# Welcome

Thank you for choosing Freenove products!

## Getting Started

When reading this, you should have downloaded the ZIP file for this product.
Unzip it and you will get a folder containing tutorials and related files. Please start with this PDF tutorial.

! Unzip the ZIP file instead of opening the file in the ZIP file directly.
! Do not move, delete or rename files in the folder just unzipped.

## Get Support

Encounter problems? Don't worry! Refer to "TroubleShooting.pdf" or contact us.

When there are packaging damage, quality problems, questions encountering in use, etc., just send us an email. We will reply to you within one working day and provide a solution.

support@freenove.com

## Safety and Precautions

Please follow the following safety precautions when using or storing this product:

- Keep this product out of the reach of children under 6 years old.
- This product should be used only when there is adult supervision present as young children lack necessary judgment regarding safety and the consequences of product misuse.
- This product contains small parts and parts, which are sharp. This product contains electrically conductive parts. Use caution with electrically conductive parts near or around power supplies, batteries and powered (live) circuits.
- When the product is turned ON, activated or tested, some parts will move or rotate. To avoid injuries to hands and fingers, keep them away from any moving parts!
- It is possible that an improperly connected or shorted circuit may cause overheating. Should this happen, immediately disconnect the power supply or remove the batteries and do not touch anything until it cools down! When everything is safe and cool, review the product tutorial to identify the cause.
- Only operate the product in accordance with the instructions and guidelines of this tutorial, otherwise parts may be damaged or you could be injured.
- Store the product in a cool dry place and avoid exposing the product to direct sunlight.
- After use, always turn the power OFF and remove or unplug the batteries before storing.

## About Freenove

Freenove provides open source electronic products and services worldwide.

Freenove is committed to assist customers in their education of robotics, programming and electronic circuits so that they may transform their creative ideas into prototypes and new and innovative products. To this end, our services include but are not limited to:

- Educational and Entertaining Project Kits for Robots, Smart Cars and Drones
- Educational Kits to Learn Robotic Software Systems for Arduino, Raspberry Pi and micro: bit
- Electronic Component Assortments, Electronic Modules and Specialized Tools
- **Product Development and Customization Services**

You can find more about Freenove and get our latest news and updates through our website:

http://www.freenove.com

## Copyright



**Any concerns? ✉ support@freenove.com**

# Contents

# Preface

Raspberry Pi Pico is a tiny, fast, and versatile board built using RP2040, a brand new microcontroller chip designed by Raspberry Pi in the UK. Getting started is as easy as dragging and dropping a file, it is suitable for beginners and makers to develop, design and research.

Raspberry Pi Pico is programmable in C/C++ and MicroPython. In this tutorial, we use Micropython to develop, which is a very easy to learn language with lean and simple code, hence it is very suitable for beginners to learn and for secondary development.

If you haven't downloaded the related material for Raspberry Pi Pico tutorial, you can download it from this link:
https://github.com/Freenove/Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico

In this tutorial, we devide each project into 4 sections:

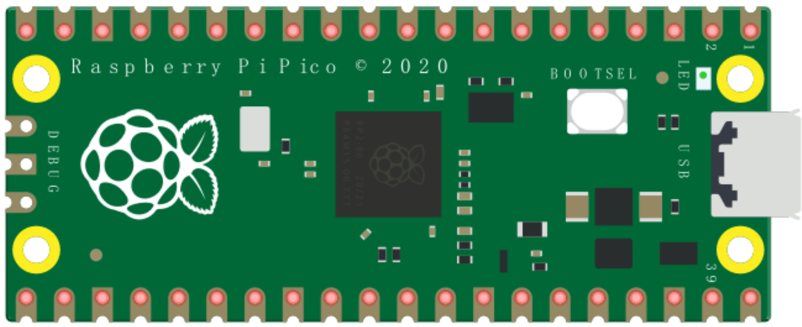
1, Component list: helps users to learn and find what components needed in each project.
2, Component knowledge: allows you to learn the features and usage of the components.
3, Circuit: assists to build circuit for each project.
4, Code and annotation: makes it easier for users to learn to use Raspberry Pi Pico and make secondary development.

After completing the projects in this tutorial, you can also combine the components in different projects to make your own smart homes, smart car, robot, etc., bringing your imagination and creativity to life with Raspberry Pi Pico.

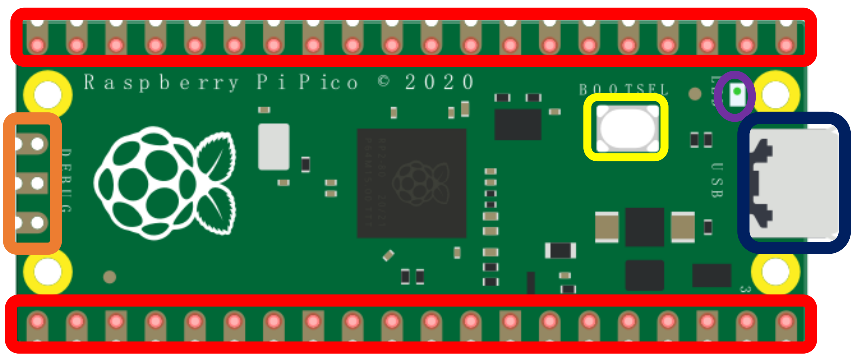
If you have any problems or difficulties using this product, please contact us for quick and free technical support: support@freenove.com

## Raspberry Pi Pico

Raspberry Pi Pico is a light-weight electronic product with tiny size and low price. From the picture below we can see that its onboard resources have been connected to the edge interface, which is very suitable for electronic enthusiasts to use in DIY.

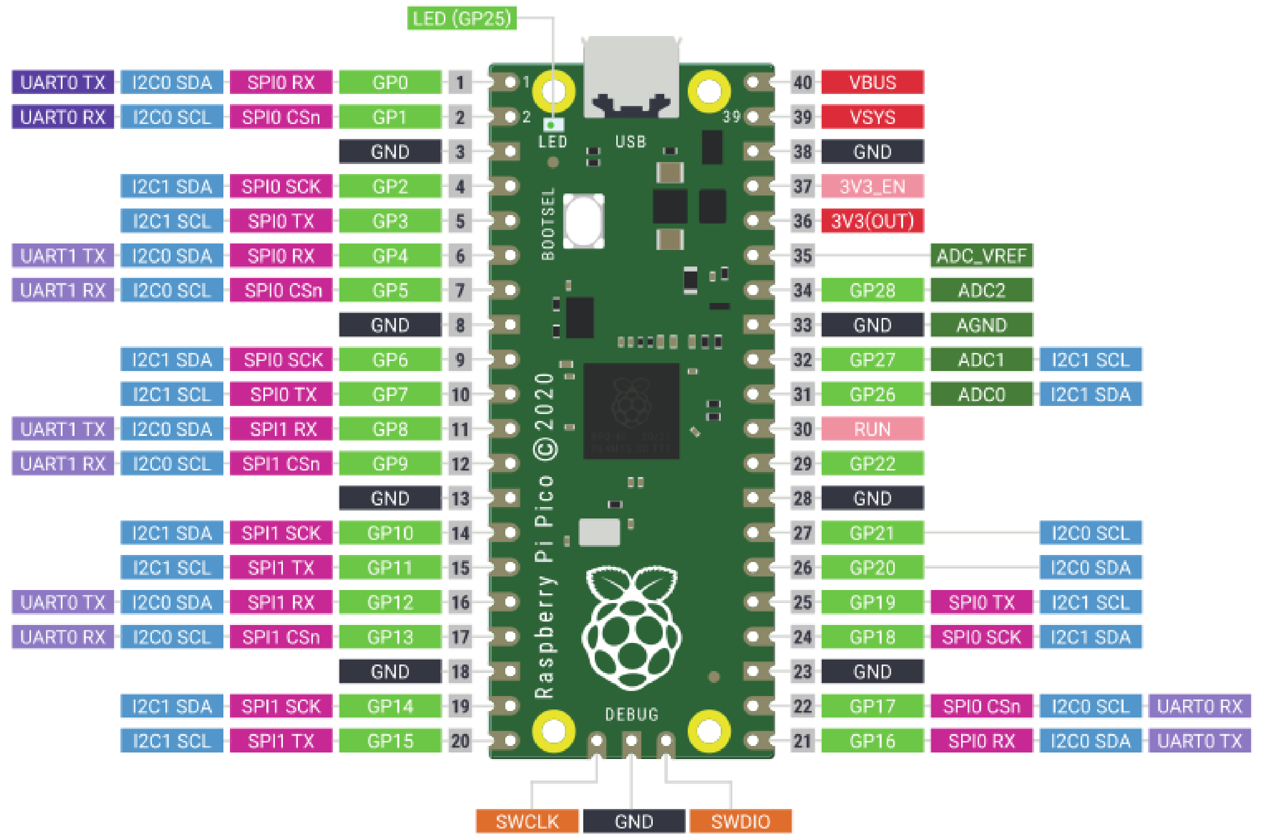



The hardware interfaces are distributed as follows:




| Frame color | Description |
|---|---|
| | Pins |
| | BOOTSE button |
| | USB port |
| | LED |
| | Debugging |

Function definition of pins:




| Color | Pins | Color | Pins |
|---|---|---|---|
| | GND | | Power |
| | GPIO | | ADC |
| | UART(defualt) | | UART |
| | SPI | | I2C |
| | System Control | | Debugging |

For details: https://datasheets.raspberrypi.org/pico/pico-datasheet.pdf

| | |
|---|---|
| GND | Ground Pin |
| Power | VBUS(microUSB Voltage)、VSYS(2-5VDC Input)、3V3(3.3V OUT)、3V3_EN(Enables Pico) |
| System Control | Run(Start or disable RP2040 microcontroller or reset) |
| ADC | Raspberry Pi Pico has a total of 5 ADC with a resolution of 12 bits, which are ADC0(GP26)， ADC1(GP27)， ADC2(GP28)， ADC3(GP29), ADC4 respectively. Among them, ADC3(GP29) is used to measure the VSYS on Pico board; ADC4 is derectly connected to the RP2040's built-in tempertature sensor. ADC_VREF can connect to external accurate voltmeter as ADC reference. ADC_GND pin is used as the reference point for grounding. |
| PWM | There are 16 PWM channels on Raspberry Pi Pico, each of which can control frequency and duty cycle independently. The GPIO pins are switched to PWM function. |
| UART | There are 2 UART: UART0， UART1. |
| SPI | There are 2 SPI: SPI0， SPI1. |
| I2C | 2 I2C: I2C0， I2C1. |
| Debugging | It is used when debugging code. |

## UART, I2C, SPI Defalt Pin

### UART

| Function | Default |
|---|---|
| UART_BAUDRATE | 115200 |
| UART_BITS | 8 |
| UART_STOP | 1 |
| UART0_TX | Pin 0 |
| UART0_RX | Pin 1 |
| UART1_TX | Pin 4 |
| UART1_RX | Pin 5 |

### I2C

| Function | Default |
|---|---|
| I2C Frequency | 400000 |
| I2C0 SCL | Pin 9 |
| I2C0 SDA | Pin 8 |
| I2C1 SCL | Pin 7 |
| I2C1 SDA | Pin 6 |

### SPI

| Function | Default |
|---|---|
| SPI_BAUDRATE | 1000000 |
| SPI_POLARITY | 0 |
| SPI_PHASE | 0 |
| SPI_BITS | 8 |
| SPI_FIRSTBIT | MSB |
| SPI0_SCK | Pin 6 |
| SPI0_MOSI | Pin 7 |
| SPI0_MISO | Pin 4 |
| SPI1_SCK | Pin 10 |
| SPI1_MOSI | Pin 11 |
| SPI1_MISO | Pin 8 |

For more detailed information, please refer to:
https://datasheets.raspberrypi.org/pico/raspberry-pi-pico-python-sdk.pdf

# Chapter 0 Getting Ready (Important)

Before starting building the projects, you need to make some preparation first, which is so crucial that you must not skip.

## 0.1 Installing Thonny (Important)

Thonny is a free, open-source software platform with compact size, simple interface, simple operation and rich functions, making it a Python IDE for beginners. In this tutorial, we use this IDE to develop Raspberry Pi Pico during the whole process.
Thonny supports various operating system, including Windows、Mac OS、Linux.

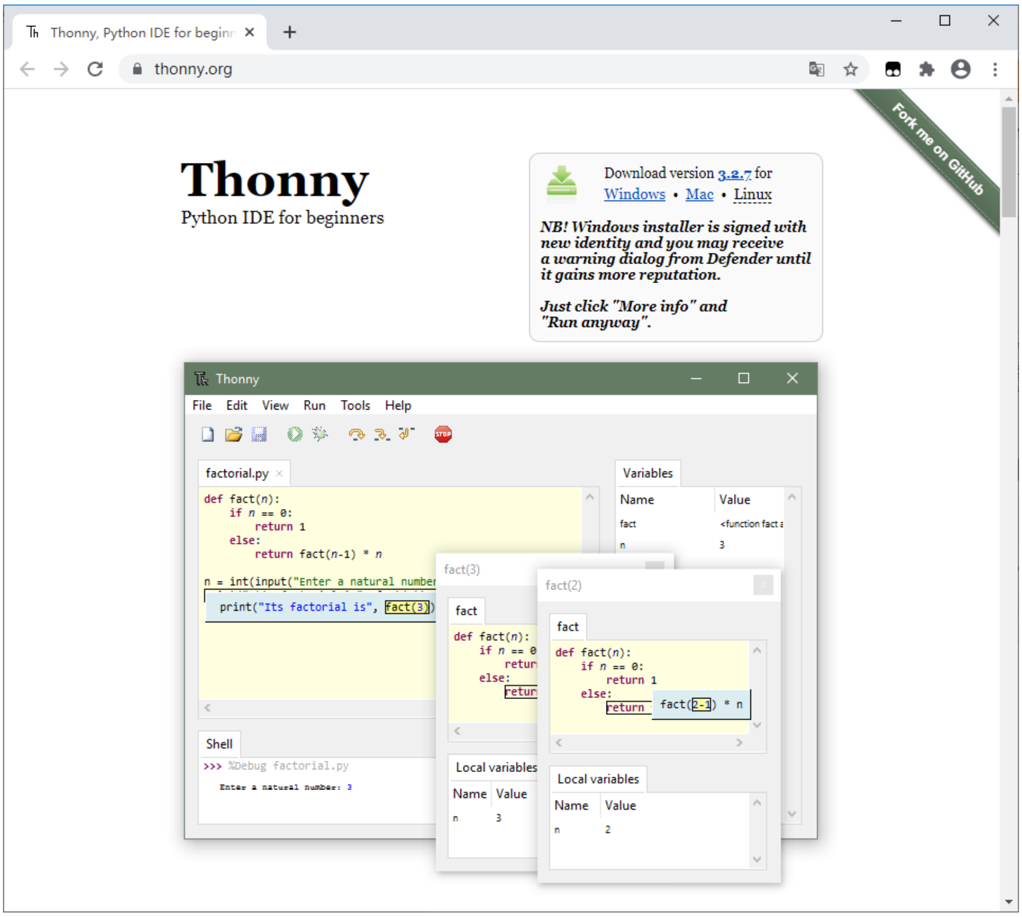
### Downloading Thonny

Official website of Thonny: https://thonny.org
Open-source code repositories of Thonny: https://github.com/thonny/thonny

Follow the instruction of official website to install Thonny or click the links below to download and install. (Select the appropriate one based on your operating system.)

| Operating System | Download links/methods |
|---|---|
| **Windows** | https://github.com/thonny/thonny/releases/download/v3.2.7/thonny-3.2.7.exe |
| **Mac OS** | https://github.com/thonny/thonny/releases/download/v3.2.7/thonny-3.2.7.pkg |
| **Linux** | **The latest version:**<br>**Binary bundle for PC (Thonny+Python):**<br>bash <(wget -O - https://thonny.org/installer-for-linux)<br><br>**With pip:**<br>pip3 install thonny<br><br>**Distro packages (may not be the latest version):**<br>**Debian, Rasbian, Ubuntu, Mint and others:**<br>sudo apt install thonny<br><br>**Fedora:**<br>sudo dnf install thonny |

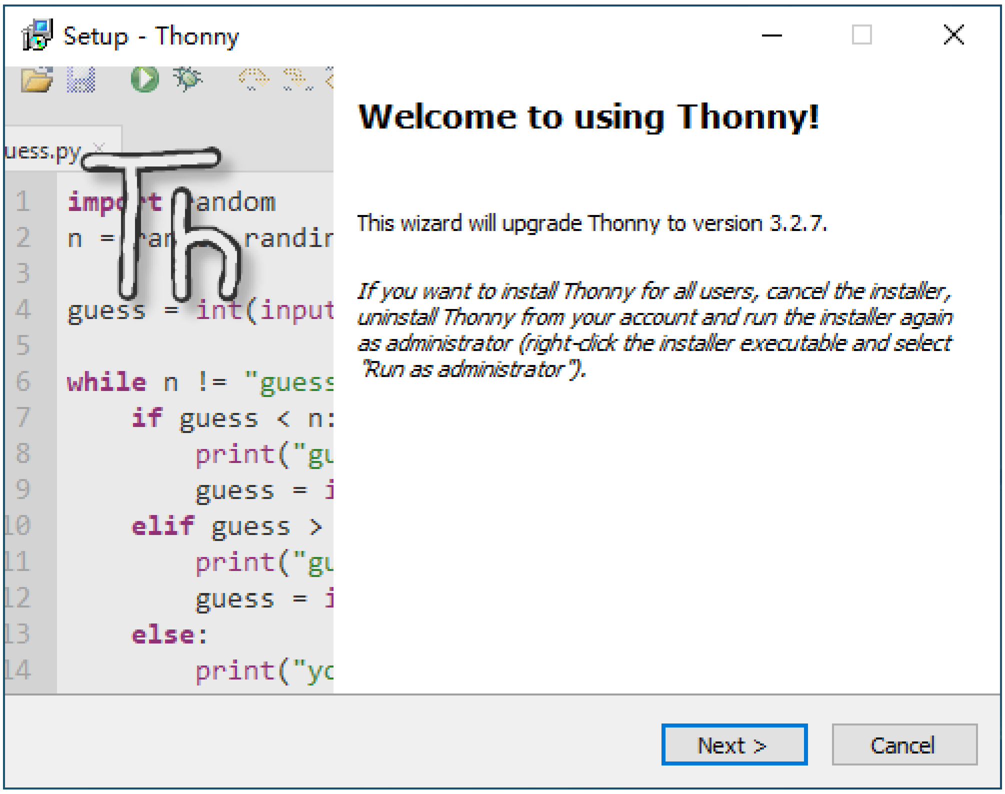
You can also open "**Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Python_Software**", we have prepared it in advance.




## Installing on Windows

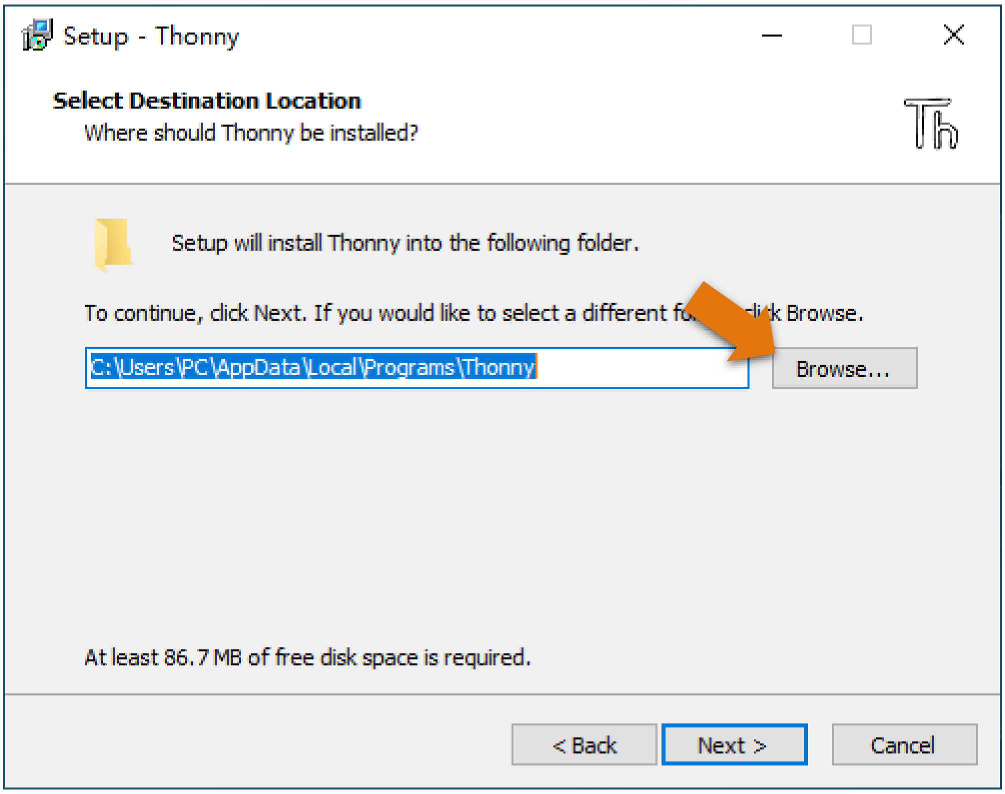
The icon of Thonny after downloading is as below. Double click "thonny-3.2.7.exe".

If you're not familiar with computer software installation, you can simply keep clicking "Next" until the installation completes.

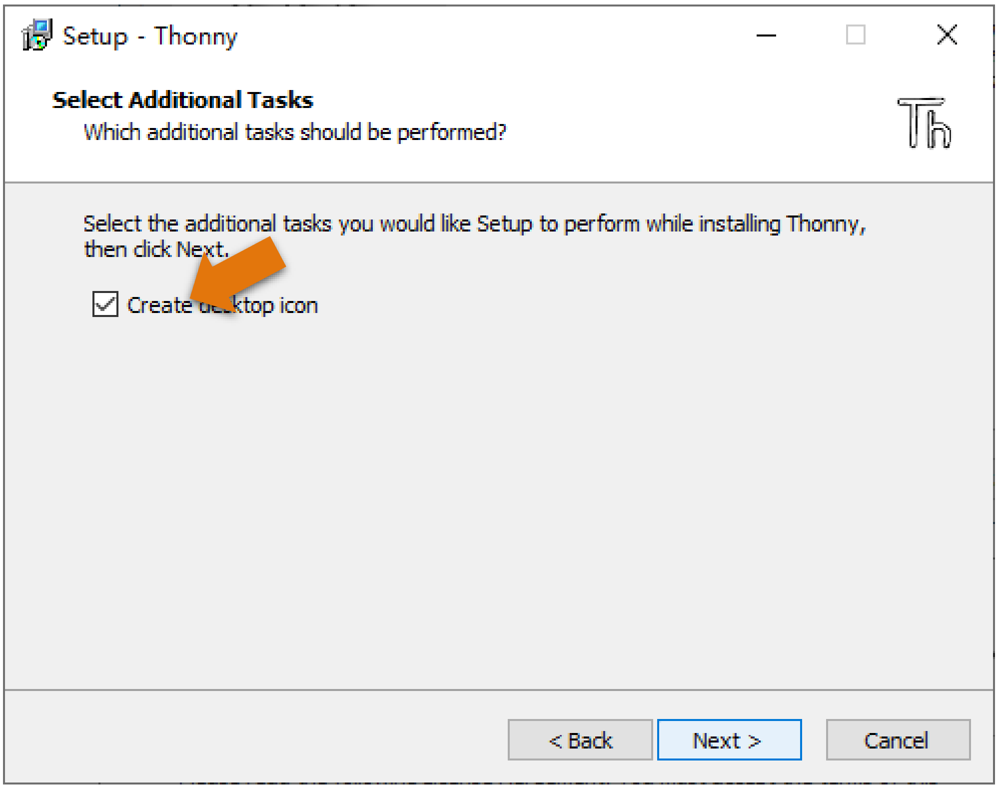



If you want to change Thonny's installation path, you can click "Browse" to modify it. After selecting installation path, click "OK".

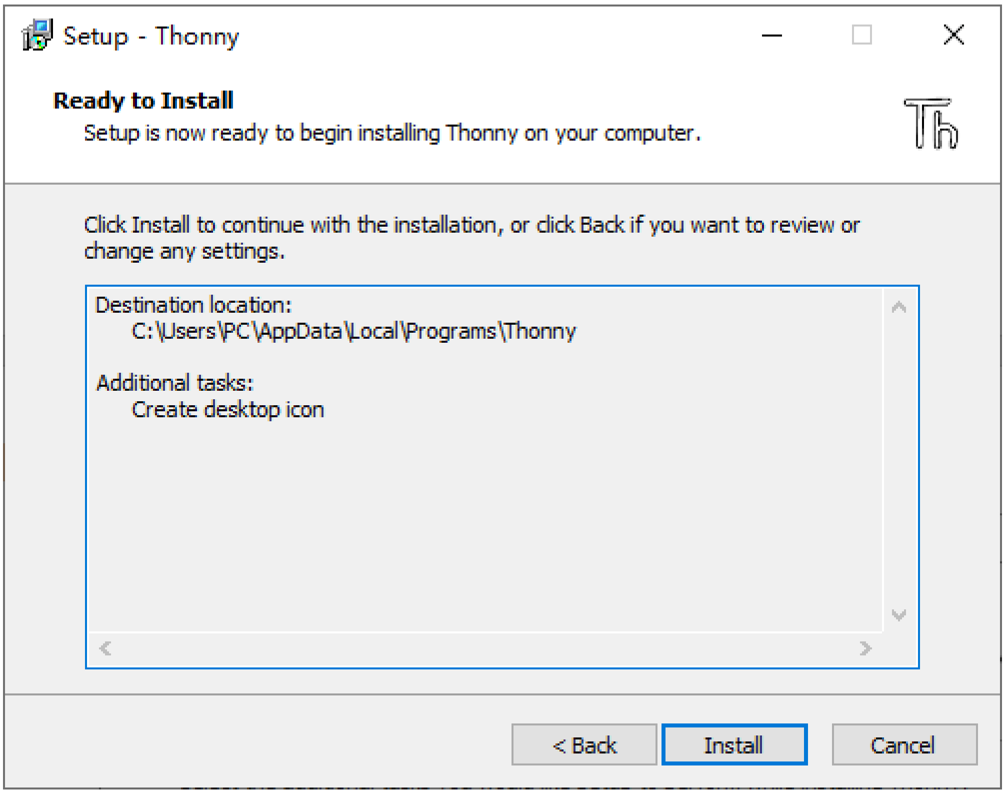
If you do not want to change it, just click "Next".



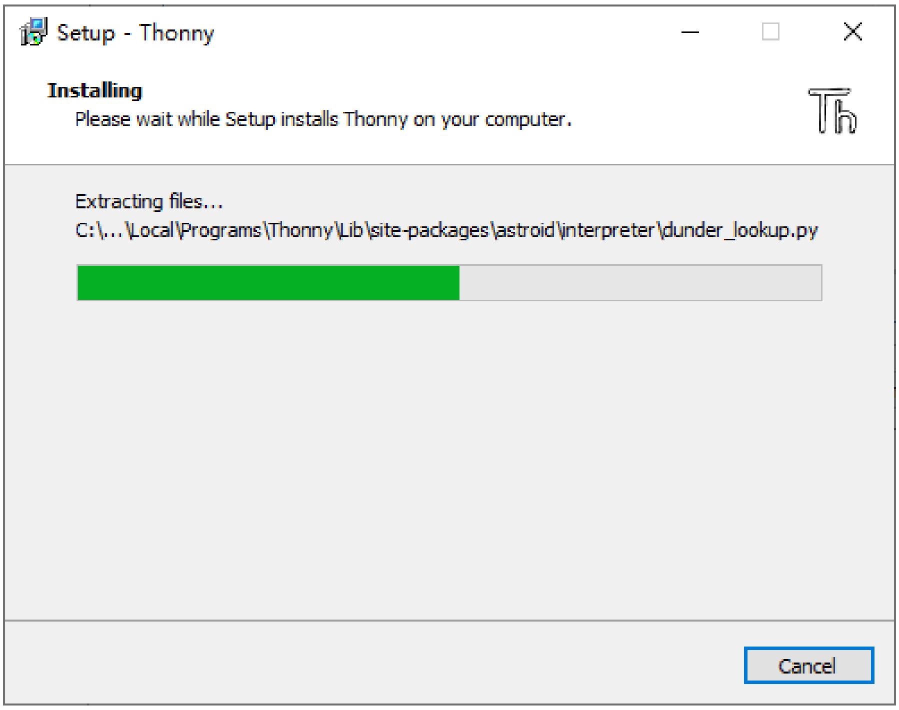
Check “Create desktop icon” and then it will generate a shortcut on your desktop to facilitate you to open Thonny later.



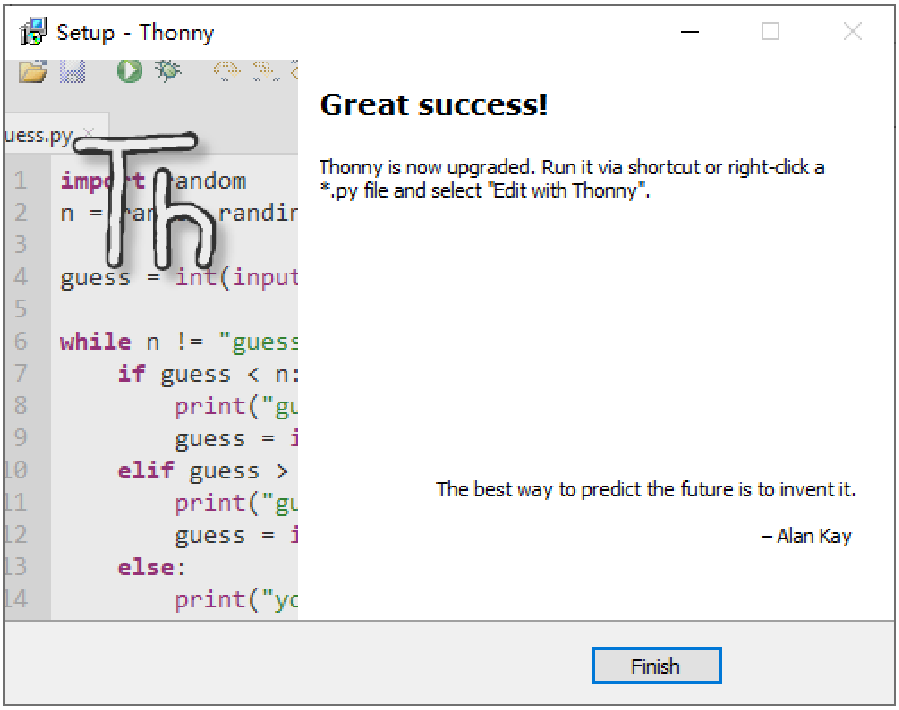

Click “install” to install the software.



During the installation process, you only need to wait for the installation to complete, and you msut not click "Cancel", otherwise Thonny will fail to be installed.



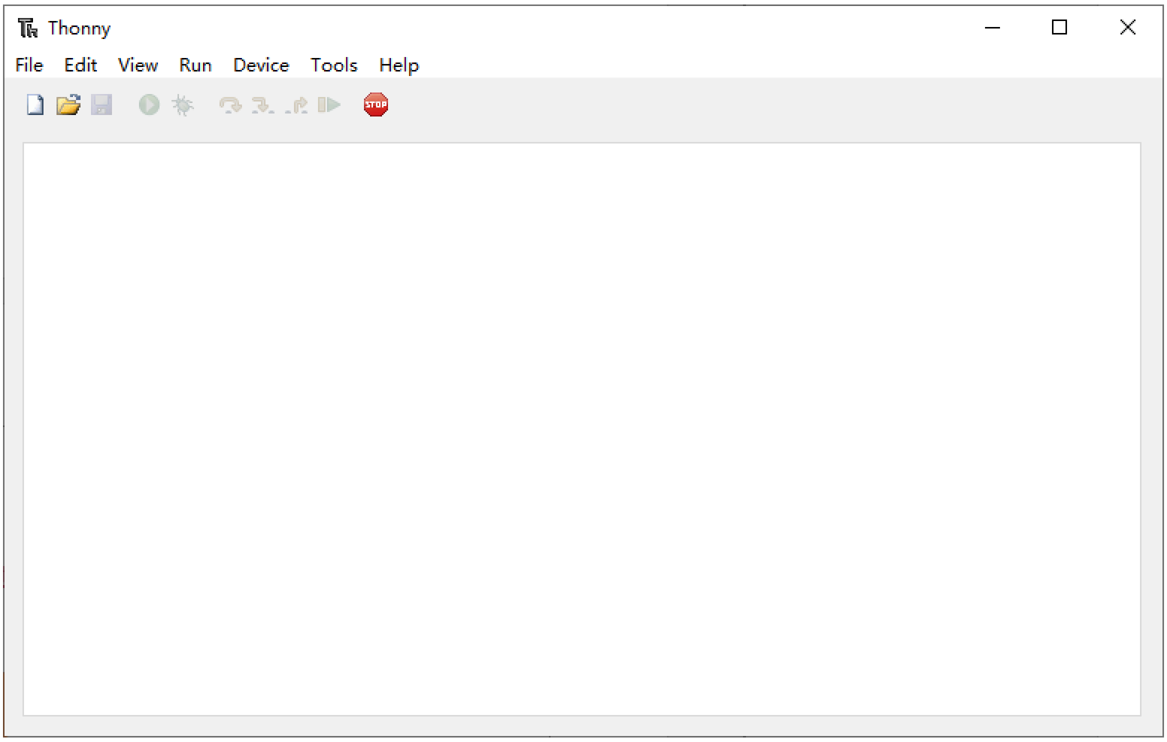

Once you see the interface as below, Thonny has been installed successfully.



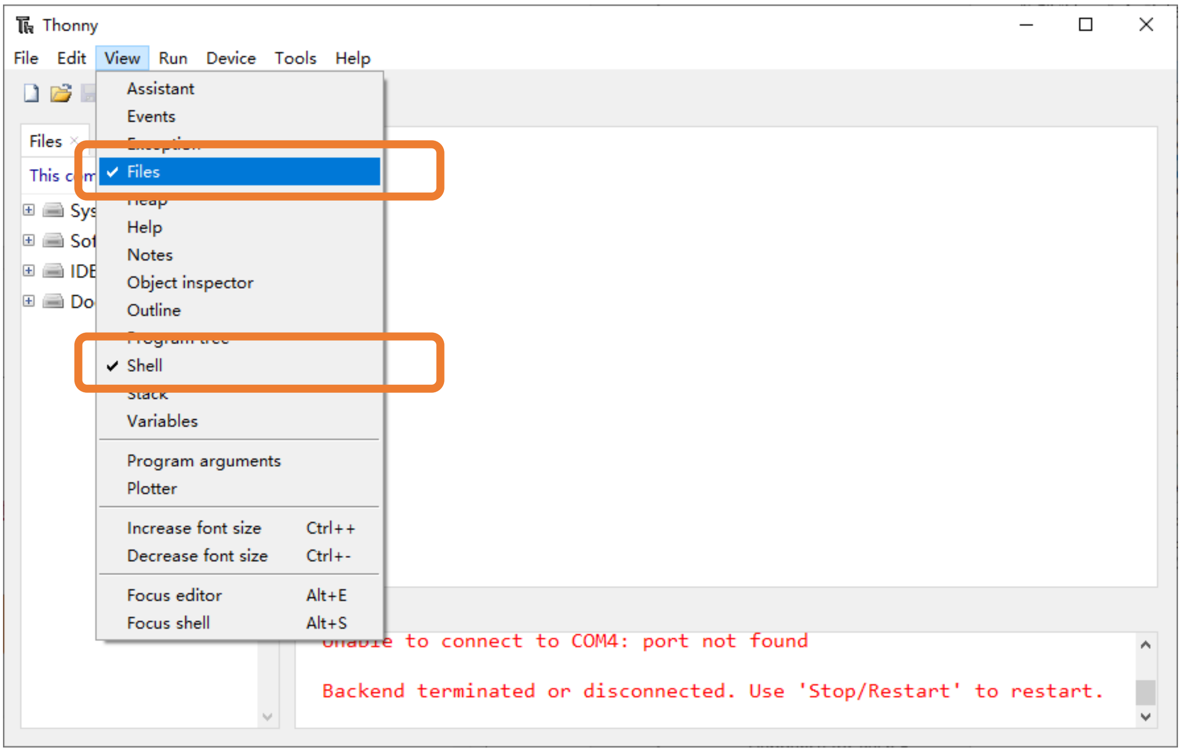

If you've check "Create desktop icon" during the installation process, you can see the below icon on your desktop.

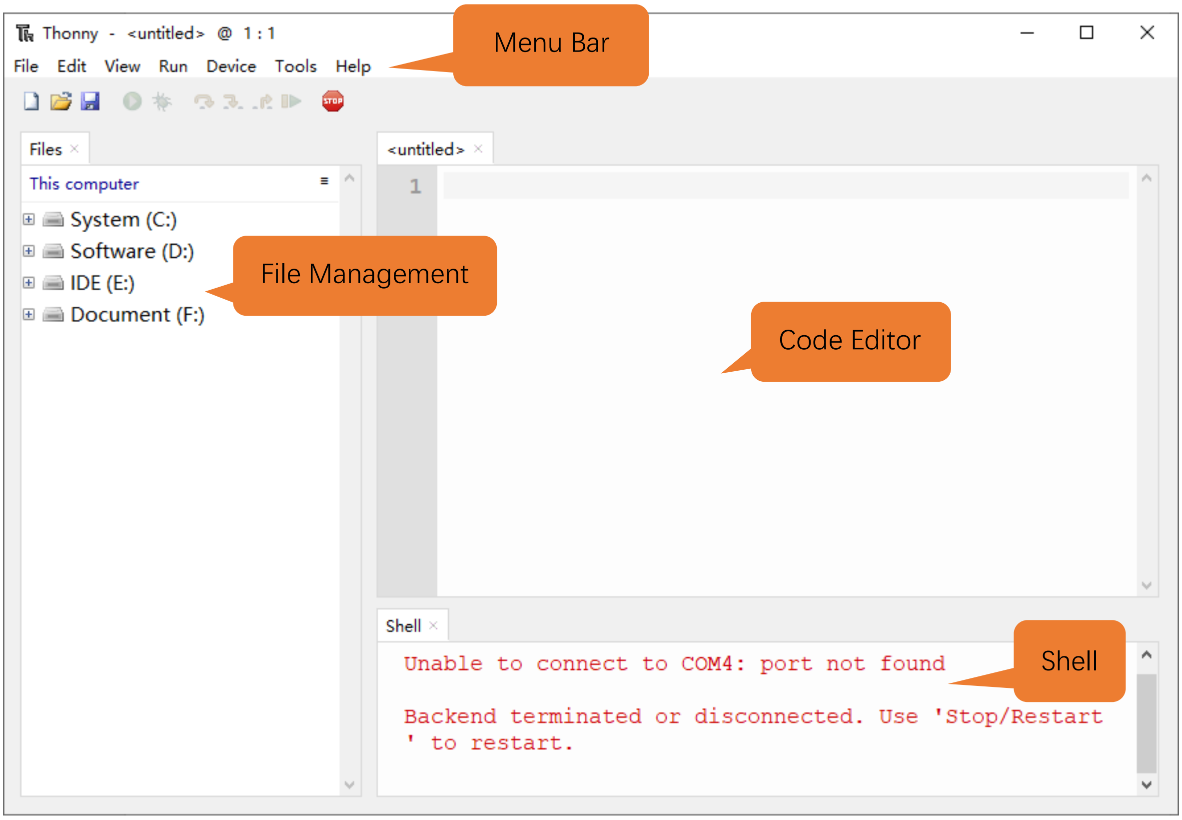
## 0.2 Basic Configuration of Thonny

Click the desktop icon of Thonny and you can see the interface of it as follows:




Select "View"→ "Files" and "Shell".




Thonny - <untitled> @ 1 : 1
File Edit View Run Device Tools Help
Menu Bar
Files
This computer
System (C:)
Software (D:)
IDE (E:)
Document (F:)
File Management
<untitled>
Code Editor
Shell
Unable to connect to COM4: port not found
Backend terminated or disconnected. Use 'Stop/Restart' to restart.
Shell

# 0.3 Burning Micropython Firmware (Important)

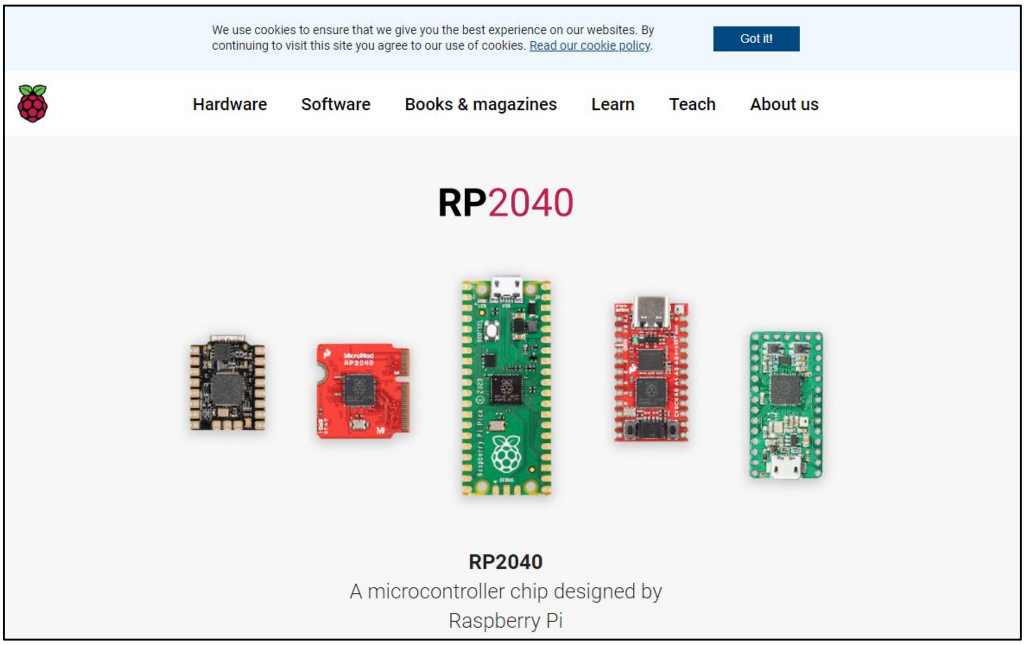
To run Python programs on Raspberry Pi Pico, we need to burn a firmware to Raspberry Pi Pico first.

## Downloading Micropython Firmware

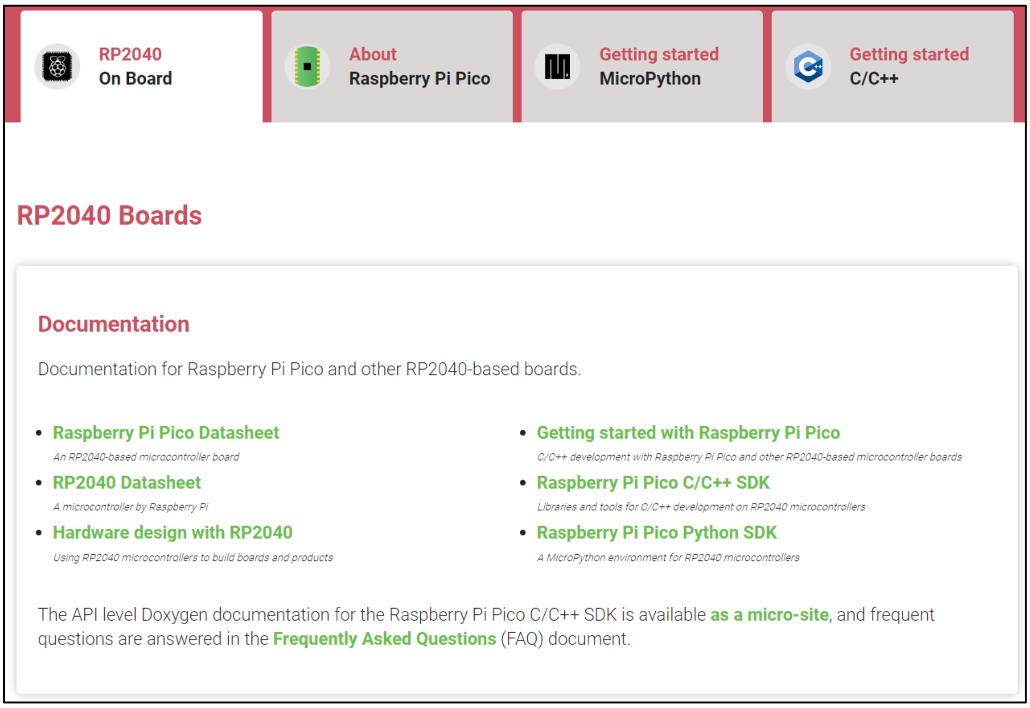
Option 1:

Raspberry Pi Pico official website: https://www.raspberrypi.org/documentation/rp2040/getting-started/

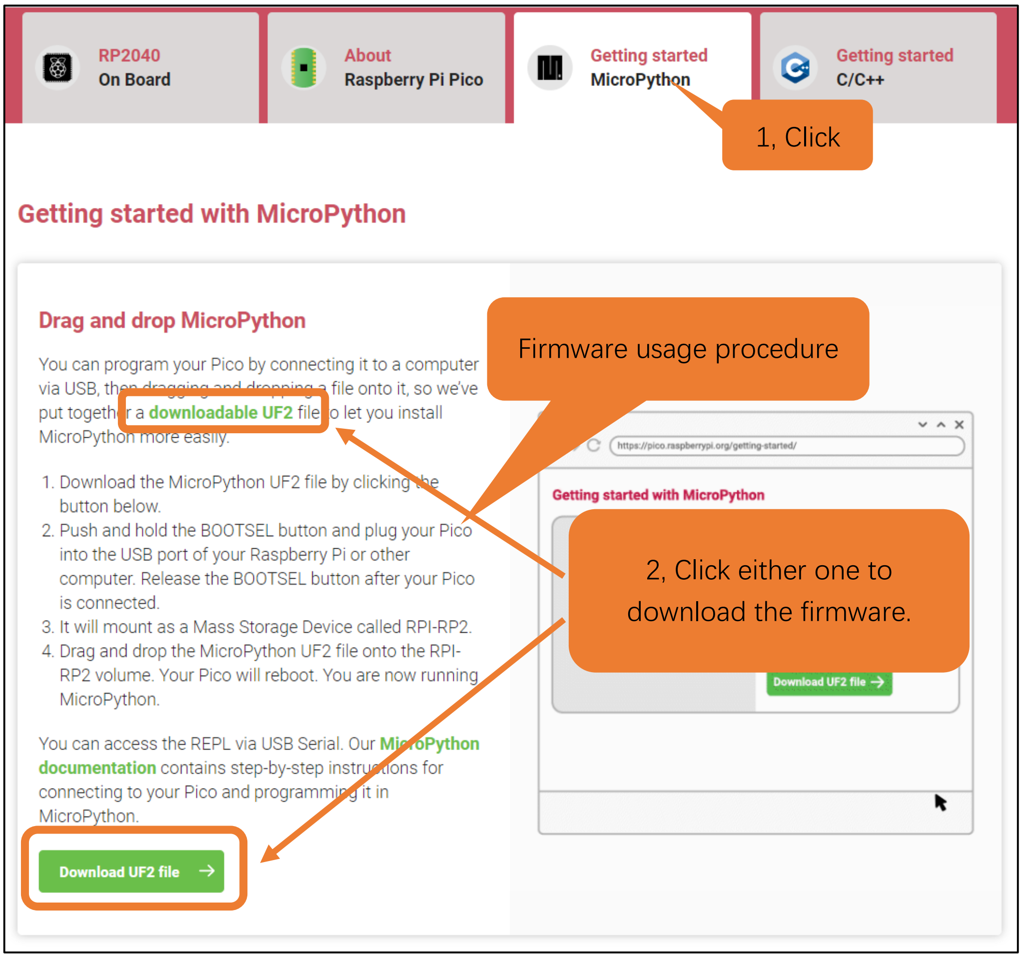
1， Click the link above and you can see the following interface




2， Roll the mouse and you can see the links for RP2040 documents.



3, Click "Getting started MicroPython" to enter the firmware downloading page.



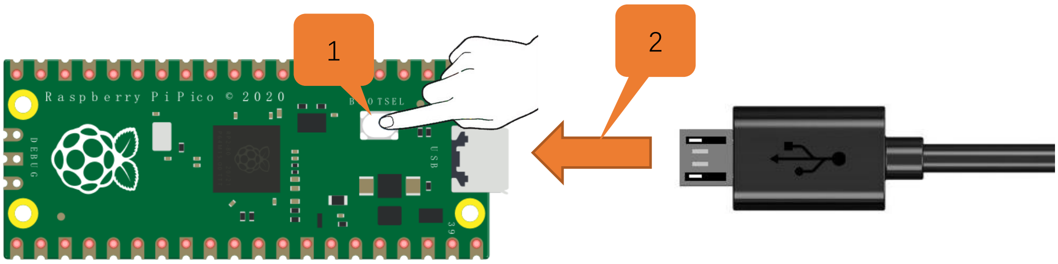

Option 2: Directly download via [rp2-pico-20210618-v1.16.uf2]

Option 3: If you cannot download it due to network issue or other reasons, you can use the one we have prepared, which locates at the following file path:

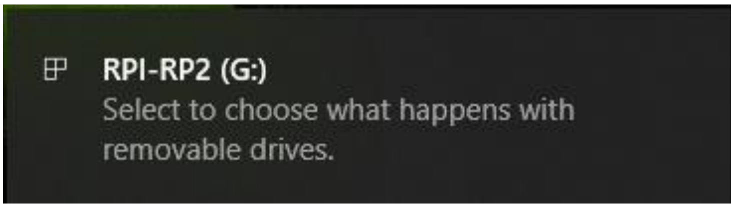
**Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Python_Firmware**

# Burning a Micropython Firmware

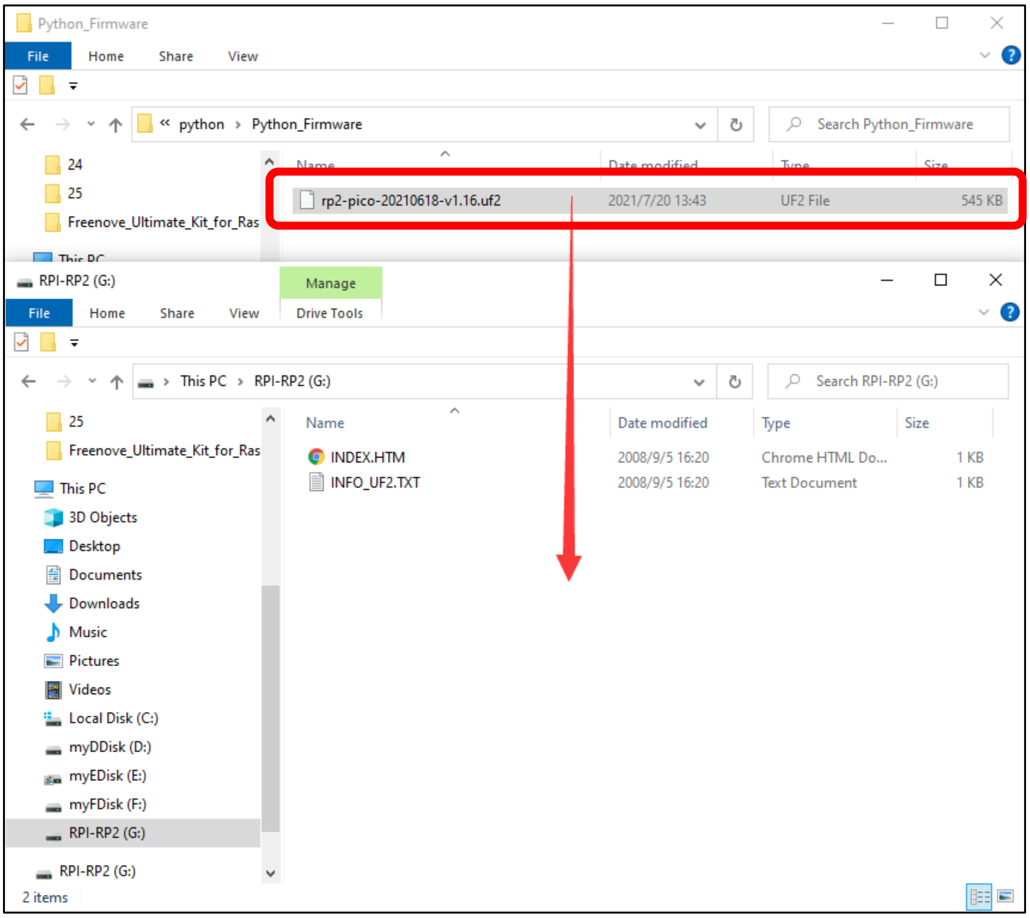
1. Connect a USB cable to your computer.
2. Long press BOOTSEL button on Raspberry Pi Pico and connect it to your computer with the USB cable.




3. When the connection succeeds, the following information will pop up on your computer.



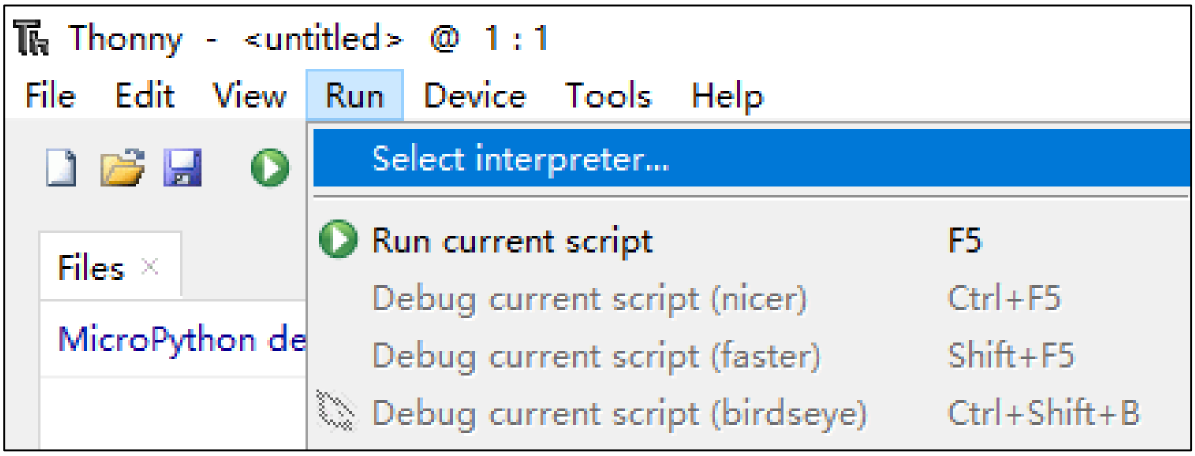

4. Copy the file(**rp2-pico-20210618-v1.16.uf2**) to RPI-RP2 and wait for it to finish, just like copy file to a U disk.



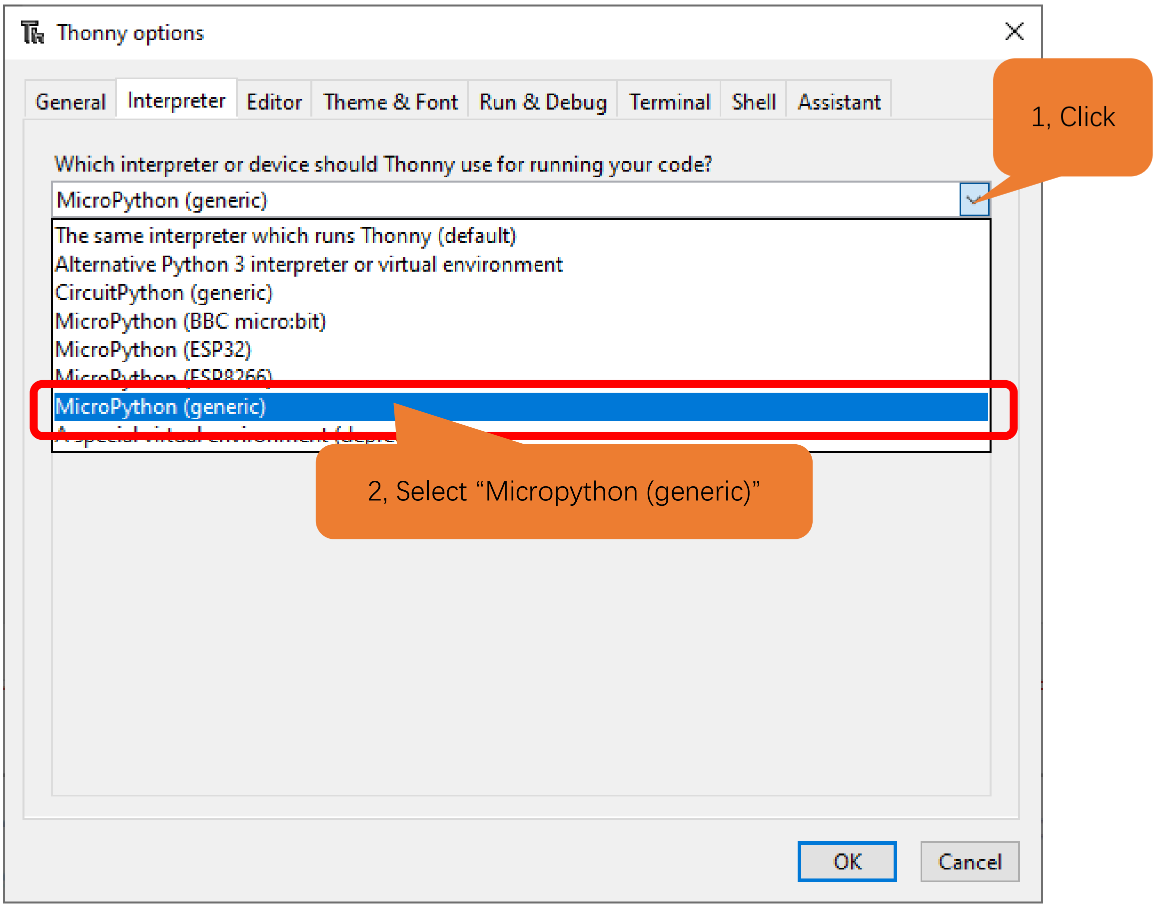

5. When the firmware finishes programming, Raspberry Pi Pico will reboot automatically. After that, you can run Micropython.

## 0.4 Thonny Connected to Raspberry Pi Pico

1. Open Thonny, click “run” and select “Select interpreter...”



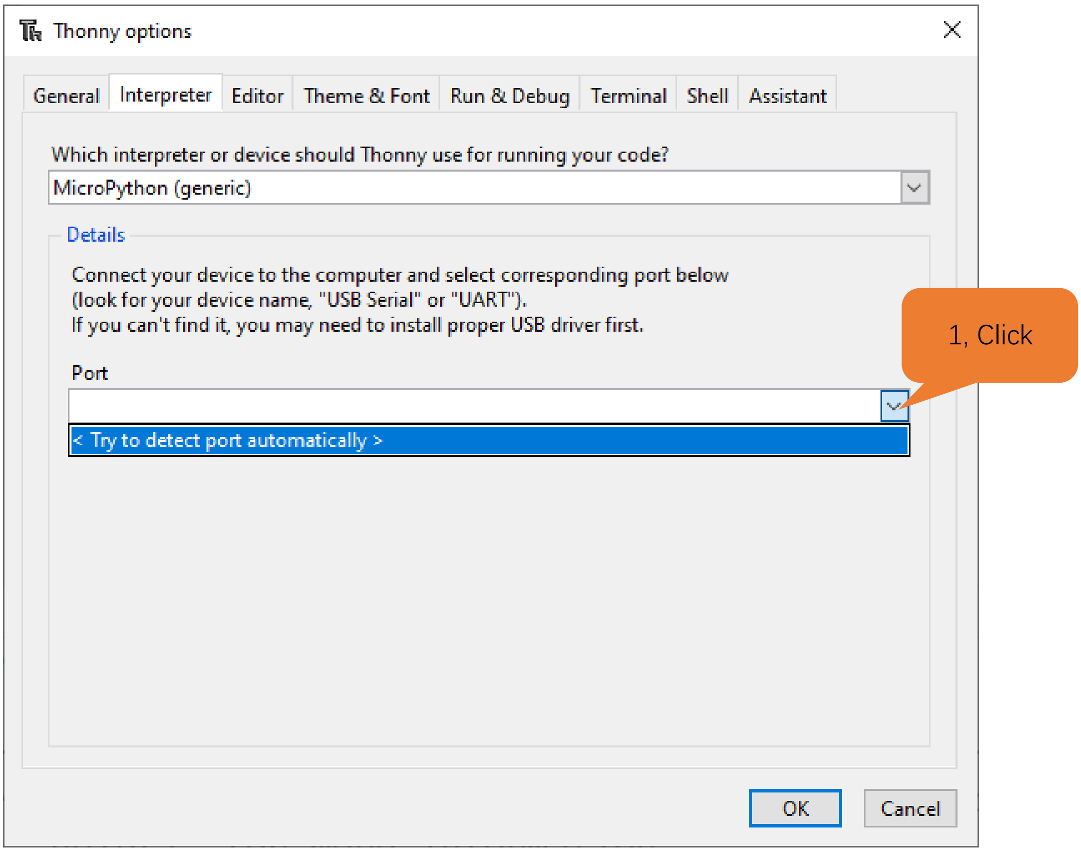

2. Select “Micropython (generic)” or “Micropython (Raspberry Pi Pico)”.
   How to select “Micropython (generic)”? As shown below:



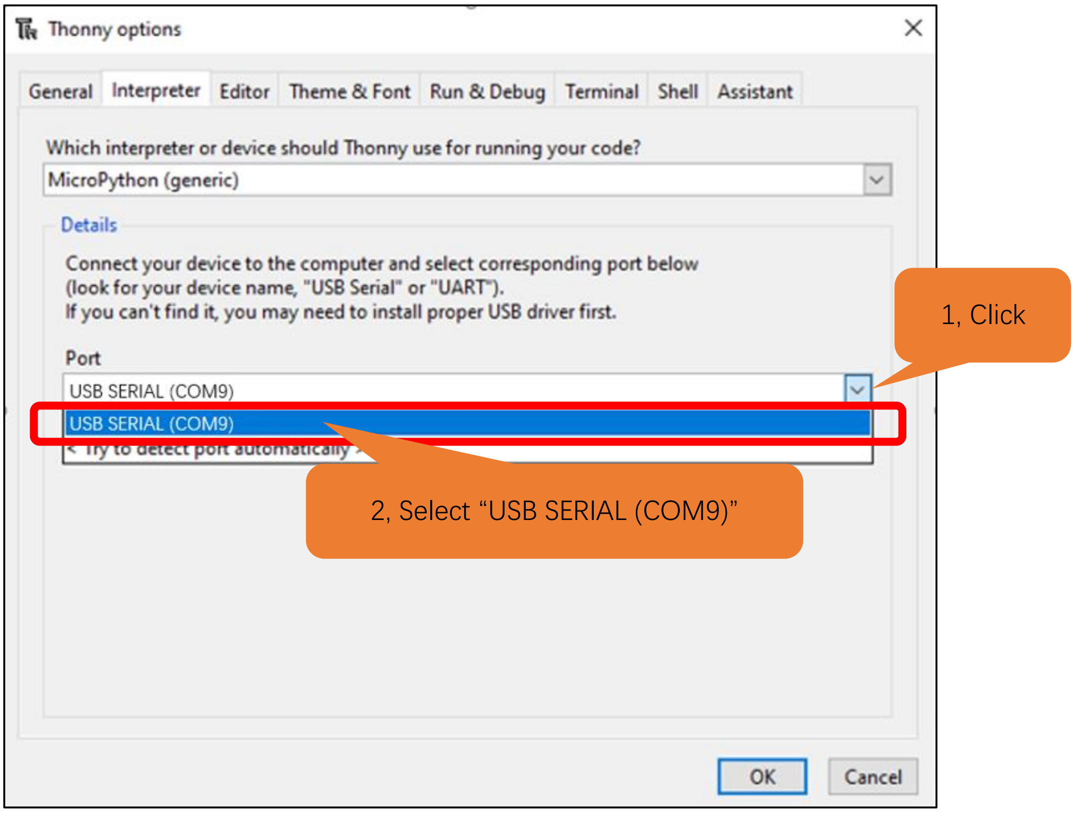
3. Select "USB-SERIAL (COMx)", The number x of COMx may varifies among different computers. You only need to make sure selecting USB-SERIAL (COMx).

How to determine the port on which your Raspberry Pi Pico communicates with your computer?

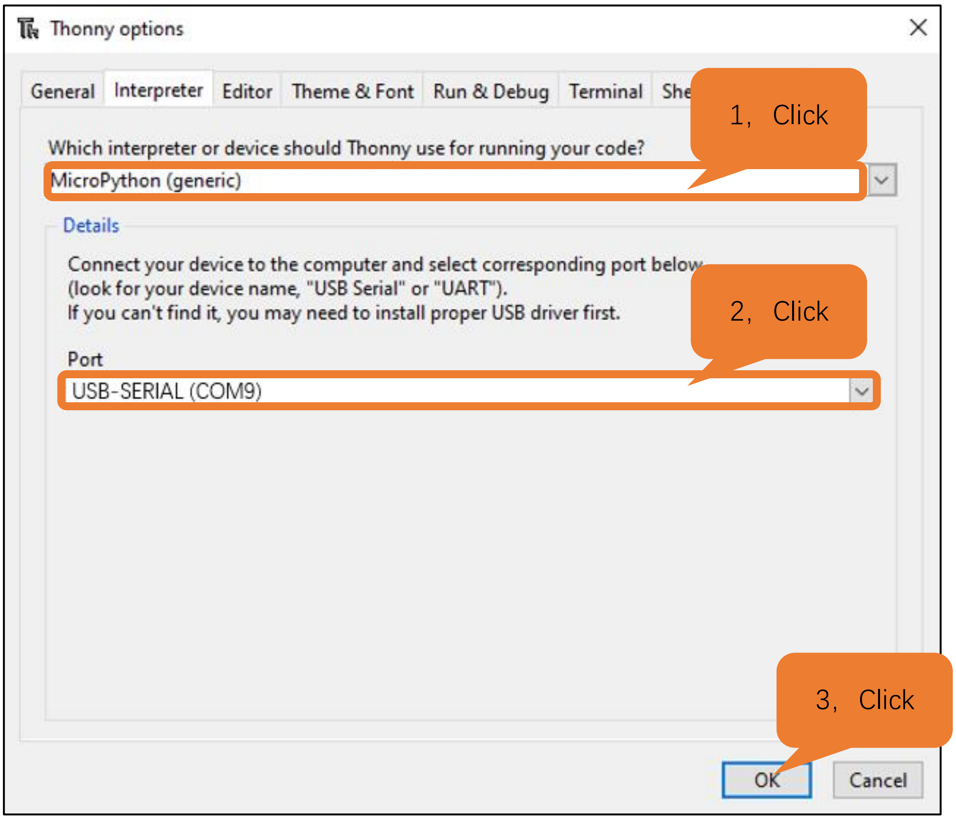
Step 1: When Pico **doesn't** connect to computer, open Thonny, click "Run", select "Select interpreter" and then a dialog box will pop up, click "Port" and you can check the ports currently connected to your computer, as shown below:



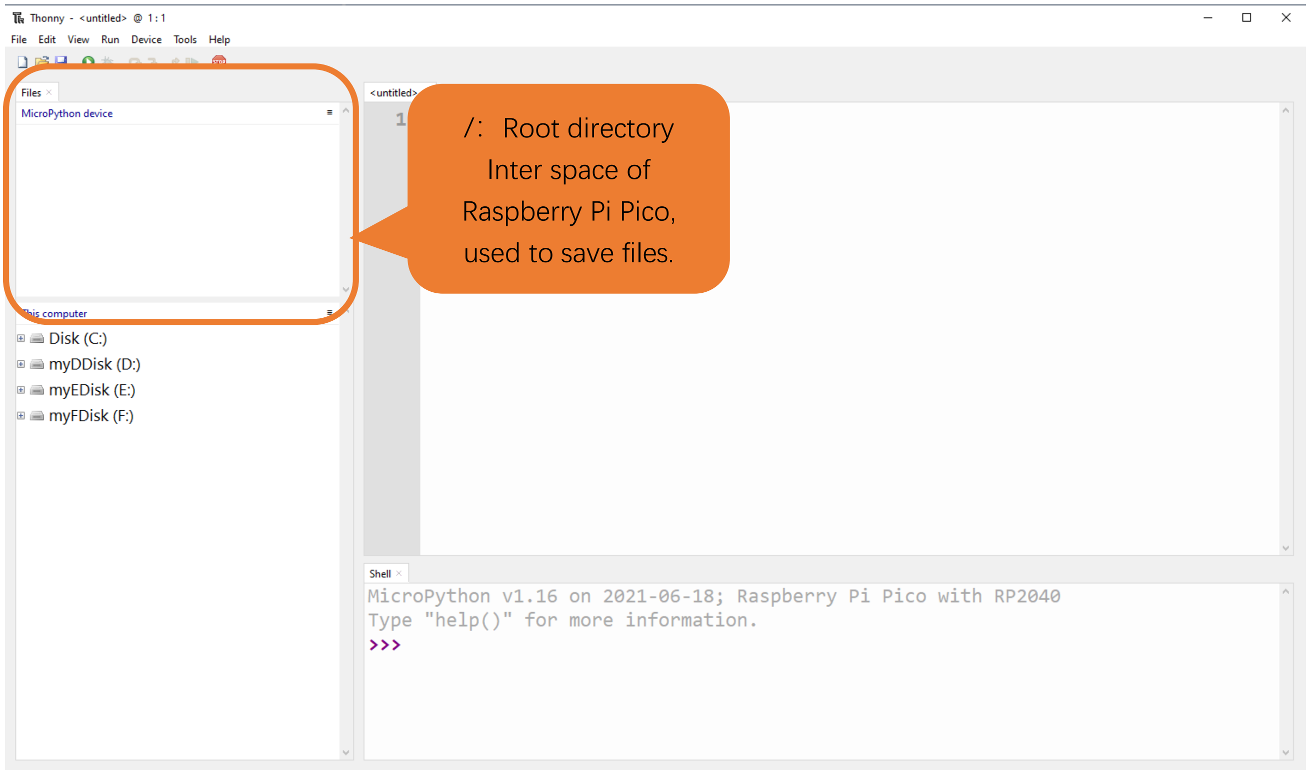
Step 2: Close the dialog box. Connect Pico to your computer, click "Run" again and select "Select interpreter". Click "Port" on the pop-up window and check the current ports. Now there is a newly added port, with which Pico commucates with the computer.




4. After selecting "Micropython (generic)" and port, click "OK"



5. When the following message displays on Thonny, it indicates Thonny has successfully connected to Pico.




So far, all the preparations have been made.

# 0.5 Testing codes (Important)

## Testing Shell Command

Enter “print(“hello world!”)” in “Shell” and press Enter.

## Running Online

To run Raspberry Pi Pico online, you need to connect it to computer. Users can use Thonny to compile or debug programs.

Advantages:

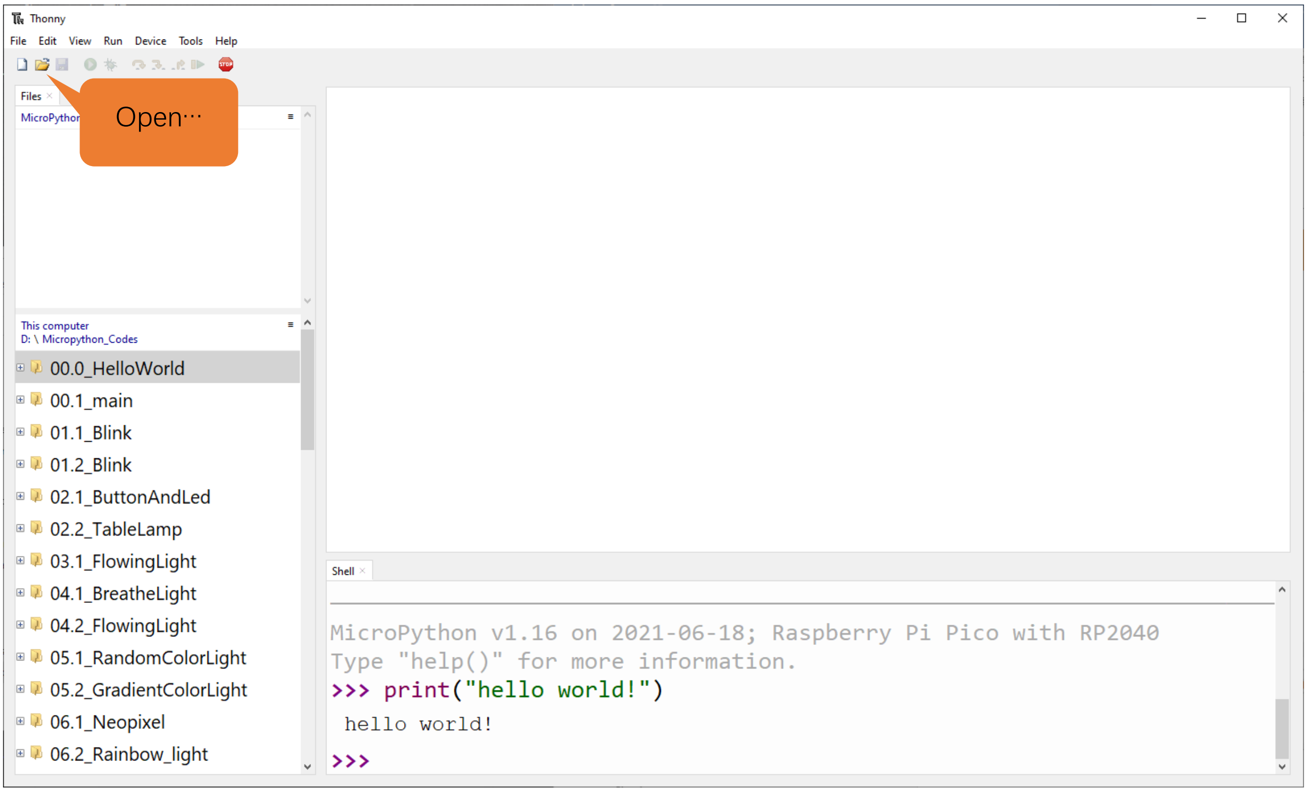
1. Users can use Thonny to compile or debug programs.
2. Through the "Shell" window, users can read the error information and output results generated during the running of the program and query related function information online to help improve the program.

Disvantages:

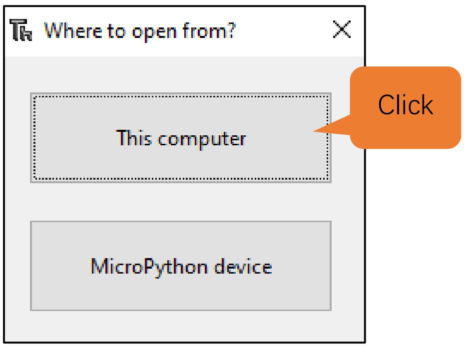
1. To run Raspberry Pi Pico online, you have to be connected to a computer and run with Thonny.
2. If Raspberry Pi Pico disconnects from computer, the program won't run again when they reconnect to each other.

### Opertation

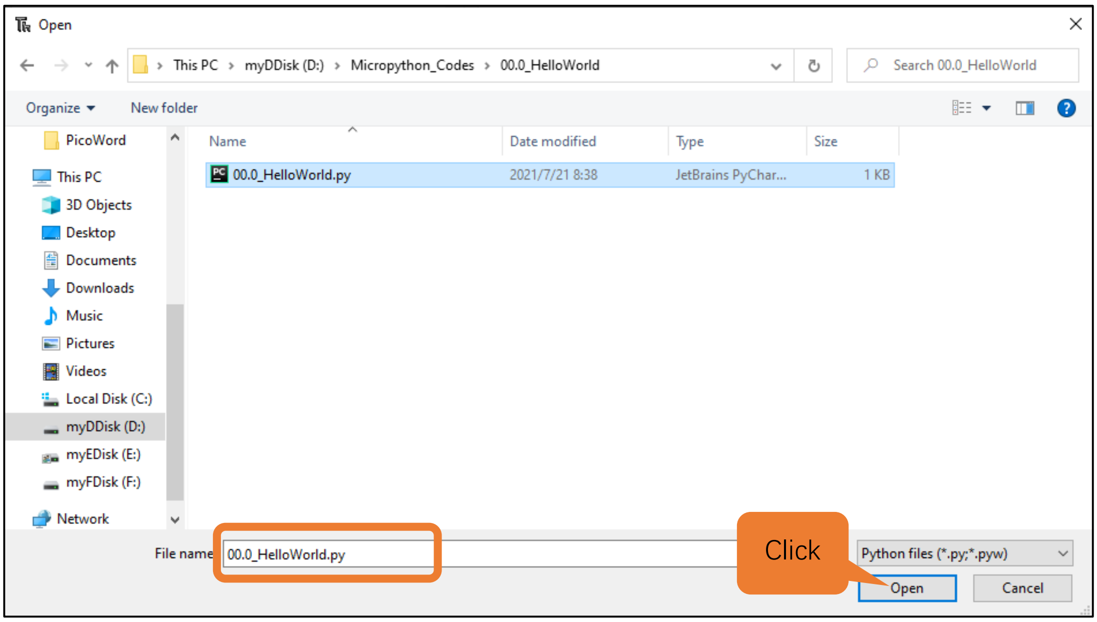
1. Open Thonny and click "Open…".



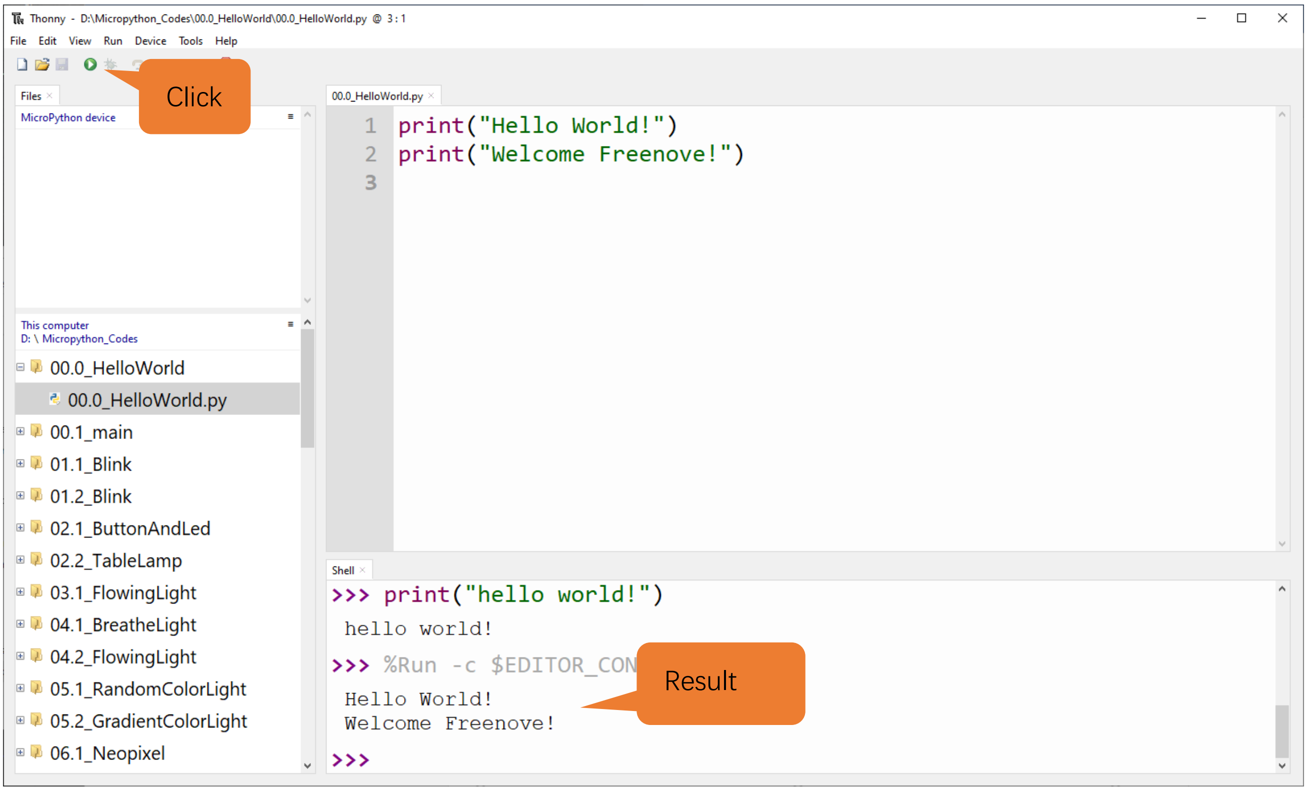

2. On the newly pop-up window, click "This computer".



In the new dialog box, select “**00.0_HelloWorld.py**” in “**Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/00.0_HelloWorld**” folder.




Click “Run current script” to execute the program and “Hello World!”, “Welcome Freenove” will be printed in “Shell”.




## Exiting Running Online

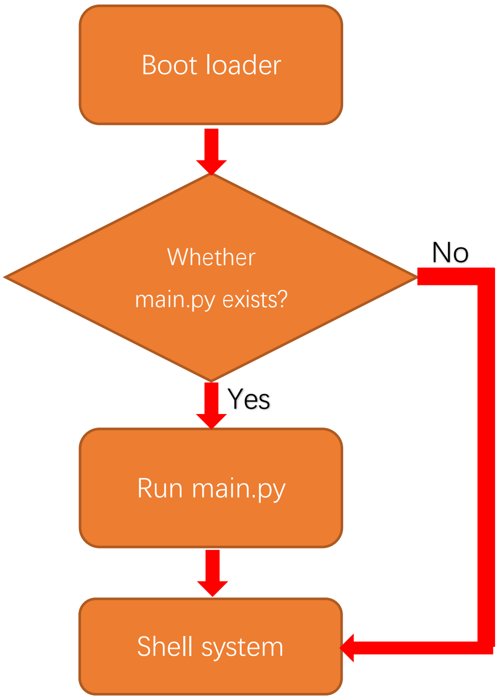
When running online, click “Stop /Restart backend” on Thonny or press Ctrl+C to exit the program.

## Running Offline

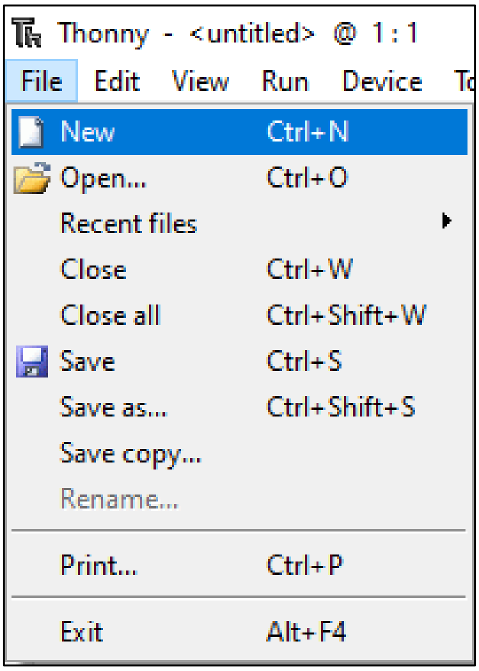
When running offline, Raspberry Pi Pico doesn't need to connect to computer and Thonny. It can run the programs stored in main.py on the device once powered up.
Advantage: It can run programs when powered up without connected to computer and Thonny.
Disvantage: The program will stop automatically when error occurs or Raspberry Pi Pico is out of power. Code cannot be changed easily.

### Operation

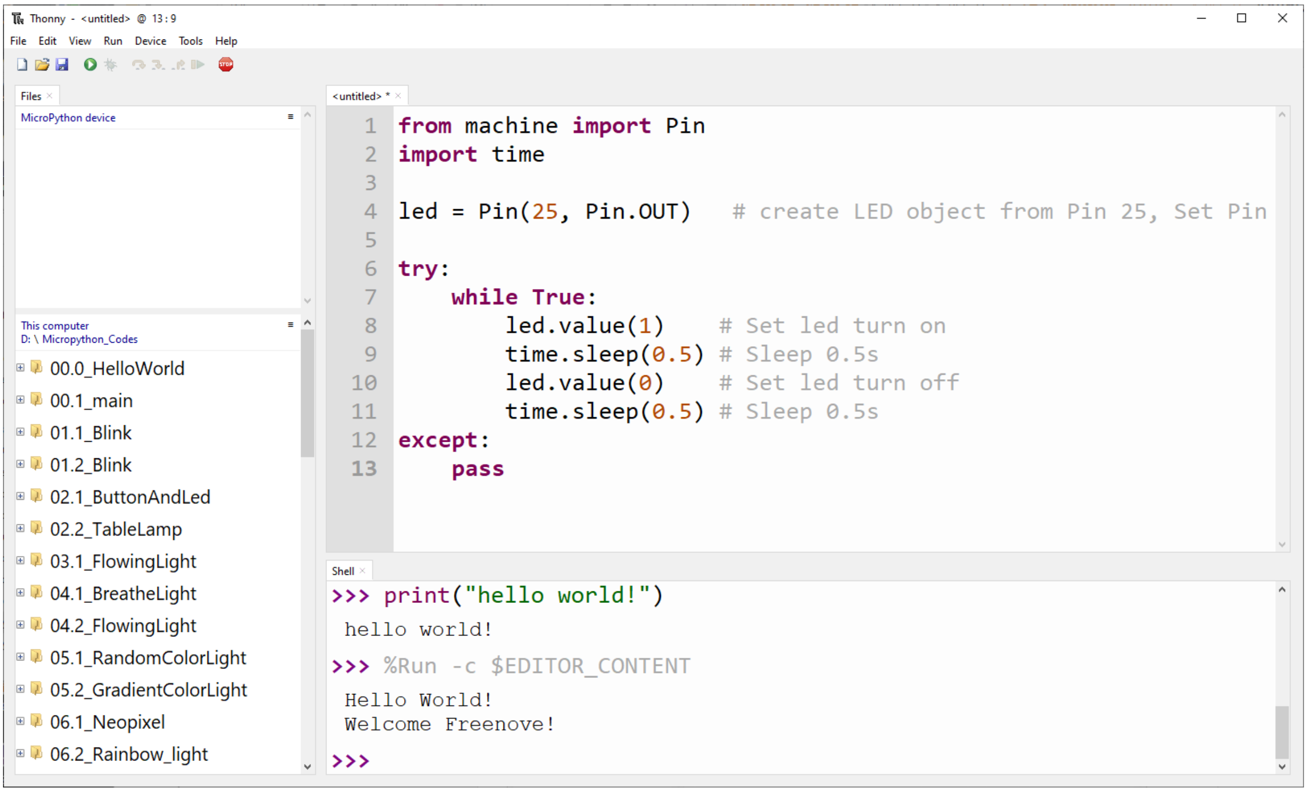
Once powered up, Raspberry Pi Pico will automatically check whether there is main.py existing on the device. If there is, it runs the programs in main.py and then enter shell command system. (If you want the code to run offline, you can save it as main.py); If main.py doesn't exist, it will enter shell command system directly.



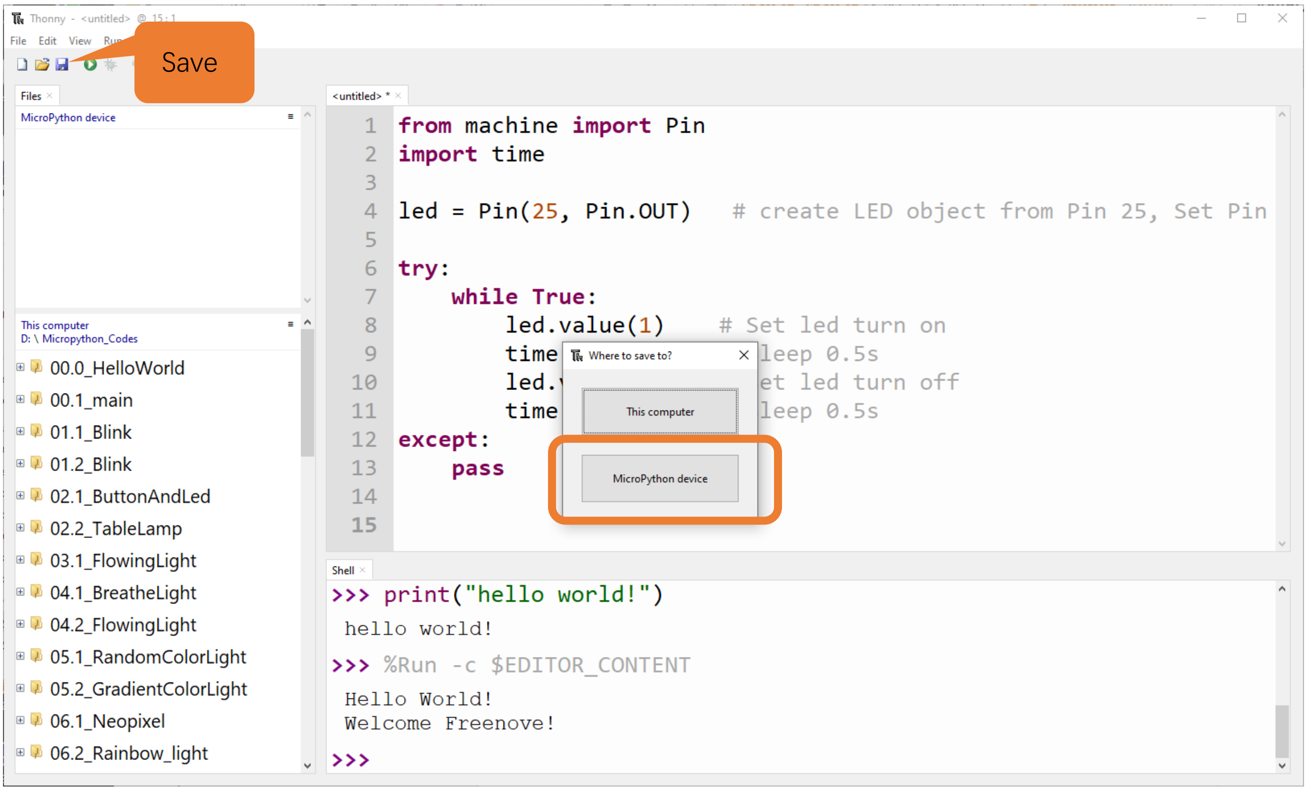
1. Click "File"→"New" to create and write codes.



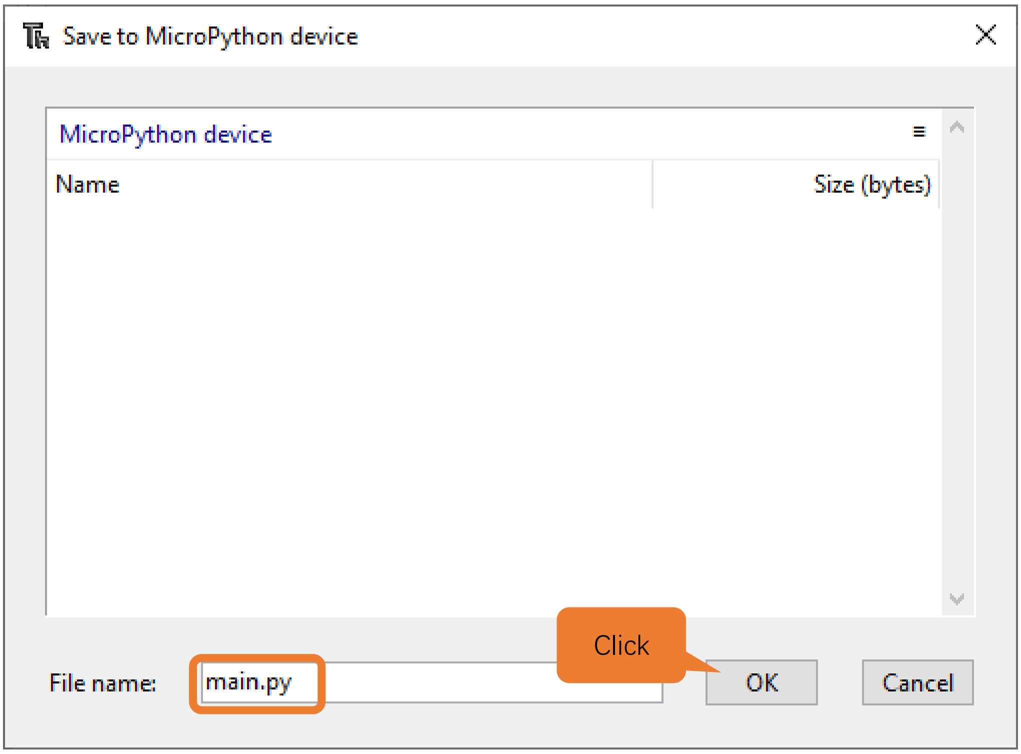

2. Enter codes in the newly opened file. Here we use codes of "01.1_Blink.py" as an example.



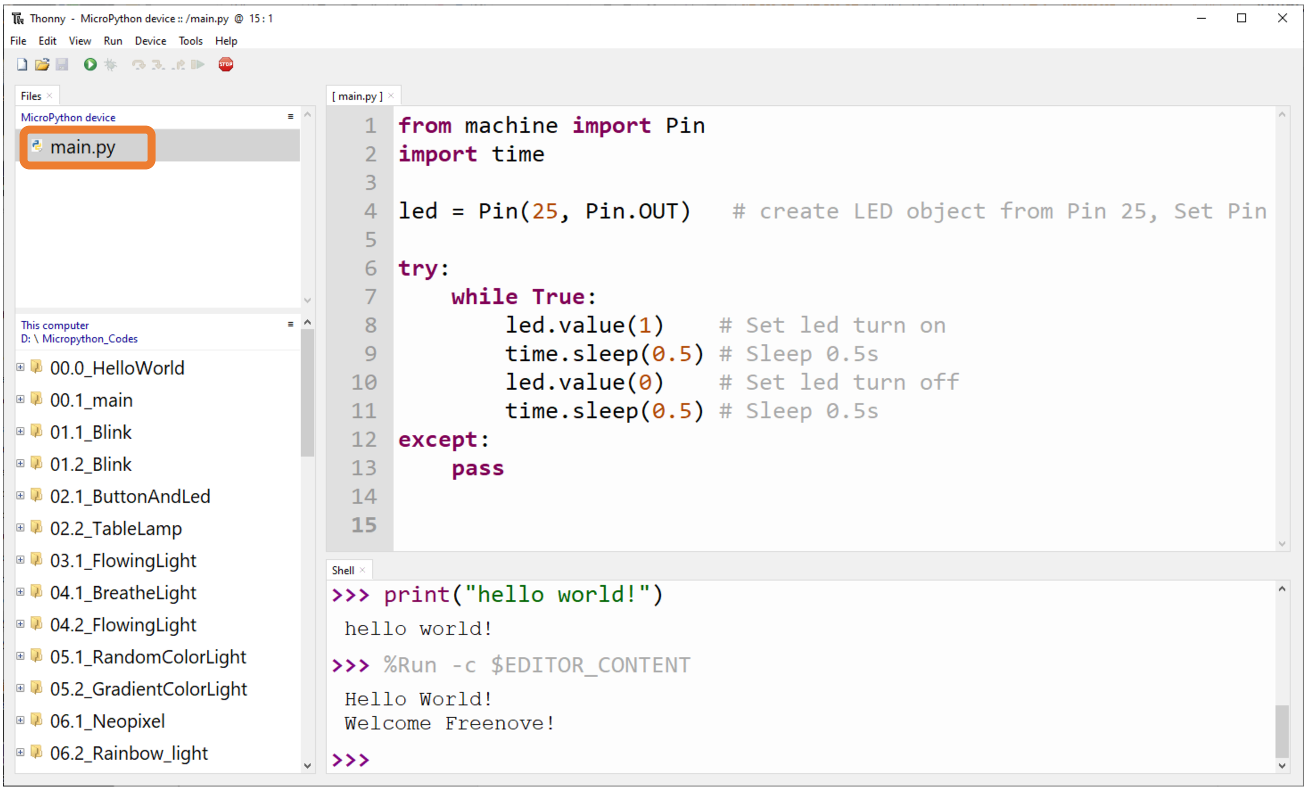
3. Click “Save” on the menu bar. You can save the codes either to your computer or to Raspberry Pi Pico.



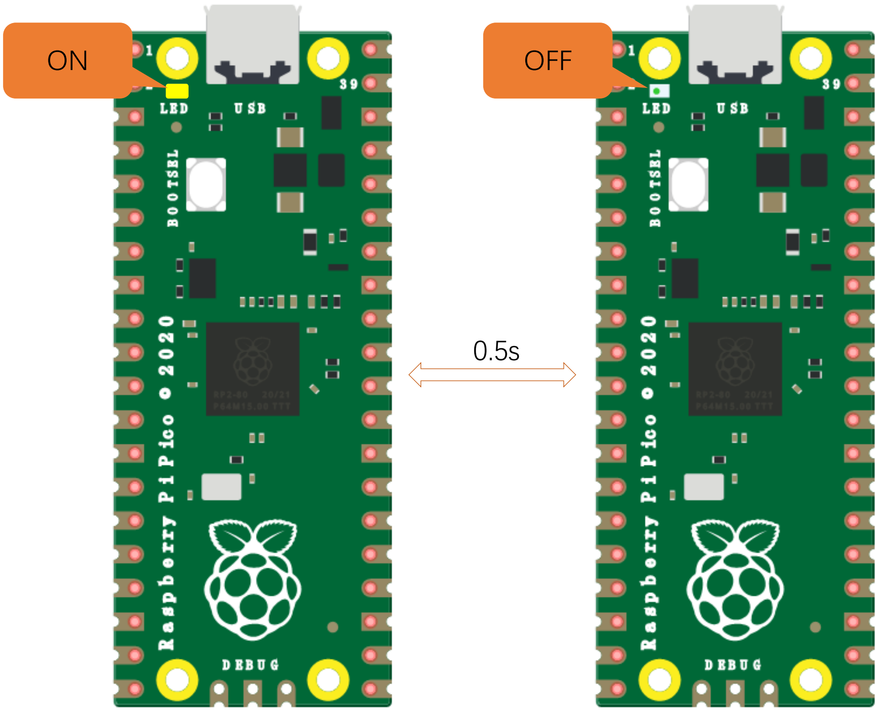

4. Select “MicroPython device”, enter “main.py” in the newly pop-up window and click “OK”.



5. You can see that codes have been uploaded to Raspberry Pi Pico.

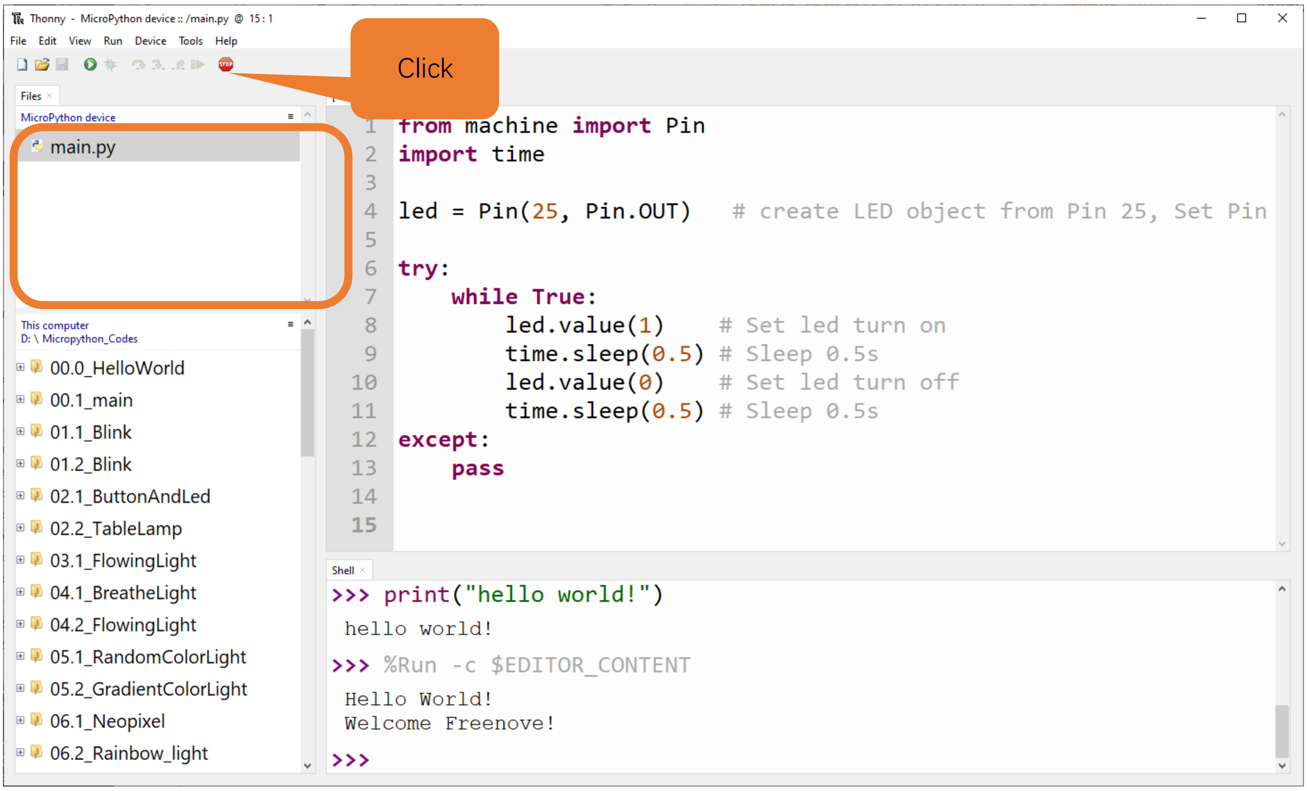



6. Disconnect Raspberry Pi Pico USB cable and then reconnect it, the LED on Raspberry Pi Pico will blink repeatedly.



## Exiting Offline Running

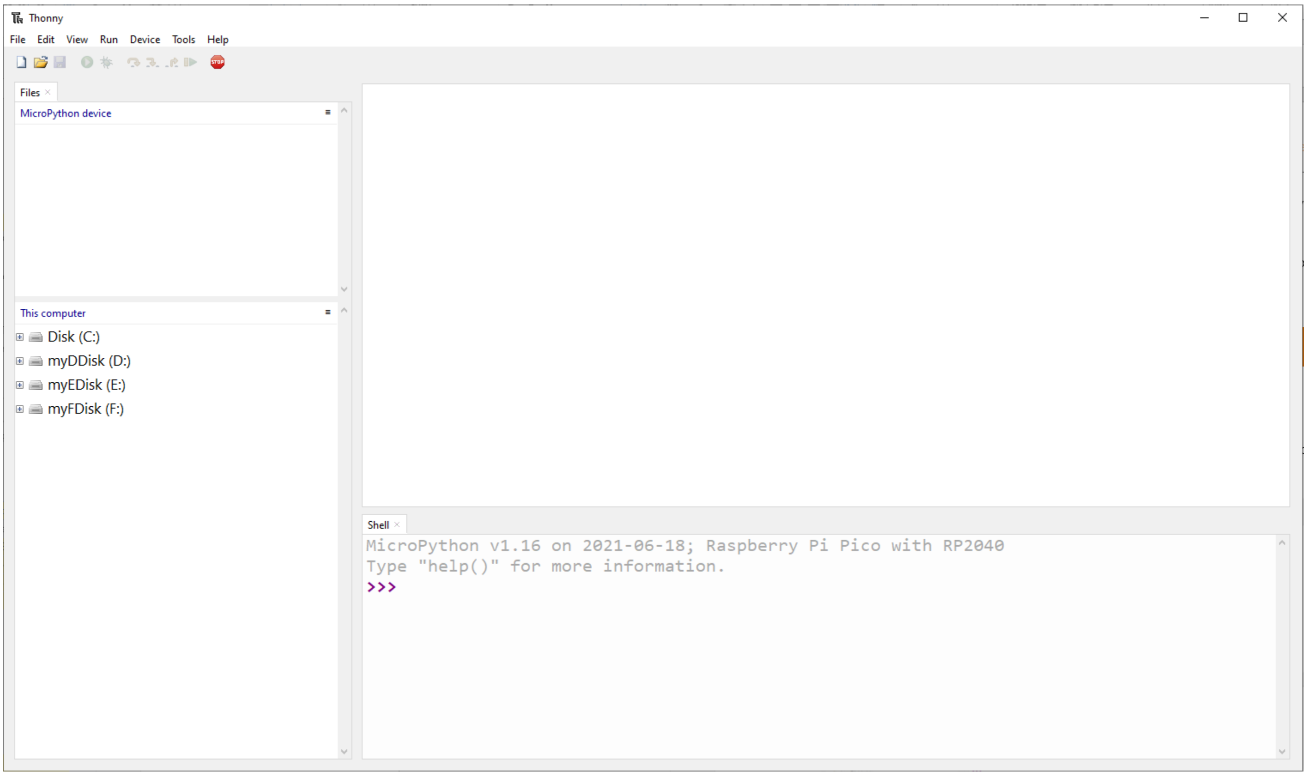
Connect Raspberry Pi Pico to computer, click “stop/restart backend” on Thonny to end running offline.



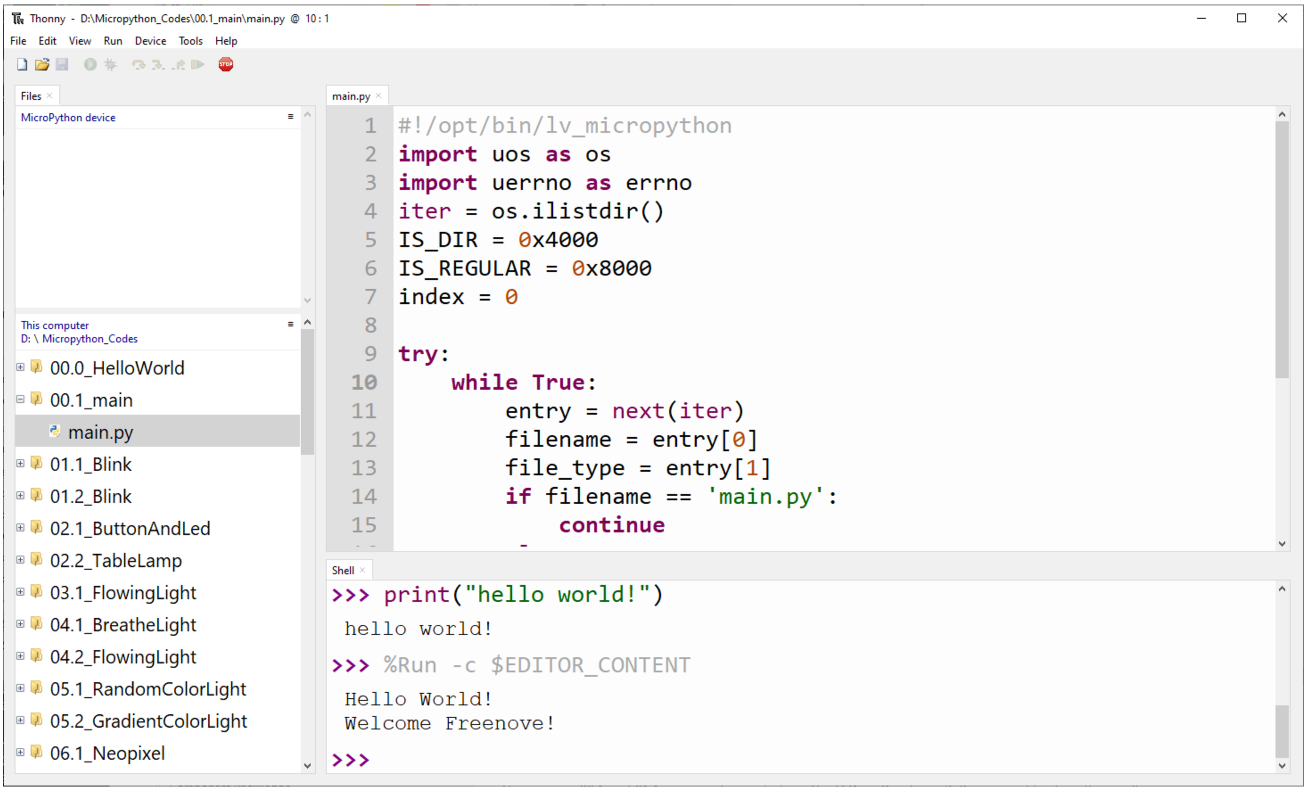

If it doesn’t work, please click on “stop/restart backend” for more times or reconnect Pico.

We provide a main.py file for running offline. The code added to main.py is a bootstrap that executes the user's code file. All you need to do is upload the offline project's code file (.py) to the Raspberry Pi Pico device.

1. Move the program folder "**Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**". Open "Thonny"。

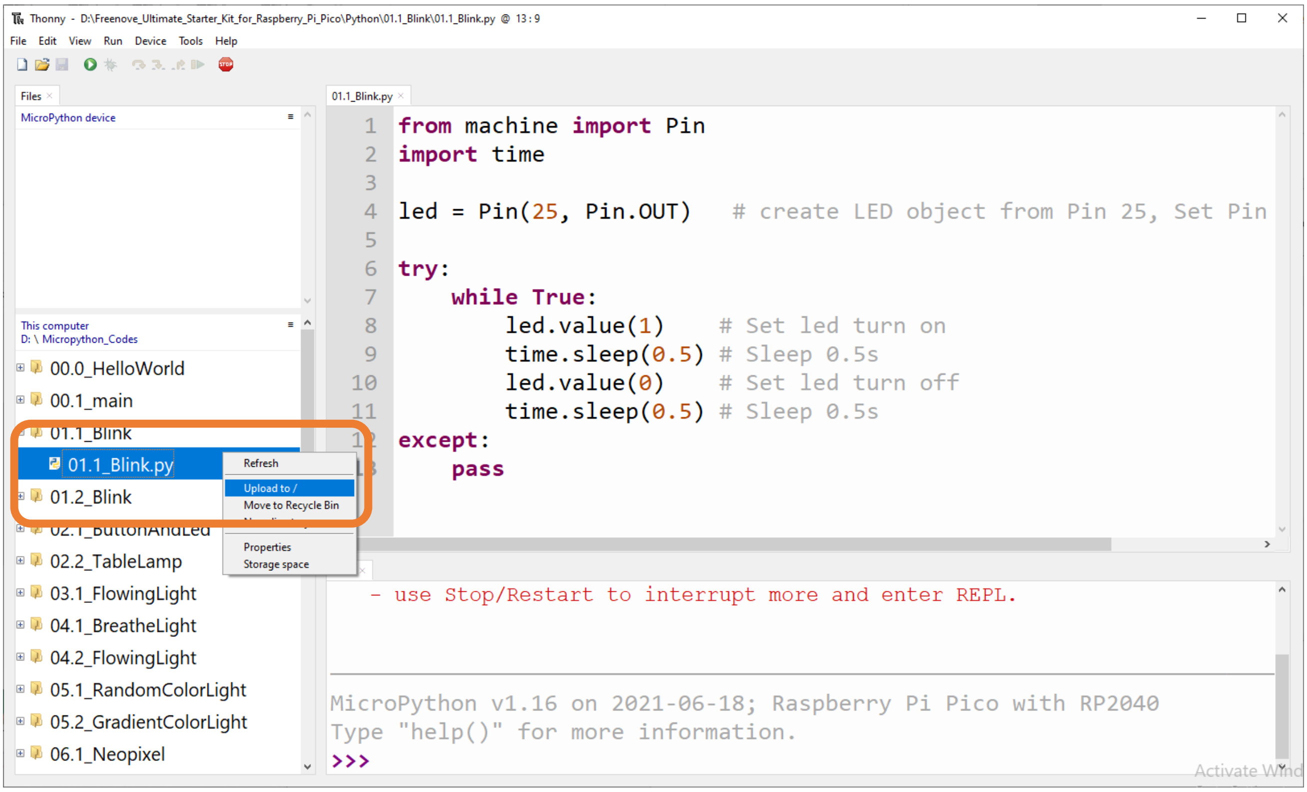



2. Expand "00.1_main" in the "Micropython_Codes" in the directory of disk(D), and double-click main.py, which is provided by us to enable programs in "MicroPython device" to run offline.

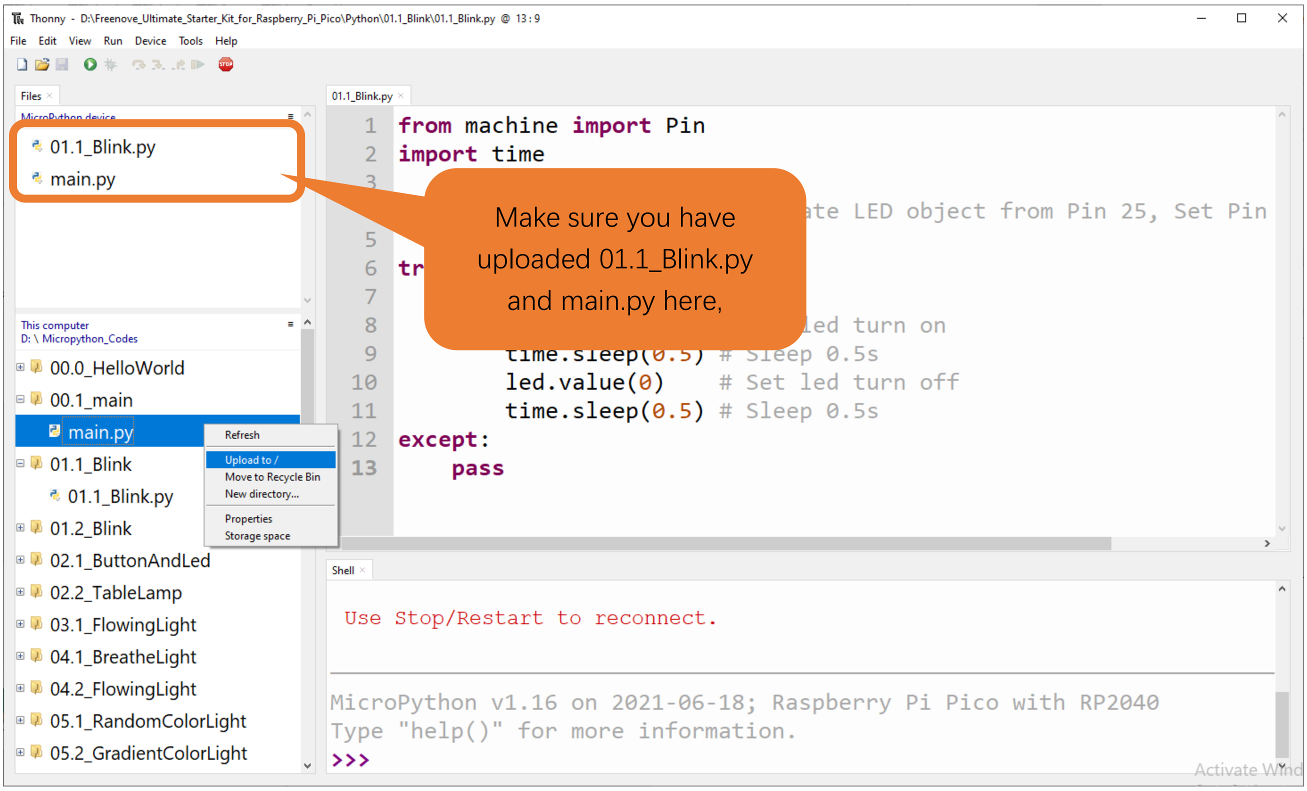


Here we use 00.1 and 01.1 cases as demonstration. The LED on Raspberry Pi Pico is used to show the result, which uses GP25 pin. If you have modified 01.1_Blick.py file, you need to change it accordingly.

As shown in the following illustration, right-click the file 01.1_Blink.py and select “Upload to /” to upload code to Raspberry Pi Pico.



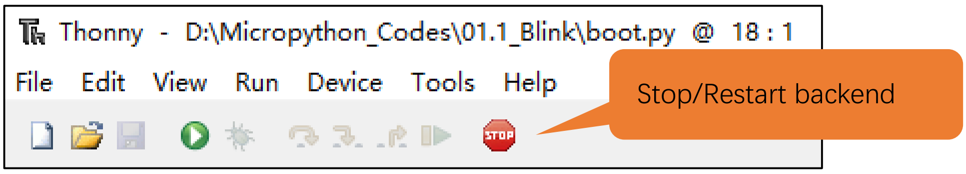

Upload main.py in the same way.




Disconnect Raspberry Pi Pico USB cable and reconnect it, the LED on pico will blink repeatedly.

Note:

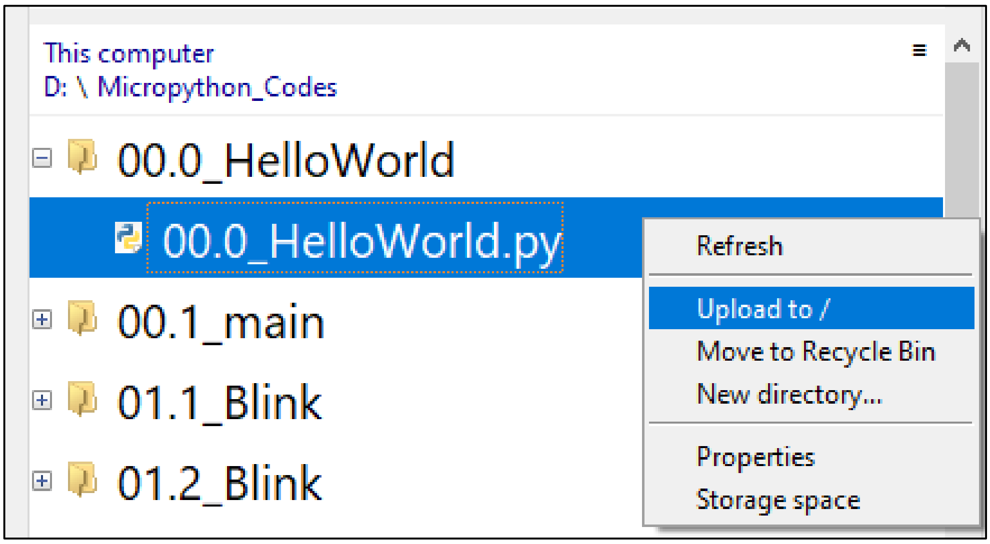
Codes here are run offline. If you want to stop running offline and enter Shell, just click “Stop” in Thonny.



# 0.6 Thonny Common Operation

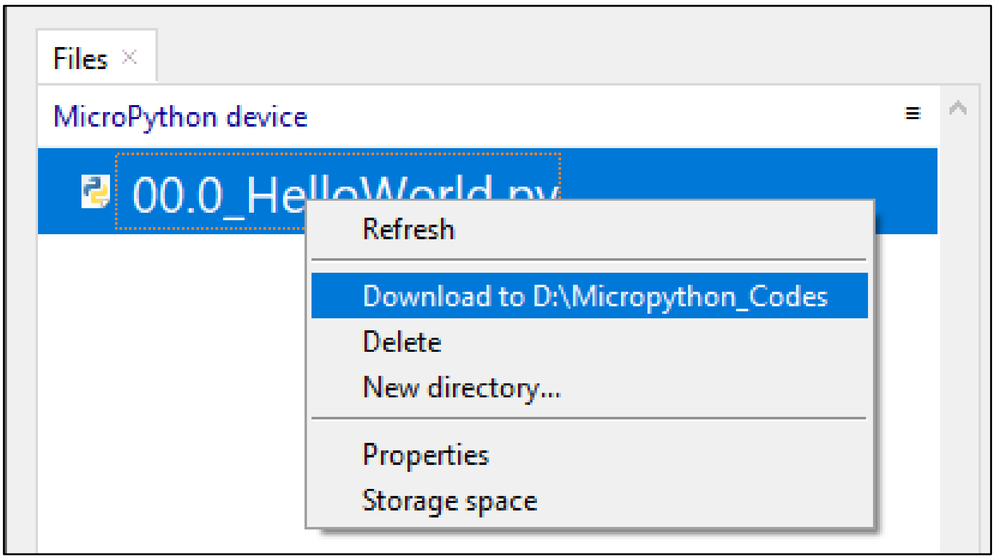
## Uploading Code to Raspberry Pi Pico

Select “00.0_HelloWorld.py” in “00.0_HelloWorld”, right-click your mouse and select “Upload to /” to upload code to Raspberry Pi Pico’s root directory.



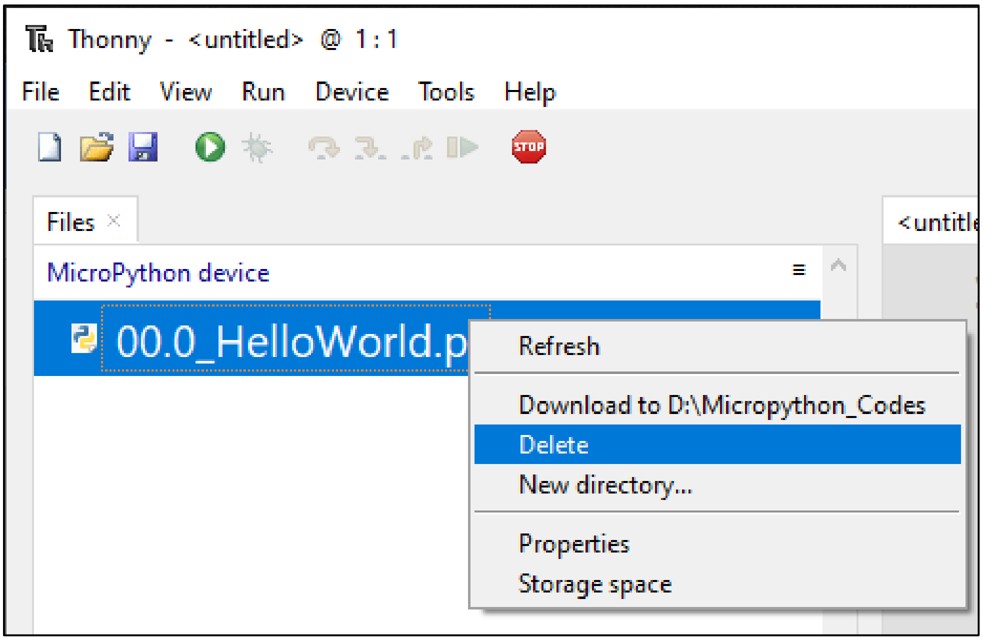

## Downloading Code to Computer

Select “00.0_HelloWorld.py” in “MicroPython device”, right-click to select “Download to ...” to download the code to your computer.



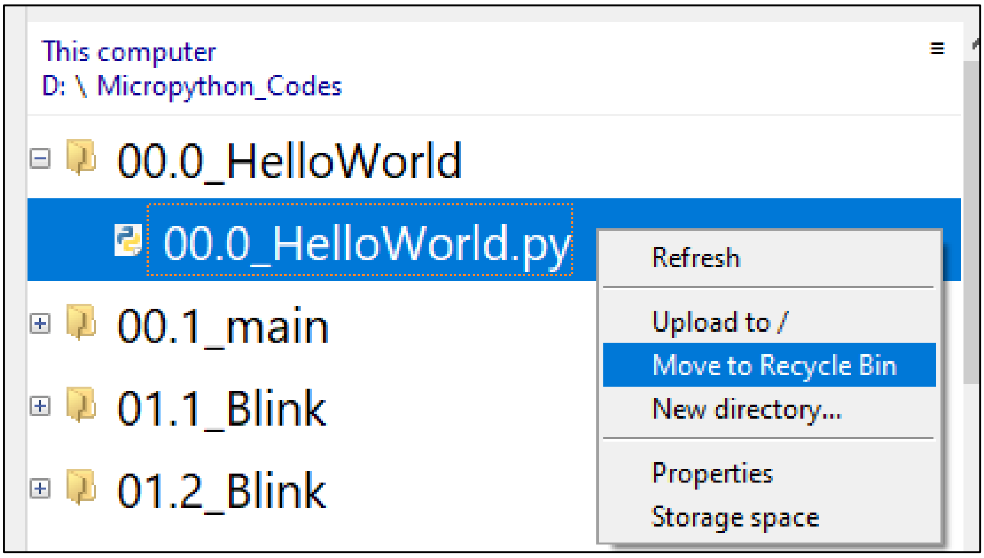
## Deleting Files from Raspberry Pi Pico's Root Directory

Select "00.0_HelloWorld.py" in "MicroPython device", right-click it and select "Delete" to delete "00.0_HelloWorld.py" from Raspberry Pi Pico's root directory.



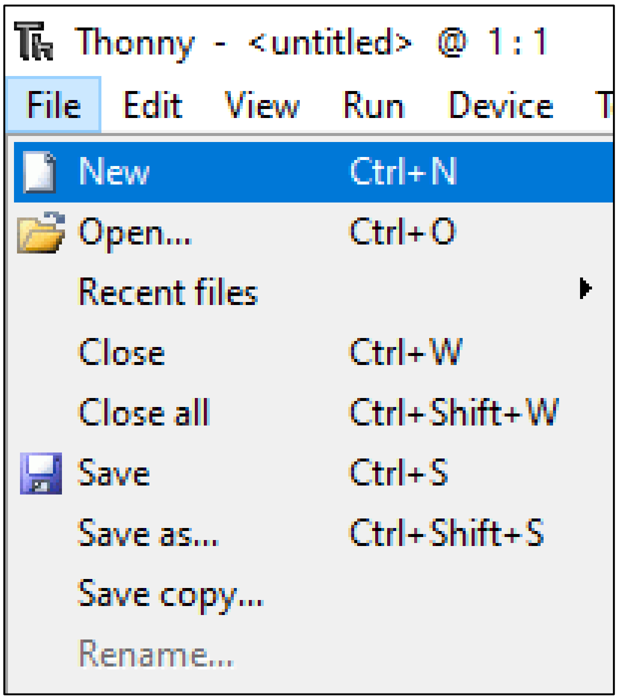

## Deleting Files from your Computer Directory

Select "00.0_HelloWorld.py" in "00.0_HelloWorld", right-click it and select "Move to Recycle Bin" to delete it from "00.0_HelloWorld".

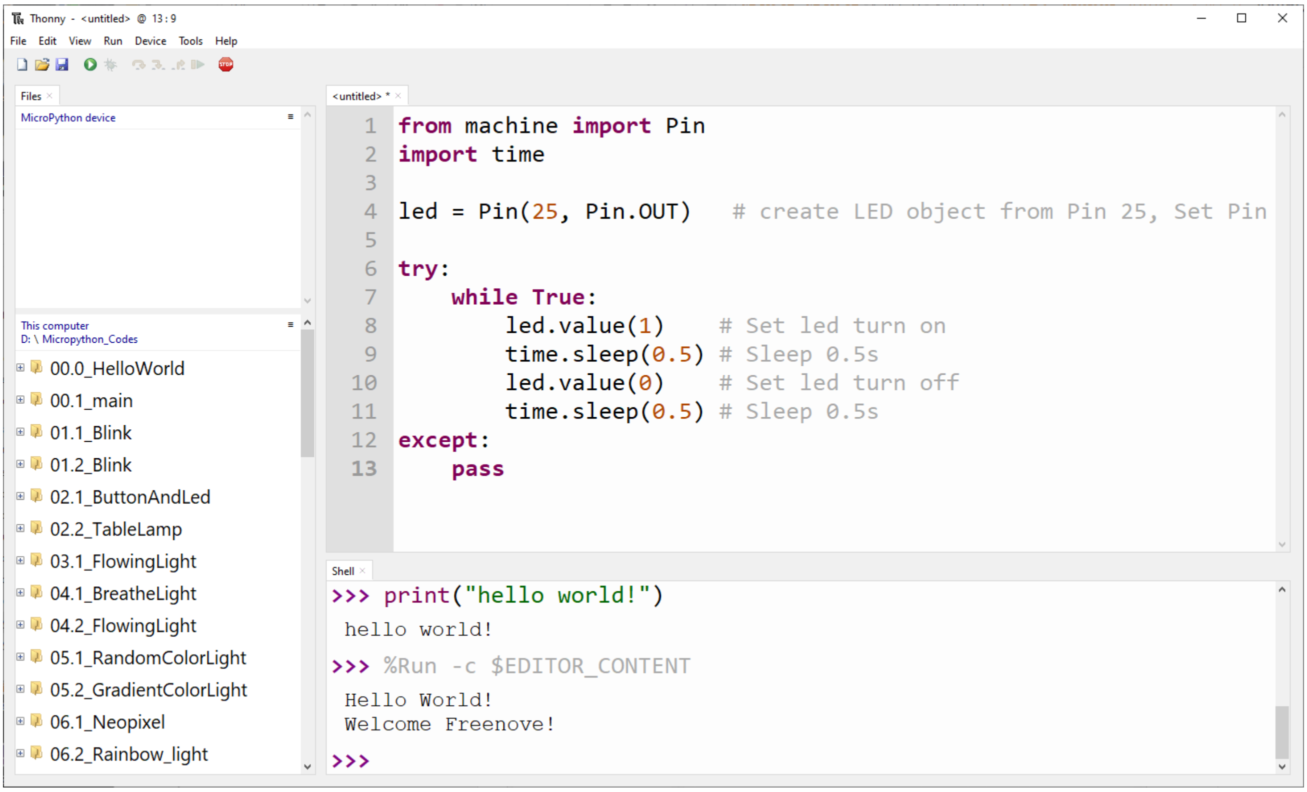


## Creating and Saving the code

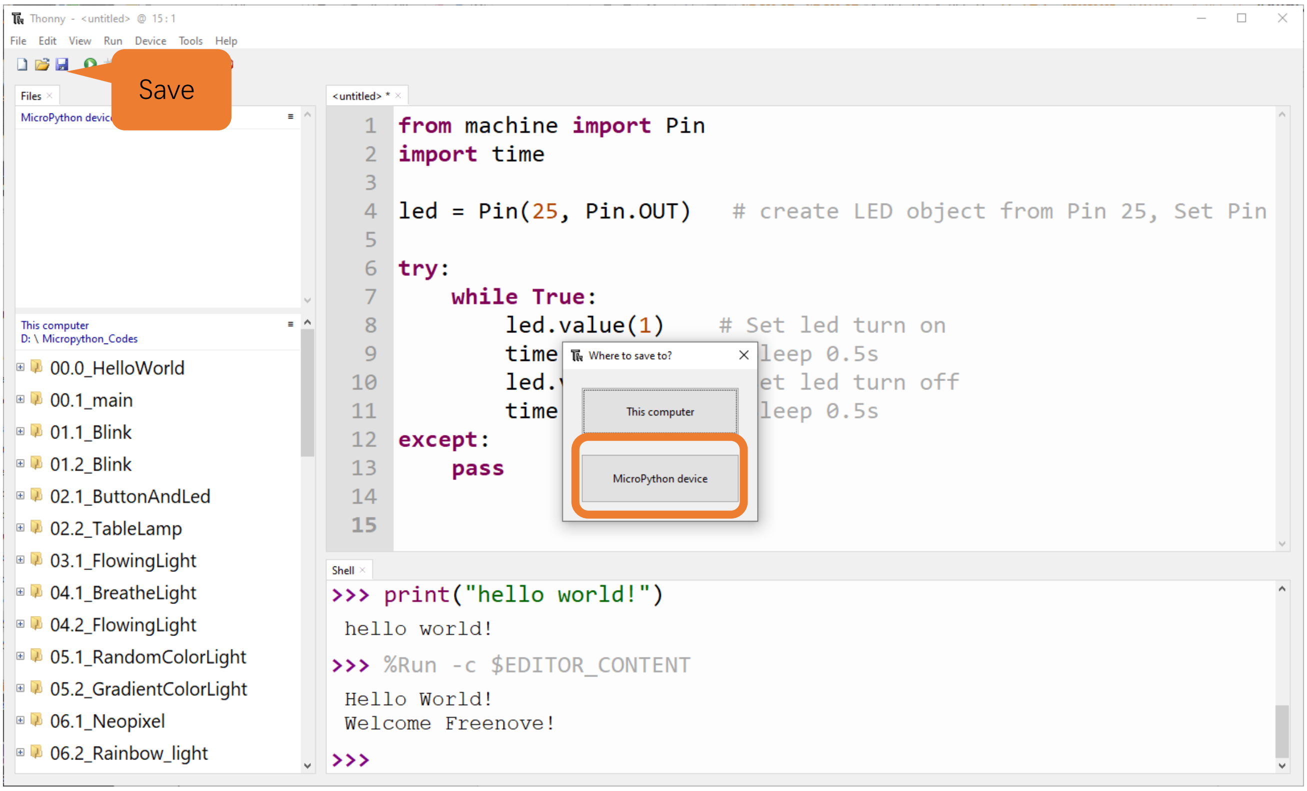
Click "File"→"New" to create and write codes.



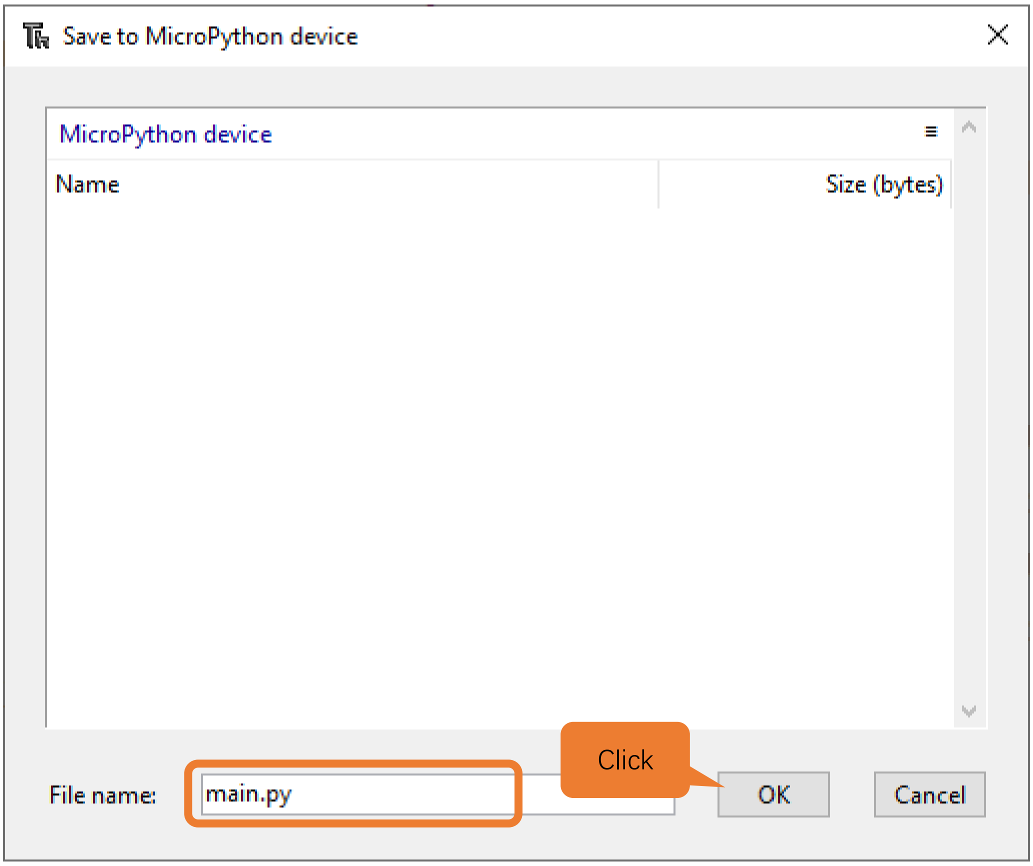

Enter codes in the newly opened file. Here we use codes of "01.1_Blink.py" as an example.



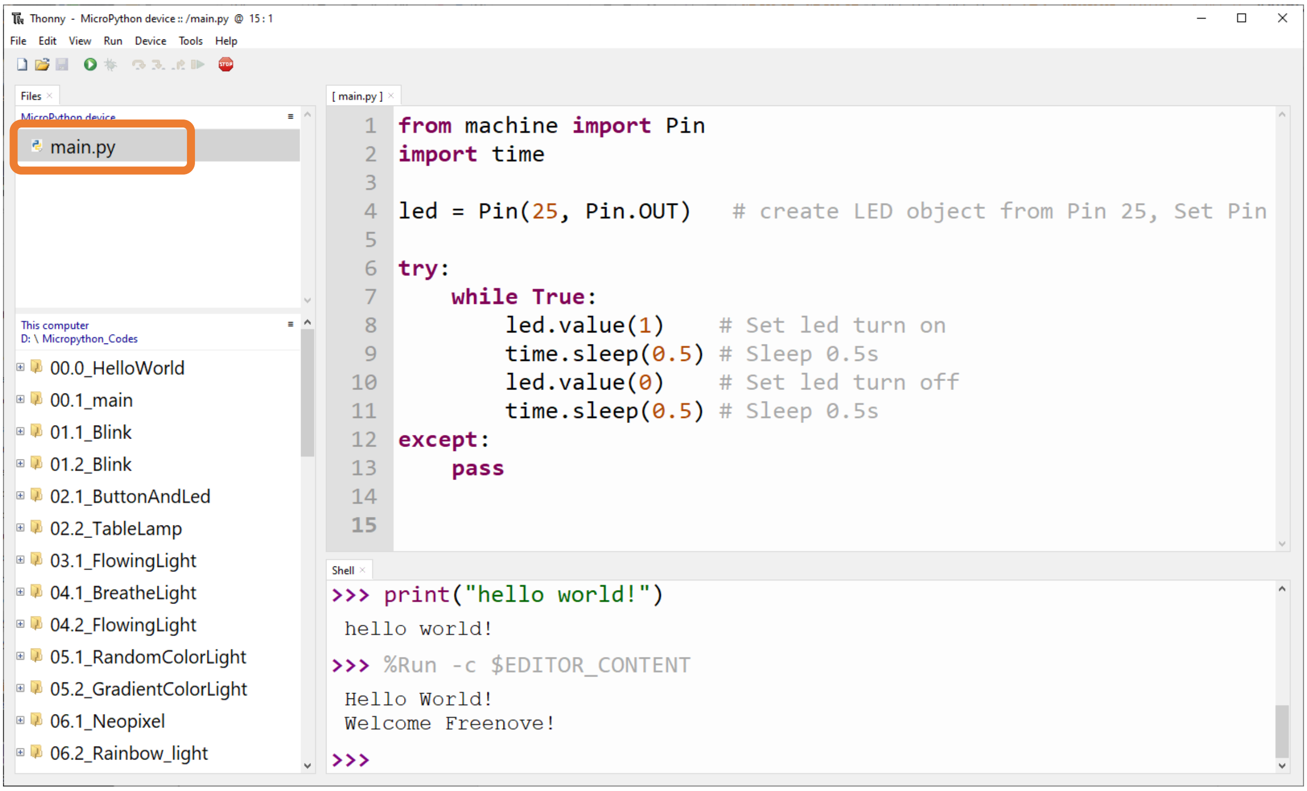
Click “Save” on the menu bar. You can save the codes either to your computer or to Raspberry Pi Pico.



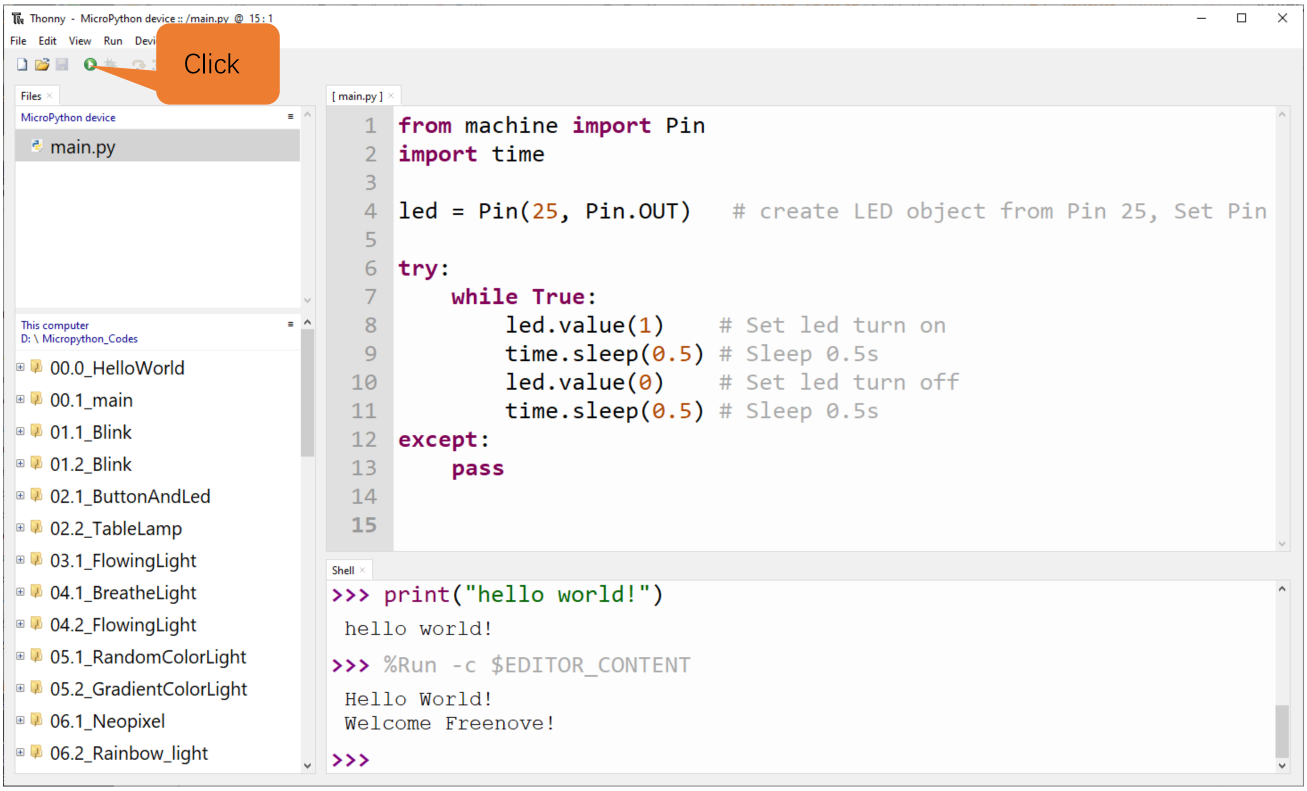

Select “MicroPython device”, enter “main.py” in the newly pop-up window and click “OK”.



You can see that codes have been uploaded to Raspberry Pi Pico.

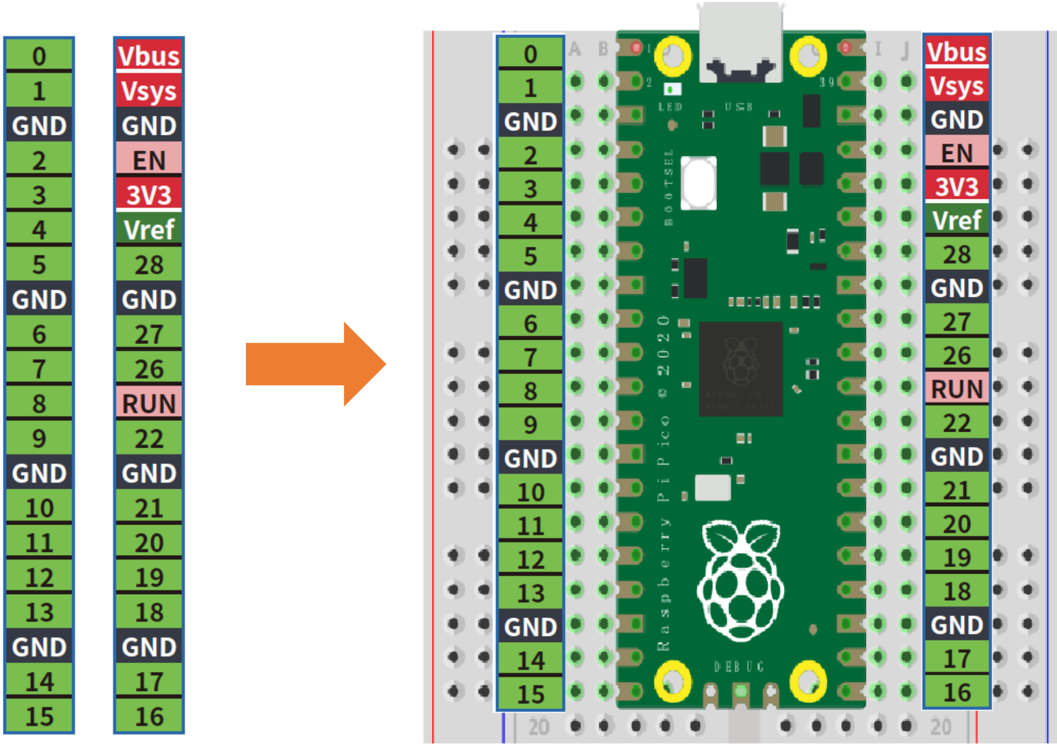



Click "Run" and the LED on Raspberry Pi Pico will blink periodically.



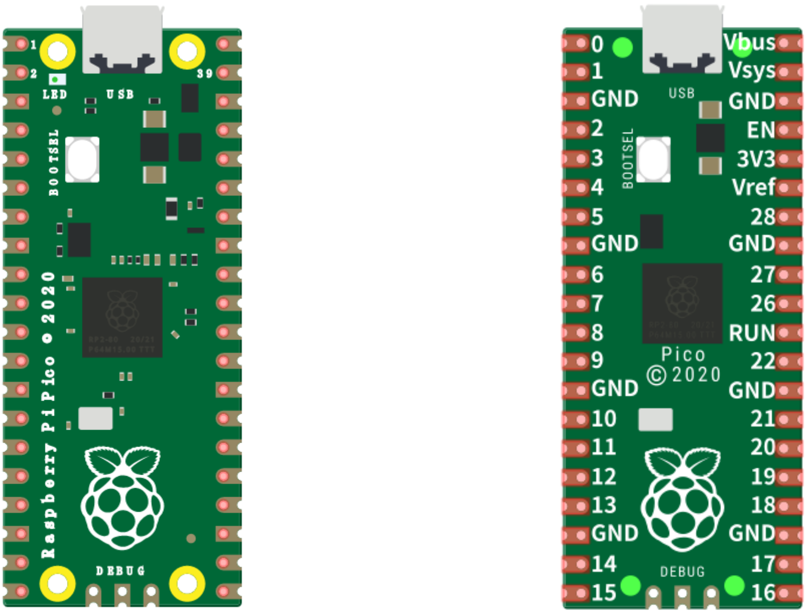
# 06. Paste the Sticker on the Breadboard

It is not difficult to use the Pico. However, officially, the pin functions are printed on the back of the board, which makes it inconvenient to use. To help users finish each project in the tutorial faster and easier, we provide stickers of the pin functions as follows:




You can paste the sticker on the blank area of the breadboard as above.

To make the tutorial more intuitive, we've made some changes to the simulation diagram as below. The left one is the actual Pico and the right one is its simulation diagram. Please note that to avoid misunderstanding.



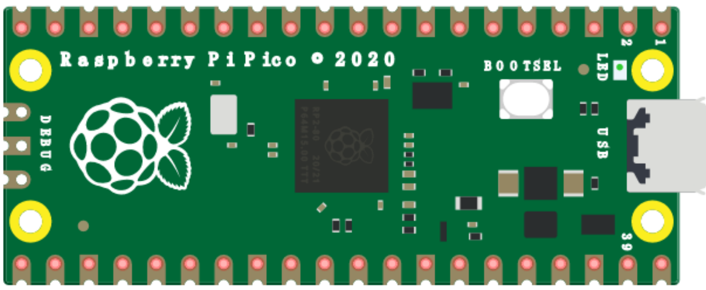
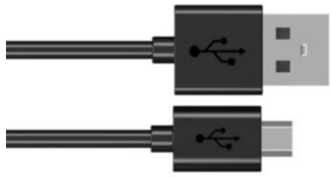
# Chapter 1 LED (Important)

This chapter is the Start Point in the journey to build and explore Raspberry Pi Pico electronic projects. We will start with simple "Blink" project.

## Project 1.1 Blink

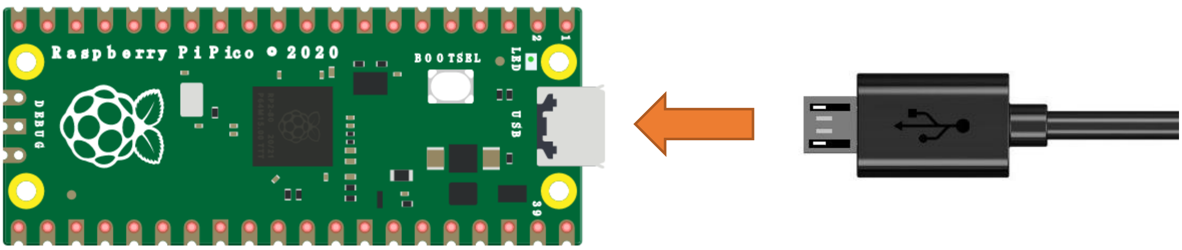
In this project, we will use Raspberry Pi Pico to control blinking a common LED.
If you have not yet installed Thonny, click here.
If you have not yet downloaded Micropython Firmware, click here.
If you have not yet loaded Micropython Firmware, click here.

### Component List

| Raspberry Pi Pico x1 | USB cable x1 |
|---|---|
|  |  |

Power

In this tutorial, we connect Raspberry Pi Pico and compter with a USB cable.

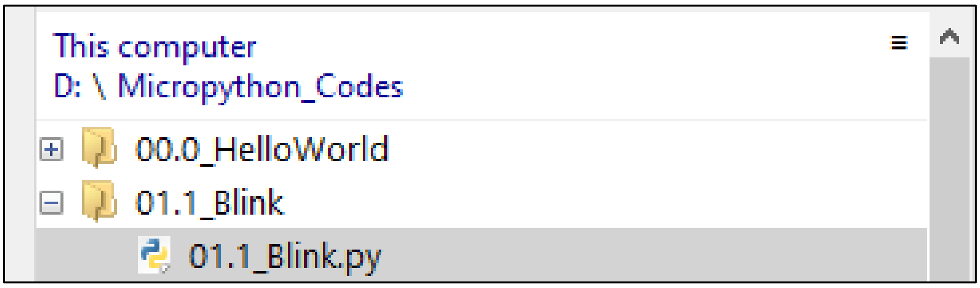


# Code

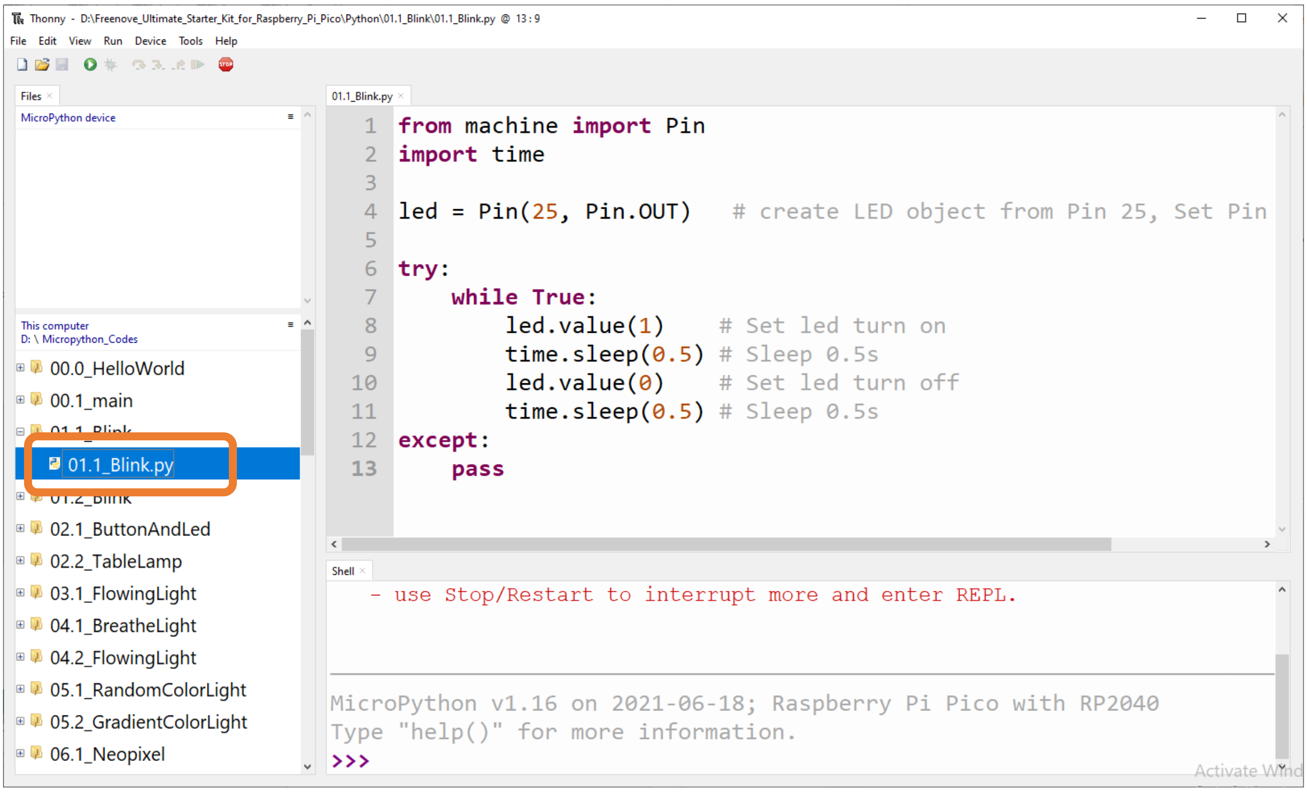
Codes used in this tutorial are saved in "**Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Python_Codes**". You can move the codes to any location. For example, we save the codes in Disk(D) with the path of "**D:/Micropython_Codes**".

## 01.1_Blink

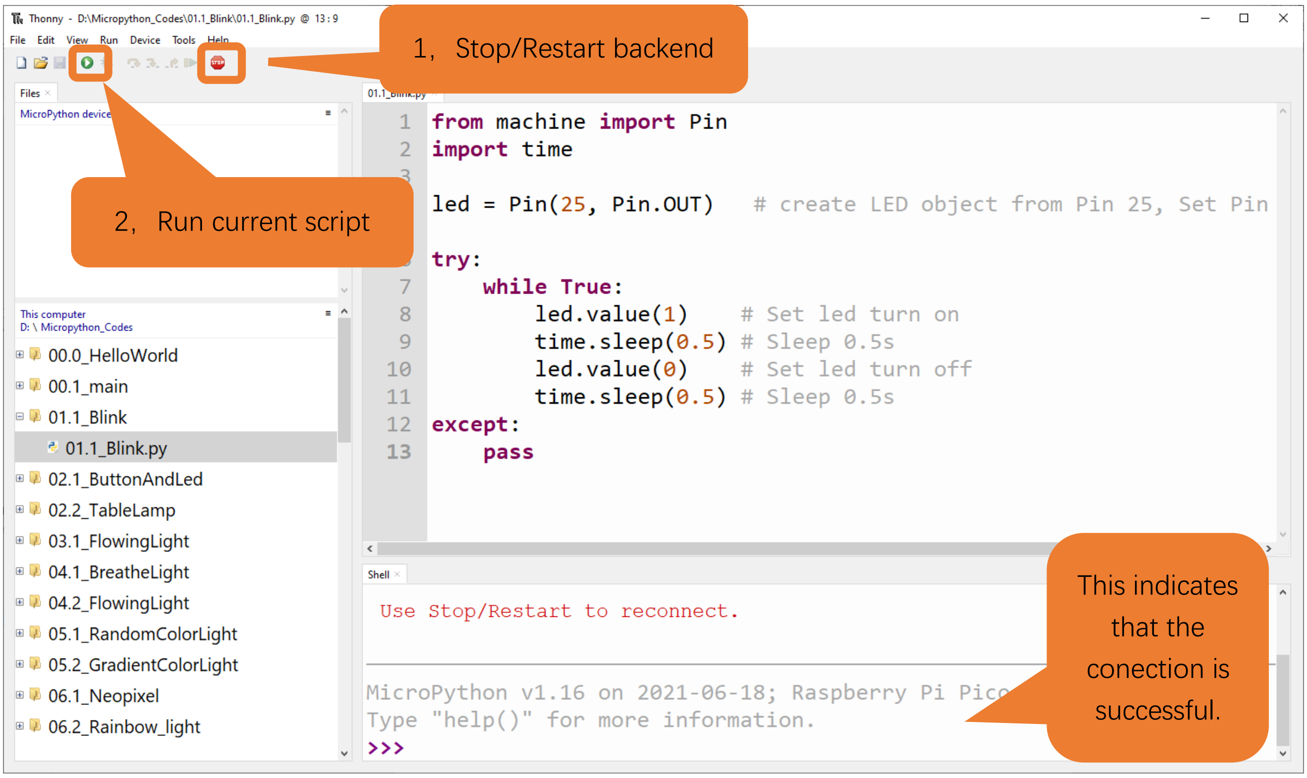
Open "Thonny", click "This computer"→"D:"→"Micropython_Codes".



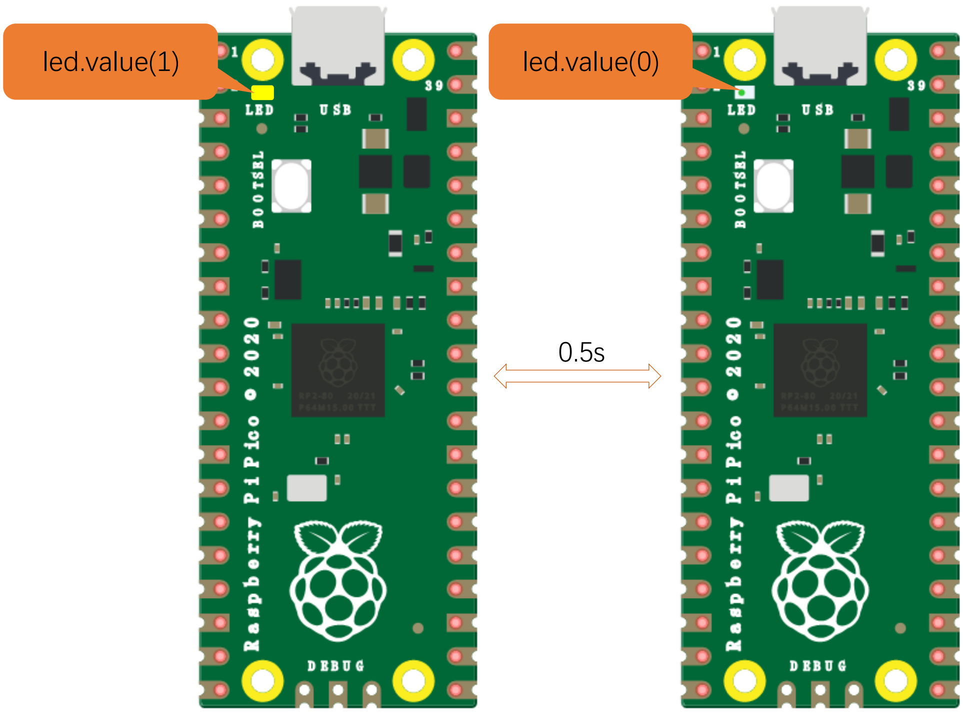

Expand folder "01.1_Blink" and double click "01.1_Blink.py" to open it. As shown in the illustration below.



Make sure Raspberry Pi Pico has been connected with the computer. Click “Stop/Restart backend”, and then wait to see what interface will show up.

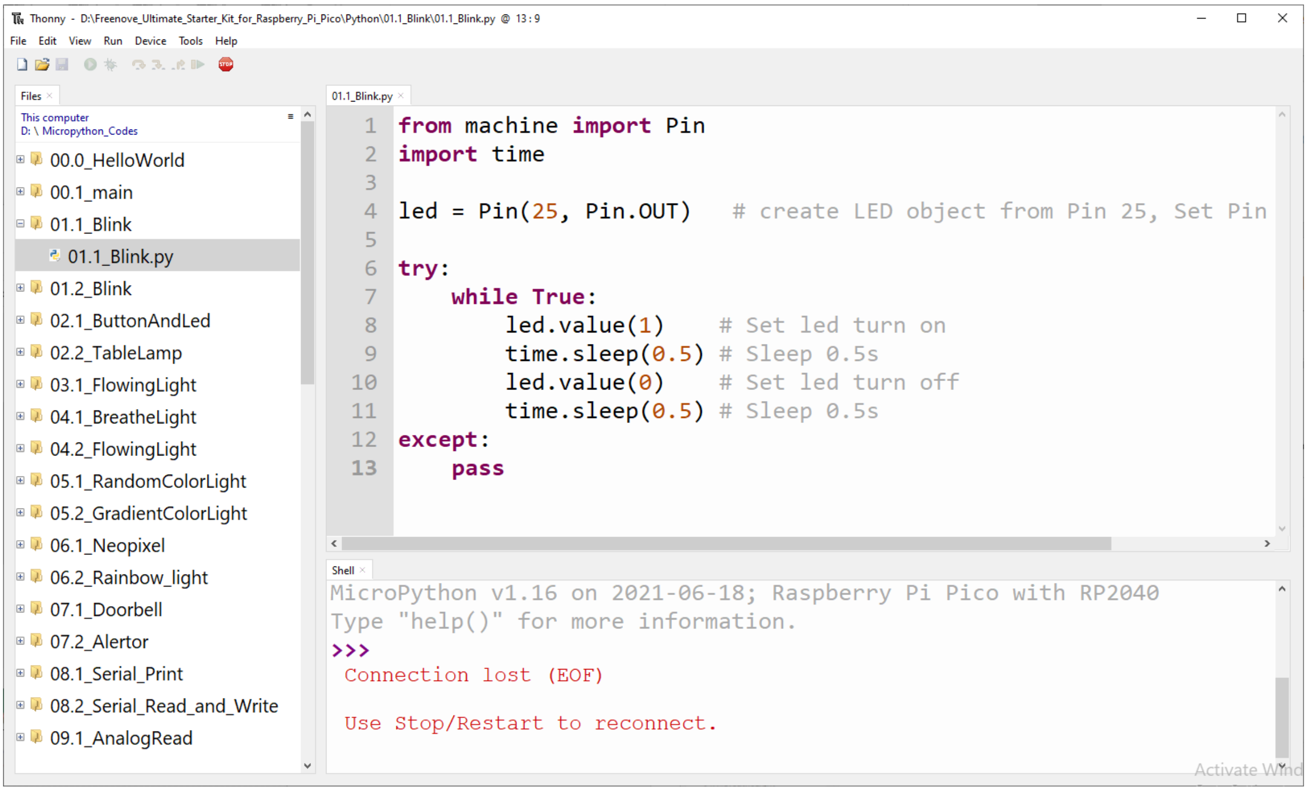



Click “Run current script” shown in the box above, the code starts to be executed and the LED in the circuit starts to blink. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



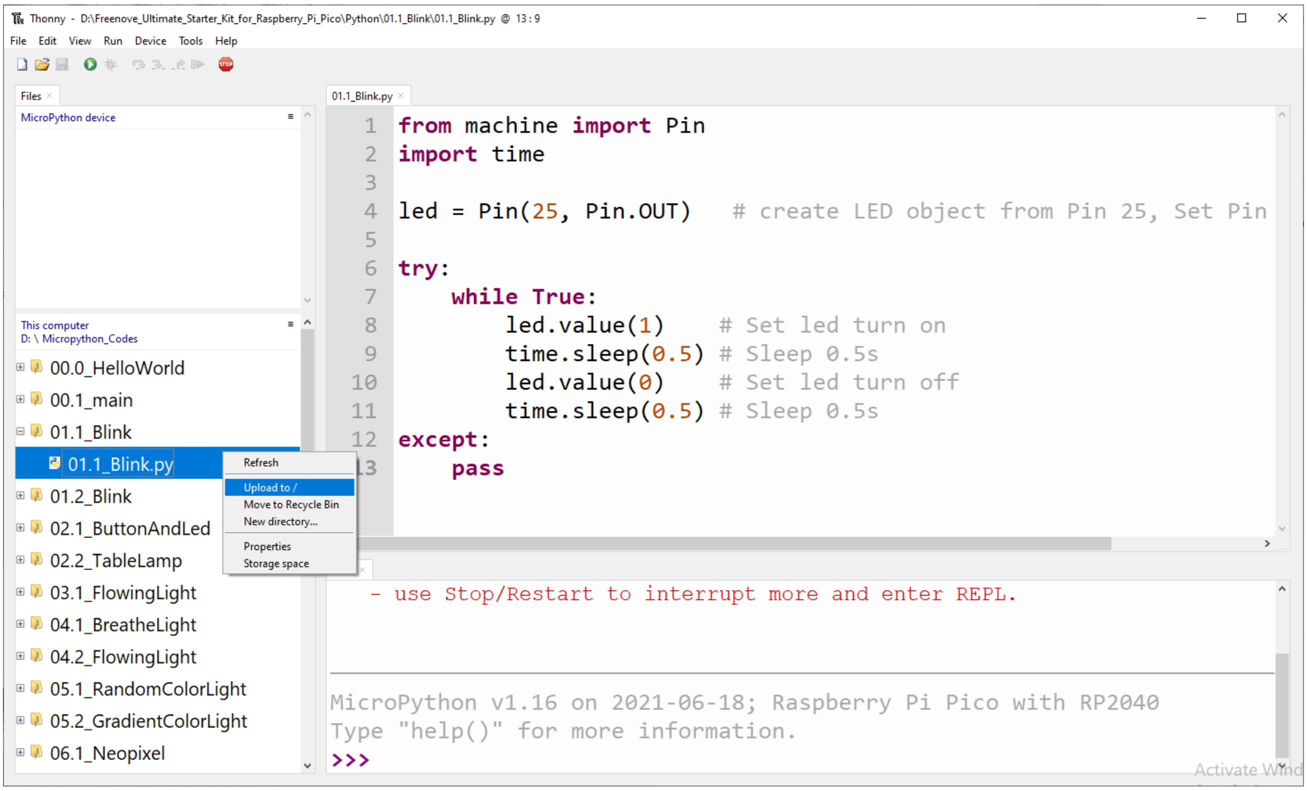
Note:

This is the code running online. If you disconnect USB cable and repower Raspberry Pi Pico, LED stops blinking and the following messages will display in Thonny.

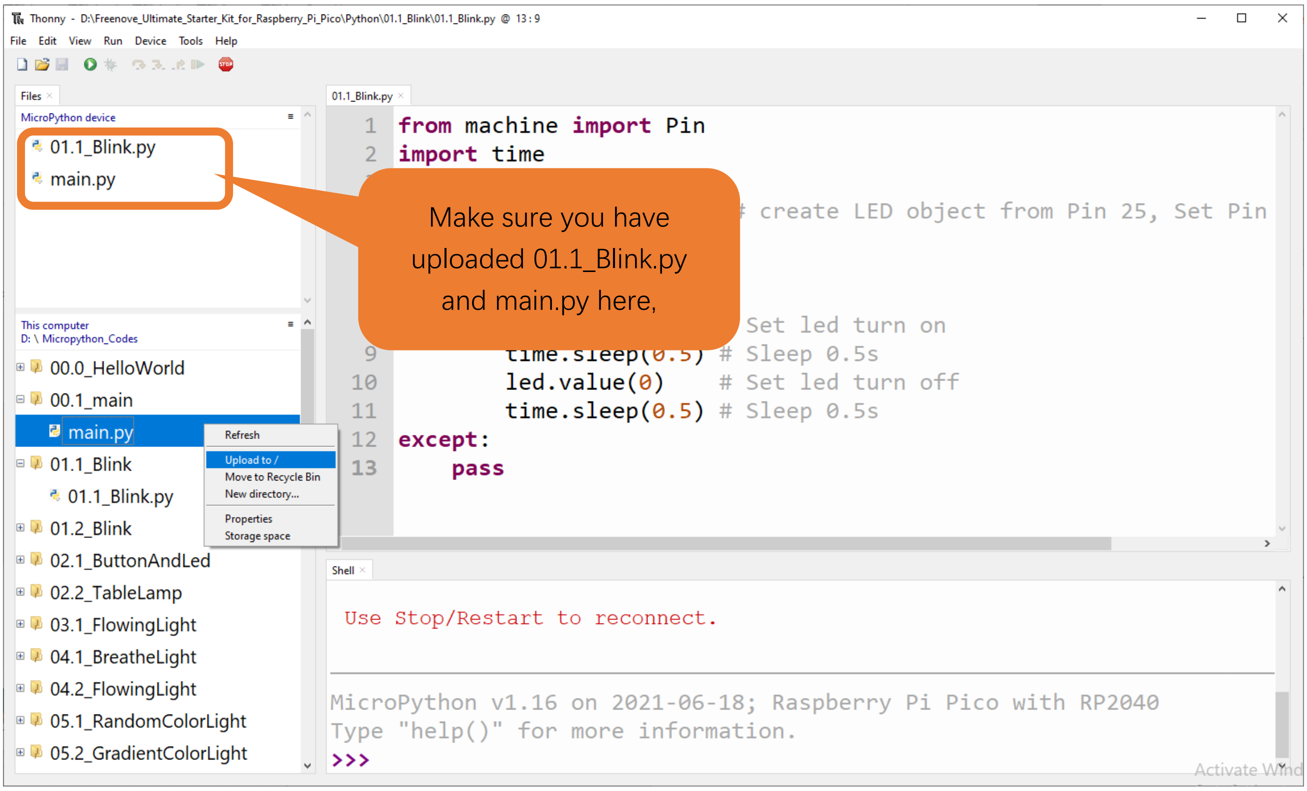


## Uploading code to Raspberry Pi Pico

As shown in the following illustration, right-click the file 01.1_Blink.py and select "Upload to /" to upload code to Raspberry Pi Pico.



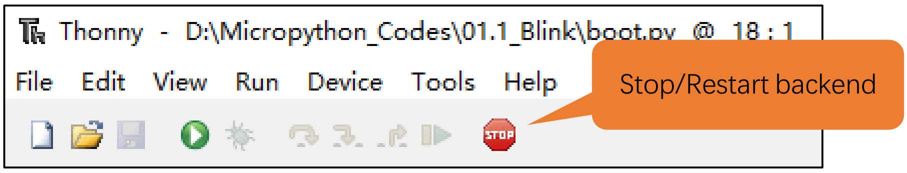
Upload main.py in the same way.




Disconnect Raspberry Pi Pico USB cable and reconnect it, LED on Pico will blink repeatedly.

Note:

Codes here is run offline. If you want to stop running offline and enter Shell, just click “Stop” in Thonny.




**If you have any concerns, please contact us via: support@freenove.com**

The following is the program code:

```
import time
from machine import Pin

led=Pin(25,Pin.OUT)     #create LED object from Pin 25,Set Pin 25 to output

try:
    while True:
        led.value(1)     #Set led turn on
        time.sleep(0.5) #Sleep 0.5s
        led.value(0)     #Set led turn off
        time.sleep(0.5) #Sleep 0.5s
except:
    pass
```

Each time a new file is opened, the program will be executed from top to bottom. When encountering a loop construction, it will execute the loop statement according to the loop condition.

Setup

Loop

```
import time
from machine import Pin

led=Pin(25,Pin.OUT)     #create LED object from Pin 25,Set Pin 25 to output
try:
    while True:
        ...
except:
    pass
```

Print() function is used to print data to Terminal. It can be executed in Terminal directly or be written in a Python file and executed by running the file.

```
print("Hello world!")
```

Each time when using the functions of Raspberry Pi Pico, you need to import modules corresponding to those functions: Import time module and Pin module of machine module.

```
import time
from machine import Pin
```

Configure GP25 of Raspberry Pi Pico to output mode and assign it to an object named "led".

```
led=Pin(25,Pin.OUT) #create LED object from Pin 25,Set Pin 25 to output
```

It means that from now on, LED representing GP25 is in output mode.

Set the value of LED to 1 and GP25 will output high level.

```
led.value(1) #Set led turn on
```

Set the value of LED to 0 and GP25 will output low level.

```
led.value(0) #Set led turn on
```

Execute codes in a while loop.

```
    while True:
        ...
```

Put statements that may cause an error in "try" block and the executing statements when an error occurs in "except" block. In general, when the program executes statements, it will execute those in "try" block. However, when an error occurs to Raspberry Pi Pico due to some interference or other reasons, it will execute statements in "except" block.

"Pass" is an empty statement. When it is executed, nothing happens. It is useful as a placeholder to make the structure of a program look better.

```
try:
    ...
except:
    pass
```

The single-line comment of Micropython starts with a "#" and continues to the end of the line. Comments help us to understand code. When programs are running, Thonny will skip comments.

```
#Set led turn on
```

MicroPython uses indentations to distinguish different blocks of code instead of braces. The number of indentations is changeable, but it must be consistent throughout one block. If the indentation of the same code block is inconsistent, it will cause errors when the program runs.

```
    while True:
        led.value(1)     #Set led turn on
        time.sleep(0.5) #Sleep 0.5s
        led.value(0)    #Set led turn off
        time.sleep(0.5) #Sleep 0.5s
```

## How to import python files

Whether to import the built-in python module or to import that written by users, the command "import" is needed.

If you import the module directly you should indicate the module to which the function or attribute belongs when using the function or attribute (constant, variable) in the module. The format should be: <module name>.<function or attribute>, otherwise an error will occur.

```
import random

num = random.randint(1, 100)
print(num)
```

If you only want to import a certain function or attribute in the module, use the from...import statement. The format is as follows

```
from random import randint
num = randint(1, 100)
print(num)
```

When using "from...import" statement to import function, to avoid conflicts and for easy understanding, you can use "as" statement to rename the imported function, as follows

```
from random import randint as rand
num = rand(1, 100)
print(num)
```

Reference

**Class machine**

Before each use of the **machine** module, please add the statement "**import machine**" to the top of python file.

**machine.freq(freq_val):** When "freq_val" is not specified, it is to return to the current CPU frequency; Otherwise, it is to set the current CPU frequency.

**freq_val**: 125000000Hz(125MHz).

**machine.reset()**: A reset function. When it is called, the program will be reset.

**machine.unique_id()**: Obtains MAC address of the device.

**machine.idle()**: Turns off any temporarily unused functions on the chip and its clock, which is useful to reduce power consumption at any time during short or long periods.

**machine.disable_irq()**: Disables interrupt requests and return the previous IRQ state. The disable_irq () function and enable_irq () function need to be used together; Otherwise the machine will crash and restart.

**machine.enable_irq(state)**: To re-enable interrupt requests. The parameter **state** should be the value that was returned from the most recent call to the disable_irq() function.

**machine.time_pulse_us(pin, pulse_level, timeout_us=1000000)**:

Tests the duration of the external pulse level on the given pin and returns the duration of the external pulse level in microseconds. When pulse level = 1, it tests the high level duration; When pulse level = 0, it tests the low level duration.

If the setting level is not consistent with the current pulse level, it will wait until they are consistent, and then start timing. If the set level is consistent with the current pulse level, it will start timing immediately.

When the pin level is opposite to the set level, it will wait for timeout and return "-2". When the pin level and the set level is the same, it will also wait timeout but return "-1". **timeout_us** is the duration of timeout.

For more information about class and function, please refer to:

https://docs.micropython.org/en/latest/rp2/quickref.html

**Class time**

Before each use of the **time** module, please add the statement "**import time**" to the top of python file

**time.sleep(sec)**: Sleeps for the given number of seconds.

**sec**: This argument should be either an int or a float.

**time.sleep_ms(ms)**: Sleeps for the given number of milliseconds, ms should be an int.

**time.sleep_us(us)**: Sleeps for the given number of microseconds, us should be an int.

**time.time()**: Obtains the timestamp of CPU, with second as its unit.

**time.ticks_ms()**: Returns the incrementing millisecond counter value, which recounts after some values.

**time.ticks_us()**: Returns microsecond.

**time.ticks_cpu()**: Similar to ticks_ms() and ticks_us(), but it is more accurate(return clock of CPU).

**time.ticks_add(ticks, delta)**: Gets the timestamp after the offset.

**ticks**: ticks_ms()、ticks_us()、ticks_cpu().

**delta**: Delta can be an arbitrary integer number or numeric expression.

**time.ticks_diff(old_t, new_t)**: Calculates the interval between two timestamps, such as ticks_ms(), ticks_us() or ticks_cpu().

**old_t**: Starting time.

**new_t**: Ending time.

**Class Pin(id, mode, pull, value)**

Before each use of the **Pin** module, please add the statement "**from machine import Pin**" to the top of python file.

**id:** Arbitrary pin number.

**mode**: Mode of pins.

**Pin.IN**: Input Mode.

**Pin.OUT**: Output Mode.

**Pin.OPEN_DRAIN**: Open-drain Mode.

**Pull**: Whether to enable the internal pull up and down mode.

**None**: No pull up or pull down resistors.

**Pin.PULL_UP**: Pull-up Mode, outputting high level by default.

**Pin.PULL_DOWN**: Pull-down Mode, outputting low level by default.

**Value**: State of the pin level, 0/1.

**Pin.init(mode, pull)**: Initialize pins.

**Pin.value([value])**: Obtain or set state of the pin level, return 0 or 1 according to the logic level of pins. Without parameter, it reads input level. With parameter given, it is to set output level.

**value**: It can be either True/False or 1/0.

**Pin.irq(trigger, handler)**: Configures an interrupt handler to be called when the pin level meets a condition.

**trigger**:

**Pin.IRQ_FALLING**: interrupt on falling edge.

**Pin.IRQ_RISING**: interrupt on rising edge.

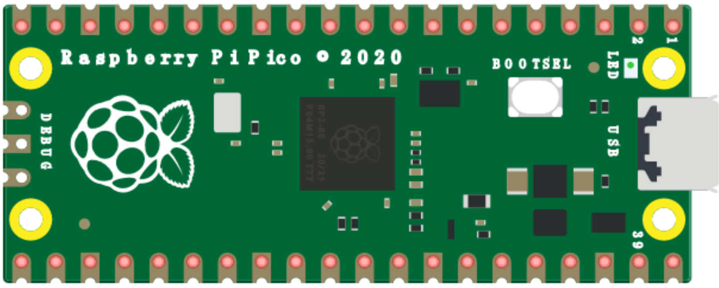
**Handler**: callback function.

# Project 1.2 Blink

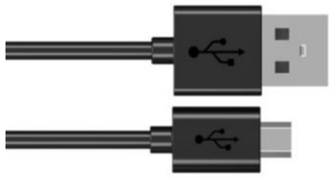
In this project, we will use Raspberry Pi Pico to control blinking a common LED.

## Component List

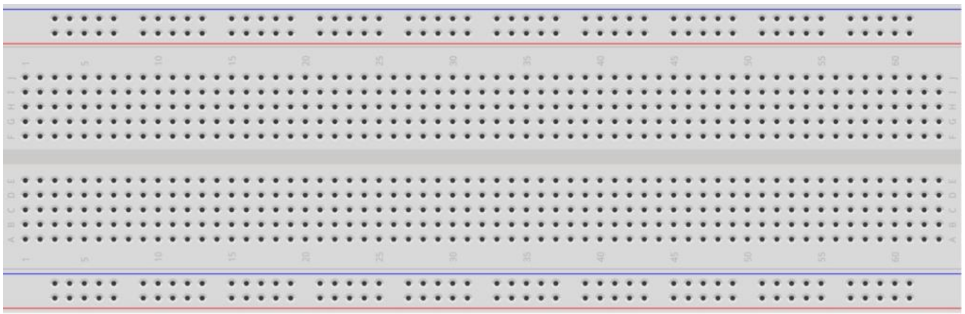
Raspberry Pi Pico x1




USB Cable x1



Breadboard x1



LED x1

Resistor 220Ω x1

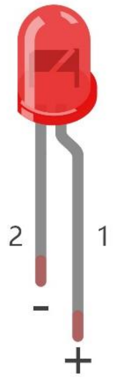
Jumper

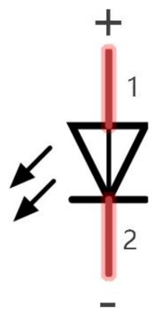
## Component knowledge

### LED

An LED is a type of diode. All diodes only work if current is flowing in the correct direction and have two Poles. An LED will only work (light up) if the longer pin (+) of LED is connected to the positive output from a power source and the shorter pin is connected to the negative (-). Negative output is also referred to as Ground (GND). This type of component is known as "Polar" (think One-Way Street).

All common 2 lead diodes are the same in this respect. Diodes work only if the voltage of its positive electrode is higher than its negative electrode and there is a narrow range of operating voltage for most all common diodes of 1.9 and 3.4V. If you use much more than 3.3V the LED will be damaged and burn out.




| LED | Voltage | Maximum current | Recommended current |
|---|---|---|---|
| Red | 1.9-2.2V | 20mA | 10mA |
| Green | 2.9-3.4V | 10mA | 5mA |
| Blue | 2.9-3.4V | 10mA | 5mA |
| Volt ampere characteristics conform to diode | | | |

Note: LEDs cannot be directly connected to a power supply, which usually ends in a damaged component. A resistor with a specified resistance value must be connected in series to the LED you plan to use.

### Resistor

Resistors use Ohms (Ω) as the unit of measurement of their resistance (R). 1MΩ=1000kΩ, 1kΩ=1000Ω.

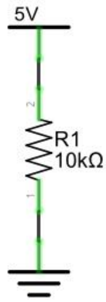
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



The bands of color on a resistor is a shorthand code used to identify its resistance value. For more details of resistor color codes, please refer to the appendix of this tutorial.

With a fixed voltage, there will be less current output with greater resistance added to the circuit. The relationship between Current, Voltage and Resistance can be expressed by this formula: I=V/R known as Ohm's Law where I = Current, V = Voltage and R = Resistance. Knowing the values of any two of these allows you to solve the value of the third.

In the following diagram, the current through R1 is: I=U/R=5V/10kΩ=0.0005A=0.5mA.

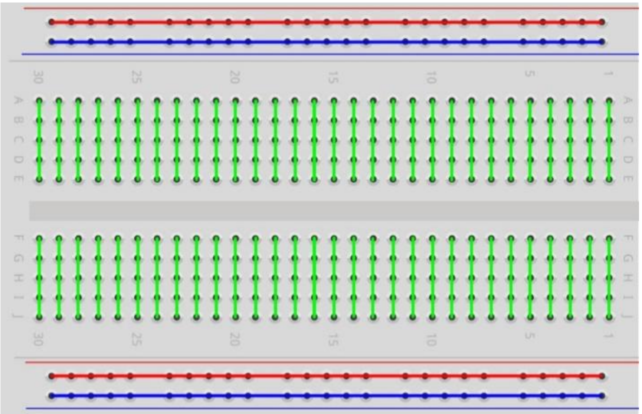



WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

Note: Unlike LEDs and Diodes, Resistors have no poles and re non-polar (it does not matter which direction you insert them into a circuit, it will work the same)

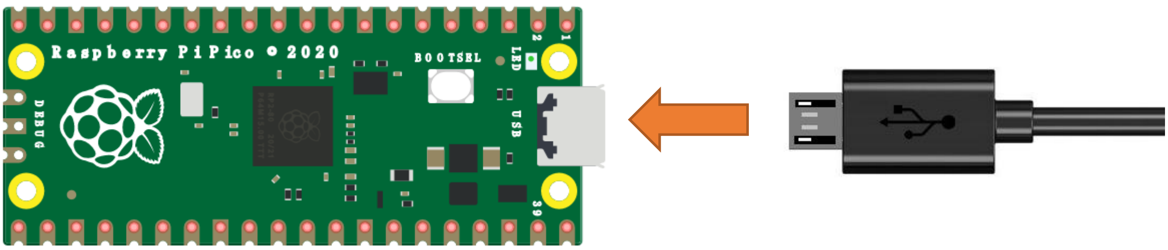
## Breadboard

Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached. The left picture shows the way to connect pins. The right picture shows the practical internal structure.



## Power

In this tutorial, we connect Raspberry Pi Pico and compter with a USB cable.

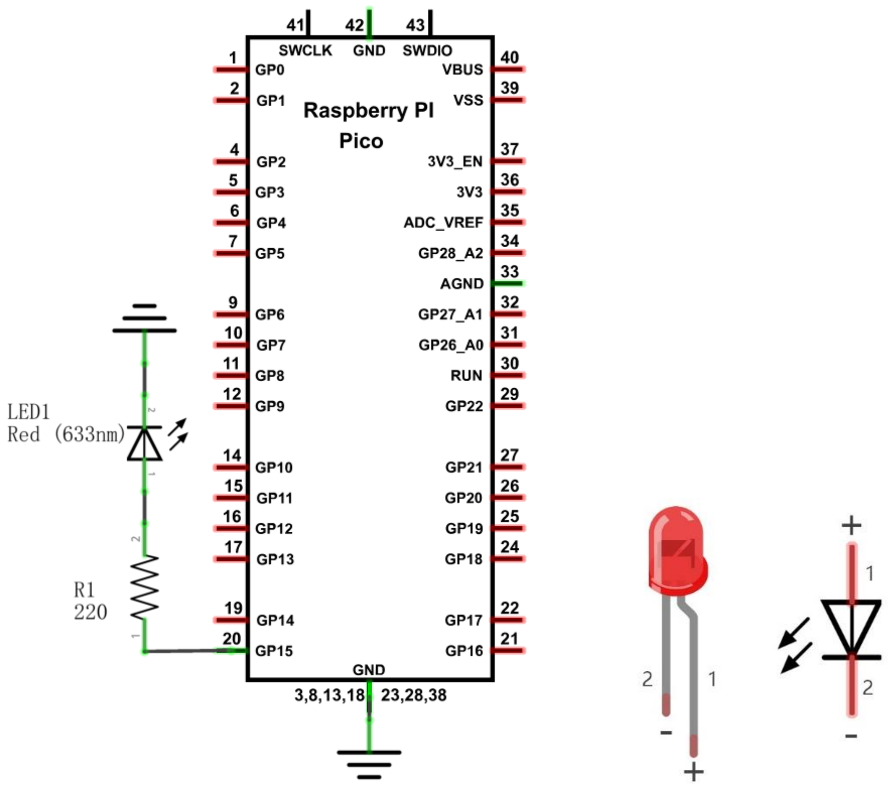


## Circuit

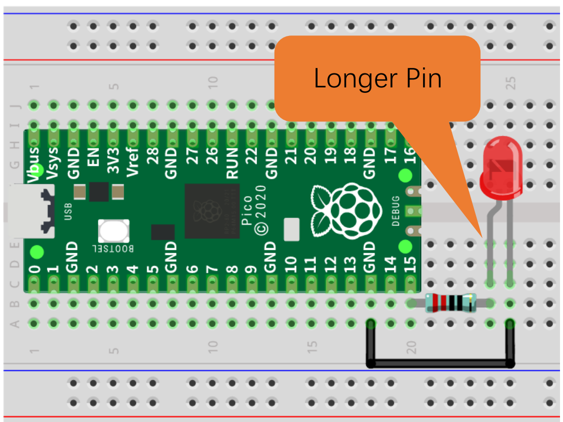
First, disconnect all power from the Raspberry Pi Pico. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to Raspberry Pi Pico.

CAUTION: Avoid any possible short circuits (especially connecting 3.3V and GND)! WARNING: A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your hardware!

Schematic diagram




Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



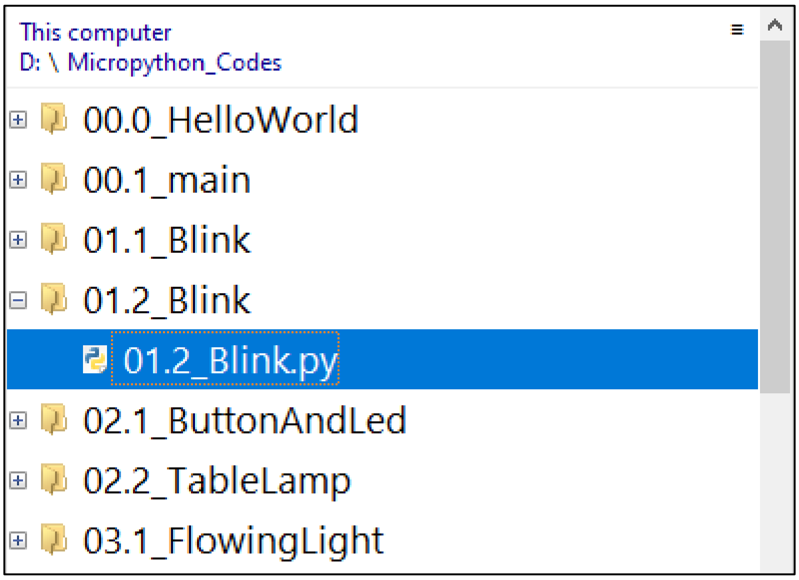

Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

## Code

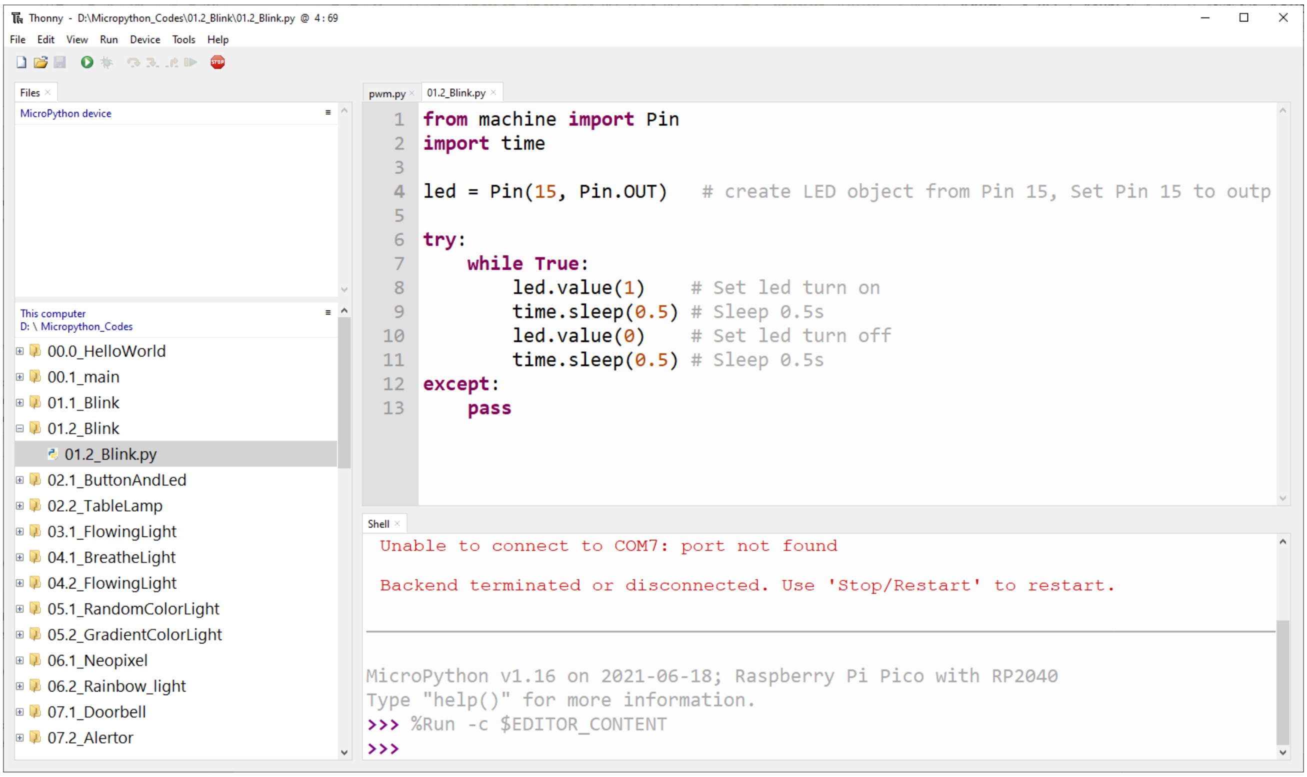
Codes used in this tutorial are saved in "**Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Python_Codes**". You can move the codes to any location. For example, we save the codes in Disk(D) with the path of "**D:/Micropython_Codes**".

01.2_Blink

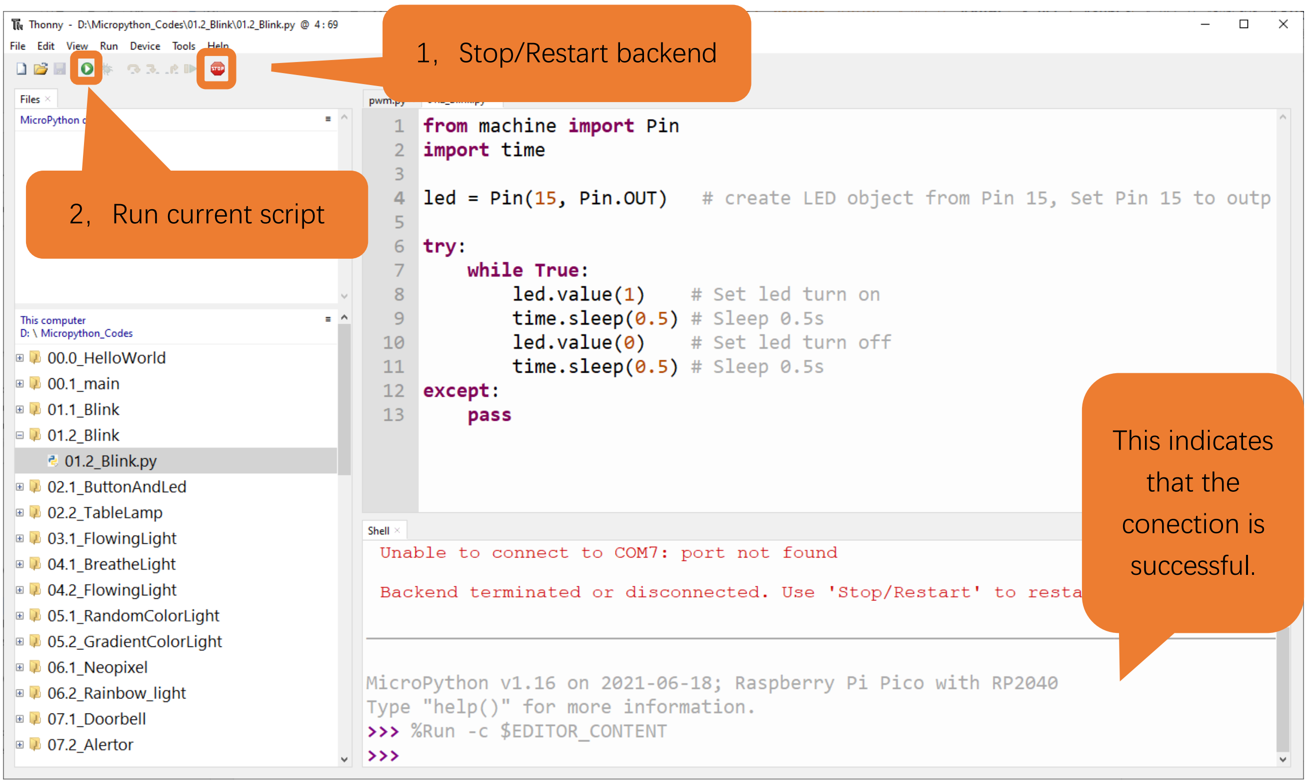
Open "Thonny", click "This computer"→"D:"→"Micropython_Codes".



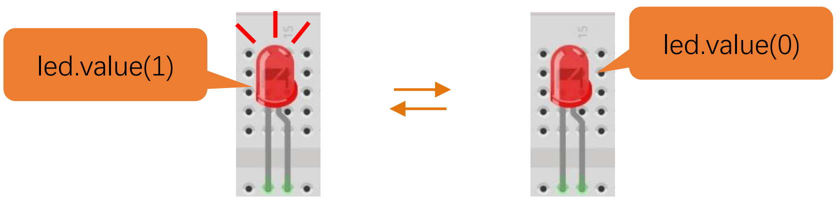

Expand folder "01.2_Blink" and double click "01.2_Blink.py" to open it. As shown in the illustration below.



Make sure Raspberry Pi Pico has been connected with the computer. Click "Stop/Restart backend", and then wait to see what interface will show up.

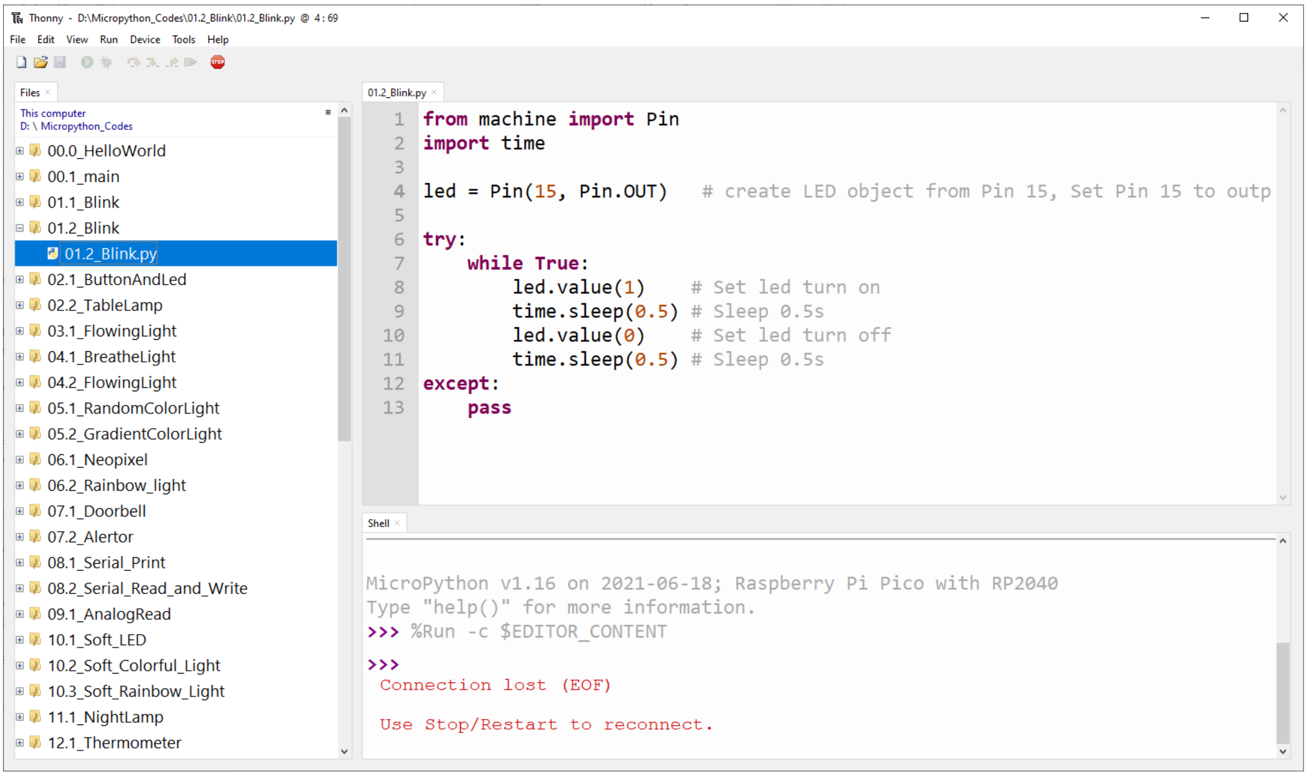



Click "Run current script" shown in the box above, the code starts to be executed and the LED in the circuit starts to blink. Press Ctrl+C or click "Stop/Restart backend" to exit the program.



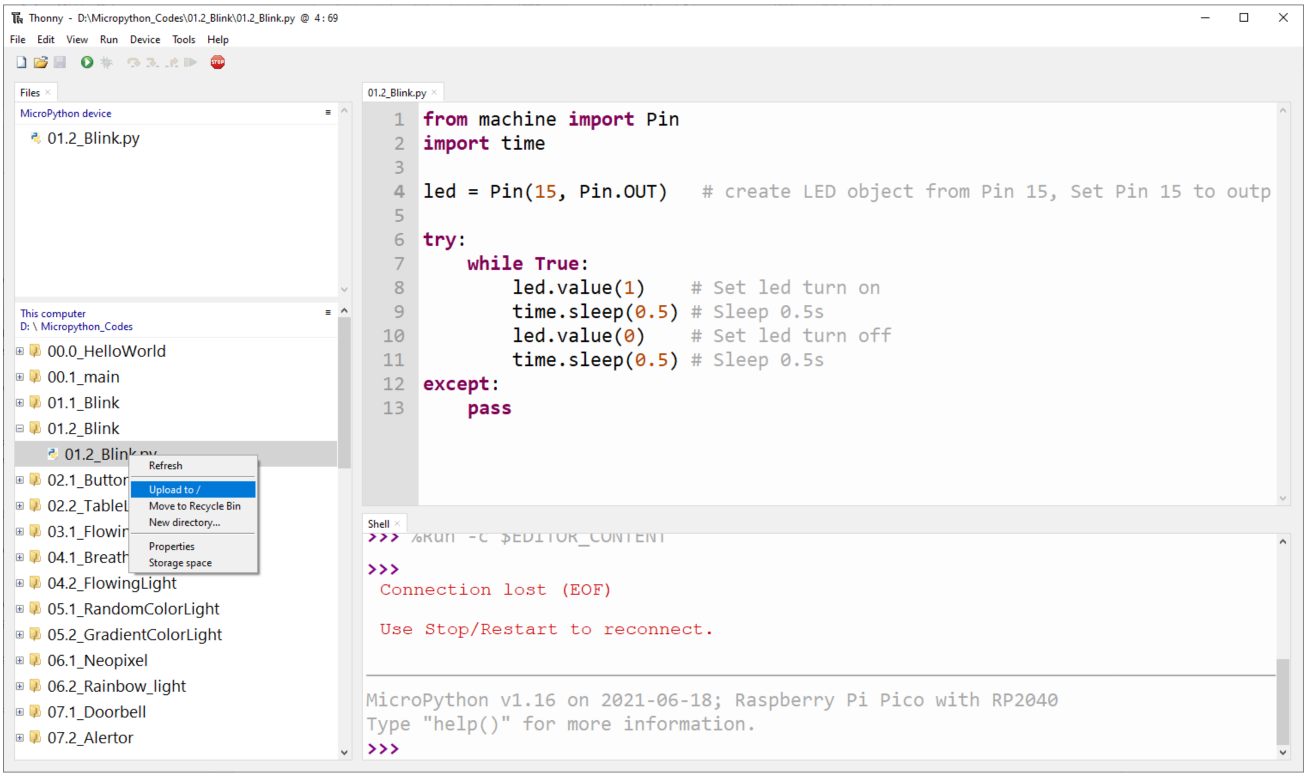
Note:

This is the code running online. If you disconnect USB cable and repower Raspberry Pi Pico, LED stops blinking and the following messages will be displayed in Thonny.

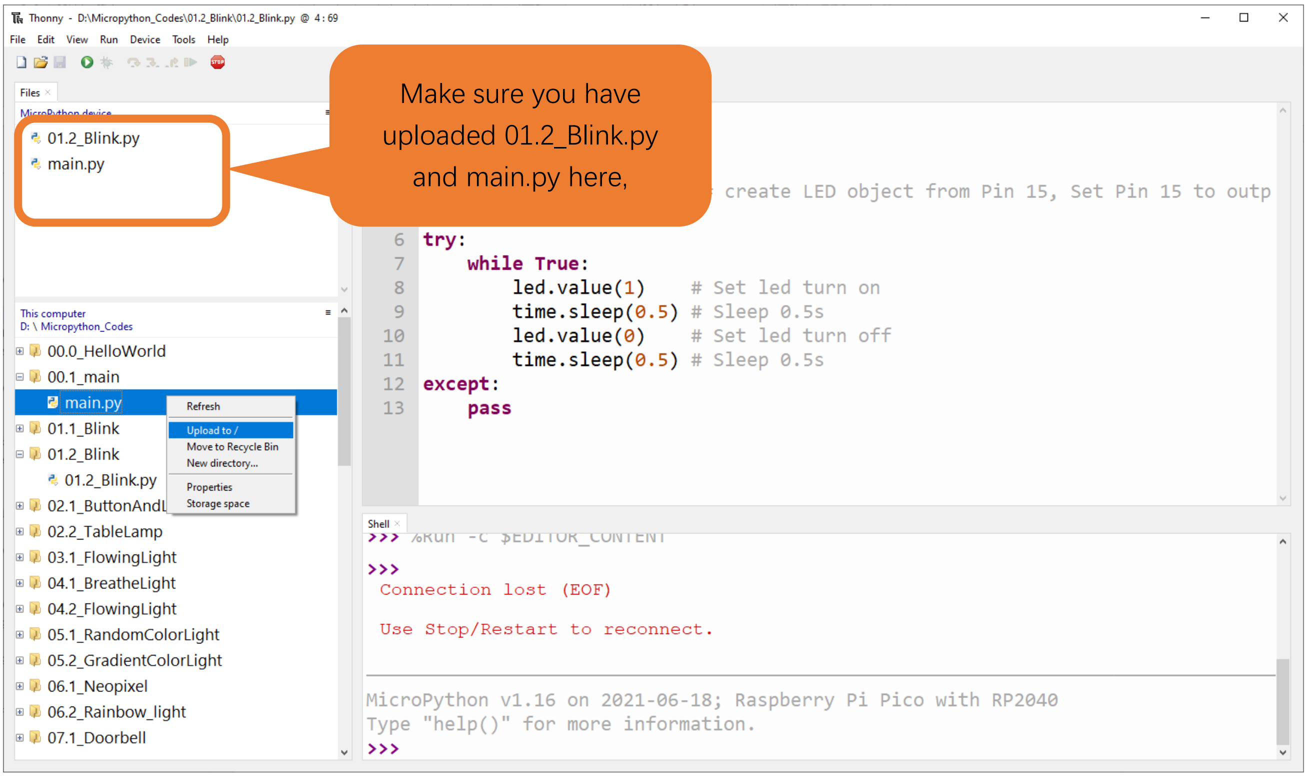



## Uploading code to Raspberry Pi Pico

As shown in the following illustration, right-click the file 01.2_Blink.py and select “Upload to /” to upload code to Raspberry Pi Pico.



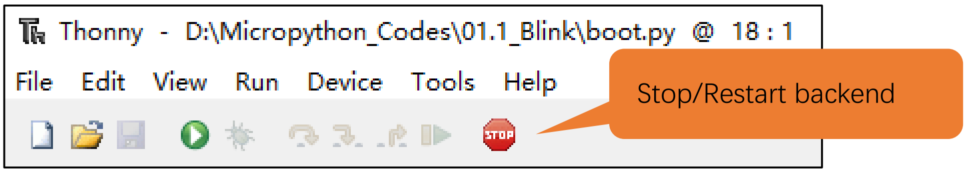
Upload main.py in the same way.




Disconnect Raspberry Pi Pico USB cable and reconnect it, LED on Pico will blink repeatedly.

Note:

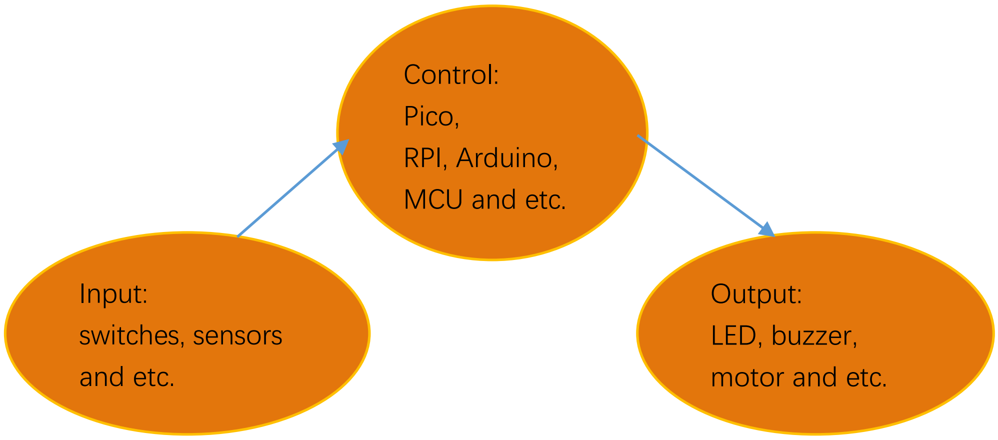
Codes here is run offline. If you want to stop running offline and enter Shell, just click “Stop” in Thonny.




**If you have any concerns, please contact us via: support@freenove.com**

# Chapter 2 Button & LED

Usually, there are three essential parts in a complete automatic control device: INPUT, OUTPUT, and CONTROL. In last section, the LED module was the output part and Raspberry Pi Pico was the control part. In practical applications, we not only make LEDs blink, but also make a device sense the surrounding environment, receive instructions and then take the appropriate action such as LEDs light up, make a buzzer turn ON and so on.

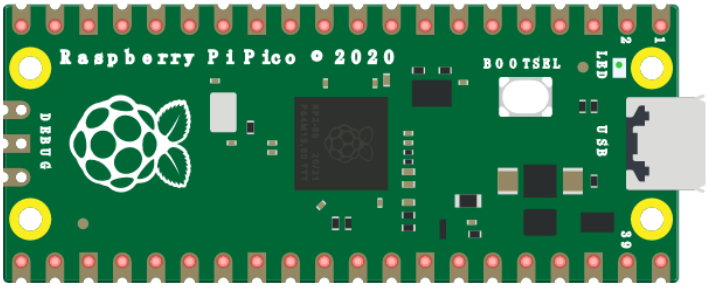
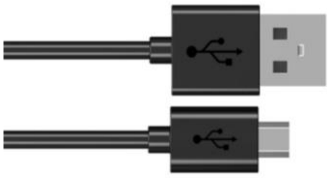
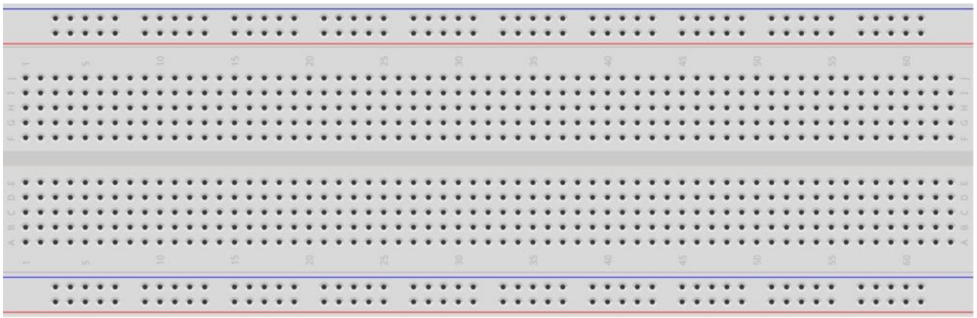



Next we make a simple project: build a control system with button, LED and Raspberry Pi Pico.
Input: Button
Control: Raspberry Pi Pico
Output: LED

# Project 2.1 Button & LED

In the project, we will control the LED state through a Push Button Switch. When the button is pressed, our LED will turn ON, and when it is released, the LED will turn OFF. This describes a Momentary Switch.

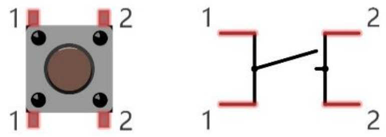
## Component List

<table>
<tr><td colspan="3">Raspberry Pi Pico x1<br></td><td colspan="2">USB cable x1<br></td></tr>
<tr><td colspan="5">Breadboard x1<br></td></tr>
<tr><td>Jumper<br></td><td>LED x1<br></td><td>Resistor 220Ω x1<br></td><td>Resistor 10kΩ x2<br></td><td>Push button x1<br></td></tr>
</table>

# Component knowledge

### Push button

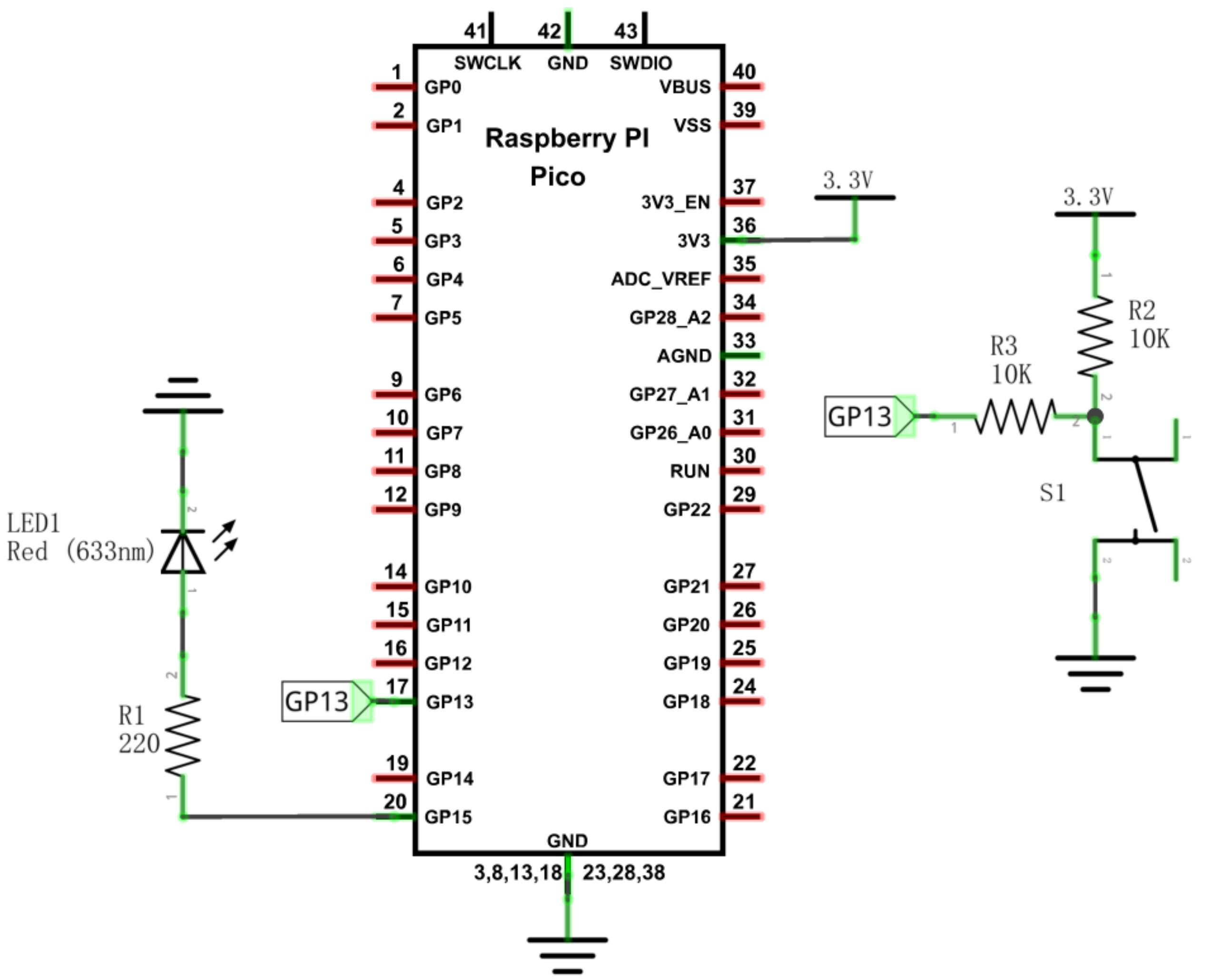
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



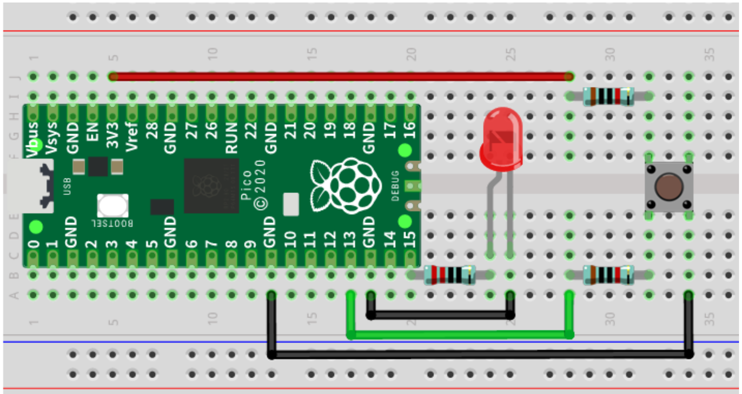

When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

# Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

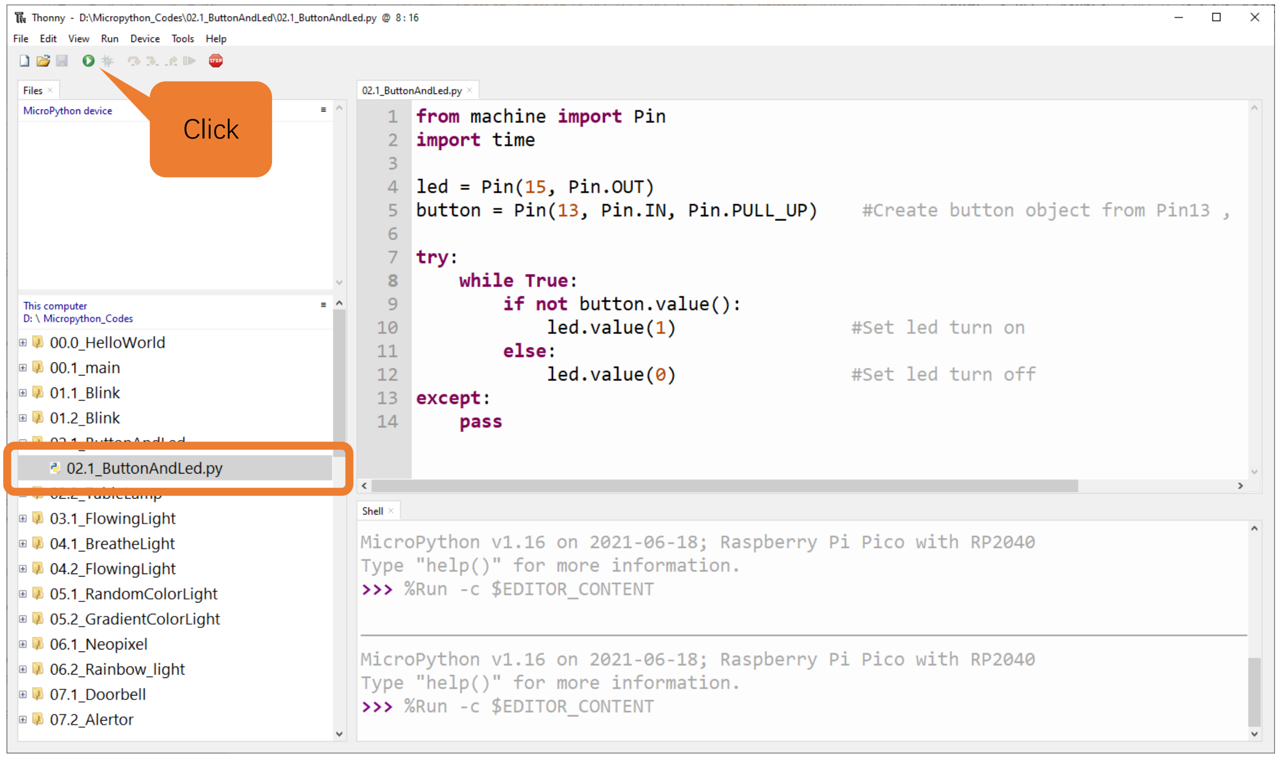
## Code

This project is designed to learn to control an LED with a push button switch. First, we need to read the state of the switch and then decide whether the LED is turned on or not based on it.

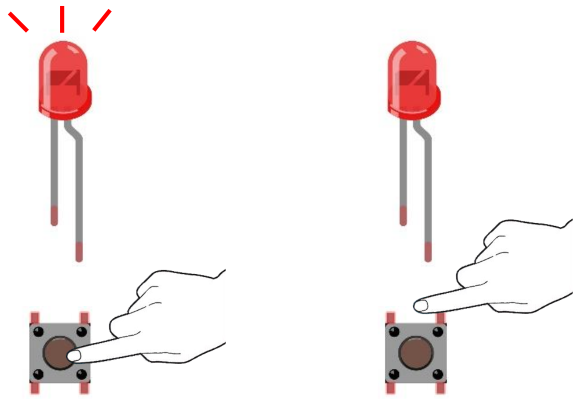
Move the program folder "**Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "02.1_ButtonAndLed" and double click "02.1_ButtonAndLed.py".

### 02.1_ButtonAndLed



Click "Run current script" shown in the box of the above illustration, press the push button switch, LED turns ON; release the switch, LED turns OFF. Press Ctrl+C or click "Stop/Restart backend" to exit program.



The following is the program code:

```
from machine import Pin
import time

led = Pin(15, Pin.OUT)
button = Pin(13, Pin.IN, Pin.PULL_UP)    #Create button object from Pin13 , Pin13 to input

try:
    while True:
        if not button.value():
            led.value(1)            #Set led turn on
        else:
            led.value(0)            #Set led turn off
except:
    pass
```

In this project, we use the Pin module of the machine, so before initializing the Pin, we need to import this module first.

```
from machine import Pin
```

In the circuit connection, LED and Button are connected with GP15 and GP13 respectively, so define LED and button as 0 and 3 respectively.

```
led = Pin(15, Pin.OUT)
button = Pin(13, Pin.IN, Pin.PULL_UP)    #Create button object from Pin13, Set Pin13 to input
```

Read the pin state of button with value() function. Press the button switch, the function returns low level and the result of "if" is true, and then LED will be turned ON; Otherwise, LED is turned OFF.

```
        if not button.value():
            led.value(1)            #Set led turn on
        else:
            led.value(0)            #Set led turn off
```

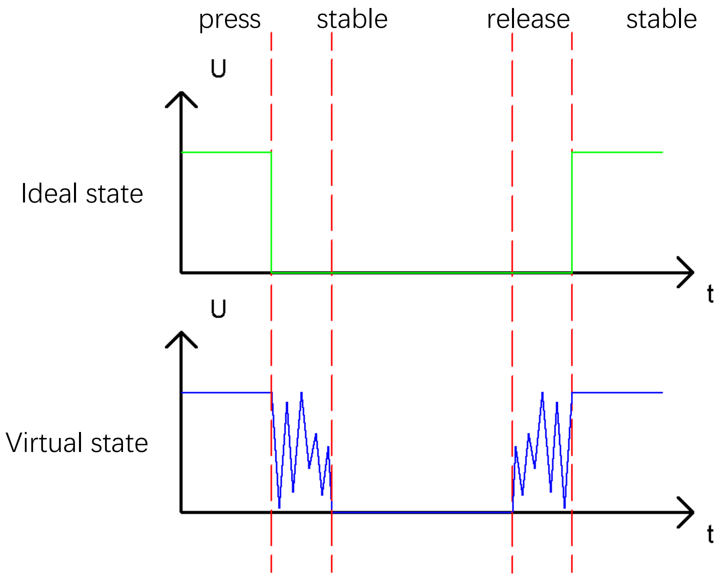
# Project 2.2 MINI table lamp

We will also use a Push Button Switch, LED and Raspberry Pi Pico to make a MINI Table Lamp but this will function differently: Press the button, the LED will turn ON, and pressing the button again, the LED turns OFF. The ON switch action is no longer momentary (like a door bell) but remains ON without needing to continually press on the Button Switch.

First, let us learn something about the push button switch.

## Debounce for Push Button

When a Momentary Push Button Switch is pressed, it will not change from one state to another state immediately. Due to tiny mechanical vibrations, there will be a short period of continuous buffeting before it completely reaches another state too fast for Humans to detect but not for computer microcontrollers. The same is true when the push button switch is released. This unwanted phenomenon is known as “bounce”.




Therefore, if we can directly detect the state of the Push Button Switch, there are multiple pressing and releasing actions in one pressing cycle. This buffeting will mislead the high-speed operation of the microcontroller to cause many false decisions. Therefore, we need to eliminate the impact of buffeting. Our solution: to judge the state of the button multiple times. Only when the button state is stable (consistent) over a period of time, can it indicate that the button is actually in the ON state (being pressed).

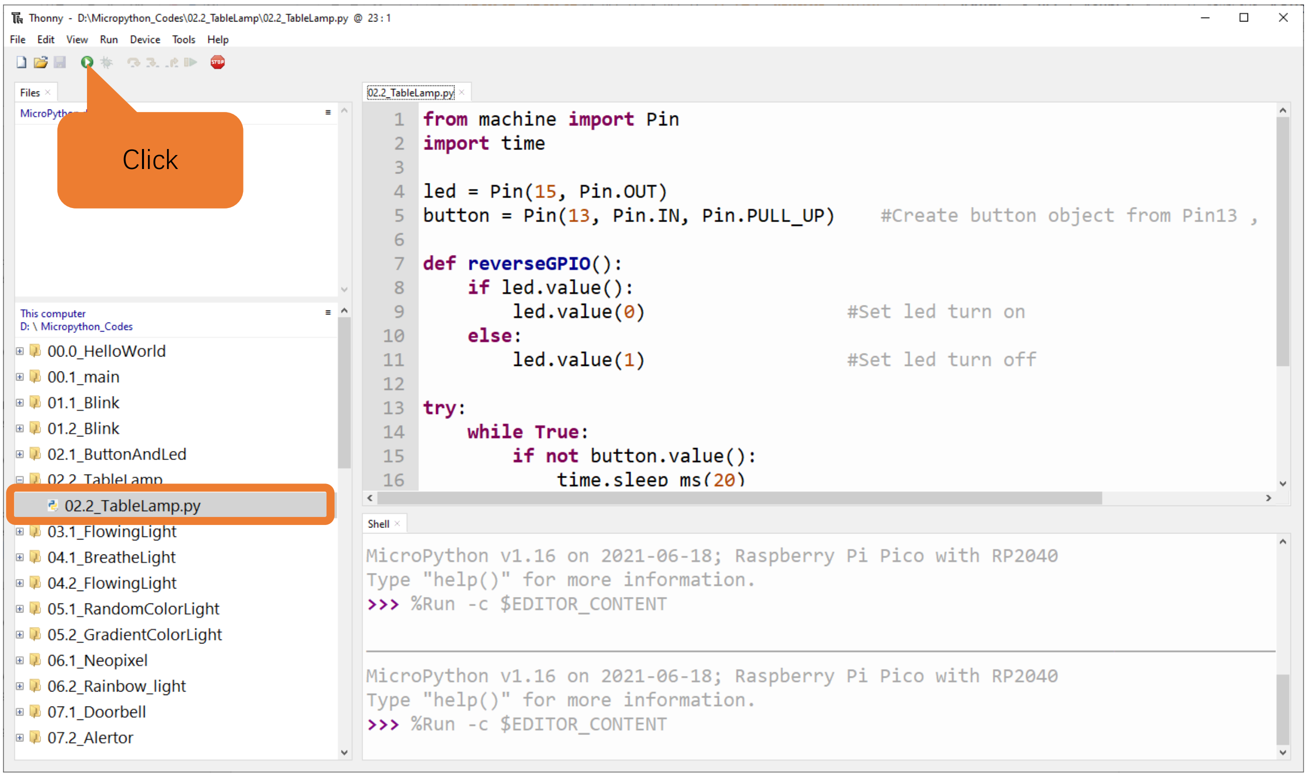
This project needs the same components and circuits as we used in the previous section.

# Code

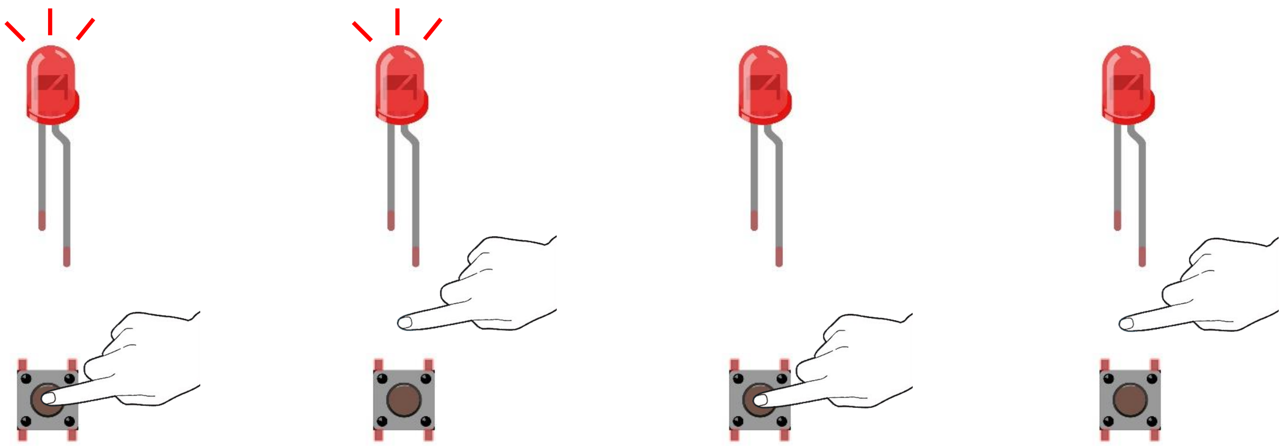
## 02.2_Tablelamp

Move the program folder "**Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Python_Codes**" to disk(D) in advance with the path of "**D:/Micropython_Codes**".

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "02.2_TableLamp"and double click "02.2_TableLamp.py".




Click "Run current script" shown in the box of the above illustration, press the push button switch, LED turns ON ; press it again, LED turns OFF. Press Ctrl+C or click "Stop/Restart backend" to eixt file.



**If you have any concerns, please contact us via: support@freenove.com**

The following is the program code:

```
from machine import Pin
import time

led = Pin(15, Pin.OUT)
button = Pin(13, Pin.IN, Pin.PULL_UP)     #Create button object from Pin13 , Set Pin13 to input

def reverseGPIO():
    if led.value():
        led.value(0)                  #Set led turn on
    else:
        led.value(1)                  #Set led turn off

try:
    while True:
        if not button.value():
            time.sleep_ms(20)
            if not button.value():
                reverseGPIO()
                while not button.value():
                    time.sleep_ms(20)
except:
    pass
```

When the button is detected to be pressed, delay 20ms to avoid the effect of bounce, and then check whether the button has been pressed again. If so, the conditional statement will be executed, otherwise it will not be executed.

```
while True:
    if not button.value():
        time.sleep_ms(20)
        if not button.value():
            reverseGPIO()
            while not button.value():
                time.sleep_ms(20)
```

Customize a function and name it reverseGPIO(), which reverses the output level of the LED.

```
def reverseGPIO():
    if led.value():
        led.value(0)                  #Set led turn on
    else:
        led.value(1)                  #Set led turn off
```

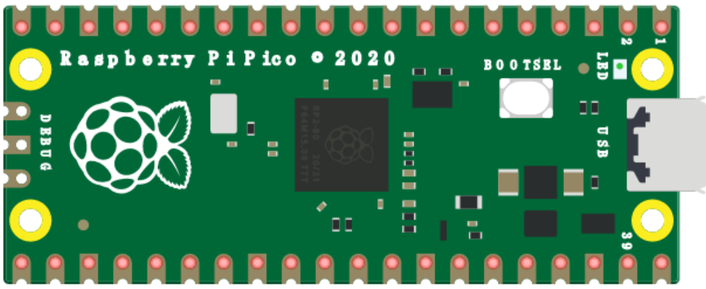
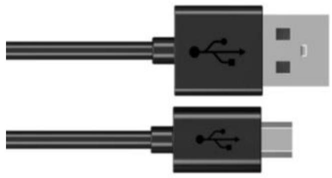
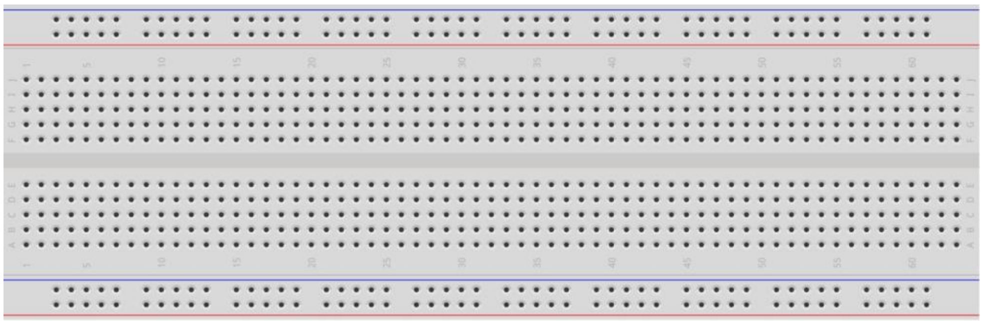
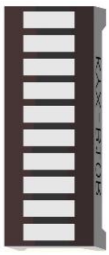
# Chapter 3 LED Bar

We have learned how to control an LED blinking, next we will learn how to control a number of LEDs.

## Project 3.1 Flowing Light

In this project, we use a number of LEDs to make a flowing light.

### Component List

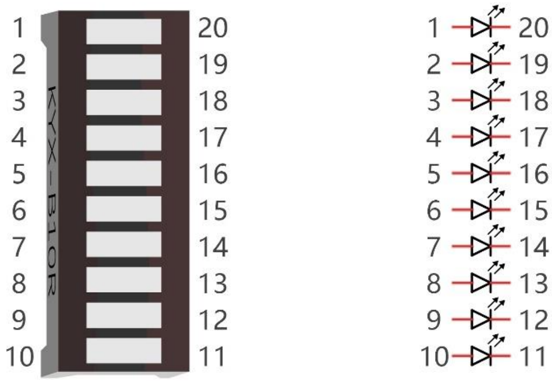
<table>
<tr><td colspan="2">Raspberry Pi Pico x1<br></td><td>USB cable x1<br></td></tr>
<tr><td colspan="3">Breadboard x1<br></td></tr>
<tr><td>Jumper<br></td><td>LED bar graph x1<br></td><td>Resistor 220Ω x10<br></td></tr>
</table>

## Component knowledge

Let us learn about the basic features of these components to use and understand them better.

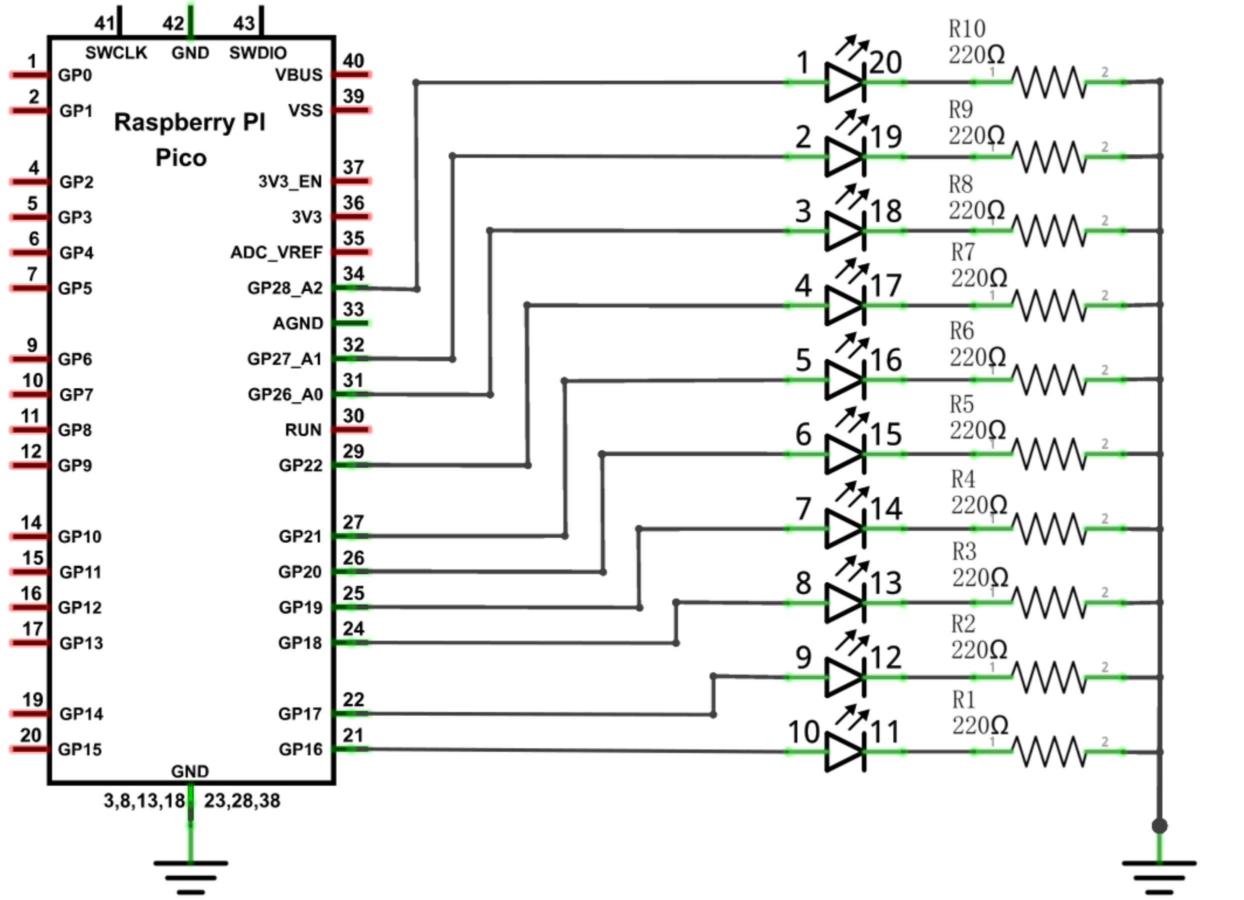
### LED bar

A Bar Graph LED has 10 LEDs integrated into one compact component. The two rows of pins at its bottom are paired to identify each LED like the single LED used earlier.

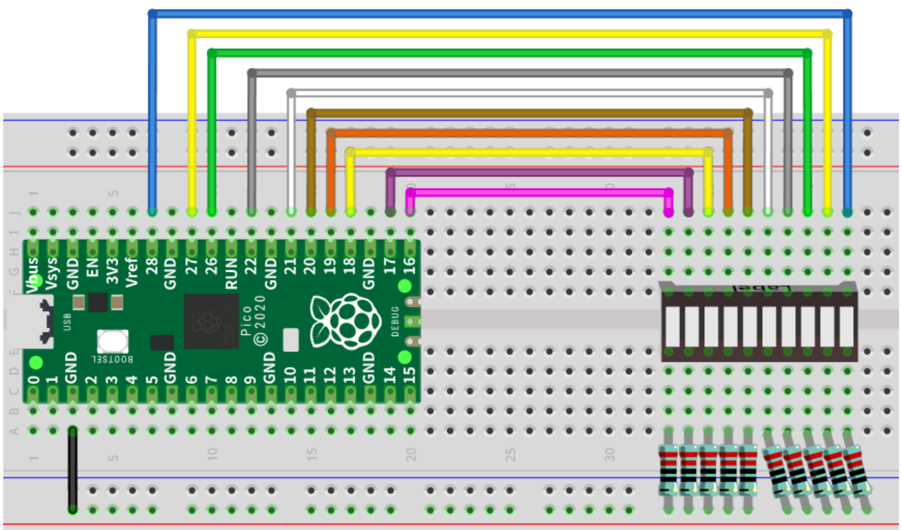


## Circuit

Schematic diagram




Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

**If LEDbar doesn't work, try to rotate LEDbar for 180°. The label is random.**

**Any concerns? ✉ support@freenove.com**

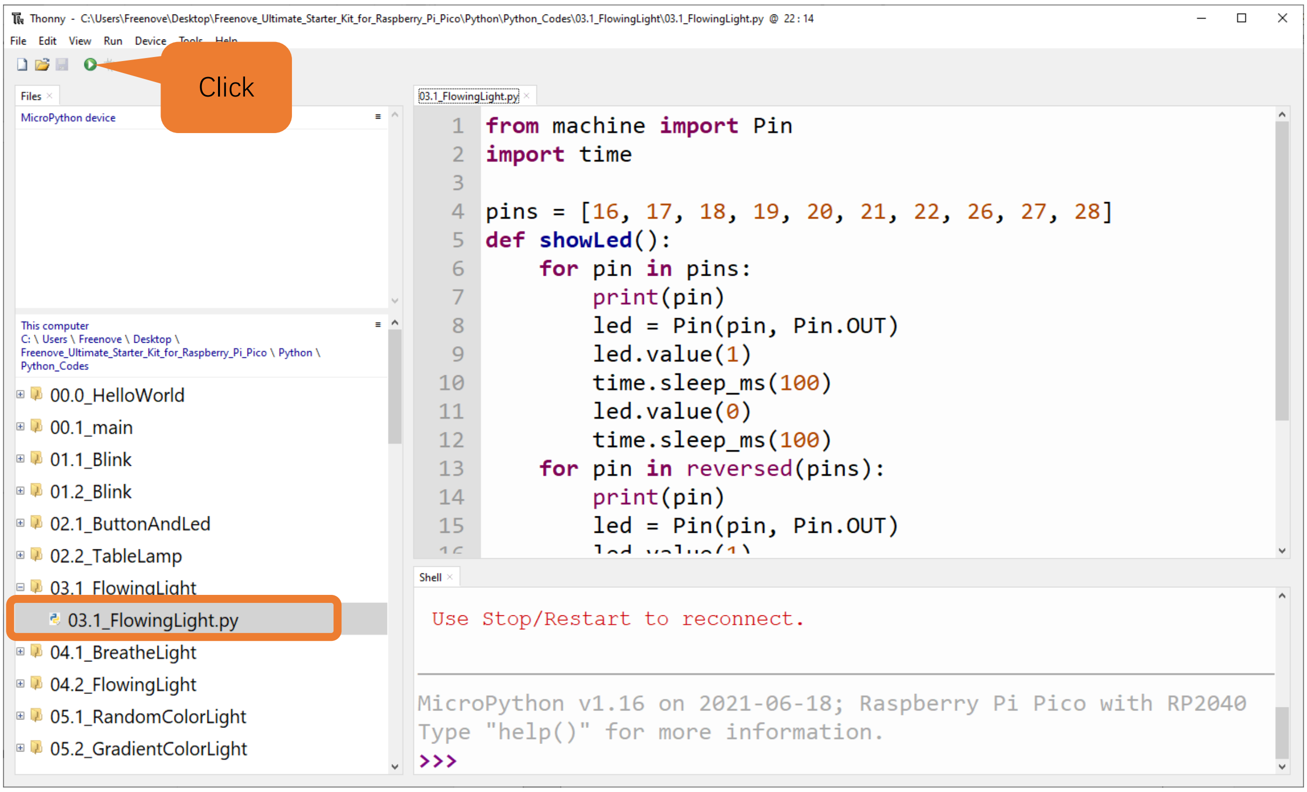
# Code

This project is designed to make a flowing water lamp. Which are these actions: First turn LED #1 ON, then turn it OFF. Then turn LED #2 ON, and then turn it OFF... and repeat the same to all 10 LEDs until the last LED is turns OFF. This process is repeated to achieve the “movements” of flowing water.

03.1_FlowingLight

Move the program folder “**Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico/Python_Codes**” to disk(D) in advance with the path of “**D:/Micropython_Codes**”.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “03.1_FlowingLight” and double click “03.1_FlowingLight.py”.




Click “Run current script” shown in the box above, LED Bar Graph will light up from left to right and then back from right to left. Press Ctrl+C or click “Stop/Restart backend” to exit the program.

**If you have any concerns, please contact us via: support@freenove.com**

The following is the program code:

```
from machine import Pin
import time

pins = [16, 17, 18, 19, 20, 21, 22, 26, 27, 28]
def showLed():
    for pin in pins:
        led = Pin(pin, Pin.OUT)
        led.value(1)
        time.sleep_ms(100)
        led.value(0)
        time.sleep_ms(100)
    for pin in reversed(pins):
        led = Pin(pin, Pin.OUT)
        led.value(1)
        time.sleep_ms(100)
        led.value(0)
        time.sleep_ms(100)

while True:
    showLed()
```

Use an array to define 10 GPIO ports connected to LED Bar Graph for easier operation.

```
pins = [16, 17, 18, 19, 20, 21, 22, 26, 27, 28]
```

Use two for loops to turn on LEDs separately from left to right and then back from right to left.

```
def showLed():
    for pin in pins:
        led = Pin(pin, Pin.OUT)
        led.value(1)
        time.sleep_ms(100)
        led.value(0)
        time.sleep_ms(100)
    for pin in reversed(pins):
        led = Pin(pin, Pin.OUT)
        led.value(1)
        time.sleep_ms(100)
        led.value(0)
        time.sleep_ms(100)
```

Reference

| for |
| --- |
| For loop is used to execute a program endlessly and iterate in the order of items (a list or a string) in the sequence.<br>Commonly used:<br>“for pin in pins”<br>Pins is a list of elements that are iterated over by a for loop and assigned to pin each time.<br>“for i in range(start, end, num: 1)”<br>start: initial value, from which the for loop starts counting. The default initial value is 0. For example, range(5) equals to range(0, 5).<br>end: the value with which the function ends. For loop counts till it arrives at this value, but this value isn’t included in the counting.<br>num: Num is automatically added each time to the data. The default value is 1.<br>range() function returns a sequence number which is assigned to I by for loop. |

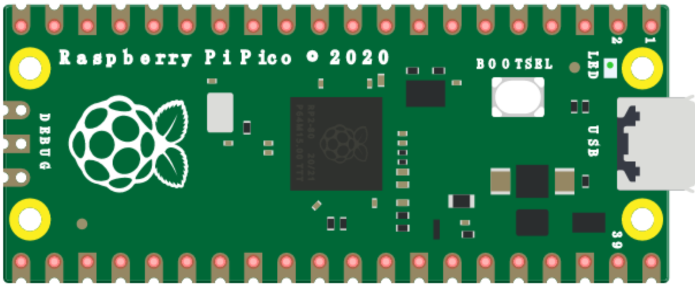
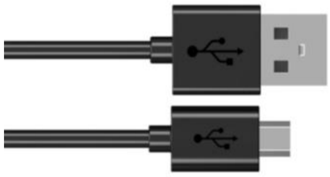
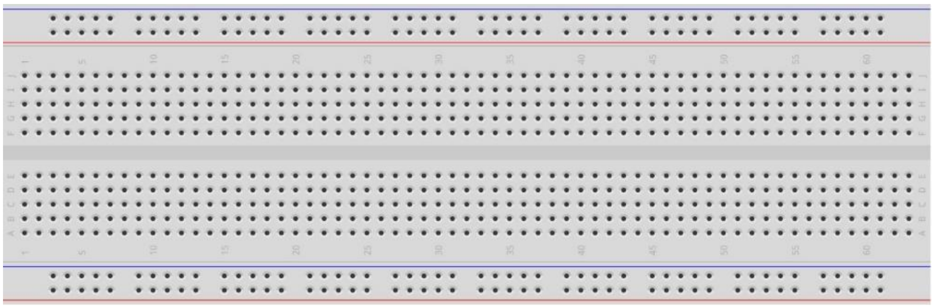
# Chapter 4 Analog & PWM

In previous study, we have known that one button has two states: pressed and released, and LED has light-on/off state, then how to enter a middle state? How to output an intermediate state to let LED "semi bright"? That's what we're going to learn.
First, let's learn how to control the brightness of an LED.

## Project 4.1 Breathing LED

Breathing light, that is, LED is turned from off to on gradually, and gradually from on to off, just like "breathing". So, how to control the brightness of an LED? We will use PWM to achieve this target.

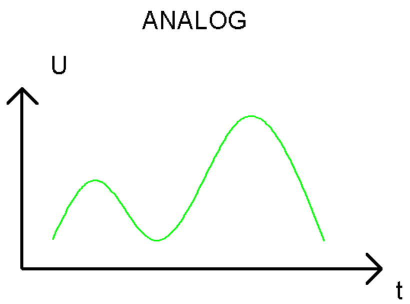
### Component List

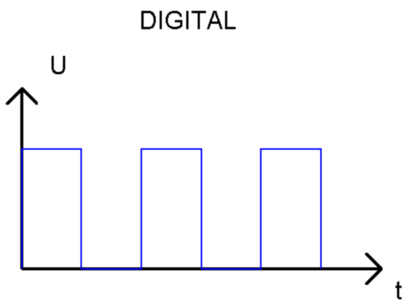
<table>
<tr><td colspan="2">Raspberry Pi Pico x1<br>
</td><td>USB cable x1<br></td></tr>
<tr><td colspan="3">Breadboard x1<br></td></tr>
<tr><td>LED x1<br></td><td>Resistor 220Ω x1<br></td><td>Jumper<br></td></tr>
</table>

# Related knowledge

## Analog & Digital

An Analog Signal is a continuous signal in both time and value. On the contrary, a Digital Signal or discrete-time signal is a time series consisting of a sequence of quantities. Most signals in life are analog signals. A familiar example of an Analog Signal would be how the temperature throughout the day is continuously changing and could not suddenly change instantaneously from 0℃ to 10℃. However, Digital Signals can instantaneously change in value. This change is expressed in numbers as 1 and 0 (the basis of binary code). Their differences can more easily be seen when compared when graphed as below.



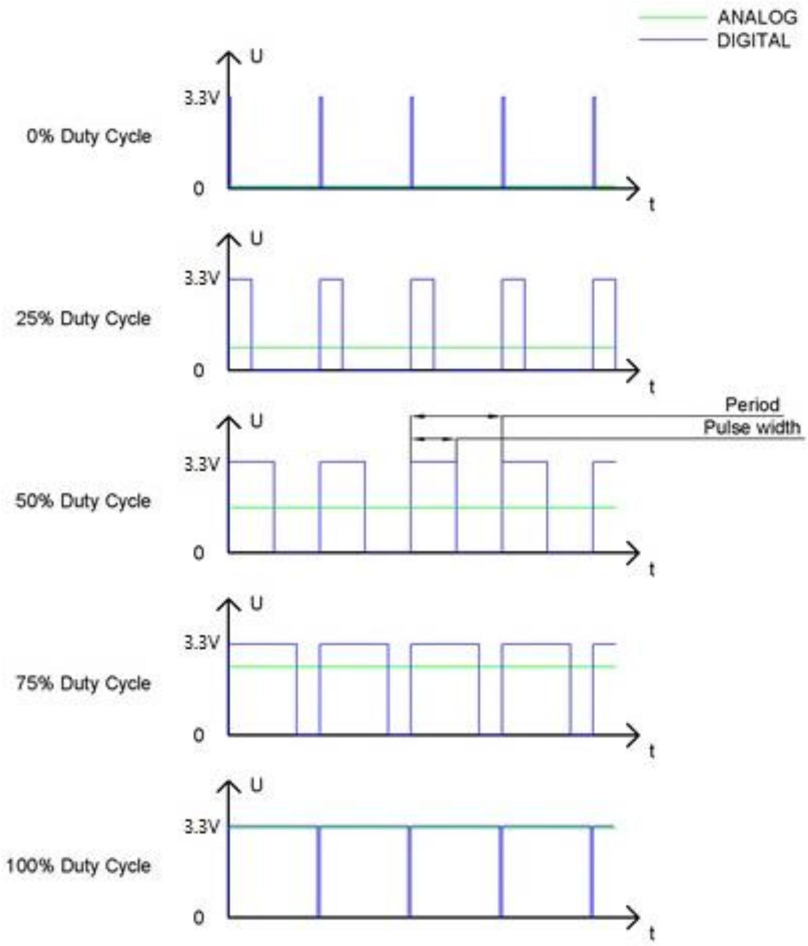

In practical application, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

## PWM

PWM, Pulse-Width Modulation, is a very effective method for using digital signals to control analog circuits. Common processors cannot directly output analog signals. PWM technology makes it very convenient to achieve this conversion (translation of digital to analog signals)

PWM technology uses digital pins to send certain frequencies of square waves, that is, the output of high levels and low levels, which alternately last for a while. The total time for each set of high levels and low levels is generally fixed, which is called the period (Note: the reciprocal of the period is frequency). The time of high level outputs are generally called "pulse width", and the duty cycle is the percentage of the ratio of pulse duration, or pulse width (PW) to the total period (T) of the waveform.

The longer the output of high levels last, the longer the duty cycle and the higher the corresponding voltage in the analog signal will be. The following figures show how the analog signal voltages vary between 0V-5V (high level is 5V) corresponding to the pulse width 0%-100%:




The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on.
It is evident from the above that PWM is not real analog, and the effective value of the voltage is equivalent to the corresponding analog. so, we can control the output power of the LED and other output modules to achieve different effects.

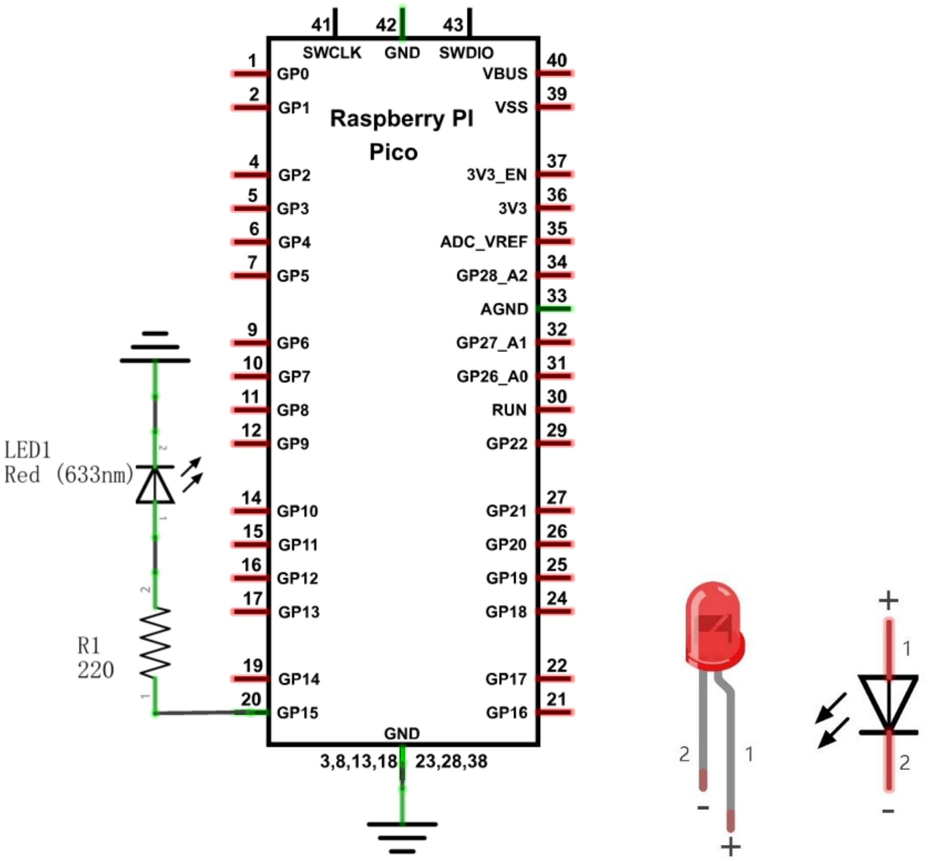
### Raspberry Pi Pico and PWM

Raspberry Pi Pico has 16 PWM channels, each of which can control frequency and duty cycle independently, with the clock frequency ranges from 7Hz to 125MHz. Every pin on Raspberry Pi Pico can be configured as PWM output.

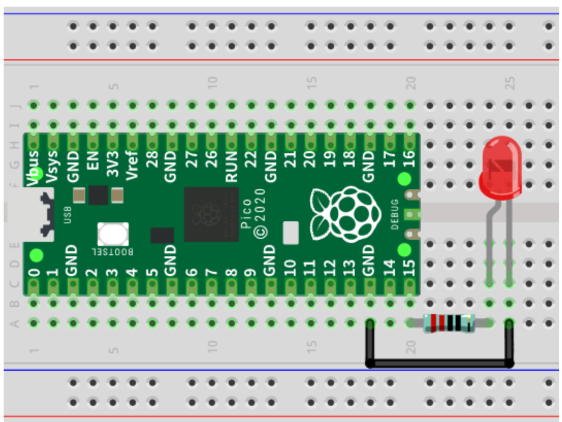
# Circuit

This circuit is the same as the one in project Blink.

Schematic diagram




Hardware connection. If you need any support, please feel free to contact us via: [support@freenove.com](mailto:support@freenove.com)



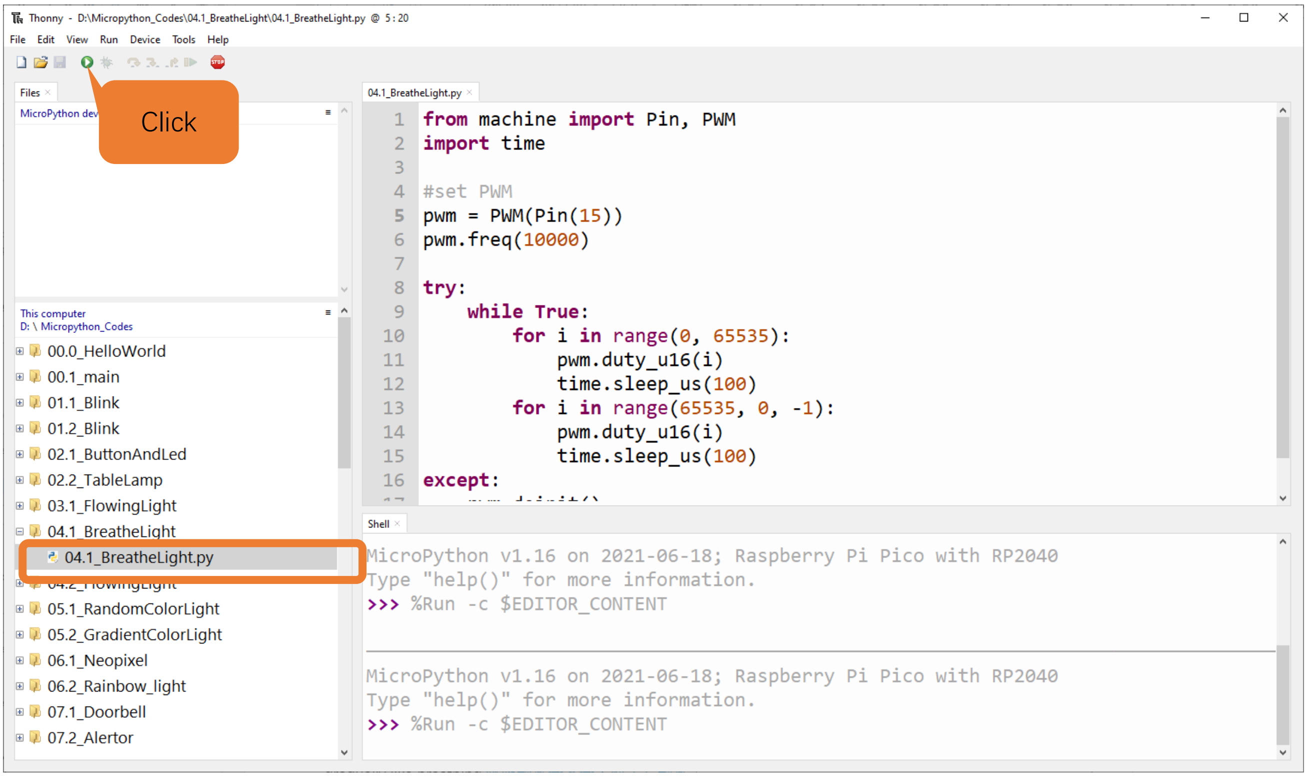
Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

# Code

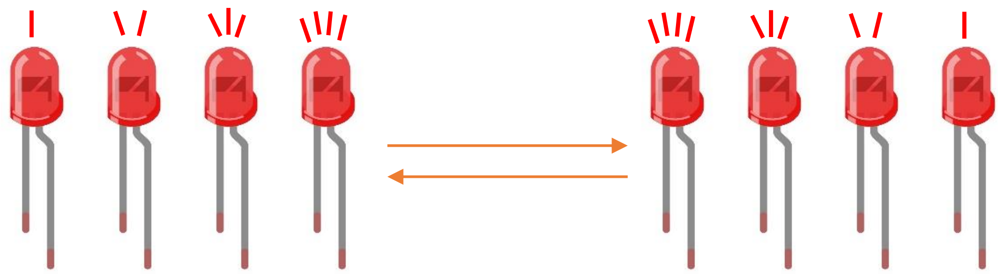
This project is designed to make PWM output GP15 with pulse width increasing from 0% to 100%, and then reducing from 100% to 0% gradually.

Open “Thonny”, click“This computer” → “D:” → “Micropython_Codes” → “04.1_BreatheLight” and double click “04.1_BreatheLight.py”.

04.1_BreatheLight




Click “Run current script”, and you'll see that LED is turned from ON to OFF and then back from OFF to ON gradually like breathing. Press Ctrl+C or click “Stop/Restart backend” to exit program.



The following is the program code:

```
from machine import Pin, PWM
import time

#set PWM
pwm = PWM(Pin(15))
pwm.freq(10000)

try:
    while True:
        for i in range(0, 65535):
            pwm.duty_u16(i)
            time.sleep_us(100)
        for i in range(65535, 0, -1):
            pwm.duty_u16(i)
            time.sleep_us(100)
except:
    pwm.deinit()
```

Create a PWM object and configure GP15 as PWM output pin. Call freq() to set PWM output frequency of GP15 to 10000Hz.

```
pwm = PWM(Pin(15))
pwm.freq(10000)
```

The range of duty cycle is 0-65535, so we use the first for loop to control PWM to change the duty cycle value, making PWM output 0% -100%; Use the second for loop to make PWM output 100%-0%.

```
        for i in range(0, 65535):
            pwm.duty_u16(i)
            time.sleep_us(100)
        for i in range(65535, 0, -1):
            pwm.duty_u16(i)
            time.sleep_us(100)
```

Each time PWM is used, the hardware Timer will be turned ON to cooperate it. Therefore, after each use of PWM, deinit() needs to be called to turned OFF the timer. Otherwise, the PWM may fail to work next time.

```
    pwm.deinit()
```

## Reference

**Class PWM(pin)**

Before each use of PWM module, please add the statement "**from machine import PWM**" to the top of the python file.

**pin**: PWM pins are supported, such as GP(0~22)、GP(25)、GP(26~28).

**PWM.freq(freq_val)**: the function is used to set PWM frequency and returns nothing; when there is no parameter, the function obtains and returns PWM frequency.

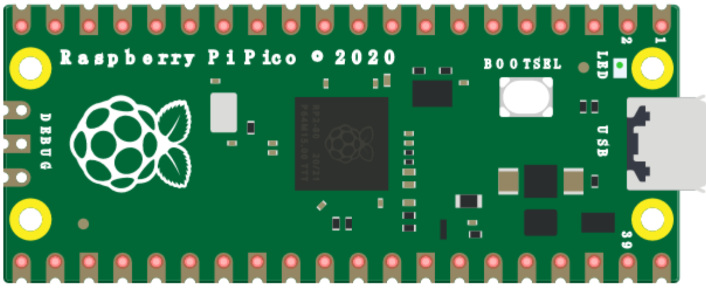
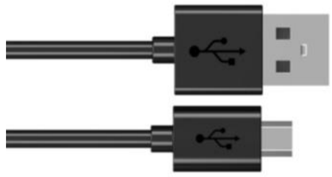
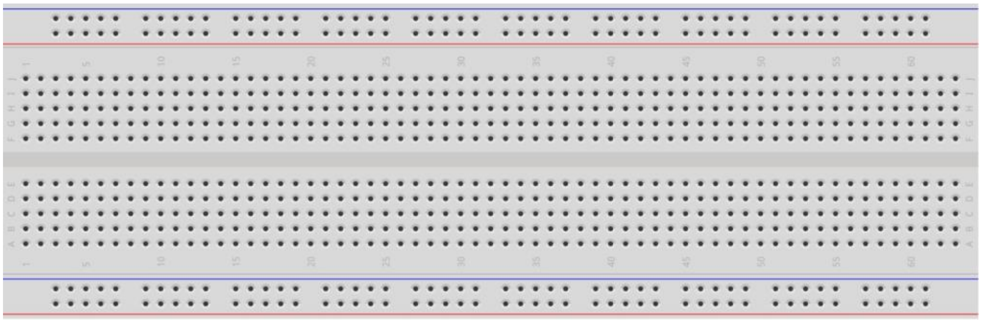
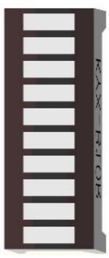
**PWM.duty_u16(duty_val)**: the function is used to set PWM duty cycle, among which, duty_val ranges from 0 to 65535. If there is no parameter, the function returns to currently set duty cycle. If duty cycle has not yet been set, it returns 0.

**PWM.deinit()**: Turn OFF PWM.

# Project 4.2 Meteor Flowing Light

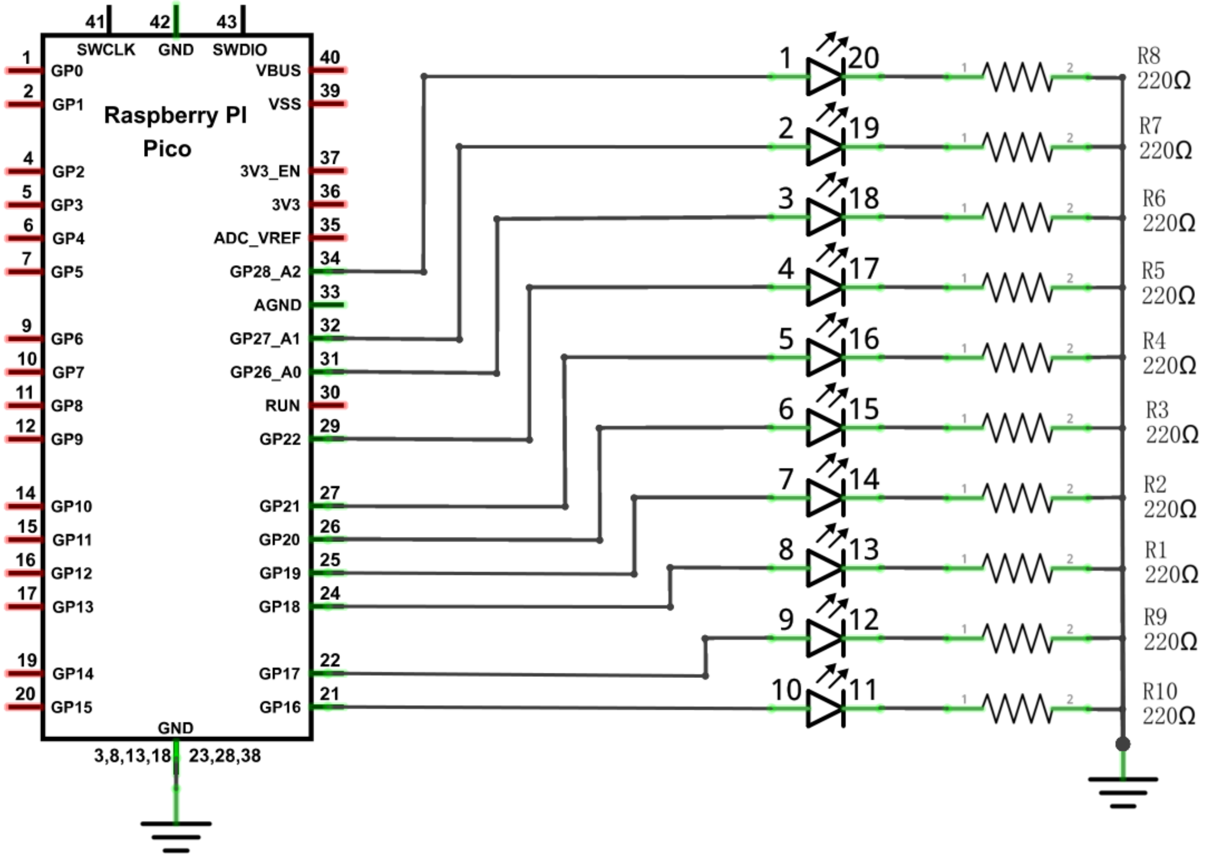
Having learned about PWM, we can use it to control LED Bar Graph and realize a cooler Flowing Light.

## Component List

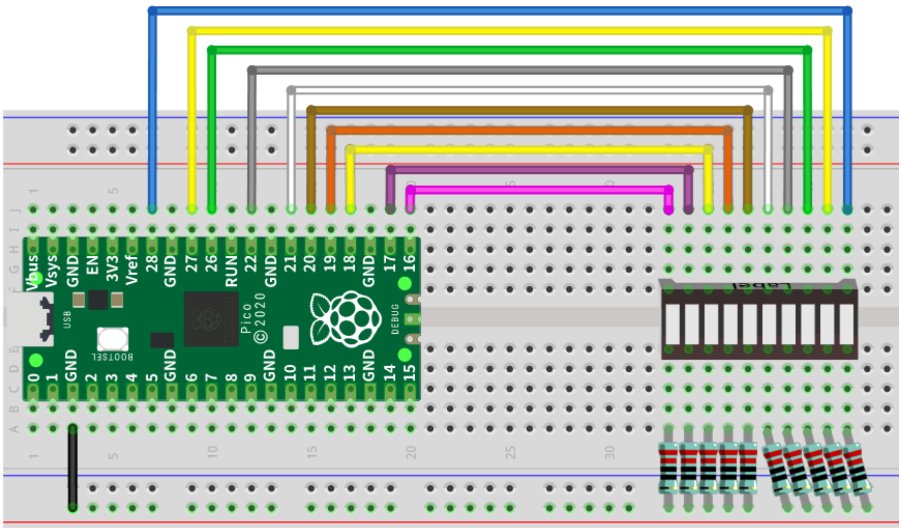
<table>
<tr><td colspan="2">Raspberry Pi Pico x1<br></td><td>USB cable x1<br></td></tr>
<tr><td colspan="3">Breadboard x1<br></td></tr>
<tr><td>Jumper<br></td><td>LED bar graph x1<br></td><td>Resistor 220Ω x10<br></td></tr>
</table>

## Circuit

Schematic diagram




Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

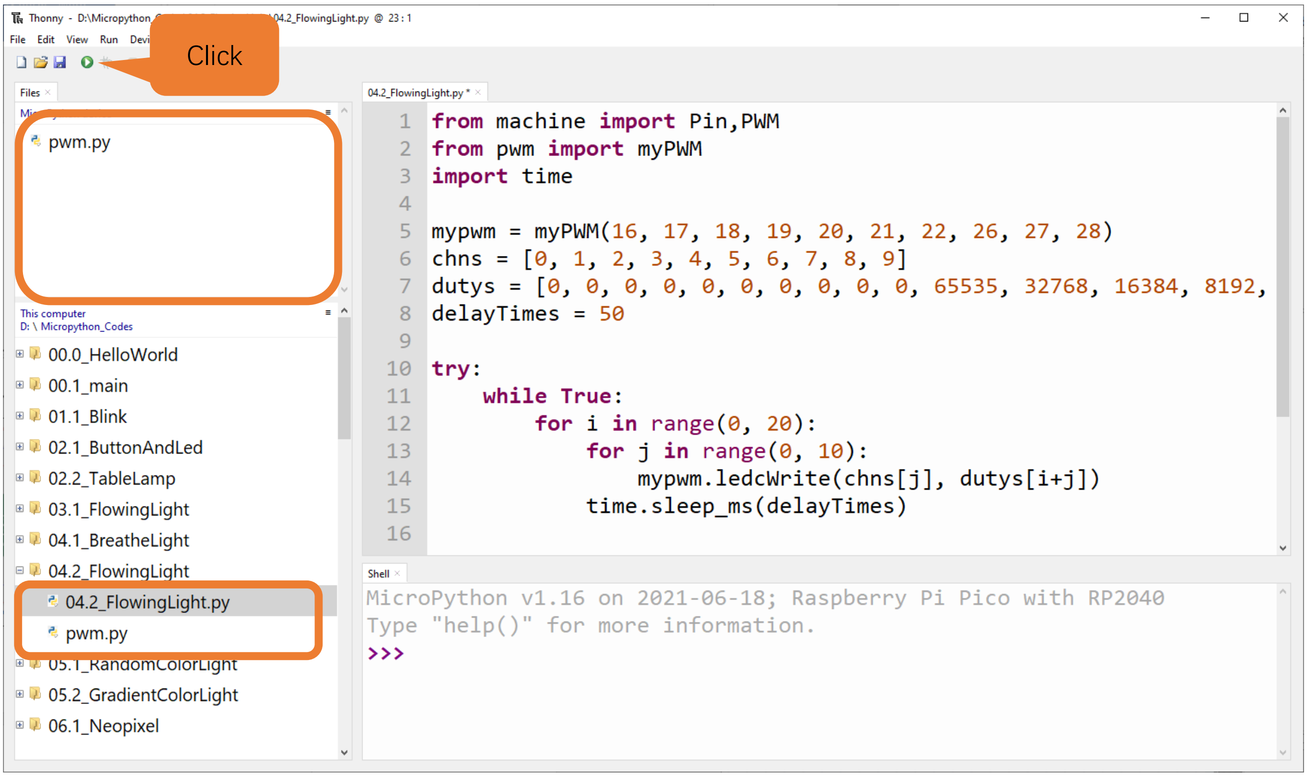
**If LEDbar doesn't work, try to rotate LEDbar for 180°. The label is random.**

## Code

Flowing Light with tail was implemented with PWM.

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "04.2_FlowingLight". Select "pwm.py", right click to select "Upload to /", wait for "pwm.py" to be uploaded to Raspberry Pi Pico and then double click "04.2_FlowingLight.py".

04.2_FlowingLight




Click "Run current script", and LED Bar Graph will gradually light up and out from left to right, then light up and out from right to left. Press Ctrl+C or click "Stop/Restart backend" to exit program.

The following is the program code:

```
from machine import Pin,PWM
from pwm import myPWM
import time

mypwm = myPWM(16, 17, 18, 19, 20, 21, 22, 26, 27, 28)
chns = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
dutys = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 65535, 32768, 16384, 8192, 4096, 2048, 1024, 512, 256, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
delayTimes = 50

try:
    while True:
        for i in range(0, 20):
```

```
            for j in range(0, 10):
                mypwm.ledcWrite(chns[j], dutys[i+j])
            time.sleep_ms(delayTimes)

        for i in range(0, 20):
            for j in range(0, 10):
                mypwm.ledcWrite(chns[9-j], dutys[i+j])
            time.sleep_ms(delayTimes)
except:
    mypwm.deinit()
```

Import the object myPWM from pwm.py and set corresponding pins for PWM channel.

```
from pwm import myPWM
...
mypwm = myPWM(16, 17, 18, 19, 20, 21, 22, 26, 27, 28)
```

Create an object for myPWM and configure 10 PWM output pins. Define 10 PWM channels and 30 pulse width values.

```
mypwm = myPWM(16, 17, 18, 19, 20, 21, 22, 26, 27, 28)
chns = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
dutys = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 65535, 32768, 16384, 8192, 4096, 2048, 1024, 512, 256, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Call ledcWrite() to set duty cycle dutys[i+j] for the chns[j] channel of PWM.

```
mypwm.ledcWrite(chns[j], dutys[i+j])
```

Turn OFF the PWM of the object myPWM.

```
mypwm.deinit()
```

In the code, a nesting of two for loops are used to achieve this effect.

```
    for i in range(0, 20):
        for j in range(0, 10):
            mypwm.ledcWrite(chns[j], dutys[i+j])
        time.sleep_ms(delayTimes)

    for i in range(0, 20):
        for j in range(0, 10):
            mypwm.ledcWrite(chns[9 -j], dutys[i+j])
        time.sleep_ms(delayTimes)
```

In the main function, a nested for loop is used to control the pulse width of the PWM. Every time i in the first for loop increases by 1, the LED Bar Graph will move one grid, and gradually change according to the value in the array dutys. As shown in the following table, the value in the second row is the value of the array dutys, and the 10 green grids in each row below represent the 10 LEDs on the LED Bar Graph. Each time i increases by 1, the value of the LED Bar Graph will move to the right by one grid, and when it reaches the end, it will move from the end to the starting point, achieving the desired effect.

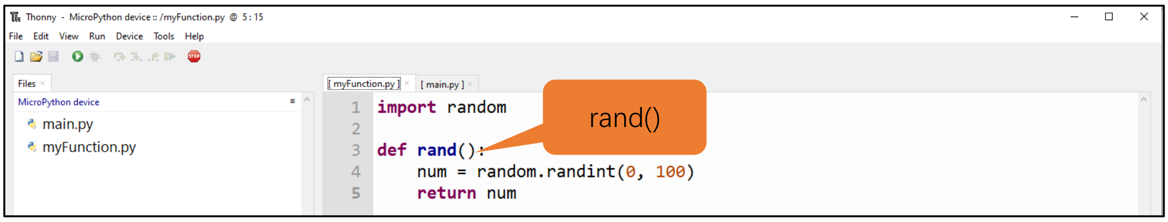
| 0 | 1 | 2 | 3 | … | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | … | 28 | 29 | 30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| di | 0 | 0 | 0 | … | 0 | 0 | 0 | 0 | 65535 | 32768 | 16384 | 8192 | 4096 | 2048 | 1024 | 512 | 256 | 128 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | | | | | | | | | | | | |
| … | | | | | | | | | | | | | | | | | | | | | | | | |
| 18 | | | | | | | | | | | | | | | | | | | | | | | | |
| 19 | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | | | | | | | | | | | | | | | | | | | | | | | | |

# How to import a custom python module

Each Python file, as long as it's stored on the file system of Raspberry Pi Pico, is a module. To import a custom module, the module file needs to be located in the MicroPython environment variable path or in the same path as the currently running program.

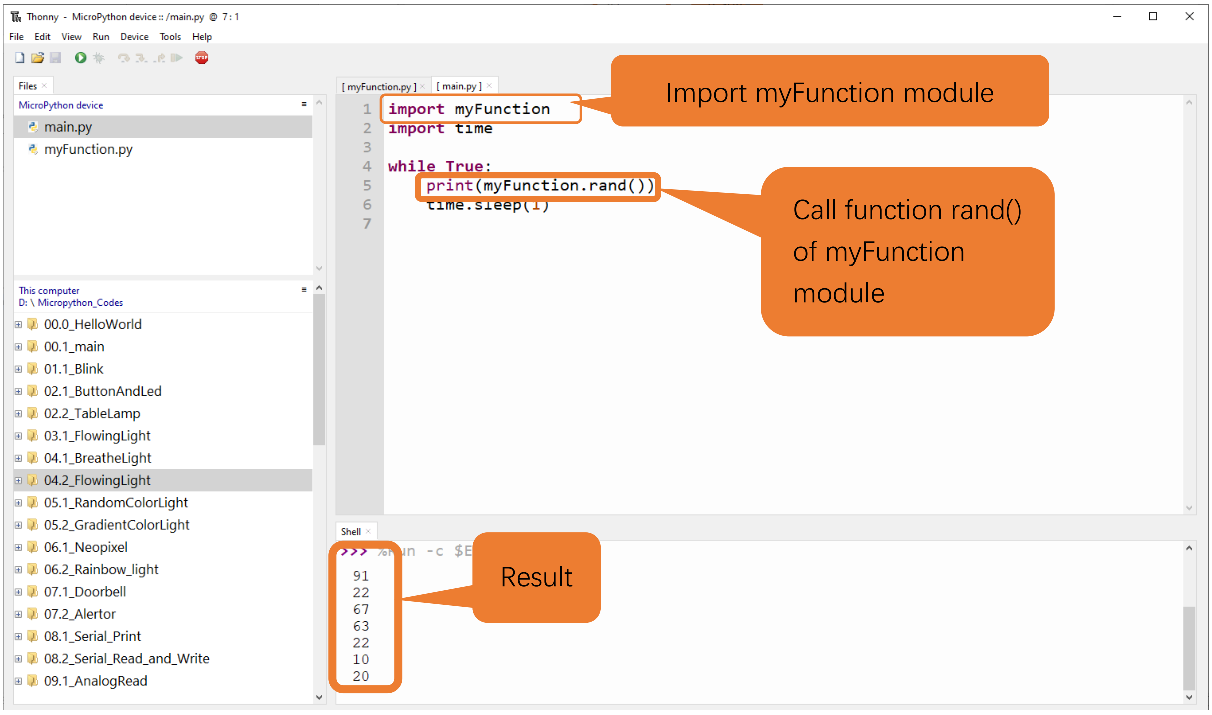
## Code

First, customize a python module “myFunction.py”. Create a new py file and name it “myFunction.py”. Write code to it and save it to Raspberry Pi Pico.




rand() function randomly creates an integer ranging from 0 to 99.

Second, import myFunction module “myFunction” to main.py.




The following is the program code:

1. myFunction.py

```
import random

def rand():
    num = random.randint(0, 100)
    return num
```

Import random module.

```
import random
```

Call randit() function in random module to randomly generate an integer at the range of 0-99 and assign it to num variable.

```
    num = random.randint(0, 100)
```

2. main.py

```
import myFunction
import time

while True:
    print(myFunction.rand())
    time.sleep(1)
```

Import myFunction module “myFunction” to main.py.

```
import myFunction
```

Call rand() in myFunction module.

```
    print(myFunction.rand())
```

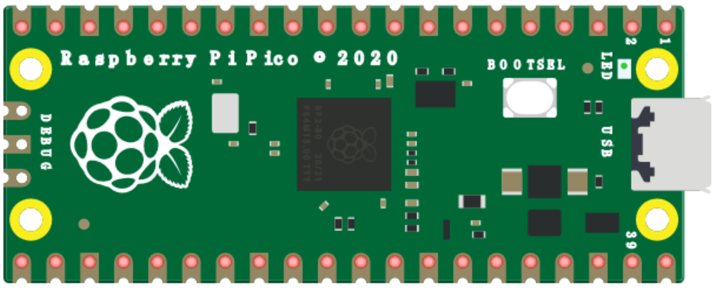
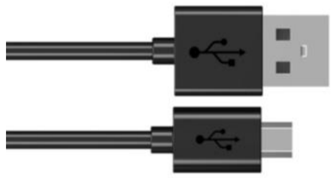
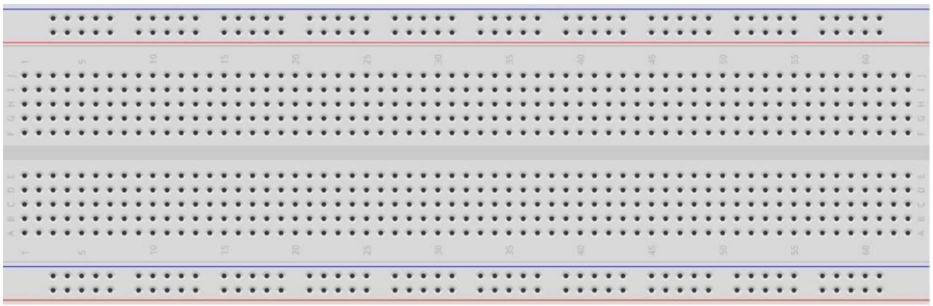
# Chapter 5 RGBLED

In this chapter, we will learn how to control an RGBLED. It can emit different colors of light. Next, we will use RGBLED to make a multicolored light.

## Project 5.1 Random Color Light

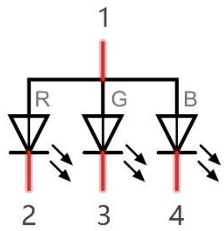
In this project, we will make a multicolored LED. And we can control RGBLED to switch different colors automatically.

### Component List

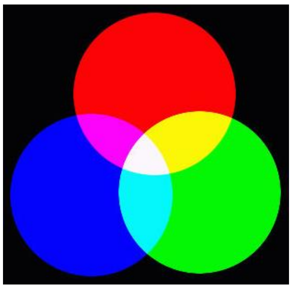
<table>
<tr><td colspan="2">Raspberry Pi Pico x1<br></td><td>USB cable x1<br></td></tr>
<tr><td colspan="3">Breadboard x1<br></td></tr>
<tr><td>RGBLED x1<br></td><td>Resistor 220Ω x3<br></td><td>Jumper<br></td></tr>
</table>

## Related knowledge

RGB LED has integrated 3 LEDs that can respectively emit red, green and blue light. And it has 4 pins. The long pin (1) is the common port, that is, 3 LED 's positive or negative port. The RGB LED with common positive port and its symbol is shown below. We can make RGB LED emit various colors of light by controlling these 3 LEDs to emit light with different brightness.


Red, green, and blue light are known as three primary colors. When you combine these three primary-color lights with different brightness, it can produce almost all kinds of visible lights. Computer screens, single pixel of cell phone screen, neon, and etc. are working under this principle.

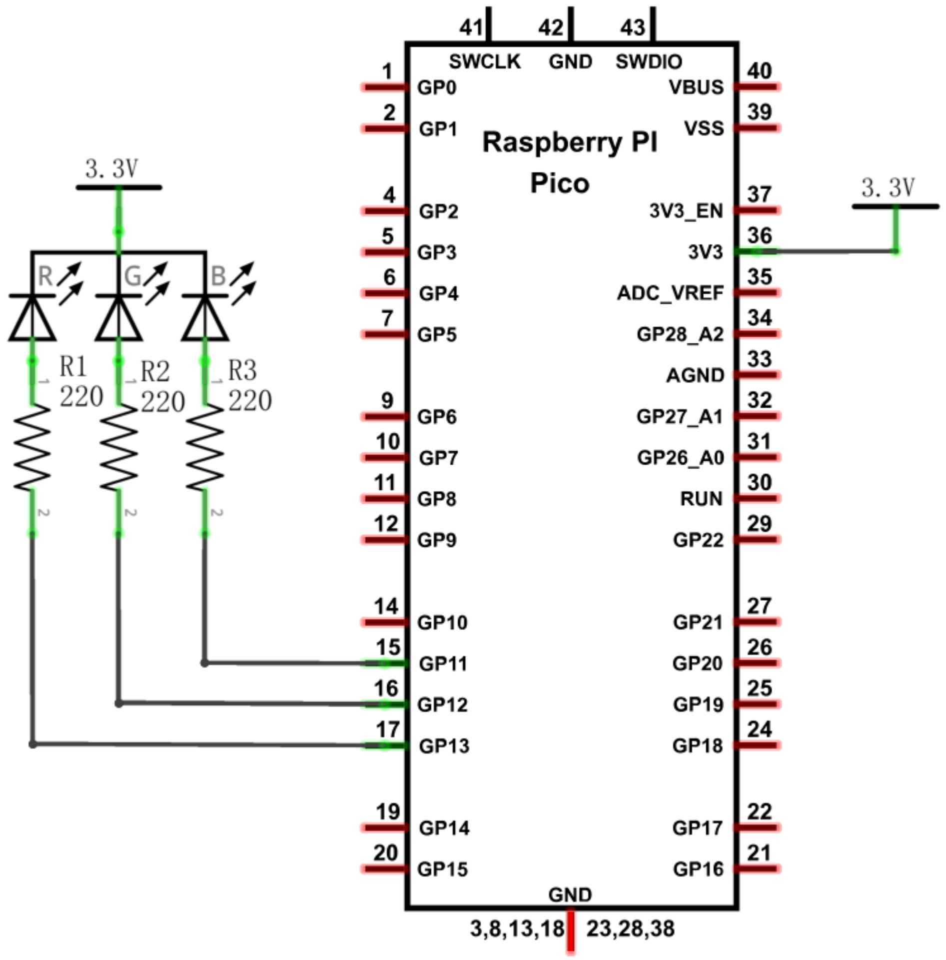


RGB

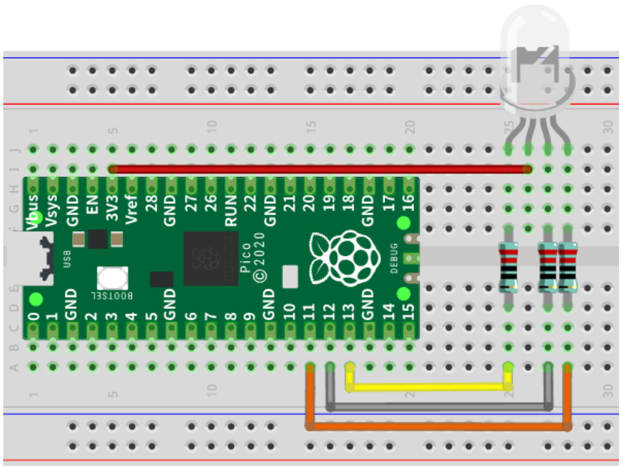
If we use three 10-bit PWM to control the RGBLED, in theory, we can create $2^{10}*2^{10}*2^{10}$= 1,073,741,824(1 billion) colors through different combinations.

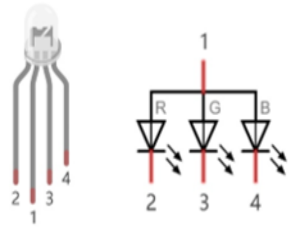
## Circuit

Schematic diagram




Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com





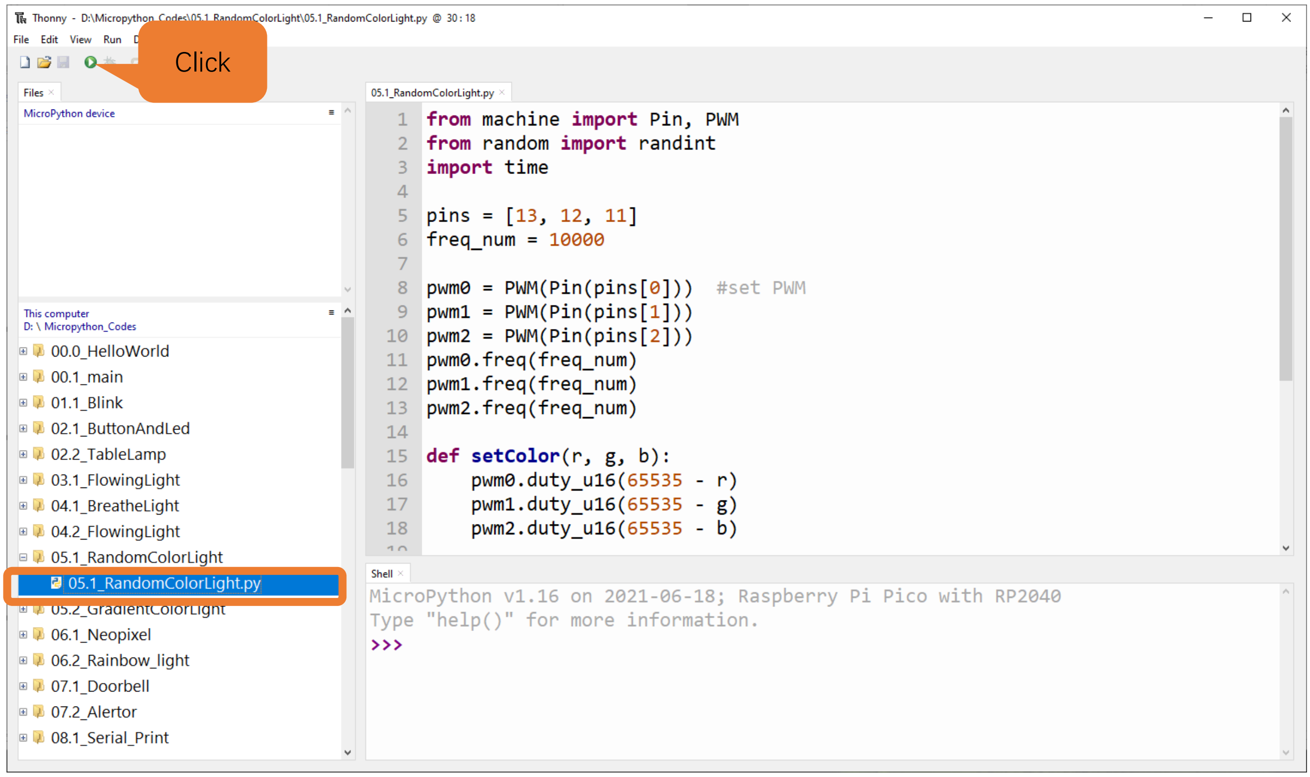
Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

# Code

We need to create three PWM channels and use random duty cycle to make random RGBLED color.

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “05.1_RandomColorLight”and double click “05.1_RandomColorLight.py”.

05.1_RandomColorLight




Click “Run current script”, RGBLED begins to display random colors. Press Ctrl+C or click “Stop/Restart backend” to exit program.

**If you have any concerns, please contact us via: support@freenove.com**

The following is the program code:

```
from machine import Pin, PWM
from random import randint
import time

pins = [13, 12, 11]
freq_num = 10000

pwm0 = PWM(Pin(pins[0]))  #set PWM
pwm1 = PWM(Pin(pins[1]))
pwm2 = PWM(Pin(pins[2]))
pwm0.freq(freq_num)
pwm1.freq(freq_num)
pwm2.freq(freq_num)

def setColor(r, g, b):
    pwm0.duty_u16(65535 - r)
    pwm1.duty_u16(65535 - g)
    pwm2.duty_u16(65535 - b)

try:
    while True:
        red   = randint(0, 65535)
        green = randint(0, 65535)
        blue  = randint(0, 65535)
        setColor(red, green, blue)
        time.sleep_ms(200)
except:
    pwm0.deinit()
    pwm1.deinit()
    pwm2.deinit()
```

Import Pin, PWM and Random function modules.

```
from machine import Pin, PWM
from random import randint
import time
```

Configure ouput mode of GP11, GP12 and GP13 as PWM output and PWM frequency as 10000Hz.

```
pins = [13, 12, 11]
freq_num = 10000

pwm0 = PWM(Pin(pins[0])) #set PWM
pwm1 = PWM(Pin(pins[1]))
pwm2 = PWM(Pin(pins[2]))
pwm0.freq(freq_num)
pwm1.freq(freq_num)
```

```
pwm2.freq(freq_num)
```

Define a function to set the color of RGBLED.

```
def setColor(r, g, b):
    pwm0.duty_u16(65535 - r)
    pwm1.duty_u16(65535 - g)
    pwm2.duty_u16(65535 - b)
```

Call random function randint() to generate a random number in the range of 0-65535 and assign the value to red.

```
    red = randint(0, 65535)
```

Obtain 3 random numbers every 200 milliseconds and call function setColor to make RGBLED display dazzling colors.

```
    while True:
        red   = randint(0, 65535)
        green = randint(0, 65535)
        blue  = randint(0, 65535)
        setColor(red, green, blue)
        time.sleep_ms(200)
```

Reference

**Class random**

Before each use of the module **random,** please add the statement "**import random**" to the top of Python file.

**randint(start, end)**: Randomly generates an integer between the value of start and end.

**start**: Starting value in the specified range, which would be included in the range.

**end**: Ending value in the specified range, which would be included in the range.

**random()**: Randomly generates a floating point number between 0 and 1.

**random.uniform(start, end)**: Randomly generates a floating point number between the value of start and end.

**start**: Starting value in the specified range, which would be included in the range.

**end**: Ending value in the specified range, which would be included in the range.

**random.getrandbits(size)**: Generates an integer with size random bits.

For example:

size = 4, it generates an integer in the range of 0 to 0b1110.

size = 8, it generates an integer in the range of 0 to 0b11111110.

**random.randrange(start, end, step)**: Randomly generates a positive integer in the range from start to end and increment to step.

**start**: Starting value in the specified range, which would be included in the range.

**end**: Ending value in the specified range, which would be included in the range.

**step**: An integer specifying the incrementation.

**random.seed(sed)**: Specifies a random seed, usually being applied in conjunction with other random number generators.

**sed**: Random seed, a starting point in generating random numbers.

**random.choice(obj)**: Randomly generates an element from the object obj.

**obj**: list of elements.

# Project 5.2 Gradient Color Light

In the previous project, we have mastered the usage of RGBLED, but the random color display is rather stiff. This project will realize a fashionable Light with soft color changes.

Component list, the circuit is exactly the same as the project random color light.

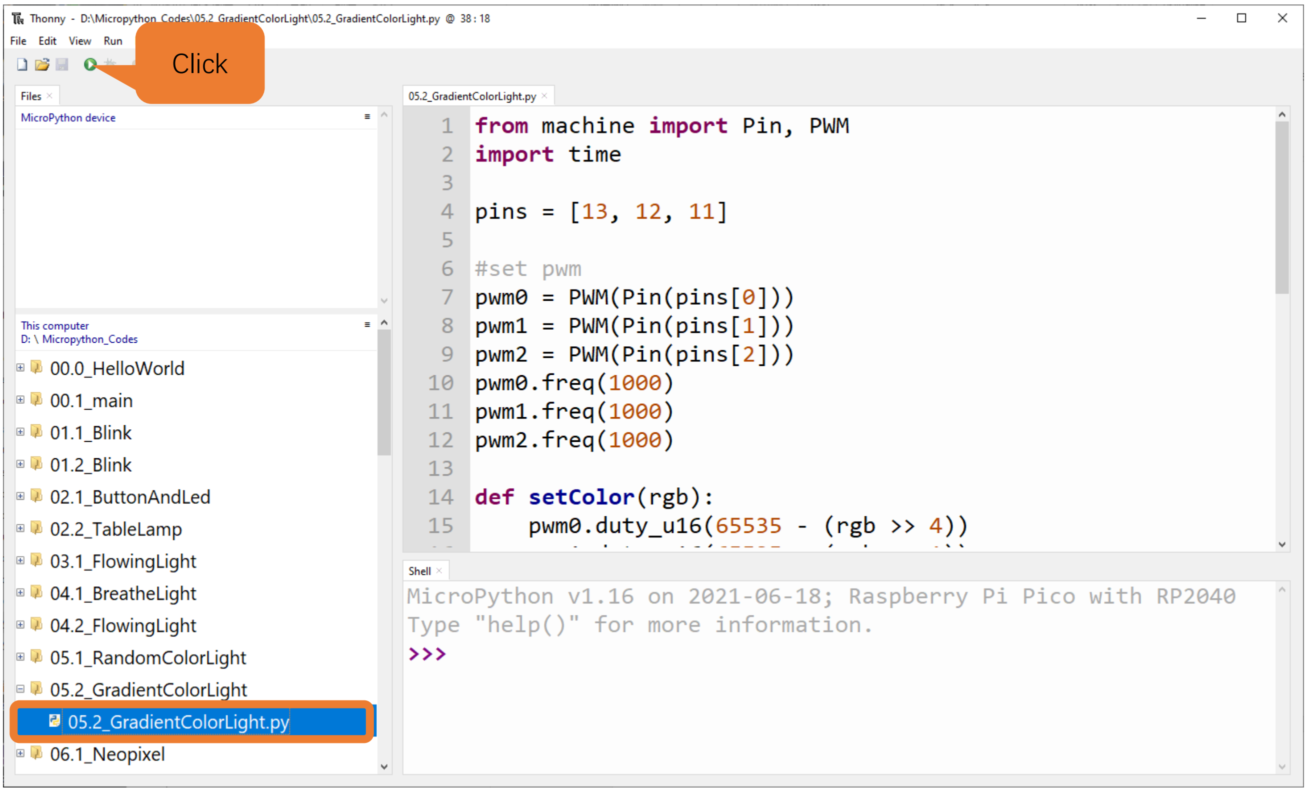
Using a color model, the color changes from 0 to 255 as shown below.


## Code

In this code, the color model will be implemented and RGBLED will change colors along the model.

Open “Thonny”, click“This computer” → “D:” → “Micropython_Codes” → “05.2_GradientColorLight” and double click “05.2_GradientColorLight.py”.

05.2_GradientColorLight




Click “Run current script”, and the light emited by RGBLED will change gradually. Press Ctrl+C or click “Stop/Restart backend” to exit program.

The following is the program code:

```
from machine import Pin, PWM
import time

pins = [13, 12, 11]

#set pwm
pwm0 = PWM(Pin(pins[0]))
pwm1 = PWM(Pin(pins[1]))
pwm2 = PWM(Pin(pins[2]))
pwm0.freq(1000)
pwm1.freq(1000)
pwm2.freq(1000)

def setColor(rgb):
    pwm0.duty_u16(65535 - (rgb >> 4))
    pwm1.duty_u16(65535 - (rgb >> 1))
    pwm2.duty_u16(65535 - (rgb >> 0))

def wheel(pos):
    WheelPos = pos % 65535
    if WheelPos < 21845:
        return (((65535 - WheelPos*3) << 4) | ((WheelPos*3) << 1))
    elif WheelPos >= 21845 and WheelPos < 43690:
        WheelPos -= 21845
        return (((65535 - WheelPos*3) << 1) | (WheelPos*3))
    else :
        WheelPos -= 43690
        return (((WheelPos*3) << 4) | (65535 - WheelPos*3))

try:
    while True:
        for i in range(0, 65535):
            setColor(wheel(i))
            time.sleep_ms(10)
except:
    pwm0.deinit()
    pwm1.deinit()
    pwm2.deinit()
```

In the function **setColor()**, we use a variable to represent the value of RGB, making it more convenient for the passing of parameters. As the range of PWM's duty cycle is 0-65535, which is 2 to the sixteen power, when spilt, the value of each color channel can be obtained with a simple bitwise operation.

```
def setColor(rgb):
    pwm0.duty_u16(65535 - (rgb >> 4))
    pwm1.duty_u16(65535 - (rgb >> 1))
    pwm2.duty_u16(65535 - (rgb >> 0))
```

The function wheel() is a color selection method of the color model introduced earlier. The value range of the parameter pos is 0-65535. The function will return a data containing the duty cycle values of 3 pins.

```
def wheel(pos):
    WheelPos = pos % 65535
    if WheelPos < 21845:
        return (((65535 - WheelPos*3) << 4) | ((WheelPos*3) << 1))
    elif WheelPos >= 21845 and WheelPos < 43690:
        WheelPos -= 21845
        return (((65535 - WheelPos*3) << 1) | (WheelPos*3))
    else :
        WheelPos -= 43690
        return (((WheelPos*3) << 4) | (65535 - WheelPos*3))
```

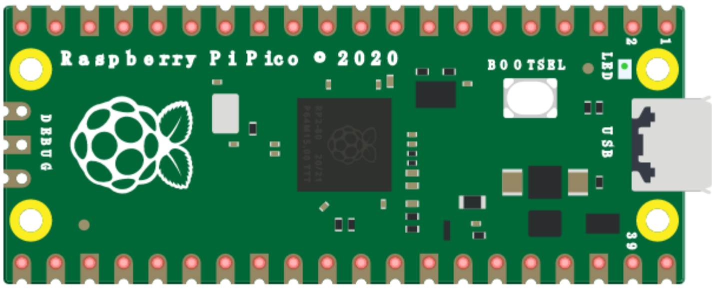
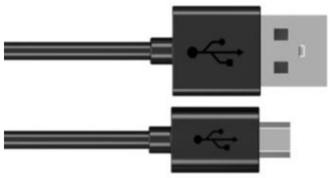
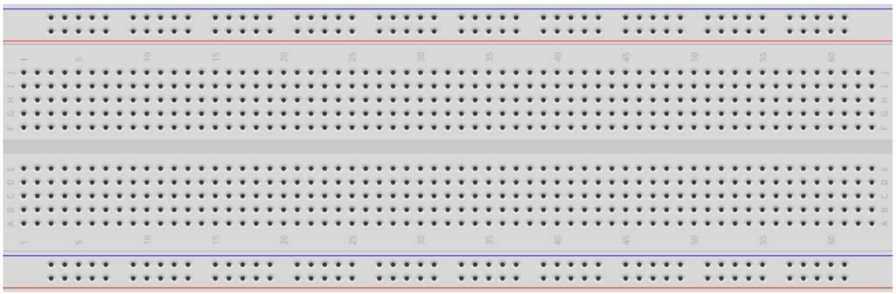
# Chapter 6 Buzzer

In this chapter, we will learn about buzzers and the sounds they make.

## Project 6.1 Doorbell

We will make this kind of doorbell: when the button is pressed, the buzzer sounds; and when the button is released, the buzzer stops sounding.

### Component List

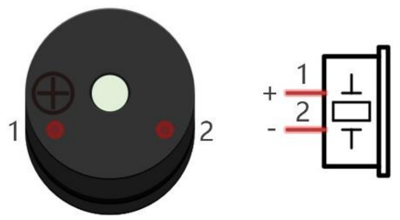
<table>
<tr><td colspan="2">Raspberry Pi Pico x1<br>
</td><td colspan="3">USB cable x1<br></td></tr>
<tr><td colspan="5">Breadboard x1<br></td></tr>
<tr><td colspan="5">Jumper<br></td></tr>
<tr><td>NPN transistorx1<br>(S8050)<br></td><td>Active buzzer x1<br></td><td>Push button x1<br></td><td>Resistor<br>1kΩ x1<br></td><td>Resistor<br>10kΩ x2<br></td></tr>
</table>

## Component knowledge

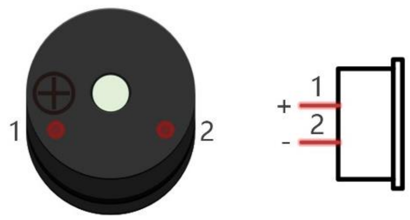
### Buzzer

Buzzer is a sounding component, which is widely used in electronic devices such as calculator, electronic warning clock and alarm. Buzzer has two types: active and passive. Active buzzer has oscillator inside, which will sound as long as it is supplied with power. Passive buzzer requires external oscillator signal (generally use PWM with different frequency) to make a sound.

Active buzzer




Passive buzzer




Active buzzer is easy to use. Generally, it can only make a specific frequency of sound. Passive buzzer requires an external circuit to make a sound, but it can be controlled to make a sound with different frequency. The resonant frequency of the passive buzzer is 2kHz, which means the passive buzzer is loudest when its resonant frequency is 2kHz.
Next, we will use an active buzzer to make a doorbell and a passive buzzer to make an alarm.

### How to identify active and passive buzzer?

1. Usually, there is a label on the surface of active buzzer covering the vocal hole, but this is not an absolute judgment method.
2. Active buzzers are more complex than passive buzzers in their manufacture. There are many circuits and crystal oscillator elements inside active buzzers; all of this is usually protected with a waterproof coating (and a housing) exposing only its pins from the underside. On the other hand, passive buzzers do not have protective coatings on their underside. From the pin holes viewing of a passive buzzer, you can see the circuit board, coils, and a permanent magnet (all or any combination of these components depending on the model.

Active buzzer

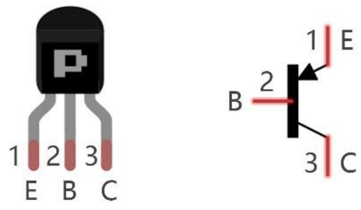
Passive buzzer

### Transistor

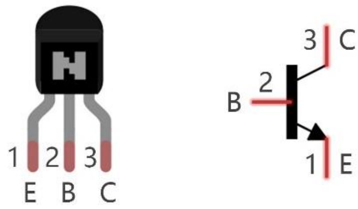
Because the buzzer requires such large current that GP of Raspberry Pi Pico output capability cannot meet the requirement, a transistor of NPN type is needed here to amplify the current.
Transistor, the full name: semiconductor transistor, is a semiconductor device that controls current. Transistor

can be used to amplify weak signal, or works as a switch. It has three electrodes(PINs): base (b), collector (c) and emitter (e). When there is current passing between "be", "ce" will allow several-fold current (transistor magnification) pass, at this point, transistor works in the amplifying area. When current between "be" exceeds a certain value, "ce" will not allow current to increase any longer, at this point, transistor works in the saturation area. Transistor has two types as shown below: PNP and NPN.

PNP transistor




NPN transistor

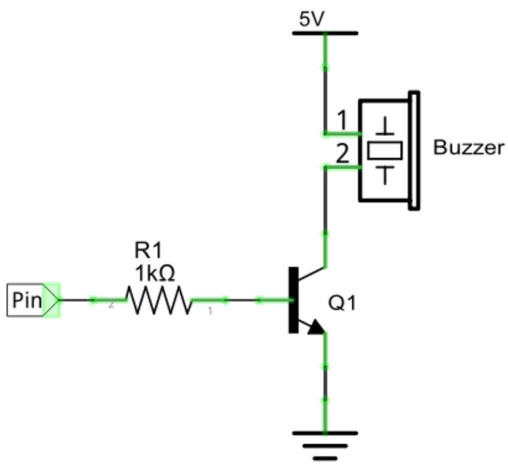
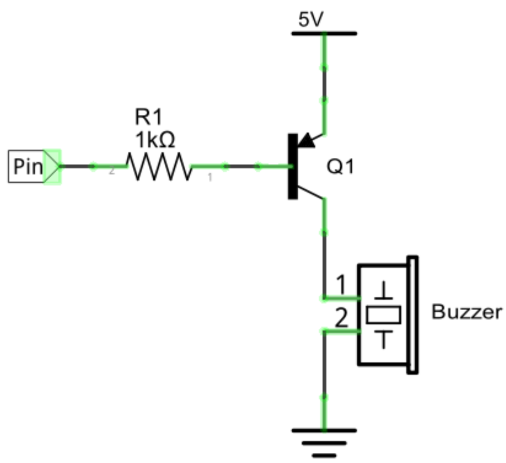



In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

Based on the transistor's characteristics, it is often used as a switch in digital circuits. As micro-controller's capacity to output current is very weak, we will use transistor to amplify current and drive large-current components.

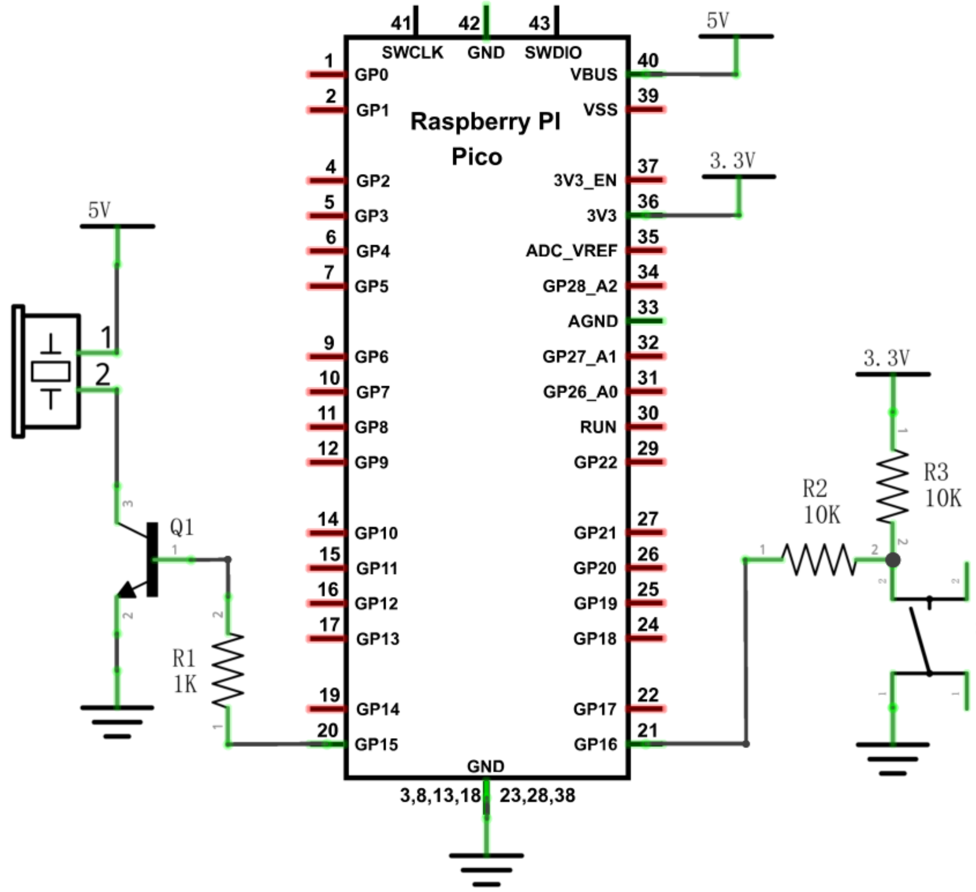
When using NPN transistor to drive buzzer, we often adopt the following method. If GP outputs high level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GP outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.

When using PNP transistor to drive buzzer, we often adopt the following method. If GP outputs low level, current will flow through R1, the transistor will get conducted, and the buzzer will sound. If GP outputs high level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.

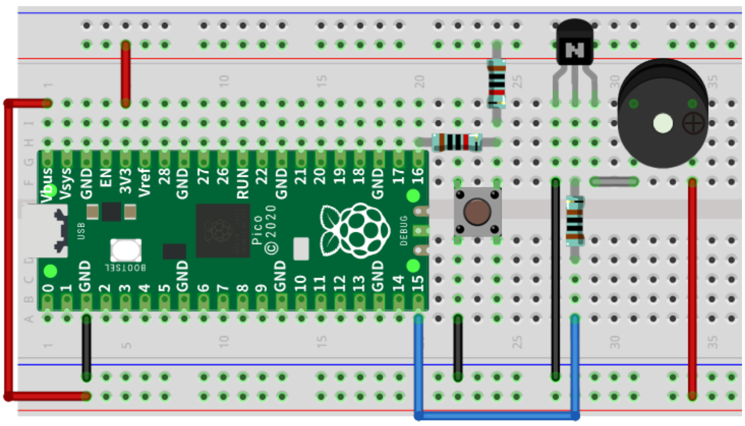
| NPN transistor to drive buzzer | PNP transistor to drive buzzer |
|---|---|
|   |   |

## Circuit

Schematic diagram




Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Note:

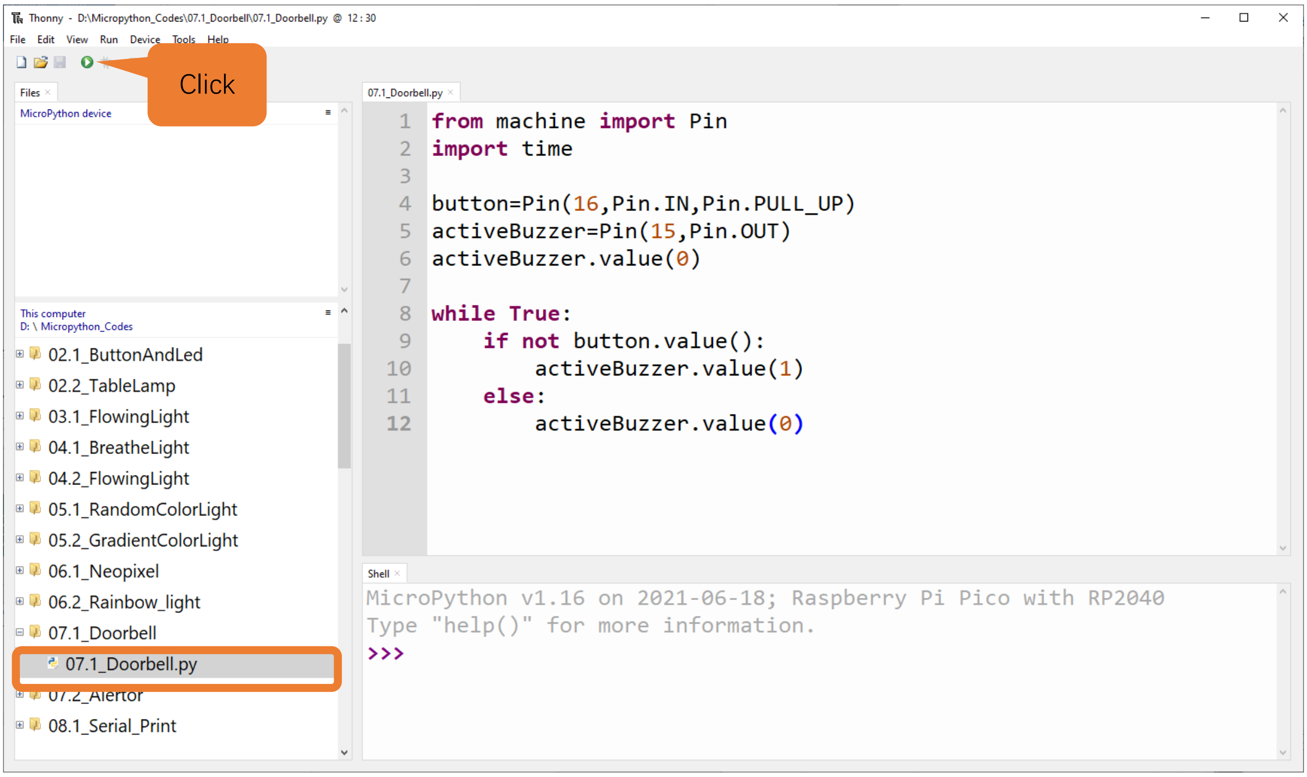
1. in this circuit, the power supply for buzzer is 5V, and pull-up resistor of the button connected to the power 3.3V. The buzzer can work when connected to power 3.3V, but it will reduce the loudness.
2. VBUS should be connect to the positive end of USB cable. If it connects to GND, it may burn the computer or Raspberry Pi Pico. Similarly, please be careful when wiring pins 36-40 of Pico to avoid short circuit.

# Code

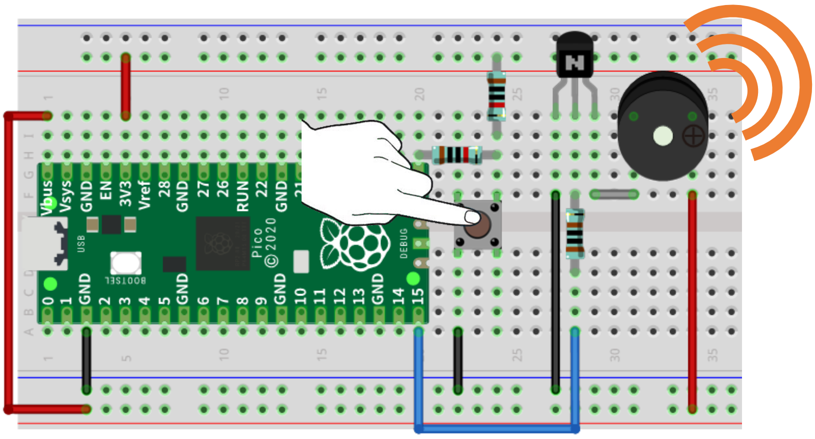
In this project, a buzzer will be controlled by a push button switch. When the button switch is pressed, the buzzer sounds and when the button is released, the buzzer stops. It is analogous to our earlier project that controlled an LED ON and OFF.

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "06.1_Doorbell" and double click "06.1_Doorbell.py".

06.1_Doorbell



Click “Run current script”, press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.Press Ctrl+C or click “Stop/Restart backend” to exit program.




The following is the program code:

```
from machine import Pin
import time

button=Pin(16,Pin.IN,Pin.PULL_UP)
activeBuzzer=Pin(15,Pin.OUT)
activeBuzzer.value(0)

while True:
    if not button.value():
        activeBuzzer.value(1)
    else:
        activeBuzzer.value(0)
```

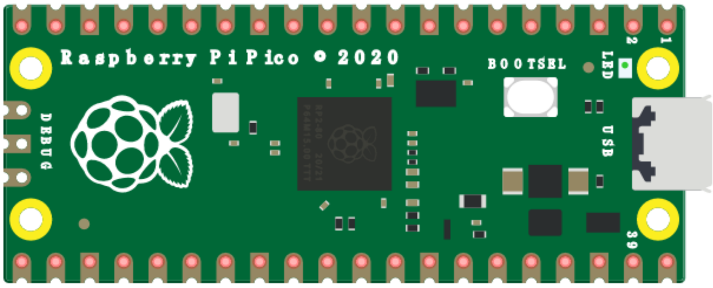
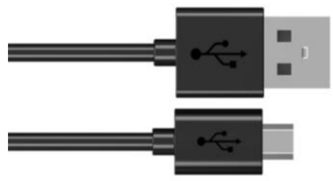
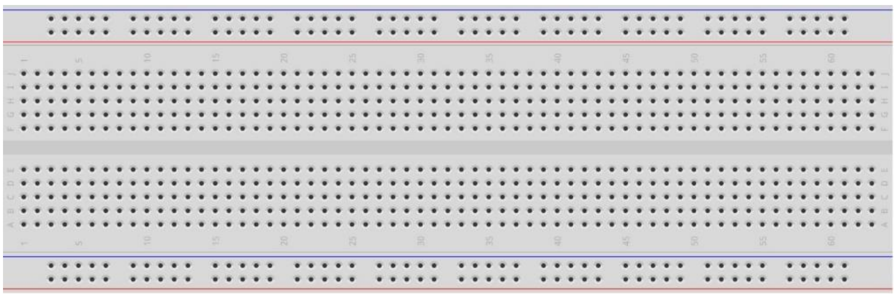
The code is logically the same as using button to control LED.

# Project 6.2 Alertor

Next, we will use a passive buzzer to make an alarm.

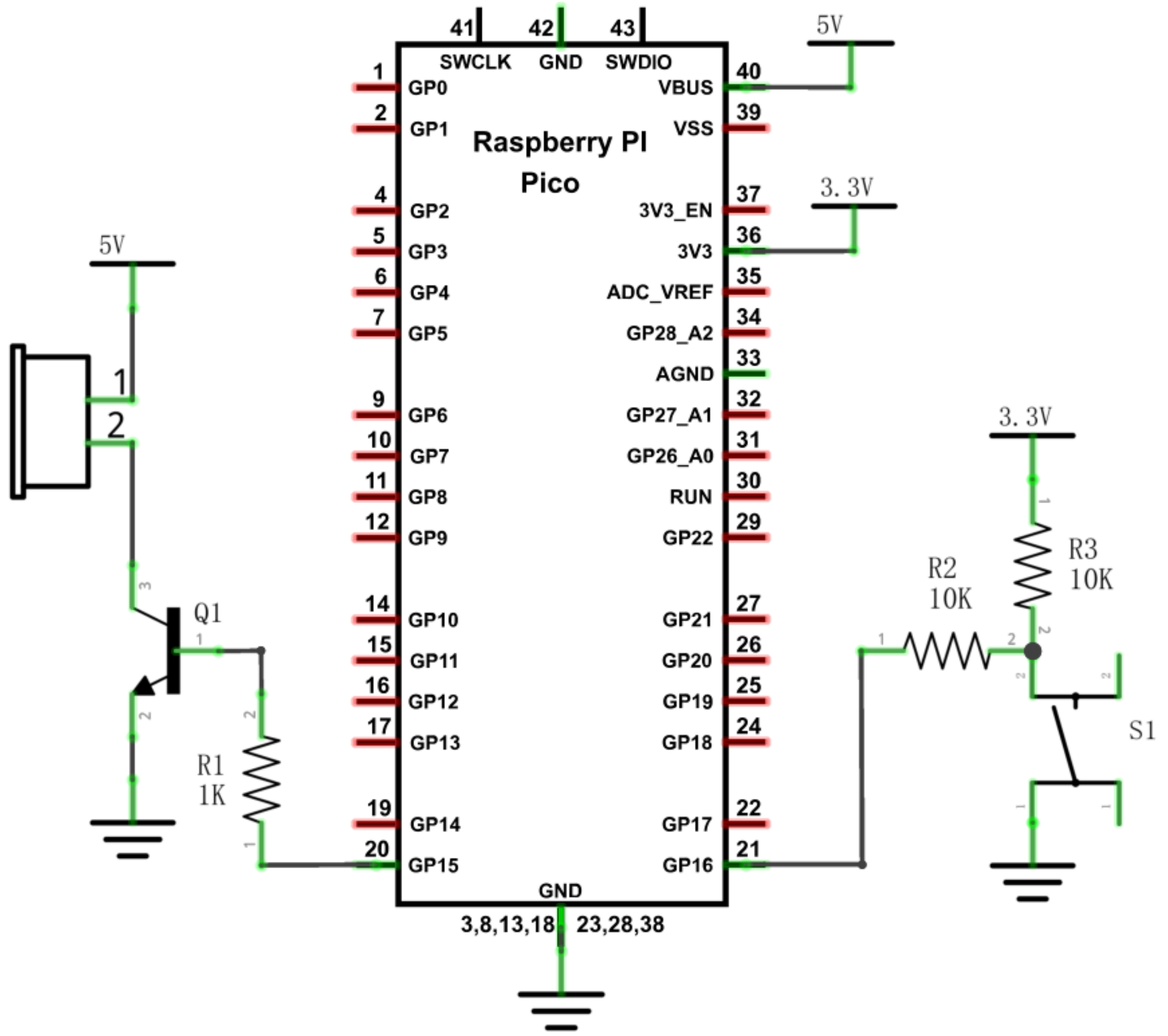
Component list and the circuit part is similar to last section. In the Doorbell circuit only the **active buzzer** needs to be **replaced** with a **passive buzzer.**

## Component List

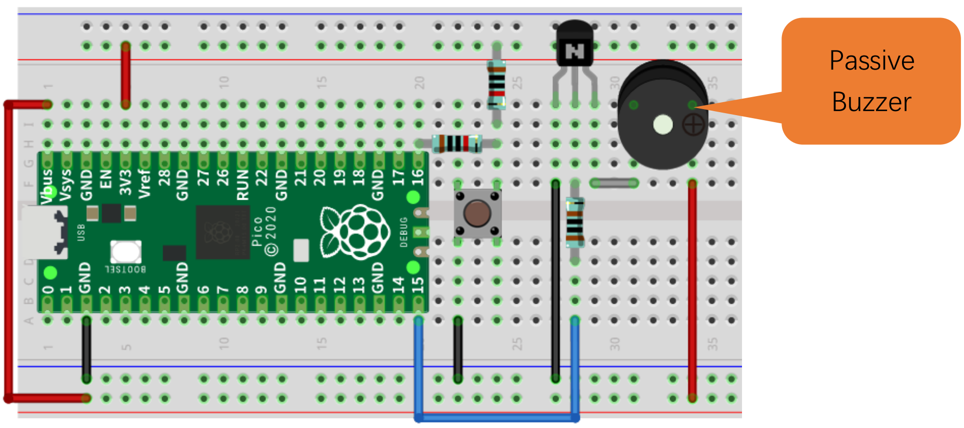
<table>
<tr><td colspan="3">Raspberry Pi Pico x1<br>
</td><td colspan="2">USB cable x1<br></td></tr>
<tr><td colspan="5">Breadboard x1<br></td></tr>
<tr><td colspan="5">Jumper<br></td></tr>
<tr><td>NPN transistorx1<br>(S8050)<br></td><td>Passive buzzer x1<br></td><td>Push button x1<br></td><td>Resistor<br>1kΩ x1<br></td><td>Resistor<br>10kΩ x2<br></td></tr>
</table>

## Circuit

Schematic diagram




Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

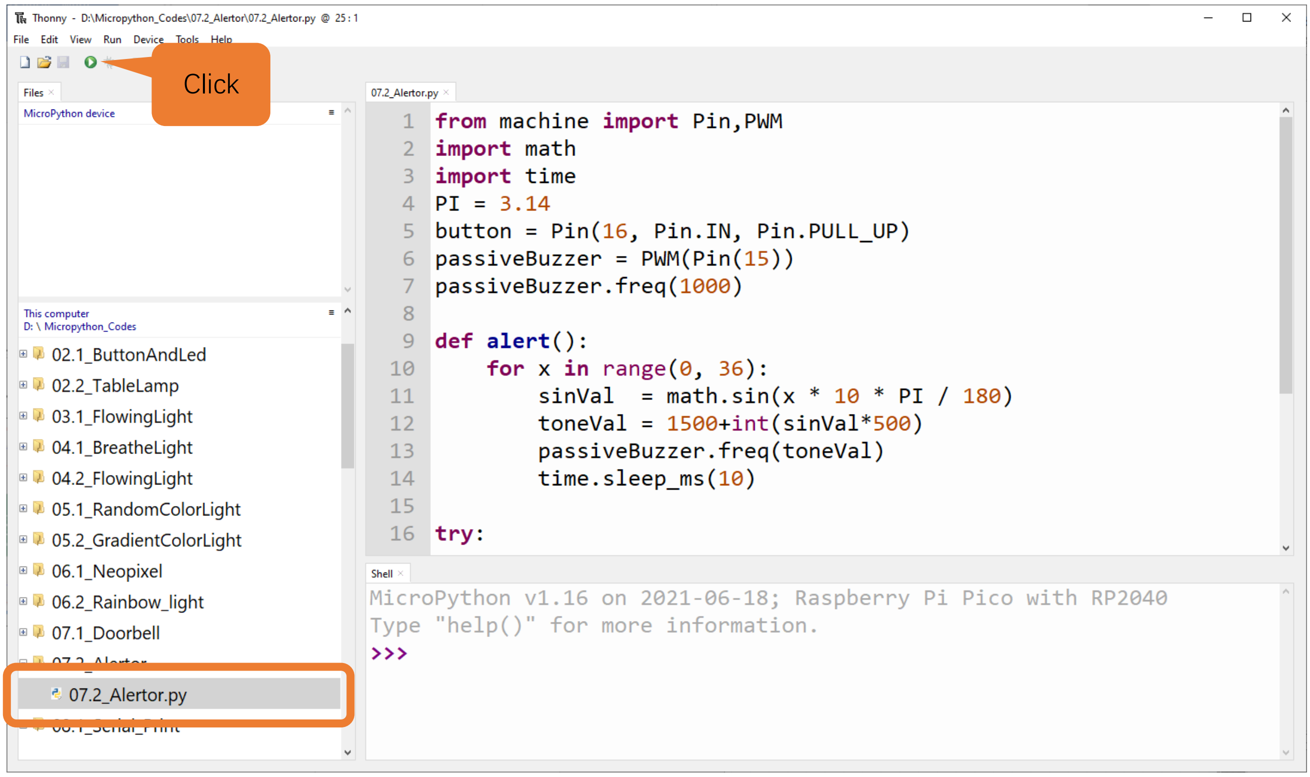


# Code

In this project, the buzzer alarm is controlled by the button. Press the button, then buzzer sounds. If you release the button, the buzzer will stop sounding. Logically, it is the same as using button to control LED. As to the control method, passive buzzer requires PWM of certain frequency to sound.

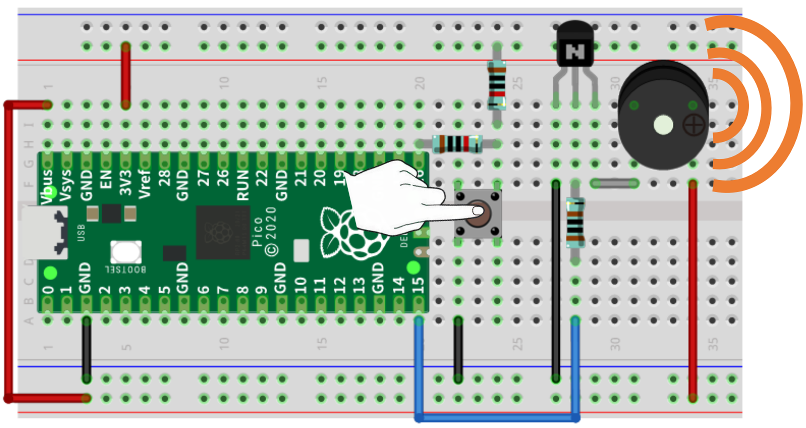
Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “06.2_Alertor”, and double click “06.2_Alertor.py”.

06.2_Alertor




Click “Run current script”, press the button,and the alarm sounds. And when the button is released, the alarm will stop sounding. Press Ctrl+C or click “Stop/Restart backend” to exit program.

If the buzzer sound is too loud or too low for you, you can modify duty cycle or frequency of PWM.



The following is the program code:

```
from machine import Pin, PWM
import math
import time
PI = 3.14
button = Pin(16, Pin.IN, Pin.PULL_UP)
passiveBuzzer = PWM(Pin(15))
passiveBuzzer.freq(1000)

def alert():
    for x in range(0, 36):
        sinVal  = math.sin(x * 10 * PI / 180)
        toneVal = 1500 + int(sinVal*500)
        passiveBuzzer.freq(toneVal)
        time.sleep_ms(10)

try:
    while True:
        if not button.value():
            passiveBuzzer.duty_u16(4092*2)
            alert()
        else:
            passiveBuzzer.duty_u16(0)
except:
    passiveBuzzer.deinit()
```

Import PWM, Pin, math and time modules.

```
from machine import Pin, PWM
import math
import time
```

Define the pins of the button and passive buzzer.

```
PI = 3.14
button = Pin(16, Pin.IN, Pin.PULL_UP)
passiveBuzzer = PWM(Pin(15))
passiveBuzzer.freq(1000)
```

Call sin function of math module to create the frequency data of the passive buzzer.

```
def alert():
    for x in range(0, 36):
        sinVal  = math.sin(x * 10 * PI / 180)
        toneVal = 1500 + int(sinVal*500)
        passiveBuzzer.freq(toneVal)
        time.sleep_ms(10)
```

When not using PWM, please turn it OFF in time.

```
    passiveBuzzer.deinit()
```

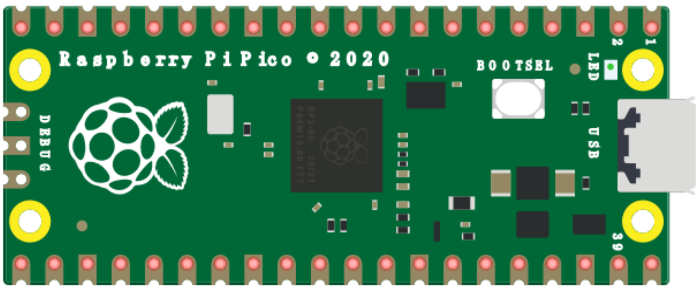
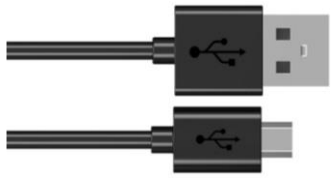
# Chapter 7 Serial Communication

Serial Communication is a means of Communication between different devices. This section describes Raspberry Pi Pico Serial Communication.

## Project 7.1 Serial Print

This project uses Raspberry Pi Pico serial communicator to send data to the computer and print it on the serial monitor.

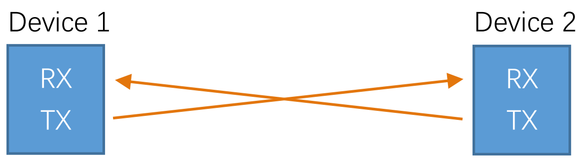
### Component List

| Raspberry Pi Pico x1 | USB cable x1 |
|---|---|
|  |  |

# Related knowledge

## Serial communication

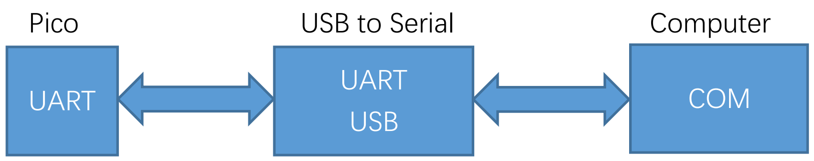
Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines, one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication connections two devices use is as follows:




Before serial communication starts, the baud rate of both sides must be the same. Communication between devices can work only if the same baud rate is used. The baud rates commonly used is 9600 and 115200.

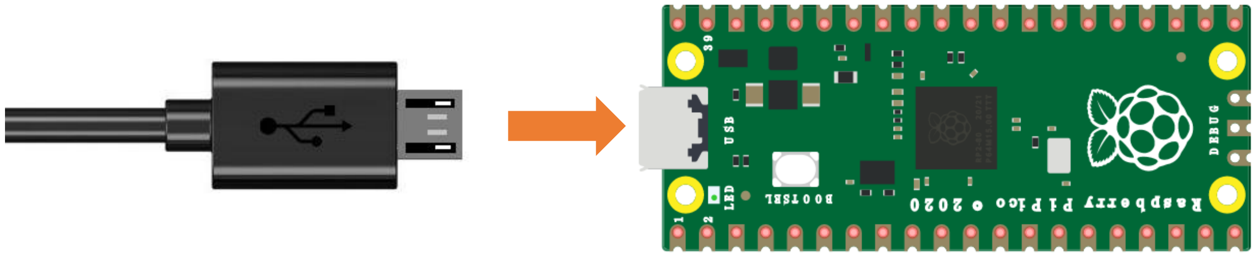
## Serial port on Raspberry Pi Pico

Raspberry Pi Pico has integrated USB to serial transfer, so it could communicate with computer connecting to USB cable.




# Circuit

Connect Raspberry Pi Pico to the computer with USB cable.

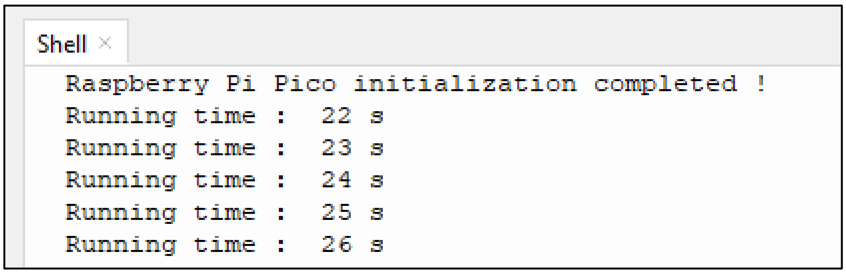


## Code

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "07.1_Serial_Print" and double "07.1_Serial_Print.py".

07.1_Serial_Print


Click "Run current script" and observe the changes of "Shell", which will display the time when Raspberry Pi Pico is powered on once per second.



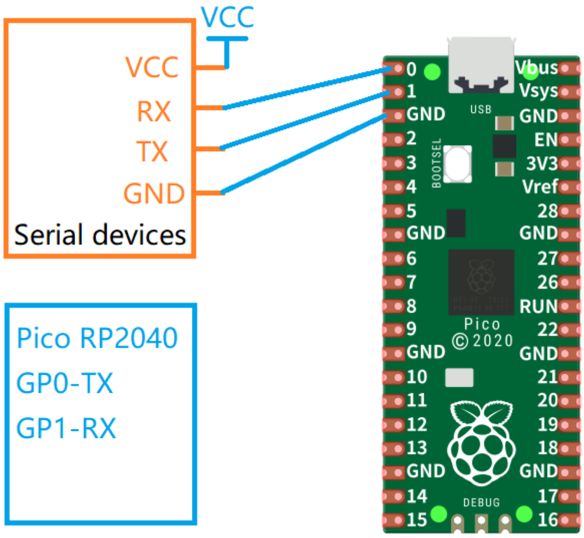

The following is the program code:

```
import time

print("Raspberry Pi Pico initialization completed !")

while True:
    print("Running time : ", time.time()%60,"s")
    time.sleep(1)
```

There are two serial communications on Raspberry Pi Pico: UART0 and UART1.
You can use them to communicate with serial devices.



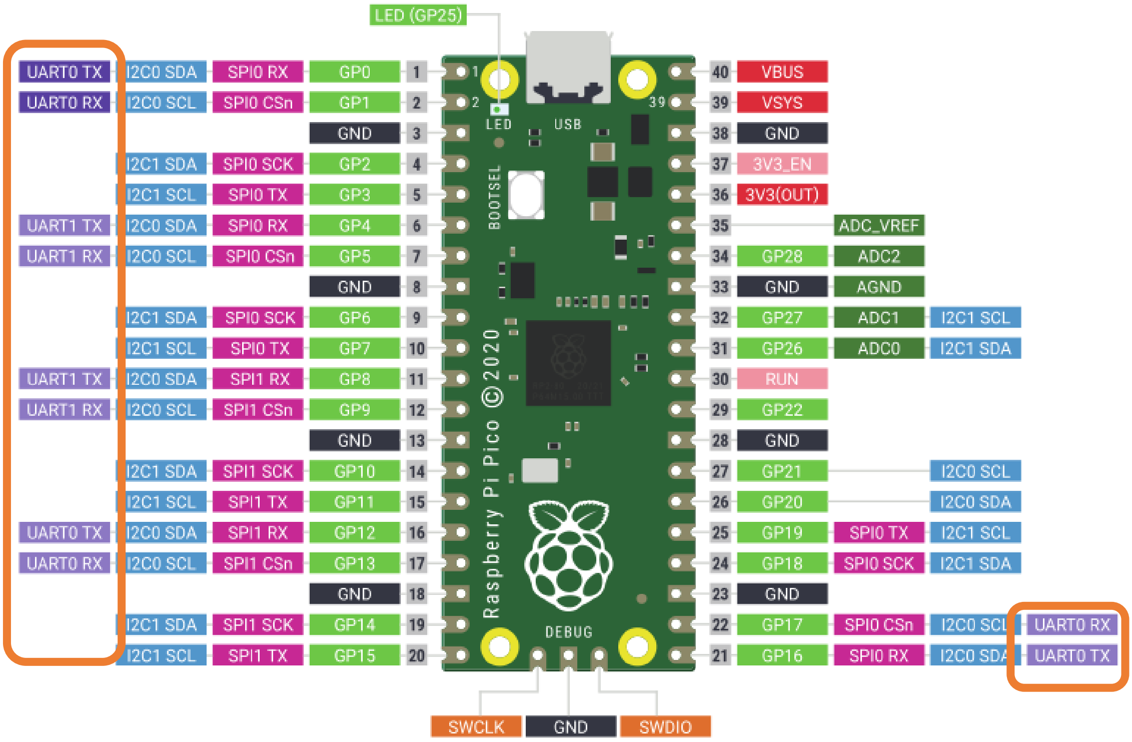

Default pin for UART0

| | |
|---|---|
| UART0_TX | Pin 0 |
| UART0_RX | Pin 1 |

Default pin for UART1

| | |
|---|---|
| UART1_TX | Pin 4 |
| UART1_RX | Pin 5 |

For details, please refer to UART, I2C, SPI default pin.
And you can also change settings according to the distribution of pins.



Reference

**Class UART**

Before each use of **UART** module, please add the statement "**from machine import UART**" to the top of python file.

**UART(id, baudrate, bits, parity, rx, tx, stop, timeout)**: Define serial ports and configure parameters for them.

**id**: Serial Number. The available serial port number is 0 or 1.

**baudrate:** Baud rate.

**bits**: The number of each character.

**parity**: Check even or odd, with 0 for even checking and 1 for odd checking.

**rx, tx**: UAPT's reading and writing pins, GP0, GP1, GP4, GP5, GP8, GP9, GP12, GP13, GP16, GP17.

**stop**: The number of stop bits, and the stop bit is 1 or 2.

**timeout**: timeout period (Unit: millisecond).

0 < timeout ⩽ 0x7FFF FFFF (decimal: 0 < timeout ⩽ 2147483647).

**UART.read(nbytes)**: Read nbytes bytes.

**UART.read()**: Read data.

**UART.write(buf)**: Write byte buffer to UART bus.

**UART.readline()**: Read a line of data, ending with a newline character.

**UART.readinto(buf)**: Read and write data into buffer.

**UART.readinto(buf, nbytes)**: Read and write data into buffer.

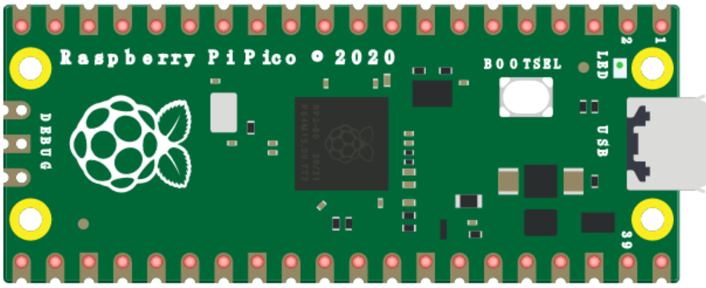
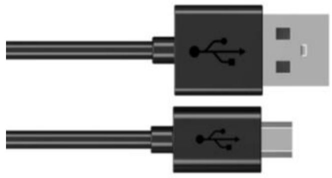
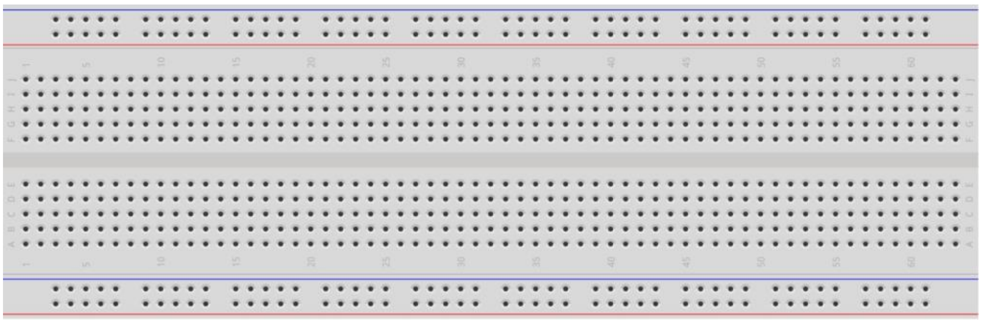
**UART.any()**: Determine whether there is data in serial ports. If there is, return the number of bytes; Otherwise, return 0.

Note: When usART0 or USART1 is used, it must be used with the serial port adapter board or serial port device. If not, you may not observe any symptoms.

# Project 7.2 Serial Read and Write

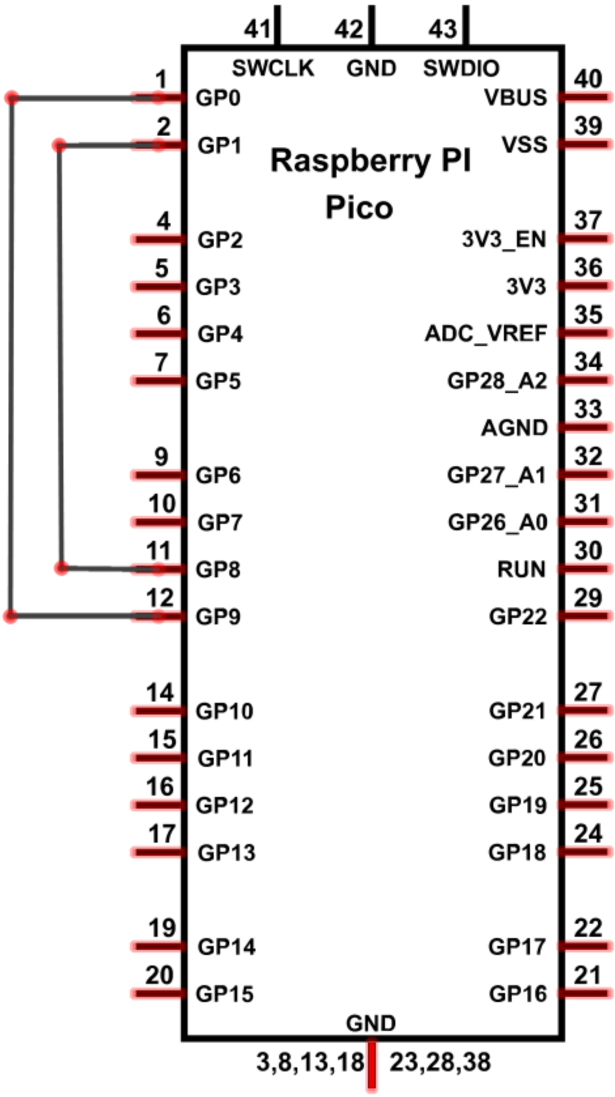
In the following example, we use Raspberry Pi Pico's UART1 to send data to UART0, and read the data received by UART0 and print it out through the "Shell".

## Component List

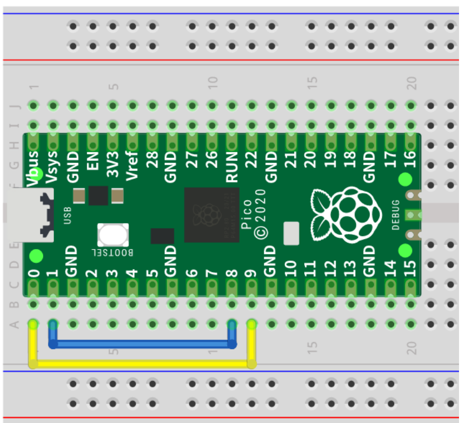
| Raspberry Pi Pico x1 | USB cable x1 |
|---|---|
|   |  |
| Breadboard x1 | |
|  | |
| Jumper | |
|  | |

## Circuit

Schematic diagram



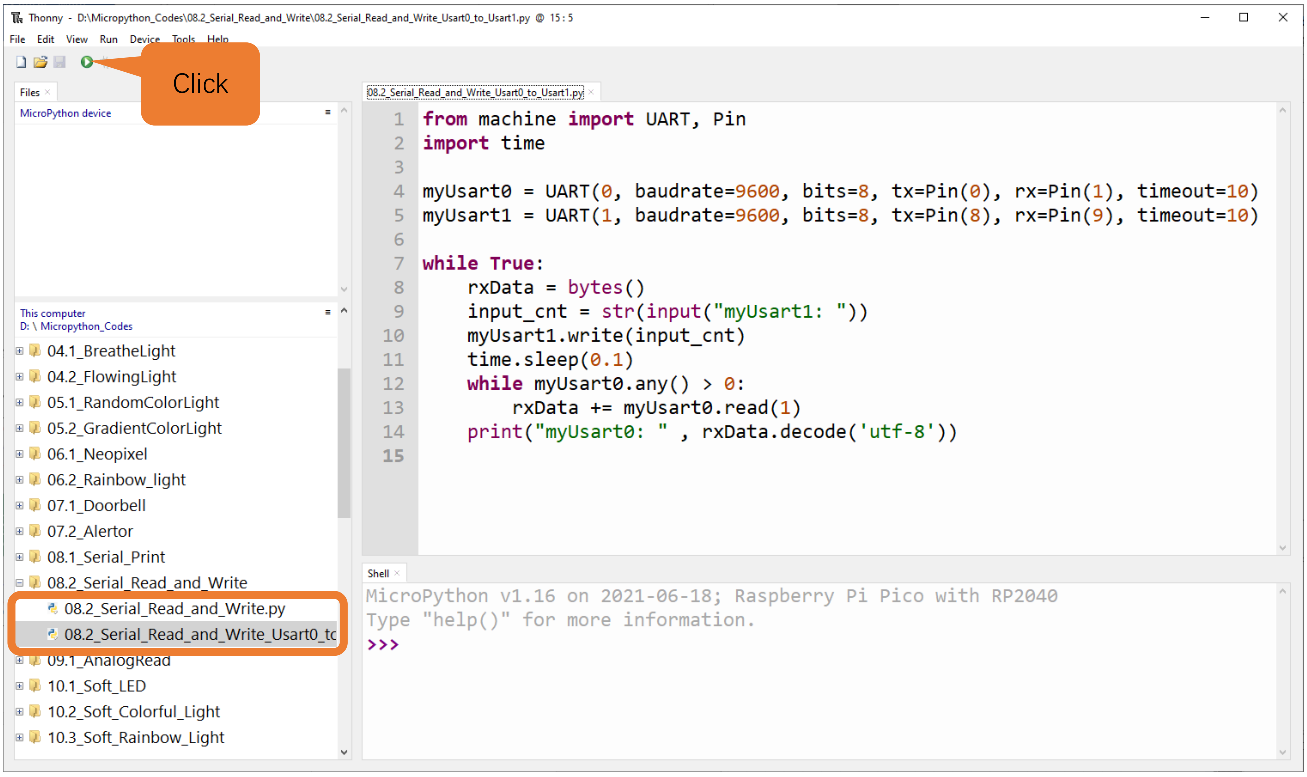

Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



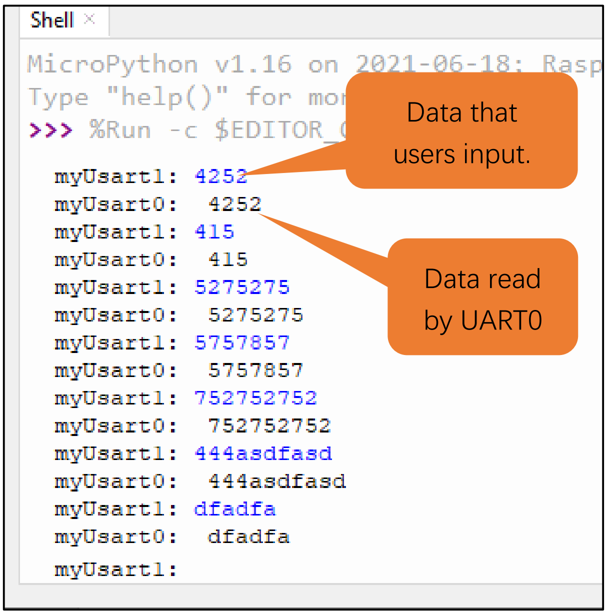
## Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “07.2_Serial_Read_and_Write” and double click “07.2_Serial_Read_and_Write_UART1_to_UART0.py”.

07.2_Serial_Read_and_Write_UART1_to_UART0




Click “Run current script”. Users can enter any data in “Shell” and press Enter. The input data will be written to UART1 and sent to UART0 to read, and printed out on “Shell”. Press “Ctrl+C” or click “Stop/Restart backend” to exit the program.



The following is the program code:

```
from machine import UART, Pin
import time

myUsart0 = UART(0, baudrate=9600, bits=8, tx=Pin(0), rx=Pin(1), timeout=10)
myUsart1 = UART(1, baudrate=9600, bits=8, tx=Pin(8), rx=Pin(9), timeout=10)

while True:
    rxData = bytes()
    input_cnt = str(input("myUsart1: "))
    myUsart1.write(input_cnt)
    time.sleep(0.1)
    while myUsart0.any() > 0:
        rxData += myUsart0.read(1)
    print("myUsart0: " , rxData.decode('utf-8'))
```

Import UART、Pin and time modules.

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

Create two UART objects and configure them as the parameters of UART0 and UART1.

```
myUsart0 = UART(0, baudrate=9600, bits=8, tx=Pin(0), rx=Pin(1), timeout=10)
myUsart1 = UART(1, baudrate=9600, bits=8, tx=Pin(8), rx=Pin(9), timeout=10)
```

Define a bytes value and assign it to rxDate.

```
    rxData = bytes()
```

Define input_cnt to receive user input and convert it to a string format.

```
    input_cnt = str(input("myUsart1: "))
```

myUsart1 calls write() function and writes the user input to UART1.

```
    myUsart1.write(input_cnt)
```

myUsart0 calls the read() function to read the data sent by UART1 bit by bit and save the rxData in the received variable. When myUsart0 calls any() to determine whether UART0 has read the data, when any() returns 0, UART0 has read the data sent by UART1.

```
    while myUsart0.any() > 0:
        rxData += myUsart0.read(1)
```

The decode() function is called to decode the data and print it out to "Shell."

```
    print("myUsart0: " , rxData.decode('utf-8'))
```

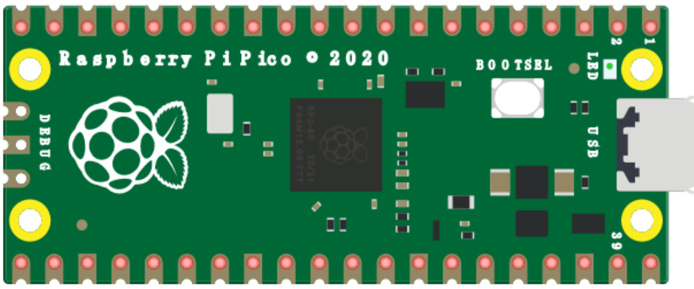
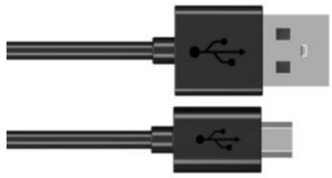
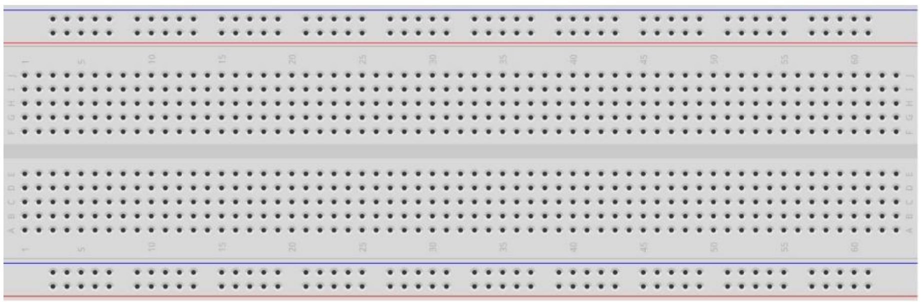
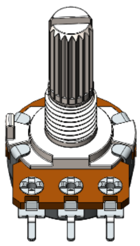
# Chapter 8 AD Converter

This chapter we learn to use the ADC function of Rasepberry Pi Pico.

## Project 8.1 Read the Voltage of Potentiometer

In this chapter, we use ADC function of Pico to read the voltage output by potentiometer.

### Component List

| Raspberry Pi Pico x1 | USB cable x1 |
| --- | --- |
|   |  |
| Breadboard x1 | |
|  | |
| Rotary potentiometer x1 | Jumper |
|  |  |

## Related knowledge

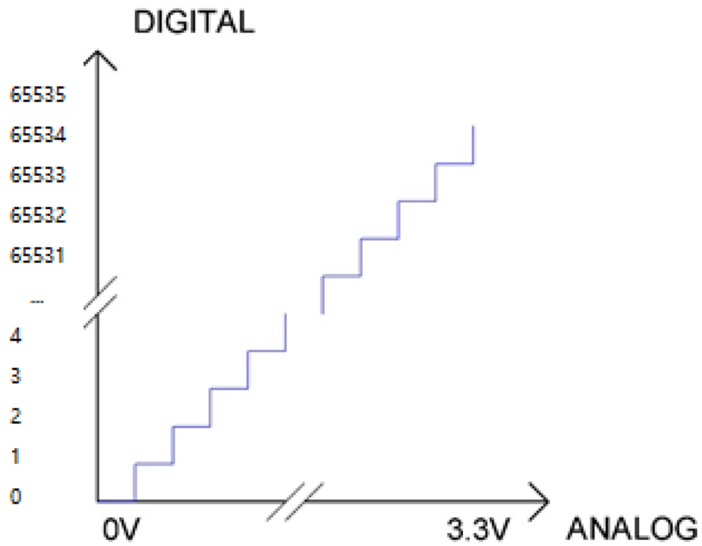
### ADC

An analog-to-digital converter(ADC) converts a measured analog signal into a digital code. ADC has two key features: resolution and channels.

### ADC resolution of Raspberry Pi Pico

Raspberry Pi Pico uses RP2040 chip. With a 12-bit ADC resolution, it can convert any analog signal to digital signal, ranging from 0-4095. For example, if the analog voltage range it measures is 0-3.3V, the ADC can divide it into 4096 equal parts.

However, when using Micropython firmware to call Raspberry Pi Pico ADC, the digital signal it obtains ranges from 0 to 65535. This is because Micropython internally processes Pico's ADC resolution to 16 bits, and the values is also changed to 0-65535, so as to make it the same as the ADC of other Micropython microcontrollers. The way that ADC converts doesn't change but only the resolution changes. So if the measured analog voltage ranges from 0-3.3V, ADC can devide it into 65536 equal parts.




Subsection 1: the analog in a range of 0V---3.3/65535 V corresponds to digital 0;

Subsection 2: the analog in a range of 3.3/65535 V---2*3.3 /65535 V corresponds to digital 1;

...

Subsection 65535: the analog in range of 65534*3.3/65535 V---65535*3.3 /65535 V corresponds to digital 65534;

The following analog will be divided accordingly.

The conversion formula is as follows:

$$ADCValue = \frac{\text{Analog Voltage}}{3.3} * 65535$$

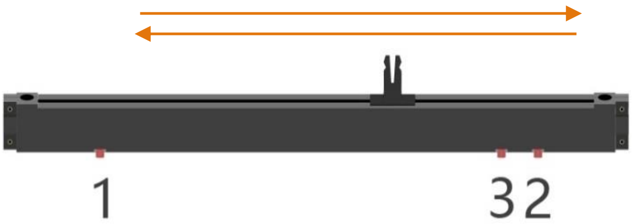
### ADC ChannelsRaspberry Pi Pico

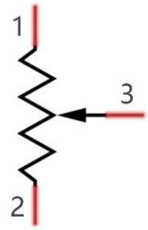
Raspberry Pi Pico has 5 ADC channels, which are ADC0(GP26), ADC1(GP27), ADC2(GP28), ADC3(GP29): used to measure VSYS on Pico board, and ADC4, which directly connects to the built-in temperature sensor of RP2040 chip. Therefore, there are only three generic ADC channels that can be directly used, namely, ADC0, ADC1 and ADC2.

## Component knowledge

### Potentiometer

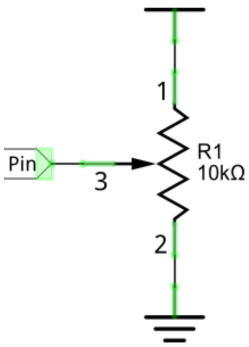
Potentiometer is a resistive element with three Terminal parts. Unlike the resistors that we have used thus far in our project which have a fixed resistance value, the resistance value of a potentiometer can be adjusted. A potentiometer is often made up by a resistive substance (a wire or carbon element) and movable contact brush. When the brush moves along the resistor element, there will be a change in the resistance of the potentiometer's output side (3) (or change in the voltage of the circuit that is a part). The illustration below represents a linear sliding potentiometer and its electronic symbol on the right.



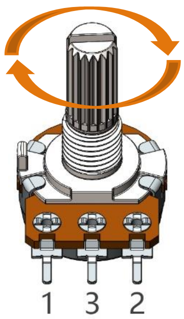

Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly (until it reaches the highest value of the resistive element) and at the same time the resistance between pin 2 and pin 3 will decrease linearly and conversely down to zero. At the midpoint of the slider the measured resistance values between pin 1 and 3 and between pin 2 and 3 will be the same.

In a circuit, both sides of resistive element are often connected to the positive and negative electrodes of power. When you slide the brush "pin 3", you can get variable voltage within the range of the power supply.

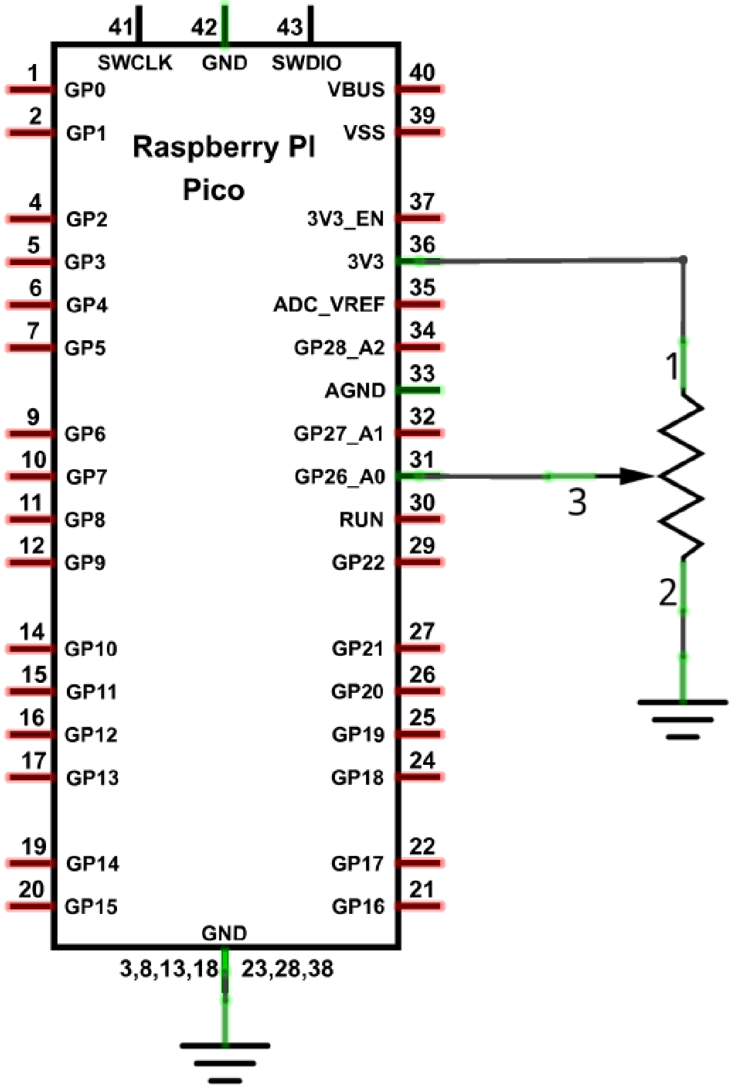
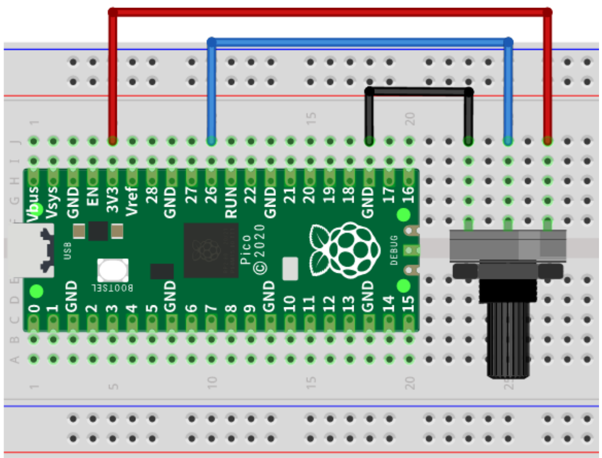



### Rotary potentiometer

Rotary potentiometers and linear potentiometers have the same function; the only difference being the physical action being a rotational rather than a sliding movement.



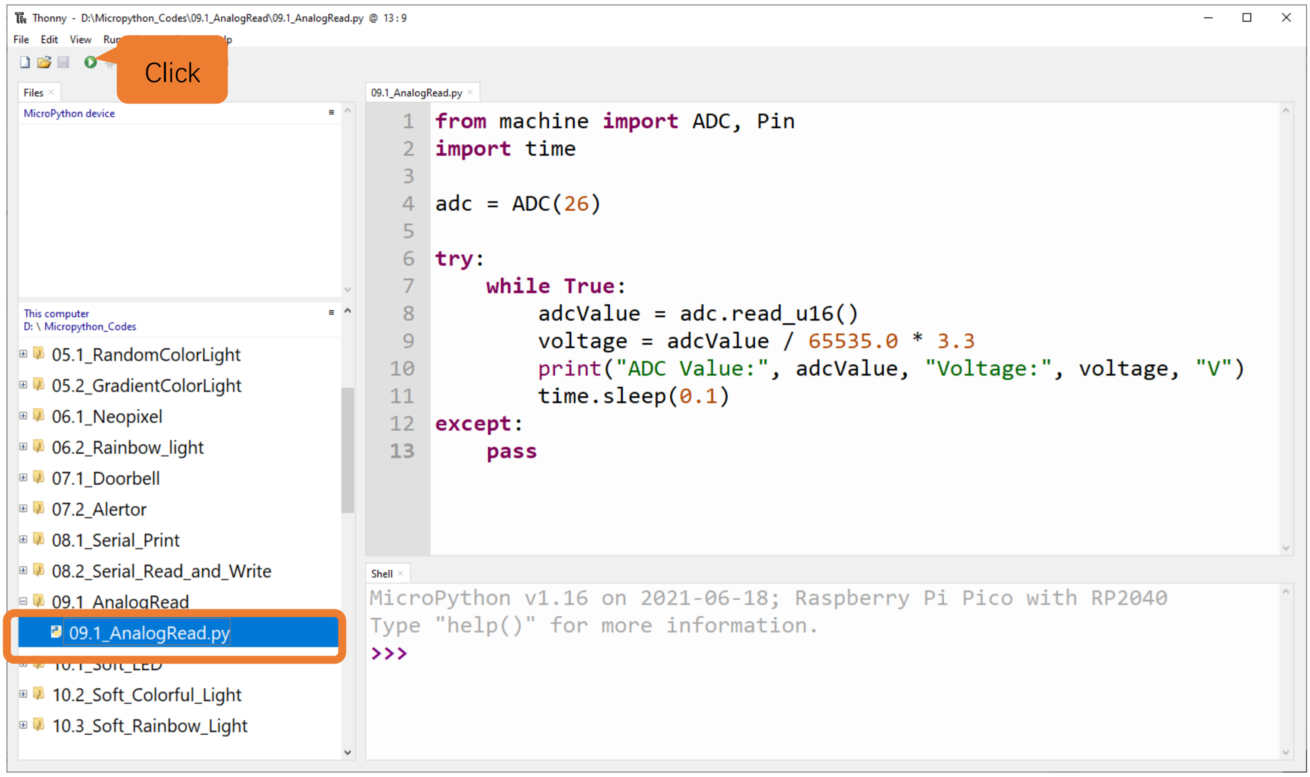
## Circuit

| |
|---|
| Schematic diagram<br> |
| Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com<br> |

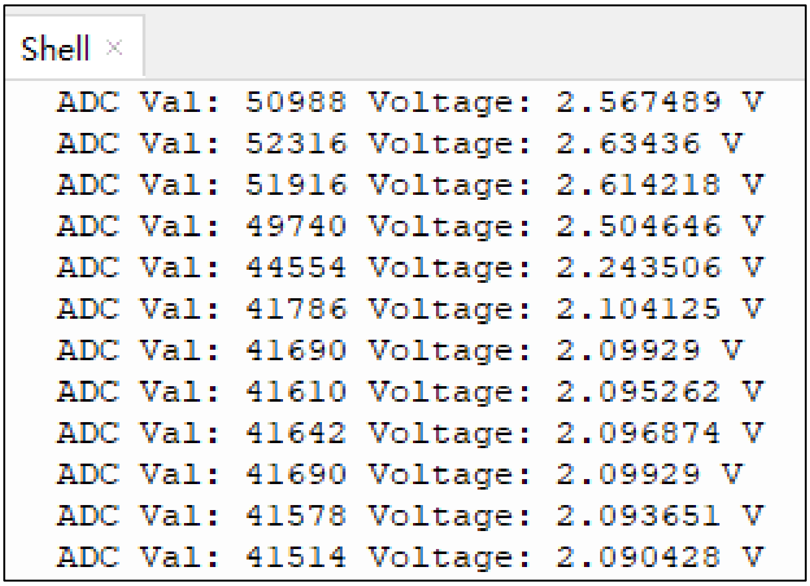
# Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “08.1_AnalogRead and then click “08.1_AnalogRead.py”.

08.1_AnalogRead




Click “Run current script” and observe the message printed in “Shell”. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



The following is the code:

```
from machine import ADC, Pin
import time

adc = ADC(26)

try:
    while True:
        adcValue = adc.read_u16()
        voltage = adcValue / 65535.0 * 3.3
        print("ADC Value:", adcValue, "Voltage:", voltage, "V")
        time.sleep(0.1)
except:
    pass
```

Import Pin, ADC and time modules.

```
from machine import ADC, Pin
import time
```

Create an ADC object and connect GP26, which corresponds to ADC0 channel of Raspberry Pi Pico.

```
adc = ADC(26)
```

Read ADC value every 0.1 second. Calculate the current voltage based on the formula ADCValue= (Analog Voltage)/3.3*65535 and print it to "Shell".

```
    while True:
        adcValue = adc.read_u16()
        voltage = adcValue / 65535.0 * 3.3
        print("ADC Value:", adcValue, "Voltage:", voltage, "V")
        time.sleep(0.1)
```

Reference

**Class ADC**

Before each use of ADC module, please add the statement "**from machine import ADC**" to the top of the python file.

**machine.ADC(pin or channel_num)**: Create an ADC object associated with the given pin.

**pin**: Available pins are:GP26,GP27,GP28,GP29.

**channel_num**: Available channel 0, 1, 2, 3, 4.

For example:

machine.ADC(0) = machine.ADC(26)

machine.ADC(1) = machine.ADC(27)

machine.ADC(2) = machine.ADC(28)

machine.ADC(3) = machine.ADC(29)

machine.ADC(4) Connects to the internal temperature sensor.

**ADC.read_16()**: reads the current ADC value and returns it, with a range of 0-65535.

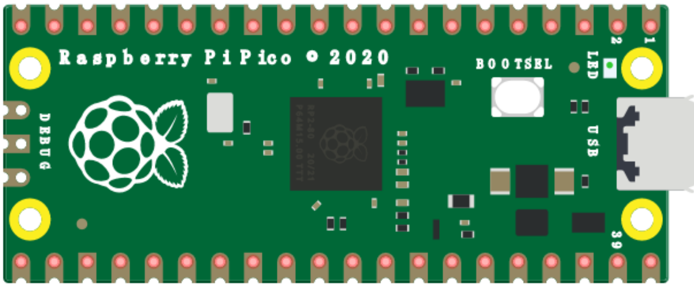
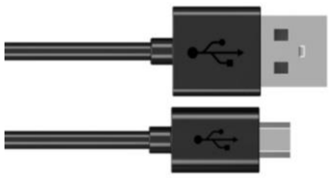
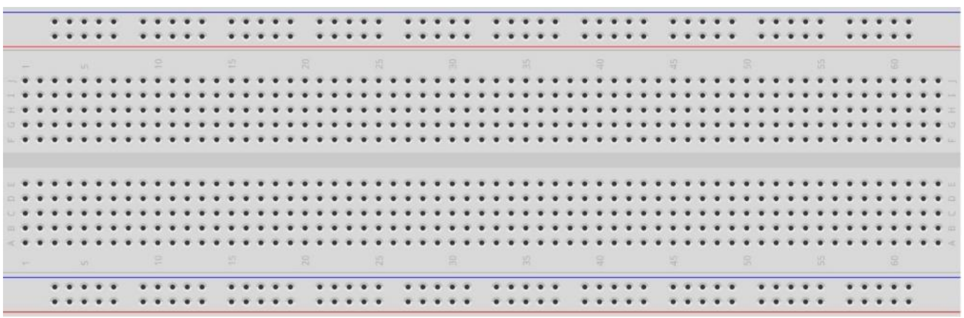
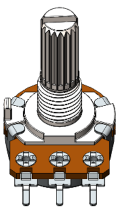
# Chapter 9 Potentiometer & LED

We have learnt to use ADC in the previous chapter. In this chapter, we will combine PWM and ADC to use potentiometer to control LED, RGBLED and Neopixel.

## Project 9.1 Soft Light

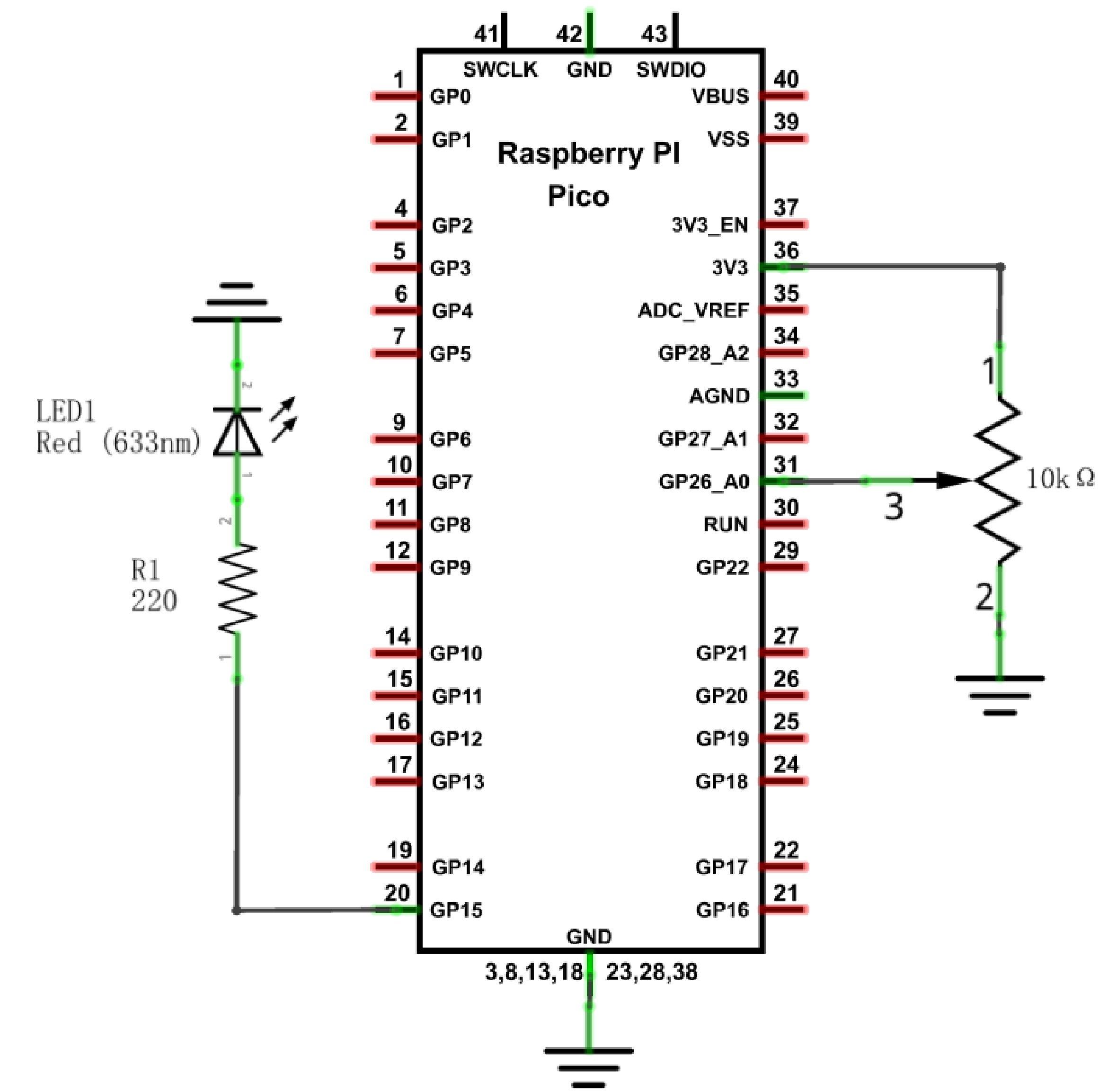
In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle of the PWM used to control the brightness of an LED. Then you can change the brightness of an LED by adjusting the potentiometer.

### Component List

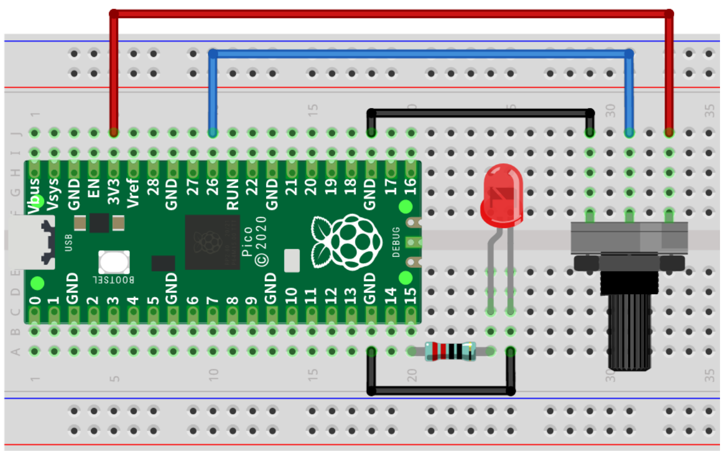
<table>
<tr><td colspan="3">Raspberry Pi Pico x1<br><br></td><td>USB cable x1<br></td></tr>
<tr><td colspan="4">Breadboard x1<br></td></tr>
<tr><td>Rotary potentiometer x1<br></td><td>Resistor 220Ω x1<br></td><td>LED x1<br></td><td>Jumper<br></td></tr>
</table>

## Circuit

Schematic diagram




Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

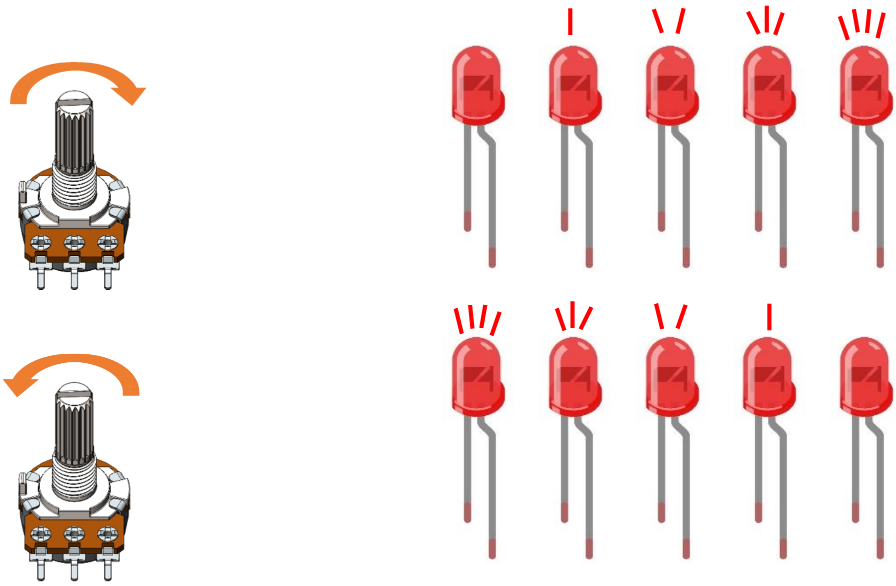


## Code

Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “09.1_Soft_LED” and double click “09.1_Soft_LED.py”.

09.1_Soft_LED


Click “Run current script”. Rotate the handle of potentiometer and the brightness of LED will change correspondingly. Press Ctrl+C or click “Stop/Restart backend” to exit the program.



**If you have any concerns, please contact us via: support@freenove.com**

The following is the code:

```
from machine import ADC, Pin, PWM
import time
adc = ADC(26)
pwm = PWM(Pin(15))
pwm.freq(1000)
try:
    while True:
        adcValue = adc.read_u16()
        pwm.duty_u16(adcValue)
        time.sleep(0.1)
except:
    pwm.deinit()
```

In the code, read the ADC value of potentiometer and map it to the duty cycle of PWM to control LED brightness.

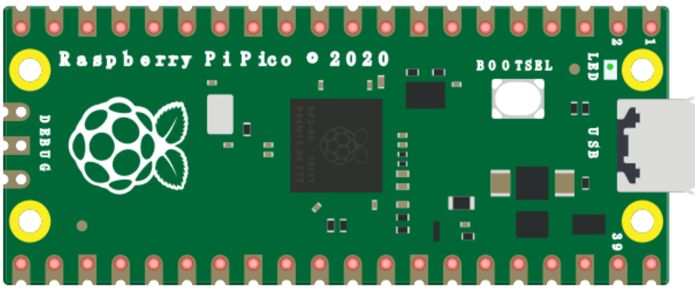
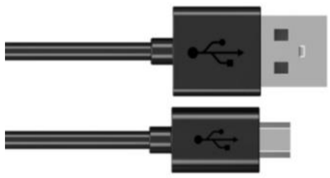
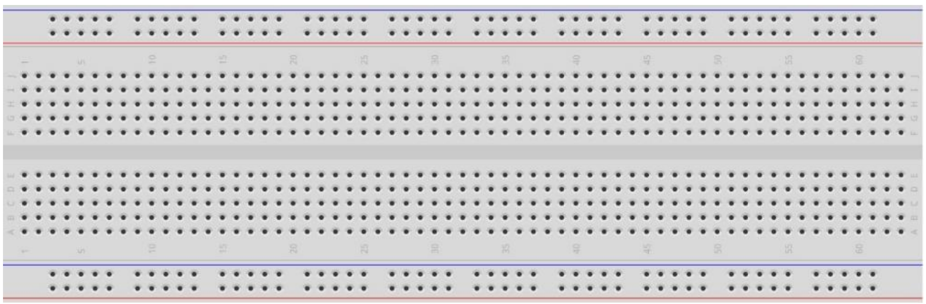
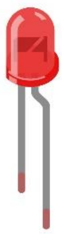
# Chapter 10 Photoresistor & LED

In this chapter, we will learn how to use photoresistor.

## Project 10.1 Control LED through Photoresistor

A Photoresistor is very sensitive to the amount of light present. We can take advantage of the characteristic to make a night lamp with the following function: when the ambient light is less (darker environment) the LED will automatically become brighter to compensate and when the ambient light is greater (brighter environment) the LED will automatically dim to compensate.

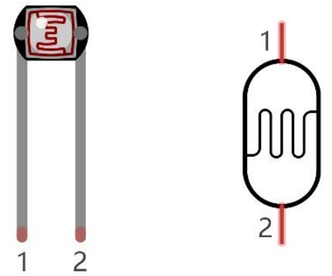
### Component List

<table>
<tr><td colspan="4">Raspberry Pi Pico x1<br>
</td><td colspan="2">USB cable x1<br></td></tr>
<tr><td colspan="6">Breadboard x1<br></td></tr>
<tr><td rowspan="2">Photoresistor x1<br></td><td colspan="2">Resistor</td><td rowspan="2">LED x1<br></td><td colspan="2" rowspan="2">Jumper<br></td></tr>
<tr><td>220Ω x1<br></td><td>10KΩ x1<br></td></tr>
</table>

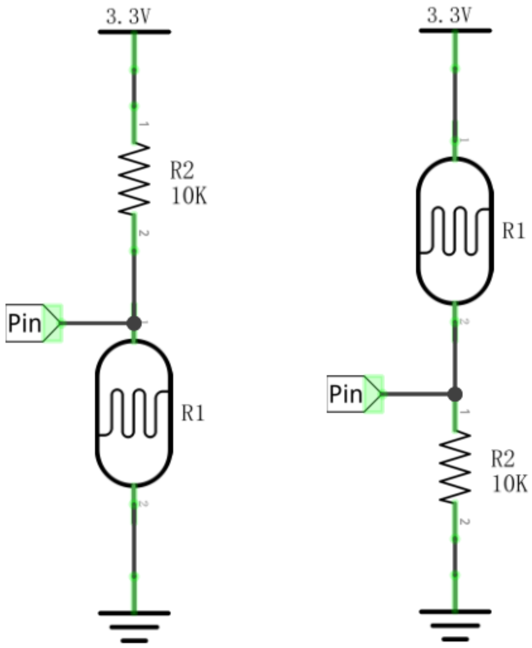
## Component knowledge

### Photoresistor

Photoresistor is simply a light sensitive resistor. It is an active component that decreases resistance with respect to receiving luminosity (light) on the component's light sensitive surface. Photoresistor's resistance value will change in proportion to the ambient light detected. With this characteristic, we can use a Photoresistor to detect light intensity. The Photoresistor and its electronic symbol are as follows.




The circuit below is used to detect the change of a Photoresistor's resistance value:

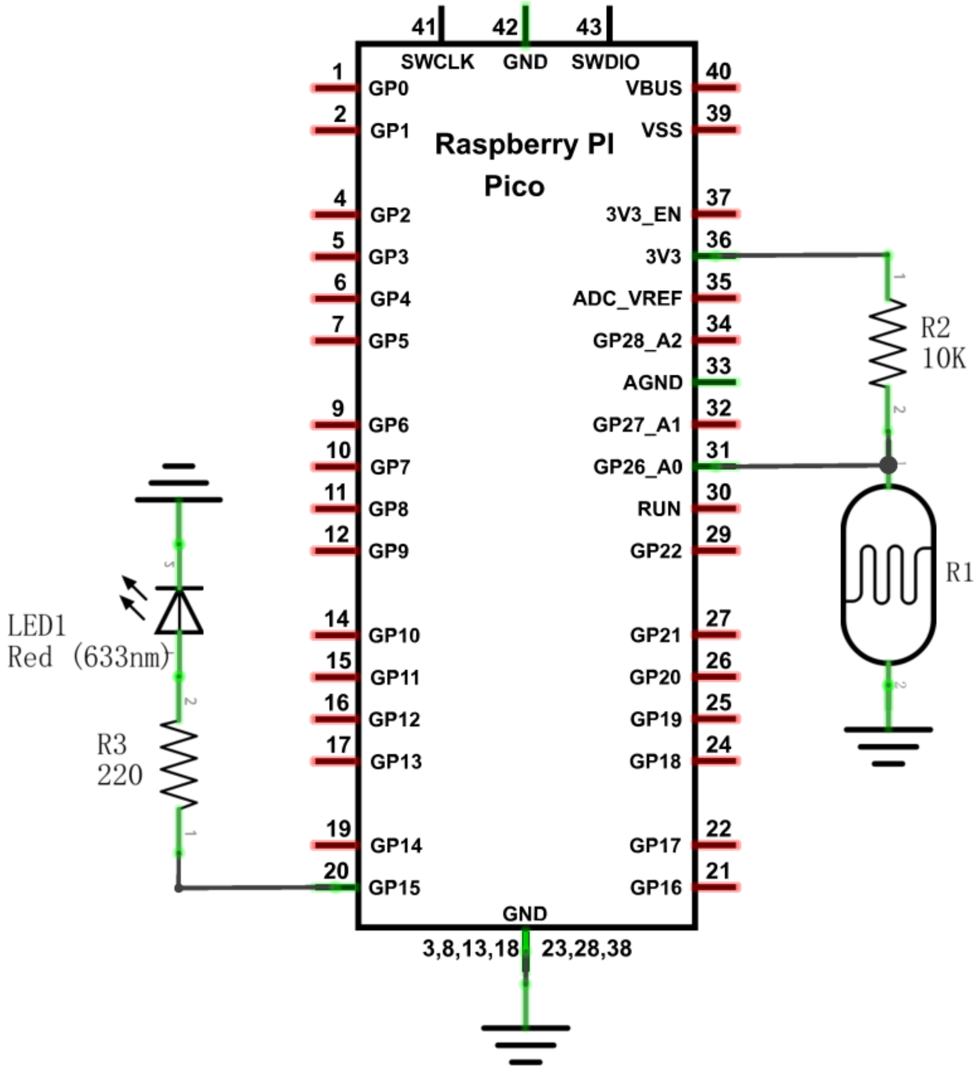



In the above circuit, when a Photoresistor's resistance value changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

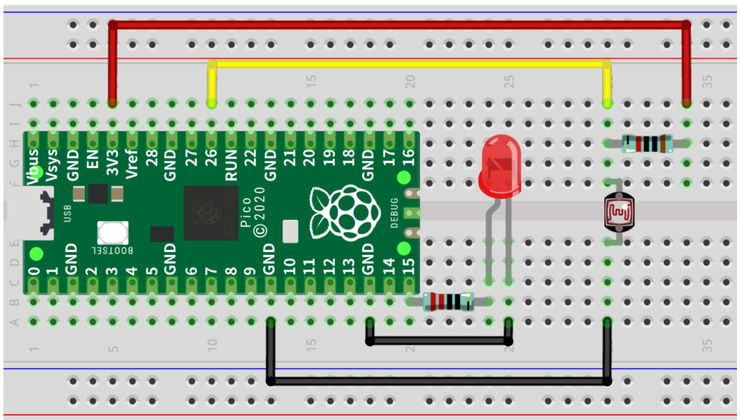
## Circuit

The circuit of this project is similar to SoftLight. The only difference is that the input signal is changed from a potentiometer to a combination of a photoresistor and a resistor.

Schematic diagram




Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

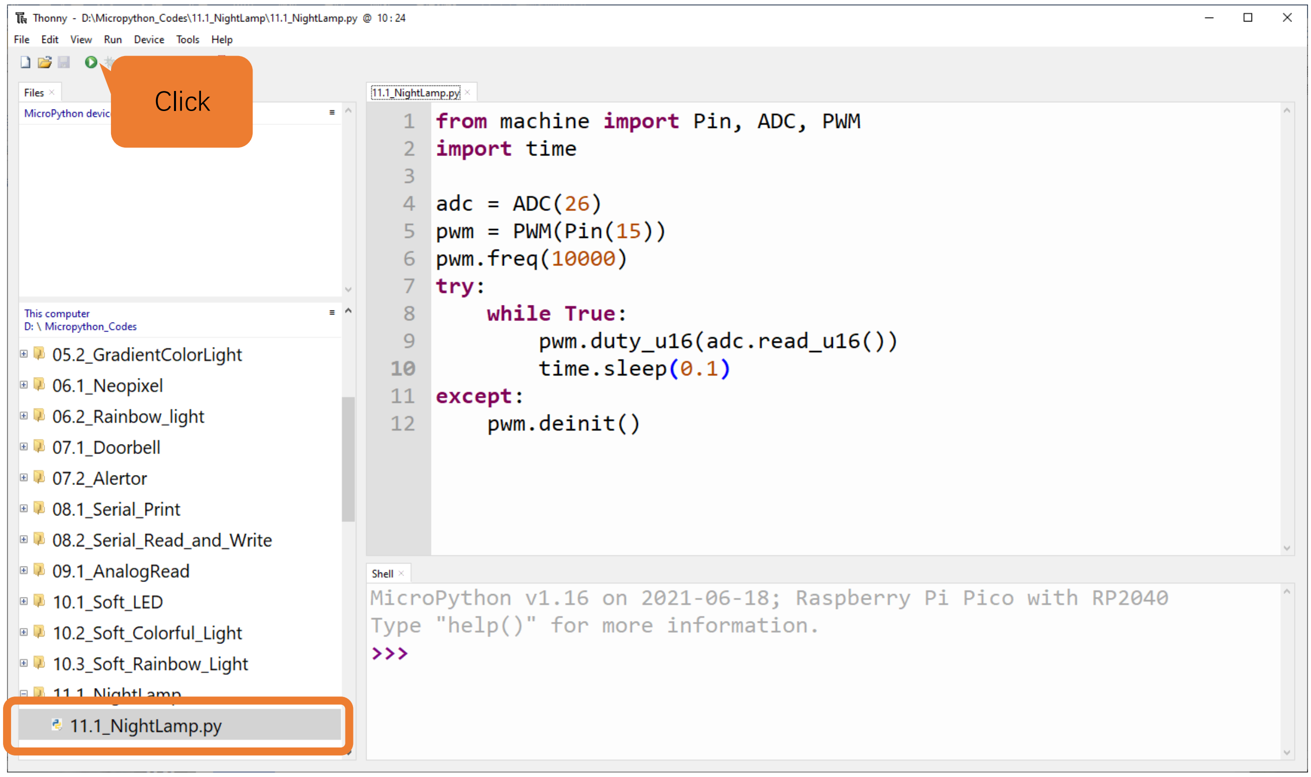


## Code

Codes of this project is logically the same as the project Soft Light.

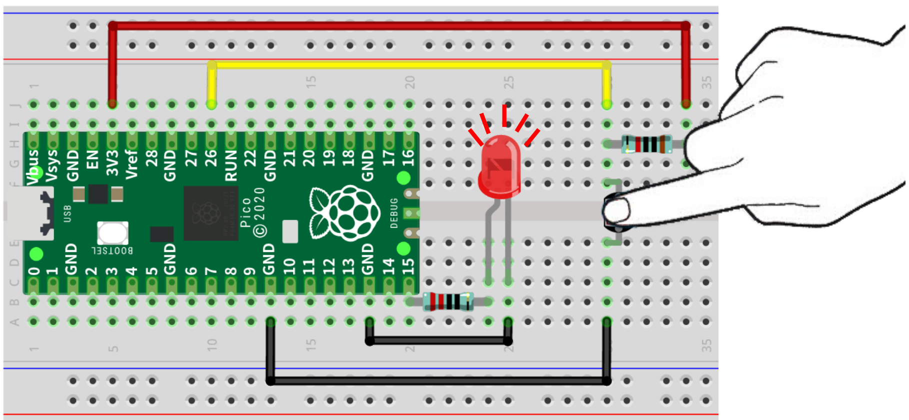
Open “Thonny”, click “This computer” → “D:” → “Micropython_Codes” → “10.1_Photoresistor” and double click “10.1_Photoresistor.py”.

10.1_Photoresistor

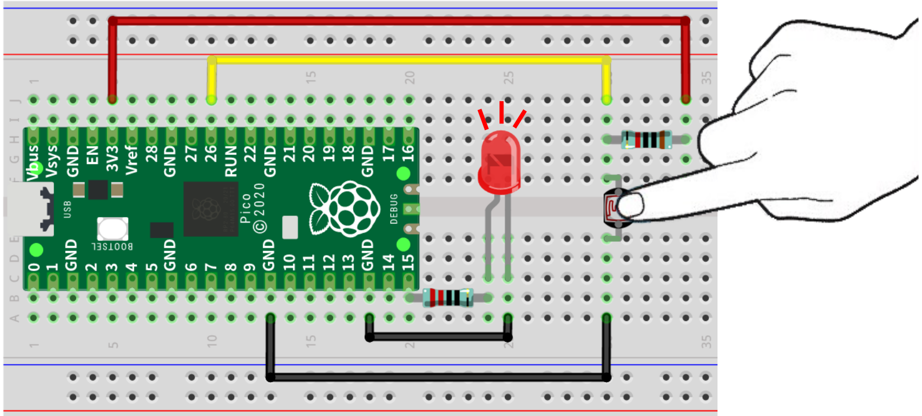



Click “Run current script”. Cover the photoresistor with your hands or illuminate it with lights, the brightness of LEDs will change.

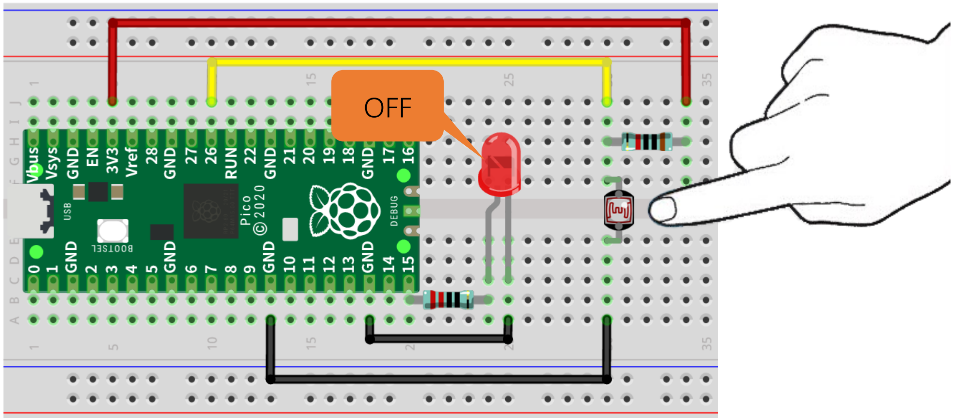
Fully cover the photoresistor:



Half cover the photoresistor:



Not cover the photoresistor:




The following is the program code:

```
from machine import Pin, ADC, PWM
import time

adc = ADC(26)
pwm = PWM(Pin(15))
pwm.freq(10000)
try:
    while True:
        pwm.duty_u16(adc.read_u16())
        time.sleep(0.1)
except:
    pwm.deinit()
```

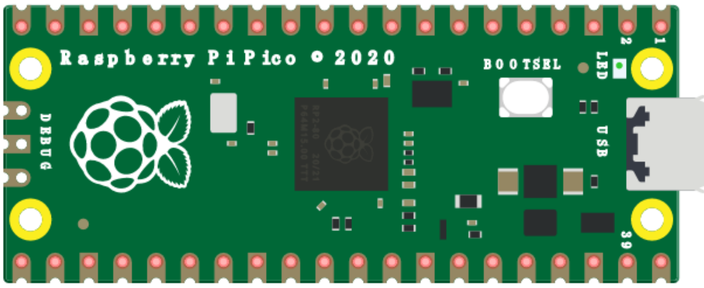
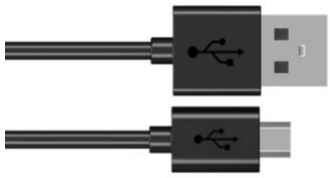
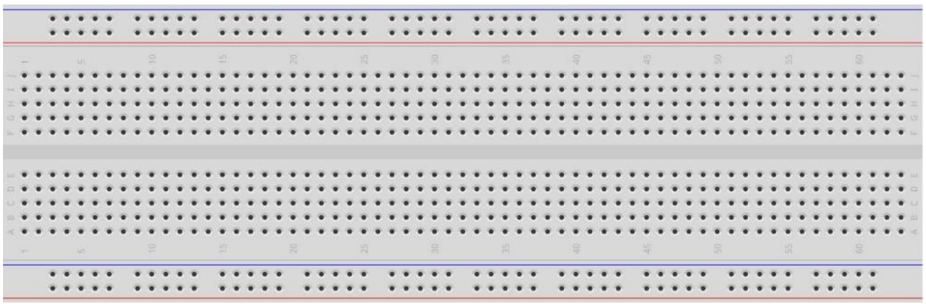
# Chapter 11 Thermistor

In this chapter, we will learn about Thermistors which are another kind of Resistor.

## Project 11.1 Thermometer

A Thermistor is a type of Resistor whose resistance value is dependent on temperature and changes in temperature. Therefore, we can take advantage of this characteristic to make a Thermometer.

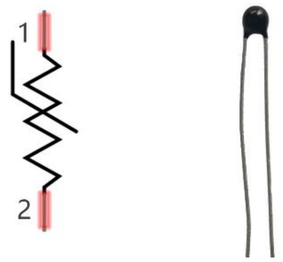
### Component List

<table>
<tr><td colspan="2">Raspberry Pi Pico x1<br>
</td><td>USB cable x1<br></td></tr>
<tr><td colspan="3">Breadboard x1<br></td></tr>
<tr><td>Thermistor x1<br></td><td>Resistor 10kΩ x1<br></td><td>Jumper<br></td></tr>
</table>

## Component knowledge

### Thermistor

A Thermistor is a temperature sensitive resistor. When it senses a change in temperature, the resistance of the Thermistor will change. We can take advantage of this characteristic by using a Thermistor to detect temperature intensity. A Thermistor and its electronic symbol are shown below.




The relationship between resistance value and temperature of a thermistor is:

$$\mathrm{Rt} = \mathrm{R} * \mathrm{EXP}[\ \mathrm{B} * (\frac{1}{\mathrm{T2}} - \frac{1}{\mathrm{T1}})\ ]$$

**Where:**

Rt is the thermistor resistance under T2 temperature;

**R** is the nominal resistance of thermistor under T1 temperature;

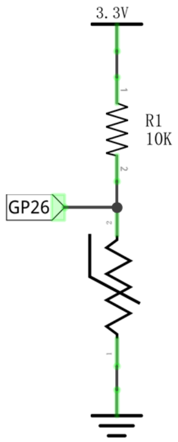
**EXP[n]** is nth power of e;

**B** is for thermal index;

T1, T2 is Kelvin temperature (absolute temperature). Kelvin temperature=273.15 + Celsius temperature.

For the parameters of the Thermistor, we use: B=3950, R=10kΩ, T1=25℃.

The circuit connection method of the Thermistor is similar to photoresistor, as the following:




We can use the value measured by the ADC converter to obtain the resistance value of Thermistor, and then we can use the formula to obtain the temperature value.

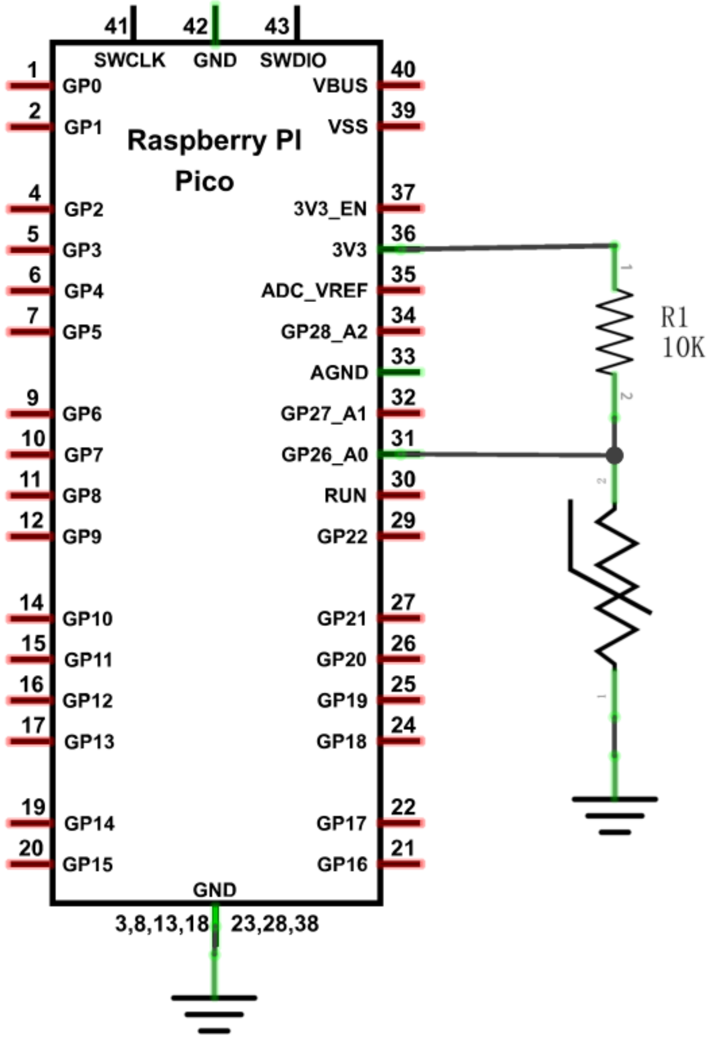
Therefore, the temperature formula can be derived as:

$$\mathrm{T2} = 1/(\frac{1}{\mathrm{T1}} + \ln(\frac{Rt}{\mathrm{R}})/B)$$

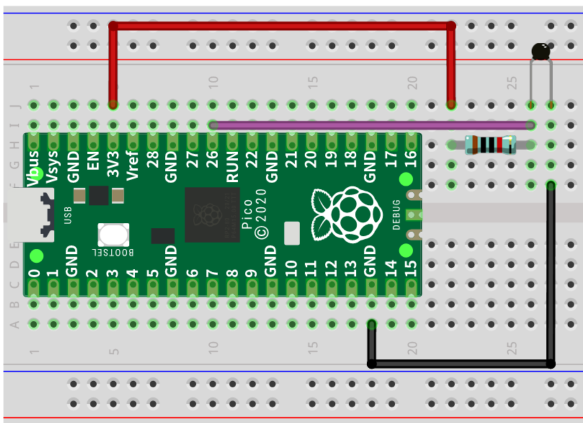
## Circuit

The circuit of this project is similar to the one in the previous chapter. The only difference is that the Photoresistor is replaced by a Thermistor.

Schematic diagram



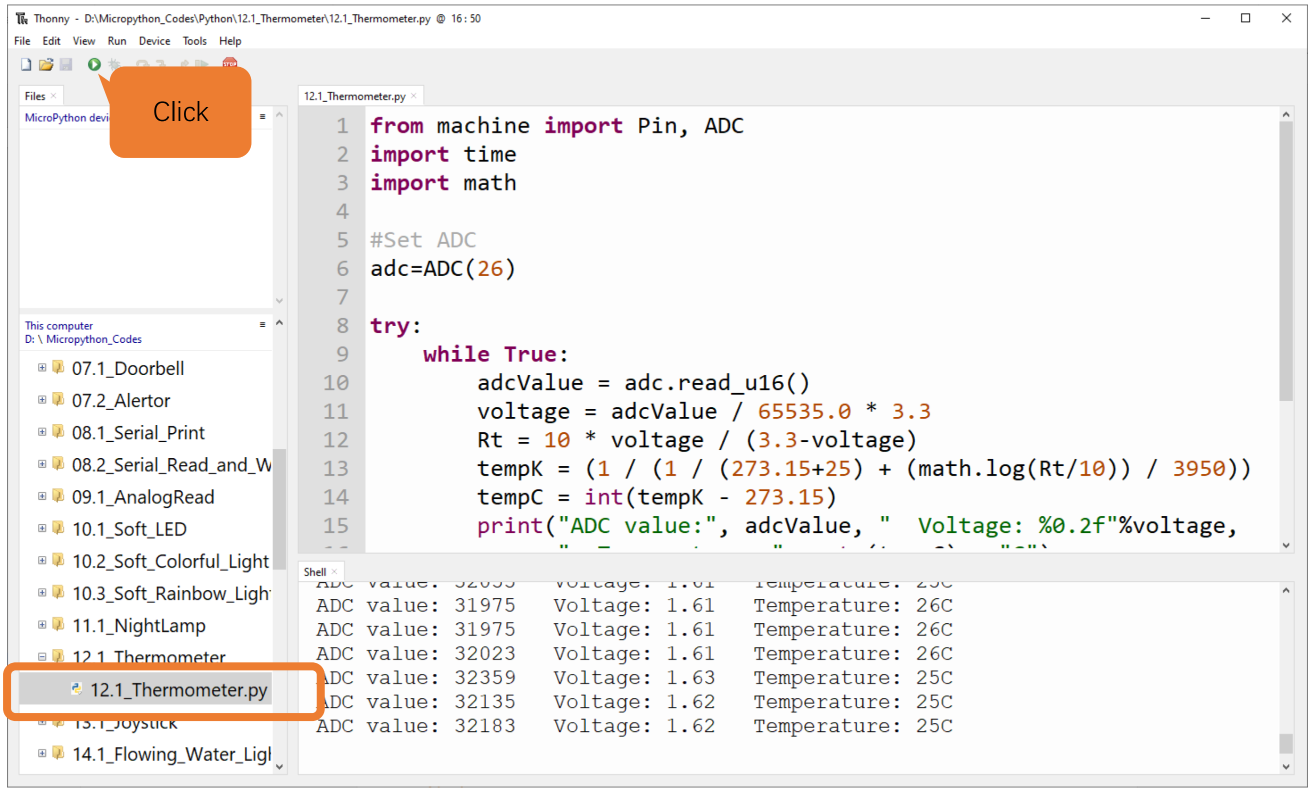

Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



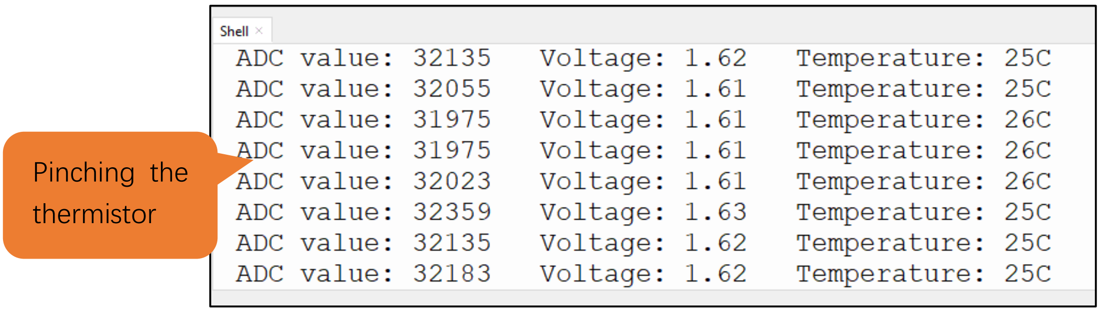
## Code

Open "Thonny", click "This computer" → "D:" → "Micropython_Codes" → "11.1_Thermometer" and double click "11.1_Thermometer.py".

11.1_Thermometer




Click "Run current script" and "Shell" will constantly display the current ADC value, voltage value and temperature value. Try to "pinch" the thermistor (without touching the leads) with your index finger and thumb for a brief time, you should see that the temperature value increases. Press Ctrl+C or click "Stop/Restart backend" to exit the program.




**If you have any concerns, please contact us via: support@freenove.com**

The following is the code:

```
from machine import Pin, ADC
import time
import math

#Set ADC
adc=ADC(26)

try:
    while True:
        adcValue = adc.read_u16()
        voltage = adcValue / 65535.0 * 3.3
        Rt = 10 * voltage / (3.3-voltage)
        tempK = (1 / (1 / (273.15+25) + (math.log(Rt/10)) / 3950))
        tempC = int(tempK - 273.15)
        print("ADC value:", adcValue, "  Voltage: %0.2f"%voltage,
              "  Temperature: " + str(tempC) + "C")
        time.sleep(1)
except:
    pass
```

read_u16() function is called to read the value of ADC0.

```
    adcValue = adc.read_u16()
```

Convert the read ADC0 value to get the current Thermistor voltage value.

```
    voltage = adcValue / 65535.0 * 3.3
```

The current Thermistor resistance (Rt) is calculated by Ohm's law.

```
    Rt = 10 * voltage / (3.3-voltage)
```

According to the formula: $\mathrm{T2} = 1/(\frac{1}{\mathrm{T1}} + \ln(\frac{Rt}{\mathrm{R}})/B)$, where T1 = 25℃, R = 10KΩ, B = 3950 and the Rt calculated in the previous step, substitute the formula to calculate tempK(T2) . Get the value of the temperature unit K.

```
    tempK = (1 / (1 / (273.15+25) + (math.log(Rt/10)) / 3950))
```

Finally, tempK (unit: K) is converted to tempC (unit: °C). You can also convert to Fahrenheit based on your needs.

```
    tempC = int(tempK - 273.15)
```

# What's Next?

THANK YOU for participating in this learning experience!

We have reached the end of this Tutorial. If you find errors, omissions or you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us: **support@freenove.com**
We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.

If you want to learn more about Arduino, Raspberry Pi, Smart Cars, Robotics and other interesting products in science and technology, please continue to visit our website. We will continue to launch fun, cost-effective, innovative and exciting products.
https://www.freenove.com/

Thank you again for choosing Freenove products.