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

FREENOVE®

FREE YOUR INNOVATION

Contents

Preface

Do you want to learn programming?

Nowadays, Program is developed into the younger age group, and everyone programming is a trend. From Arduino and Raspberry Pi to micro:bit, simple graphical programming makes programming for kids possible. 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, its simple graphical programming interface allows you to master it easily. It doesn't require any professional programming software; just simply a browser is enough to program it. So, no matter your computer system is Windows, Linux or Mac, you can program it. And you can also program it with Python.

It attracts a lot of fans in the world who are keen to exploration, innovation and DIY and have 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.

With Micro:bit, we can make a lot of projects and by adding kits to breadboard, we can carry out more interesting projects like ultrasonic ranging, gravity control, playing music, etc.

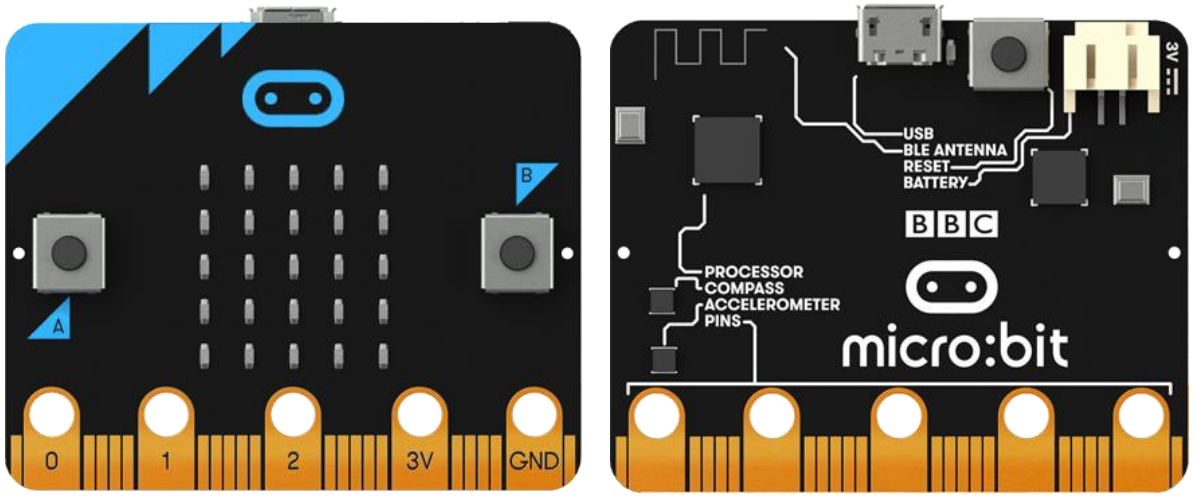
In each learning chapter of this tutorial, we provide program source code with detailed program explanations and burnable binaries, so that you can understand the meaning of each section of program.

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

This chapter is the Start Point in the journey to build and explore Micro:bit and Micro:Rover electronic projects.

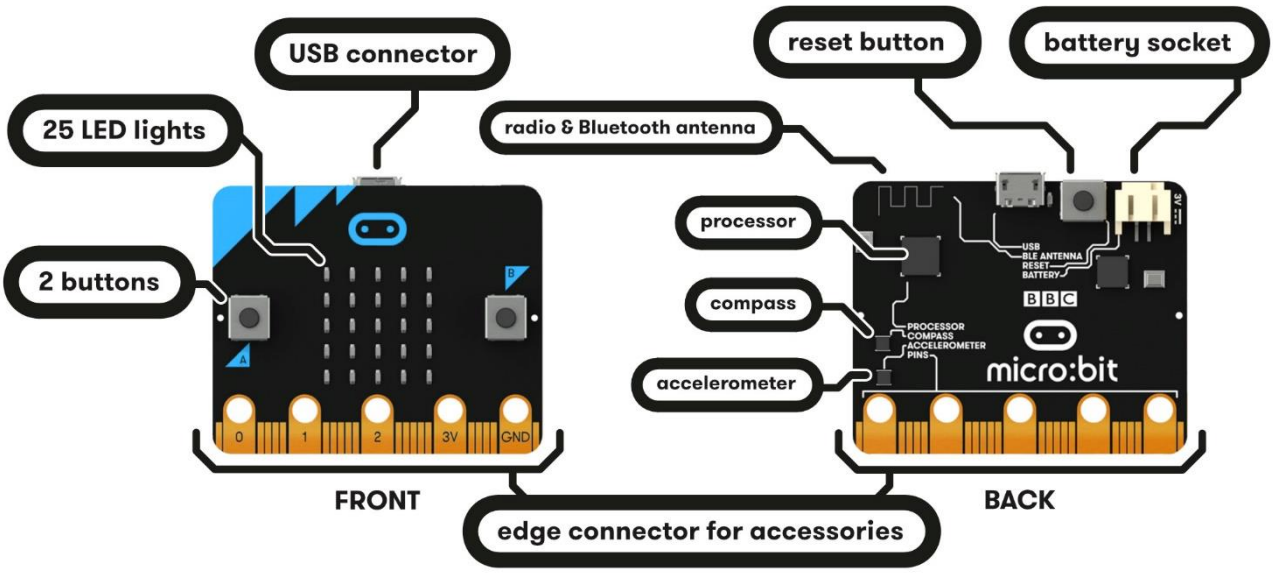
Meet micro:bit



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

For more contents, please refer to:
https://microbit.org/guide/

Features



Your micro:bit has the following physical features:

- 25 individual 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

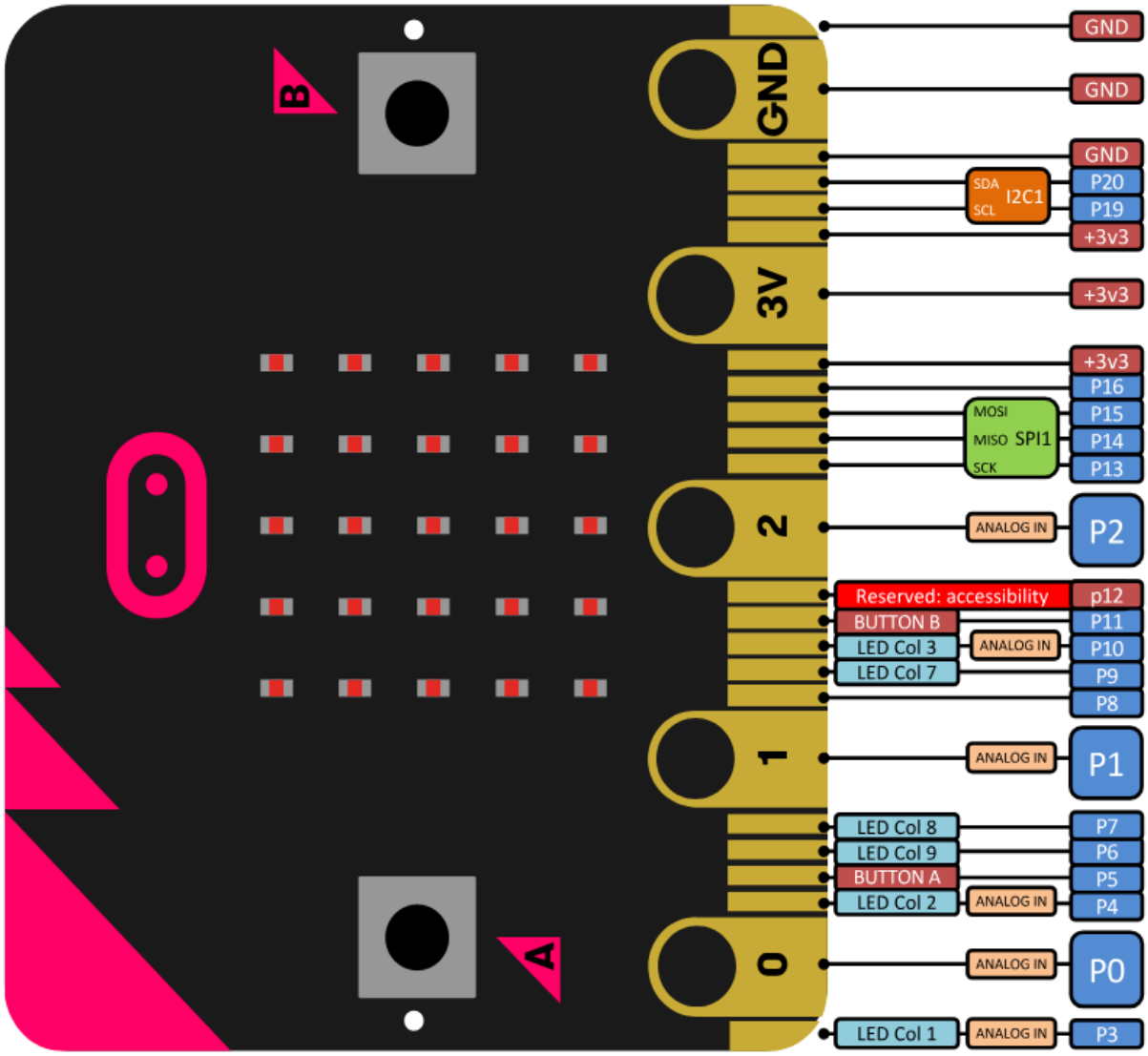
For more details, please refer to:
https://microbit.org/guide/features/

Hardware

It is not required for beginners to master this section, but a brief understanding is necessary. However, 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

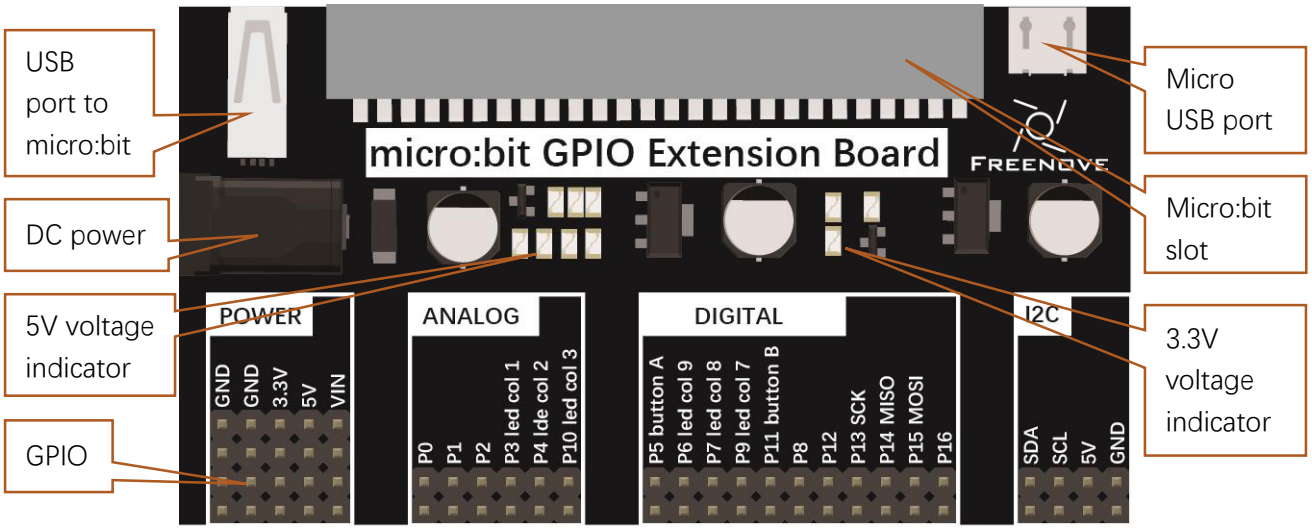
GPIO, namely General Purpose Input/output Pins, is an important part of micro:bit for connecting external devices. All sensors and devices on Rover communicate with each other through micro:bit GPIO. The following is the GPIO serial number and function diagram of micro:bit:



Micro:bit GPIO Extension Board

Hardware and Feature

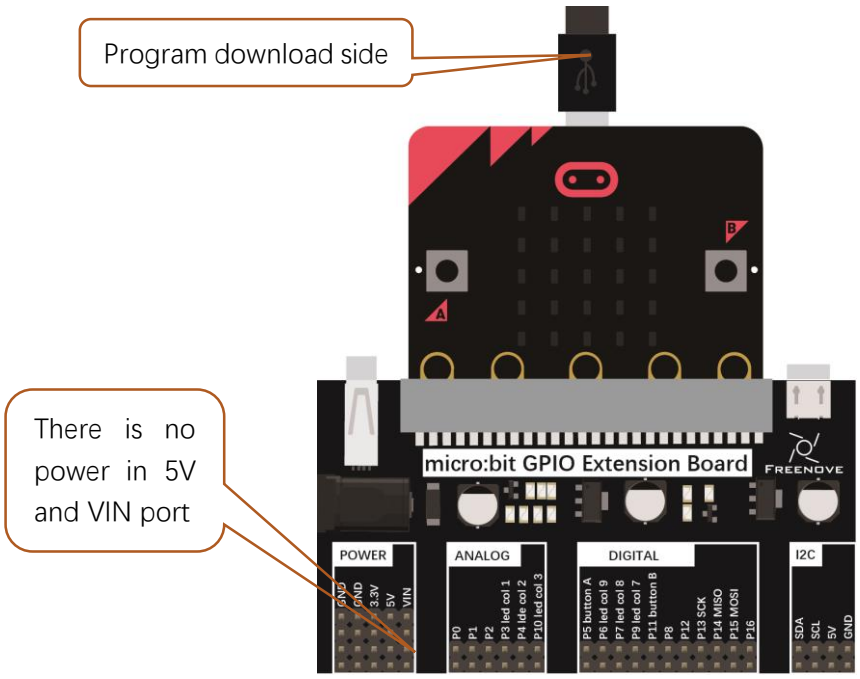
Micro:bit GPIO Extension Board is shown as below:



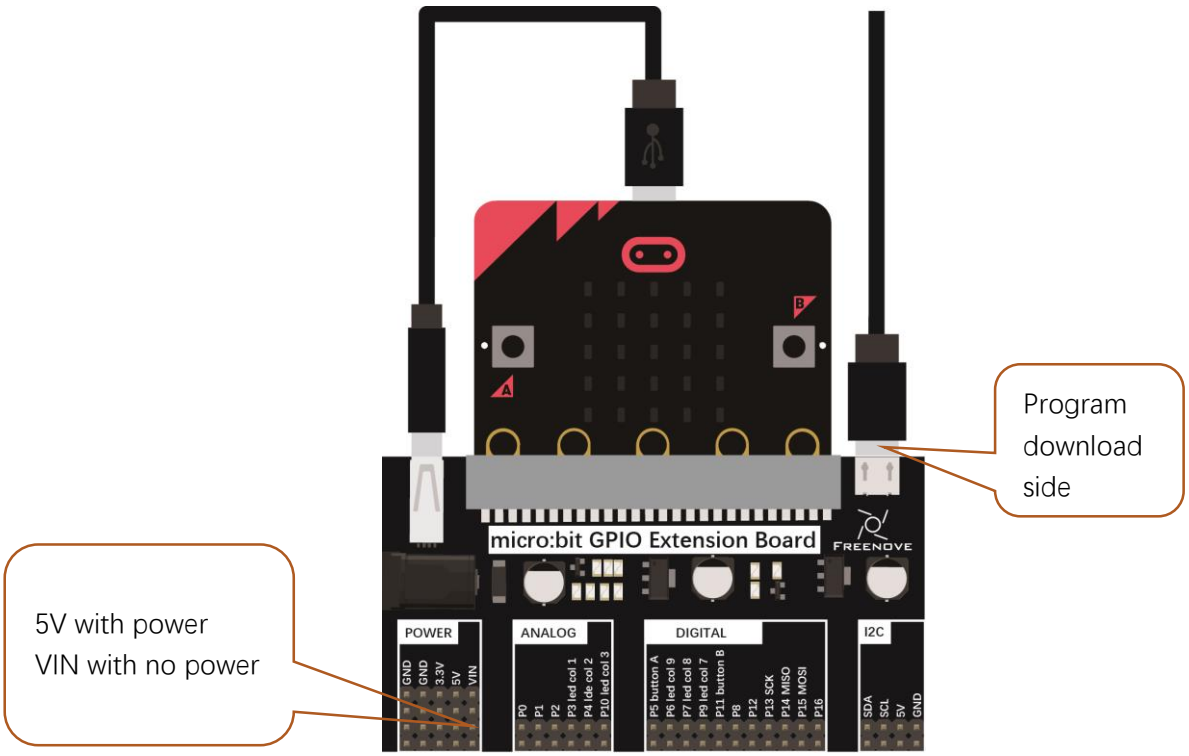
Micro:bit GPIO Extension Board is connected to micro:bit board via a slot with its GPIO connected to micro:bit's GPIO. In addition, there are also 5V and VIN(9V) IO port on the extension board to meet requirement of more devices.

How to use?

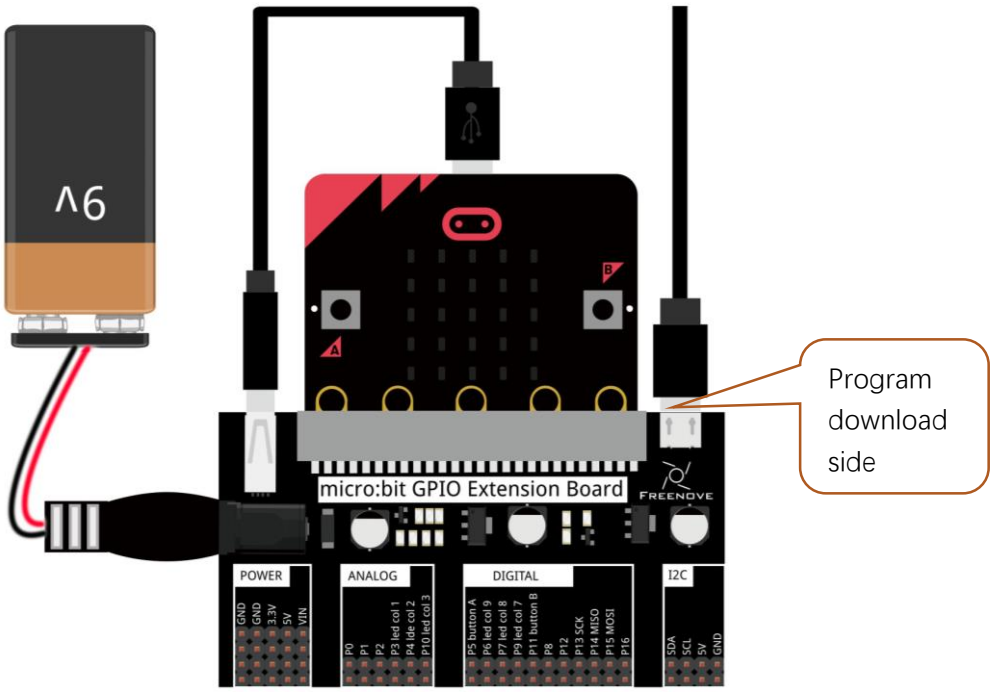
If external device doesn't require voltage of 5V and 9V, you can use the following wiring.



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



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



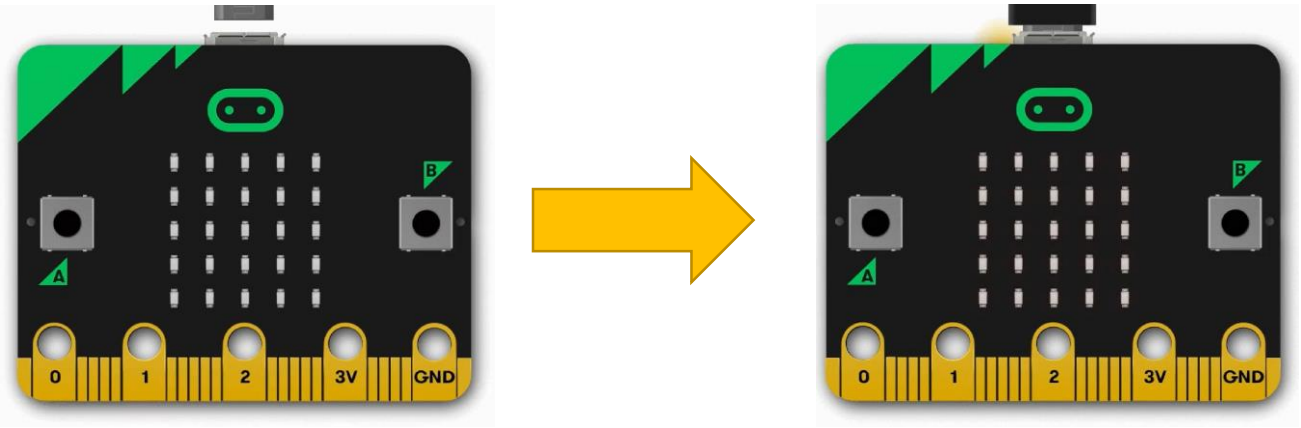
Code & Programming

Quick Start

This section describes how to write programs 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/.

Step 1: Connecting Micro:bit

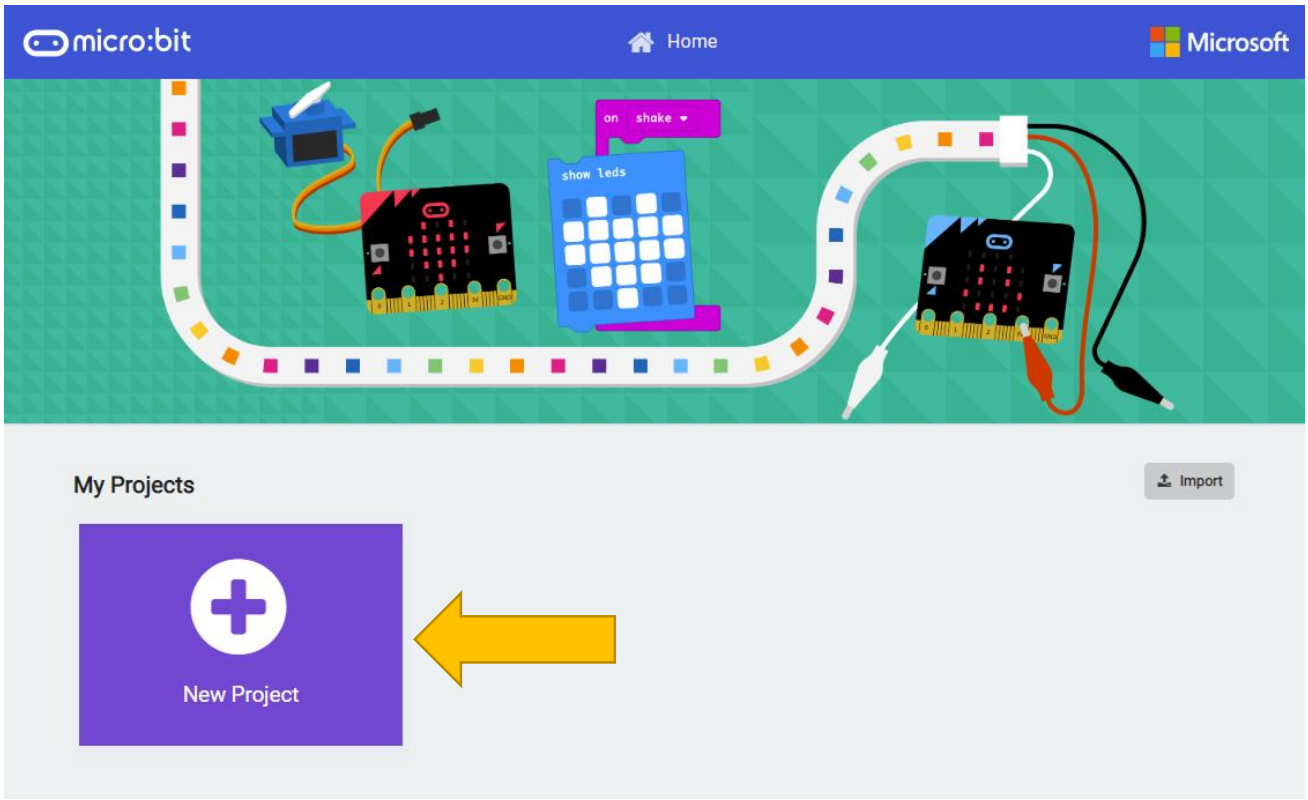
Connect the micro:bit to your computer via a micro USB cable. Macs, PCs, Chromebooks and Linux systems (including Raspberry Pi) are all supported.



Step 2: Write Program

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

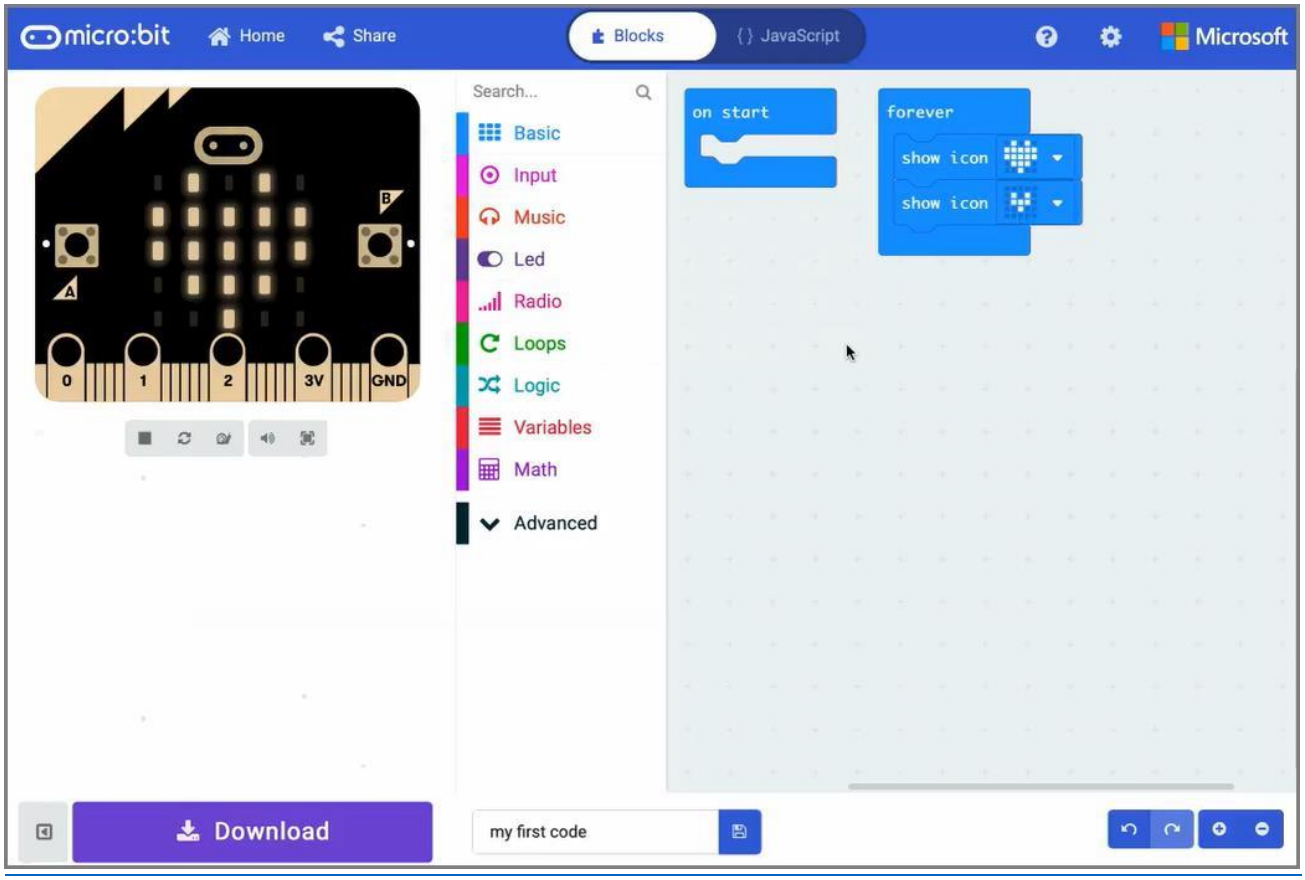
If your computer has Windows 10 operating system, you can also use Windows 10 App for programming, which is exactly the same as programming on browsers. Get windows 10 App(Click).



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

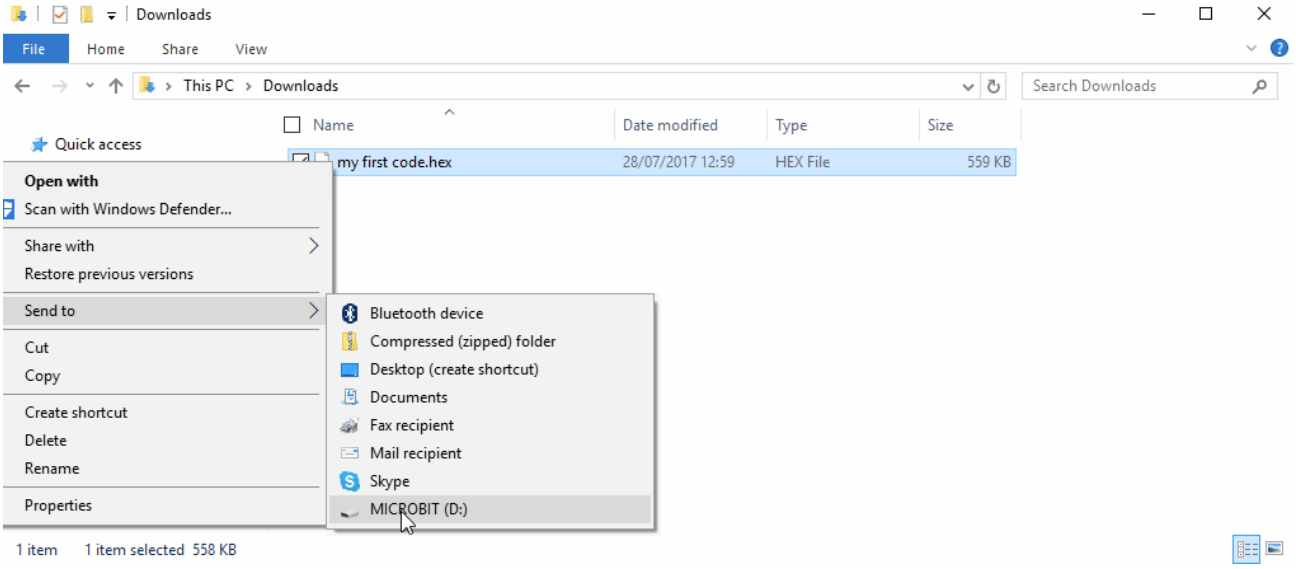
MakeCode will be further introduced in next section.



Step 3: Flashing Code to your Micro:bit

The process of transferring the .HEX file to the BBC micro:bit is called flashing.
If you write program using Windows 10 App, you just need to click the “Download” button, then the program will be downloaded directly to micro:bit without any other actions.

If you write program 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 been 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: Run the Program

The micro:bit will pause and the yellow LED on the back of the micro:bit will blink while your code is flashed. Once that's finished the code will run automatically! The micro:bit can only run one program at a time - every time you drag-and-drop a hex file onto the device over USB it will erase the current program and replace it with the new one.

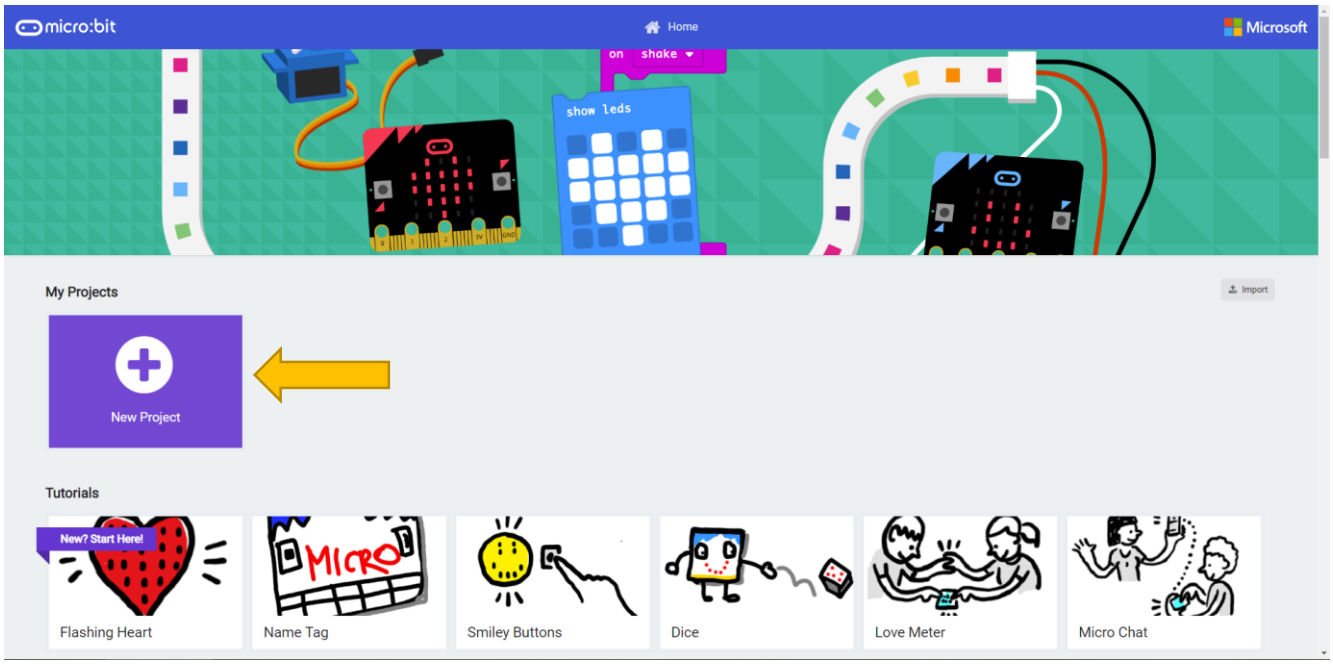
Warning

The MICROBIT drive will automatically eject and reconnect 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!

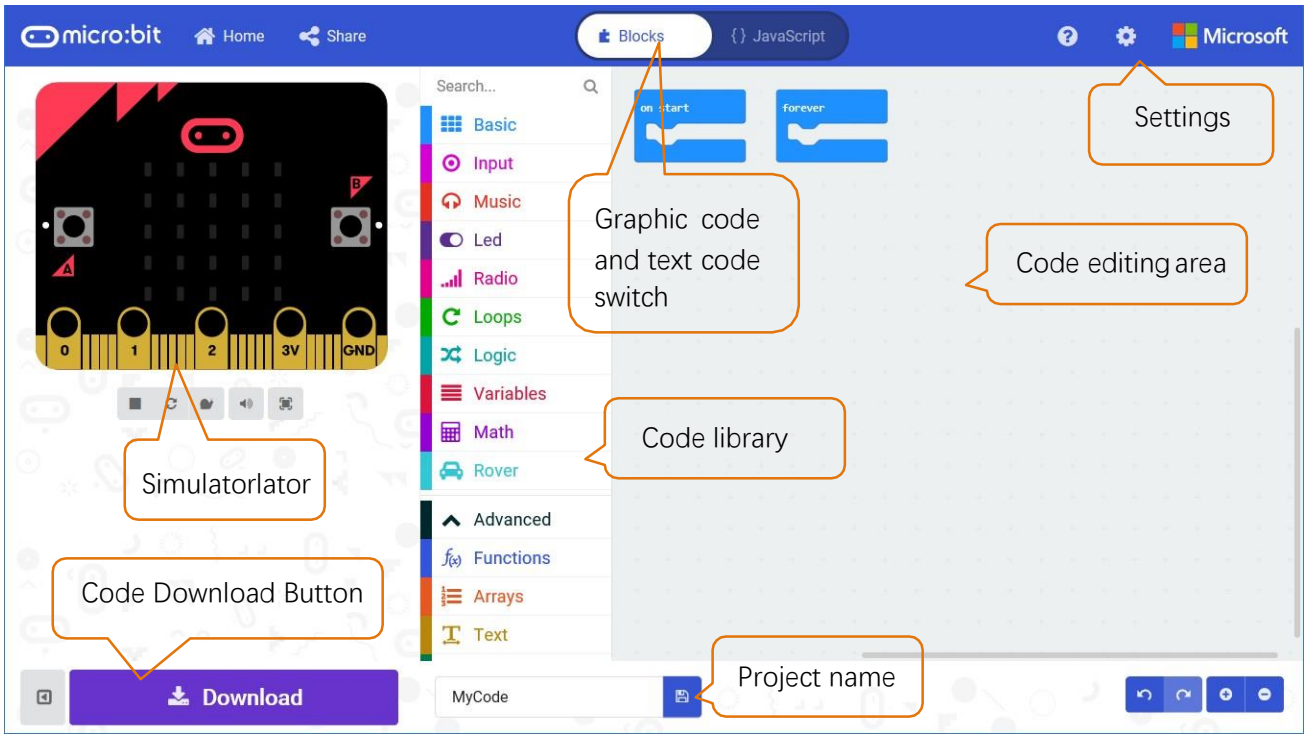
MakeCode

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 circularly.

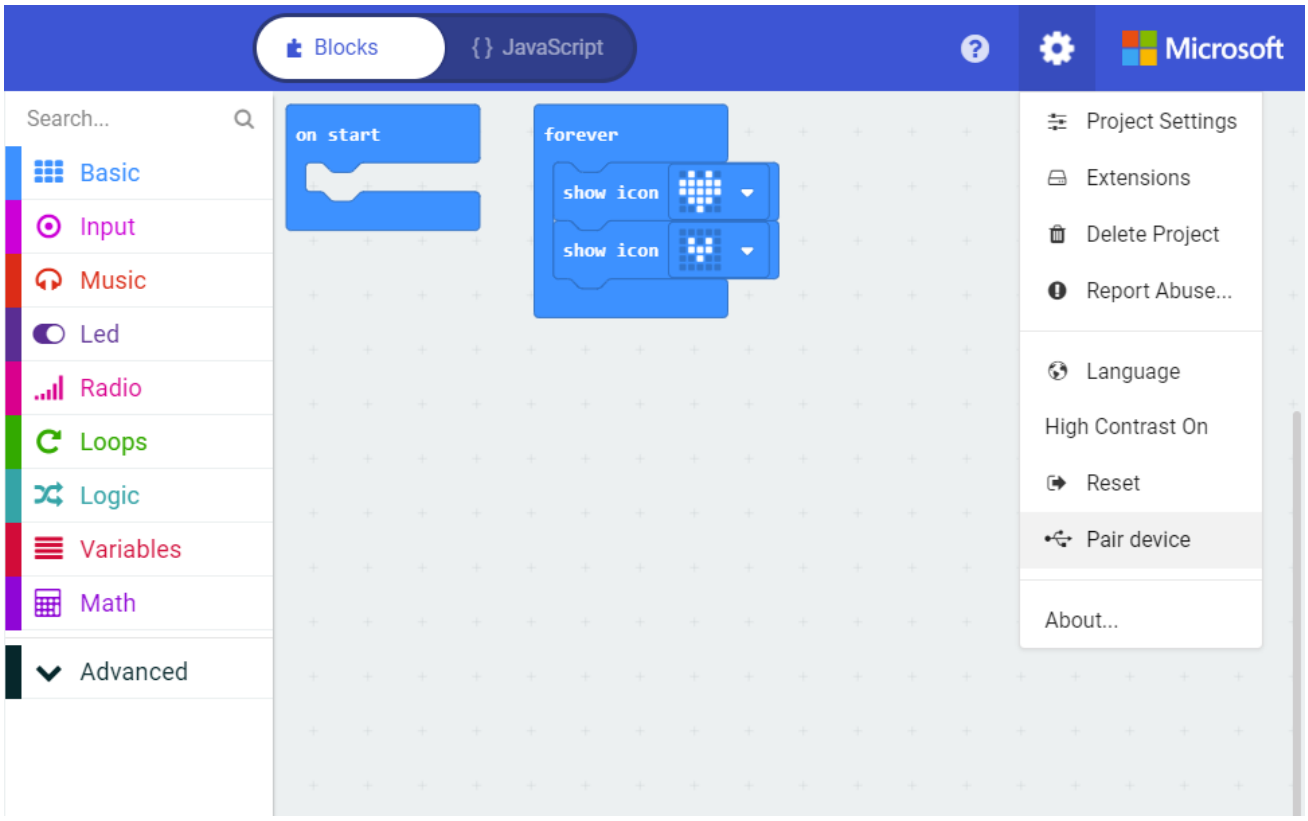
Quick Download

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**.

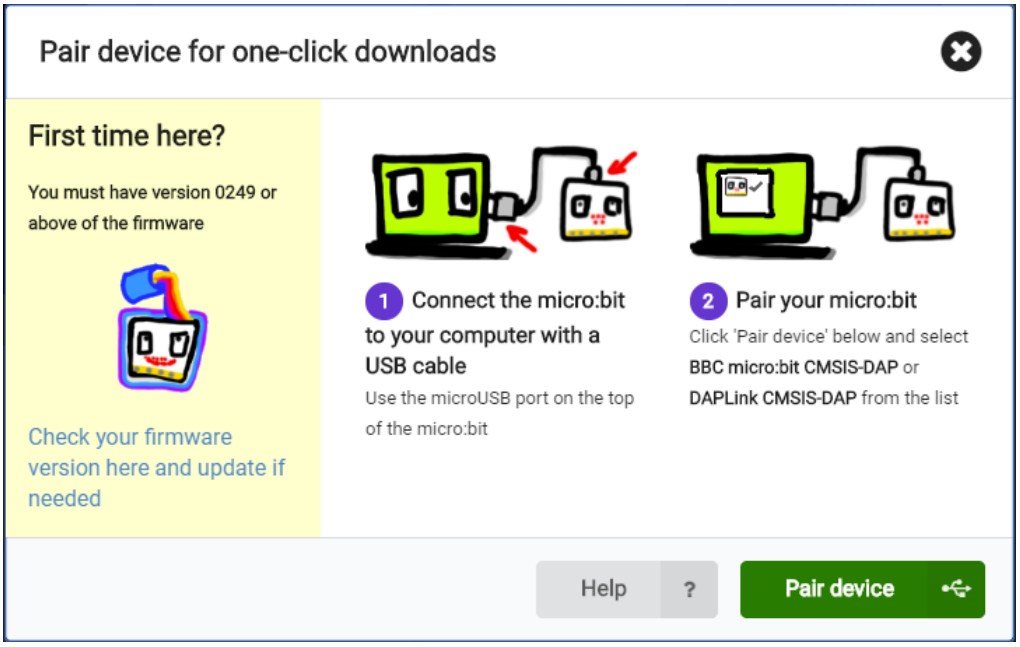
If you use browser version of MakeCode on **Google Chrome 65+ for platform Android, Chrome OS, Linux, macOS and Windows 10**, you can also download file quickly.
Here we use webUSB feature of Chrome, which allows web pages to access your USB hardware devices. We will complete the connection and pairing of the micro:bit device with the webpage in the following steps.

Pair device

Connect your 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.



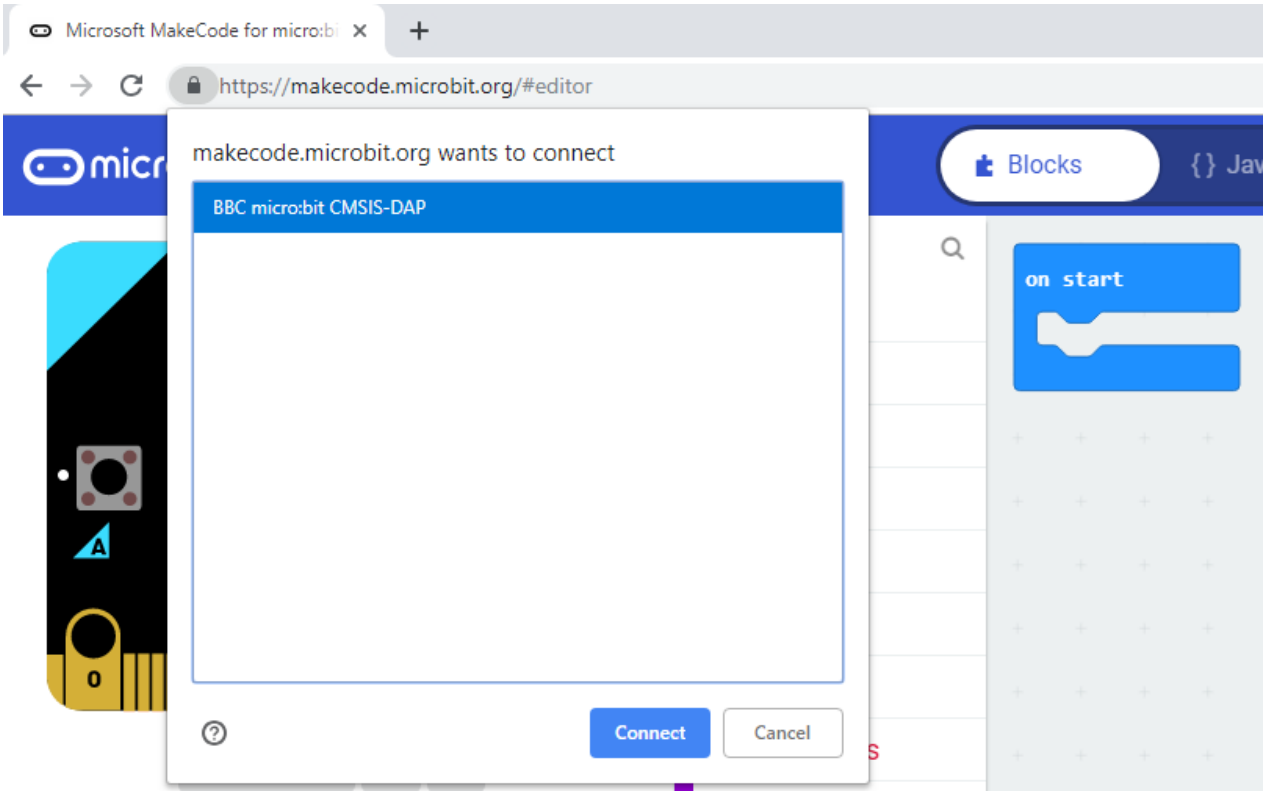
Select device on the pop-up window and click "Connect" button. If there are no devices shown on the pop-up window, please refer to following content:

https://makecode.microbit.org/device/usb/webusb/troubleshoot

We have 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, which 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 to add necessary extensions.

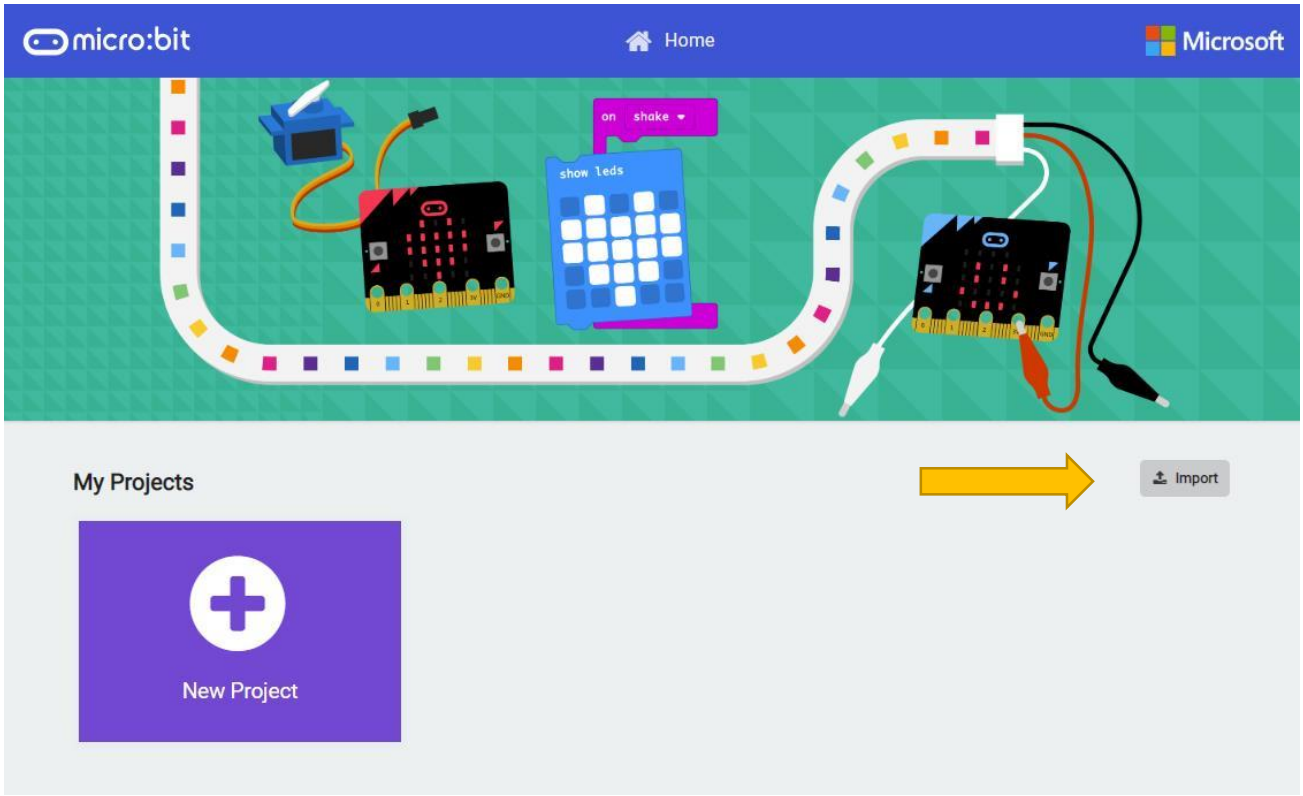
As for simple projects, it is recommended to complete the project by dragging code block.

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

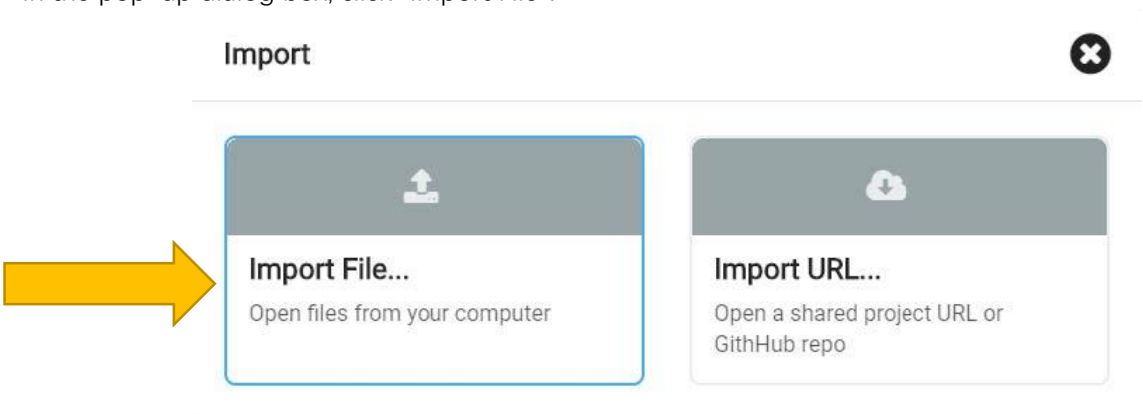
Next, we will take "Heartbeat" project as an example to introduce how to load code.

Open web version of makecode or windows 10 app version of MakeCode.

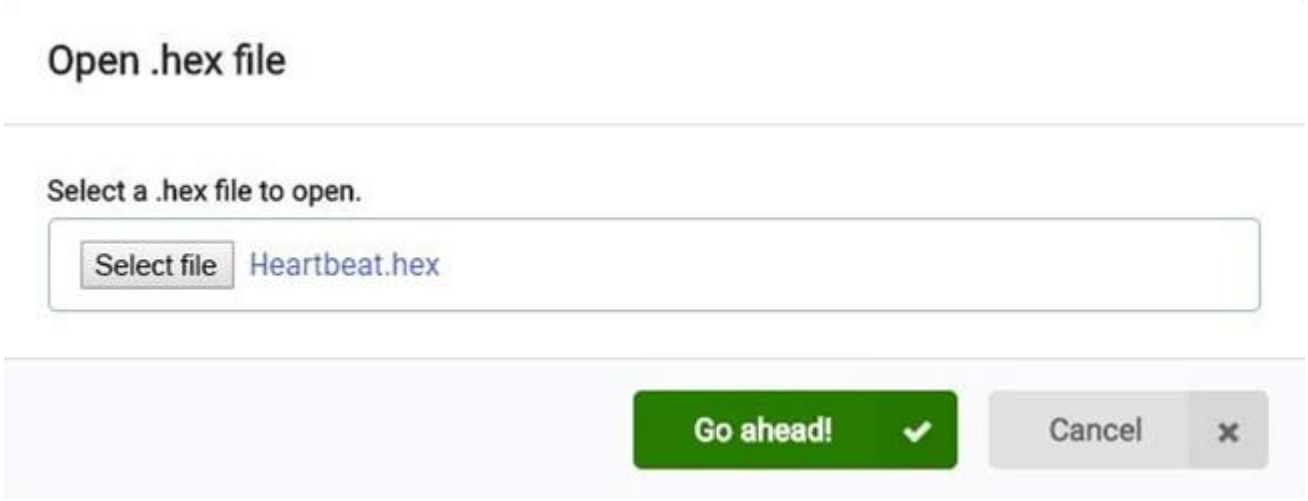
Click "Import" button on the right of HOME page.



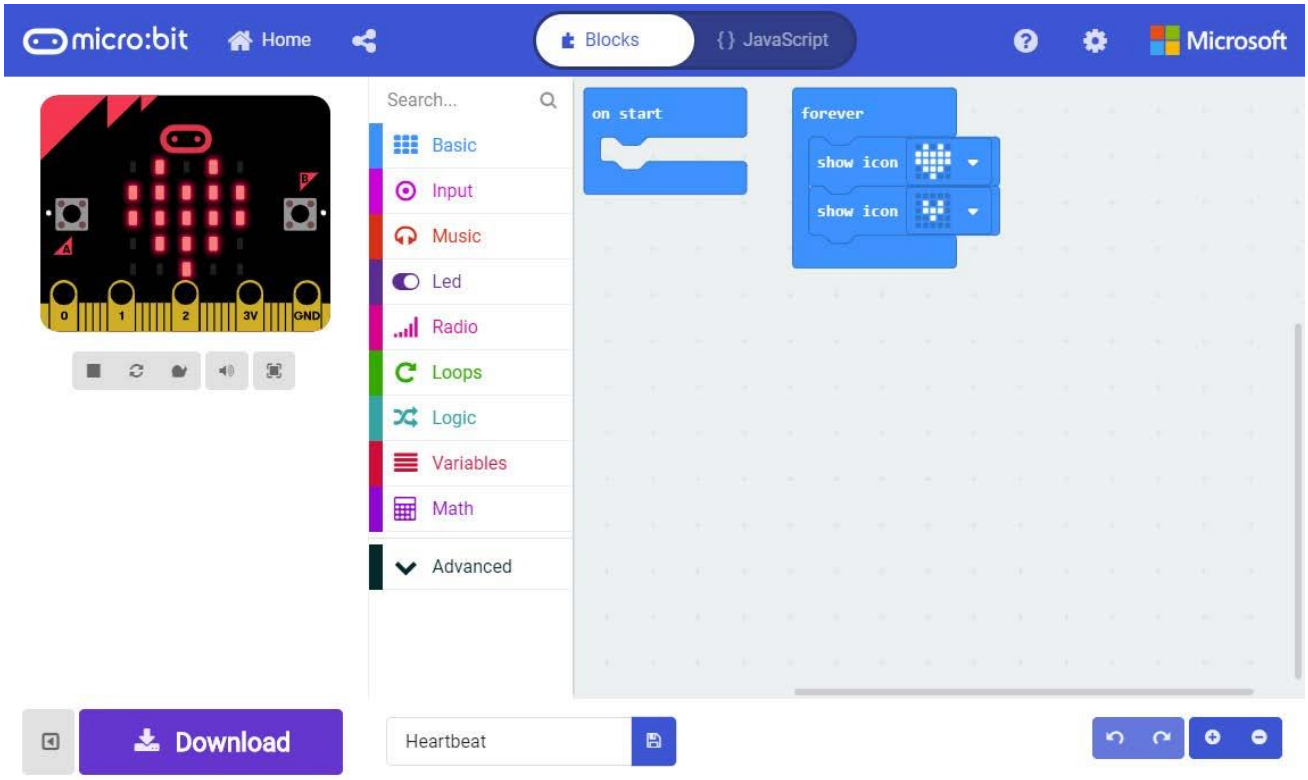
In the pop-up dialog box, click "Import File".



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



A few seconds later, the project is loaded successfully.

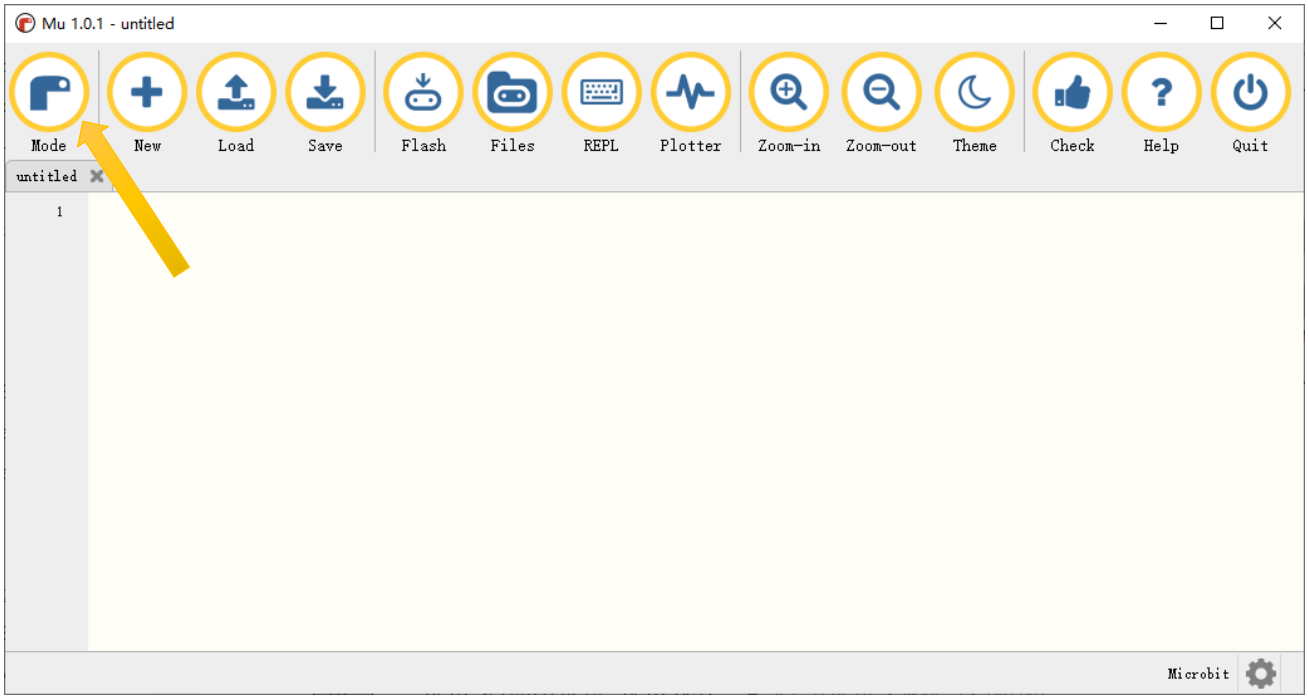


Python

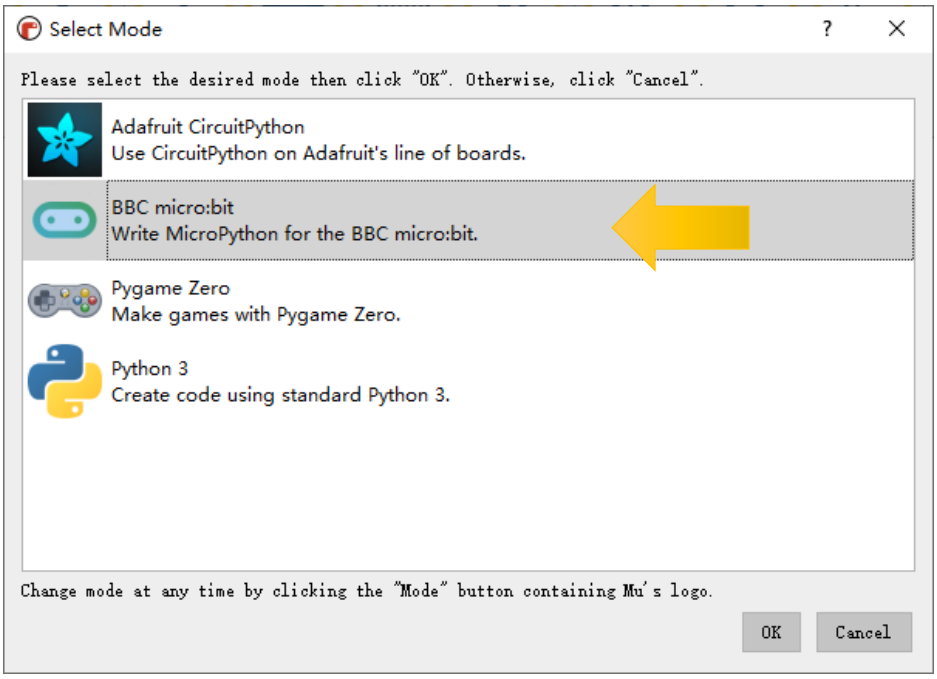
If you are not interested in python, you can skip this section.
Micro:bit can be programmed in Python. Since micro:bit is a microcontroller, the hardware difference makes it not support pure Python. Here we use MicroPython, which is specially designed for micro:bit.
MicroPython is a lean and efficient 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 a 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.
And you will see the following interface when opening it.



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

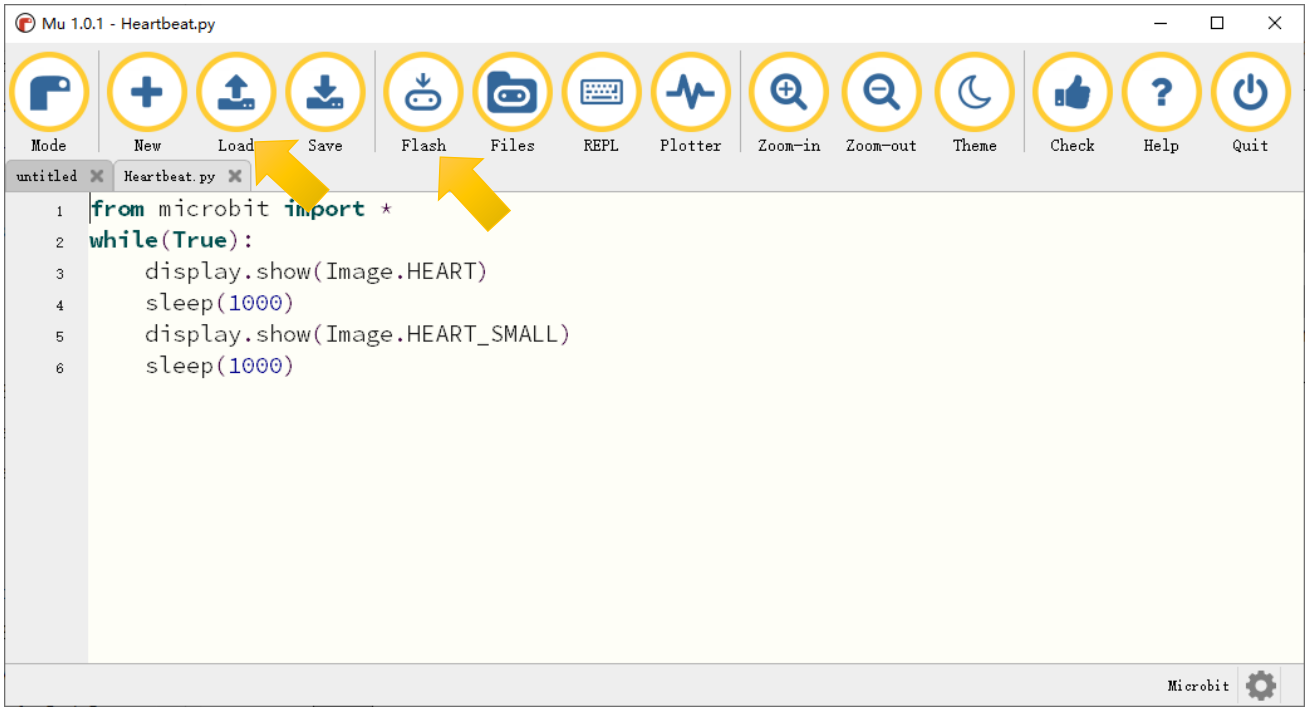


Import the .hex file. The path is as below:

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

Successful loading is shown below.

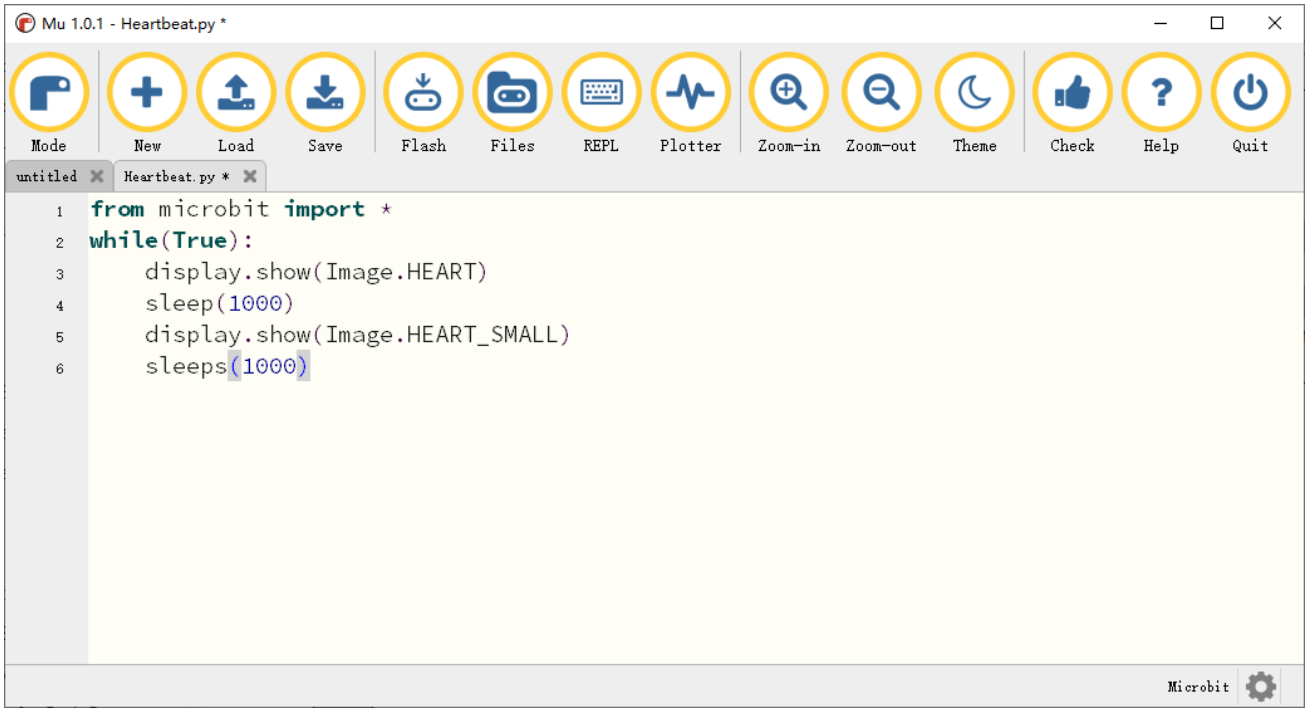
You can also type the code by yourself.



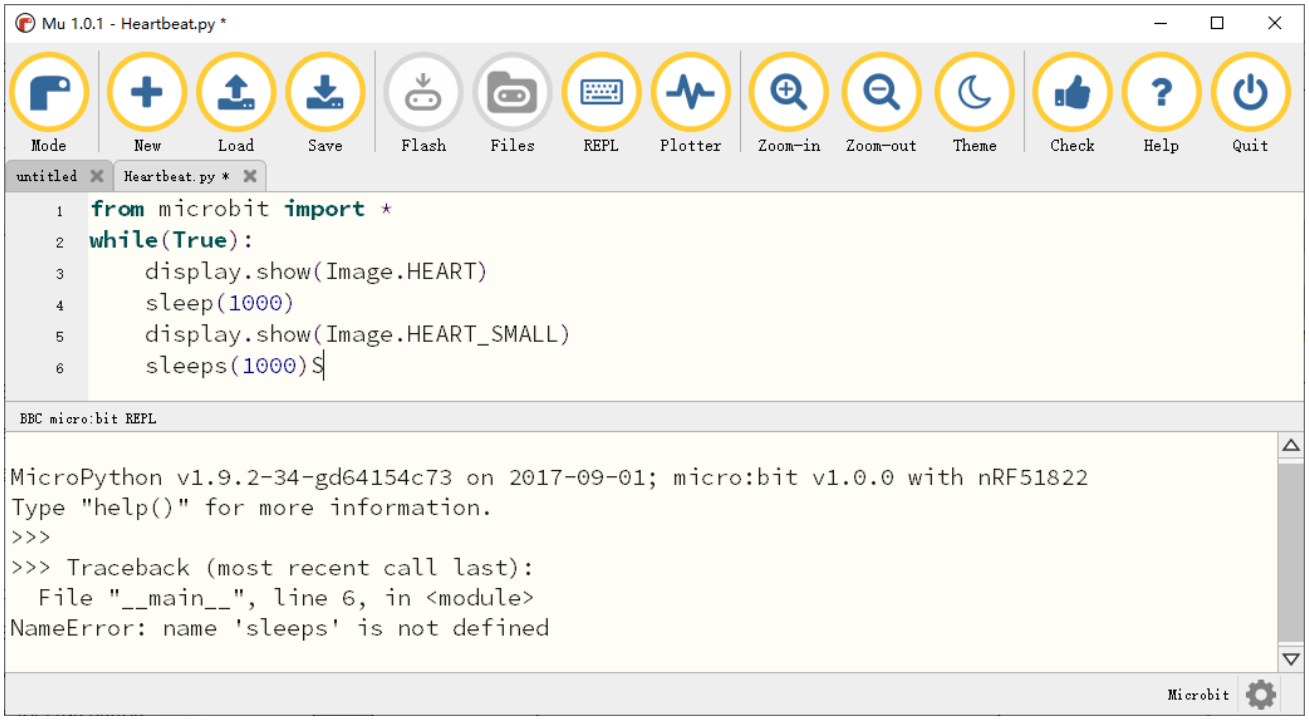
Use the micro USB cable to connect micro:bit and PC, and click the **Flash** button to download the program into micro:bit.

If there are errors in your code, you may be able to successfully download it to micro:bit, but it will not work properly.

For example, the function sleep() was written as sleeps() in the following illustration. Click the button and the code can be uploaded to Micro:bit successfully. However, after the downloading completes, LED matrix prompts some error information and the number of the wrong line.

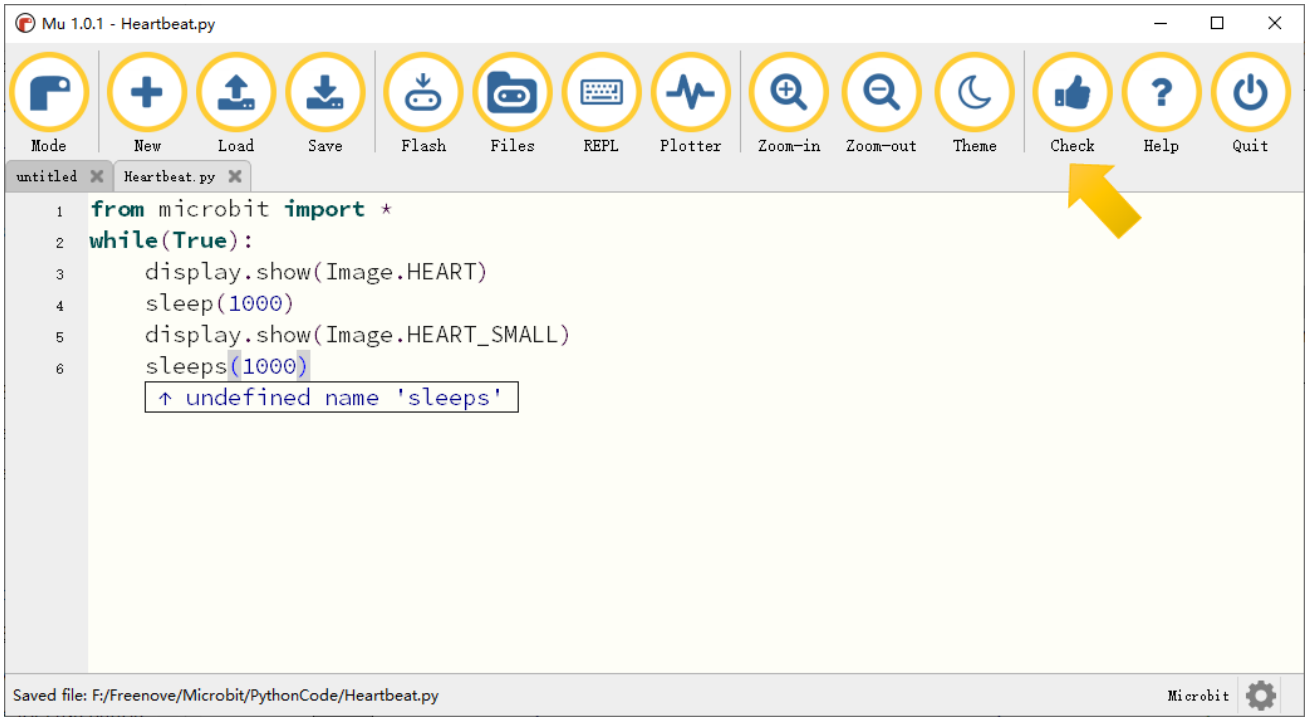


Click the “REPL” button and press the reset button (the button on the back, not A, B) on micro:bit. The error message will be displayed in the REPL box, as shown below:

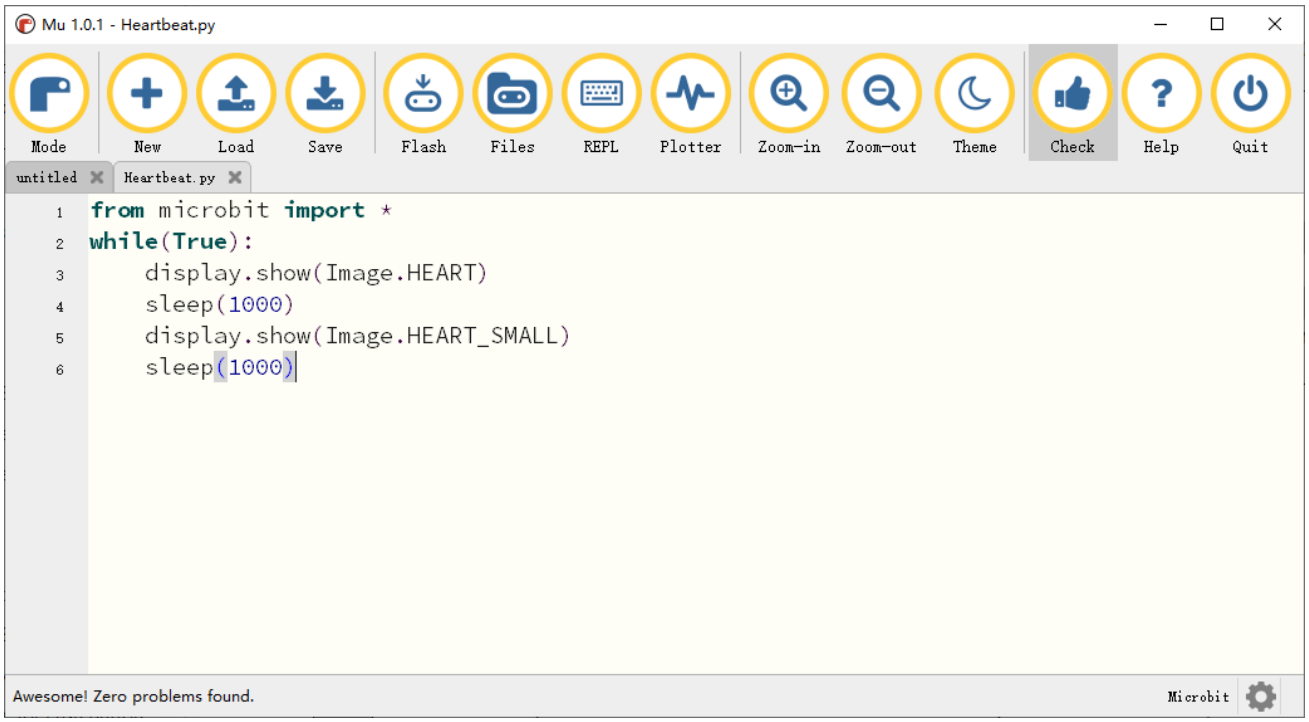


Click REPL again, you will close REPL mode. And then you can flash new code.

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.



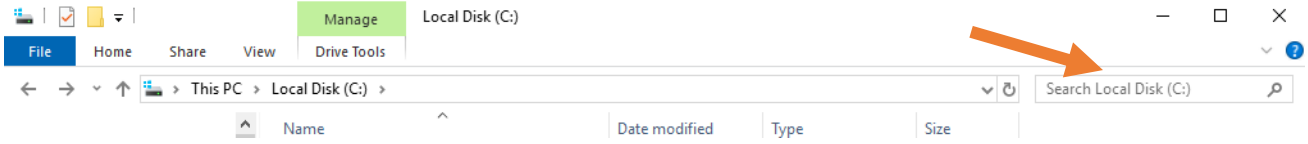
Correct the code according to the error prompt. Then click the "Check" button again, Mu displays no error on the bar below.



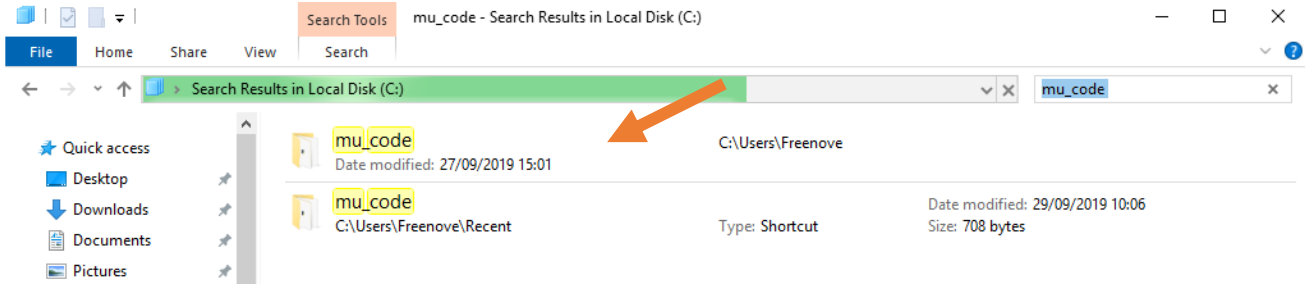
Import necessary Python file into micro:bit

In the code of this tutorial, the LCD1602 module and DHT11 module are used, so 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.



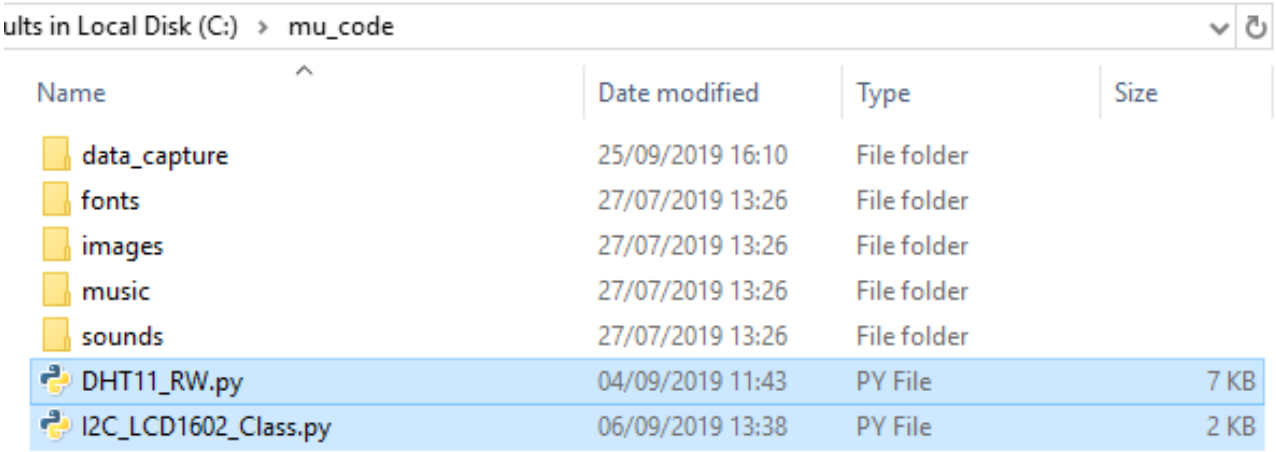
Double click on "mu_code" to enter the folder.



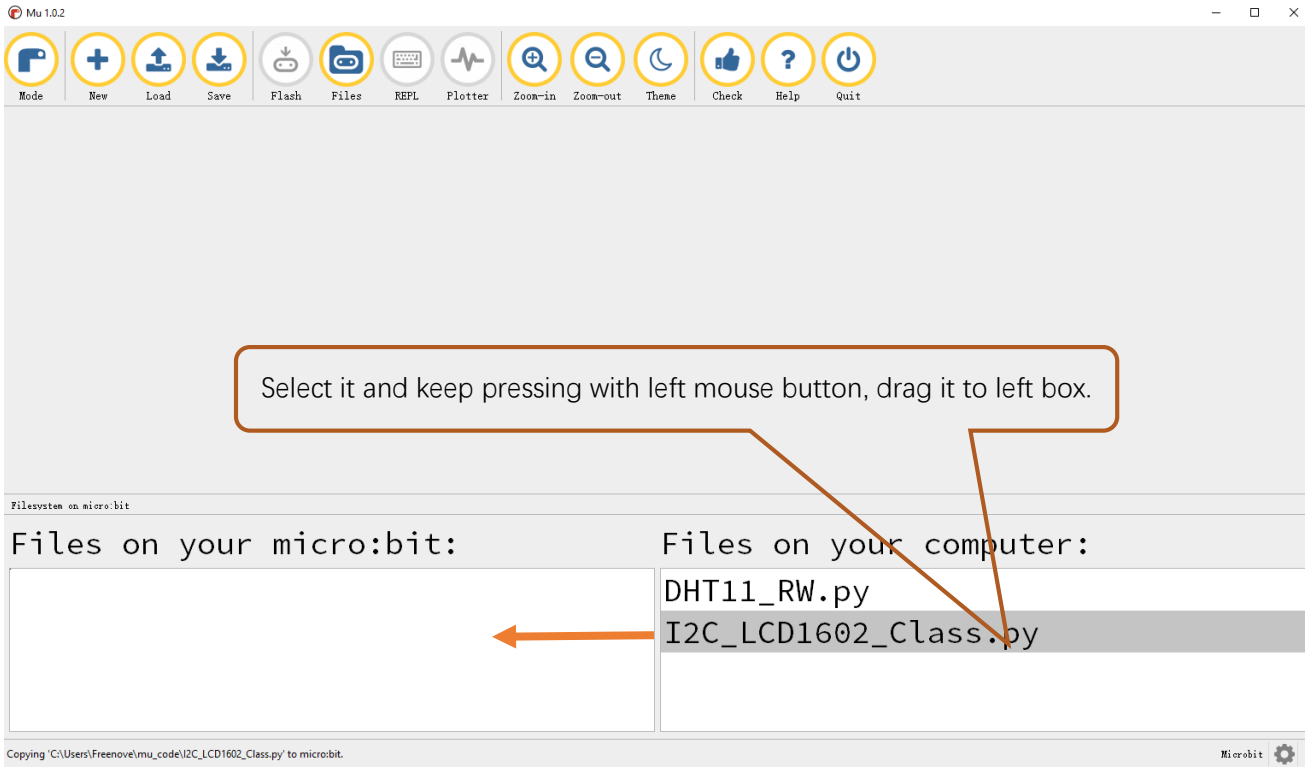
Copy "I2C_LCD1602_Class.py" and "DHT11_RW.py" from following path into "mu_code" directory.

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

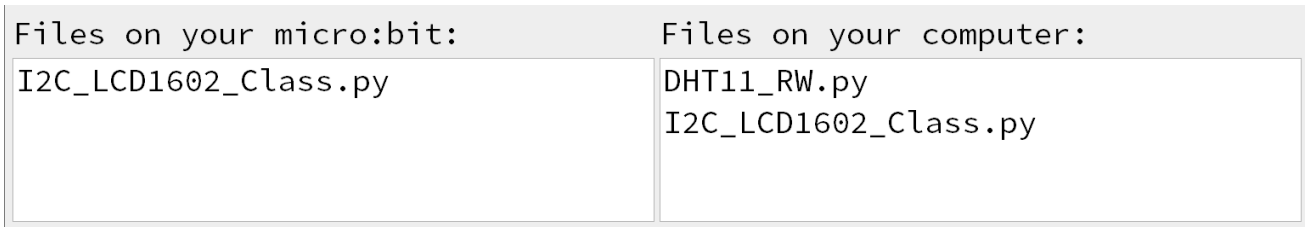
After pasting successfully, you can see them as below:



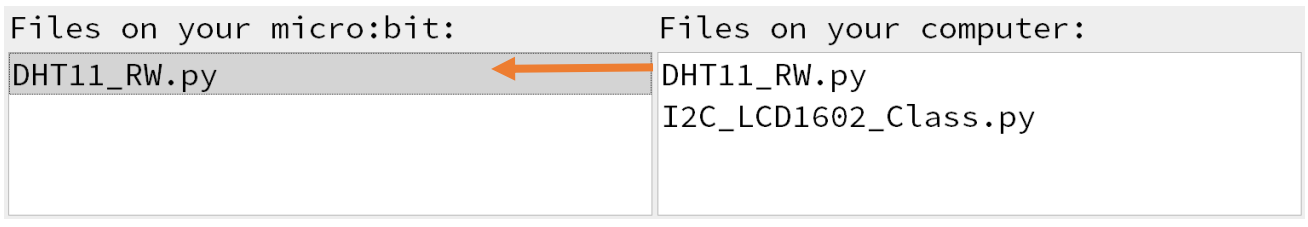
Open the Mu software, click "Files". Here we take "I2C_LCD1602_Class.py" as an example, drag "I2C_LCD1602_Class.py" into micro:bit.



After importing successfully, you will see it on the left.



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



Note, after you upload other file into micro:bit, the original content will be covered. You need to import it next time you use it.

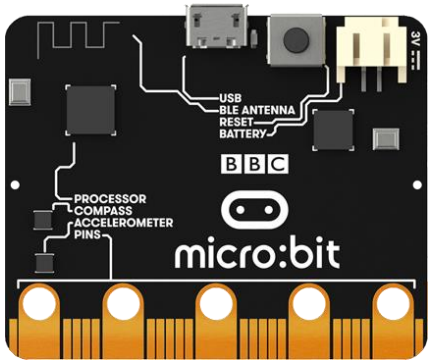
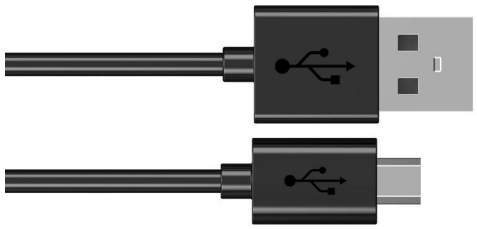
Chapter 1 LED matrix

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.

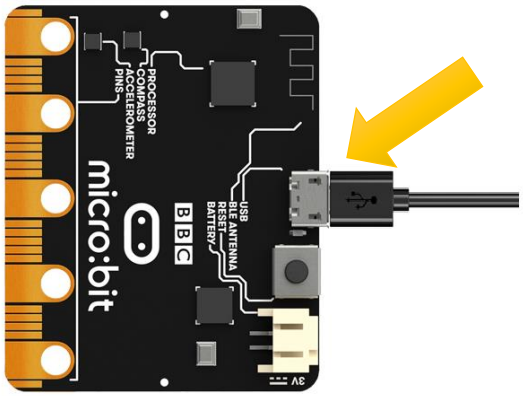
Component List

micro:bit x1 	micro USB cable x1 

Circuit

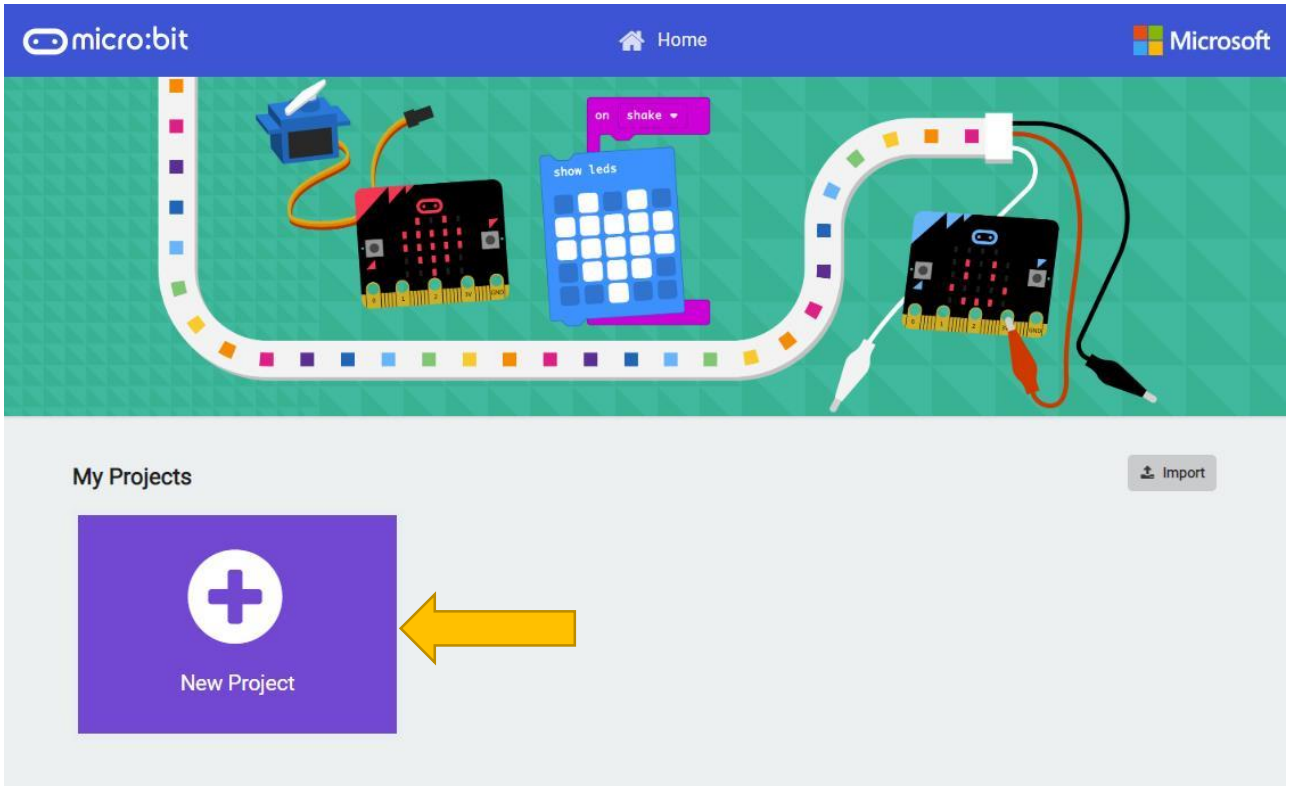
Connect micro:bit and PC via a micro USB cable.

Hardware connection

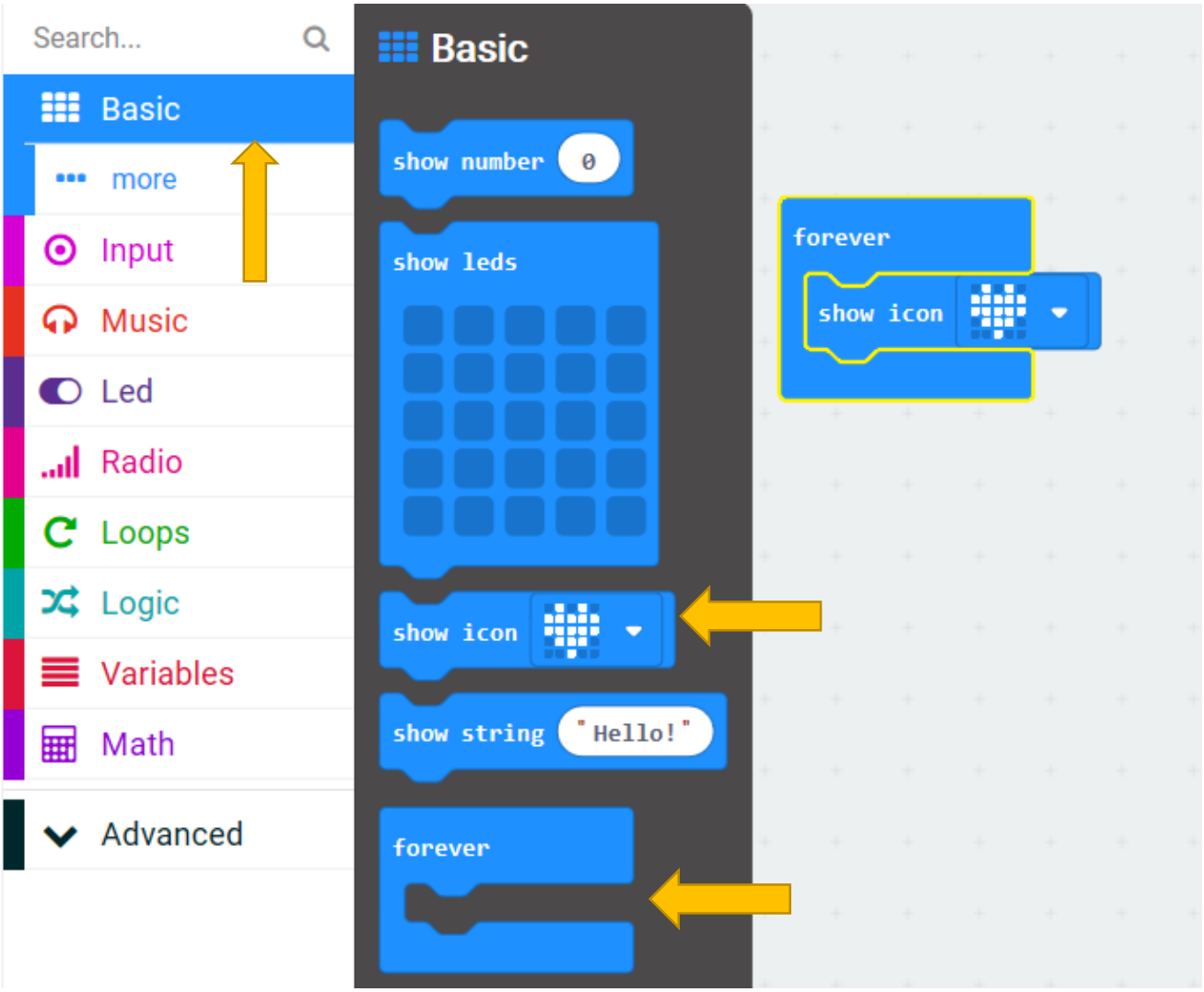


Block code

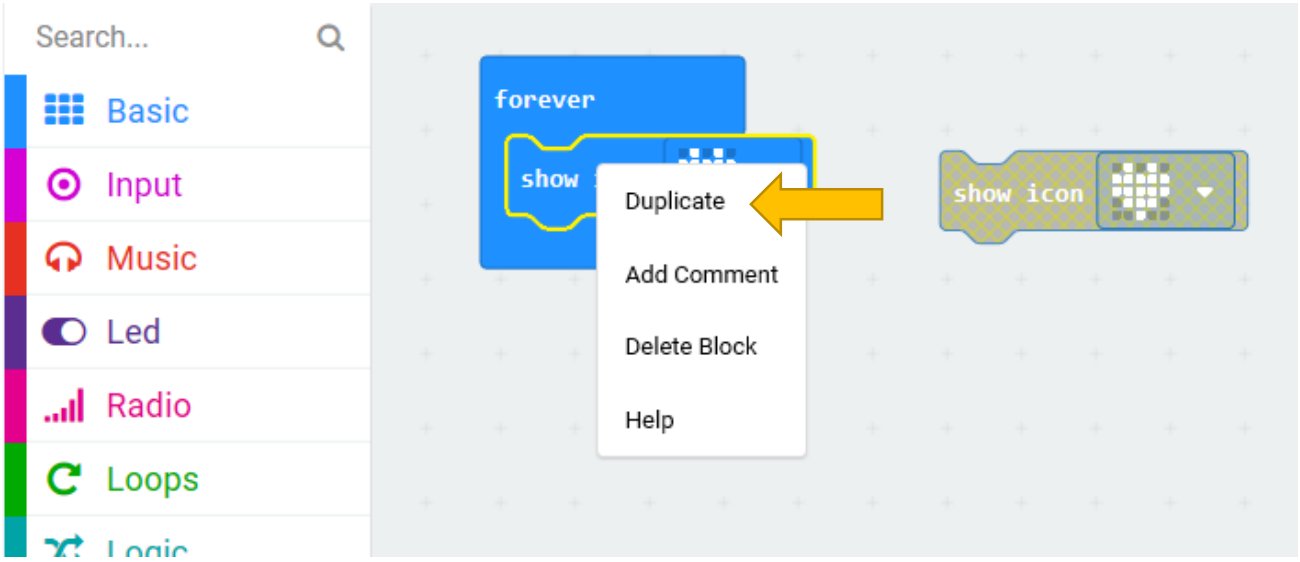
Open the MakeCode for the web version or MakeCode for the win10 version. Click on "New Project"



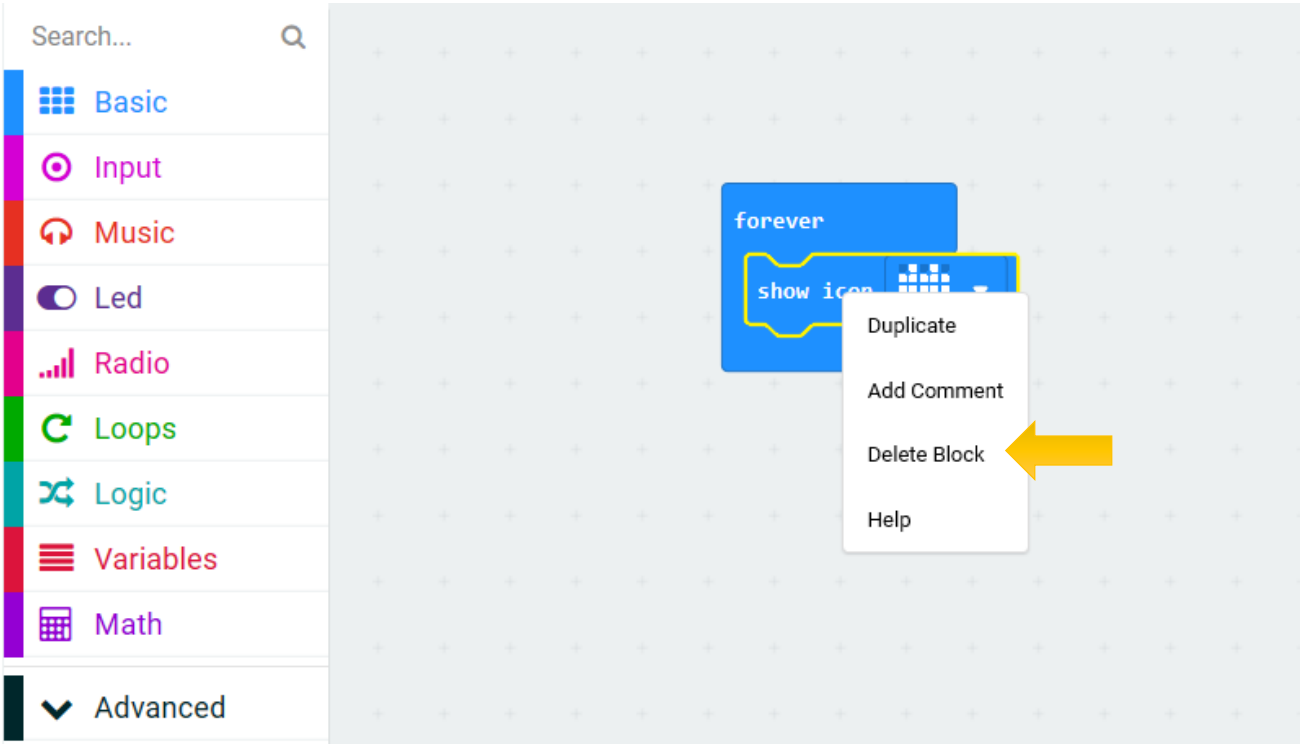
Click **basic** in the list on the left, select the desired code block, and drag it into the right code editing area.



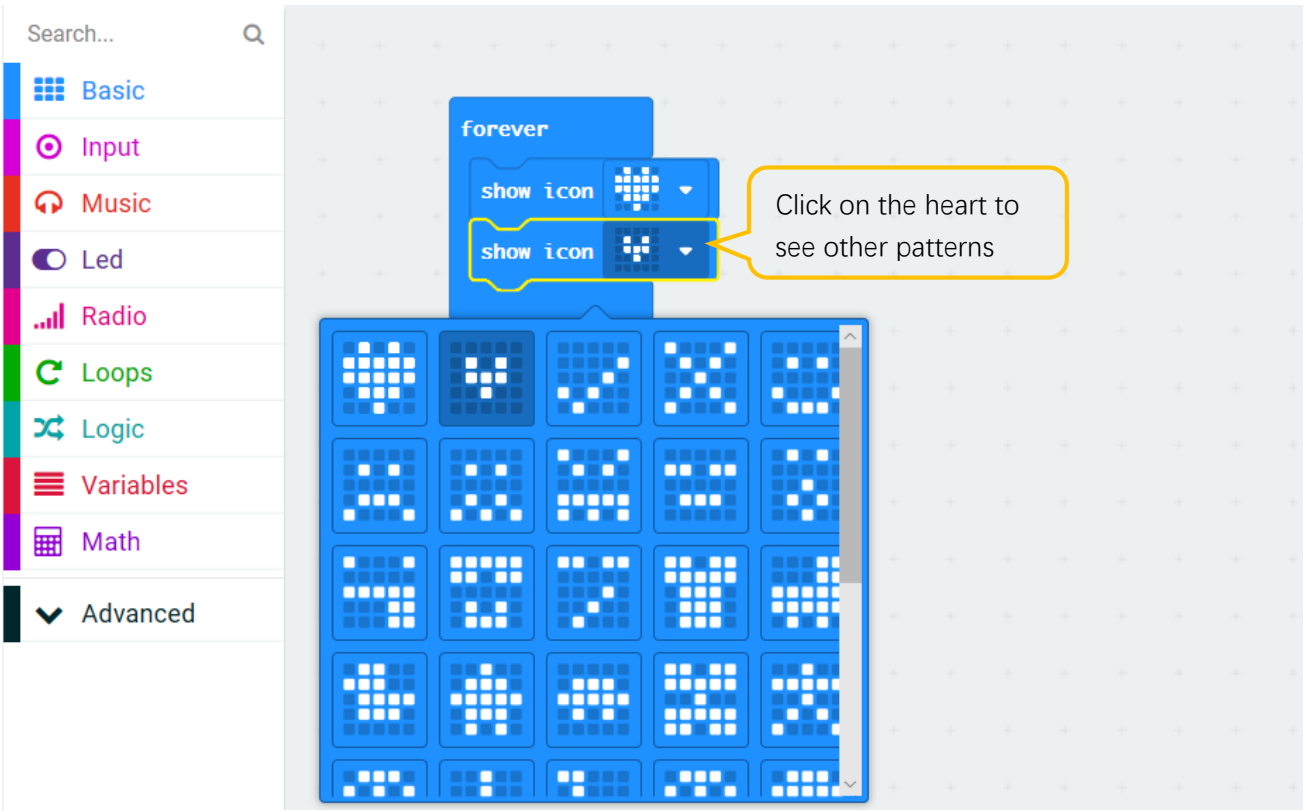
Right click the mouse and select Duplicate to duplicate the code block.



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.**

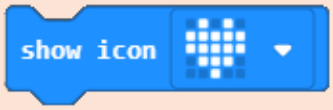


On the second block, click on the drop-down triangle next to the heart-shaped pattern on the block to display all the optional built-in patterns, select the second pattern, a small heart shape.



This completes the block code thie 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 heartbeating.
If you did not master downloading, please refer to contents (How to download? How to quick download?).

Reference

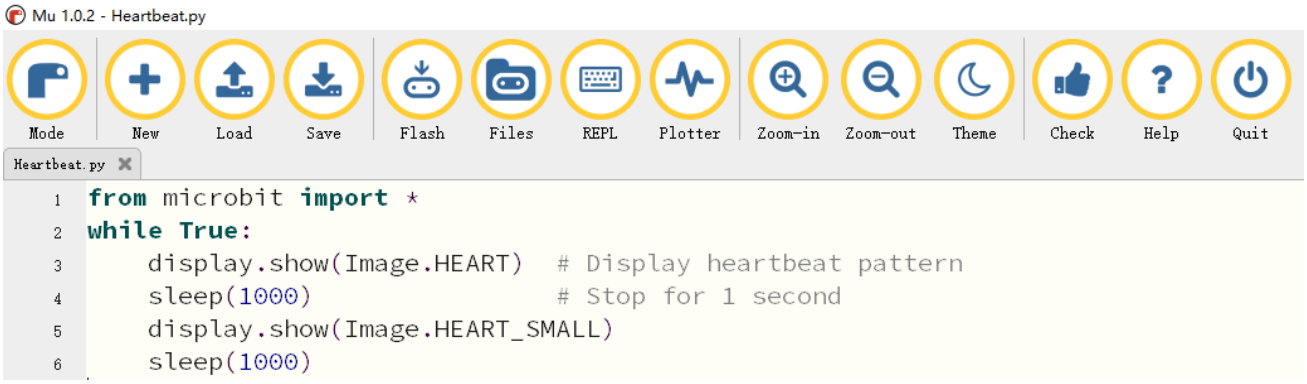
Block	Function
	Shows the selected icon on the LED screen

Python Code

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

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

After loading successfully, the code is shown below:



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

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 infinite 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, etc. 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:
```

An infinite loop that will be executed circularly by microbit constantly.

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

Display heart pattern on LED matrix.

```
sleep(1000)
```

Delay for 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, and then display the small heart pattern for another second.

Reference

from microbit import *
Import everything in the microbit module, including functions, classes, variables, etc. You can use all the available contents 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 circularly.
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

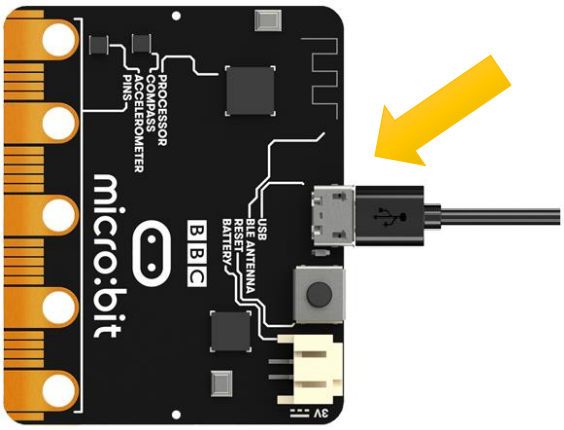
Project 1.2 Displaying Number

In this project, we will use the LED matrix of the micro:bit to display numbers.

Circuit

Connect micro:bit and PC via a micro USB cable.

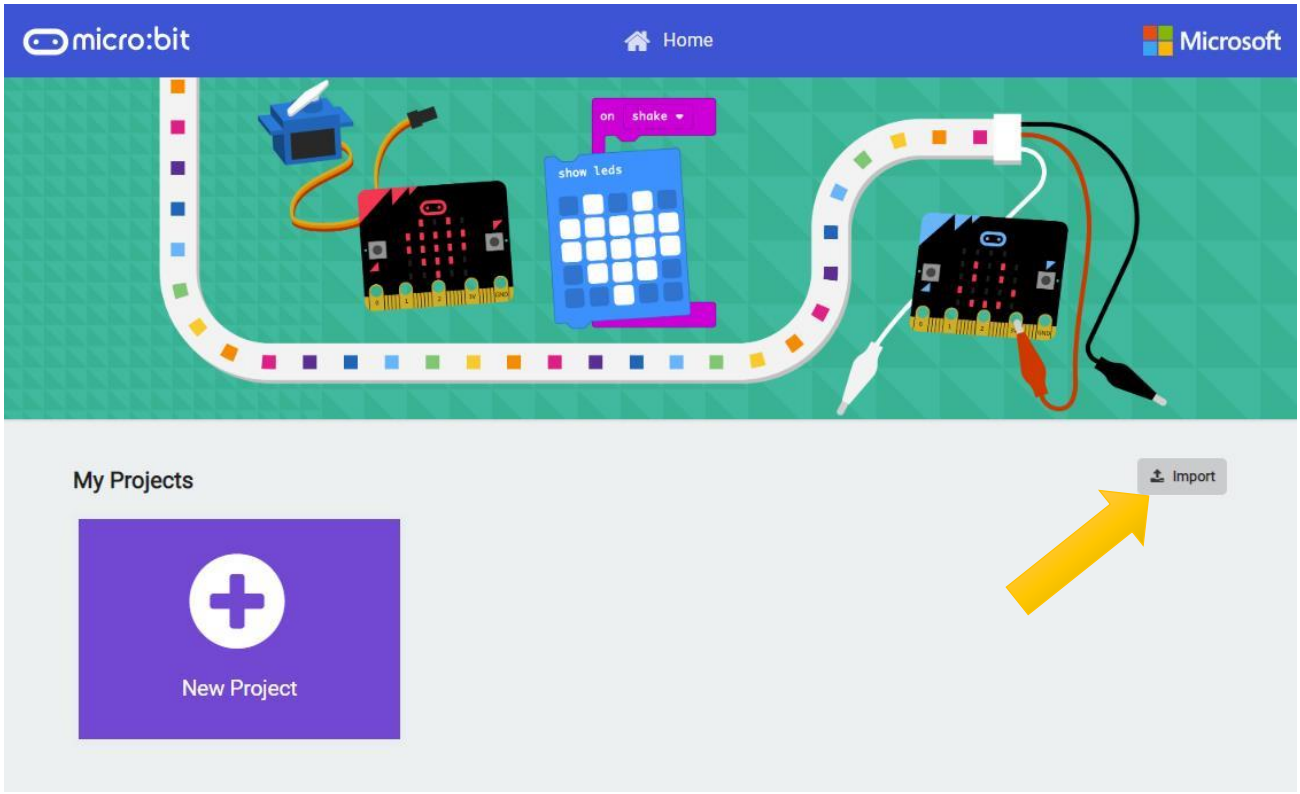
Hardware connection



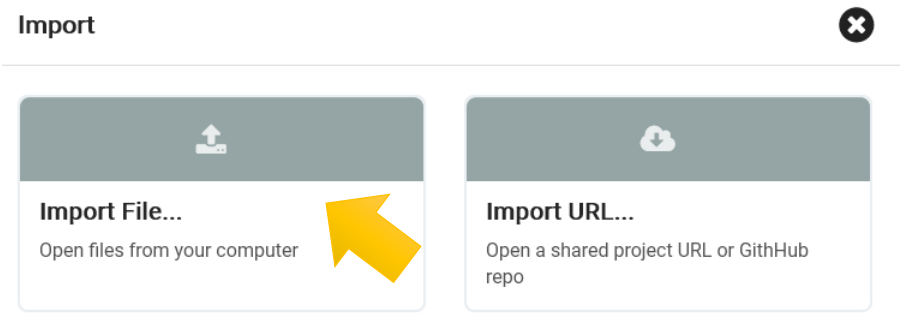
Block code

Open MakeCode first.

In this project, we will import the block code.



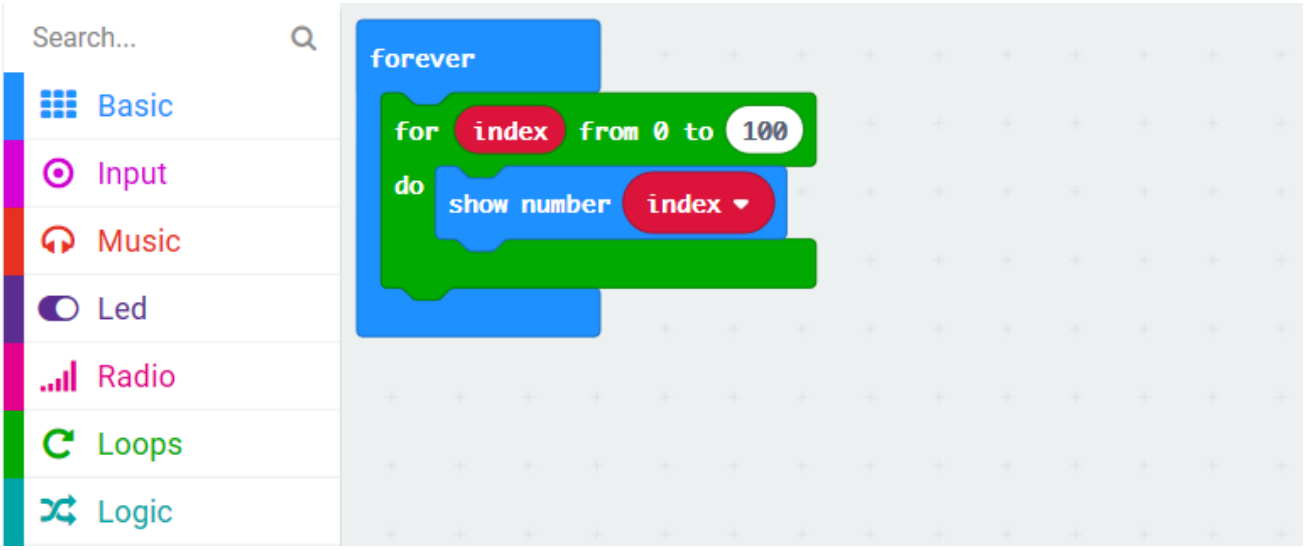
Click **Import**. Then click **Import File**.



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

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

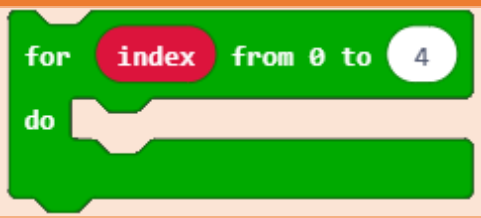
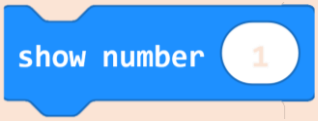
After load successfully, the code is shown as below:



Download the code into the micro:bit. After the downloading completes, the micro:bit LED matrix will start to display the numbers 0, 1, 2, 3, 4...99. Then start again from 0 to 99, 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 99, the for loop is exited. In the body of the loop, the value of the numeric index is displayed.

Reference

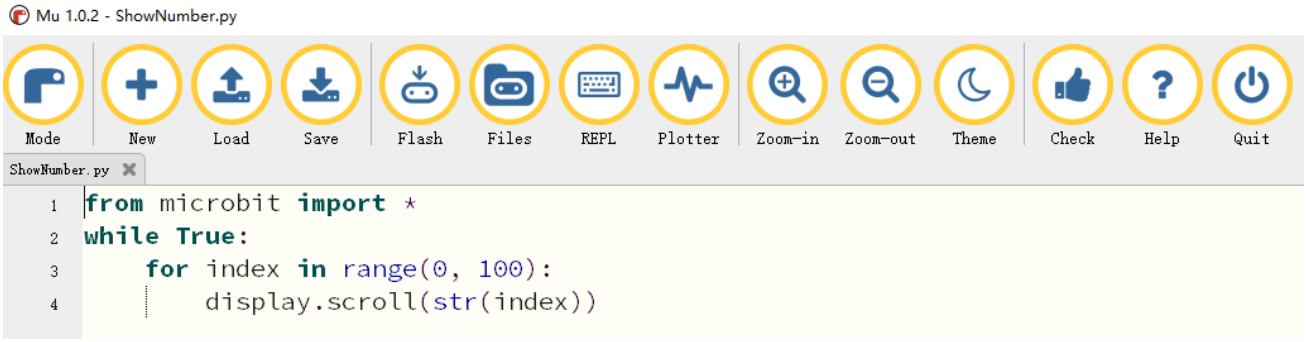
Block	Function
	This is a for loop, the number (4) of loops can be changed, each time the index is incremented by 1. The loop won't end until the index is greater than the set value.
	Show a number on the LED screen. It will slide left if the number is more than one digit..

Python code

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

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

After loading successfully, the code is shown as below:



Download the code into the microbit. After the downloading completes, 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 repeat endlessly.

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

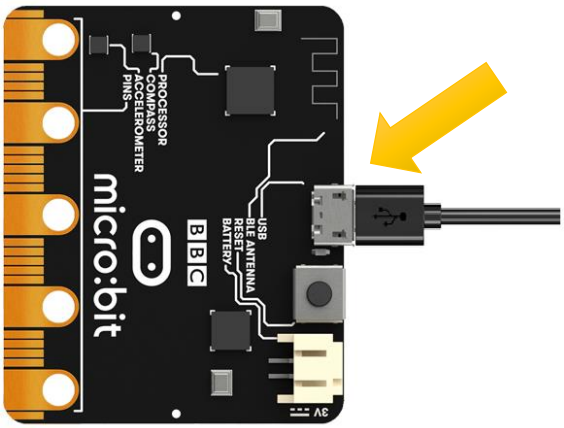
Project 1.3 Displaying Text

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 loading 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!"

Reference

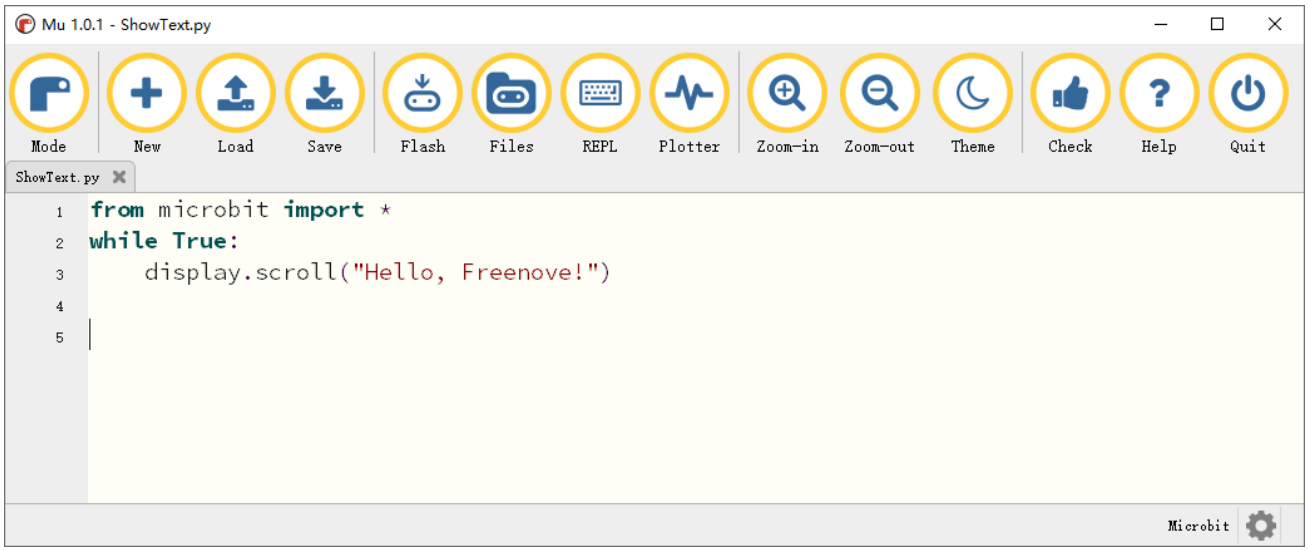
Block	Function
show string Hello	Displays a string on the LED screen. It will scroll to left if it's beyond the screen.

Python code

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

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

After loading 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.

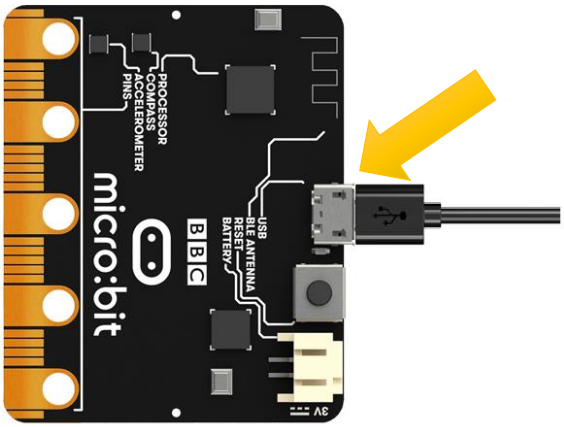
Project 1.4 Displaying Custom

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

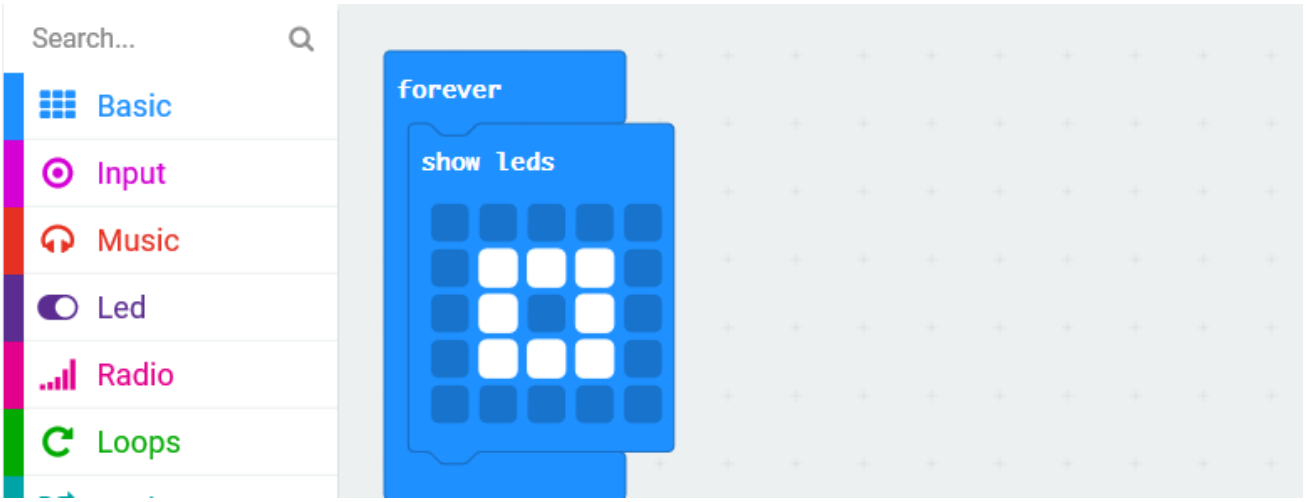
Open MakeCode first.

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

(How to import project)

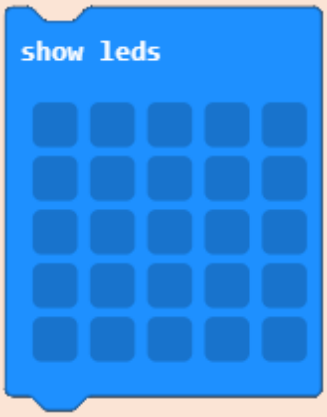
File type	Path	File name
HEX file	../Projects/BlockCode/01.4_ShowCustom	ShowCustom.hex

After loading successfully, the code is shown as below:



Check the connection of the circuit and verify it correct.. Then download the code into the microbit, and the square pattern shown above will appear on the LED matrix of the micro:bit.

Reference

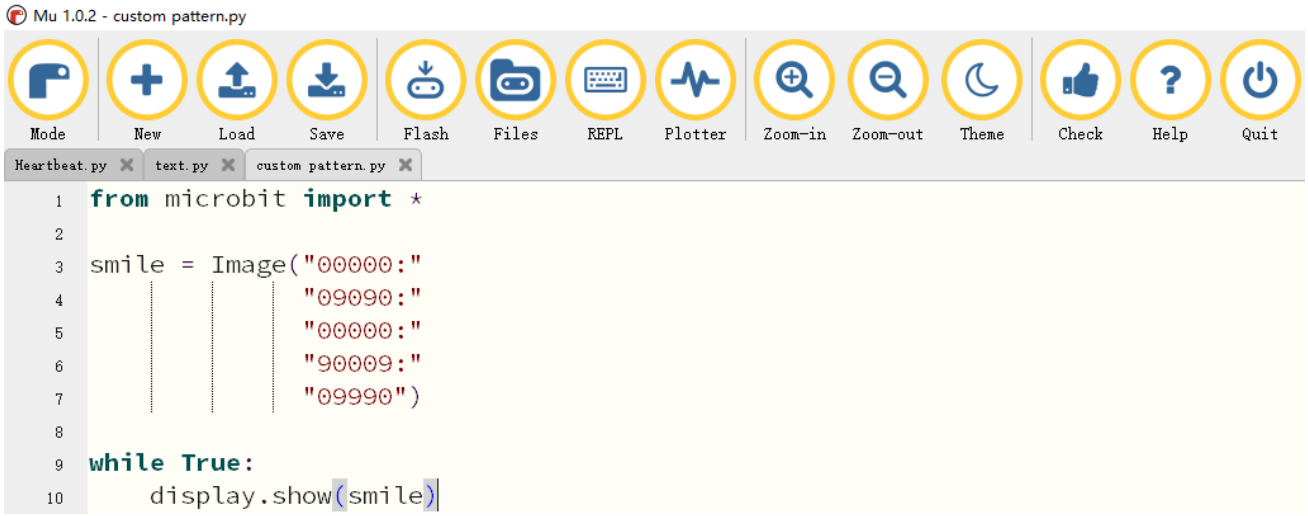
Block	Function
	Shows a picture on the LED screen.

Python code

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

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

After loading successfully, the code is shown as below:



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:

```
from microbit import *

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

while True:
    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 represents the brightness of an LED. The value ranges from 0 to 9, the larger the number, the brighter the LED.

```
img = Image("00000:"
            "09990:"
            "09090:"
            "09990:"
            "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, etc. Two programmable buttons A and B are integrated on the micro:bit to easily control the micro:bit to make actions.

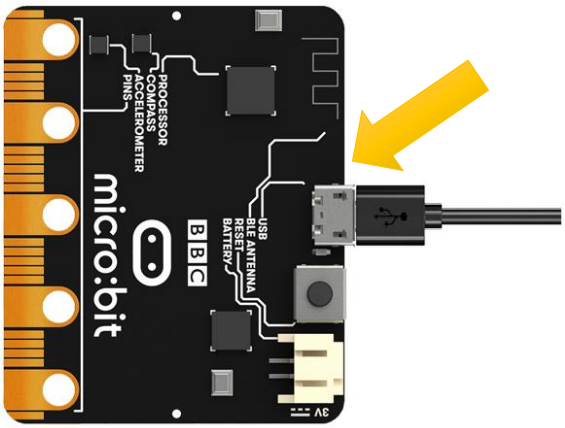
Project 2.1 Button A and B

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

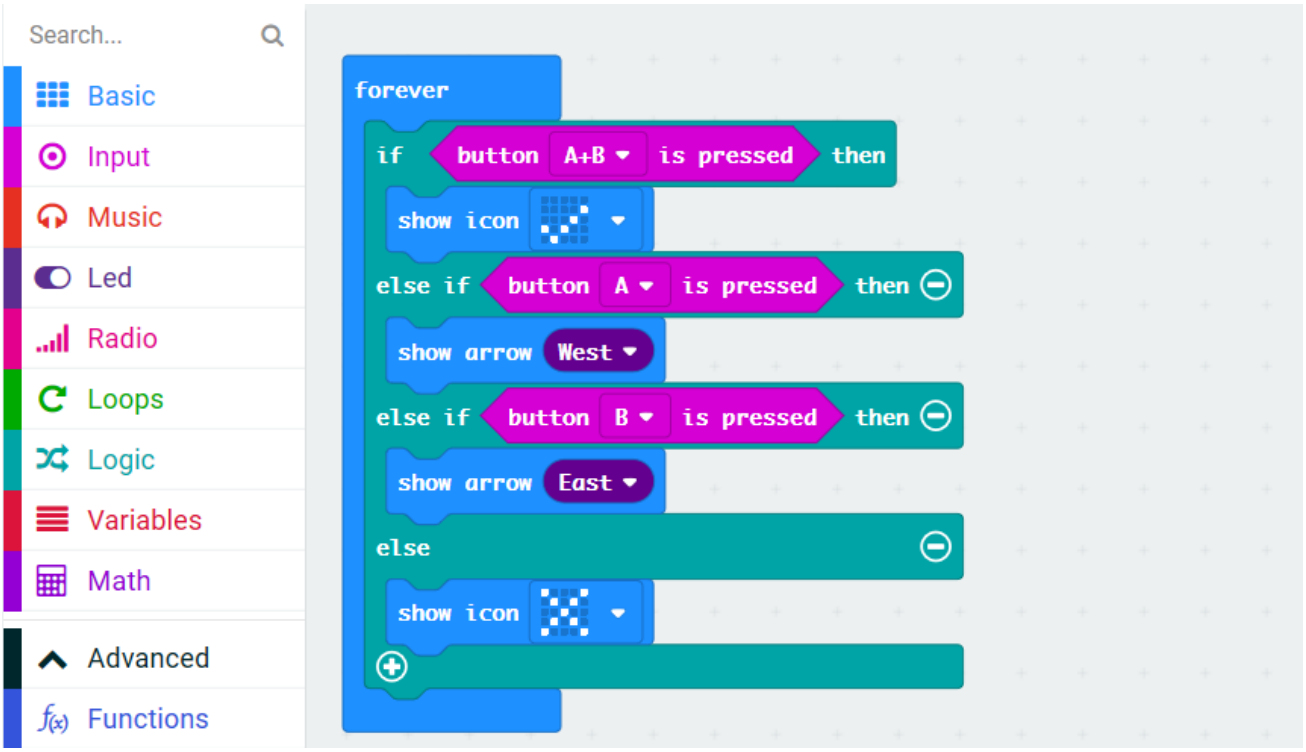
Open MakeCode first.

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 loading 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 an arrow pointing to 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

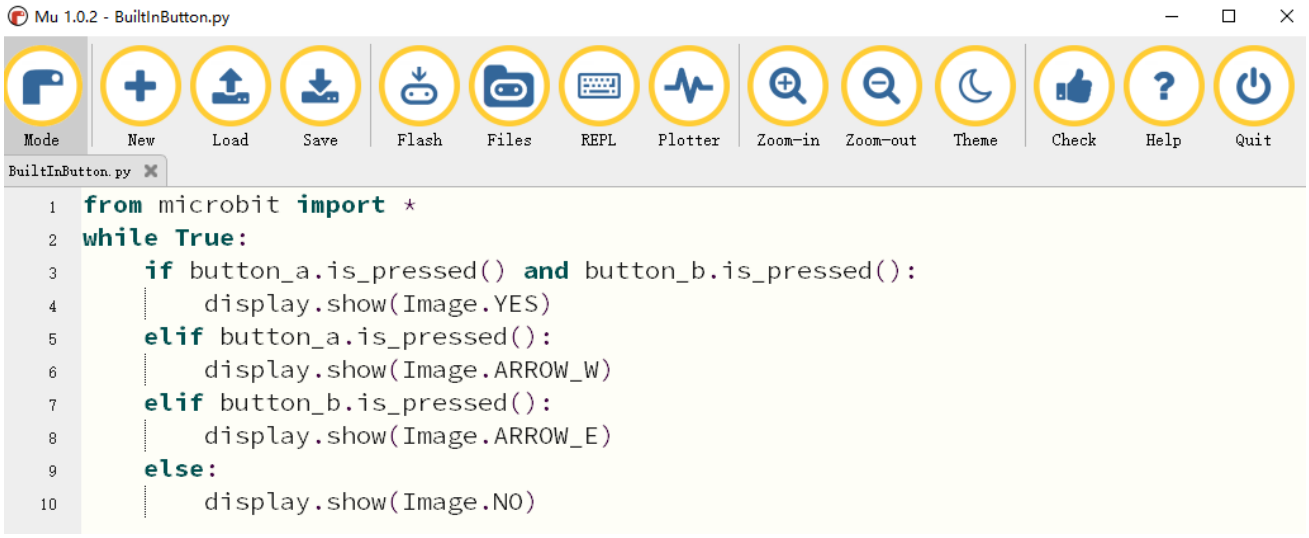
Block	Function
button A is pressed	Check whether a button is pressed at the moment. The micro:bit has two buttons: button A and button B.
on button A pressed	This handler works when button A or B is pressed, or A and B together.

Python code

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

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

After loading 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 an arrow pointing to 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 pressed seperately, 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 buttoon pressed is placed in last, the result of pressing A or B will appear first, then the statement will end, and then sentence met the state with two button pressed will never be executed.

Reference

is_pressed()

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

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

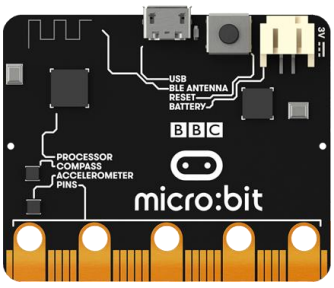
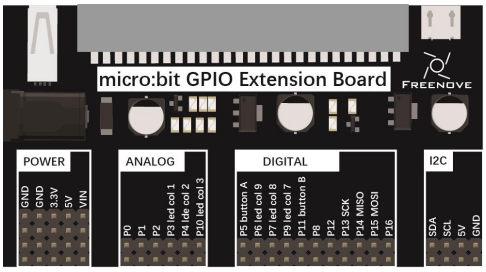
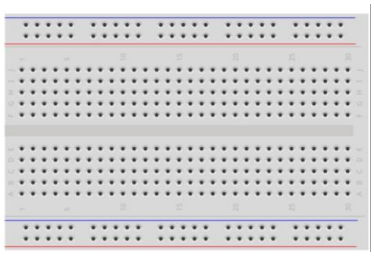
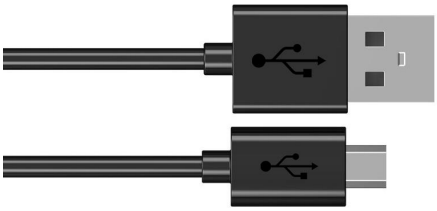
Chapter 3 LED

This section we will learn how to control external LEDs.

Project 3.1 Blink

This project uses micro:bit to control LED blinking.

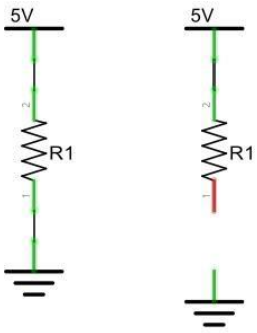
Component List

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

Circuit Knowledge

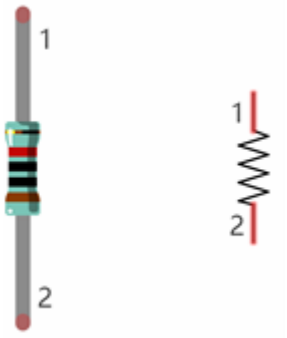
Current

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

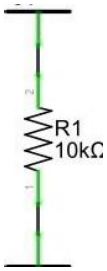


Resistor

Resistors use Ohms (Ω) as the unit of measurement of their resistance (R). 1MΩ=1000kΩ, 1kΩ=1000Ω.
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit.
On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



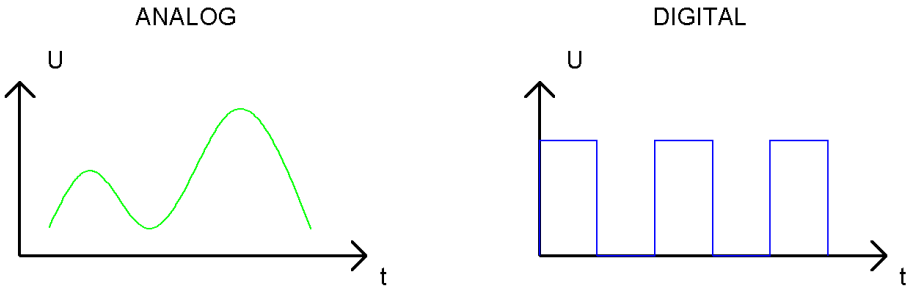
The bands of color on a resistor is a shorthand code used to identify its resistance value. For more details of resistor color codes, please refer to the card in the kit package.
With a fixed voltage, there will be less current output with greater resistance added to the circuit. The relationship between Current, Voltage and Resistance can be expressed by this formula: I=V/R known as Ohm's Law where I = Current, V = Voltage and R = Resistance. Knowing the values of any two of these allows you to solve the value of the third.
In the following diagram, the current through R1 is: I=U/R=5V/10kΩ=0.0005A=0.5mA.



WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.
Note: Unlike LEDs and Diodes, Resistors have no poles and re non-polar (it does not matter which direction you insert them into a circuit, it will work the same)

Analog signal and Digital signal

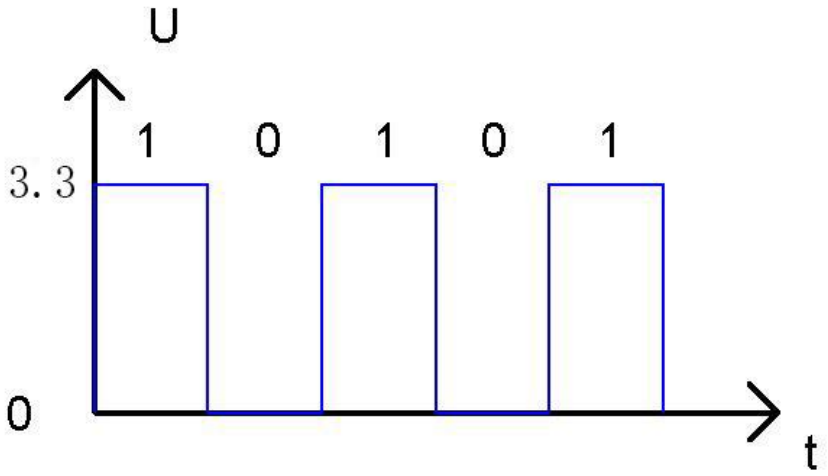
An Analog Signal is a continuous signal in both time and value. On the contrary, a Digital Signal or discrete-time signal is a time series consisting of a sequence of quantities. Most signals in life are analog signals. A familiar example of an Analog Signal would be how the temperature throughout the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



Note that the Analog signals are curved waves and the Digital signals are “Square Waves”.
In practical applications, we often use binary as the digital signal, that is a series of 0’s and 1’s. Since a binary signal only has two values (0 or 1) it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

Low level and high level

In circuit, the form of binary (0 and 1) is presented 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, low-power components 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, which is designed to connect the components together by inserting its two terminals.

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

Jumper M/M

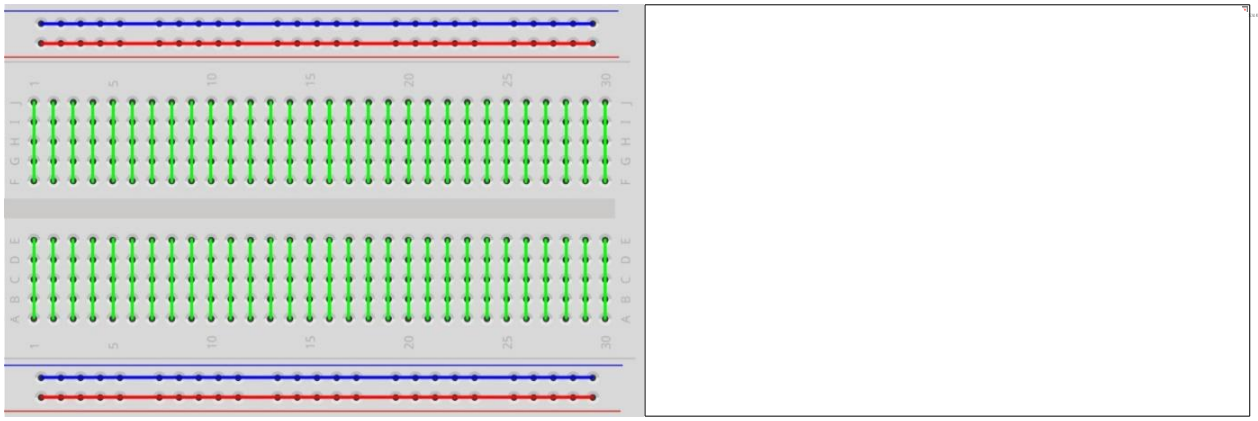
Jumper F/F

Jumper F/M

Breadboard

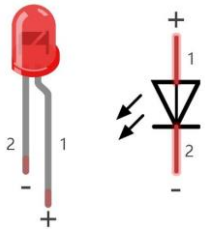
There are many small holes on breadboard to connect Jumper.

Some small holes are connected inside breadboard. Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached. The left picture shows the ways the pins have shared electrical connection and the right picture shows the actual internal metal, which connect these rows electrically.



LED

An LED is a type of diode. All diodes only work if current is flowing in the correct direction and have two Poles. An LED will only work (light up) if the longer pin (+) of LED is connected to the positive output from a power source and the shorter pin is connected to the negative (-) negative output also referred to as Ground (GND). This type of component is known as “Polar” (think One-Way Street).
All common 2 lead diodes are the same in this respect. Diodes work only if the voltage of its positive electrode is higher than its negative electrode and there is a narrow range of operating voltage for most all common diodes of 1.9 and 3.4V. If you use much more than 3.3V the LED will be damaged and burn out.



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

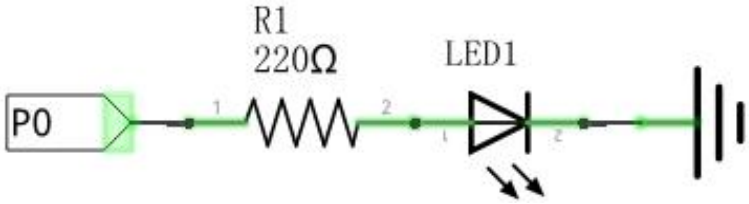
Circuit

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

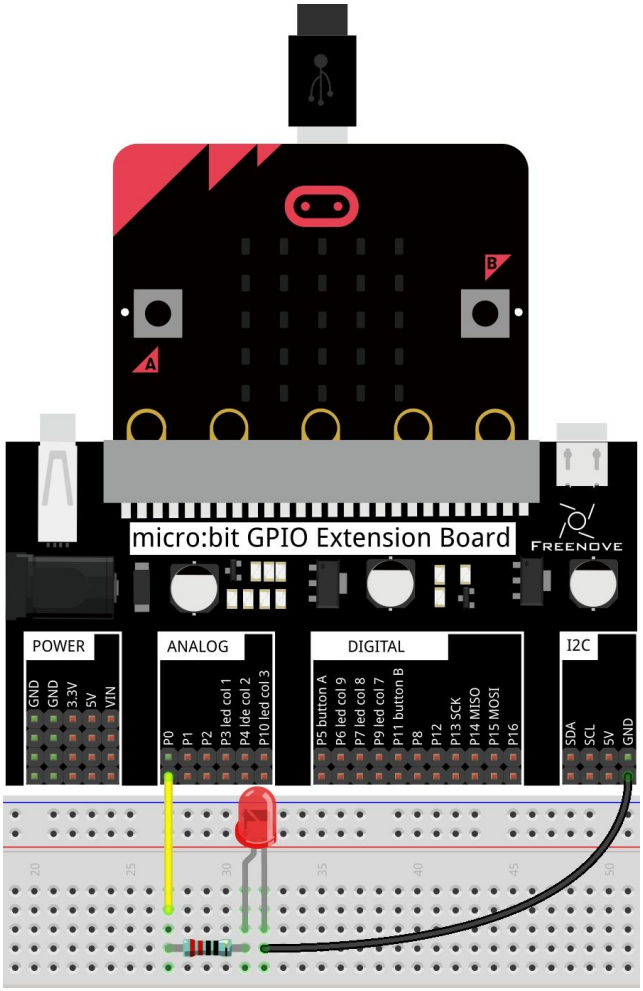
LED’s positive pole (long pin) should be connected to resistor while its negative pole (short pin) should be connected to GND. After the circuit is built and verified correct, use the USB cable to connect the PC to the micro:bit to power the circuit.

CAUTION: Avoid any possible short circuits (especially connecting 5V or GND, 3.3V and GND)!
WARNING: A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your micro:bit!

Schematic diagram



Hardware connection

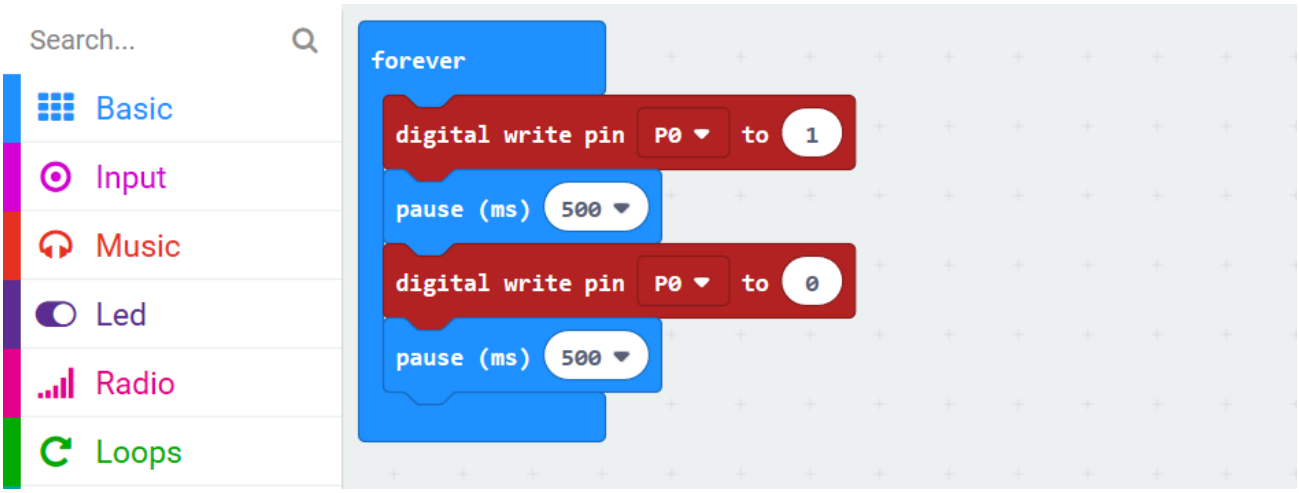


Block code

Open MakeCode first. Import the .hex file. The path is as below:
(How to import project)

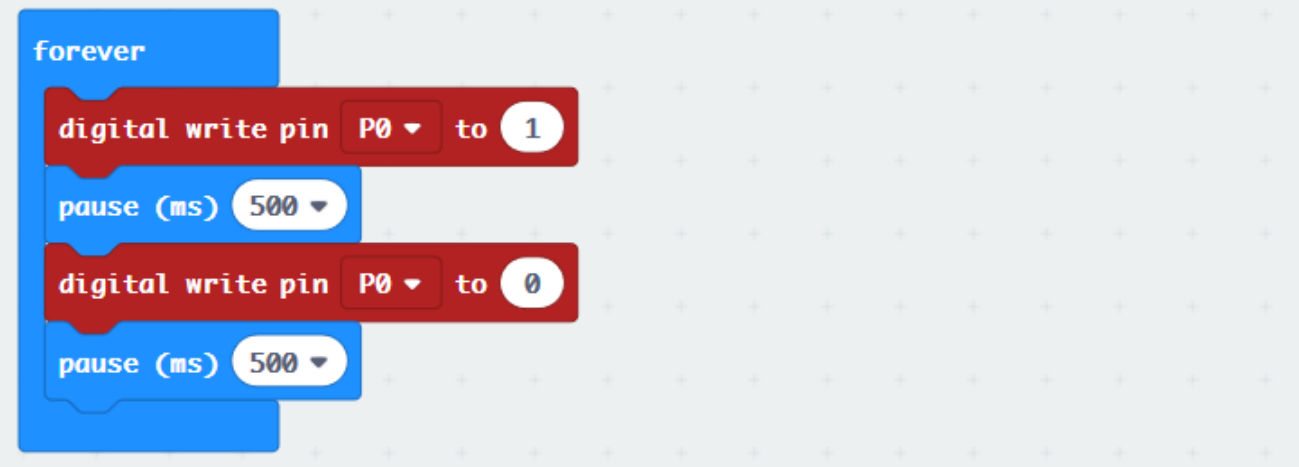
File type	Path	File name
HEX file	../Projects/BlockCode/03.1_Blink	Blink.hex

After import successfully, the code is shown as below:

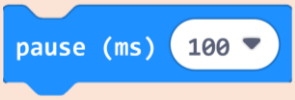


Download the code into the micro:bit and the LED on the breadboard will begin to blink.

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. Repeat the loop, then LED will start blinking.



Reference

Block	Function
	Pause the program for the number of milliseconds you set. 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, the path is as below:

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

After loading successfully, the code is shown as below:

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. Repeat the loop, then LED will start blinking.

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

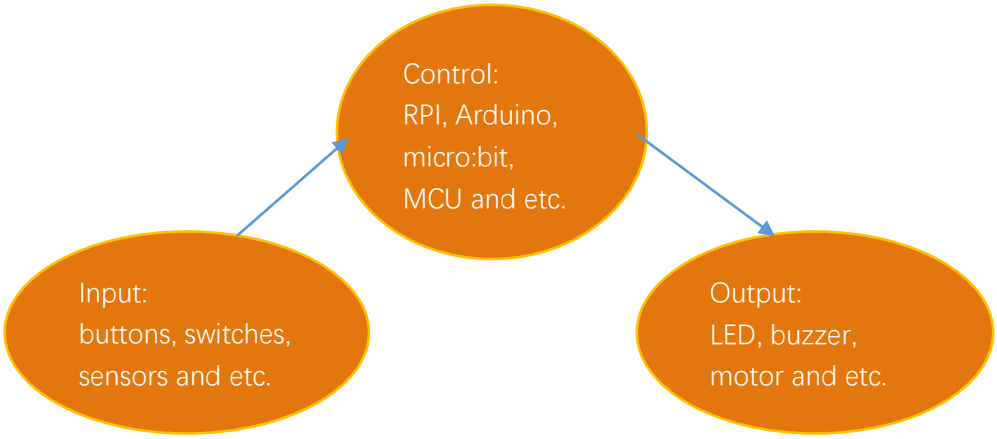
pin.write_digital(value)

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 was the output part and micro:bit was the control part. In practical applications, we not only make the LEDs flash, but also make a device sense the surrounding environment, receive instructions and then take the appropriate action such as turn on LEDs, make a buzzer beep and so on.

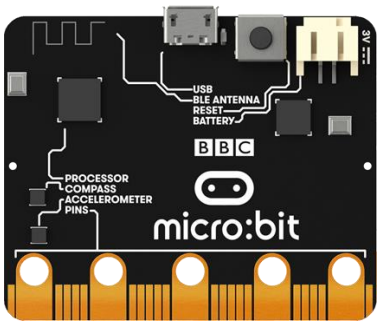
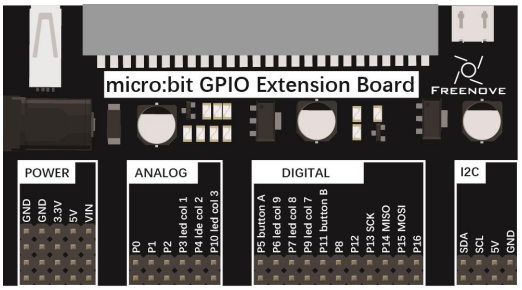
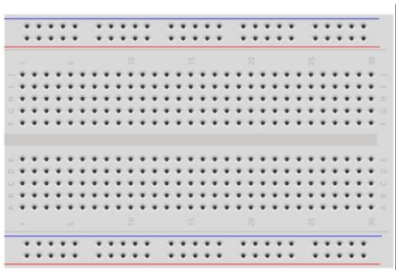
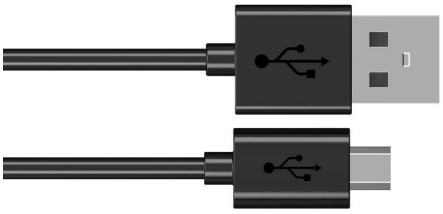


Next, we will build a simple control system to control an LED through a push button switch.

Project 4.1 Control LED by Button

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

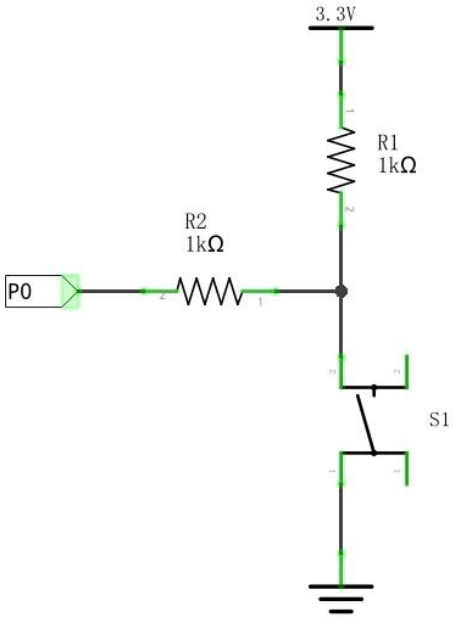
Component list

<table>
<tr><td>Microbit x1
  </td><td colspan="3">Expansion board x1
  </td></tr>
<tr><td>Breakboard x1
  </td><td colspan="3">USB cable x1
  </td></tr>
<tr><td>FM x4 MM x1
  </td><td>Resistor
220Ω x1
 
1kΩ x2
  </td><td>LED x1
  </td><td>Push Button Switch x1
  </td></tr>
</table>

Circuit knowledge

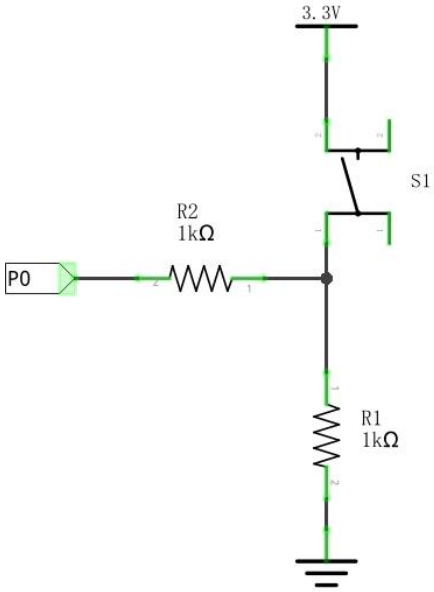
Connection of Push Button Switch

We connect a push button switch directly to the circuit to turn ON or OFF the LED. In digital circuits, we need to use the push button switch as an 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 when the button is pressed, it will be 0V (low level). Resistor R2 here is used to prevent the port from being set to output high level by accident. Without R2, the port maybe connected directly to the cathode and cause a short circuit when the button is pressed.

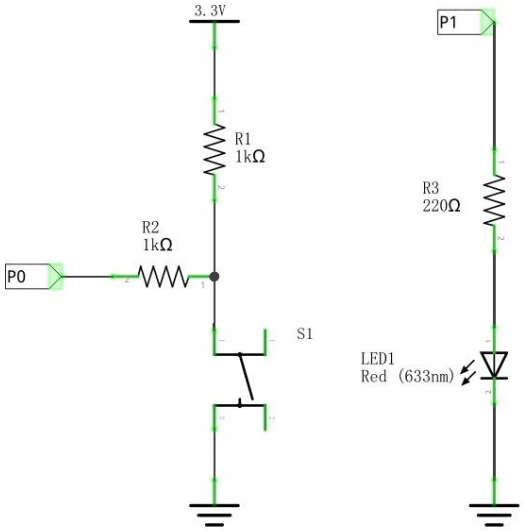
The following diagram shows another connection, in which the level detected by I/O port is opposite to above diagram, when the button is pressed or not.



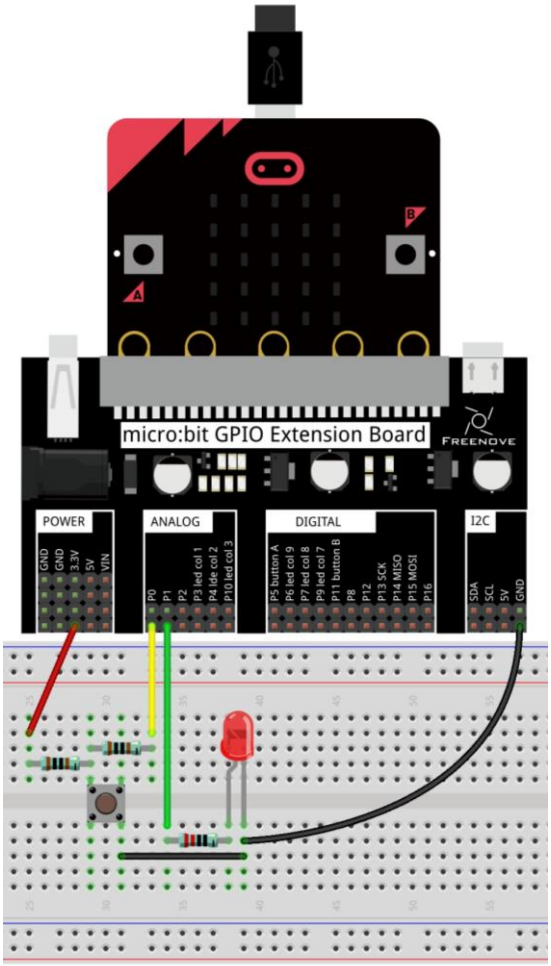
Circuit

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

Schematic diagram



Hardware connection

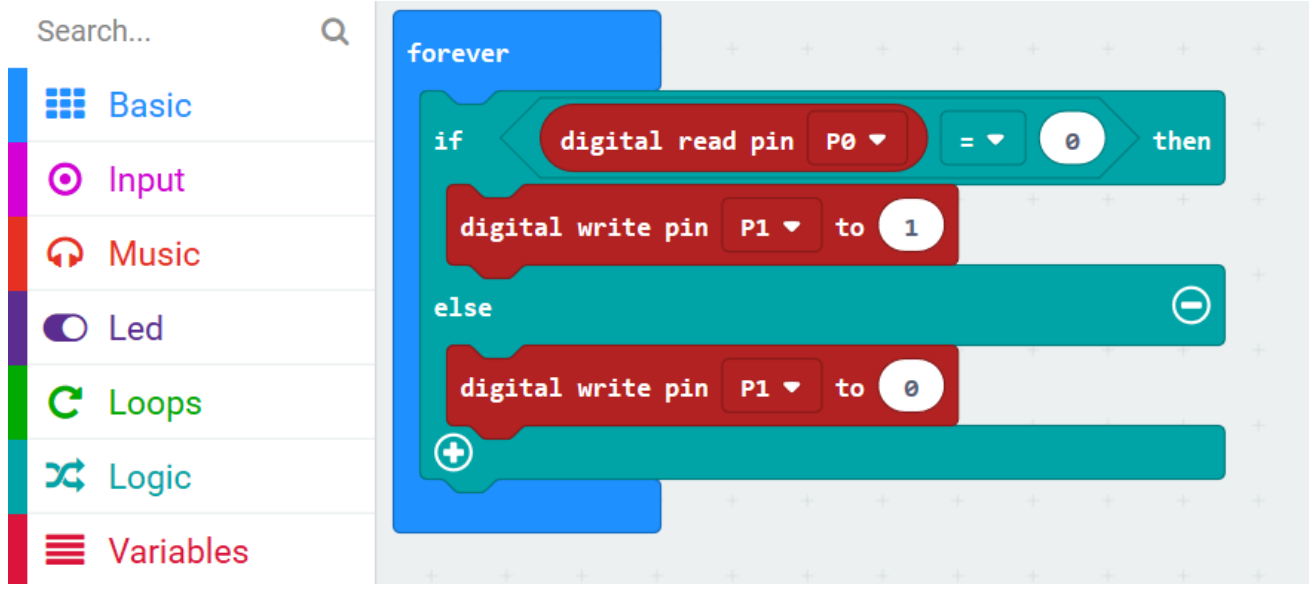


Block code

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

Otherwise, P1 pin outputs 0, and the LED will be turned OFF.

Reference

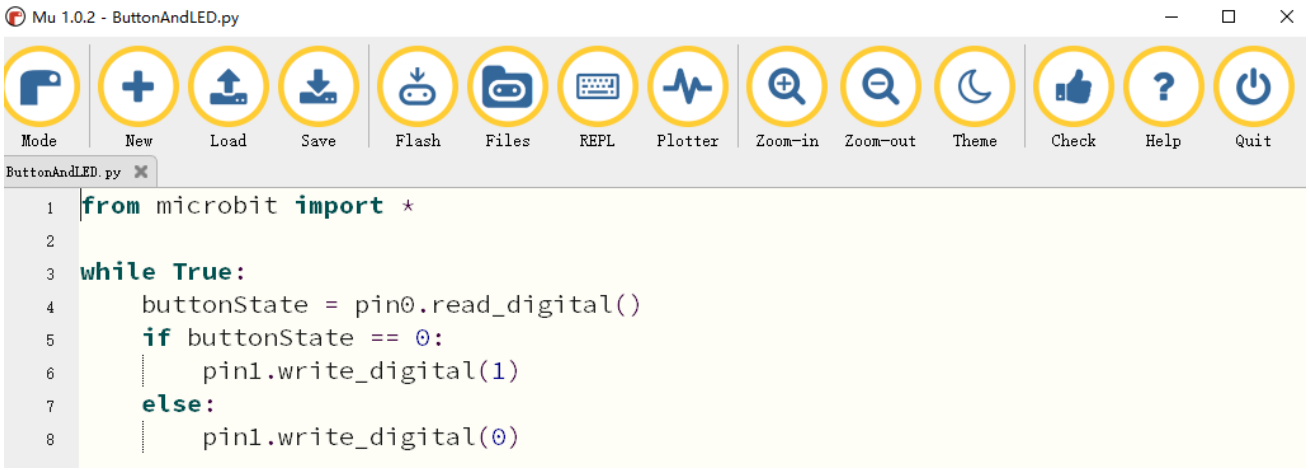
Block	Function
digital read pin P8	Read a digital (0 or 1) signal from a pin on the micro:bit board.

Python code

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

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

After loading 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.
(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, so 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 level, 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 this will function differently: Press the button, the LED will turn ON, and pressing the button again, the LED turns OFF. The ON switch action is no longer momentary (like a door bell) but remains ON without needing to continually press on the Button Switch.

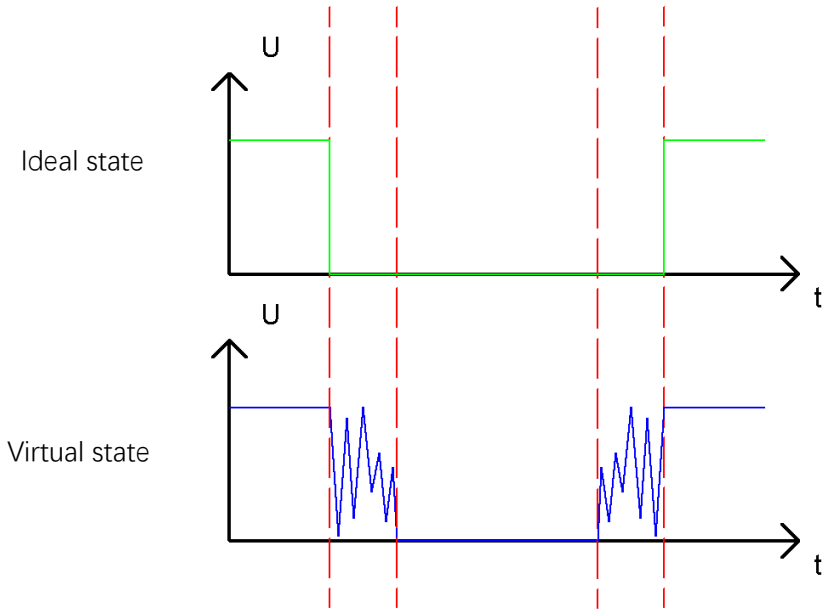
Component list

It is same as the previous project.

Circuit knowledge

Debounce for Push Button

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



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

Circuit

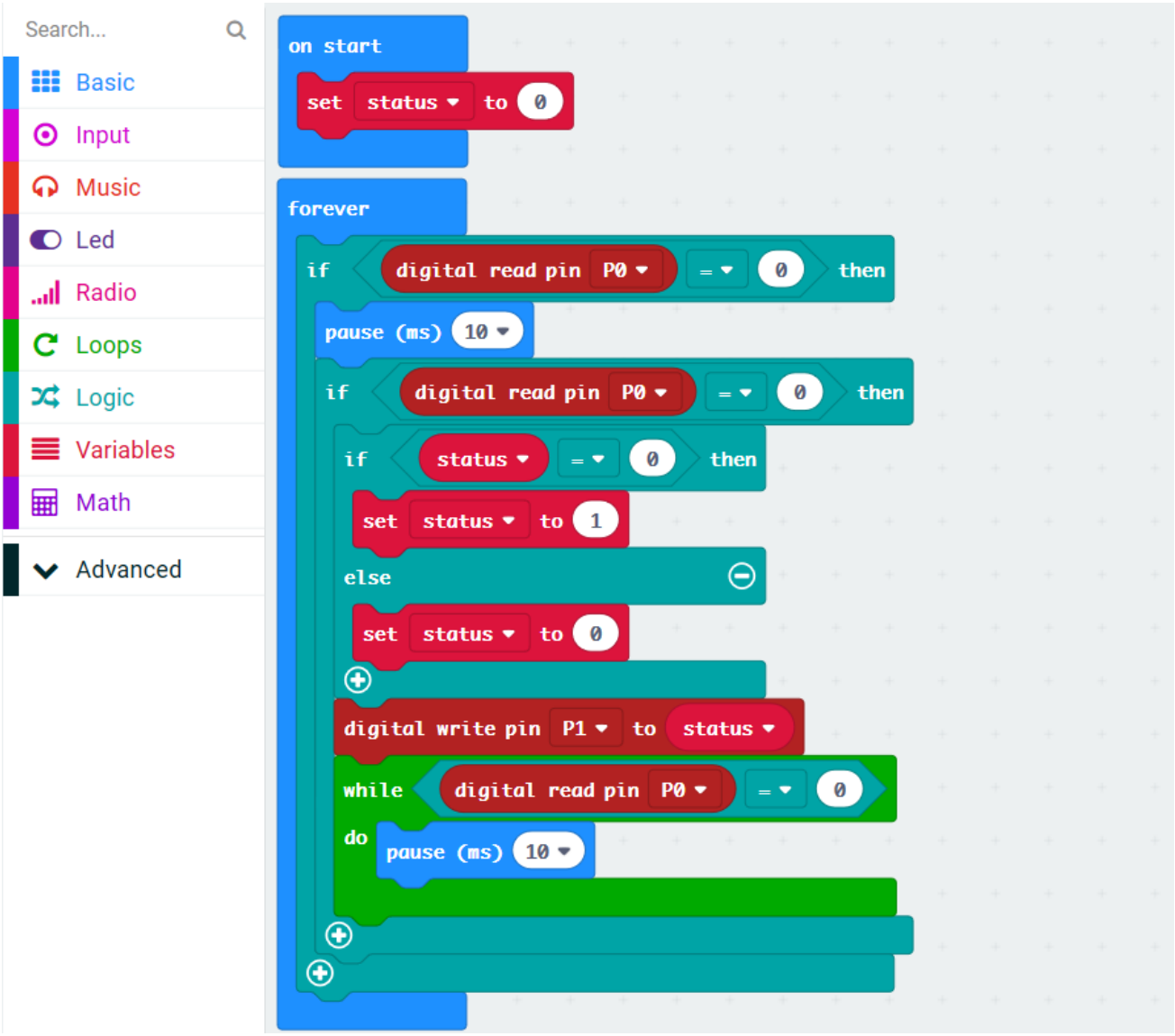
It is same as the previous section.

Block code

Open MakeCode first. Import the .hex file. The path is as below:
(How to import project)

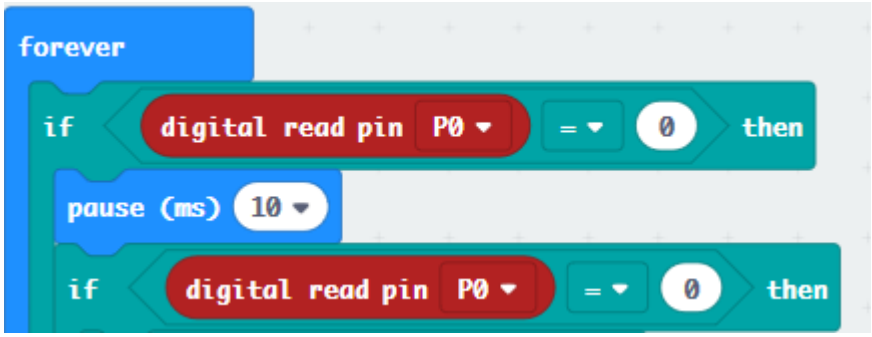
File type	Path	File name
HEX file	../Projects/BlockCode/04.2_TableLamp	TableLamp.hex

After importing successfully, the code is shown as below:

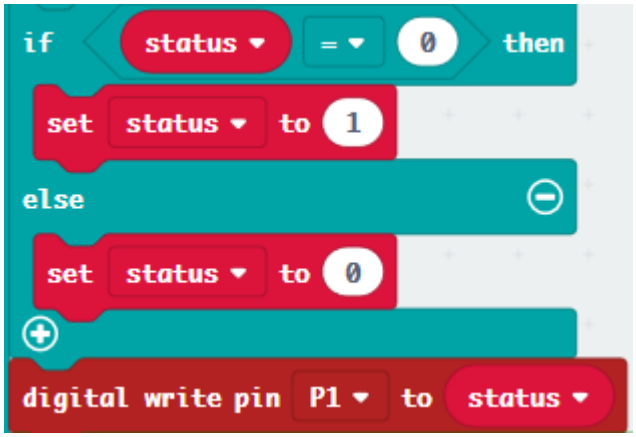


Download the code to micro:bit, press the button once, the LED turns ON, press the button again, the LED turns OFF. .

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 detected as pressed for the second time, the button is considered to have been pressed and in a steady state. Otherwise, it is considered to be a bounce and the program will stop detecting..

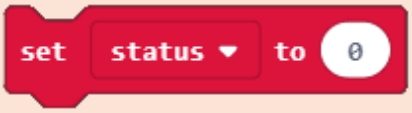


When it is determined that the key is pressed, change the status value. Status is used to save the state of LED. And then write the new value of status to the P1 port to control the LED.



After the above operations done, the program will detect whether the button is released. And similarly, it will first eliminate the bounce of the button.

Reference

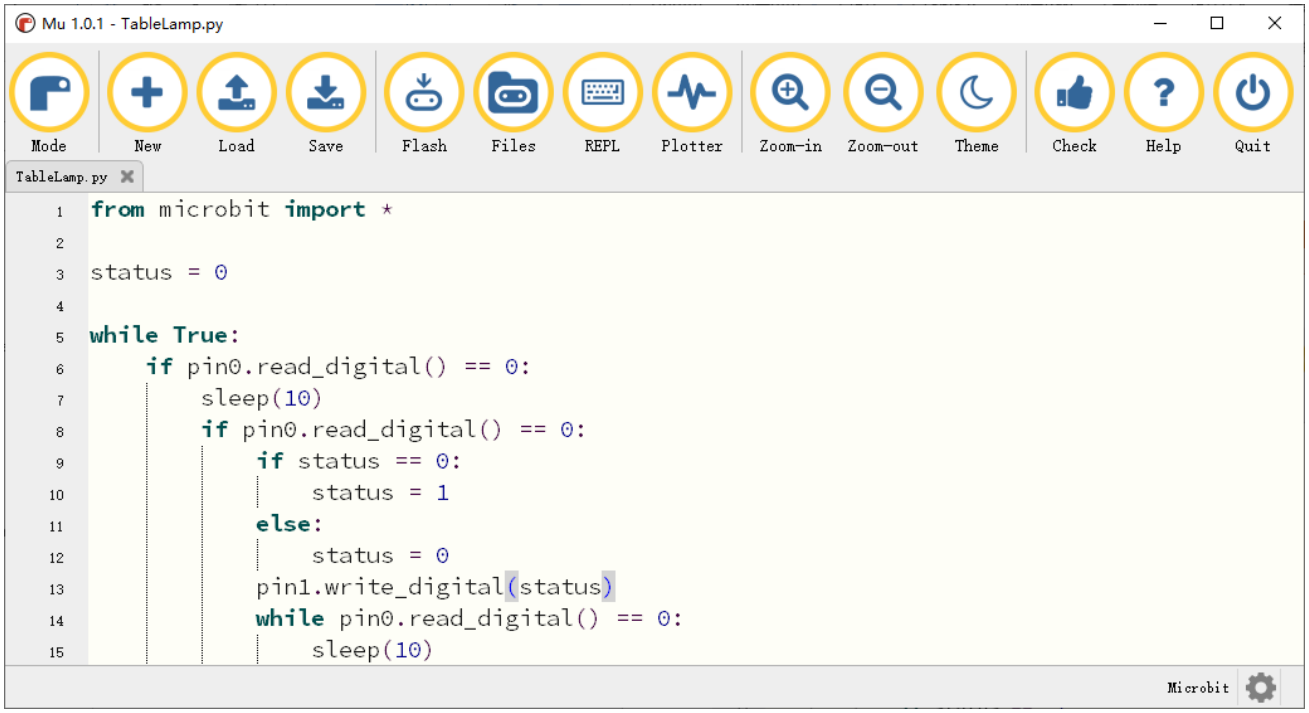
Block	Function
	Use an equal sign to make a variable store the number or string you set.

Python code

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

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

After loading successfully, the code is shown as below:



Download the code to micro:bit, press the button once, the LED turns ON; press the button again, the LED turns OFF.

The following is the program code:

```
from microbit import *
status = 0
while True:
    if pin0.read_digital() == 0:
        sleep(10)
        if pin0.read_digital() == 0:
            if status == 0:
                status = 1
            else:
                status = 0
            pin1.write_digital(status)
            while pin0.read_digital() == 0:
                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 eliminate the impact of 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 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 the above operations done, the program will detect whether the button is released. And similarly, it will first eliminate the bounce of the button.

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

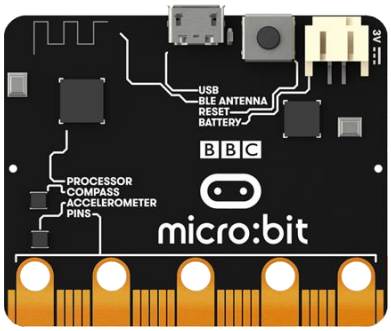
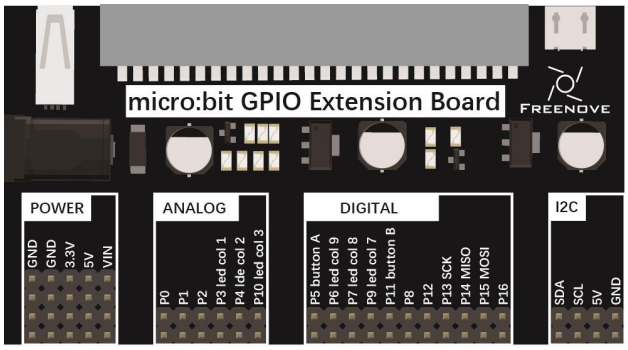
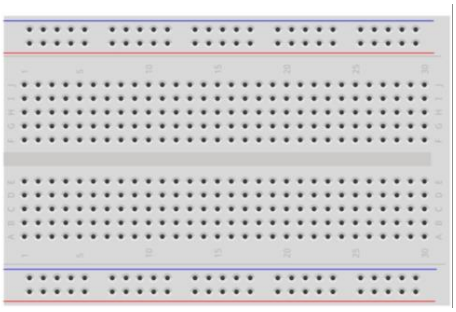
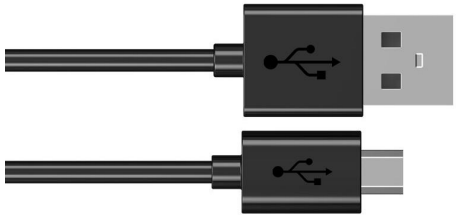
Chapter 5 LED Bar Graph

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

Project 5.1 Flowing Light

In this project, we use LED Bar Graph to make a flowing water light.

Component list

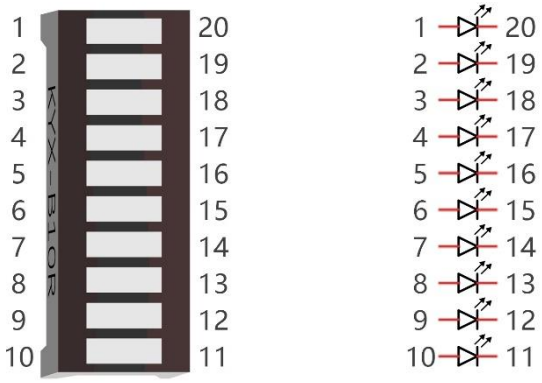
Microbit x1 	Expansion board x1 	
Breakboard x1 	Resistor 220Ω x10 	Jumper F/M x11 
LED bar graph x1 	USB cable x1 	

Component knowledge

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

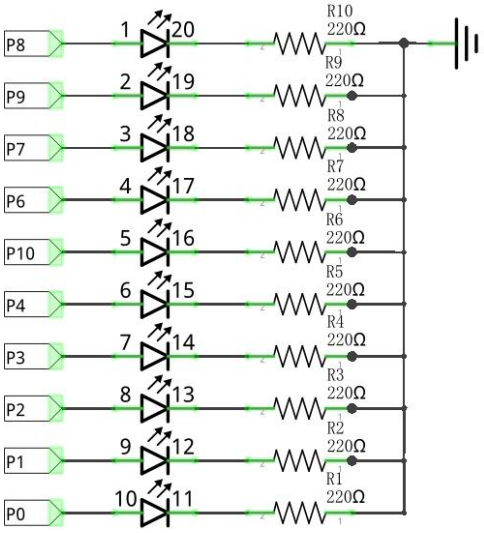
LED Bar Graph

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



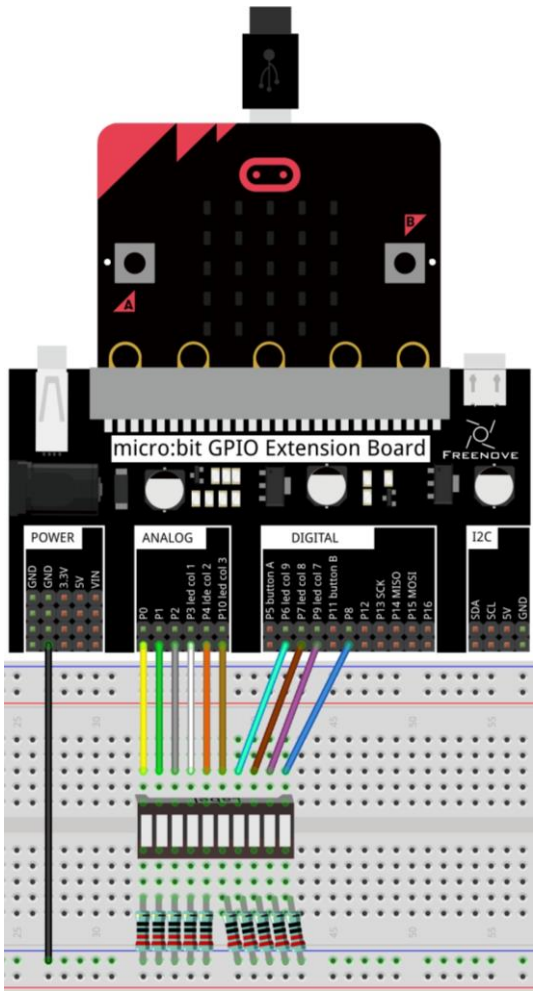
Circuit

Schematic diagram



Hardware connection

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

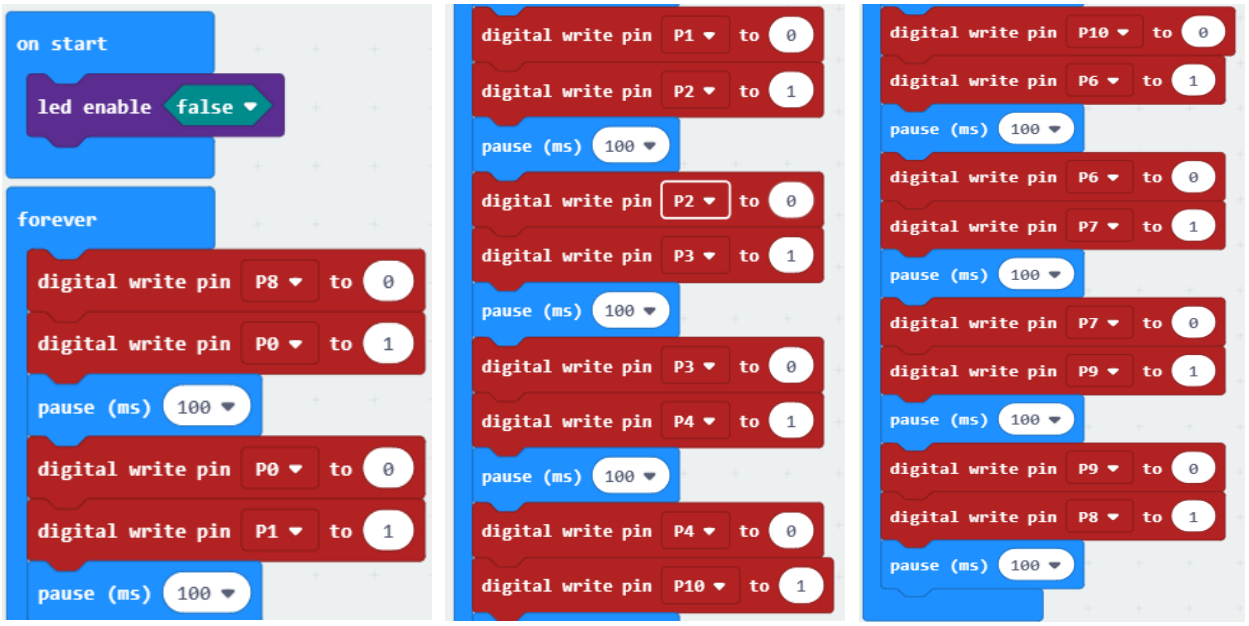


Block code

Open MakeCode first. Import the .hex file. The path is as below:
(How to import project)

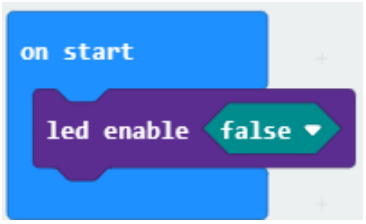
File type	Path	File name
HEX file	../Projects/BlockCode/05.1_FlowingLight01	FlowingLight01.hex

After importing successfully, the code is shown as below:

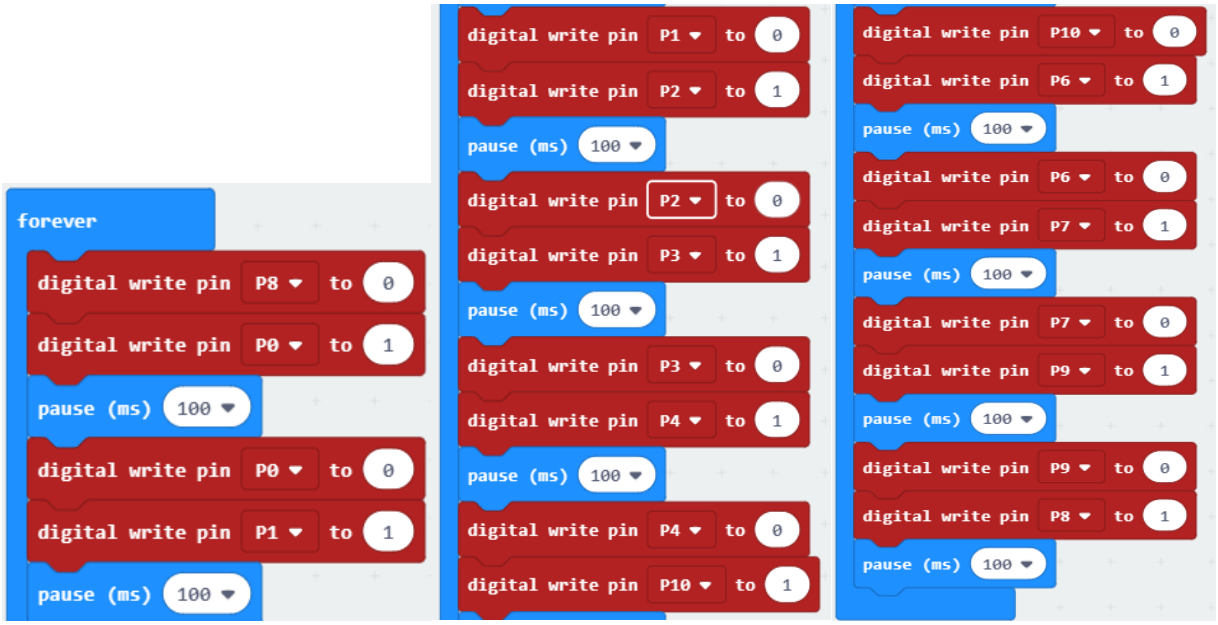


Check the connection of the circuit, verify that the circuit is connected correctly, download the code into micro:bit, after the program is executed, you will see the LED turns ON from left to right, which repeats This process is repeated to achieve the “movements” of flowing water.

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).



Set one pin to high and the rest of 9 pins to low level at a time; Until all the 10 pins are set to high and low level in turn.



Reference

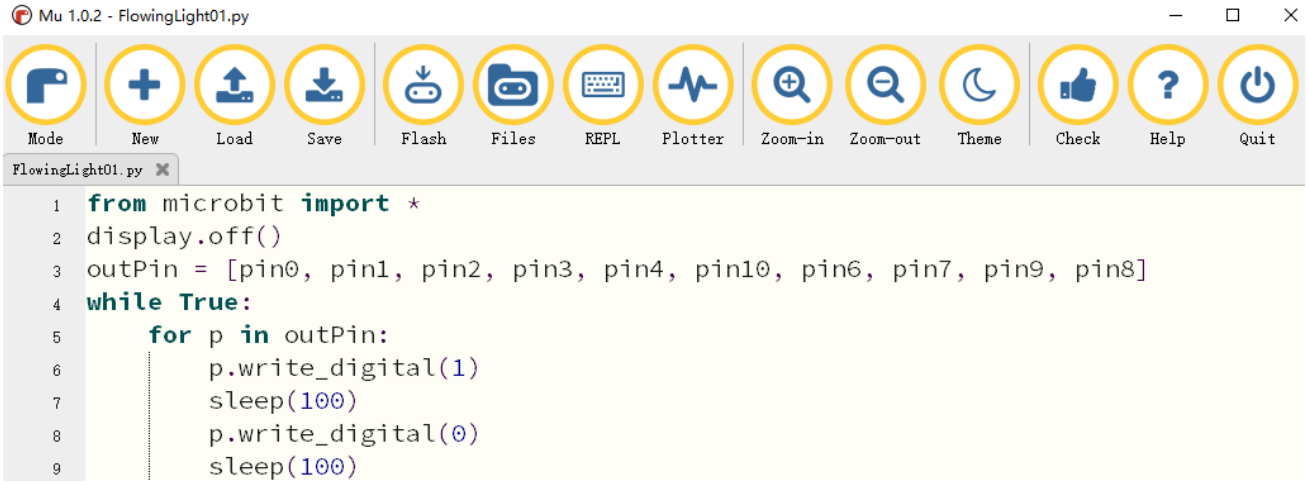
Block	Function
led enable false	Turns the LED screen on and off (thus allowing you to re-use the GPIO pins associated with the display for other purposes).

Python code

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

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

After loading successfully, the code is shown as below:



Check the connection of the circuit, verify that the circuit is connected correctly, download the code into micro:bit, after the program is executed, you will see the LED turns ON from left to right, which repeats to achieve the “movements” of flowing water.

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 to 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 effect of flowing water light.

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

Reference

display.off()

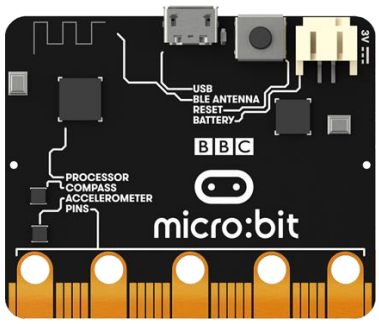
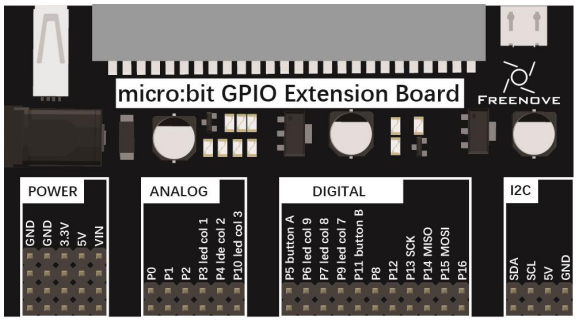
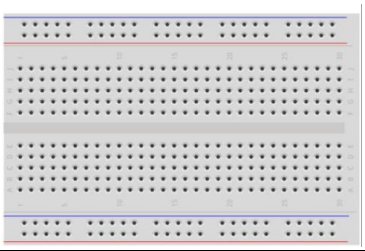
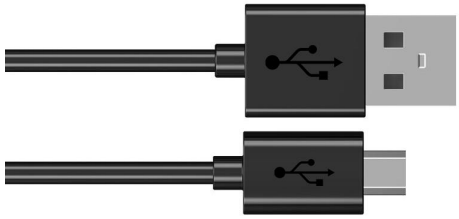
Use off() to turn OFF the display (thus allowing you to re-use the GPIO pins associated with the display for other purposes)

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

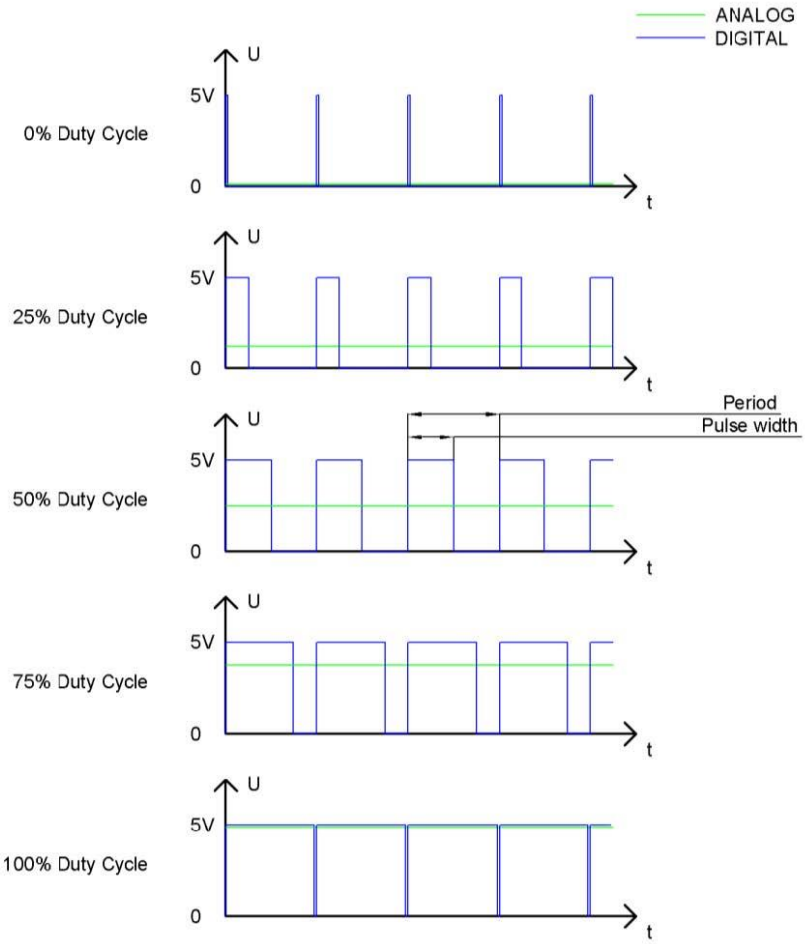
At first, let us learn the knowledge how to use the circuit to make LED emit different brightness of light,

PWM

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

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

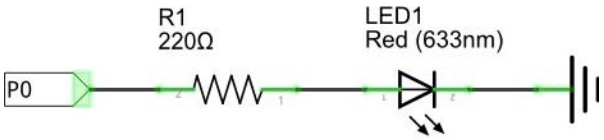
The longer the output of high levels last, the longer the duty cycle and the higher the corresponding voltage in the analog signal will be. The following figures show how the analog signal voltages vary between 0V-5V (high level is 5V) corresponding to the pulse width 0%-100%:



The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on.

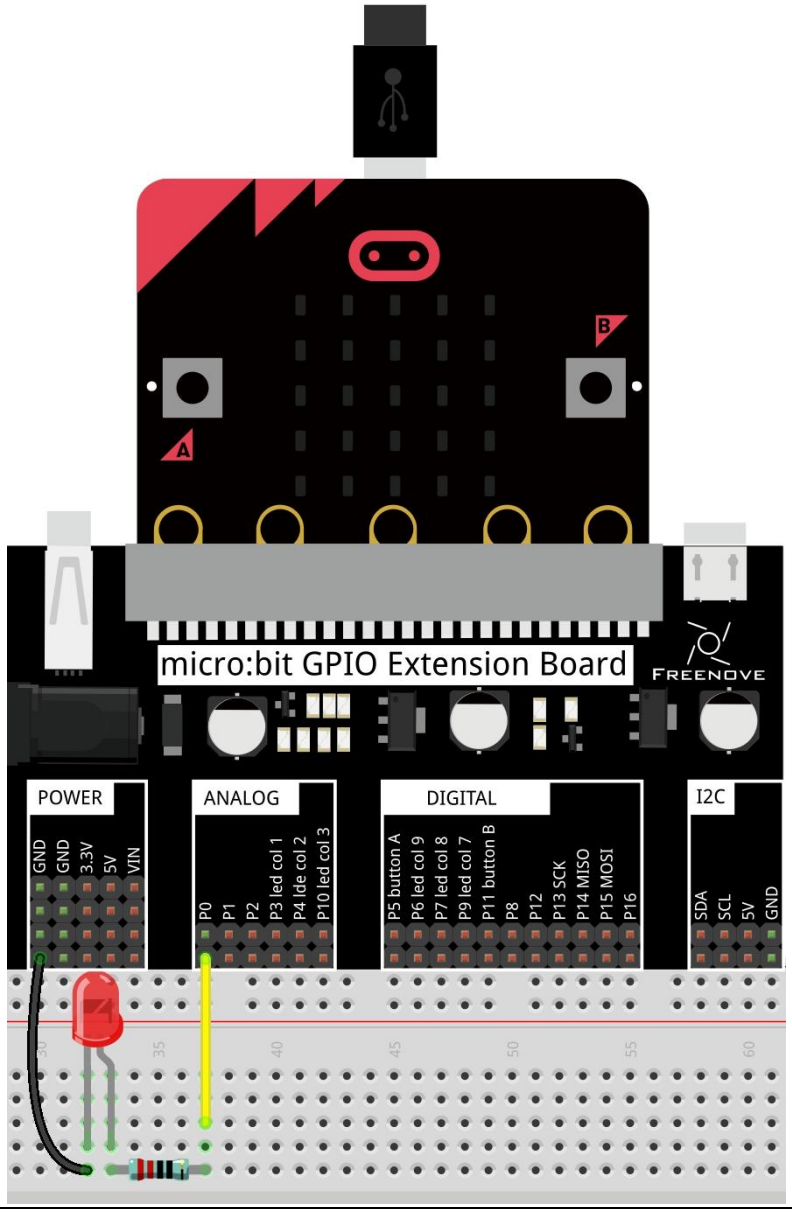
Circuit

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.

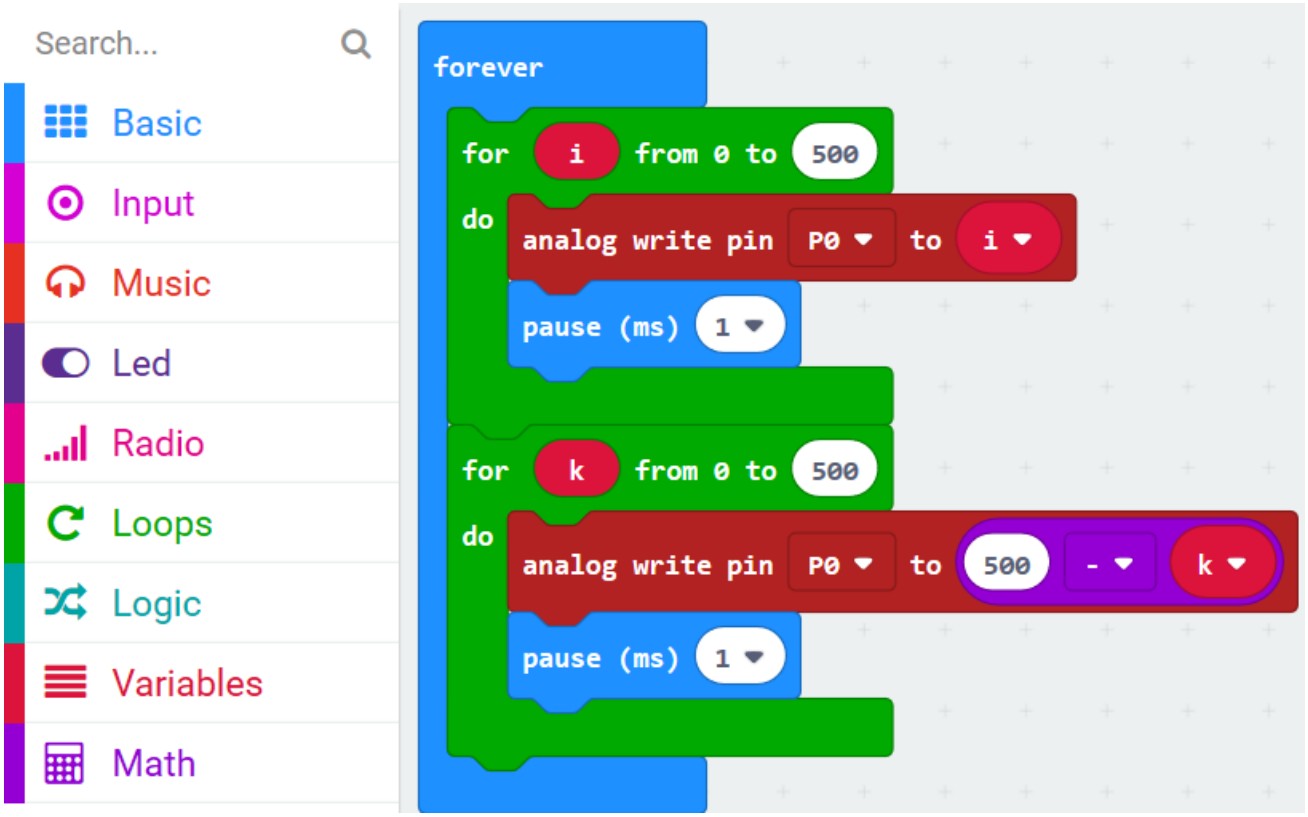


Block code

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 will becomes brighter gradually, and then dimmer and dimmer. This process will be repeated to achieve the effect of breathing.

P0 outputs PWM signal, from 0 to 500, then from 500 to 0.

Reference

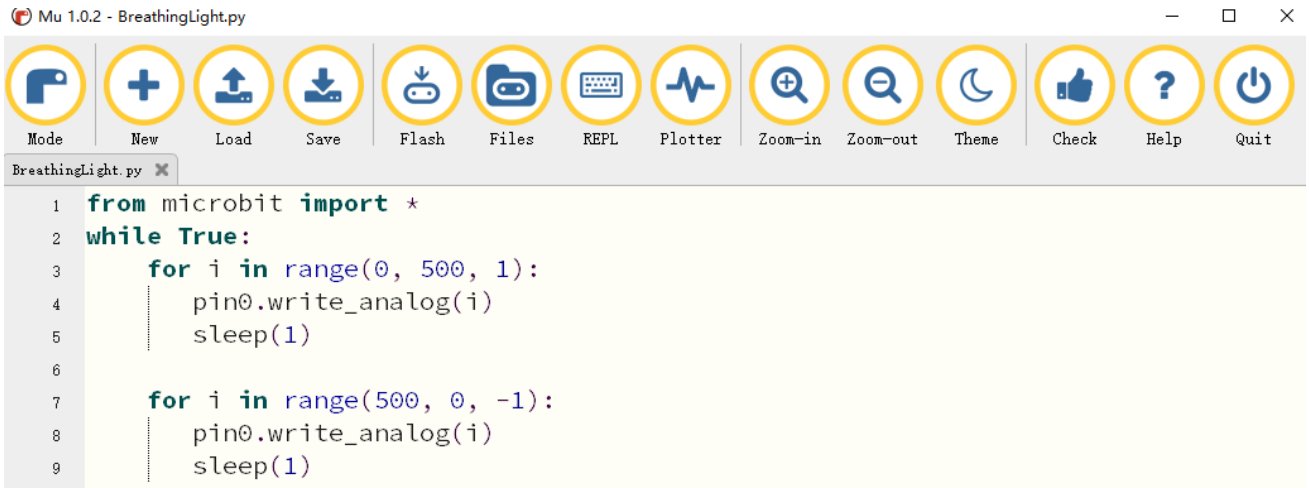
Block	Function
	Write an analog signal (0 through 1023) to the pin you set.

Python code

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

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

After loading successfully, the code is shown as below:



Check the connection of the circuit, verify it correct and download the code into micro:bit, the LED will becomes brighter gradually, and then dimmer and dimmer. This process will be repeated to achieve the effect of breathing. (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 outputs 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

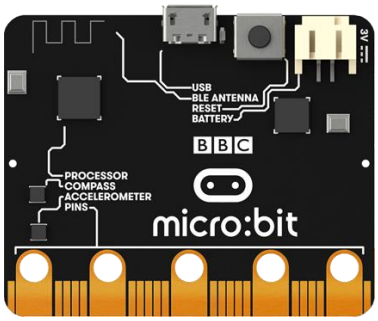
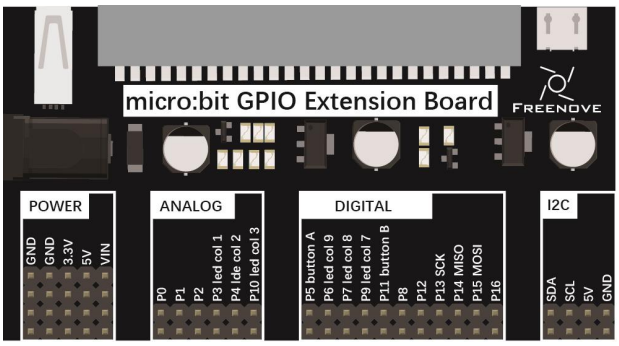
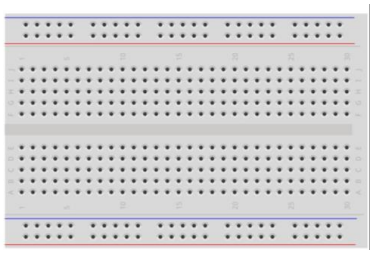
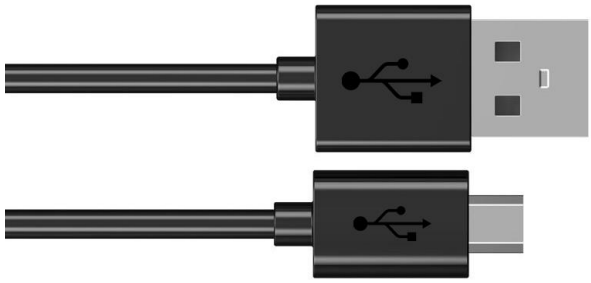
Chapter 7 RGBLED

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

Project 7.1 Monochromatic Light

This project will use RGBLED to show one color.

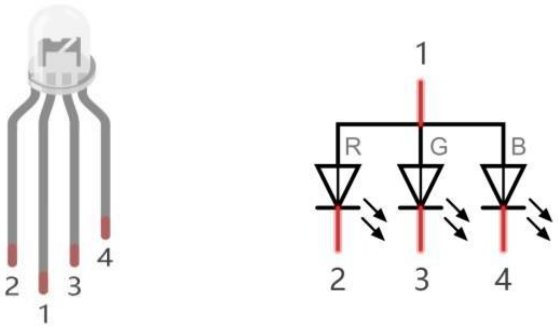
Component list

Microbit x1 	Extension board x1 	
Breadboard x1 	USB cable x1 	
RGB_LED x1 	Jumper F/M x4	Resistor 220Ω x3 

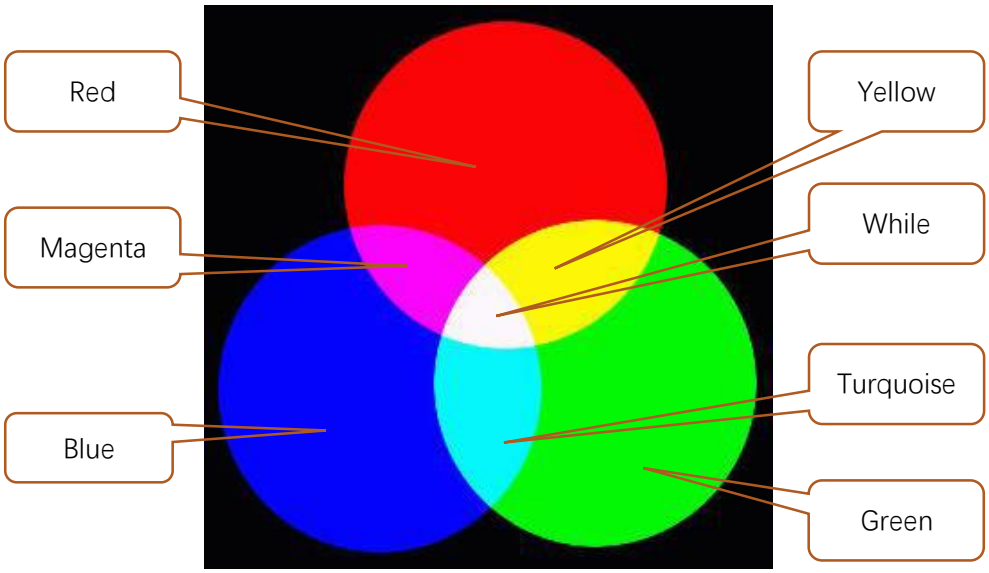
Component knowledge

RGB LED

A RGB LED has 3 LEDs integrated into one LED component. It can respectively emit Red, Green and Blue light. In order to do this, it requires 4 pins (this is also how you identify it). The long pin (1) is the common which is the Anode (+) or positive lead, the other 3 are the Cathodes (-) or negative leads. A rendering of a RGB LED and its electronic symbol are shown below. We can make RGB LED emit various colors of light and brightness by controlling the 3 Anodes (2, 3 & 4) of the RGB LED



Red, Green, and Blue light are called 3 Primary Colors when discussing light (Note: for pigments such as paints, the 3 Primary Colors are Red, Blue and Yellow). When you combine these three Primary Colors of light with varied brightness, they can produce almost any color of visible light. Computer screens, single pixels of cell phone screens, neon lamps, etc. can all produce millions of colors due to phenomenon

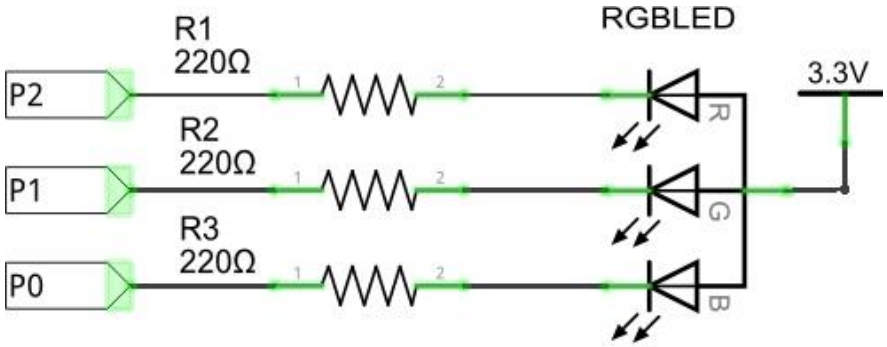


If we use a three 8 bit PWM to control the RGB LED, in theory, we can create 28*28*28=16777216 (16 million) colors through different combinations of RGB light brightness.

Circuit

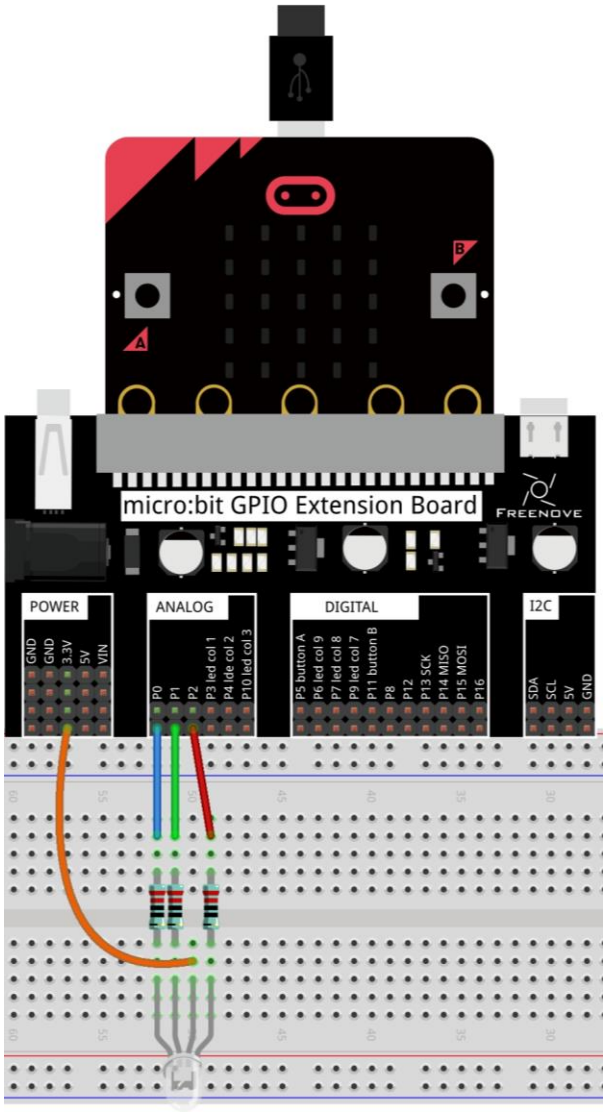
This circuit uses pins P2, P1, and P0 to connect the negative electrodes of the RGBLED.

Schematic diagram



Hardware connection

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

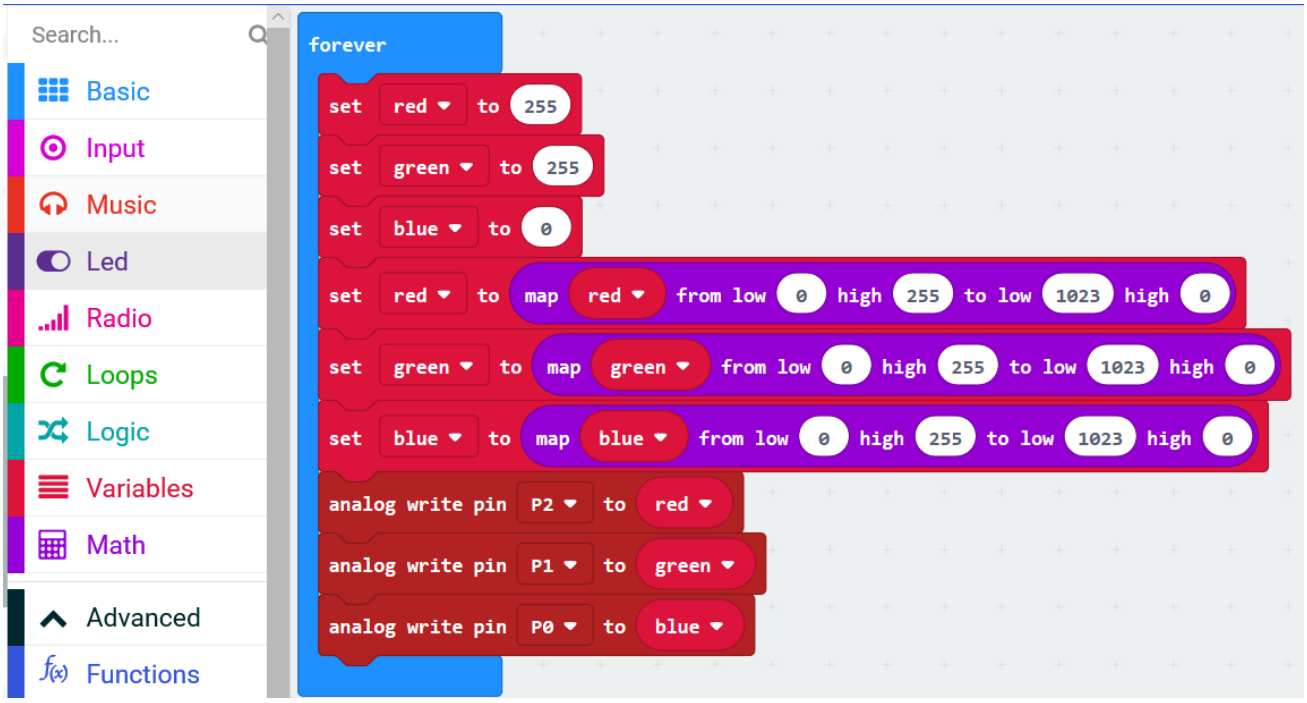


Block code

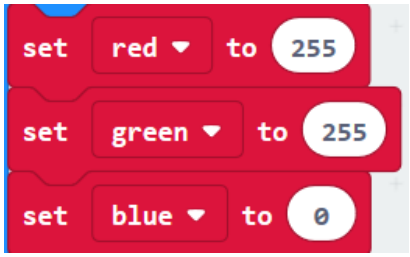
Open MakeCode first. Import the .hex file. The path is as below: (how to import project)

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

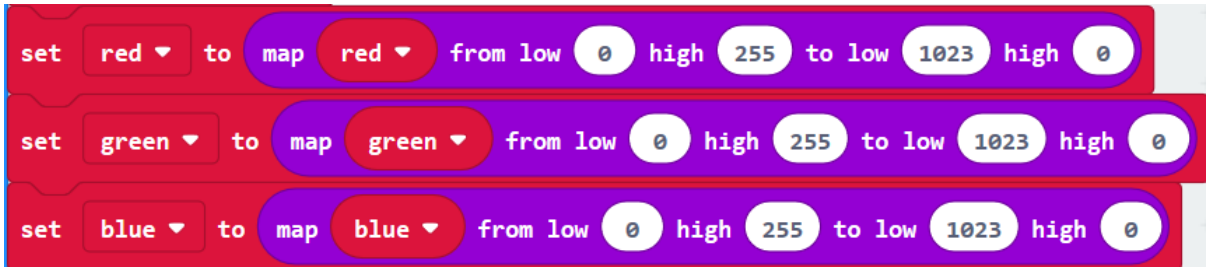
After importing successfully, the code is shown as below:



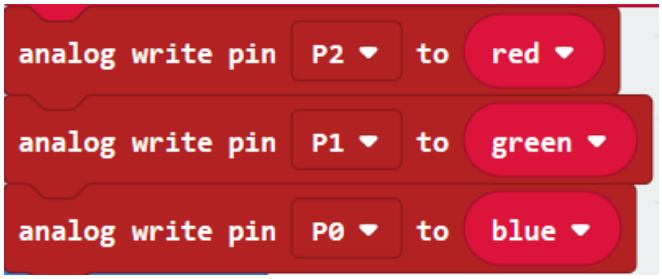
Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into micro:bit, and RGBLED will emit yellow color. RGB value of yellow is (255,255,0).



In this kit, three LEDs of RGB LED share a common anode(+) and their negative pins need to be set to LOW level to have the RGB LED work. And the value of variables 'red', 'green', and 'blue' need to be converted from the value ranging from 0-255 to analog signal values ranging from1023-0.



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



Reference

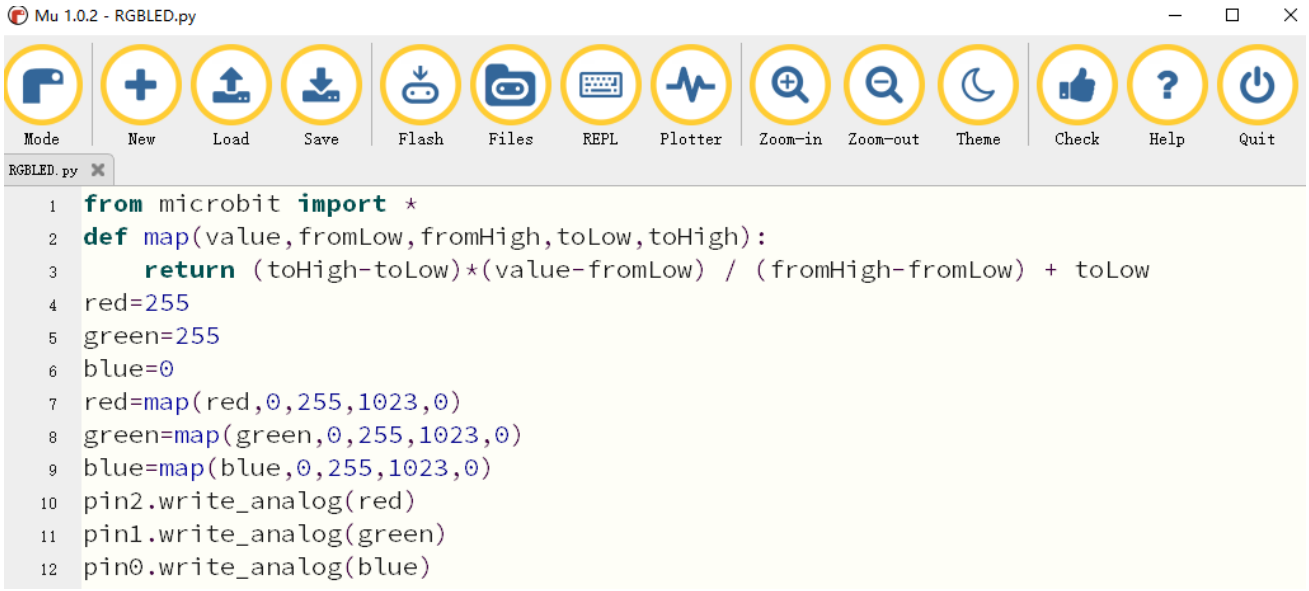
Block	Function
map 0 from low 0 high 1023 to low 0 high 4	Convert a value in one number range to a value in another number range.

Python code

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

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

After loading 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, and RGBLED will emit yellow color. (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  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 to convert a value in one range to another range.

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

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

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

In this kit, three LEDs of RGB LED share a common anode(+) and their negative pins need to be set to LOW level to have the RGB LED work. And the value of variables 'red', 'green', and 'blue' need to be converted from the value ranging from 0-255 to analog signal values ranging from1023-0.

```
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 Multicolored Light

In this project, we will use an RGB LED to emit different colors.

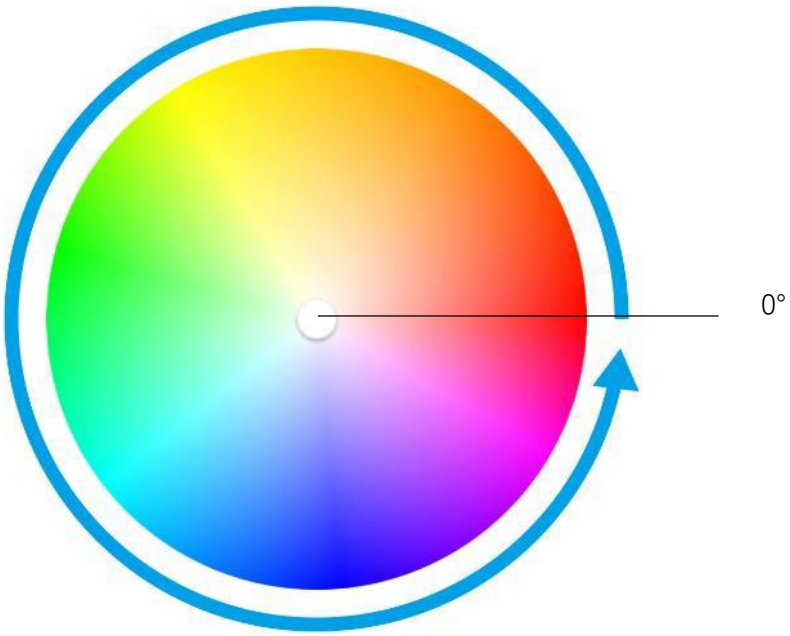
Component list

It is same with the previous project.

HSL color

The HSL color mode is another color standard in the industry. It obtains a variety of colors by changing the three color channels of hue (H), saturation (S), and lightness (L) and superimposing them with each other. This color mode covers 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. Each angle represents a color. The default saturation (S) takes the maximum value 100, the brightness (L) takes 50. If the hue angle is changed, the color will be changed. And the HSL color system can be converted to the RGB color system, to change the color of the LED.



Circuit

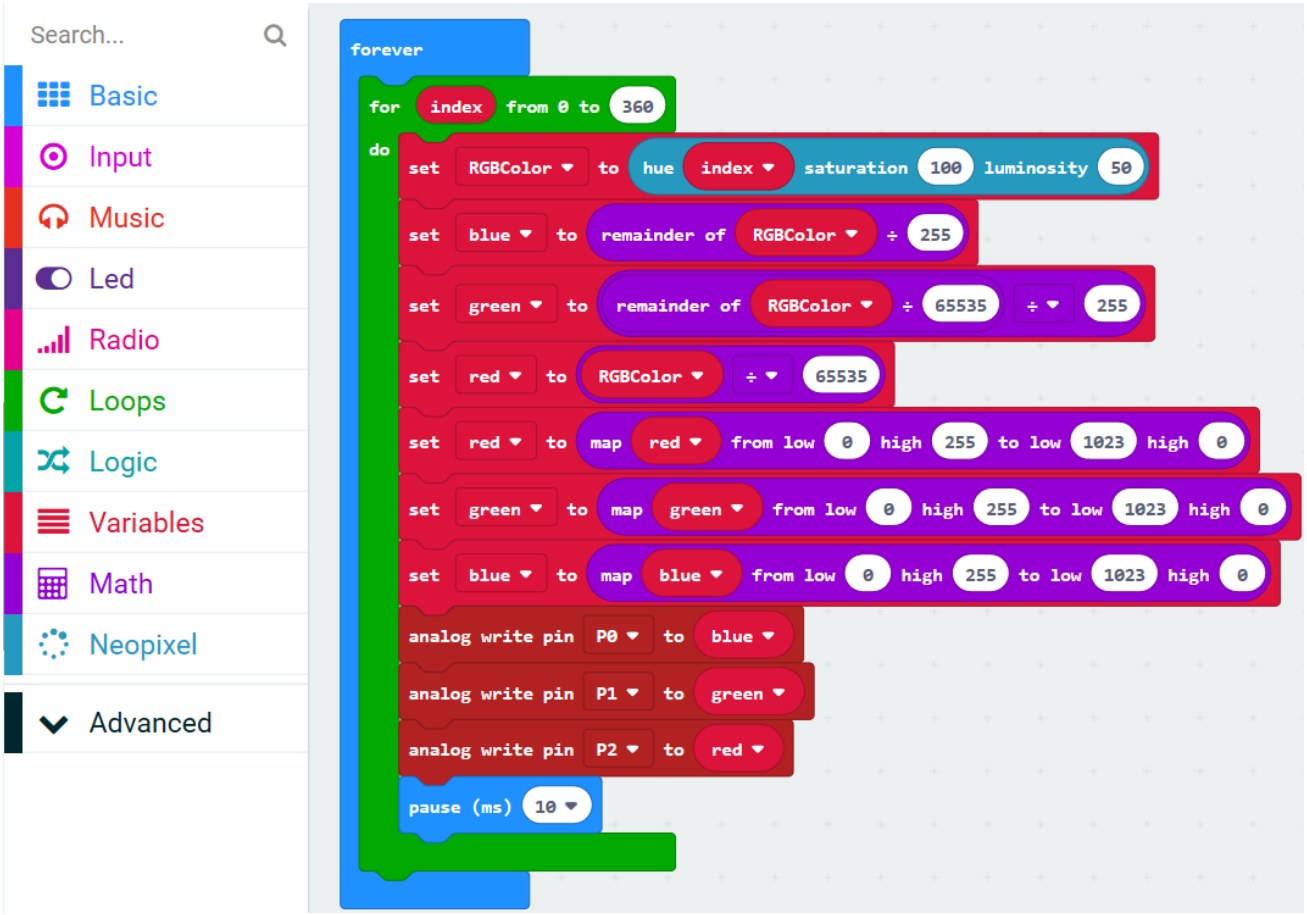
It is same with last project.

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 importing successfully, the code is shown as below:



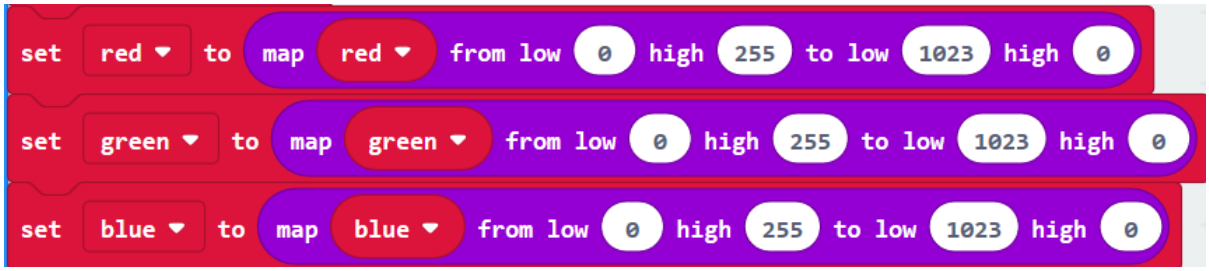
Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into the microbit, and RGBLED will emit different colors.

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.

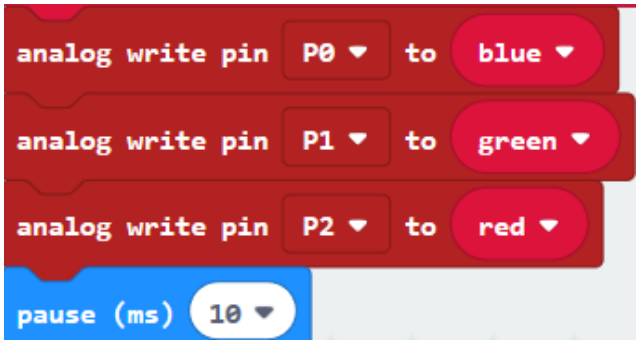
This block is to convert the HSL color system to the RGB color system, return the RGB value corresponding to the current hue angle, and store the value in the variable RGBColor. For example: hexadecimal RGB value 0xFF0000 means red, FF is the value of the red channel in RGB, and 00 and 00 are the values 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.

In this kit, three LEDs of RGB LED share a common anode (+) and their negative pins need to be set to LOW level to have the RGB LED work. And the variables 'red', 'green', and 'blue' need to be converted from the value ranging from 0-255 to analog signal values ranging from1023-0.



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

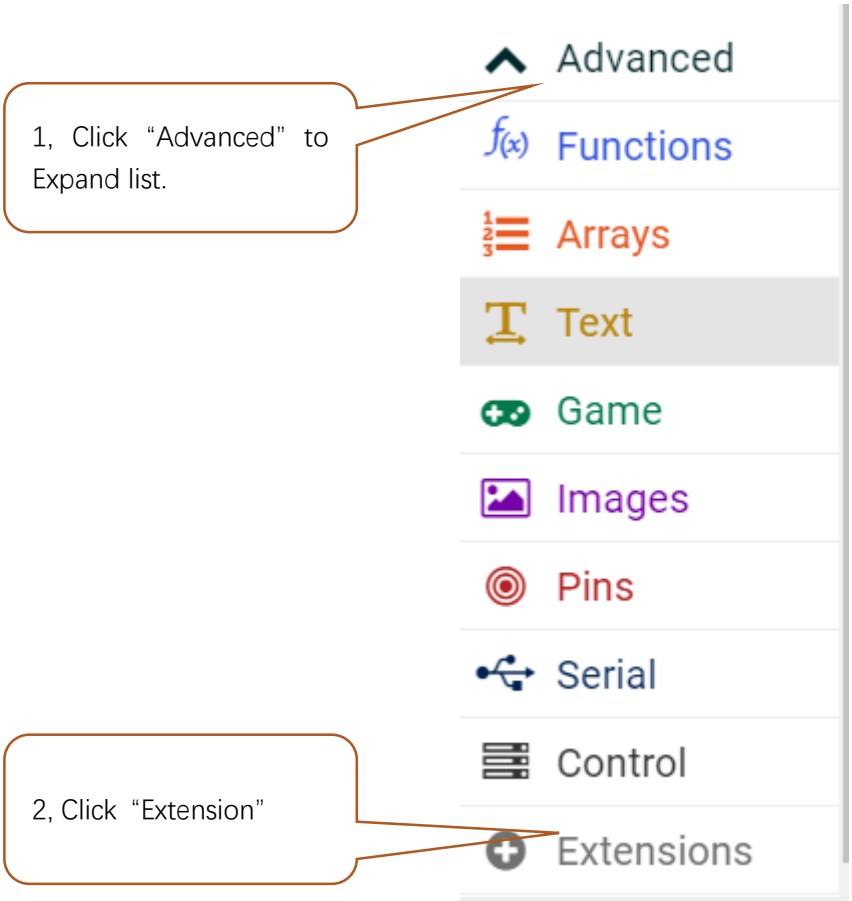


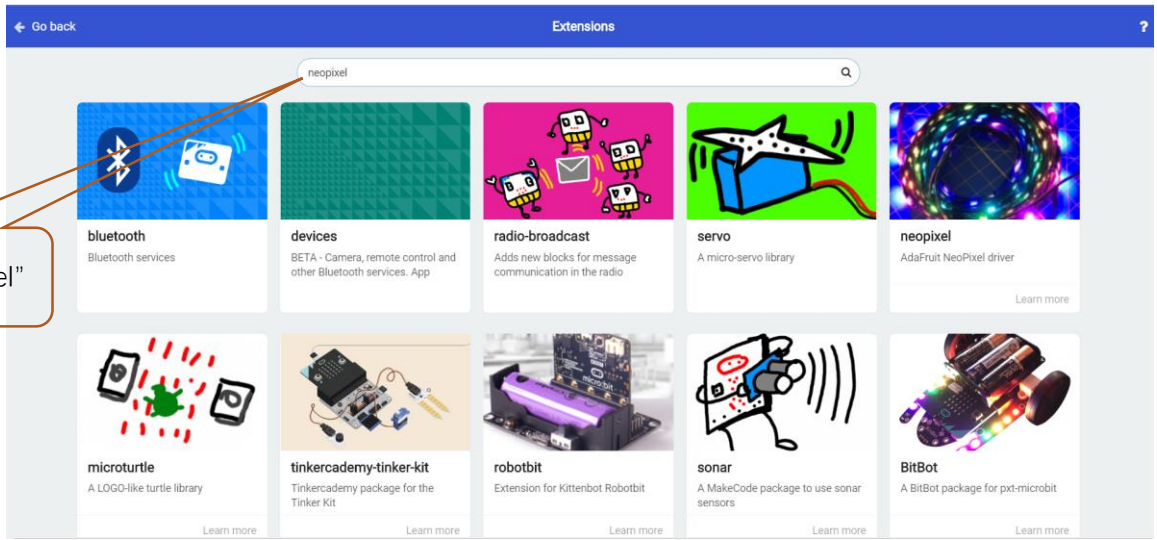
Reference

Block	Function
	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.
	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.

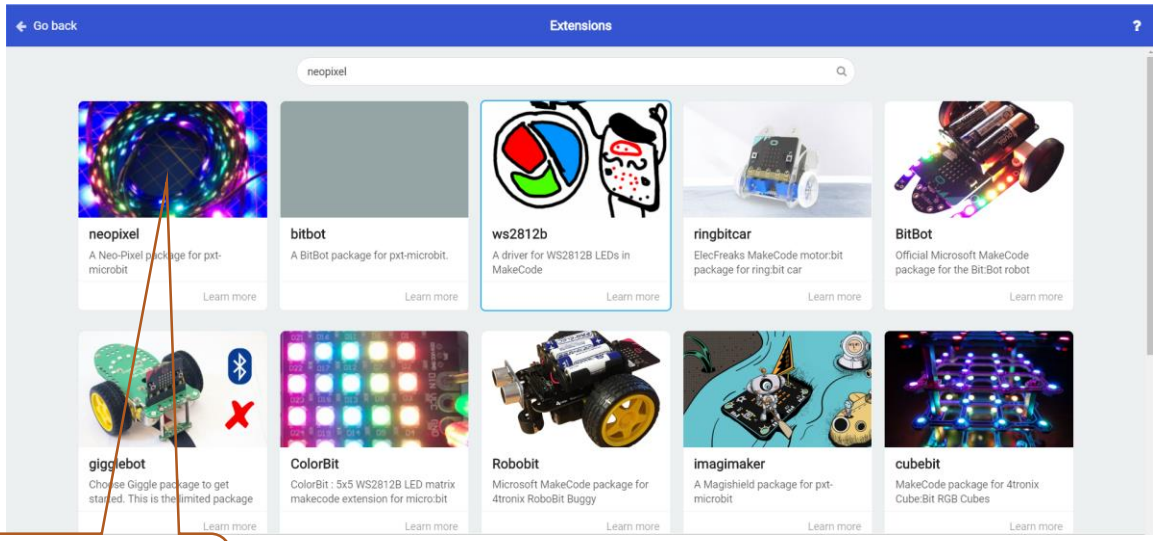
Extensions

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

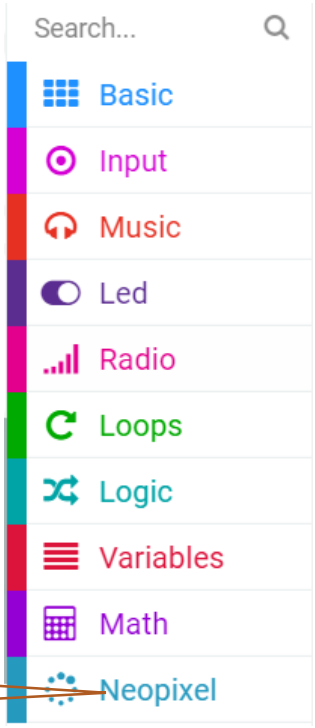




3, Search "neopixel"



4, Click this neopixel



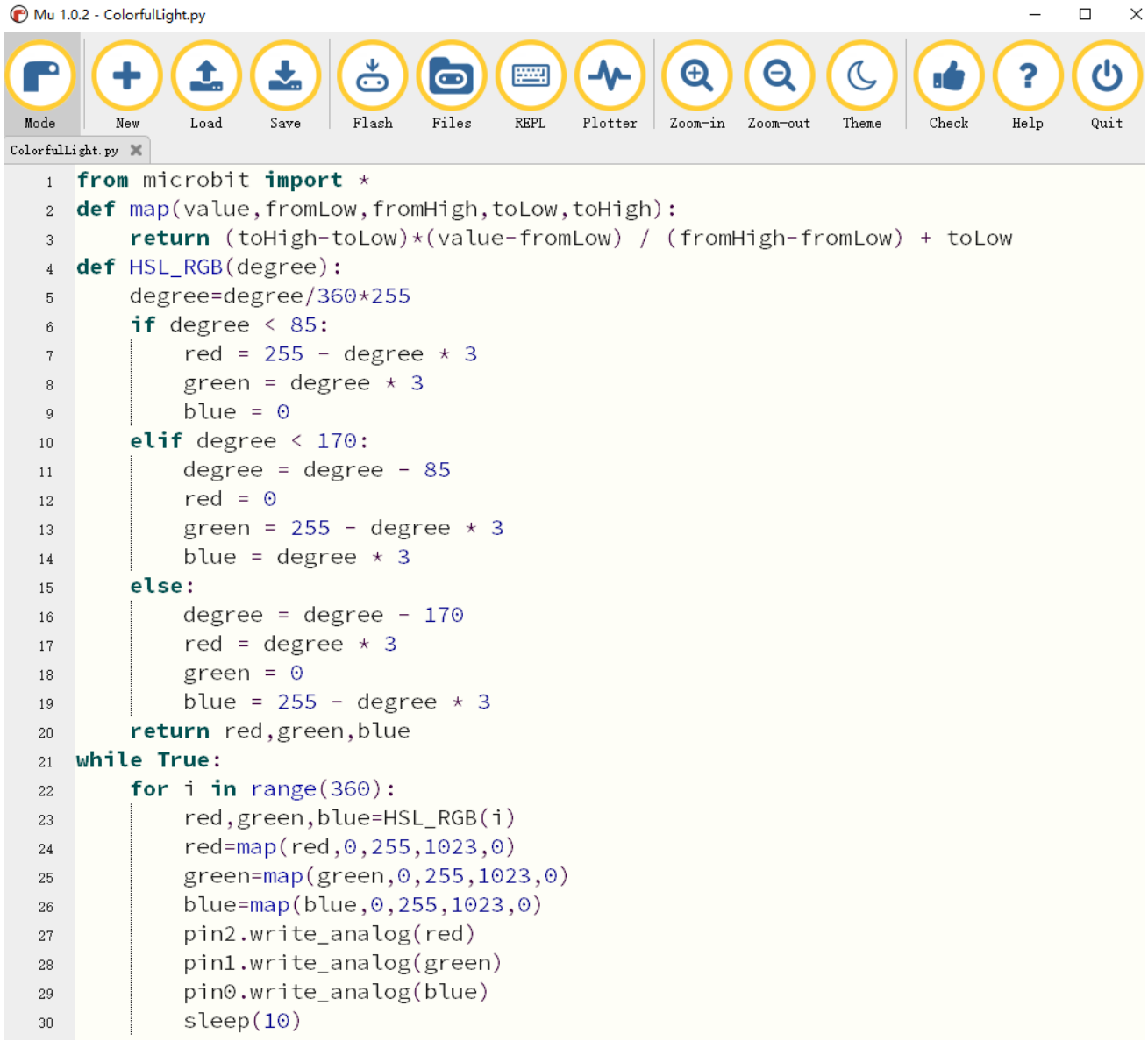
It is completed.

Python code

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

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

After loading successfully, the code is shown as below:




```
from microbit import *
def map(value,fromLow,fromHigh,toLow,toHigh):
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
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
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(10)
```

Check the connection of the circuit, confirm that the circuit is connected correctly, download the code into the microbit, and RGBLED will emit different colors. (How to download?)

The following is the program code:

```
from microbit import *
def map(value,fromLow,fromHigh,toLow,toHigh):
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
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
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(10)
```

The map() function is used to convert 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
```

Repeat 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(10)
```

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 Buzzer

In this chapter, we will learn about buzzers and the sounds they make. There are two kinds of buzzer: active buzzer and passive buzzer.

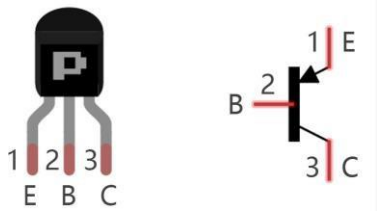
Component knowledge

Transistor

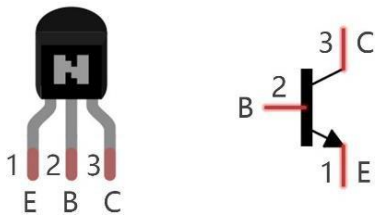
A transistor is required in this project due to the buzzer's current being so great that GPIO of RPi's output capability cannot meet the power requirement necessary for operation. A NPN transistor is needed here to amplify the current.

Transistors, full name: semiconductor transistor, is a semiconductor device that controls current (think of a transistor as an electronic "amplifying or switching device". Transistors can be used to amplify weak signals, or to work as a switch. Transistors have three electrodes (PINs): base (b), collector (c) and emitter (e). When there is current passing between "be" then "ce" will have a several-fold current increase (transistor magnification), in this configuration the transistor acts as an amplifier. When current produced by "be" exceeds a certain value, "ce" will limit the current output. at this point the transistor is working in its saturation region and acts like a switch. Transistors are available as two types as shown below: PNP and NPN,

PNP transistor



NPN transistor



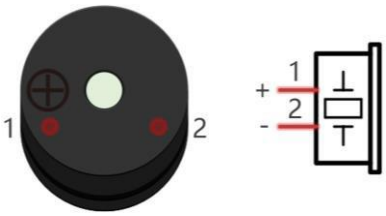
In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

Thanks to the transistor's characteristics, they are often used as switches in digital circuits. As micro-controllers output current capacity is very weak, we will use a transistor to amplify its current in order to drive components requiring higher current.

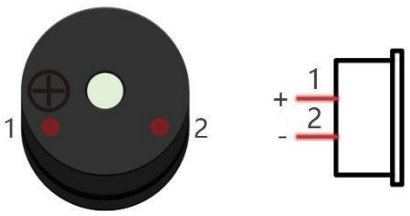
Buzzer

A buzzer is an audio component. They are widely used in electronic devices such as calculators, electronic alarm clocks, automobile fault indicators, etc. There are both active and passive types of buzzers. Active buzzers have oscillator inside, these will sound as long as power is supplied. Passive buzzers require an external oscillator signal (generally using PWM with different frequencies) to make a sound.

Active buzzer



Passive buzzer

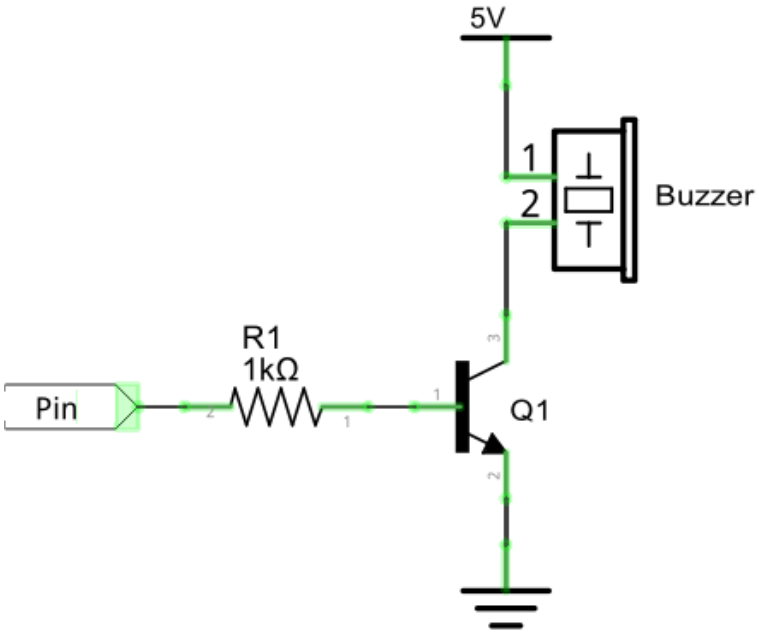


(A white label is attached on the active buzzer)

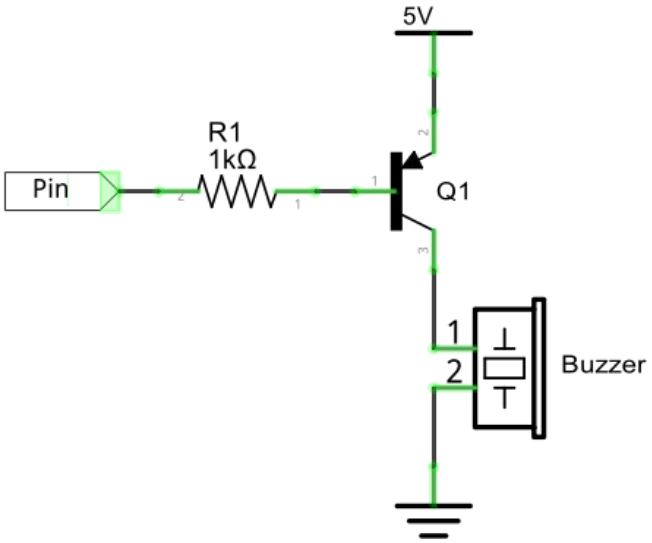
Active buzzers are easier to use. Generally, they only make a specific sound frequency. Passive buzzers require an external circuit to make sounds, but passive buzzers can be controlled to make sounds of various frequencies. The resonant frequency of the passive buzzer in this Kit is 2kHz, which means the passive buzzer is the loudest when its resonant frequency is 2kHz.

Buzzer requires large current when it works. But generally, microcontroller port cannot provide enough current for that. In order to control buzzer through micro:bit, a transistor can be used to drive a buzzer indirectly.

When we use a NPN transistor to drive a buzzer, we often use the following method. If GPIO outputs high level, current will flow through R1 (Resistor 1), the transistor conducts current and the buzzer will make sounds. If GPIO outputs low level, no current will flow through R1, the transistor will not conduct current and buzzer will remain silent (no sounds).



When we use a PNP transistor to drive a buzzer, we often use the following method. If GPIO outputs low level, current will flow through R1. The transistor conducts current and the buzzer will make sounds. If GPIO outputs high level, no current flows through R1, the transistor will not conduct current and buzzer will remain silent (no sounds). Below are the circuit schematics for both a NPN and PNP transistor to power a buzzer.



How to identify active and passive buzzer?

1. As a rule, there is a label on an active buzzer covering the hole where sound is emitted, but there are exceptions to this rule.

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

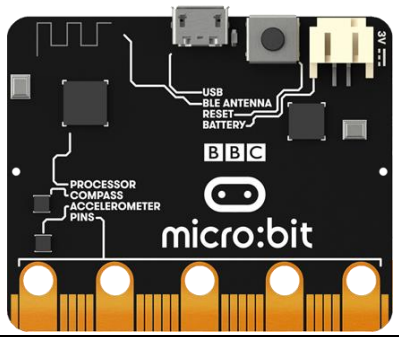
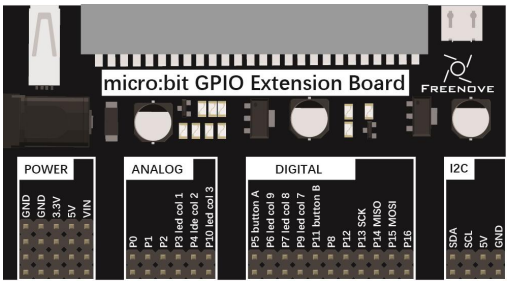
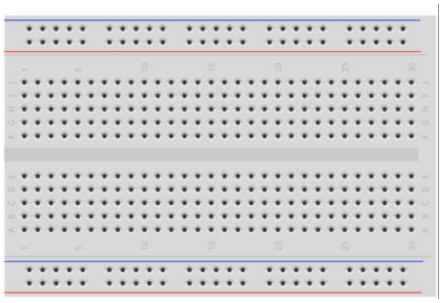
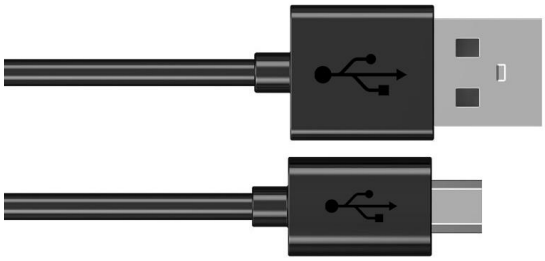
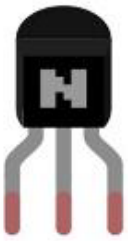
Active buzzer bottom

Passive buzzer bottom

Project 8.1 Active Buzzer

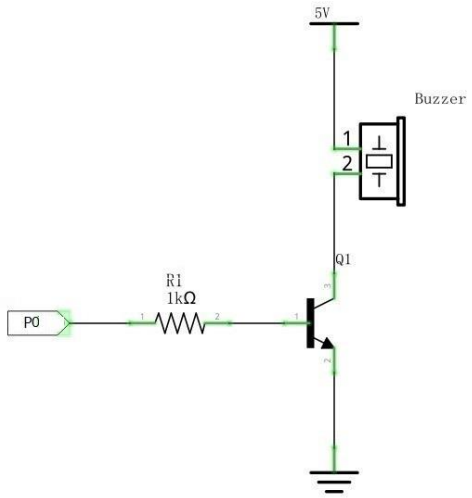
In this project, we will use an active buzzer to play a fixed melody.

Component list

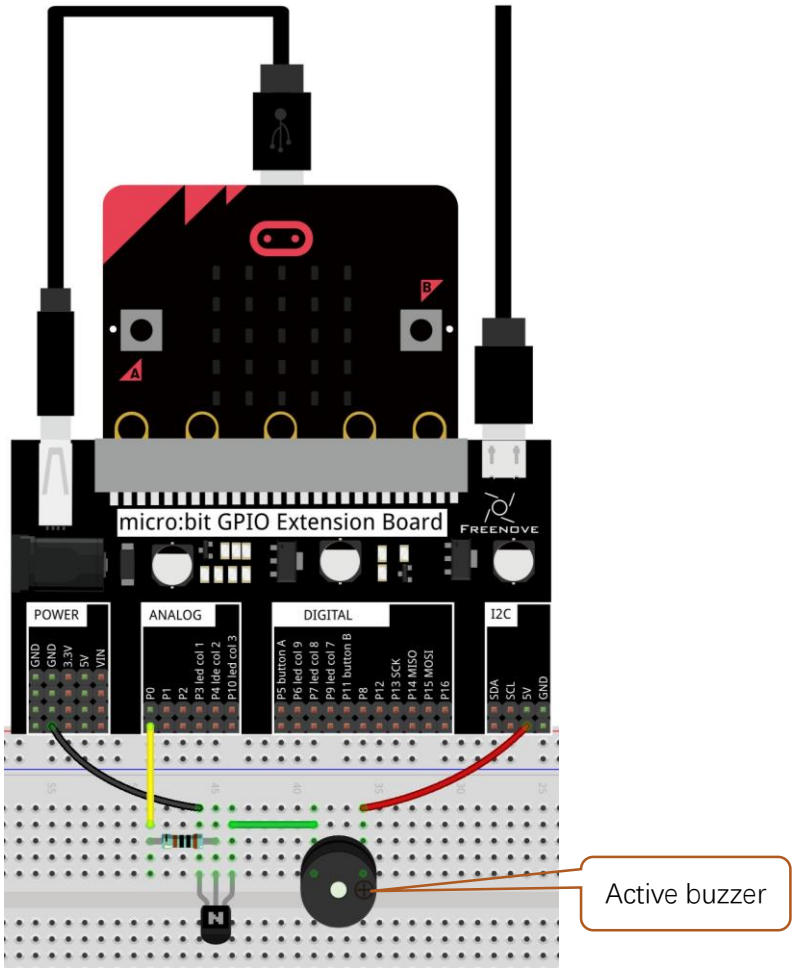
<table>
<tr><td>Microbit x1
  </td><td colspan="3">Extension board x1
  </td></tr>
<tr><td>Breadboard x1
  </td><td colspan="3">USB cable x2
  </td></tr>
<tr><td>F/M x 3 M/M x1
  </td><td>NPN(8050) transistor x1
  </td><td>Resistor 1kΩ x1
  </td><td>Active buzzer x1
  </td></tr>
</table>

Circuit

Schematic diagram



Hardware connection

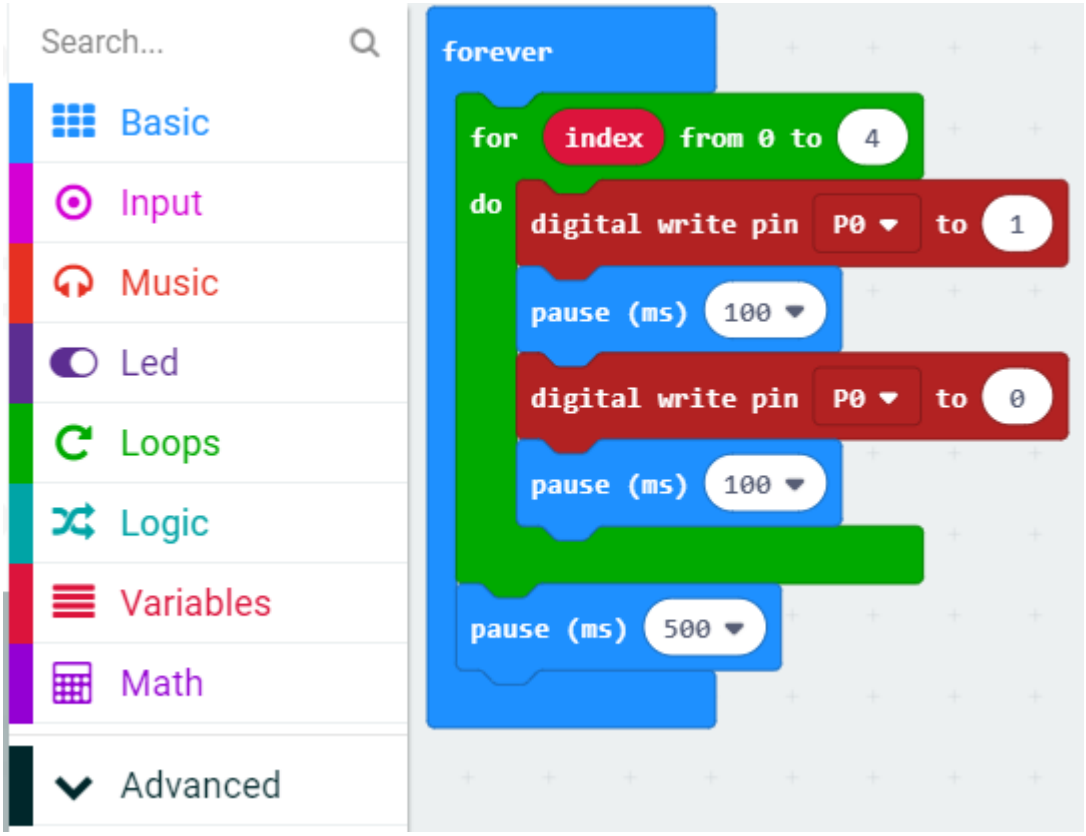


Block code

Open MakeCode first. Import the .hex file. The path is as below:
(How to import project)

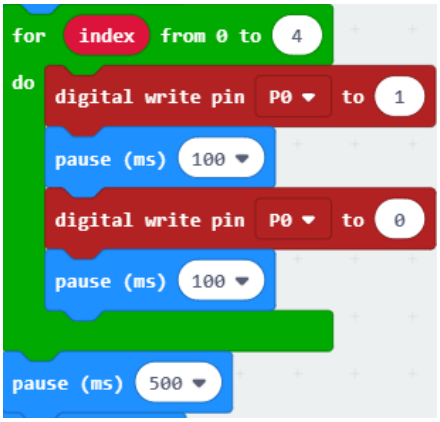
File type	Path	File name
HEX file	../Projects/BlockCode/08.1_ActiveBuzzer	ActiveBuzzer.hex

After importing 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.

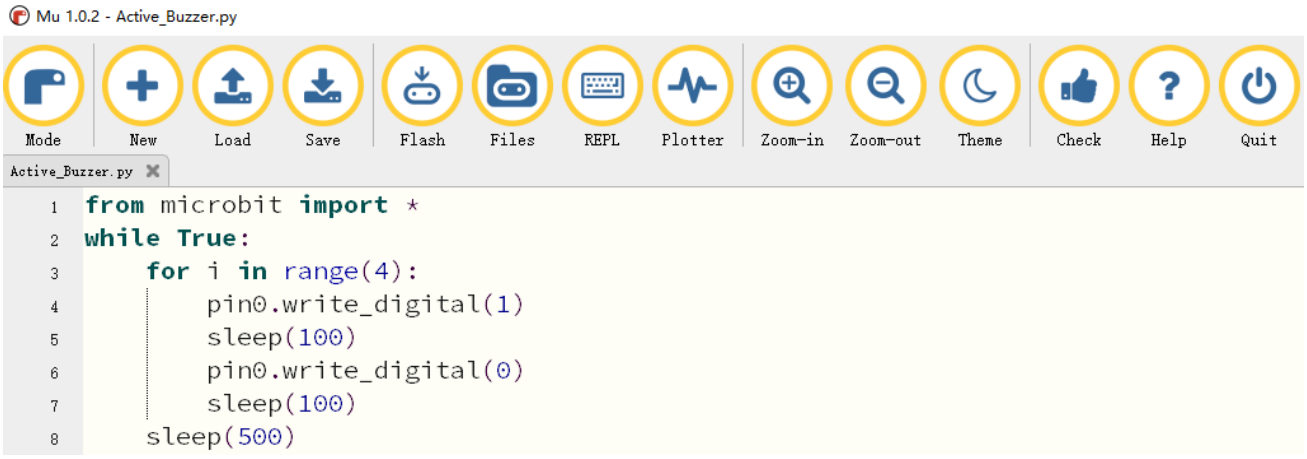


Python code

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

File type	Path	File name
Python file	../Projects/PythonCode/08.1_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 sound.

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)
```

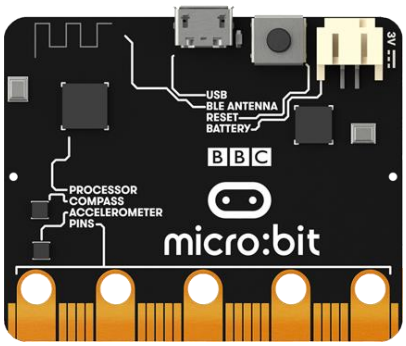
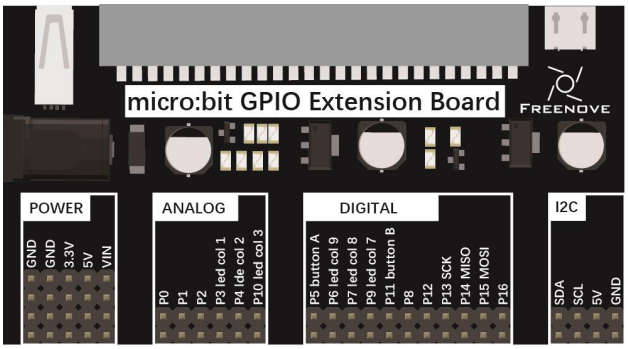
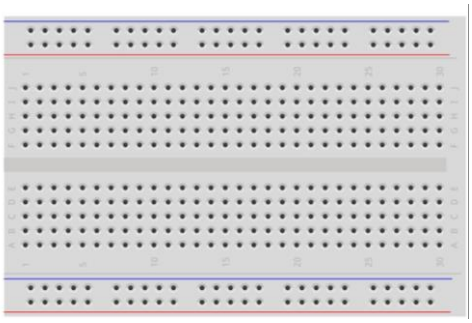
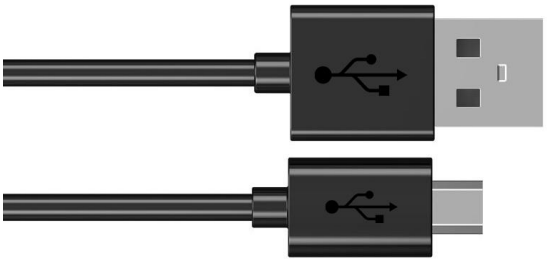
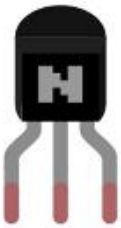
In the for loop, P0 outputs a high level to make the buzzer sound, then delay 100ms. And then P0 outputs a low level to stop the buzzer. Then delay 100ms. After the loop ends, delay 500ms.

```
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 8.2 Happy Birthday Melody

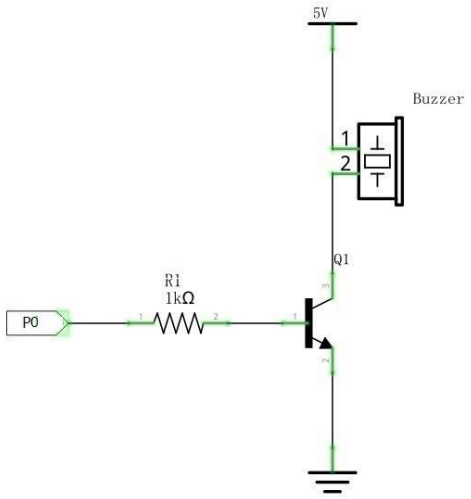
In this project, we will make a passive buzzer to play a happy birthday melody.

Component list

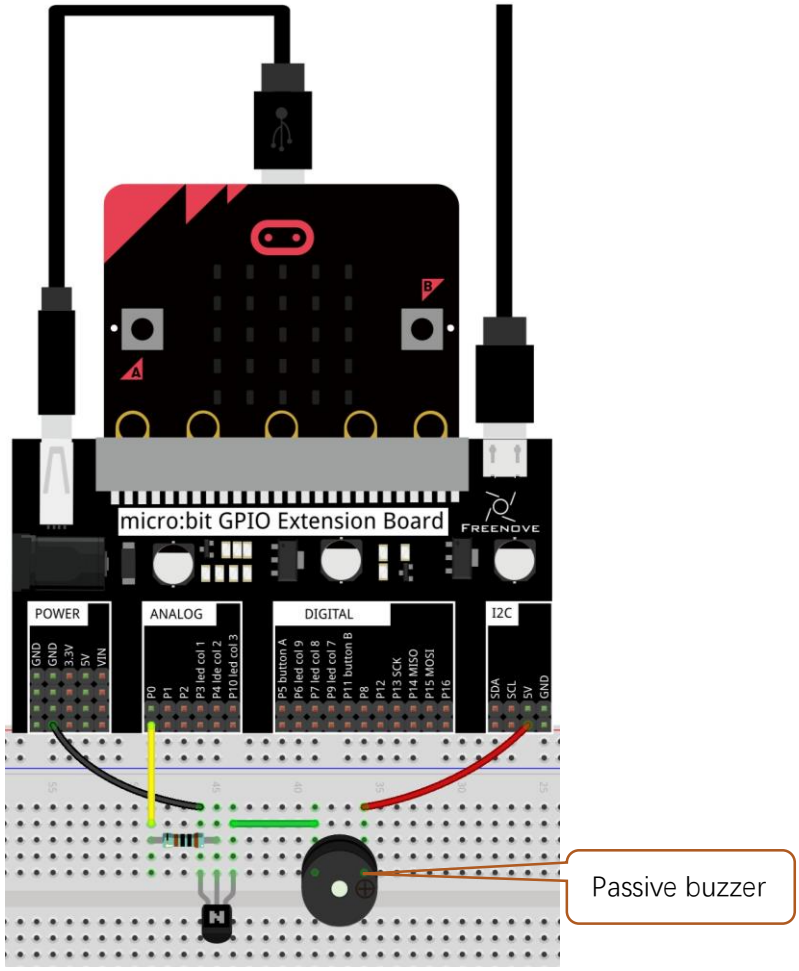
Microbit x1 		Extension board x1 	
Breadboard x1 		USB cable x2 	
F/M x3 M/M x1 	NPN(8050) transistor x1 	Resistor 1kΩ x1 	Passive buzzer x1 

Circuit

Schematic diagram



Hardware connection

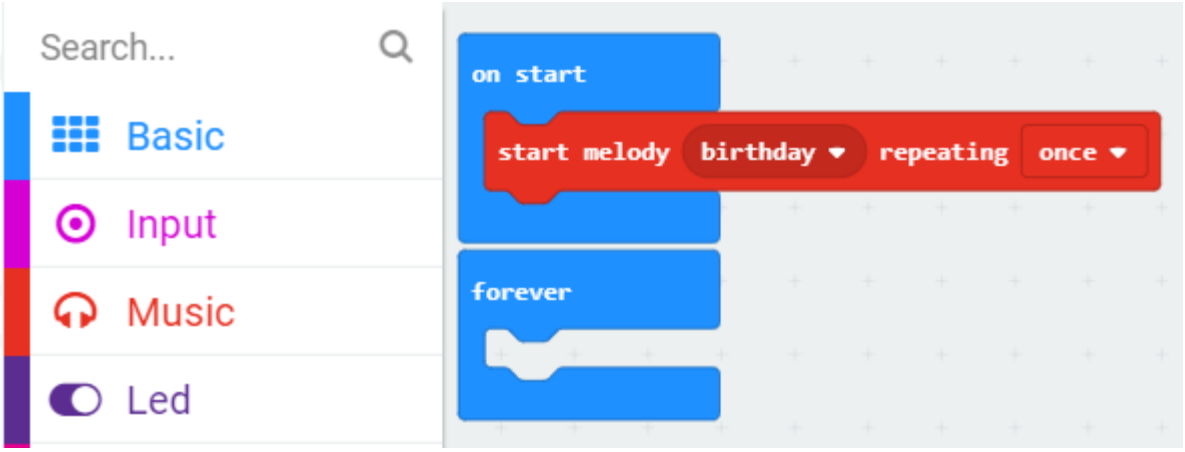


Block code

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/08.2_Play-a-melody	Play-a-melody.hex

After importing 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 song “happy birthday”.

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

Reference

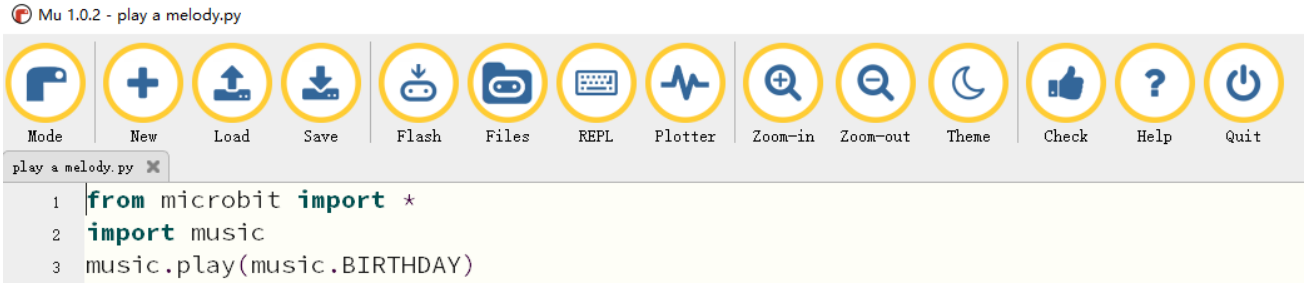
Block	Function
	Begin playing a musical melody through pin P0 of the micro:bit. There are built-in melodies that you can choose from the start melody block. These are already composed for you and are easy to use by just selecting the one you want.

Python code

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

File type	Path	File name
Python file	../Projects/PythonCode/08.2_Play-a-melody	Play-a-melody.py

After loading 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 song “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.

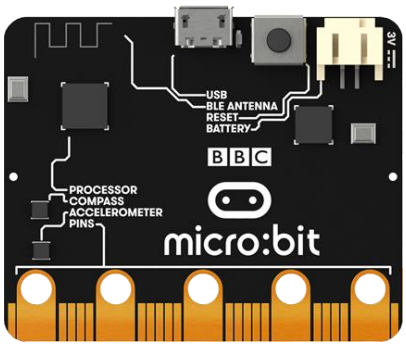
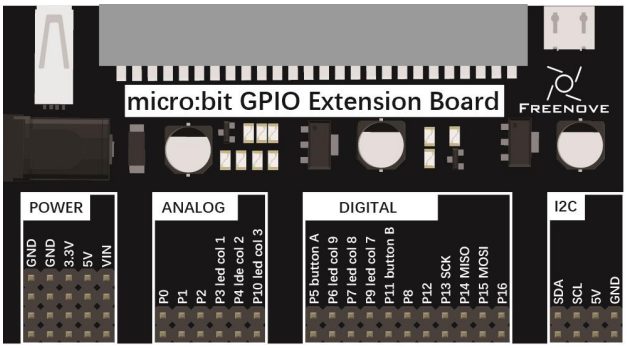
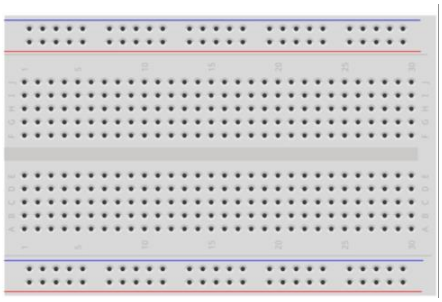
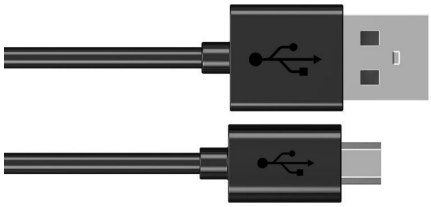
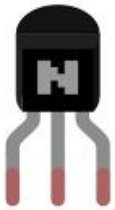
For more information, please refer to:

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

Project 8.3 Custom Melody

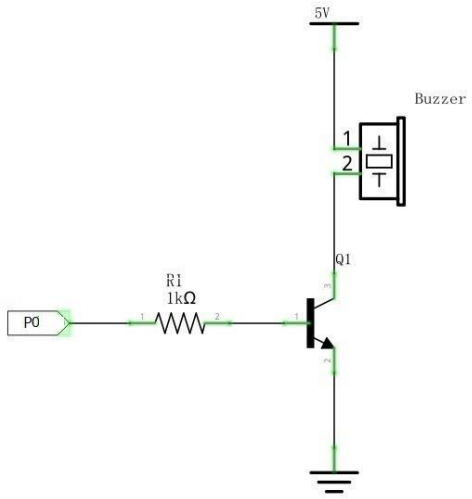
In this project, we will make the passive buzzer play a custom melody.

Component list

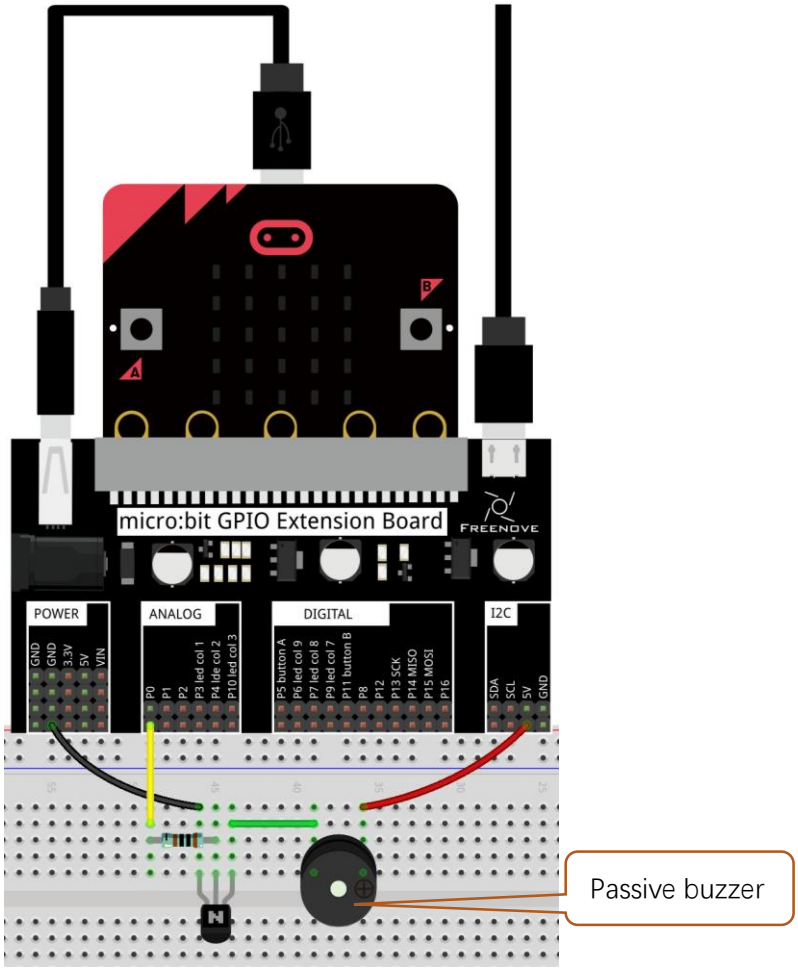
<table>
<tr><td>Microbit x1
  </td><td colspan="3">Extetnsion board x1
  </td></tr>
<tr><td>Breadboard x1
  </td><td colspan="3">USB cable x2
  </td></tr>
<tr><td>F/M x3 M/M x1
  </td><td>NPN(8050) transistor x1
  </td><td>Resistor 1kΩ x1
  </td><td>Passive buzzer x1
  </td></tr>
</table>

Circuit

Schematic diagram



Hardware connection

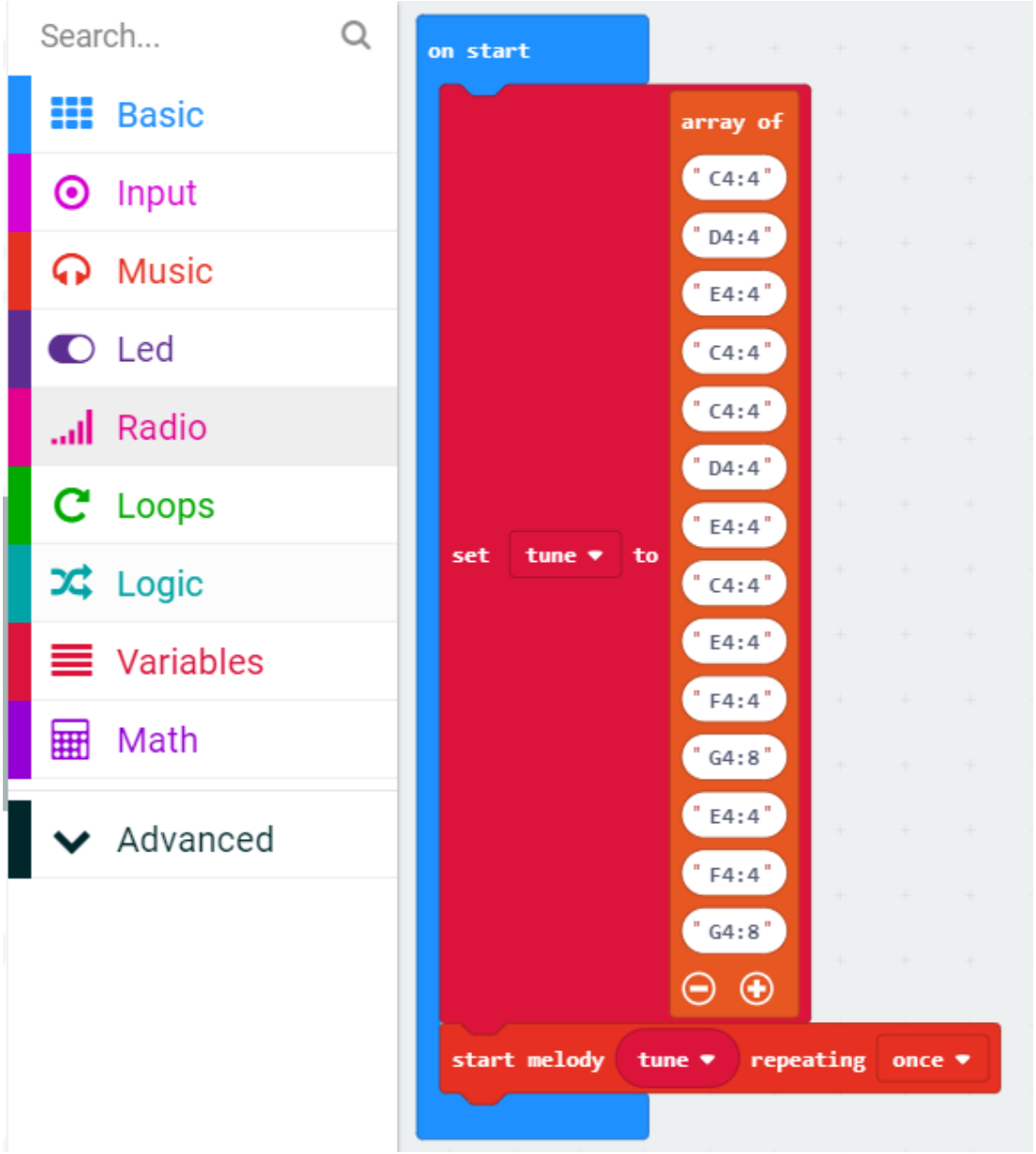


Block code

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/08.3_Play-a-custom-melody	Play-a-custom-melody.hex

After importing 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 custom melody.

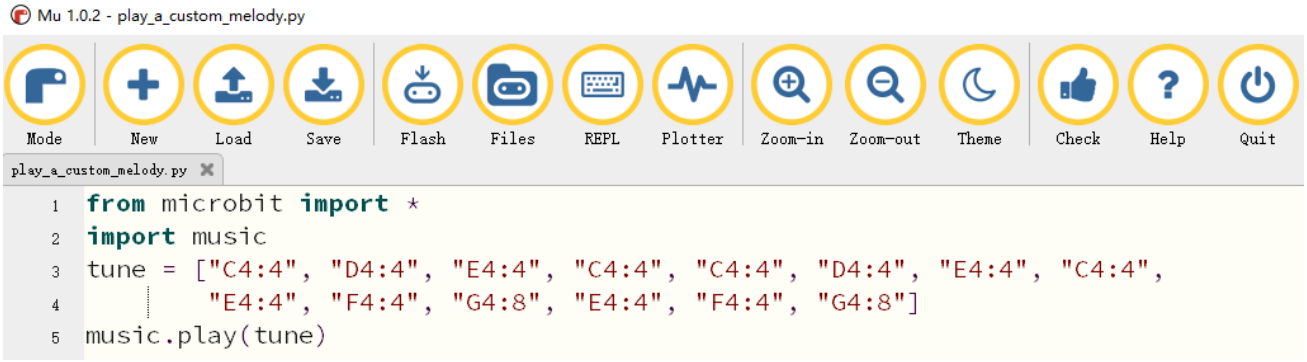
The tune array holds a custom melody, and each element in the array contains notes and beats. For example, "A1:4" refers to the note named A in octave number 1 to be played for a duration of 4.

Python code

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

File type	Path	File name
Python file	../Projects/PythonCode/08.3_Play-a-custom-melody	Play-a-custom-melody.py

After loading 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 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)
```

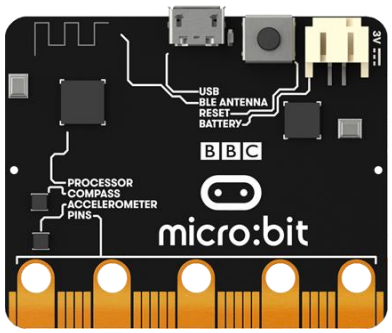
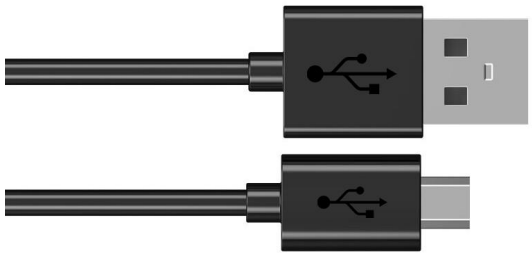
Chapter 9 Serial Communication

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

Project 9.1 Display the Data

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

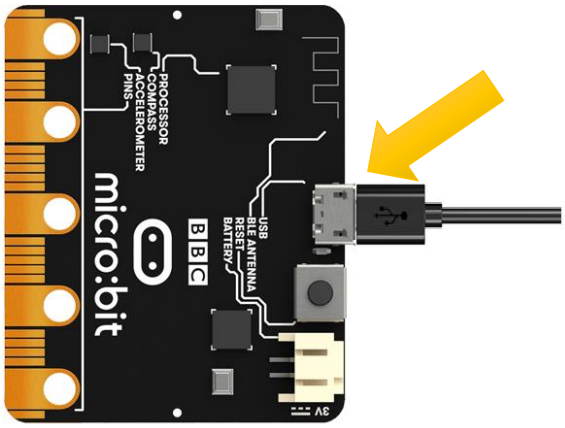
Component list

Microbit x1 	USB cable x1 

Circuit

Connect micro:bit and PC via a micro USB cable.

Hardware connection

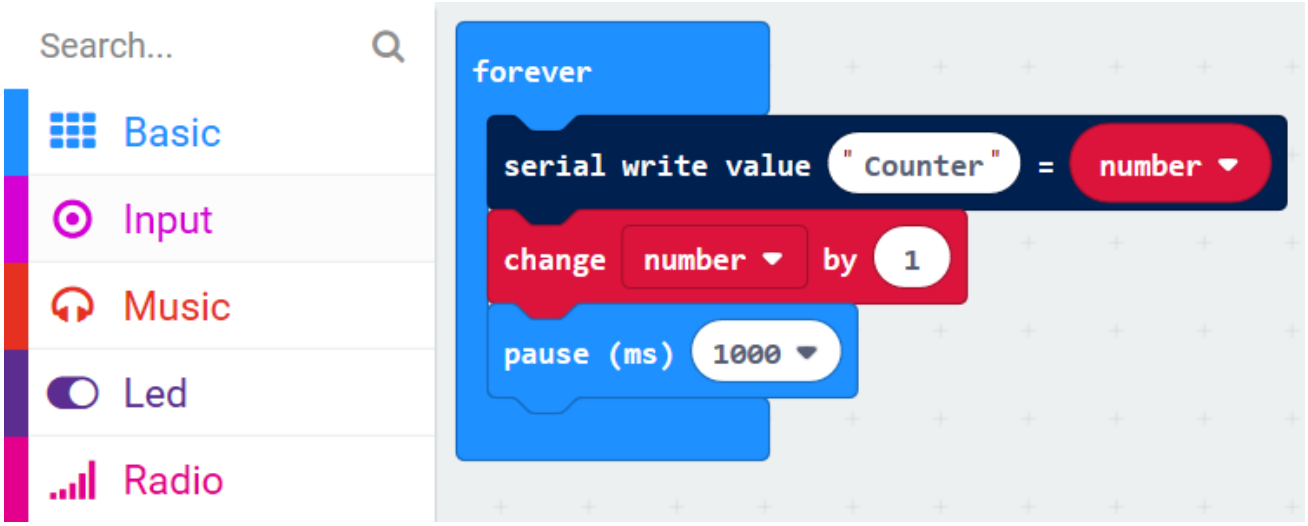


Block code

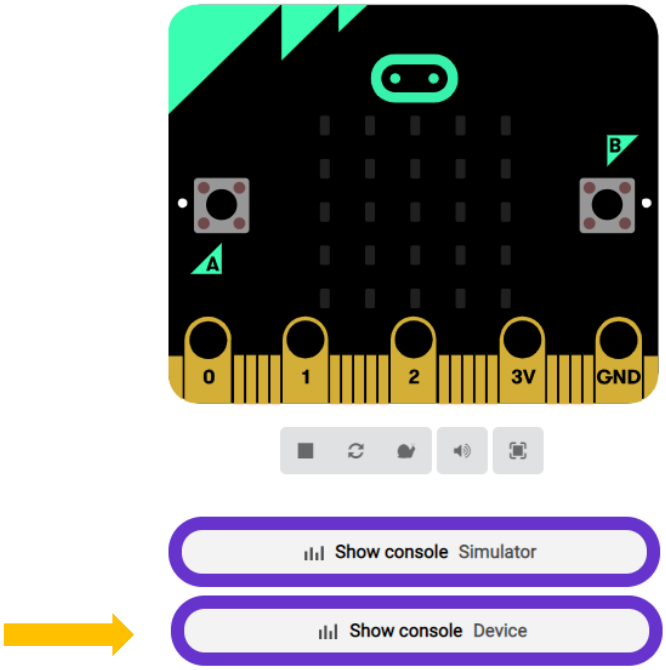
Open MakeCode first. Import the .hex file. The path is as below:
(How to import project)

File type	Path	File name
HEX file	../Projects/BlockCode/09.1_SerialPort	SerialPort.hex

After importing successfully, the code is shown as below:



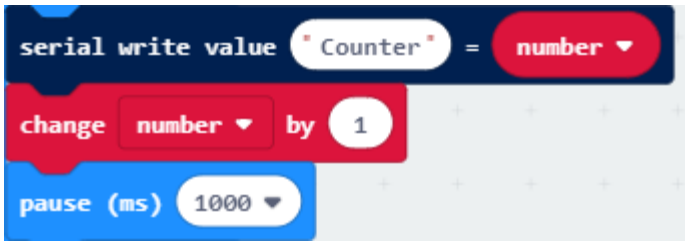
Check the connection of the circuit and verify it correct download the code into the micro:bit, and then open the serial controller, as shown below:



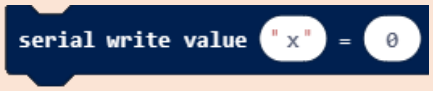
On the serial console, you can see the data sent by the microbit.


```
Counter:162
Counter:163
Counter:164
Counter:165
Counter:166
Counter:167
Counter:168
```

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

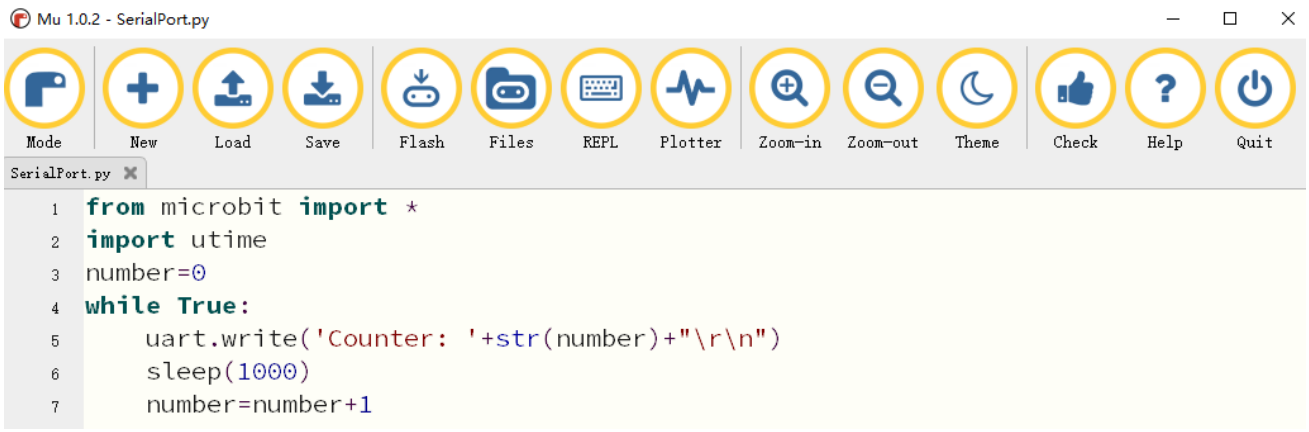
Block	Function
	Write a name:value pair and a newline character (\r\n) to the serial port.
	The change blocks increase the value in the variable by the amount you want. This is also known as an addition assignment operation.

Python code

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

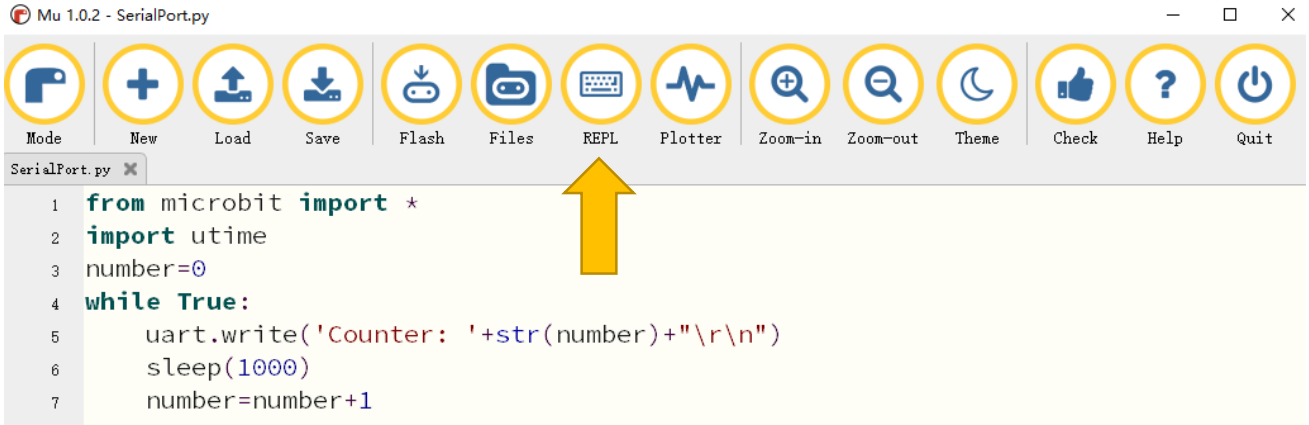
File type	Path	File name
Python file	../Projects/PythonCode/09.1_SerialPort	SerialPort.py

After loading successfully, the code is shown as below:

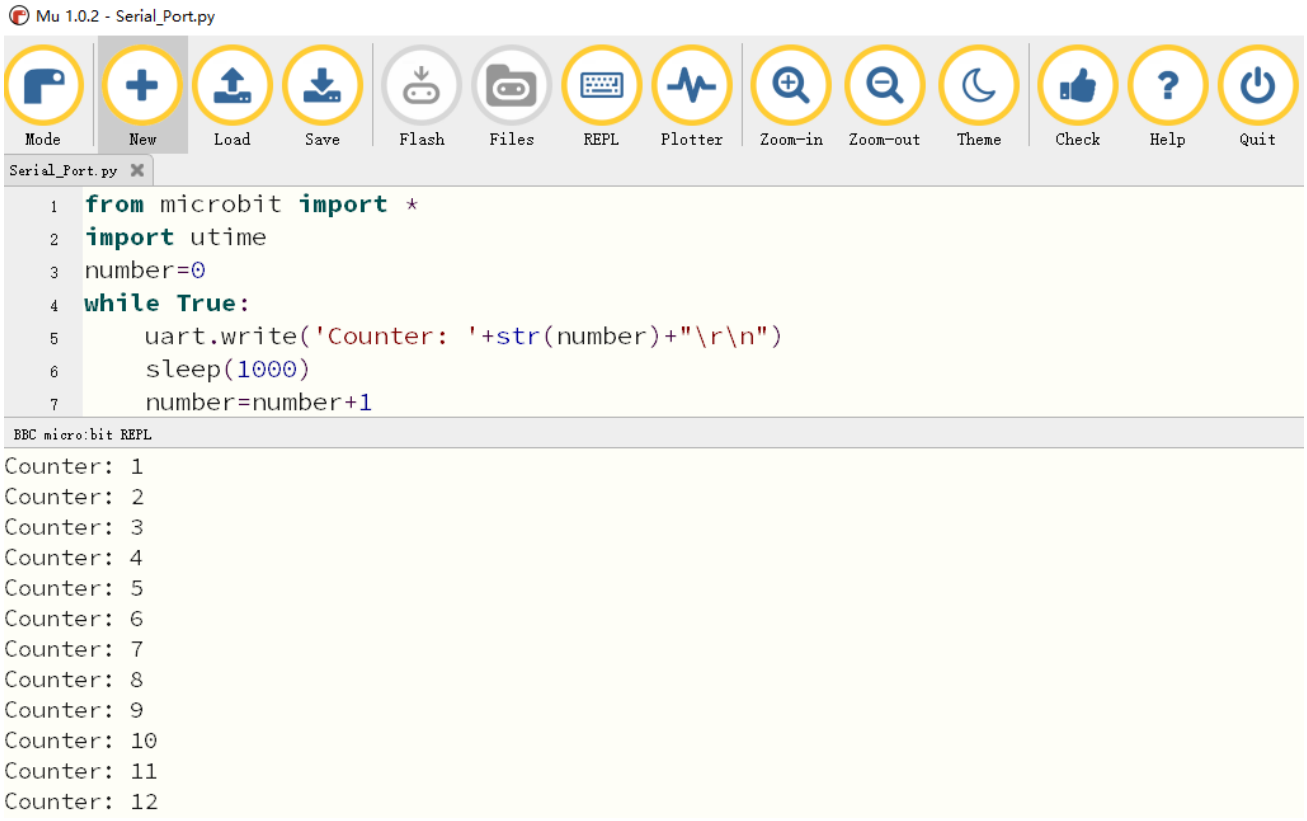


Check the connection of the circuit and verify it correct and then download the code into the micro:bit.

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



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



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

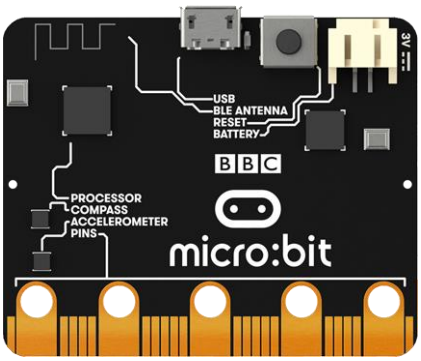
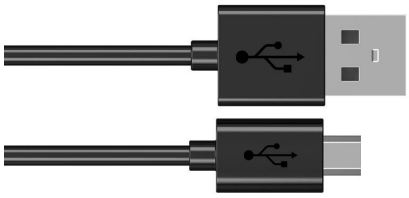
Chapter 10 Magnetometer

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

Project 10.1 Display Magnetometer Data

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

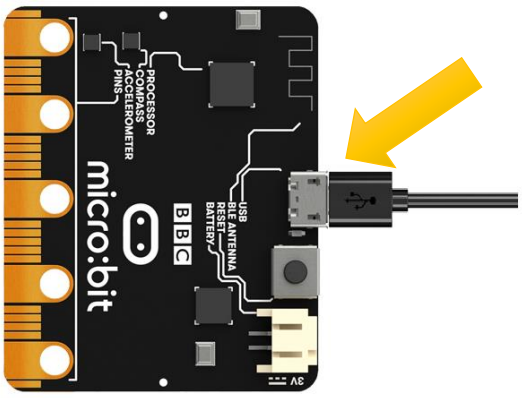
Component list

Microbit x1	USB cable x1
	

Circuit

Connect micro:bit and PC via micro USB cable.

Hardware connection



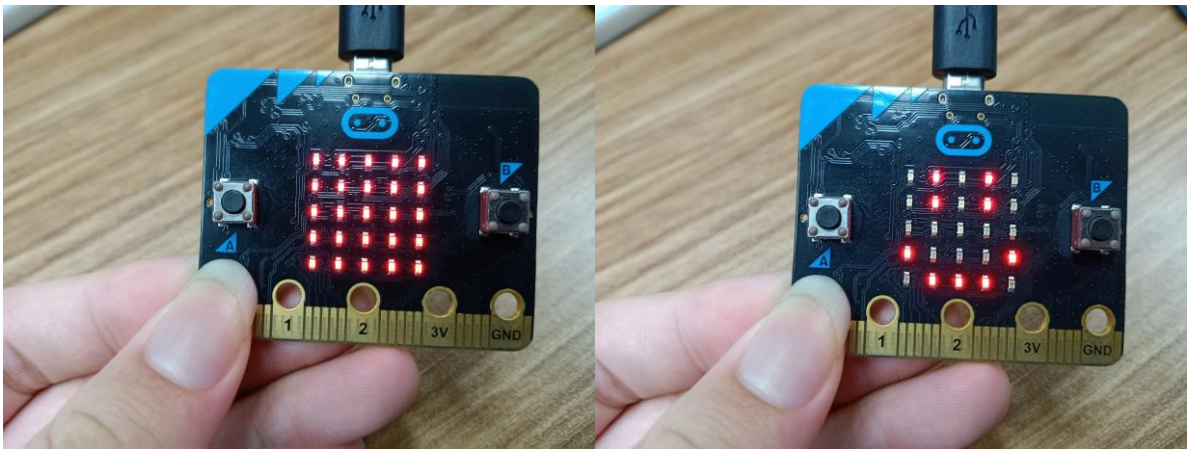
Block code

Open MakeCode first. Import the .hex file. The path is as below:
(How to import project)

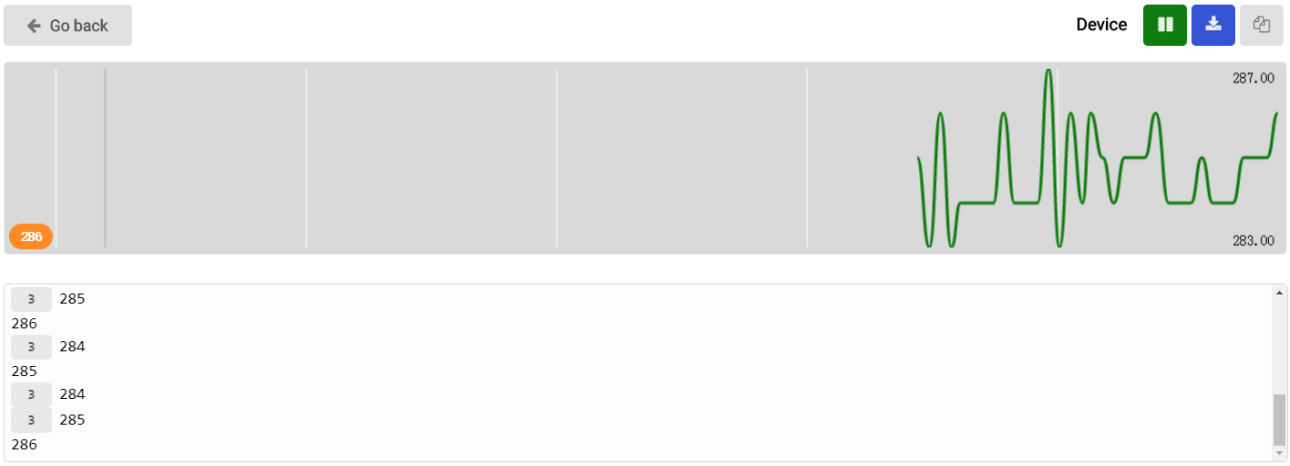
File type	Path	File name
HEX file	../Projects/BlockCode/10.1_DisplayMagnetometerData	DisplayMagnetometerData.hex

After importing successfully, the code is shown as below:

Check the connection of the circuit and verify it correct, and then download the code into the micro:bit. After completing downloading, 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 completed. Start the calibration process, a prompt will scroll on the LED matrix, which indicates that you need to rotate the micro:bit until **all** LEDs on the LED screen are illuminated, and then a smile is displayed which means the calibration is completed, as shown below:

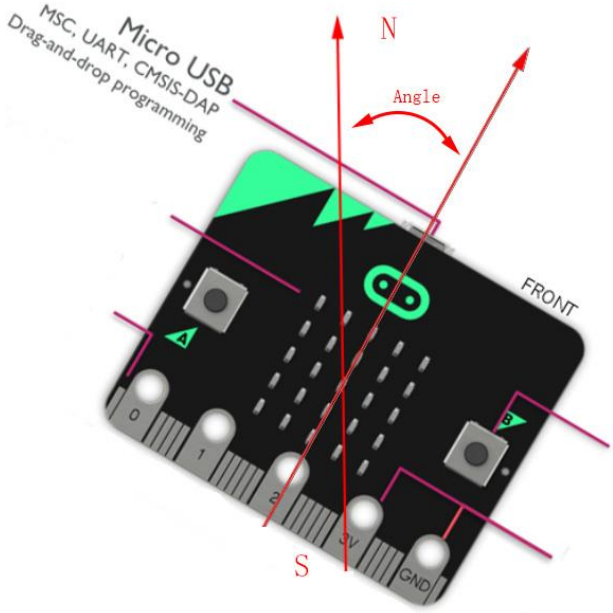


Then open the serial console (Open the Serial Port), place the micro:bit horizontally on the desktop, and rotate the micro:bit (clockwise or counterclockwise) to see the angular offset read from the magnetometer chip. As shown below:



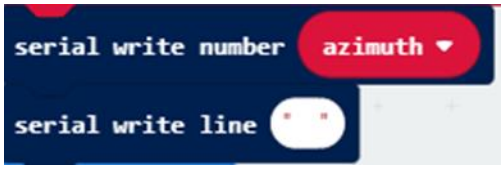
3 indicates the number of times of consecutive readings of the same value.

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

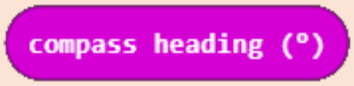


The angular offset read from the magnetometer chip is stored in the variable azimuth.

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



Reference

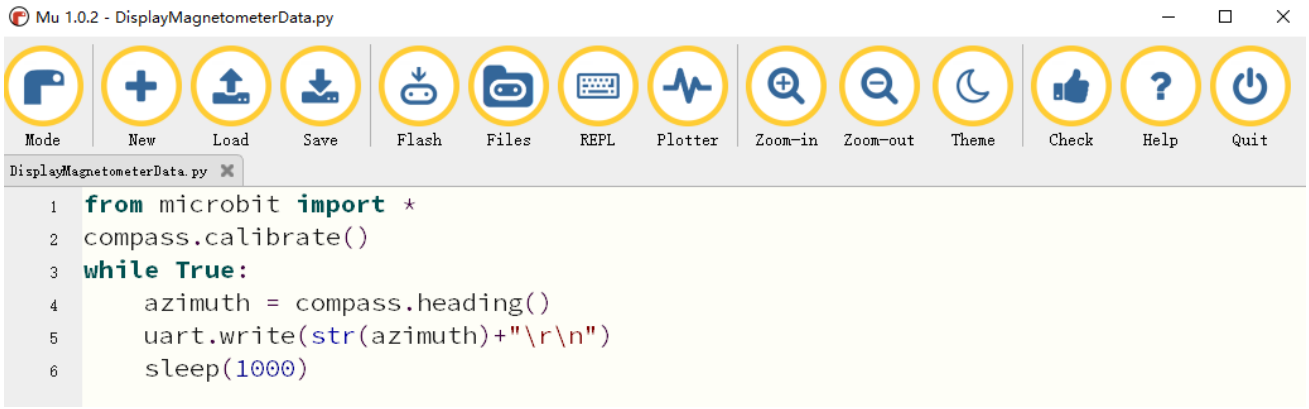
Block	Function
	Write a string to the serial port and start a new line of text by writing \r\n.
	Write a number to the serial port.
	The micro:bit measures the compass heading from 0 to 359 degrees with its magnetometer chip. Different numbers mean north, east, south, and west.

Python code

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

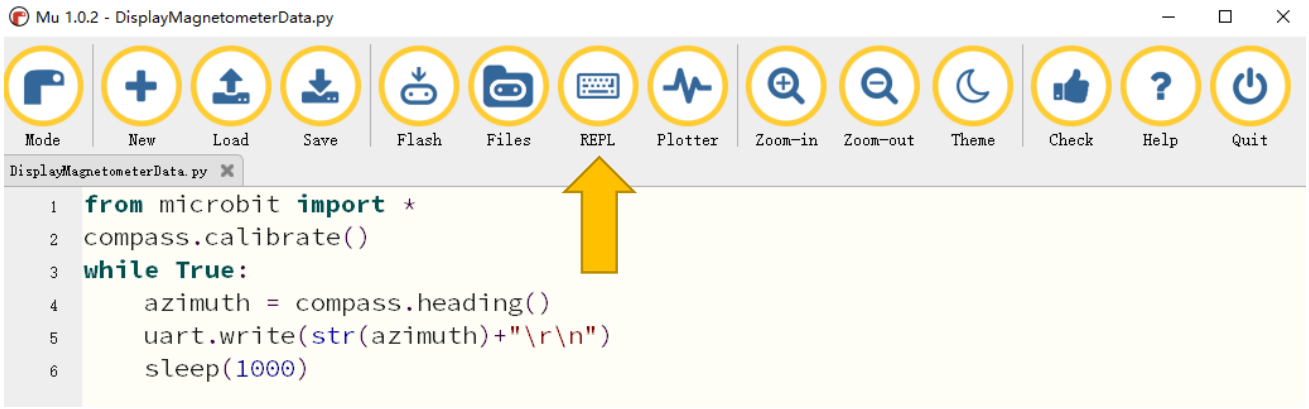
File type	Path	File name
Python file	../Projects/PythonCode/10.1_DisplayMagnetometerData	DisplayMagnetometerData.py

After loading successfully, the code is shown as below:



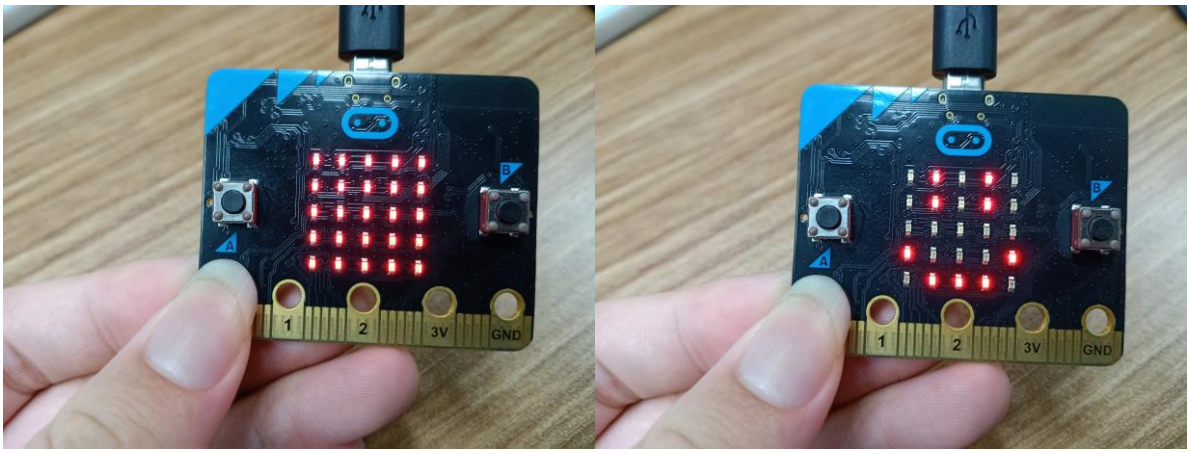
After checking the connection of the circuit and verify it correct, download the code into micro:bit.

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



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, a prompt will scroll on the LED matrix, which indicates that you need to rotate the micro:bit until **all** LEDs on the LED screen are illuminated, and then a smile is displayed which means the calibration is completed, 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

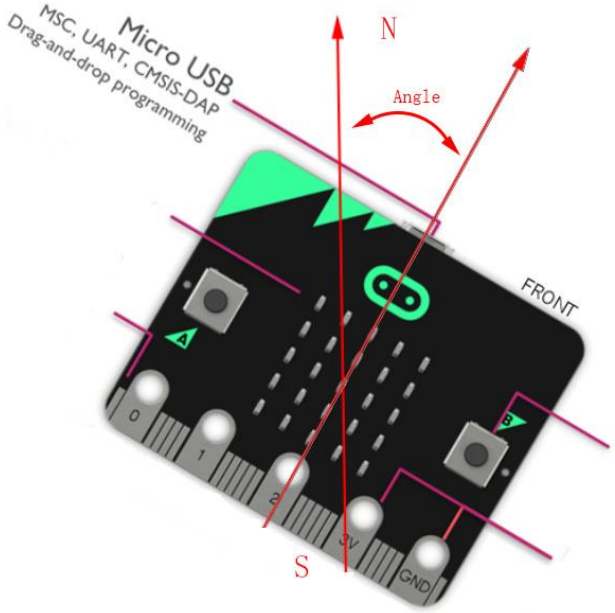
untitled compass.py

```
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 scroll on the LED matrix, which indicates that you need to rotate the micro:bit until all LEDs are illuminated.

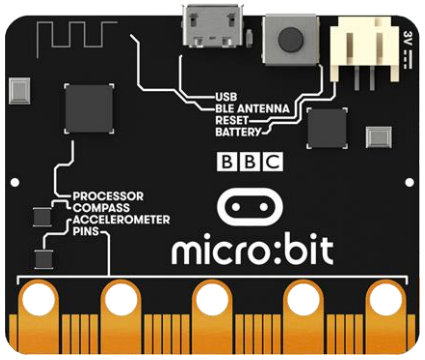
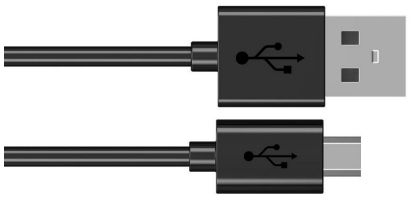
compass.heading()

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 10.2 Electronic Compass

In this project, we will use micro:bit to make an electronic compass, displaying an arrow on the micro:bit, and the arrow always points to the geographic north pole.

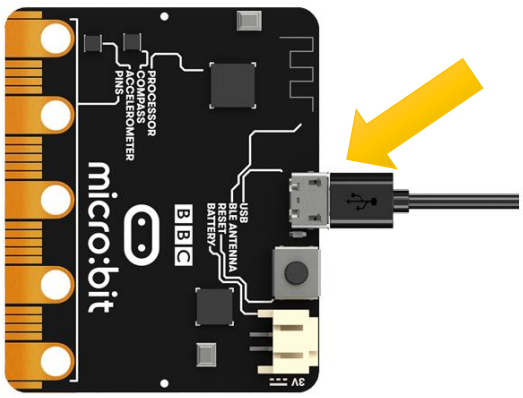
Component list

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/10.2_ElectronicCompass	ElectronicCompass.hex

After importing successfully, the code is shown as below:

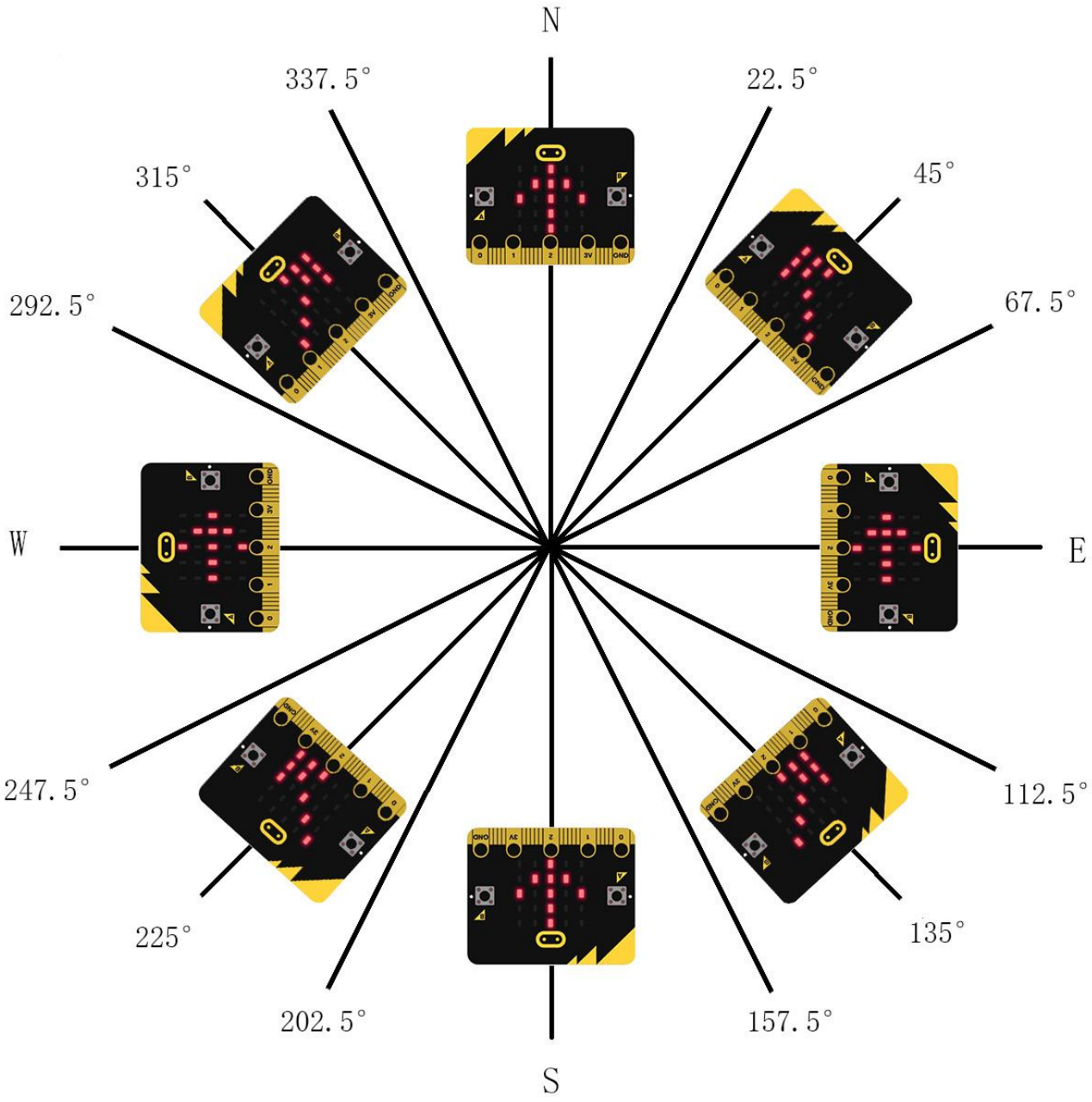
Check the connection of the circuit and verify it correct, and then download the code into the micro:bit. Calibrate the electronic compass. After the calibration is successful, place the micro:bit horizontally and turn the micro:bit to see that the arrow points to the geography Arctic.

The arrow will point to eight directions: 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 from the north to the northeast of the geography, the arrow shown should be reversed, that is, it rotates -45 degrees, pointing to the northwest of the micro:bit, which is the geographic north pole. Therefore, we can adjust the direction of the arrow according to its angular offset from the geographic North Pole.

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

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

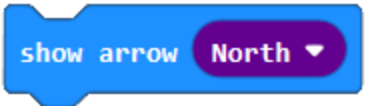
And so on in the same fashion, in every 45 degrees, the arrow points to a particular direction indicating the geographic north, as shown in the following illustration:



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

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

Reference

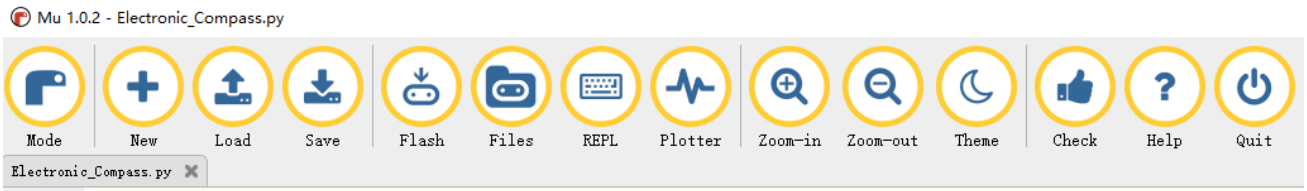
Block	Function
	Run code depending on whether a Boolean condition is true or false.
	Shows the selected arrow on the LED screen

Python code

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

File type	Path	File name
Python file	../Projects/PythonCode/10.2_ElectronicCompass	ElectronicCompass.py

After loading successfully, the code is shown as below:




```
from microbit import *
compass.calibrate()
while True:
    azimuth = compass.heading()
    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 or azimuth>337.5:
        display.show(Image.ARROW_N)
```

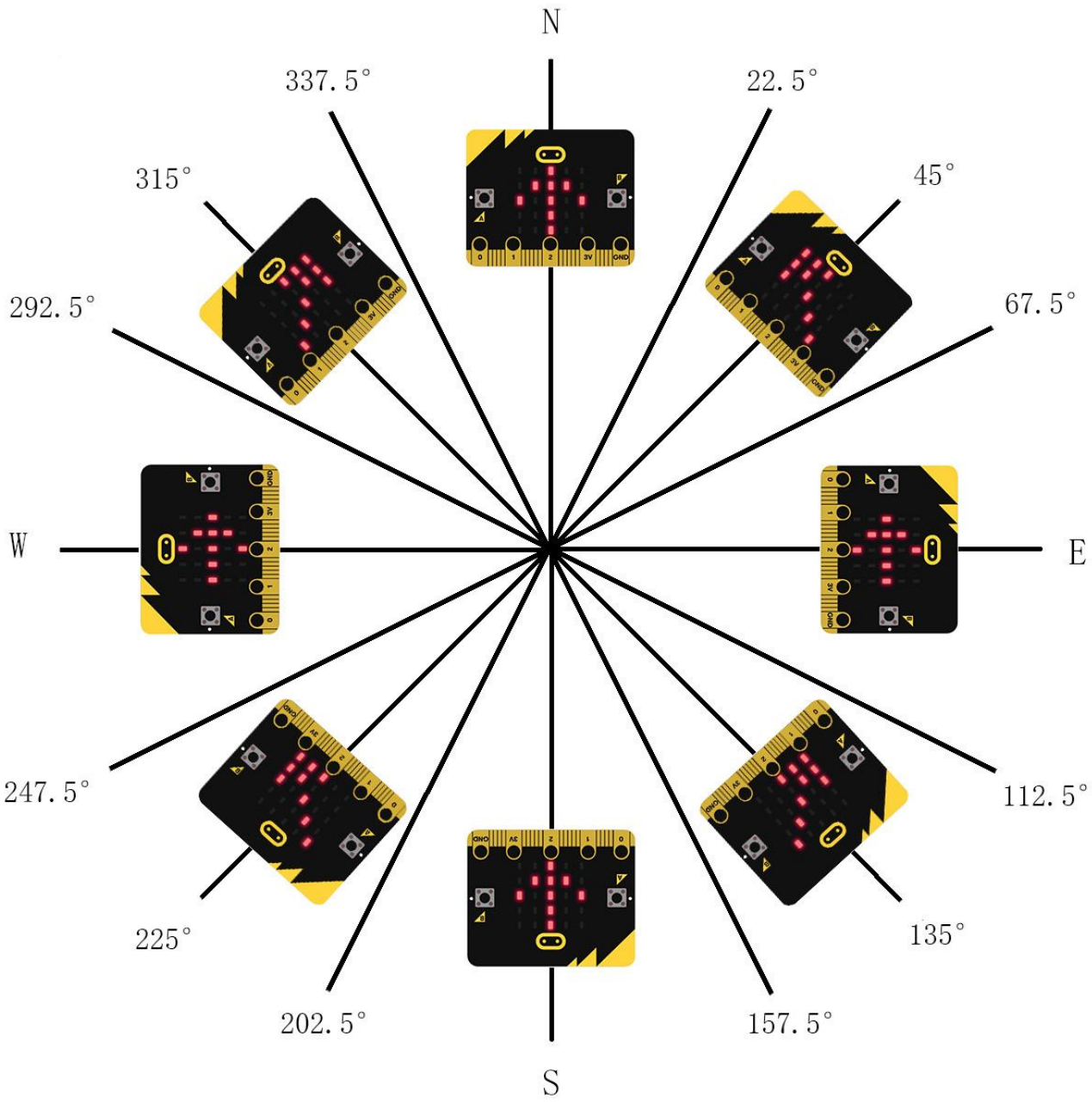
Check the connection of the circuit and verify it correct,, download the code into the micro:bit and calibrate the electronic compass. After the calibration is successful, place the micro:bit horizontally and rotate the micro:bit to see that the arrow points to the geography Arctic.

The arrow will point to eight directions:: 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 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, which is 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 or greater than 337.5, the arrow points to the true north of the micro:bit.

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

And so on in the same fashion, in every 45 degrees, the arrow points to a particular direction indicating the geographic north, as shown in the following figure:



The following is the program code:

```
from microbit import *
compass.calibrate()
while True:
    azimuth = compass.heading()
    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)
```

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)
```

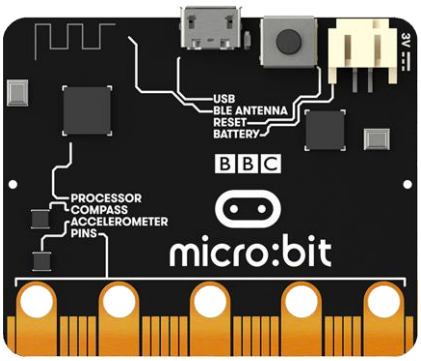
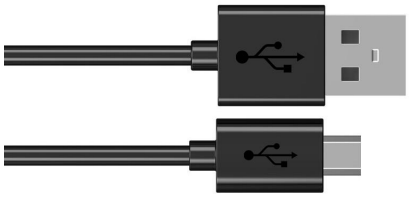
Chapter 11 Accelerometer

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

Project 11.1 Display Accelerometer Data

In this project, we will obtain data from the accelerometer sensor and print it on the serial console.

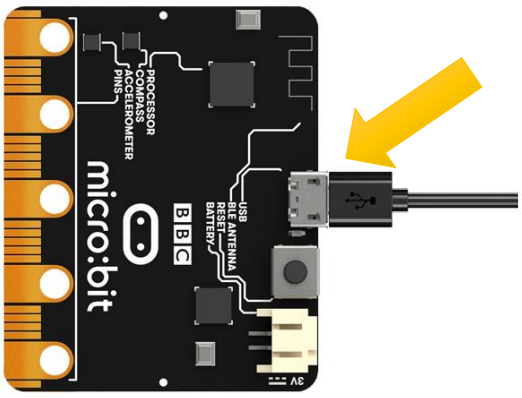
Component list

Microbit x1	USB cable x1
	

Circuit

Connect micro:bit and PC via a micro USB cable.

Hardware connection

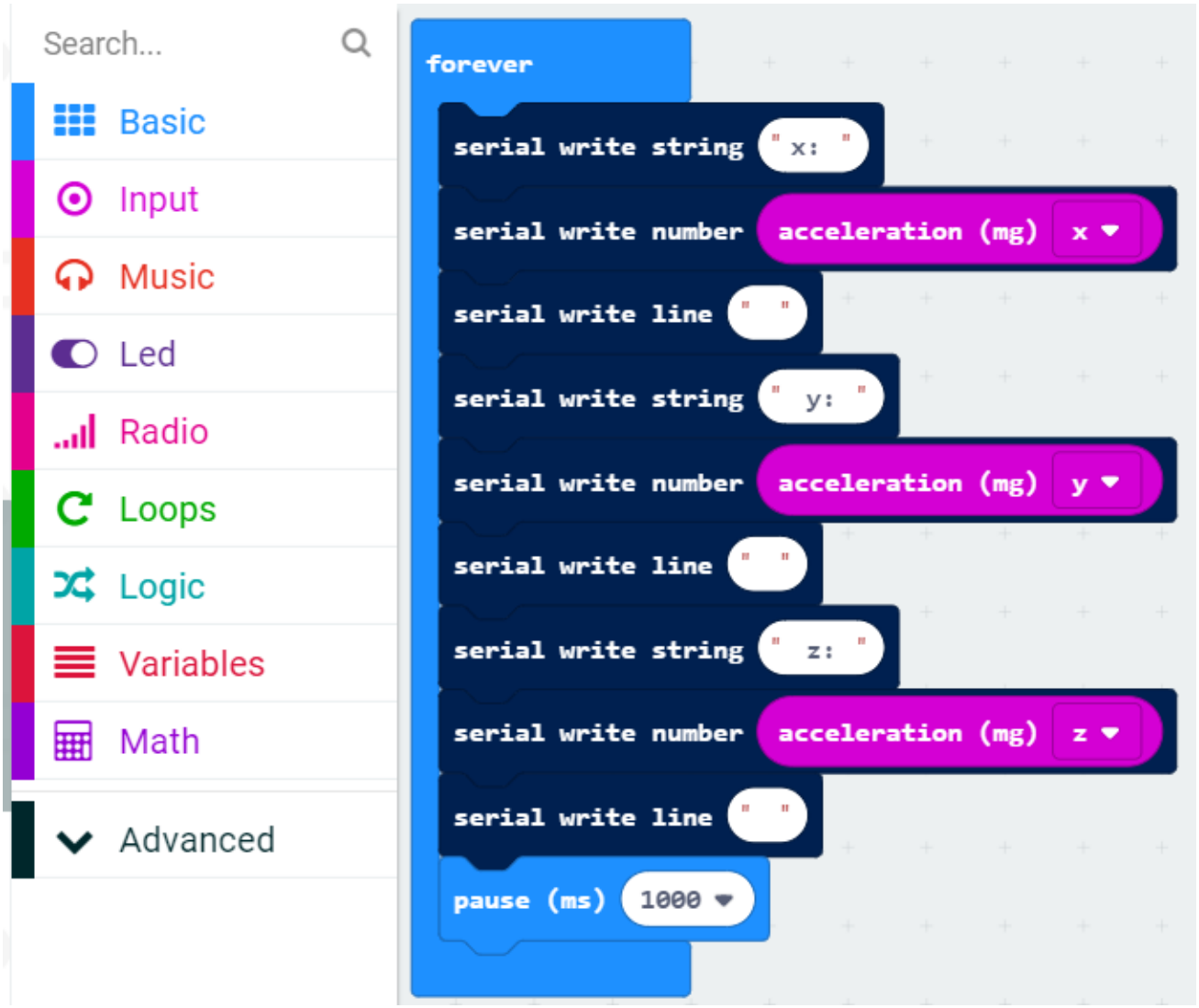


Block code

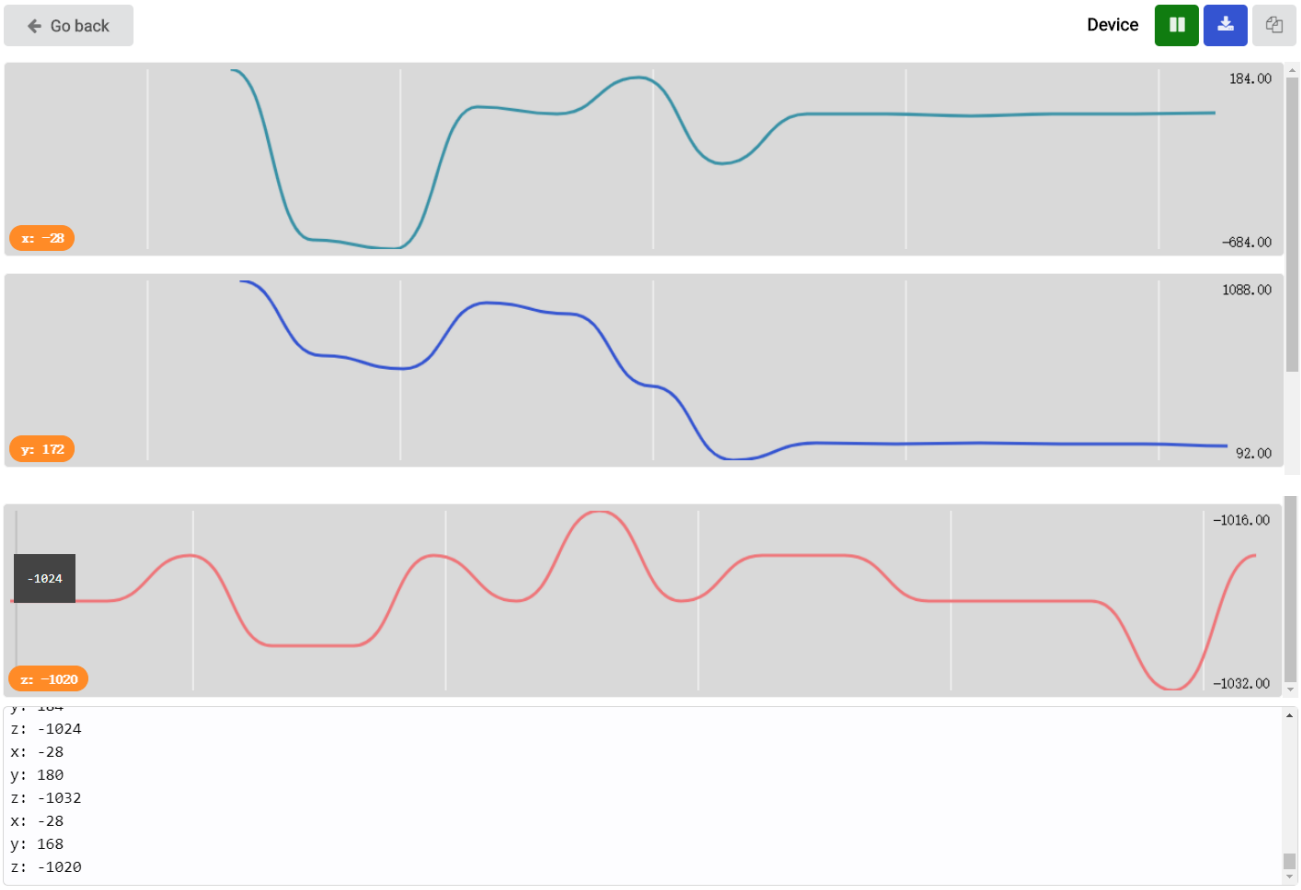
Open MakeCode first.
Import the .hex file. The path is as below:
(How to import project)

File type	Path	File name
HEX file	../Projects/BlockCode/11.1_DisplayAccelerometerData	DisplayAccelerometerData.hex

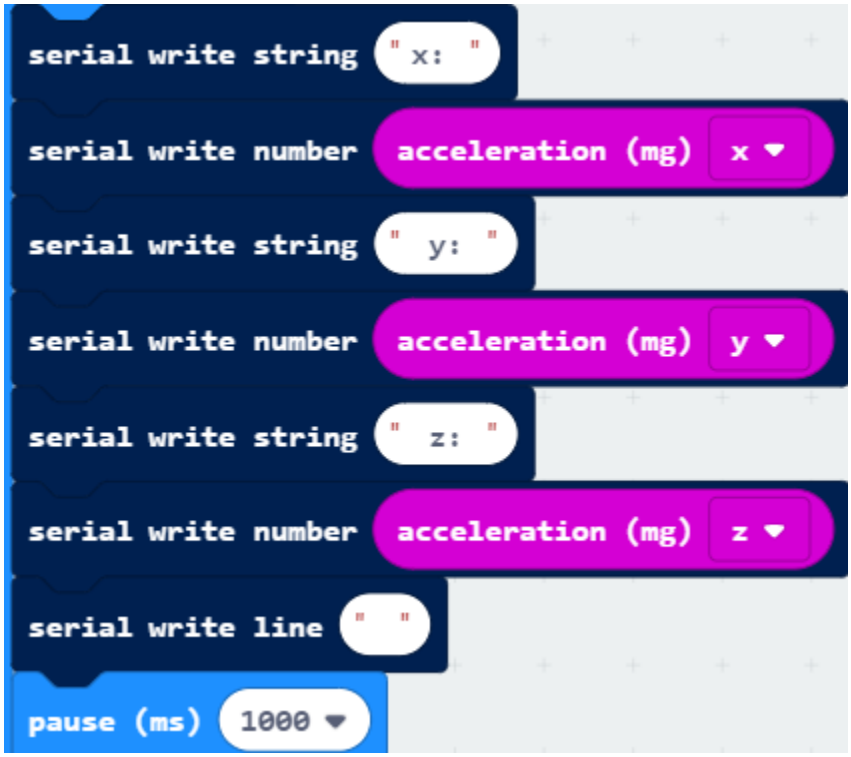
After importing successfully, the code is shown as below:



Check the connection of the circuit and verify it correct, 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 value in all directions (x, y, and z).

Python code

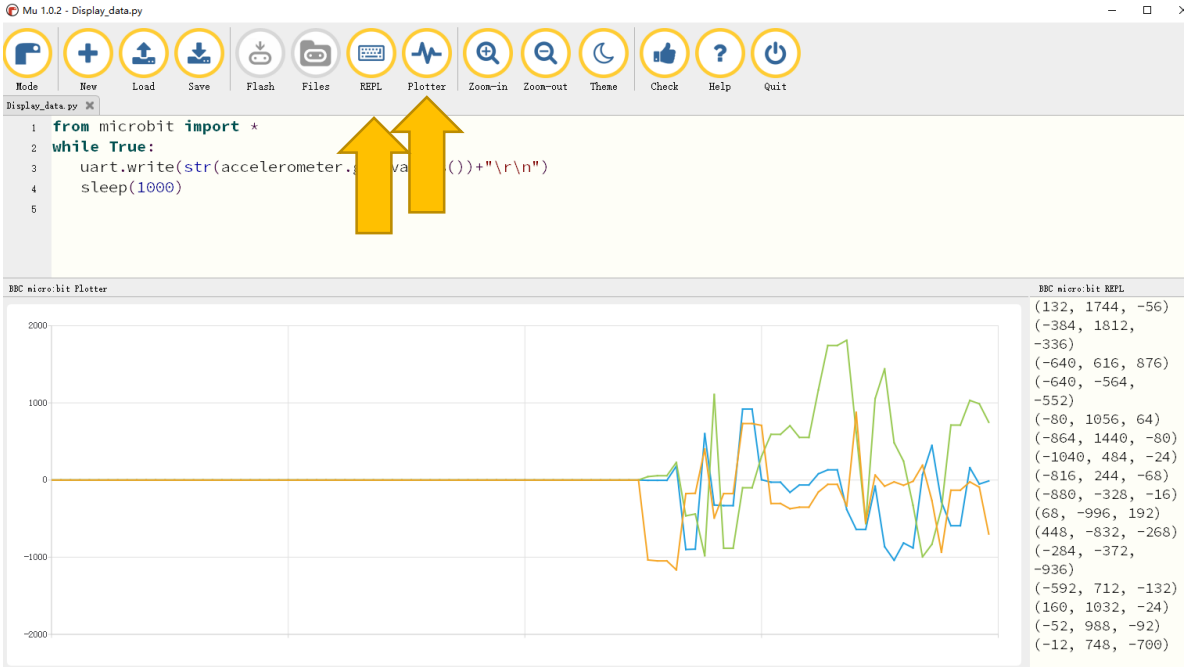
Open the .py file with Mu. Code, the path is as below:

File type	Path	File name
Python file	../PythonCode/11.1_DisplayAccelerometerData	DisplayAccelerometerData.py

After loading successfully, the code is shown as below:

Check the connection of the circuit and verify it correct, and then 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:

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

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

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

Reference

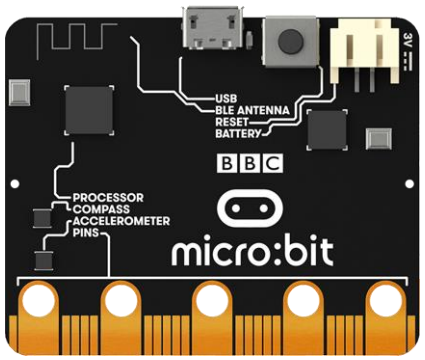
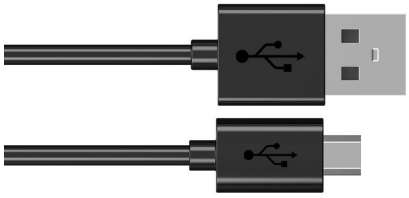
accelerometer.get_values()

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, so X, Y, and Z will be within the range of +/-2000mg.

Project 11.2 Gradiometer

In this project, we will use the accelerometer to make a level instrument.

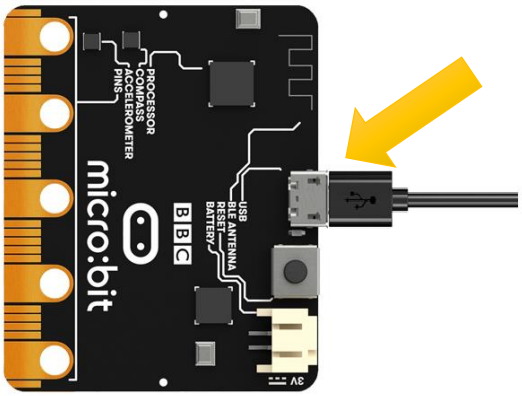
Component list

Microbit x1	USB cable x1
	

Circuit

Connect micro:bit and PC via a micro USB cable.

Hardware connection

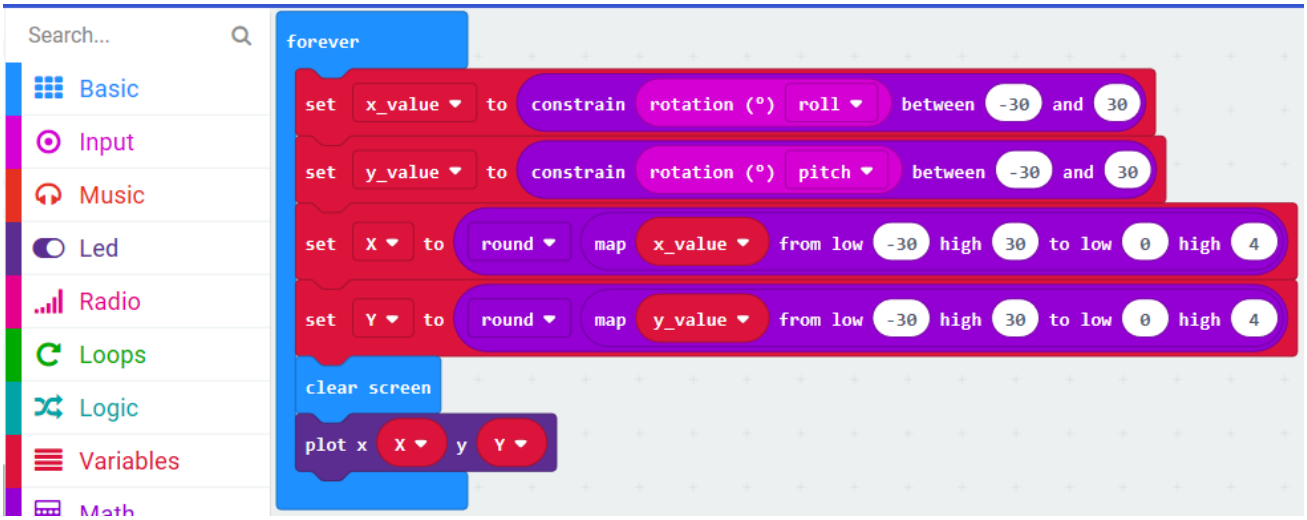


Block code

Open MakeCode first.
Import the .hex file. The path is as below:
(How to import project)

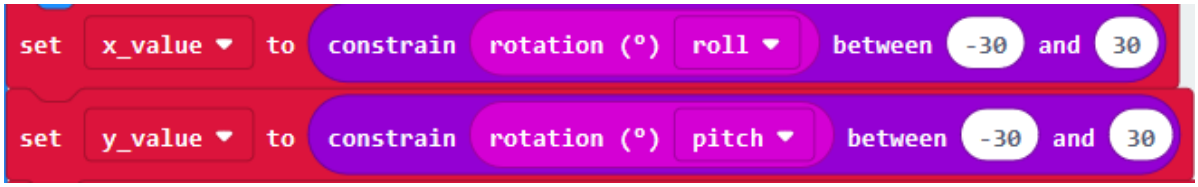
File type	Path	File name
HEX file	../Projects/BlockCode/11.2_Gradienter	Gradienter.hex

After importing successfully, the code is shown as below:



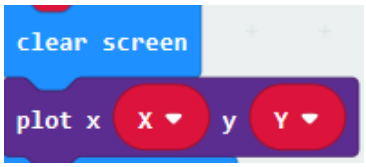
Check the connection of the circuit and verify it correct and download the code into micro:bit, you will observe that the LED dot matrix will change with the tilt of micro:bit.

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 require such a wide range of flip angles, so we just set it within -30 to 30 degrees.

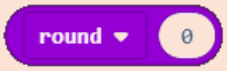
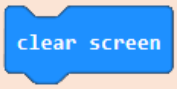


Since the LED screen is 5x5, map the range of -30-30 to the range of 0-4, and assign it to the X, Y variable.

Turn OFF all the LED first, then turn ON the corresponding LED according to the value of the X, Y variables.



Reference

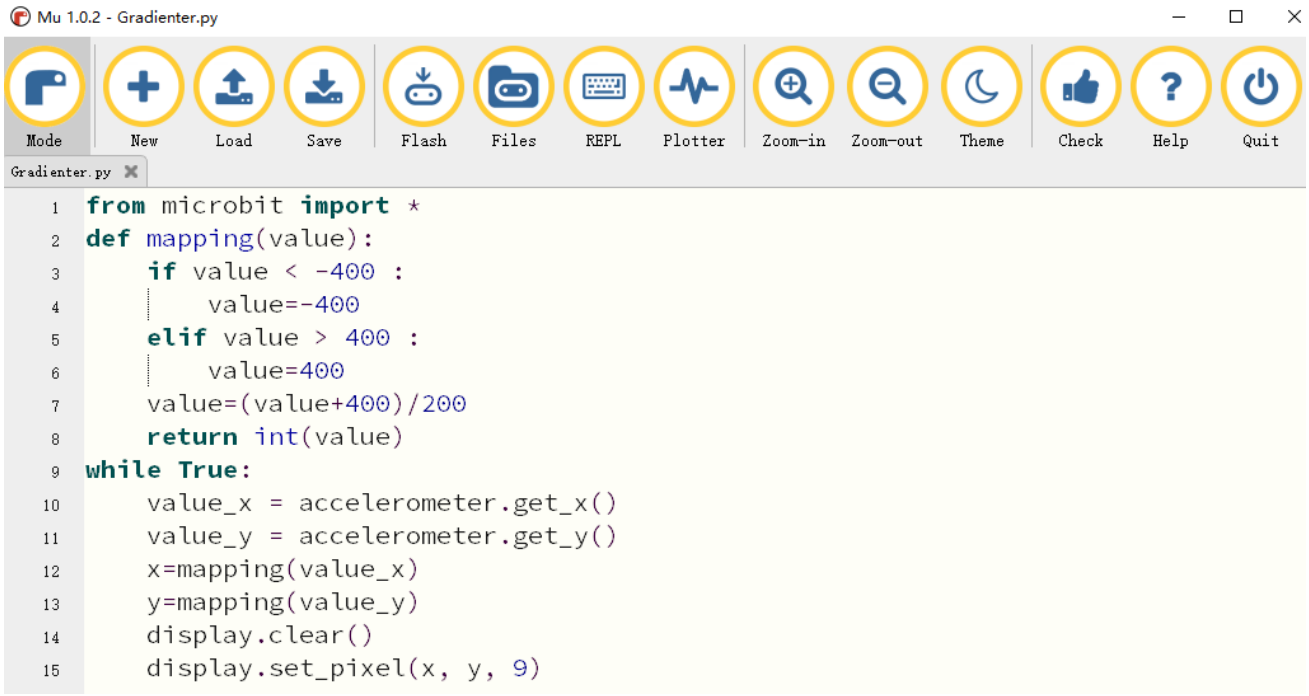
Block	Function
	Find how much the micro:bit is tilted in different directions.
	Turn ON the LED you set on the LED screen.
	If a number has a fractional part, you can change the number to the nearest integer value.
	A map is a conversion of one span of numbers to another.
	Turn OFF all the LED lights on the LED screen.
	Make sure that the value of the number you give is within the range.

Python code

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

File type	Path	File name
Python file	../Projects/PythonCode/11.2_Gradienter	Gradienter.py

After loading successfully, the code is shown as below:



Check the connection of the circuit and verify it correct, 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 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 require such a wide range, So we set it to the range of -400to 400. Call the mapping() function to return the value ranging from 0-4 , 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 value of the LED at column x and row y, 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 a value 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 a value within the range of +/- 2000mg.

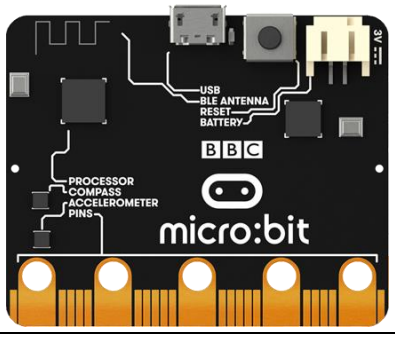
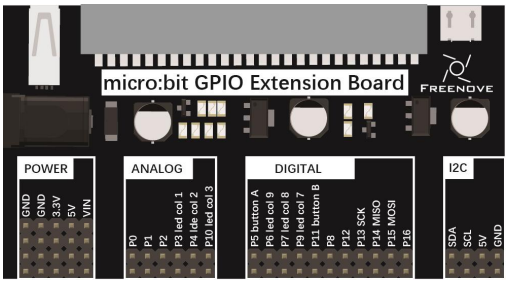
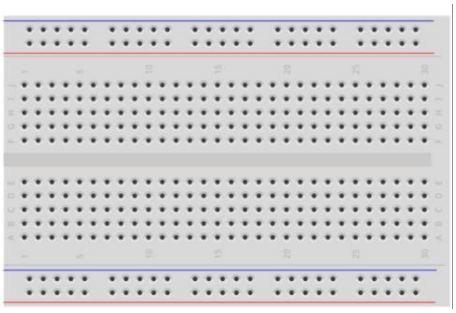
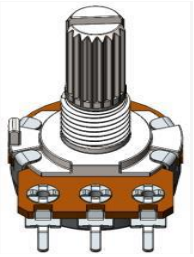
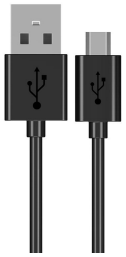
Chapter 12 Potentiometer

In this chapter, we will learn a new component: potentiometer

Project 12.1 Potentiometer

This project enables a rotatary potentiometer to output different voltages.

Component list

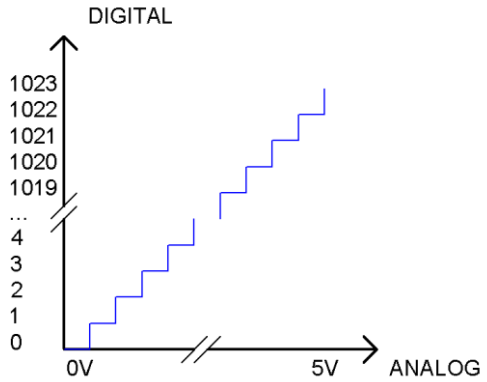
Microbit x1 	Extension board x1 		
Breadboard x1 	Potentiometer x1 	USB cable x1 	F/M x3 

Component knowledge

ADC

An ADC is an electronic integrated circuit used to convert analog signals such as voltages to digital or binary form consisting of 1s and 0s. The range of our ADC module is 10 bits, that means the resolution is 2^10=1024, so that its range (at 3.3V) will be divided equally to 1024 parts.

Any analog value can be mapped to one digital value using the resolution of the converter. So the more bits the ADC has, the denser the partition of analog will be and the greater the precision of the resulting conversion.

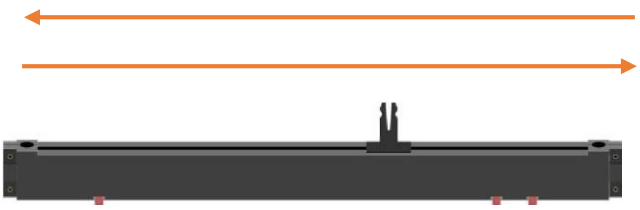


Subsection 1: the analog in rang of 0V-3.3/1024V corresponds to digital 0;
Subsection 2: the analog in rang of 3.3 /1024V-2*3.3/1024V corresponds to digital 1;
...

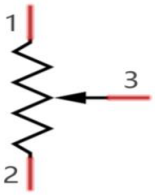
The resultant analog signal will be divided accordingly.

Potentiometer

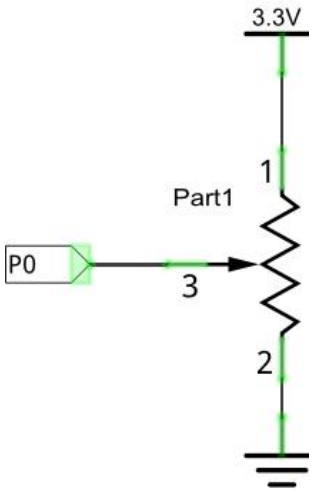
Potentiometer is a resistive element with three Terminal parts. Unlike the resistors that we have used thus far in our project which have a fixed resistance value, the resistance value of a potentiometer can be adjusted. A potentiometer is often made up by a resistive substance (a wire or carbon element)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



1 32

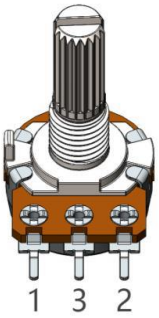


Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly (until it reaches the highest value of the resistive element) and at the same time the resistance between pin 2 and pin 3 will decrease linearly and conversely down to zero. At the midpoint of the slider the measured resistance values between pin 1 and 3 and between pin 2 and 3 will be the same.
In a circuit, both sides of resistive element are often connected to the positive and negative electrodes of power. When you slide the brush "pin 3", you can get variable voltage within the range of the power supply.



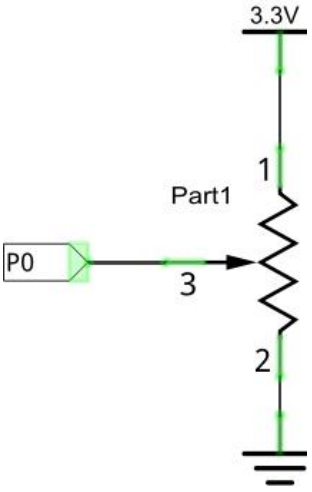
Rotary potentiometer

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

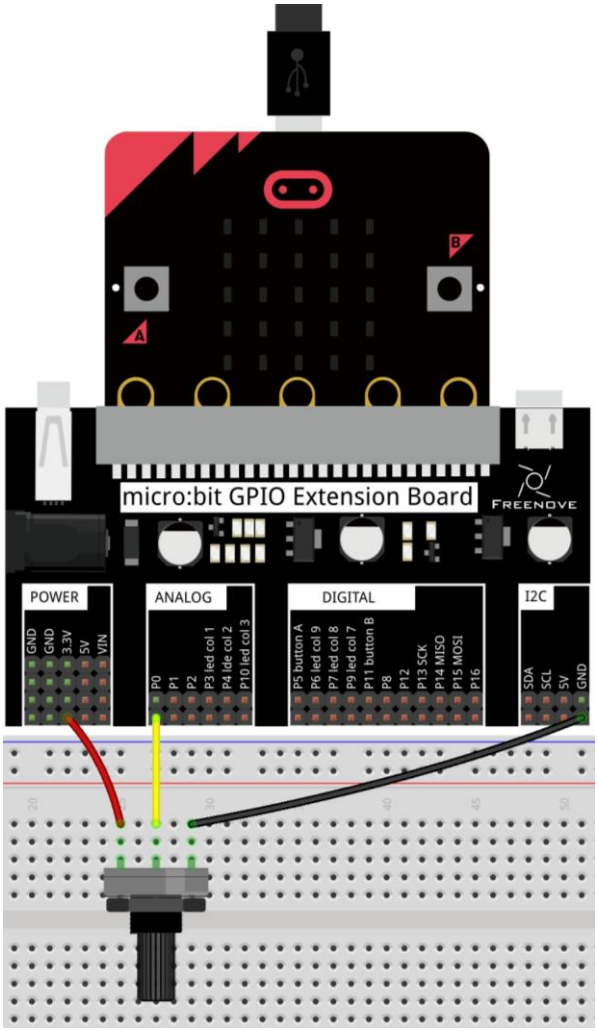


Circuit

Schematic diagram



Hardware connection



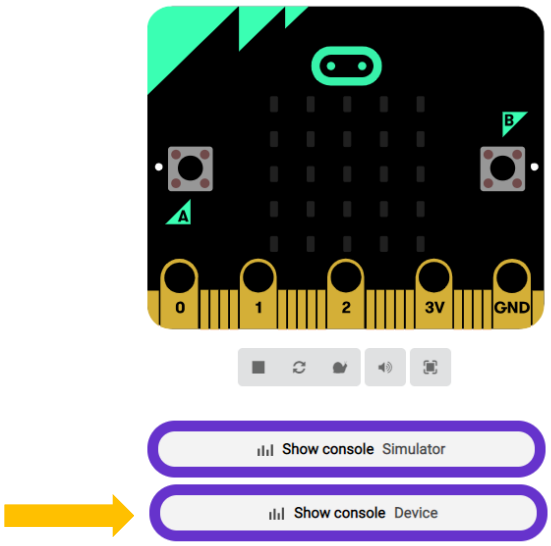
Block code

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_Potentiometer	Potentiometer.hex

After importing successfully, the code is shown as below:

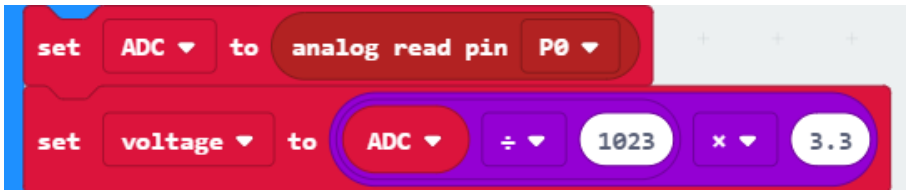
Check the connection of the circuit and verify it correct and then download the code into micro:bit.



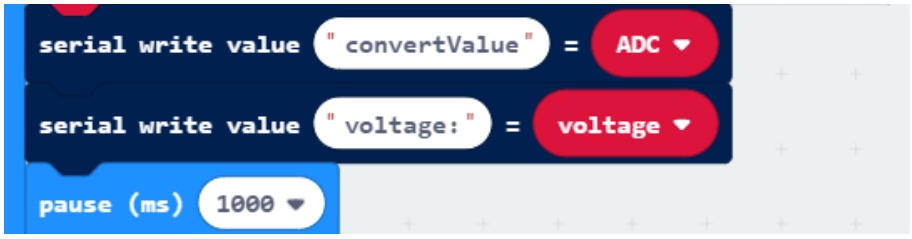
Click on the console device, rotate the potentiometer, you will see the output ADC value and voltage value.

```
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

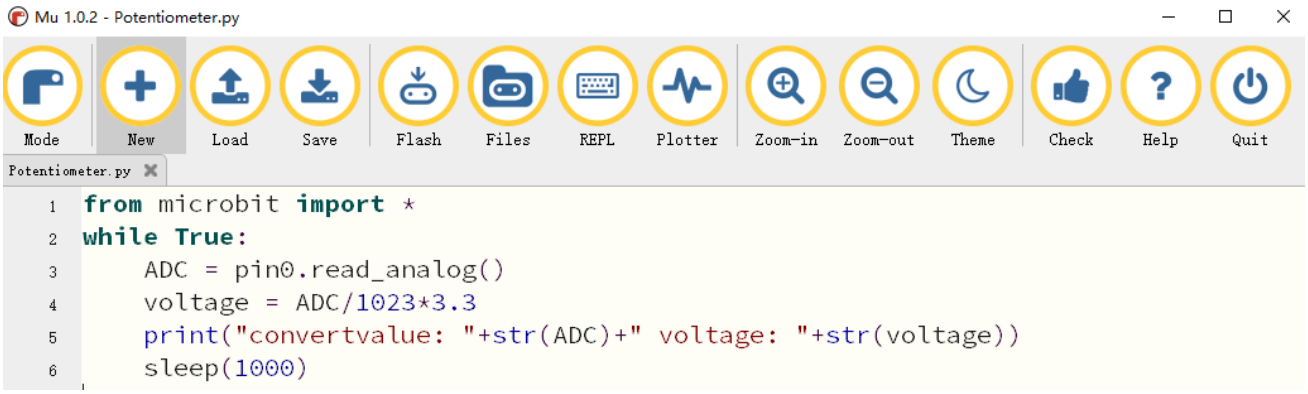
Block	Function
	Read an analog signal (0 to 1023) from the pin you set.)

Python code

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

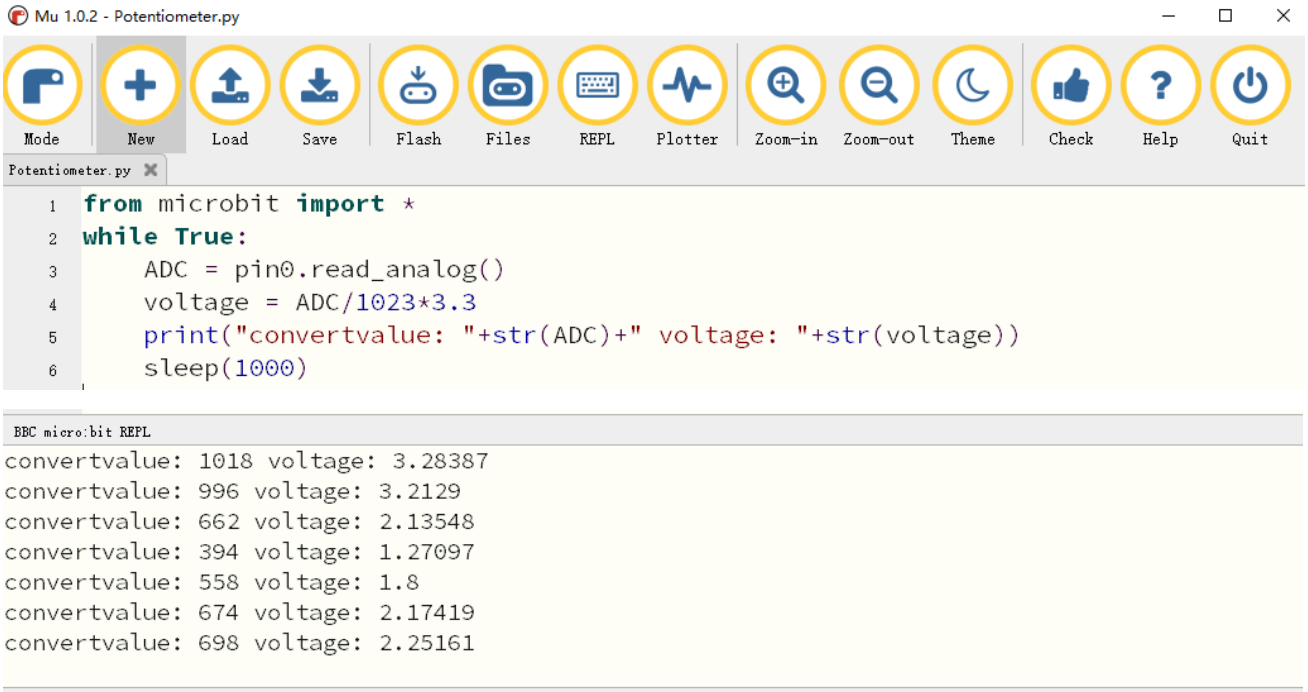
File type	Path	File name
Python file	../Projects/PythonCode/12.1_Potentiometer	Potentiometer.py

After loading 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. (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.



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 to 1023) from the pin you set.
print()
Print() is a Python built-in function for printing.

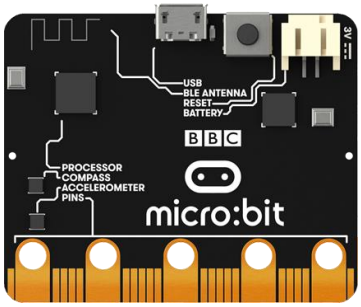
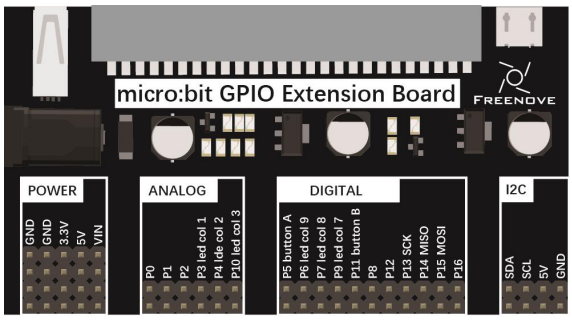
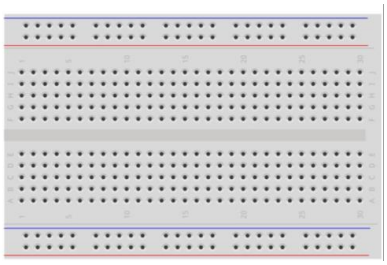
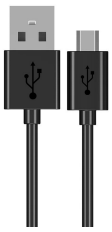
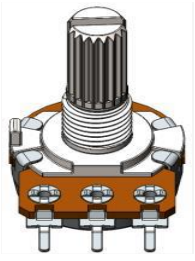
Chapter 13 Potentiometer and LED

This chapter is a comprehensive application of potentiometer and LED.

Project 13.1 Soft Light

In this project, we will make an LED with adjustable brightness.

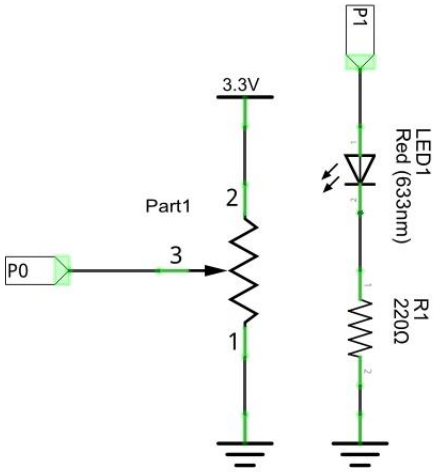
Component list

Microbit x1 	Extension board x1 	
Breadboard x1 	USB cable x1 	F/M x4 M/M x1 
LED x1 	Resistor 220Ω x1 	Potentiometer x1 

Circuit

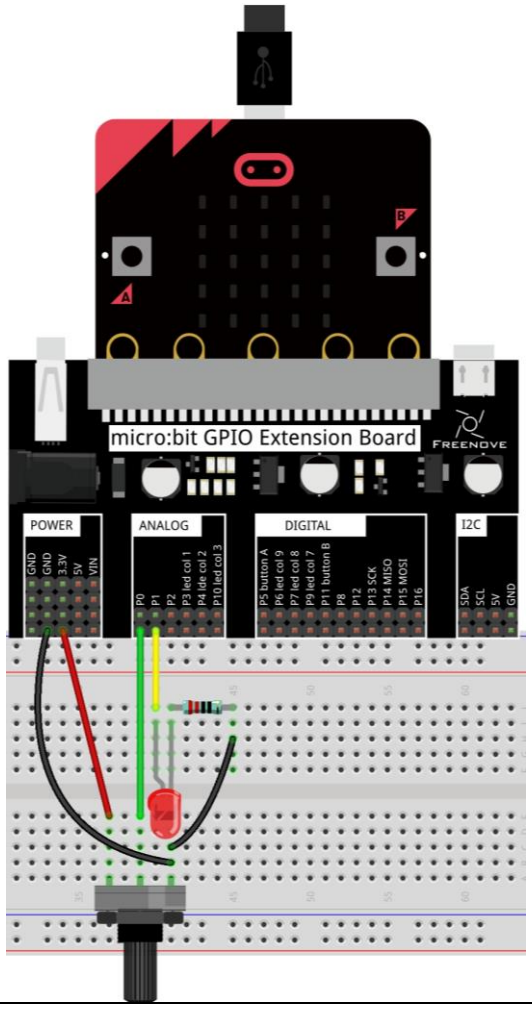
In this circuit, the port 1 and 2 of the potentiometer are respectively connected to the two ends of the power supply, and the port3 is connected to the P0 pin of the micro:bit.

Schematic diagram



Hardware connection

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

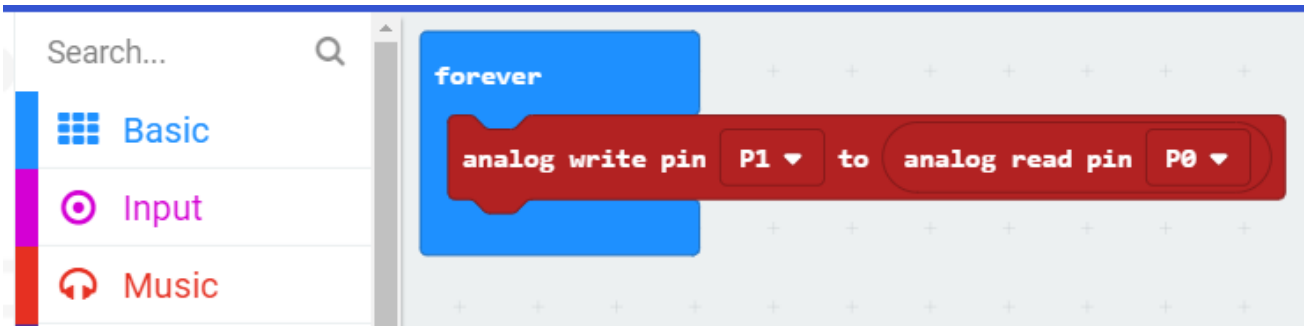


Block code

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/13.1_SoftLight	SoftLight.hex

After importing successfully, the code is shown as below:



Check the connection of the circuit, verify it correct,, download the code into the micro:bit, and rotate the potentiometer to see the change of the brightness.

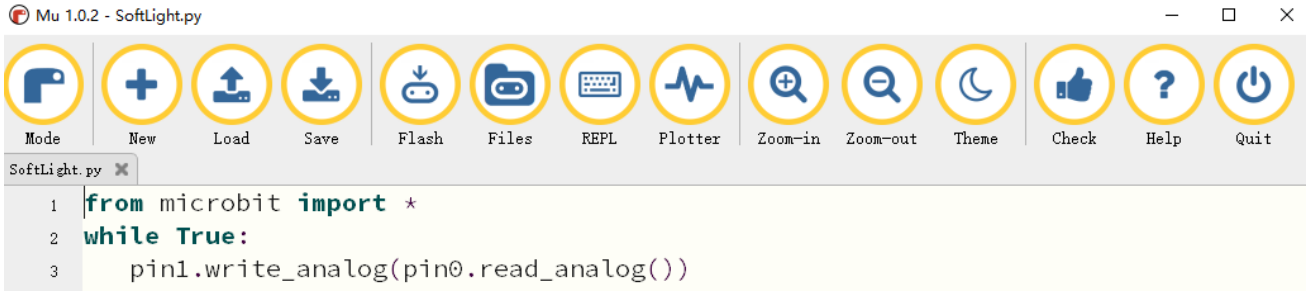
Read the analog voltage value of the P0 pin, then the P1 pin outputs the same analog voltage value.

Python code

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

File type	Path	File name
Python file	../Projects/PythonCode/13.1_SoftLight	SoftLight.py

After load successfully, the code is shown as below:



Check the connection of the circuit, verify it correct, download the code into the micro:bit, and rotate the potentiometer to see the change of the brightness of the LED.

The following is the program code:

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

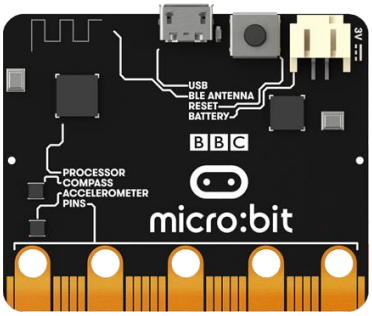
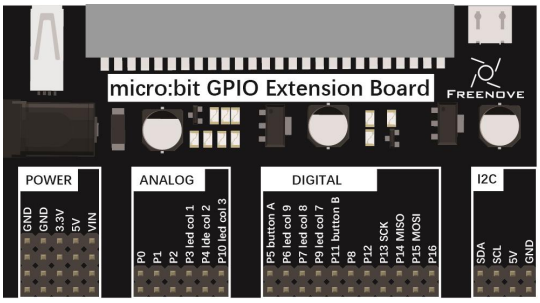
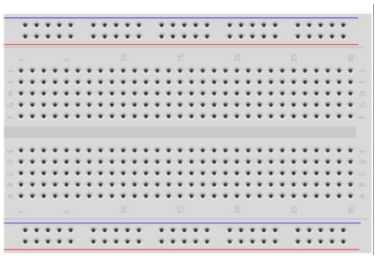
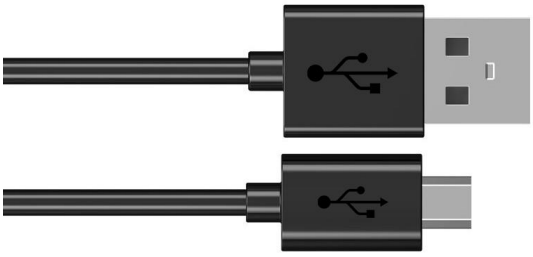
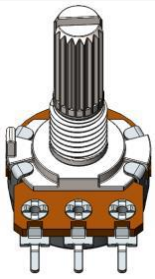
Read the analog voltage value of the P0 pin, then the P1 pin outputs the same analog voltage value.

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

Project 13.2 Multicolored Soft Light

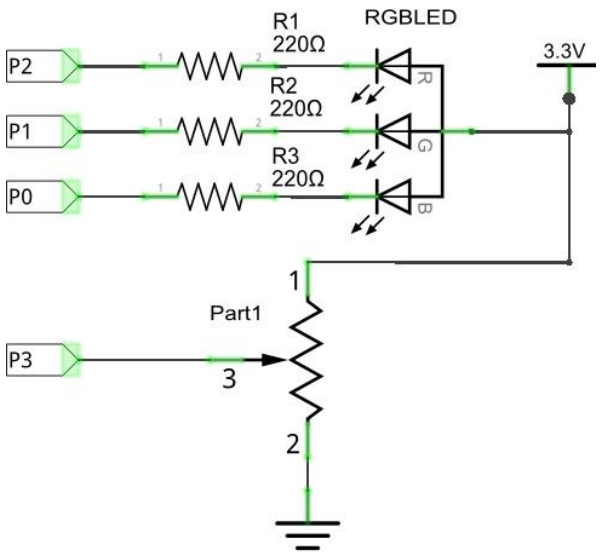
In this project, we control the color of the RGBLED with a potentiometer.

Component list

Microbit x1 	Extension board x1 
Breadboard x1 	USB cable x1 
Rotary potentionmeter x1 	RGB_LED x1 
F/M x6 M/M x1 	Resistor 220Ω x3 

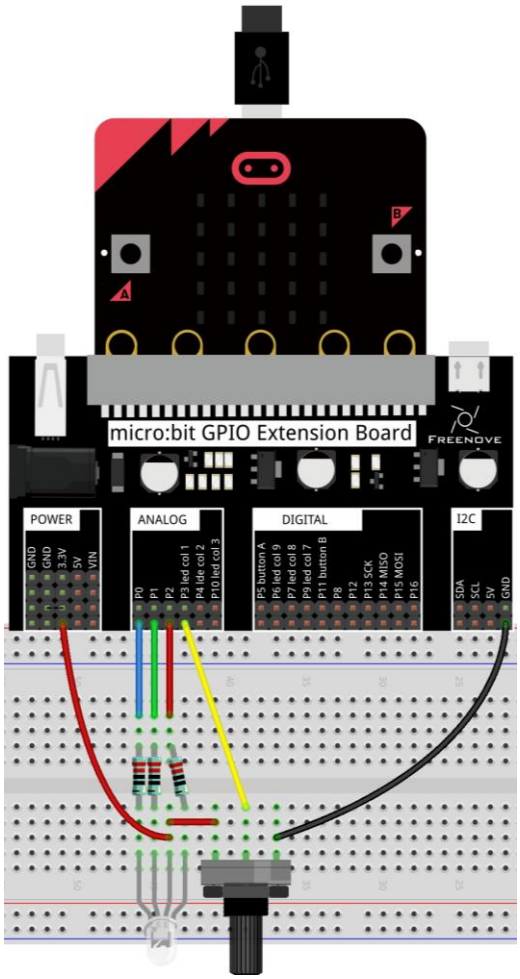
Circuit

Schematic diagram



Hardware connection

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

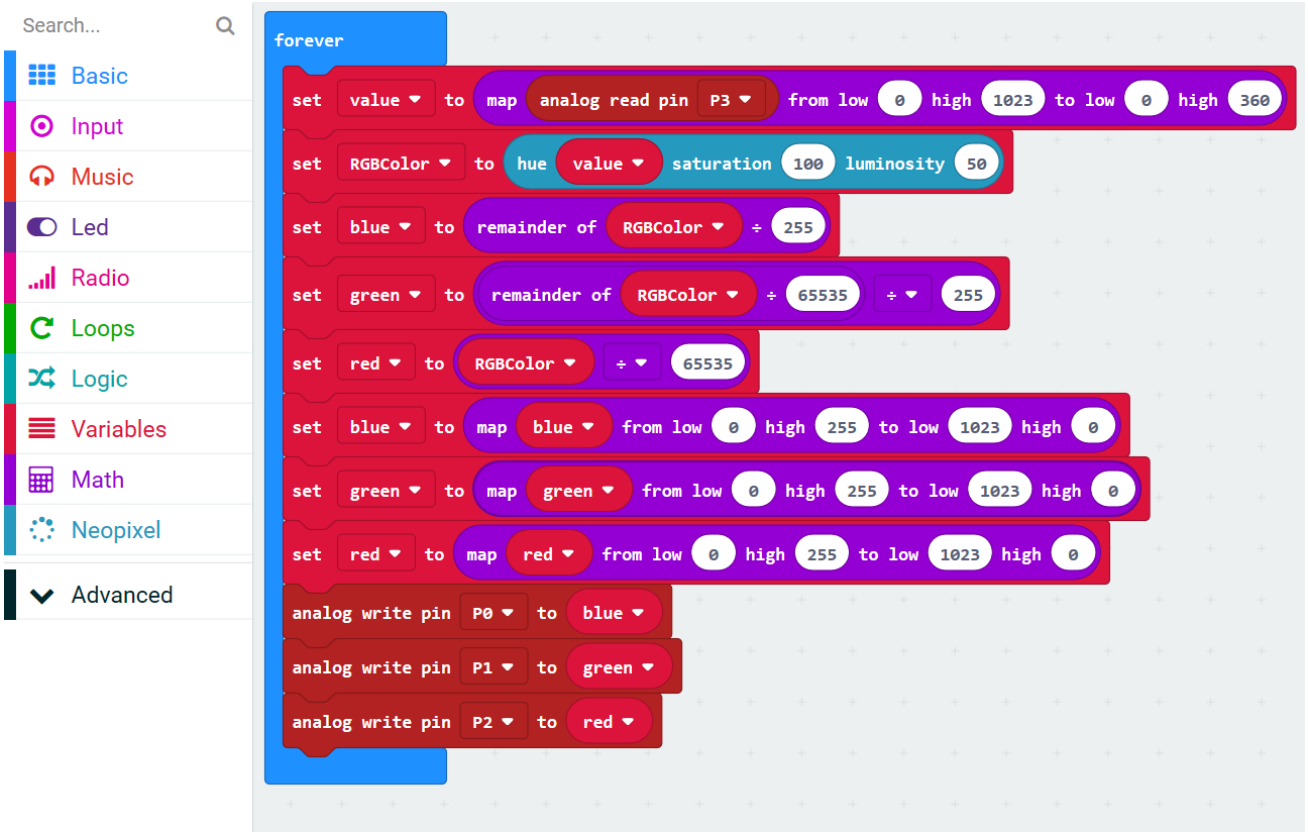


Block code

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/13.2_ColorfulSoftLight	ColorfulSoftLight.hex

After importing successfully, the code is shown as below:



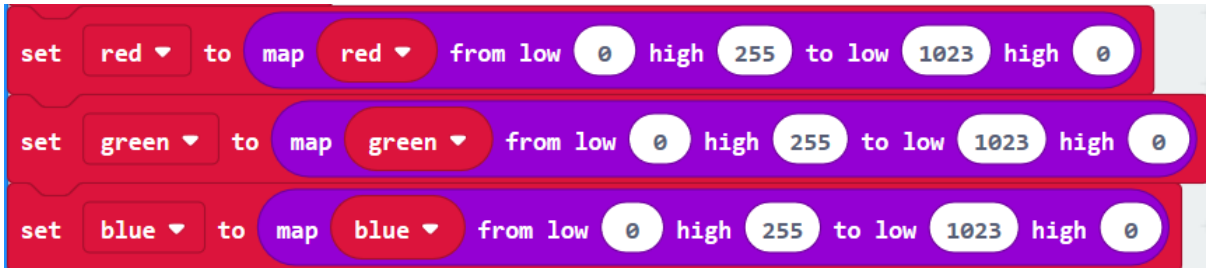
Download the code into micro:bit, rotate the potentiometer, you can see that the color of the RGBLED is changing.

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

Convert the HSL color system to the RGB color system, return the RGB value corresponding to the current hue angle, and store 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. In this kit, three LEDs of RGB LED share a common anode(+) and their negative pins need to be set to LOW level to turn ON the RGB LED work. And the value of variables ‘red’, ‘green’, and ‘blue’ need to be converted from the value ranging from 0-255 to analog signal values ranging from 1023-0, and then reassigned to them.



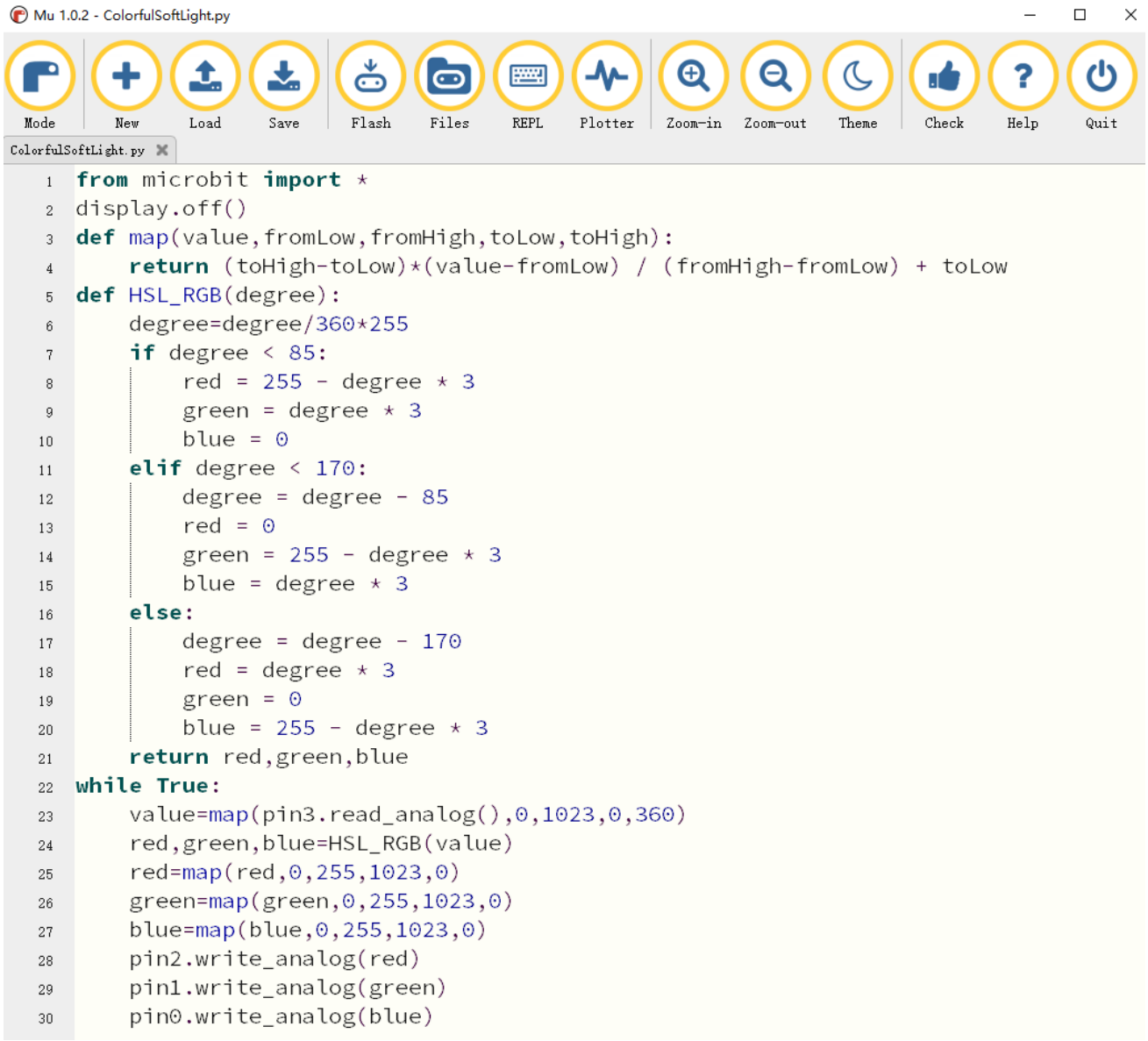
Write the analog voltage value of the red, green, and blue variables to the corresponding P0, P1, and P2 pins to change the LED color.

Python code

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

File type	Path	File name
Python file	../Projects/PythonCode/13.2_ColorfulSoftLight	ColorfulSoftLight.py

After loading successfully, the code is shown as below:



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

The following is the program code:

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

Turn OFF the LED screen to use the P3 pin. A custom map() function 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 function HSL_RGB()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)
```

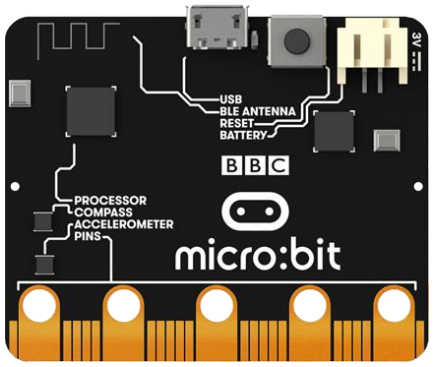
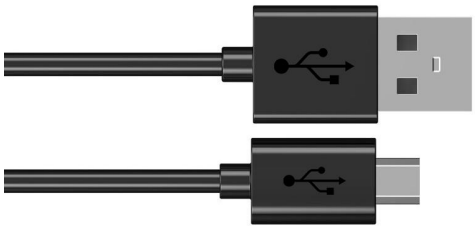
Chapter 14 Light Sensor

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

Project 14.1 Built-in Light Sensor

In this project, we use the micro:bit built-in light sensor to measure the brightness of light.

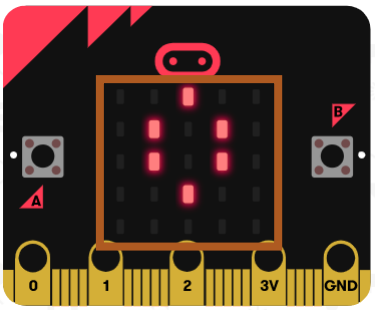
Component list

Microbit x1 	USB cable x1 

Component knowledge

Light sensor

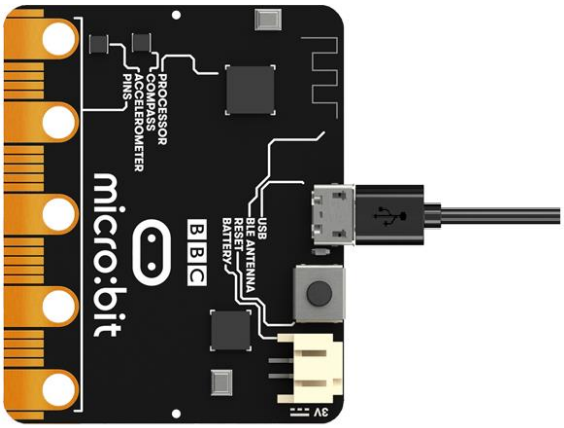
Micro:bit detects the ambient light intensity through the LED matrix. In forward bias mode, the LED screen works as a display. In reverse bias mode, the LED screen works as a basic light sensor that can be used to detect ambient light.



Circuit

Connect micro:bit and PC via a micro USB cable.

Hardware connection



Block code

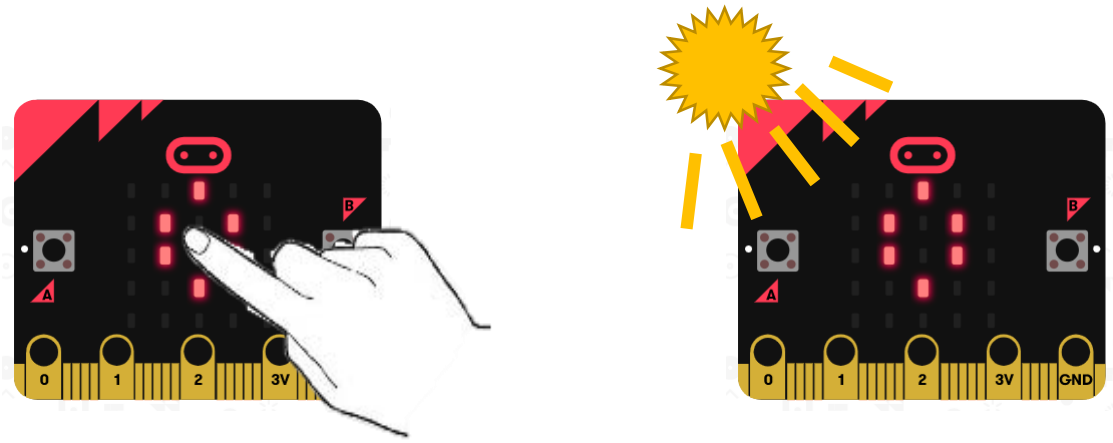
Open MakeCode first. Import the .hex file. The path is as below:

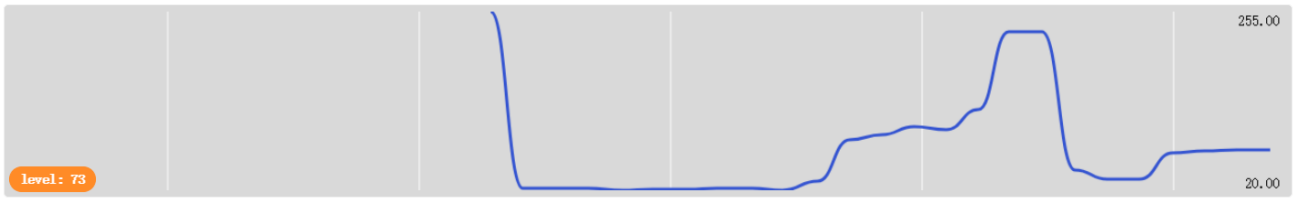
(How to import project)

File type	Path	File name
HEX file	../Projects/BlockCode/14.1_LightIntensityMeter	LightIntensityMeter.hex

After importing successfully, the code is shown as below:

Check the connection of the circuit, verify it correct and download the code into the micro:bit. Open the serial port console, then you can see the reading light intensity. Cover the LED screen with your hand or increase the light shining on it, 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

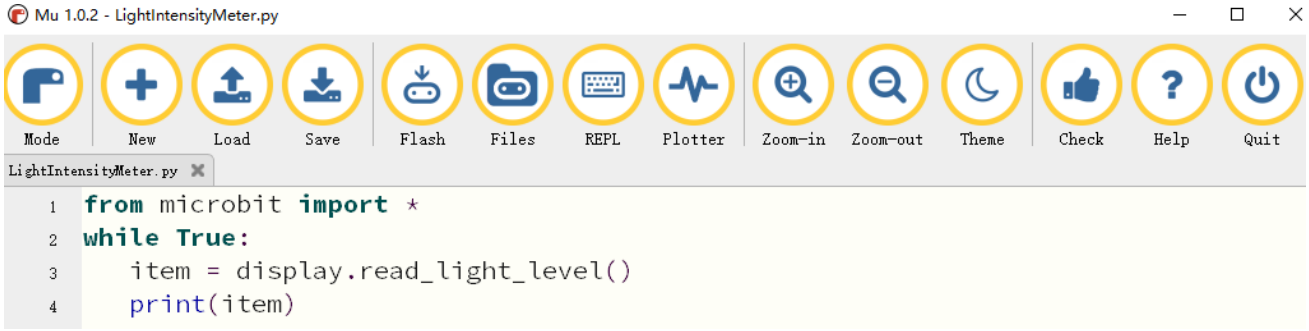
Block	Function
light level	Detect the light level (how bright or dark it is) of the environment where you are. The light level 0 means darkness and 255 means bright light.

Python code

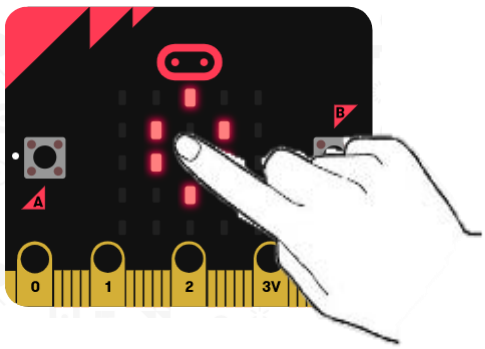
Open the .py file with Mu. Code, the path is as below:

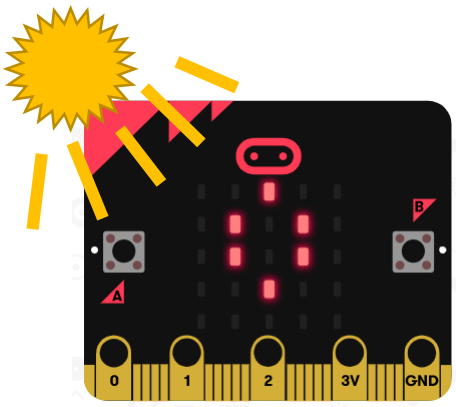
File type	Path	File name
Python file	../Projects/PythonCode/14.1_LightIntensityMeter	LightIntensityMeter.py

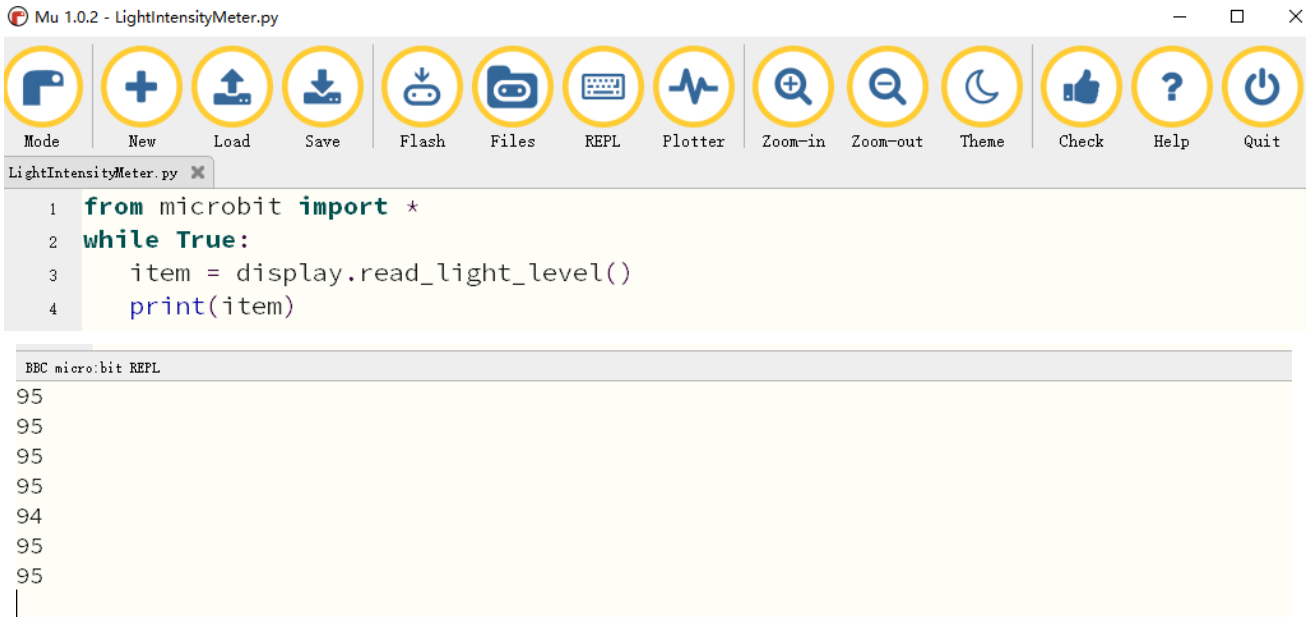
After loading successfully, the code is shown as below:



Check the connection of the circuit, verify it correct, download the code into the micro:bit. Open the serial port console, then you can see the reading light intensity. Cover the LED screen with your hand or increase the light shining on it, 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 following is the program code:

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

Reference

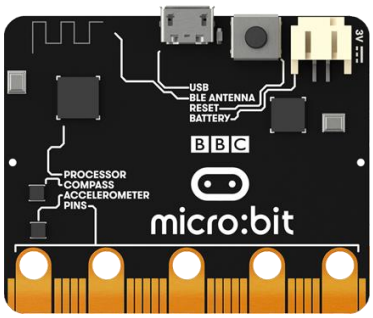
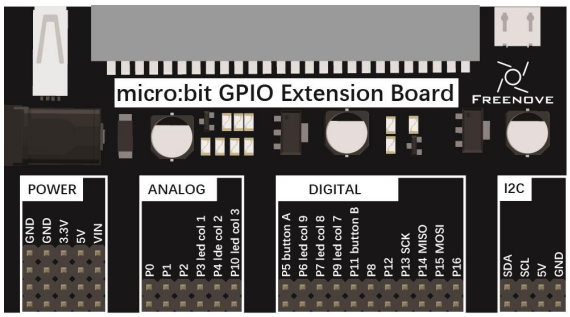
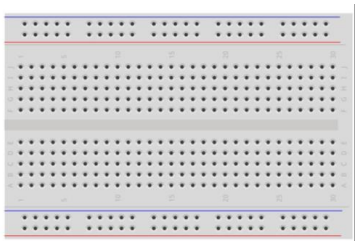
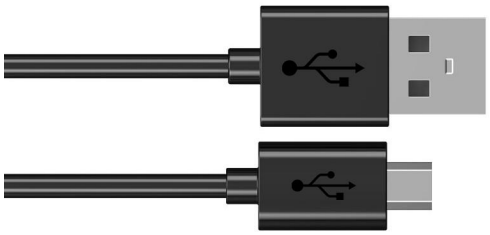
display.read_light_level()

Use the display's LEDs in reverse-bias mode to sense the amount of light falling on the display, return an integer between 0 and 255 representing the light level. The larger the value, the brighter the light..

Project 14.2 Night Light

In this project, we will make a night light.

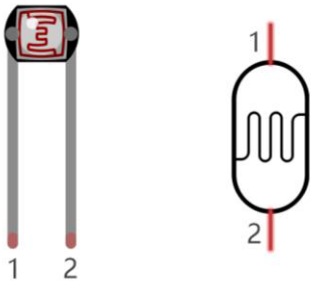
Component list

<table>
<tr><td colspan="2">Microbit x1
  </td><td colspan="2">Extension board x1
  </td></tr>
<tr><td colspan="2">Breadboard x1
  </td><td colspan="2">USB cable x1
  </td></tr>
<tr><td>F/M x 4 M/M x1
  </td><td>Photoresistance x1
  </td><td>LED x1
  </td><td>Resistor 10kΩ x1
  </td></tr>
<tr><td colspan="3"></td><td>Resistor 220Ω x1
  </td></tr>
</table>

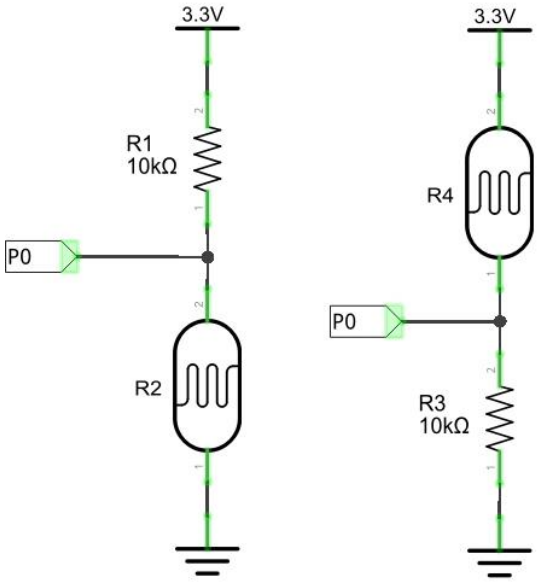
Component knowledge

Photoresistor

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



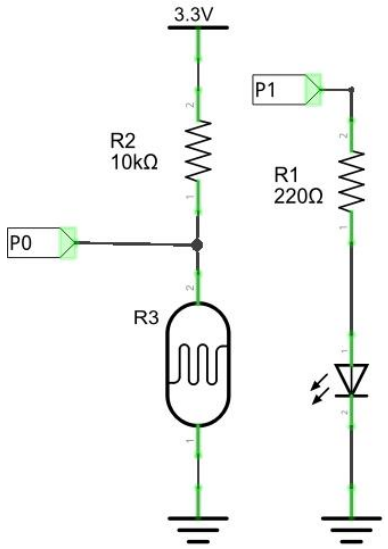
The circuit below is often used to detect the change of a photoresistor's resistance value:



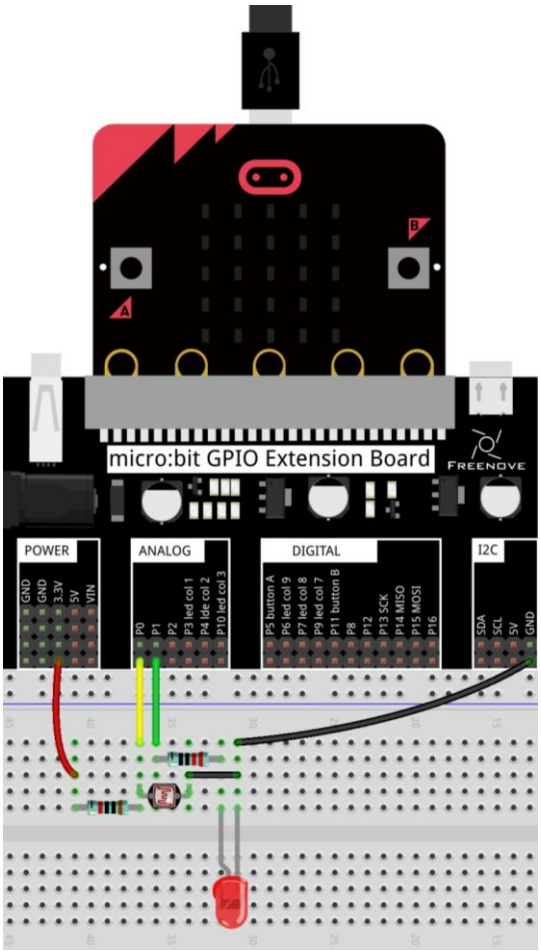
In the above circuit, when a Photoresistor's resistance vale changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

Circuit

Schematic diagram



Hardware connection

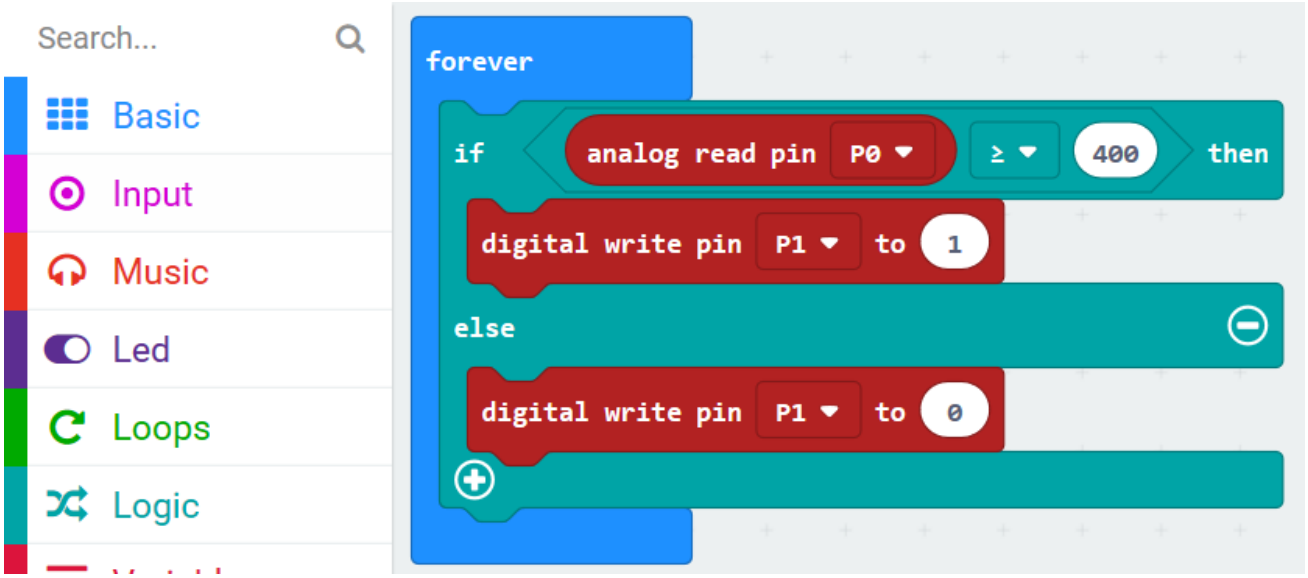


Block code

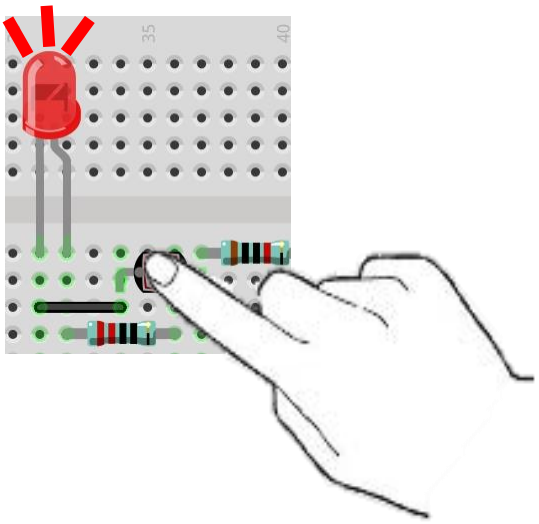
Open MakeCode first. Import the .hex file. The path is as below:
(How to import project)

File type	Path	File name
HEX file	../Projects/BlockCode/14.2_NightLight	NightLight.hex

After importing successfully, the code is shown as below:

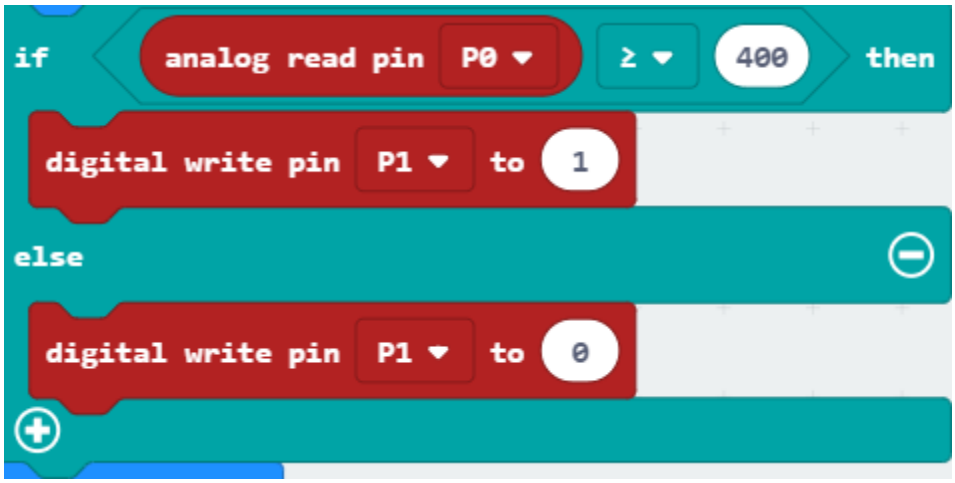


Check the connection of the circuit, verify it correct, and download the code into the micro:bit. Cover the photoresistor with your hand, the LED is turned ON. Move the hand away, the LED is turned OFF.



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.

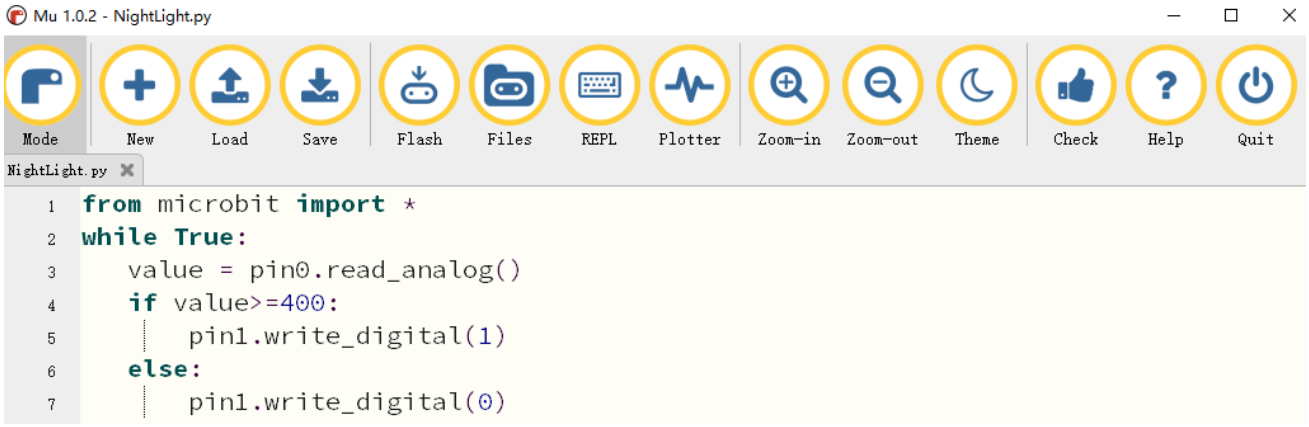


Python code

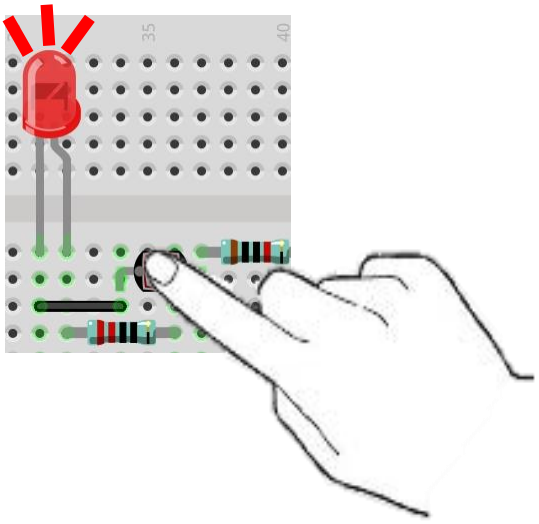
Open the .py file with Mu. Code, the path is as below:

File type	Path	File name
Python file	../Projects/PythonCode/14.2_NightLight	NightLight.py

After loading successfully, the code is shown as below:



Check the connection of the circuit, verify it correct, 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. Otherwise the LED is turned OFF.

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

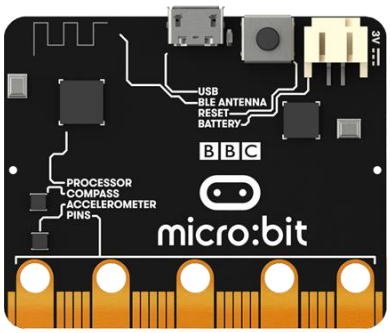
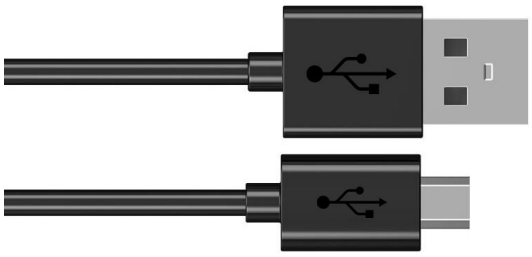
Chapter 15 Temperature Sensor

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

Project 15.1 Built-in Temperature Sensor

In this project, we measure the temperature with the micro:bit's built-in temperature sensor.

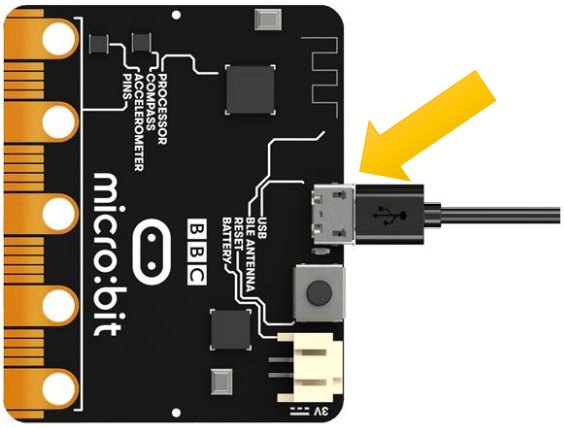
Component list

Microbit x1 	USB cable x1 

Circuit

Connect micro:bit and PC via micro the 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_BuiltInThermometer	BuiltInThermometer.hex

After importing successfully, the code is shown as below:

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

Reference

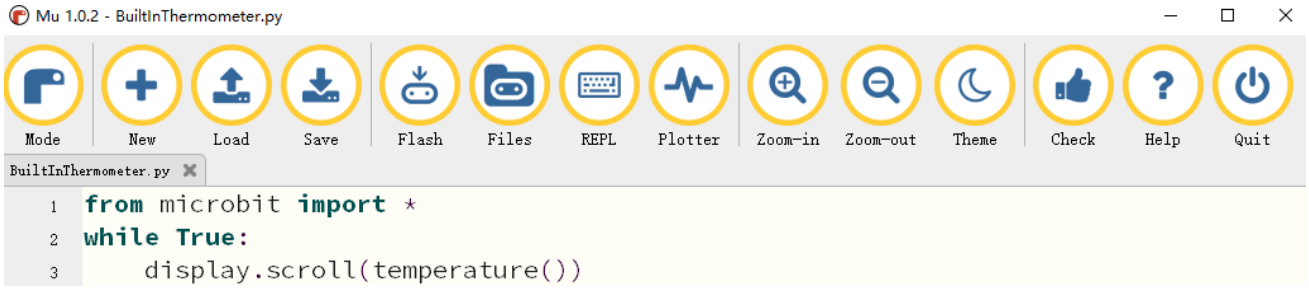
Block	Function
temperature (°C)	Detect the temperature of the environment where you are. The temperature is measured in Celsius (metric).

Python code

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

File type	Path	File name
Python file	../Projects/PythonCode/15.1_BuiltInThermometer	BuiltInThermometer.py

After loading successfully, the code is shown as below:



Check the connection of the circuit, verify it correct, and download the code into micro:bit, and 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())
```

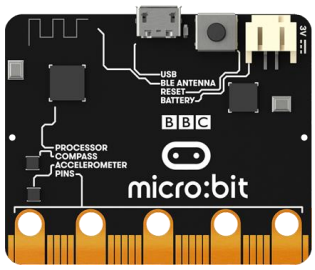
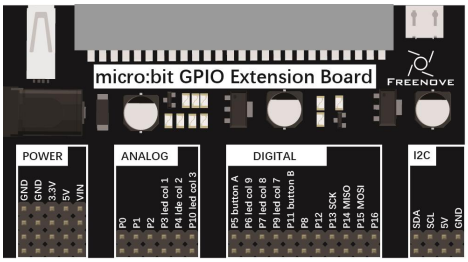
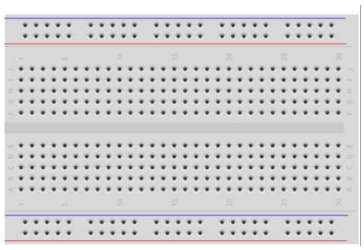
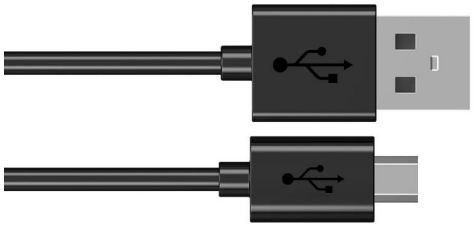
Reference

`temperature()`
Return the temperature of the micro:bit in Celcius.
`display.scroll()`
scrolls a string across the display

Project 15.2 Thermistor

In this project, we will use a thermistor to detect the ambient temperature.

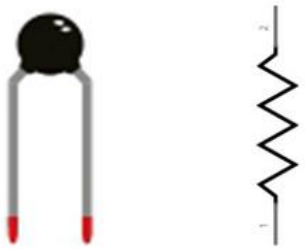
Component list

Microbit x1 	Extension board x1 	
Breadboard x1 	USB cable x1 	
Jumper F/M x3 	Thermistor x1 	Resistor 10kΩ x1 

Component knowledge

Thermistor

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



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

$$Rt=R*EXP[B*(1/T2-1/T1)]$$

Where:

Rt is the thermistor resistance under T2 temperature;

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

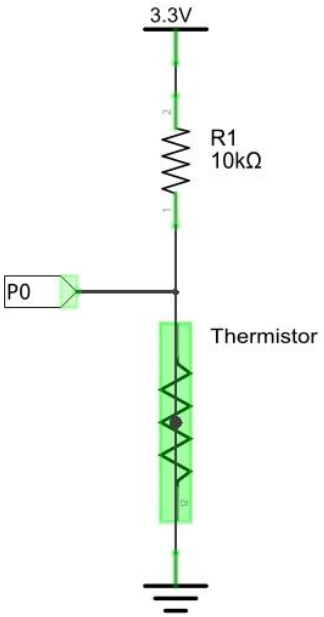
EXP[n] is nth power of e;

B is for thermal index;

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

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

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



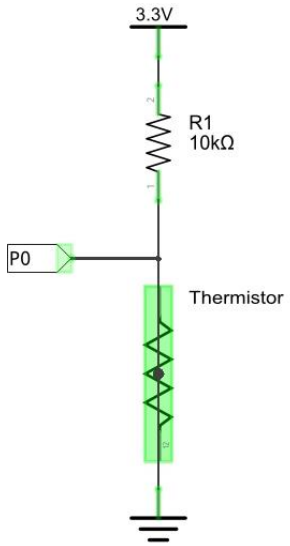
We can use the value measured by the analog pin of micro:bit to obtain resistance value of thermistor, and then we can use the formula to obtain the temperature value.

Therefore, the temperature formula can be derived as:

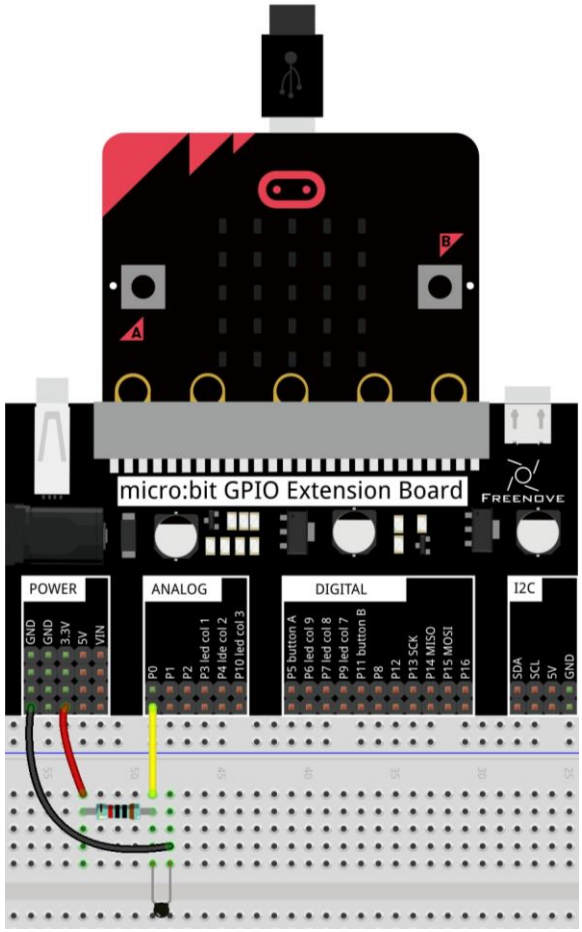
$$T2 = 1/(1/T1 + ln(Rt/R)/B)$$

Circuit

Schematic diagram



Hardware connection



Block code

Open MakeCode first. Import the .hex file. The path is as below:
(How to import project)

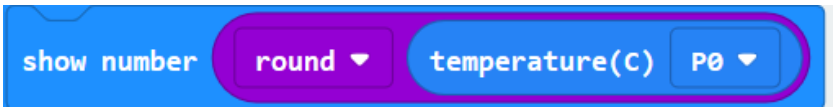
File type	Path	File name
HEX file	../Projects/BlockCode/15.2_ExternalThermometer	ExternalThermometer.hex

After importing successfully, the code is shown as below:

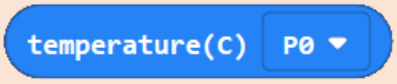
Check the connection of the circuit, verify it correct, download the code into the micro:bit, and the LED dot matrix will display the detected temperature.

The temperature measured by the thermistor and the temperature measured by the micro:bit's 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, and then round up and display on the LED display.

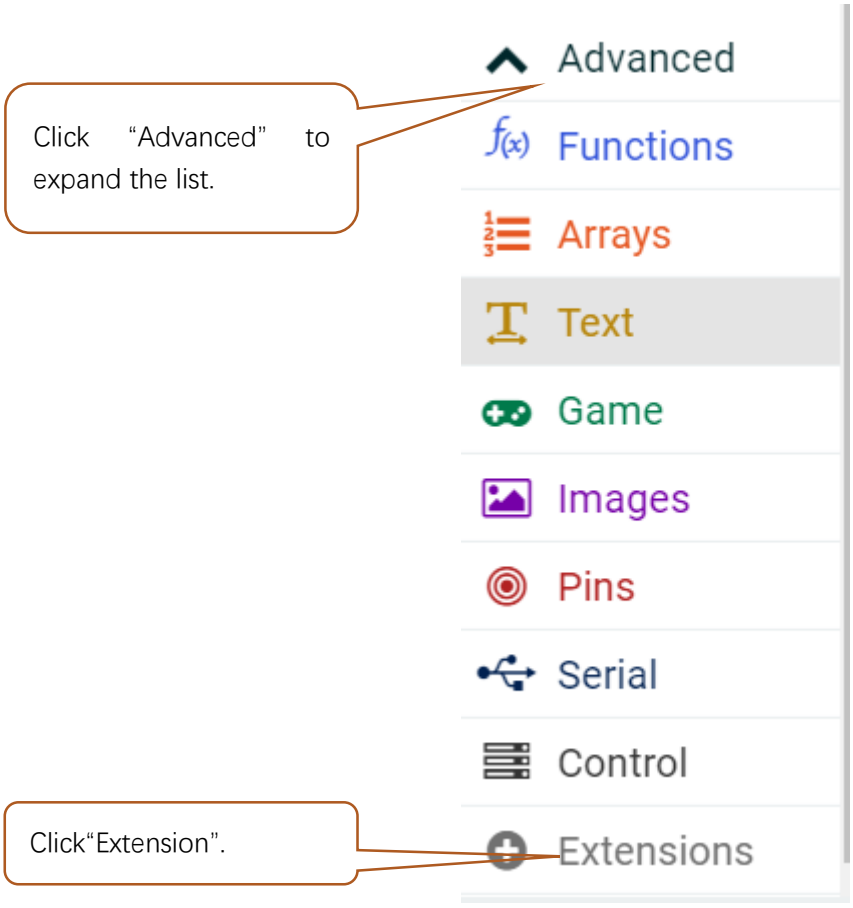


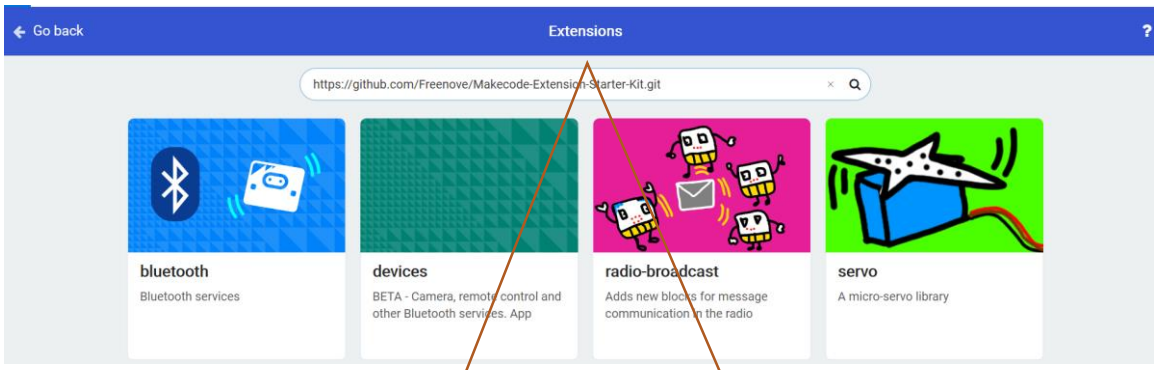
Reference

Block	Function
	Belongs to the Freenove extension block. Get the temperature data measured by the thermistor.

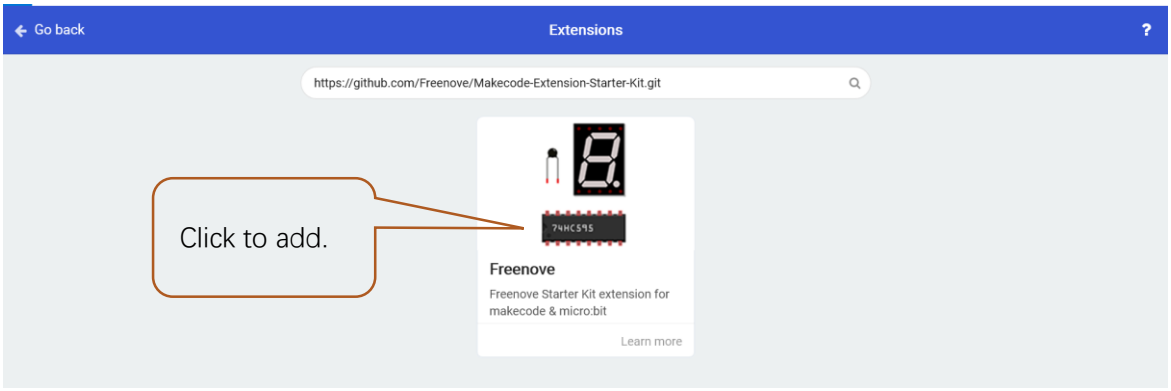
Extensions

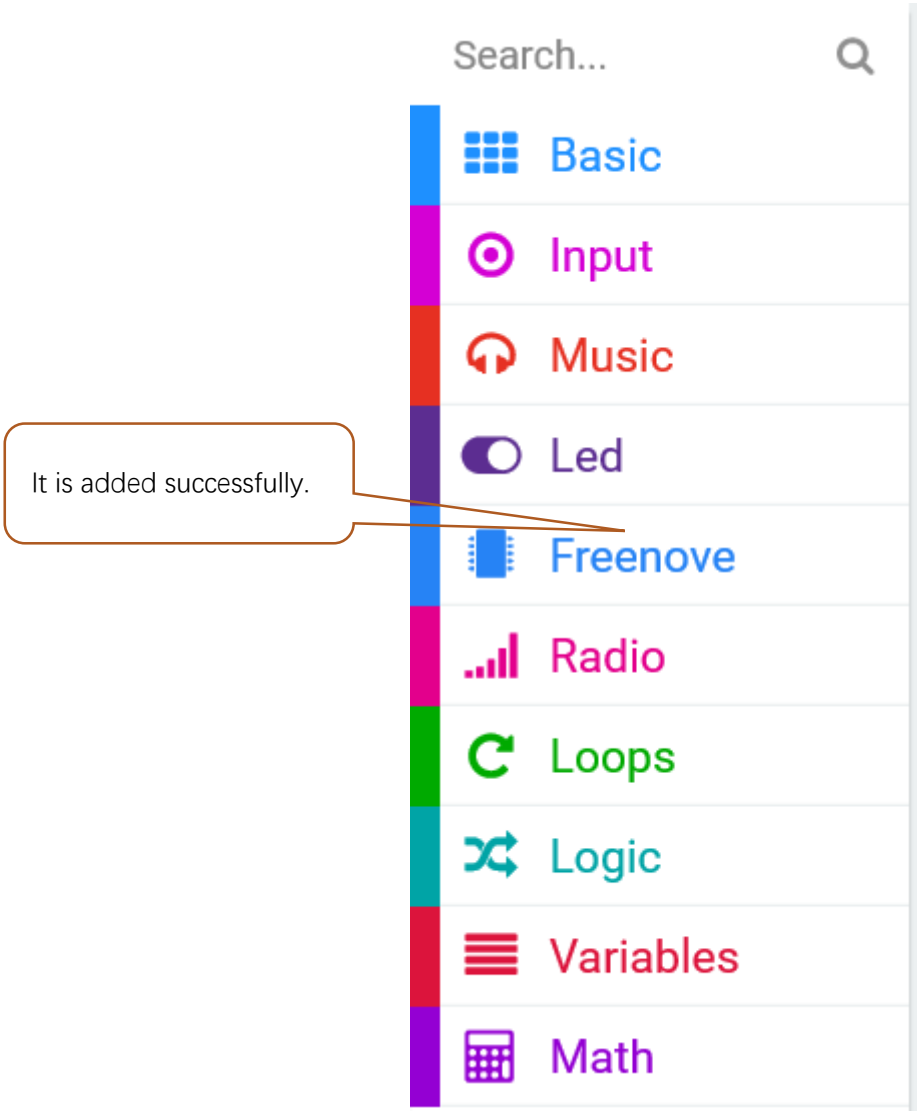
If you want to import Freenove extensions in a new project, follow the steps below to add them.





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


Go back
Extensions
https://github.com/Freenove/Makecode-Extension-Starter-Kit.git
Click to add.
74HC595
Freenove
Freenove Starter Kit extension for makecode & micro:bit
Learn more

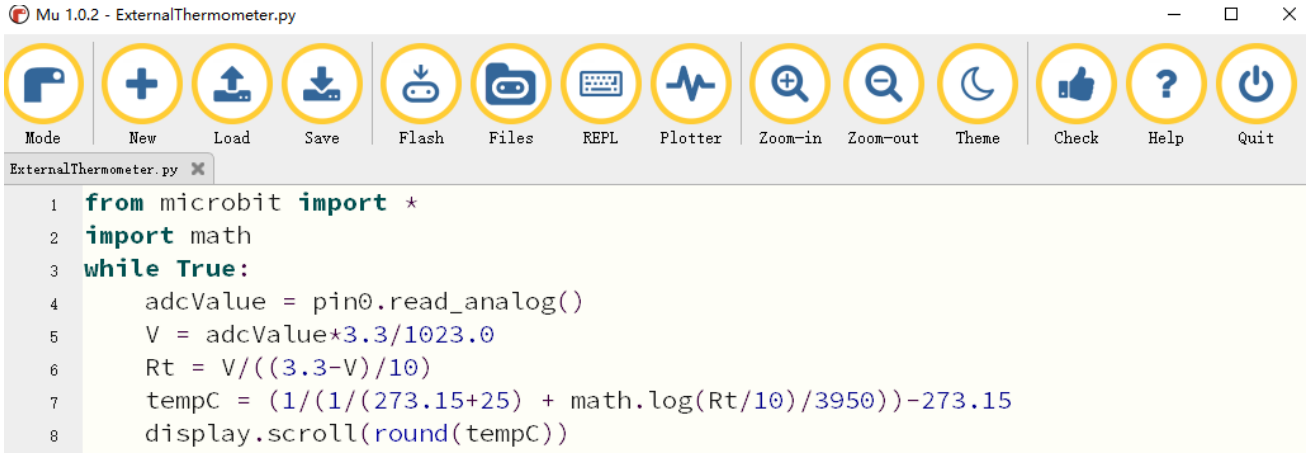

Search...
Basic
Input
Music
Led
Freenove
Radio
Loops
Logic
Variables
Math
It is added successfully.

Python code

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

File type	Path	File name
Python file	../Projects/PythonCode/15.2_ExternalThermometer	ExternalThermometer.py

After loading successfully, the code is shown as below:



Check the connection of the circuit, verify it correct, 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 and then display it on the LED dot matrix screen. For the formula, please refer to the component knowledge..

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

What's Next?

THANK YOU for participating in this learning experience!

We have reached the end of this Tutorial. If you find errors, omissions or you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us: support@freenove.com
We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.
If you want to learn more about micro:bit, we have a smart Car named Micro:bit Rover. You can visit our website to purchase. http://www.freenove.com/store.html

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

Thank you again for choosing Freenove products.