
Ezblock

sunfounder

Jun 07, 2021

CONTENTS

1	Quick User Guide for Ezblock 3.0	1
1.1	About Ezblock 3	1
1.2	Download and Write Ezblock 3 image	1
1.3	Install Ezblock Studio	2
1.4	How to enter the V3.0 version?	3
1.5	How to connect the robot and Ezblock Studio?	5
1.6	How to Open and Run examples?	12
1.7	How to go back to V2.0?	14
2	Tutorials	17
2.1	Get Started	18
2.2	Remote Control	24
2.3	IOT-Sensor-Kit	29
2.3.1	Open the Example	29
2.3.2	Twilight Switch	34
2.3.3	Sound Lamp	35
2.3.4	Noisy	37
2.3.5	Theremin Organ	38
2.3.6	Magic Wand	39
2.3.7	Hot Weather Alarm	40
2.3.8	Swaying Rainbow	41
2.3.9	Smart Garage Door	42
2.3.10	Count 100	44
2.3.11	Plant Monitoring	45
3	References	47
3.1	language	47
3.1.1	Block	47
3.1.2	Python	74

QUICK USER GUIDE FOR EZBLOCK 3.0

1.1 About Ezblock 3

The biggest optimization of Ezblock App upgrade from Ezblock 2 to Ezblock 3 is to modify the communication method, v2.0 uses **Bluetooth communication**, Ezblock 3 uses **Websocket communication**, which is network communication.

Ezblock 3's network communication speed will be much faster than Ezblock 2; for the Ezblock 3 version, we have also optimized and improved the connection process and almost all interfaces to make the APP more smooth and easy to use.

The Ezblock 3 version should be used with the Ezblock 3 image ([Ezblock Studio Download Center](#)). The v3.0 version is currently in the public test stage.

If there is a problem during use, please send an email to us, and we will test it several times to ensure that there is no problem in use.

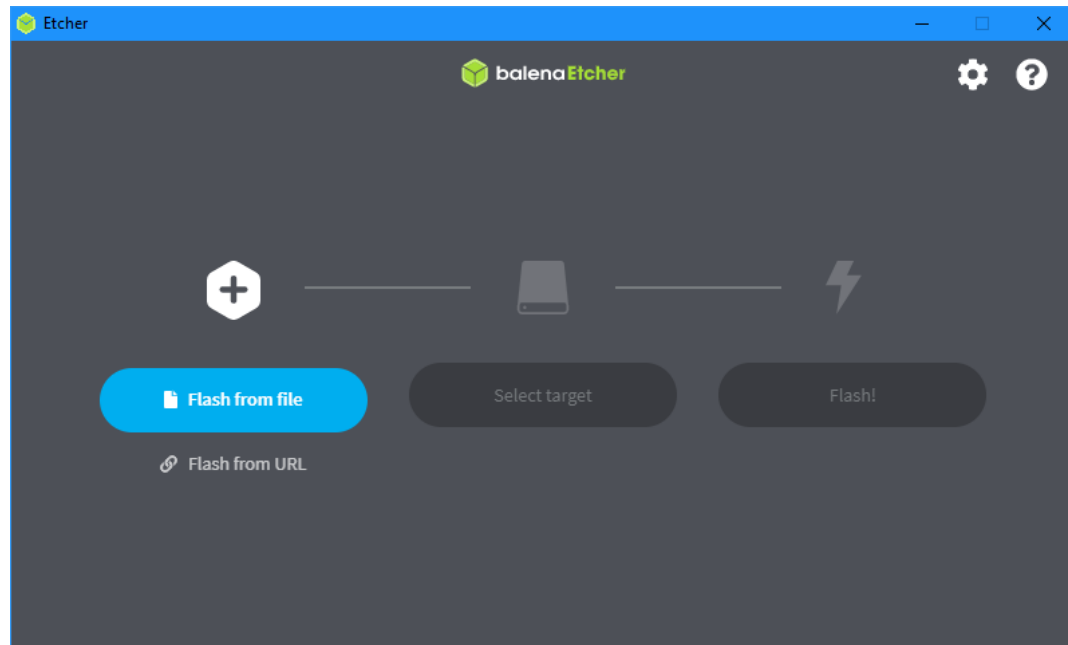
Here is the Email: cs@sunfounder.com.

1.2 Download and Write Ezblock 3 image

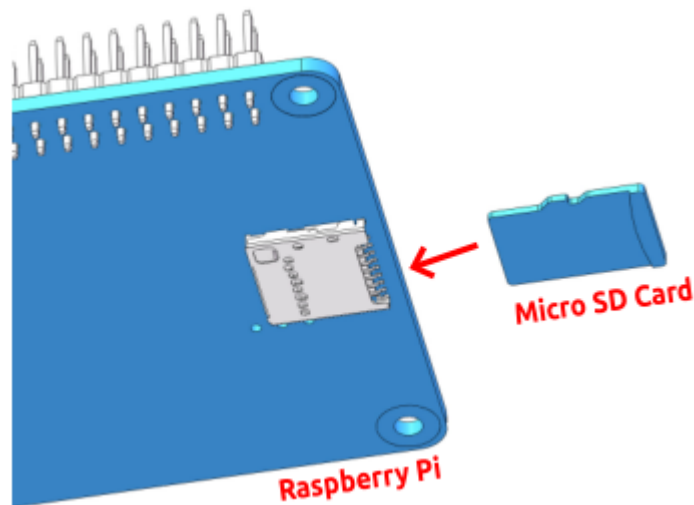
1. Prepare the tool of image burning. Here we use the Etcher. Now download the software from here: [Etcher](#).
2. Download the **Raspberry Pi OS for Ezblock 3 Beta** image file here: [Ezblock Studio Download Center](#).
3. Unzip the package downloaded and you will see the .img file inside.

Note: Do not extract the .img file.

4. With Etcher, flash the image file into the Micro SD card.



5. At this point, Ezblock for Raspberry Pi is installed. Please insert the Micro SD card into your Raspberry Pi.



1.3 Install Ezblock Studio

Ezblock Studio is a development platform developed by SunFounder designed for beginners to lower the barriers to getting started with Raspberry Pi.

It has two programming languages: Graphical and Python, and available on almost all different types of devices.

With Bluetooth and Wi-Fi support, you can download code, remote control a Raspberry Pi, on Ezblock Studio.

Open App Store (iOS/Mac OS X system) or Play Store (Android/Windows/Linux system), then search and download Ezblock Studio.

4:53 PM Fri May 21

14% [Search](#)

Ezblock Studio

SunFounder



OPEN

4 RATINGS

5.0

★★★★★

AGE

4+

Years Old

CATEGORY



Productivity

DEVELOPER



SunFounder

LANGUAGE

EN

+ 1 More

SIZE

8.9

MB

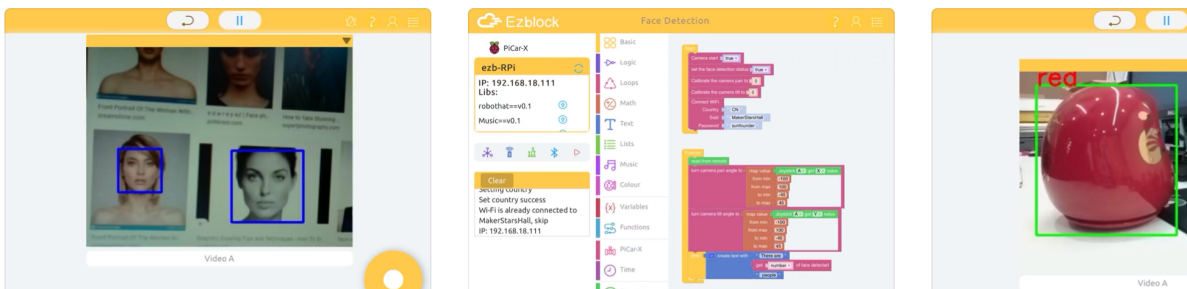
What's New

Fix multiple bugs
Remove swift language

[Version History](#)

9mo ago
Version 0.1.2

Preview



Today

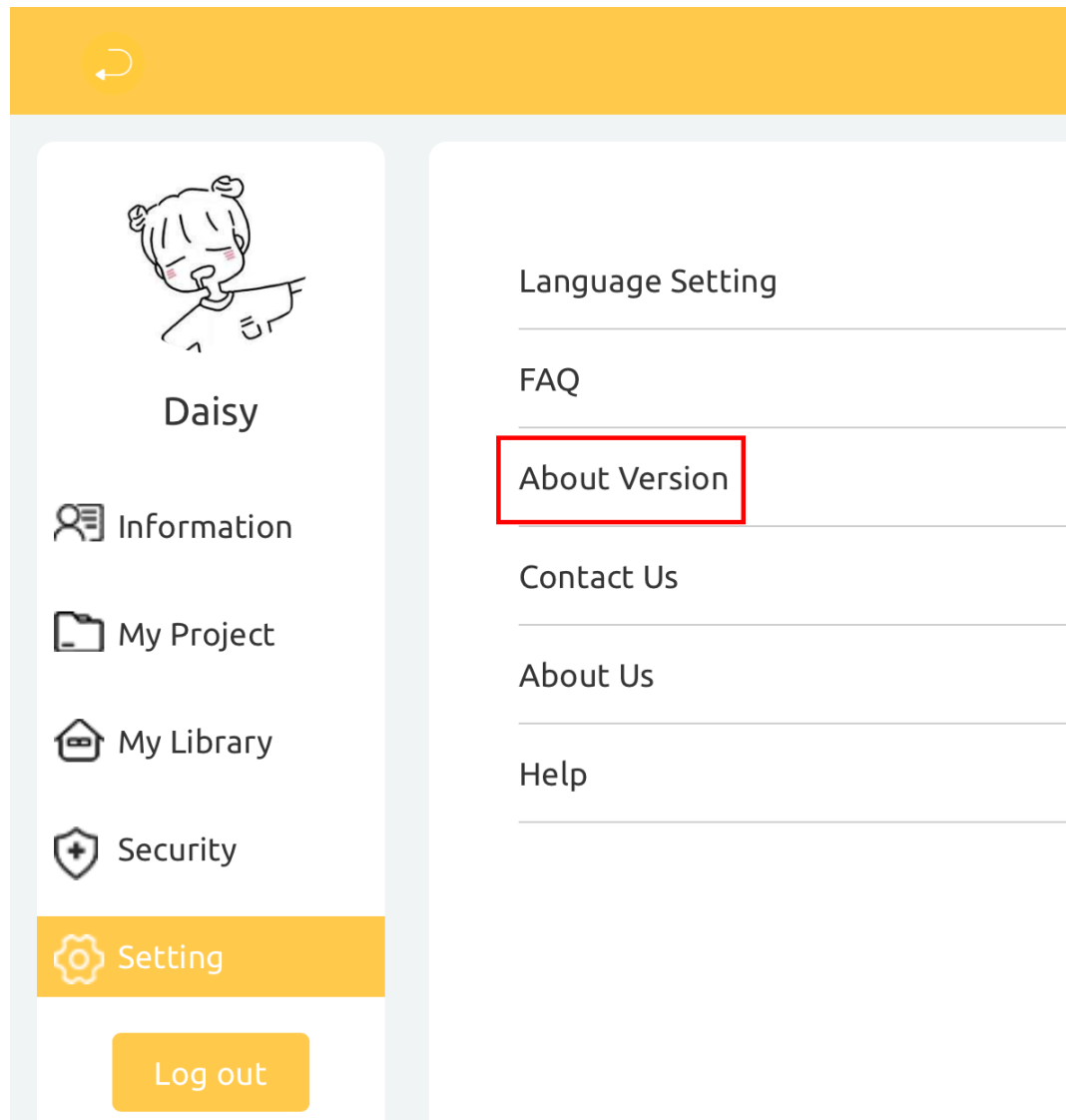
Games

Apps

Search

1.4 How to enter the V3.0 version?

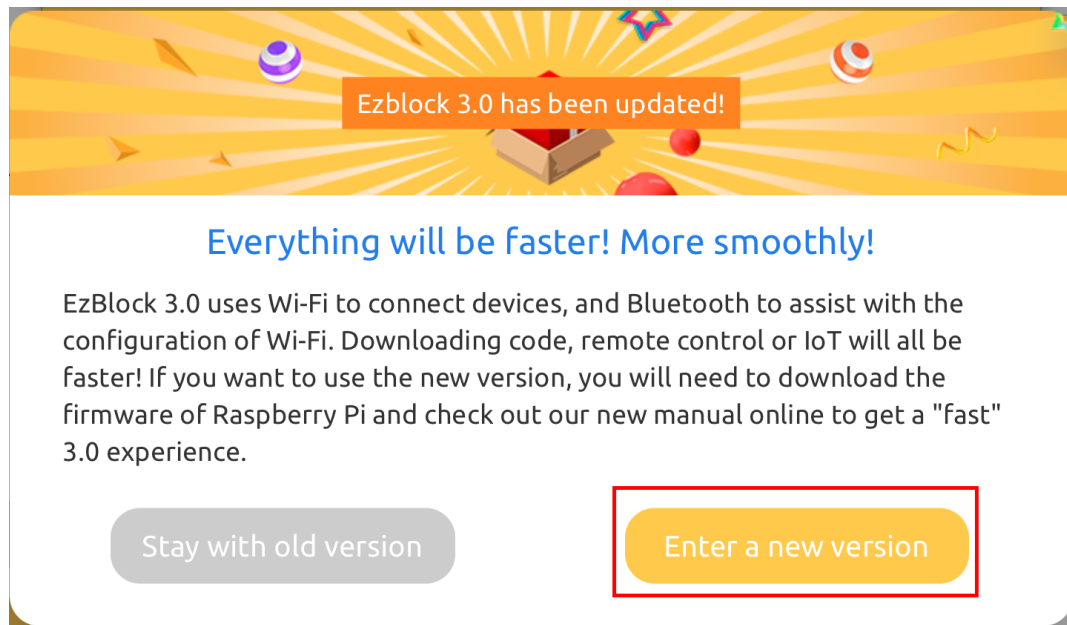
1. Open Ezblock Studio and log in to your account.
2. Go to your account page by clicking on your avatar in the upper right corner of the home page.
3. Go to the **Setting** page, and then click **About Version**.



4. Click **Enter a new version** in the pop-up window.



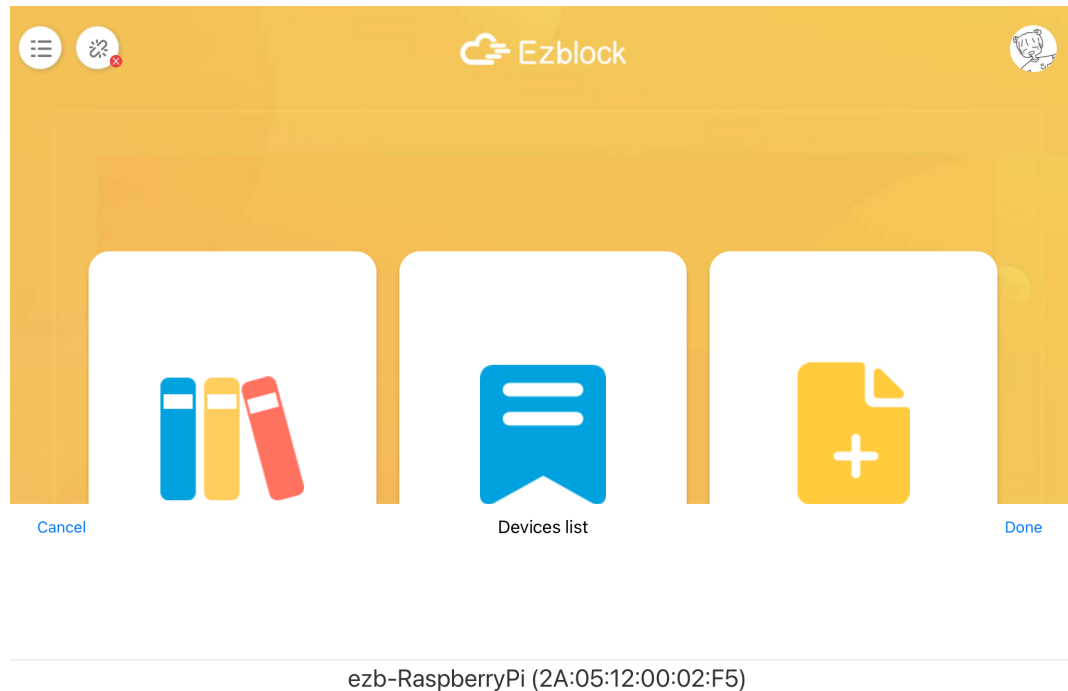
5. A new pop-up window will appear , select **Enter a new version** again.



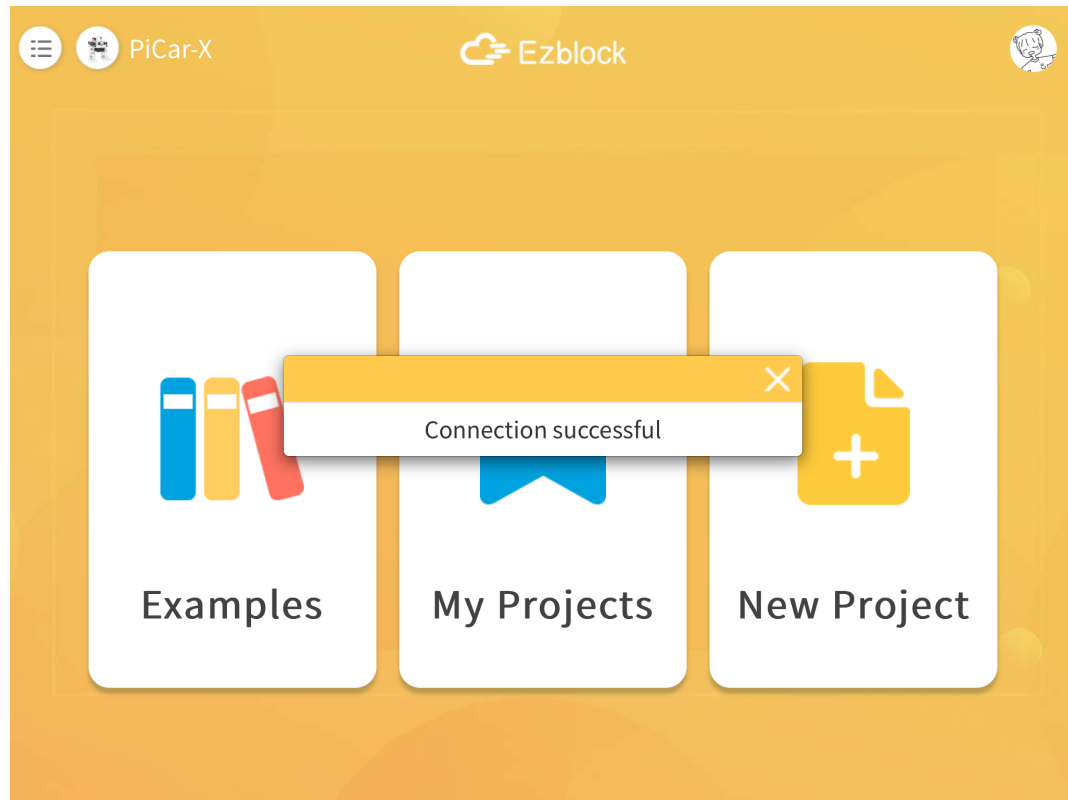
6. After a few minutes of loading, you will enter the V3.0 version.

1.5 How to connect the robot and Ezblock Studio?

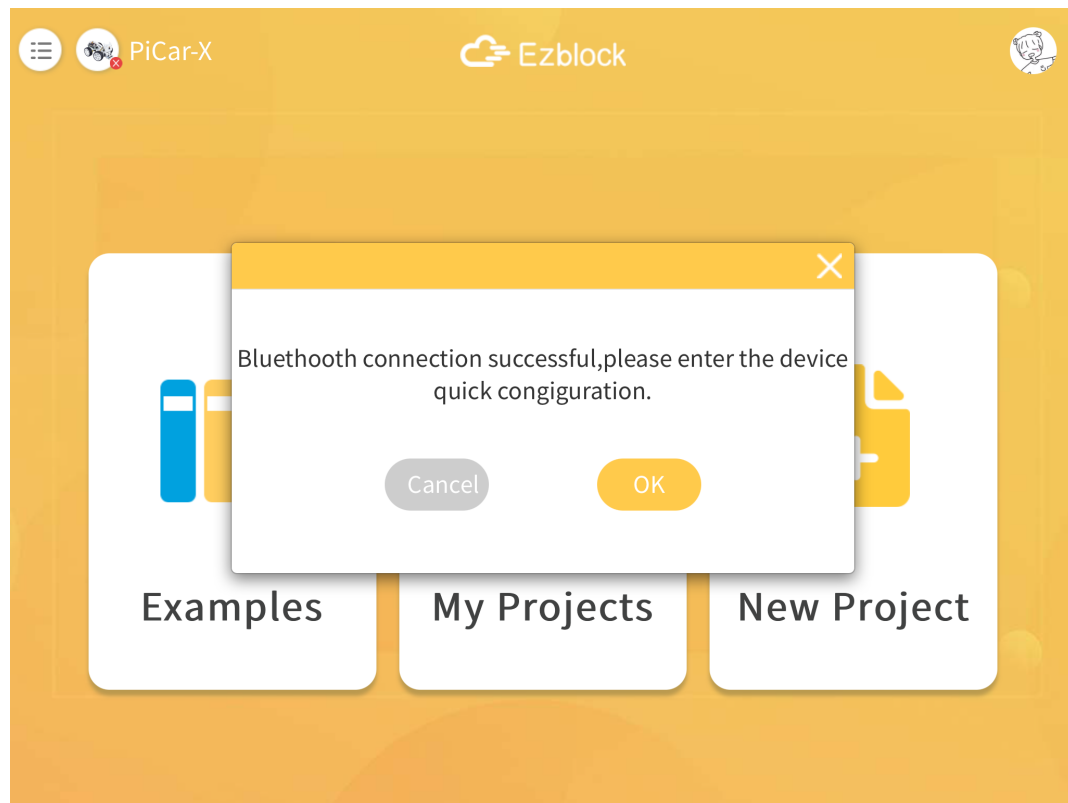
1. At this time, there is a pop-up window with an empty device list. You need to power on your robot and turn on the Bluetooth of your mobile device at the same time, then the robot number will appear.



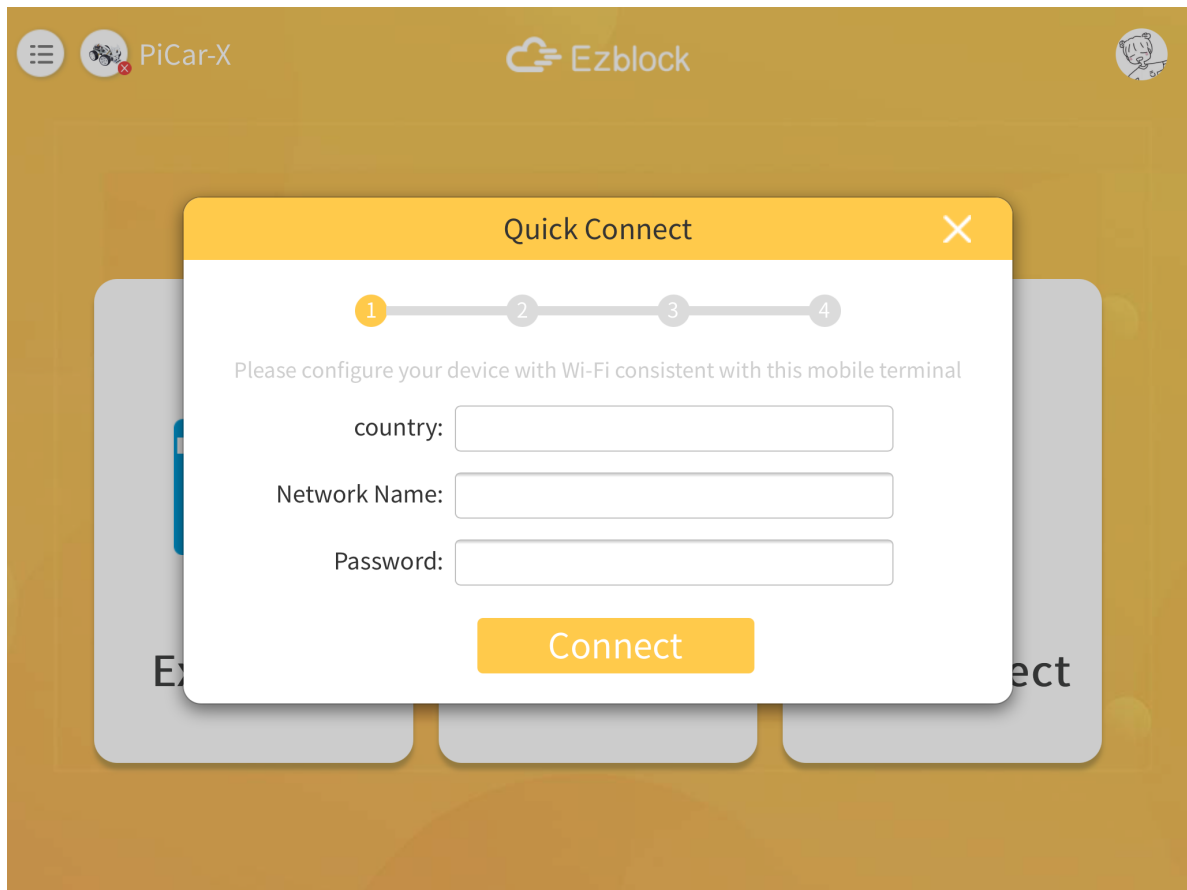
2. Click **Done** in the upper right corner, and after a while, **Connection Successful** will appear.



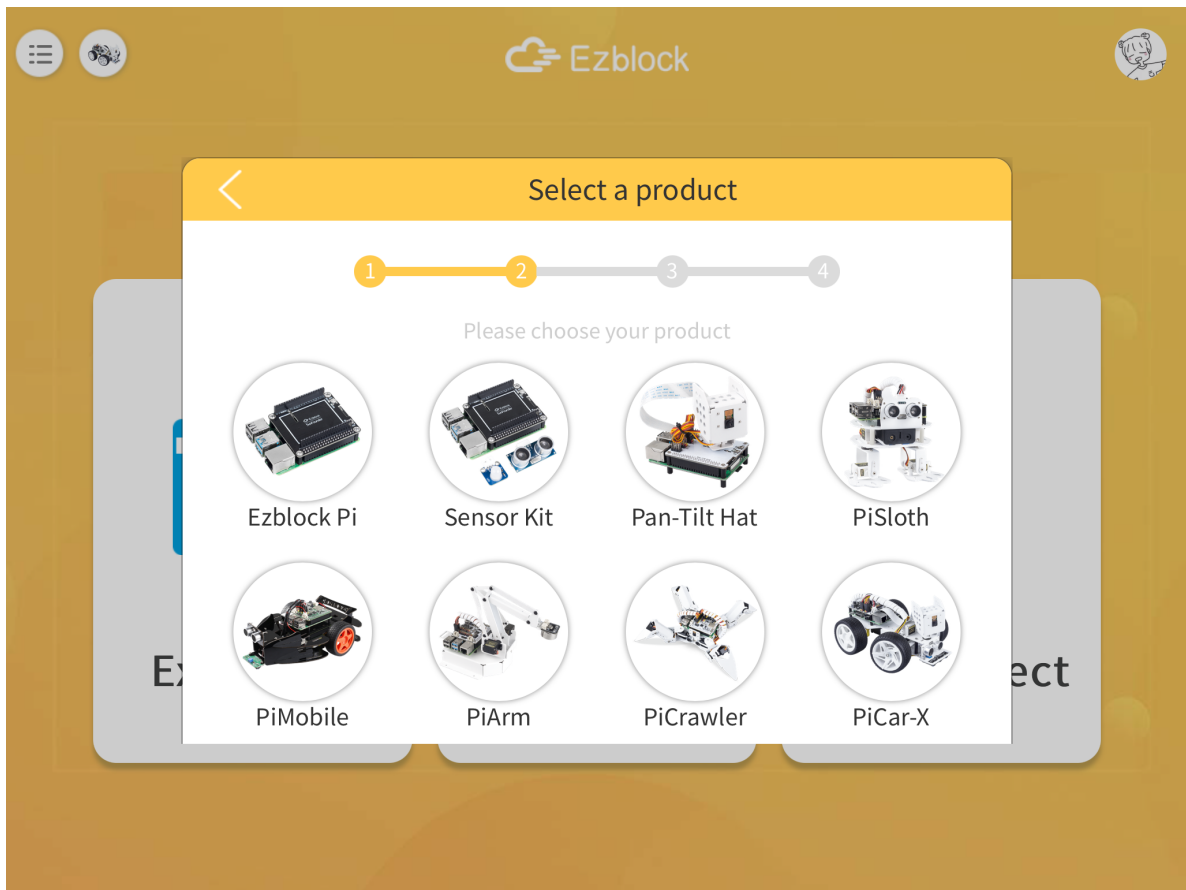
3. At this point you need to click **OK** to quickly configure your robot.



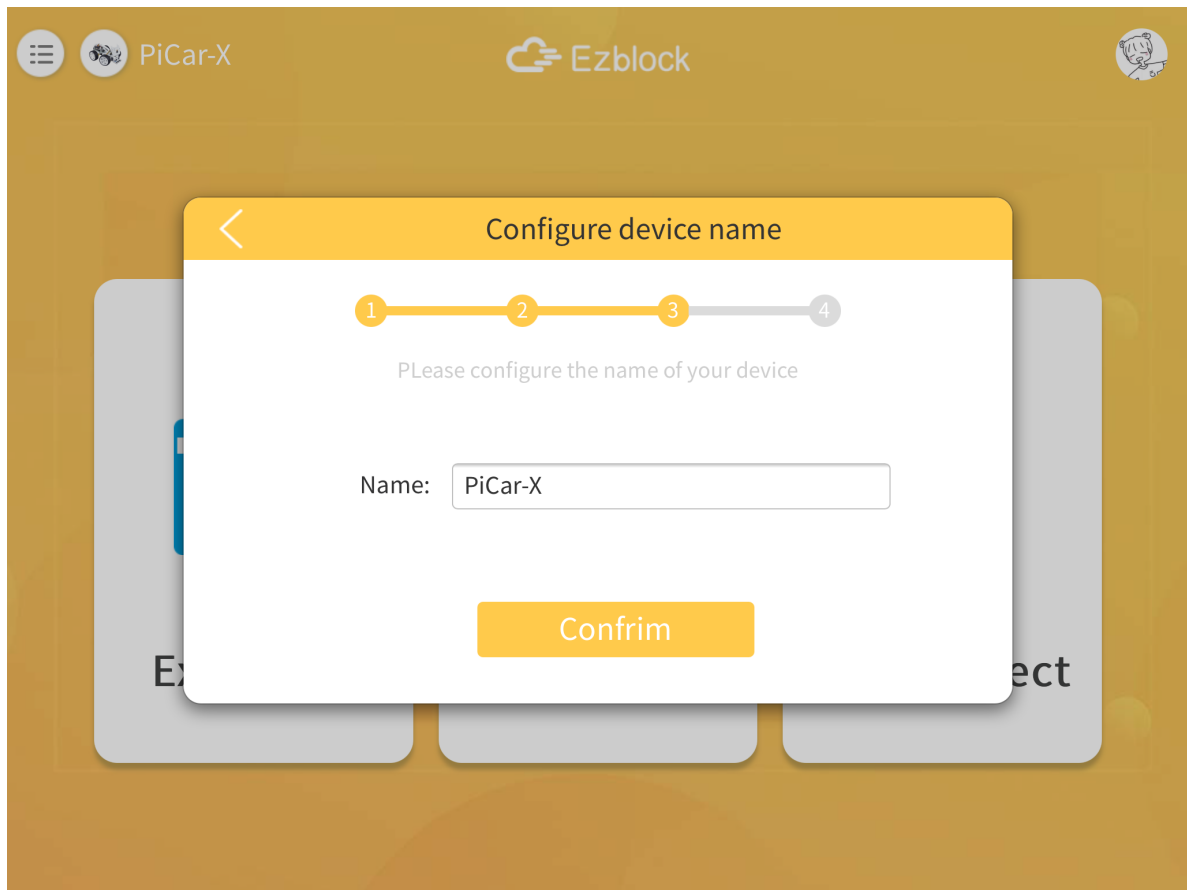
4. Enter your Wi-Fi account and password.



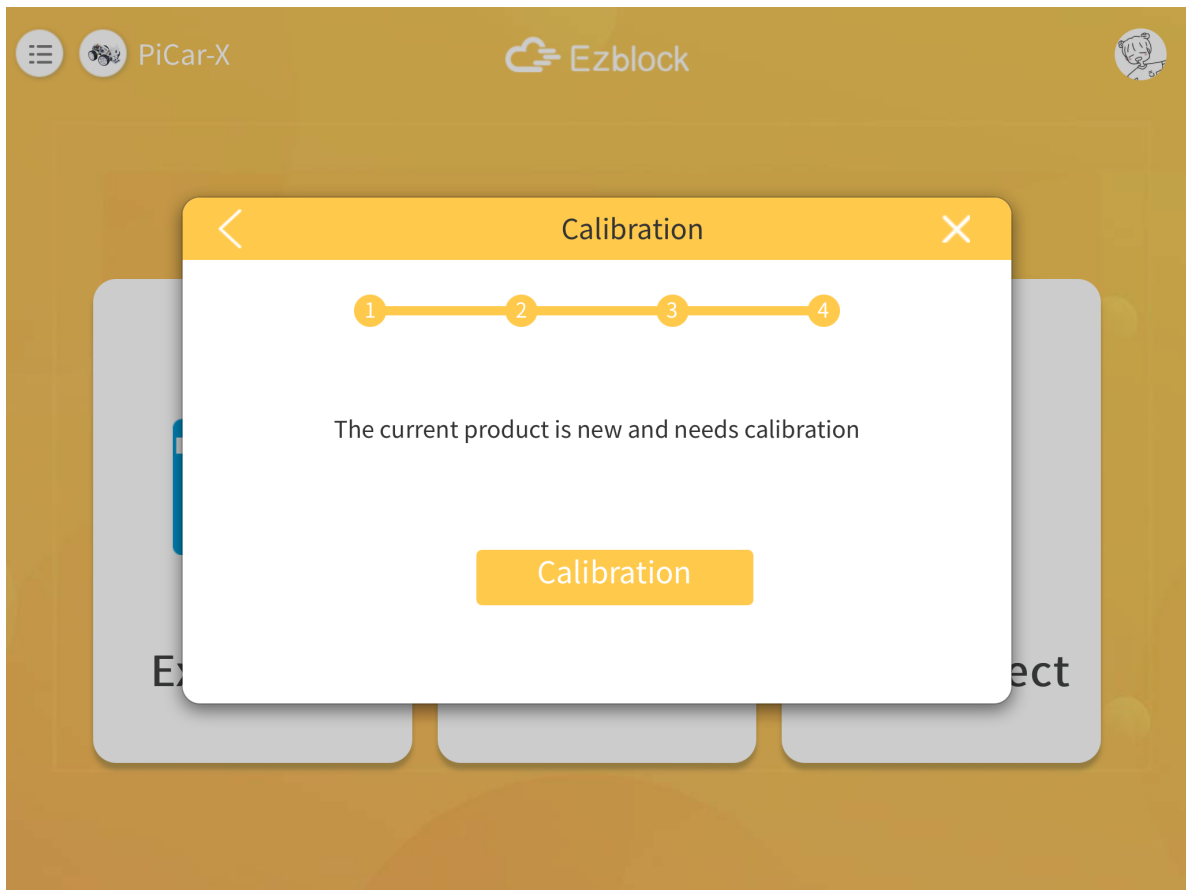
5. Choose the product corresponding to your robot.



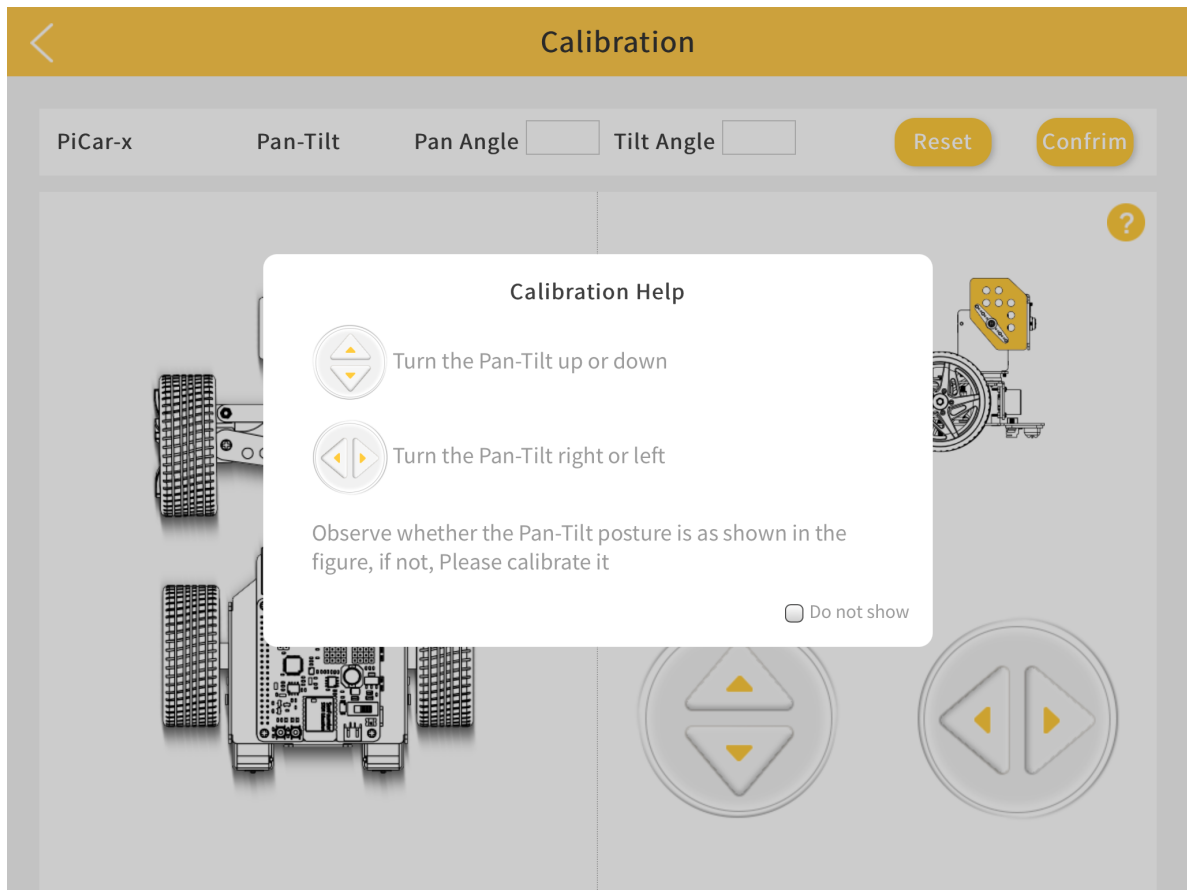
6. Enter a name for your robot.



7. If your robot needs to be calibrated, there will be a prompt telling you that you can enter the calibration page by clicking **Calibration**. If it is not needed, the pop-up window disappears and returns to the home page.

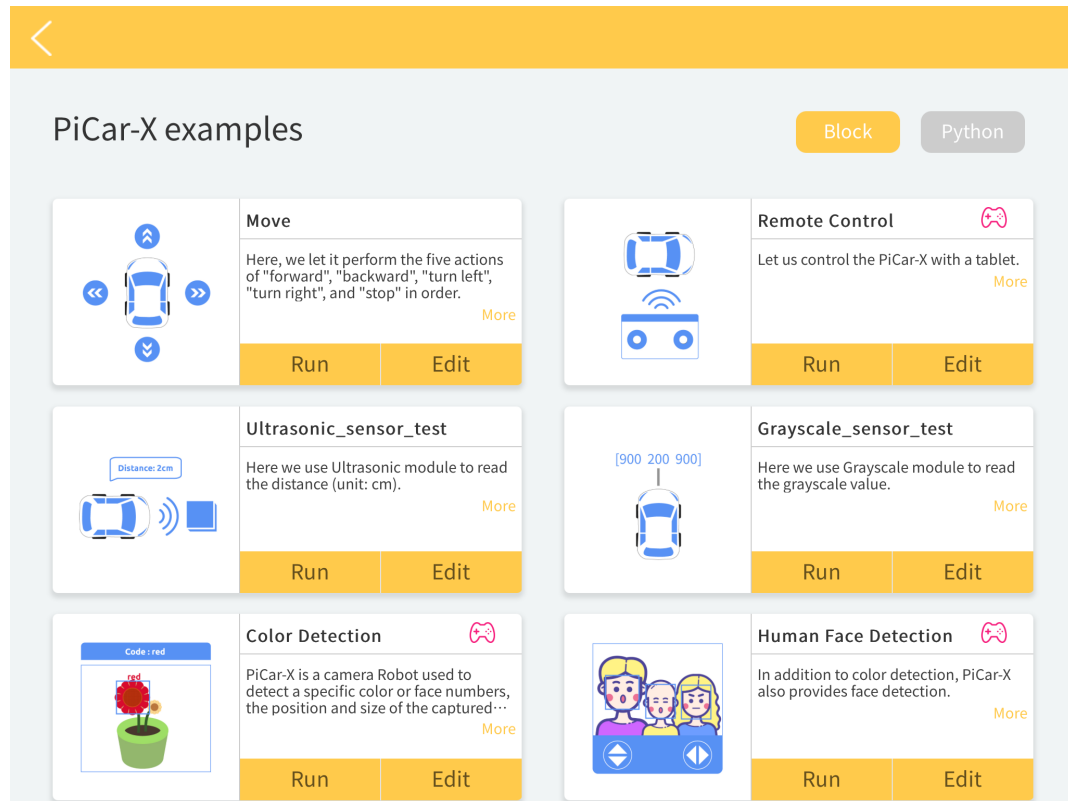


8. The calibration page of each product is different, but there is a reminder which part needs to be calibrated. You can click the corresponding part, and then refer to the **Calibration Help** to calibrate. After the calibration is completed, click **Comfirm**.

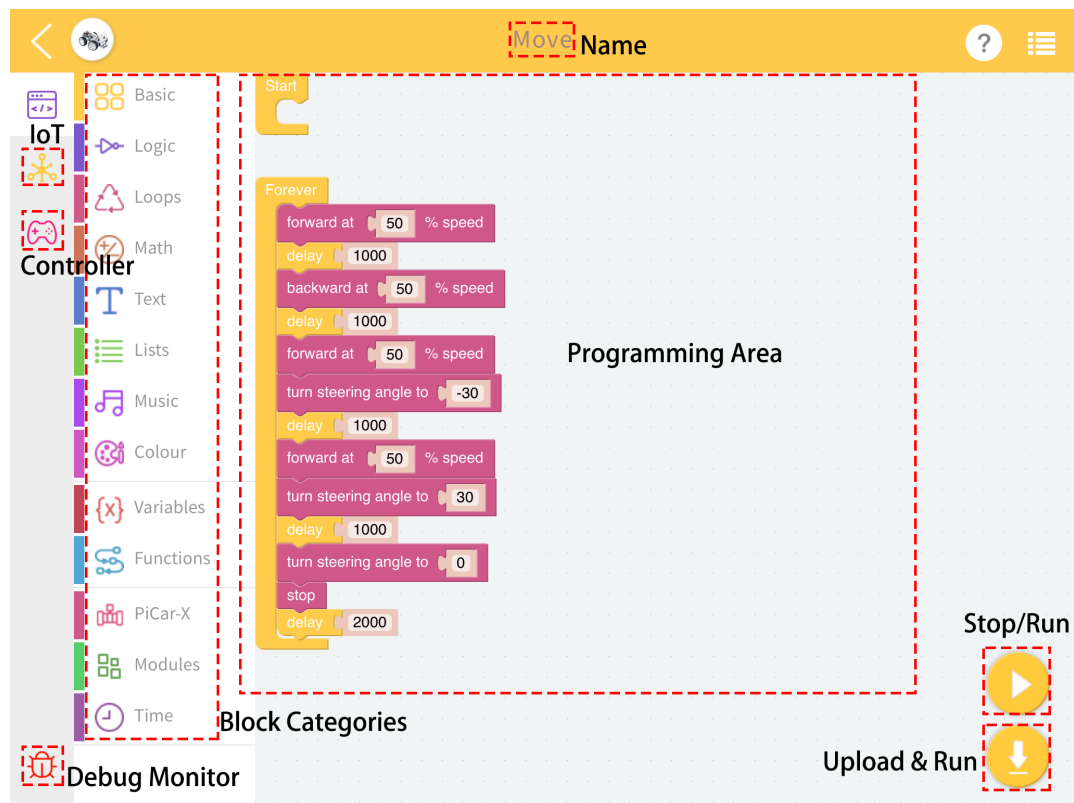


1.6 How to Open and Run examples?

1. On the homepage, click Examples to enter the Examples page. If you just need to simply test these examples, you only need to click **RUN** to make your robot work.

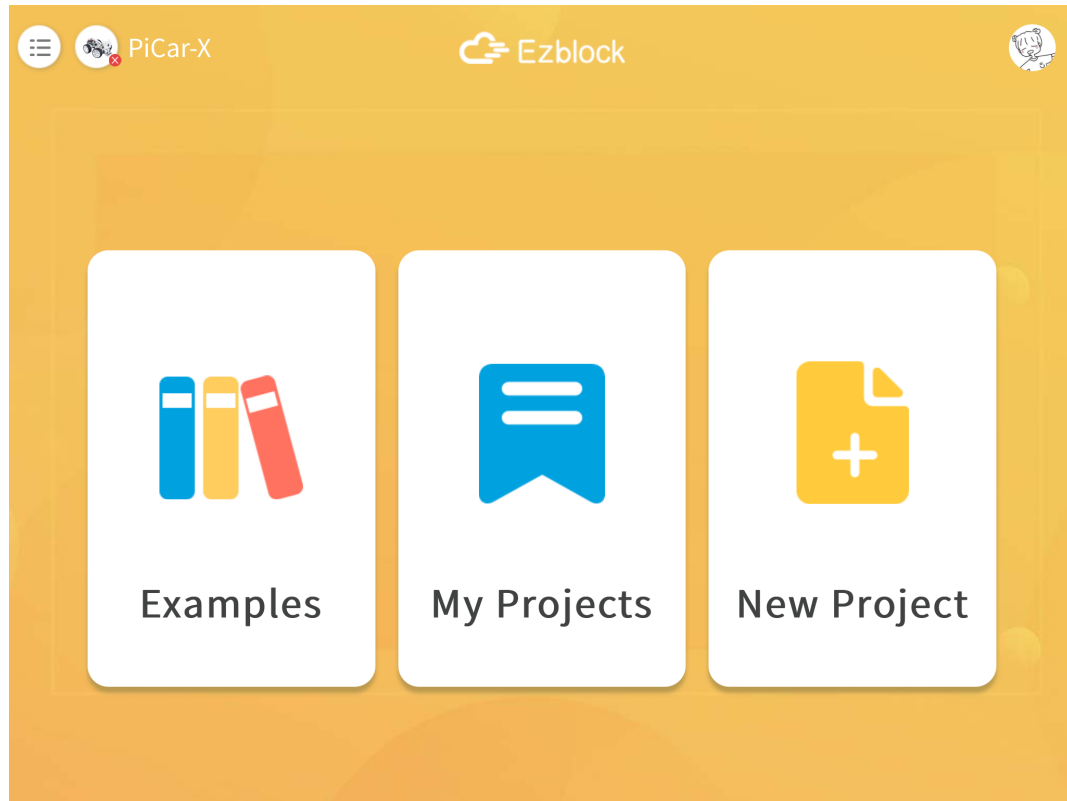


2. If you want to view and modify the code inside, then you need to click **Edit**. The following picture is the programming page.

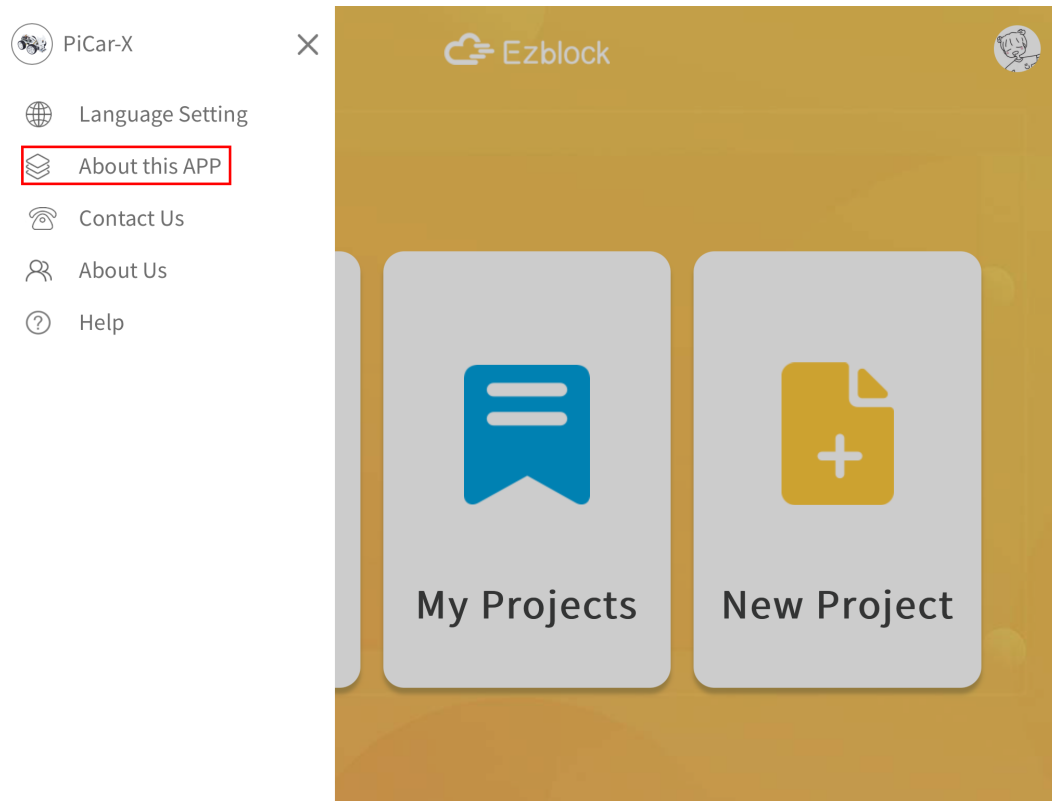


1.7 How to go back to V2.0?

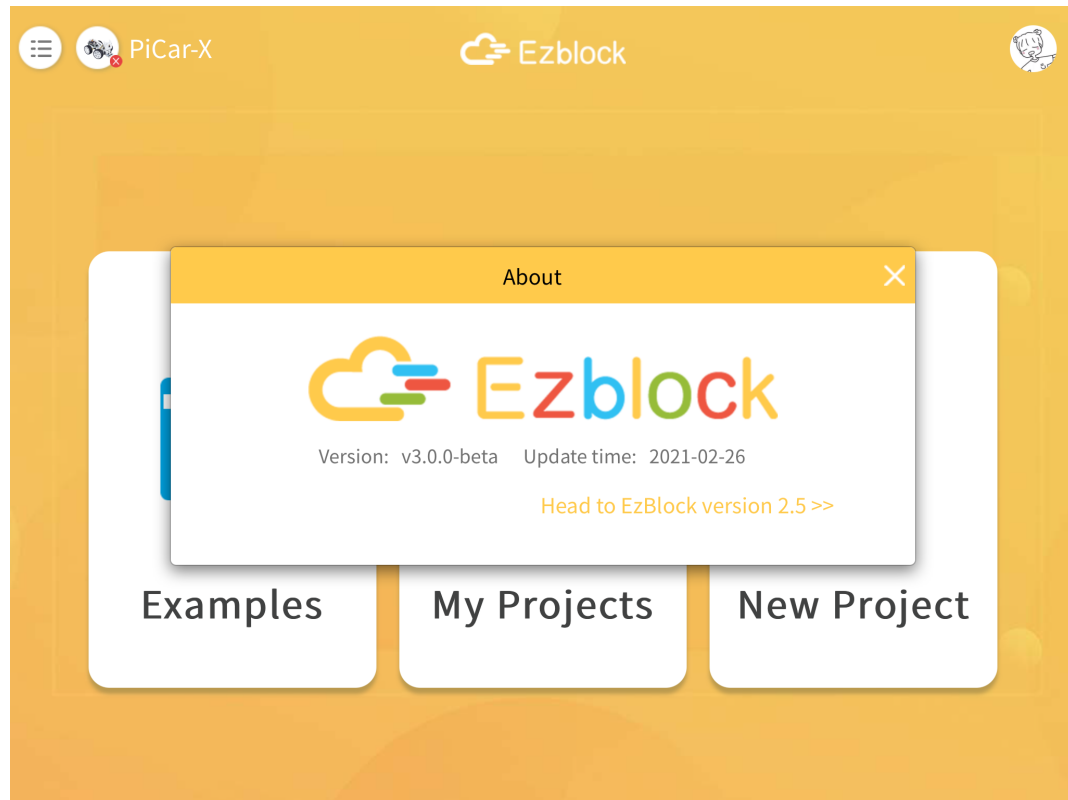
1. If you are not used to the V3.0 version of Ezblock, then you can go back to the V2.0 version.
2. Click the **menu** icon in the upper right corner of the homepage.



3. Click **About this App**.



4. Click **Head to EzBlock version 2.5** to jump to v2.0 version!



TUTORIALS

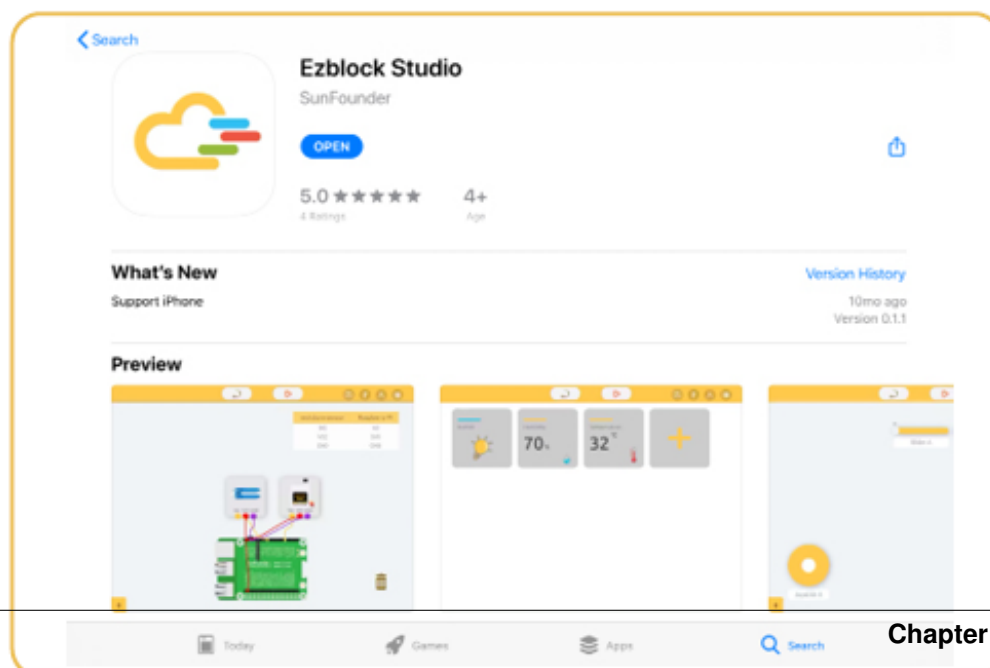
2.1 Get Started

Introduce Ezblock Studio

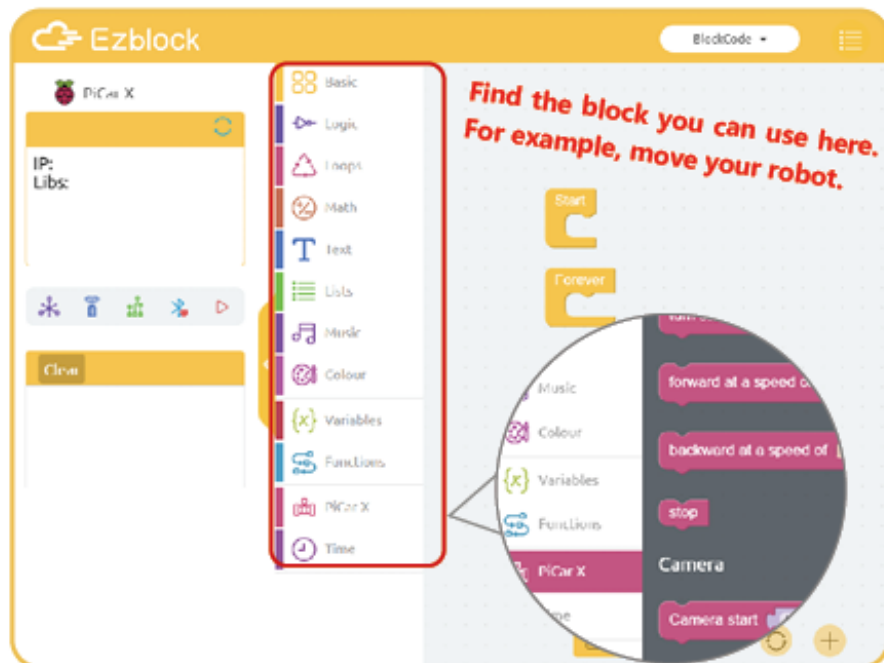
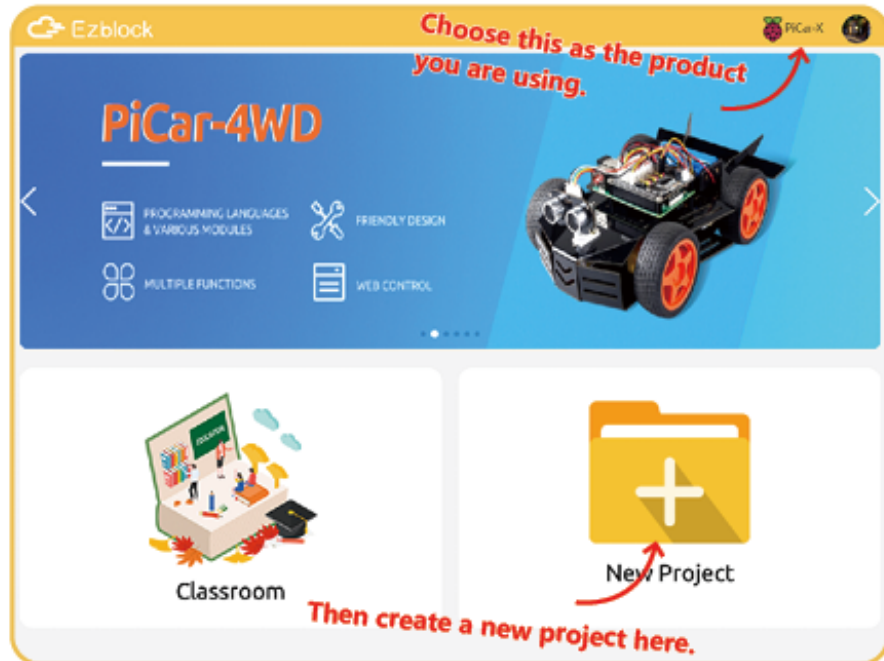
Ezblock Studio is a new open-source platform for building electronic projects and graphical programming.

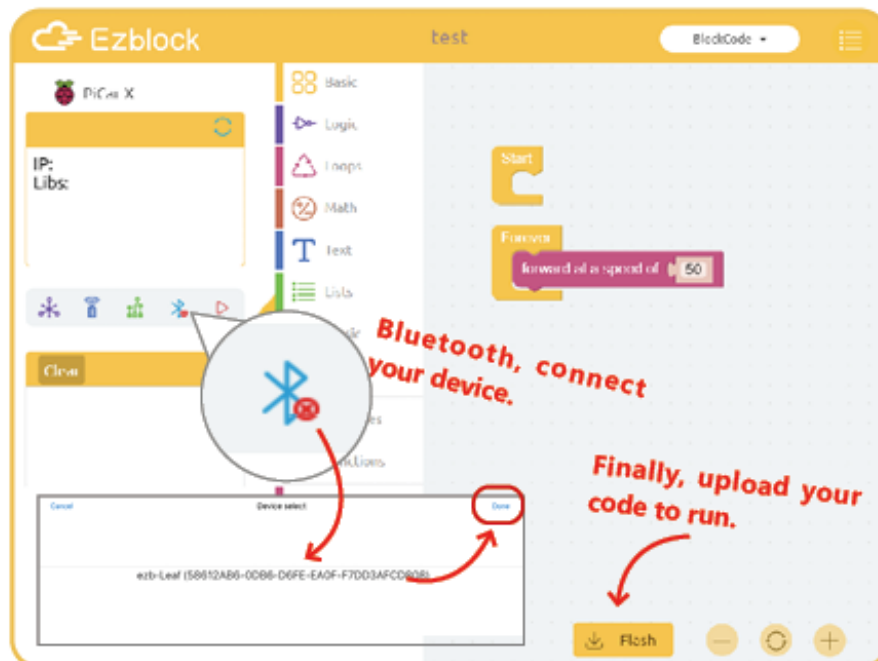
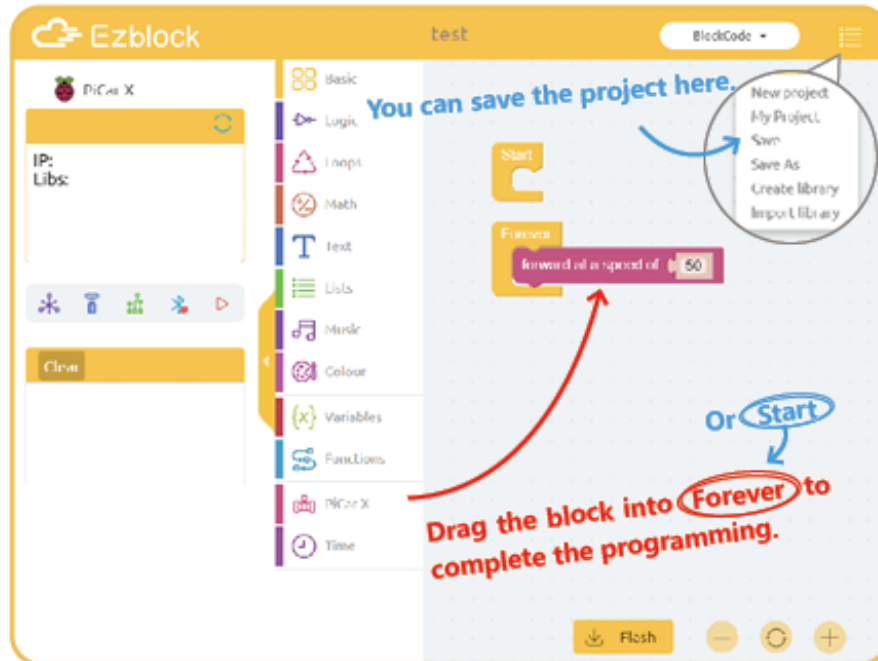
Ezblock Studio also serves as a coding platform that runs on your phone, tablet and computer. In addition, the Ezblock Studio applies Blocks and Python, making it easier to learn programming. By and large, Ezblock Studio integrates Hardware Simulator, Bluetooth Debugger, IoT Panel and Customizable Remote Controller, which are conducive to the operation of prototyping, debugging, and so on.

Open App Store (iOS/Mac OS X system) or Play Store (Android/Windows/Linux system), then search and download Ezblock Studio.



Start Using

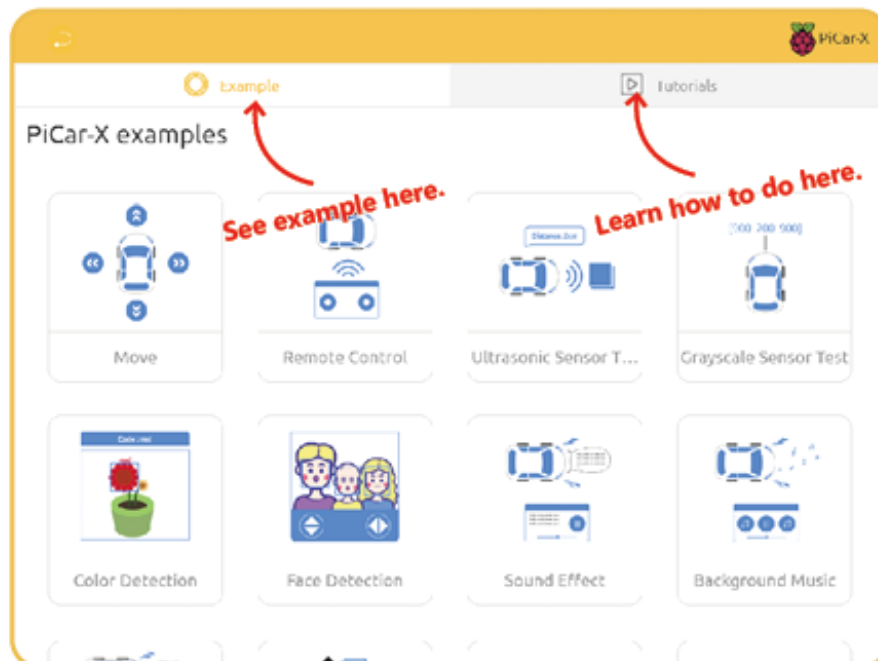
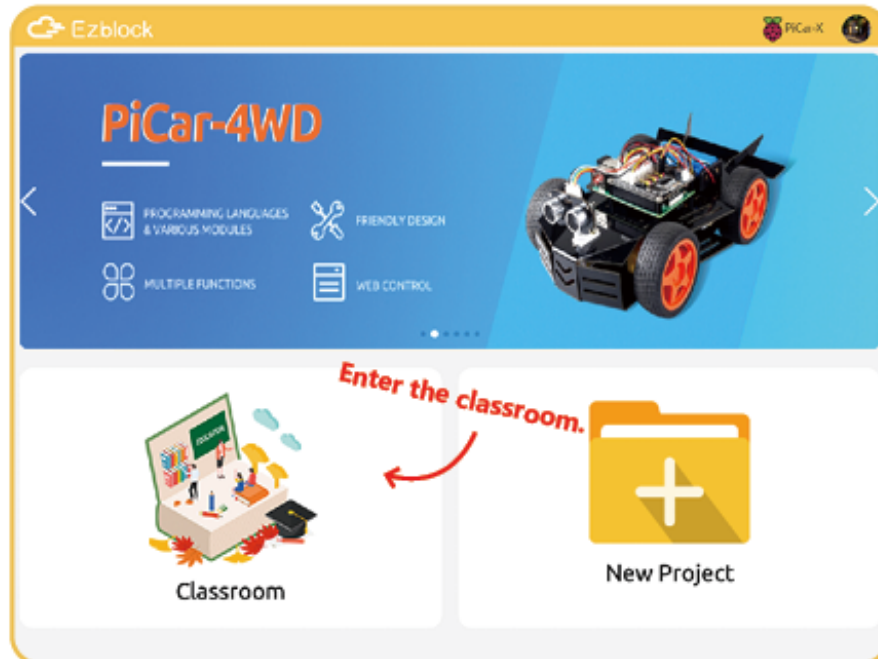




Open the Saved Project



Open the Example



2.2 Remote Control

Quick Guide on Remote Control

①. To use the remote control function, you need to enter the Bluetooth control page.

②. Drag the required control to the blank area.

③. Drag [read from remote] into Forever block.

④. Complete your program.

⑤. Flash it.

Find the category, Remote.

Go back.

Joystick

Slider

read from remote

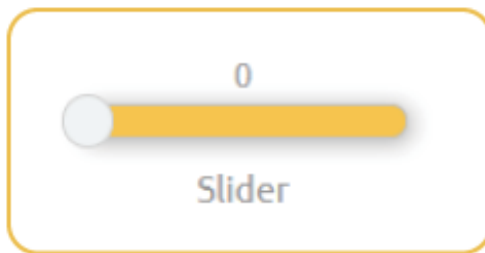
Slider B get value

Forever

do something with: x

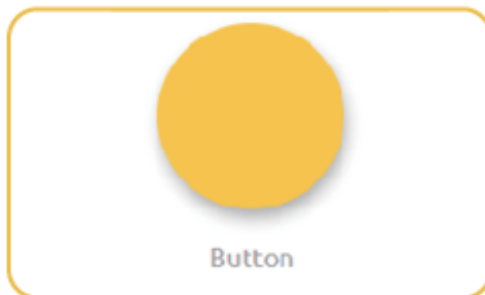
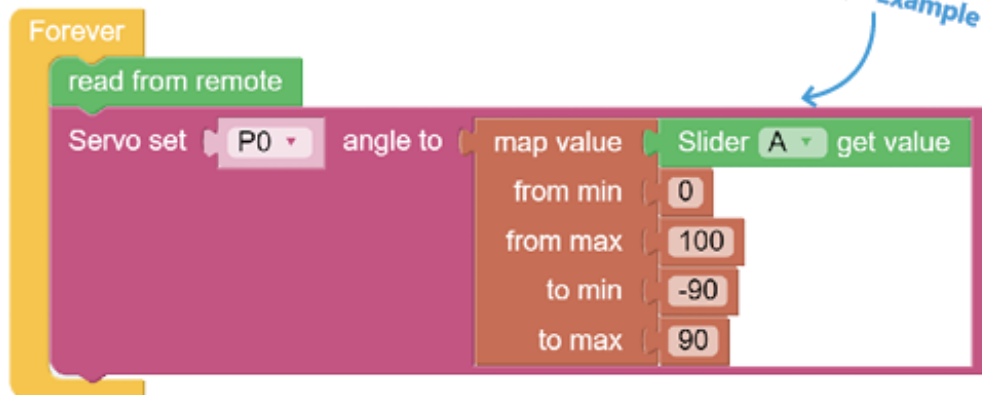
Slider B get value

Flash



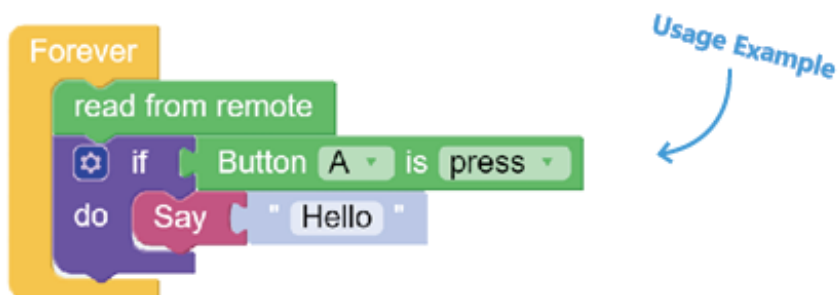
Slider can be used to adjust the speed of the robot, the angle of the steering gear and so on.

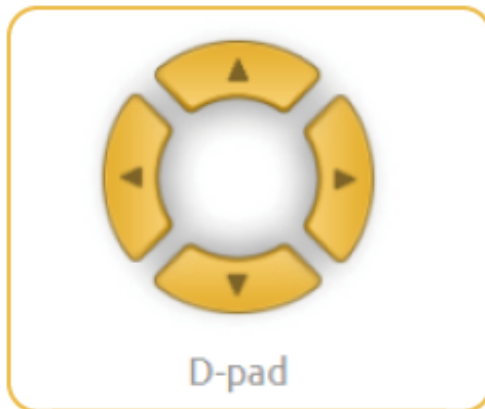
By default, the control point is on the far left and the reading is 0. The reading range is 0~100.



The button is often used to initiate specific events, such as ringing the doorbell, clearing the score, and so on.

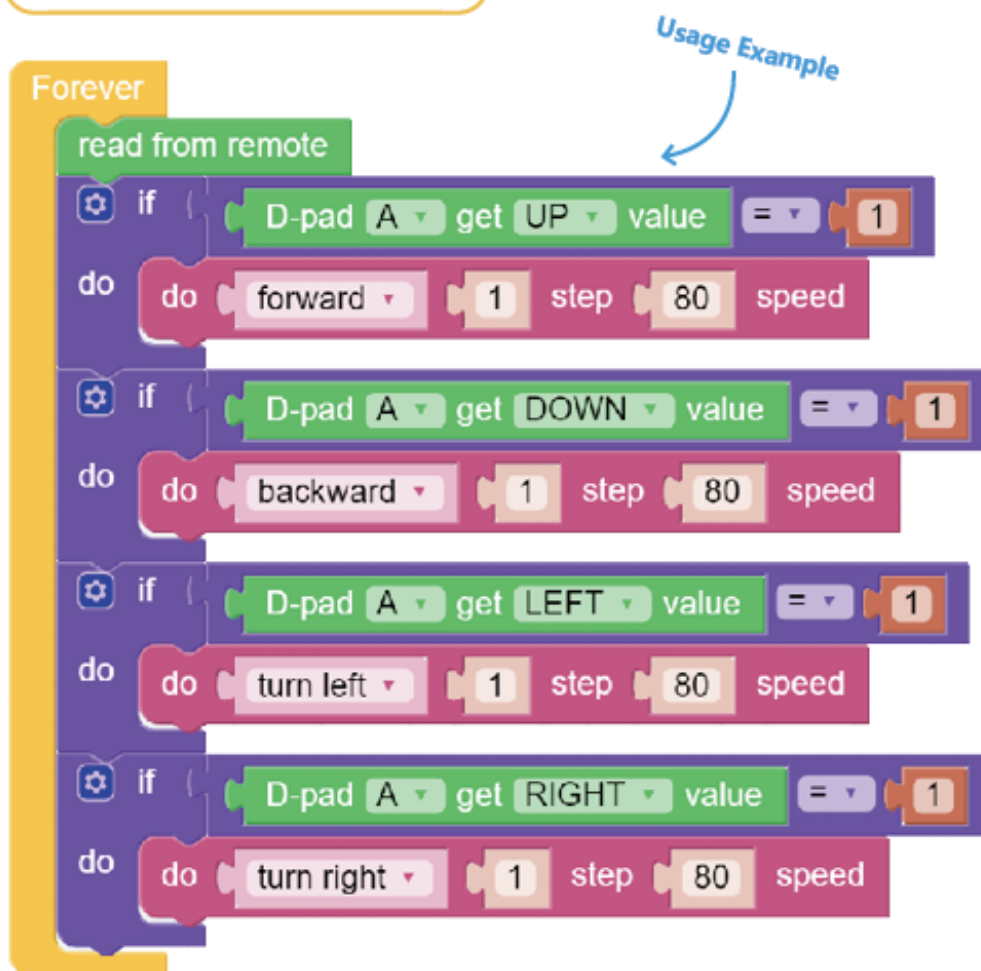
By default, the value of the button is 0, and when it is tapped, the value of the button is 1.

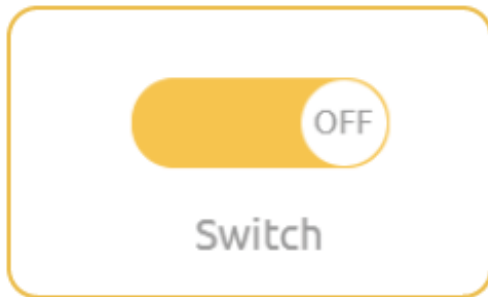




D-pad consists of four buttons, which can be used to control the movement of the robot.

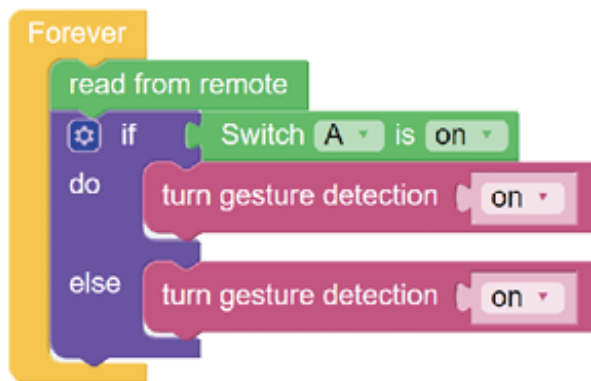
By default, the value of the D-pad button is 0, and when the button is clicked, the value of the button is 1.



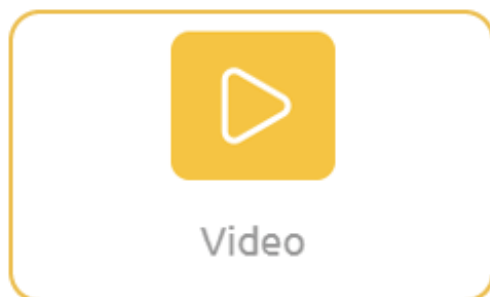


Switch is very suitable for turning on and off LEDs. You can also use it to activate specific functions.

By default, it is OFF and the value is 0; after clicking, the value is switched to ON and the value is 1.



Usage Example

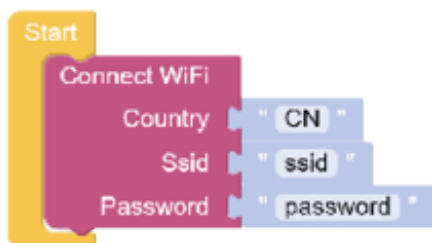


Video is used with Camera Module. It allows you to see the shot.

To use Video, your Raspberry Pi and the operating terminal must be in the same WLAN.

How to Use ?

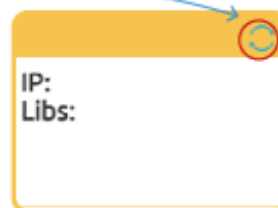
First, burn a "Connect WiFi" block in your Raspberry Pi.



Modify to your WiFi

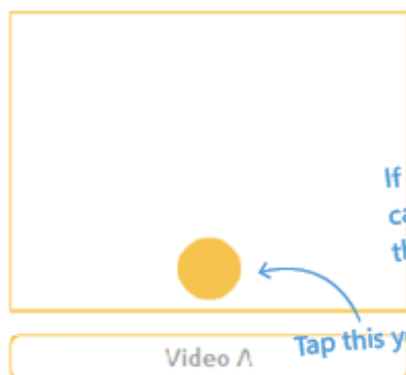
WiFi configuration requires you to click the upper left button of Ezblock to update the device information to take effect.

Then wait for about five seconds, the Debug Monitor will show the IP address.



Then, burn a "turn video monitor" block in your Raspberry Pi.

Finally, drag and drop a Video on the remote control page, and you can see the image.



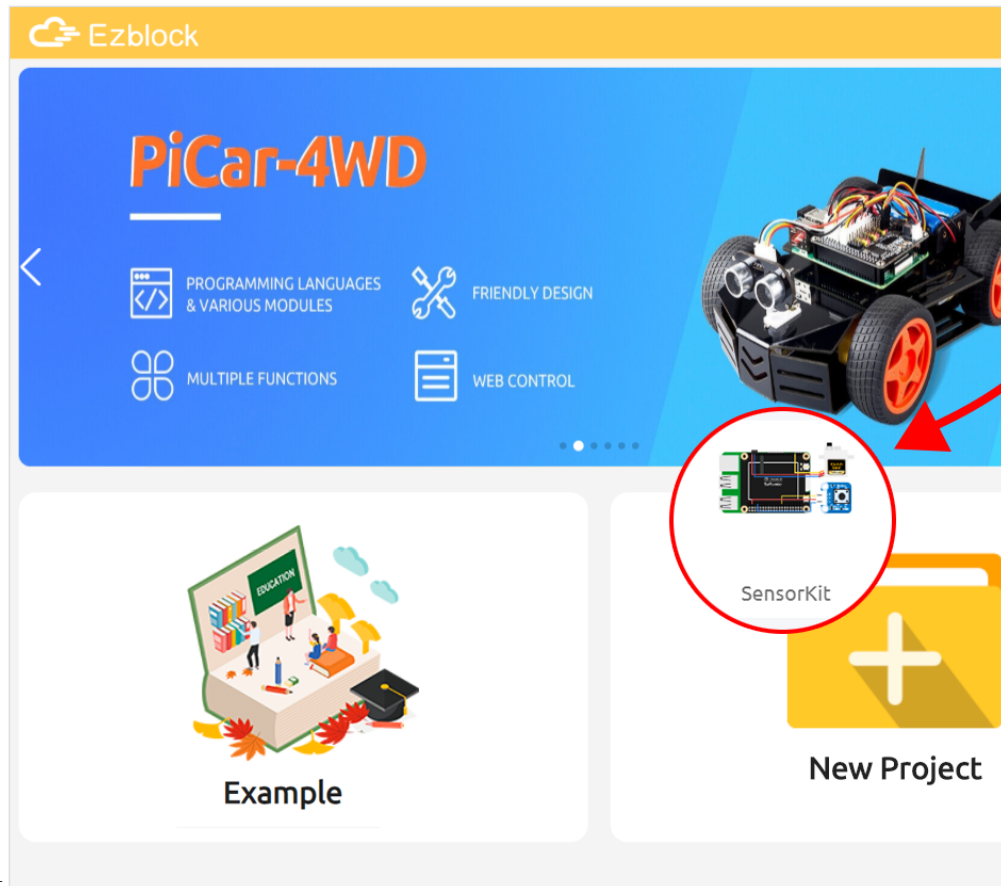
If there is no image, please make sure your camera module is properly connected, and then wait a while and try again.

Tap this yellow dot to save the picture to your tablet !

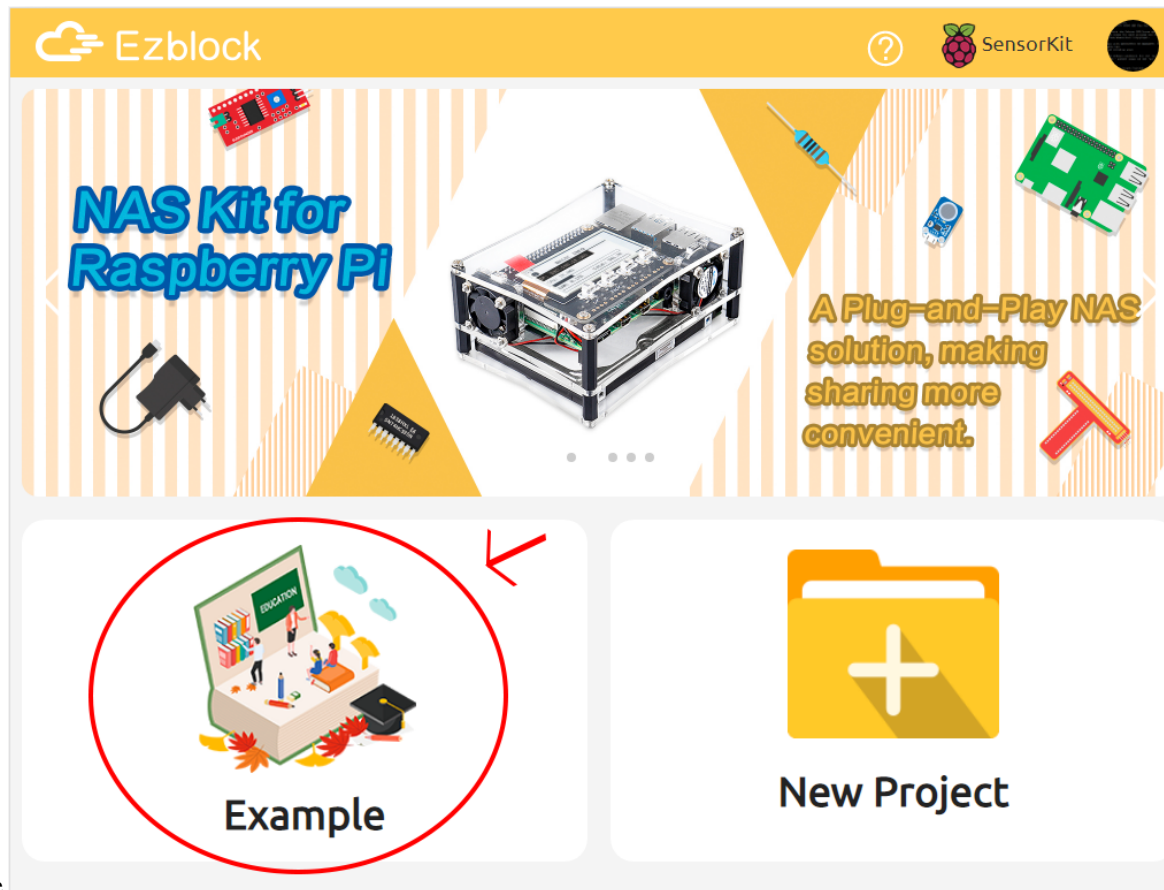
2.3 IOT-Sensor-Kit

This page show you the examples provided with IoT Sensor Kit.

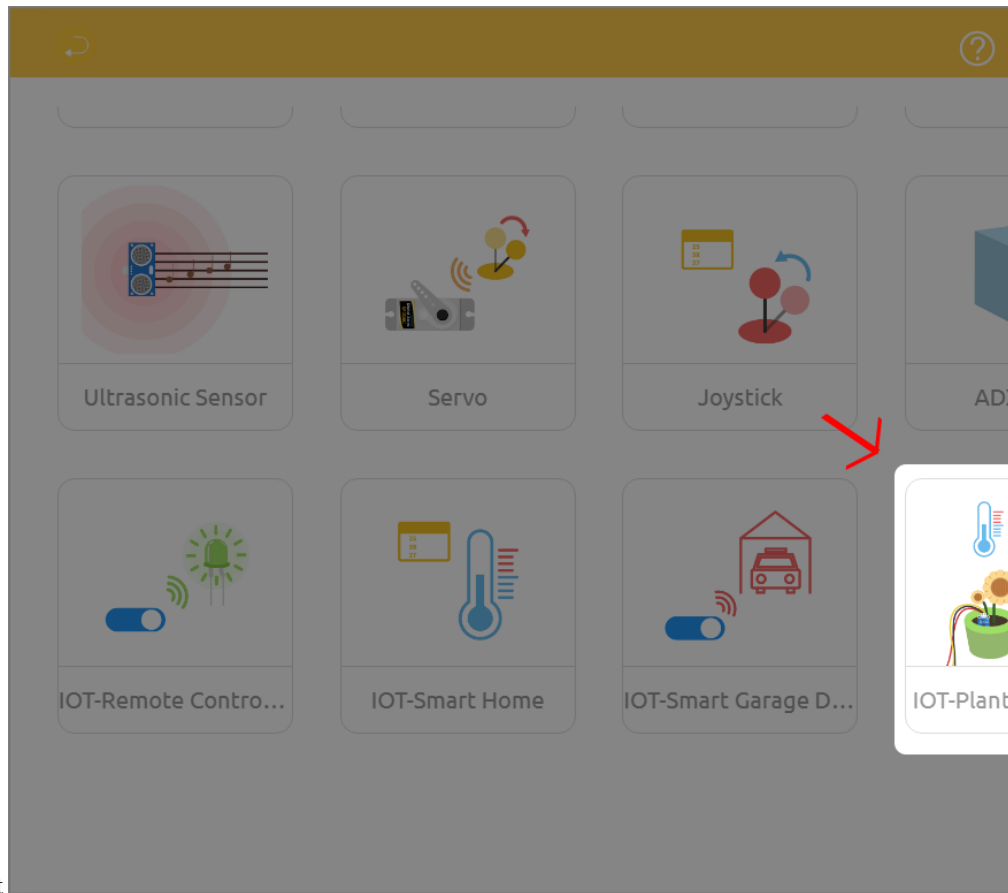
2.3.1 Open the Example



1. Select the product as IoT Sensor Kit



2. Go to Example page



3. Open the example you dare interest

The screenshot displays the Ezblock development environment for a 'Plant Monitoring' project. The interface includes a top header, a left sidebar with various block categories, a central workspace with a script, and a bottom toolbar. A red arrow points to a help icon in the top right corner.

Top Header: Ezblock | Plant Monitoring | ? | ☰

Left Sidebar:

- Basic
- Logic
- Loops
- Math
- Text
- Lists
- Music
- Colour
- Variables
- Functions
- Raspberry Pi
- SensorKit
- Time
- Sensor Kit

Main Script:

```
Start
  Connect WiFi
    Country: "CN"
    Ssid: "sunfounder"
    Password: "123456"

  Forever
    set moVal to Moisture sensor get A5 value
    set phoVal to Photoresistor get A0 value
    post Moisture Sensor-2 moVal
    post Photoresistor-1 phoVal
    if get Switch-1 value == 0
      do LED module set D0 status 0
    else
      LED module set D0 status 1
```

Bottom Toolbar: Flash, —, ↺, +

4. Open tutorials

5. Find the corresponding tutorial

Plant Monitoring

Make a monitor for your pott IoT, you can use photoresisto module to know the current environment of your plants e are not at home, and to give light.

The circuit is built as follows:

The IoT control is shown below:

Usage:

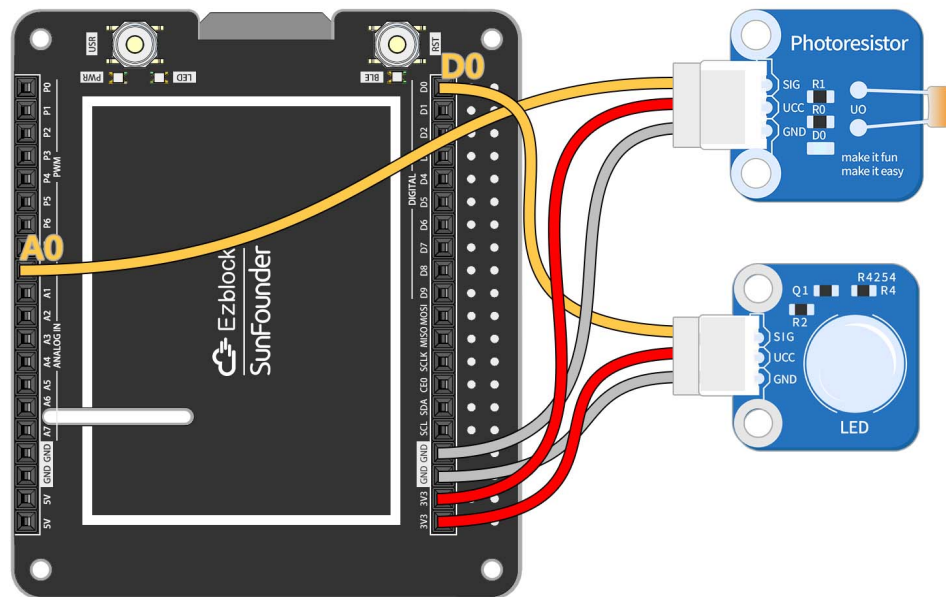
6. Follow the tutorial to build the circuit

7. Play according to USAGE.

2.3.2 Twilight Switch

In this example, you will use photoresistor to make an automatic sensor light. Before sunset, the light will remain off; after sunset, the light will automatically turn on.

The circuit is built as follows:

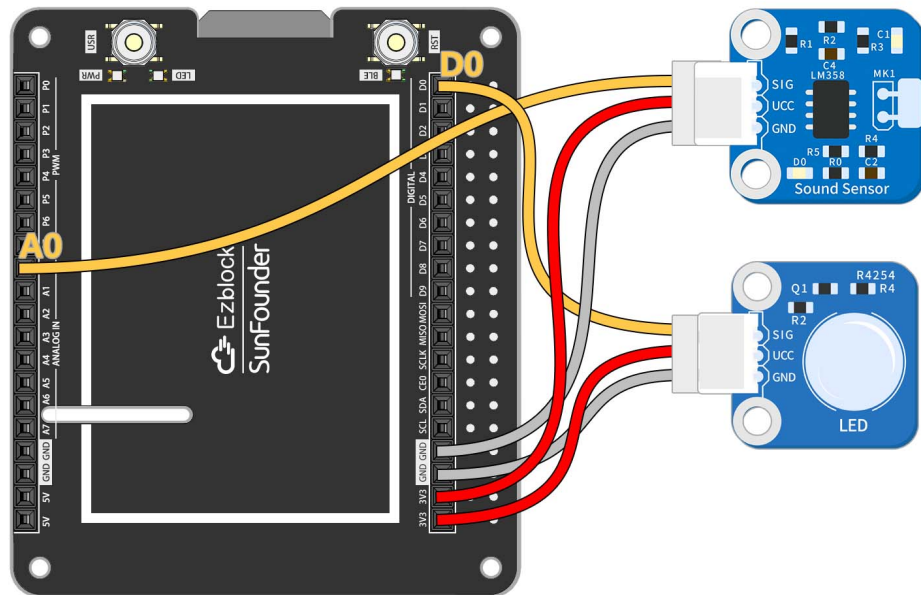


Usage:

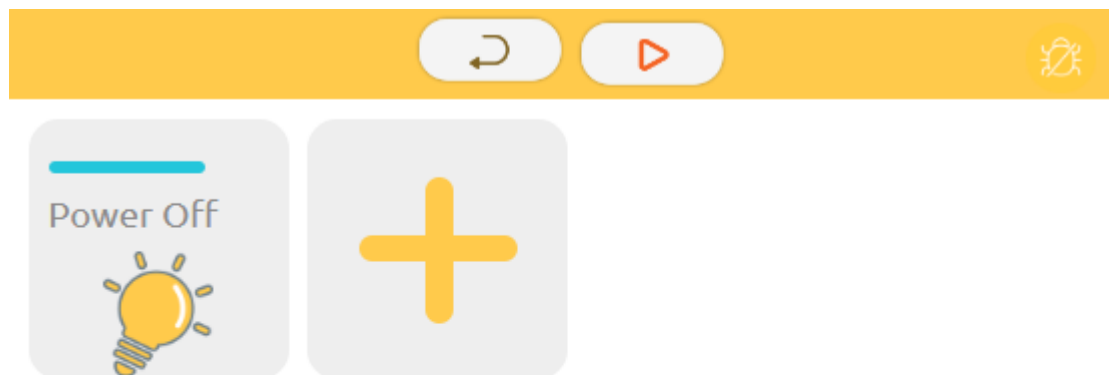
1. Flash the code
2. Simulate the “twilight” lighting environment.
3. Change the threshold to the current photoresistor value.
4. Reflash the code

2.3.3 Sound Lamp

In this example, you will create a voice-activated light using a sound sensor. The light will turn on automatically when there is enough sound. Connecting this light to the Internet allows us to turn off the light remotely at the touch of a button on the IoT interface.



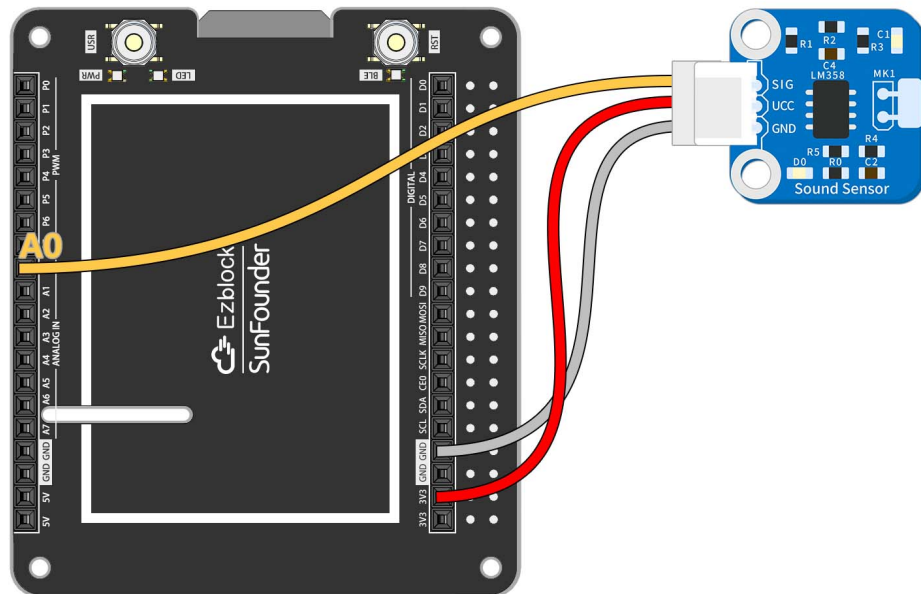
The circuit is built as follows:

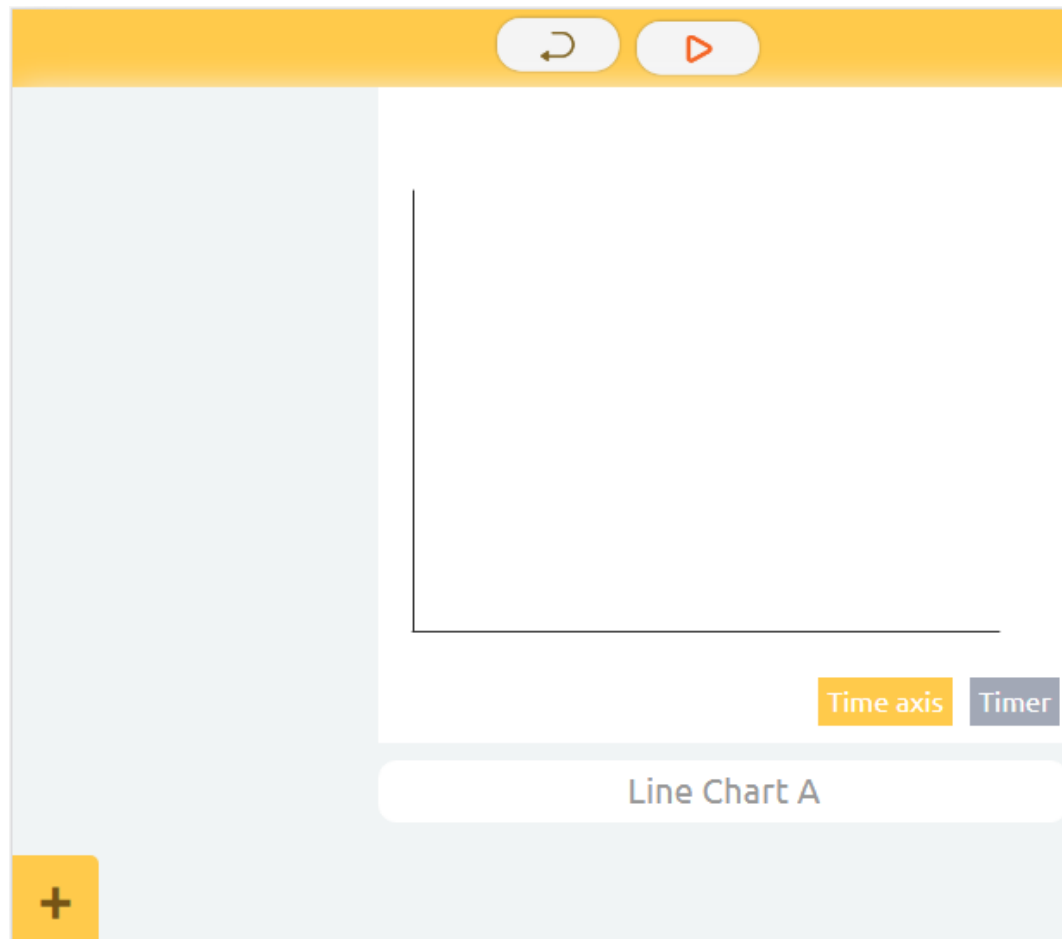


The IoT control is shown below:

Usage:

1. Modify Wifi Configuration
2. Flash the code
3. Run IoT





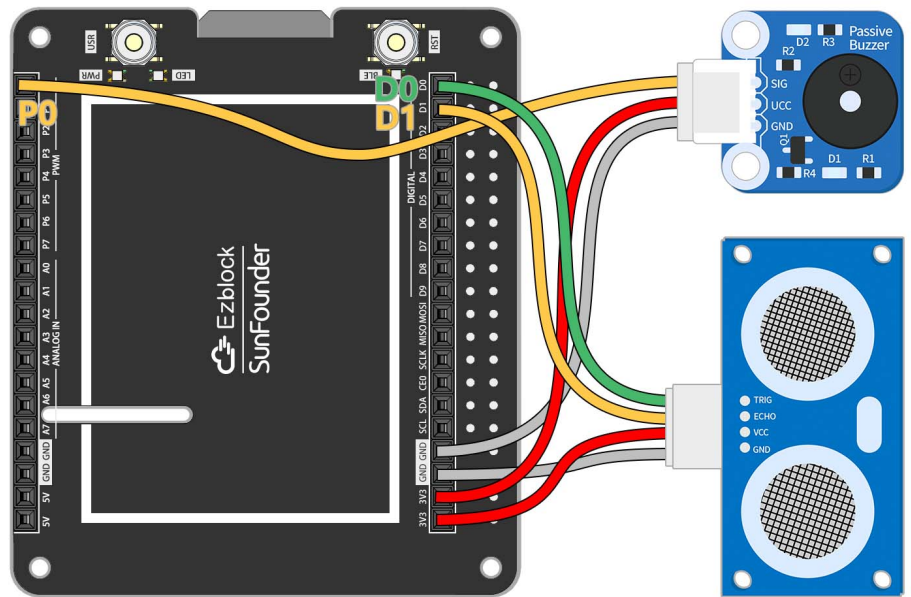
The remote controls are as follows:

Usage:

1. Flash the code
2. Run remote control
3. Perform noise testing
4. View Line Chart

2.3.5 Theremin Organ

The Theremin is the only electronic instrument in the world that does not require physical contact to play. Let's use ultrasonic sensors to make an instrument that also doesn't require physical contact!



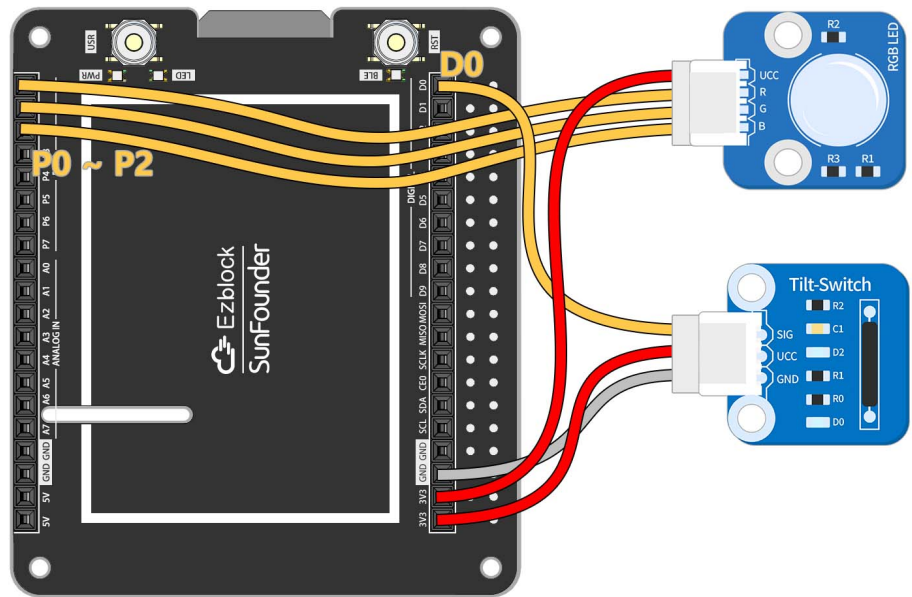
The circuit is built as follows:

Usage:

1. Flash the code
2. Place your hand directly in front of the ultrasonic sensor
3. Play by swinging the hand to different distances

2.3.6 Magic Wand

Let's make a magic wand. Tie the RGB module and tilt switch to a small stick. Wave the stick and the light will change to different colors.



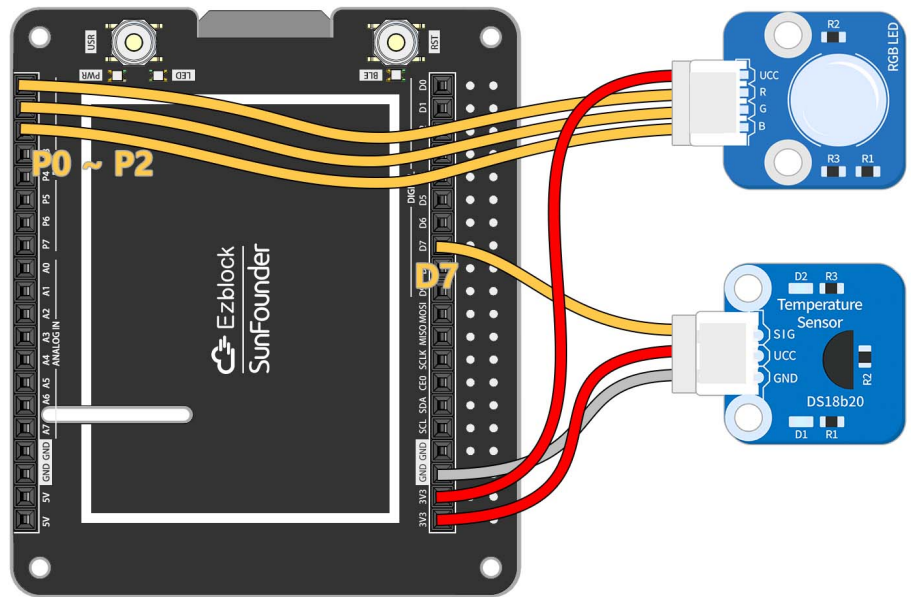
The circuit is built as follows:

Usage:

1. Flash the code
2. Wave tilt switch

2.3.7 Hot Weather Alarm

Monitor the temperature with the DS18B20! The real-time temperature of the room is seen in the IoT interface. The circuit is also connected to an RGB light that glows red when it is hot, blue when it is cold, and green when it is comfortable. You can test the temperature experiment with a cold water bottle or with warm hands.



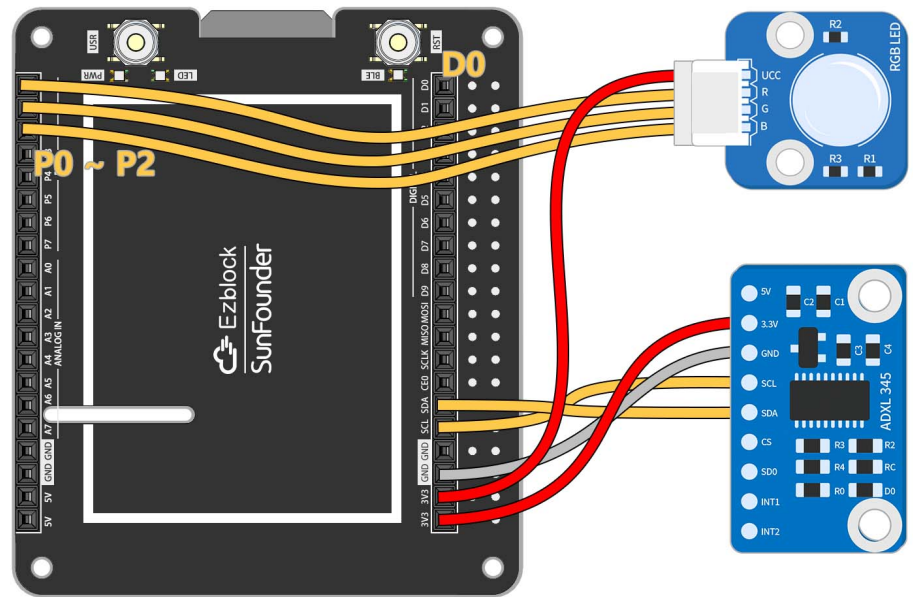
The circuit is built as follows:

Usage:

1. Flash the code
2. Run IoT

2.3.8 Swaying Rainbow

Control the color of the RGB LEDs with the acceleration in the three axis directions of ADXL345. What happens? When you wave the ADXL345 at will, the RGB LEDs will change color. When you wave the ADXL345 and the RGB LED together, a rainbow of colors appears!



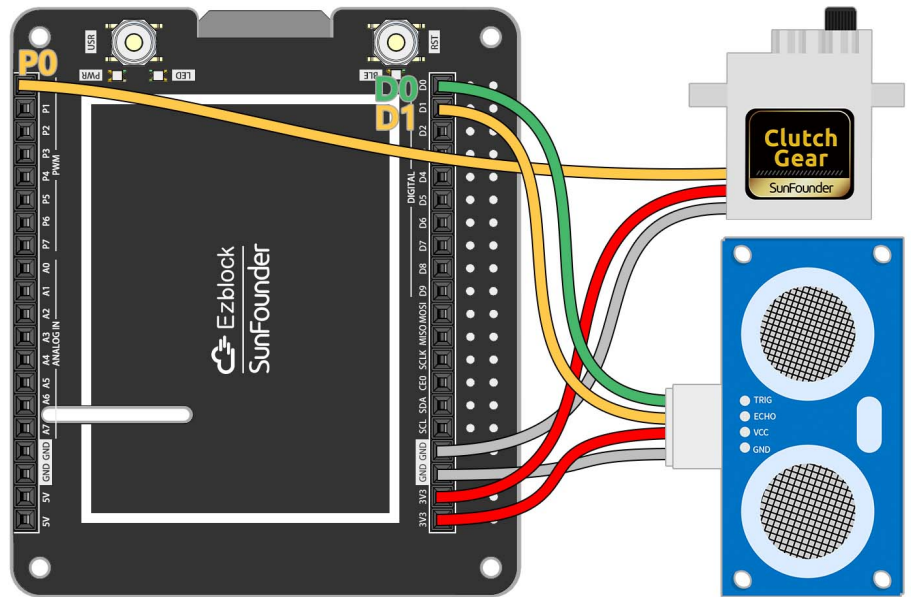
The circuit is built as follows:

Usage:

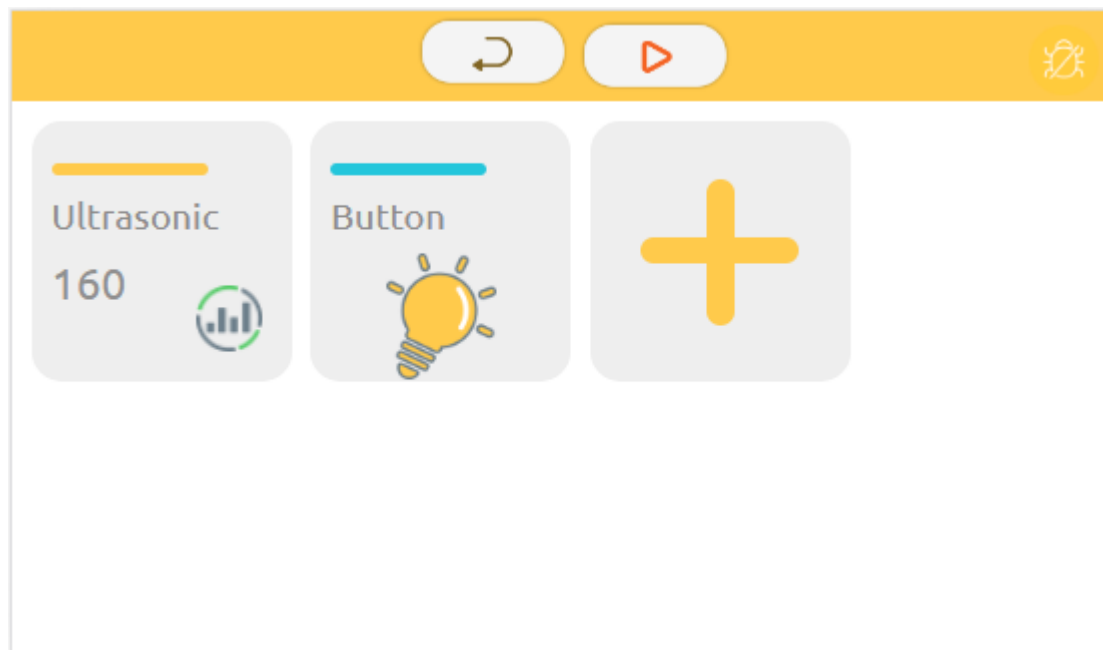
1. Flash the code
2. Play!

2.3.9 Smart Garage Door

Let's build a smart garage system where we press button in the IoT interface, the garage door (controlled by servo) will open and the ultrasonic sensor will sense the car's position. When the car is far away, the garage door will close automatically.



The circuit is built as follows:



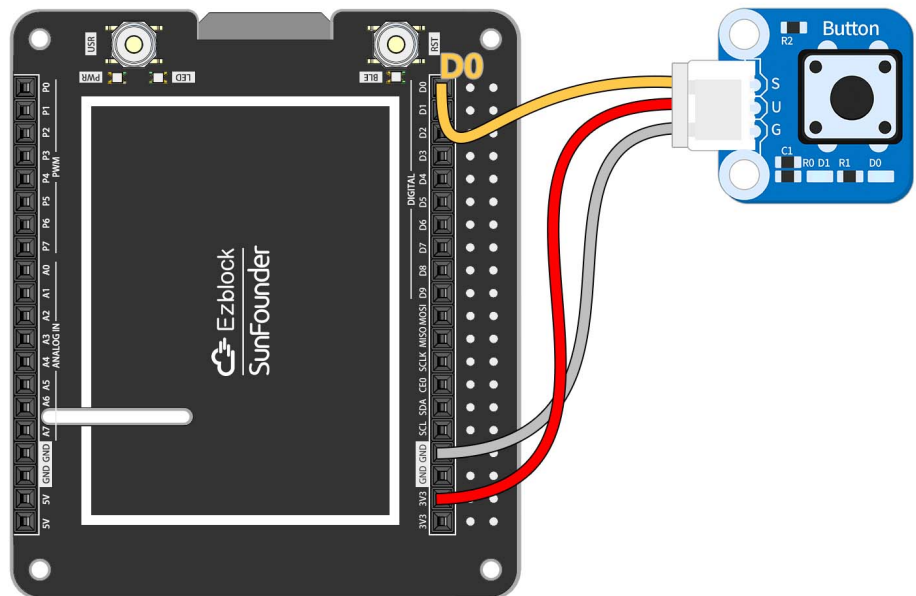
The IoT control is shown below:

Usage:

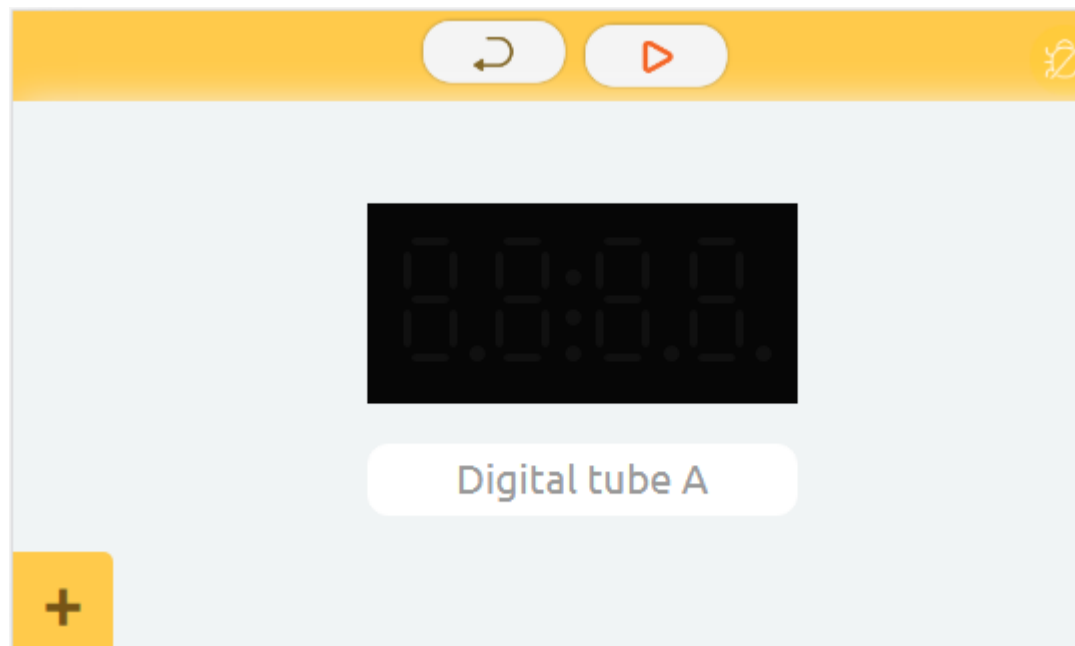
1. Modify Wifi Configuration
2. Flash the code
3. Run IoT

2.3.10 Count 100

Make a little game with button and digital display! The numbers will increase rapidly, so you have to press the button when the number reaches 100, and if the pinch is successful, You Win!



The circuit is built as follows:



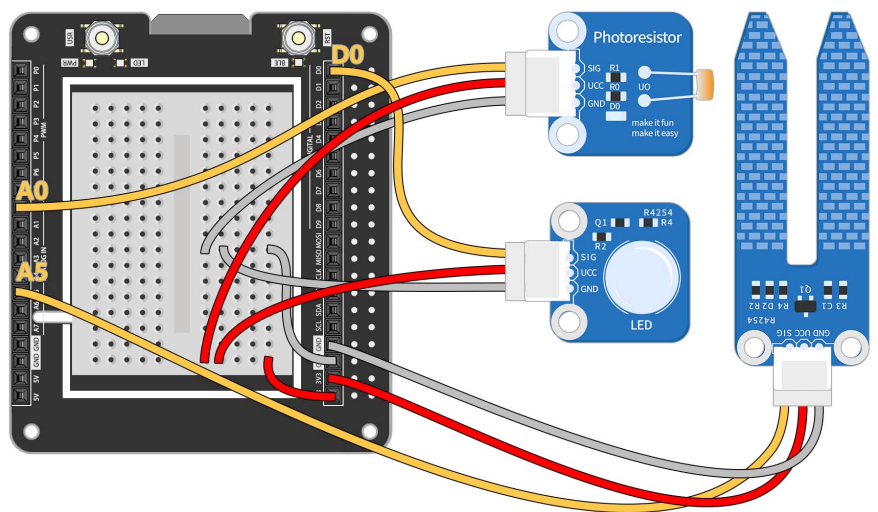
The remote controls are as follows:

Usage:

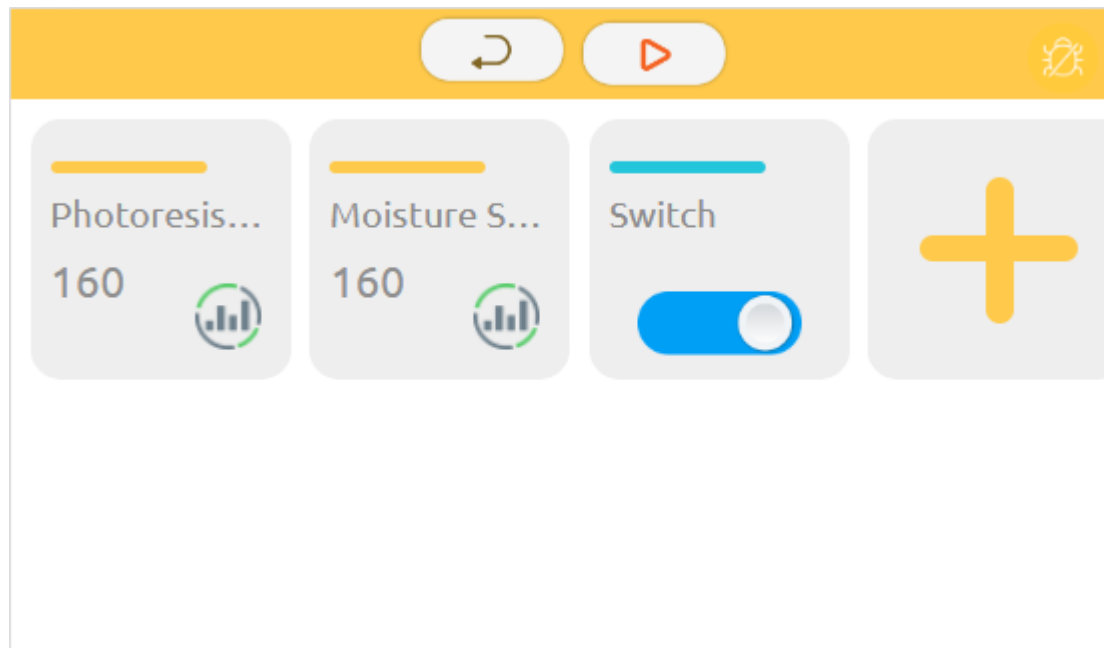
1. Flash the code
2. Run remote control
3. Play

2.3.11 Plant Monitoring

Make a monitor for your potted plant! With IoT, you can use photoresistor and moisture module to know the current growing environment of your plants even when you are not at home, and to give them additional light.



The circuit is built as follows:



The IoT control is shown below:

Usage:

1. Modify Wifi Configuration
2. Flash the code
3. Run IoT

REFERENCES

3.1 language

3.1.1 Block

This article introduces the use and annotation of blocks in block programming

Basic



- **effect** The block placed in Start will only be executed once, which is suitable for some initialization operations in the block



- **effect** The content in the block will loop indefinitely



- **effect** The corresponding content can be output to the console, which can be used for debugging
- **parameter** "abc" is what you want to output, it can be any type



- **effect** Delay a certain timeSet duration for operation
 - **parameter** "100" Can only be a number type, in milliseconds
-

Logic



- **effect** Used to capture changes in variables to respond When the condition is met, that is, when the parameter value is true, the content of the block will be executed
-



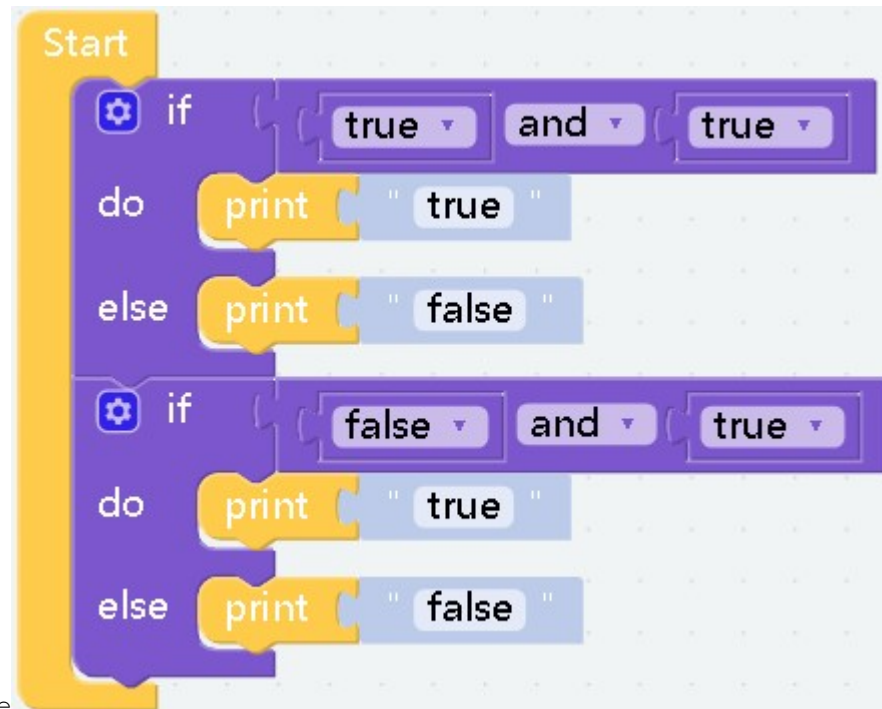
- **effect** Compare the left and right parameters, return a Boolean value, return true if the condition is met, otherwise return false
 - **parameter** The value type on the left and right sides must be the same, for example, both are numbers or both are characters
-



- **effect** It will return a Boolean value, and the parameters on both sides must also be Boolean values. **If both sides are true, return true. If either party is false or both parties are false, return false.**



- **effect** Returns a boolean value, the parameters on both sides must also be boolean values, **as long as either of them is true, it will return true, and only when both are false will it**



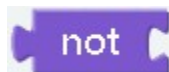
return false
results:

computational

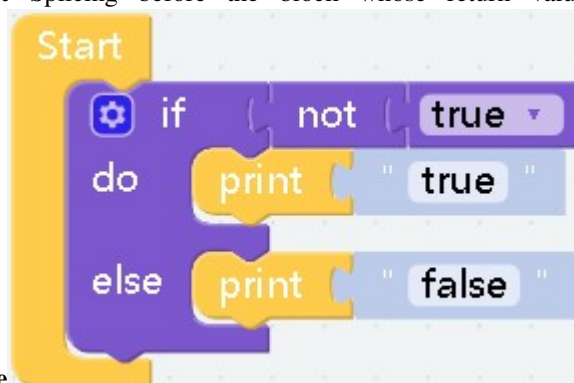
```

true
false

```



- **effect** Splicing before the block whose return value is Boolean will get the **opposite Boolean**



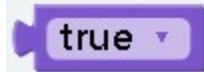
value

computational results:

```

false

```



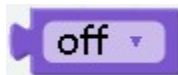
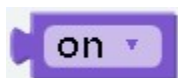
- **effect** That is, the boolean value `true`.



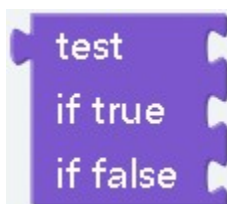
- **effect** That is, the boolean value `false`.
-



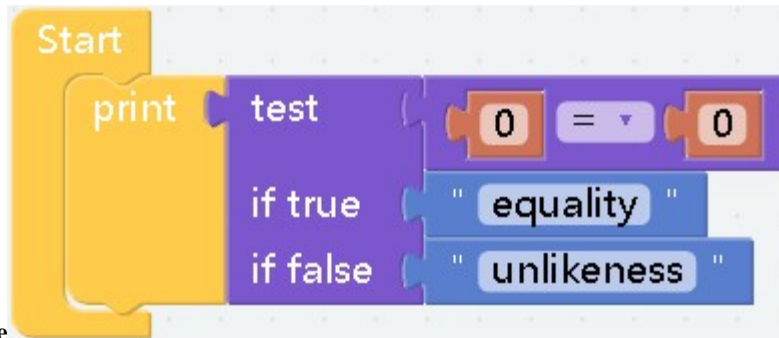
- **effect** Generally used to compare or assign values to variables
-



- **effect** In fact, it has the same function as the `true` block and the `false` block. They are all boolean values, but this block is more suitable for switch state assignment.
-



- **effect** This is a ternary expression, if the return value of `test` is `true` then the block after `if true` will be executed, and vice versa



- **example**

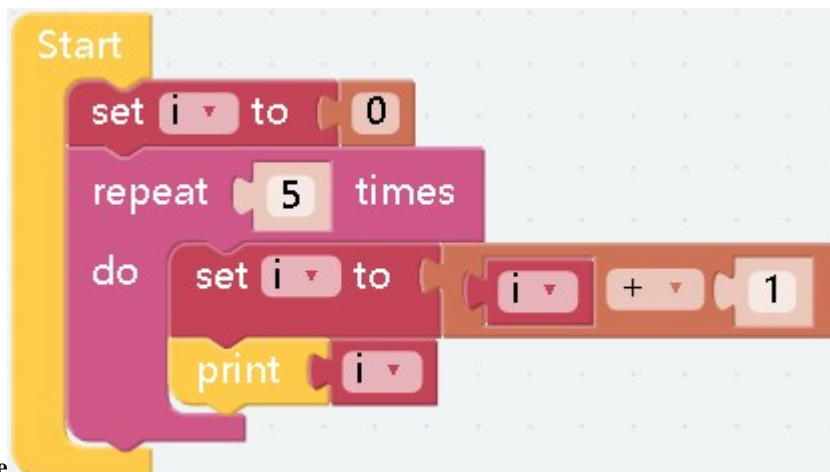
computational results:

equality

Loops



- **effect** The code in the block will be executed a certain number of times
- **parameter** The parameter must be an integer, used to specify the number of cycles



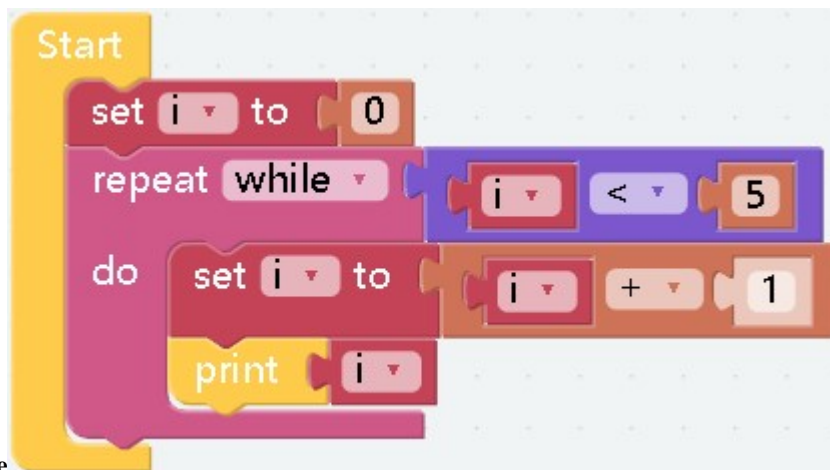
- **example**

computational results:

1
2
3
4
5



- **effect** When the conditions are met, the content in the block will be executed repeatedly
- **parameter while** The type of the parameter is Boolean. When the value of the parameter is **true**, it will enter the loop. **until** The type of the parameter is Boolean. When the value of the parameter is **false**, it will enter the loop.



- **example**

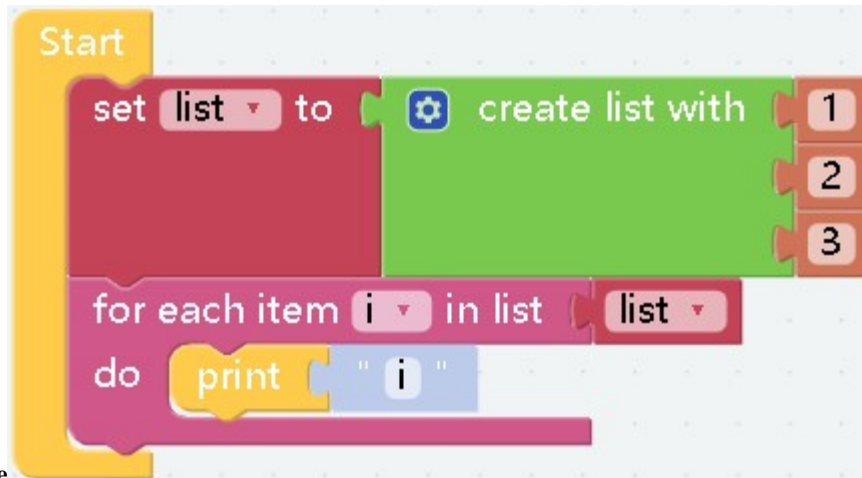
computational results:

```
1
2
3
4
5
```

If it is `until` then nothing will be output



- **effect** Read the specified list from the subscript [0] in order
- **parameter** The parameter can only be a value of type list



- **example** results:

computational

```
1
2
3
```



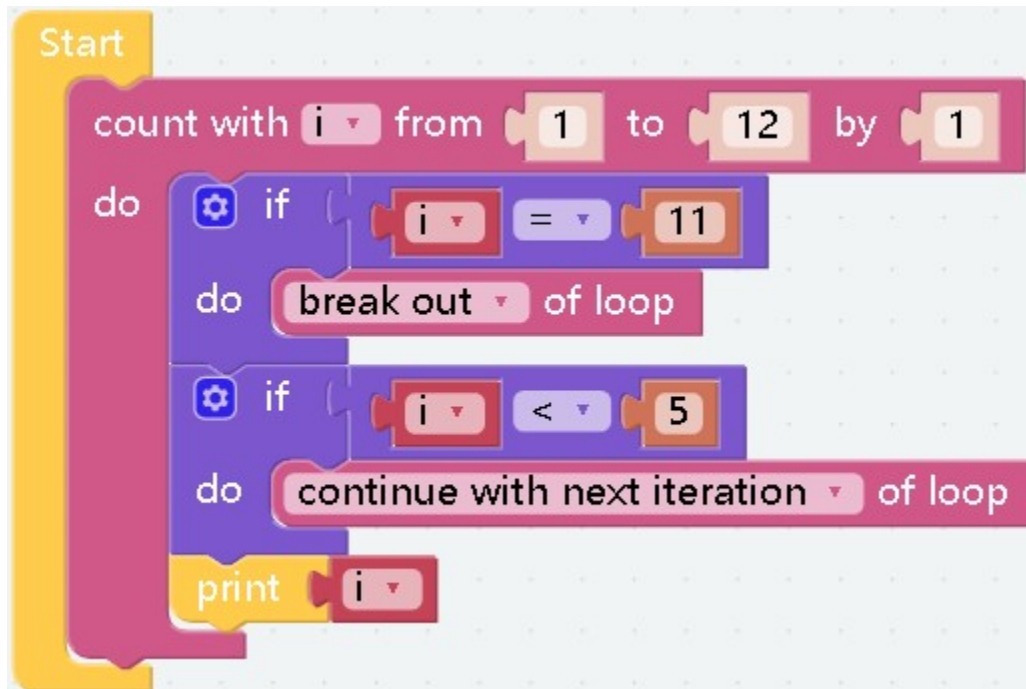
- **effect** Set the start number and increment number on the basis of repeat 10 times
- **parameter** The parameters can only be integers. During the loop, the current subscript value will be assigned to j, 5 means starting number, 10 means ending number, 2 means increasing number, and in non-special cases, second Parameter is not greater than the third parameter



- **example** results:

computational

```
5
6
7
8
9
10
```



- **effect** Break and continue can only be used in a loop block. The execution of break in a loop block will terminate the entire loop. If continue is executed, it will jump out of this loop and start the next loop.

computational results:

```
5
6
7
8
9
10
```

Math



- **effect** Usually a digital block for assignment, the parameter can be negative or decimals.



- **effect** Hexadecimal
-



- **effect** The two sets of data are mapped in equal proportions, which are commonly used to control analog value modules.
-



- **effect** Generally used for assignment after calculation
-



- **effect** Perform selected operations on parameters
-



- **effect** Graph formula related operations
-



- **effect** Needless to say
-



- **effect** Determine the type of the parameter and return a Boolean value
-



- **effect** Perform operations such as rounding parameters
-



- **effect** Get the relevant attributes of the list
-



- **effect** Take the remainder
-



- **effect** Limit the range of numbers. If it is less than the range, it is assigned to parameter 2, and if it is greater than the range, it is assigned to parameter 3.
-



- **effect** Get a random integer in the specified interval



- **effect** Get random scores

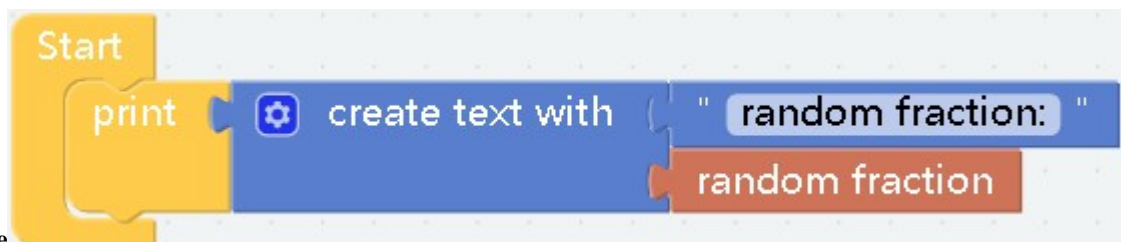
Basic



- **effect** String value, generally used for assignment or judgment



- **effect** Used to concatenate strings
- **parameter** The number of parameters can be any number, the parameters can also be any type, but they will all be converted to string



- **example** results:

0.14

computational

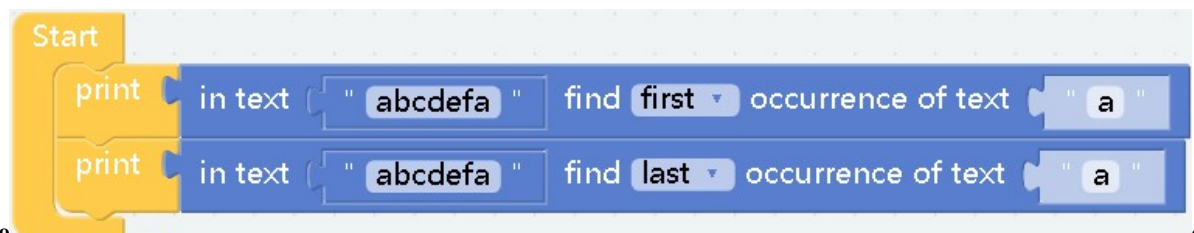


- **effect** Concatenated string
- **parameter** the parameters can be any type, but they will all be converted to string



- **effect** Returns the length of the string, the return value of int type

- **effect** Check whether there is a corresponding character in the character, and return the position, if not, return 0 You can set to return the position of the first character found, or return the position of the last character
- **parameter** Both parameters are of string type, the first parameter is the main string, and the second parameter represents the character to be found in the first parameter



- **example** results:

1
7

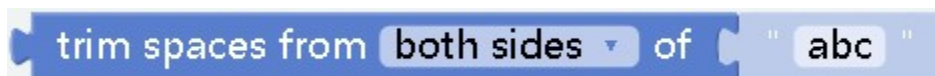


- **effect** Get the string at the specified position in the text

- **effect** Get the characters between the specified subscripts in the text
-



- **effect** don't say so much
-

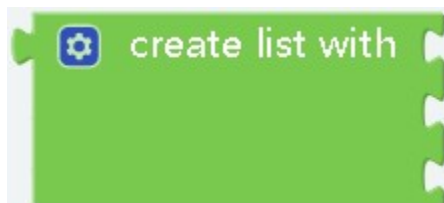


- **effect** Needless to say
-

Lists



- **effect** More often, an empty list is assigned to a variable, or it can be output directly.
-



- **effect** Initialize a list, or reassign the list
-



- **effect** Assign the same value to the list a certain number of times
-



- **effect** Read or delete the value of the corresponding position in the list



- **effect** Modify the value of the corresponding position in the list
- **parameter** When the position parameter is #, the first parameter indicates that the position can only be of type int, and the second parameter indicates that the value to be modified can be of any type

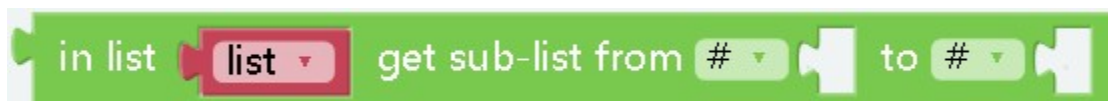


- **effect** Find the corresponding value in the list and return the subscript



- **example**
computational results:

2



- **effect** Get a part of the list by subscript



- **effect** Convert the string to a list, add subscripts through the specified symbols
- **parameter** Only string

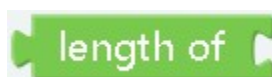


- **effect** Convert the list to a string, and separate them by the specified symbol
- **parameter** Only list



- **example**
computational results:

```
['a', 'b', 'c']  
a,*,b,*,c
```



- **effect** Get list length



- **effect** Determine whether the list is empty, return a Boolean value



- **effect** Sort the list in the specified way and return the sorted list
-

Music

These blocks can only be used in conjunction with some specific blocks, such as `picar-x` car



- **effect** Set tone
-



- **effect** Set music speed
-



- **effect** Set the beat
-

Colour

Prologue

Colour can only be used with some special modules, such as `rgb` modules



- **effect** Variable for outputting a colorm,computer-based color
-



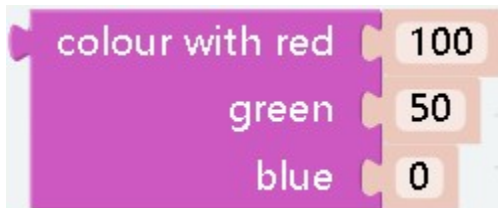
- **effect** Variable for outputting a color, Based on the colors that the rgb module can output



- **effect** The color is composed of three colors of r, g, and b according to a certain ratio. This block can read the ratio of r, g, and b of the color value



- **effect** Get random colors



- **effect** Create a color according to the custom r, g, b ratio
- **parameter** The rgb value of ezblock is different from the normal rgb value. Normally, it is 0~255, but the chromaticity in ezblock is 0~100 (this is also the range of the parameter)



- **effect** You can mix two colors in a certain ratio
- **parameter** The parameter can only be a decimal (fraction) less than 1, which will be allocated to colour2 and the rest will be allocated to colour1. For example, if the parameter is 0.4, then the colors will be mixed in the ratio of colour10.6, colour20.4

Variables



effect Add one to the variable. Doing so will make him eat an int type. This block can be used for technology in the logic block of the loop.

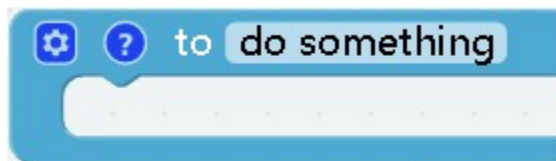


effect Assignment, the value can be any type

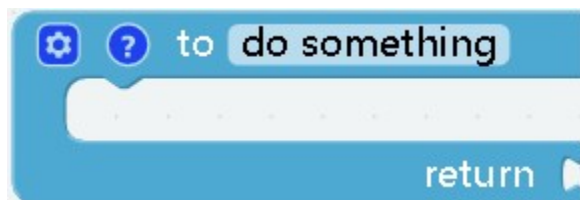


effect Variable value, can be judged again, or used in the case of assignment lamp

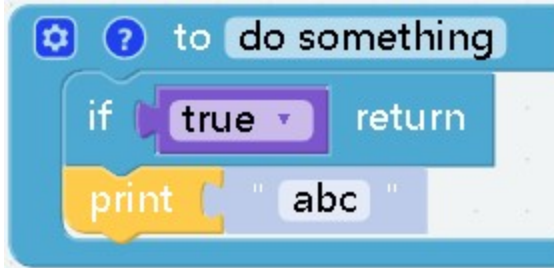
Functions



- **effect** Create a function with no return value
-



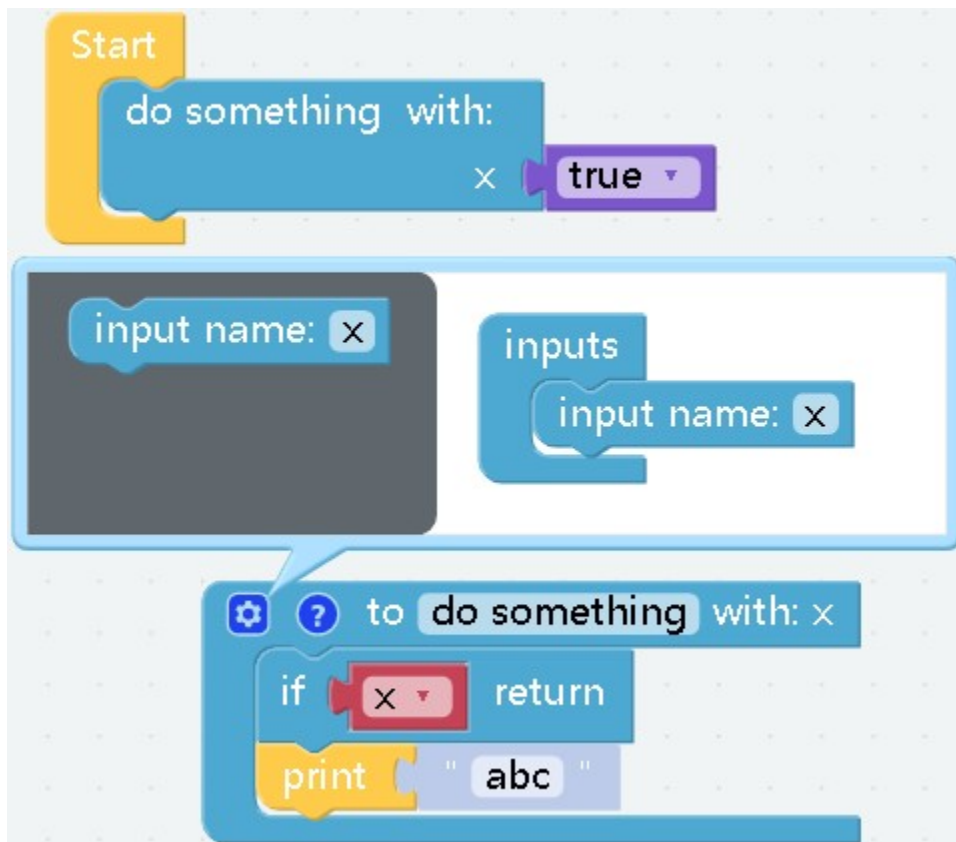
- **effect** Create a function with a return value
-



- **effect** When certain conditions are met, the function will be terminated and subsequent statements will not be executed



- **effect** Call a function you created and execute its contents



- **effect** Optional when creating a function. Click the gear in the upper left corner of the function block to configure the parameters. The transmitted parameters can be used directly inside the function **computational results**:

RaspberryPi

Pin



- **effect:** pin object
-



- **effect:** pin objects of buttons and led lights on the expansion board
-



- **effect:** Set the mode of the pin to output or input, Set the mode of the pin to output or input, for example, the led light is output, and the button is input
-



- **effect:** Get the status of the specified pin will return 0 or 1
-



- **effect:** Setting the pin to off means setting the value of the pin to 0
-

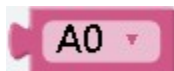


- **effect:** Setting the pin to on means setting the value of the pin to 1
-



- **effect:** Set the state of the pin, the parameter can only be 0 or 1
-

ADC



- **effect** Pin object to input analog value, Such as joystick, potentiometer, sound sensor and other modules



- **effect** Get the value of the corresponding analog pin

PWM



- **effect** pwm pin object
-



- **effect** set the servo angle
 - **parameter** It can only be a numerical value, indicating the angle you want to set, between -90 and 90
-



- **effect** Set the pwm value directly to adjust the “strength” of the pin
 - **parameter** 0~4095
-



- **effect** Set the pwm value by percentage
 - **parameter** 0.00~1.00
-



- **effect** Set the evaluation rate of the pin to 50 (cycle 20 milliseconds)
-

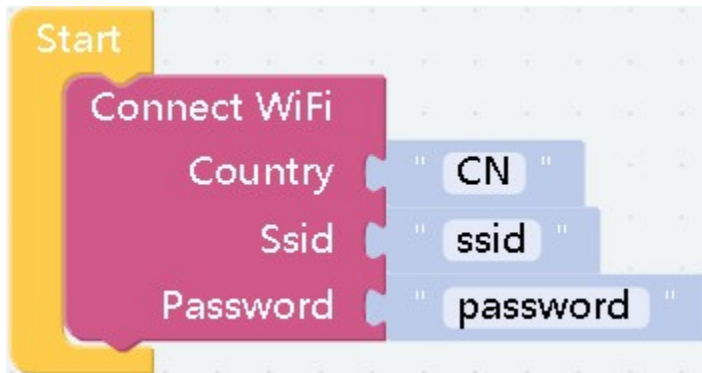


- **effect** Set the clock rate of the pin pwm unit
-



- **effect** Set period
-

System



- **effect** Connect to wifi



- **effect** Get the status of the Raspberry Pi



- **effect** Whether there is this i2c address in the connected device, the return value is Boolean



- **effect** List all i2c addresses



- **effect** Send data to the corresponding address



- **effect** Read the data corresponding to the address
-



- **effect** Send data to the corresponding address
-



- **effect** Read the data corresponding to the address
-

TextToSpeech



- **effect** Select the language you want to convert
-



- **effect** The input language will be played
-

Camera



- **effect** The width or height of the detected color. If there are multiple detected colors in the camera, they will be returned in the order of detection. All widths
-



- **effect** Returns the height or width of the face
-



- **effect** Return the result of the QR code
-



- **effect** Color object for color recognition
-



- **effect** Set the color to be detected
-



- **effect** Enable or disable color recognition
-



- **effect** Turn on or turn off face recognition
-



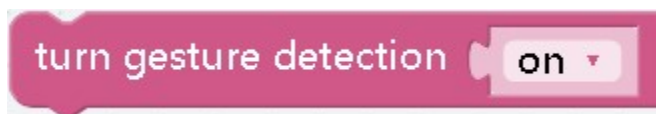
- **effect** Enable or disable color recognition
-



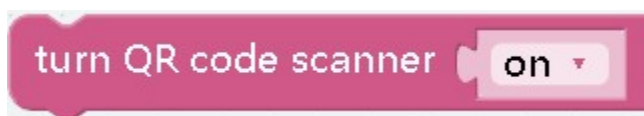
- **effect** Turn on or turn off face recognition
-



- **effect** Enable or disable gesture calibration
-



- **effect** Enable or disable gesture detection
-



- **effect** Enable or disable QR code detection
-

A pink rectangular block with a notch on the left and a tab on the right. The text "turn traffic sign detection" is in white, and "on" is in a smaller white box with a dropdown arrow on the right.


turn traffic sign detection on ▾

- **effect** Enable or disable traffic sign detection
-

A pink rectangular block with a notch on the left and a tab on the right. The text "turn video monitor" is in white, and "on" is in a smaller white box with a dropdown arrow on the right.

turn video monitor on ▾

- **effect** Start or close the camera screen
-

A pink rectangular block with a notch on the left and a tab on the right. The text "width" is in white, followed by a dropdown arrow.

width ▾

- **effect** Object to be detected
-

A pink rectangular block with a notch on the left and a tab on the right. The text "width" is in white, followed by a dropdown arrow, and then "of detected color" in white.

width ▾ of detected color

- **effect** Color width or height
-

A pink rectangular block with a notch on the left and a tab on the right. The text "width" is in white, followed by a dropdown arrow, and then "of detected face" in white.

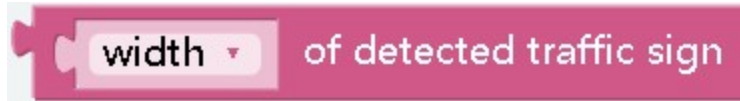
width ▾ of detected face

- **effect** The width or height of the face
-

A pink rectangular block with a notch on the left and a tab on the right. The text "width" is in white, followed by a dropdown arrow, and then "of detected gesture" in white.

width ▾ of detected gesture

- **effect** The width or height of the gesture
-



- **effect** The width or height of the traffic sign
-



- **effect** Start camera
-

3.1.2 Python

ezblock

Under ezblock, the classes and methods used import everything from `ezblock import *`

Methods

- `delay` - Delay for the given number of milliseconds.

`delay(ms)`

- `print` - replace the original print function to print via bluetooth.

`print(msg, end="/n", tag=' [DEBUG] ')`

- `mapping` - map a value from a range to another

`mapping(x, inmin, inmax, outmin, outmax)`

Classes

class `Pin` - control I/O pins

Usage:

```
from ezblock import Pin

pin = Pin("D0")           # create an Pin object from a pin
val = pin.value()         # read an analog value
```

Constructors

`class ezblock.Pin(value)` A pin is the basic object to control I/O pins. It has methods to set the mode of the pin (input, output, etc) and methods to get and set the digital logic level.

Methods

- value - Read the value on the analog pin and return it. The returned value will be between 0 and 4095.

```
Pin.value()
```

Available pins

- "D0 "
- "D1 "
- "D2 "
- "D3 "
- "D4 "
- "D5 "
- "D6 "
- "D7 "
- "D8 "
- "D9 "
- "SW "
- "LED "

class ADC - analog to digital converter

Usage:

```
from ezblock import ADC

adc = ADC("A0")           # create an analog object from a pin
val = adc.read()          # read an analog value
```

Constructors

`class ezblock.ADC(pin)` Create an ADC object associated with the given pin. This allows you to then read analog values on that pin.

Methods

- read - Read the value on the analog pin and return it. The returned value will be between 0 and 4095.

```
ADC.read()
```

111

class PWM - pulse width modulation

Usage:

```
from ezblock import PWM

pwm = PWM('P0')           # create an pwm object from a pin
pwm.freq(50)               # set freq 50Hz
pwm.prescaler(2)           # set prescaler
pwm.period(100)            # set period

pwm.pulse_width(10)        # set pulse_width
pwm.pulse_width_percent(50) # set pulse_width_percent
```

Constructors

`class ezblock.PWM(channel)` Create an PWM object associated with the given pin. This allows you set up the pwm function on that pin.

Methods

- `freq` - set the pwm channel freq.

```
PWM.freq(50)
```

- `prescaler` - set the pwm channel prescaler.

```
PWM.prescaler(50)
```

- `period` - set the pwm channel period.

```
PWM.period(100)
```

- `pulse_width` - set the pwm channel pulse_width.

```
PWM.pulse_width(10)
```

- `pulse_width_percent` - set the pwm channel pulse_width_percent.

```
PWM.pulse_width_percent(50)
```

class Servo - 3-wire pwm servo driver

Usage:

```
from ezblock import Servo, PWM

pin = PWM("P0")
ser = Servo(pin)           # create an Servo object from a pin
val = ser.angle(60)        # set the servo angle
```

Constructors

`class ezblock.Servo(pin)` Create an Servo object associated with the given pin. This allows you to set the angle values.

Methods

- `angle` - set the angle values between -90 and 90.

```
Servo.angle(90)
```

class UART - serial communication bus

Usage:

```
from ezblock import UART

# On Raspberry Pi and init later
uart = UART("/dev/serial0")      # create an UART object
uart.init(9600)                  # uart init
uart.read(5)                     # read up 5 bytes

# On Ezblock One and init
uart = UART(1, tx=25, rx=26, baudrate=115200)  # create an UART object
buf = [1,3,5]
buf = bytearray(buf)
uart.write(buf)                  # send a buf
```

Constructors

`class ezblock.UART(device, tx=None, rx=None, baudrate=115200)` Create an UART object.
device: On Ezblock Pi, it's the serial path, like: /dev/xxx(). On Ezblock One, it's the uart id: 0, 1, 2 tx: pin of tx. rx: pin of rx

Methods

- `init` - init the uart.

```
UART.init(baudrate)
```

- `read` - read data.

```
UART.read(num)
```

- `write` - send a buf of bytes.

```
UART.write(buf)
```

class I2C - IIC bus

Usage:

```
from ezblock import I2C

i2c = I2C(1) # create on bus 1
i2c = I2C(1, I2C.MASTER) # create and init as a master

i2c.send('abc') # send 3 bytes
i2c.send(0x42) # send a single byte, given by the number
data = i2c.recv(3) # receive 3 bytes

i2c.is_ready(0x42) # check if slave 0x42 is ready
i2c.scan() # scan for slaves on the bus, returning
# a list of valid addresses
i2c.mem_read(3, 0x42, 2) # read 3 bytes from memory of slave 0x42,
# starting at address 2 in the slave
i2c.mem_write('abc', 0x42, 2, timeout=1000) # write 'abc' (3 bytes) to memory of
↪ slave 0x42 # starting at address 2 in the slave,
↪ timeout after 1 second
```

Constructors

`class ezblock.I2C(num)` Create an I2C object associated with the given num. This allows you to use i2c on that device.

Methods

- `is_ready` - check if slave 0x42 is ready

```
I2C.is_ready(addr)
```

- `scan` - scan for slaves on the bus, returning

```
I2C.scan()
```

- `send` - send several bytes to slave with address

```
I2C.send(send, addr, timeout)
```

- `recv` - # receive one or several bytes

```
data = i2c.recv(recv, addr, timeout) # receive 3 bytes
```

- `mem_write` - Write to the memory of an I2C device

```
I2C.mem_write(data, addr, memaddr, timeout)
```

- `mem_read` - Read from the memory of an I2C device

```
I2C.mem_read(data, addr, memaddr, timeout)
```

class Remote - remote with ble

Usage:

```
from ezblock import Remote

remote = Remote()           # create an Remote object from
val = remote.read()         # read an analog value

slider_val = remote.get_slider_value() # get slider value
```

Constructors

`class ezblock.Remote()` Create an Remote object associated with the device.

Methods

- read - Read the name and value of device.

```
Remote.read()
```

- get_value - get the value of device.

```
Remote.get_value()
```

- get_joystick_value - get the joystick_value of device.

```
Remote.get_joystick_value()
```

- get_slider_value - get the slider_value of device.

```
Remote.get_slider_value()
```

- get_dpad_value - get the dpad_value of device.

```
Remote.get_dpad_value()
```

- get_button_value - get the button_value of device.

```
Remote.get_button_value()
```

- get_switch_value - get the switch_value of device.

```
Remote.get_switch_value()
```

class IOT - internet of things

Usage:

```
from ezblock import IOT

iot = IOT()
```

Constructors

`class ezblock.IOT(iot_token, device)` Create an ADC object associated with the given pin. This allows you to then read analog values on that pin.

Methods

- `post` - 1

```
IOT.post(sensorname, value)
```

- `get` - 1

```
IOT.get(sensorname)
```

class Music - notes and beats

Usage:

```
from ezblock import Music, Buzzer

m = Music()                # create an music object
buzzer = Buzzer("P0")
m.tempo(120)               # set current tempo to 120 beat per minute

# play middle C, D, E, F ,G, A, B every 1 beat.
buzzer.play(m.note("Middle C"), m.beat(1))
buzzer.play(m.note("Middle D"), m.beat(1))
buzzer.play(m.note("Middle E"), m.beat(1))
buzzer.play(m.note("Middle F"), m.beat(1))
buzzer.play(m.note("Middle G"), m.beat(1))
buzzer.play(m.note("Middle A"), m.beat(1))
buzzer.play(m.note("Middle B"), m.beat(1))
```

Constructors

`class ezblock.Music()` Create an Music object. This allows you to then get or control music!

Methods

- `note` - get frequency of the note. Input string must be in Constant NOTE

```
Music().note("Middle D")
Music().note("High A#")
```

- `beat` - get milisecond from beats. Input value can be float, like 0.5 as half beat, or 0.25 as quarter beat

```
Music().beat(0.5)
Music().beat(0.125)
```

- `tempo` - get/set the tempo. input value is in bmp(beat per second)


```
Music().tempo()
Music().tempo(120)
```

- `play_tone_for` - Play tone. Input is note and beat, like `Music.note("Middle D"), Music.beat(0.5)`

```
Music().play_tone_for(Music.note("Middle D"), Music.beat(0.5))
```

class Color - rgb color

Usage:

```
from ezblock import Color

c = Color()                                # create an color object
white = c.color("#ffffff")                 # hex color
white_led = c.led_color("#ffffff")         # hex color for led
color_red = c.get_from("red", "#ffffff")   # get red from a rgb color
random_color = c.random()                  # get random color
color = c.rgb(200, 20, 40)                  # get color from RGB value
blended = c.blend("#ff0000", "#00ff00", 0.5) # blend 2 color with specific ratio
```

Constructors

`class ezblock.Color()` Create an Color object. This allows you to then get or control colors!

Methods

- `color` - get color from a hex string. this function only test the value if is color format, then returns the input value.

```
Color().color("#88ff44")
```

- `led_color` - get color from a hex string. this function only test the value if is color format, then returns the input value. same as `color()`

```
Color().led_color("#88ff44")
```

- `get_from` - get Red/Green/Blue value from a color.

```
Color().get_from("red", "#88ff44")
```

- `random` - get random color.

```
Color().random()
```

- `rgb` - get color from RGB value. range values 0~255.

```
Color().rgb(200,100,20)
```

- `blend` - blend two color with specific ratio. Ratio ranges 0~1, float

```
Color().blend("#ff0000", "#00ff00", 0.5)
```

class Camera - camera module

Usage:

```
from ezblock import Camera

cam = Camera()           # create an camera object
cam.start()              # start camera streaming
cam.stop()               # stop camera streaming
```

Constructors

`class ezblock.Camera(res=1, fps=12, port=9000, rotation=0)` Create an Camera object. This allows you to then control the camera!

- `res` resolution. 0: 320x240, 1: 640x480, 2: 1024x576, 3: 1280x800, default to 1
- `fps` frame per second. default to 12
- `port` Streaming port, default to 9000 for ezblock remote panel
- `rotation` rotation of the video.

Methods

- `start` - start video streaming value.

```
Camera().start()
```

- `stop` - stop video streaming

```
Camera().stop()
```

class TTS - text to speech

Usage:

```
from ezblock import *

tts = TTS()               # create an TTS object
tts.say('hello')          #write word

# tts.write('hi')         #write word
tts.lang('en-GB')         #change language

tts.supported_lang()      #return language
```

Constructors

`class ezblock.TTS(engine)` Create an TTS object. engine could be "espeak" as Espeak, "gtts" as Google TTS and polly as AWS Polly

Methods

- say - Write word on TTS.

```
TTS.say(words)
```

- lang - Change on TTS.

```
TTS.lang(language)
```

- supported_lang - Inquire all supported language.

```
TTS.supported_lang()
```

class IRQ - external interrupter

Usage:

```
from ezblock import IRQ

def callback(line):
    print("line =", line)
irq = IRQ('D1', IRQ.IRQ_RISING, callback('D1'))
```

Constructors

`class ezblock.IRQ(pin, trigger, callback)` Create an IRQ object associated with the given pin.

Methods

- disable - Disable the interrupt associated with the ExtInt object. This could be useful for debouncing.

```
IRQ.disable()
```

- enable - Enable a disabled interrupt.

```
IRQ.enable()
```

- line - Return the line number that the pin is mapped to.

```
IRQ.line()
```

- swint - Trigger the callback from software.

```
IRQ.swint()
```

Constance

- `IRQ_FALLING - 0`
- `IRQ_FALLING -`
- `IRQ_FALLING -`

class `wiFi` - Wi-Fi set up

Usage:

```
from ezblock import WiFi

wifi = WiFi()                                # create an WiFi object
wifi.write('CN','sunfounder','sunfounder')
```

Constructors

`class ezblock.WiFi(pin)` Create an WiFi object to connect internet.

Methods

- `write` - write the information of wifi then will connect the wifi.

```
WiFi.write()
```

class `Taskmgr` - task manager

Usage:

```
from ezblock import *

taskmgr = Taskmgr()                        # create an Taskmgr object
temp_cpu_val = taskmgr.cpu_temperature()   # read the temperature of   
↳CPU
temp_gpu_val = taskmgr.gpu_temperature()   # read the temperature of   
↳GPU
cpu_usage_val = taskmgr.cpu_usage()        # read the cpu_usage
disk_space_val = taskmgr.disk_space()      # read the disk_space
disk_used_val = taskmgr.disk_used()        # read the disk_used
ram_info_val = taskmgr.ram_info()          # read the ram_info
ram_used_val = taskmgr.ram_used()          # read the ram_used
read_val = taskmgr.read()                  # read all the system parameter of Pi
```

Constructors

`class ezblock.Taskmgr(pin)` Create an Taskmgr object to inquire the parameter of Pi.

Methods

- `cpu_temperature` - inquire the temperature of CPU.

```
Taskmgr.cpu_temperature()
```

- `gpu_temperature` - inquire the temperature of GPU.

```
Taskmgr.gpu_temperature()
```

- `cpu_usage` - inquire the usage of CPU.

```
Taskmgr.cpu_usage()
```

- `disk_space` - inquire the disk_space of Pi.

```
Taskmgr.disk_space()
```

- `disk_used` - inquire the disk_space of Pi.

```
Taskmgr.disk_used()
```

- `ram_info` - inquire the ram_info of Pi.

```
Taskmgr.ram_info()
```

- `ram_used` - inquire the ram_used of Pi.

```
Taskmgr.ram_used()
```

- `read` - inquire all the system parameter of Pi.

```
Taskmgr.read()
```

class SendMail - email library

Usage:

```
from ezblock import *

sendmail = SendMail(mail_host, sender, mail_pass)           # create an
↳ SendMail object
sendmail.send(receivers, msg, subject)                       # send a e-Mail
```

Constructors

`class ezblock.SendMail(pin)` Create an SendMail object associated with the given pin. This allows you to then read analog values on that pin.

Methods

- `send` - You can send mail by this function.

```
SendMail.send(receivers, msg, subject)
```

class Ultrasonic - ultrasonic ranging sensor

Usage:

```
from ezblock import Ultrasonic, Pin

trig = Pin("D0")
echo = Pin("D1")

ultrasonic = Ultrasonic(trig, echo)           # create an Ultrasonic object from ↵
↵pin
val = ultrasonic.read()                       # read an analog value
```

Constructors

`class ezblock.Ultrasonic(trig, echo)` Create an Ultrasonic object associated with the given pin. This allows you to then read distance values.

Methods

- `read` - Read the value on the analog pin and return it. The returned value will be between 0 and 4095.

```
Ultrasonic.read(trig, echo)
```

class DS18X20 - ds18x20 series temperature sensor

Usage:

```
from ezblock import Pin, DS18X20

pin = Pin("D0")           # create pin object
ds = DS18X20(pin)         # create an DS18X20 object with PWM object

ds.read(ds.C)             # read temperature in celsius(1)
ds.read(0)                # read temperature in Fahrenheit(0)
```

Raspberry Pi only have one one-wire pin “D7”. While Leaf works with almost every digital pins.

Constructors

`class ezblock.DS18X20(pin)` Create an DS18X20 object associated with the given pin object. This allows you to then read temperature from DS18X20.

Methods

- `DS18X20.read` - read temperature with the giving unit. it returns a `float` if only one `ds18x20` is connected to the pin, or it will return a list of all sensor values.

```
DS18X20.read(unit)
```

- `DS18X20.scan` - Scan the `ds18x20(s)` connected to the pin, returns a list of roms address

```
DS18X20.scan()
```

- `DS18X20.read_temp` - Read temperature(s) with the givving rom(s)

```
DS18X20.read_temp(rom)
```

Constants

- `DS18X20.C` - Celsius
- `DS18X20.F` - Fahrenheit

class ADXL345 - accelemeter

Usage:

```
from ezblock import ADXL345

accel = ADXL345()                # create an ADXL345 object
x_val = accel.read(accel.X)      # read an X(0) value
y_val = accel.read(1)            # read an Y(1) value
z_val = accel.read(2)            # read an Z(2) value
```

Constructors

`class ezblock.ADXL345(address=0x53)` Create an `ADXL345` object. This allows you to then read `adxl345` accelerator values.

Methods

- `read` - Read the value with the axis and return it. Value unit is gravity acceleration(about 9.8m/s²).

```
ADXL345.read(axis)
```

Constants

- `X` - x axis
- `Y` - y axis
- `Z` - z axis

class RGB_LED - rgb LED

Usage:

```
from ezblock import PWM, RGB_LED

r = PWM("P0")
g = PWM("P1")
b = PWM("P2")

rgb = RGB_LED(r, g, b)           # create an RGB_LED object from a pin
val = rgb.write('#FFFFFF')      # write value of value
```

Constructors

`class ezblock.RGB_LED(Rpin, Gpin, Bpin)` Create an RGB_LED object associated with the given pin. This allows you set the color of an RGB LED module. Input Rpin, Gpin, Bpin must be PWM object from ezblock. PWM.

Methods

- `write` - Read the value on the analog pin and return it. The returned value will be between 0 and 4095.

```
RGB_LED.write(color)
```

class Buzzer - passive buzzer

Usage:

```
from ezblock import PWM, Buzzer, Music

pwm = PWM("A0")                 # create pwm object
buzzer = Buzzer(pwm)            # create an Buzzer object with PWM object
music = Music()                 # create music object

buzzer.play(music.note("Low C"), music.beat(1))  # play low C for 1 beat
buzzer.play(music.note("Middle C#"))            # play middle C sharp
buzzer.off()                                    # turn buzzer off
```

Constructors

`class ezblock.Buzzer(pwm)` Create an Buzzer object associated with the given pwm object. This allows you to then control buzzer.

Methods

- `on` - Turn the buzzer on with a square wave


```
Buzzer.on()
```

- off - Turn the buzzer off

```
Buzzer.off()
```

- freq - Set the square wave frequency

```
Buzzer.freq(frequency)
```

- play - Play freq, set off if a ms delay argument is provided.

```
Buzzer.play(freq, ms)
Buzzer.play(freq)
```

class Sound - sound sensor

Usage:

```
from ezblock import Sound, ADC

pin = ADC("A0")
sound = Sound(pin)                # create an Sound object from a pin
val = sound.read_raw()            # read an analog value

average_val = sound.read_raw(time = 100) # read an average analog value
```

Constructors

`class ezblock.Sound(pin)` Create an Sound object associated with the given pin. This allows you to then read analog values on that pin.

Methods

- read - Read the value on the analog pin and return it. The returned value will be between 0 and 4095.

```
Sound.read_raw()
```

class Joystick - 3-axis joystick

Usage:

```
from ezblock import Joystick, ADC, Pin

x_pin = ADC("A0")
y_pin = ADC("A1")
btn_pin = Pin("D1")

joystick = Joystick(x_pin, y_pin, btn_pin) # create an Joystick object from a
↪ pin
```

(continues on next page)

(continued from previous page)

```
val = joystick.read(0)           # read an axis value
status = joystick.read_status()  # read the status of joystick
```

Constructors

`class ezblock.Joystick(pin)` Create an Joystick object associated with the given pin. This allows you to then read values on that pin.

Methods

- `read` - Read the value on the given pin and return it.

```
Joystick.read(Xpin, Ypin, Btpin)
```

- `read_status` - Read the value on the given pin and return it.

```
Joystick.read_status()
```

class BLE - bluetooth driver

PiMobile

Methods

Classes

robothat Module - Robot Control Boards

Usage:

```
from robothat import *
motor_direction_calibration(1, 1)      # Calibrate motor direction of rotation
set_motor_speed(1, 50)                 # Set the speed of the motor
motor_speed_calibration(10)            # Speed calibration of motors
```

Constructors

`Robothat Module` `robothat Module` allows you to control the robothat board.

Methods

- `motor_direction_calibration` - Calibrate motor direction of rotation.(motor: 1 or 2,value: 0 or 1))

```
motor_direction_calibration(1,0)
```

- set_motor_speed - Set the speed of the motor.(motor: 1 or 2,value:0 ~ 100)

```
set_motor_speed(1,50)
```

- motor_speed_calibration - #Speed calibration of motors.(value: -100 ~ 100,If the value is greater than zero,the speed of motor 1 will increase the value.

```
motor_speed_calibration(10)
```