```
# shell print
print_weather(weather_data)
```

In the `while True` loop, the `get_weather()` function is called first to retrieve the `weather`, `temperature` and `humidity` information needed for this project.

```
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"]
```

Get the local time. The `time.localtime()` function is called here to return a set of tuples (year, month, mday, hour, minute, second, weekday, yearday). We have taken `hour` and `minute` out of it.

Note that we have already synchronized Pico W to Greenwich Mean Time, so we need to add the time zone of your location.

```
# get time (+24 allows for western hemisphere)
# if negative, add 24
# hours = time.localtime()[3] + int(weather_data["timezone"] / 3600) + 24  #only for␣
↪west hemisphere

hours=time.localtime()[3]+int(weather_data["timezone"] / 3600)
mins=time.localtime()[4]
```

Finally, the weather information and time are just displayed in the LCD1602.

```
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)
```

Your LCD1602 will become a clock that refreshes every 30 seconds when the main loop runs every 30 seconds.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

5.5 5. Cloud Calling System with @MQTT

Message Queuing Telemetry Transport (MQTT) is a simple messaging protocol. It is also the most common messaging protocol for the Internet of Things (IoT).

MQTT protocols define the way IoT devices transfer data. They are event-driven and interconnected using the Pub/Sub model. The sender (Publisher) and the receiver (Subscriber) communicate via Topics. A device publishes a message on a specific topic, and all devices subscribed to that topic receive the message.

In this section, a service bell system will be made using Pico W, HiveMQ (a free public MQTT broker service), and four buttons. The four buttons mean four tables in the restaurant, and you will be able to see which table's guests need service on HiveMQ when the customer presses the button.

1. Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

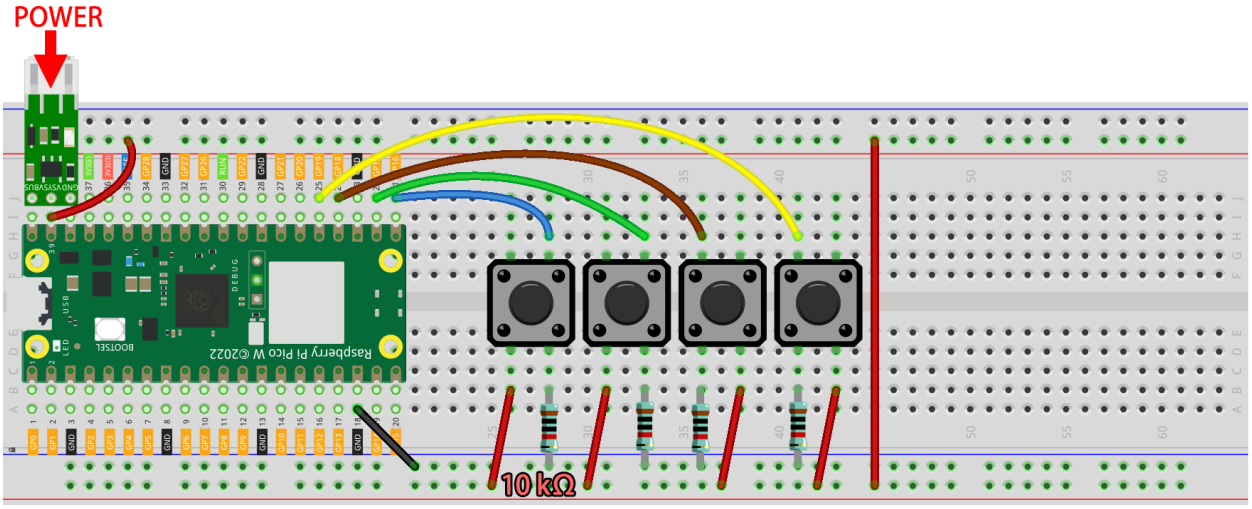
Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	4(10K)	
6	*Button*	4	
7	*Li-po Charger Module*	1	
8	18650 Battery	1	
9	Battery Holder	1	

2. Build the Circuit

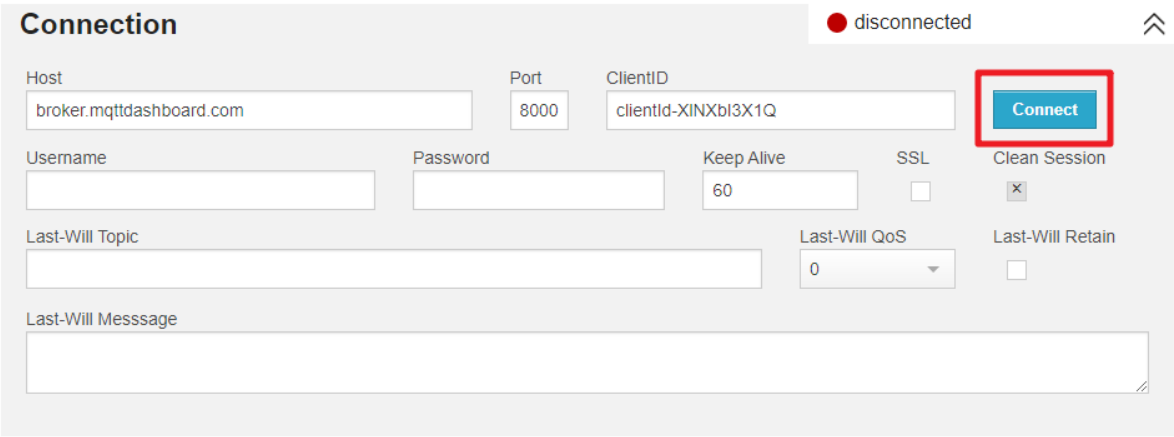
> **Warning:** Make sure your Li-po Charger Module is connected as shown in the diagram. Otherwise, a short circuit will likely damage your battery and circuitry.



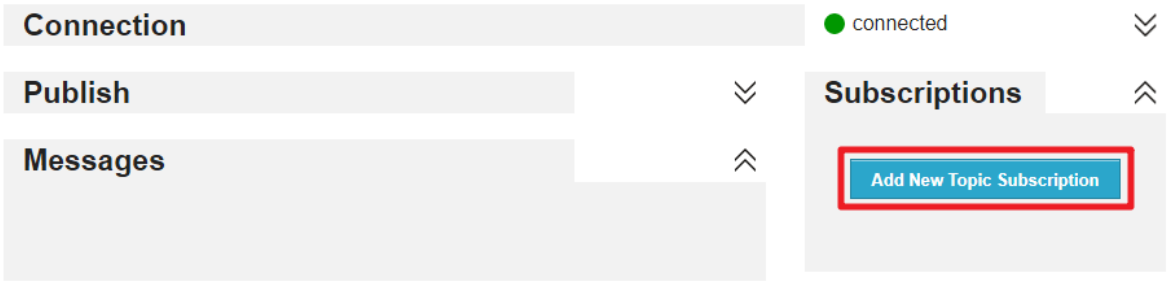
3. Visit HiveMQ

HiveMQ is an MQTT broker and client-based messaging platform that enables fast, efficient and reliable data transfer to IoT devices.

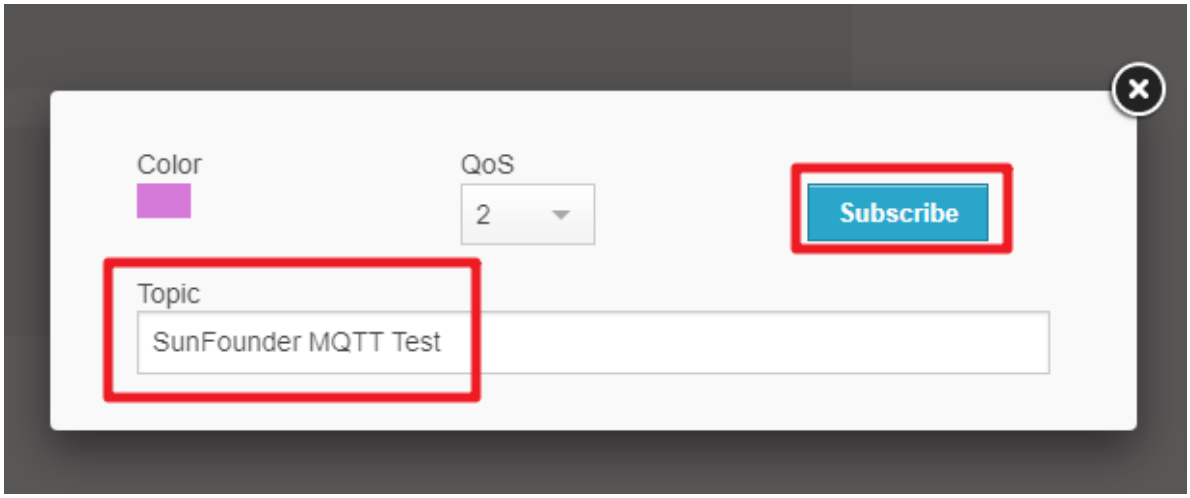
1. Open in your browser.
2. Connects the client to the default public proxy.



3. Click on **Add New Topic Subscription**.



4. Fill in the topics you want to follow and click **Subscribe**. The topics set here should be more personal to avoid getting messages from other users, and pay attention to case sensitive.



4. Install the MQTT Module

Before we can start the project, we need to install the MQTT module for Pico W.

1. Connect to the network by running `do_connect()` in the Shell, which we wrote earlier.

Note:

- Type the following commands into the Shell and press `Enter` to run them.

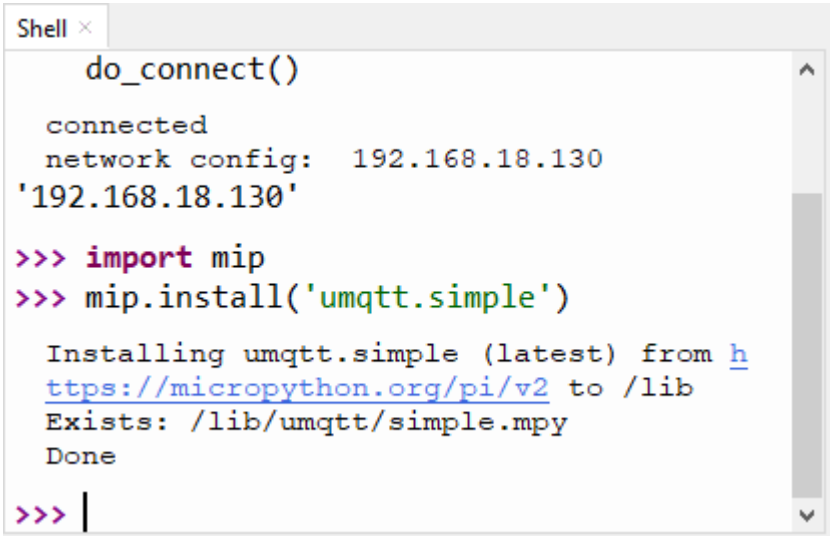
- If you don't have `do_connect.py` and `secrets.py` scripts in your Pico W, please refer to *1. Access to the Network* to create them.

```
from do_connect import *
do_connect()
```

2. After a successful network connection, import the `mip` module in the shell and use `mip` to install the `umqtt.simple` module, which is a simplified MQTT client for MicroPython.

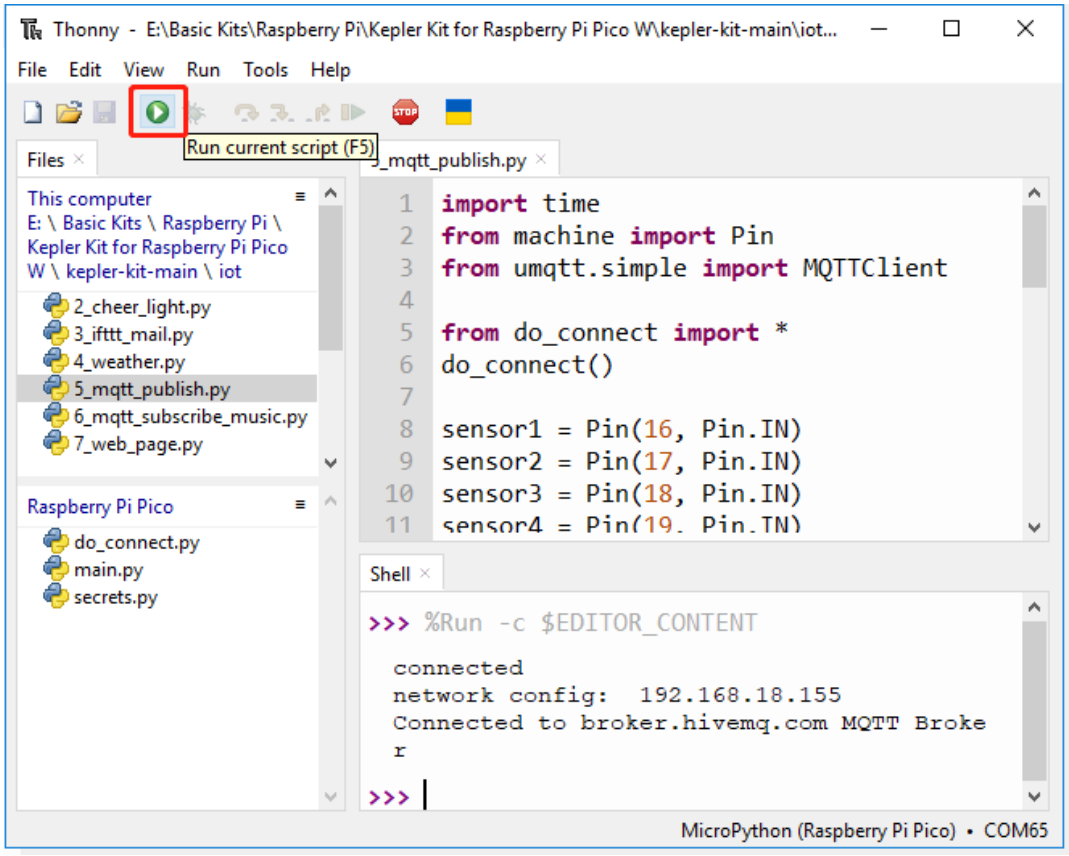
```
import mip
mip.install('umqtt.simple')
```

3. You will see that the `umqtt` module is installed under the `/lib/` path of Pico W after completion.

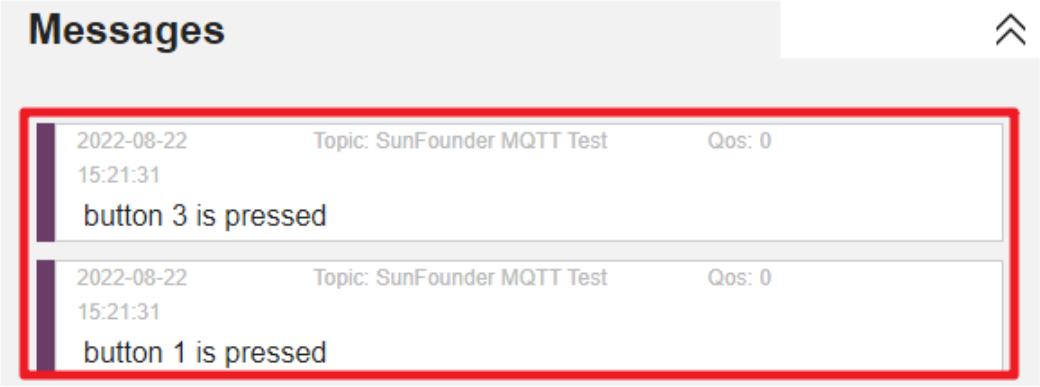


5. Run the Script

1. Open the `5_mqtt_publish.py` file under the path of `kepler-kit-main/iot`.
2. Click the **Run current script** button or press F5 to run it.



3. Go back to again and when you press one of the buttons on the breadboard, you will be able to see the Messages prompt on HiveMQ.



4. If you want this script to be able to boot up, you can save it to the Raspberry Pi Pico W as `main.py`.

How it works?

The Raspberry Pi Pico W needs to be connected to the Internet, as described in *1. Access to the Network*. For this project, just use it.

```
from do_connect import *
do_connect()
```

Initialize 4 button pins.

```
sensor1 = Pin(16, Pin.IN)
sensor2 = Pin(17, Pin.IN)
sensor3 = Pin(18, Pin.IN)
sensor4 = Pin(19, Pin.IN)
```

Create two variables to store the `URL` and `client ID` of the MQTT broker we will use to connect to it. Since we are using a public broker, our `client ID` will not be used, even if one is required.

```
mqtt_server = 'broker.hivemq.com'
client_id = 'Jimmy'
```

Connect to the MQTT agent and hold for one hour. If it fails, reset the Pico W.

```
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()
```

Create a variable `topic`, which is the topic that the subscriber needs to follow. It should be the same as the topic filled in **step 4** of **2. Visit HiveMQ** above. Incidentally, `b` here converts string to byte, because MQTT is a binary based protocol were the control elements are binary bytes and not text strings.

```
topic = b'SunFounder MQTT Test'
```

Set interrupts for each button. When a button is pressed, a message is posted under `topic`.

```
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 Client API

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

5.6 6. Cloud Player with @MQTT

It is recommended that you do the *5. Cloud Calling System with @MQTT* project first to complete the installation of some modules and to complete the configuration of the HiveMQ platform.

In this project, Pico W will act as a subscriber and receive the song name under the topic. If the song name is already in the code, Pico W will make the buzzer play the song.

1. Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

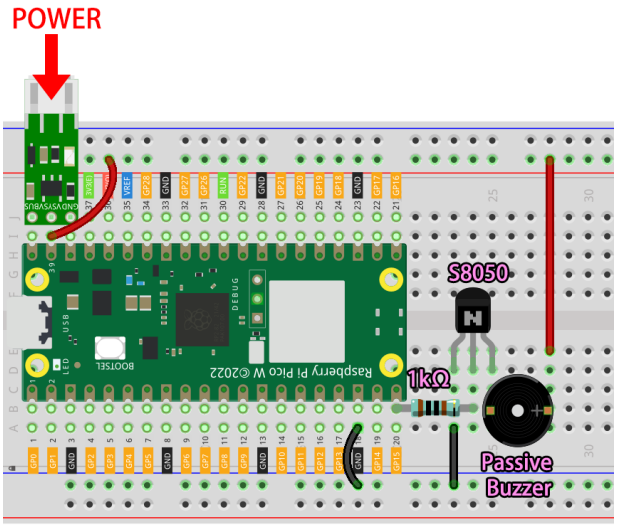
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Transistor*	1(S8050)	
6	*Resistor*	1(1K)	
7	Passive *Buzzer*	1	
8	*Li-po Charger Module*	1	
9	18650 Battery	1	
10	Battery Holder	1	

2. Build the Circuit

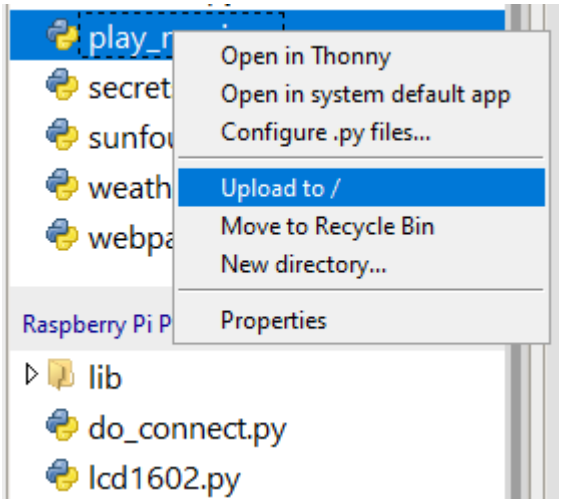
Two buzzers are included in the kit, we use a passive buzzer (one with an exposed PCB on the back). The buzzer needs a transistor to work, here we use S8050.

> **Warning:** Make sure your Li-po Charger Module is connected as shown in the diagram. Otherwise, a short circuit will likely damage your battery and circuitry.

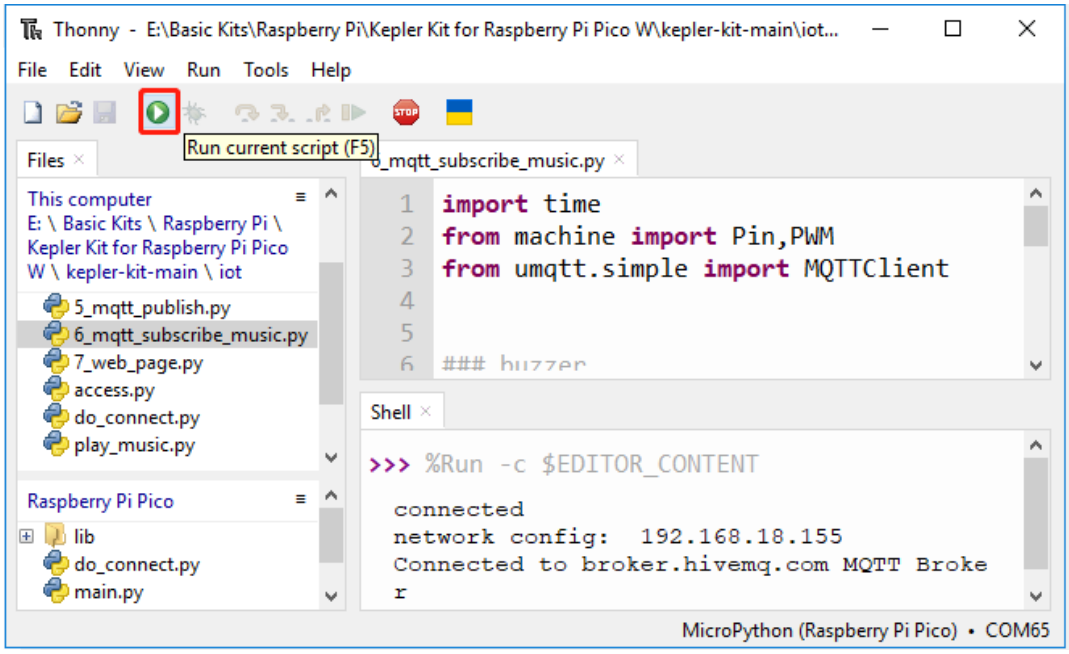


3. Run the Code

1. Upload the `play_music.py` file under the path of `kepler-kit-main/iot` to the Raspberry Pi Pico W.



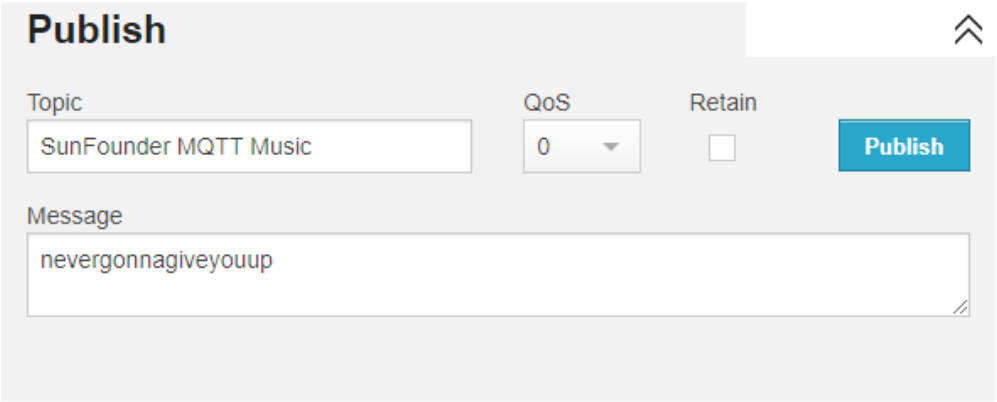
2. Open the `6_mqtt_subscribe_music.py` file under the path of `kepler-kit-main/iot` and click the **Run current script** button or press F5 to run it.



Note: Before running the code, you need to make sure you have `do_connect.py` and `secrets.py` scripts in your Pico W, if not please refer to *1. Access to the Network* to create them.

3. Open in your browser, fill in the Topic as `SunFounder MQTT Music`, fill in the song name as **Message**. After clicking **Publish** button, the buzzer connected to the Pico W will play the corresponding song.

Note: Included in play_music.py are `nokia`, `starwars`, `nevergonnagiveyouup`, `gameofthrone`, `songofstorms`, `zeldatheme`, `harrypotter`.



4. If you want this script to be able to boot up, you can save it to the Raspberry Pi Pico W as `main.py`.

How it works?

In order to make it easier to understand, we separated the MQTT code from the rest. As a result, you get the following code, which implements the most basic functionality of MQTT subscriptions in three places.

```
import time
from umqtt.simple import MQTTClient
```

(continues on next page)

(continued from previous page)

```
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)
```

When connecting to the MQTT broker, we call the `client.set_callback(callback)` function, which serves as a callback for the received subscription messages.

```
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()
```

Next is the callback function, which prints out the message from the topic that was fetched. MQTT is a binary based protocol were the control elements are binary bytes and not text strings, so these messages need to be decoded using `message.decode('utf-8')`.

```
def callback(topic, message):
    print("New message on topic {}".format(topic.decode('utf-8')))
    message = message.decode('utf-8')
    print(message)
```

Use a `While True` loop to get messages under this topic at regular intervals.

```
while True:
    client.subscribe(topic)
    time.sleep(1)
```

Next, music will be played. This function is placed in the `play_music.py` script, which consists of three main parts.

- `Tone`: Simulates a specific tone based on the fundamental , which is used to play it.

```
NOTE_B0 =  31
NOTE_C1 =  33
...
NOTE_DS8 = 4978
REST =      0
```

- `Score`: Edit the music into a format that the program can use. These scores are from Robson Couto's free sharing, you can also add your favorite music in the following format.

```
# notes of the moledy followed by the duration.
# a 4 means a quarter note, 8 an eighteenth , 16 sixteenth, so on
# !!negative numbers are used to represent dotted notes,
# so -4 means a dotted quarter note, that is, a quarter plus an␣
↪eighteenth!!
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_A4, 2],
    "starwars":[,,,],
    "nevergonnagiveyouup":[,,,],
    "gameofthrone":[,,,],
    "songofstorms":[,,,],
    "zeldatheme":[,,,],
    "harrypotter":[,,,],
}
```

- `Play`: This part is basically the same as *3.2 Custom Tone*, but slightly optimized to fit the above score.

```
import time
import machine

# change this to make the song slower or faster
tempo = 220

# this calculates the duration of a whole note in ms
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)
```

(continues on next page)

(continued from previous page)

```
    pin.duty_u16(0)

def noTone(pin):
    tone(pin,0,100)

def play(pin,melody):

    # iterate over the notes of the melody.
    # Remember, the array is twice the number of notes (notes + durations)
    for thisNote in range(0,len(melody),2):
        # calculates the duration of each note
        divider = melody[thisNote+1]
        if divider > 0:
            noteDuration = wholenote/divider
        elif divider < 0:
            noteDuration = wholenote/-(divider)
            noteDuration *= 1.5

        # we only play the note for 90% of the duration, leaving 10% as a pause
        tone(pin,melody[thisNote],int(noteDuration*0.9))

        # Wait for the specief duration before playing the next note.
        time.sleep_ms(int(noteDuration))

        # stop the waveform generation before the next note.
        noTone(pin)
```

Go back to the main function and let MQTT trigger music playback. In the callback function, determine if the message sent is the name of a song that has been included. If it is, assign the song name to the variable `melody` and set `play_flag` to `True`.

```
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
```

In the main loop, if `play_flag` is `True`, play `melody`.

```
while True:
    client.subscribe(topic)
    time.sleep(1)
    if play_flag is True:
        play(buzzer,melody)
        play_flag = False
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

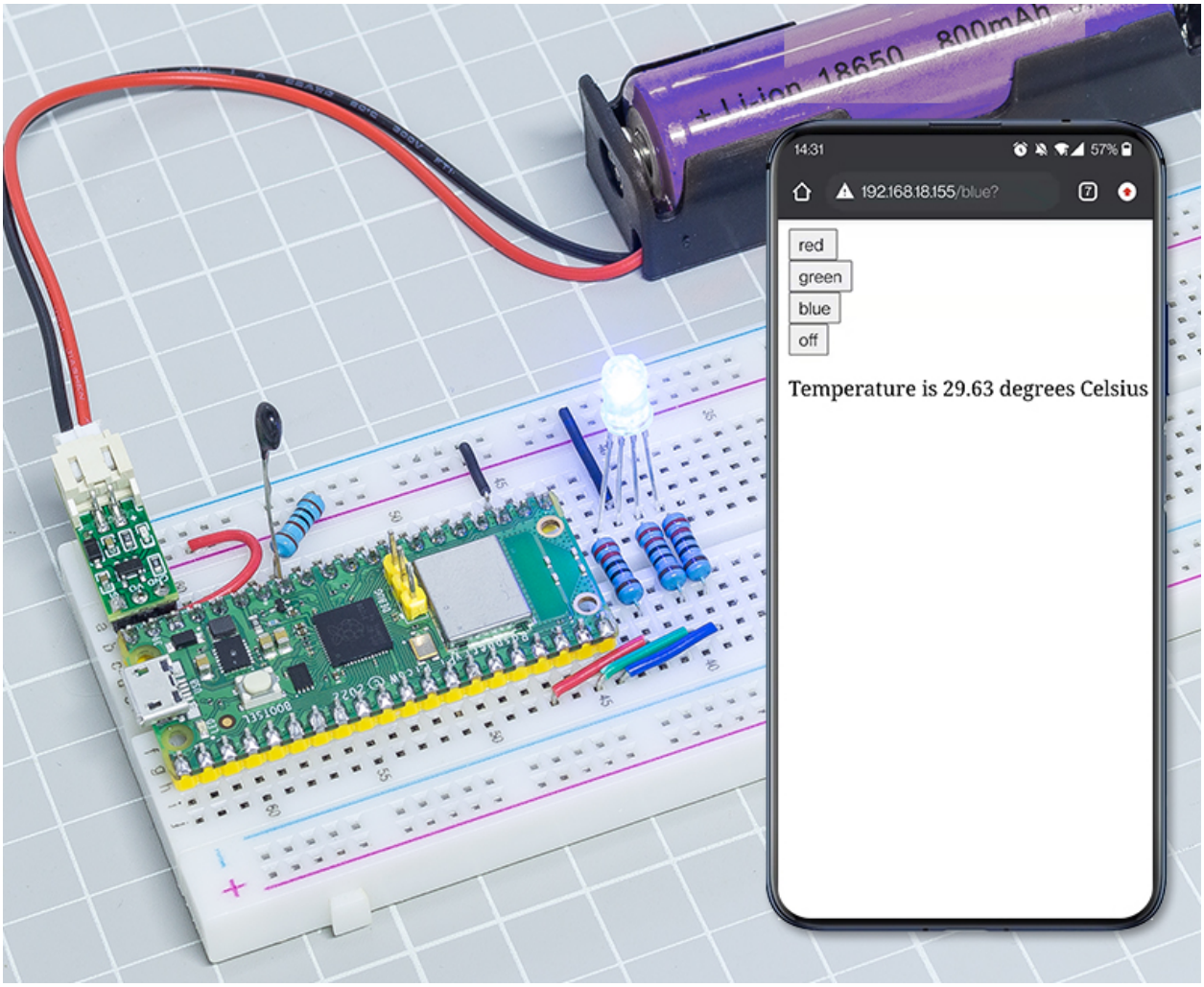
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

5.7 7. Setup a Web Server

In this article, you will learn how to make Pico W a web server that allows you to operate the circuit and get readings from the sensor from a browser.



1. Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

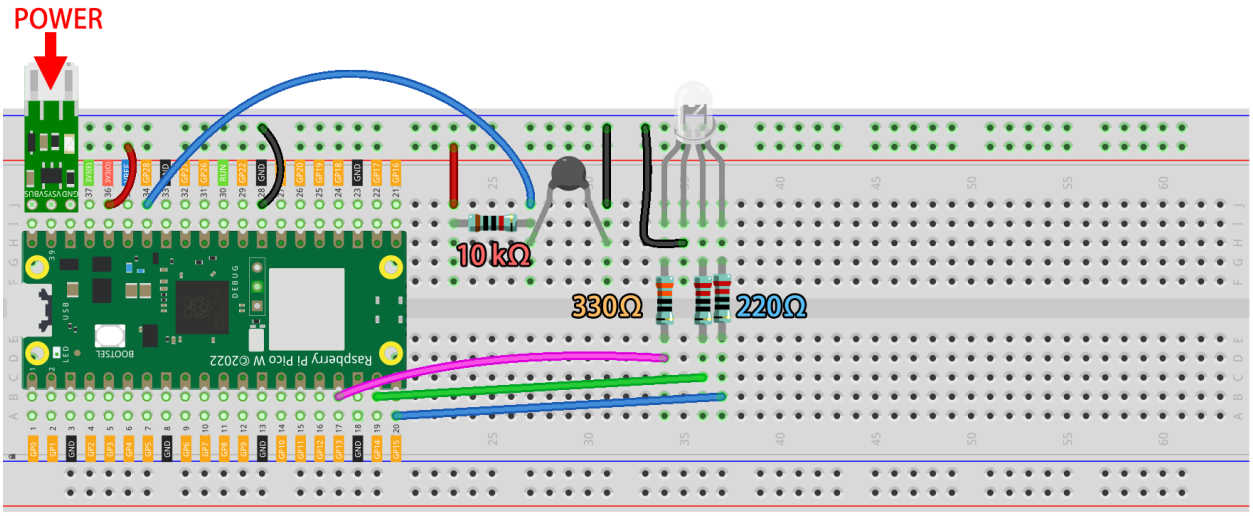
Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	4(1-330, 2-220, 1-10K)	
6	*RGB LED*	1	
7	*Thermistor*	1	
8	*Li-po Charger Module*	1	
9	18650 Battery	1	
10	Battery Holder	1	

2. Build the Circuit

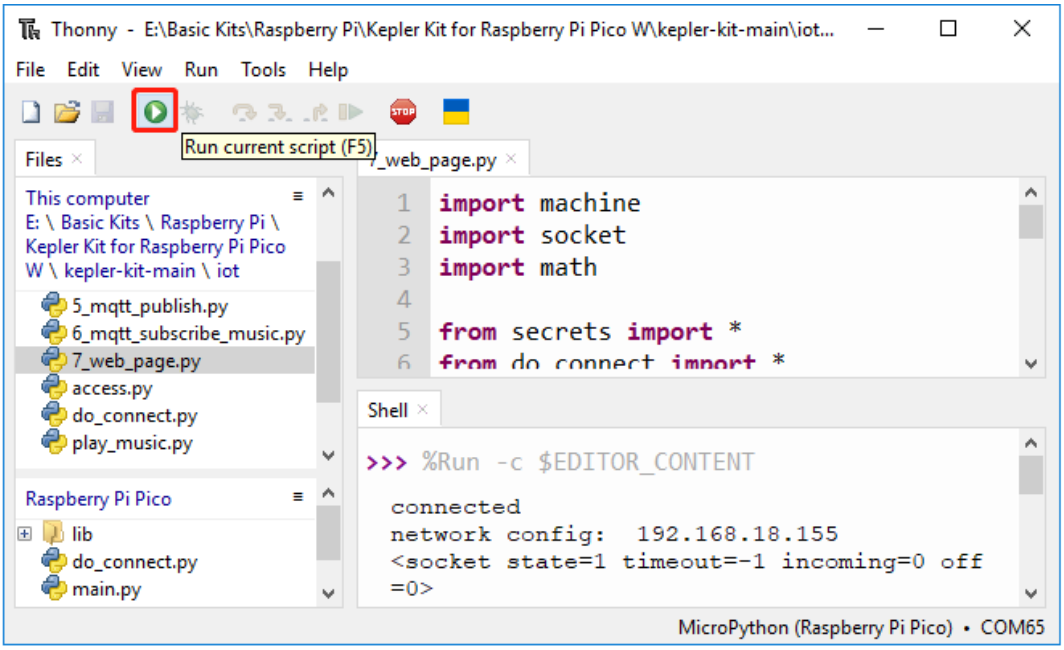
> **Warning:** Make sure your Li-po Charger Module is connected as shown in the diagram. Otherwise, a short circuit will likely damage your battery and circuitry.



3. Run the Code

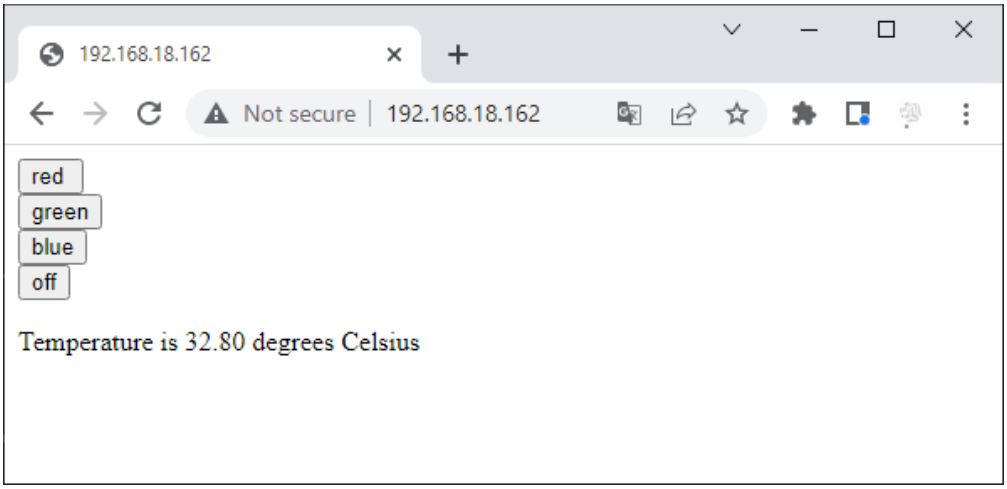
1. Open the `7_web_page.py` file under the path of `kepler-kit-main/iot`.

2. Click the **Run current script** button or press F5 to run it. After successful connection, you will see the IP of Pico W.



Note: Before running the code, you need to make sure you have `do_connect.py` and `secrets.py` scripts in your Pico W, if not please refer to *1. Access to the Network* to create them.

3. Enter the IP address of the Pico W into your browser to access the web page built for this project. Click any button to change the color of the RGB LEDs and refresh the temperature and humidity.



4. If you want this script to be able to boot up, you can save it to the Raspberry Pi Pico W as `main.py`.

How it works?

The web page you are visiting is actually hosted on some server, and the socket on the server will send the web page to us when we visit it. A socket is the way a server can listen for a client that wants to connect to it.

In this project, Pico W is your server, and your computer is accessing the web page hosted on Pico W through a browser.

First we create a socket, which requires an IP address and a . The network connection and the way to get the IP are

described in *1. Access to the Network*. And for port, we use 80. After setting up the socket, return it and use it for the next step.

socket library - Python Docs

```
import socket

def open_socket(ip):
    # Open a socket
    address = (ip, 80)
    connection = socket.socket()
    connection.bind(address)
    connection.listen(1)
    print(connection)
    return(connection)
```

Then, set up your web service where the socket you set up earlier will be used. The following code will allow your Pico W to receive access requests from your browser.

```
def serve(connection):
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        client.close()
```

Next, you need an html page to send to the visitor. This example stores a simple html page in the form of characters in the variable `html`.

Note: If you want to be able to write your own html, you can get help at .

```
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>Temperature is {value} degrees Celsius</p>
            </body>
            </html>
            """
    return html
```

Send html page to visitor.

```
def serve(connection):
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        html=webpage(0)
        client.send(html)
        client.close()
```

The page can be accessed through your browser if you combine the above parts. If you want to see the effect, run the code below with thonny.

```
import machine
import socket

from secrets import *
from do_connect import *

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>Temperature is {value} degrees Celsius</p>
            </body>
            </html>
            """
    return html

def open_socket(ip):
    # Open a socket
    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)
```

(continues on next page)

(continued from previous page)

```
        html=webpage(0)
        client.send(html)
        client.close()

try:
    ip=do_connect()
    if ip is not None:
        connection=open_socket(ip)
        serve(connection)
except KeyboardInterrupt:
    machine.reset()
```

When you run the code above, you will see that it displays only a web page, does not allow you to control RGB LEDs, nor does it show sensor readings. The web service needs to be refined further.

The first thing we need to know is what information the server receives when the browser accesses the web page. Therefore, change `serve()` slightly to print `request`.

```
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()
```

Re-run the script and the Shell will print the following message when we press a key on the web page.

```
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'
```

They are too long to read!!!

But all we really need is the small piece of information in front of `/red?`, `/blue?`. It tells us which button was pressed. So we refined `serve()` a bit to extract the keystroke information.

```
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)
```

(continues on next page)

(continued from previous page)

```
        html=webpage(0)
        client.send(html)
        client.close()
```

Re-run the program and the Shell will print the following message when we press a key on the web page.

```
/red?
/favicon.ico
/blue?
/favicon.ico
/off?
/favicon.ico
```

Then, we just need to change the color of the RGB LED according to the value of `request`.

```
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()
            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()
```

The last thing is to display the thermistor value on the web page (see *2.13 Thermometer* for details on using the thermistor). This part is actually done by modifying the text on the html. We set the parameters in the `webpage(value)` function and simply change the incoming parameters to change the number displayed on the web page.

```
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()
            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()
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

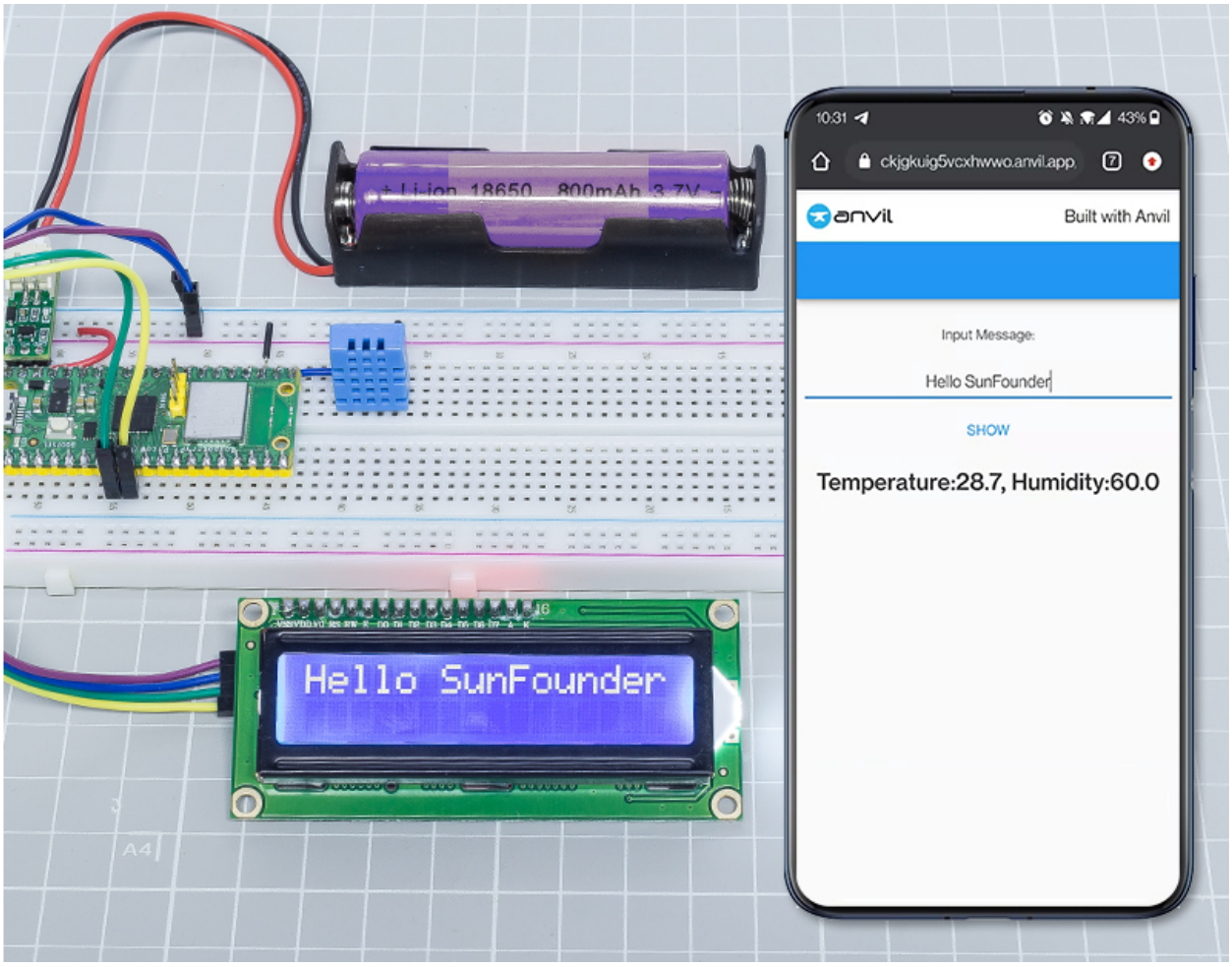
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

5.8 8. Building Web APP with @Anvil

We will write a project to communicate bi-directionally between the Raspberry Pi Pico W and Anvil's servers. The temperature and humidity transmitted from the Pico W will be displayed in real time in Anvil. Additionally, you can input messages in Anvil to be displayed on the Pico W's I2C LCD1602.



1. Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

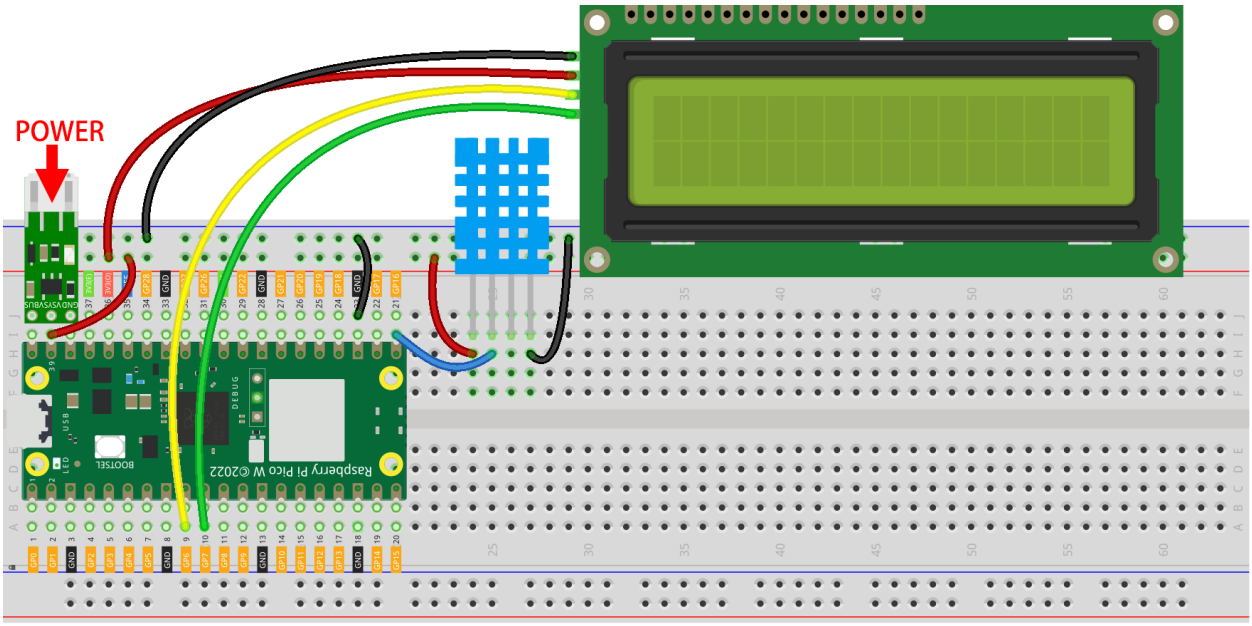
Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*I2C LCD1602*	1	
6	*DHT11 Humiture Sensor*	1	
7	*Li-po Charger Module*	1	
8	18650 Battery	1	
9	Battery Holder	1	

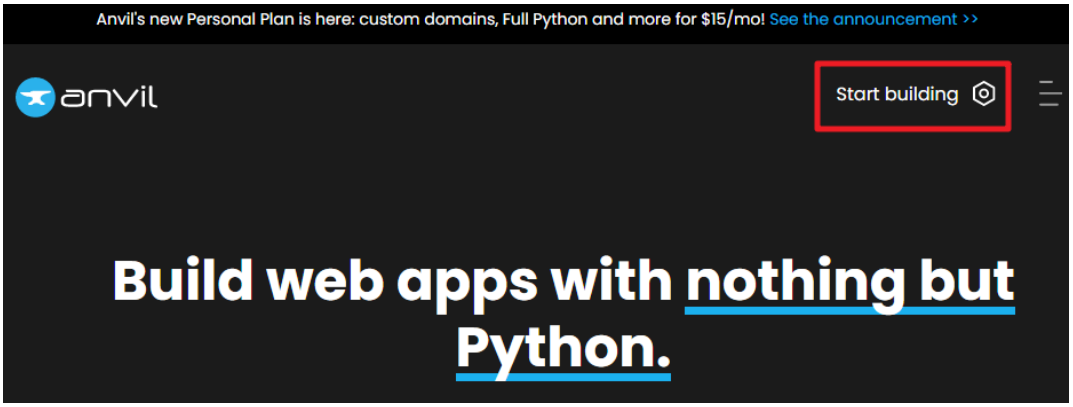
2. Build the Circuit

> **Warning:** Make sure your Li-po Charger Module is connected as shown in the diagram. Otherwise, a short circuit will likely damage your battery and circuitry.

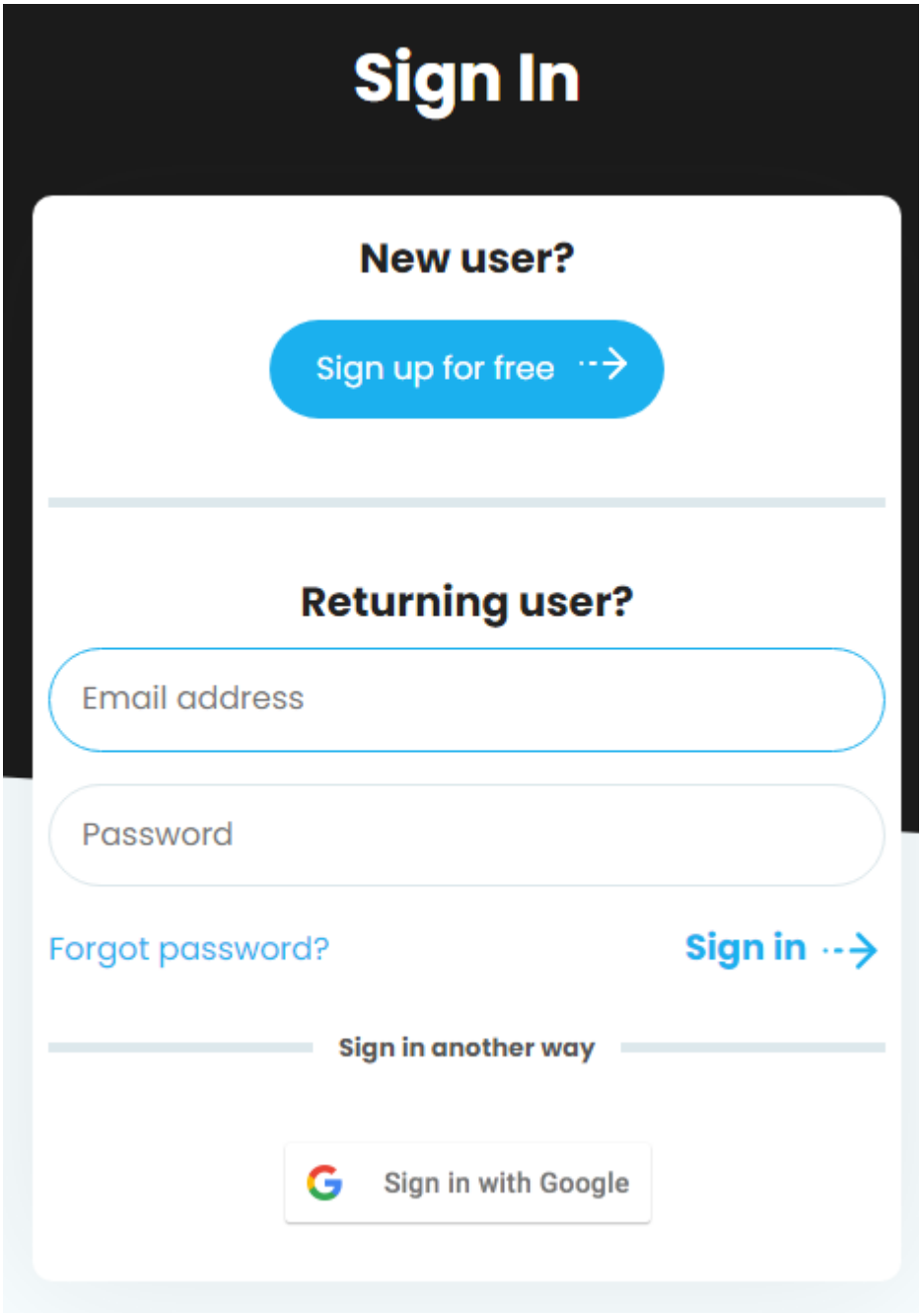


3. Create an Anvil APP

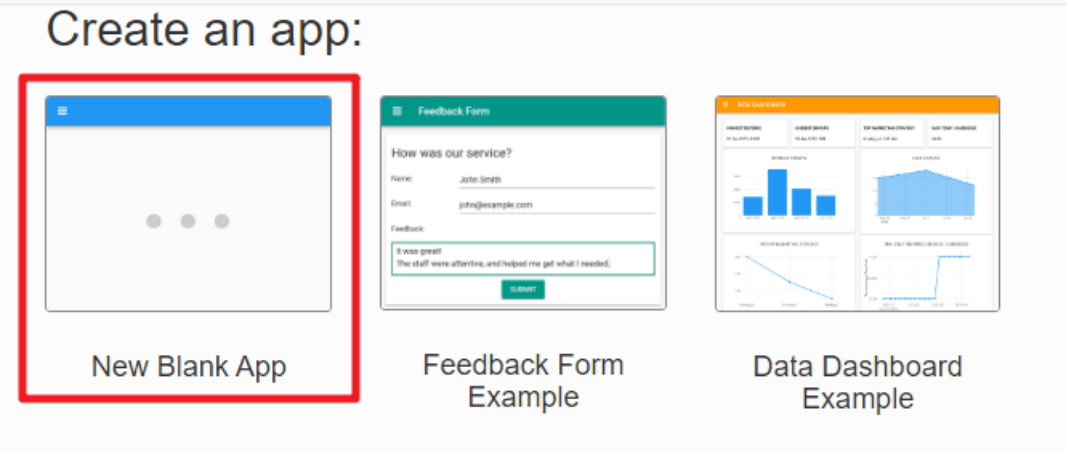
1. Go to the and click on **Start building**.



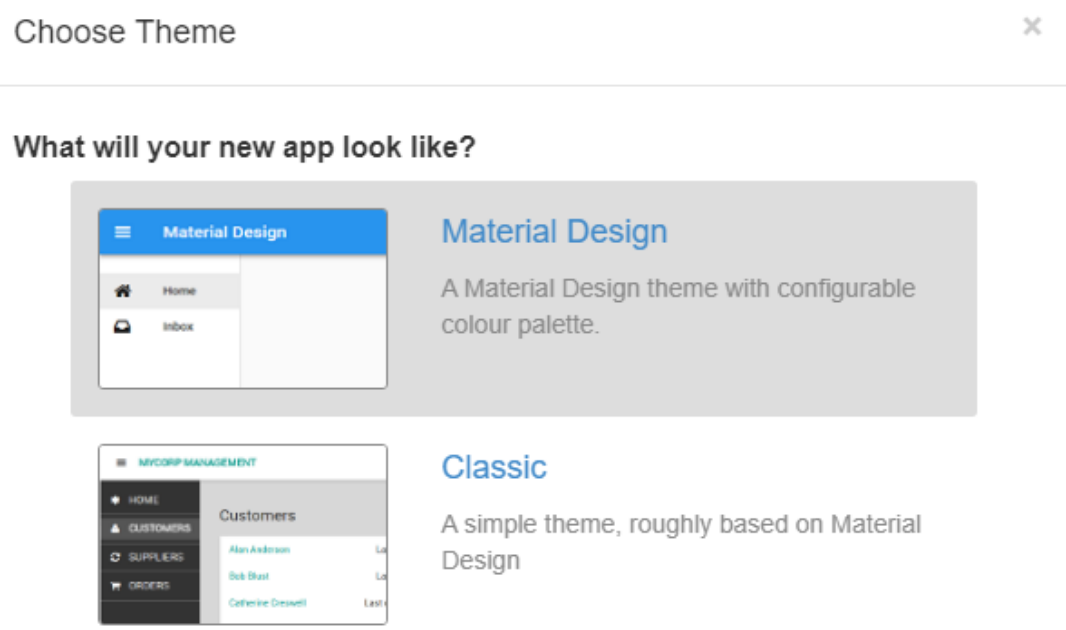
2. Sign in or Sign up.



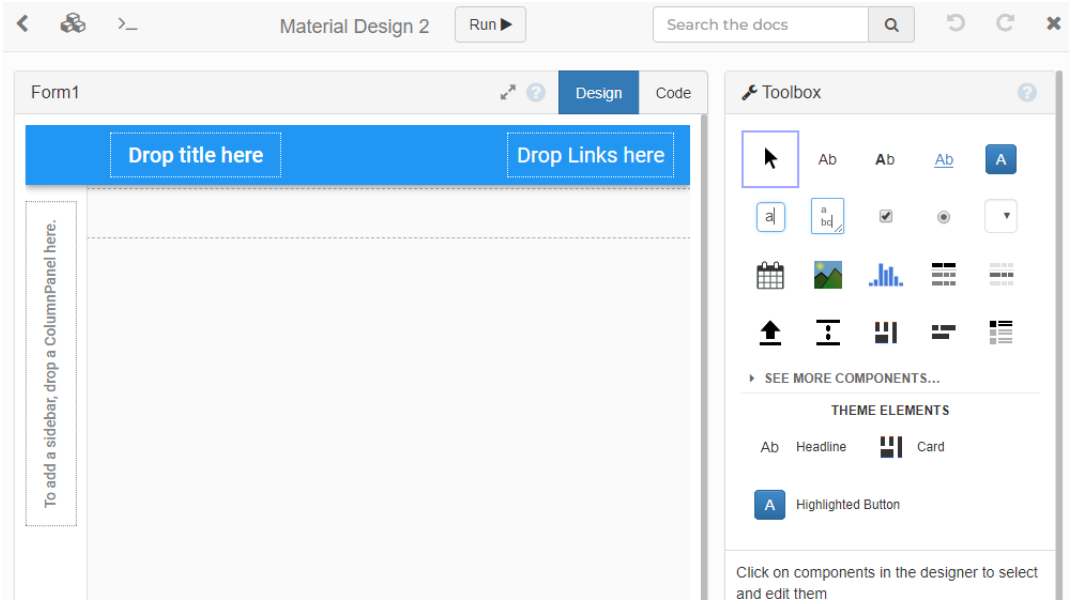
3. Create a **New Blank App**.



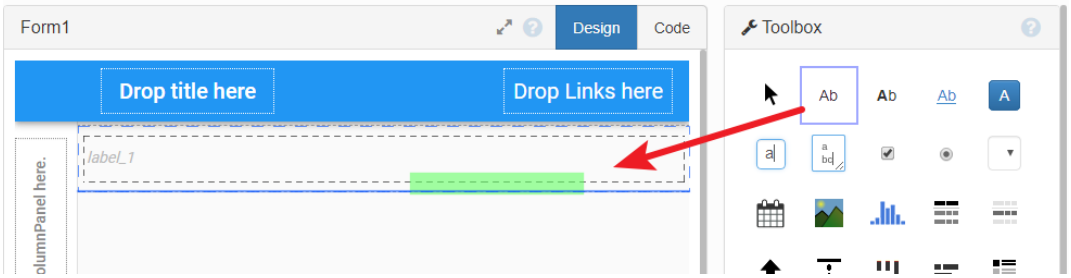
4. Select **Material Design Theme**.



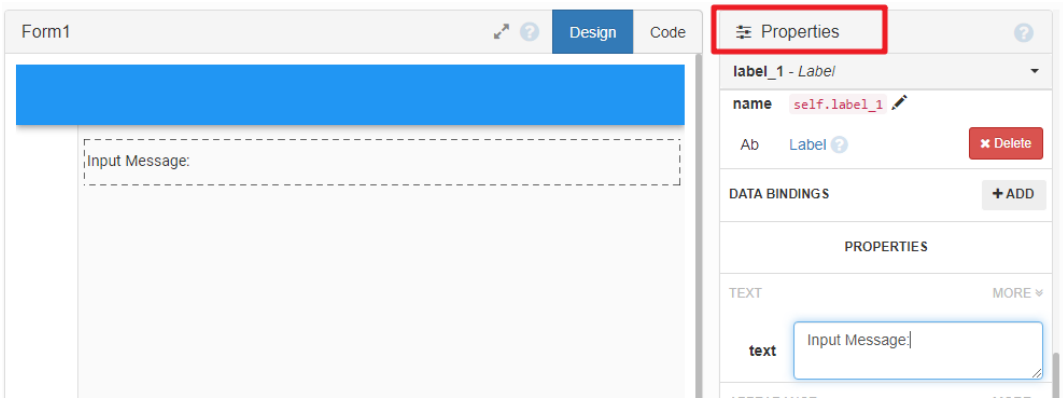
5. Now, you are on the app editing page.



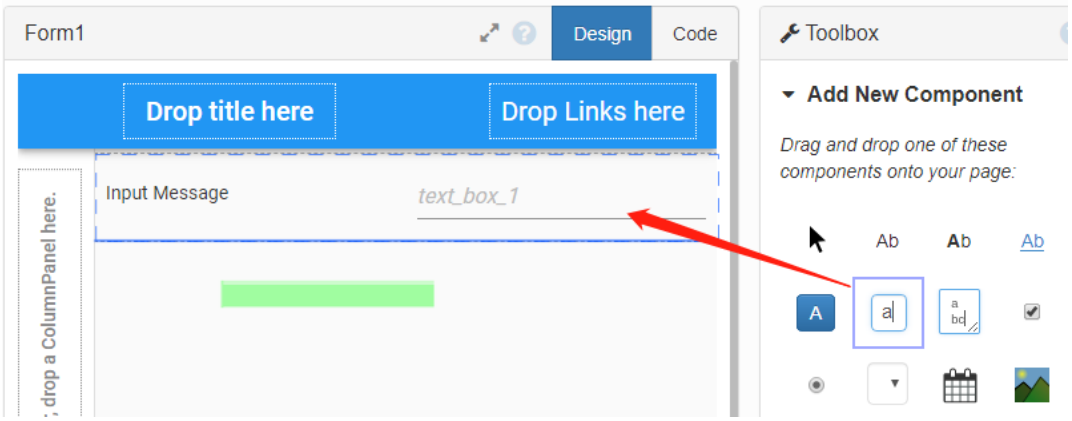
6. From the Toolbox, drag a **Label** tool and place it on **Drop title here**.



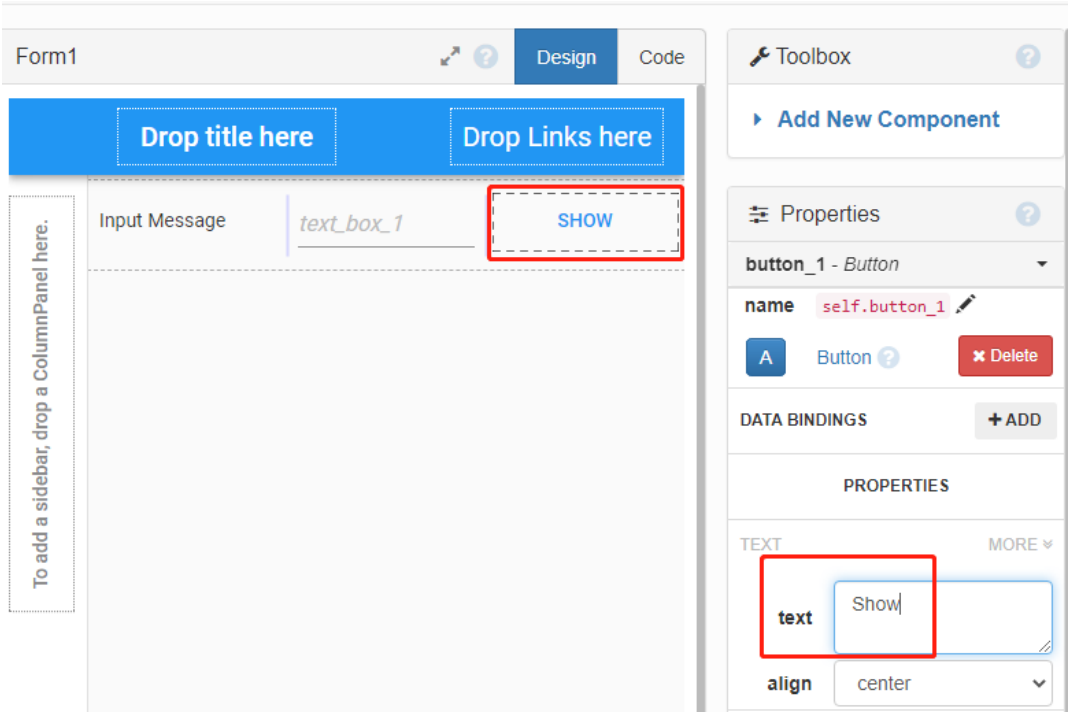
7. You can fill in the label text in the **text** field under the **Properties** menu.



8. In the same way, drag a **TextBox** to the right.



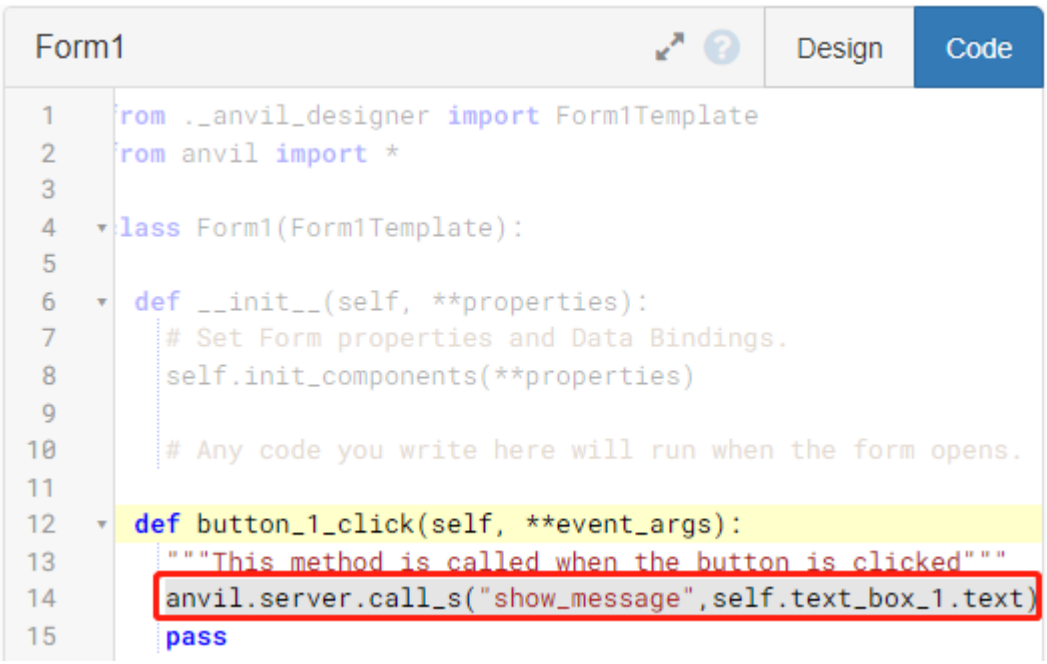
9. Drag a **button** to the far right and you can modify the **text** field. This button will be used to "send" a message to the Raspberry Pi Pico W.



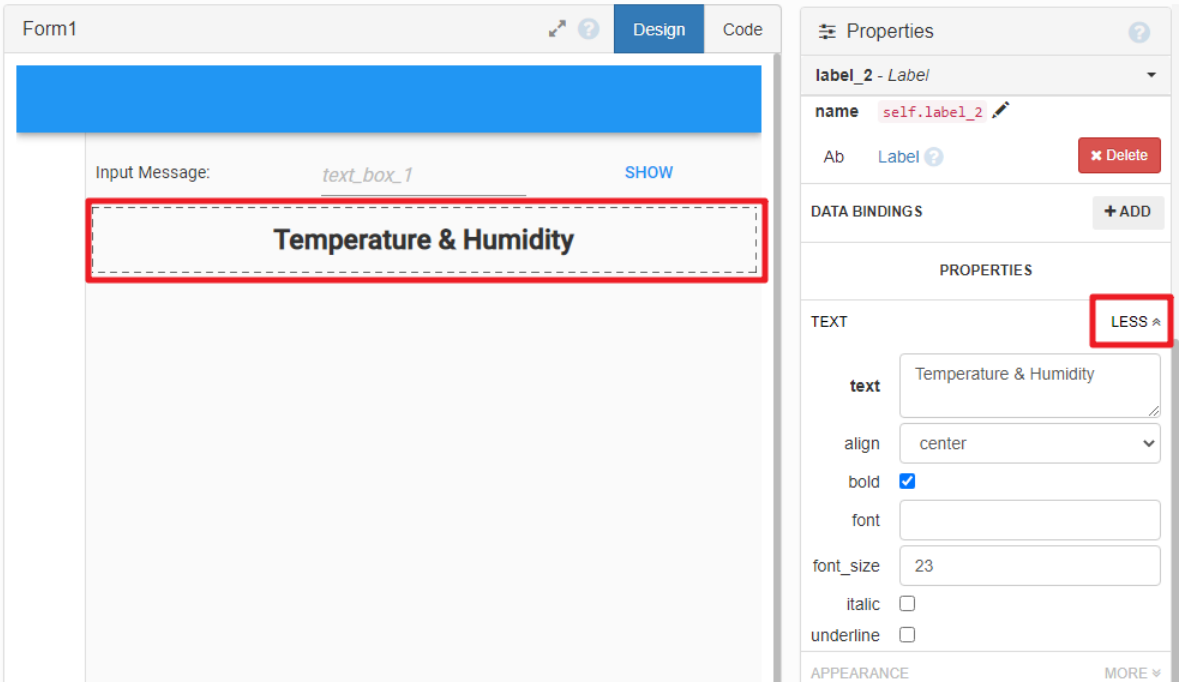
10. After double clicking the **SHOW** button, the form will switch from the Design page to the Code page and highlight the code for the **button**. You need to fill in the following code, which functions to call a function inside the server (in this case, Pico W).

```
anvil.server.call_s("show_message",self.text_box_1.text)
```

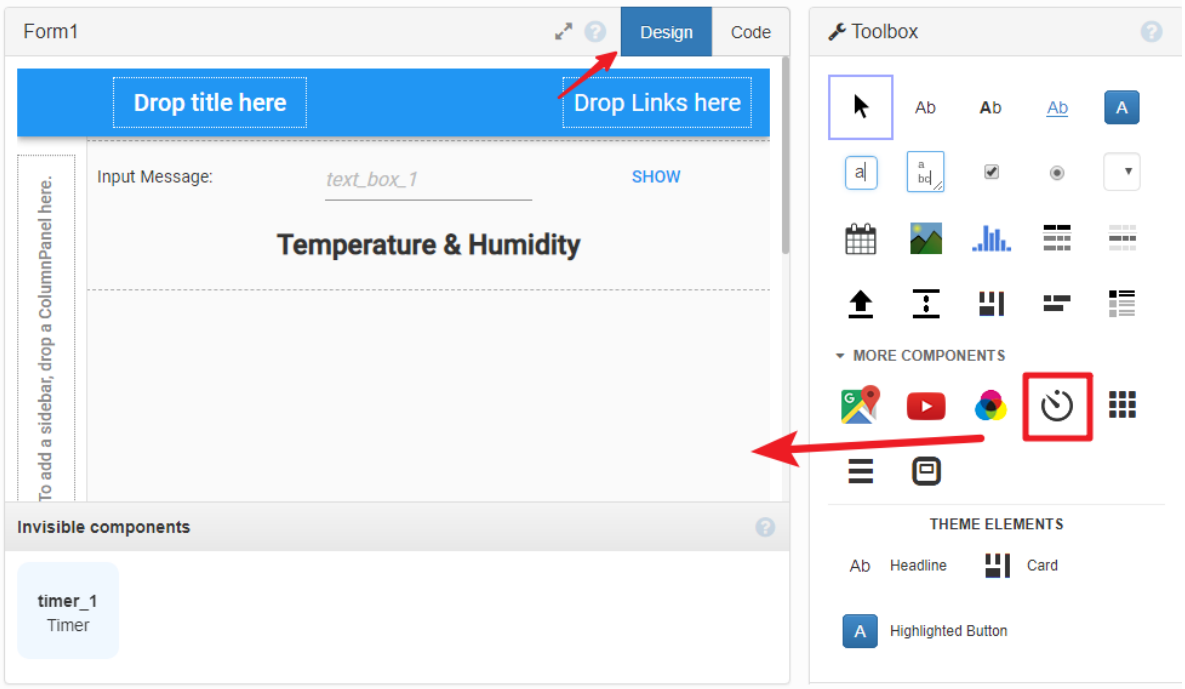
- `show_message` is the function that will be defined when Pico W is programmed.
- `self.text_box_1.text` is the message you type on the text box, which will be sent as a pass-through to `show_message()`.



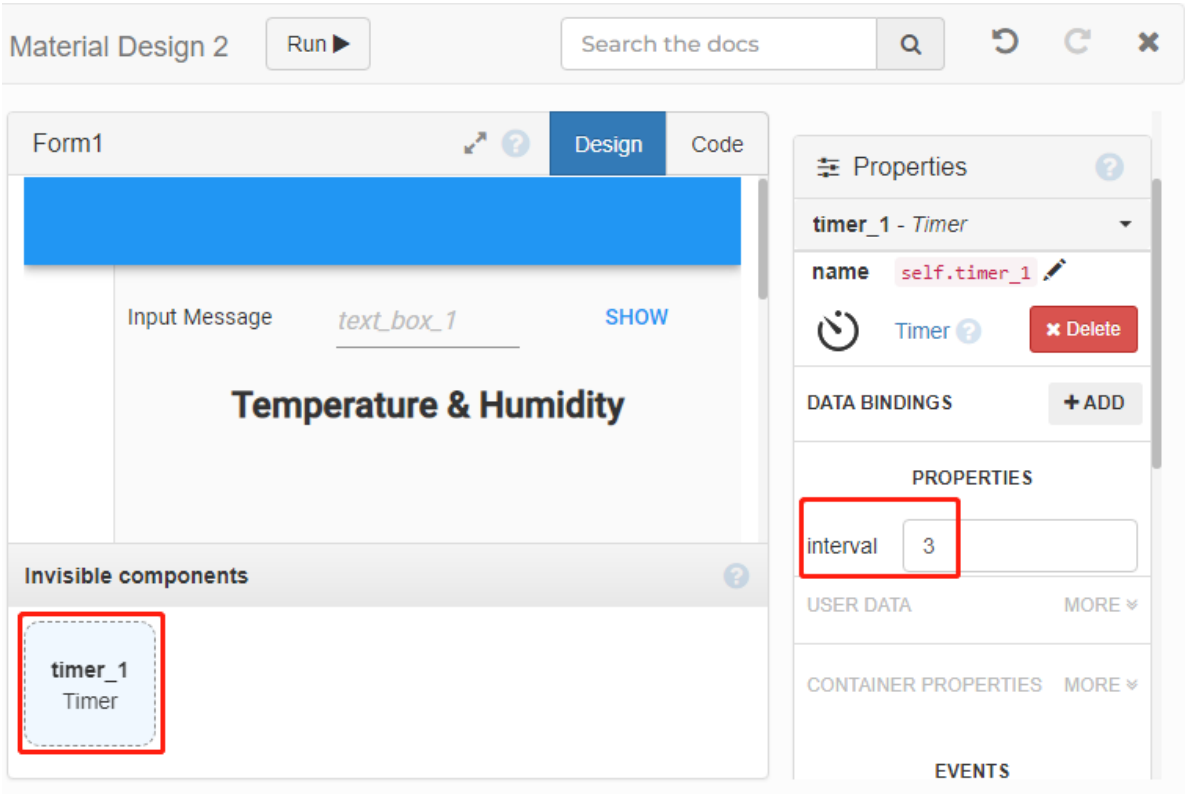
11. Go back to the Design page, drag another label and place it under the previous elements. This label will show the DHT11 sensor data from the Pico W.



12. In the **Toolbox**, click on **More Components** and drag the **Timer** into the form.

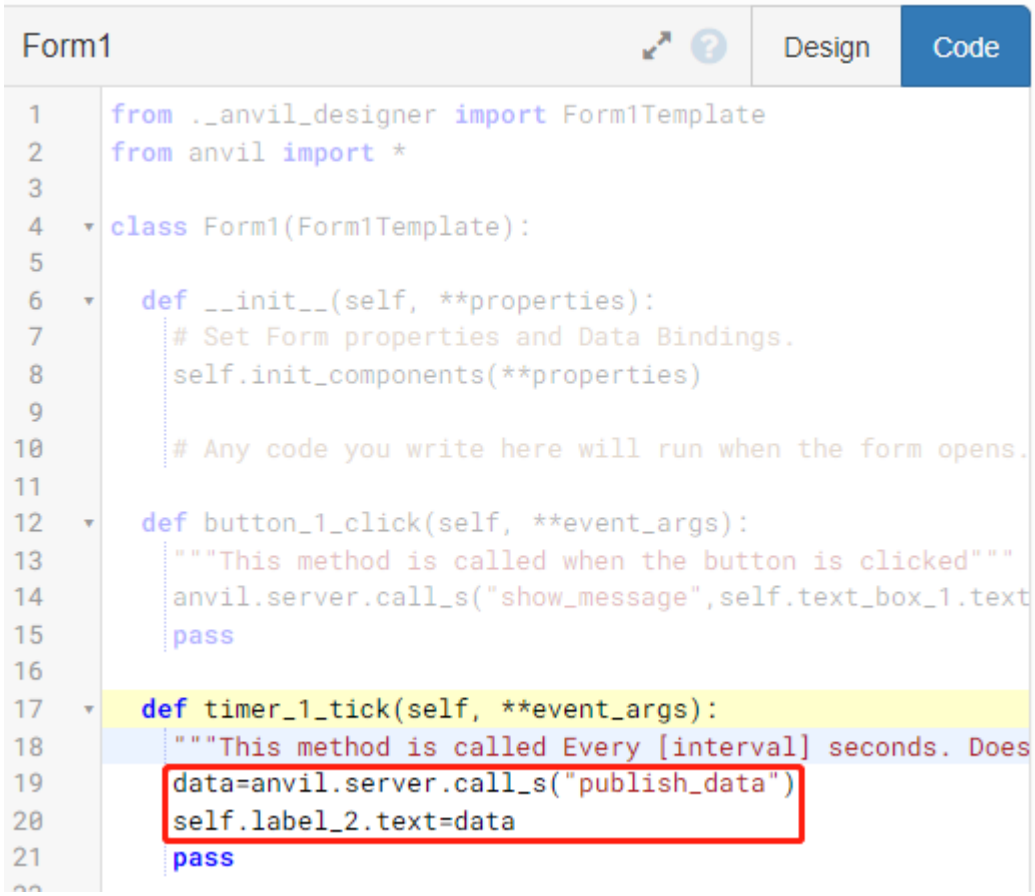


13. Using the **Properties**, set the timer to an interval of 3 seconds. This time is used to update the screen for our sensor data.



14. Double-click the **Timer** tool to program it. Use a `anvil.server.call_s()` function to call the `publish_data()` function from the server to get the message that needs to be displayed on the Anvil APP, assign it to `self.label_2.text` and you're done.

```
data=anvil.server.call_s("publish_data")
self.label_2.text=data
```

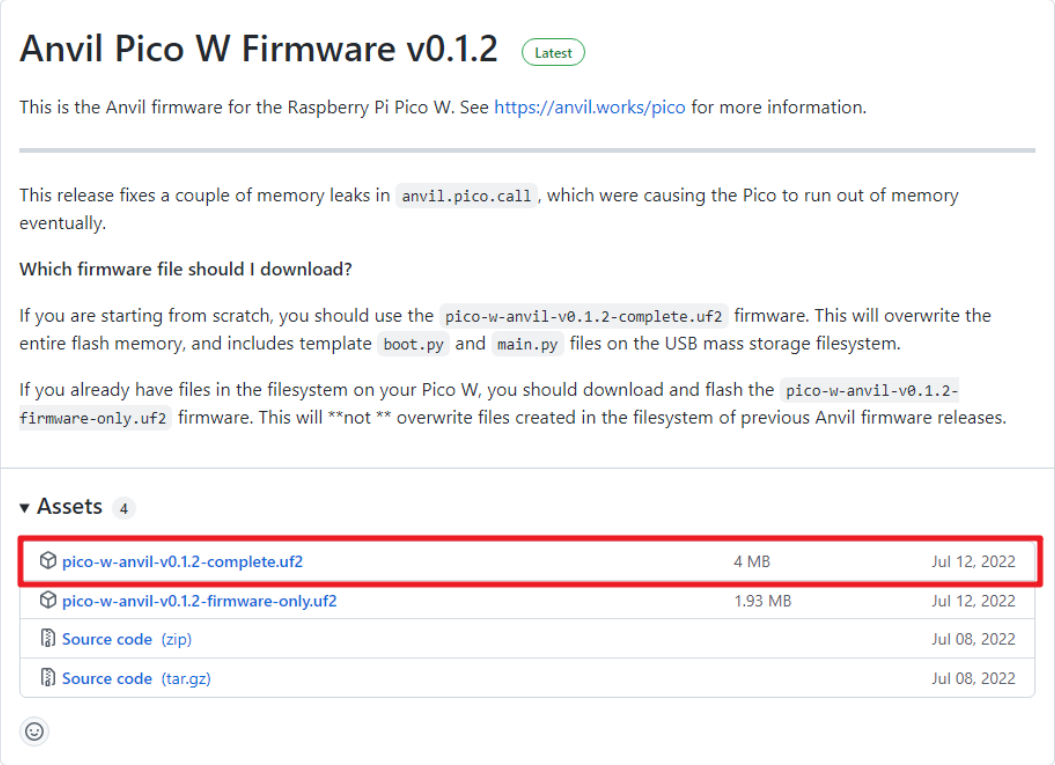


15. At this point, the part that needs to be programmed on Anvil is done. You can see more details on using Anvil from .

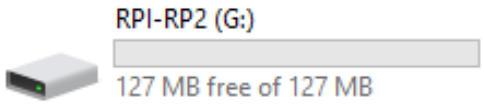
4. Setup Pico W

In order to simplify the connection of the Raspberry Pi Pico W to Anvil services, Anvil uses a custom firmware image. Pico W's firmware is written in MicroPython and appears as a USB drive with two files (boot.py and main.py). The Pico W must be flashed with custom firmware and connected to our Wi-Fi before we can begin writing code.

1. Download the custom Raspberry Pi Pico W , it is recommended to download the complete version.

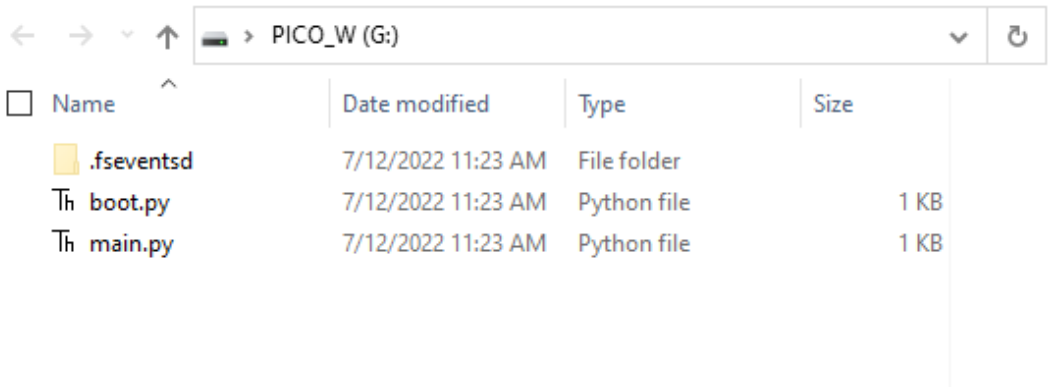


2. Push and hold the **BOOTSEL** button on the Pico W, then connect to your computer using a micro USB cable. Release BOOTSEL once the drive RPI-RP2 appears on your computer.

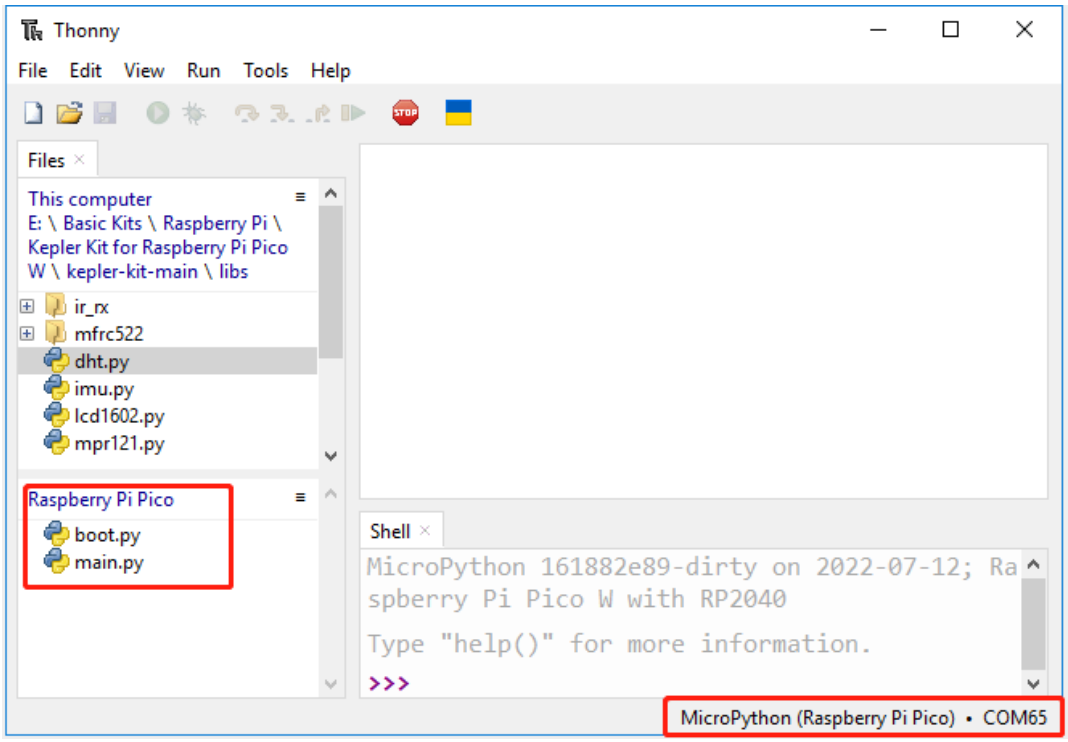


3. Drag and drop the `.uf2` file you just downloaded into it and Pico W will install the firmware at this point. When it is done, Pico W will reload the disk drive and you will see the `main.py` and `boot.py` files.

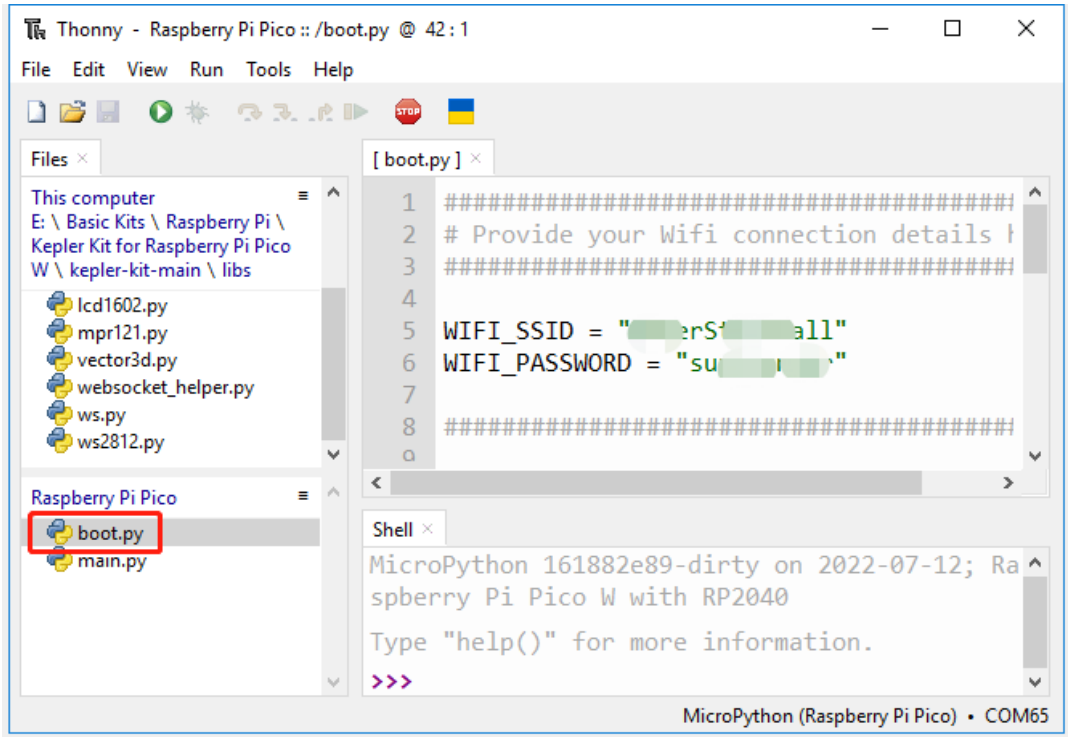
Note: Before reinstalling the firmware, make a backup of all your important files stored in Pico W.



4. Select "MicroPython(Raspberry Pi Pico).COMXX" as the interpreter in Thonny IDE. After clicking **View** -> **Files**, you will see the local drive and the hard drive of the Raspberry Pi Pico.

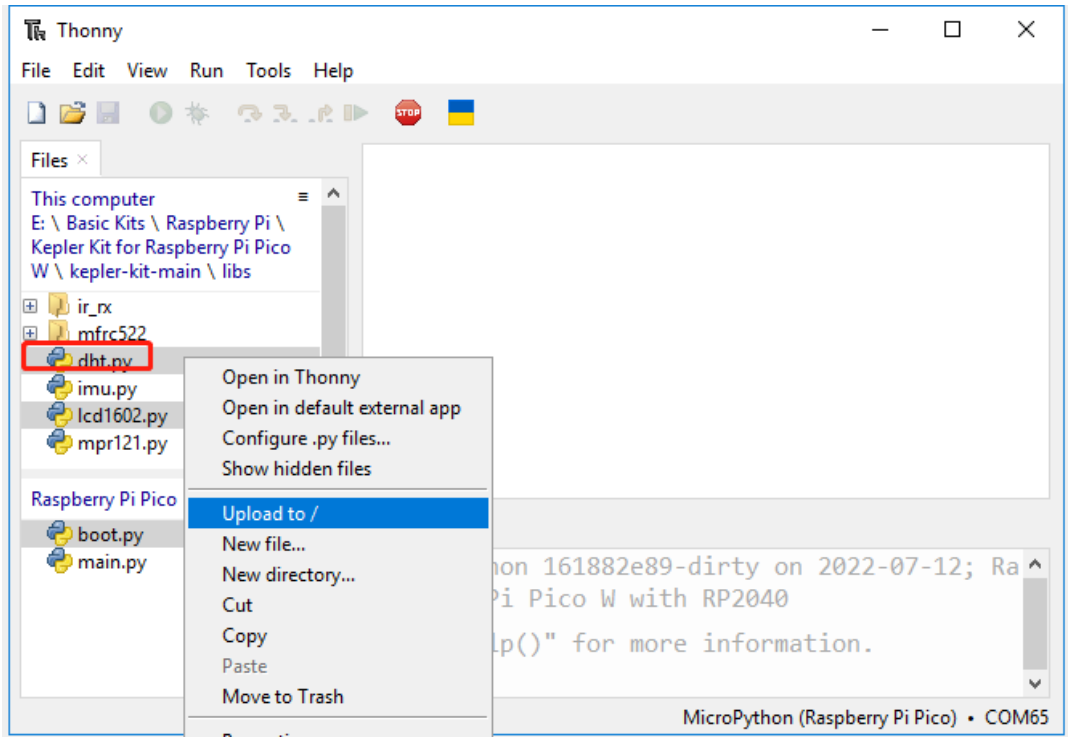


5. Double click the `boot.py` script and fill your WiFi's `SSID` and `PASSWORD`.



5. Finish the Code

1. Upload `dht.py` and `lcd1602.py` from the path of `kepler-kit-main/libs` to the Raspberry Pi Pico W.



2. Open `main.py` and replace the original code with the following code.

```
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

@anvil.pico.callable(is_async=True)
async def publish_data():
    sensor.measure()
    return "Temperature: {}, Humidity: {}".format(sensor.temperature,␣
↪sensor.humidity)

# Connect the Anvil Uplink. In MicroPython, this call will block forever.

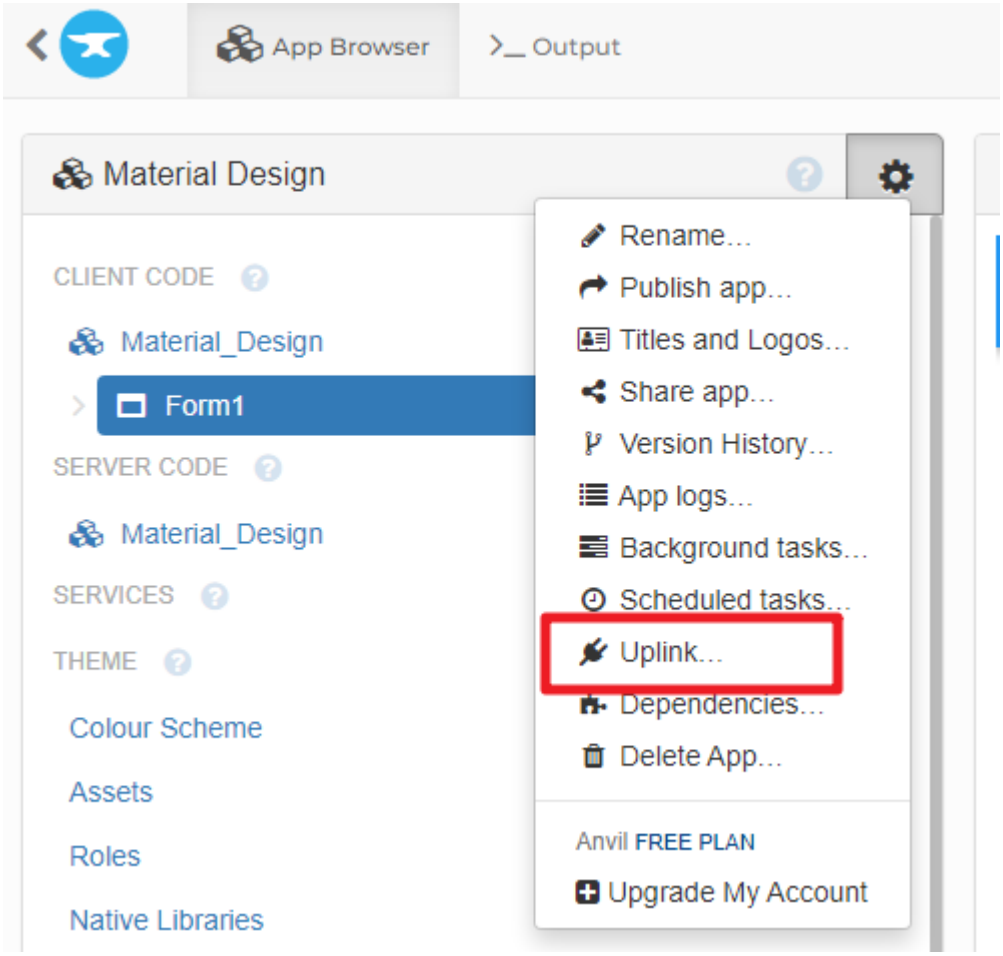
anvil.pico.connect(UPLINK_KEY)
```

(continues on next page)

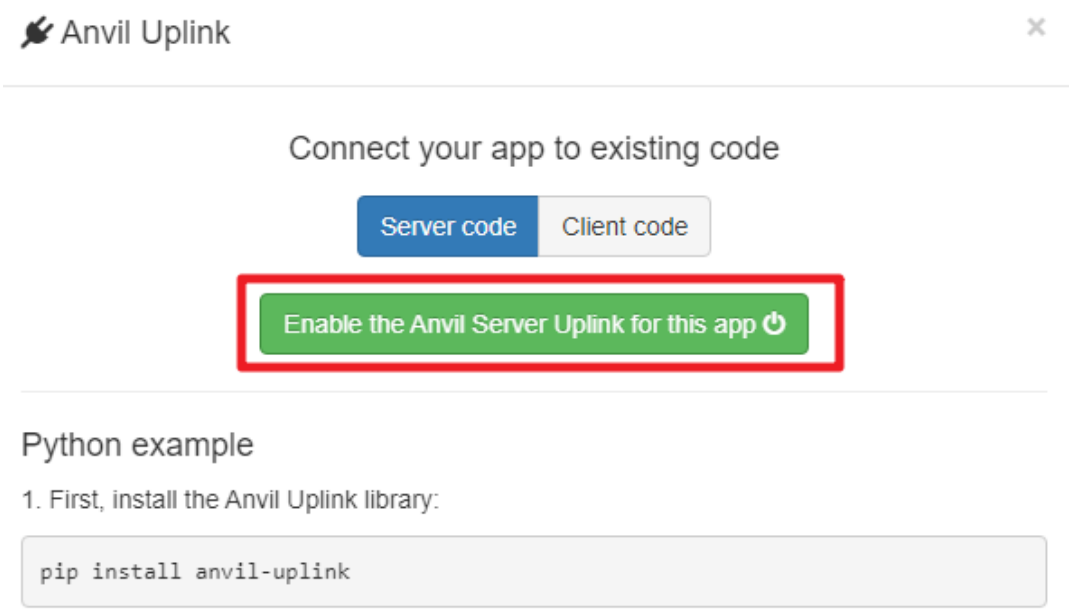
(continued from previous page)

```
# There's lots more you can do with Anvil on your Pico W.
#
# See https://anvil.works/pico for more information
```

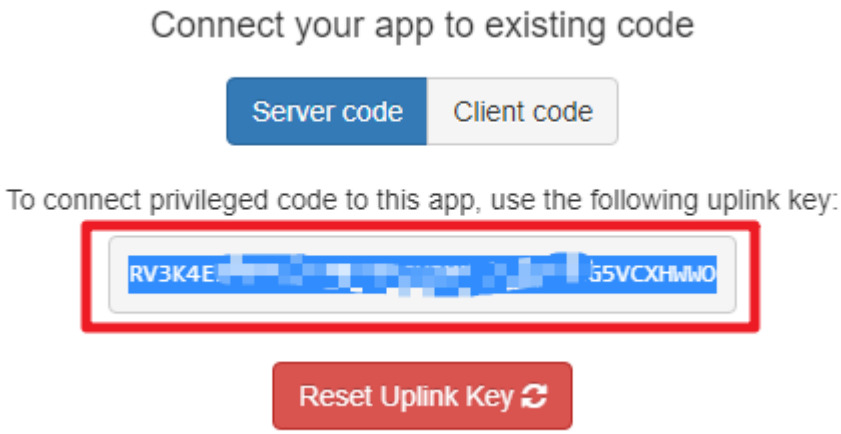
3. Go back to the Anvil interface and tap the Uplink option in the App Browser settings.



4. Click **Enable the Anvil Server Uplink for this app** to get the uplink key.

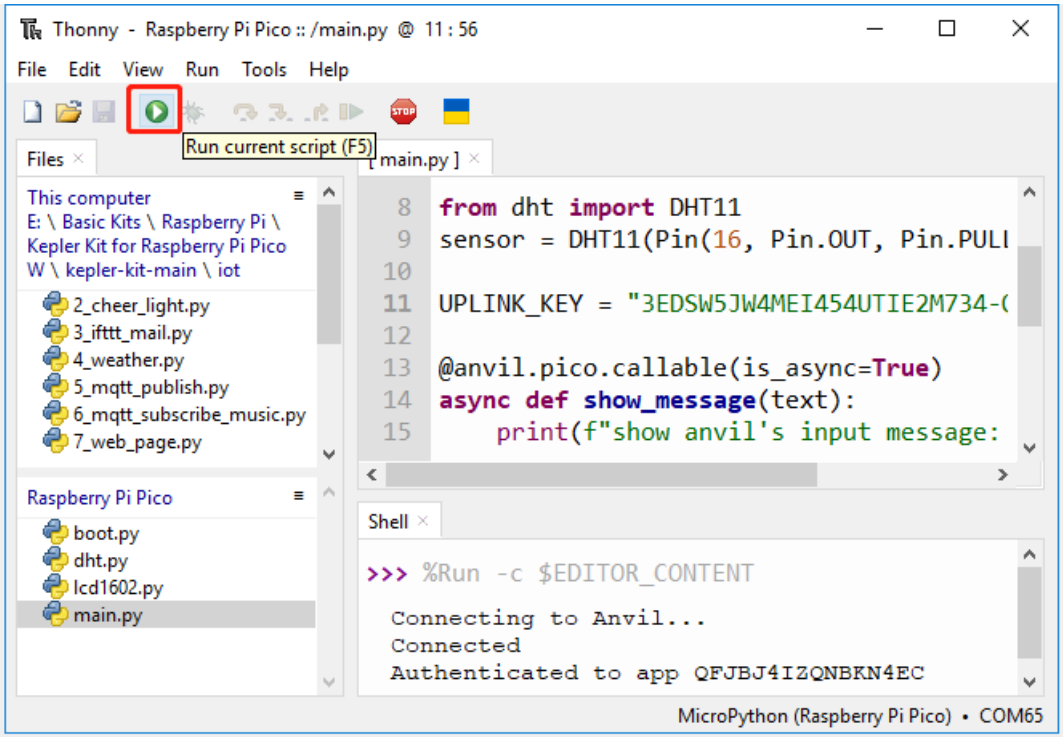


5. Copy it and replace <uplink_key_goes_here> in `main.py`, which makes your Pico W can connect to the Anvil APP you created.



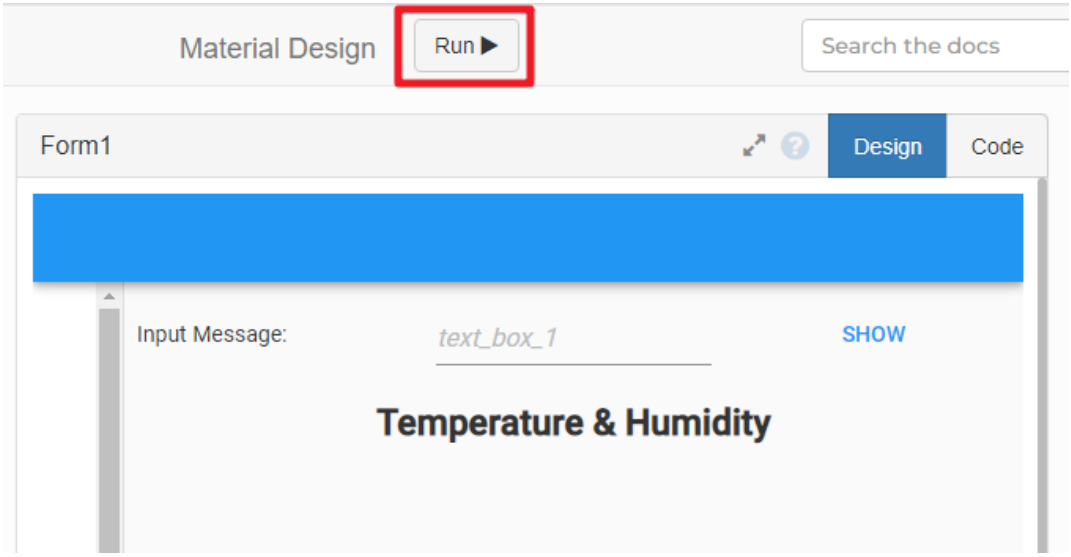
6. Run the Project

1. Click the **Run current script** button or press F5 to run it. After successful connection, you will see the connection success prompt in the Shell.



2. Run the Anvil, now you will be able to see the display temperature and humidity from the Anvil APP. If you click the **SHOW** button after you enter a message on the text box, the I2C LCD1602 will display the message you entered.

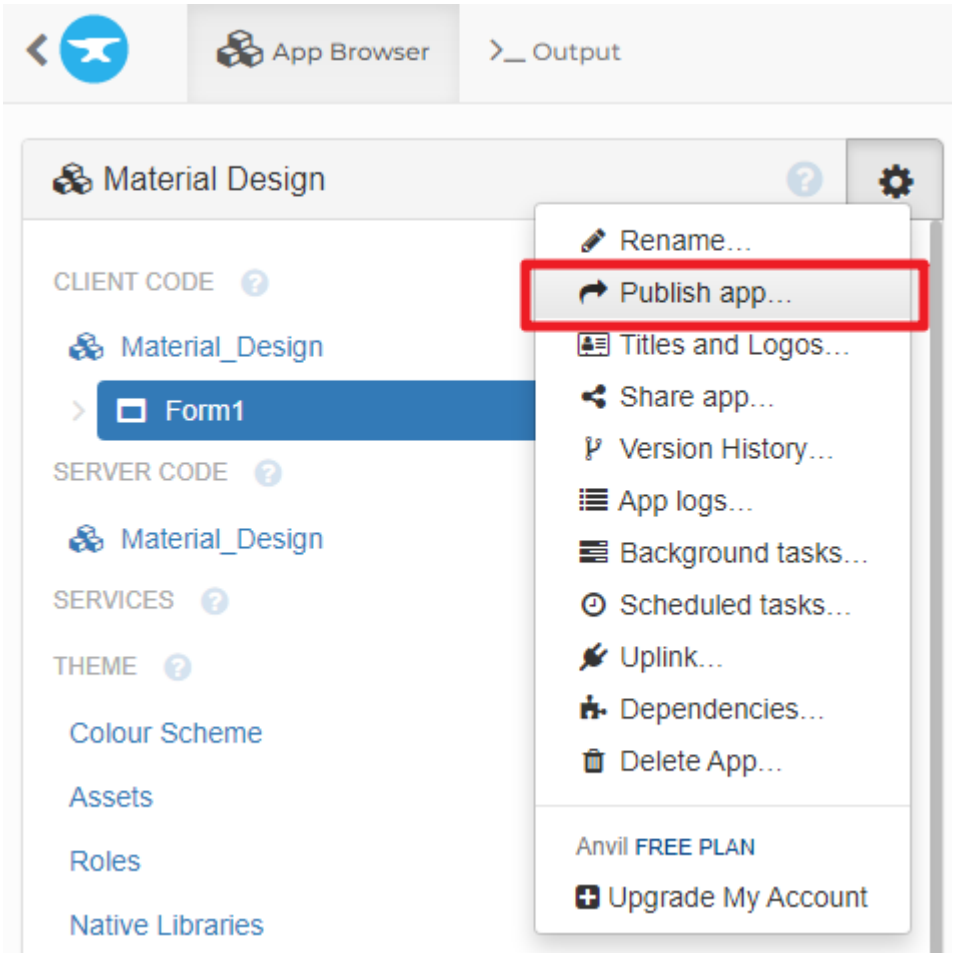
Note: If the character you entered is not displayed on the I2C LCD1602, you can turn the potentiometer on the back of the module to increase the contrast.



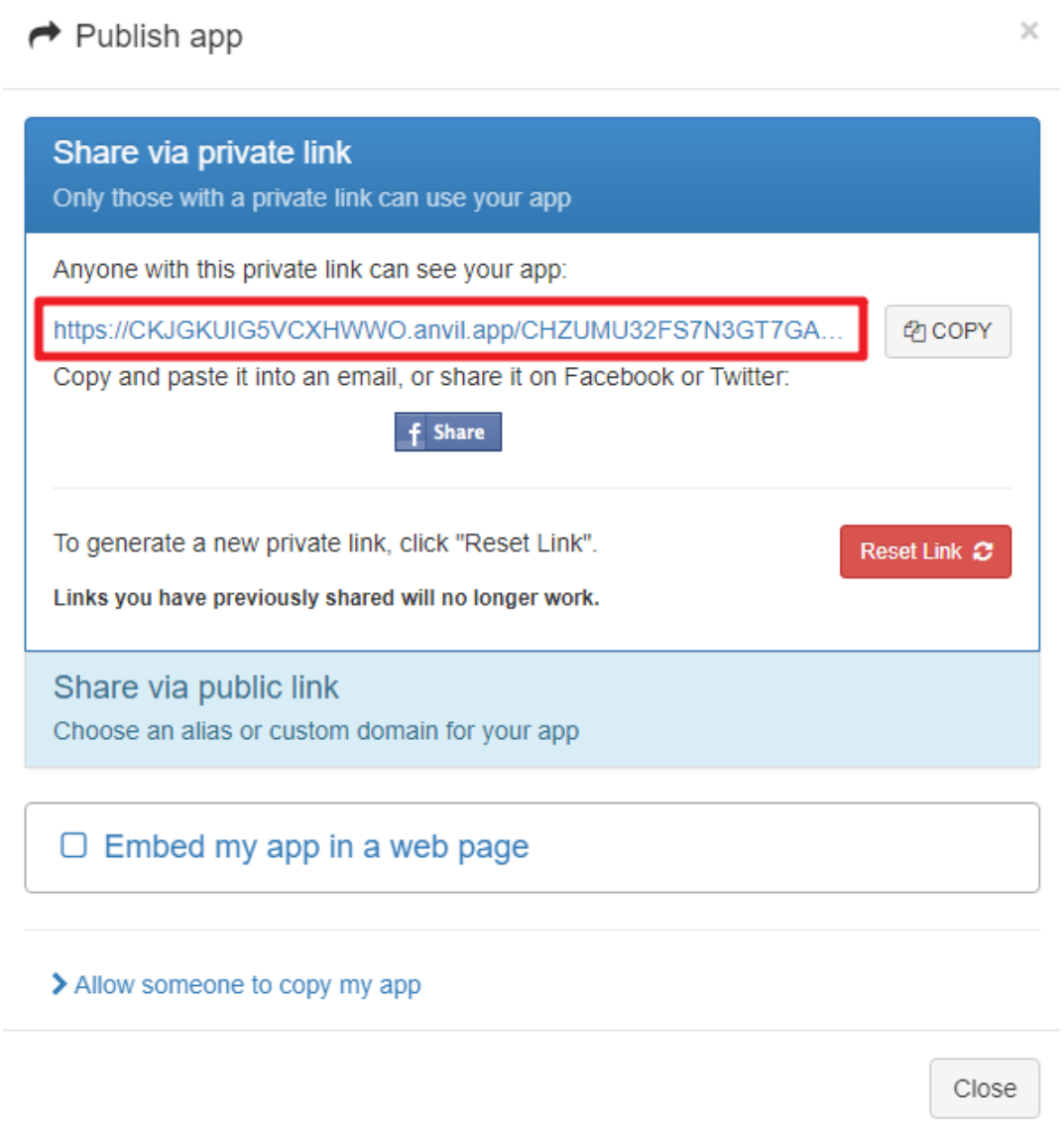
7. Publish APP

If you want to share your made app to others, you can generate a sharing link by the following method.

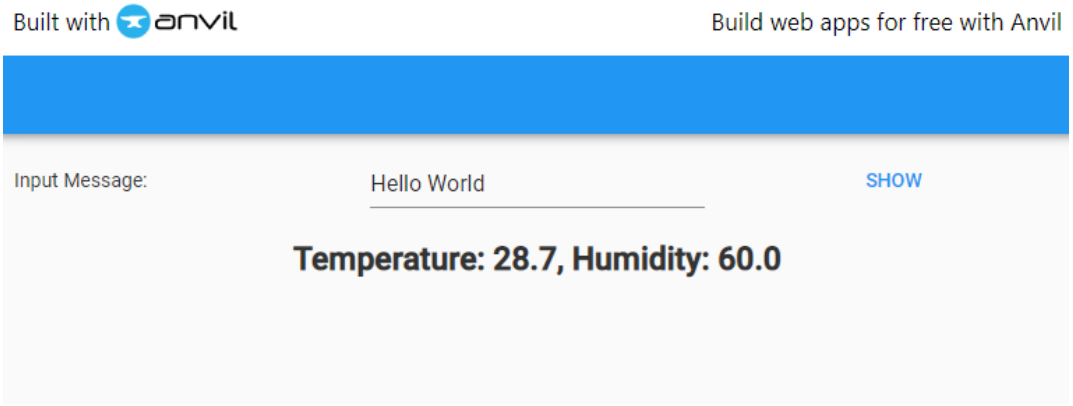
1. Go back to the **Anvil** page and click the **publish app** option in the** App Browser settings**.



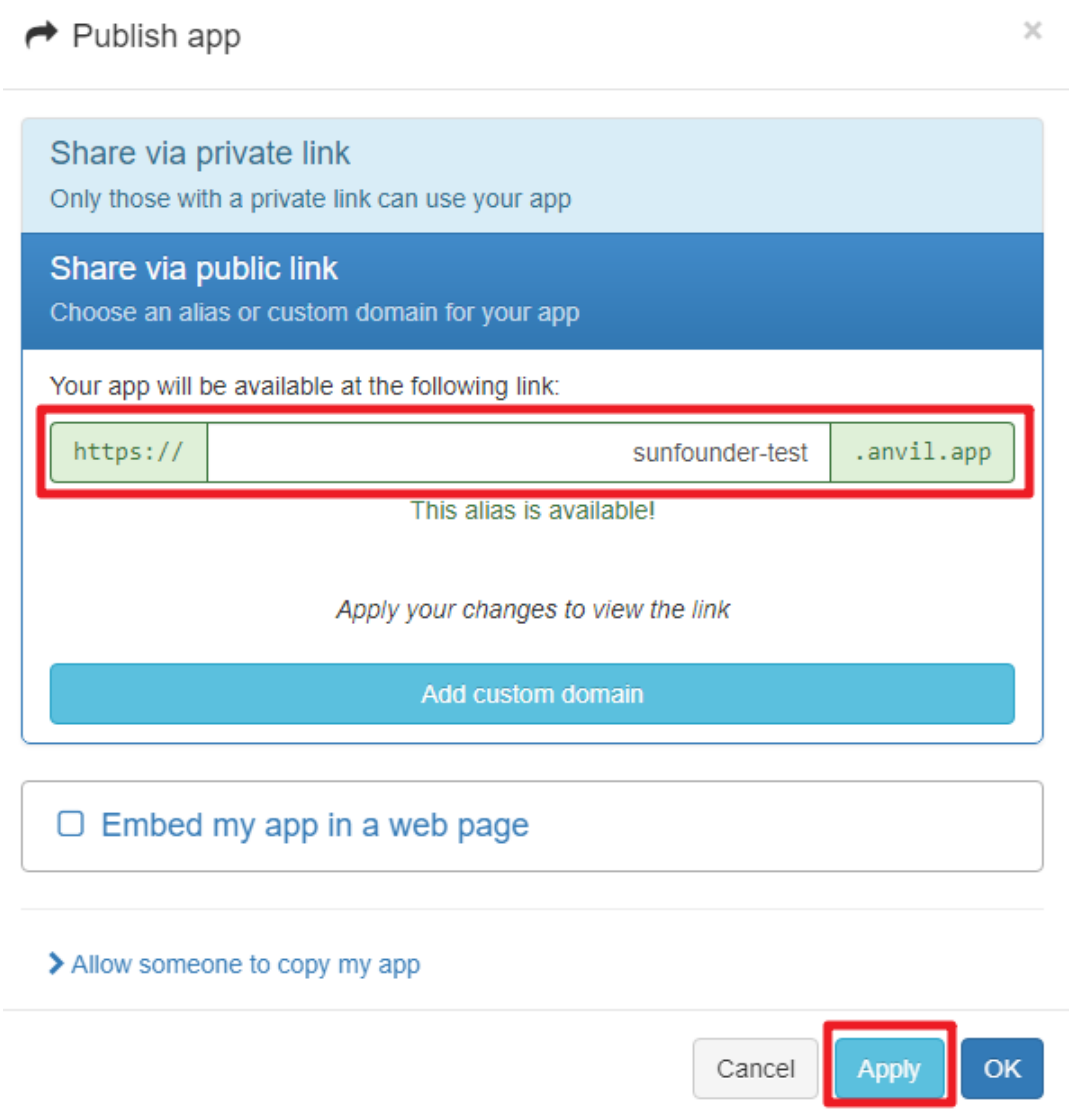
2. In the **Share via private link** tab you will see a list of links. Anyone can access your app through this link.



3. Access the link and your app is ready to use directly.



4. You can also share your app through the public link, fill in your personalized domain name and click **Apply** below to make it effective.



How it works?

Here is the basic framework of `main.py`, which forms the basis of the communication between Pico W and Anvil APP.

```
import anvil.pico
import uasyncio as a

UPLINK_KEY = "<uplink_key_goes_here>"


# Connect the Anvil Uplink. In MicroPython, this call will block forever.

anvil.pico.connect(UPLINK_KEY)


# There's lots more you can do with Anvil on your Pico W.
#
# See https://anvil.works/pico for more information
```

Set up dht11 and lcd1602. Details of the use of these two components can be found in *6.2 Temperature - Humidity* and *3.4 Liquid Crystal Display*.

```
from machine import Pin,I2C

from lcd1602 import LCD
lcd = LCD()

from dht import DHT11
sensor = DHT11(Pin(16, Pin.OUT, Pin.PULL_DOWN))
```

In Anvil's code, we have called two internal functions of the server (Pico W).

The first is `show_message()`, whose function is to make the LCD display the message typed by Anvil. The decorator `@anvil.pico.callable(is_async=True)` makes this function callable by Anvil.

```
@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
```

Next is `publish_data()`, which functions to get the value of dht11 and return the temperature and humidity to Anvil. Again, it uses the decorator `@anvil.pico.callable(is_async=True)` to make it callable by Avil.

```
@anvil.pico.callable(is_async=True)
async def publish_data():
    sensor.measure()
    return "Temperature: {}, Humidity: {}".format(sensor.temperature, sensor.humidity)
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

5.9 9. Play in @SunFounder Controller

In this project, you will learn how to build a remote project using Sunfounder Controller APP. In a LAN environment, you can control your Pico W circuit with your phone/tablet. You will find this app very useful if you want to build a simple robot with Pico W.

Here, we will use the slider bar on the APP to control the servo angle and the gauge on the APP to show the distance detected by ultrasonic.

1. Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

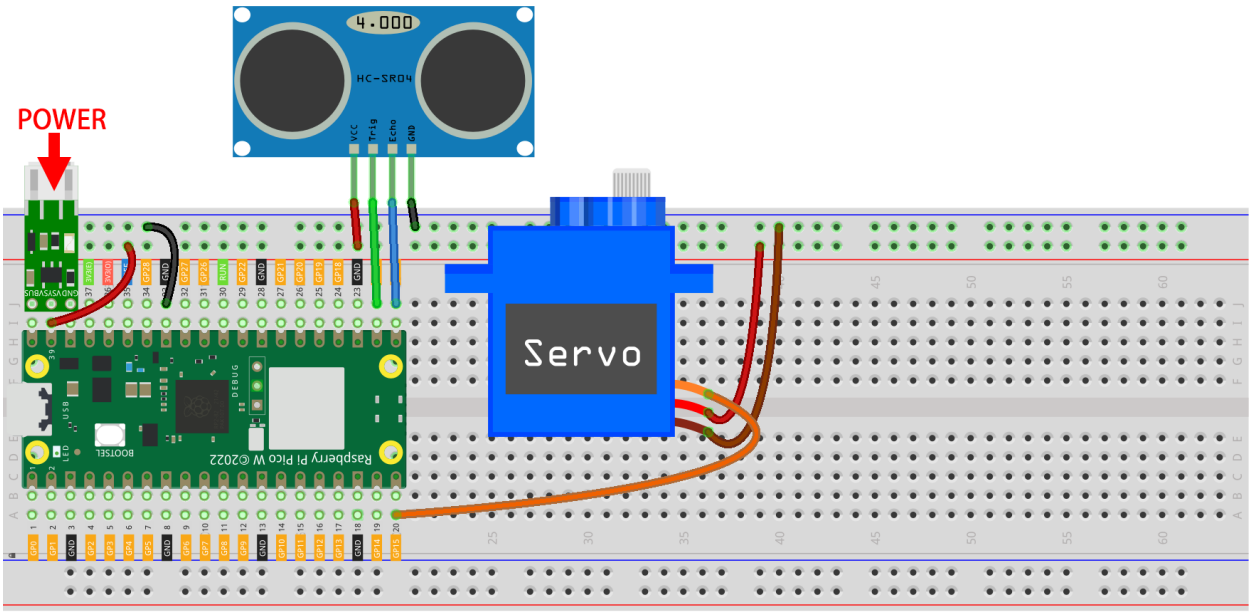
Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Servo*	1	
6	*Ultrasonic Module*	1	
7	*Li-po Charger Module*	1	
8	18650 Battery	1	
9	Battery Holder	1	

2. Build the Circuit

> **Warning:** Make sure your Li-po Charger Module is connected as shown in the diagram. Otherwise, a short circuit will likely damage your battery and circuitry.

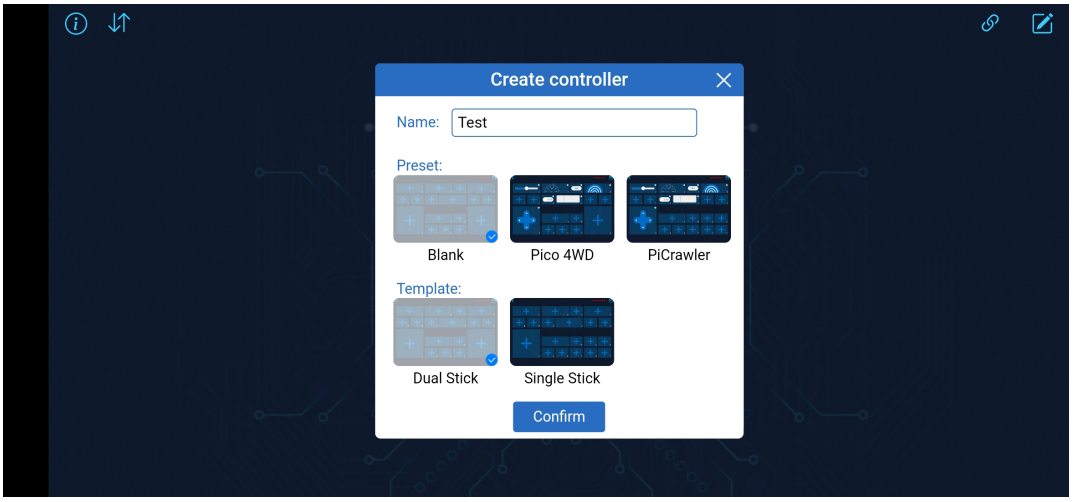


3. Setup SunFounder Controller

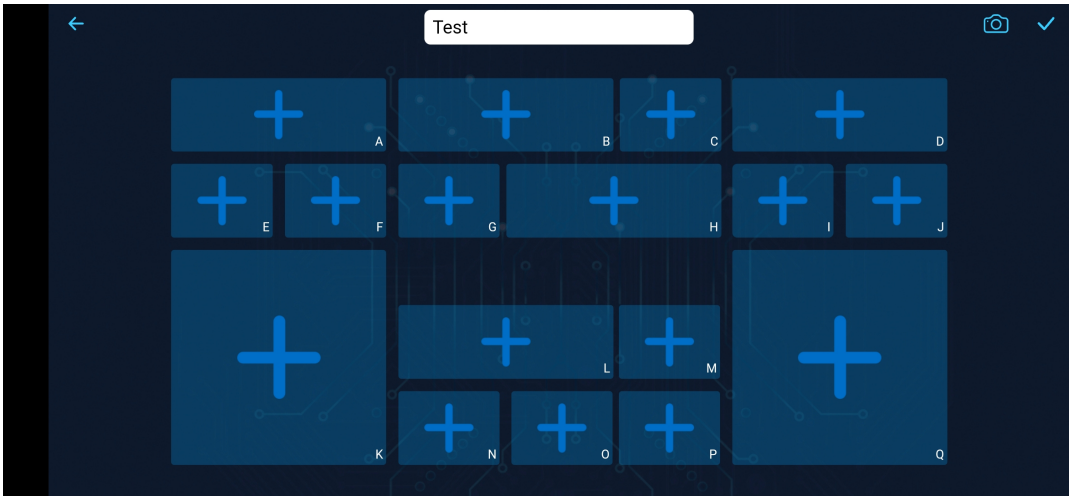
1. Install SunFounder Controller APP from **APP Store(iOS)** or **Google Play(Android)**.
2. Open the APP and click the **+** button on the home page to create a controller.



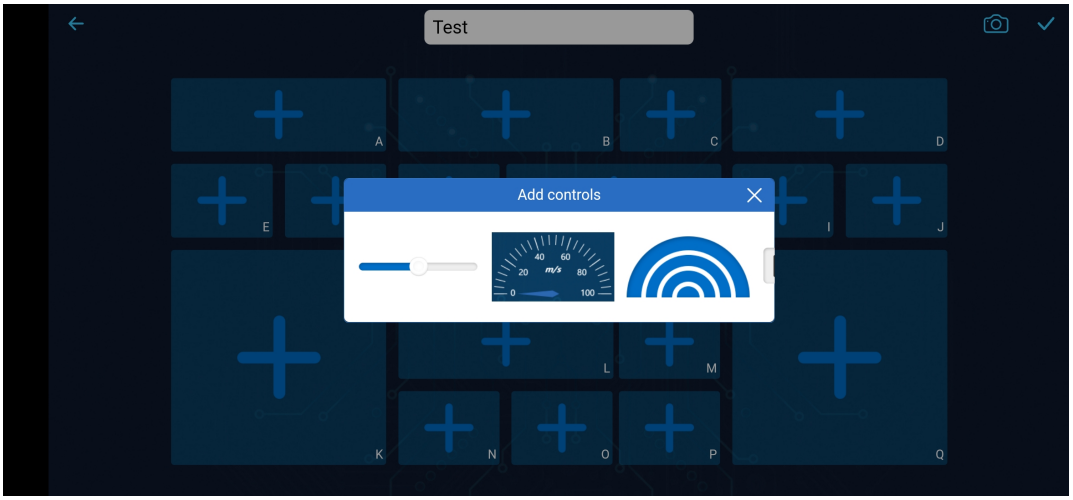
3. Here we choose **Blank** and **Dual Stick**.



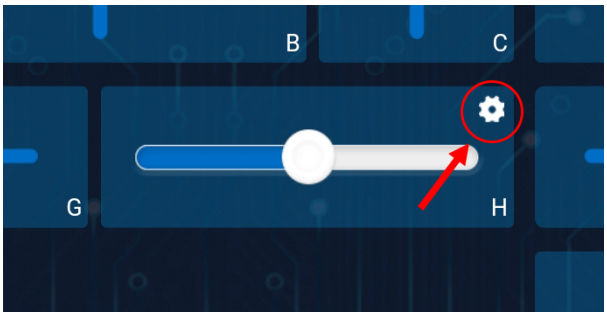
4. Now we get an empty controller.



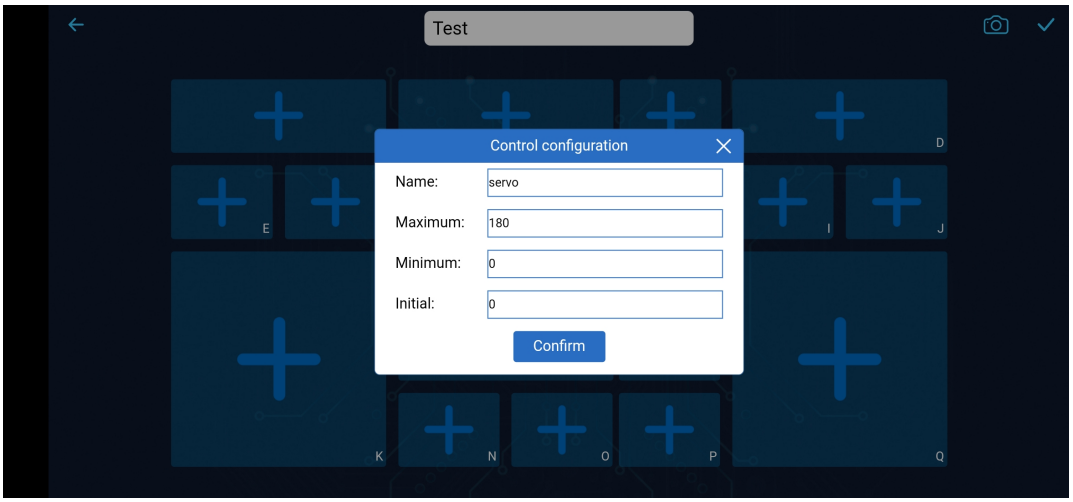
5. Click on the **H** area and add a **Slider** widget.



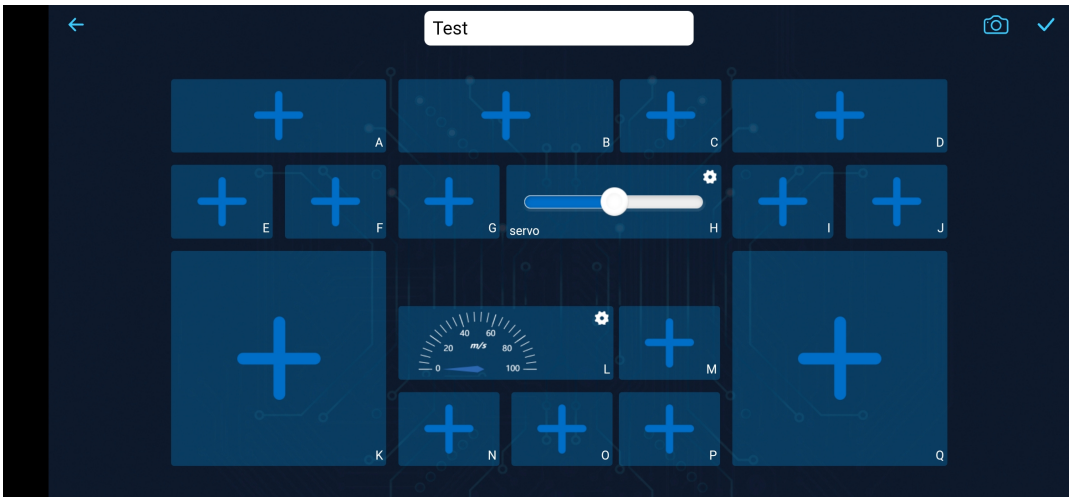
6. Click the gear on the control to open the settings window.



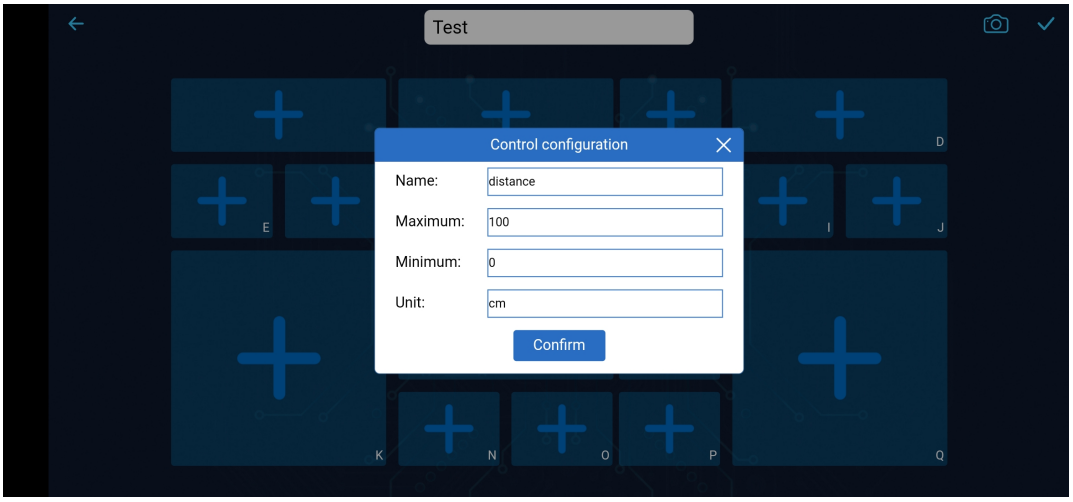
7. Set Maximum to 180 and Minimum to 0, then click to **Confirm**.



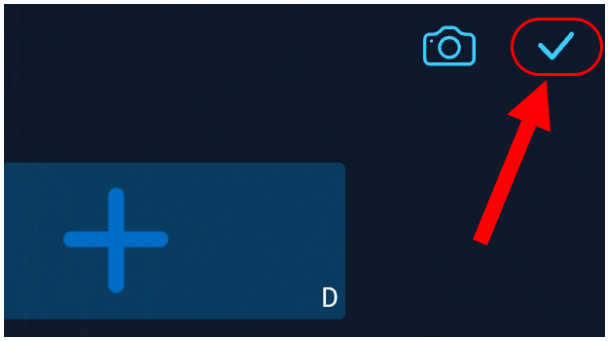
8. Click on the L area and add a Gauge widget.



9. Click the gear of the Gauge, open the settings window, set Maximum to 100, Minimum to 0, and unit to cm.



10. After finishing the widget settings, click Save.

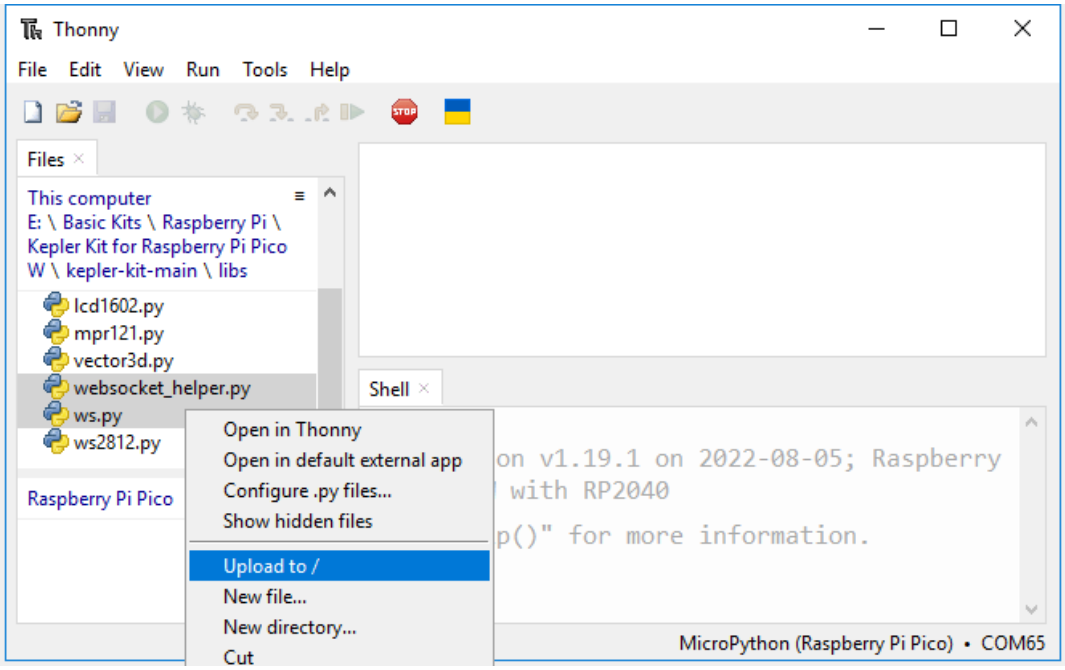


4. Run the Code

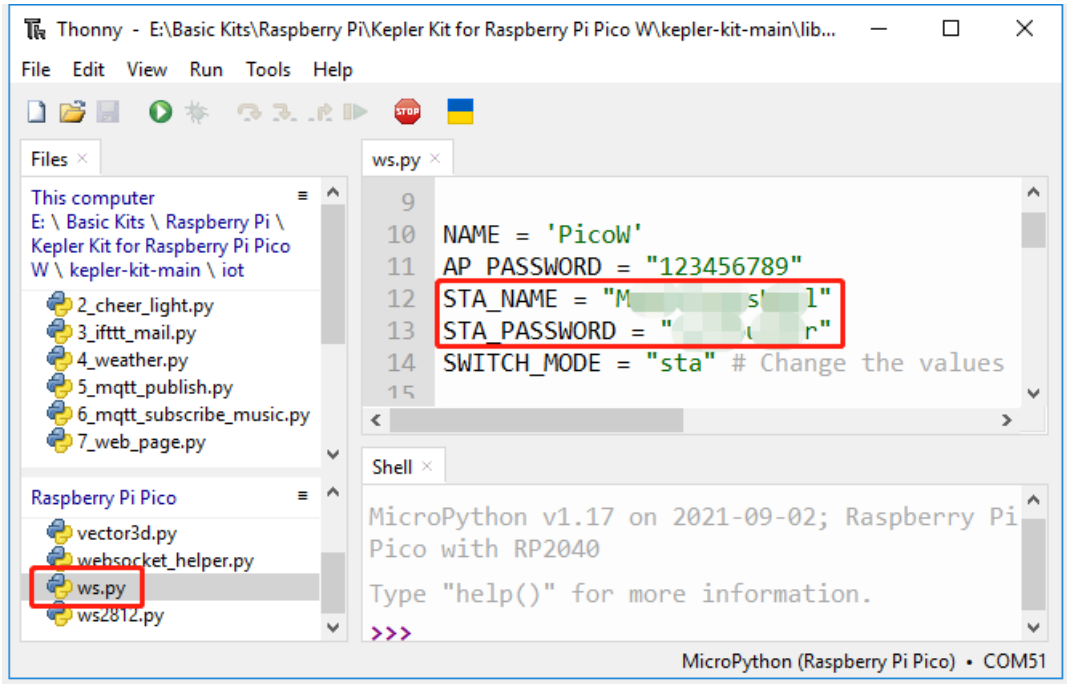
Note: If your Pico W is now using the Anvil firmware, then you will need to *1.3 Install MicroPython on Your Pico*.

---

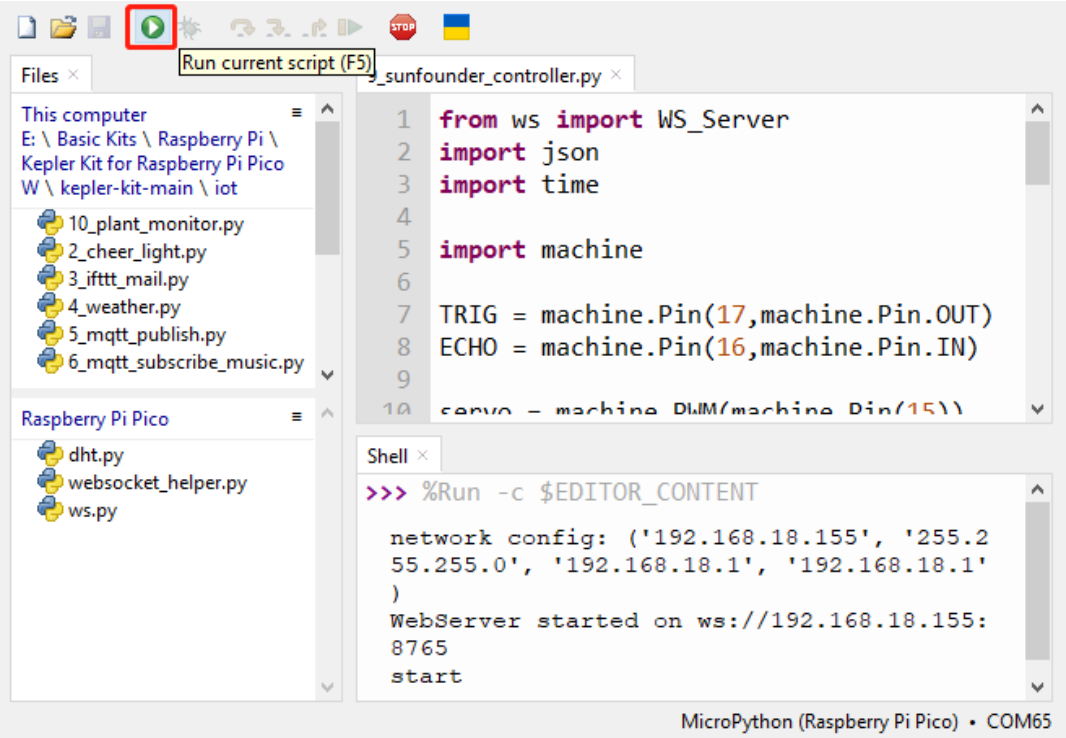
1. Upload `ws.py` and `websocket_helper.py` from the path of `kepler-kit-main/libs` to the Raspberry Pi Pico W.



2. Double click the `ws.py` script and fill your WiFi's `SSID` and `PASSWORD`.

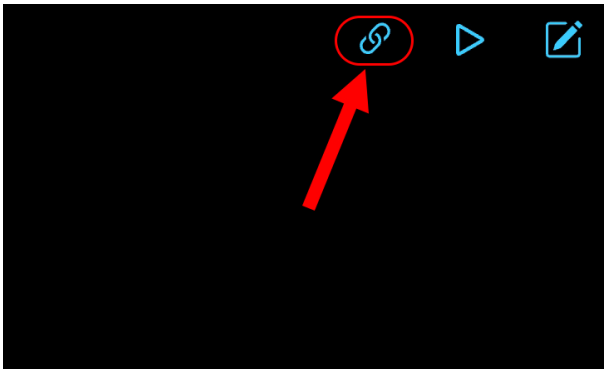


3. Open the `9_sunfounder_controller.py` under the path of `kepler-kit-main/iot`. Click the **Run current script** button or press F5 to run it. After successful connection, you will see the IP of Pico W.

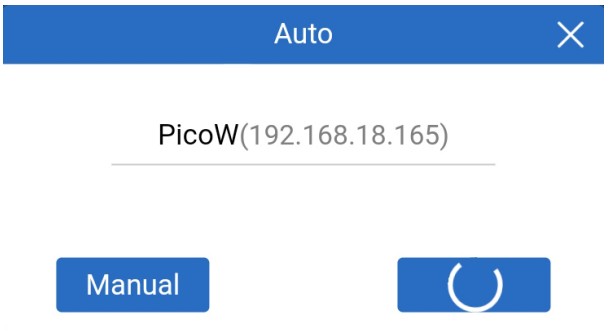


Note: If you want this script to be able to boot up, you can save it to the Raspberry Pi Pico W as `main.py`.

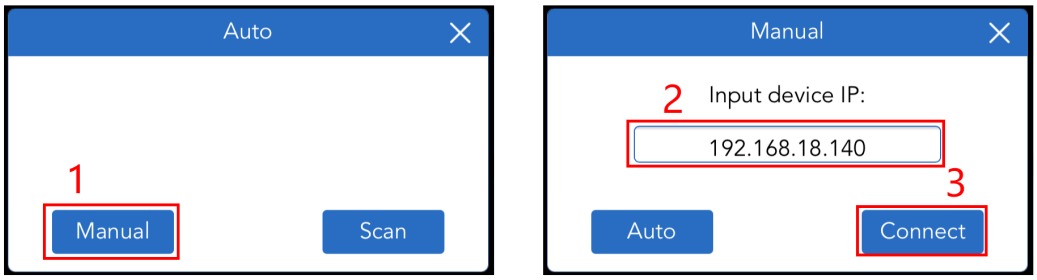
4. Back to SunFounder Controller APP, click the **Connect** button.



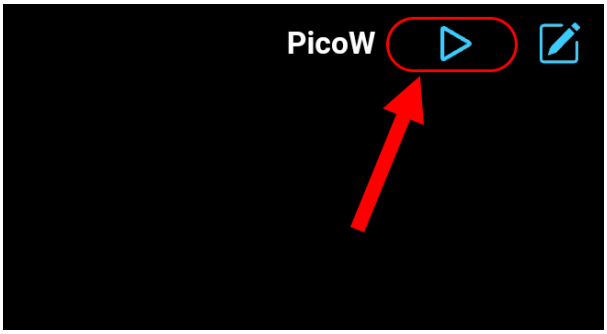
5. If PicoW is detected, tap it directly to connect.



6. If it doesn't search automatically, you can also manually enter the IP to connect.



7. When you slide the slider bar in the H area after clicking the Run button, the servo will adjust its angle. The gauge in the L area will show the distance if your hand is within 100cm of the ultrasonic sensor.



How it works?

The `WS_Server` class in the `ws.py` library implements communication with the APP. Below is the framework for implementing its basic functionality.

```
from ws import WS_Server
import json
import time

ws = WS_Server(8765) # init websocket

def main():
    ws.start()
    while True:
        status,result=ws.transfer()
        time.sleep_ms(100)

try:
    main()
finally:
    ws.stop()
```

First, we need to create a `WS_Server` object.

```
ws = WS_Server(8765)
```

Star it.

```
ws.start()
```

Next, a `while True` loop is used to perform the data transfer between Pico W and the SunFounder Controller APP.

```
while True:
    #  websocket transfer data
    status,result = ws.transfer()

    # the status of transfer data
    print(status)

    # the data you recv
    print(result)

    # the data you send
    print(ws.send_dict)


    time.sleep_ms(100)
```

`status` is `False` if it fails to get data from the SunFounder Controller APP.

And `result` is the data that Pico W fetched from the SunFounder Controller APP. Print it out and you will see something like the following. This is the value of all Widget areas.

```
{'C': None, 'B': None, 'M': None,,,,, 'A': None, 'R': None}
```

As in this case, we print the values of the H area separately and use them to operate the circuit.

```
status,result=ws.transfer()
#print(result)
if status == True:
    print(result['H'])
```

And the `ws.send_dict` dictionary is the data that Pico W sends to the SunFounder Controller APP. It is created in the `WS_Server` class. It will be sent when `ws.transfer()` is executed.

Its message is shown below.

```
{'Check': 'SunFounder Controller', 'Name': 'PicoW', 'Type': 'Blank'}
```

This is a blank message, to copy it to the widget on SunFounder Controller APP, we need to assign the value to the corresponding area in the dictionary. For example, assign the value `50` to the L area.

```
ws.send_dict['L'] = 50
```

The data is shown below:

```
{'L': 50, 'Type': 'Blank', 'Name': 'PicoW', 'Check': 'SunFounder Controller'}
```

For more details on using SunFounder Controller, please see SunFounder Controller APP.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.

- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

5.10 10. Plant Monitor in @SunFounder Controller

With this project, you will learn how to build a plant watering system using the Sunfounder Controller APP.

In the APP, you can check the current temperature and humidity of the environment, as well as the level of water in potted plants. To water the plants, you can also click the button on the APP.

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

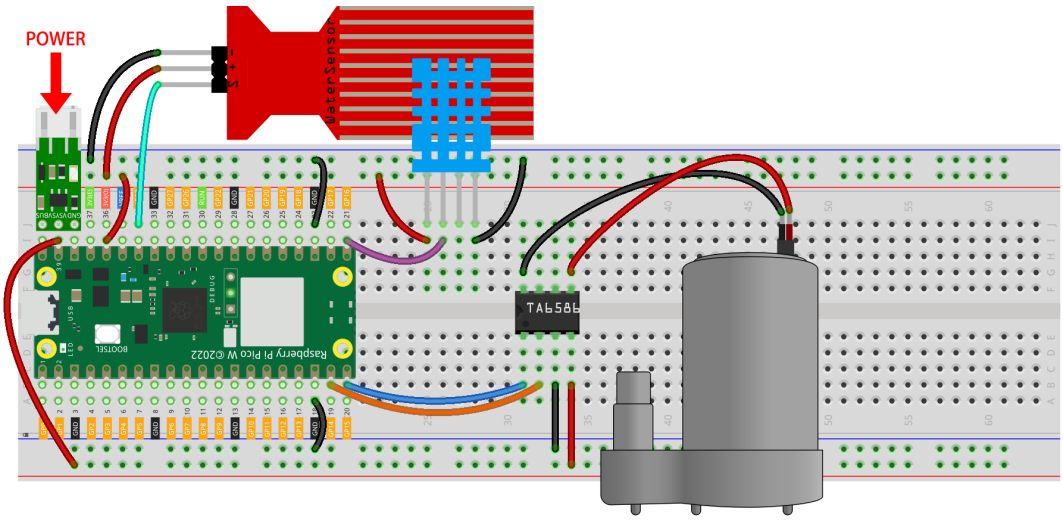
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*DHT11 Humiture Sensor*	1	
6	*Water Level Sensor Module*	1	
7	*TA6586 - Motor Driver Chip*	1	
8	*Li-po Charger Module*	1	
9	18650 Battery	1	
10	Battery Holder	1	
11	*DC Water Pump*	1	

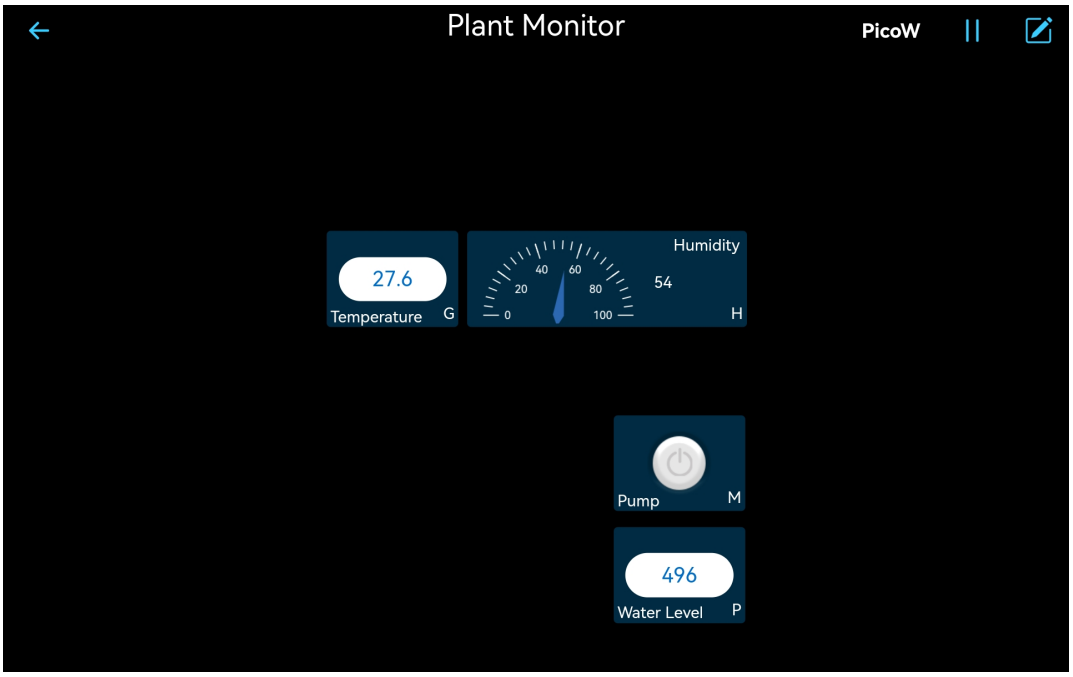
Steps

Note: It is recommended that you complete the previous project *9. Play in @SunFounder Controller*, as that will help you get to grips with the basic use of the SunFounder Controller.

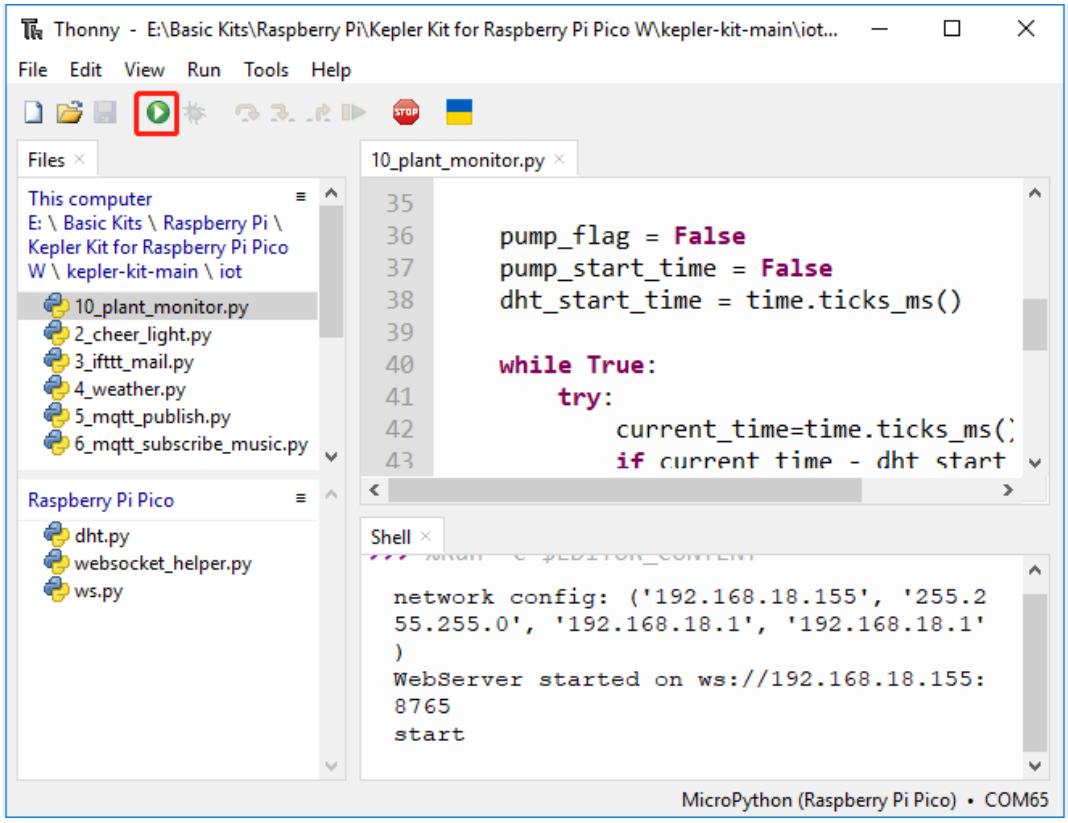
1. Build the circuit.



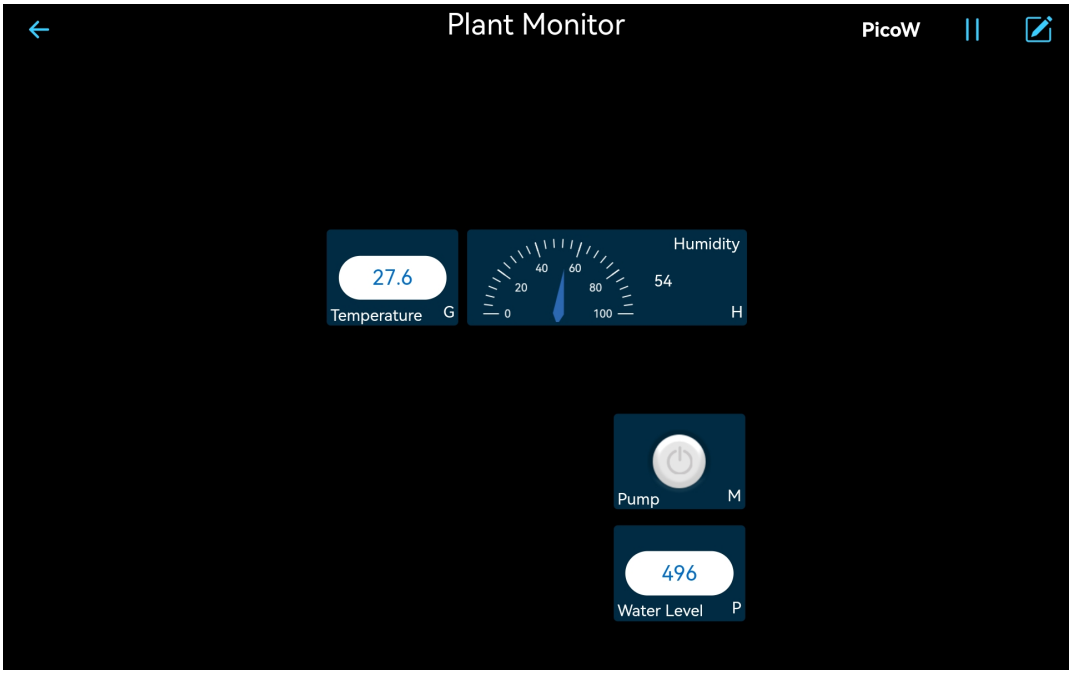
2. Create a new controller, add the following widgets and change their names.



3. Open the `10_plant_monitor.py` under the path of `kepler-kit-main/iot`. Click the **Run current script** button or press F5 to run it. After successful connection, you will see the IP of Pico W.



4. Returning to the SunFounder APP, click Run after connecting to PicoW. On the APP, you will be able to see the temperature and humidity of the environment, as well as the water level of the potted plant. You can click the button to water the potted plant for five seconds if you think it's short of water.



5. If you want this script to be able to boot up, you can save it to the Raspberry Pi Pico W as `main.py`.

How it works?

This project works basically the same way as *9. Play in @SunFounder Controller*.

In addition the project also uses DHT11, pump and water level module, you can see the details of using these components in *6.2 Temperature - Humidity* , *3.6 Pumping*, *2.14 Feel the Water Level*.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

CHAPTER

SIX

FOR ARDUINO USER

This chapter includes installing Arduino IDE, uploading code to Raspberry Pi with Arduino IDE and a dozen interesting and practical projects to help you learn Arduino code quickly.

We recommend that you read the chapters in order.

1. Get Started

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.1 1.1 Install Arduino IDE(Important)

The Arduino IDE, known as Arduino Integrated Development Environment, provides all the software support needed to complete an Arduino project. It is a programming software specifically designed for Arduino, provided by the Arduino team, that allows us to write programs and upload them to the Arduino board.

The Arduino IDE 2.0 is an open-source project. It is a big step from its sturdy predecessor, Arduino IDE 1.x, and comes with revamped UI, improved board & library manager, debugger, autocomplete feature and much more.

In this tutorial, we will show how to download and install the Arduino IDE 2.0 on your Windows, Mac, or Linux computer.

6.1.1 Requirements

- Windows - Win 10 and newer, 64 bits
- Linux - 64 bits
- Mac OS X - Version 10.14: “Mojave” or newer, 64 bits

6.1.2 Download the Arduino IDE 2.0

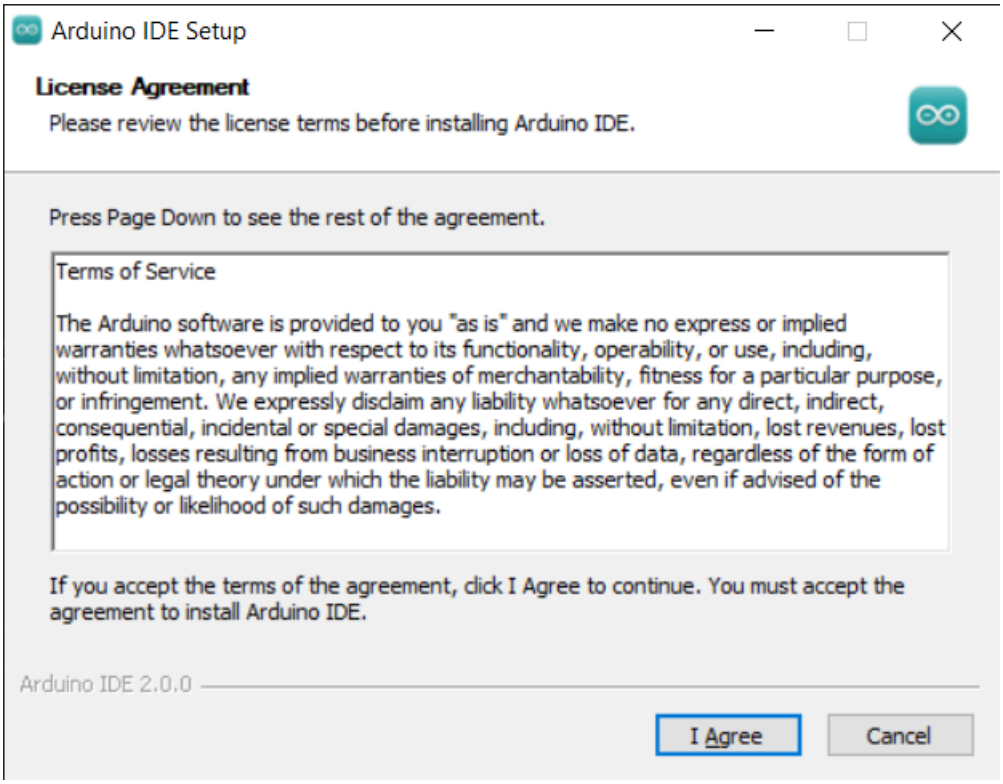
1. Visit page.
2. Download the IDE for your OS version.



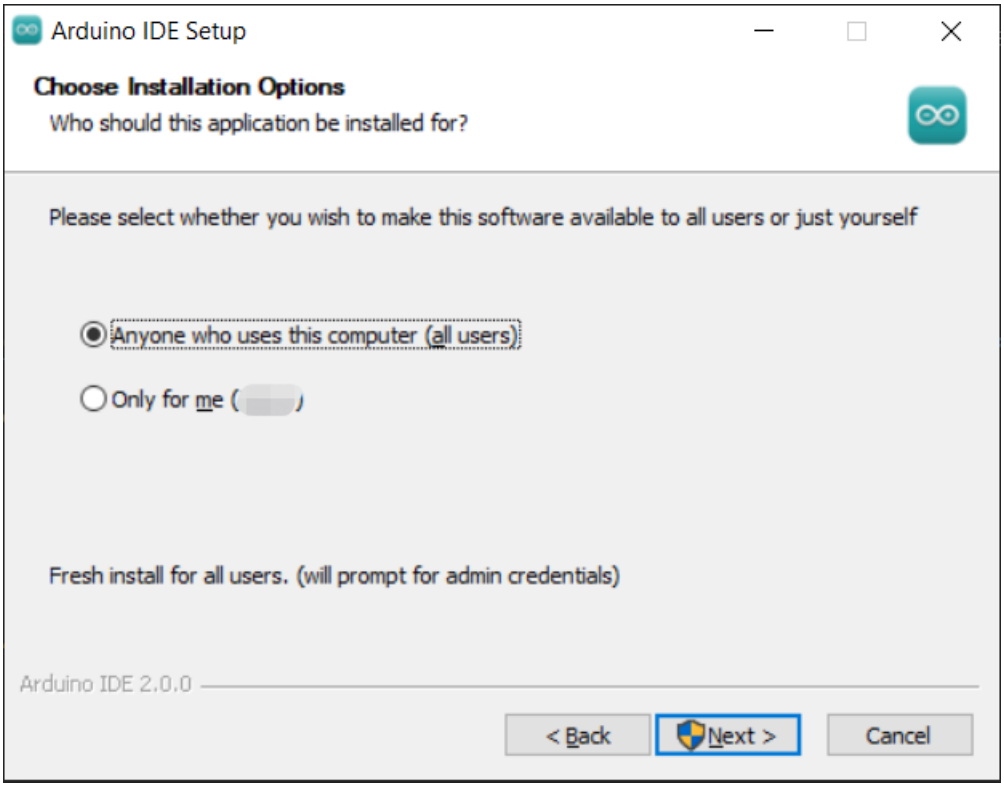
6.1.3 Installation

Windows

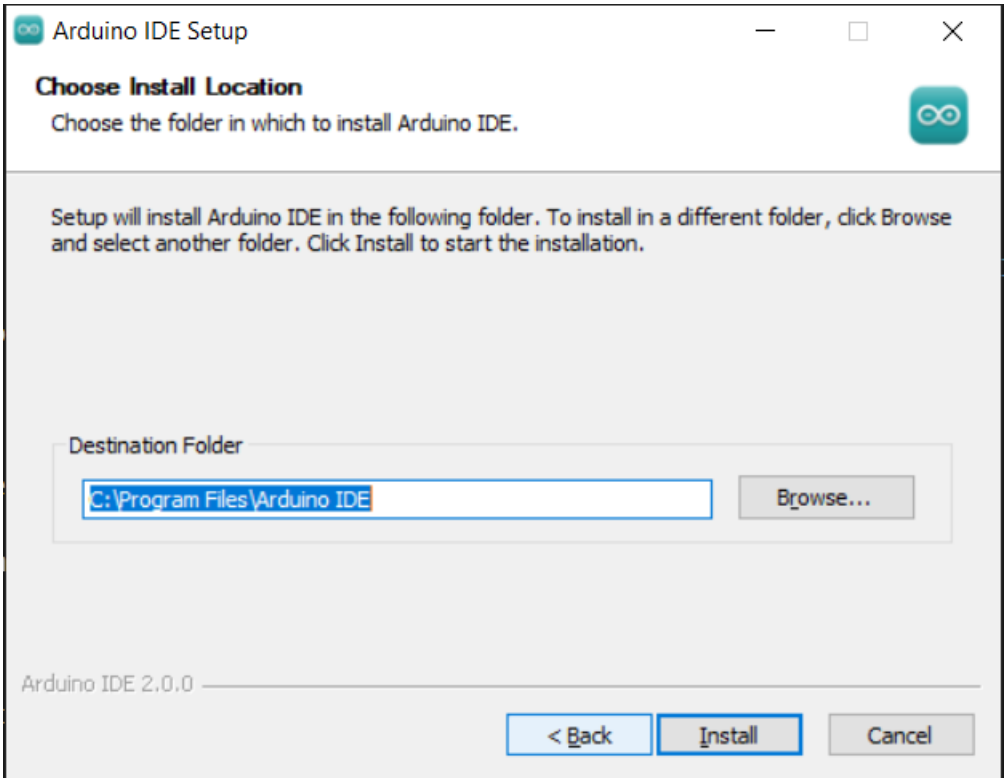
1. Double click the `arduino-ide_xxxx.exe` file to run the downloaded file.
2. Read the License Agreement and agree it.



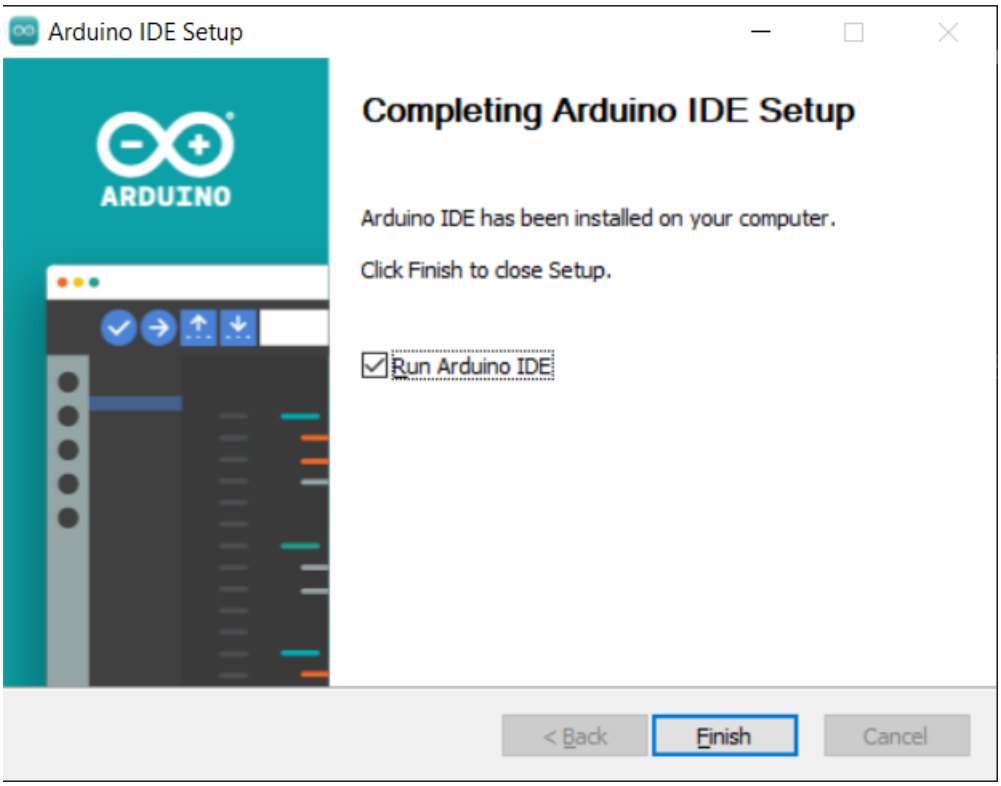
3. Choose installation options.



4. Choose install location. It is recommended that the software be installed on a drive other than the system drive.

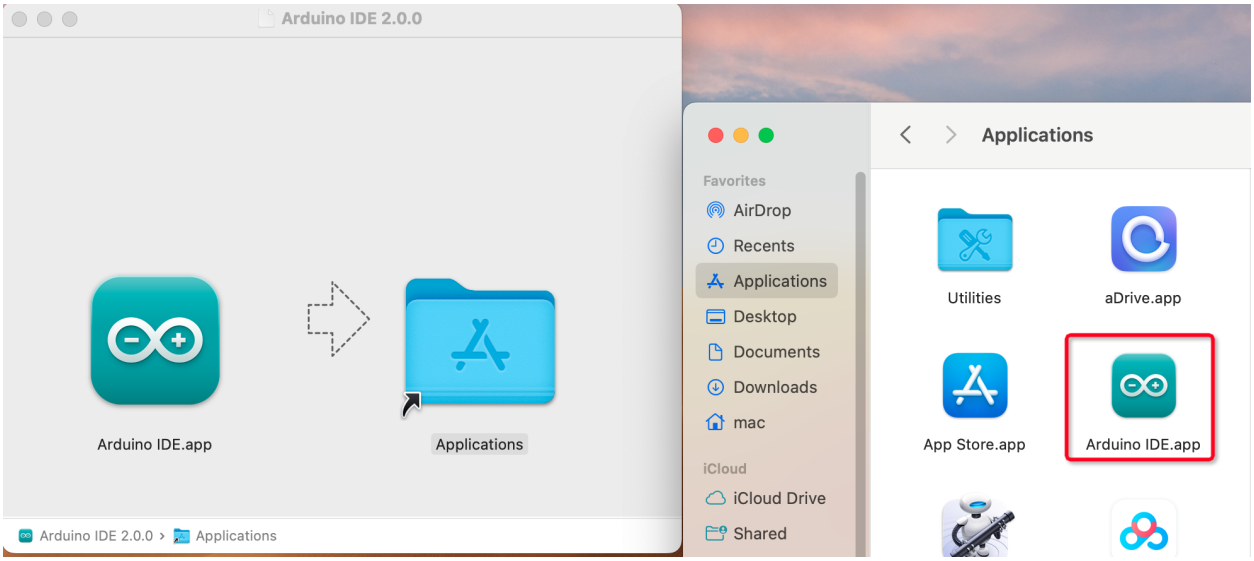


5. Then Finish.



macOS

Double click on the downloaded `arduino_ide_xxxx.dmg` file and follow the instructions to copy the **Arduino IDE.app** to the **Applications** folder, you will see the Arduino IDE installed successfully after a few seconds.

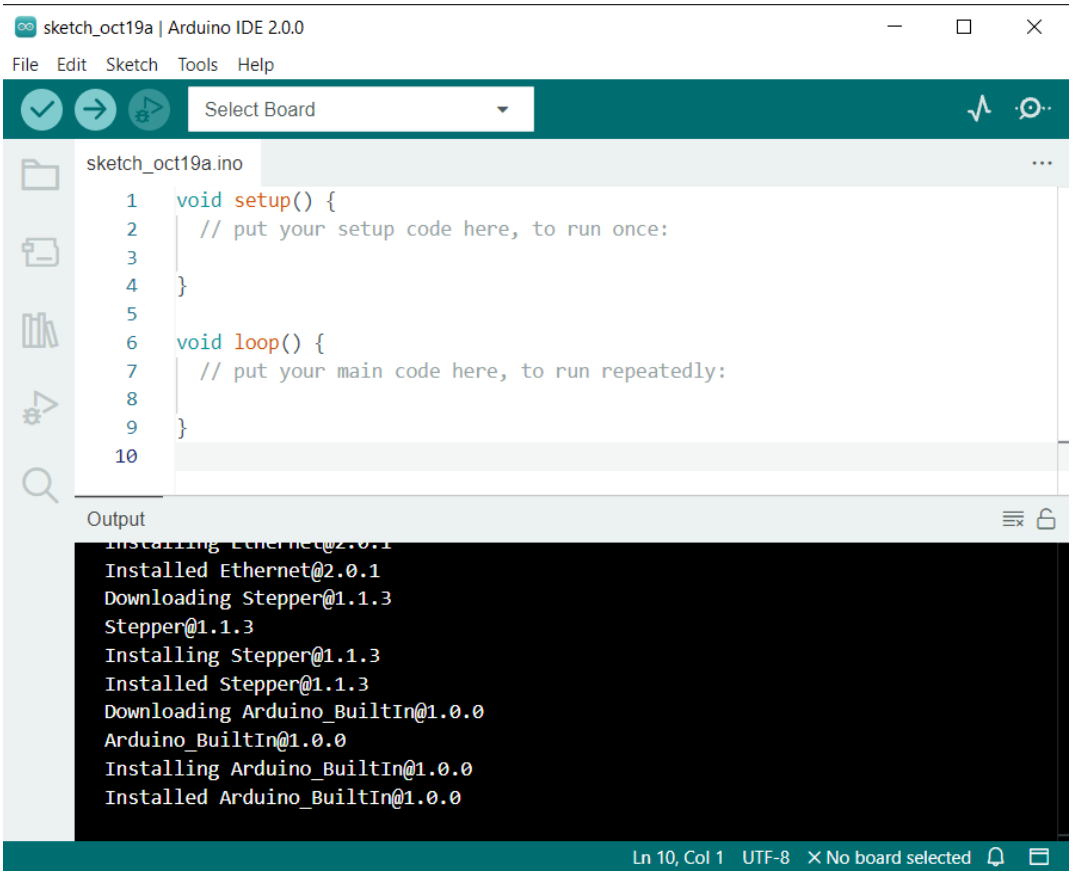


Linux

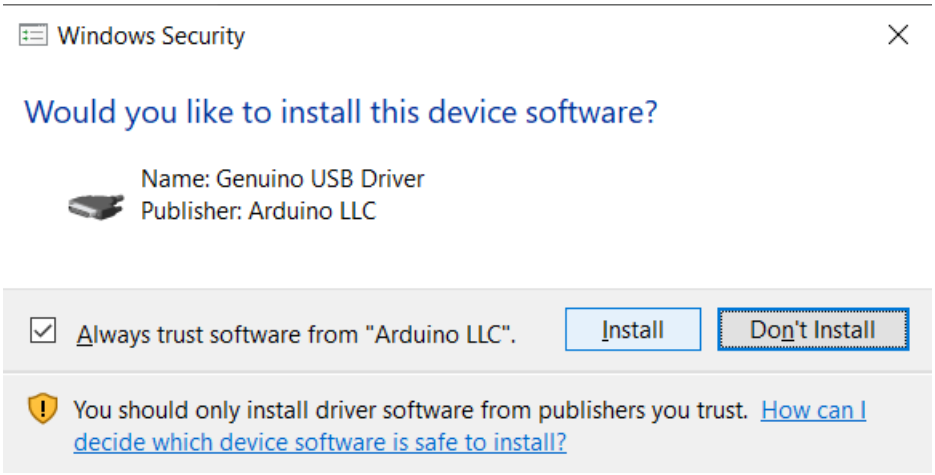
For the tutorial on installing the Arduino IDE 2.0 on a Linux system, please refer to: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started/ide-v2-downloading-and-installing#linux

6.1.4 Open the IDE

1. When you first open Arduino IDE 2.0, it automatically installs the Arduino AVR Boards, built-in libraries, and other required files.



2. In addition, your firewall or security center may pop up a few times asking you if you want to install some device driver. Please install all of them.



3. Now your Arduino IDE is ready!

Note: In the event that some installations didn't work due to network issues or other reasons, you can reopen the Arduino IDE and it will finish the rest of the installation. The Output window will not automatically open after all installations are complete unless you click Verify or Upload.

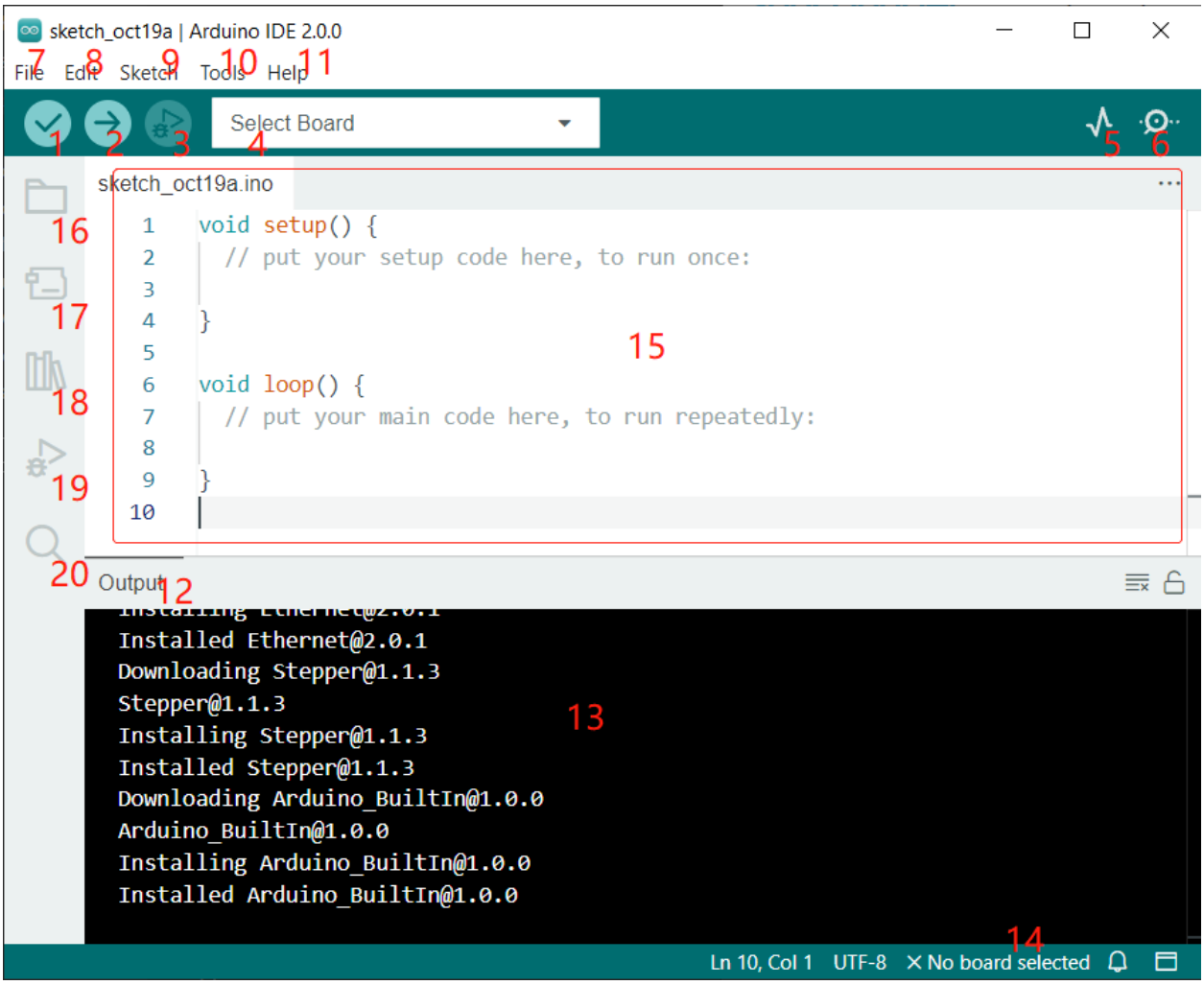
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.2 1.2 Introduce of Arduino IDE



1. **Verify**: Compile your code. Any syntax problem will be prompted with errors.

2. **Upload**: Upload the code to your board. When you click the button, the RX and TX LEDs on the board will flicker fast and won't stop until the upload is done.
3. **Debug**: For line-by-line error checking.
4. **Select Board**: Quick setup board and port.
5. **Serial Plotter**: Check the change of reading value.
6. **Serial Monitor**: Click the button and a window will appear. It receives the data sent from your control board. It is very useful for debugging.
7. **File**: Click the menu and a drop-down list will appear, including file creating, opening, saving, closing, some parameter configuring, etc.
8. **Edit**: Click the menu. On the drop-down list, there are some editing operations like **Cut**, **Copy**, **Paste**, **Find**, and so on, with their corresponding shortcuts.
9. **Sketch**: Includes operations like **Verify**, **Upload**, **Add** files, etc. More important function is **Include Library** - where you can add libraries.
10. **Tool**: Includes some tools - the most frequently used Board (the board you use) and Port (the port your board is at). Every time you want to upload the code, you need to select or check them.
11. **Help**: If you're a beginner, you may check the options under the menu and get the help you need, including operations in IDE, introduction information, troubleshooting, code explanation, etc.
12. **Output Bar**: Switch the output tab here.
13. **Output Window**: Print information.
14. **Board and Port**: Here you can preview the board and port selected for code upload. You can select them again by **Tools -> Board / Port** if any is incorrect.
15. The editing area of the IDE. You can write code here.
16. **Sketchbook**: For managing sketch files.
17. **Board Manager**: For managing board driver.
18. **Library Manager**: For managing your library files.
19. **Debug**: Help debugging code.
20. **Search**: Search the codes from your sketches.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

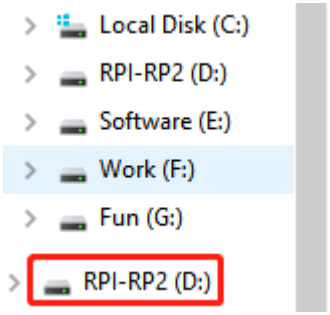
6.3 1.3 Setting Up the Raspberry Pi Pico W (Important)

6.3.1 1. Installing UF2 Firmware

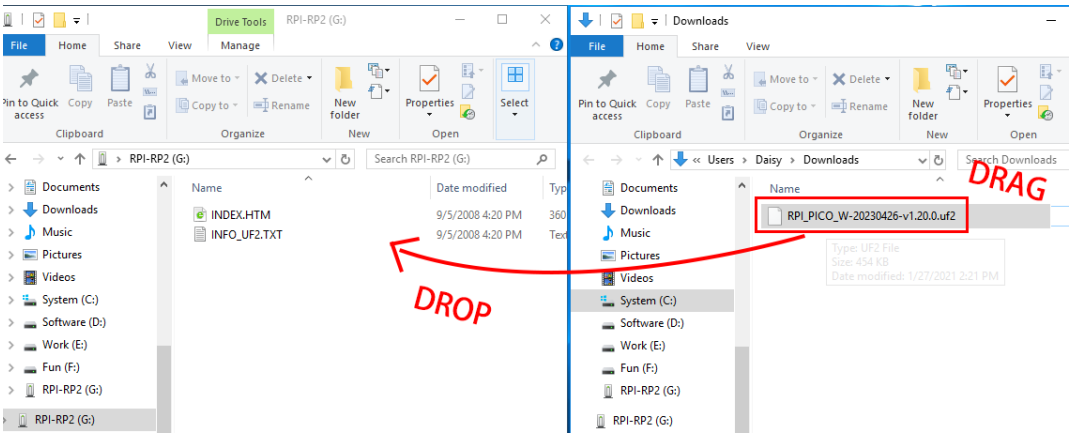
When you initially connect the Raspberry Pi Pico W or hold down the BOOTSEL button while inserting it, you'll see the device showing up as a drive without being assigned a COM port. This makes it impossible to upload code.

To fix this, you need to install UF2 firmware. This firmware supports MicroPython and is also compatible with the Arduino IDE.

1. Download the UF2 Firmware from the link below.
    - `Raspberry Pi Pico W UF2 Firmware`
2. Connect your Raspberry Pi Pico W to your computer using a Micro USB cable. Your Pico W will mount as a Mass Storage Device named **RPI-RP2**.



3. Drag and drop the downloaded UF2 firmware into the **RPI-RP2** drive.

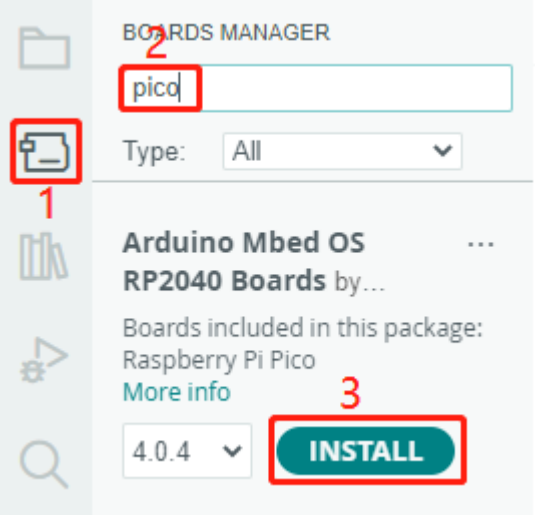


4. After this, the **RPI-RP2** drive will disappear, and you can proceed with the following steps.

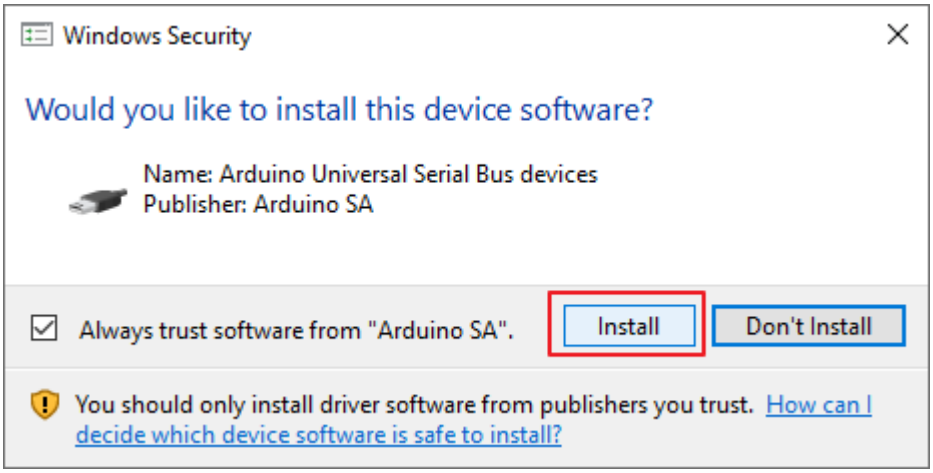
6.3.2 2. Installing the Board Package

To program the Raspberry Pi Pico W, you'll need to install the corresponding package in the Arduino IDE. Here's a step-by-step guide:

1. In the **Boards Manager** window, search for **pico**. Click the **Install** button to commence the installation. This will install the **Arduino Mbed OS RP2040 Boards** package, which includes support for the Raspberry Pi Pico W.



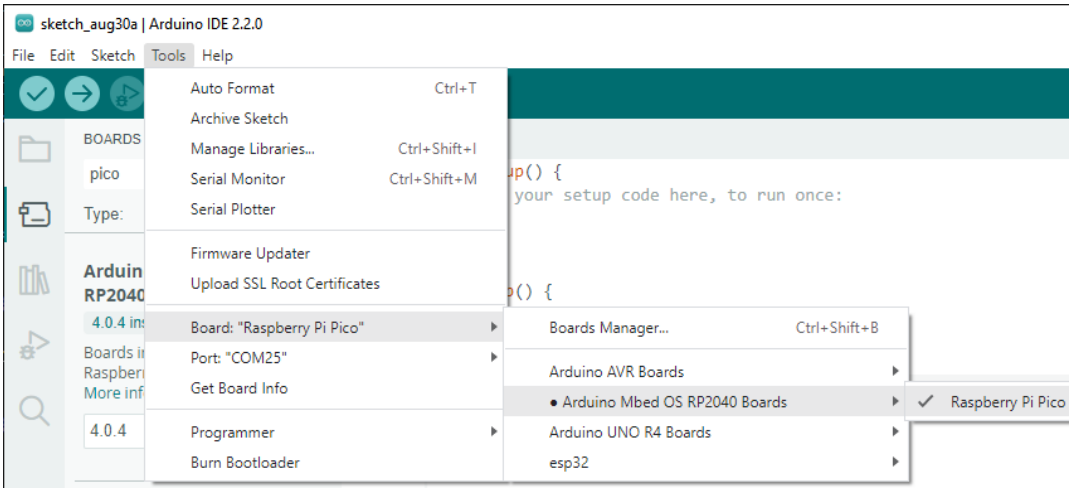
2. During the process, a few pop-up prompts will appear for the installation of specific device drivers. Select **"Install"**.



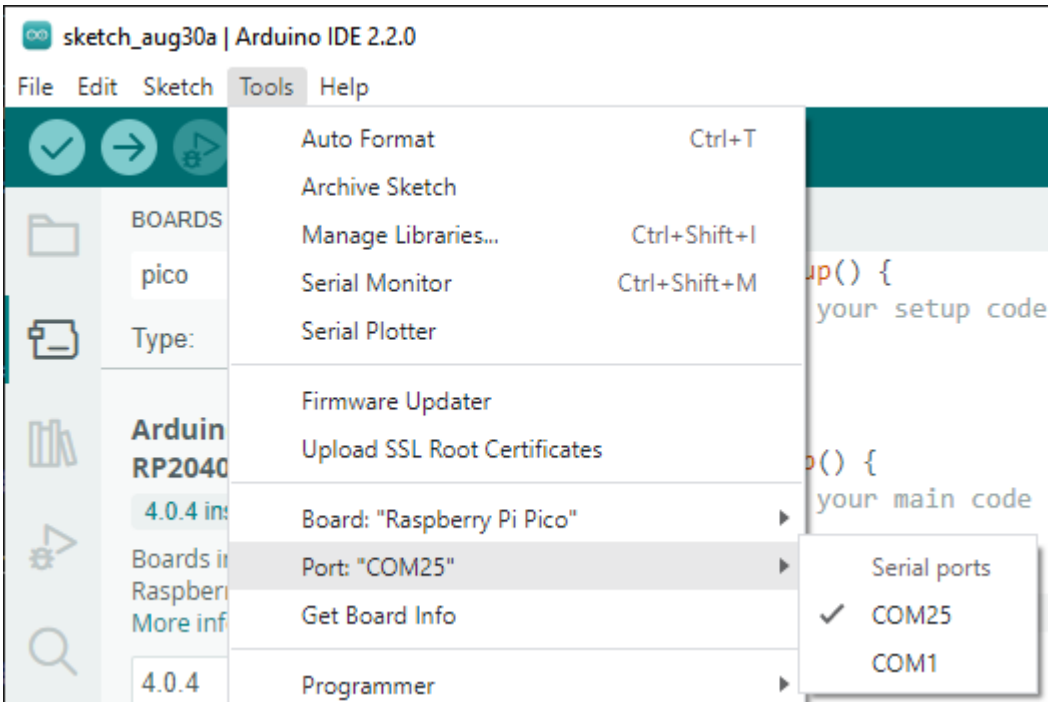
3. Afterwards, there will be a notification indicating that the installation is complete.

6.3.3 3. Selecting the Board and Port

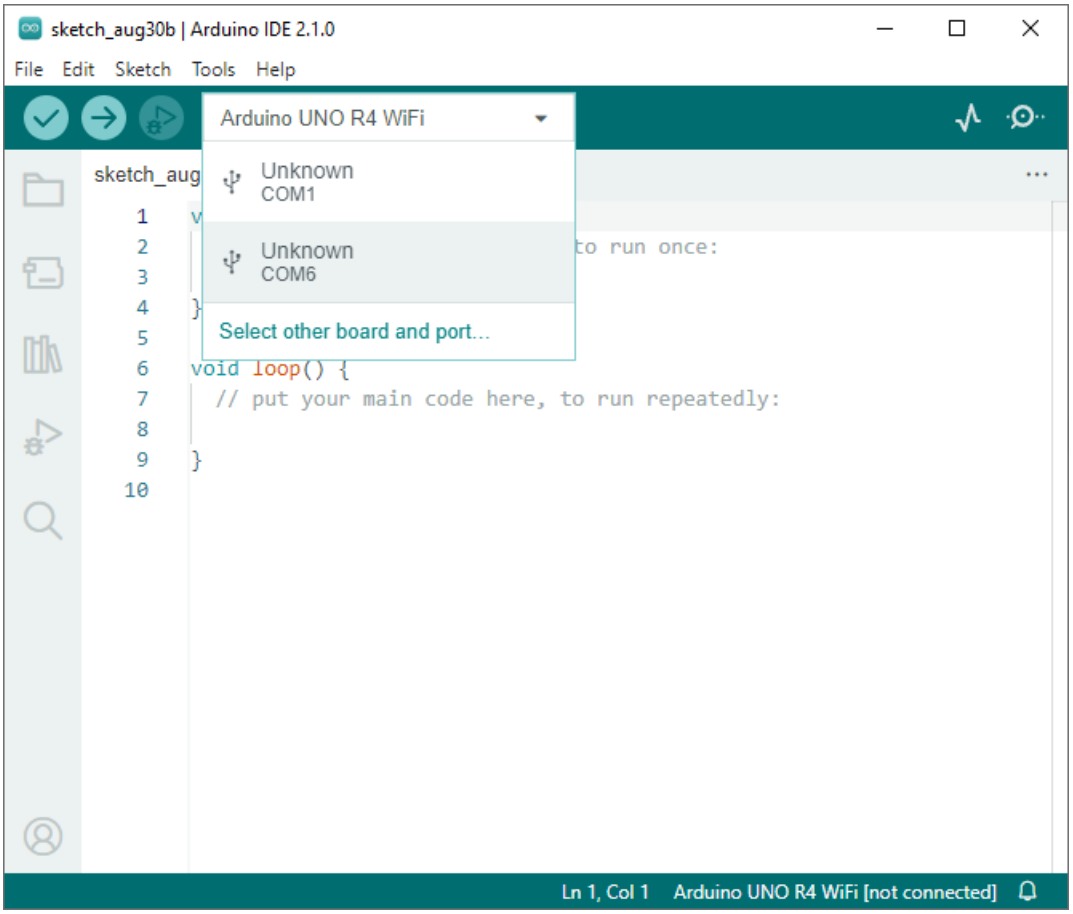
1. To select the appropriate board, navigate to **Tools** -> **Board** -> **Arduino Mbed OS RP2040 Boards** -> **Raspberry Pi Pico**.



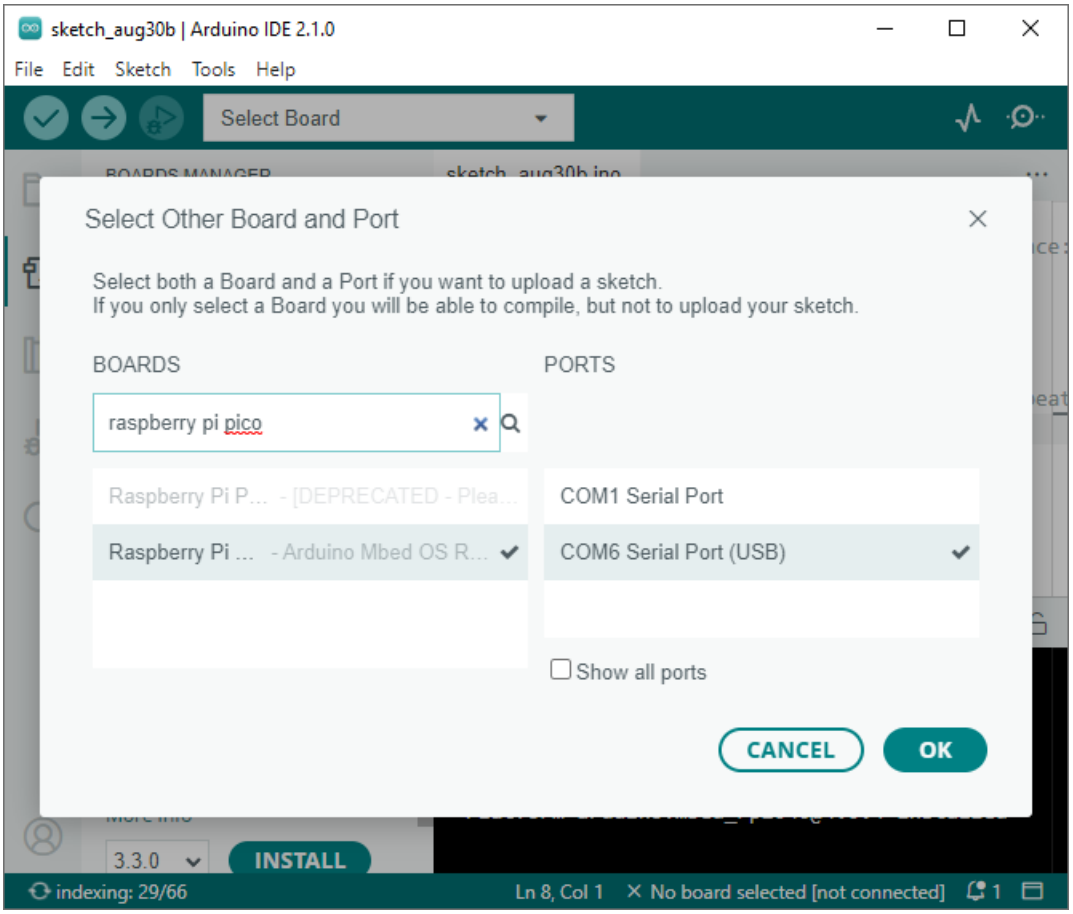
2. If your Raspberry Pi Pico W is connected to the computer, set the right port by navigating to **Tools** -> **Port**.



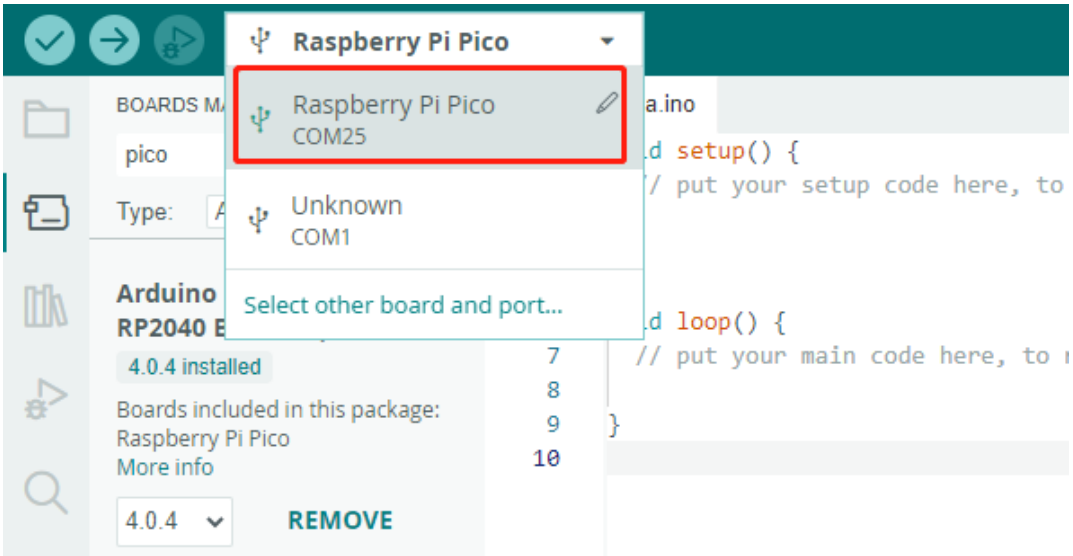
3. Arduino 2.0 offers a new quick-select feature. For the Raspberry Pi Pico W, which is typically not auto-recognized, click **Select other board and port**.



4. Type **Raspberry Pi Pico** into the search bar, select it when it shows up, choose the appropriate port, and click **OK**.



5. You can easily reselect it later through this quick access window.

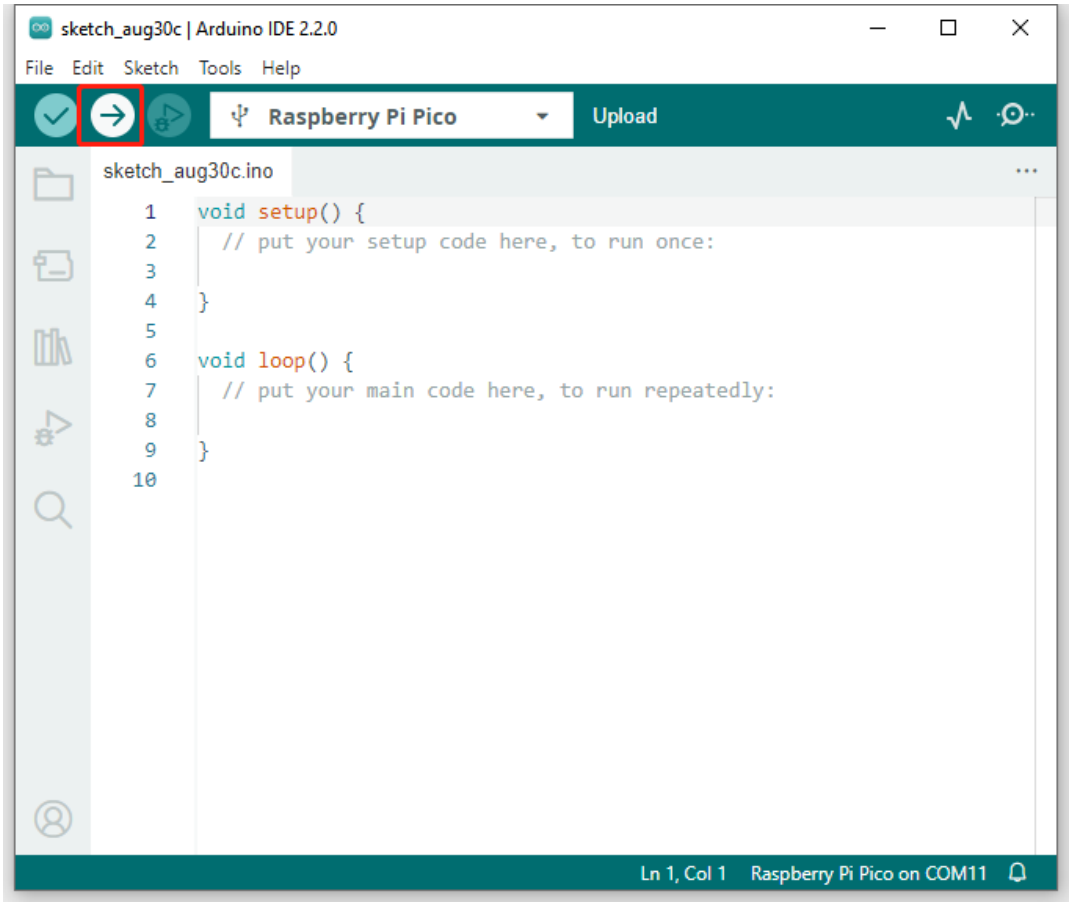


6. Either of these methods will enable you to set the correct board and port. You're now all set to upload code to the Raspberry Pi Pico W.

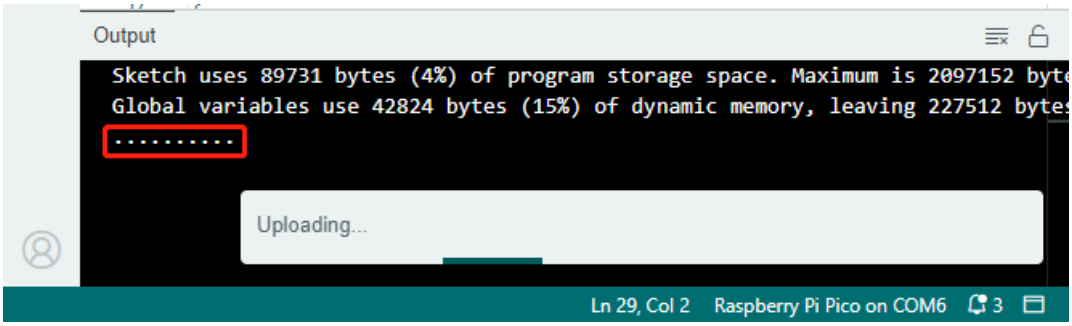
6.3.4 4. Uploading Code

Now let's dive into how to upload code to your Raspberry Pi Pico W.

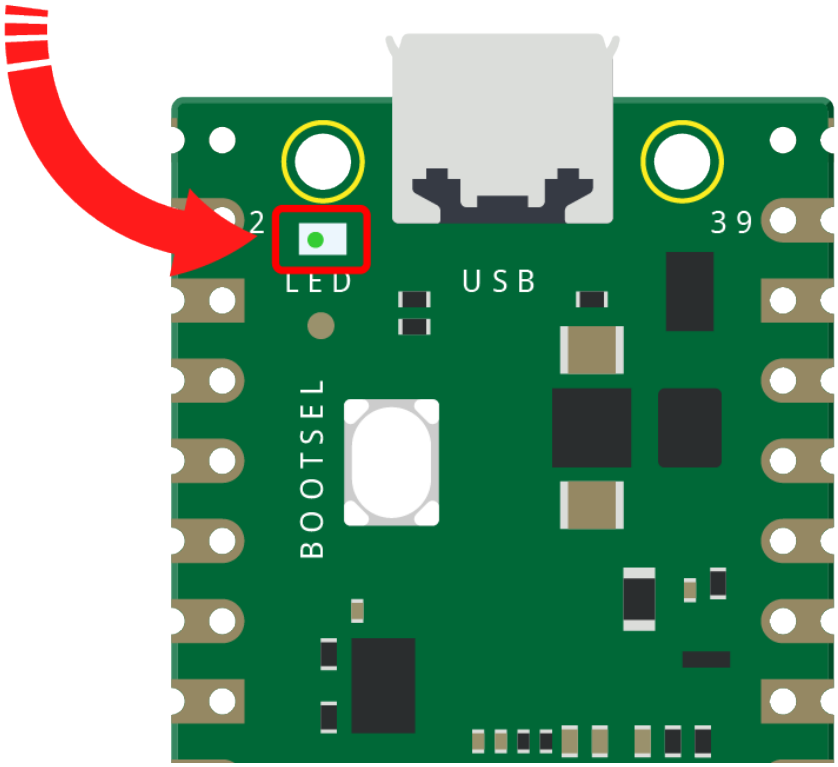
1. Open any `.ino` file or use the empty sketch currently displayed. Then, click the **Upload** button.



2. Wait for the uploading message to appear, as shown below.



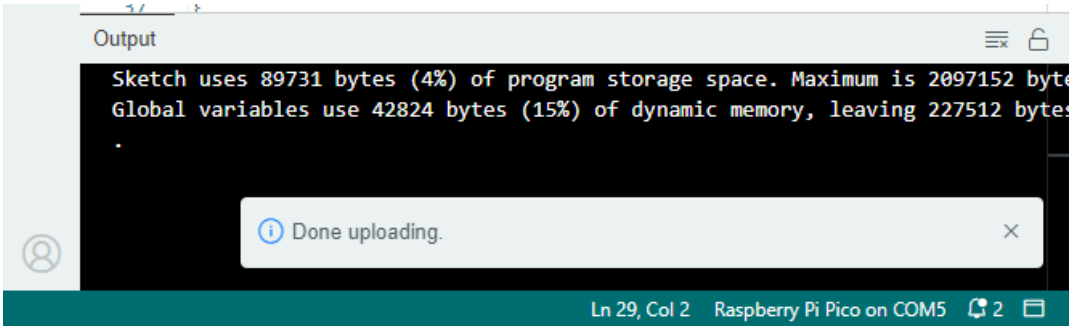
3. Hold down the **BOOTSEL** button, quickly unplug your Raspberry Pi Pico W, and plug it back in.



Note:

- This step is crucial, especially for first-time users on the Arduino IDE. Skipping this step will result in a failed upload.
- Once you successfully upload the code this time, your Pico W will be recognized by the computer. For future uses, simply plug it into the computer.

4. A prompt indicating successful upload will appear.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.4 1.4 Install libraries (Important)

Download the Code

Download the relevant code from the link below.

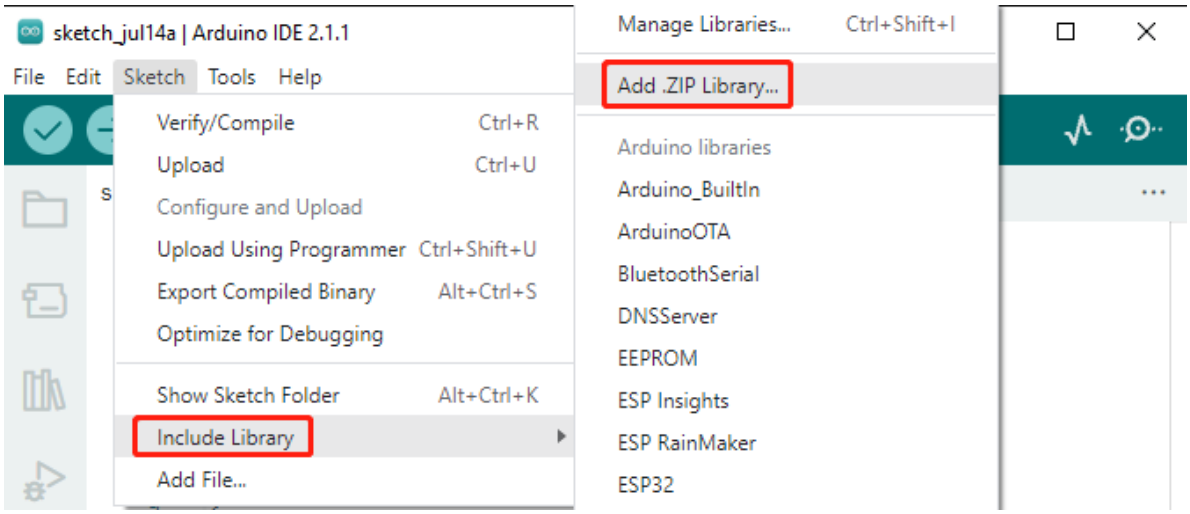
- `SunFounder Kepler Kit Example`
- Or check out the code at Kepler Kit - GitHub

6.4.1 Add libraries

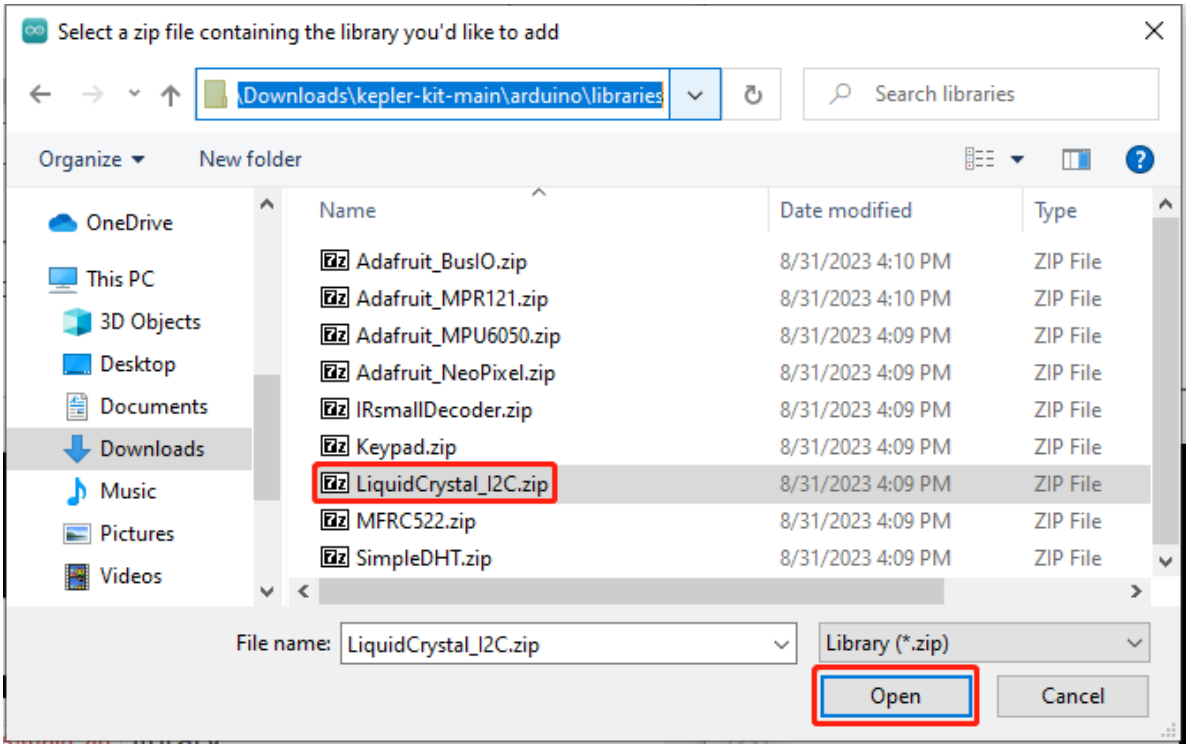
A library, gathering some function definitions and header files, usually contains two files: .h (header file, including function statement, Macro definition, constructor definition, etc.) and .cpp (execution file, with function implementation, variable definition, and so on). When you need to use a function in some library, you just need to add a header file (e.g. #include <dht.h>), and then call that function. This can make your code more concise. If you don't want to use the library, you can also write that function definition directly. Though as a result, the code will be long and inconvenient to read.

Some libraries are already built in the Arduino IDE, when some others may need to be added. So now let's see how to add them.

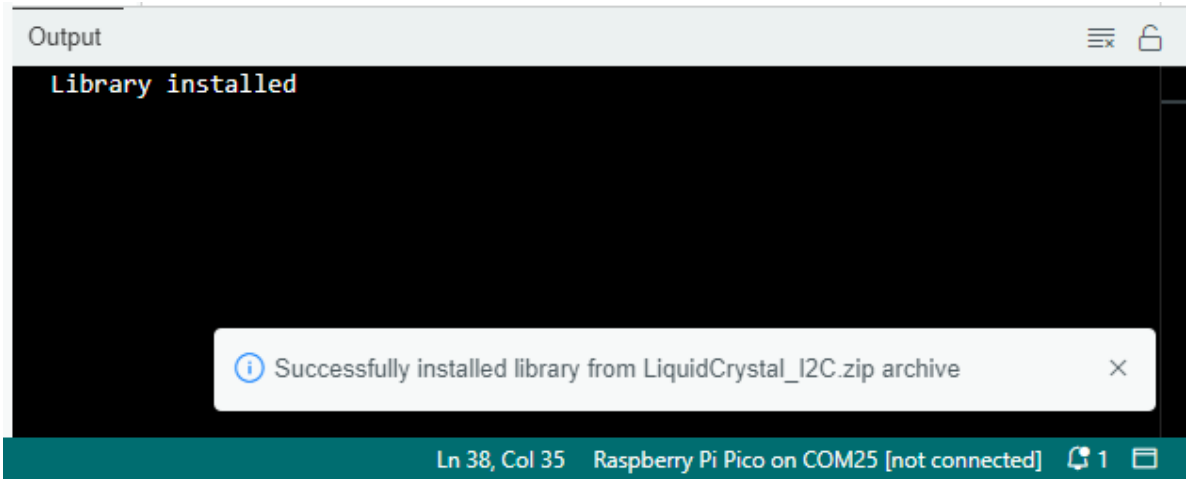
1. Open the Arduino IDE and go to **Sketch** -> **Include Library** -> **Add .ZIP Library**.



2. Navigate to the directory where the library files are located, such as the `kepler-kit-main\arduino\libraries` folder, and select the desired library file, like `LiquidCrystal_I2C.zip`. Then, click **Open**.



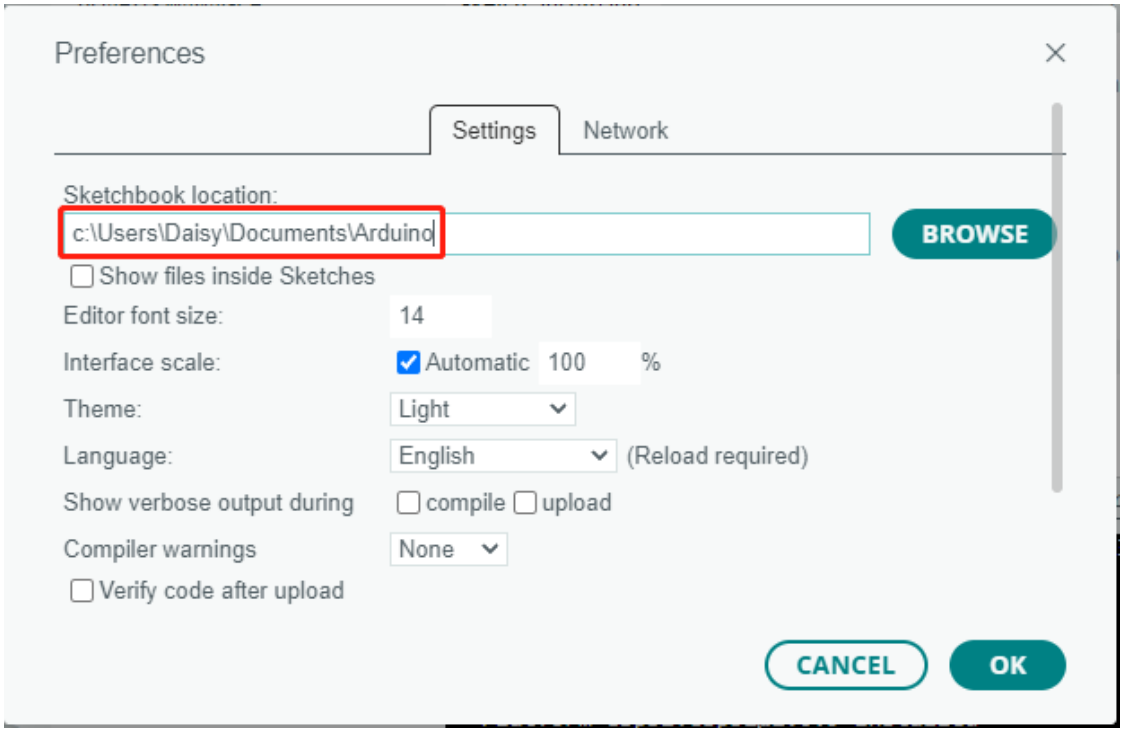
3. After a short while, you will receive a notification indicating a successful installation.



4. Repeat the same process to add the other libraries.

Note: The libraries installed can be found in the default library directory of the Arduino IDE, which is usually located at `C:\Users\xxx\Documents\Arduino\libraries`.

If your library directory is different, you can check it by going to **File** -> **Preferences**.



2. Output & input

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.5 2.1 - Hello, LED!

Just as printing "Hello, world!" is the first step in learning to program, using a program to drive an LED is the traditional introduction to learning physical programming.

- *LED*

Required Components

In this project, we need the following components.

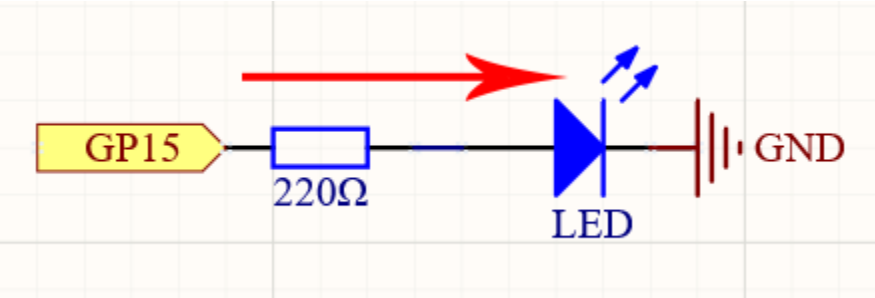
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

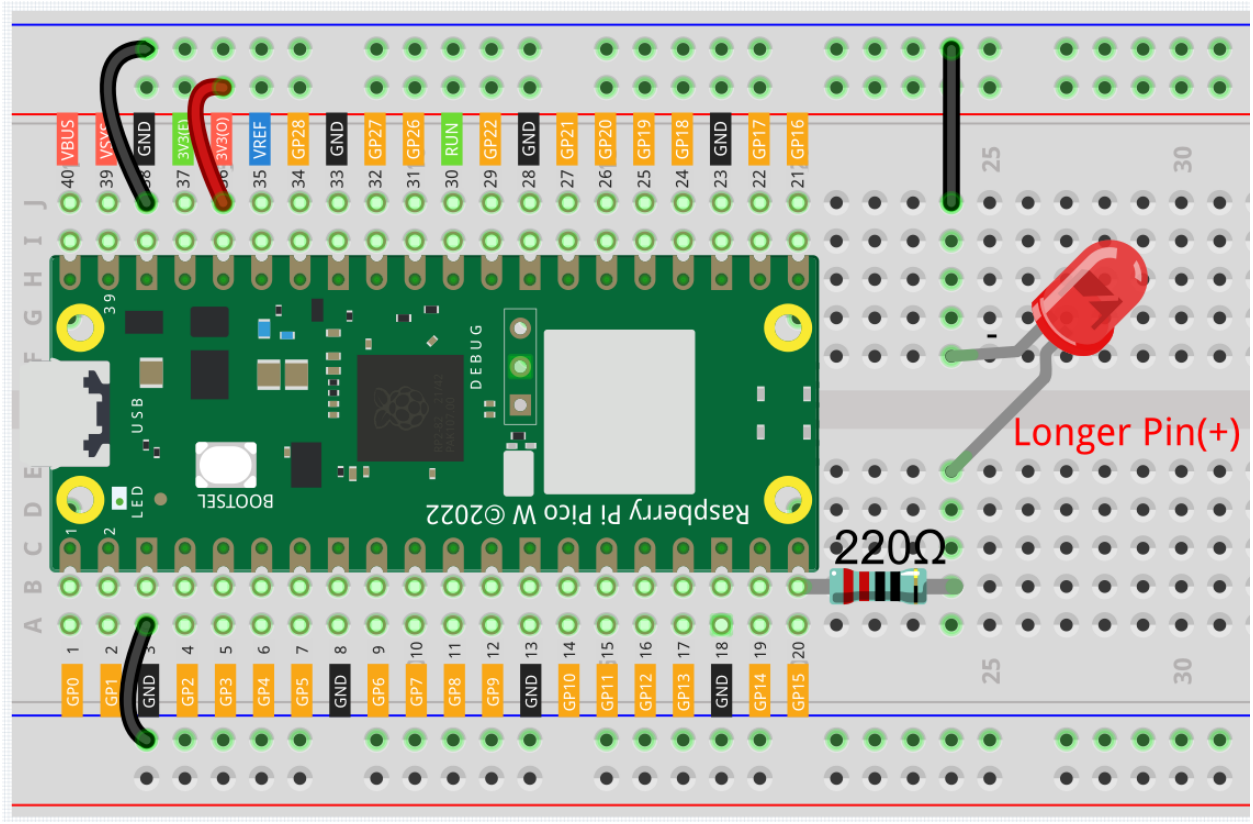
SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(220)	
6	*LED*	1	

Schematic



The principle of this circuit is simple and the current direction is shown in the figure. When GP15 outputs high level(3.3v), the LED will light up after the 220ohm current limiting resistor. When GP15 outputs low level (0v), the LED will turn off.

Wiring



Let us follow the direction of the current to build the circuit!

1. Here we use the electrical signal from the GP15 pin of the Pico W board to make the LED work, and the circuit starts from here.
2. The current needs to pass through a 220 ohm resistor (used to protect the LED). Insert one end (either end) of the resistor into the same row as the Pico W GP15 pin (row 20 in my circuit), and insert the other end into the free row of the breadboard (row 24 in my circuit).
3. Pick up the LED, you will see that one of its leads is longer than the other. Insert the longer lead into the same row as the end of the resistor, and connect the shorter lead across the middle gap of the breadboard to the same row.
4. Insert the male-to-male (M2M) jumper wire into the same row as the LED short pin, and then connect it to the negative power bus of the breadboard.
5. Use a jumper to connect the negative power bus to the GND pin of Pico W.

Code

Note:

- You can open the file `2.1_hello_led.ino` under the path of `kepler-kit-main/arduino/2.1_hello_led`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

After the code runs, you will see the LED blinking.

How it works?

Here, we connect the LED to the digital pin 15, so we need to declare an int variable called ledpin at the beginning of the program and assign a value of 15.

```
const int ledPin = 15;
```

Now, initialize the pin in the `setup()` function, where you need to initialize the pin to `OUTPUT` mode.

```
void setup() {
    pinMode(ledPin, OUTPUT);
}
```

In `loop()`, `digitalWrite()` is used to provide 3.3V high level signal for ledpin, which will cause voltage difference between LED pins and light LED up.

```
digitalWrite(ledPin, HIGH);
```

If the level signal is changed to LOW, the ledPin's signal will be returned to 0 V to turn LED off.

```
digitalWrite(ledPin, LOW);
```

An interval between on and off is required to allow people to see the change, so we use a `delay(1000)` code to let the controller do nothing for 1000 ms.

```
delay(1000);
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

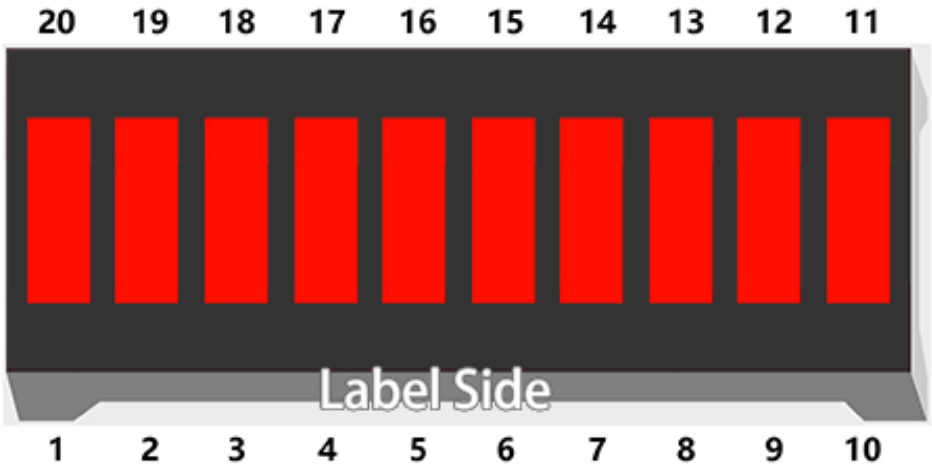
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.6 2.2 - Display the Level

The first project is simply to make the LED blink. In this project let's use the LED Bar Graph, which is made up of 10 LEDs packaged into a plastic case, generally used to display power or volume levels.



- *LED Bar Graph*

Required Components

In this project, we need the following components.

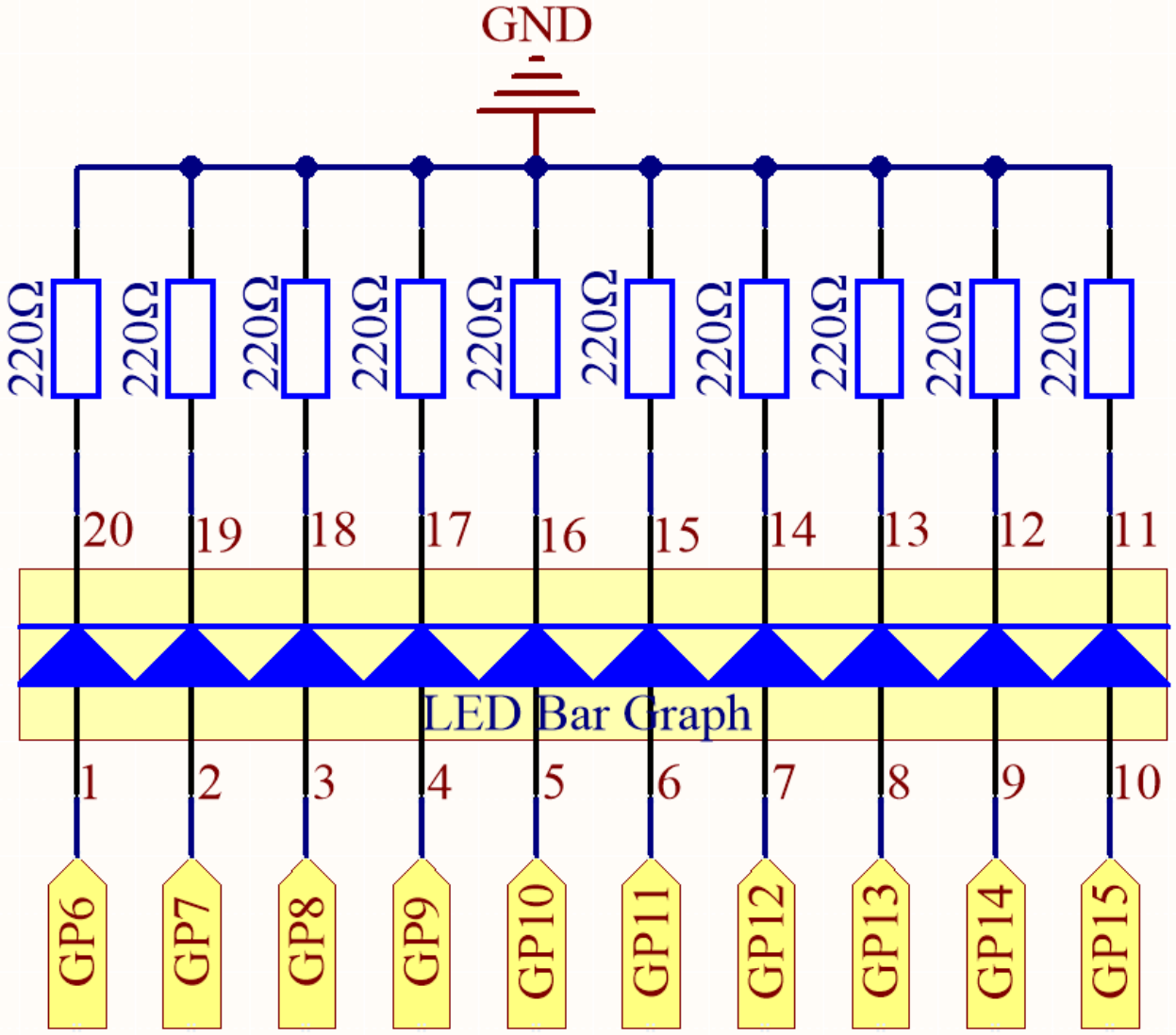
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

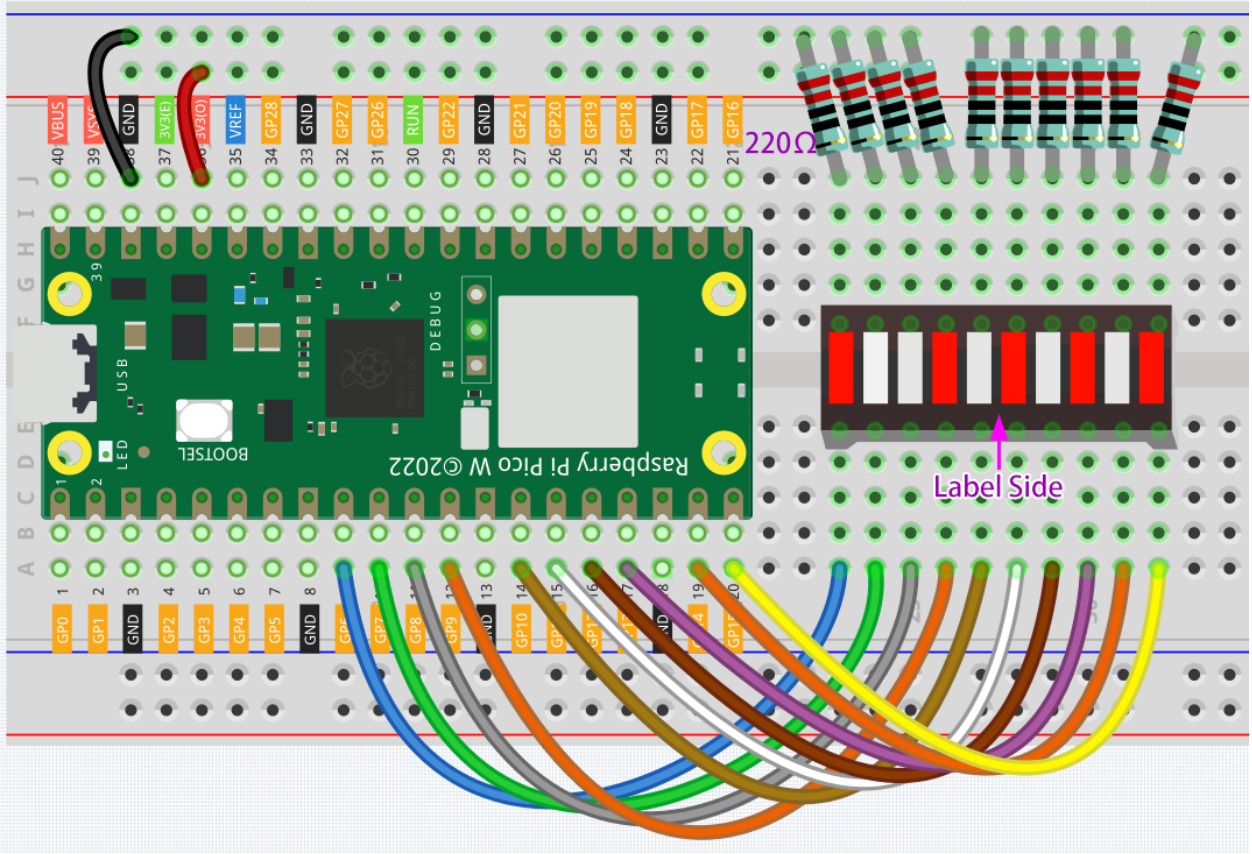
SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	10(220)	
6	*LED Bar Graph*	1	

Schematic



The LED Bar Graph contains 10 LEDs, each of which is individually controllable. Here, the anode of each of the 10 LEDs is connected to GP6~GP15, and the cathode is connected to a 220ohm resistor, and then to GND.

Wiring



Code

Note:

- You can open the file `2.2_display_the_level.ino` under the path of `kepler-kit-main/arduino/2.2_display_the_level`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

When the program is running, you will see the LEDs on the LED Bar Graph light up and then turn off in sequence.

How it works?

Each of the ten LEDs on the LED Bar needs to be controlled by a pin, which means that we define these ten pins.

The codes in `setup()` use the for loop to initialize pins 6~15 to output mode in turn.

```
for(int i=6;i<=15;i++)
{
    pinMode(i,OUTPUT);
}
```

The for loop is used in `loop()` to make the LED flash(turn on 0.5s, then turn off 0.5s) in sequence.

```
for(int i=6;i<=15;i++)
{
```

(continues on next page)

(continued from previous page)

```
    digitalWrite(i,HIGH);
    delay(500);
    digitalWrite(i,LOW);
    delay(500);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.7 2.3 - Fading LED

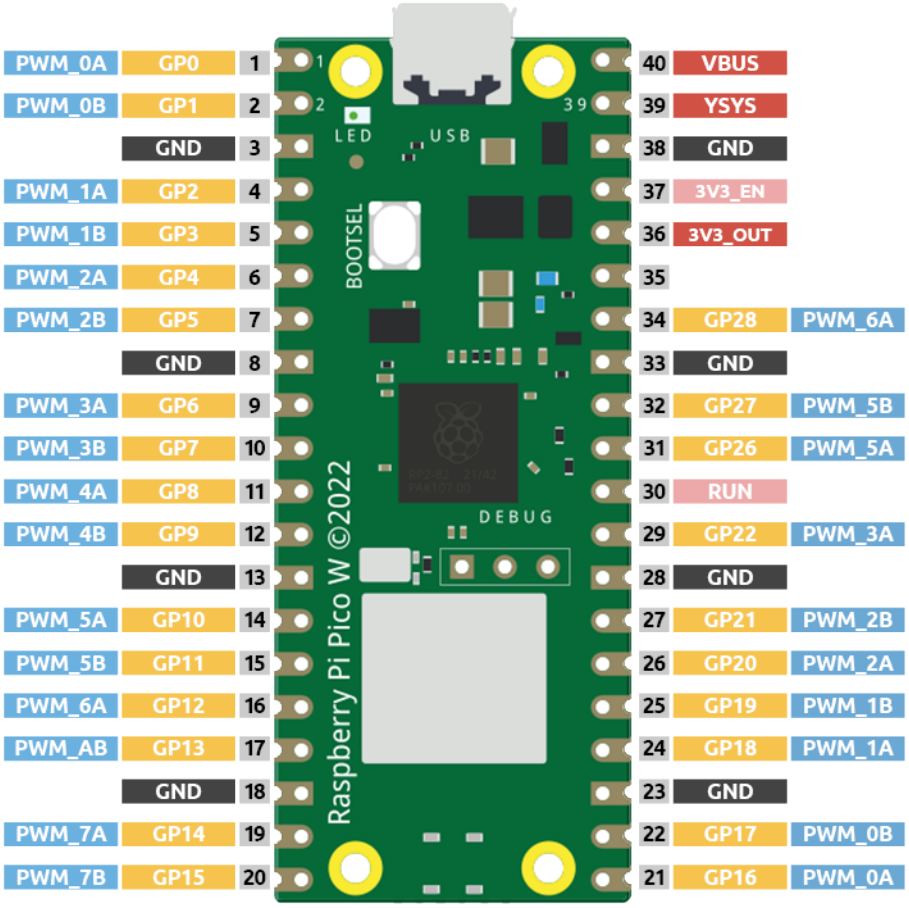
So far, we have used only two output signals: high level and low level (or called 1 & 0, ON & OFF), which is called digital output. However, in actual use, many devices do not simply ON/OFF to work, for example, adjusting the speed of the motor, adjusting the brightness of the desk lamp, and so on. In the past, a slider that can adjust the resistance was used to achieve this goal, but this is always unreliable and inefficient. Therefore, Pulse width modulation (PWM) has emerged as a feasible solution to such complex problems.

A digital output composed of a high level and a low level is called a pulse. The pulse width of these pins can be adjusted by changing the ON/OFF speed.

Simply put, when we are in a short period (such as 20ms, most people's visual retention time), Let the LED turn on, turn off, and turn on again, we won't see it has been turned off, but the brightness of the light will be slightly weaker. During this period, the more time the LED is turned on, the higher the brightness of the LED. In other words, in the cycle, the wider the pulse, the greater the "electric signal strength" output by the microcontroller. This is how PWM controls LED brightness (or motor speed).

- Pulse-width modulation - Wikipedia

There are some points to pay attention to when Pico W uses PWM. Let's take a look at this picture.



Each GPIO pin of Pico W supports PWM, but it actually has a total of 16 independent PWM outputs (instead of 30), distributed between GP0 to GP15 on the left, and the PWM output of the right GPIO is equivalent to the left copy.

What we need to pay attention to is to avoid setting the same PWM channel for different purposes during programming. (For example, GP0 and GP16 are both PWM_0A)

After understanding this knowledge, let us try to achieve the effect of Fading LED.

- *LED*

Required Components

In this project, we need the following components.

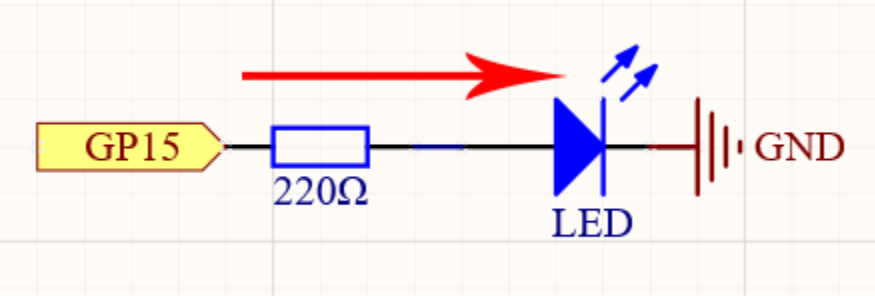
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

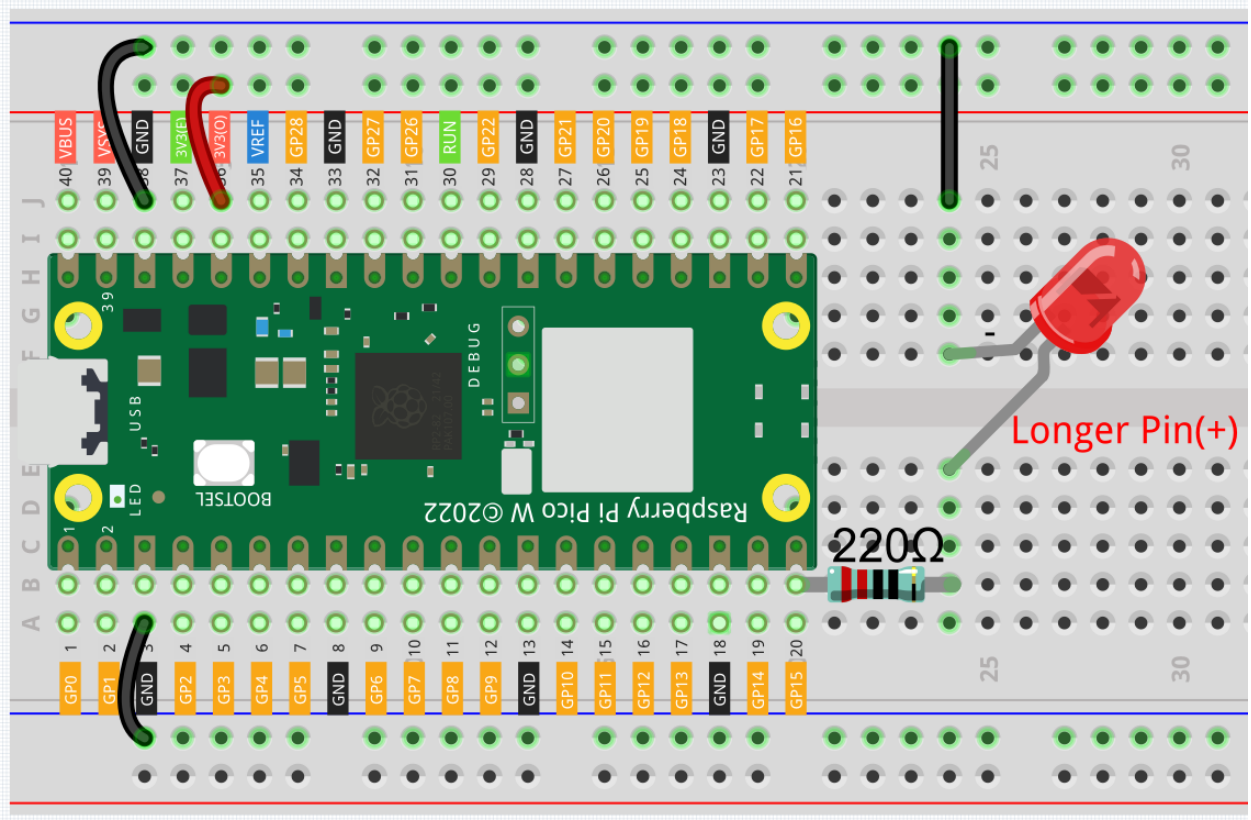
SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(220)	
6	*LED*	1	

Schematic



This project is the same circuit as the first project *2.1 - Hello, LED!*, but the signal type is different. The first project is to output digital high and low levels (0&1) directly from GP15 to make the LEDs light up or turn off, this project is to output PWM signal from GP15 to control the brightness of the LED.

Wiring



Code

Note:

- You can open the file `2.3_fading_led.ino` under the path of `kepler-kit-main/arduino/2.3_fading_led`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

The LED will gradually become brighter as the program runs.

How it works?

Declare pin 15 as ledPin.

```
const int ledPin = 15;
```

`analogWrite()` in `loop()` assigns ledPin an analog value (PWM wave) between 0 and 255 to change the brightness of LED.

```
analogWrite(ledPin, value);
```

Using a for loop, the value of `analogWrite()` can be changed step by step between the minimum value (0) and the maximum value (255).

```
for (int value = 0 ; value <= 255; value += 5) {
    analogWrite(ledPin, value);
}
```

In order to see the experimental phenomenon clearly, a `delay(30)` needs to be added to the for cycle to control the brightness change time.

```
for (int value = 0 ; value <= 255; value += 5) {
    analogWrite(ledPin, value);
    delay(30);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.8 2.4 - Colorful Light

As we know, light can be superimposed. For example, mix blue light and green light give cyan light, red light and green light give yellow light. This is called "The additive method of color mixing".

- Additive color - Wikipedia

Based on this method, we can use the three primary colors to mix the visible light of any color according to different specific gravity. For example, orange can be produced by more red and less green.

In this chapter, we will use RGB LED to explore the mystery of additive color mixing!

RGB LED is equivalent to encapsulating Red LED, Green LED, Blue LED under one lamp cap, and the three LEDs share one cathode pin. Since the electric signal is provided for each anode pin, the light of the corresponding color can be displayed. By changing the electrical signal intensity of each anode, it can be made to produce various colors.

- *RGB LED*

Required Components

In this project, we need the following components.

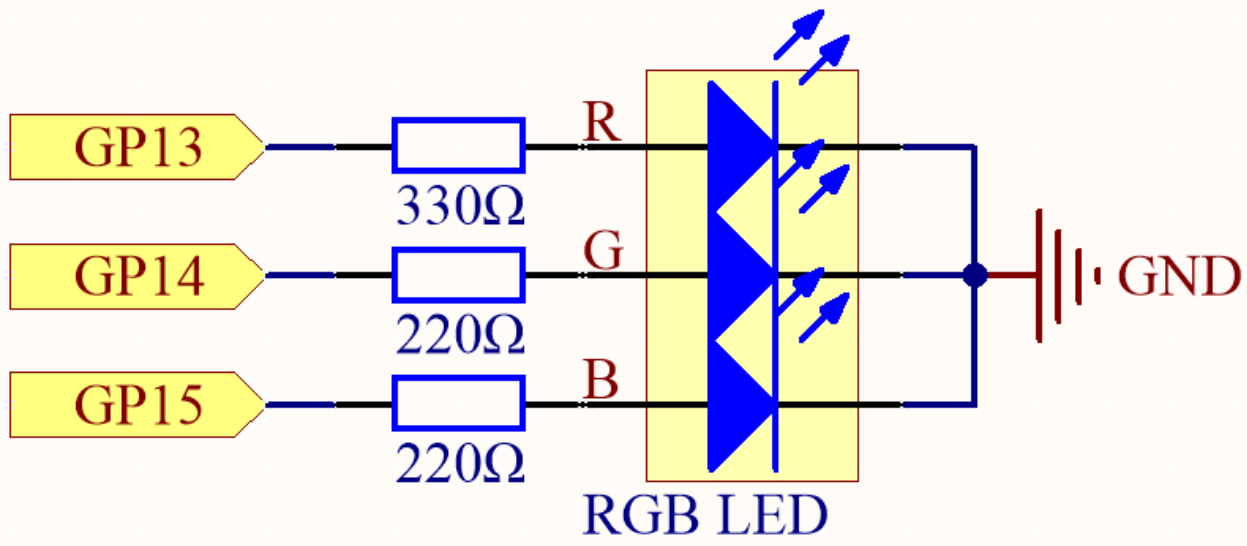
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

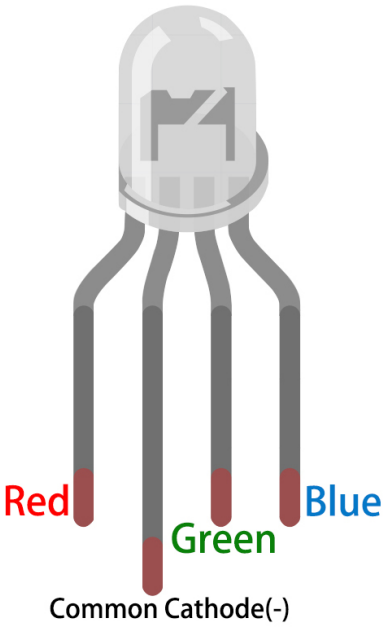
SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	3(1-330, 2-220)	
6	*RGB LED*	1	

Schematic

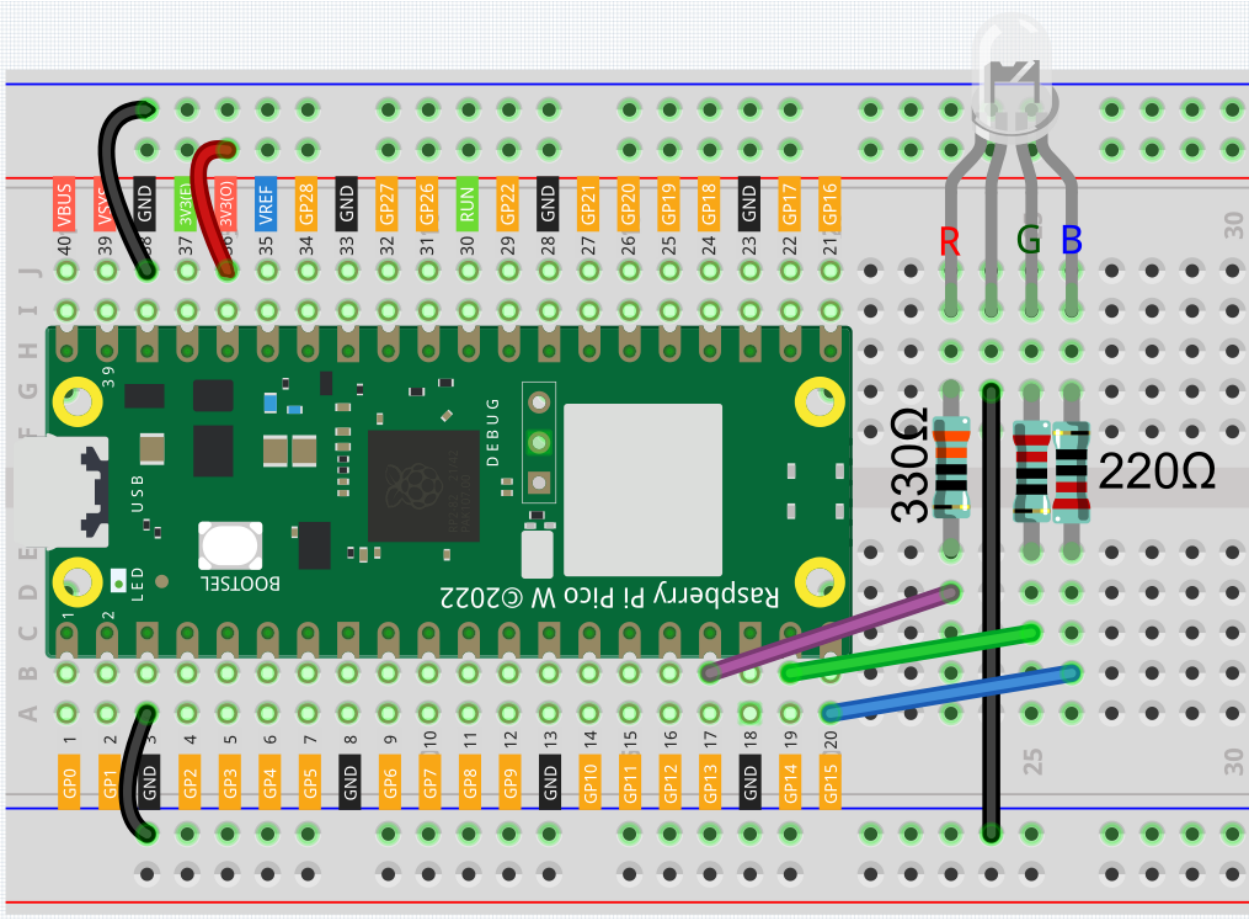


The PWM pins GP13, GP14 and GP15 control the Red, Green and Blue pins of the RGB LED respectively, and connect the common cathode pin to GND. This allows the RGB LED to display a specific color by superimposing light on these pins with different PWM values.

Wiring



An RGB LED has 4 pins: the longest pin is the common cathode pin, which is usually connected to GND, the left pin next to the longest pin is Red, and the 2 pins on the right are Green and Blue.

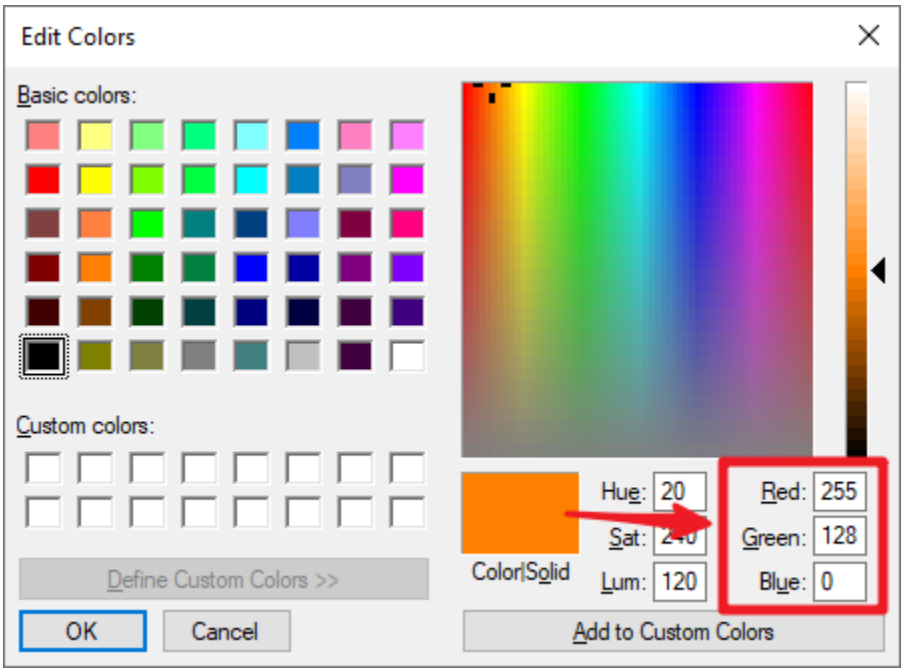


Code

Here, we can choose our favorite color in drawing software (such as paint) and display it with RGB LED.

Note:

- You can open the file `2.4_colorful_light.ino` under the path of `kepler-kit-main/arduino/2.4_colorful_light`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.



Write the RGB value into `color_set()`, you will be able to see the RGB light up the colors you want.

How it works?

In this example, the function used to assign values to the three pins of RGB is packaged in an independent subfunction `color()`.

```
void color (unsigned char red, unsigned char green, unsigned char blue)
{
    analogWrite(redPin, red);
    analogWrite(greenPin, green);
    analogWrite(bluePin, blue);
}
```

In `loop()`, RGB value works as an input argument to call the function `color()` to realize that the RGB can emit different colors.

```
void loop()
{
    color(255, 0, 0); //  red
    delay(1000);
    color(0,255, 0); //  green
    delay(1000);
```

(continues on next page)

(continued from previous page)

```
    color(0, 0, 255); //  blue
    delay(1000);
}
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.9 2.5 - Reading Button Value

From the name of GPIO (General-purpose input/output), we can see that these pins have both input and output functions. In the previous lessons, we used the output function, in this chapter we will use the input function to input read the button value.

- *Button*

Required Components

In this project, we need the following components.

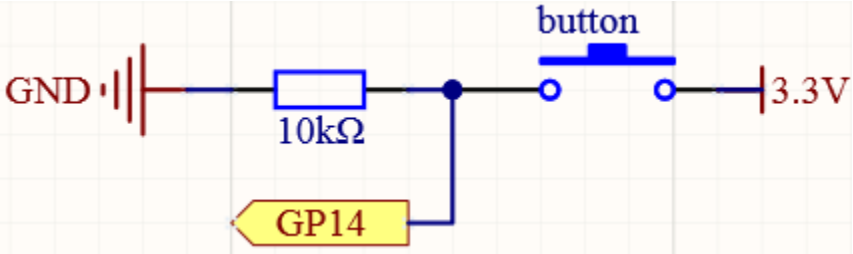
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

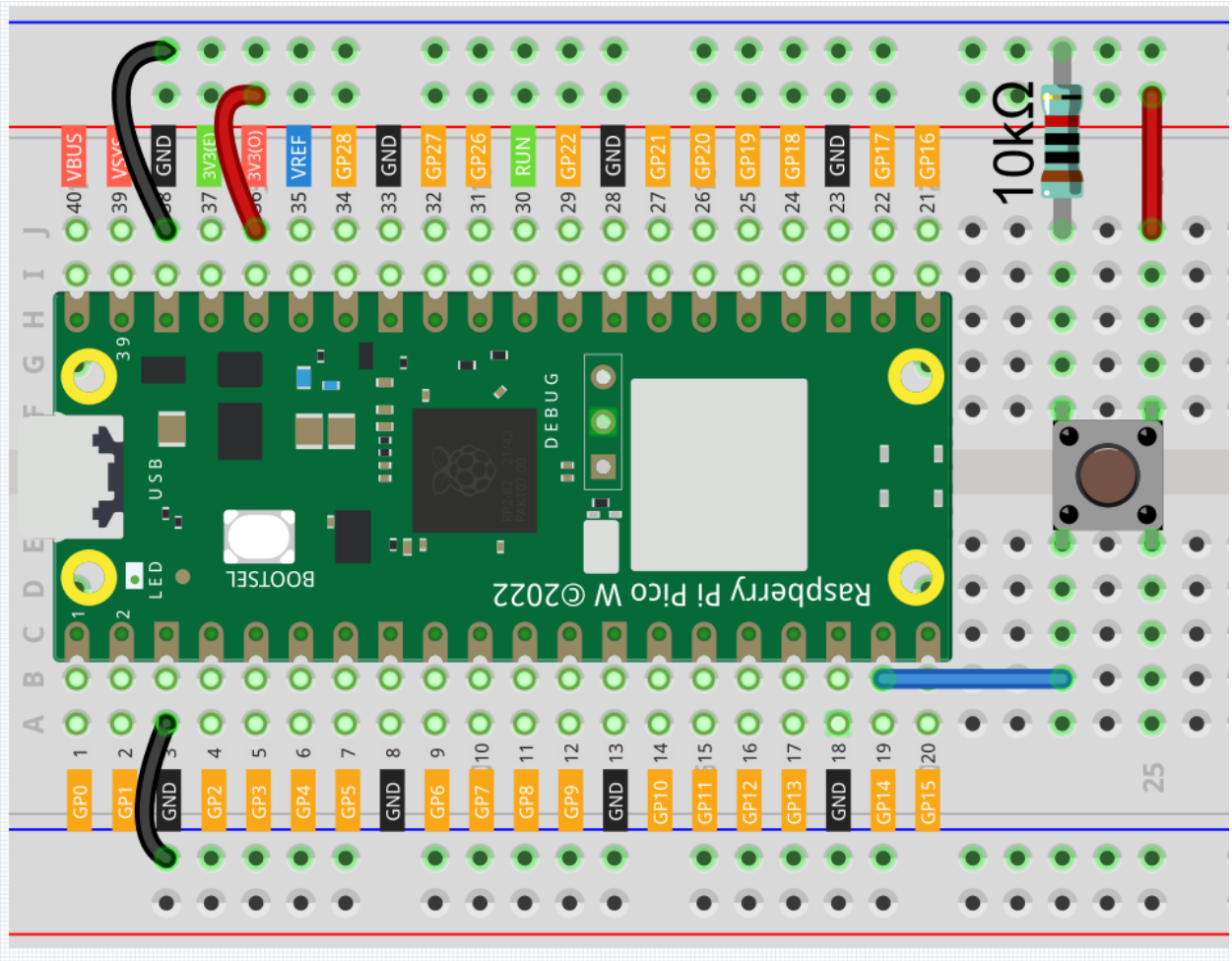
SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Button*	1	

Schematic



One side of the button pin is connected to 3.3v, and the other side pin is connected to GP14, so when the button is pressed, GP14 will be high. However, when the button is not pressed, GP14 is in a suspended state and may be high or low. In order to get a stable low level when the button is not pressed, GP14 needs to be reconnected to GND through a 10K pull-down resistor.

Wiring



Note: We can think of the four-legged button as an H-shaped button. Its left (right) two feet are connected, which means that after it straddles the central dividing line, it will connect the two half rows of the same row number together. (For example, in my circuit, E23 and F23 have been connected, as are E25 and F25).

Before the button is pressed, the left and right sides are independent of each other, and current cannot flow from one

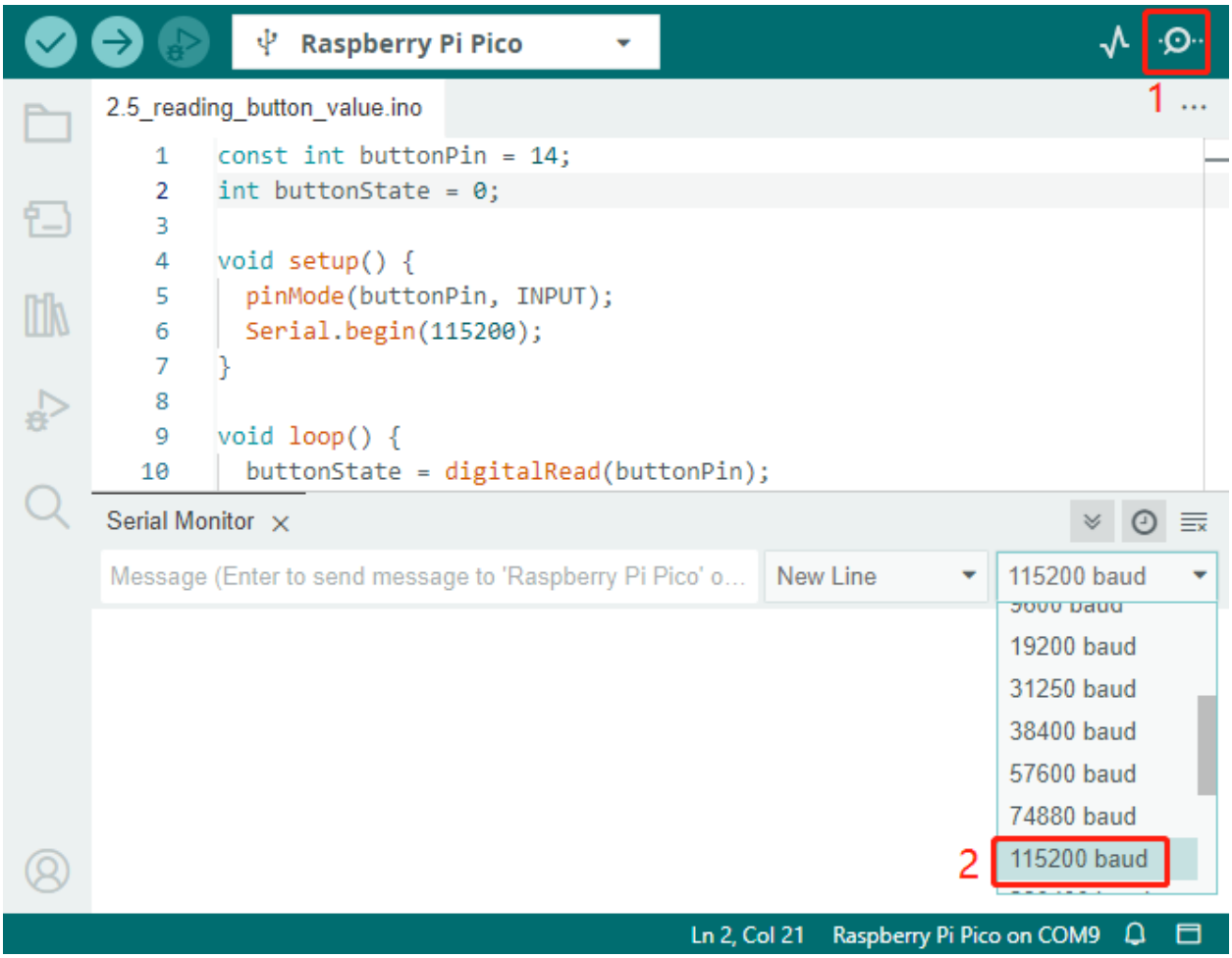
side to the other.

Code

Note:

- You can open the file `2.5_reading_button_value.ino` under the path of `kepler-kit-main/arduino/2.5_reading_button_value`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

After the code runs, Click the magnifying glass icon in the upper right corner of the Arduino IDE (Serial Monitor).



Now, when you press the button, the Serial Monitor will print "You pressed the button!".

How it works?

To enable Serial Monitor, you need to start serial communication in `setup()` and set the datarate to 9600.

```
Serial.begin(115200);
```

- Serial

For button, we need to set their mode to `INPUT` in order to be able to get their values.

```
pinMode(buttonPin, INPUT);
```

Read the status of the `buttonPin` in `loop()` and assign it to the variable `buttonState`.

```
buttonState = digitalRead(buttonPin);
```

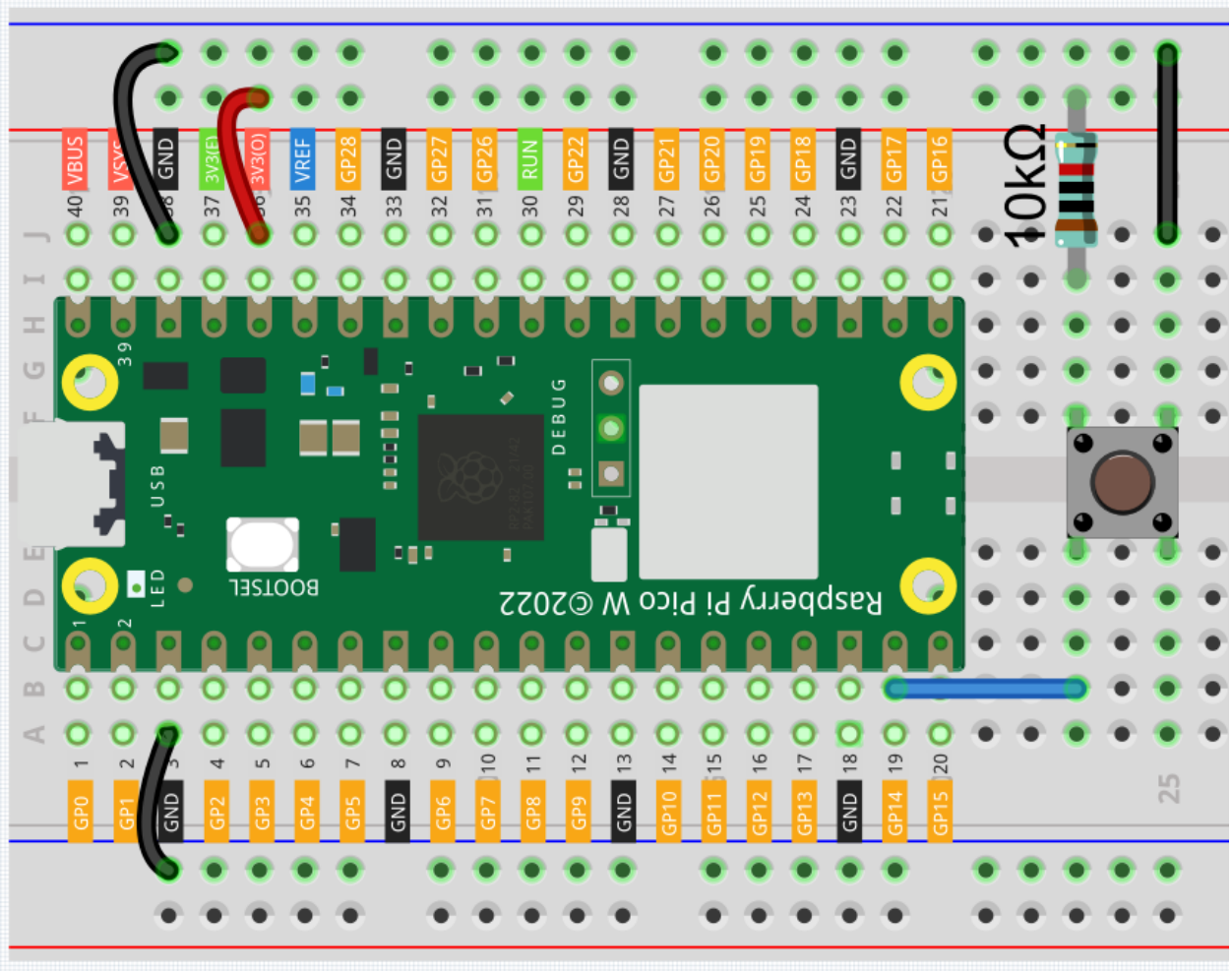
- digitalRead()

If the `buttonState` is HIGH, the LED will flash. print "You pressed the button!" on the Serial monitor.

```
if (buttonState == HIGH) {
    Serial.println("You pressed the button!");
}
```

Pull-up Working Mode

Next is the wiring and code when the button in the pull-up working mode, please try it.



The only difference you will see with the pull-down mode is that the 10K resistor is connected to 3.3V and the button is connected to GND, so that when the button is pressed, GP14 will get a low level, which is the opposite of the value obtained in pull-down mode. So just change this code to `if (buttonState == LOW)`.

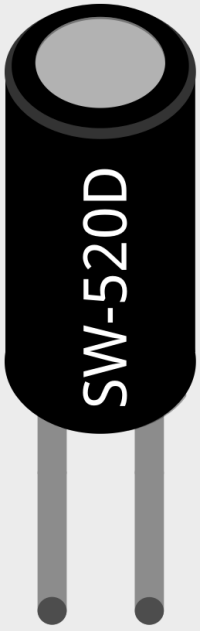
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.10 2.6 - Tilt It



The tilt switch is a 2-pin device with a metal ball in the middle. When you put it upright, the 2 pins are connected together; when you tilt the switch, 2 pins will be disconnected.

Required Components

In this project, we need the following components.

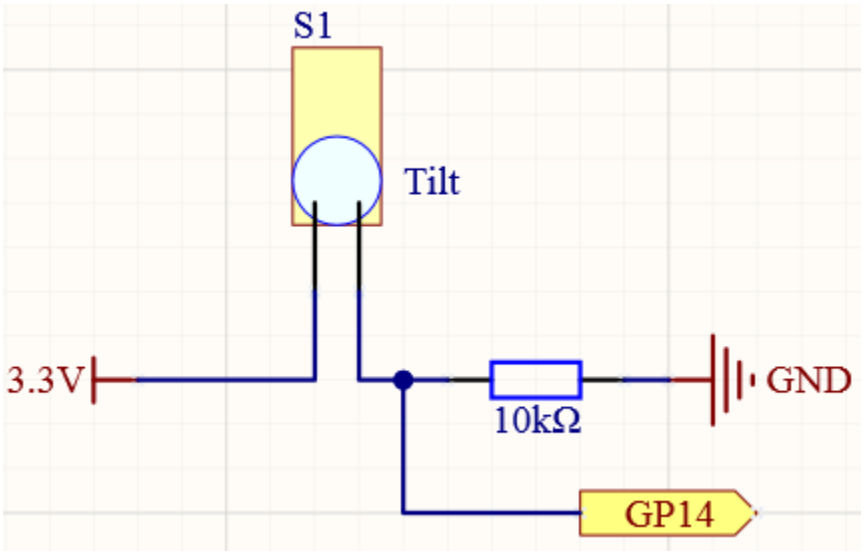
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Tilt Switch*	1	

Schematic

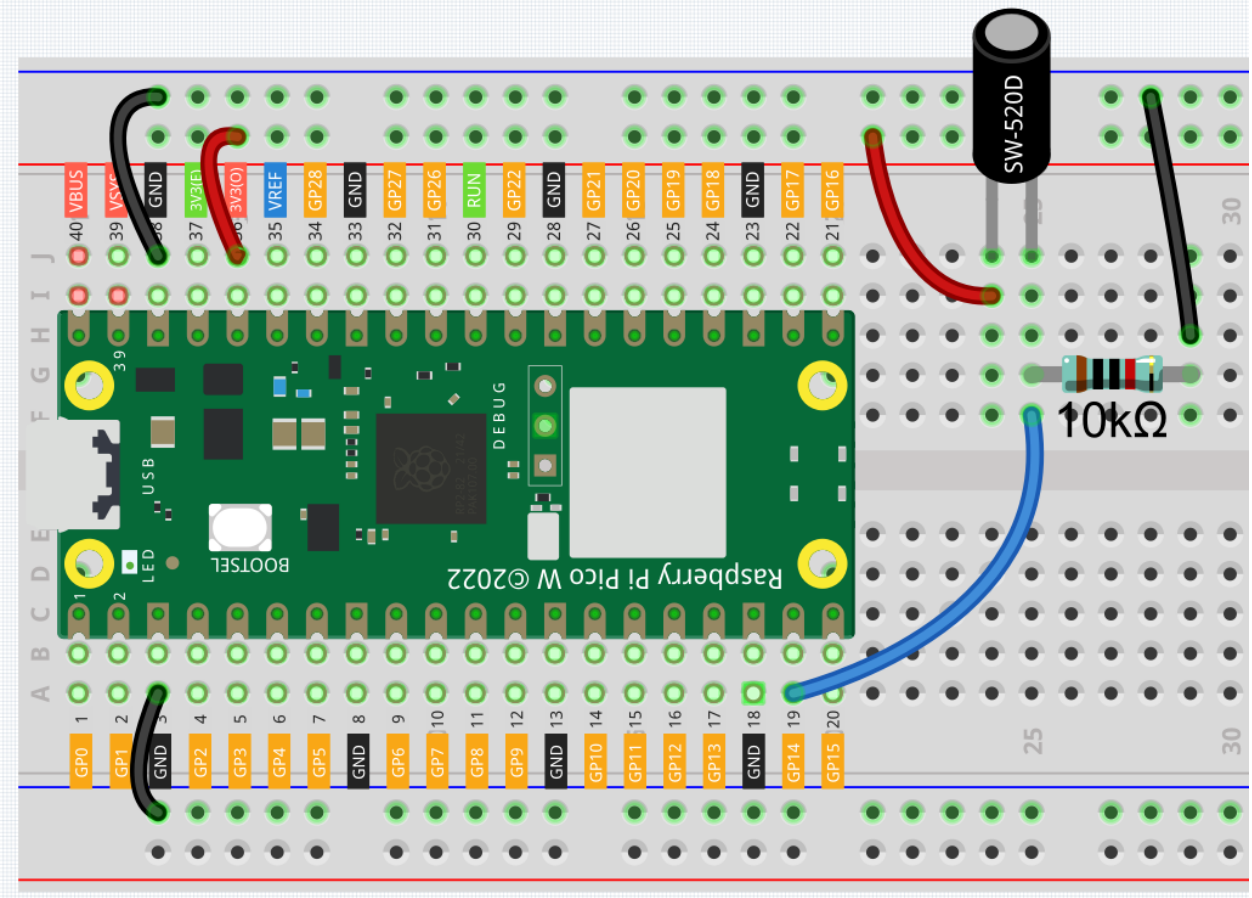


When you put it upright, GP14 will get high; after tilting it, GP14 will get low.

The purpose of the 10K resistor is to keep the GP14 in a stable low state when the tilt switch is in a tilted state.

- *Tilt Switch*

Wiring



Code

Note:

- You can open the file `2.6_tilt_it.ino` under the path of `kepler-kit-main/arduino/2.4_colorful_light`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

After the program runs, when you tilt the breadboard (tilt switch), "The switch works!" will appear in the shell.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.11 2.7 - Toggle Left and Right



The slide switch is a 3-pin device, with pin 2 (middle) being the common pin. When the switch is toggled to the left, the left two pins are connected together, and when toggled to the right, the right two pins are connected together.

Required Components

In this project, we need the following components.

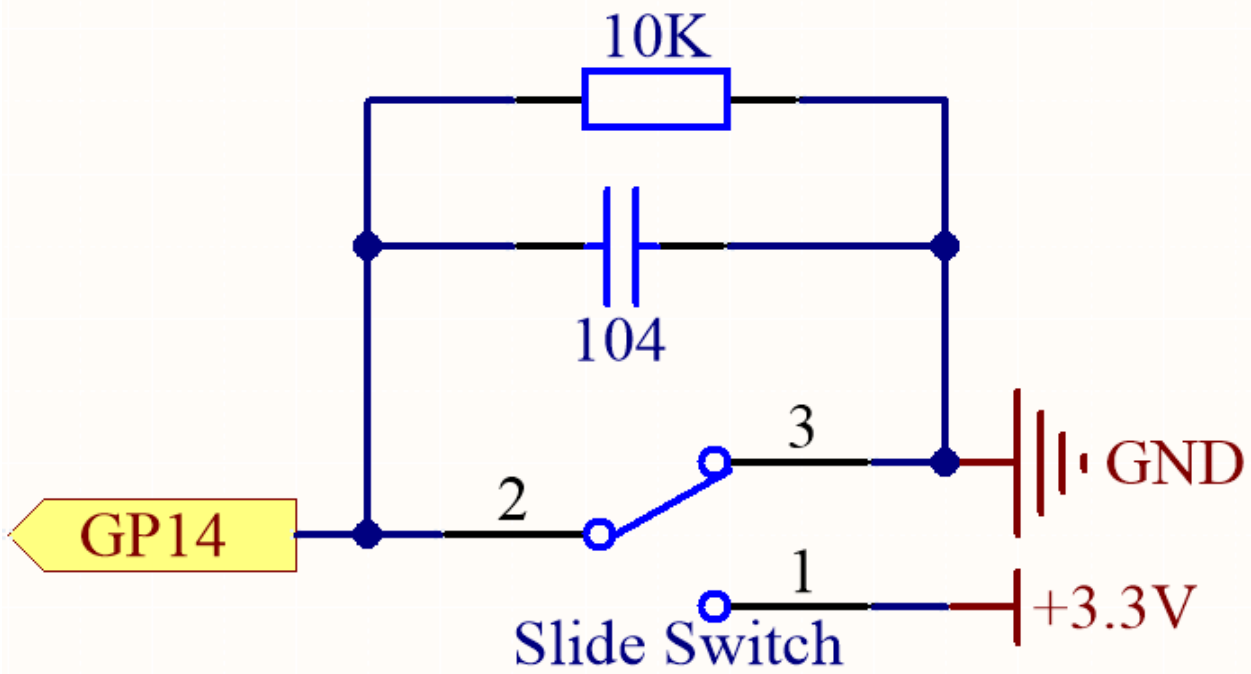
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Capacitor*	1(104)	
7	*Slide Switch*	1	

Schematic



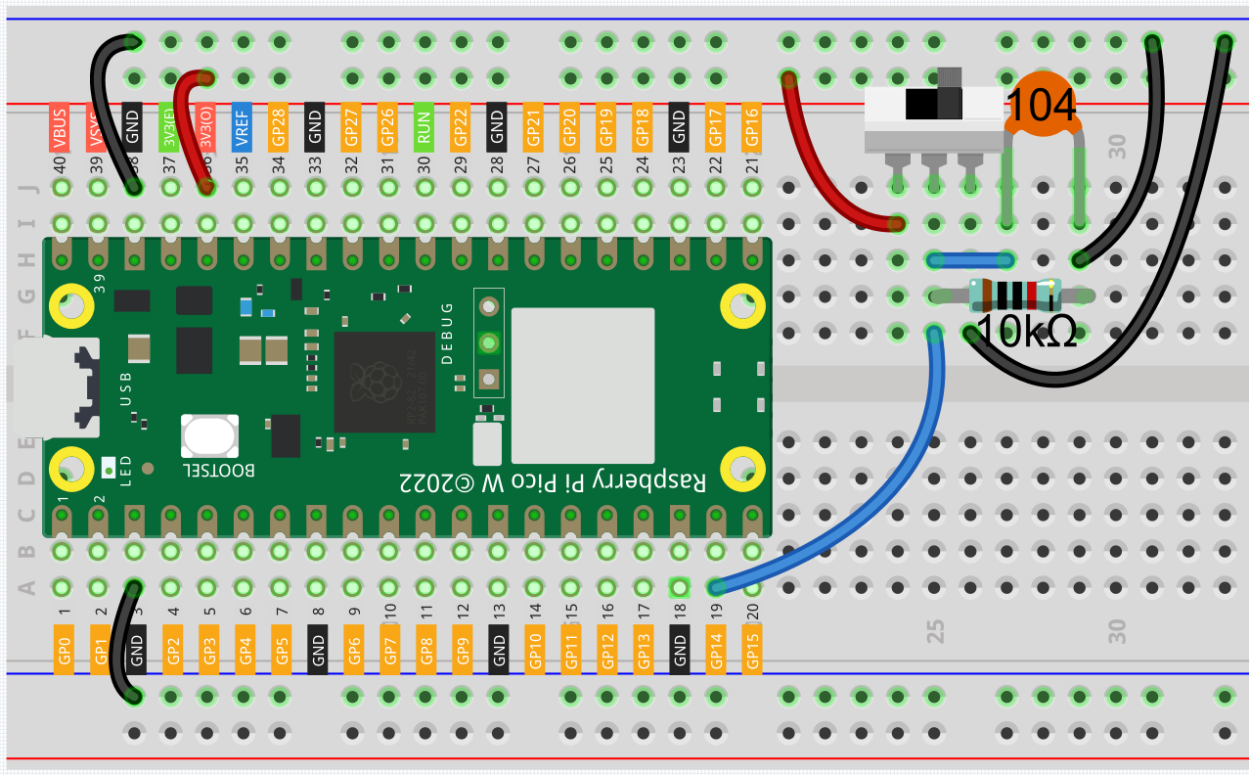
GP14 will get a different level, when you toggle the slide switch to the right or left.

The purpose of the 10K resistor is to keep the GP14 low during toggling (not toggling to the far left and not toggling to the far right).

The 104 ceramic capacitor is used here to eliminate jitter.

- *Slide Switch*
- *Capacitor*

Wiring



Code

Note:

- You can open the file `2.7_toggle_left_right.ino` under the path of `kepler-kit-main/arduino/2.7_toggle_left_right`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

When the program is running, the serial monitor will show "ON" or "OFF" when you toggle the switch to the left or right.

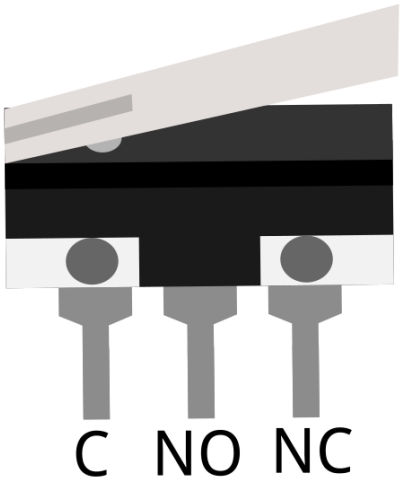
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.12 2.8 - Press Gently



Micro Switch is also a 3-pin device, the sequence of the 3 pins are C (common pin), NO (normally open) and NC (normally closed) .

When the micro switch is not pressed, 1 (C) and 3 (NC) are connected together, when pressed 1 (C) and 2 (NO) are connected together.

- *Micro Switch*

Required Components

In this project, we need the following components.

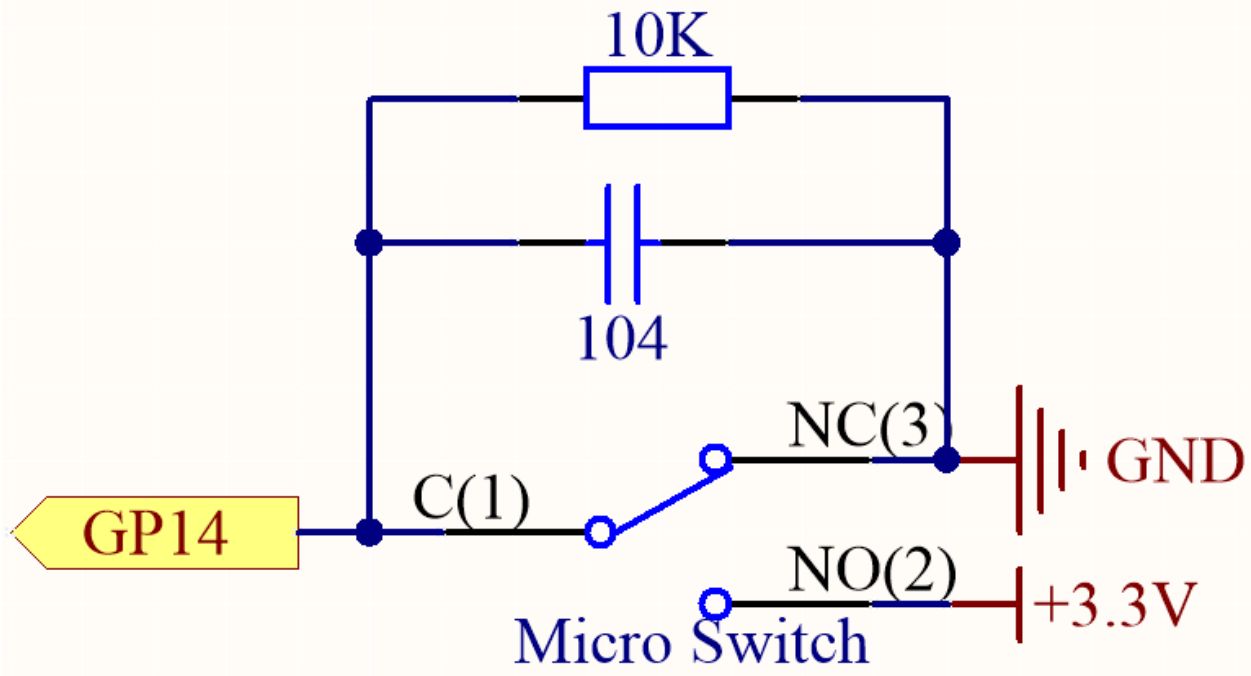
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Capacitor*	1(104)	
7	*Micro Switch*	1	

Schematic

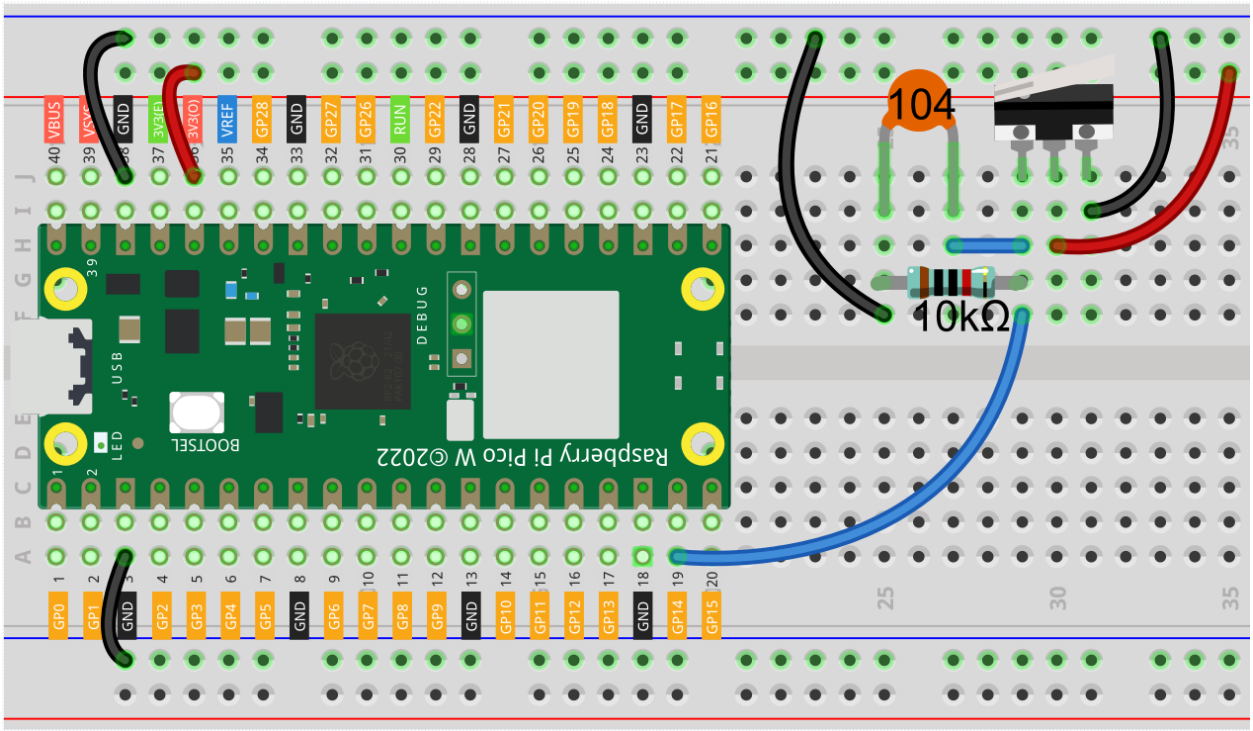


By default, GP14 is low and when pressed, GP14 is high.

The purpose of the 10K resistor is to keep the GP14 low during pressing.

The 104 ceramic capacitor is used here to eliminate jitter.

Wiring



Code

Note:

- You can open the file `2.8_press_gently.ino` under the path of `kepler-kit-main/arduino/2.8_press_gently`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

After the program runs, when you toggle the slide switch to the right, "The switch works!" will appear in the Serial Monitor.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.13 2.9 - Feel the Magnetism

The most common type of reed switch contains a pair of magnetizable, flexible, metal reeds whose end portions are separated by a small gap when the switch is open.

A magnetic field from an electromagnet or a permanent magnet will cause the reeds to attract each other, thus completing an electrical circuit. The spring force of the reeds causes them to separate, and open the circuit, when the magnetic field ceases.

A common example of a reed switch application is to detect the opening of a door or windows, for a security alarm.

- *Reed Switch*

Required Components

In this project, we need the following components.

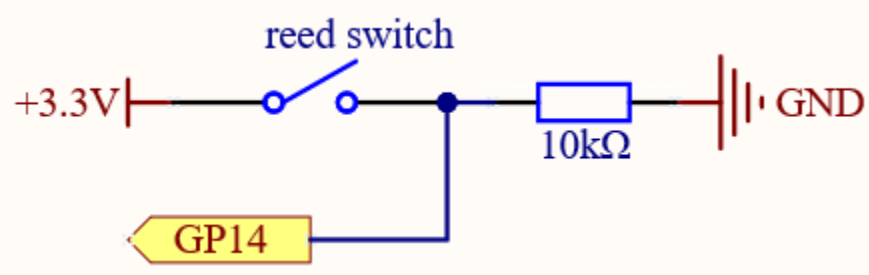
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Reed Switch*	1	

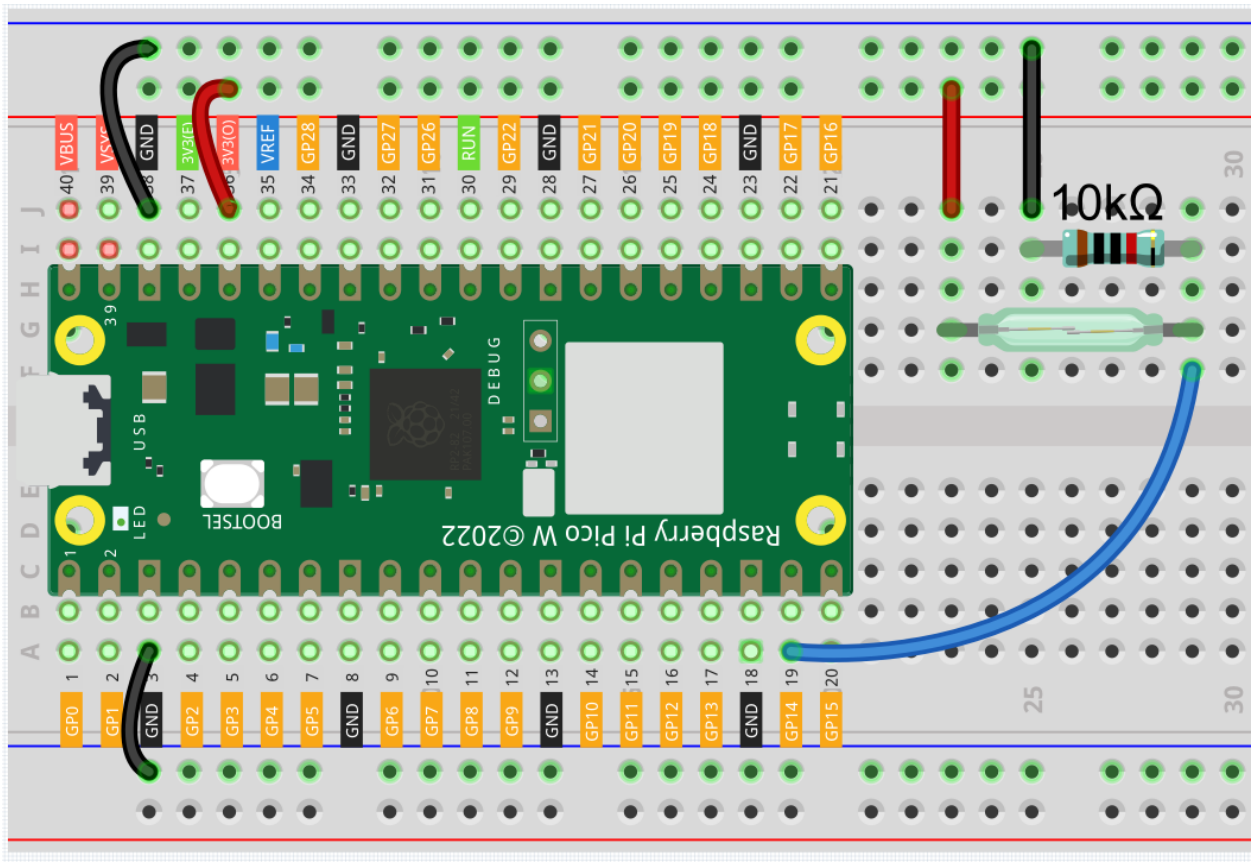
Schematic



By default, GP14 is low; and will go high when the magnet is near the reed switch.

The purpose of the 10K resistor is to keep the GP14 at a steady low level when no magnet is near.

Wiring



Code

Note:

- You can open the file `2.9_feel_the_magnetism.ino` under the path of `kepler-kit-main/arduino/2.9_feel_the_magnetism`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

When a magnet approaches, the circuit will be closed. Just like the button in the *2.5 - Reading Button Value* chapter.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.14 2.10 - Detect Human Movement

Passive infrared sensor (PIR sensor) is a common sensor that can measure infrared (IR) light emitted by objects in its field of view. Simply put, it will receive infrared radiation emitted from the body, thereby detecting the movement of people and other animals. More specifically, it tells the main control board that someone has entered your room.

PIR Motion Sensor Module

Required Components

In this project, we need the following components.

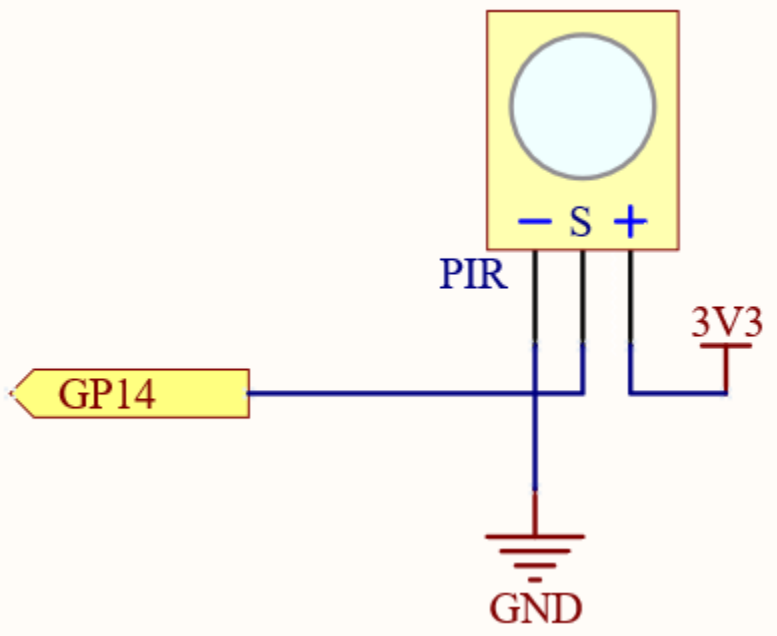
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

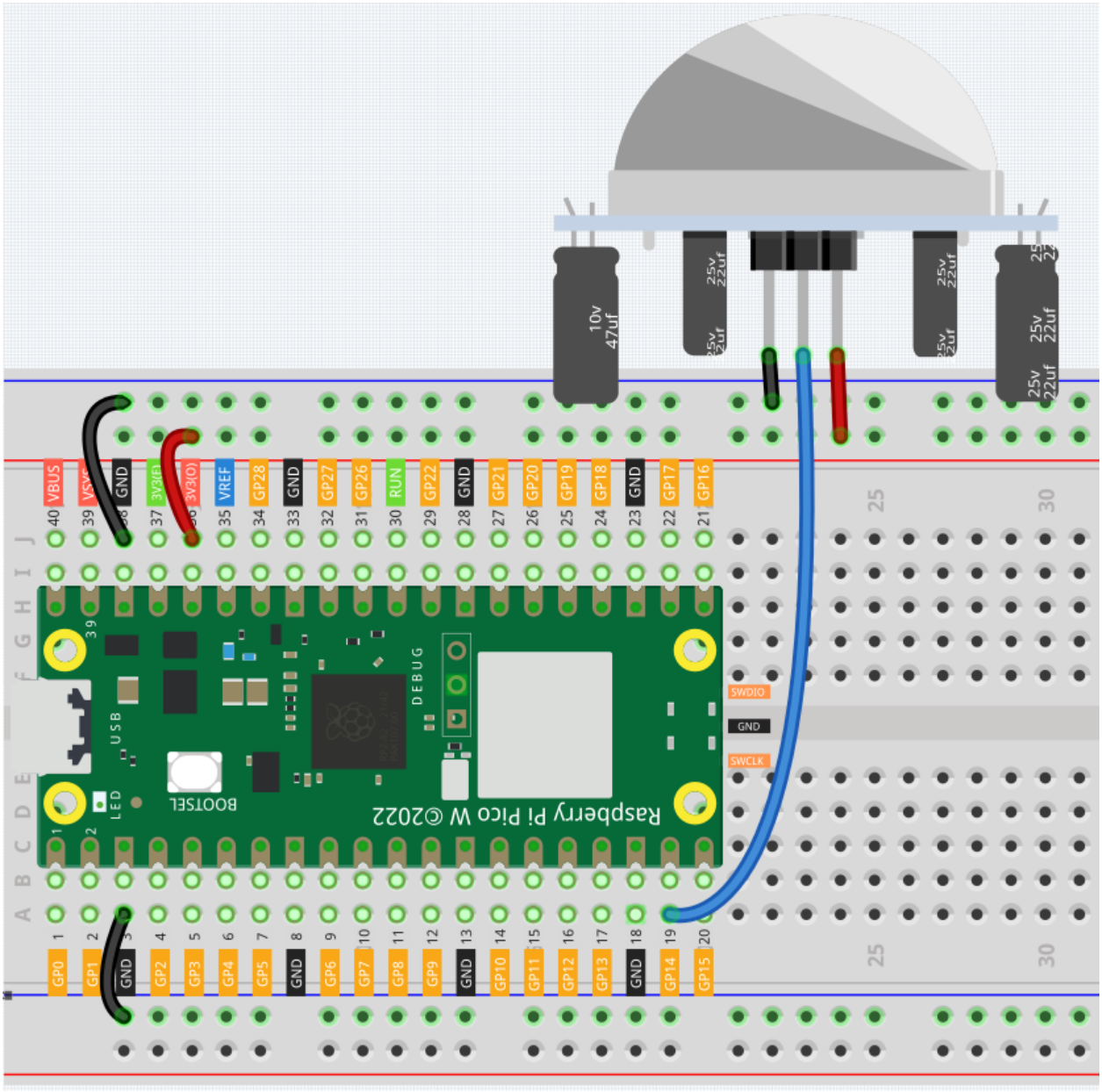
SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*PIR Motion Sensor Module*	1	

Schematic



When the PIR module detects someone passing by, GP14 will be high, otherwise it will be low.

Wiring



Code

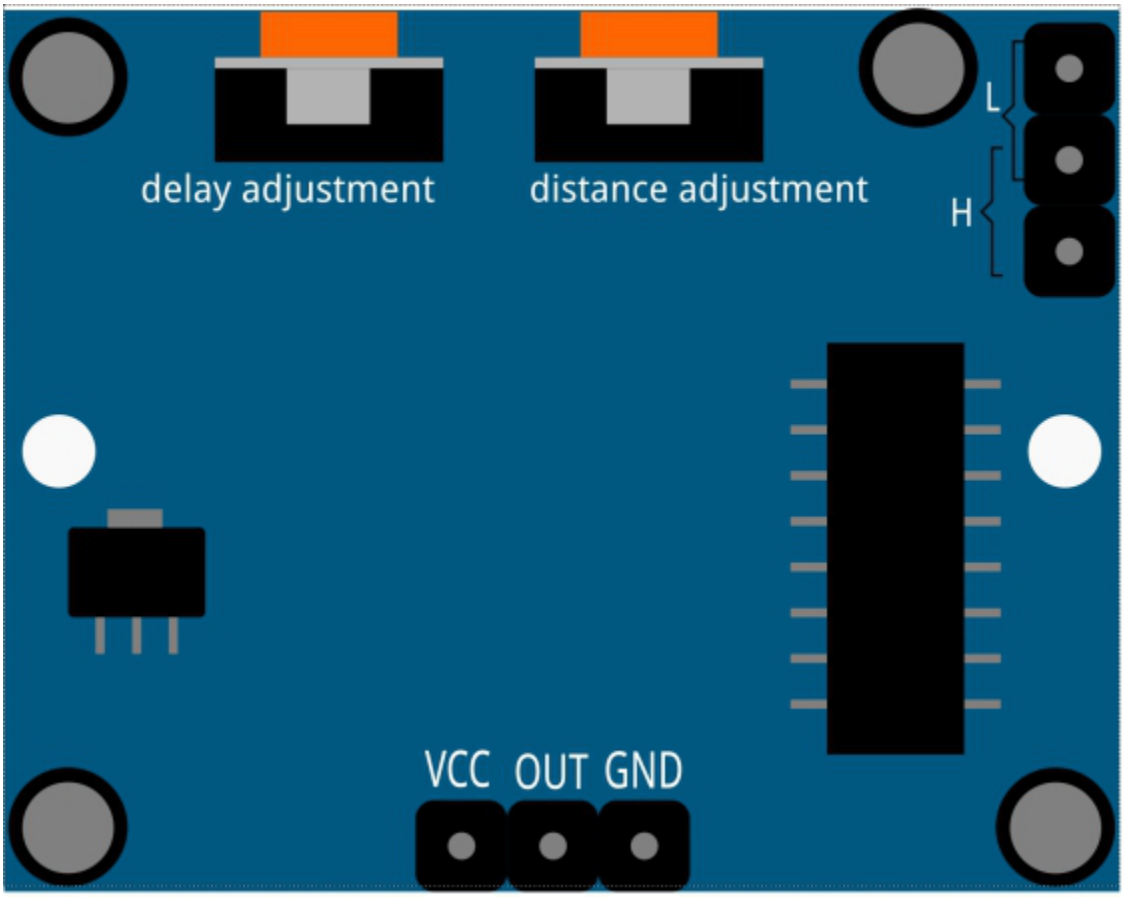
Note:

- You can open the file `2.10_detect_human_movement.ino` under the path of `kepler-kit-main/arduino/2.10_detect_human_movement`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

After the program runs, if the PIR module detects someone nearby, the Serial Monitor will print out "Somebody here!"

Learn More

PIR is a very sensitive sensor. In order to adapt it to the environment of use, it needs to be adjusted. Let the side with the 2 potentiometers facing you, turn both potentiometers counterclockwise to the end and insert the jumper cap on the pin with L and the middle pin.



1. Trigger Mode

   Let's take a look at the pins with jumper cap at the corner. It allows PIR to enter Repeatable trigger mode or Non-repeatable trigger mode

   At present, our jumper cap connects the middle Pin and L Pin, which makes the PIR in non-repeatable trigger mode. In this mode, when the PIR detects the movement of the organism, it will send a high-level signal for about 2.8 seconds to the main control board. .. We can see in the printed data that the duration of work will always be around 2800ms.

   Next, we modify the position of the lower jumper cap and connect it to the middle Pin and H Pin to make the PIR in repeatable trigger mode. In this mode, when the PIR detects the movement of the organism (note that it is movement, not static in front of the sensor), as long as the organism keeps moving within the detection range, the PIR will continue to send a high-level signal to the main control board. .. We can see in the printed data that the duration of work is an uncertain value.

2. Delay Adjustment

   The potentiometer on the left is used to adjust the interval between two jobs.

   At present, we screw it counterclockwise to the end, which makes the PIR need to enter a sleep time of about 5 seconds after finishing sending the high level work. During this time, the PIR will no longer

detect the infrared radiation in the target area. .. We can see in the printed data that the dormancy duration is always no less than 5000ms.

If we turn the potentiometer clockwise, the sleep time will also increase. When it is turned clockwise to the end, the sleep time will be as high as 300s.

3. Distance Adjustment

   The centered potentiometer is used to adjust the sensing distance range of the PIR.

   Turn the knob of the distance adjustment potentiometer **clockwise** to increase the sensing distance range, and the maximum sensing distance range is about 0-7 meters. If it rotates **counterclockwise**, the sensing distance range is reduced, and the minimum sensing distance range is about 0-3 meters.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

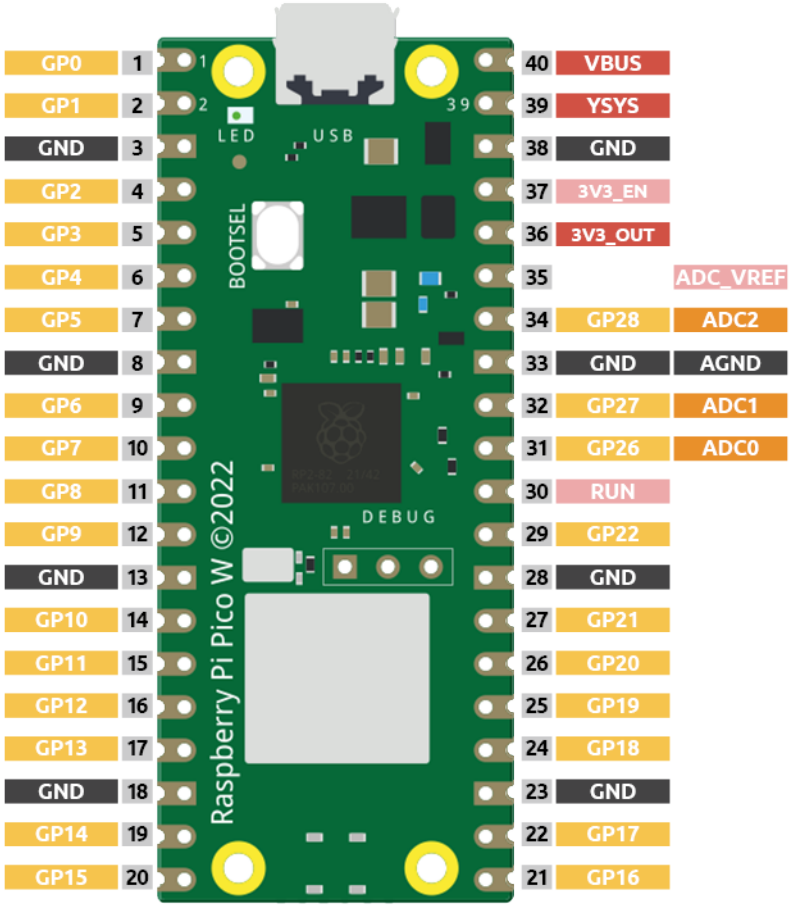
Ready to explore and create with us? Click [] and join today!

6.15 2.11 - Turn the Knob

In the previous projects, we have used the digital input on the Pico W. For example, a button can change the pin from low level (off) to high level (on). This is a binary working state.

However, Pico W can receive another type of input signal: analog input. It can be in any state from fully closed to fully open, and has a range of possible values. The analog input allows the microcontroller to sense the light intensity, sound intensity, temperature, humidity, etc. of the physical world.

Usually, a microcontroller needs an additional hardware to implement analog input-the analogue-to-digital converter (ADC). But Pico W itself has a built-in ADC for us to use directly.



Pico W has three GPIO pins that can use analog input, GP26, GP27, GP28. That is, analog channels 0, 1, and 2. In addition, there is a fourth analog channel, which is connected to the built-in temperature sensor and will not be introduced here.

In this project, we try to read the analog value of potentiometer.

- *Potentiometer*

Required Components

In this project, we need the following components.

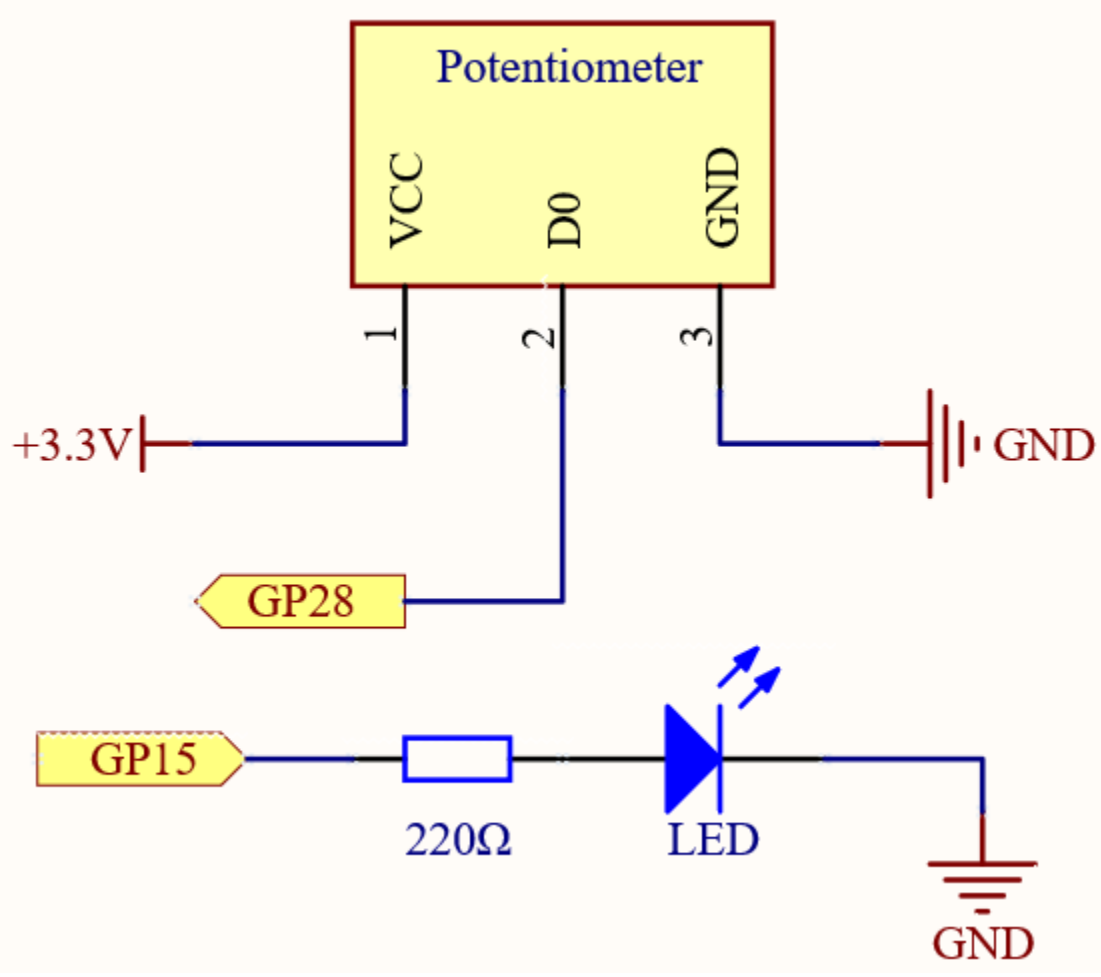
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(220)	
6	*LED*	1	
7	*Potentiometer*	1	

Schematic



The potentiometer is an analog device and when you turn it in 2 different directions.

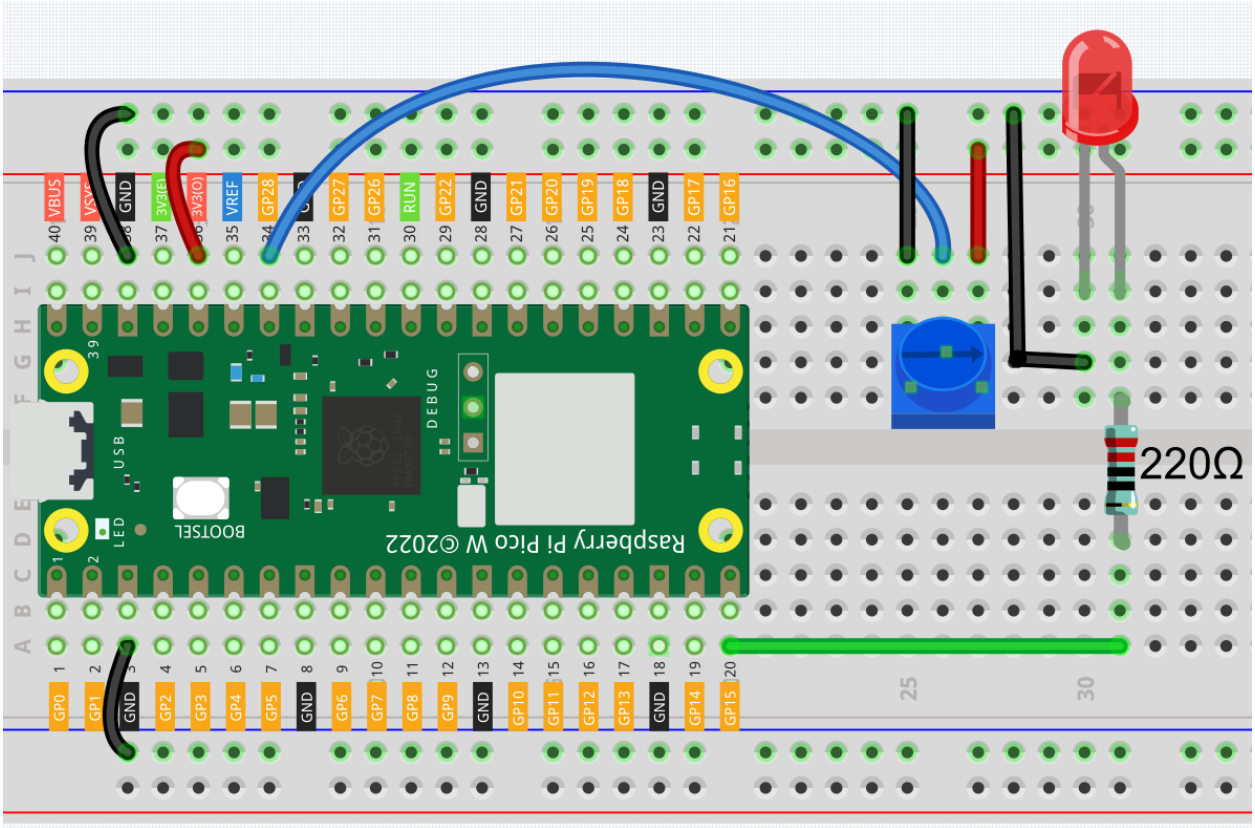
Connect the middle pin of the potentiometer to the analog pin GP28. The Raspberry Pi Pico W contains a multi-channel, 16-bit analog-to-digital converter. This means that it maps the input voltage between 0 and the operating voltage (3.3V) to an integer value between 0 and 65535, so the GP28 value ranges from 0 to 65535.

The calculation formula is shown below.

(Vp/3.3V) x 65535 = Ap

Then program the value of GP28 (potentiometer) as the PWM value of GP15 (LED). This way you will find that by rotating the potentiometer, the brightness of the LED will change at the same time.

Wiring



Code

Note:

- You can open the file `2.11_turn_the_knob.ino` under the path of `kepler-kit-main/arduino/2.11_turn_the_knob`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

When the program is running, we can see the analog value currently read by the GP28 pin in the Serial monitor. Turn the knob, and the value will change from 0 to 1023. At the same time, the brightness of the LED will increase as the analog value increases.

How it works?

To enable Serial Monitor, you need to start serial communication in `setup()` and set the datarate to 9600.

```
void setup() {
    pinMode(ledPin, OUTPUT);
```

(continues on next page)

(continued from previous page)

```
    Serial.begin(9600);
}
```

- Serial

In the loop function, the value of the potentiometer is read, then the value is mapped from 0-1023 to 0-255 and finally the value after the mapping is used to control the brightness of the LED.

```
void loop() {
    int sensorValue = analogRead(sensorPin);
    Serial.println(sensorValue);
    int brightness = map(sensorValue, 0, 1023, 0, 255);
    analogWrite(ledPin, brightness);
}
```

- analogRead() is used to read the value of the sensorPin (potentiometer) and assigns it to the variable `sensorValue`.

```
int sensorValue = analogRead(sensorPin);
```

- Print the value of SensorValue in Serial Monitor.

```
Serial.println(sensorValue);
```

- Here, the map(value, fromLow, fromHigh, toLow, toHigh) function is required as the potentiometer value read is in the range 0-1023 and the value of a PWM pin is in the range 0-255. It is used to Re-maps a number from one range to another. That is, a value of fromLow would get mapped to toLow, a value of fromHigh to toHigh, values in-between to values in-between, etc.

```
int brightness = map(sensorValue, 0, 1023, 0, 255);
```

- Now we can use this value to control the brightness of the LED.

```
analogWrite(ledPin,brightness);
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.16 2.12 - Feel the Light

The photoresistor is a typical device for analog inputs and it is used in a very similar way to a potentiometer. Its resistance value depends on the intensity of the light, the stronger the irradiated light, the smaller its resistance value; conversely, it increases.

- *Photoresistor*

Required Components

In this project, we need the following components.

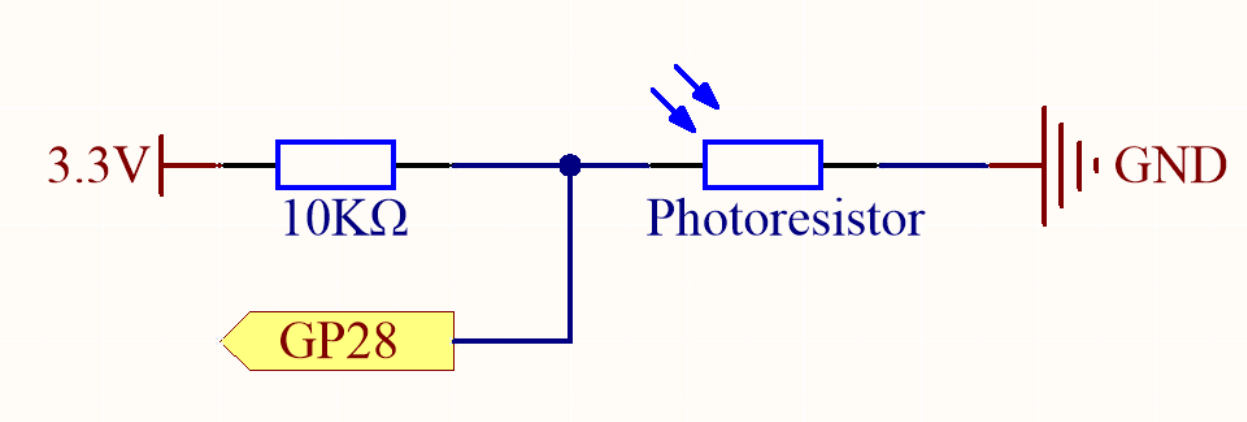
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Photoresistor*	1	

Schematic



In this circuit, the 10K resistor and the photoresistor are connected in series, and the current passing through them is the same. The 10K resistor acts as a protection, and the GP28 reads the value after the voltage conversion of the photoresistor.

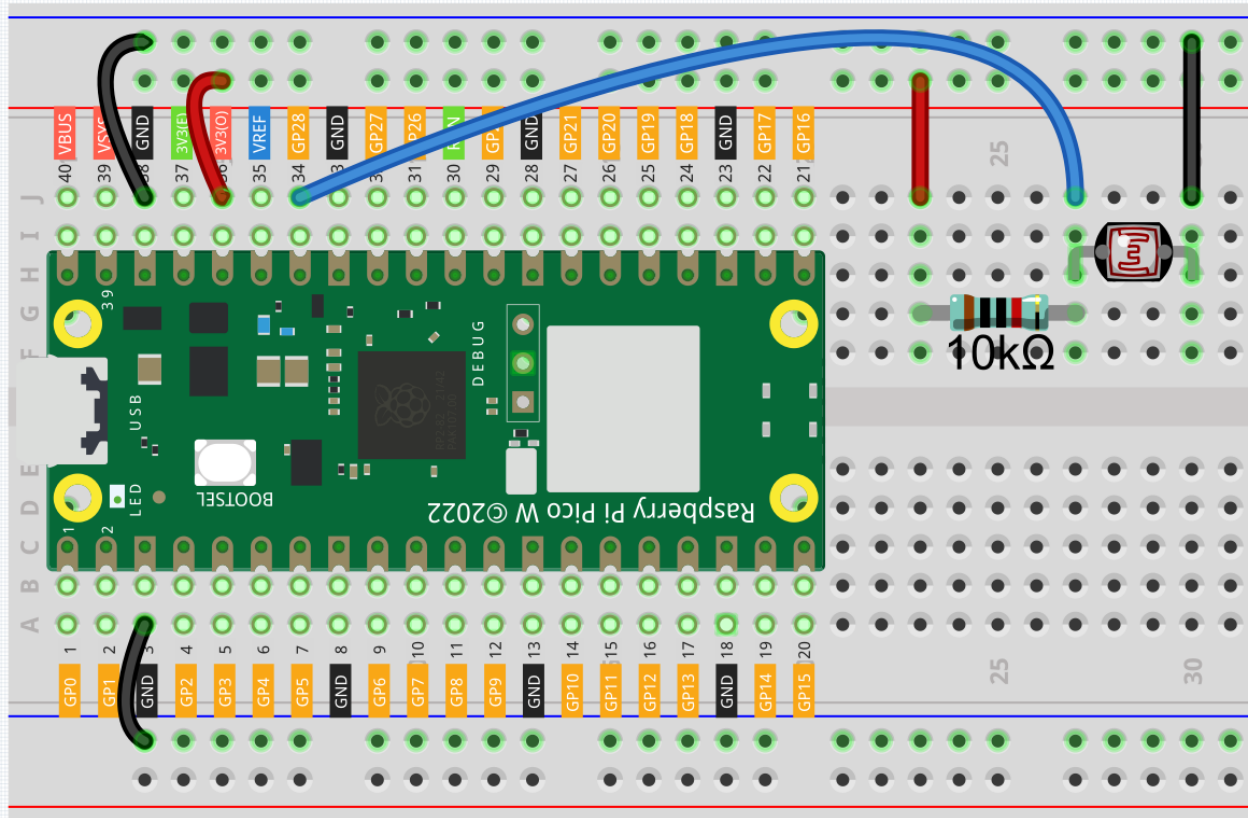
When the light is enhanced, the resistance of the photoresistor decreases, then its voltage decreases, so the value from GP28 will decrease; if the light is strong enough, the resistance of the photoresistor will be close to 0, and the value of GP28 will be close to 0. At this time, the 10K resistor plays a protective role, so that 3.3V and GND are not connected together, resulting in a short circuit.

If you place the photoresistor in a dark situation, the value of GP28 will increase. In a dark enough situation, the resistance of the photoresistor will be infinite, and its voltage will be close to 3.3v (the 10K resistor is negligible), and the value of GP28 will be close to the maximum value of 65535.

The calculation formula is shown below.

(Vp/3.3V) x 65535 = Ap

Wiring



code

Note:

- You can open the file `2.12_feel_the_light.ino` under the path of `kepler-kit-main/arduino/2.12_feel_the_light`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

After the program runs, the Serial Monitor prints out the photoresistor values. You can shine a flashlight on it or cover it up with your hand to see how the value will change.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook!

Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.17 2.13 - Thermometer

A thermometer is a device that measures temperature or a temperature gradient (the degree of hotness or coldness of an object). A thermometer has two important elements: (1) a temperature sensor (e.g. the bulb of a mercury-in-glass thermometer or the pyrometric sensor in an infrared thermometer) in which some change occurs with a change in temperature; and (2) some means of converting this change into a numerical value (e.g. the visible scale that is marked on a mercury-in-glass thermometer or the digital readout on an infrared model). Thermometers are widely used in technology and industry to monitor processes, in meteorology, in medicine, and in scientific research.

A thermistor is a type of temperature sensor whose resistance is strongly dependent on temperature, and it has two types: Negative Temperature Coefficient (NTC) and Positive Temperature Coefficient (PTC), also known as NTC and PTC. The resistance of PTC thermistor increases with temperature, while the condition of NTC is opposite to the former.

In this experiment we use an **NTC thermistor** to make a thermometer.

- *Thermistor*

Required Components

In this project, we need the following components.

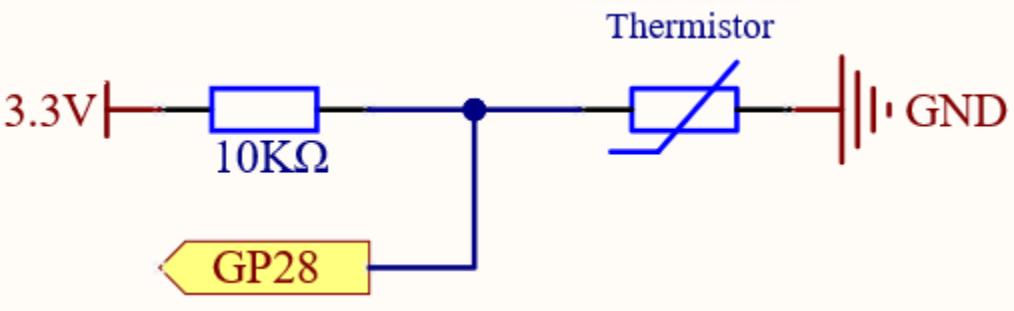
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Thermistor*	1	

Schematic



In this circuit, the 10K resistor and the thermistor are connected in series, and the current passing through them is the same. The 10K resistor acts as a protection, and the GP28 reads the value after the voltage conversion of the thermistor.

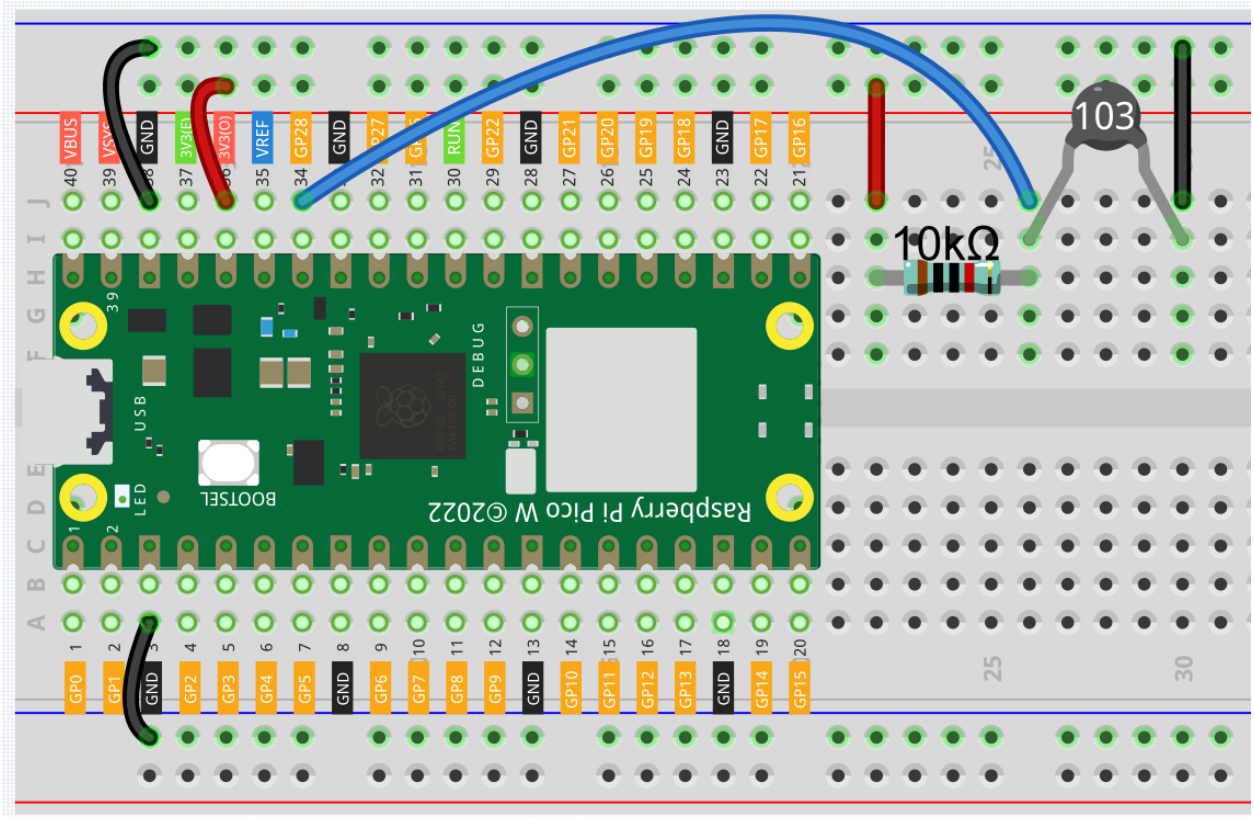
When the temperature increases, the resistance value of NTC thermistor decreases, then its voltage decreases, so the value from GP28 will decrease; If the temperature is high enough, the resistance of the thermistor will be close to 0, and the value of GP28 will be close to 0. At this time, the 10K resistor plays a protective role, so that 3.3V and GND are not connected together, resulting in a short circuit.

When the temperature drops, the value of GP28 will increase. When the temperature is low enough, the resistance of the thermistor will be infinite, and its voltage will be close to 3.3v (the 10K resistor is negligible), and the value of GP28 will be close to the maximum value of 65535.

The calculation formula is shown below.

(Vp/3.3V) x 65535 = Ap

Wiring



Note:

- The thermistor is black and marked 103.
- The color ring of the 10K ohm resistor is red, black, black, red and brown.

Code

Note:

- You can open the file `2.13_thermometer.ino` under the path of `kepler-kit-main/arduino/2.13_thermometer`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

After the program runs, the Serial Monitor will print out the Celsius and Fahrenheit temperatures.

How it works?

Each thermistor has a normal resistance. Here it is 10k ohm, which is measured under 25 degree Celsius.

When the temperature gets higher, the resistance of the thermistor decreases. Then the voltage data is converted to digital quantities by the A/D adapter.

The temperature in Celsius or Fahrenheit is output via programming.

```
long a = analogRead(analogPin);
```

This line is used to read the value of the thermistor.

```
float tempC = beta / (log((1025.0 * 10 / a - 10) / 10) + beta / 298.0) - 273.0;
float tempF = 1.8 * tempC + 32.0;
```

These calculations convert the thermistor values into centigrade degree and Fahrenheit degree.

Note: Here is the relation between the resistance and temperature:

RT =RN expB(1/TK – 1/TN)

- RT is the resistance of the NTC thermistor when the temperature is TK.
- RN is the resistance of the NTC thermistor under the rated temperature TN. Here, the numerical value of RN is 10k.
- TK is a Kelvin temperature and the unit is K. Here, the numerical value of TK is 273.15 + degree Celsius.
- TN is a rated Kelvin temperature; the unit is K too. Here, the numerical value of TN is 273.15+25.
- And B(beta), the material constant of NTC thermistor, is also called heat sensitivity index with a numerical value 3950.
- exp is the abbreviation of exponential, and the base number e is a natural number and equals 2.7 approximately.

Convert this formula TK=1/(ln(RT/RN)/B+1/TN) to get Kelvin temperature that minus 273.15 equals degree Celsius.

This relation is an empirical formula. It is accurate only when the temperature and resistance are within the effective range.

This code refers to plugging Rt into the formula TK=1/(ln(RT/RN)/B+1/TN) to get Kelvin temperature.

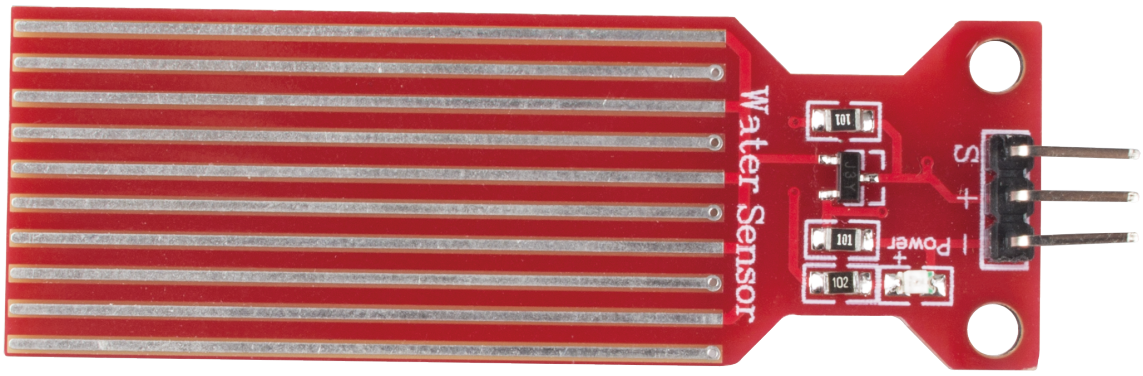
Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.18 2.14 - Feel the Water Level



Water sensor is designed for water detection, which can be widely used in sensing rainfall, water level, and even liquid leakage.

It measures the water level by having a series of exposed parallel wire traces to measure the size of the water drops/volume. The water volume is easily converted to an analog signal, and the output analog value can be read directly by the main control board to achieve the water level alarm effect.

Warning: The sensor cannot be fully submerged in water, please only leave the part where the ten Traces are located in contact with water. Also, energizing the sensor in a humid environment will accelerate the corrosion of the probe and reduce the life of the sensor, so it is recommended that you only supply power when taking readings.

- *Water Level Sensor Module*

Required Components

In this project, we need the following components.

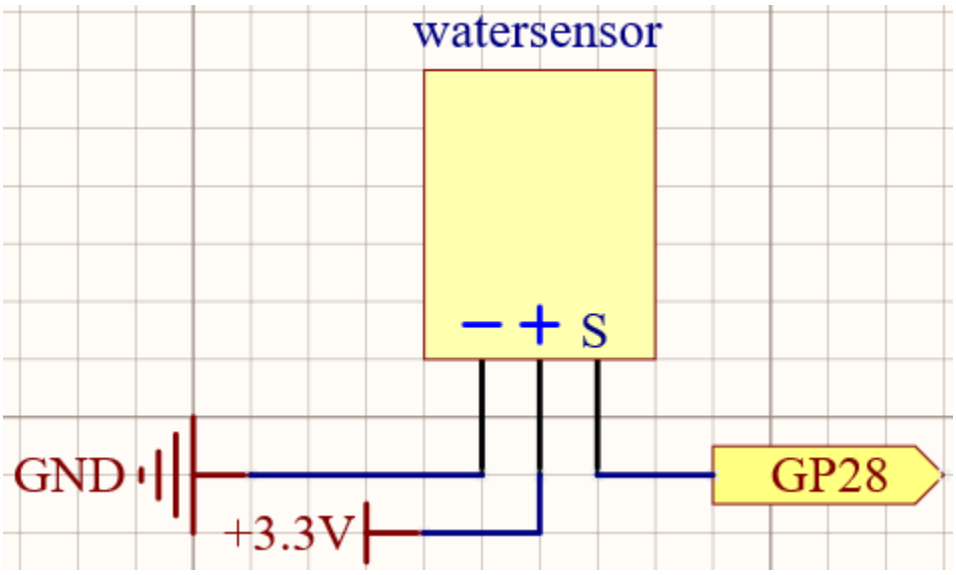
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

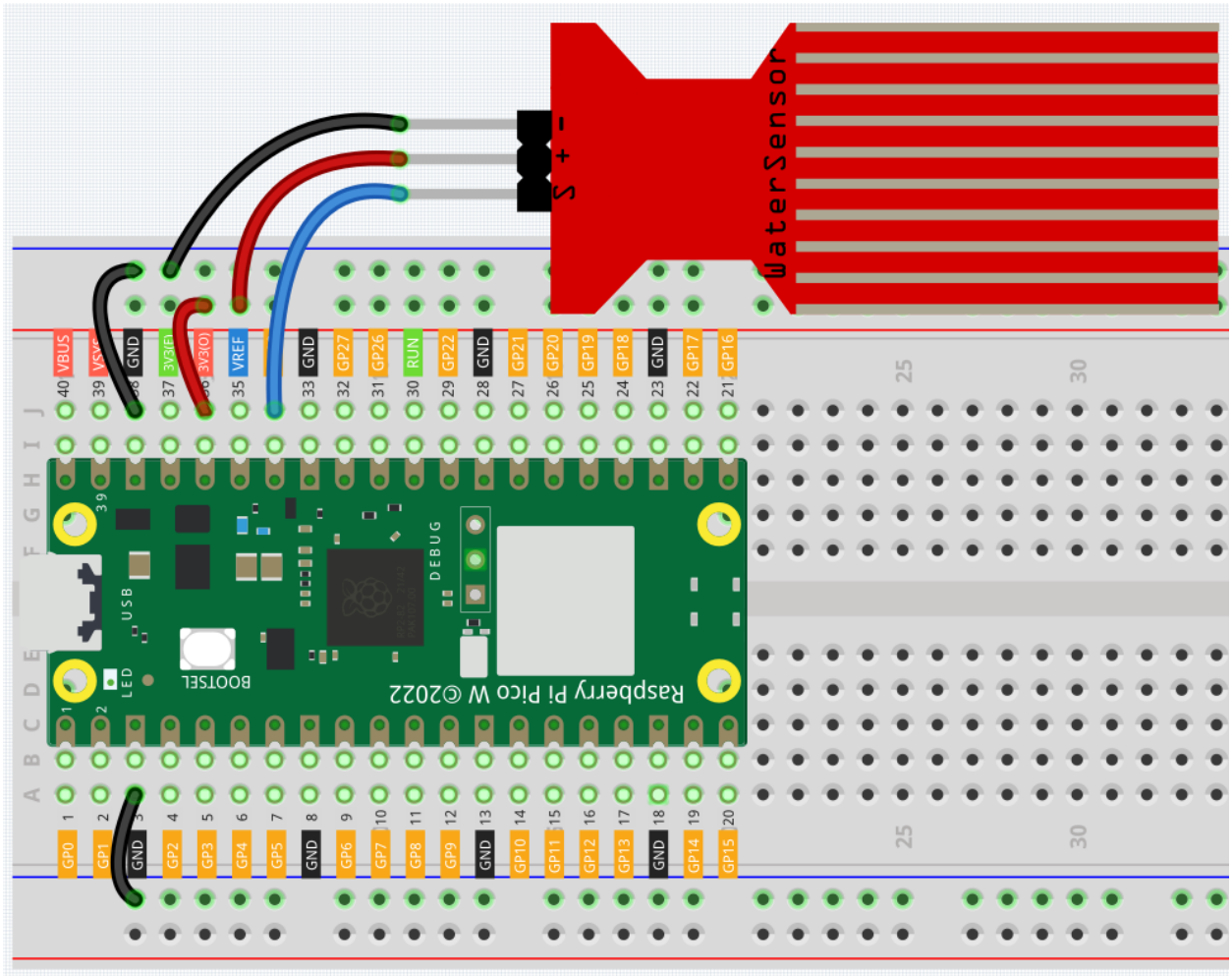
You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUANTITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Water Level Sensor Module*	1	

Schematic



Wiring



Code

Note:

- You can open the file `2.14_feel_the_water_level.ino` under the path of `kepler-kit-main/arduino/2.14_feel_the_water_level`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

After the program is run, submerge the Water Sensor module slowly into the water, and as the depth increases, the Shell will print a larger value.

Learn More

There is a way to use the analog input module as a digital module.

First, take a reading of the Water Sensor in a dry environment first, record it, and use it as a threshold value. Then, complete the programming and re-read the reading of the water sensor. When the reading of the water sensor deviates significantly from the reading in a dry environment, it is exposed to liquid. In other words, by placing this device near a water pipe, it can detect if a water pipe is leaking.

Note:

- You can open the file `2.14_water_level_threshold.ino` under the path of `kepler-kit-main/arduino/2.14_water_level_threshold`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

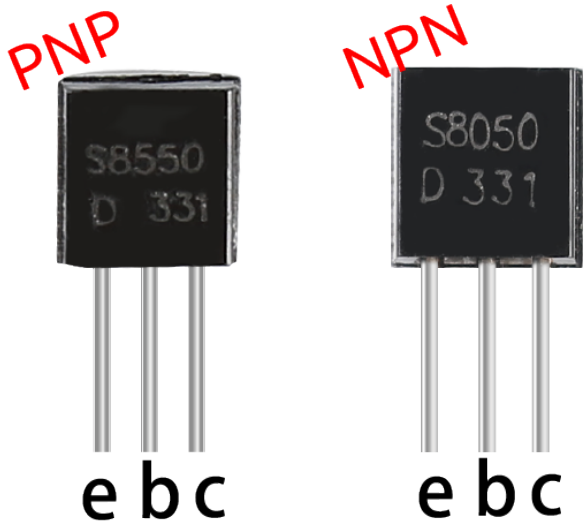
Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.19 2.15 - Two Kinds of Transistors

This kit is equipped with two types of transistors, S8550 and S8050, the former is PNP and the latter is NPN. They look very similar, and we need to check carefully to see their labels. When a High level signal goes through an NPN transistor, it is energized. But a PNP one needs a Low level signal to manage it. Both types of transistor are frequently used for contactless switches, just like in this experiment.



Let's use LED and button to understand how to use transistor!

Transistor

Required Components

In this project, we need the following components.

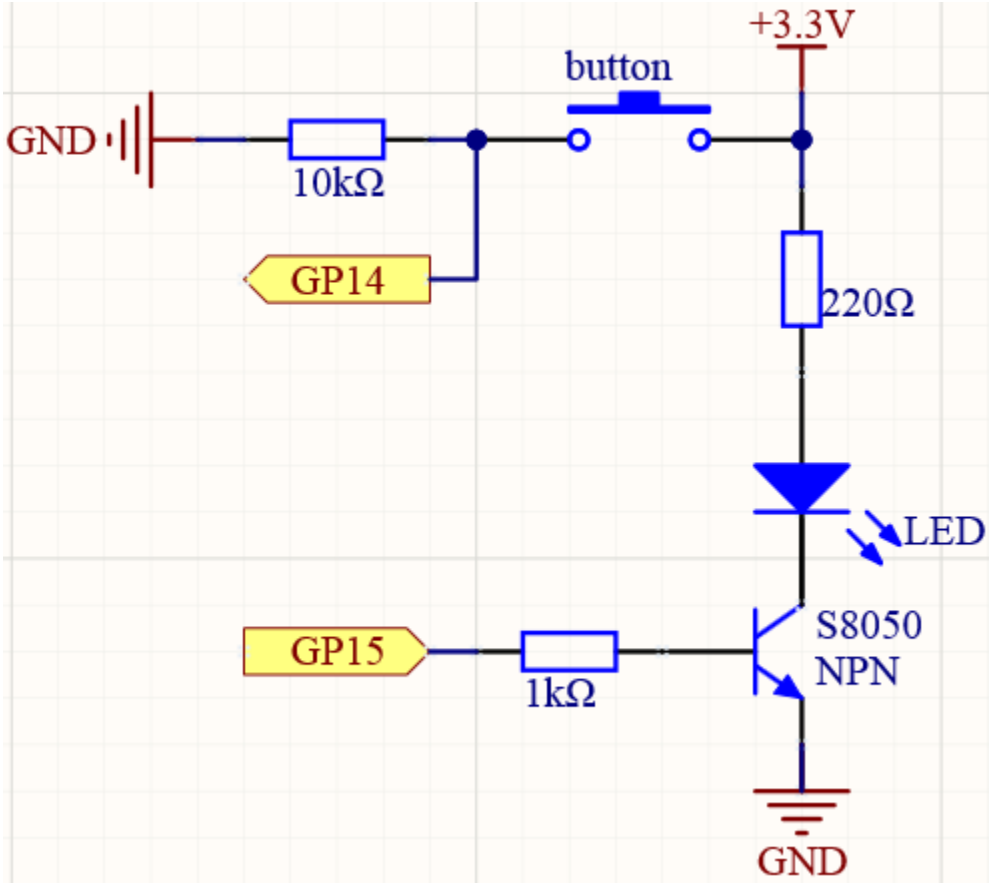
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

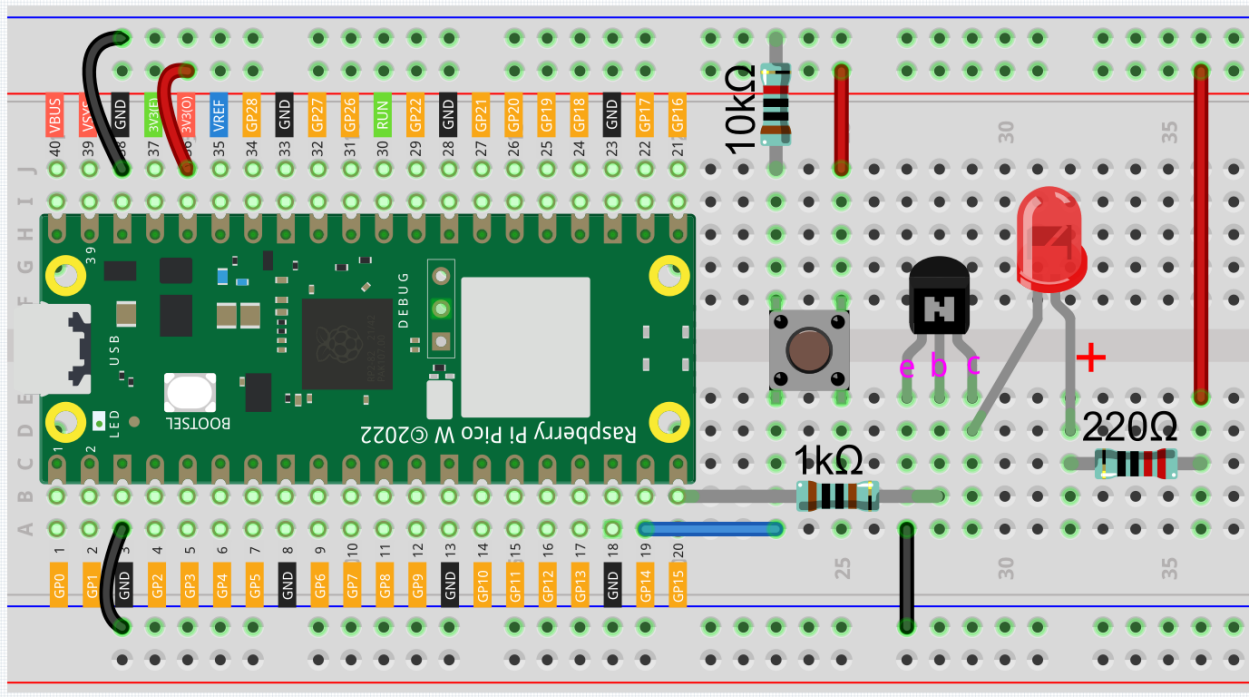
SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	3(220, 1K, 10K)	
6	*LED*	1	
7	*Button*	1	
8	*Transistor*	1(S8050/S8550)	

Way to connect NPN (S8050) transistor

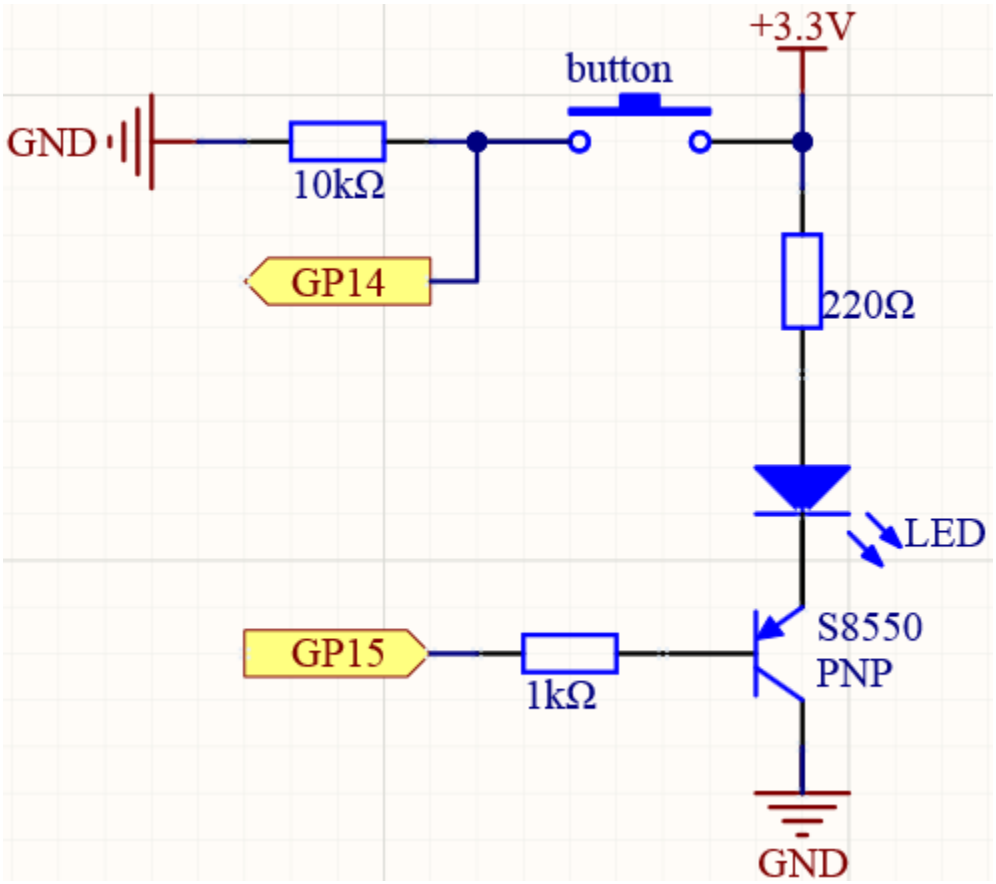


In this circuit, when the button is pressed, GP14 is high.

By programming GP15 to output high, after a 1k current limiting resistor (to protect the transistor), the S8050 (NPN transistor) is allowed to conduct, thus allowing the LED to light up.



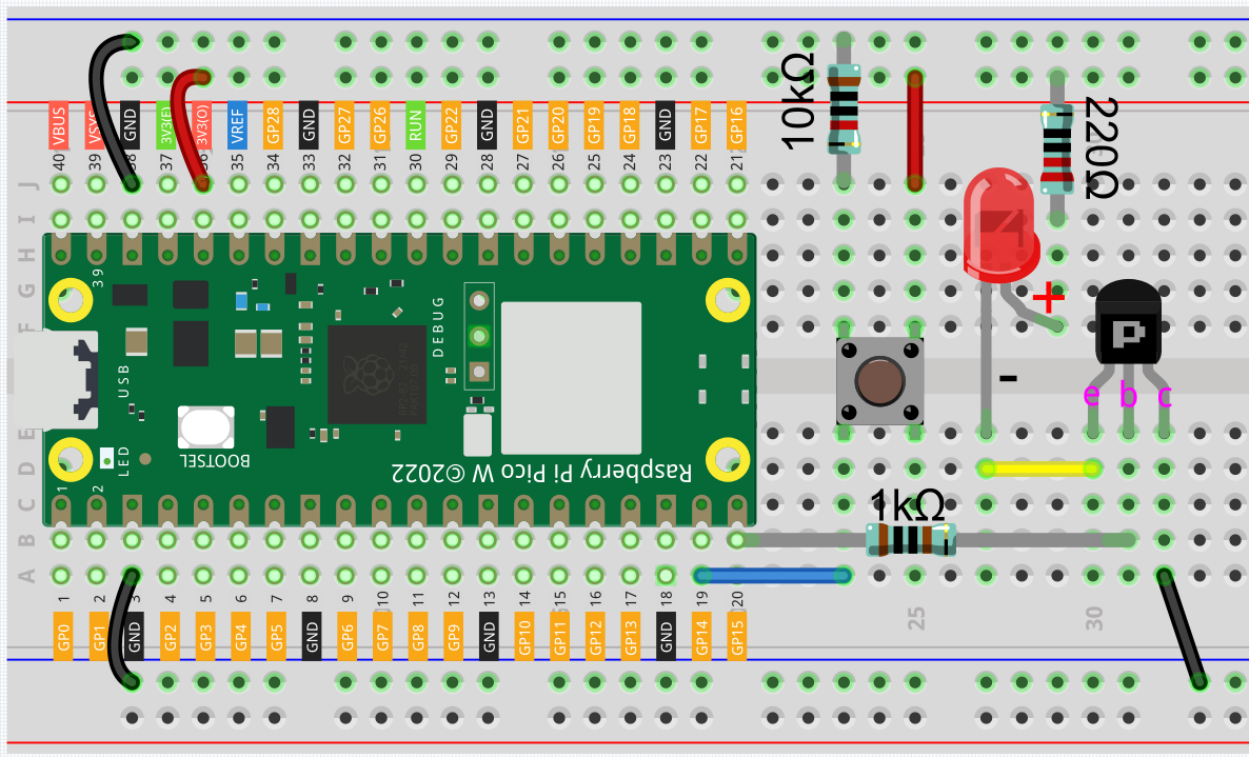
Way to connect PNP(S8550) transistor



In this circuit, GP14 is low by the default and will change to high when the button is pressed.

By programming GP15 to output **low**, after a 1k current limiting resistor (to protect the transistor), the S8550 (PNP transistor) is allowed to conduct, thus allowing the LED to light up.

The only difference you will notice between this circuit and the previous one is that in the previous circuit the cathode of the LED is connected to the **collector** of the **S8050 (NPN transistor)**, while this one is connected to the **emitter** of the **S8550 (PNP transistor)**.



Code

Note:

- You can open the file `2.15_transistor.ino` under the path of `kepler-kit-main/arduino/2.15_transistor`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

Two kinds of transistors can be controlled with the same code. When we press the button, Pico W will send a high-level signal to the transistor; when we release it, it will send a low-level signal. We can see that diametrically opposite phenomena have occurred in the two circuits.

- The circuit using the S8050 (NPN transistor) will light up when the button is pressed, which means it is receiving a high-level conduction circuit;
- The circuit that uses the S8550 (PNP transistor) will light up when it is released, which means it is receiving a low-level conduction circuit.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.20 2.16 - Control Another Circuit

In our daily life, we can press the switch to light up or turn off the lamp. But what if you want to control the lamp with Pico W so that it can turn off automatically after ten minutes?

A relay can help you accomplish this idea.

A relay is actually a special kind of switch that is controlled by one side of the circuit (usually a low-voltage circuit) and used to control the other side of the circuit (usually a high-voltage circuit). This makes it practical to modify our home appliances to be controlled by a program, to become smart devices, or even to access the Internet.

Warning: Modification of electrical appliances comes with great danger, do not try it lightly, please do it under the guidance of professionals.

- *Relay*

Here we only use a simple circuit powered by a breadboard power module as an example to show how to control it using relay.

- cpn_power_module

Required Components

In this project, we need the following components.

It's definitely convenient to buy a whole kit, here's the link:

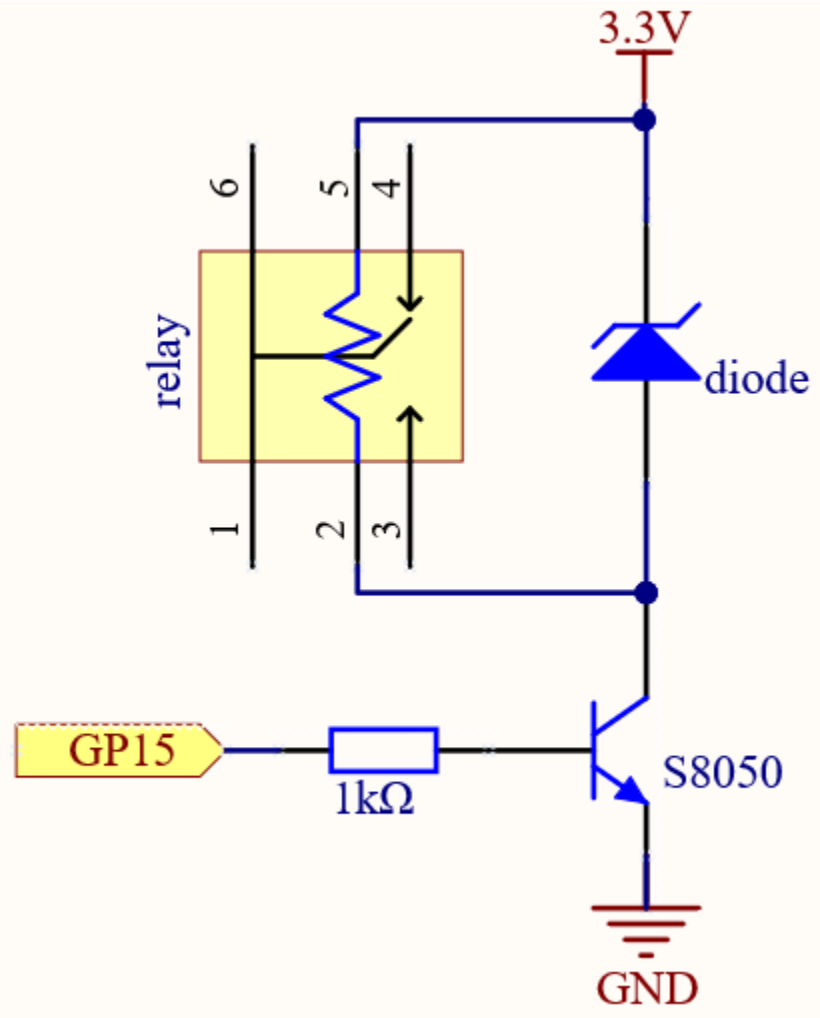
Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

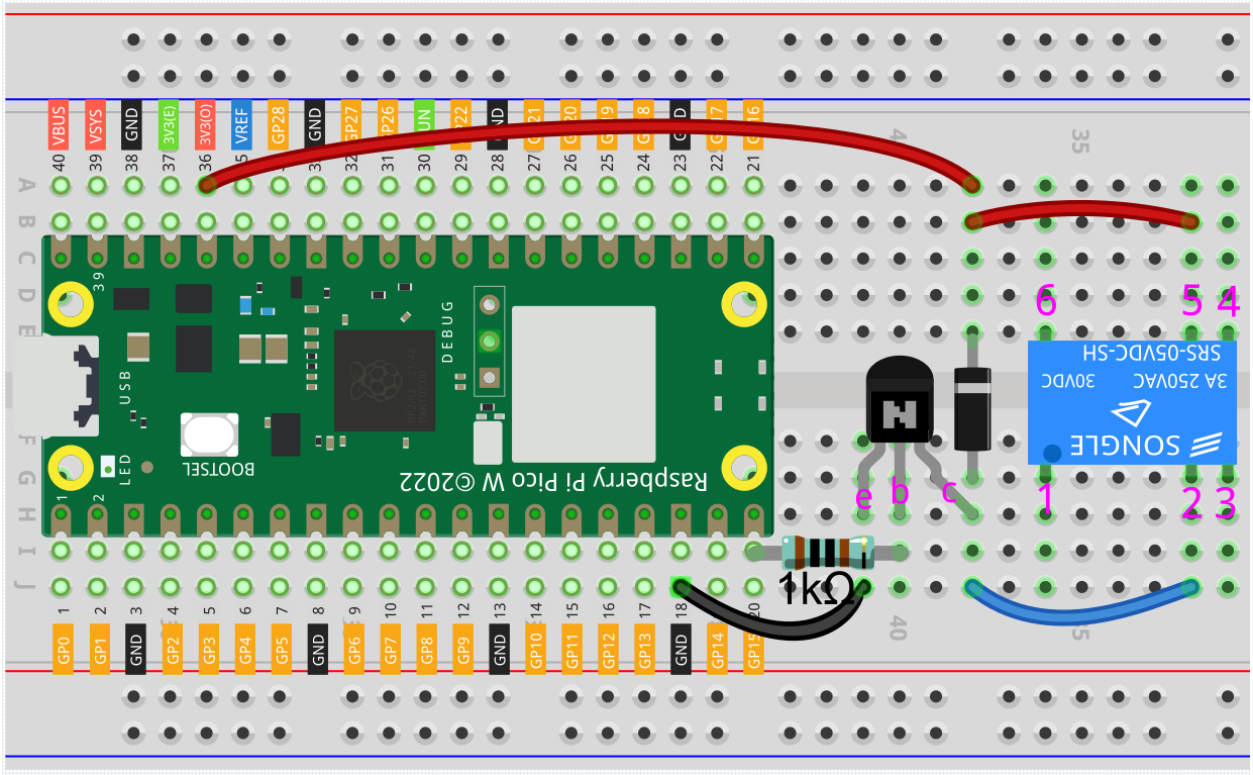
You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Transistor*	1(S8050)	
6	*Diode*	1	
7	*Relay*	1	

Wiring

First, build a low-voltage circuit for controlling a relay. Driving the relay requires a high current, so a transistor is needed, and here we use the S8050.





A diode (continuity diode) is used here to protect the circuit. The cathode is the end with the silver ribbon connected to the power supply, and the anode is connected to the transistor.

When the voltage input changes from High (5V) to Low (0V), the transistor changes from saturation (amplification, saturation, and cutoff) to cutoff, and there is suddenly no way for current to flow through the coil.

At this point, if this freewheeling diode does not exist, the coil will produce a self-induced electric potential at both ends that is several times higher than the supply voltage, and this voltage plus the voltage from the transistor power supply is enough to burn it.

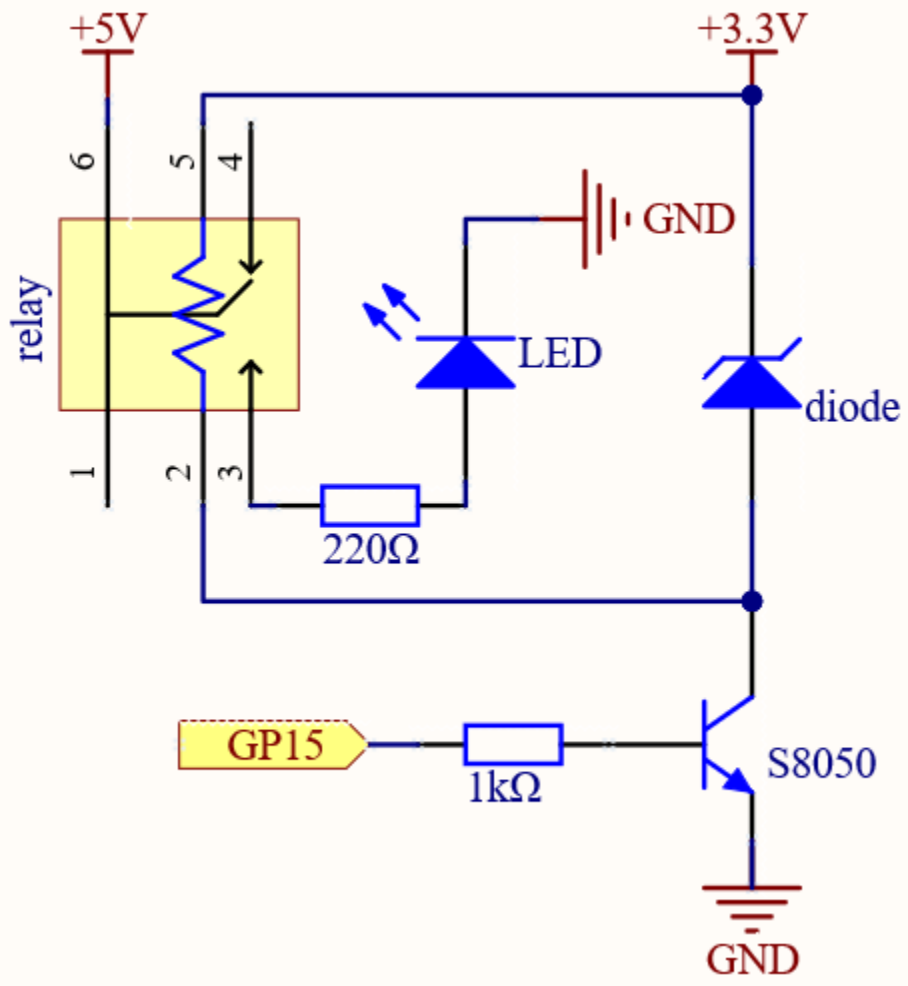
After adding the diode, the coil and the diode instantly form a new circuit powered by the energy stored in the coil to discharge, thus avoiding the excessive voltage will damage devices such as transistors on the circuit.

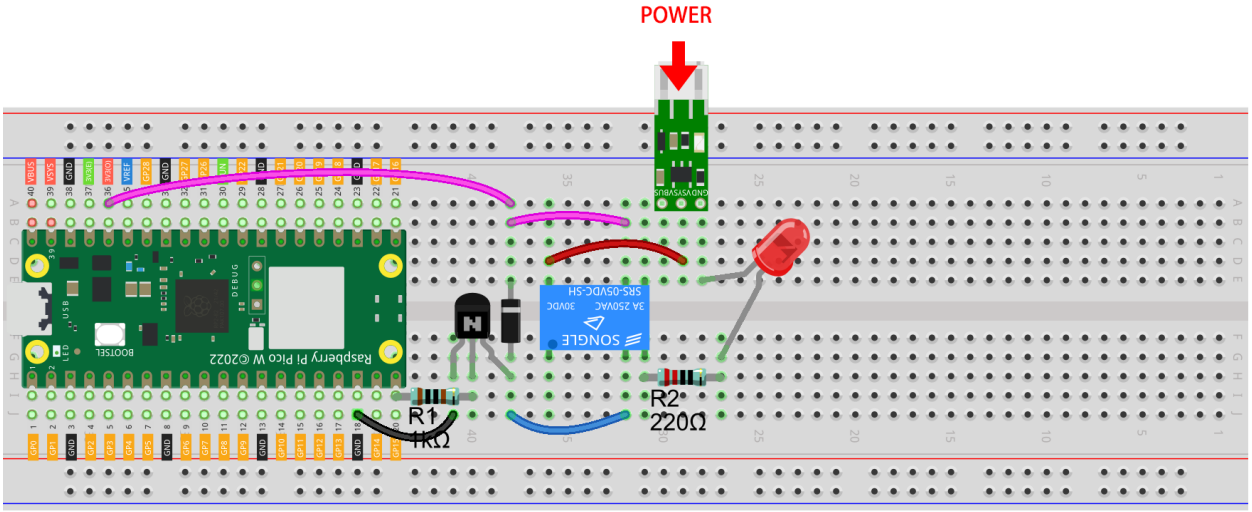
- *Diode*
- Flyback Diode - Wikipedia

At this point the program is ready to run, and after running you will hear the “tik tok” sound, which is the sound of the contactor coil inside the relay sucking and breaking.

Then we connect the two ends of the load circuit to pins 3 and 6 of the relay respectively.

..(Take the simple circuit powered by the breadboard power module described in the previous article as an example.)





At this point, the relay will be able to control the load circuit on and off.

Code

Note:

- You can open the file `2.16_relay.ino` under the path of `kepler-kit-main/arduino/2.16_relay`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

When the code is run, the relay will switch the operating state of the controlled circuit every two seconds. You can manually comment out one of the lines to further clarify the correspondence between the relay circuit and the load circuit.

Learn More

Pin 3 of the relay is normally open and only turns on when the contactor coil is operating; pin 4 is normally closed and turns on when the contactor coil is energized. Pin 1 is connected to pin 6 and is the common terminal of the load circuit.

By switching one end of the load circuit from pin 3 to pin 4, you will be able to get exactly the opposite operating state.

3. Sound & Display & Movement

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.21 3.1 - Beep

The active buzzer is a typical digital output device that is as easy to use as lighting up an LED!

- *Buzzer*

Required Components

In this project, we need the following components.

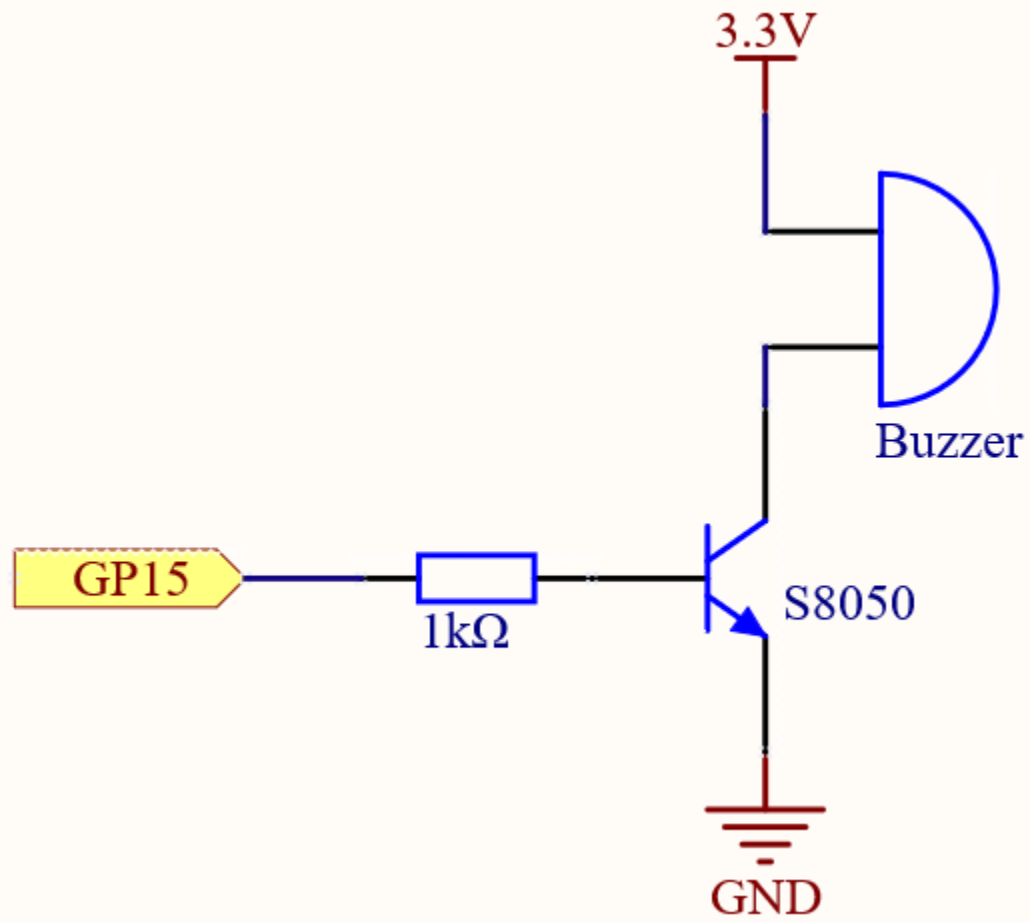
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Transistor*	1(S8050)	
6	*Resistor*	1(1K)	
7	Active *Buzzer*	1	

Schematic



When the GP15 output is high, after the 1K current limiting resistor (to protect the transistor), the S8050 (NPN transistor) will conduct, so that the buzzer will sound.

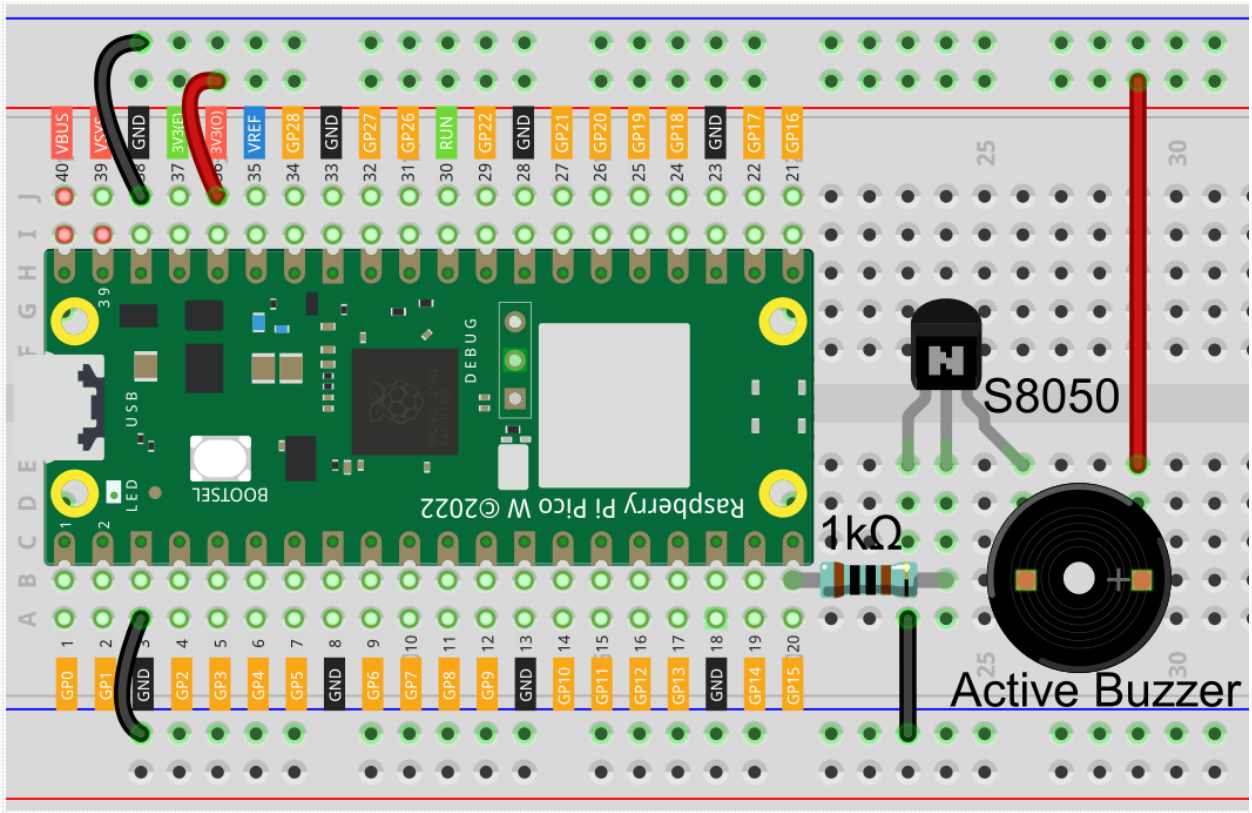
The role of S8050 (NPN transistor) is to amplify the current and make the buzzer sound louder. In fact, you can also connect the buzzer directly to GP15, but you will find that the buzzer sound is smaller.

Wiring

Two types of buzzers are included in the kit. We need to use active buzzer. Turn them around, the sealed back (not the exposed PCB) is the one we want.



The buzzer needs to use a transistor when working, here we use S8050 (NPN Transistor).



Code

Note:

- You can open the file `3.1_beep.ino` under the path of `kepler-kit-main/arduino/3.1_beep`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

After the code runs, you will hear a beep every second.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.22 3.2 - Custom Tone

We have used active buzzer in the previous project, this time we will use passive buzzer.

Like the active buzzer, the passive buzzer also uses the phenomenon of electromagnetic induction to work. The difference is that a passive buzzer does not have oscillating source, so it will not beep if DC signals are used. But this allows the passive buzzer to adjust its own oscillation frequency and can emit different notes such as "doh, re, mi, fa, sol, la, ti".

Let the passive buzzer emit a melody!

- *Buzzer*

Required Components

In this project, we need the following components.

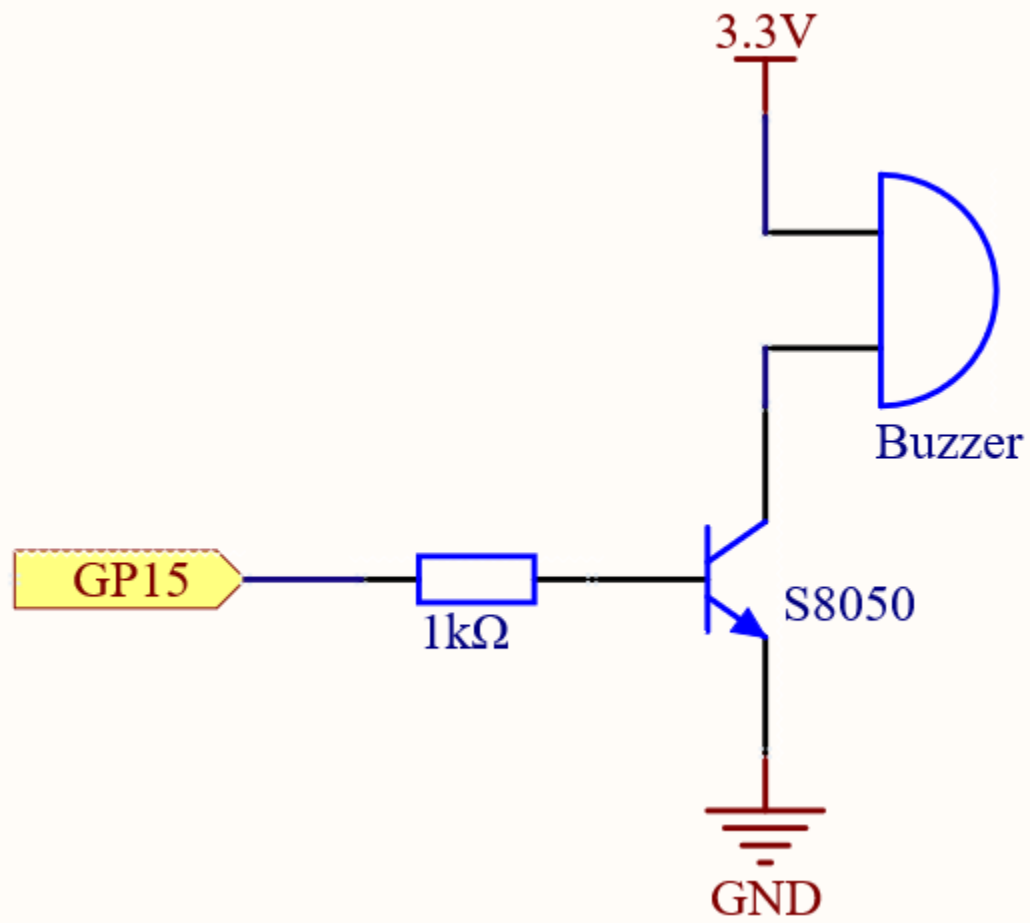
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Transistor*	1(S8050)	
6	*Resistor*	1(1K)	
7	Passive *Buzzer*	1	

Schematic



When the GP15 output is high, after the 1K current limiting resistor (to protect the transistor), the S8050 (NPN transistor) will conduct, so that the buzzer will sound.

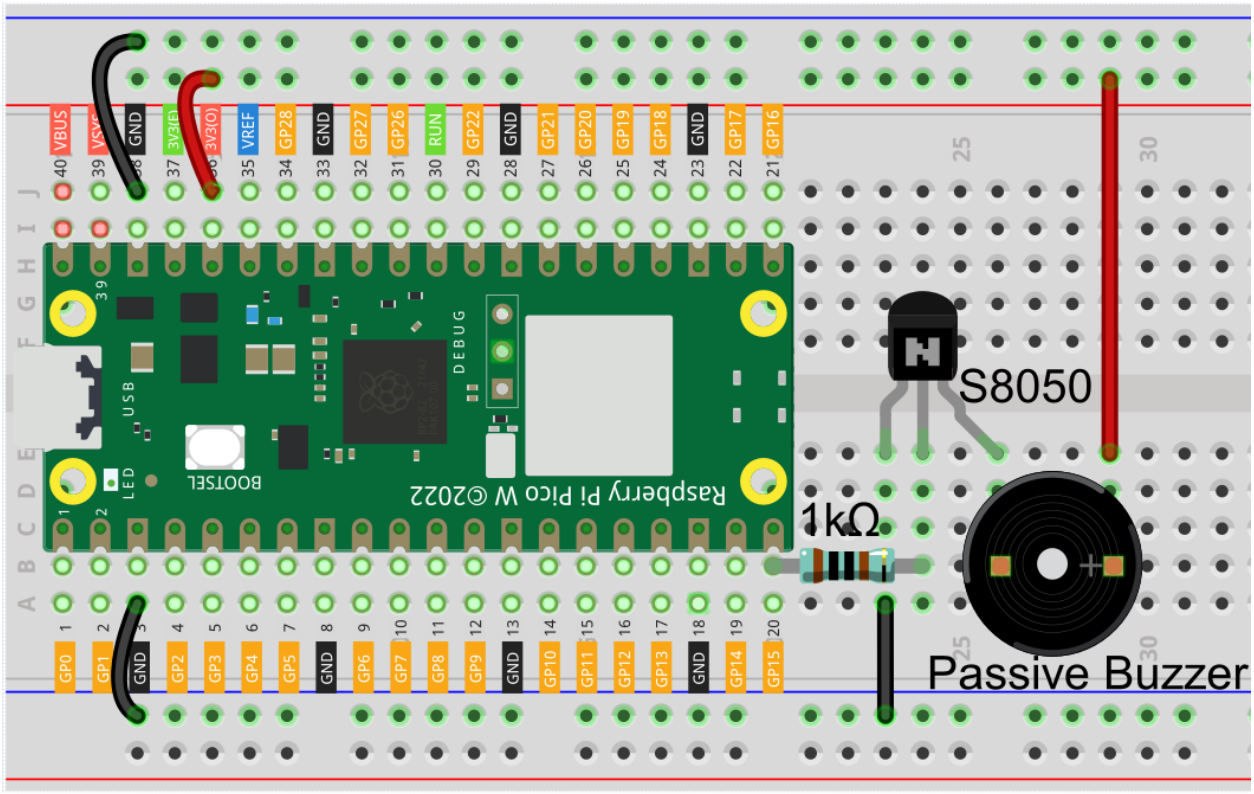
The role of S8050 (NPN transistor) is to amplify the current and make the buzzer sound louder. In fact, you can also connect the buzzer directly to GP15, but you will find that the buzzer sound is smaller.

Wiring



Two buzzers are included in the kit, we use a passive buzzer (one with an exposed PCB on the back).

The buzzer needs a transistor to work, here we use S8050.



Code

Note:

- You can open the file `3.2_custom_tone.ino` under the path of `kepler-kit-main/arduino/3.2_custom_tone`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

How it works?

If the passive buzzer given a digital signal, it can only keep pushing the diaphragm without producing sound.

Therefore, we use the `tone()` function to generate the PWM signal to make the passive buzzer sound.

This function has three parameters:

- **pin**, the GPIO pin that controls the buzzer.
- **frequency**, the pitch of the buzzer is determined by the frequency, the higher the frequency, the higher the pitch.
- **Duration**, the duration of the tone.
- tone

Learn More

We can simulate the specific tone according to the fundamental frequency of the piano, so as to play a complete piece of music.

- Piano key frequencies - Wikipedia

Note:

- You can open the file `3.2_custom_tone_2.ino` under the path of `kepler-kit-main/arduino/3.2_custom_tone_2`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.23 3.3 WS2812 RGB Strip

WS2812 is a intelligent control LED light source that the control circuit and RGB chip are integrated in a package of 5050 components. It internal include intelligent digital port data latch and signal reshaping amplification drive circuit. Also include a precision internal oscillator and a programmable constant current control part, effectively ensuring the pixel point light color height consistent.

The data transfer protocol use single NZR communication mode. After the pixel power-on reset, the DIN port receive data from controller, the first pixel collect initial 24bit data then sent to the internal data latch, the other data which reshaping by the internal signal reshaping amplification circuit sent to the next cascade pixel through the DO port. After transmission for each pixel, the signal to reduce 24bit. pixel adopt auto reshaping transmit technology, making the pixel cascade number is not limited the signal transmission, only depend on the speed of signal transmission.

- *WS2812 RGB 8 LEDs Strip*

Required Components

In this project, we need the following components.

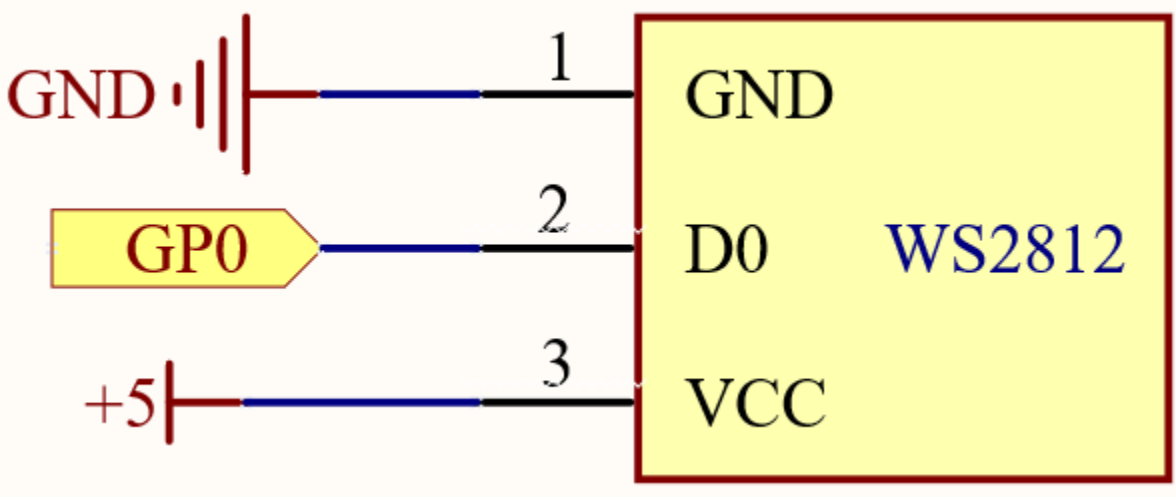
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

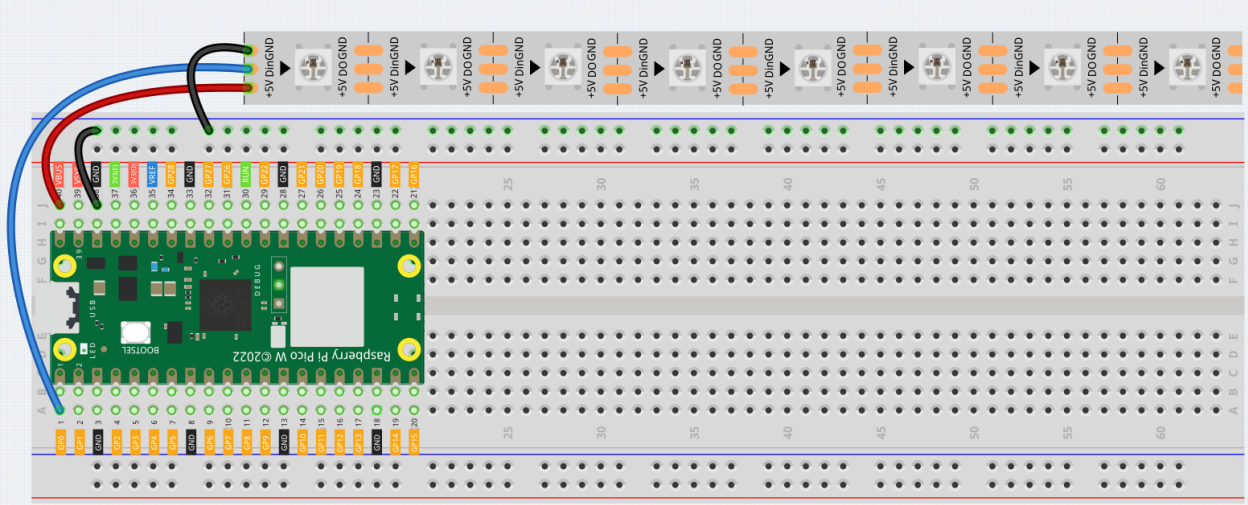
You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*WS2812 RGB 8 LEDs Strip*	1	

Schematic



Wiring



Warning: One thing you need to pay attention to is current.

Although the LED Strip with any number of LEDs can be used in Pico W, the power of its VBUS pin is limited. Here, we will use eight LEDs, which are safe. But if you want to use more LEDs, you need to add a separate power supply.

Code

Note:

- You can open the file `3.3_rgb_led_strip.ino` under the path of `kepler-kit-main/arduino/3.3_rgb_led_strip`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.
- The library `Adafruit_NeoPixel` is used here. Please refer to *Add libraries* for adding it to the Arduino IDE.

Let's select some favorite colors and display them on the RGB LED Strip!

How it works?

Declare a Adafruit_NeoPixel type object, it is connected to `PIXEL_PIN`, there are `PIXEL_COUNT` RGB LEDs on the strip.

```
#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)
```

(continues on next page)

(continued from previous page)

```
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
//   NEO_RGBW    Pixels are wired for RGBW bitstream (NeoPixel RGBW products)
```

Initialize strip object and initialize all pixels to 'off'.

Function

- `strip.begin()` : Initialize NeoPixel strip object (REQUIRED).
- `strip.setPixelColor(index, color)` : Set pixel's color (in RAM), the `color` must be a single 'packed' 32-bit value.
- `strip.Color(red, green, blue)` : Color as a single 'packed' 32-bit value.
- `strip.show()` : Update strip with new contents.

Learn More

We can randomly generate colors and make a colorful flowing light.

Note:

- You can open the file `3.3_rgb_led_strip_flowing.ino` under the path of `kepler-kit-main/arduino/3.3_rgb_led_strip_flowing`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

Or have this WS2812 LED Strip rainbow cycle around the color wheel (range 65535).

Note:

- You can open the file `3.3_rgb_led_strip_rainbow.ino` under the path of `kepler-kit-main/arduino/3.3_rgb_led_strip_rainbow`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

- `strip.getPixelColor(index)` : Query the color of a previously-set pixel.
- `strip.ColorHSV(pixelHue)` : Convert hue, saturation and value into a packed 32-bit RGB color that can be passed to `setPixelColor()` or other RGB-compatible functions.
- `strip.gamma32()` : Provides a "truer" color before assigning to each pixel.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

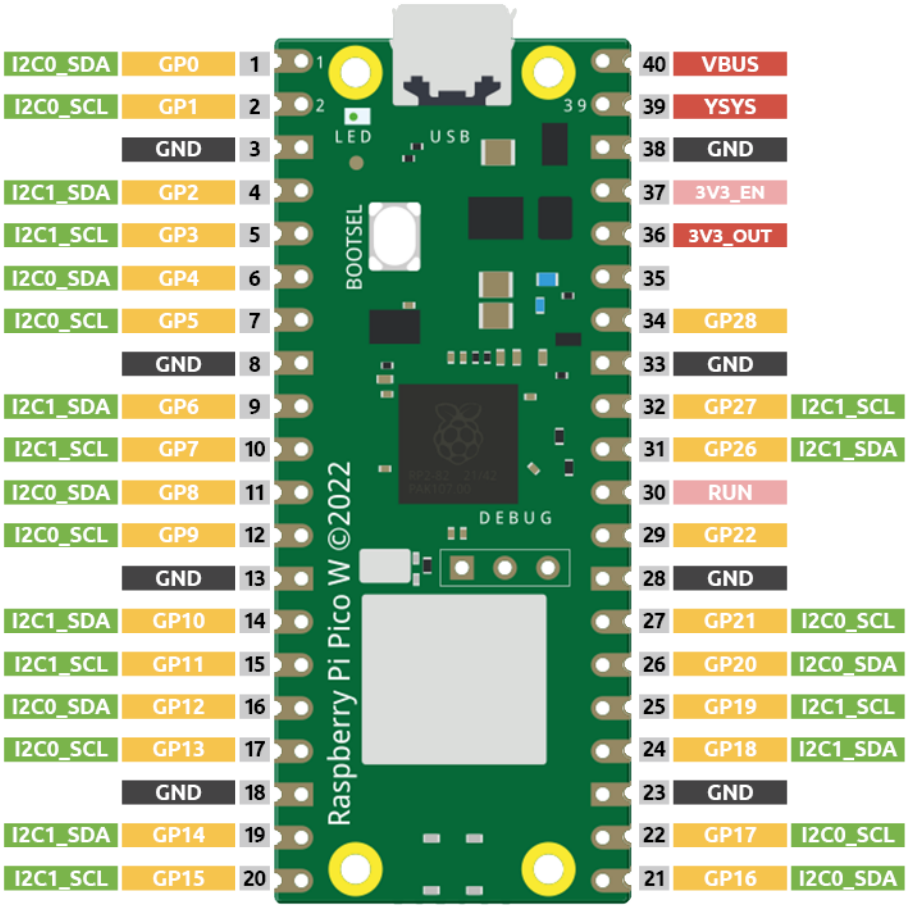
Ready to explore and create with us? Click [] and join today!

6.24 3.4 - Liquid Crystal Display

LCD1602 is a character type liquid crystal display, which can display 32 (16*2) characters at the same time.

As we all know, though LCD and some other displays greatly enrich the man-machine interaction, they share a common weakness. When they are connected to a controller, multiple IOs will be occupied of the controller which has no so many outer ports. Also it restricts other functions of the controller. Therefore, LCD1602 with an I2C bus is developed to solve the problem.

- *I2C LCD1602*
- Inter-Integrated Circuit - Wikipedia



Here we will use the I2C0 interface to control the LCD1602 and display text.

Required Components

In this project, we need the following components.

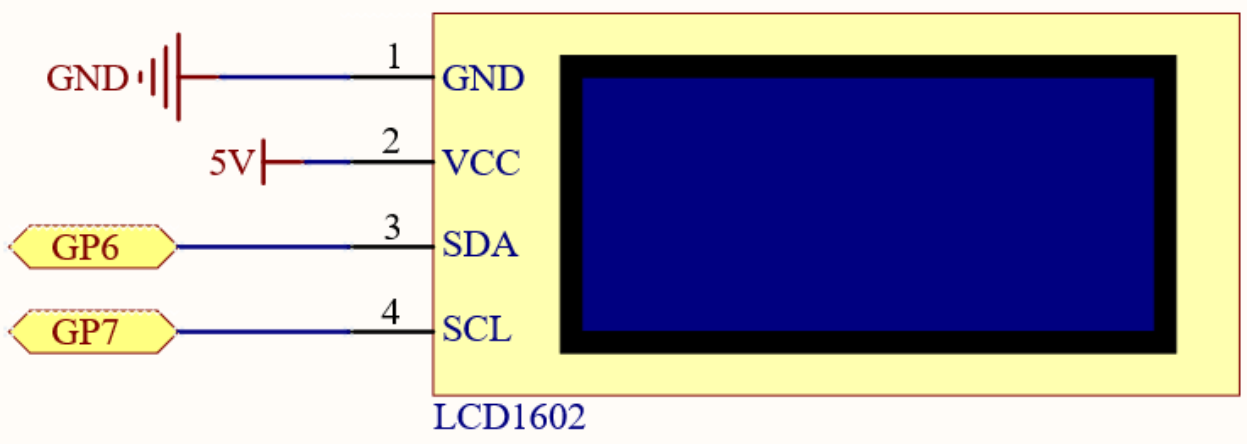
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

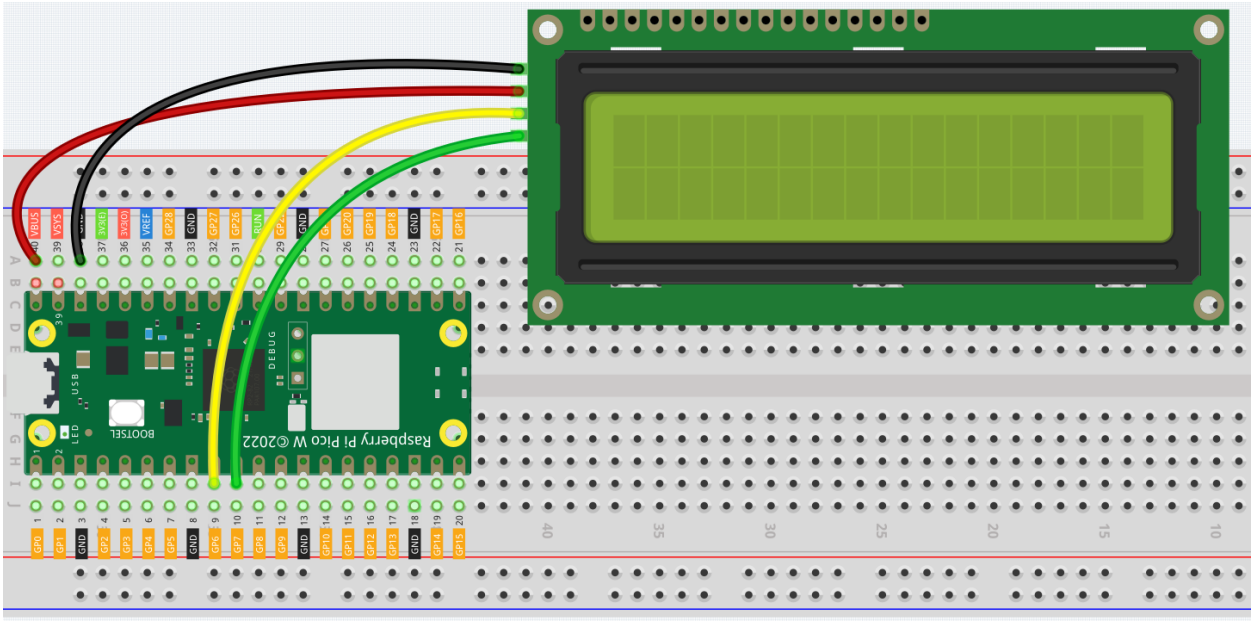
You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*I2C LCD1602*	1	

Schematic



Wiring



Code

Note:

- You can open the file `3.4_liquid_crystal_display.ino` under the path of `kepler-kit-main/arduino/3.4_liquid_crystal_display`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.
- The library `LiquidCrystal_I2C` is used here. Please refer to *Add libraries* for adding it to the Arduino IDE.

After the program runs, you will be able to see two lines of text appear on the LCD in turn, and then disappear.

Note: If the code and wiring are fine, but the LCD still does not display content, you can turn the potentiometer on the back to increase the contrast.

How it works?

By calling the library `LiquidCrystal_I2C.h`, you can easily drive the LCD.

```
#include "LiquidCrystal_I2C.h"
```

Library Functions

```
LiquidCrystal_I2C(uint8_t lcd_Addr,uint8_t lcd_cols,uint8_t lcd_rows)
```

Creates a new instance of the `LiquidCrystal_I2C` class that represents a particular LCD attached to your Arduino board.

- **lcd_AddR**: The address of the LCD defaults to 0x27.
- **lcd_cols**: The LCD1602 has 16 columns.
- **lcd_rows**: The LCD1602 has 2 rows.

```
void init()
```

Initialize the lcd.

```
void backlight()
```

Turn the (optional) backlight on.

```
void nobacklight()
```

Turn the (optional) backlight off.

```
void display()
```

Turn the LCD display on.

```
void nodisplay()
```

Turn the LCD display off quickly.

```
void clear()
```

Clear display, set cursor position to zero.

```
void setCursor(uint8_t col,uint8_t row)
```

Set the cursor position to col,row.

```
void print(data,BASE)
```

Prints text to the LCD.

- **data**: The data to print (char, byte, int, long, or string).
- **BASE (optional)**: The base in which to print numbers: BIN for binary (base 2), DEC for decimal (base 10), OCT for octal (base 8), HEX for hexadecimal (base 16).

Learn More

Upload the codes to the Pico W, the content that you input in the serial monitor will be printed on the LCD.

Note:

- You can open the file `3.4_liquid_crystal_display_2.ino` under the path of `kepler-kit-main/arduino/3.4_liquid_crystal_display_2`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

In addition to reading data from the electronic components, the Pico W can read the data input in the serial port monitor, and you can use `Serial.read()` as the controller of the circuit experiment.

Run the serial communication in `setup()` and set the data rate to 9600.

```
Serial.begin(9600);
```

The state of serial port monitor is judged in `loop()`, and the information processing will be carried out only when the data are received.

```
if (Serial.available() > 0){}
```

Clear the screen.

```
lcd.clear();
```

Reads the input value in the serial port monitor and stores it to the variable incomingByte.

```
char incomingByte = Serial.read();
```

Display each character to the LCD and skip the line-feed character.

```
while (Serial.available() > 0) {
    char incomingByte=Serial.read();
    if(incomingByte==10){break;}// skip the line-feed character
    lcd.print(incomingByte);// display each character to the LCD
}
```

- Serial Read

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.25 3.5 - Small Fan

Now we use the TA6586 to drive the DC motor to make it rotate clockwise and counterclockwise. Since the DC motor requires a relatively large current, for safety reasons, here we use a power module to supply power to the motor.

- *DC Motor*
- *TA6586 - Motor Driver Chip*
- cpn_power_module

Required Components

In this project, we need the following components.

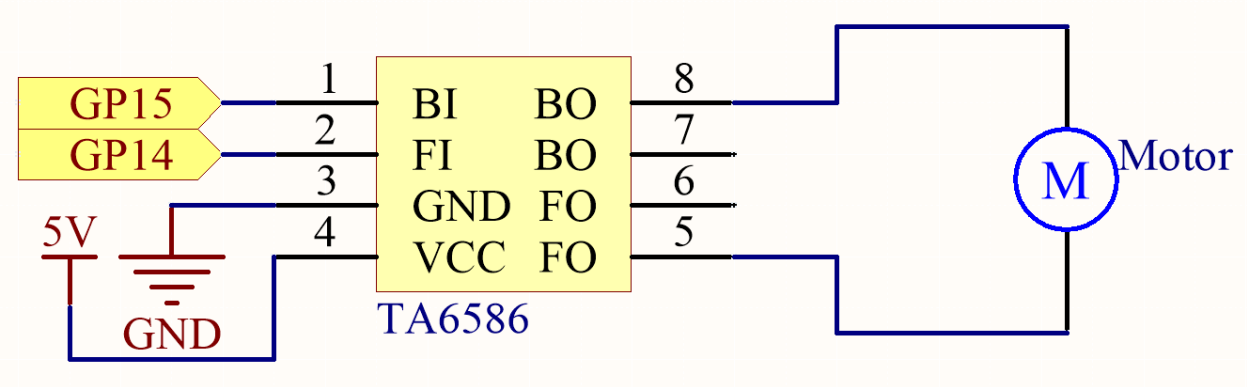
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*TA6586 - Motor Driver Chip*	1	
6	*DC Motor*	1	
7	*Li-po Charger Module*	1	
8	18650 Battery	1	
9	Battery Holder	1	

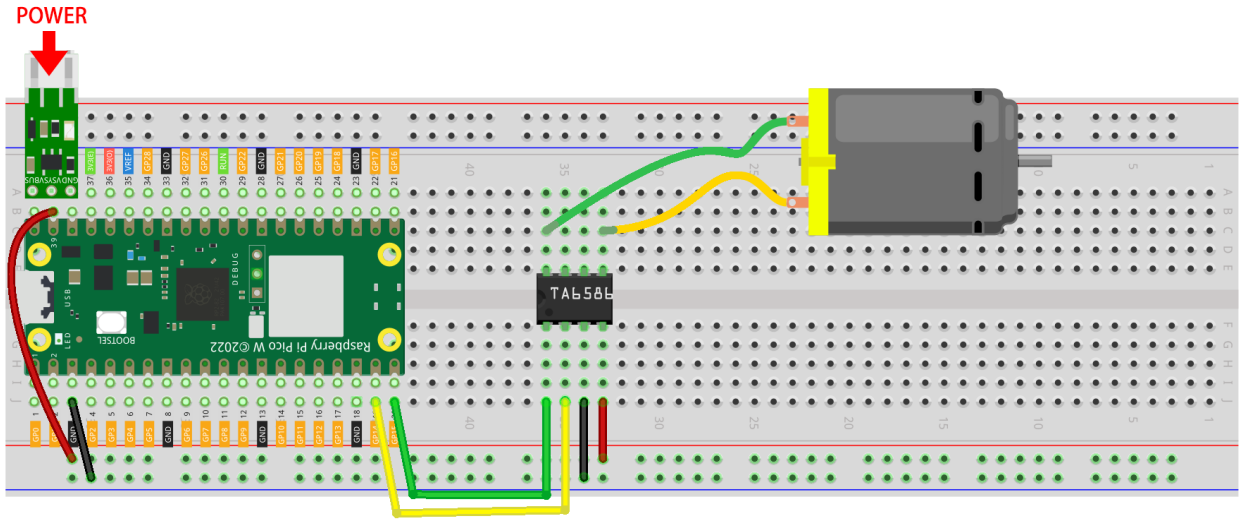
Schematic



Wiring

Note:

- Since DC motors require a high current, we use a Li-po Charger module to power the motor here for safety reasons.
- Make sure your Li-po Charger Module is connected as shown in the diagram. Otherwise, a short circuit will likely damage your battery and circuitry.



Code

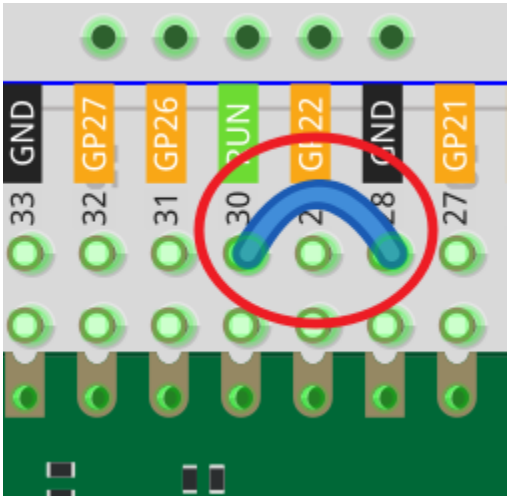
Note:

- You can open the file `3.5_small_fan.ino` under the path of `kepler-kit-main/arduino/3.5_small_fan`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

Once the program is running, the motor will rotate back and forth in a regular pattern.

Note:

- If you can not upload the code again, this time you need to connect the **RUN** pin on the Pico W with a wire to GND to reset it, and then unplug this wire to run the code again.
- This is because the motor is operating with too much current, which may cause the Pico W to disconnect from the computer.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.26 3.6 - Pumping

Small centrifugal pumps are suitable for projects with automatic plant watering. It can also be used to make tiny smart water features.

Its power component is an electric motor, driven in exactly the same way as a normal motor.

- *DC Water Pump*
- *DC Motor*
- *TA6586 - Motor Driver Chip*
- cpn_power_module

Note:

1. Connect the tube to the motor outlet, submerge the pump in water, and then power it on.
2. You need to make sure that the water level is always higher than the motor. Idling may damage the motor due to heat generation and will also generate noise.
3. If you are watering plants, you need to avoid soil being drawn in, as this can clog the pump.
4. If water does not come out of the tube, there may be residual water in the tube blocking the air flow and needs to be drained first.

Required Components

In this project, we need the following components.

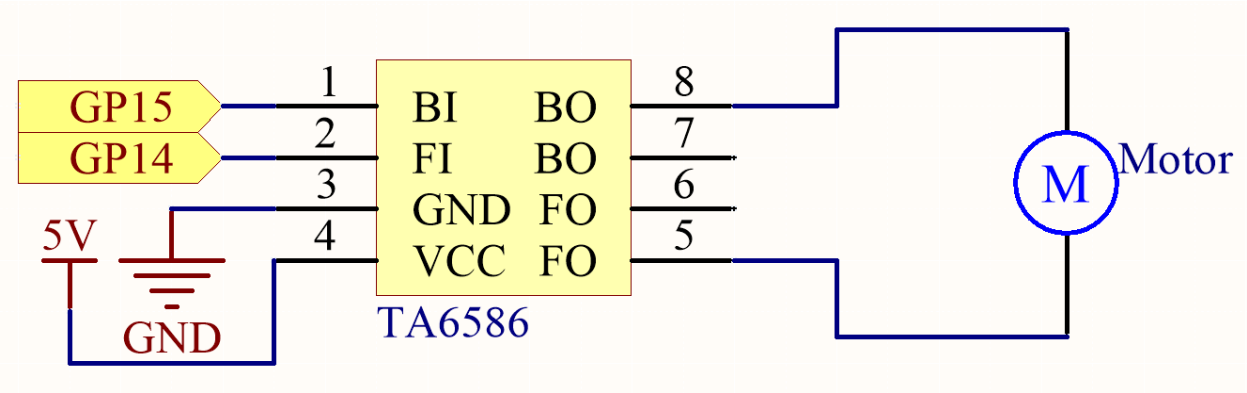
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*TA6586 - Motor Driver Chip*	1	
6	*Li-po Charger Module*	1	
7	18650 Battery	1	
8	Battery Holder	1	
9	*DC Water Pump*	1	

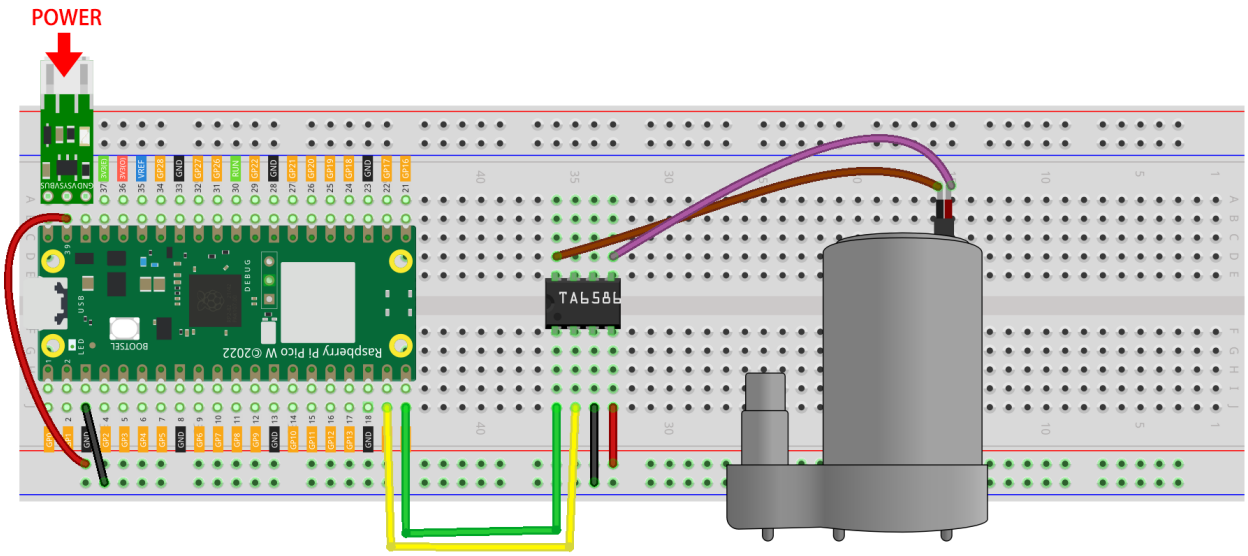
Schematic



Wiring

Note:

- Since pump require a high current, we use a Li-po Charger module to power the motor here for safety reasons.
- Make sure your Li-po Charger Module is connected as shown in the diagram. Otherwise, a short circuit will likely damage your battery and circuitry.



Code

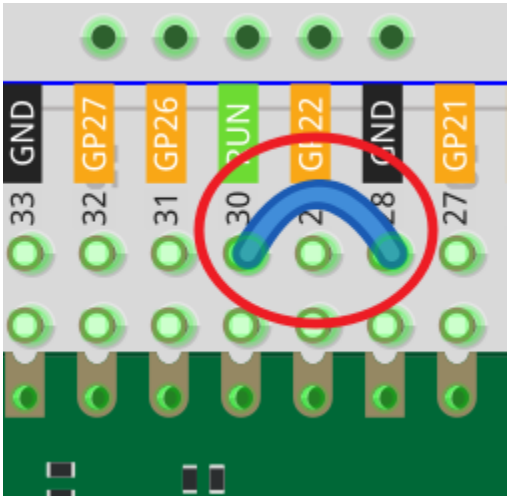
Note:

- You can open the file `3.6_pumping.ino` under the path of `kepler-kit-main/arduino/3.6_pumping`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

After the code is run, the pump starts working and you will see water flowing out of the tube at the same time.

Note:

- If you can not upload the code again, this time you need to connect the **RUN** pin on the Pico W with a wire to GND to reset it, and then unplug this wire to run the code again.
- This is because the motor is operating with too much current, which may cause the Pico W to disconnect from the computer.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.27 3.7 - Swinging Servo

In this kit, in addition to LED and passive buzzer, there is also a device controlled by PWM signal, Servo.

Servo is a position (angle) servo device, which is suitable for those control systems that require constant angle changes and can be maintained. It has been widely used in high-end remote control toys, such as airplanes, submarine models, and remote control robots.

Now, try to make the servo sway!

- *Servo*

Required Components

In this project, we need the following components.

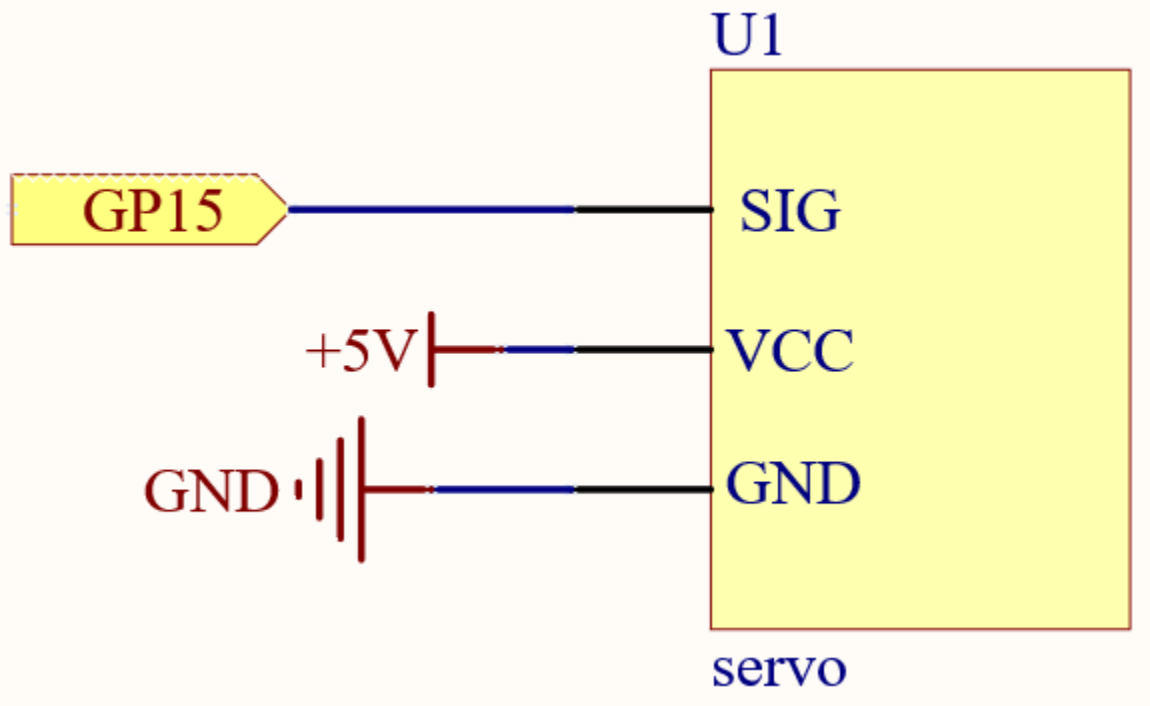
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

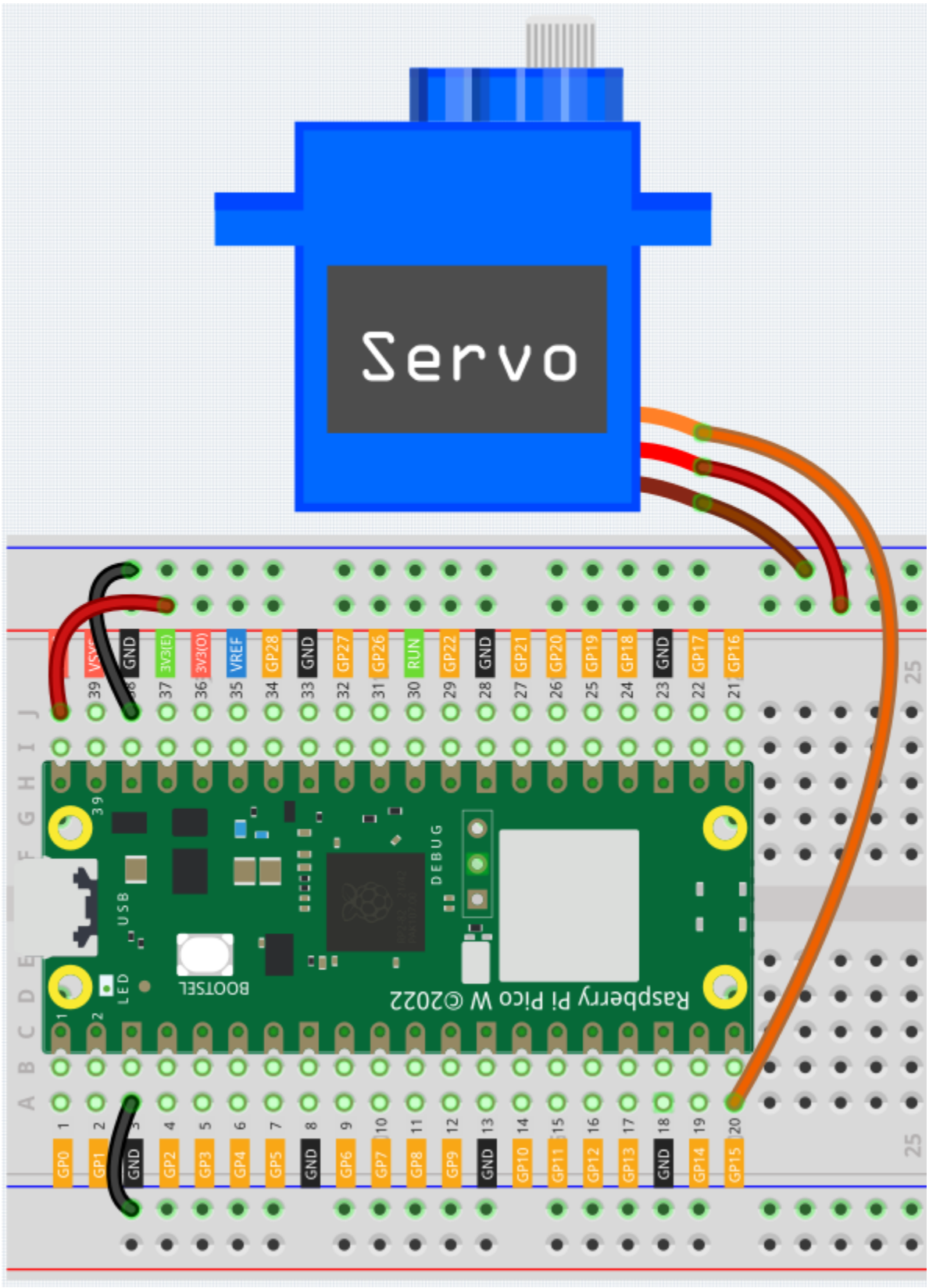
You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUANTITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Servo*	1	

Schematic



Wiring



- Orange wire is signal and connected to GP15.

- Red wire is VCC and connected to VBUS(5V).
- Brown wire is GND and connected to GND.

Code

Note:

- You can open the file `3.7_swinging_servo.ino` under the path of `kepler-kit-main/arduino/3.7_swinging_servo`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

When the program is running, we can see the Servo Arm swinging back and forth from 0° to 180°.

How it works?

By calling the library `Servo.h`, you can drive the servo easily.

```
#include <Servo.h>
```

Library Functions

```
Servo
```

Create **Servo** object to control a servo.

```
uint8_t attach(int pin);
```

Turn a pin into a servo driver. Calls pinMode. Returns 0 on failure.

```
void detach();
```

Release a pin from servo driving.

```
void write(int value);
```

Set the angle of the servo in degrees, 0 to 180.

```
int read();
```

Return that value set with the last write().

```
bool attached();
```

Return 1 if the servo is currently attached.

4. Controller

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.28 4.1 - Toggle the Joystick

If you play a lot of video games, then you should be very familiar with the Joystick. It is usually used to move the character around, rotate the screen, etc.

The principle behind Joystick's ability to allow the computer to read our actions is very simple. It can be thought of as consisting of two potentiometers that are perpendicular to each other. These two potentiometers measure the analog value of the joystick vertically and horizontally, resulting in a value (x,y) in a planar right-angle coordinate system.

The joystick of this kit also has a digital input, which is activated when the joystick is pressed.

- *Joystick Module*

Required Components

In this project, we need the following components.

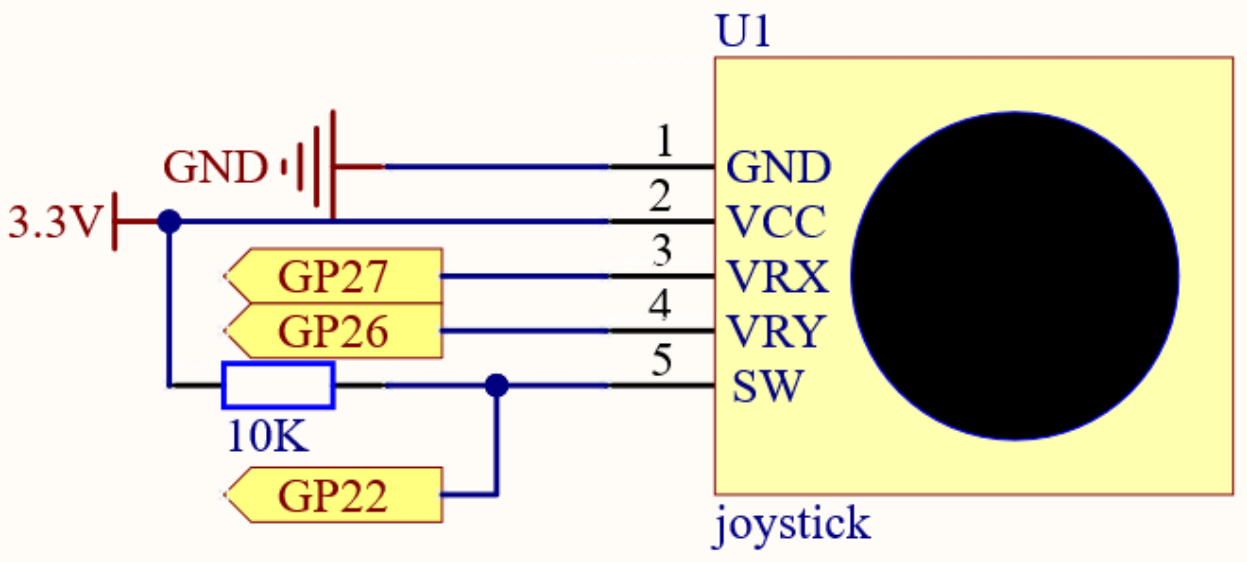
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

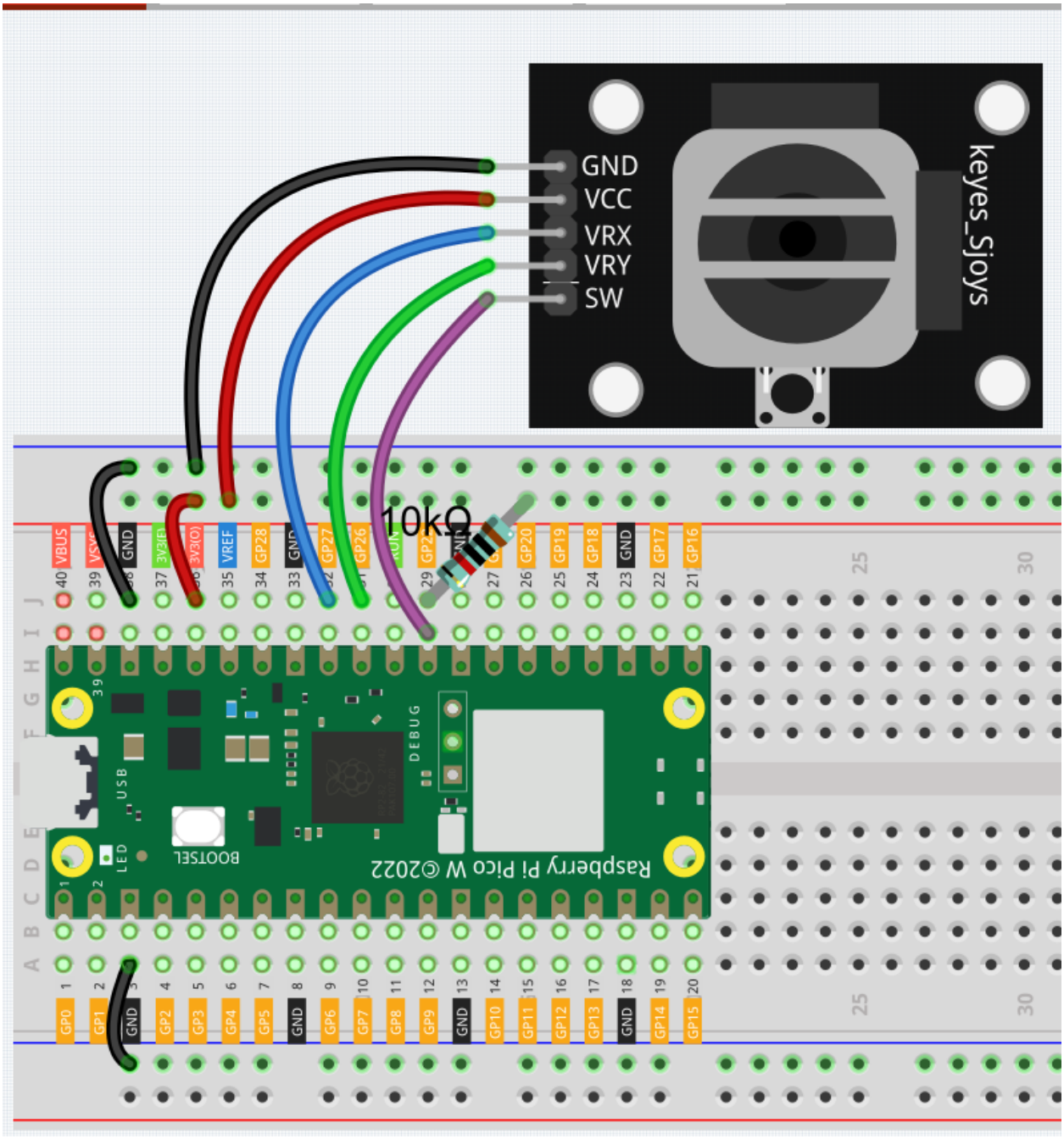
SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Joystick Module*	1	

Schematic



The SW pin is connected to a 10K pull-up resistor, the reason is to be able to get a stable high level on the SW pin (Z axis) when the joystick is not pressed; otherwise the SW is in a suspended state and the output value may vary between 0/1.

Wiring



Code

Note:

- You can open the file `4.1_toggle_the_joyostick.ino` under the path of `kepler-kit-main/arduino/4.1_toggle_the_joyostick`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

After the program runs, the Shell prints out the x,y,z values of joystick.

- The x-axis and y-axis values are analog values that vary from 0 to 65535.
- The Z-axis is a digital value with a status of 1 or 0.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.29 4.2 4x4 Keypad

The 4x4 keyboard, also known as the matrix keyboard, is a matrix of 16 keys excluded in a single panel.

The keypad can be found on devices that mainly require digital input, such as calculators, TV remote controls, push-button phones, vending machines, ATMs, combination locks, and digital door locks.

In this project, we will learn how to determine which key is pressed and get the related key value.

- *4x4 Keypad*
- E.161 - Wikipedia

Required Components

In this project, we need the following components.

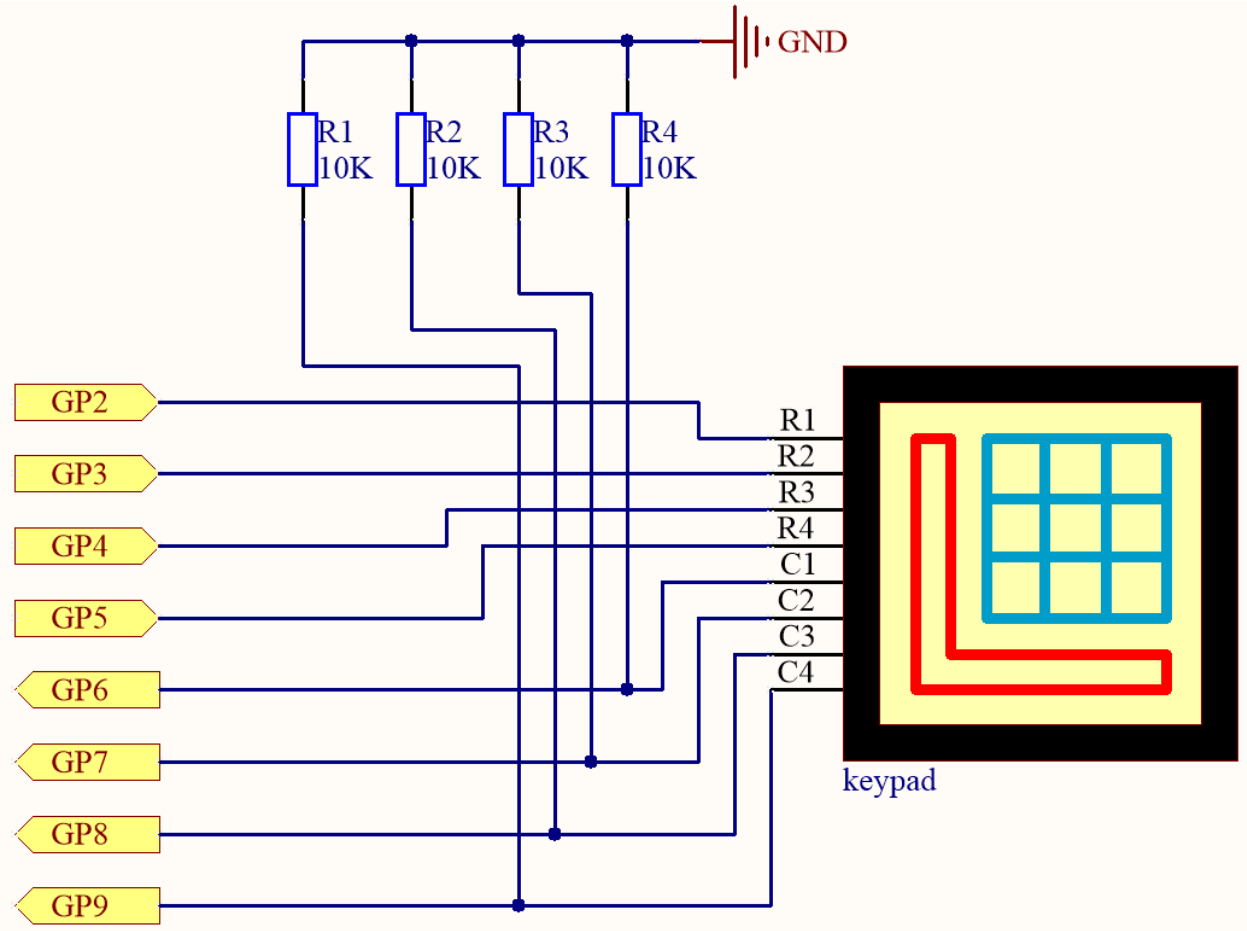
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	4(10K)	
6	*4x4 Keypad*	1	

Schematic

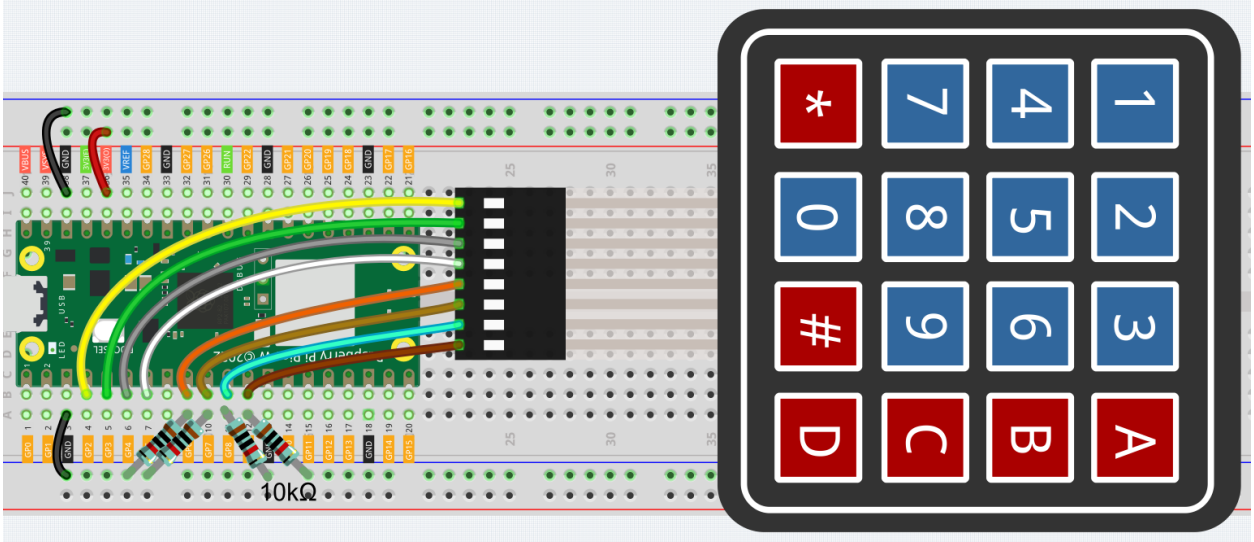


4 pull-down resistors are connected to each of the columns of the matrix keyboard, so that G6 ~ G9 get a stable low level when the keys are not pressed.

The rows of the keyboard (G2 ~ G5) are programmed to go high; if one of G6 ~ G9 is read high, then we know which key is pressed.

For example, if G6 is read high, then numeric key 1 is pressed; this is because the control pins of numeric key 1 are G2 and G6, when numeric key 1 is pressed, G2 and G6 will be connected together and G6 is also high.

Wiring



To make the wiring easier, in the above diagram, the column row of the matrix keyboard and the 10K resistors are inserted into the holes where G6 ~ G9 are located at the same time.

Code

Note:

- You can open the file `4.2_4x4_keypad.ino` under the path of `kepler-kit-main/arduino/4.2_4x4_keypad`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.
- The library `Keypad` is used here. Please refer to *Add libraries* for adding it to the Arduino IDE.

After the program runs, the Shell will print out the keys you pressed on the Keypad.

How it works

By calling the `Keypad.h` library, you can easily use Keypad.

```
#include <Keypad.h>
```

Library Functions:

```
Keypad(char *userKeymap, byte *row, byte *col, byte numRows, byte numCols)
```

Initializes the internal keymap to be equal to `userKeymap`.

`userKeymap`: The symbols on the buttons of the keypads.

`row`, `col`: Pin configuration.

`numRows`, `numCols`: Keypad sizes.

```
char getKey()
```

Returns the key that is pressed, if any. This function is non-blocking.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.30 4.3 - Electrode Keyboard

The MPR121 is a good choice when you want to add a large number of touch switches to your project. It has electrodes that can be extended with conductors. If you connect the electrodes to a banana, you can turn the banana into a touch switch.

- *MPR121 Module*

Required Components

In this project, we need the following components.

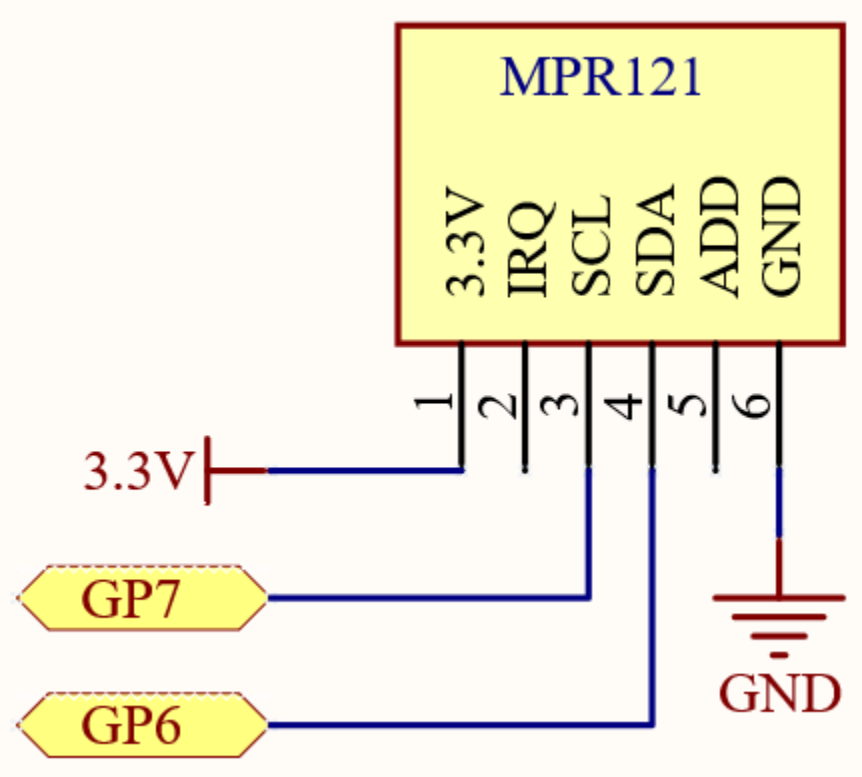
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

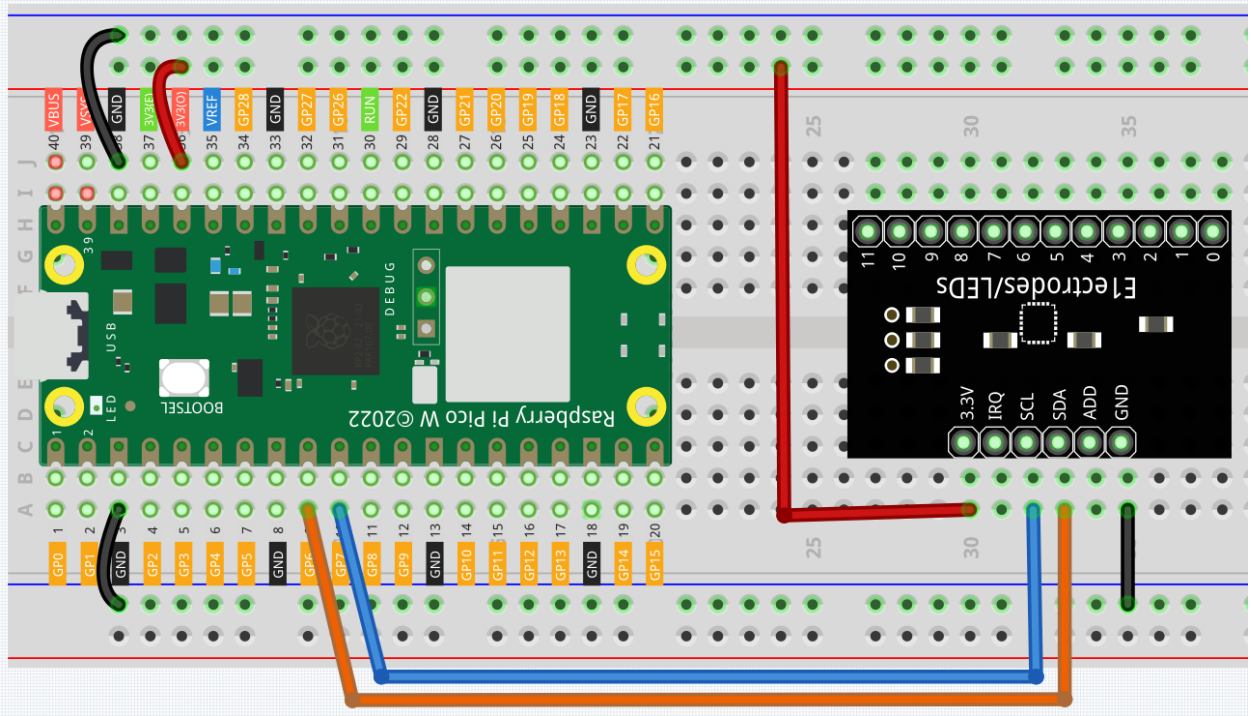
You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUANTITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*MPR121 Module*	1	

Schematic



Wiring



Code

Note:

- You can open the file `4.3_electrode_keyboard.ino` under the path of `kepler-kit-main/arduino/4.3_electrode_keyboard`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.
- The two libraries `Adafruit_MPR121` and `Adafruit_BusIO` are used here. Please refer to *Add libraries* for adding it to the Arduino IDE.

After the program runs, you can touch the twelve electrodes on the MPR121 module by hand and the touch status of these electrodes will be recorded in a 12-bit Boolean type array that will be printed on the serial monitor. If the first and eleventh electrodes are touched, `100000000010` is printed.

You can extend the electrodes by connecting other conductors such as fruit, wire, foil, etc. This will give you more ways to trigger these electrodes.

How it works?

Initialize the `MPR121` object. At this point the state of the module's electrodes will be recorded as initial values. If you extend the electrodes, you need to rerun the example to reset the initial values.

```
#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!");
}
```

Gets the value of the current electrode, it will get a 12-bit binary value. If you touch the first and the eleventh electrode, it gets `100000000010`.

```
// 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;
}
```

If a change in electrode state is detected, the values of `currtouched` are stored in the `touchStates[12]` array bit by bit. Finally, the array is printed.

```
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]);
    }
    Serial.println();
}
```

5. Microchip

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.31 5.1 Microchip - 74HC595

Integrated circuit (integrated circuit) is a kind of miniature electronic device or component, which is represented by the letter "IC" in the circuit.

A certain process is used to interconnect the transistors, resistors, capacitors, inductors and other components and wiring required in a circuit, fabricate on a small or several small semiconductor wafers or dielectric substrates, and then package them in a package , it has become a micro-structure with the required circuit functions; all of the components have been structured as a whole, making electronic components a big step towards micro-miniaturization, low power consumption, intelligence and high reliability.

The inventors of integrated circuits are Jack Kilby (integrated circuits based on germanium (Ge)) and Robert Norton Noyce (integrated circuits based on silicon (Si)).

This kit is equipped with an IC, 74HC595, which can greatly save the use of GPIO pins. Specifically, it can replace 8 pins for digital signal output by writing an 8-bit binary number.

- Binary number - Wikipedia
- *74HC595*

Required Components

In this project, we need the following components.

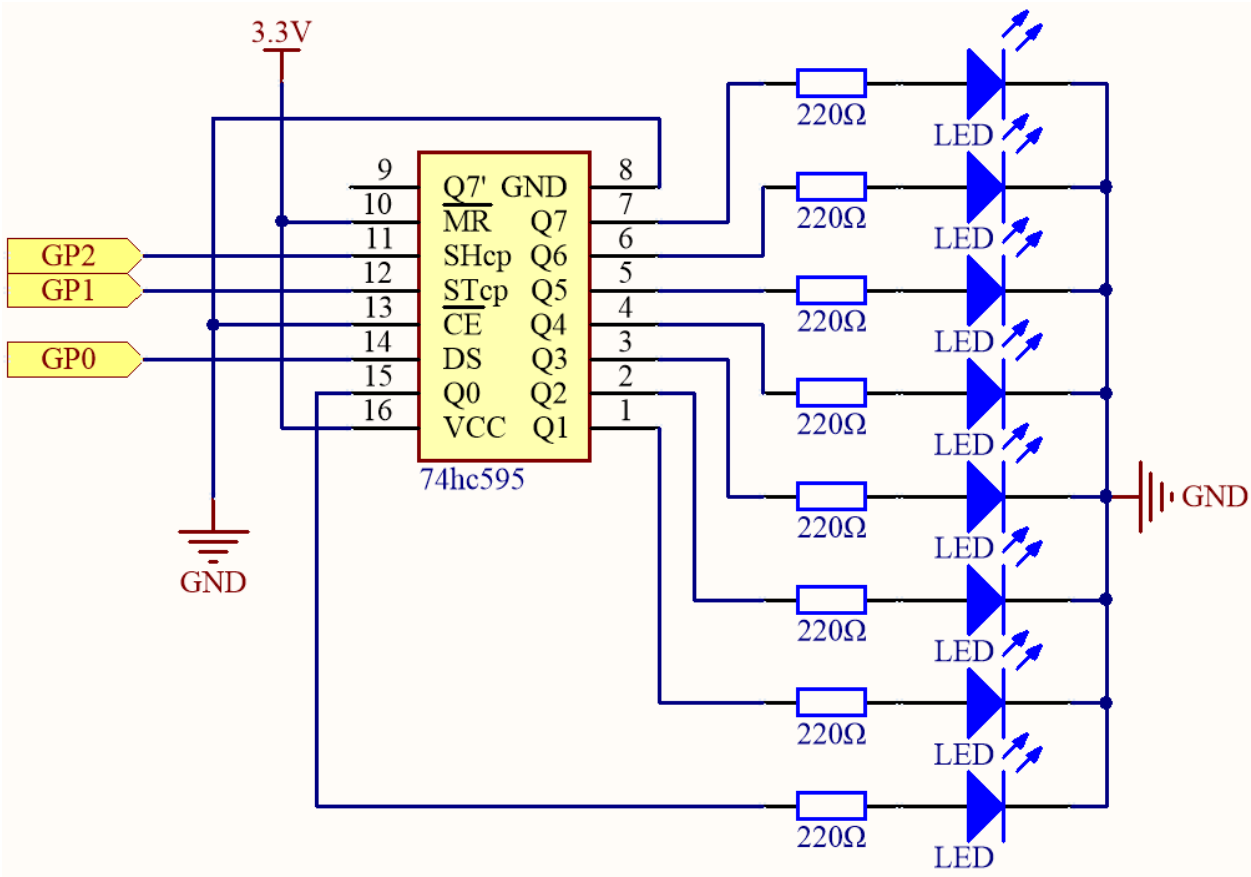
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	8(220)	
6	*LED*	8	
7	*74HC595*	1	

Schematic

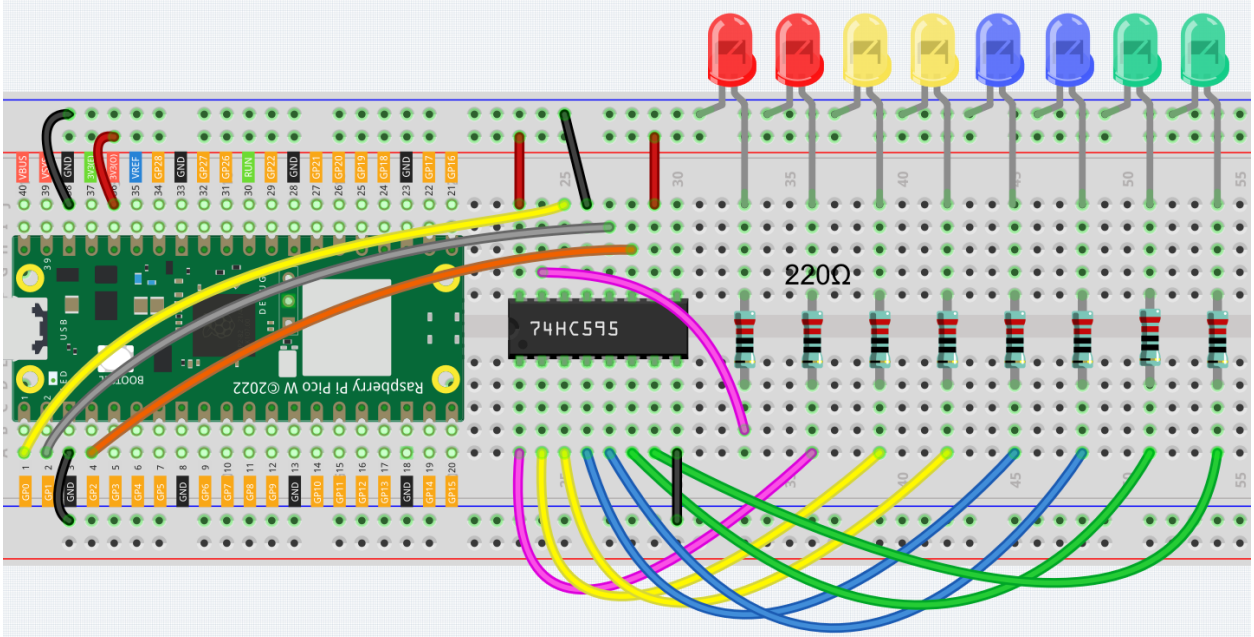


- When MR (pin10) is high level and OE (pin13) is low level, data is input in the rising edge of SHcp and goes to

the memory register through the rising edge of SHcp.

- If the two clocks are connected together, the shift register is always one pulse earlier than the memory register.
- There is a serial shift input pin (Ds), a serial output pin (Q) and an asynchronous reset button (low level) in the memory register.
- The memory register outputs a Bus with a parallel 8-bit and in three states.
- When OE is enabled (low level), the data in memory register is output to the bus(Q0 ~ Q7).

Wiring



Code

Note:

- You can open the file `5.1_microchip_74hc595.ino` under the path of `kepler-kit-main/arduino/5.1_microchip_74hc595`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

When the program is running, you can see the LEDs turning on one after another.

How it works?

Declare an array, store several 8 bit binary numbers that are used to change the working state of the eight LEDs controlled by 74HC595.

```
int datArray[] = {0b00000000, 0b00000001, 0b00000011, 0b00000111, 0b00001111, 0b00011111,
↪ 0b00111111, 0b01111111, 0b11111111};
```

Set STcp to low level first and then high level. It will generate a rising edge pulse of STcp.

```
digitalWrite(STcp,LOW);
```

`shiftOut()` is used to shift out a byte of data one bit at a time, which means to shift a byte of data in datArray[num] to the shifting register with the DS pin. MSBFIRST means to move from high bits.

```
shiftOut(DS,SHcp,MSBFIRST,datArray[num]);
```

After `digitalWrite(STcp,HIGH)` is run, the STcp will be at the rising edge. At this time, the data in the shift register will be moved to the memory register.

```
digitalWrite(STcp,HIGH);
```

A byte of data will be transferred into the memory register after 8 times. Then the data of memory register are output to the bus (Q0-Q7). For example, shiftout `B00000001` will light up the LED controlled by Q0 and turn off the LED controlled by Q1~Q7.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

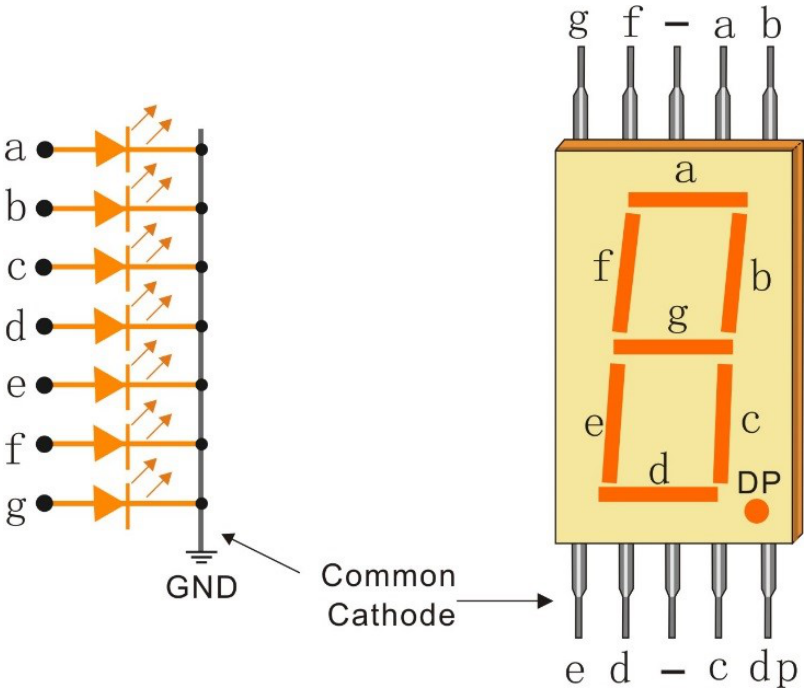
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.32 5.2 - Number Display

LED Segment Display can be seen everywhere in life. For example, on an air conditioner, it can be used to display temperature; on a traffic indicator, it can be used to display a timer.

The LED Segment Display is essentially a device packaged by 8 LEDs, of which 7 strip-shaped LEDs form an “8” shape, and there is a slightly smaller dotted LED as a decimal point. These LEDs are marked as a, b, c, d, e, f, g, and dp. They have their own anode pins and share cathodes. Their pin locations are shown in the figure below.



This means that it needs to be controlled by 8 digital signals at the same time to fully work and the 74HC595 can do this.

- *7-segment Display*

Required Components

In this project, we need the following components.

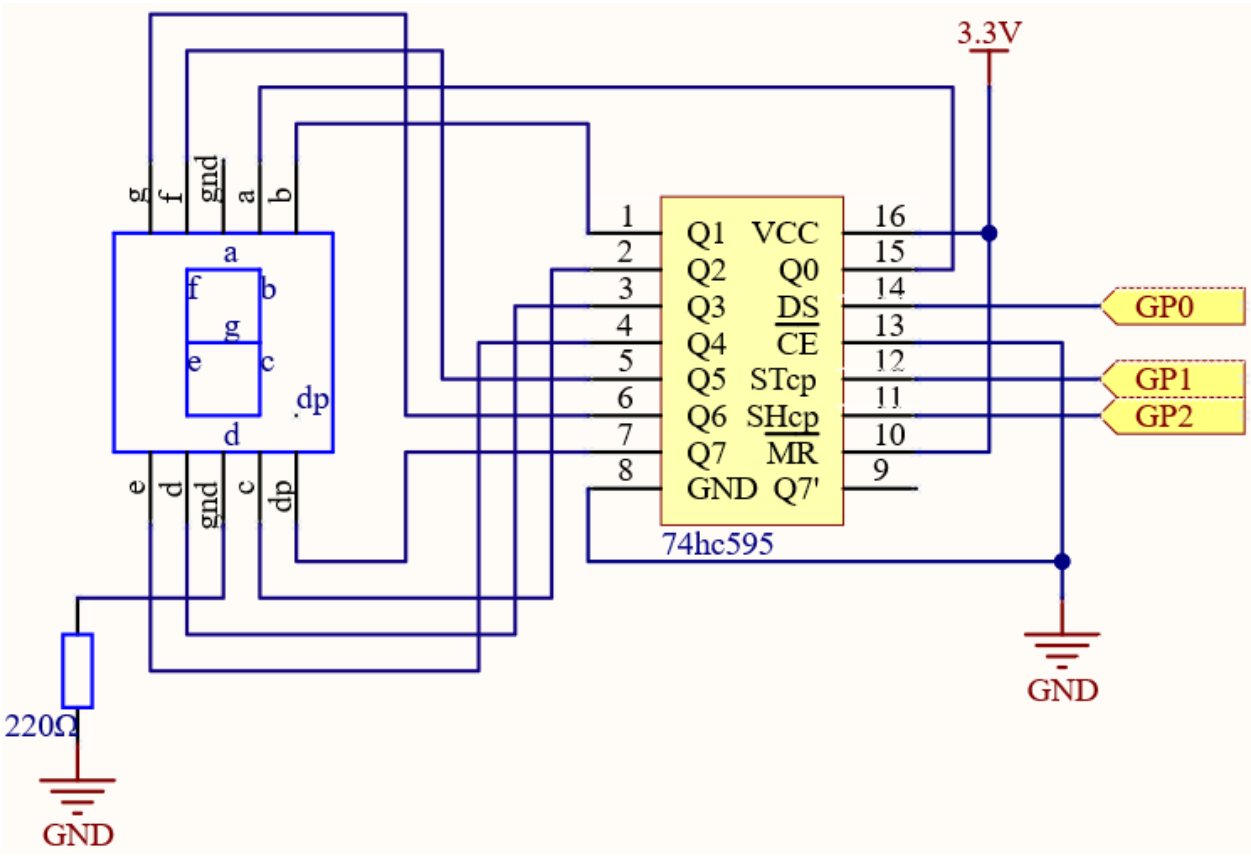
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(220)	
6	*7-segment Display*	1	
7	*74HC595*	1	

Schematic



Wiring

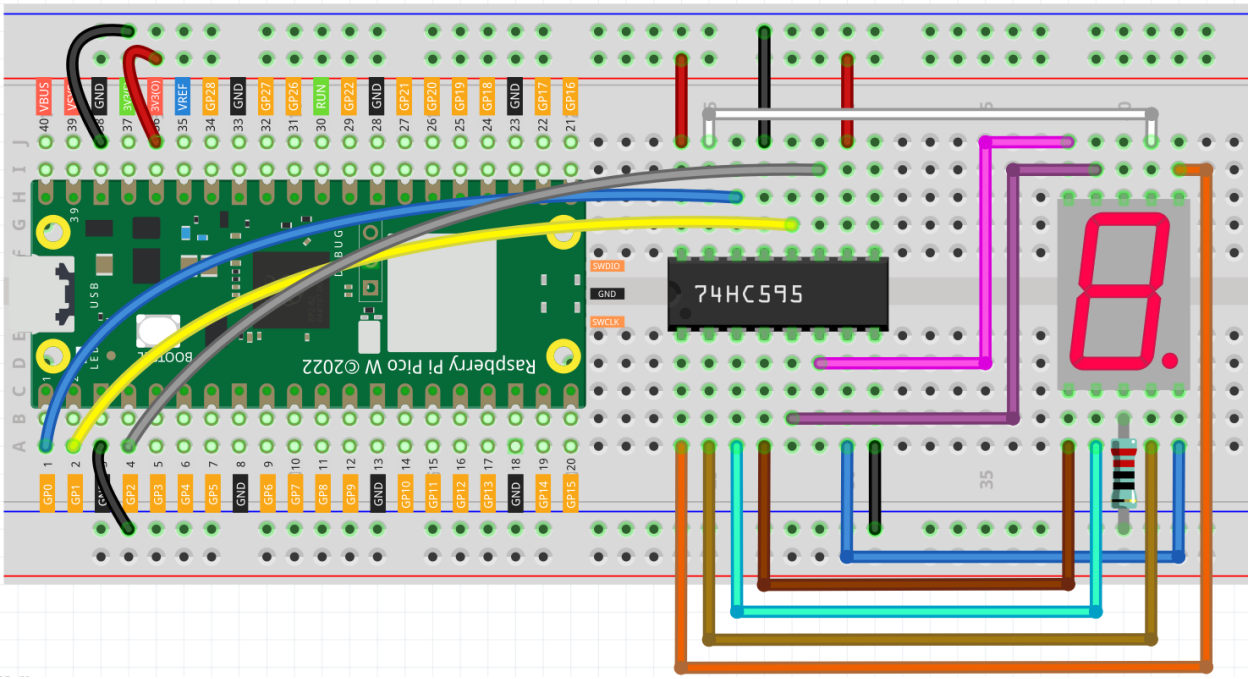


Table 1: Wiring

74HC595	*LED* Segment Display
Q0	a
Q1	b
Q2	c
Q3	d
Q4	e
Q5	f
Q6	g
Q7	dp

Code

Note:

- You can open the file `5.2_number_display.ino` under the path of `kepler-kit-main/arduino/5.2_number_display`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

When the program is running, you will be able to see the LED Segment Display display 0~9 in sequence.

How it works?

`shiftOut()` will make 74HC595 output 8 digital signals. It outputs the last bit of the binary number to Q0, and the output of the first bit to Q7. In other words, writing the binary number "00000001" will make Q0 output high level and Q1~Q7 output low level.

Suppose that the 7-segment Display display the number "1", we need to write a high level for b, c, and write a low level for a, d, e, f, g, and dg. That is, the binary number "00000110" needs to be written. For readability, we will use hexadecimal notation as "0x06".

- Hexadecimal
- BinaryHex Converter

Similarly, we can also make the LED Segment Display display other numbers in the same way. The following table shows the codes corresponding to these numbers.

Table 2: Glyph Code

Numbers	Binary Code	Hex Code
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

Write these codes into `shiftOut()` to make the LED Segment Display display the corresponding numbers.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.33 5.3 - Time Counter

4-Digit 7-segment display consists of four 7- segment displays working together.

The 4-digtal 7-segment display works independently. It uses the principle of human visual persistence to quickly display the characters of each 7-segment in a loop to form continuous strings.

For example, when "1234" is displayed on the display, "1" is displayed on the first 7-segment, and "234" is not displayed. After a period of time, the second 7-segment shows "2", the 1st 3th 4th of 7-segment does not show, and so on, the four digital display show in turn. This process is very short (typically 5ms), and because of the optical afterglow effect and the principle of visual residue, we can see four characters at the same time.

Required Components

In this project, we need the following components.

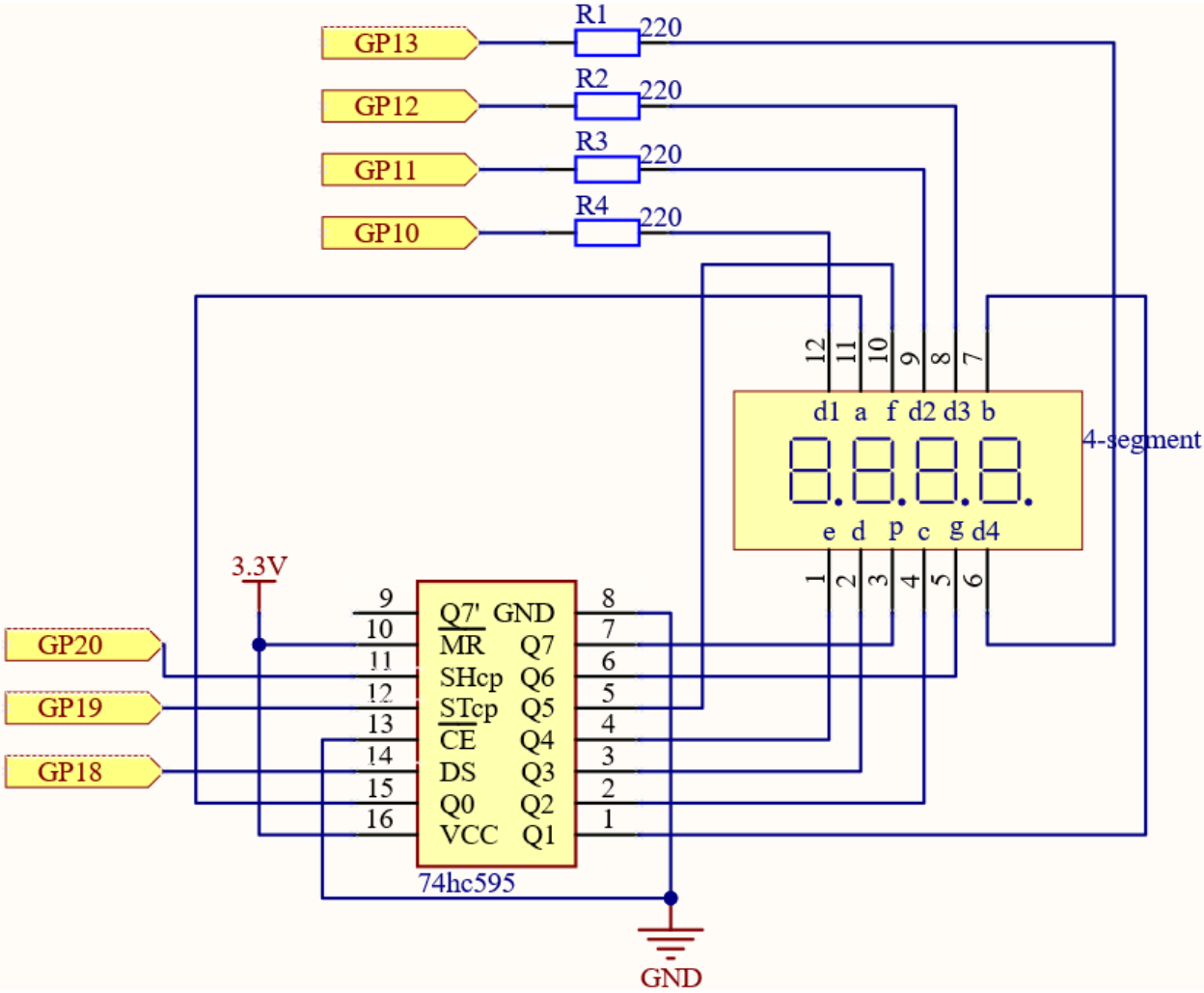
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	4(220)	
6	*4-Digit 7-Segment Display*	1	
7	*74HC595*	1	

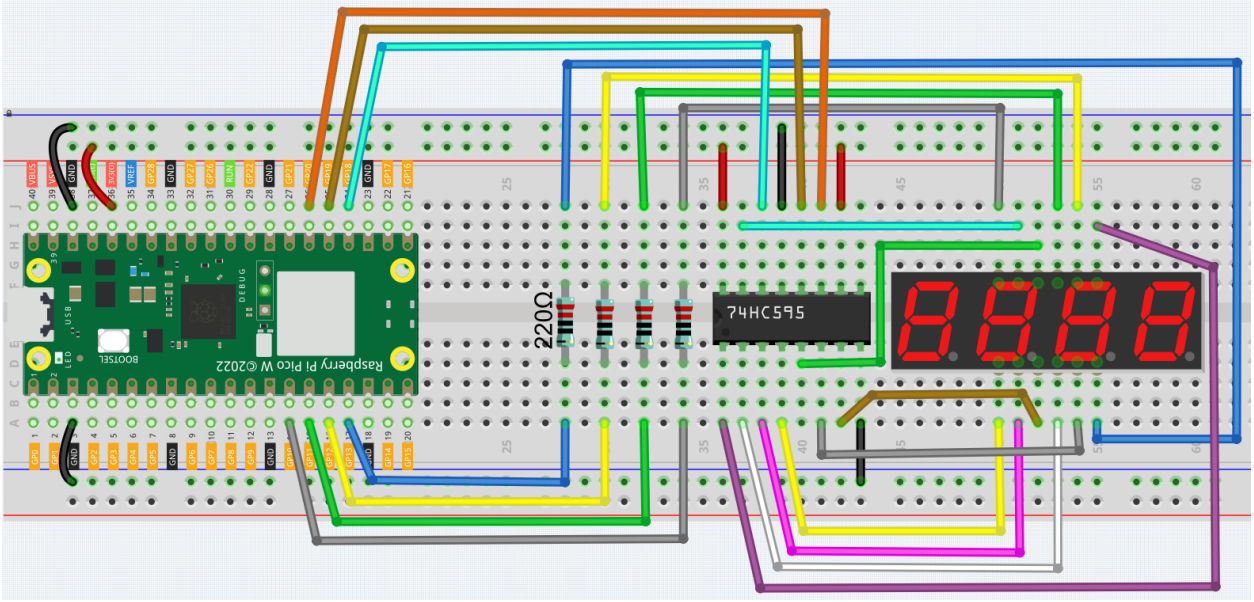
Schematic



Here the wiring principle is basically the same as *5.1 Microchip - 74HC595*, the only difference is that Q0-Q7 are connected to the a ~ g pins of the 4-digit 7-segment display.

Then G10 ~ G13 will select which 7-segment display to work.

Wiring



Code

Note:

- You can open the file `5.3_time_counter.ino` under the path of `kepler-kit-main/arduino/5.3_time_counter`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

After the program is run, you will see the 4-digit 7-segment display become a counter and the number increases by 1 per second.

How it works?

Writing signals to each 7-segment display is done in the same way as *5.2 - Number Display*, using the `hc595_shift()` function. The core point of the 4-digit 7-segment display is to selectively activate each 7-segment display. The code associated with this is as follows.

```
const int placePin[4] = {13,12,11,10};

void setup ()
{
    for (int i = 0; i<4;i++){
        pinMode(placePin[i],OUTPUT);
    }
}

void loop()
{
    pickDigit(0);
    hc595_shift(count%10/1);

    pickDigit(1);
```

(continues on next page)

(continued from previous page)

```
    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);
}
```

Here, four pins (GP10, GP11, GP12, GP13) are used to control each bit of the 4-digit 7-segment display individually. When the status of these pins is `LOW`, the corresponding 7-segment display is active; when the status is `HIGH`, the 7-segment display does not work.

Here the `pickDigit(digit)` function is used to unable all 7-segment displays and then enable a particular digit individually. After that, `hc595_shift()` is used to write the corresponding 8 bits code for the 7-segment display.

The 4-digit 7-segment display needs to be continuously activated in turn so that we can see it display four digits, which means that the main program cannot easily add code that would affect the timing.

However, we need to add a timing function to this example, if we add a `delay (1000)`, we will be able to detect the illusion of its four 7-segment displays working at the same time, exposing the fact that only one 7-segment display at a time to light.

Then, using the `millis()` function is an excellent way to do this.

```
void setup ()
{
    timerStart = millis();
}

void loop()
{
    unsigned int count = (millis()-timerStart)/1000;
}
```

The `millis()` function gets the number of milliseconds that have passed since the current program was started. We record the first time value as `timerStart`;

then when we need to get the time again, we call the `millis()` function again and subtract `timerStart` from the value to get how long the program has been running.

Finally, convert this time value and let the 4-digit 7-segment display to display it.

- millis()

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.34 5.4 - 8x8 Pixel Graphics

ED matrix is a low-resolution dot-matrix display. it uses an array of light-emitting diodes as pixels for patterned displays.

They are bright enough to be visible in outdoor sunlight, and you can see them on some stores, billboards, signs, and variable message displays (such as those on public transit vehicles).

Used in this kit is an 8x8 dot matrix with 16 pins. Their anodes are connected in rows and their cathodes are connected in columns (at the circuit level), which together control these 64 LEDs.

To light the first LED, you should provide a high level for Row1 and a low level for Col1. To light the second LED, it should provide a high level for Row1, a low level for Col2, and so on. By controlling the current through each pair of rows and columns, each LED can be controlled individually to display characters or pictures.

- *LED Dot Matrix*
- *74HC595*

Required Components

In this project, we need the following components.

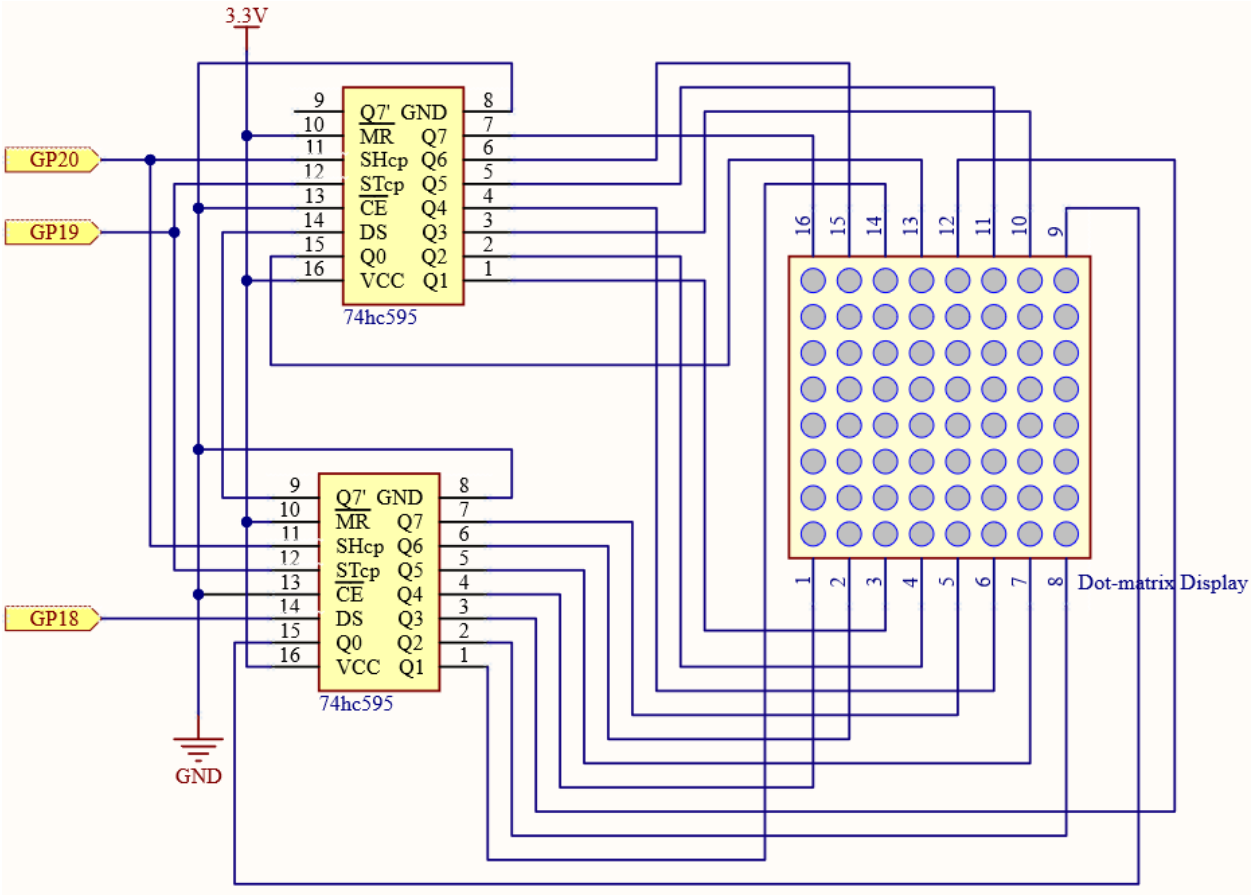
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUANTITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*LED Dot Matrix*	1	
6	*74HC595*	2	

Schematic



The 8x8 dot matrix is controlled by two 74HC595 chips, one controlling the rows and one controlling the columns, while these two chips share G18~G20, which can greatly save the I/O ports of the Pico W board.

Pico W needs to output a 16-bit binary number at a time, the first 8 bits are given to the 74HC595 which controls the rows, and the last 8 bits are given to the 75HC595 which controls the columns, so that the dot matrix can display a specific pattern.

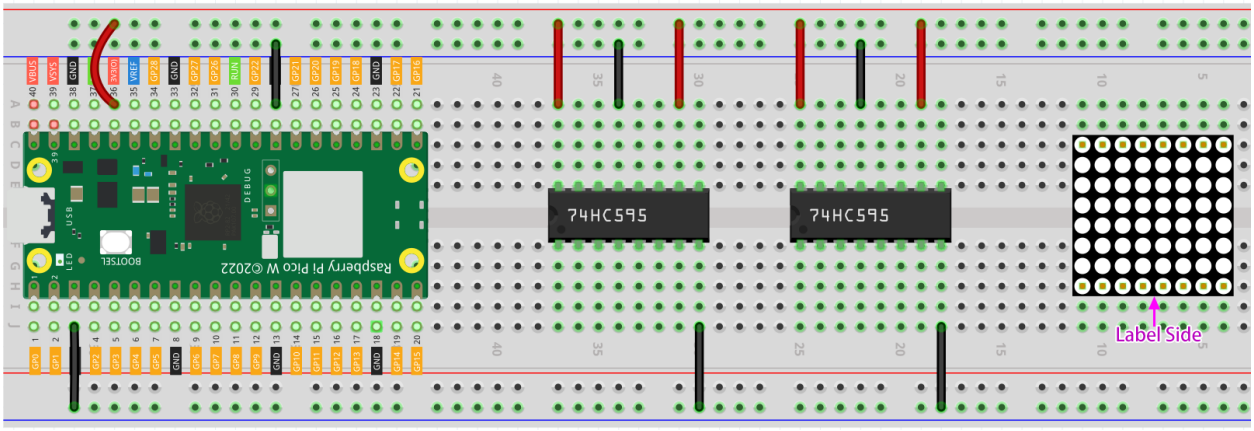
Q7': Series output pin, connected to DS of another 74HC595 to connect multiple 74HC595s in series.

Wiring

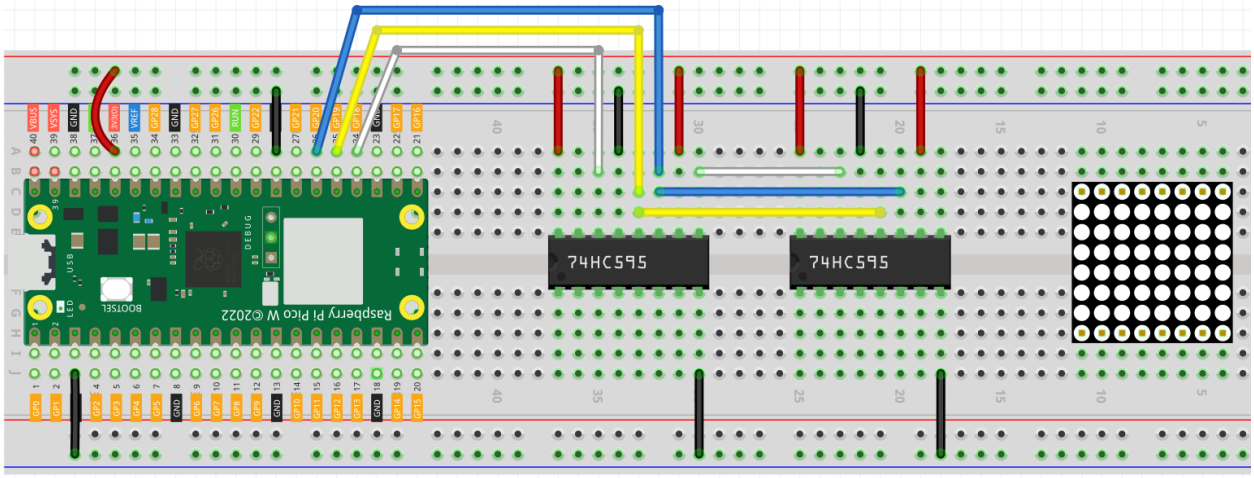
Build the circuit. Since the wiring is complicated, let's make it step by step.

Step 1: First, insert the Pico W, the LED dot matrix and two 74HC595 chips into breadboard. Connect the 3.3V and GND of the Pico W to holes on the two sides of the board, then hook up pin16 and 10 of the two 74HC595 chips to VCC, pin 13 and pin 8 to GND.

Note: In the Fritzing image above, the side with label is at the bottom.

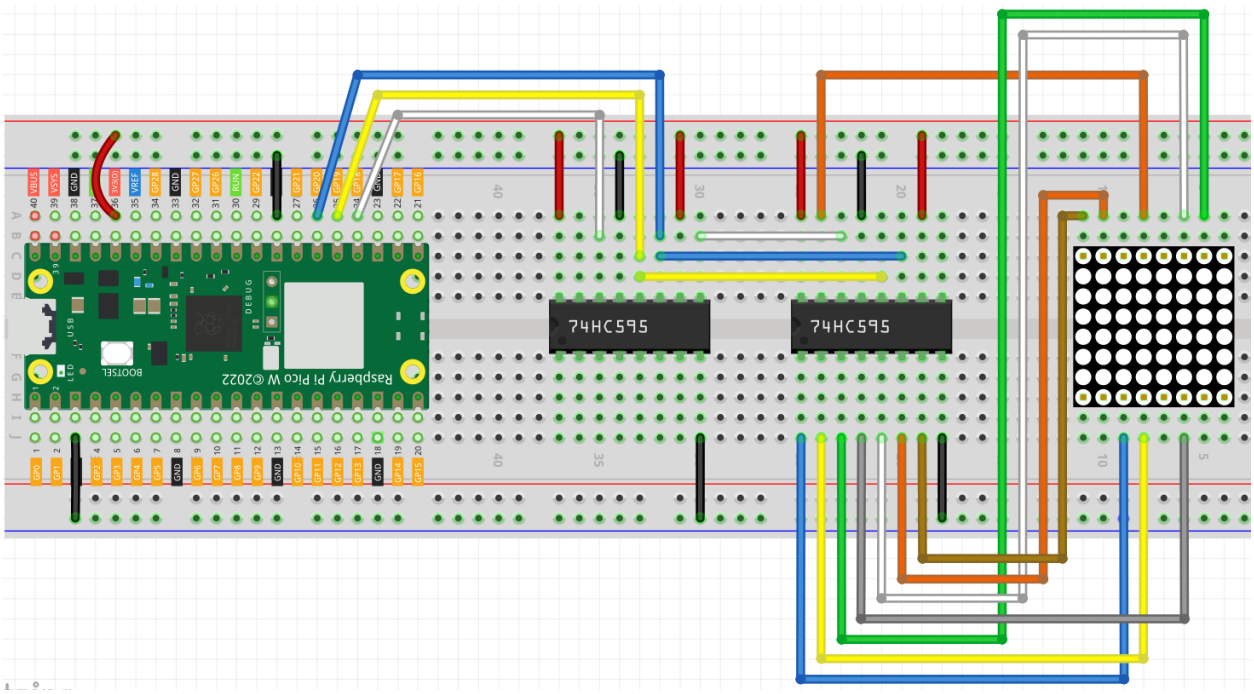


Step 2: Connect pin 11 of the two 74HC595 together, and then to GP20; then pin 12 of the two chips, and to GP19; next, pin 14 of the 74HC595 on the left side to GP18 and pin 9 to pin 14 of the second 74HC595.



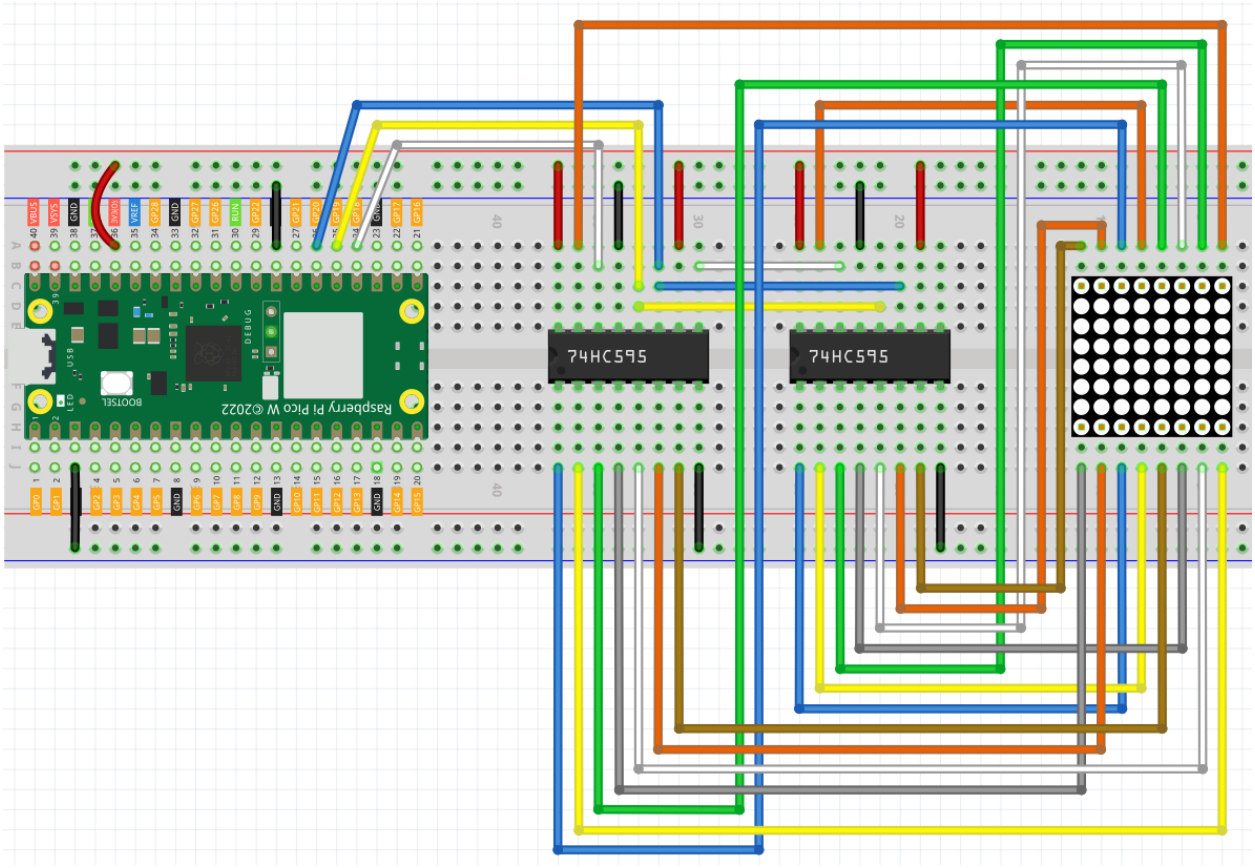
Step 3: The 74HC595 on the right side is to control columns of the LED dot matrix. See the table below for the mapping. Therefore, Q0-Q7 pins of the 74HC595 are mapped with pin 13, 3, 4, 10, 6, 11, 15, and 16 respectively.

74HC595	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
LED Dot Matrix	13	3	4	10	6	11	15	16



Step 4: Now connect the ROWs of the LED dot matrix. The 74HC595 on the left controls ROW of the LED dot matrix. See the table below for the mapping. We can see, Q0-Q7 of the 74HC595 on the left are mapped with pin 9, 14, 8, 12, 1, 7, 2, and 5 respectively.

74HC595	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
LED Dot Matrix	9	14	8	12	1	7	2	5



Code

Note:

- You can open the file `5.4_8x8_pixel_graphics.ino` under the path of `kepler-kit-main/arduino/5.4_8x8_pixel_graphics`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

Once the program is running, you will see a **x** graphic displayed on the 8x8 dot matrix.

How it works?

Here we use two 74HC595s to provide signals for the rows and columns of the dot matrix. The method of supplying signals is the same as `shiftOut()` in the previous chapters, except that here we need to write the 16-bit binary number at a time.

The main loop calls `shiftOut()` twice, writes two 8-bit binary numbers and then outputs them to the bus, so that a pattern can be displayed.

However, since the LEDs in the dot matrix use common poles, controlling multiple rows/multiple columns at the same time will interfere with each other (e.g, if (1,1) and (2,2) are lit at the same time, (1,2) and (2,1) will inevitably be lit together). Therefore, it is necessary to activate one column (or one row) at a time, cycle 8 times, and use the residual image principle to let the human eye merge 8 patterns, so as to let get a pair of patterns containing 8x8 amount of information.

```
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
}
```

In this example, the main function nests a `for` loop, and when `i` is 1, only the first line is activated (the chip in the control line gets the value `0x80`) and the image of the first line is written. When `i` is 2, the second line is activated (the chip of the control line gets the value `0x40`) and the image of the second line is written. And so on, completing 8 outputs.

Incidentally, like the 4-digit 7-segment display, it has to maintain the refresh rate to prevent flickering by the human eye, so the extra `sleep()` in the main loop should be avoided as much as possible.

Learn More

Try replacing `datArray` with the following array and see what images appear!

```
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};
```

Or, you can try drawing your own graphics.

6. Advanced

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.35 6.1 - Measuring Distance

The ultrasonic sensor module works on the principle of sonar and radar systems for determining the distance to an object.

- *Ultrasonic Module*

Required Components

In this project, we need the following components.

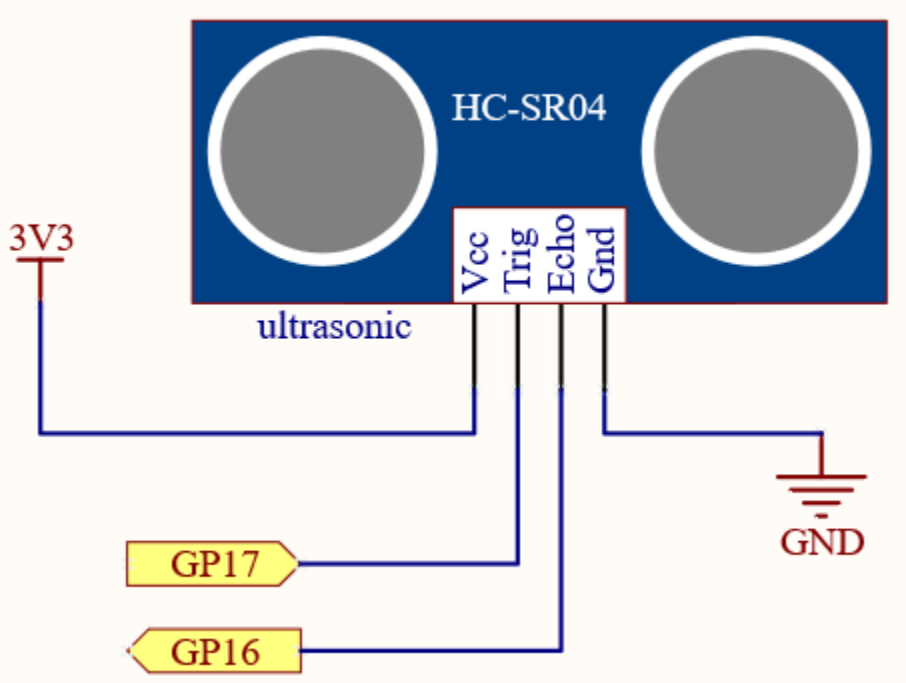
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

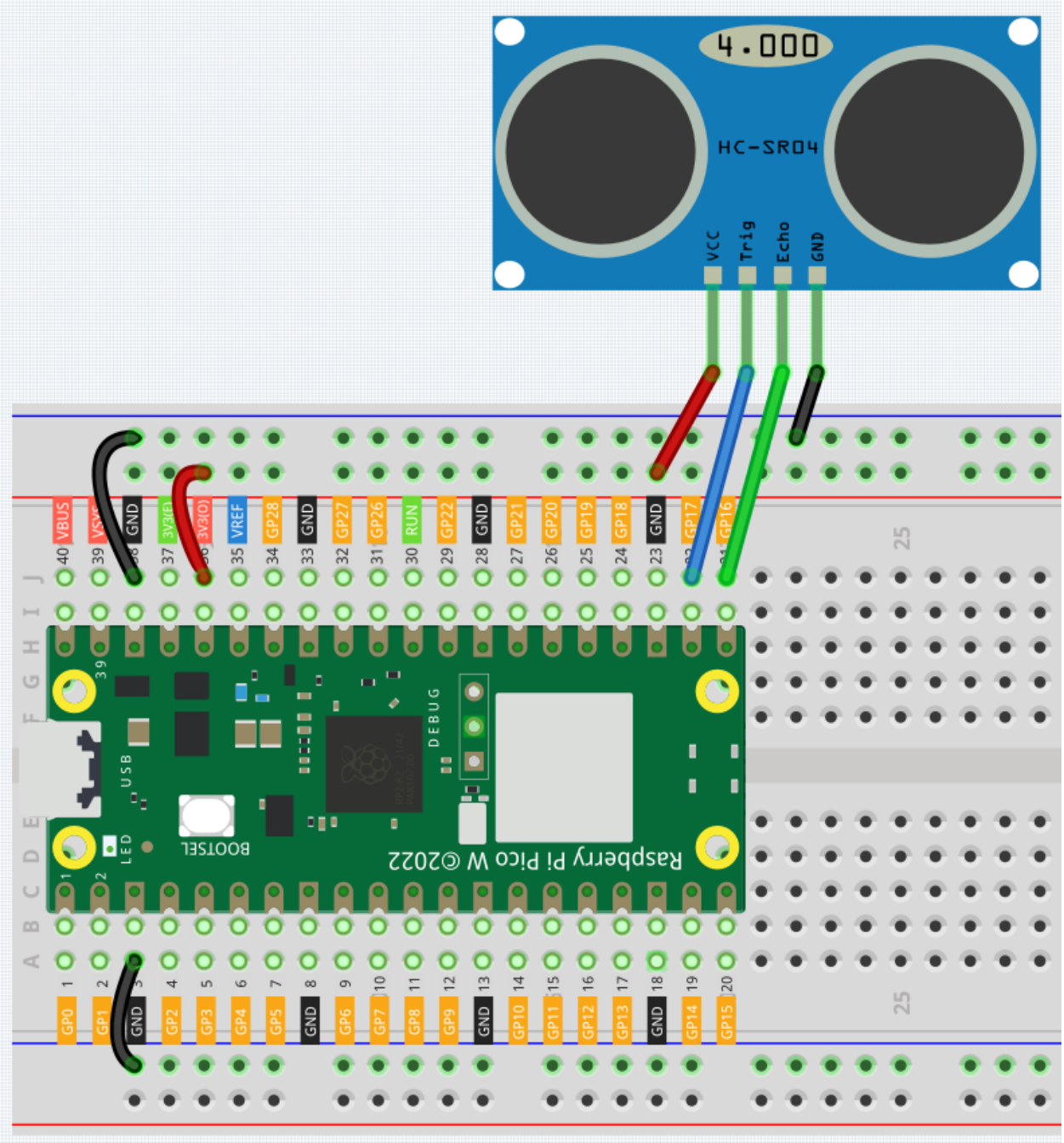
You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Ultrasonic Module*	1	

Schematic



Wiring



Code

Note:

- You can open the file `6.1_ultrasonic.ino` under the path of `kepler-kit-main/arduino/6.1_ultrasonic`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

Once the program is running, the Serial Monitor will print out the distance of the ultrasonic sensor from the obstacle ahead.

How it works?

About the application of ultrasonic sensor, we can directly check the subfunction.

```
float readSensorData(){// ...}
```

PING is triggered by a HIGH pulse of 2 or more microseconds. (Give a short LOW pulse beforehand to ensure a clean HIGH pulse.)

```
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
```

The echo pin is used to read signal from PING, a HIGH pulse whose duration is the time (in microseconds) from the sending of the ping to the reception of echo of the object.

```
microsecond=pulseIn(echoPin, HIGH);
```

The speed of sound is 340 m/s or 29 microseconds per centimeter.

This gives the distance travelled by the ping, outbound and return, so we divide by 2 to get the distance of the obstacle.

```
float distance = microsecond / 29.00 / 2;
```

Note that the ultrasonic sensor will pause the program when it is working, which may cause some lagging when writing complex projects.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

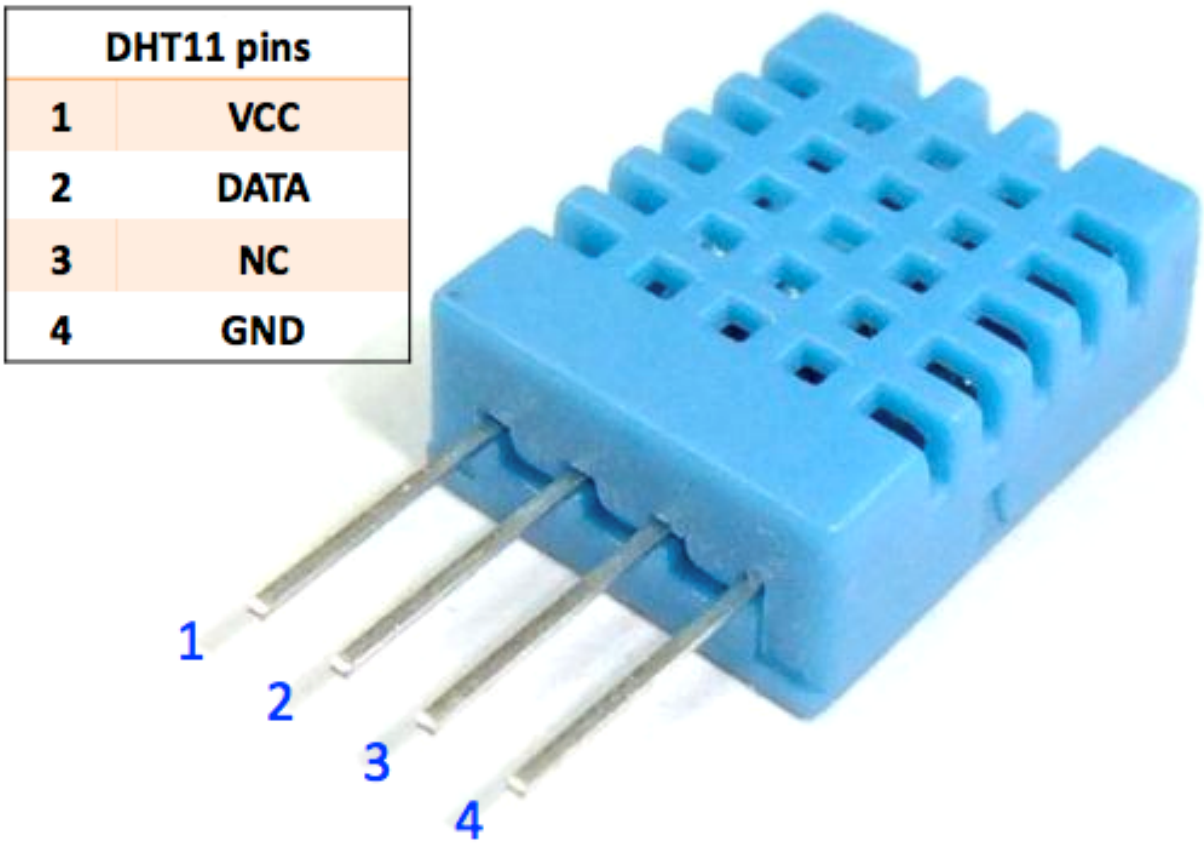
Ready to explore and create with us? Click [] and join today!

6.36 6.2 - Temperature - Humidity

Humidity and temperature are closely related from the physical quantity itself to the actual people's life. The temperature and humidity of human environment will directly affect the thermoregulatory function and heat transfer effect of human body. It will further affect the thinking activity and mental state, thus affecting the efficiency of our study and work.

Temperature is one of the seven basic physical quantities in the International System of Units, which is used to measure the degree of hot and cold of an object. Celsius is one of the more widely used temperature scales in the world, expressed by the symbol "℃".

Humidity is the concentration of water vapor present in the air. The relative humidity of air is commonly used in life and is expressed in %RH. Relative humidity is closely related to temperature. For a certain volume of sealed gas, the higher the temperature, the lower the relative humidity, and the lower the temperature, the higher the relative humidity.



A basic digital temperature and humidity sensor, the **DHT11**, is provided in this kit. It uses a capacitive humidity sensor and thermistor to measure the surrounding air and outputs a digital signal on the data pins (no analog input pins are required).

- *DHT11 Humiture Sensor*

Required Components

In this project, we need the following components.

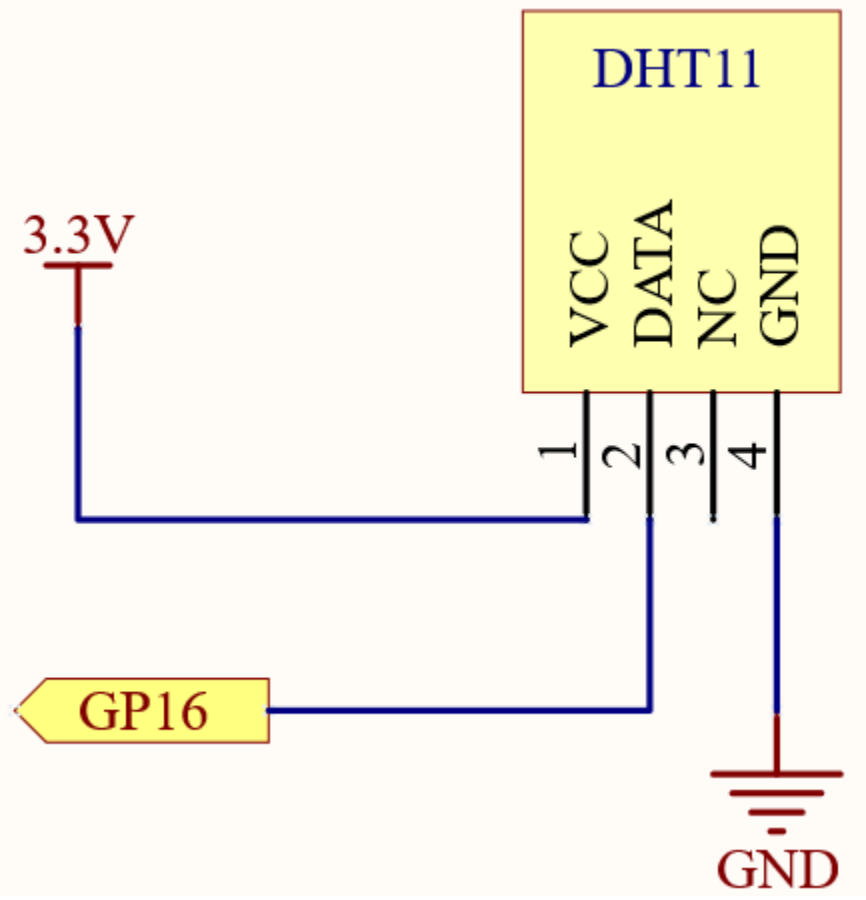
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

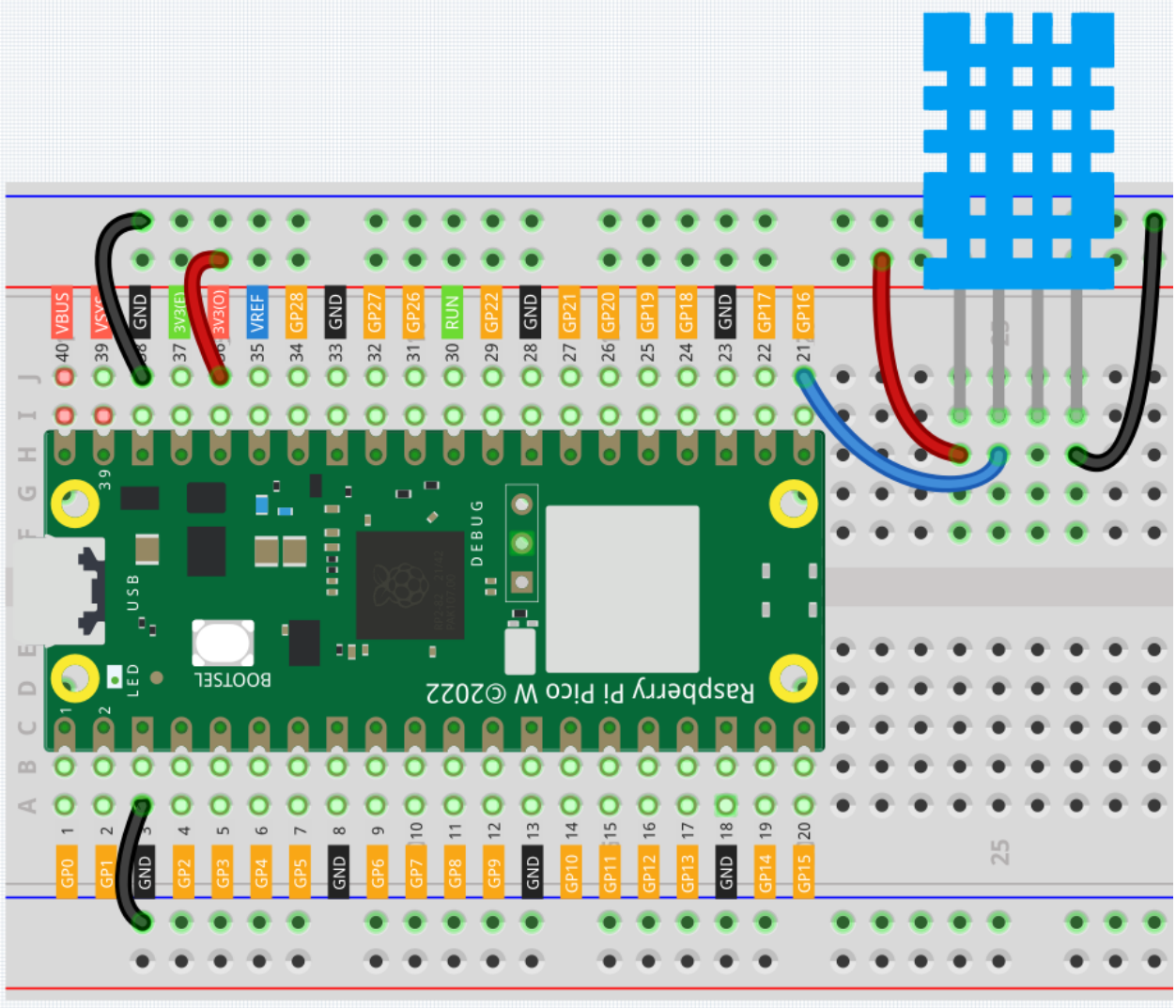
You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*DHT11 Humiture Sensor*	1	

Schematic



Wiring



Code

Note:

- You can open the file `6.2_dht11.ino` under the path of `kepler-kit-main/arduino/6.2_dht11`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.
- The library `SimpleDHT` is used here. Please refer to *Add libraries* for adding it to the Arduino IDE.

After the code is run, you will see the Serial Monitor continuously print out the temperature and humidity, and as the program runs steadily, these two values will become more and more accurate.

How it works?

Initialize the DHT11 object. This device requires only a digital input to be used.

```
int pinDHT11 = 16;
SimpleDHT11 dht11(pinDHT11);
```

Reads the current temperature and humidity, which are stored in the variables `temperature` and `humidity`. `err` is used to determine the validity of the data.

```
byte temperature = 0;
byte humidity = 0;
int err = dht11.read(&temperature, &humidity, NULL);
```

Filter invalid data.

```
if (err != SimpleDHTErrSuccess) {
    Serial.print("Read DHT11 failed, err=");
    Serial.print(SimpleDHTErrCode(err));
    Serial.print(",");
    Serial.println(SimpleDHTErrDuration(err));
    delay(1000);
    return;
}
```

Print temperature and humidity.

```
Serial.print((int)temperature);
Serial.print(" *C, ");
Serial.print((int)humidity);
Serial.println(" H");
```

Finally, the DHT11 sampling rate is 1HZ, a `delay(1500)` is needed in the loop.

```
delay(1500);
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.37 6.3 - 6-axis Motion Tracking

The MPU-6050 is a 6-axis(combines 3-axis Gyroscope, 3-axis Accelerometer) motion tracking devices.

An accelerometer is a tool that measures proper acceleration.For example, an accelerometer at rest on the surface of the Earth will measure an acceleration due to Earth's gravity, straight upwards[3] (by definition) of g 9.81 m/s2.

Accelerometers have many uses in industry and science. For example: inertial navigation systems for aircraft and missiles, for keeping images on tablets and digital cameras vertical, etc.

Gyroscopes are used to measure orientation and angular velocity of a device or maintenance. Applications of gyroscopes include anti-rollover and airbag systems for automobiles, motion sensing systems for smart devices, attitude stabilization systems for drones, and more.

- *MPU6050 Module*

Required Components

In this project, we need the following components.

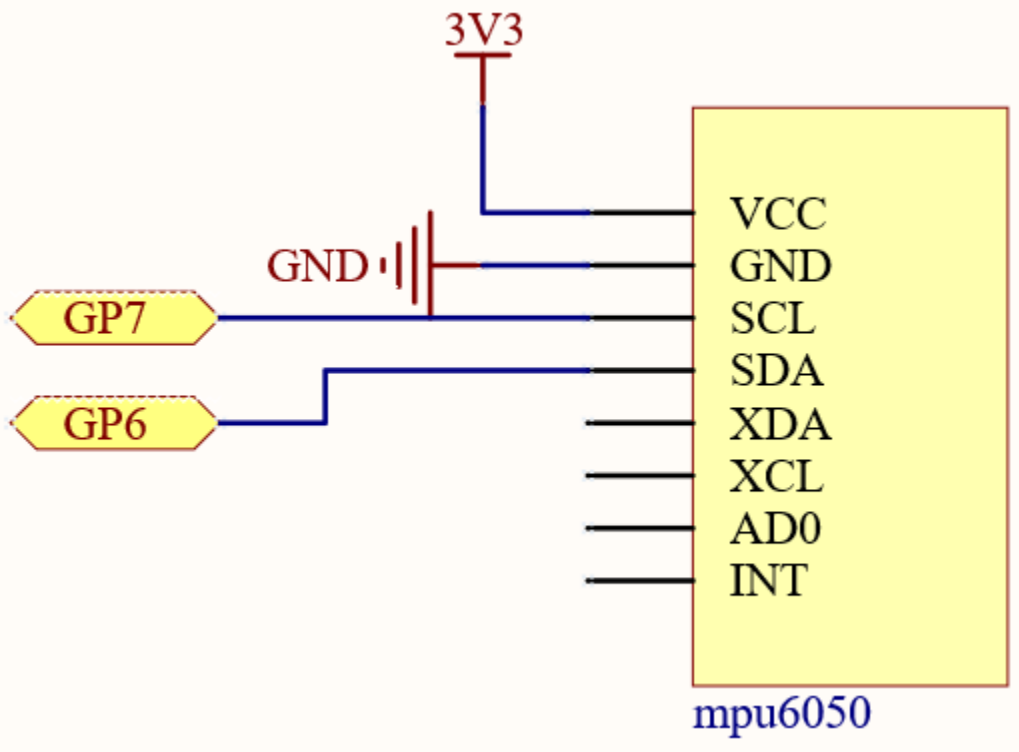
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

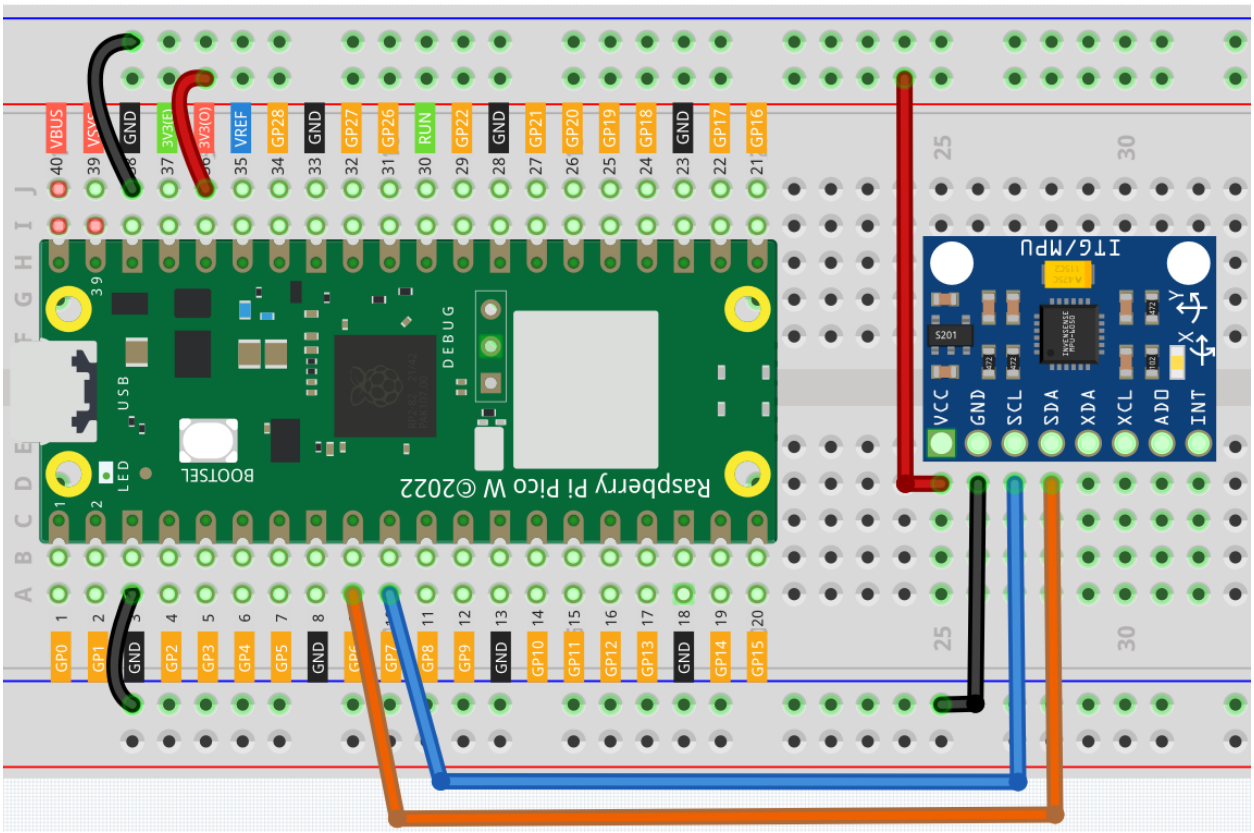
You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*MPU6050 Module*	1	

Schematic


3V3
GND
GP7
GP6
VCC
GND
SCL
SDA
XDA
XCL
AD0
INT
mpu6050

Wiring


VCC
GND
SCL
SDA
XDA
XCL
AD0
INT
BOOTSEL
Raspberry Pi Pico W ©2022

Code

Note:

- You can open the file `6.3_6axis_motion_tracking.ino` under the path of `kepler-kit-main/arduino/6.3_6axis_motion_tracking`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.
- The library `Adafruit_MPU6050` is used here. Please refer to *Add libraries* for adding it to the Arduino IDE.

After running the program, you can see the 3-axis accelerometer values and 3-axis gyroscope values cycling through the output. At this point you rotate the MPU6050 at random, and these values will appear to change accordingly. To make it easier to see the changes, you can comment out one of the print lines and concentrate on another set of data.

How it works?

Instantiate an `MPU6050` object.

```
#include <Adafruit_MPU6050.h>
#include <Wire.h>

Adafruit_MPU6050 mpu;
```

Initialize the MPU6050 and set its accuracy.

```
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);
        }
    }
    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);
}
```

Get new sensor events with the readings.

```
sensors_event_t a, g, temp;
mpu.getEvent(&a, &g, &temp);
```

Subsequently, you will be able to get real-time acceleration and angular velocity values in the data `a.acceleration.x`, `a.acceleration.y`, `a.acceleration.z`, `g.gyro.x`, `g.gyro.y`, `g.gyro.z`.

```
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");
```

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.38 6.4 - IR Remote Control

In consumer electronics, remote controls are used to operate devices such as televisions and DVD players. In some cases, remote controls allow people to operate devices that are out of their reach, such as central air conditioners.

IR Receiver is a component with photocell that is tuned to receive to infrared light. It is almost always used for remote control detection - every TV and DVD player has one of these in the front to receive for the IR signal from the clicker. Inside the remote control is a matching IR LED, which emits IR pulses to tell the TV to turn on, off or change channels.

- *Infrared Receiver*

Required Components

In this project, we need the following components.

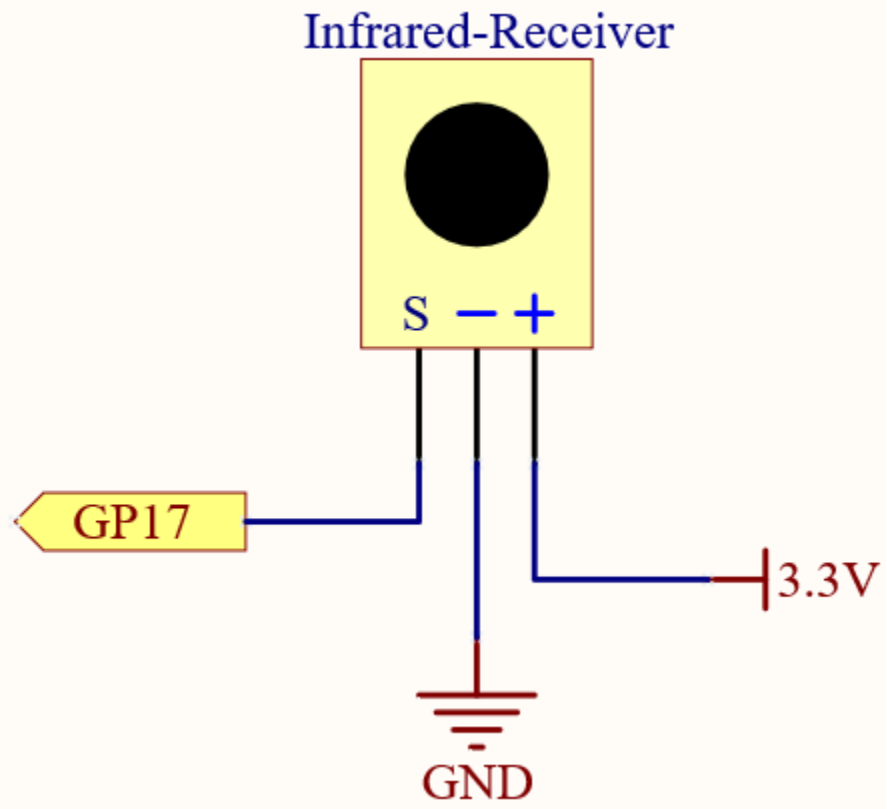
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

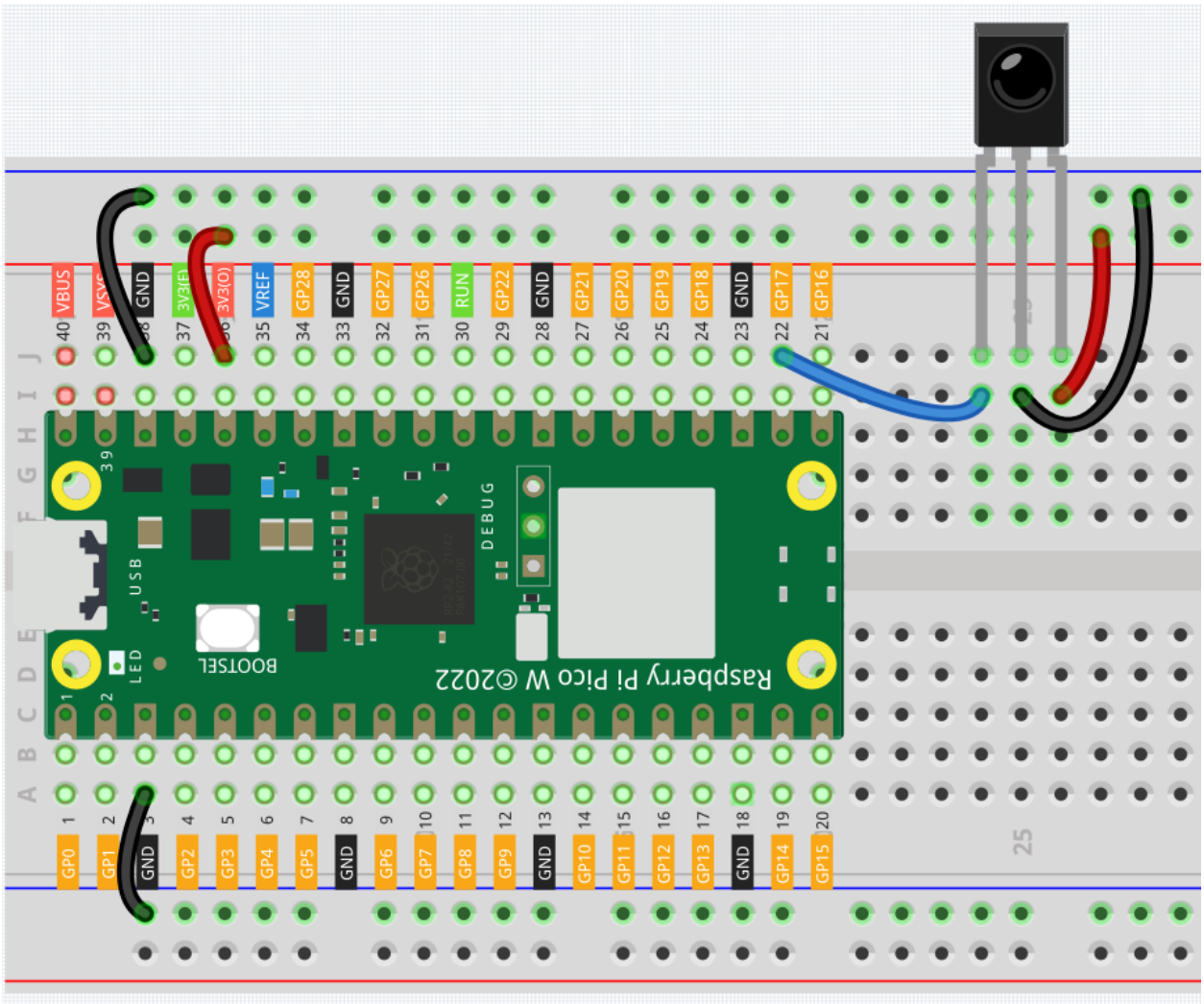
You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUAN-TITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Infrared Receiver*	1	

Schematic



Wiring



Code

Note:

- You can open the file `6.4_ir_remote_control.ino` under the path of `kepler-kit-main/arduino/6.4_ir_remote_control`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.
- The library `IRsmallDecoder` is used here. Please refer to *Add libraries* for adding it to the Arduino IDE.

The new remote control has a plastic piece at the end to isolate the battery inside. You need to pull out this plastic piece to power up the remote when you are using it. Once the program is running, when you press the remote control, the Serial Monitor will print out the key you pressed.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

6.39 6.5 - Radio Frequency Identification

Radio Frequency Identification (RFID) refers to technologies that involve using wireless communication between an object (or tag) and an interrogating device (or reader) to automatically track and identify such objects. The tag transmission range is limited to several meters from the reader. A clear line of sight between the reader and tag is not necessarily required.

Most tags contain at least one integrated circuit (IC) and an antenna. The microchip stores information and is responsible for managing the radio frequency (RF) communication with the reader. Passive tags do not have an independent energy source and depend on an external electromagnetic signal, provided by the reader, to power their operations. Active tags contain an independent energy source, such as a battery. Thus, they may have increased processing, transmission capabilities and range.

- *MFRC522 Module*

Required Components

In this project, we need the following components.

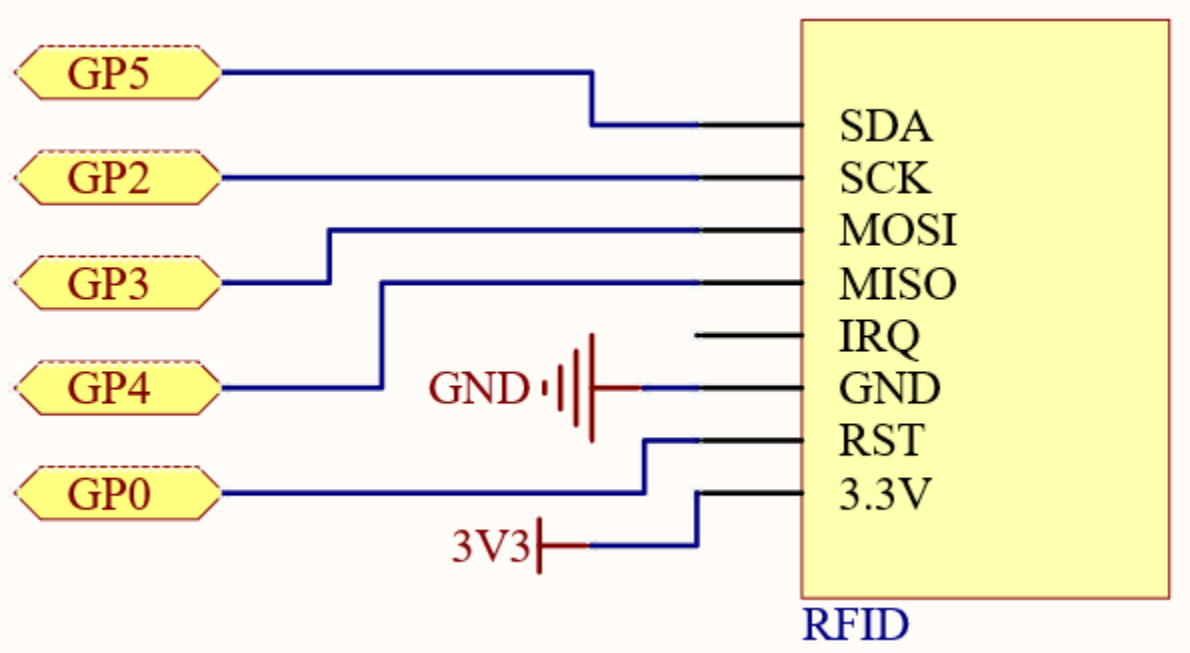
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	PURCHASE LINK
Kepler Kit	450+	

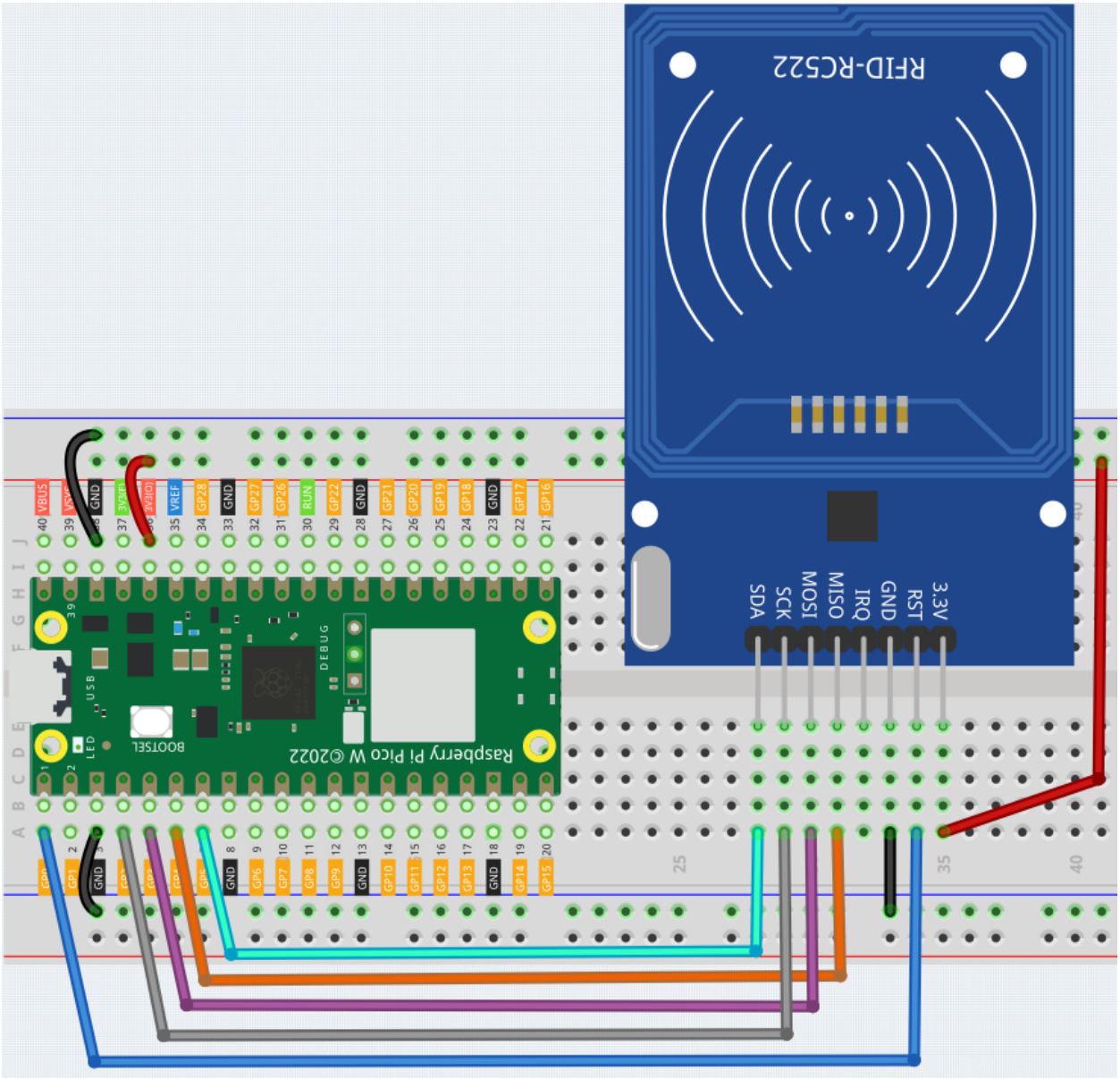
You can also buy them separately from the links below.

SN	COMPONENT INTRODUCTION	QUANTITY	PURCHASE LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*MFRC522 Module*	1	

Schematic


GP5
GP2
GP3
GP4
GP0
GND
3V3
SDA
SCK
MOSI
MISO
IRQ
GND
RST
3.3V
RFID

Wiring



Code

Note:

- You can open the file `6.5_rfid_write.ino` under the path of `kepler-kit-main/arduino/6.5_rfid_write`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.
- The library `MFRC522` is used here. Please refer to *Add libraries* for adding it to the Arduino IDE.

The main function is divided into two:

- `6.5_rfid_write.ino`: Used to write information to the card (or key).

- `6.5_rfid_read.ino`: used to read the information in the card (or key)

Note:

- You can open the file `6.5_rfid_write.ino` under the path of `kepler-kit-main/arduino/6.5_rfid_write`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

After running you will be able to enter message in the serial monitor, ending with `#`, and then write the message to the card by placing the card (or key) close to the MFRC522 module.

Note:

- You can open the file `6.5_rfid_read.ino` under the path of `kepler-kit-main/arduino/6.5_rfid_read`.
- Or copy this code into **Arduino IDE**.
- Don't forget to select the board(Raspberry Pi Pico) and the correct port before clicking the **Upload** button.

After running, you will be able to read the message stored in the card (or key).

How it works?

```
#include <MFRC522.h>

#define RST_PIN         0
#define SS_PIN          5

MFRC522 mfrc522(SS_PIN, RST_PIN);
```

First, instantiate `MFRC522()` class.

For simplicity of use, the `MFRC522` library is further encapsulated with the following functions.

- `void simple_mfrc522_init()` : Starts SPI communication and initializes the mfrc522 module.
- `void simple_mfrc522_get_card()` : Suspends the program until the card (or key) is detected, prints the card UID and PICC type.
- `void simple_mfrc522_write(String text)` : Write a string for the card (or key).
- `void simple_mfrc522_write(byte* buffer)` : Writes information for the card (or key), which usually comes from the serial port.
- `void simple_mfrc522_write(byte section, String text)` : Writes a string for a specific sector. `section` is set to 0 to write sectors 1-2; `section` is set to 1 to write sectors 3-4.
- `void simple_mfrc522_write(byte section, byte* buffer)` : Writes information for a specific sector, usually from the serial port. `section` set to 0, writes 1-2 sectors; `section` set to 1, writes 3-4 sectors.
- `String simple_mfrc522_read()` : Reads the information in the card (or key), returns a string.
- `String simple_mfrc522_read(byte section)` : Reads the information in a specific sector, returns a string. `section` is set to 0, writes 1-2 sectors; `section` is set to 1, writes 3-4 sectors.

In the `6.5_rfid_write.ino` example, the `Serial.readBytesUntil()` function is used, which is a common serial input method.

- Serial.readBytesUntil

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

CHAPTER

SEVEN

FOR PIPER MAKE

This chapter contains an introduction to Piper Make, how to connect and program Pico W win Piper Make, and several interesting projects to help you get up and running with Piper quickly.

We recommend that you read this chapter in order.

Piper Make is a super easy and fun way to make projects using Raspberry Pi Pico W. It uses blocks like Scratch, so you don't need any programming experience to use it. The underlying principle is to use CircuitPython with auxiliary libraries.

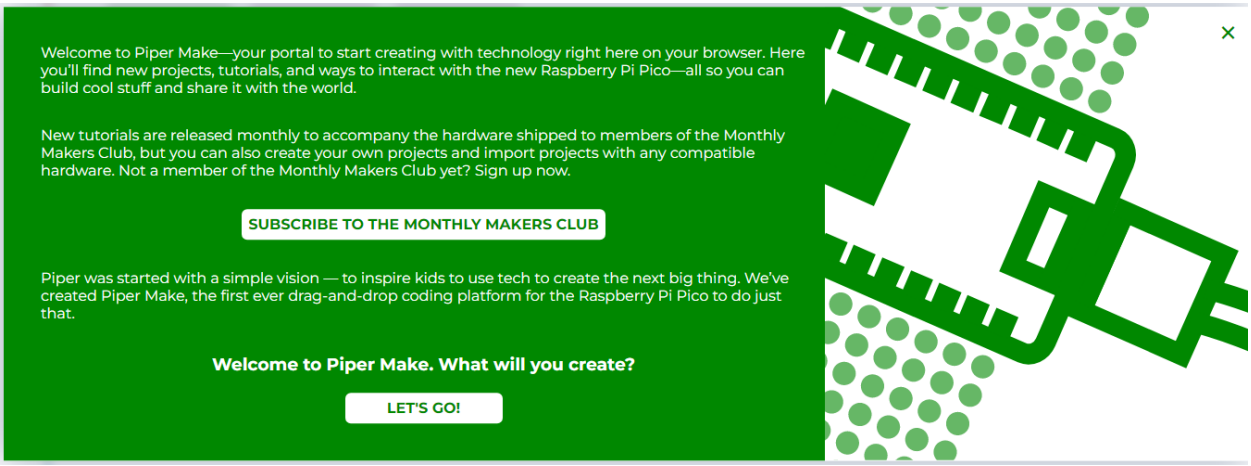
1. Get Started

7.1 1.1 Set up the Pico

First, visit Piper Make through the following link:

https://make.playpiper.com/

In the pop-up page, if you don't need to subscribe for more tutorials, you can just click **Let's Go!** or the **x** button.

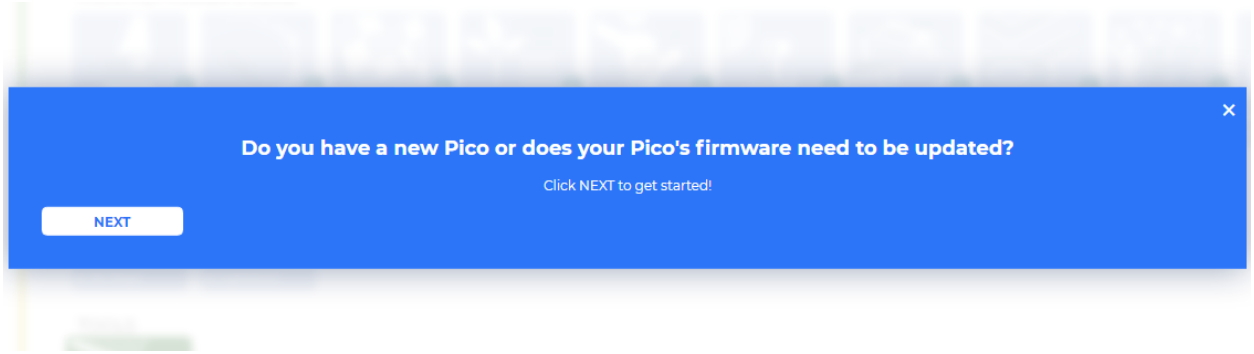


Note: If you see a different pop-up window, your browser version is not supported, please update your browser and try again.

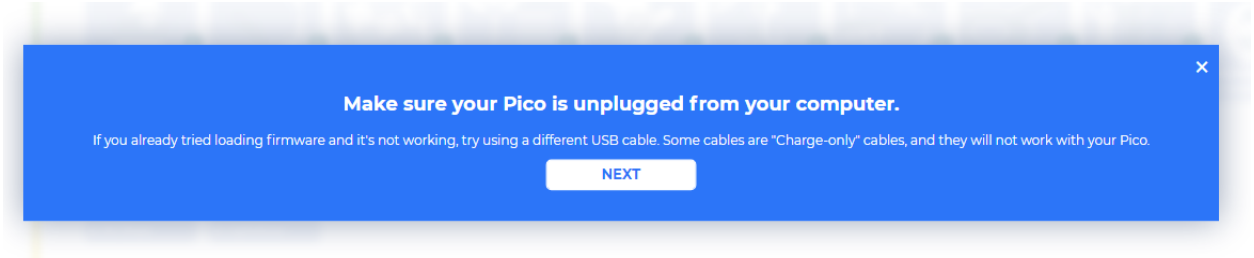
Find the `SETUP MY PICO` button, click it, and follow the prompts to configure it.



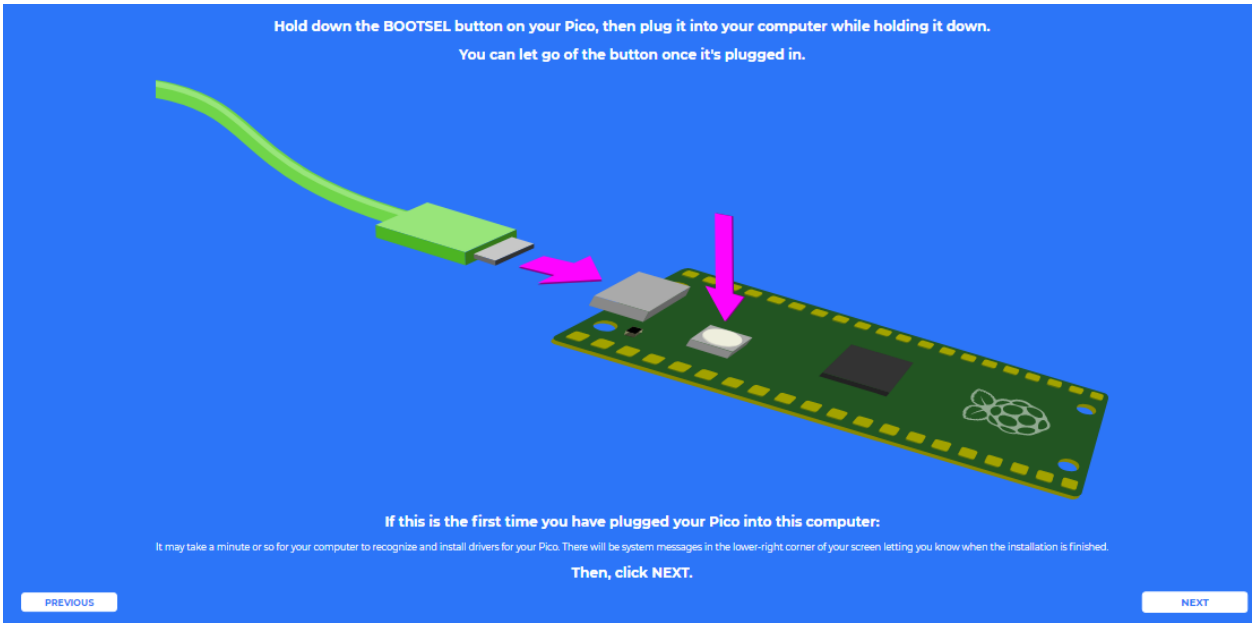
Click **Next** to start configuring your Pico W, even if you have set it up before, these are the same steps you will use to update your Pico W firmware.



In this step, you need to make sure that your Pico W is unplugged from your computer, as it needs to be plugged in in a specific way in the next step. Make sure your cable can handle power and data, as many micro USB cables only have power.

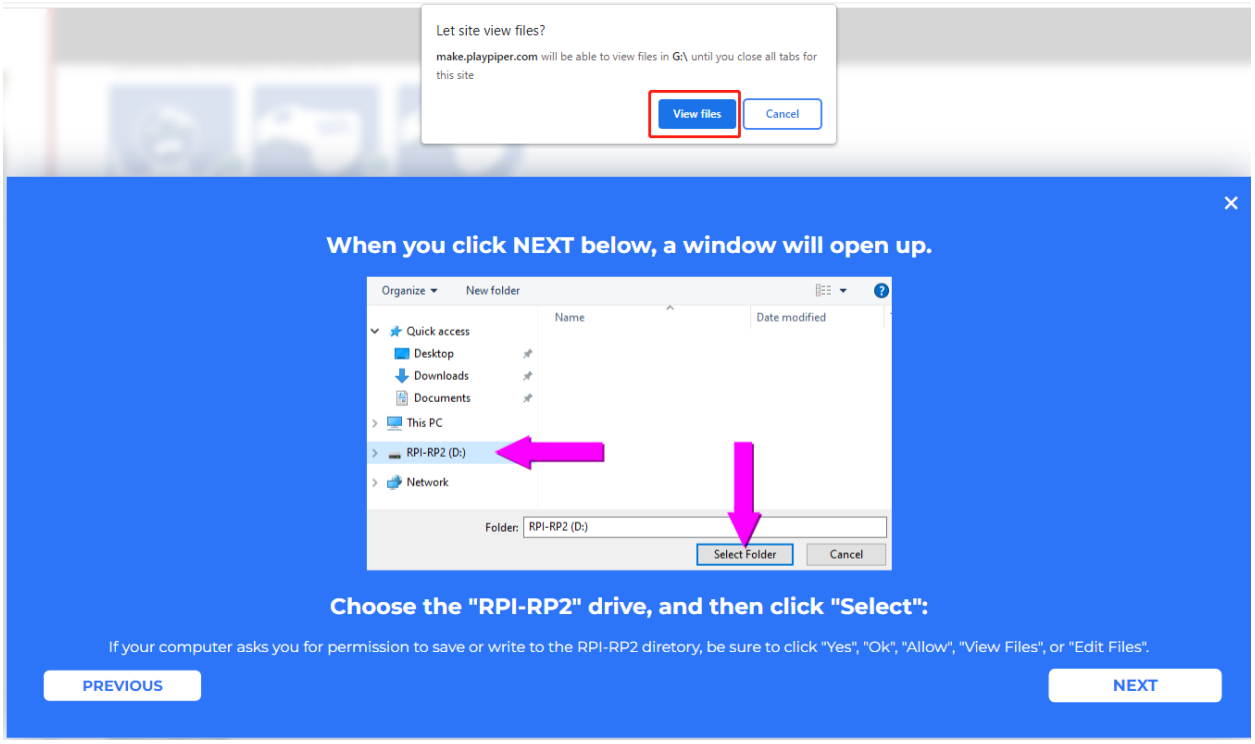


Now, press and hold the RST (white) button on the Pico W and plug the Pico W into your computer. Once plugged in, you can release the button.

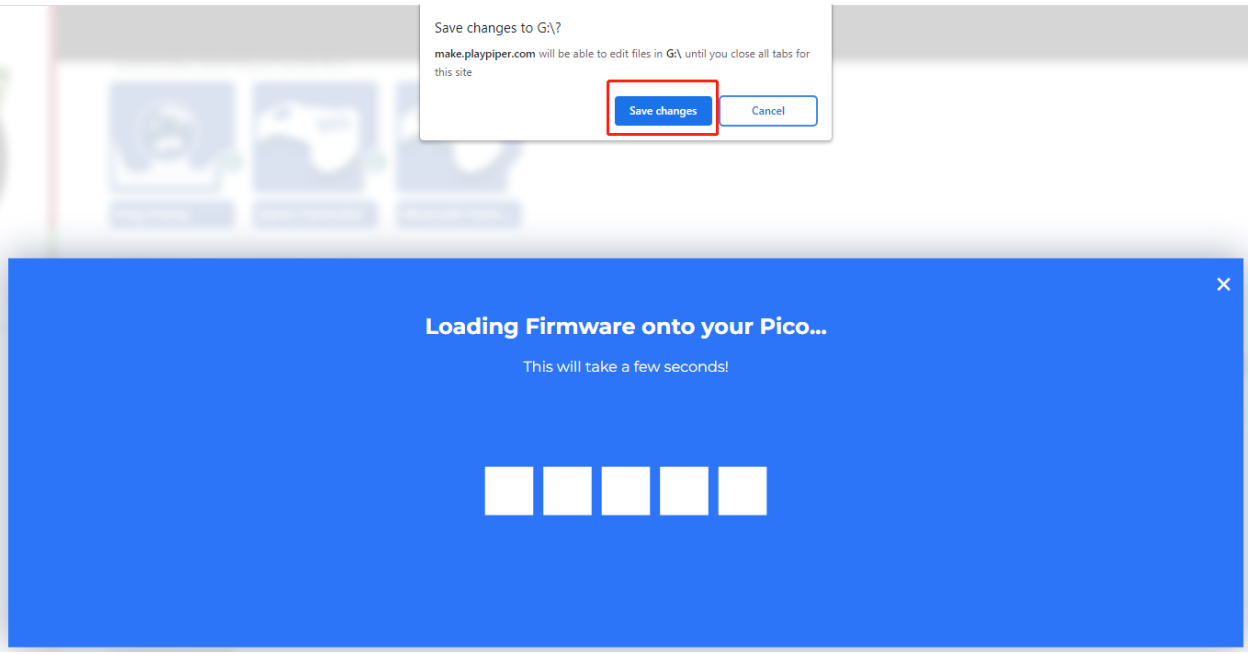


Your Pico W will appear as a USB drive, click **Next** after that select **RPI-RP2** drive.

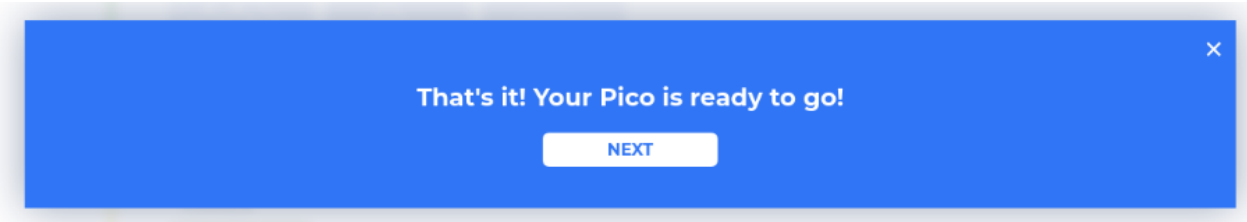
Note: After select **RPI-RP2** drive, there will be a pop up window at the top that you need to allow the web page to view files.



Now Piper Make will load the firmware to your Pico W, again you need to allow save changes to the hard drive where the Pico W is located.



When this prompt appears, it means your Pico W is set up and you can start using it.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

7.2 1.2 Quick Guide on Piper Make

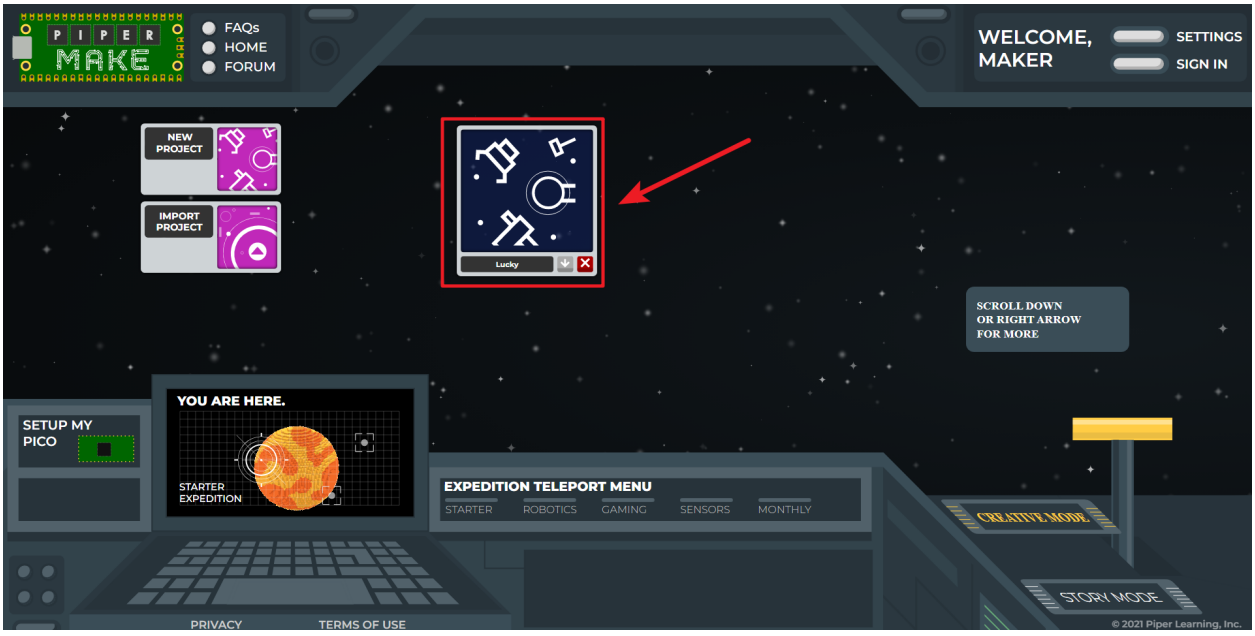
7.2.1 1. Create New Project

Now that you have set up Pico W, it is time to learn how to program it. Now let's light up the onboard LED.

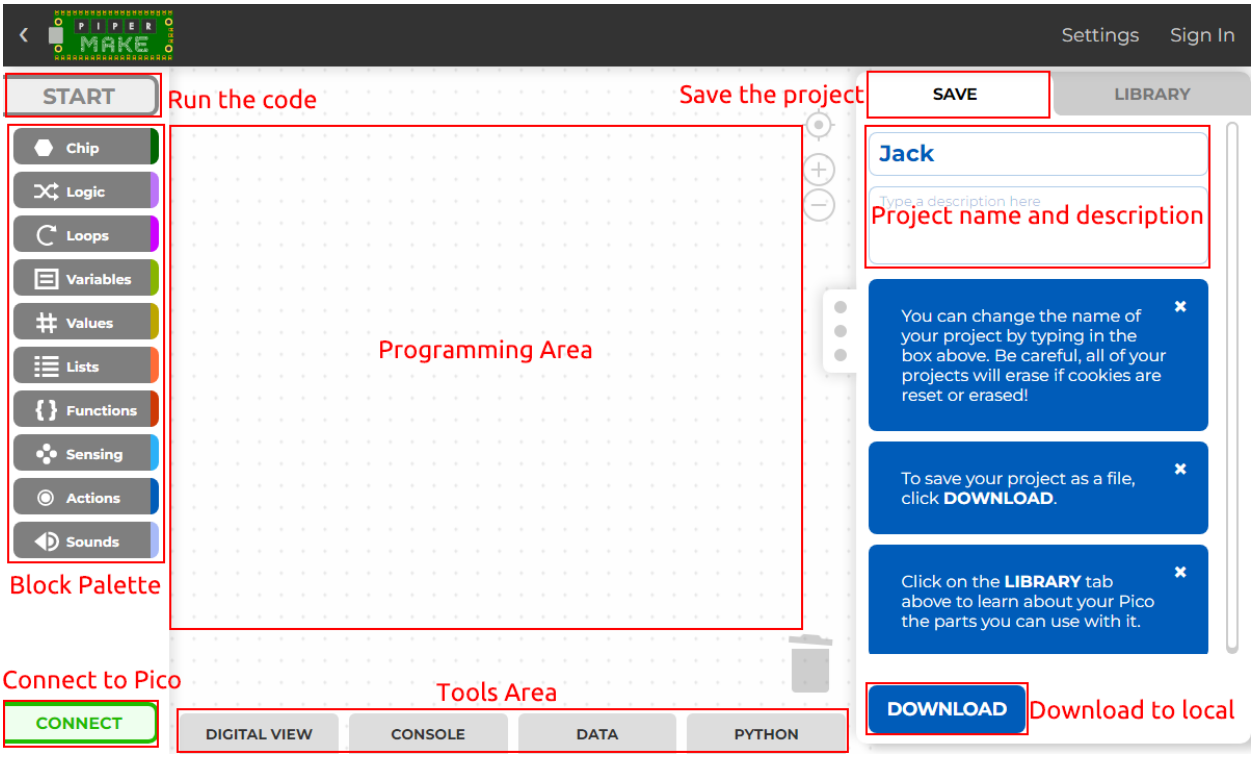
Switch to `CREATIVE MODE` and click on the **New Project** button, and a new project will appear in the **MY PROJECTS** section and will be assigned a random name that can be changed from the programming page.



Then open the new project just created.

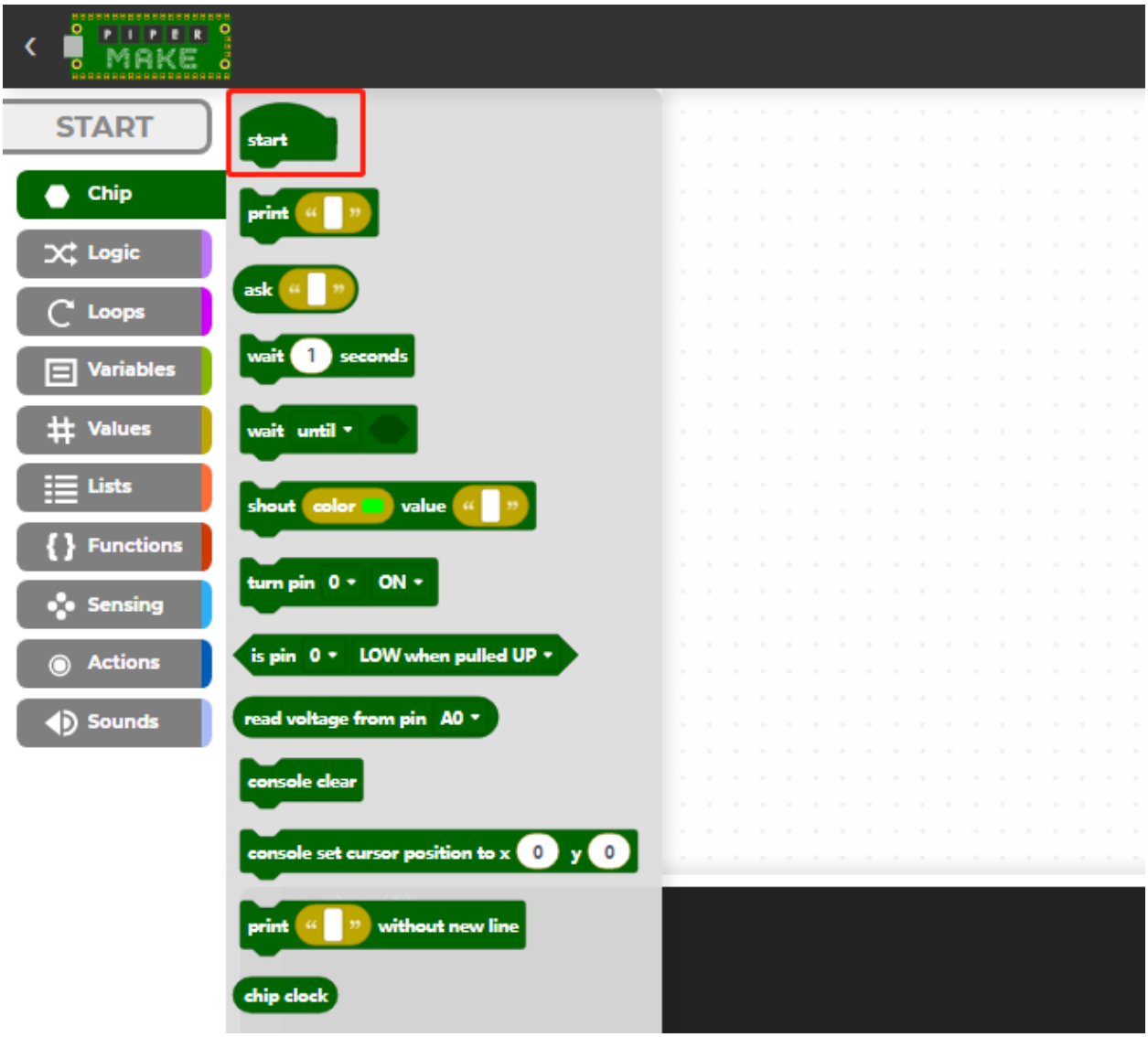


Now go to the Piper Make programming page.

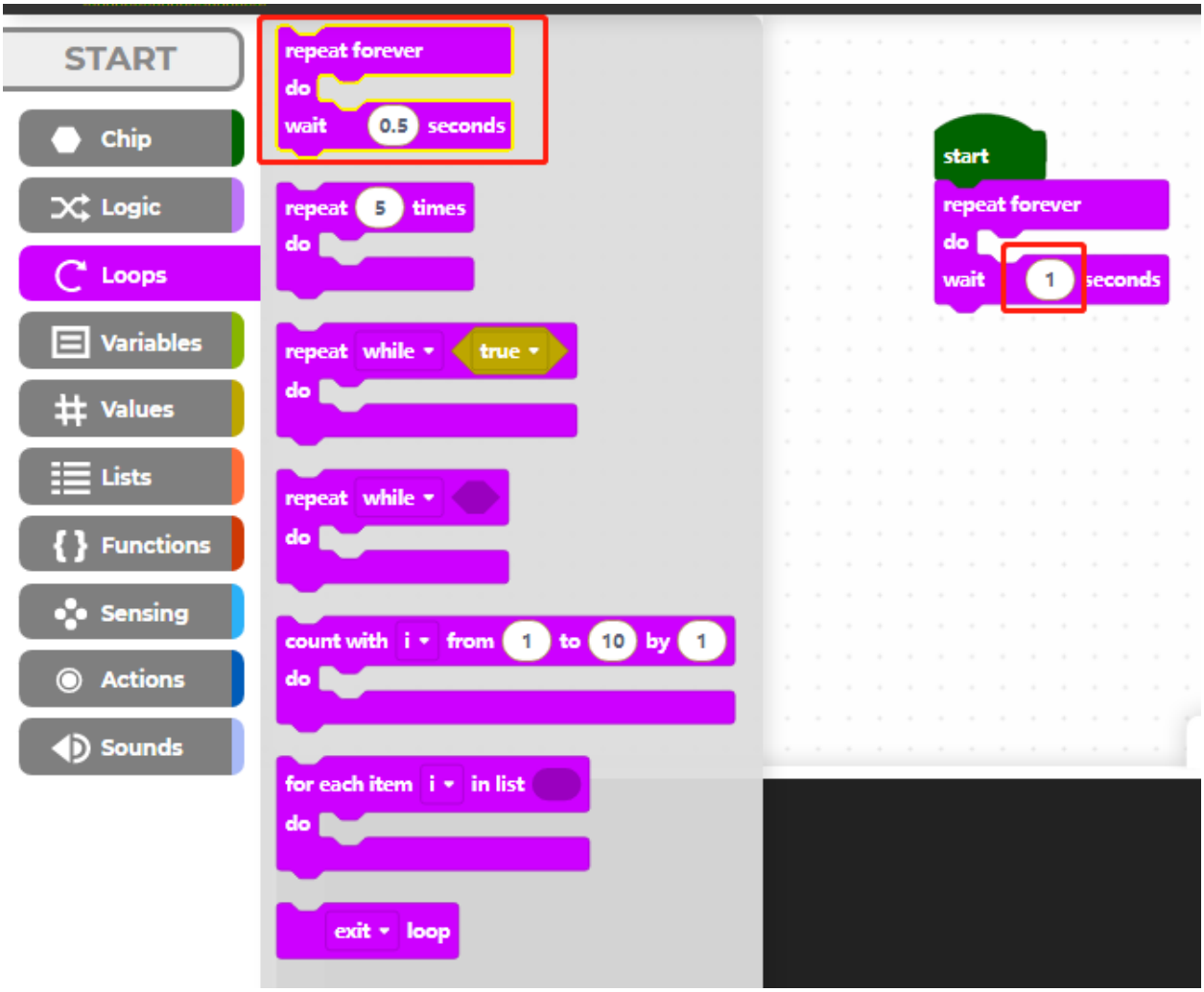


- **START**: Used to run the code, if it's gray, it's not connected to Pico W at this time.
- **Block palette**: contains different types of blocks.
- **CONNECT**: Used to connect to Pico W, it is green when not connected to Pico W, when connected it will become **DISCONNECT(red)**.
- **Programming Area**: Drag blocks here to finish programming by stacking them.
- **Tools Area**: You can click **DIGITAL VIEW** to see the pin distribution of Pico W; you can view the print information in **CONSOLE**; you can read data from **DATA**, and you can click **Python** to view the Python source code.
- **Project name and description**: You can change the project name and description.
- **DOWNLOAD**: You can click the **DOWNLOAD** button to save it locally, usually in **|** format. Next time you can import it via the **Import Project** button on the home page.

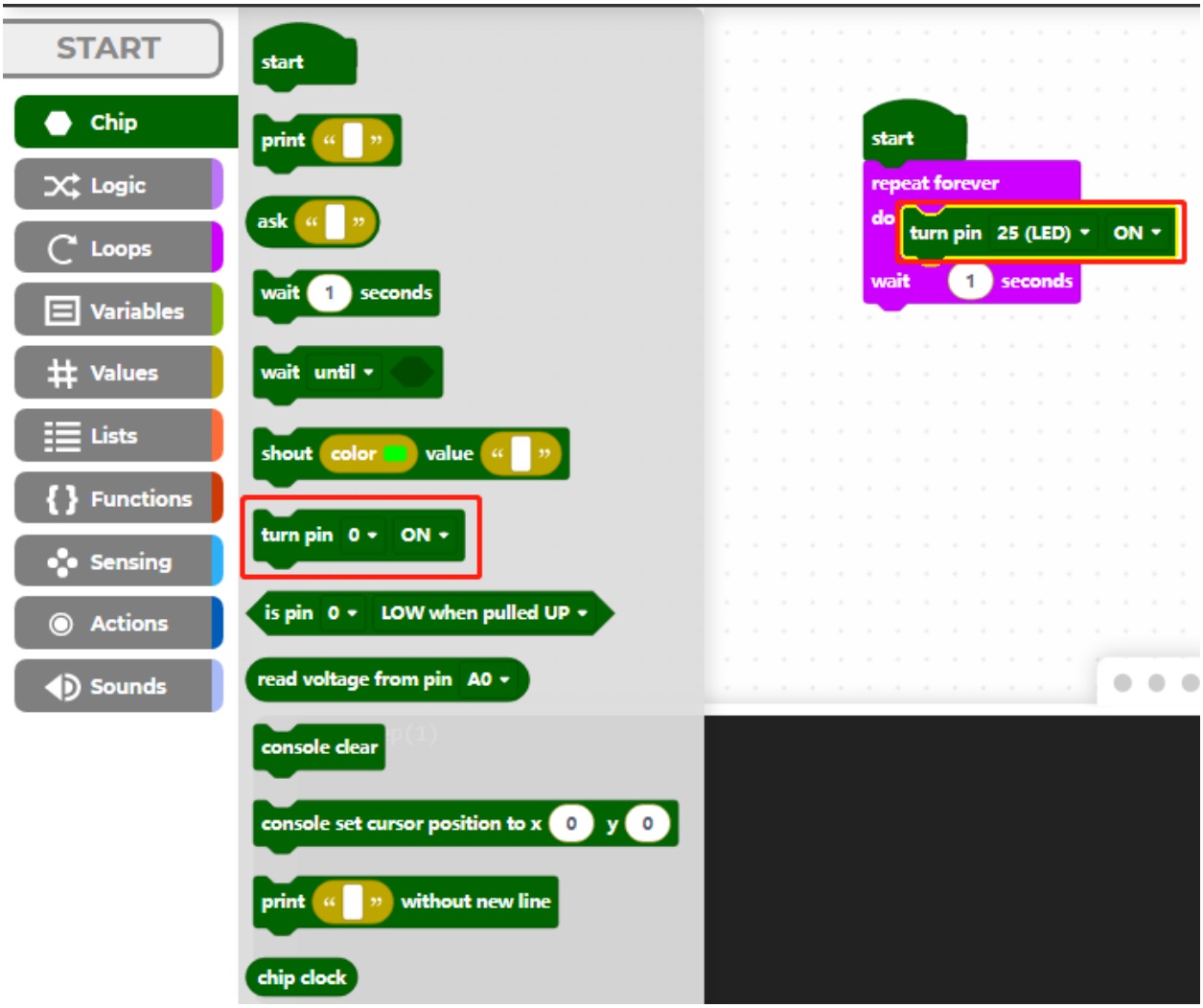
Click on the **Chip** palette and drag the [start] block to the **Programming Area**.



Then drag the [loop] block in **loops** palette to the bottom of the [start] block, and set the loop interval to 1 second.

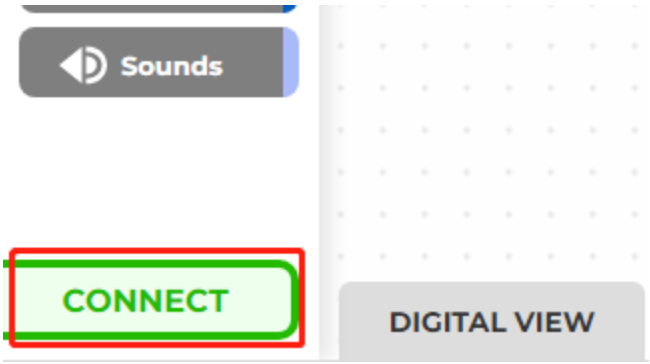


The Raspberry Pi Pico's onboard LED is at pin25, so we use the [turn pin () ON/OFF] block on the **Chip** palette to control it.

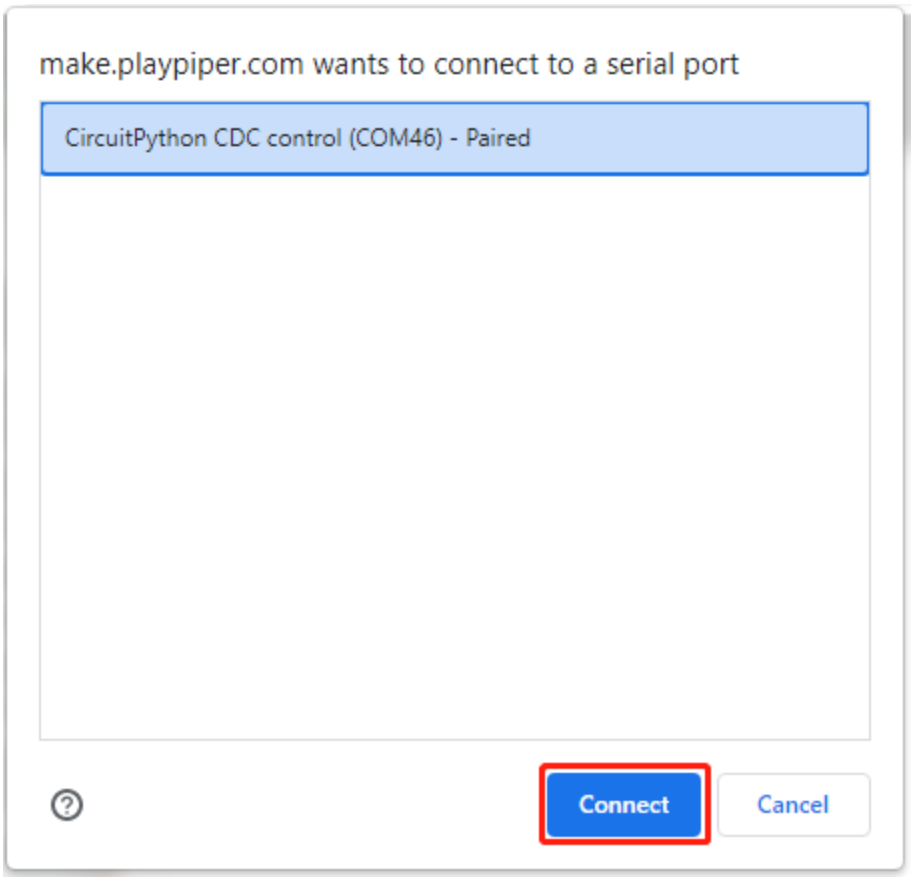


7.2.2 2. Connect to Pico W

Now click on the **CONNECT** button to connect to Pico W, after clicking on it a new popup will appear.



Select the recognized **CircuitPython CDC control (COMXX)** port, then click on **Connect**.

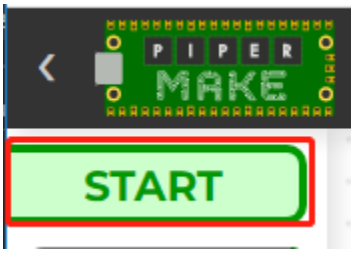


When the connection is successful, the green **CONNECT** in the bottom left corner will change to a red **DISCONNECT**.

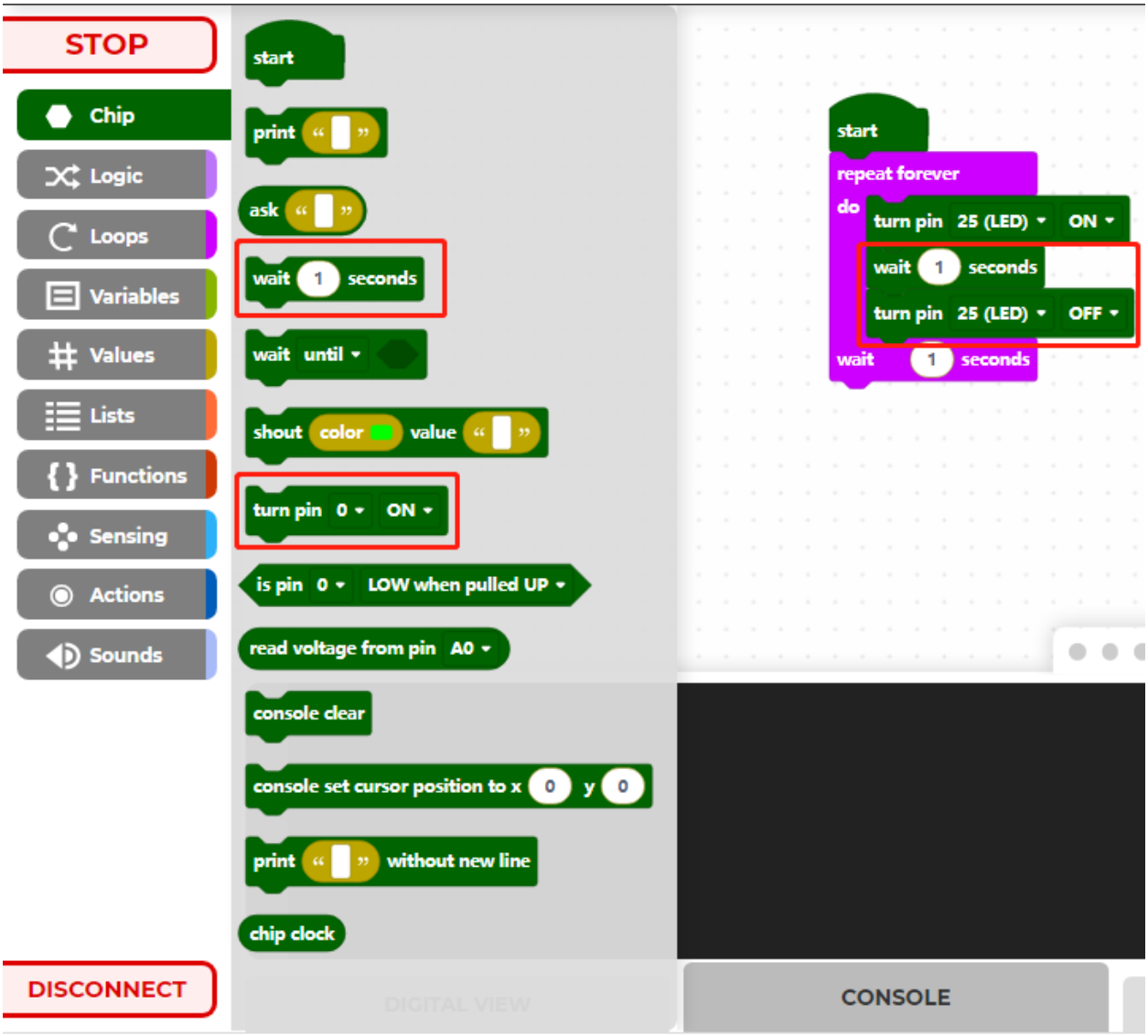


7.2.3 3. Run the Code

Now click on the **START** button to run this code and you will see the LED on the Pico W lit up. If yours is gray, it means that the Pico W is not connected, please reconnect it.



Then turn off pin25 every second in the cycle, and click **START** on the upper left again, so that you can see the onboard LED lights flashing.



Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook!

Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

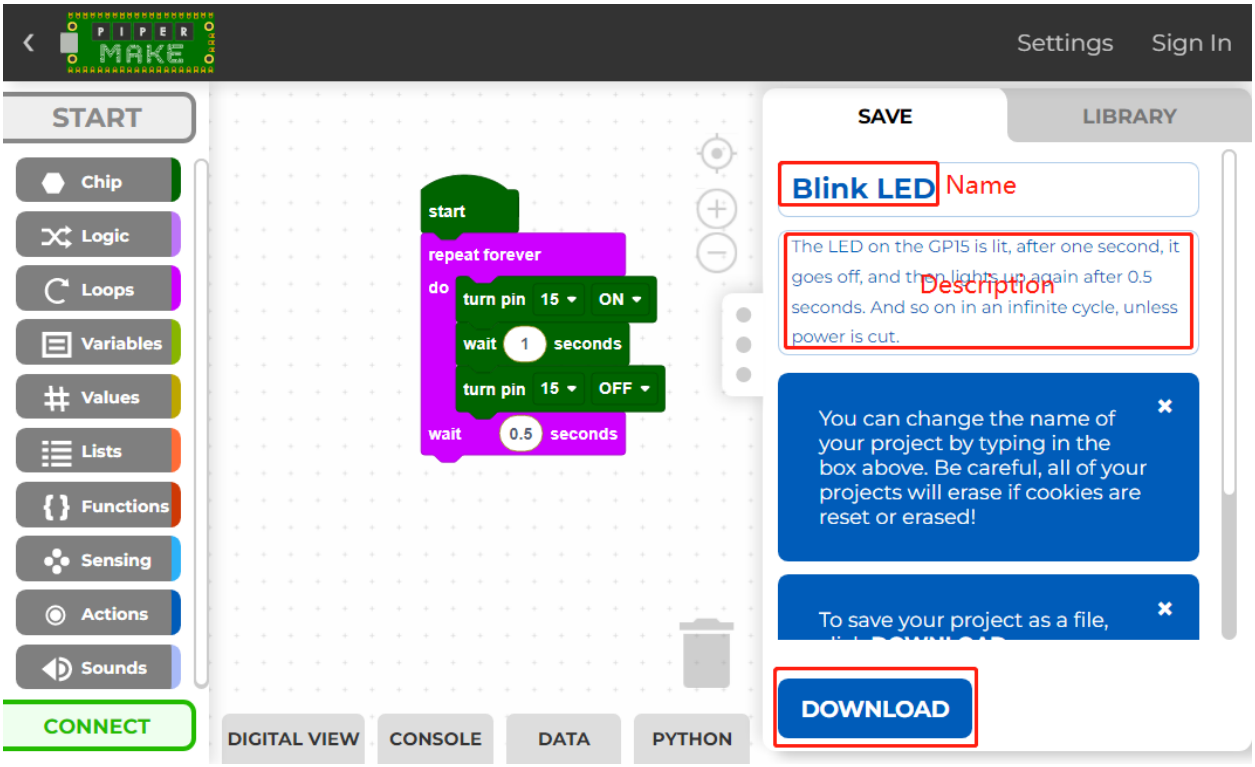
- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

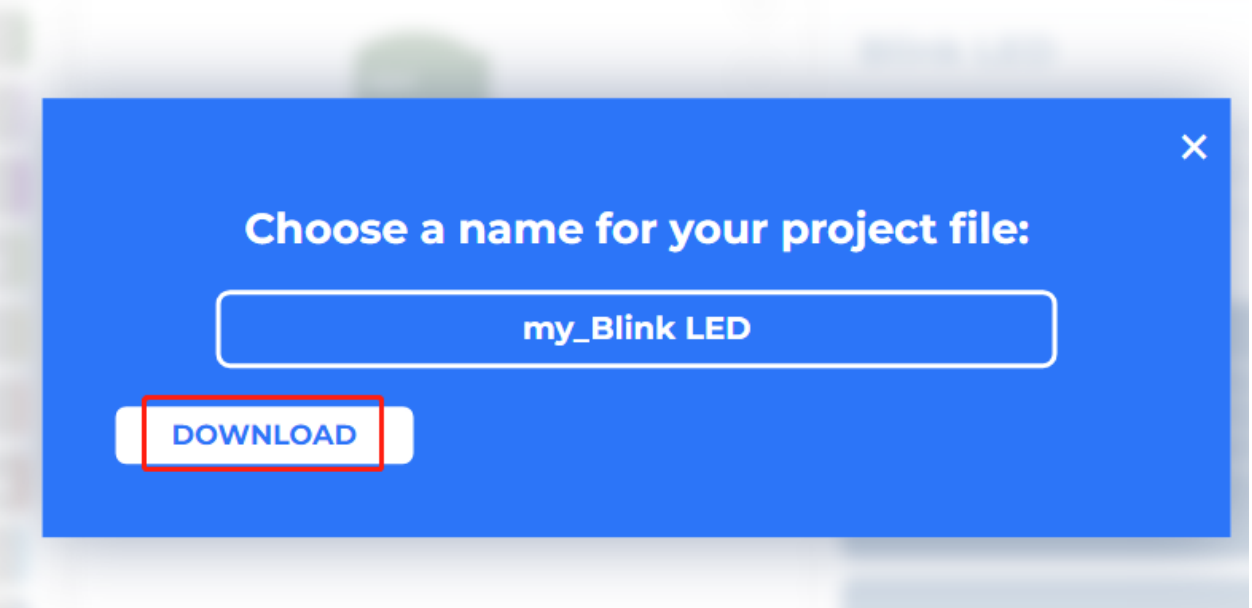
7.3 1.3 How to Save or Import Code?

7.3.1 Save the Code

After writing the code, you can change the code name and description, and then click the **Download** button to save the code locally or share it with others.

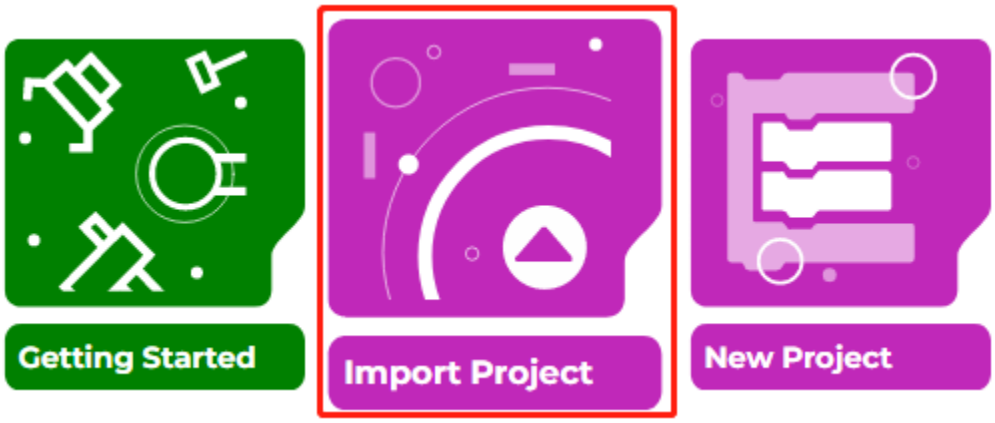


Then enter the file name and click the **Download** button again to save the code as a `.png` file.

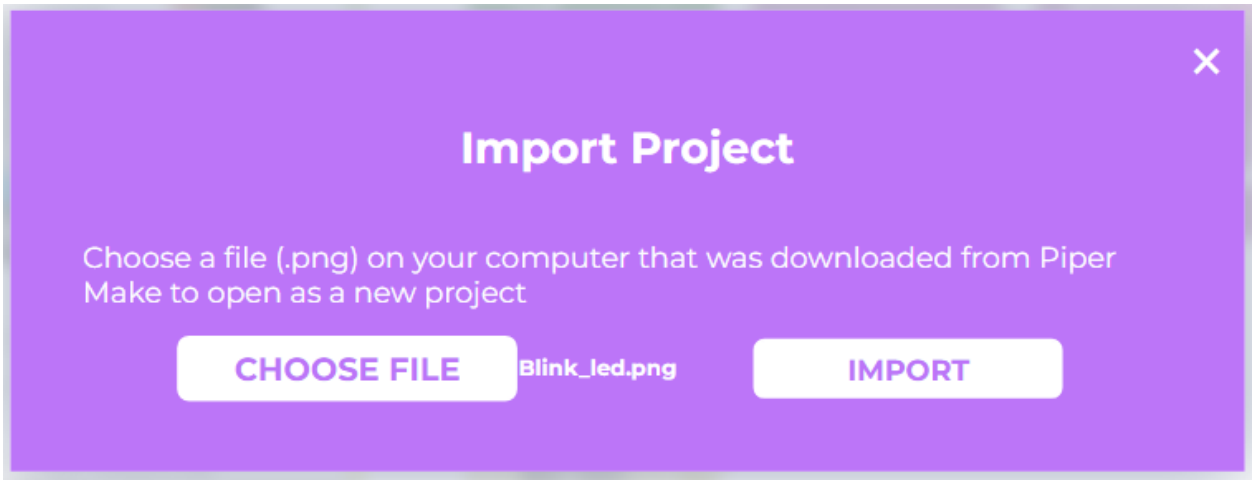


7.3.2 Import the Code

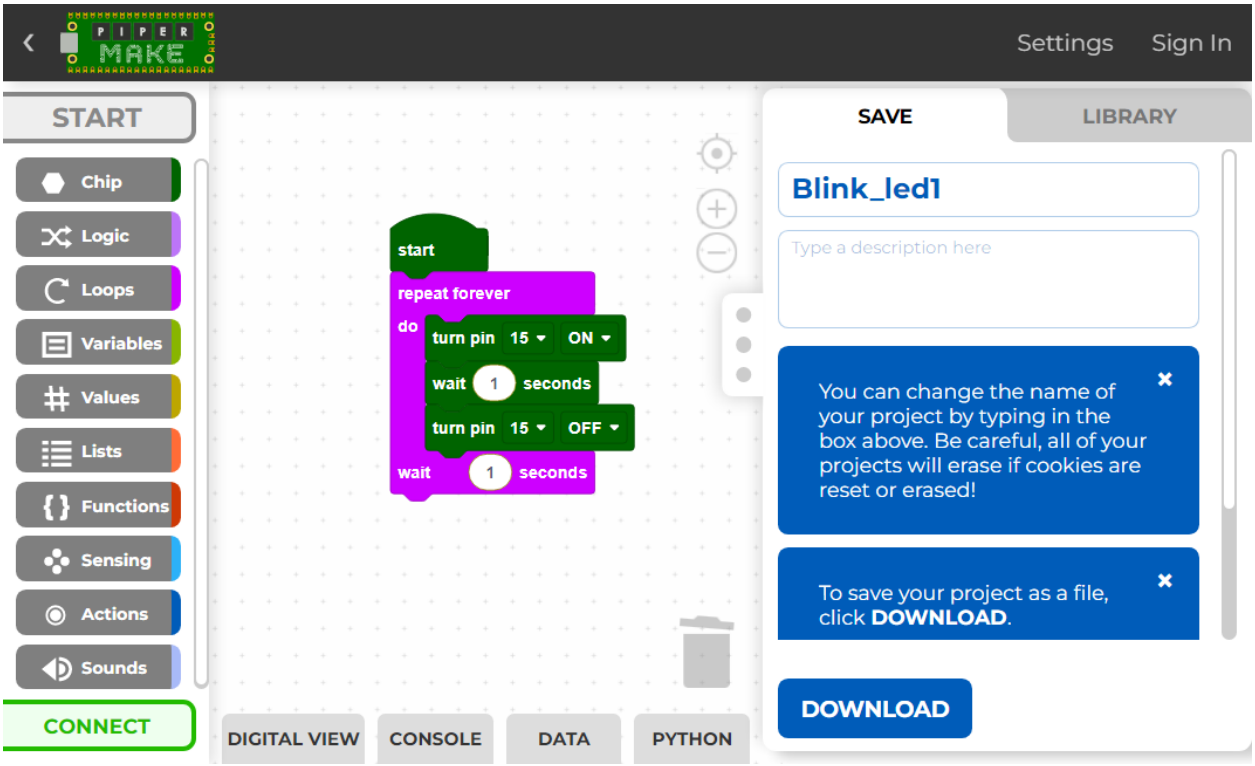
In Piper Make's home page, click **Import Project**.



Select the `.png` file in the path `kepler-kit-main\piper` and click **Import**. Note that you need to download the SunFounder Kepler Kit package first. Or check out the code at Kepler Kit - GitHub.



Now you can see the file you imported.



2. Projects

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.

- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

7.4 2.1 Blink LED

For this project, we intend to make the extended LED light blink. To use extended electronic components, a solderless breadboard will be the most powerful partner for novice users.

The breadboard is a rectangular plastic plate with a bunch of small holes in it. These holes allow us to easily insert electronic components and build electronic circuits. The breadboard does not permanently fix the electronic components, which makes it easy for us to repair the circuit and start over when we make a mistake.

Required Components

In this project, we need the following components.

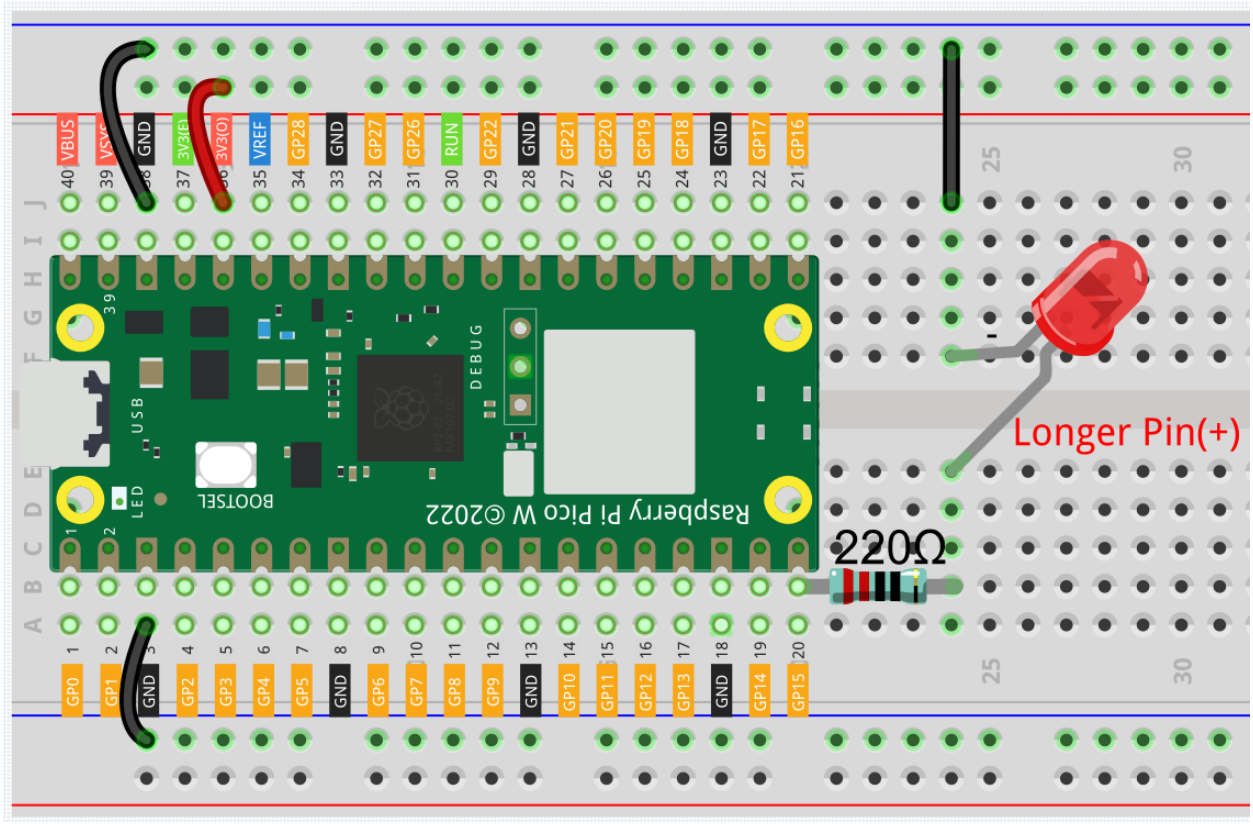
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUANTITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(220)	
6	*LED*	1	

Wiring

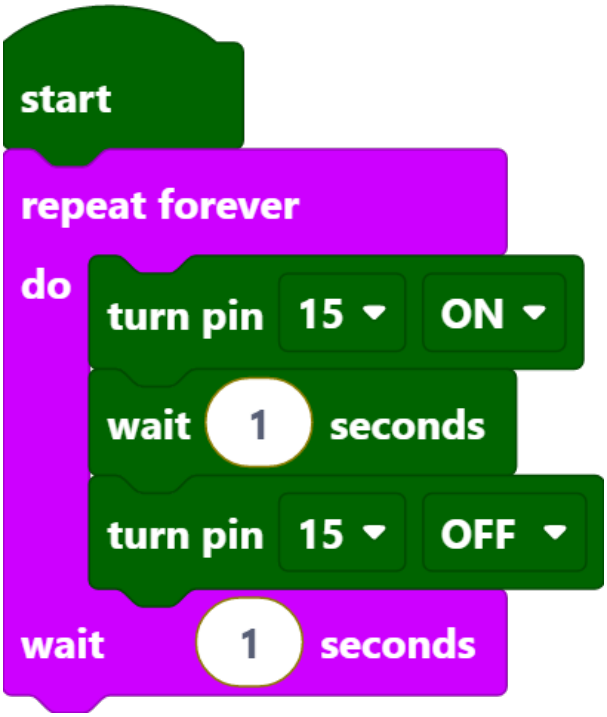


- The color ring of the 220 ohm resistor is red, red, black, black and brown.
- The longer lead of the LED is known as the anode (+), the shorter lead is the cathode (-).

Code

Note:

- You can refer to the image below to write code by dragging and dropping.
- Import `2.1_blink_led.png` from the path of `kepler-kit-main\piper`. For detailed tutorials, please refer to *Import the Code*.



After connecting Pico W, click the **Start** button and you will see the LED blinking. For more details, you can refer to *1.2 Quick Guide on Piper Make*.

How it Works?

This is the body of the loop: turn the pin15 on to light up the LED, wait one second, then turn the pin15 off to make the LED go off. Wait 1 second and then re-run the previous cycle, so you can see the LED has been in the state of alternating between light and off.

- [start]: This block is the basic framework of the program and represents the beginning of the program.
- [repeat forever do() wait()seconds]: Means that the blocks in it will be executed repeatedly, and the execution time interval is defined by yourself.
- [turn pin () ON/OFF]: Indicates that a certain pin is placed in a high level (ON) or a low level (OFF).
- [wait () seconds]: Set the execution interval between blocks.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

7.5 2.2 Button

In this project, we will learn how to turn on or off the LED by using a button.

Required Components

In this project, we need the following components.

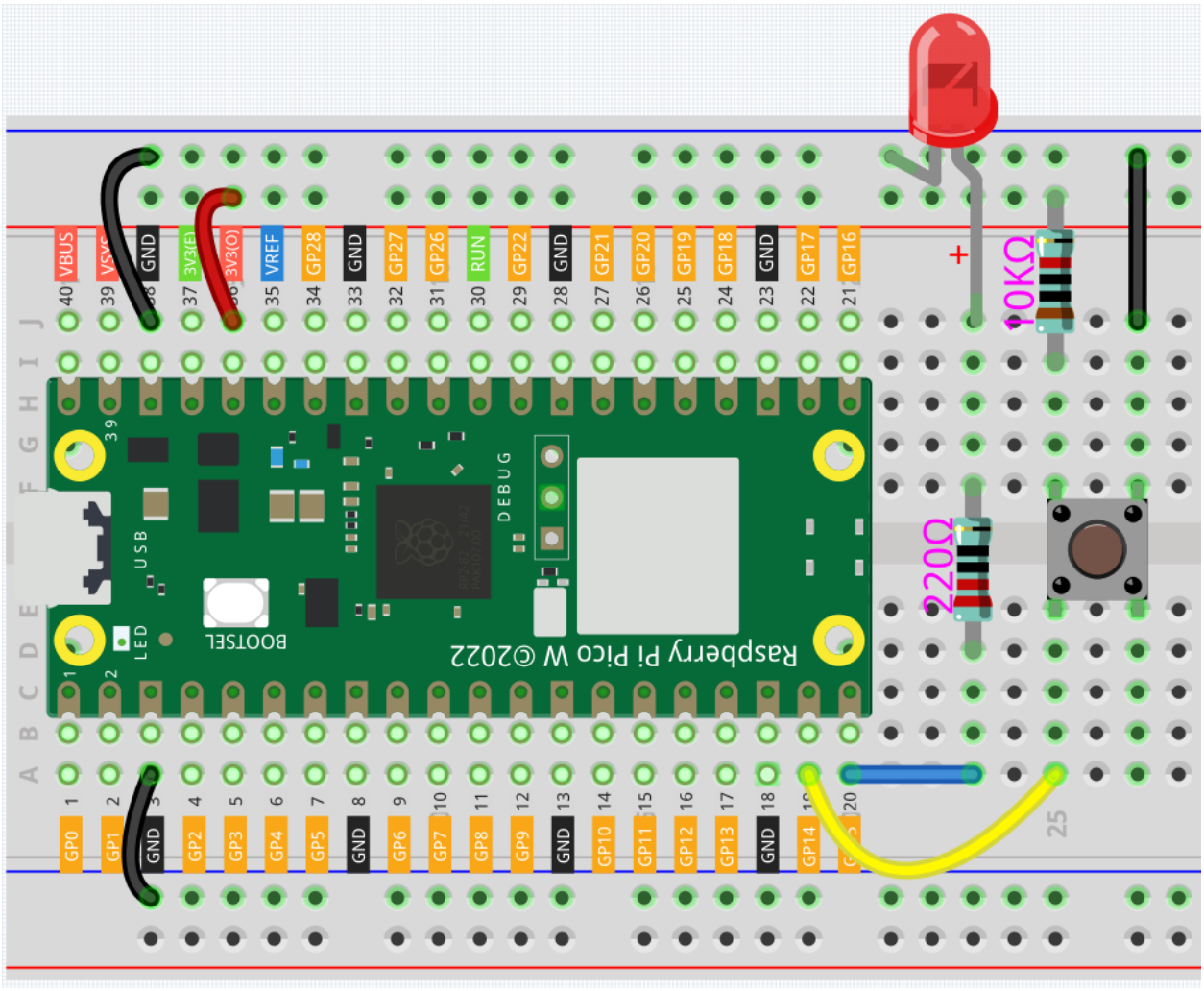
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	2(220, 10K)	
6	*LED*	1	
7	*Button*	1	

Wiring

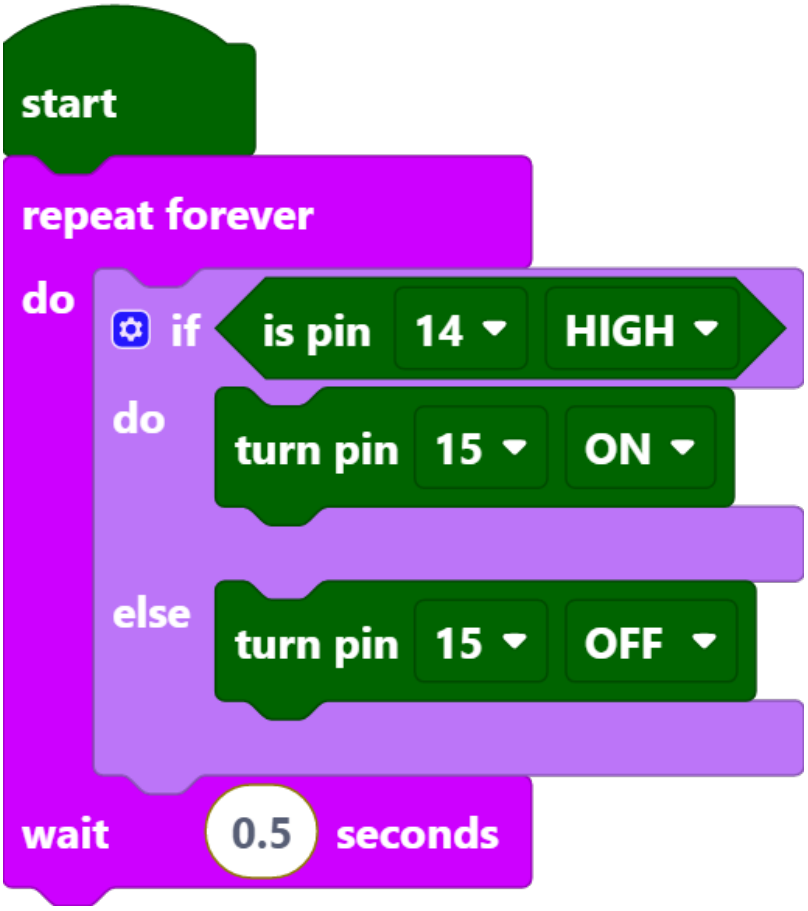


- One side of the button pin is connected to 3.3v, and the other side pin is connected to GP14, so when the button is pressed, GP14 will be high.
- However, when the button is not pressed, GP14 is in a suspended state and may be high or low. In order to get a stable low level when the button is not pressed, GP14 needs to be reconnected to GND through a 10K pull-down resistor.

Code

Note:

- You can refer to the image below to write code by dragging and dropping.
- Import `2.2_button.png` from the path of `kepler-kit-main\piper`. For detailed tutorials, please refer to *Import the Code*.



After connecting Pico W, click the **Start** button and the code starts to run. When the button is pressed, the LED will be lit. When the button is released, the LED will go out.

How it Works?

When the button is pressed, pin14 is high. So if the read pin14 is high, turn the pin15 on (LED is lit); else, turn off the pin15 (LED is off).

- [if () do () else ()]: This is a judgment block, depending on the condition after the [if] block to determine whether to run the blocks inside the [do] block, or the blocks inside the [else] block.
- [is pin () HIGH]: This is used to read the level of a specific pin, if the level read is the same as the set HIGH/LOW, then execute the blocks inside [do] block, otherwise execute the blocks inside [else].

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

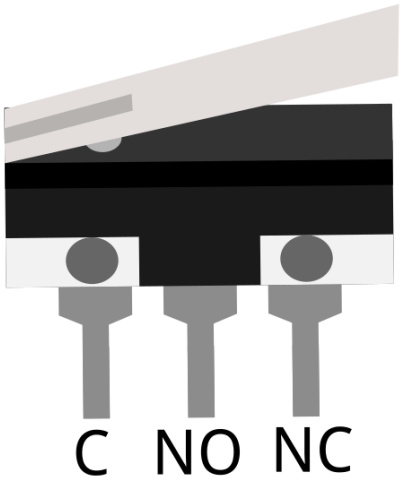
Ready to explore and create with us? Click [] and join today!

7.6 2.3 Service Bell

For this project we used a microswitch and an active buzzer to create a service bell. Tap the switch and the buzzer makes a sound.

Micro Switch is also a 3-pin device, the sequence of the 3 pins are C (common pin), NO (normally open) and NC (normally closed).

When the micro switch is not pressed, 1 (C) and 3 (NC) are connected together, when pressed 1 (C) and 2 (NO) are connected together.



Required Components

In this project, we need the following components.

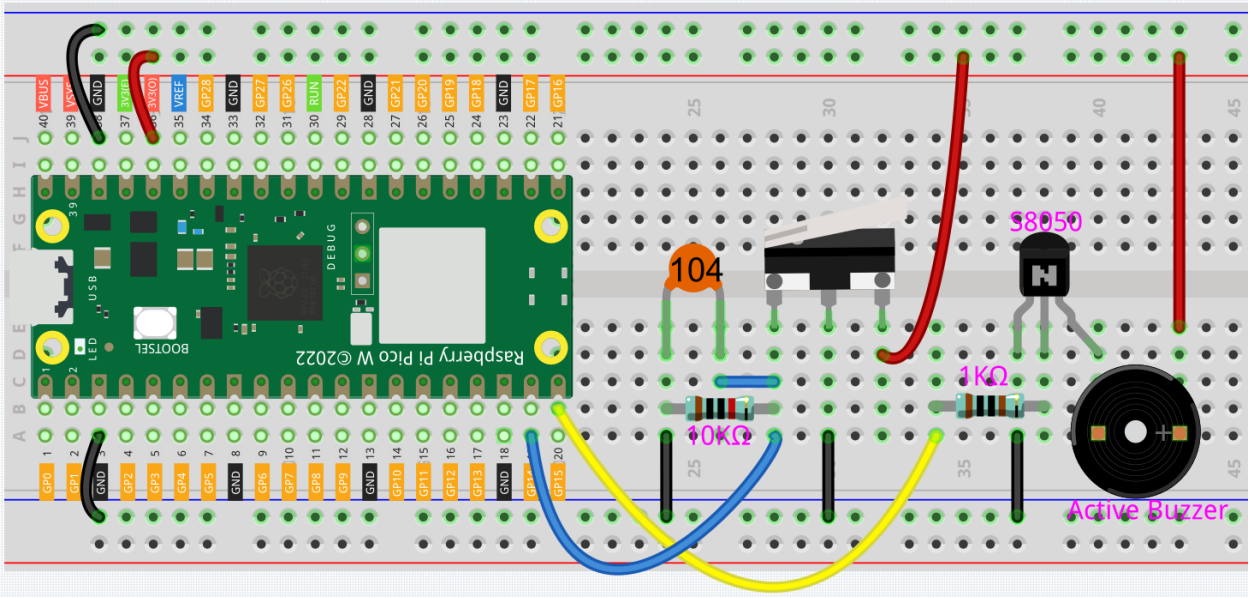
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	2(1K, 10K)	
6	*Capacitor*	1(104)	
7	*Micro Switch*	1	
8	*Transistor*	1(S8050)	
9	Active *Buzzer*	1	

Wiring

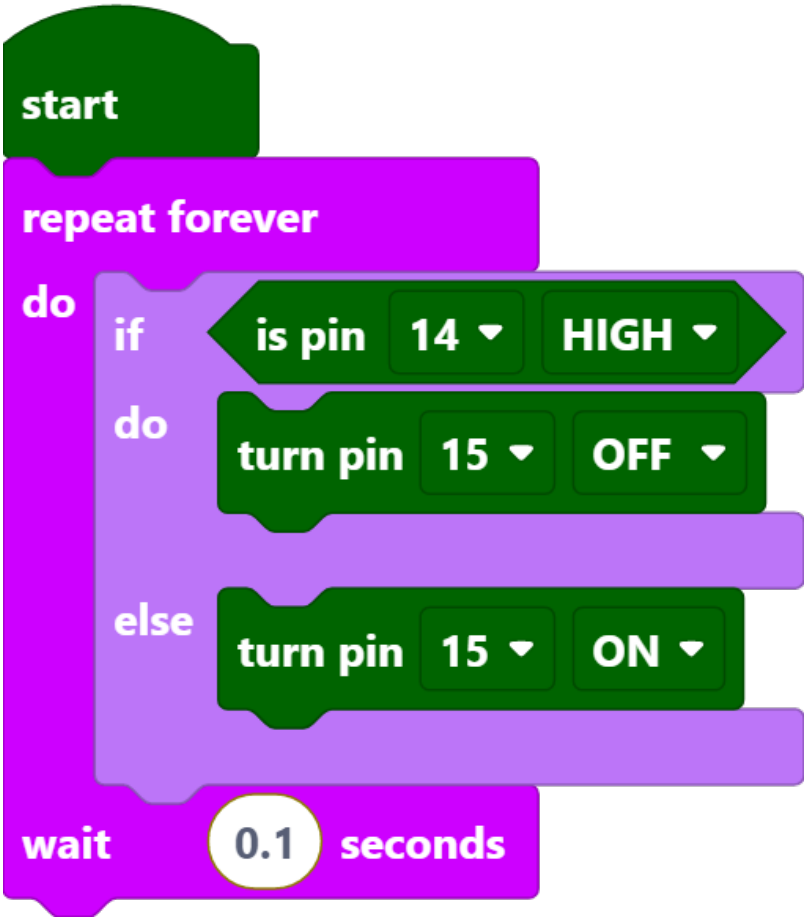


- By default, pins 1 and 3 of the Micro Switch are connected together and GP14 is low. When the Micro Switch is pressed, GP14 is high.
- GP15 outputs high to make the buzzer sound.

Code

Note:

- You can refer to the image below to write code by dragging and dropping.
- Import `2.3_service_bell.png` from the path of `kepler-kit-main\piper`. For detailed tutorials, please refer to *Import the Code*.



After connecting Pico W, click the **Start** button and the code starts to run. Tap the switch and the buzzer makes a sound.

Note: This project code is exactly the same as the previous project *2.2 Button*.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

7.7 2.4 Rainbow Light

In this project, we will make the RGB LEDs display a rainbow of colors.

RGB LED is equivalent to encapsulating Red LED, Green LED, Blue LED under one lamp cap, and the three LEDs share one cathode pin. Since the electric signal is provided for each anode pin, the light of the corresponding color can be displayed. By changing the electrical signal intensity of each anode, it can be made to produce various colors.

Required Components

In this project, we need the following components.

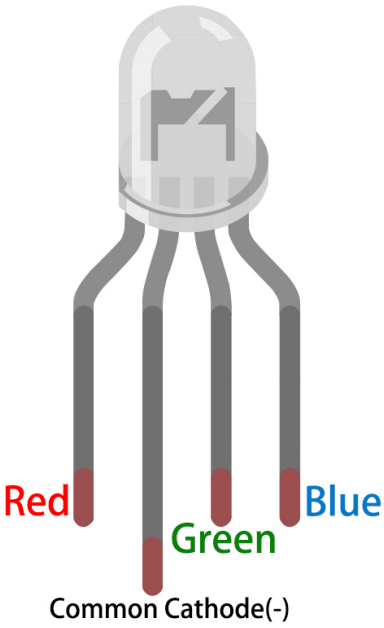
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

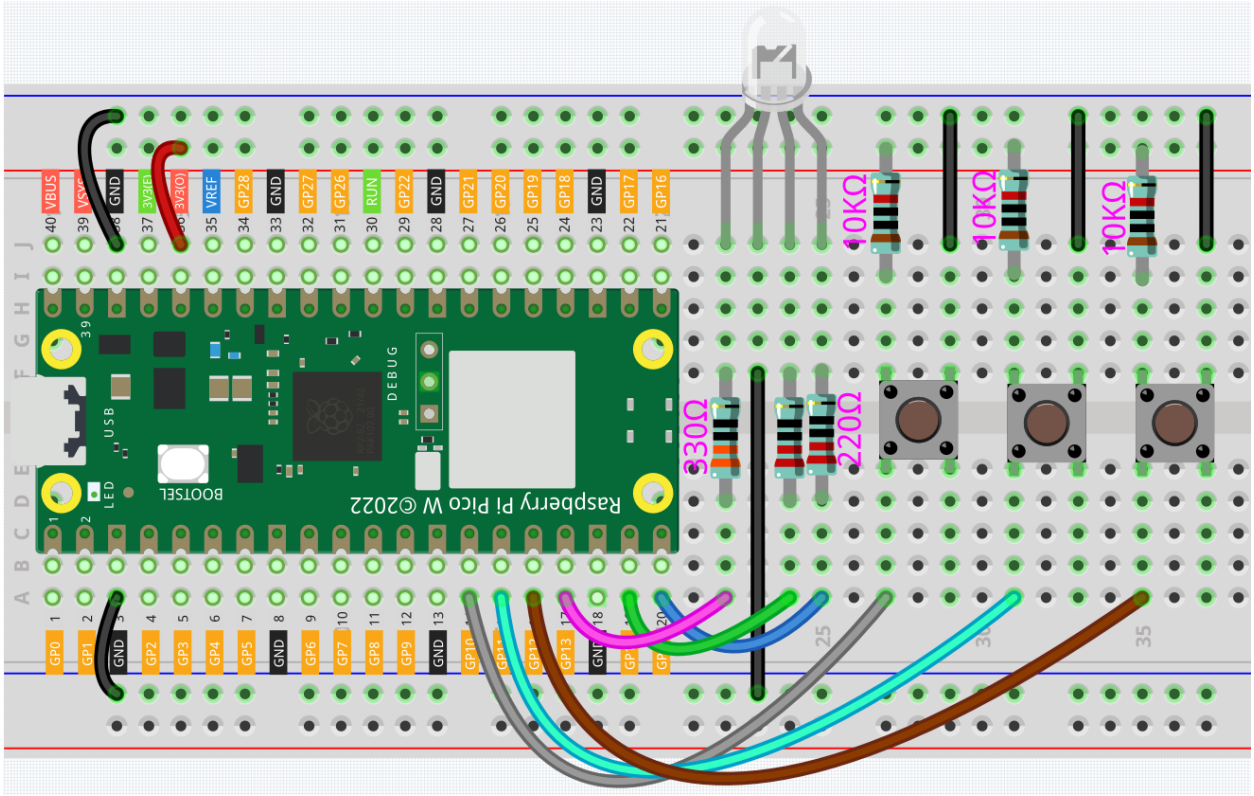
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	6(1-330, 2-220, 3-10K)	
6	*Button*	3	
7	*RGB LED*	1	

Wiring



An RGB LED has 4 pins: the longest pin is the common cathode pin, which is usually connected to GND, the left pin next to the longest pin is Red, and the 2 pins on the right are Green and Blue.

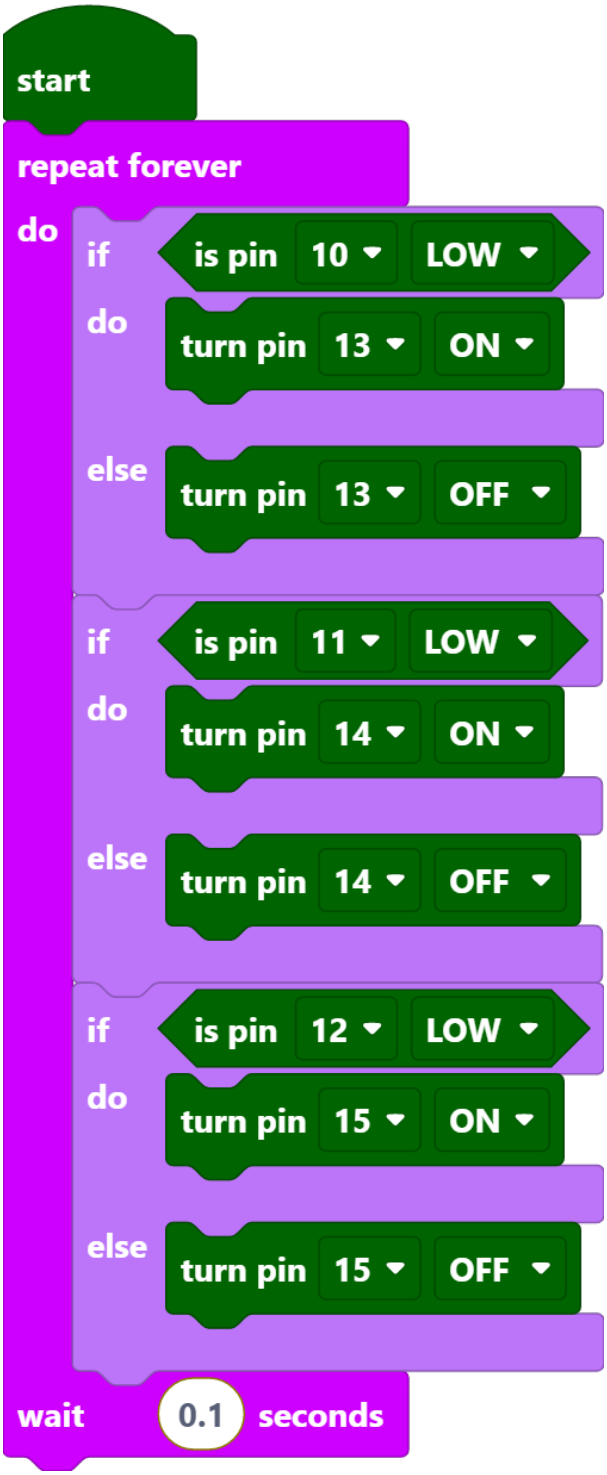


- When using the same power supply intensity, the Red LED will be brighter than the other two, and a slightly larger resistor(330) needs to be used to reduce its brightness.
- The 3 buttons are used to control the lighting of the Red, Green and Blue LEDs respectively.

Code

Note:

- You can refer to the image below to write code by dragging and dropping.
- Import `2.4_rainbow_light.png` from the path of `kepler-kit-main\piper`. For detailed tutorials, please refer to *Import the Code*.



After connecting Pico W, click the **Start** button and the code starts to run. Pressing these buttons individually will emit a single color of light, but if two of the buttons are pressed at the same time, or all 3 buttons are pressed at the same time, the RGB LEDs will emit a variety of different colors, up to a maximum of 7.

Note: In fact, RGB LED can emit up to 16 million colors, but since Piper Make does not have a block to output PWM signal, here we just use the [turn pin() (ON/OFF)] block to make RGB LEDs show 7 colors.

How it Works?

You can think of this project as using three buttons to control the RGB LED, and setting three if judgment conditions to determine whether the three buttons are pressed or not. When the buttons are pressed, the levels of the corresponding pins are pulled high, causing the RGB LED to display different colors.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

7.8 2.5 Drum Kit

For this project, let's make a drum kit with three buttons and a slide switch. Now, play your own drums.

Required Components

In this project, we need the following components.

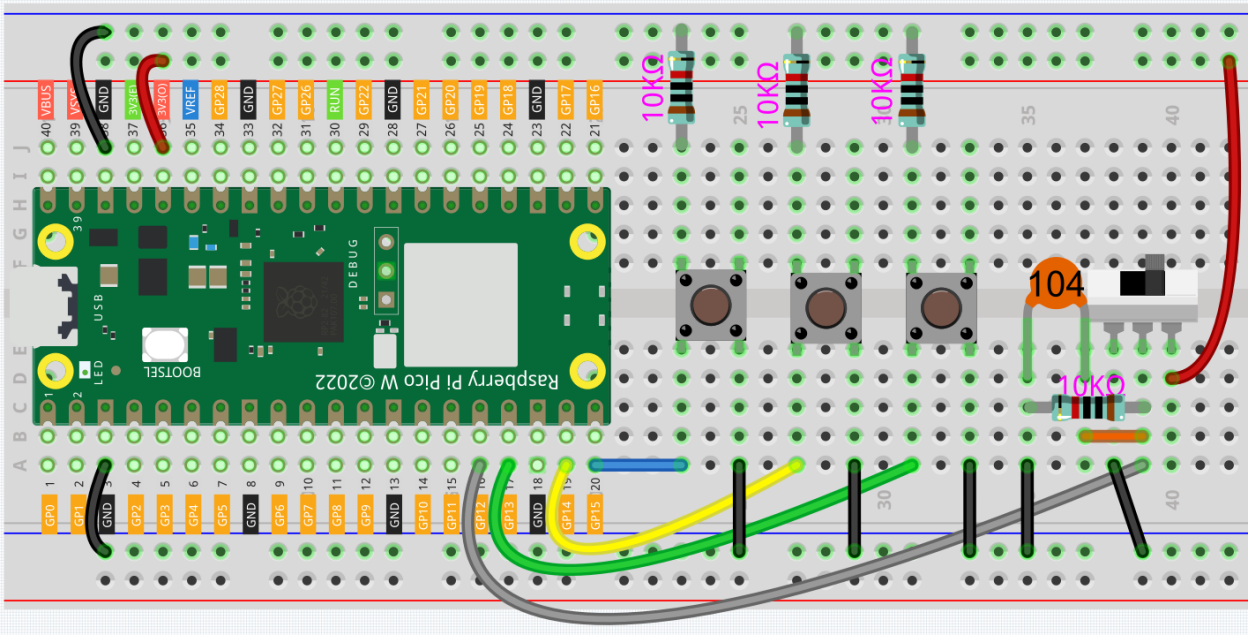
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	4(10K)	
6	*Button*	3	
7	*Capacitor*	1(104)	
8	*Micro Switch*	1	

Wiring



- When the slide switch is toggled to the right, GP12 is high; when toggled to the left, GP12 is low.
- Each of the three buttons is connected to a pull-down resistor, GP13~GP15 are low by default; when the button is pressed, GP13~GP15 are high.

Code

Note:

- You can refer to the image below to write code by dragging and dropping.
- Import `2.5_drum_kit.png` from the path of `kepler-kit-main\piper`. For detailed tutorials, please refer to *Import the Code*.



After connecting Pico W, click the **Start** button and the code starts to run. Pressing different buttons or toggling the slide switch will produce different drum sounds, just like a drum kit.

Note: If you are using a computer, you will need to plug in headphones or audio to your computer to hear the sound emitted.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

7.9 2.6 Smart Water Tank

In this project, we use a water level sensor module and a servo to simulate a smart water tank. The water level sensor is fixed inside the tank to measure the water level, and when the water level is below a threshold, the valve controlled by the servo is allowed to open to let water in.

Required Components

In this project, we need the following components.

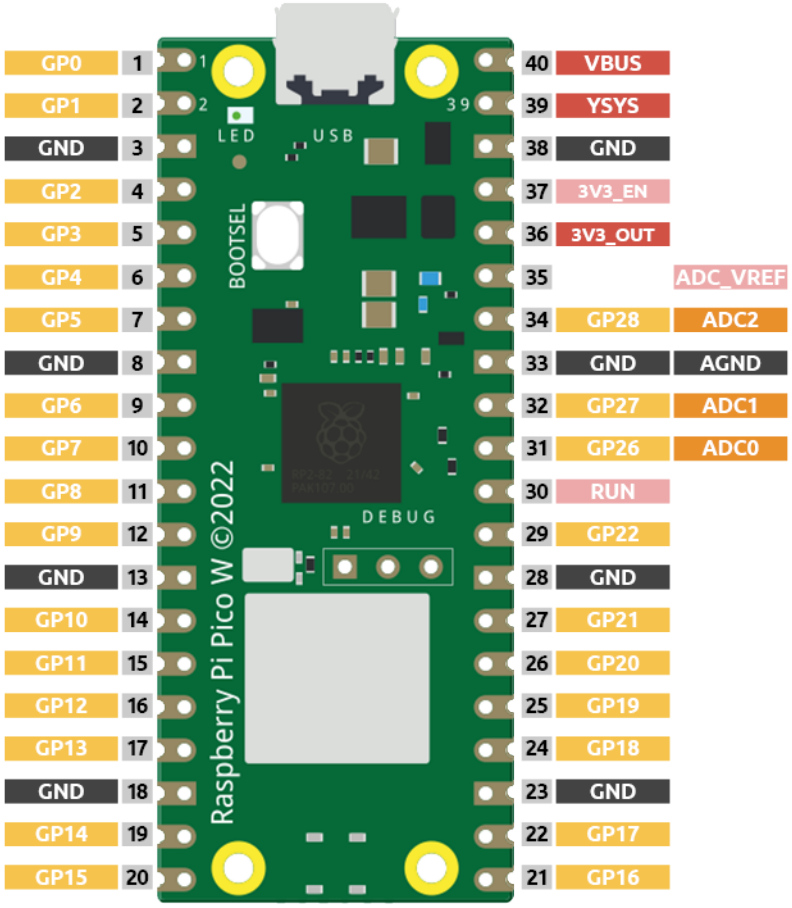
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

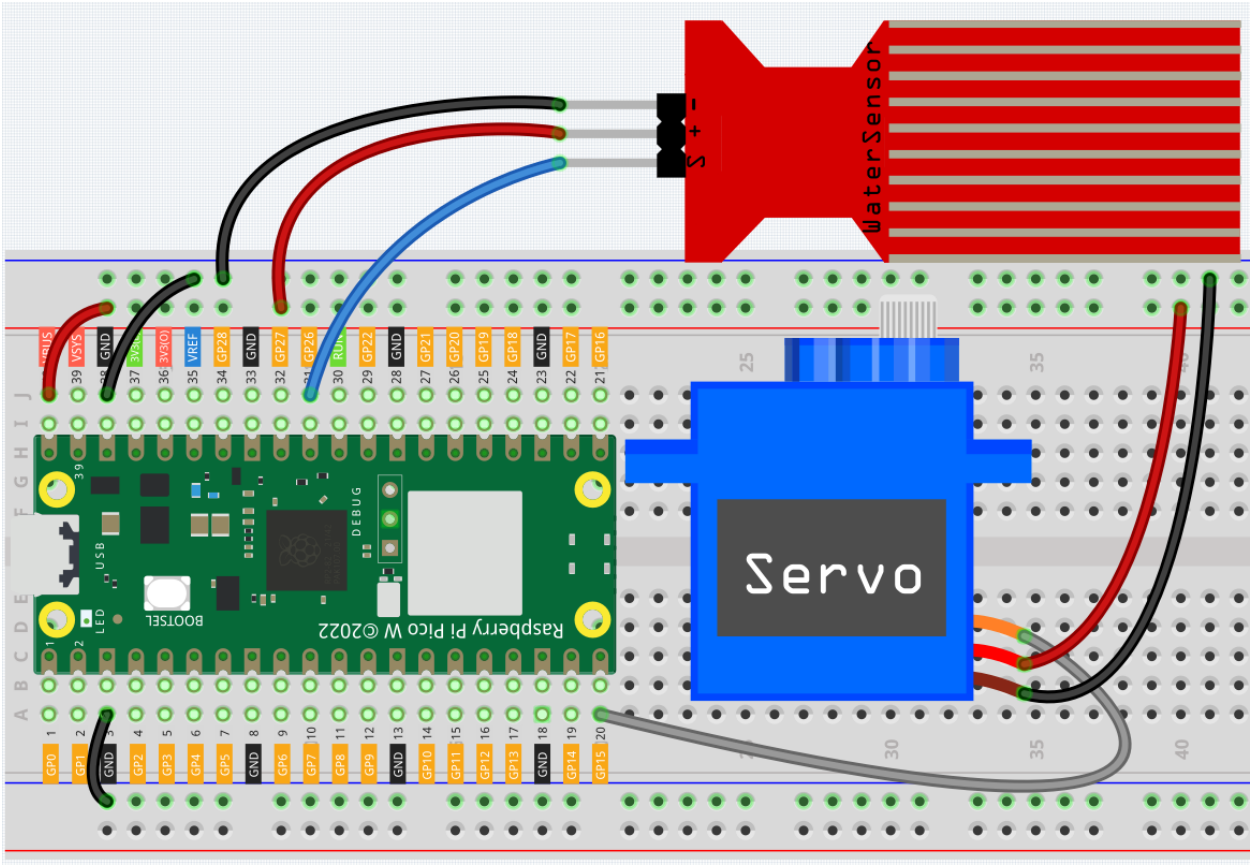
You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Servo*	1	
6	*Water Level Sensor Module*	1	

Wiring



Pico W has three GPIO pins that can use analog input, GP26, GP27, GP28. That is, analog channels 0, 1, and 2. In addition, there is a fourth analog channel, which is connected to the built-in temperature sensor and will not be introduced here.

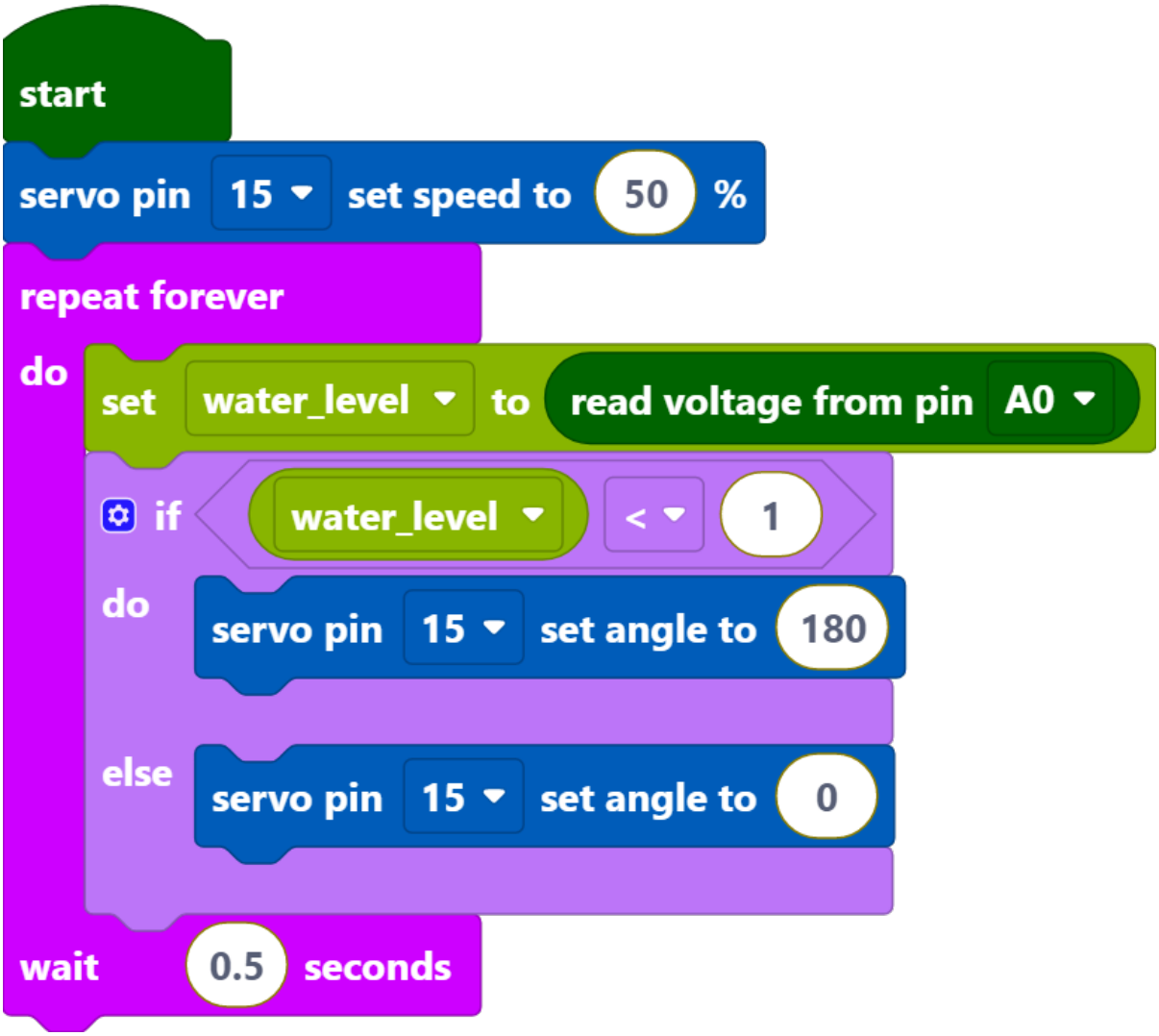


- S of water level sensor is connected to GP26(A0), + to VBUS, - pin to GND.
- The orange wire (signal) of the servo is connected to GP15, the red wire (power) is connected to VBUS, and the brown wire (ground) is connected to GND.

Code

Note:

- You can refer to the image below to write code by dragging and dropping.
- Import `2.6_water_tank.png` from the path of `kepler-kit-main\piper`. For detailed tutorials, please refer to *Import the Code*.



After connecting Pico W, click the **Start** button and the code starts to run. When the water level is lower than 1/3 of the water level sensor, the servo rotates to 180 degrees to simulate the inlet opening; when the water level is higher than 1/3 of the water level sensor, the servo rotates to 0 degrees to imitate the inlet closing.

How it Works?



Set the rotation speed of pin15 (servo) to 15%.

- [servo pin() set speed to ()%]: Used to set the rotation speed of the servo pin, the range is 0%~100%.



Reads the value of pin A0 and stores it in the variable [water_level].

- [set (water_level) to]: Used to set the value of the variable, you need to create the variable from the **Variables** palette.
- [read voltage from pin ()]: Used to read the voltage of the analog pins (A0~A2), the range is 0 ~ 3.3V.



Set the voltage threshold to 1. When the voltage of water level sensor is less than 1, let the servo rotate to 180° position, otherwise rotate to 0° position.

- [servo pin () set angle to ()]: Set the angle of servo pin to, the range is 0 ~ 180°.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

7.10 2.7 Swing Servo

In this project, we use a servo and a potentiometer to simulate a steering wheel. Rotating the potentiometer will drive the servo to turn together.

Required Components

In this project, we need the following components.

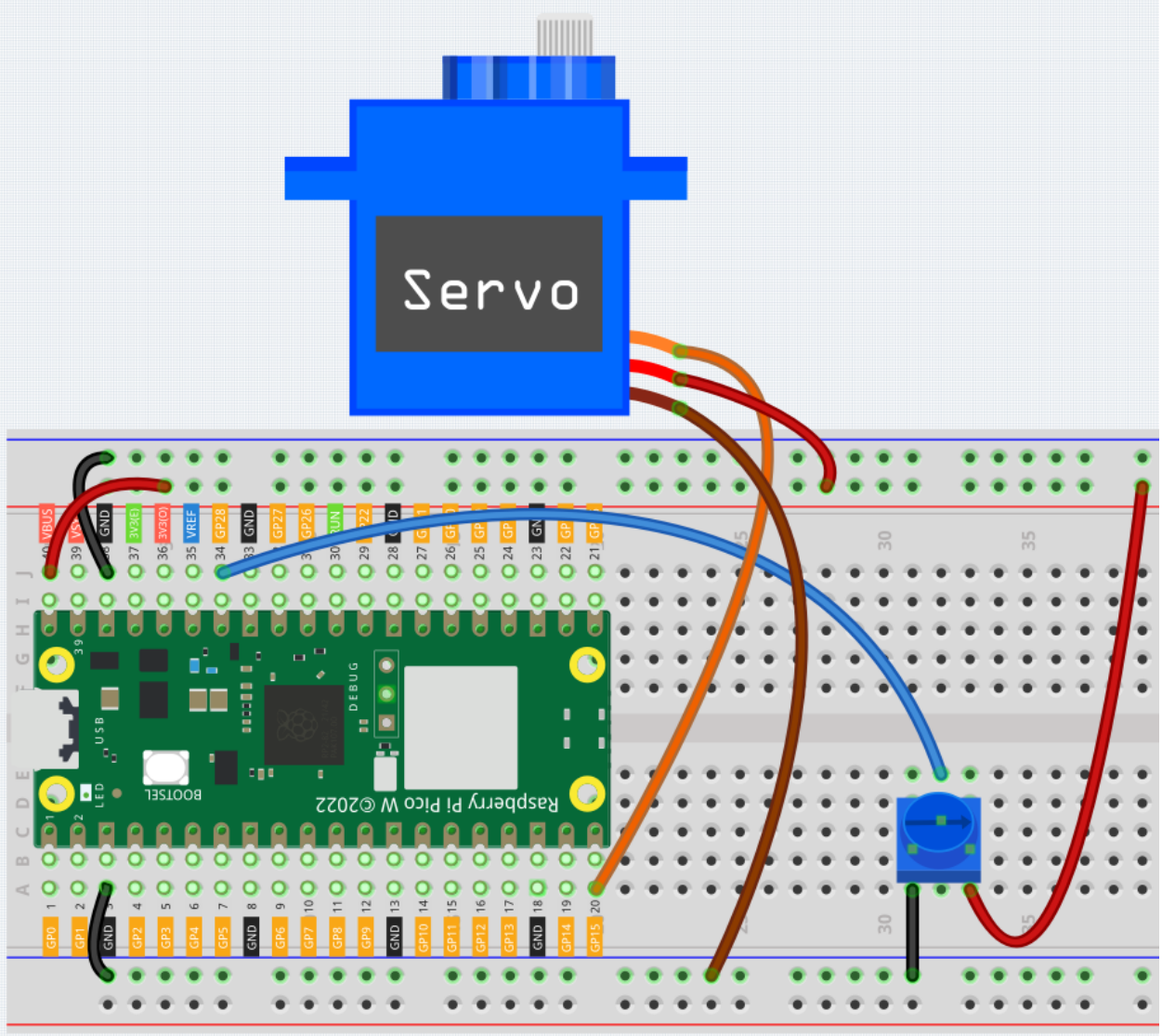
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Servo*	1	
6	*Potentiometer*	1	

Wiring

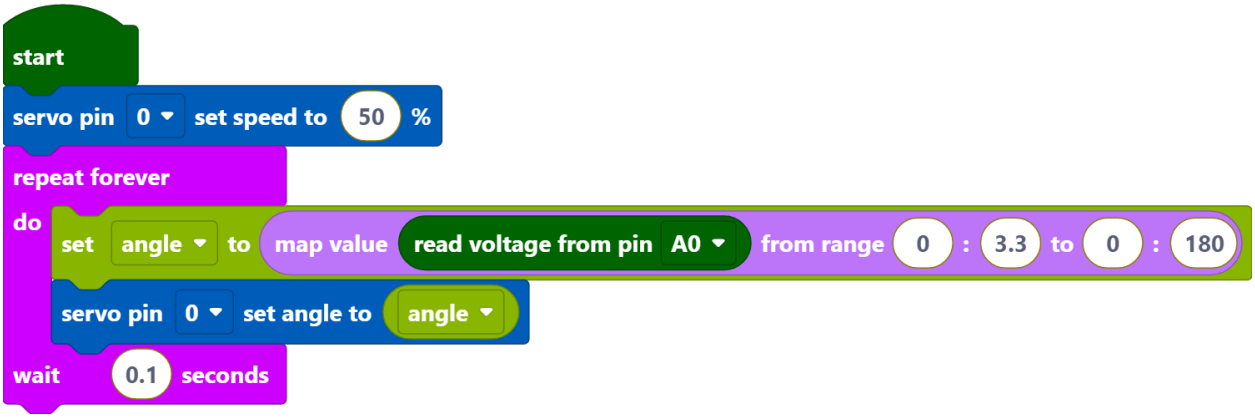


- The orange wire (signal) of the servo is connected to GP15, the red wire (power) is connected to VBUS, and the brown wire (ground) is connected to GND.
- Potentiometer is a resistive element with 3 terminals, the 2 side pins are connected to 5V and GND, and the middle pin is connected to GP26(A0).

Code

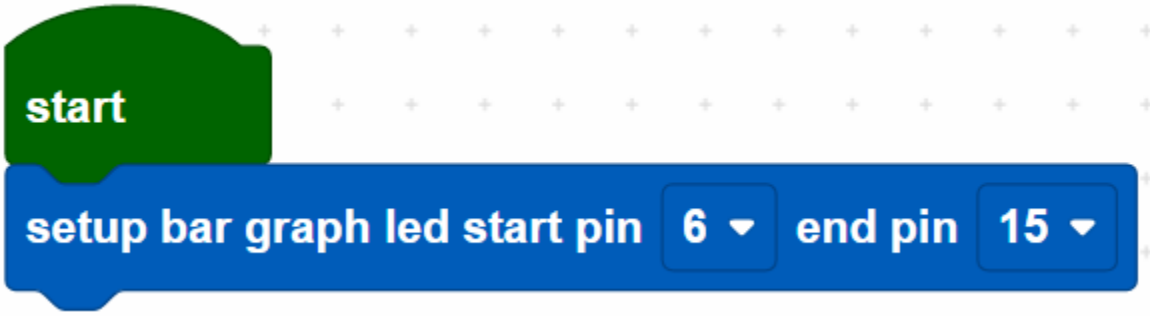
Note:

- You can refer to the image below to write code by dragging and dropping.
- Import `2.7_swing_servo.png` from the path of `kepler-kit-main\piper`. For detailed tutorials, please refer to *Import the Code*.



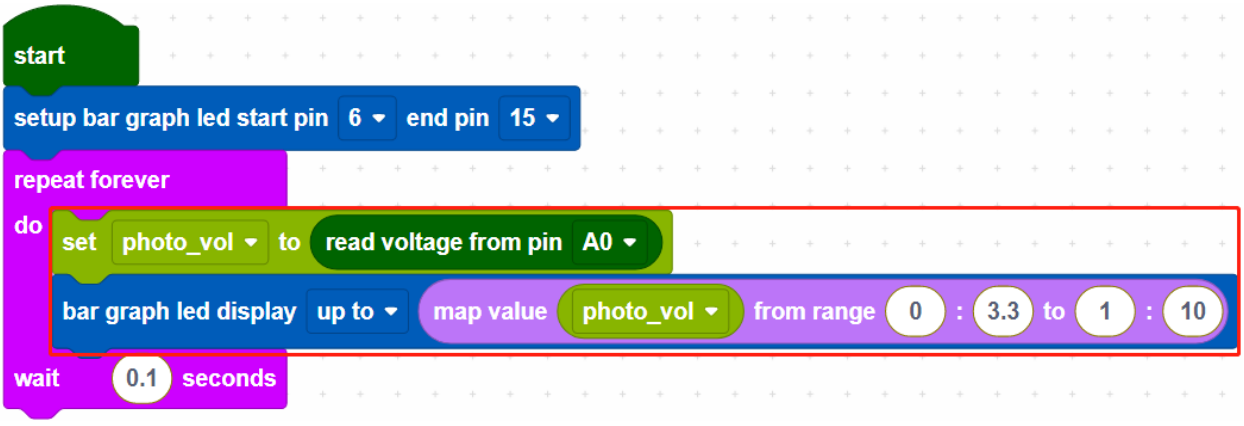
- After connecting Pico W, click the **Start** button and the code starts to run.
- Turn the potentiometer and the servo will follow. To see it clearly, you can insert a rocker arm in the servo shaft.

How it Works?



Set the rotation speed of pin15 (servo) to 15%.

- [servo pin() set speed to ()%]: Used to set the rotation speed of the servo pin, the range is 0%~100%.



Create a variable [angle], then read the voltage of A0. Use the [map value () from () to ()] block, map the voltage of A0 from 0 to 3.3V voltage range to 0 to 180°, and then use the mapped angle as the rotation angle of the servo.

- [map value () from () to ()]: map a value from one range to another.

Note: The voltage of A0~A2 takes the range of 0~3.3V, even if your power supply is connected to VBUS (5V).

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

7.11 2.8 Light Intensity Display

In this project, we use a photoresistor and the LED Bar Graph to make a light intensity display, where the stronger the light is the more the number of the LED Bar Graph lights up.

Required Components

In this project, we need the following components.

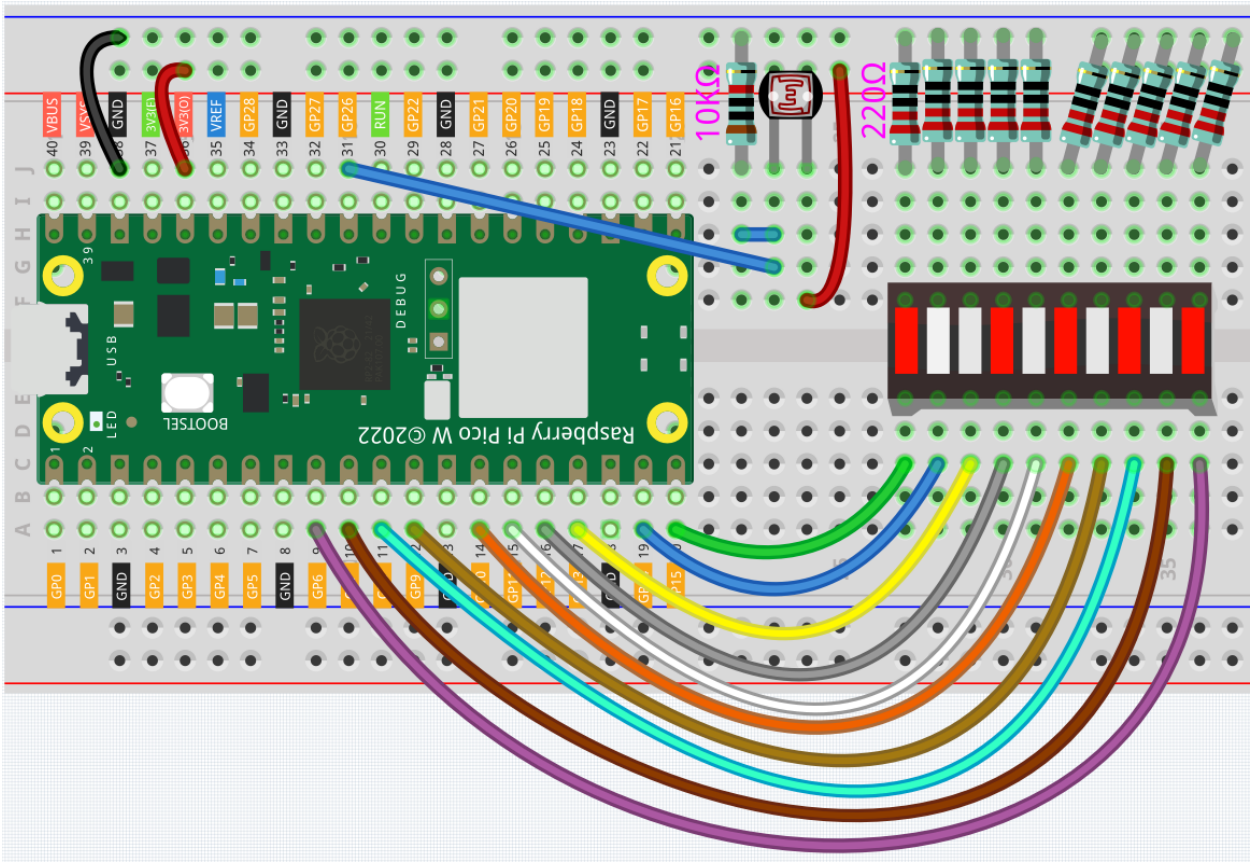
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUANTITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	11(10-220, 1-10K)	
6	*LED Bar Graph*	1	
7	*Photoresistor*	1	

Wiring

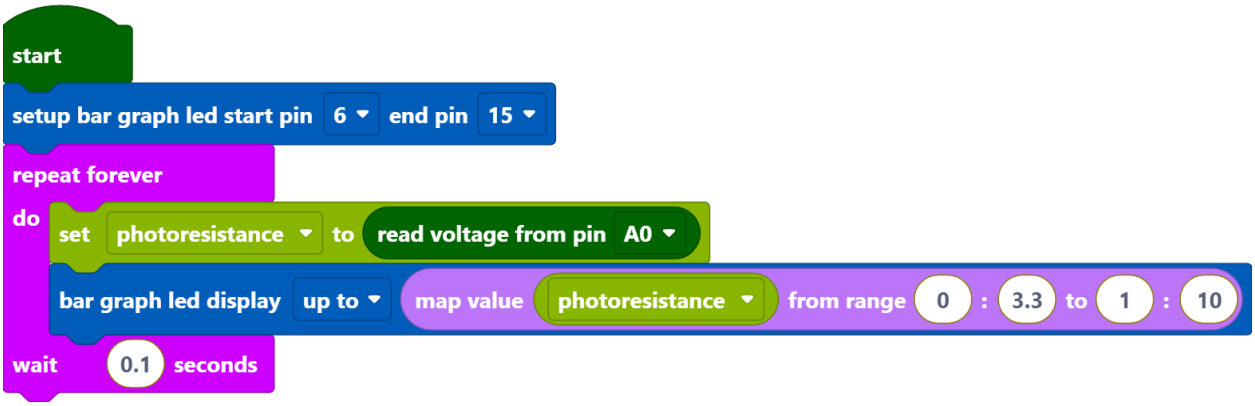


- The LED Bar Graph consists of 10 LEDs, with the side with the label as the anode and the other side as the cathode.
- The anode of the LED Bar Graph are connected to GP6~GP15. The cathode are connected to a 220 ohm resistor and then to GND.
- Connect one end of the photoresistor to 3.3V and the other end to GP26 (A0). At the same time, GP26 needs to be connected to another 10K resistor to GND. This way, when the light is stronger, the resistance of the photoresistor decreases and the voltage of A0 increases.

Code

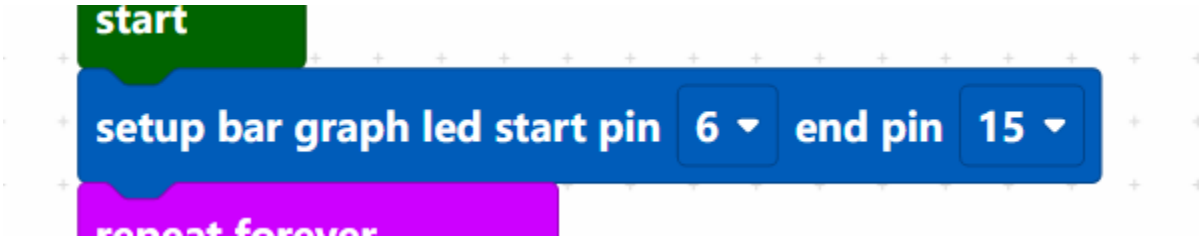
Note:

- You can refer to the image below to write code by dragging and dropping.
- Import `2.8_light_intensity_display.png` from the path of `kepler-kit-main\piper`. For detailed tutorials, please refer to *Import the Code*.

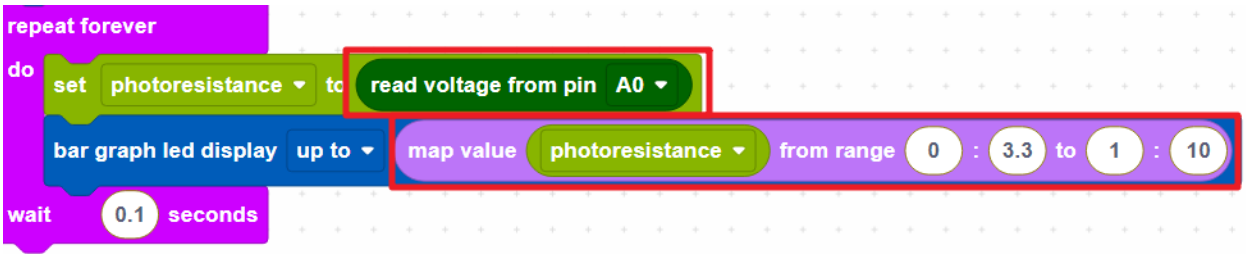


- After connecting Pico W, click the **Start** button and the code starts to run.
- When the light is stronger, more LEDs are lit on the LED Bar Graph.
- If running the code does not light up properly, you can try flipping the led bar.

How it Works?



Set the pins of LED Bar Graph connected to GP6 ~ GP15.



Store the voltage value of A0 (GP26) read into the variable [photo_vol]. Use the [map value () from () to ()] block to map the variable [photo_vol] from 0 to 3.3V to 0 to 10 (the number of LEDs on the LED Bar Graph).

- [map value () from () to ()]: map a value from one range to another.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

7.12 2.9 Lucky Cat

In this project, we use pir and servo to make a lucky cat. The PIR Module is used to detect the visitors and servo is used to imitate the beckoning motion of the lucky cat.

Required Components

In this project, we need the following components.

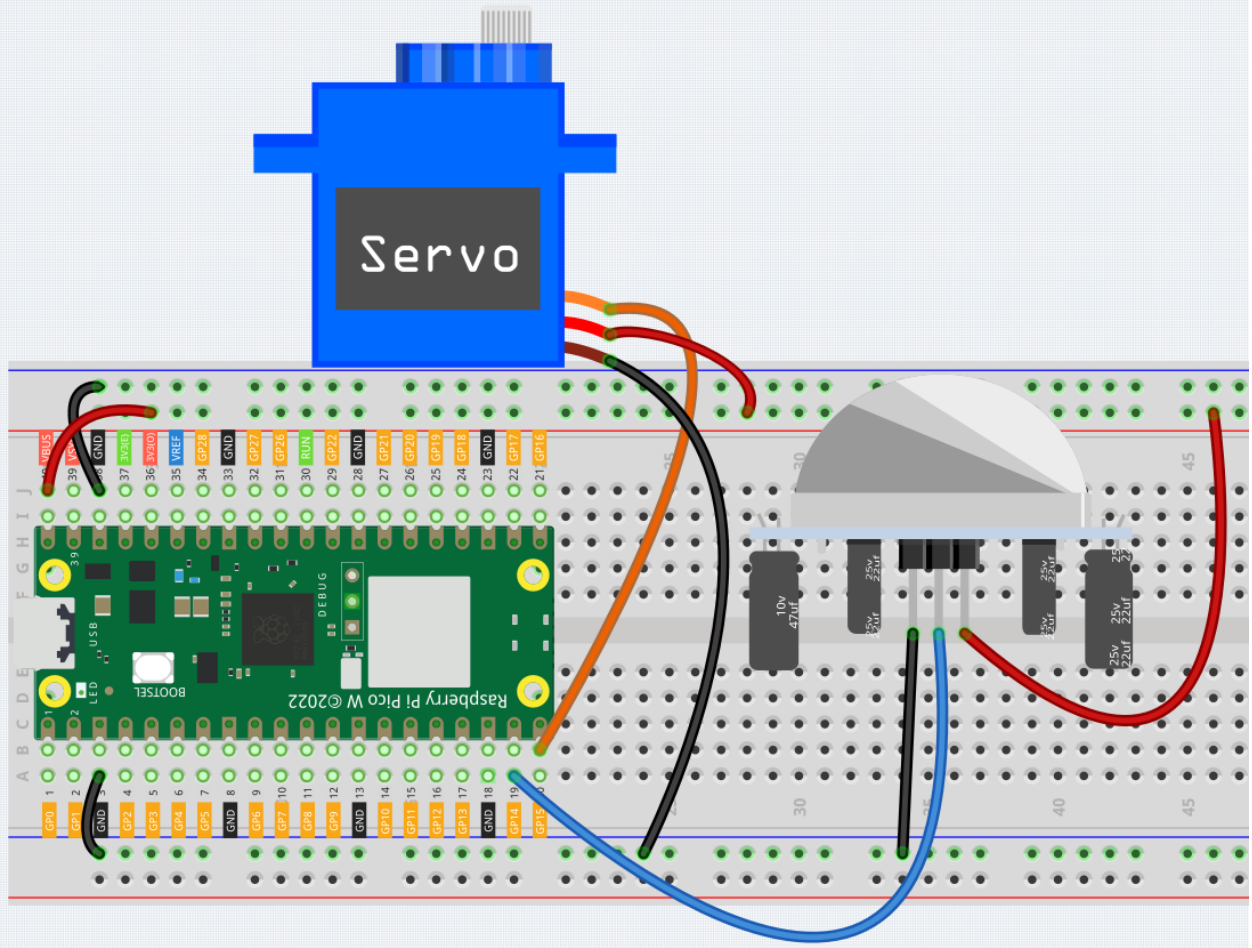
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Servo*	1	
6	*PIR Motion Sensor Module*	1	

Wiring

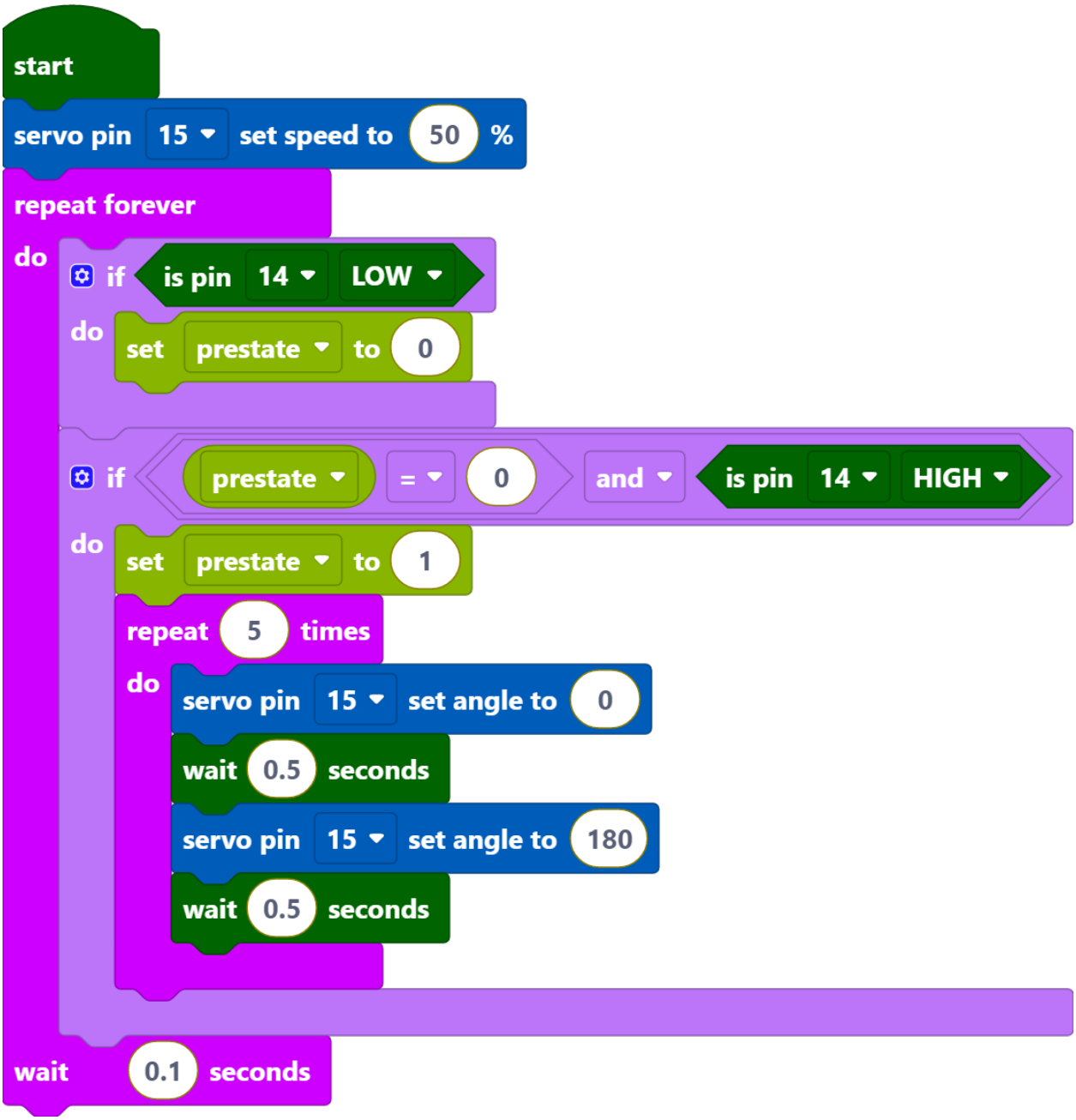


- The orange wire (signal) of the servo is connected to GP15, the red wire (power) is connected to VBUS, and the brown wire (ground) is connected to GND.
- The middle pin of the PIR module is connected to the GP3.

Code

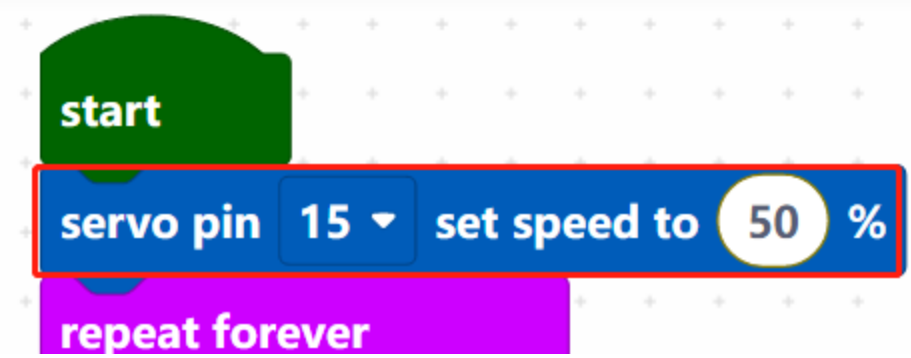
Note:

- You can refer to the image below to write code by dragging and dropping.
- Import `2.9_lucky_cat.png` from the path of `kepler-kit-main\piper`. For detailed tutorials, please refer to *Import the Code*.

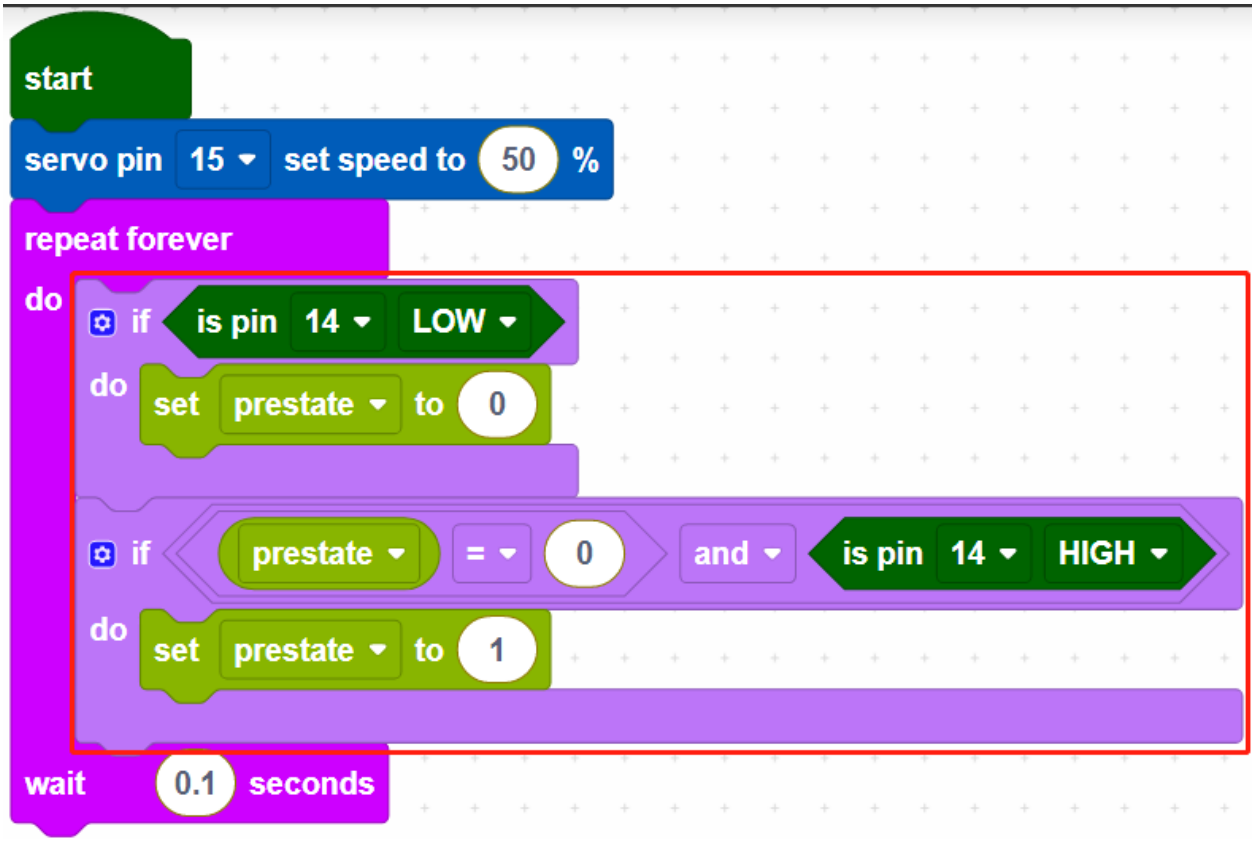


- After connecting Pico W, click the **Start** button and the code starts to run.
- When the PIR module detects a customer arrival, the servo oscillates back and forth five times and then stops.

How it Works?

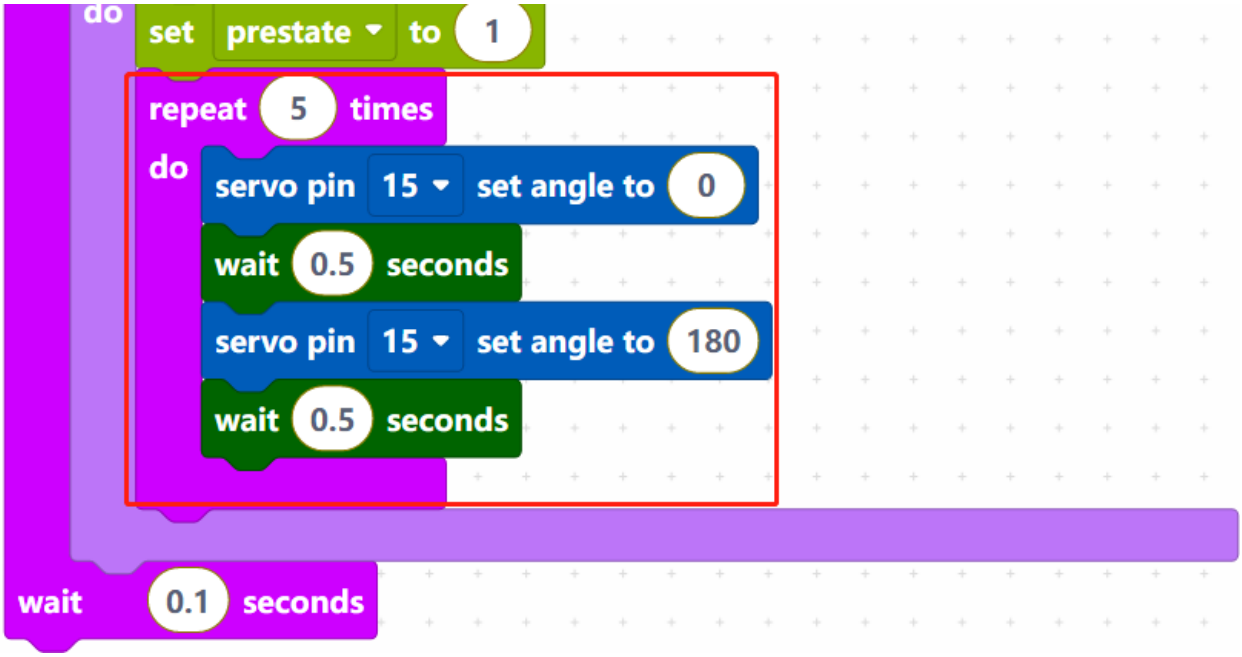


Set the rotation speed of pin15 (servo) to 15%.



If GP14 is low, set the variable [prestate] to 0. When the variable [prestate] is 0 and GP14 is high (human detected), set the variable [prestate] to 1.

The purpose of this is to make the main code work only when the GP14 changes from low to high, and to respond only once if the PIR module keeps detecting people.



Make the servo cycle 5 times from 0 to 180 degrees rotation.

- [repeat () times do]: Loop through the code inside the do block a finite number of times.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

7.13 2.10 Flowing LEDs

The kit is equipped with a WS2812 RGB LED Strip, which can display colorful colors, and each LED can be independently controlled.

Here we try to use the tilt switch to control the flow direction of the LEDs on the WS2812 RGB LED Strip.

Required Components

In this project, we need the following components.

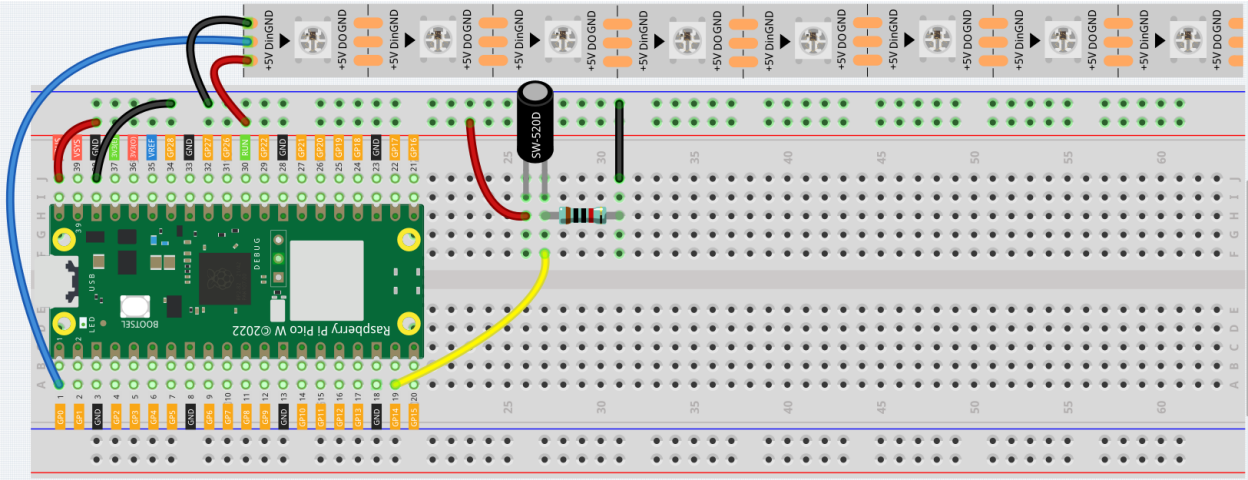
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Resistor*	1(10K)	
6	*Tilt Switch*	1	
7	*WS2812 RGB 8 LEDs Strip*	1	

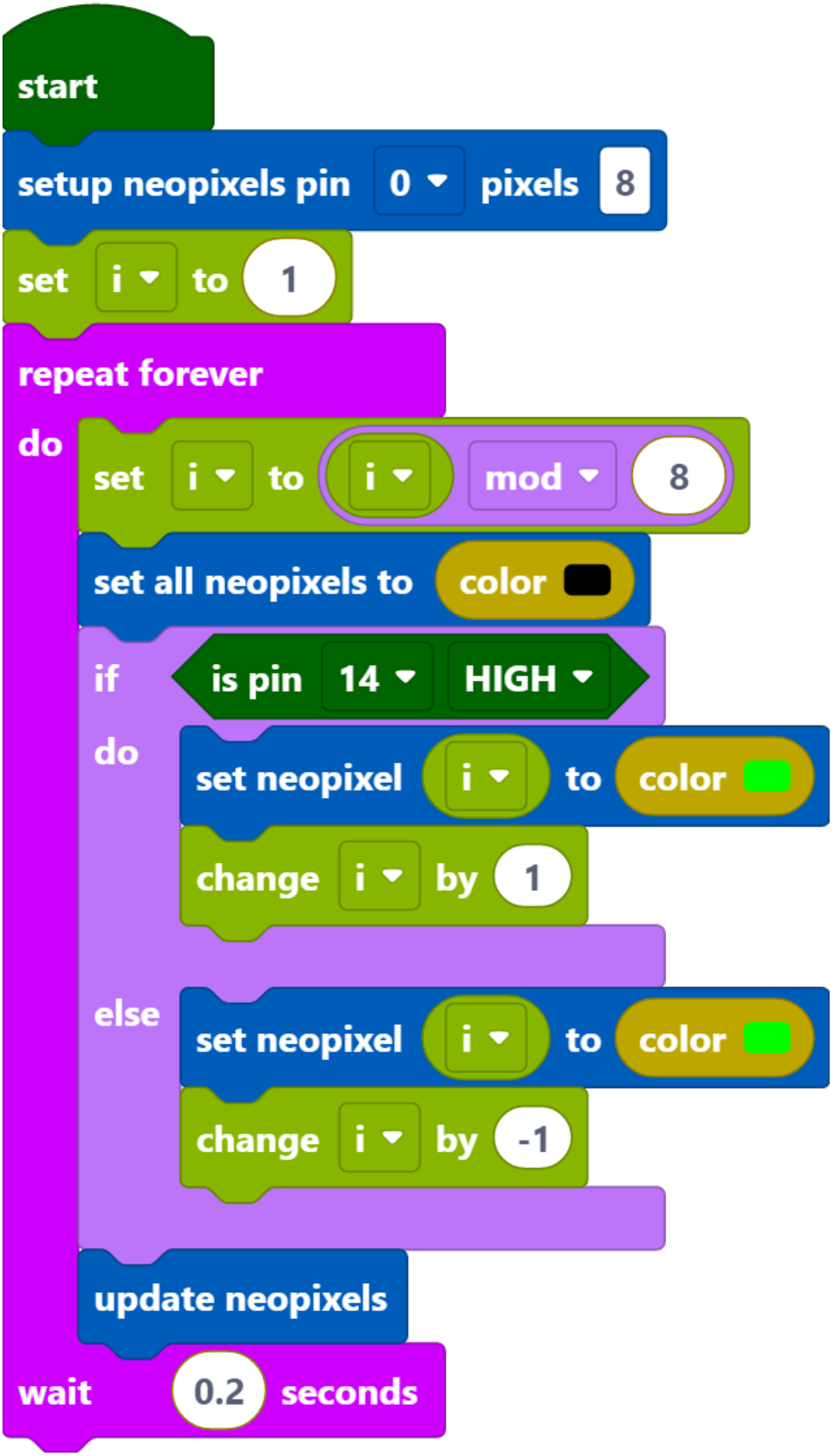
Wiring



Code

Note:

- You can refer to the image below to write code by dragging and dropping.
- Import `2.10_flowing_led.png` from the path of `kepler-kit-main\piper`. For detailed tutorials, please refer to *Import the Code*.

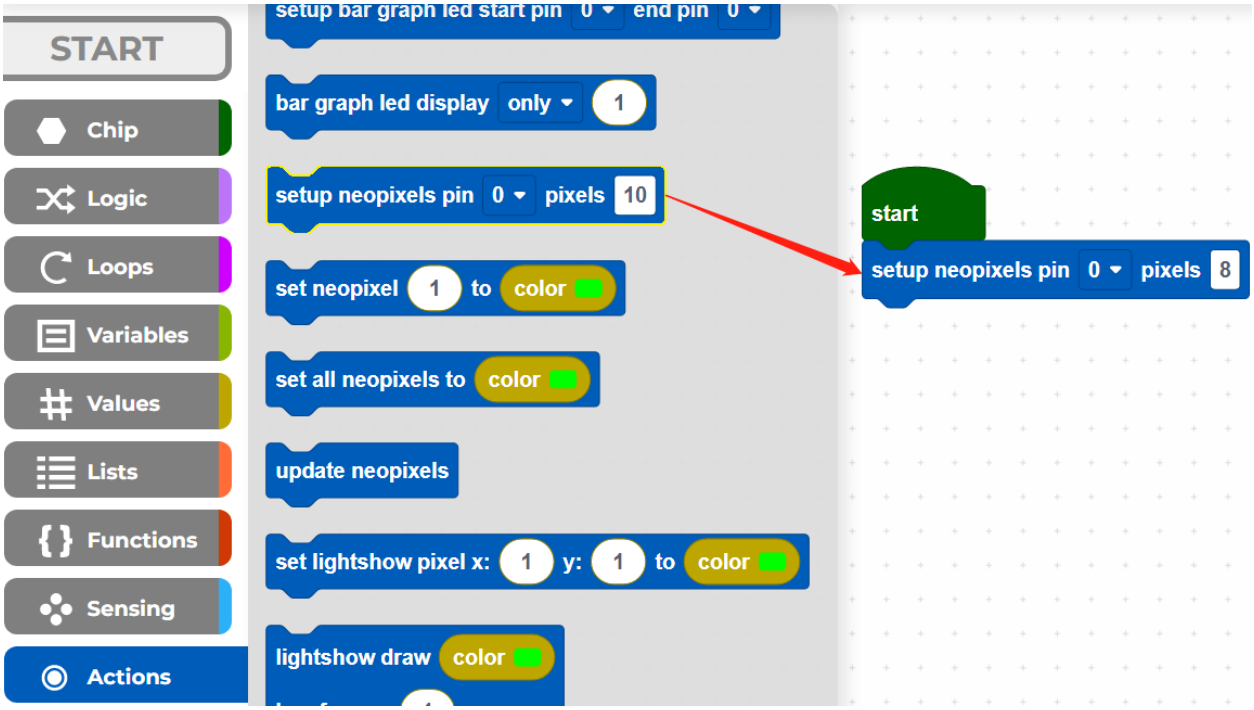

start
setup neopixels pin 0 pixels 8
set i to 1
repeat forever
do
set i to i mod 8
set all neopixels to color
if is pin 14 HIGH
do
set neopixel i to color
change i by 1
else
set neopixel i to color
change i by -1
update neopixels
wait 0.2 seconds

After connecting Pico W, click the **Start** button and the code starts to run.

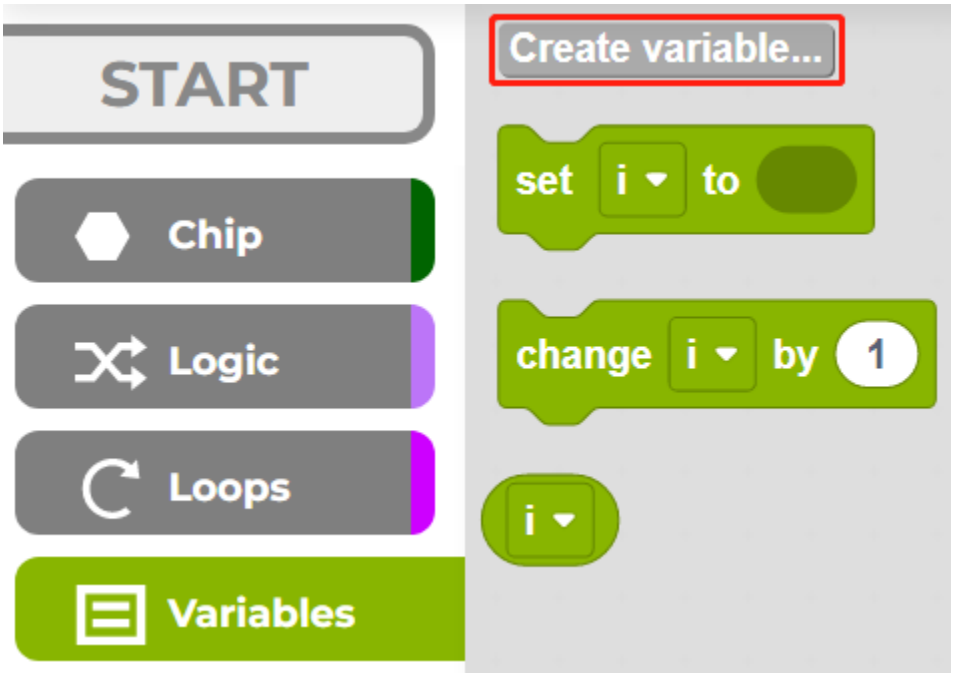
When the tilt switch is placed vertically, it makes the LEDs on the WS2812 RGB LED Strip light up one by one in green, and when the tilt switch is placed horizontally, the LEDs light up one by one in the opposite direction in green.

Programming

Step 1: Use the [setup neopixel pin() pixels()] block in the **Actions** palette to initialize the WS2812 RGB LED Strip. **0** means the connected pin is GP0 and **8** means there are 8 RGB LEDs on the WS2812 RGB LED Strip.

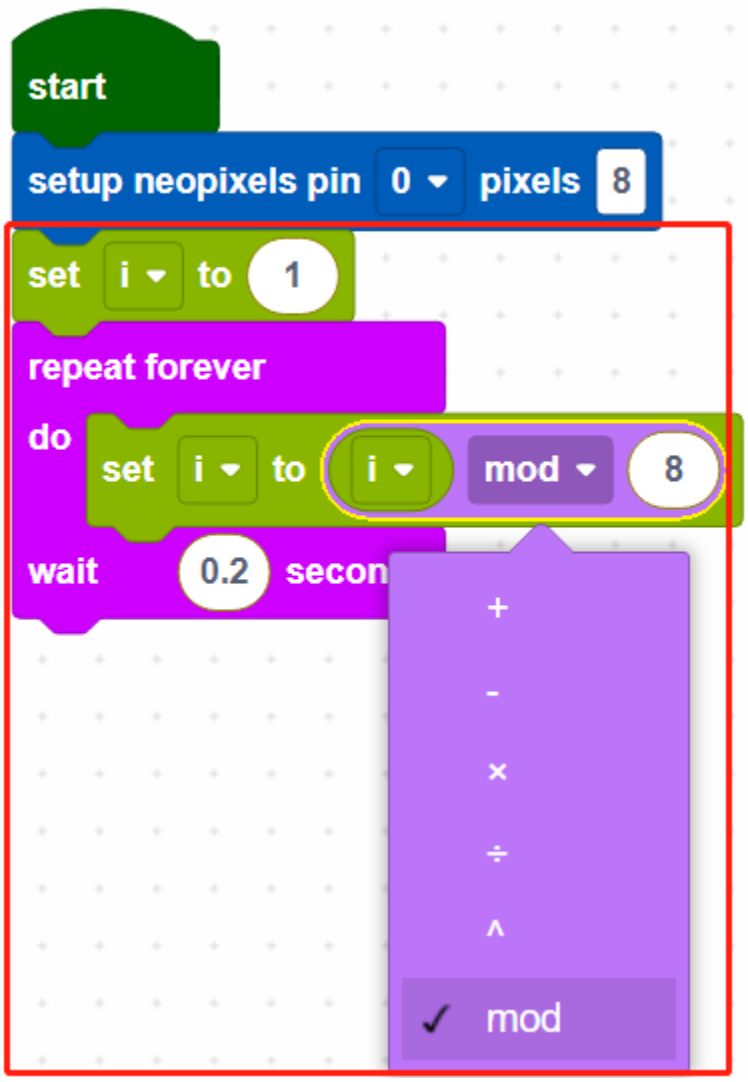


Step 2: In the **Variables** palette, click the **Create variable** button to create a variable called **i** to represent the LEDs on the WS2812 RGB LED Strip.

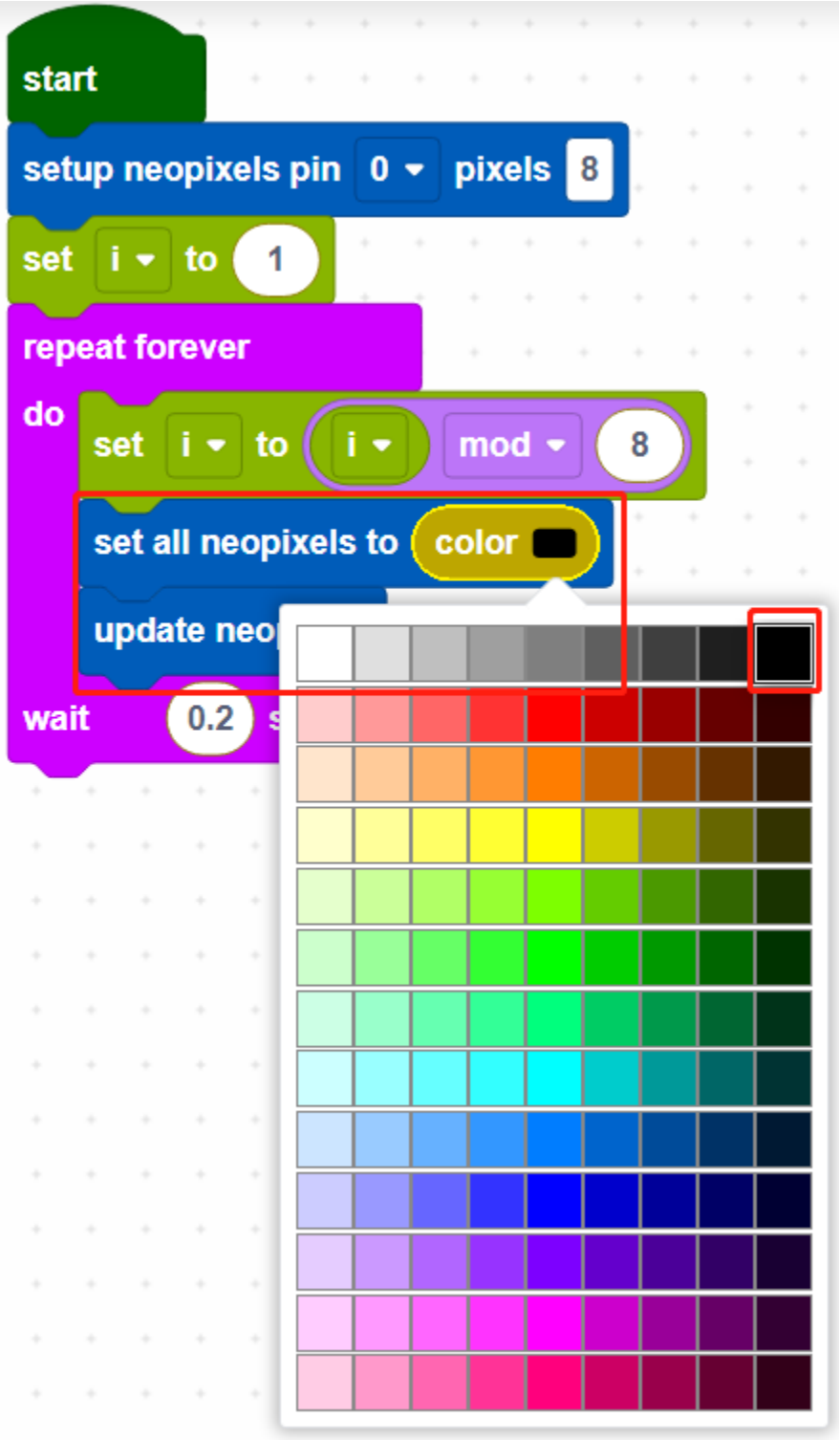


Step 3: Set the initial value of variable **i** to 1 (the LED near the wires), then in [repeat forever] block, use [() mod ()] to set the value of i from 0 to 7. e.g. 1 mod 8 = 1… 8 mod 8 =0, 9 mod 8 =1, etc.

- [() mod ()]: This is the modulo operator block, from the **Loops** palette, drop down [() = ()] to select **mod**.

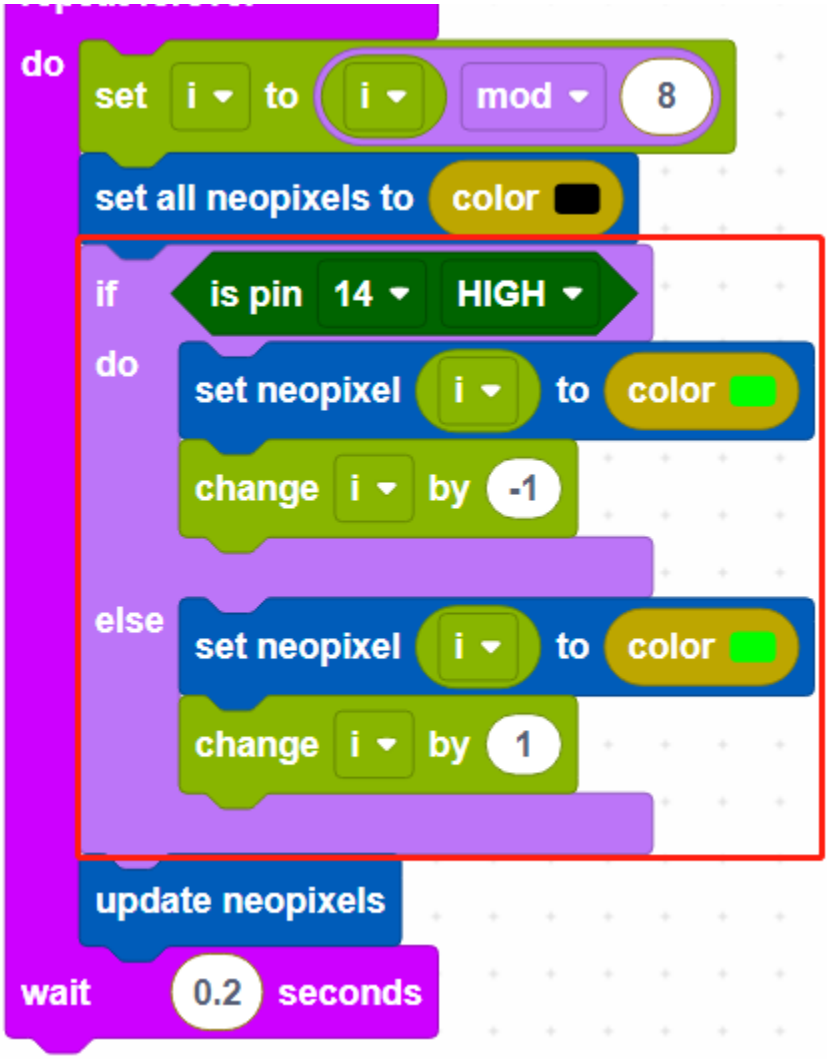


Step 4: Set all neopixels to black to make all LEDs go off, then use [updates neopixels] to make this effect update to the WS2812 RGB LED Strip.



- [set all neopixels to ()]: Use to set a color for all LEDs, there are 13*9 colors, the top right color is black to make LEDs to go off.
- [updates neopixels]: Update the effect to the WS2812 RGB LED Strip.

Step 5: If pin14 is read high, let the LEDs on the WS2812 RGB LED Strip light up one by one in green, otherwise light up green one by one in the opposite direction.



- [change () by ()]: Used to increase (positive) or decrease (negative) the value of a variable by a specific step.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

7.14 2.11 Reversing System

For this project we used an ultrasonic module and an active buzzer to create a reversing alarm system. The ultrasonic module is used to detect the distance, and the buzzer emits different frequencies of alarm sounds depending on the distance.

Required Components

In this project, we need the following components.

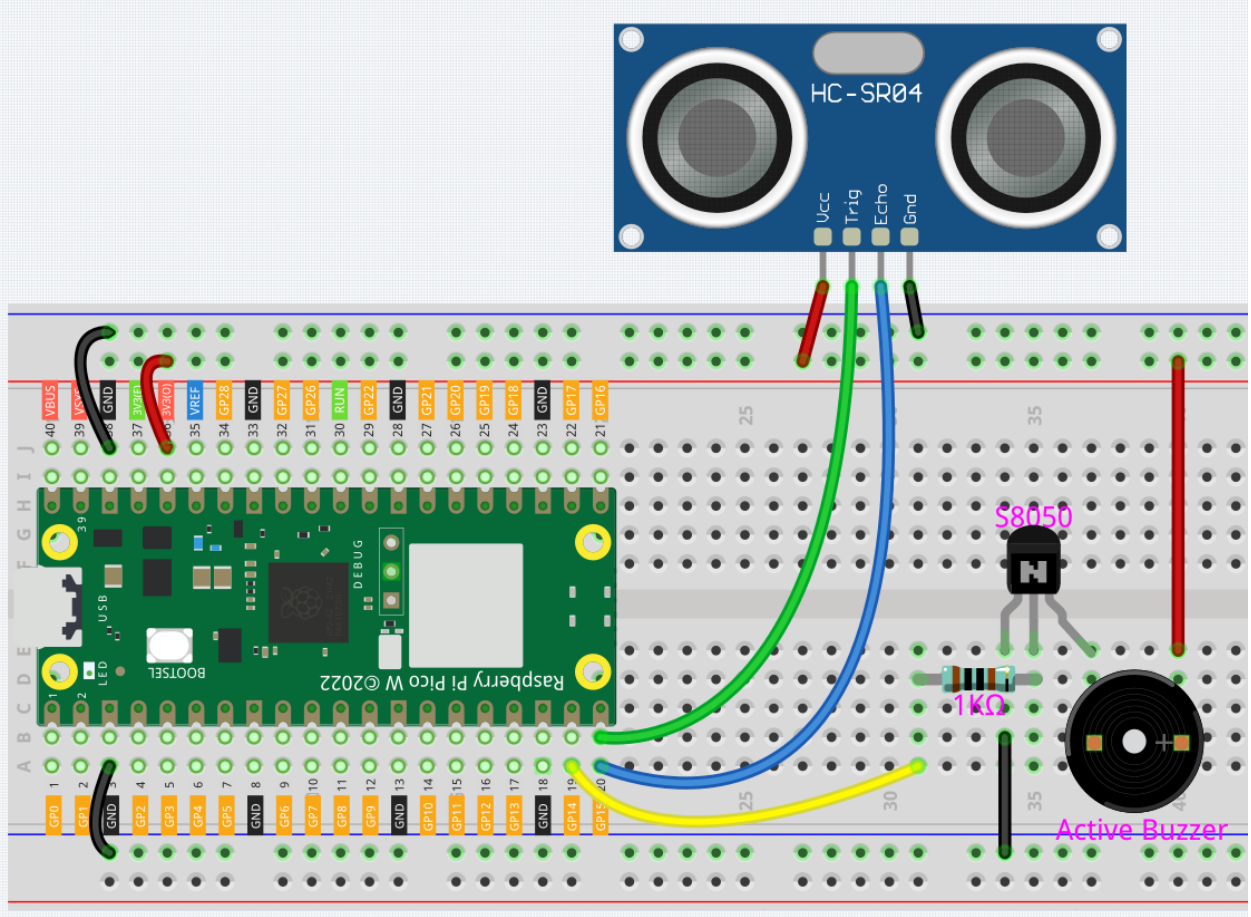
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Transistor*	1(S8050)	
6	*Resistor*	1(1K)	
7	Active *Buzzer*	1	
8	*Ultrasonic Module*	1	

Wiring

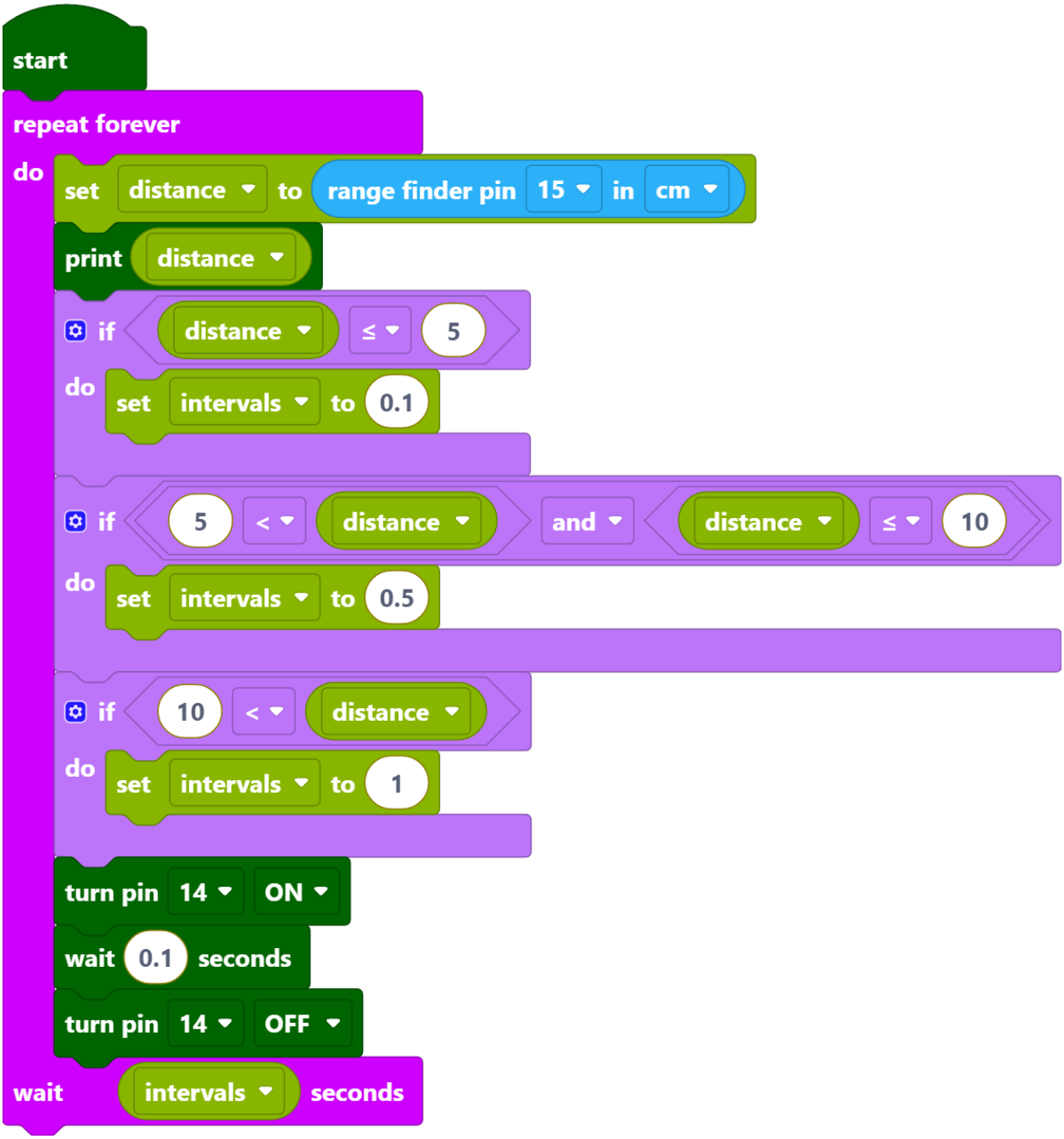


- The Echo and Trig pins of the ultrasonic module are connected to GP15 at the same time, so that the ultrasonic module sends and receives signals from GP15.
- The middle pin of the transistor connected to the buzzer is connected to GP14 through a 1k resistor.

Code

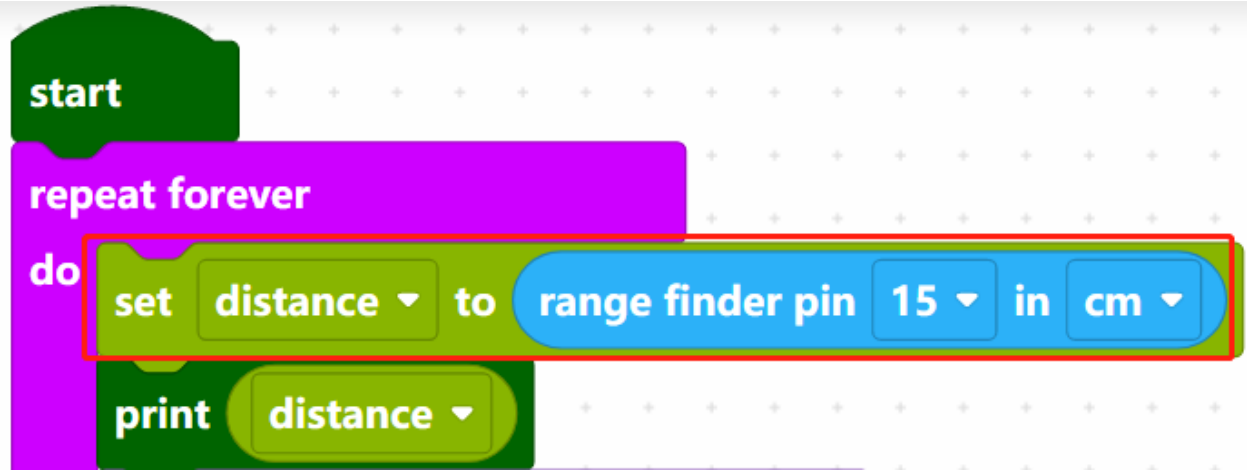
Note:

- You can refer to the image below to write code by dragging and dropping.
- Import `2.11_reversing_system.png` from the path of `kepler-kit-main\piper`. For detailed tutorials, please refer to *Import the Code*.

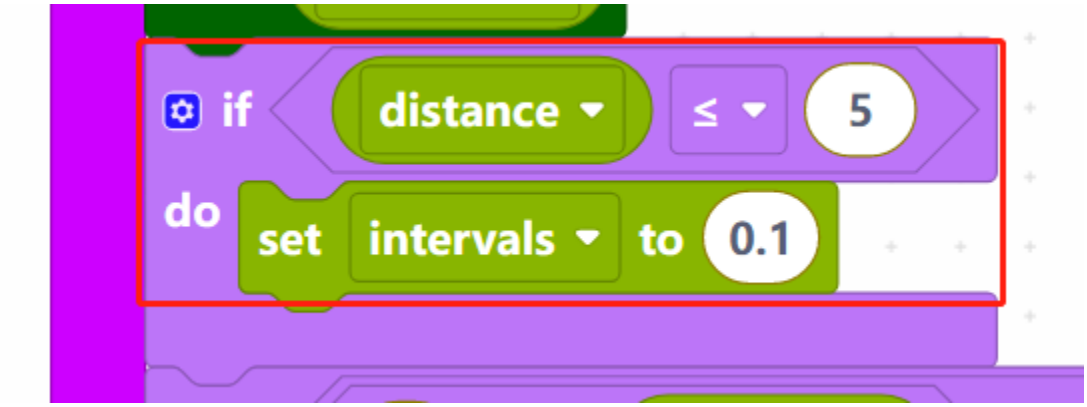


- After connecting Pico W, click the **Start** button and the code starts to run.
- When the ultrasonic detection distance is less than 5cm, the buzzer emits a sharp sound (0.1s).
- When the detection distance is between 5~10cm, the buzzer emits a slightly slow sound (0.5s).
- When the detection distance is greater than 10cm, there is a sound prompt every 1 second.

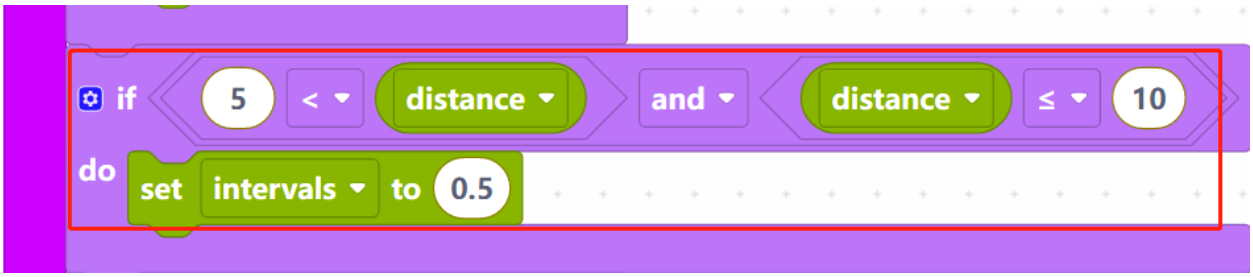
How it Works?



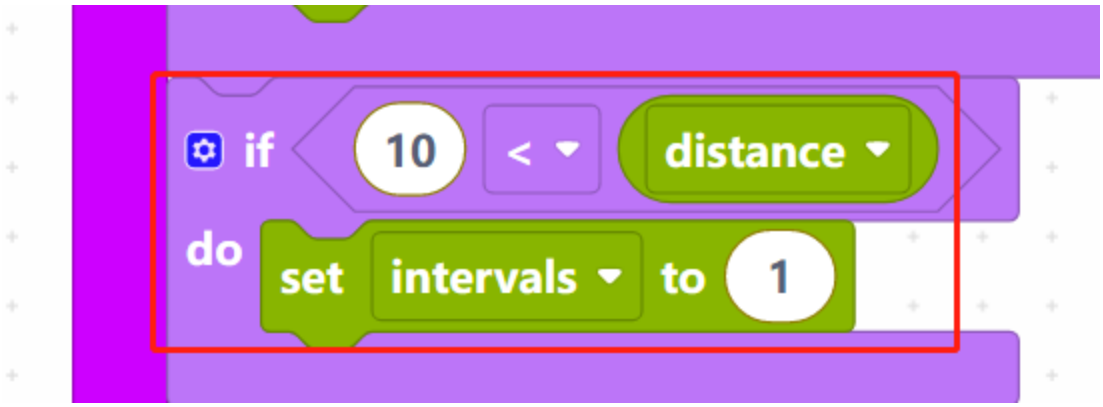
Read the distance (in cm) of the ultrasonic detection, and store it in the variable [distance].



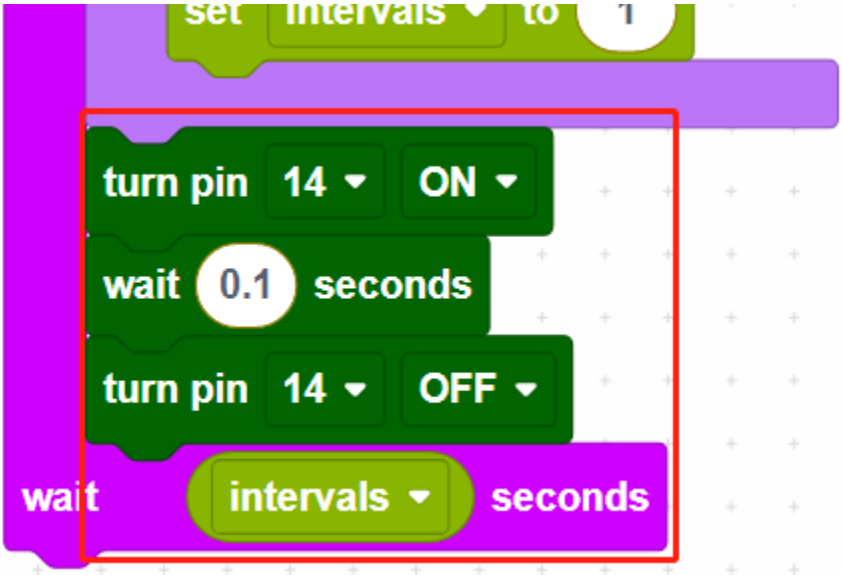
When distance is less than or equal to 5, set the variable intervals to 0.1s. The variable [intervals] is the interval between buzzer sounds.



When distance is greater than 5 and less than or equal to 10, set the [intervals] to 0.5s.



When distance is greater than 10, set the [intervals] time to 1s.



Finally, make the buzzer sound every [intervals] seconds.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

7.15 2.12 Smart Fan

Here we make a temperature controlled smart fan with thermistor, TA6586, motor, and power module. The fan rotates automatically when the set temperature is reached.

In this project, we need the following components.

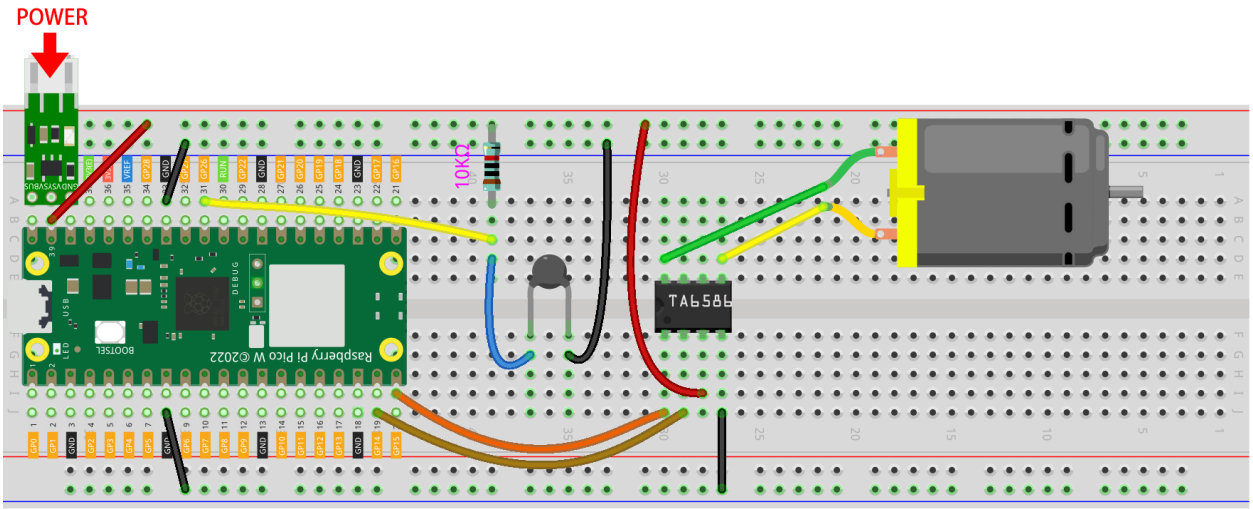
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUAN-TITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*TA6586 - Motor Driver Chip*	1	
6	*DC Motor*	1	
7	*Li-po Charger Module*	1	
8	18650 Battery	1	
9	Battery Holder	1	
10	*Resistor*	1(10K)	
11	*Photoresistor*	1	

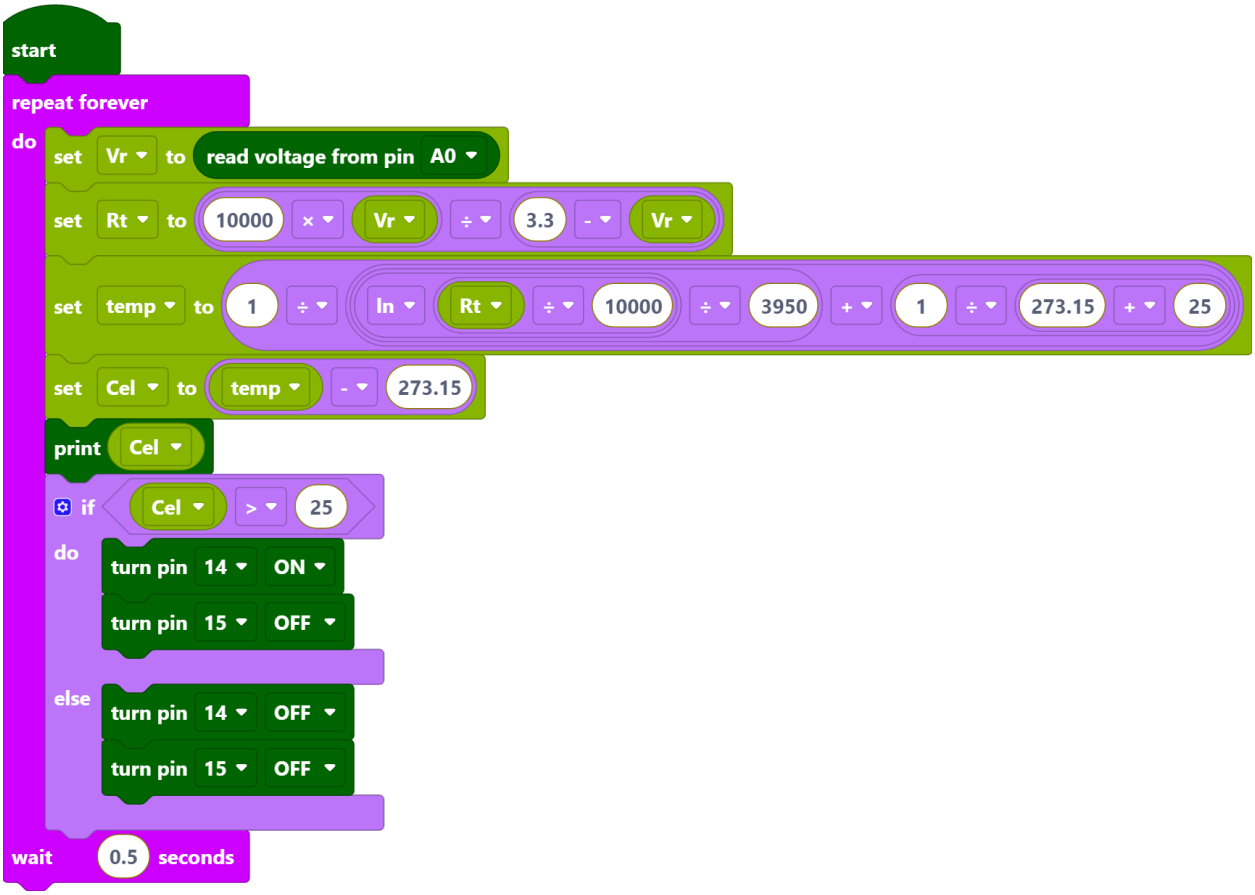
Wiring



Code

Note:

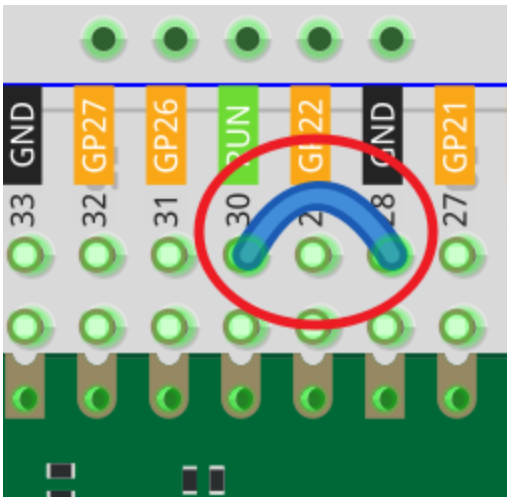
- You can refer to the image below to write code by dragging and dropping.
- Import `2.12_smart_fan.png.png` from the path of `kepler-kit-main\piper`. For detailed tutorials, please refer to *Import the Code*.



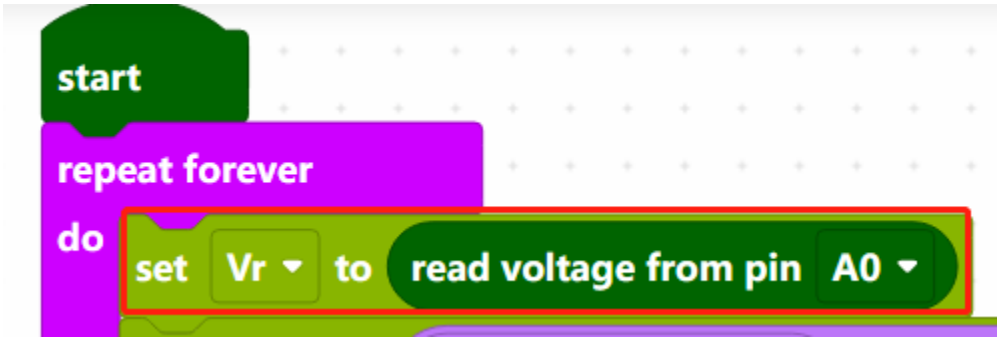
- After connecting Pico W, click the **Start** button and the code starts to run.
- Click CONSLE, you will see the current temperature in Celsius.
- The fan starts spinning when it is above 25 degrees, and stops spinning when it is below 25 degrees.

Note:

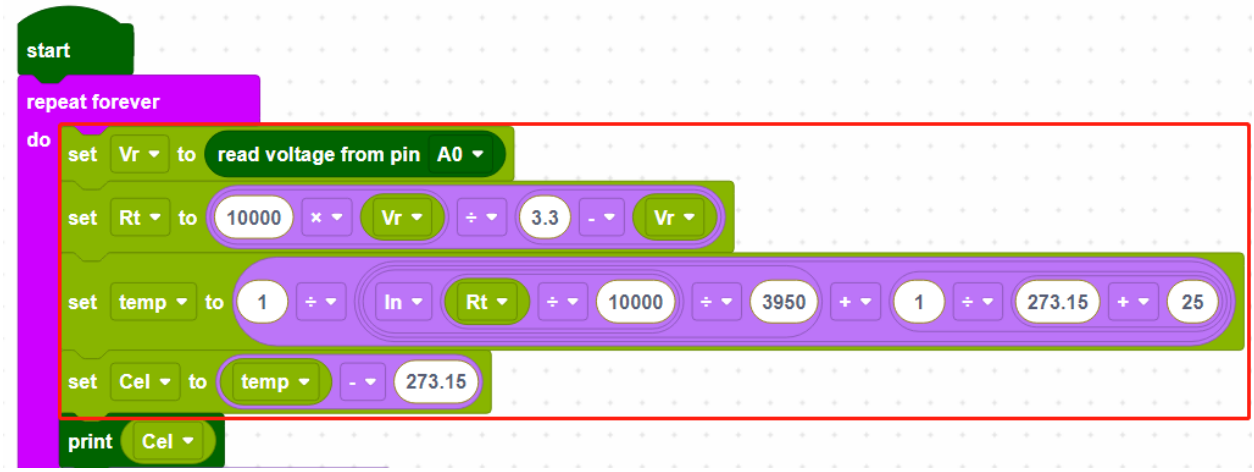
- If the motor is still spinning after you click the Stop button, you need to reset the Run pin on the Pico W with a wire to GND at this time, and then unplug this wire to run the code again.
- This is because the motor is operating with too much current, which may cause the Pico W to disconnect from the computer.



How it Works?



The voltage of A0 (GP26) is read and assigned to the variable [Vr].



These calculations convert the thermistor values into centigrade degree.

Note: Here is the relation between the resistance and temperature:

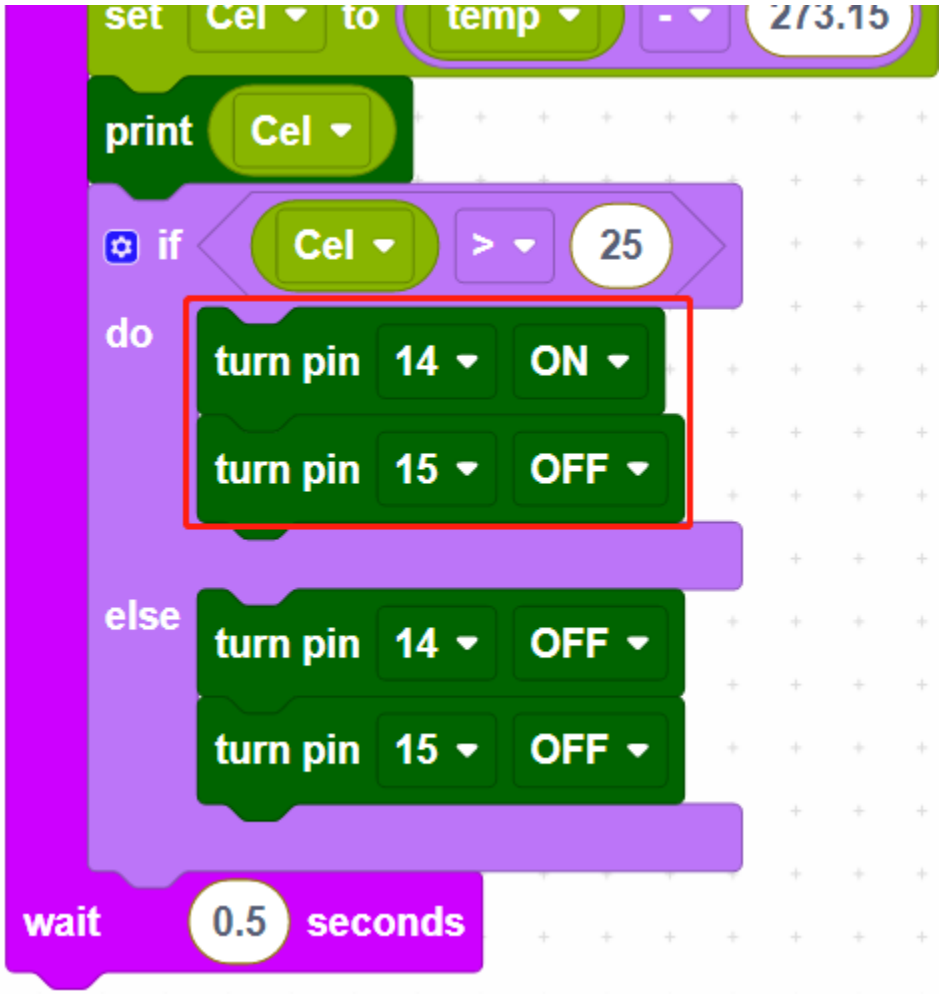
RT =RN expB(1/TK – 1/TN)

- RT is the resistance of the NTC thermistor when the temperature is TK.

- RN is the resistance of the NTC thermistor under the rated temperature TN. Here, the numerical value of RN is 10k.
- TK is a Kelvin temperature and the unit is K. Here, the numerical value of TK is 273.15 + degree Celsius.
- TN is a rated Kelvin temperature; the unit is K too. Here, the numerical value of TN is 273.15+25.
- And B(beta), the material constant of NTC thermistor, is also called heat sensitivity index with a numerical value 3950.
- exp is the abbreviation of exponential, and the base number e is a natural number and equals 2.7 approximately.

Convert this formula **TK=1/(ln(RT/RN)/B+1/TN)** to get Kelvin temperature that minus 273.15 equals degree Celsius.

This relation is an empirical formula. It is accurate only when the temperature and resistance are within the effective range.



When the temperature is higher than 25 degrees Celsius, set GP14 to ON and GP15 to OFF to make the motor rotate, or you can reverse their levels. When the temperature is lower than 25 degrees Celsius, set GP14 and GP15 both low to stop the motor.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

7.16 2.13 Reaction Game

Here we use several buttons, buzzer and LEDs to make a reaction game.Press the referee button to start the game, the buzzer will also keep ringing to prompt the game to continue; press the 2 player buttons quickly respectively; when the referee button is pressed again, the game is over the buzzer stops working. At this point, look at the CONSOLE on Piper Make to see whose hand speed is fast.

Required Components

In this project, we need the following components.

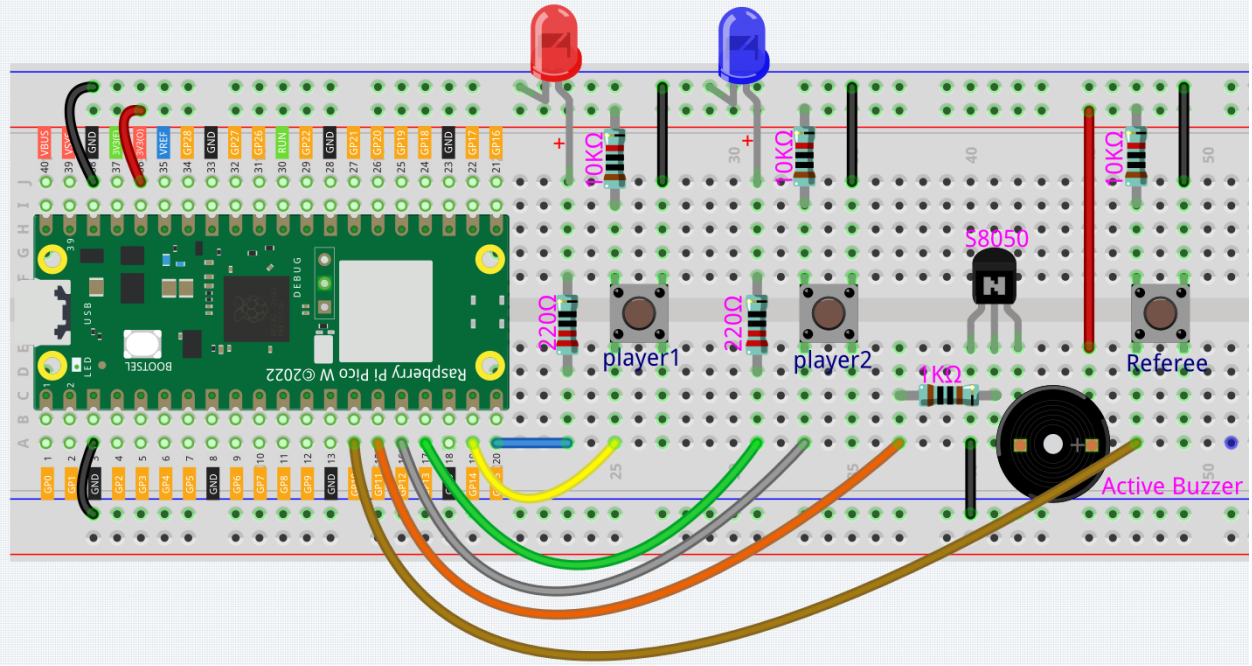
It's definitely convenient to buy a whole kit, here's the link:

Name	ITEMS IN THIS KIT	LINK
Kepler Kit	450+	

You can also buy them separately from the links below.

SN	COMPONENT	QUANTITY	LINK
1	*Raspberry Pi Pico W*	1	
2	Micro USB Cable	1	
3	*Breadboard*	1	
4	*Jumper Wires*	Several	
5	*Transistor*	1(S8050)	
6	*Resistor*	6(2-220, 1-1K, 3-10K)	
7	Active *Buzzer*	1	
8	*Button*	3	
9	*LED*	2	

Wiring

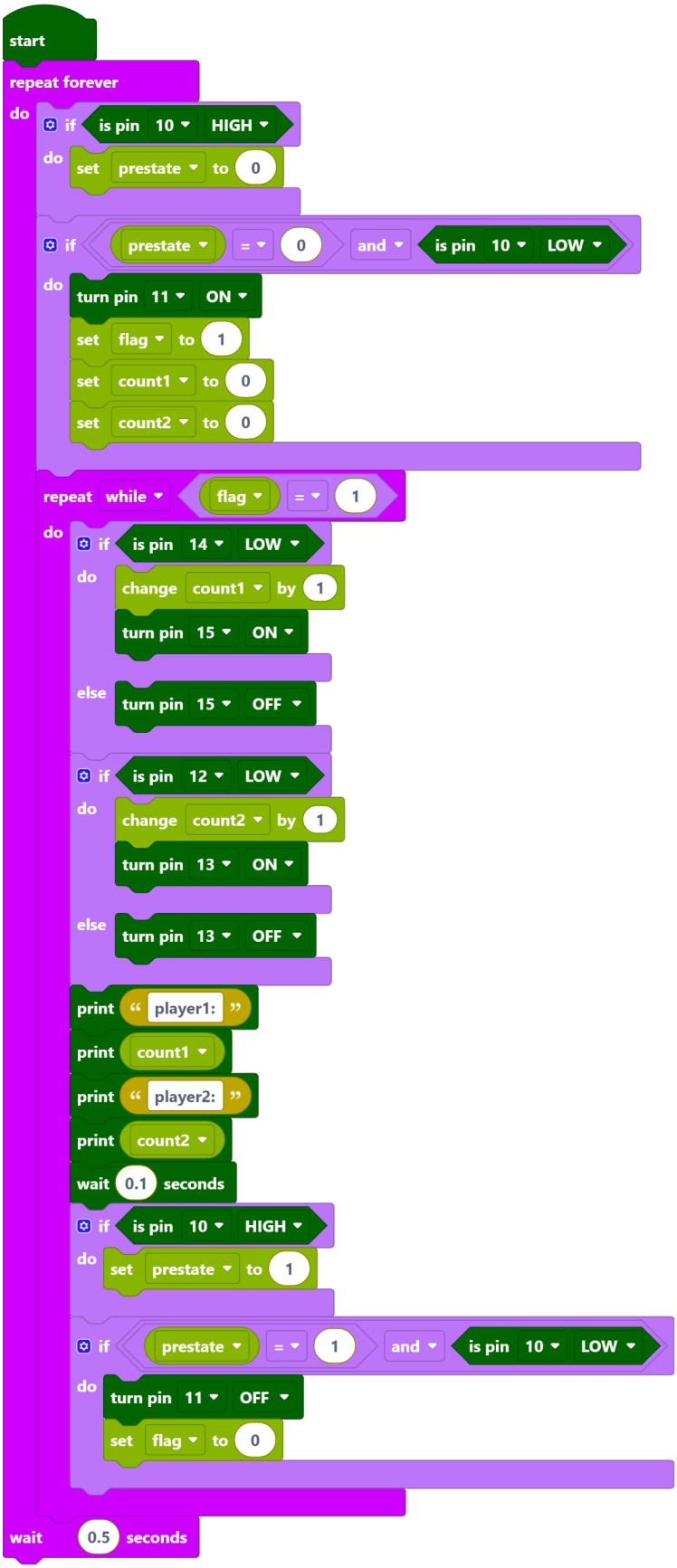


- Define 2 buttons as player1 (GP14) and player2 (GP12), both have a pull-up resistor connected. When the buttons are pressed, GP14 and GP12 are each low.
- Their indicators are connected to GP15 and GP13, and output high to light them up.
- Define a referee button connected to GP10, when the button is pressed, GP10 goes low.
- Active buzzer is connected to GP11, when GP11 output high, the buzzer will sound.

Code

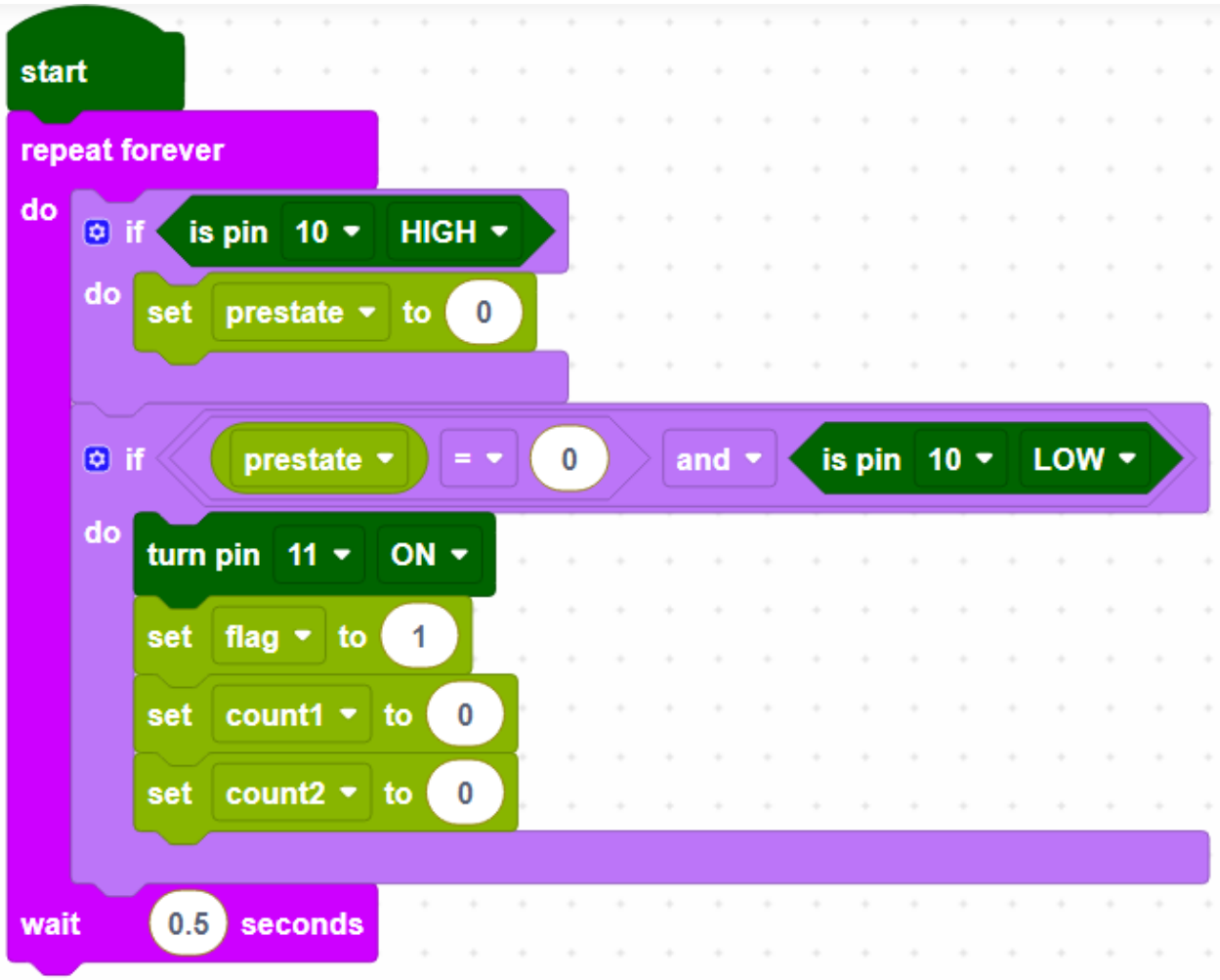
Note:

- You can refer to the image below to write code by dragging and dropping.
- Import `2.13_reaction_game.png` from the path of `kepler-kit-main\piper`. For detailed tutorials, please refer to *Import the Code*.

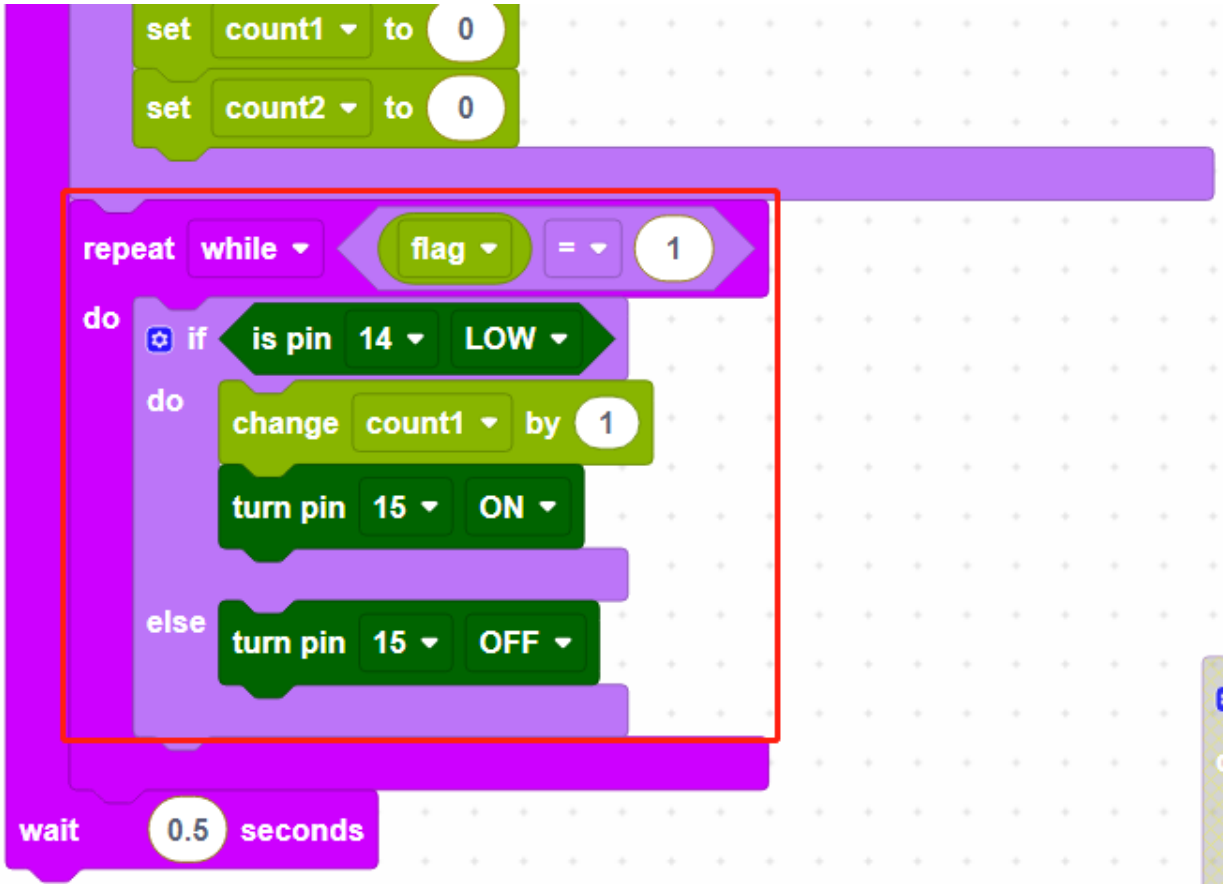

start
repeat forever
do
if is pin 10 HIGH
do
set prestate to 0
if prestate = 0 and is pin 10 LOW
do
turn pin 11 ON
set flag to 1
set count1 to 0
set count2 to 0
repeat while flag = 1
do
if is pin 14 LOW
do
change count1 by 1
turn pin 15 ON
else
turn pin 15 OFF
if is pin 12 LOW
do
change count2 by 1
turn 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

- After connecting Pico W, click the **Start** button and the code starts to run.
- Press the referee button and the buzzer starts to sound (continuously), representing the start of the game.
- At this point, press the **player** buttons separately and quickly, and the corresponding LEDs will light up.
- When the **referee** button is pressed again, the buzzer stops, which means the game is over.
- Click CONSOLE at this point to see who presses it more times.

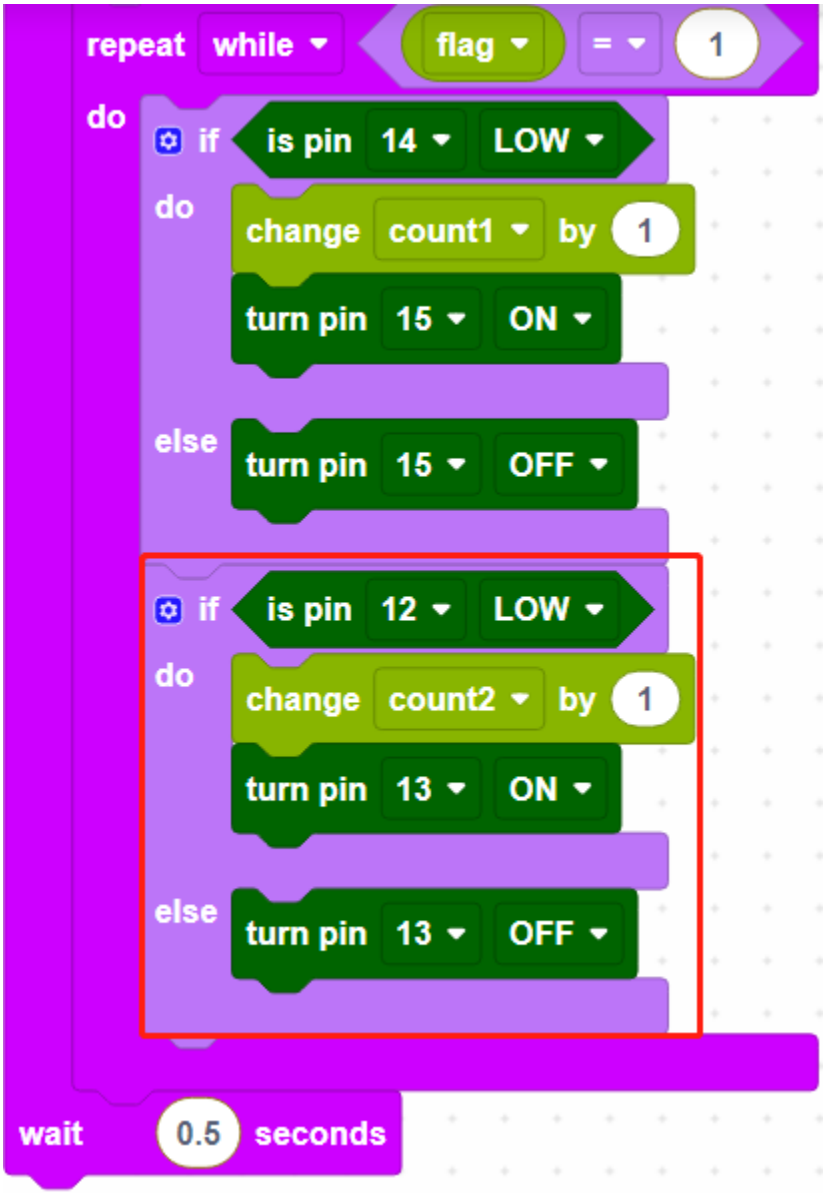
How it Works?



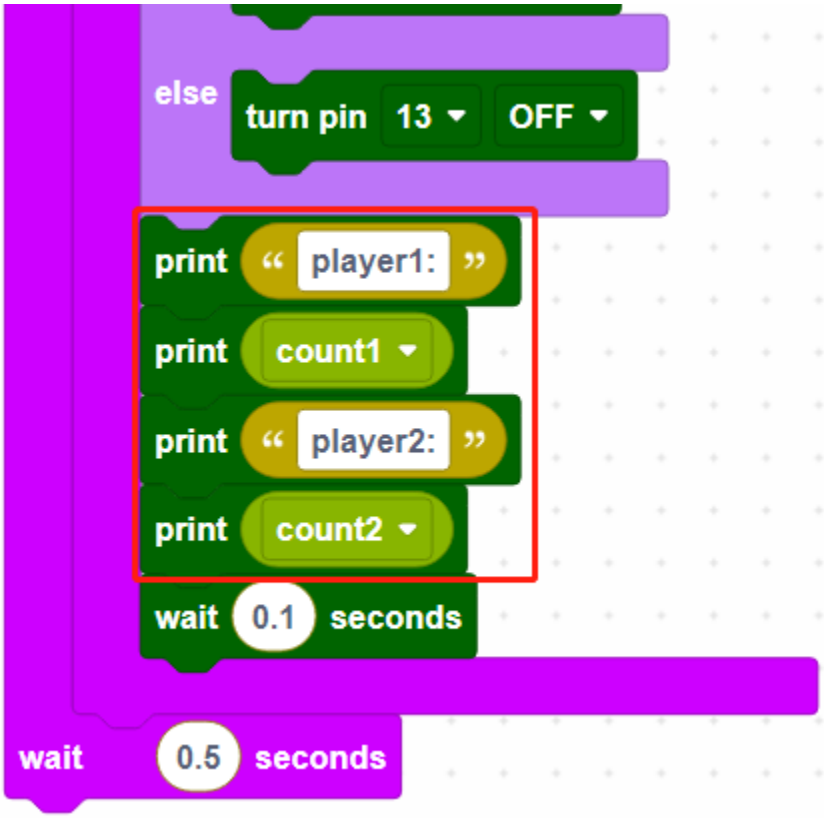
GP10 is the referee button, when the referee button is not pressed it is high, the game has not started yet. When GP10 is low (referee button pressed), the game starts; set GP11 to high (buzzer), create variables and set initial values.



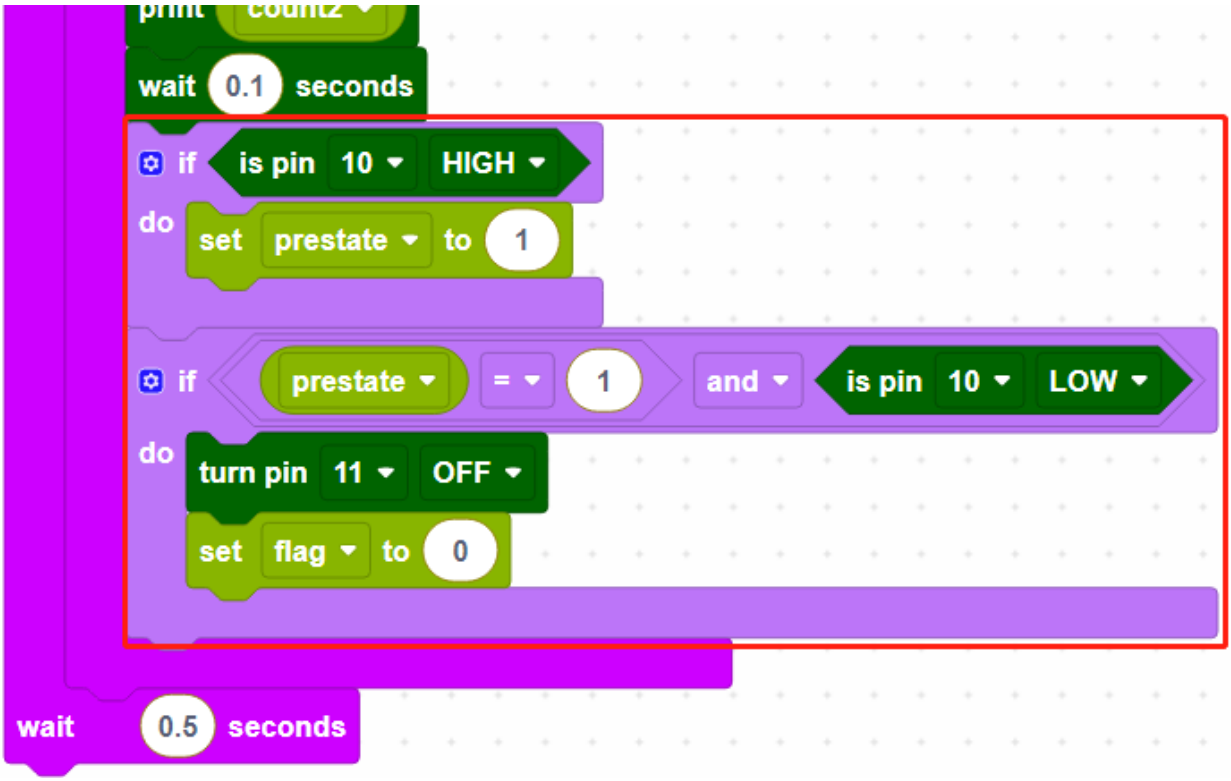
When flag equals to 1 for game start, then read the value of GP14 (player1); if player1 button is pressed, store the number of presses in variable [count1], and make the indicator of GP15 light up.



Read the number of presses of GP12(player2) in the same way.



Print the number of presses for player1 and player2 respectively.



When the referee button is pressed again, the buzzer stops working and the game ends.

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

CHAPTER

EIGHT

VIDEO COURSES

If you're finding the content in the online documentation a bit challenging to grasp, don't worry. A step-by-step video course is here to make your learning journey smoother and more engaging.

project

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.1 lesson 1: Write Your First Program for Absolute Beginners

This tutorial covers setting up and programming the Raspberry Pi Pico W using the SunFounder Kepler Kit:

- **Introduction and Setup**: Overview of the tutorial series and sponsor shoutout to SunFounder. Instructions on unboxing and connecting the Pico W to the PC using a USB cable.
- **MicroPython Installation**: Step-by-step guide to downloading and installing MicroPython on the Pico W, including necessary steps to make the Pico W recognizable as a USB drive.
- **Thonny IDE Installation**: Instructions for downloading and installing the Thonny IDE on the PC, including handling potential installation warnings and setting up Thonny to communicate with the Pico W.
- **First Program**: Writing and running the first program in Thonny to print "Hello, World!" to confirm the setup is working correctly.
- **GPIO Interaction**: Explanation of GPIO pins and a simple program to control the onboard LED by turning it on and off. Introduction to loops and delays for creating a blinking LED effect.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.2 lesson 2: Understanding and Using Breadboards

This tutorial covers using a breadboard and controlling an external LED with the Raspberry Pi Pico W from the SunFounder kit:

- **Introduction**: Overview of the tutorial series and sponsor shoutout to SunFounder.
- **Review of Previous Lesson**: Recap of installing MicroPython, setting up Thonny, and writing basic programs to control GPIO pins.
- **Breadboard Basics**: Detailed explanation of how a breadboard works, including understanding rows, columns, and connections.
- **Building an LED Circuit**: Step-by-step guide to build an external LED circuit using the breadboard, GPIO pins, and a 220 Ohm resistor.
- **Programming the LED**: Writing a Python program in Thonny to control the external LED, including turning it on, off, and making it blink.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.3 lesson 3: Understanding and Using Binary Numbers

This tutorial covers the basics of binary numbers and how they are used in computing, specifically focusing on creating a binary counter using LEDs with the Raspberry Pi Pico W.

- **Introduction**: Overview of the tutorial series and sponsor shoutout to SunFounder.
- **Importance of Binary Numbers**: Explanation of why binary numbers are fundamental in computing, highlighting the on-off nature of digital switches.
- **Counting in Binary**: Detailed explanation of how to count in binary using only two symbols: 0 and 1. Demonstrates the concept of place values and how binary numbers represent decimal numbers.
- **Binary Arithmetic**: Shows how to perform addition, subtraction, multiplication, and division using binary numbers. Validates the correctness of binary arithmetic by converting results back to decimal.
- **Binary Representation of Data**: Discusses how binary numbers are used to represent colors, pictures, and videos in computers. Introduces the concept of grayscale and RGB color models.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.4 lesson 4: Create a Binary Counter Using the Pico W

This tutorial covers creating a 4-bit digital counter using the SunFounder Kepler Kit for Raspberry Pi Pico W:

- **Introduction**: Introduction to the tutorial series and sponsor shoutout to SunFounder. Instructions on unboxing and connecting the Pico W to the PC.
- **Homework Review**: Review of the previous homework assignment to create a 4-bit digital counter using four LEDs.
- **Schematic and Circuit Setup**: Detailed schematic explanation and physical setup for connecting four LEDs with 220 Ohm resistors to the Pico W. Instructions on establishing a ground rail and connecting GPIO pins to the LEDs.
- **Code Explanation**: Writing and explaining Python code to control the LEDs as a 4-bit counter. Describes creating a loop to count from 0 to 15 in binary, turning LEDs on and off accordingly.
- **Demonstration**: Demonstrating the working 4-bit digital counter and verifying the circuit and code functionality.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.5 lesson 5: Reading Analog Voltages Using a Potentiometer

This tutorial covers reading analog voltages using the SunFounder Kepler Kit for Raspberry Pi Pico W:

- **Analog Voltage Reading**: Explains the importance of reading analog voltages for various sensor inputs and user inputs, like potentiometers, to control aspects such as volume or brightness.
- **Wiring Diagram and Setup**: Provides a detailed explanation of the potentiometer's workings and how to connect it to the Pico W. Describes setting up ground and 3.3V rails and connecting the potentiometer's middle pin to GPIO pin 28.
- **Code Explanation**: Describes writing Python code to read analog voltages. Covers importing necessary libraries, setting up GPIO pins, creating an object for the potentiometer, and using a while loop to continuously read and print voltage values.
- **Mathematical Conversion**: Teaches how to convert the raw ADC values (0 to 65535) to actual voltage values (0 to 3.3V) using linear equations. Demonstrates the process of deriving the equation of the line from two known points and applying it in the code.
- **Practical Demonstration**: Shows the code in action, reading and converting the potentiometer values to voltage and displaying them. Discusses the accuracy and expected results when adjusting the potentiometer.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.6 lesson 6: Understanding If Statements in MicroPython

This tutorial covers using if statements and conditionals in MicroPython with the Raspberry Pi Pico W:

- **User Input Handling**: Demonstrates how to get input from the user using the input() function and convert it into different data types, such as integers and floating-point numbers.
- **Basic If Statements**: Explains how to use if statements to execute code based on certain conditions, including checking for equality (==), inequality (!=), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=).
- **Conditional Logic**: Illustrates how to implement logic based on user input to control the state of an LED (on, off, toggle) using if, elif, and else statements.
- **Practical Demonstration**: Provides a detailed walkthrough of setting up an LED circuit on a breadboard and controlling the LED based on user input through conditional statements in MicroPython.
- **Homework Assignment**: Assigns a project to build a circuit with a potentiometer and three LEDs (red, yellow, green). The task involves converting potentiometer readings to a 0-100 scale and lighting up the LEDs based on the input range:

Green LED for 0-79 Yellow LED for 80-94 Red LED for 95-100 Encourages posting the project on YouTube and engaging with others to develop a sense of community.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.7 lesson 7: Controlling 3 LED with a Potentiometer in Micropython

This tutorial covers using a potentiometer to control three LEDs (green, yellow, and red) with the Raspberry Pi Pico W:

- **Homework Solution Review**: Recaps the previous assignment to connect a potentiometer and three LEDs, and map the potentiometer readings from 0 to 100 to control the LEDs based on the input values.
- **Circuit Setup**: Provides a detailed wiring diagram and setup for connecting the potentiometer and LEDs to the Raspberry Pi Pico W. Includes creating ground and 3.3V rails, and connecting the components to the appropriate GPIO pins.
- **Reading and Mapping Potentiometer Values**: Demonstrates how to read analog values from the potentiometer and map them from their original range (432 to 65,535) to a 0 to 100 scale using mathematical equations.

- **Conditional Logic for LED Control**: Implements if statements to control the LEDs based on the potentiometer readings:Green LED for values between 0 and 79.Yellow LED for values between 80 and 94.Red LED for values between 95 and 100.
- **Practical Demonstration**: Shows the working circuit and code in action, with the LEDs lighting up based on the potentiometer's position.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.8 lesson 8: Compound Conditionals and If Statements in MicroPython

This tutorial covers using a potentiometer to control three LEDs (green, yellow, and red) with the Raspberry Pi Pico W and identifies a critical logical error in the previous implementation:

- **Homework Solution Review**: Recaps the previous assignment to find the logical flaw in the code that controlled the LEDs based on potentiometer readings.
- **Logical Error Explanation**: Explains the logical error in the code where the LED states could overlap, causing unintended behavior. Emphasizes the importance of ensuring that logical conditions are mutually exclusive to avoid such errors.
- **Real-World Application Example**: Illustrates the potential dangers of logical errors using a hypothetical scenario of UV sterilization rooms, where incorrect LED control could cause harm.
- **Compound Conditionals Introduction**: Introduces compound conditionals using logical operators (AND, OR) to create precise conditions for controlling LEDs. Demonstrates how to ensure that each condition is correctly implemented to avoid overlap.
- **Circuit Setup and Code Explanation**: Provides a detailed wiring diagram and setup for connecting the potentiometer and LEDs to the Raspberry Pi Pico W. Discusses the correct way to write conditional statements to control the LEDs based on potentiometer readings.
- **Practical Demonstration**: Shows the revised code in action, highlighting the correct LED behavior with no overlaps or missed conditions. Emphasizes the importance of careful logical planning in coding projects.
- **Next Steps**: Previews the next lesson on simulating analog output, which will further expand the toolkit for working with the Raspberry Pi Pico W.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.9 lesson 9: Getting Analog Output Using PWM (Pulse Width Modulation)

This tutorial covers generating analog output using PWM (Pulse Width Modulation) on the Raspberry Pi Pico W:

- **Analog Output Explanation**: Discusses the concept of PWM and how it simulates analog output by switching between 3.3V and 0V at different duty cycles. Explains the relationship between voltage, time, and PWM duty cycle.
- **GPIO Pin Configuration**: Explains which GPIO pins can generate analog output and the importance of not using pins with the same PWM label simultaneously.
- **Circuit Setup and Code**: Provides a detailed wiring diagram and setup for connecting an LED to the Raspberry Pi Pico W. Includes code to set up PWM on a GPIO pin, take user input for desired voltage, and convert it to a corresponding PWM duty cycle.
- **Practical Demonstration**: Shows how to measure and visualize the PWM signal using an oscilloscope, illustrating how different duty cycles correspond to different average voltages.
- **Homework Assignment**: Asks users to create a circuit where they can input a desired voltage to control the brightness of an LED, reinforcing the concepts of PWM and analog output.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.10 lesson 10: Controlling a Dimmable LED with a Potentiometer

This tutorial covers controlling LED brightness using PWM (Pulse Width Modulation) on the Raspberry Pi Pico W:

- **LED Brightness Control**: Explains how to control LED brightness using PWM. Discusses digital signals, duty cycles, and how varying duty cycles control LED brightness.
- **Wiring Diagram and Setup**: Provides a detailed wiring diagram for connecting an LED with a 220 Ohm resistor to the Raspberry Pi Pico W. Demonstrates the physical setup on a breadboard.
- **Code Explanation**: Describes the code for setting up PWM on a GPIO pin, taking user input for the desired voltage, and converting it to the corresponding PWM duty cycle. Explains the use of libraries, PWM objects, and frequency settings.
- **Practical Demonstration**: Shows how to measure and visualize the PWM signal using an oscilloscope, illustrating how different duty cycles correspond to different average voltages.
- **Homework Assignment**: Asks users to create a circuit that reads input from a potentiometer to control the brightness of an LED, reinforcing the concepts of PWM and analog output.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.11 lesson 11: Understanding and Controlling an RGB LED in MicroPython

This tutorial covers controlling an RGB LED using the SunFounder Kepler Kit and Raspberry Pi Pico W:

- **RGB LED Control**: Explains controlling RGB LED colors using PWM (Pulse Width Modulation). Discusses the structure of the RGB LED, which has three LEDs (red, green, and blue) with a common ground. Emphasizes the importance of using separate current-limiting resistors for each color channel to prevent crosstalk.
- **Wiring Diagram and Setup**: Provides a detailed wiring diagram for connecting the RGB LED and the required resistors to the Raspberry Pi Pico W. Demonstrates the physical setup on a breadboard, including connections for red, green, and blue channels to GPIO pins 13, 14, and 15, respectively.
- **Code Explanation**: Describes the code for setting up PWM on each GPIO pin and controlling the brightness of each color channel. Covers the setup of PWM frequency, duty cycle, and the initialization of GPIO pins for red, green, and blue channels.
- **Practical Demonstration**: Shows how to run the code to change the color of the RGB LED. Demonstrates turning individual color channels on and off, and adjusting brightness levels.

- **Homework Assignment**: Asks users to create a program that prompts the user for a color (red, green, blue, cyan, magenta, yellow, orange, or white) and adjusts the RGB LED to display the specified color. Encourages users to mix and match PWM values to achieve accurate color representation.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.12 lesson 12: Create a Dimmable LED in Micropython

This tutorial covers controlling LED brightness using a potentiometer and PWM (Pulse Width Modulation) on the Raspberry Pi Pico W:

- **LED Brightness Control**: Explains controlling LED brightness using PWM. Discusses digital signals, duty cycles, and how varying duty cycles control LED brightness. Introduces the concept of exponential scaling to make the LED brightness perception more linear to the human eye.
- **Wiring Diagram and Setup**: Provides a detailed wiring diagram for connecting a potentiometer and an LED with a 220 Ohm resistor to the Raspberry Pi Pico W. Demonstrates the physical setup on a breadboard.
- **Code Explanation**: Describes the code for setting up PWM on a GPIO pin, reading the analog value from a potentiometer, and converting it to the corresponding PWM duty cycle. Explains the use of libraries, PWM objects, and frequency settings.
- **Practical Demonstration**: Shows how to measure and visualize the PWM signal using an oscilloscope, illustrating how different duty cycles correspond to different average voltages. Demonstrates the effect of using exponential scaling for smoother brightness transitions.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.13 lesson 13: User Specified RGB LED Colors Using Micropython

This tutorial covers controlling an RGB LED using the SunFounder Kepler Kit and Raspberry Pi Pico W:

- **RGB LED Control**: Demonstrates controlling an RGB LED using PWM (Pulse Width Modulation) to achieve different colors. Explains the structure of the RGB LED with its four legs for red, green, blue, and ground.
- **Wiring Diagram and Setup**: Provides a detailed wiring diagram for connecting the RGB LED to the Raspberry Pi Pico W. Each color channel (red, green, blue) is connected to a GPIO pin (13, 14, 15) with a 220 Ohm resistor for each channel to limit the current.
- **Code Explanation**: Describes the MicroPython code for setting up PWM on each GPIO pin and controlling the brightness of each color channel. Includes code for turning on specific colors (red, green, blue, cyan, magenta, yellow, orange, white) based on user input.
- **Practical Demonstration**: Shows how to run the code and change the color of the RGB LED. Demonstrates the process of inputting a color and observing the RGB LED change accordingly.
- **Homework Assignment**: Asks users to extend the project by connecting three potentiometers to control the RGB LED colors manually. Encourages users to mix and match potentiometer values to achieve any desired color.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.14 lesson 14: Understanding Variable Types and Arrays in Micropython

This tutorial covers how MicroPython handles variables and arrays using the Raspberry Pi Pico W:

- **Variables in MicroPython**: Explains the dynamic typing in MicroPython, showing how variables can be integers, floats, strings, or Booleans without explicit declarations. Discusses common pitfalls like unintended type conversions and how to avoid them.
- **Basic Operations**: Demonstrates basic arithmetic operations with variables and highlights how division always results in a float, even when dividing integers.

- **String Operations**: Explains how strings are handled in MicroPython, including concatenation using the plus sign and type errors that can occur when mixing strings with other data types.
- **Boolean Operations**: Introduces Boolean variables and logical operations like AND, OR, and NOT. Shows how true and false are represented and used in MicroPython.
- **Arrays (Lists) in MicroPython**: Shows how to create and manipulate one-dimensional arrays (lists) and access individual elements using indexing. Explains how to modify elements and append new elements to the list.
- **Two-Dimensional Arrays**: Demonstrates the creation and use of two-dimensional arrays (lists of lists), including how to access and modify individual elements within the nested lists.
- **Advanced Array Manipulations**: Covers how to concatenate arrays, create arrays with mixed data types, and handle unexpected behaviors that arise from such operations.
- **Practical Examples**: Provides practical examples throughout the tutorial, showing real-time execution of code snippets to illustrate how variables and arrays work in MicroPython.
- **Homework Assignment**: Encourages viewers to practice these concepts by creating and manipulating arrays and experimenting with different data types and operations.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.15 lesson 15: For Loops in Micropython

This tutorial covers using for loops in MicroPython with the Raspberry Pi Pico W:

- **Introduction to For Loops**: Explains the difference between for loops and while loops, and when to use each type of loop. Highlights the usefulness of for loops in cases where a specified number of iterations is needed.
- **Basic For Loop Syntax**: Demonstrates the basic syntax of a for loop in MicroPython, showing how to loop through a range of numbers. Explains the importance of specifying the start, stop, and step values in the range function.
- **Looping Through Arrays**: Shows how to use for loops to iterate through arrays, including arrays with numbers, floating point numbers, and strings. Illustrates the flexibility of for loops in handling different data types.
- **Nested Loops**: Introduces nested loops, where one loop runs inside another. Provides examples of how nested loops can be used to iterate through multi-dimensional arrays or perform complex operations.
- **Practical Example with Hardware**: Connects the Raspberry Pi Pico W to an LED and demonstrates how to use a for loop to control the blinking of the LED. Shows how to get user input for the number of blinks and use a for loop to execute the blinking sequence.

- **Combining Loops and User Input**: Combines user input with for loops to create more dynamic programs. Illustrates how to convert user input from strings to integers to avoid common errors.
- **Homework Assignment**: Assigns a project where the user needs to replace the LED with an RGB LED and write a program that asks for the number of colors to display. The program should then ask for each color and display them in sequence on the RGB LED. The colors should include red, green, blue, cyan, magenta, yellow, orange, white, and off.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.16 lesson 16: Sequence of Colors in RGB LED With Micropython

This tutorial covers using for loops in MicroPython with the Raspberry Pi Pico W to control an RGB LED:

- **Introduction**: The lesson starts with a brief introduction and an overview of the lesson, focusing on using for loops and PWM (Pulse Width Modulation) to control the brightness and color of an RGB LED.
- **Circuit Setup**: Demonstrates the wiring of the RGB LED to the Raspberry Pi Pico W, using GPIO pins 13, 14, and 15 for the red, green, and blue channels, respectively. Emphasizes the importance of using 330 Ohm current limiting resistors.
- **PWM Setup**: Explains setting up PWM channels for each LED color, establishing a frequency of 1000 Hz to ensure smooth transitions without visible flickering.
- **Color Sequence Input**: Shows how to prompt the user for a sequence of colors and capture the input. Uses a for loop to store the user-defined colors in an array, ensuring that the input is converted to lowercase for consistency.
- **Color Control Logic**: Details the logic for setting the brightness of each LED color based on the user's input. Implements if statements to assign PWM values to create different colors, including red, green, blue, cyan, magenta, yellow, orange, and off.
- **Continuous Loop for Displaying Colors**: Uses a while true loop to continuously cycle through the user-defined color sequence. Adds sleep statements to control the duration each color is displayed.
- **Practical Demonstration**: Provides a step-by-step demonstration of running the code, entering a sequence of colors, and observing the RGB LED displaying the colors in the specified order.
- **Next Steps**: Announces that the next lesson will cover using push buttons with the Raspberry Pi Pico W to read digital inputs, completing the learning of input and output methods before moving on to more advanced components.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.17 lesson 17: Buttons and Pull Up Resistors in Micropython

This tutorial covers using push buttons in MicroPython with the Raspberry Pi Pico W:

- **Introduction**: The lesson starts with a brief introduction and an overview of the lesson, focusing on using push buttons to control components like LEDs on the Raspberry Pi Pico W.
- **Types of Push Buttons**: Describes different types of push buttons available in the SunFounder kit, explaining their functionality and how they connect within a circuit.
- **Circuit Setup**: Demonstrates the wiring of a push button to GPIO pin 14 on the Raspberry Pi Pico W, including the use of internal pull-up resistors to simplify the circuit. Provides a detailed schematic and visual setup.
- **How Push Buttons Work**: Explains the concept of open and closed states of push buttons and how they are used to control circuits by making or breaking connections.
- **Activating Internal Pull-Up Resistors**: Shows how to activate the internal pull-up resistors on the Raspberry Pi Pico W using MicroPython, eliminating the need for external resistors.
- **Code Explanation**: Describes the MicroPython code for reading the state of a push button. Covers setting up the GPIO pin, reading the button state, and printing the button state to the console.
- **Practical Demonstration**: Provides a step-by-step demonstration of running the code, pressing the push button, and observing the changes in the button state readings on the console.
- **Homework Assignment**: Assigns a project to use a push button to control an LED. The task is to write a program where pressing the button turns the LED on, and pressing it again turns the LED off.
- **Next Steps**: Announces that the next lesson will continue exploring the use of push buttons and LEDs, focusing on integrating these components into more complex projects.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.18 lesson 18: Push Button Control of LEDs in Micropython

This tutorial covers using push buttons to toggle an LED on and off with the Raspberry Pi Pico W:

- **Introduction**: The lesson starts with a brief introduction and an overview of the lesson, focusing on using push buttons to control LEDs on the Raspberry Pi Pico W.
- **Homework Solution**: Provides a solution to the previous lesson's homework, which involved creating a toggle switch for an LED using a push button.
- **Circuit Setup**: Demonstrates the wiring of a push button to GPIO pin 14 and an LED to GPIO pin 15 on the Raspberry Pi Pico W. Includes connecting the push button and LED to a common ground.
- **Toggle Switch Logic**: Explains the logic required to toggle the LED state with each press and release of the push button. Introduces the concepts of *button state now*, *button state old*, and *LED state*.
- **Code Explanation**: Initializes the GPIO pins for the LED and push button.Uses a while loop to continuously read the button state.Toggles the LED state based on the button press and release actions.Includes print statements for debugging and verifying the states.
- **Practical Demonstration**: Provides a step-by-step demonstration of running the code, pressing the push button, and observing the changes in the LED state.
- **Homework Assignment**: Assigns a new project to use multiple push buttons to control the colors of an RGB LED. The task is to write a program where pressing the red, green, or blue button changes the color of the RGB LED accordingly.
- **Next Steps**: Announces that future lessons will continue exploring the use of push buttons and LEDs, focusing on integrating these components into more complex projects.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.19 lesson 19: Button Control of RGB LED in Micropython

This tutorial covers using multiple push buttons to control the RGB channels of an LED on the Raspberry Pi Pico W:

- **Introduction**: The lesson starts with an overview of the lesson, focusing on using three push buttons to control the red, green, and blue channels of an RGB LED on the Raspberry Pi Pico W.
- **Homework Solution**: Provides a solution to the previous lesson's homework, which involved creating three toggle switches for an RGB LED using push buttons.
- **Circuit Setup**: Demonstrates the wiring of three push buttons and an RGB LED to the Raspberry Pi Pico W. Each button is connected to a GPIO pin, and the RGB LED channels are connected through resistors to separate GPIO pins. The ground rail is established to complete the circuit.
- **Code Explanation**:
- Initializes the GPIO pins for the RGB LED and push buttons.
- Creates objects for each LED channel and button, setting them as inputs or outputs as needed.
- Defines variables for button states and LED states.
- Implements logic to toggle the LED channels based on button presses and releases.
- Demonstrates how to write and organize the code methodically, ensuring consistent variable naming for readability and ease of debugging.
- **Practical Demonstration**:
- Provides a step-by-step demonstration of running the code, pressing the push buttons, and observing the changes in the RGB LED channels.
- Shows how to test the button functionality and verify that each button correctly toggles its respective LED channel.
- Discusses debugging techniques to identify and fix errors in the code or circuit.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.20 lesson 20: Using the DHT11 Temperature and Humidity Sensor in MicroPython

This tutorial covers measuring temperature and humidity using the DHT11 sensor with the Raspberry Pi Pico W:

- **Introduction**: introduce the lesson, focuse on using the DHT11 sensor to measure temperature and humidity.
- **Previous Lessons Recap**: Recaps the foundational skills learned in previous lessons, such as digital writes, analog writes using PWM, digital reads, and analog reads.
- **Component Introduction**: Introduces the DHT11 sensor from the SunFounder Kepler kit and demonstrates how to find it within the kit.
- **Circuit Setup**:
- Establishes ground and power rails on the breadboard.
- Connects the DHT11 sensor to the Raspberry Pi Pico W: - Pin 1 of the sensor to 3.3V. - Pin 2 of the sensor to GPIO pin 16 (physical pin 21). - Pin 4 of the sensor to ground.
- **Code Explanation**:
- Imports necessary libraries: machine, utime (as time), and DHT.
- Sets up the GPIO pin for data input with a pull-down resistor.
- Initializes the DHT11 sensor.
- Enters an infinite loop to continuously measure and print temperature and humidity.
- Explains how to format the printed output to display temperature and humidity in a single line using *r*.
- **Practical Demonstration**:
- Runs the code and observes temperature and humidity readings in real-time.
- Discusses the importance of not using methods that might cause condensation on the sensor to test changes in readings.
- **Formatting Output**:
- Demonstrates how to format the output to show the temperature in degrees Celsius and humidity in percentage.
- Explains how to print the degree symbol using ASCII character codes.
- **Homework Assignment**:
- Adds a push button to the circuit.
- Modifies the code to toggle between displaying temperature in Celsius and Fahrenheit when the button is pressed.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.

- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.21 lesson 21: Temperature and Humidity Measurements with Toggle Switch

This tutorial covers adding a toggle push button to switch between temperature readings in Fahrenheit and Celsius using the DHT11 sensor with the Raspberry Pi Pico W:

- **Introduction**: introduces the tutorial and acknowledges the sponsor, SunFounder. He explains the goal of adding a toggle push button to the existing temperature and humidity measurement setup.
- **Previous Lessons Recap**: Reviews the previous lesson on using the DHT11 sensor and sets the context for the current task.

Component Introduction and Circuit Setup: - Reintroduces the DHT11 sensor and explains the addition of a push button to the circuit. - Describes the connections:

- DHT11 sensor: - Pin 1 to 3.3V - Pin 2 to GPIO pin 16 - Pin 4 to ground
- Push button: - One leg to ground - The other leg to GPIO pin 15
- **Code Explanation**:
- Imports necessary libraries: machine, utime (as time), and DHT.
- Sets up GPIO pins for the DHT11 sensor and the push button.
- Creates a toggle mechanism to switch between temperature units (Celsius and Fahrenheit).
- Reads the button state and toggles the temperature unit when the button is pressed and released.
- Measures and converts temperature from Celsius to Fahrenheit.
- Prints temperature and humidity readings in a single line using *r* for a clean output.
- Addresses formatting issues to ensure the output displays correctly when toggling between Celsius and Fahrenheit.
- **Practical Demonstration**:
- Runs the code to observe temperature and humidity readings.
- Demonstrates the toggle functionality between Celsius and Fahrenheit when the button is pressed.
- Fixes formatting issues to ensure the output is clean and consistent.
- **Homework Assignment**:
- Adds more toggle functionality to the project.
- Implements additional toggles to switch between displaying temperature in Celsius, Fahrenheit, and humidity one at a time.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.22 lesson 22: Using an LCD Display with the Pico W

This tutorial covers connecting and using an LCD 1602 display with the Raspberry Pi Pico W:

- **Introduction**: introduces the tutorial, acknowledges the sponsor SunFounder, and explains the goal of adding an LCD display to the Raspberry Pi Pico W project for mobile use.
- **Component Introduction and Setup**:
- Describes the required components: LCD 1602 display and female-to-male wires.
- Details the connections: - LCD 1602 pins to Raspberry Pi Pico W:
  - Ground to pin 38
  - VCC (5V) to the rightmost pin
  - SDA (data) to GPIO pin 6
  - SCL (clock) to GPIO pin 7
- **Library Installation**:
- Guides on how to download and install the necessary library for the LCD 1602 display from toptechboy.com.
- Provides instructions to save and import the library in Thonny IDE.
- **Code Explanation**:
- Describes the creation of an LCD object and writing text to the LCD.
- Provides a sample program that prompts for the user's name and displays a greeting message on the LCD.
- Addresses potential issues with text overlap by using *LCD.clear()* to clear the screen before writing new text.
- **Practical Demonstration**:
- Shows the program in action, displaying names on the LCD.
- Explains adjusting the LCD contrast using a potentiometer on the back of the display.
- **Homework Assignment**:
- Assigns a task to integrate the LCD display with the DHT11 temperature and humidity sensor project from lesson 21.
- Instructs to display temperature in Celsius or Fahrenheit based on a toggle button and display humidity on the LCD.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.23 lesson 23: Temperature and Humidity Sensor with LCD Display

This tutorial covers creating a temperature and humidity project using the Raspberry Pi Pico W and DHT-11 sensor with an LCD display:

- **Introduction**: introduces the tutorial, acknowledges the sponsor SunFounder, and reviews the goal of the lesson, which is to create a temperature and humidity sensor project with an LCD display.
- **Component Introduction and Setup**:
- Describes the required components: Raspberry Pi Pico W, DHT-11 sensor, push button, and LCD 1602 display.
- Provides a schematic and instructions for connecting these components, referring to previous lessons for detailed wiring instructions.
- **Library Installation**:
- Guides on how to download and install the necessary library for the LCD 1602 display from toptechboy.com.
- Provides instructions to save and import the library in Thonny IDE.
- **Code Explanation**:
- Describes setting up the DHT-11 sensor and push button using GPIO pins.
- Introduces a flag variable for toggling between Celsius and Fahrenheit.
- Explains the code for reading temperature and humidity from the DHT-11 sensor.
- Details the implementation of a toggle function to switch between Celsius and Fahrenheit.
- Explains how to display the readings on both the console and the LCD.
- **Practical Demonstration**:
- Shows the program in action, displaying temperature and humidity readings on the LCD.
- Demonstrates the toggle functionality to switch between Celsius and Fahrenheit.
- Addresses potential issues with text overlap on the LCD by adding spaces and using *LCD.clear()* to clear the screen before writing new text.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.24 lesson 24: Power Portable Projects with LiPo Rechargeable Battery

This tutorial covers powering a Raspberry Pi Pico W project with a rechargeable LiPo battery, making the project portable:

- **Introduction**: introduces the tutorial, acknowledging the sponsor SunFounder, and reviews the goal of the lesson: making the DHT-11 temperature and humidity sensor project portable.
- **Component Review and Setup**:
- Recaps the previous project setup: Raspberry Pi Pico W, DHT-11 sensor, push button, and LCD 1602 display.
- Discusses the need to remove the USB cable and power the project using a LiPo battery.
- **Library Installation**:
- Guides viewers on how to install the LCD 1602 library if they haven't already.
- **Code Explanation**:
- Describes the necessity of saving the program as *main.py* so it runs automatically when the Pico W is powered.
- Demonstrates saving the code and verifying its functionality.
- **Powering the Project with a Battery**:
- Introduces the rechargeable LiPo battery and the battery holder from the SunFounder Kepler kit.
- Details connecting the battery to the Pico W using a power management module included in the kit.
- Explains adjusting the LCD power connection from 5V (pin 40) to 3.7V (pin 39) due to the battery output.
- Shows the setup and verifies the project runs correctly on battery power.
- **Adjustments and Demonstration**:
- Demonstrates adjusting the LCD contrast for optimal display when powered by the battery.
- Shows the project working correctly, including toggling between Fahrenheit and Celsius.
- **Conclusion and Next Steps**:
- Encourages viewers to ensure their project is working and introduces the idea of using an OLED display for future lessons.

- Recommends purchasing an OLED display for a more compact and energy-efficient setup.
- Teases future lessons focused on creating portable, deployable projects with enhanced functionality.
- **Homework and Final Notes**:
- Suggests viewers order the OLED display to prepare for upcoming lessons.
- Concludes with reminders to like, comment, subscribe, and share the video.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.25 lesson 25: Getting Started with OLED 1306 in Micropython

This tutorial covers using the Raspberry Pi Pico W and an OLED display for portable projects:

- **Introduction**:
- Highlights the goal: continuing to untether projects from the desktop by making them portable and more power-efficient.
- **Component Review and Setup**:
- Recaps the previous lesson: using a rechargeable LiPo battery to power the Raspberry Pi Pico W with a DHT-11 sensor, push button, and LCD display.
- Discusses the drawbacks of using an LCD display, including higher power consumption and larger size.
- **Introducing the OLED Display**:
- Recommends using a small, low-power, high-contrast OLED display for portable projects.
- Shows how to connect the OLED display to the Raspberry Pi Pico W using the I2C bus, specifically on GPIO pins 2 (SDA) and 3 (SCL).
- **Library Installation and Initial Setup**:
- Demonstrates how to install the SSD1306 library for the OLED display.
- Explains the basic code setup, including importing necessary libraries and creating the I2C and display objects.
- **Displaying Text and Graphics**:
- Provides code examples to display text on the OLED screen.
- Shows how to address individual pixels and draw horizontal, vertical, and arbitrary lines.

- Demonstrates how to create and fill rectangles on the display.
- **Power Management**:
- Explains how to turn the display on and off to save battery power using software commands.
- **Practical Demonstration**:
- Runs the code to show text and graphics on the OLED display.
- Emphasizes the high contrast and low power consumption of the OLED display compared to the LCD.
- **Homework Assignment**:
- Assigns a task: create a program that displays the title "My Circle" at the top of the screen and draws a circle with a 20-pixel radius in the center of the screen.
- Encourages viewers to post their homework on YouTube and share the link in the comments.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.26 lesson 26: Drawing a Circle on the OLED 1306 Display

This tutorial covers drawing shapes on an OLED display using the Raspberry Pi Pico W:

- **Introduction**:
- Highlights the goal: drawing a circle on an OLED display using the Raspberry Pi Pico W.
- **Recap and Setup**:
- Reviews the previous lesson on using the OLED 1306 display and setting up the SSD1306 library.
- Discusses the importance of using mathematical functions to draw shapes on the OLED display.
- **Drawing a Circle**:
- Explains the math behind drawing a circle using trigonometric functions: (x = r cos(theta)) and (y = r sin(theta)).
- Converts degrees to radians for use in programming languages.
- Shows how to center the circle on the OLED display by adjusting the x and y coordinates.
- Provides a step-by-step code example to draw a circle by iterating through 360 degrees and calculating the x and y positions.

- **Enhancing the Circle Drawing**:
- Demonstrates how to draw a filled circle by iterating over a range of radii.
- Shows how to draw a partial circle or arc by adjusting the angle range.
- **Practical Demonstration**:
- Runs the code to show the drawing of a circle and a filled circle on the OLED display.
- Explains how to optimize the drawing speed by updating the display only after all points are calculated.
- **Homework Assignment**:
- Assigns a task: create a program that draws a "floating potato chip" shape on the OLED display.
- Encourages viewers to figure out the math behind the shape and post their homework on YouTube.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.27 lesson 27: Creating Lissajous Patterns on an OLED Display

This tutorial covers generating Lissajous patterns on an OLED display using the Raspberry Pi Pico W:

- **Introduction**:
- Highlights the goal: generating the floating "potato chip" Lissajous pattern on an OLED display using the Raspberry Pi Pico W.
- **Recap and Setup**:
- Reviews the previous lesson on drawing a circle on the OLED display.
- Explains the math behind creating Lissajous patterns using trigonometric functions.
- **Generating the Pattern**:
- Provides a step-by-step code example to draw a circle and then modify it to generate the Lissajous pattern.
- Discusses the importance of converting degrees to radians and using sine and cosine functions for the calculations.
- **Enhancing the Pattern**:
- Shows how to adjust the parameters to change the appearance of the pattern.
- Demonstrates adding phase shift to create a rotating effect for the pattern.

- **Practical Demonstration**:
- Runs the code to show the generation of the Lissajous pattern on the OLED display.
- Explains how to optimize the display update speed and clear the display for smooth animations.
- **Homework Assignment**:
- Assigns a task: create a Lissajous pattern on the OLED display and post the solution on YouTube.
- Encourages viewers to experiment with different parameters and share their results.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.28 lesson 28: Connecting the Raspberry Pi Pico W to WiFi

This tutorial covers setting up a client-server relationship between a Raspberry Pi Pico W and a PC over Wi-Fi:

- **Introduction**:
- Highlights the goal: connecting the Raspberry Pi Pico W to Wi-Fi and creating a client-server relationship with a PC.
- **Understanding Client-Server Relationship**:
- Explains the concept using the analogy of a server and client at McDonald's.
- Clarifies that the PC will be the client, and the Raspberry Pi Pico W will be the server.
- **Setting Up the Raspberry Pi Pico W as a Server**:
- Details the necessary libraries and steps to create a Wi-Fi network.
- Explains how to connect to Wi-Fi and retrieve an IP address.
- Describes setting up a UDP server on the Raspberry Pi Pico W and binding it to the IP address and port.
- **Creating a Client on the PC**:
- Provides code for the PC client to connect to the Raspberry Pi Pico W server.
- Explains sending commands from the client to the server.
- **Sending and Receiving Data**:
- Shows how the server receives commands from the client and decodes them.
- Demonstrates sending a response back from the server to the client.

- **Practical Demonstration**:
- Runs the server and client programs, showing the interaction and data exchange.
- Displays a battery-powered Raspberry Pi Pico W with an OLED display showing its IP address.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.29 lesson 29: Simple Client Server Project to Control RGB LED

This tutorial covers setting up a remote-controlled RGB LED using a Raspberry Pi Pico W and a PC over Wi-Fi:

- **Introduction**:
- The goal is to control an RGB LED on a Raspberry Pi Pico W remotely from a PC using Wi-Fi.
- **Wiring Diagram and Setup**:
- RGB LED's red, green, and blue channels are connected to GPIO pins 16, 17, and 18, respectively.
- OLED display is connected via I2C to GPIO pins 2 (SDA) and 3 (SCL).
- **Server Side Setup (Raspberry Pi Pico W)**:
- Import libraries: *socket*, *time*, *network*, *machine*, *ssd1306*.
- Initialize GPIO pins for the RGB LED and OLED display.
- Connect to Wi-Fi and obtain an IP address.
- Create a UDP server socket and bind it to the IP address and a port.
- Display the IP address and port on the OLED screen.
- Listen for incoming commands, decode them, and display the command and sender's address.
- **Client Side Setup (PC)**:
- Import the *socket* library.
- Define the server address and port.
- Create a UDP client socket.
- Get user input for the LED color, encode the command, and send it to the server.
- Wait for and print the server's response.

- **Practical Demonstration**:
- Demonstrate sending commands from the client to change the RGB LED color on the server.
- OLED display shows the received commands and the sender's IP address.
- **Final Setup and Testing**:
- Disconnect the Raspberry Pi Pico W from USB and power it with a battery.
- Save the code as *main.py* to run on startup.
- Demonstrate fully wireless operation by sending commands from the PC and observing the RGB LED changes and OLED updates.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.30 lesson 30: Project to Connect Your Raspberry Pi Pico W to the Internet

This tutorial covers connecting the Raspberry Pi Pico W to the internet and fetching data from APIs:

Introduction: - The goal is to connect the Raspberry Pi Pico W to the internet and fetch real-time data from APIs. - No additional hardware setup required.

Connecting to WiFi: - Import necessary libraries: *network*, *time*, *urequests*. - Create a WLAN object and connect to the WiFi network. - Ensure successful connection before proceeding.

Fetching Data from APIs: - Introduction to JSON files and their structure (arrays, dictionaries, nested elements). - Example API used: Fetching real-time data about astronauts in space. - Parse and print the data structure to understand its format.

Code Explanation: - Use *urequests.get()* to fetch JSON data from a specified API endpoint. - Parse the JSON data to extract relevant information. - Example: List names of astronauts currently in space and their respective spacecraft.

Practical Demonstration: - Code snippet to fetch and display data about astronauts. - Demonstrates how to navigate nested JSON structures to extract specific data. - Example output: Number of astronauts, their names, and spacecraft.

Homework Assignment: - Find an interesting real-time data set (e.g., weather, stock prices, earthquakes). - Fetch and parse the data using the Raspberry Pi Pico W. - Display or utilize the data in a meaningful way (e.g., sensorless weather station).

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.31 lesson 31: Sensorless Remote Weather Station Project

This tutorial covers creating a sensorless weather station using the Raspberry Pi Pico W:

- **Connecting to WiFi**:
- Import necessary libraries.
- Create a WLAN object and connect to the WiFi network.
- **Fetching Weather Data**:
- Use the OpenWeatherMap API to fetch real-time weather data.
- Obtain an API key from OpenWeatherMap by signing up for a free account.
- **Parsing JSON Data**:
- Extract relevant weather information such as temperature, humidity, barometric pressure, sunrise, and sunset times.
- **Code Explanation**:
- Use *urequests.get()* to fetch JSON data from the API endpoint.
- Convert Unix epoch time to local time for sunrise and sunset.
- Convert barometric pressure from HPA to atmospheres.
- **Displaying Weather Data**:
- Print weather information like temperature, humidity, barometric pressure, conditions, and wind speed.
- **Homework Assignment**:
- Enhance the project by adding a display to show the weather information.
- Create a portable, battery-powered weather station.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.32 lesson 32: Mobile Weather Station Project

This tutorial covers creating a portable weather station using the Raspberry Pi Pico W:

- **Connecting to WiFi**:
- Import necessary libraries.
- Create a WLAN object and connect to the WiFi network.
- **Fetching Weather Data**:
- Use the OpenWeatherMap API to fetch real-time weather data.
- Obtain an API key from OpenWeatherMap by signing up for a free account.
- **Parsing JSON Data**:
- Extract relevant weather information such as temperature, humidity, barometric pressure, sunrise, and sunset times.
- **Displaying Data on OLED**:
- Set up an OLED display and connect it to the Raspberry Pi Pico W.
- Use the *ssd1306* library to display weather data on the OLED screen.
- Create a loop to periodically update the weather data on the screen.
- **Powering the Device**:
- Connect the Raspberry Pi Pico W to a battery for portable use.
- **Code Explanation**:
- Step through initializing the OLED display and connecting to WiFi.
- Fetch and parse weather data, then display it on the OLED screen.
- Implement a loop to update the weather data every few minutes.
- **Homework Assignment**:
- Enhance the project by adding an RGB LED to visually indicate weather conditions such as temperature, humidity, or wind speed.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.33 lesson 33 : Display the HSV Color Wheel on an RGB LED

This tutorial covers displaying the HSV (Hue, Saturation, Value) color wheel on an RGB LED using the Raspberry Pi Pico W:

- **Introduction to HSV Color Wheel**:
- Introduction to the concept of HSV color representation and its importance in visualizing temperature data.
- Explanation of the HSV color wheel and how different angles represent different colors.
- **Project Setup and Goal**:
- Recap of the previous lesson where a remote sensorless weather station was built using the Raspberry Pi Pico W and an OLED display.
- Introduction to the new objective of displaying a visual representation of temperature using an RGB LED.
- **Understanding HSV to RGB Conversion**:
- Detailed explanation of the HSV color wheel, focusing on angles and corresponding colors.
- Explanation of the mathematical conversion from HSV to RGB values.
- **Algorithm Development**:
- Step-by-step breakdown of how to convert HSV angles to RGB values.
- Explanation of how to map temperature values to specific angles on the HSV color wheel.
- Practical application of this algorithm to the RGB LED to display temperature data.
- **Homework Assignment**:
- Task to write a function that converts an angle from the HSV color wheel to RGB values.
- Encouragement to use the RGB LED to visually represent weather conditions, such as temperature and wind speed.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.34 lesson 34 : Convert HSV to RGB in Micropython

This tutorial covers converting HSV (Hue, Saturation, Value) color values to RGB (Red, Green, Blue) values and displaying them on an RGB LED using the Raspberry Pi Pico W:

- **Introduction to the HSV Color Wheel**:
- Explanation of the HSV color wheel and its significance in representing smooth color transitions.
- Overview of how the HSV color wheel can be used to visually represent temperature data.
- **Project Setup and Goal**:
- Recap of the weather station project that fetches weather data and displays it on an OLED screen.
- Introduction to the goal of adding a visual representation of temperature using an RGB LED.
- **Understanding HSV to RGB Conversion**:
- Breakdown of the HSV color wheel into a mathematical representation for conversion.
- Explanation of the six zones in the color wheel and the respective RGB values and slopes.
- **Algorithm Development**:
- Step-by-step creation of a function to convert HSV angles to RGB values.
- Explanation of setting up the RGB LED circuit with PWM control on the Raspberry Pi Pico W.
- **Code Implementation**:
- Detailed walkthrough of the Python code to set up PWM for the RGB LED and perform the HSV to RGB conversion.
- Creation of a library function for the conversion to simplify the main program.
- **Practical Demonstration**:
- Demonstration of the code in action, showing the RGB LED changing colors according to the HSV color wheel.
- Assignment to integrate the RGB LED with the weather station project to display temperature data visually.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.35 lesson 35 : Remote Weather Station with RGB LED Temperature Indicator

This tutorial covers integrating an RGB LED to display temperature data on a weather station using the Raspberry Pi Pico W:

- **Project Overview**:
- Build a remote weather station using Raspberry Pi Pico W, an OLED display, and an RGB LED to show temperature visually.
- **HSV to RGB Conversion**:
- Convert temperature values to colors using the HSV color wheel.
- Map temperatures from -20°F (deep violet) to 120°F (red) to corresponding angles on the color wheel.
- **Circuit Setup**:
- Connect the OLED display and RGB LED to the Raspberry Pi Pico W.
- Configure GPIO pins and PWM for the RGB LED.
- **Coding**:
- Fetch temperature data, calculate the hue angle, convert it to RGB values, and apply these to the RGB LED.
- Integrate the HSV to RGB conversion library into the weather station code.
- **Demonstration**:
- Display temperature data on the OLED and RGB LED.
- Save and run the code, making the setup portable with battery power.
- **Conclusion**:
- Customize the project by adjusting color mappings and temperature ranges.
- Encouragement to subscribe, comment, and share the tutorial.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.36 lesson 36 : Control a Servo With MicroPython

This tutorial covers controlling a servo motor using the Raspberry Pi Pico W:

- **Servo Motor Control**:
- Introduction to using the SG90 servo motor with the Raspberry Pi Pico W.
- Explanation of servo motor connections: brown (ground), red (power), orange (control).
- Emphasis on not using the Raspberry Pi Pico W to power larger servos due to potential damage.
- **Wiring Diagram and Setup**:
- Detailed wiring instructions for connecting the SG90 servo to the Raspberry Pi Pico W.
- Use of GPIO pin 15 for control, and creating a power rail from pin 1 for 5V supply.
- **PWM Basics**:
- Explanation of Pulse Width Modulation (PWM) and its role in controlling servo position.
- Calculation of pulse widths corresponding to different servo angles (0.5ms for 0 degrees, 2.5ms for 180 degrees).
- Setting PWM frequency to 50Hz to match the servo's requirements.
- **Code Explanation**:
- Step-by-step coding tutorial to set up PWM on GPIO pin 15.
- Creating a function to convert desired angles to PWM duty cycle values.
- Example code to control the servo by entering desired angles.
- **Practical Demonstration**:
- Running the code to move the servo to specific angles.
- Ensuring safe servo operation by avoiding manual rotation of the servo horn.
- **Homework Assignment**:
- Task to integrate a potentiometer with the setup to control the servo position by adjusting the potentiometer.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.37 lesson 37 : Control a Servo With a Potentiometer in MicroPython

This tutorial covers controlling a servo motor using a potentiometer with the Raspberry Pi Pico W:

- **Servo Motor Control**:
- Connect the SG90 servo motor to the Raspberry Pi Pico W.
- Brown wire to ground, red wire to power (5V), orange wire to GPIO pin 15 for control.
- **Wiring Setup**:
- Connect the potentiometer: power to 3.3V, ground to ground rail, and signal to GPIO pin 26.
- **PWM Basics**:
- Use Pulse Width Modulation (PWM) to control servo position.
- Set PWM frequency to 50Hz for the servo.
- **Code Explanation**:
- Set up PWM on GPIO pin 15.
- Convert potentiometer readings to servo angles.
- Example code provided to move the servo based on potentiometer input.
- **Practical Demonstration**:
- Run the code to control the servo with the potentiometer.
- Avoid manually rotating the servo horn to prevent damage.
- **Application Ideas**:
- Control larger servos with an external power supply for more advanced projects.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.38 lesson 38 : Learn Trigonometry Fundamentals in One Session (EASY)

This tutorial covers the basics of trigonometry and its applications using the Raspberry Pi Pico W:

- **Introduction to Trigonometry**:
- Explanation of common issues and misconceptions in learning trigonometry.
- Focus on simplifying concepts using circles instead of triangles.
- **Understanding Circles**:
- Explanation of unit circles and how rotating a vector traces a circle.
- Introduction to radians and degrees, including conversion between the two.
- **Basic Trigonometric Functions**:
- Explanation of sine, cosine, and tangent functions using circles.
- Demonstration of calculating sine and cosine for given angles.
- – Introduction to inverse functions (arcsine, arccosine) to find angles from sine and cosine values.
- **Programming Examples**:
- Using MicroPython to calculate sine and cosine values.
- Example code for converting degrees to radians and calculating trigonometric functions.
- **Homework Assignment**:
- Find sine and cosine values for 0°, 90°, 180°, 270°, and 360° without using a calculator, focusing on understanding the concepts visually.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.39 lesson 39 : Practical Trigonometry for Easy Problem Solving

This tutorial covers practical trigonometry necessary for real-life applications using the Raspberry Pi Pico W:

- **Introduction to Practical Trigonometry**:
- Recap of previous lesson on trigonometry, focusing on understanding circles and angles.
- Explanation of using trigonometry to solve problems involving the MPU 6050 sensor for tilt measurement.
- **Key Concepts**:
- Understanding trigonometric functions (sine, cosine) in relation to circles.
- Converting between degrees and radians for calculations.
- Using trigonometric functions to find coordinates (x, y) on a circle based on an angle.
- Calculating angles from given coordinates using arc functions (arcsine, arccosine).
- **Applying Trigonometry to Real-World Problems**:
- Explanation of hypotenuse, adjacent, and opposite sides in right triangles.
- Rewriting trigonometric equations using hypotenuse, adjacent, and opposite sides.
- Solving vector problems and inclined plane problems using trigonometry.
- Example problems demonstrating practical applications.
- **Summary and Homework**:
- Recap of key trigonometric equations and concepts.
- Homework assignment to practice finding sine and cosine values for specific angles without using a calculator.
- Emphasis on understanding the concepts to apply them in future projects involving the MPU 6050 sensor.
- **Next Steps**:
- Preview of upcoming lessons on using the MPU 6050 sensor for tilt measurement projects.
- Encouragement to review and understand the trigonometry concepts for successful project implementation.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.40 lesson 40 : Measuring Acceleration with the MPU6050 Six Axis IMU

This tutorial covers using the MPU6050 accelerometer and gyroscope with the Raspberry Pi Pico W:

- **Introduction**:
- Introduction to using the MPU6050 sensor for measuring acceleration and gyroscopic movements.
- Discussion on the significance of MEMS (Microelectromechanical Systems) technology in creating these sensors.
- **Setting Up the Hardware**:
- Explanation of the hardware components: MPU6050 sensor and OLED display from the SunFounder Kepler Kit.
- Detailed wiring instructions for connecting the MPU6050 and OLED display to the Raspberry Pi Pico W.
- **Understanding the Sensor**:
- Overview of how the MPU6050 measures acceleration using MEMS technology.
- Explanation of the physical principles behind accelerometers, including Hooke's Law and capacitance measurement.
- Description of the MPU6050's capability to measure acceleration in X, Y, and Z axes.
- **Installing Required Libraries**:
- Instructions for downloading and installing the necessary libraries from the SunFounder GitHub repository.
- Steps to import and set up the libraries in Thonny IDE.
- **Coding and Testing**:
- Step-by-step guide to writing MicroPython code to read acceleration values from the MPU6050.
- Explanation of initializing the I2C bus and creating the MPU6050 object.
- Code snippets to measure and print acceleration in the X and Y axes.
- Demonstration of the Thonny plotter to visualize acceleration readings in real-time.
- **Homework Assignment**:
- Task to measure acceleration in the Z-axis and predict its value when the sensor is stationary.
- Instructions to compare the predicted value with the actual measurement and analyze any discrepancies.
- **Conclusion**:
- Summary of the lesson and a preview of future lessons focusing on practical applications of the MPU6050 sensor.
- Encouragement to share findings and engage with the community for further learning.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.41 lesson 41 : Inferring Tilt From a 3 Axis Accelerometer

This tutorial covers measuring acceleration in three axes using the MPU6050 sensor with the Raspberry Pi Pico W:

- **Introduction**:
- Measuring acceleration in X, Y, and Z axes.
- Understanding the Z-axis acceleration due to gravity.
- **Review of Previous Lesson**:
- Recap of measuring X and Y axes acceleration.
- Homework assignment: measure and predict Z-axis behavior.
- **Z-Axis Acceleration**:
- Explanation of 1G acceleration in the Z-axis when stationary due to gravity.
- Demonstration of tilting the sensor affecting X, Y, and Z readings.
- **Practical Application**:
- Using the sensor to measure tilt (pitch and roll) for applications in carpentry, off-roading, and aviation.
- **Homework Assignment**:
- Calculate tilt angles (pitch and roll) using sensor data.
- Apply mathematical concepts from previous lessons.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.42 lesson 42 : Measuring Tilt With an MPU6050 Accelerometer

This tutorial covers using the MPU6050 sensor with the Raspberry Pi Pico W to measure and calculate tilt angles:

- **Introduction:**
  - Measuring acceleration in X, Y, and Z axes with the MPU6050.
  - Setting up the hardware and software.
- **Previous Lesson Recap:**
  - Measuring X, Y, Z axes acceleration.
  - Homework to calculate tilt based on accelerometer data.
- **Z-Axis Acceleration:**
  - 1G acceleration in the Z-axis due to gravity when stationary.
  - Tilt affects X, Y, and Z readings.
- **Calculating Tilt Angles:**
  - Using trigonometry to derive equations for tilt angles (pitch and roll).
  - Implementing equations in Python to measure tilt.
- **Practical Application:**
  - Testing code for accurate tilt measurement.
  - Homework to modify code for positive and negative pitch and roll.
- **Conclusion:**
  - Encouragement to understand principles rather than relying on libraries.
  - Support through Patreon and sharing the video.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.43 lesson 43 : Measure Pitch and Roll Using a 3 Axis Accelerometer

This tutorial covers using the MPU6050 sensor with the Raspberry Pi Pico W to measure pitch and roll angles:

- **Setup:**
  - Connect the MPU6050 to the Raspberry Pi Pico W using the provided schematic and code.
- **Trigonometry for Tilt Measurement:**
  - Use trigonometric functions to calculate pitch and roll from accelerometer data.
- **Coding:**
  - Implement and test the calculations in Python to ensure accurate pitch and roll measurements.
- **Homework:**
  - Find an orientation where X, Y, and Z accelerations read 0G simultaneously and share the solution.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.44 lesson 44 : Measure Simultaneous 0 G on All Three Axis of MPU6050

This tutorial covers using the MPU6050 sensor with the Raspberry Pi Pico W to measure pitch, roll, and vertical displacement:

- **Setup:**
  - Connect the MPU6050 to the Raspberry Pi Pico W using the provided schematic.
- **Measuring Zero G:**
  - Determine conditions where all accelerometer axes read zero G simultaneously by dropping the sensor.
- **Homework:**
  - Create a method to determine the height from which the sensor was dropped by calculating the time it was in freefall and using the acceleration of gravity.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.45 lesson 45 : Calculating Height from a Dropped Object in Freefall

This tutorial covers using the MPU6050 sensor with the Raspberry Pi Pico W to measure vertical distances:

- **Setup:**
  - Connect the MPU6050 and OLED 1306 to the Raspberry Pi Pico W using the provided schematic. Ensure all connections are secure to avoid electrical noise.
- **Concept:**
  - Measure vertical distances by dropping the sensor and calculating the time it takes to fall (T_drop). Use the time in freefall to calculate the height dropped.
- **Equation:**
  - The height (H) is calculated using the formula: (H = 16 times (T_{drop})^2). Convert the drop time from milliseconds to seconds before applying the formula.
- **Code Implementation:**
  - Set up the MPU6050 and OLED 1306 libraries.
  - Measure the Z-axis acceleration to detect when the sensor is in freefall (measuring 0G).
  - Start a timer when the sensor is dropped and stop it when it hits the ground.
  - Display the calculated height and drop time on the OLED.
- **Practical Demonstration:**
  - Test the setup by dropping the sensor from known heights and verifying the accuracy of the measurements. Adjust as needed to improve precision.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.

- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.46 lesson 46 : Build a 2 Axis Tilt Meter with Display Using the MPU6050

This tutorial covers using the MPU6050 sensor with the Raspberry Pi Pico W to create a two-axis tilt meter:

- **Setup:**
  - Connect the MPU6050 and OLED 1306 to the Raspberry Pi Pico W using the provided schematic. Ensure all connections are secure to avoid electrical noise.
- **Concept:**
  - Measure the tilt of the sensor using the MPU6050's accelerometer data to calculate pitch and roll angles. Use these angles to create a visual representation of a bubble level on the OLED display.
- **Equation:**
  - Calculate pitch and roll angles using the following formulas: - Pitch: (text{Pitch} = arctanleft(frac{text{Y acceleration}}{text{Z acceleration}}right)) - Roll: (text{Roll} = arctanleft(frac{text{X acceleration}}{text{Z acceleration}}right))
  - Convert these angles from radians to degrees.
- **Code Implementation:**
  - Set up the MPU6050 and OLED 1306 libraries.
  - Measure the X, Y, and Z acceleration values.
  - Calculate pitch and roll angles in degrees.
  - Display the calculated pitch and roll angles on the OLED along with a visual representation of a bubble moving within a rectangle to indicate tilt.
- **Practical Demonstration:**
  - Test the setup by tilting the sensor and observing the bubble's movement on the OLED display. Adjust the sensitivity of the bubble's movement to make the tilt meter more responsive.
  - Ensure the readings are accurate and the bubble moves smoothly according to the tilt angles.
- **Advanced Considerations:**
  - Address the challenge of the accelerometer interpreting acceleration and deceleration as tilt. Think about strategies to stabilize the readings and prevent incorrect tilt detection due to vibrations or movements.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.47 lesson 47 : Improving Sensor Data With a Low Pass Filter

This tutorial covers using the MPU6050 sensor with the Raspberry Pi Pico W to create a stable two-axis tilt meter by implementing a low-pass filter:

- **Setup:**
  - Connect the MPU6050 to the Raspberry Pi Pico W using the provided schematic.
- **Concept:**
  - Measure tilt using the MPU6050's accelerometer data to calculate pitch and roll angles.
  - Address errors caused by interpreting acceleration as tilt.
- **Low-Pass Filter:**
  - Implement a low-pass filter to smooth out data and reduce noise.
  - Equation: (text{new value} = text{sensor confidence} times text{measurement} + (1 - text{sensor confidence}) times text{old value})
  - Adjust the confidence value for the best balance between responsiveness and noise reduction.
- **Code:**
  - Set up the MPU6050 to measure X, Y, and Z acceleration.
  - Calculate and filter pitch and roll angles.
  - Display the filtered values.
- **Homework:**
  - Implement and test the low-pass filter.
  - Experiment with different confidence values.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.

- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.48 lesson 48 : Measuring Rotation Using the Gyros on the MPU6050

This tutorial covers using the MPU6050 sensor with the Raspberry Pi Pico W to create a stable tilt meter by combining accelerometer and gyroscope data:

- **Setup:**
  - Connect the MPU6050 to the Raspberry Pi Pico W using the provided schematic.
- **Concept:**
  - Measure tilt using both accelerometer and gyroscope data.
  - Address errors from acceleration noise and gyroscope drift.
- **Low-Pass Filter:**
  - Smooth accelerometer data to reduce noise.
- **Gyroscope Integration:**
  - Calculate tilt angles using rotational velocity.
  - Update pitch, roll, and yaw angles continuously.
- **Combining Data:**
  - Fuse accelerometer and gyroscope data to minimize noise and drift.
- **Homework:**
  - Implement and fine-tune the described method for stable tilt measurement.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.49 lesson 49 : Improving IMU Performance with a Complimentary Filter

This tutorial covers improving tilt measurement accuracy using the MPU6050 sensor and Raspberry Pi Pico W:

- **Setup:**
  - Connect the MPU6050 to the Raspberry Pi Pico W using the provided schematic.
- **Challenges:**
  - Accelerometers alone are fast and accurate but noisy due to vibration and acceleration.
  - Gyroscopes alone are fast and low noise but suffer from drift over time.
- **Solution:**
  - Combine accelerometer and gyroscope data using a complementary filter to get the best of both sensors.
  - Use a low-pass filter for accelerometer data to reduce noise.
  - Use gyroscope data for short-term accuracy and accelerometer data for long-term stability.
- **Implementation:**
  - Calculate roll and pitch from both the accelerometer and gyroscope.
  - Blend these values using the complementary filter to achieve accurate, fast, and low-noise measurements.
- **Results:**
  - The complementary filter provides accurate and responsive tilt measurements with minimal noise and drift.
- **Homework:**
  - Implement the described method to achieve stable tilt measurement.
  - Explore ways to eliminate any remaining steady-state error in the measurements.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.50 lesson 50 : Removing Long Term Steady State Error from Sensor Data

This tutorial covers improving tilt measurement accuracy using the MPU6050 sensor and Raspberry Pi Pico W:

- **Setup:**
  - Connect the MPU6050 to the Raspberry Pi Pico W using the provided schematic.
- **Challenges:**
  - Accelerometers alone are noisy.
  - Gyroscopes alone drift over time.
- **Solution:**
  - Combine accelerometer and gyroscope data using a complementary filter.
  - Use a low-pass filter for accelerometer data to reduce noise.
  - Use gyroscope data for short-term accuracy.
  - Add error correction to handle steady state errors.
- **Results:**
  - Achieve accurate, fast, and low-noise tilt measurements.
- **Homework:**
  - Implement the filter and error correction.
  - Create a visual display of tilt using an OLED screen.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.51 lesson 51 : Ultimate Pitch and Roll Gadget Using the MPU6050

This tutorial covers creating a precise tilt meter using the MPU6050 sensor and Raspberry Pi Pico W:

- **Setup:**
  - Connect the MPU6050 and OLED 1306 to the Raspberry Pi Pico W using the provided schematic.
- **Challenges:**
  - Raw accelerometer data is noisy.
  - Gyroscope data drifts over time.
- **Solution:**
  - Combine accelerometer and gyroscope data using a complementary filter to achieve accurate, fast, and low-noise tilt measurements.
  - Implement error correction to handle steady state errors.
- **Implementation:**
  - Initialize the MPU6050 and OLED 1306.
  - Collect data from both accelerometer and gyroscope.
  - Apply a complementary filter to combine short-term gyroscope data with long-term accelerometer data.
  - Add error correction to compensate for any drift in the measurements.
  - Display the results on the OLED screen, showing both qualitative (bubble level) and quantitative (degree readout) tilt information.
- **Demonstration:**
  - The tilt meter is tested to show stable and accurate pitch and roll readings even with vibrations.
  - The device is made portable using a battery pack, enabling it to operate untethered.
- **Additional Improvements:**
  - Suggestions include making the device wireless for remote monitoring or designing a 3D printed case for portability.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.52 lesson 52 : Controlling Neopixels in MicroPython

This tutorial covers controlling NeoPixel LED strips using the Raspberry Pi Pico W:

- **Introduction:**
  - Overview of NeoPixel strip with 8 LEDs and simple wiring (5V, GND, Data).
- **Concept:**
  - Explanation of power and data flow through the NeoPixel strip.
- **Programming:**
  - Import libraries: *neopixel* and *machine*.
  - Set up NeoPixel object, define colors, and demonstrate controlling individual pixels.
  - Introduce brightness scaling.
- **Demonstration:**
  - Show different colors on the NeoPixel strip.
  - Display a pattern with a running green light over a red background.
- **Conclusion:**
  - Encourage experimenting with patterns.
  - Preview upcoming advanced NeoPixel projects.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.53 lesson 53 : Animating NeoPixel Strips Using MicroPython

This tutorial covers controlling a NeoPixel array with the Raspberry Pi Pico W using MicroPython:

- **Introduction:**
  - Overview of NeoPixel array with 8 LEDs.
  - Simple wiring: 5V, GND, and Data pin connected to GPIO pin 0.
- **Programming Basics:**
  - Import necessary libraries: *neopixel* and *machine*.

  - Setup NeoPixel object, define colors, and demonstrate controlling individual pixels.
- **Animating the NeoPixel Array:**
  - Code to create a running pixel animation on a colored background.
  - Explanation of forward and backward looping for smooth animation.
- **Advanced Animation:**
  - Homework assignment: Create a smooth rainbow color transition using the HSV color wheel.
  - Encourage experimenting with various animations and patterns.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.54 lesson 54 : Animated Dynamic Rainbow on NeoPixel in MicroPython

This tutorial covers controlling a NeoPixel array with the Raspberry Pi Pico W using MicroPython:

- **Introduction:**
  - Overview of NeoPixel array with 8 LEDs.
  - Simple wiring: 5V, GND, and Data pin connected to GPIO pin 0.
- **Programming Basics:**
  - Import necessary libraries: *neopixel* and *machine*.
  - Setup NeoPixel object, define colors, and demonstrate controlling individual pixels.
- **Animating the NeoPixel Array:**
  - Code to create a running pixel animation on a colored background.
  - Explanation of forward and backward looping for smooth animation.
- **Advanced Animation:**
  - Homework assignment: Create a smooth rainbow color transition using the HSV color wheel.
  - Encourage experimenting with various animations and patterns.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.55 lesson 55 : Create a Dynamic Rainbow With Neopixels in MicroPython

This tutorial covers creating a running rainbow pattern on a NeoPixel array using the Raspberry Pi Pico W and MicroPython:

- **Wiring Setup:**
  - Connect 5V, GND, and Data pin to GPIO pin 0 on the Pico W.
- **Concept Explanation:**
  - Create a running rainbow pattern by incrementing HSV color values and applying them to each pixel.
- **Code Implementation:**
  - Use a function to convert HSV to RGB.
  - Use nested loops to cycle through the HSV color wheel and update the NeoPixel array.
- **Demonstration and Testing:**
  - Show the running rainbow pattern on the NeoPixel array.
  - Provide troubleshooting tips.
- **Homework Assignment:**
  - Experiment with the code to create additional patterns.
  - Share results with the community.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.

- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.56 lesson 56 : Using a Joystick With MicroPython

This tutorial covers using a joystick with the Raspberry Pi Pico W:

- **Wiring Setup:**
  - Connect ground, 3.3V, VRX to GPIO pin 27, and VRY to GPIO pin 26.
- **Code Implementation:**
  - Import necessary libraries (*machine*, *time*, *math*).
  - Set up ADC for joystick axes.
  - Read and print joystick values to understand range and behavior.
- **Calibration:**
  - Adjust readings to be more intuitive, converting them to a scale from -100 to +100 for easier interpretation.
- **Homework Assignment:**
  - Write a program to calibrate the joystick so that the center position reads (0,0), and edges read ± 100.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.57 lesson 57 : Calibrating a Joystick in MicroPython

This tutorial covers calibrating a joystick with the Raspberry Pi Pico W:

- **Wiring Setup:**
  - Connect ground to physical pin 38, 3.3V to pin 36, VRX to GPIO pin 27, and VRY to GPIO pin 26.
- **Code Implementation:**
  - Import necessary libraries (*machine*, *time*, *math*).
  - Set up ADC for joystick axes.
  - Read and print joystick values for initial calibration.
- **Calibration:**
  - Calculate and apply calibration equations to convert raw ADC values to a scale from -100 to +100 for both axes.
  - Adjust for intuitive coordinate system and eliminate noise around the neutral position.
- **Homework Assignment:**
  - Write a program to calculate and report the angle of the joystick based on its position.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.58 lesson 58 : Determine Angular Position of a Joystick in MicroPython

This tutorial covers calibrating a joystick with the Raspberry Pi Pico W:

- **Wiring Setup**:Connect ground to pin 38, 3.3V to pin 36, VRX to GPIO pin 27, VRY to GPIO pin 26.
- **Code Implementation**:Import necessary libraries.Set up ADC for joystick axes and read values for calibration.
- **Calibration**:Convert raw ADC values to a scale from -100 to +100.Use trigonometry to calculate the joystick's angle.
- **Homework Assignment**:Write a program to control a servo motor based on the joystick's angle, ensuring accurate tracking between 0 and 180 degrees.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.59 lesson 59 : Controlling a Servo with a Joystick

This tutorial covers controlling a servo with a joystick using the Raspberry Pi Pico W:

- **Wiring Setup**:Connect joystick ground to pin 38, 3.3V to pin 36, VRX to GPIO pin 27, VRY to GPIO pin 26.Connect servo 5V to pin 40, ground to pin 38, and control to GPIO pin 15.
- **Code Implementation**:Import necessary libraries (*machine*, *time*, *math*).Set up ADC for joystick axes and PWM for the servo.Read and print joystick values for calibration and angle calculation.
- **Calibration and Control**: Convert raw ADC values to a scale from -100 to +100.Calculate the joystick's angle using trigonometry.Map the angle to the appropriate PWM value for the servo.
- **Homework Assignment**:Write a program to control a servo motor based on the joystick's angle, ensuring accurate tracking between 0 and 180 degrees.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.60 lesson 60 : Control NeoPixel Colors with a Joystick in MicroPython

This tutorial covers controlling an LED strip with a joystick using the Raspberry Pi Pico W:

- **Wiring Setup**:- Connect joystick ground to pin 38, 3.3V to pin 36, VRX to GPIO pin 27, VRY to GPIO pin 26.Connect Neopixel ground to pin 38, 5V to pin 40, and data to GPIO pin 0.
- **Code Implementation**:Import necessary libraries (*machine*, *time*, *math*, *neopixel*).Set up ADC for joystick and Neopixel.Read joystick values and calculate angles.Convert angles to RGB values for the Neopixel.
- **Homework Assignment**:Write a program to control the Neopixel color and brightness based on the joystick's angle and distance from the center.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.61 lesson 61 : How to Write Modular Code With Micropython Functions

This tutorial covers using functions in MicroPython on the Raspberry Pi Pico W:

- **Concept Overview**:Introduction to modular programming using functions.
- **Example Implementation**:Calculate the perimeter, area, and diagonal of a rectangle with functions.Define and call functions in MicroPython.Use return values to pass results from functions to the main program.
- **Steps**:1. Define the main function to call sub-functions.2. Create functions to calculate the area, perimeter, and diagonal.3. Call these functions from the main program and display results.
- **Homework Assignment**:Write a program to input grades, find the highest and lowest grades, calculate the average, and sort the grades.Use functions to handle each specific task.
- **Key Takeaways**:Understand modular programming and its benefits.Gain experience in defining and using functions in MicroPython.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.62 lesson 62 : Create Functions to Average and Sort Grades in Micropythons

This tutorial covers using functions in MicroPython on the Raspberry Pi Pico W:

Task Overview:

- Input grades from the user.
- Print grades.
- Calculate the average grade.
- Find the highest and lowest grades.
- Sort grades from highest to lowest.

Key Steps:

- Define and call functions.
- Pass variables and arrays to functions.
- Implement a sorting algorithm.

- **Homework Assignment**:
- Write a program to input grades, calculate the average, find the highest and lowest grades, and sort them from highest to lowest using functions without pre-built libraries.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.63 lesson 63 : Object Oriented Programming in MicroPython with Classes and Methods

This tutorial covers object-oriented programming (OOP) with the Raspberry Pi Pico W:

Task Overview:

- Introduce modular programming with functions.
- Transition to object-oriented programming using classes and methods.
- Create a *Rectangle* class with methods to calculate area, perimeter, and diagonal.

- **Key Steps**:
- Define a *Rectangle* class.
- Use the *__init__* method to initialize object attributes.
- Implement methods within the class to perform calculations.
- Create instances (objects) of the *Rectangle* class.
- Modify object attributes and recalculate values using class methods.
- **Homework Assignment**:
- Write a program using the *Rectangle* class to create multiple rectangle objects, calculate their properties, modify attributes, and recalculate as needed.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.64 lesson 64 : Object Oriented Programming Example in MicroPython with LEDs

This tutorial covers object-oriented programming (OOP) with the Raspberry Pi Pico W, focusing on controlling LEDs:

- **Wiring Setup**:Connect a red LED to GPIO pin 15 and a green LED to GPIO pin 14, with 330-ohm resistors to the ground.
- **Class and Methods**:1. Define an *LED* class to manage LED objects.2. Initialize the LED with the *__init__* method, setting up the pin.3. Implement a *blink* method to control the blinking of the LED.

- **Code Implementation**:1. Import necessary libraries (*machine* and *time*).2. Create the *LED* class with the *__init__* and *blink* methods.3. Instantiate objects for the red and green LEDs.4. Use a loop to blink the LEDs according to specified parameters.
- **Homework Assignment**:Create a class for a Servo motor, allowing easy control of its position. The class should include methods for initializing the Servo and setting its angle. Review lesson 36 for details on working with Servo motors in MicroPython.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.65 lesson 65 : Create a Servo Class and Method in MicroPython

This tutorial covers creating a Servo class using object-oriented programming (OOP) with the Raspberry Pi Pico W:

- **Wiring Setup**:

Connect the red wire of the servo to physical pin 40 (3.3V), the brown wire to pin 38 (ground), and the orange control wire to GPIO pin 17. * **Class and Methods**:

- Define a *Servo* class to manage servo objects.
- Initialize the servo with the *__init__* method, setting up the PWM pin.
- Implement a *pos* method to control the servo's position.
- **Code Implementation**:
- Import necessary libraries (*machine* and *time*).
- Create the *Servo* class with the *__init__* and *pos* methods.
- Instantiate a servo object and control its position using the *pos* method.
- **Homework Assignment**:
- Review lesson 36 for details on working with servos. Create a Servo class that allows easy control of the servo's position by setting angles. Implement a method to move the servo based on user input.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.66 lesson 66 : Create Your Own Libraries in Micropython

This tutorial covers creating and using a Servo library with the Raspberry Pi Pico W:

- **Wiring Setup**:
- Connect the red wire of the servo to physical pin 40 (3.3V), the brown wire to pin 38 (ground), and the orange control wire to GPIO pin 17.
- **Class and Methods**:
- Define a *Servo* class to manage servo objects.
- Initialize the servo with the *__init__* method, setting up the PWM pin.
- Implement a *pos* method to control the servo's position.
- **Library Creation**:
- Write the *Servo* class code and save it as a library file (*ServoLib.py*).
- Create a *lib* directory on the Raspberry Pi Pico W and save the library file in this directory.
- **Code Implementation**:
- Import necessary libraries (*machine*, *time*, and *ServoLib*).
- Instantiate a servo object using the *Servo* class from the *ServoLib* library.
- Control the servo's position by calling the *pos* method with desired angles.
- **Homework Assignment**:
- Create a similar library for another component, like an LED, to streamline its usage in future projects.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.67 lesson 67 : Use Both Cores on Your Pi Pico with MicroPython

This tutorial covers using both cores of the Raspberry Pi Pico W:

- **Wiring Setup**:
- Connect a green LED to GPIO pin 14 with a 330-ohm resistor to ground.
- Connect a red LED to GPIO pin 15 with a 330-ohm resistor to ground.
- **Code Implementation**:
- Import necessary libraries (*machine*, *time*, *_thread*).
- Set up pins for the LEDs.
- Define parameters for LED blink times.
- **Create functions to control LED blinking:**
  - *other_core* for the red LED on the second core.
  - *green_blink* for the green LED on the main core.
- Use *_thread.start_new_thread* to run *other_core* on the second core.
- **Homework Assignment**:
- Connect a servo.
- Control the servo and LEDs: - Blink red LED when servo moves backward. - Blink green LED when servo moves forward.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.68 lesson 68 : MicroPython Multicore Threading Example with LEDs and Servo

This tutorial covers controlling a servo and LEDs with the Raspberry Pi Pico W using both cores:

- **Wiring Setup**:
- Connect a red LED to GPIO pin 15 with a 330-ohm resistor to ground.
- Connect a green LED to GPIO pin 14 with a 330-ohm resistor to ground.
- Connect the servo control wire to GPIO pin 17, power wire to physical pin 40, and ground wire to physical pin 38.
- **Code Implementation**:
- Import necessary libraries (*machine*, *time*, *_thread*, *Servo*).
- Set up pins for the LEDs and servo.
- Define a function *other_core* to blink the LEDs based on the servo direction using a global variable.
- Create a loop to move the servo and set the LED direction.
- **Homework Assignment**:
- Extend the code to blink the red LED when the servo moves clockwise and the green LED when it moves counterclockwise.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.69 lesson 69 : Cleanly Exit MicroPython Threads When Program Terminates

This tutorial covers controlling a servo and LEDs with the Raspberry Pi Pico W using both cores:

- **Wiring Setup**:
- Connect a red LED to GPIO pin 15 with a 330-ohm resistor to ground.
- Connect a green LED to GPIO pin 14 with a 330-ohm resistor to ground.
- Connect the servo control wire to GPIO pin 17, power wire to physical pin 40, and ground wire to physical pin 38.

- **Code Implementation**:
- Import necessary libraries (*machine*, *time*, *_thread*, *Servo*).
- Set up pins for the LEDs and servo.
- Define a function *other_core* to blink the LEDs based on the servo direction using a global variable.
- Create a loop to move the servo and set the LED direction.
- **Homework Assignment**:
- Extend the code to blink the red LED when the servo moves clockwise and the green LED when it moves counterclockwise.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.70 lesson 70 : Example of Cleanly Exiting Dual Core Program in MicroPython

This tutorial covers using threading to control a servo and a button with the Raspberry Pi Pico W:

- **Wiring Setup**:
- Connect the servo's control wire to GPIO pin 17, power wire to physical pin 40, and ground wire to physical pin 38.
- Connect a button to GPIO pin 16 and ground.
- **Code Implementation**:
- Import necessary libraries (*machine*, *time*, *_thread*, *Servo*).
- Set up pins for the button and servo.
- Implement a toggle switch for the button to control the servo's movement.
- Define a main loop to detect button presses and toggle the servo between 0 and 180 degrees.
- Use threading to handle the servo movement on a separate core, allowing for clean program exits.
- **Homework Assignment**:
- Modify the program to ensure it exits cleanly even if interrupted while the servo is moving.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.71 lesson 71 : Allow Thread to Complete Task Before Termination

This tutorial covers gracefully terminating a multi-threaded program on the Raspberry Pi Pico W:

- **Wiring Setup**: - Connect the servo's control wire to GPIO pin 17, power wire to physical pin 40, and ground wire to physical pin 38. - Connect a button to GPIO pin 16 and ground.
- **Code Implementation**: - Import necessary libraries (*machine*, *time*, *_thread*, *Servo*). - Set up pins for the button and servo. - Implement a toggle switch for the button to control the servo's movement. - Define a main loop to detect button presses and toggle the servo between 0 and 180 degrees. - Use threading to handle the servo movement on a separate core, allowing for clean program exits.
- **Handling Clean Termination**: - Use a global *running* variable to control the loop execution. - Implement a baton (lock) to ensure only one part of the program runs at a time during critical sections. - Wait for the servo movement to complete before terminating the program.
- **Homework Assignment**: - Modify the program to handle additional components or sensors, ensuring clean termination in all scenarios.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.72 lesson 72 : Understanding Using Dictionaries in MicroPython

This tutorial covers using dictionaries in Python with the Raspberry Pi Pico W:

- **Introduction**: - Learn about dictionaries, a powerful data structure in Python. - Understand how dictionaries fit into object-oriented programming with classes, methods, and functions.
- **Concepts Covered**: - Review of Lists:
  - Create and manipulate lists (arrays).Access elements using indices.
  - Introduction to Dictionaries:
  - Create dictionaries with key-value pairs.Access dictionary elements using keys.Add, remove, and modify dictionary elements.
  - Dictionary Operations:
  - Use *keys()* and *values()* methods to retrieve keys and values.Check if a key exists using *in* and *not in* operators.Iterate through keys and values in a dictionary.
  - Practical Example:
  - Create a dictionary to store personal information (name, age, weight, etc.).- Demonstrate adding, modifying, and accessing elements in the dictionary.
  - Advanced Dictionary Use:
  - Store lists and other dictionaries as values within a dictionary.Utilize dictionaries for efficient data management and retrieval.
- **Homework Assignment:**
  - Write a program to control an RGB LED using a dictionary to map color names to their respective RGB values.Implement a user interface to input color names and change the LED color accordingly.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.73 lesson 73 : Control a RGB LED Using a Dictionary in MicroPython

This tutorial covers controlling an RGB LED with the Raspberry Pi Pico W using dictionaries:

- **Wiring Setup**:
- Connect the RGB LED to the Raspberry Pi Pico W: - R leg to GPIO pin 14 through a 330 Ohm resistor. - G leg to GPIO pin 13. - B leg to GPIO pin 12 through a 330 Ohm resistor. - Ground leg to the ground rail.
- **Code Implementation**:
- **Create Dictionary:**
  - Define a dictionary with color names as keys and their respective RGB values as lists.
- **Import Libraries:**
  - Import necessary libraries (*machine*, *time*).
  - Set up PWM for the RGB LED pins.
- **Main Program Loop:**
  - Continuously prompt the user to input a desired color.
  - Convert input to lowercase and check if it's a valid color.
  - If valid, adjust the LED colors using PWM duty cycles based on the dictionary values.
  - Implement error handling for invalid inputs.
- **Function to Set Color:**
  - Define a function *make_color* that takes the desired color and sets the RGB LED accordingly using PWM.
- **Homework Assignment:**
  - Extend the program by moving the *make_color* function into a library and importing it into the main program.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

8.74 lesson 74 : Create a MicroPython Class for Controlling RGB LED

This tutorial covers creating a MicroPython library for controlling an RGB LED with the Raspberry Pi Pico W:

- **Wiring Setup**: - Connect the RGB LED to the Raspberry Pi Pico W: - R leg to GPIO pin 14 through a 330 Ohm resistor. - G leg to GPIO pin 13. - B leg to GPIO pin 12 through a 330 Ohm resistor. - Ground leg to the ground rail.
- **Code Implementation**:
- **Create Library**:Define a class *RGB_LED* in a MicroPython library to handle the RGB LED.Include a dictionary for color values within the class.Create methods for initializing the LED pins and setting the colors using PWM.
- **Import Libraries**:Import necessary libraries (*machine*, *time*).Set up PWM for the RGB LED pins within the class methods.
- **Main Program**:Import the custom library and create an object for the RGB LED.Use a while loop to continuously prompt the user to input a desired color.Validate the input and set the LED color accordingly.Implement error handling for invalid inputs and clean exit functionality using a keyboard interrupt.
- **Homework Assignment**:Extend the program by adding more features to the MicroPython library, such as additional color settings or patterns, and ensure that the main program remains simple and easy to read by utilizing the library effectively.

Video

Note: Hello, welcome to the SunFounder Raspberry Pi & Arduino & ESP32 Enthusiasts Community on Facebook! Dive deeper into Raspberry Pi, Arduino, and ESP32 with fellow enthusiasts.

Why Join?

- **Expert Support**: Solve post-sale issues and technical challenges with help from our community and team.
- **Learn & Share**: Exchange tips and tutorials to enhance your skills.
- **Exclusive Previews**: Get early access to new product announcements and sneak peeks.
- **Special Discounts**: Enjoy exclusive discounts on our newest products.
- **Festive Promotions and Giveaways**: Take part in giveaways and holiday promotions.

Ready to explore and create with us? Click [] and join today!

CHAPTER

NINE

FAQ

9.1 Arduino

1. **Code upload failed in Arduino IDE?**
    - Check that your Pico is correctly recognised by the Arduino IDE, the port should be COMXX (Raspberry Pi Pico), for instructions please refer to *1.3 Setting Up the Raspberry Pi Pico W (Important)*.
    - Check that the Board(Raspberry Pi Pico) or port(COMXX (Raspberry Pi Pico))is selected correctly.
    - If your code is OK and you have selected the correct board and port, but the upload is still not successful. At this point you can click on the **Upload** icon again, when the progress below shows "Upload…", unplug the USB cable, then press and hold the **BOOTSEL** button to plug it in and the code will be uploaded successfully.

9.2 MicroPython

1. **How to open and run the code?**
   For detailed tutorials, please refer to *Open and Run Code Directly*.
2. **How to upload library to Raspberry Pi Pico W**
   For detailed tutorials, please refer to *1.4 Upload the Libraries to Pico*.
3. **NO MicroPython(Raspberry Pi Pico W) Interpreter Option on Thonny IDE?**
    - Check that your Pico W is plugged into your computer via a USB cable.
    - Check that you have installed MicroPython for Pico W (*1.3 Install MicroPython on Your Pico*).
    - The Raspberry Pi Pico W interpreter is only available in version 3.3.3 or higher version of Thonny. If you are running an older version, please update (*1.2 Install Thonny IDE*).
    - If the Li-po Charger module is plugged into the breadboard at this point, unplug it first and then re-plug the Pico W into the computer.
4. **Cannot open Pico W code or save code to Pico W via Thonny IDE?**
    - Check that your Pico W is plugged into your computer via a USB cable.
    - Check that you have selected the Interpreter as **MicroPython (Raspberry Pi Pico)**.
5. **Can Raspberry Pi Pico W be used on Thonny and Arduino at the same time?**
   NO, you need to do some different operations.
    - If you used it on Arduino first, and now you want to use it on Thonny IDE, you need to *1.3 Install MicroPython on Your Pico* on it.

- If you used it on Thonny first, and now you want to use it on Arduino IDE, you need to *1.3 Setting Up the Raspberry Pi Pico W (Important)*.

6. **If your computer is win7 and Pico W cannot be detected.**

    - Download the USB CDC driver from http://aem-origin.microchip.com/en-us/mindi-sw-library?swsearch=Atmel%2520USB%2520CDC%2520Virtual%2520COM%2520Driver
    - Unzip the `amtel_devices_cdc.inf` file to a folder named `pico-serial`.
    - Change the name of `amtel_devices_cdc.inf` file to `pico-serial.inf`.
    - Open/edit the `pico-serial.inf` in a basic editor like notepad
    - Remove and replace the lines under the following headings:

```
[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."
PI_CDC_PICO = "Pi Pico Serial Port"
Serial.SvcDesc = "Pi Pico Serial Driver"
```

    1. Close and save and make sure your retain the name as pico-serial.inf
    2. Go to your pc device list, find the pico under Ports, named something like CDC Device. A yellow exclamation mark indicates it.
    3. Right click on the CDC Device and update or install driver choosing the file you created from the location you saved it at.

9.3 Piper Make

1. **How to set up the Pico W on Piper Make?**
   For detailed tutorials, please refer to *1.1 Set up the Pico*.
2. **How to download or import code?**
   For detailed tutorials, please refer to *1.3 How to Save or Import Code?*.
3. **How to connect to Pico W?**
   For detailed tutorials, please refer to *2. Connect to Pico W*.

CHAPTER

TEN

COPYRIGHT NOTICE

All contents including but not limited to texts, images, and code in this manual are owned by the SunFounder Company. You should only use it for personal study,investigation, enjoyment, or other non-commercial or nonprofit purposes, under therelated regulations and copyrights laws, without infringing the legal rights of the author and relevant right holders. For any individual or organization that uses these for commercial profit without permission, the Company reserves the right to take legal action.