Table of Contents

1. torial Description

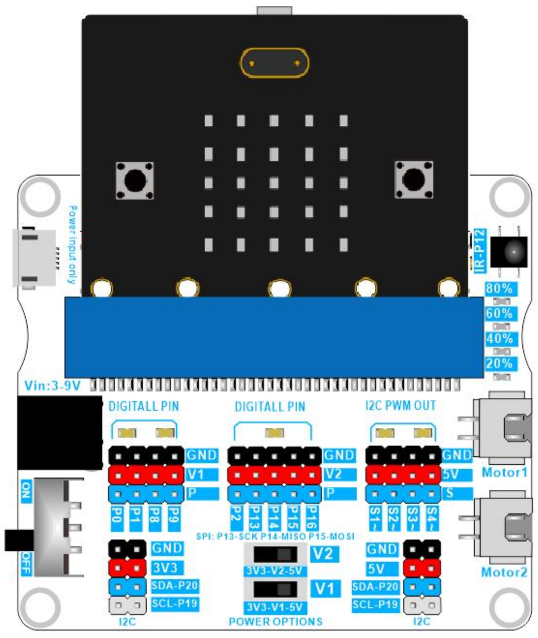
1.1 Python is a general-purpose programming language that runs on powerful hardware (such as desktops and servers). MicroPython is a simplified version of Python designed for microcontrollers and embedded devices.

1.2 This tutorial uses MicroPython for programming. The Micro:bit does not have a dedicated NEC infrared MicroPython library. If using Pin to read NEC infrared signals, the reading speed is too slow, causing severe signal loss. Therefore, this tutorial cannot implement an NEC infrared remote control.

2. Pybit Introduction

The mShield expansion board is our latest easy-to-use micro:bit expansion board. It integrates powerful features such as 2-channel motor drive, infrared reception, battery level reading, 4-channel servo drive, 4-channel PWM signal output, LED indicators, and selectable 3V or 5V power output.

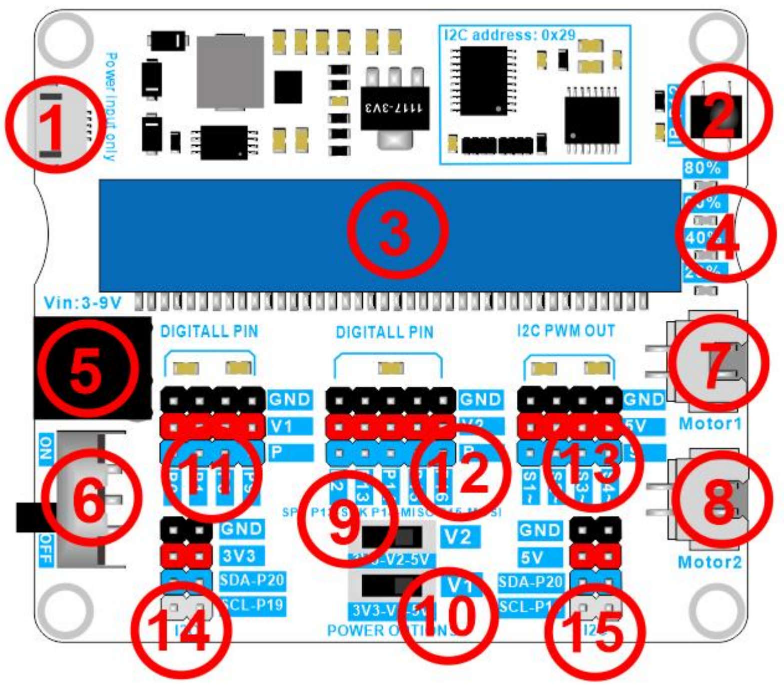
The expansion board extends the commonly used pins of the micro:bit in the form of pin headers, allowing connection to other sensors. The expansion board can be connected to two types of multi-cell batteries and also reserves a micro USB port, allowing connection to a power bank, making power supply more adaptable and safer. To facilitate fixing the control board to other devices, the expansion board comes with 4 fixing holes, making it easy to use in other projects.



3. Specification parameter

Shield	
Name	mShield
SKU	M1E0002
USB	**Applicable motherboard**
Micro USB	Micro:bit
Pins	
Digital I/O Pins	9 (P0, P1, P2, P8, P9, P13, P14, P15, P16)
Analog read pins	3 (P0, P1, P2)
Analog write pins	9 (P0, P1, P2, P8, P9, P13, P14, P15, P16)
PWM pins	4 (S1~, S2~, S3~, S4~), period: 2ms.
Servo pins	4 (S1~, S2~, S3~, S4~)
Communication	
UART	Yes
I2C	Yes
SPI	Yes
IR receiver	Yes (NEC)
Power	
Input voltage (nominal)	3--9V
DC Current for 3.3V Pin	500 mA
DC Current for 5V Pin	2000 mA
Motors	
Connectors	Motor1, Motor2 (XH2.54 2P)
Output voltage	3--9V
Maximum drive current	1.2A
Recommended driving	1A
LEDs	
Number	4 (20%, 40%, 60%, 80%)
Dimensions	
Width	62 mm
Length	73 mm
Height	14mm
Weight	29.16 g

4. Interface specification



Number	Description
1	Micro USB port, only for external power supply.
2	Infrared receiver, uses P12 pin.
3	Micro:bit slot.
4	LED indicators, controlled by an internal I2C chip.
5	Power socket (DC005), can input 3-9V voltage, can be connected to 3, 4, 5, and 6 AA batteries, or 1 and 2 lithium batteries, has reverse connection protection function.
6	Power socket switch.
7、8	Motor interfaces, can be connected to 3-9V DC motors, the motor voltage equals the power socket voltage. Controlled by an internal I2C chip.
9、10	V1 and V2 pin header output voltage selection switch.
11、12	Microbit IO expansion port.
13	mShield internal expansion port, controlled by an internal I2C chip.
14	3.3V power I2C interface, P19(SCL), P20(SDA).
15	5V power I2C interface P19(SCL), P20(SDA).

5. Python Editor Basics

The Python Editor is the perfect way to start MicroPython programming with the BBC Micro:bit. The Python Editor is free and runs in a browser on all platforms.

We recommend using the Google Chrome or Microsoft Edge browser. WebUSB is a newly developed web feature that allows you to access the Micro:bit directly from a web page. It also allows you to send data directly from the Micro:bit to the Python Editor. It works with Google Chrome and Microsoft Edge browsers.

WebUSB support micro:bit version

If you are not using the latest version of Google Chrome or Microsoft Edge, please ensure they are this version or newer:

Google Chrome (version 79 and above) for Android, Chrome OS, Linux, macOS, and Windows 10.

Microsoft Edge (version 79 and above) for Android, Chrome OS, Linux, macOS, and Windows 10.

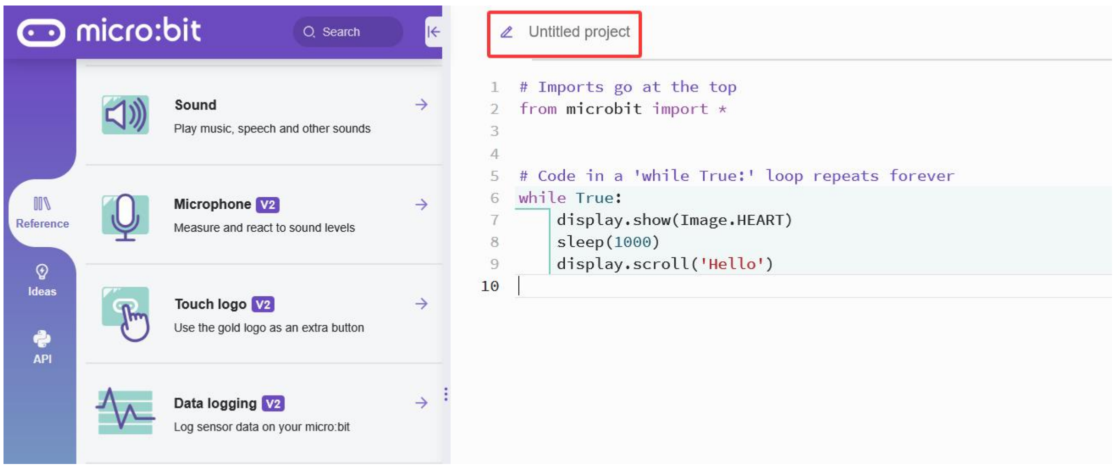
Link to download the latest Google Chrome browser:

https://www.google.com/chrome/

Link to download the latest Microsoft Edge:
https://www.microsoft.com/en-us/edge/download

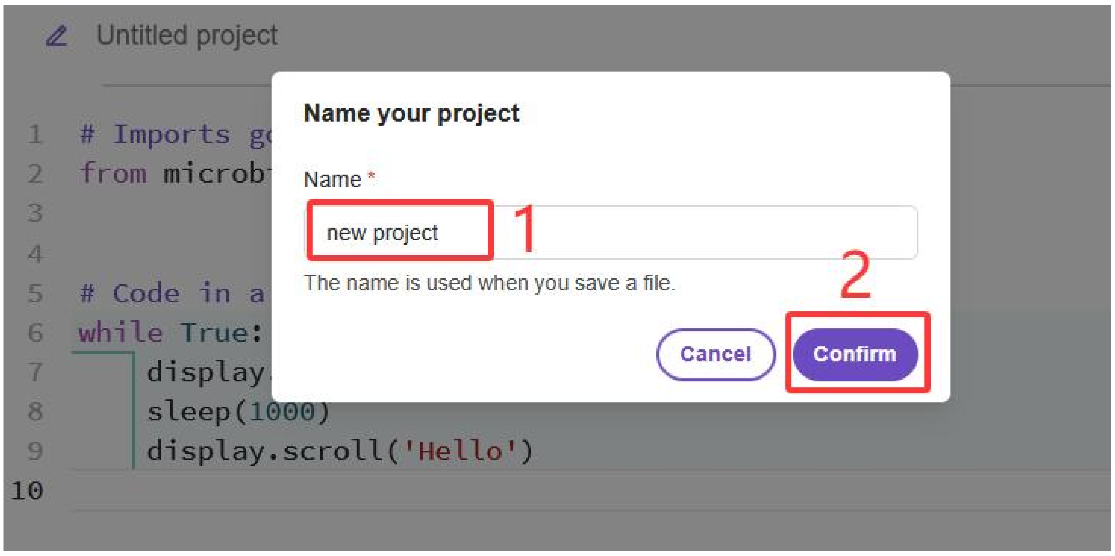
5.1 Create a new project

Open the online programming page in the PC's Google Chrome browser: https://python.microbit.org/v/3, click the red box:


micro:bit
Search
Reference
Ideas
API
Sound
Play music, speech and other sounds
Microphone V2
Measure and react to sound levels
Touch logo V2
Use the gold logo as an extra button
Data logging V2
Log sensor data on your micro:bit
Untitled project

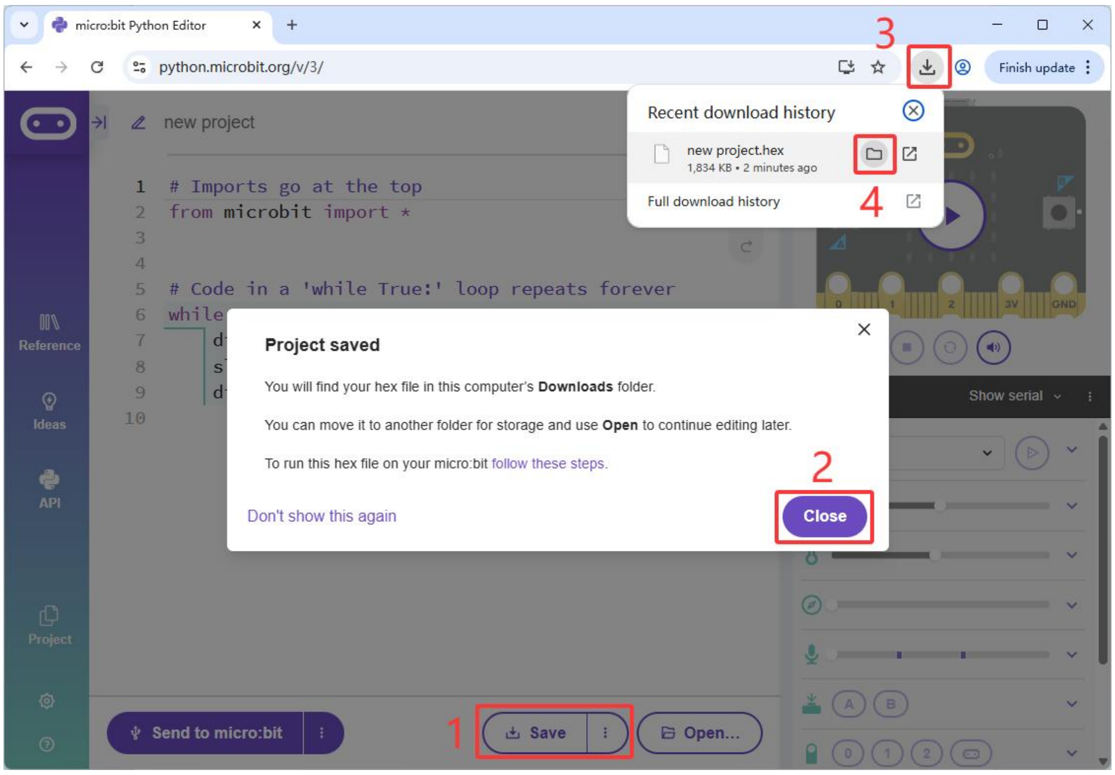
```
# Imports go at the top
from microbit import *


# Code in a 'while True:' loop repeats forever
while True:
    display.show(Image.HEART)
    sleep(1000)
    display.scroll('Hello')
```

Then enter the project name and click "Confirm":



5.2 Save project locally

Click the Step 1 "Save" button, then click the Step 2 "Close" button to save the project locally. Then follow steps 3 and 4 to find the saved ".hex" file, as shown below:



```
# Imports go at the top
from microbit import *


# Code in a 'while True:' loop repeats forever
while
    d
    s
    d
```

Project saved

You will find your hex file in this computer's **Downloads** folder.

You can move it to another folder for storage and use **Open** to continue editing later.

To run this hex file on your micro:bit follow these steps.

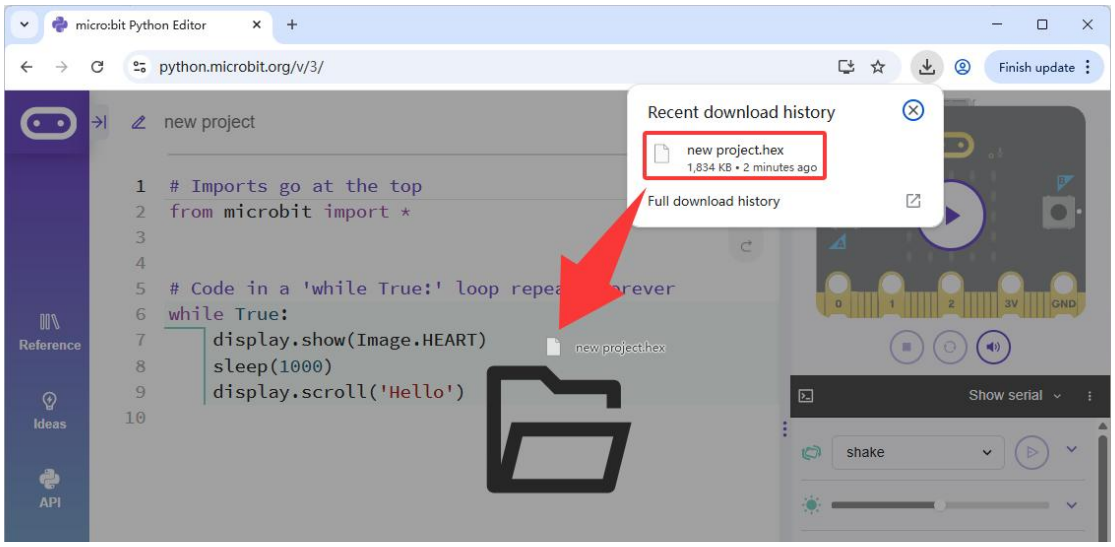
Don't show this again

Close

Note: If you do not want the pop-up window to appear again, click "Don't show this again" in step 2!

5.3 Import local project

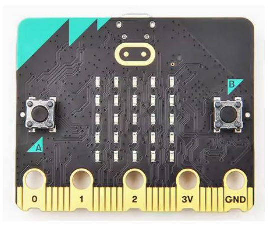
Directly drag the local "HEX" project file into the workspace of the Python Editor, as shown below:



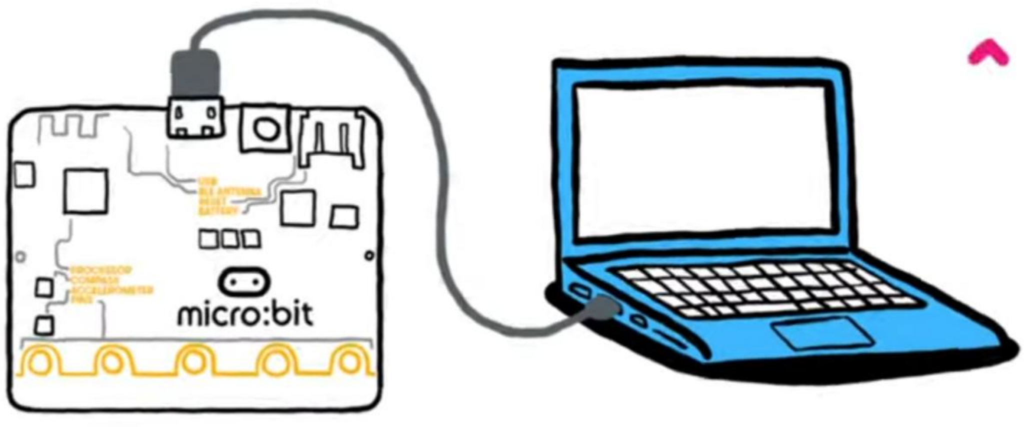
```
# Imports go at the top
from microbit import *


# Code in a 'while True:' loop repeats forever
while True:
    display.show(Image.HEART)
    sleep(1000)
    display.scroll('Hello')
```

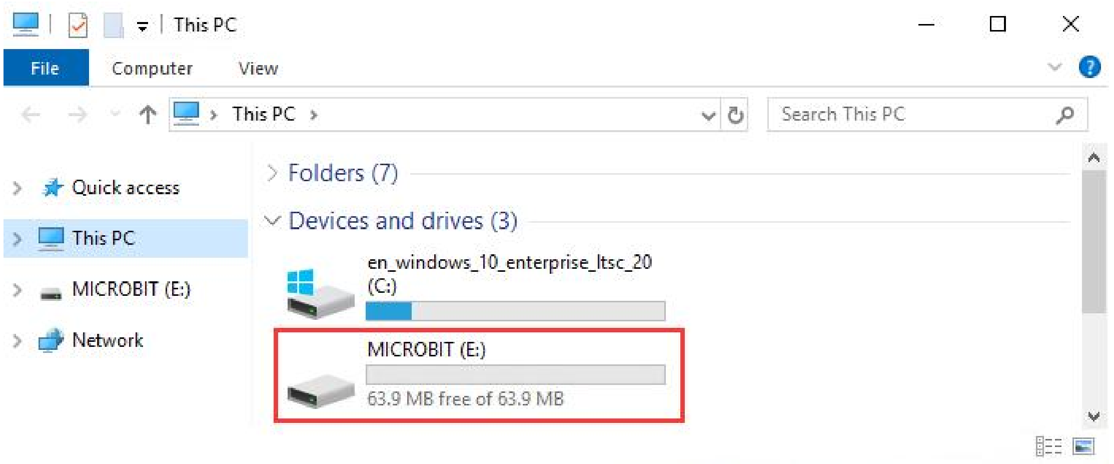
5.4 Upload project

Tools:

PC	Micro:bit v2.x.x	Micro USB cable
		

Connect the Micro:bit motherboard to the PC using a Micro USB cable. A storage drive will appear on the PC:





This PC

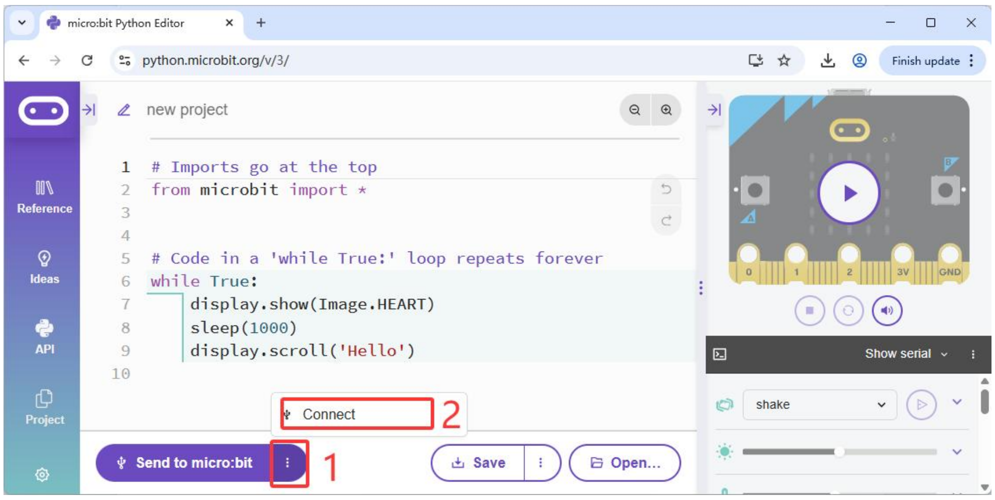
File Computer View

This PC › Search This PC

Quick access
This PC
MICROBIT (E:)
Network

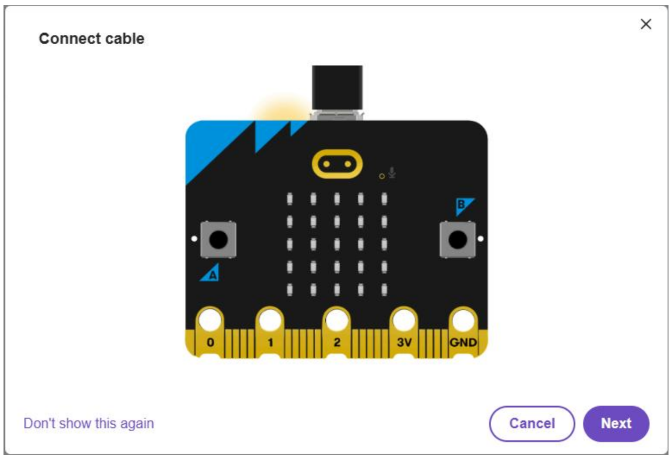
Folders (7)
Devices and drives (3)
en_windows_10_enterprise_ltsc_20 (C:)
MICROBIT (E:)
63.9 MB free of 63.9 MB

Click the "Connect" button:

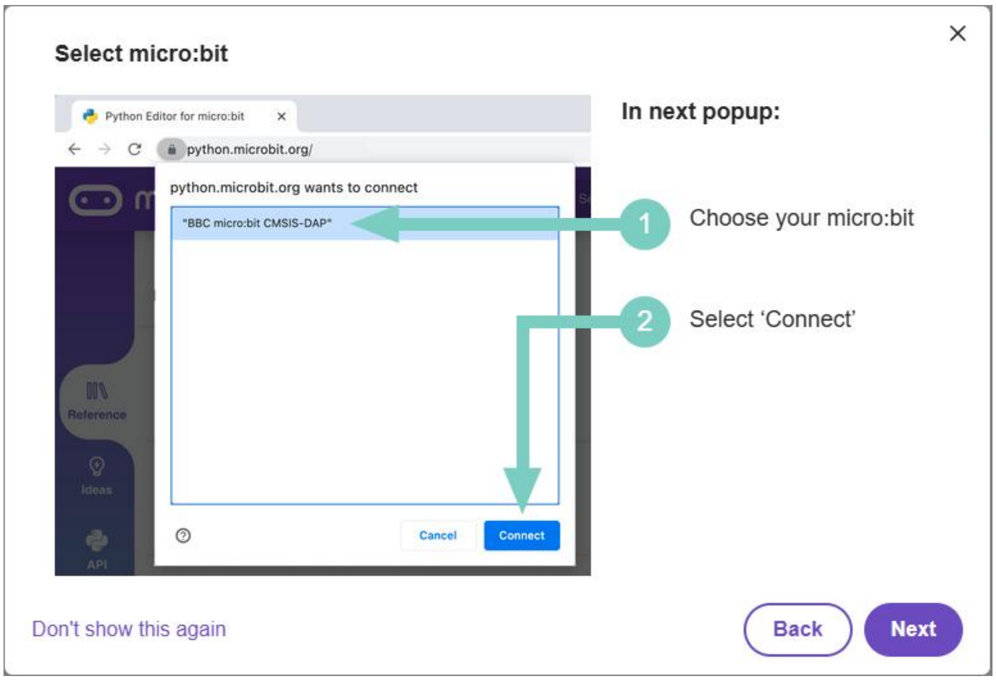


```
# Imports go at the top
from microbit import *


# Code in a 'while True:' loop repeats forever
while True:
    display.show(Image.HEART)
    sleep(1000)
    display.scroll('Hello')
```

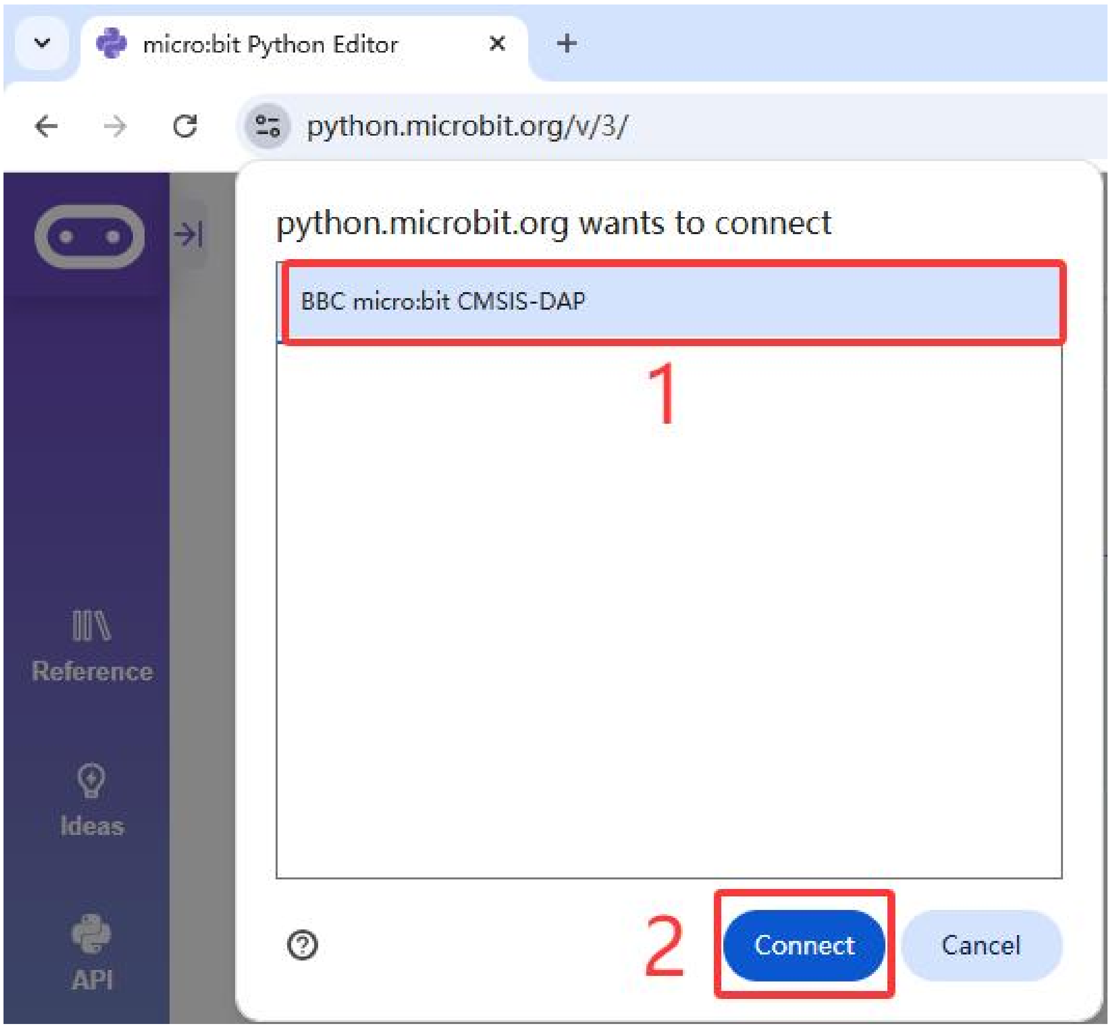
Click "Next":



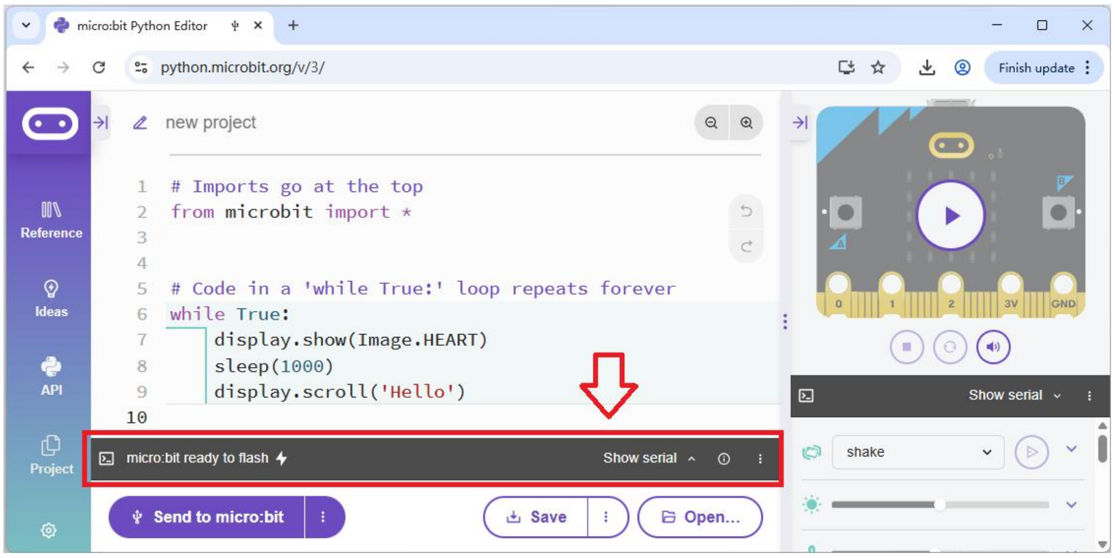
Click "Next":



Then follow these steps:

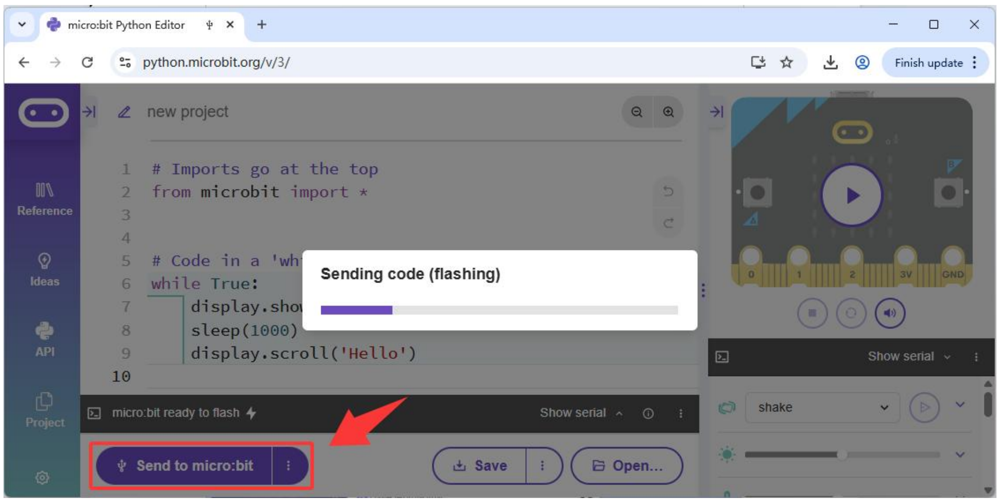


When the following appears, it indicates a successful connection between the Micro:bit and the Python Editor:



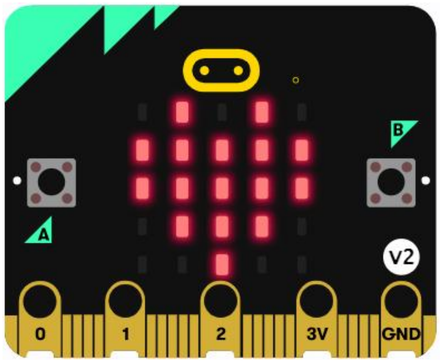
```
# Imports go at the top
from microbit import *


# Code in a 'while True:' loop repeats forever
while True:
    display.show(Image.HEART)
    sleep(1000)
    display.scroll('Hello')
```

Now click "Send to micro:bit" to upload the code from the Python Editor to the Micro:bit:



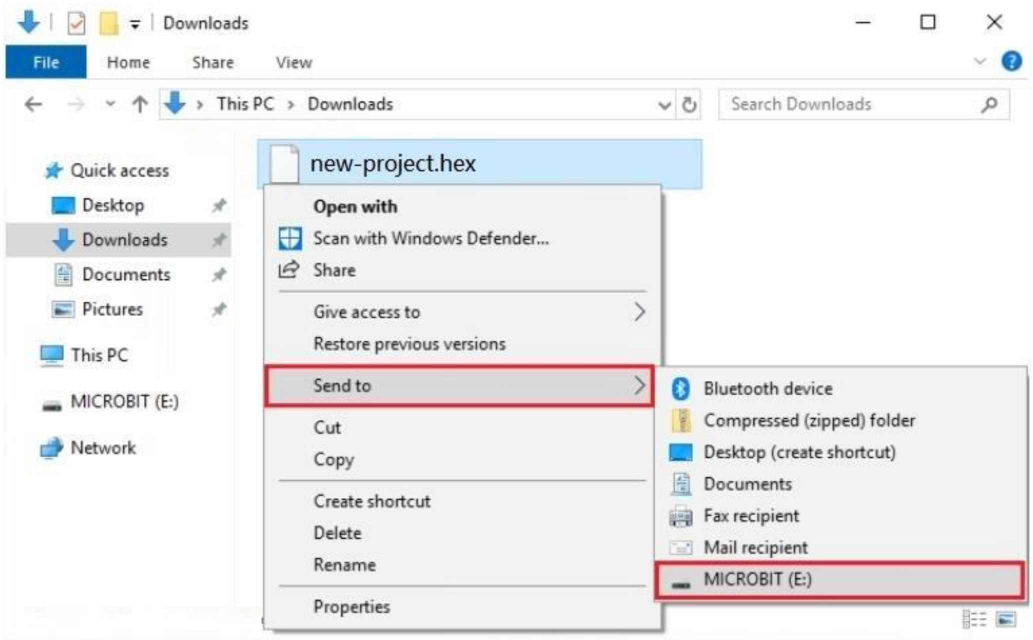
```
# Imports go at the top
from microbit import *


# Code in a 'wh
while True:
    display.sho
    sleep(1000)
    display.scroll('Hello')
```

After the code is uploaded, the Micro:bit motherboard's dot matrix displays a heart shape, then scrolls "Hello":

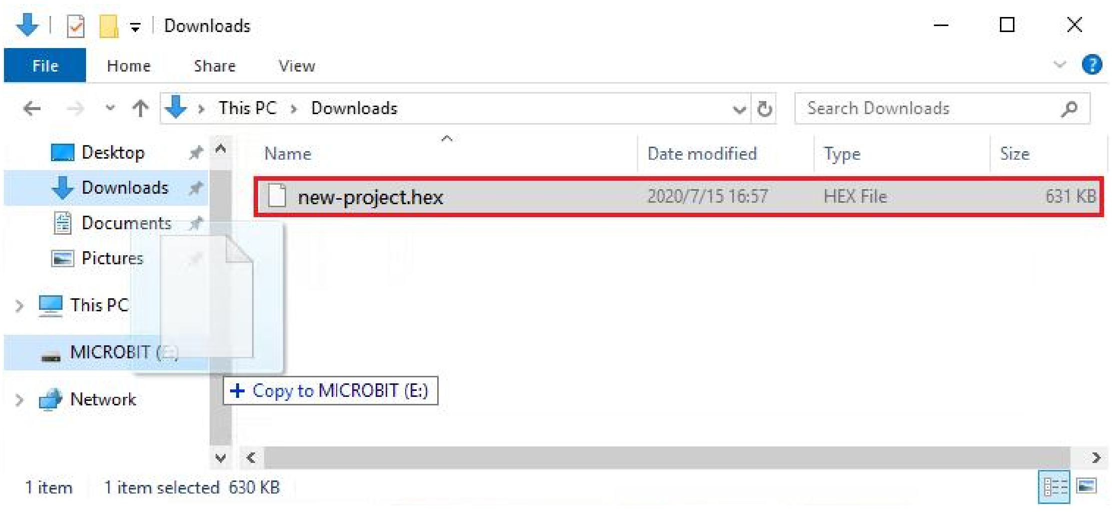


Other methods to upload code

Select the "HEX" file, right-click, and send the "HEX" file to the Micro:bit mainboard:

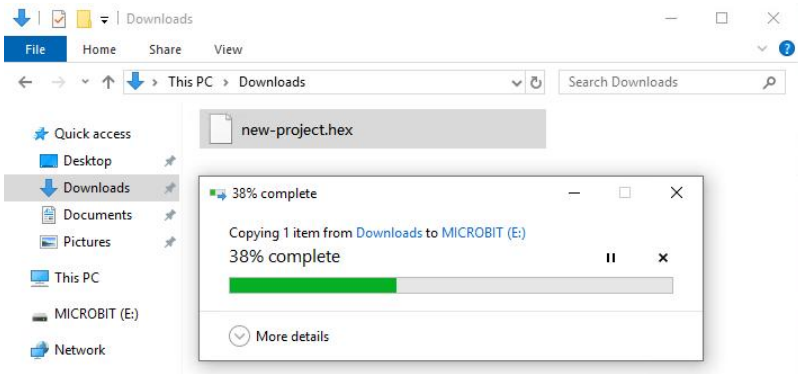


Or directly drag and drop it onto the Micro:bit mainboard:



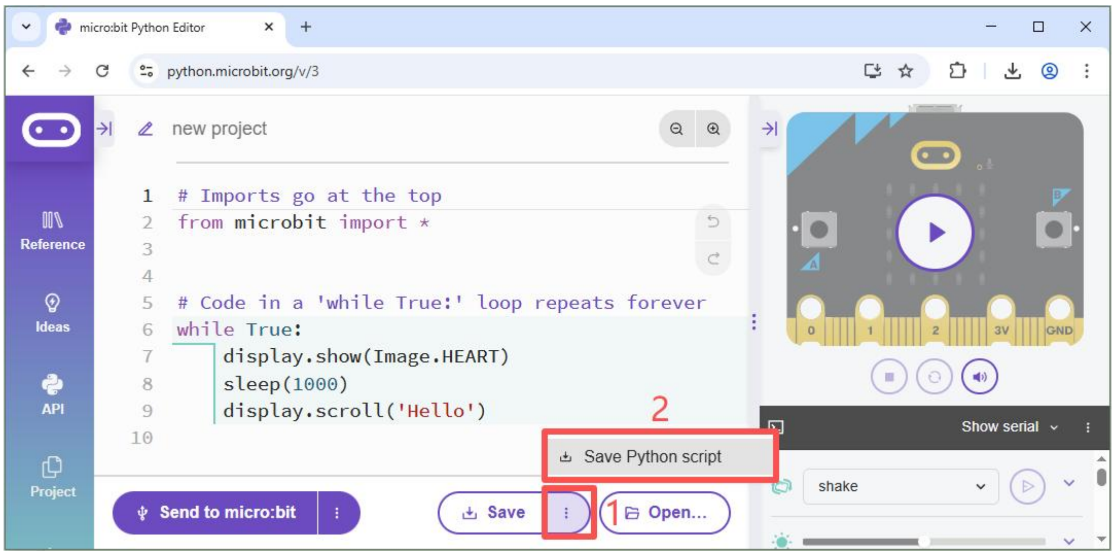
Name	Date modified	Type	Size
new-project.hex	2020/7/15 16:57	HEX File	631 KB

The following interface indicates that the code is being uploaded:

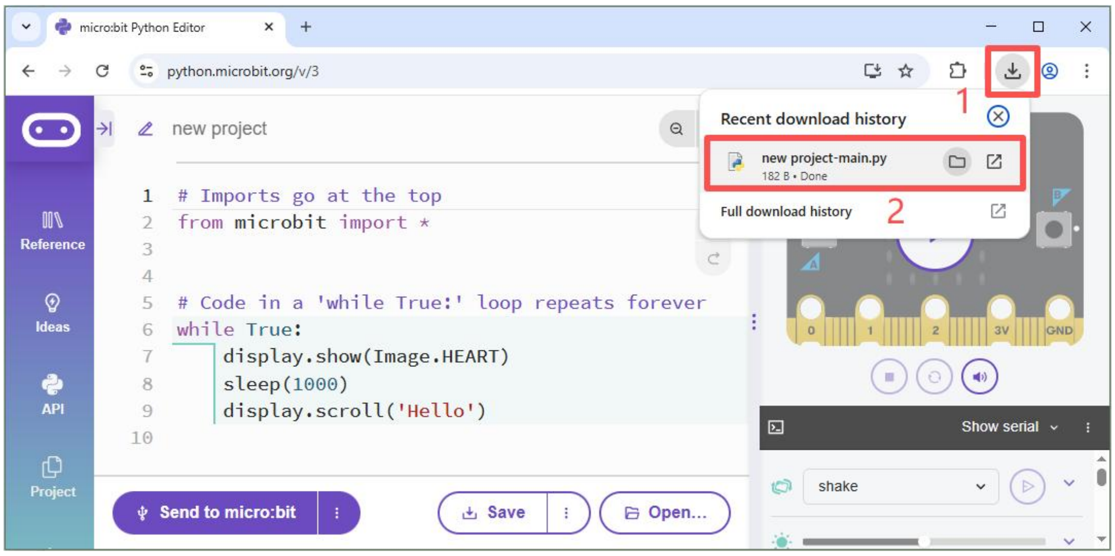


Copying 1 item from Downloads to MICROBIT (E:)
38% complete

5.5 Save project as Python file



```
# Imports go at the top
from microbit import *


# Code in a 'while True:' loop repeats forever
while True:
    display.show(Image.HEART)
    sleep(1000)
    display.scroll('Hello')
```



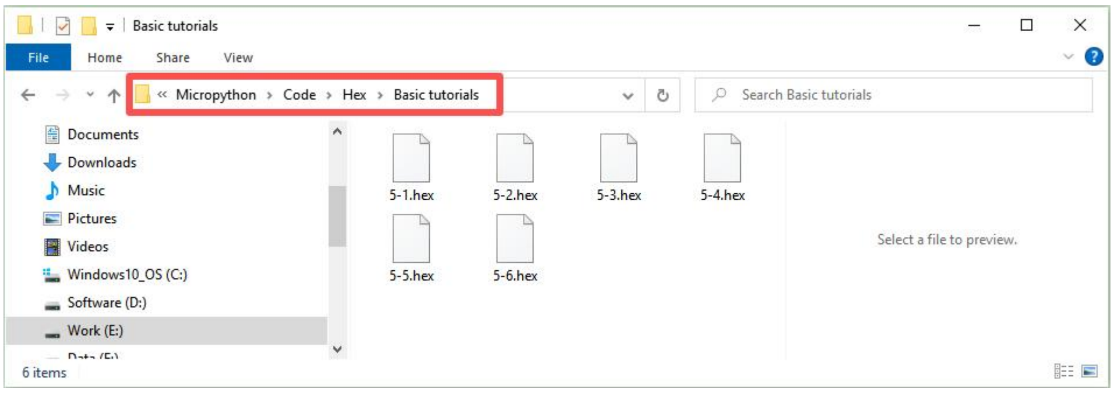
Note: Python "xx.py" files cannot run directly on the Micro:bit; only the project's "xx.hex" file can run on the Micro:bit. "xx.py" files can be opened with general text editing software, while "xx.hex" files cannot!

5.6 Python Editor basic syntax learning

The Micro:bit platform provides official Python Editor usage reference documentation.
User Guide: https://microbit.org/get-started/user-guide/python-editor/
MicroPython documentation: https://microbit-micropython.readthedocs.io/en/v2-docs/

6. Example Code

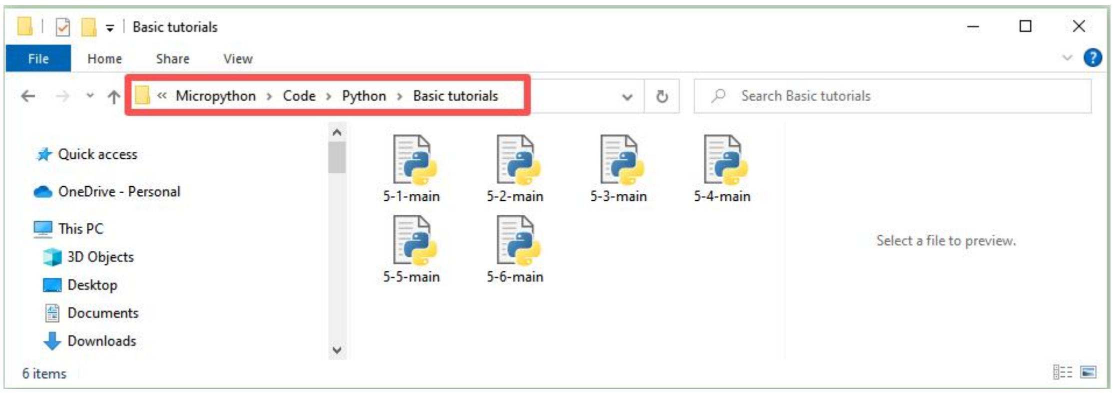
The example projects for the basic tutorials are all saved in the "Micropython -> Code -> Hex -> Basic tutorials" folder:



Special reminder!!

For the following tutorials, if you want to import our provided local example projects, please refer to section 5.3. If you want to create a new project, please refer to section 5.1.

The example Python files for the basic tutorials are saved in the "Micropython -> Code -> Python -> Basic tutorials" folder:

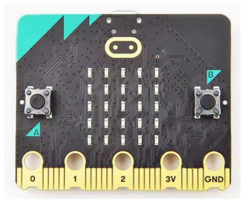


6.1 Firmware version

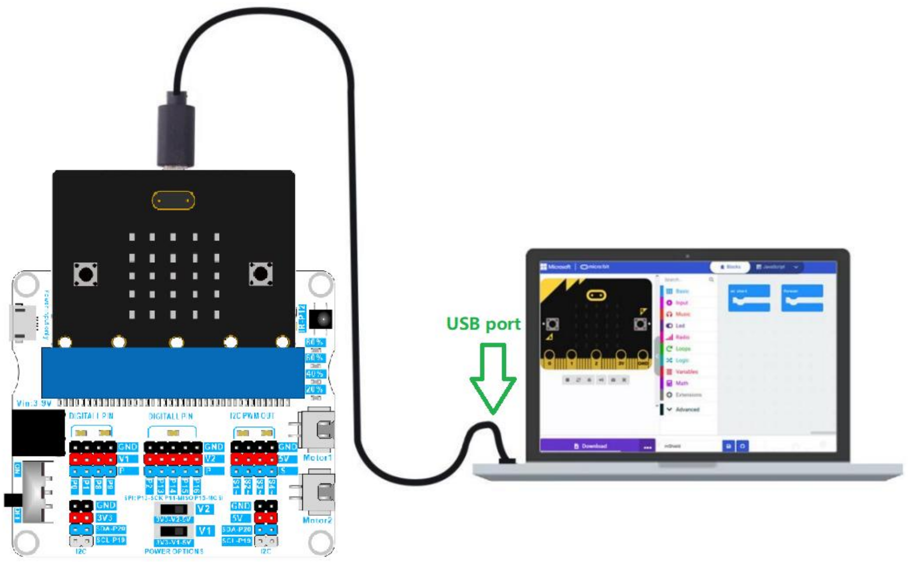
Objective

Read the firmware version of the mShield expansion.

Tools:

PC	Micro:bit V2.x.x	USB cable
		

Wiring



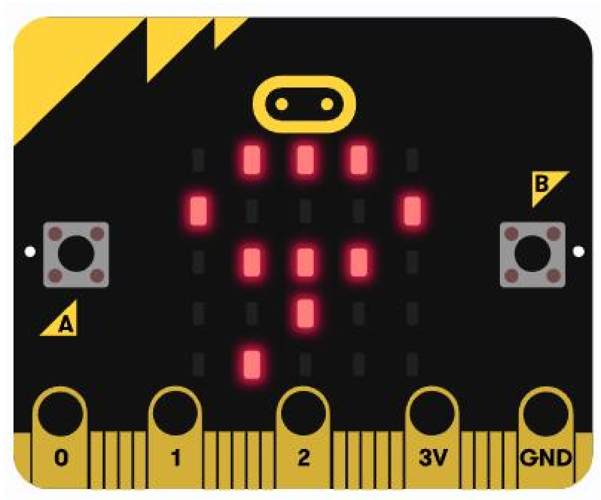
Programming

```
# Imports go at the top
from microbit import *

# Car I2C address
i2cAddr = 0x29

# Firmware version register
ver = bytearray([0x00])

# Initialize the I2C communication interface
```

```
i2c.init()


# Code in a 'while True:' loop repeats forever
while True:
    # Read the firmware version
    i2c.write(i2cAddr, ver, True)
    version = i2c.read(i2cAddr, 1)

    sleep(1000)                  # Delay: 1000 milliseconds
    display.scroll(version[0])  # Scroll display of battery level
    sleep(3000)                  # Delay: 3000 milliseconds
```

Conclusion

The mShield expansion board integrates an 8-bit microcontroller for driving motors, LEDs, and S1-S4 ports. We have pre-loaded the firmware before leaving the factory. Through the above code, you can read the firmware version and display the version number on the micro:bit LED matrix.



Thinking

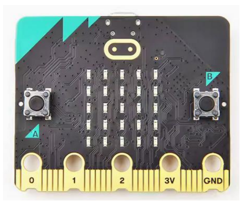
Common Issues

6.2 LEDs

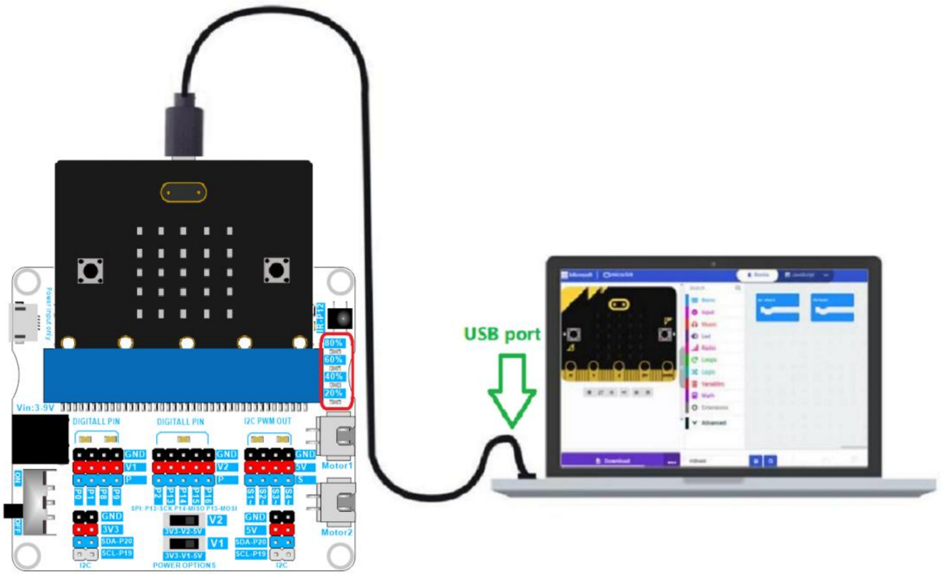
Objective

Light up the 4 LEDs on the expansion board.

Tools

PC	Micro:bit V2.x.x	USB cable
		

Wiring



Programming

```
# Imports go at the top
from microbit import *

# Car I2C address
i2cAddr = 0x29
# For LEDs
led1  = 11       #20%
led2  = 12       #40%
led3  = 13       #60%
```

```
led4  = 14       #80%
i2cBuf = bytearray([0x00, 0x00])

# Control the LED function
# led: led1 -- led4
# onOff: 0 = off, 1 = on
def setLed(led, onOff):
    if led == led1:
        i2cBuf[0] = led1
    elif led == led2:
        i2cBuf[0] = led2
    elif led == led3:
        i2cBuf[0] = led3
    elif led == led4:
        i2cBuf[0] = led4
    else:
        return
    i2cBuf[1] = onOff
    i2c.write(i2cAddr, i2cBuf)

# Code in a 'while True:' loop repeats forever
while True:
    setLed(led1, 1)    # led1 on
    sleep(1000)
    setLed(led1, 0)    # led1 off
    setLed(led2, 1)    # led2 on
    sleep(1000)
    setLed(led2, 0)    # led2 off
    setLed(led3, 1)    # led3 on
    sleep(1000)
    setLed(led3, 0)    # led3 off
    setLed(led4, 1)    # led4 on
    sleep(1000)
    setLed(led4, 0)    # led4 off
```

Conclusion

The 4 LEDs will light up in a cycle.

Thinking

Modify the code above to make the lights cycle faster.

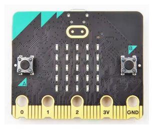
Common Issues

6.3 PWM

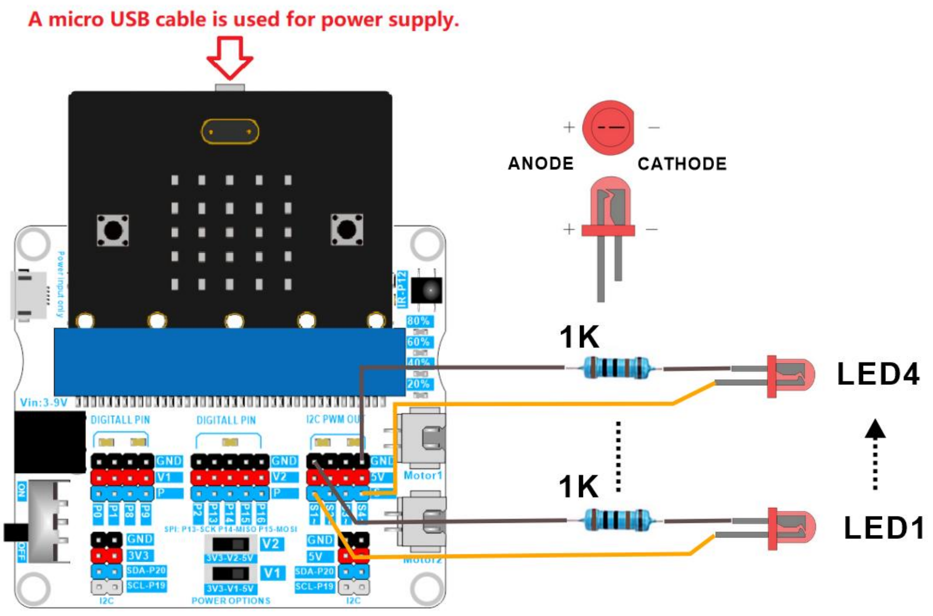
Objective

Use S1 to S4 pins to drive LEDs and change the brightness of the LEDs.

Tools

PC	Micro:bit V2.x.x	USB cable	LED
			
1K resistor	Female-to-female Dupont wires		

Wiring



Programming

```
# Imports go at the top
from microbit import *

# Car I2C address
i2cAddr = 0x29
```

```
# For Pins
s1  = 16       #S1 Pin
s2  = 17       #S2 Pin
s3  = 18       #S3 Pin
s4  = 19       #S4 Pin
i2cBuf = bytearray([0x00, 0x00])

# S1--S4 pins mode
s1ToS4 = 15
pwmMode = 1
servoMode = 2

# Control S1--S4 pins output PWM signal function
# pin: s1 -- s4
# pwm: 0 -- 200
def setPinPwm(pin, pwm):
    if pin == s1:
        i2cBuf[0] = s1
    elif pin == s2:
        i2cBuf[0] = s2
    elif pin == s3:
        i2cBuf[0] = s3
    elif pin == s4:
        i2cBuf[0] = s4
    else:
        return
    i2cBuf[1] = pwm
    i2c.write(i2cAddr, i2cBuf)

# Set the S1--S4 pin mode
# mode: 1 = PWM mode, 2 = servo mode
def setPinMode(mode):
    if mode == servoMode:
        i2cBuf[1] = servoMode
    elif mode == pwmMode:
        i2cBuf[1] = pwmMode
    else:
        return
    i2cBuf[0] = s1ToS4
    i2c.write(i2cAddr, i2cBuf)

# Set the pins S1--S4 to PWM mode
setPinMode(pwmMode)
```

```
# Code in a 'while True:' loop repeats forever
while True:
    for pwm in range(200):
        setPinPwm(s1, pwm)
        setPinPwm(s2, pwm)
        setPinPwm(s3, pwm)
        setPinPwm(s4, pwm)
        sleep(10)
```

Conclusion

LED1, LED2, LED3, and LED4 brightness slowly changes from dark to bright.

Thinking

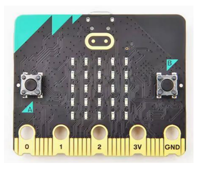
Common Issues

6.4 Battery level

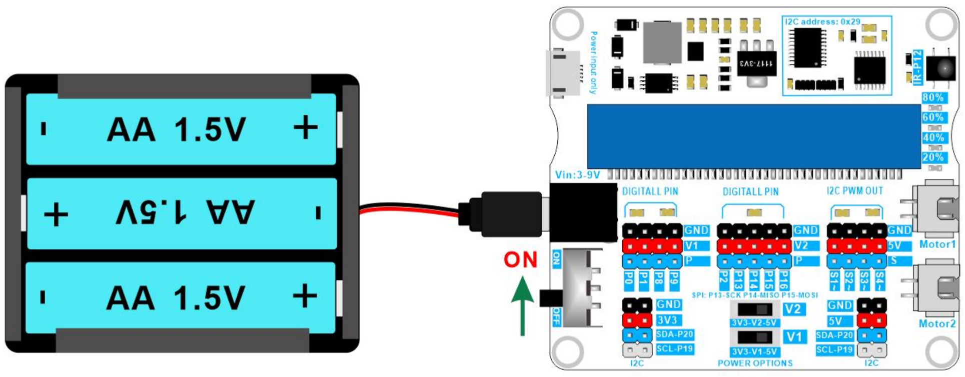
Objective

Use Pybit's 3 infrared tracking sensors to identify black and white.

Tools

PC	Micro:bit V2.x.x	USB cable	3xAA battery power
			

Wiring



Programming

```
# Imports go at the top
from microbit import *

# Car I2C address
i2cAddr = 0x29

# Type of battery
aaBattery3 = bytearray([0x01])        # 3 AA batteries
aaBattery4 = bytearray([0x02])        # 4 AA batteries
aaBattery5 = bytearray([0x03])        # 5 AA batteries
aaBattery6 = bytearray([0x04])        # 6 AA batteries
lithiumBattery1 = bytearray([0x05])   # 1 lithium battery
lithiumBattery2 = bytearray([0x06])   # 2 lithium batteries

# Initialize the I2C communication interface
i2c.init()


# Code in a 'while True:' loop repeats forever
while True:
    # Read the level of 3 AA batteries
    i2c.write(i2cAddr, aaBattery3, True)
    batLevel = i2c.read(i2cAddr, 1)

    sleep(1000)                  # Delay: 1000 milliseconds
    display.scroll(batLevel[0])  # Scroll display of battery level
    sleep(3000)                  # Delay: 3000 milliseconds
```

Conclusion

When the mShield is connected to three AA batteries in series, the micro:bit can read the battery voltage ratio (0-100%) and display this value on the micro:bit LED matrix.

Thinking

Please combine with section 6.2 to write a battery level indicator program, where the 4 LEDs represent 20%, 40%, 60%, and 80% battery levels respectively.

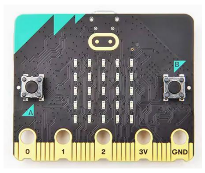
Common Issues

6.5 Motor

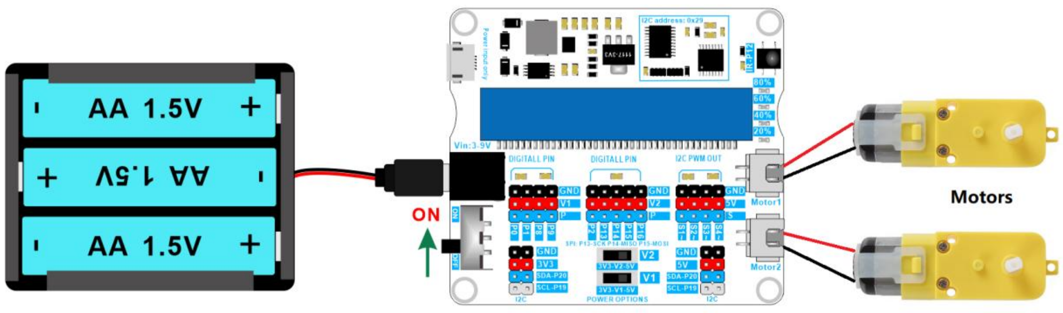
Objective

Use the 2 motor interfaces on the expansion board to drive 2 motors.

Tools

PC	Micro:bit V2.x.x	USB cable	Motor
			
3xAA battery power			

Wiring



Programming

```
# Imports go at the top
from microbit import *

# Car I2C address
i2cAddr = 0x29

# For motor
motor1  = 1
motor2 = 2
CW   = 1    # clockwise
CCW  = 2    # counterclockwise
i2cBuf = bytearray([0x00, 0x00])

# Set the motor speed function
# motor: 1 = motor1, 2 = motor2
# direction: 1 = CW, 2 = CCW
# speed: 0--100
def setMotorSpeed(motor, direction, speed):
    # Important! The speed is between 0 and 100.
    if speed > 100:
        speed = 100
    elif speed < 0:
        speed = 0

    if motor == motor1:
        i2cBuf[0] = 0x09  # motor1 register
        if direction == CW:
            i2cBuf[1] = speed
        elif direction == CCW:
            # speed value, 101 is the default required data.
            i2cBuf[1] = speed + 101
        i2c.write(i2cAddr, i2cBuf)

    if motor == motor2:
        i2cBuf[0] = 0x0a  # motor2 register
        if direction == CW:
            i2cBuf[1] = speed
        elif direction == CCW:
            # speed value, 101 is the default required data.
            i2cBuf[1] = speed + 101
        i2c.write(i2cAddr, i2cBuf)

# Code in a 'while True:' loop repeats forever
while True:
    # CW
    setMotorSpeed(motor1, CW, 100)
    setMotorSpeed(motor2, CW, 100)
    sleep(1000)
    # stop
    setMotorSpeed(motor1, CW, 0)
    setMotorSpeed(motor2, CW, 0)
    sleep(1000)
    # CCW
    setMotorSpeed(motor1, CCW, 100)
    setMotorSpeed(motor2, CCW, 100)
    sleep(1000)
    # stop
    setMotorSpeed(motor1, CCW, 0)
    setMotorSpeed(motor2, CCW, 0)
    sleep(1000)
```

Conclusion

When external motors are connected to the Motor1 and Motor2 interfaces, motor 1 and motor 2 first rotate clockwise for 1 second, then stop for 1 second; next, they will rotate counterclockwise for 1 second, then stop for 1 second, repeating this cycle continuously.

Thinking

Common Issues

If the compensation value is not set at the factory, it may cause the motor not to rotate. Just reset the compensation value to solve this problem!

If the speeds of the two motors connected to the expansion board's motor interfaces are different due to hardware differences in the motors, even though the speed values are the same, we can use the following statement to resolve this discrepancy. The compensated speed parameters can be permanently saved in the mShield.

```
# Imports go at the top
from microbit import *

# Car I2C address
i2cAddr = 0x29
i2cBuf = bytearray([0x00, 0x00])

# For wheel
leftwheel  = 0
rightwheel = 1

# Calibrate the wheel speed function
# leftWheelOffset:  0--10
# rightWheelOffset: 0--10
def wheelsAdjustment(leftWheelOffset, rightWheelOffset):
    # The limit variable is between 0 and 10.
    leftWheelOffset = max(0, min(10, leftWheelOffset))
    rightWheelOffset = max(0, min(10, rightWheelOffset))

    i2cBuf[0] = 0x07  # Left wheel offset register
    i2cBuf[1] = leftWheelOffset
    i2c.write(i2cAddr, i2cBuf)
    i2cBuf[0] = 0x08  # Right wheel offset register
    i2cBuf[1] = rightWheelOffset
    i2c.write(i2cAddr, i2cBuf)

# The compensation values of the left and right motors are set to 0
wheelsAdjustment(0, 0)
```

```
while True:
    None
```

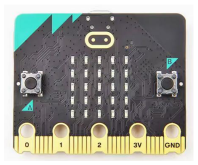
Motor speed compensation is achieved by reducing the speed of one motor to align the speeds of the two motors. The compensation value range is 0 ~ 10. This statement only needs to be executed once when the program runs, and the compensated speed parameters can be permanently saved in the mShield. Subsequent code implementations do not need to include this statement unless further adjustment of the motor compensation speed is needed.

6.6 180 degree servo

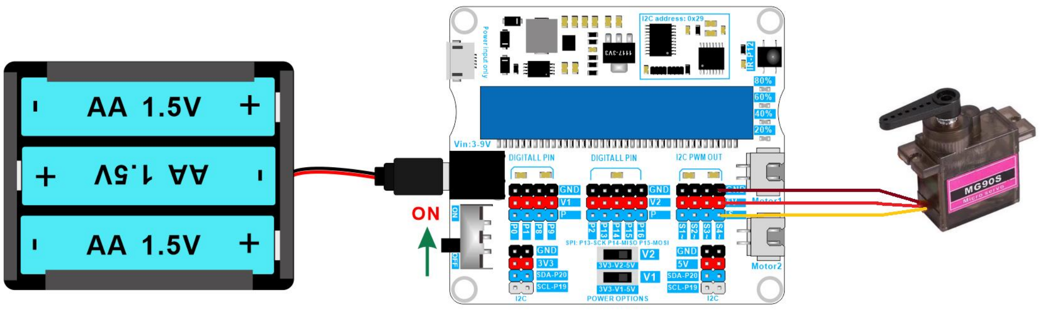
Objective

Learn how to drive a 180-degree servo.

Tools

PC	Micro:bit V2.x.x	USB cable	180 degree servo
			
3xAA battery power			

Wiring



Programming

```
# Imports go at the top
from microbit import *

# Car I2C address
i2cAddr = 0x29
i2cBuf = bytearray([0x00, 0x00])

# S1--S4 pins mode
s1ToS4 = 15
pwmMode = 1
servoMode = 2

# Servo type
servo90  = 0
servo180 = 1
servo270 = 2
pinS1 = 20    # S1 pin
pinS2 = 21    # S2 pin
pinS3 = 22    # S3 pin
pinS4 = 23    # S4 pin

# Set the S1--S4 pin mode
# mode: 1 = PWM mode, 2 = servo mode
def setPinMode(mode):
    if mode == servoMode:
        i2cBuf[1] = servoMode
    elif mode == pwmMode:
        i2cBuf[1] = pwmMode
    else:
        return
    i2cBuf[0] = s1ToS4
    i2c.write(i2cAddr, i2cBuf)

# Set the Angle function of 90, 180 and 270 servo.
# index: 20 = S1 pin, 21 = S2 pin, 22 = S3 pin, 23 = S4 pin
# servoType: 0 = 90 servo, 1 = 180 servo, 2 = 270 servo
# angle: 90 servo -> 0-90, 180 servo -> 0-180, 270 servo -> 0-270
def setServo(index, servoType, angle):
    angleMap = 0
    if servoType == servo90:
        # Map 0-90 to 50-200
        angleMap = scale(angle, from_=(0, 90), to=(50, 200))
    if servoType == servo180:
        # Map 0-90 to 50-200
        angleMap = scale(angle, from_=(0, 180), to=(50, 200))
    if servoType == servo270:
        # Map 0-90 to 50-200
        angleMap = scale(angle, from_=(0, 270), to=(50, 200))

    if index == pinS1:
        i2cBuf[0] = pinS1  # S1 pin
    if index == pinS2:
        i2cBuf[0] = pinS2  # S2 pin
    if index == pinS3:
        i2cBuf[0] = pinS3  # S3 pin
    if index == pinS4:
        i2cBuf[0] = pinS4  # S4 pin

    i2cBuf[1] = angleMap
    i2c.write(i2cAddr, i2cBuf)

# Set the pins S1--S4 to servo mode
setPinMode(servoMode)

# Code in a 'while True:' loop repeats forever
while True:
    # The 180 degree servo of the S1 pin turns to the 0 degree position
    setServo(pinS1, servo180, 0)
    # The 180 degree servo of the S2 pin turns to the 0 degree position
    setServo(pinS2, servo180, 0)
    # The 180 degree servo of the S3 pin turns to the 0 degree position
    setServo(pinS3, servo180, 0)
    # The 180 degree servo of the S4 pin turns to the 0 degree position
    setServo(pinS4, servo180, 0)
    sleep(1000)

    # The 180 degree servo of the S1 pin turns to the 180 degree position
    setServo(pinS1, servo180, 180)
    # The 180 degree servo of the S2 pin turns to the 180 degree position
    setServo(pinS2, servo180, 180)
    # The 180 degree servo of the S3 pin turns to the 180 degree position
    setServo(pinS3, servo180, 180)
    # The 180 degree servo of the S4 pin turns to the 180 degree position
    setServo(pinS4, servo180, 180)
    sleep(1000)
```

Conclusion

The servos cycle between 0 and 180 degrees.

Thinking

How to drive 90-degree and 270-degree servos?

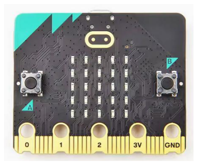
Common Issues

6.7 360 degree servo

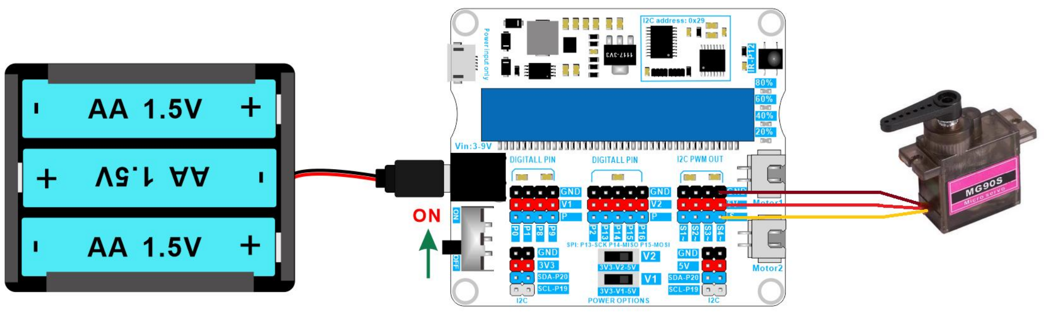
Objective

Learn how to drive a 360-degree servo.

Tools

PC	Micro:bit V2.x.x	USB cable	360 degree servo
			
3xAA battery power			

Wiring



Programming

```
# Imports go at the top
from microbit import *

# Car I2C address
i2cAddr = 0x29
i2cBuf = bytearray([0x00, 0x00])

# S1--S4 pins mode
s1ToS4 = 15
pwmMode = 1
servoMode = 2

# Servo type
servo90  = 0
servo180 = 1
servo270 = 2
pinS1 = 20    # S1 pin
pinS2 = 21    # S2 pin
pinS3 = 22    # S3 pin
pinS4 = 23    # S4 pin

# Set the S1--S4 pin mode
# mode: 1 = PWM mode, 2 = servo mode
def setPinMode(mode):
    if mode == servoMode:
        i2cBuf[1] = servoMode
    elif mode == pwmMode:
        i2cBuf[1] = pwmMode
    else:
        return
    i2cBuf[0] = s1ToS4
    i2c.write(i2cAddr, i2cBuf)

# Set the Angle function of 90, 180 and 270 servo.
# index: 20 = S1 pin, 21 = S2 pin, 22 = S3 pin, 23 = S4 pin
# servoType: 0 = 90 servo, 1 = 180 servo, 2 = 270 servo
# angle: 90 servo -> 0-90, 180 servo -> 0-180, 270 servo -> 0-270
def setServo(index, servoType, angle):
    angleMap = 0
    if servoType == servo90:
        # Map 0-90 to 50-200
        angleMap = scale(angle, from_=(0, 90), to=(50, 200))
    if servoType == servo180:
        # Map 0-90 to 50-200
        angleMap = scale(angle, from_=(0, 180), to=(50, 200))
    if servoType == servo270:
        # Map 0-90 to 50-200
        angleMap = scale(angle, from_=(0, 270), to=(50, 200))

    if index == pinS1:
        i2cBuf[0] = pinS1  # S1 pin
    if index == pinS2:
        i2cBuf[0] = pinS2  # S2 pin
    if index == pinS3:
        i2cBuf[0] = pinS3  # S3 pin
    if index == pinS4:
        i2cBuf[0] = pinS4  # S4 pin

    i2cBuf[1] = angleMap
    i2c.write(i2cAddr, i2cBuf)

# Set the speed of 360 servo
# index: 20 = S1 pin, 21 = S2 pin, 22 = S3 pin, 23 = S4 pin
# speed: -100 to +100
def setServo360(index, speed):
    # Map -100 - 100 to 0 - 180
    angle = scale(speed, from_=(-100, 100), to=(0, 180))
    setServo(index, servo180, angle)

# Set the pins S1--S4 to servo mode
setPinMode(servoMode)

# Code in a 'while True:' loop repeats forever
while True:
```

```
# The 360-degree servo of the S1--S4 pins is forward
# at the maximum speed
setServo360(pinS1, 100)
setServo360(pinS2, 100)
setServo360(pinS3, 100)
setServo360(pinS4, 100)
sleep(1000)

# The 360-degree servo of the S1--S4 pins is backward
# at the maximum speed
setServo360(pinS1, -100)
setServo360(pinS2, -100)
setServo360(pinS3, -100)
setServo360(pinS4, -100)
sleep(1000)
```

Conclusion

When 360-degree servos are externally connected to the S1, S2, S3, and S4 pins, the servos cycle forward and backward at maximum speed with 1-second intervals.

Thinking

How to make the 360 servo start slowly and uniformly?

Common Issues

7. QA

7.1 Unable to upload code to micro:bit

☛ Are you using a USB cable with data communication function?

☛ Is the USB cable in good condition?

7.2 Micro:bit drive letter displays incorrectly

☛ The drive letter displays as: MAINTENANCE (normally MICROBIT). This happens because the micro:bit reset button was accidentally pressed while powering on the micro:bit, causing it to enter firmware update mode. Re-update the firmware to return to normal. You can refer to the following link for operation:

https://microbit.org/get-started/user-guide/firmware/

7.3 Motor still rotates after re-uploading code

☛ This is because the function to stop the motor in section 5.6 was not called again:

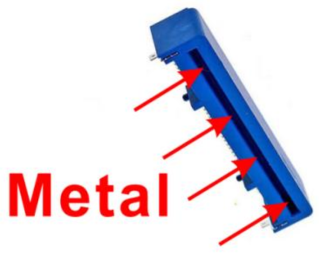
```
setWheelSpeed(wheel, direction, 0)
```

7.4 Motor not working

☛ Ensure the micro:bit is fully inserted into the Pybit's slot.

☛ Ensure the micro:bit edge connector is clean.

☛ Ensure the Pybit's slot is clean.



☛ Please refer to the common issues in section 6.5 and reset the wheel compensation value to 0.

7.5 Other failures

☛Please check if the assembly is correct?

☛Please check if the battery power is sufficient?

☛Please check if the batteries used meet the specifications?

8. Contact Us

If you cannot find the solution above, please contact our support team.

To help us assist you quickly, please have the following information ready:

Your Order Number.

Product model (e.g., mShield expansion board M1E0002) and software (e.g., MakeCode or MicroPython).

A detailed description of the problem or question.

The steps you have already tried.

Any relevant error message screenshots, photos, or code snippets.

Other inquiries?

We value your feedback and are always seeking to improve. Please feel free to contact:

Tutorial errors and feedback: Help us improve our documentation.

Product ideas and suggestions: We'd love to hear your great ideas.

Partnership and collaboration: Interested in collaborating? Let's talk.

Discounts and promotions: Inquire about educational or bulk pricing.

Anything else: For all other non-technical questions.

Support Channels

✉ support@siyeenove.com

🌐 https://siyeenove.com