

Welcome

Thank you for choosing Freenove products!

Get Support & Offer Input

You may find somethings missing or broken, or some difficulty to learn the kit.

Freenove provides free and quick support, including but not limited to:

- Quality problems of products
- Problems in using products
- Questions for learning and technology
- Opinions and suggestions
- Ideas and thoughts

If you have any concerns, please send email to us:

support@freenove.com

And suggestions and feedbacks are welcomed. Many customers offered great feedbacks. According to that, we are keeping updating the kit and the tutorial to make it better. Thank you.

Safety

Pay attention to safety when using and storing this product:

- Do not expose children under 6 years of age to this product. Put it out of their reach.
- Children lack safety ability should use this product under the guardianship of adults.
- This product contains small and sharp parts. Do not swallow, prick and scratch to avoid injury.
- This product contains conductive parts. Do not hold them to touch power supply and other circuits.
- Some parts will rotate or move when it works. Do not touch them to avoid being bruised or scratched.
- The wrong operation may cause overheat. Do not touch and disconnect the power supply immediately.
- Operate in accordance with the requirements of the tutorial. Otherwise, the parts may be damaged.
- Store the product in a dry place and avoid direct sunlight.
- Turn off the power of the circuit before leaving.

About

Freenove provides open source electronic products and services.

Freenove is committed to helping customers learn programming and electronic knowledge, quickly realize their creative ideas and product prototypes and launching innovative products. Our services include:

- Kits of robots, smart cars and drones
- Kits for learning Arduino, Raspberry Pi and micro:bit
- Electronic components and modules, tools
- **Product customization service**

You can learn more about us or get our latest information through our website:

<http://www.freenove.com>

Copyright

All the files we provided are released under [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](#). You can find a copy of the license in the folder.



This means you can use them on your own derived works, in part or completely. But NOT for the purpose of commercial use.

Freenove brand and logo are copyright of Freenove Creative Technology Co., Ltd. Cannot be used without formal permission.



Contents

Contents.....	1
Preface.....	1
Micro:bit.....	2
Meet micro:bit.....	2
Features.....	3
Hardware	4
Micro:bit GPIO Extension Board.....	5
Hardware and Feature.....	5
How to use?.....	6
Code & Programming.....	8
Quick Start.....	8
Makecode.....	12
Quick download	13
Import Code.....	15
Python.....	17
Chapter 1 Led matrix	23
Project 1.1 Heartbeat.....	23
Project 1.2 Show Number	29
Project 1.3 Show Text	33
Project 1.4 Show Custom	35
Chapter 2 Built In Button.....	38
Project 2.1 Button A and B.....	38
Chapter 3 LED	42
Project 3.1 Blink	42
Chapter 4 Button and LED	50
Project 4.1 Control LED by Button	51
Project 4.2 Table Lamp.....	57
Chapter 5 LED Bar Graph	62
Project 5.1 Flowing Light	62
Chapter 6 PWM.....	69



Project 6.1 Breathing Light	69
Chapter 7 RGBLED	76
Project 7.1 Monochromatic Light	76
Project 7.2 Colorful Light.....	83
Chapter 8 Neopixel	91
Project 8.1 Rainbow Water Light	91
Chapter 9 Buzzer	99
Component knowled	99
Project 9.1 Active Buzzer	102
Project 9.2 Happy Birthday Melody	106
Project 9.3 Custom Melody	110
Chapter 10 Serial Communication	114
Project 10.1 Display the Data.....	114
Chapter 11 Magnetometer	119
Project 11.1 Display Magnetometer Data.....	119
Project 11.2 Electronic Compass.....	126
Chapter 12 Accelerometer	133
Project 12.1 Display Accelerometer Data	133
Project 12.2 Gradienter	138
Chapter 13 Potentiometer	143
Project 13.1 Potentiometer	143
Chapter 14 Potentiometer And LED	151
Project 14.1 Soft Light.....	151
Project 14.2 Colorful Soft Light	155
Project 14.3 Rainbow Light.....	162
Chapter 15 Light Sensor	168
Project 15.1 Built In Light Sensor	168
Project 15.2 Night light	173
Chapter 16 Temperature Sensor	180
Project 16.1 Built In Temperature Sensor	180
Project 16.2 Thermistor	183
Chapter 17 Joystick	190
Project 17.1 Display Joystick Data	190

Project 17.2 Show Direction	196
Chapter 18 74HC595 and LED Bar Graph.....	203
Project 18.1 Flowing Light.....	203
Chapter 19 74HC595 and 7-segment display.....	211
Project 19.1 7-segment display	211
Chapter 20 LCD1602	220
Project 20.1 I2C LCD1602.....	220
Chapter 21 Motor	230
Project 21.1 Rlay And Motor.....	230
Project 21.2 Potentiometer and Motor.....	236
Chapter 22 Servo	245
Project 22.1 Sweep	245
Project 22.2 Knob.....	252
Chapter 23 Stepper Motor	256
Project 23.1 Stepper Motor	256
Chapter 24 Hygrothermograph	263
Project 24.1 Hygrothermograph.....	263
Chapter 25 Matrix Keypad	272
Project 25.1 Keypad	272
Project 25.2 Countdown Timer.....	282
Chapter 26 Infrared Motion Sensor.....	291
Project 26.1 Sense Light.....	291
Chapter 27 Ultrasonic Ranging	296
Project 27.1 Ultrasonic Ranging.....	296
What's next?	306

Preface

Do you want to learn programming?

Currently, Program is developed into the lower age group, and everyone programming is a trend. From Arduino and Raspberry Pi to micro:bit, simple graphical programming makes child programming a possibility. Maybe you haven't heard of them, it doesn't matter. With this product and the tutorial, you can easily complete a programming project and experience the fun as a Maker.

Micro:bit is a powerful and simple development board. Even if you've never programmed before, simple graphical programming interface can make you easy to learn. It doesn't need any professional programming software. You just need a browser to program it. So, whether your computer system is Windows, Linux or Mac. They can all do it. And you can also program it with Python.

It have a lot of fans in the world. They are keen to exploration, innovation and DIY and they contributed a great number of high-quality open source code, circuit and rich knowledge base. So we can realize our own creativity more efficiently by using these free resource. Of course, you can also contribute your own strength to the resource.

Microbit can do a lot of projects. We can add a lot of kits to develop on the breadboard. We can carry out interesting projects such as ultrasonic ranging, gravity control, playing music, etc.

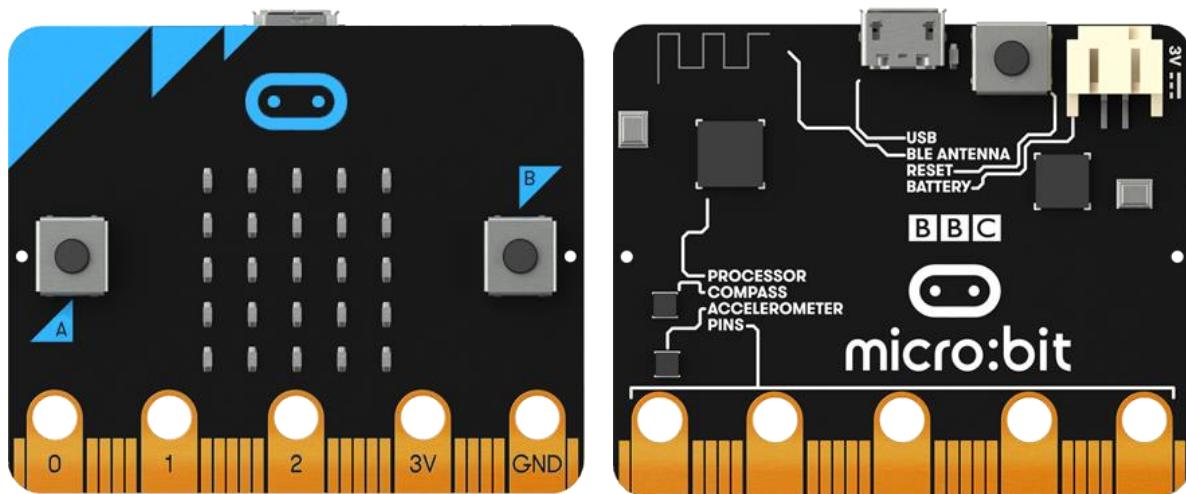
In each learning chapter of this tutorial, we provide program source code with detailed program explanations and burnable binaries. So you can really understand the meaning of each section of program.

Additionally, if you have any difficulties or questions about this tutorial and the kit, you can always ask us for quick and free technical support.

Micro:bit

Hello, welcome to the world of electronic and programming. Let's start Micro:bit and Micro:Rover tour from here.

Meet micro:bit

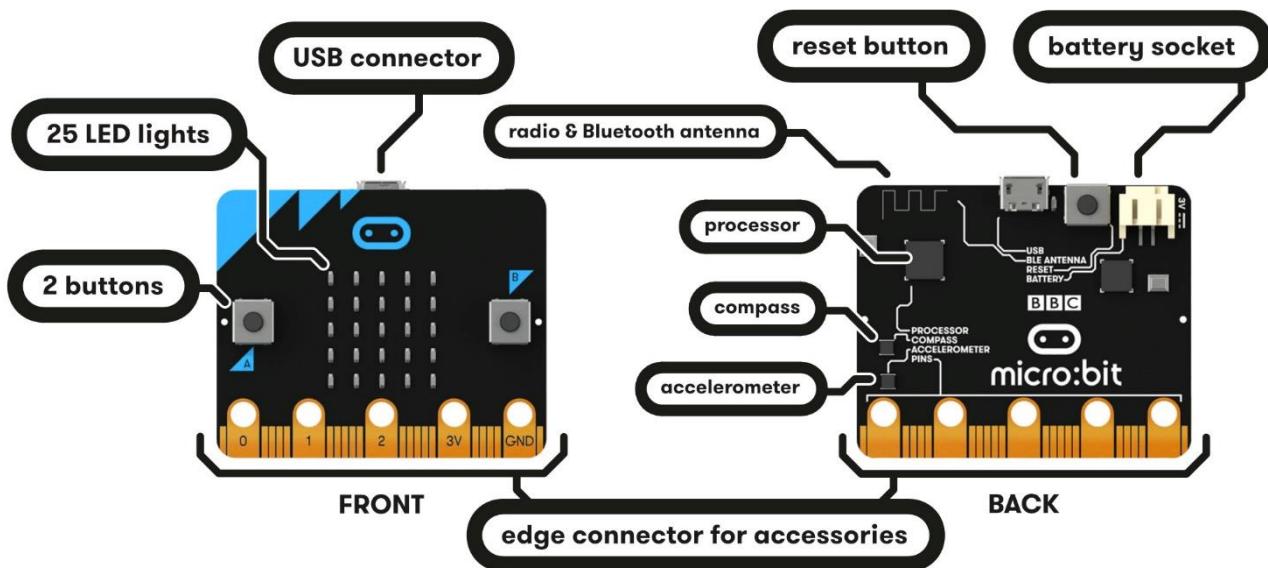


The BBC micro:bit is a handheld, programmable micro-computer that can be used for all sorts of cool creations, from robots to musical instruments – the possibilities are endless.

For more contents, please refer to:

<https://microbit.org/guide/>

Features



Your micro:bit has the following physical features:

- 25 individually-programmable LEDs
- 2 programmable buttons
- Physical connection pins
- Light and temperature sensors
- Motion sensors (accelerometer and compass)
- Wireless Communication, via Radio and Bluetooth
- USB interface

For more contents, please refer to:

<https://microbit.org/guide/features/>

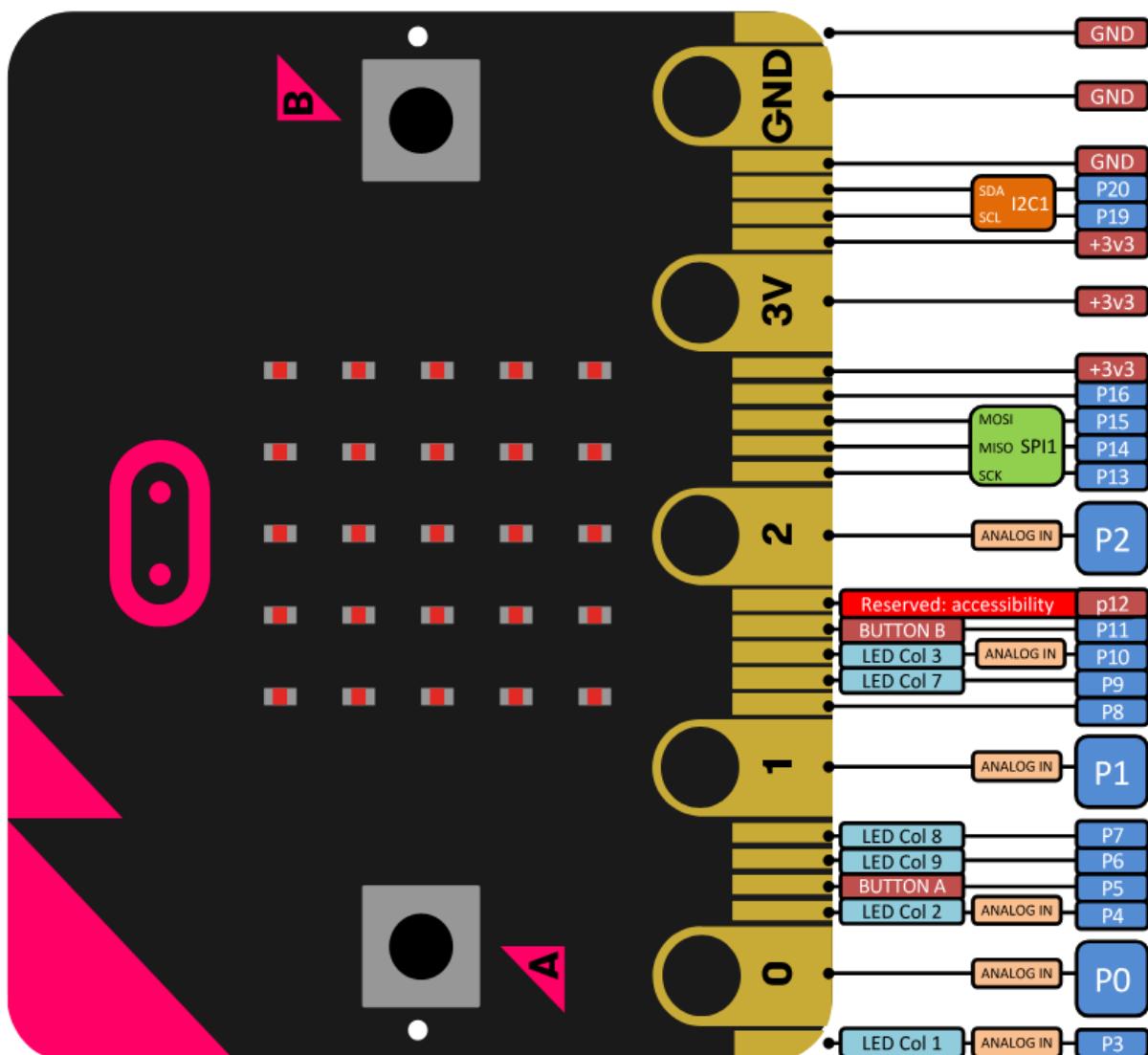
Hardware

For beginners, this part of information is not necessary. Just have a brief understanding. But if you want to be a developer, hardware information will be very helpful. Detailed hardware information about micro:bit can be found here: <https://tech.microbit.org/hardware/>.

First, get to know the micro:bit GPIO.

GPIO

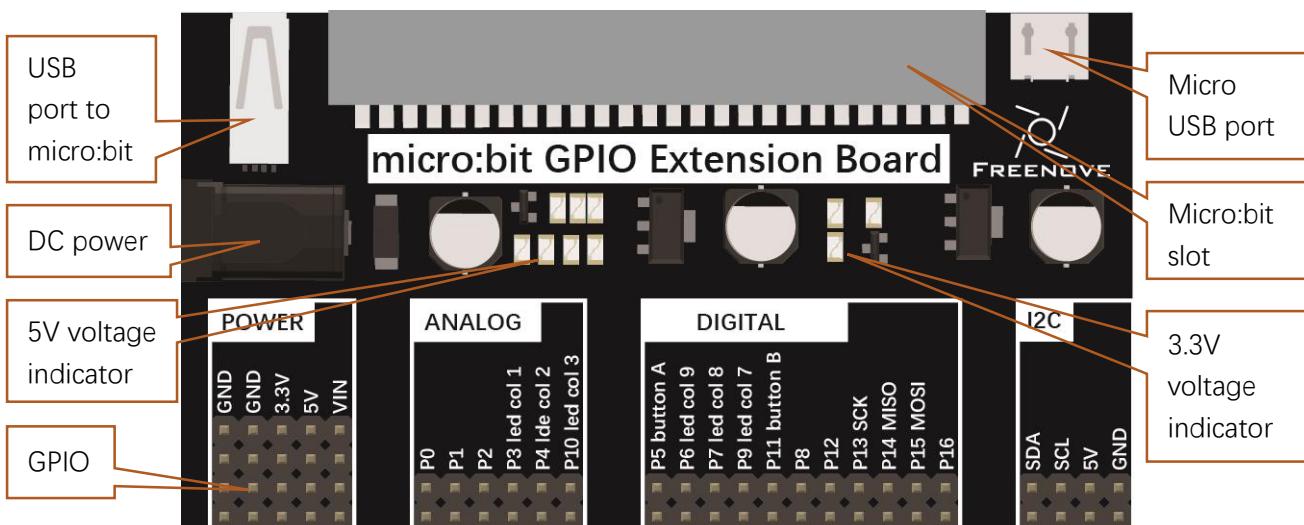
GPIO, namely General Purpose Input/output Pins, is a part of micro:bit. It is an important part for connecting external devices. All sensors and devices on Rover communicate with each other through micro:bit GPIO. The following is the GPIO serial number and function diagram of micro:bit:



Micro:bit GPIO Extension Board

Hardware and Feature

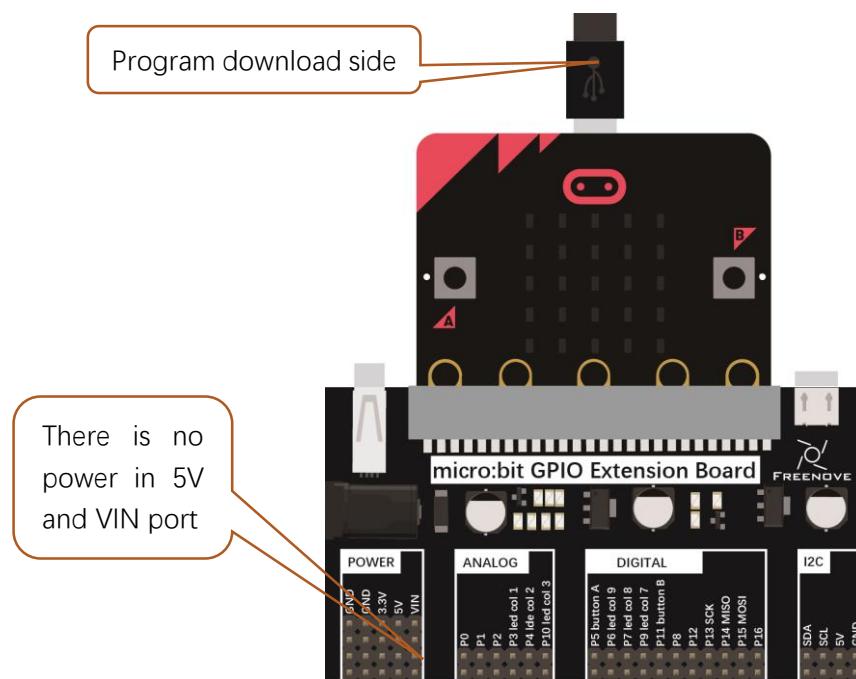
Micro:bit GPIO Extension Board is shown as below:



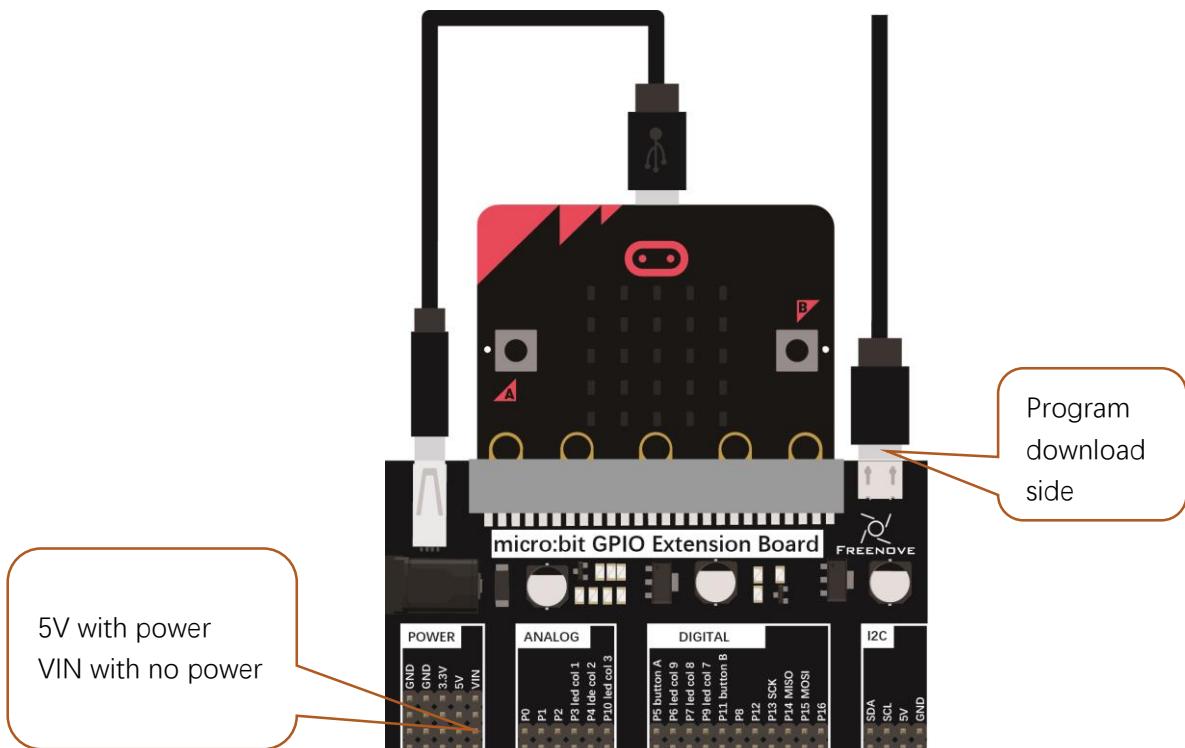
Micro:bit GPIO Extension Board is connected to micro:bit board via slot. GPIO(General-purpose input/output) on extension board is connected to GPIO of micro:bit. It also add 5V and VIN(9V) IO port to meet requirement of more devices.

How to use?

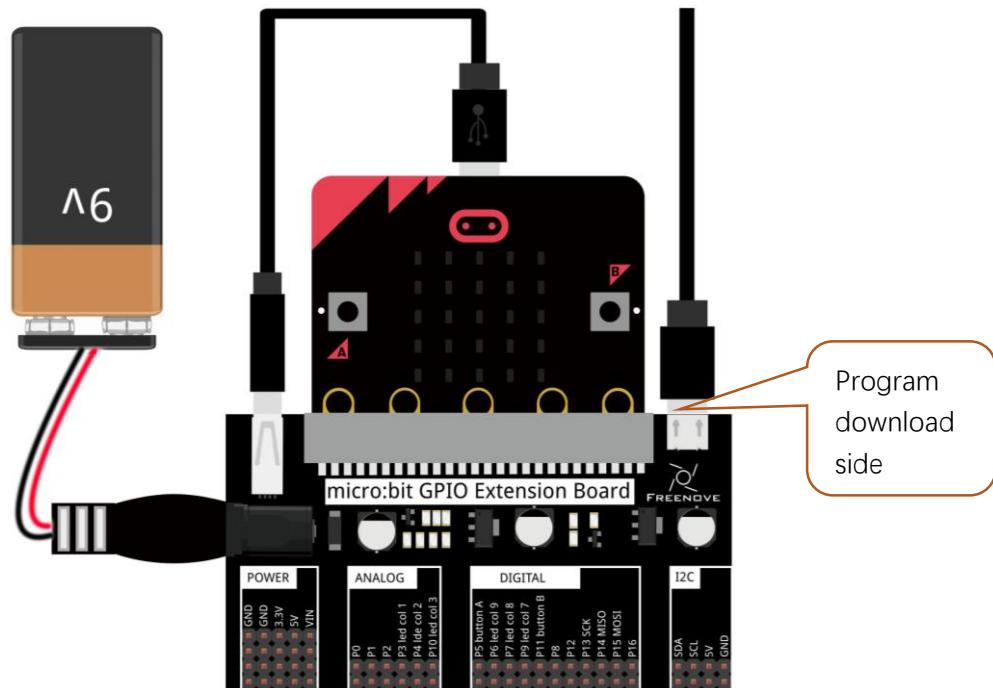
If external device doesn't need 5v voltage and 9v, you can use following wiring.



If external device uses 5v voltage, but power needed is not large, you can use following wiring.



If external device use 5v voltage, but power needed is large, you can use following wiring.



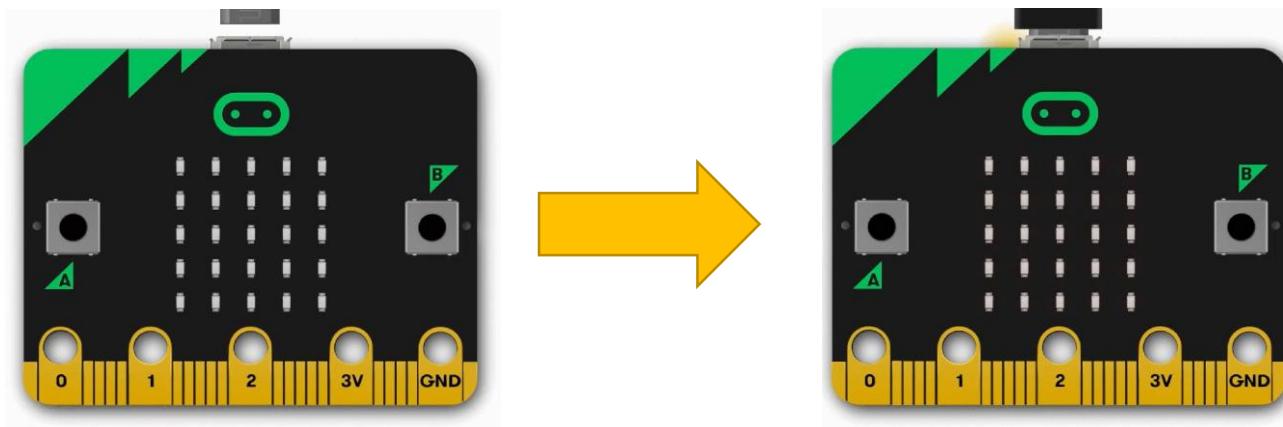
Code & Programming

Quick Start

This section describes how to write program for micro:bit and how to download them to micro:bit. There are very detailed tutorials on the official website. You can refer to: [Https://microbit.org/guide/quick/](https://microbit.org/guide/quick/).

Step 1: Connect It

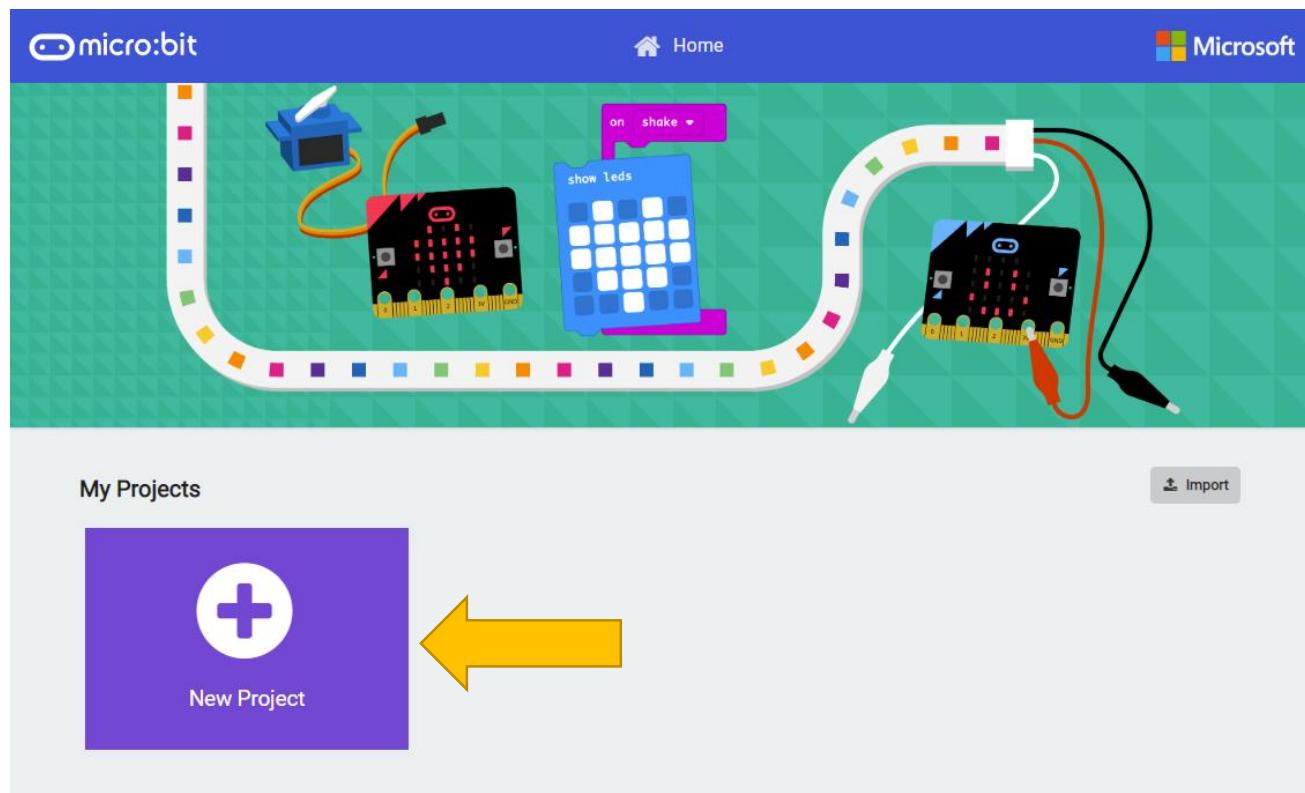
Connect the micro:bit to your computer via a micro USB cable. Macs, PCs, Chromebooks and Linux systems (including Raspberry Pi) are all supported.



Step 2: Program It

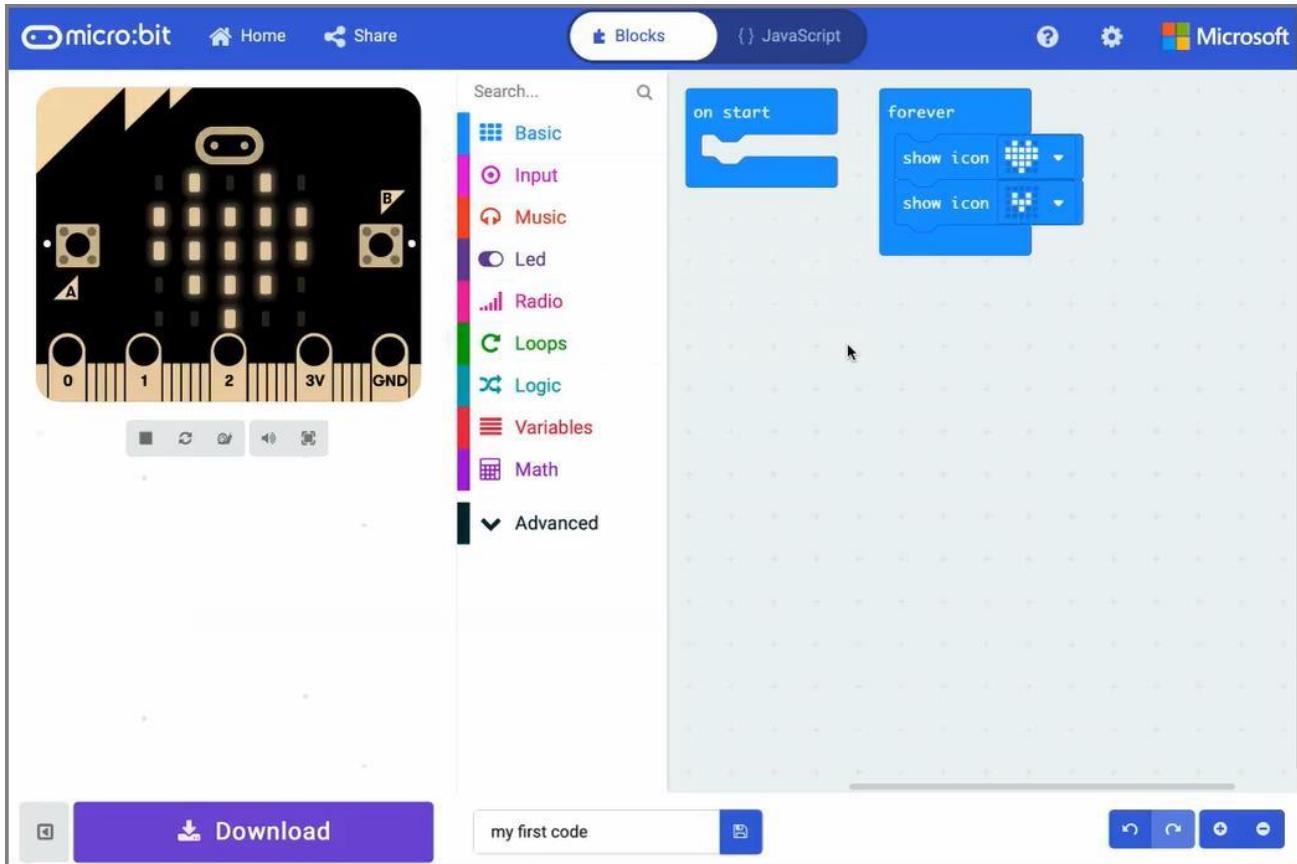
Visit <https://makecode.microbit.org/>. Then click "New Project" and start programming.

If your computer has Windows 10 operating system, you can also use Windows 10 App for programming, which is exactly the same as programming on browsers. [Get windows 10 App\(Click\)](#).



Write your first micro:bit code. For example, drag and drop some blocks and try your program on the Simulator in the MakeCode Editor, like in the image below that shows how to program a Flashing Heart.

Here is a demo video: <https://microbit.org/images/quickstart/makecode-heart.mp4> Makecode will be further introduced in next section.

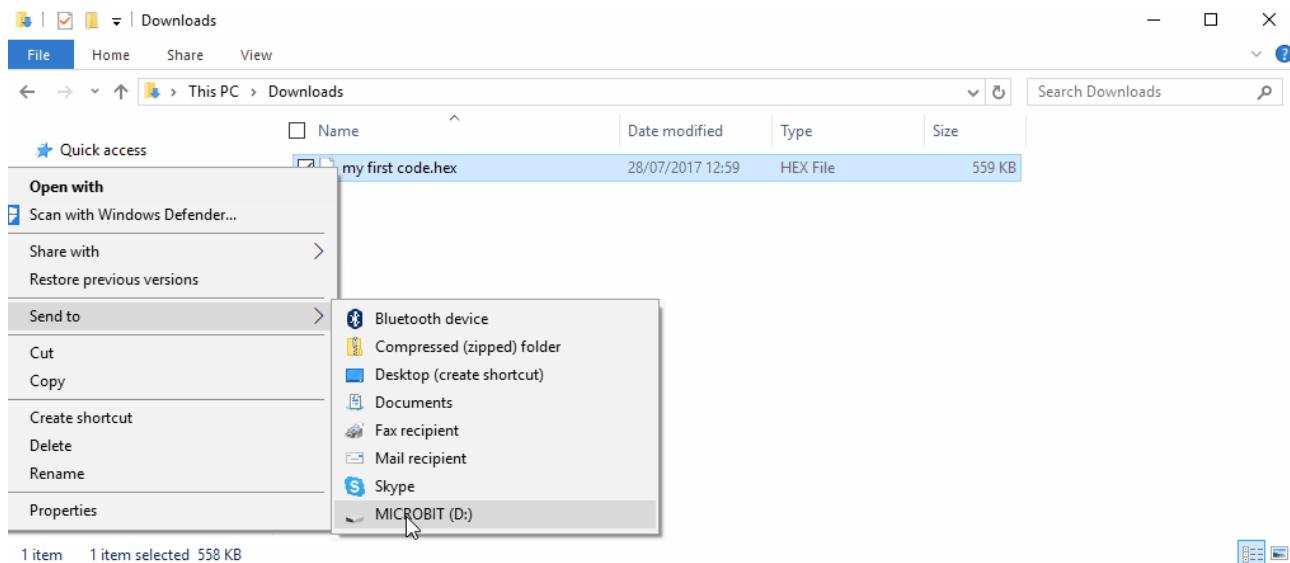


Step 3: Download It

If you write program by using Windows 10 App, you just need click the "Download" button, then the program will be downloaded directly to micro:bit without any other actions.

If you write program by using browser, please follow steps below:

Click the Download button in the editor. This will download a 'hex' file, which is a compact format of your program that your micro:bit can read. Once the hex file has downloaded, copy it to your micro:bit just like copying a file to a USB drive. On Windows you can right click and choose "Send to→MICROBIT."



Step 4: Play It

The micro:bit will pause and the yellow LED on the back of the micro:bit will blink while your code is programmed. Once that's finished the code will run automatically!

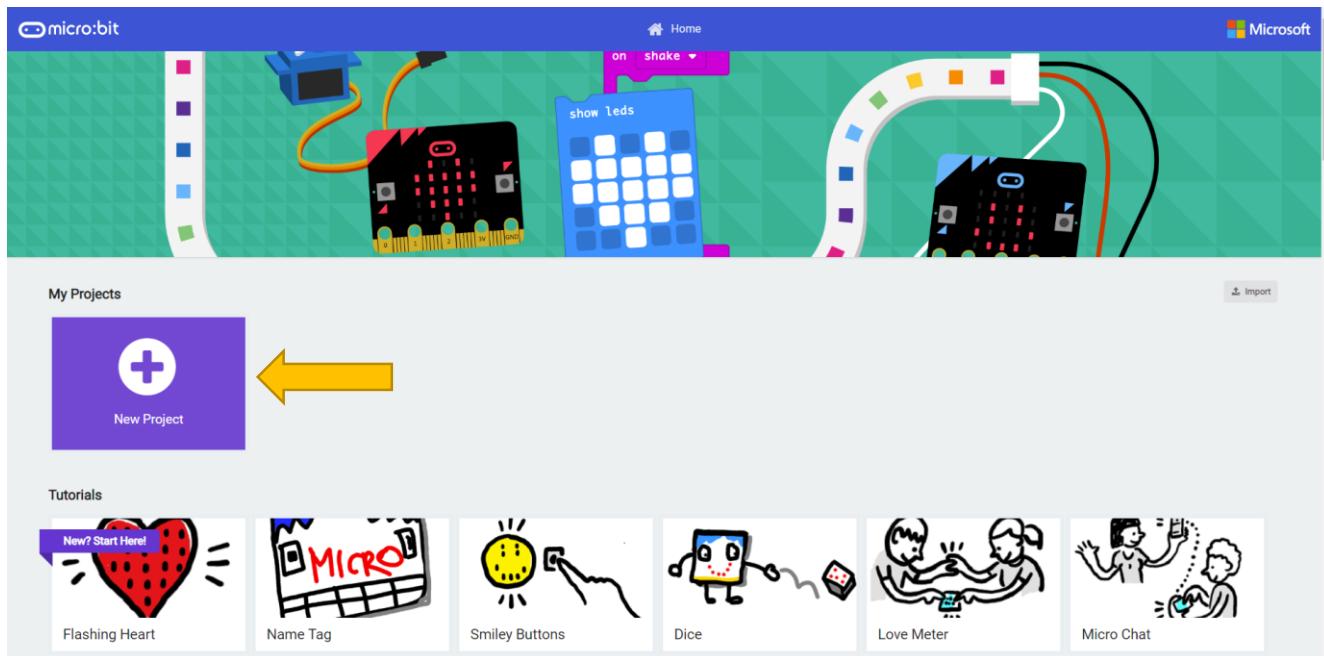
Waring

The MICROBIT drive will automatically eject and come back each time you program it, but your hex file will be gone. The micro:bit can only receive hex files and won't store anything else!

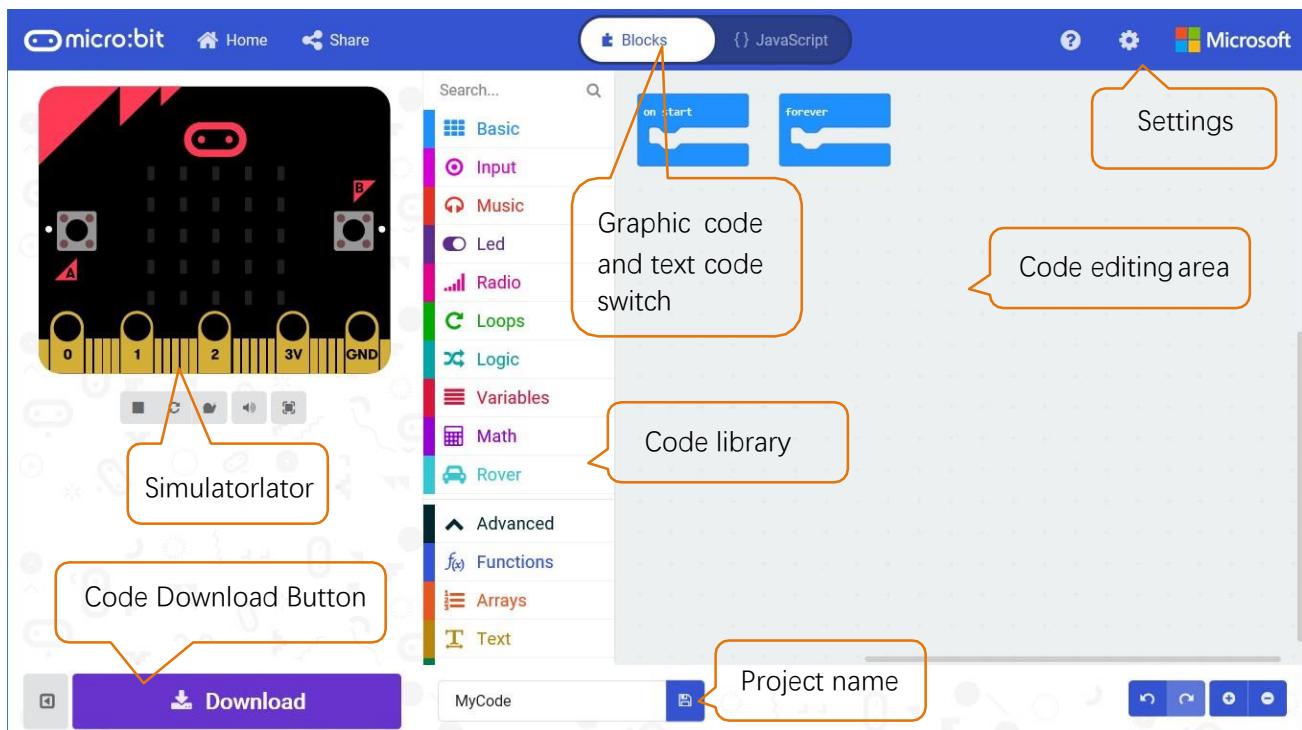
Makecode

Open web version of [makecode](#) or **windows 10 app version of makecode**, which can be downloaded on [Microsoft store](#).

<https://makecode.microbit.org/>



Click “New Project”, Makecode editor is as below:



In the code area, there are two fixed blocks “on start” and “forever”.

The code in the “on start” block will be executed only once after power-on or reset. And the code in “forever” block will be executed cyclically.

Quick download

As mentioned earlier, if you use **Windows 10 App of makecode (recommended)**, you can **quickly** download the code to micro:bit by **clicking the download button**. Using **browser version of makecode** may require **more steps**.

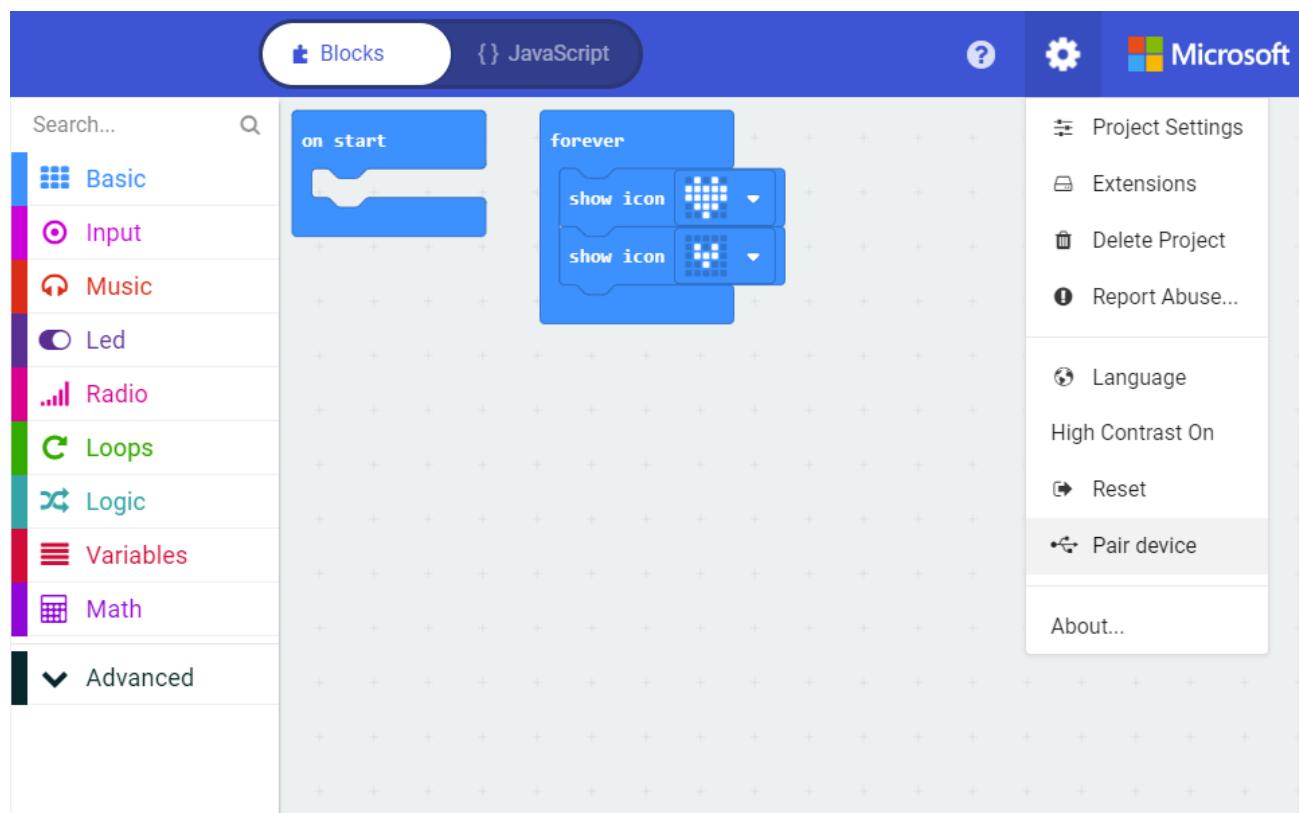
If you use browser version of makecode on **Google Chrome 65+ for platform Android, Chrome OS, Linux, macOS and Windows 10**, you can also realize the function of quick download.

Here we use webUSB feature of Chrome, which allows web pages to access your USB hardware devices. We will complete the connection and pairing of the micro:bit device with the webpage in following steps.

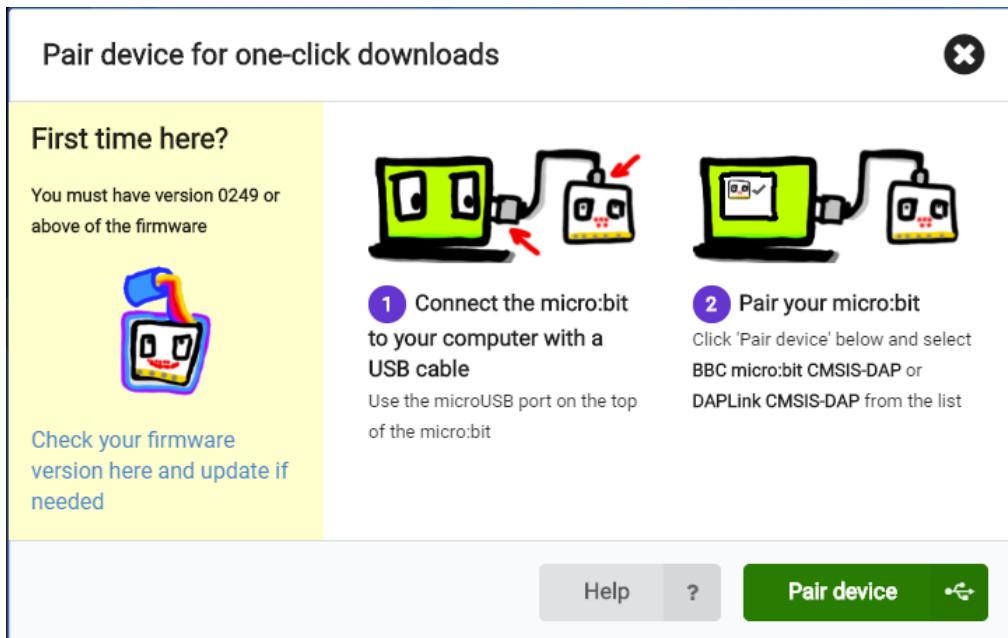
Pair device

Connect the computer and Micro:bit with a USB cable.

Click the gear menu in the top right corner and then click on "Pair device".



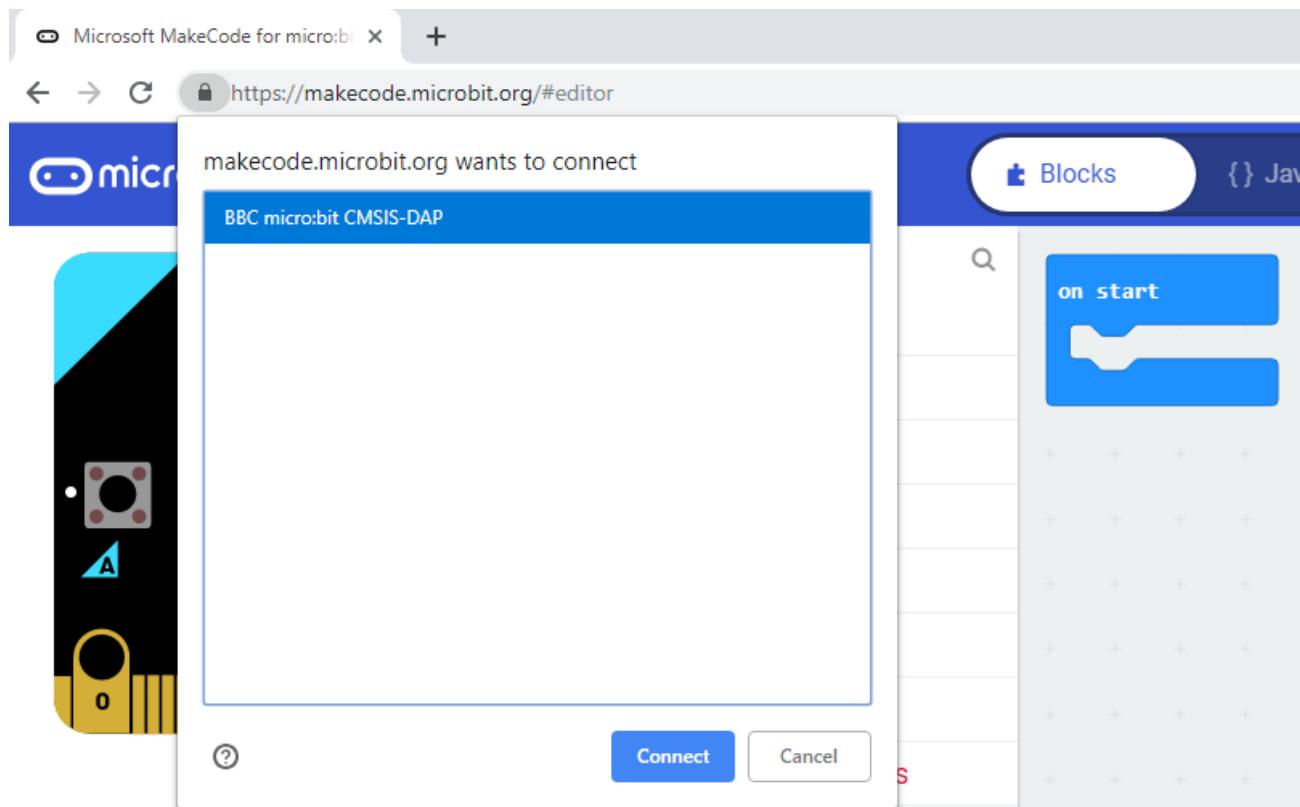
Then continue to click "Pair device" button.



Select device in popup window and click "Connect" button. If there are no devices in the pop window, please refer to following content: <https://makecode.microbit.org/device/usb/webusb/troubleshoot>

And we save the page as a file “**Troubleshooting downloads with WebUSB - Microsoft MakeCode.pdf**”. You can read it directly in the folder of this tutorial.

And the file “**Firmware microbit.pdf**” introduces how to update firmware of micro:bit. Its content come from: <https://microbit.org/guide/firmware/>



After the connection succeeds, click the Download button and the program will be downloaded directly to Micro: bit.

Import Code

We provide hex file (project files) for each project. Hex file contains all the contents of the project and can be imported directly. You can also complete the code of project manually. If you choose to complete the code by dragging code block, you may need add necessary extensions.

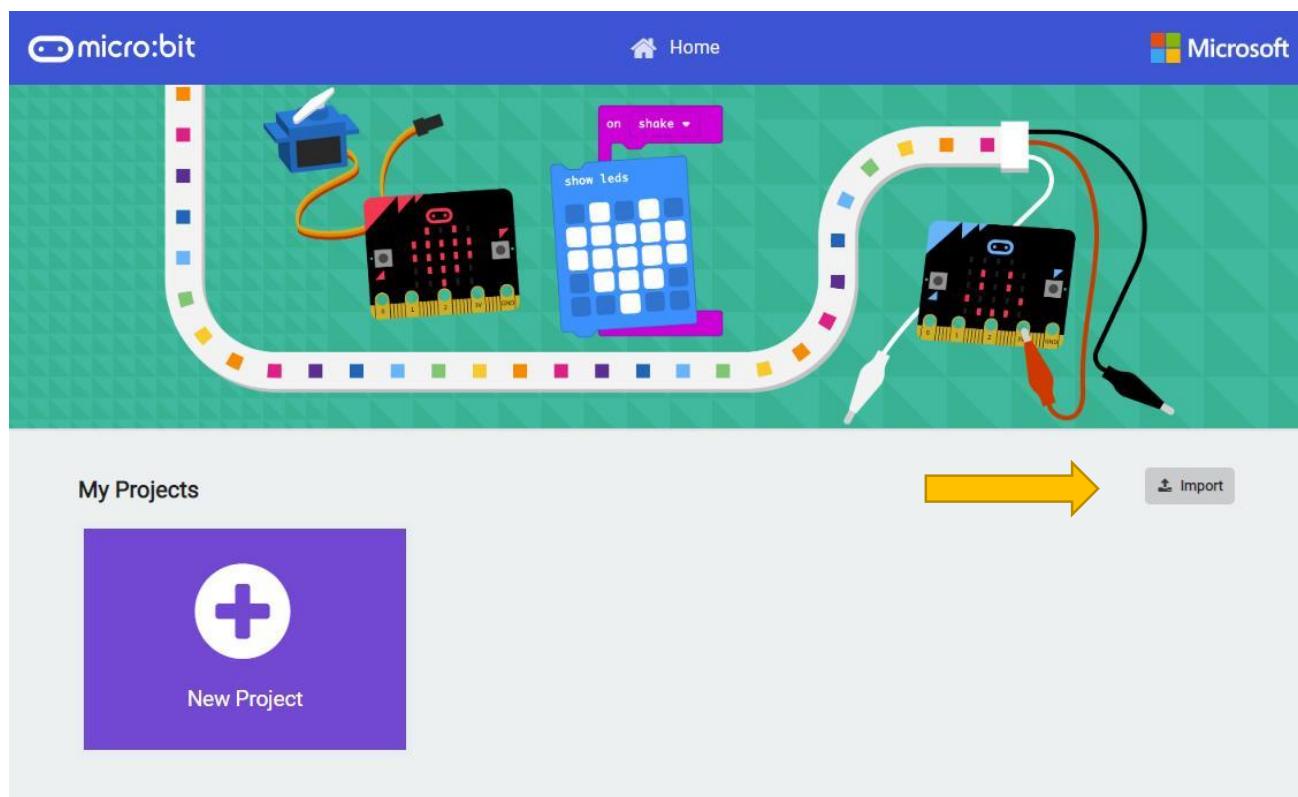
As for simple projects, it is recommended to complete the project by dragging code block.

As for complicated projects, it is recommended to complete the project by importing Hex code file.

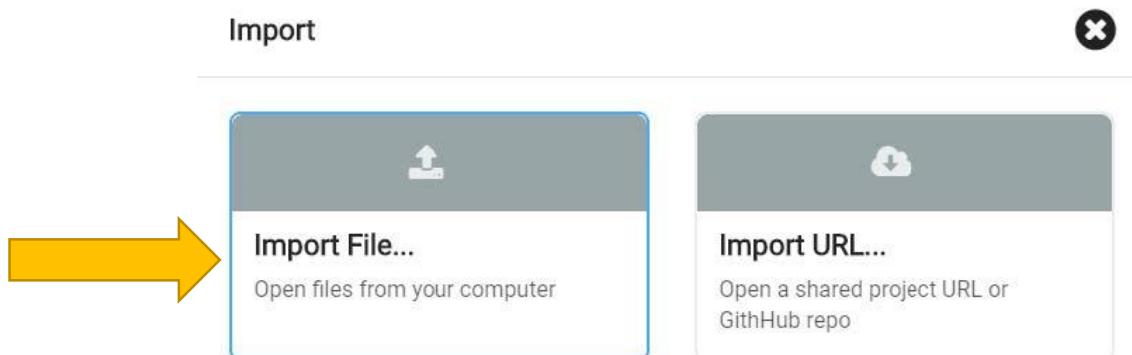
Next, we will take “Heartbeat” project as an example to introduce how to load code.

Open web version of [makecode](#) or windows 10 app version of makecode.

Click “Import” button on the right side of HOME page.



In the pop-up dialog box, click "Import File".



Select file “.. Projects/BlockCode/01.1_Heartbeat/Heartbeat.hex”. Then click “Go ahead!”

Open .hex file

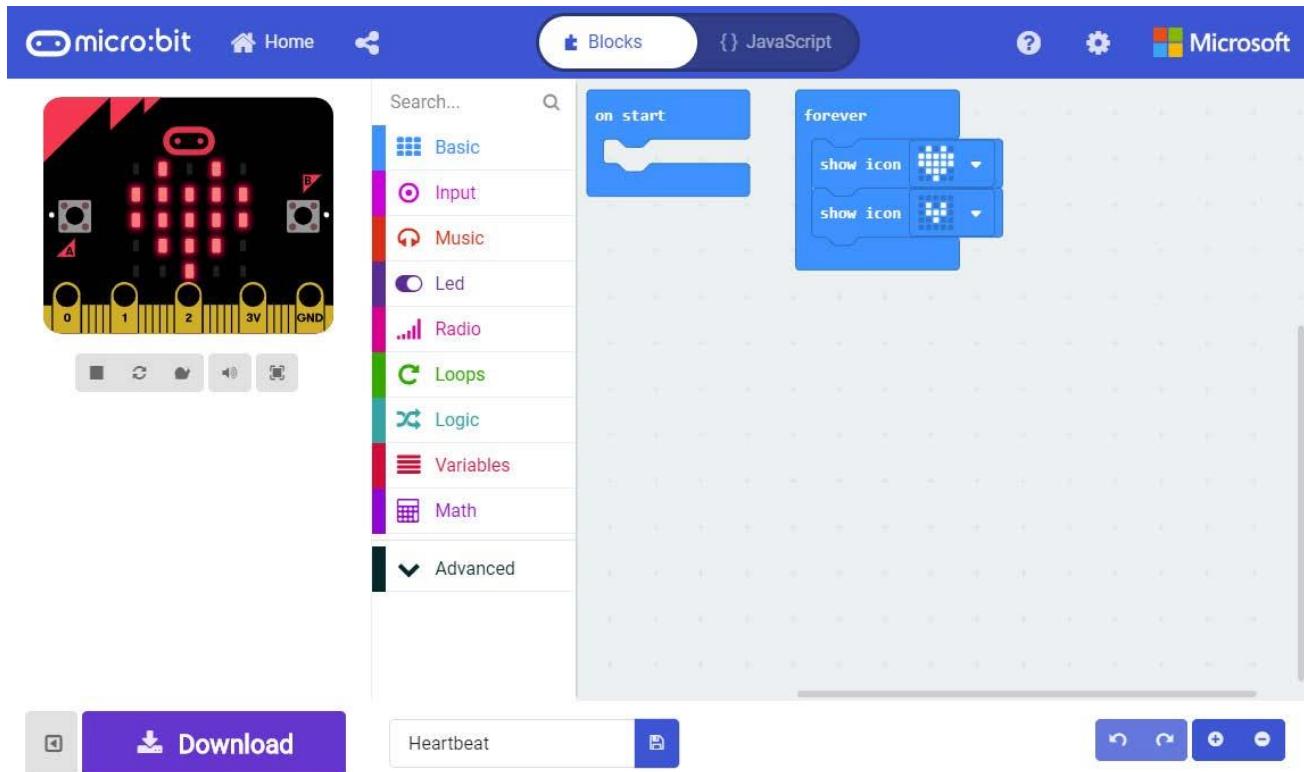
Select a .hex file to open.

Select file Heartbeat.hex

Go ahead!

Cancel

A few seconds later, the project is loaded successfully.



Python

If you are not interesting in python, you can skip this section.

Micro:bit can be programmed in Python. Since micro:bit is a microcontroller, the hardware difference does not support full Python. Here we use MicroPython, which is specially designed for micro:bit.

MicroPython is a lean and efficient implementation of the Python 3 programming language that includes a small subset of the Python standard library and is optimized to run on microcontrollers and in constrained environments.

We designed block code and Python code with similar function for each project.

There are two kinds of python editors for micro:bit, web version and software.

It is highly recommended to use software Mu as Python educator.

Mu. (<https://codewith.mu/en/download>)

Next, we will introduce Mu.

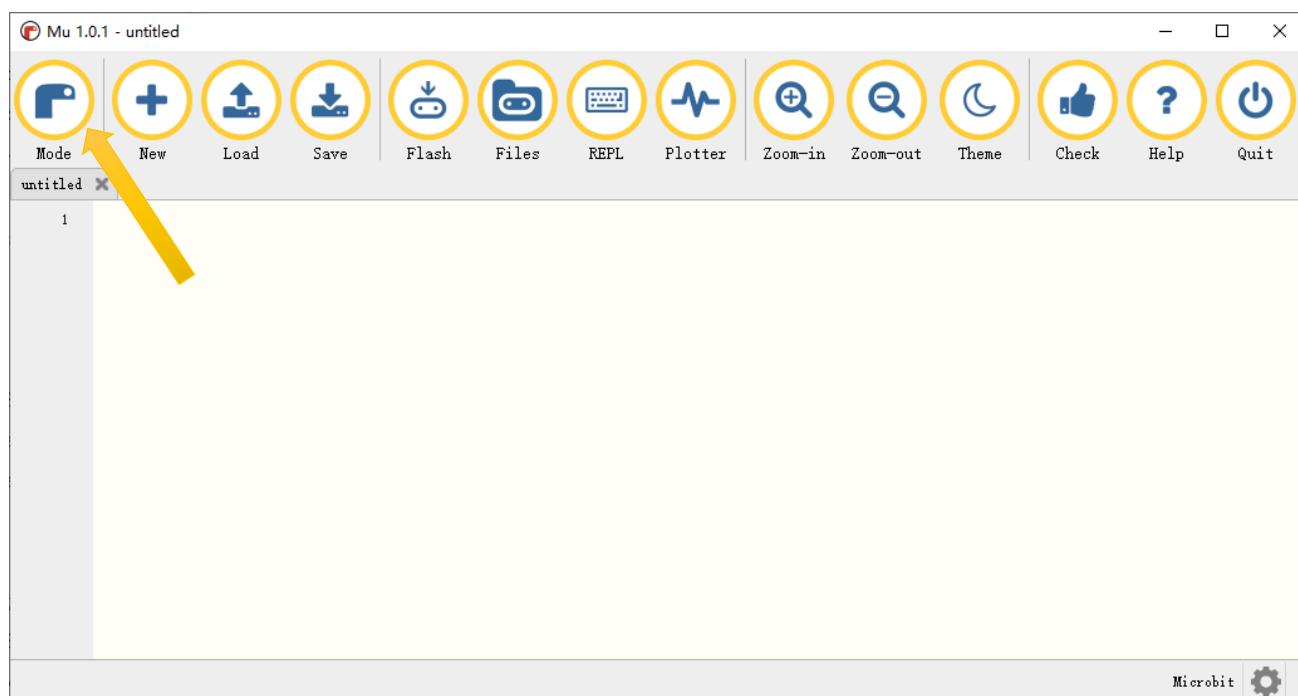
Mu

Mu is a Python code editor for beginner programmers based on extensive feedback given by teachers and learners.

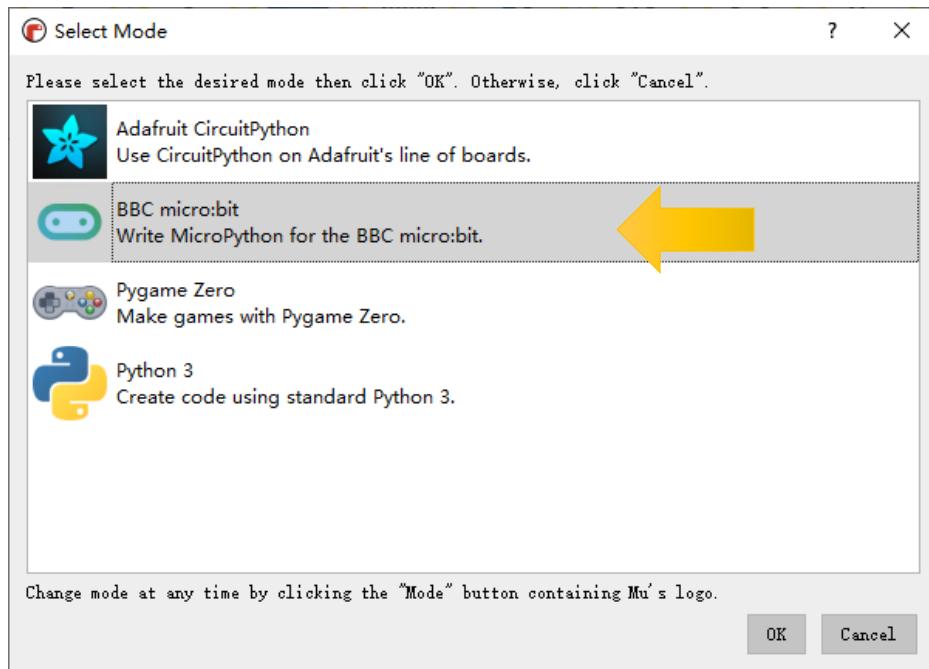
Official website: <https://codewith.mu/>

You can download it here: <https://codewith.mu/en/download>

Download and install it. Then open it, following interface appears:



Click the Mode button in the menu bar and select "BBC micro:bit" in the pop-up dialog box. Click "OK".



Import the .hex file. The path is as below:

File type	Path	File name
Python file	../project/1.1.Heartbeat	Heartbeat.py

Successful loading is show below.

You can also type the code by yourself.

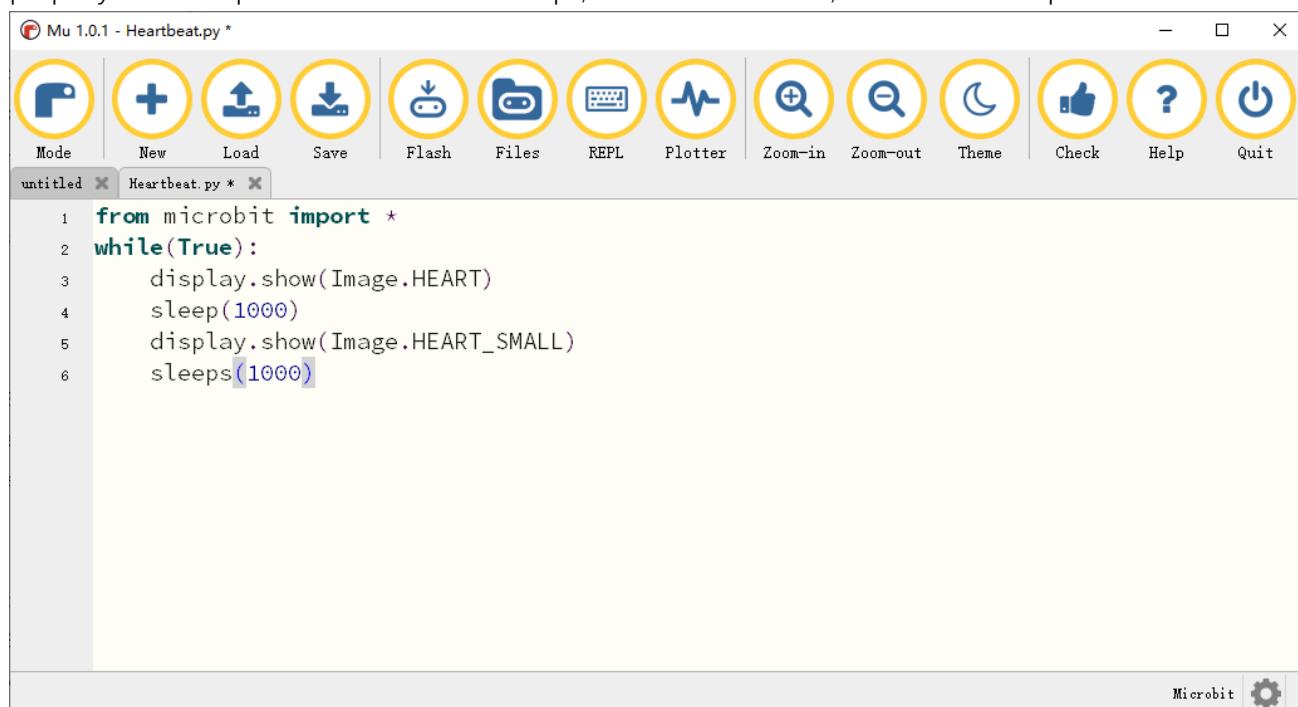
```

from microbit import *
while(True):
    display.show(Image.HEART)
    sleep(1000)
    display.show(Image.HEART_SMALL)
    sleep(1000)

```

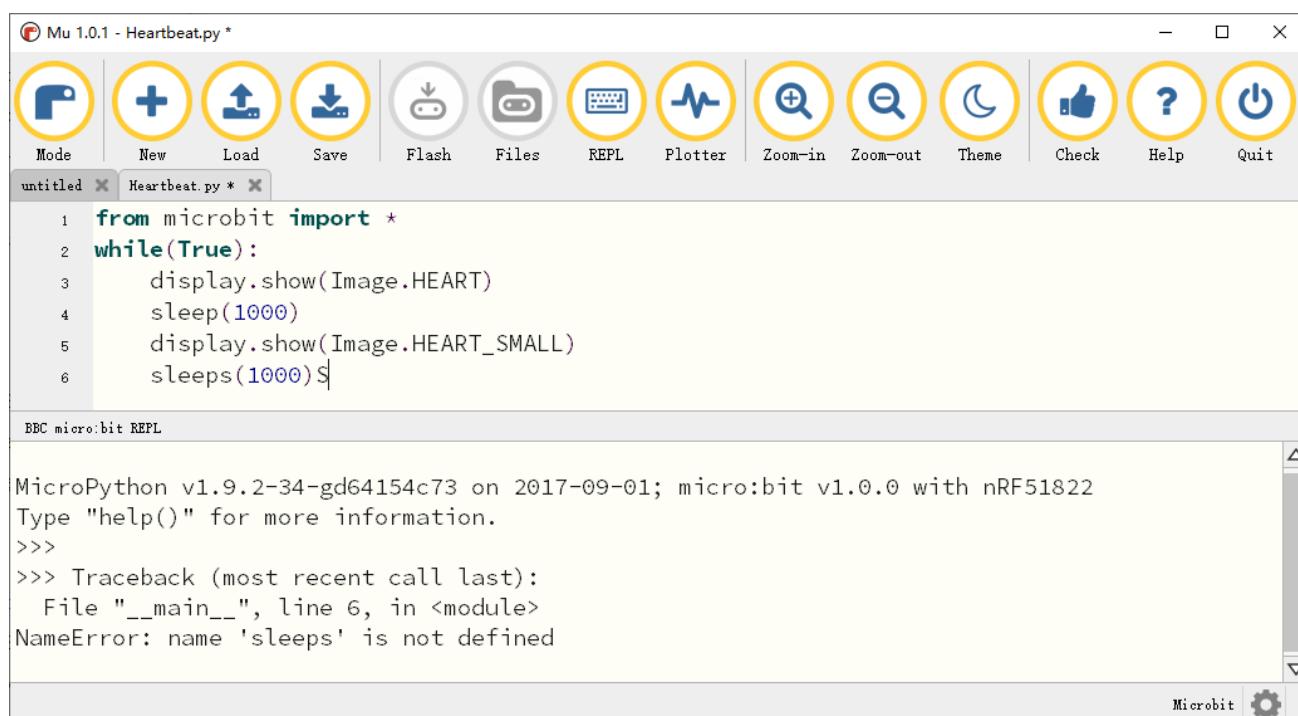
Use the micro USB cable to connect micro:bit and PC, and click the **Flash** button to download the program into micro:bit.

If your code has errors, you may be able to successfully download the code to micro:bit, but it will not work properly. If the sleep function is written as sleeps, click on flash button, the code will be uploaded into micro:bit.



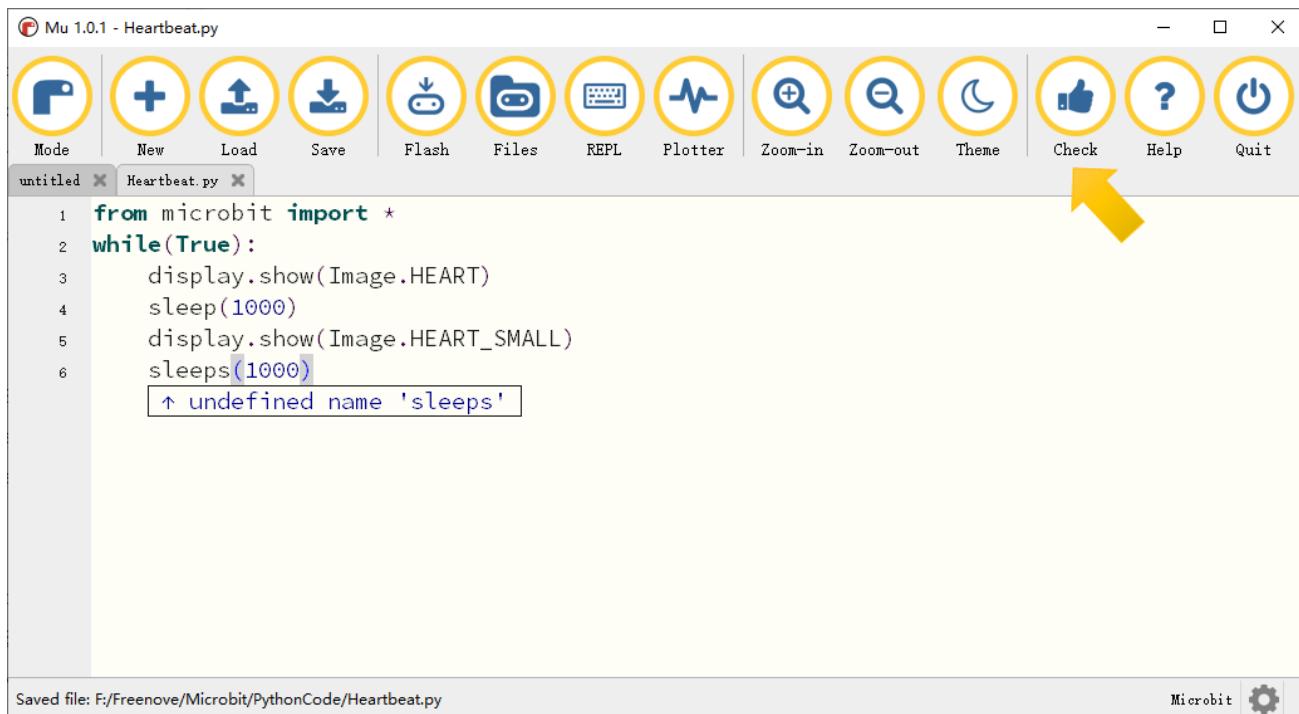
After the download is complete, led matrix prompts some error information, and the wrong line number.

Click the “REPL” button and press the reset button (the button on the back, not A, B) on micro:bit. The error message will be displayed in the REPL box, as shown below:

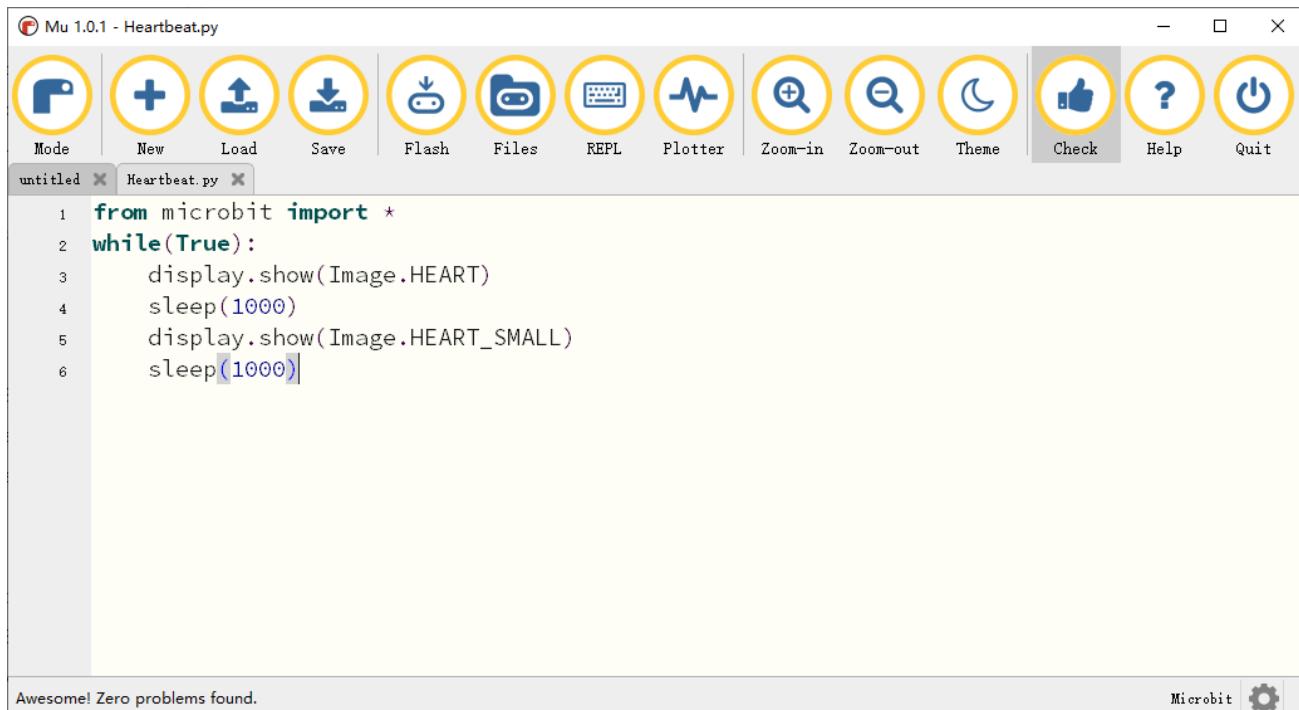


Click REPL again, you will close REPL mode. And then you can flash new code.

In order to ensure the code is correct, after completing the code, click the "Check" button to check the code for errors. As shown below, click the "Check" button. Then Mu will indicate the error of the code.



According to the error prompt, modify the code correctly. Then click the "Check" button again, Mu display no problem on the bar below.

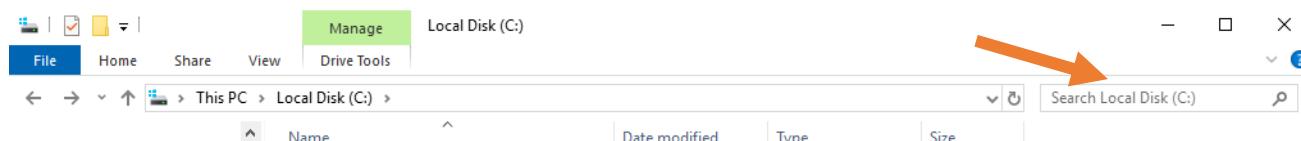


Import necessary Python file into micro:bit

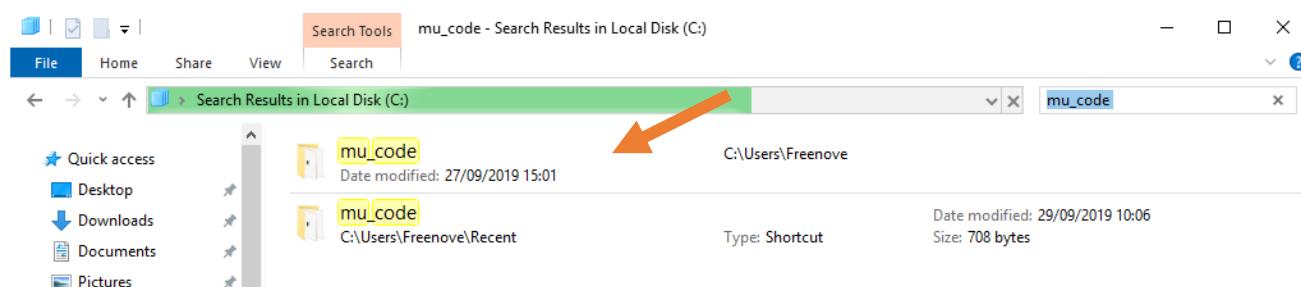
In the code of this tutorial, the LCD1602 module and DHT11 module are used. It is necessary to import "I2C_LCD1602_Class.py" and "DHT11_RW.py" into the micro:bit. You can skip this section if you don't use them. When you need, you can come back to import them.

The import method is as follows:

Search on the C drive and find the "mu_code" folder.



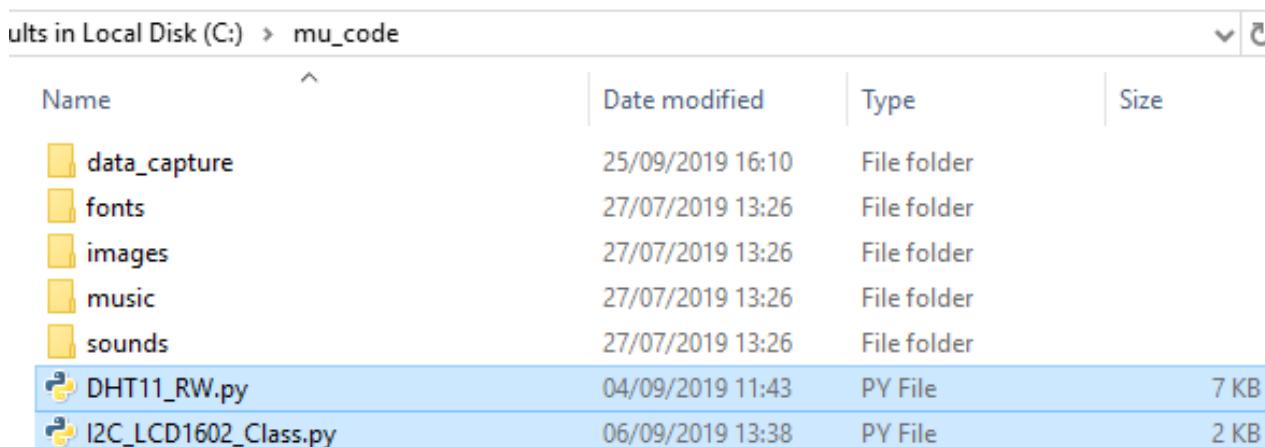
Double click on "mu_code" to enter the folder.



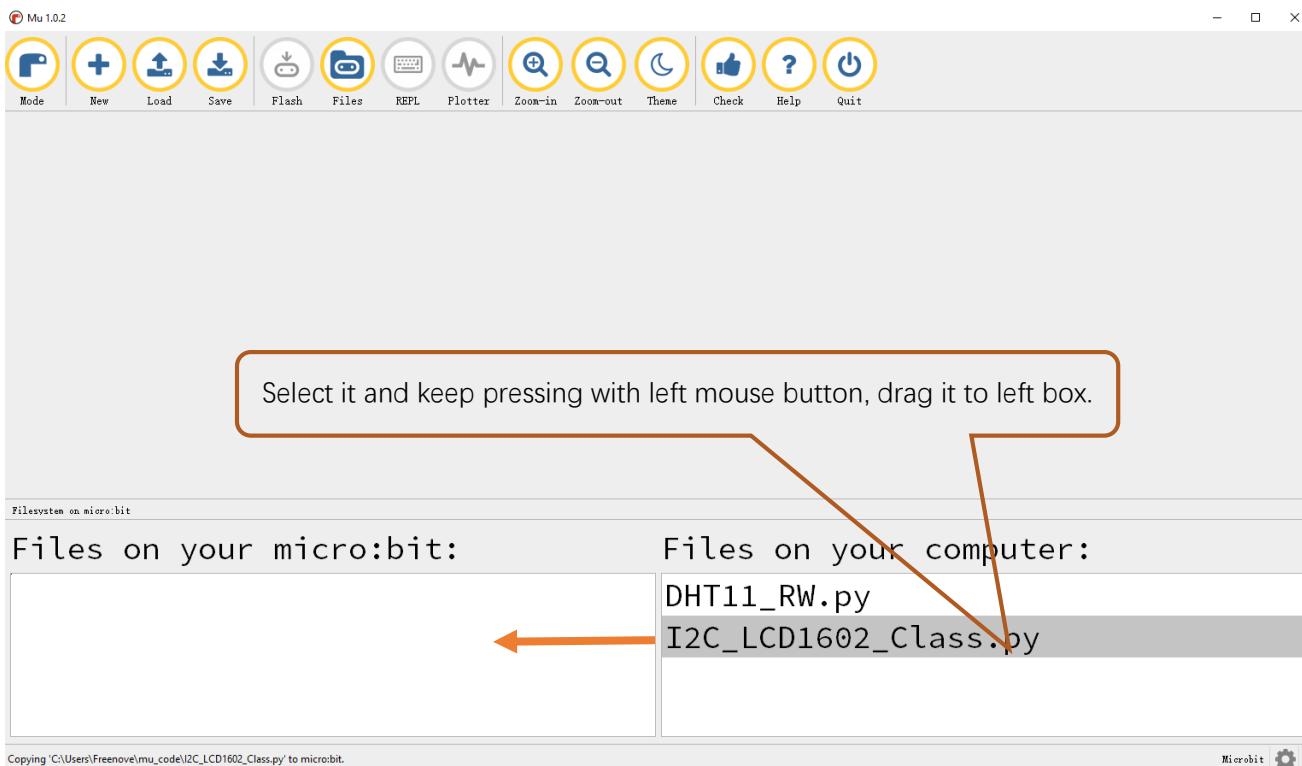
Copy "I2C_LCD1602_Class.py" and "DHT11_RW.py" from following path into "mu_code" directory.

File type	Path	File name
Python file	.. /Projects/PythonLibrary	I2C_LCD1602_Class.py DHT11_RW.py

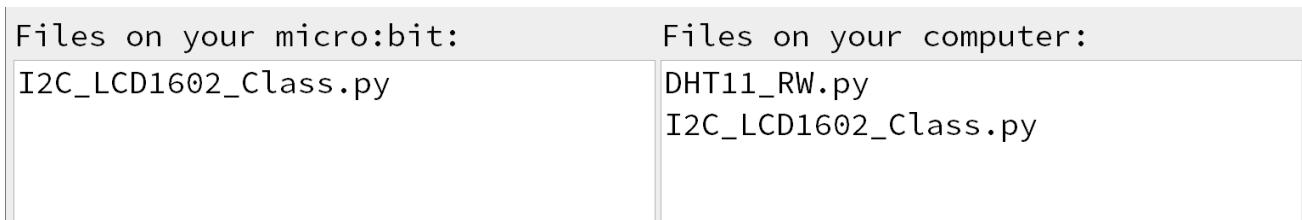
After paste successfully, you can see them as below:



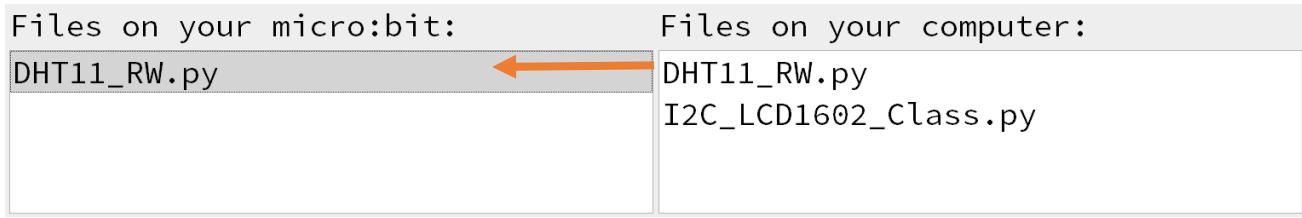
Open the Mu software, click "Files", and take "I2C_LCD1602_Class.py" as an example, drag "I2C_LCD1602_Class.py" into micro:bit.



After import successfully, you will see it on left.



The import method of "DHT11_RW.py" is the same as described above. You just need import the one you need to use.



Note, after you upload other into micro:bit. The content will be covered. You need to import them when next using.

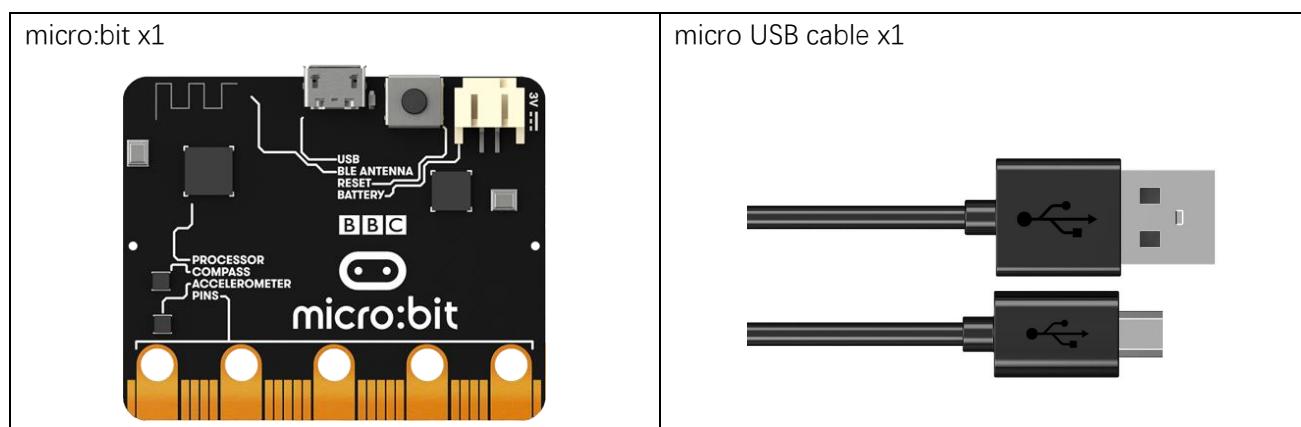
Chapter 1 Led matrix

The micro:bit integrates a 5x5 led matrix, which is used as a display to display numbers, text, or simple images, which is useful and interesting.

Project 1.1 Heartbeat

This project uses a pattern built in makecode to make a heartbeat animation.

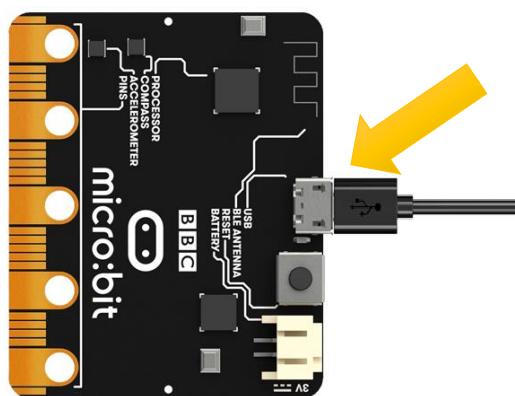
Component List



Circuit

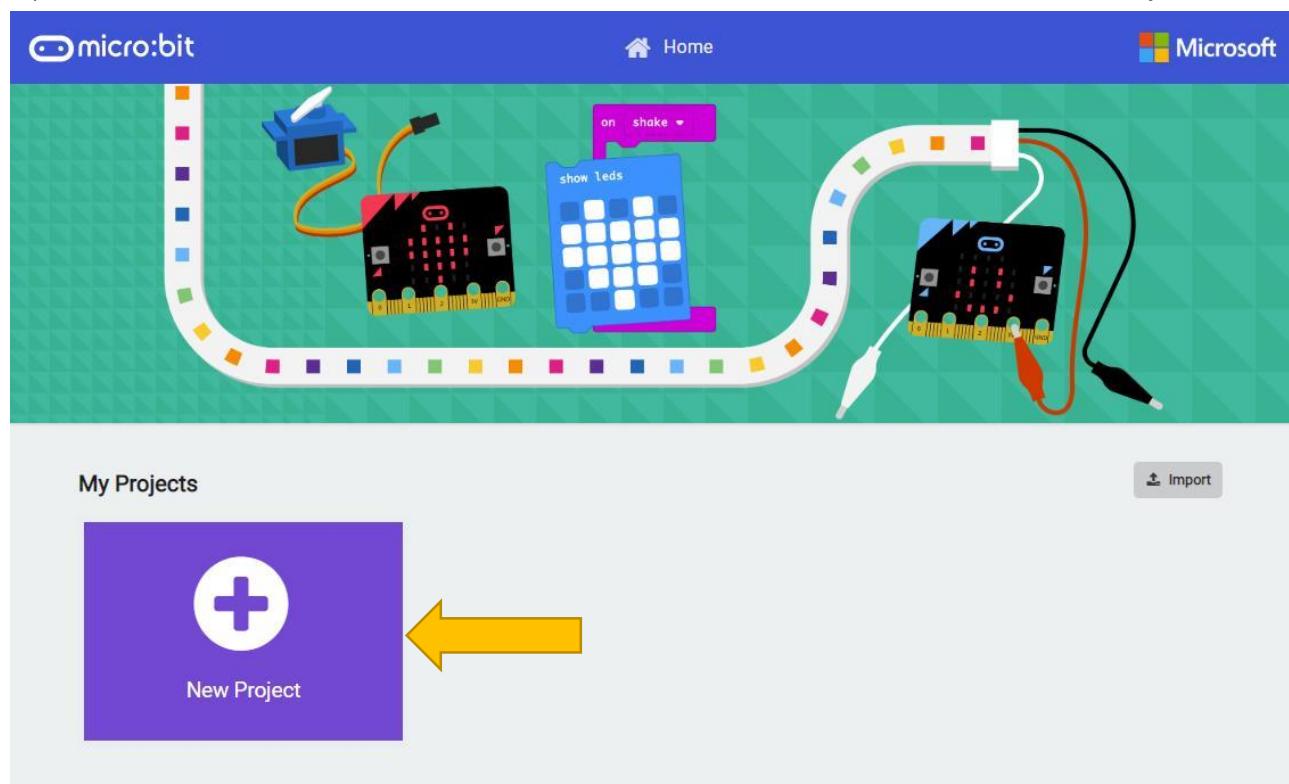
Connect micro:bit and PC via micro USB cable.

Hardware connection

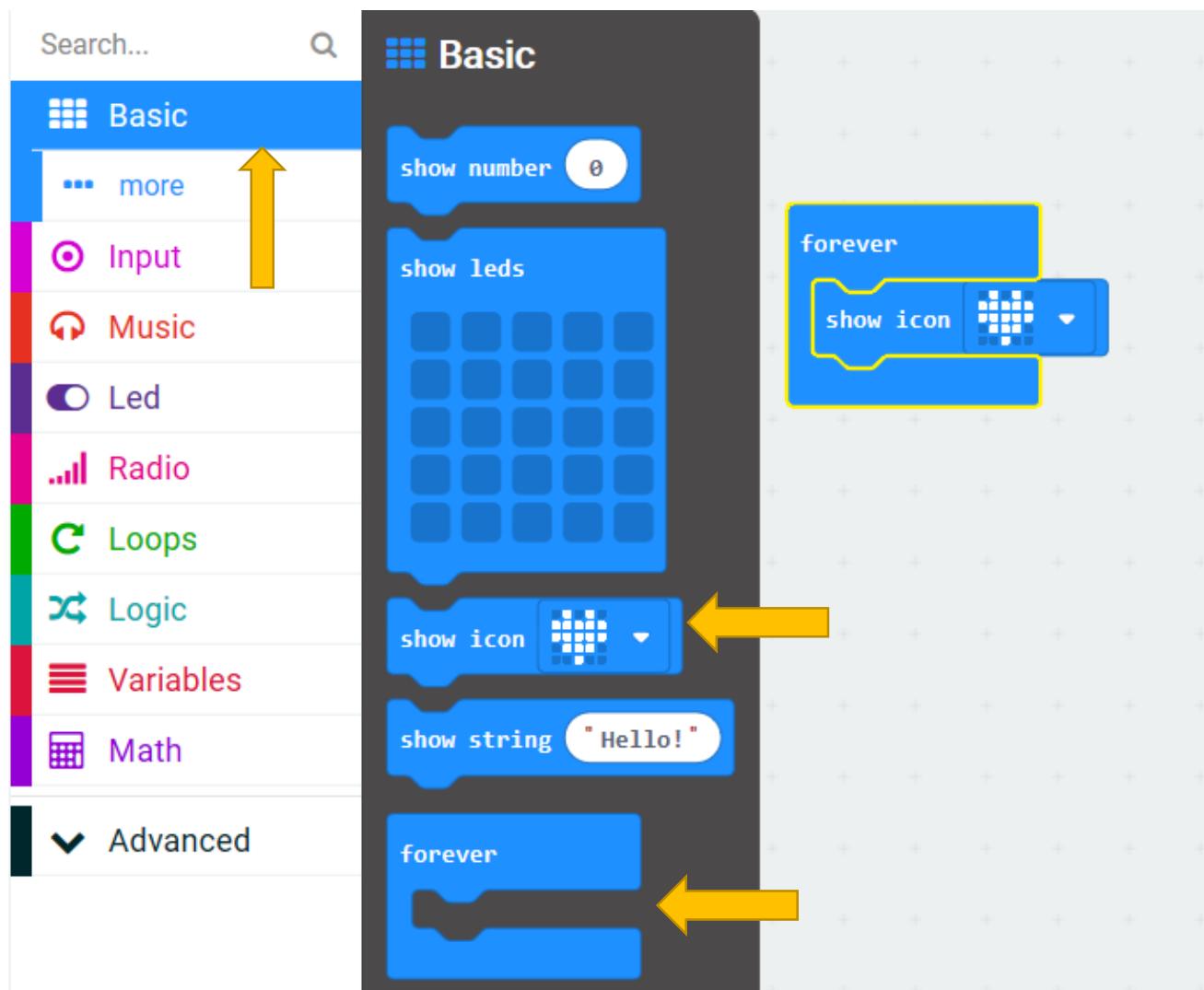


Block code

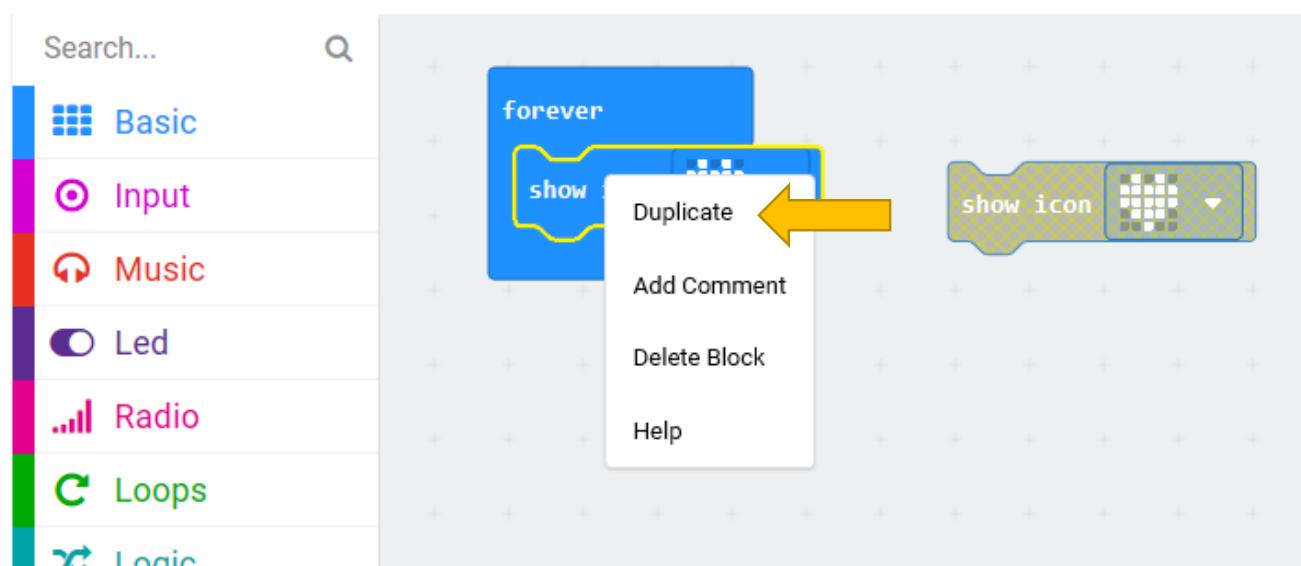
Open the makecode for the web version or makecode for the win10 version. Click on "New Project"



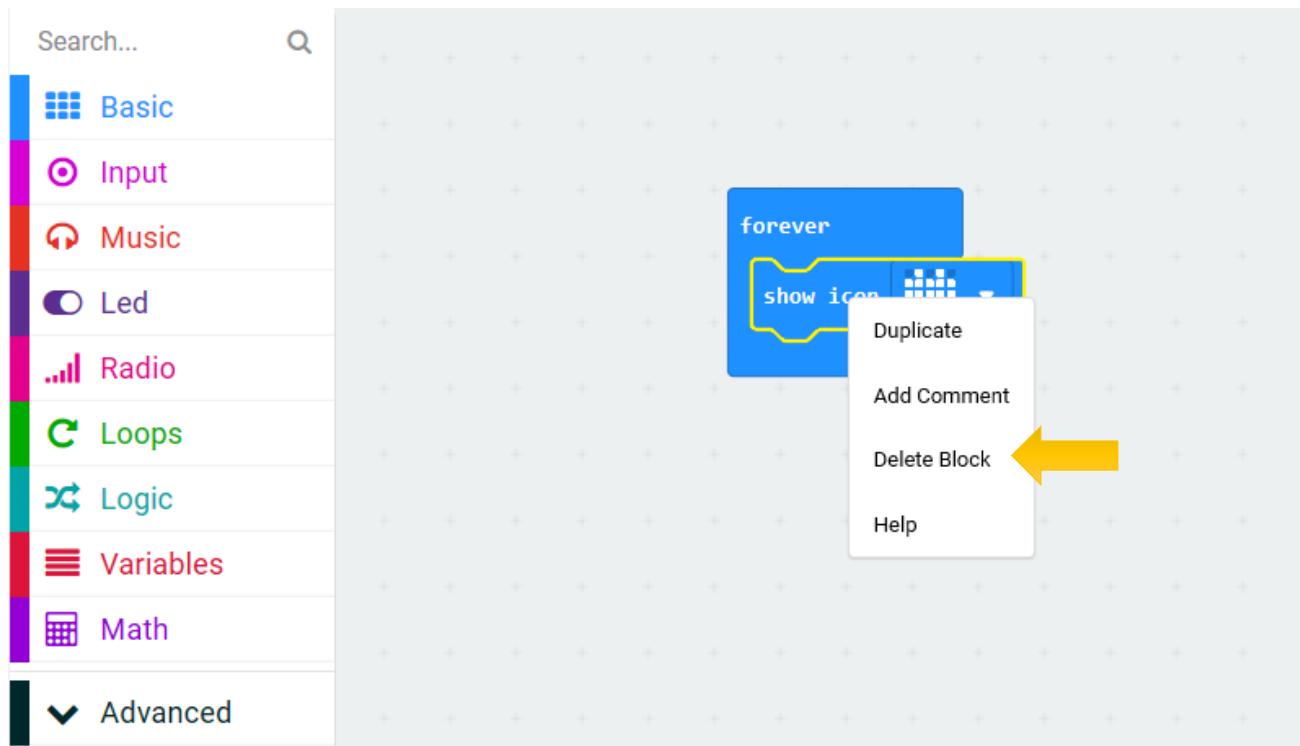
Click **basic** in the list on the left, select the desired code block, and drag it into the right code editing area.



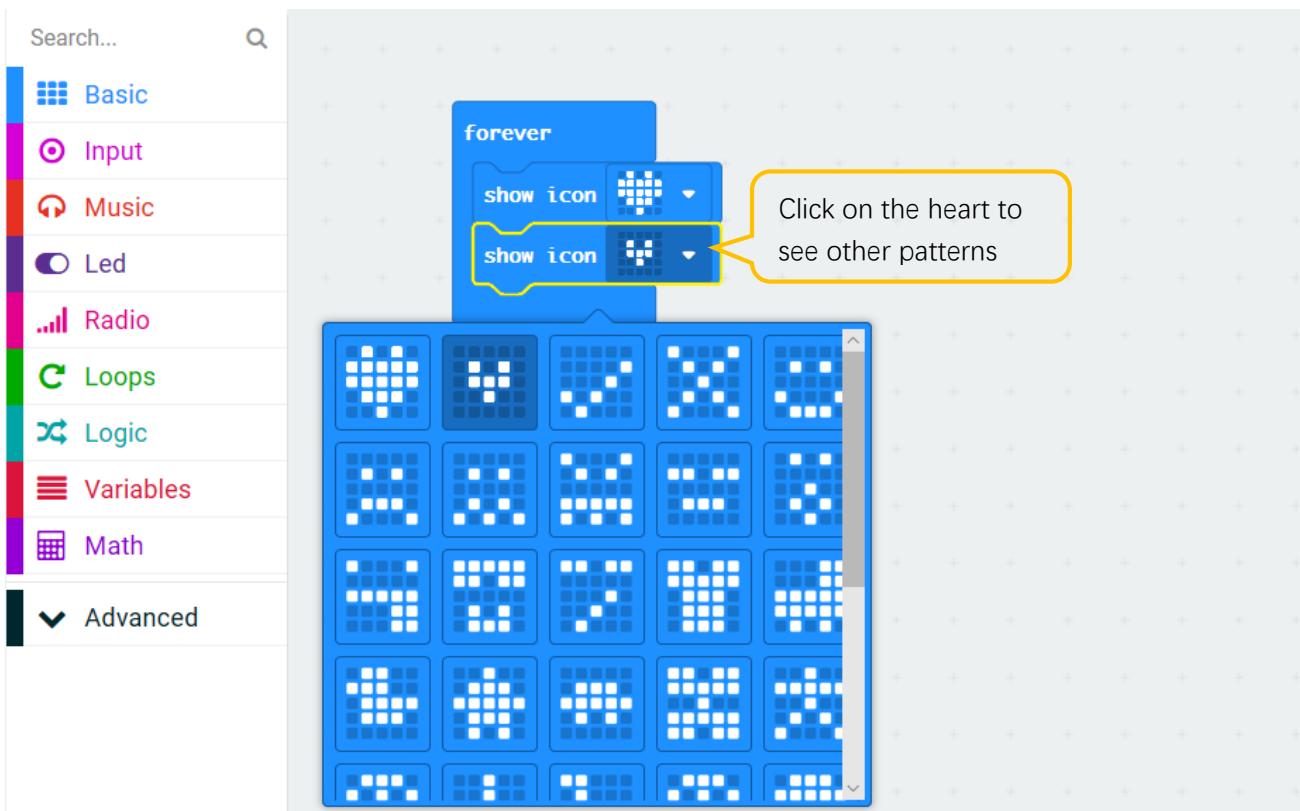
Right click the mouse and select Duplicate to duplicate the code block.



If you want to delete the block, you can right click on the block and select “Delete Block”. **You can also drag it to left to delete it.**



On the second block, click on the drop-down triangle next to the heart-shaped pattern on the block to display all the optional built-in patterns, select the second pattern, and a small heart shape.



If you did not master download, please refer to contents ([How to download?](#) [How to quick download?](#)).

This completes the block code part of the project.

Download the program to the microbit, the led matrix on the micro:bit will continue to display a large heart-shaped pattern and a small heart-shaped pattern, just like heartbeat.

Reference

Block	Function
	Shows the selected icon on the LED screen

Python Code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/01.1_Heartbeat	Heartbeat.py

After load successfully, the code is shown below:



```

Mu 1.0.2 - Heartbeat.py
Mode New Load Save Flash Files REPL Plotter Zoom-in Zoom-out Theme Check Help Quit
Heartbeat.py X
1 from microbit import *
2 while True:
3     display.show(Image.HEART) # Display heartbeat pattern
4     sleep(1000) # Stop for 1 second
5     display.show(Image.HEART_SMALL)
6     sleep(1000)

```

Download the program to the microbit, the led matrix on the micro:bit will continue to display a large heart-shaped pattern and a small heart-shaped pattern, just like heartbeat.

The following is the program code:

1	from microbit import *
2	while True:
3	display.show(Image.HEART)
4	sleep(1000)
5	display.show(Image.HEART_SMALL)
6	sleep(1000)

Python language is an interpreted language that is executed sequentially. In the code of this project, the micro:bit module is first imported, and then in a permanent loop statement, a large heart pattern and a small heart pattern are alternately displayed.

Next, we will explain the code line by line.

```
from microbit import *
```

Import everything in the microbit module, including functions, classes, variables, and more. You can also use **import microbit** directly. If you do this, you need to add "microbit." when you call the contents of this module in the program.

```
while True:
```

A permanent loop that will be executed by microbit constantly cyclically.

```
    display.show(Image.HEART)
```

Display heart pattern on LED matrix.

```
    sleep(1000)
```

Delay one second.

```
    display.show(Image.HEART)
    sleep(1000)
    display.show(Image.HEART_SMALL)
    sleep(1000)
```

Display the heart pattern on the LED matrix for one second, then display the small heart pattern for one second.

Reference

```
from microbit import *
```

Import everything in the microbit module, including functions, classes, variables, etc. You can use all the available content in the micro:bit module in the next program.

```
while True:
```

While is a loop statement, if the condition is true, the code in while is executed.

This code with True means that the code in the while is always executed cyclically.

```
display.show(image)
```

Display the image.

For more details about display,

please refer to: <https://microbit-micropython.readthedocs.io/en/latest/display.html>

For more details about image,

Please refer to: <https://microbit-micropython.readthedocs.io/en/latest/image.html>

```
sleep(t)
```

Delay for given number of milliseconds, should be positive or 0.

For more details about sleep function,

Please refer to: <https://microbit-micropython.readthedocs.io/en/latest/utime.html>

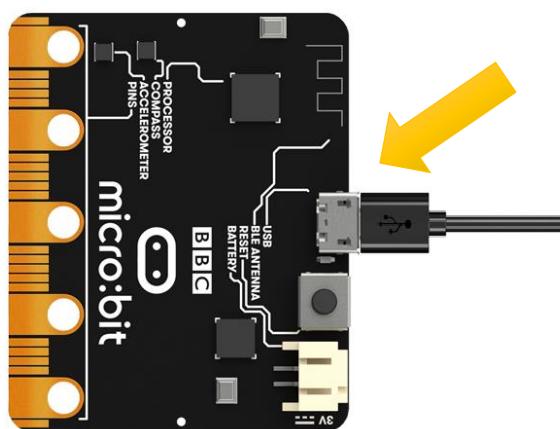
Project 1.2 Show Number

This project uses the led matrix of the micro bit to display numbers.

Circuit

Connect micro:bit and PC via micro USB cable.

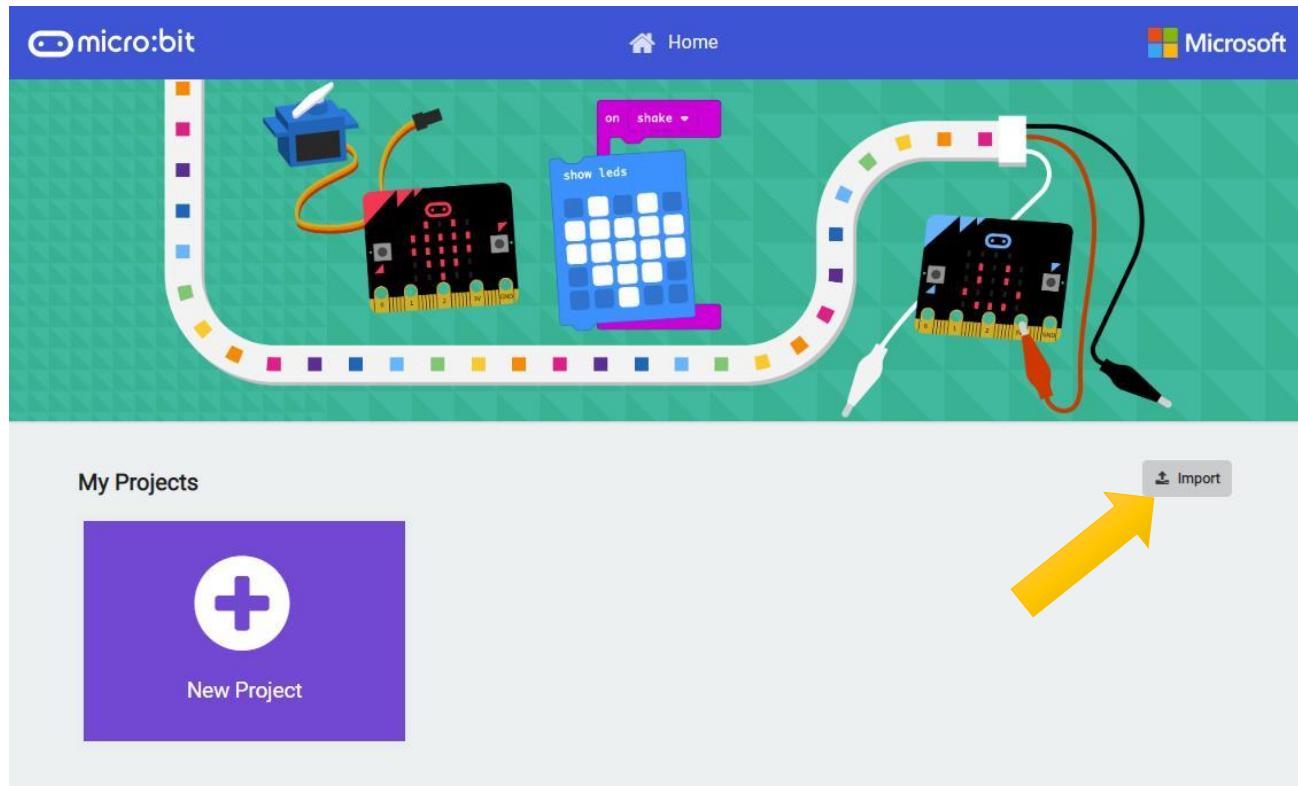
Hardware connection



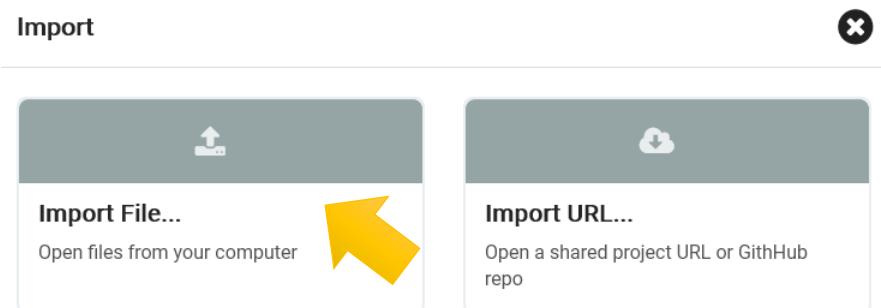
Block code

Open makecode first.

In this project, we will import the block code.



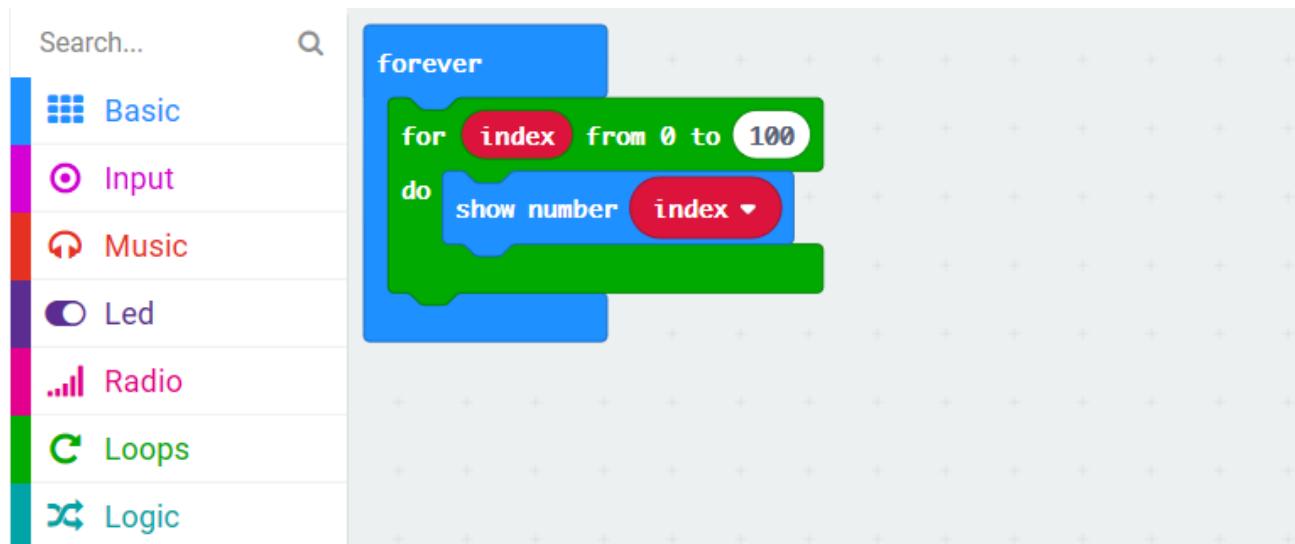
Click **Import**. Then click **Import File...**.



Import the .hex file. The path is as below:

File type	Path	File name
HEX file/Projects/BlockCode/01.2_ShowNumber	ShowNumber.hex

After load successfully, the code is shown as below:



Download the code into the micro:bit. After the download is complete, the micro:bit led matrix will start to display the numbers 1, 2, 3, 4...100. Then start again from 1 to 100, so that it will cycle permanently.

In this code, a for loop is used. Each time the loop is executed, the value of the variable index is increased by 1. When the value is greater than 100, the for loop is exited. In the body of the loop, the value of the numeric index is displayed.

Reference

Block	Function
	This is a for loop, the number (4) of loops can be set, each time the index is incremented by 1. Until the index is greater than the set value, then the loop ends.
	Show a number on the LED screen. It will slide left if it has more than one digit.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/01.2_ShowNumber	ShowNumber.py

After load successfully, the code is shown as below:



Download the code into the microbit. After the download is complete, the micro:bit led matrix will start to display the numbers 1, 2, 3, 4...100. Then start again from 1 to 100, so that it will cycle permanently.

The following is the program code:

```

1 from microbit import *
2 while True:
3     for index in range(0, 100):
4         display.scroll(str(index))
    
```

The code of this project, in a 0-100 for loop, scrolls through the cyclic number index, which is incremented by 1.

Reference

display.scroll(value)

Scrolls value horizontally on the display. If value is an integer or float it is first converted to a string using str().

For more information, please refer to: <https://microbit-micropython.readthedocs.io/en/latest/utime.html>

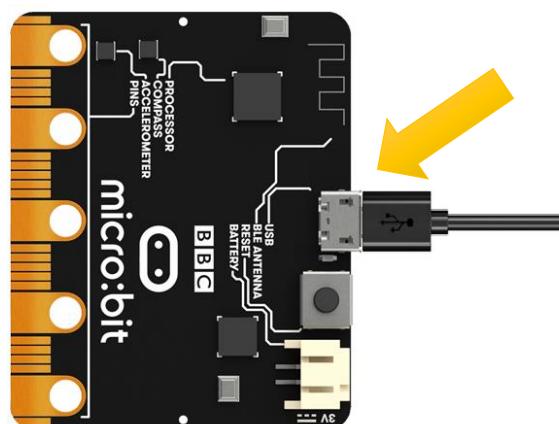
Project 1.3 Show Text

This project uses the LED matrix of micro:bit to display text (ASCII).

Circuit

Connect micro:bit and PC via micro USB cable.

Hardware connection



Block code

Open makecode first.

Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file	../Projects/BlockCode/01.3_ShowText	ShowText.hex

After load successfully, the code is shown as below:

A screenshot of the MakeCode editor interface. On the left, there is a sidebar with categories: Basic, Input, Music, Led, Radio, Loops, and Logic. The Basic category is currently selected. On the right, the main workspace shows a 'forever' loop block. Inside the loop, there is a 'show string' block with the text "Hello, Freenove!". The background of the workspace is a light gray grid representing the micro:bit's 8x8 pixel display.

Download the code into the microbit, the micro:bit led matrix will scroll from left to right to display "Hello, Freenove!"

Reference

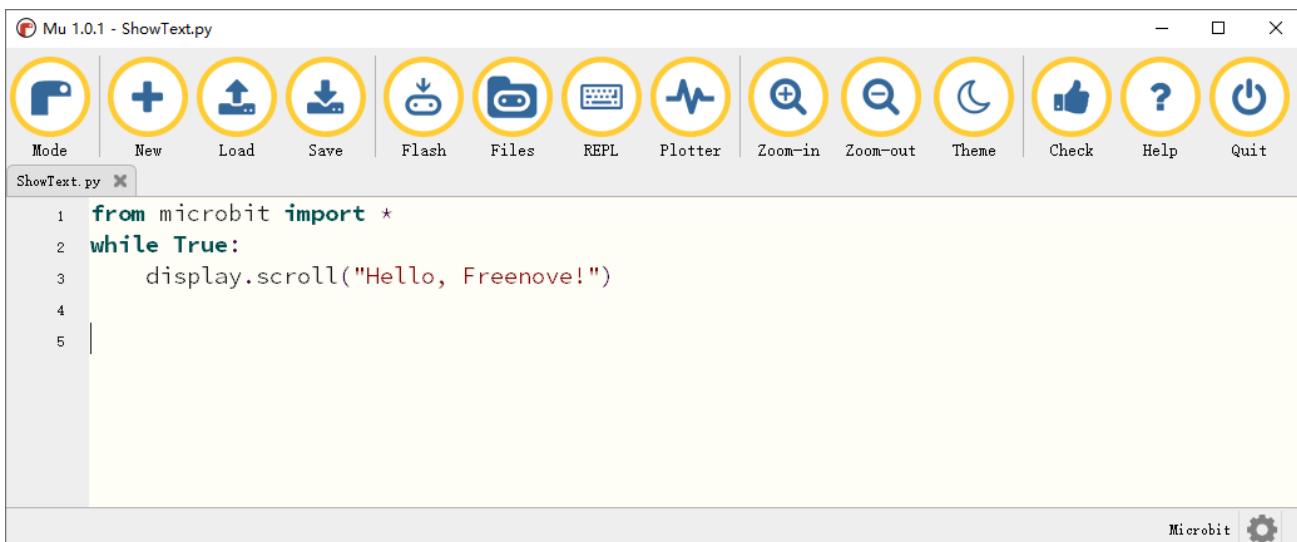
Block	Function
	Show a string on the LED screen. It will scroll to left if it's bigger than the screen.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file/Projects/PythonCode/01.3_ShowText	ShowText.py

After load successfully, the code is shown as below:



Download the code into the microbit, the micro:bit led matrix will scroll from left to right to display "Hello, Freenove!"

The following is the program code:

1	from microbit import *
2	while True:
3	display.scroll("Hello, Freenove!")

This code scrolls through the text "Hello, Freenove!" in a while loop.

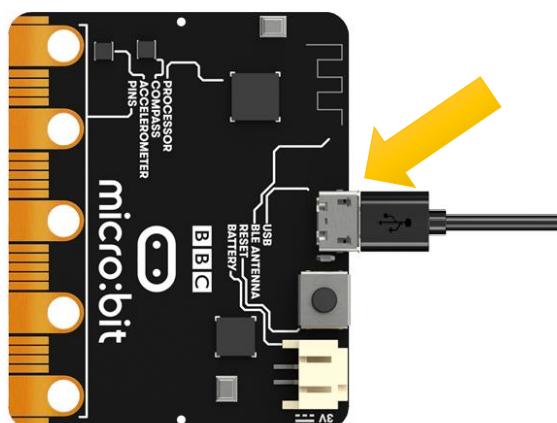
Project 1.4 Show Custom

This project uses a micro bit led matrix to display a custom pattern.

Circuit

Connect micro:bit and PC via micro USB cable.

Hardware connection



Block code

Open makecode first.

Import the .hex file. The path is as below:

[\(How to import project\)](#)

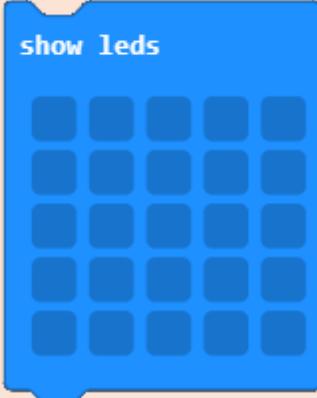
File type	Path	File name
HEX file/Projects/BlockCode/01.4_ShowCustom	ShowCustom.hex

After load successfully, the code is shown as below:



Check the connection of the circuit, and confirm that the circuit is connected correctly. Then download the code into the microbit, and the square pattern shown above will appear on the led matrix of the micro:bit.

Reference

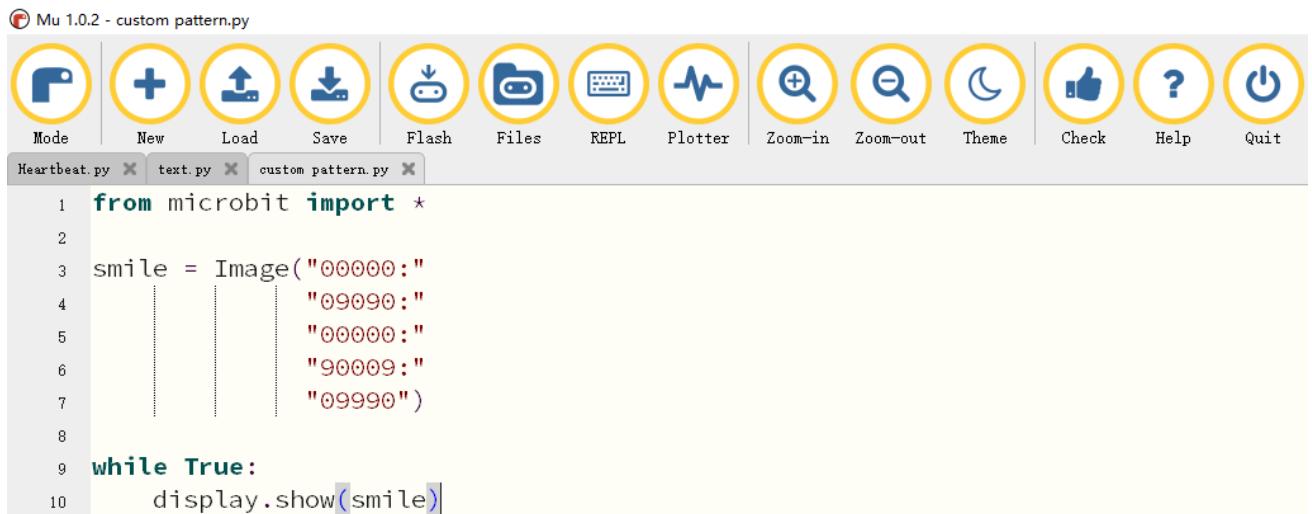
Block	Function
	Shows a picture on the LED screen.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/01.4_ShowCustom	ShowCustom.py

After load successfully, the code is shown as below:



```

Mu 1.0.2 - custom pattern.py
Mode New Load Save Flash Files REPL Plotter Zoom-in Zoom-out Theme Check Help Quit
Heartbeat.py x text.py x custom pattern.py x
1 from microbit import *
2
3 smile = Image("00000:"+
4 "09090:"+
5 "00000:"+
6 "90009:"+
7 "09990")
8
9 while True:
10     display.show(smile)

```

Download the code into the microbit, and a square pattern will appear on the micro:bit led matrix.

([How to download?](#))

The following is the program code:

```
1 from microbit import *
2
3 img = Image("00000:"
4             "09990:"
5             "09090:"
6             "09990:"
7             "00000")
8
9 while True:
10    display.show(img)
```

Create an image in the code and define it as **img**, then display the defined image in a while loop. As shown in the code below, the parameters in Image consist of 5 strings. Each line of characters corresponds to a row of LEDs. Each digit in represents the brightness of an LED. The value ranges from 0 to 9, the larger the number, the higher the brightness of the LED.

```
1 img = Image("00000:"
2             "09990:"
3             "09090:"
4             "09990:"
5             "00000")
```

Reference

```
img = Image("00000:"
           "09990:"
           "09090:"
           "09990:"
           "00000")
```

Create an image of LED, and set brightness of LED.

For more information, please refer to:

<https://microbit-micropython.readthedocs.io/en/latest/image.html>

Chapter 2 Built In Button

Keyboards or buttons are important tools for human-computer interaction. We often use keyboards to enter text, type commands, control devices, and more. Two programmable buttons A and B are integrated on the micro:bit to easily control the micro:bit to do some action.

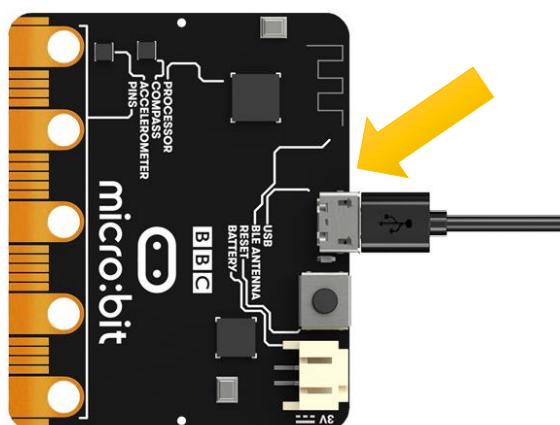
Project 2.1 Button A and B

This project uses micro:bit integrated buttons A and B. When different buttons are pressed, micro:bit displays different patterns.

Circuit

Connect micro:bit and PC via micro USB cable.

Hardware connection



Block code

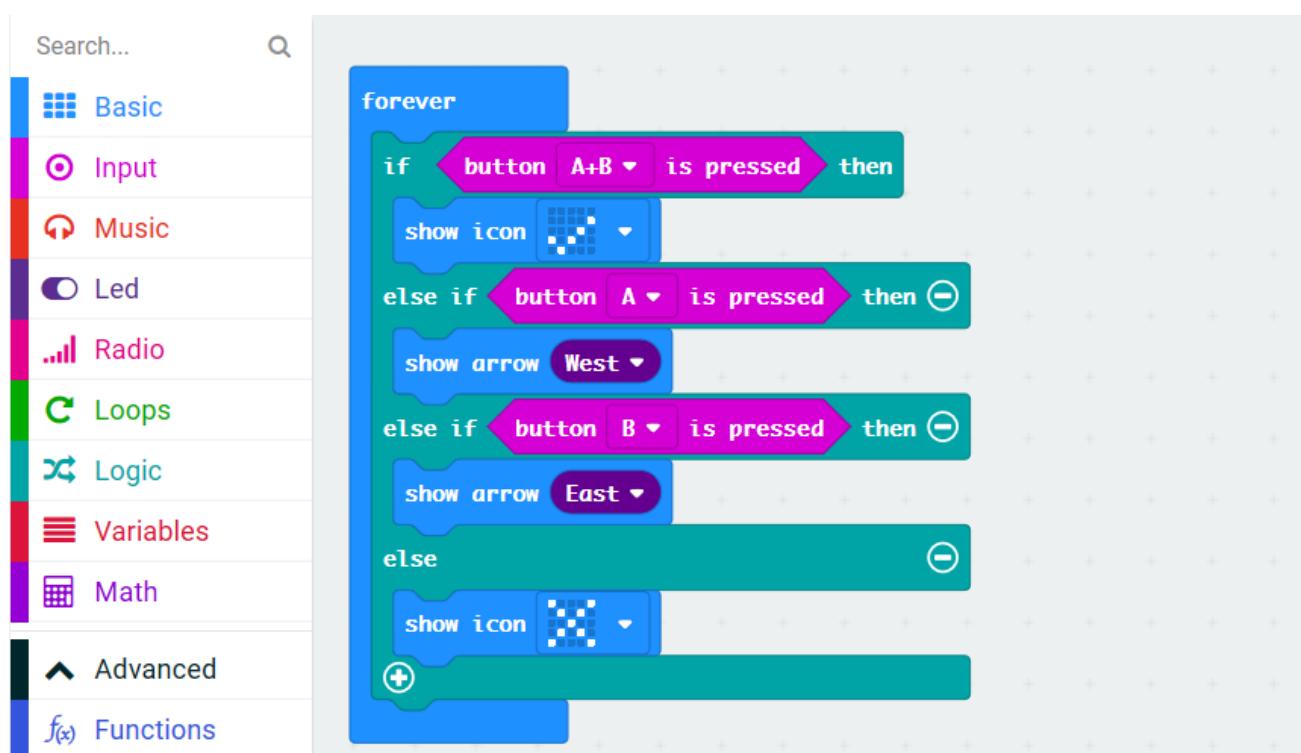
Open makecode first.

Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file	./Projects/BlockCode/02.1_BuiltInButton	BuiltInButton.hex

After load successfully, the code is shown as below:



Download the code into micro:bit. When button A is pressed, the micro:bit led matrix will display an arrow pointing to button A. When button B is pressed, the micro:bit led matrix will display a pointing point. The arrow of button B, when the buttons A and B are pressed at the same time, the micro:bit led matrix will display a check mark. When no button is pressed, the micro:bit led matrix displays a cross.

Reference

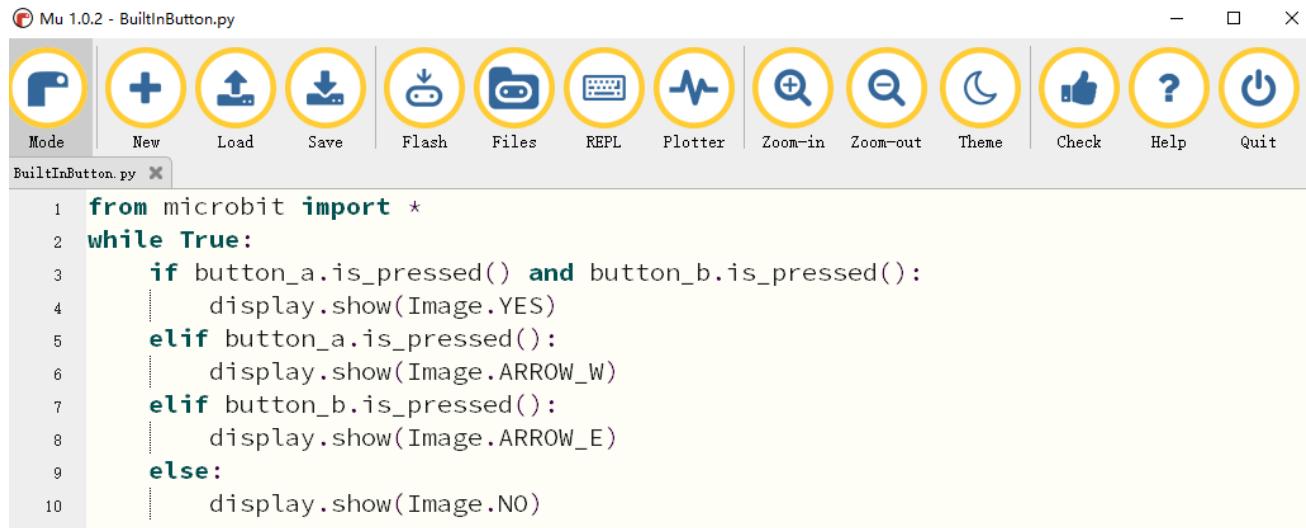
Block	Function
button A ▾ is pressed	Check whether a button is pressed right now. The micro:bit has two buttons: button A and button B.
on button A ▾ pressed	This handler works when button A or B is pressed, or A and B together.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file/Projects/PythonCode/02.1_BuiltInButton	BuiltInButton

After load successfully, the code is shown as below:



```

1 from microbit import *
2 while True:
3     if button_a.is_pressed() and button_b.is_pressed():
4         display.show(Image.YES)
5     elif button_a.is_pressed():
6         display.show(Image.ARROW_W)
7     elif button_b.is_pressed():
8         display.show(Image.ARROW_E)
9     else:
10        display.show(Image.NO)

```

Download the code into micro:bit. When button A is pressed, the micro:bit led matrix will display an arrow pointing to button A. When button B is pressed, the micro:bit led matrix will display a pointing point. The arrow of button B, when the buttons A and B are pressed at the same time, the micro:bit led matrix will display a check mark. When no button is pressed, the micro:bit led matrix displays a cross.

The following is the program code:

```

1 from microbit import *
2
3 while True:
4     if button_a.is_pressed() and button_b.is_pressed():
5         display.show(Image.YES)
6     elif button_a.is_pressed():
7         display.show(Image.ARROW_W)
8     elif button_b.is_pressed():
9         display.show(Image.ARROW_E)
10    else:
11        display.show(Image.NO)

```

Use the if-elif-else statement to determine when the button is pressed. First, when the buttons A and B are pressed at the same time, a check mark is displayed.

```
if button_a.is_pressed() and button_b.is_pressed():
    display.show(Image.YES)
```

Then, determine in turn if the buttons A or B is individually pressed, and the case where no button is pressed.

```
elif button_a.is_pressed():
    display.show(Image.ARROW_W)
elif button_b.is_pressed():
    display.show(Image.ARROW_E)
else:
    display.show(Image.NO)
```

Note that it is necessary to first determine if buttons A and B are pressed at the same time. If-elif-else statement will make the micro:bit execute only one situation. If the state with two button pressed is placed in last, the situation of pressing A or B will appear ahead, then the statement will end, and then sentence met the state with two button pressed will never be execute.

Reference

is_pressed()

Returns True if the specified button is currently being held down, and False otherwise.

For more information, please refer to <https://microbit-micropython.readthedocs.io/en/latest/button.html>



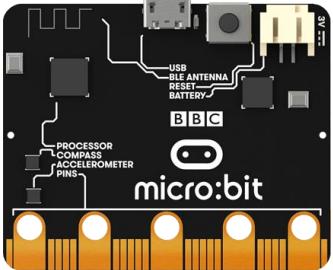
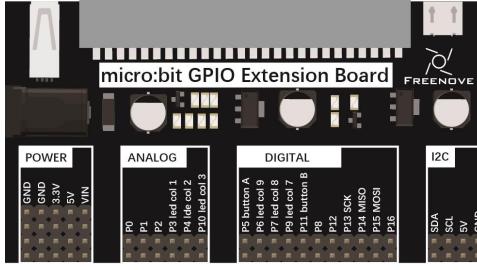
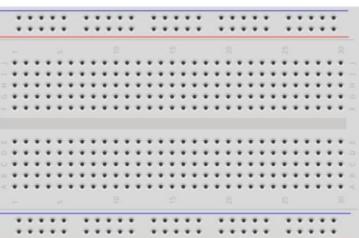
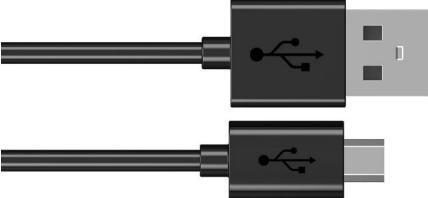
Chapter 3 LED

This section we will learn how to control external LEDs.

Project 3.1 Blink

This project uses micro:bit to control LED blinking.

Component list

Microbit x1	Expansion board x1	
		
Breakboard x1	USB cable x1	
		
Jumper F/M x2	Resistor 220Ω x1	LED x1
		

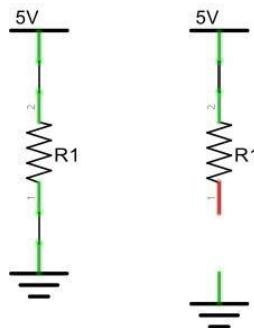
Circuit knowledge

Current

The unit of current (I) is ampere (A). $1A=1000mA$, $1mA=1000\mu A$.

Closed loop consisting of electronic components is necessary for current.

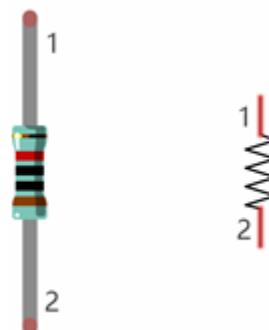
In the figure below: the left is a loop circuit, so current flows through the circuit. The right is not a loop circuit, so there is no current.



Resistor

The unit of resistance(R) is ohm(Ω). $1M\Omega=1000k\Omega$, $1k\Omega=1000\Omega$.

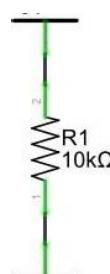
Resistor is an electrical component that limits or regulates the flow of current in an electronic circuit. The left is the appearance of resistor. And the right is the symbol of resistor represented in circuit.



Color rings attached to the resistor are used to indicate its resistance. For more details of resistor color code, please refer to the appendix of this book.

With the same voltage there will be less current with more resistance. And the links among current, voltage and resistance can be expressed by the formula below: $I=U/R$.

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



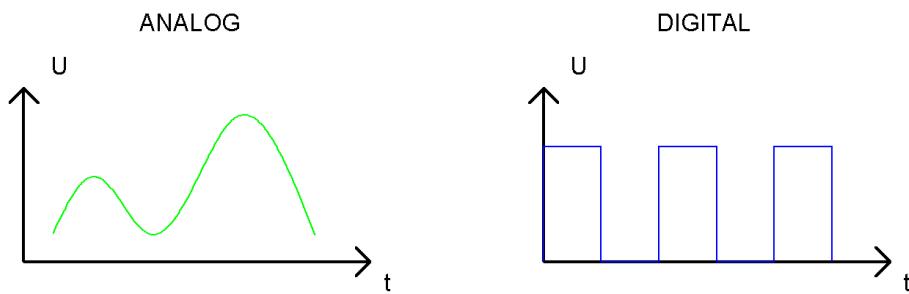
Do not connect the two poles of power supply with low resistance, which will make the current too high to damage electronic components.

Analog signal and Digital signal

The analog signal is a continuous signal in time and value. On the contrary, digital signal is a discrete signal in time and value.

Most signals in life are analog signals, for example, the temperature in one day is continuously changing, and will not appear a sudden change directly from 0°C to 10°C, while the digital signal is a jump change, which can be directly from 1 to 0.

Their difference is shown as following figure.



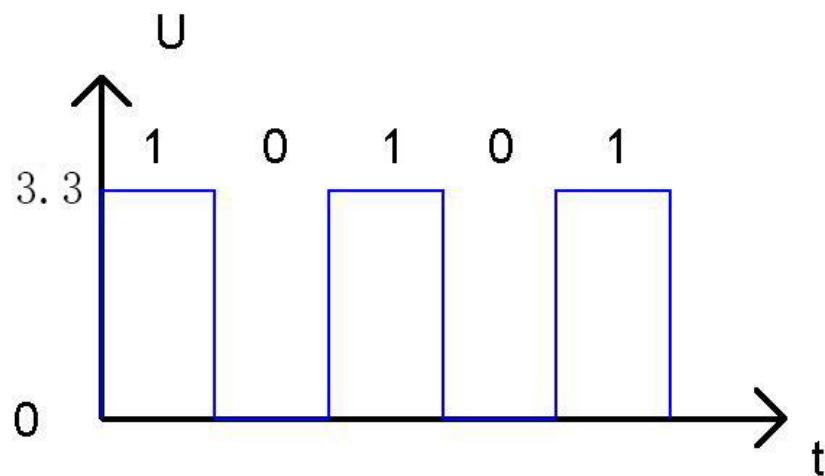
In practical application, the frequently used digital signal is binary signal, that is 0 and 1. The binary signal only has two forms (0 or 1), so it has strong stability. And digital signal and analog signal can be converted to each other.

Low level and high level

In circuit, the form of binary (0 and 1) is present as low level and high level.

Low level is generally equal to ground voltage (0V). High level is generally equal to the operating voltage of components.

The low level of Micro:bit is 0V and high level is 3.3V, as shown below. When IO port on Micro:bit outputs high level, components of small power can be directly driven, like LED.



Component knowledge

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

Jumper

Jumper is a kind of wire. It is designed to connect the components together with its two terminals by inserting

Jumpers have male end (pin) and female end (slot), so jumpers can be divided into the following 3 types.

Jumper M/M



Jumper F/F



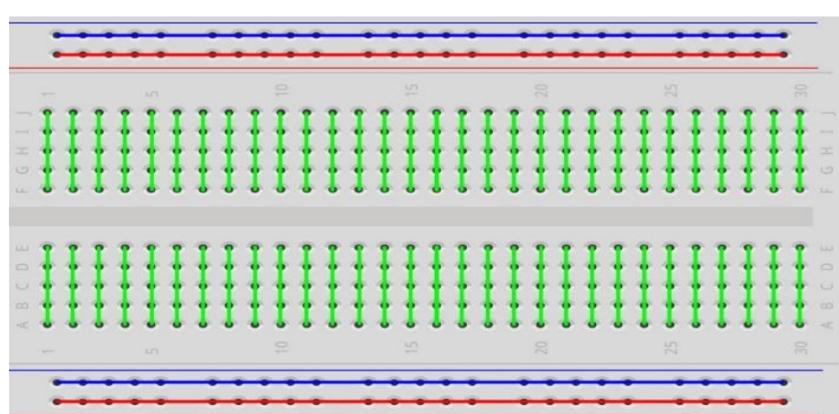
Jumper F/M



Breadboard

There are many small holes on breadboard to connect Jumper.

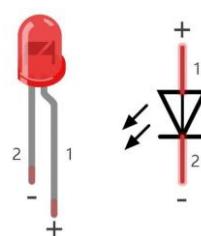
Some small holes are connected inside breadboard. The following figure shows the inner links among those holes.



LED

LED is a kind of diode. LED will shine only if the long pin of LED is connected to the positive electrode and the short pin is connected to negative electrode.

This is also the features of the common diode. Diode works only if the voltage of its positive electrode is higher than its negative electrode.



The LED can not be directly connected to power supply, which can damage component. A resistor with certain resistance must be connected in series in the circuit of LED.

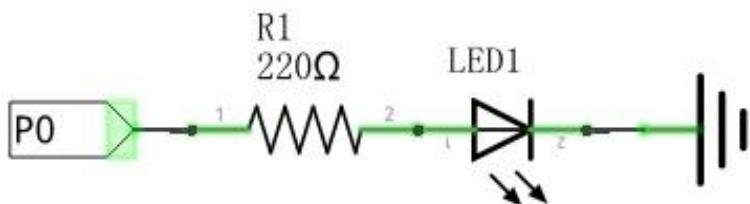
Circuit

When wiring, it is recommended to disconnect all the power supplies in the circuit, and then follow the circuit diagram to build the circuit (**micro:bit board can not be inserted reverse**),

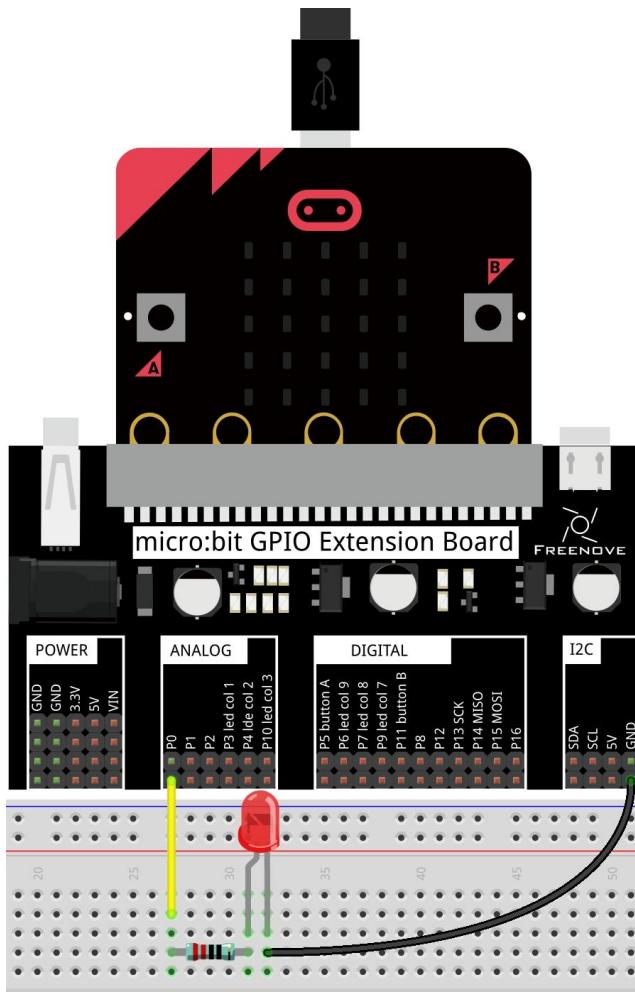
LED long pin is positive pole, connecting to resistor. LED short pin is negative pole, connected to GND. After the circuit is built and confirmed, use the USB cable to connect the PC to the micro:bit to power the circuit.

Note that there should be no short circuit in the circuit, especially 5V and GND, 3.3V and GND. A short circuit can cause the circuit to malfunction or even damage your micro:bit.

Schematic diagram



Hardware connection



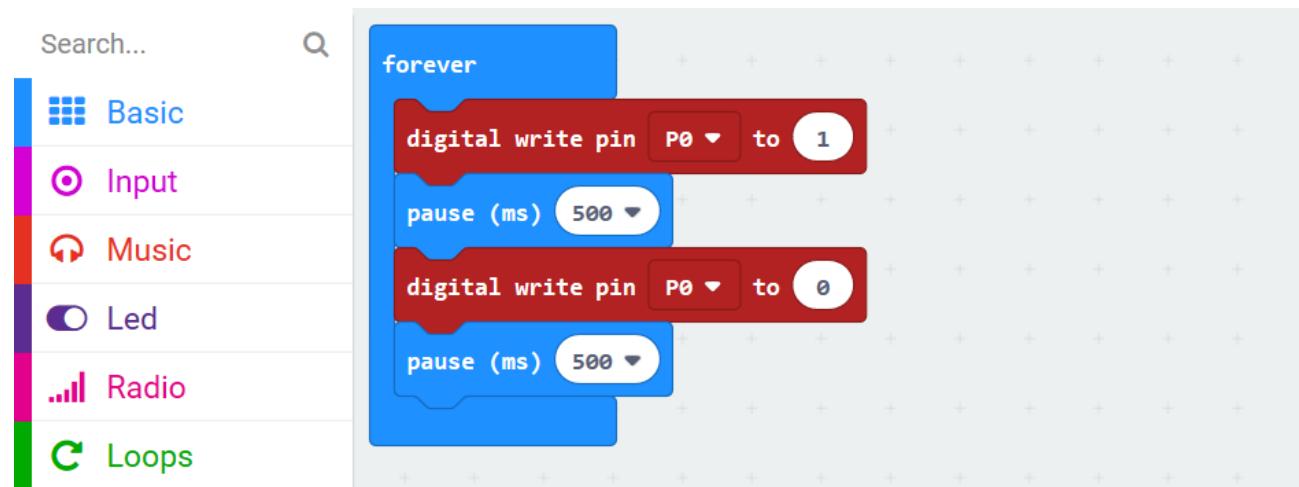
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

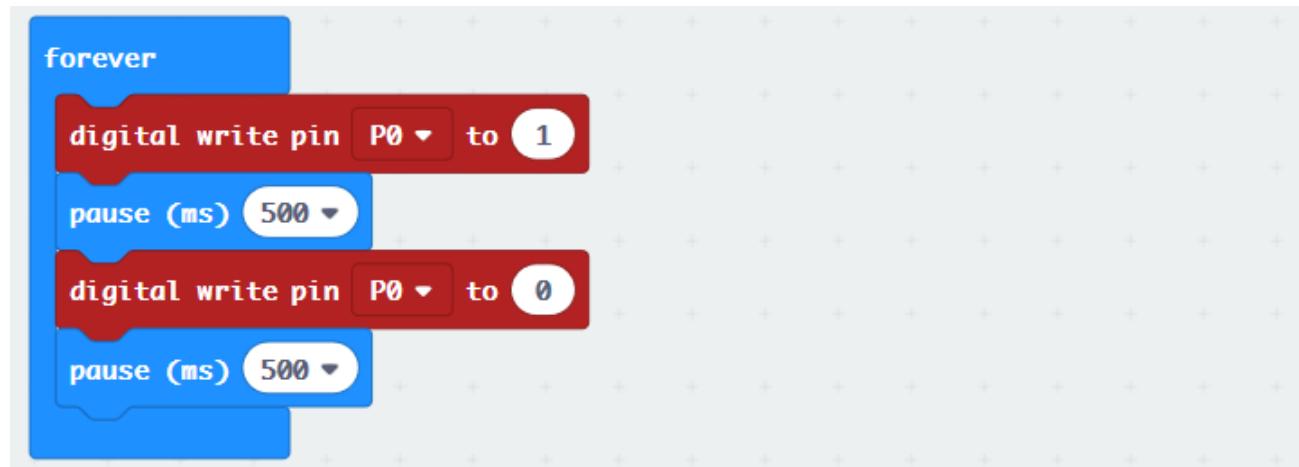
File type	Path	File name
HEX file	./Projects/BlockCode/03.1_Blink	Blink.hex

After import successfully, the code is shown as below:



Download the code into the micro:bit and the LED on the breadboard will begin to blink.

In the code, write 1 to the P0 port to turn on the LED. After waiting for 500ms, write 0 to the P0 port to turn off the LED. After waiting for 500ms, the LED will be turned on again, so that it will cycle continuously to achieve blink effect.



Reference

Block	Function
	Pause the program for the number of milliseconds you say. You can use this function to slow your program down.
	Write a digital (0 or 1) signal to a pin on the micro:bit board.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/03.1_Blink	Blink.py

After load successfully, the code is shown as below:

The screenshot shows the Mu 1.0.1 IDE interface. The title bar says "Mu 1.0.1 - Blink.py". The toolbar has icons for Mode (Microbit), New, Load, Save, Flash, Files, REPL, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The code editor window contains the following Python code:

```

1 from microbit import *
2
3 while True:
4     pin0.write_digital(1)
5     sleep(500)
6     pin0.write_digital(0)
7     sleep(500)

```

The status bar at the bottom right shows "Microbit" and a gear icon.

Download the code into micro:bit and the LED on the breadboard will start to blink.

[\(How to download?\)](#)

The following is the program code:

```
1 from microbit import *
2
3 while True:
4     pin0.write_digital(1)
5     sleep(100)
6     pin0.write_digital(0)
7     sleep(100)
```

In the code, write 1 to the P0 port to turn on the LED. After waiting for 500ms, write 0 to the P0 port to turn off the LED. After waiting for 500ms, the LED will be turned on again, so that it will cycle continuously to achieve blink effect.

Write a high level to pin P0.

```
pin0.write_digital(1)
```

Delay 500ms

```
sleep(500)
```

Then write low level, and then delay 500ms. Repeat actions above.

Reference

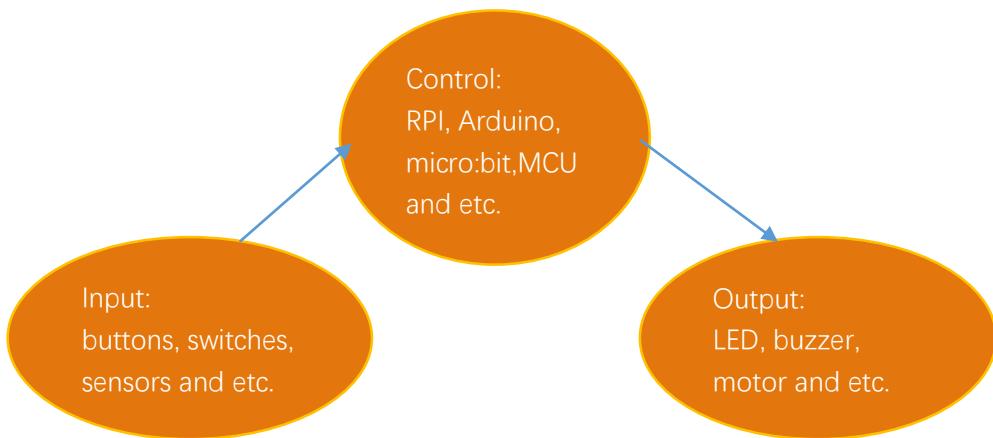
```
pin.write_digital(value)
```

Set the pin to high if **value** is 1, or to low, if it is 0.

For more information, please refer to:<https://microbit-micropython.readthedocs.io/en/latest/pin.html>

Chapter 4 Button and LED

Usually, there are three essential parts in a complete automatic control device: INPUT, OUTPUT, and CONTROL. In last section, the LED module is the output part and micro:bit is the control part. In practical applications, we not only just let the LED lights flash, but make the device sense the surrounding environment, receive instructions and then make the appropriate action such as lights the LED, make a buzzer beep and so on.

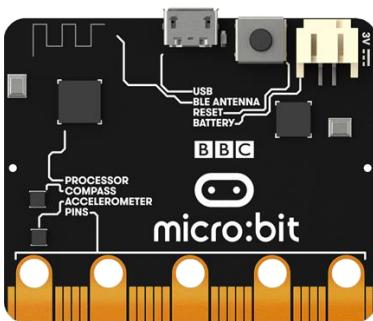
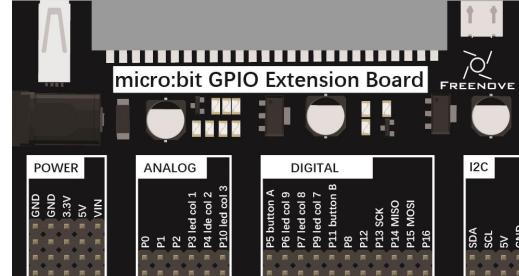
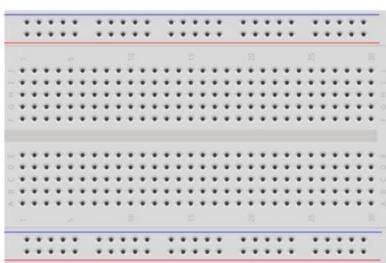
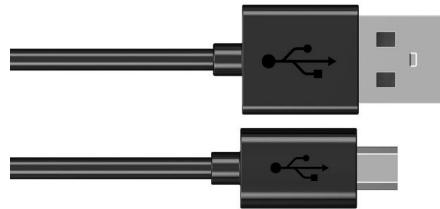


Next, we will build a simple control system to control LED through a button.

Project 4.1 Control LED by Button

In the project, we will control the LED state through a button. When the button is pressed, LED will be turn on, and when it is released, LED will be turn off.

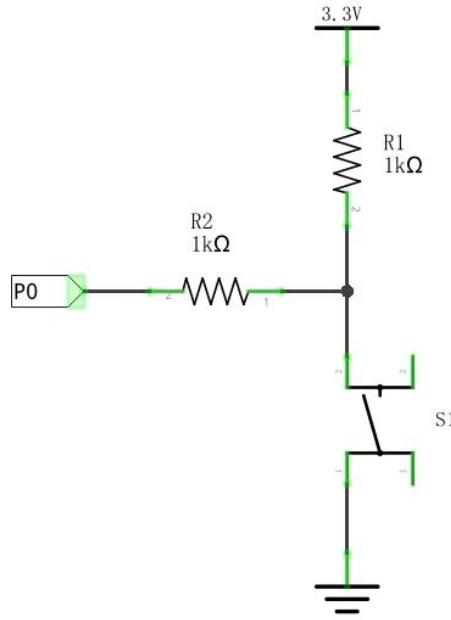
Component list

Microbit x1	Expansion board x1		
			
Breakboard x1	USB cable x1		
			
FM x4 MM x1	Resistor 220Ω x1  10kΩ x2 	LED x1	Push Button x1

Circuit knowledge

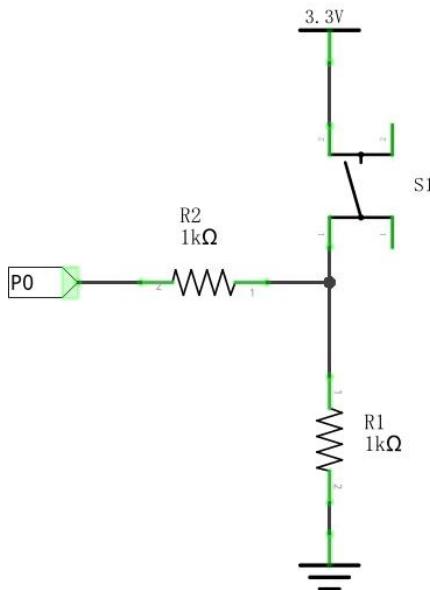
Connection of Push Button

We connect push button directly to power line of the circuit to control the LED to light on or off. In digital circuits, we need to use the push button as the input signal. The recommended connection is as follows:



In the above circuit diagram, when the button is not pressed, 3.3V (high level) will be detected by I/O port; and 0V (low level) when the button is pressed. The role of Resistor R2 here is to prevent the port from being set to output high level by accident, which could be connected directly to the cathode and cause a short circuit when the button is pressed.

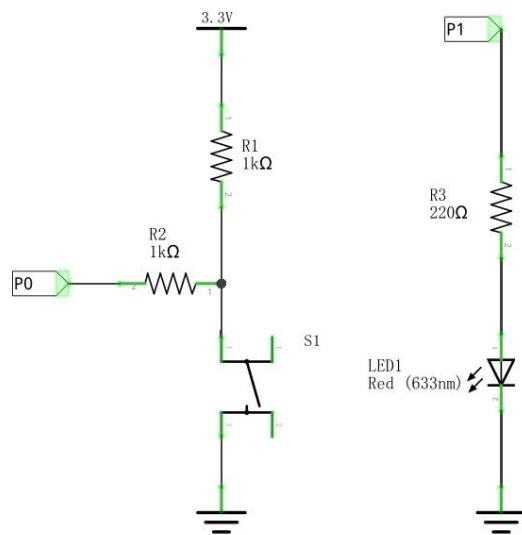
The following diagram shows another connection, in which the level detected by I/O port is opposite to above diagram, when the button is pressed or not.



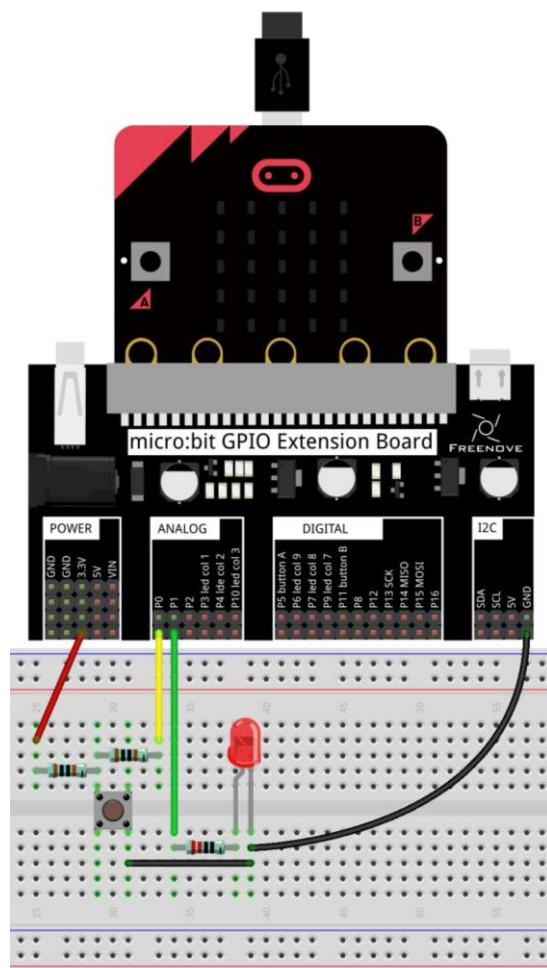
Circuit

The P0 pin detects the button and the P1 pin controls the LED.

Schematic diagram



Hardware connection



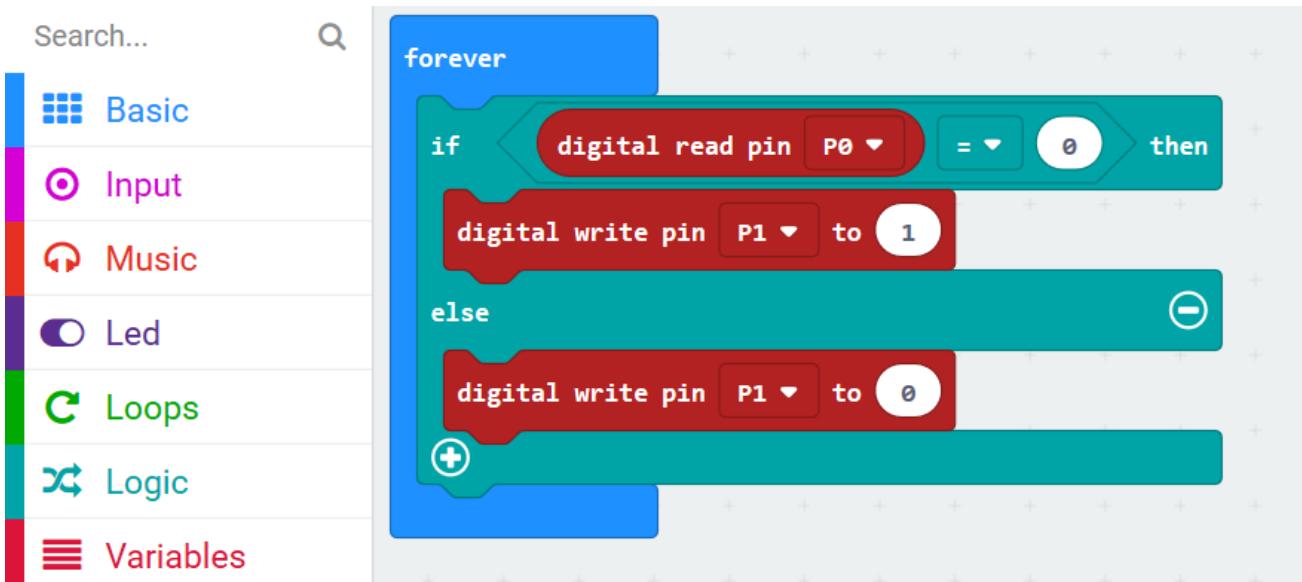
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file	./Projects/BlockCode/04.1_ButtonAndLED	ButtonAndLED.hex

After import successfully, the code is shown as below:



Download the code into micro:bit. When the button is pressed, the led will be turned on. When the button is released, the led will be turned off.

In the program, read the level of the P0 pin to determine if the button is pressed.



If the read P0 pin is low, it indicates that the button is pressed, and make P1 pin output 1, then LED will be turned on.



Otherwise, P1 pin output 0, and the LED will be turned off.



Reference

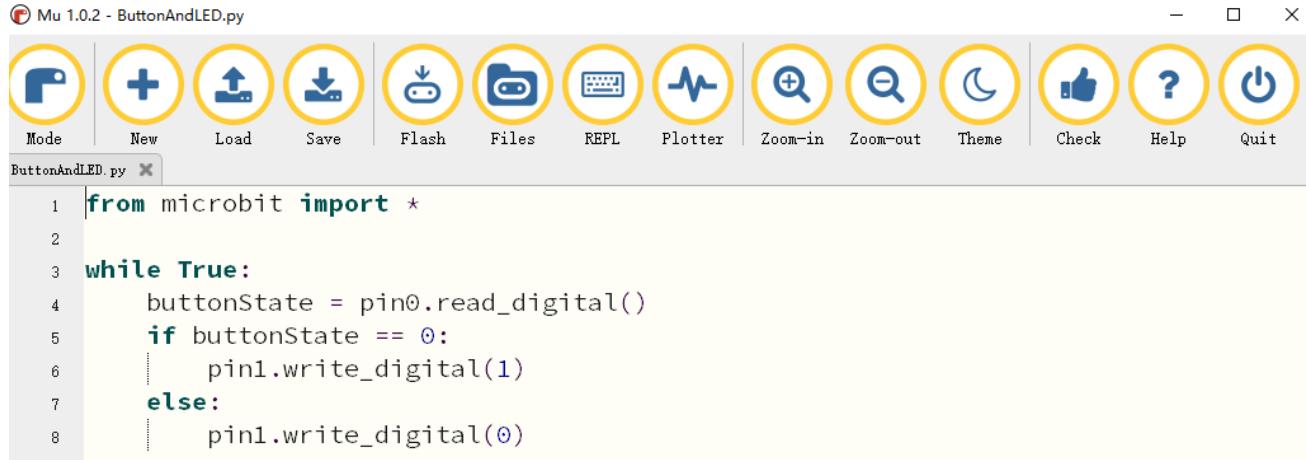
Block	Function
<code>digital read pin P0</code>	Read a digital (0 or 1) signal from a pin on the micro:bit board.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	./Projects/PythonCode/04.1_ButtonAndLED	ButtonAndLED.py

After load successfully, the code is shown as below:



The screenshot shows the Mu 1.0.2 IDE interface. The title bar says "Mu 1.0.2 - ButtonAndLED.py". The toolbar icons include Mode, New, Load, Save, Flash, Files, REPL, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The code editor window contains the following Python code:

```

1 from microbit import *
2
3 while True:
4     buttonState = pin0.read_digital()
5     if buttonState == 0:
6         pin1.write_digital(1)
7     else:
8         pin1.write_digital(0)

```

Download the code into micro:bit. When the button is pressed, the led will be turned on. When the button is released, the led will be turned off.

([How to download?](#))

The following is the program code:

```

1 from microbit import *
2 while True:
3     buttonState = pin0.read_digital()
4     if buttonState == 0:
5         pin1.write_digital(1)
6     else:
7         pin1.write_digital(0)

```

In the program, read the level of the P0 pin, save the read level value in the variable buttonState, and then determine whether the button is pressed.

```
buttonState = pin0.read_digital()
```

If the read P0 pin is low, it indicates that the button is pressed, and then make P1 pin output 1, and then LED will be turned on. Otherwise, the P1 pin outputs 0, and the LED will be turned off.

```
if buttonState == 0:  
    pin1.write_digital(1)  
else:  
    pin1.write_digital(0)
```

Reference

`pin.read_digital()`

Return 1 if the pin is high, and 0 if it's low.

For more information, please refer to:<https://microbit-micropython.readthedocs.io/en/latest/pin.html>

Project 4.2 Table Lamp

In this project, we will make a table lamp. The components and circuits used are exactly the same as the previous one, but the functions are different: press the button once, then turn on the LED; press again, turn off the LED. That is press the button once to change the LED status once.

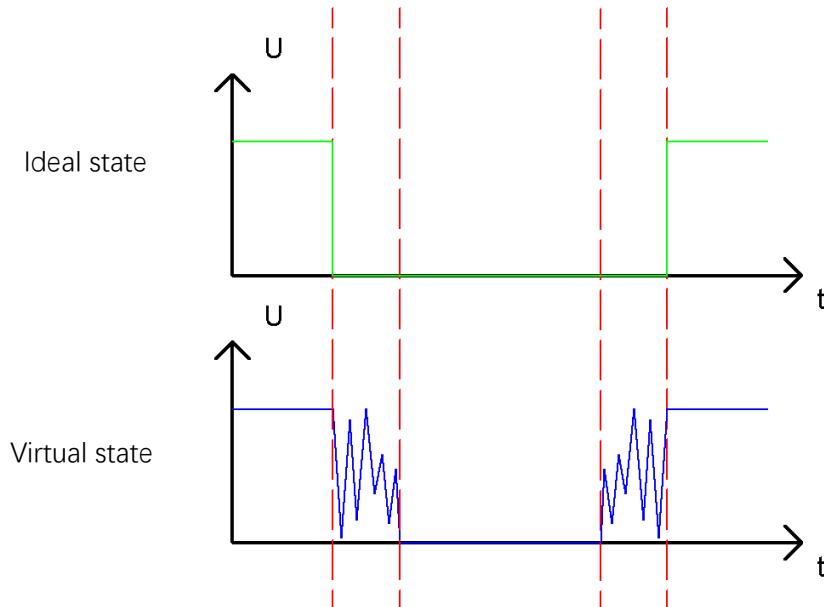
Component list

It is same with last project.

Circuit knowledge

Debounce for Push Button

When a Push Button is pressed, it will not change from one state to another state immediately. Due to mechanical vibration, there will be a continuous buffeting before it becomes another state. And the releasing-situation is similar with that process.



Therefore, if we directly detect the state of Push Button, there may be multiple pressing and releasing action in one pressing process. The buffeting will mislead the high-speed operation of the microcontroller to cause many false judgments. So we need to eliminate the impact of buffeting. Our solution is to judge the state of the button several times. Only when the button state is stable after a period, can it indicate that the button is pressed down.

This project needs the same components and circuits with the last section.

Circuit

It is same with last project.

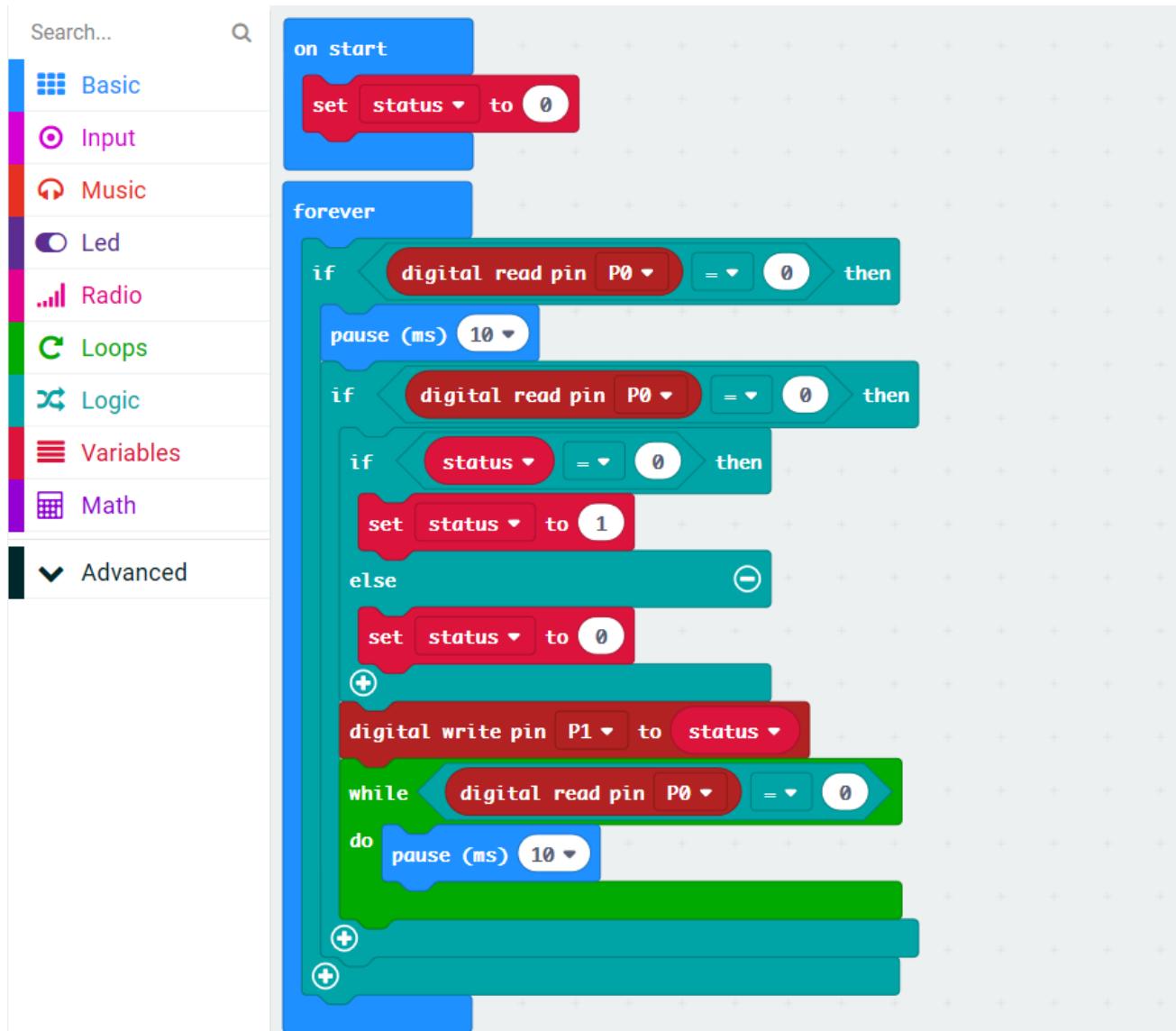
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file	./Projects/BlockCode/04.2_TableLamp	TableLamp.hex

After import successfully, the code is shown as below:

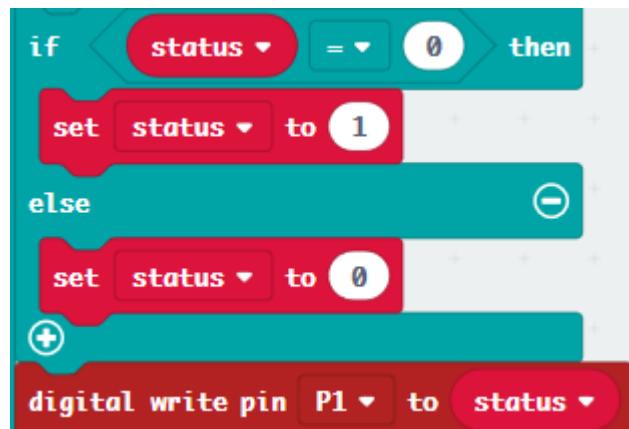


Download the code to micro:bit, press the button once, the led light is on, press the button again, the led light goes out.

In the program, when it is detected for the first time that the button is pressed, wait for 10ms to detect whether the button is pressed again, to skip the bounce when the button is pressed. And if the button is still pressed for the second time, the button is considered to have been pressed and is in a steady state. Otherwise, it is considered to be a bounce and exit this judgment.



When it is determined that the key is pressed, change the status value. Status is used to save the state of LED. And then write the new value of status to the P1 port to control the LED.



After content above is completed, the button release will wait to be detected, and the bounce of the button needs to be eliminated.



Reference

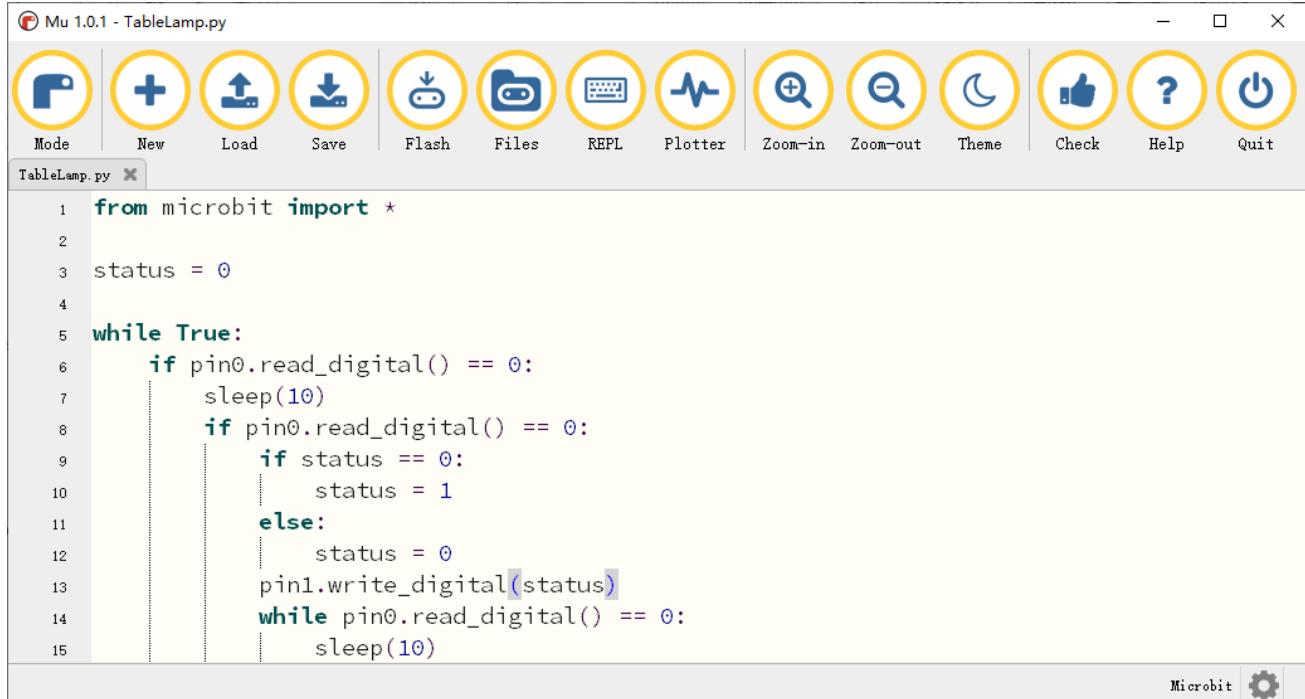
Block	Function
	Use an equals sign to make a variable store the number or string you say.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file/Projects/PythonCode/04.2_TableLamp	TableLamp.py

After load successfully, the code is shown as below:



The screenshot shows the Mu 1.0.1 IDE interface. The title bar says "Mu 1.0.1 - TableLamp.py". The toolbar has icons for Mode, New, Load, Save, Flash, Files, REPL, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The main code editor window contains the following Python code:

```

1 from microbit import *
2
3 status = 0
4
5 while True:
6     if pin0.read_digital() == 0:
7         sleep(10)
8         if pin0.read_digital() == 0:
9             if status == 0:
10                 status = 1
11             else:
12                 status = 0
13             pin1.write_digital(status)
14             while pin0.read_digital() == 0:
15                 sleep(10)

```

The code uses a button on Pin 0 to toggle the state of Pin 1, which controls an LED. The status variable is used to keep track of the previous state to avoid double-toggling.

Download the code to micro:bit, press the button once, the led light is on, press the button again, the led light goes out.

The following is the program code:

```

1 from microbit import *
2
3 status = 0
4
5 while True:
6     if pin0.read_digital() == 0:
7         sleep(10)
8         if pin0.read_digital() == 0:
9             if status == 0:
10                 status = 1
11             else:
12                 status = 0
13             pin1.write_digital(status)
14             while pin0.read_digital() == 0:
15                 sleep(10)

```

In the program, when it is detected for the first time that the button is pressed, wait for 10ms to detect whether the button is pressed again, to skip the bounce when the button is pressed. And if the button is still pressed for the second time, the button is considered to have been pressed and is in a steady state. Otherwise, it is considered to be a bounce and exit this judgment.

```
if pin0.read_digital() == 0:  
    sleep(10)  
    if pin0.read_digital() == 0:
```

When it is determined that the key is pressed, change the status value. Status is used to save the state of LED. And then write the new value of status to the P1 port to control the LED.

```
if status == 0:  
    status = 1  
else:  
    status = 0  
pin1.write_digital(status)
```

After content above is completed, the button release will wait to be detected, and the bounce of the button needs to be eliminated.

```
while pin0.read_digital() == 0:  
    sleep(10)
```

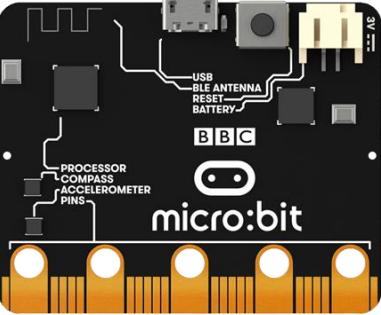
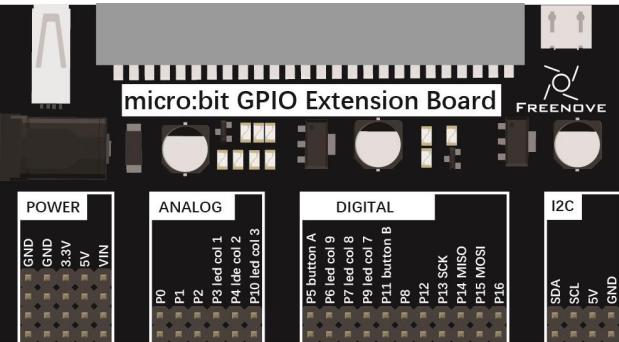
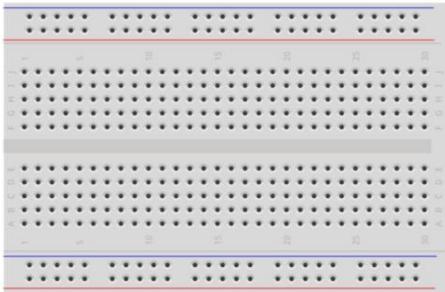
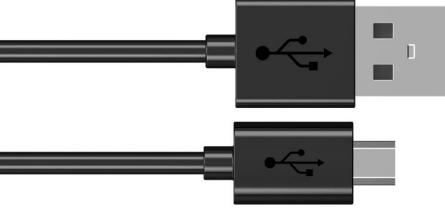
Chapter 5 LED Bar Graph

We have learned how to control LED blink. Next step, we will learn a new component LED bar graph.

Project 5.1 Flowing Light

This project uses LED bar graph to make flowing water light.

Component list

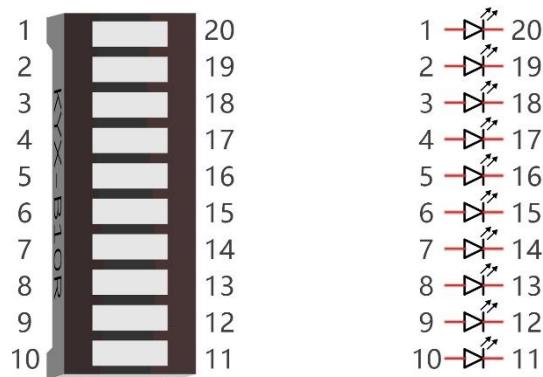
Microbit x1	Expansion board x1	
		
Breakboard x1	Resistor 220Ω x10	Jumper F/M x11
		
LED bar graph x1	USB cable x1	
		

Component knowledge

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

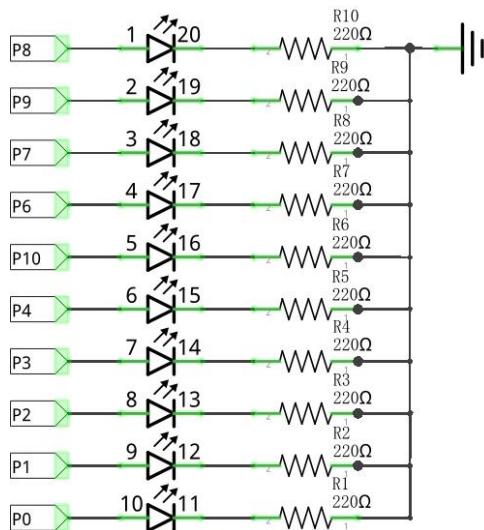
LED bar graph

LED bar graph is a component Integration consist of 10 LEDs. There are two rows of pins at its bottom.



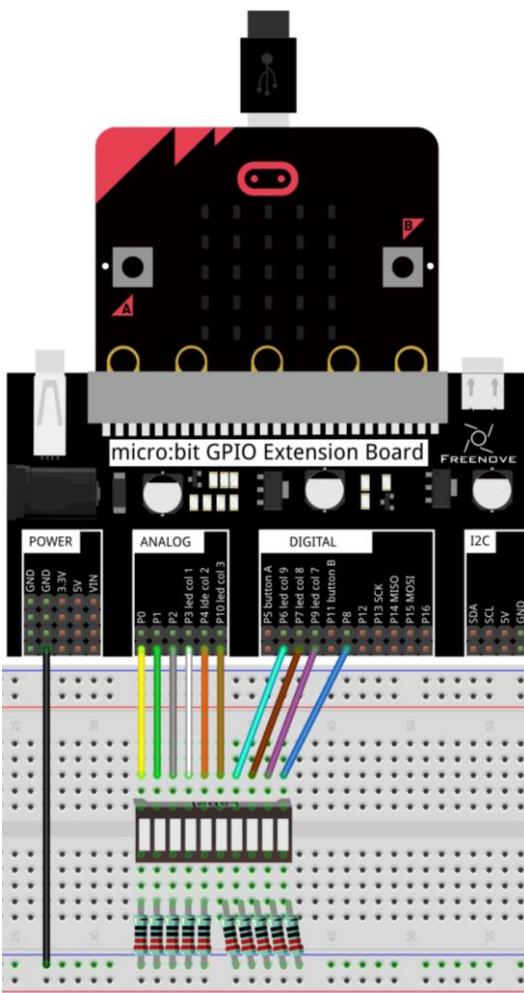
Circuit

Schematic diagram



Hardware connection

If your project doesn't work, please rotate LED bar 180°.



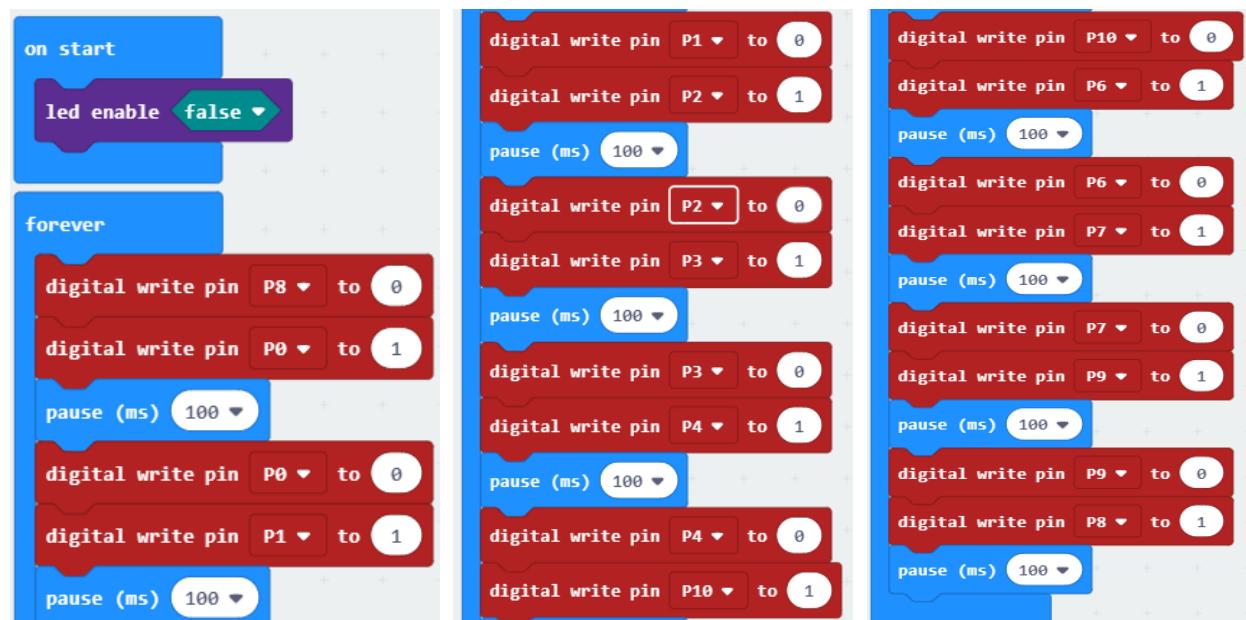
Block code

Open makecode first. Import the .hex file. The path is as below:

[\(How to import project\)](#)

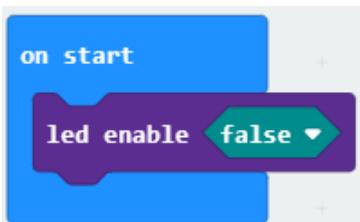
File type	Path	File name
HEX file	../Projects/BlockCode/05.1_FlowingLight01	FlowingLight01.hex

After import successfully, the code is shown as below:

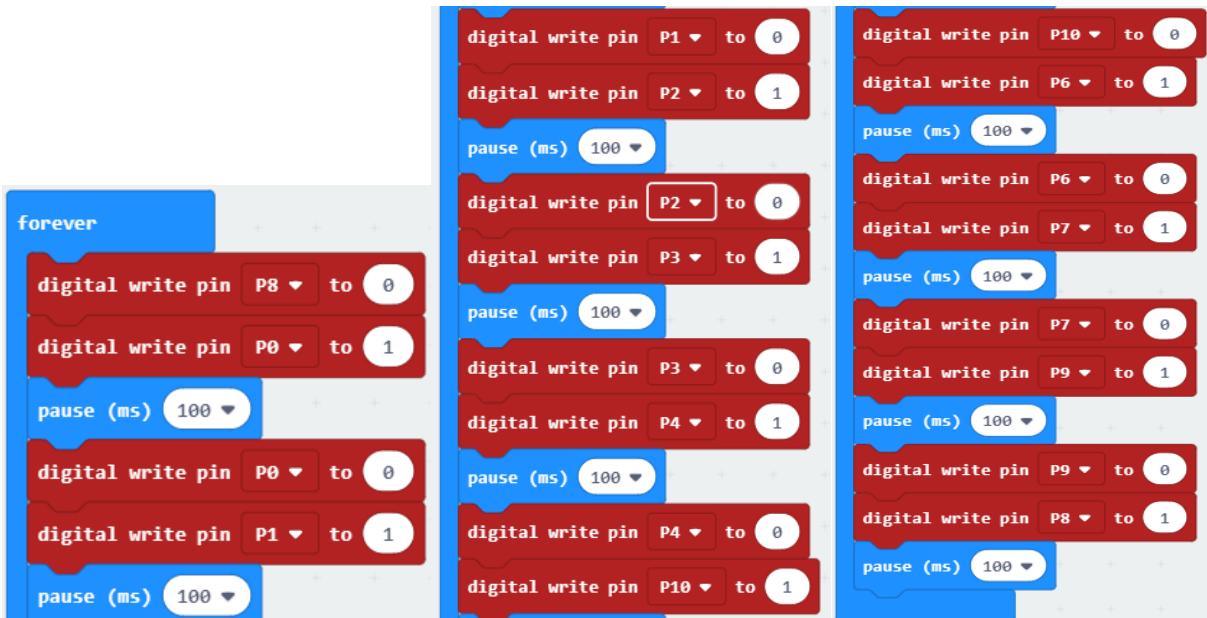


Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into micro:bit, after the program is executed, you will see the LED from left to right cycle.

In the code, we need turn off the LED screen (which allows the GPIO pin associated with the LED screen to be reused for other purposes).



When set one pin to high once, set the remaining 7 pins to low. Set all pin one by one to high in turn.



Reference

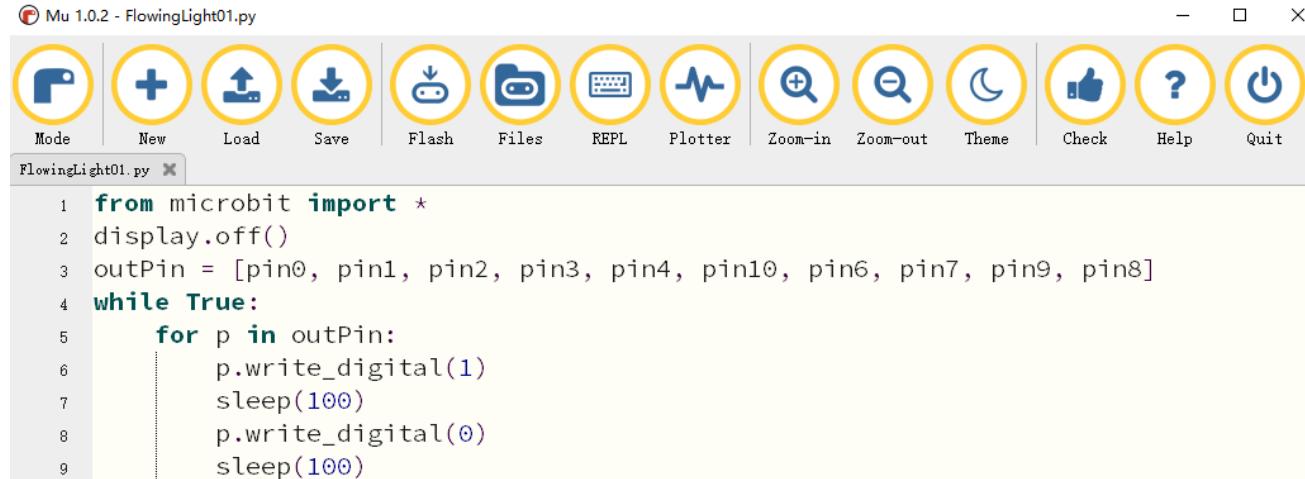
Block	Function
led enable set to false	Turns the LED screen on and off (thus allowing you to re-use the GPIO pins associated with the display for other purposes).

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/05.1_FlowingLight01	FlowingLight01.py

After load successfully, the code is shown as below:



The screenshot shows the Mu 1.0.2 IDE interface. The title bar says "Mu 1.0.2 - FlowingLight01.py". The toolbar icons include Mode, New, Load, Save, Flash, Files, REPL, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The code editor contains the following Python code:

```
1 from microbit import *
2 display.off()
3 outPin = [pin0, pin1, pin2, pin3, pin4, pin10, pin6, pin7, pin9, pin8]
4 while True:
5     for p in outPin:
6         p.write_digital(1)
7         sleep(100)
8         p.write_digital(0)
9         sleep(100)
```

Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into micro:bit, after the program is executed, you will see the LED from left to right cycle.

The following is the program code:

```
1 from microbit import *
2 display.off()
3 outPin = [pin0, pin1, pin2, pin3, pin4, pin10, pin6, pin7, pin9, pin8]
4 while True:
5     for p in outPin:
6         p.write_digital(1)
7         sleep(100)
8         p.write_digital(0)
9         sleep(100)
```

In the code, we need turn off the LED screen (which allows the GPIO pin associated with the LED screen to be reused for other purposes).

```
display.off()
```

Define an array to save the pin variable.

```
outPin = [pin0, pin1, pin2, pin3, pin4, pin6, pin7, pin8, pin9, pin10]
```

Change the level of the currently selected pin every 100ms to implement the flowing water light.

```
for p in outPin:
    p.write_digital(1)
    sleep(100)
    p.write_digital(0)
    sleep(100)
```

Reference

display.off()

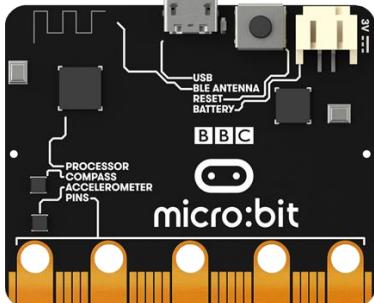
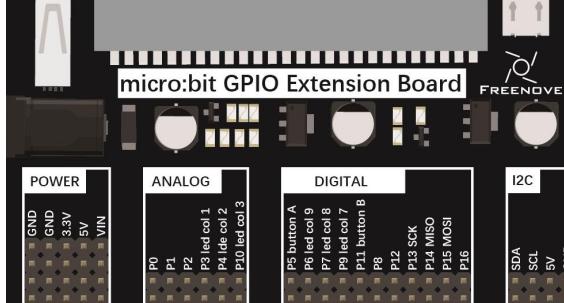
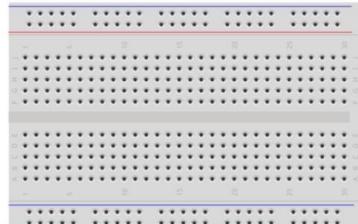
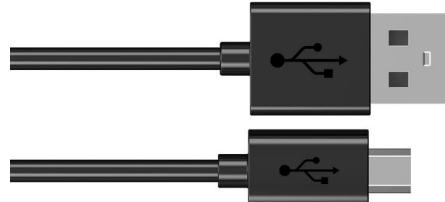
Use off() to turn off the display (thus allowing you to re-use the GPIO pins associated with the display for other purposes)

Chapter 6 PWM

In this chapter, we will learn how to make a breathing LED.

Project 6.1 Breathing Light

Component list

Microbit x1	Expansion board x1	
		
Breakboard x1	USB cable x1	
		
Jumper F/M x2	Resistor 220Ω x1	LED x1
		

Circuit knowledge

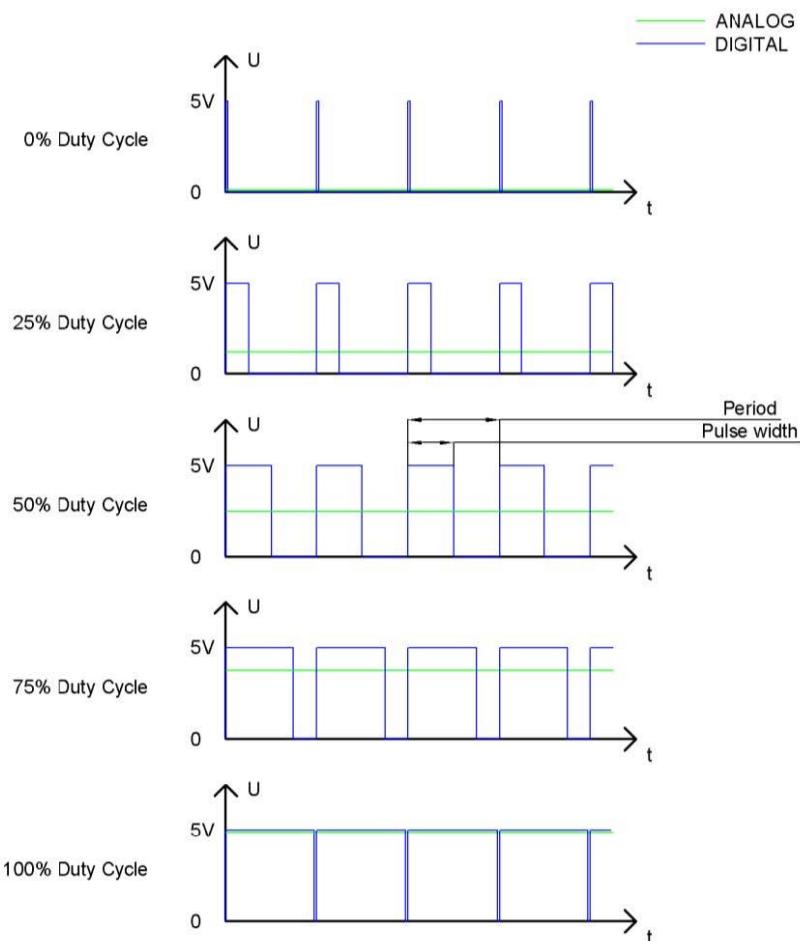
At first, let us learn the knowledge how to use the circuit to make LED emit different brightness of light,

PWM

PWM, namely Width Modulation Pulse, is a very effective technique for using digital signals to control analog circuits. The common processors can not directly output analog signals. PWM technology make it very convenient to achieve this purpose.

PWM technology uses digital pins to send certain frequency of square waves, that is, the output of high level and low level last for a period alternately. The total time for each set of high level and low level is generally fixed, which is called period (the reciprocal of the period is frequency). Output high time is generally called pulse width, and the percentage of pulse width is called duty cycle.

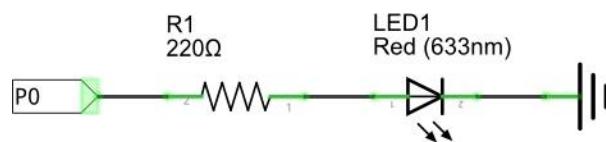
The longer the output of high level last, the larger the duty cycle and the larger the corresponding voltage in analog signal. The following figures show how the analogs signal voltage vary between 0V-5V corresponding to the pulse width 0%-100%:



The larger PWM duty ratio, the larger the output power. So we can use PWM to control the brightness of LED, and the speed of DC motor and so on.

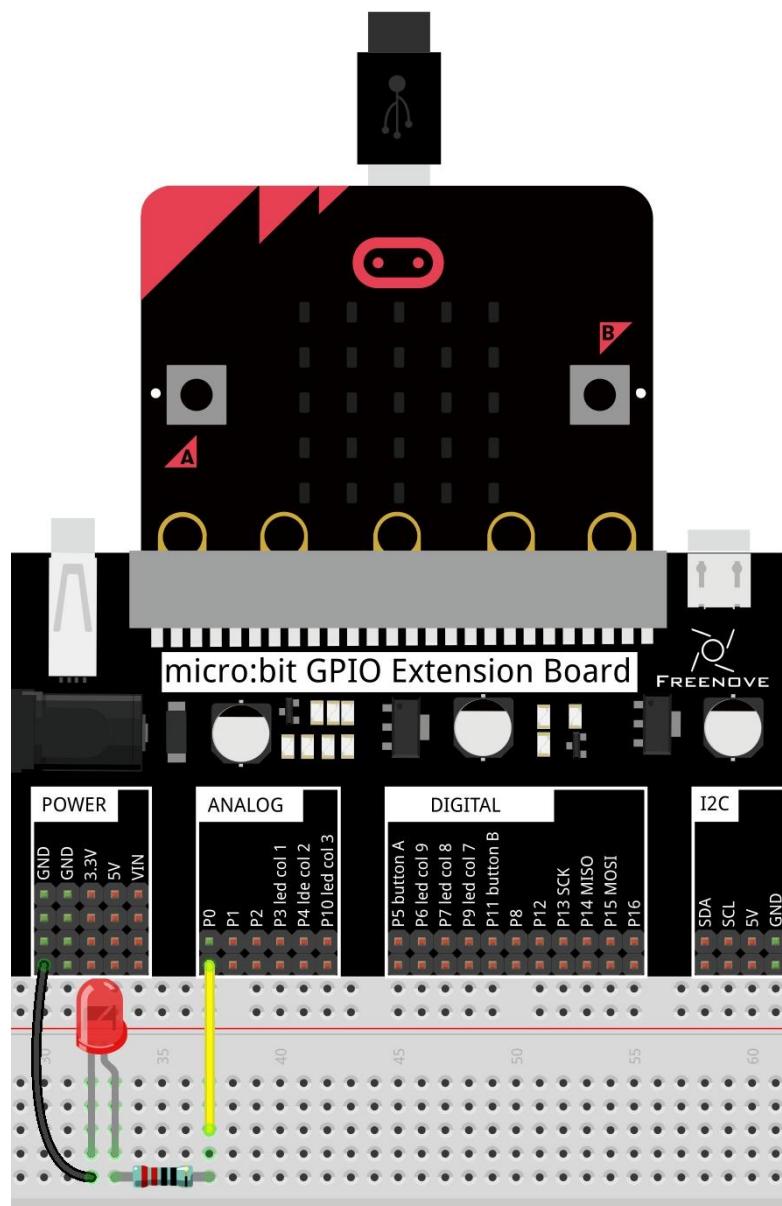
Circuit

Schematic Diagram



hardware connection

The pin for the circuit is P0. The long pin (positive) of LED is connected to the resistor, and the short pin (negative) to ground.



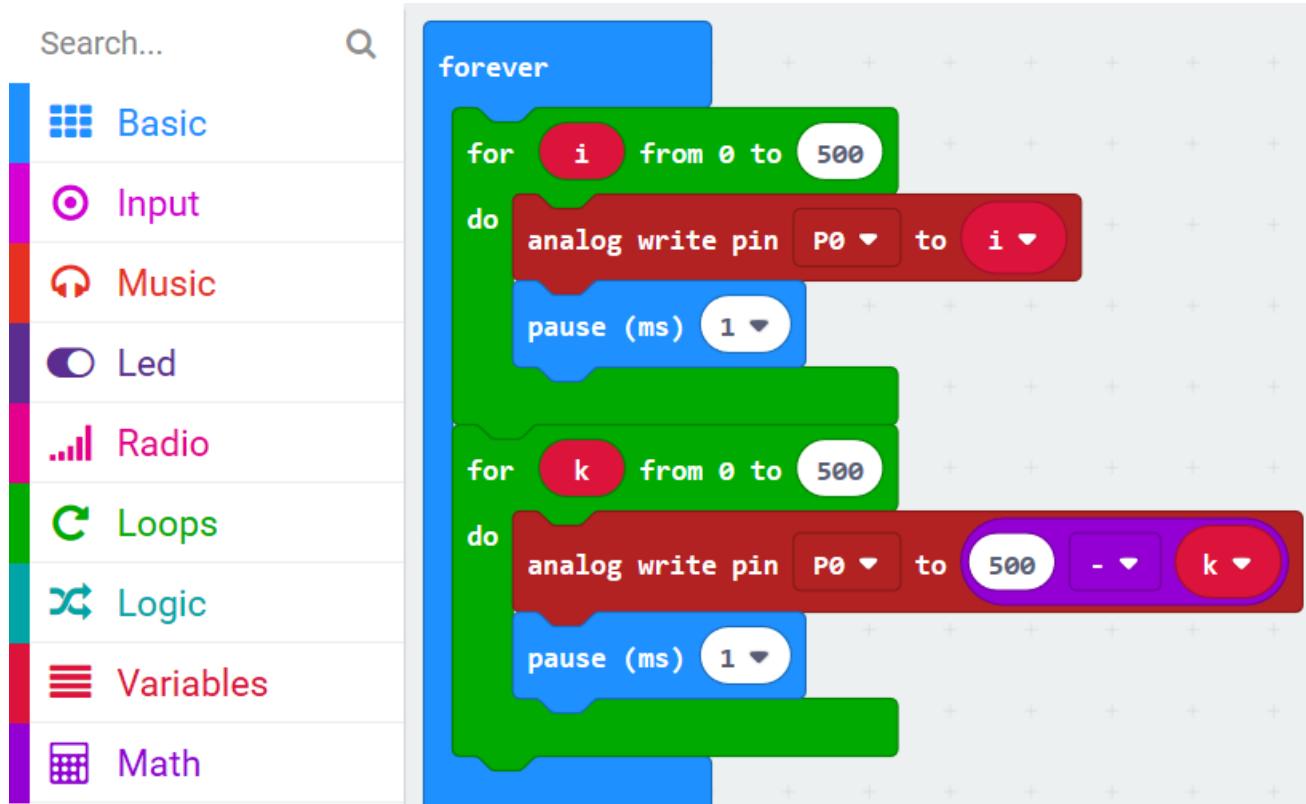
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

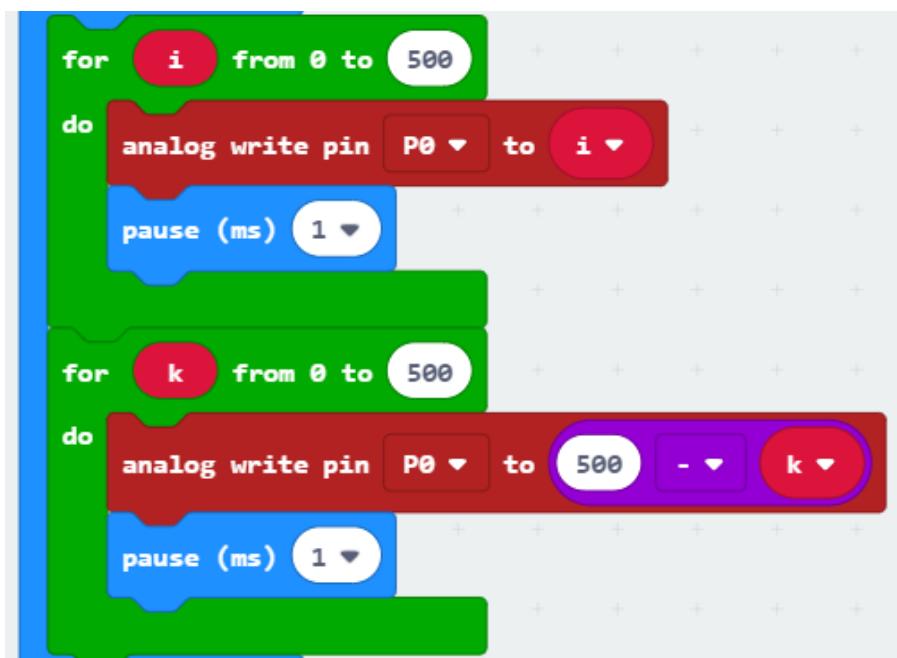
File type	Path	File name
HEX file	../Projects/BlockCode/06.1_BreathingLight	BreathingLight.hex

After import successfully, the code is shown as below:



Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into micro:bit. The LED brightness will increase gradually. Then the brightness will decrease gradually. This process will be continuously cycled.

P0 out put PWM signal, from 0 to 500, then from 500 to 0.



Reference

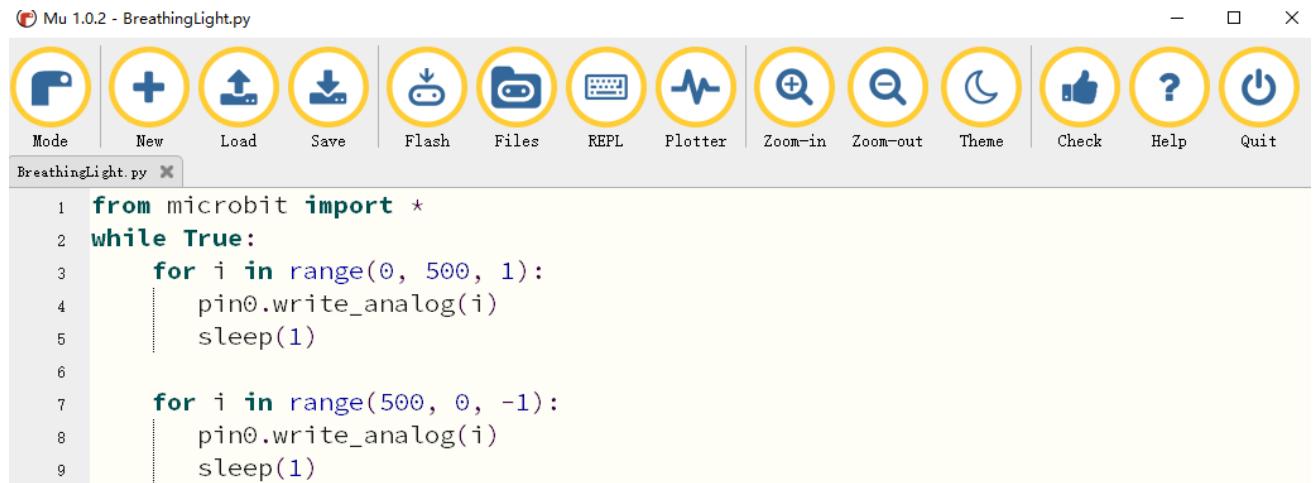
Block	Function
analog write pin P0 to i	Write an analog signal (0 through 1023) to the pin you say.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/06.1_BreathingLight	BreathingLight.py

After load successfully, the code is shown as below:



The screenshot shows the Mu 1.0.2 IDE interface. The title bar says "Mu 1.0.2 - BreathingLight.py". The menu bar has "File", "Edit", "Run", "Terminal", "Help", and "About". The toolbar includes icons for Mode, New, Load, Save, Flash, Files, REPL, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The code editor contains the following Python code:

```
1 from microbit import *
2 while True:
3     for i in range(0, 500, 1):
4         pin0.write_analog(i)
5         sleep(1)
6
7     for i in range(500, 0, -1):
8         pin0.write_analog(i)
9         sleep(1)
```

Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into micro:bit. The LED brightness will increase gradually. Then the brightness will decrease gradually. This process will be continuously cycled. ([How to download?](#))

The following is the program code:

```
1 from microbit import *
2
3 while True:
4     for i in range(0, 500, 1):
5         pin0.write_analog(i)
6         sleep(1)
7
8     for i in range(500, 0, -1):
9         pin0.write_analog(i)
10        sleep(1)
```

P0 out put PWM signal, from 0 to 500,

```
for i in range(0, 500, 1):
    pin0.write_analog(i)
    sleep(1)
```

Then from 500 to 0.

```
for i in range(500, 0, -1):
    pin0.write_analog(i)
    sleep(1)
```

Reference

`pin.write_analog(value)`

Output a PWM signal on the pin, with the duty cycle proportional to the provided value. The value may be either an integer or a floating point number between 0 (0% duty cycle) and 1023 (100% duty)..

For more information, please refer to:

<https://microbit-micropython.readthedocs.io/en/latest/pin.html>

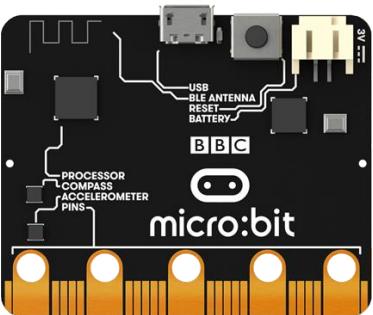
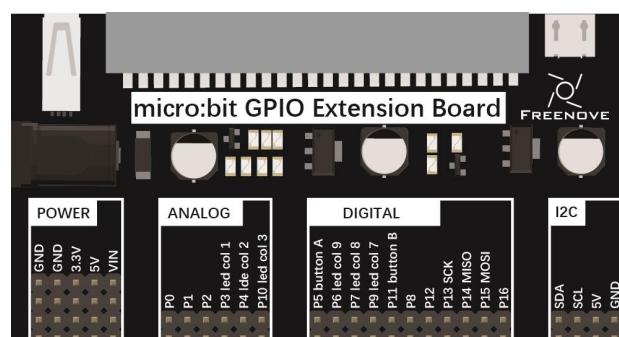
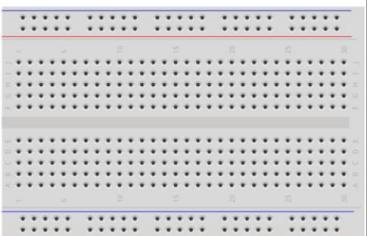
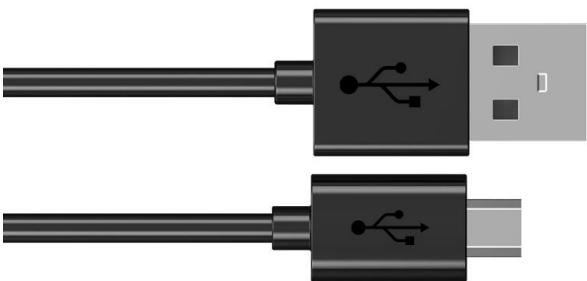
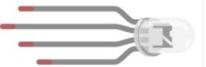
Chapter 7 RGBLED

In this chapter, we will learn a new component, RGBLED.

Project 7.1 Monochromatic Light

This project will use RGBLED to show one color.

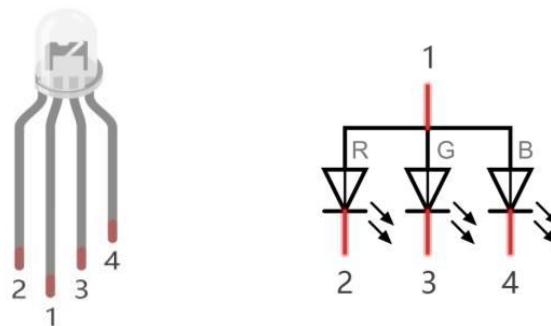
Component list

Microbit x1	Expansion board x1	
		
Breakboard x1	USB cable x1	
		
RGB_LED x1	Jumper F/M x4	Resistor 220Ω x3
		

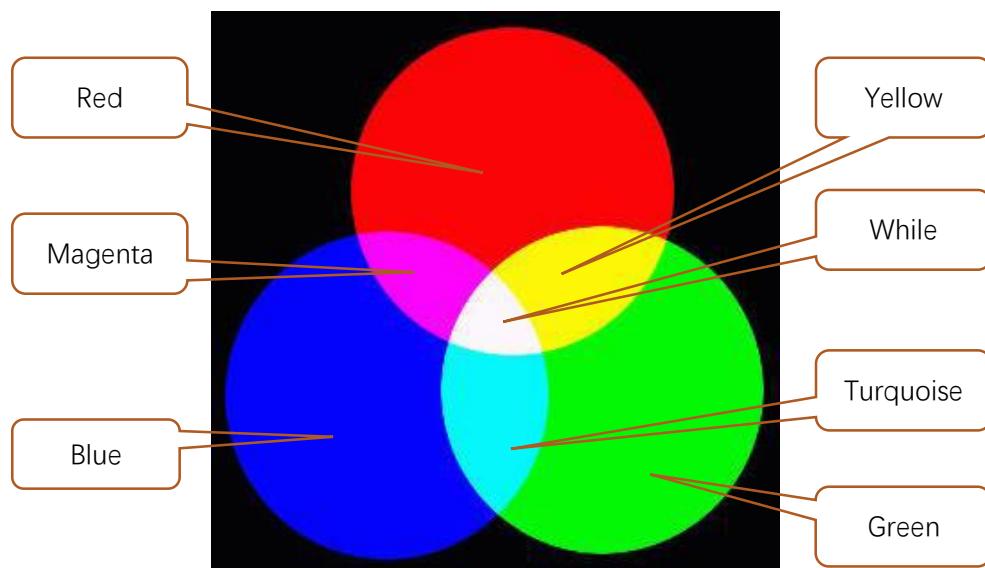
Component knowledge

RGB LED

RGB LED has integrated 3 LEDs that can respectively emit red, green and blue light, and it has 4 pins. The long pin (1) is the common port, that is, 3 LED's cathode or anode. The RGB LED with common anode and its symbol are as follows. By controlling these 3 LED to emit light with different brightness, we can make RGB LED emit various colors of light.



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

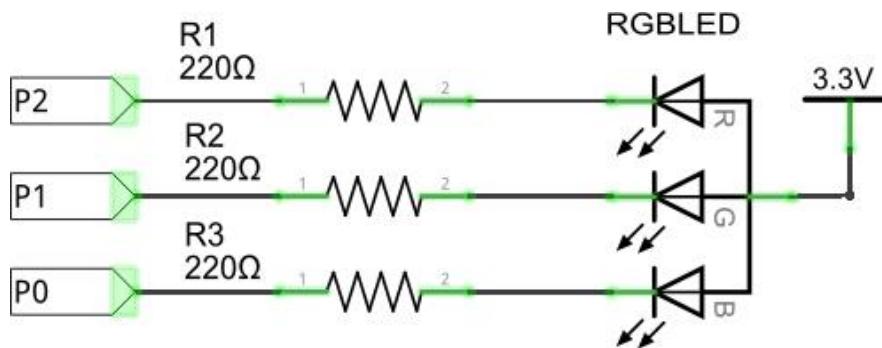


We know from previous section that, we controls LED to emit a total 256(0-255) different brightness by PWM. So, through the combination of three different colors of LED, RGB LED can emit $256^3=16777216$ Colors, 16Million colors.

Circuit

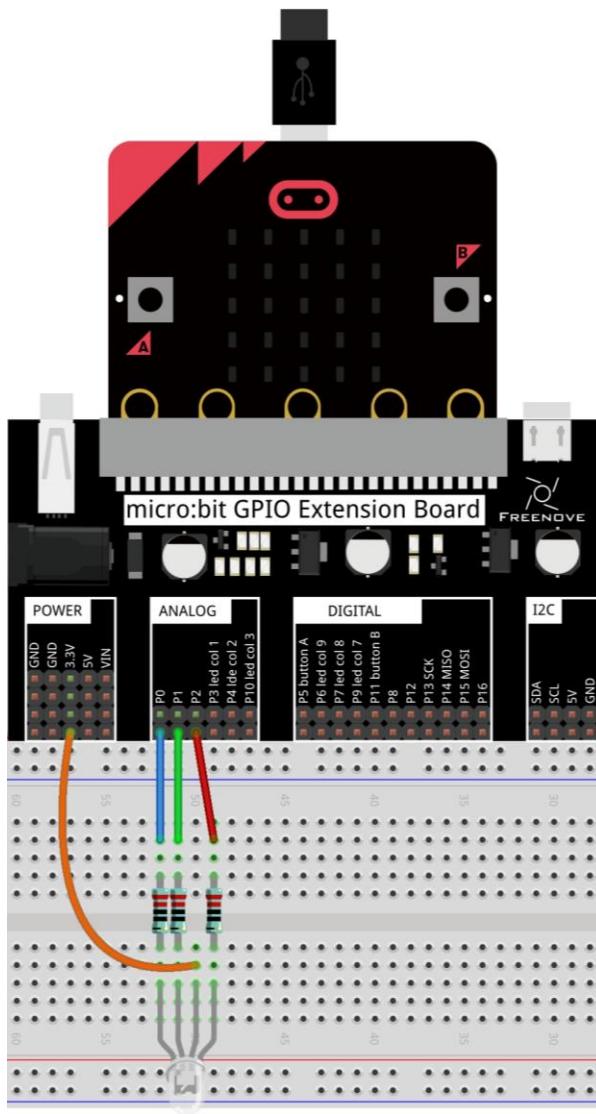
This circuit uses pins P2, P1, and P0 to connect the red, green, and blue LED negative electrodes of the RGBLED.

Schematic diagram



Hardware connection

The longest pin of the RGB LED is connected to the power supply 3.3V.

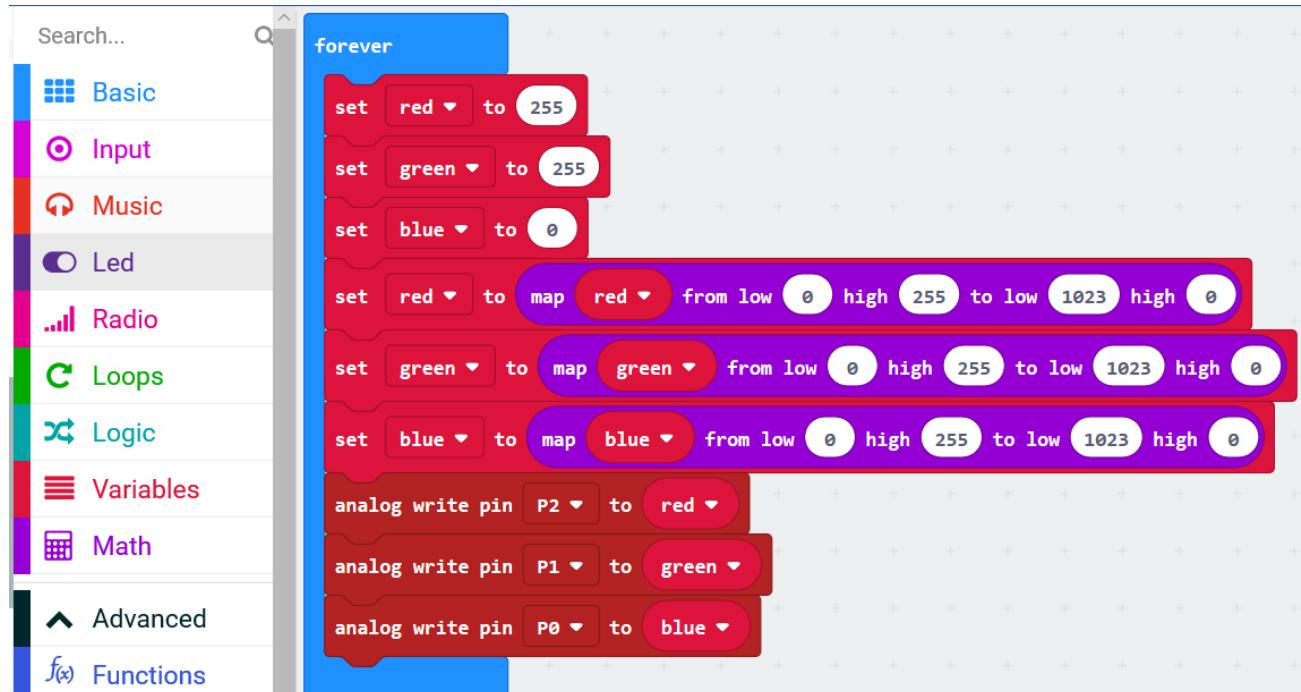


Block code

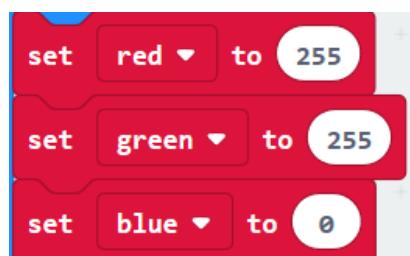
Open makecode first. Import the .hex file. The path is as below: ([how to import project](#))

File type	Path	File name
HEX file	../Projects/BlockCode/07.1_RGBLED	RGBLED.hex

After import successfully, the code is shown as below:



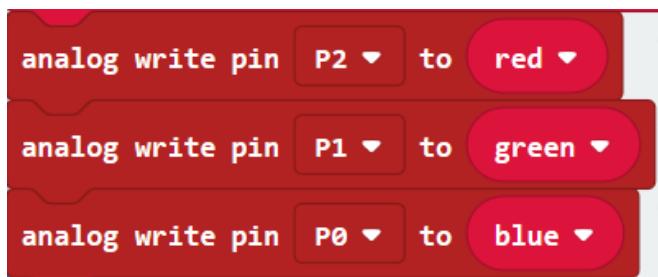
Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into micro:bit. RGBLED will show yellow color. RGB value of yellow is (255,255,0).



RGBLED is a common anode, the r g b pins need to be set LOW to get RGBLED on. And the values of the variables 'red', 'green', and 'blue' need to be converted from range of 0-255 to analog signal values range of 1023-0.



Write the 'red','green','blue' to corresponding pins P2, P1, P0.



Reference

Block	Function
	Convert a value in one number range to a value in another number range.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file/Projects/PythonCode/07.1_RGBLED	RGBLED.py

After load successfully, the code is shown as below:

```

1 from microbit import *
2 def map(value,fromLow,toHigh,toLow):
3     return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
4 red=255
5 green=255
6 blue=0
7 red=map(red,0,255,1023,0)
8 green=map(green,0,255,1023,0)
9 blue=map(blue,0,255,1023,0)
10 pin2.write_analog(red)
11 pin1.write_analog(green)
12 pin0.write_analog(blue)

```

Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into micro:bit. RGBLED will show yellow color. ([How to download?](#))

The following is the program code:

```

1 from microbit import *
2 def map(value,fromLow,toHigh,toLow):
3     return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
4 red=255
5 green=255
6 blue=0
7 red=map(red,0,255,1023,0)
8 green=map(green,0,255,1023,0)
9 blue=map(blue,0,255,1023,0)
10 pin2.write_analog(red)
11 pin1.write_analog(green)
12 pin0.write_analog(blue)

```

A custom map() function is used converts a value in one range to another range.

```

def map(value,fromLow,toHigh,toLow):
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow

```

RGB value of yellow is (255,255,0).

```
red=255  
green=255  
blue=0
```

RGBLED is a common anode, the r g b pins need to be set LOW to get it on. And the values of the variables 'red', 'green', and 'blue' need to be converted from range of 0-255 to analog signal values range of 1023-0.

```
red=map(red, 0, 255, 1023, 0)  
green=map(green, 0, 255, 1023, 0)  
blue=map(blue, 0, 255, 1023, 0)
```

Write the 'red', 'green', 'blue' to corresponding pins P2, P1, P0.

```
pin2.write_analog(red)  
pin1.write_analog(green)  
pin0.write_analog(blue)
```

Reference

map(value, fromLow, fromHigh, toLow, toHigh)

A custom function that converts a value in a range of numbers to a value in another range of numbers. For example, map(8,0,10,0,100) returns a value of 80; map(8,0,10,100,0)=20.

map(8,0,10,0,100)=80.

Project 7.2 Colorful Light

This project will use RGB LED to show different colors.

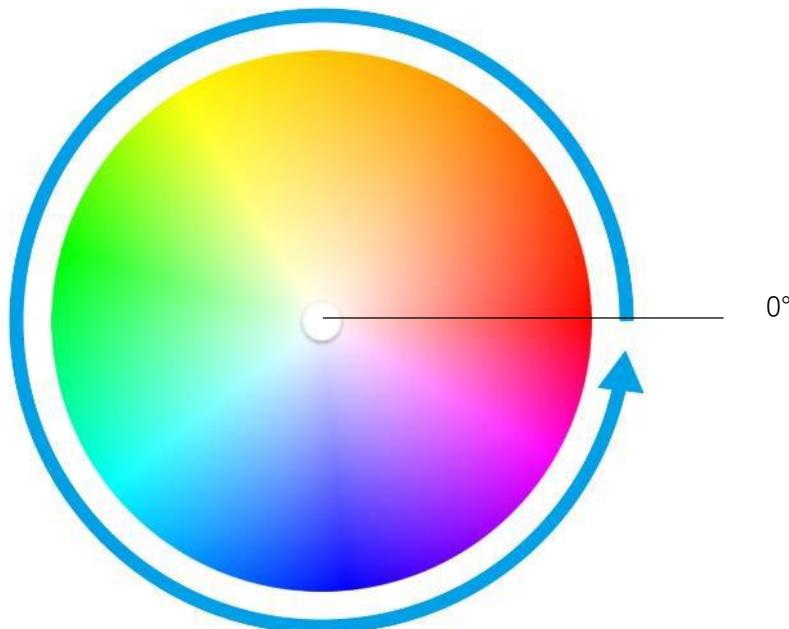
Component list

It is the same as the last project.

HSL color

The HSL color mode is a color standard in the industry. It changes the color through Hue (H), Saturation (S), and Lightness (L) to obtain different colors. This standard includes almost all colors that human vision can perceive. It is one of the most widely used color systems to date.

As shown in the hue circle below, the 0 degree of the hue is R (red) color, 120 degrees is G (green) color, and 240 degrees is B (blue) color. The different colors of the angle are different. The default saturation (S) takes the maximum value 100, the brightness (L) takes 50. If the hue angle is changed, the color will change. And the HSL color system can be converted to the RGB color system, to change the color of the LED.



Circuit

It is same with last project.

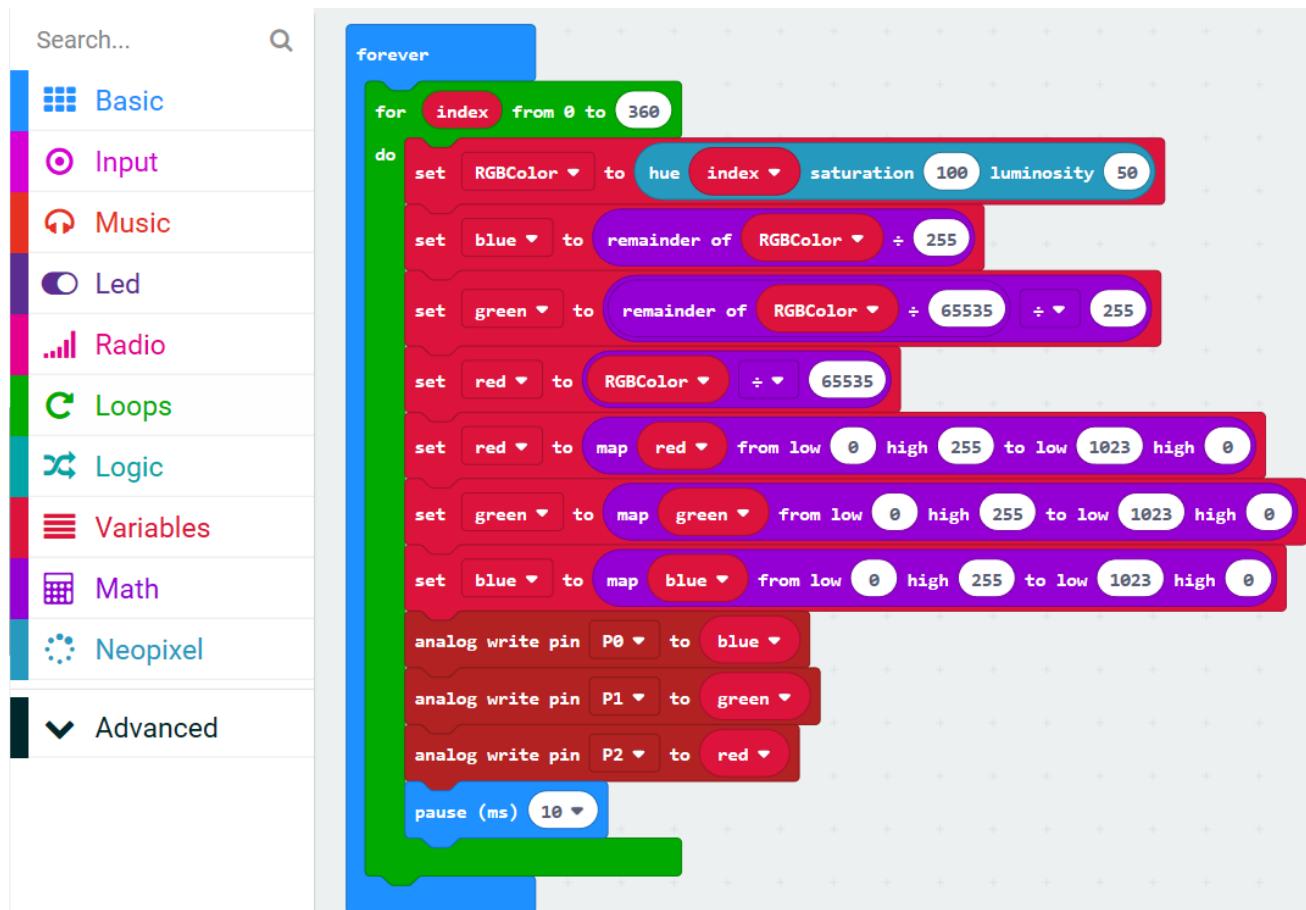
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file/Projects/BlockCode/07.2_ColorfulLight	ColorfulLight.hex

After import successfully, the code is shown as below:



Check the connection of the circuit, confirm that the circuit is connected correctly, then download the code into the microbit, RGBLED will change different colors.

From the knowledge of HSL color, we can know that different hue angles correspond different colors. The variable index represents hue angle, ranging from 0 to 360.



Converts the HSL color system to the RGB color system, returns the RGB value corresponding to the current hue angle, and stores the value in the variable RGBColor. For example: hexadecimal RGB value 0xFF0000

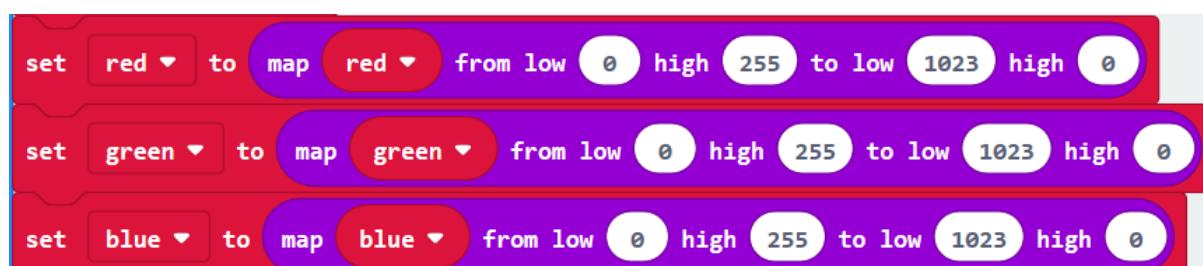
means red, FF is the value of the red channel in RGB, and 00 and 00 are the values of the green and blue channels, respectively.



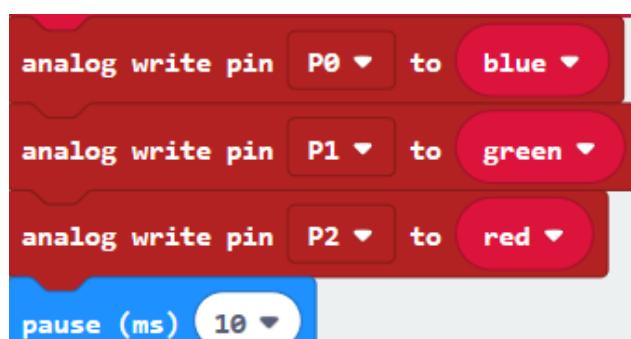
Assign the value of the lower eight-bit to blue channel, the value of the middle eight-bit to green channel, and the value of the upper eight-bit to red channel.



RGBLED is a common anode, the r g b pins need to be set LOW. And the values of the variables 'red', 'green', and 'blue' need to be converted from range of 0-255 to analog signal values range of 1023-0.



Every 10ms, write the 'red', 'green', 'blue' to corresponding pins P2, P1, P0.

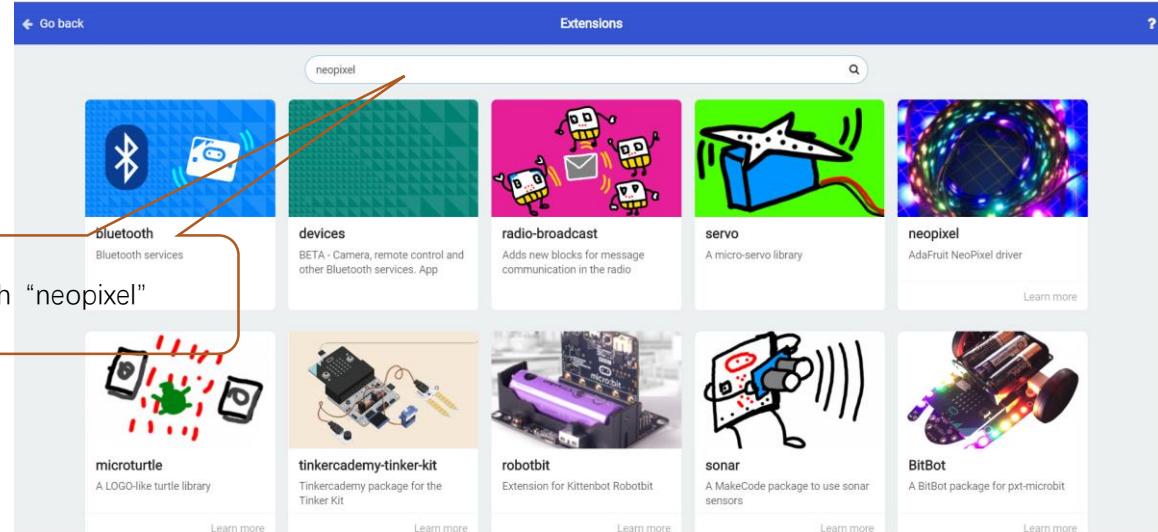
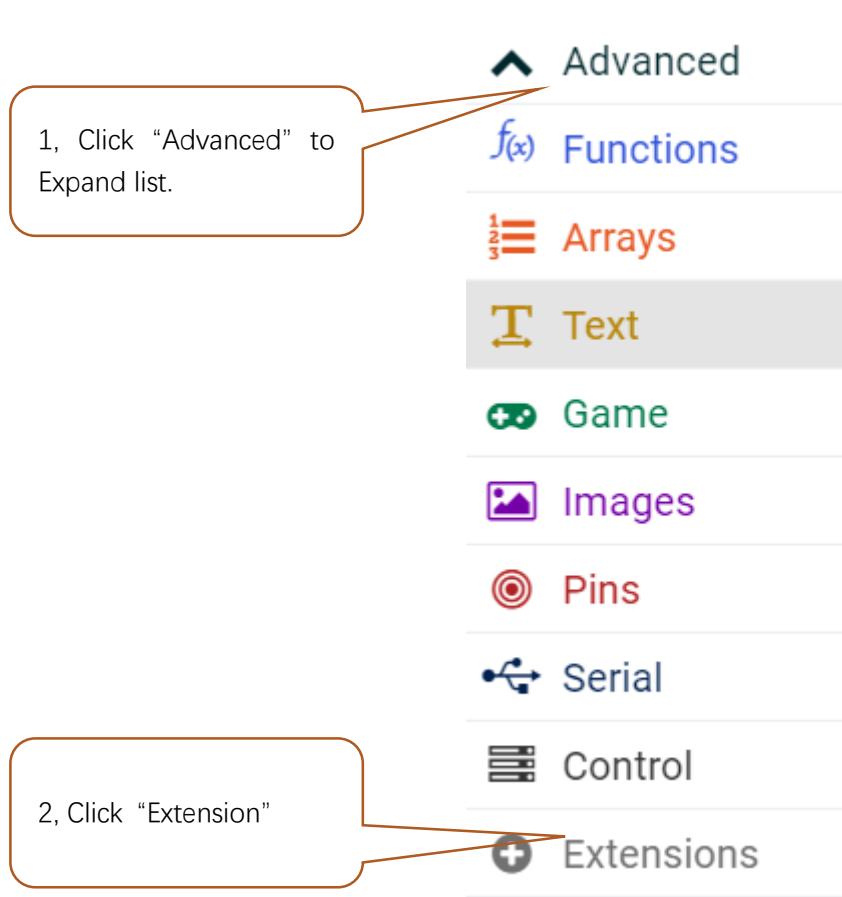


Reference

Block	Function
remainder of $\theta \div 1$	This is an extra operator for division. You can find out how much is left over if one number doesn't divide into the other number evenly.
hue θ saturation θ luminosity θ	Convert HSL color system to RGB color system, and return the RGB value corresponding to the current hue angle. It belongs to Neopixel expansion block.

Extensions

You can import Neopixel expansion block into new project as below:



Extensions

neopixel

bitbot

ws2812b

ringbitcar

BitBot

gigglebot

ColorBit

RoboBit

imaginemaker

cubebit

Learn more

Search...

Basic

Input

Music

Led

Radio

Loops

Logic

Variables

Math

Neopixel

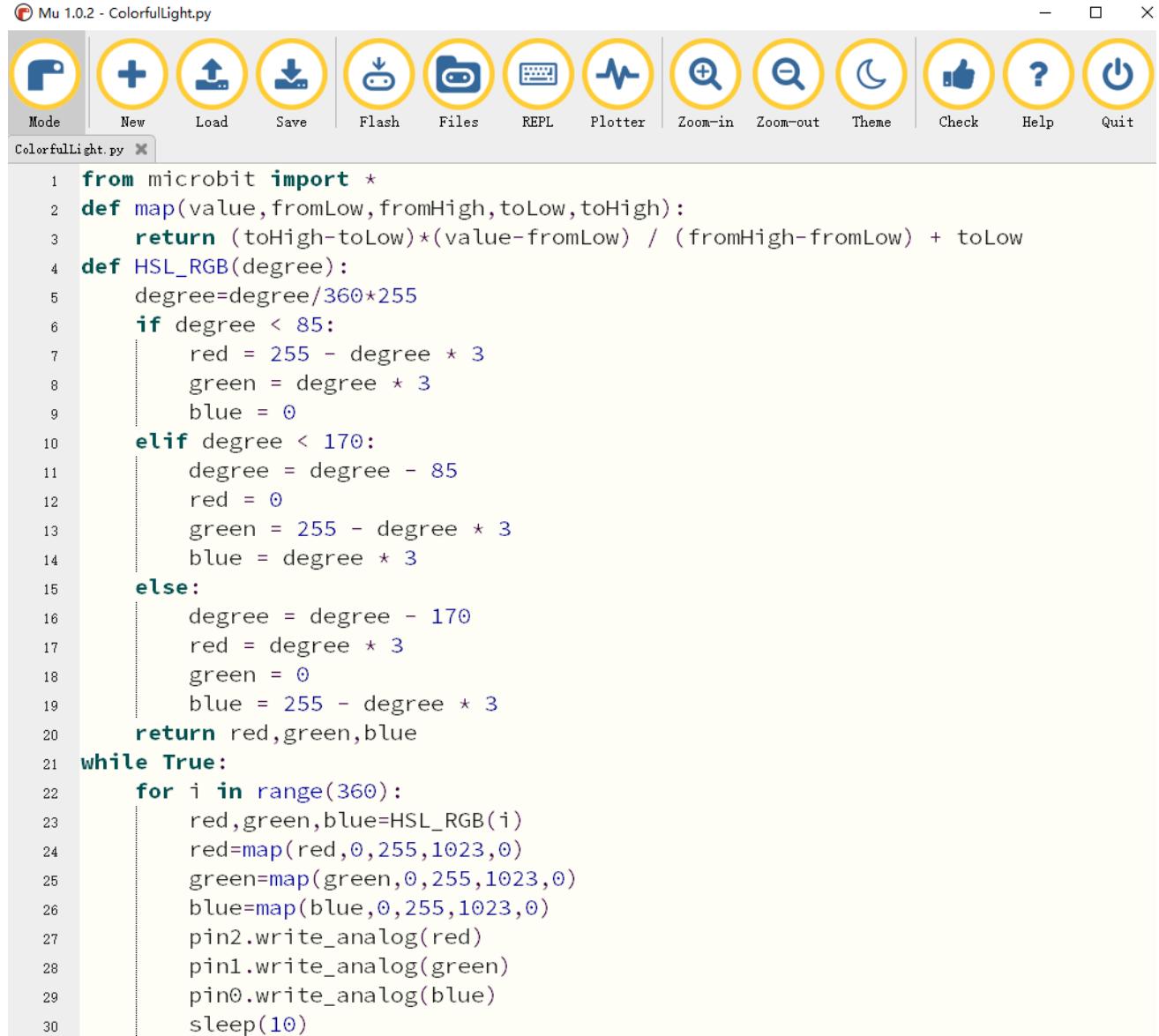
It is completed.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	./Projects/PythonCode/07.2_ColorfulLight	ColorfulLight.py

After load successfully, the code is shown as below:



```

1  from microbit import *
2  def map(value,fromLow,fromHigh,toLow,toHigh):
3      return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
4  def HSL_RGB(degree):
5      degree=degree/360*255
6      if degree < 85:
7          red = 255 - degree * 3
8          green = degree * 3
9          blue = 0
10     elif degree < 170:
11         degree = degree - 85
12         red = 0
13         green = 255 - degree * 3
14         blue = degree * 3
15     else:
16         degree = degree - 170
17         red = degree * 3
18         green = 0
19         blue = 255 - degree * 3
20     return red,green,blue
21 while True:
22     for i in range(360):
23         red,green,blue=HSL_RGB(i)
24         red=map(red,0,255,1023,0)
25         green=map(green,0,255,1023,0)
26         blue=map(blue,0,255,1023,0)
27         pin2.write_analog(red)
28         pin1.write_analog(green)
29         pin0.write_analog(blue)
30         sleep(10)

```

Check the connection of the circuit, confirm that the circuit is connected correctly, then download the code into the microbit, RGBLED will change different colors. ([How to download?](#))

The following is the program code:

```
1  from microbit import *
2  def map(value, fromLow, fromHigh, toLow, toHigh):
3      return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
4  def HSL_RGB(degree):
5      degree=degree/360*255
6      if degree < 85:
7          red = 255 - degree * 3
8          green = degree * 3
9          blue = 0
10     elif degree < 170:
11         degree = degree - 85
12         red = 0
13         green = 255 - degree * 3
14         blue = degree * 3
15     else:
16         degree = degree - 170
17         red = degree * 3
18         green = 0
19         blue = 255 - degree * 3
20     return red,green,blue
21 while True:
22     for i in range(360):
23         red,green,blue=HSL_RGB(i)
24         red=map(red, 0, 255, 1023, 0)
25         green=map(green, 0, 255, 1023, 0)
26         blue=map(blue, 0, 255, 1023, 0)
27         pin2.write_analog(red)
28         pin1.write_analog(green)
29         pin0.write_analog(blue)
30         sleep(10)
```

The map() function is used converts a value in one range to another range.

```
def map(value, fromLow, fromHigh, toLow, toHigh):
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
```

The HSL_RGB() function is used to convert the HSL color system to RGB color, and return the RGB value corresponding to the current hue angle.

```
def HSL_RGB (degree):
    degree=degree/360*255
    if degree < 85:
        red = 255 - degree * 3
        green = degree * 3
        blue = 0
```

```
    elif degree < 170:  
        degree = degree - 85  
        red = 0  
        green = 255 - degree * 3  
        blue = degree * 3  
    else:  
        degree = degree - 170  
        red = degree * 3  
        green = 0  
        blue = 255 - degree * 3  
    return red, green, blue
```

Cycle 360 times, display the hue angle color corresponding to 0 to 360 degrees, and replace it every 10ms.

```
while True:  
    for i in range(360):  
        red, green, blue=HSL_RGB(i)  
        red=map(red, 0, 255, 1023, 0)  
        green=map(green, 0, 255, 1023, 0)  
        blue=map(blue, 0, 255, 1023, 0)  
        pin2.write_analog(red)  
        pin1.write_analog(green)  
        pin0.write_analog(blue)  
        sleep(0.01)
```

Reference

HSL_RGB(degree)

Custom function, used to convert HSL color system to RGB color system, return the RGB value corresponding to the current hue angle, for example: HSL_RGB(0), return red RGB value: red=255, green=0, blue=0.

Chapter 8 Neopixel

In this chapter, we will learn Freenove 8 RGB LED Module

Project 8.1 Rainbow Water Light

This project will achieve rainbow water light.

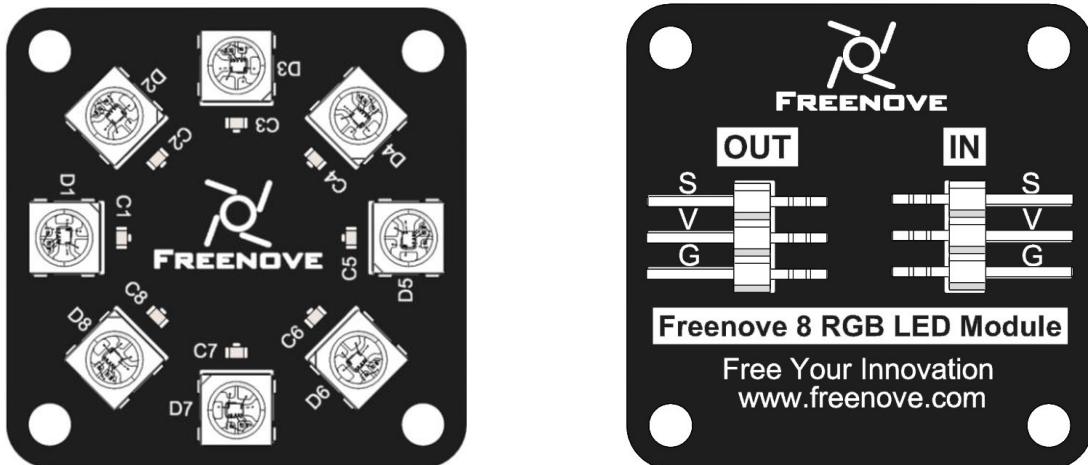
Component list

Microbit x1	Expansion board x1
Jumper F/F x3	USB cable x2
Freenove 8 RGB LED Module x1	

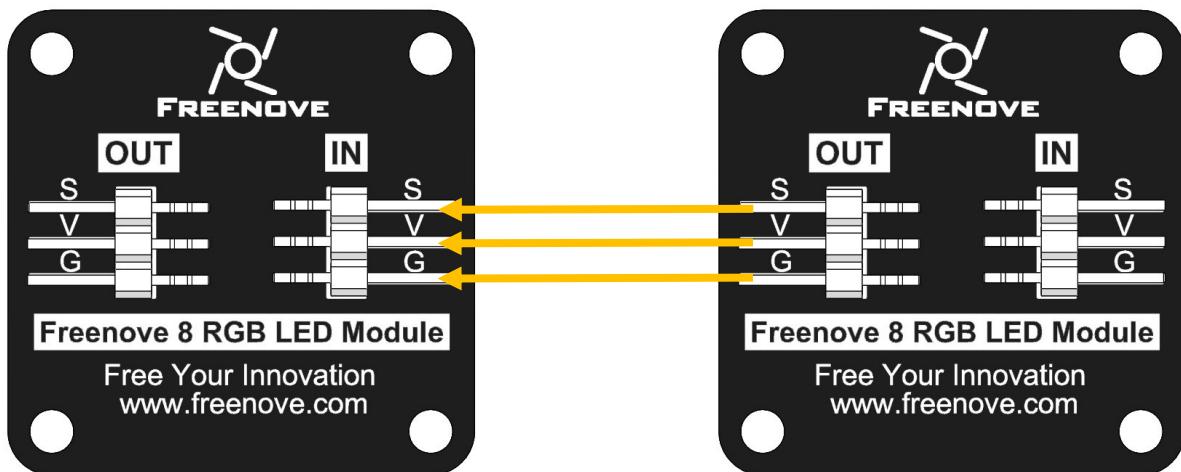
Component knowledge

Freenove 8 RGB LED Module

The Freenove 8 RGB LED Module is as below. You can use only one data pin to control the eight LED on the module. As shown below:



And you can also control many modules at the same time. Just connect OUT pin of one module to IN pin of another module. In such way, you can use one data pin to control 8, 16, 32 ... LEDs.



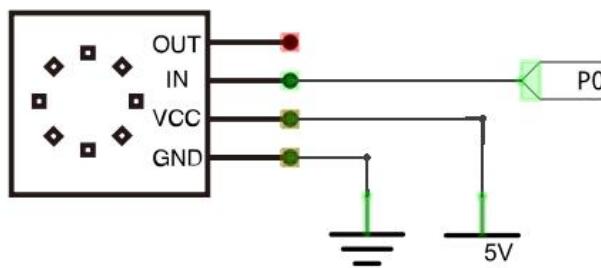
Pin description:

(IN)		(OUT)	
symbol	Function	symbol	Function
S	Input control signal	S	Output control signal
V	Power supply pin, +3.5V~5.5V	V	Power supply pin, +3.5V~5.5V
G	GND	G	GND

Circuit

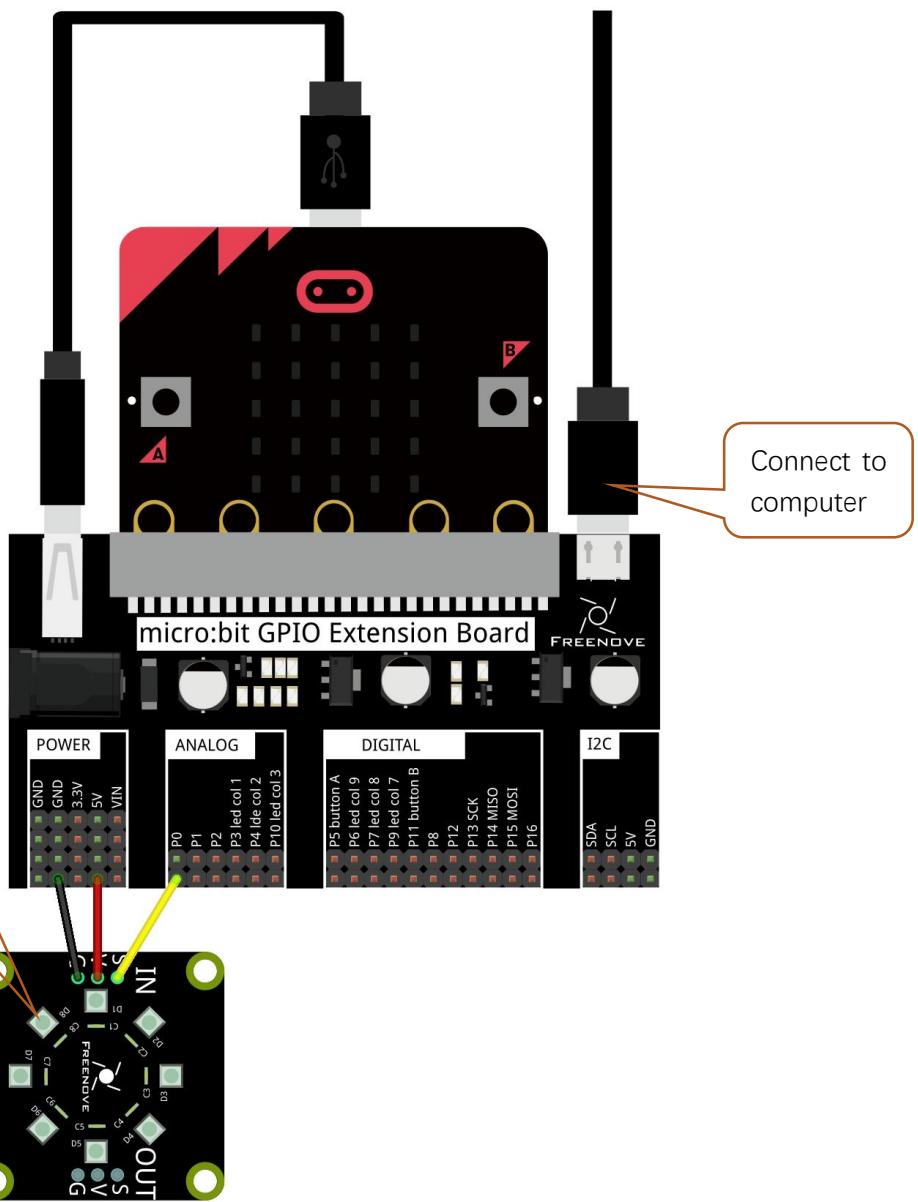
Schematic diagram

Freenove 8 RGB LED Module



Hardware connection

8 RGB LED Module need use 5V. Pay attention to the wiring.



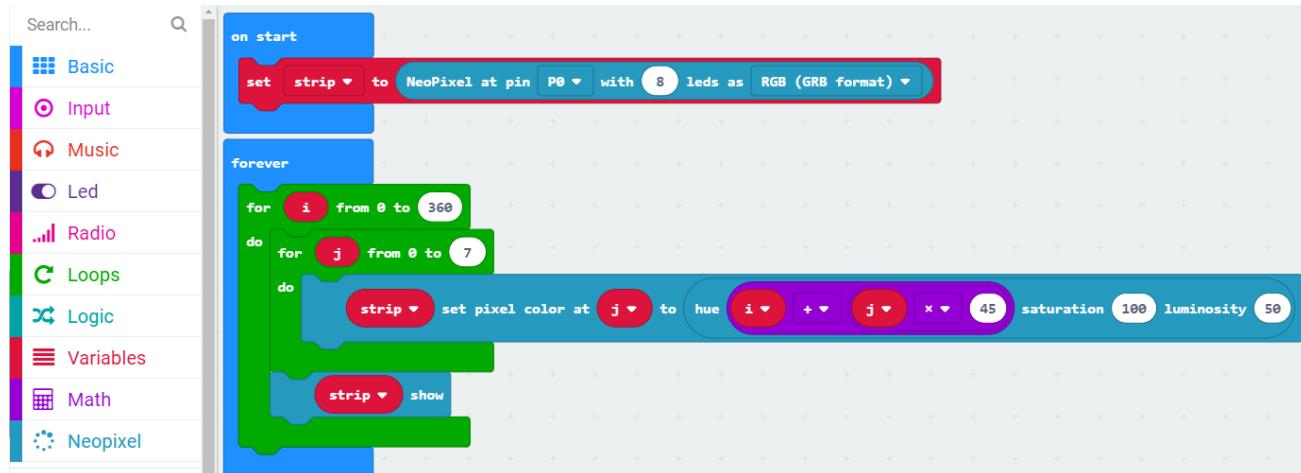
Block code

Open makecode first. Import the .hex file. The path is as below:

[\(How to import project\)](#)

File type	Path	File name
HEX file	./Projects/BlockCode/08.1_Neopixel	Neopixel.hex

After import successfully, the code is shown as below:

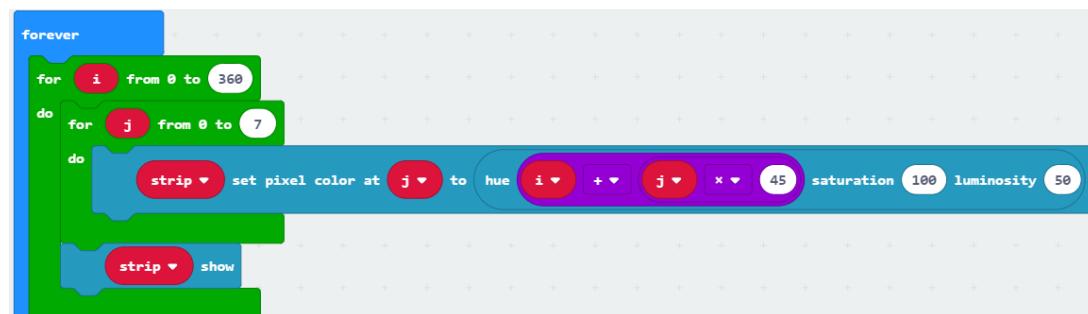


Check the connection of the circuit, confirm the correct connection of the circuit, download the code into the micro:bit, then you can see the rainbow water light.

Set the number of data pins and LEDs at "one start", as well as the type of led.



In the 0-360 for loop, the hue difference between each two LEDs is 45. When the hue changes, each led inherits the hue of the previous led. And then converts the HSL color system to the RGB color system, and return the RGB value corresponding to the current hue angle, write the value to LED to achieve the effect of the rainbow water light.



Reference

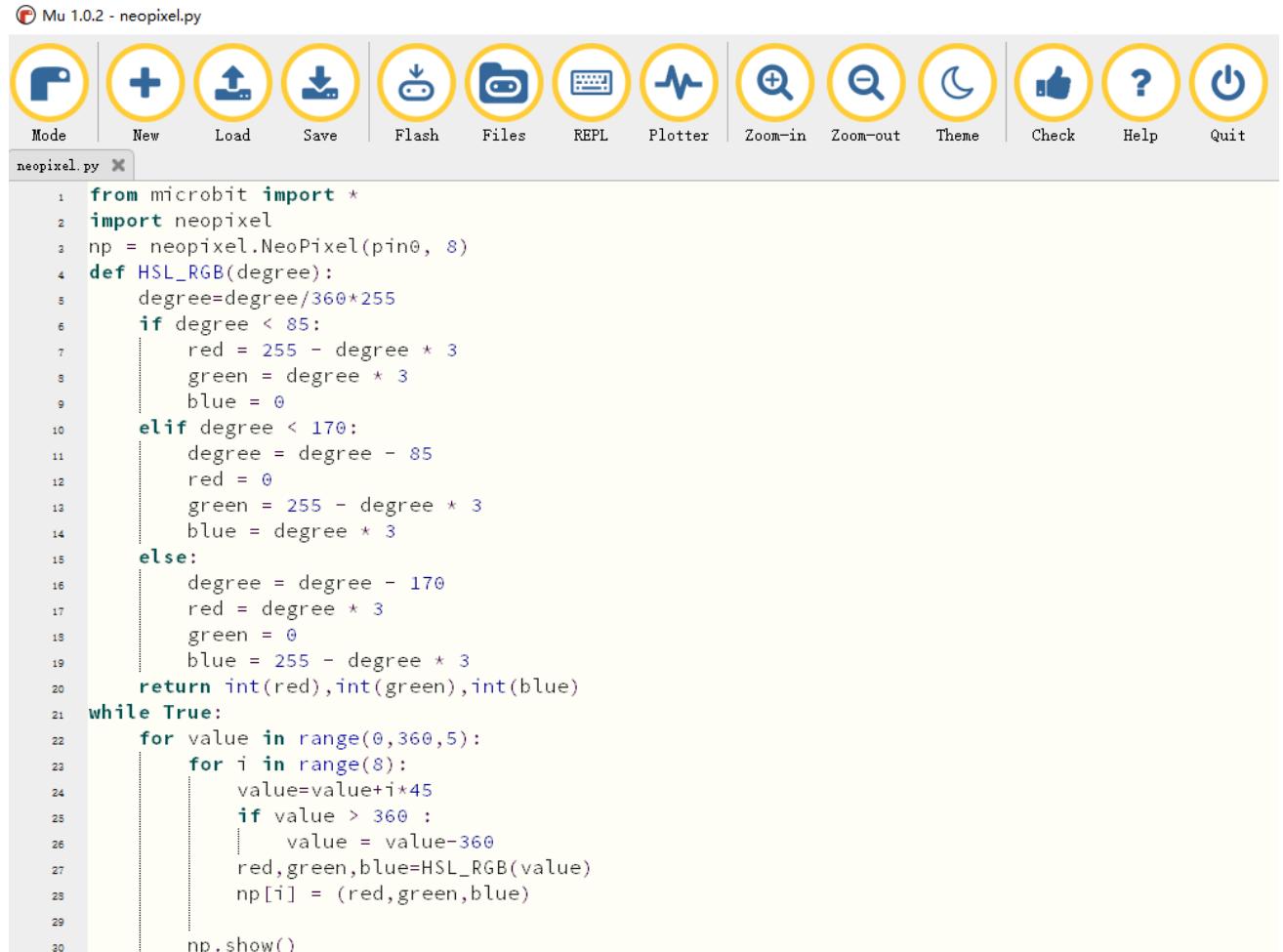
Block	Function
	Set the number of data pins and LEDs, as well as the type of led.
	Set led color
	Turn on LED

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file/Projects/PythonCode/08.1_Neopixel	Neopixel.py

After load successfully, the code is shown as below:



```

1  from microbit import *
2  import neopixel
3  np = neopixel.NeoPixel(pin0, 8)
4  def HSL_RGB(degree):
5      degree=degree/360*255
6      if degree < 85:
7          red = 255 - degree * 3
8          green = degree * 3
9          blue = 0
10     elif degree < 170:
11         degree = degree - 85
12         red = 0
13         green = 255 - degree * 3
14         blue = degree * 3
15     else:
16         degree = degree - 170
17         red = degree * 3
18         green = 0
19         blue = 255 - degree * 3
20     return int(red),int(green),int(blue)
21 while True:
22     for value in range(0,360,5):
23         for i in range(8):
24             value=value+i*45
25             if value > 360 :
26                 value = value-360
27             red,green,blue=HSL_RGB(value)
28             np[i] = (red,green,blue)
29
30     np.show()

```

Check the connection of the circuit, confirm the correct connection of the circuit, download the code into the micro:bit, then you can see the rainbow water light. ([How to download?](#))

The following is the code:

```
1 from microbit import *
2 import neopixel
3 np = neopixel.NeoPixel(pin0, 8)
4 def HSL_RGB(degree):
5     degree=degree/360*255
6     if degree < 85:
7         red = 255 - degree * 3
8         green = degree * 3
9         blue = 0
10    elif degree < 170:
11        degree = degree - 85
12        red = 0
13        green = 255 - degree * 3
14        blue = degree * 3
15    else:
16        degree = degree - 170
17        red = degree * 3
18        green = 0
19        blue = 255 - degree * 3
20    return int(red), int(green), int(blue)
21 while True:
22     for value in range(0,360,5):
23         for i in range(8):
24             value=value+i*45
25             if value > 360 :
26                 value = value-360
27             red, green, blue=HSL_RGB(value)
28             np[i] = (red, green, blue)
29     np.show()
```

Set the number of data pins and LEDs.

```
np = neopixel.NeoPixel(pin0, 8)
```

Custom HSL_RGB() function is used to convert HSL color to RGB color, returning the RGB value corresponding to the current hue angle.

```
def HSL_RGB(degree):
    degree=degree/360*255
    if degree < 85:
        red = 255 - degree * 3
        green = degree * 3
        blue = 0
    elif degree < 170:
        degree = degree - 85
```

```

red = 0
green = 255 - degree * 3
blue = degree * 3
else:
    degree = degree - 170
    red = degree * 3
    green = 0
    blue = 255 - degree * 3
return int(red), int(green), int(blue)

```

In the 0-360 for loop, the hue difference between each two LEDs is 45. When the hue changes, each led inherits the hue of the previous led. And then converts the HSL color system to the RGB color system, and return the RGB value corresponding to the current hue angle, write the value to LED to achieve the effect of the rainbow water light.

```

while True:
    for value in range(0, 360, 5):
        for i in range(8):
            value=value+i*45
            if value > 360 :
                value = value-360
            red, green, blue=HSL_RGB(value)
            np[i] = (red, green, blue)
    np.show()

```

Reference

neopixel.NeoPixel(pin, n)

Initialise a new strip of **n** number of neopixel LEDs controlled via pin **pin**.

show()

Show the pixels. Must be called for any updates to become visible.

np[i]

Set pixels by indexing them (like with a Python list).

Chapter 9 Buzzer

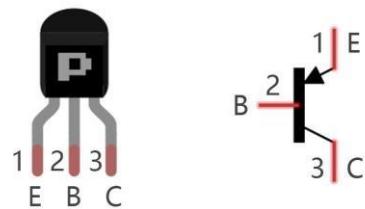
In this chapter, we use buzzers. There are two kinds of buzzer: active buzzer and passive buzzer. Buzzer is a high-power device, which needs to use transistor to drive.

Component knowledge

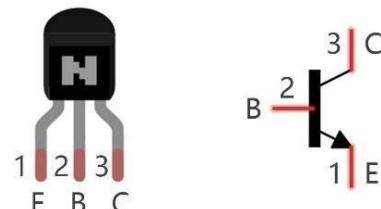
Transistor

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

PNP transistor



NPN transistor



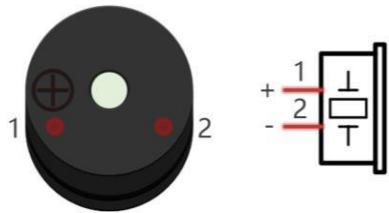
(The transistor marked 8550 is PNP, and 8050 is NPN.)

According to the transistor's characteristics, it is often used as a switch in digital circuits. For microcontroller's capacity of output current is very weak, we will use transistor to amplify current and drive large-current components.

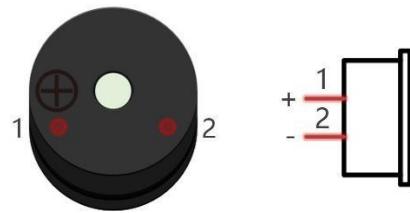
Buzzer

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

Active buzzer



Passive buzzer

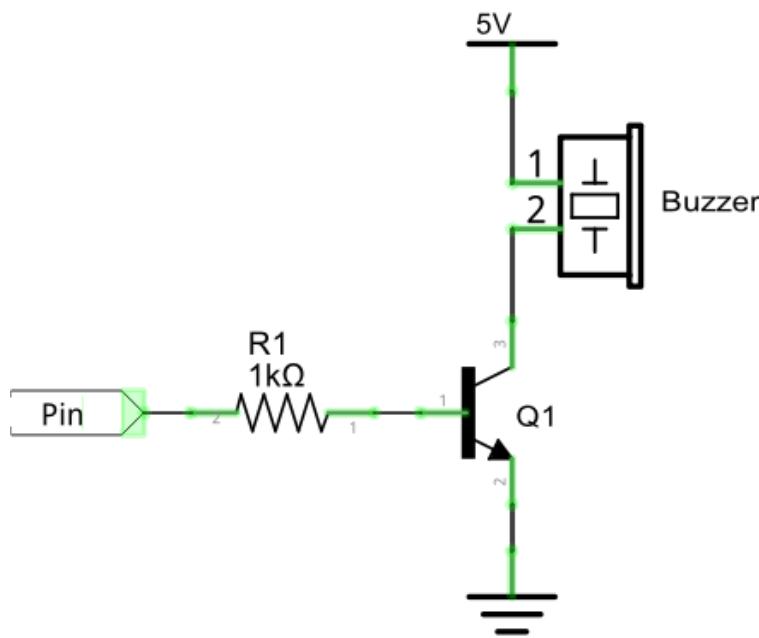


(A white label is attached on the active buzzer)

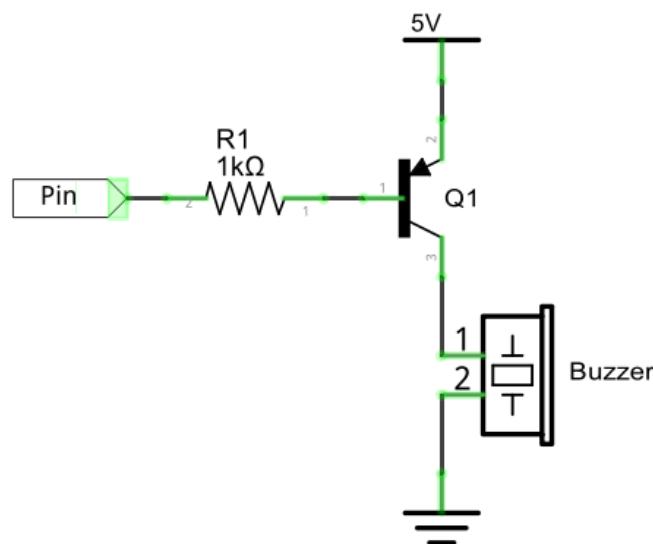
Active buzzer is easy to use. Generally, it can only make a specific frequency of sound. Passive buzzer requires an external circuit to make a sound, but it can be controlled to make a sound with different frequency. The resonant frequency of the passive buzzer is 2kHz, which means the passive buzzer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

Buzzer needs larger current when it works. But generally, microcontroller port cannot provide enough current for that. In order to control buzzer by microbit, transistor can be used to drive a buzzer indirectly.

When use NPN transistor to drive buzzer, we often adopt the following method. If mic pin outputs high level, current will flow through R1, the transistor gets conducted, and the buzzer make a sound. If microbit pin outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.



When use PNP transistor to drive buzzer, we often adopt the following method. If microbit pin outputs low level, current will flow through R1, the transistor gets conducted, buzzer make a sound. If microbit pin outputs high level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.



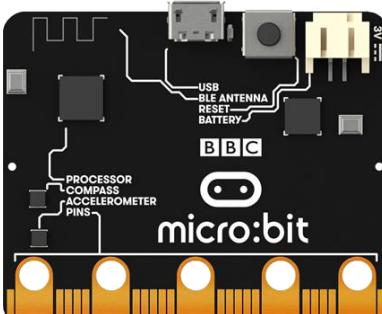
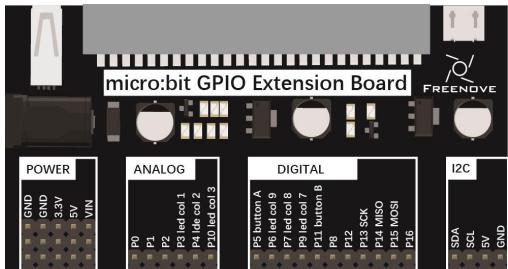
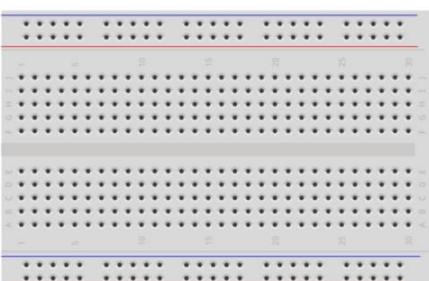
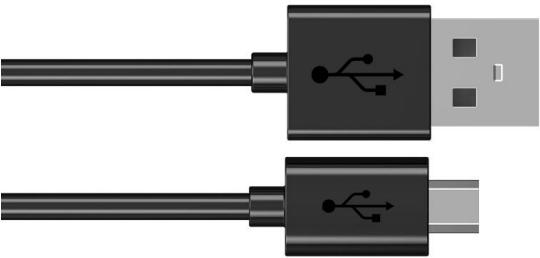
How to identify active and passive buzzer?

1. Usually, there is a label on the surface of active buzzer covering the vocal hole, but this is not an absolute judgment method.
2. Active buzzer is more complex than passive buzzer in manufacture. There are a lot of circuits and crystal oscillator elements inside active buzzer, which are covered with a waterproof coating, around its pins. Passive buzzer doesn't have these coatings. From the pins of passive buzzer, you can even see the circuit board, coils, and permanent magnets directly.

Project 9.1 Active Buzzer

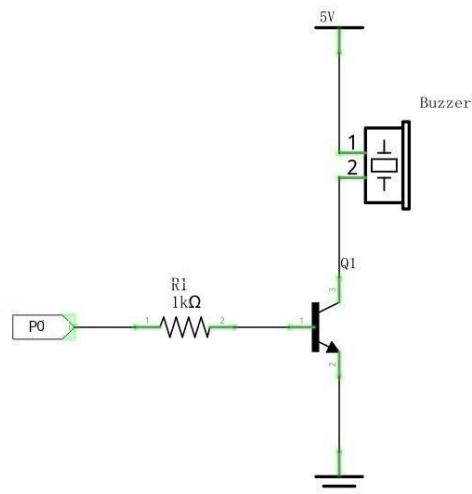
This project uses an active buzzer to play a fixed melody.

Component list

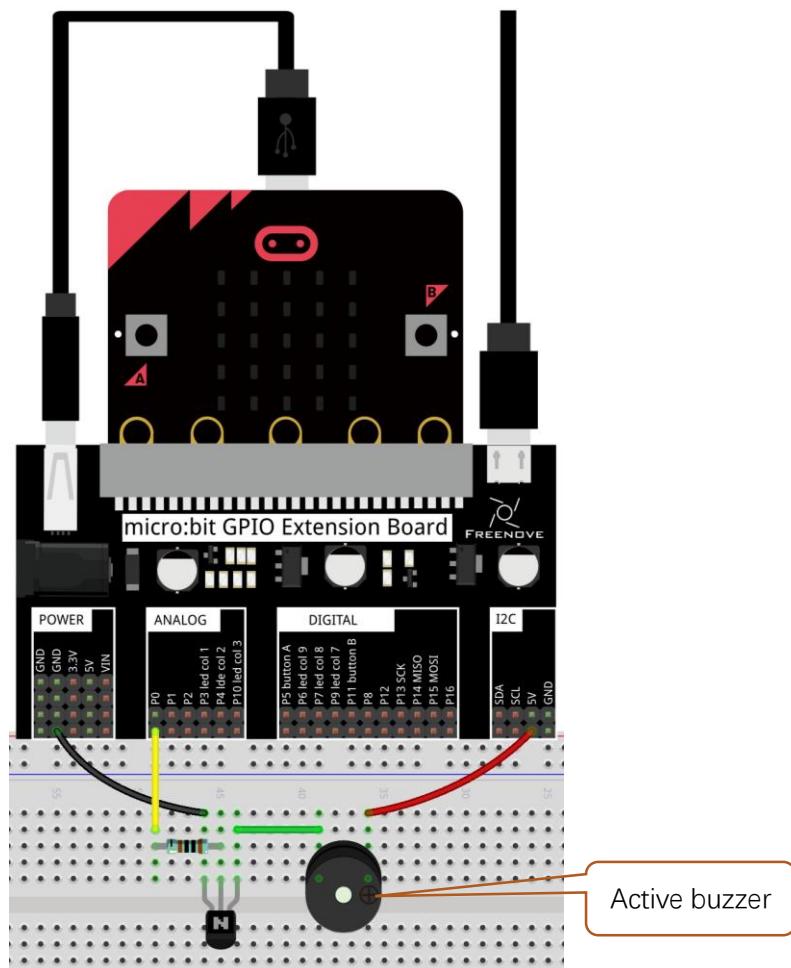
Microbit x1	Expansion board x1		
			
Breakboard x1	USB cable x2		
			
F/M x 3 M/M x1	NPN(8050) transistor x1	Resistor 1kΩ x1	Active buzzer x1
			

Circuit

Schematic diagram



Hardware connection



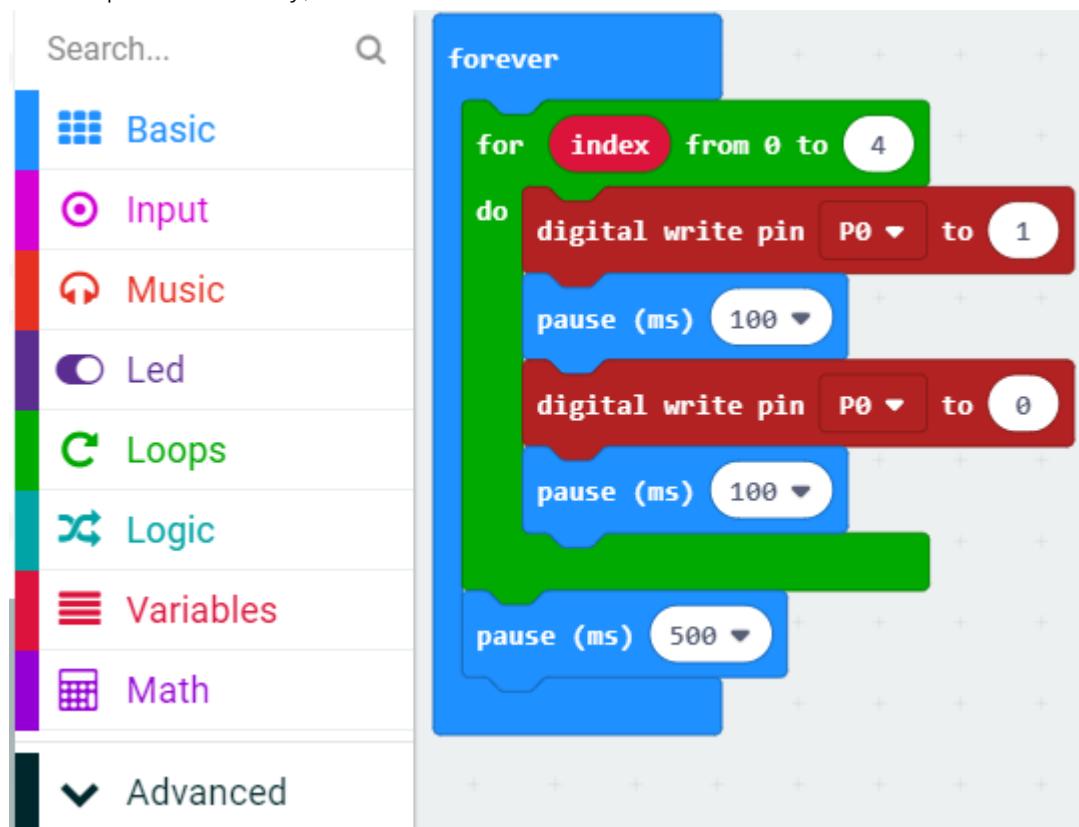
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file/Projects/BlockCode/09.1_ActiveBuzzer	ActiveBuzzer.hex

After import successfully, the code is shown as below:



Check the connection of the circuit, download the code into the micro:bit, and the buzzer on the breadboard will make sounds.

In the for loop, P0 outputs a high level to make the buzzer sounds, then delay 100ms. And then P0 outputs a low level to stop the buzzer. Then delay 100ms. After the loop ends, delay 500ms.

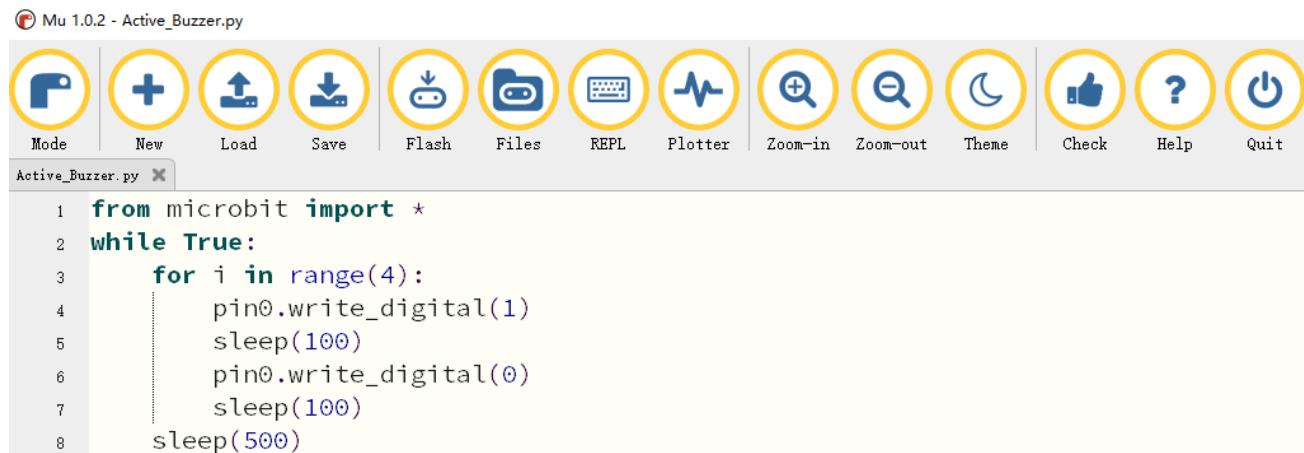


Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/09.1_ActiveBuzzer	ActiveBuzzer.py

After load successfully, the code is shown as below:



Check the connection of the circuit, download the code into the micro:bit, and the buzzer on the breadboard will make sounds.

The following is the program code:

```

1 from microbit import *
2 while True:
3     for i in range(4):
4         pin0.write_digital(1)
5         sleep(100)
6         pin0.write_digital(0)
7         sleep(100)
8     sleep(500)

```

In the for loop, P0 outputs a high level to make the buzzer sounds, then delay 100ms. And then P0 outputs a low level to stop the buzzer. Then delay 100ms. After the loop ends, delay 500ms.

```

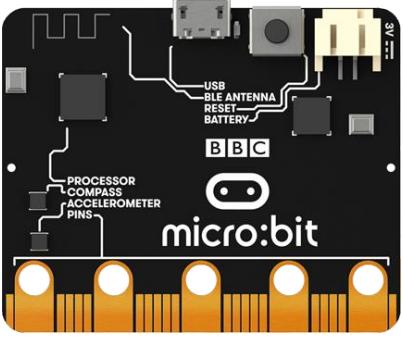
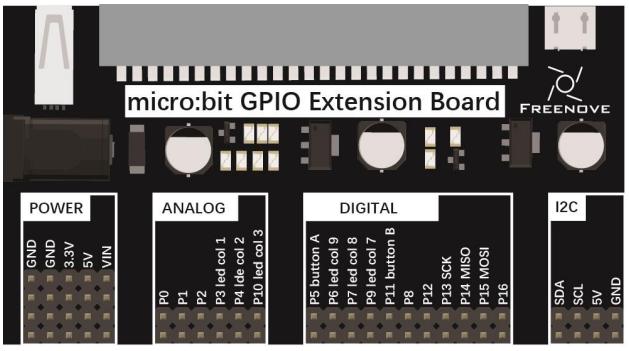
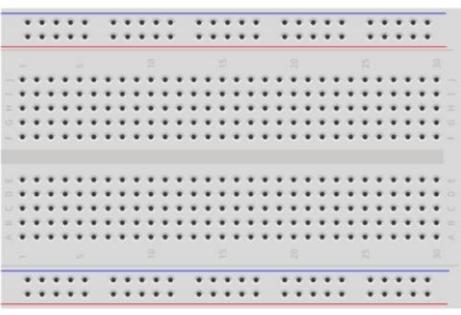
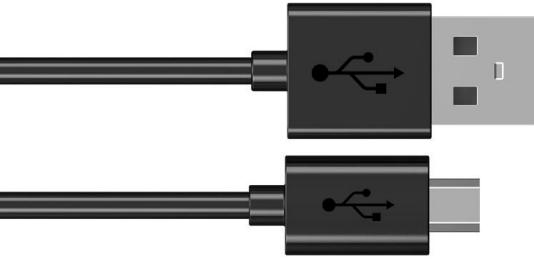
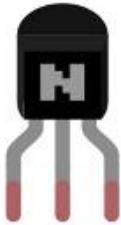
1 while True:
2     for i in range(4):
3         pin0.write_digital(1)
4         sleep(100)
5         pin0.write_digital(0)
6         sleep(100)
7     sleep(500)

```

Project 9.2 Happy Birthday Melody

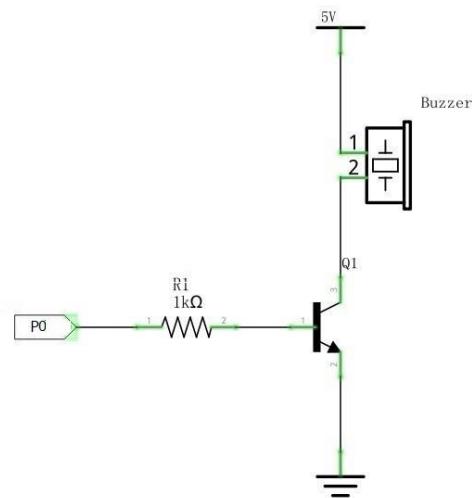
This project makes the passive buzzer to play a happy birthday melody.

Component list

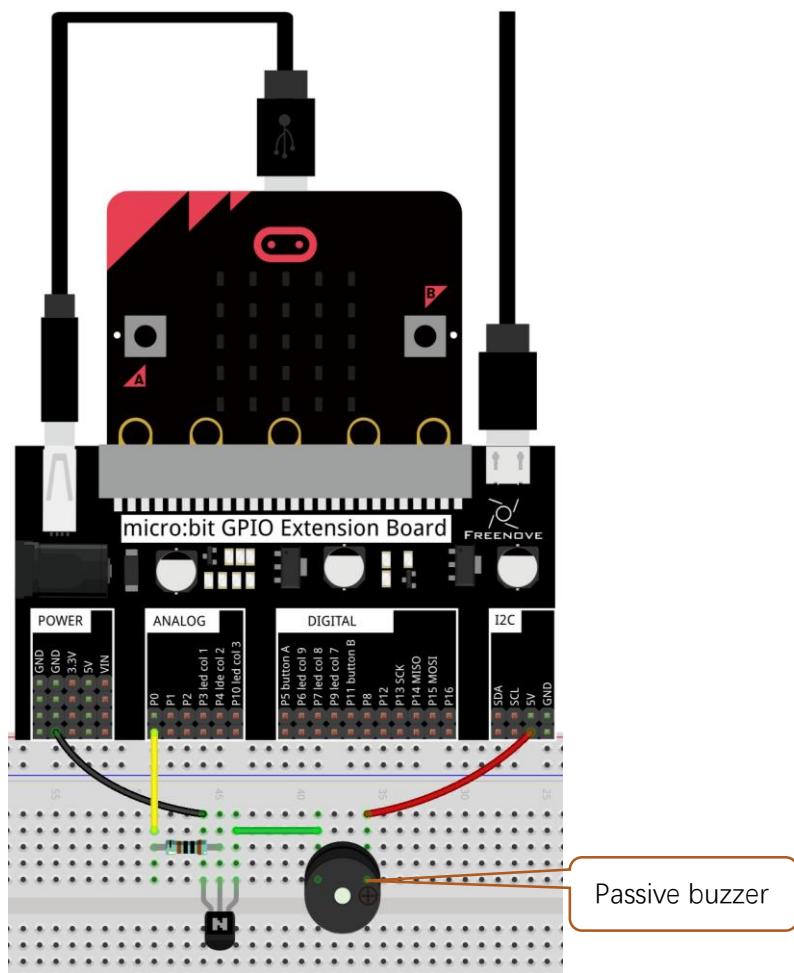
Microbit x1	Expansion board x1		
			
Breakboard x1	USB cable x2		
			
F/M x3 M/M x1	NPN(8050) transistor x1	Resistor 1kΩ x1	Passive buzzer x1
			

Circuit

Schematic diagram



Hardware connection



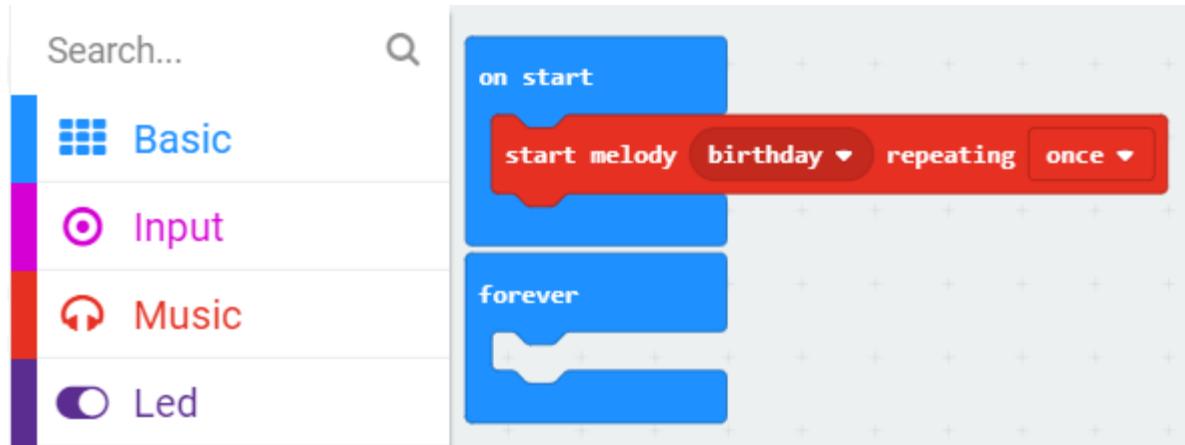
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file	../Projects/BlockCode/09.2_Play-a-melody	Play-a-melody.hex

After import successfully, the code is shown as below:



Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into the micro:bit, and the buzzer on the breadboard will play a happy birthday.

You can click on the small triangle next to "birthday" to expand the list, select other melody, and select the number of plays by clicking on the small triangle next to "once".

Reference

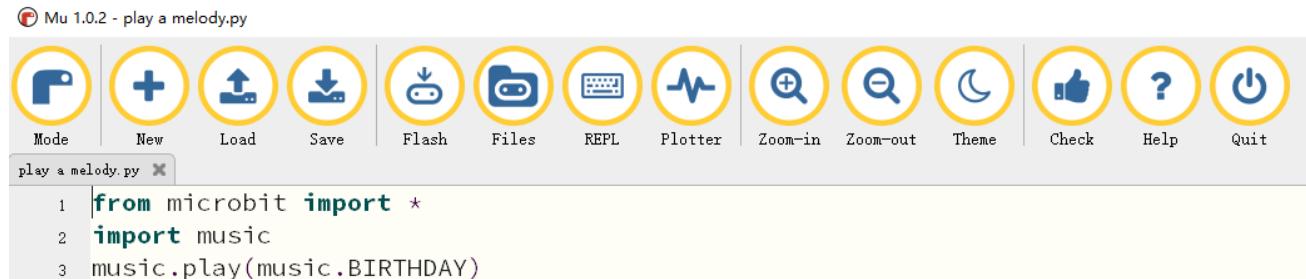
Block	Function
A red 'start melody' block with a dropdown menu showing 'dadadum' under 'melody'. It also has 'repeating' and 'once' dropdown menus under 'plays'.	Begin playing a musical melody through pin P0 of the micro:bit. There are built-in melodies that you can choose from the start melody block. These are already composed for you and are easy to use by just selecting the one you want.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/09.2_Play-a-melody	Play-a-melody.py

After load successfully, the code is shown as below:



Check the connection of the circuit, confirm that the circuit is connected correctly, and download the code into the micro:bit, and the buzzer on the breadboard will play a happy birthday. ([How to download?](#))

The following is the program code:

```

1 from microbit import *
2 import music
3 music.play(music.BIRTHDAY)

```

Reference

music.play()

It is used to play music. MicroPython has quite a lot of built-in melodies.

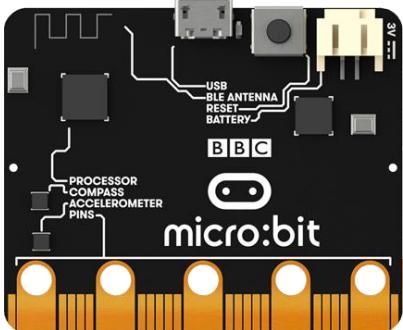
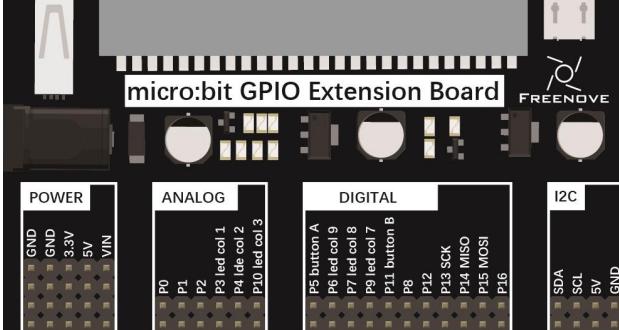
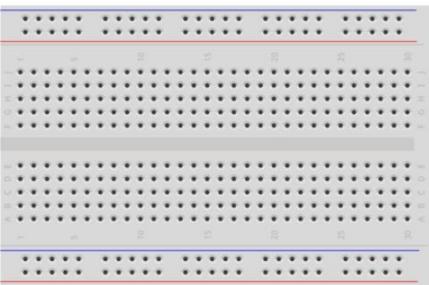
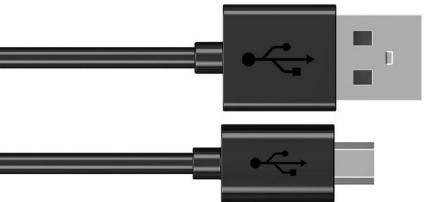
For more information, please refer to:

<https://microbit-micropython.readthedocs.io/en/latest/tutorials/music.html>

Project 9.3 Custom Melody

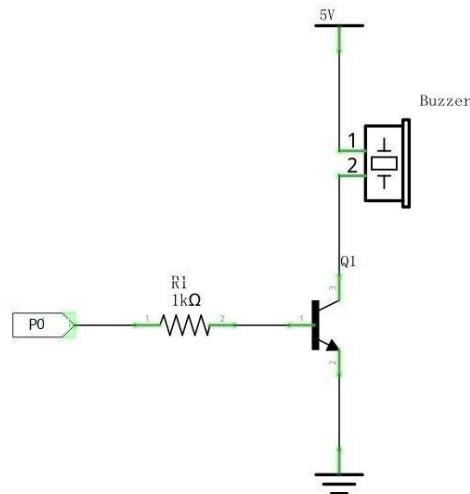
This project allows the passive buzzer to play a custom melody.

Component list

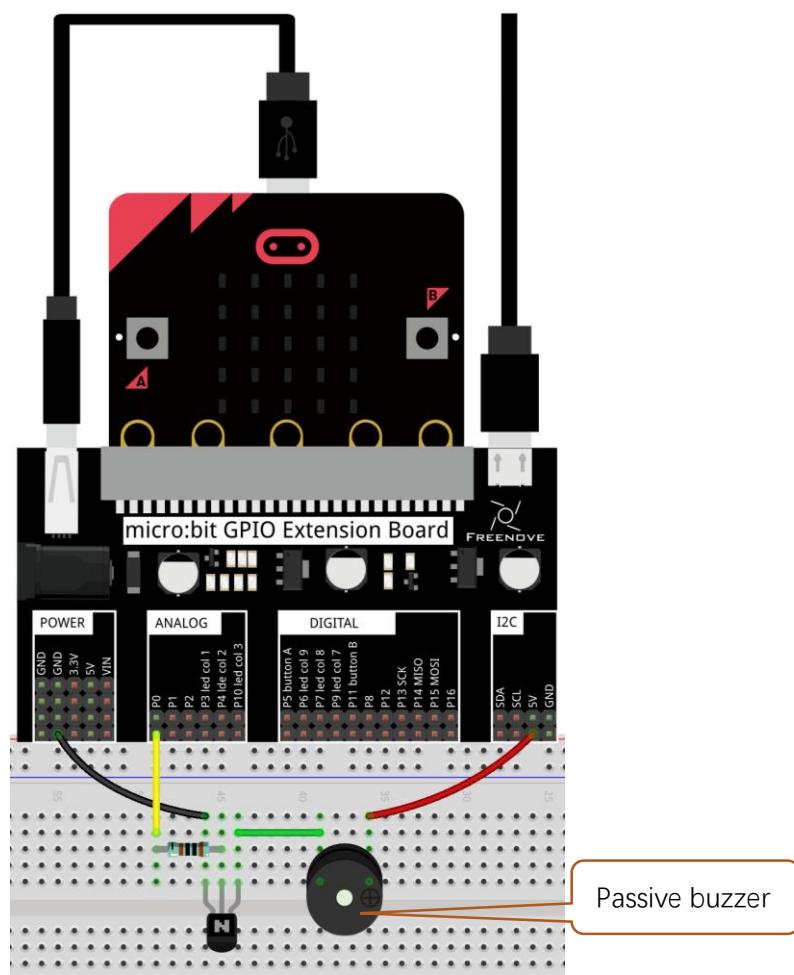
Microbit x1	Expansion board x1		
			
Breakboard x1	USB cable x2		
			
F/M x3 M/M x1	NPN(8050) transistor x1	Resistor 1kΩ x1	Passive buzzer x1
			

Circuit

Schematic diagram



Hardware connection



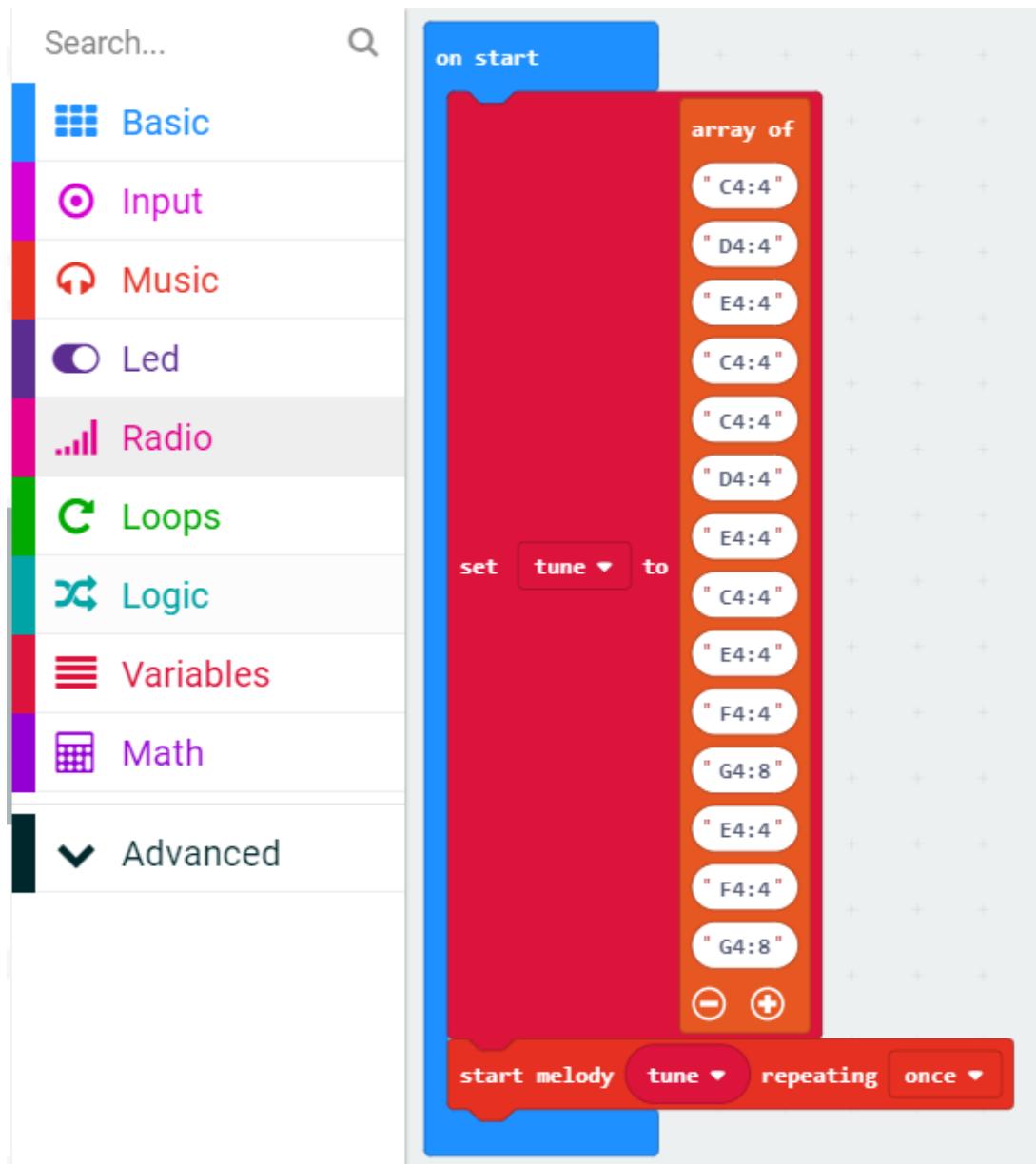
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file	./Projects/BlockCode/09.3_Play-a-custom-melody	Play-a-custom-melody.hex

After import successfully, the code is shown as below:



Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into the micro:bit, and the buzzer on the breadboard will sound a custom melody.

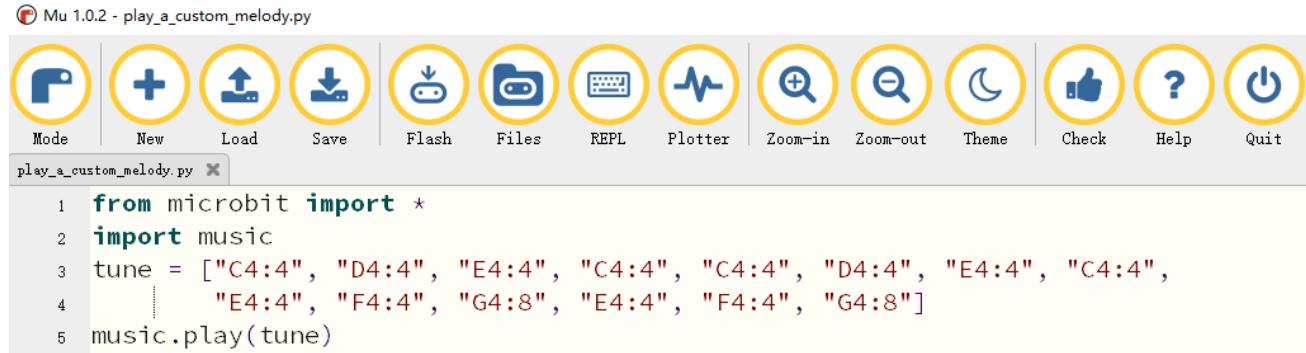
The tune array holds a custom melody, and each element in the array contains notes and beats. For example, "A1:4" refers to the note named A in octave number 1 to be played for a duration of 4.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/09.3_Play-a-custom-melody	Play-a-custom-melody.py

After load successfully, the code is shown as below:



The screenshot shows the Mu 1.0.2 IDE interface. The title bar says "Mu 1.0.2 - play_a_custom_melody.py". The menu bar has "File type" set to "Python file", "Path" set to "../Projects/PythonCode/09.3_Play-a-custom-melody", and "File name" set to "Play-a-custom-melody.py". The toolbar includes icons for Mode, New, Load, Save, Flash, Files, REPL, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The code editor window displays the following Python code:

```

1 from microbit import *
2 import music
3 tune = ["C4:4", "D4:4", "E4:4", "C4:4", "C4:4", "D4:4", "E4:4", "C4:4",
4         "E4:4", "F4:4", "G4:8", "E4:4", "F4:4", "G4:8"]
5 music.play(tune)

```

Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into the micro:bit, and the buzzer on the breadboard will sound a custom song.

The tune array holds a custom melody, and each element in the array contains notes and beats. For example, "A1:4" refers to the note named A in octave number 1 to be played for a duration of 4.

The following is the program code:

```

1 from microbit import *
2 import music
3 tune = ["C4:4", "D4:4", "E4:4", "C4:4", "C4:4", "D4:4", "E4:4", "C4:4",
4         "E4:4", "F4:4", "G4:8", "E4:4", "F4:4", "G4:8"]
5 music.play(tune)

```

Chapter 10 Serial Communication

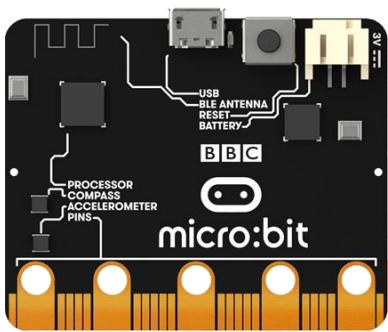
In this chapter, we will learn how to use serial port.

Project 10.1 Display the Data

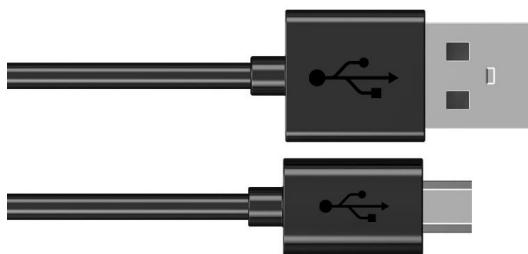
This project uses serial ports to transmit data and display data.

Component list

Microbit x1



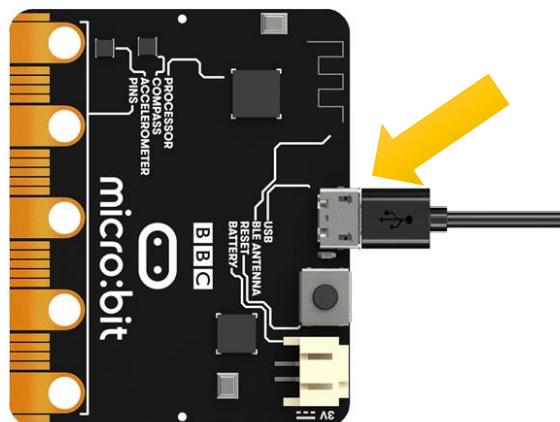
USB cable x1



Circuit

Connect micro:bit and PC via micro USB cable.

Hardware connection



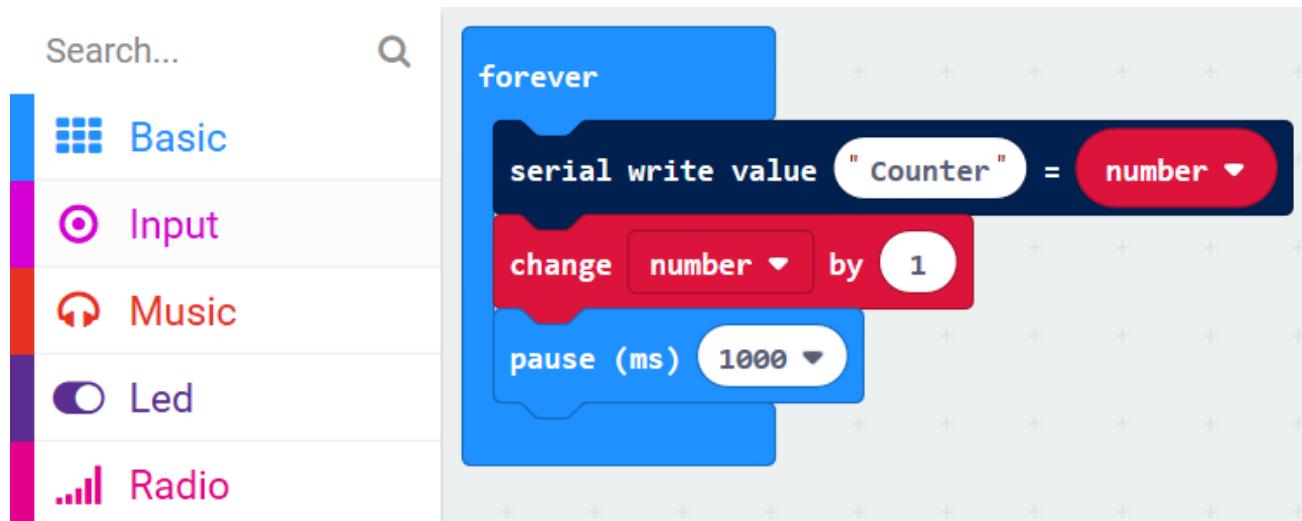
Block code

Open makecode first. Import the .hex file. The path is as below:

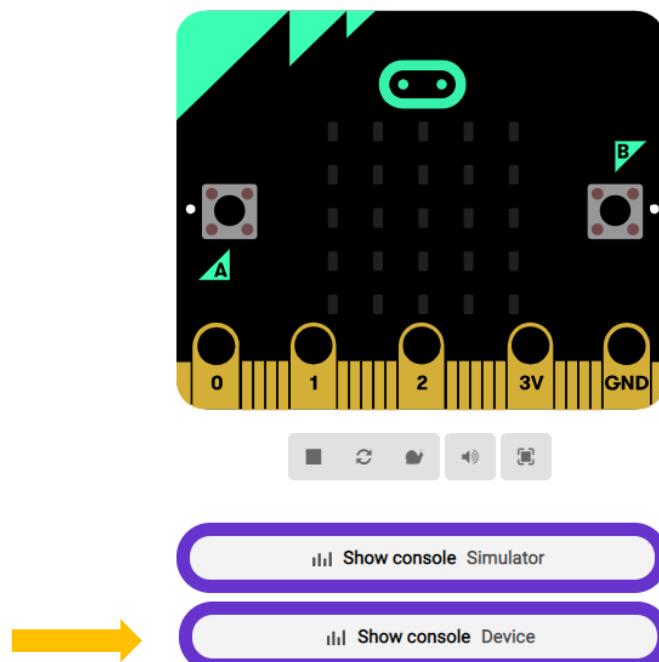
([How to import project](#))

File type	Path	File name
HEX file	./Projects/BlockCode/10.1_SerialPort	SerialPort.hex

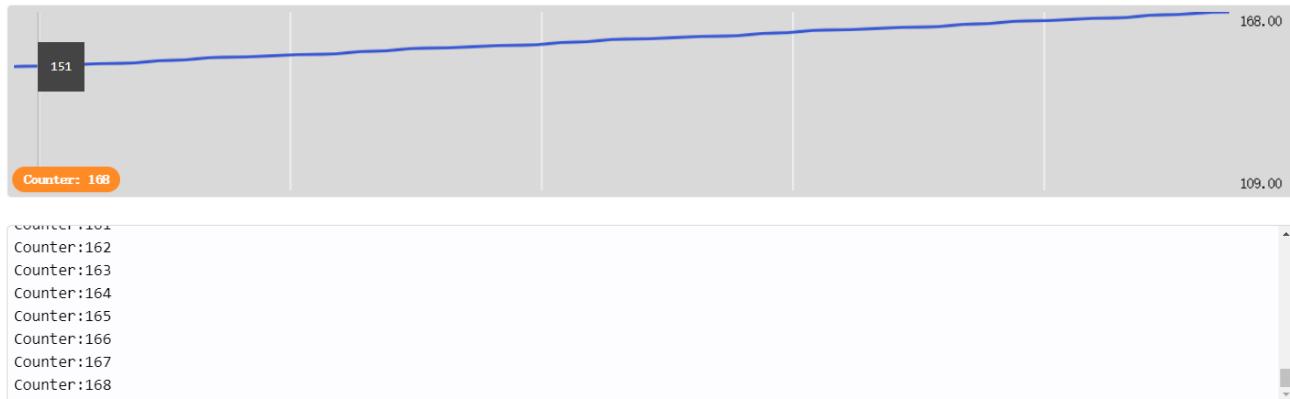
After import successfully, the code is shown as below:



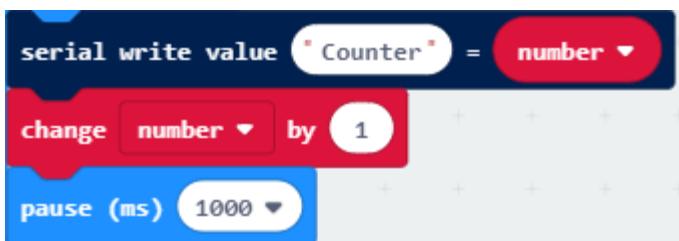
Check the connection of the circuit, confirm the correct connection of the circuit, and download the code into the micro:bit, and then open the serial controller, as shown below:



On the serial console, you can see the data sent by the microbit.



Every 1 second, the value of the variable number is incremented by 1, and the new value will be sent to the serial port.



Reference

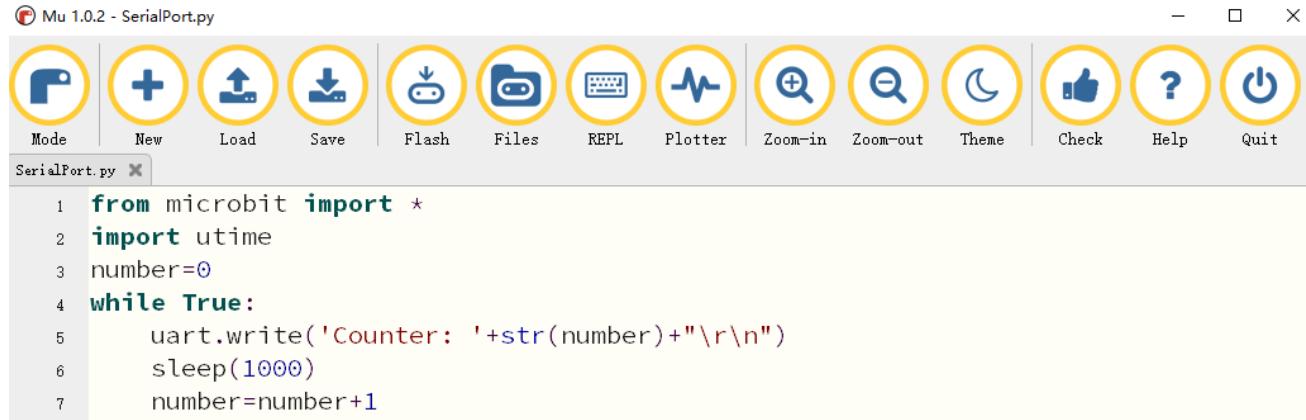
Block	Function
[serial write value] [x] = [0]	Write a name:value pair and a newline character (\r\n) to the serial port.
[change] [status v.] by [1]	The change blocks increase the value in the variable by the amount you want. This is also known as an addition assignment operation.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/10.1_SerialPort	SerialPort.py

After load successfully, the code is shown as below:

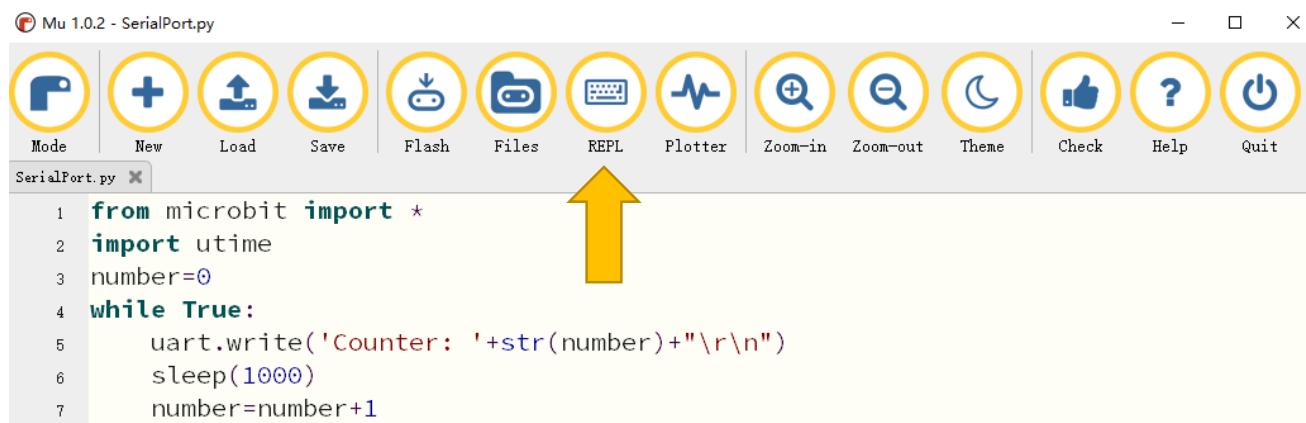


```

from microbit import *
import utime
number=0
while True:
    uart.write('Counter: '+str(number)+"\r\n")
    sleep(1000)
    number=number+1
  
```

Check the connection of the circuit and confirm that the circuit is connected correctly. Download the code into the micro:bit.

After the program is downloaded, click on REPL as shown below.



```

from microbit import *
import utime
number=0
while True:
    uart.write('Counter: '+str(number)+"\r\n")
    sleep(1000)
    number=number+1
  
```

Then press the reset button (the button on the back) of the Micro:bit and we will see the change of value.

The screenshot shows the Mu 1.0.2 interface. The top menu bar includes 'Mode' (selected), 'New', 'Load', 'Save', 'Flash', 'Files', 'REPL', 'Plotter', 'Zoom-in', 'Zoom-out', 'Theme', 'Check', 'Help', and 'Quit'. Below the menu is a code editor window titled 'Serial_Port.py' containing the following Python code:

```

1 from microbit import *
2 import utime
3 number=0
4 while True:
5     uart.write('Counter: '+str(number)+"\r\n")
6     sleep(1000)
7     number=number+1

```

Below the code editor is a 'BBC micro:bit REPL' window displaying the output of the program:

```

Counter: 1
Counter: 2
Counter: 3
Counter: 4
Counter: 5
Counter: 6
Counter: 7
Counter: 8
Counter: 9
Counter: 10
Counter: 11
Counter: 12

```

The following is the program code:

```

1 from microbit import *
2 import utime
3 number=0
4 while True:
5     uart.write('Counter: '+str(number)+"\r\n")
6     sleep(1000)
7     number=number+1

```

Every 1 second, the value of the variable number is incremented by 1, and the new value will be sent to the serial port, where "\r\n" is the meaning of the newline.

```

uart.write('Counter: '+str(number)+"\r\n")
sleep(1000)
number=number+1

```

Reference

`uart.write(x)`

Write the buffer to the bus, it can be a bytes object or a string:

```
uart.write('hello world')
```

```
uart.write(b'hello world')
```

```
uart.write(bytes([1, 2, 3]))
```

For more information, please refer to:

<https://microbit-micropython.readthedocs.io/en/latest/uart.html>

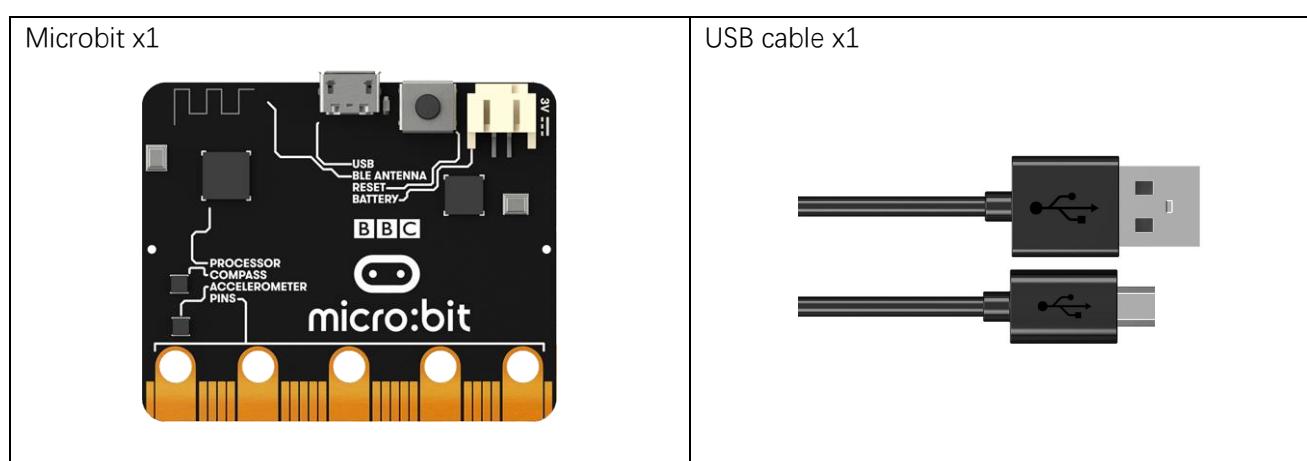
Chapter 11 Magnetometer

In this chapter, we will learn the micro:bit built-in magnetometer chip.

Project 11.1 Display Magnetometer Data

This project will print the data obtained from the magnetometer chip on the serial console.

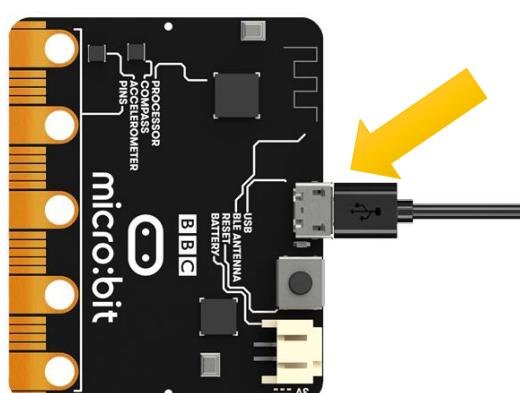
Component list



Circuit

Connect micro:bit and PC via micro USB cable.

Hardware connection



Block code

Open makecode first. Import the .hex file. The path is as below:

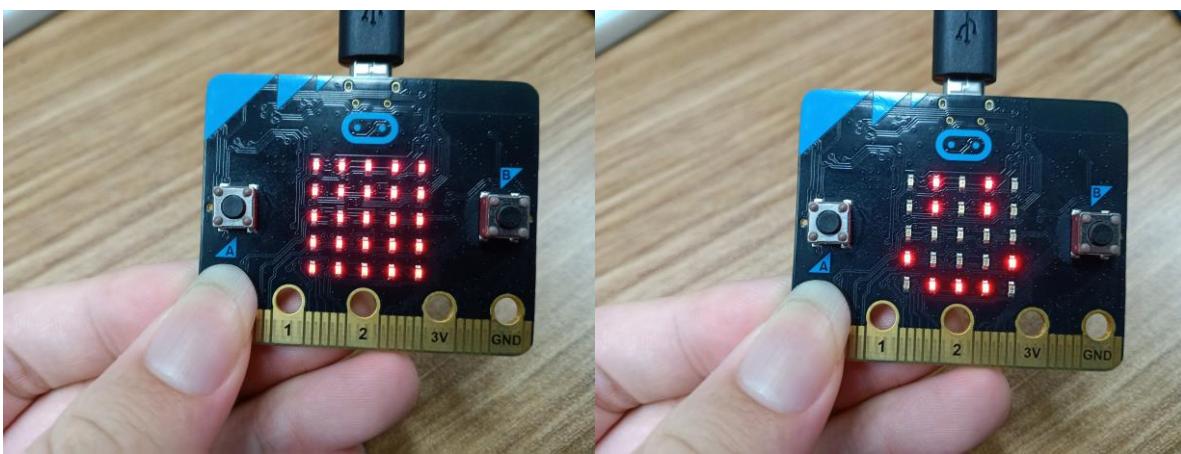
([How to import project](#))

File type	Path	File name
HEX file	../Projects/BlockCode/11.1_DisplayMagnetometerData	DisplayMagnetometerData.hex

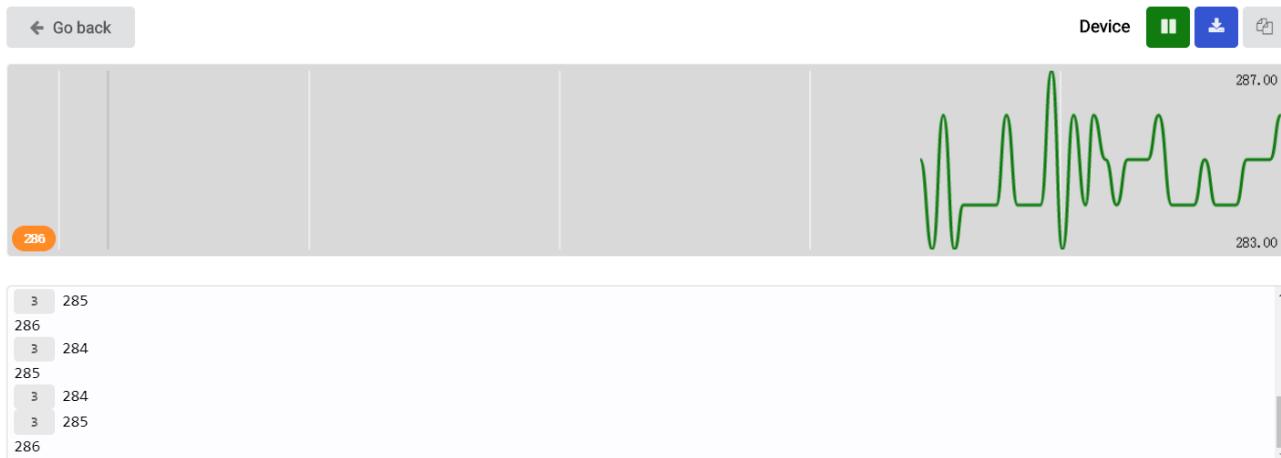
After import successfully, the code is shown as below:



Check the connection of the circuit and confirm that the circuit is connected correctly. Download the code into the micro:bit. After the download is complete, the magnetometer needs to be calibrated (calibration must be performed using the magnetometer program). Calibrating the magnetometer will cause the program to pause until the calibration is complete. Start the calibration process. An indication will be scrolled to the user, then the user will need to rotate the device so that **all** LEDs on the LED screen are illuminated, and the smile is displayed which means the calibration is complete, as shown below:

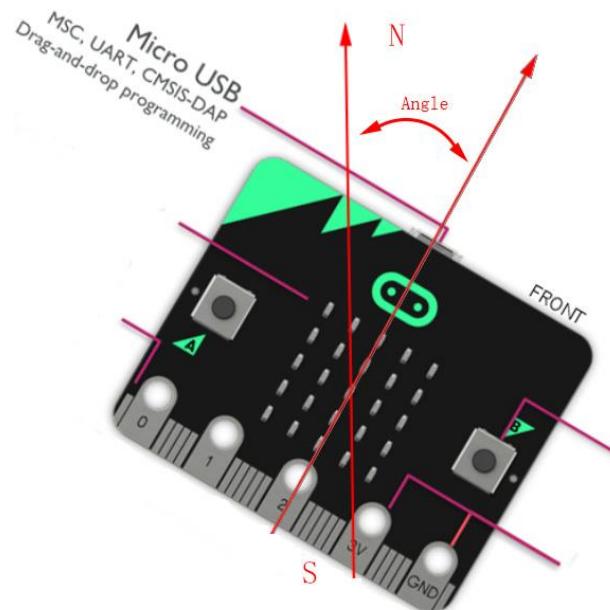


Then open the serial console ([Open the Serial Port](#)), place the micro:bit horizontally on the desktop, and rotate the micro:bit (clockwise or counterclockwise) to see the angular offset read from the magnetometer chip. As shown below:



3 indicates the number of times of consecutive readings of the same value.

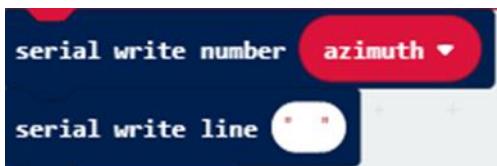
The angular offset is the angle between the directions of the micro:bit and the geographic North Pole, as shown in the following figure.



The angular offset read from the magnetometer chip is stored in the variable azimuth.



Then the value of the variable azimuth is printed on the serial port interface.



Reference

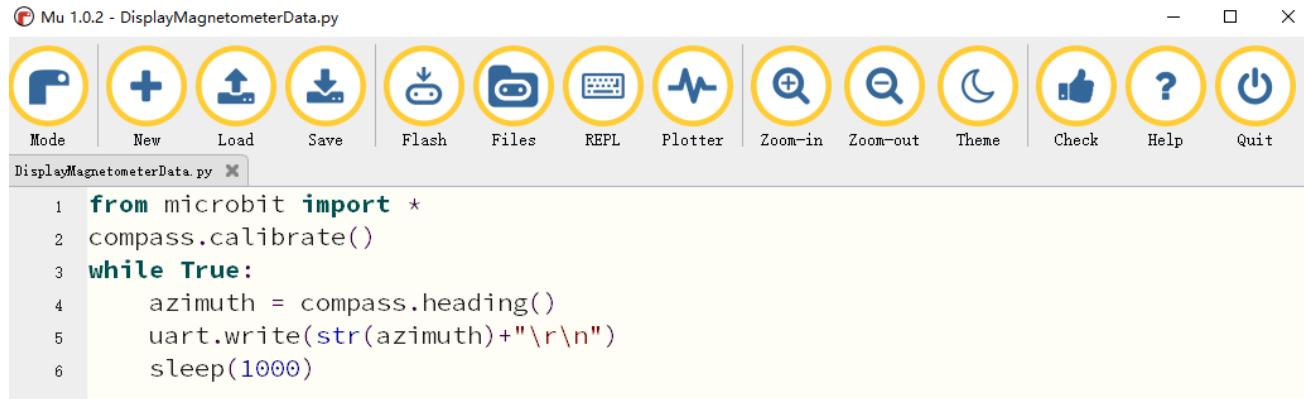
Block	Function
serial write line [\" \"]	Write a string to the serial port and start a new line of text by writing \r\n.
serial write number [0]	Write a number to the serial port.
compass heading (°)	The micro:bit measures the compass heading from 0 to 359 degrees with its magnetometer chip. Different numbers mean north, east, south, and west.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	./Projects/PythonCode/11.1_DisplayMagnetometerData	DisplayMagnetometerData.py

After load successfully, the code is shown as below:



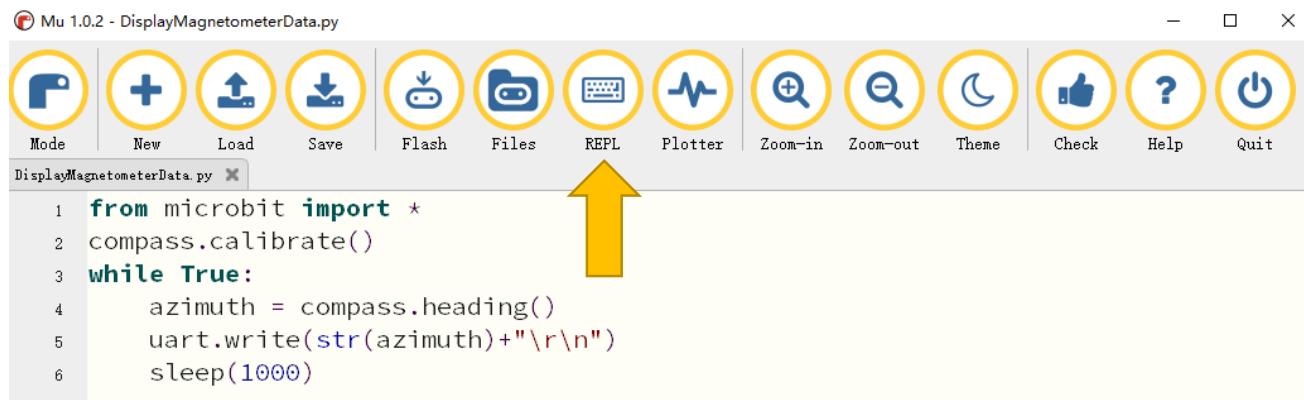
```

1 from microbit import *
2 compass.calibrate()
3 while True:
4     azimuth = compass.heading()
5     uart.write(str(azimuth)+"\r\n")
6     sleep(1000)

```

After checking the connection of the circuit and confirming that the connection of the circuit is correct, the code is downloaded into micro:bit.

After downloading the program, click REPL, as shown below.



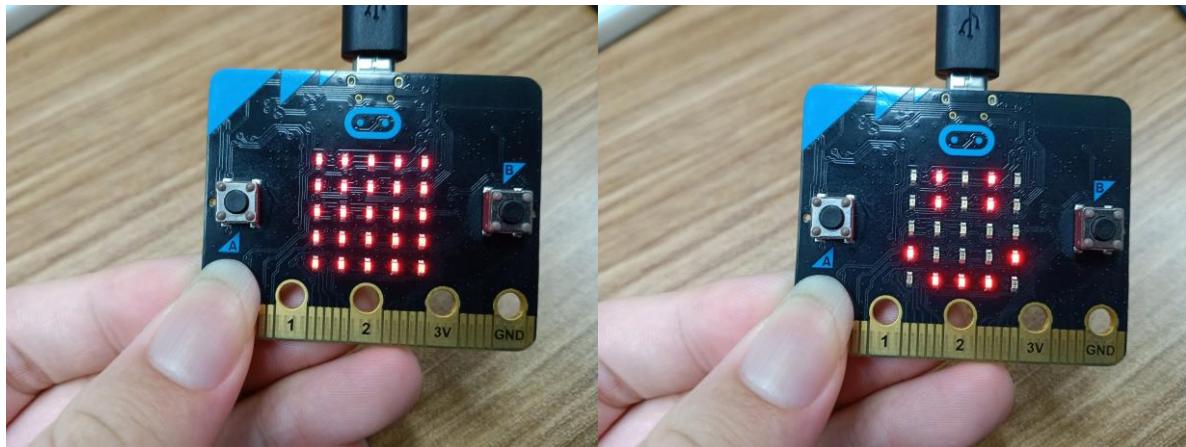
```

1 from microbit import *
2 compass.calibrate()
3 while True:
4     azimuth = compass.heading()
5     uart.write(str(azimuth)+"\r\n")
6     sleep(1000)

```

Then press the reset button of the Micro bit, you need to calibrate the magnetometer (the calibration must be performed when downloading using the magnetometer program).

Calibrating the magnetometer will cause the program to pause until the calibration is complete. Start the calibration process. An indication will be scrolled to the user, then the user will need to rotate the device so that **all** LEDs on the LED screen are illuminated, and the smile is displayed which means the calibration is complete, as shown below:



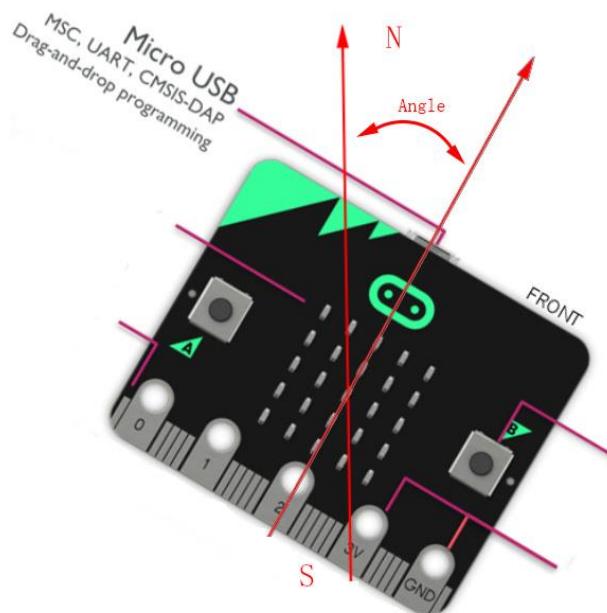
Place the micro:bit horizontally on the desktop, and rotate the micro:bit (clockwise or counterclockwise) to see the angular offset read from the magnetometer chip. As shown below:

Mu 1.0.2 - compass.py

```
Mode New Load Save Flash Files REPL Plotter Zoom-in Zoom-out Theme Check Help Quit
untitled x compass.py x
1 from microbit import *
2 compass.calibrate()
3 while True:
4     azimuth = compass.heading()
5     uart.write(str(azimuth)+"\r\n")
6     sleep(1000)

BBC micro:bit REPL
304
304
304
302
304
304
303
```

The angular offset is the angle between the direction of the micro:bit and the geographic north pole, as shown in the following figure.



The following is the program code:

```

1  from microbit import *
2  compass.calibrate()
3  while True:
4      azimuth = compass.heading()
5      uart.write(str(azimuth)+"\r\n")
6      sleep(1000)

```

Magnetometer calibration.

```
compass.calibrate()
```

The angular offset read from the magnetometer chip is stored in the variable azimuth and then printed out every 1s through a serial port.

```

azimuth = compass.heading()
uart.write(str(azimuth)+"\r\n")
sleep(1000)

```

Reference

compass.calibrate()

Starts the calibration process. An instructive message will be scrolled to the user after which they will need to rotate the device in order to draw a circle on the LED display.

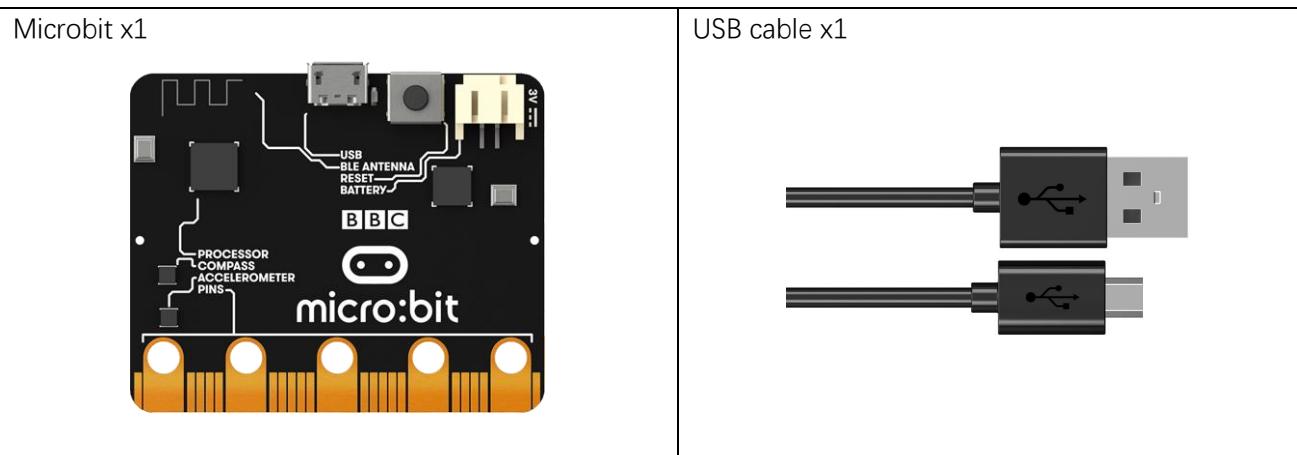
compass.heading()

Gives the compass heading, calculated from the above readings, as an integer in the range from 0 to 360, representing the angle in degrees, clockwise, with north as 0.

Project 11.2 Electronic Compass

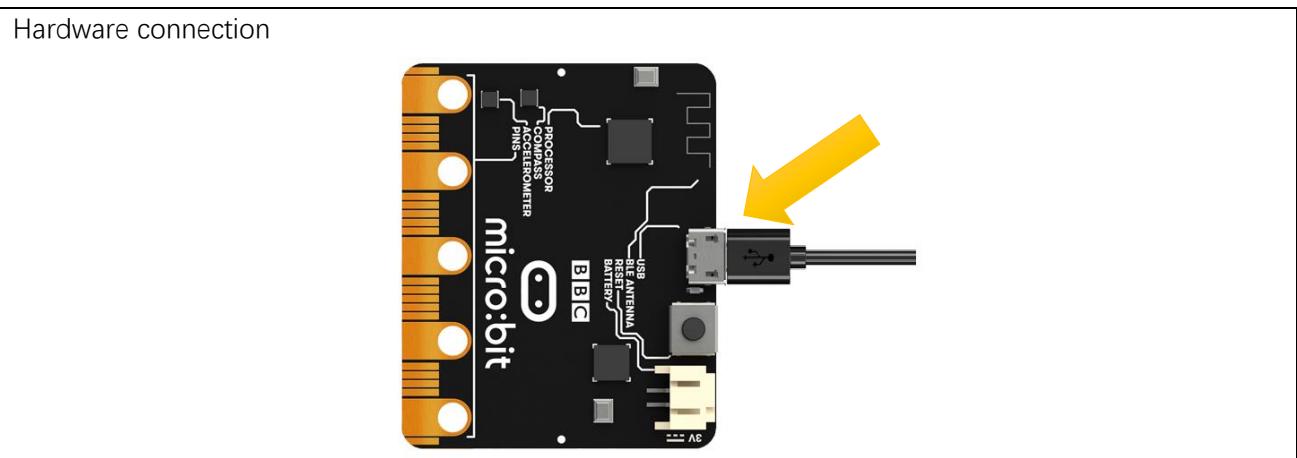
This project uses micro:bit to make an electronic compass, displaying an arrow on the micro:bit, and the arrow always points to the geographic north pole.

Component list



Circuit

Connect micro:bit and PC via micro USB cable.



Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file	./Projects/BlockCode/11.2_ElectronicCompass	ElectronicCompass.hex

After import successfully, the code is shown as below:



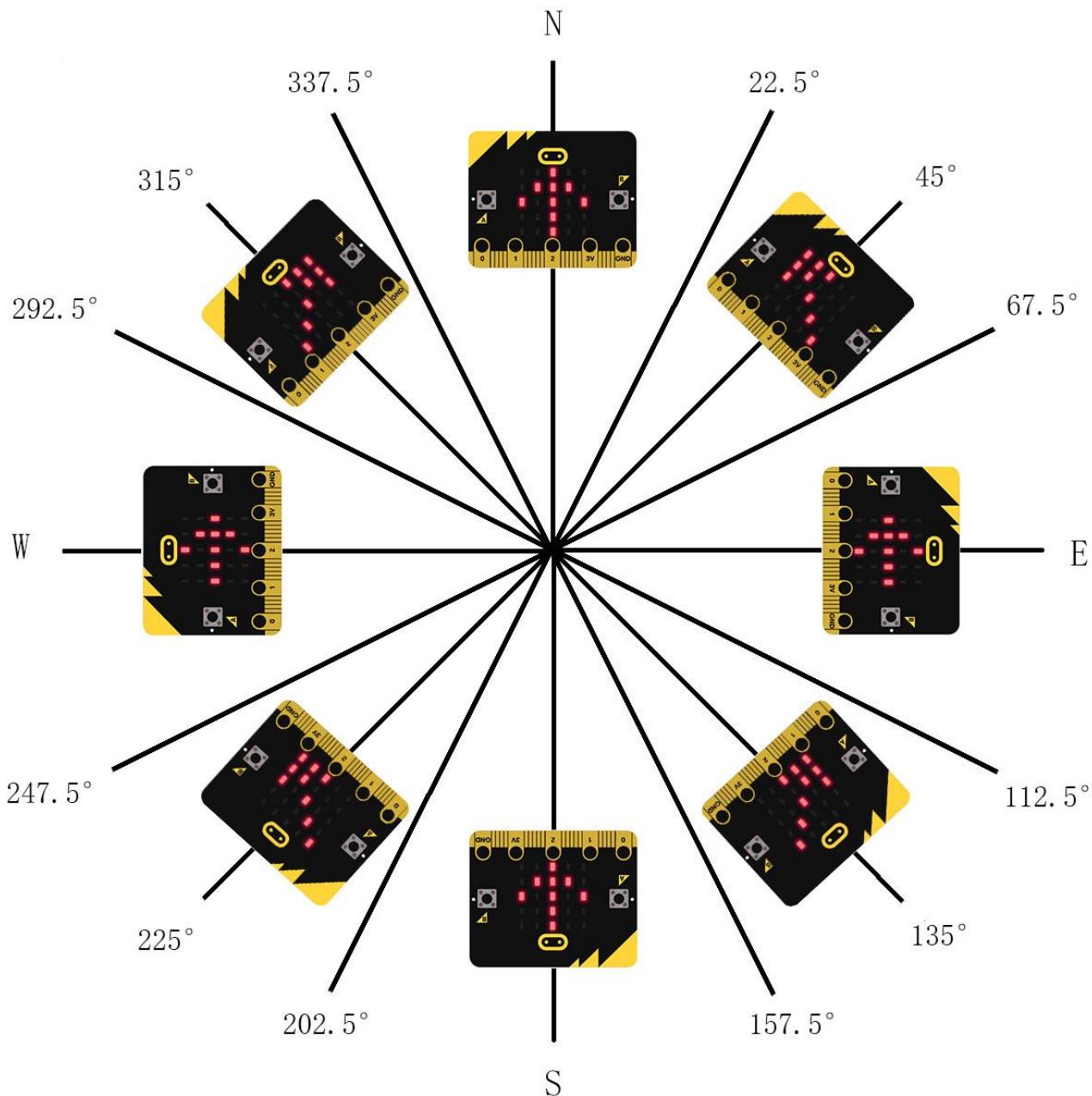
Check the connection of the circuit, confirm the correct connection of the circuit, and download the code into the micro:bit. Calibrate the electronic compass. After the calibration is successful, place the micro:bit horizontally and turn the micro:bit to see that the arrow points to the geography Arctic.

There are eight kinds of arrows in the direction: northwest, west, southwest, south, southeast, east, northeast, north, each direction is 45 degrees apart. Assuming that the direction of the micro:bit is rotated 45 degrees north from the north to the northeast of the geography, the arrow shown should be reversed, that is, rotated -45 degrees, pointing to the northwest of the micro:bit, the geographic north pole. Therefore, the direction of the arrow is adjusted according to the angular offset from the geographic North Pole.

When the variable azimuth is less than 22.5, greater than 337.5, the arrow points to the true north of the micro:bit.

When the variable azimuth is greater than 22.5, less than 67.5, the arrow points to the northwest of the micro:bit.

Each direction is 45 degrees apart, and so on, the arrow always points to the geographic north, as shown in the following figure:



The angular offset read from the magnetometer chip is stored in the variable azimuth

set azimuth to compass heading (°)

Determine the value of the variable azimuth to change the direction of the arrow.



Reference

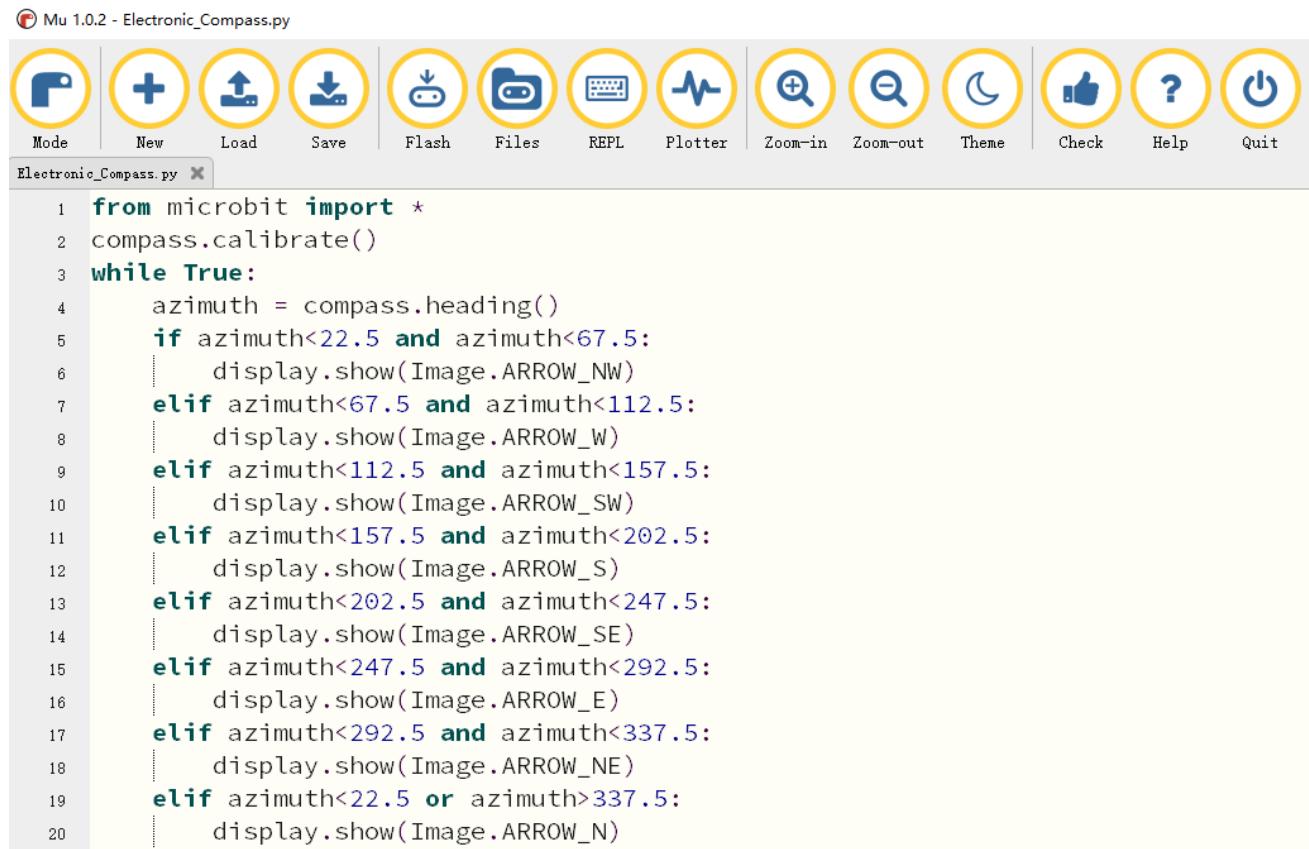
Block	Function
if true then 	Run code depending on whether a Boolean condition is true or false.
show arrow North 	Shows the selected arrow on the LED screen

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/11.2_ElectronicCompass	ElectronicCompass.py

After load successfully, the code is shown as below:



```

1 from microbit import *
2 compass.calibrate()
3 while True:
4     azimuth = compass.heading()
5     if azimuth<22.5 and azimuth<67.5:
6         display.show(Image.ARROW_NW)
7     elif azimuth<67.5 and azimuth<112.5:
8         display.show(Image.ARROW_W)
9     elif azimuth<112.5 and azimuth<157.5:
10        display.show(Image.ARROW_SW)
11    elif azimuth<157.5 and azimuth<202.5:
12        display.show(Image.ARROW_S)
13    elif azimuth<202.5 and azimuth<247.5:
14        display.show(Image.ARROW_SE)
15    elif azimuth<247.5 and azimuth<292.5:
16        display.show(Image.ARROW_E)
17    elif azimuth<292.5 and azimuth<337.5:
18        display.show(Image.ARROW_NE)
19    elif azimuth<22.5 or azimuth>337.5:
20        display.show(Image.ARROW_N)

```

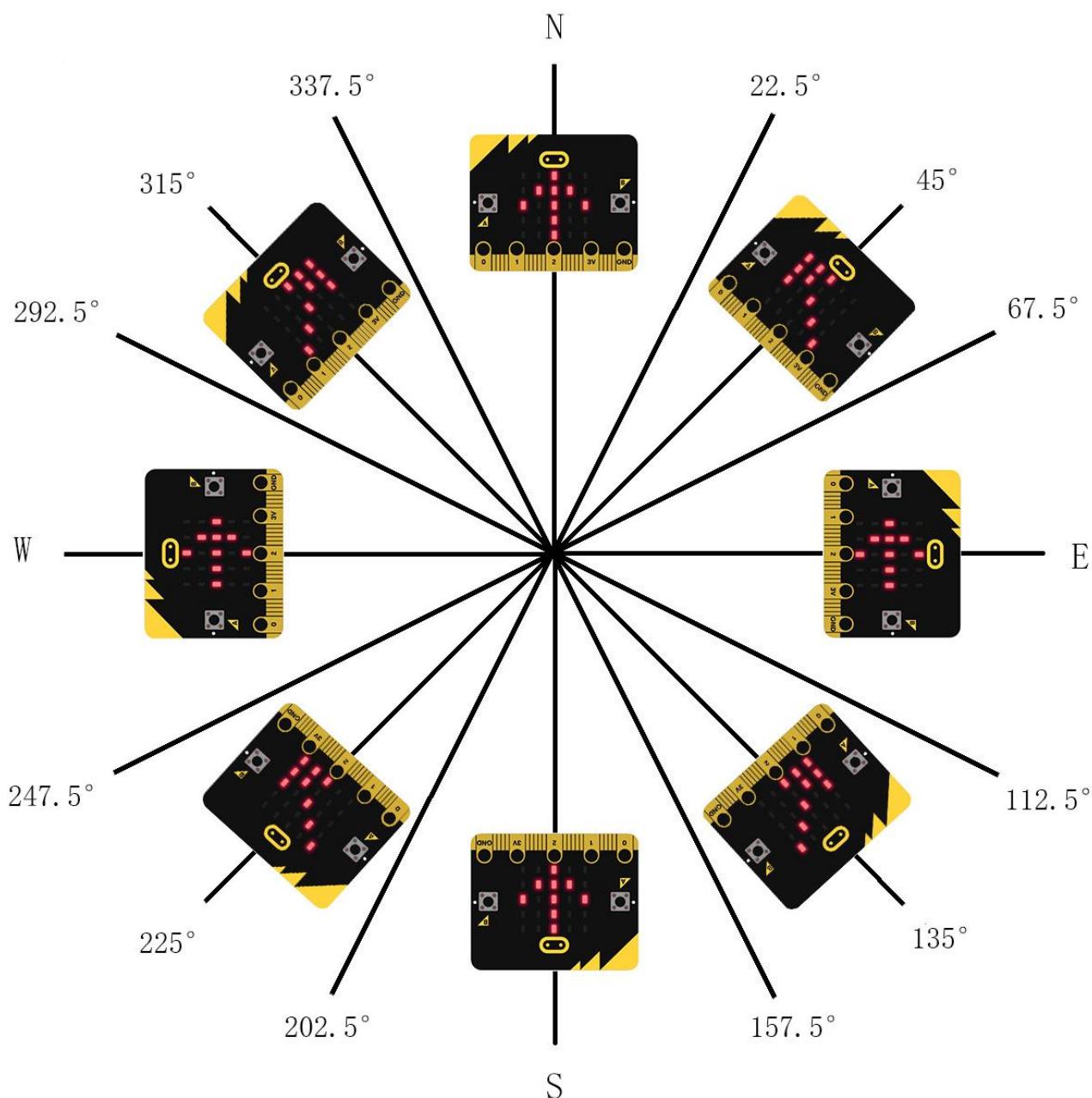
Check the connection of the circuit, confirm the correct connection of the circuit, and download the code into the micro:bit. Calibrate the electronic compass. After the calibration is successful, place the micro:bit horizontally and turn the micro:bit to see that the arrow points to the geography Arctic.

There are eight kinds of arrows in the direction: northwest, west, southwest, south, southeast, east, northeast, north, each direction is 45 degrees apart. Assuming that the direction of the micro:bit is rotated 45 degrees north from the north to the northeast of the geography, the arrow shown should be reversed, that is, rotated -45 degrees, pointing to the northwest of the micro:bit, the geographic north pole. Therefore, the direction of the arrow is adjusted according to the angular offset from the geographic North Pole.

When the variable azimuth is less than 22.5, greater than 337.5, the arrow points to the true north of the micro:bit.

When the variable azimuth is greater than 22.5, less than 67.5, the arrow points to the northwest of the micro:bit.

Each direction is 45 degrees apart, and so on, the arrow always points to the geographic north, as shown in the following figure:



The following is the program code:

```
1 from microbit import *
2 compass.calibrate()
3 while True:
4     azimuth = compass.heading()
5     if azimuth<22.5 and azimuth<67.5:
6         display.show(Image.ARROW_NW)
7     elif azimuth<67.5 and azimuth<112.5:
8         display.show(Image.ARROW_W)
9     elif azimuth<112.5 and azimuth<157.5:
10        display.show(Image.ARROW_SW)
11    elif azimuth<157.5 and azimuth<202.5:
12        display.show(Image.ARROW_S)
13    elif azimuth<202.5 and azimuth<247.5:
14        display.show(Image.ARROW_SE)
15    elif azimuth<247.5 and azimuth<292.5:
16        display.show(Image.ARROW_E)
17    elif azimuth<292.5 and azimuth<337.5:
18        display.show(Image.ARROW_NE)
19    elif azimuth<22.5 and azimuth>337.5:
20        display.show(Image.ARROW_N)
```

Calibrate the electronic compass first and store the data on the variable azimuth.

```
compass.calibrate()
azimuth = compass.heading()
```

Determine the value of the variable azimuth and change the direction of the arrow.

```
if azimuth<22.5 and azimuth<67.5:
    display.show(Image.ARROW_NW)
elif azimuth<67.5 and azimuth<112.5:
    display.show(Image.ARROW_W)
elif azimuth<112.5 and azimuth<157.5:
    display.show(Image.ARROW_SW)
elif azimuth<157.5 and azimuth<202.5:
    display.show(Image.ARROW_S)
elif azimuth<202.5 and azimuth<247.5:
    display.show(Image.ARROW_SE)
elif azimuth<247.5 and azimuth<292.5:
    display.show(Image.ARROW_E)
elif azimuth<292.5 and azimuth<337.5:
    display.show(Image.ARROW_NE)
elif azimuth<22.5 and azimuth>337.5:
    display.show(Image.ARROW_N)
```

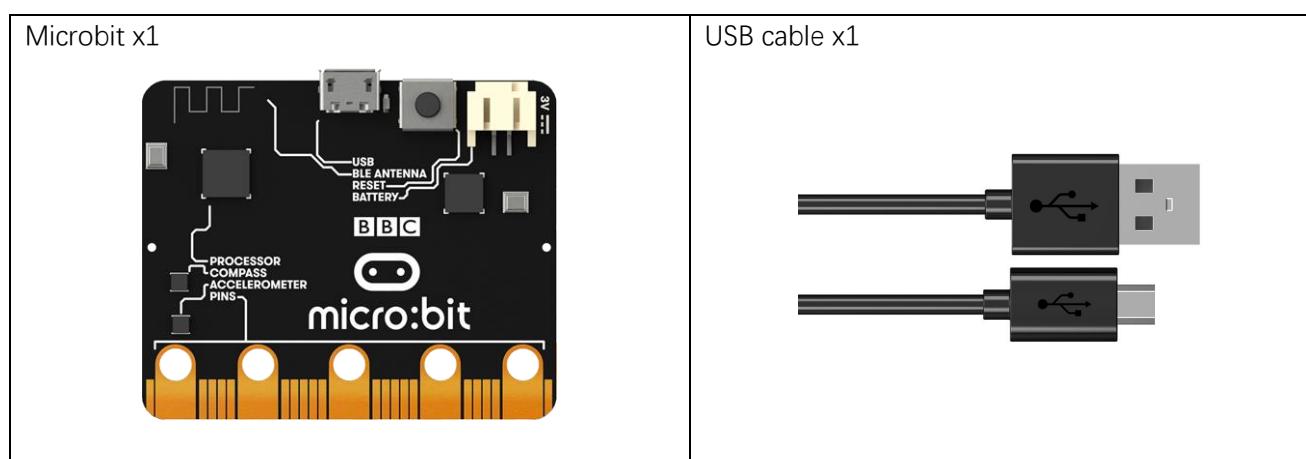
Chapter 12 Accelerometer

In this chapter, we will learn about micro:bit built-in accelerometer sensor.

Project 12.1 Display Accelerometer Data

This project will obtain data from the accelerometer sensor and print it on the serial console.

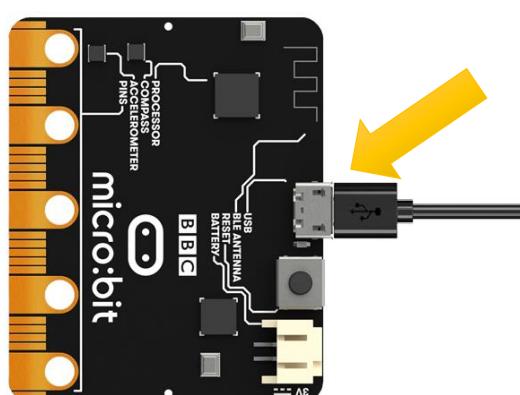
Component list



Circuit

Connect micro:bit and PC via micro USB cable.

Hardware connection



Block code

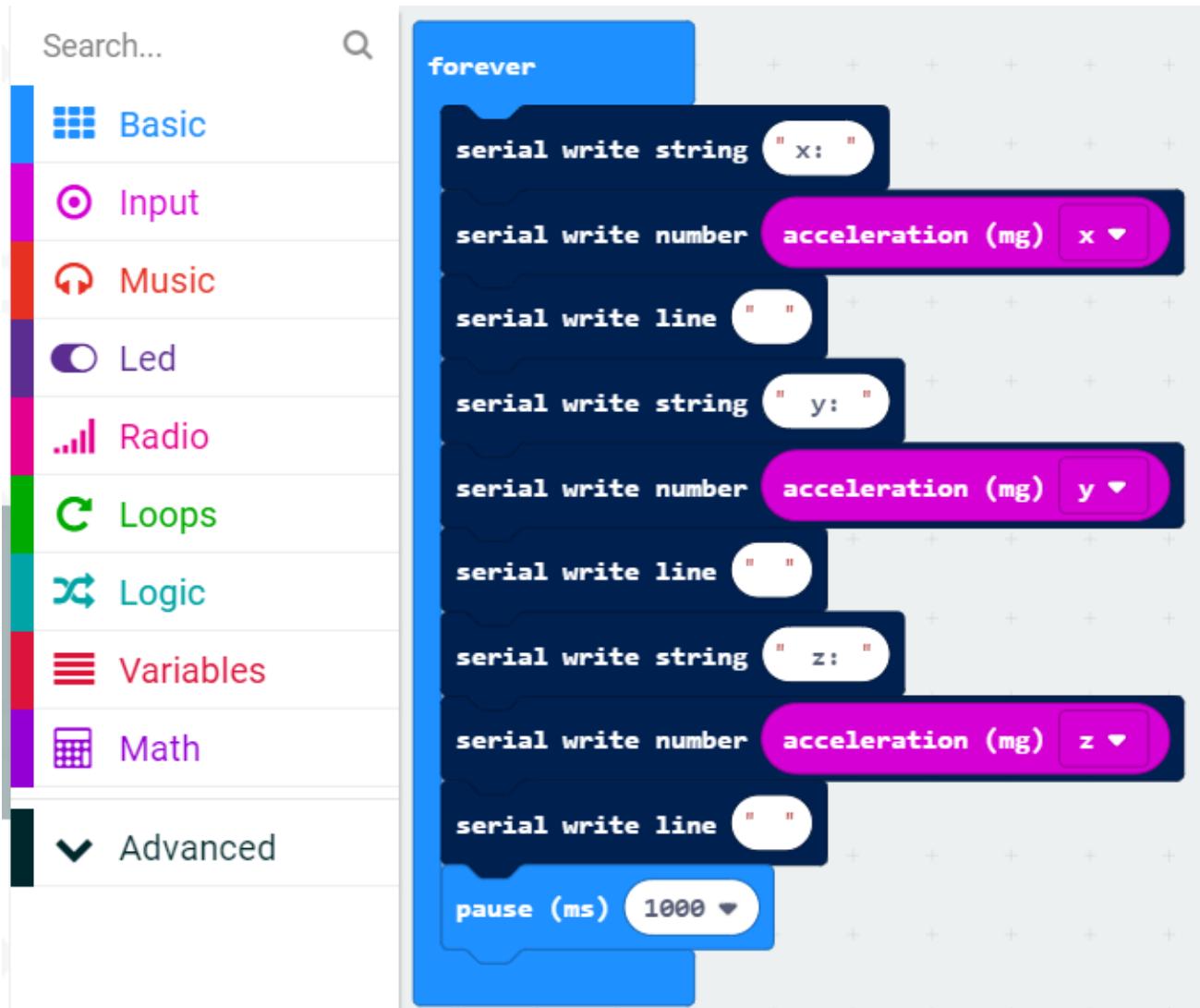
Open makecode first.

Import the .hex file. The path is as below:

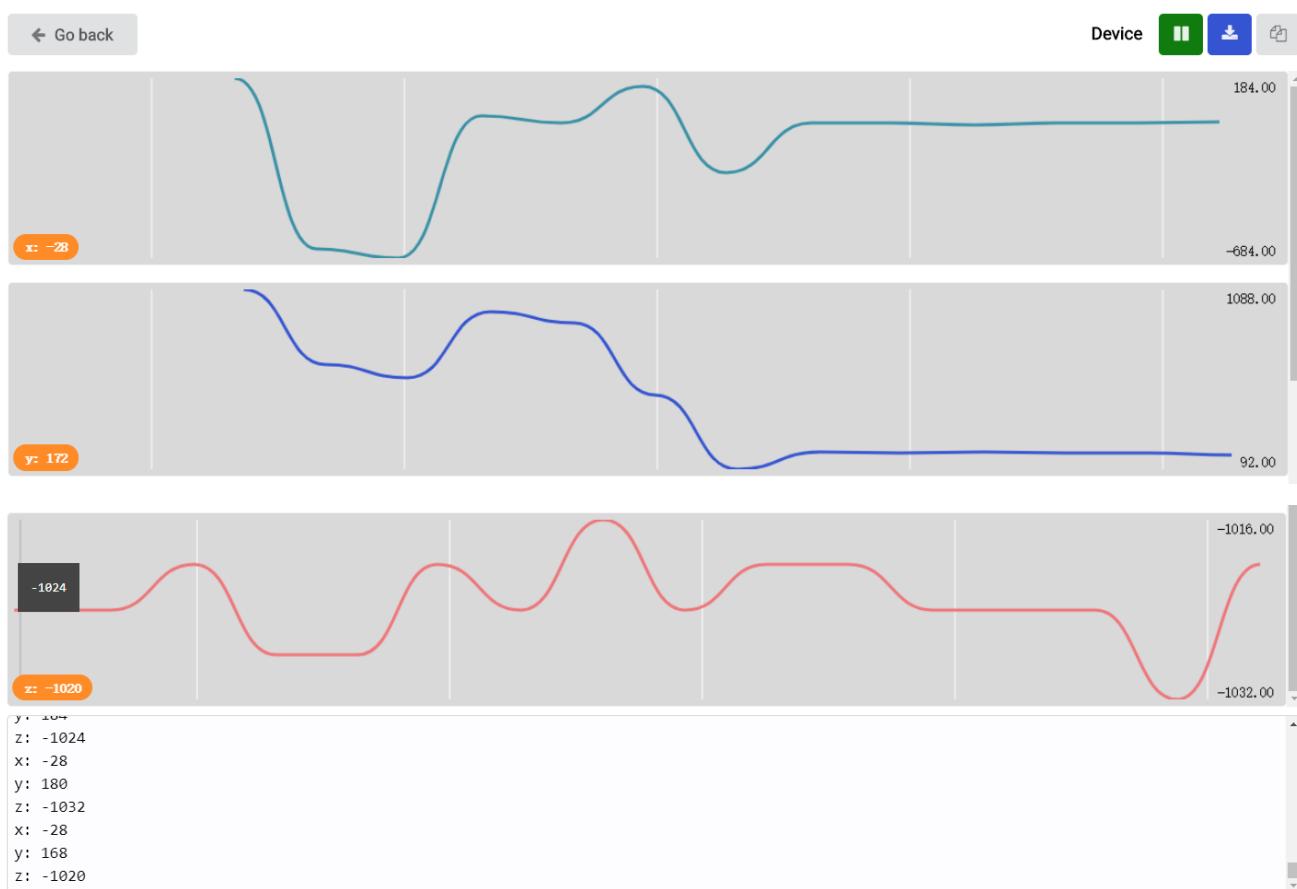
([How to import project](#))

File type	Path	File name
HEX file	../Projects/BlockCode/12.1_DisplayAccelerometerData	DisplayAccelerometerData.hex

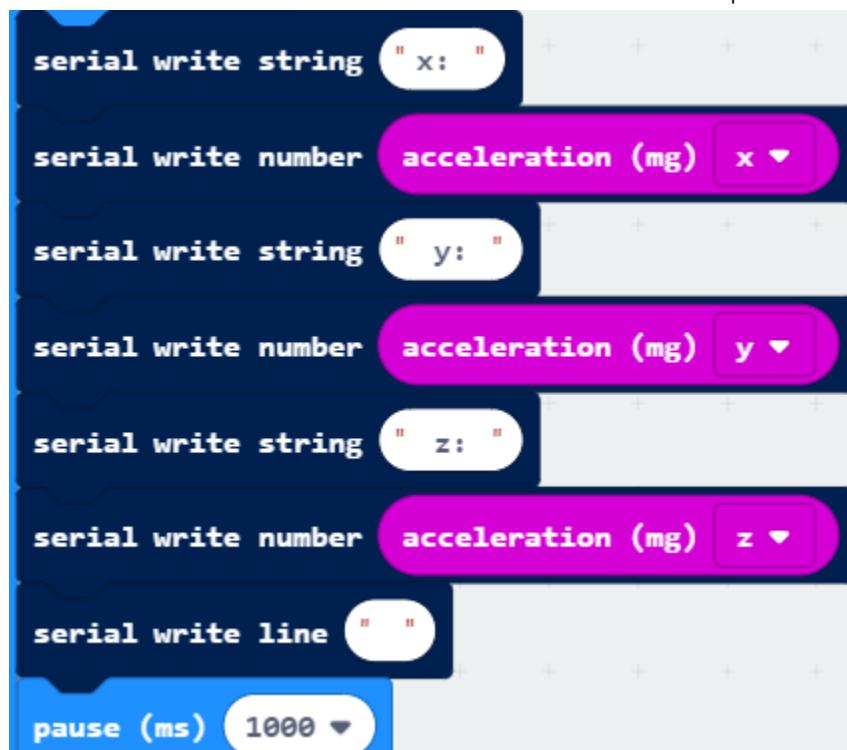
After import successfully, the code is shown as below:



Check the connection of the circuit, confirm the correct connection of the circuit, download the code into the micro:bit, and then open the serial console, you can see the data of the accelerometer, as shown below:



Read the value of the accelerometer in three directions and print it out through the serial port every second.



Reference

Block	Function
	Get the acceleration value in one of three dimensions, or the combined force in all directions (x, y, and z).

Python code

Open the .py file with Mu. Code path is as below:

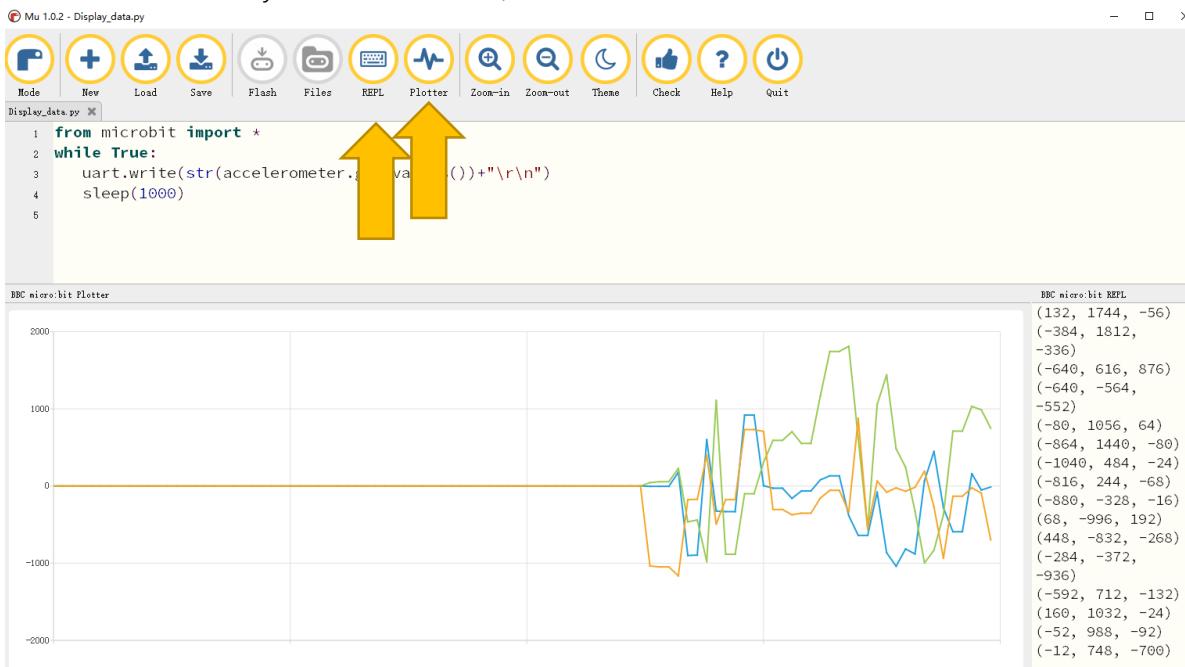
File type	Path	File name
Python file	./PythonCode/12.1_DisplayAccelerometerData	DisplayAccelerometerData.py

After load successfully, the code is shown as below:

```
from microbit import *
while True:
    uart.write(str(accelerometer.get_values())+"\r\n")
    sleep(1000)
```

Check the connection of the circuit and confirm that the circuit is connected correctly. Download the code into the micro:bit.

After the program is downloaded, open the plotter (Plotter), click on the REPL, you can see the x-axis, y-axis, z-axis data collected by the accelerometer, as shown below:



The following is the program code:

```
1 from microbit import *
2 while True:
3     uart.write(str(accelerometer.get_values())+"\r\n")
4     sleep(1000)
```

Every 1 second, the accelerometer data will be obtained and printed through the serial port.

```
1 uart.write(str(accelerometer.get_values())+"\r\n")
2 sleep(1000)
```

Reference

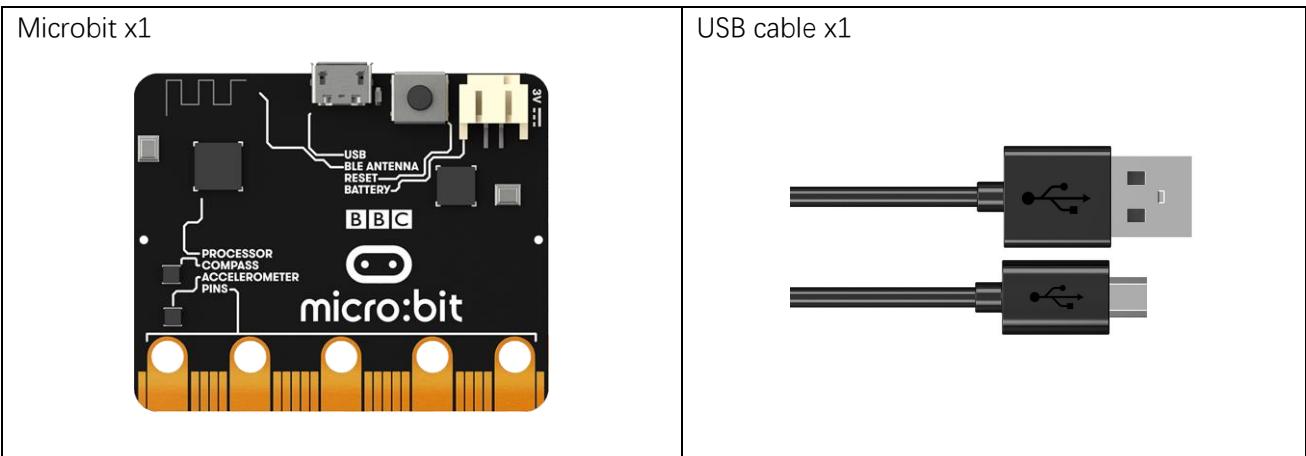
accelerometer.get_values()

Get the acceleration measurements in all axes at once, as a three-element tuple of integers ordered as X, Y, Z. By default the accelerometer is configured with a range of +/- 2g, and so X, Y, and Z will be within the range of +/-2000mg.

Project 12.2 Gradienter

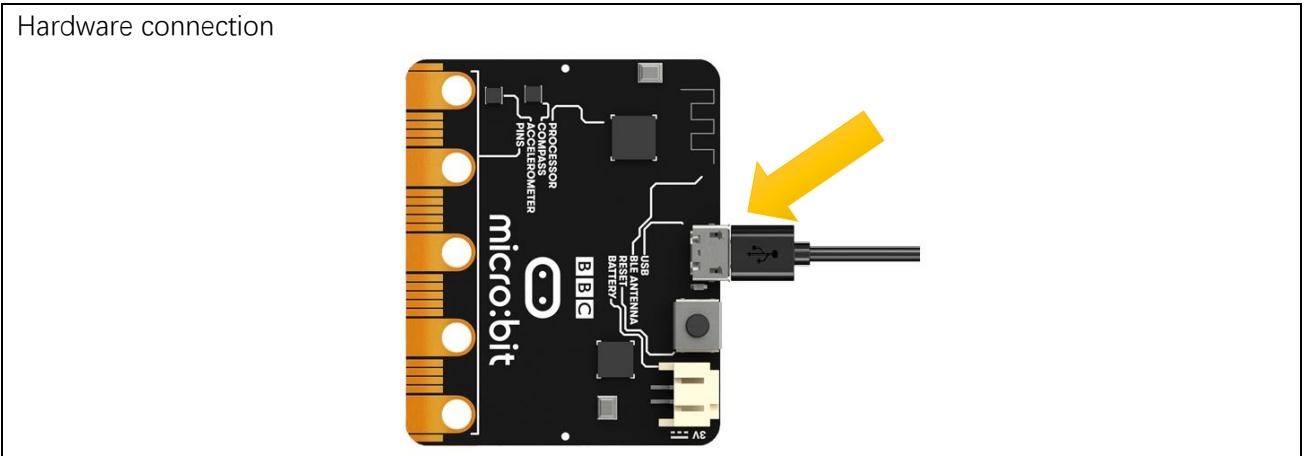
This project uses accelerometer to make a level instrument.

Component list



Circuit

Connect micro:bit and PC via micro USB cable.



Block code

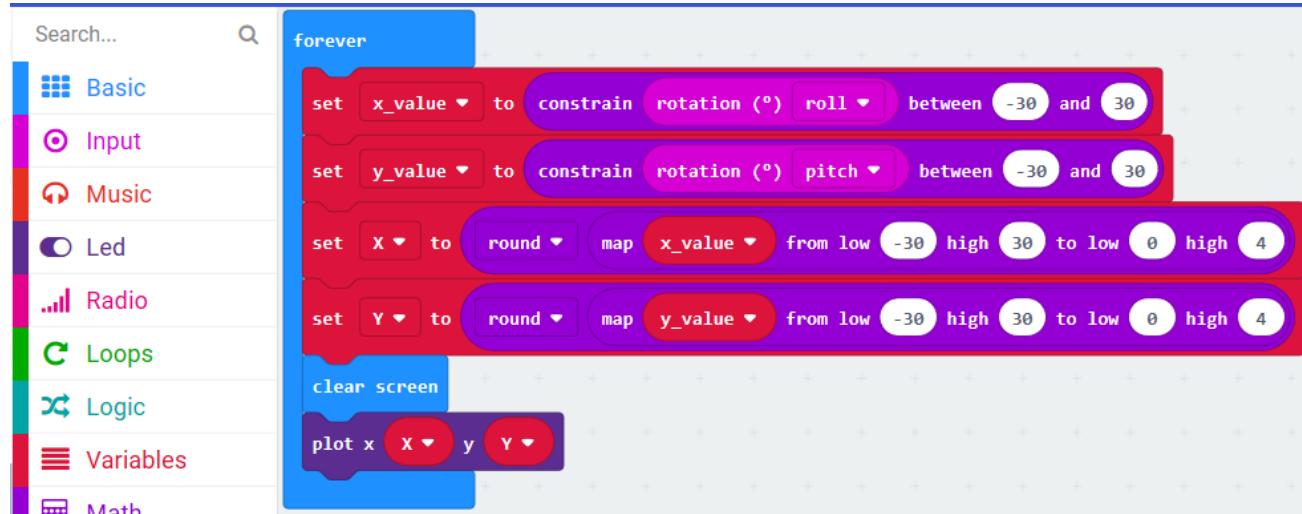
Open makecode first.

Import the .hex file. The path is as below:

([How to import project](#))

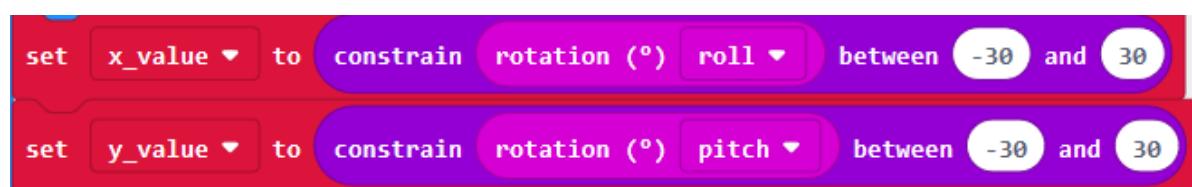
File type	Path	File name
HEX file	../Projects/BlockCode/12.2_Gradienter	Gradienter.hex

After import successfully, the code is shown as below:

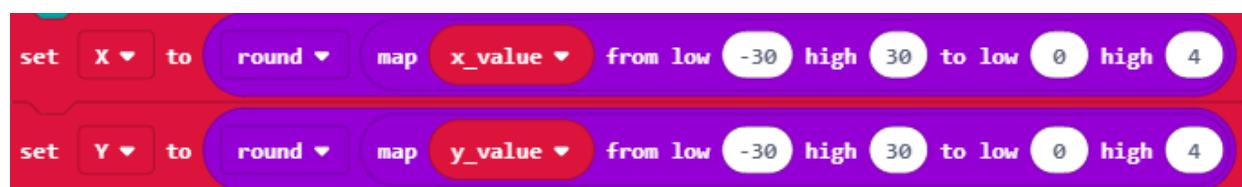


Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into micro:bit, you will observe that the LED dot matrix will change with the tilt of micro:bit.

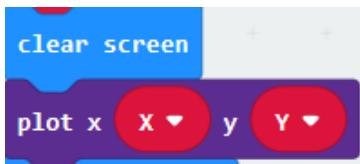
Detect the flip angle of the microbit in the x-axis and the y-axis. The return value ranges from -180 to 180 degrees. This project does not need to use such a wide range of flip angles, so we just set it within -30-30 degrees.



Since the LED screen is 5x5, map the range of -30-30 to the range of 0-4, and assign it to the X, Y variable.



Turn off all the LED first, then light the corresponding LED according to the value of the X, Y variables.



Reference

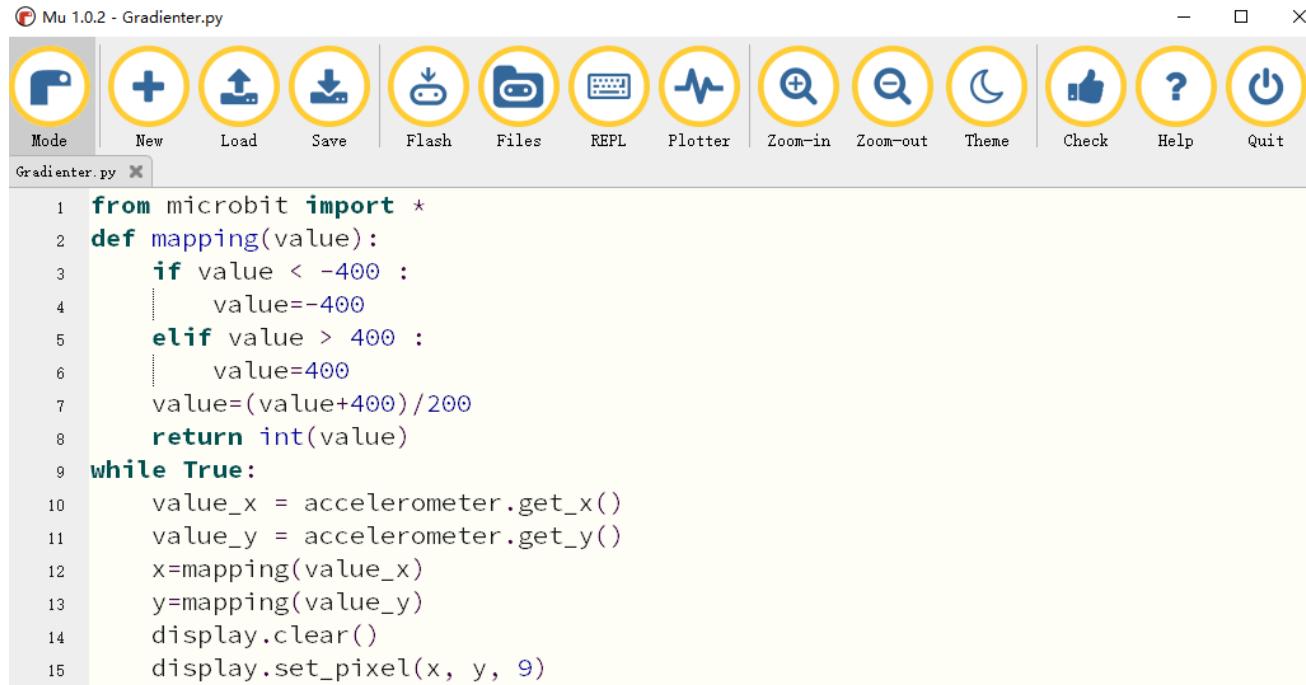
Block	Function
rotation (°) pitch ▾	Find how much the micro:bit is tilted in different directions.
plot x 0 y 0	Turn on the LED light you say on the LED screen.
round ▾ 0	If a number has a fractional part, you can make the number change to be the closest, next integer value
map 0 from low 0 high 1023 to low 0 high 4	A map is a conversion of one span of numbers to another.
clear screen	Turn off all the LED lights on the LED screen.
constrain 0 between 0 and 0	Make certain that the value of a number you give is no smaller and no bigger than two other numbers.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/12.2_Gradienter	Gradienter.py

After load successfully, the code is shown as below:



```

1 from microbit import *
2 def mapping(value):
3     if value < -400 :
4         value=-400
5     elif value > 400 :
6         value=400
7     value=(value+400)/200
8     return int(value)
9 while True:
10     value_x = accelerometer.get_x()
11     value_y = accelerometer.get_y()
12     x=mapping(value_x)
13     y=mapping(value_y)
14     display.clear()
15     display.set_pixel(x, y, 9)

```

Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into micro:bit, you will observe that the LED dot matrix will change with the tilt of micro:bit.

The following is the program code:

```

1 from microbit import *
2 def mapping(value):
3     if value < -400 :
4         value=-400
5     elif value > 400 :
6         value=400
7     value=(value+400)/200
8     return int(value)
9 while True:
10     value_x = accelerometer.get_x()
11     value_y = accelerometer.get_y()
12     x=mapping(value_x)
13     y=mapping(value_y)
14     display.clear()
15     display.set_pixel(x, y, 9)

```

A custom mapping() function that limits the input value to a range of -400 to 400 and maps to a range of 0-4.

```
def mapping(value):
    if value < -400 :
        value=-400
    elif value > 400 :
        value=400
    value=(value+400)/200
    return int(value)
```

Read the value of the accelerometer X, Y-axis direction. The return value range is -2000-2000. This project does not need to use such a large range. Only take -400-400. Call the mapping() function to return the value of 0-4 range , lighting the LED corresponding to the x row and the y column.

```
while True:
    value_x = accelerometer.get_x()
    value_y = accelerometer.get_y()
    x=mapping(value_x)
    y=mapping(value_y)
    display.clear()
    display.set_pixel(x, y, 9)
```

Reference

display.clear()

Set the brightness of all LEDs to 0 (off).

display.set_pixel(x, y, 9)

Set the brightness of the LED at column x and row y to value, which has to be an integer between 0 and 9.

accelerometer.get_x()

Get the acceleration measurement in the x axis, as a positive or negative integer, depending on the direction. The measurement is given in milli-g. By default the accelerometer is configured with a range of +/- 2g, and so this method will return within the range of +/- 2000mg

accelerometer.get_y()

Get the acceleration measurement in the y axis, as a positive or negative integer, depending on the direction. The measurement is given in milli-g. By default the accelerometer is configured with a range of +/- 2g, and so this method will return within the range of +/- 2000mg.

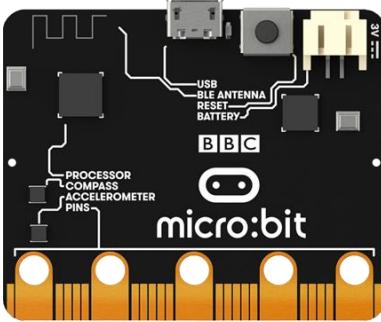
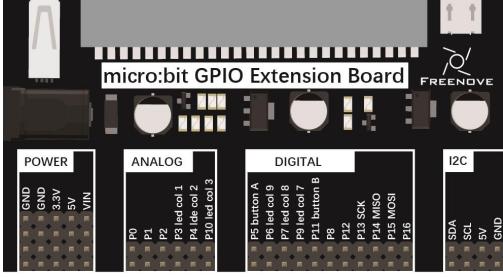
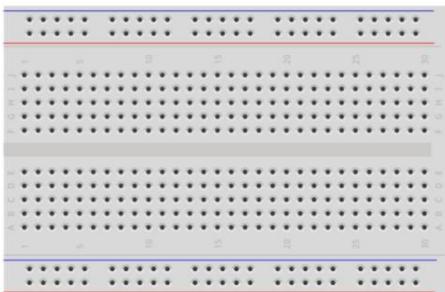
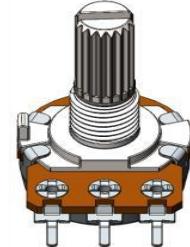
Chapter 13 Potentiometer

This chapter learns a new component: potentiometer

Project 13.1 Potentiometer

This project enables rotating potentiometer to output different voltages.

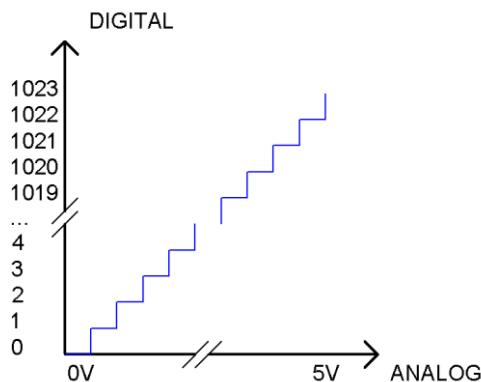
Component list

Microbit x1	Expansion board x1
	
Breakboard x1	Potentiometer x1
	
USB cable x1	F/M x3

Component knowledge

ADC

ADC, Analog-to-Digital Converter, is a device used to convert analog to digital. microbit has a 10 bits ADC, that means the resolution is $2^{10}=1024$, and the range (here is 3.3V) will be divided equally into 1024 parts. The analog of each section corresponds to one ADC value. So the more bits ADC has, the denser the partition of analog will be, also the higher precision of the conversion will be.



Subsection 1: the analog in rang of 0V-3.3/1024V corresponds to digital 0;

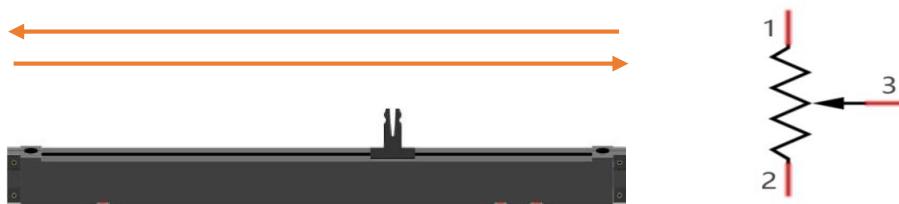
Subsection 2: the analog in rang of 3.3 /1024V-2*3.3/1024V corresponds to digital 1;

...

The following analog will be divided accordingly.

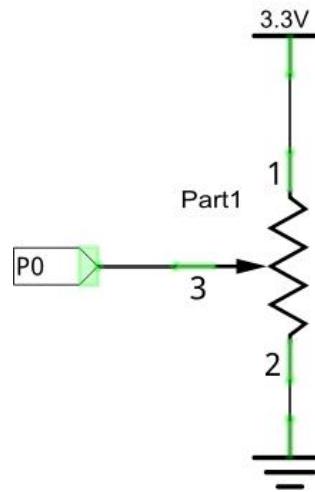
Potentiometer

Potentiometer is a resistive component with three terminal parts and its resistance can be adjusted in accordance with according to a certain change rule. Potentiometer is often made up by resistance and removable brush. When the brush moves along the resistor body, there will be resistance or voltage that has a certain relationship with displacement on the output side (3). Figure shown below is the linear sliding potentiometer and its symbol.



Pin 1 and pin 2 of the potentiometer is connected to two ends of a resistor body respectively, and pins 3 is connected to a brush. When the brush moves from pin 1 to pin 2, resistance between pin 1 and pin 3 will increase up to max resistance of the resistor body linearly, and resistance between pin 2 and pin 3 will decrease down to 0 linearly.

In the circuit, the two sides of resistor body are often connected to the positive and negative electrodes of a power respectively. When you slide the brush of pin 3, you can get a certain voltage in the range of negative voltage to positive voltage of the power supply.



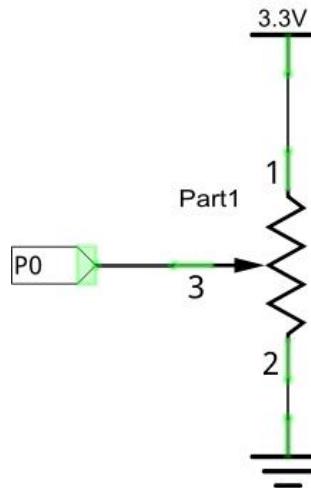
Rotary potentiometer

Rotary potentiometer and linear potentiometer have similar function; the only difference is: the resistance is adjusted through rotating the potentiometer.

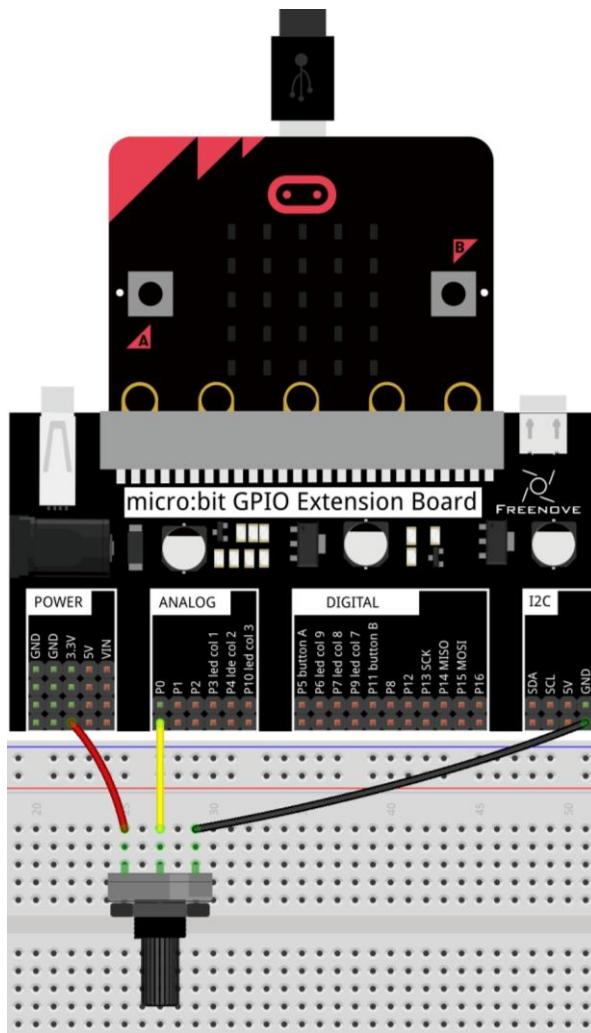


Circuit

Schematic diagram



Hardware connection



Block code

Open makecode first. Import the .hex file. The path is as below:

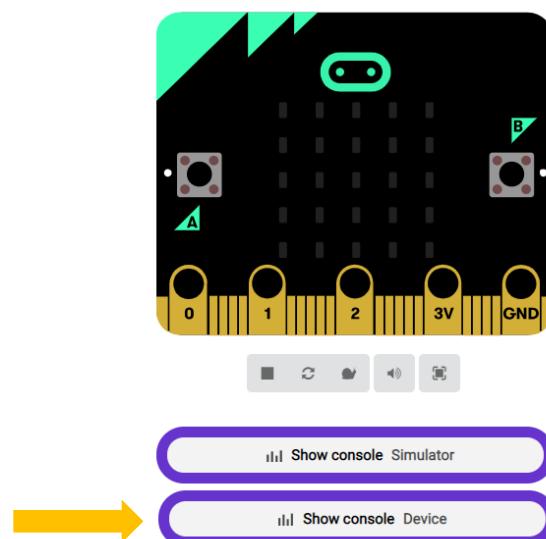
([How to import project](#))

File type	Path	File name
HEX file	./Projects/BlockCode/13.1_Potentiometer	Potentiometer.hex

After import successfully, the code is shown as below:



Check the connection of the circuit and confirm that the circuit is connected correctly. Download the code into micro:bit.

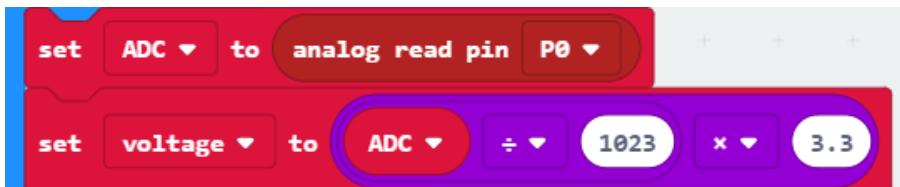


Click on the console device, rotate the potentiometer, you will see the output ADC value and voltage value.

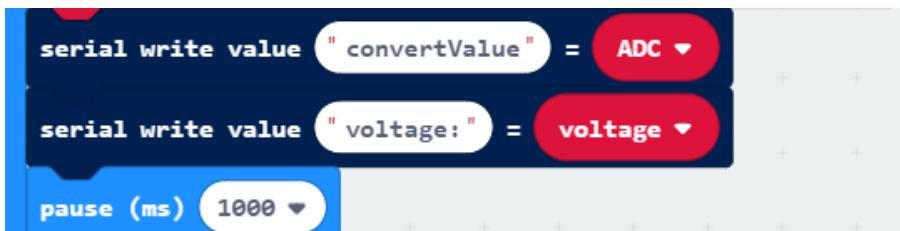
```

convertValue:0
voltage::2.6129032258064515
convertValue:1022
voltage::3.2967741935483867
convertValue:793
voltage::2.5580645161290318
convertValue:731
voltage::2.3580645161290321
  
```

Read the analog voltage value of the P0 pin, the range is 0-1023, and then convert the analog voltage value into a digital voltage value.



Print the analog voltage and digital voltage of P0 pin every 1 second.



Reference

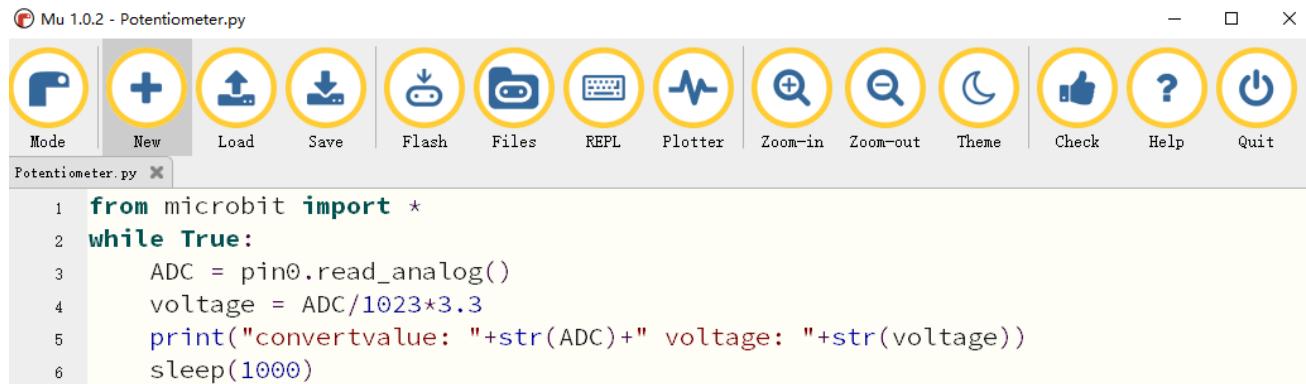
Block	Function
analog read pin P0	Read an analog signal (0 through 1023) from the pin you say.)

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	./Projects/PythonCode/13.1_Potentiometer	Potentiometer.py

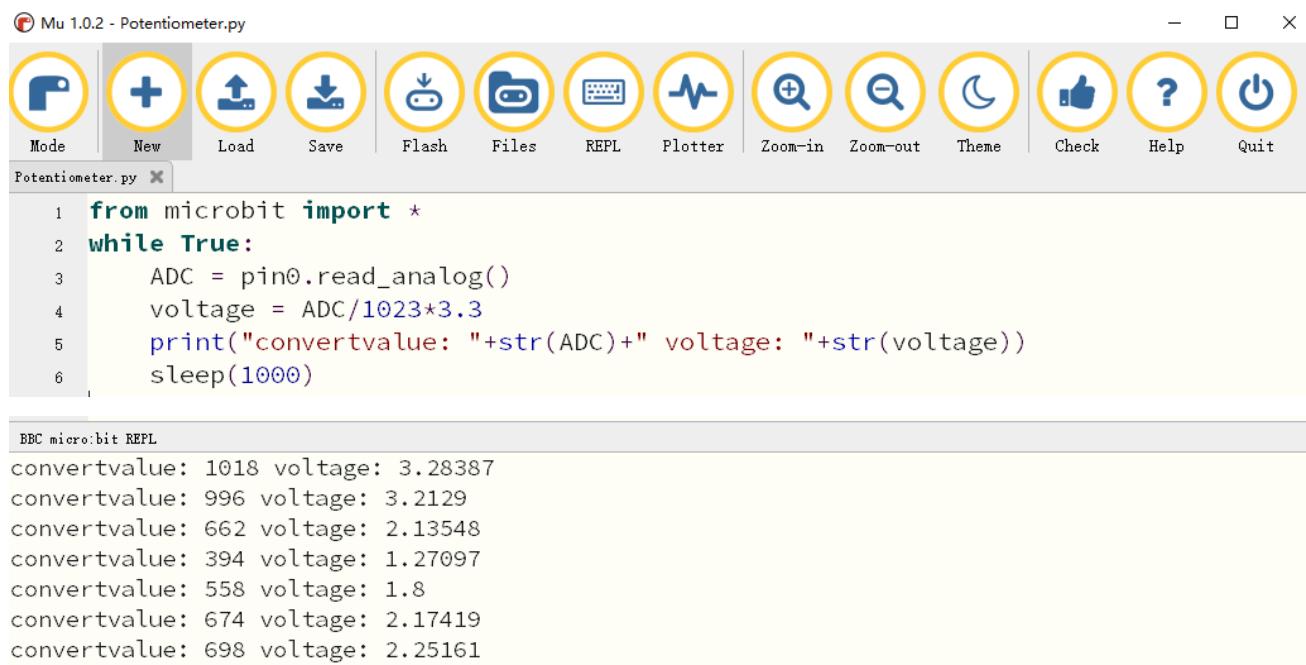
After load successfully, the code is shown as below:



```
from microbit import *
while True:
    ADC = pin0.read_analog()
    voltage = ADC/1023*3.3
    print("convertvalue: "+str(ADC)+" voltage: "+str(voltage))
    sleep(1000)
```

Check the connection of the circuit and confirm that the circuit is connected correctly. Download the code into the micro:bit. (How to download?)

Click on the REPL, press the micro:bit reset button, and then rotate the potentiometer. You can see the change in the value on the software Mu, as shown below.



```
from microbit import *
while True:
    ADC = pin0.read_analog()
    voltage = ADC/1023*3.3
    print("convertvalue: "+str(ADC)+" voltage: "+str(voltage))
    sleep(1000)
```

BBC micro:bit REPL

```
convertvalue: 1018 voltage: 3.28387
convertvalue: 996 voltage: 3.2129
convertvalue: 662 voltage: 2.13548
convertvalue: 394 voltage: 1.27097
convertvalue: 558 voltage: 1.8
convertvalue: 674 voltage: 2.17419
convertvalue: 698 voltage: 2.25161
```

The following is the program code:

```
1 from microbit import *
2 while True:
3     ADC = pin0.read_analog()
4     voltage = ADC/1023*3.3
5     print("convertvalue: "+str(ADC)+" voltage: "+str(voltage))
6     sleep(1000)
```

Read the analog voltage value of the P0 pin, the range is 0-1023, and then convert the analog voltage value into a digital voltage value.

```
ADC = pin0.read_analog()
voltage = ADC/1023*3.3
```

Print the analog voltage and digital voltage of P0 pin every 1 second.

```
print("convertvalue: "+str(ADC)+" voltage: "+str(voltage))
sleep(1000)
```

Reference

read_analog()

Read an analog signal (0 through 1023) from the pin you say.

print()

Print() is a Python built-in function for printing.

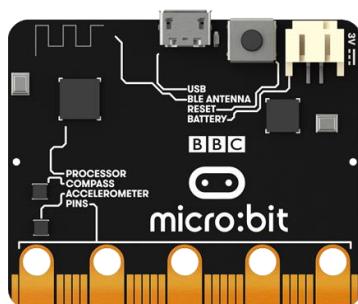
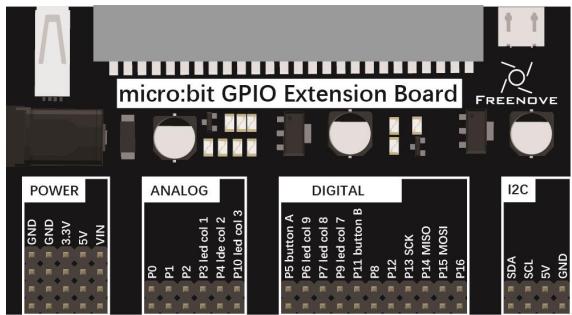
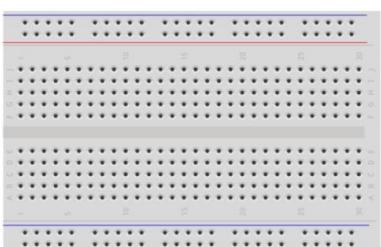
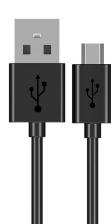
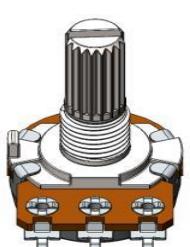
Chapter 14 Potentiometer And LED

This chapter is a comprehensive application of potentiometer and LED.

Project 14.1 Soft Light

This project makes a LED with adjustable brightness.

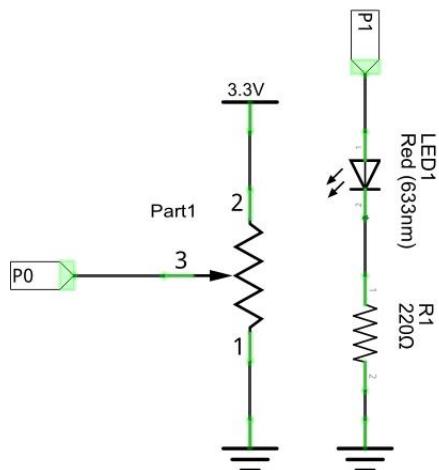
Component list

Microbit x1	Expansion board x1	
		
Breakboard x1	USB cable x1	F/M x4 M/M x1
		
Led x1	Resistor 220Ω x1	Potentiometer x1
		

Circuit

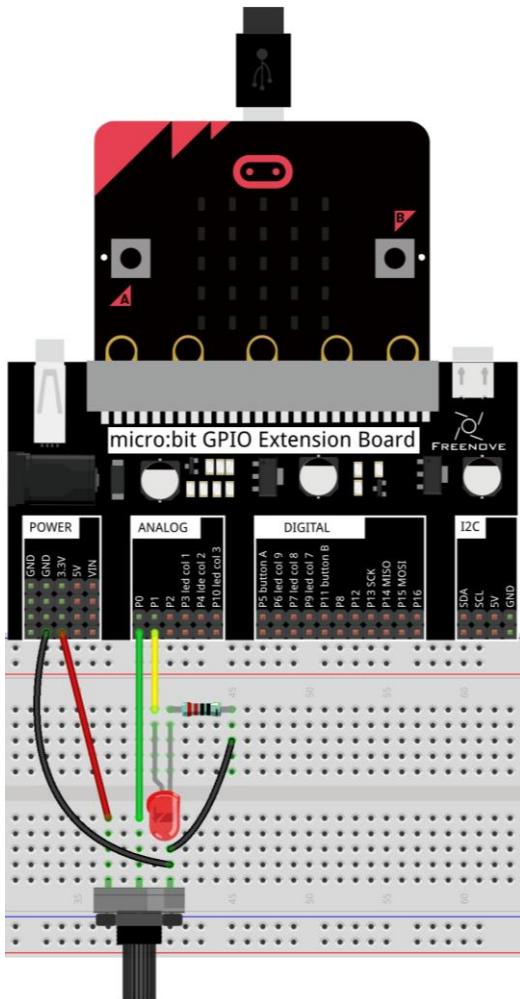
In this circuit, the port 1 and 2 of the potentiometer are respectively connected to the two ends of the power supply, and the 3 port is connected to the P0 pin of the micro:bit.

Schematic diagram



Hardware connection

P1 pin is connected to LED long pin (positive), and short pin (negative) is connected to resistor.



Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file	./Projects/BlockCode/14.1_SoftLight	SoftLight.hex

After import successfully, the code is shown as below:



Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into the micro:bit, and turn the potentiometer to see that the brightness of the LED is changing.

Read the analog voltage value of the P0 pin, then the P1 pin outputs the same analog voltage value.



Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/14.1_SoftLight	SoftLight.py

After load successfully, the code is shown as below:



Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into the micro:bit, and turn the potentiometer to see that the brightness of the LED is changing.

The following is the program code:

```

1 from microbit import *
2 while True:
3     pin1.write_analog(pin0.read_analog())

```

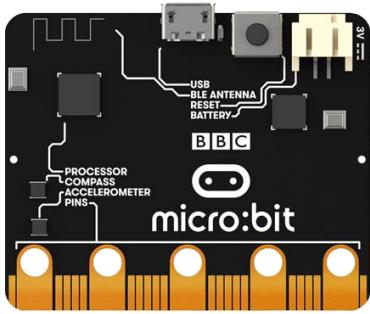
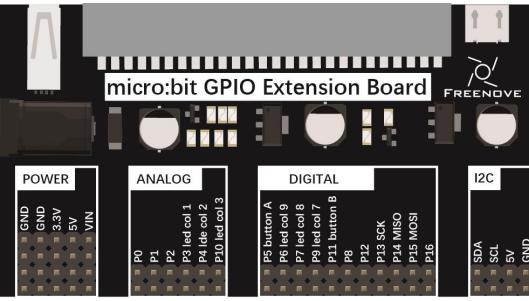
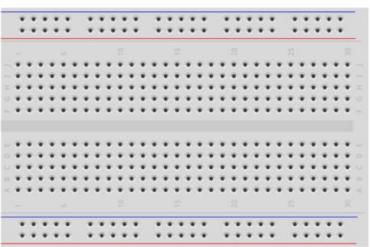
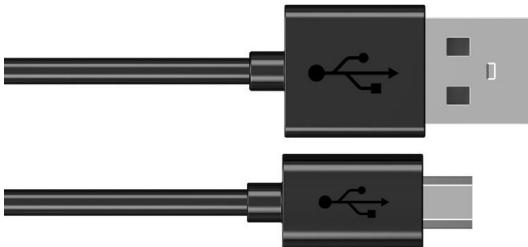
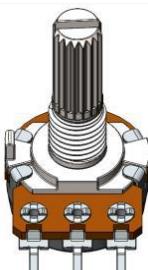
Read the analog voltage value of the P0 pin, then the P1 pin outputs the same analog voltage value.

```
pin1.write_analog(pin0.read_analog())
```

Project 14.2 Colorful Soft Light

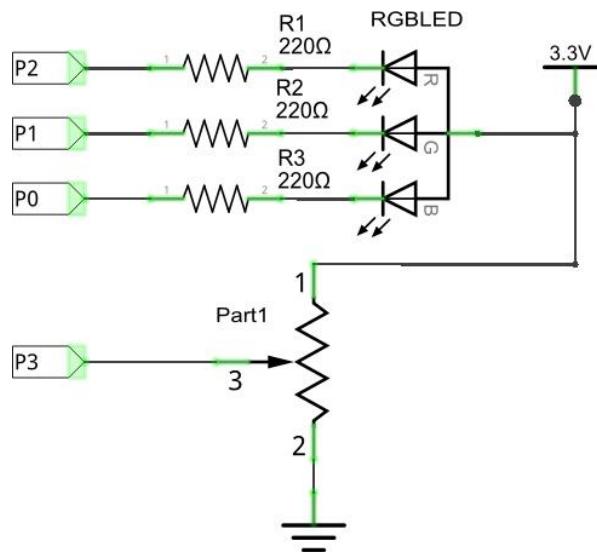
This project controls the color of the RGBLED through a potentiometer.

Component list

Microbit x1	Expansion board x1
	
Breakboard x1	USB cable x1
	
Rotary potentiometer x1	RGB_LED x1
	
F/M x6 M/M x1	Resistor 220Ω x3
	

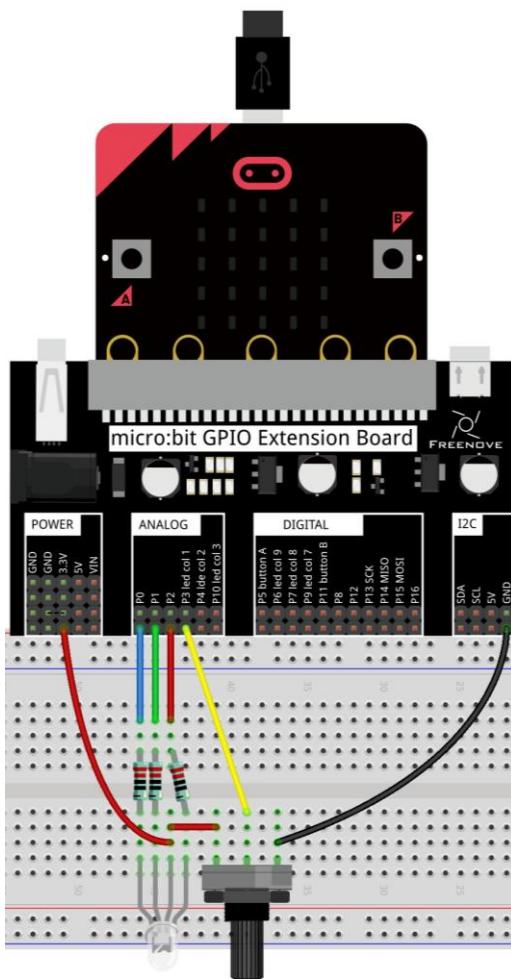
Circuit

Schematic diagram



Hardware connection

RGBLED long pin (anode) connected to 3.3V power supply.



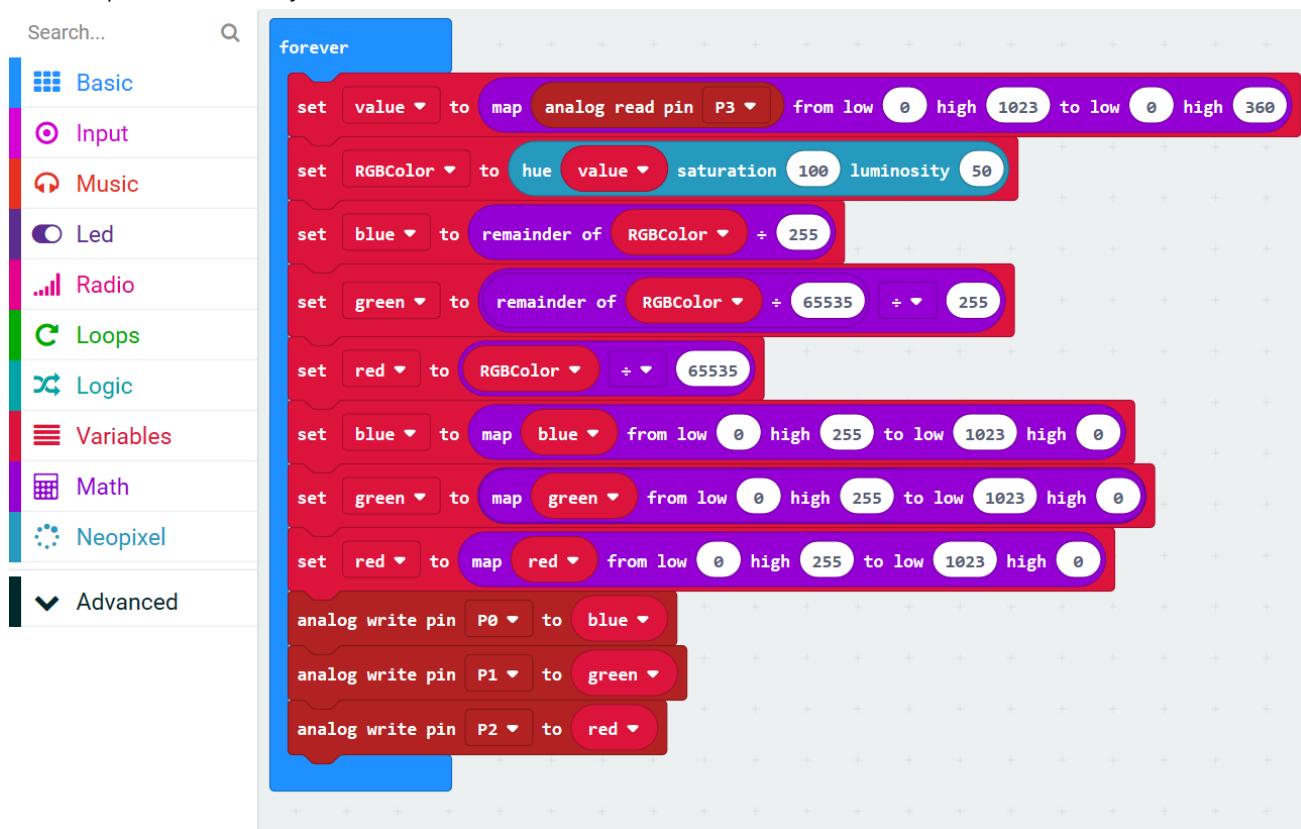
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file	./Projects/BlockCode/14.2_ColorfulSoftLight	ColorfulSoftLight.hex

After import successfully, the code is shown as below:



Download the code into micro:bit, rotate the potentiometer, you can see that the color of the RGBLED is changing.

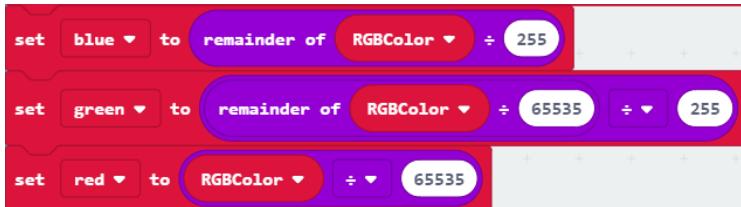
Read the potentiometer's analog voltage and map the potentiometer's analog voltage range 0-1023 to the hue angle range 0-360.



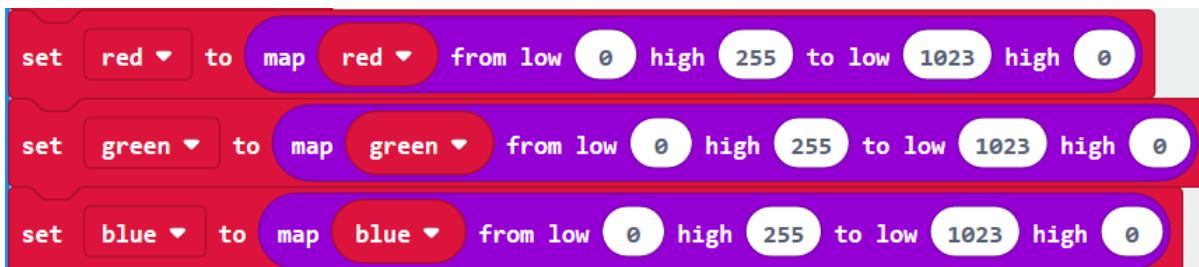
Converts the HSL color system to the RGB color system, returns the RGB value corresponding to the current hue angle, and stores the value in the variable RGBColor.



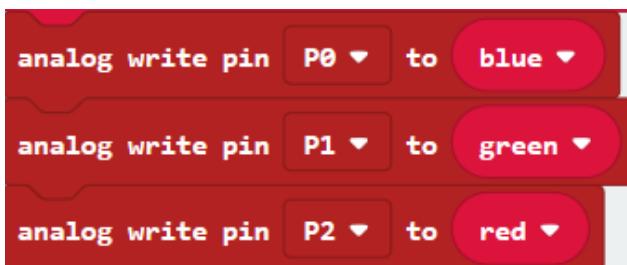
The value of the lower eight-bit blue channel of the variable RGBColor is assigned to the variable blue, the value of the middle eight-bit green channel is assigned to the variable green, and the value of the upper eight-bit red channel is assigned to the variable red.



RGBLED is a common anode, so the pins are set to low to turn on the RGBLED. The values of the variables 'red', 'green', and 'blue' are converted from 0-255 to analog signal values in the range of 1023-0, and then reassigned to 'red', 'green', 'blue' variable.



Write the analog voltage value of the red, green, and blue variables to the corresponding P0, P1, and P2 pins to change the LED color.



Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/14.2_ColorfulSoftLight	ColorfulSoftLight.py

After load successfully, the code is shown as below:

```

1  from microbit import *
2  display.off()
3  def map(value,fromLow,fromHigh,toLow,toHigh):
4      return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
5  def HSL_RGB(degree):
6      degree=degree/360*255
7      if degree < 85:
8          red = 255 - degree * 3
9          green = degree * 3
10         blue = 0
11     elif degree < 170:
12         degree = degree - 85
13         red = 0
14         green = 255 - degree * 3
15         blue = degree * 3
16     else:
17         degree = degree - 170
18         red = degree * 3
19         green = 0
20         blue = 255 - degree * 3
21     return red,green,blue
22 while True:
23     value=map(pin3.read_analog(),0,1023,0,360)
24     red,green,blue=HSL_RGB(value)
25     red=map(red,0,255,1023,0)
26     green=map(green,0,255,1023,0)
27     blue=map(blue,0,255,1023,0)
28     pin2.write_analog(red)
29     pin1.write_analog(green)
30     pin0.write_analog(blue)

```

After checking the connection of the circuit and confirming the correct connection of the circuit, the code is downloaded into micro:bit. By rotating the potentiometer, you can see that the color of RGB LED is changing.

The following is the program code:

```

1  from microbit import *
2  display.off()
3  def map(value, fromLow, fromHigh, toLow, toHigh):
4      return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
5  def HSL_RGB(degree):
6      degree=degree/360*255
7      if degree < 85:
8          red = 255 - degree * 3
9          green = degree * 3
10         blue = 0
11     elif degree < 170:
12         degree = degree - 85
13         red = 0
14         green = 255 - degree * 3
15         blue = degree * 3
16     else:
17         degree = degree - 170
18         red = degree * 3
19         green = 0
20         blue = 255 - degree * 3
21     return red,green,blue
22 while True:
23     value=map(pin3.read_analog(),0,1023,0,360)
24     red,green,blue=HSL_RGB(value)
25     print(red,green,blue)
26     red=map(red,0,255,1023,0)
27     green=map(green,0,255,1023,0)
28     blue=map(blue,0,255,1023,0)
29     pin2.write_analog(red)
30     pin1.write_analog(green)
31     pin0.write_analog(blue)
32     sleep(10)

```

Turn off the LED screen to use the P3 pin. A custom map() function that converts values in one range of numbers to values in another range of numbers.

```

display.off()
def map(value, fromLow, fromHigh, toLow, toHigh):
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow

```

The custom HSL_RGB() function is used to convert the HSL color system to the RGB color system and return the RGB value corresponding to the current hue angle.

```
def HSL_RGB(degree):
    degree=degree/360*255
    if degree < 85:
        red = 255 - degree * 3
        green = degree * 3
        blue = 0
    elif degree < 170:
        degree = degree - 85
        red = 0
        green = 255 - degree * 3
        blue = degree * 3
    else:
        degree = degree - 170
        red = degree * 3
        green = 0
        blue = 255 - degree * 3
    return red, green, blue
```

Read the analog voltage value of the P3 pin and convert it to the corresponding hue angle. Call the HSL_RGB() function to return the RGB value corresponding to the current hue angle, and then write the corresponding RGB values to the P0, P1, and P2 pins to change the LED color.

```
while True:
    value=map(pin3.read_analog(),0,1023,0,360)
    red,green,blue=HSL_RGB(value)
    print(red,green,blue)
    red=map(red,0,255,1023,0)
    green=map(green,0,255,1023,0)
    blue=map(blue,0,255,1023,0)
    pin2.write_analog(red)
    pin1.write_analog(green)
    pin0.write_analog(blue)
    sleep(10)
```

Project 14.3 Rainbow Light

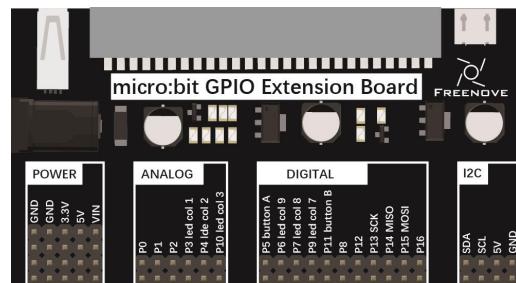
This project uses a potentiometer to control the RGB LED module.

Component list

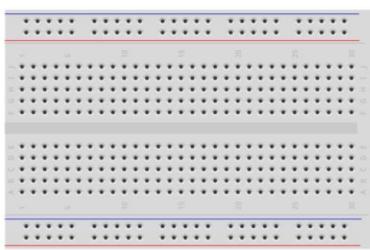
Microbit x1



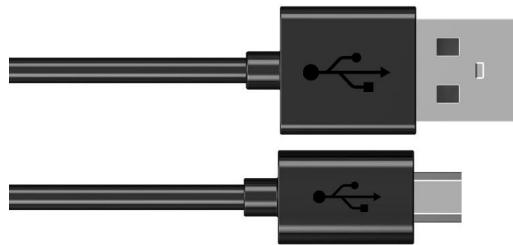
Expansion board x1



Breakboard x1



USB cable x2



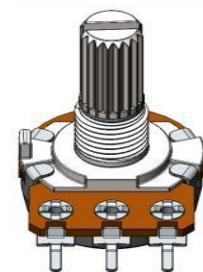
Freenove 8 RGB LED Module x1



F/M x3 F/F x3

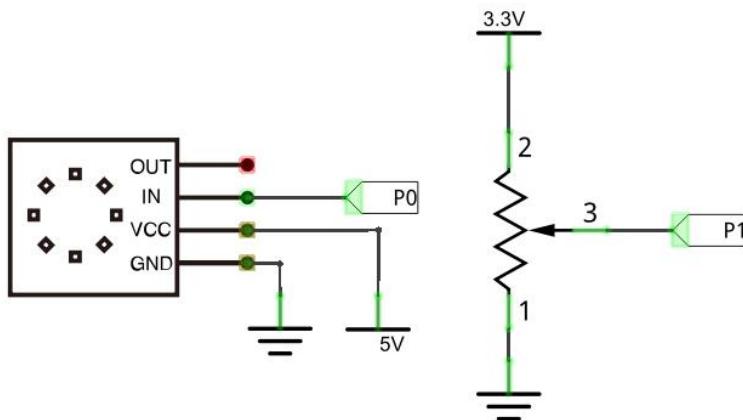


Potentiometer x1

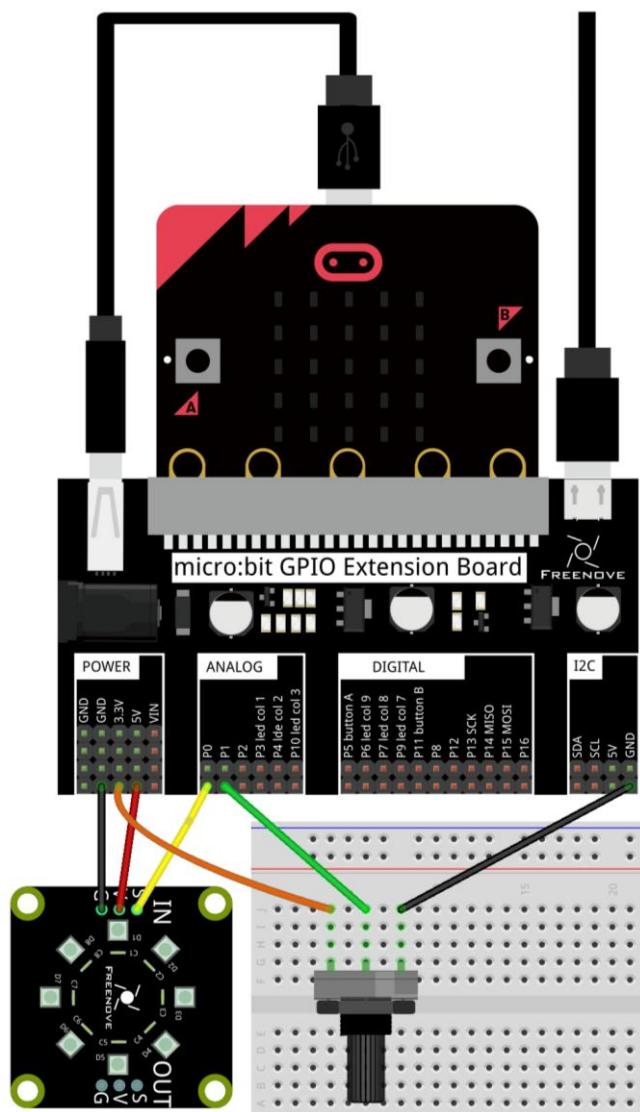


Circuit

Schematic diagram



Hardware connection



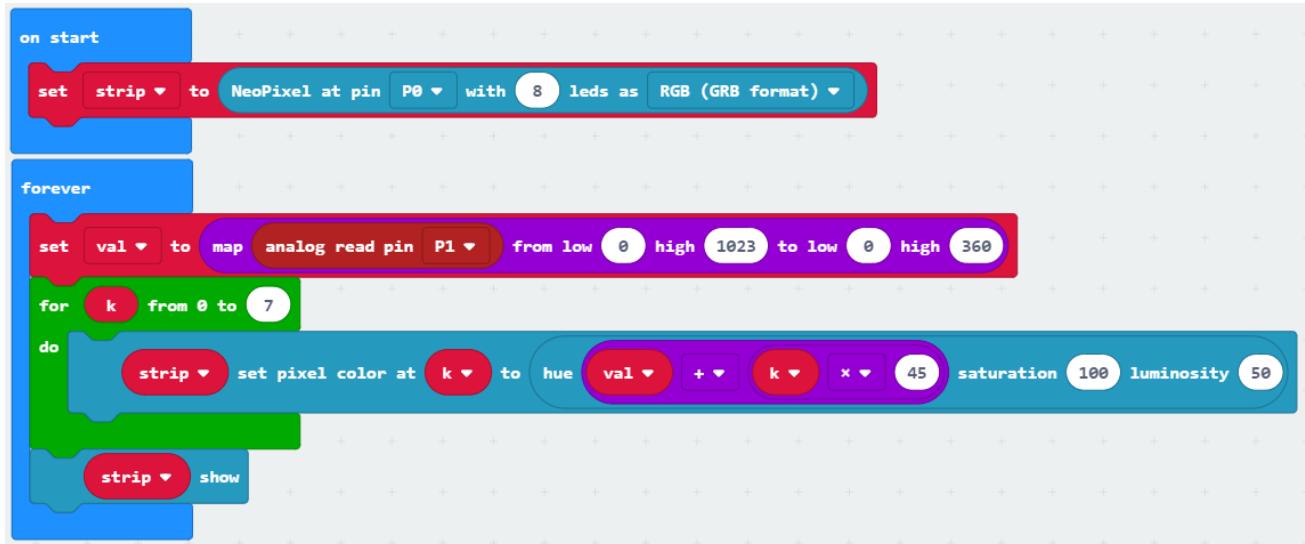
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file	./Projects/BlockCode/14.3_RainbowLight	RainbowLight.hex

After import successfully, the code is shown as below:



Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into the micro:bit, rotate the potentiometer, and the color ring of the RGB LED module also rotates.

Set the number of pins and LEDs for the RGB LED module, as well as the type of LED.



Read the voltage of the potentiometer and map the analog value of 0-1023 to an angle of 0-360.



In the for loop, the hue difference between each two LED is 45. When the hue changes, it ensures that the led inherits the hue of the previous led. Then convert the HSL color system to the RGB color system, and returns the RGB value corresponding to the angle, to make the LED to achieve the effect of the rainbow.

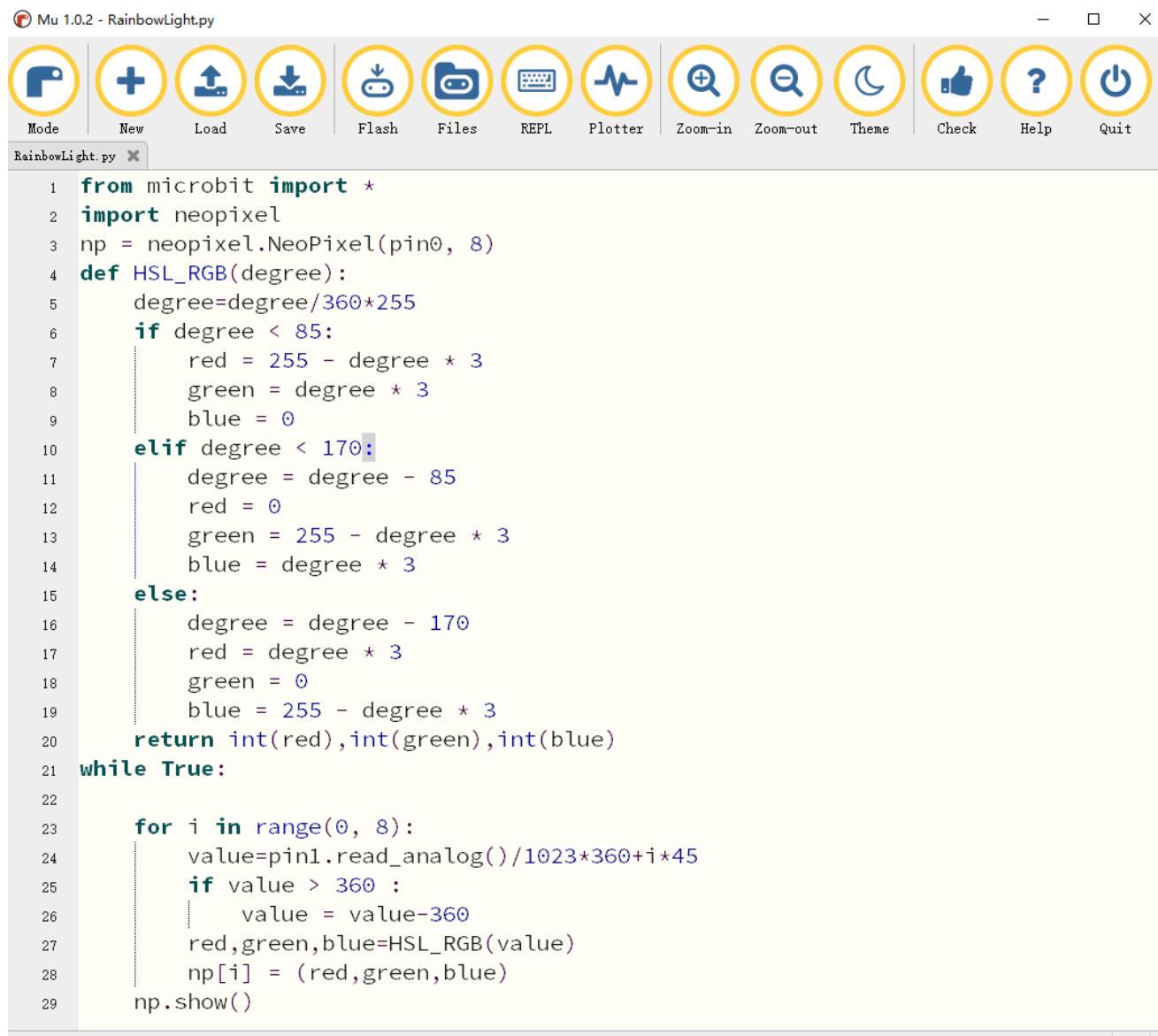


Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/14.3_RainbowLight	RainbowLight.py

After load successfully, the code is shown as below:



```

1  from microbit import *
2  import neopixel
3  np = neopixel.NeoPixel(pin0, 8)
4  def HSL_RGB(degree):
5      degree=degree/360*255
6      if degree < 85:
7          red = 255 - degree * 3
8          green = degree * 3
9          blue = 0
10     elif degree < 170:
11         degree = degree - 85
12         red = 0
13         green = 255 - degree * 3
14         blue = degree * 3
15     else:
16         degree = degree - 170
17         red = degree * 3
18         green = 0
19         blue = 255 - degree * 3
20     return int(red),int(green),int(blue)
21 while True:
22
23     for i in range(0, 8):
24         value=pin1.read_analog()/1023*360+i*45
25         if value > 360 :
26             value = value-360
27         red,green,blue=HSL_RGB(value)
28         np[i] = (red,green,blue)
29     np.show()

```

Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into the micro:bit. Rotate the potentiometer, and then the color ring of the RGB LED module also rotates.

The following is the program code:

```

1  from microbit import *
2  import neopixel
3  np = neopixel.NeoPixel(pin0, 8)
4  def HSL_RGB(degree):
5      degree=degree/360*255
6      if degree < 85:
7          red = 255 - degree * 3
8          green = degree * 3
9          blue = 0
10     elif degree < 170:
11         degree = degree - 85
12         red = 0
13         green = 255 - degree * 3
14         blue = degree * 3
15     else:
16         degree = degree - 170
17         red = degree * 3
18         green = 0
19         blue = 255 - degree * 3
20     return int(red), int(green), int(blue)
21 while True:
22     for i in range(0, 8):
23         value=pin1.read_analog()/1023*360+i*45
24         if value > 360 :
25             value = value-360
26         red, green, blue=HSL_RGB(value)
27         np[i] = (red, green, blue)
28     np.show()

```

Set the number of pins and led for control the RGB LED module.

```
np = neopixel.NeoPixel(pin0, 8)
```

Custom HSL_RGB() function is used to convert HSL color to RGB color, returning the RGB value corresponding to the current hue angle.

```

def HSL_RGB(degree):
    degree=degree/360*255
    if degree < 85:
        red = 255 - degree * 3
        green = degree * 3
        blue = 0
    elif degree < 170:
        degree = degree - 85
        red = 0

```

```
green = 255 - degree * 3
blue = degree * 3
else:
    degree = degree - 170
    red = degree * 3
    green = 0
    blue = 255 - degree * 3
return int(red), int(green), int(blue)
```

In the for loop, the analog voltage of the potentiometer is read and converted to the corresponding hue angle. The hue difference between each two led is 45. The HSL_RGB() function is called to convert the HSL color system to the RGB color system. Returns the RGB value corresponding to the current hue angle to make the LED achieve the effect of rainbow.

```
while True:
    for i in range(0, 8):
        value=pin1.read_analog()/1023*360+i*45
        if value > 360 :
            value = value-360
        red, green, blue=HSL_RGB(value)
        np[i] = (red, green, blue)
    np.show()
```



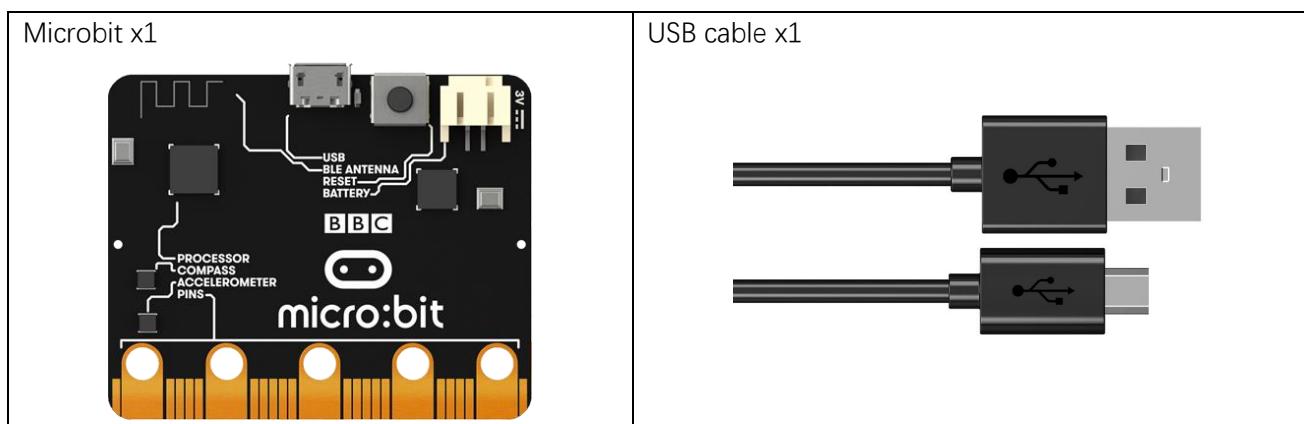
Chapter 15 Light Sensor

In this chapter, we will learn the micro:bit built-in light sensor and photoresistor.

Project 15.1 Built In Light Sensor

This project uses the micro:bit built-in light sensor to measure the brightness of light.

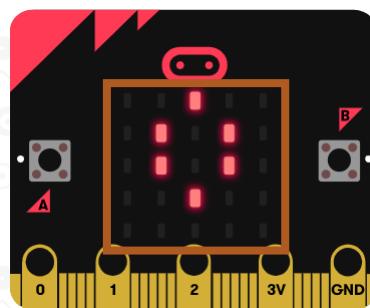
Component list



Component knowledge

Light sensor

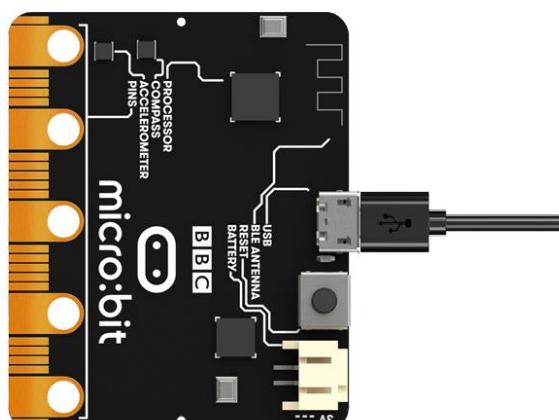
Micro:bit detects the ambient light intensity through the LED matrix. In forward bias mode, the LED screen works as display. In reverse bias mode, the LED screen works as a basic light sensor that can be used to detect ambient light.



Circuit

Connect micro:bit and PC via micro USB cable.

Hardware connection



Block code

Open makecode first. Import the .hex file. The path is as below:

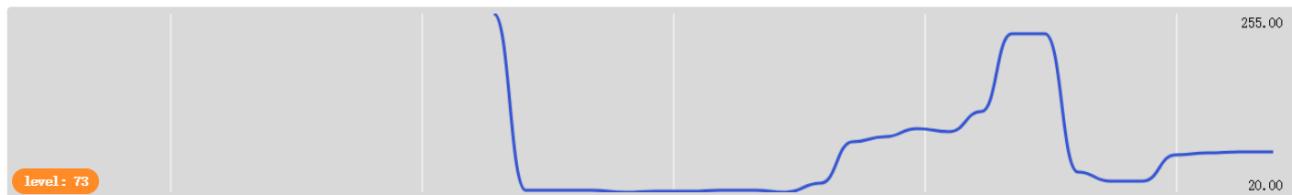
([How to import project](#))

File type	Path	File name
HEX file	./Projects/BlockCode/15.1_LightIntensityMeter	LightIntensityMeter.hex

After import successfully, the code is shown as below:



Check the connection of the circuit, confirm the correct connection of the circuit, download the code into the micro:bit. Open the serial port console, then you can see the read light intensity. Block the LED screen by hand or use the light source to illuminate the LED. You can see the change in value, the range of values is 0-255, 0 is dark, 255 is the brightest, as shown below.



```
level:228  
level:229  
level:46  
level:35  
level:34  
level:69  
level:72  
2 level:73
```

Reference

Block	Function
light level	Find the light level (how bright or dark it is) where you are. The light level 0 means darkness and 255 means bright light.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/15.1_LightIntensityMeter	LightIntensityMeter.py

After load successfully, the code is shown as below:

Mu 1.0.2 - LightIntensityMeter.py

The screenshot shows the Mu 1.0.2 interface with the title bar "Mu 1.0.2 - LightIntensityMeter.py". Below the title bar is a toolbar with icons for Mode, New, Load, Save, Flash, Files, REPL, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The main area displays the Python code:

```
1 from microbit import *
2 while True:
3     item = display.read_light_level()
4     print(item)
```

Check the connection of the circuit, confirm the correct connection of the circuit, download the code into the micro:bit. Open the serial port console, then you can see the read light intensity. Block the LED screen by hand or use the light source to illuminate the LED. You can see the change in value, the range of values is 0-255, 0 is dark, 255 is the brightest, as shown below.



The screenshot shows the Mu 1.0.2 Python editor interface. The top menu bar includes 'File', 'Edit', 'Run', 'Terminal', 'Help', and 'About'. Below the menu is a toolbar with icons for Mode, New, Load, Save, Flash, Files, REPL, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The main workspace shows a file named 'LightIntensityMeter.py' with the following code:

```
from microbit import *
while True:
    item = display.read_light_level()
    print(item)
```

Below the code is a BBC micro:bit REPL window displaying the output of the program, which is a series of '95' characters.

The following is the program code:

```
from microbit import *
while True:
    item = display.read_light_level()
    print (item)
```

Reference

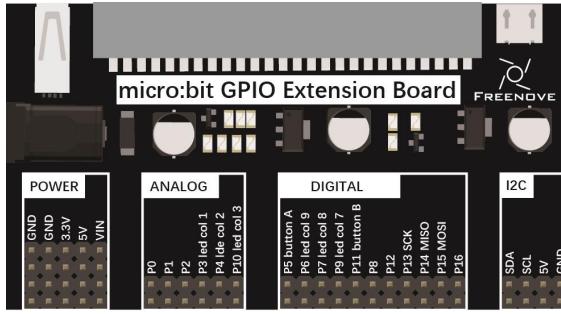
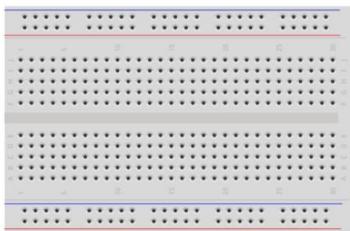
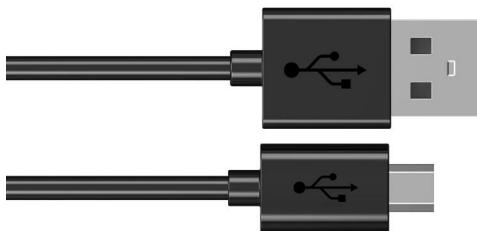
`display.read_light_level()`

Use the display's LEDs in reverse-bias mode to sense the amount of light falling on the display. Returns an integer between 0 and 255 representing the light level, with larger meaning more light.

Project 15.2 Night light

This project makes a night light.

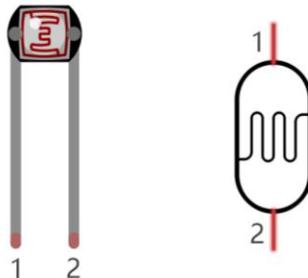
Component list

Microbit x1	Expansion board x1		
			
Breakboard x1	USB cable x1		
			
F/M x 4 M/M x1	Photoresistance x1	Led x1	Resistor 10kΩ x1
			
			Resistor 220Ω x1
			

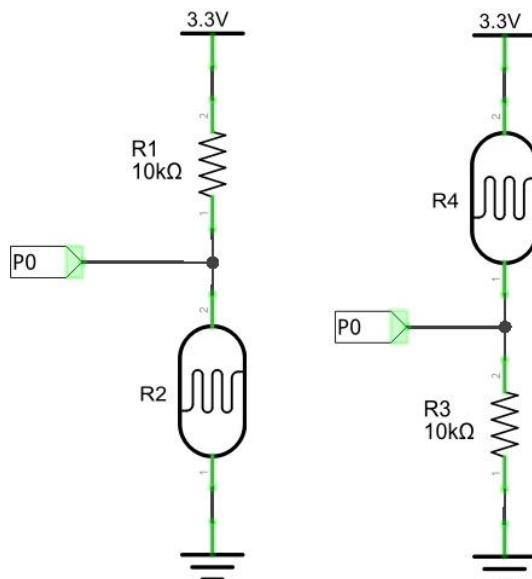
Component knowledge

Photoresistor

Photoresistor is a light sensitive resistor. When the strength that light casts onto the photoresistor surface is not the same, resistance of photoresistor will change. With this feature, we can use photoresistor to detect light intensity. Photoresistor and symbol are as follows.



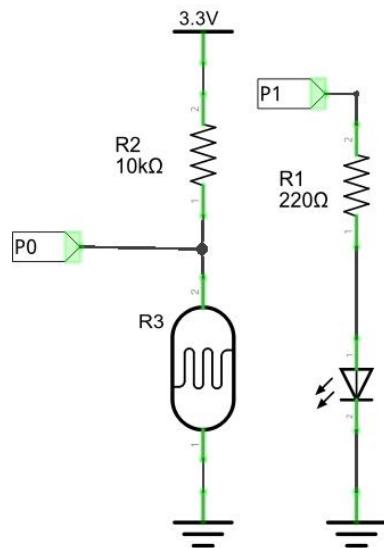
The circuit below is often used to detect the change of photoresistor resistance:



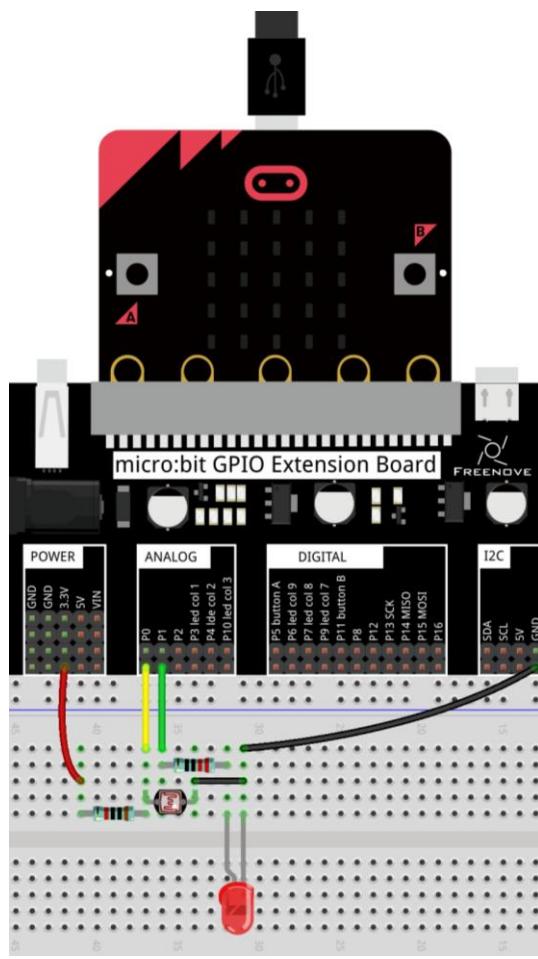
In the above circuit, when photoresistor resistance changes due to light intensity, voltage between photoresistor and resistor R1 will change, so light's intensity can be obtained by measuring the voltage.

Circuit

Schematic diagram



Hardware connection



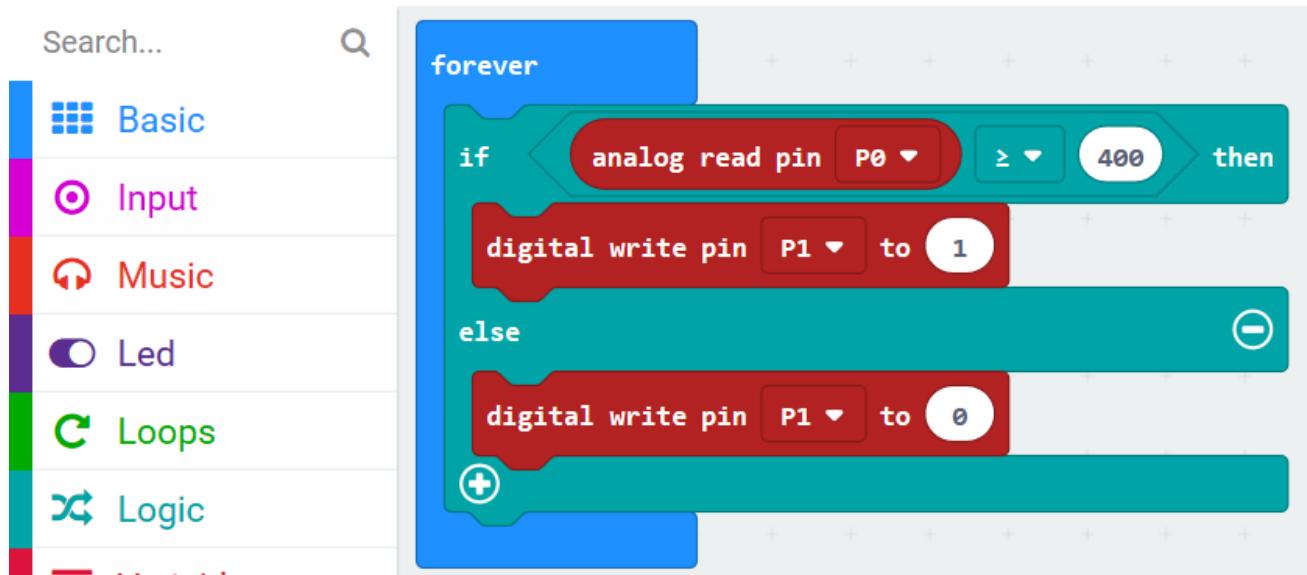
Block code

Open makecode first. Import the .hex file. The path is as below:

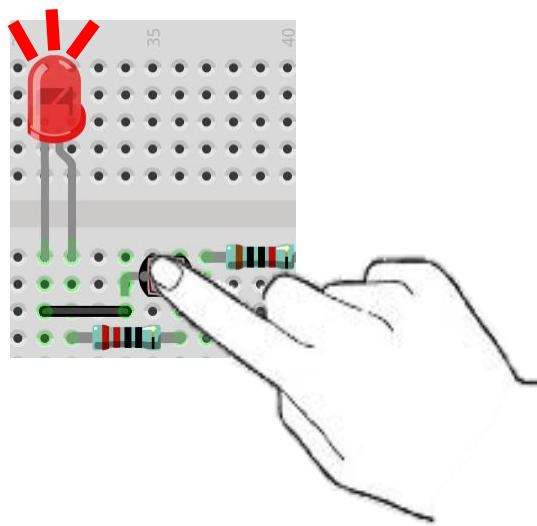
([How to import project](#))

File type	Path	File name
HEX file	../Projects/BlockCode/15.2_NightLight	NightLight.hex

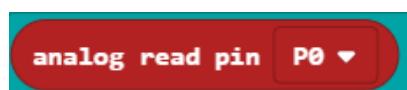
After import successfully, the code is shown as below:



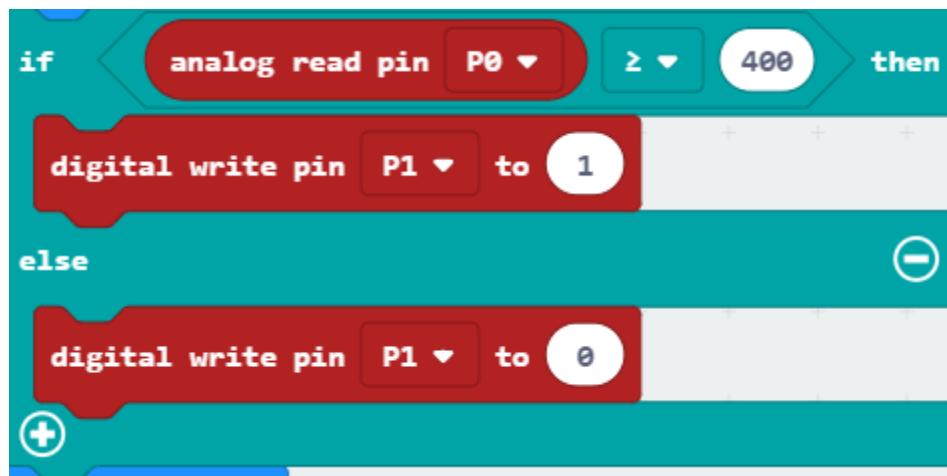
Check the connection of the circuit, confirm the correct connection of the circuit, and download the code into the micro:bit. Cover the photoresistor with your hand, the LED is turned on. Move the hand away, the LED is turned off.



Read the analog voltage value of the P0 pin.



If the analog voltage read is greater than or equal to 400, it is considered to be occluded, and the LED is turned on. Or the LED is turned off.

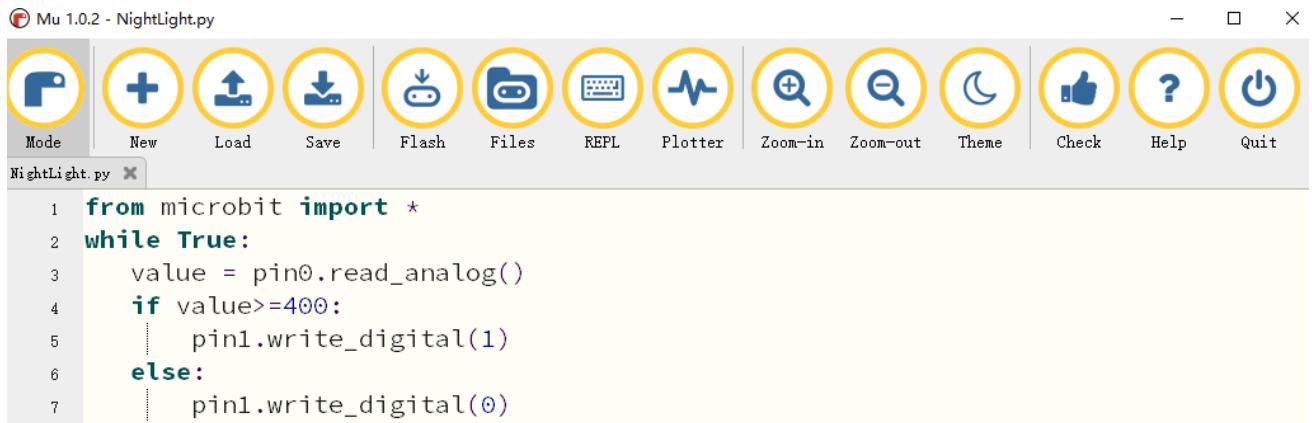


Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	./Projects/PythonCode/15.2_NightLight	NightLight.py

After load successfully, the code is shown as below:

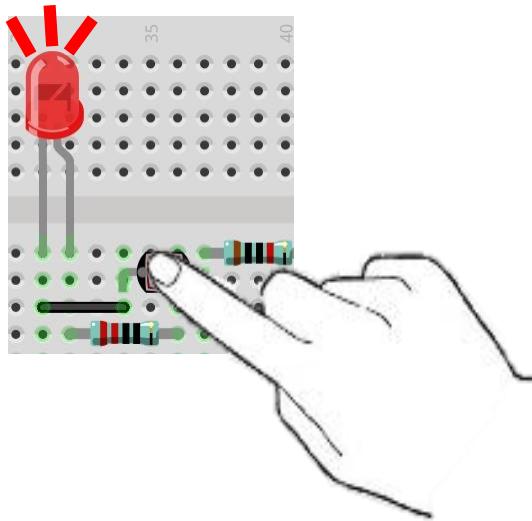


```

1 from microbit import *
2 while True:
3     value = pin0.read_analog()
4     if value>=400:
5         pin1.write_digital(1)
6     else:
7         pin1.write_digital(0)

```

Check the connection of the circuit, confirm the correct connection of the circuit, and download the code into the micro:bit. Cover the photoresistor with your hand, then the LED is turned on. Move the hand away, then the LED is turned off.



The following is the program code:

```
1 from microbit import *
2 while True:
3     value = pin0.read_analog()
4     if value>=400:
5         pin1.write_digital(1)
6     else:
7         pin1.write_digital(0)
```

Read the analog voltage value of the P0 pin.

```
value = pin0.read_analog()
```

If the analog voltage read is greater than or equal to 400, it is considered to be occluded, and the LED is turned on. Or the LED is turned off.

```
if value>=400:
    pin1.write_digital(1)
else:
    pin1.write_digital(0)
```

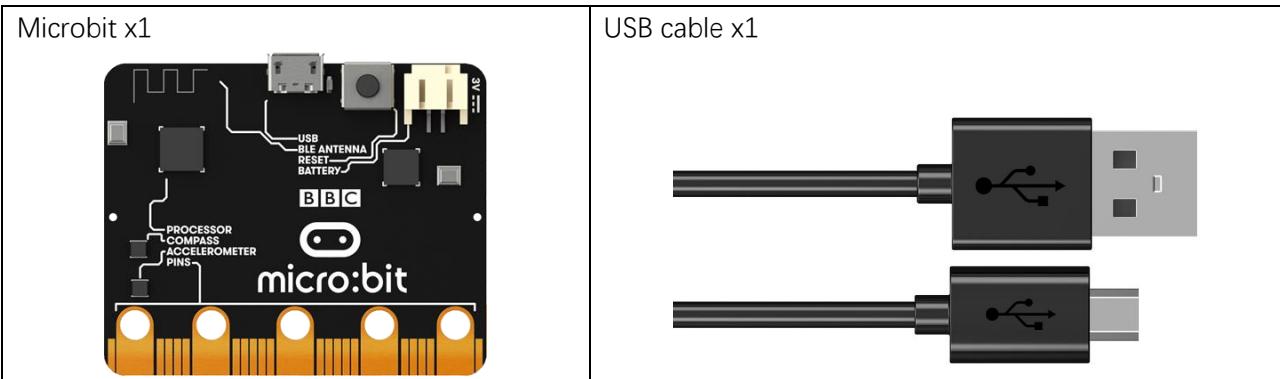
Chapter 16 Temperature Sensor

In this chapter, we will learn the micro:bit built-in temperature sensor and thermistor.

Project 16.1 Built In Temperature Sensor

This project measures the temperature with a micro:bit built-in temperature sensor.

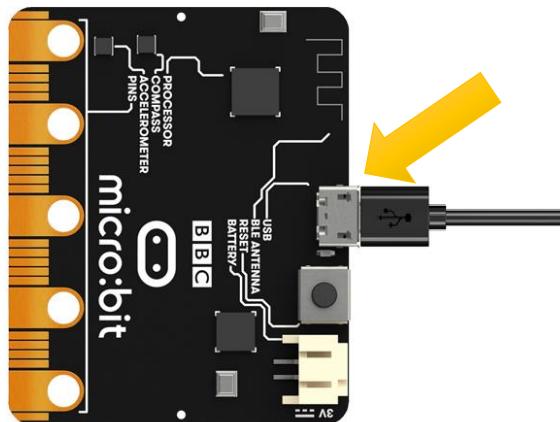
Component list



Circuit

Connect micro:bit and PC via micro USB cable.

Hardware connection



Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file	../Projects/BlockCode/16.1_BuiltInThermometer	BuiltInThermometer.hex

After import successfully, the code is shown as below:



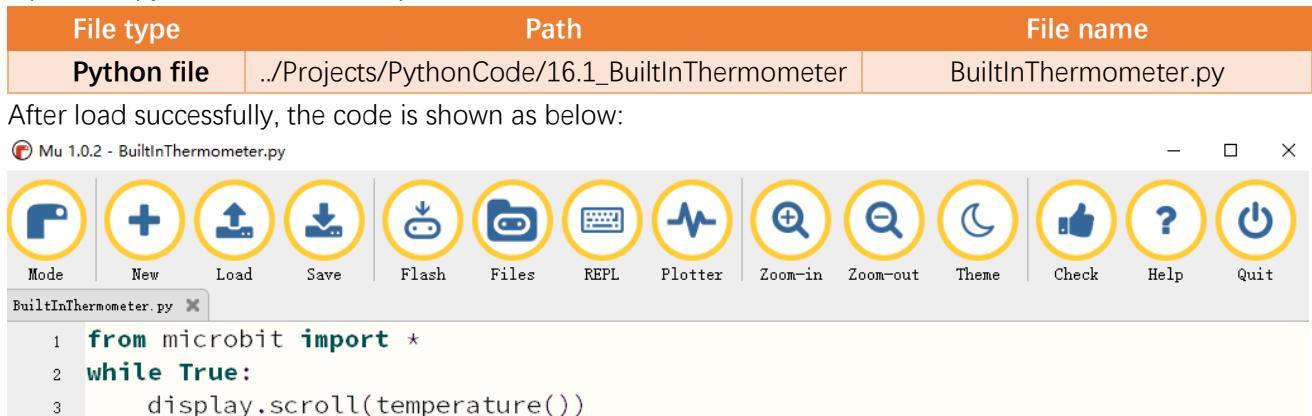
Check the connection of the circuit, confirm the correct connection of the circuit, and download the code into micro:bit. Then LED dot matrix screen will display the current detected temperature.

Reference

Block	Function
temperature (°C)	Find the temperature where you are. The temperature is measured in Celsius (metric).

Python code

Open the .py file with Mu. Code path is as below:



Check the connection of the circuit, confirm the correct connection of the circuit, and download the code into micro:bit. Then LED dot matrix screen will display the current detected temperature.

The following is the program code:

```

1 from microbit import *
2 while True:
3     display.scroll(temperature())

```

Display the detected temperature on the LED dot matrix.

```
display.scroll(temperature())
```

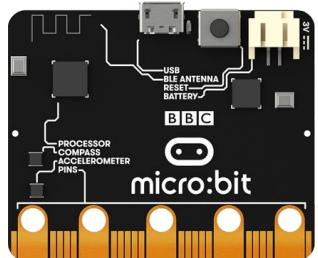
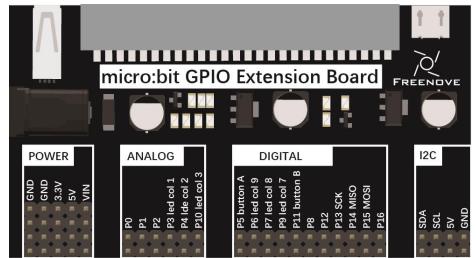
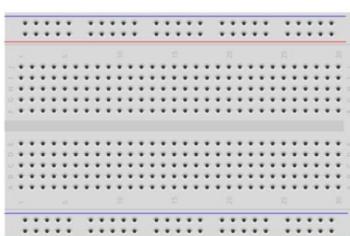
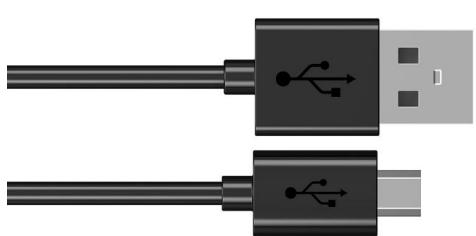
Reference

<code>temperature()</code>	Return the temperature of the micro:bit in degrees Celcius.
<code>display.scroll()</code>	scrolls a string across the display

Project 16.2 Thermistor

This project uses a thermistor to detect the ambient temperature.

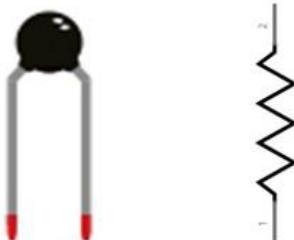
Component list

Microbit x1	Expansion board x1	
		
Breakboard x1	USB cable x1	
		
Jumper F/M x3	Thermistor x1	Resistor 10kΩ x1
		

Component knowledge

Thermistor

Thermistor is a temperature sensitive resistor. When the temperature changes, resistance of thermistor will change. With this feature, we can use thermistor to detect temperature intensity. Thermistor and symbol are as follows.



The relationship between resistance value and temperature of thermistor is:

$$R_t = R \cdot \exp[B \cdot (1/T_2 - 1/T_1)]$$

Where:

R_t is the thermistor resistance under T₂ temperature;

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

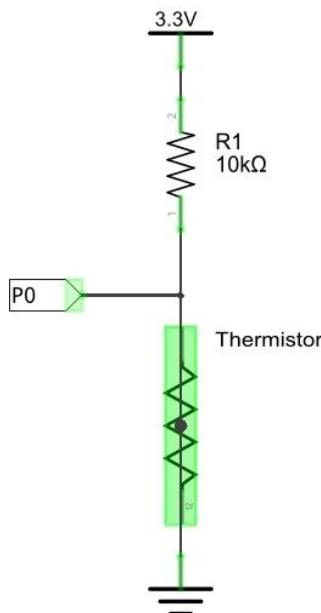
EXP[n] is nth power of E;

B is for thermal index;

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

Parameters of the thermistor we use is: B=3950, R=10k, T1=25.

The circuit connection method of the thermistor is similar to photoresistor, like the following method:



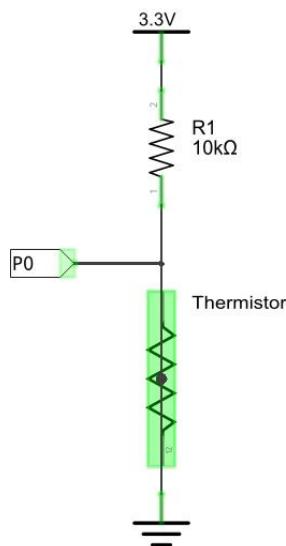
We can use the value measured by the analog pin of micro:bit to obtain resistance value of thermistor, and then can use the formula to obtain the temperature value.

Consequently, the temperature formula can be concluded:

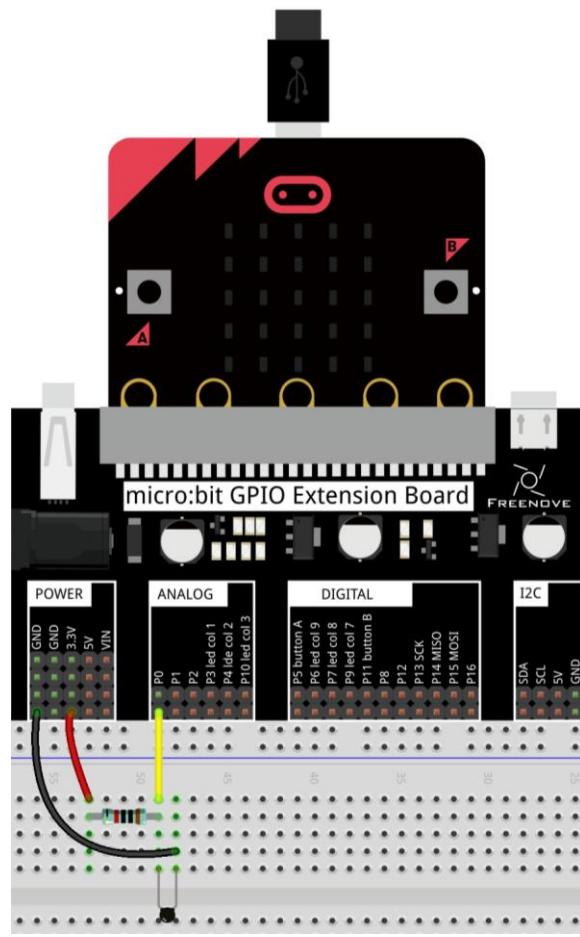
$$T_2 = 1/(1/T_1 + \ln(R_t/R)/B)$$

Circuit

Schematic diagram



Hardware connection



Block code

Open makecode first. Import the .hex file. The path is as below:

[\(How to import project\)](#)

File type	Path	File name
HEX file	../Projects/BlockCode/16.2_ExternalThermometer	ExternalThermometer.hex

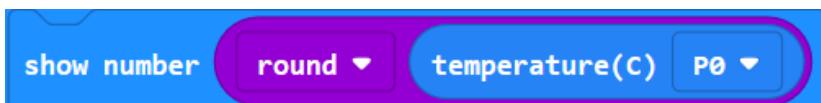
After import successfully, the code is shown as below:



Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into the micro:bit, and the LED dot matrix will display the detected temperature.

The temperature measured by the thermistor and the temperature measured by the micro:bit built-in temperature sensor may have slight difference. This is because the hardware used is different, and there are certain differences in the manufacturing of the components. It is within a reasonable range and can be ignored.

Obtain the temperature data measured by the thermistor. Then round up and display on the LED display.

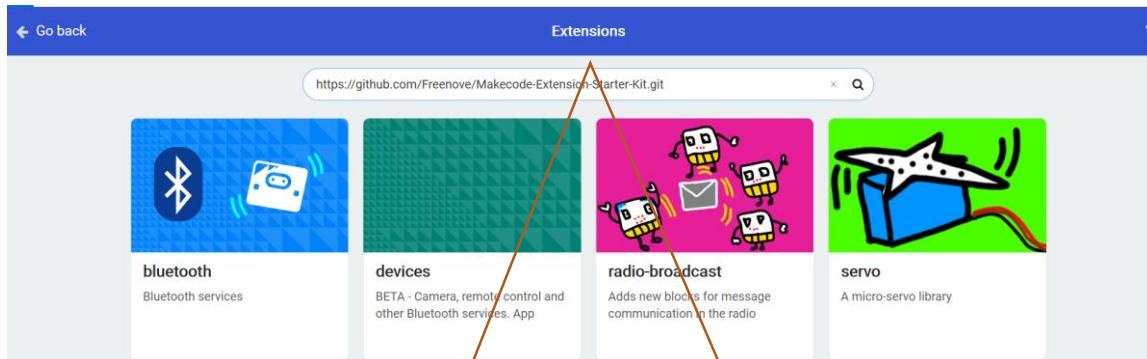
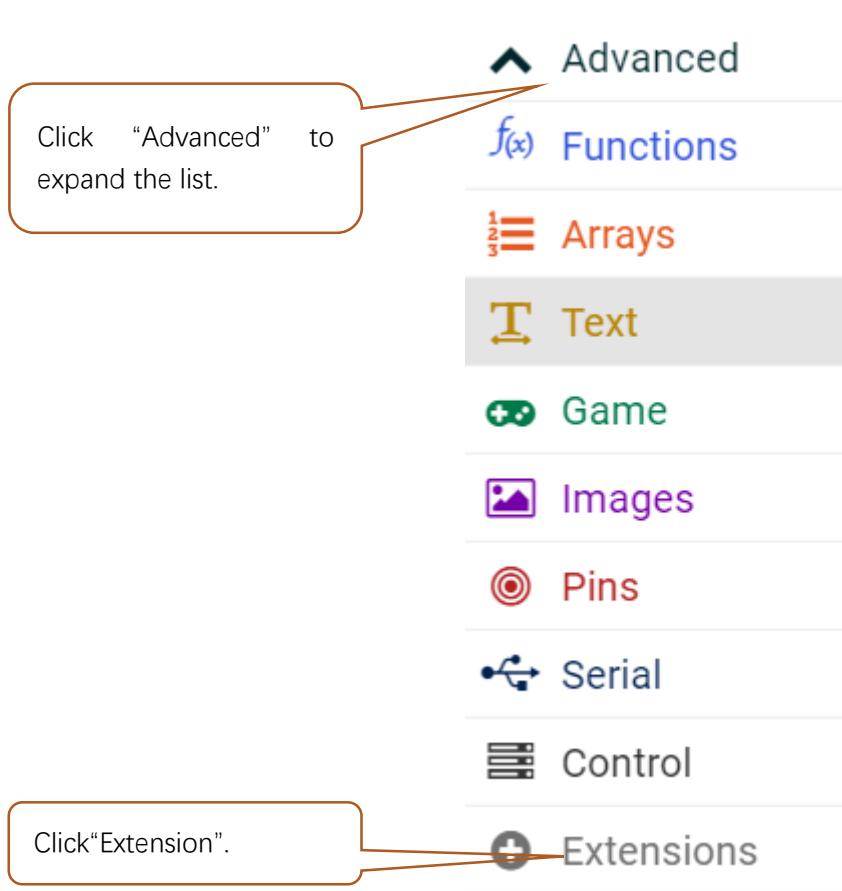


Reference

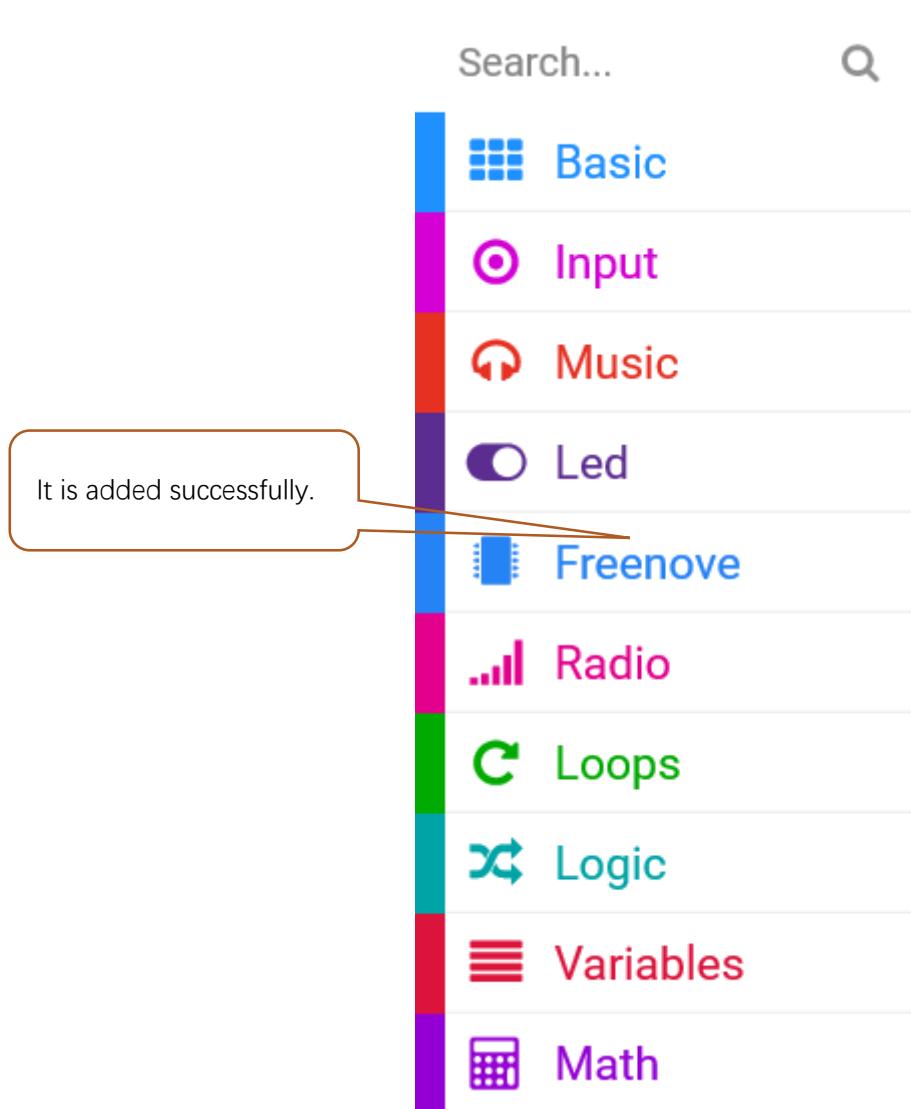
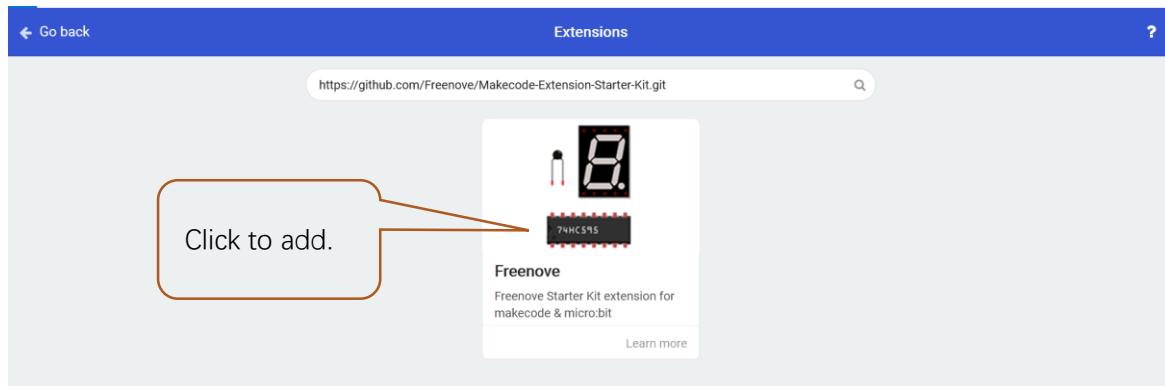
Block	Function
temperature(C) P0 ▾	Belongs to the Freenove extension block. Get the temperature data measured by the thermistor.

Extensions

If you want to import Freenove extensions in a new project, follow the steps below to add them.



Enter "<https://github.com/Freenove/Makecode-Extension-Starter-Kit.git>" to search.

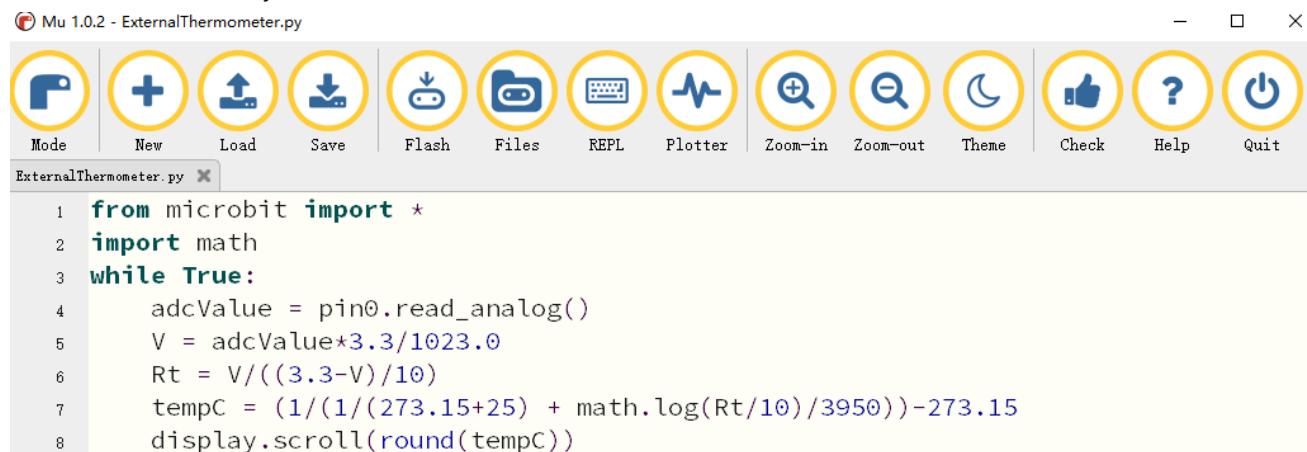


Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	./Projects/PythonCode/16.2_ExternalThermometer	ExternalThermometer.py

After load successfully, the code is shown as below:



```

1 from microbit import *
2 import math
3 while True:
4     adcValue = pin0.read_analog()
5     V = adcValue*3.3/1023.0
6     Rt = V/((3.3-V)/10)
7     tempC = (1/(1/(273.15+25) + math.log(Rt/10)/3950))-273.15
8     display.scroll(round(tempC))

```

Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into the micro:bit, and the LED dot matrix will display the detected temperature.

The following is the program code:

```

1 from microbit import *
2 import math
3 while True:
4     adcValue = pin0.read_analog()
5     V = adcValue*3.3/1023.0
6     Rt = V/((3.3-V)/10)
7     tempC = (1/(1/(273.15+25) + math.log(Rt/10)/3950))-273.15
8     display.scroll(round(tempC))

```

Read the analog voltage of the thermistor and calculate the resistance of the thermistor.

```

adcValue = pin0.read_analog()
V = adcValue*3.3/1023.0
Rt = V/((3.3-V)/10)

```

Calculate the current temperature according to the resistance value of the thermistor. For the formula, please refer to the component knowledge, and then display it on the LED dot matrix screen.

```

tempC = (1/(1/(273.15+25) + math.log(Rt/10)/3950))-273.15
display.scroll(round(tempC))

```

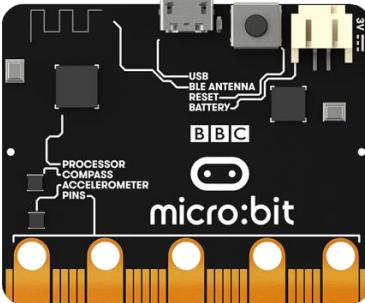
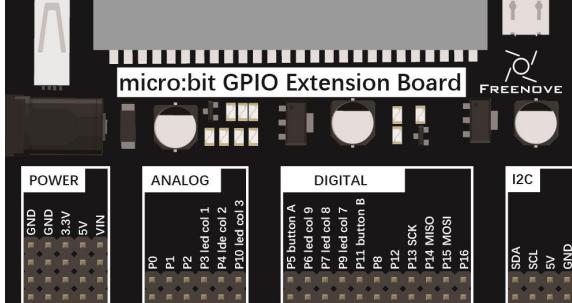
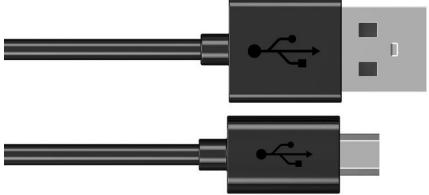
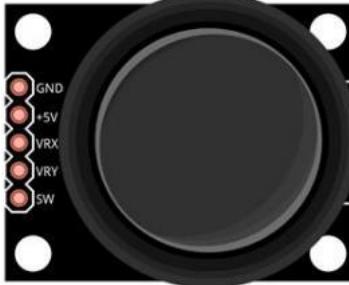
Chapter 17 Joystick

In the previous chapter, we have learned the potentiometer. Now, we will learn a new electronic module: the joystick, which has similar work principle with potentiometer.

Project 17.1 Display Joystick Data

This project will print the Joystick data read.

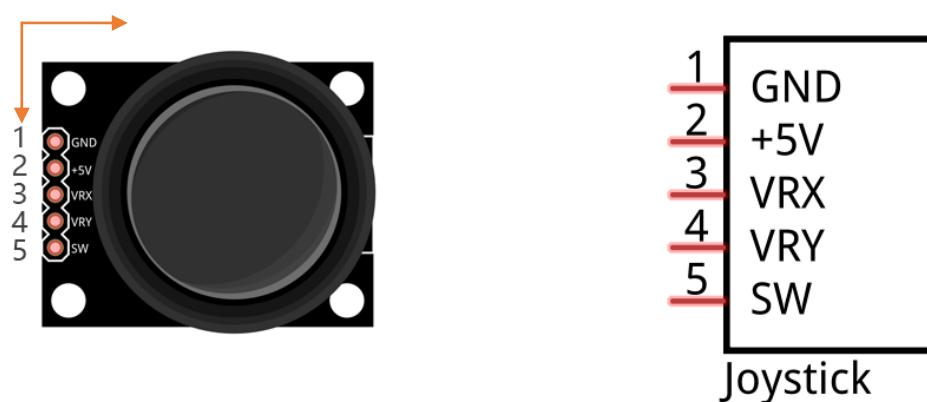
Component list

Microbit x1	Expansion board x1
	
USB cable x2	Joystick x1
	
F/F x4 F/M x3	Resistor 10kΩ x1
	

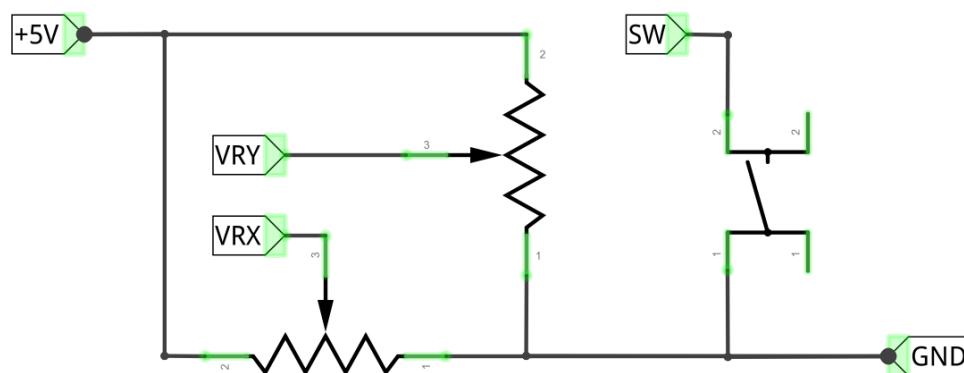
Component knowledge

Joystick

Joystick is a kind of sensor used with your fingers, which is widely used in gamepad and remote controller. It can shift in direction Y or direction X at the same time. And it can also be pressed in direction Z.

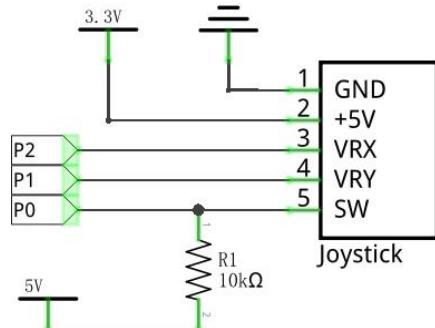


Two rotary potentiometers inside the joystick are set to detect the shift direction of finger, and a push button in vertical direction is set to detect the action of pressing.

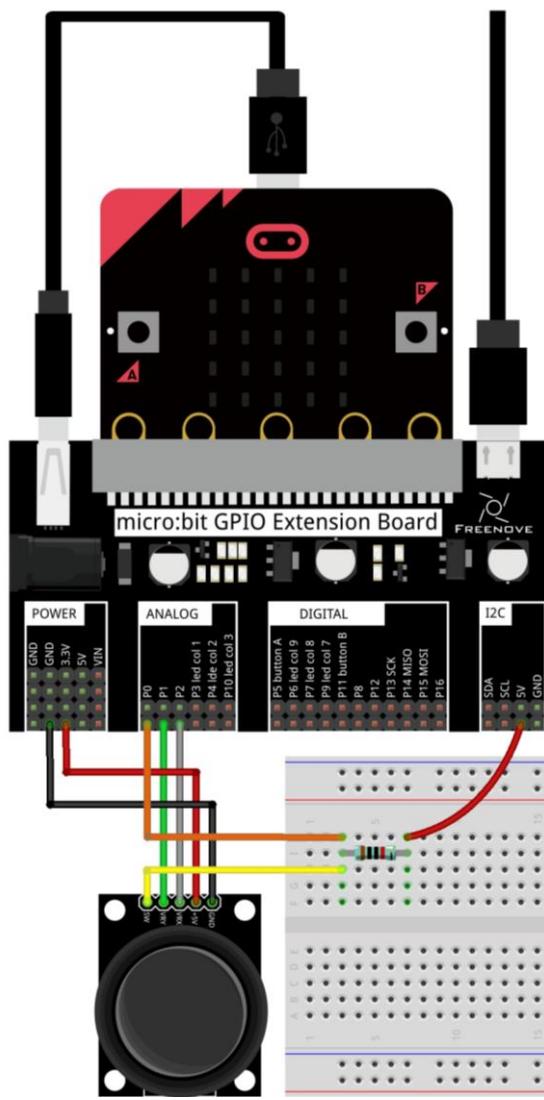


Circuit

Schematic diagram



Hardware connection



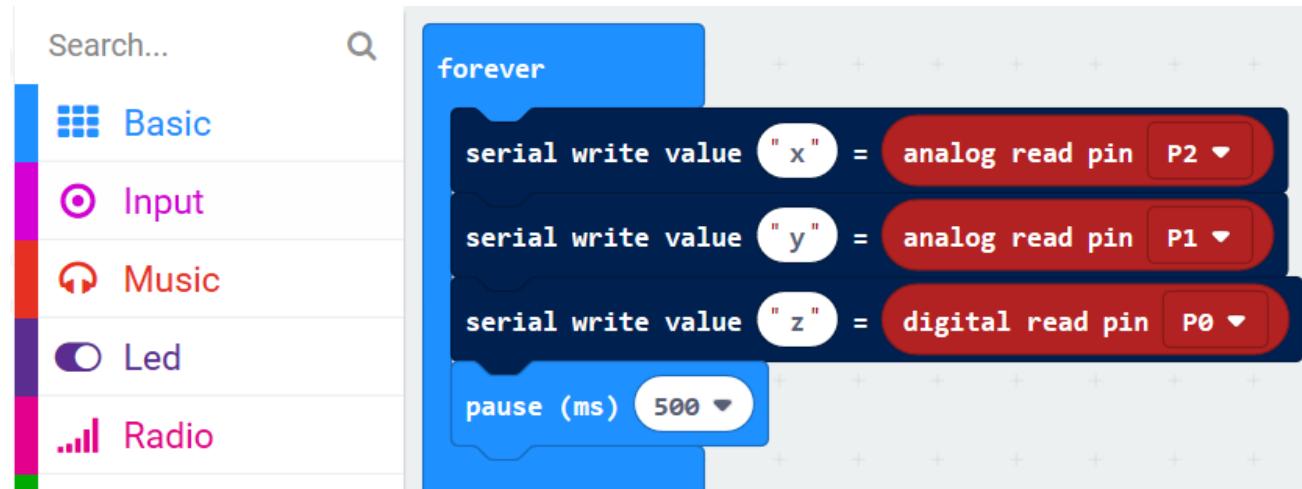
Block code

Open makecode first. Import the .hex file. The path is as below:

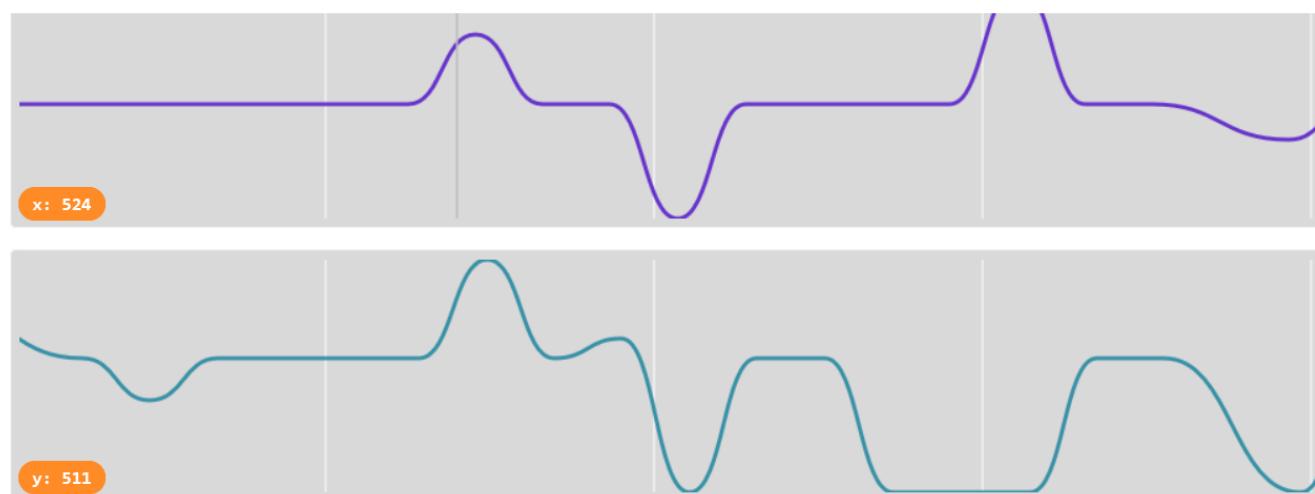
([How to import project](#))

File type	Path	File name
HEX file	../Projects/BlockCode/17.1_DisplayJoystickData	DisplayJoystickData.hex

After import successfully, the code is shown as below:



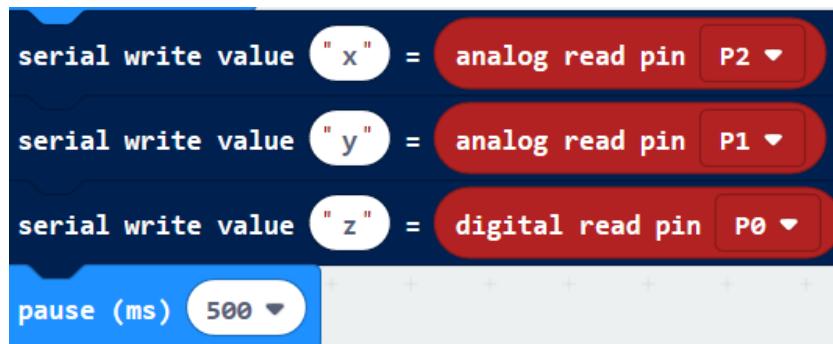
Check the connection of the circuit, confirm the correct connection of the circuit, and download the code into the micro:bit. Open the serial console, then you can see the Joystick data, as shown below.





```
x:524  
y:511  
z:0  
x:524  
y:512  
z:1  
x:524  
y:512
```

Read the analog voltage value of P1 and P2 pins and the digital voltage value of P0 pin, and print the values every 500ms.

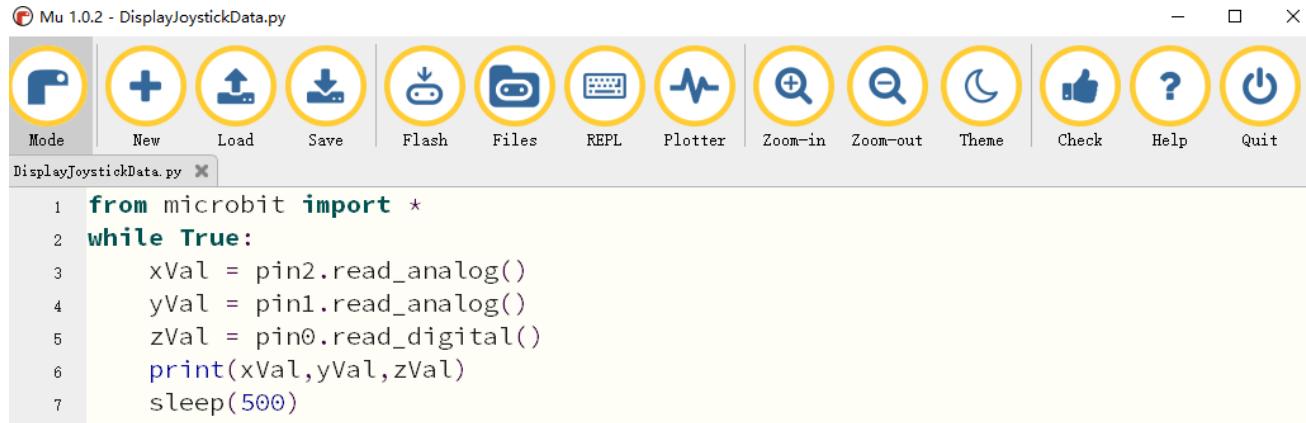


Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/17.1_DisplayJoystickData	DisplayJoystickData.py

After load successfully, the code is shown as below:



```
from microbit import *
while True:
    xVal = pin2.read_analog()
    yVal = pin1.read_analog()
    zVal = pin0.read_digital()
    print(xVal,yVal,zVal)
    sleep(500)
```

Check the connection of the circuit, confirm that the circuit is connected correctly, and download the code into the micro:bit. Click on the REPL, and then press the micro:bit reset button to see the Joystick data.



```
524 512 1
3 512 1
961 3 1
524 512 1
524 511 1
524 512 0
524 512 1
524 512 1
```

The following is the program code:

```
from microbit import *
while True:
    xVal = pin2.read_analog()
    yVal = pin1.read_analog()
    zVal = pin0.read_digital()
    print(xVal, yVal, zVal)
    sleep(500)
```

Read the analog voltage value of P0, P1, P2 pins.

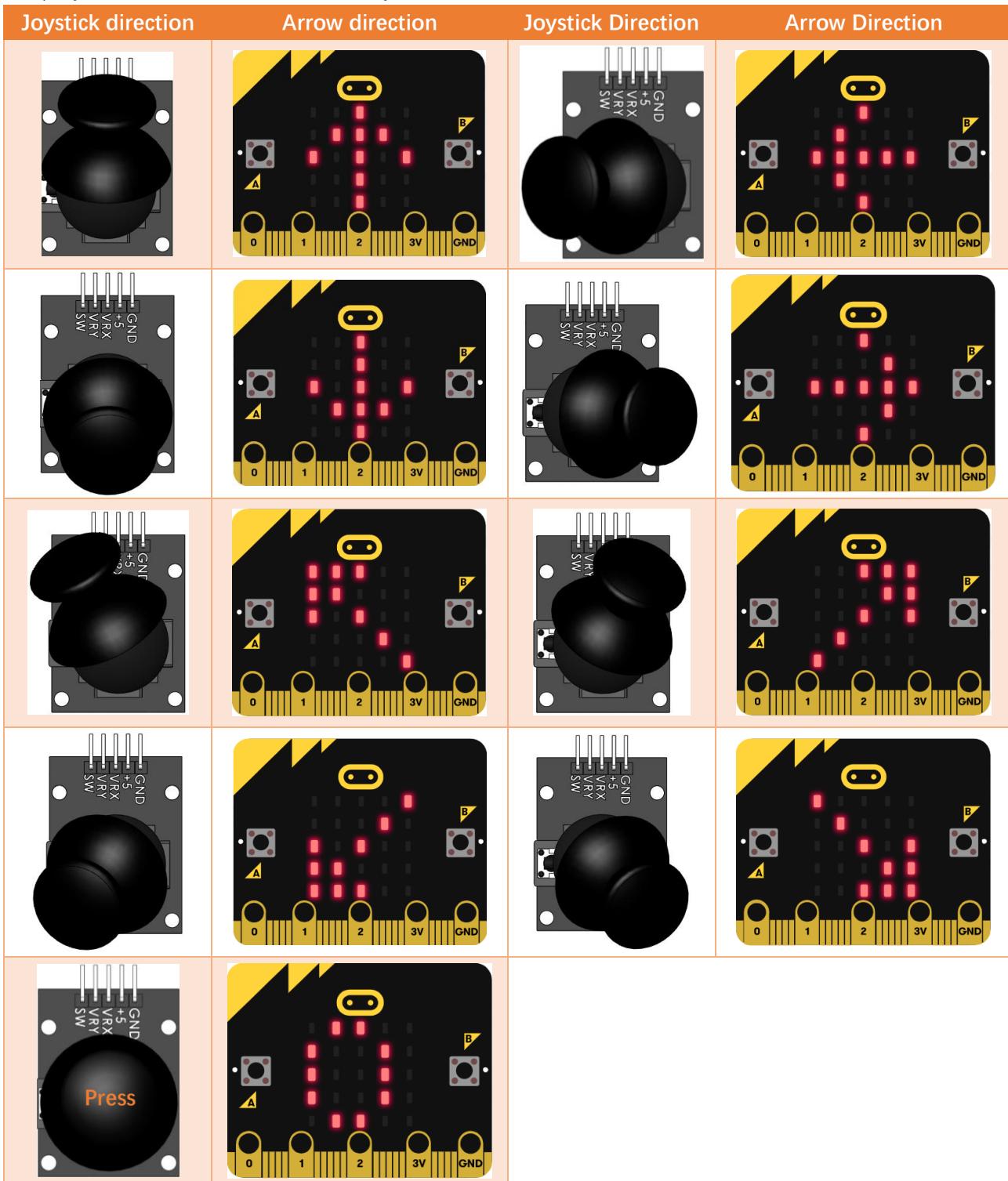
```
xVal = pin2.read_analog()
yVal = pin1.read_analog()
zVal = pin0.read_digital()
```

Print data every 500ms.

```
print(xVal, yVal, zVal)
sleep(500)
```

Project 17.2 Show Direction

This project shows the direction of the Joystick with arrows on the dot matrix.



Component list

It is same with last project.

Circuit

It is same with last project.

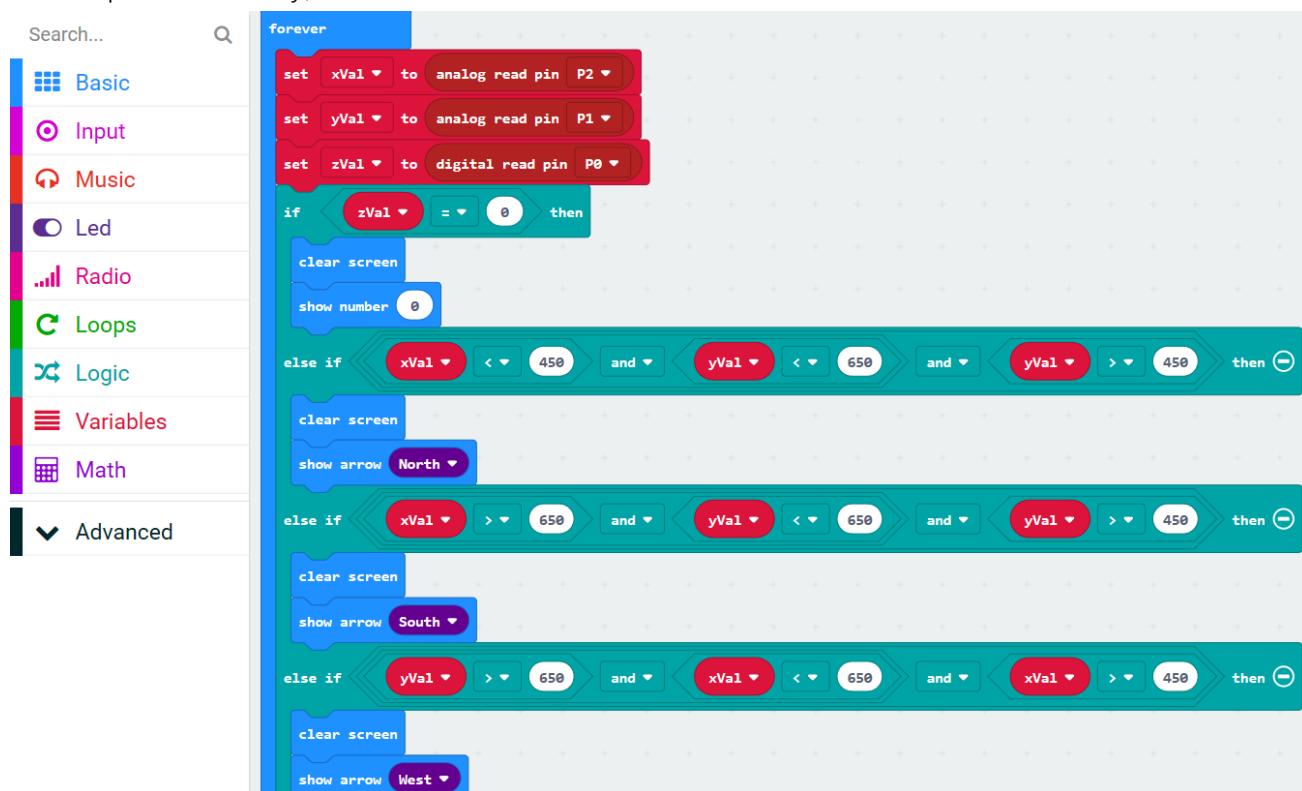
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file	./Projects/BlockCode/17.2_Joystick	Joystick.hex

After import successfully, the code is shown as below:





Check the connection of the circuit and confirm that the circuit is connected correctly. Download the code into the micro:bit. Then the direction of the Joystick will be displayed on the dot matrix.

Get the values in the X, Y, and Z directions.



Display the corresponding arrow image according to the values of the X, Y, and Z directions.

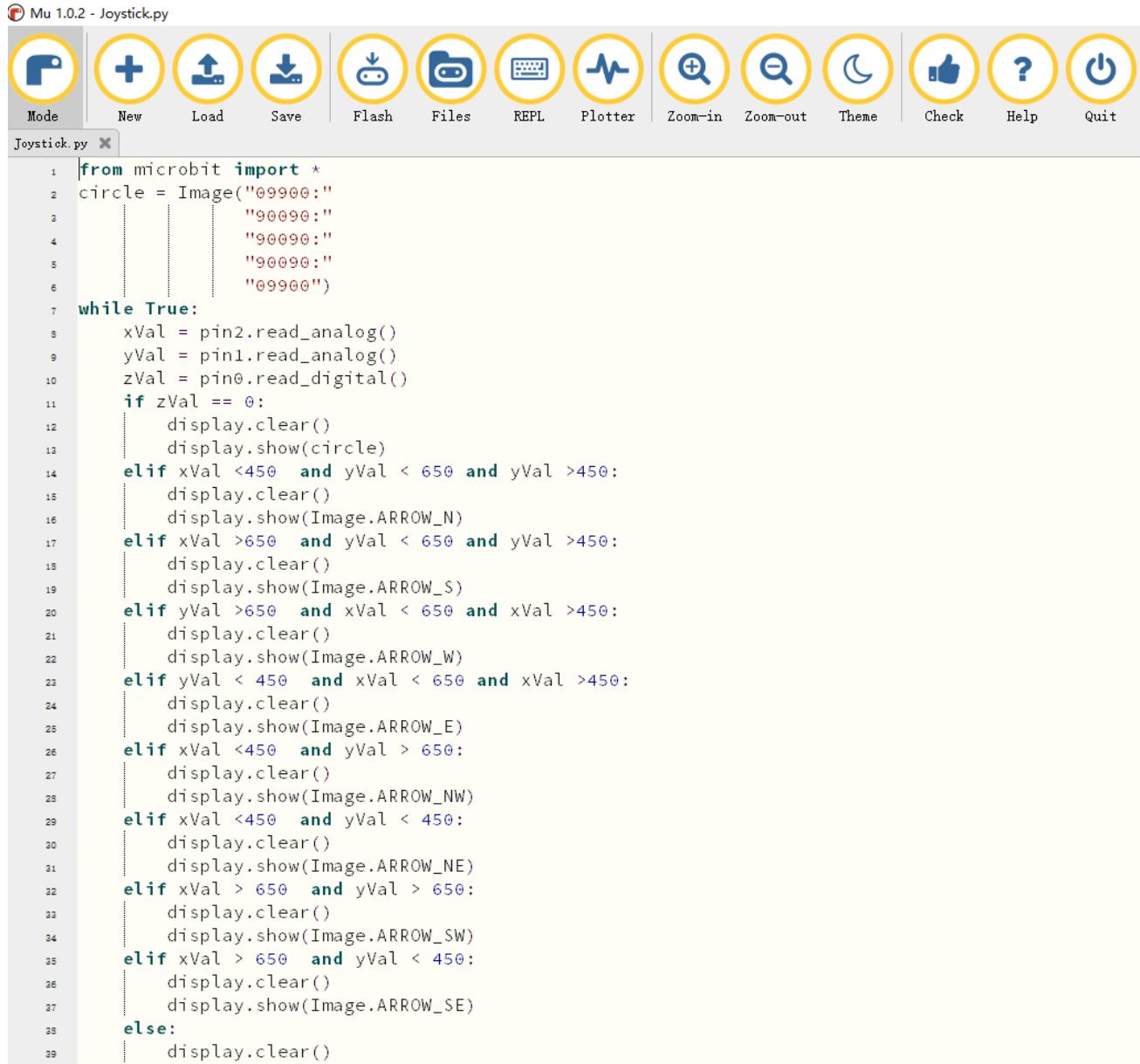


Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/17.2_Joystick	Joystick.py

After load successfully, the code is shown as below:



```

from microbit import *
circle = Image("09900:00000:00000:00000:09900")
while True:
    xVal = pin2.read_analog()
    yVal = pin1.read_analog()
    zVal = pin0.read_digital()
    if zVal == 0:
        display.clear()
        display.show(circle)
    elif xVal < 450 and yVal < 650 and yVal > 450:
        display.clear()
        display.show(Image.ARROW_N)
    elif xVal > 650 and yVal < 650 and yVal > 450:
        display.clear()
        display.show(Image.ARROW_S)
    elif yVal > 650 and xVal < 650 and xVal > 450:
        display.clear()
        display.show(Image.ARROW_W)
    elif yVal < 450 and xVal < 650 and xVal > 450:
        display.clear()
        display.show(Image.ARROW_E)
    elif xVal < 450 and yVal > 650:
        display.clear()
        display.show(Image.ARROW_NW)
    elif xVal < 450 and yVal < 450:
        display.clear()
        display.show(Image.ARROW_NE)
    elif xVal > 650 and yVal > 650:
        display.clear()
        display.show(Image.ARROW_SW)
    elif xVal > 650 and yVal < 450:
        display.clear()
        display.show(Image.ARROW_SE)
    else:
        display.clear()

```

Check the connection of the circuit and confirm that the circuit is connected correctly. Download the code into the micro:bit and then the direction of the Joystick will be displayed on the dot matrix.

The following is the program code:

```
1 from microbit import *
2 circle = Image("09900:"
3                 "90090:"
4                 "90090:"
5                 "90090:"
6                 "09900")
7 while True:
8     xVal = pin2.read_analog()
9     yVal = pin1.read_analog()
10    zVal = pin0.read_digital()
11    if zVal == 0:
12        display.clear()
13        display.show(circle)
14    elif xVal <450 and yVal < 650 and yVal >450:
15        display.clear()
16        display.show(Image.ARROW_N)
17    elif xVal >650 and yVal < 650 and yVal >450:
18        display.clear()
19        display.show(Image.ARROW_S)
20    elif yVal >650 and xVal < 650 and xVal >450:
21        display.clear()
22        display.show(Image.ARROW_W)
23    elif yVal < 450 and xVal < 650 and xVal >450:
24        display.clear()
25        display.show(Image.ARROW_E)
26    elif xVal <450 and yVal > 650:
27        display.clear()
28        display.show(Image.ARROW_NW)
29    elif xVal <450 and yVal < 450:
30        display.clear()
31        display.show(Image.ARROW_NE)
32    elif xVal > 650 and yVal > 650:
33        display.clear()
34        display.show(Image.ARROW_SW)
35    elif xVal > 650 and yVal < 450:
36        display.clear()
37        display.show(Image.ARROW_SE)
38    else:
39        display.clear()
```

Custom image number "0", which will be displayed when pressing the Joystick.

```
circle = Image("09900:"  
              "90090:"  
              "90090:"  
              "90090:"  
              "09900")
```

Get the values in the X, Y, and Z directions.

```
xVal = pin2.read_analog()  
yVal = pin1.read_analog()  
zVal = pin0.read_analog()
```

Display the corresponding arrow image according to the value of X, Y, Z in three directions

```
if zVal == 1:  
    display.clear()  
    display.show(circle)  
elif xVal < 450 and yVal < 650 and yVal > 450:  
    display.clear()  
    display.show(Image.ARROW_N)  
elif xVal > 650 and yVal < 650 and yVal > 450:  
    display.clear()  
    display.show(Image.ARROW_S)  
elif yVal > 650 and xVal < 650 and xVal > 450:  
    display.clear()  
    display.show(Image.ARROW_W)  
elif yVal < 450 and xVal < 650 and xVal > 450:  
    display.clear()  
    display.show(Image.ARROW_E)  
elif xVal < 450 and yVal > 650:  
    display.clear()  
    display.show(Image.ARROW_NW)  
elif xVal < 450 and yVal < 450:  
    display.clear()  
    display.show(Image.ARROW_NE)  
elif xVal > 650 and yVal > 650:  
    display.clear()  
    display.show(Image.ARROW_SW)  
elif xVal > 650 and yVal < 450:  
    display.clear()  
    display.show(Image.ARROW_SE)  
else:  
    display.clear()
```

Chapter 18 74HC595 and LED Bar Graph

In this chapter, we will learn a new component: 74HC595

Project 18.1 Flowing Light

This project use 74HC595 and LEDBar to realize flowing light.

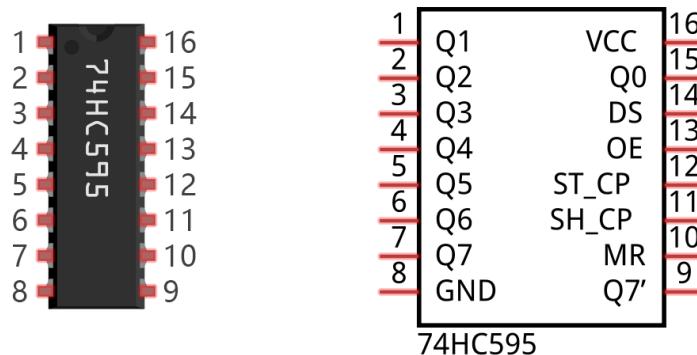
Component list

Microbit x1	Expansion board x1
Breakboard x1	USB cable x1
LED bar graph x1	74HC595 x1
	F/M x5 M/M x11
	Resistor 220Ω x8

Component knowledge

74HC595

74HC595 chip is used to convert serial data into parallel data. 74HC595 can convert the serial data of one byte to 8 bits, and send its corresponding level to the corresponding 8 ports. With this feature, 74HC595 can be used to expand the IO port of Micro:bit board. At least 3 ports on the Micro:bit board are need to control 8 ports of 74HC595.



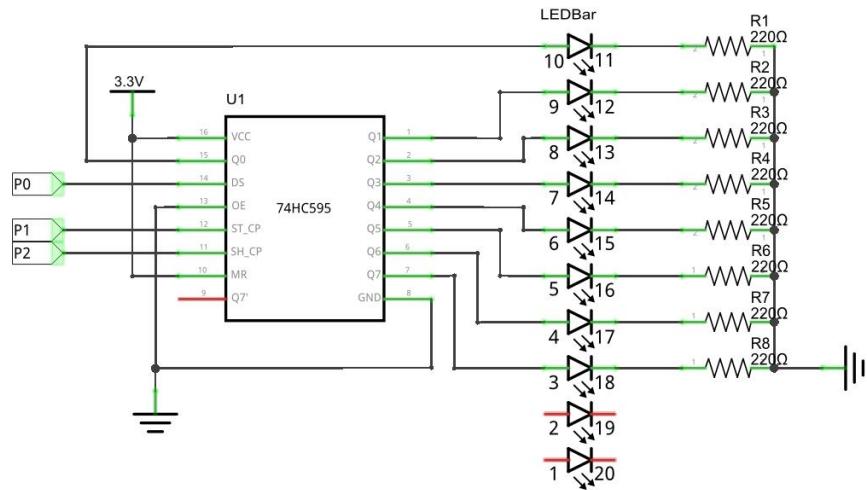
The ports of 74HC595 are described as follows:

Pin name	Pin number	Description
Q0-Q7	15, 1-7	Parallel data output
VCC	16	The positive electrode of power supply, the voltage is 2~6V
GND	8	The negative electrode of power supply
DS	14	Serial data Input
OE	13	Enable output, When this pin is in high level, Q0-Q7 is in high resistance state When this pin is in low level, Q0-Q7 is in output mode
ST_CP	12	Parallel update output: when its electrical level is rising, it will update the parallel data output.
SH_CP	11	Serial shift clock: when its electrical level is rising, serial data input register will do a shift.
MR	10	Remove shift register: When this pin is in low level, the content in shift register will be cleared .
Q7'	9	Serial data output: it can be connected to more 74HC595 in series.

For more detail, please refer to the datasheet.

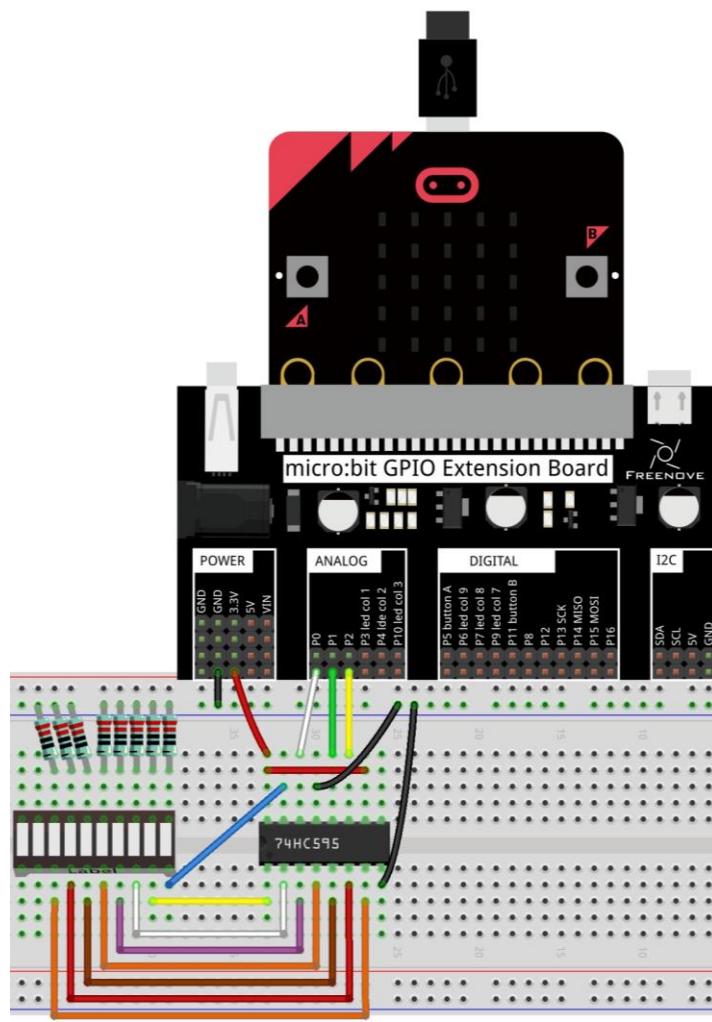
Circuit

Schematic diagram



Hardware connection

If LED bar doesn't work, try rotating the LED bar 180°.



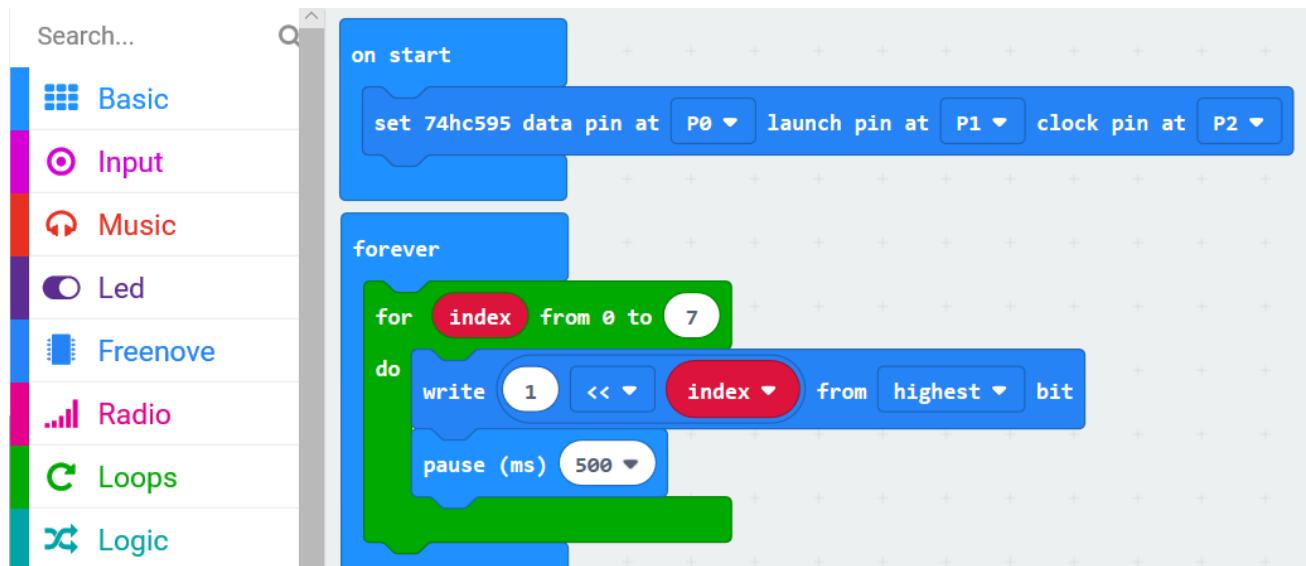
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

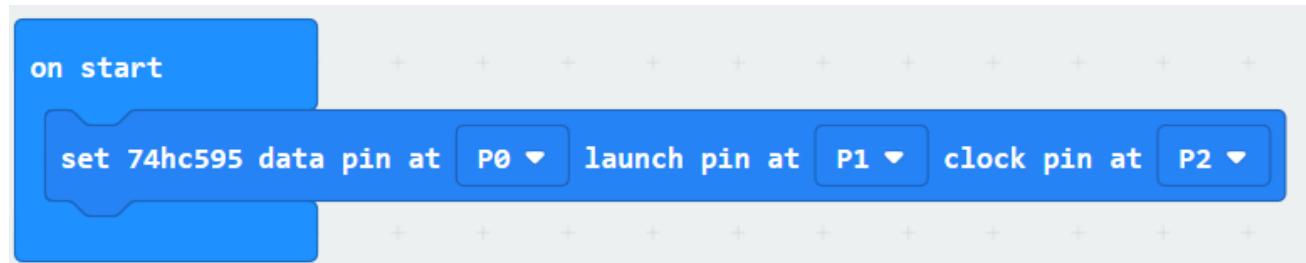
File type	Path	File name
HEX file	./Projects/BlockCode/18.1_FlowingLight02	FlowingLight02.hex

After import successfully, the code is shown as below:

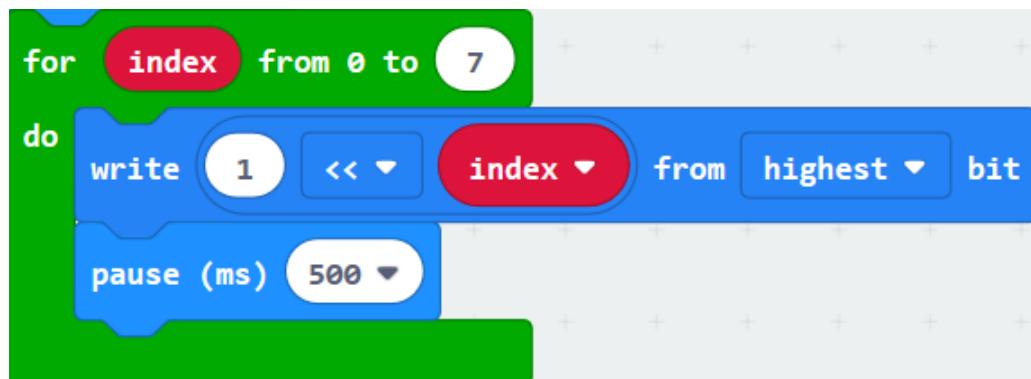


After checking the connection of the circuit and confirming the correct connection of the circuit, download the code into micro:bit, and you can see that the LED flow from left to right in turn cyclically.

Set 74HC595 pin data as P0, launch as P1, and clock as P2.



In the for loop, the number '1' moves index bit to the left, writes the shifted value to 74HC595 serially, and then turn on the LED through parallel output of Q0-Q7 to realize flowing water light.



Reference

Block	Function
set 74hc595 data pin at P0 launch pin at P0 clock pin at P0	It belongs to Freenove Extension Block. It is used to set data pin, launch pin, clock pin for 74HC595.
write 0 from highest bit	It belongs to Freenove Extension Block. The data of 0-255 is serially written to 74HC595, and then output in parallel through Q0-Q7. The order of data writing is highest bit or least bit.
0 << 0	It belongs to Freenove Extension Block. Move data to the left (x) bit or to the right (x) bit.

python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/18.1_FlowingLight02	FlowingLight02.py

After load successfully, the code is shown as below:

```

1 from microbit import *
2 LSBFIRST=1
3 MSBFIRST=2
4 #define the pins connect to 74HC595
5 dataPin=pin0 #DS Pin of 74HC595(Pin14)
6 latchPin=pin1 #ST_CP Pin of 74HC595(Pin12)
7 clockPin=pin2 #SH_CP Pin of 74HC595(Pin11)
8 def shiftOut(value,dPin,cPin,order):
9     for i in range (8):
10         cPin.write_digital(0)
11         if order==MSBFIRST:
12             flag=value<<i & 0x80
13             if flag==0x80:
14                 dPin.write_digital(1)
15             else:
16                 dPin.write_digital(0)
17         else:
18             flag=value>>i & 0x01
19             if flag==0x01:
20                 dPin.write_digital(1)
21             else:
22                 dPin.write_digital(0)
23         cPin.write_digital(1)
24 while True:
25     for i in range(8):
26         value=0x01<<i
27         latchPin.write_digital(0)
28         shiftOut(value,dataPin,clockPin,LSBFIRST)
29         latchPin.write_digital(1)
30         sleep(500)

```

After checking the connection of the circuit and confirming the correct connection of the circuit, download the code into micro:bit, and you can see that the LED flow from left to right in turn cyclically.

The following is the program code:

```

1 from microbit import *
2 LSBFIRST=1
3 MSBFIRST=2
4 #define the pins connect to 74HC595

```

```

5  dataPin=pin0 #DS Pin of 74HC595(Pin14)
6  latchPin=pin1 #ST_CP Pin of 74HC595(Pin12)
7  clockPin=pin2 #SH_CP Pin of 74HC595(Pin11)
8  def shiftOut(value, dPin, cPin, order):
9      for i in range (8):
10         cPin.write_digital(0)
11         if order==MSBFIRST:
12             flag=value<<i & 0x80
13             if flag==0x80:
14                 dPin.write_digital(1)
15             else:
16                 dPin.write_digital(0)
17         else:
18             flag=value>>i & 0x01
19             if flag==0x01:
20                 dPin.write_digital(1)
21             else:
22                 dPin.write_digital(0)
23         cPin.write_digital(1)
24     while True:
25         for i in range(8):
26             value=0x01<<i
27             latchPin.write_digital(0)
28             shiftOut(value, dataPin, clockPin, LSBFIRST)
29             latchPin.write_digital(1)
30             sleep(500)

```

Define pins P0, P1, P2 for 74HC595.

```

from microbit import *
LSBFIRST=1
MSBFIRST=2
#define the pins connect to 74HC595
dataPin=pin0 #DS Pin of 74HC595(Pin14)
latchPin=pin1 #ST_CP Pin of 74HC595(Pin12)
clockPin=pin2 #SH_CP Pin of 74HC595(Pin11)

```

Custom shiftOut() function is used to write data to 74HC595 serially.

```

def shiftOut(value, dPin, cPin, order):
    for i in range (8):
        cPin.write_digital(0)
        if order==MSBFIRST:
            flag=value<<i & 0x80
            if flag==0x80:
                dPin.write_digital(1)

```

```

        else:
            dPin.write_digital(0)
    else:
        flag=value>>i & 0x01
        if flag==0x01:
            dPin.write_digital(1)
        else:
            dPin.write_digital(0)
cPin.write_digital(1)

```

In the for loop, the value of the variable value shifts to the left i-bit, then the value of the variable value is written to 74HC595, and then turn the LED through Q0-Q7 to realize the flowing water light.

```

while True:
    for i in range(8):
        value=0x01<<i
        latchPin.write_digital(0)
        shiftOut(value, dataPin, clockPin, LSBFIRST)
        latchPin.write_digital(1)
        sleep(500)

```

Reference

shiftOut(value, dPin, cPin, order)

This function is used to serially write 8 bits of data to 74HC595. Value represents data to be written to 74HC595 registers, dPin represents data pins, cPin represents clock pins, and order represents priority bit flags (high or low). About order, LSBFIRST starts writing from low data, MSBFIRST starts writing from high data.

<< operator

"<<" is a left shift operator that moves all bits of byte data to the left (high) direction by a few bits and add 0 on the right (low). For example, shift binary 0001 1110 to the left by 1 bit to get 0011 1100. If you shift 1 bit to the right, it is 0000 1111.

">>" is the right shift operator, as opposed to the left shift operator, which moves all bits of byte data to the right (low) direction by a few bits and add 0 on the left(high).

& operator

& is a bitwise AND operation, which performs an AND operation on binary bit. Operation rules:

$$0\&0=0;$$

$$0\&1=0;$$

$$1\&0=0;$$

$$1\&1=1$$

For example:

$$A=0011\ 1100$$

$$B=0000\ 1101$$

$$-----$$

$$A\&B=0000\ 1100$$

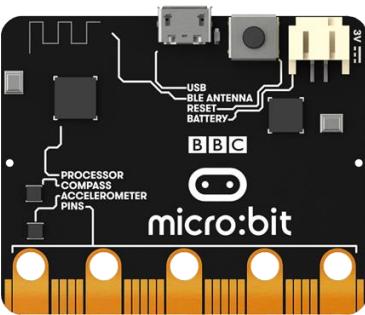
Chapter 19 74HC595 and 7-segment display

In this chapter, we will learn a new component: 7-segment display.

Project 19.1 7-segment display

This project uses the 74HC595 and 7-segment digital tube display to show the numbers 0 to 9.

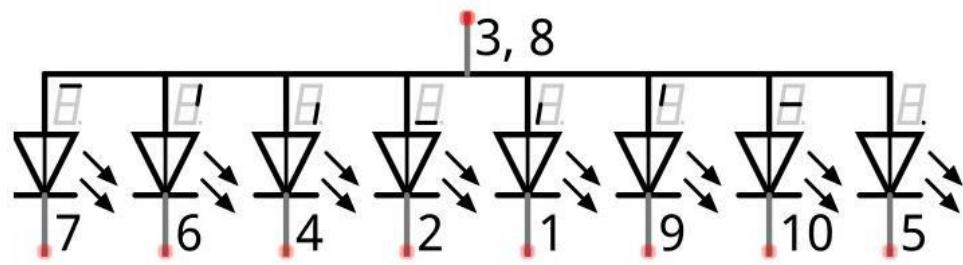
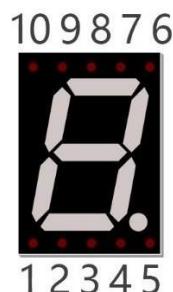
Component list

Microbit x1		Expansion board x1	
Breakboard x1		USB cable x1	
digit-7-segment-display x1		74HC595 x1	
M/M x12 F/M x5		Resistor 220Ω x8	

Component knowledge

digit 7-segment display

1-digit 7-segment display is a kind of electron component that can display numbers. It has a figure of "8" and a decimal point, which consists of 8 LEDs. It can display numbers of 0~9 by lighting some of its LED segment. There are two types of LED common port inside, that is, common cathode port and common anode port. Common anode 1-digit 7-segment display is as follows:



If you want to display 0, you can light the following LEDs that connected with pin 7, 6, 4, 2, 1, 9.



If we use a byte to show the state of the LEDs that connected to pin 5, 10, 9, 1, 2, 4, 6, 7, we can make 0 represent the state of on and 1 for off. Then the number 0 can be expressed as a binary number 11000000, namely hex 0xc0.

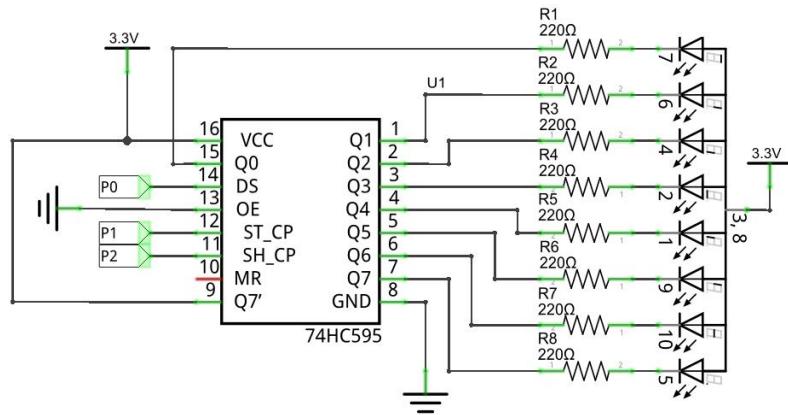
The numbers and letters that can be display are shown below:

Number/Letter	Binary number	Hexadecimal number	Decimal number
0	11000000	0xc0	192
1	11111001	0xf9	249
2	10100100	0xa4	164
3	10110000	0xb0	176
4	10011001	0x99	153
5	10010010	0x92	146
6	10000010	0x82	130
7	11111000	0xf8	248
8	10000000	0x80	128
9	10010000	0x90	144

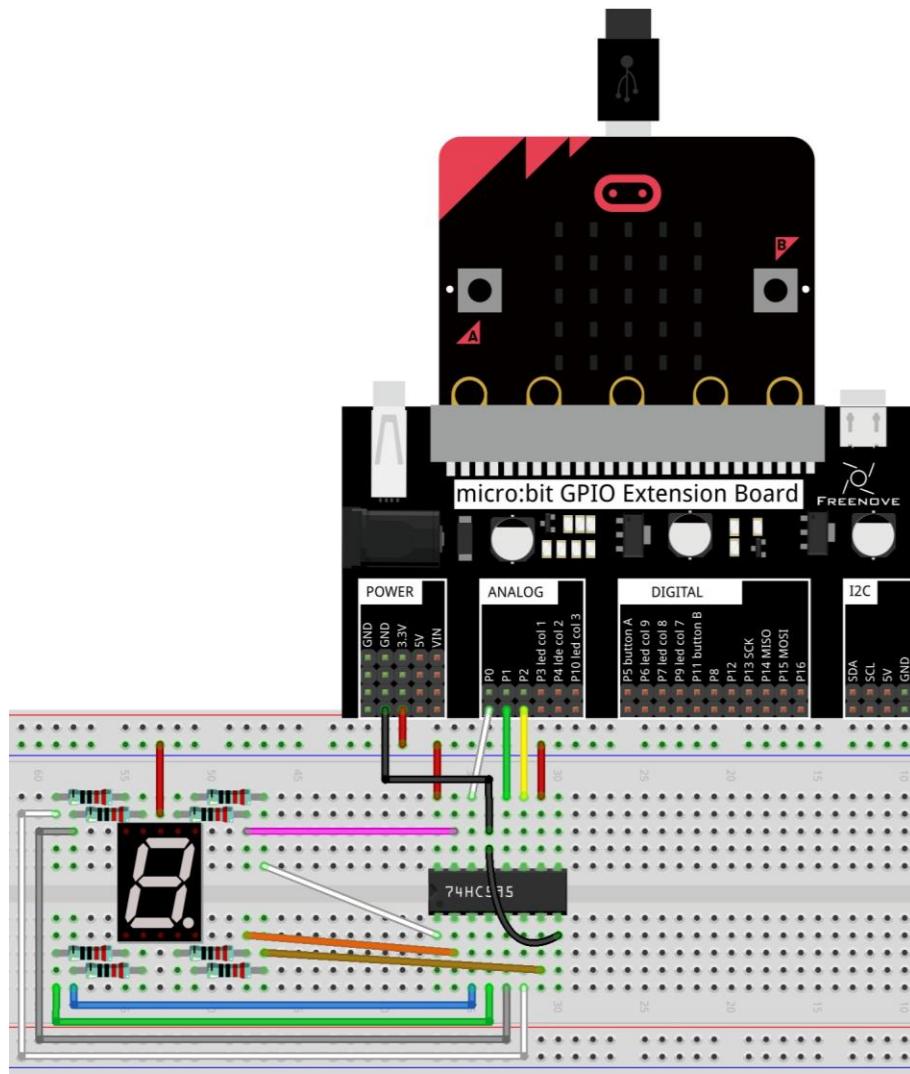
A	10001000	0x88	136
b	10000011	0x83	131
C	11000110	0xc6	198
d	10100001	0xa1	161
E	10000110	0x86	134
F	10001110	0x8e	142

Circuit

Schematic diagram



Hardware connection



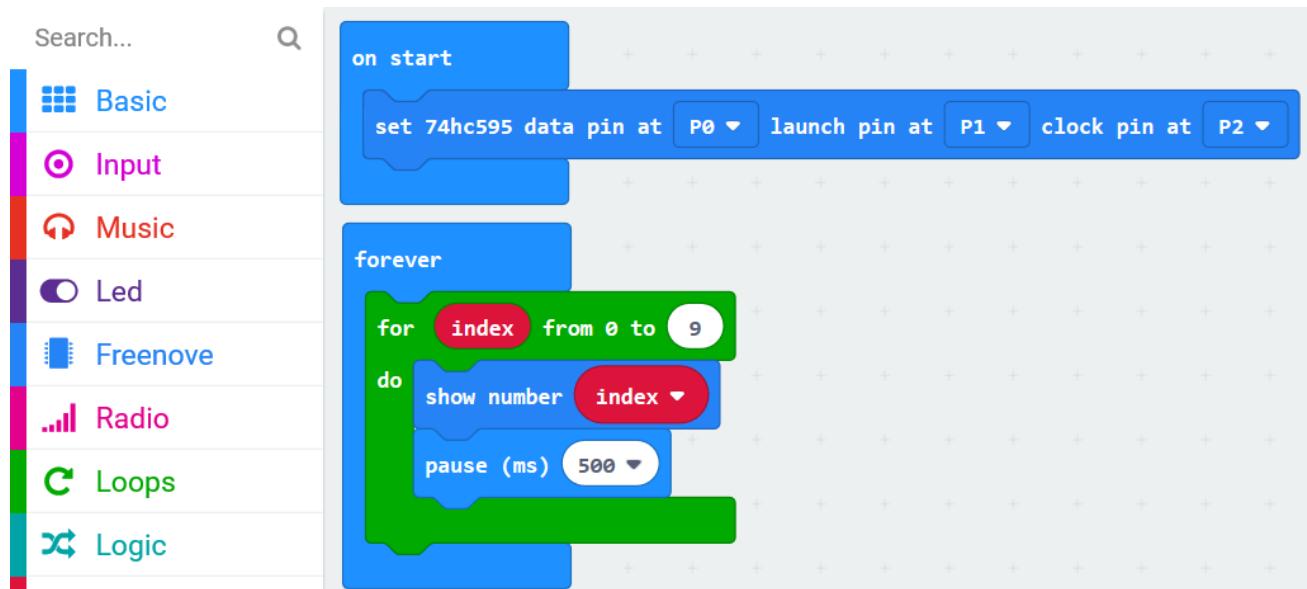
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

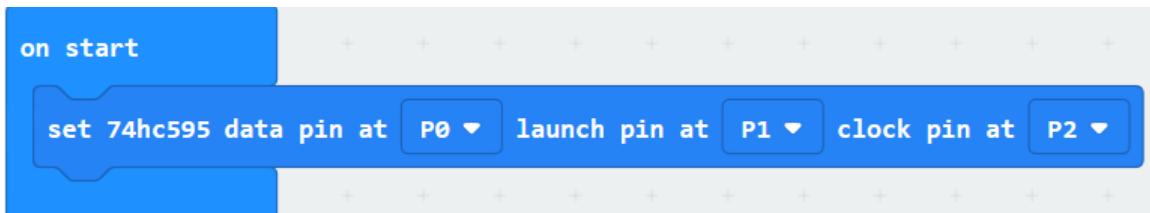
File type	Path	File name
HEX file	./Projects/BlockCode/19.1_SevenSegmentDisplay	SevenSegmentDisplay.hex

After import successfully, the code is shown as below:



After checking the connection of the circuit and confirming the correct connection of the circuit, the code is downloaded into micro:bit. You can see that the 7-segment display shows 0, 1... 9 in turn.

Set 74HC595 pin data as P0, launch as P1, and clock as P2.



In the for loop, the digital tube displays the numbers 0 to 9 in turn, and change the number every 500ms.



Reference

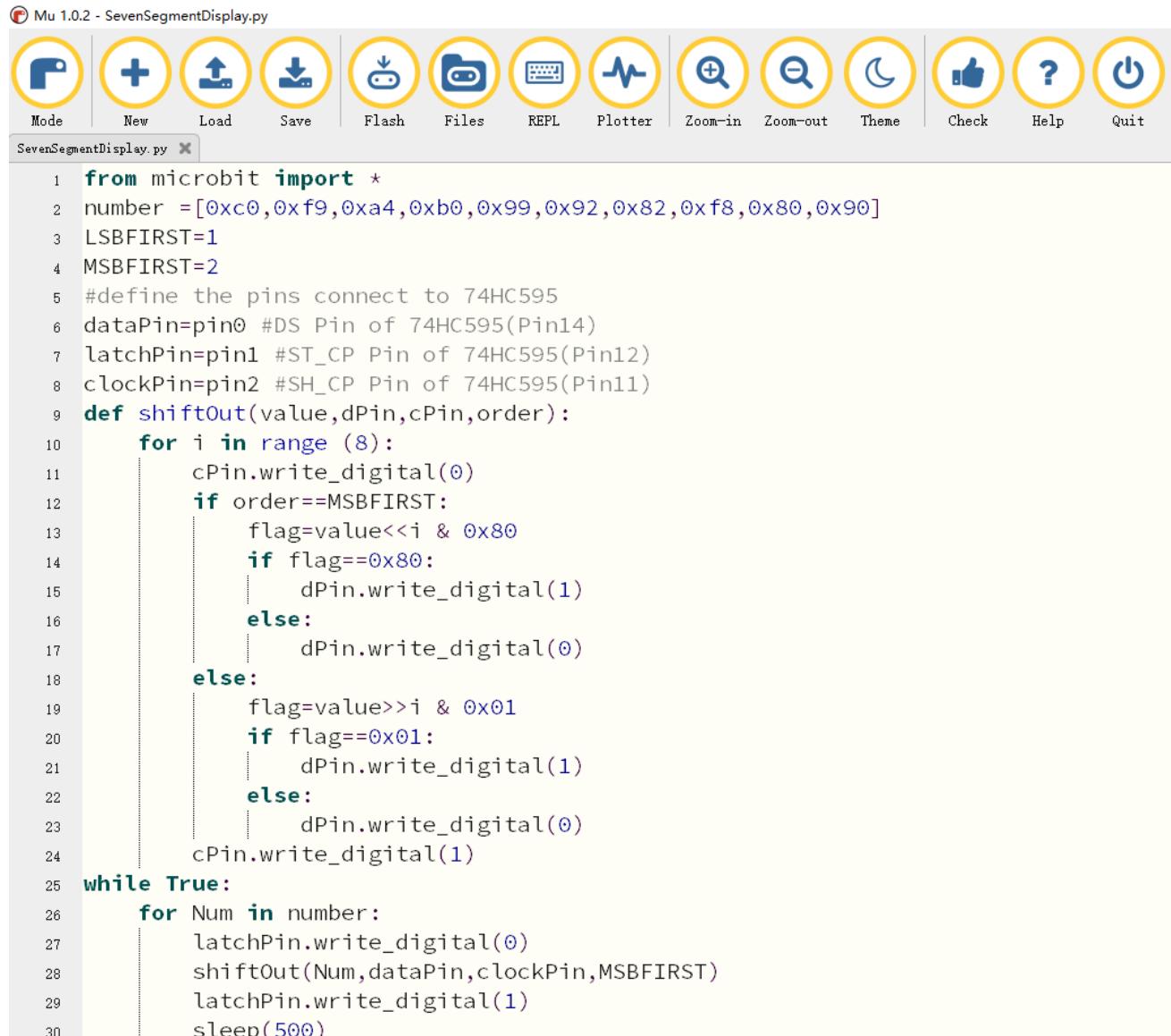
Block	Function
Show Number <code>0</code>	It belongs to Freenove Extension Block. It is used to control digital tube to display number and character 0-F by using 74HC595.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/19.1_SevenSegmentDisplay	SevenSegmentDisplay.py

After load successfully, the code is shown as below:



```

Mu 1.0.2 - SevenSegmentDisplay.py
Mode New Load Save Flash Files REPL Plotter Zoom-in Zoom-out Theme Check Help Quit
SevenSegmentDisplay.py x

1 from microbit import *
2 number =[0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90]
3 LSBFIRST=1
4 MSBFIRST=2
5 #define the pins connect to 74HC595
6 dataPin=pin0 #DS Pin of 74HC595(Pin14)
7 latchPin=pin1 #ST_CP Pin of 74HC595(Pin12)
8 clockPin=pin2 #SH_CP Pin of 74HC595(Pin11)
9 def shiftOut(value,dPin,cPin,order):
10     for i in range (8):
11         cPin.write_digital(0)
12         if order==MSBFIRST:
13             flag=value<<i & 0x80
14             if flag==0x80:
15                 dPin.write_digital(1)
16             else:
17                 dPin.write_digital(0)
18         else:
19             flag=value>>i & 0x01
20             if flag==0x01:
21                 dPin.write_digital(1)
22             else:
23                 dPin.write_digital(0)
24         cPin.write_digital(1)
25 while True:
26     for Num in number:
27         latchPin.write_digital(0)
28         shiftOut(Num,dataPin,clockPin,MSBFIRST)
29         latchPin.write_digital(1)
30         sleep(500)

```

After checking the connection of the circuit and confirming the correct connection of the circuit, the code is downloaded into micro:bit. You can see that the 7-segment display shows 0, 1... 9 in turn.

The following is the program code:

```

1  from microbit import *
2  number =[0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90]
3  LSBFIRST=1
4  MSBFIRST=2
5  #define the pins connect to 74HC595
6  dataPin=pin0 #DS Pin of 74HC595(Pin14)
7  latchPin=pin1 #ST_CP Pin of 74HC595(Pin12)
8  clockPin=pin2 #SH_CP Pin of 74HC595(Pin11)
9  def shiftOut(value, dPin, cPin, order):
10     for i in range (8):
11         cPin.write_digital(0)
12         if order==MSBFIRST:
13             flag=value<<i & 0x80
14             if flag==0x80:
15                 dPin.write_digital(1)
16             else:
17                 dPin.write_digital(0)
18         else:
19             flag=value>>i & 0x01
20             if flag==0x01:
21                 dPin.write_digital(1)
22             else:
23                 dPin.write_digital(0)
24         cPin.write_digital(1)
25     while True:
26         for Num in number:
27             latchPin.write_digital(0)
28             shiftOut(Num, dataPin, clockPin, MSBFIRST)
29             latchPin.write_digital(1)
30             sleep(500)

```

Define variable number to store numbers 0, 1, 2...9 in Hexadecimal.

Define pins P0, P1, P2 for 74HC595.

```

number =[0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90]
LSBFIRST=1
MSBFIRST=2
#define the pins connect to 74HC595
dataPin=pin0 #DS Pin of 74HC595(Pin14)
latchPin=pin1 #ST_CP Pin of 74HC595(Pin12)
clockPin=pin2 #SH_CP Pin of 74HC595(Pin11)

```

Custom shiftOut () function is used to for writing data to 74HC595 serially.

```
def shiftOut(value, dPin, cPin, order):
    for i in range (8):
        cPin.write_digital(0)
        if order==MSBFIRST:
            flag=value<<i & 0x80
            if flag==0x80:
                dPin.write_digital(1)
            else:
                dPin.write_digital(0)
        else:
            flag=value>>i & 0x01
            if flag==0x01:
                dPin.write_digital(1)
            else:
                dPin.write_digital(0)
    cPin.write_digital(1)
```

Call the shiftOut() function, and write the hexadecimal number stored in the number variable to 74HC595 serially, then turn on the LED through parallel output of Q0 ~ Q7.

```
while True:
    for Num in number:
        latchPin.write_digital(0)
        shiftOut(Num, dataPin, clockPin, MSBFIRST)
        latchPin.write_digital(1)
        sleep(500)
```

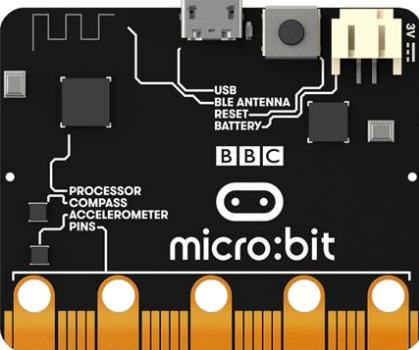
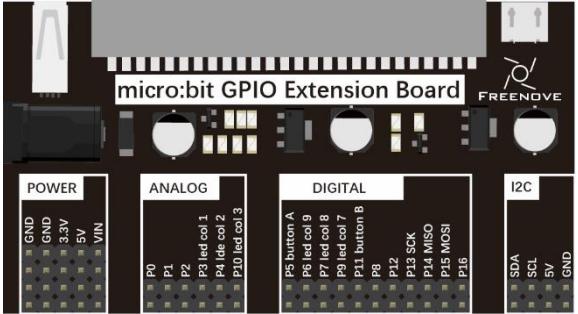
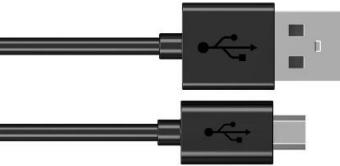
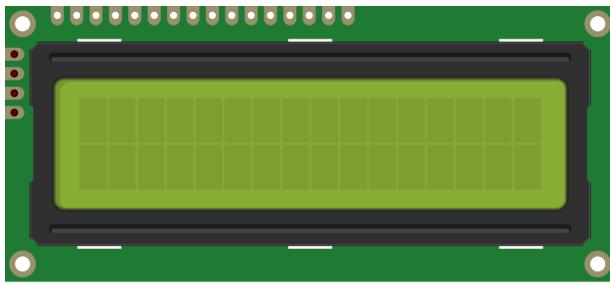
Chapter 20 LCD1602

In this chapter, we will learn the LCD1602 display.

Project 20.1 I2C LCD1602

This project realizes the display of the current ambient temperature on the LCD1602 display.

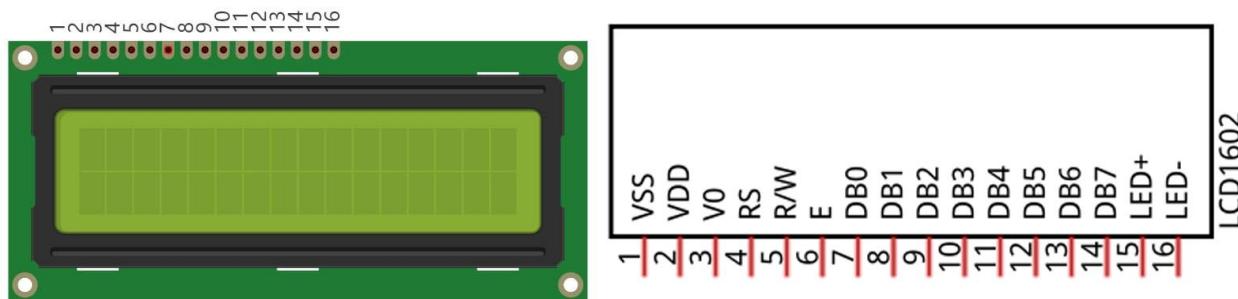
Component list

Microbit x1	Expansion board x1
	
USB cable x2	F/F x4
	
LCD1602	

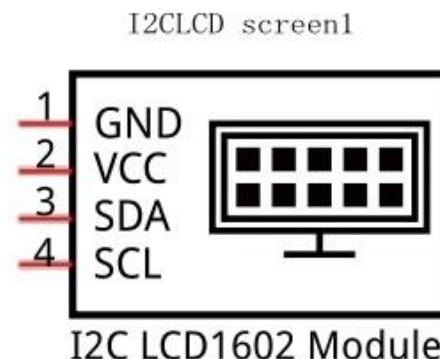
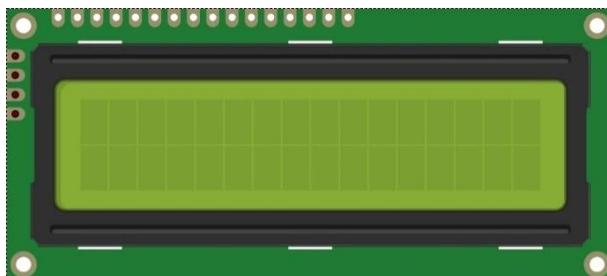
Component knowledge

LCD1602

LCD1602 can display 2 lines of characters in 16 columns. It can display numbers, letters, symbols, ASCII code and so on. As shown below is a monochrome LCD1602 display screen, and its circuit pin diagram:

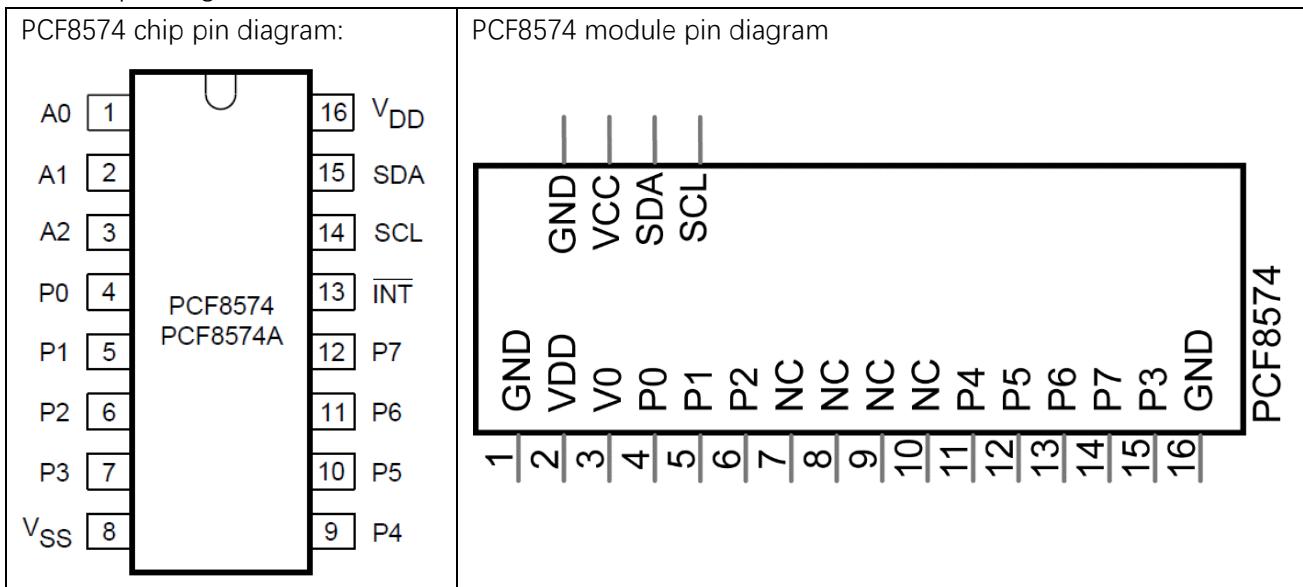


I²C LCD1602 integrates a I²C interface, which connects the serial-input ¶llel-output module to LCD1602.

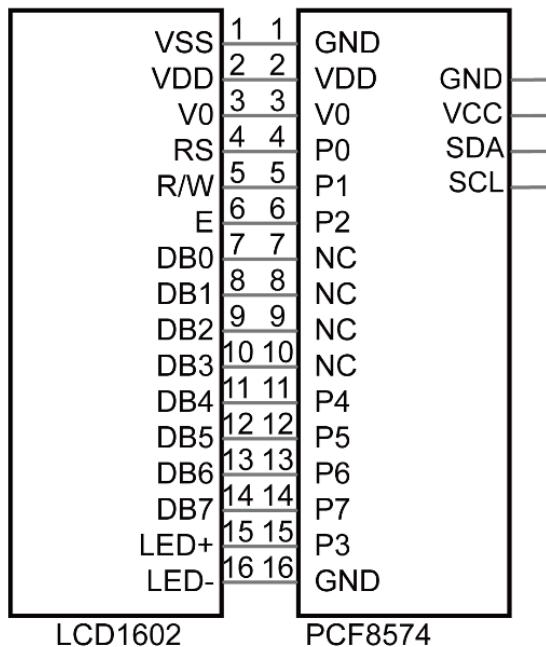


The serial-to-parallel chip used in this module is PCF8574(PCF8574A), and its default I²C address is 0x27(0x3F),

PCF8574 pin diagram:



PCF8574 module pin and LCD1602 pin are corresponding to each other and connected with each other:

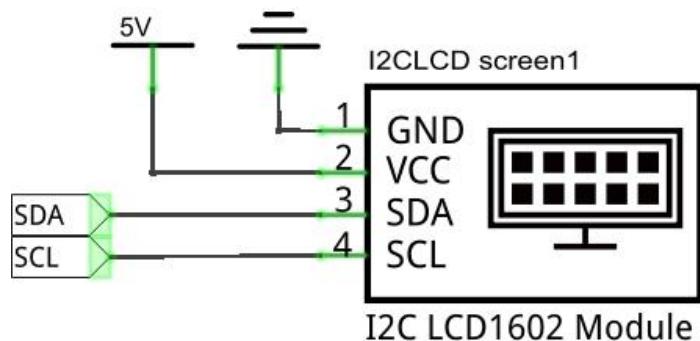


So, we can use just 4 pins to control LCD1602 with 16 pins easily through I2C interface.

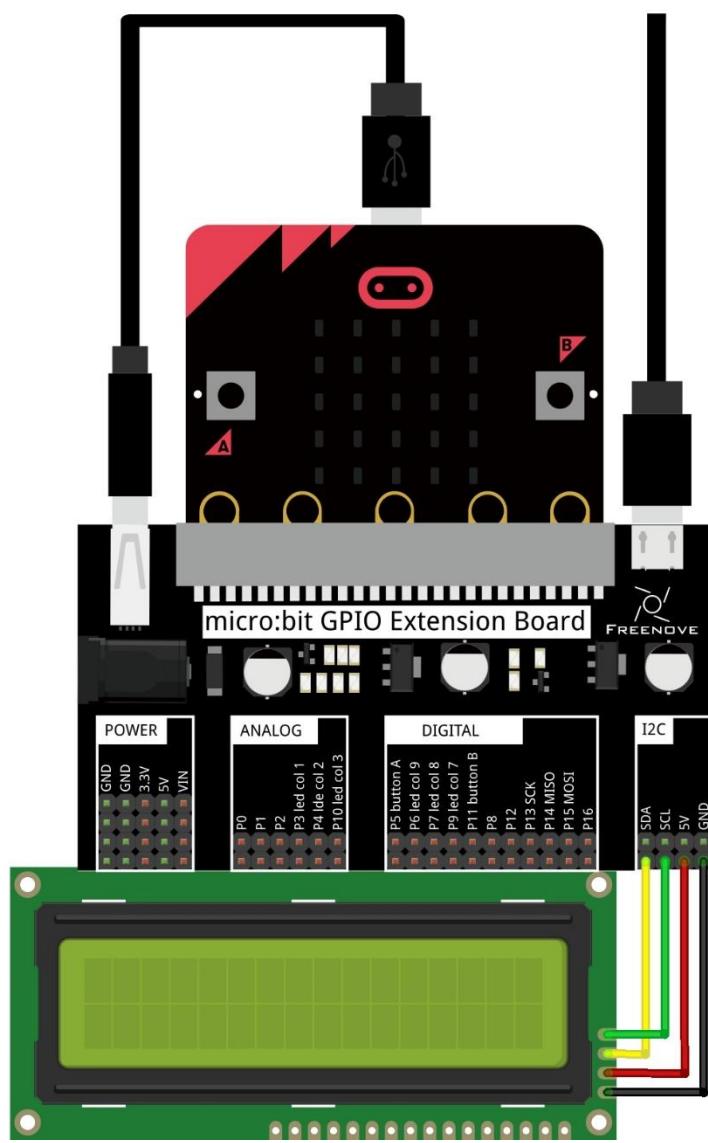
In this project, we will use I2CLCD1602 to display some static characters and dynamic variables.

Circuit

Schematic diagram



Hardware connection



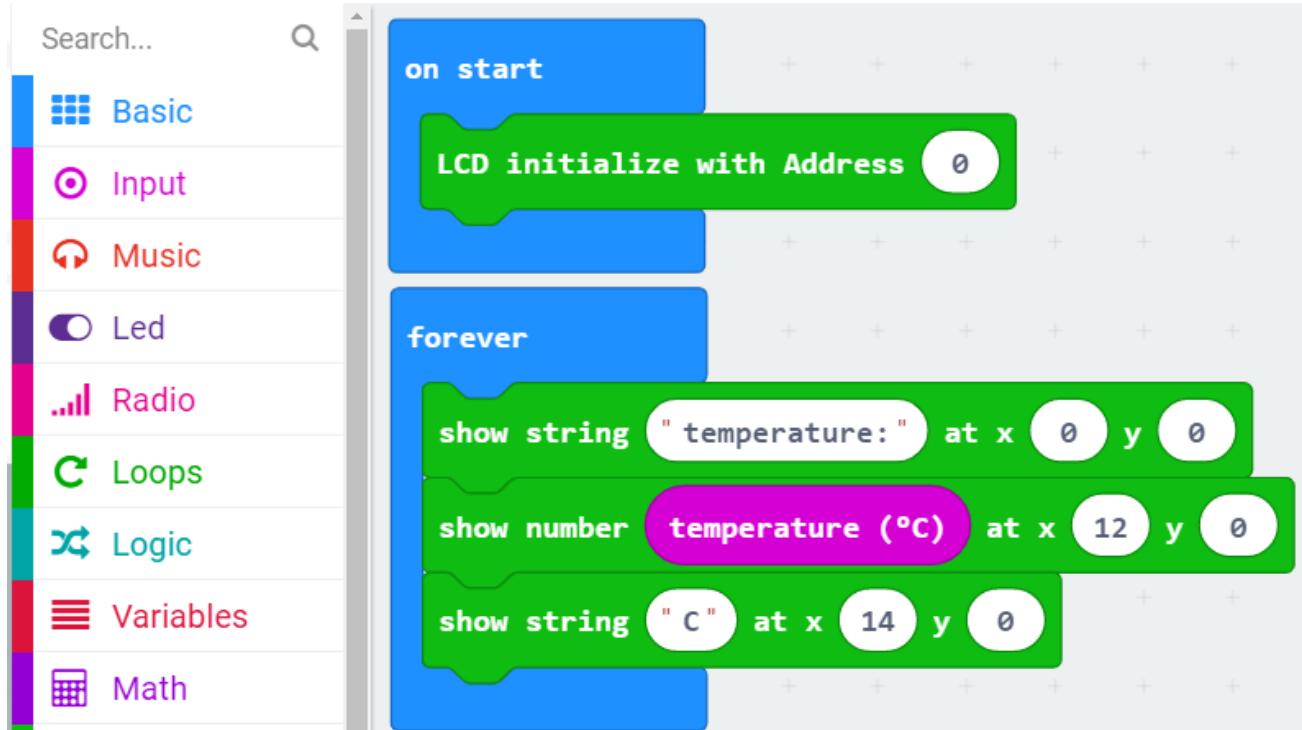
Block code

Open makecode first. Import the .hex file. The path is as below:

[\(How to import project\)](#)

File type	Path	File name
HEX file	../Projects/BlockCode/20.1_LCD1602	LCD1602.hex

After import successfully, the code is shown as below:

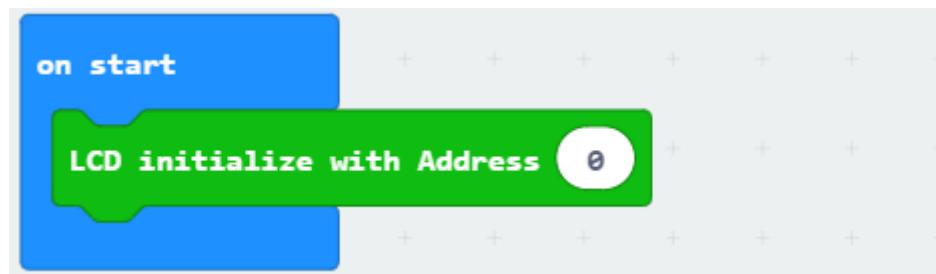


Check the connection of the circuit, confirm that the circuit is connected correctly, and download the code into the micro:bit. Then the LCD screen will display the current ambient temperature.

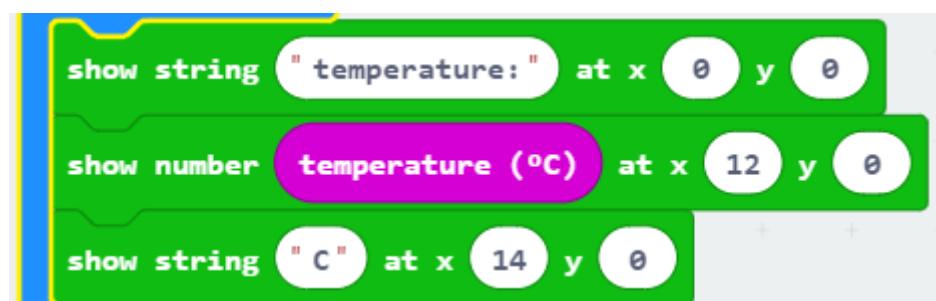
If there is no display or the display is not clear, rotate white knob on back of LCD to adjust the contrast of LCD1602. And observe the change until the screen can display clearly.



LCD initialization, we provide two kinds of LCD screen, you can write I2C address (0x27 or 0x3F) according to the LCD screen you receive. If you enter 0, it will automatically search for the correct i2c address and connect.



Display the temperature of the current environment on the LCD display.

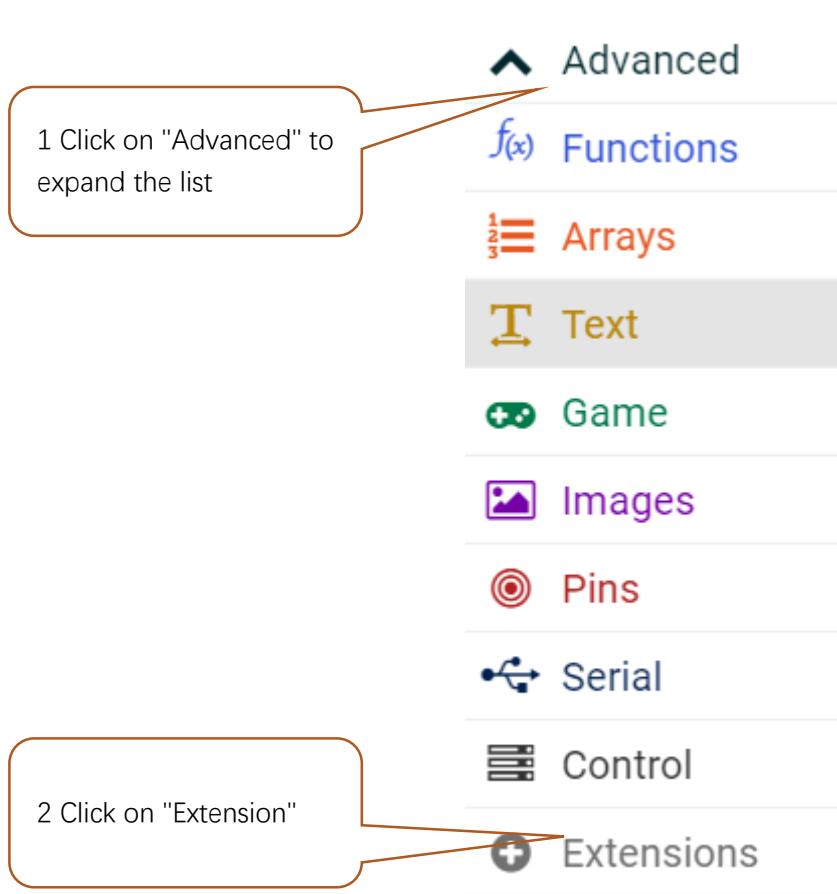


Reference

Block	Function
LCD initialize with Address 0	LCD initialization, input LCD I2C address (0x27 or 0x3F). If you enter 0, it will automatically find the correct address and connect.
show number 10 at x 0 y 0	The LCD screen displays the number in the x column, y column of the screen.
show string "Hello" at x 0 y 0	The LCD screen displays the string in the x column, the y column of the screen

Extensions

If you want to import the LCD1602 expansion block in a new project, follow the steps below to add it.



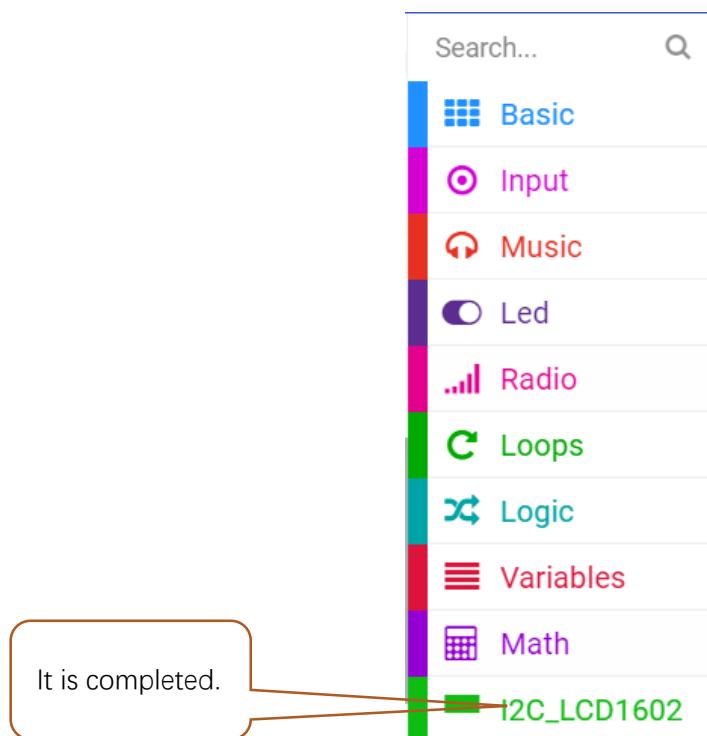
Enter "i2Clcd1602" to search.

Go back

Extensions

i2cLCD1602

bluetooth Bluetooth services	devices BETA - Camera, remote control and other Bluetooth services. App	radio-broadcast Adds new blocks for message communication in the radio	servo A micro-servo library	neopixel AdaFruit NeoPixel driver
microturtle A LOGO-like turtle library	tinkercademy-tinker-kit Tinkercademy package for the Tinker Kit	robotbit Extension for Kittenbot Robotbit	sonar A MakeCode package to use sonar sensors	BitBot A BitBot package for pxt-microbit



Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/20.1_LCD1602	LCD1602.py

After the code is loaded, as shown below. Import the "I2C_LCD1602_Class.py" file to micro:bit before downloading the code.

([How to import?](#))



```

1 from microbit import *
2 from I2C_LCD1602_Class import *
3 lcd = I2C_LCD1602(0x27)
4 while True:
5     lcd.puts("temperature:"+str(temperature()), 0, 0)

```

After the "I2C_LCD1602_Class.py" file is imported, check the connection of the circuit and confirm that the circuit is connected correctly. Download the code into the micro:bit and the LCD screen will display the current ambient temperature.

If there is **no display or the display is not clear, rotate white knob** on back of LCD to **adjust the contrast** of LCD1602. And observe the change until the screen can display clearly.



The following is the program code:

```
1 from microbit import *
2 from I2C_LCD1602_Class import *
3 lcd = I2C_LCD1602(0x27)
4 while True:
5     lcd.puts("temperature:"+str(temperature()), 0, 0)
```

Export everything from the I2C_LCD1602_Class module, create the object lcd of the I2C_LCD1602 class, and enter the I2C address of the LCD screen. Change the I2C address according to the LCD type.

If the chip of LCD is PCF8574 T, the i2c address is 0x27.

If the chip of LCD is PCF8574A T, the i2c address is 0x3F.

```
from I2C_LCD1602_Class import *
lcd = I2C_LCD1602(0x27)
```

Read the temperature data, then call the puts function in the I2C_LCD1602 class to display the temperature on the LCD screen.

```
lcd.puts("temperature:"+str(temperature()), 0, 0)
```

Reference

puts(String, x, y)

This function is defined in the I2C_LCD1602 class. The function is to display the string on the LCD screen x column, y row, x range is 0-15, y range is 0-1.

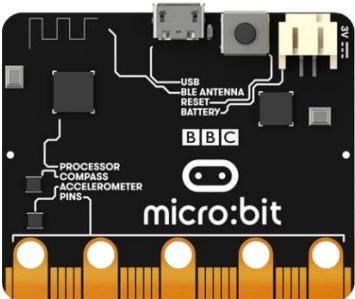
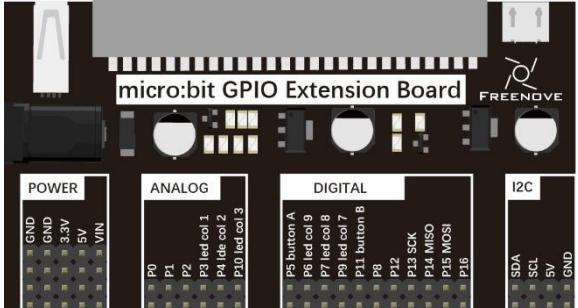
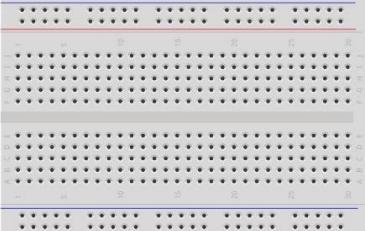
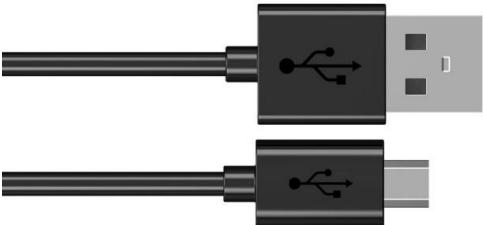
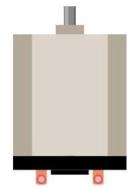
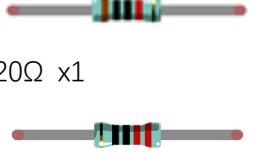
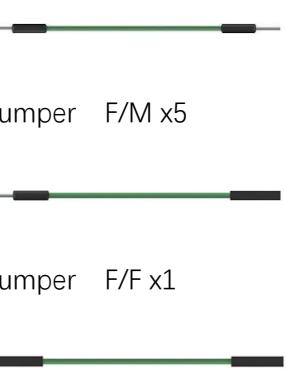
Chapter 21 Motor

In this chapter, we will learn the comprehensive application of motor.

Project 21.1 Rlay And Motor

This project will control the relay to drive the motor.

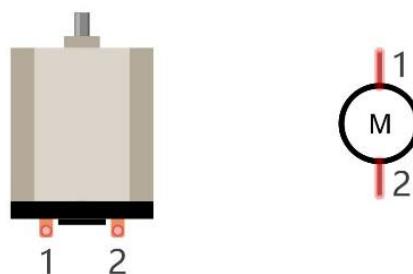
Component list

Microbit x1	Expansion board x1		
			
Breakboard x1	USB cable x2		
			
NPN(8050)transistor x 1	Motor x1	1kΩ x1 220Ω x1	Jumper M/M x4 Jumper F/M x5 Jumper F/F x1
			
Relay x1	Led x1	Diode x1	
			

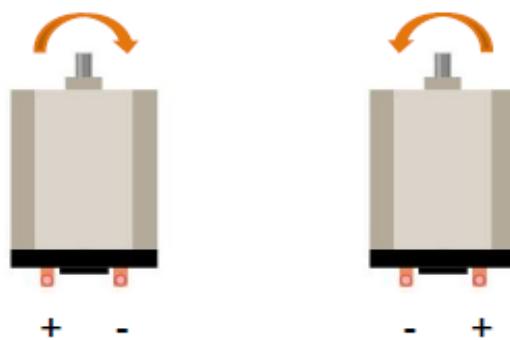
Component knowledge

Motor

Motor is a device that converts electrical energy into mechanical energy. Motor consists of two parts: stator and rotor. When motor works, the stationary part is stator, and the rotating part is rotor. Stator is usually the outer case of motor, and it has terminals to connect to the power. Rotor is usually the shaft of motor, and can drive other mechanical devices to run. Diagram below is a small DC motor with two pins.



When motor get connected to the power supply, it will rotate in one direction. Reverse the polarity of power supply, then motor rotates in opposite direction.



Relay

Relay is a safe switch which can use low power circuit to control high power circuit. It consists of electromagnet and contacts. The electromagnet is controlled by low power circuit and contacts is used in high power circuit. When the electromagnet is energized, it will attract contacts.

The following is a principle diagram of common relay and the feature and circuit symbol of 5V relay used in this project:

Diagram	Feature:	Symbol
	<p>24 6 13 5</p> <p>Relay DC 5V 3A 120VAC 3A 24VDC</p>	

Pin 5 and pin 6 are connected to each other inside. When the coil pin 3 and 4 get connected to 5V power supply, pin 1 will be disconnected to pin 5&6 and pin 2 will be connected to pin 5&6. So pin 1 is called close end, pin 2 is called open end.

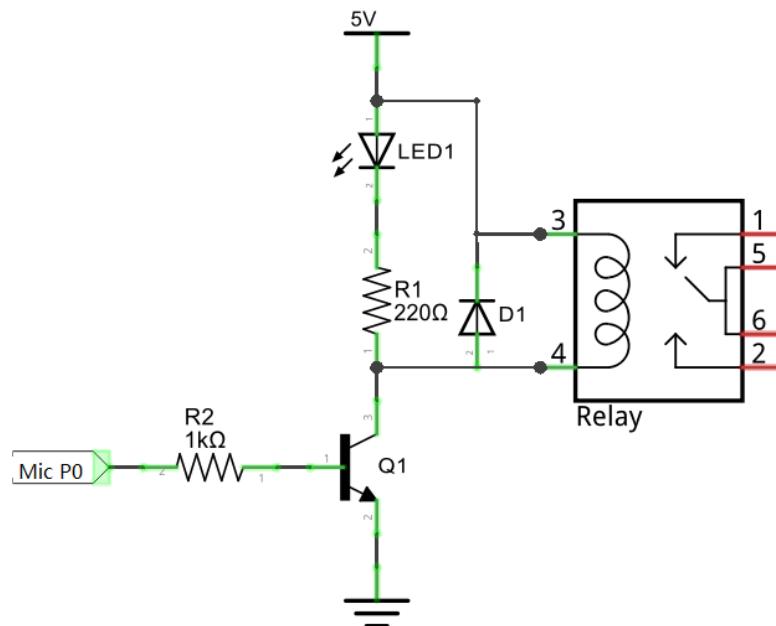
Inductor

The unit of inductance(L) is the henry (H). $1\text{H}=1000\text{mH}$, $1\text{mH}=1000\mu\text{H}$.

Inductor is an energy storage device that converts electrical energy into magnetic energy. Generally, it consists of winding coil, with a certain amount of inductance. Inductor will hinder the changing current passing through the inductor. When the current passing through inductor increases, it will attempt to hinder the increasing trend of current; and when the current passing through the inductor decreases, it will attempt to hinder the decreasing trend of current. So the current passing through inductor is not transient.

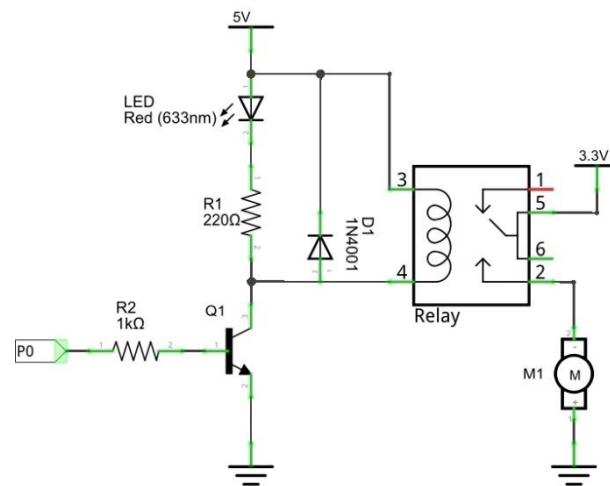


The reference circuit for relay is as follows. The coil of relay can be equivalent to inductor, when the transistor disconnects power supply of the relay, the current in the coil of the relay can't stop immediately, causing an impact on power supply. So a parallel diode will get connected to both ends of relay coil pin in reversing direction, then the current will pass through diode, avoiding the impact on power supply.

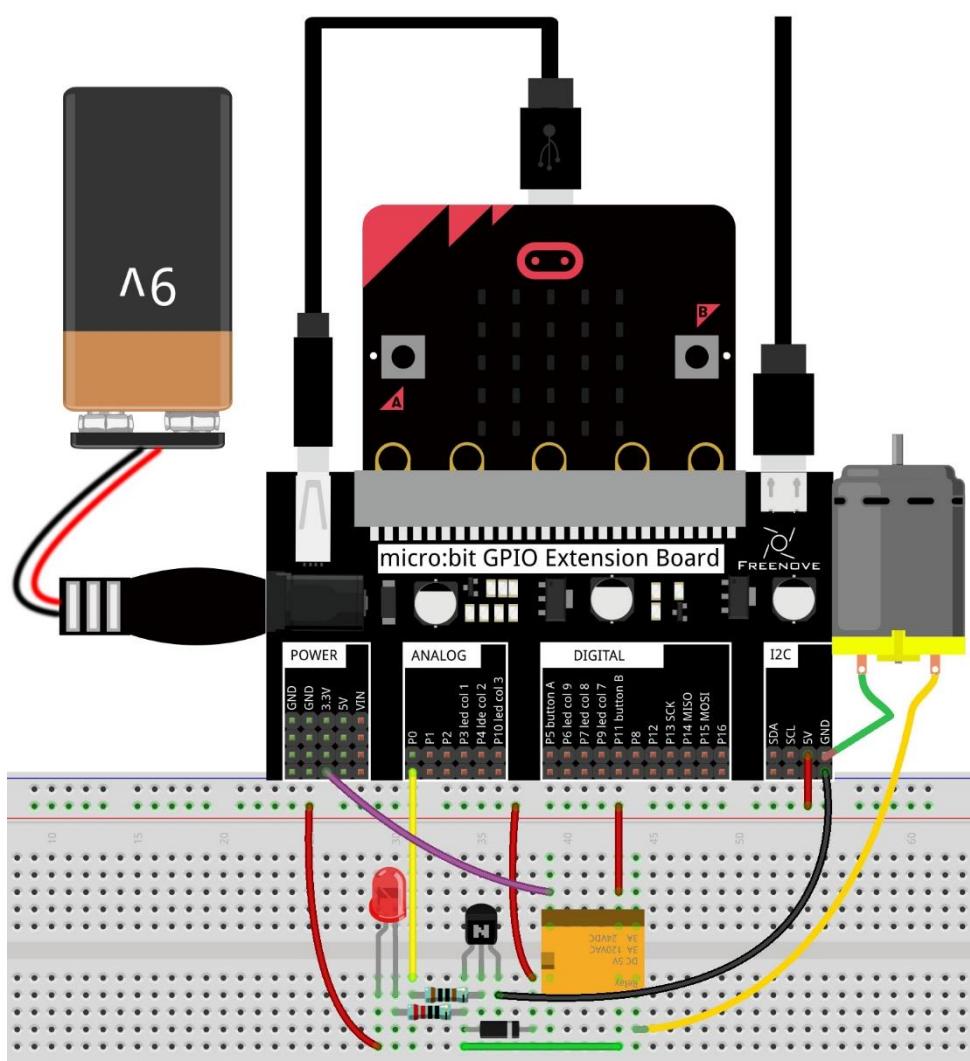


Circuit

Schematic diagram



Hardware connection



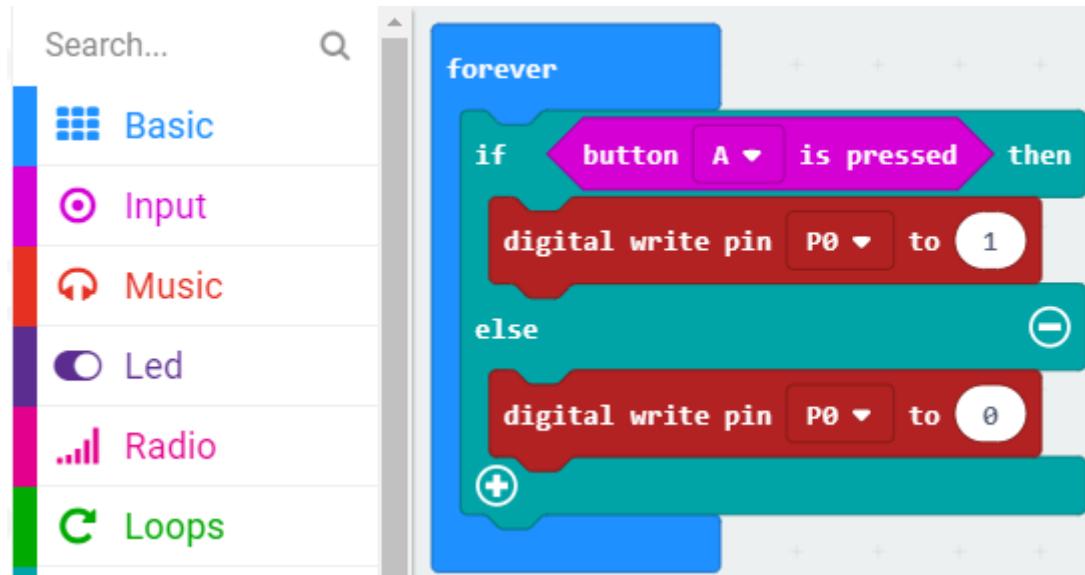
Block code

Open makecode first. Import the .hex file. The path is as below:

[\(How to import project\)](#)

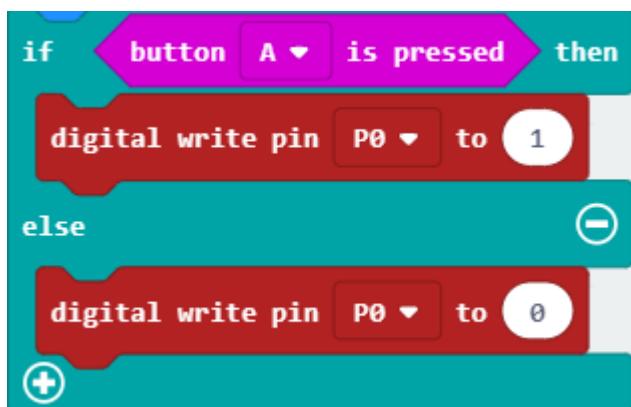
File type	Path	File name
HEX file	./Projects/BlockCode/21.1_Relay	Relay.hex

After import successfully, the code is shown as below:



Check the connection of the circuit, confirm that the circuit is connected correctly, and download the code into micro:bit. Press button A, then the motor starts to rotate. Release the button, then the motor stops.

It is judged whether the button A is pressed. If pressed, P0 outputs a high level, the relay is turned on, and the motor is running; if it is not pressed, P0 outputs a low level, and the motor does not run.

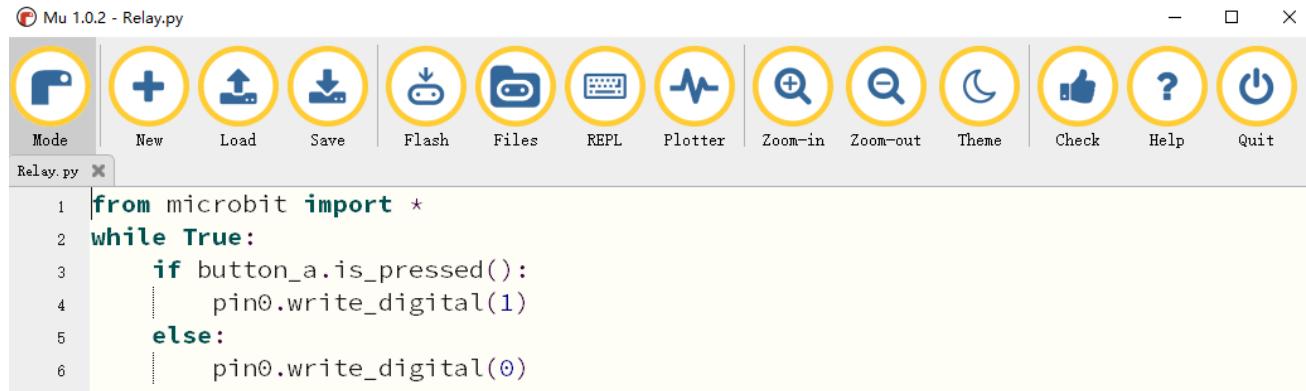


Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	./Projects/PythonCode/21.1_Relay	Relay.py

After load successfully, the code is shown as below:



The screenshot shows the Mu 1.0.2 IDE interface. The title bar says "Mu 1.0.2 - Relay.py". The toolbar includes icons for Mode (Micro:bit), New, Load, Save, Flash, Files, REPL, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The main code editor window displays the following Python code:

```

1 from microbit import *
2 while True:
3     if button_a.is_pressed():
4         pin0.write_digital(1)
5     else:
6         pin0.write_digital(0)

```

Check the connection of the circuit, confirm that the circuit is connected correctly, and download the code into micro:bit. Press button A, then the motor starts to rotate. Release the button, then the motor stops.

The following is the program code:

```

1 from microbit import *
2 while True:
3     if button_a.is_pressed():
4         pin0.write_digital(1)
5     else:
6         pin0.write_digital(0)

```

If A is pressed, P0 outputs a high level, the relay is turned on, and the motor is running. If it is not pressed, P0 outputs a low level, and the motor does not run.

```

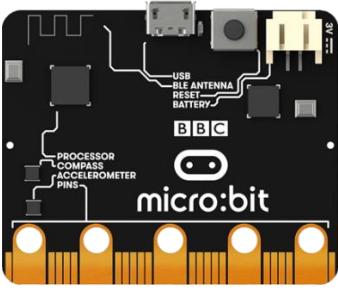
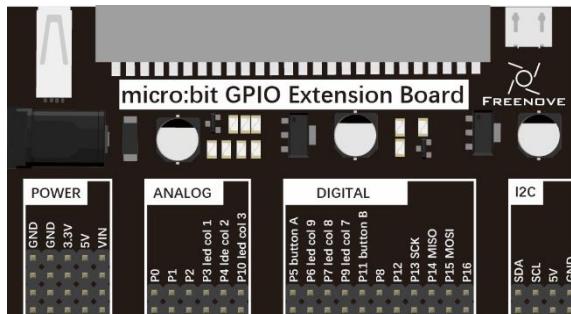
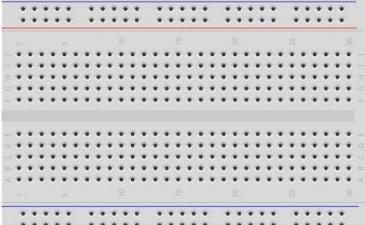
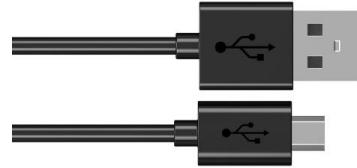
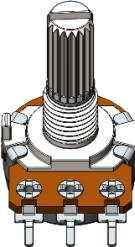
if button_a.is_pressed():
    pin0.write_digital(1)
else:
    pin0.write_digital(0)

```

Project 21.2 Potentiometer and Motor

In this project, a potentiometer is used to control motor.

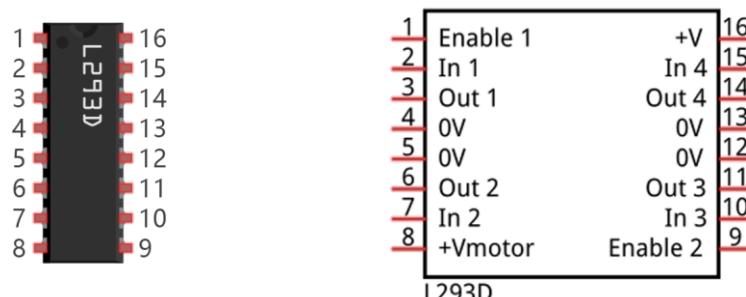
Component list

Microbit x1	Expansion board x1
	
Breakboard x1	USB cable x2
	
F/M x9 M/M x3	Rotary potentiometer x1
	
Motor x1	L293D x1
	

Component knowledge

L293D

L293D is a chip integrated with 4-channel motor drive. You can drive a unidirectional motor with 4 ports or a bi-directional motor with 2 port or a stepper motor.



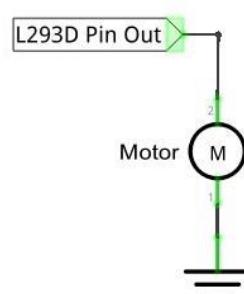
Port description of L293D module is as follows:

Pin name	Pin number	Description
In x	2, 7, 10, 15	Channel x digital signal input pin
Out x	3, 6, 11, 14	Channel x output pin, input high or low level according to In x pin, get connected to +Vmotor or 0V
Enable1	1	Channel 1 and channel 2 enable pin, high level enable
Enable2	9	Channel 3 and channel 4 enable pin, high level enable
0V	4, 5, 12, 13	Power cathode (GND)
+V	16	Positive electrode (VCC) of power supply, supply voltage 4.5~36V
+Vmotor	8	Positive electrode of load power supply, provide power supply for the Out pin x, the supply voltage is +V~36V

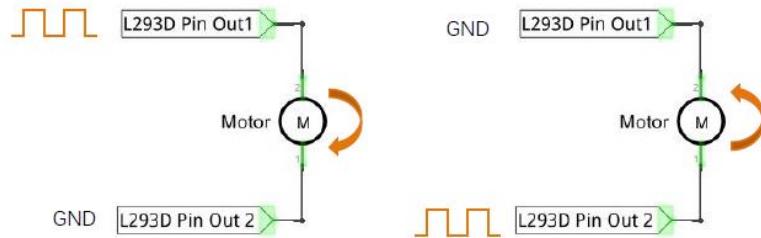
For more details, please see datasheet.

When using L293D to drive DC motor, there are usually two kinds of connection.

Following connection uses one channel, and it can control motor speed through PWM, but the motor can only rotate in one direction.



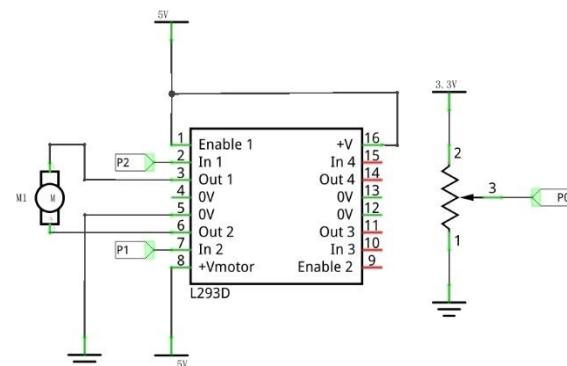
Following connection uses two channels: one channel outputs PWM wave, and another channel connects GND, so you can control the speed of motor. When these two channel signals are exchanged, the current direction of the motor can be reversed, and the motor will rotate in reverse direction. This can not only control the speed of motor, but also can control the steering of motor.



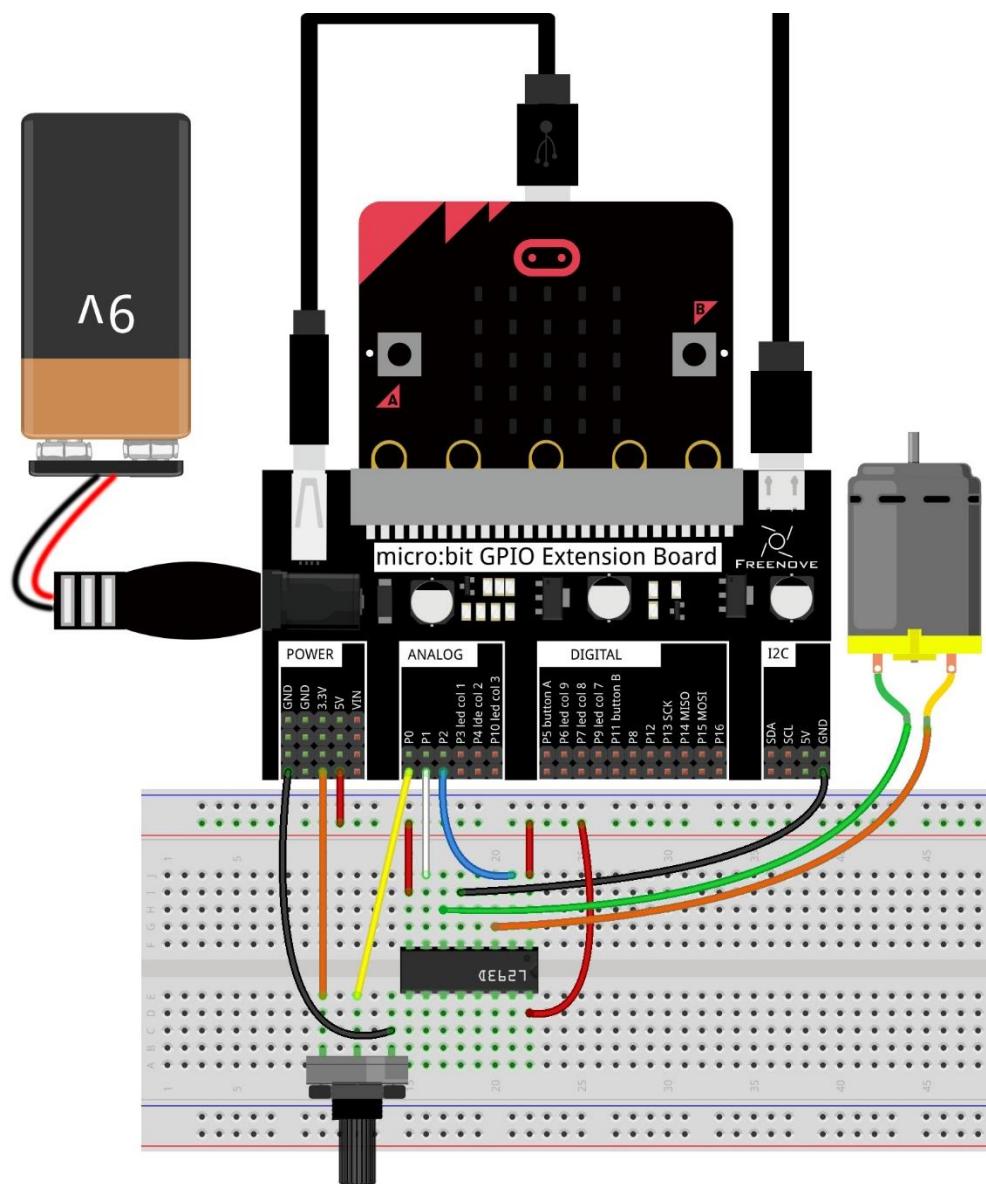
In actual use, motor is usually connected to the channel 1 and 2, output different level to in1 and in2 to control the rotation direction of the motor, and output PWM wave to Enable1 port to control the motor rotation speed. Or, get motor connected to the channel 3 and 4, output different level to in3 and in4 to control the motor's rotation direction, and output PWM wave to Enable2 pin to control the motor rotation speed.

Circuit

Schematic diagram



Hardware connection



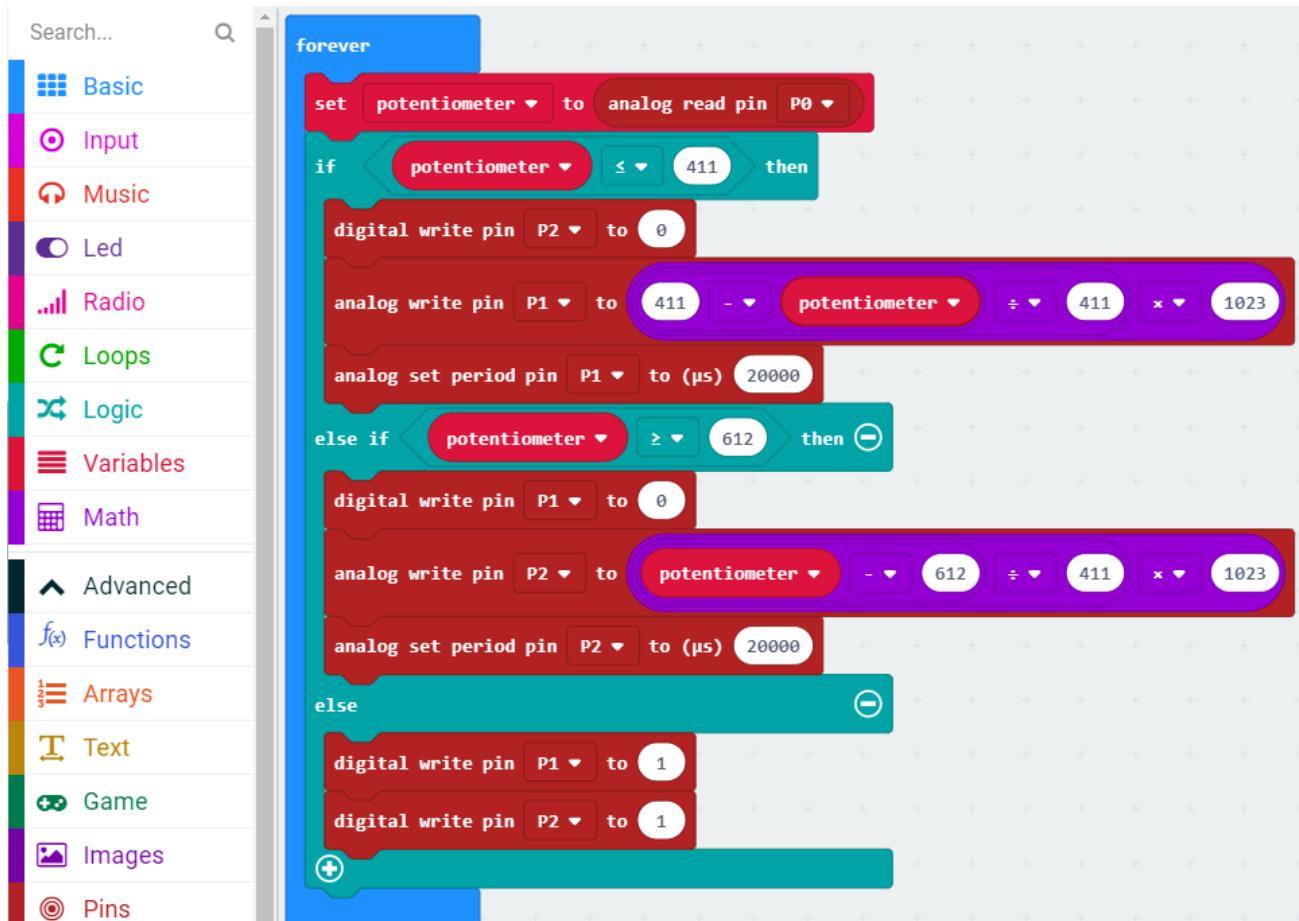
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

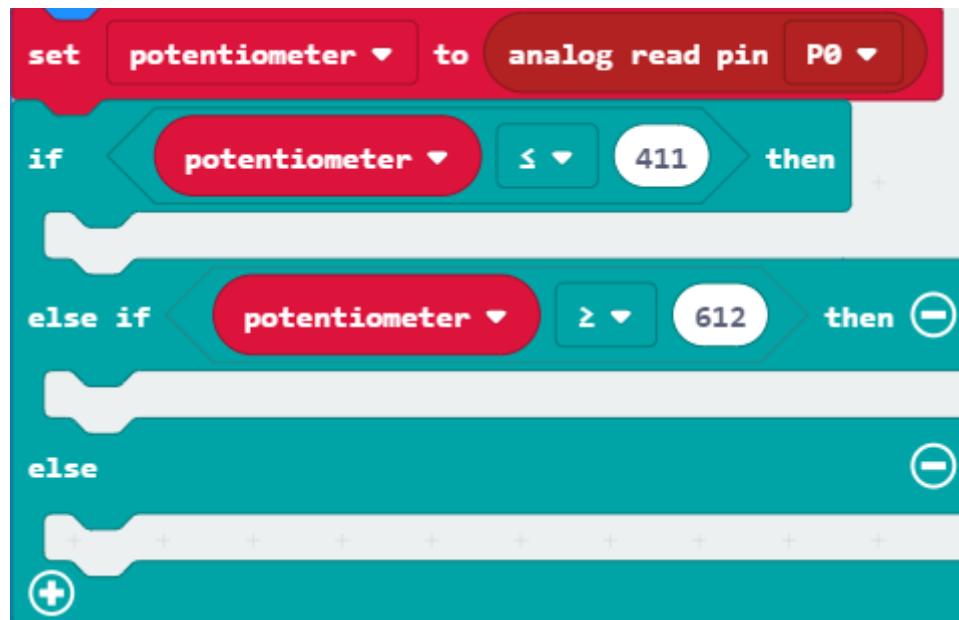
File type	Path	File name
HEX file	../Projects/BlockCode/21.2_Motor	Motor.hex

After import successfully, the code is shown as below:



Check the connection of the circuit and confirm that the circuit is connected correctly. Download the code into the micro:bit and turn the potentiometer. When the potentiometer is in the middle position, the motor stops rotating. When the potentiometer gets away from middle position, the motor speed increases. The potentiometer moves to the limit and the motor speed reaches its maximum value. When the potentiometer is on a different side, the direction of motor rotation is different.

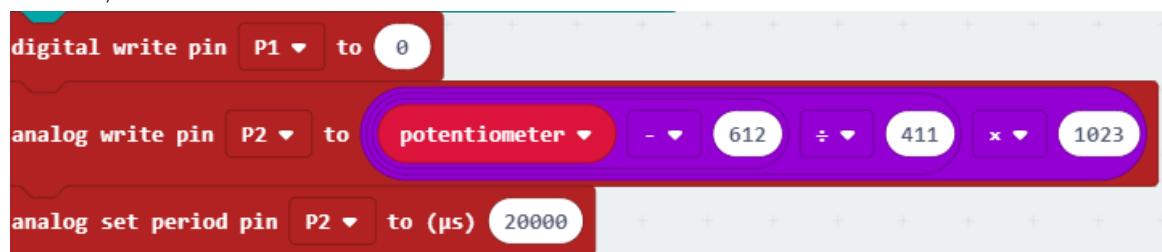
Read the analog value of the P0 pin of the potentiometer. When the analog value is less than 411, the motor rotates forward. When the analog value is greater than 612, the motor reverses. When the analog value is between 411 and 612, the motor does not rotate.



Set P2 to low level, P1 output PWM signal with period of 20ms, duty cycle changes with potentiometer variable, then motor rotates forward.



Set P1 to low level, P2 output PWM signal with period of 20ms, duty cycle changes with potentiometer variable, then motor reverse.



When P1, P2 pin output high level, motor does not rotate.



Reference

Block	Function
	Write an analog signal (0 through 1023) to the pin you say.
	Configure the period of Pulse Width Modulation (PWM) on the specified analog pin. Before you call this function, you should set the specified pin as analog.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/21.2_Motor	Motor.py

After load successfully, the code is shown as below:



```

1 from microbit import *
2 while True:
3     potentiometer=pin0.read_analog()
4     if potentiometer<=411:
5         pin2.write_digital(0)
6         pin1.write_analog((411-potentiometer)/411*1023)
7         pin1.set_analog_period(20)
8     elif potentiometer>=612:
9         pin1.write_digital(0)
10        pin2.write_analog((potentiometer-612)/411*1023)
11        pin2.set_analog_period(20)
12    else:
13        pin1.write_digital(1)
14        pin2.write_digital(1)

```

Check the connection of the circuit and confirm that the circuit is connected correctly. Download the code into the micro:bit and turn the potentiometer. When the potentiometer is in the middle position, the motor stops rotating. When the potentiometer gets away from middle position, the motor speed increases. The potentiometer moves to the limit and the motor speed reaches its maximum value. When the potentiometer is on a different side, the direction of motor rotation is different.

The following is the program code:

```

1 from microbit import *
2 while True:
3     potentiometer=pin0.read_analog()
4     if potentiometer<=411:
5         pin2.write_digital(0)
6         pin1.write_analog((411-potentiometer)/411*1023)
7         pin1.set_analog_period(20)
8     elif potentiometer>=612:
9         pin1.write_digital(0)
10        pin2.write_analog((potentiometer-612)/411*1023)
11        pin2.set_analog_period(20)
12    else:
13        pin1.write_digital(1)
14        pin2.write_digital(1)

```

Read the analog value of the P0 pin of the potentiometer. When the analog value is less than 411, the motor rotates forward. When the analog value is greater than 612, the motor reverses. When the analog value is between 411 and 612, the motor does not rotate.

```
potentiometer=pin0.read_analog()
if potentiometer<=411:

elif potentiometer>=612:

else:
```

Set P2 to low level, P1 output PWM signal with period of 20ms, duty cycle changes with potentiometer variable, then motor rotates forward.

```
pin2.write_digital(0)
pin1.write_analog((411-potentiometer)/411*1023)
pin1.set_analog_period(20)
```

Set P1 to low level, P2 output PWM signal with period of 20ms, duty cycle changes with potentiometer variable, then motor reverse.

```
pin1.write_digital(0)
pin2.write_analog((potentiometer-612)/411*1023)
pin2.set_analog_period(20)
```

When P1, P2 pin output high level, motor does not rotate.

```
pin1.write_digital(1)
pin2.write_digital(1)
```

Reference

`pin.set_analog_period(int)`

sets the period of the PWM output of the pin in milliseconds
see https://en.wikipedia.org/wiki/Pulse-width_modulation

`pin.write_analog(value)`

value is between 0 and 1023

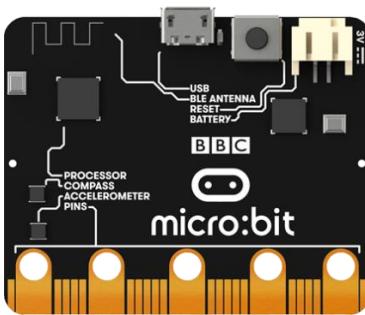
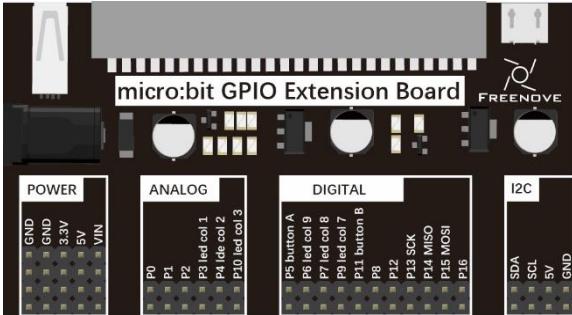
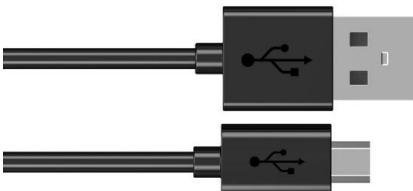
Chapter 22 Servo

In this chapter, we will learn a motor that can rotate to a specific angle - the servo.

Project 22.1 Sweep

This project will uses servo.

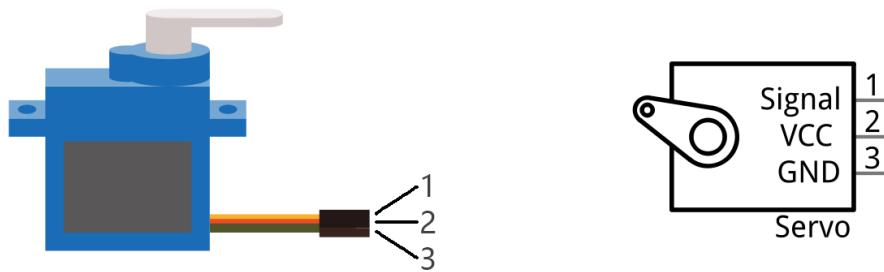
Component list

Microbit x1	Expansion board x1
	
USB cable x2	Servo x1
	
Jumper F/M x3	

Component knowledge

Servo

Servo is an auto-control system, consisting of DC motor, reduction gear, sensor and control circuit. Usually, it can rotate in the range of 180 degrees. Servo can output larger torque and is widely used in model airplane, robot and so on. It has three lines, including two for electric power line positive (2-VCC, red), negative (3-GND, brown), and the signal line (1-Signal, orange).



We use 50Hz PWM signal with a duty cycle in a certain range to drive the servo. The lasting time 0.5ms-2.5ms of PWM single cycle high level corresponds to the servo angle 0 degrees - 180 degree linearly. Part of the corresponding values are as follows:

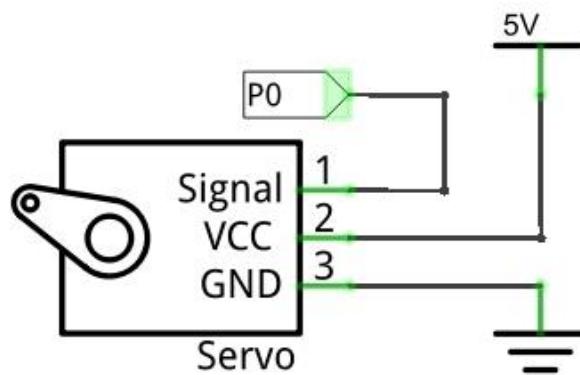
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	90 degree
2ms	135 degree
2.5ms	180 degree

As can be seen from the above table, the servo rotates from 0 to 180 degrees, and the corresponding pulse width is 0.5-2.5ms. Then the analog voltage value written to the micro:bit pin ranges from 25.6 to 128.

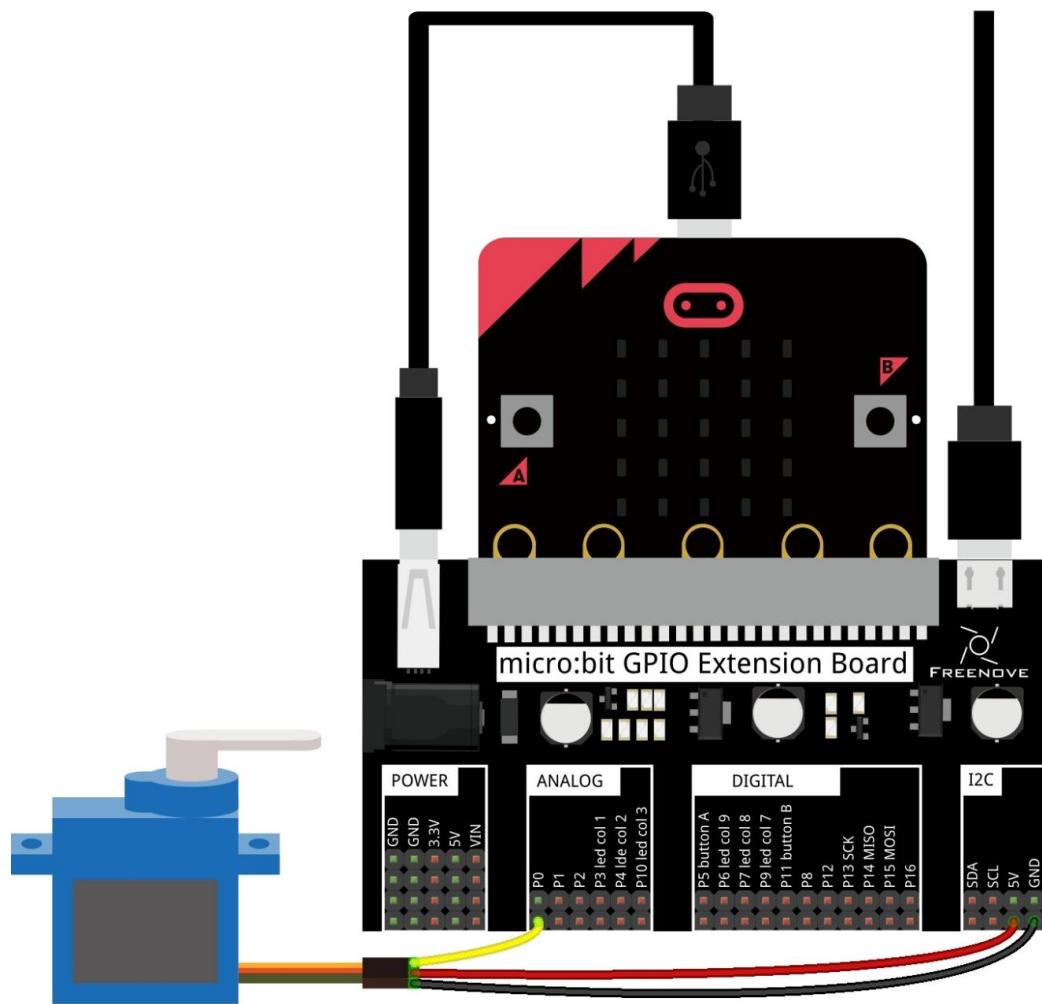
Circuit

This circuit Servo is powered by 5V, and the Micro:bit P0 pin controls the Servo rotation angle.

Diagram schematic



Hardware connection



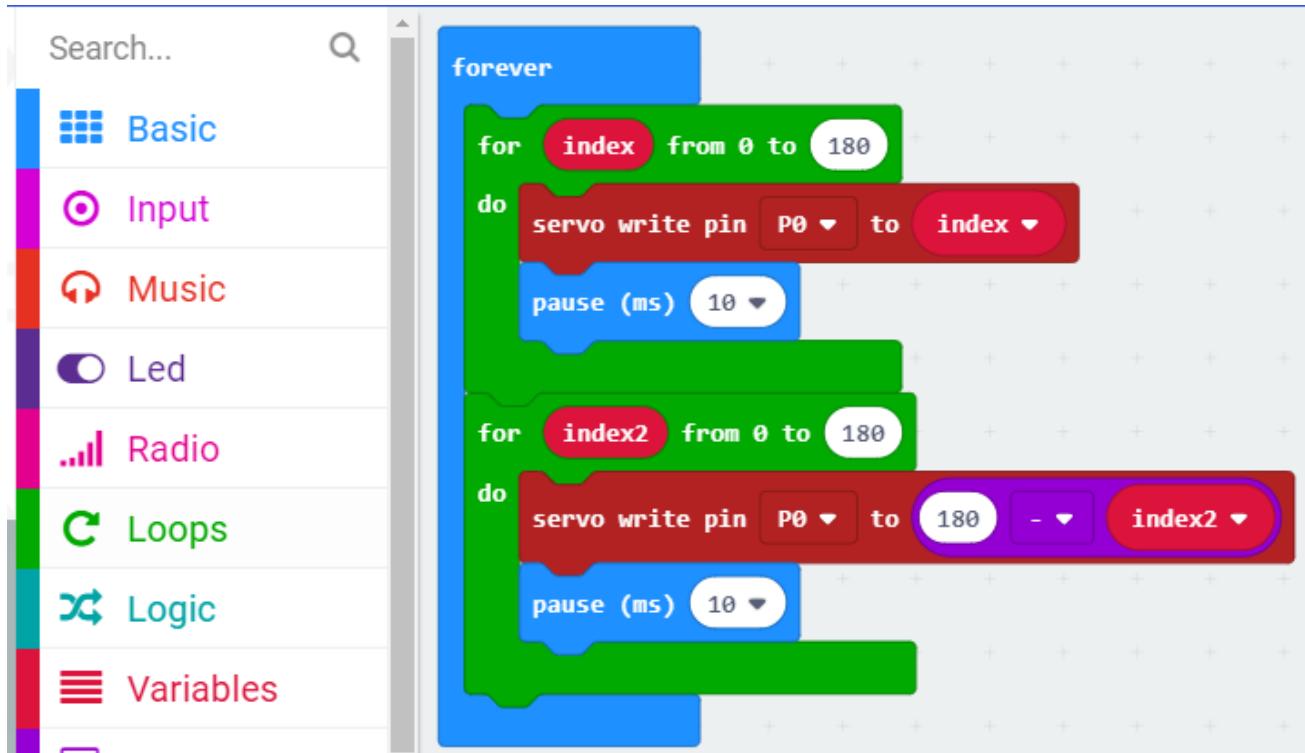
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

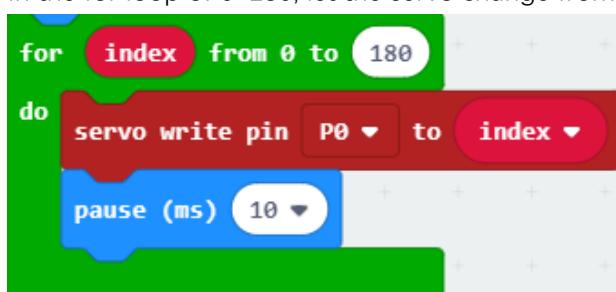
File type	Path	File name
HEX file	./Projects/BlockCode/22.1_Sweep	Sweep.hex

After import successfully, the code is shown as below:



Check the connection of the circuit, confirm that the circuit is connected correctly, and download the code into micro:bit. The servo turns from 0 degrees to 180 degrees, then from 180 degrees to 0 degrees, and cycles.

In the for loop of 0-180, let the servo change from 0 to 180 degrees.



In the for loop of 0-180, take the difference between 180 and index2, and let the servo turns from 180 degrees to 0 degrees.



Reference

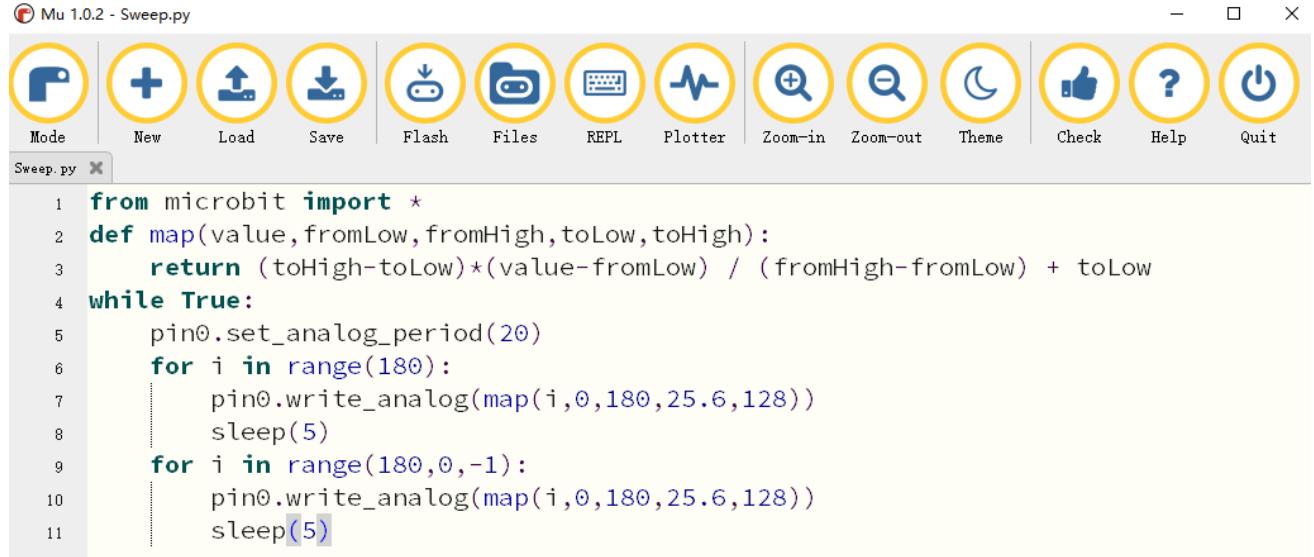
Block	Function
servo write pin P0 to 180	Write a value to the servo on the specified pin and control the shaft.

python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/22.1_Sweep	Sweep.py

After load successfully, the code is shown as below:



```

1 from microbit import *
2 def map(value,fromLow,fromHigh,toLow,toHigh):
3     return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
4 while True:
5     pin0.set_analog_period(20)
6     for i in range(180):
7         pin0.write_analog(map(i,0,180,25.6,128))
8         sleep(5)
9     for i in range(180,0,-1):
10        pin0.write_analog(map(i,0,180,25.6,128))
11        sleep(5)

```

After checking the connection of the circuit and confirming the correct connection of the circuit, download the code into micro:bit. The servo will from 0 degree to 180 degree, and then from 180 degree to 0 degree. Then loop like this.

The following is the program code:

```

1 from microbit import *
2 def map(value, fromLow, fromHigh, toLow, toHigh):
3     return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
4 while True:
5     pin0.set_analog_period(20)
6     for i in range(180):
7         pin0.write_analog(map(i, 0, 180, 25.6, 128))
8         sleep(5)
9     for i in range(180, 0, -1):
10        pin0.write_analog(map(i, 0, 180, 25.6, 128))
11        sleep(5)

```

Define map functions to convert values in one range to values in another range.

```
def map(value, fromLow, fromHigh, toLow, toHigh):  
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
```

Set the period of the PWM signal to 20ms. In a 0-180 for loop, convert the value in the range 0-180 to an analog voltage value in the range of 25.6~128, and then output the corresponding PWM signal to turn the servo from 0 degrees to 180 degrees.

```
pin0.set_analog_period(20)  
for i in range(180):  
    pin0.write_analog(map(i, 0, 180, 25.6, 128))  
    sleep(5)
```

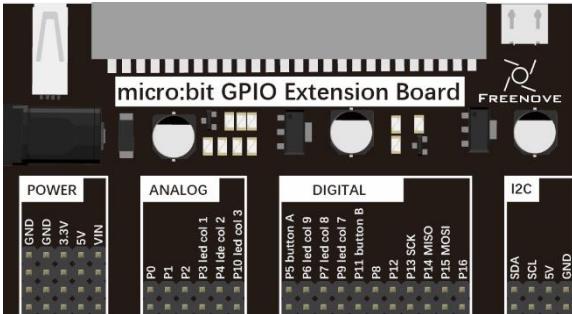
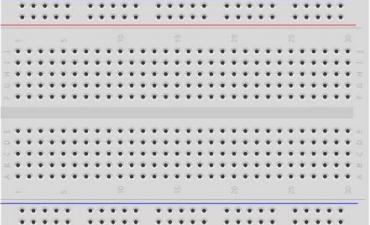
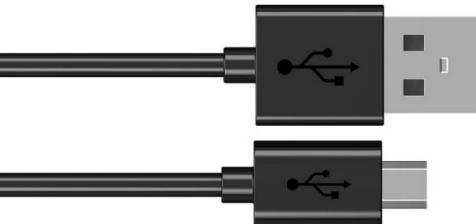
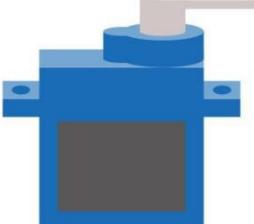
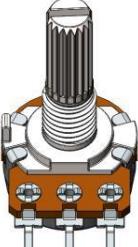
In a 180-0 for loop, convert the value in the range 0-180 to the analog voltage value in the range of 25.6~128, and then output the corresponding PWM signal to turn the servo from 180 degrees to 0 degrees.

```
for i in range(180, 0, -1):  
    pin0.write_analog(map(i, 0, 180, 25.6, 128))  
    sleep(5)
```

Project 22.2 Knob

This project realizes the use of potentiometer to control the rotation angle of Servo.

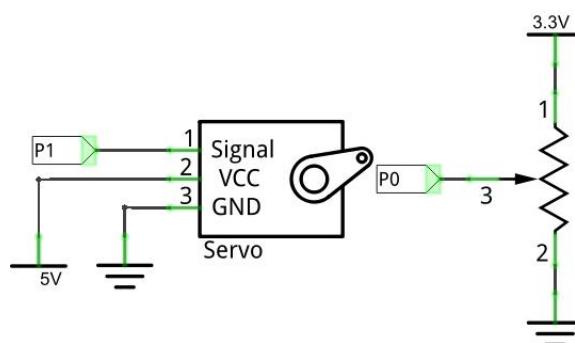
Component list

Microbit x1	Expansion board x1	
		
Breakboard x1	USB cable x2	
		
Servo x1	Rotary potentiometer x1	F/M x6
		

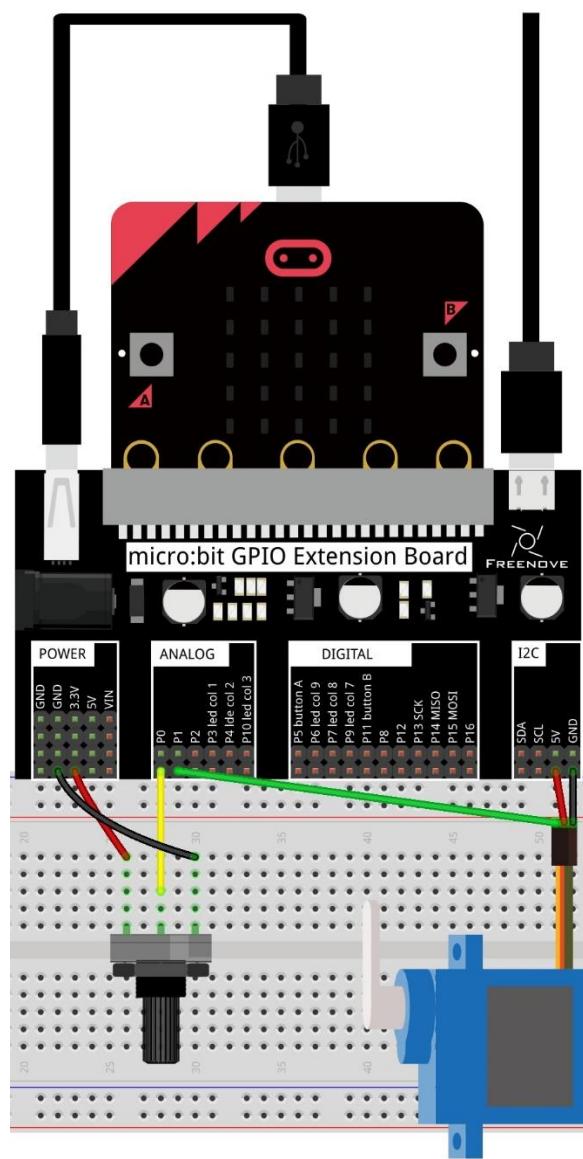
Circuit

The P0 pin of this circuit microbit reads the voltage of the potentiometer, and the P1 pin drives the servo.

Schematic diagram



Hardware connection



Block code

Open makecode first. Import the .hex file. The path is as below:

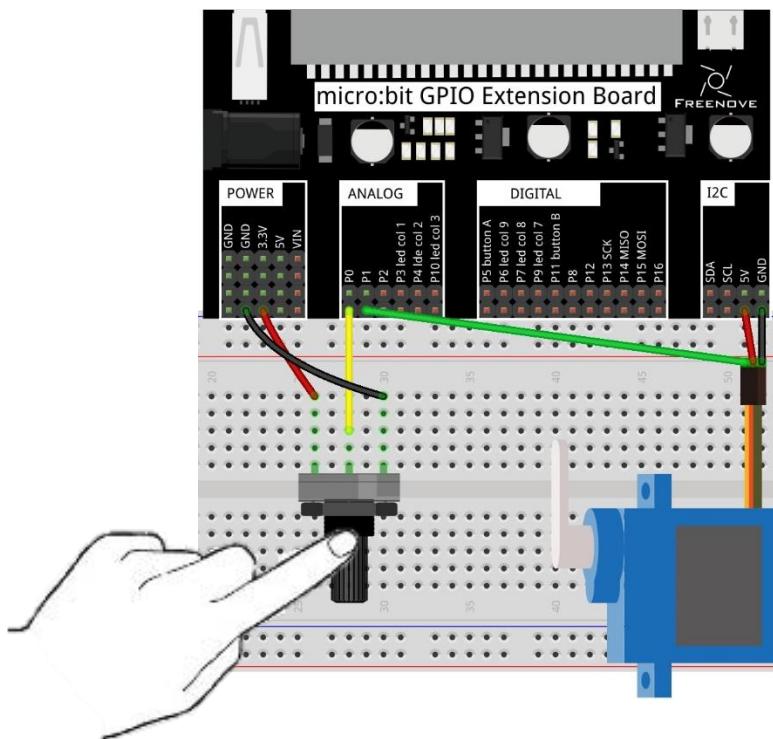
[\(How to import project\)](#)

File type	Path	File name
HEX file	./Projects/BlockCode/22.2_Knob	Knob.hex

After import successfully, the code is shown as below:



Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into the micro:bit, turn the potentiometer, and the servo will follow the rotation.



Read the analog voltage value of the P0 pin, map the analog voltage value in the range of 0-1023 to the angle of the servo in the range of 0-180, and then drive the steering gear to rotate the corresponding angle through the P1 pin.

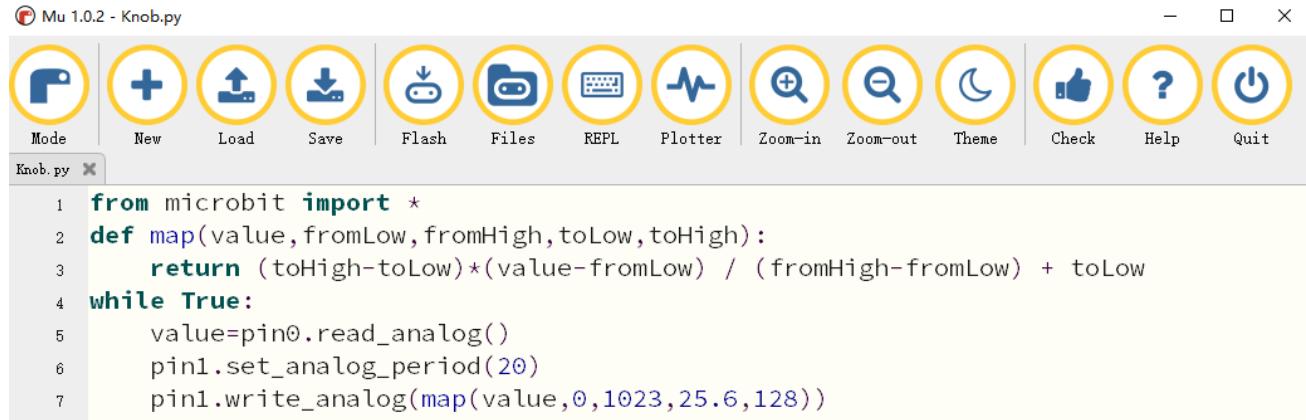


Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	./Projects/PythonCode/22.2_Knob	Knob.py

After load successfully, the code is shown as below:



```

1 from microbit import *
2 def map(value,fromHigh,toLow,toHigh):
3     return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
4 while True:
5     value=pin0.read_analog()
6     pin1.set_analog_period(20)
7     pin1.write_analog(map(value,0,1023,25.6,128))

```

Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into the micro:bit, turn the potentiometer, and the servo will follow the rotation.

The following is the program code:

```

1 from microbit import *
2 def map(value,fromHigh,toLow,toHigh):
3     return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
4 while True:
5     value=pin0.read_analog()
6     pin1.set_analog_period(20)
7     pin1.write_analog(map(value,0,1023,25.6,128))

```

Define map functions to convert values in one range to values in another range.

```

def map(value,fromHigh,toLow,toHigh):
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow

```

Set the period of PWM signal to 20 ms. Read the analog voltage value of P0 foot, convert the analog voltage value in the range of 0-1023 to the analog voltage value in the range of 25.6-128, and then output the corresponding PWM signal to rotate the steering gear at the corresponding angle.

```

value=pin0.read_analog()
pin1.set_analog_period(20)
pin1.write_analog(map(value,0,1023,25.6,128))

```

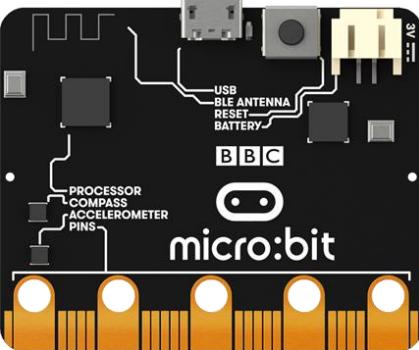
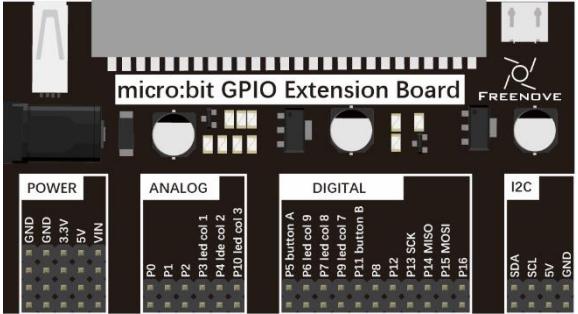
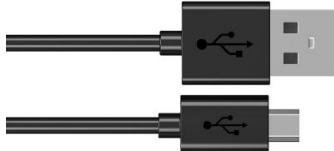
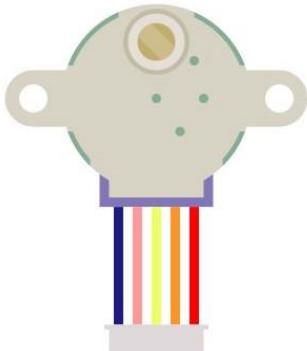
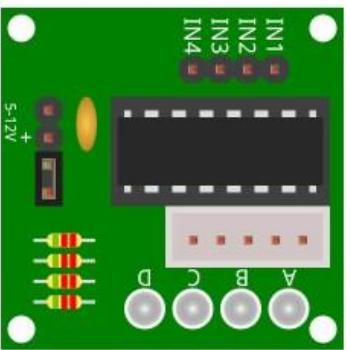
Chapter 23 Stepper Motor

In this chapter, we will learn a new component: stepper motor.

Project 23.1 Stepper Motor

This project uses micro:bit to control stepper motor.

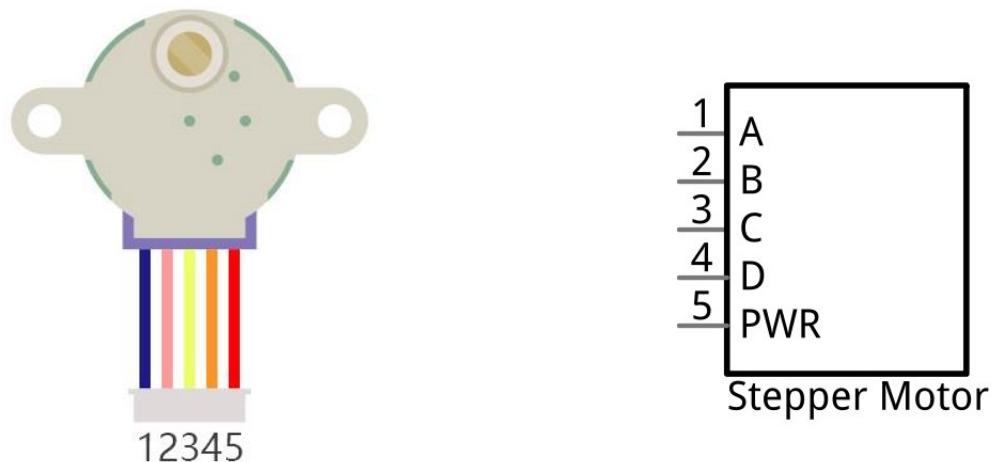
Component list

Microbit x1	Expansion board x1
	
USB cable x2	Jumper F/F x6
	
Stepping Motor x1	ULN2003 Stepper motorDriver x1
	

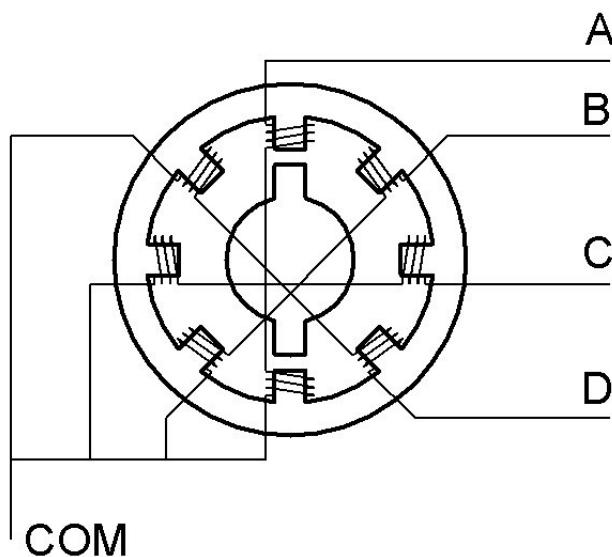
Component knowledge

Stepping Motor

Stepping motor is an open-loop control device which converts the electric pulse signal into angular displacement or linear displacement. In non-overload condition, the speed of the motor and the location of the stop depends only on the pulse signal frequency and pulse number, and not affected by the load changes. A small four-phase deceleration stepping motor is shown as follows:

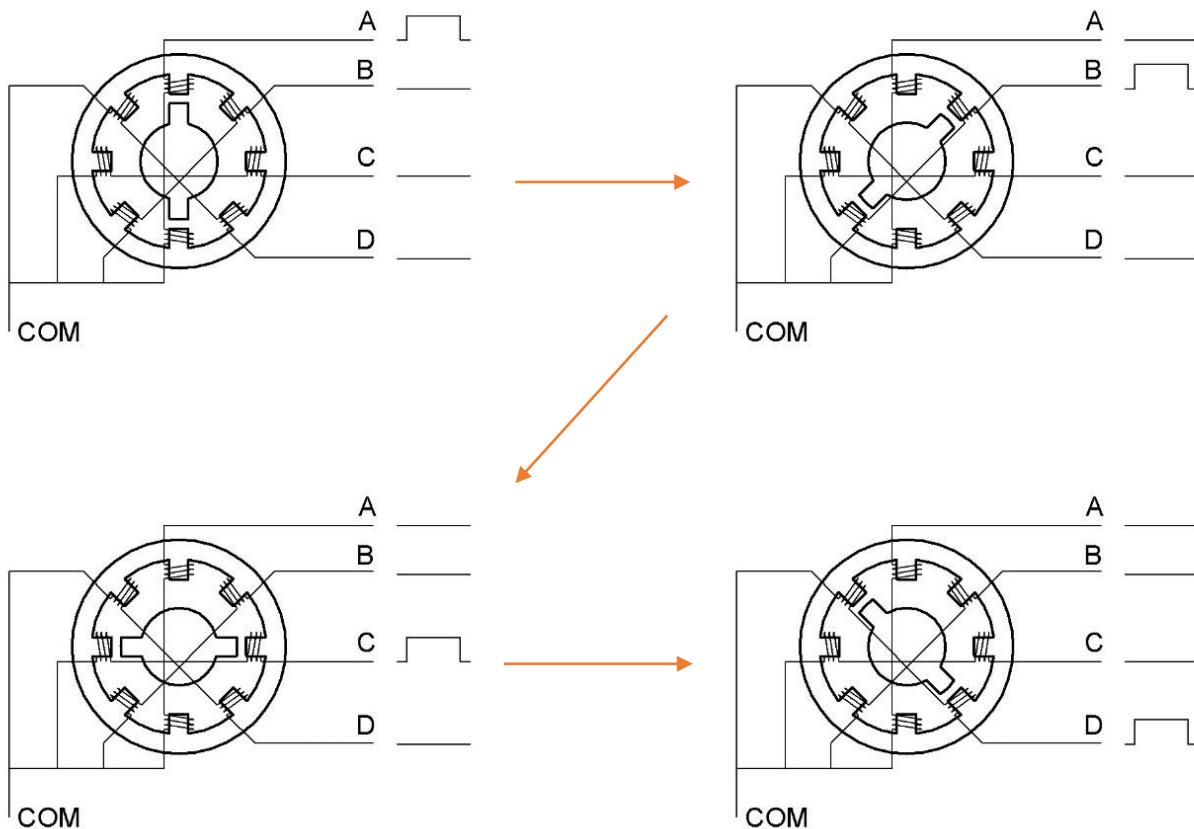


The schematic diagram of four-phase stepping motor is shown below:



The outside piece is the stator and the inside is the rotor of the motor. There are a certain number of coils, usually integer multiple of phases number, in the stator and when powered on, an electromagnet will be formed to attract a convex part (usually iron or permanent magnet) of the rotor. Therefore, the electric motor can be driven by conducting the coils on stator orderly.

A common driving process is as follows:



In the course above, the stepping motor rotates a certain angle once, which is called a step. By controlling the number of rotation steps, you can control the stepping motor rotation angle. By controlling the time between two steps, you can control the stepping motor rotation speed. When rotating clockwise, the order of coil powered on is: A B C D A And the rotor will rotate in accordance with the order, step by

step down, called four steps four pats. If the coils is powered on in the reverse order, D C B A D , the rotor will rotate in anti-clockwise direction.

Stepping motor has other control methods, such as connect A phase, then connect A B phase, the stator will be located in the middle of the A B, only a half-step. This way can improve the stability of stepping motor, and reduce noise, the sequence of coil powered on is: A AB B BC C CD D DA A , the rotor will rotate in accordance with the order, a half step by a half step, called four step eight pat.

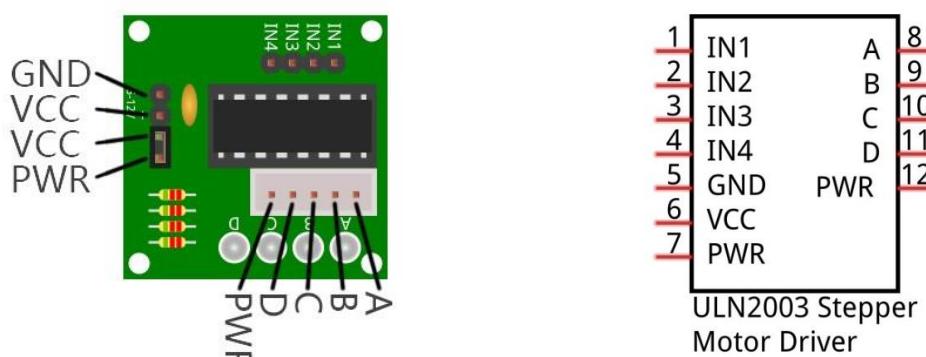
Equally, if the coil is powered on in reverse order, the stepping motor will rotate in reverse rotation.

The stator of stepping motor we use has 32 magnetic poles, so a circle needs 32 steps. The output shaft of the stepping motor is connected with a reduction gear set, and the reduction ratio is 1/64. So the final output shaft rotates a circle requiring a $32 \times 64 = 2048$ step

ULN2003 Stepping motor driver

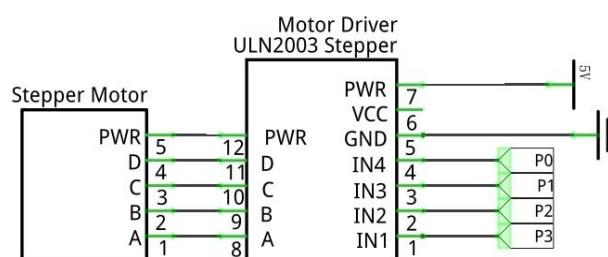
ULN2003 stepping motor driver is used to convert the weak signal into powerful control signal to drive the stepping motor. The input signal IN1-IN4 corresponds to the output signal A-D, and 4 LED is integrated in the board to indicate the state of signals. The PWR interface can be used as a power supply for stepping

motor. By default, PWR and VCC are connected by a short circuit.

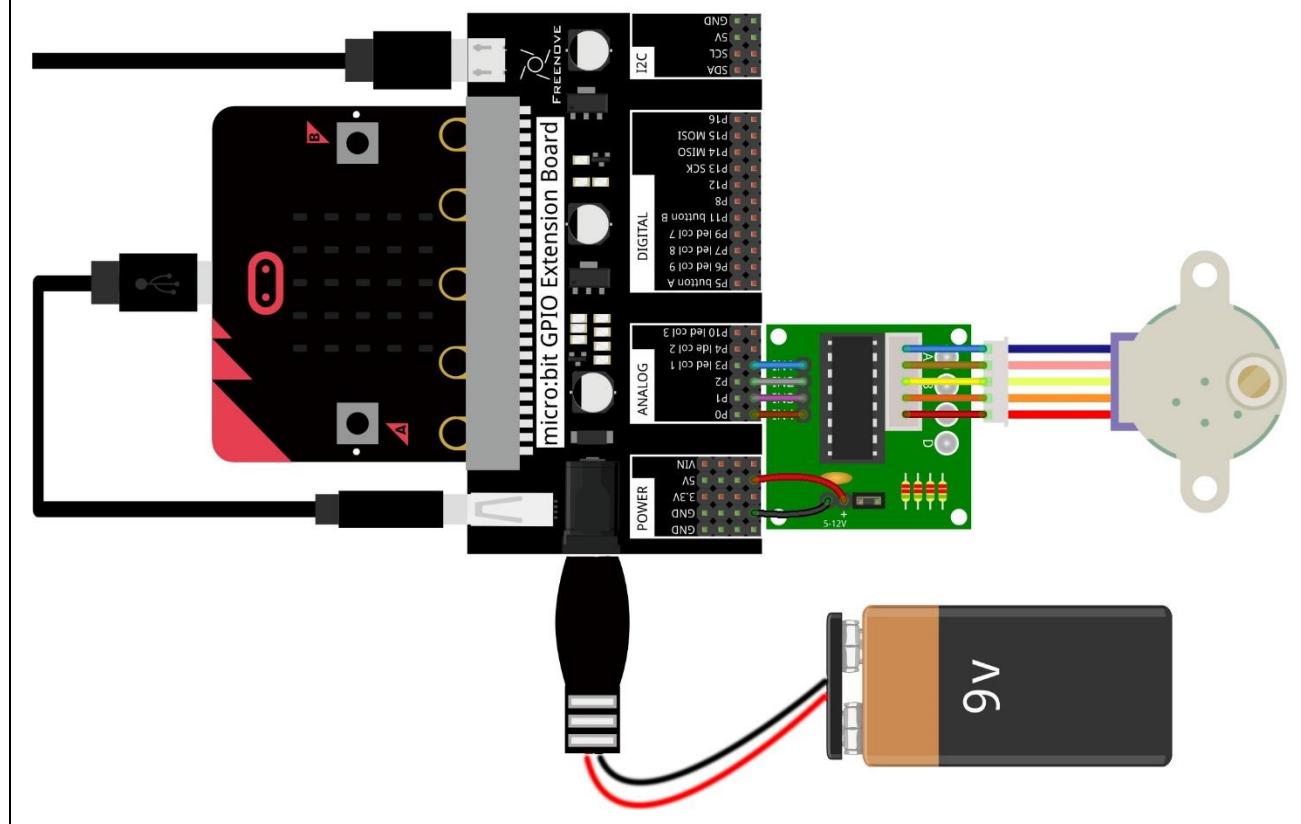


Circuit

Schematic diagram



Hardware connection



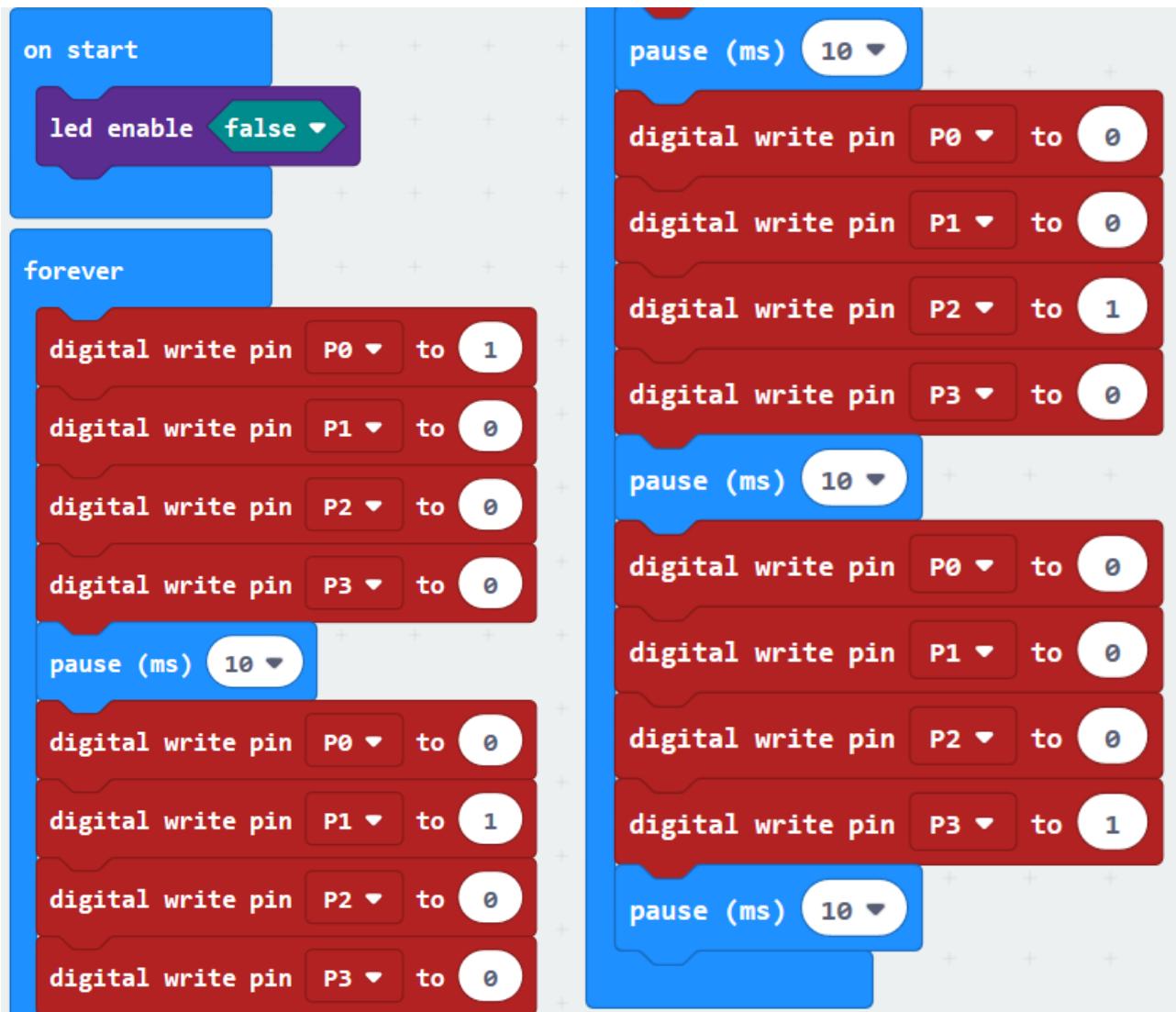
Block code

Open makecode first. Import the .hex file. The path is as below:

[\(How to import project\)](#)

File type	Path	File name
HEX file	./Projects/BlockCode/23.1_StepperMotor	StepperMotor.hex

After import successfully, the code is shown as below:



After checking the connection of the circuit and confirming the correct connection of the circuit, down the code into micro:bit, and then the stepper motor will rotate slowly.

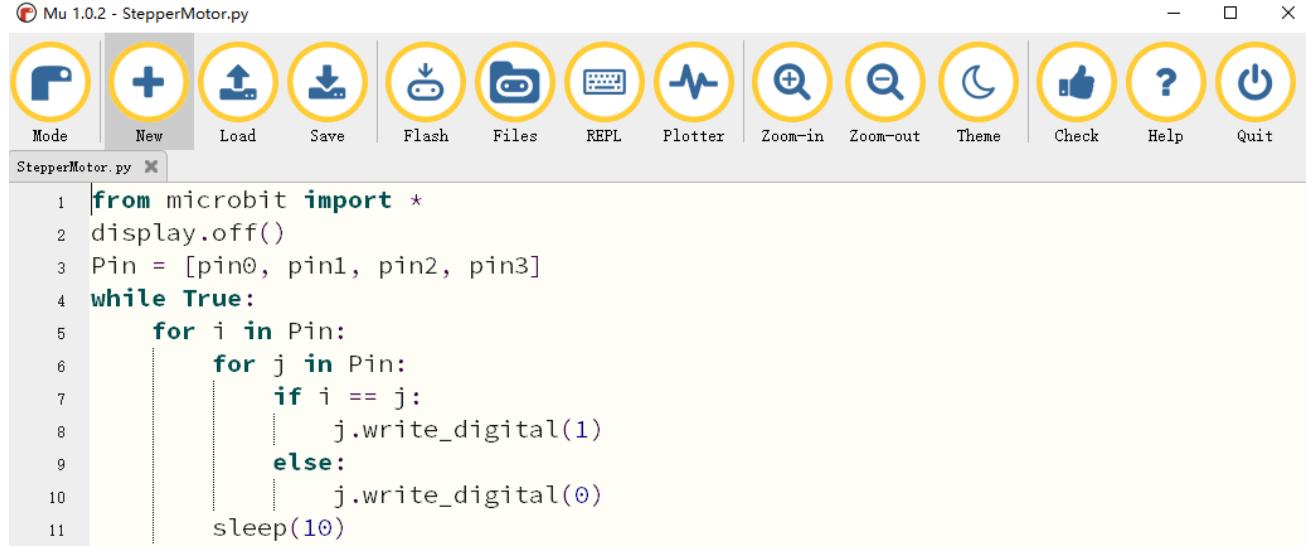
In the code, the pins of P 0, P 1, P 2 and P 3 is set to high level in turn. When one pin is at a high level, set the other three pins to low level. So the coil is energized as follows: A_B_C_D_A... A_B_C_D_A_C_D... to make the stepper motor rotate.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/23.1_StepperMotor	StepperMotor.py

After load successfully, the code is shown as below:



The screenshot shows the Mu 1.0.2 IDE interface. The title bar says "Mu 1.0.2 - StepperMotor.py". The toolbar icons include Mode, New, Load, Save, Flash, Files, REPL, Plotter, Zoom-in, Zoom-out, Theme, Check, Help, and Quit. The code editor window contains the following Python code:

```
1 from microbit import *
2 display.off()
3 Pin = [pin0, pin1, pin2, pin3]
4 while True:
5     for i in Pin:
6         for j in Pin:
7             if i == j:
8                 j.write_digital(1)
9             else:
10                 j.write_digital(0)
11             sleep(10)
```

After checking the connection of the circuit and confirming that the connection of the circuit is correct, the code is downloaded into micro:bit. The stepper motor rotates slowly.

The following is the program code:

```
1 from microbit import *
2 display.off()
3 Pin = [pin0, pin1, pin2, pin3]
4 while True:
5     for i in Pin:
6         for j in Pin:
7             if i == j:
8                 j.write_digital(1)
9             else:
10                j.write_digital(0)
11 sleep(10)
```

Close the LED dot matrix screen to allow the GPIO pins associated with the LED dot matrix screen to be reused for other purposes. Define Pin list to store P0, P1, P2, P3 pin variables.

```
display.off()
Pin = [pin0, pin1, pin2, pin3]
```

In the for cycle, the pin order with output high level is P0, P1, P3 P0.... When one pin outputs high level, make the other three pins output low levels. Make the coil electrified as follows: A_B_C_D_A... to make stepper motor rotate.

```
for i in Pin:
    for j in Pin:
        if i == j:
            j.write_digital(1)
        else:
            j.write_digital(0)
sleep(10)
```

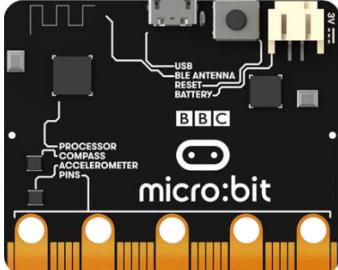
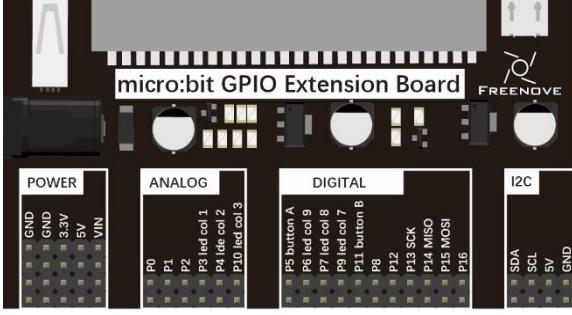
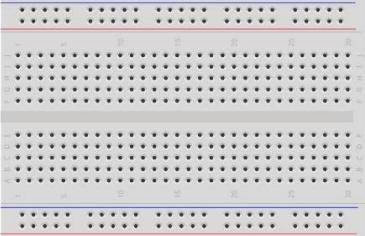
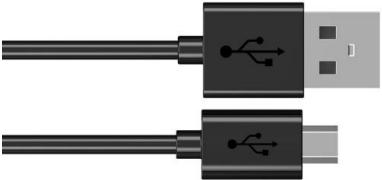
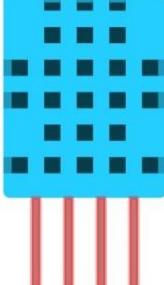
Chapter 24 Hygrothermograph

In this chapter, we will learn a commonly used sensor - Thermohygrometer DHT11.

Project 24.1 Hygrothermograph

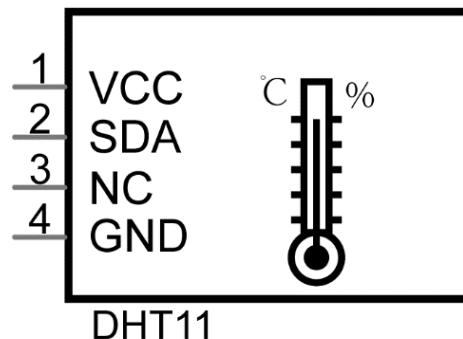
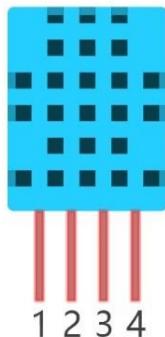
This project uses micro:bit to read and print the temperature and humidity data of DHT11.

Component list

Microbit x1	Expansion board x1	
		
Breakboard x1	USB cable x2	
		
DHT11 x1	Resistor 10kΩ x1	Jumper M/M x1 F/M x3
		

Component knowledge

Temperature & Humidity Sensor DHT11 is a compound temperature & humidity sensor, and the output digital signal has been calibrated inside.

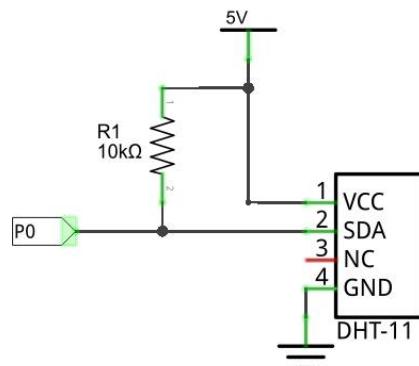


It has 1S's initialization time after powered up. The operating voltage is within the range of 3.3V-5.5V. SDA pin is a data pin, which is used to communicate with other device.

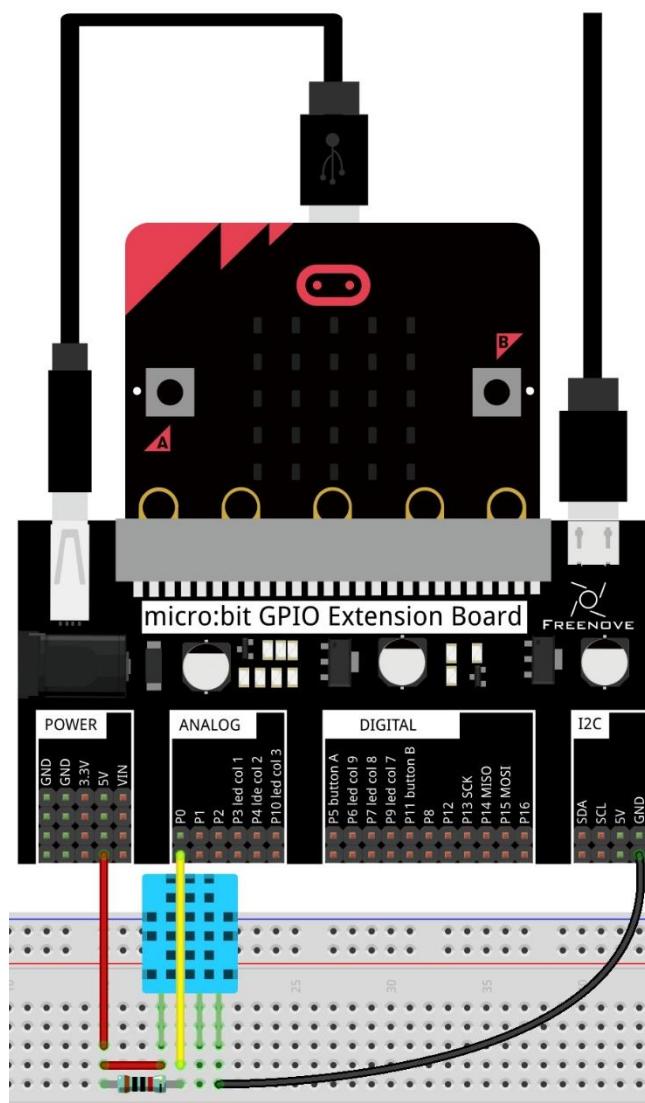
NC pin (Not Connected Pin) are pins found on various integrated circuit packages. Those pins have no functional purpose to the outside circuit (but may have unexposed functionality). Those pins should not be connected to any of the circuit connections.

Circuit

Schematic diagram



Hardware connection



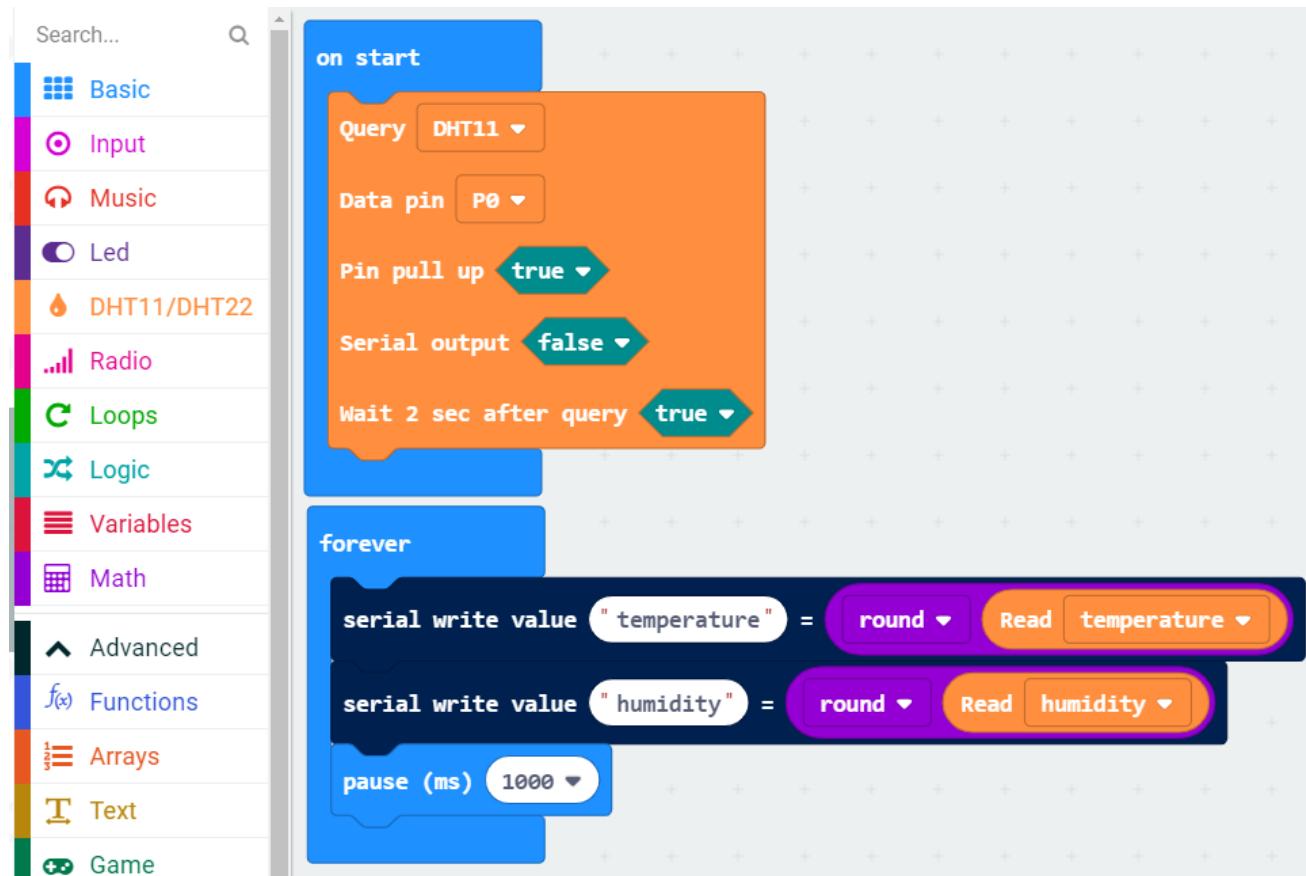
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file	../Projects/BlockCode/24.1_DHT11	DHT11.hex

After import successfully, the code is shown as below:

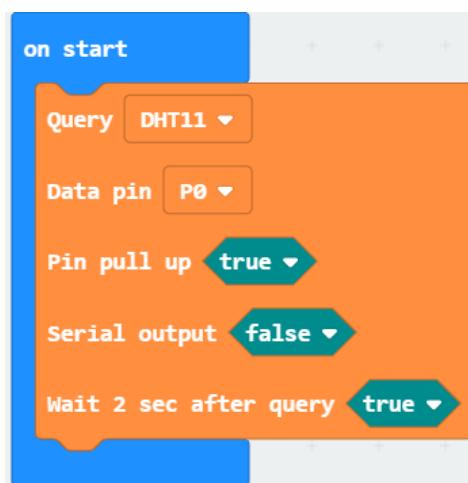


After checking the connection of the circuit and confirming the correct connection of the circuit, download the code into micro:bit and open the serial port controller, you can see the temperature and humidity of the current environment, as shown in the following figure:

```

temperature:27
humidity:56
temperature:27
humidity:56
temperature:27
  
```

Set sensor type to DHT11, data pin to P 0, initialize DHT11 module, wait 2 seconds



Every 1 s, the temperature and humidity data read will be printed out by serial port controller.

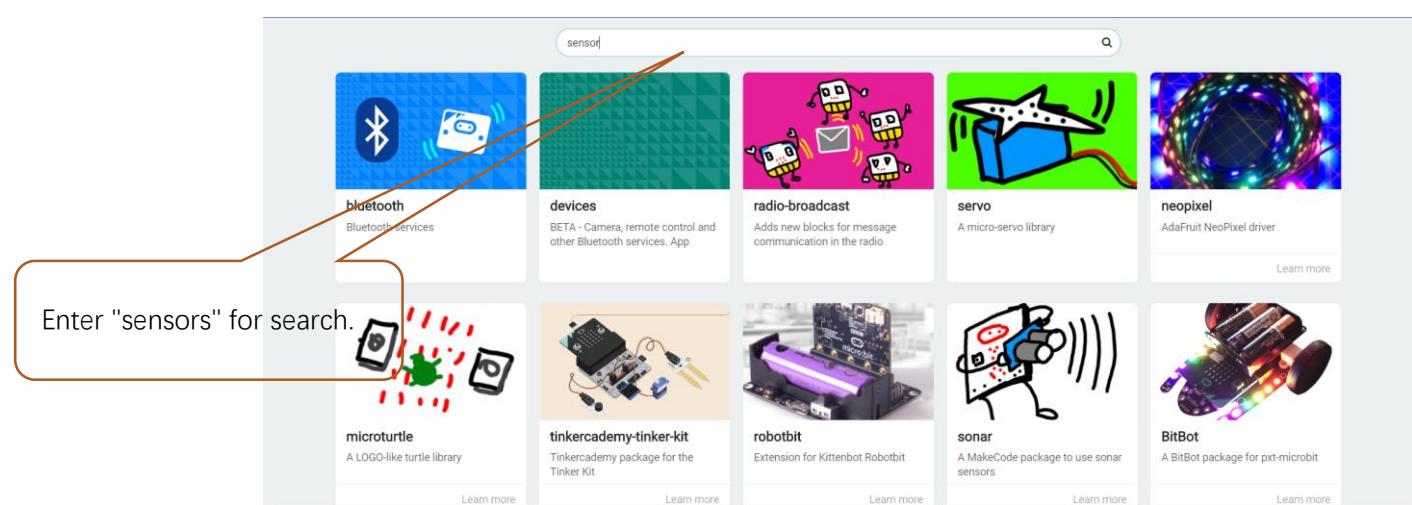
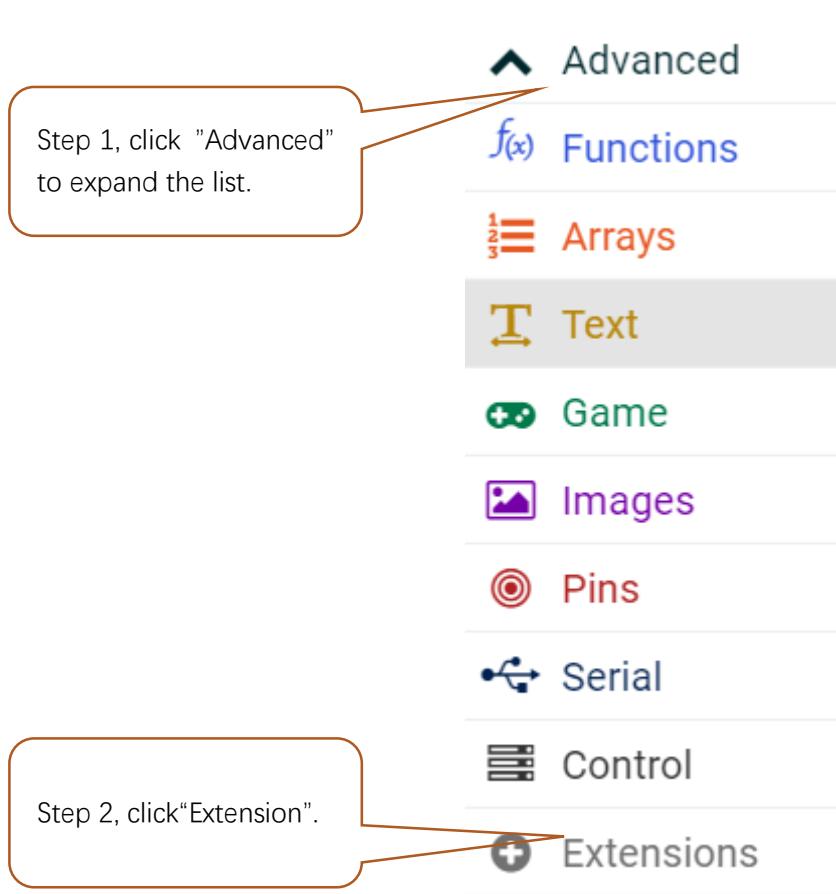


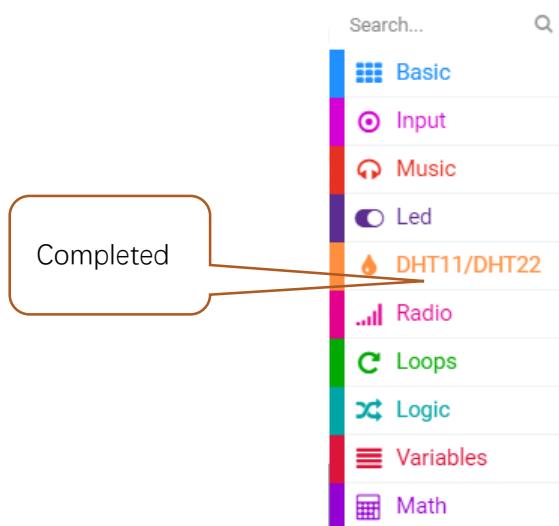
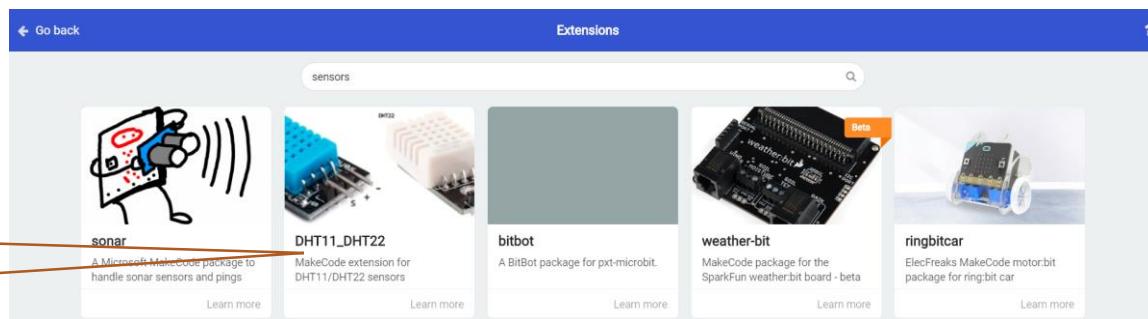
Reference

Block	Function
<pre> Query DHT11 Data pin P0 Pin pull up true Serial output false Wait 2 sec after query true </pre>	Set sensor type DHT11 or DHT22, select data pin and initialize DHT module.
<pre> Read humidity </pre>	Read humidity level (%) or temperature (Celsius).

Extensions

If you want to import the DHT Sensor Extension Block into your new project, follow these steps to add it.



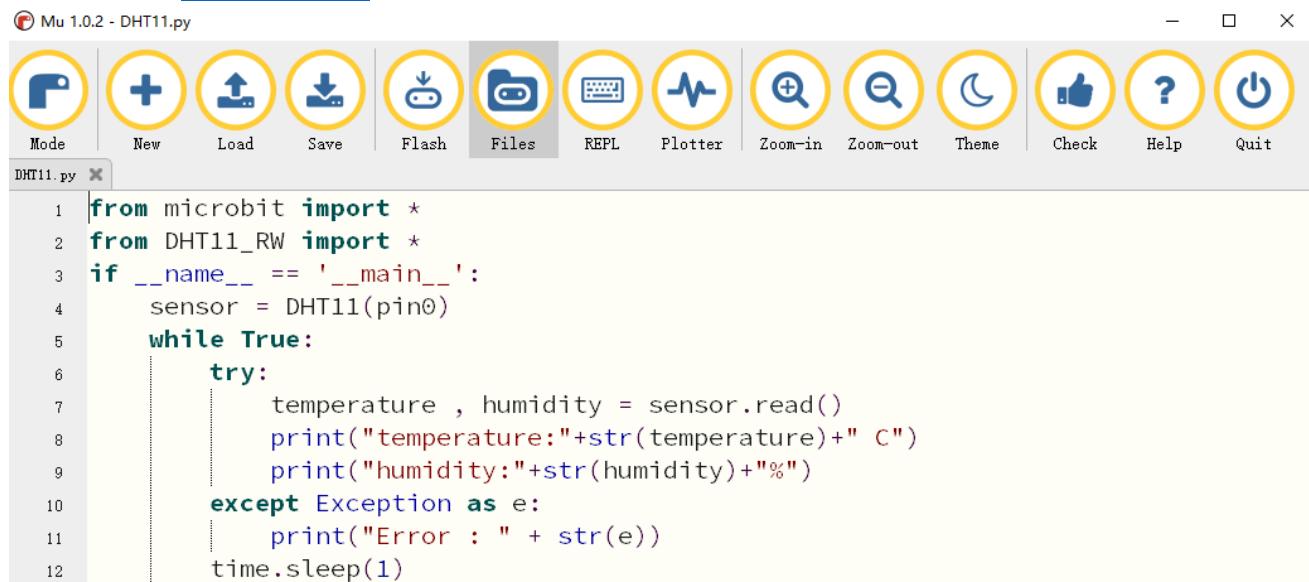


Python code

Open the .py file with Mu. Code path is as below::

File type	Path	File name
Python file/Projects/PythonCode/24.1_DHT11	DHT11.py

After loading the code, as shown below. Before downloading the code, import the "DHT11_RW.py" file into micro:bit first.([How to import?](#))



```

1 from microbit import *
2 from DHT11_RW import *
3 if __name__ == '__main__':
4     sensor = DHT11(pin0)
5     while True:
6         try:
7             temperature , humidity = sensor.read()
8             print("temperature:"+str(temperature)+" C")
9             print("humidity:"+str(humidity)+"%")
10        except Exception as e:
11            print("Error : " + str(e))
12            time.sleep(1)

```

After importing the DHT11_RW.py file, check the connection of the circuit, confirm the correct connection of the circuit. Then download the code into micro:bit, click "REPL", press the micro:bit reset button. You can see the temperature and humidity of the current environment, as shown below:

```

BBC micro:bit REPL
humidity:46.0%
temperature:28.0 C
humidity:46.0%
temperature:28.0 C
humidity:46.0%
temperature:28.0 C
humidity:46.0%
temperature:28.0 C
humidity:46.0%

```

The following is the program code:

```
1 from microbit import *
2 from DHT11_RW import *
3 if __name__ == '__main__':
4     sensor = DHT11(pin0)
5     while True:
6         try:
7             temperature , humidity = sensor.read()
8             print("temperature:"+str(temperature)+" C")
9             print("humidity:"+str(humidity)+"%")
10        except Exception as e:
11            print("Error : " + str(e))
12            time.sleep(1)
```

Export everything of DHT11_RW module, create object of DHT class, and set data pin to P0

```
from DHT11_RW import *
sensor = DHT11(pin0)
```

Every 1 second, call the read() function in DHT11 class, get the temperature and humidity data, and print them out separately.

```
while True:
    try:
        temperature , humidity = sensor.read()
        print("temperature:"+str(temperature)+" C")
        print("humidity:"+str(humidity)+"%")
    except Exception as e:
        print("Error : " + str(e))
    time.sleep(1)
```

Reference

sensor.read()

The read() function is defined in DHT11 to obtain temperature and humidity data.

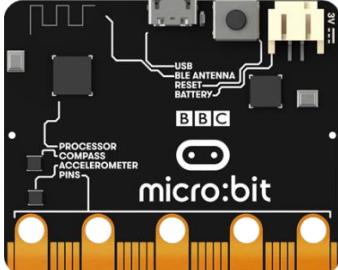
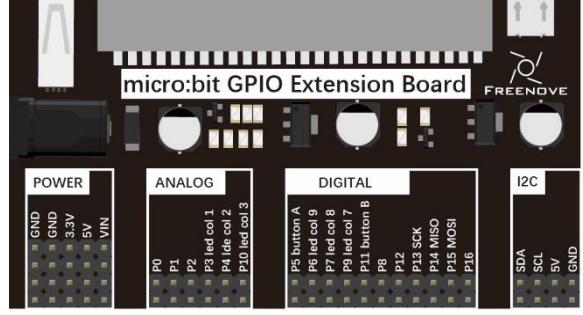
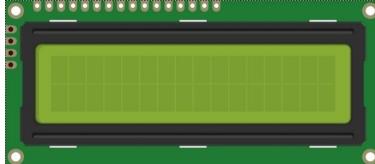
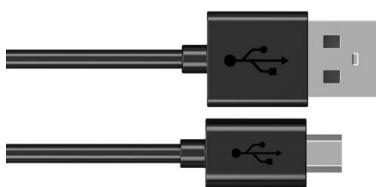
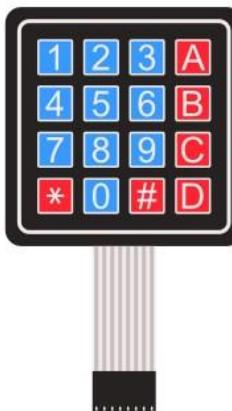
Chapter 25 Matrix Keypad

In this chapter, we will learn a new component: 4X4 matrix keyboard.

Project 25.1 Keypad

This project realizes LCD screen display number and character according to matrix keyboard.

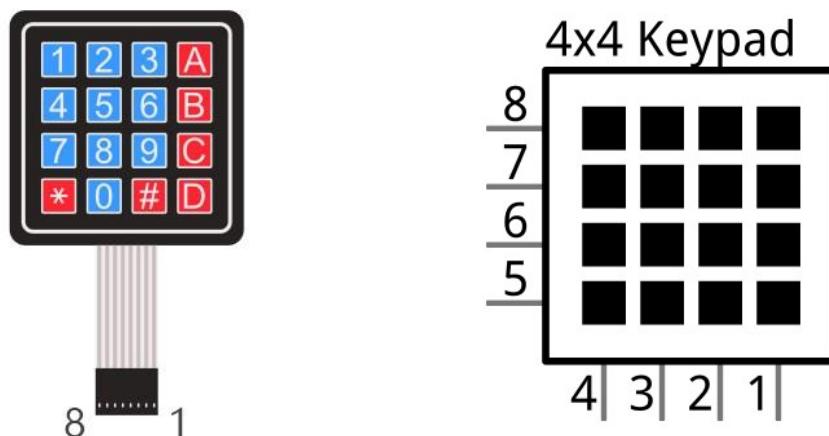
Component list

Microbit x1	Expansion board x1
	
I2C LCD1602 Module x1	USB cable x2
	
F/M x8 F/F x4	4x4 Matrix Keypad x1
	

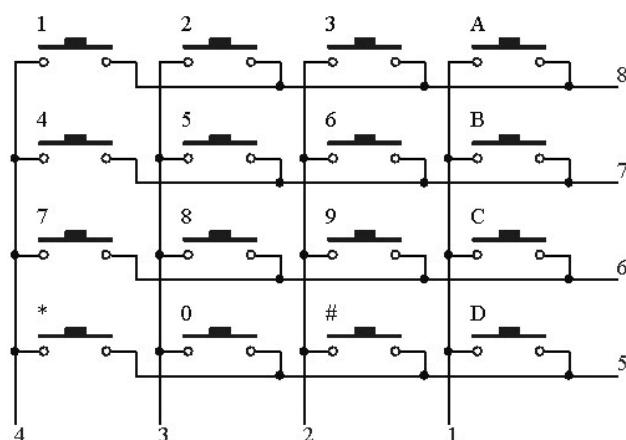
Component knowledge

4x4 Matrix Keypad

Keypad is a device that integrates a number of keys. As is shown below, a 4x4 Keypad integrates 16 keys:



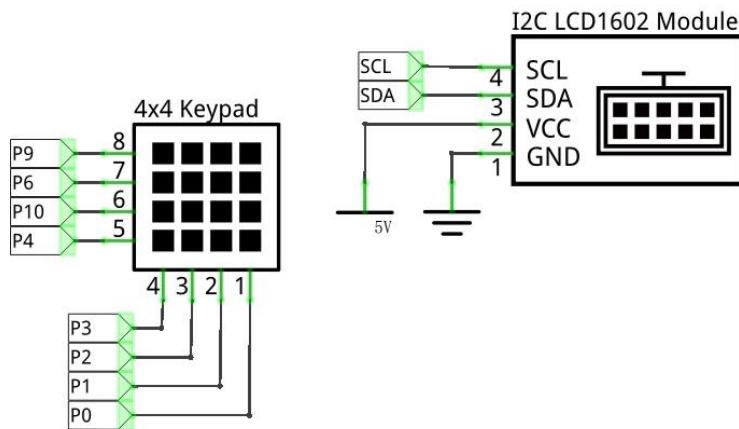
Like the integration of LED matrix, in 4x4 Keypad each row of keys is connected in with one pin and it is the same as each column. Such connection can reduce the occupation of processor port. Internal circuit is shown below.



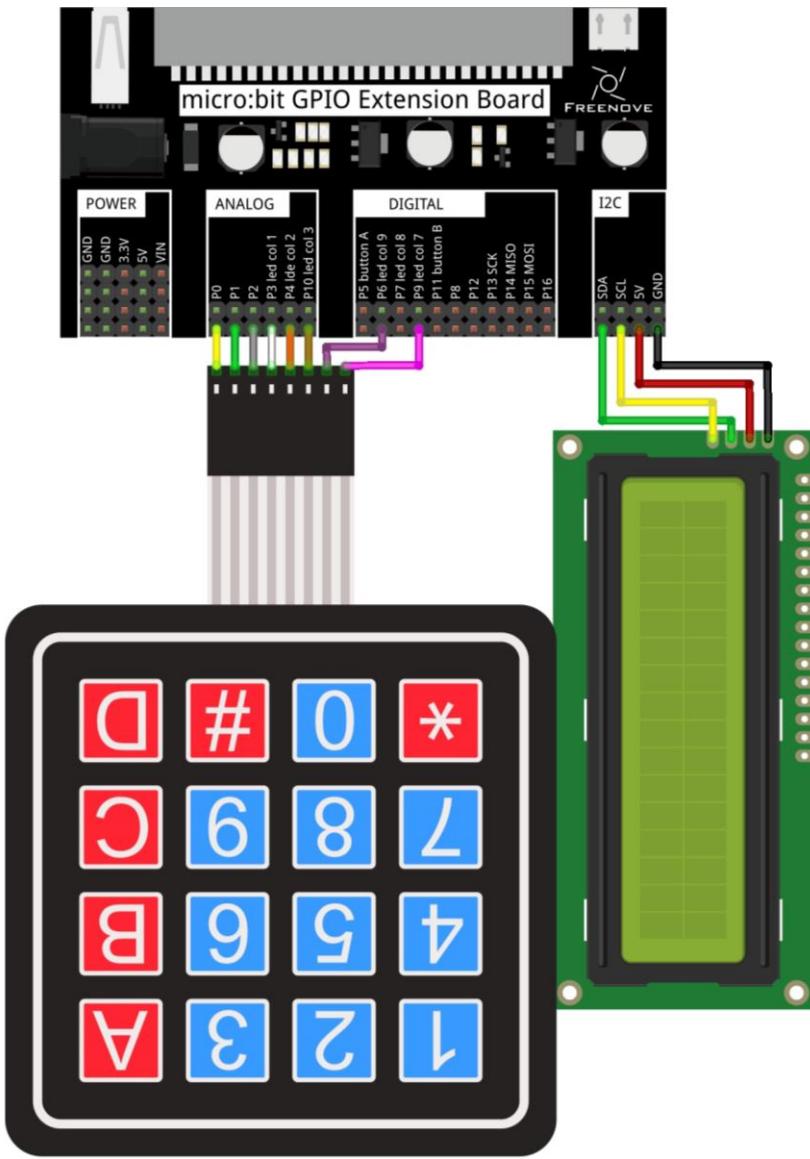
The usage method is similar to the Matrix LED, namely, uses a row scan or column scanning method to detect the state of key on each column or row. Take column scanning method as an example, send low level to the first 1 column (Pin1), detect level state of row 5, 6, 7, 8 to judge whether the key A, B, C, D are pressed. And then send low level to column 2, 3, 4 in turn to detect whether other keys are pressed. Then, you can get the state of all keys.

Circuit

Schematic diagram



Hardware connection



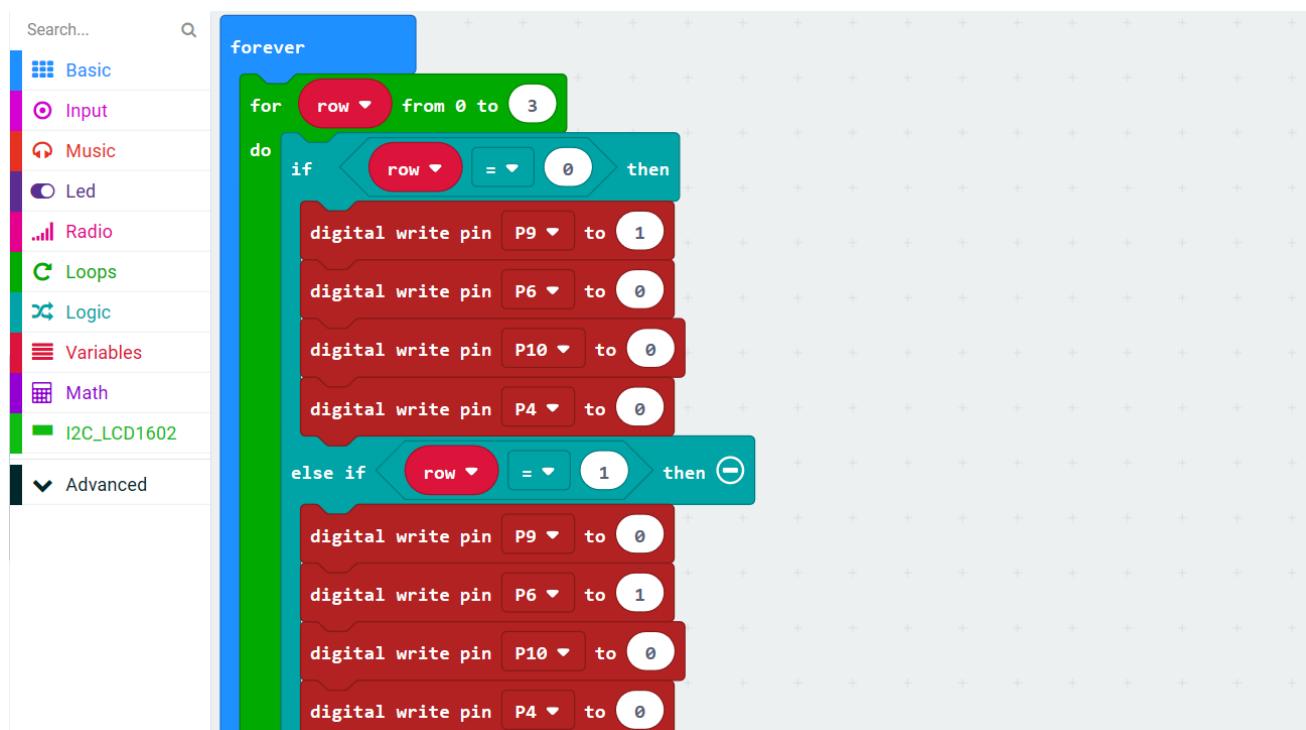
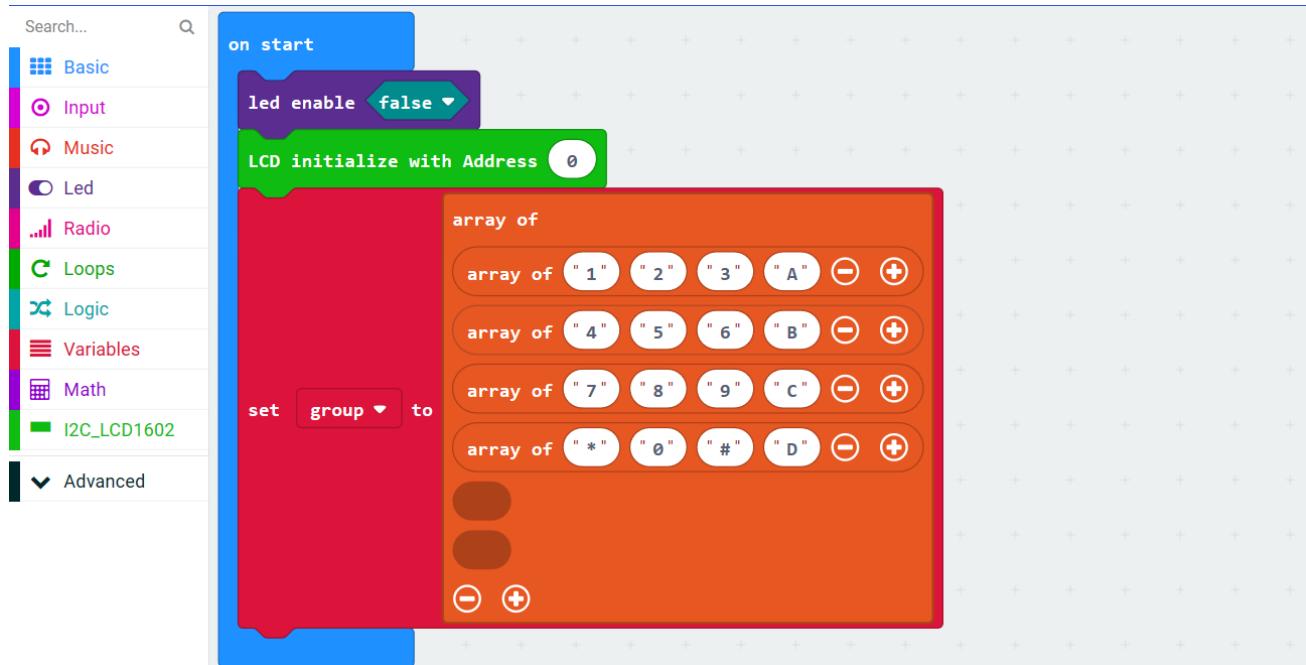
Block code

Open makecode first. Import the .hex file. The path is as below::

([How to import project](#))

File type	Path	File name
HEX file	./Projects/BlockCode/25.1_Keypad	Keypad.hex

After import successfully, the code is shown as below:



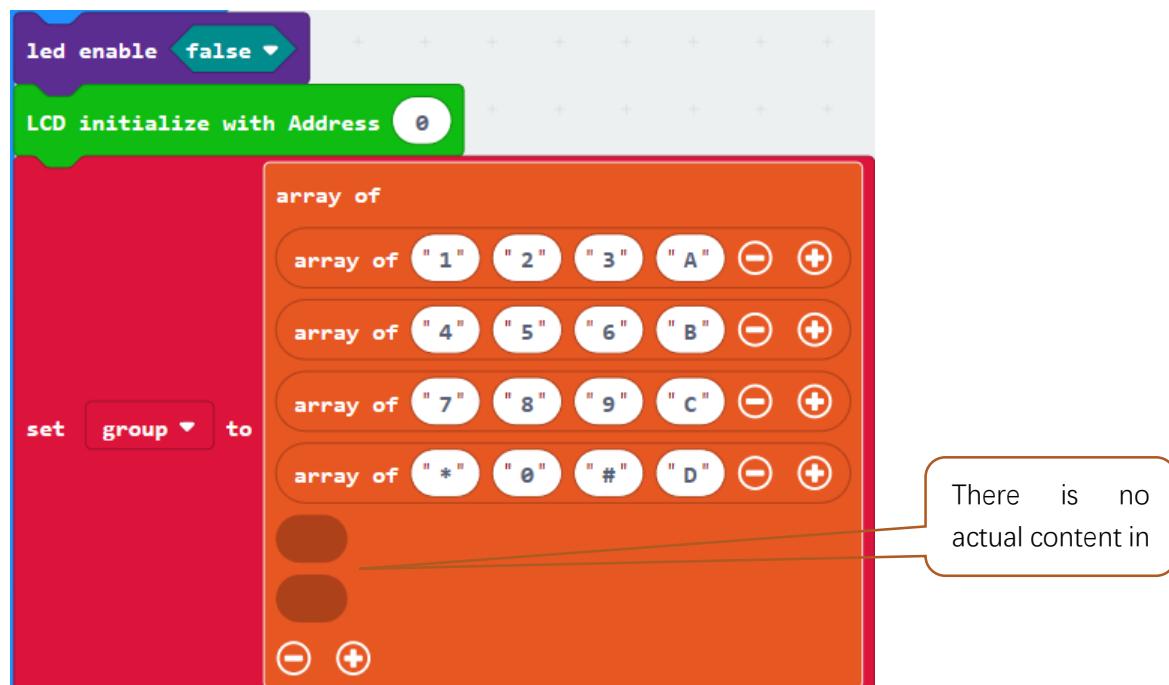
The image shows two Scratch scripts side-by-side, each consisting of a vertical stack of blocks. The top script is for row 2 and row 3, and the bottom script is for columns 1 through 4. Both scripts use digital pins P9, P6, P10, and P4 for row and column control, and pin P3 or P2 for reading key presses. They also utilize the I2C_LCD1602 block to display the character at the intersection of the row and column.

```

    [Top Script]
    else if [row v] = [2] then
        digital write pin [P9 v] to [0]
        digital write pin [P6 v] to [0]
        digital write pin [P10 v] to [1]
        digital write pin [P4 v] to [0]
    else if [row v] = [3] then
        digital write pin [P9 v] to [0]
        digital write pin [P6 v] to [0]
        digital write pin [P10 v] to [0]
        digital write pin [P4 v] to [1]
    end
    if [digital read pin [P3 v] = [1] then
        set [column v] to [0]
        show string [group v] [get value at [row v] [get value at [column v] [at x [y [0]]]]]
    end
    [Bottom Script]
    else if [digital read pin [P2 v] = [1] then
        set [column v] to [1]
        show string [group v] [get value at [row v] [get value at [column v] [at x [y [0]]]]]
    else if [digital read pin [P1 v] = [1] then
        set [column v] to [2]
        show string [group v] [get value at [row v] [get value at [column v] [at x [y [0]]]]]
    else if [digital read pin [P0 v] = [1] then
        set [column v] to [3]
        show string [group v] [get value at [row v] [get value at [column v] [at x [y [0]]]]]
    end
    end
  
```

After checking the connection of the circuit and confirming the correct connection of the circuit, the code is downloaded into micro:bit. When the key of the matrix keyboard is pressed, the LCD will display the corresponding numbers or characters.

Close the LED dot matrix screen, initialize the LCD, and store the values of the matrix keyboard 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, A, B, C, D, #, *, in the array group variable.



Row scanning. Pins P9, P6, P10, P4 corresponds to the first, second, third and fourth rows. Make the pins output high level in turn, the other pins output low level.



Column scanning. Pins P3, P2, P1, P0 corresponds to the first, second, third and fourth columns. Reading high level means the key of current line and column is pressed. And the corresponding number or character in the group array will be displayed on the LCD.



Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file/Projects/PythonCode/25.1_Keypad	Keypad.py

After the code is loaded, as shown below, import the "I2C_LCD1602_Class.py" file to micro:bit before downloading the code. ([How to import?](#))



```

Mu 1.0.2 - Keypad.py
Keypad.py X

1 from microbit import *
2 from I2C_LCD1602_Class import *
3 display.off()
4 group=[[1, 2, 3, "A"], [4, 5, 6, "B"], [7, 8, 9, "C"], ["*", "0", "#", "D"]]
5 Pin_row = [pin9, pin6, pin10, pin4]
6 Pin_column = [pin3, pin2, pin1, pin0]
7 lcd = I2C_LCD1602(0x27)
8 lcd.clear()
9 for i in range(4):
10     Pin_row[i].write_digital(0)
11 for i in range(4):
12     Pin_column[i].write_digital(0)
13 while True:
14     for i in range(4):
15         Pin_row[i].write_digital(1)
16         for j in range(4):
17             if Pin_column[j].read_digital()==1:
18                 lcd.puts(group[i][j], 0, 0)
19             Pin_row[i].write_digital(0)

```

After importing the I2C_LCD1602_Class.py file, check the connection of the circuit, confirm the correct connection of the circuit. Then download the code into micro:bit, and press the key of the matrix keyboard. And then LCD will display the corresponding number or character.

The following is the program code:

```

1 from microbit import *
2 from I2C_LCD1602_Class import *
3 display.off()
4 group=[[1, 2, 3, "A"], [4, 5, 6, "B"], [7, 8, 9, "C"], ["*", "0", "#", "D"]]
5 Pin_row = [pin9, pin6, pin10, pin4]
6 Pin_column = [pin3, pin2, pin1, pin0]
7 lcd = I2C_LCD1602(0x27)
8 lcd.clear()
9 for i in range(4):
10     Pin_row[i].write_digital(0)
11 for i in range(4):
12     Pin_column[i].write_digital(0)
13 while True:
14     for i in range(4):

```

```
15     Pin_row[i].write_digital(1)
16     for j in range(4):
17         if Pin_column[j].read_digital()==1:
18             lcd.puts(group[i][j], 0, 0)
19     Pin_row[i].write_digital(0)
```

Close the LED dot matrix screen, store the values of matrix keyboard 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, A, B, C, D., *, in the array group. Store the pin variables of the control keyboard row to Pin_row, and store the pin variables of the control keyboard column to Pin_column. Create the I2C_LCD1602 object lcd, enter the I2C address and clear the screen. Set the pins connecting the matrix keyboard to low level.

```
display.off()
group=[["1", "2", "3", "A"], ["4", "5", "6", "B"], ["7", "8", "9", "C"], ["*", "0", "#", "D"]]
Pin_row = [pin9, pin6, pin10, pin4]
Pin_column = [pin3, pin2, pin1, pin0]
lcd = I2C_LCD1602(0x27)
lcd.clear()
for i in range(4):
    Pin_row[i].write_digital(0)
for i in range(4):
    Pin_column[i].write_digital(0)
```

Scan rows and columns. If the key in certain row and column is pressed, the corresponding number or character in the group array will be displayed on the LCD.

```
while True:
    for i in range(4):
        Pin_row[i].write_digital(1)
        for j in range(4):
            if Pin_column[j].read_digital()==1:
                lcd.puts(group[i][j], 0, 0)
        Pin_row[i].write_digital(0)
```

Project 25.2 Countdown Timer

This project makes a countdown timer.

Component list

It is same with last project.

Circuit

It is same with last project.

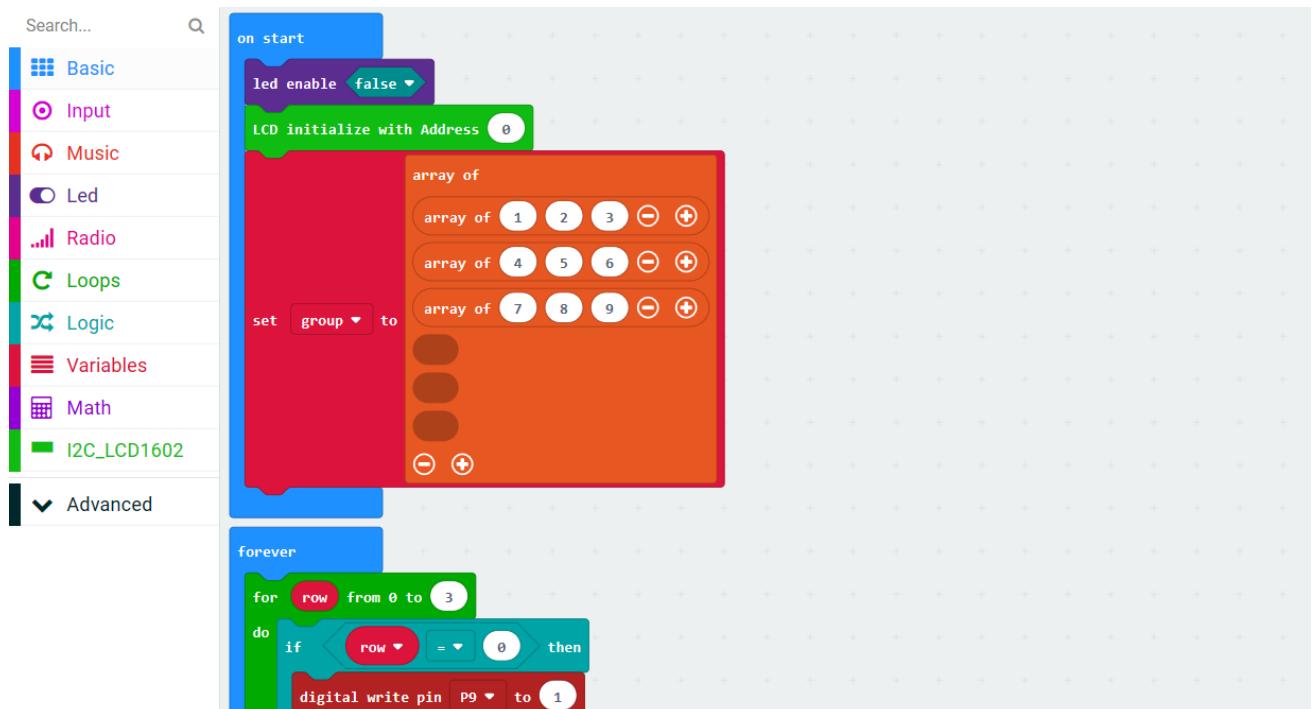
Block code

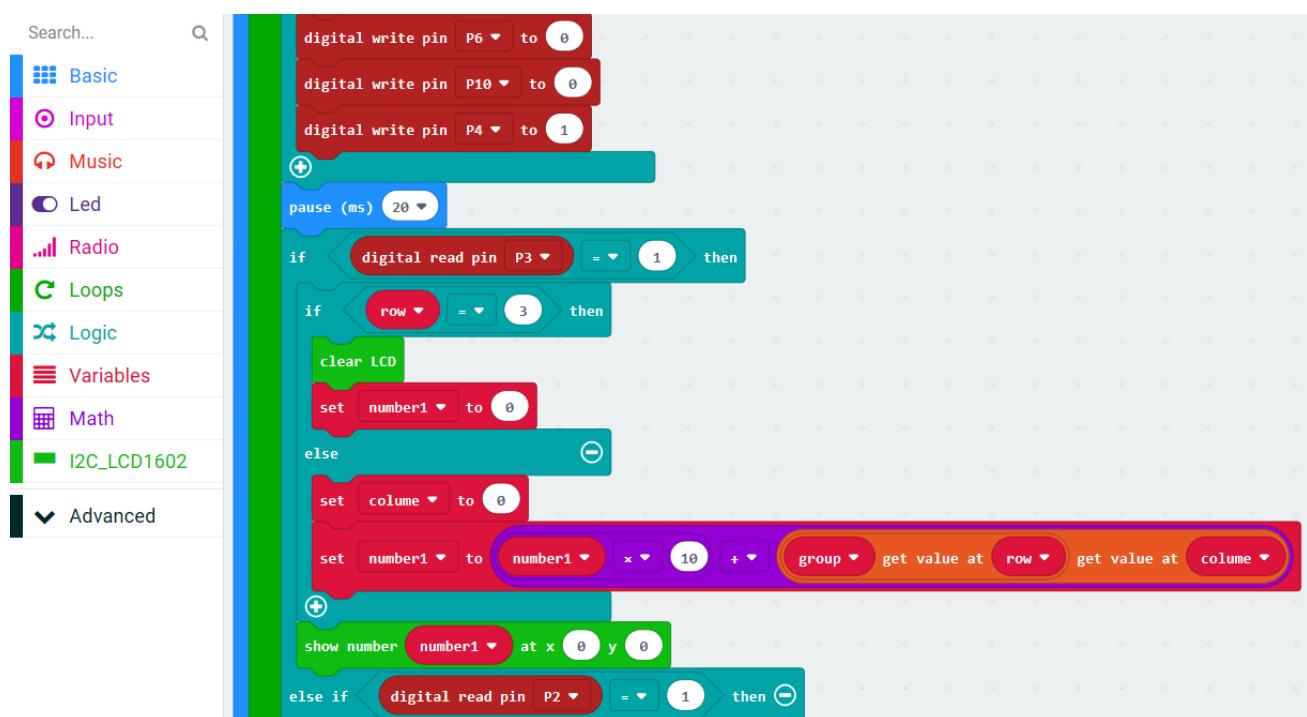
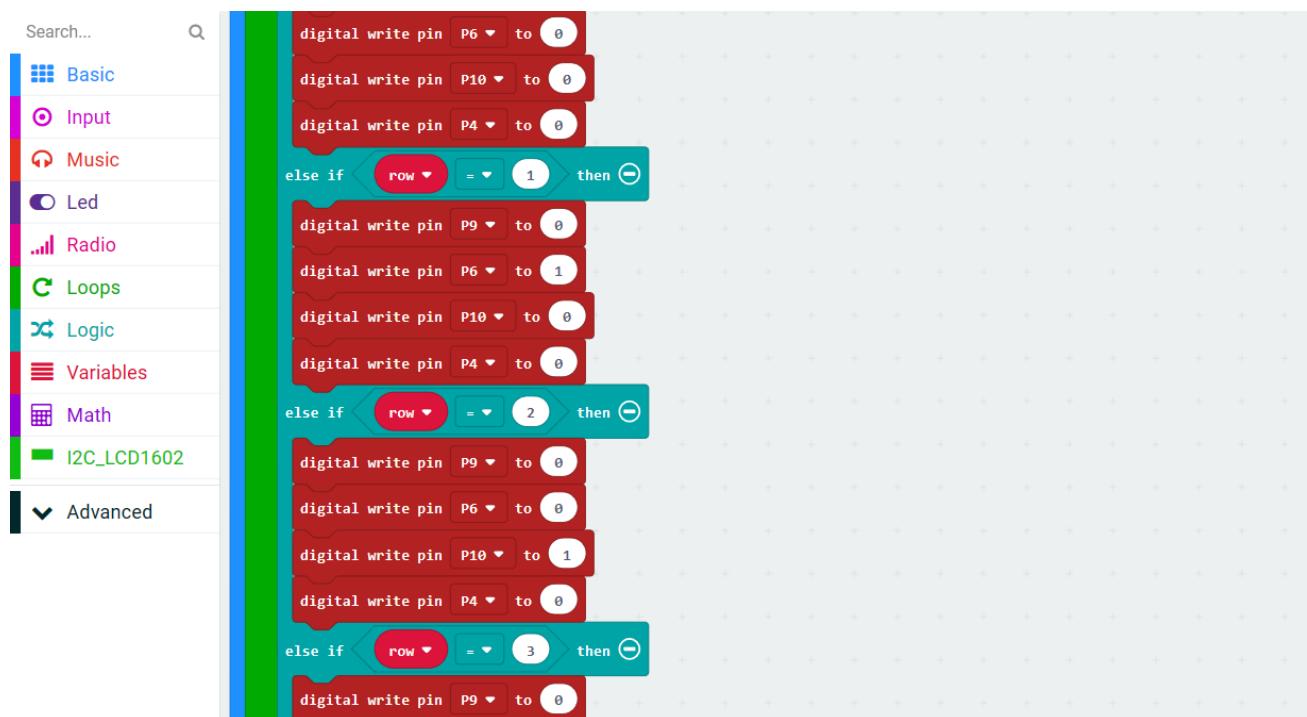
Open makecode first. Import the .hex file. The path is as below:

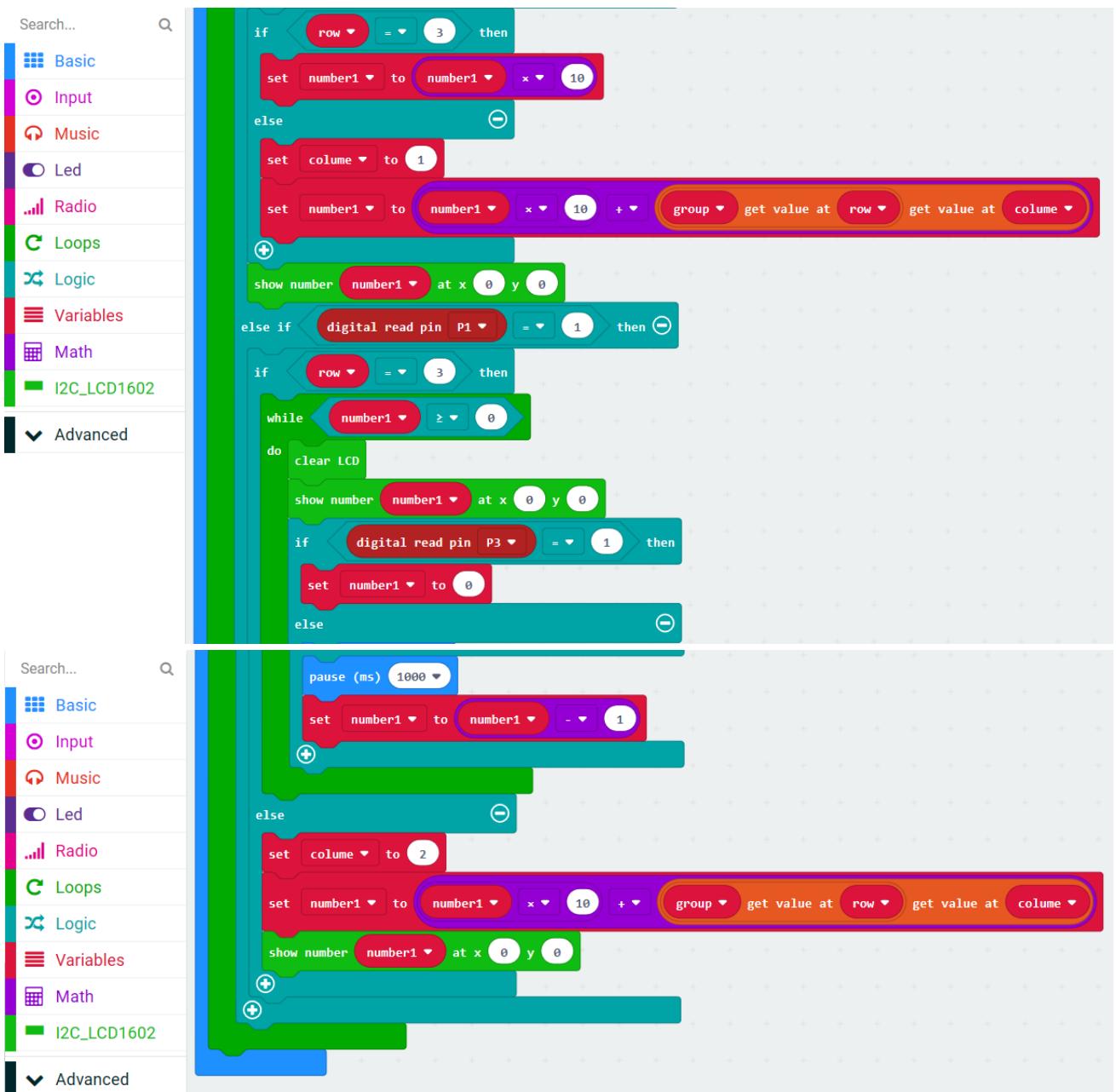
([How to import project](#))

File type	Path	File name
HEX file	../Projects/BlockCode/25.2_CountdownTimer	CountdownTimer.hex

After import successfully, the code is shown as below:

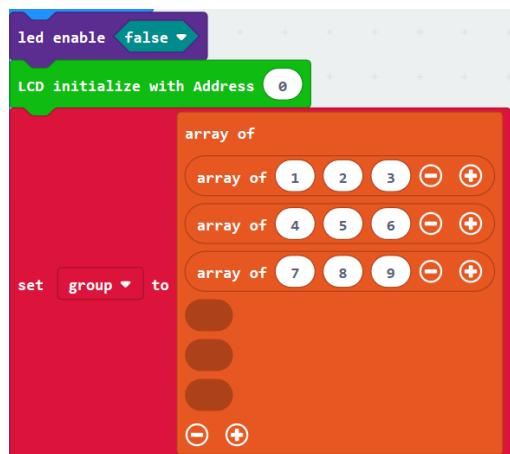






After checking the connection of the circuit and confirming the correct connection of the circuit, and download the code into micro:bit, type in the value, press the '#'key, start countdown, the key '*' is reset.

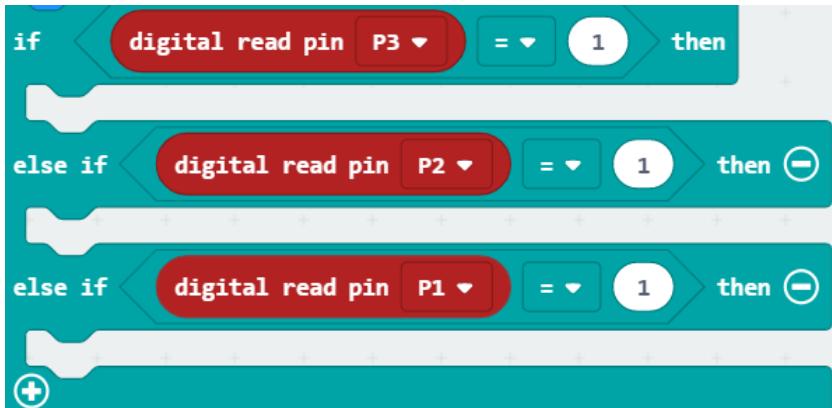
Close the LED dot matrix screen, initialize the LCD and store the values of matrix keyboard 1, 2, 3, 4, 5, 6, 7, 8, 9 in the array group.



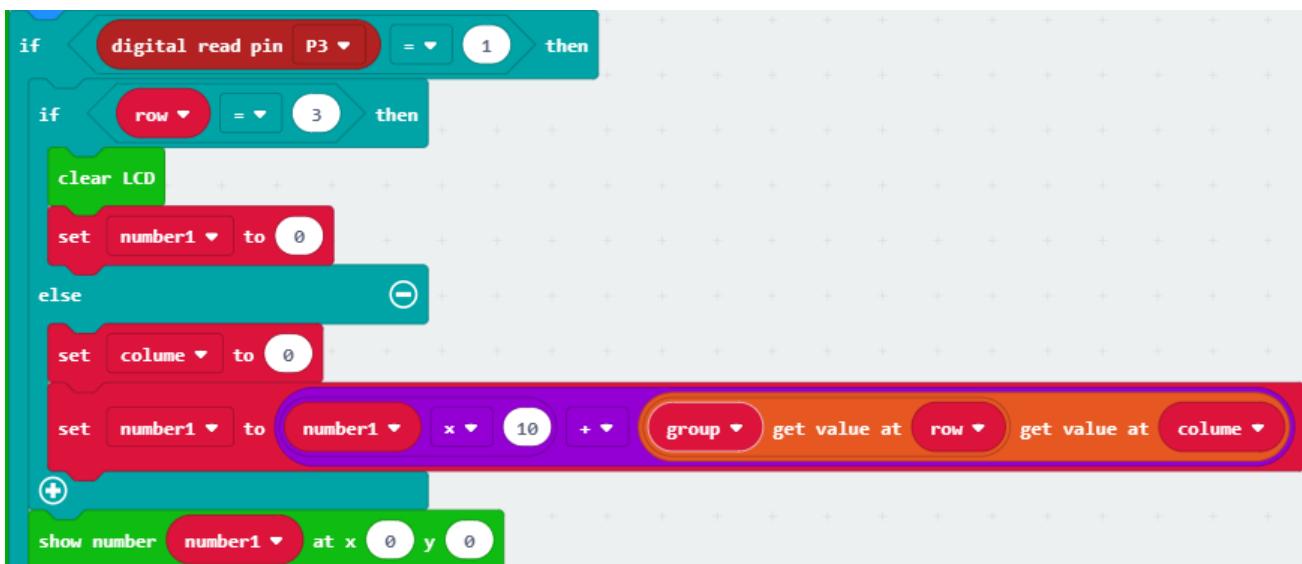
Row scanning. Pins P9, P6, P10, P4 corresponds to the first, second, third and fourth rows. Make the pins output high level in turn, the other pins output low level.



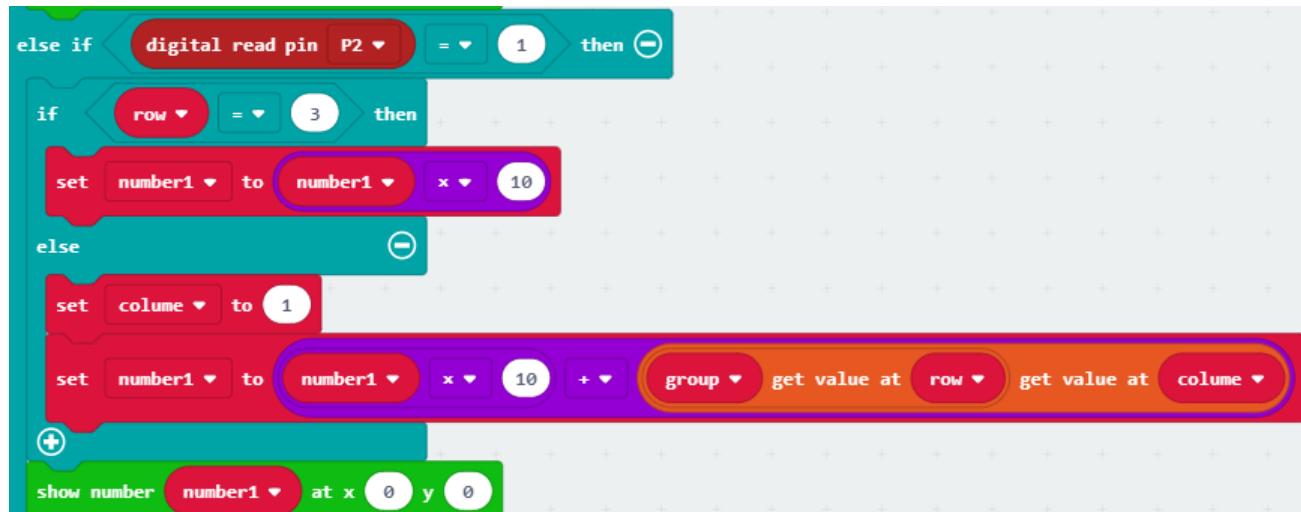
Column scanning. Pins P3, P2, P1 corresponds to the first, second and third columns. When reading high level, the corresponding reaction will be executed.



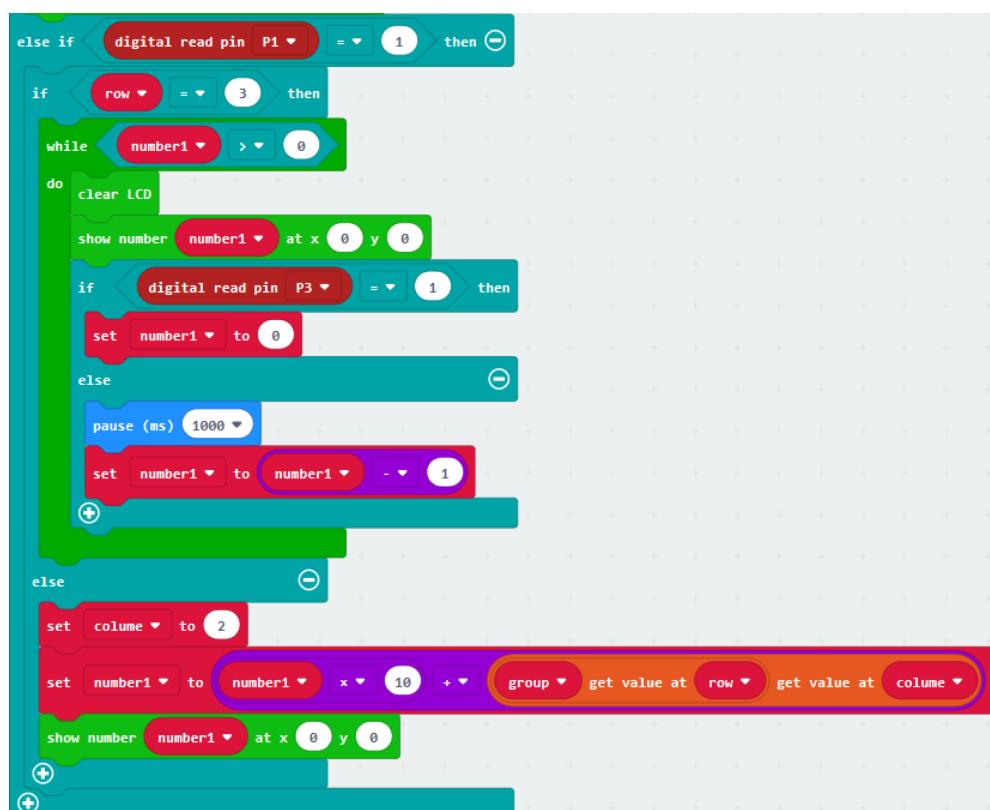
Determine whether the first column on the left is pressed, and then determine which row of the column is pressed. If the "*" key of the fourth row and first column is pressed, clear the LCD screen content, and the Number1 variable is assigned 0; if the other keys 1, 4 or 7 is pressed, the Number1 variable is multiplied by 10 and the corresponding key value is added, and then reassigned to Number1 to achieve carry effect.



Determine whether the second column on the left is pressed, and then determine which row of the column is pressed. If the "0" key of the fourth row and second column is pressed, the value of Number1 is multiplied by 10, and then reassigned to Number1 variable. If the other keys 2, 5 or 8 is pressed, the value of Number1 is multiplied by 10 and the corresponding key value is added, and then reassigned to Number1 to achieve carry effect.



Determine whether the third column on the left is pressed, and then determine which row of the column is pressed. If the "#" key of the fourth row and the third column is pressed, the countdown is performed by the while loop. If the other keys 3, 6, 9 is pressed, the value of Number1 will be multiplied by 10 and the corresponding key value is reassigned to the Number1 variable to achieve carry effect. The Number1 variable will decrease by 1 every 1s. During the cycle, if the "*" key is pressed or the value of Number1 variable is less than or equal to 0, the loop will jumper out.



Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file/Projects/PythonCode/25.2_CountdownTimer	CountdownTimer.py

After the code is loaded, as shown below, import the "I2C_LCD1602_Class.py" file to micro:bit before downloading the code.([How to import?](#))



```

1  from microbit import *
2  from I2C_LCD1602_Class import *
3  display.off()
4  group=[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
5  Pin_row = [pin9, pin6, pin10, pin4]
6  Pin_column = [pin3, pin2, pin1, pin0]
7  lcd = I2C_LCD1602(0x27)
8  lcd.clear()
9  number=0
10 for i in range(4):
11     Pin_row[i].write_digital(0)
12 for i in range(4):
13     Pin_column[i].write_digital(0)
14 while True:
15     for i in range(4):
16         Pin_row[i].write_digital(1)
17         for j in range(3):
18             if Pin_column[j].read_digital()==1:
19                 sleep(100)
20                 if Pin_column[j].read_digital()==1:
21                     if i==3 and j==0:      #Press the "*" button
22                         number=0
23                         lcd.clear()
24                     elif i==3 and j==1:    #Press the "0" button
25                         number=number*10
26                     elif i==3 and j==2:    #Press the "#" button
27                         while True:
28                             lcd.clear()
29                             lcd.puts(str(number), 0, 0)
30                             number-=1
31                             sleep(1000)
32                             if pin3.read_digital()==1 or number==0:
33                                 number=0
34                                 lcd.clear()
35                                 break
36             else:                  #Press the number button
37                 number=number*10+group[i][j]
38                 lcd.puts(str(number), 0, 0)
39     Pin_row[i].write_digital(0)

```

After importing the I2C_LCD1602_Class.py file, check the connection of the circuit, confirm the correct connection of the circuit, and download the code into micro:bit, type in the value, press the '#' key, start countdown, the key'*' is reset.

The following is the program code:

1	from microbit import *
---	------------------------

```
2 from I2C_LCD1602_Class import *
3 display.off()
4 group=[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
5 Pin_row = [pin9, pin6, pin10, pin4]
6 Pin_column = [pin3, pin2, pin1, pin0]
7 lcd = I2C_LCD1602(0x27)
8 lcd.clear()
9 number=0
10 for i in range(4):
11     Pin_row[i].write_digital(0)
12 for i in range(4):
13     Pin_column[i].write_digital(0)
14 while True:
15     for i in range(4):
16         Pin_row[i].write_digital(1)
17         for j in range(3):
18             if Pin_column[j].read_digital()==1:
19                 sleep(100)
20             if Pin_column[j].read_digital()==1:
21                 if i==3 and j==0:      #Press the "*" button
22                     number=0
23                     lcd.clear()
24                 elif i==3 and j==1:  #Press the "0" button
25                     number=number*10
26                 elif i==3 and j==2:  #Press the "#" button
27                     while True:
28                         lcd.clear()
29                         lcd.puts(str(number), 0, 0)
30                         number-=1
31                         sleep(1000)
32                         if pin3.read_digital()==1 or number==0:
33                             number=0
34                             lcd.clear()
35                             break
36             else:                  #Press the number button
37                 number=number*10+group[i][j]
38                 lcd.puts(str(number), 0, 0)
39             Pin_row[i].write_digital(0)
```

Close the LED dot matrix screen, store the values of matrix keyboard 1, 2, 3, 4, 5, 6, 7, 8, 9 in the array group, store the pin variables of the control keyboard row in Pin_row, and store the pin variables of the control keyboard column in Pin_column. Create the object lcd of class I2C_LCD1602, enter the I2C address and clear the screen, and set the pins connecting the matrix keyboard to low level.

	display.off()
--	---------------

```

group=[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
Pin_row = [pin9, pin6, pin10, pin4]
Pin_column = [pin3, pin2, pin1, pin0]
lcd = I2C_LCD1602(0x27)
lcd.clear()
number=0
for i in range(4):
    Pin_row[i].write_digital(0)
for i in range(4):
    Pin_column[i].write_digital(0)

```

Scan the rows and columns to check if the key is pressed.

If the "*" key is pressed, the number variable is assigned 0

If the "#" key is pressed, then the content of the while loop is executed. Number variable decreases by 1 every 1 s to achieve the countdown effect. During the period, if the "*" key is pressed or the value of number variable is 0, the while loop jumps out and the number variable is assigned with 0.

If the number key is pressed, the number variable is multiplied by 10 and add the corresponding key value, then reassigned to the number variable.

```

while True:
    for i in range(4):
        Pin_row[i].write_digital(1)
        for j in range(3):
            if Pin_column[j].read_digital()==1:
                sleep(100)
            if Pin_column[j].read_digital()==1:
                if i==3 and j==0:      #Press the "*" button
                    number=0
                    lcd.clear()
                elif i==3 and j==1:   #Press the "0" button
                    number=number*10
                elif i==3 and j==2:   #Press the "#" button
                    while True:
                        lcd.clear()
                        lcd.puts(str(number), 0, 0)
                        number-=1
                        sleep(1000)
                        if pin3.read_digital()==1 or number==0:
                            number=0
                            lcd.clear()
                            break
                else:                  #Press the number button
                    number=number*10+group[i][j]
                    lcd.puts(str(number), 0, 0)
    Pin_row[i].write_digital(0)

```

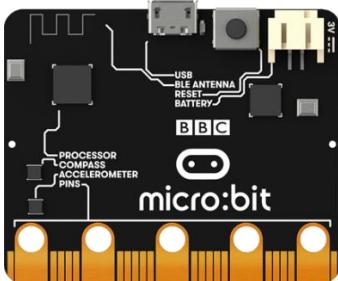
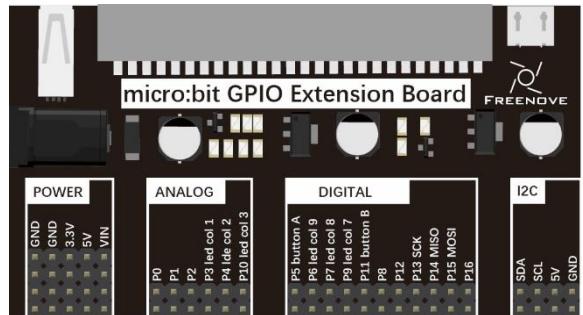
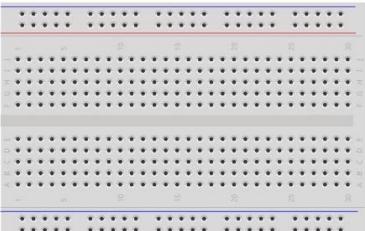
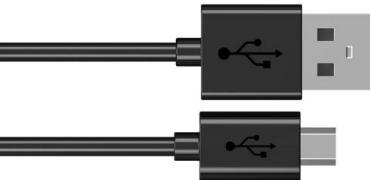
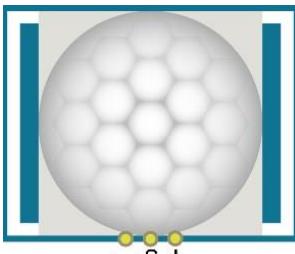
Chapter 26 Infrared Motion Sensor

In this chapter, we will learn a widely used sensor, Infrared Motion Sensor.

Project 26.1 Sense Light

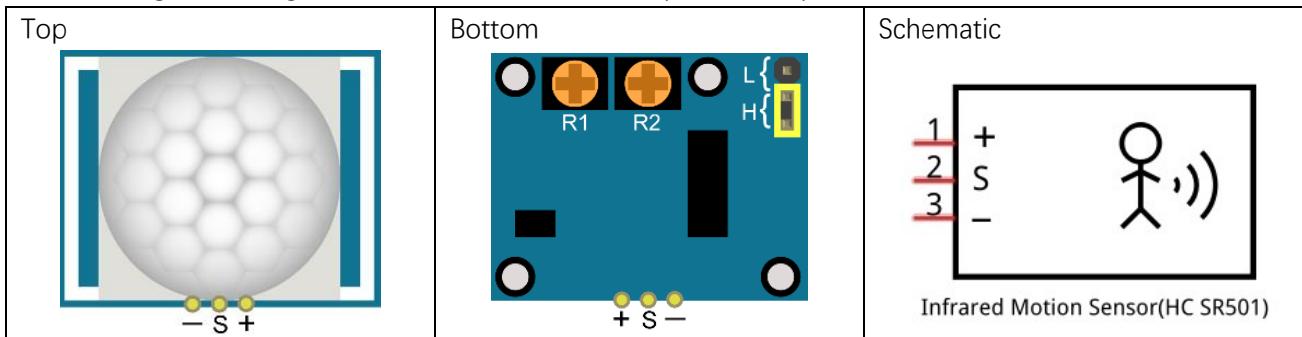
In this project, we will make a sense LED, with the human body infrared pyroelectric sensors. When someone get close to the LED, it will light automatically. On the contrary, it will be out. This infrared motion sensor is a kind of sensor, which can detect the infrared emitted by human and animals.

Component list

Microbit x1		Expansion board x1	
Breakboard x1		USB cable x2	
HC-SR501 x1		LED x1	
		Resistor 220Ω x1	
		F/M x4 M/M x1	

Component knowledge

The following is the diagram of infrared Motion sensor(HC SR-501):



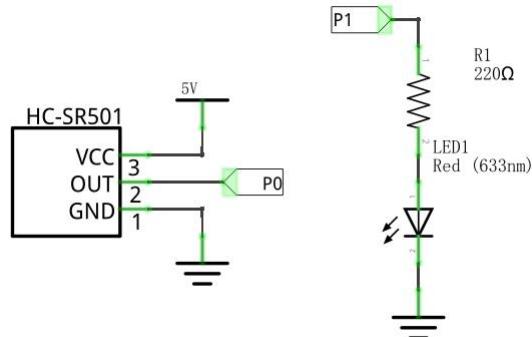
Description:

1. Working voltage: 5v-20v(DC) Static current: 65uA.
2. Automatic trigger. When the body enter into the active area of sensor, the module will output high level (3.3V). When body leave out, it will output high level lasting for time T, then output low level(0V). Delay time T can be adjusted by potentiometer R1.
3. According to the position of jumper cap, you can choose non-repeatable trigger mode or repeatable mode.
L: non-repeatable trigger mode. The module output high level after sensing body, then when delay time is over, the module will output low level. And during high level time, the sensor doesn't sense body anymore.
H: repeatable trigger mode. The distinction from L is that it can sense body until body leaves during the period of outputting high level. Then it starts to time and output low level after delaying T time.
4. Induction block time: the induction will stay in block condition and do not induce external signal at a little time (less than delay time) after outputting high level or low level
5. Initialization time: the module needs about 1 minute to initialize after powered on. During this period, it will output high or low level alternately.
6. In consideration of feature of this sensor, when body get close or away from edgewise side, the sensor will work with high sensitively. When body get close or away in vertical direction, the sensor can't work well, which should get your attention. Sensing distance is adjusted by potentiometer.

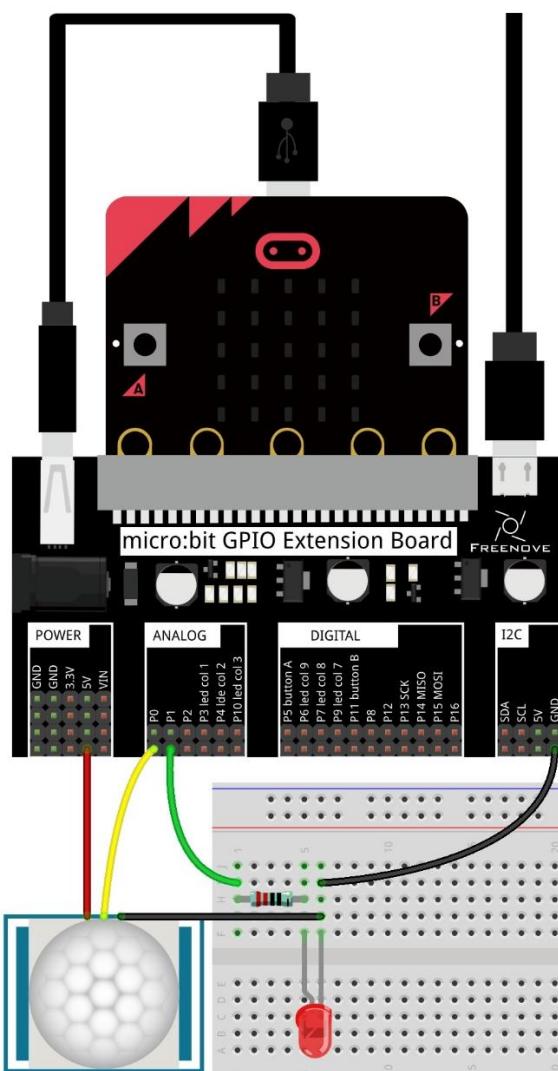
We can regard this sensor as a simple inductive switch when in use.

Circuit

Schematic diagram



Hardware connection



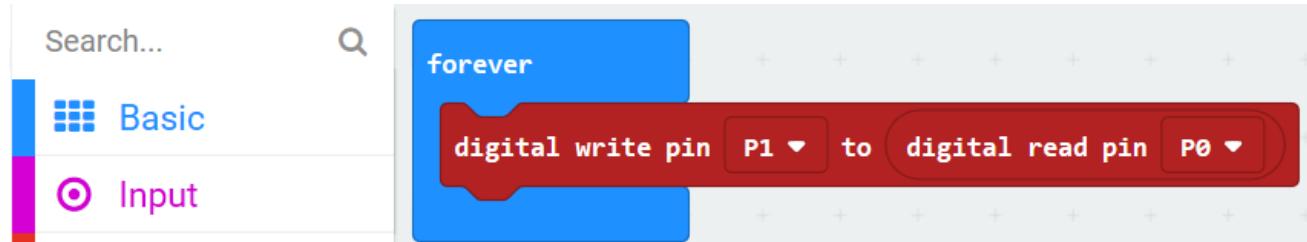
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

File type	Path	File name
HEX file	../Projects/BlockCode/26.1_SenseLight	SenseLight.hex

After import successfully, the code is shown as below:



After checking the connection of the circuit and confirming that the connection of the circuit is correct, download the code into micro:bit. Then HC-SR501 module is initialized for about one minute. After initialization, when someone is active in its detection range, the LED will light up, or it will not respond. Two potentiometers can adjust the detection distance and Induction block time.

If human activity is detected, the module will output high level. If not, the module will always output low level. The level read from P0 pin is written to the P1 pin, so that the level of P0 and P1 pin will be consistent to control the LED.

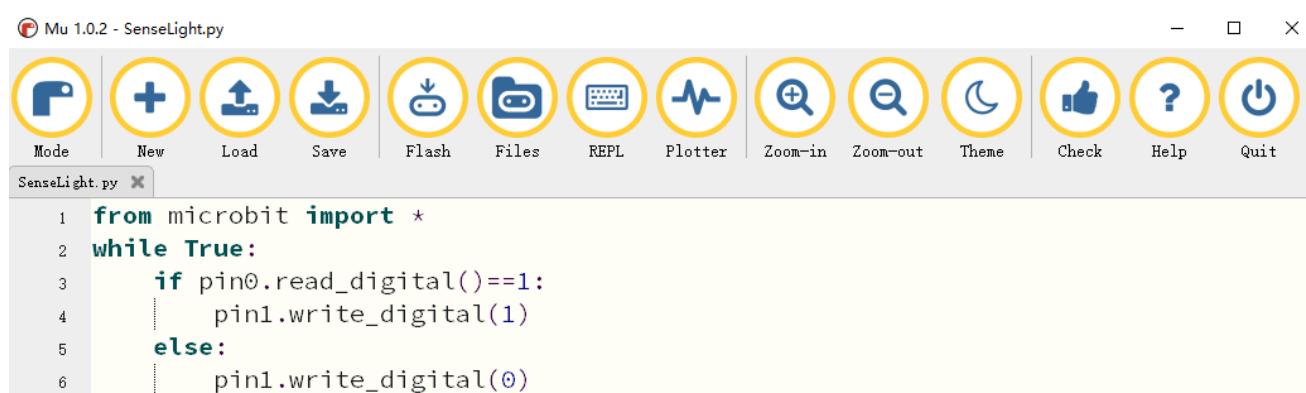


Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file/Projects/PythonCode/26.1_SenseLight	SenseLight.py

After load successfully, the code is shown as below:



```

from microbit import *
while True:
    if pin0.read_digital()==1:
        pin1.write_digital(1)
    else:
        pin1.write_digital(0)

```

After checking the connection of the circuit and confirming that the connection of the circuit is correct, download the code into micro:bit. Then HC-SR501 module is initialized for about one minute. After initialization, when someone is active in its detection range, the LED will light up, or it will not respond. Two potentiometers can adjust the detection distance and Induction block time.

The following is the program code:

```

1 from microbit import *
2 while True:
3     if pin0.read_digital()==1:
4         pin1.write_digital(1)
5     else:
6         pin1.write_digital(0)

```

If human activity is detected, the module will output high level. If not, the module will always output low level. According to the read level of the P1 pin, the P0 pin also outputs the same level to control the LED.

```

if pin0.read_digital()==1:
    pin1.write_digital(1)
else:
    pin1.write_digital(0)

```

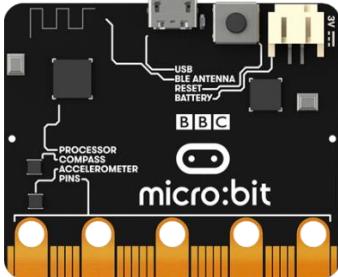
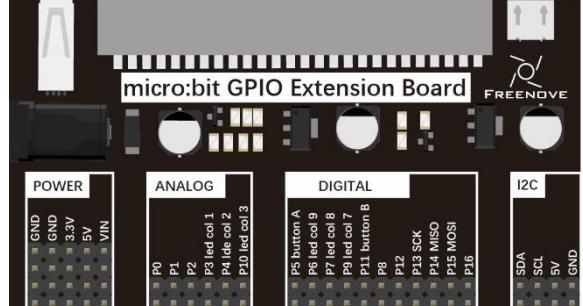
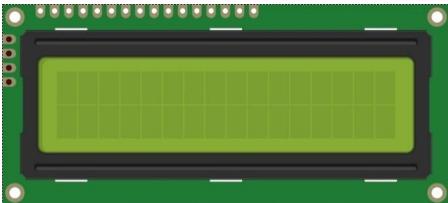
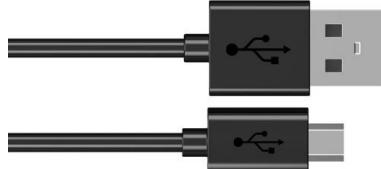
Chapter 27 Ultrasonic Ranging

In this chapter, we will learn the common ultrasonic module HC-SR04.

Project 27.1 Ultrasonic Ranging

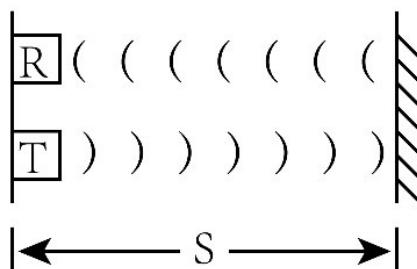
This project uses HC-SR04 ultrasonic module to measure the distance of obstacles in front of it and displays it on LCD screen.

Component list

Microbit x1	Expansion board x1
	
I2C LCD1602 Module x1	USB cable x2
	
F/F x8	HC-SR04 x1
	

Component Knowledge

Ultrasonic ranging module use the principle that ultrasonic will reflect when it encounters obstacles. Start counting the time when ultrasonic is transmitted. And when ultrasonic encounters an obstacle, it will reflect back. The counting will end after ultrasonic is received, and the time difference is the total time of ultrasonic from transmitting to receiving. Because the speed of sound in air is constant, and is about $v=340\text{m/s}$. So we can calculate the distance between the model and the obstacle: $s=vt/2$.



Ultrasonic module integrates a transmitter and a receiver. The transmitter is used to convert electrical signals (electrical energy) into sound waves (mechanical energy) and the function of the receiver is opposite. The object picture and the diagram of HC-SR04 ultrasonic module are shown below:



Pin description:

VCC	power supply pin
Trig	trigger pin
Echo	Echo pin
GND	GND

Technical specs:

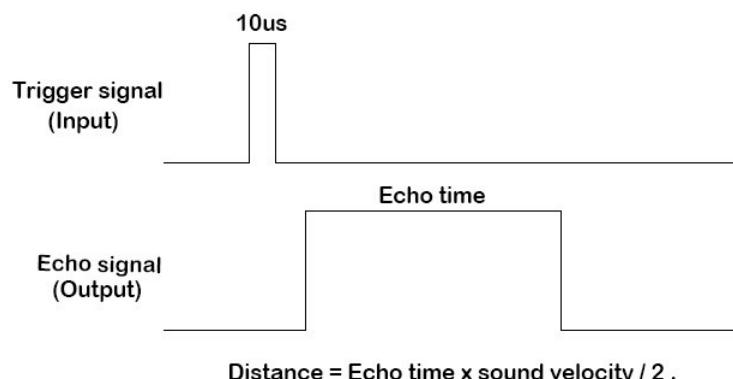
Working voltage: 5V

Working current: 12mA

Minimum measured distance: 2cm

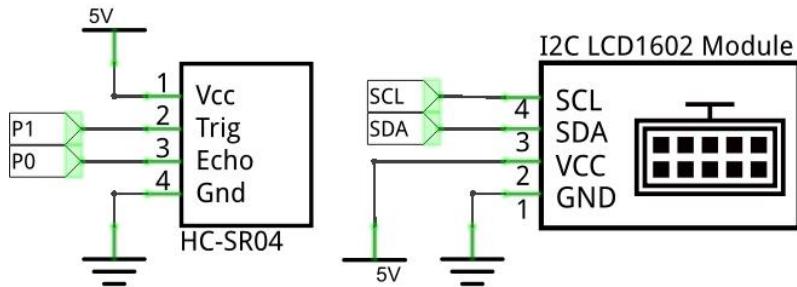
Maximum measured distance: 200cm

Instructions for use: output a high-level pulse in Trig pin lasting for least 10uS. Then the module begins to transmit ultrasonic. At the same time, the Echo pin will be pulled up. When the module receives the returned ultrasonic, the Echo pin will be pulled down. The duration of high level in Echo pin is the total time of the ultrasonic from transmitting to receiving, $s=vt/2$.

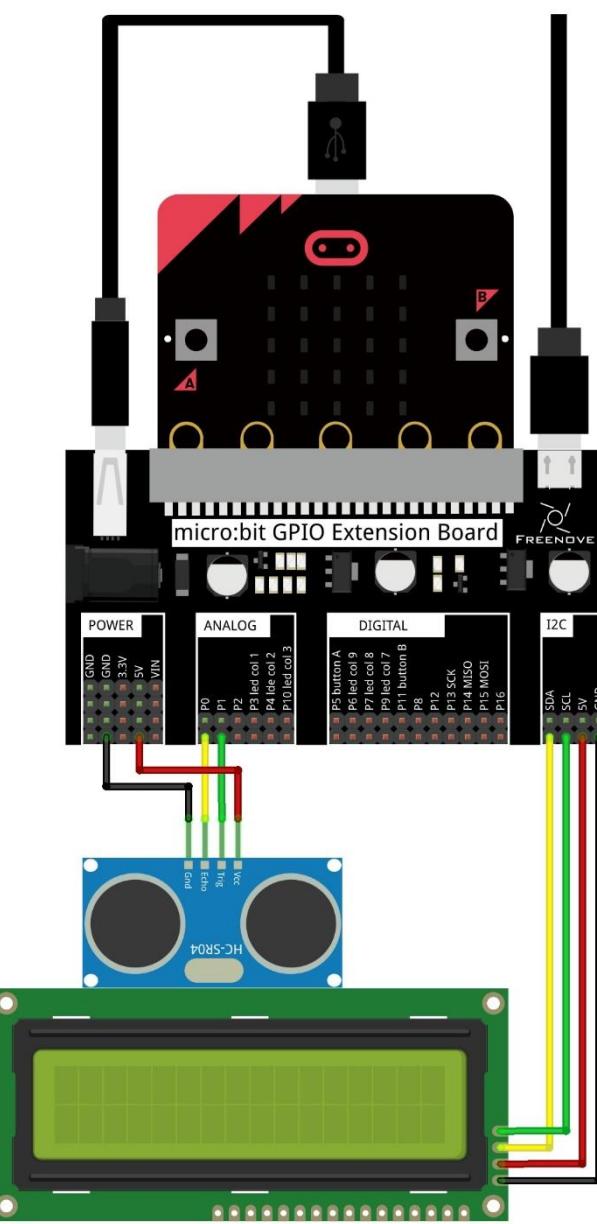


Circuit

Schematic diagram



Hardware connection



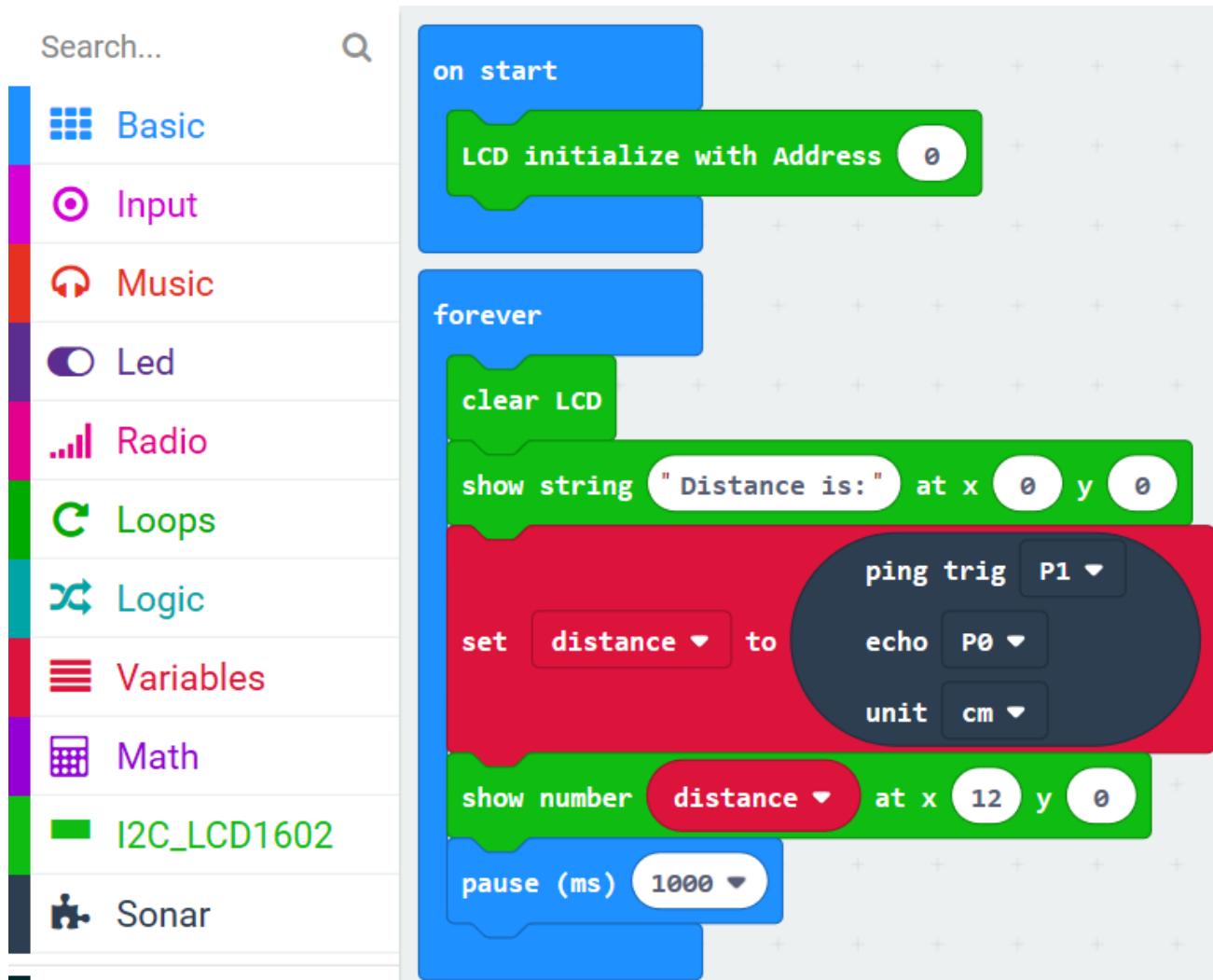
Block code

Open makecode first. Import the .hex file. The path is as below:

([How to import project](#))

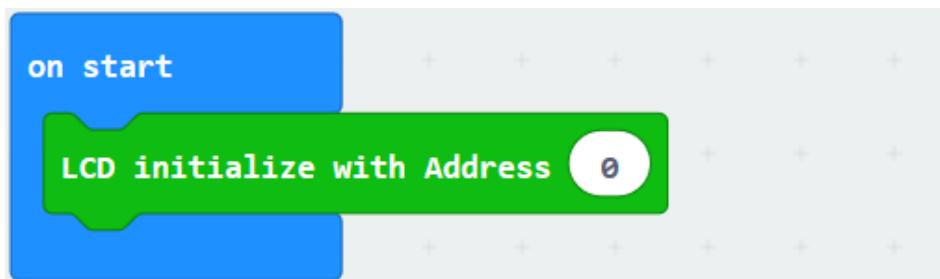
File type	Path	File name
HEX file	../Projects/BlockCode/27.1_UltrasonicRanging	UltrasonicRanging.hex

After import successfully, the code is shown as below:

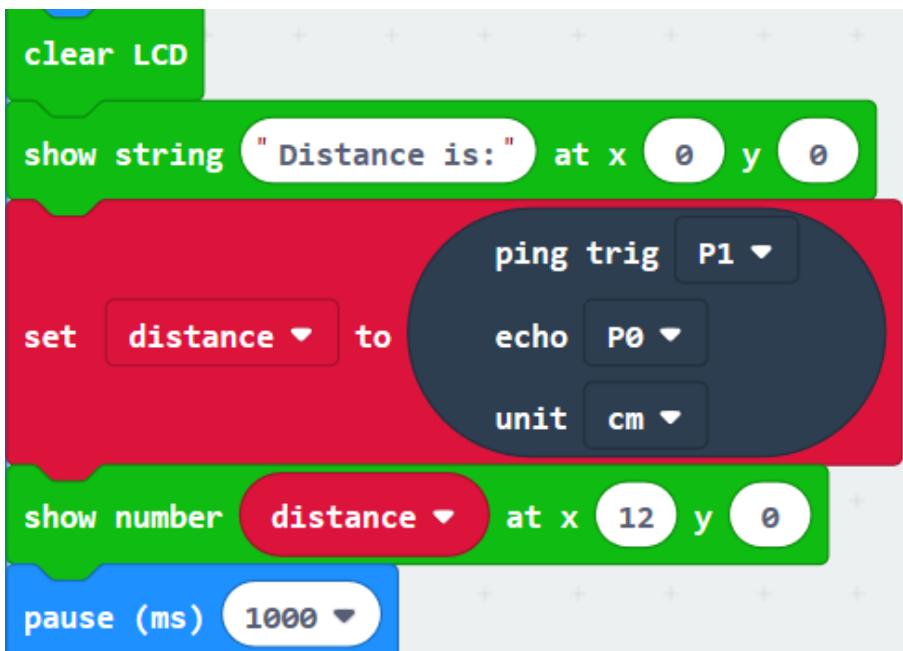


After checking the connection of the circuit and confirming the correct connection of the circuit, download the code into micro:bit. The LCD screen will show the distance between the obstacle and the ultrasonic module in CM.

Initialize LCD.



The distance of obstacles measured by the ultrasonic module will be assigned to the variable **distance**, and then displayed on LCD. LCD refreshes every 1 second.

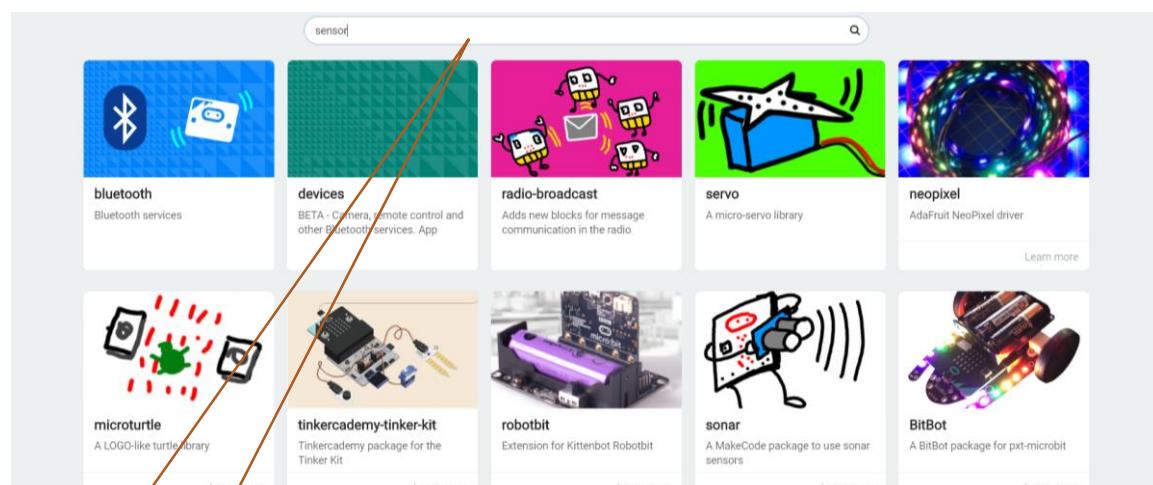
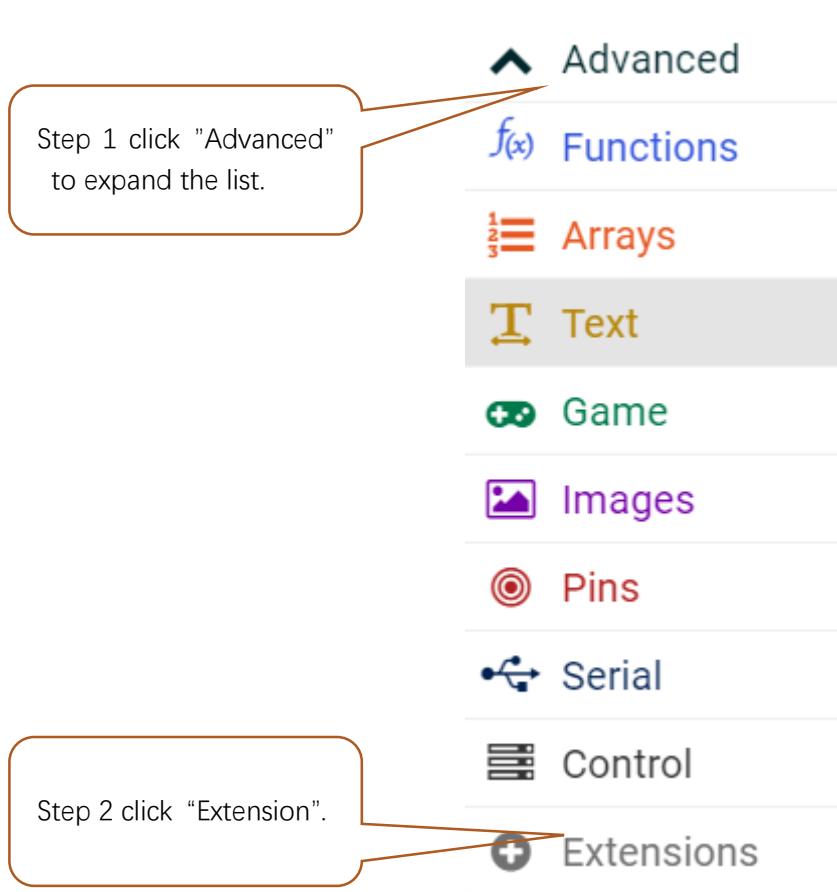


Reference

Block	Function
<p>This block is a composite of three smaller blocks: ping trig (with pin P0), echo (with pin P0), and unit (μs). It is used to measure the distance to an object by sending a short pulse of sound and measuring the time it takes for the echo to return.</p>	<p>It can return the distance of obstacles detected by sensor.</p>

Extensions

If you want to import the ultrasonic module expansion block into the new project, follow the steps below to add it.



Type "sensors" for search.

The image shows the Microsoft MakeCode Extensions page and the block palette.

Extensions Page:

- sonar:** A Microsoft MakeCode package to handle sonar sensors and pings.
- DHT11_DHT22:** MakeCode extension for DHT11/DHT22 sensors.
- bitbot:** A BitBot package for pxt-microbit.
- weather-bit:** MakeCode package for the SparkFun weather:bit board - beta.
- ringbitcar:** ElecFreaks MakeCode motor:bit package for ring:bit car.

Block Palette:

- Search bar: Search...
- Basic
- Input
- Music
- Led
- Radio
- Loops
- Logic
- Variables
- Math
- Sonar

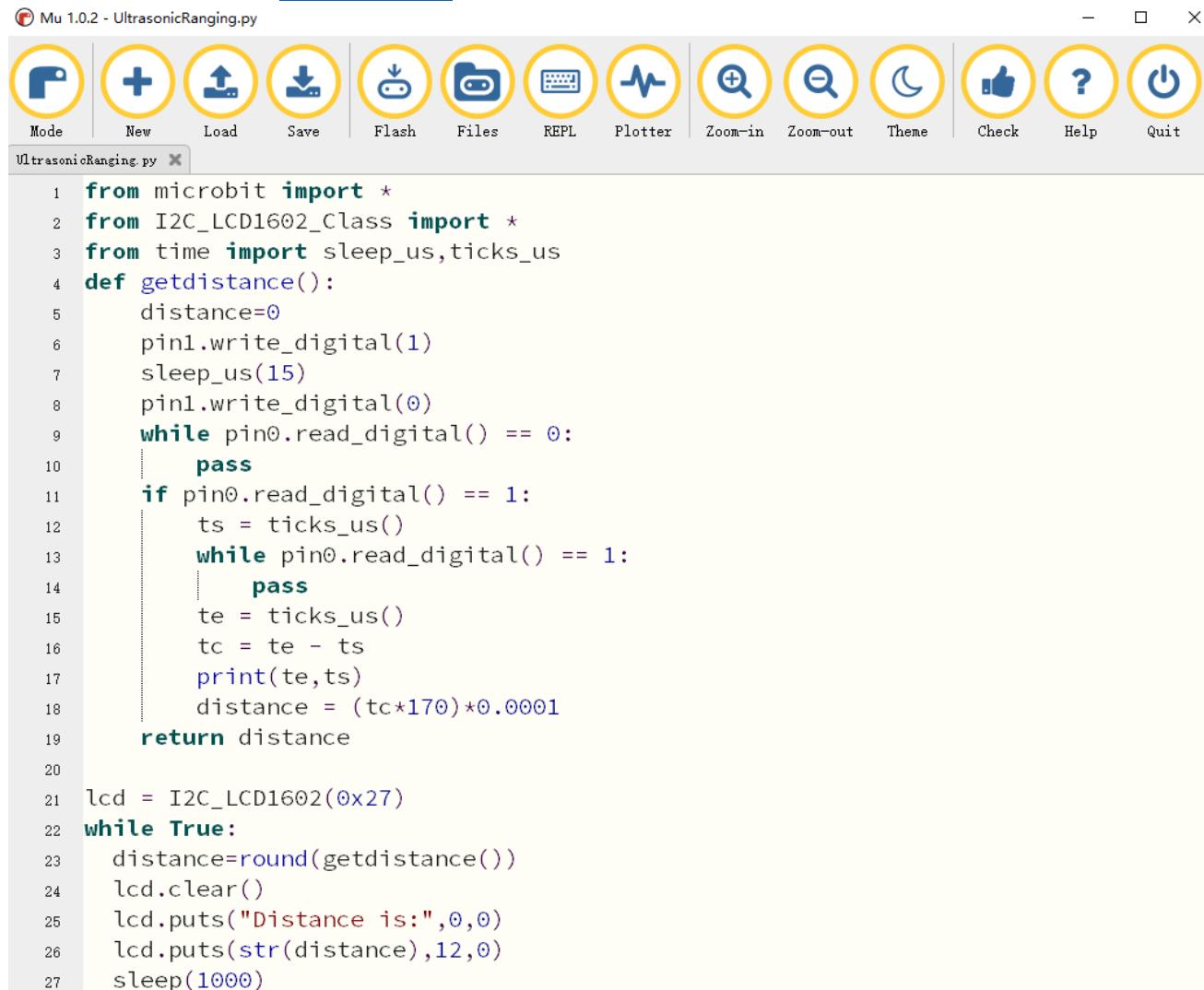
A callout bubble on the left says "Click to add." pointing to the Sonar category in the palette. A callout bubble on the right says "Completed." pointing to the Sonar category in the palette.

Python code

Open the .py file with Mu. Code path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/27.1_UltrasonicRanging	UltrasonicRanging.py

After the code is loaded, as shown below, import the "I2C_LCD1602_Class.py" file to micro:bit before downloading the code. ([How to import?](#))



```

1 from microbit import *
2 from I2C_LCD1602_Class import *
3 from time import sleep_us,ticks_us
4 def getdistance():
5     distance=0
6     pin1.write_digital(1)
7     sleep_us(15)
8     pin1.write_digital(0)
9     while pin0.read_digital() == 0:
10         pass
11     if pin0.read_digital() == 1:
12         ts = ticks_us()
13         while pin0.read_digital() == 1:
14             pass
15         te = ticks_us()
16         tc = te - ts
17         print(te,ts)
18         distance = (tc*170)*0.0001
19     return distance
20
21 lcd = I2C_LCD1602(0x27)
22 while True:
23     distance=round(getdistance())
24     lcd.clear()
25     lcd.puts("Distance is:",0,0)
26     lcd.puts(str(distance),12,0)
27     sleep(1000)

```

After importing the I2C_LCD1602_Class.py file, check the connection of the circuit and confirm that the connection of the circuit is correct. After downloading the code into micro:bit, you can see that the LCD screen will show the distance between the obstacle and the ultrasonic module. The unit is CM.

The following is the program code:

```
1  from microbit import *
2  from I2C_LCD1602_Class import *
3  from time import sleep_us, ticks_us
4  def getdistance():
5      distance=0
6      pin1.write_digital(1)
7      sleep_us(15)
8      pin1.write_digital(0)
9      while pin0.read_digital() == 0:
10         pass
11     if pin0.read_digital() == 1:
12         ts = ticks_us()
13         while pin0.read_digital() == 1:
14             pass
15         te = ticks_us()
16         tc = te - ts
17         print(te, ts)
18         distance = (tc*170)*0.0001
19     return distance
20
21 lcd = I2C_LCD1602(0x27)
22 while True:
23     distance=round(getdistance())
24     lcd.clear()
25     lcd.puts("Distance is:", 0, 0)
26     lcd.puts(str(distance), 12, 0)
27     sleep(1000)
```

The custom getdistance() function is used to get the distance between the obstacle and the ultrasonic module. The unit of return is CM.

```
def getdistance():
    distance=0
    pin1.write_digital(1)
    sleep_us(15)
    pin1.write_digital(0)
    while pin0.read_digital() == 0:
        pass
    if pin0.read_digital() == 1:
        ts = ticks_us()
        while pin0.read_digital() == 1:
            pass
        te = ticks_us()
        tc = te - ts
        print(te,ts)
        distance = (tc*170)*0.0001
    return distance
```

Create the object lcd of I2C_LCD1602 class, input I2C address 0x27, call getdistance() function, get the distance of the obstacle to the ultrasonic module, assign it to the distance variable, and then display the value of the distance variable on the LCD.

```
lcd = I2C_LCD1602(0x27)
while True:
    distance=round(getdistance())
    lcd.clear()
    lcd.puts("Distance is:", 0, 0)
    lcd.puts(str(distance), 12, 0)
    sleep(1000)
```

Reference

getdistance()

Get the distance from the ultrasonic module to the obstacle. The unit is CM.



What's next?

Thanks for your reading.

This tutorial is all over here. If you find any mistakes, missions or you have other ideas and questions about contents of this tutorial or the kit and etc, please feel free to contact us, and we will check and correct it as soon as possible.support@freenove.com

If you want to learn more about micro:bit, we have a smart Car,micro:bit Rover. You can visit our website to purchase.<http://www.freenove.com/store.html>

We will continue to launch cost-effective, innovative and exciting products.

Thank you again for choosing Freenove products.