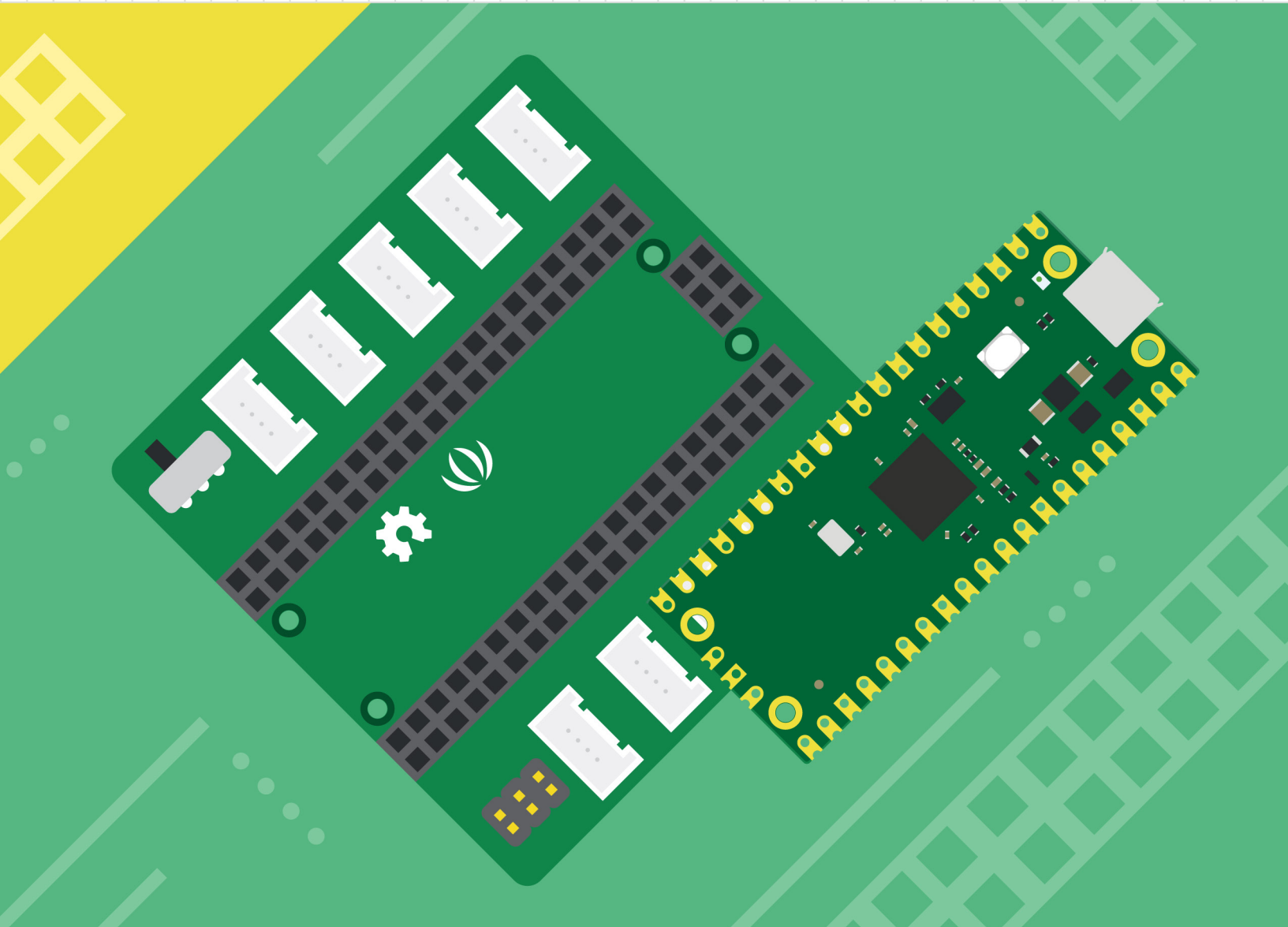
TinkerGen
STEM made simple

seeed studio

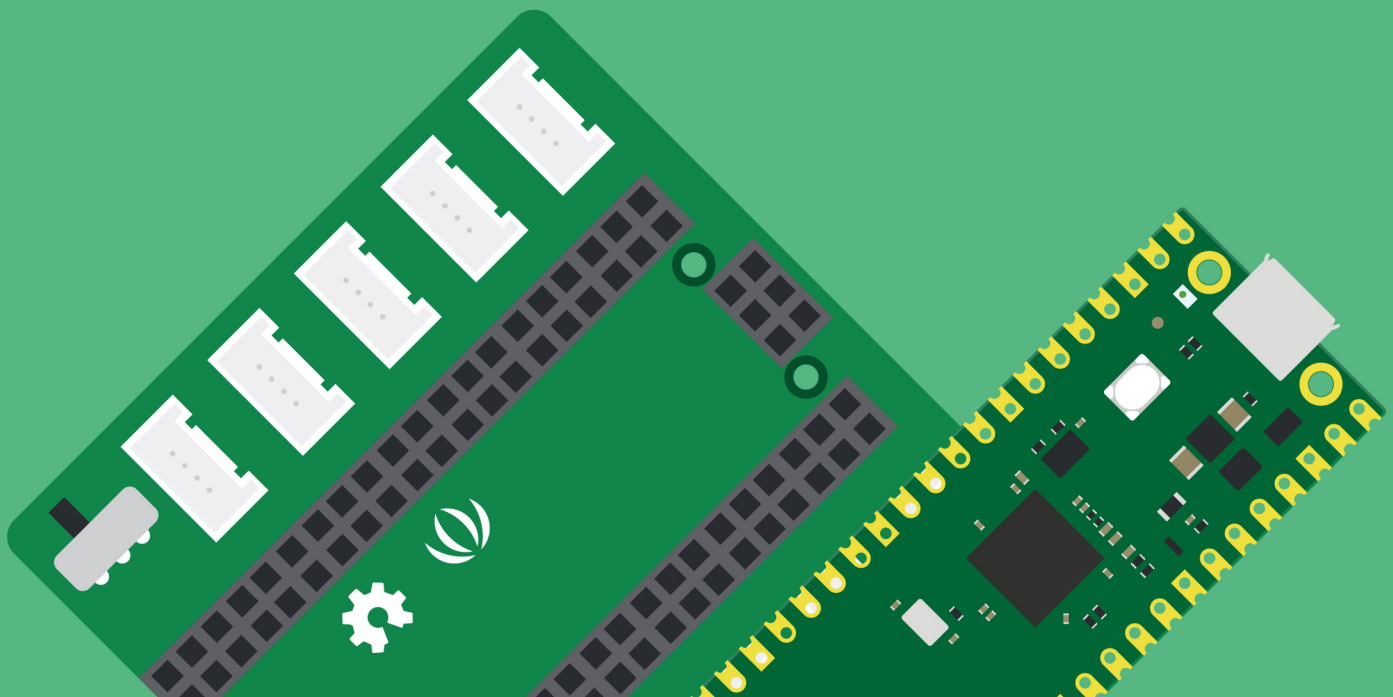
Beginner's Guide for Raspberry Pi Pico



Catalog

Preface

Raspberry Pi Pico is a microcontroller board based on the Raspberry Pi RP2040 microcontroller chip. It is designed as a low-cost, high-performance microcontroller board with flexible digital interfaces. In the field of MCUs, Raspberry Pi Pico has become one of the most popular topics recently. To help beginners get started with Raspberry Pi Pico quickly, we are releasing the Starter Guide of Raspberry Pi Pico Based on MicroPython. Through four chapters and a total of 16 lessons of study, this course can help you get started with Raspberry Pi Pico easily. With its help, you can learn MicroPython from scratch, and build some interesting projects. You don't need to have knowledge of MicroPython programming or electronics to begin. The course will take you through learning this knowledge step by step, and quickly put it into practice in each project.



The first unit is an introduction to basics, which presents the hardware and software required for the course. In the hardware portion, we need to understand the functions and features of Raspberry Pi Pico, and the basic functions of Grove Shield for Pi Pico and Grove modules. In software, we need to understand the concepts of Python and MicroPython, learn the installation and use of the Thonny integrated development environment or IDE, and write our first program to output characters through serial port: Hello World.

The second unit continues with more basic electronic hardware knowledge. We will learn the basic syntax of MicroPython, various conditional statements, operators, variables and other programming knowledge, and how to set the pins of electronic hardware through the standard library. We will understand digital and analog signals in the circuit and be able to control the electronic module to achieve different effects. At the same time, we will learn to use third-party libraries and form a basic hardware programming knowledge framework through project practice.

The third unit begins project practice. We will integrate applications with real world scenarios and realize various prototypes through different electronic module combinations and programming, such as temperature and humidity monitoring, intelligent fan, intelligent light, etc. In this way, we can skillfully apply what we have learned to life, and in so doing practice and develop project thinking.The fourth unit outlines the steps to develop your own self-research project. We will use the knowledge and skills learned before to complete the project production and show the results at a project conference.

The specific framework is as follows:

Lesson	Title	Summary
Introduction to Basis		
Preface		Course introduction, course outline, hardware list and main hardware parameters
Lesson 1	Introduction to Hardware	Introduce the functions and features of Raspberry Pi Pico; Introduce Grove Shield for Pi Pico; Introduce Grove electronic modules in the kit.
Lesson 2	Introduction to Software	Introduce what is Python/ MicroPython; Introduce the basic status of integrated development environment Thonny; Explain the basic usage of Thonny; Explain how to connect Pico to a computer; Project: output characters with serial port: Hello World.

Introduction to Electronic Hardware		
Lesson 3	First Project: Light up LED	Explain the basic syntax of MicroPython, understand the meaning of indentation in MicroPython; Explain the pin layout of Pico, and how to solder pins for Pico; Explain how to plug Pico into Grove Shield for Pi Pico, and how to connect an electronic module with the Shield. Project 1: soldering the headers; Project 2: light up the LED module.
Lesson 4	Hello World in Electronic Hardware Programming: Blink	Explain the basic syntax of Python/ MicroPython; learn how to use loop statements; learn how to use conditional judgment statements; learn how to set variables in MicroPython. Project 1: control LED on and off with for-loop; Project 2: realize Blink with while-loop.
Lesson 5	Digital Signal in Circuit — Playing with LED	Introduce digital signal; Introduce selection structure; Learn to master the condition judgment and the use of variables to achieve different program effects. Project 1: control LED on and off with button; Project 2: button light.
Lesson 6	Analog Signal in Circuit (1) — Rotary Angle Sensor	Introduce analog signal, Analog to Digital Converter, operator, and other knowledge; learn to control the LED light on and off with the Rotary Angle Sensor. Project 1: check return value of Rotary Angle Sensor; Project 2: control LED on and off with Rotary Angle Sensor.
Lesson 7	Analog Signal in Circuit (2) — the Use of PWM	Learn the concept and use of pulse width modulation or PWM; learn to use PWM signals for analog output and control the brightness of the LED. Project 1: control LED brightness with Rotary Angle Sensor; Project 2: breathing light.
Lesson 8	Make Buzzer Sing — the Use of Function	Introduce buzzer; introduce the concept of function; learn to customize function and call function, and complete the project. Project 1: playing basic notes; Project 2: two tigers.
Lesson 9	Journey of Data Visualization — the Use of LCD	Introduce the use of LCD; learn to load and install libraries; learn the reading and display of sensors; learn the concept of data types. Project 1: display "Hello, world!" with LCD; Project 2: display Rotary Angle Sensor reading with LCD.

Project Practice		
Lesson 10	Temperature and Humidity Monitoring	Introduce Temperature & Humidity Sensor; learn multiple condition judgment, logical operators (AND/ OR/ NOT); learn to complete the temperature and humidity monitoring project with electronic module and program. Project 1: display temperature and humidity value with LCD screen; Project 2: add alarm function.
Lesson 11	Intelligent Fan	Introduce Mini Fan; review multiple conditional judgment and the use of logical operators. Project 1: control fan on or off with button; Project 2: control fan on or off with Temperature & Humidity Sensor.
Lesson 12	Intelligent Light	Introduce light sensor, sound sensor and RGB LED; learn to complete the intelligent light project according to the functions of light sensor and sound sensor; learn to skillfully use multiple condition judgment and logical operators. Project 1: control RGB LED to display different colors in turn; Project 2: intelligent light.
Lesson 13	Automatic Door	Introduce PIR Motion Sensor and Servo; learn how to write a library and call library function to simplify program and complete project production. Project 1: control Servo; Project 2: control Servo rotation with PIR Motion Sensor.
Lesson 14	Welcome Device	Summarize the projects of this chapter; explore new projects through different combinations of electronic hardware; prepare for creating your own projects; consolidate the creation and use of library through practice. Project 1: play welcome music with buzzer; Project 2: welcome device.
Self-research Project		
Lesson 15	Create Your Own Interesting Projects	Divide into groups, review and introduce the initial product prototyping design process again, and make it in groups.
Lesson 16	Project Conference	Production, display, communication and summary.

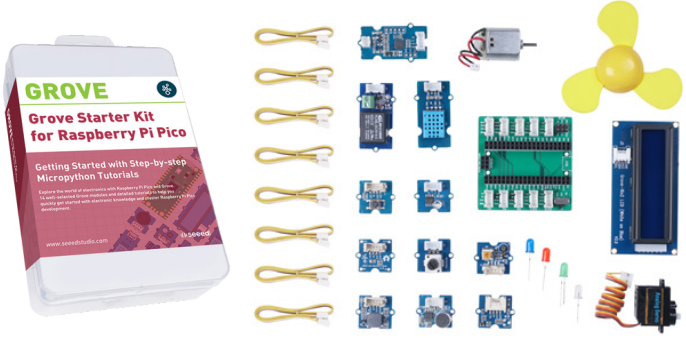
The course uses Pico's Grove Starter Kit, including Grove Shield for Pi Pico and common Grove modules, which can enable quick project protyping without jumper wires and welding, and consolidate the learned knowledge in practice. Because the Grove Starter Kit does not include the Raspberry Pi Pico itself, we will need to prepare the following two kits before the start of the course.

Raspberry Pi Pico Basic Kit:

- Raspberry Pi Pico * 1
- USB cable * 1
- 20 Pin Header * 2
- 3 Pin Header * 1

Grove Starter Kit of Raspberry Pi Pico:

- Grove – Light Sensor
- Grove – Sound Sensor
- Grove – Mini PIR Motion Sensor
- Grove – Temperature & Humidity Sensor
- Grove – Rotary Angle Sensor
- Grove – Button
- Grove – LED Pack
- Grove – RGB LED (WS2813 Mini)
- Grove – Passive Buzzer
- Grove – Relay
- Grove – 16x2 LCD
- Grove – Servo
- Grove – Mini Fan



Purchase link:

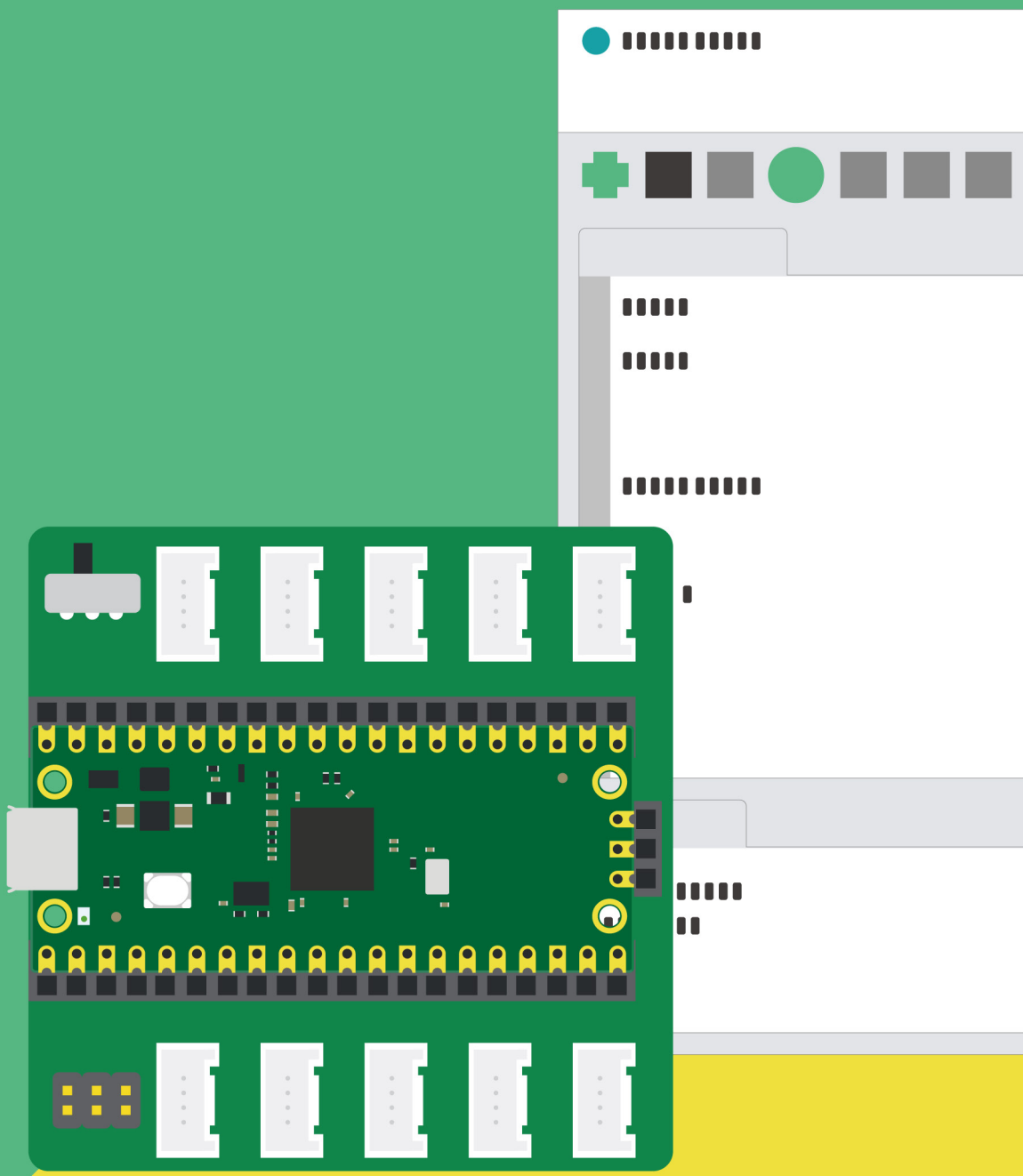
Grove Starter Kit of Raspberry Pi Pico
Raspberry Pi Pico Pre-Soldered

Visit the following URL to get the codes & Libraries:

Codes: https://files.seeedstudio.com/wiki/Grove_Shield_for_Pi_Pico_V1.0/Codes.rar
Libraries: https://files.seeedstudio.com/wiki/Grove_Shield_for_Pi_Pico_V1.0/Libraries.rar

Are you ready? Our journey with Raspberry Pi Pico and MicroPython is about to begin.

Unit 1
Getting started



Lesson 1 Introduction to Hardware

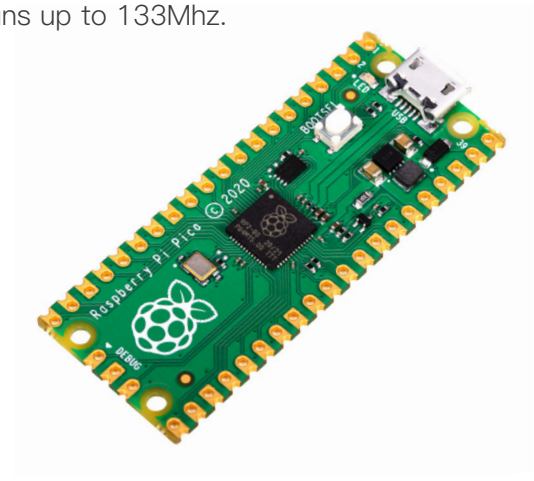
Hello! Welcome to the starter guide of Raspberry Pi Pico based on MicroPython. This guide is a course based on the Starter Kit developed by Raspberry Pi Pico. In this course, we will use Raspberry Pi Pico as the controller, along with the Grove Starter Kit of Raspberry Pi Pico, to learn how to program Pico with MicroPython. In the first lesson of this series, I will give you a detailed introduction to the main characters of this series: Raspberry Pi Pico and Pico's Grove Starter Kit.

Knowledge Base

Raspberry Pi Pico

Introduction of Pico

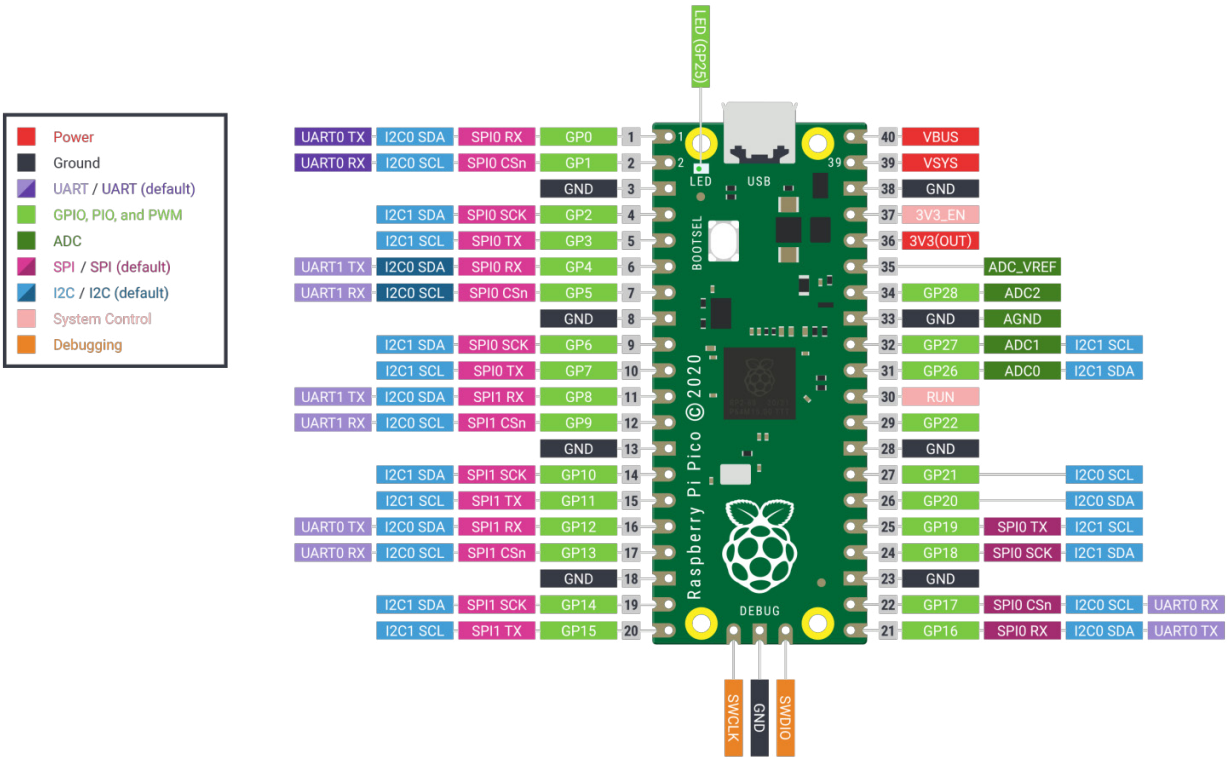
No one expected that Raspberry Pi, the most popular single-board computer maker in the world, would suddenly release a microcontroller of its own. What's more surprising is that Raspberry Pi Pico does not base its design on the common ESP32 or SAMD21, but instead a brand new microcontroller chip: the RP2040 microcontroller. The RP2040 microcontroller is a microcontroller chip independently designed by Raspberry Pi, and is powered by a dual-core ARM Cortex-M0+ processor that runs up to 133Mhz.



GPIO Pins

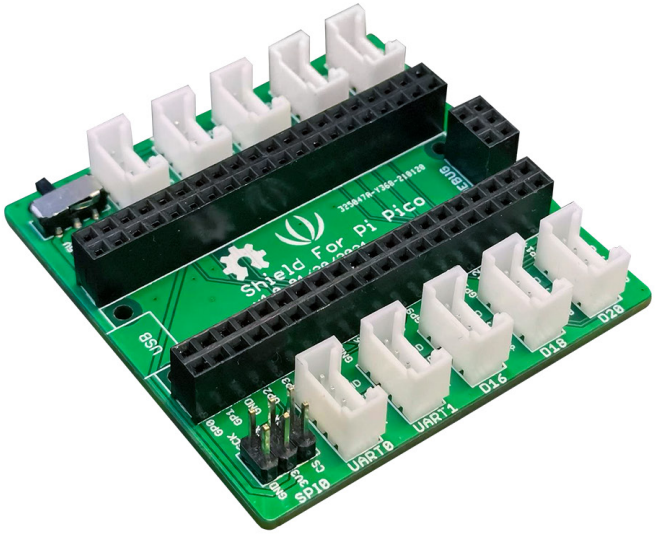
In addition to the powerful new chip, the board of Raspberry Pi Pico exposes 26 multi-function GPIO pins, including 2 * SPI, 2 * I2C, 2 * UART, 3 * 12-bit ADC, and 16 controllable PWM channels. I will explain the functions of these pins in later chapters.

In addition to these GPIO pins, Pico also has eight ground pins and a series of power pins. In this course, however, we will not utilize them as we will not be performing tedious wiring with breadboards or DuPont wires when building projects. The expansion board — Grove Shield for Pi Pico — will helps you do this instead.



Grove Shield for Pi Pico

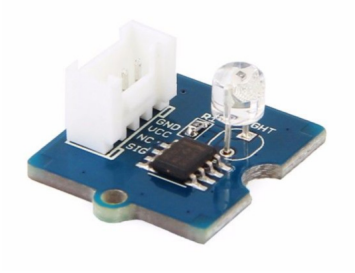
Grove Shield for Pi Pico is a plug-and-play expansion board for Raspberry Pi Pico. It integrates various kinds of Grove connectors on one compact expansion board, including 2 * I2C, 3 * ADC, 2 * UART, 3 * Digital ports, SWD debug interface and an SPI pin. When using Pico and Grove modules to build a prototype or project, you no longer need to do complicated wiring work. You simply need to plug the Pico pins into the interface of the Shield, and directly connect various Grove electronic modules to Pico using Grove wires.



There is also a 3.3V/5V power switch on the Shield, which can switch the voltage between 3.3V/5V to meet the needs of a greater variety of projects.

Grove modules in Starter Kit

The Grove Starter Kit of Raspberry Pi Pico is a learning kit that can help you quickly start building projects with Raspberry Pi Pico. The kit contains 14 handpicked modules, including 5 sensors/ 5 actuators/ 2LED/ 1 LCD display/ 1 Grove Shield for Pi Pico. Let's take a look at these Grove modules one by one.



Grove – Light Sensor

Grove – Light Sensor is an analog module, which can output the amount of light detected as various electrical signals to an attached microcontroller board. It is an ideal LDR sensor module for light measurement, light detection and light-controlled switch.

Grove – Sound Sensor

Grove – Sound Sensor can detect the sound intensity in the environment. The main component of the module is a simple microphone, which consists of a L358 amplifier and an electret microphone.

Grove – Mini PIR Motion Sensor

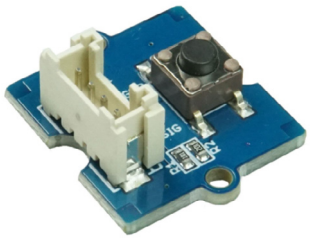
Grove – Mini PIR Motion Sensor uses passive infrared radiation to detect motion, and is often used in projects such as alarm systems, visitor systems, light switches, etc.

Grove – Temperature & Humidity Sensor

Grove – Temperature & Humidity Sensor is mainly used to detect the temperature and humidity values in the environment. The early temperature & humidity sensor was DHT11. It was discontinued and replaced with DHT20 later. Please refer to Lesson 10 for the difference between the two.

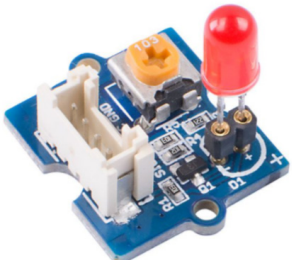
Grove – Rotary Angle Sensor

Grove – Rotary Angle Sensor can change the resistance value to achieve different output levels by rotating the 300-degree adjusting knob on the potentiometer. It has a maximum resistance value of 10KΩ.



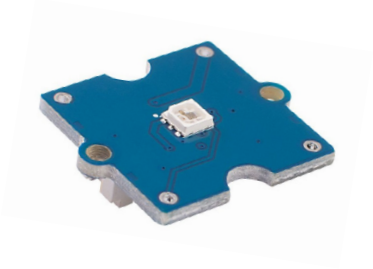
Grove – Button

Grove – Button is an independent "momentary on/off" button. "Momentary" means that the button rebounds on its own after it is released. The button outputs a high signal when pressed, and a low when released.



Grove – LED Pack

Grove – LED Pack includes LEDs in four different colours: red, green, blue and white. You can change the light color at any time by swapping out the LED with other LEDs in the LED Pack. There is also a knob on the module to control the LED brightness.



Grove – RGB LED (WS2813 mini)

Grove – RGB LED (WS2813 Mini) can be set to full-color display and can be connected to other RGB LED rings or RGB LED bars in series.

Grove – Passive Buzzer

Grove – Passive Buzzer is a low-cost tunable passive buzzer.

Grove – Relay

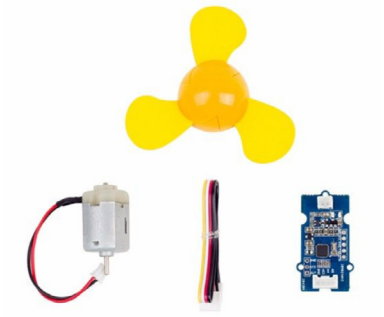
Grove – Relay is used to supply greater power to modules that require it, such as an external motor. It can handle currents of up to 5A at 250VAC for long periods of time. When the relay is switched on, the LED on the module will light up.

Grove – 16x2 LCD

Grove – 16x2 LCD is a blue backlight 16x2 LCD display. It has excellent high contrast and is easy to use.

Grove – Servo

Grove – Servo is a rotary actuator that allows for precise control of angular position, which makes them suitable for projects where precise position control is needed.



Grove – Mini Fan

Grove – Mini Fan is equipped with a DC motor, a motor drive board and a soft fan-blade. The soft fan-blade is safe for use, and the easy-to-use Grove connector can let you build your own project prototype quickly.

That's the basic information about the electronic modules included in the Raspberry Pi Pico and the Grove Starter Kits of Raspberry Pi Pico. More interesting information about them will be presented as we go along in the later courses.

Thinking Expanding

Try remembering the function of each Grove module. You can ask questions with your partners.

Lesson 2 Introduction to Software

To build interesting projects with Raspberry Pi Pico, you need to learn how to program it first. Raspberry Pi Pico supports two types of programming languages: C++ and MicroPython.

C++ is a general-purpose language developed on the basis of the C language. It is often used in the development of ESP32 and Arduino, so most microcontroller users are familiar with this language. However, to make things simple in this course, we will use another language to program Pico: MicroPython.

Knowledge Base

MicroPython

Python is an interpreted general-purpose language. Since the first version of Python was designed and released by Guido van Rossum of the National Research Institute for Mathematics and Computer Science in 1991, Python has rapidly become one of the most popular languages in the world with its concise syntax, excellent code readability and hot community resources.

·Note·

What is an interpreted language?

The computer cannot directly understand the high-level language we use when programming. When the computer executes the program, it needs to translate the code written in a high-level language into a machine language that it can understand in some way and then execute it. There are two ways to translate high-level language into machine language: using a compiler or using an interpreter to translate a high-level language.

Therefore, we refer to programming languages that use a compiler for translation as a compiled language, while a language that uses an interpreter for translation is known as an interpreted language.

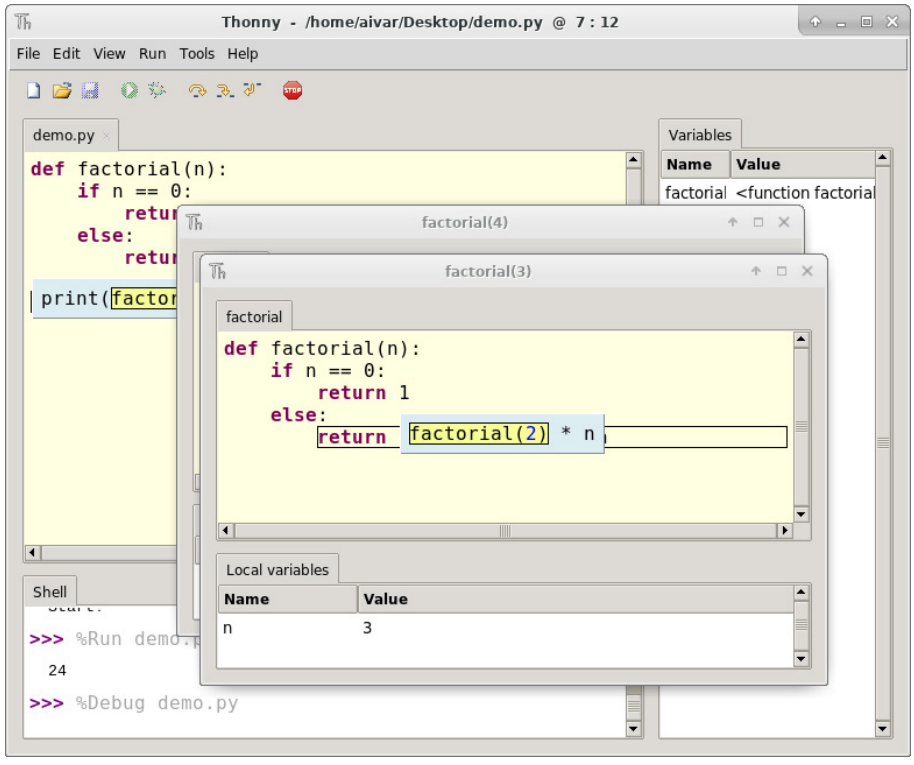
However, the powerful Python is not omnipotent. Although Python performs well on desktop computers, large terminals and servers, it cannot be deployed on microcontrollers with limited resources and small memory. As a result, MicroPython, a Python-based interpreted language specially designed for microcontrollers, was born. MicroPython optimizes Python cleverly, and the optimized MicroPython can run perfectly in all kinds of resource-constrained microcontrollers. MicroPython also inherits various advantages of Python, such as its ease of use.

MicroPython has made strides in keeping compatibility with ordinary Python as much as possible. If you are a beginner, you will learn a lot about Python while learning MicroPython. On the other hand, if you already have some understanding of Python, you will be able to quickly get started with MicroPython while deepening your knowledge.

Thonny

Thonny is a Python/ MicroPython IDE (integrated development environment) for programming beginners, developed by the University of Tartu in Estonia. The platform integrates many tools that users need to use when developing. Its interface is simple and easy to understand, which is very suitable for beginners.

In this course, all of our development work will be done in Thonny.



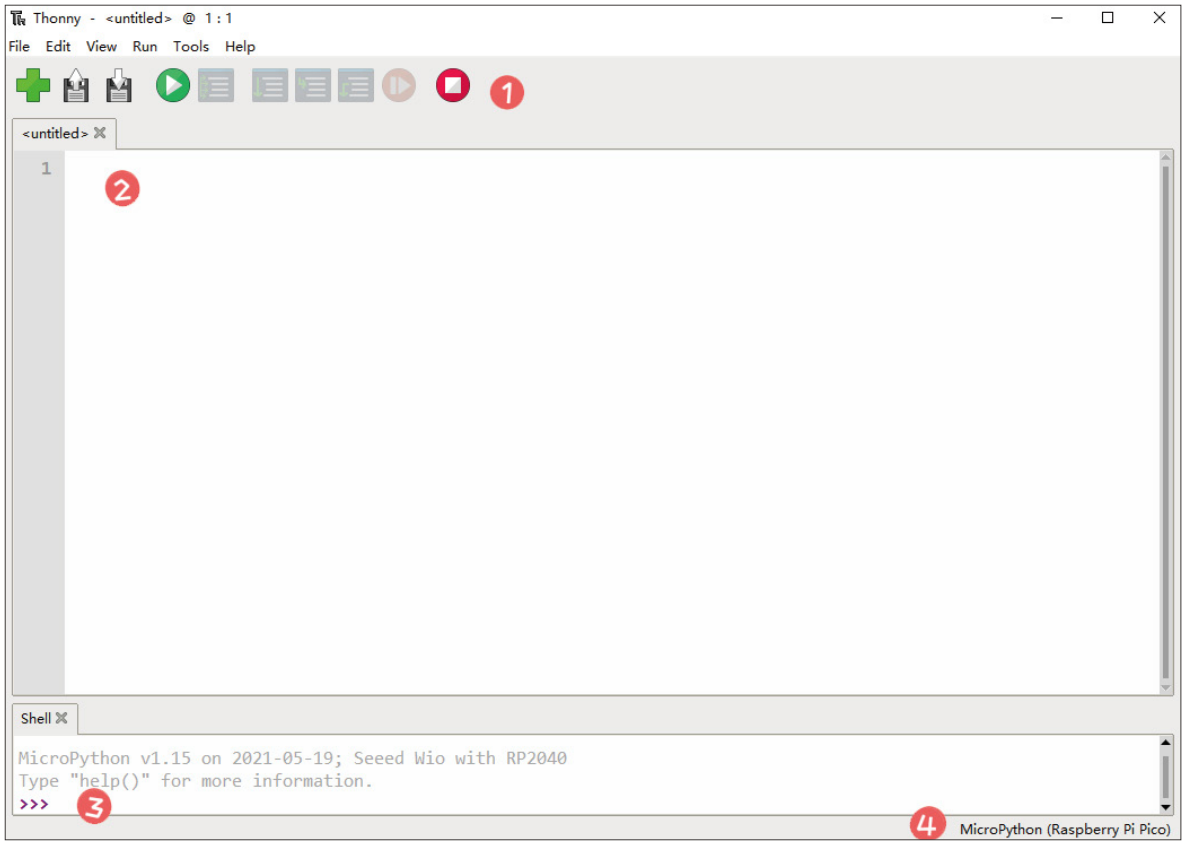
Practice & Operation

In this lesson, we will write and run our first program on Pico: Hello World! But before we start, we will need to do some preparatory work.

1. Install Thonny

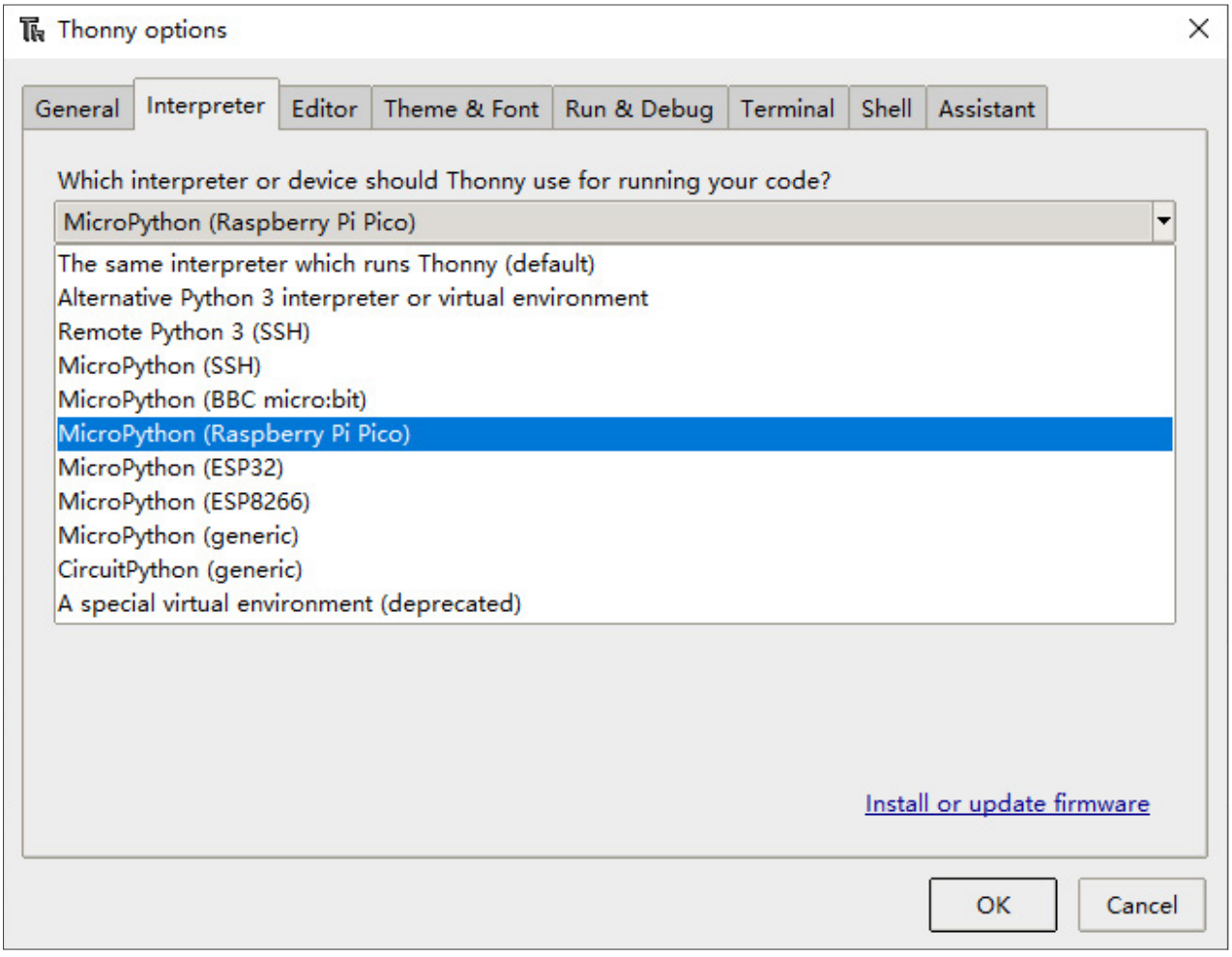
The installation of Thonny is very simple. Since Python 3.7 is built into Thonny, all you need is a simple installer and you're ready to start learning to program. First, click Thonny.org to enter the download page, and select the installation package of Thonny to download in the upper right corner of the page according to your operating system. After downloading the installation package, double-click to open it and follow the on-screen instructions to install it.

After the installation, open Thonny. You should be directly greeted by the main interface.



The main interface of Thonny is simple, and can be divided into the following four parts:

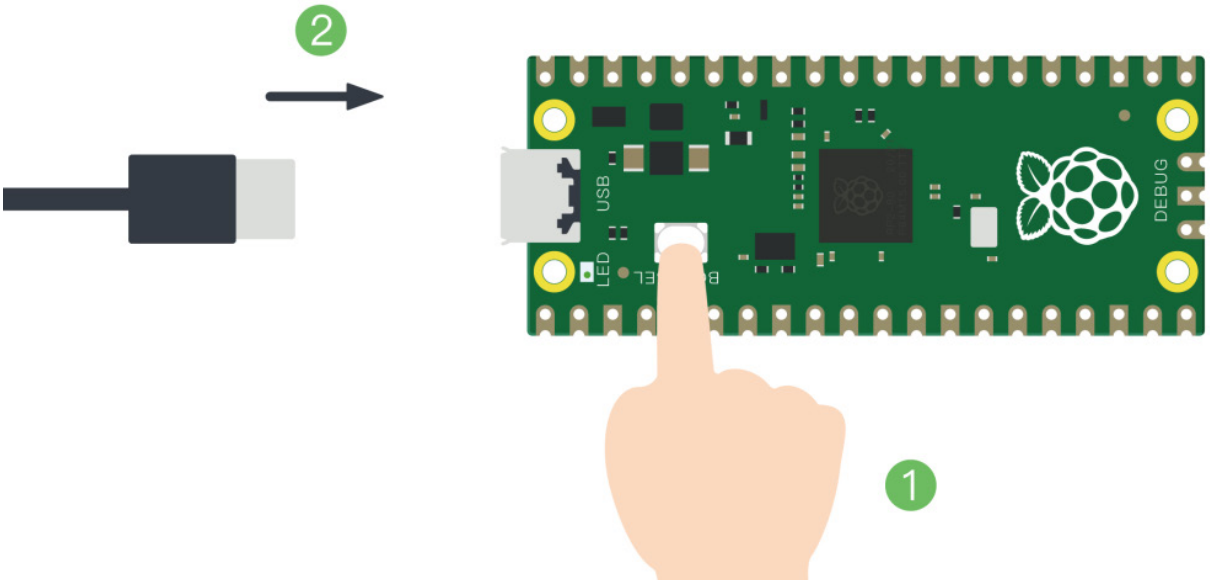
1. Toolbar: the toolbar offers icon-based quick-access to commonly used program functions, such as new, open, save, run the current script, stop, etc.
2. Script Area: the script area is the core area of Thonny, where your Python programs are written.
3. Python Shell: the Shell allows you to run console commands and also provides information about running programs.
4. Interpreter: the interpreter allows you to change the interpreter version used to run your programs. Click on "Python 3.7.9", look for "MicroPython (Raspberry Pi Pico)" in the menu, click "OK", and switch it to Pico's exclusive interpreter.We can also switch languages in "Theme & Font".



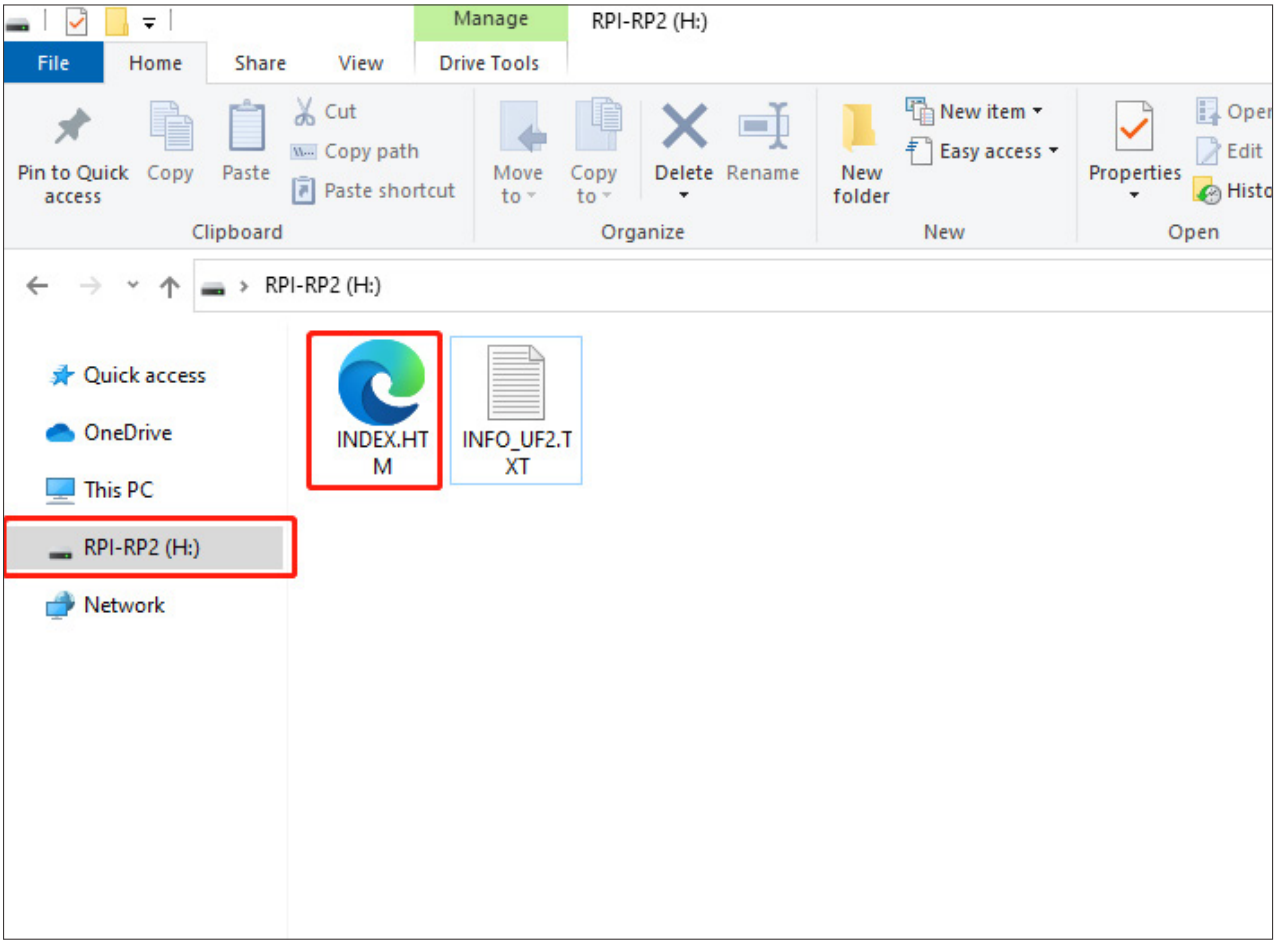
2. Install MicroPython for Pico

In addition to installing Thonny on the computer, we also need to install the MicroPython firmware on the Pico itself.The MicroPython firmware download link is preloaded inside Pico, so we can directly use this link to enter the download interface.

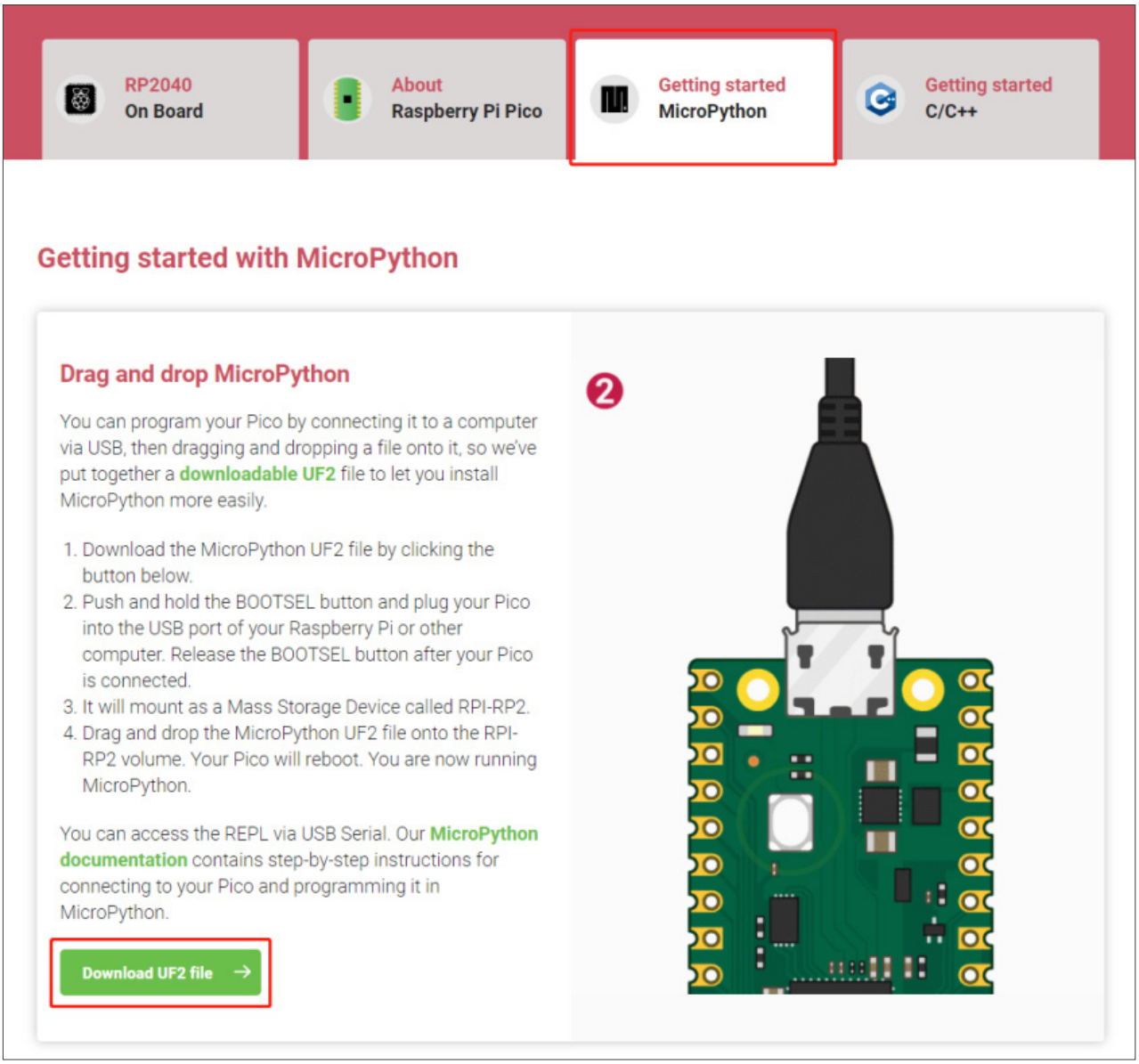
First, hold down the "BOOTSEL" button on Pico; then, while still holding it down, connect Pico to a computer using a USB cable.



At this point, a drive called RPI–RP2 should show up as a removable device on your computer. Navigate to it to find this file: INDEX.HTM.



Open INDEX.HTM to be taken to the download page. Drop down and click on the MicroPython tab where you can find information about Pico and MicroPython. Click on the “Download UF2.file” button in this tab to download the MicroPython firmware.

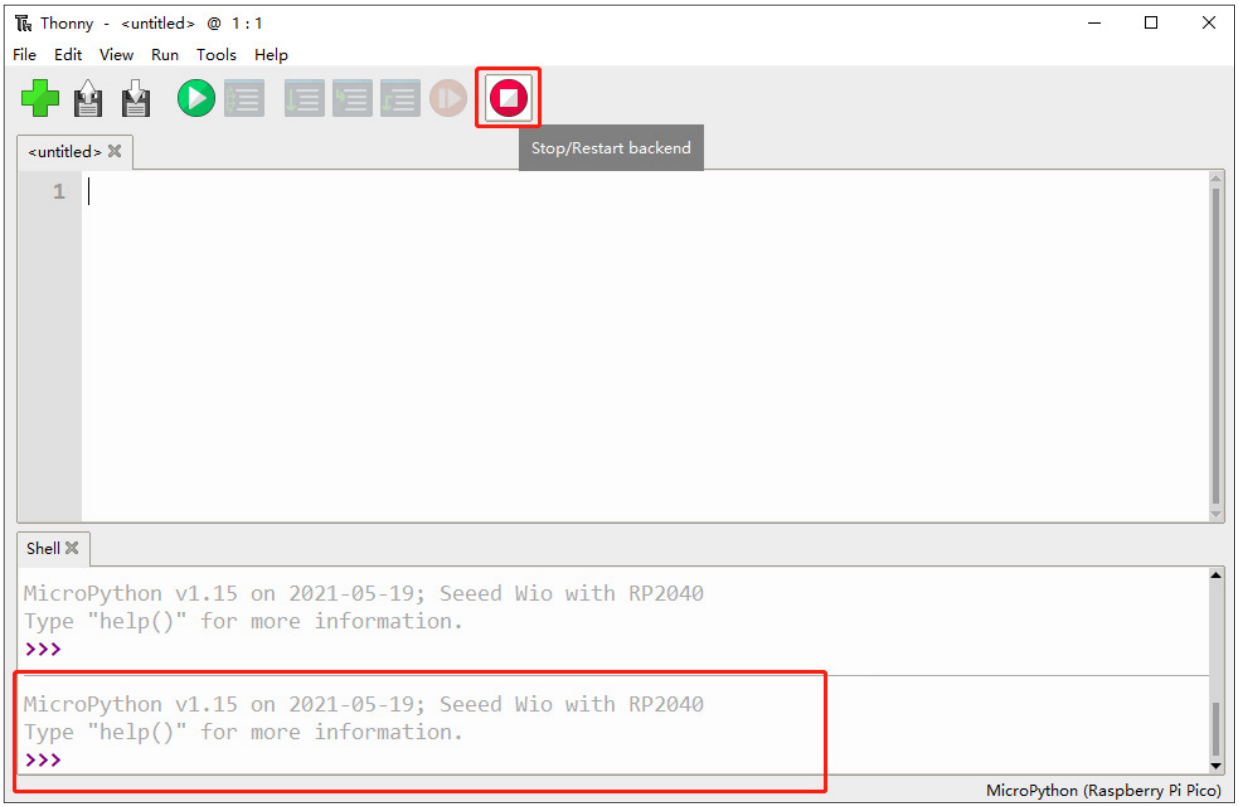


Finally, drag the downloaded MicroPython firmware into the RPI–RP2 directory to complete the firmware installation.

3. Write Your First Program: Hello World!

Just as a young child learns to crawl before they walk, programmers learn how to output "Hello, world" as the first step in learning a new language. Now, let's take a look at how to write your first program.

First, connect Pico to the computer using a USB cable; then, open Thonny, and click on the "restart back–end process" button on the toolbar. If you successfully connect Pico to your computer, you will see the MicroPython version information and device name returned by Pico in the Shell area.

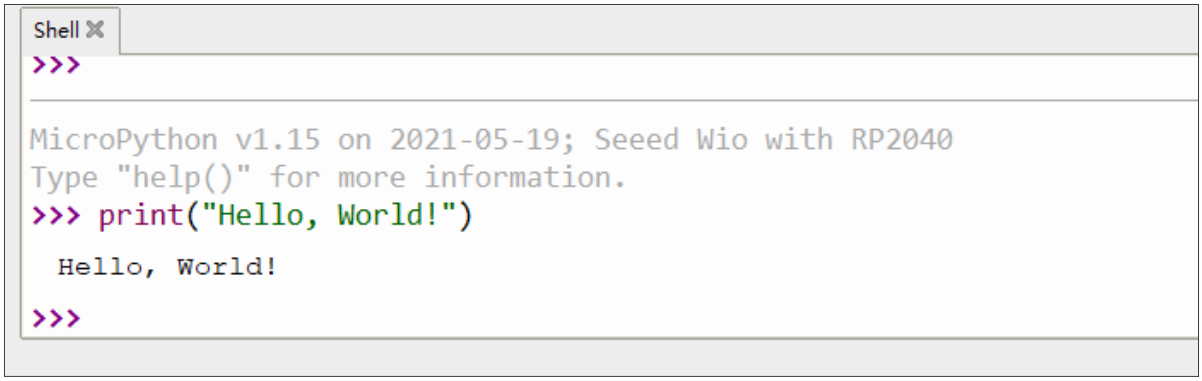


After connected, click on the Shell area and type the instruction:

```
1 print("Hello, World!")
```

print() is a common function used as a printout. When you need to output some data to the Shell, you can use this function.

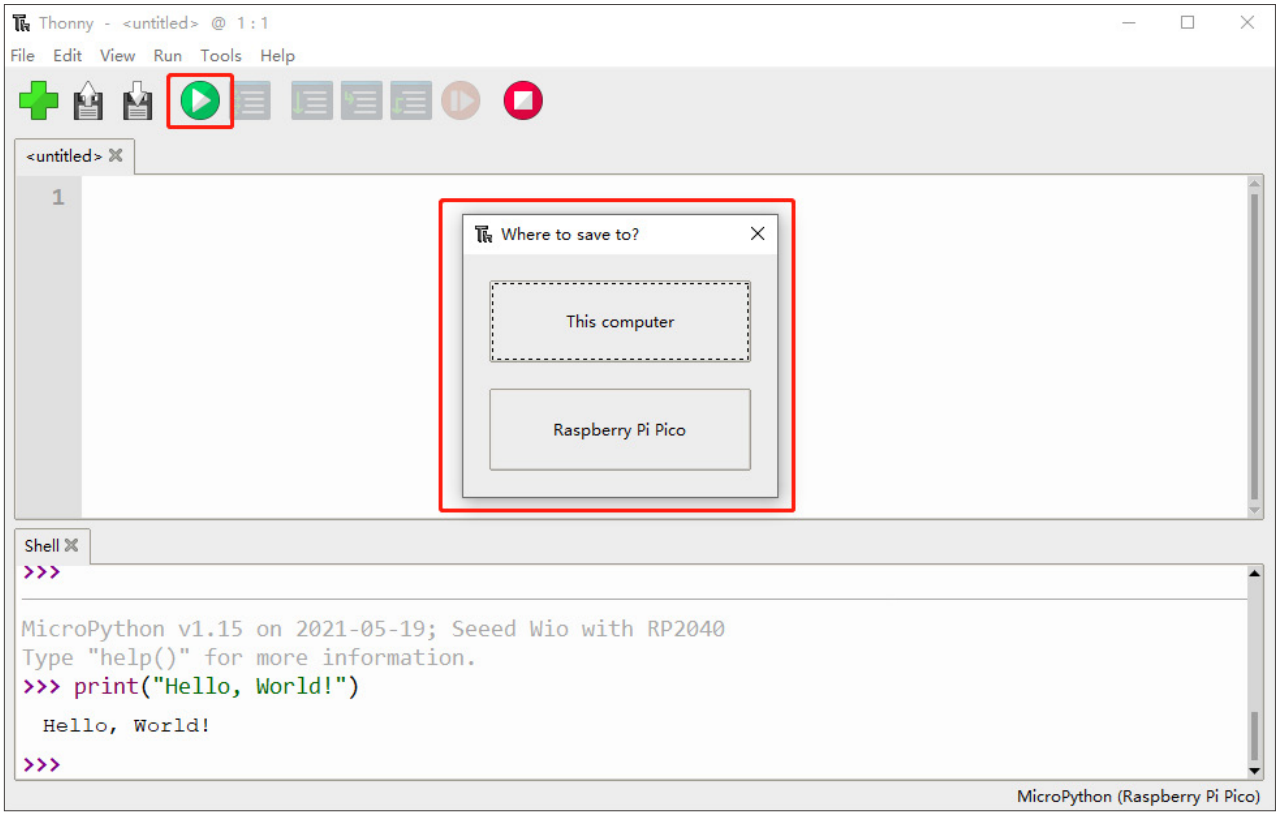
When you press ENTER after you type the instruction, you'll see the execution result of the input instruction is displayed in the Python Shell area, which means, the message "Hello, World!" is printed out in the Python Shell area.



Now let's try to type the same instruction in the script area. When you press ENTER this time, nothing happens — except that a new, blank line is created in the script.

To make your program in the script work, you'll have to click the "Run current script" button in the toolbar.

If you are running a new program that has never been saved, then when you click the "Run current script" button, Thonny automatically shows you a menu with two options — save the program to "This computer" or "Raspberry Pi Pico". You can choose where to store the program according to your own needs. Once your program is saved, you'll see the execution result of the program in the Shell area.

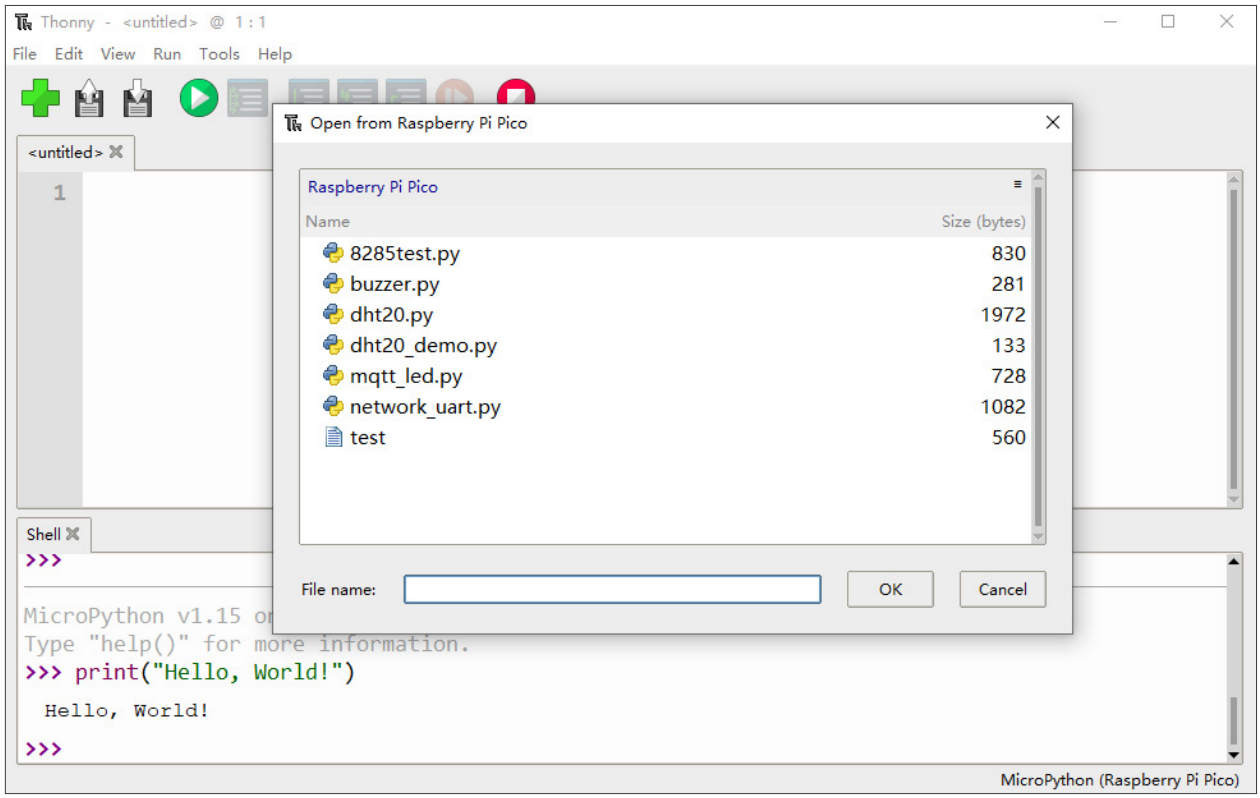


·Note·

Where should I save the program?

To be honest, there's no definitive answer. Where you save your program depends on your needs. If you want to show your program to your friends, you should save it to your Pico and take it with you. If you simply want to save the program for later use, just save it to your computer.

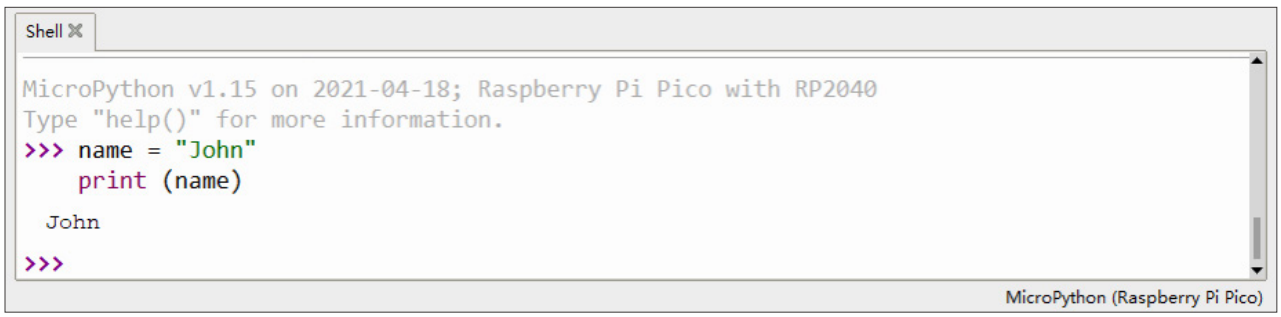
The next time you want to open your saved program, click the "open" button on the toolbar to find the corresponding program.



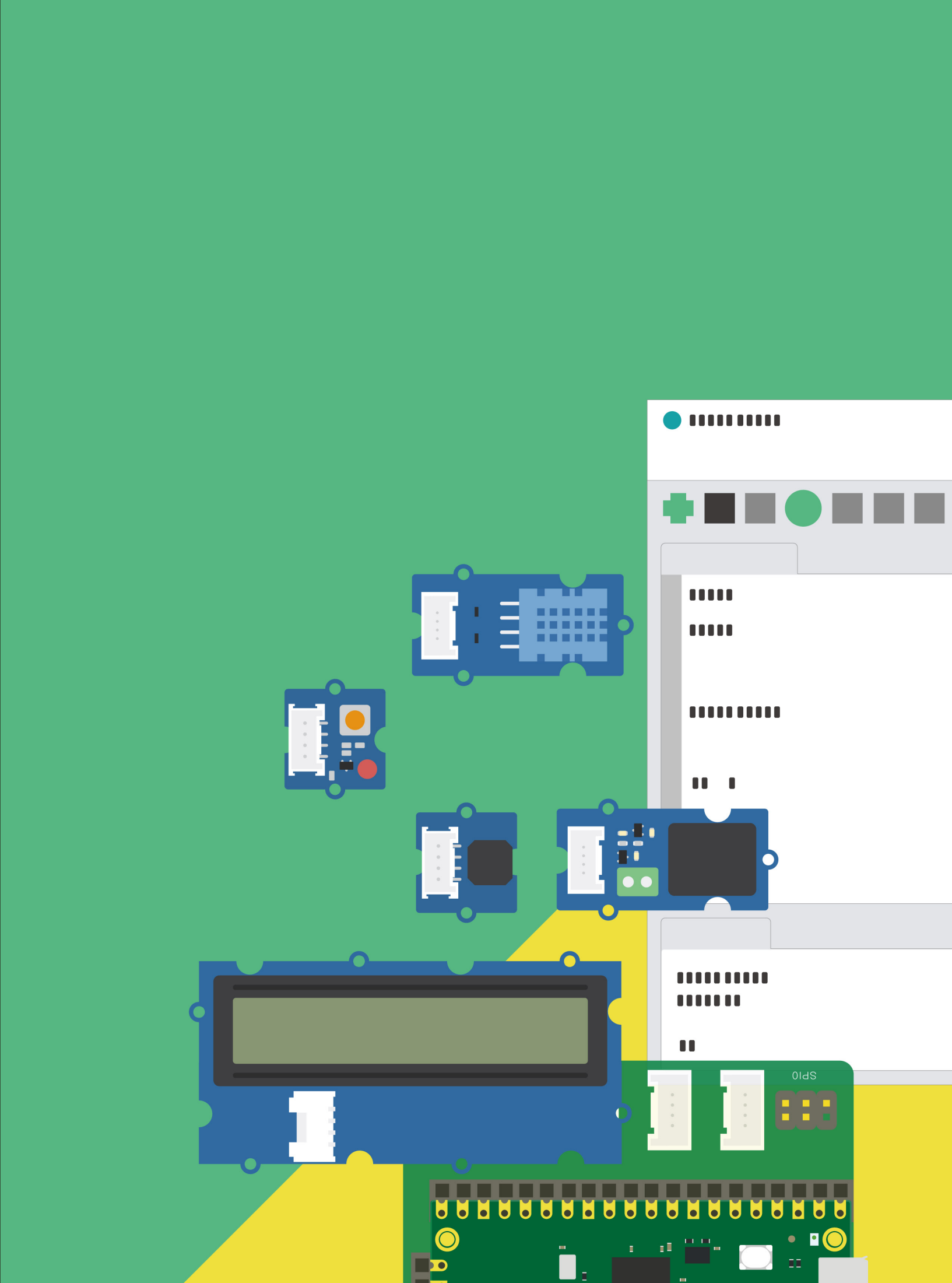
Thinking Expanding

Try using the print() function to print other information, such as your name, age, etc.

```
1 name = "John"
2 print (name)
```



Unit 2
Hardware and Programming


SPI0

Lesson 3 First Project: Light up LED

In the previous lesson, you've learned a little about how to work with Pico and write your very first program.

However, we so far run the program on Pico without using any other external electronic hardware. In this lesson, we will try to attach other electronic hardware to Pico. More specifically, we will use an LED, and write a program to light up that LED. But before you start programming, let's first learn a little more about the program structure.

Knowledge Base

Basic Program Structure

The programs that we write with MicroPython are generally composed of three basic program structures. Complex programs are built with a combination of these basic structures, so it is important to learn them first. They are: sequence structure, loop structure and selection structure.

Sequence Structure

Sequence structure is the most basic program structure. In sequence structure, programs run line-by-line, top-to-bottom, in turn. For example, when the following program runs, it prints"Hello" first, and then "World".

```
1 print("Hello")
2 print("World")
```

Execution result:

```
1 Hello
2 World
```

If you look closely you can find that, unlike other languages where a specific terminator is required to end each line of the statement, you simply need to press ENTER to end a line in MicroPython.

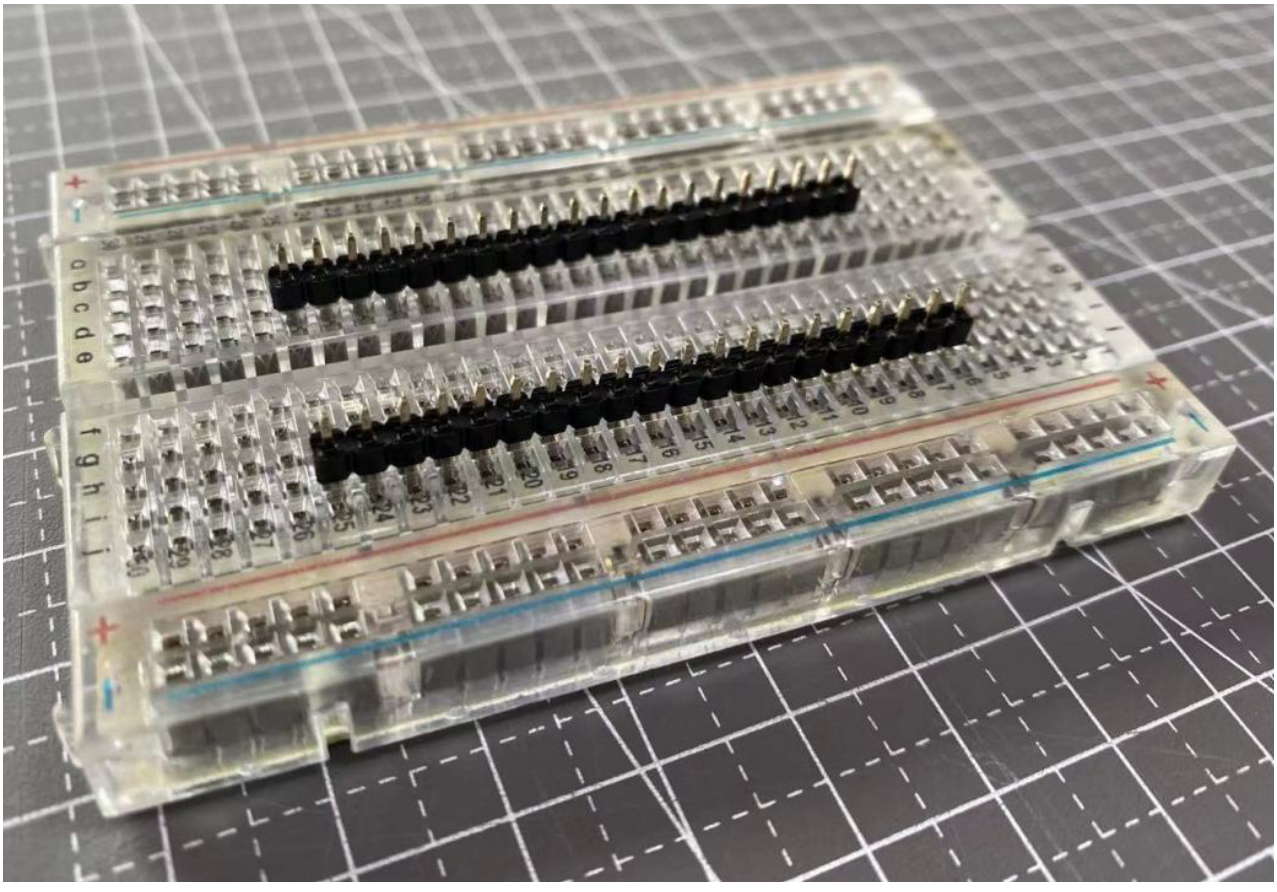
Practice & Operation

Now, let's go back to the project and practice what we've just learned. To get started with this lesson, we will use the Grove Shield for Pi Pico to connect Pico to other Grove electronics for more interesting projects. Upon observation, you might notice that Pico does not come with the metal pins to connect the Grove Shield for Pi Pico. To do so, you need to solder the headers for Pico.

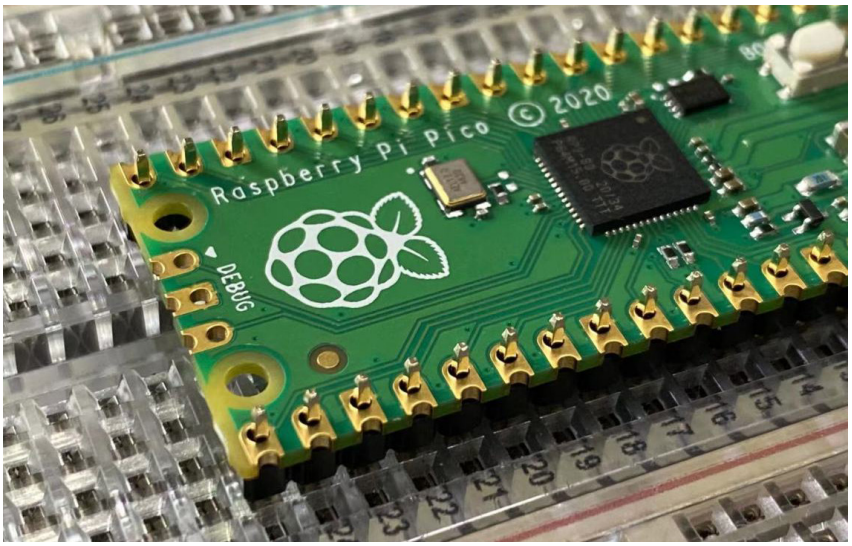
Project 1: Soldering the Headers

First, get everything you need to solder the headers: a soldering iron, some solder, a cleaning sponge, a stand, two 20-pin headers, a breadboard, and of course, your Pico.

For easy soldering, we can use a breadboard to fix the two 20-pin headers first. With the black plastic block on the headers as the boundary, slowly insert the long end of the headers into the breadboard. When inserting, make sure that the two headers are aligned up and down, and spaced the same width as Pico.



Then, turn your Pico right side up and make sure the reserved pinholes on the PCB board are aligned with the headers that are fixed on the breadboard. After alignment, slowly insert the two headers into the reserved hole of Pico, and keep pushing until the black plastic block on the headers are sandwiched between your Pico and your breadboard. At this time, you'll see a small length of each pin sticking up out of the reserved pinholes on the PCB board.

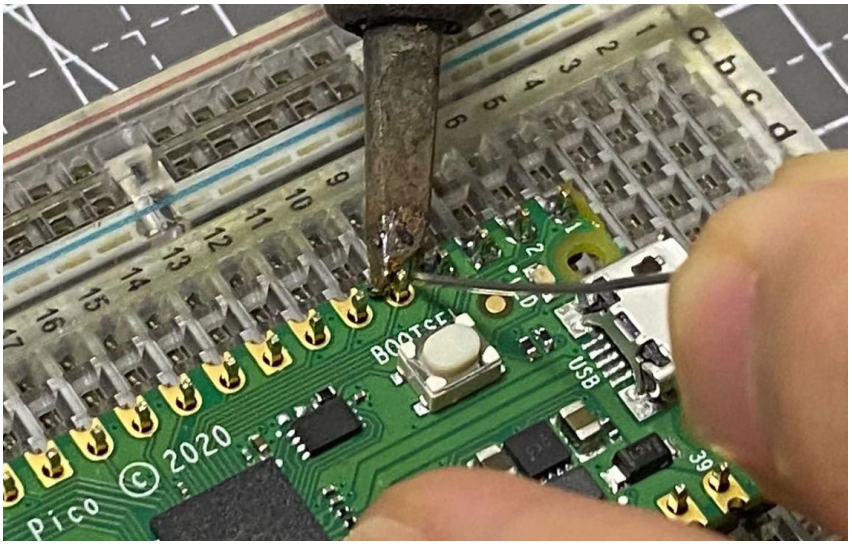


Put your soldering iron in its stand, and turn on the switch to heat it. It will take 3—5 minutes for the tip of the iron to get hot. Make sure the metal tip isn't resting up against anything when heating.During heating, dampen your cleaning sponge first and put it in a convenient place for clean the iron later.

After heating, pick up your soldering iron by the handle, brush the metal tip repeatedly on the sponge you prepared until the tip looks shiny and clean.

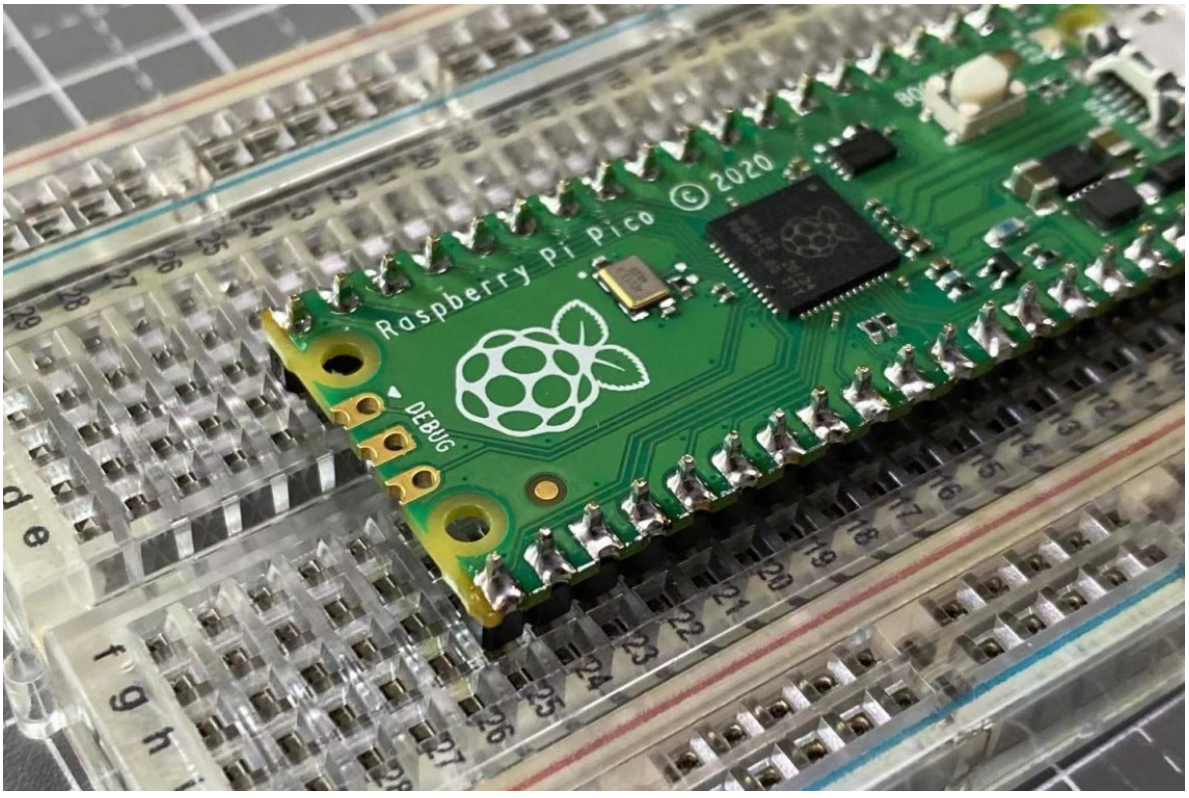
Attention! The metal parts of soldering iron are very, very hot. Under no circumstances should you ever touch the metal parts of a soldering iron.

After cleaning, use the tip of the soldering iron to heat the pin and the gold-colored pad beneath it closest to you. Pick up some solder with the other hand, and slowly push it from the opposite direction of the soldering iron into the joint of the pin and the pad. The heated pin and pad melt the solder and make it flow around the pad. After completing the soldering in one direction, continue to push the solder in other directions until the pad is completely covered by the solder.



Attention! Do not use too much solder when soldering. If the solder overflows to adjacent pads, a short circuit will occur later when using the Pico.

Well, congratulations on your soldering of the first pin! Now, you just need to solder the remaining 39 pins in the same way. When soldering, remember to wipe your iron with a clean sponge from time to time to keep the tip clean and shiny.



After soldering, slowly pull Pico out of the breadboard. If the pins and the breadboard are plugged too tightly, forcibly pulling Pico out will easily cause the pads to fall off. If this happens, you can shake Pico from side to side, and try to move Pico out bit by bit.

All right, we're done! Let's get started on our first task!

Project 2: Light up LED Module

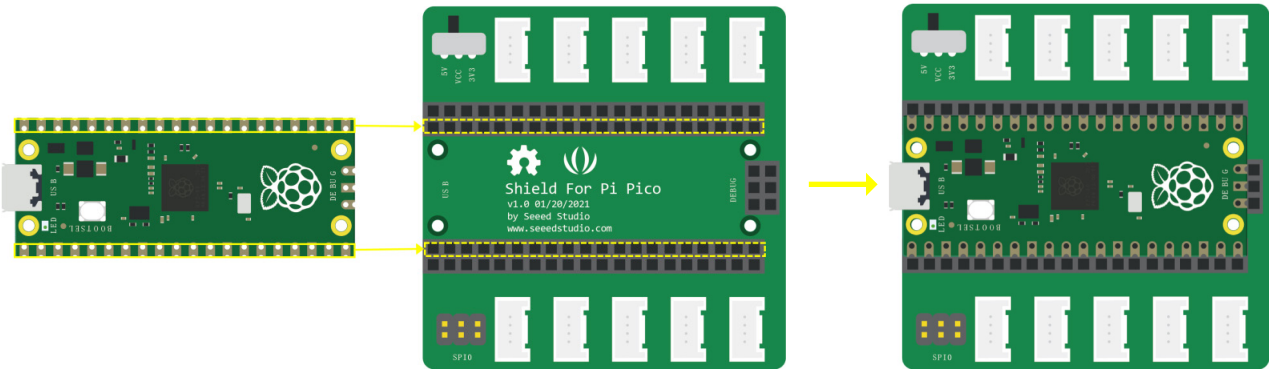
Now that you have completed the soldering of Pico pins, we can finally connect the Grove modules using the Grove Shield for Pi Pico. Let's try to light up an LED first.

Hardware Connection

In this project, we'll use the following electronic hardware:

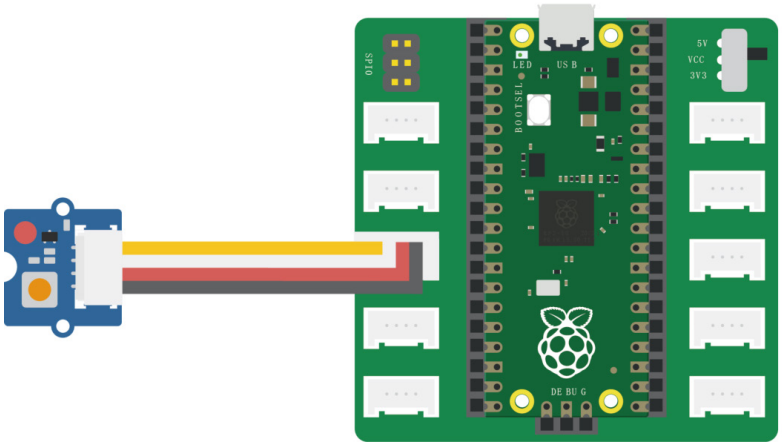
- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – LED Pack

With the help of the Grove Shield for Pi Pico, the work of connecting the electronic hardware becomes very easy. First, we just need to insert the soldered pins of Pico into the Shield:



When inserting, you can observe the screen print of pins on the back of Pico and the screen print on the Shield to check whether your inserting direction is correct.

Then, connect the Grove – LED to the D16 port of the Shield using the Grove cables.



Write a Program

First, use a USB cable to connect Pico and the computer, then open Thonny and click the"new" button in the toolbar to create a new program. Click into the script area, and start your program with the following line:

```
1 import machine
```

This line of code imports a MicroPython function library named "machine". To understand exactly what this line of code does, we need to understand the concept of "library".

·Note·

Function library

Simply put, a library is a collection of programs with related functions written by other senior developers. Libraries are not standalone programs; they are code that provides services to other programs. When we do project development, we can directly import the written functional modules from these libraries to build programs to reduce the development cost and time cost.

In this line of code, we import a MicroPython library named "machine", which contains various functions related to specific hardware. It can directly access the hardware functions of the system (such as CPU, timer, bus, etc.) without restriction, so that we can use MicroPython to control other electronic hardware connected to Pico more efficiently. This includes setting pins for various electronic hardware connected to Pico.

In the development of hardware projects, it is not sufficient to simply connect the hardware to the Pico. For example, we have connected LED to D16 of the Shield, but Pico would not be aware of this unless we specifically informed it — you have to write a program to define the pins that control the electronic hardware.In the machine library, there is a class of functions called "Pin".

·Note·

Pin class

We can use the Pin class function to set the pins of electronic hardware. In the pin class, there are functions for setting pin mode (in, out, etc.), and functions for setting digital logic. Among them, the common functions are:

- Create Pin

Pin(num,Pin.OUT/IN)

"num" represents pin number; "IN/OUT" represents input and output respectively.

- Write Pin value

p0.value(0)/p0.value(1)

- Get Pin value

p0.value()

- Set a pin with pull-up resistor

p2 = Pin(num, Pin.IN, Pin.PULL_UP)

With the help of the machine library, you can easily define the pin of the LED in the next line of the program.

```
1 LED = machine.Pin(16,machine.Pin.OUT)
```

In this line of code, we created an object called "LED" to help us control the LED, and used the Pin function in the machine library to define the pin number and mode for the LED. The pin function has two parameters in total. The first parameter 16 represents the pin number you defined. Because we connected the LED light to D16 when building the project hardware, we set it to 16 here. The second parameter, machine.Pin.OUT, tells Pico the pin should be used as an output rather than an input.

OK, now type the last line:

```
1 LED.value(1)
```

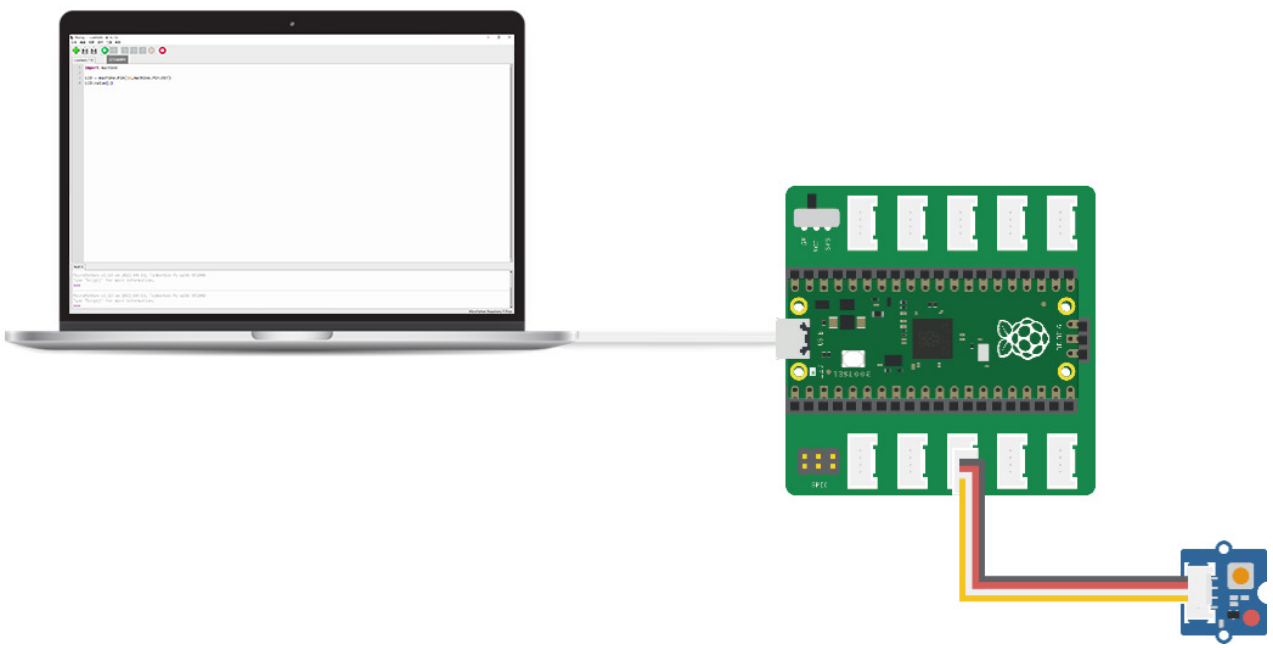
In this line of code, we use the value function to write the value for the pin we just defined to turn on the light. When controlling pins of Pin class, we usually use a value of 1 to assign a high level (on) and a value of 0 to assign a low level (off).

Our first hardware program is finished. The complete program code is as follows:

led.py

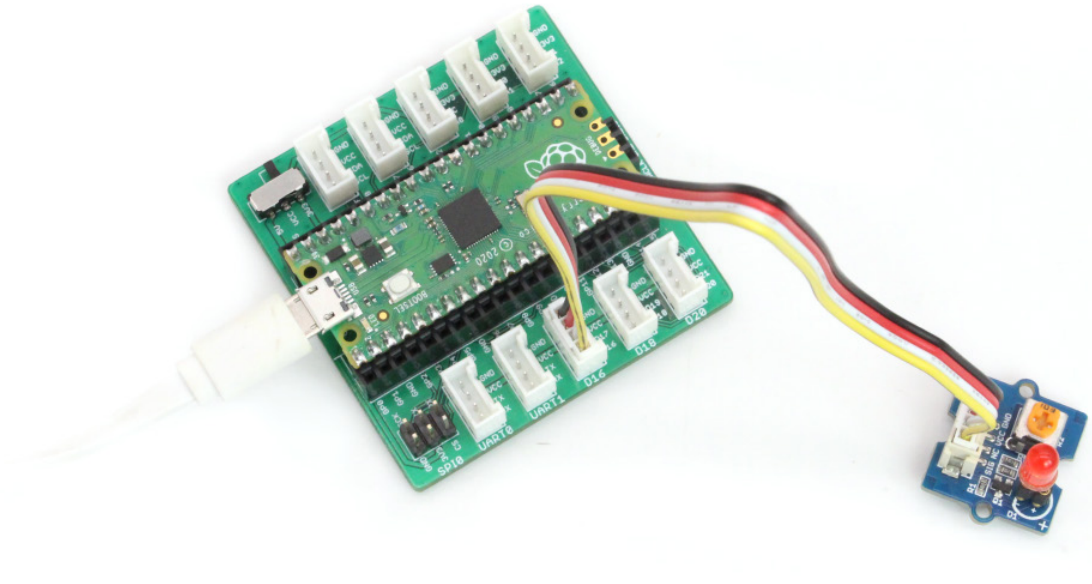
```
1 import machine
2 LED = machine.Pin(16,machine.Pin.OUT)
3 LED.value(1)
```

After we finish the program, use a USB cable to connect Pico to the computer, as shown in the following figure. In the following lessons, when we want to run a program, we always need to connect Pico and the computer with a USB cable.



Click the "run" buttonin  the toolbar, save the program to any location, and you can see the LED plugged into D16 is lit up.

If you want to turn off the LED light, simply change the value from 1 to 0.



Thinking Expanding

Try to turn off the LED using the program.

```
1 import machine
2 LED = machine.Pin(16,machine.Pin.OUT)
3 LED.value(0)
```

led2.py

Lesson 4 Hello World in Electronic Hardware Programming: Blink

When we try software programming with a certain language, "Hello World" is often the first program we write. Blink, which is the task of flashing an LED, is the Hello World in the electronic hardware world. In this lesson, we will try to write a Blink program. To do this, we will need to use another of the three basic programming structures — the loop structure!

Knowledge Base

Loop Structure

Unlike the sequence structure, a program with a loop structure repeatedly executes one or more instructions. According to the number of repeated execution, the loop structure can be subdivided into definite loop structure and indefinite loop structure. When a program with definite loop structure is executed, it only repeats a limited number of times; when it meets a certain condition, the loop will automatically terminate. However, a program with indefinite loop structure will continue to repeat the loop without stopping. In Python, we often use the for–loop and the while–loop.

while–loop

Let's first take a look at a program written with while–loop statement:

```
1 a = 0
2 print("loop start")
3 while a < 5:
4     print(a)
5     a = a + 1
6 print("loop end")
```

The program is first executed in the sequence structure. At the beginning of the program, we first declare a variable "a" and print a line of characters "loop start" with print().

```
1 a = 0
2 print("loop start")
```

·Note·

What is a variable?

Variables are values that have no fixed values and can be changed at any time. We use variables to store data so that they can be called in later portions of our code. In a program, a variable generally has two aspects: variable name and variable value. The variable name is the identity of the variable, whereas the variable value is the value stored in the variable. For example, in the above code, "a" is the variable name whereas 0 is the variable value. When multiple variables are used in a programme, unique names are used to distinguish them.

Variables in Python do not need to be declared. Each variable must be assigned before use, and will be created at the same time as when it is first assigned.

Next, we use the while statement to create a definite loop structure. This specifies that when the variable "a" is less than 5, the program should repeatedly execute the instructions contained in the while statement. That is, it should print out the value of "a", and add one to "a", until "a" is greater than or equal to 5.

```
while a < 5:
    print(a)
    a = a + 1
```

The output of the program is as follows:

```
loop start
0
1
2
3
4
loop end
```

Looking at the execution results of the sample program, we can find that the program does not start to execute the last print() statement until it has executed several loops within the while statement.

But, how does the computer know which statements need to be repeated in the while loop statement and which statements are outside the while loop statement? MicroPython uses indentation and colon ":" to distinguish the levels between code blocks, unlike other programming languages (such as Java and C) which use braces "{}" to separate code blocks. This change makes code written in MicroPython clear and easy to understand.

·Note·

Python is very strict on code indentation. The same level of code block indentation must be consistent. In general, we can use BACKSPACE or TAB to complete the indentation. However, whether we manually type multiple spaces or use TAB to indent, we usually define the length on an indentation as four spaces (by default, one tab is equal to four spaces).

for–loop

A for–loop is generally used to traverse all elements in a sequence, such as:

```
1 for i in range(10):
2     print(i)
```

The output of the program is as follows:

```
1 0
2 1
3 2
4 3
5 4
6 5
7 6
8 7
9 8
10 9
```

Where, range() is a built–in function of MicroPython, which can generate a list of integers. Generally, we use this function in for–loop. For example, range (10) creates a list of nine integers from 0 to 9.

In this program, we will use the for–loop to traverse and use print() to output all the integers to the Shell.

Practice & Operation

Project 1: Control LED On and Off with For–loop

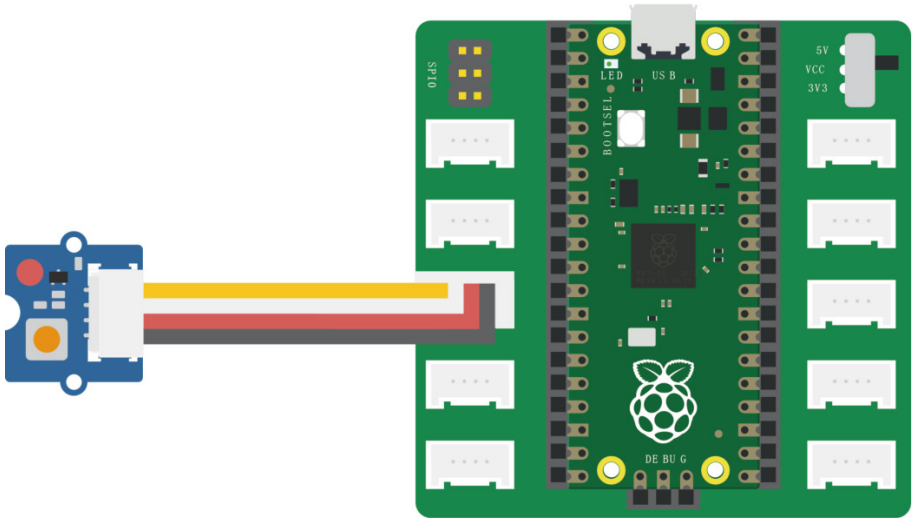
In this project, we will control the LED on and off with a for–loop to achieve flashing.

Hardware Connection

In this project, we will use the following electronics hardware:

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – LED Pack

Similar to before, we connect the LED to D16.



In the last lesson, we have learned how to control the LED on and off. To realize the final Blink program, we only need to make some slight modifications.

Write a Program

First, let's try to modify the program like this:

```
1 import machine
2 LED = machine.Pin(16,machine.Pin.OUT)
3 LED.value(1)
4 LED.value(0)
```

Click the "run" button and look at the LED. You will find that the LED just flashes only once very quickly. This is because:

1. The execution speed of the program is so fast that the flash effect is not obvious.

2. We did not set the loop structure for the program, and the flashing part of the program was executed only once.

Let's solve the first problem. Since the problem is that the program runs too fast, we can set a delay between each LED on and off.

At the beginning of the program, we introduce a new function library: utime.

·Note·

Utime Function Library

The utime function library is one of the standard libraries of MicroPython, and its functions provide us with time-related functions such as obtaining time and date, measuring time interval, setting delay and so on.

The commonly used functions are:

- Declare the library used — import utime
- Sleep for the given number of seconds — utime.sleep(seconds)
- Sleep for the given number of milliseconds — utime.sleep_ms(ms)
- Convert the time expressed in seconds into a date-time tuple — utime.localtime([secs])

```
import machine
import utime
```

Next, we use the sleep function in utime to add a delay after each program that operates the LED.

```
LED.value(1)
utime.sleep(1)
LED.value(0)
utime.sleep(1)
```

The default unit of the sleep function is in seconds unless otherwise specified. By modifying the program, we have set the time for each light on and off to 1 second.

Let's first try to use the for-loop to simply turn the LED on and off. The program is as follows:

```
import machine
import utime

LED = machine.Pin(16,machine.Pin.OUT)
for i in range(10):
    LED.value(1)
    utime.sleep(1)
    LED.value(0)
    utime.sleep(1)
```

for_blink.py

When we execute this program, we can find that the LED stops flashing after ten loops. This is because there are only 10 integers generated by using the range() function, so the for–loop only loops 10 times. When we change range(10) to range(20), the LED loops 20 times.

Of course, no matter how large we set this value to, the loop is always a definite loop. If we want to keep the program running, we need to use the indefinite loop structure to write the program.

Project 2: Realize Blink with While–loop

In this project, we will use the while–loop to achieve the effect of a continuously flashing LED.

Write a Program

Using the while loop, we can easily make the program repeat indefinitely. We only need to change the "for i in range(10)" in the program to "while True":

```
while True:
    LED.value(1)
    utime.sleep(1)
    LED.value(0)
    utime.sleep(1)
```

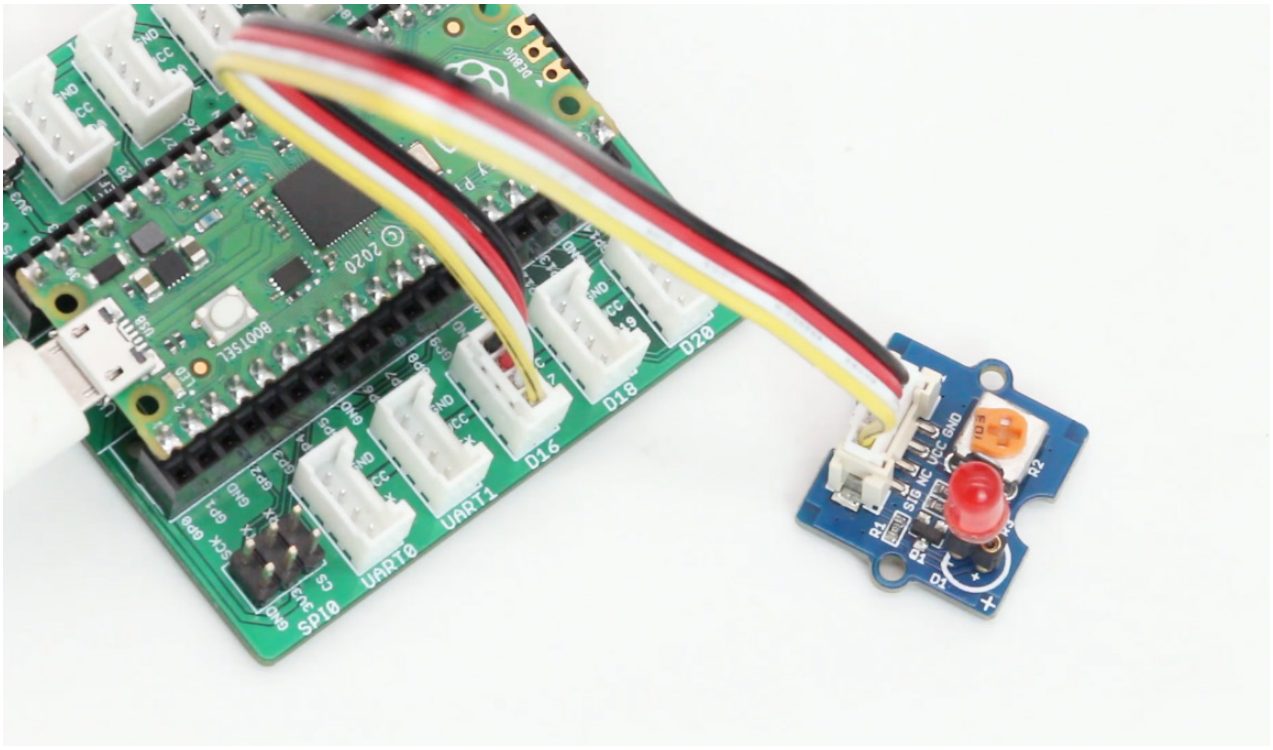
The "while True" is a use of while–loop statement. Unlike a general while–loop that only executes the loop when a certain condition is met, this "while True" is an indefinite loop statement, which means that the program will be executed repeatedly and continuously until it is artificially terminated.

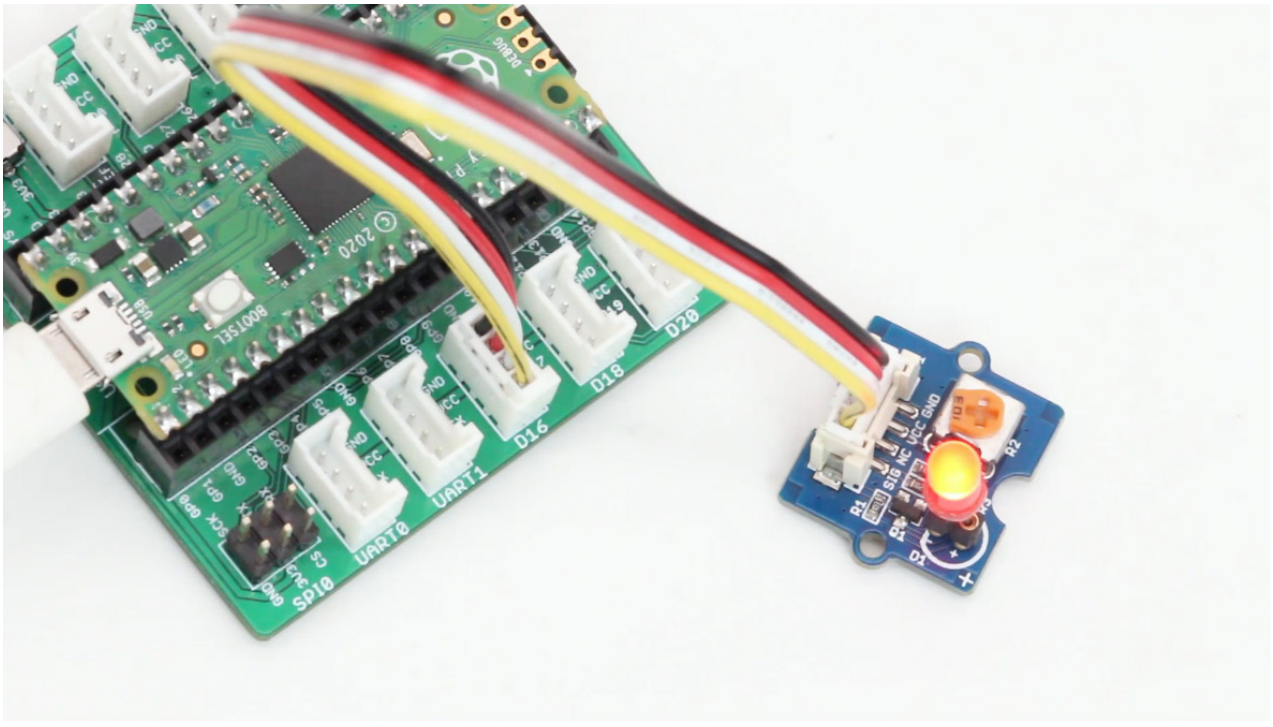
The complete code is as follows:

```
import machine
import utime
LED = machine.Pin(16,machine.Pin.OUT)

while True:
    LED.value(1)
    utime.sleep(1)
    LED.value(0)
    utime.sleep(1)
```

while_blink.py

Click the "run" button again, and the LED connected to the Shield starts flashing. Unless the process is artificially stopped, the LED will continue to flash.





⭐ Thinking Expanding

Try to use the two loop structures and sleep() function to achieve different lighting effects. For example, you can reduce the delay time to produce more rapid flashing.

```
import machine
import utime

LED = machine.Pin(16,machine.Pin.OUT)

while True:
    LED.value(1)
    utime.sleep(0.1)
    LED.value(0)
    utime.sleep(0.1)
    LED.value(0)
    utime.sleep(0.1)
```

more_blink.py

Lesson 5 Digital Signal in Circuit — Playing with LED

In the last lesson, we learned the loop structure, and put it to practice by flashing an LED continuously. In reality, it is impossible to rewrite the program to control the LED every time we need to turn it on or off. Most of the time, we use some external hardware devices to control it on and off — for example, a simple button. In this lesson, we will learn how to use a button to control the LED.

Knowledge Base

Whether it is to use the button to control the LED on or off, or use Pico to control other types of electronic modules, the key lies in the signal transmission. In an electronic circuit, we generally divide signals into two types: digital signal and analog signal.

Digital Signal

A digital signal is a signal expressed in the binary form such as 0 and 1. The use of binary form to express the signal signifies the presence of only two states in the circuit, that is, on and off.

For Pico, many pins can be used for digital signal input and output. When using these pins to transmit digital signals, we only need to configure them as different types according to our needs. In fact, in the last lesson, we have configured and used digital pins many times. For example:

```
1 import machine
2 LED = machine.Pin(16,machine.Pin.OUT)
3 LED.value(1)
```

This program has set Pin 16 as the output mode and written a high level to the pin to light the LED.

Selection Structure

In previous lessons, we have learned sequence structure and loop structure in the three basic program structures. The final one is selection structure. The selection structure makes a judgement according to the defined conditions and executes the corresponding operation according to the result of the judgment. For example, after judging the weather, we choose to

take an umbrella if it's raining, and vice versa. In Python, you can use the "if... else" statement to write such evaluative condition in the following three forms.

"if" statement

This is the simplest conditional judgment. If the expression is true, the code block will be executed. If the expression is not true, nothing will be done. For example, to take an umbrella if it's raining, but do nothing if not.

```
if expression:
    code block
```

"if……else" statement

If the expression is true, code block 1 will be executed. If the expression is not true, code block 2 will be executed instead. For example, to take an umbrella if it's raining, and not take an umbrella otherwise.

```
if expression:
    code block 1
else:
    code block 2
```

"if……elif……else" statement

This statement is used to judge a variety of conditions. Python will judge whether an expression is true one by one from top to bottom. Once an expression is true, the following code block will be executed. At this point, the remaining conditions will no longer be evaluated and thus not executed, regardless of whether they would have been true. If none of the expressions is true, the code block following "else" will be executed.

```
if expression 1:
    code block 1
elif expression 2:
    code block 2
elif expression 3:
    code block 3
...//other elif statements
else:
    code block n
```

Practice & Operation

Next, let's go back to the project to practice what we just learned. In the first project, let's take a look at how to use a button to control the LED on and off.

Project 1: Control LED On and Off with Button

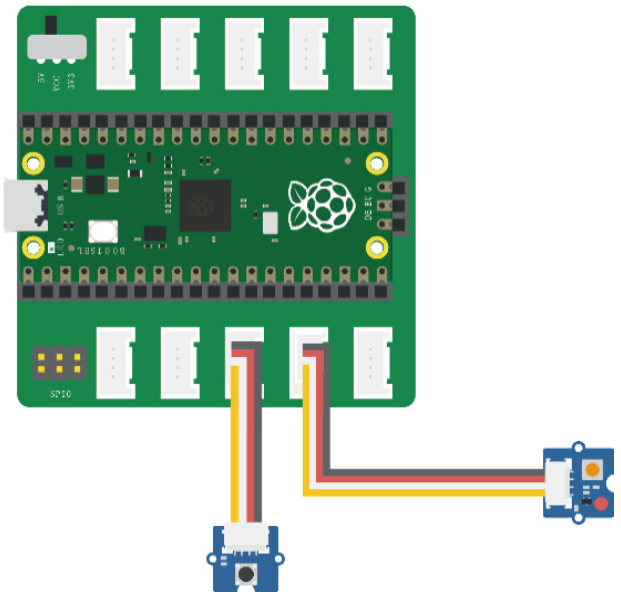
In this project, we use a button to turn the LED on and off. When the button is pressed, the LED light is switched on. When the button is released, the LED light is switched off.

Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – LED Pack
- Grove – Button

The Grove – Button is a simple module. When the button is pressed, its return value is 1. When the button is released, its return value is 0.Connect Pico and the Shield. Use Grove data cables to connect the button to D16, and the LED to D18.



Write a Program

At the beginning of the program, we need to import the machine library and use the Pin function in the library to set the pins for the two electronic modules.

```
1 import machine
2 BUTTON = machine.Pin(16,machine.Pin.IN)
3 LED = machine.Pin(18,machine.Pin.OUT)
```

In addition to the previous program, when we introduce the function in the library, apart from using "Pin.OUT" to set Pin 18 as the output pin, we also need to use "Pin.IN" to set Pin 16 as the input pin to receive the data returned by the button.

After setting the pins, we can use the following code to obtain the value returned by the button:

```
1 val = BUTTON.value()
```

In this line of code, we use the value() function to read the return value of the button. If you still remember, we have also used the value() function to write the value for the LED in the previous program.The difference between them is that when a value is written in brackets of the value() function, it will output the value to the pin, for example:

```
LED.value(1)
```

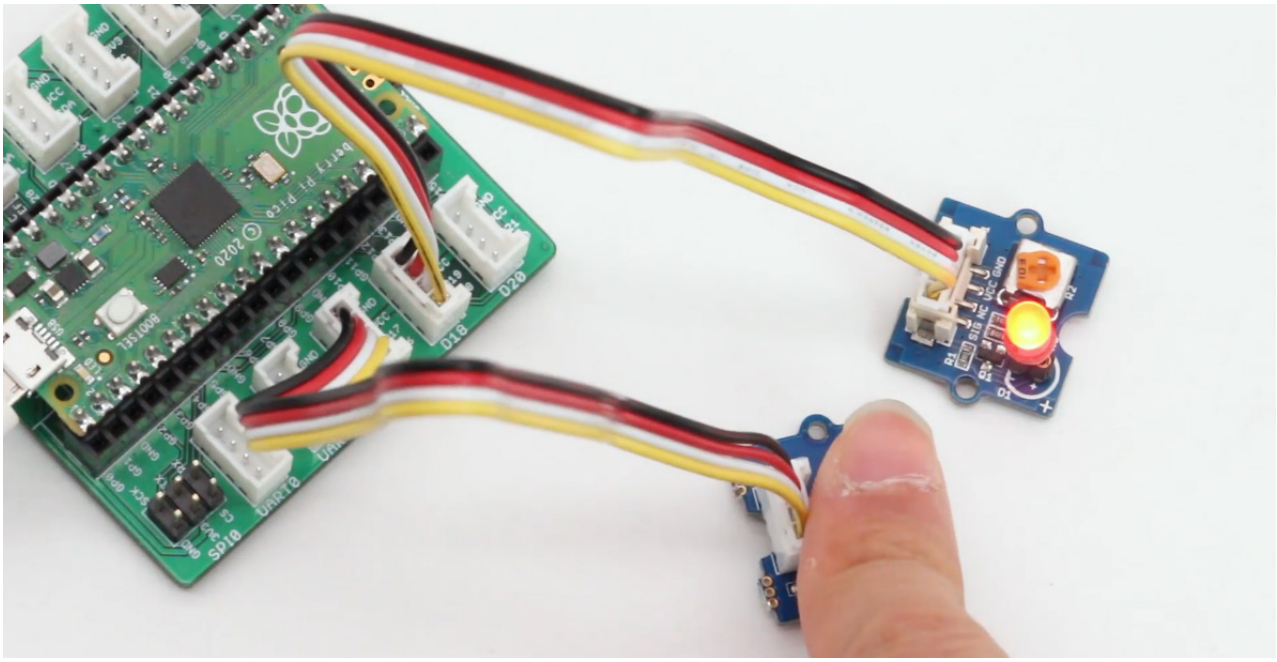
And when there is no value in brackets, it will read the input value of the pin.

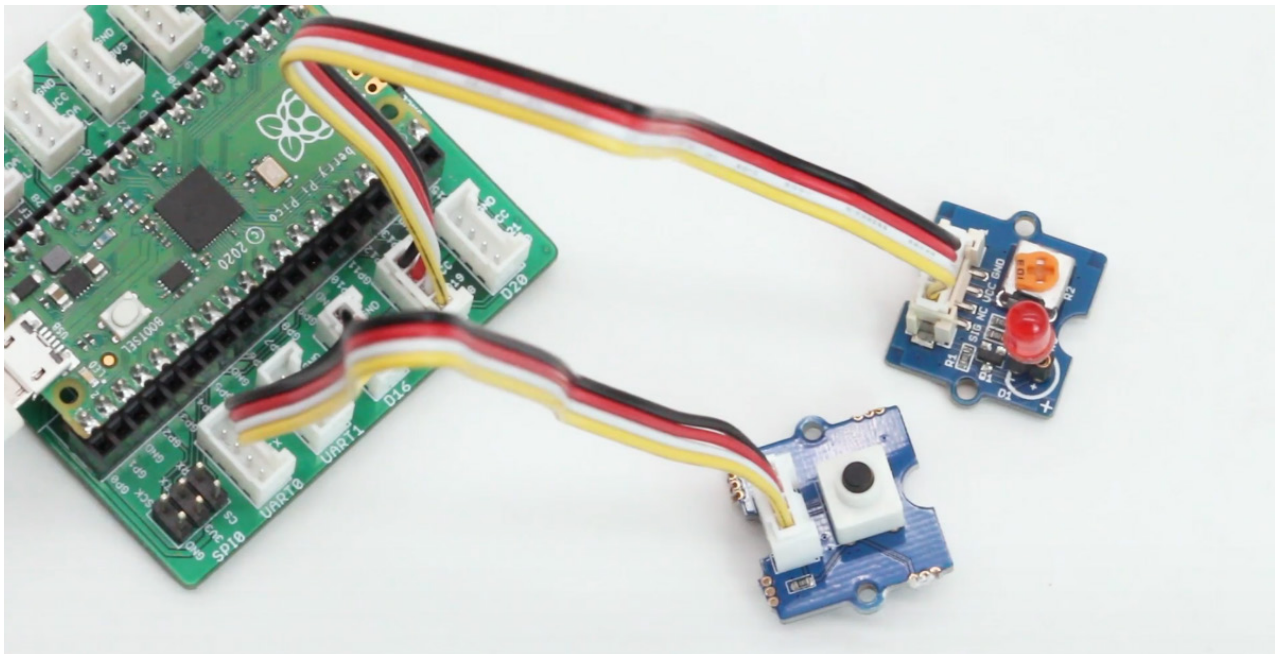
Finally, we use the conditional statement "if" to evaluate the following condition. When the button is pressed, the return value is 1 and the LED is switched on; otherwise, the LED remains off:

```
while True:
    val = BUTTON.value()
    if val == 1:
        LED.value(1)
    else:
        LED.value(0)
```

button_led.py

Use a USB cable to connect Pico to the computer, click the "run" button to save the program to any location. Now, press the Grove button to see the program in action!





Project 2: Button Light

By observing the implementation of the program in Project 1, we find that the LED only lights up when the button is pressed. When we release it, the LED will turn off immediately. This is not the same as a typical button light in our daily life, which turns on when pressed, and off when pressed again. Let's now try to achieve this.

Write a Program

First, import the required machine library and utime library, and configure the pins for the button and the LED.

```
1 import machine
2 import utime
3
4 BUTTON = machine.Pin(16,machine.Pin.IN)
5 LED = machine.Pin(18,machine.Pin.OUT)
```

Previously, we directly used the value returned by the button to control the LED, and the button returns 1 only when it is pressed and returns 0 when it is released — producing the outcome that we observed.Therefore, we cannot directly use the button to control the LED, but will instead need to create a variable to indirectly control the LED state.

First, we need to create a new variable in the program and set its initial value to 0

```
1 val = 0
```

Then, write a program to run in an indefinite "while True" loop. Each time the button is pressed, the value of this variable will be increased by 1:

```
while True:
    if BUTTON.value() == 1:
        val = val+1
        sleep(1)
```

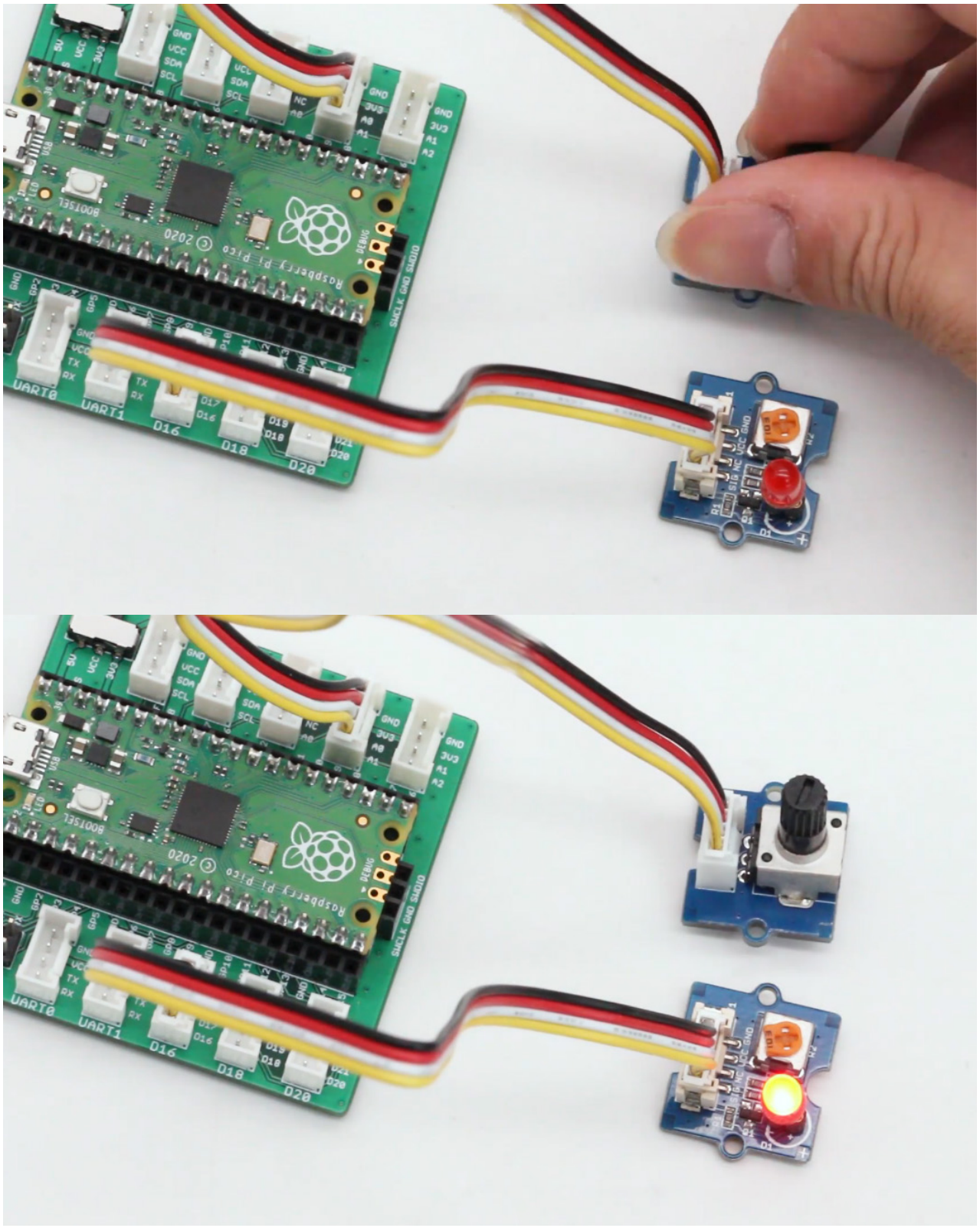
We just need to switch the value of the variable between 0 and 1. Thus, when the value of the variable is increased to 2, we will reset it to 0. Here, we can use "elif" to add the additional condition to the "if" statement:

```
while True:
    if BUTTON.value() == 1:
        val = val+1
        sleep(1)
    elif val == 2:
        val = 0
        sleep(1)
```

In this way, the value of the variable "val" that we set can be switched between 0 and 1 every time we press the button. Finally, we can use this variable to control the LED. The complete program is as follows:

```
import machine
import utime

BUTTON = machine.Pin(16,machine.Pin.IN)
LED = machine.Pin(18,machine.Pin.OUT)

val = 0

while True:
    if BUTTON.value() == 1:
        val = val+1
        utime.sleep(1)
    elif val == 2:
        val = 0
        utime.sleep(1)
    LED.value(val)
```

button_led2.py



Thinking Expanding

When you write the program, you will find that every time you use a function from a library, you need to define the function fully as referenced from the machine library. It is a bit inconvenient to write the program:

```
import machine
import utime

BUTTON = machine.Pin(16,machine.Pin.IN)
LED = machine.Pin(18,machine.Pin.OUT)

utime.sleep(1)
```

Here, we can write the program in a more concise way:

```
from machine import Pin
from utime import sleep

BUTTON = Pin(16,Pin.IN)
PWM_LED = PWM(Pin(18))

sleep(1)
```

Unlike “import”, which imports an entire library, “from...import” only imports some functions in the library. After we use ”from...import” to import the class of the function we want to use, we can use the function directly in the later program without having to continuously declare which library the function comes.

Lesson 6 Analog Signal in Circuit (1) — Rotary Angle Sensor

In the last lesson, we learned to use digital signals to control LED lights (button control). In this lesson, we will learn to use another signal to control a device: the analog signal.

Knowledge Base

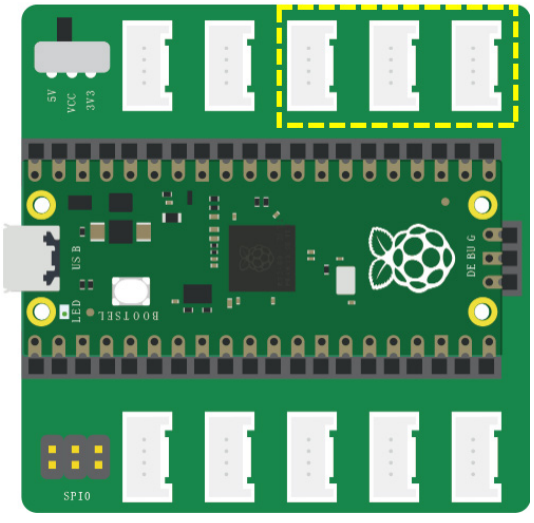
Analog Signal

Analog signal refers to the electrical information represented by continuous physical quantities. The amplitude and frequency of the signal change continuously with the change of time. In electronics, we use sensors to measure the data of light, humidity, temperature and other physical quantities, obtain the numerical changes over a period of time, and use electrical signals to simulate its numerical changes. For example, when a sound signal is received by a microphone or other sensor and converted into an electrical signal, the voltage directly reflects the volume, and the frequency (tone) of the sound is directly converted into the frequency of the electrical signal. This is the meaning of the word "analog" in the analog signal.

Of course, an analog signal also has the difference between input and output, but unlike a digital signal, the input analog signal can not be directly used by Pico.

ADC

As a microcontroller, Pico's chip is composed of thousands of transistors like other microcontrollers. These transistors can only switch between two states: on (1) or off (0) — just like a digital signal, which enables Pico to process digital signals directly. For the analog signals which cannot be described with 0s and 1s, the analog signals must be converted into digital signals before they can be recognized by Pico. The electronic component for this operation is called "ADC (Analog to Digital Converter)". On Pico, there are three ADC channels that can be used. They are located on GP26, GP27 and GP28. When you plug Pico into the Shield, these pins are then connected to the A0, A1 and A2 of the Shield. So when we use sensors that need to utilise the ADC channel to return analog signals, we can only plug them into the A0, A1 or A2 ports of the Shield.

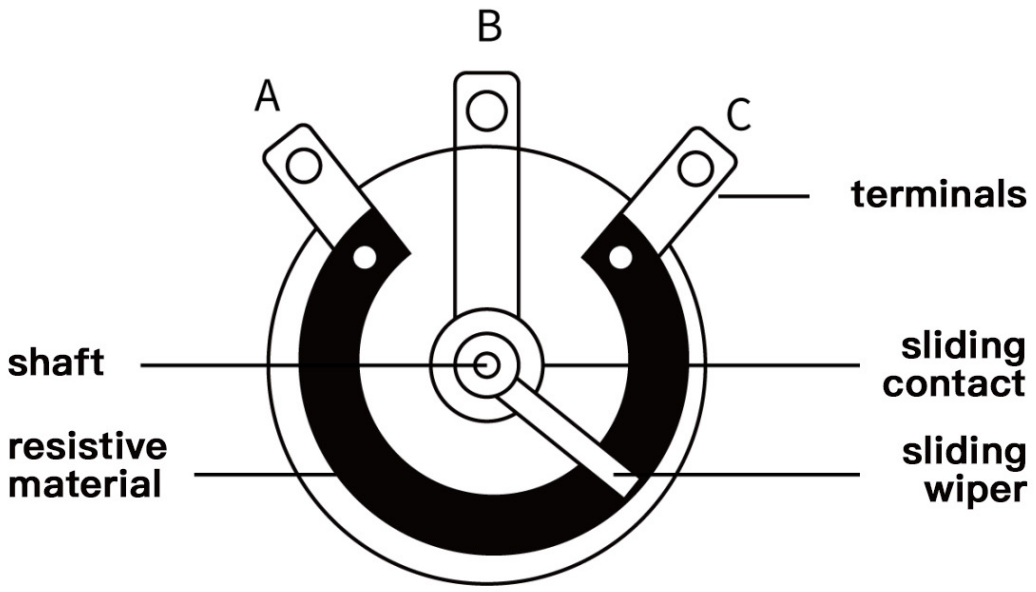


In this lesson, we will first use a Rotary Angle Sensor that inputs analog signals to Pico, to help you understand the input of analog signals.

Rotary Angle Sensor

By definition, a rotary potentiometer refers to a potentiometer that can change its resistance value according to the rotation of the adjusting knob on the potentiometer. This method of adjusting resistance can effectively reduce the size of the potentiometer so that it can be used in many commercial electronic products. Therefore, although rotary potentiometers sound foreign, many products in our life that we are familiar with are actually implemented with rotary potentiometers, such as volume adjustment knobs on audio, on various industrial equipment consoles, etc. So how does a rotary potentiometer work?

The following figure shows the internal structure of an ordinary rotary potentiometer:



When you rotate the knob, the sliding wiper connected to the knob slides on the resistive material with the rotation. With the sliding, the length of resistive material between A—B and B—C changes, which causes the resistance value and thus the output voltage to change accordingly. Using the Rotary Angle Sensor, we can easily input analog signals of different values to Pico. However, to make use of these input analog signals to complete the project, we need to learn how to use comparison operators to process them.

Operator

Operators are symbols used to perform code operations in various expressions of programs, and operands are the objects of operators. For example, 2 > 3, where the operands are 2 and 3 and the operator is ">". Depending on different functions, operators can be divided into many kinds, such as arithmetic operators, comparison operators, etc. In this lesson, we will first learn about comparison and logical operators.

Comparison Operator

Comparison operators, also known as relational operators, are used to compare the values of constants, variables, or expressions. If the comparison is true, it returns True; otherwise, it returns False.

In fact, you would have seen them in various conditional judgment statements in the projects of previous lessons. For example, when we programmed a button to control the state of an LED in the last lesson, we used the comparison operator "==" to evaluate whether the return value of the button is 1:

```
while True:
    val = BUTTON.value()
    if val == 1:
        LED.value(1)
    else:
        LED.value(0)
```

Next, I will list all the comparison operators in Python and their usage, where a = 1, B = 2:

Operator	Description	Example
==	equal — compare whether objects are equal	(a == b) return False
!=	not equal — compare whether two objects are not equal	(a != b) return True
>	greater than — return whether x is greater than y	(a > b) return False
<	less than — return whether x is less than y	(a < b) return True
>=	greater than or equal to — return whether x is greater than or equal to y	(a >= b) return False
<=	less than or equal to — return whether x is less than or equal to y	(a <= b) return True

Logical operators

Logical operators can concatenate two or more simple statements to form even more complex statements. We often use them in all kinds of conditional statements.

The common logical operators are as follows:

Operator	Logical expression	Description
and	x and y	Boolean "and" — if x is False, it returns False; otherwise it returns the computed value of y.

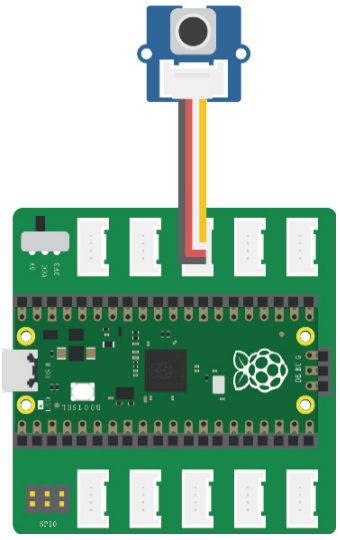
or	x or y	Boolean "or" — if x is not 0, it returns the value of x; otherwise it returns the computed value of y.
not	not x	Boolean "not" — if x is True, it returns False; if x is False, it returns True.

Practice & Operation

Next, we will use the Rotary Angle Sensor to input analog signal to Pico to control the LED. But before that, we need to understand the range of values for the analog signal returned by the Rotary Angle Sensor.

Project 1: Check Return Value of Rotary Angle Sensor

This project displays the analog value returned by the Rotary Angle Sensor in the Shell.



Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – Rotary Angle Sensor

Plug the Pico and the Shield, and connect the Rotary Angle Sensor to A0.

Write a Program

First, import the functions we need: the ADC function in the machine library and the sleep() function in the utime library:

```
1 from machine import ADC
2 from utime import sleep
```

Using the “from... import” we learned in the last lesson to import functions can make our program more concise.

Then, set and define the pins. It is very easy to set the pins by using the ADC function, which can be written as follows:

```
1 ROTARY_ANGLE_SENSOR = ADC(0)
```

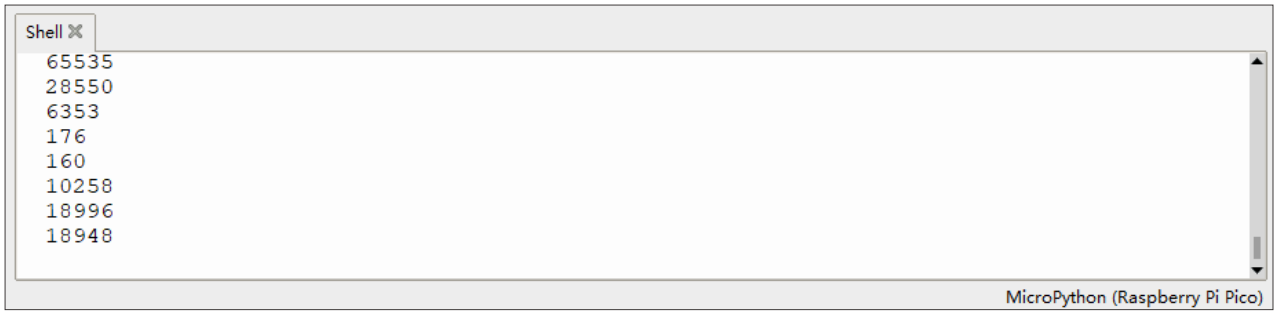
ADC(0) means setting A0 as the ADC pin, while ADC(1) and ADC(2) means setting A1 and A2 as the ADC pin.

Then set up a loop, using "read_u16()" which reads the return value of the pin and prints it out. The complete program is as follows:

```
1 from machine import ADC
2 from utime import sleep
3
4 ROTARY_ANGLE_SENSOR = ADC(0)
5
6 while True:
7     print(ROTARY_ANGLE_SENSOR.read_u16())
8     sleep(1)
```

konb.py

Use a USB cable to connect Pico to the computer, click the "run" button to save the program to any location, you can see the analog value returned by Rotary Angle Sensor in the Shell area.



By rotating the Rotary Angle Sensor, we can find that the value will change with the rotation of the knob, and vary within the range of 0—65535. But no matter how you rotate the knob, the return value will not be equal to 0. This is because the maximum resistance of the Rotary Angle Sensor is limited in reality, making it virtually impossible to reduce voltage to 0.

·Note·

Why 65535?

In the program, we use "read_u16()" to read the analog value returned by the pin, where "u16" means to read the analog value as an unsigned binary integer of up to 16 bits. The maximum value of the 16-bit binary integer is 1111111111111111, which is equal to 65535 after being converted into decimal form.

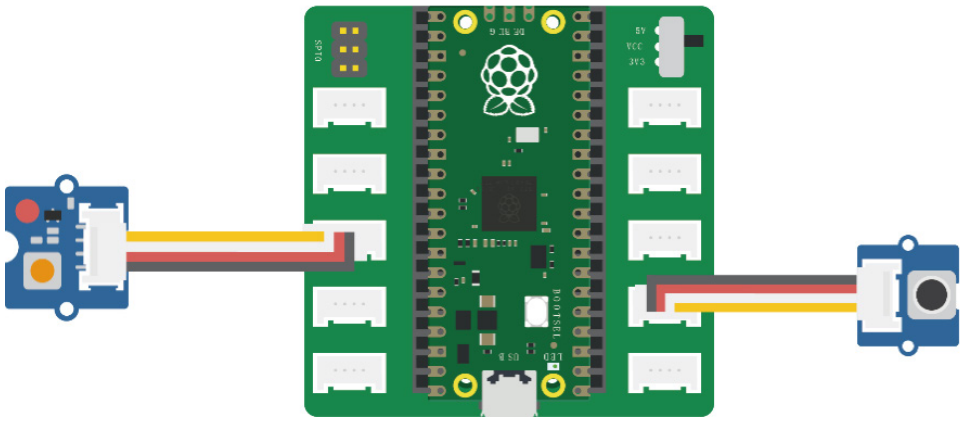
Project 2: Turn an LED On and Off with a Rotary Angle Sensor

First, let's try to write a simple program to turn on the LED when the Rotary Angle Sensor rotates through the center point.

Hardware Connection

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – LED Pack
- Grove – Rotary Angle Sensor

Plug the Pico and the Shield, use a Grove data cable to connect the LED to D16, and connect the Rotary Angle Sensor to A1.



Write a Program

Similarly, at the beginning of the program, import the functions that need to be used in the machine library, and define pins:

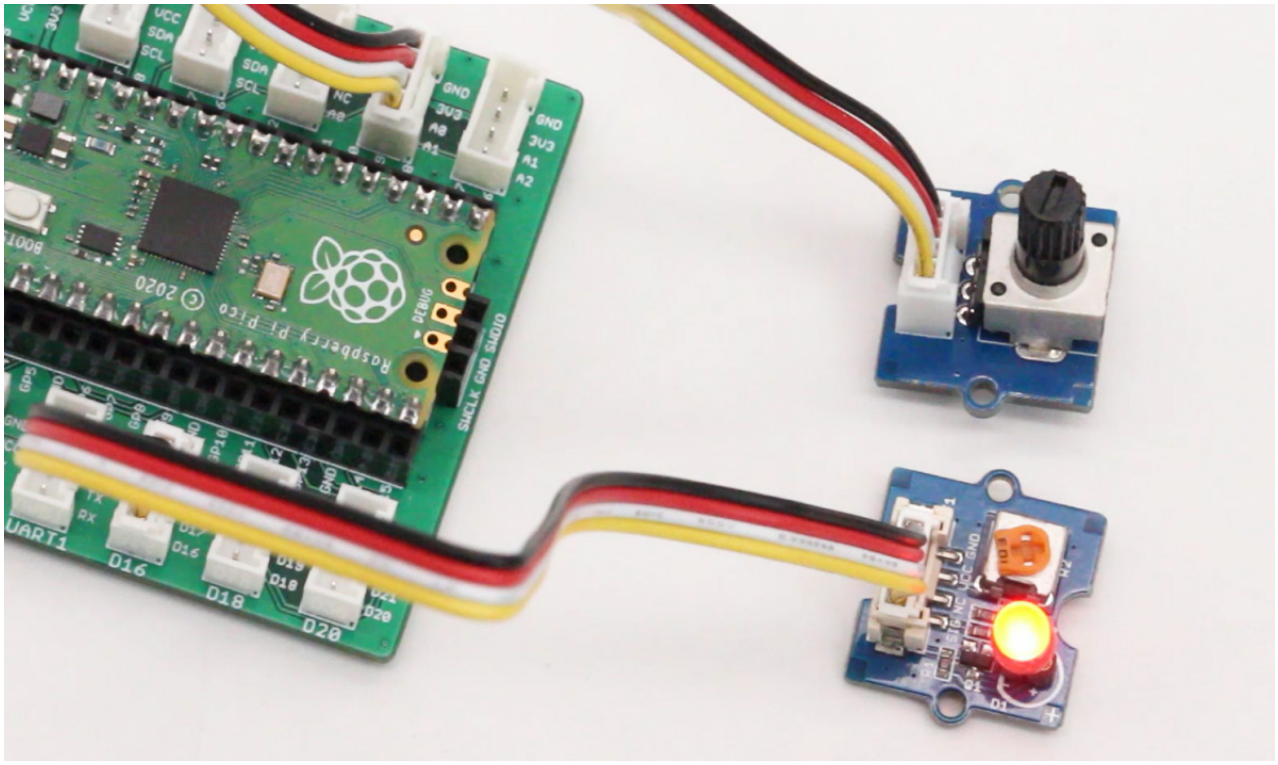
```
from machine import ADC,Pin
from time import sleep

LED = Pin(16,Pin.OUT)
ROTARY_ANGLE_SENSOR = ADC(1)
```

Then use the "if" statement to judge the condition. When the return value of the Rotary Angle Sensor is greater than 30000, the LED turns on; otherwise, the LED turns off.

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

LED = Pin(16,Pin.OUT)
ROTARY_ANGLE_SENSOR = ADC(1)
while True:
    print(ROTARY_ANGLE_SENSOR.read_u16())
    if ROTARY_ANGLE_SENSOR.read_u16() > 30000:
        LED.value(1)
        sleep(1)
    else:
        LED.value(0)
        sleep(1)
```

knob2.py

In the "if" statement, we use the comparison operator ">" to compare the relationship between the analog value read from the Rotary Angle Sensor and the 30000 threshold.

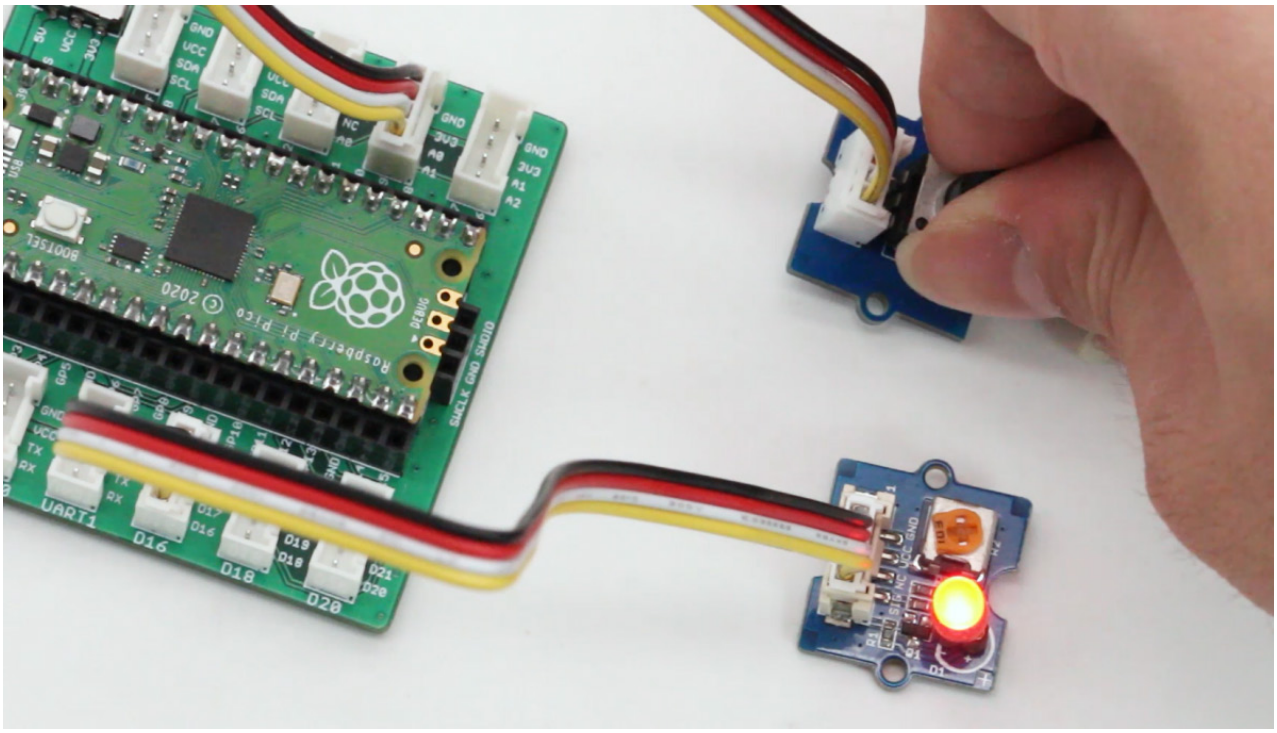


You can test the effect of the program by changing the comparison operator used in the "if" statement. Note that if we use "==" to write expressions in the program, it is difficult to achieve the effect we want. This is because the value range of analog value is wide. When we rotate the knob, it is difficult to turn it to a specific value to fulfill the condition of the expression.

In addition to using the comparison operator to complete a single expression statement, we can also use the logical operator to concatenate two independent statements to control the LED to light only within a specific value range:

```
1  from machine import ADC,Pin
2  from time import sleep
3
4  LED = Pin(16,Pin.OUT)
5  ROTARY_ANGLE_SENSOR = ADC(1)
6  while True:
7      print(ROTARY_ANGLE_SENSOR.read_u16())
8      if ROTARY_ANGLE_SENSOR.read_u16() > 20000 and
   ROTARY_ANGLE_SENSOR.read_u16() < 40000:
9          LED.value(1)
10         sleep(1)
11     else:
12         LED.value(0)
13         sleep(1)
```

knob3.py



Thinking Expanding

Try to explore more uses of operators with the LED and the Rotary Angle Sensor. For example, when the reading of the Rotary Angle Sensor is greater than 30000, program the LED to turn off; otherwise, turn the LED on. You can also try to use the logical operator "or". If you change the "and" operator in Project 2 to "or", what will happen?

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

LED = Pin(16,Pin.OUT)
ROTARY_ANGLE_SENSOR = ADC(1)
while True:
    print(ROTARY_ANGLE_SENSOR.read_u16())
    if ROTARY_ANGLE_SENSOR.read_u16() > 20000 and
ROTARY_ANGLE_SENSOR.read_u16() < 40000:
        LED.value(1)
        sleep(1)
    else:
        LED.value(0)
        sleep(1)
```

Lesson 7 Analog Signal in Circuit (2) — the Use of PWM

You should have noticed that with a simple digital signal output, we can only switch the output value between 0 and 1, that is, to turn the light on or off. We can't control the brightness of the light, let alone any other advanced effects. In order to realize more effects, we need to use analog signals for output to control the brightness of the light.

Knowledge Base

As mentioned in the last lesson, Pico, as a microcontroller, cannot work with an analog signal directly. The analog signal must be converted into the digital signal by the ADC (Analog to Digital Converter) for Pico to process and understand the data.

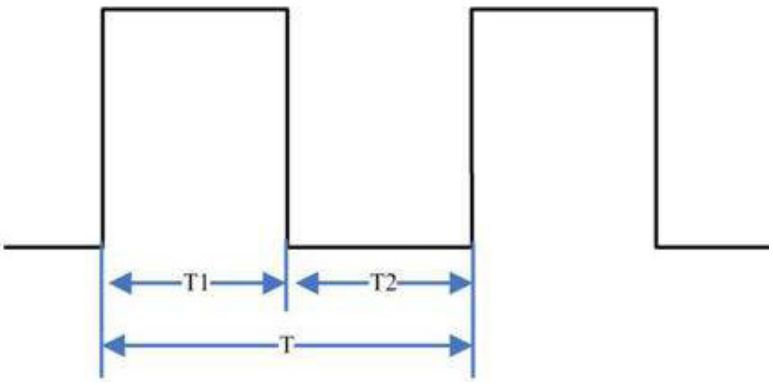
Similarly, Pico cannot directly send an analog signal to control the analog circuit. It must first convert its digital signal into an analog signal in some way. However, Pico itself does not have a DAC (Digital to Analog Converter) to convert a digital signal into an analog signal. So when Pico needs to output an analog signal, it needs to use another method to control the analog circuit — PWM.

PWM

PWM is short for pulse width modulation. Where, "pulse" refers to the pulse signal, which is a digital signal sent out at periodic continuous on and off patterns by the microcontroller. Because the signal sent out by the transistor only has two states, on and off, the signal switches only between high level and low level. PWM pulse signal is mainly defined by three components: cycle, duty cycle and frequency.

As shown in the figure below, cycle (T) is the total time for a pulse signal to go through a high-level state (T1) and a low-level state (T2), that is, cycle (T) = T1 + T2. Duty cycle represents the ratio of the time when the signal is at a high level to the cycle in a cycle, so a duty cycle can be calculated as cycle (T) / T1.

Frequency represents the number of cycles per second (1000 Hz means 1000 cycles per second). The shorter the cycle, the higher the frequency.



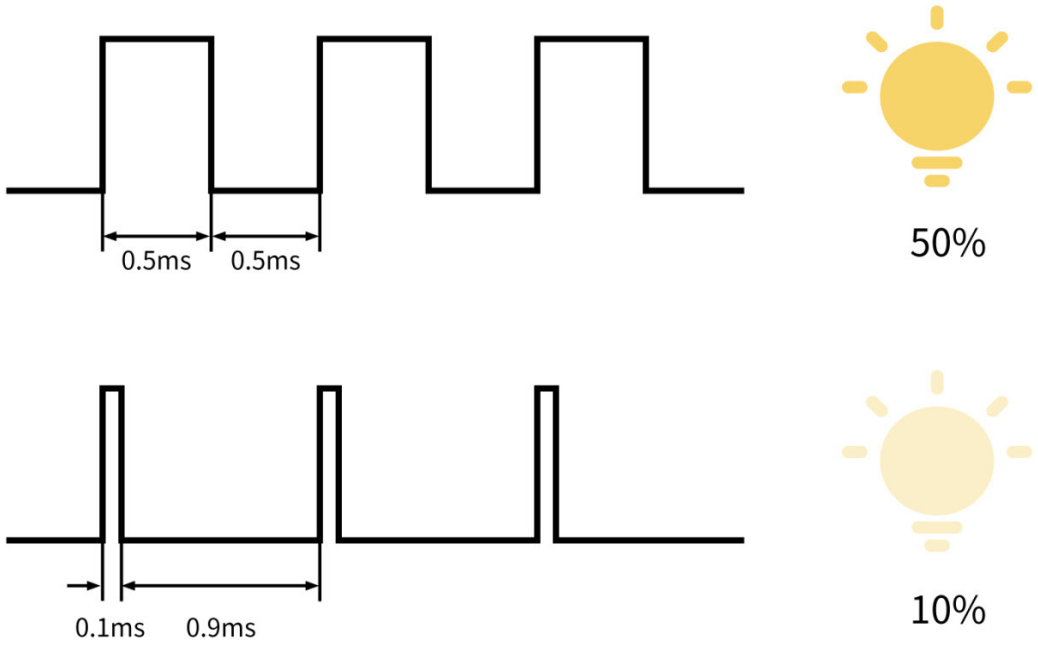
The objective of pulse width modulation is to modulate the duty cycle of the pulse signal according to the demand without changing other parameters of the pulse signal, so as to disguise a digital signal as a constant voltage analog signal.

So how is pulse width modulation realized? Take controlling the brightness of LED as an example. When we input a PWM signal to the LED, the LED actually receives a digital signal that constantly switches between high level and low level. This digital signal makes the LED turn on and off constantly.

When the frequency of on and off is fast enough, that is, the frequency of PWM is high enough, this flashing of light will not be recognized by human eyes.

At this time, under the conditions of a constant cycle, we only need to change the ratio of the time when the LED is turned on relative to the time that it is turned off. Thus, by adjusting the duty cycle, we can then change the brightness of the LED.

You can better understand this concept by bringing in some numbers. If the LED is turned on in 0.5 ms and turned off in 0.5 ms within 1 ms, you will not notice the light flashing since the frequency is too fast. You will instead only observe the brightness of the light as reduced to half of its always–on state. On the other hand, when the light turned on in 0.1 ms and turned off in 0.9 ms in a 1 ms cycle, the light perceived by your eyes will be only one tenth of its original brightness.



To operate on the duty cycle, you need to use arithmetic operators.

Arithmetic Operator

Arithmetic operators, as the name suggests, are operators for mathematical operations. The “+” “–” “*” “/” that we often use belong to arithmetic operators. By using arithmetic operators, we can do mathematical operations on variables or other objects in the program to change their values.

The common arithmetic operators are described as follows, where a = 1, b = 2:

Operator	Description	Example
+	add — add two objects	"a + b" outputs 3
-	minus — get a negative number or subtract one number from another	"a - b" outputs -1
*	multiply — multiply two numbers or return a string that has been repeated several times	"a * b" outputs 2
/	divide — x divided by y	"b / a" outputs 2
%	modulo — return remainder of a division	"b % a" outputs 0

//	floor division — return the integral part of a quotient (floor division)	>>> 9//2
		4
		>>> -9//2
		-5

Practice & Operation

Project 1: Check Return Value of Rotary Angle Sensor

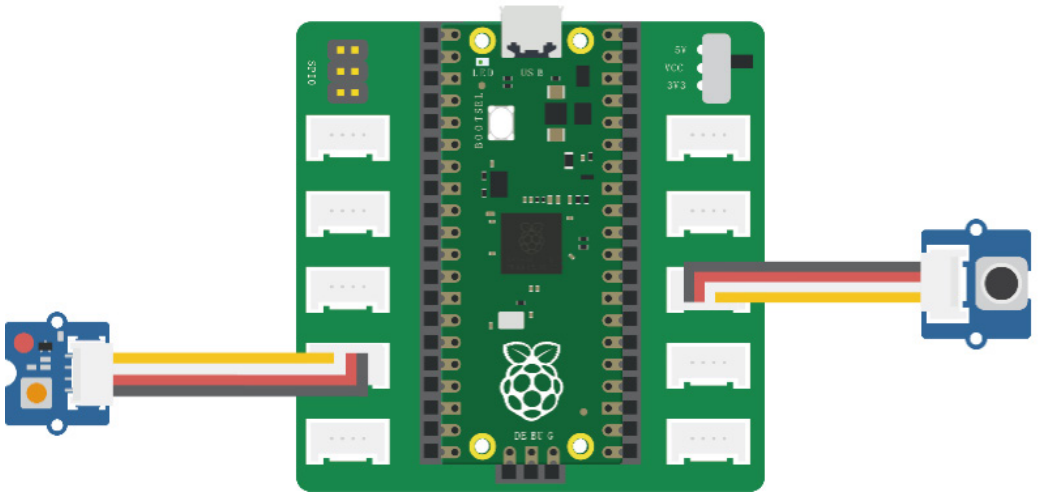
First, let's learn how to use the PWM signal to output analog signal, and use the return value of the Rotary Angle Sensor to control the brightness of the LED.

Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – Rotary Angle Sensor

Plug the Pico and the Shield, use a Grove data cable to connect the LED to D18, and connect the Rotary Angle Sensor to A0.

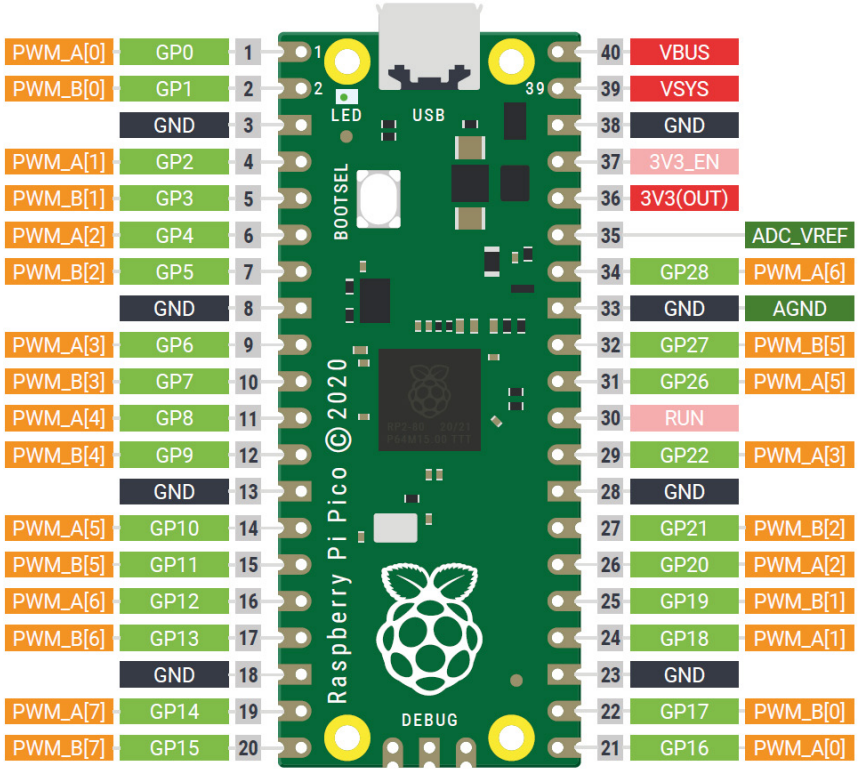


Write a Program

Just like the Pin function, the function of setting pins to PWM mode is also available in the machine library:

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

We need to use PWM to set related pins. Although each pin of Pico can be set as a PWM pin, some pins cannot be simultaneously set to PWM mode. This is because the pulse width modulator on Pico is divided into eight parts, where each part has two outputs — A and B. Pico has 26 pins, which causes some pins on Pico to share the same pulse width modulator. Therefore, when you need to set multiple PWM pins, you need to be careful to avoid conflicting pin settings.



Fortunately, we don't need to account for this problem in this program. All we have to do is set a single PWM pin:

```
1 ROTARY_ANGLE_SENSOR = ADC(0)
2 LED_PWM = PWM(Pin(18))
```

After setting the pin, use the "freq()" function to set the frequency of the PWM signal to 500:

```
1 LED_PWM.freq(500)
```

It is very important to choose the right frequency. If the frequency is very low, the flash of the LED during power on and off will be visible to the naked eye and occur intermittently. If the frequency is very high, it will cause the LED switch to fail to open and close normally, affecting the stability.

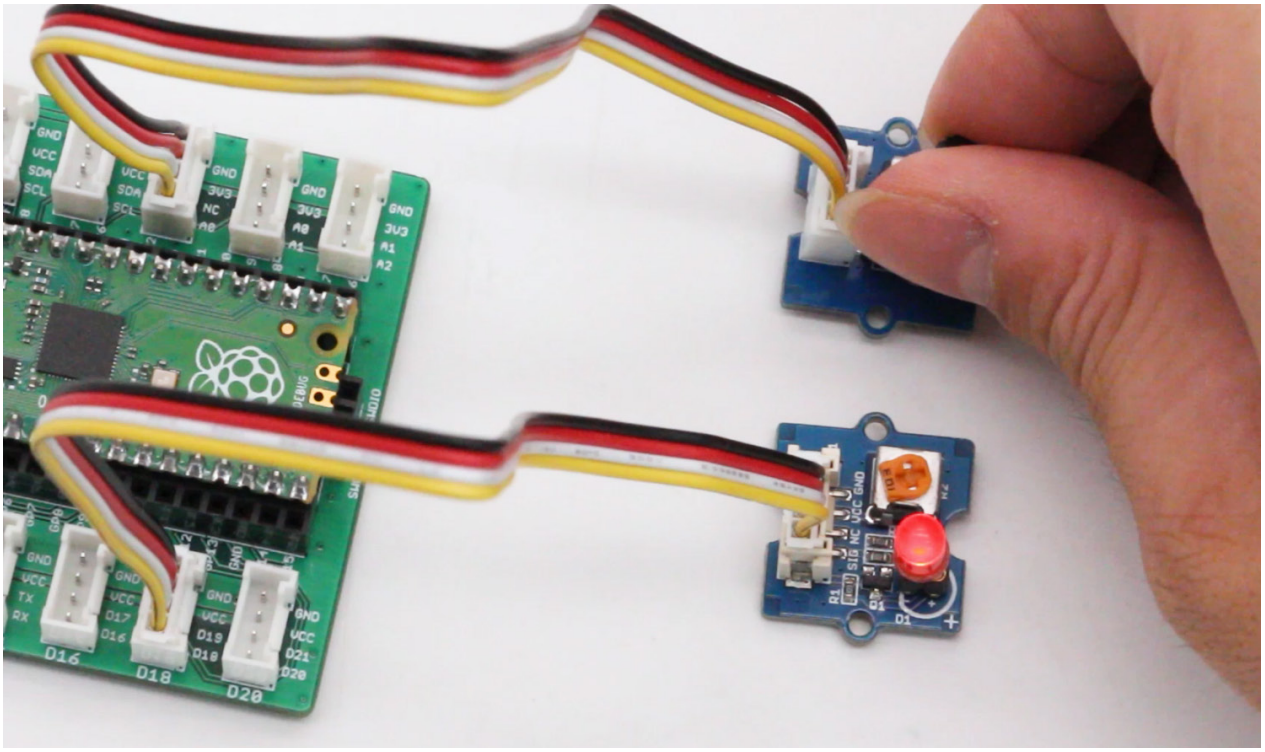
In this program, we use "read_u16()" to read the return value of the Rotary Angle Sensor, and use "duty_u16()" to output analog value to the LED. As the range of the two values is the same: 0—65535, we can directly use the return value of the Rotary Angle Sensor to control the brightness of the LED.

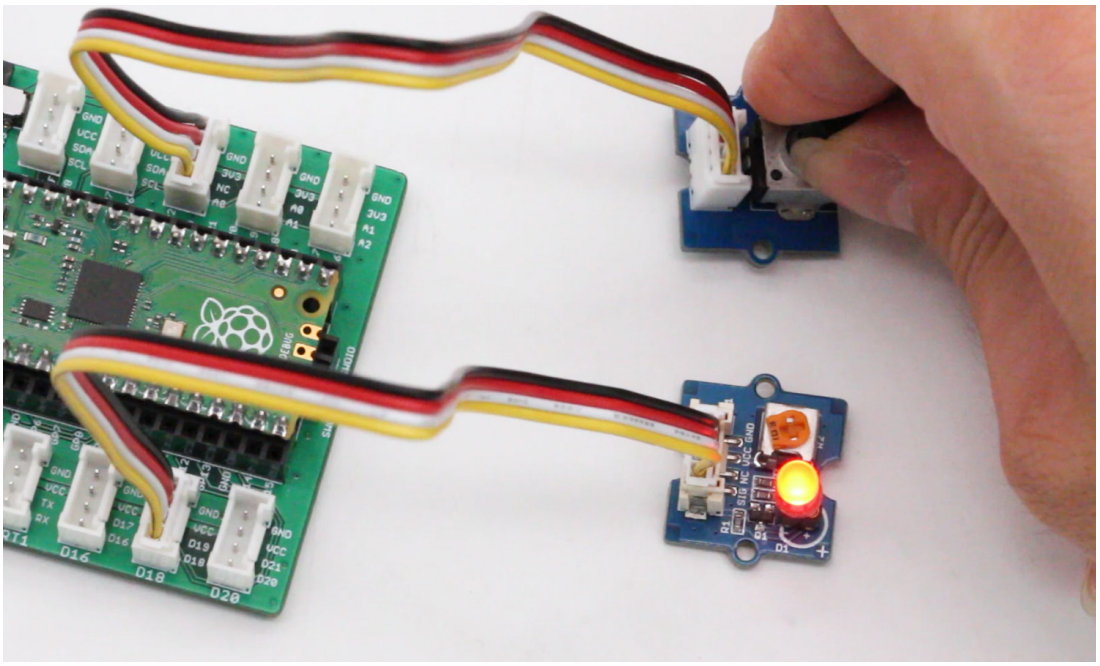
In the cycle, we first save the input analog value of the Rotary Angle Sensor to the variable "val", and then set the duty cycle of the PWM signal of the LED to "val". The complete program is as follows:

```
from machine import Pin,ADC,PWM

ROTARY_ANGLE_SENSOR = ADC(0)
LED_PWM = PWM(Pin(18))

LED_PWM.freq(500)

while True:
    val = ROTARY_ANGLE_SENSOR.read_u16()
    LED_PWM.duty_u16(val)
```

knob_led.py

Use a USB cable to connect Pico to the computer, click the "run" button to save the program to any location, you can see the actual running effect of the program.





Project 2: Breathing Light

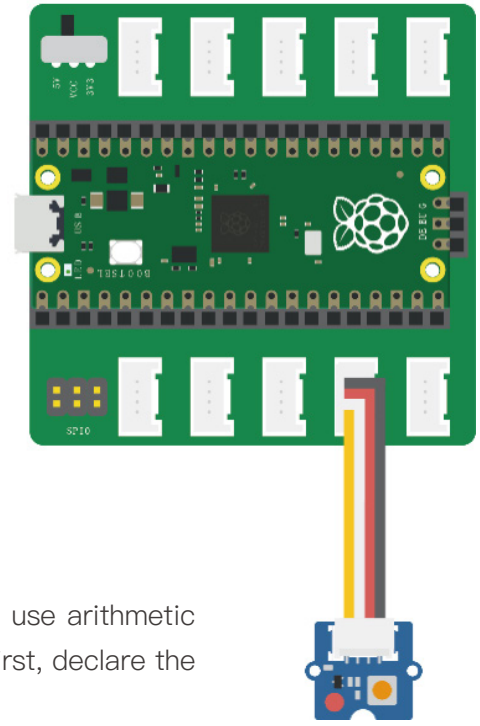
In this project, we will realize the effect of breathing light. That is, the brightness of LED will be changed from dark to light, and then from light to dark.

Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – LED Pack

Connect the LED to D18.



Write a Program

To realize the effect of breathing light, we need to use arithmetic operators to operate the duty cycle of the PWM signal.First, declare the functions in the library we need to use, and set the pins.

```
1 from machine import Pin,PWM
2 import utime
3
4 LED_PWM=PWM(Pin(18))
```

After setting the pin, use the "freq()" function to set the frequency of the PWM signal to 500:

```
LED_PWM.freq(500)
```

Through analysis, we can find that the breathing light is actually composed of two parts. The first part is that the LED lights up gradually from the dim state, and enters the second part when the brightness reaches the maximum. In the second part, the brightness of the LED will begins to decay until it goes out. Repeating these two parts in alternation, we can realize the effect of a breathing light.

We can use two while-loop statements to realize the effects of these two parts. First, we set a variable "val" to help us adjust the duty cycle of the PWM signal. At the beginning of the program, we set val to 0:

```
val = 0
```

Then we'll work on the first part, where the LED lights up gradually:

```
while val<65535:
    val=val+50
    utime.sleep_ms(1)
    LED_PWM.duty_u16(val)
```

In this program, we increase the value of "val" by 50 every millisecond until it is greater than 65535, and use the "duty_u16" function to adjust the duty cycle of the PWM signal using this same value.

When the value of "val" is greater than 65535, our code will exit the first while-loop and enter the next while-loop. Similarly, to make the light dim gradually, we keep decreasing the value of "val" until it is 0.

```
while val>0:
    val=val-50
    utime.sleep_ms(1)
    LED_PWM.duty_u16(val)
```

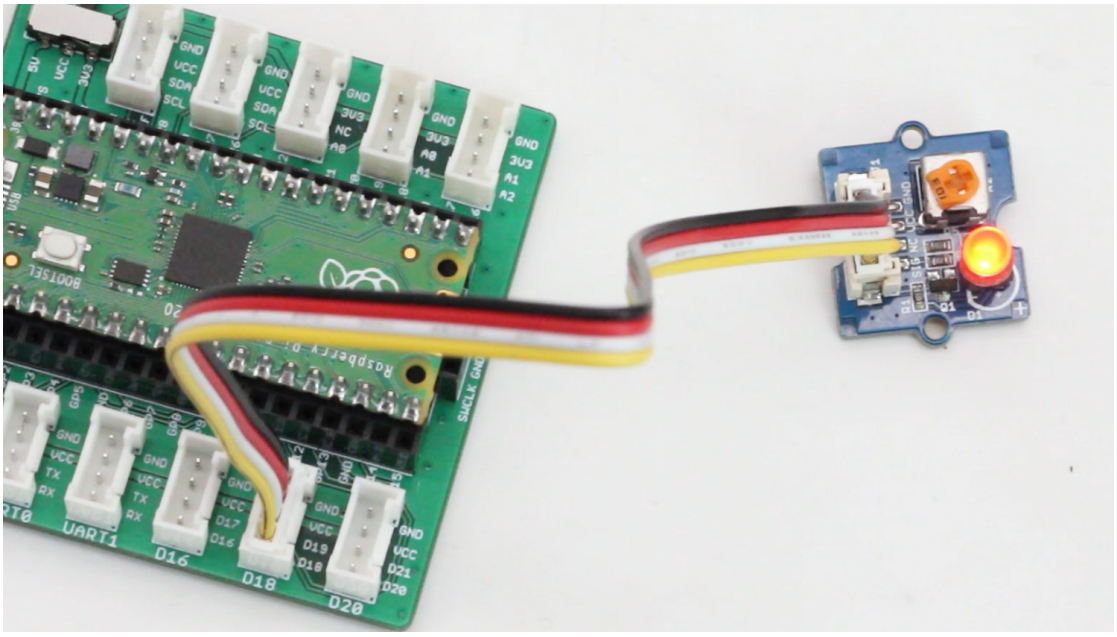
Then, we will place both loops in another "while True" loop to make the two parts of the program run continuously, and the effect of breathing light is completed.

```
import utime
from machine import Pin,PWM
```

```
4  LED_PWM=PWM(Pin(18))
5
6  val=0
7  LED_PWM.freq(500)
8  while True:
9      while val<65535:
10         val=val+50
11         utime.sleep_ms(1)
12         LED_PWM.duty_u16(val)
13     while val>0:
14         val=val-50
15         utime.sleep_ms(1)
16         LED_PWM.duty_u16(val)
```

fade.py

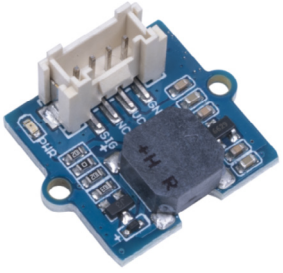
Use a USB cable to connect Pico to the computer, click the "run" button to save the program to any location, you can see the program in action.



Thinking Expanding

Besides LED, do you know which electronic modules can be controlled with a PWM signal?

Buzzer:



Servo:

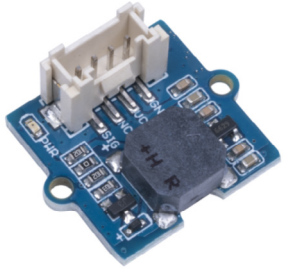
Lesson 8 Make Buzzer Sing — the Use of Function

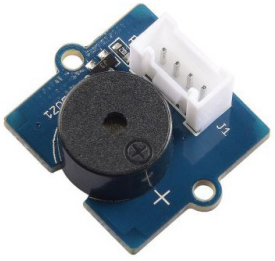
The term "function" in mathematics generally refers to a corresponding relationship and change process between values. In the field of programming, a function is a program with specific functions that can complete specific tasks, thus reducing the workload of repetitive programming. The use of PWM analog signal can not only control the brightness of the light but also control many other electronic devices, and the buzzer is one of them. In this lesson, we will learn about the basic knowledge and usage of a buzzer, and how to use functions to simplify programs and even create simple music.

Knowledge Base

Buzzer

A buzzer is a common sound device, and is often used as a sound generator in computers, printers, copiers, alarms, electronic toys, automotive electronic equipment, telephones, timers and other electronic products. They can be divided into either passive or active buzzers.





The "source" here does not refer to power source but instead the source of the oscillation.

The oscillation source is the oscillation circuit, which can convert the input DC signal into a physical oscillation which drives the buzzer to sound.The active buzzer has a built-in oscillation source, which can work by providing a constant DC voltage when it is in use. Its sound frequency is determined by the internal oscillation circuit, and its tone generally cannot be changed.

The passive buzzer has no oscillation source, and can only rely on the external input PWM (square wave) signal to achieve the alternating magnetic field. So, we can use the PWM signal to drive the vibration device to make the buzzer sound. By adjusting the frequency and duty cycle of the PWM signal, we can control the tone and volume of the buzzer.

The passive buzzer has no oscillation source, and can only rely on the external input PWM (square wave) signal to achieve the alternating magnetic field. So, we can use the PWM signal to drive the vibration device to make the buzzer sound. By adjusting the frequency and duty cycle of the PWM signal, we can control the tone and volume of the buzzer.

In this lesson, we will use the buzzer to play simple music. But before we start, we will learn how to reduce our workload by defining functions.

Define Function

In mathematics, “function” refers to a corresponding relationship and change process between values. In the field of programming, a function is a program with specific functions and can complete specific tasks. When we complete this program, we can call it repeatedly in later programs to deal with the same task without writing more code!In the previous lessons, you have used many Python built-in functions, such as print(), pin(), etc. Similarly, we can also use our own custom functions.

In Python, we use “def” to define functions. The syntax of defining functions is as follows:

```
def function name(parameter list):
    function body
```

When defining a function, the function code block starts with “def”, followed by the function name and the parenthesis “()”. The colon “:” after the parenthesis indicates the beginning of the function body. The function body contains all the code that define a function. When you complete the function definition, the system can execute the code in the function body every time you call the function.

Let’s look at the example program:

```
def HelloWorld():
    print("Hello World")

HelloWorld()
```

The execution result of the program is as follows:

```
Hello World
```

In this program, we define a function named "HelloWorld". Every time we call the function with "HelloWorld()", we execute the "print("Hello World")" statement in the function body.

In the brackets after the function name, we can also write the parameters that may need to be used by the function, for example:

```
a = 4
b = 5
def max(a, b):
    if a > b:
        return a
    else:
        return b
print(max(a, b))
```

In this program, we create a function named "max()", which introduces two parameters into the parameter list. In the function body, the program compares the two numbers and returns the larger number.The "return [expression]" represents the end of the function, and it returns a value to the function calling it. For example, in this program, we judge the larger number between a and b through the "if" condition and return the chosen value. Therefore, the execution result of the program is 5, that is, the value of b which is greater.

```
5
```

And that's how functions are defined.

Practice & Operation

Project 1: Playing Basic Notes

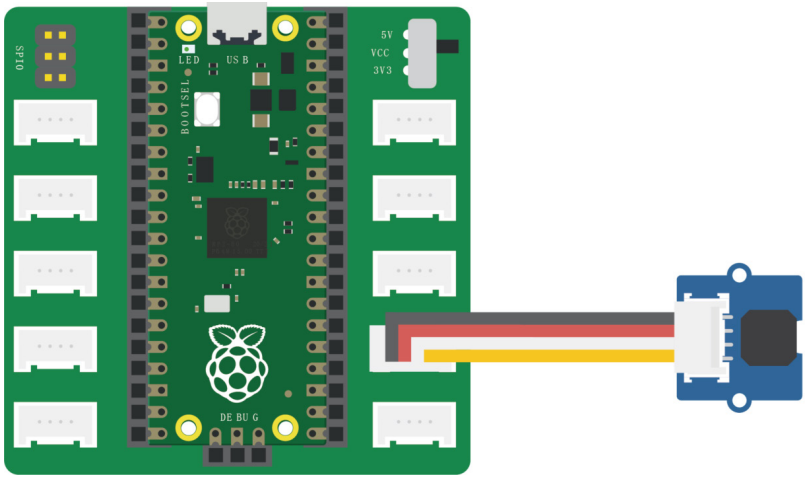
First, let's try to control the buzzer to play seven basic notes: do, re, mi, fa, sol, la, si.

Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico
- Shield for Pi Pico
- Grove – Passive Buzzer

Attach the Pico and the Shield, and use a Grove data cable to connect the buzzer to A1.



Write a Program

If we want to use the buzzer to play music, we need to learn to control the two dimensions of the sound emitted by the buzzer: volume and tone.

For the volume, like adjusting the brightness of the LED, we can control it by changing the duty cycle of the PWM signal as well. On the other hand, since the tone of a sound is determined by its frequency, we can control it by just modifying the frequency of the PWM signal.

Let's write a program to let the buzzer send out seven notes: do, re, mi, fa, sol, la, si.

In this program, we need to define pins and use the "sleep()" function to control the duration of each note. As usual, we need to declare the function library at the beginning of the program:

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

Next, define the pin:

```
buzzer = PWM(Pin(27))
```

Next, we need to use "freq()" and "duty_u16()" to define the tone and volume:

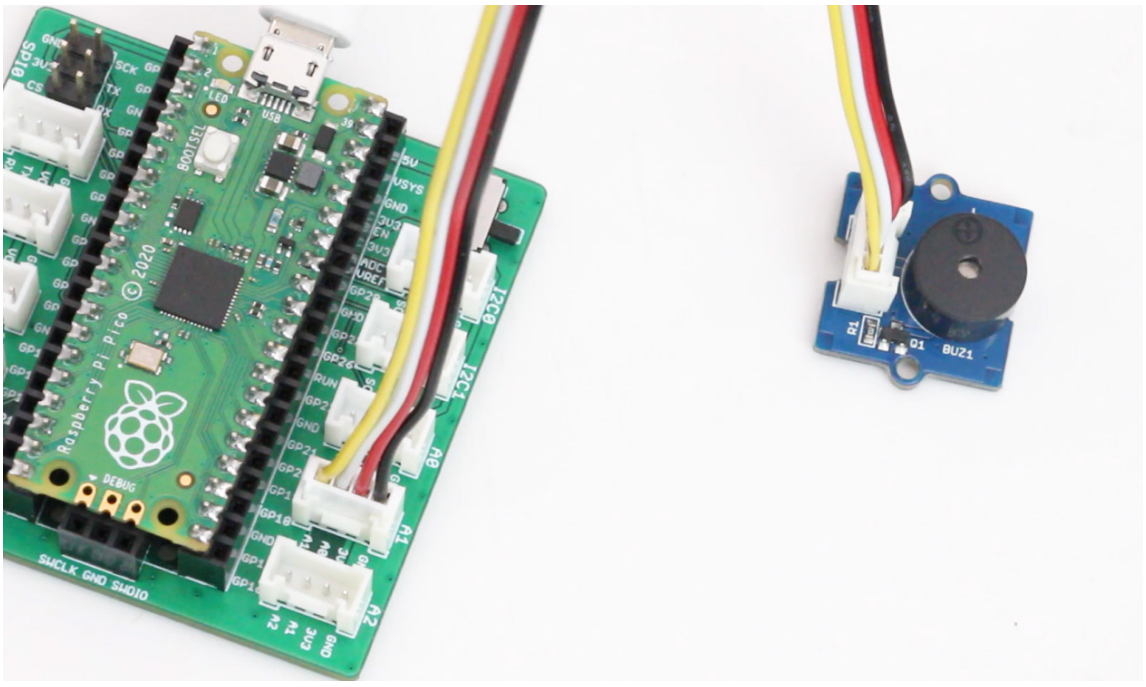
```
buzzer.freq(1046) #DO
buzzer.duty_u16(1000)

buzzer.freq(1175) #RE
buzzer.duty_u16(1000)

buzzer.freq(1318) #MI
buzzer.duty_u16(1000)

buzzer.freq(1397) #FA
buzzer.duty_u16(1000)

buzzer.freq(1568) #SO
buzzer.duty_u16(1000)

buzzer.freq(1760) #LA
buzzer.duty_u16(1000)

buzzer.freq(1967) #SI
buzzer.duty_u16(1000)
```

Finally, use "sleep()" to specify the duration of each note, and complete the program. The complete program is as follows:

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

buzzer = PWM(Pin(27))
while True:
    buzzer.freq(1046) #DO
    buzzer.duty_u16(1000)
    sleep(1)
    buzzer.freq(1175) #RE
    buzzer.duty_u16(1000)
    sleep(1)
    buzzer.freq(1318) #MI
    buzzer.duty_u16(1000)
    sleep(1)
    buzzer.freq(1397) #FA
    buzzer.duty_u16(1000)
    sleep(1)
    buzzer.freq(1568) #SO
    buzzer.duty_u16(1000)
    sleep(1)
    buzzer.freq(1760) #LA
    buzzer.duty_u16(1000)
    sleep(1)
    buzzer.freq(1967) #SI
    buzzer.duty_u16(1000)
    sleep(1)
```

buzzer.py

Use a USB cable to connect Pico to the computer, click the "run" button to save the program to any location. Now, you can hear the buzzer play seven basic notes.



Project 2: Two Tigers

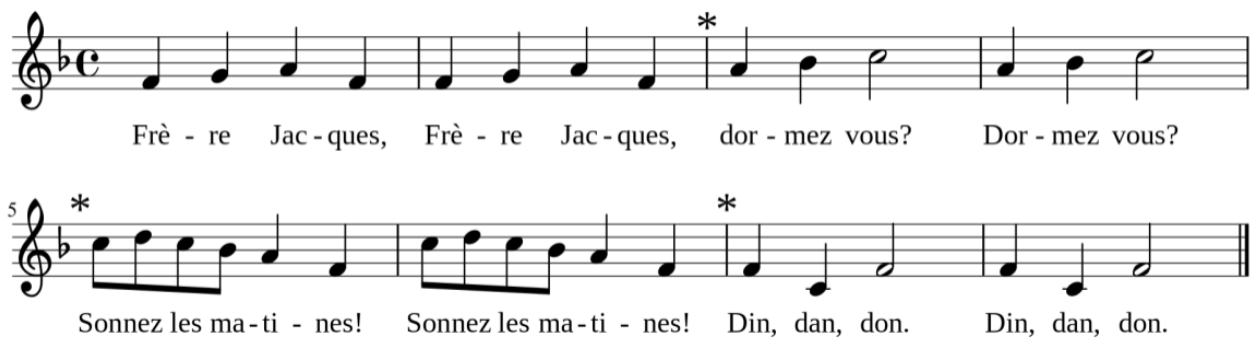
Next, let's try to control the buzzer to play the complete music of Two Tigers by defining a function to simplify the program.

Write a Program

When we write a simple piece of music, we can write the program line by line. However, complex programs written this way are too long and repetitive. If we want to reproduce whole

music, whether it is from the perspective of improving code reuse or program readability, we need to use custom functions to simplify the program. Following Project 1, we can define every note as a function that can be called when we want to play that note.

Taking DO as an example, the program to define the function is as follows:




```
def DO():
    buzzer.freq(1046) #1
    buzzer.duty_u16(1000)
```

Define seven notes and a stop-sound program, N(), as functions in the same way:

```
def DO():
    buzzer.freq(1046) #1
    buzzer.duty_u16(1000)
def RE():
    buzzer.freq(1175) #2
    buzzer.duty_u16(1000)
def MI():
    buzzer.freq(1318) #3
    buzzer.duty_u16(1000)
def FA():
    buzzer.freq(1397) #4
    buzzer.duty_u16(1000)
def SO():
    buzzer.freq(1568) #5
    buzzer.duty_u16(1000)
def LA():
    buzzer.freq(1760) #6
    buzzer.duty_u16(1000)
def SI():
    buzzer.freq(1967) #7
    buzzer.duty_u16(1000)
def N():
    buzzer.duty_u16(0) #close
```

In the later program, we only need to call these functions to complete the music. The program of playing Two Tigers is then as follows:

```
from machine import Pin, I2C, ADC, PWM
from time import sleep

```

```
buzzer = PWM(Pin(27))
vol = 1000
def DO():
    buzzer.freq(1046) #1
    buzzer.duty_u16(vol)
def RE():
    buzzer.freq(1175) #2
    buzzer.duty_u16(vol)
def MI():
    buzzer.freq(1318) #3
    buzzer.duty_u16(vol)
def FA():
    buzzer.freq(1397) #4
    buzzer.duty_u16(vol)
def SO():
    buzzer.freq(1568) #5
    buzzer.duty_u16(vol)
def LA():
    buzzer.freq(1760) #6
    buzzer.duty_u16(vol)
def SI():
    buzzer.freq(1967) #7
    buzzer.duty_u16(vol)
def N():
    buzzer.duty_u16(0) #close

while True:

    DO()
    sleep(0.25)
    RE()
    sleep(0.25)
    MI()
    sleep(0.25)
    DO()
    sleep(0.25)
    N()
    sleep(0.01)

    DO()
    sleep(0.25)
    RE()
    sleep(0.25)
    MI()
    sleep(0.25)
    DO()
    sleep(0.25)

    MI()
    sleep(0.25)
    FA()
    sleep(0.25)
    SO()
    sleep(0.5)

    MI()
    sleep(0.25)
    FA()
    sleep(0.25)
    SO()
    sleep(0.5)
    N()
    sleep(0.01)

    SO()
    sleep(0.125)
    LA()
    sleep(0.125)
    SO()
    sleep(0.125)
    FA()
    sleep(0.125)
    MI()
    sleep(0.25)
    DO()
    sleep(0.25)

    SO()
    sleep(0.125)
    LA()
    sleep(0.125)
    SO()
    sleep(0.125)
    FA()
```

```
88      sleep(0.125)
89      MI()
90      sleep(0.25)
91      DO()
92      sleep(0.25)
93
94      RE()
95      sleep(0.25)
96      SO()
97      sleep(0.25)
98      DO()
99      sleep(0.5)
100     N()
101     sleep(0.01)
102
103     RE()
104     sleep(0.25)
105     SO()
106     sleep(0.25)
107     DO()
108     sleep(0.5)
```

In the fifth line of the program, we defined a variable named "vol" to represent the duty cycle value of the PWM signal in later programs. Now, whenever we want to modify the volume of the buzzer, we only need to modify the initial value of the variable "vol".

```
1 vol = 1000
```

If you look at our program carefully, you will find it's much simpler than writing the program line by line. Now, the only part that is always repeated in this program is the "sleep()" function. In complete music, many notes sound at different times — thus we cannot simply add the "sleep()" function to the function body. Instead, we need to create a custom function with parameters. Similarly, taking the note DO as an example, the program to DO(0.5)the note DO with duration is as follows:

```
1 def DO(time):
2     buzzer.freq(1046) #1
3     buzzer.duty_u16(vol)
4     sleep(time)
```

When we use this function, we need to type the parameter of the function body in the parentheses. The parameter is called by the "sleep() function" in the function body so that we can conveniently control the sound time of the note. For example, when the note DO needs to be sounded for 0.5 seconds, we can write the program like this:

```
1 DO(0.5)
```

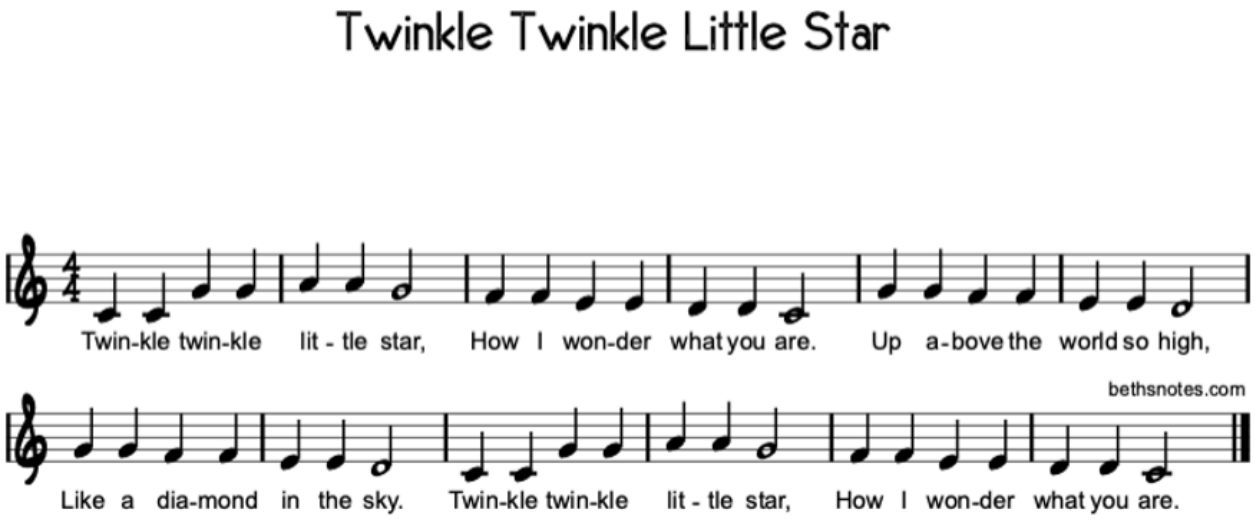
How about it? Is it more concise? By defining functions this way, we can make the program both more efficient and easier to read. The final program is as follows:

```
1 from machine import Pin, I2C, ADC, PWM
```

```
from time import sleep

buzzer = PWM(Pin(27))
vol = 1000
def DO(time):
    buzzer.freq(1046) #1
    buzzer.duty_u16(vol)
    sleep(time)
def RE(time):
    buzzer.freq(1175) #2
    buzzer.duty_u16(vol)
    sleep(time)
def MI(time):
    buzzer.freq(1318) #3
    buzzer.duty_u16(vol)
    sleep(time)
def FA(time):
    buzzer.freq(1397) #4
    buzzer.duty_u16(vol)
    sleep(time)
def SO(time):
    buzzer.freq(1568) #5
    buzzer.duty_u16(vol)
    sleep(time)
def LA(time):
    buzzer.freq(1760) #6
    buzzer.duty_u16(vol)
    sleep(time)
def SI(time):
    buzzer.freq(1967) #7
    buzzer.duty_u16(vol)
    sleep(time)
def N(time):
    buzzer.duty_u16(0) #close
    sleep(time)
while True:

    DO(0.25)
    RE(0.25)
    MI(0.25)
    DO(0.25)
    N(0.01)

    DO(0.25)
    RE(0.25)
    MI(0.25)
    DO(0.25)

    MI(0.25)
    FA(0.25)
    SO(0.5)

    MI(0.25)
    FA(0.25)
    SO(0.5)
    N(0.01)

    SO(0.125)
    LA(0.125)
    SO(0.125)
    FA(0.125)
    MI(0.25)
    DO(0.25)

    SO(0.125)
    LA(0.125)
    SO(0.125)
    FA(0.125)
    MI(0.25)
    DO(0.25)

    RE(0.25)
    SO(0.25)
    DO(0.5)
    N(0.01)

    RE(0.25)
    SO(0.25)
    DO(0.5)
```

two_tiger.py

Use a USB cable to connect Pico to the computer, and click the "run" button to save the program to any location. Now, you can hear the buzzer play Two Tigers.



Thinking Expanding

Try to use what you learned in this lesson to write a program that can play Twinkle Twinkle Little Star.

The reference program is as follows:

little_star.py

Lesson 9 Journey of Data Visualization — the Use of LCD

In our lives, we see displays everywhere. All kinds of visualization devices, such as TV, computer, mobile phone, car display, LCD billboards in shopping malls and so on, provide various possibilities for human–computer interaction. In our previous projects, if we want to visualize data, we normally rely on the Thonny Shell. However, in some projects, we cannot always connect Pico to the computer. At this time, we need to use some other electronic hardware to complete this work, and a variety of display devices come in handy.

In various open–source hardware projects, the common displays are LCD, LED, 3D, OLED, etc. They have their own advantages and disadvantages as display devices and can be used in different fields and scenarios according to their characteristics. In this lesson, we will specifically learn how to use LCDs to realize data visualization.

Knowledge Base

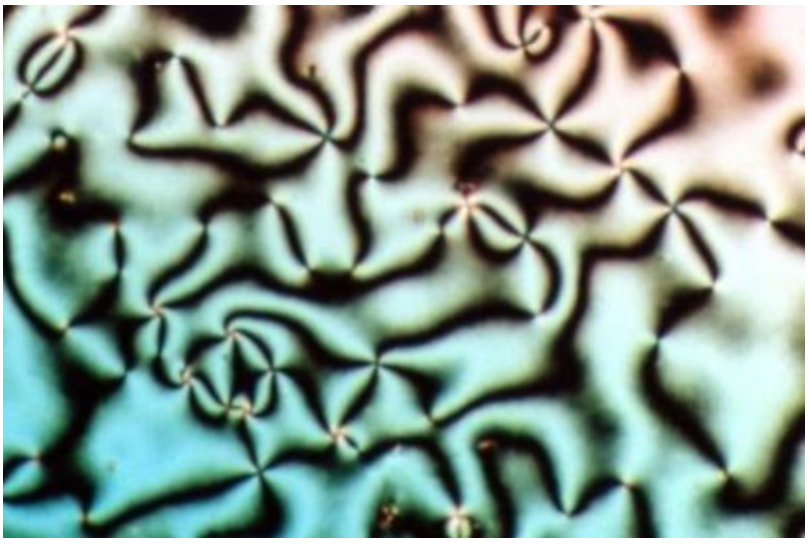
LCD

LCD is short for liquid–crystal display. In 1888, Austrian botanist Reinitzer discovered liquid crystals, which is a unique organic compound with two melting points.

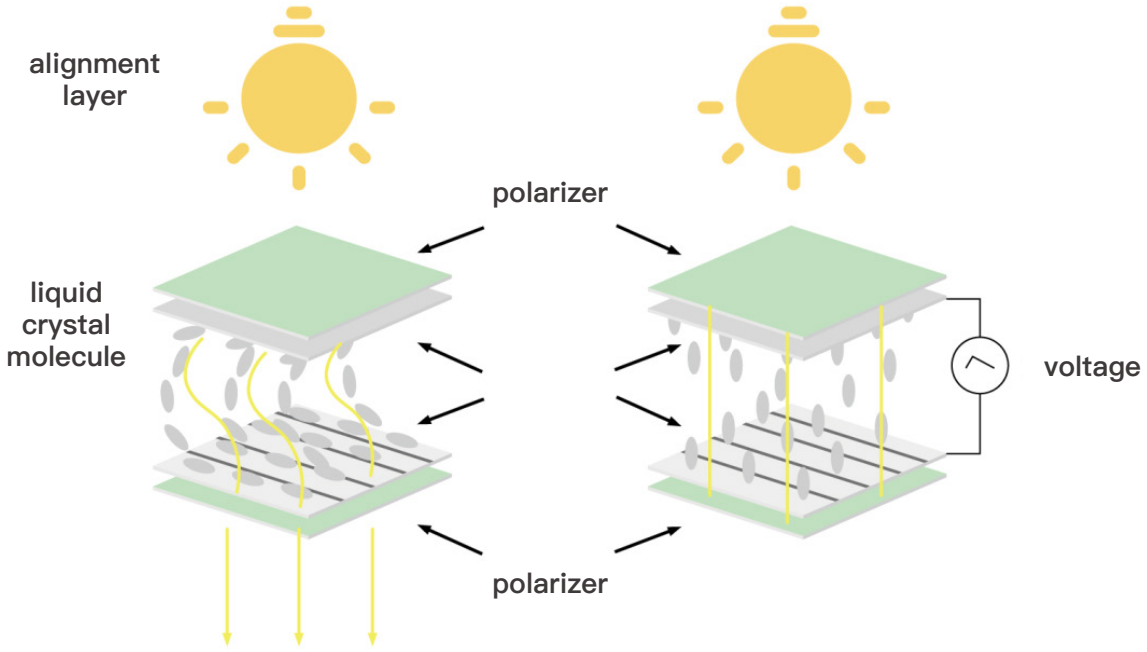
Reinitzer

Lehmann

When scientists heat the solid liquid crystal to 145℃, the liquid crystal first melts into a turbid liquid. When they continue to heat the liquid crystal to 175℃, it melts again and becomes a clear and transparent liquid. Later, through careful observation, the German physicist Lehmann found that when the liquid crystal melts into a turbid liquid, it is liquid in appearance, but it shows the unique birefringence of crystal. Thus, Lehmann named it "liquid crystal", which is the origin of its name. Reinitzer and Lehman were later known as the father of liquid crystals. Since liquid crystals were discovered, people did not know what they were used for. In fact, it was not until 1968 that people began to use it as a material for the electronics industry.



In 1968, RCA found that when the liquid crystal is affected by different electric fields, its molecules will turn in different directions, thus changing the refractive index of the liquid crystal, and affecting the direction of passing light. Therefore, the final imaging effect of LCD can be determined as long as the undesired light is filtered out through a polarizer.



Significantly, the liquid crystal itself cannot produce light, it can only change the direction of the light passing through it. So, in addition to the liquid crystal layer, we often need a backlight panel as the light source in an LCD. According to this characteristic, RCA developed a dynamic scattering LCD, and used this technology to develop the first LCD watch in 1972. Since then, LCD technology has become tremendously practical for commercial use.

The Grove – 16 x 2 LCD module we use in this lesson is an LCD display module. 16 x 2 means that the LCD display datas in two rows, and each line can display 16 characters, to make a total of 32 possible characters shown at a time.

I2C Bus

The Grove – 16 x 2 LCD module we use is an electronic module that uses an I2C communication protocol to communicate with the motherboard. I2C is a simple two–way two–wire synchronous serial bus developed by Philips. It only needs two wires to transmit information between devices connected to the bus. Communication over I2C takes place on two lines: a clock line (called SCL) and a data line (called SDA).

Therefore, when configuring pins for I2C access modules, we need to configure pins for both lines at the same time.

External Function Library

In previous lessons, we defined and used functions in the program But if we want to rewrite a new program, all the functions and variables we defined will disappear.

Python provides a solution to this problem by storing these programs that define functions or variables in specific files. When we want to use these definitions, we only need to use "import" at the beginning of the program to refer to this file, just like we do for a library file.

In short, a library file is a file that contains all the functions and variables you define, with the

extension ".py". Library files can be imported by other programs to use the functions in the library files.

For Python, libraries are generally divided into two categories. One is the standard library of python, which contains many function libraries commonly used in project development. In previous lessons, the machine library and utime library we used are all the standard libraries of Python.

Another category is the third-party library developed by Python users. Thanks to the growth of the Python community, you can find many very helpful function libraries when using Python to complete projects.

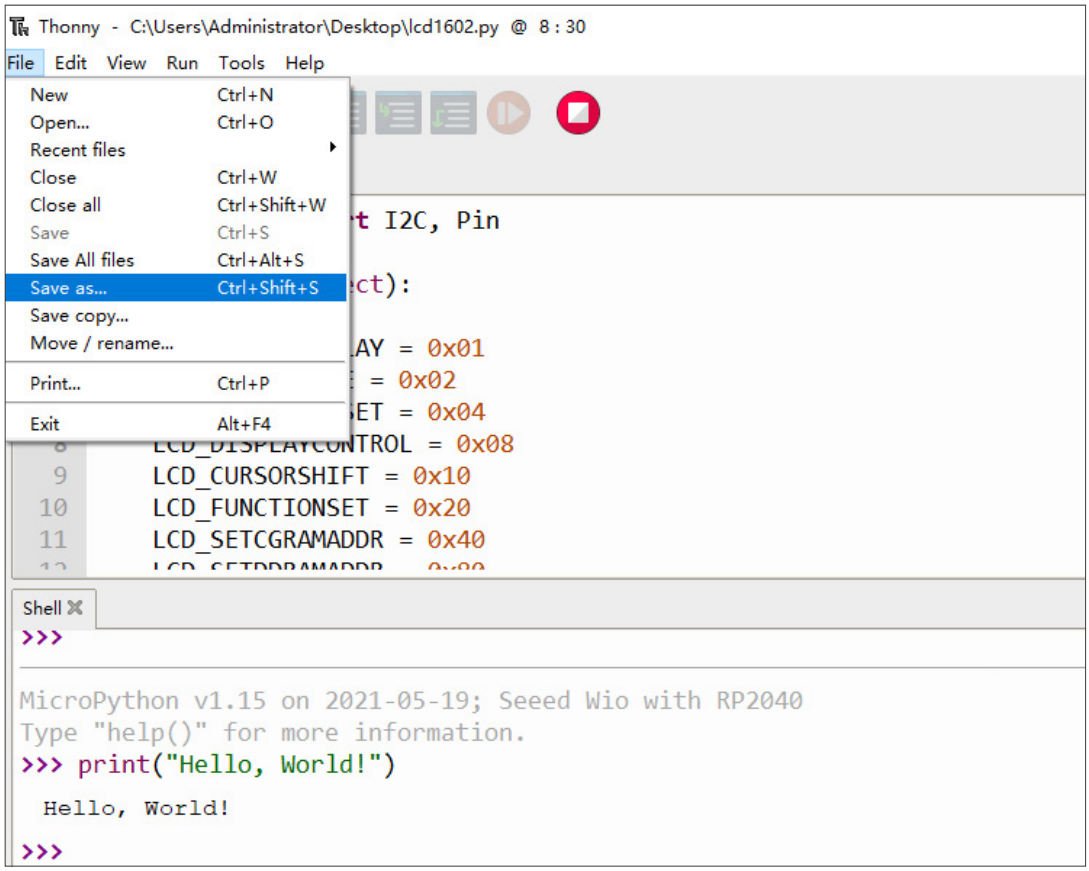
For standard libraries, we don't need to do extra work when using them. We can simply refer to them directly in the program. For third-party libraries, however, we need to save library files lcd1602.py onto the Pico before using them.

Note: you can also download all the libraries we used in this course through Pico-micropython.

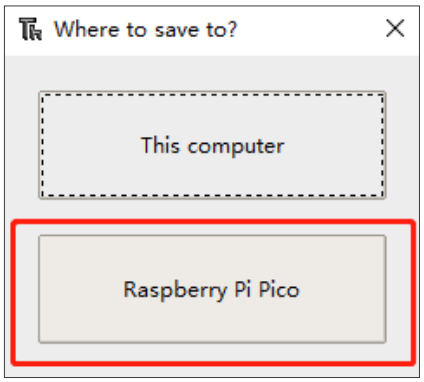
For example, in this lesson, if we want to use a third-party library to control the Grove – 16 x 2 LCD, we need to save it to Pico first.

1. Click to download the third-party library file we want to use, and save it to the computer.

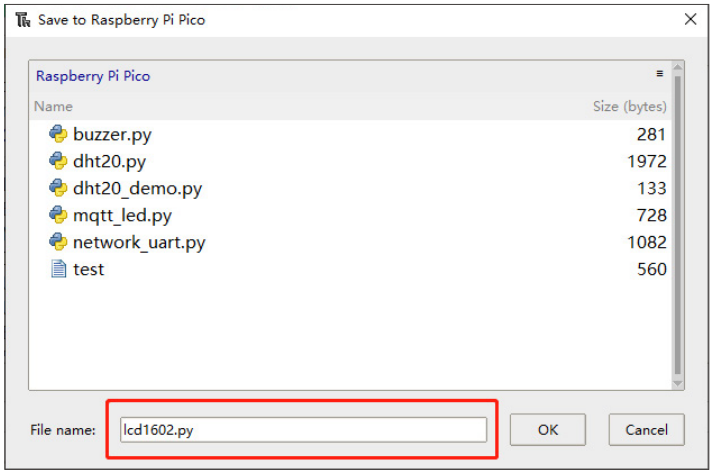
2. Connect Pico to the computer, use Thonny to open the file, click on the "file" option at the top left, and select "save as".



3. Select "Raspberry Pi Pico" and save it in Pico.

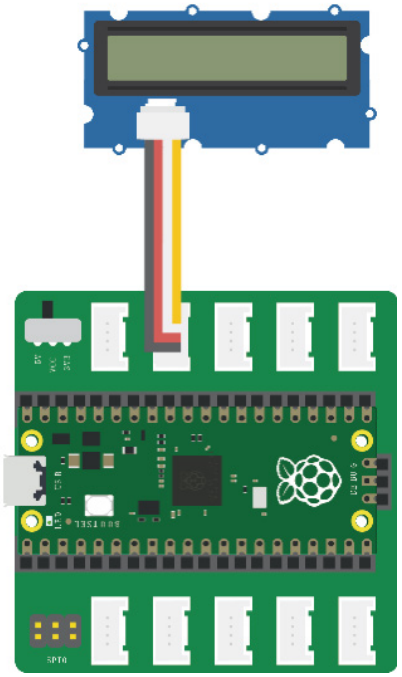


When saving, Thonny asks you to name the file. Here we type "lcd1602.py" as its name. Note that we also need to type the extension of the file, otherwise it cannot be used.



Practice & Operation

Project 1: Display "Hello, world!" with LCD



After adding the library file, let's write a simple program to let the LCD display "Hello, world!".

Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – 16 x 2 LCD

Plug the Pico and the Shield, use a Grove data cable to connect the LCD to I2C1.

Write a Program

First, define the function libraries that we need to use, including the third-party function library lcd1602 that has just been saved in the Pico.

```
from lcd1602 import LCD1602
from machine import I2C,Pin
from utime import sleep
```

The functions available in the lcd library are as follows:

- display() — Enable the display.
- no_display() — Disable the display.
- clear() — Clear the current display and return the cursor.
- setCursor(col, row) — Set the display position, "col" is the number of columns, "row" is the number of rows.
- print(text) — Display characters.

Next, define the pin. Unlike other pin definitions, pin I2C needs to define data lines SCL and SDA respectively.

```
i2c = I2C(1,scl=Pin(7), sda=Pin(6), freq=400000)
```

It is written as "machine.I2C(id,*,scl,sda,freq=400000)", where:

- "id" identifies a specific I2C peripheral, and its allowed value depends on the specific port/board.
- "scl" uses the Pin function to specify the pin used for SCL.
- "sda" uses the Pin function to specify the pin used for SDA.
- "freq" is the integer of the maximum frequency of SCL, which is generally adjusted according to the different devices used. Here we set it to 40000.

Finally, use the functions in the library to display "Hello, World" on the LCD, and the program is completed. The complete program is as follows:

```
from lcd1602 import LCD1602
from machine import I2C,Pin
from utime import sleep
i2c = I2C(1,scl=Pin(7), sda=Pin(6), freq=400000)
d = LCD1602(i2c, 2, 16) #label data type, number of lines and
  characters per line

d.display() #enable display
sleep(1)
```

```
 9 d.clear()#clear display
10 d.print('Hello ') #display characters
11
12 sleep(1)
13 d.setCursor(0, 1)#set display position, row and column start from 0
14 d.print('world ')
```

LCD_hello_world.py

Use a USB cable to connect Pico to the computer, click the "run" button to save the program to any location, you can see the program in action.

Project 2: Display Rotary Angle Sensor Reading with LCD

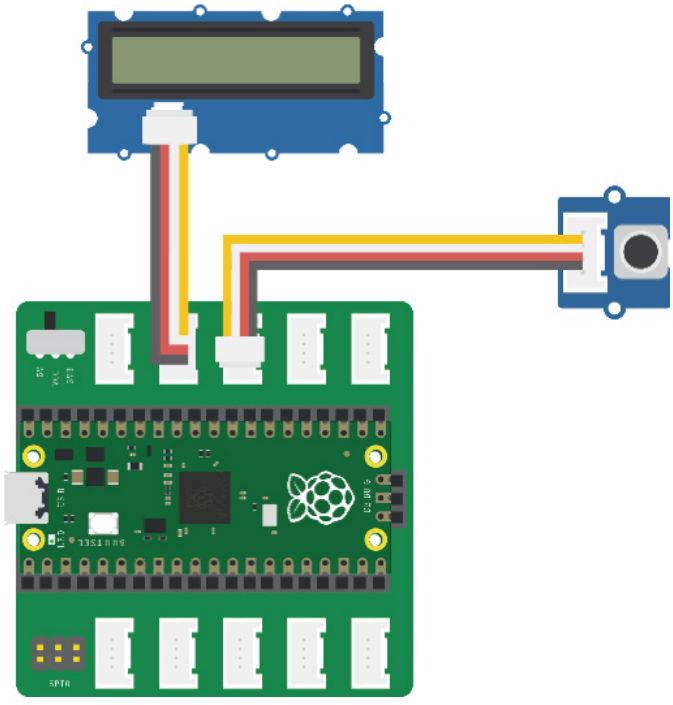
In addition to displaying the characters we have written, the LCD can also display the values returned by other sensors.

Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – 16 x 2 LCD
- Grove – Rotary Angle Sensor

Plug the Pico and the Shield, use a Grove data cable to connect the LCD to I2C1, and connect the Rotary Angle Sensor to A0.



Write a Program

In Lesson 6, we learned how to use ADC to read the Rotary Angle Sensor value. Can we use "d.print()" to display the reading directly? Let's try the following program first:

```
1  from lcd1602 import LCD1602
2  from machine import I2C,Pin,ADC
3  from utime import sleep
4
5  ROTARY_ANGLE_SENSOR = ADC(0)
6  i2c = I2C(1,scl=Pin(7), sda=Pin(6), freq=400000)
7  d = LCD1602(i2c, 2, 16)
8  d.display()
9
10 while True:
11     sleep(1)
12     d.clear()
13     d.setCursor(0, 0)
14     d.print(ROTARY_ANGLE_SENSOR.read_u16())
15     sleep(1)
```

Running the program, Thonny tells us that there is an error in the program, and the error is on line 14. The error message is:

TypeError: 'int' object isn't iterable

This means that the parameter we pass into the print() function is not of the type it can execute.

·Note·

Data Type

In programming, due to the different storage capacities of various data, different types of data need to be matched with the different memory spaces that they are stored in. Therefore, it is important to know about different data types. Every time we create a variable in Python, we are requesting a storage location from memory and storing a certain piece of data in that memory space. In Python, we divide all kinds of data into six basic types:

- Number
- String
- List
- Tuple
- Set
- Dictionary

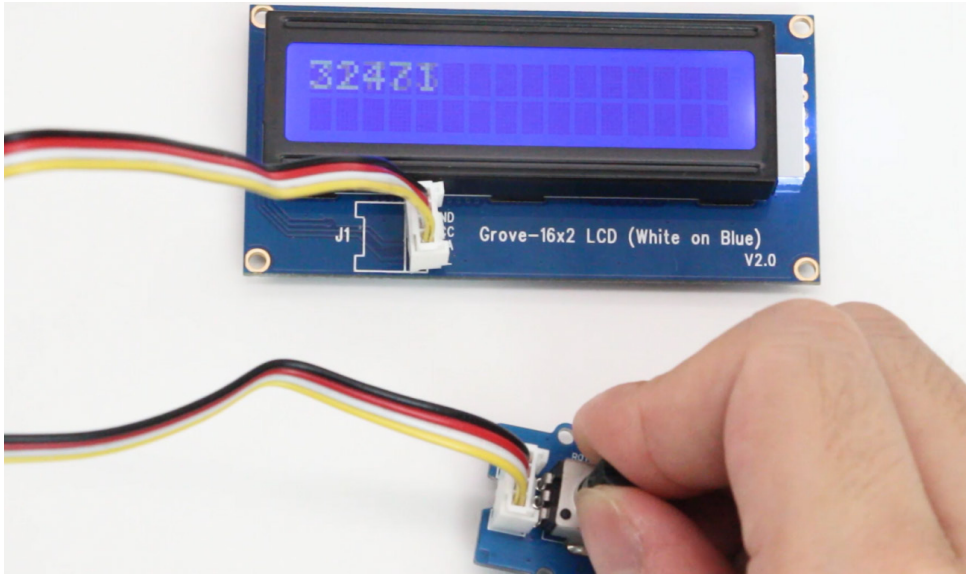
In "print()", we specify in the lcd1602 function library that it can only input parameters whose data type is String, while ROTARY_ANGLE_SENSOR.read_u16() returns integer "int" of Number. If we want to use the print() function to output the value of the Rotary Angle Sensor, we need to convert Number data into String data. There is such a function in Python built-in functions: str(), which can convert other data types into String. Then we can use print() to output the value of the Rotary Angle Sensor.

The revised program is as follows:

```
from lcd1602 import LCD1602
from machine import I2C,Pin,ADC,PWM
from utime import sleep

ROTARY_ANGLE_SENSOR = ADC(0)
i2c = I2C(1,scl=Pin(7), sda=Pin(6), freq=400000)
d = LCD1602(i2c, 2, 16)
d.display()

while True:
    sleep(1)
    d.clear()
    d.setCursor(0, 0)
    d.print(str(ROTARY_ANGLE_SENSOR.read_u16()))
    sleep(1)
```

LCD_konb.py

Use a USB cable to connect Pico to the computer, click the "run" button to save the program to any location, rotate the Rotary Angle Sensor, you can see the actual running effect of the program.



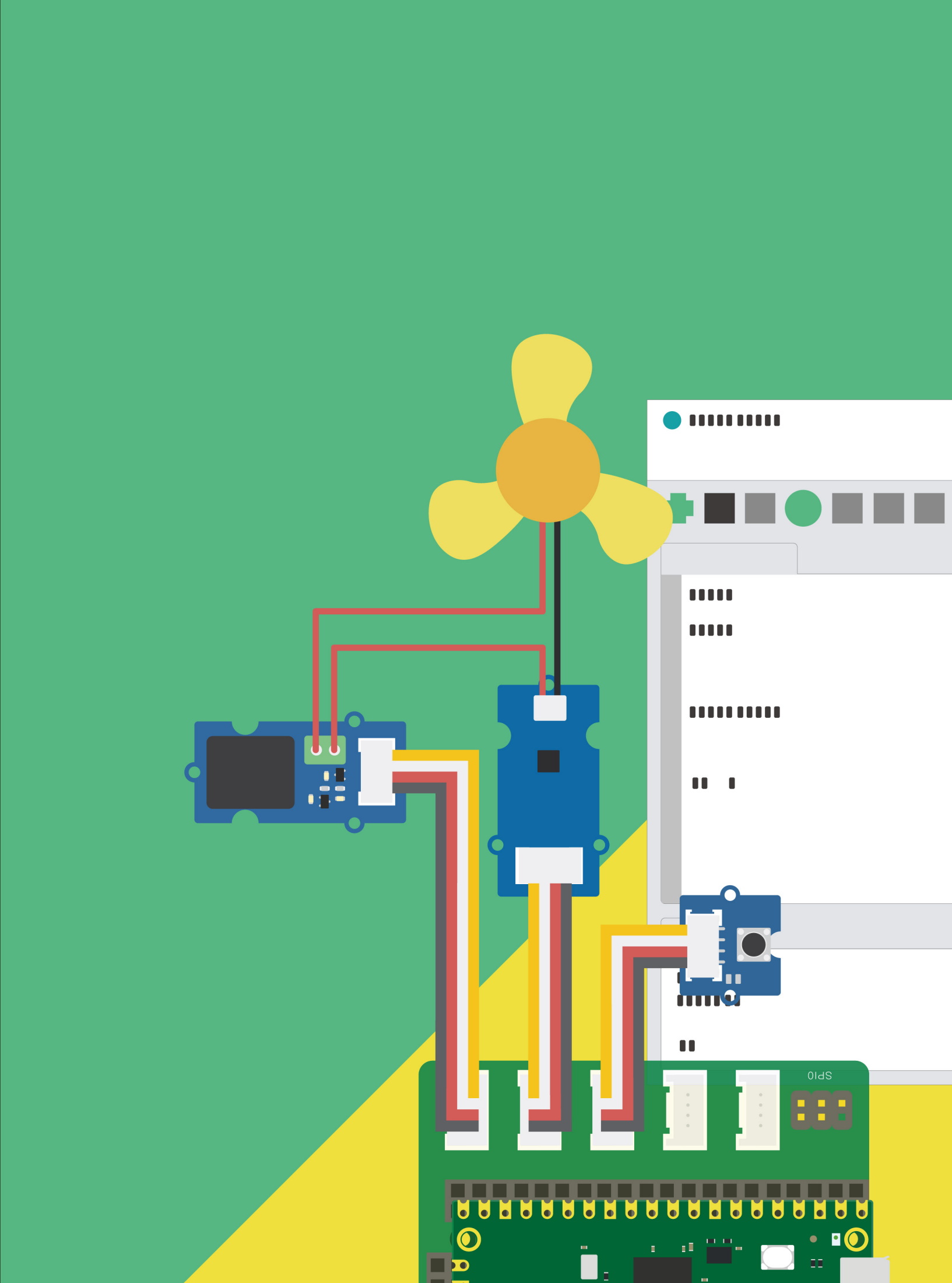
Thinking Expanding

Try to add an LED controlled by PWM into the program used for this lesson, and let the LCD display its brightness.

```
from lcd1602 import LCD1602
from machine import I2C,Pin,ADC,PWM
from utime import sleep

ROTARY_ANGLE_SENSOR = ADC(0)
LED_PWM = PWM(Pin(18))
i2c = I2C(1,scl=Pin(7),sda=Pin(6),freq=400000)
LED_PWM.freq(500)
d = LCD1602(i2c,2,16)
d.display()

while True:
    val=ROTARY_ANGLE_SENSOR.read_u16()
    LED_PWM.duty_u16(val)
    sleep(1)
    d.clear()
    d.setCursor(0,0)
    d.print(str(val))
    sleep(1)
```

LCD_led.py

Unit 3
Project Practice


SPI0

Lesson 10 Temperature and Humidity Monitoring

There are plenty of uses for temperature and humidity monitoring, like weather forecasting. We pay attention to the temperature every day, especially for seasonal changes in weather where we may need to alter our attire according to changes in temperature. In smart agriculture, farmers pay close attention to the growing environment of crops. The most basic thing is to monitor if the temperature of the growing environment is suitable, and whether the soil moisture is sufficient. In the medical field, the most common use of temperature measurements are found in thermometers. In this lesson, we will make an intelligent thermo-hygrograph to monitor surrounding temperature and humidity.

Knowledge Base

Temperature and Humidity Sensor

A temperature and humidity sensor, as its name suggests, is a sensor that can detect temperature and humidity in the environment. There are many kinds of temperature and humidity sensors, among which the most commonly used is the DHT series. This kind of sensor can be calibrated automatically and has a digital output. The sensor includes a capacitive sensor element that can measure relative humidity, and an NTC thermistor that can measure temperature. In this course, we initially chose the DHT11 temperature and humidity sensor. However, DHT11 has been discontinued and upgraded to DHT20. DHT20 uses I2C bus output, and it has higher accuracy and larger measurement range. For the difference between the two, please refer to the following link:https://www.seeedstudio.com/blog/2021/05/25/introducing-dht20-temperature-and-humidity-sensor-and-comparing-it-with-dht11/

DHT11

DHT20

·Note·

If you are using the DHT20 module, please refer to the section: Thinking Expanding.

Logical Operators

In previous lessons, we have learned and used common operators, such as the comparison operators, arithmetic operators, and "and" logic operator, that is, Boolean "and". First, let's review these common logical operators.

Operator	Logical expression	Description
and	x and y	Boolean "and" — if x is False, it returns False; otherwise it returns the computed value of y.
or	x or y	Boolean "or" — if x is not 0, it returns the value of x; otherwise it returns the computed value of y.
not	not x	Boolean "not" — if x is True, it returns False; if x is False, it returns True.

In this lesson, we will use a new logical operator Boolean "or". An example is as follows:

```
1 if x or y :
2    print "variables x and y are both true, or one of them is true"
3 else:
4    print "variables x and y are not true"
```

The output result of the above example is "variables x and y are both true, or one of them is true". Similarly, we can use Boolean "or" to judge whether the temperature or humidity in the environment is within an appropriate range. If one of them fails to meet the requirements, we will sound an alarm.

Practice & Operation

Project 1: Display Temperature and Humidity Value with LCD

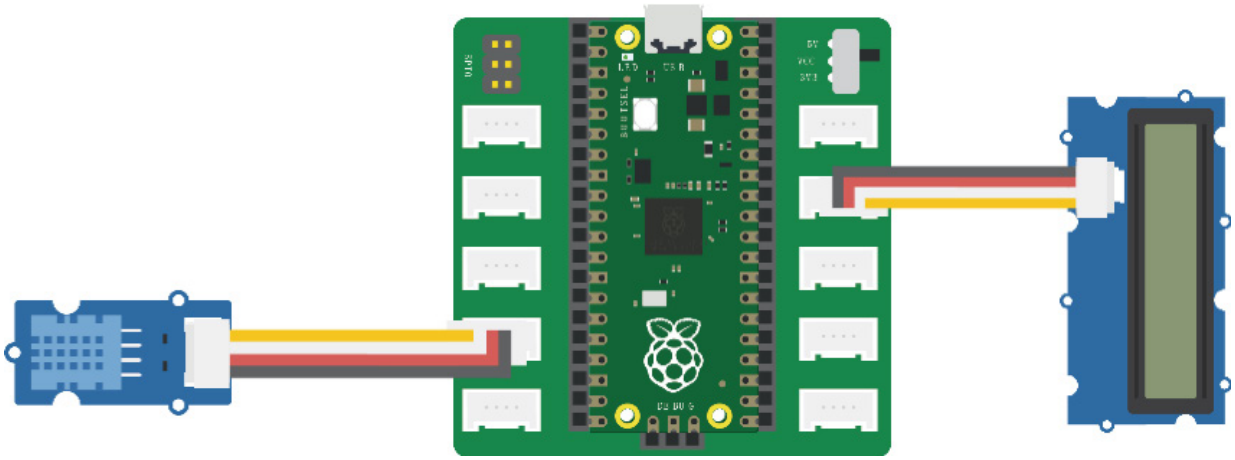
In this project, we will use the Shield to connect both the temperature and humidity sensor and LCD module, and use the Pico to display the temperature and humidity value obtained by the sensor on the LCD. First of all, we need to use the LCD to display the value of temperature and humidity.

Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – Temperature & Humidity Sensor
- Grove – 16 x 2 LCD

Connect Pico and the Shield, use a Grove data cable to connect the LCD to I2C1, and connect the Temperature & Humidity Sensor to D18.



Write a Program

In this project, we need to use not only the third-party function library lcd1602 which has been saved in Pico, but also a new library dht11.py(3kB) Download the library file in the same way as in previous lessons and save it in Pico. After that, we can start to import the libraries that we need to use.

```
from lcd1602 import LCD1602
from dht11 import *
from machine import I2C,Pin,ADC
from utime import sleep
```

Recall how we defined the I2C pin in the last lesson. To achieve I2C, we must define both the SCL and SDA data lines and mark the data type, number of lines and the number of characters in each line. Then, we add the pin definitions for the temperature and humidity sensor.

```
i2c = I2C(1,scl=Pin(7), sda=Pin(6), freq=400000)
d = LCD1602(i2c, 2, 16)
d.display()
dht = DHT(18)
```

Next, we will read the values detected by the temperature and humidity sensor and save them in the variables "temp" and "humid" respectively.

```
 temp,humid = dht.readTempHumid()
```

Finally, we will display the read values on the LCD. There are two values, one each for temperature and humidity, so their labels should be displayed on the screen at the same time to distinguish them. The effect is as follows:

Temp: 25

Humid: 40

The characters of "Temp" and "Humid" can be directly displayed on the LCD, while the values read by "temp" and "humid" cannot be directly displayed on the LCD. So, we need to use the "str()" function to convert them into the String data type. It should also be noted that for temperature and humidity to be displayed on different lines, the coordinate position needs to be set as shown below:

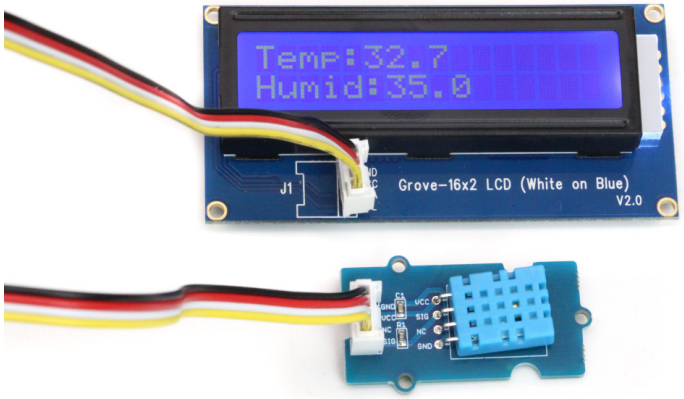
```
d.setCursor(0,0)
d.print("Temp:"+str(temp))
d.setCursor(0,1)
d.print("Humid:"+str(humid))
```

The complete program is as follows:

```
from lcd1602 import LCD1602
from dht11 import *
from machine import I2C,Pin,ADC
from utime import sleep

i2c = I2C(1,scl=Pin(7), sda=Pin(6), freq=400000)
d = LCD1602(i2c, 2, 16)
d.display()
dht = DHT(18)

while True:
    temp,humid = dht.readTempHumid()#temp:  humid:
    sleep(1)
    d.clear()
    d.setCursor(0,0)
    d.print("Temp:"+str(temp))
    d.setCursor(0,1)
    d.print("Humid:"+str(humid))
    sleep(1)
```

temp_humid.py

Use a USB cable to connect Pico to the computer, click the "run" button to save the program to any location, and you can see program in action.

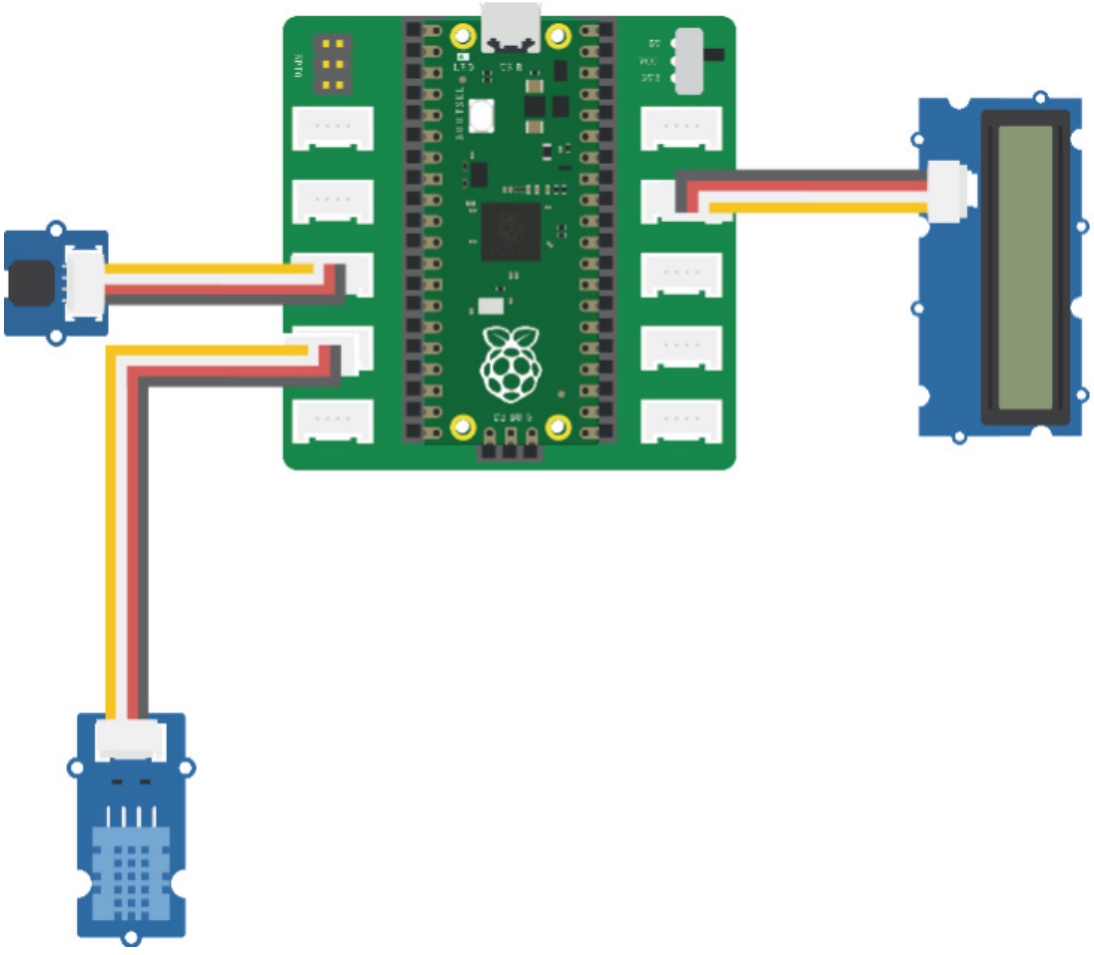


Project 2: Adding an Alarm Function

In Project 1, we developed the function of displaying the temperature and humidity value on the LCD. We can now also further develop the functions of monitoring and alerting by adding a buzzer.

Hardware Connection

In this project, we will use the Grove – Passive Buzzer as the alarm device. Connect the buzzer to D16.



Write a Program

Expanding from Project 1, we only need to add code to judge the temperature and humidity and control the buzzer for the alarm behaviour. To do this, we need to use the logical operator. Logical operators can connect two or more simple statements to form more complex statements. We often use them in all kinds of conditional judgment statements. Here, we need to use Boolean "or". When the temperature "temp" is greater than 30 or the humidity "humid" is less than 30, we would like the buzzer to give an alarm; otherwise, do nothing. To achieve this, we will use the "if...else..." conditional statement. To make the buzzer sound, recall that we can use the freq() function and duty_u16() function to define the tone and volume, and use "buzzer.duty_u16(0)" to turn off the buzzer.

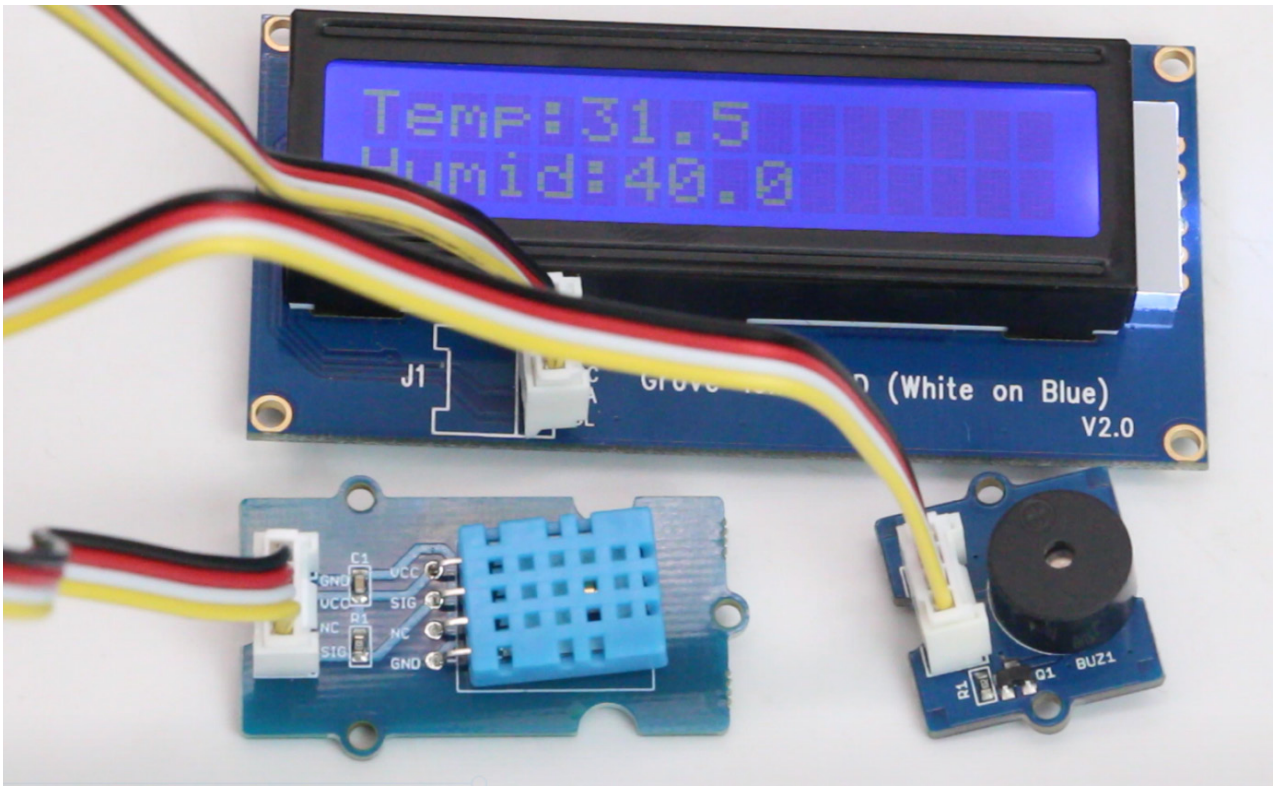
```
if temp >30 or humid <30:
    buzzer.freq(1000)
    buzzer.duty_u16(1000)
else:
    buzzer.duty_u16(0)#close
```

The complete program is as follows:

```
from lcd1602 import LCD1602
from dht11 import *
from machine import I2C,Pin,ADC,PWM
from utime import sleep

i2c = I2C(1,scl=Pin(7), sda=Pin(6), freq=400000)
d = LCD1602(i2c, 2, 16)
d.display()
dht2 = DHT(18)
buzzer = PWM(Pin(16))

while True:
    temp,humid = dht2.readTempHumid()#temp:  humid:
    sleep(1)
    d.clear()
    d.setCursor(0,0)
    d.print("Temp:"+str(temp))
    d.setCursor(0,1)
    d.print("Humid:"+str(humid))
    sleep(1)

    if temp >30 or humid <30:
```

```
23          buzzer.freq(1000)
24          buzzer.duty_u16(1000)
25      else:
26          buzzer.duty_u16(0)#close
```

temp_humid_alarm.py

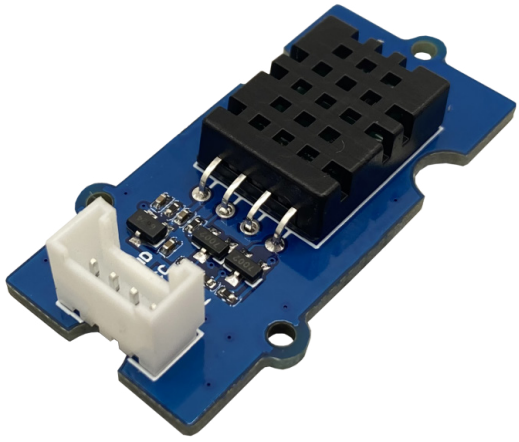


Use a USB cable to connect Pico to the computer and click the "run" button to save the program to any location. Hold the temperature and humidity sensor with your hand, and you should see the changing values of temperature and humidity on the LCD. Give it a try. Will the buzzer sound an alarm?

⭐ Thinking Expanding

DHT20 Temperature and Humidity Sensor

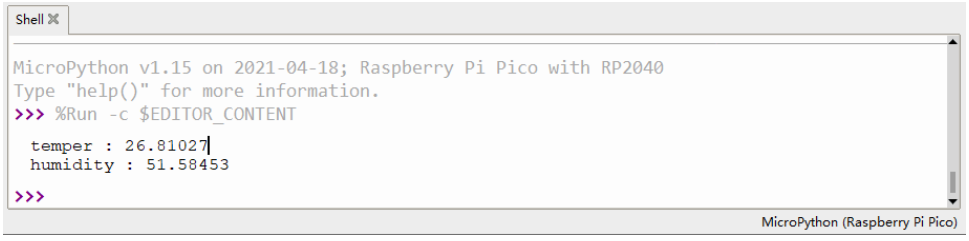
Compared with DHT11, the DHT20 Temperature and Humidity Sensor has a clear advantage in supply voltage, measurement range of temperature and humidity, precision and quality of output signal. It is a product with low power consumption, high precision and high stability, equipped with a fully calibrated digital I2C interface, which is the biggest difference between DHT20 and DHT11.



Take note that they have slightly different definitions of pins. Download dht20.pylibrary(2kB), use DHT20 to detect the temperature and humidity in the environment, and print the value to the Shell area. With the "dht20.py" library, we can use the "dht20_temperature()" and "dht20_humidity()" functions to read the temperature and humidity values returned by the temperature and humidity sensors respectively, as shown below.

```
from machine import I2C
from dht20 import DHT20
i2c = I2C(0)
dht20 = DHT20(i2c)
while True:
    temper = dht20.dht20_temperature()
    humidity = dht20.dht20_humidity()
    print("temper : " + str(temper))
    print("humidity : " + str(humidity))
```

You can see the temperature and humidity values detected by the DHT20 temperature and humidity sensor in the Shell area.



Next, try to complete the program for both Project 1 and Project 2. For reference, the complete code is as follows:

Project 1: display temperature and humidity value with LCD

dht20_tem_hum.py

Project 2: add alarm function

dht20_alarm.py

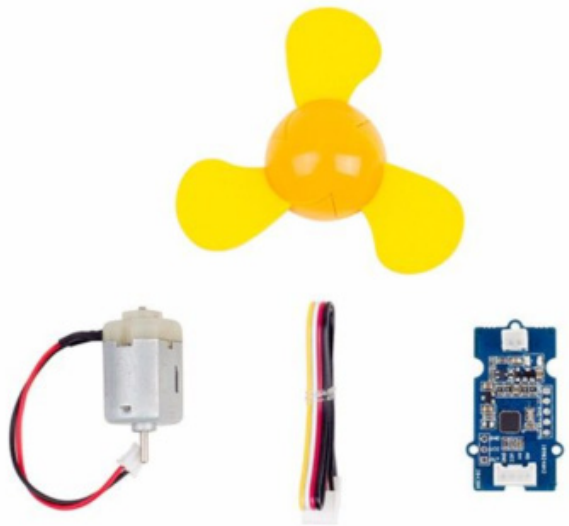
Lesson 11 Intelligent Fan

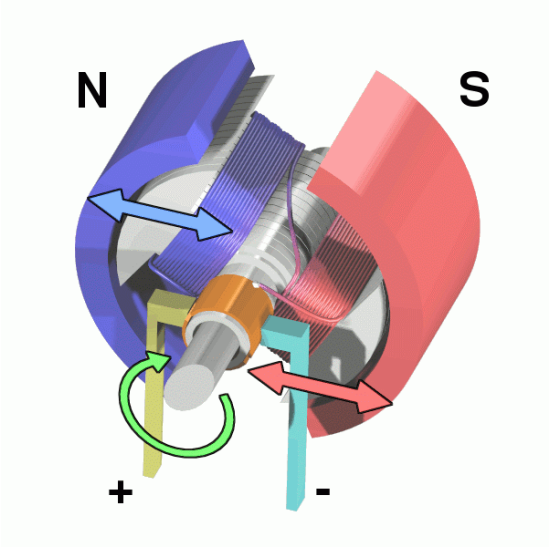
Although air conditioning is now the people's first choice for cooling, fans have always been popular in daily life due to their portability, energy-saving characteristics and availability. In this lesson, we will make an intelligent fan that can be automatically controlled according to the surrounding temperature.

Knowledge Base

Mini Fan

To make the intelligent fan, we will use the Grove – Mini Fan, which consists of a motor drive board, DC motor, film blade and Grove connecting cables, as shown in the figure below. In general, motors can be divided into DC motors and AC motors according to the different input currents. Here, we use a DC motor. DC motors can realize the mutual conversion of electrical energy and mechanical energy so that electricity supplied to the motor can be used to drive other mechanical components.





A DC motor is composed of a stator and a rotor. The main function of the stator is to produce a magnetic field, and the main function of the rotor is to produce electromagnetic torque and induced electromotive force. In short, the DC motor works according to the principle that a conductor which carries an electrical current experiences a force in the magnetic field. Motors are widely used in our daily life, such as in industry, agriculture, and other applications where power is needed.

Practice & Operation

Project 1: Control Fan On and Off with Button

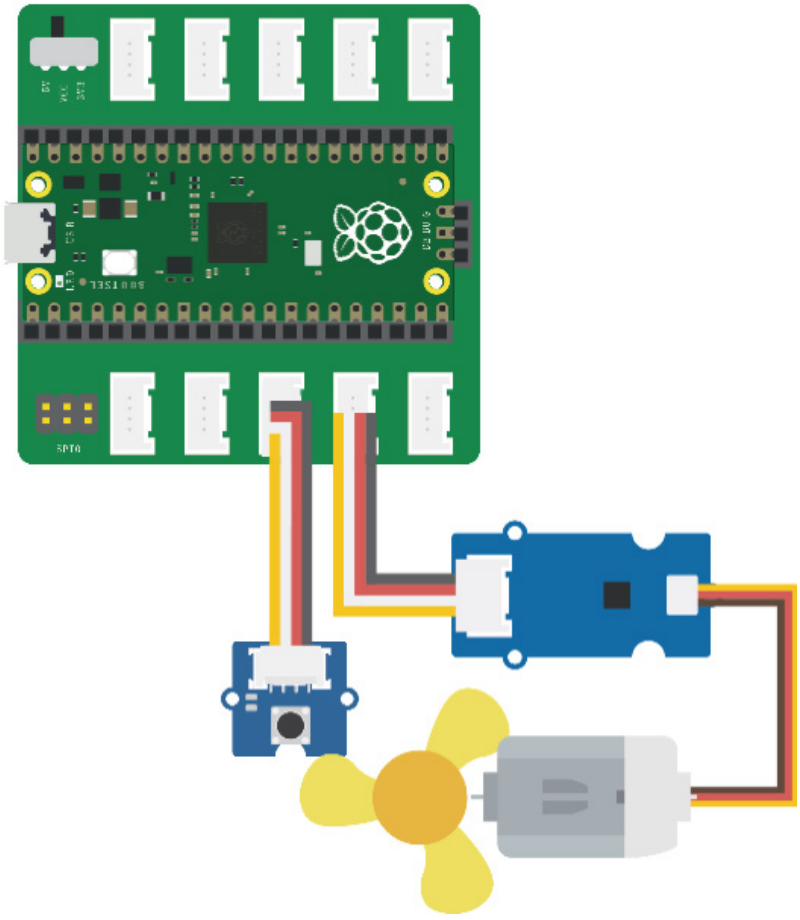
First, we will try to control the fan rotation with the press of a button and achieve manual control of the fan. At this point, you can try to recall the program that we worked on to control an LED on and off with the button, as they have the same logic.

Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – Mini Fan
- Grove – Button

Use a Grove data cable to connect the Button to D16 on the Grove Shield, and connect the Mini Fan to D18.



Write a Program

Similarly, we have to first import the required libraries to set the pins for the button and the fan. When the button is pressed, the return value is 1 and the fan turns on; otherwise, the fan is turned off. The complete code is as follows:

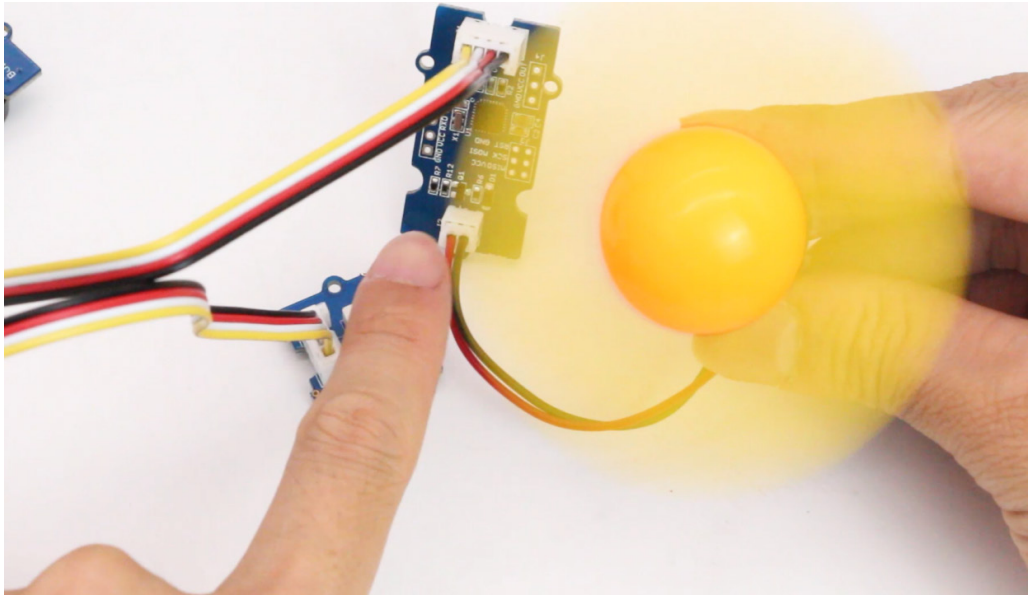
```
from machine import Pin
BUTTON = machine.Pin(16,machine.Pin.IN)
miniFan = machine.Pin(18,machine.Pin.OUT)

while True:
    val = BUTTON.value()
    if val == 1:
        miniFan.value(1)
    else:
        miniFan.value(0)
```

You might realise that when the fan turns on when the button is pressed, and off when the button is released. However, this is not in line with our usage habits. Generally, after we turn on the fan, we don't expect to have to continue pressing the button all the time. In the previous lessons, we also learned how to create variables to solve the similar problem encountered in achieving control of the LED with a button. In this lesson, we will instead introduce another method, using the toggle() function to switch the fan between the high and low levels. When you press the button, the fan will be switched to high; when you press the button again, it will be switched to the low level, so as to "toggle" the fan on and off.

```
import machine
import utime
button = machine.Pin(18, machine.Pin.IN)
miniFan = machine.Pin(16, machine.Pin.OUT)
val = 0
while True:
    val = button.value()
    if val == 1:
        miniFan.toggle()
        utime.sleep_ms(100)
```

minifan.py

Use a USB cable to connect Pico to the computer, click the "run" button to save the program to any location, and then test the running effect by pressing the button.



Project 2: Temperature Sensing Intelligent Fan

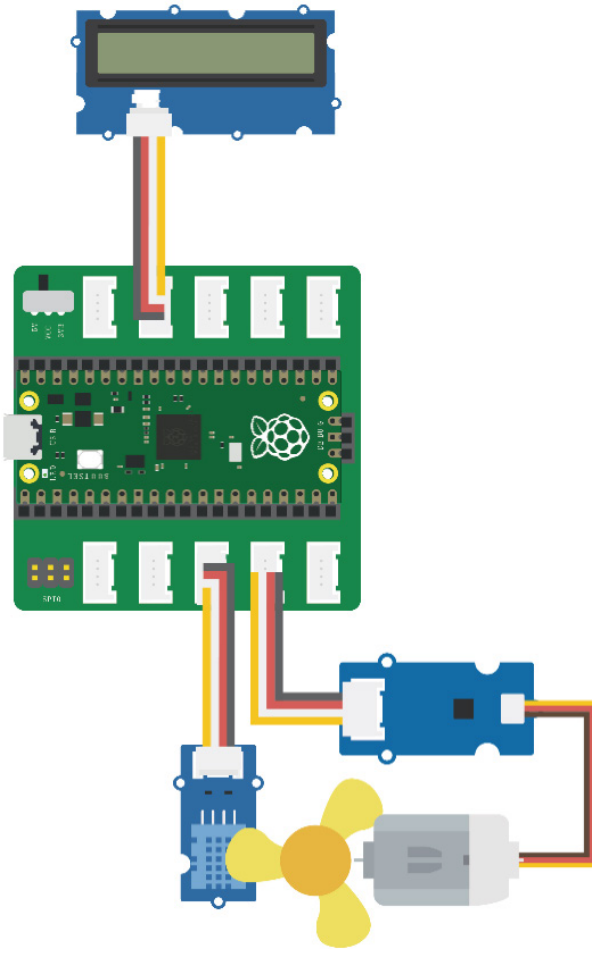
The function of the fan is mainly to reduce the temperature, so we can use the temperature to control the fan on and off. Set a value that represents the maximum comfortable temperature. When the temperature detected is higher than this value, the fan will turn on, and off when the temperature drops below it. At the same time, the temperature value can be displayed on the LCD for monitoring.

Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – Mini Fan
- Grove — Temperature & Humidity Sensor
- Grove – 16 x 2 LCD

Use a Grove data cable to connect the Temperature & Humidity Sensor to D16, connect the Mini Fan to D18, and connect the LCD to I2C1.



Write a Program

First of all, import the required libraries and set the pins of the temperature and humidity sensor, LCD and mini fan.

```
from lcd1602 import LCD1602
from dht11 import *
from machine import I2C,Pin,ADC
from utime import sleep

i2c = I2C(1,scl=Pin(7), sda=Pin(6), freq=400000)
d = LCD1602(i2c, 2, 16)
d.display()
dht2 = DHT(16)
miniFan = machine.Pin(18,machine.Pin.OUT)
```

Next, read the value detected by the temperature and humidity sensor — here we only need to read the temperature value — and display the value on the LCD.

```
while True:
    temp = dht2.readTemperature()#temp:
    sleep(1)
    d.clear()
    d.setCursor(0,0)
    d.print("Temp:"+str(temp))
    sleep(1)
```

Finally, we judge the temperature value through the "if" conditional judgment statement, with 26 as the maximum comfortable value of temperature, and then use the comparison operator ">" to compare the current temperature with the set temperature. When the current temperature is higher than the set temperature, the fan turns on; otherwise, the fan turns off:

```
if temp > 26:
    miniFan.value(1)
else:
    miniFan.value(0)
```

The complete program is as follows:

```
from lcd1602 import LCD1602
from dht11 import *
from machine import I2C,Pin,ADC
from utime import sleep

i2c = I2C(1,scl=Pin(7), sda=Pin(6), freq=400000)
d = LCD1602(i2c, 2, 16)
d.display()
dht2 = DHT(16)
miniFan = machine.Pin(18,machine.Pin.OUT)

while True:
    temp = dht2.readTemperature()#temp:
    sleep(1)
    d.clear()
    d.setCursor(0,0)
    d.print("Temp:"+str(temp))
    sleep(1)
    if temp > 26:
        miniFan.value(1)
    else:
        miniFan.value(0)
```

temp_minifan.py

If you are using a DHT20 temperature and humidity sensor, refer to the following program:

temp_minifan2.py

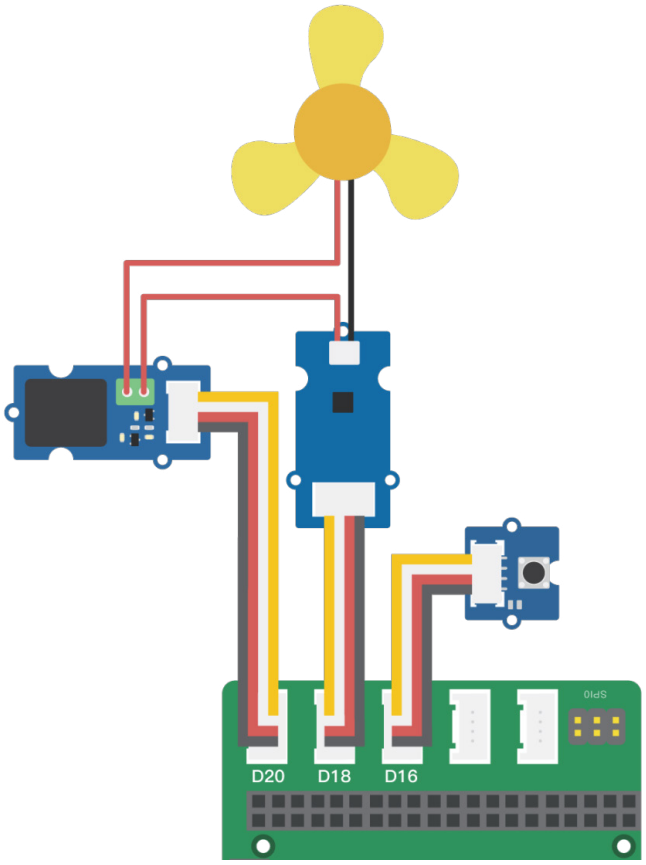
Use a USB cable to connect Pico to the computer, and click the “run” button to save the program to any location. Now, you can see the actual running effect of the program.

In addition to controlling the fan through a button and temperature sensing, what other control methods can you think of? Give it a try.

⭐ Thinking Expanding

The Grove Relay is a digital normally–open switch, which can handle up to 5A current under 250VAC for a long time. It can control simple switch modules such as the LED and Mini Fan. Let's take the Mini Fan as an example. It should be noted that the interface of the Mini Fan is packaged, we need to remove the interface for operation. Once removed, it cannot be installed back. So, this is only for reference.

First, we need to connect the relay to the circuit as shown in the figure below. Connect the positive line of the DC motor (the fan) to the position as shown in the figure, then connect another black negative connecting line of the fan to the negative interface of the motor drive board. Now, find another DuPont wire to connect the positive interface of the motor drive board to the rest of the relay interfaces. Finally, use Grove cables to connect the relay, motor drive board and the button to D20, D18 and D16 interfaces respectively.



After the circuit is connected, we can begin to write the program. We want to write a high output level to the fan and control the switch of the relay through the button, so as to control the opening and closing of the whole circuit. The button should be used to toggle the fan's circuit between on and off.

```
from machine import Pin
from time import sleep
button = Pin(16,Pin.IN)
fan = Pin(18,Pin.OUT)
relay = Pin(20,Pin.OUT)

while True:
    fan.value(1)
    while button.value() == 1:
        relay.toggle()
        sleep(1)
```

relay_fan.py

Lesson 12 Intelligent Light

Light is an essential electrical appliance in life. Since Edison invented the electric light bulb in 1879, the technology of electric light has been developing and becoming more and more intelligent. We can switch a light on or off, control its brightness or color through a button; or similarly, control them through touch or a knob. In addition to controlling lights through these artificial ways, there are also intelligent ways, such as controlling them through sound, light and other environmental signals so as to bring convenience to our lives.

Knowledge Base

Sound Sensor and Light Sensor

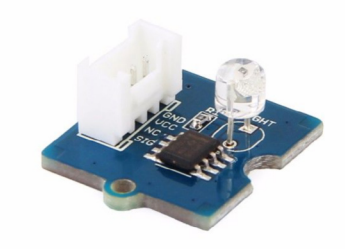
The realization of sound-controlled light and light-controlled light would be impossible to achieve without the help of a sound sensor and light sensor. First of all, let's review their functions.

Sound Sensor

The main component of the sound sensor is a microphone, which can detect sound from the environment and convert it into a corresponding analog signal output. It is widely used in mobile phones, tape recorders, sound-controlled lighting, medical equipment, noise detection, etc.

Sound Sensor

The light sensor integrates a high-sensitivity photodiode LS06-S photoresistor to detect the light intensity in the environment. When the light intensity increases, the resistance across the photoresistor decreases, and the output analog quantity increases. It is widely used in mobile phones, laptops, displays, televisions, and other applications where the backlight brightness of the screen needs to be automatically adjusted.

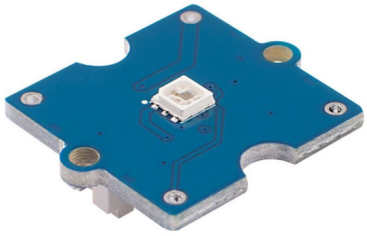


When the sound sensor and the light sensor are combined with lights, sound-controlled and light-controlled intelligent lights can be achieved.In sound-controlled lighting, the sound wave generated in the environment makes the electret film on the sound sensor vibrate, resulting in the change of capacitance, and generating a small voltage. This voltage is then converted into 0-5V voltage and accepted by the controller after ADC, so as to control the light on and off.

In light-controlled lighting, the photoresistor of the light sensor adjusts the resistance value according to the light intensity in the environment. When the light intensity is larger, the resistance value decreases and the analog value of the light sensor output is larger. We can control the light on or off or brightness according to this value.

RGB LED (ws2813 mini)

In this lesson, we will use the RGB LED (ws2813 mini) instead of a simple LED, which can be programmed for a full-color display and different light effects.



When we set RGB LED to display different colors, we only need to set the values of red (R), green (g), and blue (b) respectively. The red, green and blue color channels are divided into 256 levels of brightness, that is, 0-255. The value of 0 is the least bright, which is synonymous with turning off the light, while the most bright at the value of 255. The common color values are listed below:

```
BLACK = (0, 0, 0)
RED = (255, 0, 0)
YELLOW = (255, 150, 0)
GREEN = (0, 255, 0)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)
WHITE = (255, 255, 255)
```

Above, we defined tuples for different colors to store corresponding color data. A new data type is introduced here — tuple. Do you know what it is and how to use it?

List and Tuple

In previous lessons, we have learned data types such as Number and String. In this lesson, we will introduce the other two data types: List and Tuple.

List

List is a common data type in Python. It is a built-in variable sequence to store any number and any type of data set. It is created as follows:

```
List1=[1,2,3,4]
List2=["a","b","c","d"]
List3=['apple','red',3,True]
```

Tuple

Tuple is similar to List. The difference between them is that the element of Tuple cannot be modified, and List uses square brackets while Tuple uses parentheses. Tuple is also created simply by adding elements in parentheses and separating them with commas. After a Tuple is created, we can access the values in the Tuple.

```
tup1= (1,2,3,4)
tup2= "a","b","c","d"
tup3= ('apple','red',3,True)
```

Practice & Operation

Project 1: Control RGB LED to Display Different Colors in Turn

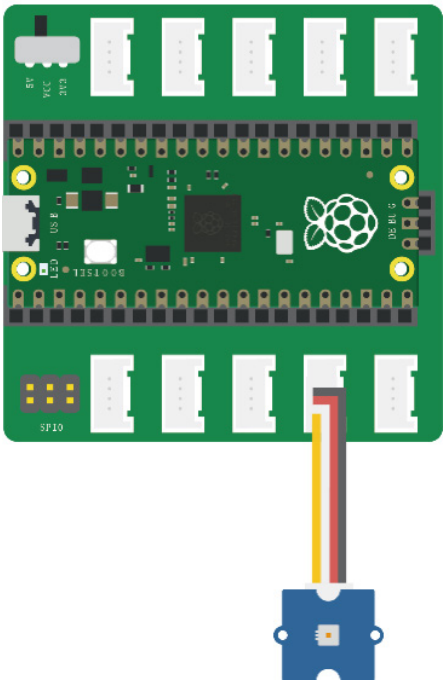
In the Introduction to Electronic Hardware, we mainly used the LED, which only has a single color. We can control the brightness of the light through the program, but we cannot change the color to present more diverse effects. In Project 1, we will control the RGB LED (ws2813 mini) to display different colors in turn.

Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – RGB LED (WS2813 Mini)

Connect the Pico and the Grove Shield, use a Grove data cable to connect the RGB LED to D18.



Write a Program

To interface with the new WS2813 RGB LED module, we will need a new library ws2812.py(2kB) to control the RGB LED. After downloading the library file and saving it in Pico, we can start to introduce the function library we need.

The functions used in the ws2812 library are as follows:

- pixels_fills() — RGB LED fill in color
- pixels_show() — RGB LED show color

First, introduce the function library that we need:

```
from ws2812 import WS2812
import utime
```

Next, define the display color of the RGB LED. We have listed the codes of common colors. If you want other colors, you can find them in the RGB Color Code List.

```
BLACK = (0, 0, 0)
RED = (255, 0, 0)
YELLOW = (255, 150, 0)
GREEN = (0, 255, 0)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)
WHITE = (255, 255, 255)
```

After defining the colors, create a Tuple named COLORS to collect all the colors that need to be displayed.

```
COLORS = (BLACK, RED, YELLOW, GREEN, CYAN, BLUE, PURPLE, WHITE)
```

Finally, define the RGB LED, traverse all the color elements in the COLORS Tuple through the "for" statement, and display them on the RGB LED. In order to display each color clearly, we will buffer them with a delay.

```
led = WS2812(18,1)#WS2812(pin_num,led_count)

while True:
    print("fills")
    for color in COLORS:
        led.pixels_fill(color)#RGB LED fill in color
```

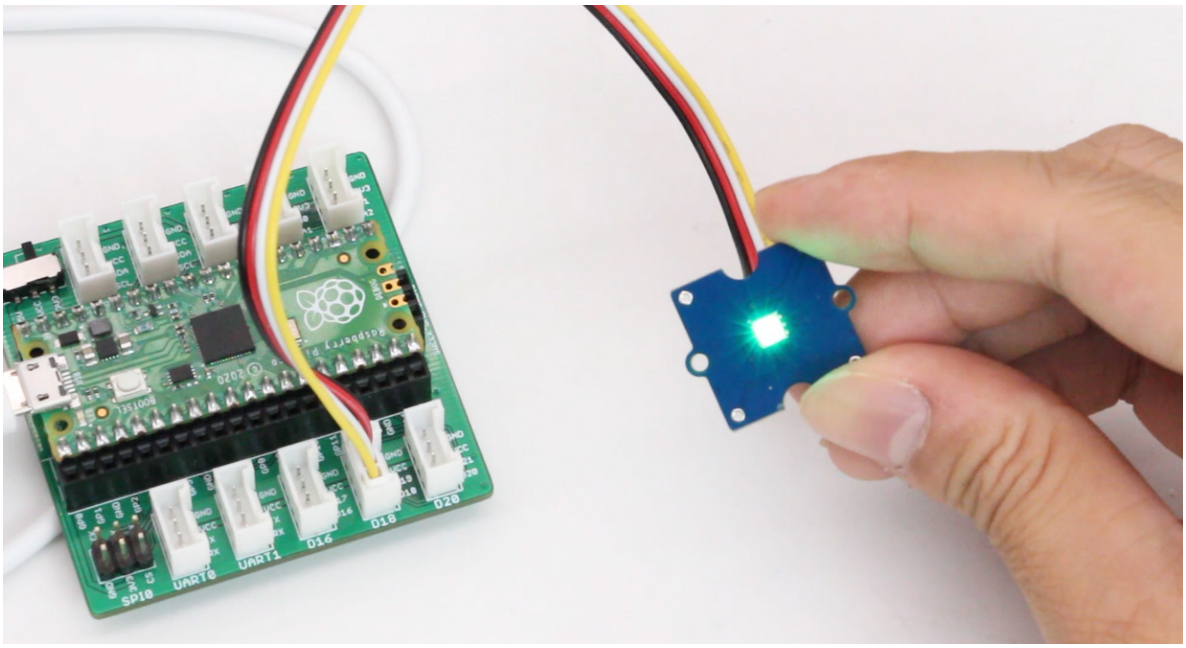
```
        led.pixels_show()#RGB LED show color
        utime.sleep(0.2)#delay
```

The complete program is as follows:

```
from ws2812 import WS2812
import utime

BLACK = (0, 0, 0)
RED = (255, 0, 0)
YELLOW = (255, 150, 0)
GREEN = (0, 255, 0)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)
WHITE = (255, 255, 255)
COLORS = (BLACK, RED, YELLOW, GREEN, CYAN, BLUE, PURPLE, WHITE)

led = WS2812(18,1)#WS2812(pin_num,led_count)

while True:
    print("fills")
    for color in COLORS:
        led.pixels_fill(color)
        led.pixels_show()
        utime.sleep(0.2)
```

rgb_led.py

Use a USB cable to connect Pico to the computer, click the "run" button to save the program to any location. Now, you can see the RGB LED displays different colors.



Project 2: Intelligent Light

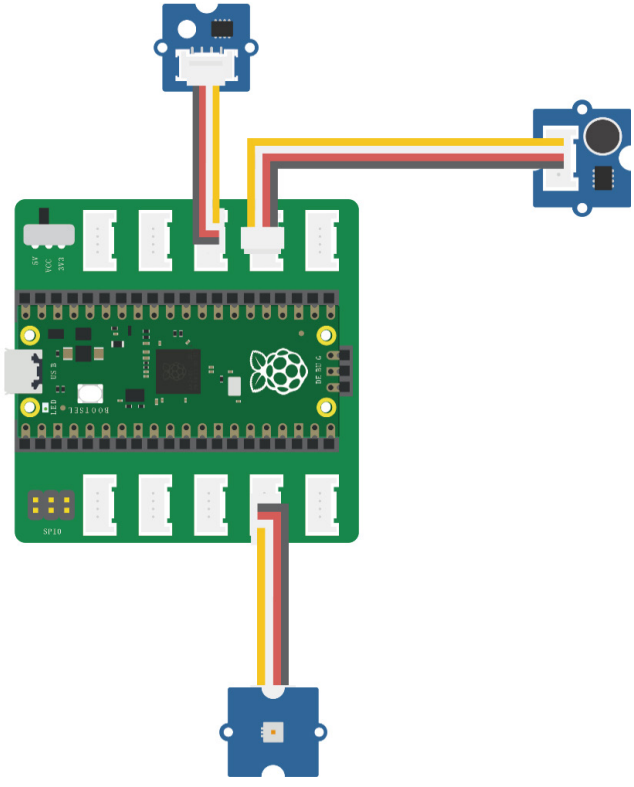
After knowing how to control the RGB LED, we can start to make an intelligent light. The effect we want to achieve is a combination of light control and sound control. When the light brightness is less than a certain value, the light turns on and displays as white; otherwise, the sound sensor will start to work. When the detected analog signal value is greater than a certain value, the red light turns on; otherwise, the green light turns on. Besides lighting, the light can also be used as a noise warning device.

Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – RGB LED (WS2813 Mini)
- Grove – Light Sensor
- Grove – Sound Sensor

Connect the Pico and the Shield, use a Grove data cable to connect the light sensor to A0, sound sensor to A1, and RGB LED to D18.



Write a Program

First, introduce the library, and define the pins of the light sensor, the sound sensor and the RGB LED.

```
from ws2812 import WS2812
from machine import I2C,Pin,ADC
from utime import sleep

led = WS2812(18,1)
LIGHT_SENSOR = ADC(0)
SOUND_SENSOR = ADC(1)
```

Next, we use read_u16() to read the values of the light sensor and the sound sensor, store them in the "light" and "noise" variables respectively. Then, we use the print() function to display the values in the Shell area.

```
light = LIGHT_SENSOR.read_u16()
noise = SOUND_SENSOR.read_u16()
print(light)
```

The figure below shows the data collected by the light sensor. The range of values is between 0–65535. Do you remember why it is 65535? In this program, we use read_u16() to read the analog value returned by the pin, where u16 refers to an unsigned binary integer up to 16 bits. Since the maximum value of the 16-bit binary integer is 1111111111111111, we arrive at 65535 after conversion to decimal form.

However, the value range of RGB LED is 0–255. We will need to divide the value read by 256 to combine the inputs of the light sensor, the sound sensor and the output of the RGB LED. Thus, we will use "if……else" to judge the value detected by light sensor and sound sensor. Here, we set that when the value detected by the light sensor is less than 80, the light turns on, and the displays white; otherwise, the sound sensor starts to work. When the value detected by the sound sensor is greater than 50, the red light turns on for warning; otherwise, the green light turns on.

```
while True:
    light = LIGHT_SENSOR.read_u16()/256
    noise = SOUND_SENSOR.read_u16()/256
    print(noise)
```

```
    if light < 80:
        led.pixels_fill((255,255,255))
        led.pixels_show()
        sleep(0.1)
    else:
        if noise > 50:
            led.pixels_fill((255,0,0))
            led.pixels_show()
            sleep(1)
        else:
            led.pixels_fill((0,255,0))
            led.pixels_show()
            sleep(1)
```

The complete program is as follows:

```
from ws2812 import WS2812
from machine import I2C,Pin,ADC
from utime import sleep

led = WS2812(18,1)
LIGHT_SENSOR = ADC(0)
SOUND_SENSOR = ADC(1)

while True:
    light = LIGHT_SENSOR.read_u16()/256
    noise = SOUND_SENSOR.read_u16()/256
    print(light)
    if light < 80:
        led.pixels_fill((255,255,255))
        led.pixels_show()
        sleep(0.1)
    else:
        if noise > 50:
            led.pixels_fill((255,0,0))
            led.pixels_show()
            sleep(1)
        else:
            led.pixels_fill((0,255,0))
            led.pixels_show()
            sleep(1)
```

Running the program, we find that the effect of the light sensor is as expected, but the effect of the sound sensor appears to be unstable. Amend print(light) to print(noise), and print the value detected by the sound sensor in the Shell area. As shown in the figure below, we find that the value returned by the sound sensor is unstable, and 0 appears frequently. At this time, we will need to process the data by taking 1000 data samples and calculating the average value.

Use "for" to loop the value process 1000 times and calculate the average of the 1000 values. To calculate the average, simply add the values of all the collected data and divide it by the number of samples collected. Here we use the addition assignment operator "+=" to calculate the sum of 1000 sound data, and then divide by 1000. The reference program is as follows:

```
from ws2812 import WS2812
from machine import I2C,Pin,ADC
from utime import sleep

led = WS2812(18,1)
LIGHT_SENSOR = ADC(0)
SOUND_SENSOR = ADC(1)

while True:
    average = 0
    light = LIGHT_SENSOR.read_u16()/256
    for i in range (1000):
        noise = SOUND_SENSOR.read_u16()/256
        average += noise
    noise = average/1000
    print(noise)
```

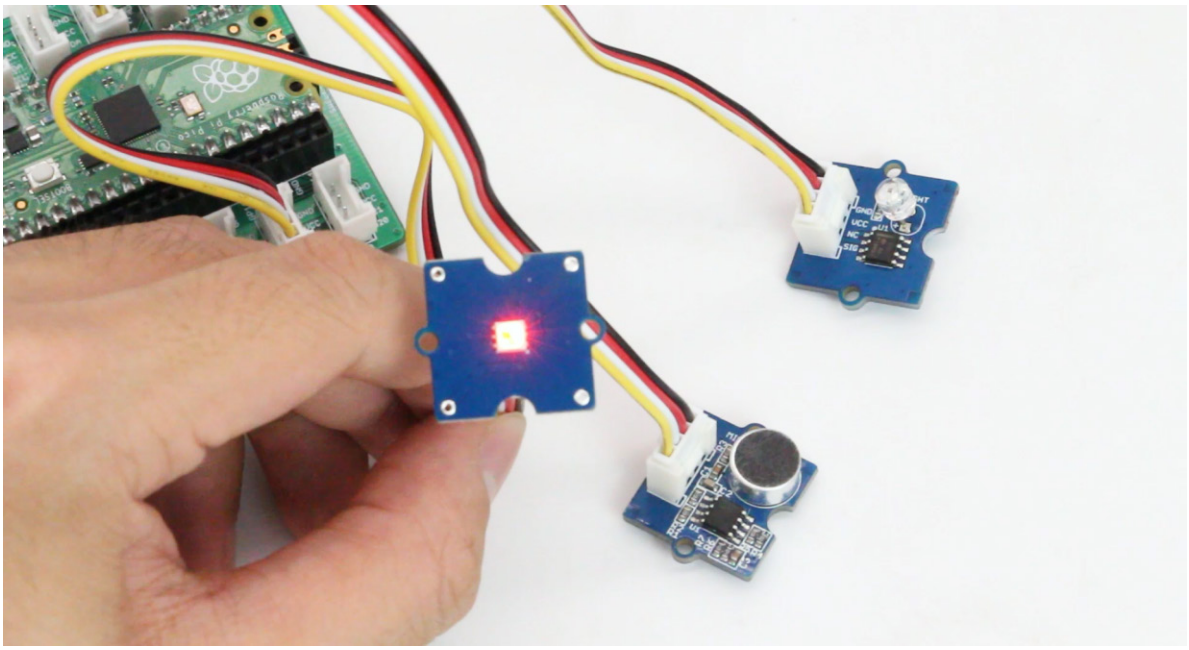
Running this part of the program, we find that the sound value returned in the Shell area is now stable, and we can perform the subsequent conditional evaluation. We can divide the sound range and the corresponding prompt color into red, yellow and green to make a more detailed distinction. The complete program is as follows:

```
from ws2812 import WS2812
from machine import I2C,Pin,ADC
```

```
from utime import sleep

led = WS2812(18,1)
LIGHT_SENSOR = ADC(0)
SOUND_SENSOR = ADC(1)

while True:
    average = 0
    light = LIGHT_SENSOR.read_u16()/256
    for i in range (1000):
        noise = SOUND_SENSOR.read_u16()/256
        average += noise
    noise = average/1000
    print(noise)

    if light < 80:
        led.pixels_fill((255,255,255))
        led.pixels_show()
        sleep(0.2)
    else:
        if noise < 25:
            led.pixels_fill((0, 255, 0))
            led.pixels_show()
            sleep(1)
        if noise >= 25 and noise < 50:
            led.pixels_fill((255, 255, 0))
            led.pixels_show()
            sleep(1)
        if noise >= 50:
            led.pixels_fill((255, 0, 0))
            led.pixels_show()
            sleep(1)
```

smart_light.py

Running the program, we can find that the lighting part and the noise warning part are all successfully realized.



Thinking Expanding

In Project 1, we displayed different colors in turn by controlling the RGB LED. In the process, we called the functions set in the ws2812 library. There are other light effects in this library; let's try to run the following code to see what happens.

```
from ws2812 import WS2812
import time

BLACK = (0, 0, 0)
RED = (255, 0, 0)
YELLOW = (255, 150, 0)
GREEN = (0, 255, 0)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)
WHITE = (255, 255, 255)
COLORS = (BLACK, RED, YELLOW, GREEN, CYAN, BLUE, PURPLE, WHITE)

#WS2812(pin_num,led_count)
led = WS2812(18,1)

print("fills")
for color in COLORS:
```

```
    led.pixels_fill(color)
    led.pixels_show()
    time.sleep(0.2)

print("chases")
for color in COLORS:
    led.color_chase(color, 0.01)

print("rainbow")
led.rainbow_cycle(0)
```

rgb_leds.py

Lesson 13 Automatic Door

An automatic door. You arrive in front of the door and it opens automatically. Isn't it convenient? Well, that's what we're going to achieve this lesson. Looking back, we haven't touched on any electronic module that can detect the human body or actuate the door's motion. At the same time, a typical motor can rotate, but its speed is too fast and difficult to control for opening or closing a door. What other electronic modules should we use to achieve the effect of an automatic door?

Knowledge Base

PIR Motion Sensor

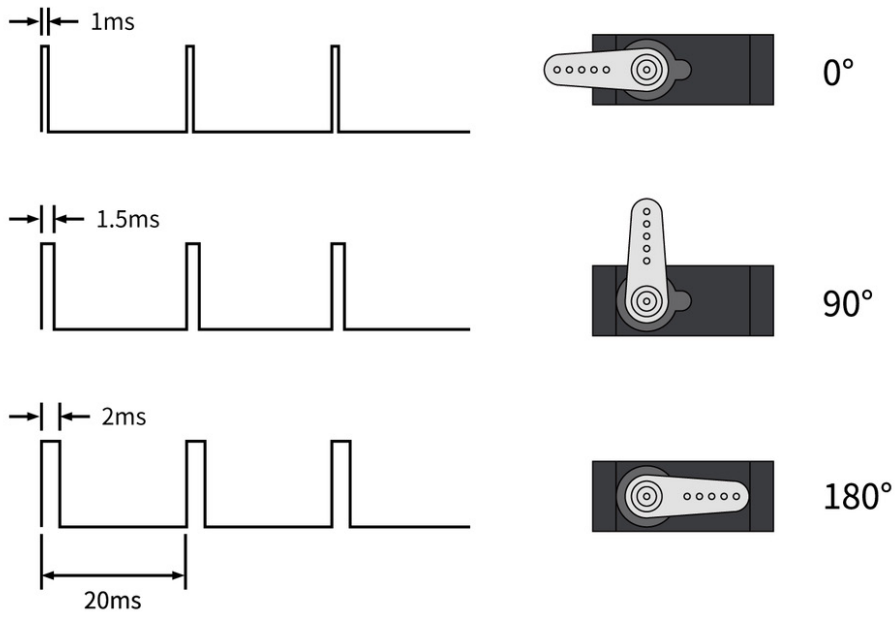
A PIR Motion Sensor is a passive infrared sensor, which is used to detect whether people or animals are moving in its detection range. When the sensor is idle, the infrared intensity emitted by objects within the detection range of the sensor, such as rooms, walls and tables, is almost constant; When people or animals pass by, the infrared intensity detected by the sensor changes. By detecting these changes, the sensor is able to determine whether people or animals are or have passed by. This sensor is often used in the security alarms, motion detection, intelligent switches and robotics. Note that the maximum detection distance of the PIR sensor is 3m to 5m, with an optimal detection distance of about 2m.

Servo

In addition to PIR Motion Sensor the Servo is our indispensable tool for creating an automatic door. A servo is a DC motor with a gear and feedback system, whose position can be controlled accurately. It is also called a servo motor, and often used in control systems that need angular change that can be maintained, such as robotics equipment. In the automatic door project, we can sense whether anyone is approaching through PIR Motion Sensor, send the signal to Servo to control its rotation, and finally drive the door to a fixed position. When people leave, the Servo can then close the door.

Practice & Operation

When we need to control the Servo rotation, we need to control the output to the Servo through a PWM signal. The duty cycle of the PWM directly determines the rotation position of the Servo output shaft. The figure below shows the relationship between different pulse signals and the Servo rotation angle when the frequency is 50 Hz.



However, the pulse of different Servo models may be different. The frequency of our Servo is 100Hz, and the relationship among the corresponding pulse width, rotation angle and duty_u16 value are as follows:

pulse width (ms)	rotation angle (°)	duty_u16
0.61	0°	4000
1.1	45°	7250
1.6	90°	10500
2.6	180°	17000

Next, we will try to control the Servo by PWM signal.

Project 1: Control Servo

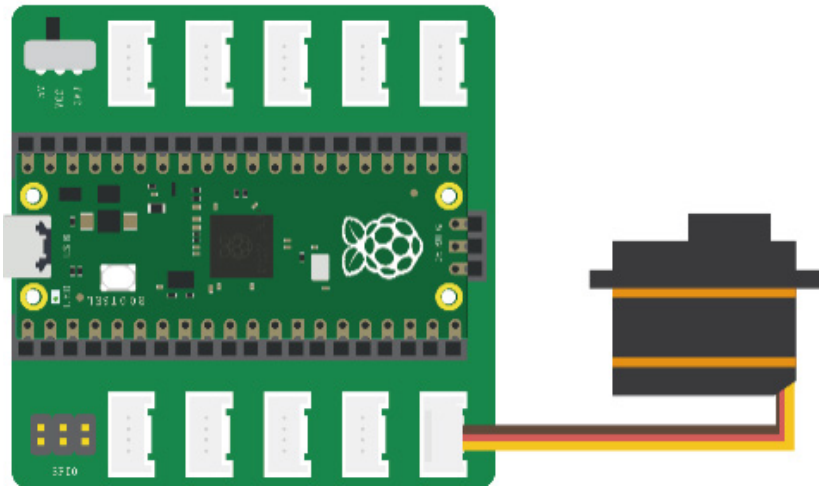
Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico

- Grove Shield for Pi Pico
- Grove – Servo

Use a Grove data cable to connect Servo to D20. According to the table above, we will try to make the Servo turn back and forth between 45° and 90°.



Write a Program

First, define the required libraries and define the Servo pin.

```
from machine import Pin,PWM
from utime import sleep

pwm_servo = PWM(Pin(20))
pwm_servo.freq(100)
```

Next, rotate the Servo from 0° to 180° ten times using a for–loop. The complete program is as follows:

```
from machine import Pin,PWM
from utime import sleep

pwm_servo = PWM(Pin(20))
pwm_servo.freq(100)

for i in range(10):
    pwm_servo.duty_u16(7250)
    sleep(1)

    pwm_servo.duty_u16(10500)
    sleep(1)
```

Running the program, we can see the Servo turns back and forth between 0° and 180° for ten times, which is the effect we want. If we want to control Servo to other angles, we will need to calculate the corresponding value beforehand. If we want to use Servo in other projects, we will also need to redefine the function and the corresponding value. It doesn't seem convenient. For this, we need function libraries. In short, a library file is a file that contains all the functions and variables you define. It can be introduced by other programs to use the functions in the library file. In previous lessons, we used a lot of standard libraries of MicroPython such as machine and utime, and third-party libraries developed by users such as lcd1602 and dht11. In this lesson, we will instead learn to write a simple library file for the Servo to define its pin and rotation angle.

• Define class

In short, a class is used to describe a collection of objects with the same properties and methods. It defines the properties and methods common to all objects in the collection. In Python, classes are defined by using "class", and the class name ends with a colon. For example, in previous lessons, we introduced the Pin class in the machine library, which is the standard library of MicroPython. This class contains all the functions related to Pin. As shown below, we define a class of Servo.

```
1 class SERVO:
```

•Constructors

Next, we use "def __init__()" to add a constructor to the Servo class. It can contain multiple parameters. When we need to refer to a variable in the function, we need to pass in the value of the parameter through the function body itself. Therefore, we need to set the parameter in the function body first. For example, the function of our constructor below is to set a pin as the pin of the output PWM signal, so the parameter we need to introduce in the function body is the pin number. In addition to "pin", we also introduce a parameter called "self". Self represents the instance of a class, not the class itself. This is also the difference between a class function and an ordinary function.

```
1 def __init__(self,pin):
2     self.pin = pin
3     self.pwm = PWM(self.pin)
```

Next, we define the "turn" function to calculate the rotation angle "val" of the Servo. In this function, we output the PWM signal to the Servo through freq(), and output the analog value to the Servo by using the duty_u16() function to control its rotation. Note that the analog value is calculated by the formula: val/180*13000+4000. We need to convert the string or number calculated by the formula into the integer through the int() function.

```
def turn(self, val):
    self.pwm.freq(100)
    self.pwm.duty_u16(int(val/180*13000+4000))
```

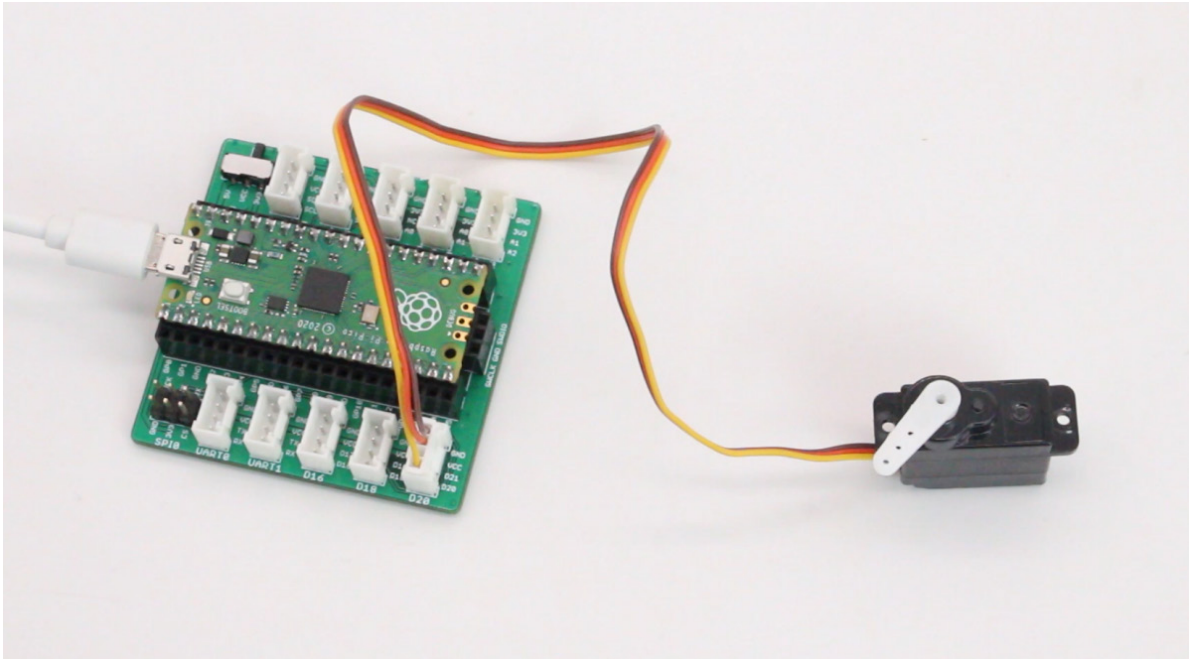
The completed library file is as follows:

```
from machine import Pin, PWM #call machine library
#define a servo object below:
class SERVO:
    #add a constructor to the created class:
    def __init__(self, pin):
        #define two instance variables, pin and PWM:
        self.pin = pin
        self.pwm = PWM(self.pin)
    #define example method of turn to calculate the rotation angle of
Servo:
    def turn(self, val):
        self.pwm.freq(100)
        self.pwm.duty_u16(int(val/180*13000+4000))
```

After completing the library file, we name it "servo.py" and store it in Pico. Then we can call the functions in the library directly. Modifying the program of Project 1 as follows, we can input the rotation angle of Servo, run the program, and watch the magic happen!

```
from machine import Pin
from utime import sleep
from servo import SERVO
servo = SERVO(Pin(20))

for i in range(10):
    servo.turn(20)
    sleep(1)

    servo.turn(160)
    sleep(1)
```

servo.py



Project 2: Control Servo Rotation with PIR Motion Sensor

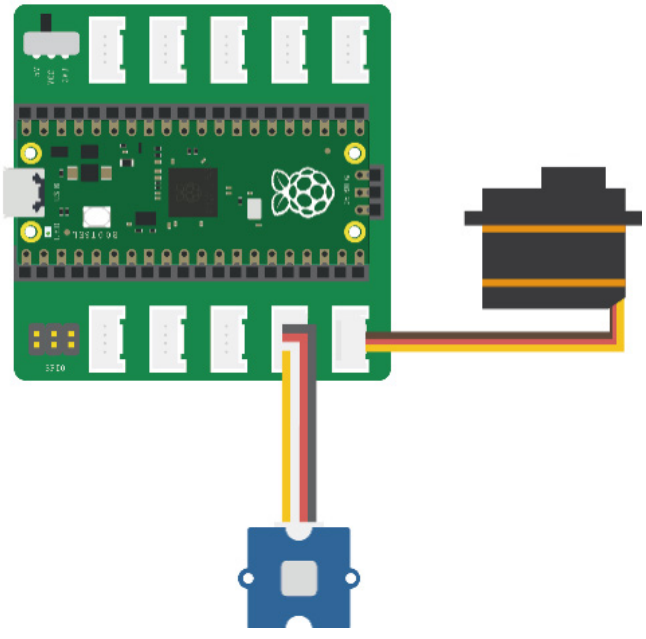
Next, we will use the PIR Motion Sensor to control the Servo to realize the automatic door.

Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove — Servo
- Grove — PIR Motion Sensor

Connect the PIR Motion Sensor to D18, and connect the Servo to D20.



Write a Program

First, introduce the required library, and define the pins of the Servo and the PIR Motion Sensor. Here, we refer to the newly built servo library.

```
from machine import Pin
from utime import sleep
from servo import SERVO
servo = SERVO(Pin(20))
miniPir = Pin(18, Pin.IN)
```

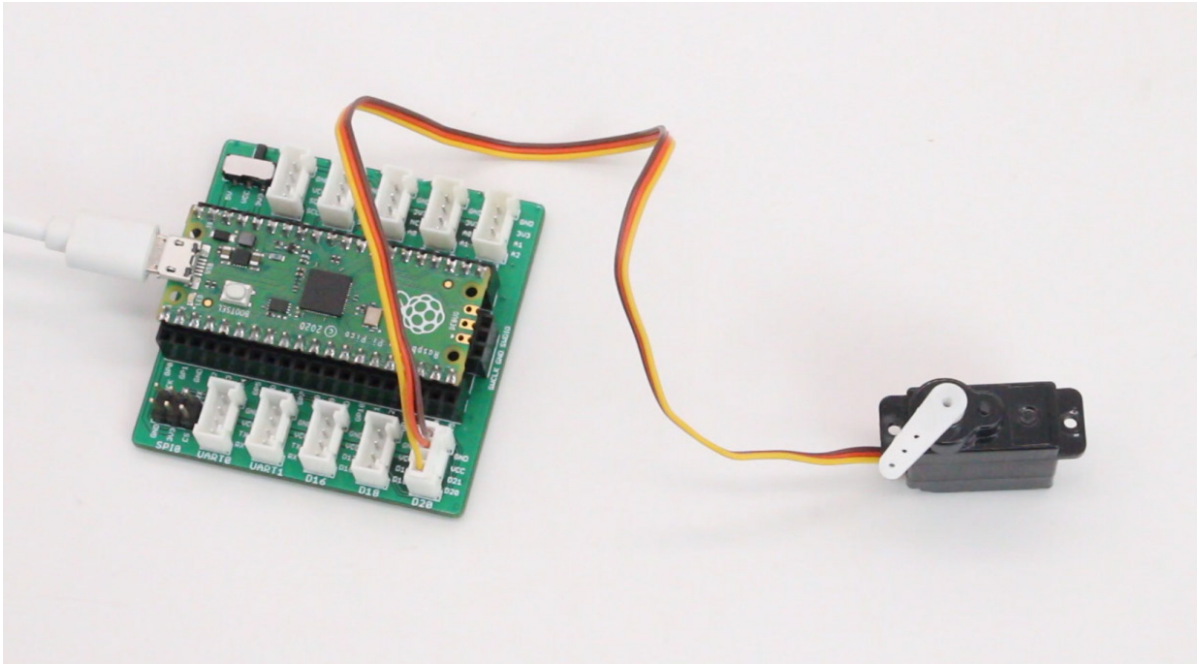
There are only two states of the PIR Motion Sensor, triggered and not triggered. Therefore, we only need to judge whether the returned value is 1. If so, we print "Motion Detected" in the Shell area to indicate that the sensor is triggered.

When the Servo turns to 160°, it means that the door is open. After waiting for 10 seconds, the door will close automatically. The complete program is as follows:

```
from machine import Pin
from utime import sleep
from servo import SERVO
servo = SERVO(Pin(20))
miniPir = Pin(18, Pin.IN)

while True:
    if miniPir.value()== 1:
        print('Motion Detected')
        servo.turn(160)
        sleep(10)

        servo.turn(20)
```

pir_servo.py

Run the program, save it in any location of the computer. Wave your hand around the PIR Motion Sensor to see the effect of the Servo rotation.



Thinking Expanding

In Project 1 and 2, we controlled the Servo rotation to the desired angle by writing the analog value to the Servo pin. Can you try to write a program to make the Servo gradually rotate from 0° to 180° ?

```
from machine import Pin
from servo import SERVO
from time import sleep
servo = SERVO(Pin(20))
a = 180
while True:
    a = a == 0 and 180 or a - 1
    servo.turn(a)
    sleep(0.1)
```

servo_2.py

Lesson 14 Welcome Device

In this lesson, we will summarize the projects of this chapter and explore new projects through different combinations of electronic hardware. We will also prepare for creating your own projects by consolidating the creation and use of software libraries through practice.

Knowledge Base

Through the study of this chapter, we have put electronic hardware and programming knowledge to practice in solving some practical problems. First of all, let's review what projects we have done and what functions we have built!

Project Name	Sensor	Actuator/Display	Project Realization
Temperature and Humidity Monitoring	Temperature & Humidity Sensor	LCD, Buzzer	Detect the environment temperature and humidity, display on the LCD, and give an alarm when it exceeds a certain range.
Intelligent Fan	Temperature & Humidity Sensor	Mini Fan, LCD	Detect the temperature, turn on the fan when it exceeds the comfortable temperature, and display the real-time temperature on the LCD.
Intelligent Light	Light Sensor, Sound Sensor	RGB LED (WS2813 Mini)	Realize light-controlled light by the light sensor. When the brightness is not enough, turn on the light. When the brightness is enough, turn on noise detection, and display the different colors warning lights for different noise levels.
Automatic Door	PIR Motion Sensor	Servo	When detect someone approaching, the Servo turns to open the door. Wait for 10 seconds, the Servo turns to close the door.

There is a lot we can do by combining different sensors and actuators. For example, LEDs can be used as lighting, but also as warning lights or reminder devices; buzzers can be used to play music, but serve as alarm devices. Combining buzzers with LEDs, we can achieve more effective outcomes with both sound and light. Therefore, we should aim to have a more flexible understanding of different electronic modules. Try to expand or transform the projects in the above table and fill in the following:

Project Name	Sensor	Actuator/Display	Project Realization

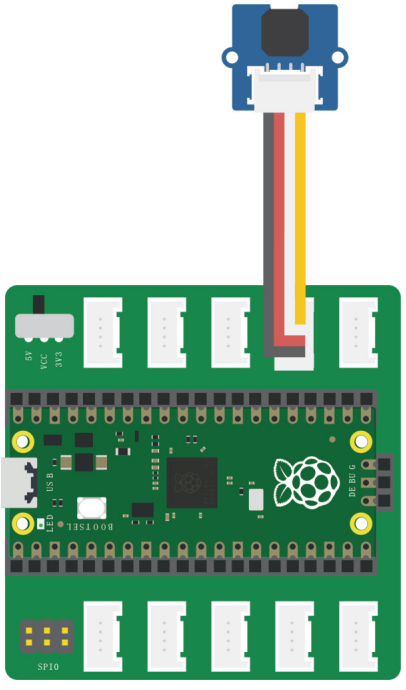
Here's an example: We can add a welcome feature to our automatic door project.

Project Name	Sensor	Actuator/Display	Project Realization
Welcome Device	PIR Motion Sensor	Servo, LCD, Buzzer	When detect someone approaching, the Servo turns to open the door, the LCD displays "welcome", and the buzzer plays the welcome music.

Now, let's work together to realize the project.

Practice & Operation

The difficulty of this project is in programming the buzzer to play the welcome music. Looking back at Lesson 8, we need to adjust the frequency and duty cycle of the PWM signal to control the passive buzzer, and we also need to set the seven basic notes individually. Like before, it would greatly simplify our code to use custom functions for each note. In this lesson, we will further simplify the program by encapsulating the functions of notes into a library.



Project 1: Play Welcome Music with Buzzer

Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico
- Grove Shield for Pi Pico
- Grove – Passive Buzzer

Plug the Pico and the Shield, use a Grove data cable to connect the buzzer to A1.

Write a Program

First, create a buzzer library. In it, we define the PWM pin and seven basic notes of the buzzer as follows:

```
from machine import PWM
#create a Freq tuple,including mute,DO,RE,MI,FA,SO,LA,SI
Freq = (30000,1046,1174,1318,1396,1567,1780,1975,2085)
#define Music object below
class Music:
    #add a constructor to the created class:
    def __init__(self,pwm_pin):
        #define PWM pin variable
        self.pwm = pwm_pin
    #define an example method of Music to play different notes
    def music(self, number):
        self.pwm.freq(Freq[number]) #List Freq[number] contains eight elements from 0 to 7. When we want to play RE, just write music(3)
        self.pwm.duty_u16(5000)
```

After creating the library file, name it "buzzer.py" and save it in the Pico. Now, you can call the buzzer library to play music.

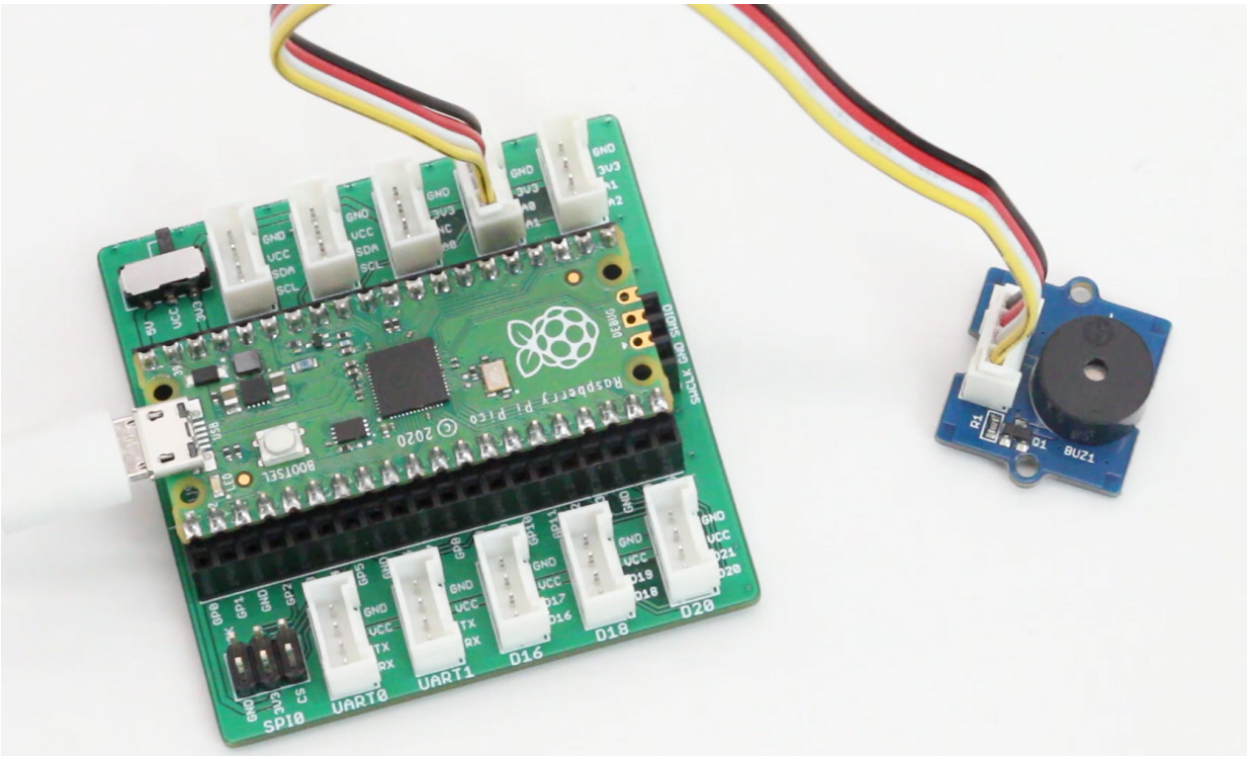
Let's play out a single line of of Jingle Bells. The program is as follows:

```
from machine import Pin,PWM
from buzzer import Music
from utime import sleep

pwm = PWM(Pin(27))
mu = Music(pwm)

while True:
    mu.music(4)
    sleep(0.25)
    mu.music(0)
    sleep(0.1)
    mu.music(4)
    sleep(0.25)
    mu.music(0)
    sleep(0.1)
    mu.music(4)
    sleep(0.5)
    mu.music(0)
    sleep(0.1)

    mu.music(4)
    sleep(0.25)
    mu.music(0)
    sleep(0.1)
    mu.music(4)
    sleep(0.25)
    mu.music(0)
    sleep(0.1)
    mu.music(4)
    sleep(0.5)
    mu.music(0)
    sleep(0.1)

    mu.music(4)
    sleep(0.25)
    mu.music(6)
    sleep(0.25)
```

```
    mu.music(2)
    sleep(0.35)
    mu.music(3)
    sleep(0.15)
    mu.music(4)
    sleep(1)
    mu.music(0)
    sleep(0.1)
```

buzzer_lib.py

Run the program and listen, does it sound right?



Project 2: Welcome Device

Next, we will realize the function of the welcome device. When we detect someone approaching, the Servo should turn to open the door, the LCD should display “welcome”, and the buzzer should finally play the welcome music.

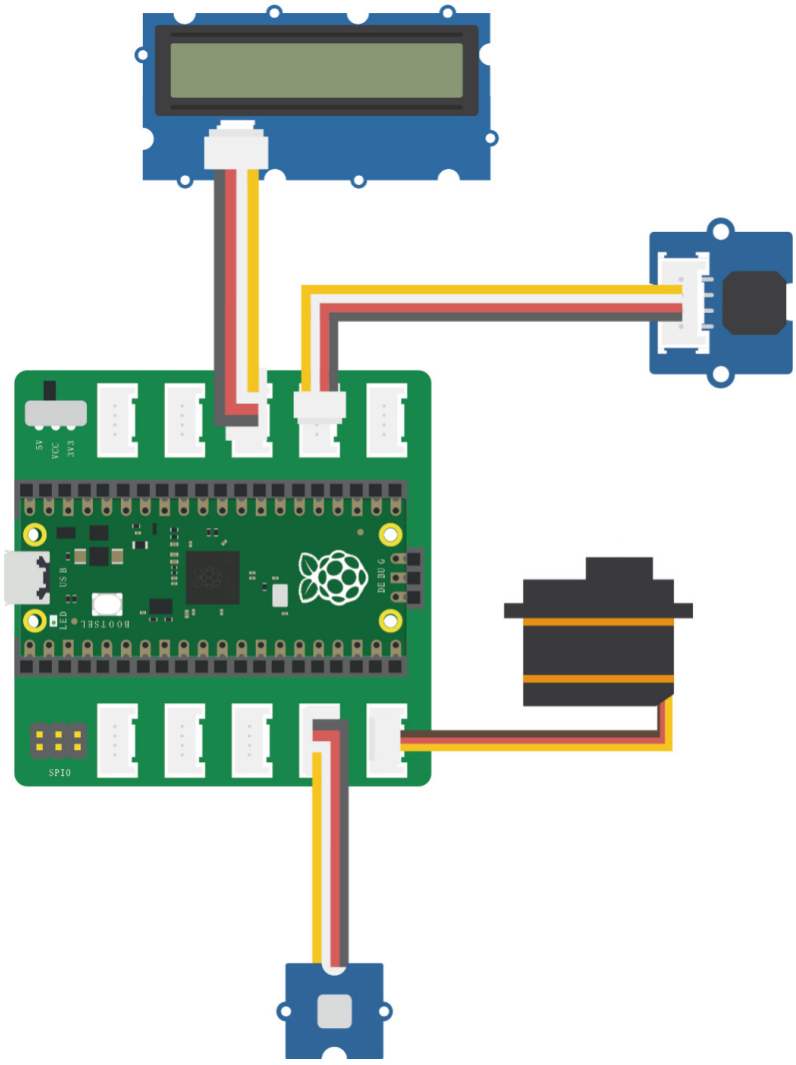
Hardware Connection

In this project, the electronic hardware we need to use is as follows:

- Raspberry Pi Pico

- Grove Shield for Pi Pico
- Grove – Passive Buzzer
- Grove – 16x2 LCD
- Grove – Servo
- Grove – PIR Motion Sensor

Connect the buzzer to A1, connect the PIR Motion Sensor to D18, connect the Servo to D20, and connect the LCD to I2C1.



Write a Program

First, introduce the libraries that we need to use, including the machine and utime standard libraries, lcd1602, servo, and the buzzer library we just created. Define the pins of the modules we want to use.

```
1 from machine import Pin,I2C,PWM
2 from utime import sleep
3 from servo import SERVO
4 from lcd1602 import LCD1602
5 from buzzer import Music
```

```
6
7 servo = SERVO(Pin(20))
8 miniPir = Pin(18, Pin.IN)
9 pwm = PWM(Pin(27))
10 mu = Music(pwm)
11 i2c = I2C(1,scl=Pin(7), sda=Pin(6), freq=400000)
12 d = LCD1602(i2c, 2, 16)
```

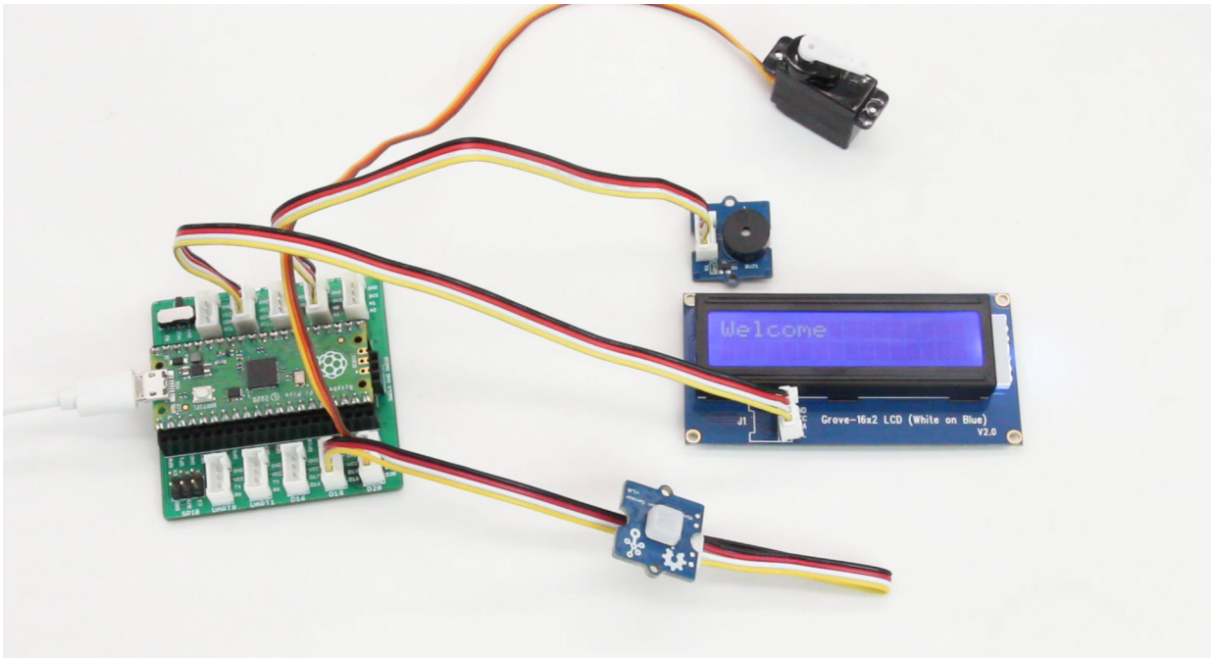
Next, we use the "if" statement to evaluate the state detected by the PIR Motion Sensor. When a person is detected, the Servo rotates 160° to open the door, the LCD displays "welcome", and the buzzer starts to play the welcome music.To achieve our welcome sequence, all we have to do is combine the steps that we have learned in the previous lessons. The complete program is as follows:

```
1 from machine import Pin,I2C,PWM
2 from utime import sleep
3 from servo import SERVO
4 from lcd1602 import LCD1602
5 from buzzer import Music
6
7 servo = SERVO(Pin(20))
8 miniPir = Pin(18, Pin.IN)
9 pwm = PWM(Pin(27))
10 mu = Music(pwm)
11 i2c = I2C(1,scl=Pin(7), sda=Pin(6), freq=400000)
12 d = LCD1602(i2c, 2, 16)
13
14 while True:
15     if miniPir.value()== 1:
16         print('Motion Detected')
17         servo.turn(180)
18         d.display()
19         sleep(1)
20         d.clear()
21         d.print('Welcome')
22
23         mu.music(4)
24         sleep(0.25)
25         mu.music(0)
26         sleep(0.1)
27         mu.music(4)
28         sleep(0.25)
```

```
        mu.music(0)
        sleep(0.1)
        mu.music(4)
        sleep(0.5)
        mu.music(0)
        sleep(0.1)

        mu.music(4)
        sleep(0.25)
        mu.music(0)
        sleep(0.1)
        mu.music(4)
        sleep(0.25)
        mu.music(0)
        sleep(0.1)
        mu.music(4)
        sleep(0.5)
        mu.music(0)
        sleep(0.1)

        mu.music(4)
        sleep(0.25)
        mu.music(6)
        sleep(0.25)
        mu.music(2)
        sleep(0.35)
        mu.music(3)
        sleep(0.15)
        mu.music(4)
        sleep(1)
        mu.music(0)
        sleep(0.1)

        servo.turn(0)
```

welcome.py

After completion, save the program in any location of the computer, move your hand around to the PIR Motion Sensor, and test the effect of the welcome device.

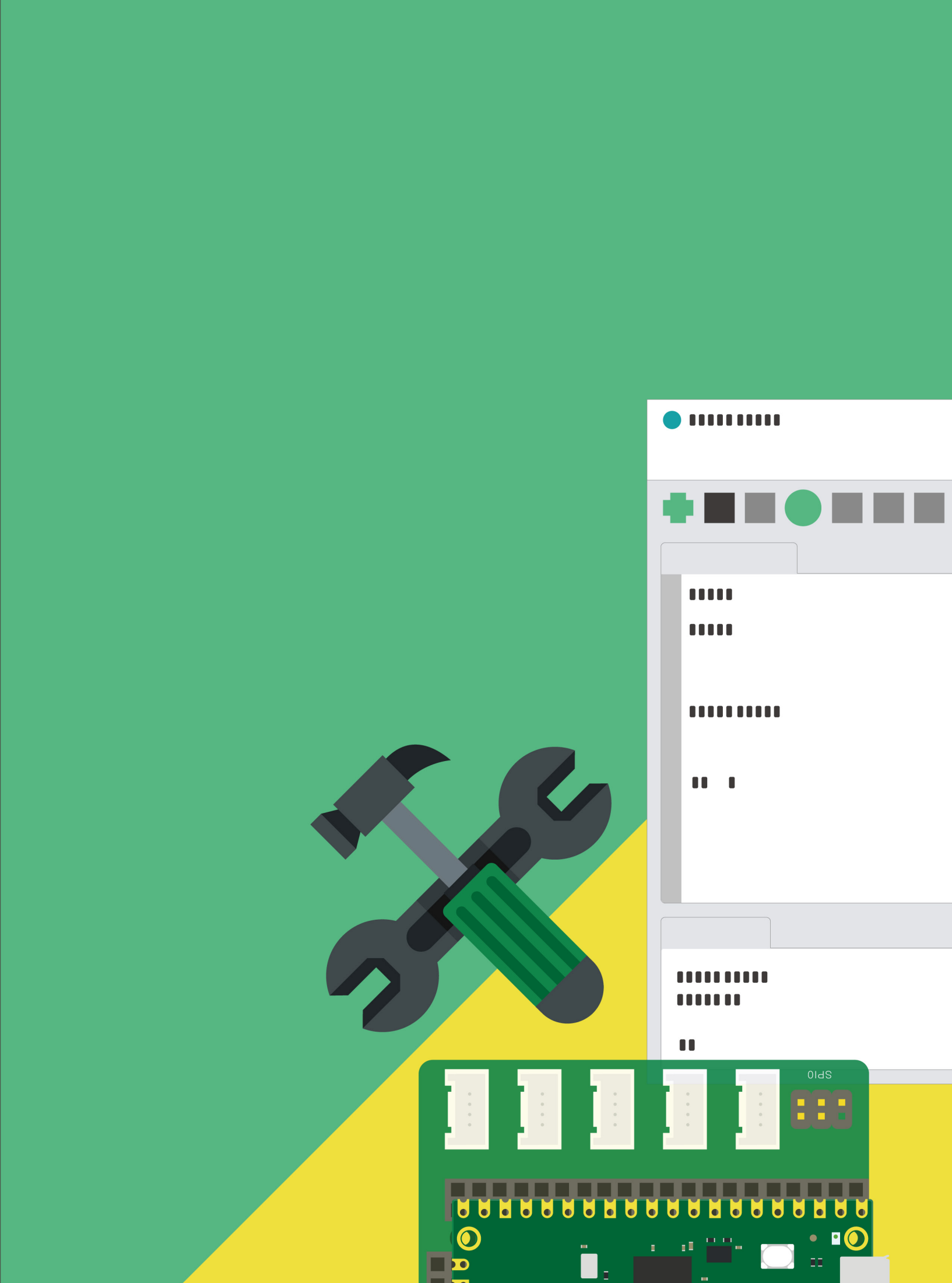


Thinking Expanding

In this lesson, we expanded the automatic door project by using what we have learned before to complete the function of the welcome device. Next, try to realize the project you conceptualized on your own!

Project Name	Sensor	Actuator/Display	Project Realization

Unit 4
Expansion Projects



Lesson 15 Create Your Own Projects

In previous lessons, we have learned a great deal of knowledge surrounding the Pico and supporting electronic hardware, in addition to programming with Thonny. At this point, we have mastered the basic programming language of MicroPython and applied it through various projects. Congratulations! You are now already have the hard skills to be a maker. However, it is now time to develop your soft skills as a maker — to know how to design a prototype product and create your own project!

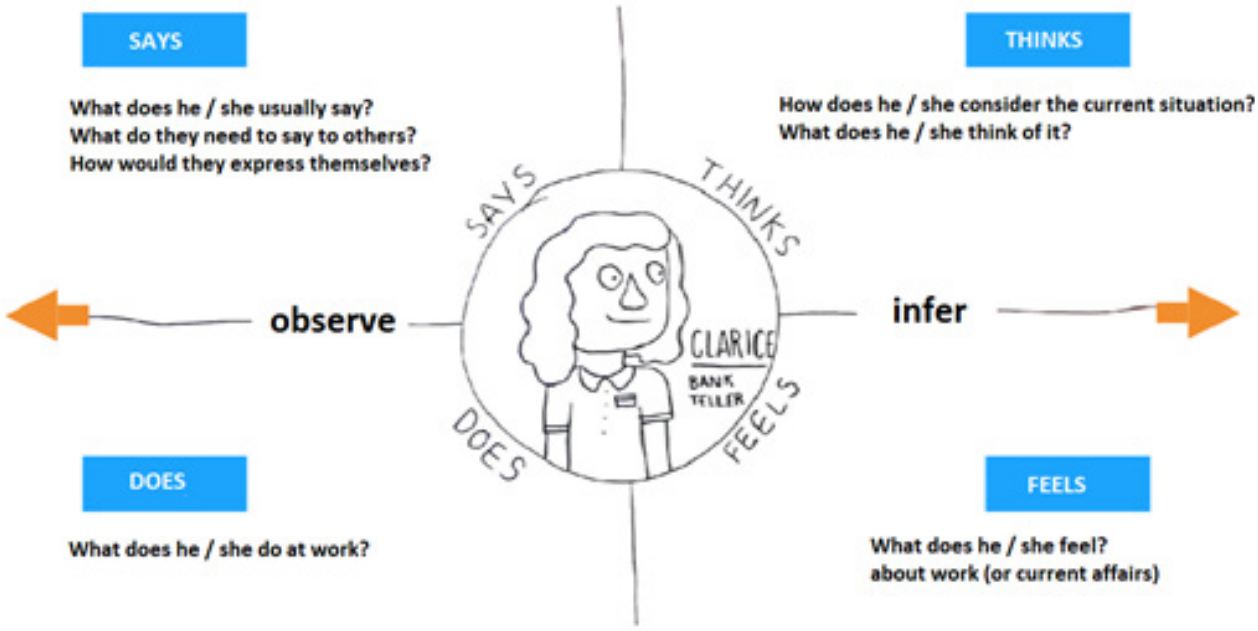
Staying Curious — Where Does Inspiration Come From?

"I recall that when I was young, I was able to open my eyes to the sun and notice the most subtle things. When I saw small things, I would observe their texture carefully, so I often felt the joy beyond the things themselves."When we were young, we were curious about the world — about the scent of flowers, the texture of soil, about teaching and interacting with others. However, as we grow older, our curiosity gradually shrank. Although there were more and more ways to get knowledge, we gradually lost the desire to actively explore, instead passively accepting new experiences. Yet there remains a group of people who still maintain a strong curiosity, who are good at discovering problems in life, and like to explore. They can be engineers, programmers, designers, teachers, singers, and the most creative people in their industry. They are makers. Makers are people who have a strong curiosity and imagination, love technology, and are keen on practicing it. They can live and work through technology, turn imagination into reality, and are willing to share and communicate.

Returning to the question of how to design a prototype product, start by carefully recalling the little things in your life. For example, you may dirty your clothes if you walk outdoors on a rainy day, but is there any way to solve it? Just like Newton was hit on the head by an apple, inspiration often comes from our life when we least expect it. Only by keeping our curiosity about the things around us can we seize these fleeting inspirations, explore deeply, and put in efforts towards a good creation.

From Divergence to Focus — Precise Positioning

There can be a lot of inspiration that you can play with at your own discretion. But for prototype products, your idea may require some precise positioning. One good method to approach this is, however, is the Empathy Map.The Empathy Map is a good way to find requirements. In short, the Empathy Map directly depicts what users think, feel, see, say, do, and hear in a visual drawing. It helps us to think for users in different scenarios, open our minds, explore the user's deep motivation, and help us to discover the real needs.



It is recommended for a large number of people to participate in the Empathy Map together. It is mainly divided into the following parts:

SAYS

What does he/she usually say?

What do they need to say to others?

How would they express themselves?

THINKS

How does he/she consider the current situation?

What does he/she think of it?

DOES

What does he/she do at work?

FEELS

What does he/she feel?

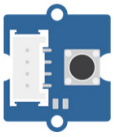
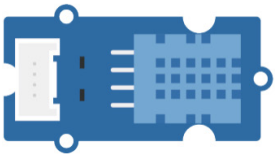
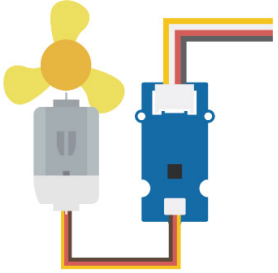
about work (or current affairs)

You can try this game with your friends. Determine an object, which can be "pet", "old man", "programmer", "policeman", "cleaner", etc. Then, focus on the problem through the Empathy Map — explore real needs so as to determine the positioning of the product.

Designers Are Not People Who Can Draw — Design Thinking in Prototype Production

When we talk about design, we often have a narrow impression that design is drawing. That is true to a certain degree, but in a broad sense, all targeted and planned creative activities are "design", which can include art design, industrial design, website design, product design, and so on. It contains a mode of thinking, a mode of problem-solving based on people's real needs.

Unlike the general straight-line path of finding and solving problems, it explores every link. Taking product design as an example, from the beginning of finding problems, we should stand in the perspective of the user, understand the user through empathy, listen to the real voice of the user, and form the user portrait. Once we make clear the user's requirements and determine the product's positioning, then can we begin on prototype design. In this process, we then need to carry out the selection of hardware, designing of software functions, materials and packaging, so as to form our preliminary product prototype.Taking the Intelligent Fan in the project practice unit as an example, we temporarily positioned the product as a small intelligent fan in a desktop setting. During the hardware selection and construction, we first selected the button to control the rotation of the Mini Fan, but decided that such manual control was too primitive. Therefore, we used the Temperature & Humidity Sensor instead of the button. When the temperature exceeds the comfortable temperature range, the fan can be turned on automatically. Similarly, you can add the LCD to display the current temperature, or also add a PIR Motion Sensor to make the fan run only when someone is nearby to save energy.

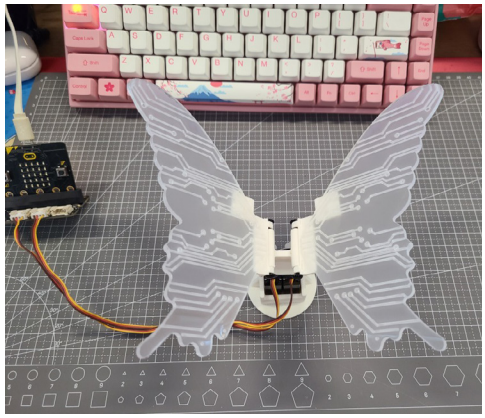
Project Name: Intelligent Fan		
Function Requirement	Hardware Selection	Function Description
Manual Switch	 Button	Grove - Button is a momentary switch, which is easy to use. When pressed, it outputs a high-level signal; when released, it outputs a low-level signal. However, it needs a manual operation, which cannot show the core of "intelligence".
Temperature Control	 Temperature & Humidity Sensor	Detect the temperature and humidity in the environment. We only need to take the temperature value and set the comfort value as the basis of the fan on and off through the program. It can achieve the effect of temperature control and energy saving.
Motion Detection	 PIR Motion Sensor	The PIR Motion Sensor can detect whether there are people around, and the fan can only be turned on when there is someone nearby. It is more intelligent than temperature control.
Fan Simulation	 Mini Fan	Mini fan is composed of motor drive, DC motor and fan blade, which can be used as a prototype product of the intelligent fan without making an additional fan.

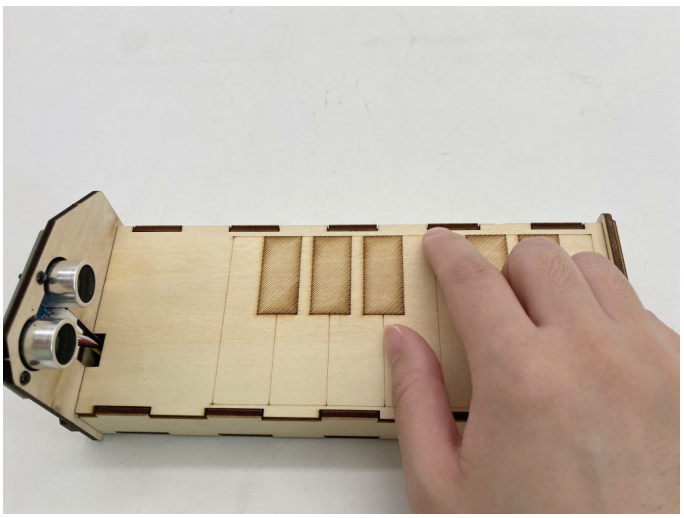
Display Temperature	 16x2 LCD	LCD can display the temperature value and time to realize visualization. However, the main purpose of the fan is to reduce the temperature. It is not practical to display the temperature value and time. And considering the factor of "desktop" and "small", the product should not be too big.

As can be seen from the above table, even for the hardware selection of prototype products, many factors can be considered and will contribute to your final design. Then, we have to devise the entire program to achieve our intended purpose in software.

```
from dht11 import *
from machine import Pin,ADC
from utime import sleep

miniPir = Pin(18, Pin.IN)
dht2 = DHT(16)
miniFan = machine.Pin(20,machine.Pin.OUT)

while True:
    temp = dht2.readTemperature()#temp
    if miniPir.value()== 1 and temp > 26:
        miniFan.value(1)
    else:
        miniFan.value(0)
```

So far, the functional part of the prototype product is completed. Next is the design of the external appearance, including the materials and and construction of structures. For example, we often use 3D printing technology or laser cutting technology to design and manufacture a housing for our products. Of course, we can also use common materials in life, such as hardboard or cardboard. Thus, even if we don't know how to use digital tools, we can still quickly build prototypes. The following picture shows various prototypes that have been made of different materials.





Go Through Fire and Water — Testing and Optimization of Prototype Works

After completing our prototype, testing and optimization are also very important. We need to verify whether its function is realized and whether it meets the original design requirements. This process should involve the target users as much as possible and lead into further improvements through their feedback. In the process of testing, it is normal to even rework the idea from scratch. However, it is only in this way that we can identify the gaps in our initial solution and work towards a better prototype that effectively serves the needs of our users.

Create Your Own Project

Next, try to create your own project based on what you learned in this and previous lessons, and develop a corresponding project scheme according to the guidelines below. After the scheme is formed, you can begin work on your project, in preparation of the presentation that will take place in the next lesson.

Project Name	
Function Realization	(Define the function that the project wants to realize, describe the effects in as much detail as possible)
Hardware Required	(List of hardware required to realize the function)

Materials Required	(In addition to hardware programming, we can use various resources to build the appearance of the project structure. Indicate the required materials.)
Prototype Sketch	(Draw the appearance of the project in the form of legend. It is helpful for us to create the prototype.)

Lesson 16 Project Presentation

In this lesson, we will share and show your projects in the form of a project presentation. You can showcase your project and the story behind it through posters, PPTs or videos. Through this process, we can consolidate the key knowledge of of previous lessons by immersing in discussion and communication, and learn to share, and think and explore problems from different angles.

Practice & Operation

If this material is used in schools, institutions, societies or other educational settings, teachers or counselors are encouraged to organize a project presentation. Students can work in groups on different projects, and receive feedback from mentors after the presentation. Teachers may also wish to invite students' parents and other school teachers to form an expert group to provide invaluable feedback to students.

Rules of Presentation

Time limit: 3 minutes (timekeeper shall remind the time)
Form: PPT or animation + explanation
Content: explain according to the content on PPT, including:

1. Product name and picture
2. Product design concept
3. Product features
4. Product structure or details
5. Product functions
6. Question and answer

Preparation before Presentation

1. Organizer of the presentation & on-site host: 1
   Timekeeper: 1
   Teller: 3
   Photographers: 2
2. Material personnel
   Material requirement: computer, page flipper
   Classroom arrangement: the arrangement of tables, chairs and benches
   Data collection: PPT collection and display
   Vote making: expert vote making
   Prize preparation: prepare prize for 6 awards

Presentation

1.Product Showcase

Each group explains and shows the products in turn under the arrangement of the host.

2.Expert Comments

The invited experts and teachers comment on the performance of the group.

3. Selection

After all products are showed, the selection shall be carried out.

4. Award Presentation

According to the results of the vote, the awards of Most Creative, Most Practical, Most Investable, Most Technological, Best Team, and Future Star are selected and presented.

5. Speech by Student Representatives

Invite the representatives of 1—2 winning groups to share.

Selection Rules

1. If there are 6 awards in total, then each group has 6 ballots to vote. However, the ballots should represent the opinions of the group instead of individuals, so please take the opinions of all the group members into consideration before making a collective vote.

2. Expert Ballots

Each expert group has 6 ballots, and each ballot is equivalent to 5 ballots of the ordinary group. These ballots are also accumulated into the total number of ballots.

3. Cumulative ballots

The ballots shall be counted by the teller and announced in time.

Award References

Most Creative Award: the product concept may not have been completed at this point, but the idea is bold and creative, and is likely to be realized in the future;

Most Practical Award: high degree of completion, comprehensive use of the knowledge learned, and practical product design;

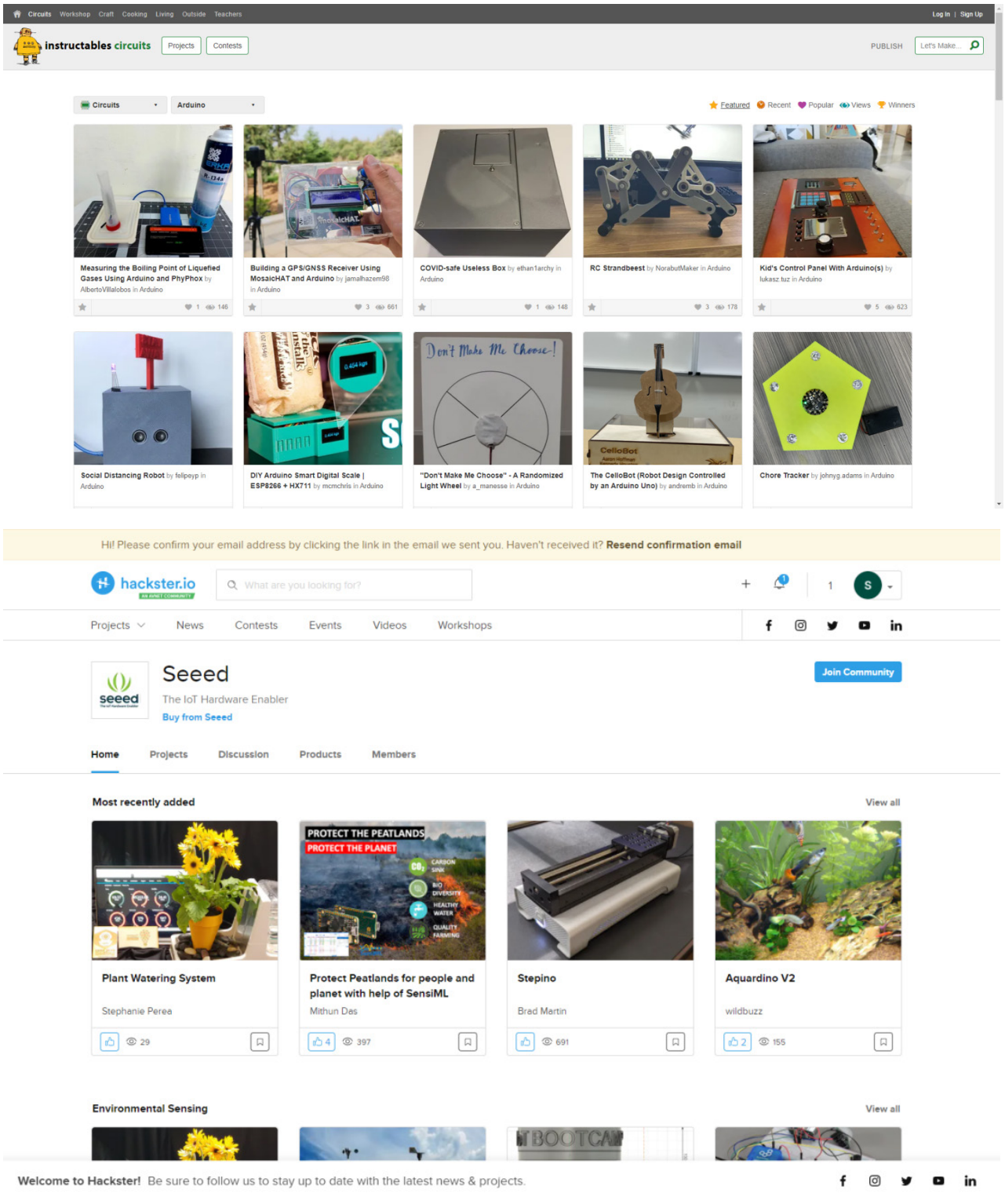
Most Investable Award: creative marketing, with clear business model market strategy;

Most Technological Award: product function design and on-site demonstration are grounded on cutting-edge science and technologies;

Best Team Award: good teamwork, clear task division, mutual help and cooperation, smooth product showing, and good product completion;

Future Star Award: team and project with strong potential, with ideas and executive abilities to show and realize their ideas well;

If this course is used by individuals or enthusiasts, you can also publish your designed projects on social media or maker community platforms, such as Instructable or hackster.io.



Of course, we would also love to have you share your project with us.

Email: contact@chaihuo.org

Course Developers

This course was co-authored by the following SeeedStudio employees:

Authors: Yimeng Shi, Haixu Liu

Designer:Yihui Meng

Proofreaders: Lei Feng,Jonathan Tan

CONTACTS

Tel: +86-0755-86716703

Address: 1002, G3 Building, TCL International E City, 1001 Zhongshan Park Road, Nanshan District, Shenzhen

General： contact@chaihuo.org

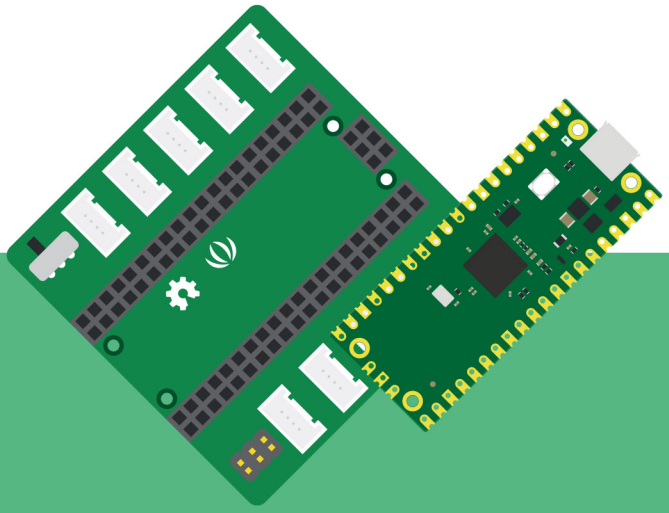
Tech Support: techsupport@chaihuo.org

www.tinkergen.com

www.seeedstudio.com

TinkerGen

STEM made simple



TinkerGen STEM made simple | seeed studio

To know more about TinkerGen

Please contact us:

contact@chaihuo.org

86-0755-86716703

www.tinkergen.com