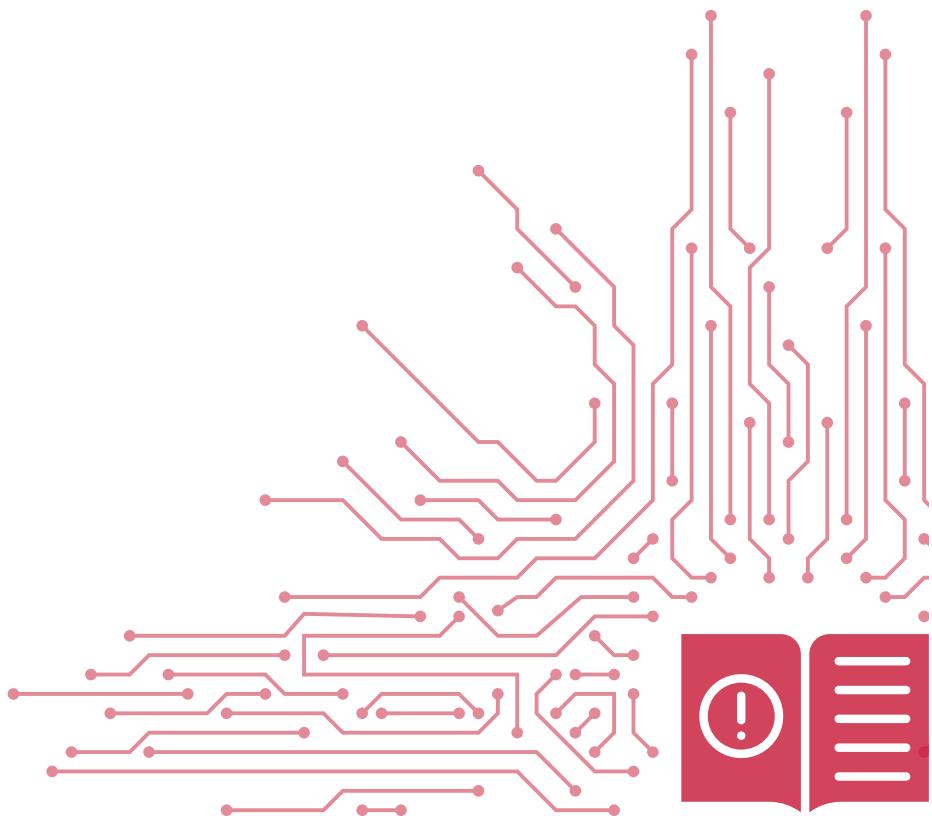


# Raspberry Pi Pico

## GUIDE



---

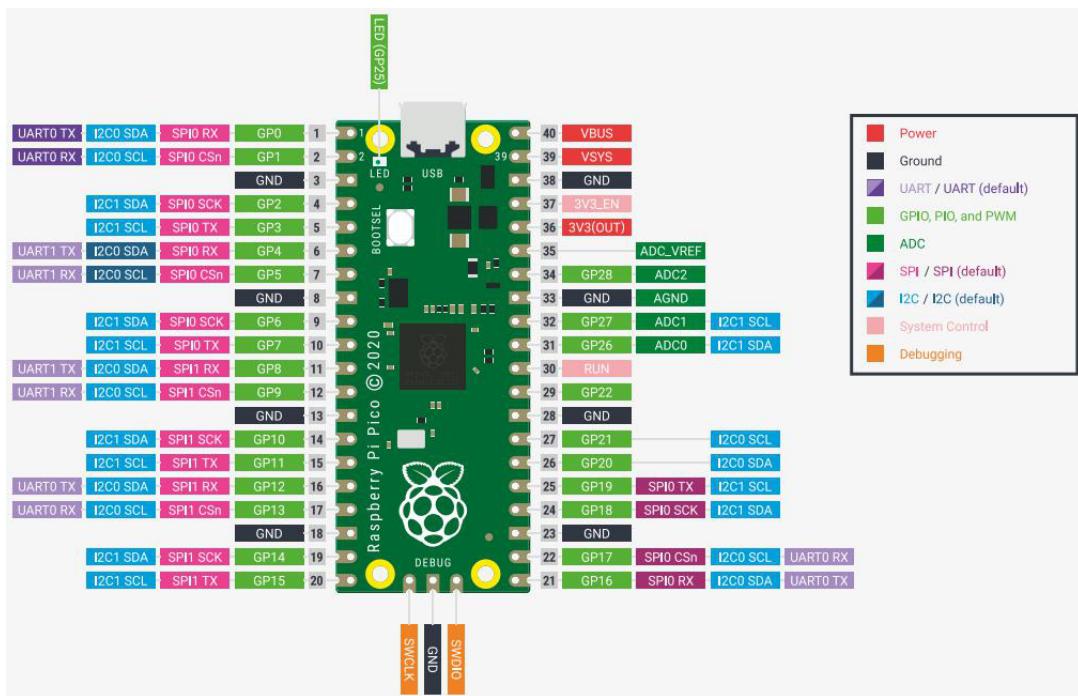
# CONTENTS

1. Learn about the Raspberry Pi Pico motherboard	EN-3
1.1 What do you need ?	EN-3
1.1.1 Raspberry Pi Pico motherboard	EN-3
1.1.2 Micro USB downloader	EN-4
1.1.3 Breadboard	EN-4
1.1.4 Jumper	EN-5
2. Raspberry Pi Pico Learning Readiness	EN-6
2.1 Introduction to the I/O mouth of Raspberry Pi Pico	EN-6
2.2 What is Micro Python programming	EN-6
2.2.1 Raspberry Pi Pico installs Micro Python support firmware	EN-7
2.3 Introduction to the Raspberry Pi Pico motherboard Micro Python library	EN-8
3 Use thonny software for Python development	EN-9
3.1 Select the right Raspberry Pi Pico programming	EN-13
3.2 Modify the font size	EN-14
3.3 Write the LED light flashing program using Thonny	EN-15
4 Case Programming	EN-19
4.1 Light-emitting diode experiments	EN-19
4.2 DC motor fan experiment	EN-20
4.3 LED strip	EN-21
4.4 Moisture Sensor	EN-22
4.5 Liquid Level Sensor	EN-23
4.6 UV Sensor	EN-24
4.7 Laser out Sensor	EN-25
4.8 Tilt Sensor	EN-26
4.9 Sound Sensor	EN-27
4.10 PIR Sensor	EN-28
4.11 Hall Sensor	EN-29
4.12 Gas Sensor	EN-30
4.13 Infrared Reflective Sensor	EN-31
4.14 Flame Sensor	EN-32
4.15 Temperature-Humidity Sensor	EN-33
4.16 Clock sensor	EN-34
4.17 Rotation Sensor	EN-36
4.18 Temperature Sensor	EN-37
4.19 Ultrasonic sensor	EN-38
4.20 Steering engine	EN-40
4.20 Steering engine	EN-40
4.21 Infrared remote control	EN-41
4.22 Color sensor	EN-42
4.23 Relay	EN-43

## 1. Learn about the Raspberry Pi Pico motherboard

On January 21, 2021, the Raspberry Pi Foundation released the Raspberry Pi Pico motherboard, a microprocessor product. The product is based on the Raspberry Pi Foundation's new, self-developed RP2040 processor, which costs just \$4. Raspberry Pi Pico is reported as a microprocessor, it specializes in low-latency I/O communication and analog signal input, low power consumption, can make up for the Raspberry Pi motherboard in the microprocessor field of the short board.

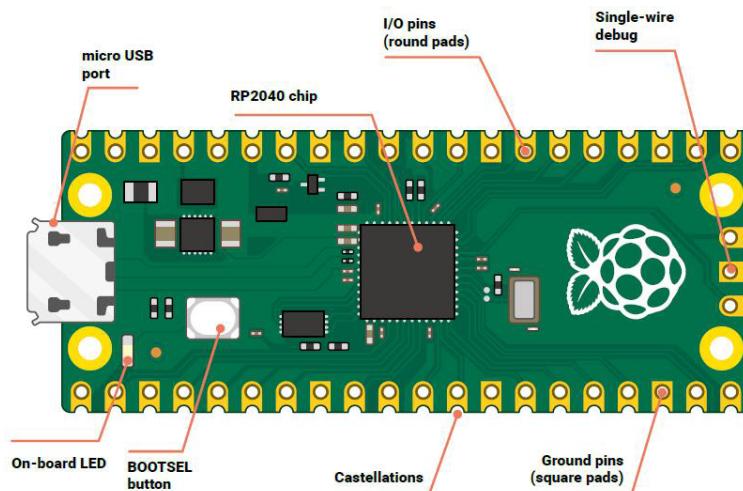
The Raspberry Pi Pico motherboard has become a new favorite of the world's founders, especially based on the new features of the currently hot Micro Python programming, allowing us to learn Python programming at a low cost entirely through the Raspberry Pi Pico motherboard.



## 1.1 What do you need ?

To build a complete Raspberry Pi Pico learning and development system, you need at least one Raspberry Pi Pico motherboard, Micro USB data cable (optional Pico Block sensor expansion board) and simple line-out IO port and breadboard, in order to better use Raspberry Pi Pico development system, you can also match accessories such as sensor modules, LCD monitors, keyboards and so on, let's go one by one:

### 1.1.1 Raspberry Pi Pico motherboard



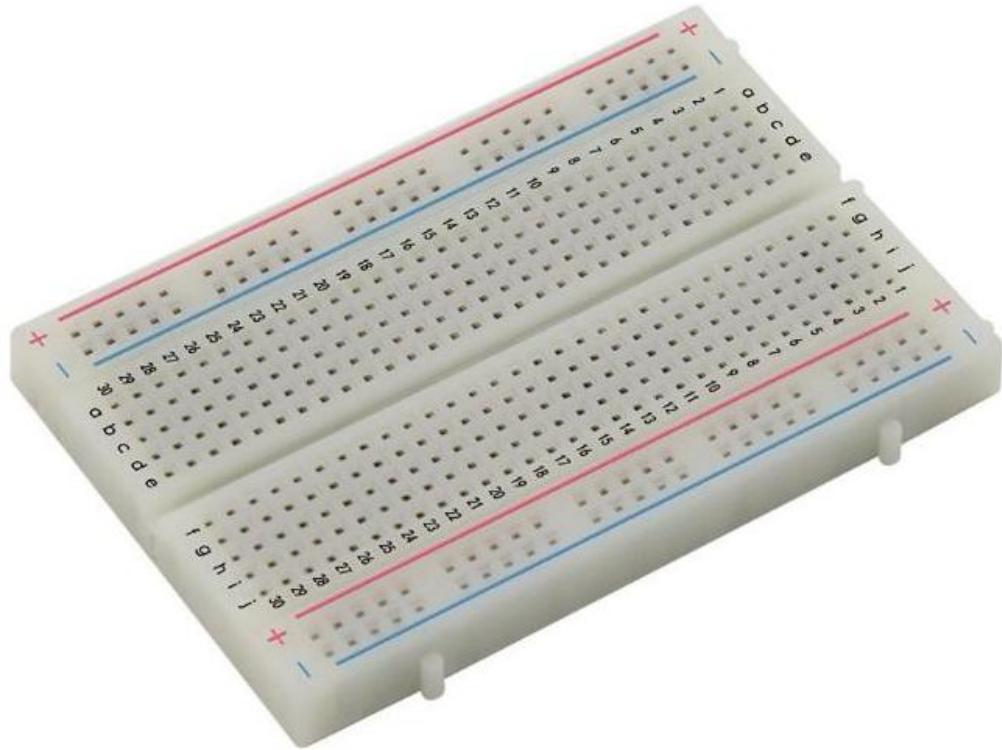
- ◆ CPU: 32-bit dual-core ARM Cortex-M0+ at 48MHz, configurable up to 133MHz
- ◆ RAM: 264kB of SRAM in six independently configurable banks
- ◆ Storage: 2MB external flash RAM
- ◆ GPIO: 26 pins
- ◆ ADC: 3 × 12-bit ADC pins
- ◆ PWM: Eight slices, two outputs per slice for 16 total
- ◆ Clock: Accurate on-chip clock and timer with year, month, day, day-of-week, hour, second, and automatic leap-year calculation
- ◆ Sensors: On-chip temperature sensor connected to 12-bit ADC channel
- ◆ LEDs: On-board user-addressable LED
- ◆ Bus Connectivity: 2 × UART, 2 × SPI, 2 × I2C, Programmable Input/Output (PIO)
- ◆ Hardware Debug: Single-Wire Debug (SWD)
- ◆ Mount Options: Through-hole and castellated pins (unpopulated) with 4 × mounting holes
- ◆ Power: 5 V via micro USB connector, 3.3 V via 3V3 pin, or 2–5V via VSYS pin

### 1.1.2 Micro USB downloader

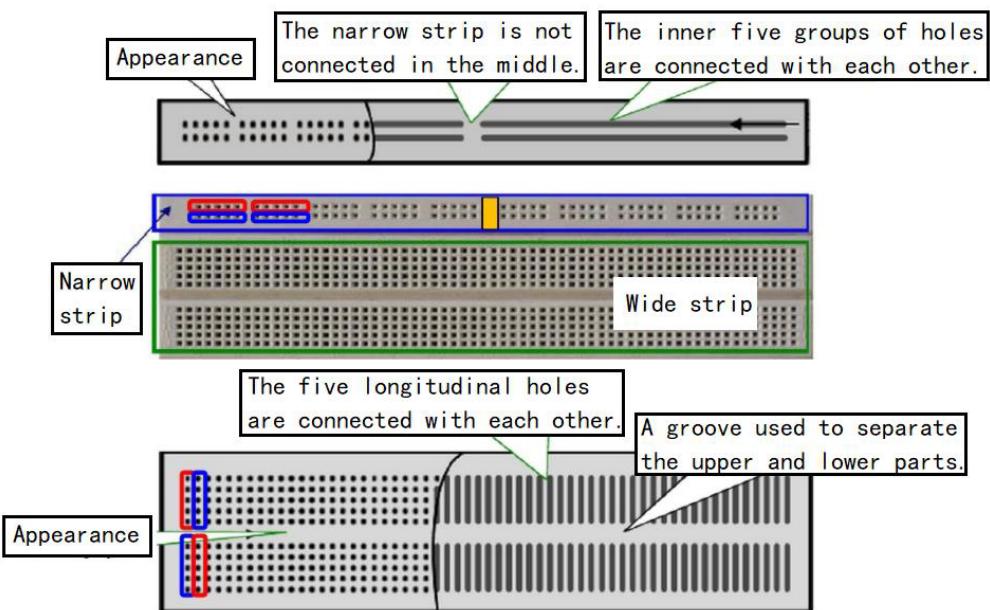
Raspberry Pi Pico motherboard program download and USB power interface for Micro USB interface, so we need to use a better quality Micro USB cable for program download, if the PC side when using, it is best not to use USB Hub extension, because it will lead to unstable download programs, recommended to use the USB with the PC side for program download and power supply, requires USB to provide a stable 5V, In subsequent sensor experiments, sufficient current is required to power the power supply.

### 1.1.3 Breadboard

The breadboard is made for non-welded experiments in electronic circuits because there are many small jacks on the board. Because a variety of electronic components can be inserted or unplugged as needed, eliminate welding, saving circuit assembly time, and components can be reused, so it is ideal for electronic circuit assembly, commissioning and training.

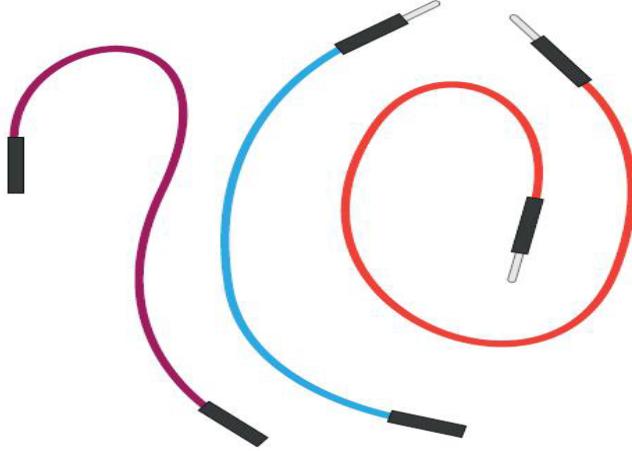


The internal structure of the breadboard



#### 1.1.4 Jumper

There are three types of jumpers: male-to-male, female-to-female, and male-to-female

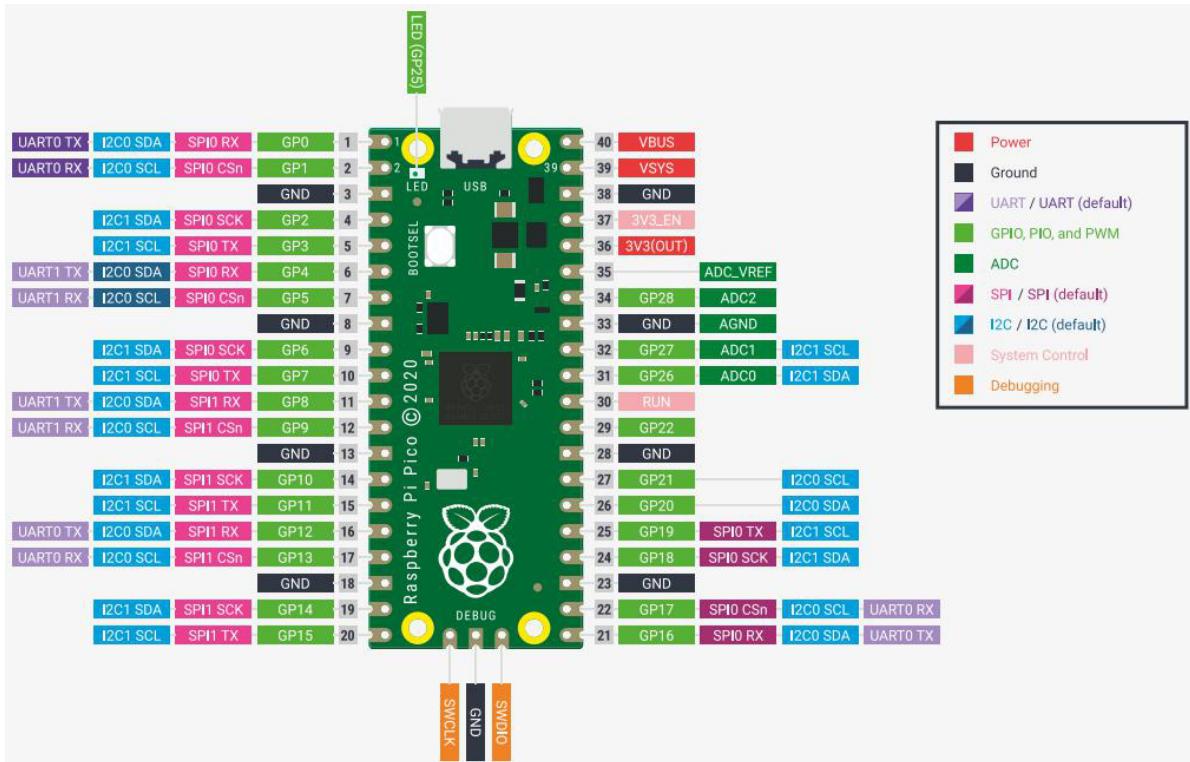


## 2. Raspberry Pi Pico Learning Readiness

The Raspberry Pi Pico motherboard, which is officially launched by the Raspberry Pico board, is programmed in the Python language and has a Micro python parser installed in the RP2040 processor, so we can program using the Python Pico motherboard, and the official programming method is also available in C/C++, an advanced player's choice that requires a lot of software to support it.

### 2.1 Introduction to the I/O mouth of Raspberry Pi Pico

Raspberry Pi Pico offers 40Pin ports for use, with 26 multifunction GPIO pins, 2 SPIs and 2 I2C controllers, 2 UART and 16 PWM control interfaces. 3 12-bit ADC analog acquisition interfaces. The pin definition is as follows:



### 2.2 What is Micro Python programming

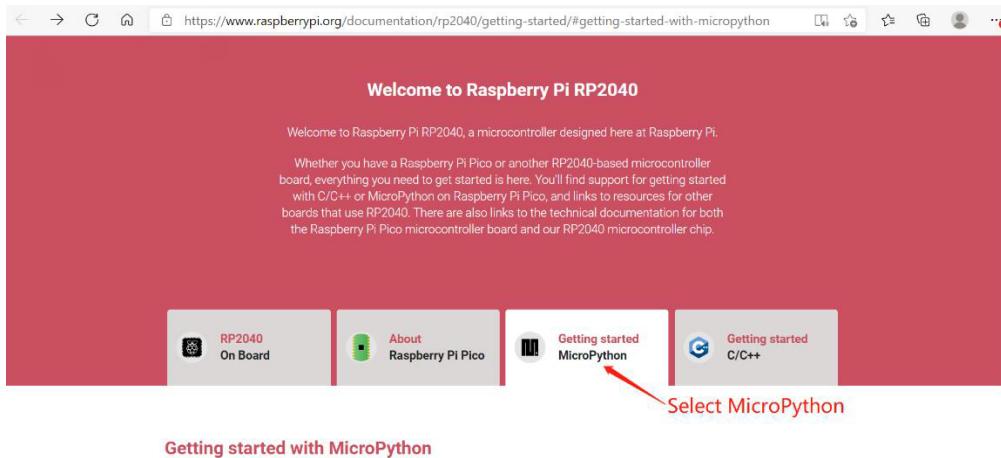
Micro Python is a complete software for the Python3 programming language, written in C and run on a microcontroller, and Micro Python is a full Python compiler and operating system running on top of microcontroller hardware. Provides the user with an interactive prompt (REPL) to

immediately execute the supported commands. In addition to the selected core Python library, Micro Python includes the underlying hardware modules that give programmers access.

Details can be viewed at: <https://www.micropython.org/>

### 2.2.1 Raspberry Pi Pico installs Micro Python support firmware

If the Raspberry Pi Pico motherboard supports Micro Python programming, we need to download the UF2 file on the official website and copy it to Pico's USB stick on the computer and go to the official website: <https://www.raspberrypi.org/documentation/rp2040/getting-started/>

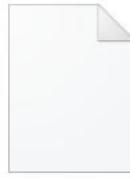


**Drag and drop MicroPython**

1

The screenshot shows the "Getting started with MicroPython" section of the Pico documentation. It features a red header bar with the title "Getting started with MicroPython". Below the header, there is a section titled "Drag and drop MicroPython" with instructions on how to program the Pico using a UF2 file. A green button labeled "Download UF2 file" is shown. A red arrow points from this button to a callout box labeled "Select download UF2 file to the local disk".

The following image is a obtained UF2 file, which may be officially updated, so the name will change:



rp2-pico-20210  
205-unstable-v  
1.14-8-g1f800c  
ac3.uf2

Connect the USB interface of the Pico motherboard with the Micro USB cable and the USB interface on the PC side, if the motherboard without burning firmware will produce a USB stick with the name: RPI-RP2;

**Step 1:** first hold down the "BOOTSEL", and then insert the computer side USB port;

**Step 2:** insert the computer end and release the button of "BOOTSEL", you can appear the name RPI-RP2 USB stick, and then the previously downloaded UF2 file copied into the USB stick, the USB stick will automatically disappear;

**Step 3:** Open PC Device Manager and we can find the virtual serial device in the port, as shown below:



**Note:**

If we need to confirm whether it is COM4 shown above, we can unplug the device to see if the port disappears, and if it does, it means that the device is our Pico motherboard and supports programming by Micro Python.

### 2.3 Introduction to the Raspberry Pi Pico motherboard Micro Python library

The Raspberry Pi Foundation provides a very detailed Python SDK PDF document that details the various built-in functions, as well as the specific usage of the corresponding functions and the case notes provided, which we need to use to query the various built-in function functions for implementation.

## Documentation

Documentation for Raspberry Pi Pico and other RP2040-based boards.

- **Raspberry Pi Pico Datasheet**

An RP2040-based microcontroller board

- **RP2040 Datasheet**

A microcontroller by Raspberry Pi

- **Hardware design with RP2040**

Using RP2040 microcontrollers to build boards and products

- **Getting started with Raspberry Pi Pico**

C/C++ development with Raspberry Pi Pico and other RP2040-based microcontroller boards

- **Raspberry Pi Pico C/C++ SDK**

Libraries and tools for C/C++ development on RP2040 microcontrollers

- **Raspberry Pi Pico Python SDK**

A MicroPython environment for RP2040 microcontrollers

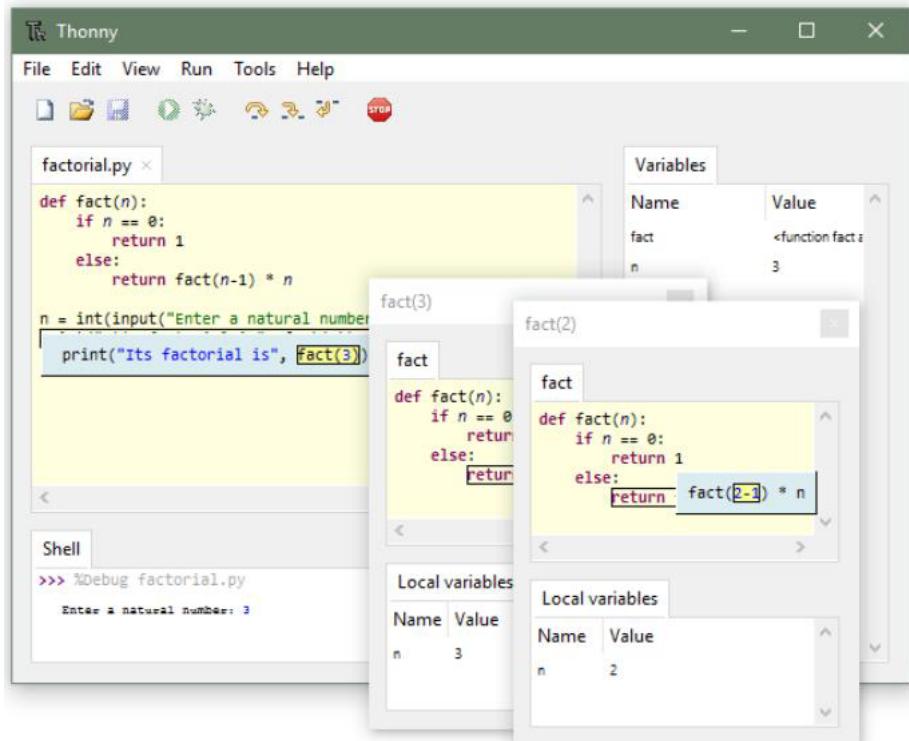
The API level Doxygen documentation for the Raspberry Pi Pico C/C++ SDK is available [as a micro-site](#), and frequent questions are answered in the [Frequently Asked Questions \(FAQ\)](#) document.

### 3 Use thonny software for Python development

Python programming has evolved to the present day with a number of very powerful IDE programming software, such as PyCharm IDE, Eclipse, Spyder, and Visual Studio Code, but for beginners, choosing a lightweight, easy-to-use IDE is the best option, and Thonny is what we call a small IDE, available on Windows, MAC, or Linux, it supports syntax coloring, code auto-completion, debug, and most importantly, the Python compiler on the Raspberry Pi Pico motherboard.

Official website: <https://thonny.org/>

Go to the website to download the latest Thonny software, as follows:

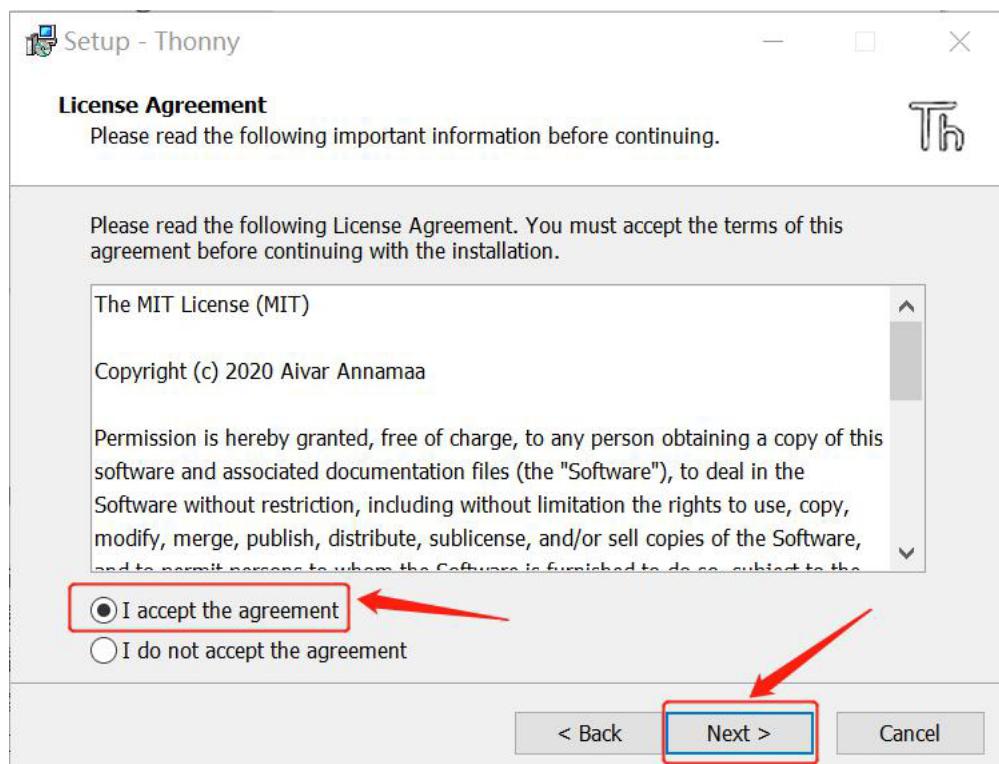
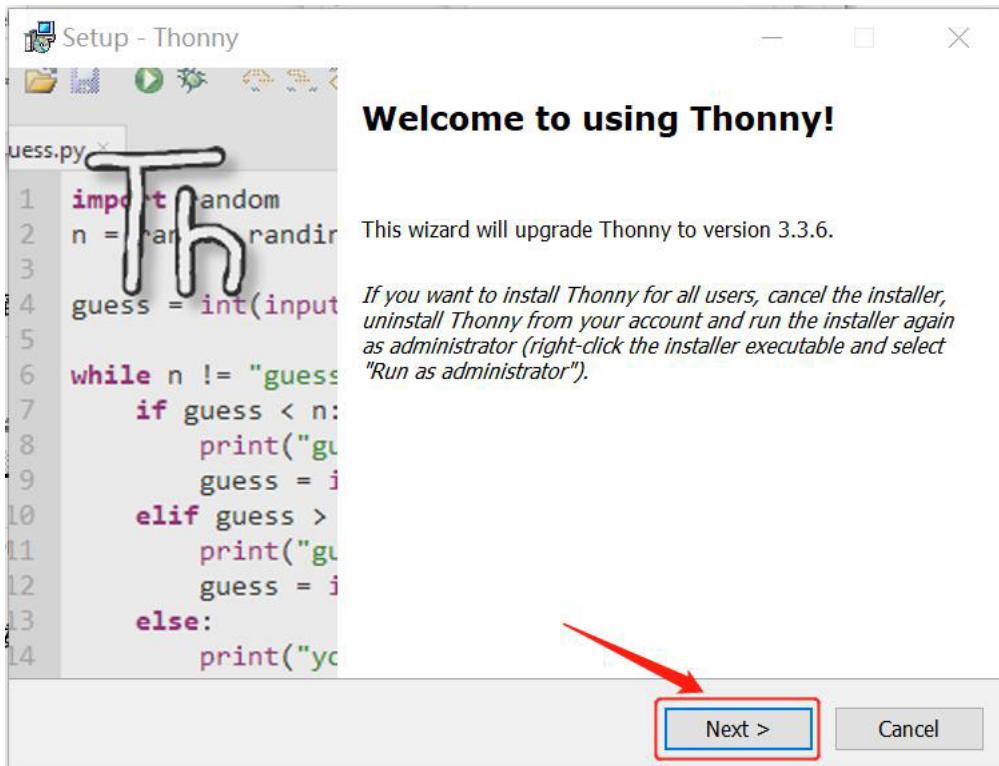


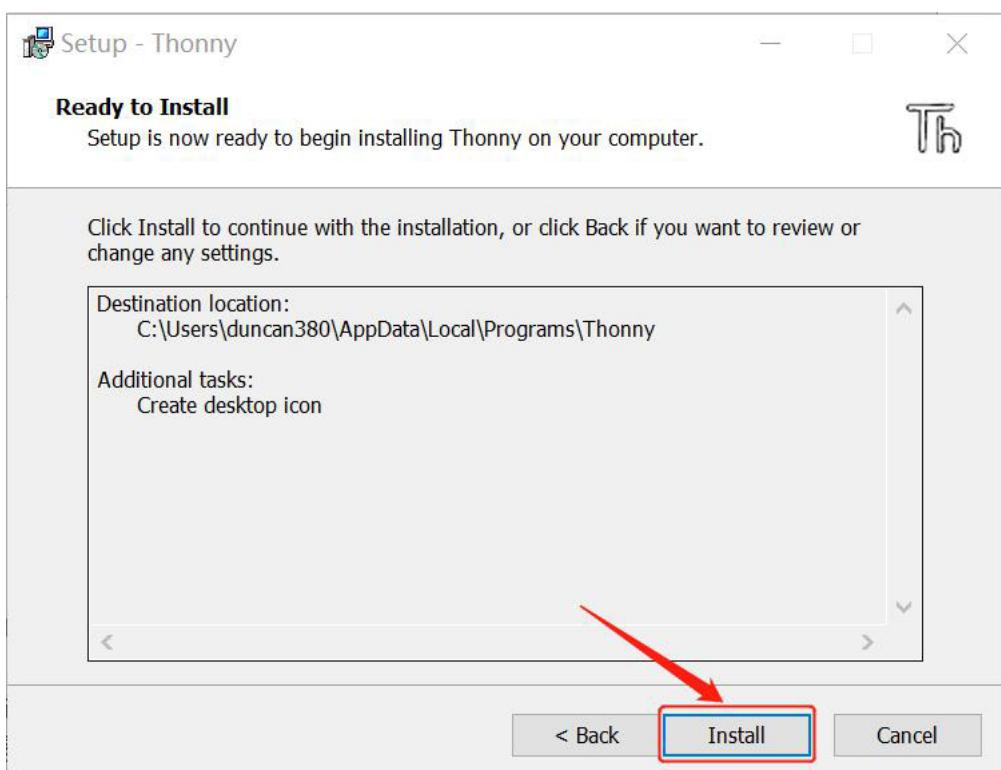
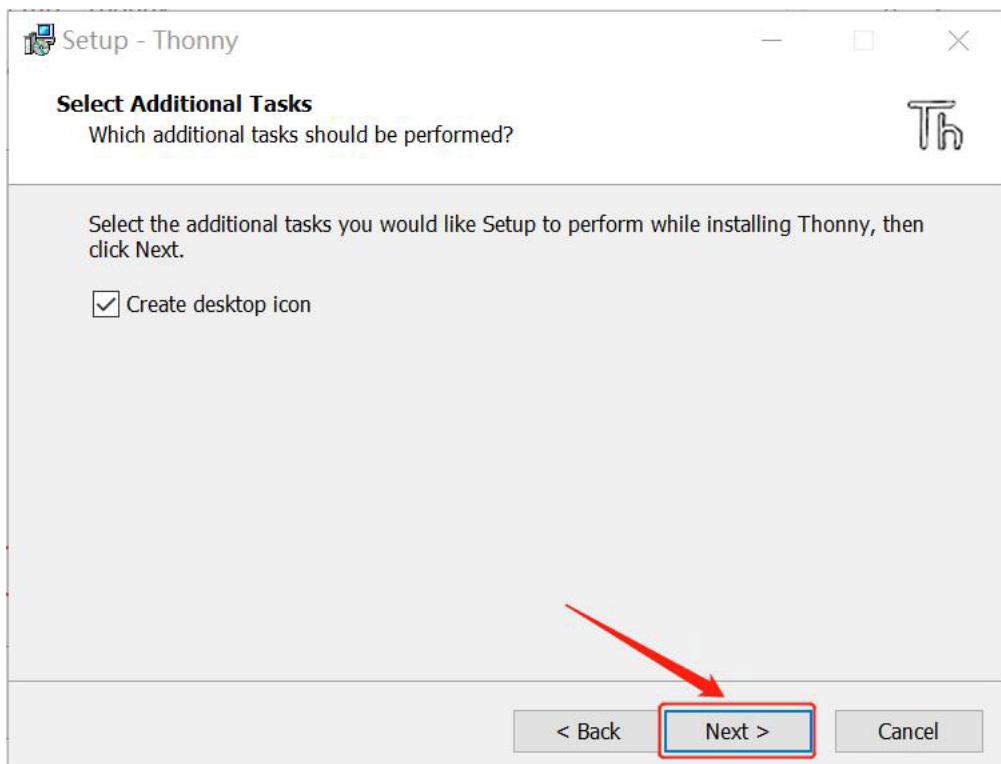
After downloading to the local area, select the corresponding installer, as shown below:

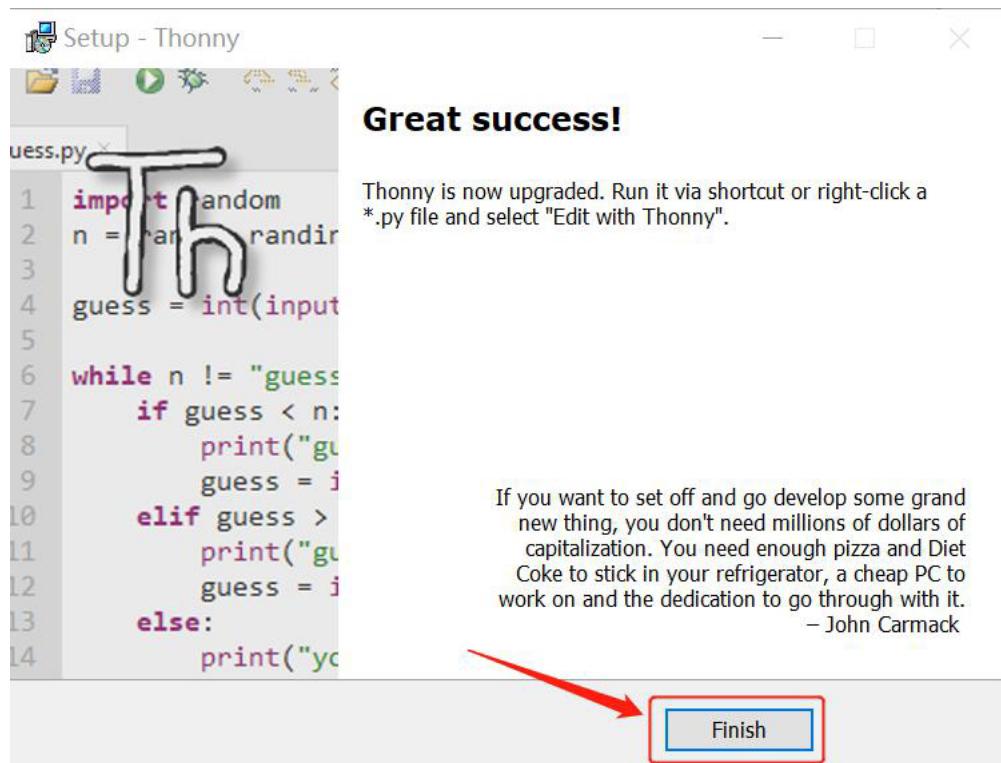
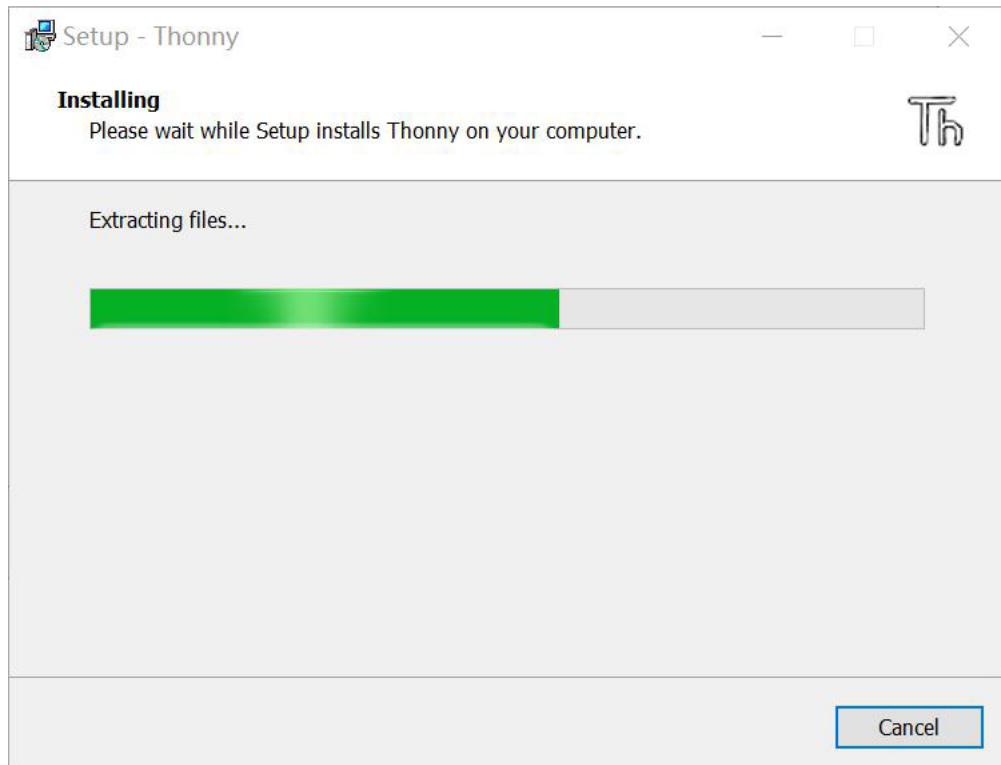


thonny-3.3.6

Double-click on the installer and pop up the following installation interface, point NEXT key:







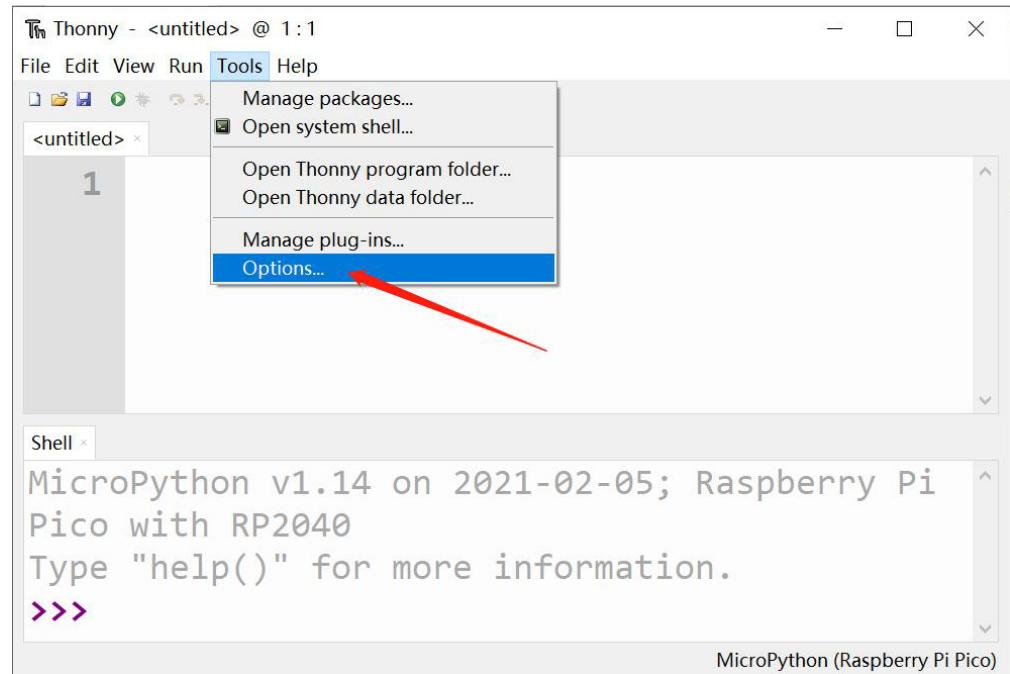
Double-click the following shortcut on the PC desktop to open the software, select the appropriate language, and enter the software interface:

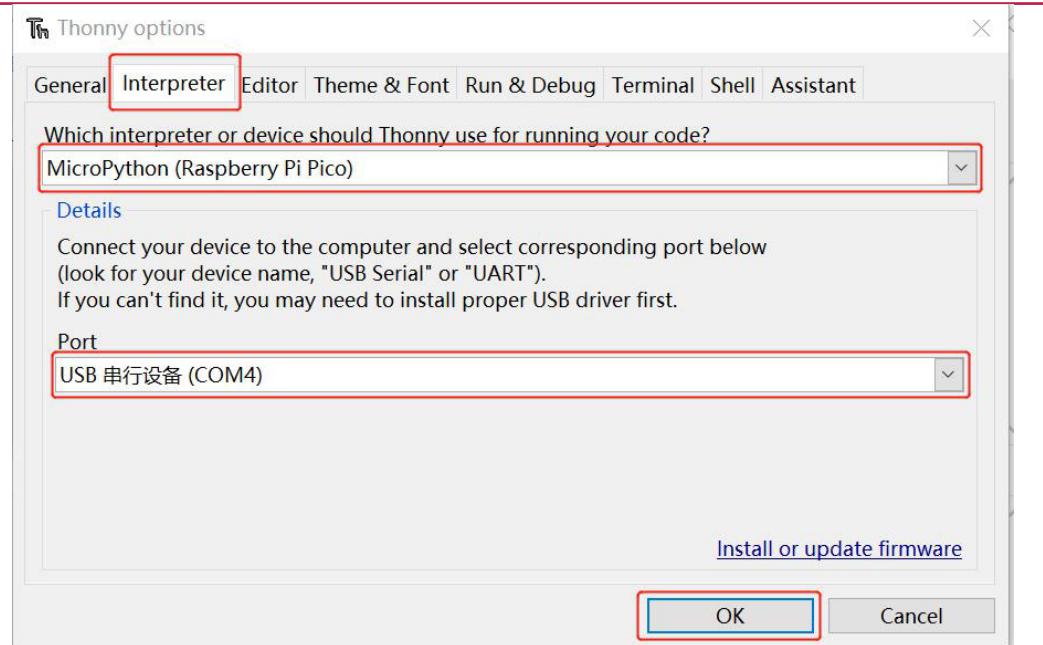


The screenshot shows the Thonny IDE interface. At the top is a menu bar with File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, and Run. A tab bar shows an untitled script. The main area has a code editor with the number '1' and a shell window below it. The shell window displays the MicroPython prompt: 'MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040'. It also says 'Type "help()" for more information.' and shows the '>>>' prompt. The status bar at the bottom right says 'MicroPython (Raspberry Pi Pico)'.

### 3.1 Select the right Raspberry Pi Pico programming

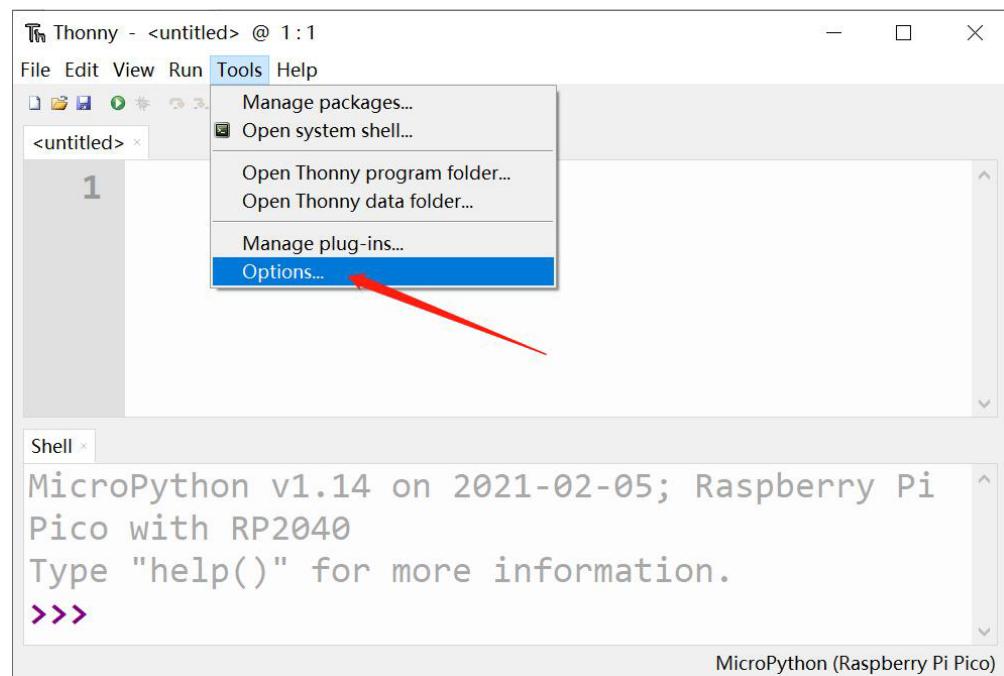
Select Tools/Settings in the menu bar and click on the pop-up settings window:

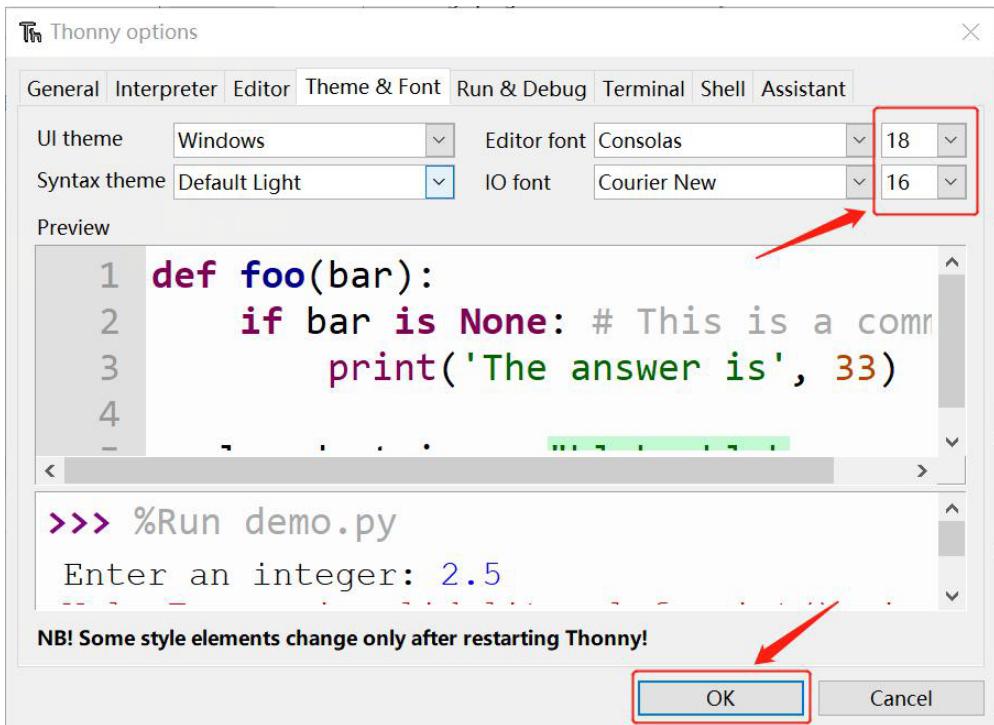




### 3.2 Modify the font size

Select Tools/Settings/Subjects and Addresses



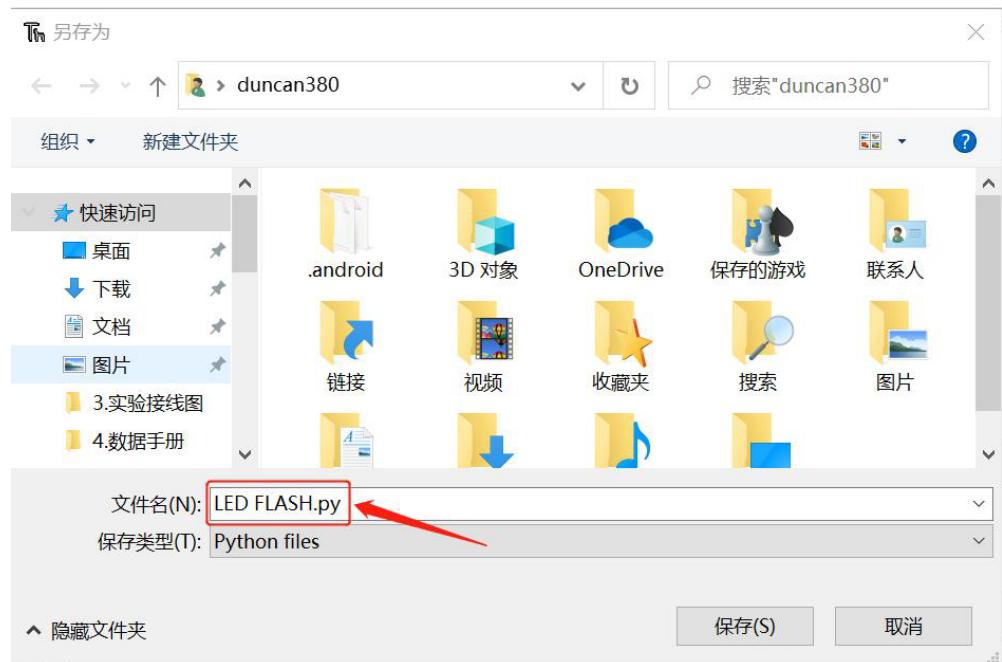
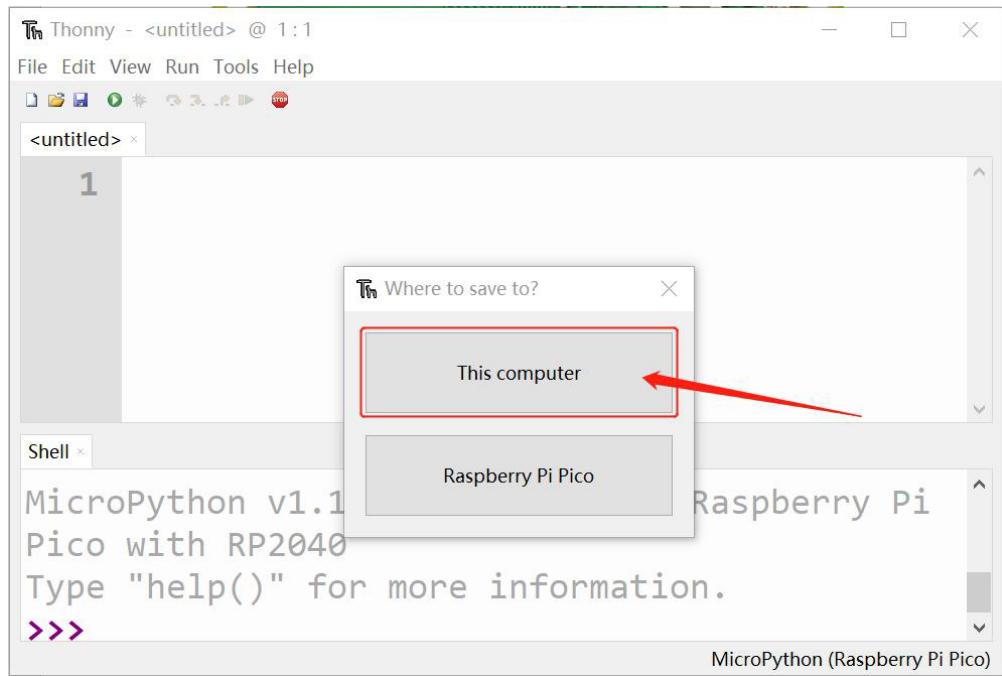


### 3.3 Write the LED light flashing program using Thonny

To light up the LED light you need to give a high level, turn off to give a low level, and now write a Thonny software that lets the LED light (GP25) on the Pico board flash, as shown below:



Step 1: Open the Thonny software and create a new file named “\*\*.py”



Step 2: After writing the following code, click the SAVE button:

The screenshot shows the Thonny IDE interface. The code editor window contains the following Python script:

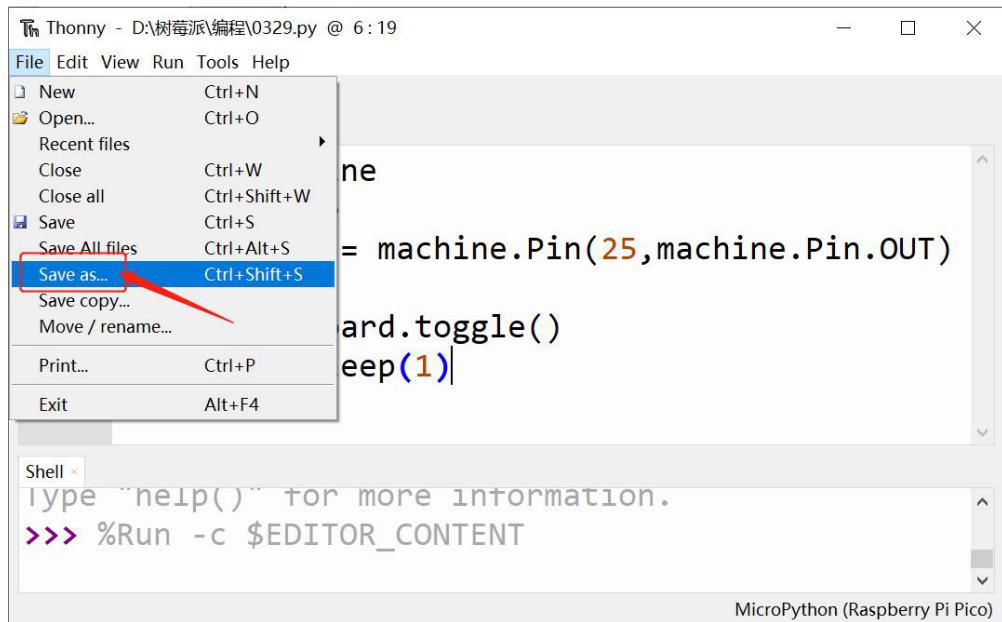
```
1 import machine
2 import utime
3 led_onboard = machine.Pin(25,machine.Pin.OUT)
4 while True:
5     led_onboard.toggle()
6     utime.sleep(1)
```

The 'Save' button in the toolbar (represented by a floppy disk icon) is highlighted with a red box. The shell window at the bottom shows the command `>>> %Run -c $EDITOR_CONTENT`.

Step 3: Click the RUN button to run the program, and the LED light will flash according to the program.

**Note:** The program saved according to the above method is on the computer, and if you unplug the USB at this time, the LED will not flash. If you want to cure the program to the Pico motherboard, you can automatically run the program as long as you power it on the device. Here's how:

All you need to do is select Save as and save as m in the file [main.py] menu



Thonny - D:\树莓派\编程\0329.py @ 6:19

File Edit View Run Tools Help

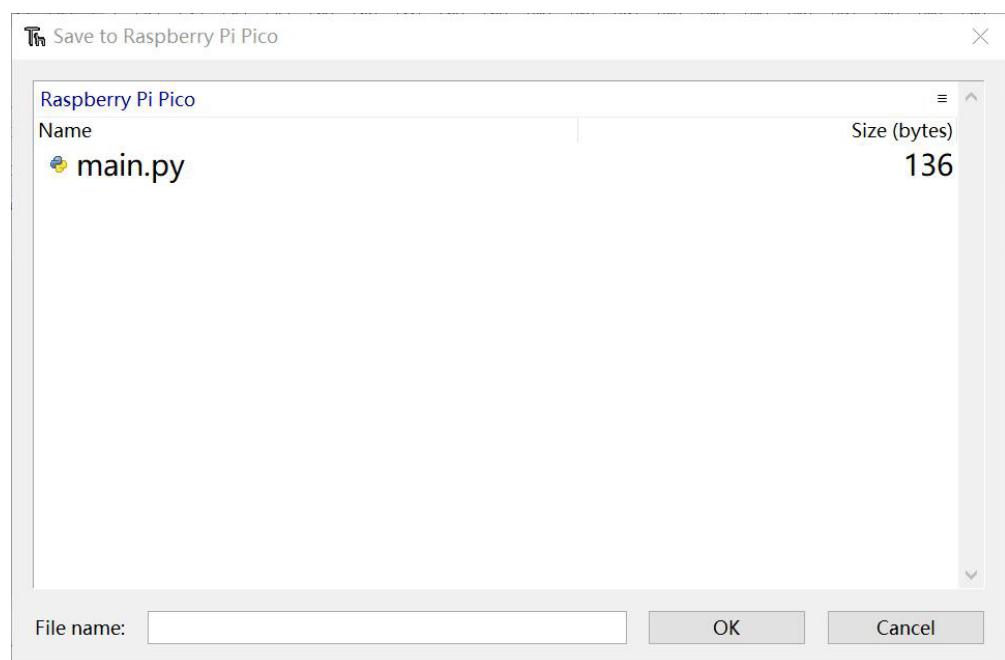
<untitled> x 0329.py \* x

```
1 import machine
2 import utime
3 led_onboard = machine.Pin(2, machine.Pin.OUT)
4 while True:
5     led_onboard.toggle()
6     utime.sleep(0.5)
```

Where to save to? This computer Raspberry Pi Pico

Shell x  
Type "help()" for more information.  
=> %Run -c \$EDITOR\_CONTENT

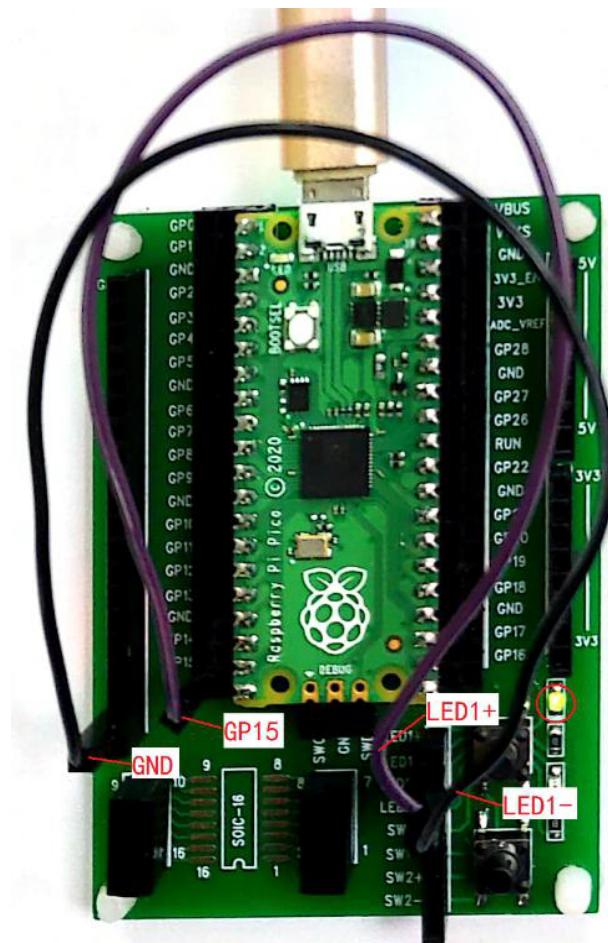
MicroPython (Raspberry Pi Pico)



## 4 Case Programming

### 4.1 Light-emitting diode experiments

The wiring is as follows:



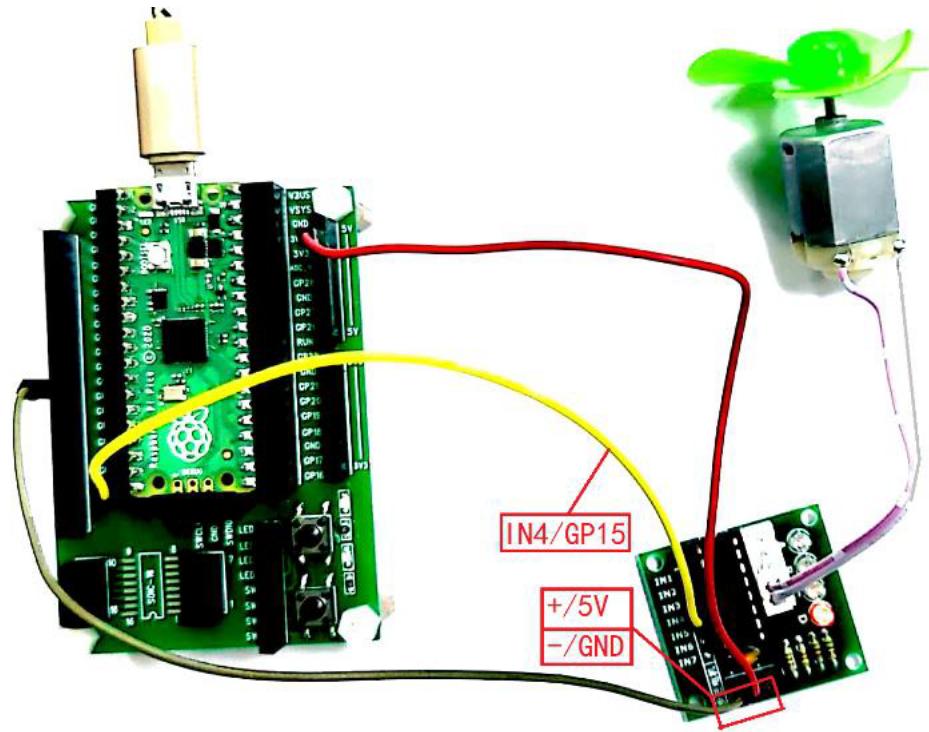
The program code is as follows:

```
Thonny - D:\树莓派\PiPico 学习资料2021-2-3\2.案例程序\2.发光二极管实验\2.LED.py @ 7:19
File Edit View Run Tools Help
2.LED.py x
1 import machine
2 import utime
3
4 led_external = machine.Pin(15, machine.Pin.OUT)
5 while True:
6     led_external.toggle()
7     utime.sleep(1)

Shell x
MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
MicroPython (Raspberry Pi Pico)
```

## 4.2 DC motor fan experiment

The wiring is as follows:



The program code is as follows:

```
Thonny - E:\Raspberry_pico\入门资料\2.案例程序\8.直流电机风扇实验\main.py @ 14 : 21
```

File Edit View Run Tools Help

Variables

Name
PWM
Pin
duty
machine
pwm
rp2
sleep

main.py x

```
1 from machine import Pin, PWM
2 from time import sleep
3
4 pwm = PWM(Pin(15))
5
6 pwm.freq(1000)
7
8 while True:
9     for duty in range(10,65025,100):
10         pwm.duty_u16(duty)
11         sleep(0.001)
12     for duty in range(65025, 10, -100):
13         pwm.duty_u16(duty)
14         sleep(0.001)
```

Shell x

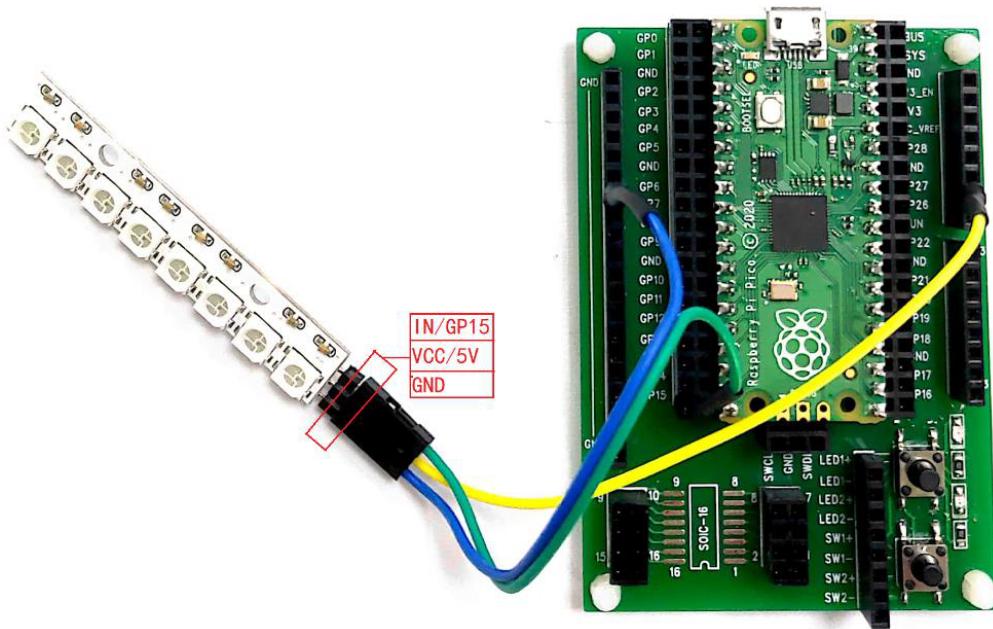
```
KeyboardInterrupt:
```

>>>

MicroPython (Raspberry Pi Pico)

#### 4.3 LED strip

The wiring is as follows:



The program code is as follows:

```
Thonny - D:\树莓派\Raspberry-Pi-Pico-Sensor\RGB_LED_ws2812b\rainbow.py @ 21 : 26
File Edit View Run Tools Help
rainbow.py > [ ws2812b.py ]>
1 import time
2 import ws2812b
3
4 numpix = 8
5 pin = 15
6 strip = ws2812b.ws2812b(numpix, 0, pin)
7
8 RED = (255, 0, 0)
9 ORANGE = (255, 165, 0)
10 YELLOW = (255, 150, 0)
11 GREEN = (0, 255, 0)
12 BLUE = (0, 0, 255)
13 INDIGO = (75, 0, 130)
14 VIOLET = (138, 43, 226)
15 COLORS = (RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO, VIOLET)      #define LED color change order
16
#include delay time
#define WS2812B
#define LED number
#define pin function:GP15
#define LED brightness(RED, GREEN, BLUE)
#define LED color change order
Shell >
MicroPython v1.13-290-g556ae7914 on 2021-01-21; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
>>> Backend terminated or disconnected. Use 'Stop/Restart' to restart.
MicroPython (Raspberry Pi Pico)
```

The screenshot shows the Thonny IDE interface. The title bar reads "Thonny - Raspberry Pi Pico :: /ws2812b.py @ 14:49". The menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations. The main window has a tab bar with "rainbow.py" and "[ws2812b.py] x". The code editor contains the following Python code:

```
1 import array, time
2 from machine import Pin
3 import rp2
4
5 @rp2.asm_pio(sideset_init=rp2 PIO.OUT_LOW, out_shiftdir=rp2 PIO.SHIFT_LEFT, autopull=True, pull_thresh=1)
6 def ws2812():
7     T1 = 2
8     T2 = 5
9     T3 = 3
10    wrap_target()
11    label("bitloop")
12    out(x, 1)           .side(0)    [T3 - 1]
13    jmp(not_x, "do_zero") .side(1)    [T1 - 1]
14    jmp("bitloop")      .side(1)    [T2 - 1]
15    label("do zero")
```

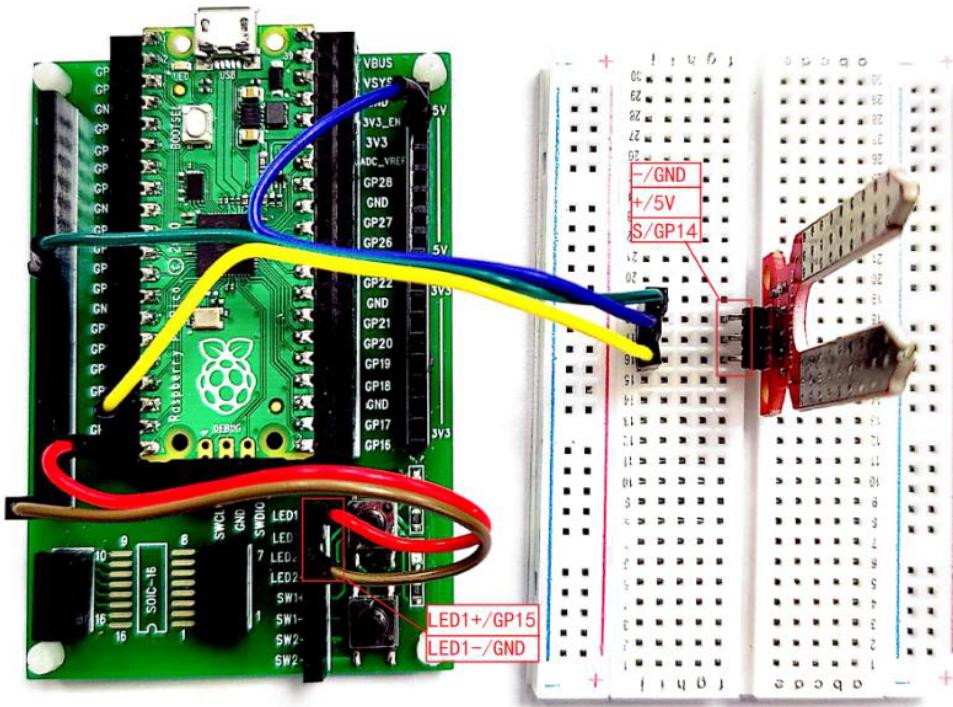
Below the code editor is a "Shell" tab with the following output:

```
Shell x

MicroPython v1.13-290-g556ae7914 on 2021-01-21; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
```

#### 4.4 Moisture Sensor

The wiring is as follows:



The program code is as follows:

The screenshot shows the Thonny IDE interface with the file 'main.py' open. The code is as follows:

```
1 import utime
2
3 led_external = machine.Pin(15, machine.Pin.OUT)
4 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
5
6 flag = 0
7
8 while True:
9     if aout.value() == 1:
10         led_external.value(1)
11         if flag:
12             print('led on')
13             flag = 0
14         utime.sleep(0.2)
15     else:
16         led_external.value(0)
17         if flag == 0:
18             print('led off')
19             flag = 1
20
```

The code uses a PULL\_UP configuration for pin 14 (AOUT). It prints 'led on' when AOUT is high and 'led off' when AOUT is low. A red box highlights the main loop, and two red arrows point to the print statements.

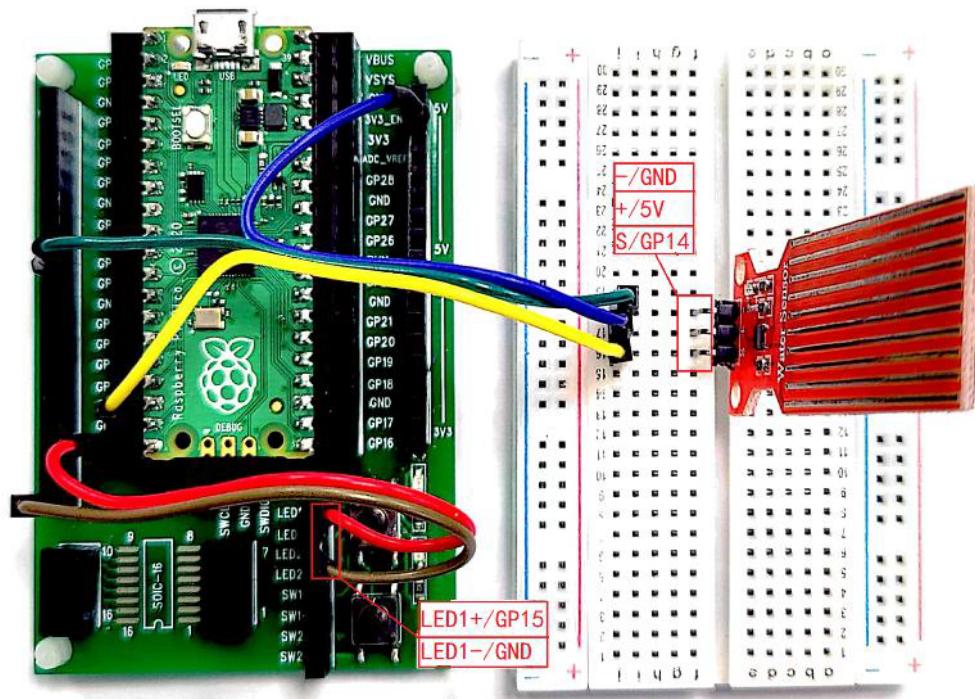
Below the code editor is a 'Shell' window showing:

```
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
```

At the bottom right of the interface is the text 'MicroPython (Raspberry Pi Pico)'.

#### 4.5 Liquid Level Sensor

The wiring is as follows:



The program code is as follows:

The screenshot shows the Thonny IDE interface with the file 'main.py' open. The code is as follows:

```
import utime
led_external = machine.Pin(15, machine.Pin.OUT)
aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
flag = 0

while True:
    if aout.value() == 1:
        led_external.value(1)
        if flag:
            print('led on')
        flag = 0
        utime.sleep(0.2)
    else:
        led_external.value(0)
        if flag == 0:
            print('led off')
        flag = 1
```

The code uses pin 15 as an output for an LED and pin 14 as an input for a liquid level sensor. It prints 'led on' when the sensor is high and 'led off' when it is low. A red box highlights the main loop and the print statements.

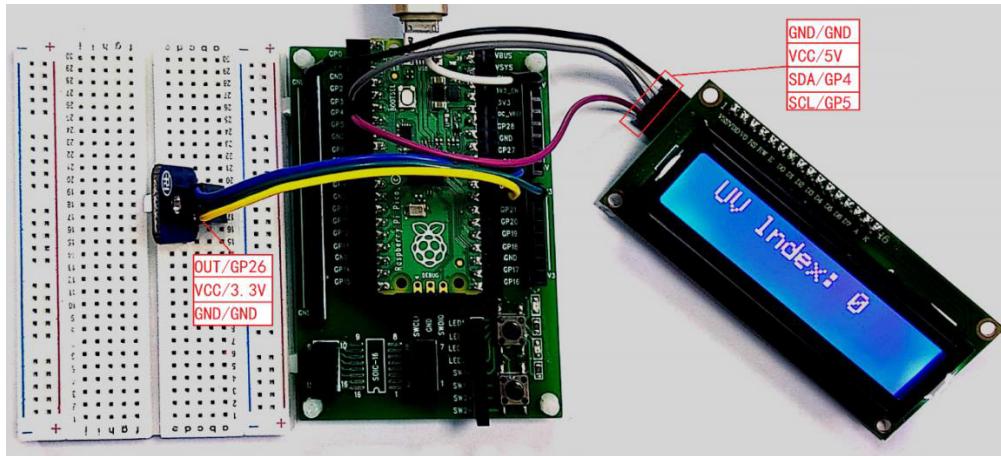
Shell

```
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
```

MicroPython (Raspberry Pi Pico)

## 4.6 UV Sensor

The wiring is as follows:



The program code is as follows:

Operation "main.py" First save the "lcd1602\_i2c.py" program to raspberry pi Pico, operation method refer to 3.3/step3.

The screenshot shows the Thonny IDE interface with the file 'main.py' open. The code is as follows:

```
import machine
import utime
led_external = machine.Pin(15, machine.Pin.OUT)
aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
flag = 0

while True:
    if aout.value() == 1:
        led_external.value(1)
        if flag:
            print('led on')
        flag = 0
        utime.sleep(0.2)
    else:
        led_external.value(0)
        if flag == 0:
            print('led off')
        flag = 1
```

The code uses pin 15 as an output for an LED and pin 14 as an input for a UV sensor. It prints 'led on' when the sensor is high and 'led off' when it is low. A red box highlights the main loop and the print statements.

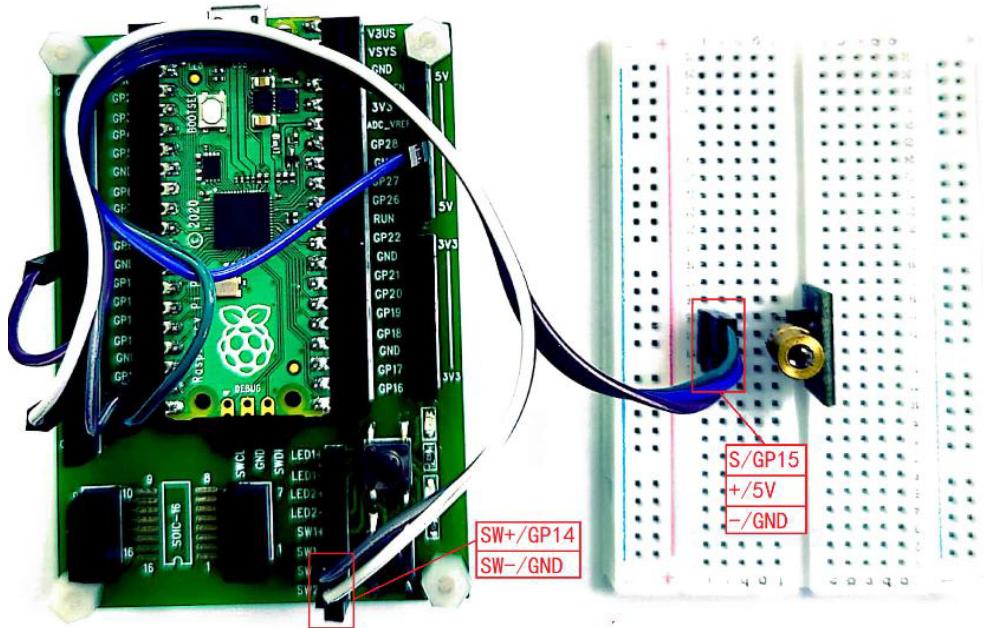
Shell

```
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
```

MicroPython (Raspberry Pi Pico)

#### 4.7 Laser out Sensor

The wiring is as follows:



The program code is as follows:

The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. The main window has two tabs: 'main.py' and 'Shell'. The 'main.py' tab contains the following Python code:

```
1 import machine
2 import utime
3
4 led_external = machine.Pin(15, machine.Pin.OUT)
5 key = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
6
7
8 while True:
9     if key.value() == 0:
10         led_external.value(1)
11         utime.sleep(0.2)
12     else:
13         led_external.value(0)
14
```

The 'Shell' tab shows the MicroPython prompt:

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

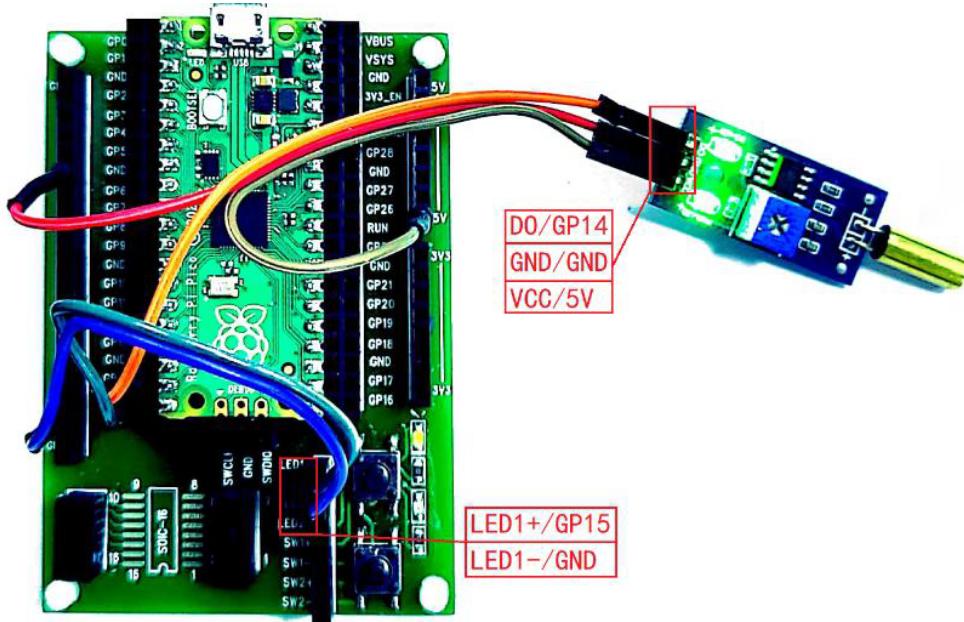
Connection lost (EOF)

Use Stop/Restart to reconnect.
```

At the bottom right, it says 'MicroPython (Raspberry Pi Pico)'.

## 4.8 Tilt Sensor

The wiring is as follows:



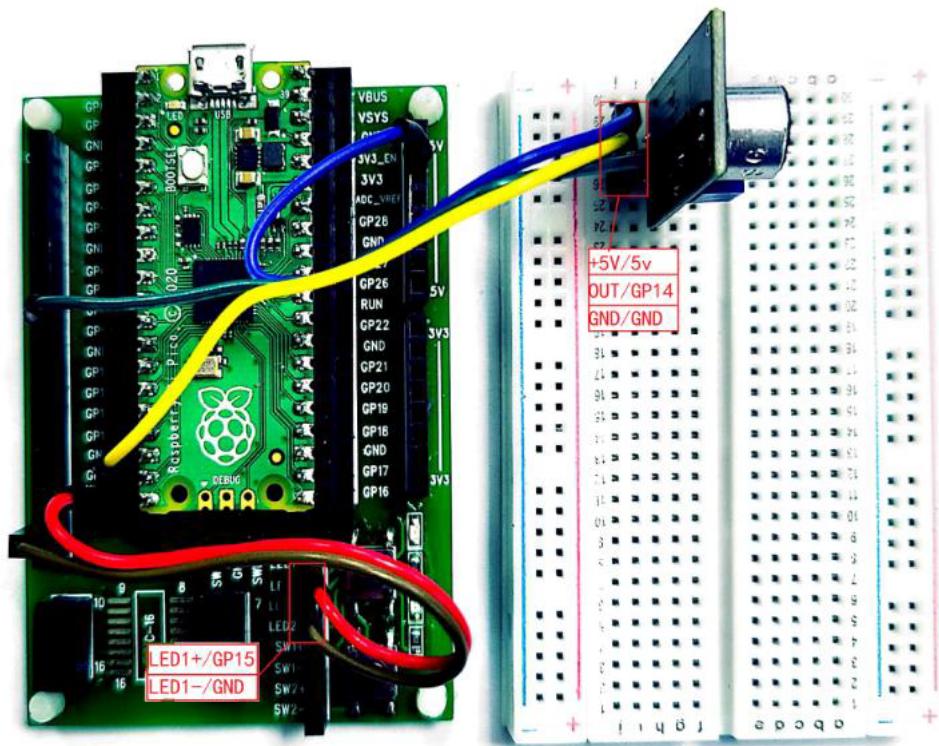
The program code is as follows:

```
Thonny - C:\Users\duncan380\Desktop\pico_0609\Tilt_sensor_pico_code\main.py @ 20 : 21
File Edit View Run Tools Help
main.py
2 import utime
3
4 led_external = machine.Pin(15, machine.Pin.OUT)
5 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
6
7 flag = 0
8
9 while True:
10     if aout.value() == 1:
11         led_external.value(1)
12         if flag:
13             print('led on')
14             flag = 0
15             utime.sleep(0.2)
16     else:
17         led_external.value(0)
18         if flag == 0:
19             print('led off')
20             flag = 1
#main loop
# check AOUT pin == input high
# LED output high
#delay 0.2sec
# AOUT pin == input low
# LED output low
#include delay time
#define LED pin function:GP15,out
#define AOUT pin function:GP14,in
```

The code in the screenshot is a Python script named `main.py` running on a MicroPython (Raspberry Pi Pico). It uses the `machine` module to control a digital output pin (GP15) and read from an analog input pin (GP14). The script enters a loop where it checks the state of the analog input. If it's high, it turns on an external LED (connected to GP15) and prints "led on". If it's low, it turns off the LED and prints "led off". A 0.2-second delay is added after each state change. The Thonny IDE interface shows the code in the editor and the output in the Shell window, which displays the printed messages and a message indicating it couldn't connect to COM4.

## 4.9 Sound Sensor

The wiring is as follows:



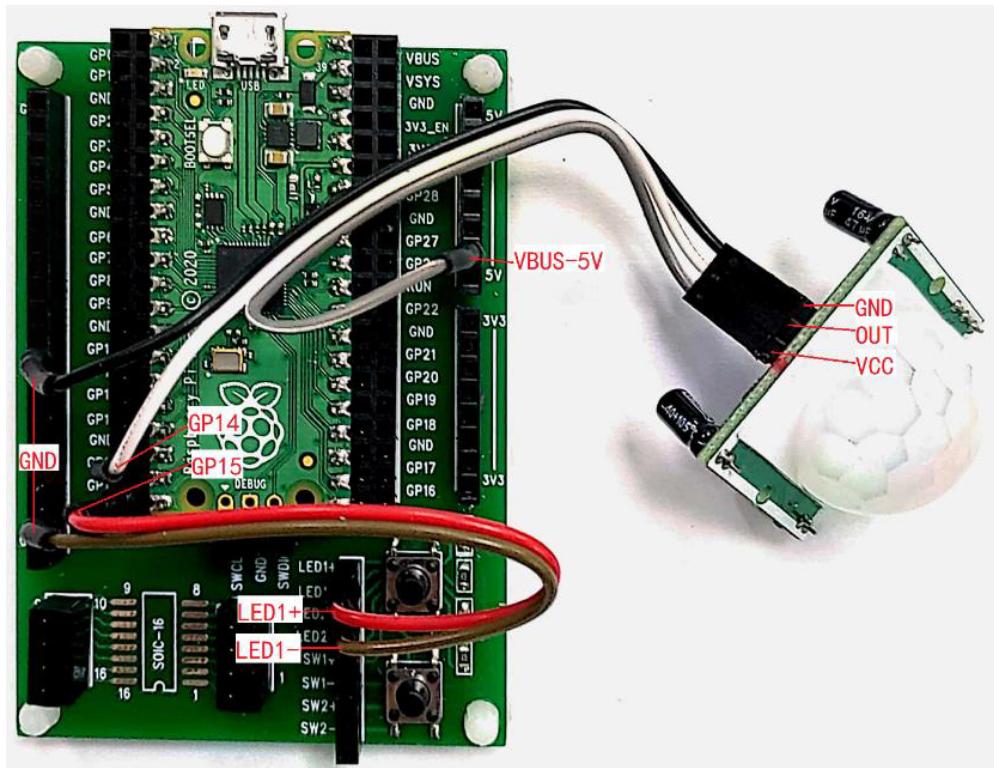
The program code is as follows:

```
Thonny - C:\Users\duncan380\Desktop\pico_0609\Sound-Sensor-pico_code\main.py @ 20:21
File Edit View Run Tools Help
main.py x
1 import time
2
3 led_external = machine.Pin(15, machine.Pin.OUT)
4 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
5
6
7 flag = 0
8
9 while True:
10     if aout.value() == 0:
11         led_external.value(1)
12         if flag:
13             print('led on')
14             flag = 0
15             time.sleep(0.2)
16     else:
17         led_external.value(0)
18         if flag == 0:
19             print('led off')
20             flag = 1
# include delay time
#define LED pin function:GP15,out
#define AOUT pin function:GP14,in
#main loop
# check AOUT pin == input high
# LED output high
#delay 0.2sec
# AOUT pin == input low
# LED output low

Shell x
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
MicroPython (Raspberry Pi Pico)
```

## 4.10 PIR Sensor

The wiring is as follows:



The program code is as follows:

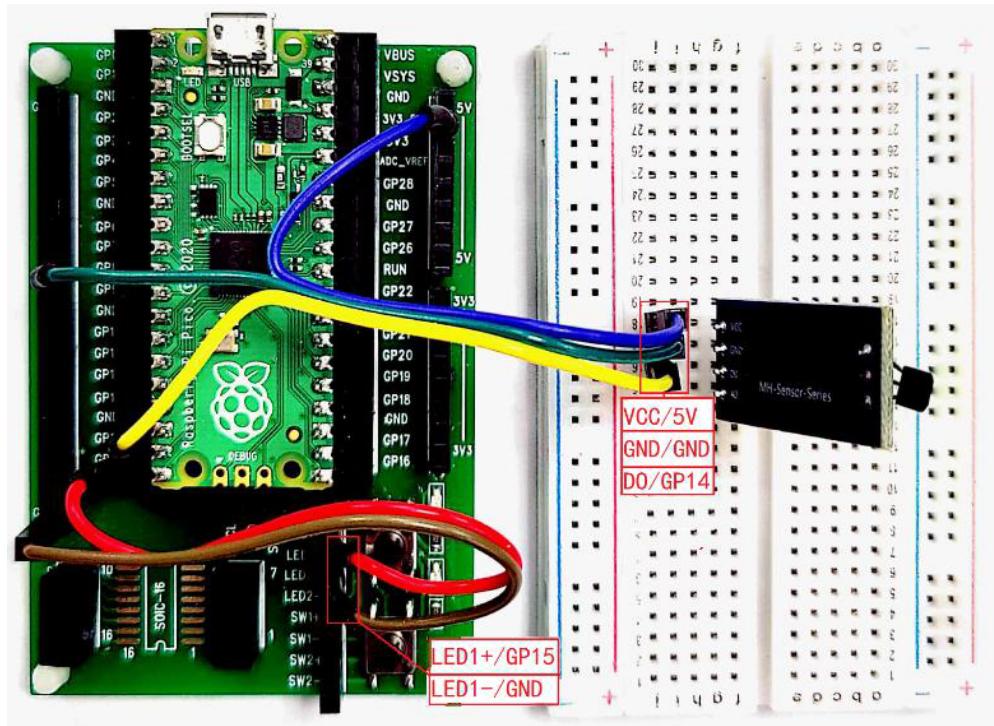
The screenshot shows the Thonny IDE interface with the file 'main.py' open. The code reads:

```
1 import utime
2
3 led_external = machine.Pin(15, machine.Pin.OUT)
4 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
5
6 flag = 1
7
8 while True:
9     if aout.value() == 1:
10         led_external.value(1)
11         if flag:
12             print('led on')
13             flag = 0
14             utime.sleep(0.2)
15     else:
16         led_external.value(0)
17         if flag == 0:
18             print('led off')
19             flag = 1
20
21
22 #include delay time
23
24 #define LED pin function:GP15,out
25 #define AOUT pin function:GP14,in
```

The code uses pins 14 and 15. Pin 14 is configured as an input with a pull-up resistor, and pin 15 is configured as an output. The program enters a loop where it checks the value of pin 14. If it is high, it prints 'led on', sets the flag to 0, and sleeps for 0.2 seconds. If it is low, it prints 'led off', sets the flag to 1, and exits the loop. The Thonny interface also shows a 'Shell' tab with error messages about connecting to COM4.

## 4.11 Hall Sensor

The wiring is as follows:



The program code is as follows:

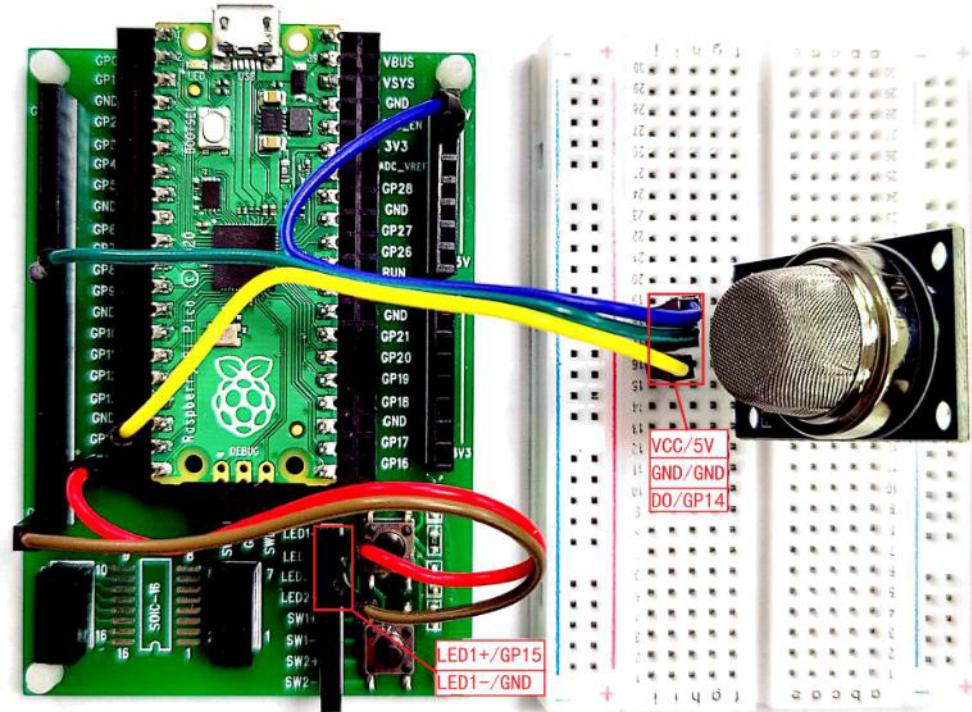
```
Thonny - C:\Users\duncan380\Desktop\pico_0609\Hall-Sensor-pico-code\main.py @ 20:21
File Edit View Run Tools Help
main.py x
2 import utime
3
4 led_external = machine.Pin(15, machine.Pin.OUT)
5 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
6
7 flag = 0
8
9 while True:
10     if aout.value() == 1:
11         led_external.value(1)
12         if flag:
13             print('led on') ←
14             flag = 0
15         utime.sleep(0.2)
16     else:
17         led_external.value(0)
18         if flag == 0:
19             print('led off') ←
20             flag = 1
# include delay time
#define LED pin function:GP15,out
#define AOUT pin function:GP14,in
#main loop
# check AOUT pin == input high
# LED output high

#delay 0.2sec
# AOUT pin == input low
# LED output low

Shell x
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
MicroPython (Raspberry Pi Pico)
```

## 4.12 Gas Sensor

The wiring is as follows:

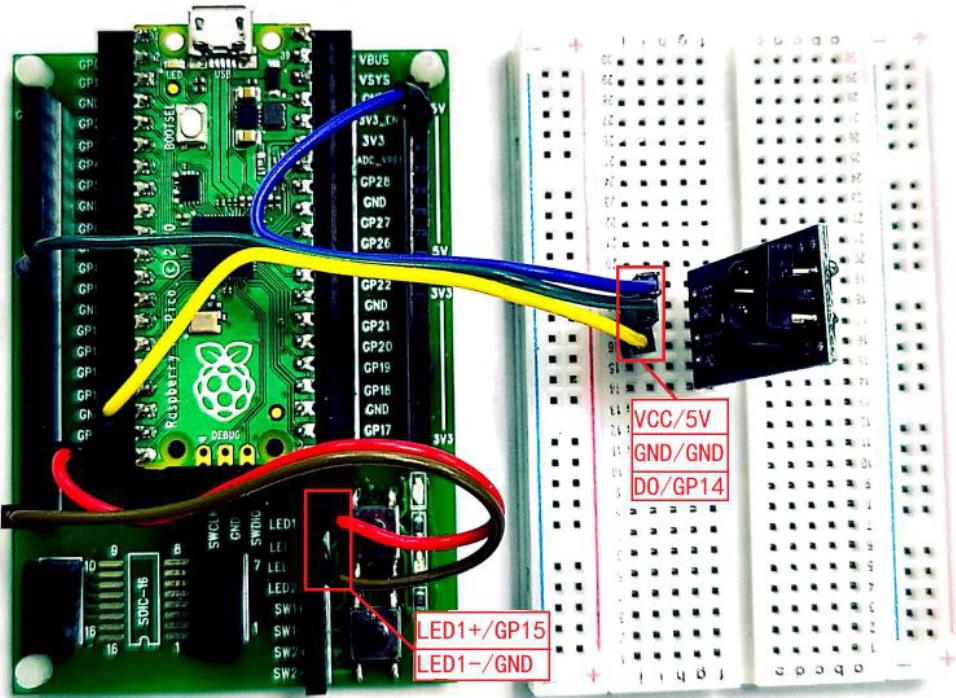


The program code is as follows:

```
Thonny - C:\Users\duncan380\Desktop\pico_0609\Gas_sensor-pico_code\main.py @ 22:21
File Edit View Run Tools Help
main.py x
3
4 led_external = machine.Pin(25, machine.Pin.OUT)
5 dout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
6
7 flag = 0
8
9 while True:
10     if dout.value() == 0:
11         if flag:
12             print('led warning')
13             flag = 0
14             led_external.value(0)
15             utime.sleep(0.2)
16             led_external.value(1)
17             utime.sleep(0.2)
18     else:
19         led_external.value(0)
20         if flag == 0:
21             print('led off')
22
23
#main loop
# check DOUT pin == input low
# LED flash warning
#delay 0.2sec
# DOUT pin == input high
# LED output low
#
# Shell
#
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
MicroPython (Raspberry Pi Pico)
```

## 4.13 Infrared Reflective Sensor

The wiring is as follows:



The program code is as follows:

The screenshot shows the Thonny IDE interface with the following details:

- Title Bar:** Thonny - C:\Users\duncan380\Desktop\pico\_0609\infrared\_Reflective\_sensor\_pico\_code\main.py @...
- Menu Bar:** File Edit View Run Tools Help
- Toolbar:** Standard file operations (New, Open, Save, etc.) and a Stop button.
- Code Editor:** The file "main.py" is open. The code uses MicroPython syntax to control an LED connected to pin 15 and read an input from pin 14. It includes a main loop with a 0.2-second sleep interval and prints "led on" or "led off" to the shell based on the input state. A red box highlights the main loop, and two red arrows point to the print statements within it.
- Code Content:**

```
import utime
# include delay time

led_external = machine.Pin(15, machine.Pin.OUT)
#define LED pin function:GP15,out

aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
#define AOUT pin function:GP14,in

flag = 0

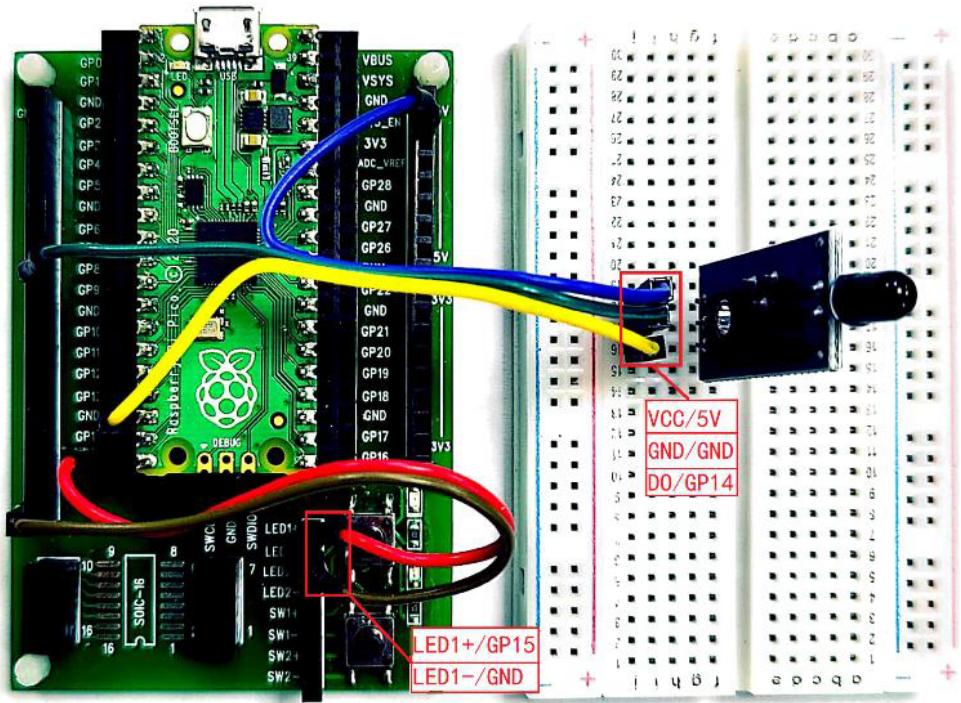
while True:
    if aout.value() == 1:
        led_external.value(1)
        if flag:
            print('led on') ←
        flag = 0
        utime.sleep(0.2)
    else:
        led_external.value(0)
        if flag == 0:
            print('led off') ←
        flag = 1

#main loop
# check AOUT pin == input high
# LED output high

#delay 0.2sec
# AOUT pin == input low
# LED output low
```
- Shell:** Shows the error message "Unable to connect to COM4: port not found" and "Backend terminated or disconnected. Use 'Stop/Restart' to restart."
- Status Bar:** MicroPython (Raspberry Pi Pico)

#### 4.14 Flame Sensor

The wiring is as follows:

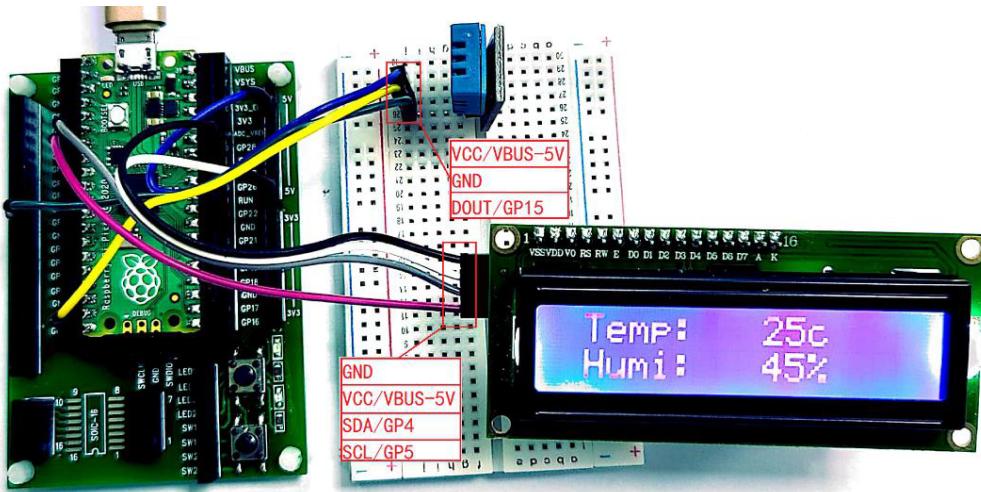


The program code is as follows:

```
Thonny - C:\Users\duncan380\Desktop\pico_0609\Flame-Sensor-pico-code\main.py @ 9:1
File Edit View Run Tools Help
main.py x
3
4     led_external = machine.Pin(15, machine.Pin.OUT)
5     aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
6
7     flag = 0
8
9     while True:
10         if aout.value() == 1:
11             led_external.value(1)
12             if flag:
13                 print('led on')
14                 flag = 0
15             utime.sleep(0.2)
16         else:
17             led_external.value(0)
18             if flag == 0:
19                 print('led off')
#define LED pin function:GP15,out
#define AOUT pin function:GP14,in
#main loop
# check AOUT pin == input high
# LED output high
#delay 0.2sec
# AOUT pin == input low
# LED output low
Shell x
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
MicroPython (Raspberry Pi Pico)
```

## 4.15 Temperature-Humidity Sensor

The wiring is as follows:



The program code is as follows:

Operation "main.py" First save "dht11.py" and "lcd1602\_i2c.py" program to raspberry pi Pico, operation method refer to 3.3/step3.

**Note: The file name must be consistent with the name of the program, the case letters and symbols must be consistent.**

The screenshots show the Thonny IDE interface with two windows open:

- File1:** Thonny - Raspberry Pi Pico :: /lcd1602\_i2c.py @ 277:1
- File2:** Thonny - Raspberry Pi Pico :: /dht11.py @ 65:48

**lcd1602\_i2c.py:**

```
from time import sleep_ms
# The PCF8574 has a jumper selectable address: 0x20 - 0x27
DEFAULT_I2C_ADDR = 0x27

# Defines shifts or masks for the various LCD line attached to the PCF8574
MASK_RS = 0x01
MASK_RW = 0x02
MASK_E = 0x04
SHIFT_BACKLIGHT = 3
SHIFT_DATA = 4

class LcdApi:
```

**Shell:**

```
temperature is 28 -wet is 78 %
Sensor is working
temperature is 28 -wet is 78 %
Sensor is working
temperature is 28 -wet is 78 %
Sensor is working
temperature is 28 -wet is 78 %
```

**MicroPython (Raspberry Pi Pico)**

**dht11.py:**

```
#import pyb
from machine import UART
from machine import Pin
from time import sleep_ms,sleep_us

class DHT11:
    def __init__(self,pin_name):
        sleep_ms(1000)
        self.N1 = Pin(pin_name, Pin.OUT)
        self.PinName=pin_name
        sleep_ms(10)
    def read_data(self):
        self.__init__(self.PinName)
```

**Shell:**

```
temperature is 28 -wet is 83 %
Sensor is working
temperature is 28 -wet is 83 %
Sensor is working
temperature is 28 -wet is 83 %
Sensor is working
temperature is 28 -wet is 83 %
```

**MicroPython (Raspberry Pi Pico)**

The screenshot shows the Thonny IDE interface with the following details:

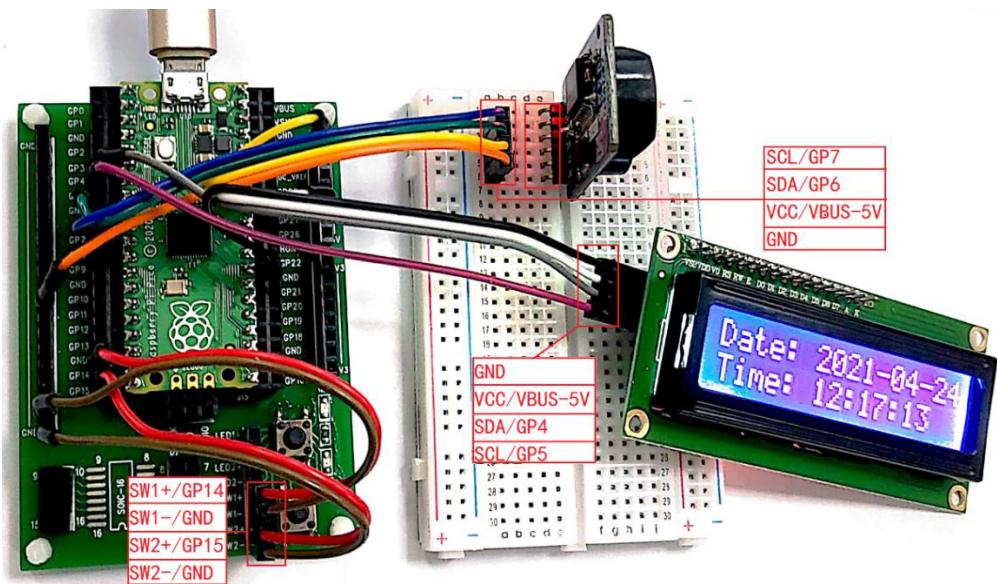
- Title Bar:** Thonny - D:\树莓派\Temperature-Humidity-Sensor-Pico\_code\main.py @ 35:1
- Menu Bar:** File Edit View Run Tools Help
- Toolbar:** Standard icons for file operations.
- Code Editor:** The main window displays the Python code for reading temperature and humidity from a DHT11 sensor. The code includes imports for time, machine, and I2c, and defines constants for the I2C address and DHT11 pin. It also includes comments for the PCF8574 jumper and hardware definitions. A red box highlights the first 15 lines of the code.
- Shell:** A terminal window at the bottom shows the output of the script execution, displaying repeated readings of 27°C and 81% humidity, followed by a final reading of 28°C and 82% humidity.
- Bottom Status Bar:** MicroPython (Raspberry Pi Pico)

## 4.16 Clock sensor

Operation "main.py" First save "DS3231.py" and "lcd1602\_i2c.py" program to raspberry pi Pico, operation method refer to 3.3/step3.

Note: The file name must be consistent with the name of the program, the case letters and symbols must be consistent.

The wiring is as follows:



The program code is as follows:

Thonny - Raspberry Pi Pico :: /ds3231.py @ 29 : 93

```

File Edit View Run Tools Help
[ds3231.py] * [lcd1602_i2c.py] [main.py]
1 import machine
2 from machine import I2C,Pin
3
4 DS3231_ADDR = 0x68 #const(0x68)
5 DS3231_REG_SEC = b'\x00'
6 DS3231_REG_MIN = b'\x01'
7 DS3231_REG_HOUR = b'\x02'
8 DS3231_REG_WEEKDAY=b'\x03'
9 DS3231_REG_DAY = b'\x04'
10 DS3231_REG_MONTH = b'\x05'
11 DS3231_REG_YEAR = b'\x06'
12 DS3231_REG_A1SEC = b'\x07'
13 DS3231_REG_A1MIN = b'\x08'
14 DS3231_REG_A1HOUR = b'\x09'
15 DS3231_REG_A1DAY = 0x0A
...
```
Shell
```python
>>> traceback (most recent call last).
File "<stdin>", line 22, in <module>
File "lcd1602_i2c.py", line 132, in move_to
File "lcd1602_i2c.py", line 258, in hal_write_command
OSError: 5
```
>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.

```

MicroPython (Raspberry Pi Pico)

Thonny - D:\树莓派\DS3231时钟模块\pico\_code\lcd1602\_i2c.py @ 277 : 1

```

File Edit View Run Tools Help
[ds3231.py] * [lcd1602_i2c.py] [main.py]
1 from time import sleep_ms
2
3 # The PCF8574 has a jumper selectable address: 0x20 - 0x27
4 DEFAULT_I2C_ADDR = 0x27#0x27#
5
6 # Defines shifts or masks for the various LCD line attached to the PCF8574
7
8 MASK_RS = 0x01
9 MASK_RW = 0x02
10 MASK_E = 0x04
11 SHIFT_BACKLIGHT = 3
12 SHIFT_DATA = 4
13
14
15 class LcdApi:
...
```
Shell
```python
>>> traceback (most recent call last).
File "<stdin>", line 22, in <module>
File "lcd1602_i2c.py", line 132, in move_to
File "lcd1602_i2c.py", line 258, in hal_write_command
OSError: 5
```
>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.

```

MicroPython (Raspberry Pi Pico)

Thonny - D:\树莓派\DS3231时钟模块\pico\_code\main.py @ 5 : 1

```

File Edit View Run Tools Help
[ds3231.py] * [lcd1602_i2c.py] [main.py]
1 from machine import I2C, Pin
2 import time
3 from ds3231 import DS3231
4 from lcd1602_i2c import I2cLcd
5
6 DEFAULT_I2C_ADDR = 0x27
7
8
9 if __name__ == "__main__":
10     i2c = I2C(0,scl=Pin(5), sda=Pin(4), freq=100000)
11     lcd = I2cLcd(i2c, DEFAULT_I2C_ADDR, 2, 16)
12     lcd.clear()
13     ds=DS3231()
14     ds.DATE([21,4,19])
15     ds.TIME([14,26,00])
...
```
Shell
```python
>>> traceback (most recent call last).
File "<stdin>", line 22, in <module>
File "lcd1602_i2c.py", line 132, in move_to
File "lcd1602_i2c.py", line 258, in hal_write_command
OSError: 5
```
>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.

```

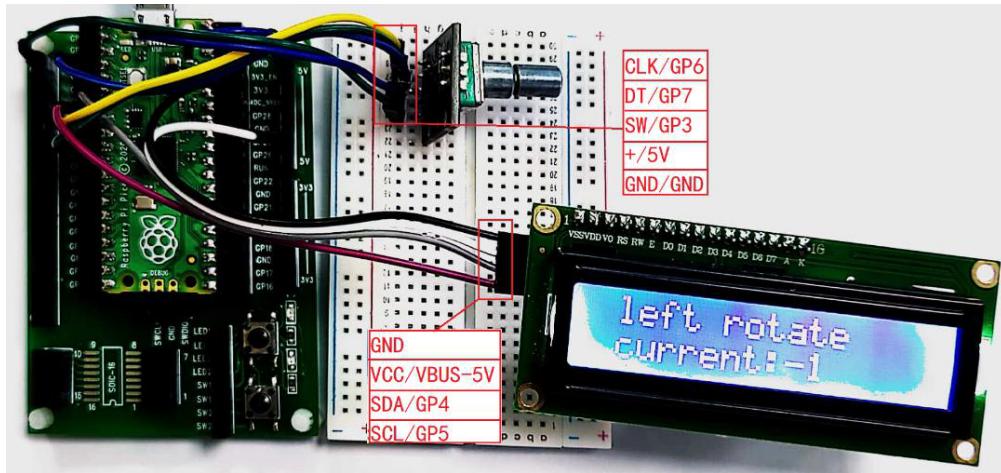
MicroPython (Raspberry Pi Pico)

#### 4.17 Rotation Sensor

Operation "main.py" First save the "lcd1602\_i2c.py" program to raspberry pi Pico, operation method refer to 3.3/step3.

**Note: The file name must be consistent with the name of the program, the case letters and symbols must be consistent.**

The wiring is as follows:



The program code is as follows:

```
Thonny - D:\树莓派\Rotation-Sensor-pico_code\lcd1602_i2c.py @ 277:1
File Edit View Run Tools Help
main.py lcd1602_i2c.py
1 from time import sleep_ms
2
3 # The PCF8574 has a jumper selectable address: 0x20 ~ 0x27
4 DEFAULT_I2C_ADDR = 0x27#0x27#
5
6 # Defines shifts or masks for the various LCD line attached to the PCF8574
7
8 MASK_RS = 0x01
9 MASK_RW = 0x02
10 MASK_E = 0x04
11 SHIFT_BACKLIGHT = 3
12 SHIFT_DATA = 4
13
14
15 class LcdApi:
Shell
current = -1
right rotate
current = 0
right rotate
current = 1
left rotate
current = 0
MicroPython (Raspberry Pi Pico)
```

Thonny - D:\树莓派\Raspberry-Pi-Pico-Sensor\push\_up\_R\main.py @ 9:4

File Edit View Run Tools Help

main.py < lcd1602\_i2c.py

```

1 from machine import I2C, Pin, Timer
2 #import time
3 from time import sleep_ms, sleep_us
4 from lcd1602_i2c import I2cLcd
5
6 DEFAULT_I2C_ADDR = 0x27
7
8 SW = machine.Pin(3, machine.Pin.IN,machine.Pin.PULL_UP)
9 CLK = machine.Pin(6, machine.Pin.IN,machine.Pin.PULL_UP)
10 DT = machine.Pin(7, machine.Pin.IN,machine.Pin.PULL_UP)
11
12 dis_cnt = 0
13
14 def tick(timer):
15     global dis_cnt
16     dis_cnt += 1

```

Shell <

```

left rotate
current = -1
fh_1 is 0
left rotate
current = -2
fh_1 is 1

```

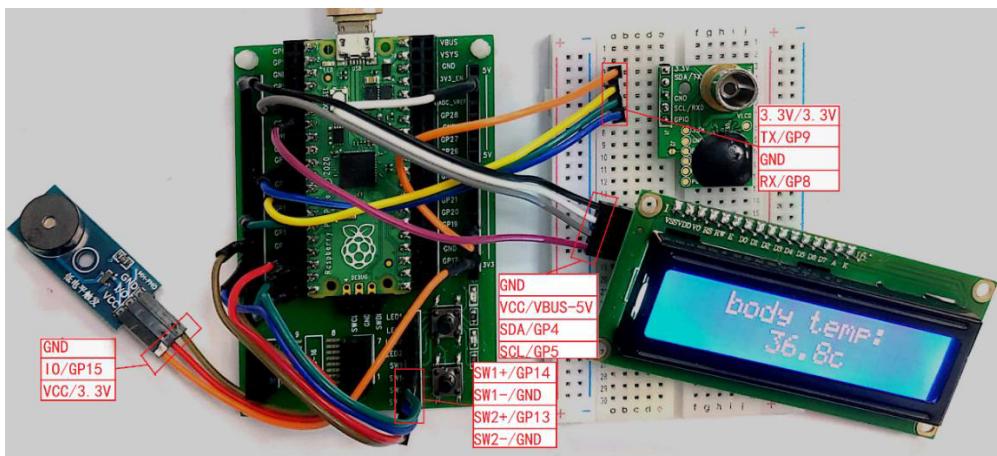
MicroPython (Raspberry Pi Pico)

#### 4.18 Temperature Sensor

Operation "main.py" First save the "lcd1602\_i2c.py" program to raspberry pi Pico, operation method refer to 3.3/step3.

**Note: The file name must be consistent with the name of the program, the case letters and symbols must be consistent.**

The wiring is as follows:



The program code is as follows:

The image contains two side-by-side screenshots of the Thonny Python IDE interface, both titled "Thonny - D:\树莓派\LS\_TM04\lcd1602\_i2c.py @ 8:15" and "Thonny - D:\树莓派\Raspberry-Pi-Pico-Sensor\LS\_TM01\main.py @ 8:22".

**Screenshot 1 (Top):**

```

from time import sleep_ms
# The PCF8574 has a jumper selectable address: 0x20 ~ 0x27
DEFAULT_I2C_ADDR = 0x27#0x27

# Defines shifts or masks for the various LCD line attached to the PCF8574
MASK_RS = 0x01
MASK_RW = 0x02
MASK_E = 0x04
SHIFT_BACKLIGHT = 3
SHIFT_DATA = 4

class LcdApi:
    pass

uart1: 11
uart1: 121
uart1: 0
uart1: 8
uart1: 0
uart1: 0
uart1: 210
uart1: 10

```

**Screenshot 2 (Bottom):**

```

from machine import I2C,UART, Pin, Timer
import utime
from lcd1602_i2c import I2cLcd
#VCC = 3.3V

DEFAULT_I2C_ADDR = 0x27
buz = machine.Pin(15, machine.Pin.OUT)
key1 = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
key2 = machine.Pin(13, machine.Pin.IN, machine.Pin.PULL_UP)

def tick(timer):
    global tb_chk_cnt
    global test_cnt
    global mode_ptr
    global key_cnt
    global key2_cnt
    global key_press

Ambient temp

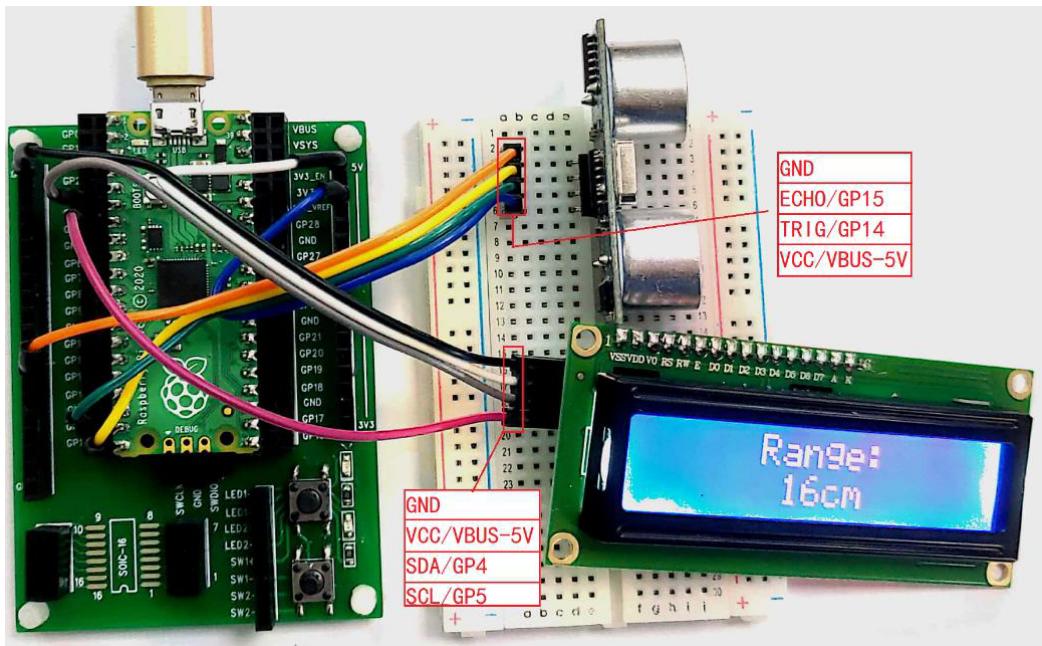
```

#### 4.19 Ultrasonic sensor

Operation “main.py” First save the “lcd1602\_i2c.py” program to raspberry pi Pico, operation method refer to 3.3/step3.

**Note: The file name must be consistent with the name of the program, the case letters and symbols must be consistent.**

The wiring is as follows:



The program code is as follows:

Thonny - Raspberry Pi Pico :: /lcd1602\_i2c.py @ 277 : 1

```

File Edit View Run Tools Help
main.py [ lcd1602_i2c.py ] x
1 from time import sleep_ms
2
3 # The PCF8574 has a jumper selectable address: 0x20 - 0x27
4 DEFAULT_I2C_ADDR = 0x27#0x27
5
6 # Defines shifts or masks for the various LCD line attached to the PCF8574
7
8 MASK_RS = 0x01
9 MASK_RW = 0x02
10 MASK_E = 0x04
11 SHIFT_BACKLIGHT = 3
12 SHIFT_DATA = 4
13
14
15 class LcdApi:
16
Shell <
>>> %Run -c $EDITOR_CONTENT
MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

```

MicroPython (Raspberry Pi Pico)

Thonny - D:\树莓派\超声测距模块\pico\_code\main.py @ 36 : 16

```

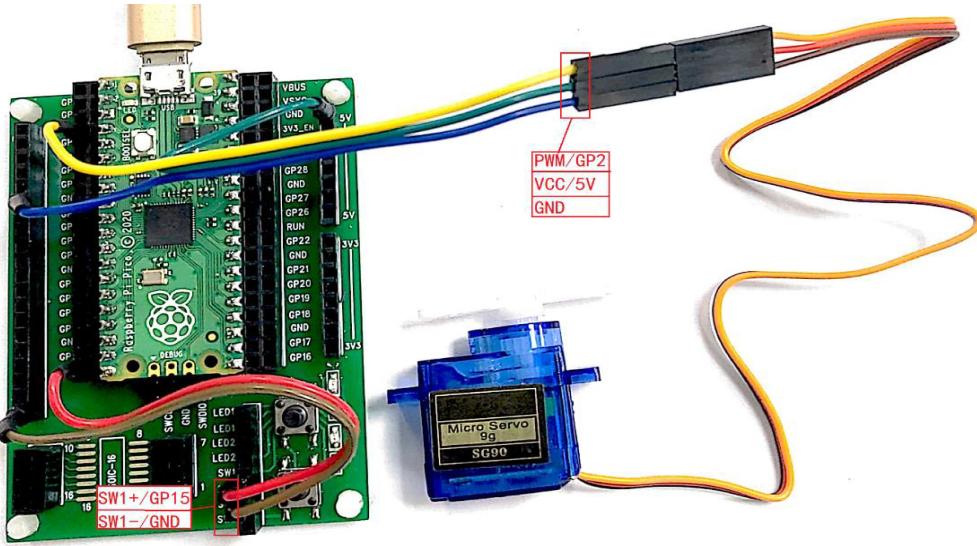
File Edit View Run Tools Help
main.py [ lcd1602_i2c.py ] x
1 from time import sleep_ms, sleep_us
2 from machine import I2C, Pin
3 from lcd1602_i2c import I2cLcd
4
5 TRIG_OUT = machine.Pin(14, machine.Pin.OUT)
6 ECHO_INT = Pin(15, Pin.IN, Pin.PULL_UP)
7
8 # The PCF8574 has a jumper selectable address: 0x20 - 0x27
9 DEFAULT_I2C_ADDR = 0x27
10
11
12 def read_Data(self):
13     global dat
14     if ECHO_INT.value() == 0:
15         dat = 0
16     else:
# include delay time
#include hardware device
#define LCD1602 Function device
#define HC-SR04 pin function
##define HC-SR04 pin function INPUT
Shell <
>>> %Run -c $EDITOR_CONTENT
MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

```

MicroPython (Raspberry Pi Pico)

## 4.20 Steering engine

The wiring is as follows:



The program code is as follows:

Thonny - C:\Users\duncan380\Desktop\pico\_0609\9g\_moto\main.py @ 20:1

File Edit View Run Tools Help

main.py

```
1 from machine import Pin, PWM, Timer
2 import time
3
4 pwm = PWM(Pin(2))
5 key1 = machine.Pin(15, machine.Pin.IN, machine.Pin.PULL_UP)
6
7
8 #moto run range:500~2500 us, precision: 0.09°/us,
9 #us= duty*20000/65535
10 #duyt=us*65535/20000
11 pwm.freq(50)                                #50hz=20000us duty times=20000/65535
12 DUTY0 = 1638                                  #us= 500us,0°
13 DUTY45 = 3276                                 #us= 1000us,45°
14 DUTY90 = 4915                                 #us= 1500us,90°
15 DUTY100 = 5278                                #us= 1611us,100°
16 DUTY135 = 6554                                #us= 2000us,135°
17 DUTY150 = 7099                                #us= 2166us,150°
18 DUTY180 = 8192                                #us= 2500us,180°
```

Shell

```
0
45
90
100
135
150
180
0
```

MicroPython (Raspberry Pi Pico)

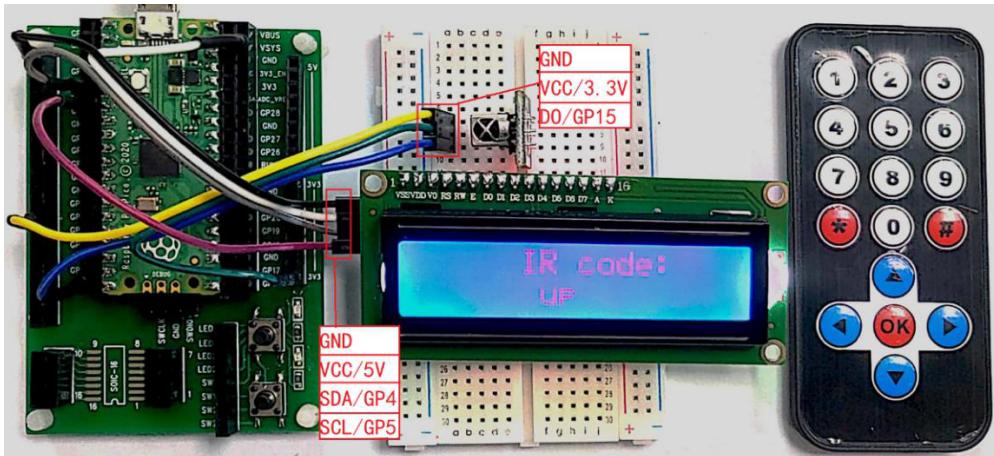
The screenshot shows the Thonny IDE interface. The main window displays the Python script "main.py" with several lines of code highlighted by a red box. The "Shell" window at the bottom shows a list of values: 0, 45, 90, 100, 135, 150, 180, and 0 again. The status bar at the bottom right indicates "MicroPython (Raspberry Pi Pico)".

## 4.21 Infrared remote control

Operation "main.py" First save the "lcd1602\_i2c.py" program to raspberry pi Pico, operation method refer to 3.3/step3.

**Note: The file name must be consistent with the name of the program, the case letters and symbols must be consistent.**

The wiring is as follows:



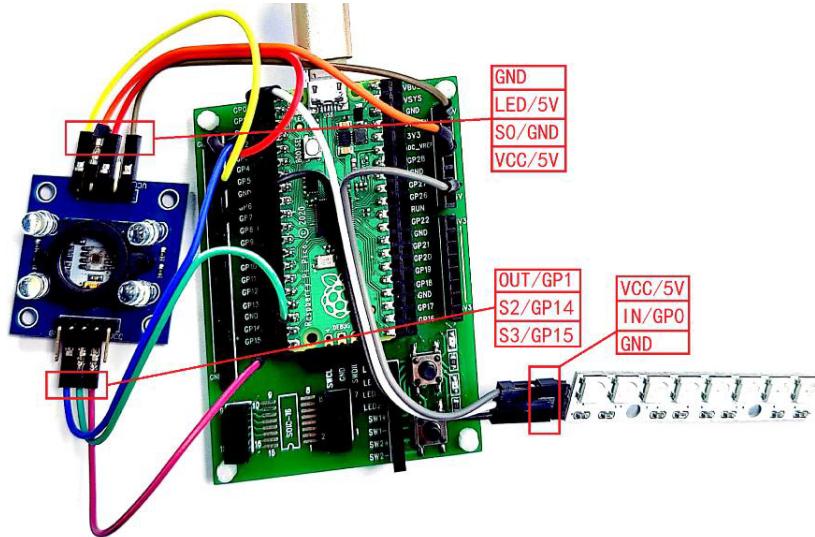
The program code is as follows:

```
Thonny - D:\树莓派\红外遥控接收解码\pico_code\main.py @ 154:33
File Edit View Run Tools Help
main.py [ lcd1602_i2c.py ] ×
1 from time import sleep_ms, sleep_us
2 from machine import I2C, Pin
3 from lcd1602_i2c import I2cLcd
4
5
6 IR_INT = Pin(15, Pin.IN, Pin.PULL_UP)           #INPUT pin
7
8 # The PCF8574 has a jumper selectable address: 0x20 - 0x27
9 DEFAULT_I2C_ADDR = 0x27
10
11
12 def read_Data(self):
13     global dat
14     global code_dis_flag
15     if IR_INT.value() == 1:
16         print('IRf')
Shell ×
IRf
IRf
IRf
IRf
IRf
IRf
MicroPython (Raspberry Pi Pico)
```

## 4.22 Color sensor

Operation "main.py" First save the "ws2812b.py" program to raspberry pi Pico, operation method refer to 3.3/step3.

The wiring is as follows:



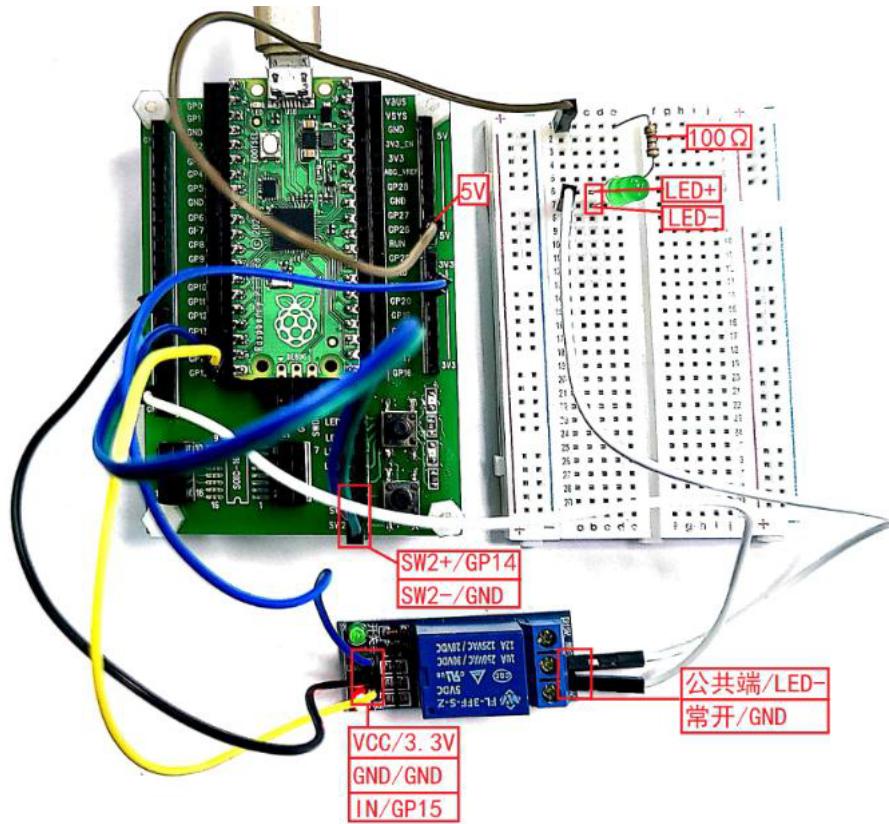
The program code is as follows:

```
Thonny - D:\树莓派\color_pico_code\ws2812b.py @ 92:1
File Edit View Run Tools Help
[ main.py ] ws2812b.py
1 import array, time
2 from machine import Pin
3 import rp2
4
5 @rp2.asm_pio(sideset_init=rp2 PIO.OUT_LOW, out_shiftdir=rp2 PIO.SHIFT_LEFT, autopull=True, pull_thresh:
6 def ws2812():
7     T1 = 2
8     T2 = 5
9     T3 = 3
10    wrap_target()
11    label("bitloop")
12    out(x, 1)      .side(0)      [T3 - 1]
13    jmp(not_x, "do_zero") .side(1)      [T1 - 1]
14    jmp("bitloop") .side(1)      [T2 - 1]
15    label("do zero")
<   >
Shell x
>>>
MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
MicroPython (Raspberry Pi Pico)
```

```
Thonny - Raspberry Pi Pico :: /main.py @ 137:8
File Edit View Run Tools Help
[ main.py ] ws2812b.py
1 from machine import Pin,Timer
2 import time
3 from ws2812b import ws2812b
4
5 num_leds = 144
6 pixels = ws2812b(num_leds, 0, 0, delay=0)
7
8 #LED = VCC
9 p2 = Pin(1, Pin.IN, Pin.PULL_UP)           #OUT pin
10 #S0_out = GND                            #2% output frequency
11 #S1_out = VCC
12 S2_out = machine.Pin(14, machine.Pin.OUT)
13 S3_out = machine.Pin(15, machine.Pin.OUT)
14
15 dat = 0
16 flag = 0
Shell x
>>>
MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
MicroPython (Raspberry Pi Pico)
```

## 4.23 Relay

The wiring is as follows:



The program code is as follows:

```
File Edit View Run Tools Help
main.py x
1 import machine
2 import utime
3
4 led_external = machine.Pin(15, machine.Pin.OUT)
5 aout = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_UP)
6
7 flag = 0
8
9 while True:
10     if aout.value() == 1:
11         led_external.value(1)
12         if flag:
13             print('led on')
14             flag = 0
15             utime.sleep(0.2)
16     else:
17         led_external.value(0)
18         if flag == 0:
19             print('led off')
20             flag = 1
21
22
#main loop
# check AOUT pin == input high
# LED output high

#delay 0.2sec
# AOUT pin == input low
# LED output low

#include hardware devices
#include delay time
#define LED pin function:GP15,out
#define AOUT pin function:GP14,in|
```

Shell x

```
led on
led off
led on
```

MicroPython (Raspberry Pi Pico)