SunFounder Thales Kit for Raspberry Pi Pico

Release 1.0

Jimmy, SunFounder

Jan 10, 2023

CONTENTS

SunFounder Thales Kit is a basic learning kit based on Raspberry Pi Pico.

The kit contains various types of components, such as displays, sounds, drivers, controllers, and sensors, allows you to learn electronic devices comprehensively.

we have prepared many interesting and practical projects for you, and collected a lot of authoritative related information, just turn on your computer and you can complete programming learning in one stop.

In addition, we provide 2 programming languages: MicroPython and Arduino (C/C++). You can view different tutorials according to your needs

If you want to learn another projects which we don't have, please feel free to send Email and we will update to our online tutorials as soon as possible, any suggestions are welcomed.

Here is the Email: cs@sunfounder.com.

About the display language

In addition to English, we are working on other languages for this course. Please contact service@sunfounder.com if you are interested in helping, and we will give you a free product in return. In the meantime, we recommend using Google Translate to convert English to the language you want to see.

The steps are as follows.

- In this course page, right-click and select **Translate to xx**. If the current language is not what you want, you can change it later.
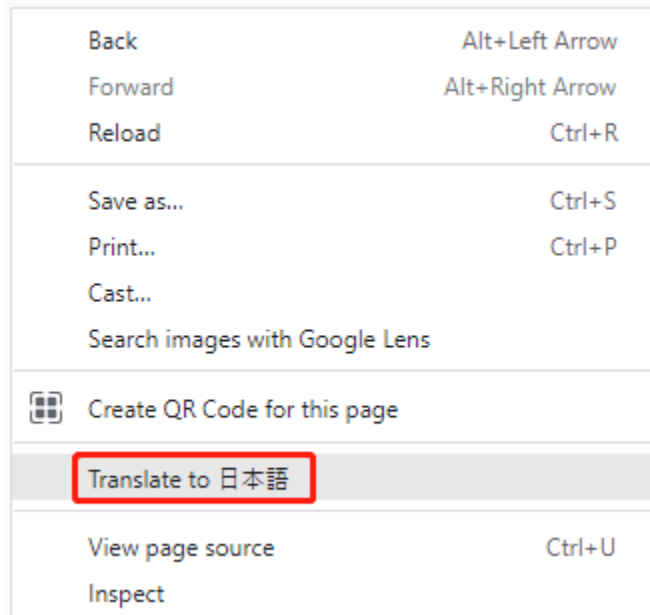
- There will be a language popup in the upper right corner. Click on the menu button to **choose another language**.
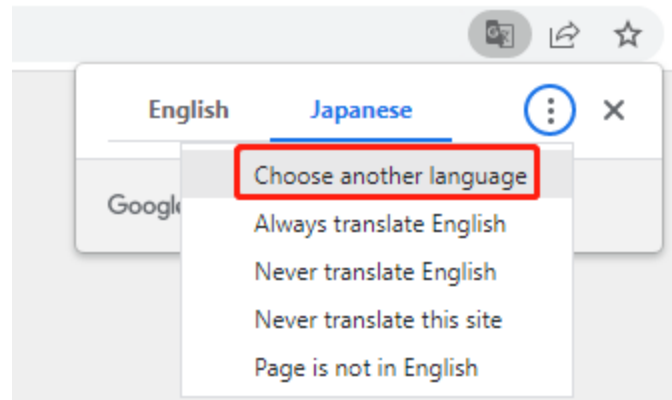
- Select the language from the inverted triangle box, and then click **Done**.
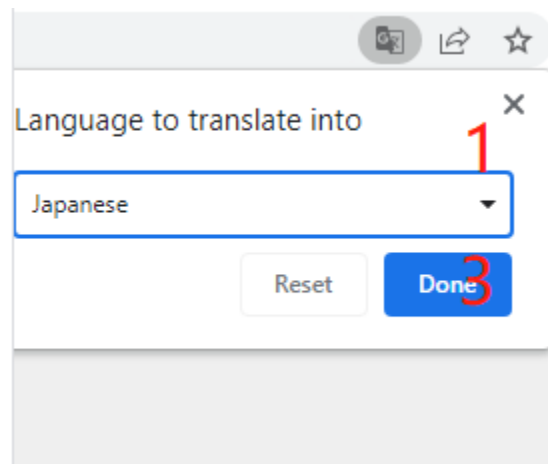
Source Code

- `SunFounder Thales Kit Scripts`

- Or check out the code at Thales Kit - GitHub

ONE

INTRODUCTION TO RASPBERRY PI PICO
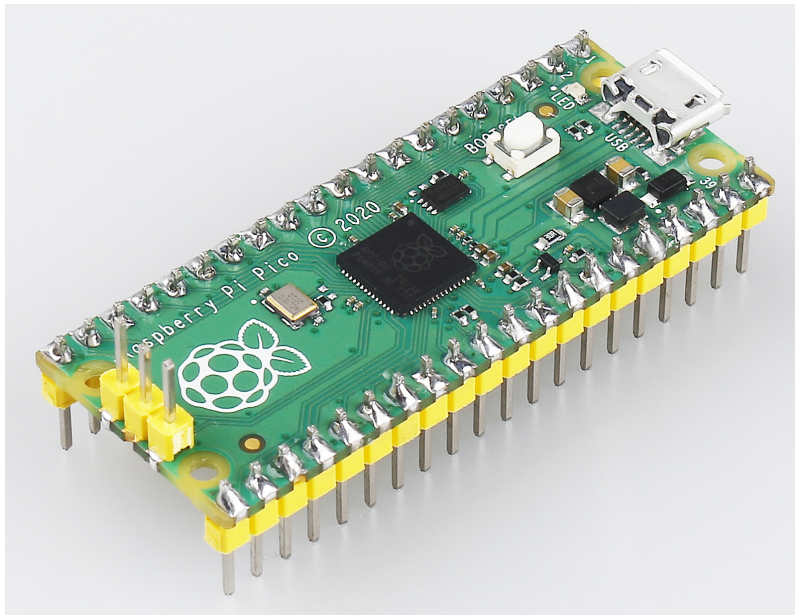
The Raspberry Pi Pico is a microcontroller board based on the Raspberry Pi RP2040 microcontroller chip.

Whether you want to learn the MicroPython programming language, take the first step in physical computing, or want to build a hardware project, Raspberry Pi Pico – and its amazing community – will support you every step of the way. In the project, it can control anything, from LEDs and buttons to sensors, motors, and even other microcontrollers.

1.1 Features

- 21 mm × 51 mm form factor

- RP2040 microcontroller chip designed by Raspberry Pi in the UK

- Dual-core Arm Cortex-M0+ processor, flexible clock running up to 133 MHz

- 264KB on-chip SRAM

- 2MB on-board QSPI Flash

- 26 multifunction GPIO pins, including 3 analog inputs

- 2 × UART, 2 × SPI controllers, 2 × I2C controllers, 16 × PWM channels

- 1 × USB 1.1 controller and PHY, with host and device support

- 8 × Programmable I/O (PIO) state machines for custom peripheral support

- Supported input power 1.8–5.5V DC

- Operating temperature -20°C to +85°C

- Castellated module allows soldering direct to carrier boards

- Drag-and-drop programming using mass storage over USB

- Low-power sleep and dormant modes

- Accurate on-chip clock

- Temperature sensor

- Accelerated integer and floating-point libraries on-chip

1.2 Pico's Pins
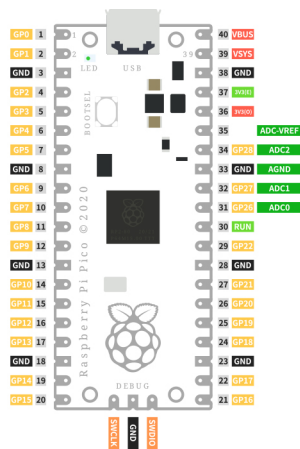
Name	Description	Function
GP0-GP28	General-purpose input/output pins	Act as either input or output and have no fixed purpose of their own
GND	0 volts ground	Several GND pins around Pico to make wiring easier.
RUN	Enables or diables your Pico	Start and stop your Pico from another microcontroller.
GPxx_ADCx	General-purpose input/output or analog input	Used as an analog input as well as a digital input or output – but not both at the same time.
ADC_VREF	Analog-to-digital converter (ADC) voltage reference	A special input pin which sets a reference voltage for any analog inputs.
AGND	Analog-to-digital converter (ADC) 0 volts ground	A special ground connection for use with the ADC_VREF pin.
3V3(O)	3.3 volts power	A source of 3.3V power, the same voltage your Pico runs at internally, generated from the VSYS input.
3v3(E)	Enables or disables the power	Switch on or off the 3V3(O) power, can also switches your Pico off.
VSYS	2-5 volts power	A pin directly connected to your Pico's internal power supply, which cannot be switched off without also switching Pico off.
VBUS	5 volts power	A source of 5V power taken from your Pico's micro USB port, and used to power hardware which needs more than 3.3V.

The best place to find everything you need to get started with your Raspberry Pi Pico.

Or you can click on the links below:

- Raspberry Pi Pico product brief
- Raspberry Pi Pico datasheet
- Getting started with Raspberry Pi Pico: C/C++ development
- Raspberry Pi Pico C/C++ SDK
- API-level Doxygen documentation for the Raspberry Pi Pico C/C++ SDK
- Raspberry Pi Pico Python SDK
- Raspberry Pi RP2040 datasheet
- Hardware design with RP2040
- Raspberry Pi Pico design files
- Raspberry Pi Pico STEP file
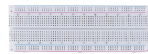
WHAT IS INCLUDED IN THIS KIT

The following is a list of this kit, after you receive the goods, you can check the contents of the list first.

Some of the components on the kit are very small and look the same, the worker may omit or send it wrong when packing. Please do not hesitate to send us the picture of the kit and the names of the missing or wrong components via email.

Here is the Email: cs@sunfounder.com.

2.1 Components List

Red LED 10 pcs	**Blue LED** 10 pcs	**Green LED** 10 pcs	**White LED** 10 pcs
Yellow LED 10 pcs	**RGB LED** 1 pcs	**Resistor(10Ω)** 10 pcs	**Resistor(100Ω)** 10 pcs
Resistor(220Ω) 10 pcs	**Resistor(330Ω)** 10 pcs	**Resistor(1KΩ)** 10 pcs	**Resistor(2KΩ)** 10 pcs
Resistor(5.1KΩ) 10 pcs	**Resistor(10KΩ)** 10 pcs	**Resistor(100KΩ)** 10 pcs	**Resistor(1MΩ)** 10 pcs
S8550 Transistor 2 pcs	**S8050 Transistor** 2 pcs	**Tilt Switch** 1 pcs	**Potentiometer** 2 pcs
100NF Capacitor 10 pcs	**10NF Capacitor** 10 pcs	**Photoresistor** 2 pcs	**Thermistor** 1 pcs
9G Servo 1 pcs	**PIR Motion Sensor** 1 pcs	**Breadboard** 1 pcs	**74HC595** 1 pcs
Passive Buzzer 1 pcs	**Active Buzzer** 1 pcs	**7-segment Display** 1 pcs	**I2C LCD 1602** 1 pcs
Slide Switch 5 pcs	**Button** 10 pcs	**WS2812 RGB 8 LEDs Strip** 1 pcs	**Micro USB Cable** 1 pcs
Jump Wires Box 1 pcs	**Jumper Wires** 65 pcs	**Dupont Wires** 10 pcs	

2.2 Components Introduction

2.2.1 Breadboard
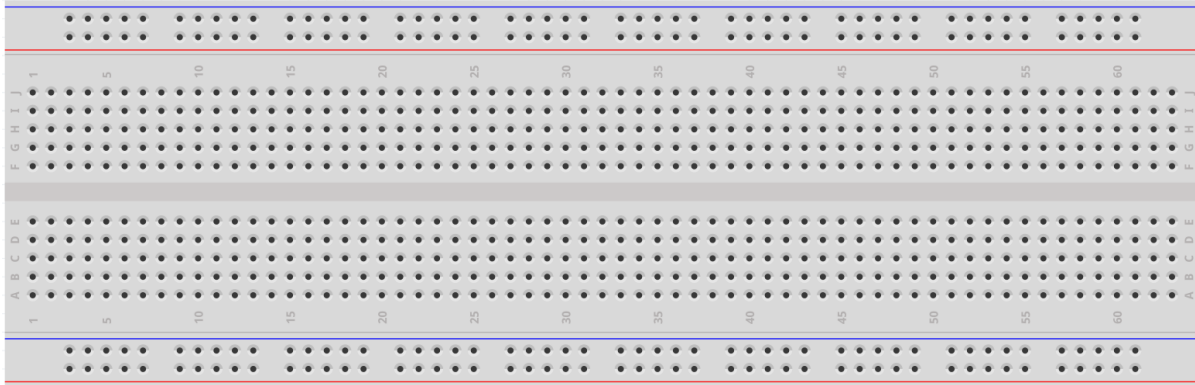
A breadboard is a construction base for prototyping of electronics. Originally the word referred to a literal bread board, a polished piece of wood used for slicing bread.[1] In the 1970s the solderless breadboard (a.k.a. plugboard, a terminal array board) became available and nowadays the term "breadboard" is commonly used to refer to these.

It is used to build and test circuits quickly before finishing any circuit design. And it has many holes into which components mentioned above can be inserted like ICs and resistors as well as jumper wires. The breadboard allows you to plug in and remove components easily.

The picture shows the internal structure of a breadboard. Although these holes on the breadboard appear to be independent of each other, they are actually connected to each other through metal strips internally.
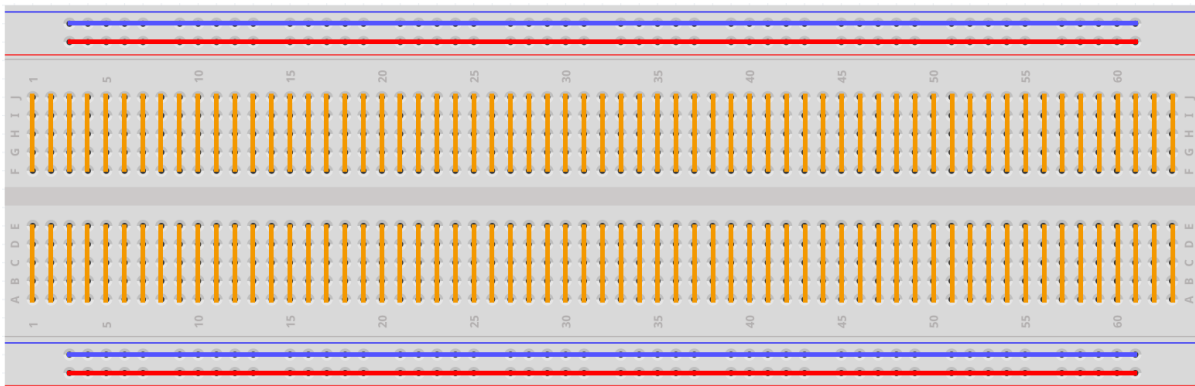
If you want to know more about breadboard, refer to: How to Use a Breadboard - Science Buddies

Example

- *2.2 Hello, Breadboard!* (For MicroPython User

2.2.2 LED
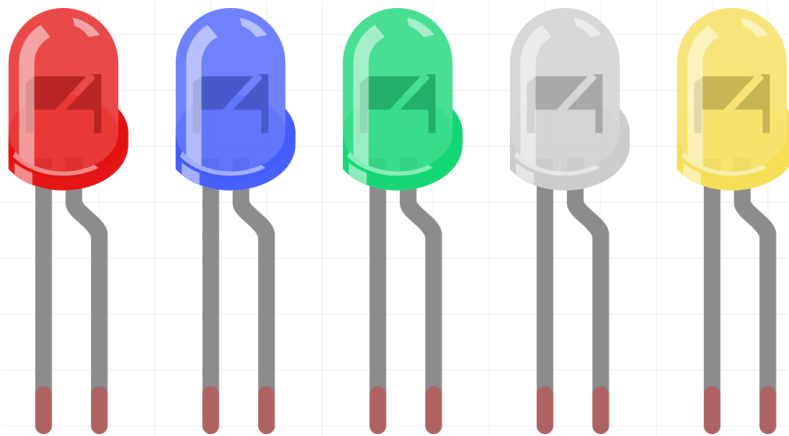
Semiconductor light-emitting diode is a type of component which can turn electric energy into light energy via PN junctions. By wavelength, it can be categorized into laser diode, infrared light-emitting diode and visible light-emitting diode which is usually known as light-emitting diode (LED).

Diode has unidirectional conductivity, so the current flow will be as the arrow indicates in figure circuit symbol. You can only provide the anode with a positive power and the cathode with a negative. Thus the LED will light up.
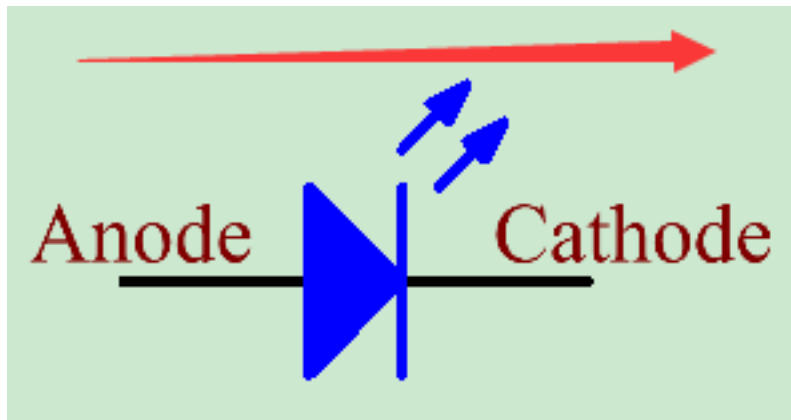
An LED has two pins. The longer one is the anode, and shorter one, the cathode. Pay attention not to connect them inversely. There is fixed forward voltage drop in the LED, so it cannot be connected with the circuit directly because the supply voltage can outweigh this drop and cause the LED to be burnt. The forward voltage of the red, yellow, and green LED is 1.8 V and that of the white one is 2.6 V. Most LEDs can withstand a maximum current of 20 mA, so we need to connect a current limiting resistor in series.

The formula of the resistance value is as follows:

R = (Vsupply – VD)/I

R stands for the resistance value of the current limiting resistor, **Vsupply** for voltage supply, **VD** for voltage drop and **I** for the working current of the LED.

Here is the detailed introduction for the LED: LED - Wikipedia.

Example

- *2.2 Hello, Breadboard!* (For MicroPython User

- *2.8 Fading LED* (For MicroPython User

- *Fading LED* (For C/C++(Arduino) User
- *Hello LED* (For C/C++(Arduino) User

2.2.3 RGB LED
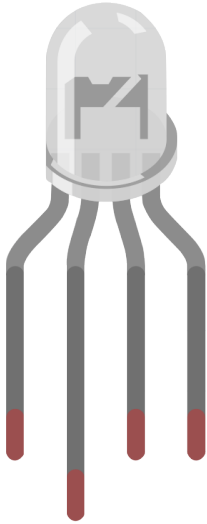
RGB LEDs emit light in various colors. An RGB LED packages three LEDs of red, green, and blue into a transparent or semitransparent plastic shell. It can display various colors by changing the input voltage of the three pins and superimpose them, which, according to statistics, can create 16,777,216 different colors.
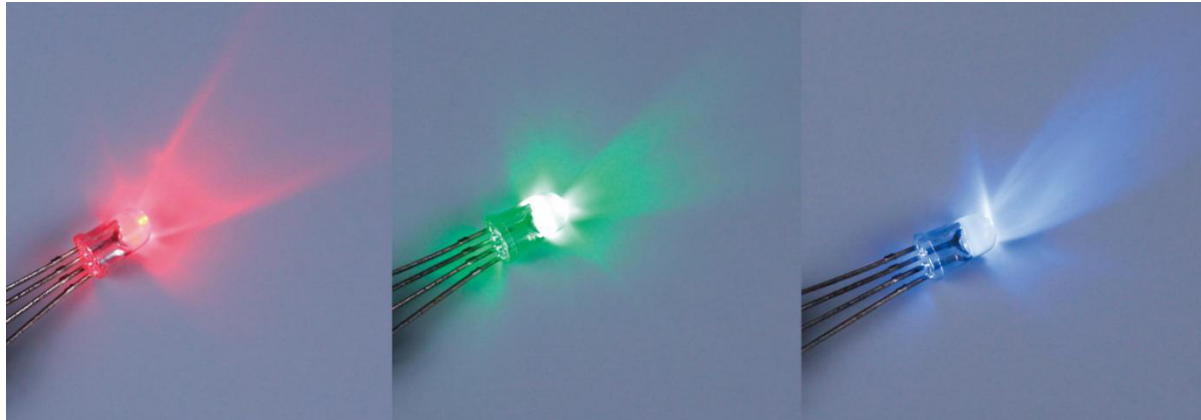
RGB LEDs can be categorized into common anode and common cathode ones. In this kit, the latter is used. The **common cathode**, or CC, means to connect the cathodes of the three LEDs. After you connect it with GND and plug in the three pins, the LED will flash the corresponding color.

Its circuit symbol is shown as figure.
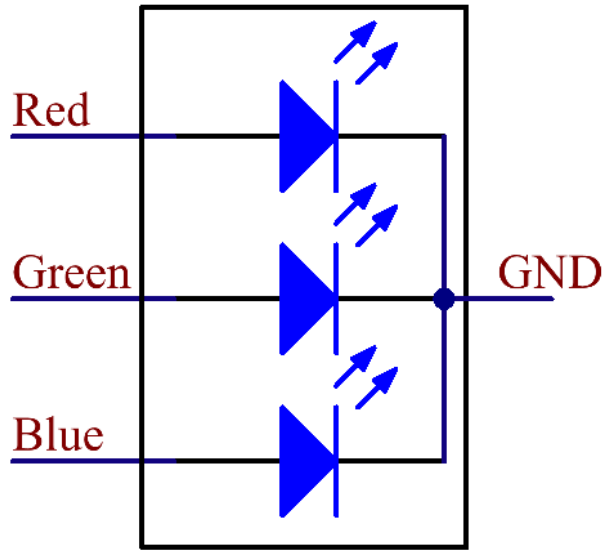
An RGB LED has 4 pins: the longest one is GND; the others are Red, Green and Blue. Touch its plastic shell and you will find a cut. The pin closest to the cut is the first pin, marked as Red, then GND, Green and Blue in turn.
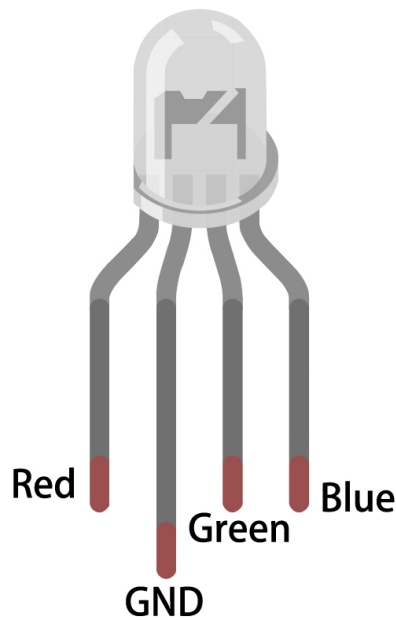
Example

- *2.9 Colorful Light* (For MicroPython User
- *RGB LED* (For C/C++(Arduino) User

2.2.4 Resistor

Resistor is an electronic element that can limit the branch current. A fixed resistor is a kind of resistor whose resistance cannot be changed, while that of a potentiometer or a variable resistor can be adjusted.

Two generally used circuit symbols for resistor. Normally, the resistance is marked on it. So if you see these symbols in a circuit, it stands for a resistor.

 is the unit of resistance and the larger units include K, M, etc. Their relationship can be shown as follows: 1 M=1000 K, 1 K = 1000 . Normally, the value of resistance is marked on it.

When using a resistor, we need to know its resistance first. Here are two methods: you can observe the bands on the resistor, or use a multimeter to measure the resistance. You are recommended to use the first method as it is more convenient and faster.
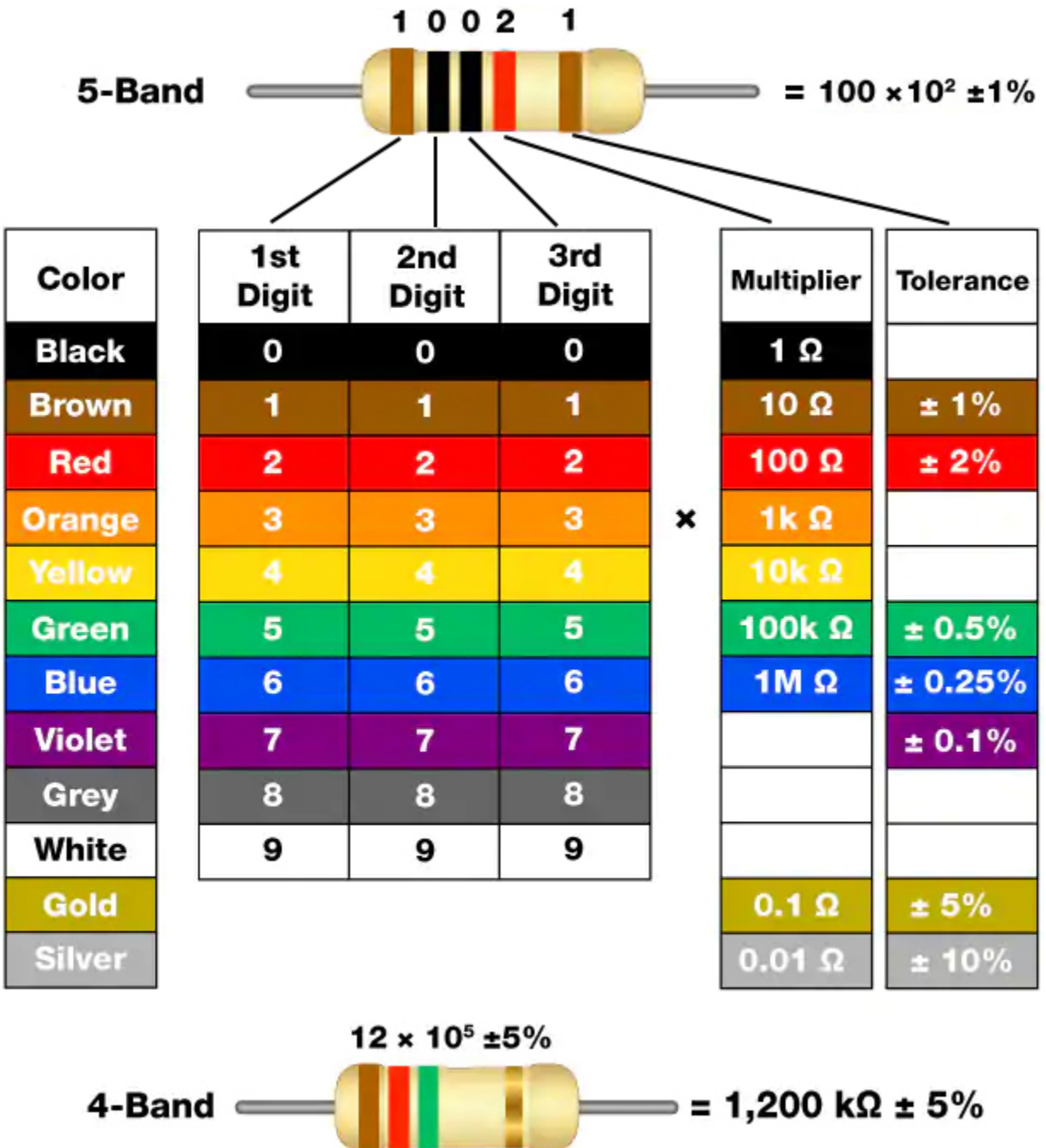
As shown in the card, each color stands for a number.

Black	Brown	Red	Orange	Yellow	Green	Blue	Violet	Grey	White	Gold	Silver
0	1	2	3	4	5	6	7	8	9	0.1	0.01

The 4- and 5-band resistors are frequently used, on which there are 4 and 5 chromatic bands.

Normally, when you get a resistor, you may find it hard to decide which end to start for reading the color. The tip is that the gap between the 4th and 5th band will be comparatively larger.

Therefore, you can observe the gap between the two chromatic bands at one end of the resistor; if it's larger than any other band gaps, then you can read from the opposite side.

Let's see how to read the resistance value of a 5-band resistor as shown below.
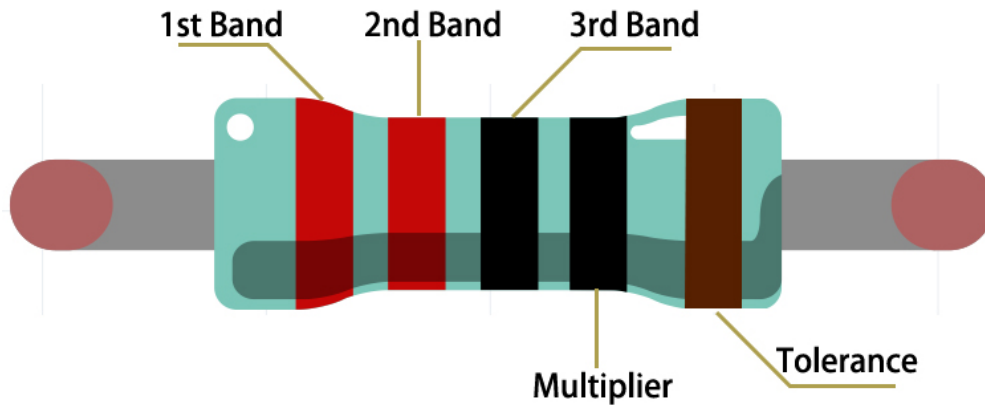
So for this resistor, the resistance should be read from left to right. The value should be in this format: 1st Band 2nd Band 3rd Band x 10^Multiplier () and the permissible error is ±Tolerance%. So the resistance value of this resistor is 2(red) 2(red) 0(black) x 10^0(black) = 220 , and the permissible error is ± 1% (brown).

You can learn more about resistor from Wiki: Resistor - Wikipedia.

2.2.5 Potentiometer
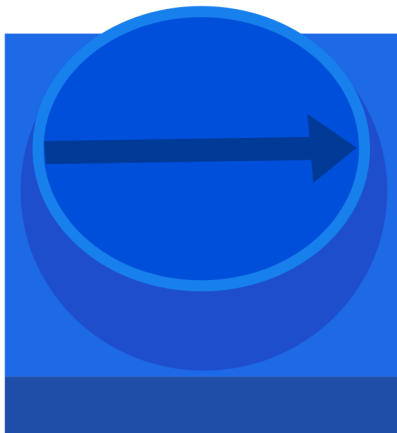
Potentiometer is also a resistance component with 3 terminals and its resistance value can be adjusted according to some regular variation.

Potentiometers come in various shapes, sizes, and values, but they all have the following things in common:

- They have three terminals (or connection points).
- They have a knob, screw, or slider that can be moved to vary the resistance between the middle terminal and either one of the outer terminals.
- The resistance between the middle terminal and either one of the outer terminals varies from 0 to the maximum resistance of the pot as the knob, screw, or slider is moved.

Here is the circuit symbol of potentiometer.
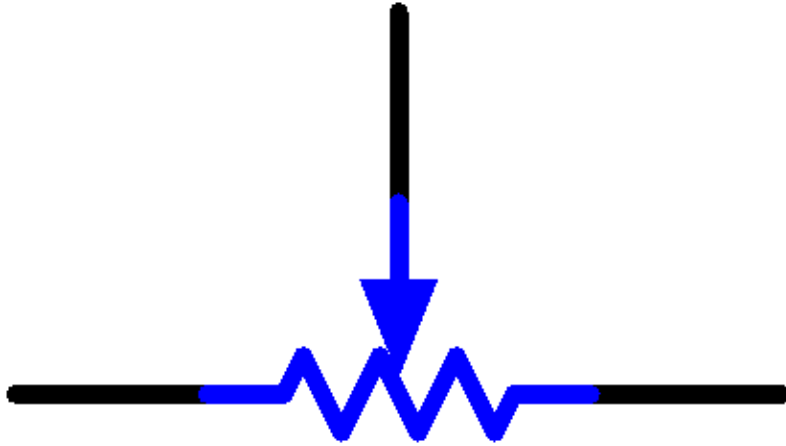
The functions of the potentiometer in the circuit are as follows:

1. Serving as a voltage divider

   Potentiometer is a continuously adjustable resistor. When you adjust the shaft or sliding handle of the potentiometer, the movable contact will slide on the resistor. At this point, a voltage can be output depending on the voltage applied onto the potentiometer and the angle the movable arm has rotated to or the travel it has made.

2. Serving as a rheostat

   When the potentiometer is used as a rheostat, connect the middle pin and one of the other 2 pins in the circuit. Thus you can get a smoothly and continuously changed resistance value within the travel of the moving contact.

3. Serving as a current controller

   When the potentiometer acts as a current controller, the sliding contact terminal must be connected as one of the output terminals.

If you want to know more about potentiometer, refer to: Potentiometer - Wikipedia

Example

- *2.12 Turn the Knob* (For MicroPython User
- *Table Lamp* (For C/C++(Arduino) User

2.2.6 Photoresistor
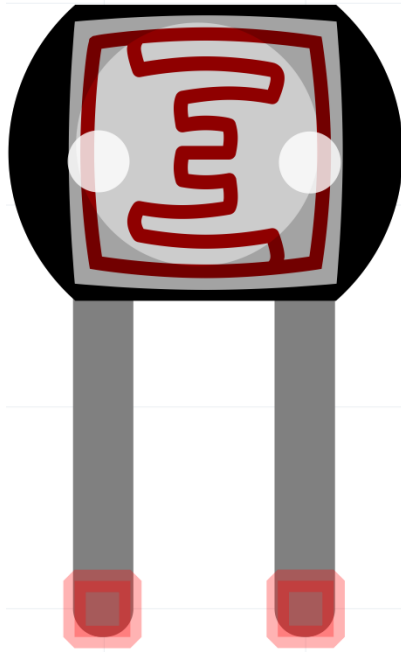
A photoresistor or photocell is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity; in other words, it exhibits photo conductivity.

A photoresistor can be applied in light-sensitive detector circuits and light-activated and dark-activated switching circuits acting as a resistance semiconductor. In the dark, a photoresistor can have a resistance as high as several megaohms (M), while in the light, a photoresistor can have a resistance as low as a few hundred ohms.

Here is the electronic symbol of photoresistor.
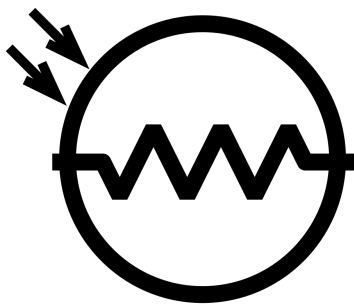
- Photoresistor - Wikipedia

Example

- *2.14 Light Theremin* (For MicroPython User
- *Measure Light Intensity* (For C/C++(Arduino) User
- *Light Theremin* (For C/C++(Arduino) User

2.2.7 Thermistor
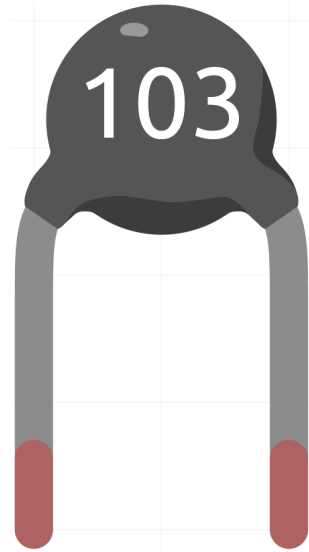
A thermistor is a type of resistor whose resistance is strongly dependent on temperature, more so than in standard resistors. The word is a combination of thermal and resistor. Thermistors are widely used as inrush current limiters, temperature sensors (negative temperature coefficient or NTC type typically), self-resetting overcurrent protectors, and self-regulating heating elements (positive temperature coefficient or PTC type typically).

- Thermistor - Wikipedia

Here is the electronic symbol of thermistor.
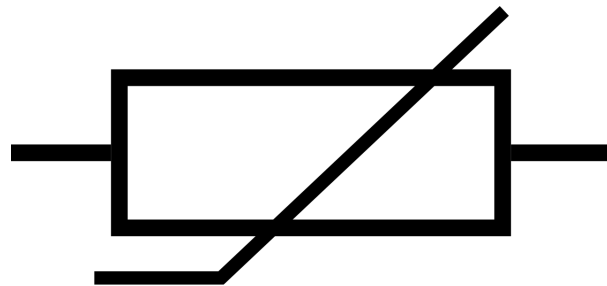
Thermistors are of two opposite fundamental types:

- With NTC thermistors, resistance decreases as temperature rises usually due to an increase in conduction electrons bumped up by thermal agitation from valency band. An NTC is commonly used as a temperature sensor, or in series with a circuit as an inrush current limiter.

- With PTC thermistors, resistance increases as temperature rises usually due to increased thermal lattice agitations particularly those of impurities and imperfections. PTC thermistors are commonly installed in series with a circuit, and used to protect against overcurrent conditions, as resettable fuses.

In this kit we use an NTC one. Each thermistor has a normal resistance. Here it is 10k ohm, which is measured under 25 degree Celsius.

Here is the relation between the resistance and temperature:

$$RT = RN * \exp B(1/TK - 1/TN)$$

- **RT** is the resistance of the NTC thermistor when the temperature is TK.

- **RN** is the resistance of the NTC thermistor under the rated temperature TN. Here, the numerical value of RN is 10k.

- **TK** is a Kelvin temperature and the unit is K. Here, the numerical value of TK is 273.15 + degree Celsius.

- **TN** is a rated Kelvin temperature; the unit is K too. Here, the numerical value of TN is 273.15+25.

- And **B(beta)**, the material constant of NTC thermistor, is also called heat sensitivity index with a numerical value 3950.

- **exp** is the abbreviation of exponential, and the base number e is a natural number and equals 2.7 approximately.

Convert this formula TK=1/(ln(RT/RN)/B+1/TN) to get Kelvin temperature that minus 273.15 equals degree Celsius.

This relation is an empirical formula. It is accurate only when the temperature and resistance are within the effective range.

Example

- *2.13 Thermometer* (For MicroPython User

2.2.8 Transistor
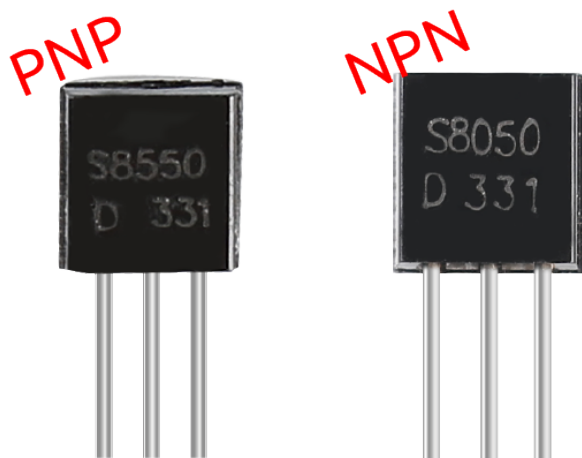
Transistor is a semiconductor device that controls current by current. It functions by amplifying weak signal to larger amplitude signal and is also used for non-contact switch.

A transistor is a three-layer structure composed of P-type and N-type semiconductors. They form the three regions internally. The thinner in the middle is the base region; the other two are both N-type or P-type ones – the smaller region with intense majority carriers is the emitter region, when the other one is the collector region. This composition enables the transistor to be an amplifier. From these three regions, three poles are generated respectively, which are base (b), emitter (e), and collector (c). They form two P-N junctions, namely, the emitter junction and collection junction. The direction of the arrow in the transistor circuit symbol indicates that of the emitter junction.

- P–N junction - Wikipedia

Based on the semiconductor type, transistors can be divided into two groups, the NPN and PNP ones. From the abbreviation, we can tell that the former is made of two N-type semiconductors and one P-type and that the latter is the opposite. See the figure below.

Note: s8550 is PNP transistor and the s8050 is the NPN one, They look very similar, and we need to check carefully to see their labels.
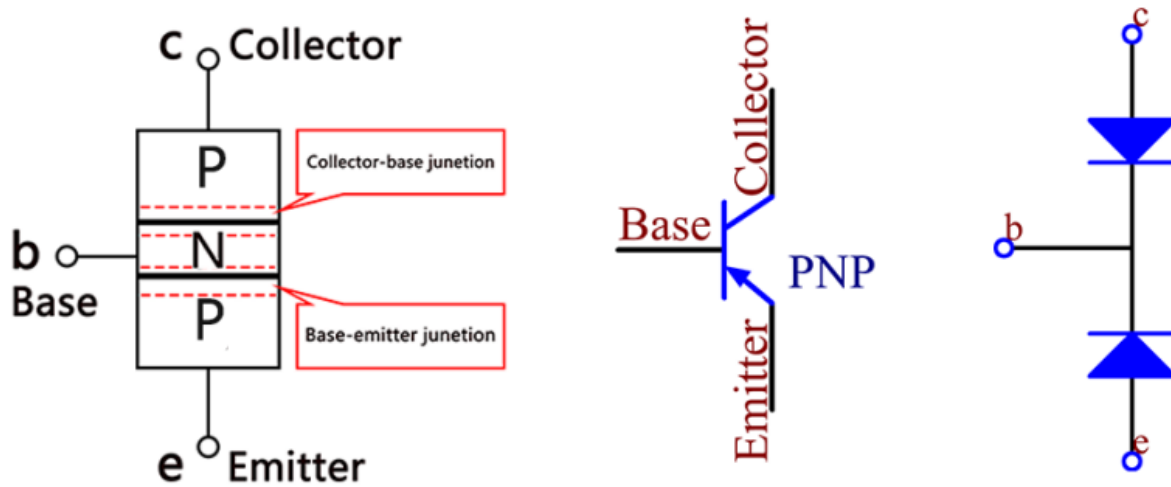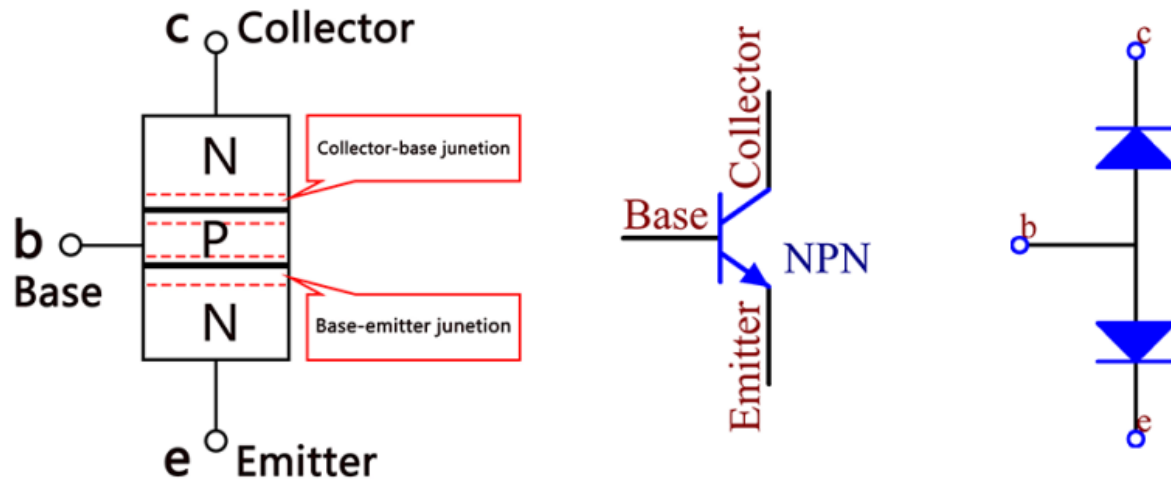
C o Collector

N

Collector-base junetion

b o P

Base

N

Base-emitter junetion

e o Emitter

Base

Collector

Emitter

NPN

c

b

e

C o Collector

P

Collector-base junetion

b o N

Base

P

Base-emitter junetion

e o Emitter

Base

Collector

Emitter

PNP

c

b

e

When a High level signal goes through an NPN transistor, it is energized. But a PNP one needs a Low level signal to manage it. Both types of transistor are frequently used for contactless switches, just like in this experiment.

Put the label side facing us and the pins facing down. The pins from left to right are emitter(e), base(b), and collector(c).

- S8050 Transistor Datasheet
- S8550 Transistor Datasheet

Example

- *2.6 Two Kinds of Transistors* (For MicroPython User
- *Light Theremin* (For C/C++(Arduino) User
- *Doorbell* (For C/C++(Arduino) User

2.2.9 Capacitor
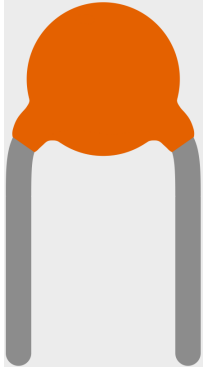
Capacitance, refers to the amount of charge storage under a given potential difference, denoted as C, and the international unit is farad (F). Generally speaking, electric charges move under force in an electric field. When there is a medium between conductors, the movement of electric charges is hindered and the electric charges accumulate on the conductors, resulting in accumulation of electric charges.

The amount of stored electric charges is called capacitance. Because capacitors are one of the most widely used electronic components in electronic equipment, they are widely used in direct current isolation, coupling, bypass, filtering, tuning loops, energy conversion, and control circuits. Capacitors are divided into electrolytic capacitors, solid capacitors, etc.

According to material characteristics, capacitors can be divided into: aluminum electrolytic capacitors, film capacitors, tantalum capacitors, ceramic capacitors, super capacitors, etc.

  • Ceramic Capacitor - Wikipedia

In this kit, ceramic capacitors are used. There are 103 or 104 label on the ceramic capacitors, which represent the capacitance value, $103=10 \times 10^3 pF$, $104=10 \times 10^4 pF$

Unit Conversion $1F=10^3 mF=10^6 uF=10^9 nF=10^{12} pF$

Example

- *2.3 Reading Button Value* (For MicroPython User

- *Warning Light* (For C/C++(Arduino) User

2.2.10 Button
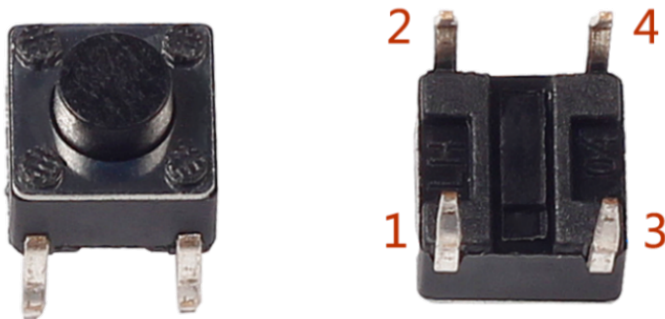
Buttons are a common component used to control electronic devices. They are usually used as switches to connect or break circuits. Although buttons come in a variety of sizes and shapes, the one used here is a 6mm mini-button as shown in the following pictures. Pin 1 is connected to pin 2 and pin 3 to pin 4. So you just need to connect either of pin 1 and pin 2 to pin 3 or pin 4.

The following is the internal structure of a button. The symbol on the right below is usually used to represent a button in circuits.
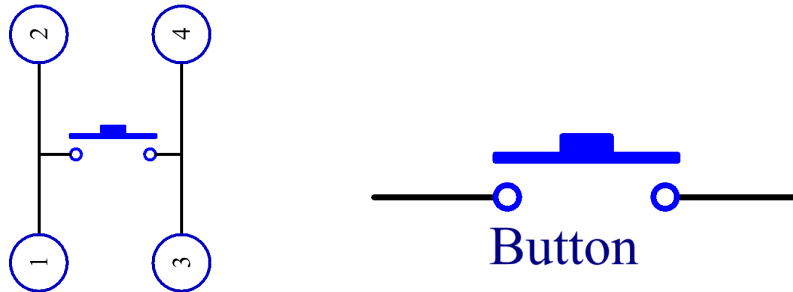
Since the pin 1 is connected to pin 2, and pin 3 to pin 4, when the button is pressed, the 4 pins are connected, thus closing the circuit.
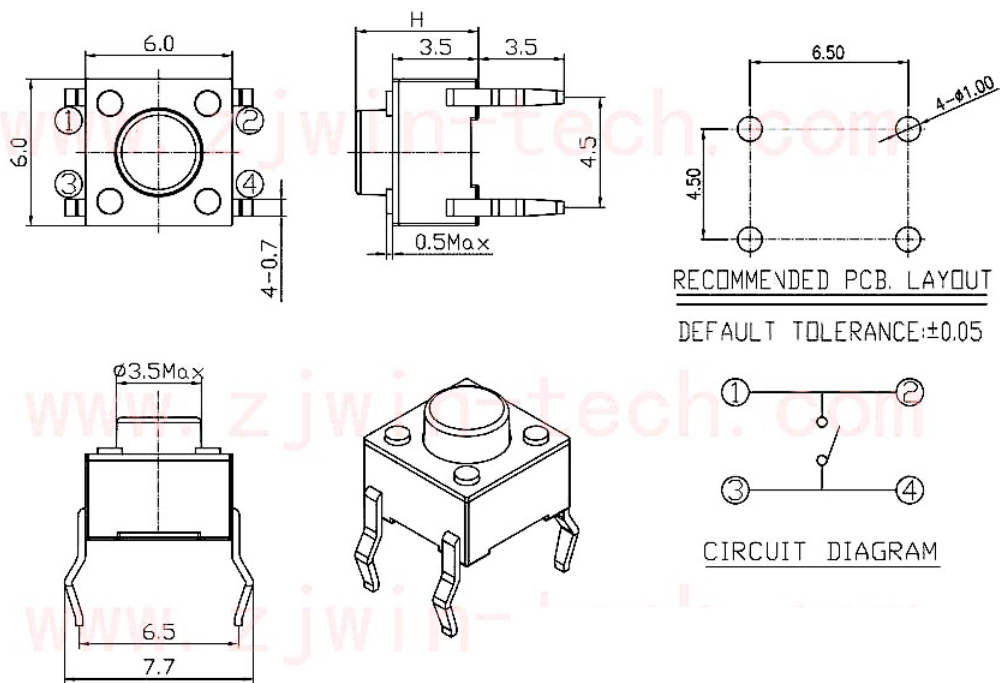
Examples

- *2.3 Reading Button Value* (For MicroPython User
- *Doorbell* (For C/C++(Arduino) User
- :*LUMI Piano* (For C/C++(Arduino) User

2.2.11 Tilt Switch

The tilt switch used here is a ball one with a metal ball inside. It is used to detect inclinations of a small angle.

The principle is very simple. When the switch is tilted in a certain angle, the ball inside rolls down and touches the two contacts connected to the pins outside, thus triggering circuits. Otherwise the ball will stay away from the contacts, thus breaking the circuits.
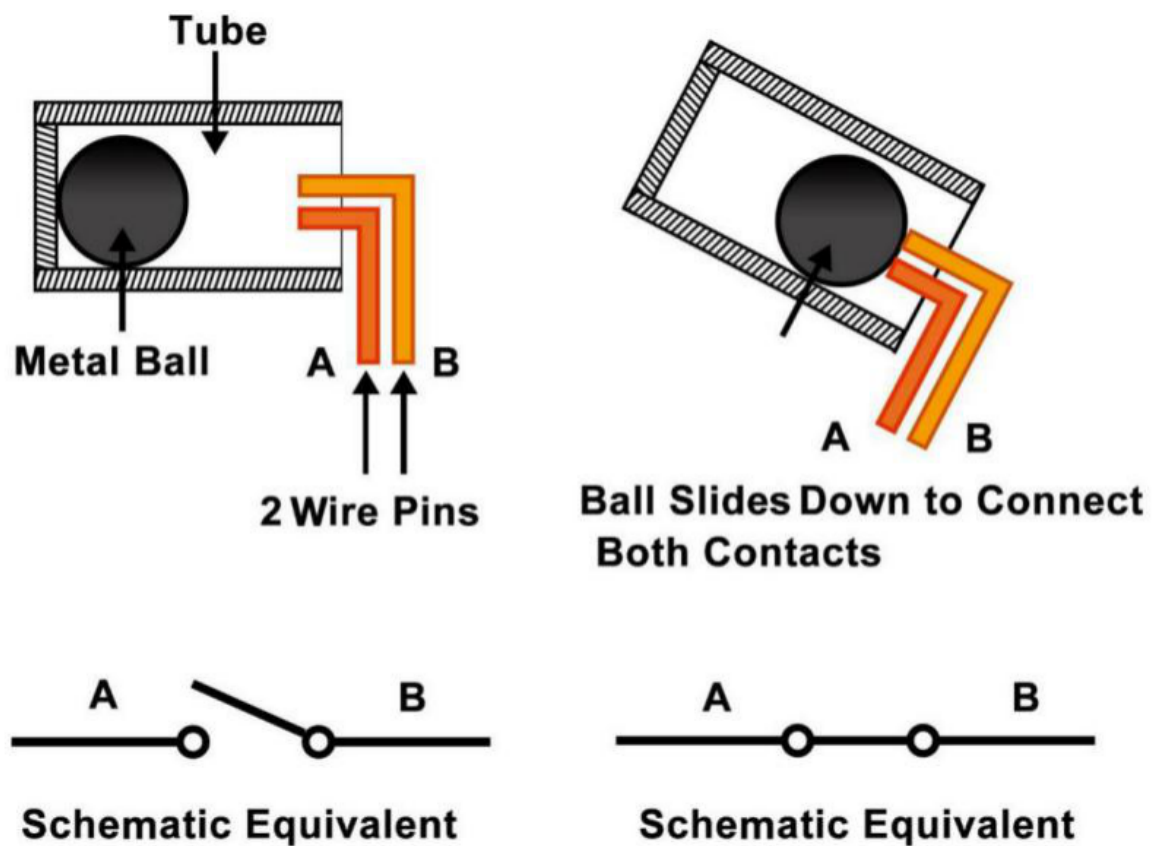
- SW520D Tilt Switch Datasheet

Examples

- *2.3 Reading Button Value* (For MicroPython User)
- *Flow LEDs* (For C/C++(Arduino) User)

2.2.12 Slide Switch
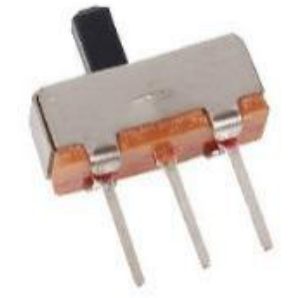
A slide switch, just as its name implies, is to slide the switch bar to connect or break the circuit, and further switch circuits. The common-used types are SPDT, SPTT, DPDT, DPTT etc. The slide switch is commonly used in low-voltage circuit. It has the features of flexibility and stability, and applies in electric instruments and electric toys widely. How it works: Set the middle pin as the fixed one. When you pull the slide to the left, the two pins on the left are connected; when you pull it to the right, the two pins on the right are connected. Thus, it works as a switch connecting or disconnecting circuits. See the figure below:

The circuit symbol of the slide switch is shown as below. The pin2 in the figure refers to the middle pin.
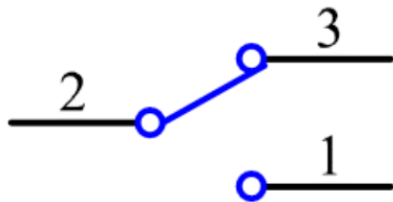
Example

- *2.3 Reading Button Value* (For MicroPython User)
- *Warning Light* (For C/C++(Arduino) User)

2.2.13 PIR Motion Sensor
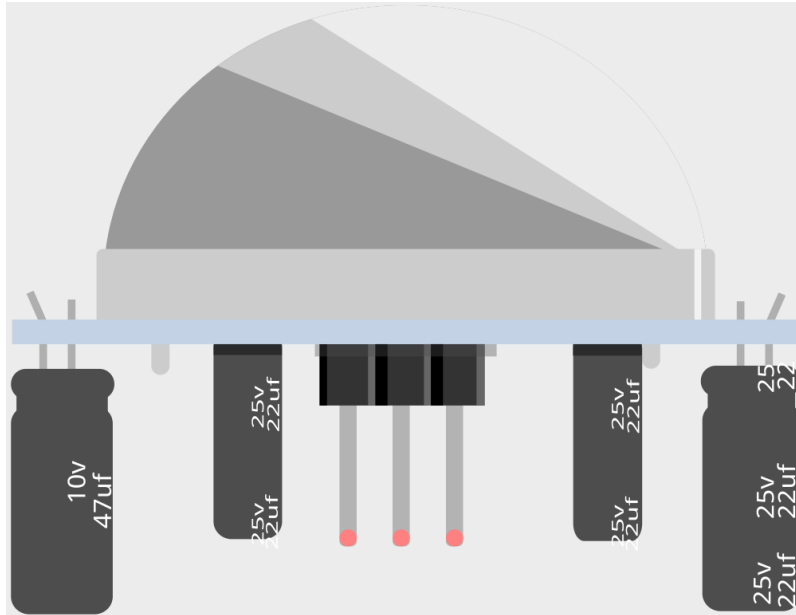
The PIR sensor detects infrared heat radiation that can be used to detect the presence of organisms that emit infrared heat radiation.

The PIR sensor is split into two slots that are connected to a differential amplifier. Whenever a stationary object is in front of the sensor, the two slots receive the same amount of radiation and the output is zero. Whenever a moving object is in front of the sensor, one of the slots receives more radiation than the other , which makes the output fluctuate high or low. This change in output voltage is a result of detection of motion.
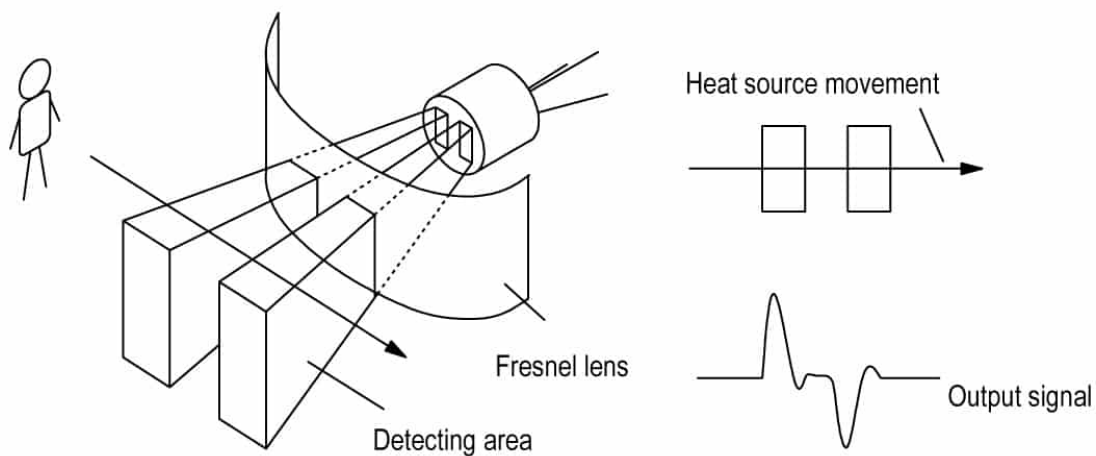
After the sensing module is wired, there is a one-minute initialization. During the initialization, module will output for 0~3 times at intervals. Then the module will be in the standby mode. Please keep the interference of light source and other sources away from the surface of the module so as to avoid the misoperation caused by the interfering signal. Even you'd better use the module without too much wind, because the wind can also interfere with the sensor.
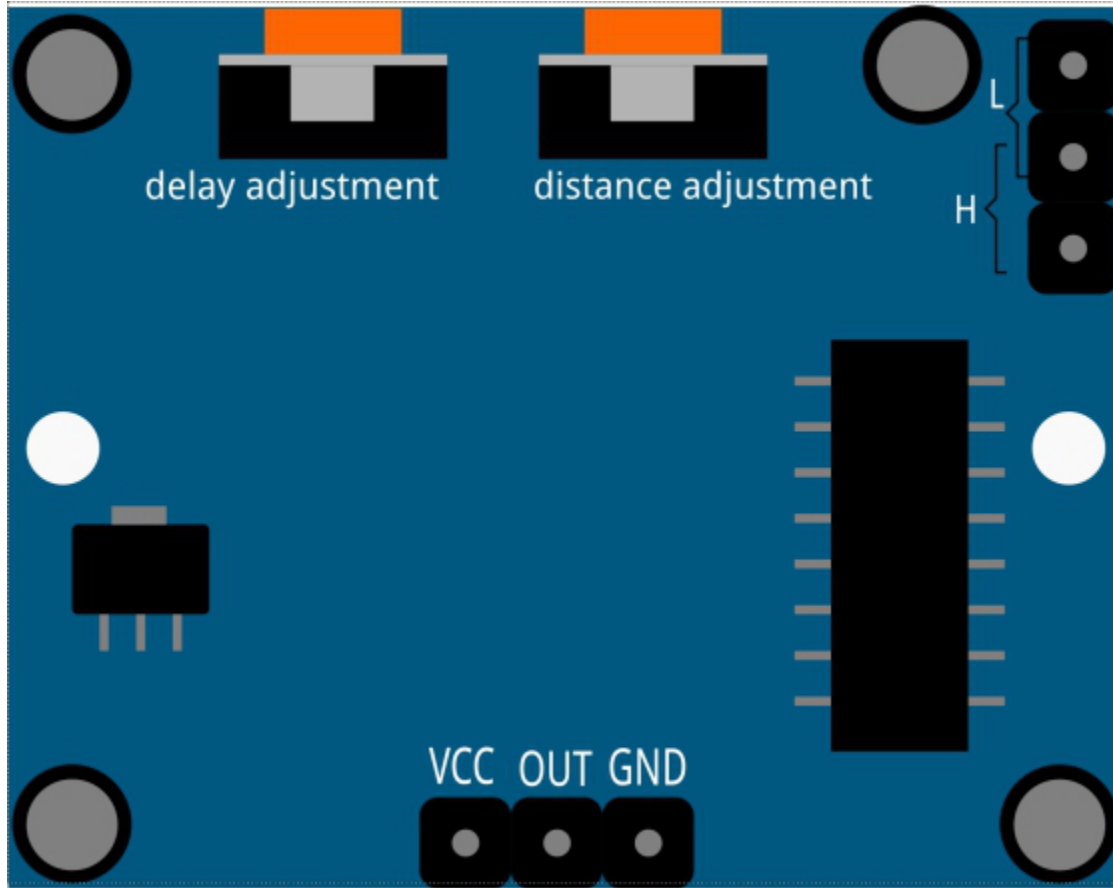
Distance Adjustment

Turning the knob of the distance adjustment potentiometer clockwise, the range of sensing distance increases, and the maximum sensing distance range is about 0-7 meters. If turn it anticlockwise, the range of sensing distance is reduced, and the minimum sensing distance range is about 0-3 meters.

Delay adjustment

Rotate the knob of the delay adjustment potentiometer clockwise, you can also see the sensing delay increasing. The maximum of the sensing delay can reach up to 300s. On the contrary, if rotate it anticlockwise, you can shorten the delay with a minimum of 5s.

Two Trigger Modes

Choosing different modes by using the jumper cap.

- **H**: Repeatable trigger mode, after sensing the human body, the module outputs high level. During the subsequent delay period, if somebody enters the sensing range,the output will keep being the high level.

- **L**Non-repeatable trigger mode, outputs high level when it senses the human body. After the delay, the output will change from high level into low level automatically.

Example

2.7 Intruder Alarm (For MicroPython User

2.2.14 Buzzer

As a type of electronic buzzer with an integrated structure, buzzers, which are supplied by DC power, are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic devices, telephones, timers and other electronic products or voice devices.

Buzzers can be categorized as active and passive ones (see the following picture). Turn the buzzer so that its pins are facing up, and the buzzer with a green circuit board is a passive buzzer, while the one enclosed with a black tape is an active one.

The difference between an active buzzer and a passive buzzer:

An active buzzer has a built-in oscillating source, so it will make sounds when electrified. But a passive buzzer does not have such source, so it will not beep if DC signals are used; instead, you need to use square waves whose frequency is between 2K and 5K to drive it. The active buzzer is often more expensive than the passive one because of multiple built-in oscillating circuits.

The following is the electrical symbol of a buzzer. It has two pins with positive and negative poles. With a + in the surface represents the anode and the other is the cathode.
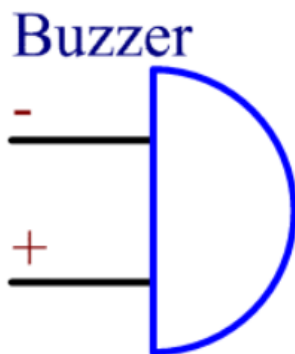
You can check the pins of the buzzer, the longer one is the anode and the shorter one is the cathode. Please don't mix them up when connecting, otherwise the buzzer will not make sound.

Buzzer - Wikipedia

Example

- *2.7 Intruder Alarm* (For MicroPython User
- *2.10 Custom Tone* (For MicroPython User
- *Doorbell* (For C/C++(Arduino) User

- :*LUMI Piano* (For C/C++(Arduino) User

2.2.15 Servo

A servo is generally composed of the following parts: case, shaft, gear system, potentiometer, DC motor, and embedded board.

It works like this: The microcontroller sends out PWM signals to the servo, and then the embedded board in the servo receives the signals through the signal pin and controls the motor inside to turn. As a result, the motor drives the gear system and then motivates the shaft after deceleration. The shaft and potentiometer of the servo are connected together. When the shaft rotates, it drives the potentiometer, so the potentiometer outputs a voltage signal to the embedded board. Then the board determines the direction and speed of rotation based on the current position, so it can stop exactly at the right position as defined and hold there.
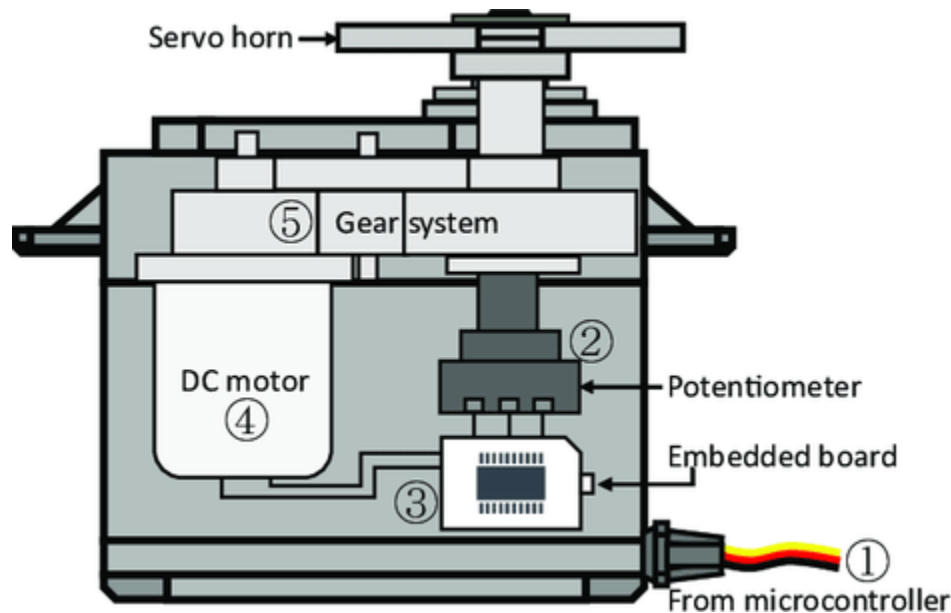
The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse width Modulation. The servo expects to see a pulse every 20 ms. The length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90 degree position (neutral position). When a pulse is sent to a servo that is less than 1.5 ms, the servo rotates to a position and holds its output shaft some number of degrees counterclockwise from the neutral point. When the pulse is wider than 1.5 ms the opposite occurs. The minimal width and the maximum width of pulse that will command the servo to turn to a valid position are functions of each servo. Generally the minimum pulse will be about 0.5 ms wide and the maximum pulse will be 2.5 ms wide.
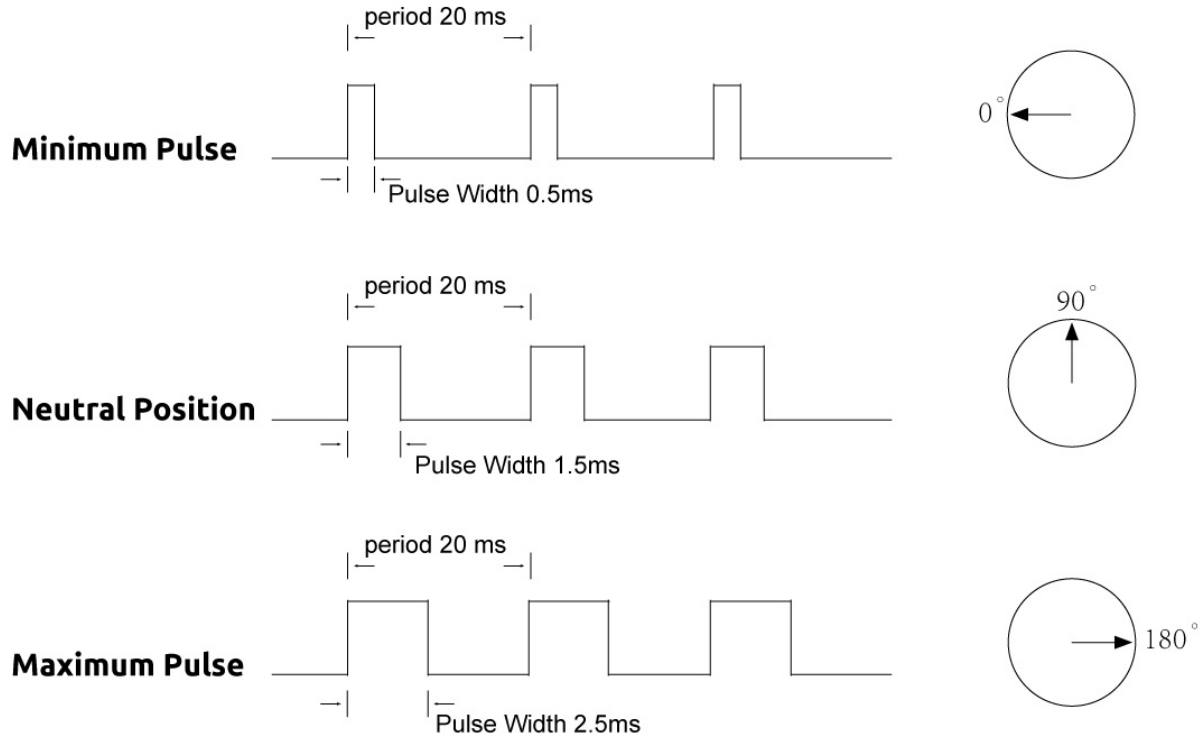
Example

- *2.11 Swinging Servo* (For MicroPython User

2.2.16 WS2812 RGB 8 LEDs Strip
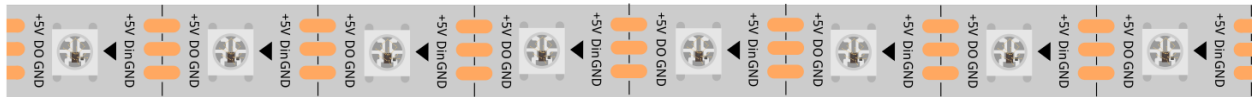
The WS2812 RGB 8 LEDs Strip is composed of 8 RGB LEDs. Only one pin is required to control all the LEDs. Each RGB LED has a WS2812 chip, which can be controlled independently. It can realize 256-level brightness display and complete true color display of 16,777,216 colors. At the same time, the pixel contains an intelligent digital interface data latch signal shaping amplifier drive circuit, and a signal shaping circuit is built in to effectively ensure the color height of the pixel point light Consistent.

It is flexible, can be docked, bent, and cut at will, and the back is equipped with adhesive tape, which can be fixed on the uneven surface at will, and can be installed in a narrow space.

Features

- Work Voltage: DC5V

- IC: One IC drives one RGB LED

- Consumption: 0.3w each LED

- Working Temperature: -15-50

- Color: Full color RGB

- RGB Type5050RGBBuilt-in IC WS2812B

- Light Strip Thickness: 2mm

• Each LED can be controlled individually

WS2812B Introdction

• WS2812B Datasheet

WS2812B is a intelligent control LED light source that the control circuit and RGB chip are integrated in a package of 5050 components. It internal include intelligent digital port data latch and signal reshaping ampli fication drive circuit. Also include a precision internal oscillator and a 12V voltage programmable constant curr e-nt control part, effectively ensuring the pixel point light color height consistent.

The data transfer protocol use single NZR communication mode. After the pixel power-on reset, the DIN port receive data from controller, the first pixel collect initial 24bit data then sent to the internal data latch, the other data which reshaping by the internal signal reshaping amplification circuit sent to the next cascade pixel through the DO port. After transmission for each pixelthe signal to reduce 24bit. pixel adopt auto resha -ping transmit technology, making the pixel cascade number is not limited the signal transmission, only depend on the speed of signal transmission.

LED with low driving voltage, environmental protection and energy saving, high brightness, scattering angl e is large, good consistency, low power, long life and other advantages. The control chip integrated in LED above becoming more simple circuit, small volume, convenient installation.

Example

• *2.18 RGB LED Strip* (For MicroPython User)

• *WS2812 Strip Multi-Mode* (For C/C++(Arduino) User)

• *Flow LEDs* (For C/C++(Arduino) User)

• :*LUMI Piano* (For C/C++(Arduino) User)

2.2.17 74HC595
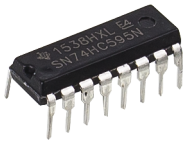
The 74HC595 consists of an 8bit shift register and a storage register with threestate parallel outputs. It converts serial input into parallel output so you can save IO ports of an MCU. When MR (pin10) is high level and OE (pin13) is low level, data is input in the rising edge of SHcp and goes to the memory register through the rising edge of SHcp. If the two clocks are connected together, the shift register is always one pulse earlier than the memory register. There is a serial shift input pin (Ds), a serial output pin (Q) and an asynchronous reset button (low level) in the memory register. The memory register outputs a Bus with a parallel 8-bit and in three states. When OE is enabled (low level), the data in memory register is output to the bus.

• 74HC595 Datasheet
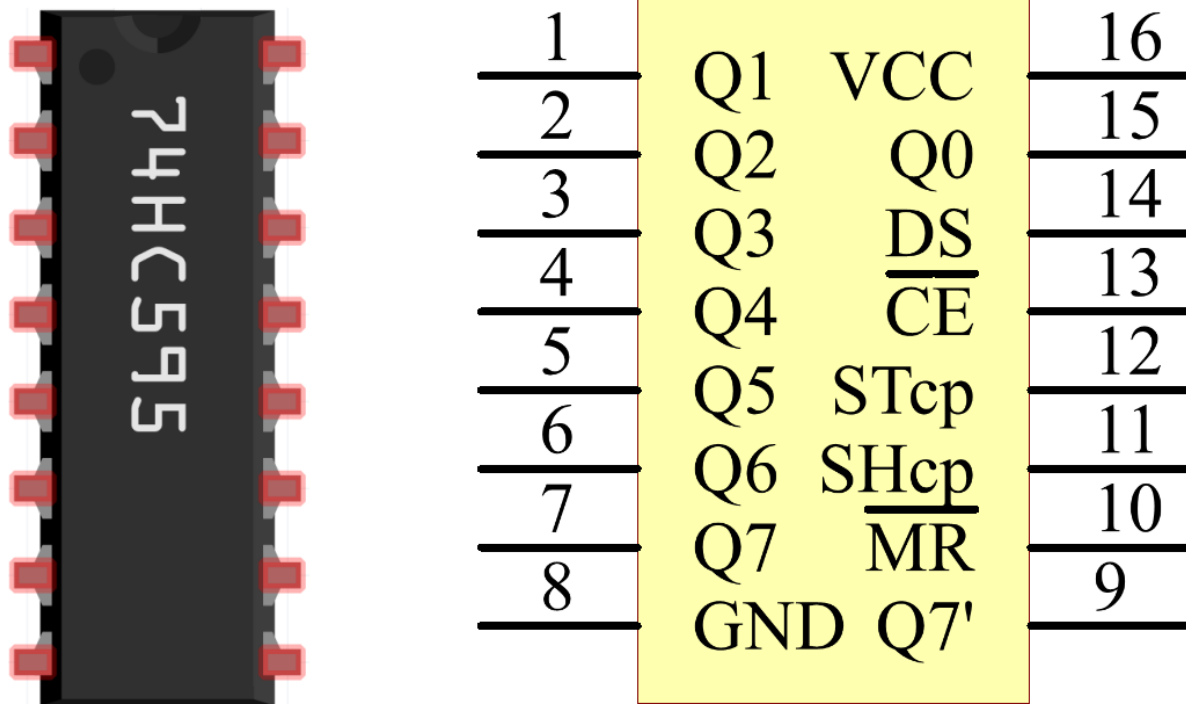
Pins of 74HC595 and their functions:

- **Q0-Q7**: 8-bit parallel data output pins, able to control 8 LEDs or 8 pins of 7-segment display directly.

- **Q7'**: Series output pin, connected to DS of another 74HC595 to connect multiple 74HC595s in series

- **MR**: Reset pin, active at low level;

- **SHcp**: Time sequence input of shift register. On the rising edge, the data in shift register moves successively one bit, i.e. data in Q1 moves to Q2, and so forth. While on the falling edge, the data in shift register remain unchanged.

- **STcp**: Time sequence input of storage register. On the rising edge, data in the shift register moves into memory register.

- **CE**: Output enable pin, active at low level.

- **DS**: Serial data input pin

- **VCC**: Positive supply voltage.

- **GND**: Ground.

Example

- *2.15 Microchip - 74HC595* (For MicroPython User

- *74HC595* (For C/C++(Arduino) User

2.2.18 7-segment Display

A 7-segment display is an 8-shaped component which packages 7 LEDs. Each LED is called a segment - when energized, one segment forms part of a numeral to be displayed.

There are two types of pin connection: Common Cathode (CC) and Common Anode (CA). As the name suggests, a CC display has all the cathodes of the 7 LEDs connected when a CA display has all the anodes of the 7 segments connected.

In this kit, we use the Common Cathode 7-segment display, here is the electronic symbol.
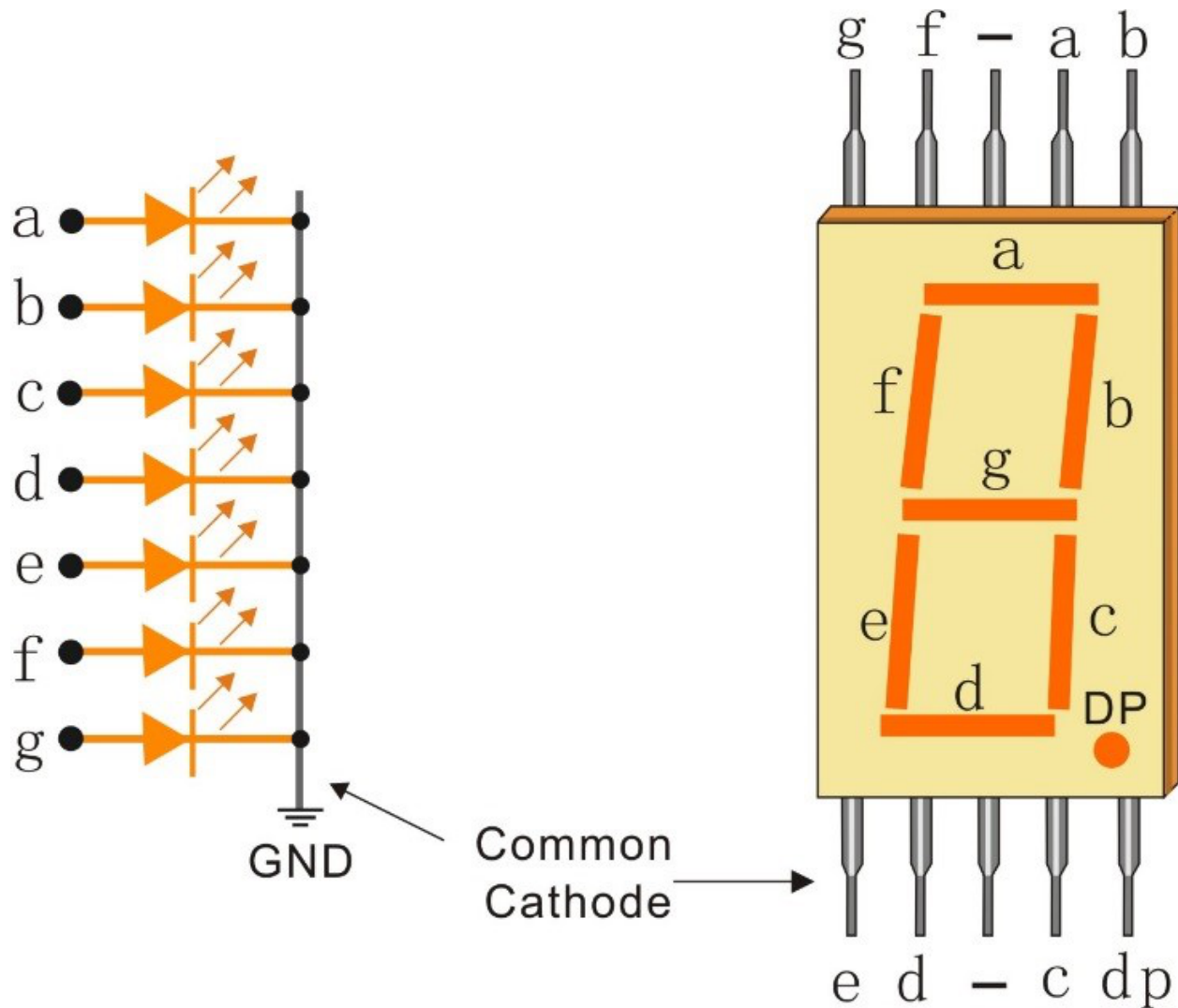
Each of the LEDs in the display is given a positional segment with one of its connection pins led out from the rectangular plastic package. These LED pins are labeled from "a" through to "g" representing each individual LED. The other LED pins are connected together forming a common pin. So by forward biasing the appropriate pins of the LED segments in a particular order, some segments will brighten and others stay dim, thus showing the corresponding character on the display.

Display Codes

To help you get to know how 7-segment displays(Common Cathode) display Numbers, we have drawn the following table. Numbers are the number 0-F displayed on the 7-segment display; (DP) GFEDCBA refers to the corresponding LED set to 0 or 1, For example, 00111111 means that DP and G are set to 0, while others are set to 1. Therefore, the number 0 is displayed on the 7-segment display, while HEX Code corresponds to hexadecimal number.

Numbers	Common Cathode		Numbers	Common Cathode	
	(DP)GFEDCBA	Hex Code		(DP)GFEDCBA	Hex Code
0	00111111	0x3f	A	01110111	0x77
1	00000110	0x06	B	01111100	0x7c
2	01011011	0x5b	C	00111001	0x39
3	01001111	0x4f	D	01011110	0x5e
4	01100110	0x66	E	01111001	0x79
5	01101101	0x6d	F	01110001	0x71
6	01111101	0x7d			
7	00000111	0x07			
8	01111111	0x7f			
9	01101111	0x6f			

Example

- *2.16 7-Segment Display* (For MicroPython User
- *Digital Dice* (For C/C++(Arduino) User
- *74HC595* (For C/C++(Arduino) User

2.2.19 I2C LCD1602
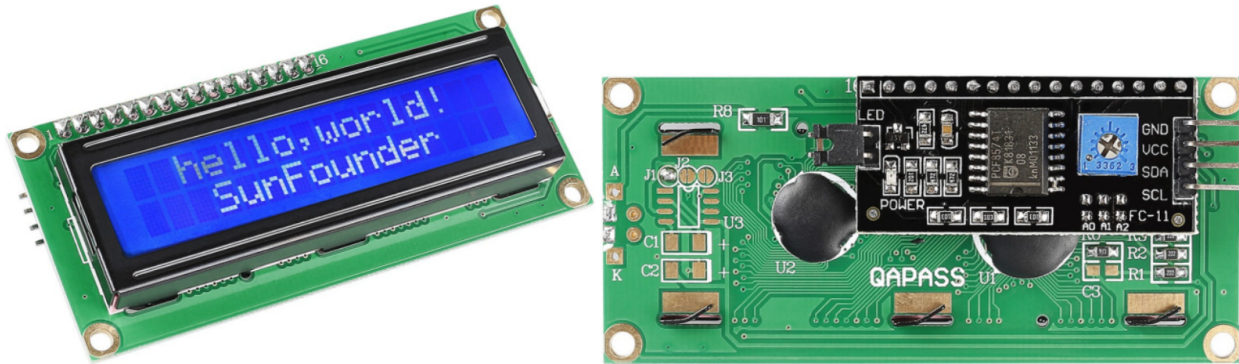
- **GND**: Ground

- **VCC**: Voltage supply, 5V.

- **SDA**: Serial data line. Connect to VCC through a pullup resistor.

- **SCL**: Serial clock line. Connect to VCC through a pullup resistor.

As we all know, though LCD and some other displays greatly enrich the man-machine interaction, they share a common weakness. When they are connected to a controller, multiple IOs will be occupied of the controller which has no so many outer ports. Also it restricts other functions of the controller.

Therefore, LCD1602 with an I2C module is developed to solve the problem. The I2C module has a built-in PCF8574 I2C chip that converts I2C serial data to parallel data for the LCD display.

- PCF8574 Datasheet

I2C Address

The default address is basically 0x27, in a few cases it may be 0x3F.

Taking the default address of 0x27 as an example, the device address can be modified by shorting the A0/A1/A2 pads; in the default state, A0/A1/A2 is 1, and if the pad is shorted, A0/A1/A2 is 0.

Slave Address

0	0	1	0	0	A2	A1	A0	
0	0	1	0	0	1	1	1	0x27
0	0	1	0	0	1	1	0	0x26
0	0	1	0	0	1	0	1	0x25
0	0	1	0	0	0	1	1	0x23
							......	
0	0	1	0	0	0	0	0	0x20

Backlight/Contrast

Backlight can be enabled by jumper cap, unplugg the jumper cap to disable the backlight. The blue potentiometer on the back is used to adjust the contrast (the ratio of brightness between the brightest white and the darkest black).
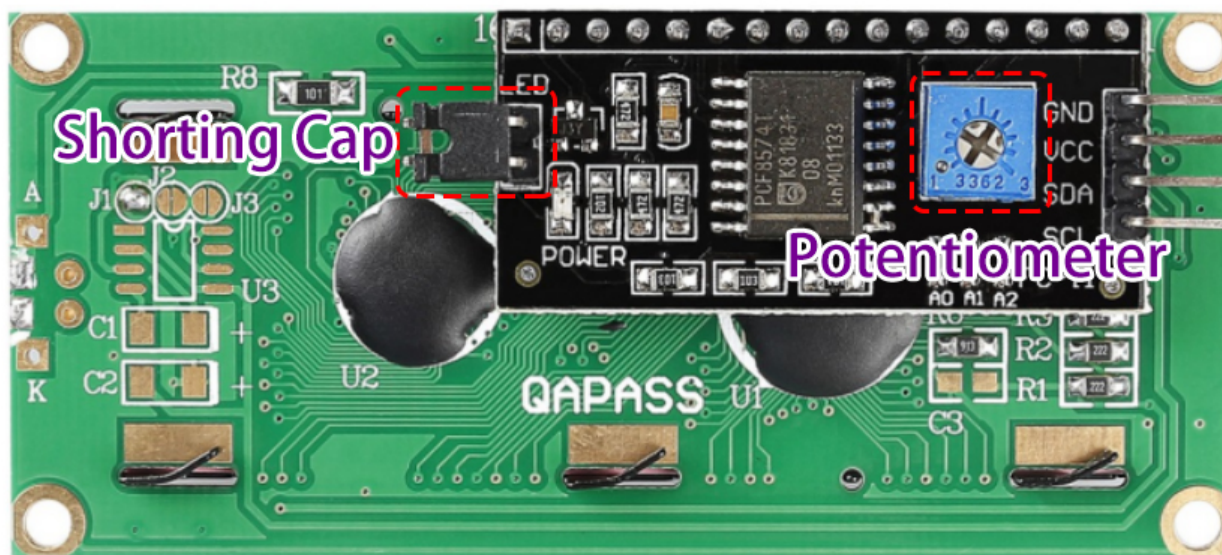
- **Shorting Cap**: Backlight can be enabled by this capunplugg this cap to disable the backlight.

- **Potentiometer**: It is used to adjust the contrast (the clarity of the displayed text), which is increased in the clockwise direction and decreased in the counterclockwise direction.

Example

- *2.17 Show Characters on I2 LCD1602* (For MicroPython User

FOR MICROPYTHON USER

This section contains the history of MicroPython, installing MicroPython in Pico, the basic syntax of MicroPython and a dozen interesting and practical projects to help you learn MicroPython quickly.

We recommend that you read the chapters in order.

1. Get Started

3.1 1.1 Introduction of MicroPython

MicroPython is a software implementation of a programming language largely compatible with Python 3, written in C, that is optimized to run on a microcontroller.[3][4]

MicroPython consists of a Python compiler to bytecode and a runtime interpreter of that bytecode. The user is presented with an interactive prompt (the REPL) to execute supported commands immediately. Included are a selection of core Python libraries; MicroPython includes modules which give the programmer access to low-level hardware.

- Reference: MicroPython - Wikipedia

3.1.1 The Story Starts Here

Things changed in 2013 when Damien George launched a crowdfunding campaign (Kickstarter).

Damien was an undergraduate student at Cambridge University and an avid robotics programmer. He wanted to reduce the world of Python from a gigabyte machine to a kilobyte. His Kickstarter campaign was to support his development while he turned his proof of concept into a finished implementation.

MicroPython is supported by a diverse Pythonista community that has a keen interest in seeing the project succeed.

Apart from testing and supporting the code base, the developers provided tutorials, code libraries, and hardware porting, so Damien was able to focus on other aspects of the project.

- Reference: realpython

3.1.2 Why MicroPython

Although the original Kickstarter campaign released MicroPython as a development board "pyboard" with STM32F4, MicroPython supports many ARM-based product architectures. The mainline supported ports are ARM Cortex-M (many STM32 boards, TI CC3200/WiPy, Teensy boards, Nordic nRF series, SAMD21 and SAMD51), ESP8266, ESP32, 16bit PIC, Unix, Windows, Zephyr and JavaScript. Second, MicroPython allows for fast feedback. This is because you can use REPL to enter commands interactively and get responses. You can even tweak code and run it immediately instead of traversing the code-compile-upload-execute cycle.

While Python has the same advantages, for some Microcontroller boards like the Raspberry Pi Pico, they are small, simple and have little memory to run the Python language at all. That's why MicroPython has evolved, keeping the main Python features and adding a bunch of new ones to work with these Microcontroller boards.

Next you will learn to install MicroPython into the Raspberry Pi Pico.

- Reference: MicroPython - Wikipedia
- Reference: realpython

3.2 1.2 Install Thonny IDE

Before you can start to program Pico with MicroPython, you need an integrated development environment (IDE), here we recommend Thonny. Thonny comes with Python 3.7 built in, just one simple installer is needed and you're ready to learn programming.

Note: Since the Raspberry Pi Pico interpreter only works with Thonny version 3.3.3 or later, you can skip this chapter if you have it; otherwise, please update or install it.

---

1. You can download it by visiting the website. Once open the page, you will see a light gray box in the upper right corner, click on the link that applies to your operating system.
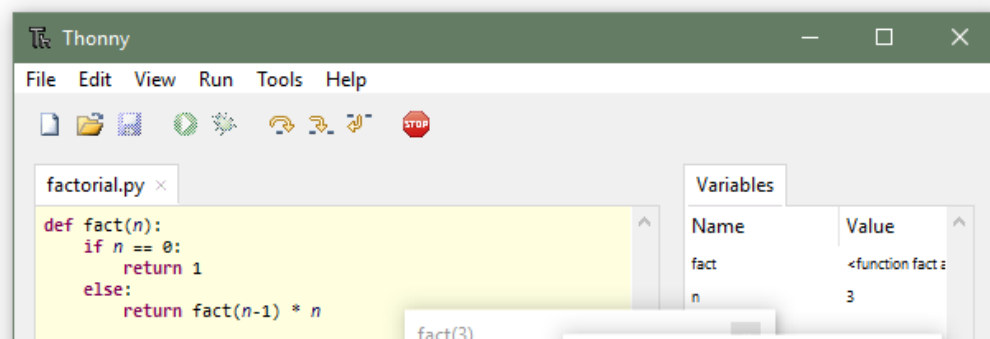
2. The installers have been signed with a new certificate which hasn't built up its reputation yet. You may need to click through your browser warning (e.g. choose "Keep" instead of "Discard" in Chrome) and Windows Defender warning (**More info Run anyway**).
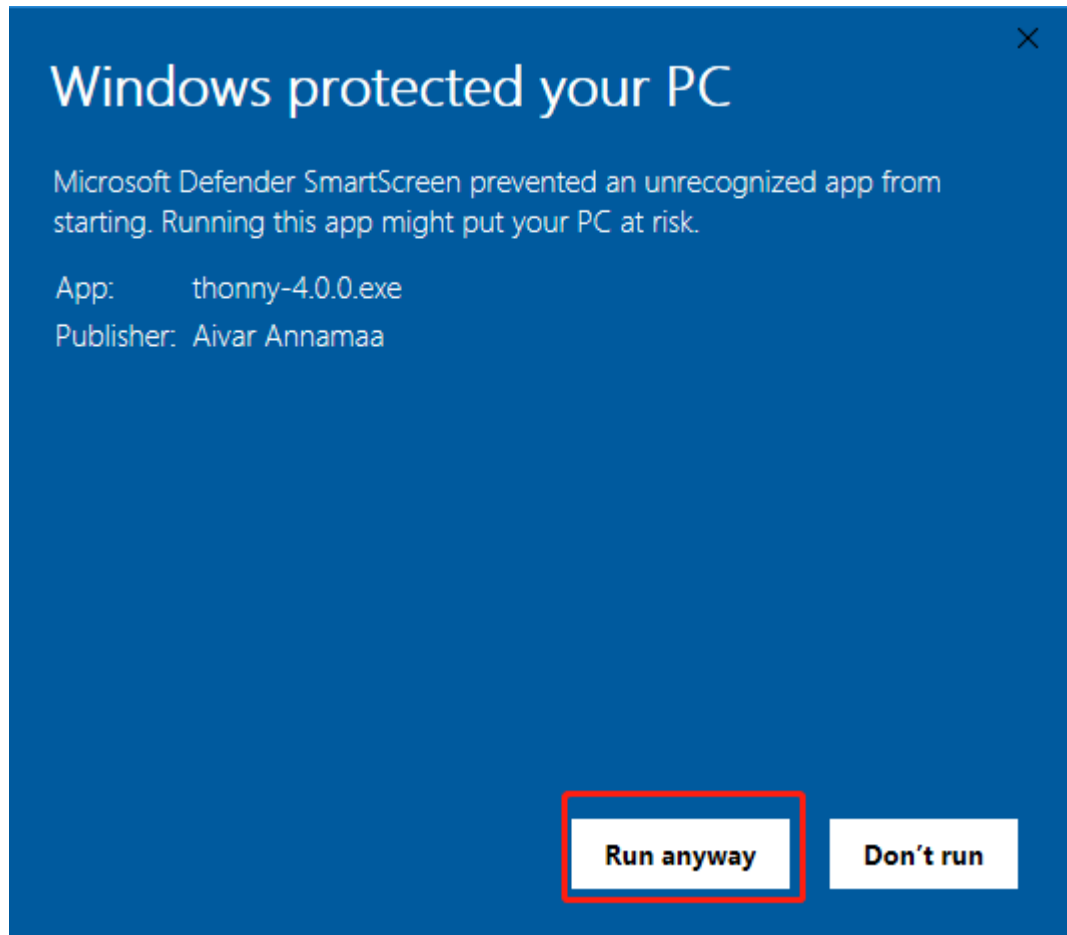
3. Next, click **Next** and **Install** to finish installing Thonny.
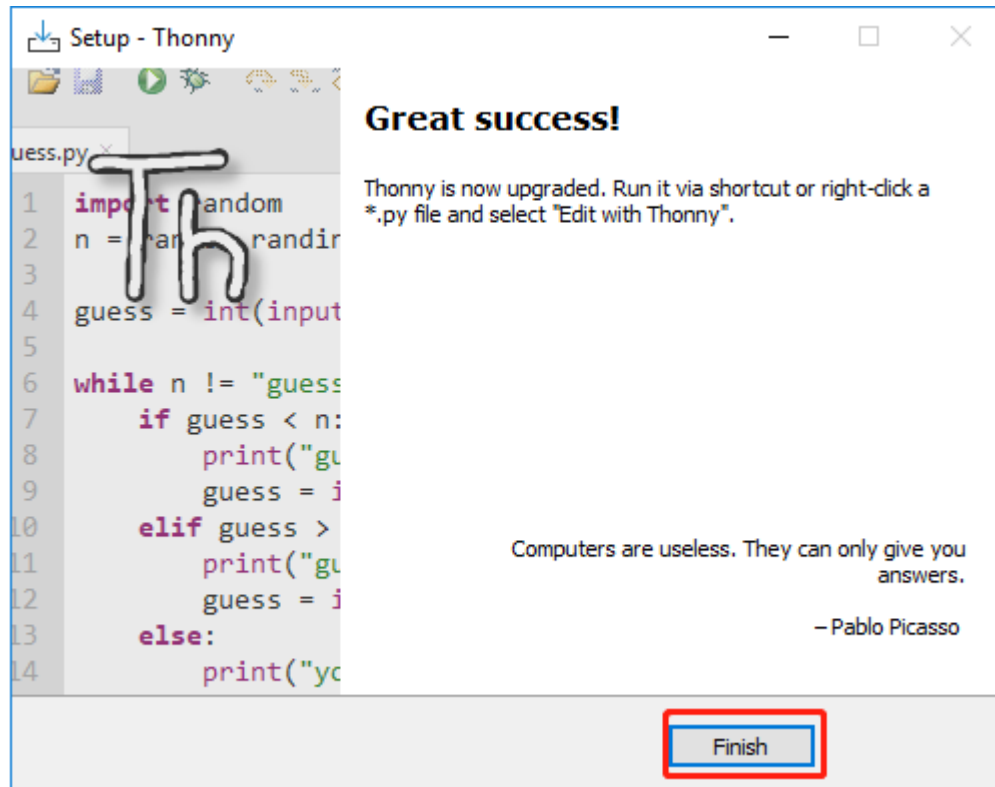
3.3 1.3 Install MicroPython on Your Pico

Now come to install MicroPython into Raspberry Pi Pico, Thonny IDE provides a very convenient way for you to install it with one click.

Note: If you do not wish to upgrade Thonny, you can use the Raspberry Pi official by dragging and dropping an `rp2_pico_xxxx.uf2` file into Raspberry Pi Pico.

---

1. Open Thonny IDE.
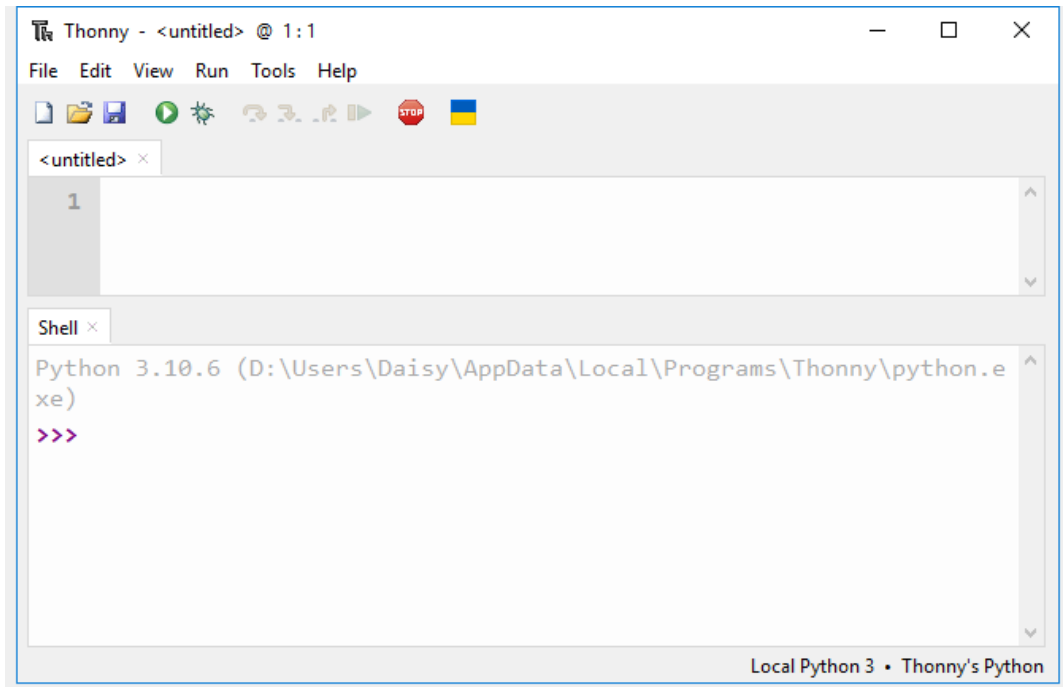
2. Press and hold the **BOOTSEL** button and then connect the Pico to computer via a Micro USB cable. Release the **BOOTSEL** button after your Pico is mount as a Mass Storage Device called **RPI-RP2**.
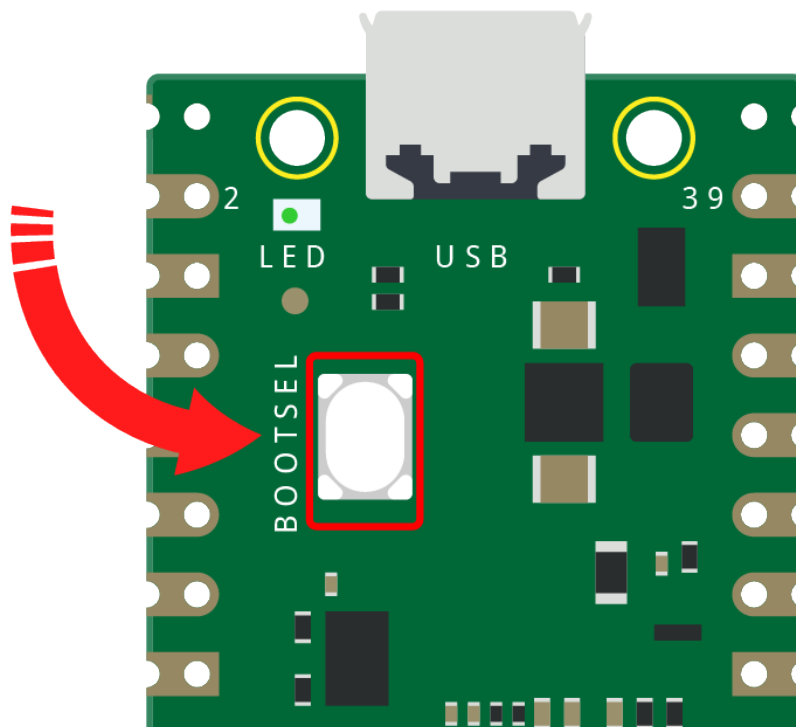
3. In the bottom right corner, click the interpreter selection button and select **Install Micropython**.

Note: If your Thonny does not have this option, please update to the latest version.

---
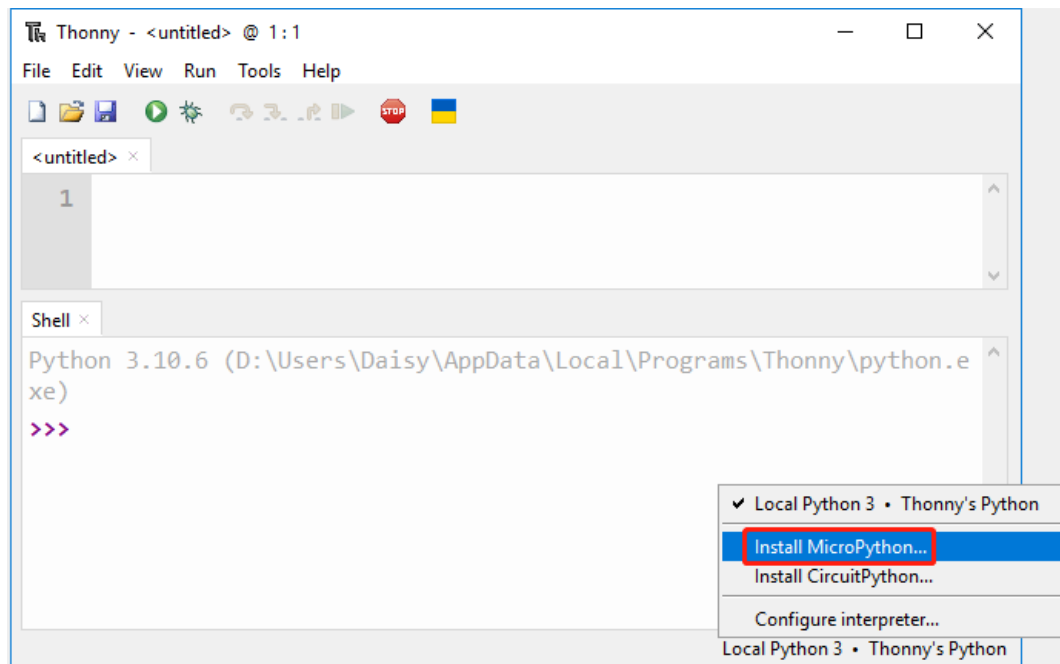
4. In the **Target volume**, the volume of the Pico you just plugged in will automatically appear, and in the **Micropython variant**, select **Raspberry Pi.Pico/Pico H**.
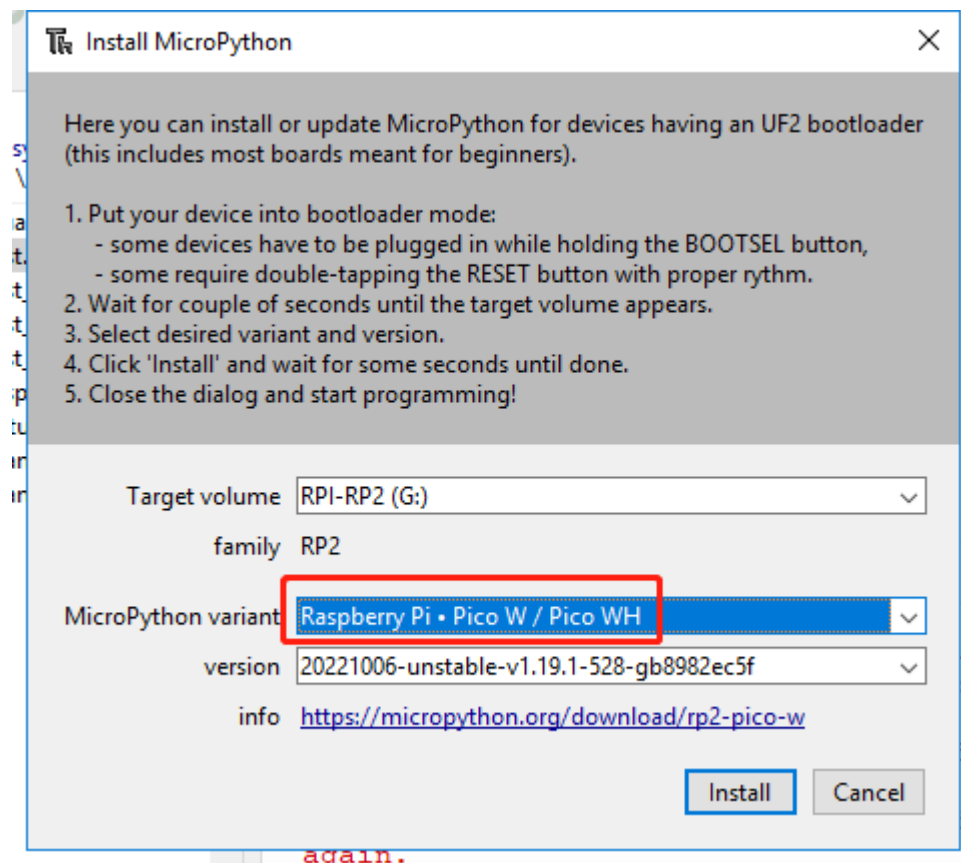
5. Click the **Install** button, wait for the installation to complete and then close this page.
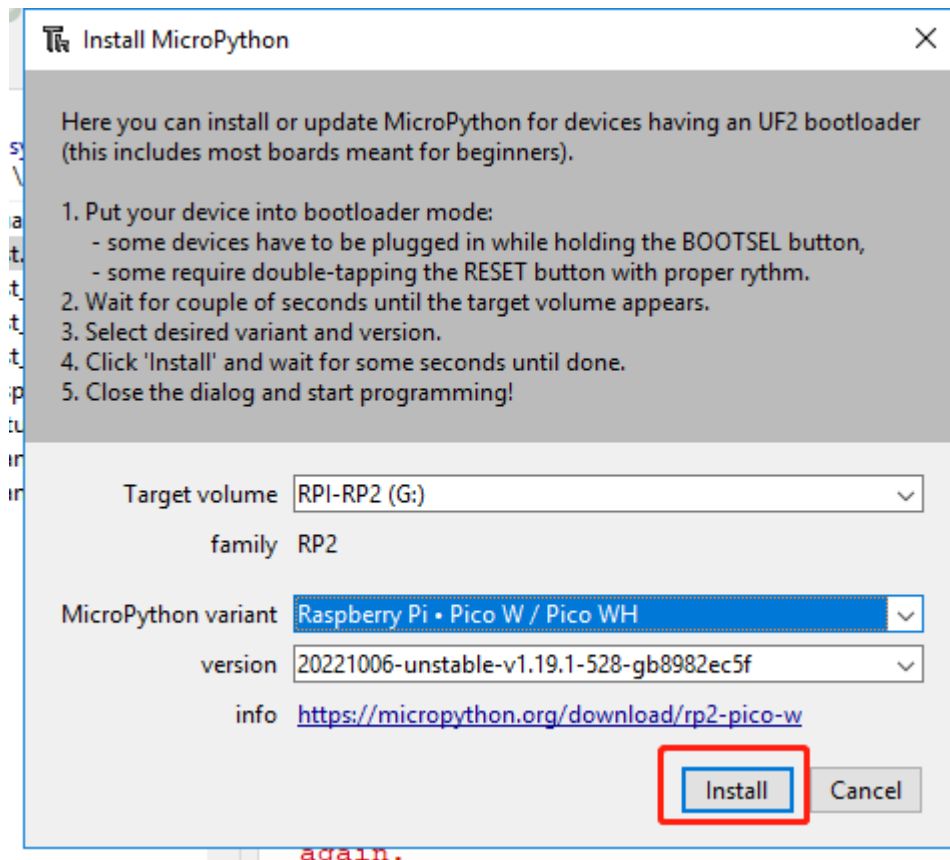
Congratulations, now your Raspberry Pi Pico is ready to go.

3.4 1.4 Your First MicroPython Program

In MicroPython, there are two options/methods for running code:

- *Interactive Mode*
- *Script Mode*

Interactive mode involves running your code directly from the Python shell, which can be accessed from the operating system's terminal. In script mode, you must create a file, save it with .py extension, and then run your code. Interactive mode is suitable when you are running a few lines of code. Script mode is recommended when you need to create large codes and save them for next time.

3.4.1 Interactive Mode

Interactive mode, also known as the REPL provides us with a quick way to run blocks or a single line of MicroPython code via the Python shell.

Click the Python Shell area, type the following command after >>>, and then Enter.

```python
print("Hello, World!")
```
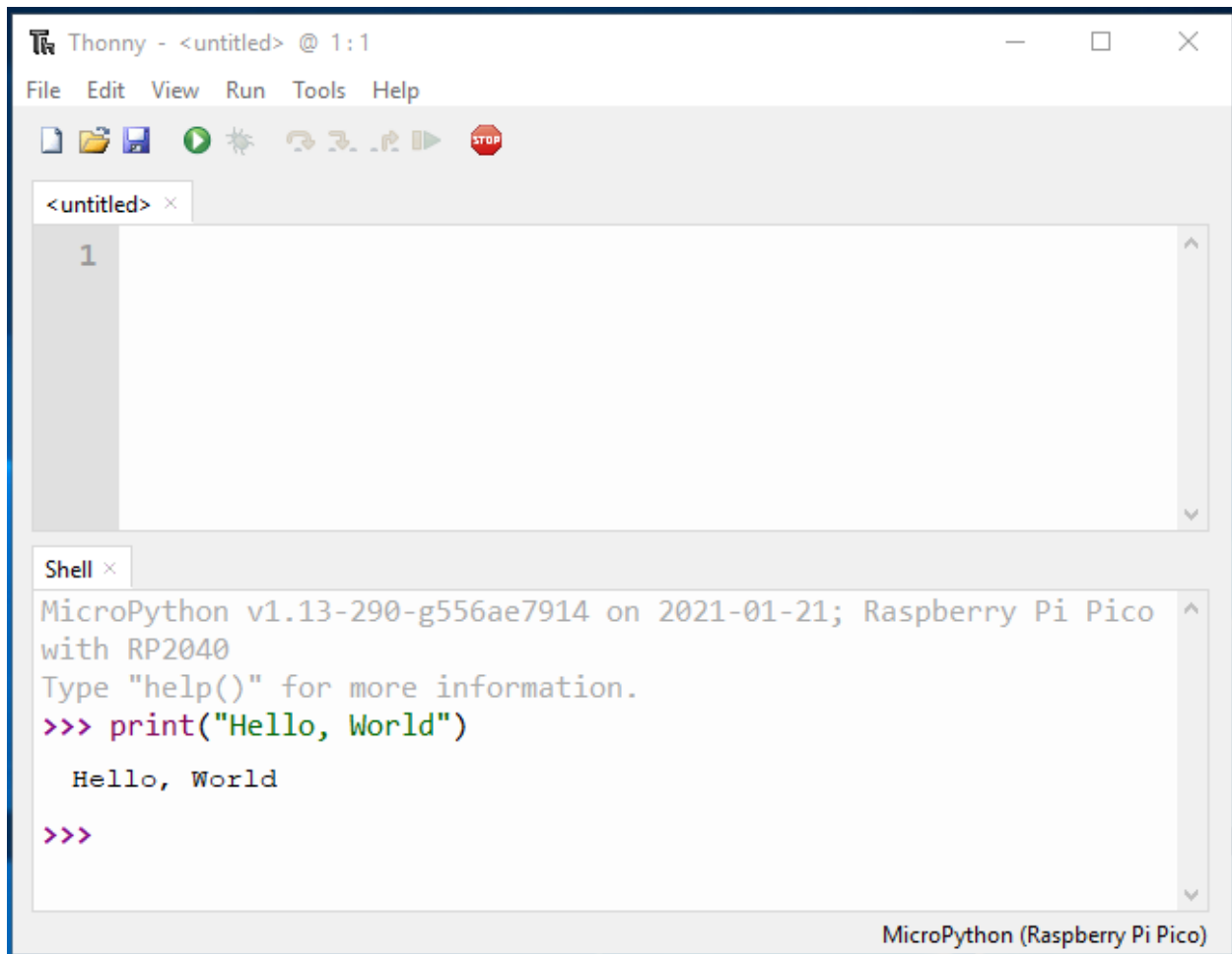
Note: If your program does not run, but prints a "syntax error" message to the Shell area, then there is an error in what you have written.

MicroPython needs to write its instructions in a very specific way: miss parentheses or quotation marks, spell `print` errors, or give it a capital P, or add extra symbols somewhere in the instruction, and it won't run . Try to type the command again and make sure it is the same as this one, and then press Enter.

You will find that the message `hello world` will be printed out immediately in the Shell area.

3.4.2 Script Mode

Interactive mode is not recommended if you need to write a long piece of Python code, or if your Python script spans multiple files. In this case, script mode can be used. In script mode, you write your code and save it with a .py extension, which stands for "Python".

Enter the same command `print("Hello, World!")` in the script area, when you press the ENTER key, the program will not run, only one more blank line in the script area.
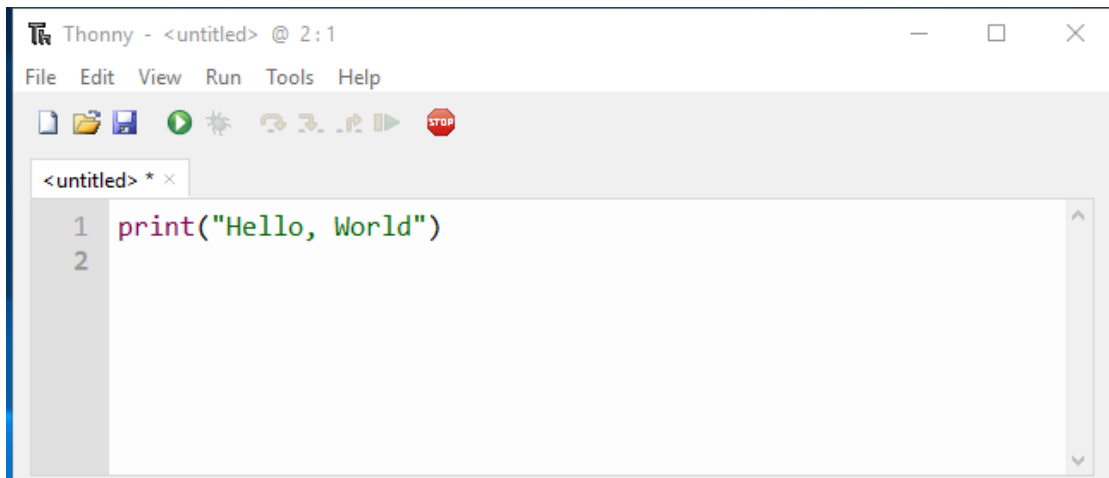
You need click **Run Current Script** or simply press F5 to run it. If your code has not been saved, a window will pop up asking to save to This computer or Raspberry Pi Pico (if your Pico is already plugged into the computer)? Also you will see MicroPython (Raspberry Pi Pico) displayed in the bottom right corner, this is the interpreter you need to select, if yours is not this one you can click on it to switch.
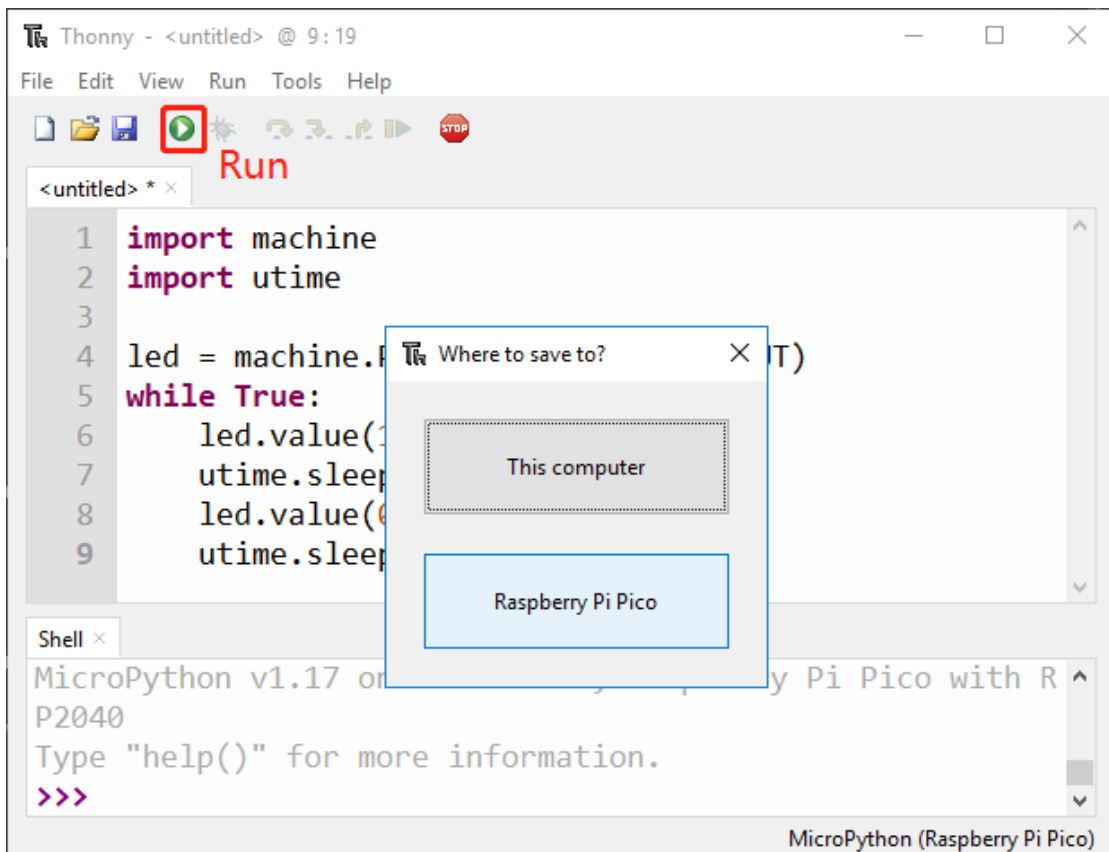
Note: When you tell Thonny to save your program on the Raspberry Pi Pico, if you unplug the Pico and plug it into someone else's computer, your program is still saved on the Pico.

Choose the location you want to save, then enter the file name `hello_world` and the extension `.py`, and then click OK.
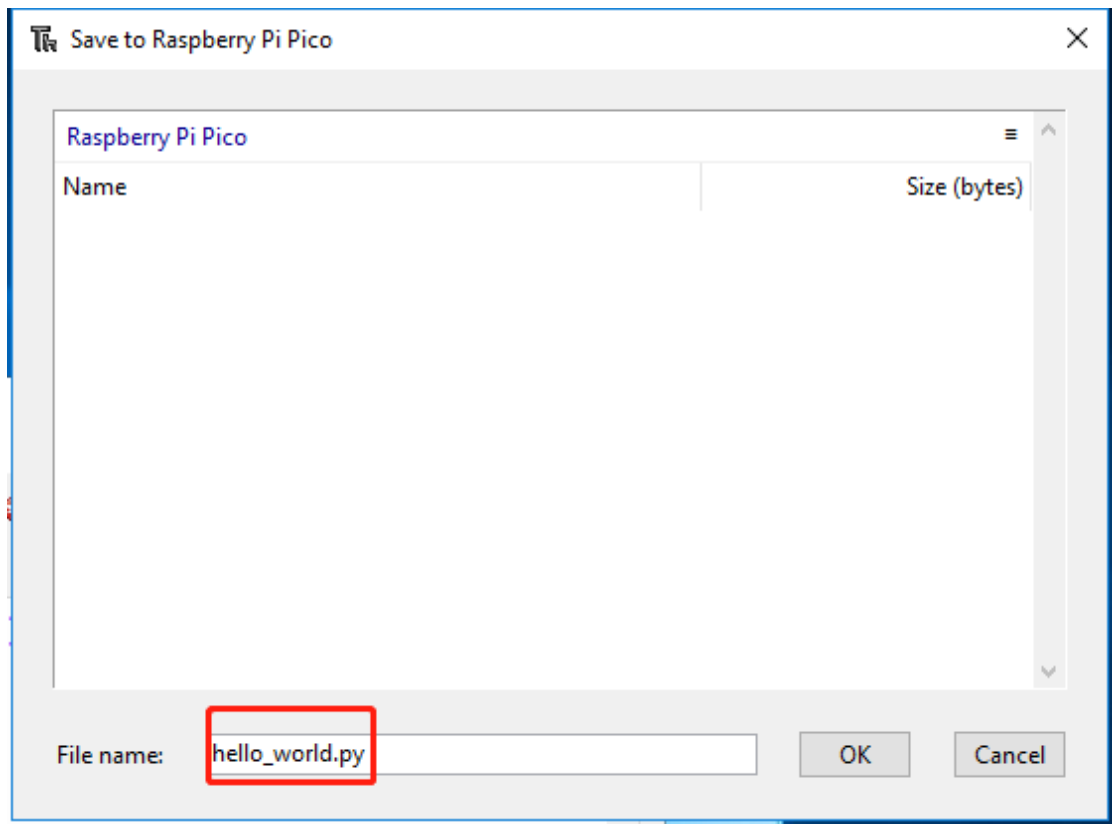
Note: You can save your code under any name, but it's best to describe what kind of code it is, and don't name it with meaningless names such as `abc.py`. It is important to note that if you save the code file name as `main.py`, it will run automatically when the power is turned on.

When your program is saved, it will run automatically and you will see 2 lines of information in the Shell area.

```
>>> %Run -c $EDITOR_CONTENT
Hello, World!
```

The first of these lines is an instruction from Thonny telling the MicroPython interpreter on your Pico to run the contents of the script area – the 'EDITOR_CONTENT'. The second is the output of the program – the message you told MicroPython to print.

In Script mode, it is very easy for you to open the previously saved code again, but the code entered in Interactive Mode will not be saved and can only be re-entered.

Click the open icon in the Thonny toolbar, just like when you save the program, you will be asked whether you want to save it to **This Computer** or **Raspberry Pi Pico**, for example, click **Raspberry Pi Pico** and you will see a list of all programs you save to your Pico.

Find hello_world.py in the list, if your Pico is new, it will be the only file there, and click it to select it, then click OK. Your program will be loaded into Thonny, ready to edit or run it again for you.

3.5 1.5 (Optional) MicroPython Basic Syntax

3.5.1 Indentation

Indentation refers to the spaces at the beginning of a code line. Like standard Python programs, MicroPython programs usually run from top to bottom: It traverses each line in turn, runs it in the interpreter, and then continues to the next line, Just like you type them line by line in the Shell. A program that just browses the instruction list line by line is not very smart, though – so MicroPython, just like Python, has its own method to control the sequence of its program execution: indentation.

You must put at least one space before print(), otherwise an error message "Invalid syntax" will appear. It is usually recommended to standardise spaces by pressing the Tab key uniformly.

```python
if 8 > 5:
print("Eight is greater than Five!")
```

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 2
SyntaxError: invalid syntax
```

You must use the same number of spaces in the same block of code, or Python will give you an error.

```python
if 8 > 5:
print("Eight is greater than Five!")
        print("Eight is greater than Five")
```

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 2
SyntaxError: invalid syntax
```

3.5.2 Comments

The comments in the code help us understand the code, make the entire code more readable and comment out part of the code during testing, so that this part of the code does not run.

Single-line Comment

Single-line comments in MicroPython begin with #, and the following text is considered a comment until the end of the line. Comments can be placed before or after the code.

```python
print("hello world") #This is a annotationhello world
```

```
>>> %Run -c $EDITOR_CONTENT
hello world
```

Comments are not necessarily text used to explain the code. You can also comment out part of the code to prevent micropython from running the code.

```python
#print("Can't run it")
print("hello world") #This is a annotationhello world
```

```
>>> %Run -c $EDITOR_CONTENT
hello world
```

Multi-line comment

If you want to comment on multiple lines, you can use multiple # signs.

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

```
>>> %Run -c $EDITOR_CONTENT
Hello, World!
```

Or, you can use multi-line strings instead of expected.

Since MicroPython ignores string literals that are not assigned to variables, you can add multiple lines of strings (triple quotes) to the code and put comments in them:

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

```
>>> %Run -c $EDITOR_CONTENT
Hello, World!
```

As long as the string is not assigned to a variable, MicroPython will ignore it after reading the code and treat it as if you made a multi-line comment.

3.5.3 Print()

The `print()` function prints the specified message to the screen, or other standard output device. The message can be a string, or any other object, the object will be converted into a string before written to the screen.

Print multiple objects:

```
print("Welcome!", "Enjoy yourself!")
```

```
>>> %Run -c $EDITOR_CONTENT
Welcome! Enjoy yourself!
```

Print tuples:

```
x = ("pear", "apple", "grape")
print(x)
```

```
>>> %Run -c $EDITOR_CONTENT
('pear', 'apple', 'grape')
```

Print two messages and specify the separator:

```
print("Hello", "how are you?", sep="---")
```

```
>>> %Run -c $EDITOR_CONTENT
Hello---how are you?
```

3.5.4 Variables

Variables are containers used to store data values.

Creating a variable is very simple. You only need to name it and assign it a value. You don't need to specify the data type of the variable when assigning it, because the variable is a reference, and it accesses objects of different data types through assignment.

Naming variables must follow the following rules:

- Variable names can only contain numbers, letters, and underscores
- The first character of the variable name must be a letter or underscore
- Variable names are case sensitive

Create Variable

There is no command for declaring variables in MicroPython. Variables are created when you assign a value to it for the first time. It does not need to use any specific type declaration, and you can even change the type after setting the variable.

```
x = 8        # x is of type int
x = "lily" # x is now of type str
print(x)
```

```
>>> %Run -c $EDITOR_CONTENT
lily
```

Casting

If you want to specify the data type for the variable, you can do it by casting.

```
x = int(5)    # y will be 5
y = str(5)    # x will be '5'
z = float(5)  # z will be 5.0
print(x,y,z)
```

```
>>> %Run -c $EDITOR_CONTENT
5 5 5.0
```

Get the Type

You can get the data type of a variable with the *type()* function.

```
x = 5
y = "hello"
z = 5.0
print(type(x),type(y),type(z))
```

```
>>> %Run -c $EDITOR_CONTENT
<class 'int'> <class 'str'> <class 'float'>
```

Single or Double Quotes?

In MicroPython, single quotes or double quotes can be used to define string variables.

```
x = "hello"
# is the same as
x = 'hello'
```

Case-Sensitive

Variable names are case-sensitive.

```
a = 5
A = "lily"
#A will not overwrite a
print(a, A)
```

```
>>> %Run -c $EDITOR_CONTENT
5 lily
```

3.5.5 If Else

Decision making is required when we want to execute a code only if a certain condition is satisfied.

if

```
if test expression:
    statement(s)
```

Here, the program evaluates the *test expression* and executes the *statement* only when the *test expression* is True.

If *test expression* is False, then *statement(s)* will not be executed.

In MicroPython, indentation means the body of the *if* statement. The body starts with an indentation and ends with the first unindented line.

Python interprets non-zero values as "True". None and 0 are interpreted as "False".

if Statement Flowchart
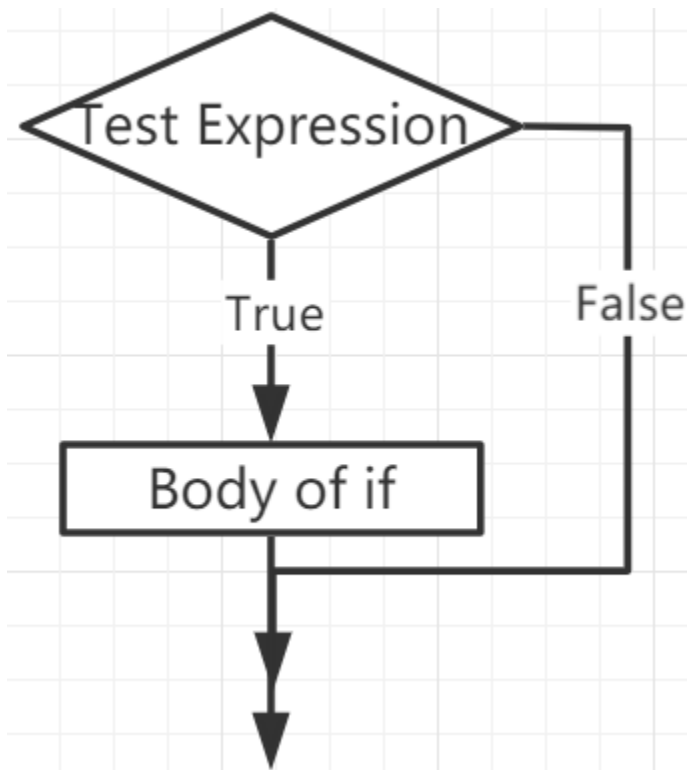
Example

```
num = 8
if num > 0:
    print(num, "is a positive number.")
print("End with this line")
```

```
>>> %Run -c $EDITOR_CONTENT
8 is a positive number.
End with this line
```

if... else

```
if test expression:
    Body of if
else:
    Body of else
```

The *if..else* statement evaluates *test expression* and will execute the body of *if* only when the test condition is *True*.

If the condition is *False*, the body of *else* is executed. Indentation is used to separate the blocks.

if... else Statement Flowchart
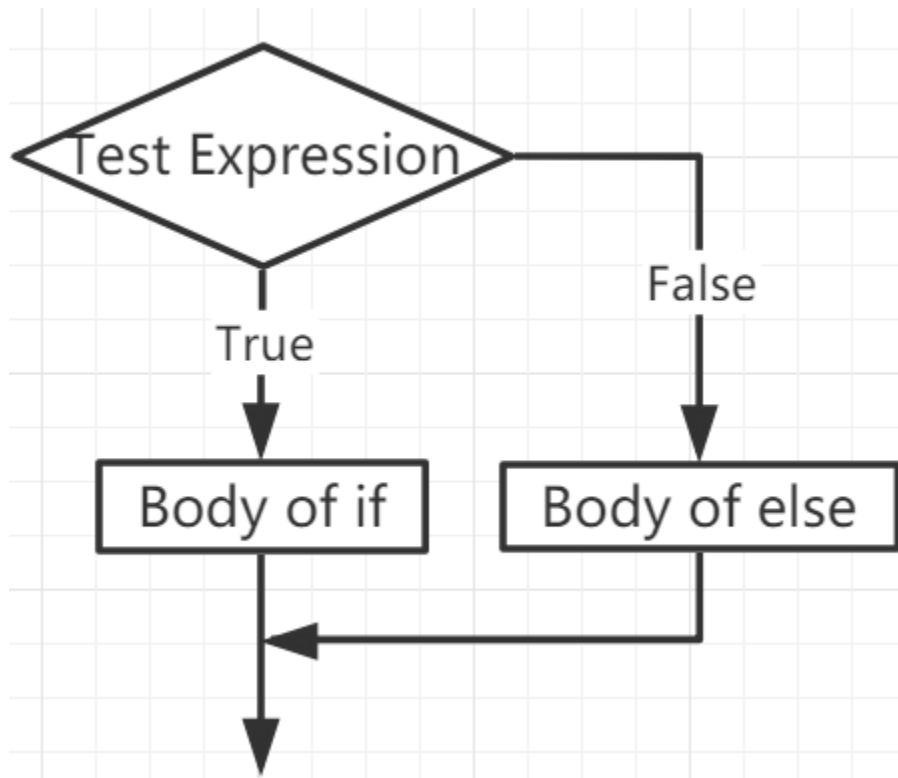
Example

```
num = -8
if num > 0:
    print(num, "is a positive number.")
else:
    print(num, "is a negative number.")
```

```
>>> %Run -c $EDITOR_CONTENT
-8 is a negative number.
```

if... elif... else

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

Elif is short for *else if*. It allows us to check multiple expressions.

If the condition of the *if* is False, the condition of the next elif block is checked, and so on.

If all conditions are *False*, the body of *else* is executed.

Only one of several *if... elif... else* blocks is executed according to the conditions.

The *if* block can only have one *else* block. But it can have multiple *elif* blocks.

if... elif... else Statement Flowchart
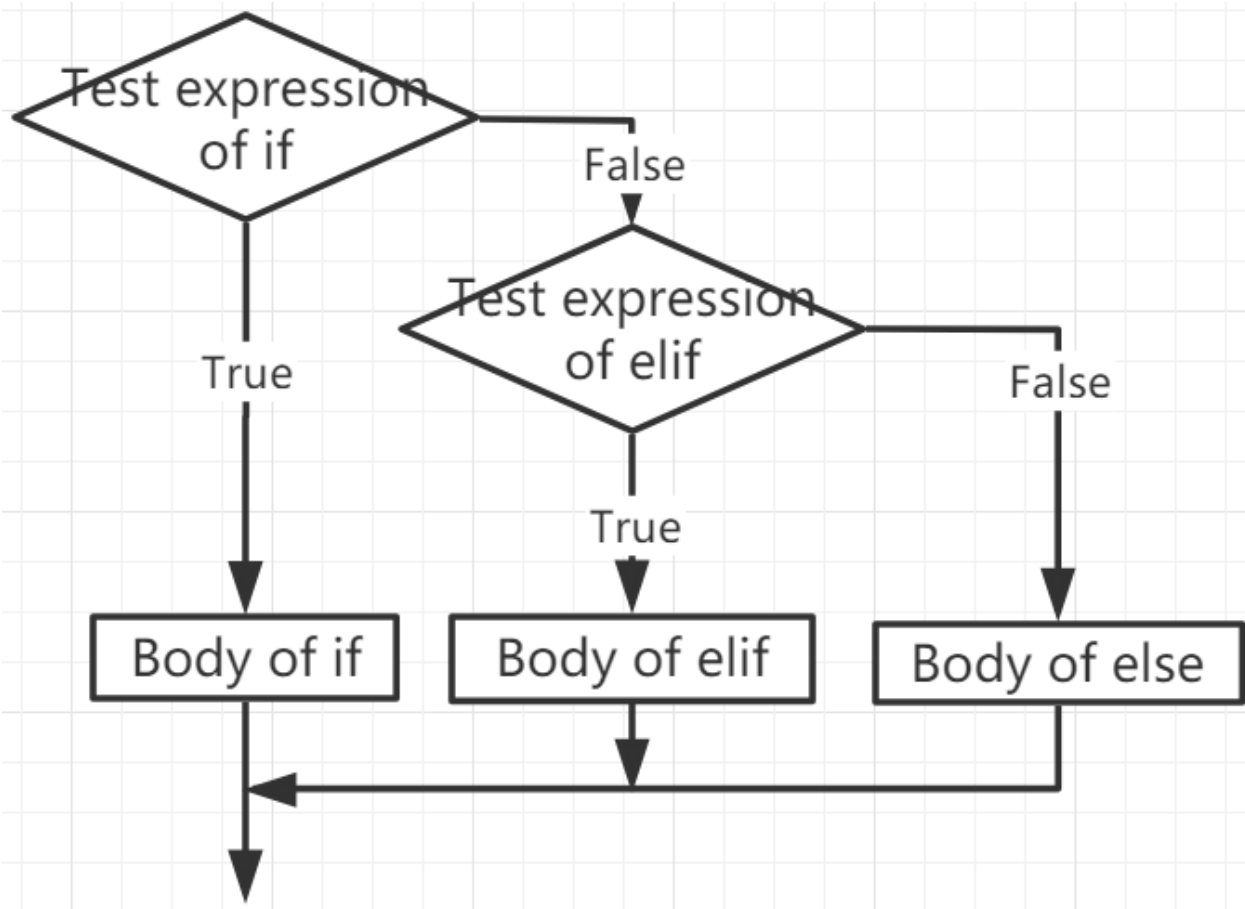
Example

```
x = 10
y = 9

if x > y:
    print("x is greater than y")
elif x == y:
    print("x and y are equal")
else:
    print("x is greater than y")
```

```
>>> %Run -c $EDITOR_CONTENT
x is greater than y
```

Nested if

We can embed an if statement into another if statement, and then call it a nested if statement.

Example

```
x = 67

if x > 10:
    print("Above ten,")
    if x > 20:
        print("and also above 20!")
    else:
        print("but not above 20.")
```

```
>>> %Run -c $EDITOR_CONTENT
Above ten,
and also above 20!
```

3.5.6 While Loops

The `while` statement is used to execute a program in a loop, that is, to execute a program in a loop under certain conditions to handle the same task that needs to be processed repeatedly.

Its basic form is:

```
while test expression:
    Body of while
```

In the `while` loop, first check the `test expression`. Only when `test expression` evaluates to `True`, enter the body of the while. After one iteration, check the `test expression` again. This process continues until `test expression` evaluates to `False`.

In MicroPython, the body of the `while` loop is determined by indentation.

The body starts with an indentation and ends with the first unindented line.

Python interprets any non-zero value as `True`. None and 0 are interpreted as `False`.

while Loop Flowchart
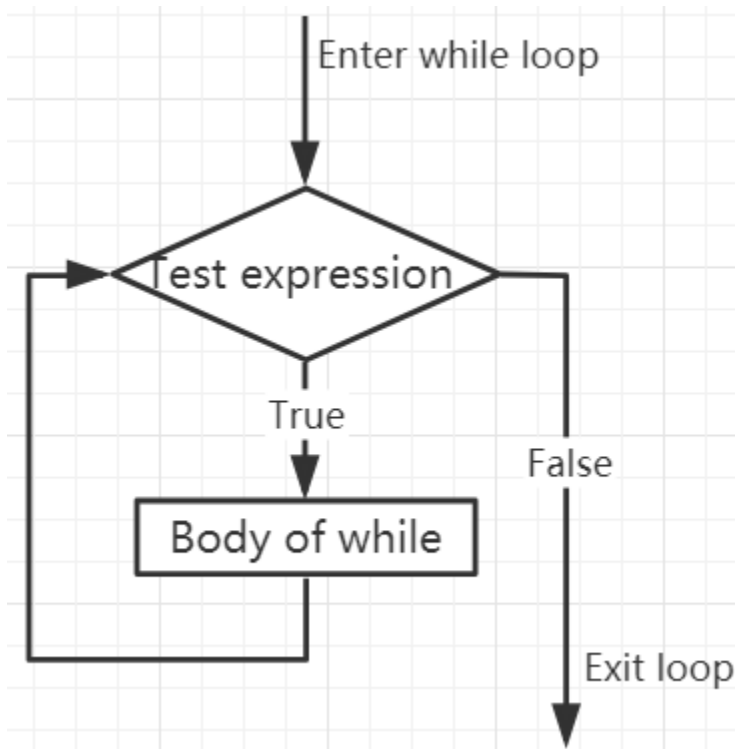
```
x = 10

while x > 0:
    print(x)
    x -= 1
```

```
>>> %Run -c $EDITOR_CONTENT
10
9
8
7
6
5
4
3
2
1
```

Break Statement

With the break statement we can stop the loop even if the while condition is true:

```
x = 10

while x > 0:
    print(x)
    if x == 6:
        break
    x -= 1
```

```
>>> %Run -c $EDITOR_CONTENT
10
9
8
7
6
```

While Loop with Else

Like the *if* loop, the *while* loop can also have an optional *else* block.

If the condition in the *while* loop is evaluated as *False*, the *else* part is executed.

```python
x = 10

while x > 0:
    print(x)
    x -= 1
else:
    print("Game Over")
```

```
>>> %Run -c $EDITOR_CONTENT
10
9
8
7
6
5
4
3
2
1
Game Over
```

3.5.7 For Loops

The *for* loop can traverse any sequence of items, such as a list or a string.

The syntax format of for loop is as follows:

```python
for val in sequence:
    Body of for
```

Here, *val* is a variable that gets the value of the item in the sequence in each iteration.

The loop continues until we reach the last item in the sequence. Use indentation to separate the body of the *for* loop from the rest of the code.

Flowchart of for Loop
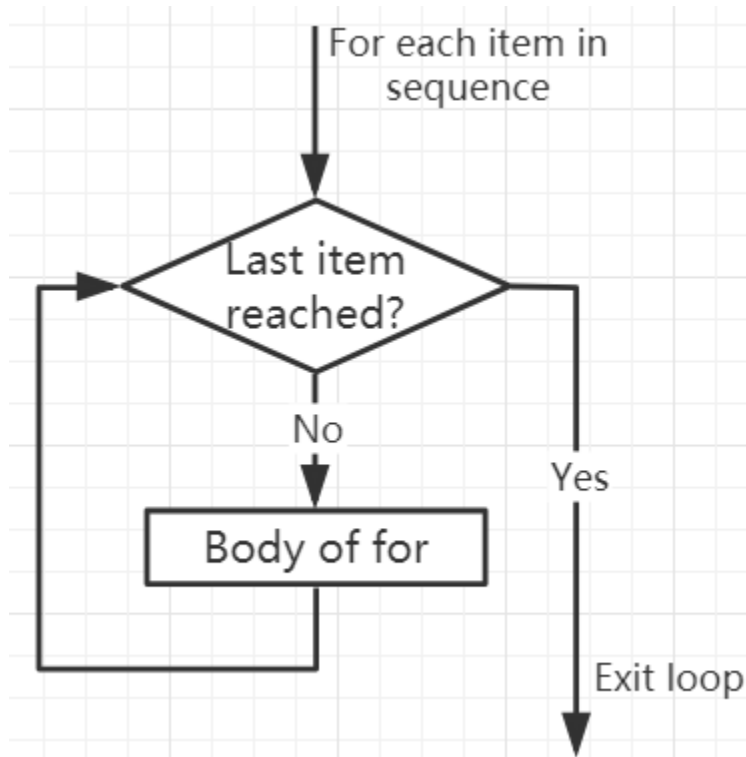
```
numbers = [1, 2, 3, 4]
sum = 0

for val in numbers:
    sum = sum+val

print("The sum is", sum)
```

```
>>> %Run -c $EDITOR_CONTENT
The sum is 10
```

The break Statement

With the break statement we can stop the loop before it has looped through all the items:

```
numbers = [1, 2, 3, 4]
sum = 0

for val in numbers:
    sum = sum+val
    if sum == 6:
        break
print("The sum is", sum)
```

```
>>> %Run -c $EDITOR_CONTENT
The sum is 6
```

The continue Statement

With the *continue* statement we can stop the current iteration of the loop, and continue with the next:

```python
numbers = [1, 2, 3, 4]

for val in numbers:
    if val == 3:
        continue
    print(val)
```

```
>>> %Run -c $EDITOR_CONTENT
1
2
4
```

The range() function

We can use the range() function to generate a sequence of numbers. range(6) will produce numbers between 0 and 5 (6 numbers).

We can also define start, stop and step size as range(start, stop, step_size). If not provided, step_size defaults to 1.

In a sense of range, the object is "lazy" because when we create the object, it does not generate every number it "contains". However, this is not an iterator because it supports in, len and __getitem__ operations.

This function will not store all values in memory; it will be inefficient. So it will remember the start, stop, step size and generate the next number during the journey.

To force this function to output all items, we can use the function list().

```python
print(range(6))

print(list(range(6)))

print(list(range(2, 6)))

print(list(range(2, 10, 2)))
```

```
>>> %Run -c $EDITOR_CONTENT
range(0, 6)
[0, 1, 2, 3, 4, 5]
[2, 3, 4, 5]
[2, 4, 6, 8]
```

We can use *range()* in a *for* loop to iterate over a sequence of numbers. It can be combined with the len() function to use the index to traverse the sequence.

```python
fruits = ['pear', 'apple', 'grape']

for i in range(len(fruits)):
    print("I like", fruits[i])
```

```
>>> %Run -c $EDITOR_CONTENT
I like pear
I like apple
I like grape
```

Else in For Loop

The *for* loop can also have an optional *else* block. If the items in the sequence used for the loop are exhausted, the *else* part is executed.

The *break* keyword can be used to stop the *for* loop. In this case, the *else* part will be ignored.

Therefore, if no interruption occurs, the *else* part of the *for* loop will run.

```python
for val in range(5):
    print(val)
else:
    print("Finished")
```

```
>>> %Run -c $EDITOR_CONTENT
0
1
2
3
4
Finished
```

The else block will NOT be executed if the loop is stopped by a break statement.

```python
for val in range(5):
    if val == 2: break
    print(val)
else:
    print("Finished")
```

```
>>> %Run -c $EDITOR_CONTENT
0
1
```

3.5.8 Functions

In MicroPython, a function is a group of related statements that perform a specific task.

Functions help break our program into smaller modular blocks. As our plan becomes larger and larger, functions make it more organized and manageable.

In addition, it avoids duplication and makes the code reusable.

Create a Function

```python
def function_nameparameters)
    """docstring"""
    statement(s)
```

- A function is defined using the `def` keyword

- A function name to uniquely identify the function. Function naming is the same as variable naming, and both follow the following rules.

  - Can only contain numbers, letters, and underscores.

  - The first character must be a letter or underscore.

- Case sensitive.

- Parameters (arguments) through which we pass values to a function. They are optional.

- The colon (:) marks the end of the function header.

- Optional docstring, used to describe the function of the function, we usually use triple quotes so that the docstring can be expanded to multiple lines.

- One or more valid Micropython statements that make up the function body. Statements must have the same indentation level (usually 4 spaces).

- Each function needs at least one statement, but if for some reason there is a function that does not contain any statement, please put in the pass statement to avoid errors.

- An optional `return` statement to return a value from the function.

Calling a Function

To call a function, add parentheses after the function name.

```
def my_function():
    print("Your first function")

my_function()
```

```
>>> %Run -c $EDITOR_CONTENT
Your first function
```

The return Statement

The return statement is used to exit a function and return to the place where it was called.

Syntax of return

```
return [expression_list]
```

The statement can contain an expression that is evaluated and returns a value. If there is no expression in the statement, or the `return` statement itself does not exist in the function, the function will return a `None` object.

```
def my_function():
        print("Your first function")

print(my_function())
```

```
>>> %Run -c $EDITOR_CONTENT
Your first function
None
```

Here, `None` is the return value, because the `return` statement is not used.

Arguments

Information can be passed to the function as arguments.

Specify arguments in parentheses after the function name. You can add as many arguments as you need, just separate them with commas.

```python
def welcome(name, msg):
    """This is a welcome function for
    the person with the provided message"""
    print("Hello", name + ', ' + msg)

welcome("Lily", "Welcome to China!")
```

```
>>> %Run -c $EDITOR_CONTENT
Hello Lily, Welcome to China!
```

Number of Arguments

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 parameters, you have to call the function with 2 arguments, not more, and not less.

```python
def welcome(name, msg):
    """This is a welcome function for
    the person with the provided message"""
    print("Hello", name + ', ' + msg)

welcome("Lily", "Welcome to China!")
```

Herethe function welcome() has 2 parameters.

Since we called this function with two arguments, the function runs smoothly without any errors.

If it is called with a different number of arguments, the interpreter will display an error message.

The following is the call to this function, which contains one and one no arguments and their respective error messages.

```
welcome("Lily")Only one argument
```

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 6, in <module>
TypeError: function takes 2 positional arguments but 1 were given
```

```
welcome()No arguments
```

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 6, in <module>
TypeError: function takes 2 positional arguments but 0 were given
```

Default Arguments

In MicroPython, we can use the assignment operator (=) to provide a default value for the parameter.

If we call the function without argument, it uses the default value.

```
def welcome(name, msg = "Welcome to China!"):
    """This is a welcome function for
    the person with the provided message"""
    print("Hello", name + ', ' + msg)
welcome("Lily")
```

```
>>> %Run -c $EDITOR_CONTENT
Hello Lily, Welcome to China!
```

In this function, the parameter `name` has no default value and is required (mandatory) during the call.

On the other hand, the default value of the parameter `msg` is "Welcome to China!". Therefore, it is optional during the call. If a value is provided, it will overwrite the default value.

Any number of arguments in the function can have a default value. However, once there is a default argument, all arguments on its right must also have default values.

This means that non-default arguments cannot follow default arguments.

For example, if we define the above function header as:

```
def welcome(name = "Lily", msg):
```

We will receive the following error message:

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
SyntaxError: non-default argument follows default argument
```

Keyword Arguments

When we call a function with certain values, these values will be assigned to arguments based on their position.

For example, in the above function welcome(), when we called it as welcome("Lily", "Welcome to China"), the value "Lily" gets assigned to the `name` and similarly "Welcome to China" to parameter `msg`.

MicroPython allows calling functions with keyword arguments. When we call the function in this way, the order (position) of the arguments can be changed.

```
# keyword arguments
welcome(name = "Lily",msg = "Welcome to China!")

# keyword arguments (out of order)
welcome(msg = "Welcome to China",name = "Lily")

#1 positional, 1 keyword argument
welcome("Lily", msg = "Welcome to China!")
```

As we can see, we can mix positional arguments and keyword arguments during function calls. But we must remember that the keyword arguments must come after the positional arguments.

Having a positional argument after a keyword argument will result in an error.

For example, if the function call as follows:

```
welcome(name="Lily","Welcome to China!")
```

Will result in an error:

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
SyntaxError: non-keyword arg after keyword arg
```

Arbitrary Arguments

Sometimes, if you do not know the number of arguments that will be passed to the function in advance.

In the function definition, we can add an asterisk (*) before the parameter name.

```
def welcome(*names):
    """This function welcomes all the person
    in the name tuple"""
    #names is a tuple with arguments
    for name in names:
        print("Welcome to China!", name)

welcome("Lily","John","Wendy")
```

```
>>> %Run -c $EDITOR_CONTENT
Welcome to China! Lily
Welcome to China! John
Welcome to China! Wendy
```

Here, we have called the function with multiple arguments. These arguments are packed into a tuple before being passed into the function.

Inside the function, we use a for loop to retrieve all the arguments.

Recursion

In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.

This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

```
def rec_func(i):
    if(i > 0):
        result = i + rec_func(i - 1)
        print(result)
    else:
        result = 0
    return result
```

(continues on next page)

```
rec_func(6)
```

```
>>> %Run -c $EDITOR_CONTENT
1
3
6
10
15
21
```

In this example, rec_func() is a function that we have defined to call itself ("recursion"). We use the `i` variable as the data, and it will decrement (-1) every time we recurse. When the condition is not greater than 0 (that is, 0), the recursion ends.

For new developers, it may take some time to determine how it works, and the best way to test it is to test and modify it.

Advantages of Recursion

- Recursive functions make the code look clean and elegant.

- A complex task can be broken down into simpler sub-problems using recursion.

- Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion

- Sometimes the logic behind recursion is hard to follow through.

- Recursive calls are expensive (inefficient) as they take up a lot of memory and time.

- Recursive functions are hard to debug.

3.5.9 Data Types

Built-in Data Types

MicroPython has the following data types:

- Text Type: str

- Numeric Types: int, float, complex

- Sequence Types: list, tuple, range

- Mapping Type: dict

- Set Types: set, frozenset

- Boolean Type: bool

- Binary Types: bytes, bytearray, memoryview

Getting the Data Type

You can get the data type of any object by using the *type()* function:

```
a = 6.8
print(type(a))
```

```
>>> %Run -c $EDITOR_CONTENT
<class 'float'>
```

Setting the Data Type

MicroPython does not need to set the data type specifically, it has been determined when you assign a value to the variable.

```
x = "welcome"
y = 45
z = ["apple", "banana", "cherry"]

print(type(x))
print(type(y))
print(type(z))
```

```
>>> %Run -c $EDITOR_CONTENT
<class 'str'>
<class 'int'>
<class 'list'>
>>>
```

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

Example	Date Type
x = int(20)	int
x = float(20.5)	float
x = complex(1j)	complex
x = str("Hello World")	str
x = list(("apple", "banana", "cherry"))	list
x = tuple(("apple", "banana", "cherry"))	tuple
x = range(6)	range
x = dict(name="John", age=36)	dict
x = set(("apple", "banana", "cherry"))	set
x = frozenset(("apple", "banana", "cherry"))	frozenset
x = bool(5)	bool
x = bytes(5)	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

You can print some of them to see the result.

```
a = float(20.5)
b = list(("apple", "banana", "cherry"))
c = bool(5)

print(a)
print(b)
print(c)
```

```
>>> %Run -c $EDITOR_CONTENT
20.5
['apple', 'banana', 'cherry']
True
>>>
```

Type Conversion

You can convert from one type to another with the int(), float(), and complex() methods: Casting in python is therefore done using constructor functions:

- int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)

- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)

- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

```
a = float("5")
b = int(3.7)
c = str(6.0)

print(a)
print(b)
print(c)
```

Note: You cannot convert complex numbers into another number type.

3.5.10 Operators

Operators are used to perform operations on variables and values.

- *Arithmetic Operators*
- *Assignment operators*
- *Comparison Operators*
- *Logical Operators*
- *Identity Operators*
- *Membership Operators*
- *Bitwise Operators*

Arithmetic Operators

You can use arithmetic operators to do some common mathematical operations.

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponentiation
//	Floor division

```python
x = 5
y = 3

a = x + y
b = x - y
c = x * y
d = x / y
e = x % y
f = x ** y
g = x // y

print(a)
print(b)
print(c)
print(d)
print(e)
print(f)
print(g)
```

```
>>> %Run -c $EDITOR_CONTENT
8
2
15
1.666667
2
125
1
8
2
15
>>>
```

Assignment operators

Assignment operators can used to assign values to variables.

Operator	Example	Same As
=	a = 6	a =6
+=	a += 6	a = a + 6
-=	a -= 6	a = a - 6
*=	a *= 6	a = a * 6
/=	a /= 6	a = a / 6
%=	a %= 6	a = a % 6
**=	a **= 6	a = a ** 6
//=	a //= 6	a = a // 6
&=	a &= 6	a = a & 6
|=	a |= 6	a = a | 6
^=	a ^= 6	a = a ^ 6
>>=	a >>= 6	a = a >> 6
<<=	a <<= 6	a = a << 6

```
a = 6

a *= 6
print(a)
```

```
>>> %Run test.py
36
>>>
```

Comparison Operators

Comparison operators are used to compare two values.

Operator	Name
==	Equal
!=	Not equal
<	Less than
>	Greater than
>=	Greater than or equal to
<=	Less than or equal to

```
a = 6
b = 8

print(a>b)
```

```
>>> %Run test.py
False
>>>
```

Return **False**, beause the **a** is less than the **b**.

Logical Operators

Logical operators are used to combine conditional statements.

Operator	Description
and	Returns True if both statements are true
or	Returns True if one of the statements is true
not	Reverse the result, returns False if the result is true

```
a = 6
print(a > 2 and a < 8)
```

```
>>> %Run -c $EDITOR_CONTENT
True
>>>
```

Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location.

Operator	Description
is	Returns True if both variables are the same object
is not	Returns True if both variables are not the same object

```
a = ["hello", "welcome"]
b = ["hello", "welcome"]
c = a

print(a is c)
# returns True because z is the same object as x

print(a is b)
# returns False because x is not the same object as y, even if they have the same
→content

print(a == b)
# returns True because x is equal to y
```

```
>>> %Run -c $EDITOR_CONTENT
True
False
True
>>>
```

Membership Operators

Membership operators are used to test if a sequence is presented in an object.

Operator	Description
in	Returns True if a sequence with the specified value is present in the object
not in	Returns True if a sequence with the specified value is not present in the object

```
a = ["hello", "welcome", "Goodmorning"]

print("welcome" in a)
```

```
>>> %Run -c $EDITOR_CONTENT
True
>>>
```

Bitwise Operators

Bitwise operators are used to compare (binary) numbers.

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

```
num = 2

print(num & 1)
print(num | 1)
print(num << 1)
```

```
>>> %Run -c $EDITOR_CONTENT
0
3
4
>>>
```

3.5.11 Lists

Lists are used to store multiple items in a single variable, and are created using square brackets:

```python
B_list = ["Blossom", "Bubbles","Buttercup"]
print(B_list)
```

List items are changeable, ordered, and allow duplicate values. The list items are indexed, with the first item having index [0], the second item having index [1], and so on.

```python
C_list = ["Red", "Blue", "Green", "Blue"]
print(C_list)              # duplicate
print(C_list[0])
print(C_list[1])          # ordered
C_list[2] = "Purple"      # changeable
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Blue', 'Green', 'Blue']
Red
Blue
['Red', 'Blue', 'Purple', 'Blue']
```

A list can contain different data types:

```python
A_list = ["Banana", 255, False, 3.14]
print(A_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Banana', 255, False, 3.14]
```

List Length

To determine how many items are in the list, use the len() function.

```python
A_list = ["Banana", 255, False, 3.14]
print(len(A_list))
```

```
>>> %Run -c $EDITOR_CONTENT
4
```

Check List items

Print the second item of the list:

```python
A_list = ["Banana", 255, False, 3.14]
print(A_list[1])
```

```
>>> %Run -c $EDITOR_CONTENT
[255]
```

Print the last one item of the list:

```
A_list = ["Banana", 255, False, 3.14]
print(A_list[-1])
```

```
>>> %Run -c $EDITOR_CONTENT
[3.14]
```

Print the second, third item:

```
A_list = ["Banana", 255, False, 3.14]
print(A_list[1:3])
```

```
>>> %Run -c $EDITOR_CONTENT
[255, False]
```

Change List Items

Change the second, third item:

```
A_list = ["Banana", 255, False, 3.14]
A_list[1:3] = [True,"Orange"]
print(A_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Banana', True, 'Orange', 3.14]
```

Change the second value by replacing it with two values:

```
A_list = ["Banana", 255, False, 3.14]
A_list[1:2] = [True,"Orange"]
print(A_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Banana', True, 'Orange', False, 3.14]
```

Add List Items

Using the append() method to add an item:

```
C_list = ["Red", "Blue", "Green"]
C_list.append("Orange")
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Blue', 'Green', 'Orange']
```

Insert an item as the second position:

```
C_list = ["Red", "Blue", "Green"]
C_list.insert(1, "Orange")
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Orange', 'Blue', 'Green']
```

Remove List Items

The remove() method removes the specified item.

```python
C_list = ["Red", "Blue", "Green"]
C_list.remove("Blue")
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Green']
```

The pop() method removes the specified index. If you do not specify the index, the pop() method removes the last item.

```python
A_list = ["Banana", 255, False, 3.14, True,"Orange"]
A_list.pop(1)
print(A_list)
A_list.pop()
print(A_list)
```

```
>>> %Run -c $EDITOR_CONTENT
255
['Banana', False, 3.14, True, 'Orange']
'Orange'
['Banana', False, 3.14, True]
```

The `del` keyword also removes the specified index:

```python
C_list = ["Red", "Blue", "Green"]
del C_list[1]
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
['Red', 'Green']
```

The clear() method empties the list. The list still remains, but it has no content.

```python
C_list = ["Red", "Blue", "Green"]
C_list.clear()
print(C_list)
```

```
>>> %Run -c $EDITOR_CONTENT
[]
```

2. Output & Input

3.6 2.1 Hello, LED!

Just as printing "Hello, world!" is the first step in learning programming, letting the LED light up is the traditional entry to learning physical programming.

There is a small LED on the top of the Pico. Like other LEDs, it will glow when power is on and go out when power is off.
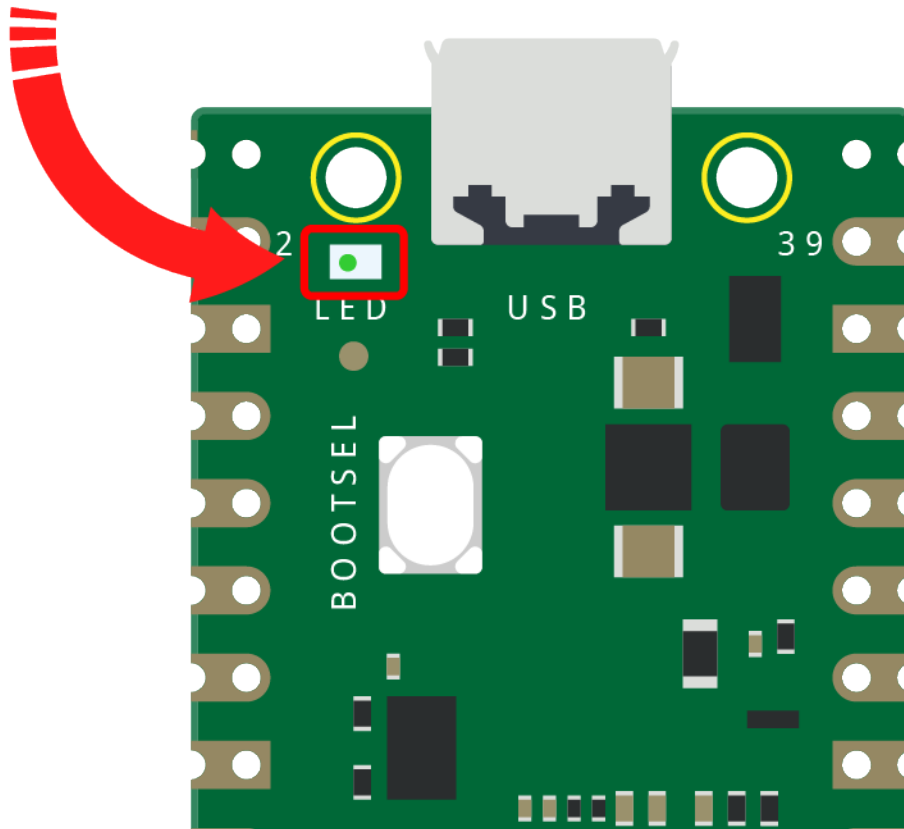
3.6.1 Code

Let's copy this code into Thonny and click "Run Current Script" or simply press F5 to run it to make the LED blink!

Don't forget to left click on the bottom right corner and switch the python version name to MicroPython (Raspberry Pi Pico).

```python
import machine
import utime

led_onboard = machine.Pin(25, machine.Pin.OUT)
while True:
    led_onboard.value(1)
    utime.sleep(2)
    led_onboard.value(0)
    utime.sleep(2)
```

3.6.2 How it works?

The onboard LED is connected to the GP25 pin, if you carefully observe the Pico pinout, you will find that GP25 is one of the hidden pins, which means that we cannot use this pin (even if GP25 is used in exactly the same way as other pins). The advantage of this design is that even if you don't connect any external components, you can still have an OUTPUT to test the program.

The machine library is required to use GPIO.

```python
import machine
```

This library contains all the instructions needed to communicate between MicroPython and Pico. Without this line of code, we will not be able to control any GPIOs (Of course including GP25).

The next thing to notice is this line:

```python
led_onboard = machine.Pin(25, machine.Pin.OUT)
```

An object named `led_onboard` is defined here. Technically, it can be any name, it can be x, y, banana, Micheal_Jackson, or any character, but it is best to use a name that describes the purpose to ensure that the program is easy to read.

The second part of this line (the part after the equal sign) calls the Pin function in the machine library. It is used to tell Pico's GPIO pins what to do. The Pin function has two parameters: the first parameter (25) means the pin you want to set; the second parameter (machine.Pin.OUT) tells that the pin should be used as an output instead of an input.

The above code has "set" the pin, but it will not light up the LED. To do this, we also need to "use" the pin.

```python
led_onboard.value(1)
```

We have set up the GP25 pin before and named it led_onboard. The function of this statement is to set the value of `led_onboard` to 1 to turn the on-board LED on.

All in all, to use GPIO, these steps are necessary:

- **import machine library**: This is necessary, and it is only executed once in the entire program.
- **Set GPIO**: Each pin should be set before use.
- **Use**: Assign a value to the pin, each assignment will change the working state of the pin.

If we follow the above steps to write an example, then you will get code like this:

```python
import machine
led_onboard = machine.Pin(25, machine.Pin.OUT)
led_onboard.value(1)
```

Run it and you will be able to light up the onboard LED.

Next, we try to add the "extinguished" statement:

```python
import machine
led_onboard = machine.Pin(25, machine.Pin.OUT)
led_onboard.value(1)
led_onboard.value(0)
```

According to the code line, this program will make the onboard LED turn on first and then turn off. But when you use it, you will find that this is not the case. The onboard LED never seems to light up. This is because the execution speed between the two lines is very fast, much faster than the reaction time of the human eye. The moment the onboard LED lights up is not enough to make us perceive the light. To fix that, we need to slow down the program.

Insert the following statement into the second line of the program:

```python
import utime
```

Like machine, the `utime` library is introduced here, which handles all time-related things, including the delay we need to use. Let's insert a delay sentence between `led_onboard.value(1)` and `led_onboard.value(0)`, let them be separated by 2 seconds:

```python
utime.sleep(2)
```

Now, the code should look like this. Run it, we will be able to see that the onboard LED turns on first and then turns off:

```python
import machine
import utime
led_onboard = machine.Pin(25, machine.Pin.OUT)
led_onboard.value(1)
utime.sleep(2)
led_onboard.value(0)
```

Finally, we should make the LED blink. Create a loop, rewrite the program, and it will be what you saw at the beginning of this chapter.

```python
import machine
import utime

led_onboard = machine.Pin(25, machine.Pin.OUT)
while True:
    led_onboard.value(1)
    utime.sleep(2)
    led_onboard.value(0)
    utime.sleep(2)
```

3.6.3 What More?

Usually, the library will have a corresponding API (Application Programming Interface) file. This is a concise reference manual that contains all the information needed to use this library, detailed introduction to functions, classes, return types, parameters, etc., and even comes with a tutorial.

In this article, we used MicroPython's `machine` and `utime` libraries, we can find more ways to use them here.

- machine.Pin

- utime

The following is also an example of making the LED blink, please try to read the API file to understand it!

```python
import machine
import utime

led_onboard = machine.Pin(25, machine.Pin.OUT)
while True:
    led_onboard.toggle()
    utime.sleep(1)
```

3.7 2.2 Hello, Breadboard!

To use extended electronic components, a solderless breadboard will be the most powerful partner for novice users.

The breadboard is a rectangular plastic plate with a bunch of small holes in it. These holes allow us to easily insert electronic components and build electronic circuits. The breadboard does not permanently fix the electronic components, which makes it easy for us to repair the circuit and start over when we make a mistake.

Note: We can use the breadboard without using special tools. But many electronic components are very small. A pair of tweezers can help us pick up small parts better.

There are many detailed breadboard knowledge on the Internet, let us use it wisely.

- How to Use a Breadboard - Science Buddies
- What is a BREADBOARD? - Makezine

For breadboards, what you need to know clearly is:

1. Each group of half rows inside the breadboard (such as column A-E in row 1 or column F-J in row 3) is connected. This means that when an electrical signal flows in from A1, it can flow from B1, C1, D1, E1, but not from F1 or A2.

2. Both sides of the breadboard are usually used as power buses, and the holes in each column (about 50 holes) are connected. Generally speaking, the hole near the red wire is used to connect the positive power supply, and the hole near the blue wire is used to connect the negative power supply.

3. When building a circuit, the current flows from the positive pole and must first flow through the consumer before it can flow into the negative pole. Otherwise, a short circuit may occur.

Now we should have a general impression of the breadboard circuit, why not try to build a "Hello, LED!" expansion circuit?

3.7.1 Schematic

3.7.2 Wiring

Let us follow the direction of the current to build the circuit!

1. Here we use the electrical signal from the GP15 pin of the Pico board to make the LED work, and the circuit starts from here.

2. The current needs to pass through a 220 ohm resistor (used to protect the LED). Insert one end (either end) of the resistor into the same row as the Pico GP15 pin (row 20 in my circuit), and insert the other end into the free row of the breadboard (row 24 in my circuit).

   Note: The color ring of the 220 ohm resistor is red, red, black, black and brown.

   ---

3. Pick up the LED, you will see that one of its leads is longer than the other. Insert the longer lead into the same row as the end of the resistor, and connect the shorter lead across the middle gap of the breadboard to the same row.

   Note: The longer lead is known as the anode, and represents the positive side of the circuit; the shorter lead is the cathode, and represents the negative side.

   The anode needs to be connected to the GPIO pin through a resistor; the cathode needs to be connected to the GND pin.

   ---

4. Insert the male-to-male (M2M) jumper wire into the same row as the LED short pin, and then connect it to the negative power bus of the breadboard.

5. Use a jumper to connect the negative power bus to the GND pin of Pico.

3.7.3 Code

The method of controlling the extended LED is the same as the method of controlling the on-board LED, the only difference is that the operating pin is changed to 15.

```python
import machine
import utime

led = machine.Pin(15, machine.Pin.OUT)
while True:
    led.toggle()
    utime.sleep(1)
```

`toggle()` can switch the pin between high level and low level, and you can see the LED blinking.

Also see reference here:

- machine.Pin

3.8 2.3 Reading Button Value

From the name of GPIO (General-purpose input/output), we can see that these pins have both input and output functions. In the previous two projects, we used the output function, in this chapter we will use the input function to input read the button value.

3.8.1 Schematic

3.8.2 Wiring

Let's follow the direction of the circuit to build the circuit!

1. Connect the 3V3 pin of Pico to the positive power bus of the breadboard.

2. Insert the button into the breadboard and straddle the central dividing line.

> **Note:** We can think of the four-legged button as an H-shaped button. Its left (right) two feet are connected, which means that after it straddles the central dividing line, it will connect the two half rows of the same row number together. (For example, in my circuit, E23 and F23 have been connected, as are E25 and F25).
>
> Before the button is pressed, the left and right sides are independent of each other, and current cannot flow from one side to the other.

3. Use a jumper wire to connect one of the button pins to the positive bus (mine is the pin on the upper right).

4. Connect the other pin (upper left or lower left) to GP14 with a jumper wire.

5. Use a 10K resistor to connect the pin on the upper left corner of the button and the negative bus.

> **Note:** The color ring of the 10k resistor is brown, black, black, red, brown.

Buttons require pull-up resistors or pull-down resistors. If there is no pull-up or pull-down resistor, the main controller may receive a 'noisy' signal which can trigger even when you're not pushing the button.

6. Connect the negative power bus of the breadboard to Pico's GND.

3.8.3 Code

When the button is pressed, the current will flow from 3V3 through the button to GP14, in other words, GP14 will read a high-level signal '1'; otherwise, it will read a low-level signal '0'.

```python
import machine
import utime
button = machine.Pin(14, machine.Pin.IN)
while True:
    if button.value() == 1:
        print("You pressed the button!")
        utime.sleep(1)
```

3.8.4 Pull-up Working Mode

Next is the wiring and code when the button in the pull-up working mode, please try it.
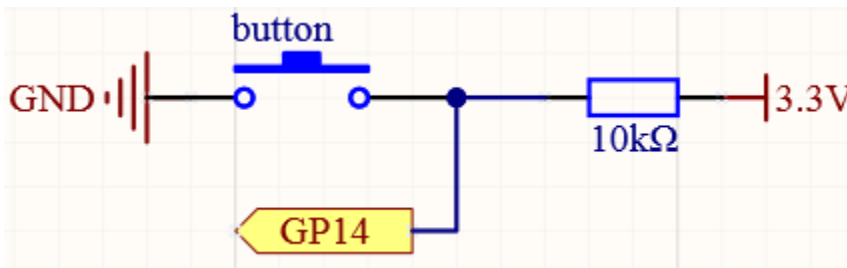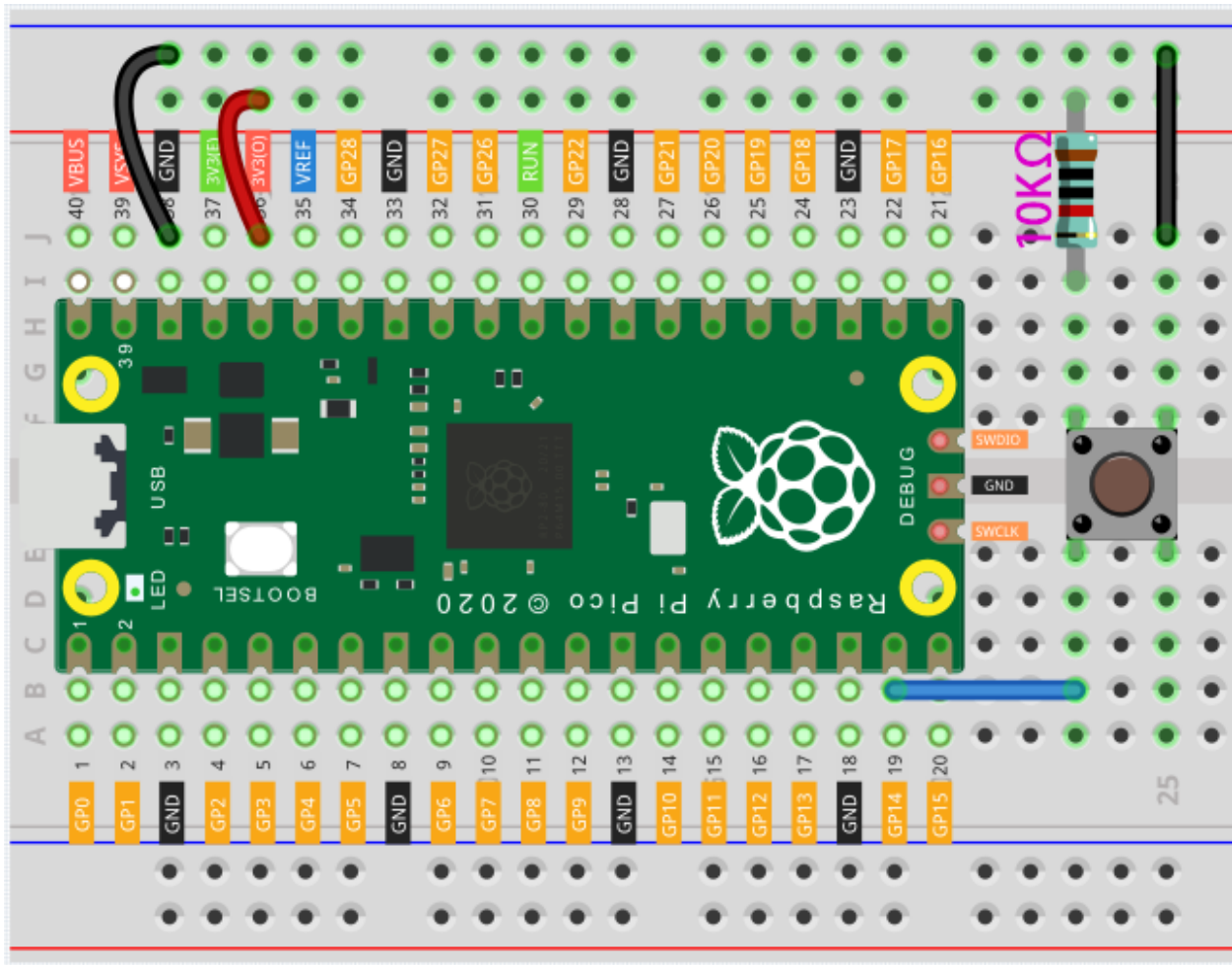
1. Connect the 3V3 pin of Pico to the positive power bus of the breadboard.

2. Insert the button into the breadboard and straddle the central dividing line.

3. Use a jumper wire to connect one of the button pins to the **negative** bus (mine is the pin on the upper right).

4. Connect the other pin (upper left or lower left) to GP14 with a jumper wire.

5. Use a 10K resistor to connect the pin on the upper left corner of the button and the **positive** bus.

6. Connect the negative power bus of the breadboard to Pico's GND.

```python
import machine
import utime
button = machine.Pin(14, machine.Pin.IN)
while True:
    if button.value() == 0:
        # When the button is pressed, GPIO will be connected to GND.
        print("You pressed the button!")
        utime.sleep(1)
```

Also see the reference here:

- machine.Pin

3.8.5 What more?

There are two components in this kit that work on the same principle as buttons, they are tilt switch and slide switch. These components can use the same code as the button. Their wiring is as follows:

Tilt switch
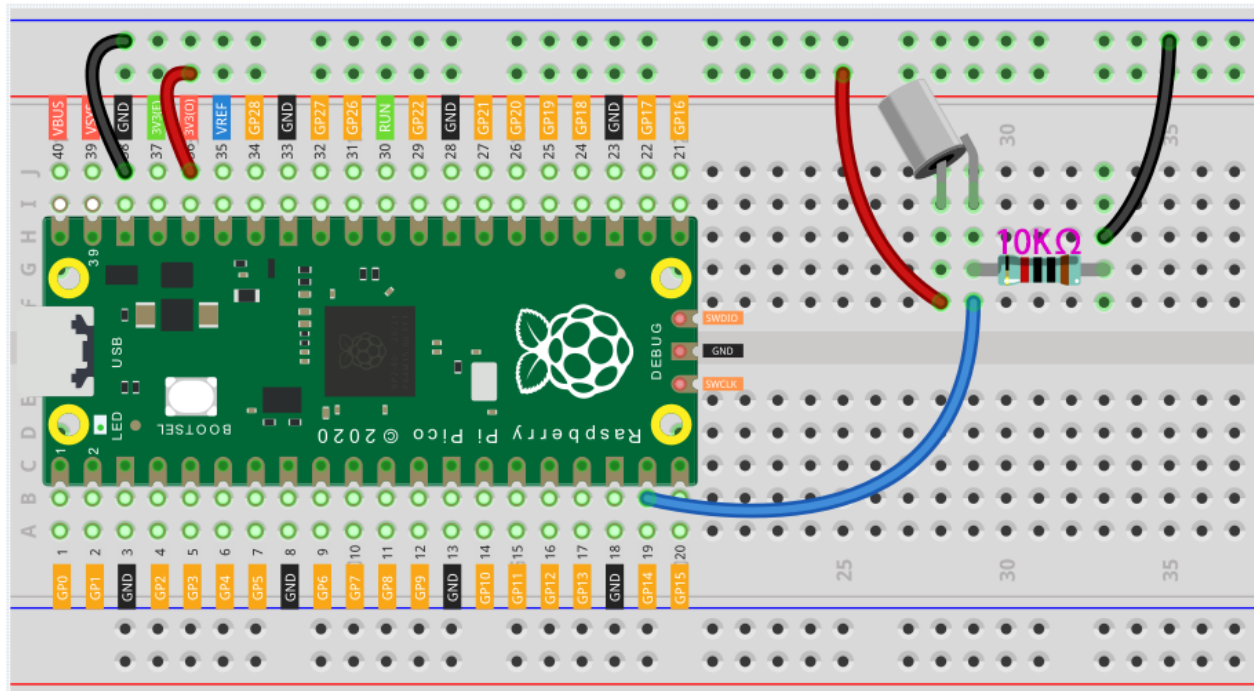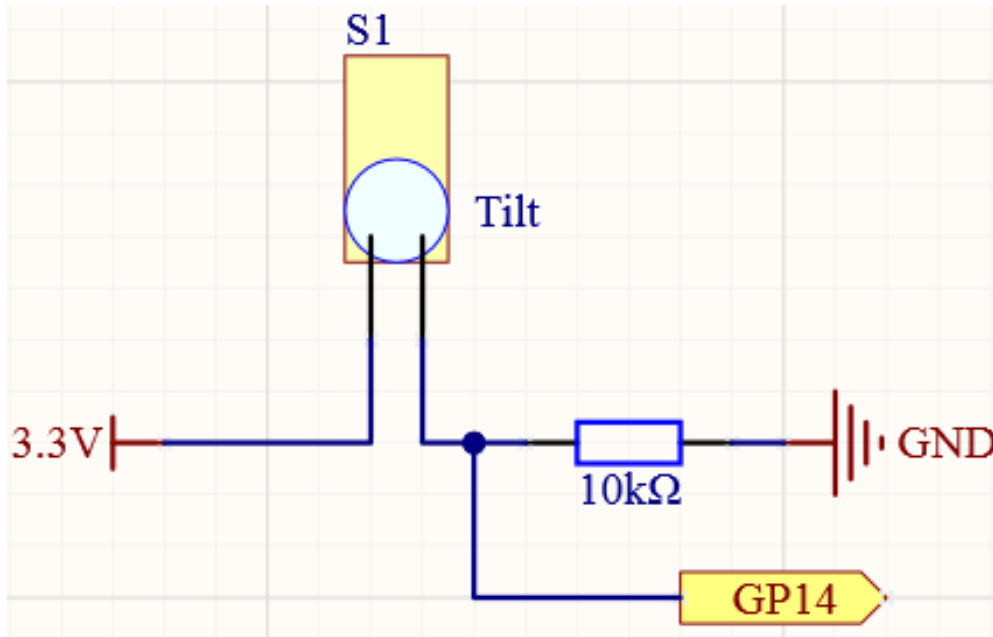
1. Connect the 3V3 pin of Pico to the positive power bus of the breadboard.

2. Insert the tilt switch into the breadboard.

3. Use a jumper wire to connect one end of tilt switch pin to the positive bus.

4. Connect the other pin to GP14 with a jumper wire.

5. Use a 10K resistor to connect the second pin (which connected to GP14) and the negative bus.

6. Connect the negative power bus of the breadboard to Pico's GND.

When you put a flat breadboard, the circuit will be closed. When you tilt the breadboard, the circuit is open.

- *Tilt Switch*

Slide switch
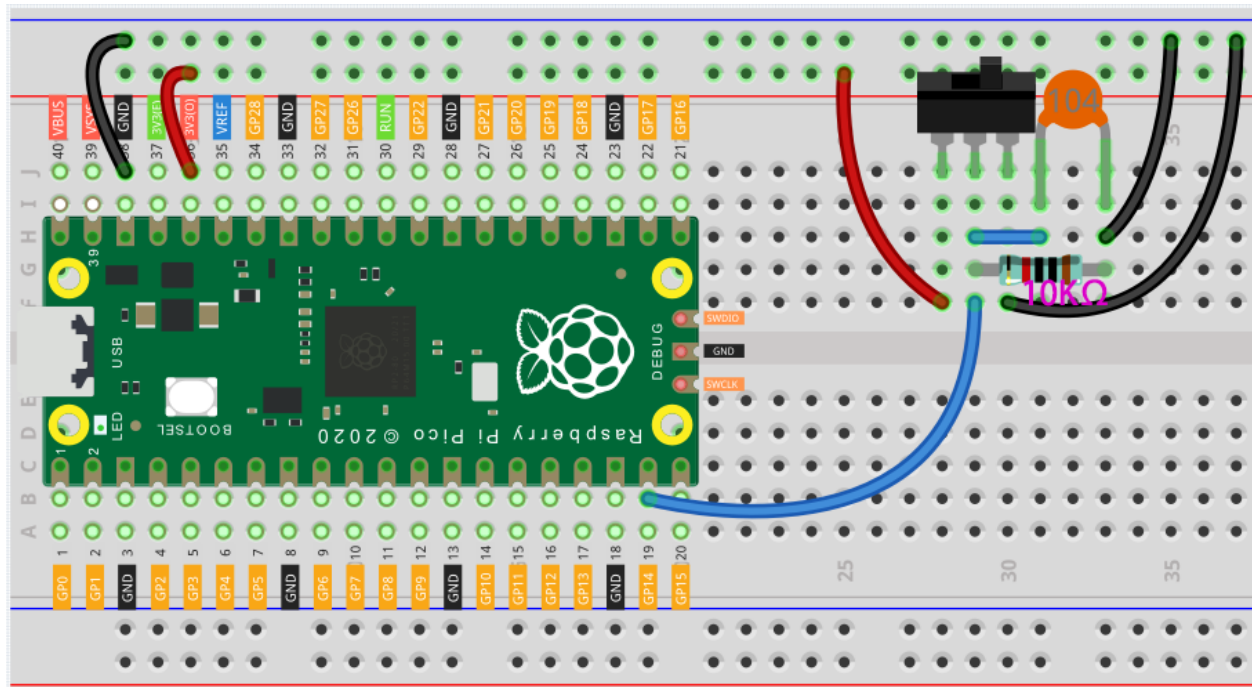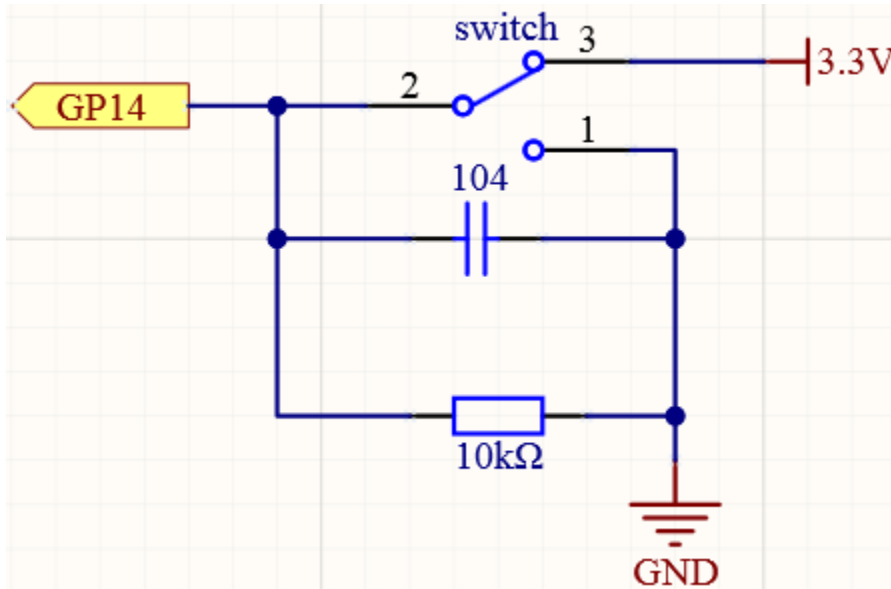
1. Connect the 3V3 pin of Pico to the positive power bus of the breadboard.

2. Insert the slide switch into the breadboard.

3. Use a jumper wire to connect one end of slide switch pin to the negative bus.

4. Connect the middle pin to GP14 with a jumper wire.

5. Use a jumper wire to connect last end of slide switch pin to the positive bus

6. Use a 10K resistor to connect the middle pin of the slide switch and the negative bus.

7. Use a 104 capacitor to connect the middle pin of the slide switch and the negative bus to realize debounce that may arise from your toggle of switch.

8. Connect the negative power bus of the breadboard to Pico's GND.

When you toggle the slide switch, the circuit will switch between closed and open.

- *Slide Switch*

- *Capacitor*

3.9 2.4 Reaction Game

Microcontrollers not only appear in industrial equipment, they are also used to control a large number of electronic devices in the home, including toys and games. In this chapter, we will use "button" and "LED" to build a simple reaction timing game.

The study of reaction time is called mental chronometry, it is a hard science, and it is also the basis of many games (Including the games you are about to make).

Reaction time is the time that elapses between a person being presented with a stimulus and the person initiating a motor response to the stimulus, in milliseconds, the average reaction time of a person is about 200-250 milliseconds. People with quick reaction time have a huge advantage in the game!

3.9.1 Schematic
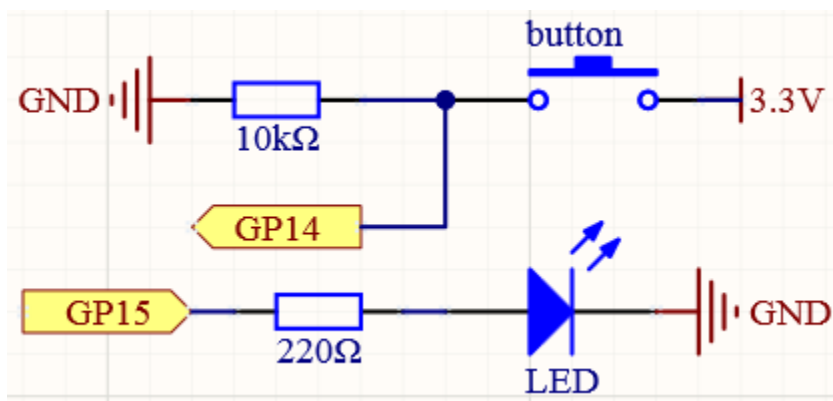
3.9.2 Wiring
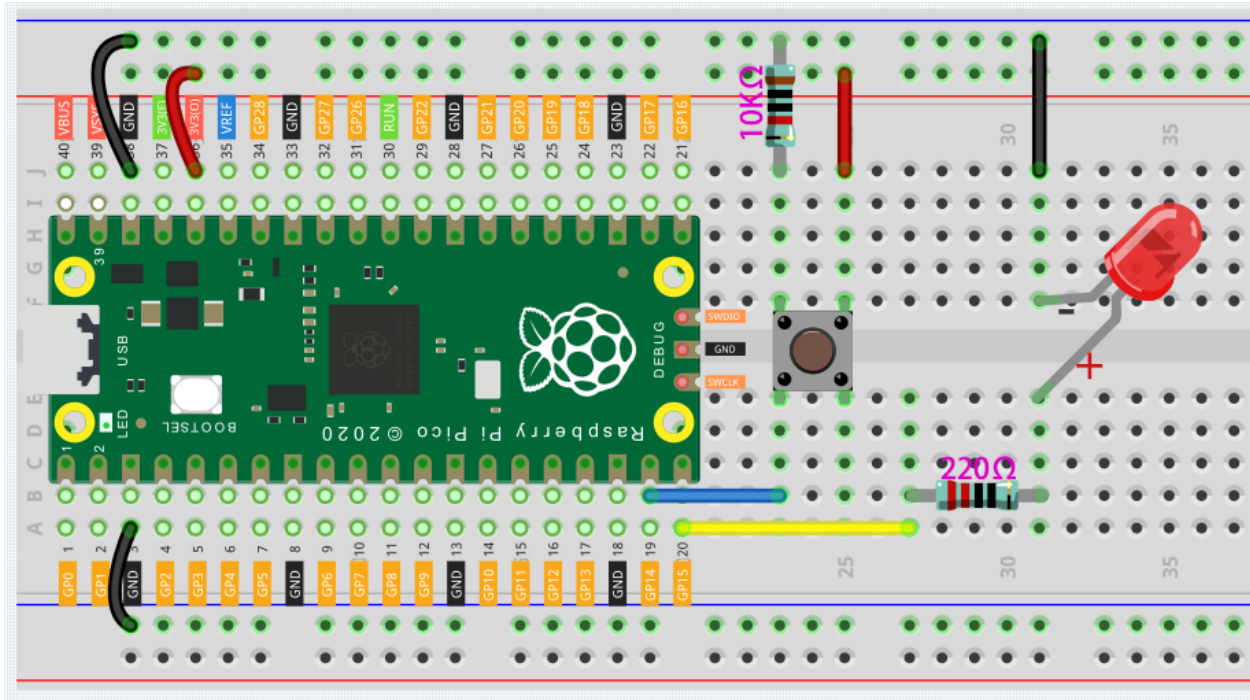
1. In general, this circuit is a combination of the circuits in the previous two projects.

2. Confirm again that the breadboard power bus is not connected wrongly or short-circuited!

3.9.3 Code

When the program starts, the LED will turn off within 5 to 10 seconds. You need to press the button as fast as possible, and the program will tell you what your reaction time is.

```python
import machine
import utime
import urandom

led = machine.Pin(15, machine.Pin.OUT)
button = machine.Pin(14, machine.Pin.IN)

def button_press(pin):
    button.irq(handler=None)
    rection_time = utime.ticks_diff(utime.ticks_ms(), timer_light_off)
    print("Your reaction time was " + str(rection_time) + " milliseconds!")

led.value(1)
utime.sleep(urandom.uniform(5, 10))
led.value(0)
timer_light_off = utime.ticks_ms()
button.irq(trigger=machine.Pin.IRQ_RISING, handler=button_press)
```

3.9.4 How it works?

In the previous project, we simply read the button value. This time, we tried a flexible way of using buttons: interrupt requests, or IRQs.

For example, you are reading a book page by page, as if a program is executing a thread. At this time, someone came to you to ask a question and interrupted your reading. Then the person is executing the interrupt request: asking you to stop what you are doing, answer his questions, and then let you return to reading the book after the end.

MicroPython interrupt request also works in the same way, it allows certain operations to interrupt the main program. It is achieved through the following two statements (the highlighted 2 lines):

```python
import machine
import utime
import urandom

led = machine.Pin(15, machine.Pin.OUT)
button = machine.Pin(14, machine.Pin.IN)

def button_press(pin):
    button.irq(handler=None)
    rection_time = utime.ticks_diff(utime.ticks_ms(), timer_light_off)
    print("Your reaction time was " + str(rection_time) + " milliseconds!")

led.value(1)
utime.sleep(urandom.uniform(5, 10))
led.value(0)
timer_light_off = utime.ticks_ms()
button.irq(trigger=machine.Pin.IRQ_RISING, handler=button_press)
```

Here, a callback function (button_press) is first defined, which is called an interrupt handler. It will be executed when an interrupt request is triggered. Then, set up an interrupt request in the main program, it contains two parts: `trigger` and `handler`.

- In this program, the `trigger` is `IRQ_RISING`, which means that the value of the pin rises from low level to high level (That is, pressing the button).
- `handler` is the callback function `button_press` we defined before.

In this example, you will find a statement `button.irq(handler=None)` in the callback function, which is equivalent to canceling the interrupt.

In order to better understand the interrupt request, we change the above code to the following (Use the same circuit):

```python
import machine
import utime

button = machine.Pin(14, machine.Pin.IN)
count = 0

def button_press(pin):
    print("You press the button!")
    utime.sleep(1)

button.irq(trigger=machine.Pin.IRQ_RISING, handler=button_press)

while True:
    count+=1
    print(count)
    utime.sleep(1)
```

When the program runs, it will start counting and print the numbers in the shell. When we press the button, it will stop counting and enter the callback function to print "You press the button!".

Go back to the original example. We need to make the LED turn off in a random time of 5 to 10 seconds, which is achieved by the following two lines:

```python
import machine
import utime
import urandom

led = machine.Pin(15, machine.Pin.OUT)
button = machine.Pin(14, machine.Pin.IN)

def button_press(pin):
    button.irq(handler=None)
    rection_time = utime.ticks_diff(utime.ticks_ms(), timer_light_off)
    print("Your reaction time was " + str(rection_time) + " milliseconds!")

led.value(1)
utime.sleep(urandom.uniform(5, 10))
led.value(0)
timer_light_off = utime.ticks_ms()
button.irq(trigger=machine.Pin.IRQ_RISING, handler=button_press)
```

The `urandom` library is loaded here. Use the `urandom.uniform(5,10)` function to generate a random number, the 'uniform' part referring to a uniform distribution between those two numbers.

If needed, try running the following example of random number generation:

```python
import machine
import utime
import urandom

while True:
    print(urandom.uniform(1, 20))
    utime.sleep(1)
```

The last two statements you need to understand are `utime.ticks_ms()` and `utime.ticks_diff()`.

```python
import machine
import utime
import urandom

led = machine.Pin(15, machine.Pin.OUT)
button = machine.Pin(14, machine.Pin.IN)

def button_press(pin):
    button.irq(handler=None)
    rection_time = utime.ticks_diff(utime.ticks_ms(), timer_light_off)
    print("Your reaction time was " + str(rection_time) + " milliseconds!")

led.value(1)
utime.sleep(urandom.uniform(5, 10))
led.value(0)
timer_light_off = utime.ticks_ms()
button.irq(trigger=machine.Pin.IRQ_RISING, handler=button_press)
```

- The `utime.ticks_ms()` function will output the number of milliseconds that have passed since the `utime` library started counting and store it in the variable `timer_light_off`.

- `utime.ticks_diff()` is used to output the time difference between two time nodes. The two time nodes in this function are `utime.ticks_ms()`, the current program time (press the button) and the reference time (light off) stored in the variable `timer_light_off`.

These two functions are usually used together to calculate the execution time of the program. Here we use it to calculate the time from when the light turns off to when the button is pressed.

Finally, this time will be printed out.

```python
print("Your reaction time was " + str(rection_time) + " milliseconds!")
```

Also see the reference here:

- machine.Pin
- urandom
- utime

3.9.5 What more?

Playing with your friends will be more fun, why not add buttons and see who can press the buttons the fastest?

Please try it.
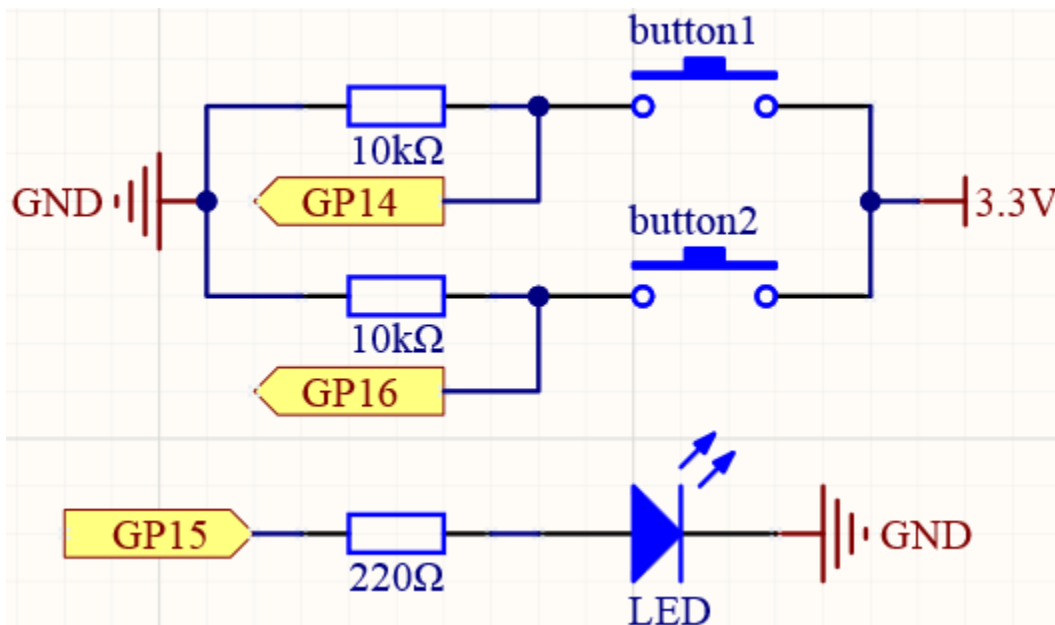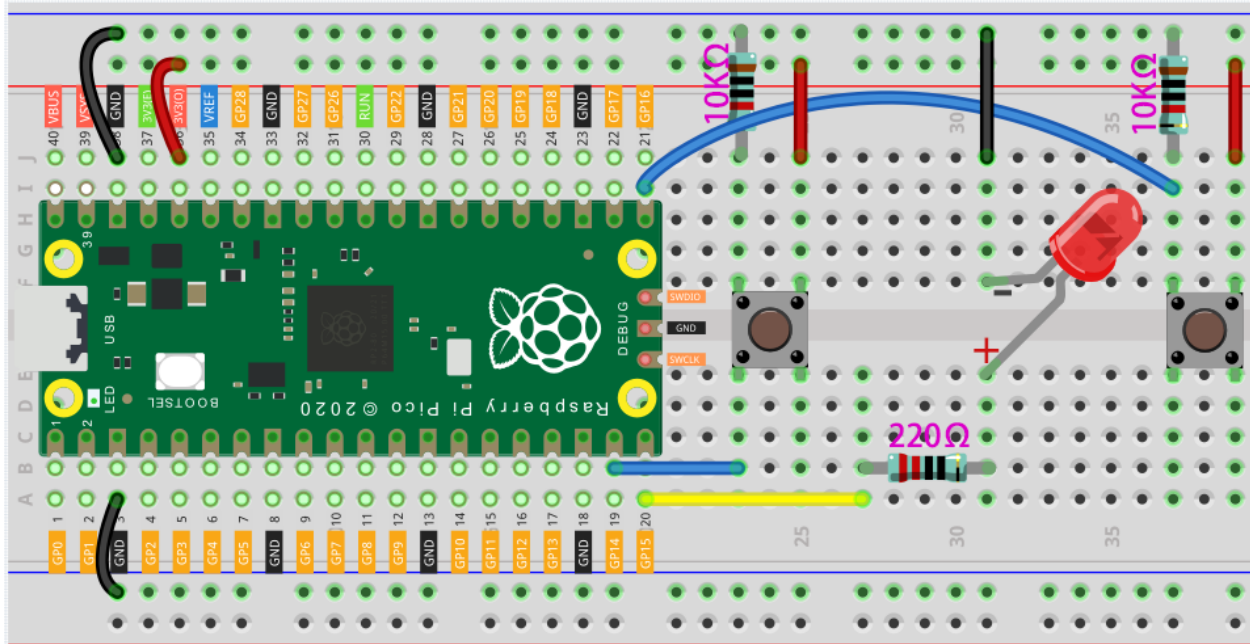
```python
import machine
import utime
import urandom

led = machine.Pin(15, machine.Pin.OUT)
left_button = machine.Pin(14, machine.Pin.IN)
right_button = machine.Pin(16, machine.Pin.IN)

def button_press(pin):
    left_button.irq(handler=None)
    right_button.irq(handler=None)
    rection_time = utime.ticks_diff(utime.ticks_ms(), timer_light_off)
    if pin == left_button:
        print("Left player is winner!")
    elif pin == right_button:
        print("Right player is winner!")
    print("Your reaction time was " + str(rection_time) + " milliseconds!")

led.value(1)
utime.sleep(urandom.uniform(5, 10))
led.value(0)
timer_light_off = utime.ticks_ms()
right_button.irq(trigger=machine.Pin.IRQ_RISING, handler=button_press)
left_button.irq(trigger=machine.Pin.IRQ_RISING, handler=button_press)
```

3.10 2.5 Traffic Light

In addition to home electronic devices, many microcontrollers can also be found in the social environment, including traffic lights. Traffic lights are used to direct traffic operation and are generally composed of red, green, and yellow lights. Building a huge traffic management system is a fairly advanced project, but using Pico to drive a miniature traffic light is a project we can try.

Let's make a mini pedestrian crossing system with a few LEDs and a button!

3.10.1 Schematic
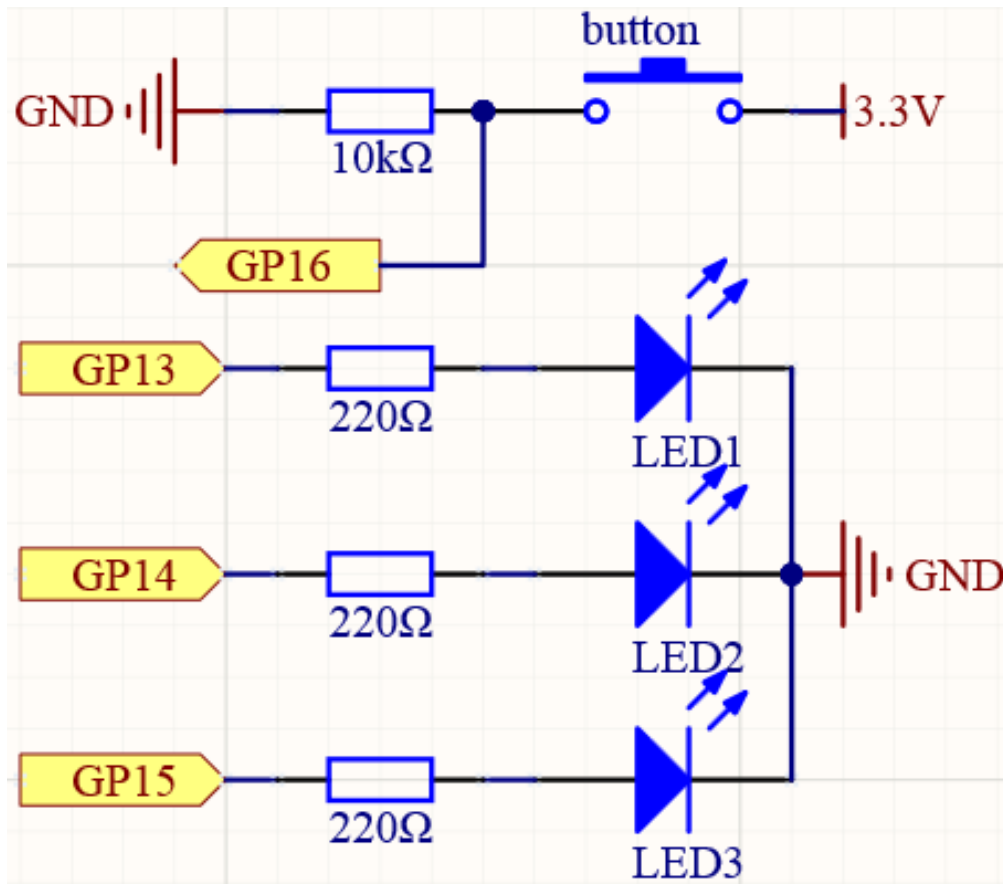
3.10.2 Wiring
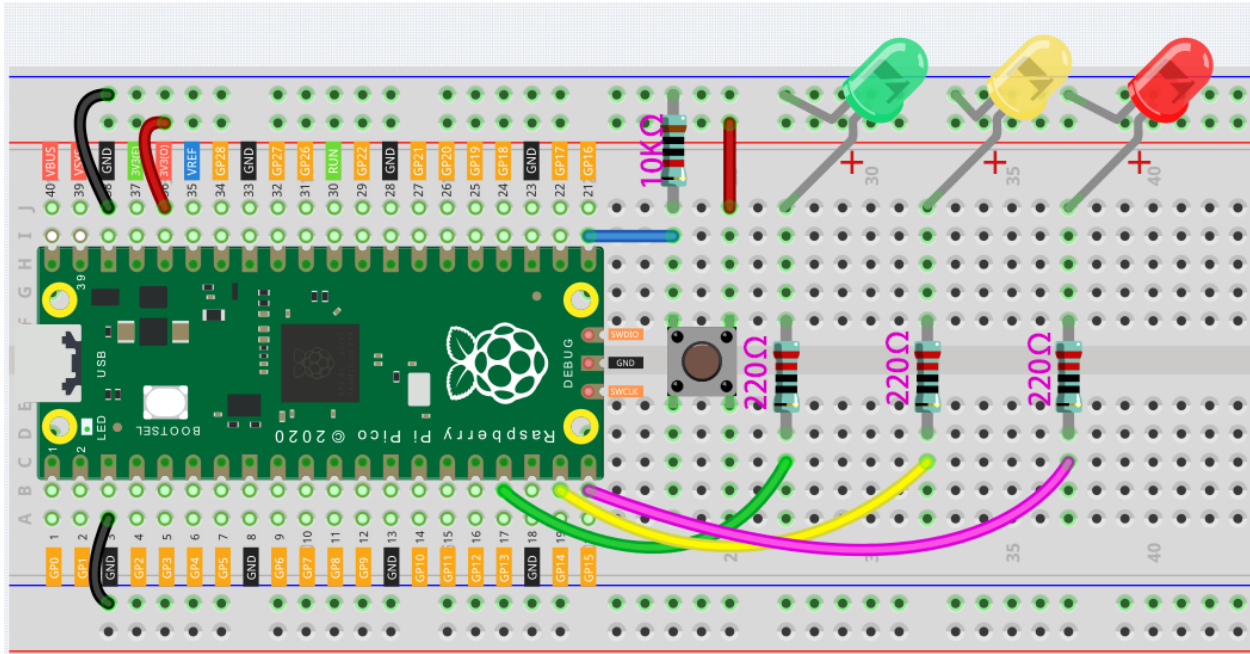
1. Connect 3V3 and GND of Pico to the breadboard power bus.

2. Insert the green, yellow and red LEDs on the breadboard.

3. GP13, GP14, GP15 are each connected to a 220 resistor and then connected to the anode of the LED.

4. Connect the cathodes of the LEDs to the negative power bus of the breadboard.

5. Insert the Button, connect one side of it to the GP16 pin, then use a 10K resistor to connect the same side and the negative bus. Do not forget to connect the other side to the positive power bus.

Note: The color ring of 220 ohm resistor is red, red, black, black and brown.

The color ring of the 10k ohm resistor is brown, black, black, red and brown.

3.10.3 Code

When the program is started, the traffic light will switch in the order of red for 5 seconds, yellow for 2 seconds, green for 5 seconds, and yellow for 2 seconds. If we (pedestrians) press the button, the red LED will be extended to 15 seconds, which will give us more time to cross the road.

```python
import machine
import utime
import _thread

led_red = machine.Pin(15, machine.Pin.OUT)
led_yellow = machine.Pin(14, machine.Pin.OUT)
led_green = machine.Pin(13, machine.Pin.OUT)
button = machine.Pin(16, machine.Pin.IN)

global button_status
```

(continues on next page)

```python
button_status = 0

def button_thread():
    global button_status
    while True:
        if button.value() == 1:
            button_status = 1

_thread.start_new_thread(button_thread, ())

while True:
    if button_status == 1:
        led_red.value(1)
        utime.sleep(10)
    global button_status
    button_status = 0

    led_red.value(1)
    utime.sleep(5)
    led_red.value(0)

    led_yellow.value(1)
    utime.sleep(2)
    led_yellow.value(0)

    led_green.value(1)
    utime.sleep(5)
    led_green.value(0)

    led_yellow.value(1)
    utime.sleep(2)
    led_yellow.value(0)
```

3.10.4 How it works?

In the previous projects, we have successfully made the LED blink. In other words, it is very simple for us to write a code that makes the traffic light cycle color. What we need to do is to add a judgment on the state of the button. But if we directly write the statement that reads the button value into the main program, we will find that it doesn't fit anywhere. Even if it is written in, it is difficult for us to read this value. This is because the program is stuck when executing `utime.sleep()`, and the statement to read the button value is not executed at this time.

Of course, we can read the button value through the IRQ in the previous project. But this time we take another approach-multithreading.

Multi-threading can be simply understood as dividing a thing into multiple parts, which are executed by different people (or processors). Just like when the chef is frying the steak, the assistant chef makes the sauce so that the newly prepared sauce can be poured on the properly prepared steak to make the best cooking.

Look at these lines:

```python
import machine
import utime
import _thread

led_red = machine.Pin(15, machine.Pin.OUT)
```

```python
led_yellow = machine.Pin(14, machine.Pin.OUT)
led_green = machine.Pin(13, machine.Pin.OUT)
button = machine.Pin(16, machine.Pin.IN)

global button_status
button_status = 0

def button_thread():
    global button_status
    while True:
        if button.value() == 1:
            button_status = 1

_thread.start_new_thread(button_thread, ())

while True:
    if button_status == 1:
        led_red.value(1)
        utime.sleep(10)
    global button_status
    button_status = 0

    led_red.value(1)
    utime.sleep(5)
    led_red.value(0)

    led_yellow.value(1)
    utime.sleep(2)
    led_yellow.value(0)

    led_green.value(1)
    utime.sleep(5)
    led_green.value(0)

    led_yellow.value(1)
    utime.sleep(2)
    led_yellow.value(0)
```

Here, the `_thread` library is imported first. This module implements multithreading support. Then define a thread `button_thread()`, which is independent of the main thread. It is used here to read the state of the button. Finally use `_thread.start_new_thread(button_thread, ())` to start the thread.

The following sample code can help you better understand multithreading:

```python
import machine
import utime
import _thread

led_red = machine.Pin(15, machine.Pin.OUT)
led_yellow = machine.Pin(14, machine.Pin.OUT)
button = machine.Pin(16, machine.Pin.IN)

def led_yellow_thread():
    while True:
        led_yellow.toggle()
        utime.sleep(2)
```

```
_thread.start_new_thread(led_yellow_thread, ())

while True:
    button_status = button.value()
    if button_status == 1:
        led_red.value(1)
    elif button_status == 0:
        led_red.value(0)
```

In the main thread, the button is used to control the red LED on and off. In the new thread (led_yellow_thread()), the yellow LED will change every 2 seconds. The two threads work independently of each other.

Let's go back to the traffic signal program. We let the main thread change the light and let the new thread read the button value. However, the threads are independent of each other, and we need a way for the new thread to pass information to the main thread, which requires the use of global variable.

The variables we have used before are all local variables, acting only in a certain part of the program (Variables declared in the main function cannot be used in sub-functions, and variables declared in the main thread cannot be used in the new thread). The global variable can be used anywhere, we change it in one thread, and the other can get its updated value.

Global variables are in these places:

```
import machine
import utime
import _thread

led_red = machine.Pin(15, machine.Pin.OUT)
led_yellow = machine.Pin(14, machine.Pin.OUT)
led_green = machine.Pin(13, machine.Pin.OUT)
button = machine.Pin(16, machine.Pin.IN)

global button_status
button_status = 0


def button_thread():
    global button_status
    while True:
        if button.value() == 1:
            button_status = 1

_thread.start_new_thread(button_thread, ())

while True:
    if button_status == 1:
        led_red.value(1)
        utime.sleep(10)
    global button_status
    button_status = 0

    led_red.value(1)
    utime.sleep(5)
    led_red.value(0)

    led_yellow.value(1)
    utime.sleep(2)
    led_yellow.value(0)
```

```
led_green.value(1)
utime.sleep(5)
led_green.value(0)

led_yellow.value(1)
utime.sleep(2)
led_yellow.value(0)
```

- When the program is just running, `button_status` is assigned a value of 0, which means that the button has not been pressed.

- In the new thread–button_thread, when the program detects that the button is pressed, `button_status` is assigned the value 1.

- At the beginning of each cycle, it will detect whether the button has been pressed, if the button is pressed (button_status == 1), the red light will be on for 10 seconds. Then `button_status` switch to 0, and wait for the next button press.

The function of `global button_status` is to tell the program that we are going to modify the value of `button_status`, but if we just want to read the variable value, this line is not needed.

3.11 2.6 Two Kinds of Transistors

Transistor is a semiconductor device that controls a large current through a small current. Its function is to amplify weak signals into larger amplitude signals, and can also be used as a non-contact switch. It is the core component of electronic circuits.

This sounds a bit complicated. In simple words, some components use high-current (such as Buzzer). If the power is directly supplied from the GPIO of the microcontroller, the power may be insufficient or the microcontroller may be damaged. Then, the transistor has played a "dam" role here. Transistor receives the weak electrical signal from the GPIO pin to control the turn-on and turn-off of the large current (from VCC to GND). In this way, high-current components can be driven and the microcontroller can be protected.

Transistor

This kit is equipped with two types of transistors, S8550 and S8050, the former is PNP and the latter is NPN. They look very similar, and we need to check carefully to see their labels. When a High level signal goes through an NPN transistor, it is energized. But a PNP one needs a Low level signal to manage it. Both types of transistor are frequently used for contactless switches, just like in this experiment.
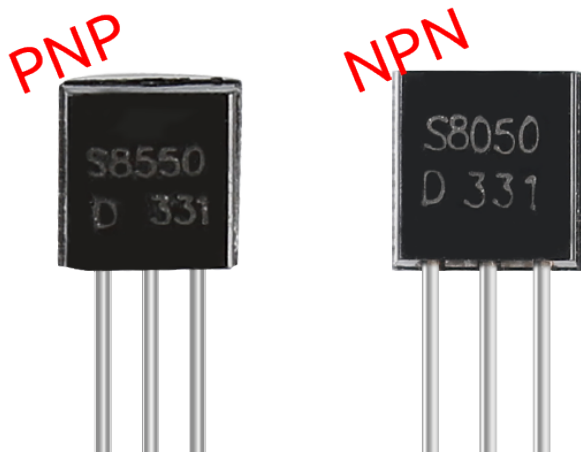
Let's use LED and button to understand how to use transistor!

3.11.1 Wiring

Put the label side facing us and the pins facing down. The pins from left to right are emitter(e), base(b), and collector(c).

Note:

- The base is the gate controller device for the larger electrical supply.

- In the NPN transistor, the collector is the larger electrical supply and the emitter is the outlet for that supply, the PNP transistor is just the opposite.

1. Way to connect NPN (S8050) transistor.
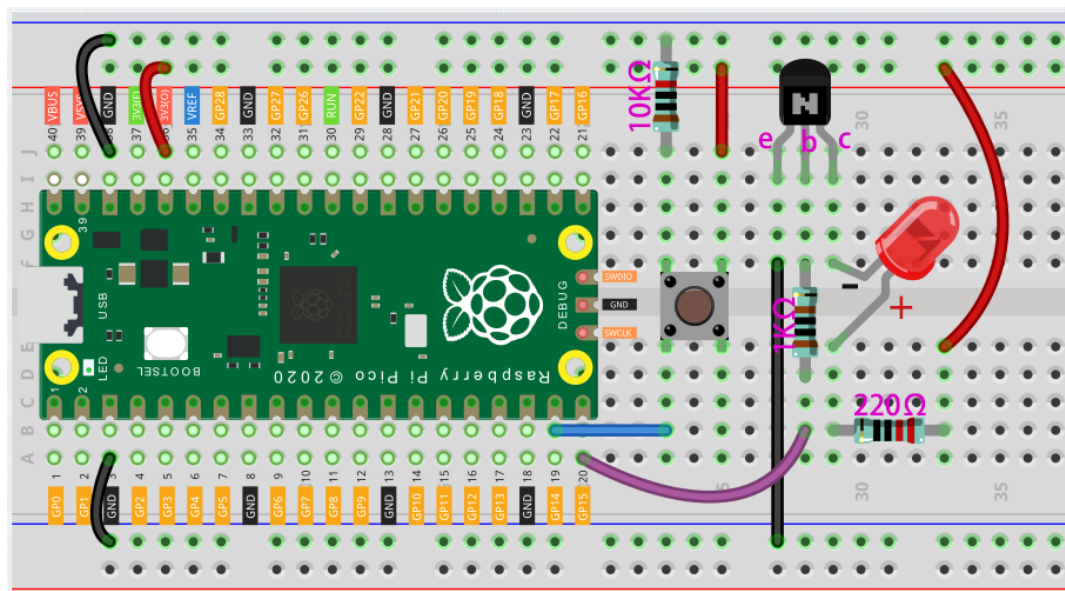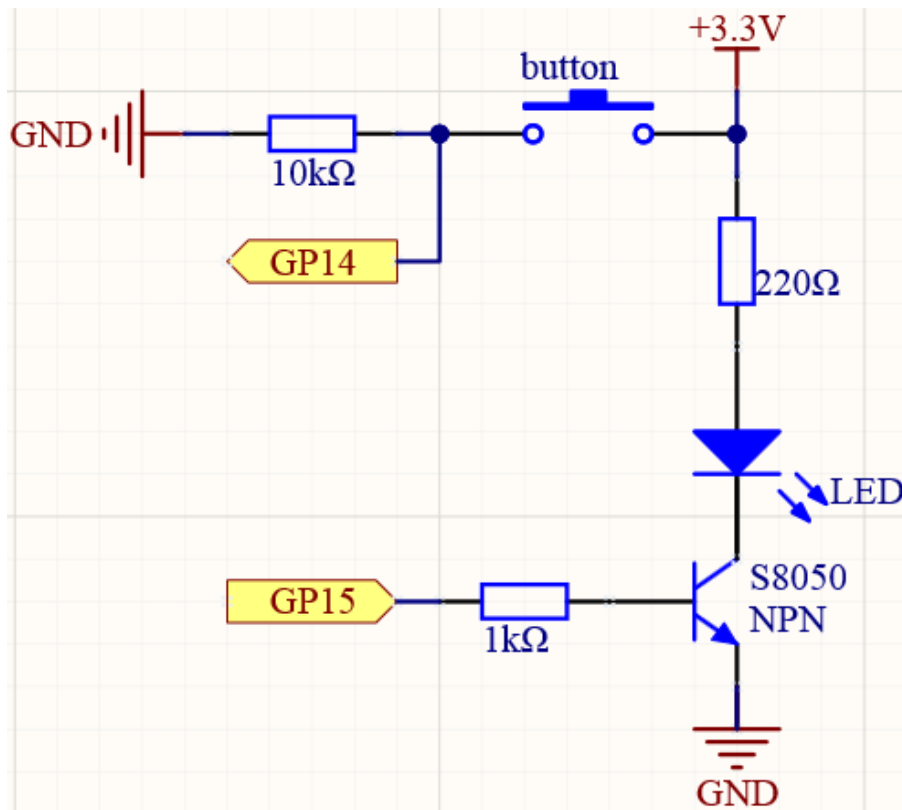
1. Connect 3V3 and GND of Pico to the power bus of the breadboard.

2. Connect the anode lead of the LED to the positive power bus via a 220 resistor.

3. Connect the cathode lead of the LED to the **collector** lead of the transistor.

4. Connect the base lead of the transistor to the GP15 pin through a 1k resistor.

5. Connect the **emitter** lead of the transistor to the negative power bus.

6. Connect one side of the button to the GP14 pin, and use a 10k resistor connect the same side and

negative power bus. The other side to the positive power bus.

Note:

- The color ring of 220 resistor is red, red, black, black and brown.
- The color ring of the 1k resistor is brown, black, black, brown and brown.
- The color ring of the 10k resistor is brown, black, black, red and brown.

---

1. Way to connect PNP(S8550) transistor.
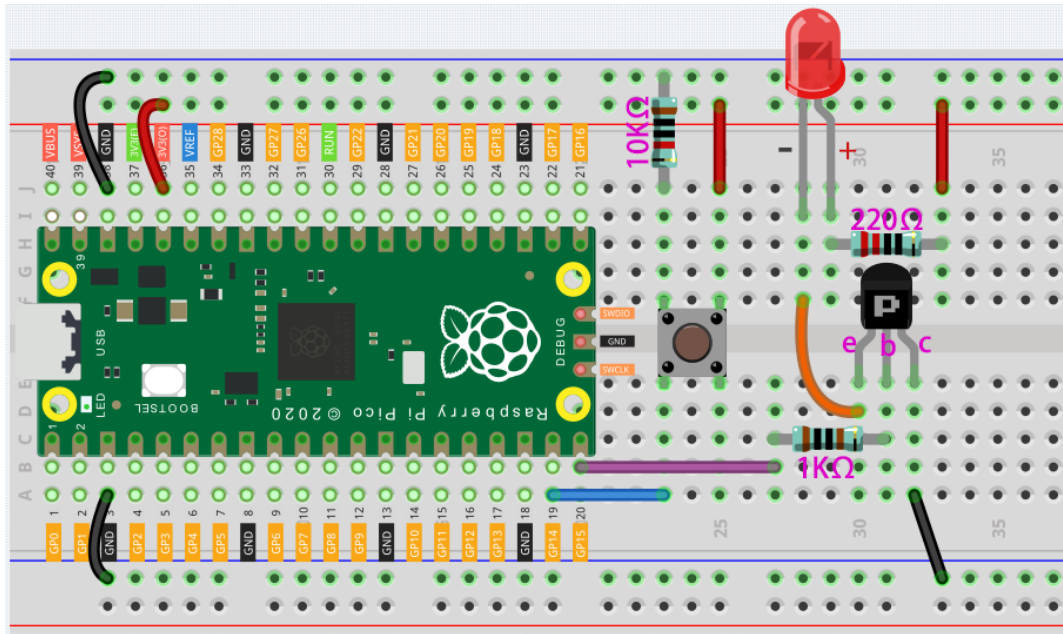
1. Connect 3V3 and GND of Pico to the power bus of the breadboard.

2. Connect the anode lead of the LED to the positive power bus via a 220 resistor.

3. Connect the cathode lead of the LED to the **emitter** lead of the transistor.

4. Connect the base lead of the transistor to the GP15 pin through a 1k resistor.

5. Connect the **collector** lead of the transistor to the negative power bus.

6. Connect one side of the button to the GP14 pin, and use a 10k resistor connect the same side and negative power bus. The other side to the positive power bus.

3.11.2 Code

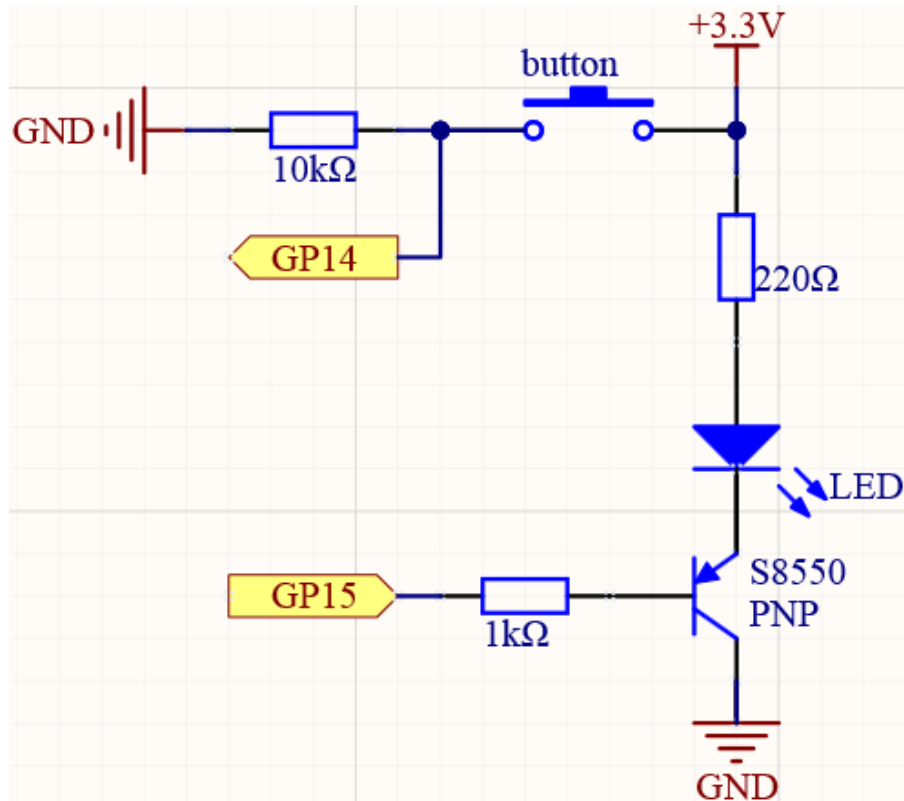Two kinds of transistors can be controlled with the same code. When we press the button, Pico will send a high-level signal to the transistor; when we release it, it will send a low-level signal. We can see that diametrically opposite phenomena have occurred in the two circuits. The circuit using the NPN transistor will light up when the button is pressed, which means it is receiving a high-level conduction circuit; The circuit that uses the PNP transistor will light up when it is released, which means it is receiving a low-level conduction circuit.

```python
import machine
button = machine.Pin(14, machine.Pin.IN)
signal = machine.Pin(15, machine.Pin.OUT)

while True:
    button_status = button.value()
    if button_status== 1:
        signal.value(1)
    elif button_status == 0:
        signal.value(0)
```

3.12 2.7 Intruder Alarm

In the previous chapters, we used simple electronic components (such as LED, button). This time we will use the sensor module - PIR.

Passive infrared sensor (PIR sensor) is a common sensor that can measure infrared (IR) light emitted by objects in its field of view. Simply put, it will receive infrared radiation emitted from the body, thereby detecting the movement of people and other animals. More specifically, it tells the main control board that someone has entered your room.

PIR Motion Sensor

Now, let's use PIR and active buzzer to build an Intruder Alarm.

3.12.1 Schematic
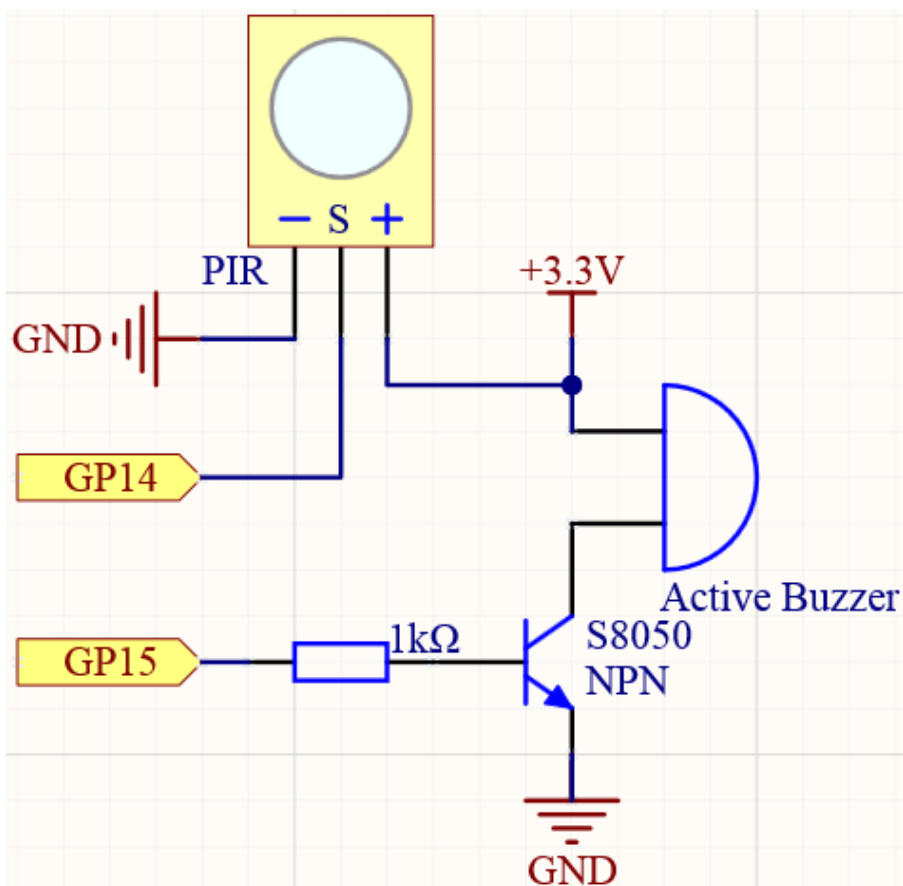
3.12.2 Wiring

Two types of buzzers are included in the kit. We need to use active buzzer. Turn them around, the sealed back (not the exposed PCB) is the one we want.

The buzzer needs to use a transistor when working, here we use S8050.
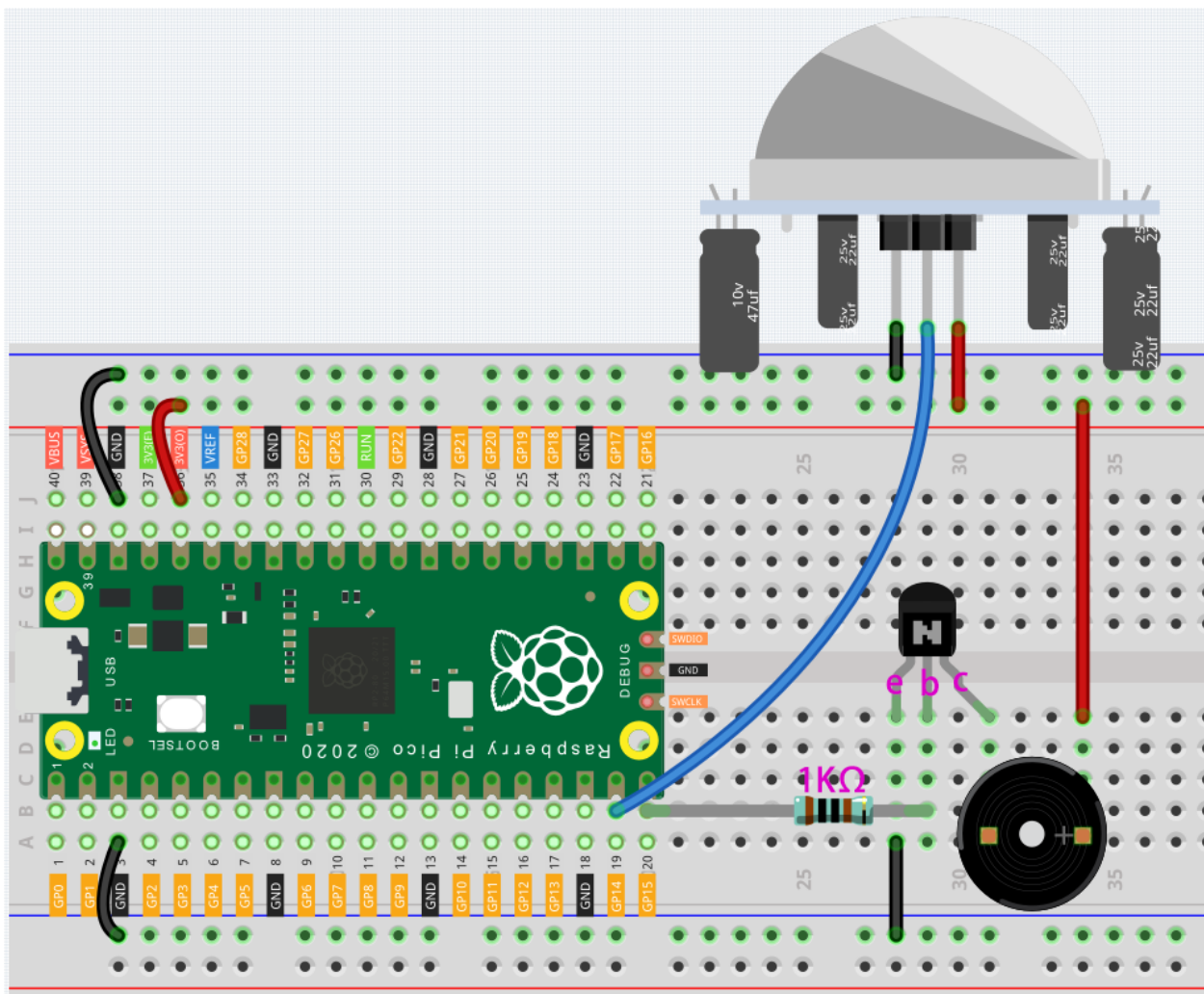
1. Connect 3V3 and GND of Pico to the power bus of the breadboard.

2. Connect the positive pin of the buzzer to the positive power bus.

3. Connect the cathode pin of the buzzer to the **collector** lead of the transistor.

4. Connect the base lead of the transistor to the GP15 pin through a 1k resistor.

5. Connect the **emitter** lead of the transistor to the negative power bus.

6. Connect the OUT of PIR to the GP14 pin, VCC to the positive power bus, and GND to the negative power bus.

Note: The color ring of the 1k resistor is brown, black, black, brown and brown.

3.12.3 Code

When the program is executed, if someone walks into the PIR detection range, the buzzer will be'BEEP BEEP' for 5 seconds!

```python
import machine
import utime

pir_sensor = machine.Pin(14, machine.Pin.IN)
buzzer = machine.Pin(15, machine.Pin.OUT)


def motion_detected(pin):
    for i in range(50):
        buzzer.toggle()
        utime.sleep_ms(100)

pir_sensor.irq(trigger=machine.Pin.IRQ_RISING, handler=motion_detected)
print("Intruder Alarm Start!")
```

3.12.4 What more?

PIR is a very sensitive sensor. In order to adapt it to the environment of use, it needs to be adjusted. Let the side with the 2 potentiometers facing you, turn both potentiometers counterclockwise to the end and insert the jumper cap on the pin with L and the middle pin.

Copy the following code into Thonny and run it, let us analyze its adjustment method along with the experimental results.

```python
import machine
import utime

pir_sensor = machine.Pin(14, machine.Pin.IN)

global timer_delay
timer_delay = utime.ticks_ms()
print("start")


def pir_in_high_level(pin):
    global timer_delay
    pir_sensor.irq(trigger=machine.Pin.IRQ_FALLING, handler=pir_in_low_level)
    intervals = utime.ticks_diff(utime.ticks_ms(), timer_delay)
```

```
    timer_delay = utime.ticks_ms()
    print("the dormancy duration is " + str(intervals) + "ms")

def pir_in_low_level(pin):
    global timer_delay
    pir_sensor.irq(trigger=machine.Pin.IRQ_RISING, handler=pir_in_high_level)
    intervals2 = utime.ticks_diff(utime.ticks_ms(), timer_delay)
    timer_delay = utime.ticks_ms()
    print("the duration of work is " + str(intervals2) + "ms")

pir_sensor.irq(trigger=machine.Pin.IRQ_RISING, handler=pir_in_high_level)
```
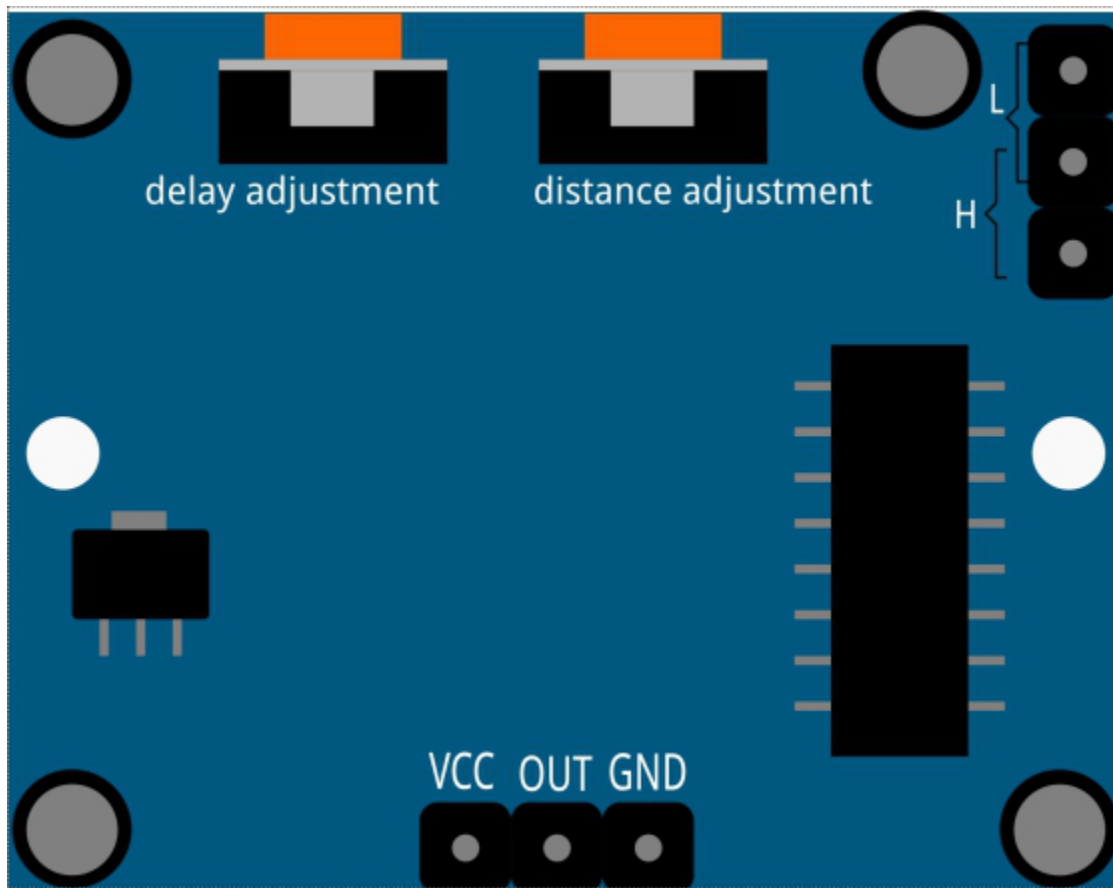

1. Trigger Mode

   Let's take a look at the pins with jumper cap at the corner. It allows PIR to enter Repeatable trigger mode or Non-repeatable trigger mode

   At present, our jumper cap connects the middle Pin and L Pin, which makes the PIR in non-repeatable trigger mode. In this mode, when the PIR detects the movement of the organism, it will send a high-level signal for about 2.8 seconds to the main control board. We can see in the printed data that the duration of work will always be around 2800ms.

   Next, we modify the position of the lower jumper cap and connect it to the middle Pin and H Pin to make the PIR in repeatable trigger mode. In this mode, when the PIR detects the movement of the organism (note that it is movement, not static in front of the sensor), as long as the organism keeps moving within the detection range, the PIR will continue to send a high-level signal to the main

control board. We can see in the printed data that the duration of work is an uncertain value.

2. Delay Adjustment

The potentiometer on the left is used to adjust the interval between two jobs.

At present, we screw it counterclockwise to the end, which makes the PIR need to enter a sleep time of about 5 seconds after finishing sending the high level work. During this time, the PIR will no longer detect the infrared radiation in the target area. We can see in the printed data that the dormancy duration is always no less than 5000ms.

If we turn the potentiometer clockwise, the sleep time will also increase. When it is turned clockwise to the end, the sleep time will be as high as 300s.

3. Distance Adjustment

The centered potentiometer is used to adjust the sensing distance range of the PIR.

Turn the knob of the distance adjustment potentiometer **clockwise** to increase the sensing distance range, and the maximum sensing distance range is about 0-7 meters. If it rotates **counterclockwise**, the sensing distance range is reduced, and the minimum sensing distance range is about 0-3 meters.

3.13 2.8 Fading LED

So far, we have used only two output signals: high level and low level (or called 1 & 0, ON & OFF), which is called digital output. However, in actual use, many devices do not simply ON/OFF to work, for example, adjusting the speed of the motor, adjusting the brightness of the desk lamp, and so on. In the past, a slider that can adjust the resistance was used to achieve this goal, but this is always unreliable and inefficient. Therefore, Pulse width modulation (PWM) has emerged as a feasible solution to such complex problems.

A digital output composed of a high level and a low level is called a pulse. The pulse width of these pins can be adjusted by changing the ON/OFF speed.

Simply put, when we are in a short period (such as 20ms, most people's visual retention time), Let the LED turn on, turn off, and turn on again, we won't see it has been turned off, but the brightness of the light will be slightly weaker. During this period, the more time the LED is turned on, the higher the brightness of the LED. In other words, in the cycle, the wider the pulse, the greater the "electric signal strength" output by the microcontroller. This is how PWM controls LED brightness (or motor speed).

- Pulse-width modulation - Wikipedia

There are some points to pay attention to when Pico uses PWM. Let's take a look at this picture.

Each GPIO pin of Pico supports PWM, but it actually has a total of 16 independent PWM outputs (instead of 30), distributed between GP0 to GP15 on the left, and the PWM output of the right GPIO is equivalent to the left copy.

What we need to pay attention to is to avoid setting the same PWM channel for different purposes during programming. (For example, GP0 and GP16 are both PWM_0A)

After understanding this knowledge, let us try to achieve the effect of Fading LED.

3.13.1 Schematic
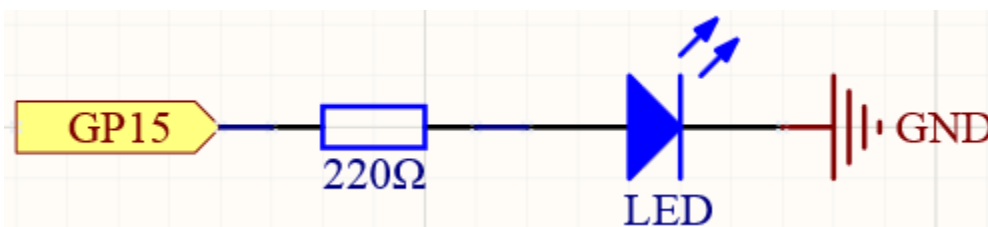
3.13.2 Wiring
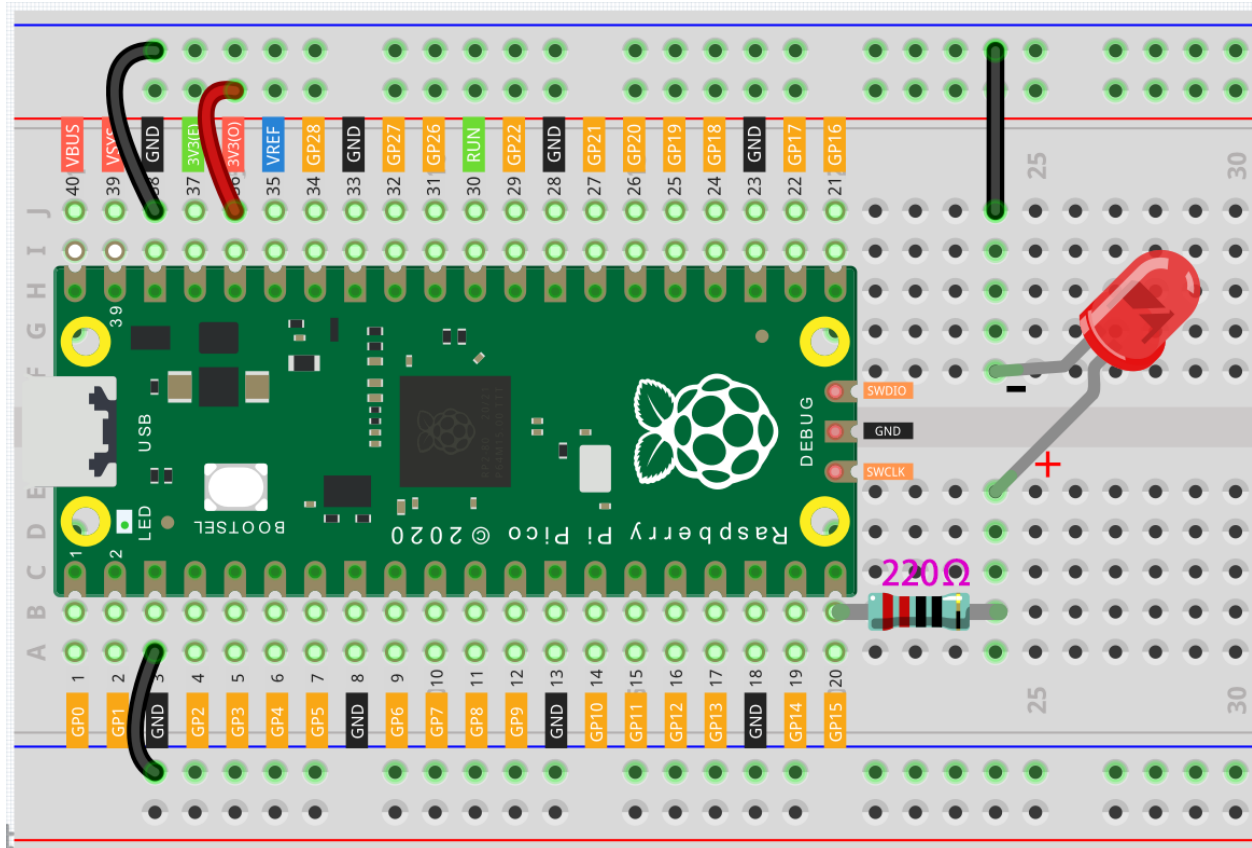
1. Here we use the GP15 pin of the Pico board.

2. Connect one end (either end) of the 220 ohm resistor to GP15, and insert the other end into the free row of the breadboard.

3. Insert the anode lead of the LED into the same row as the end of the 220 resistor, and connect the cathode lead across the middle gap of the breadboard to the same row.

4. Connect the LED cathode to the negative power bus of the breadboard.

5. Connect the negative power bus to the GND pin of Pico.

Note: The color ring of the 220 ohm resistor is red, red, black, black and brown.

3.13.3 Code

The LED will gradually become brighter as the program runs.

```python
import machine
import utime

led = machine.PWM(machine.Pin(15))
led.freq(1000)
```

```
for brightness in range(0,65535,50):
    led.duty_u16(brightness)
    utime.sleep_ms(10)
led.duty_u16(0)
```

3.13.4 How it works?

Here, we change the brightness of the LED by changing the duty cycle of the GP15's PWM output. Let's take a look at these lines.

```
import machine
import utime

led = machine.PWM(machine.Pin(15))
led.freq(1000)

for brightness in range(0,65535,50):
    led.duty_u16(brightness)
    utime.sleep_ms(10)
led.duty_u16(0)
```

- `led = machine.PWM(machine.Pin(15))` sets the GP15 pin as PWM output.

- The line `led.freq(1000)` is used to set the PWM frequency, here it is set to 1000Hz, which means 1ms (1/1000) is a cycle. The PWM frequency can be adjusted, for example, the steering wheel needs to work at 50Hz, the passive buzzer can change the tone by changing the PWM frequency. However, there is no limit when using LEDs alone, we set it to 1000Hz.

- The `led.duty_u16()` line is used to set the duty cycle, which is a 16-bit interger(2^{16}=65536). When we assign a 0 to this function, the duty cycle is 0%, and each cycle has 0% of the time to output a high level, in other words, turn off all pulses. When the value is 65535, the duty cycle is 100%, that is, the complete pulse is turned on, and the result is equal to '1' as a digital output. If it is 32768, it will turn on half a pulse, and the brightness of the LED will be half of that when it is fully turned on.

3.14 2.9 Colorful Light

As we know, light can be superimposed. For example, mix blue light and green light give cyan light, red light and green light give yellow light. This is called "The additive method of color mixing".

- Additive color - Wikipedia

Based on this method, we can use the three primary colors to mix the visible light of any color according to different specific gravity. For example, orange can be produced by more red and less green.

In this chapter, we will use RGB LED to explore the mystery of additive color mixing!

RGB LED is equivalent to encapsulating Red LED, Green LED, Blue LED under one lamp cap, and the three LEDs share one cathode pin. Since the electric signal is provided for each anode pin, the light of the corresponding color can be displayed. By changing the electrical signal intensity of each anode, it can be made to produce various colors.

3.14.1 Schematic

3.14.2 Wiring

Put the RGB LED flat on the table, we can see that it has 4 leads of different lengths. Find the longest one (GND) and turn it sideways to the left. Now, the order of the four leads is Red, GND, Green, Blue from left to right.

1. Connect the GND pin of the Pico to the negative power bus of the breadboard.

2. Insert the RGB LED into the breadboard so that its four pins are in different rows.

3. Connect the red lead to the GP13 pin via a 330 resistor. When using the same power supply intensity, the Red LED will be brighter than the other two, and a slightly larger resistor needs to be used to reduce its brightness.

4. Connect the Green lead to the GP14 pin via a 220 resistor.

5. Connect the Blue lead to the GP15 pin via a 220 resistor.

6. Connect the GND lead to the negative power bus.

7. Connect the negative power bus to Pico's GND.

Note:

- The color ring of the 220 resistor is red, red, black, black and brown.

- The color ring of the 330 resistor is orange, orange, black, black and brown.

3.14.3 Code

Here, we can choose our favorite color in drawing software (such as paint) and display it with RGB LED.

```python
import machine
import utime

red = machine.PWM(machine.Pin(13))
green = machine.PWM(machine.Pin(14))
blue = machine.PWM(machine.Pin(15))
red.freq(1000)
green.freq(1000)
blue.freq(1000)

def interval_mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def color_to_duty(rgb_value):
    rgb_value = int(interval_mapping(rgb_value,0,255,0,65535))
    return rgb_value

def color_set(red_value,green_value,blue_value):
    red.duty_u16(color_to_duty(red_value))
    green.duty_u16(color_to_duty(green_value))
    blue.duty_u16(color_to_duty(blue_value))

color_set(255,128,0)
```
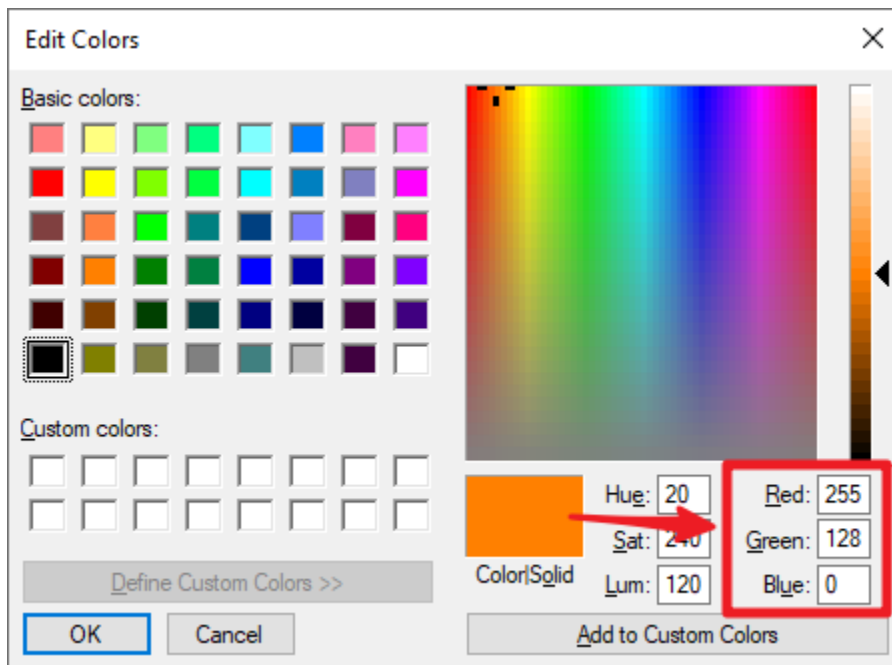

Write the RGB value into color_set(), you will be able to see the RGB light up the colors you want.

3.14.4 How it works?

We defined a `color_set()` function to let the three primary colors work together.

At present, pixels in computer hardware usually adopt a 24-bit representation method. The three primary colors are divided into 8 bits, and the color value range is 0 to 255. With 256 possible values for each of the three primary colors (don't forget to count 0!), that 256 x 256 x 256 = 16,777,216 colors can be combined in this way. The `color_set()` function also follows the 24-bit notation, which makes it easier for us to select the desired color.

And since the value range of `duty_u16()` is 0~65535 (instead of 0 to 255) when the output signals to RGB LED through PWM, we have defined `color_to_duty()` and `interval_mapping ()` function to map the color values to the duty values.

3.15 2.10 Custom Tone

We have used active buzzer in the previous project, this time we will use passive buzzer.

Like the active buzzer, the passive buzzer also uses the phenomenon of electromagnetic induction to work. The difference is that a passive buzzer does not have oscillating source, so it will not beep if DC signals are used. But this allows the passive buzzer to adjust its own oscillation frequency and can emit different notes such as "doh, re, mi, fa, sol, la, ti".

Let the passive buzzer emit a melody!

3.15.1 Schematic
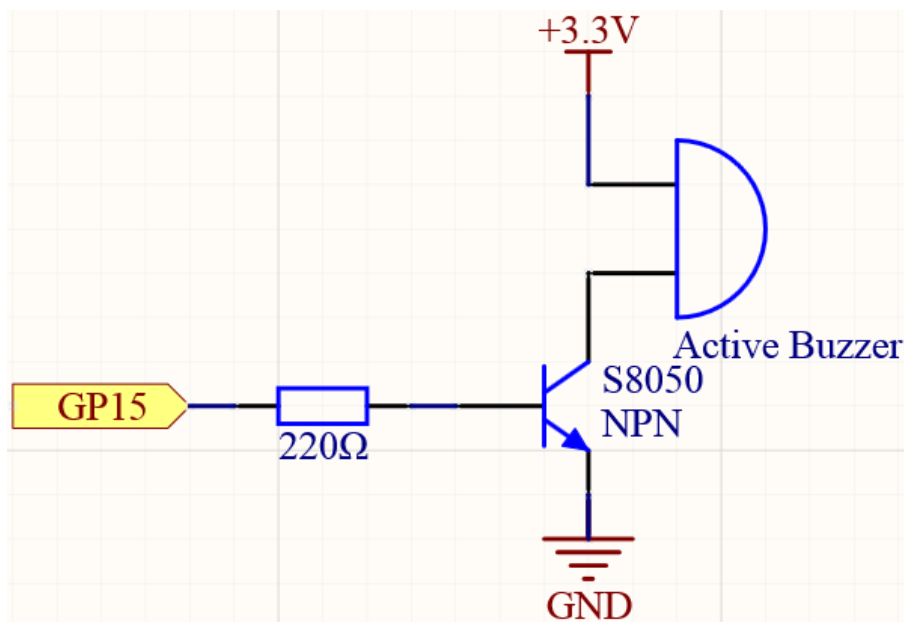
3.15.2 Wiring

Two buzzers are included in the kit, we use the one with exposed PCB behind.

The buzzer needs a transistor to work, and here we use S8050.
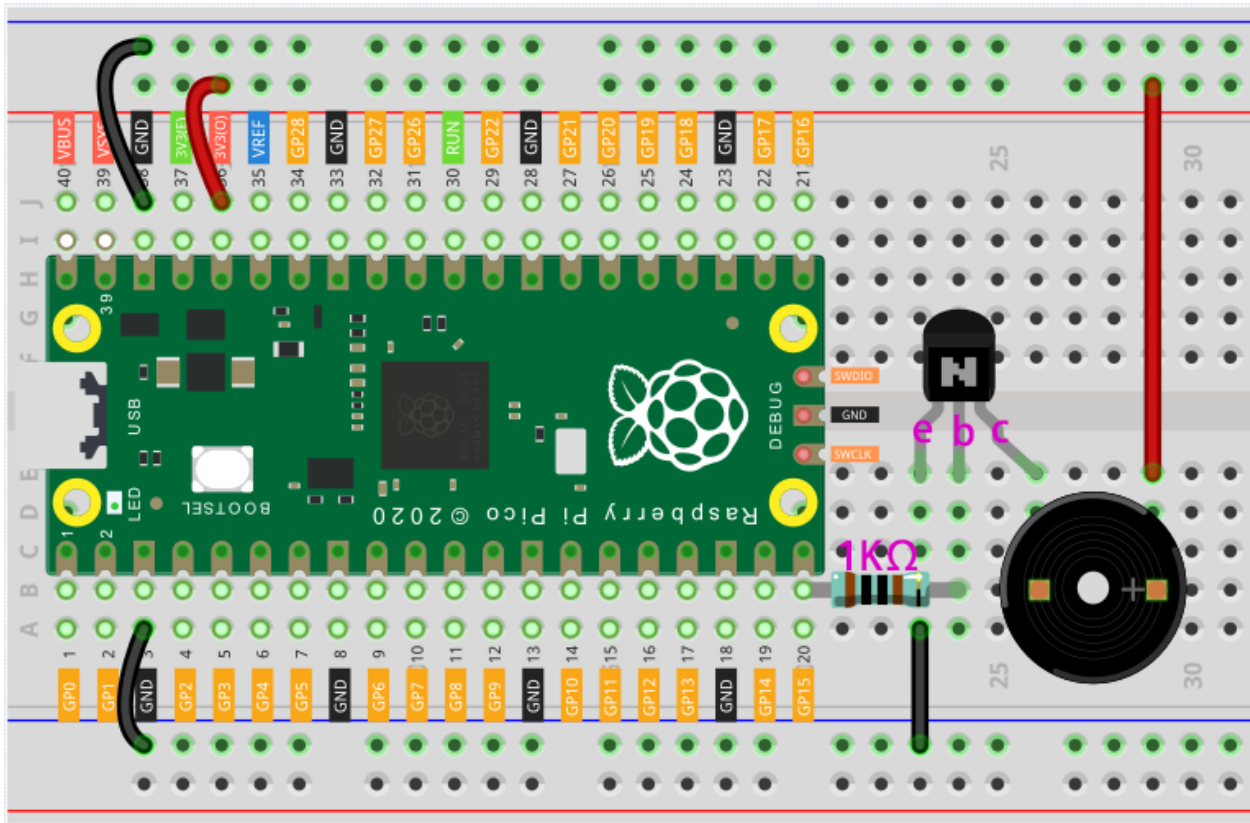
1. Connect 3V3 and GND of Pico to the power bus of the breadboard.

2. Connect the positive pin of the buzzer to the positive power bus.

3. Connect the cathode pin of the buzzer to the **collector** lead of the transistor.

4. Connect the **base** lead of the transistor to the GP15 pin through a 1k resistor.

5. Connect the **emitter** lead of the transistor to the negative power bus.

Note: The color ring of the 1k ohm resistor is brown, black, black, brown and brown.

3.15.3 Code

```python
import machine
import utime

buzzer = machine.PWM(machine.Pin(15))

def tone(pin,frequency,duration):
    pin.freq(frequency)
    pin.duty_u16(30000)
    utime.sleep_ms(duration)
    pin.duty_u16(0)

tone(buzzer,440,250)
utime.sleep_ms(500)
tone(buzzer,494,250)
utime.sleep_ms(500)
tone(buzzer,523,250)
```

3.15.4 How it works?

If the passive buzzer given a digital signal, it can only keep pushing the diaphragm without producing sound.

Therefore, we use the `tone()` function to generate the PWM signal to make the passive buzzer sound.

This function has three parameters:

- **pin**, the GPIO pin that controls the buzzer.
- **frequency**, the pitch of the buzzer is determined by the frequency, the higher the frequency, the higher the pitch.
- **Duration**, the duration of the tone.

We use the `duty_u16()` function to set the duty cycle to 30000(about 50%). It can be other numbers, and it only needs to generate a discontinuous electrical signal to oscillate.

3.15.5 What more

We can simulate the specific tone according to the fundamental frequency of the piano, so as to play a complete piece of music.

- Piano key frequencies - Wikipedia

```python
import machine
import utime

NOTE_C4 = 262
NOTE_G3 = 196
NOTE_A3 = 220
NOTE_B3 = 247
```

```
melody =[NOTE_C4,NOTE_G3,NOTE_G3,NOTE_A3,NOTE_G3,NOTE_B3,NOTE_C4]

buzzer = machine.PWM(machine.Pin(15))

def tone(pin,frequency,duration):
    pin.freq(frequency)
    pin.duty_u16(30000)
    utime.sleep_ms(duration)
    pin.duty_u16(0)

for note in melody:
    tone(buzzer,note,250)
    utime.sleep_ms(150)
```

3.16 2.11 Swinging Servo

In this kit, in addition to LED and passive buzzer, there is also a device controlled by PWM signal, Servo.

Servo is a position (angle) servo device, which is suitable for those control systems that require constant angle changes and can be maintained. It has been widely used in high-end remote control toys, such as airplanes, submarine models, and remote control robots.

Now, try to make the servo sway!

3.16.1 Schematic
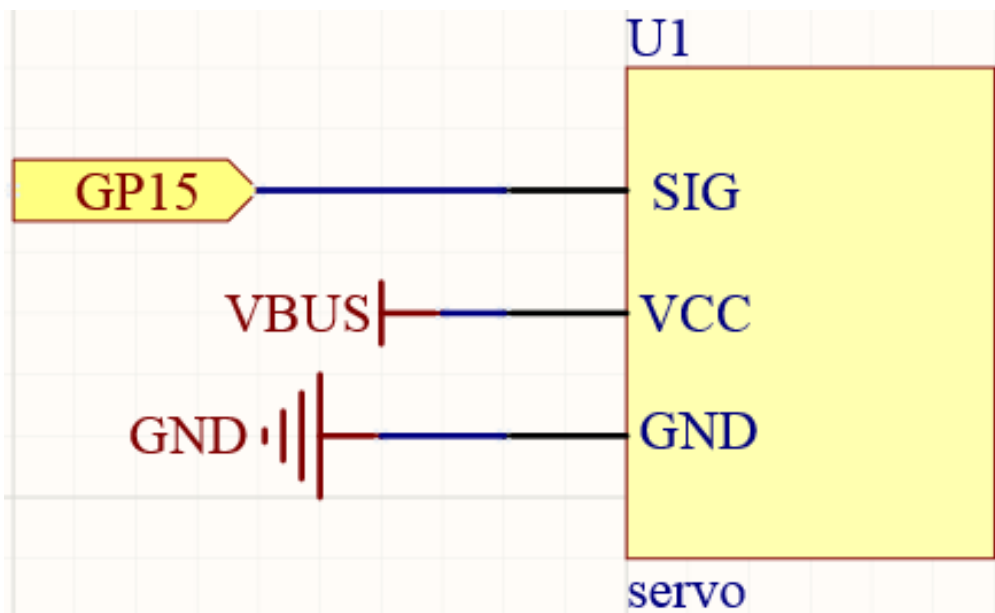
3.16.2 Wiring
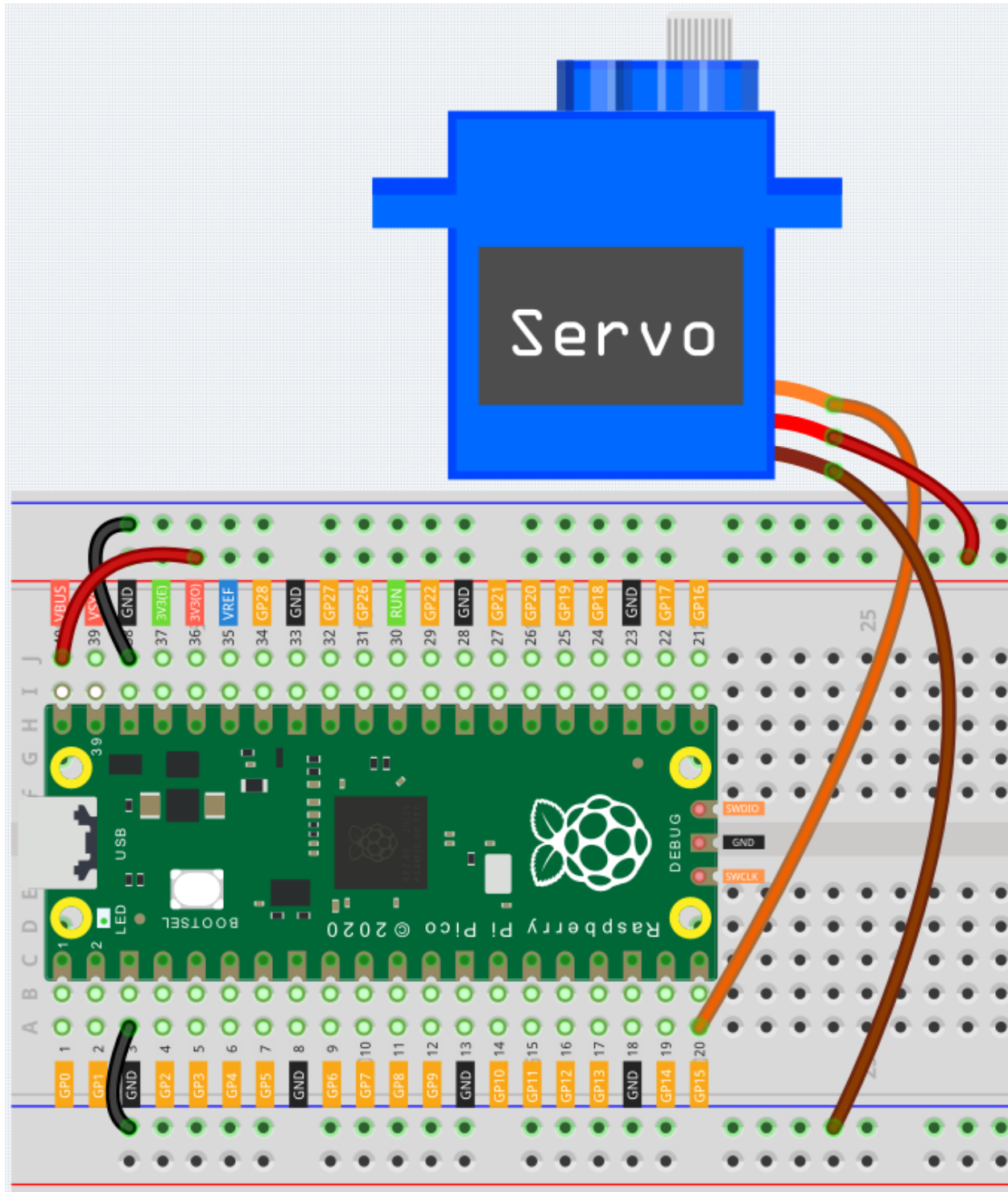
1. Press the Servo Arm into the Servo output shaft. If necessary, fix it with screws.

2. Connect **VBUS** (not 3V3) and GND of Pico to the power bus of the breadboard.

3. Connect the red lead of the servo to the positive power bus with a jumper.

4. Connect the yellow lead of the servo to the GP15 pin with a jumper wire.

5. Connect the brawn lead of the servo to the negative power bus with a jumper wire.

3.16.3 Code

When the program is running, we can see the Servo Arm swinging back and forth from 0° to 180°. The program will always run because of the `while True` loop, we need to press the Stop key to end the program.

```python
import machine
import utime

servo = machine.PWM(machine.Pin(15))
servo.freq(50)

def interval_mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def servo_write(pin,angle):
    pulse_width=interval_mapping(angle, 0, 180, 0.5,2.5)
    duty=int(interval_mapping(pulse_width, 0, 20, 0,65535))
    pin.duty_u16(duty)

while True:
    for angle in range(180):
        servo_write(servo,angle)
        utime.sleep_ms(20)
    for angle in range(180,-1,-1):
        servo_write(servo,angle)
        utime.sleep_ms(20)
```

3.16.4 How it works?

We defined the `servo_write()` function to make the servo run.

This function has two parameters:

- pin, the GPIO pin that controls the servo.
- Angle, the angle of the shaft output.

In this function, `interval_mapping()` is called to map the angle range 0 ~ 180 to the pulse width range 0.5 ~ 2.5ms.

```python
pulse_width=interval_mapping(angle, 0, 180, 0.5,2.5)
```

Why is it 0.5~2.5? This is determined by the working mode of the Servo.

Servo

Next, convert the pulse width from period to duty. Since duty_u16() cannot have decimals when used (the value cannot be a float type), we used `int()` to force the duty to be converted to an int type.

```python
duty=int(interval_mapping(pulse_width, 0, 20, 0,65535))
```

Finally, write the duty value into duty_u16().

3.17 2.12 Turn the Knob

In the previous projects, we have used the digital input on the Pico. For example, a button can change the pin from low level (off) to high level (on). This is a binary working state.

However, Pico can receive another type of input signal: analog input. It can be in any state from fully closed to fully open, and has a range of possible values. The analog input allows the microcontroller to sense the light intensity, sound intensity, temperature, humidity, etc. of the physical world.

Usually, a microcontroller needs an additional hardware to implement analog input-the analogue-to-digital converter (ADC). But Pico itself has a built-in ADC for us to use directly.

Pico has three GPIO pins that can use analog input, GP26, GP27, GP28. That is, analog channels 0, 1, and 2. In addition, there is a fourth analog channel, which is connected to the built-in temperature sensor and will not be introduced here.

In this project, we try to read the analog value of potentiometer.

3.17.1 Schematic
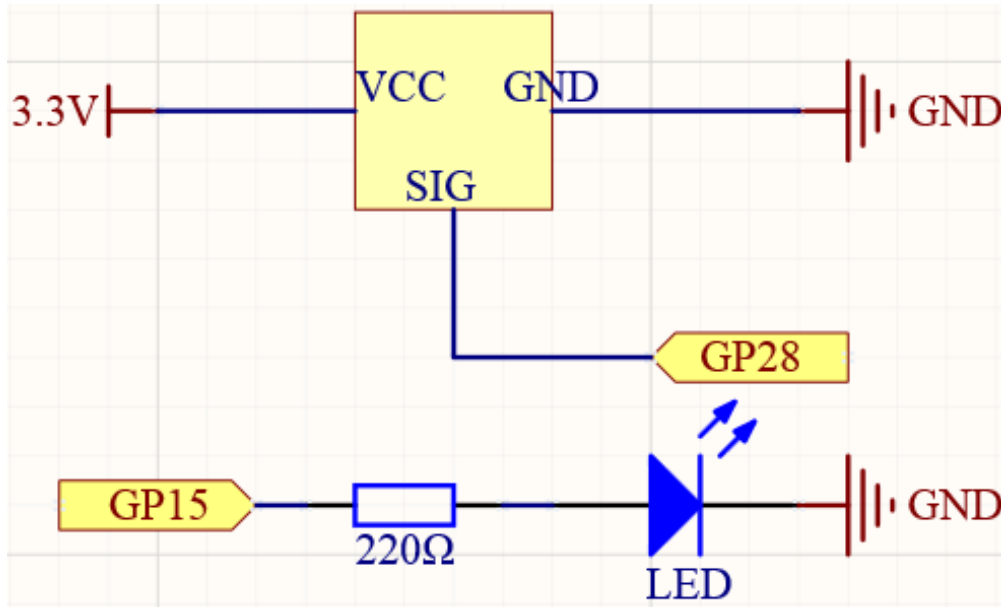
3.17.2 Wiring
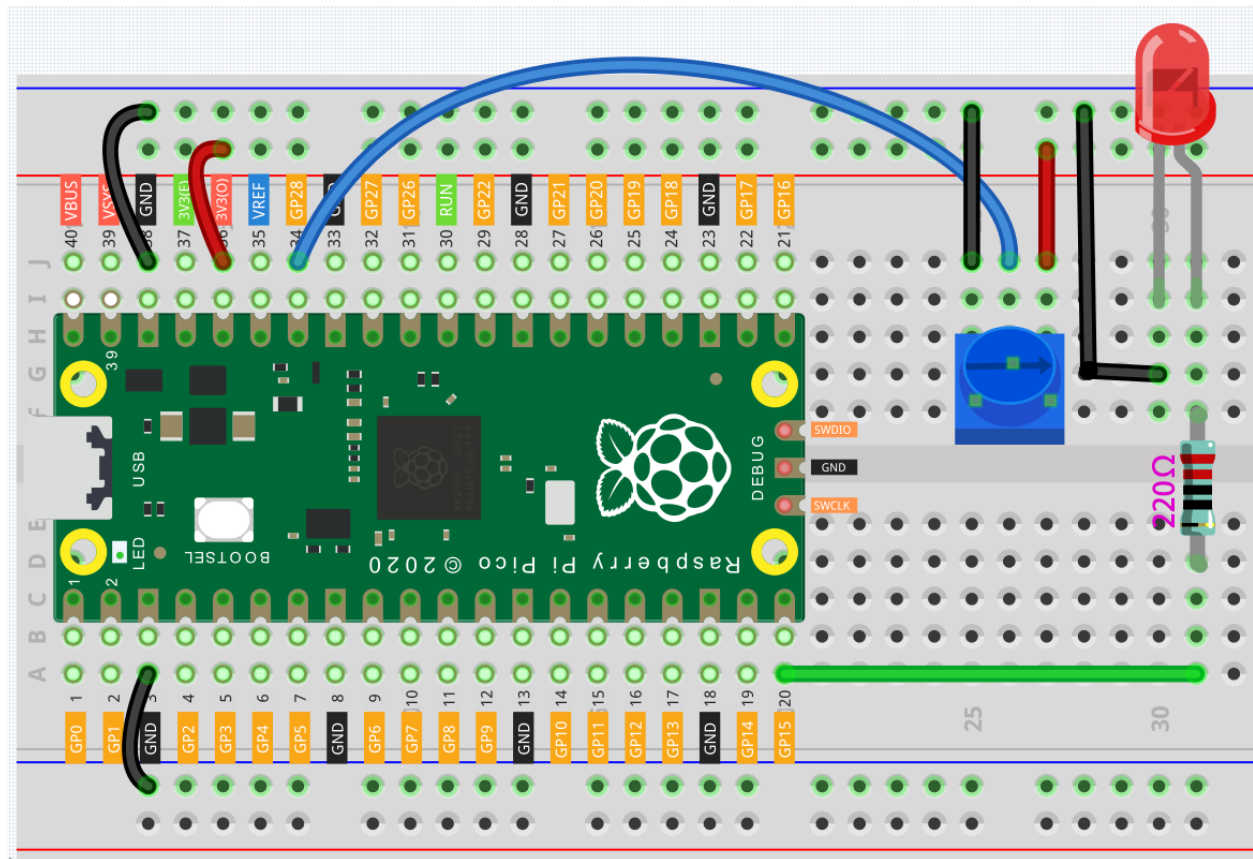
1. Connect 3V3 and GND of Pico to the power bus of the breadboard.

2. Insert the potentiometer into the breadboard, its three pins should be in different rows.

3. Use jumper wires to connect the pins on both sides of the potentiometer to the positive and negative power bus respectively.

4. Connect the middle pin of the potentiometer to GP28 with a jumper wire.

5. Connect the anode of the LED to the GP15 pin through a 220 resistor, and connect the cathode to the negative power bus.

3.17.3 Code

When the program is running, we can see the analog value currently read by the GP28 pin in the shell. Turn the knob, and the value will change from 0 to 65535. At the same time, the brightness of the LED will increase as the analog value increases.

```python
import machine
import utime

potentiometer = machine.ADC(28)
led = machine.PWM(machine.Pin(15))
led.freq(1000)

while True:
    value=potentiometer.read_u16()
    print(value)
    led.duty_u16(value)
    utime.sleep_ms(200)
```

3.17.4 What more?

Let's use the potentiometer to swing the servo from left to right!
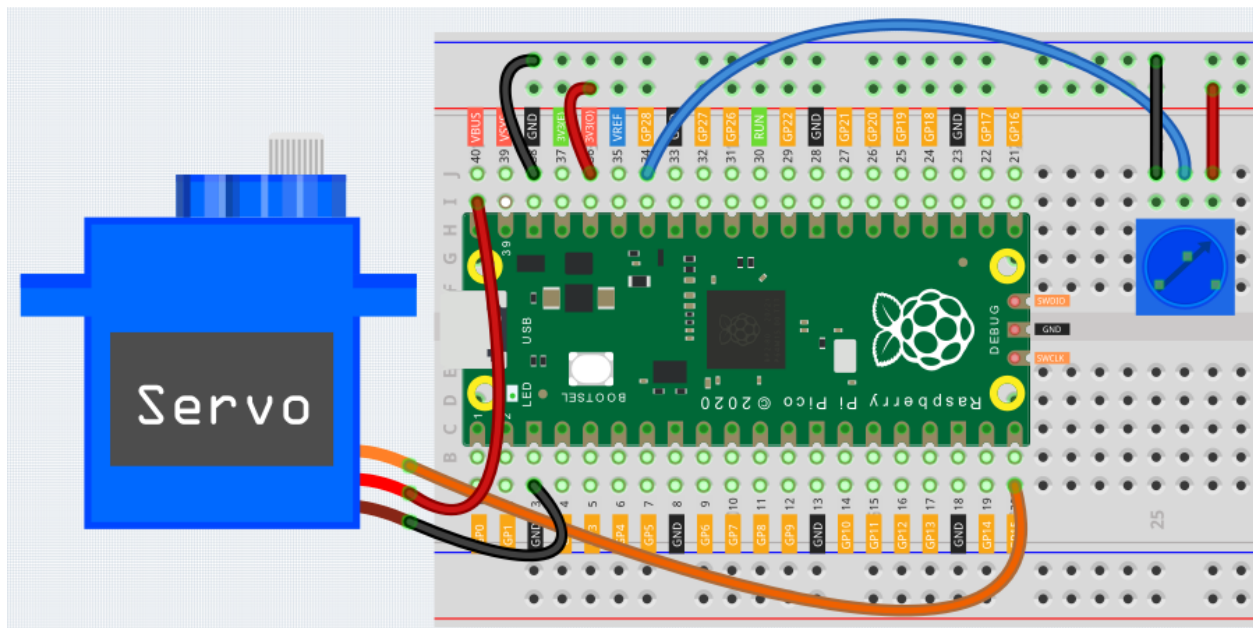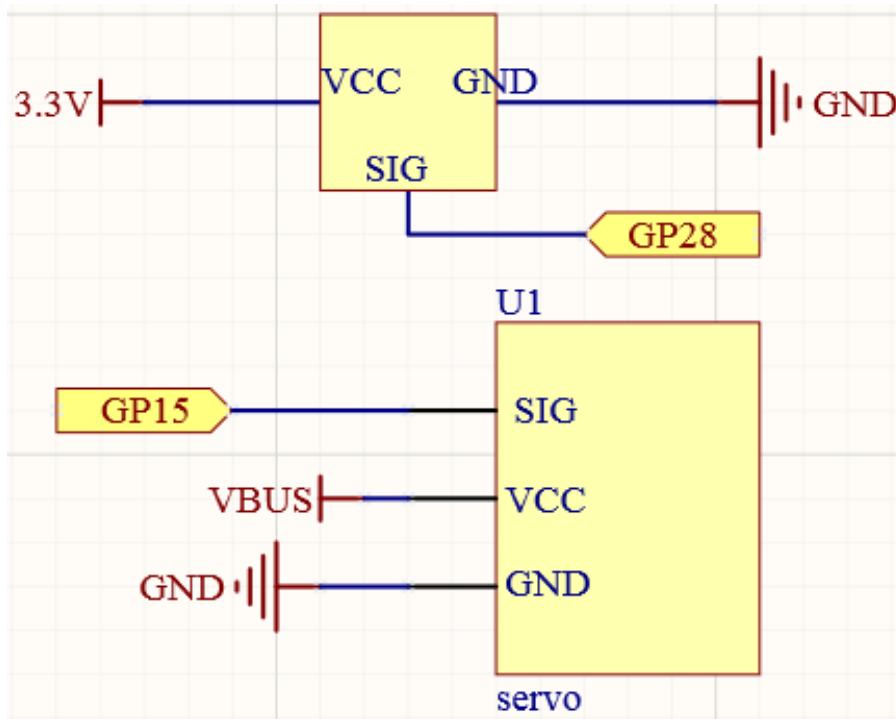
```python
import machine
import utime

potentiometer = machine.ADC(28)
servo = machine.PWM(machine.Pin(15))
servo.freq(50)

def interval_mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
```

```python
def servo_write(pin,angle):
    pulse_width=interval_mapping(angle, 0, 180, 0.5,2.5)
    duty=int(interval_mapping(pulse_width, 0, 20, 0,65535))
    pin.duty_u16(duty)

while True:
    value=potentiometer.read_u16()
    angle=interval_mapping(value,0,65535,0,180)
    servo_write(servo,angle)
    utime.sleep_ms(200)
```

3.18 2.13 Thermometer

A thermistor is a type of resistor whose resistance is strongly dependent on temperature, and it has two types: Negative Temperature Coefficient (NTC) and Positive Temperature Coefficient (PTC), also known as NTC and PTC. The resistance of PTC thermistor increases with temperature, while the condition of NTC is opposite to the former.

In this experiment we use an NTC thermistor to make a thermometer.

3.18.1 Schematic

3.18.2 Wiring

1. Connect 3V3 and GND of Pico to the power bus of the breadboard.

2. Connect one lead of the thermistor to the GP28 pin, then connect the same lead to the positive power bus with a 10K ohm resistor.

3. Connect another lead of thermistor to the negative power bus.

Note:

- The thermistor is black and marked 103.

- The color ring of the 10K ohm resistor is red, black, black, red and brown.

```python
import machine
import utime
import math

thermistor = machine.ADC(28)

while True:
    temperature_value = thermistor.read_u16()
    Vr = 3.3 * float(temperature_value) / 65535
    Rt = 10000 * Vr / (3.3 - Vr)
    temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
    Cel = temp - 273.15
    Fah = Cel * 1.8 + 32
```

```
    print ('Celsius: %.2f C  Fahrenheit: %.2f F' % (Cel, Fah))
    utime.sleep_ms(200)
```

3.18.3 How it works?

Each thermistor has a normal resistance. Here it is 10k ohm, which is measured under 25 degree Celsius. When the temperature gets higher, the resistance of the thermistor decreases. Then the voltage data is converted to digital quantities by the A/D adapter.

The temperature in Celsius or Fahrenheit is output via programming.

```
import math
```

There is a numerics library which declares a set of functions to compute common mathematical operations and transformations.

- math

```
temperature_value = thermistor.read_u16()
```

This function is used to read the value of the thermistor.

```
Vr = 3.3 * float(temperature_value) / 65535
Rt = 10000 * Vr / (3.3 - Vr)
temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
Cel = temp - 273.15
Fah = Cel * 1.8 + 32
print ('Celsius: %.2f C  Fahrenheit: %.2f F' % (Cel, Fah))
utime.sleep_ms(200)
```

These calculations convert the thermistor values into centigrade degree and Fahrenheit degree.

```
Vr = 3.3 * float(temperature_value) / 65535
Rt = 10000 * Vr / (3.3 - Vr)
```

In the two lines of code above, the voltage is first calculated using the read analoge value, and then get Rt (the resistance of the thermistor).

```
temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
```

Note: Here is the relation between the resistance and temperature:

RT =RN expB(1/TK – 1/TN)

- RT is the resistance of the NTC thermistor when the temperature is TK.

- RN is the resistance of the NTC thermistor under the rated temperature TN. Here, the numerical value of RN is 10k.

- TK is a Kelvin temperature and the unit is K. Here, the numerical value of TK is 273.15 + degree Celsius.

- TN is a rated Kelvin temperature; the unit is K too. Here, the numerical value of TN is 273.15+25.

- And B(beta), the material constant of NTC thermistor, is also called heat sensitivity index with a numerical value 3950.

- exp is the abbreviation of exponential, and the base number e is a natural number and equals 2.7 approximately.

Convert this formula TK=1/(ln(RT/RN)/B+1/TN) to get Kelvin temperature that minus 273.15 equals degree Celsius.

This relation is an empirical formula. It is accurate only when the temperature and resistance are within the effective range.

This code refers to plugging Rt into the formula TK=1/(ln(RT/RN)/B+1/TN) to get Kelvin temperature.

```
temp = temp - 273.15
```

Convert Kelvin temperature into centigrade degree.

```
Fah = Cel * 1.8 + 32
```

Convert the centigrade degree into Fahrenheit degree.

```
print ('Celsius: %.2f °C Fahrenheit: %.2f ' % (Cel, Fah))
```

Print centigrade degree, Fahrenheit degree and their units in the shell.
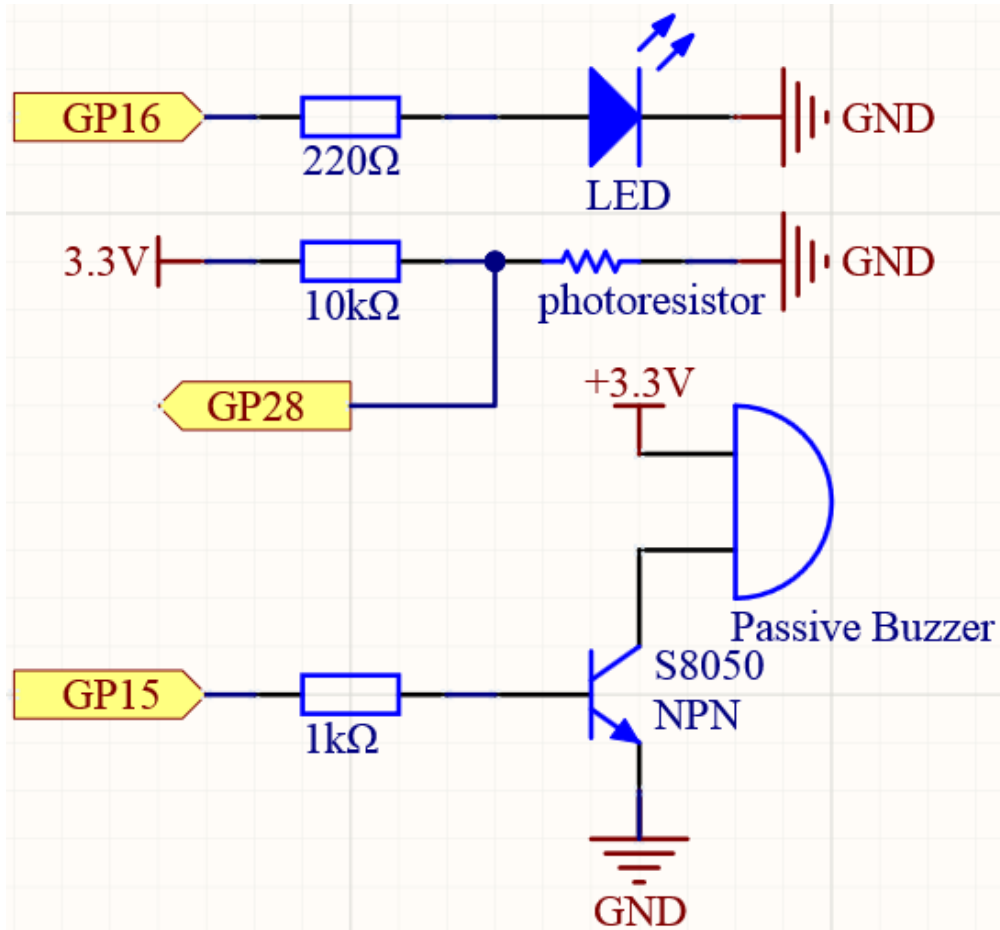
3.19 2.14 Light Theremin

Theremin is an electronic musical instrument that does not require physical contact. It produces different tones by sensing the position of the player's hand.

The instrument's controlling section usually consists of two metal antennas that sense the relative position of the thereminist's hands and control oscillators for frequency with one hand, and amplitude (volume) with the other. The electric signals from the theremin are amplified and sent to a loudspeaker.
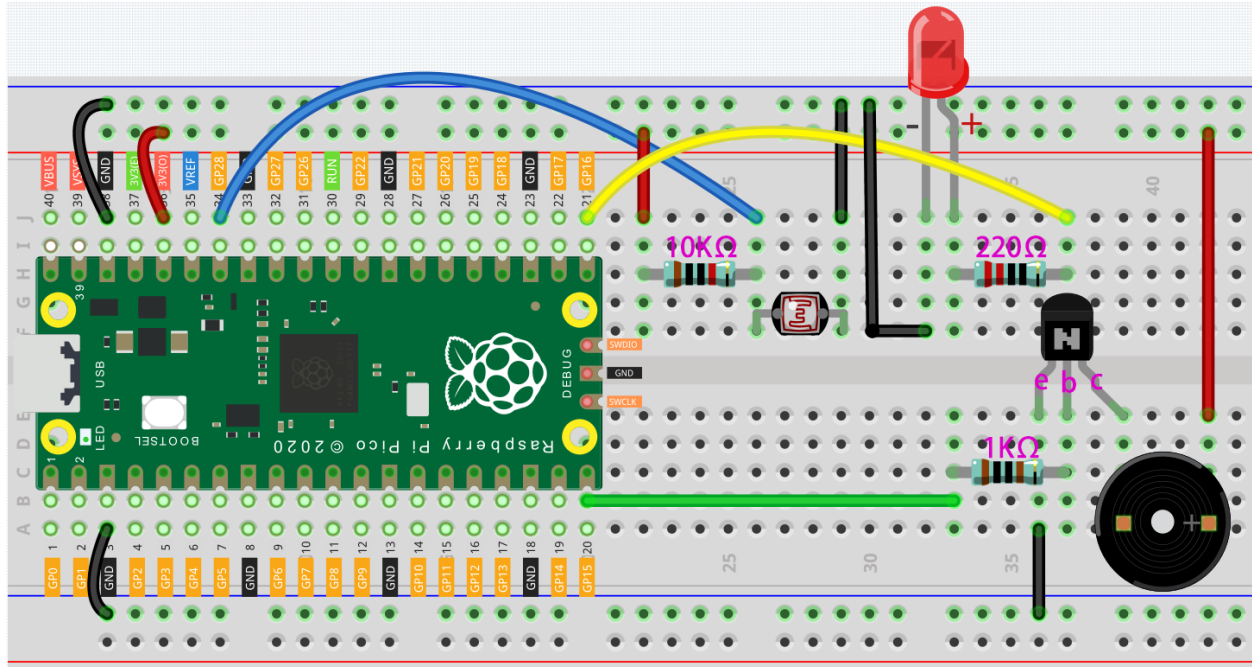
We cannot reproduce the same instrument through Pico, but we can use photoresistor and passive buzzer to achieve similar gameplay.

- Theremin - Wikipedia

3.19.1 Schematic

3.19.2 Wiring

1. Connect 3V3 and GND of Pico to the power bus of the breadboard.

2. Connect one lead of the photoresistor to the GP28 pin, then connect the same lead to the positive power bus with a 10K ohm resistor.

3. Connect another lead of photoresistor to the negative power bus.

4. Insert the LED into the breadboard, connect its anode pin to the GP16 in series with a 220 resistor, and connect its cathode pin to the negative power bus.

5. Insert the passive buzzer and S8050 transistor into the breadboard. The anode pin of the buzzer is connected to the positive power bus, the cathode pin is connected to the **collector** lead of the transistor, and the **base** lead of the transistor is connected to the GP15 pin through a 1k resistor. **emitter** lead is connected to the negative power bus.

Note:

- The color ring of the 22 resistor is red, red, black, black and brown.

- The color ring of the 10k resistor is brown, black, black, red and brown.

3.19.3 Code

```python
import machine
import utime

led = machine.Pin(16, machine.Pin.OUT)
photoresistor = machine.ADC(28)
buzzer = machine.PWM(machine.Pin(15))

light_low=65535
light_high=0

def interval_mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def tone(pin,frequency,duration):
    pin.freq(frequency)
    pin.duty_u16(30000)
    utime.sleep_ms(duration)
    pin.duty_u16(0)

# calibrate the photoresistor max & min values.
timer_init_start = utime.ticks_ms()
led.value(1)
while utime.ticks_diff(utime.ticks_ms(), timer_init_start)<5000:
    light_value = photoresistor.read_u16()
    if light_value > light_high:
        light_high = light_value
    if light_value < light_low:
        light_low = light_value
led.value(0)

# play
while True:
    light_value  = photoresistor.read_u16()
    pitch = int(interval_mapping(light_value,light_low,light_high,50,6000))
    if pitch > 50 :
        tone(buzzer,pitch,20)
    utime.sleep_ms(10)
```

When the program runs, the LED will light up, and we will have five seconds to calibrate the detection range of the photoresistor. This is because we may be in a different light environment each time we use it (e.g. the light intensity is different between midday and dusk).

At this time, we need to swing our hands up and down on top of the photoresistor, and the movement range of the hand will be calibrated to the playing range of this instrument.

After five seconds, the LED will go out and we can wave our hands on the photoresistor to play.

3.20 2.15 Microchip - 74HC595

Integrated circuit (integrated circuit) is a kind of miniature electronic device or component, which is represented by the letter "IC" in the circuit.

A certain process is used to interconnect the transistors, resistors, capacitors, inductors and other components and wiring required in a circuit, fabricate on a small or several small semiconductor wafers or dielectric substrates, and then package them in a package , it has become a micro-structure with the required circuit functions; all of the components have been structured as a whole, making electronic components a big step towards micro-miniaturization, low power consumption, intelligence and high reliability.

The inventors of integrated circuits are Jack Kilby (integrated circuits based on germanium (Ge)) and Robert Norton Noyce (integrated circuits based on silicon (Si)).

This kit is equipped with an IC, 74HC595, which can greatly save the use of GPIO pins. Specifically, it can replace 8 pins for digital signal output by writing an 8-bit binary number.
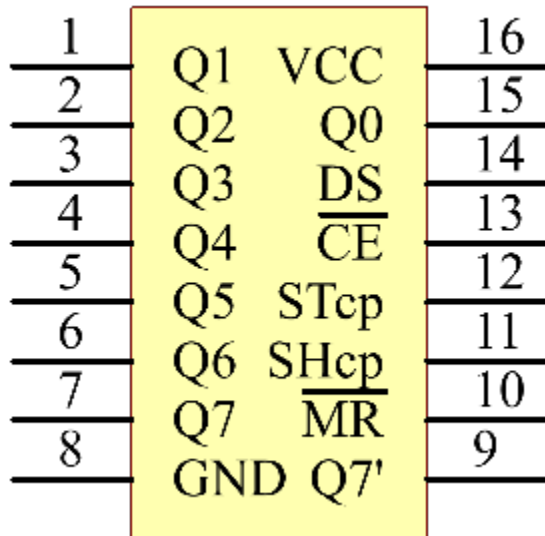
- Binary number - Wikipedia

- 74HC595
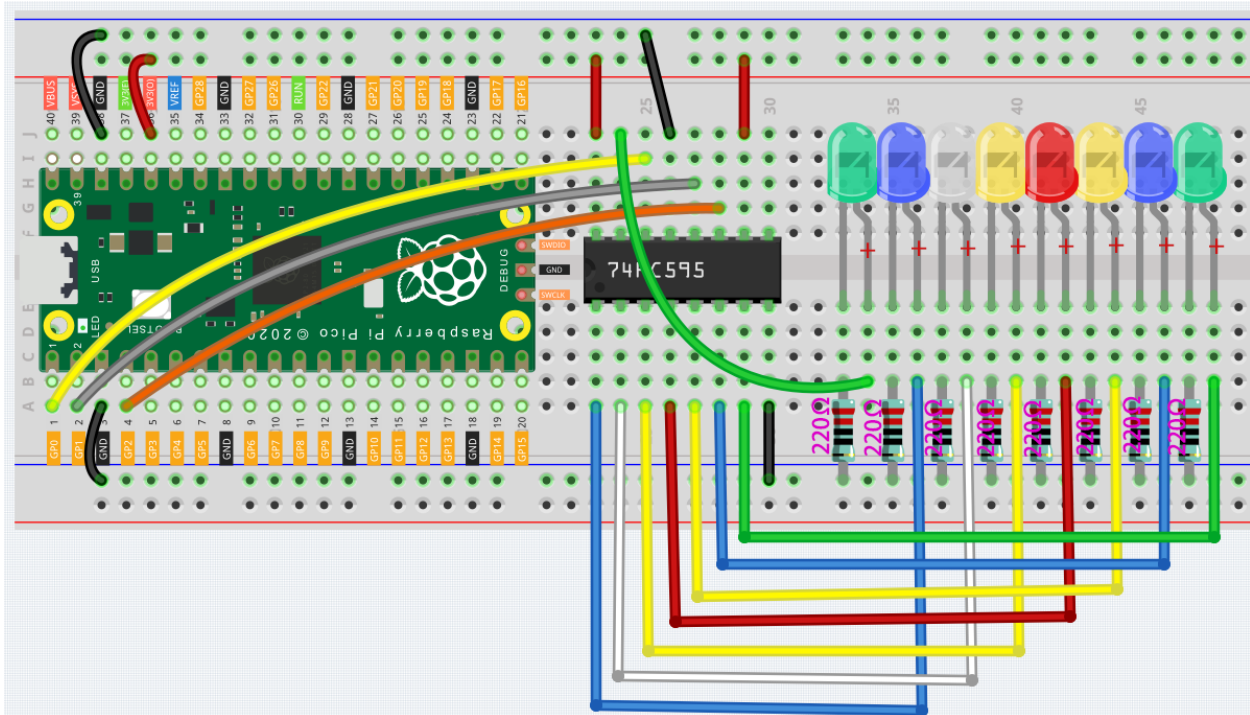
Let's use it.

3.20.1 Schematic
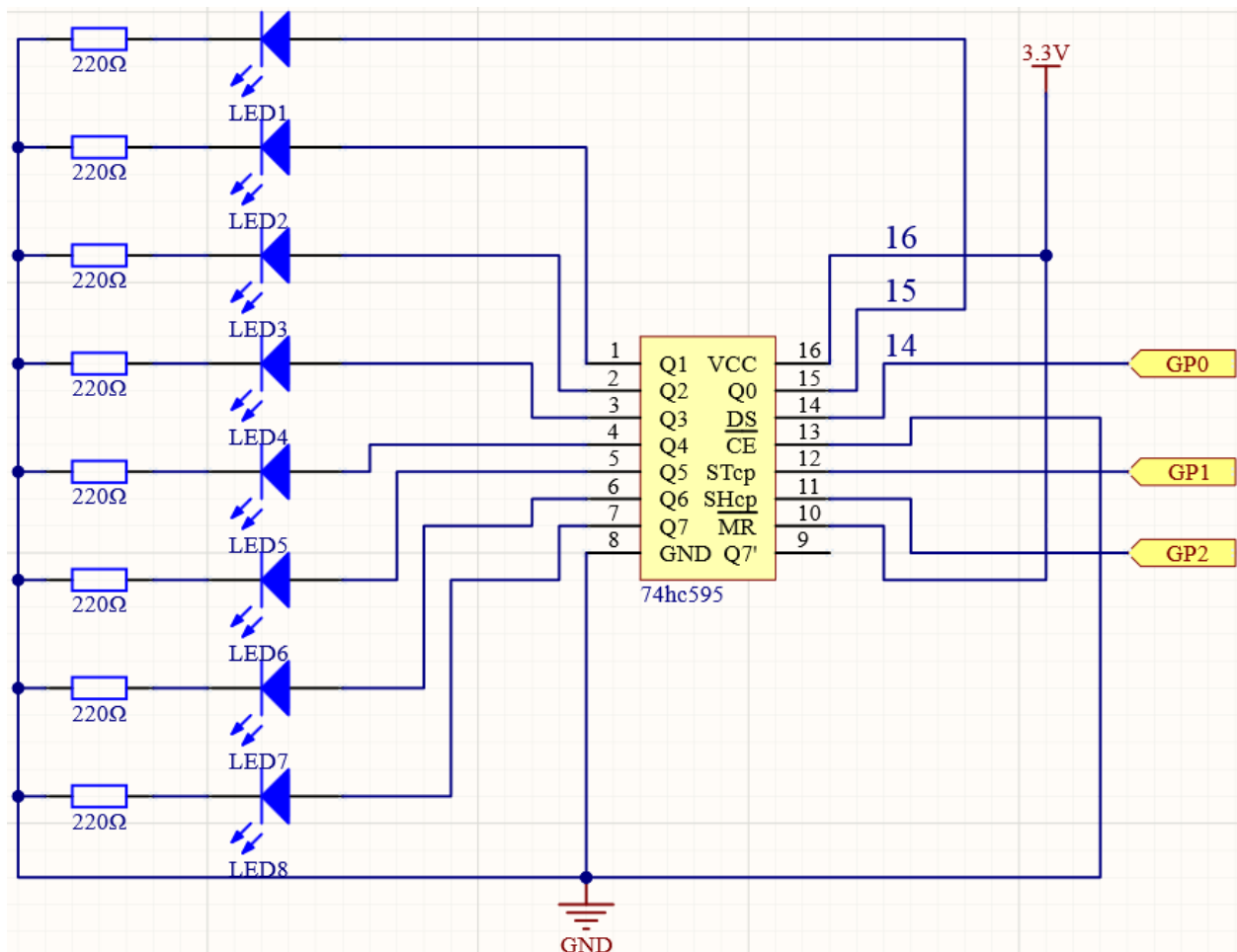
3.20.2 Wiring

The 74HC595 is a 16-pin IC with a semi-circular notch on one side (usually the left side of the label). With the notch facing upwards, its pins are shown in the diagram below.

Refer to the figure below to build the circuit.

1. Connect 3V3 and GND of Pico to the power bus of the breadboard.

2. Insert 74HC595 across the middle gap into the breadboard.

3. Connect the GP0 pin of Pico to the DS pin (pin 14) of 74HC595 with a jumper wire.

4. Connect the GP1 pin of Pico to the STcp pin (12-pin) of 74HC595.

5. Connect the GP2 pin of Pico to the SHcp pin (pin 11) of 74HC595.

6. Connect the VCC pin (16 pin) and MR pin (10 pin) on the 74HC595 to the positive power bus.

7. Connect the GND pin (8-pin) and CE pin (13-pin) on the 74HC595 to the negative power bus.

8. Insert 8 LEDs on the breadboard, and their anode leads are respectively connected to the Q0~Q1 pins (15, 1, 2, 3, 4, 5, 6, 7) of 74HC595.

9. Connect the cathode leads of the LEDs with a 220 resistor in series to the negative power bus.

3.20.3 Code

```python
import machine
import time

sdi = machine.Pin(0,machine.Pin.OUT)
rclk = machine.Pin(1,machine.Pin.OUT)
srclk = machine.Pin(2,machine.Pin.OUT)

def hc595_shift(dat):
    rclk.low()
    time.sleep_ms(5)
    for bit in range(7, -1, -1):
        srclk.low()
        time.sleep_ms(5)
        value = 1 & (dat >> bit)
        sdi.value(value)
        time.sleep_ms(5)
        srclk.high()
        time.sleep_ms(5)
    time.sleep_ms(5)
    rclk.high()
    time.sleep_ms(5)

num = 0

for i in range(16):
    if i < 8:
        num = (num<<1) + 1
    elif i>=8:
        num = (num & 0b01111111)<<1
    hc595_shift(num)
    print("{:0>8b}".format(num))
    time.sleep_ms(200)
```

When the program is running, `num` will be written into the 74HC595 chip as an eight-bit binary number to control the on and off of the 8 LEDs. We can see the current value of `num` in the shell.

3.20.4 How it works?

`hc595_shift()` will make 74HC595 output 8 digital signals. It outputs the last bit of the binary number to Q0, and the output of the first bit to Q7. In other words, writing the binary number "00000001" will make Q0 output high level and Q1~Q7 output low level.

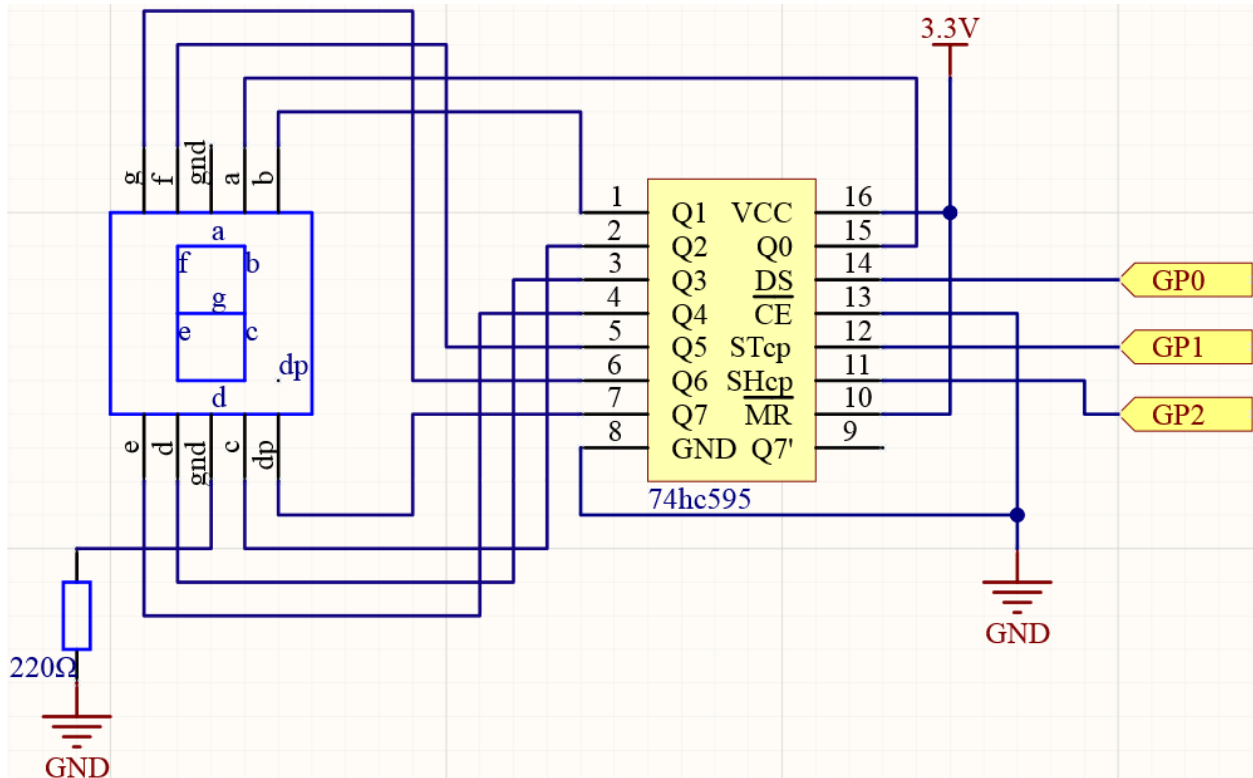## 3.21 2.16 7-Segment Display

7-Segment Display can be seen everywhere in life. For example, on an air conditioner, it can be used to display temperature; on a traffic indicator, it can be used to display a timer.

The 7-Segment Display is essentially a device packaged by 8 LEDs, of which 7 strip-shaped LEDs form an "8" shape, and there is a slightly smaller dotted LED as a decimal point. These LEDs are marked as a, b, c, d, e, f, g, and dp. They have their own anode pins and share cathodes. Their pin locations are shown in the figure below.
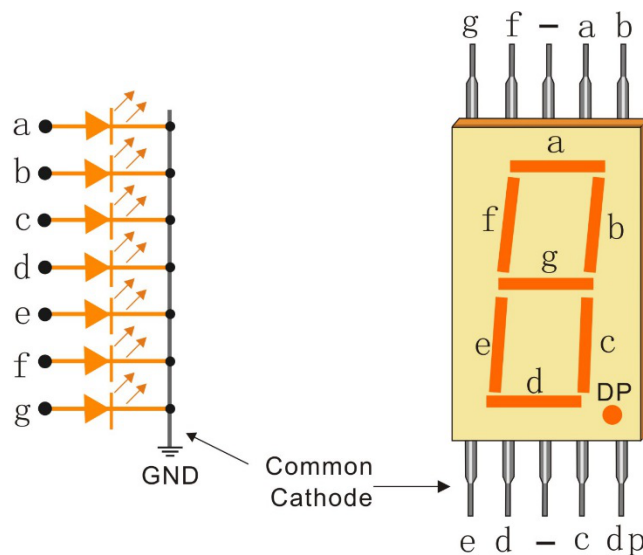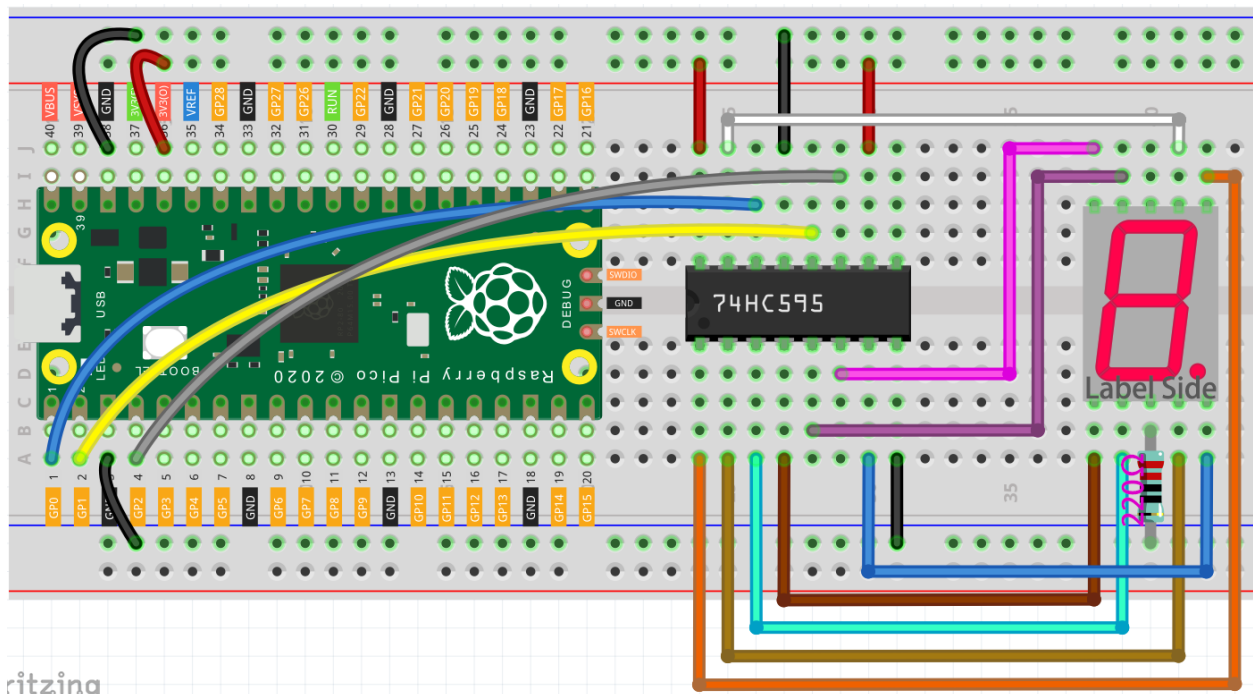

This means that it needs to be controlled by 8 digital signals at the same time to fully work and the 74HC595 can do this.

3.21.1 Schematic

3.21.2 Wiring

1. Connect 3V3 and GND of Pico to the power bus of the breadboard.

2. Insert 74HC595 across the middle gap into the breadboard.

3. Connect the GP0 pin of Pico to the DS pin (pin 14) of 74HC595 with a jumper wire.

4. Connect the GP1 pin of Pico to the STcp pin (12-pin) of 74HC595.

5. Connect the GP2 pin of Pico to the SHcp pin (pin 11) of 74HC595.

6. Connect the VCC pin (16 pin) and MR pin (10 pin) on the 74HC595 to the positive power bus.

7. Connect the GND pin (8-pin) and CE pin (13-pin) on the 74HC595 to the negative power bus.

8. Insert the LED Segment Display into the breadboard, and connect a 220 resistor in series with the GND pin to the negative power bus.

Note: The color ring of the 220 ohm resistor is red, red, black, black and brown.

---

9. Follow the table below to connect the 74hc595 and LED Segment Display.

Table 1: Wiring

74HC595	LED Segment Display
Q0	a
Q1	b
Q2	c
Q3	d
Q4	e
Q5	f
Q6	g
Q7	dp

3.21.3 Code

```python
import machine
import time

SEGCODE = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f]

sdi = machine.Pin(0,machine.Pin.OUT)
rclk = machine.Pin(1,machine.Pin.OUT)
srclk = machine.Pin(2,machine.Pin.OUT)

def hc595_shift(dat):
    rclk.low()
    time.sleep_ms(5)
    for bit in range(7, -1, -1):
        srclk.low()
        time.sleep_ms(5)
        value = 1 & (dat >> bit)
        sdi.value(value)
        time.sleep_ms(5)
        srclk.high()
        time.sleep_ms(5)
    time.sleep_ms(5)
    rclk.high()
    time.sleep_ms(5)
```

```
while True:
    for num in range(10):
        hc595_shift(SEGCODE[num])
        time.sleep_ms(500)
```

When the program is running, you will be able to see the LED Segment Display display 0~9 in sequence.

3.21.4 How it works?

`hc595_shift()` will make 74HC595 output 8 digital signals. It outputs the last bit of the binary number to Q0, and the output of the first bit to Q7. In other words, writing the binary number "00000001" will make Q0 output high level and Q1~Q7 output low level.

Suppose that the 7-segment Display display the number "1", we need to write a high level for b, c, and write a low level for a, d, e, f, g, and dg. That is, the binary number "00000110" needs to be written. For readability, we will use hexadecimal notation as "0x06".

- Hexadecimal
- BinaryHex Converter

Similarly, we can also make the LED Segment Display display other numbers in the same way. The following table shows the codes corresponding to these numbers.

Table 2: Glyph Code

Numbers	Binary Code	Hex Code
0	00111111	0x3f
1	00000110	0x06
2	01011011	0x5b
3	01001111	0x4f
4	01100110	0x66
5	01101101	0x6d
6	01111101	0x7d
7	00000111	0x07
8	01111111	0x7f
9	01101111	0x6f

Write these codes into `hc595_shift()` to make the LED Segment Display display the corresponding numbers.

3.22 2.17 Show Characters on I2 LCD1602

LCD1602 is a character type liquid crystal display, which can display 32 (16*2) characters at the same time.

As we all know, though LCD and some other displays greatly enrich the man-machine interaction, they share a common weakness. When they are connected to a controller, multiple IOs will be occupied of the controller which has no so many outer ports. Also it restricts other functions of the controller. Therefore, LCD1602 with an I2C bus is developed to solve the problem.

- Inter-Integrated Circuit - Wikipedia

I2C(Inter-Integrated Circuit) bus is a very popular and powerful bus for communication between a master device (or master devices) and a single or multiple slave devices. I2C main controller can be used to control IO expander, various

sensors, EEPROM, ADC/DAC and so on. All of these are controlled only by the two pins of host, the serial data (SDA) line and the serial clock line(SCL).

These two pins must be connected to specific pins of the microcontroller. There are two pairs of I2C communication interfaces in Pico, which are marked as I2C0 and I2C1, as shown in the figure below.
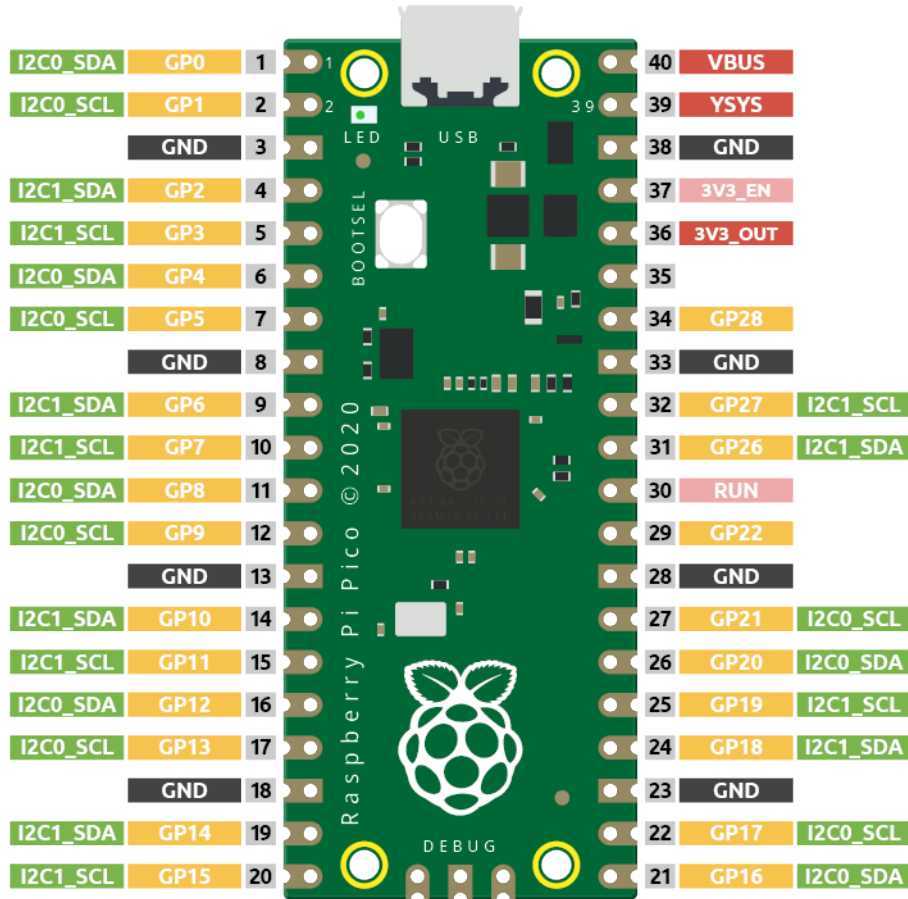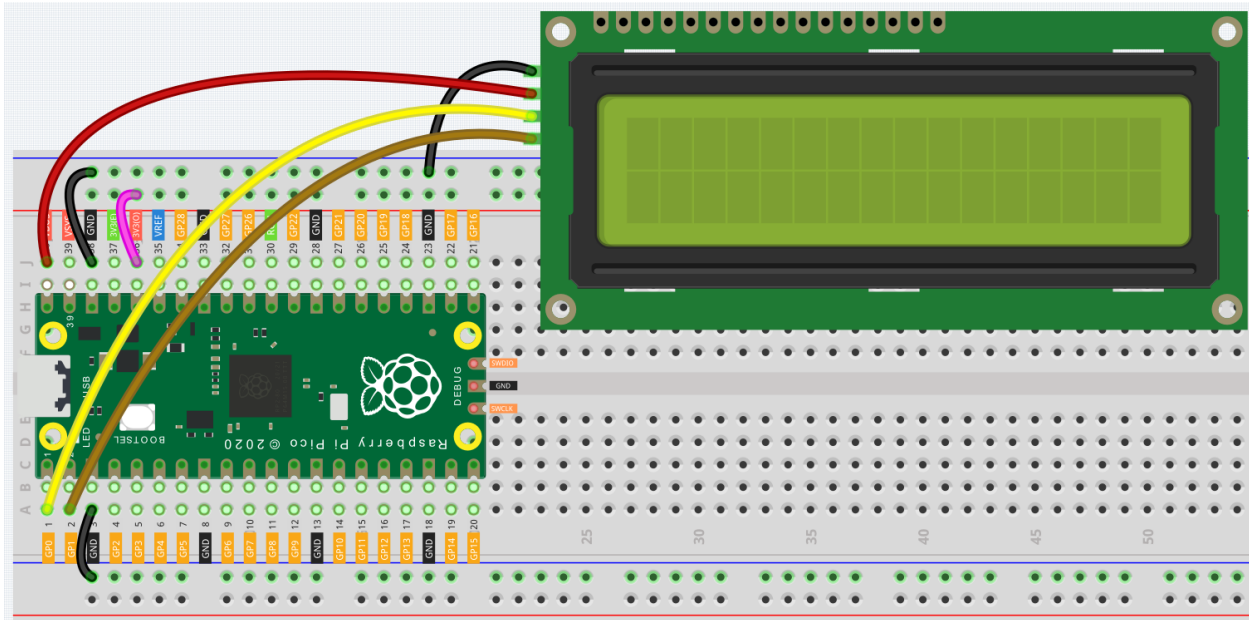

Here we will use the I2C0 interface to control the LCD1602 and display text.

3.22.1 Schematic
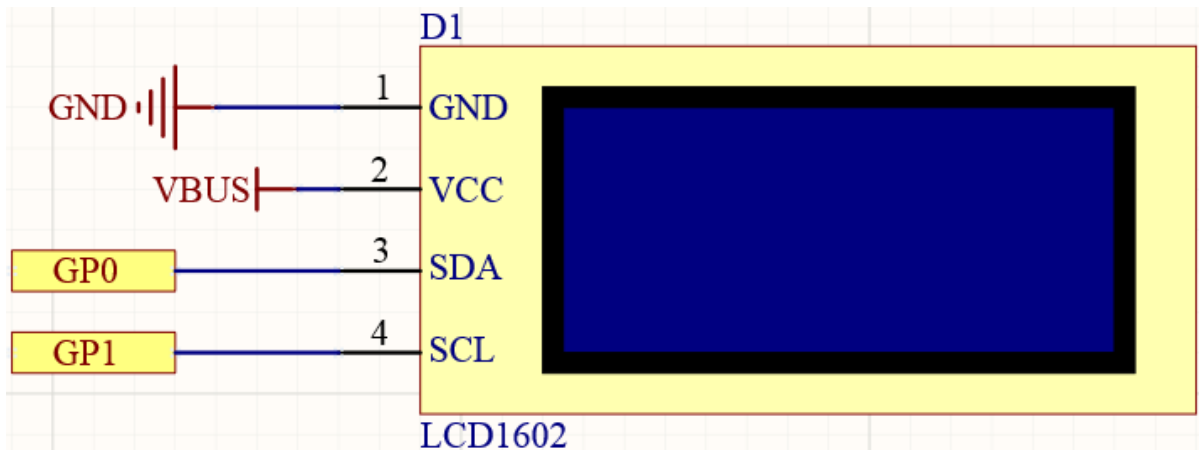
3.22.2 Wiring

1. Connect VCC of LCD to VBUS of Pico.
2. Connect the GND of LCD to the GND of Pico.
3. Connect SDA of LCD to GP0 of Pico, which is I2C0 SDA.
4. Connect SCL of LCD to GP1 of Pico, which is I2C0 SCL.

3.22.3 Create a Library

The following is the library of lcd1602 packaged by SunFounder.

You need to create a new file, copy the following code into it and save it to Raspberry Pi Pico under the name `lcd1602.py`.

You can then call it inside of other programs.

```python
import machine
import time

class LCD():
    def __init__(self, addr=0x27, blen=1):
        sda = machine.Pin(0)
        scl = machine.Pin(1)
        self.bus = machine.I2C(0,sda=sda, scl=scl, freq=400000)
        #print(self.bus.scan())
        self.addr = addr
        self.blen = blen
        self.send_command(0x33) # Must initialize to 8-line mode at first
        time.sleep(0.005)
        self.send_command(0x32) # Then initialize to 4-line mode
        time.sleep(0.005)
        self.send_command(0x28) # 2 Lines & 5*7 dots
        time.sleep(0.005)
```

(continues on next page)

```python
        self.send_command(0x0C) # Enable display without cursor
        time.sleep(0.005)
        self.send_command(0x01) # Clear Screen
        self.bus.writeto(self.addr, bytearray([0x08]))

    def write_word(self, data):
        temp = data
        if self.blen == 1:
            temp |= 0x08
        else:
            temp &= 0xF7
        self.bus.writeto(self.addr, bytearray([temp]))

    def send_command(self, cmd):
        # Send bit7-4 firstly
        buf = cmd & 0xF0
        buf |= 0x04                 # RS = 0, RW = 0, EN = 1
        self.write_word(buf)
        time.sleep(0.002)
        buf &= 0xFB                 # Make EN = 0
        self.write_word(buf)

        # Send bit3-0 secondly
        buf = (cmd & 0x0F) << 4
        buf |= 0x04                 # RS = 0, RW = 0, EN = 1
        self.write_word(buf)
        time.sleep(0.002)
        buf &= 0xFB                 # Make EN = 0
        self.write_word(buf)

    def send_data(self, data):
        # Send bit7-4 firstly
        buf = data & 0xF0
        buf |= 0x05                 # RS = 1, RW = 0, EN = 1
        self.write_word(buf)
        time.sleep(0.002)
        buf &= 0xFB                 # Make EN = 0
        self.write_word(buf)

        # Send bit3-0 secondly
        buf = (data & 0x0F) << 4
        buf |= 0x05                 # RS = 1, RW = 0, EN = 1
        self.write_word(buf)
        time.sleep(0.002)
        buf &= 0xFB                 # Make EN = 0
        self.write_word(buf)

    def clear(self):
        self.send_command(0x01) # Clear Screen

    def openlight(self):  # Enable the backlight
        self.bus.writeto(self.addr,bytearray([0x08]))
        # self.bus.close()

    def write(self, x, y, str):
        if x < 0:
            x = 0
```

```
        if x > 15:
            x = 15
        if y < 0:
            y = 0
        if y > 1:
            y = 1

        # Move cursor
        addr = 0x80 + 0x40 * y + x
        self.send_command(addr)

        for chr in str:
            self.send_data(ord(chr))

    def message(self, text):
        #print("message: %s"%text)
        for char in text:
            if char == '\n':
                self.send_command(0xC0) # next line
            else:
                self.send_data(ord(char))
```

3.22.4 Code

Then, create a new file, and call the lcd1602 library stored before in this file.

```
from lcd1602 import LCD
import utime

lcd = LCD()
string = " Hello!\n"
lcd.message(string)
utime.sleep(2)
string = "    Sunfounder!"
lcd.message(string)
utime.sleep(2)
lcd.clear()
```

After the program runs, you will be able to see two lines of text appear on the LCD in turn, and then disappear.

3.22.5 How it works?

In the lcd1602 library, we integrate the relevant functions of lcd1602 into the LCD class.

Import lcd1602 library

```
from lcd1602 import LCD
```

Declare an object of the LCD class and name it lcd.

```
lcd = LCD()
```

This statement will display the text on the LCD. It should be noted that the argument must be a string type. If we want to pass an integer or float, we must use the forced conversion statement `str()`.
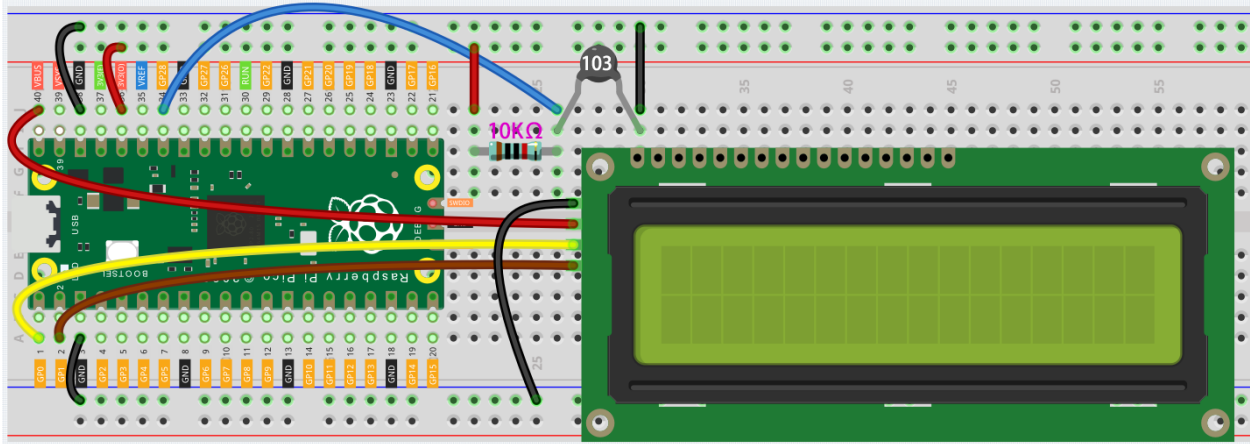
```
lcd.message(string)
```

If you call this statement multiple times, lcd will superimpose the texts. This requires the use of the following statement to clear the display.

```
lcd.clear()
```

3.22.6 What more?

We can combine thermistor and I2C LCD1602 to make a room temperature meter.


```python
from lcd1602 import LCD
import machine
import utime
import math

thermistor = machine.ADC(28)
lcd = LCD()

while True:
    temperature_value = thermistor.read_u16()
    Vr = 3.3 * float(temperature_value) / 65535
    Rt = 10000 * Vr / (3.3 - Vr)
    temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
    Cel = temp - 273.15
    #Fah = Cel * 1.8 + 32
    #print ('Celsius: %.2f C  Fahrenheit: %.2f F' % (Cel, Fah))
    #utime.sleep_ms(200)

    string = " Temperature is \n    " + str('{:.2f}'.format(Cel))+ " C"
    lcd.message(string)
    utime.sleep(1)
    lcd.clear()
```

3.23 2.18 RGB LED Strip

WS2812 is a intelligent control LED light source that the control circuit and RGB chip are integrated in a package of 5050 components. It internal include intelligent digital port data latch and signal reshaping amplification drive circuit. Also include a precision internal oscillator and a 12V voltage programmable constant current control part, effectively ensuring the pixel point light color height consistent.

The data transfer protocol use single NZR communication mode. After the pixel power-on reset, the DIN port receive data from controller, the first pixel collect initial 24bit data then sent to the internal data latch, the other data which reshaping by the internal signal reshaping amplification circuit sent to the next cascade pixel through the DO port. After transmission for each pixelthe signal to reduce 24bit. pixel adopt auto reshaping transmit technology, making the pixel cascade number is not limited the signal transmission, only depend on the speed of signal transmission.

3.23.1 Schematic
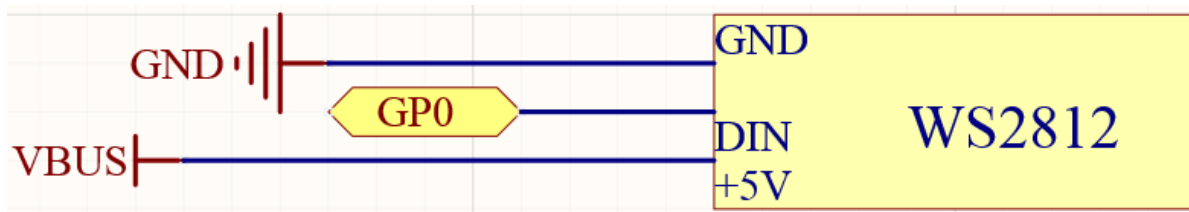
3.23.2 Wiring
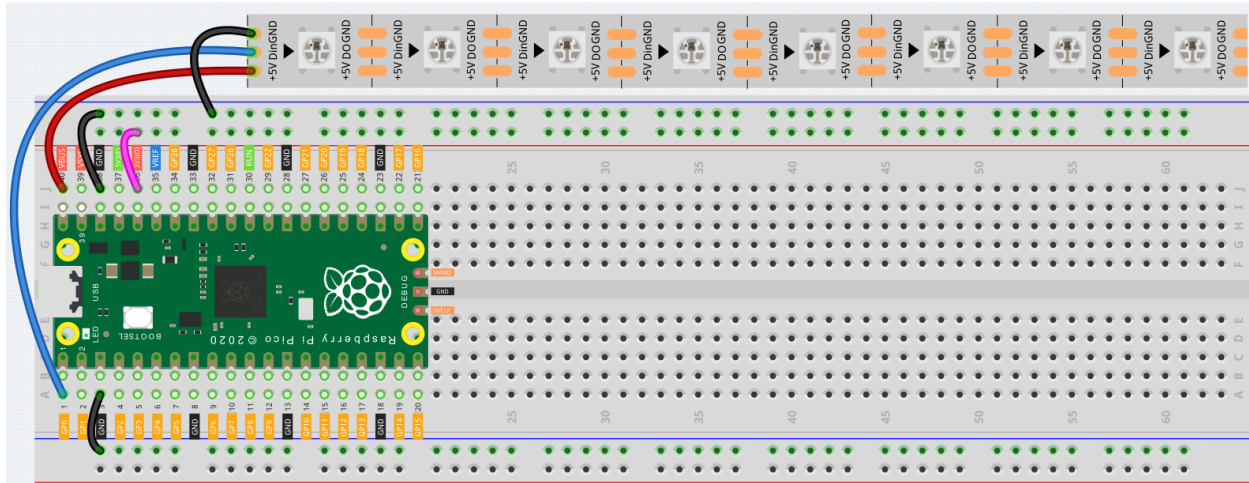
1. Connect the +5V of the LED Strip to the VBUS of the Pico.

2. Connect the GND of the LED Strip to the GND of the Pico.

3. Connect the DIN of the LED Strip to the GP0 of Pico.

> **Warning:** One thing you need to pay attention to is current.
>
> Although the LED Strip with any number of LEDs can be used in Pico, the power of its VBUS pin is limited. Here, we will use eight LEDs, which are safe. But if you want to use more LEDs, you need to add a separate power supply.

3.23.3 Code

The following is the library of ws2812 packaged by SunFounder. You need to save it in Pico and name it as `ws2812.py` for use as a library.

```python
import array, time
import rp2
from rp2 import PIO, StateMachine, asm_pio


@asm_pio(sideset_init=PIO.OUT_LOW, out_shiftdir=PIO.SHIFT_LEFT, autopull=True, pull_
→thresh=24)
def ws2812():
    T1 = 2
    T2 = 5
    T3 = 3
    label("bitloop")
    out(x, 1).side(0)[T3 - 1]
    jmp(not_x, "do_zero").side(1)[T1 - 1]
    jmp("bitloop").side(1)[T2 - 1]
    label("do_zero")
    nop().side(0)[T2 - 1]


class WS2812():

    def __init__(self, pin, num):
        # Configure the number of WS2812 LEDs.
        self.led_nums = num
        self.pin = pin
        self.sm = StateMachine(0, ws2812, freq=8000000, sideset_base=self.pin)
        # Start the StateMachine, it will wait for data on its FIFO.
        self.sm.active(1)

        self.buf = array.array("I", [0 for _ in range(self.led_nums)])

    def write(self):
        self.sm.put(self.buf, 8)

    def write_all(self, value):
        for i in range(self.led_nums):
            self.__setitem__(i, value)
        self.write()

    def list_to_hex(self, color):
        if isinstance(color, list) and len(color) == 3:
            c = (color[0] << 8) + (color[1] << 16) + (color[2])
            return c
        elif isinstance(color, int):
            value = (color & 0xFF0000)>>8 | (color & 0x00FF00)<<8 | (color & 0x0000FF)
            return value
        else:
            raise ValueError("Color must be 24-bit  RGB hex or list of 3 8-bit RGB")

    def hex_to_list(self, color):
        if isinstance(color, list) and len(color) == 3:
            return color
        elif isinstance(color, int):
            r = color >> 8 & 0xFF
            g = color >> 16 & 0xFF
```

(continues on next page)

```
            b = color >> 0 & 0xFF
            return [r, g, b]
        else:
            raise ValueError("Color must be 24-bit  RGB hex or list of 3 8-bit RGB")

    def __getitem__(self, i):
        return self.hex_to_list(self.buf[i])

    def __setitem__(self, i, value):
        value = self.list_to_hex(value)
        self.buf[i] = value
```

Then, create a new file, and call the stored ws2812 library here.

```
import machine
from ws2812 import WS2812

ws = WS2812(machine.Pin(0),8)

ws[0] = [64,154,227]
ws[1] = [128,0,128]
ws[2] = [50,150,50]
ws[3] = [255,30,30]
ws[4] = [0,128,255]
ws[5] = [99,199,0]
ws[6] = [128,128,128]
ws[7] = [255,100,0]
ws.write()
```

Let's select some favorite colors and display them on the RGB LED Strip!

3.23.4 How it works?

In the ws2812 library, we have integrated related functions into the WS2812 class.

You can use the RGB LED Strip with the following statement.

```
from ws2812 import WS2812
```

Declare a WS2812 type object, named "ws", it is connected to "pin", there are "number" RGB LEDs on the WS2812 strip.

```
ws = WS2812(pin,number)
```

ws is an array object, each element corresponds to one RGB LED on the WS2812 strip, for example, ws[0] is the first one, ws[7] is the eighth.

We can assign color values to each RGB LED, these values must be 24-bit color (represented with six hexadecimal digits) or list of 3 8-bit RGB.

For example, the red value is "0xFF0000" or "[255,0,0]".

```
ws[i] = color value
```

Then use this statement to write the color for the LED Strip and light it up.

```
ws.write()
```

You can also directly use the following statement to make all LEDs light up the same color.

```
ws.write_all(color value)
```

3.23.5 What more?

We can randomly generate colors and make a colorful flowing light.

```python
import machine
from ws2812 import WS2812
import utime
import urandom

ws = WS2812(machine.Pin(0),8)

def flowing_light():
    for i in range(7,0,-1):
        ws[i] = ws[i-1]
    ws[0] = int(urandom.uniform(0, 0xFFFFFF))
    ws.write()
    utime.sleep_ms(80)

while True:
    flowing_light()
    print(ws[0])
```
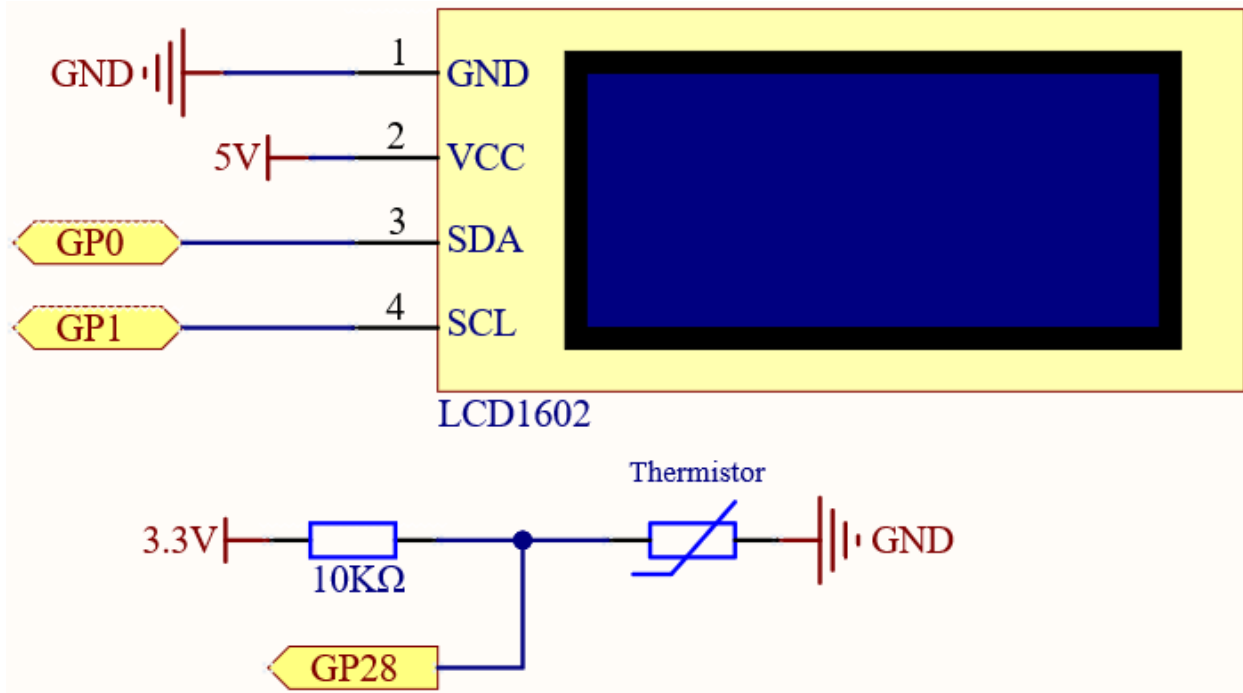
3.24 2.19 Room Temperature Meter

Using a thermistor and an I2C LCD1602, we can create a room temperature meter.

This project is very simple, it is based on *2.13 Thermometer* with I2C LCD1602 to display the temperature.

Schematic

Wiring
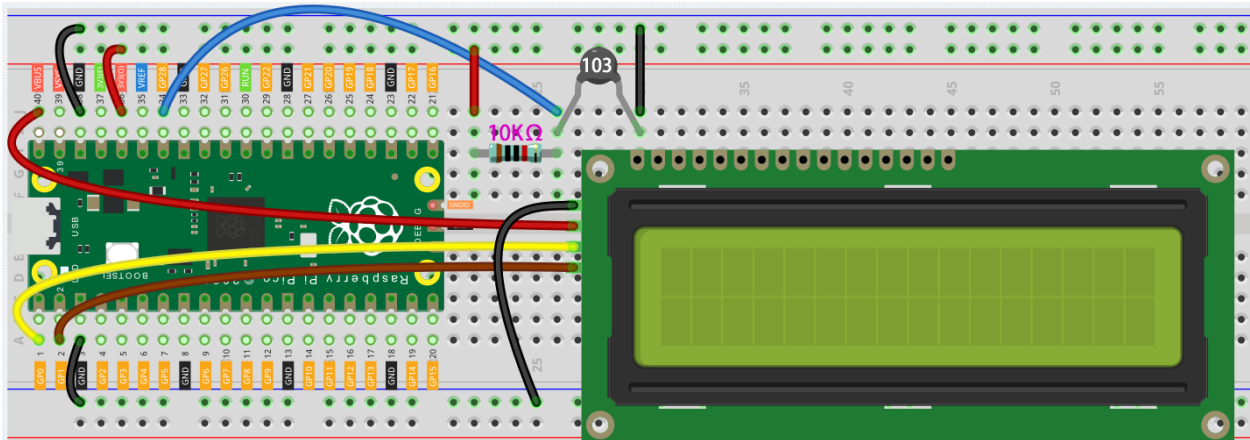
Code

Note: The `lcd1602.py` library is used here, please make sure you have it in your Raspberry Pi Pico, if not, please refer to *Create a Library*.

```python
from lcd1602 import LCD
import machine
import utime
import math

thermistor = machine.ADC(28)
lcd = LCD()

while True:
```

```
temperature_value = thermistor.read_u16()
Vr = 3.3 * float(temperature_value) / 65535
Rt = 10000 * Vr / (3.3 - Vr)
temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
Cel = temp - 273.15
#Fah = Cel * 1.8 + 32
#print ('Celsius: %.2f C  Fahrenheit: %.2f F' % (Cel, Fah))
#utime.sleep_ms(200)

string = " Temperature is \n    " + str('{:.2f}'.format(Cel))+ " C"
lcd.message(string)
utime.sleep(1)
lcd.clear()
```

The LCD will display the temperature value in the current environment after the program runs.

Note: If the code and wiring are fine, but the LCD still does not display content, you can turn the potentiometer on the back to increase the contrast.

3.25 2.20 Alarm Siren Lamp

Police lights are often visible in real life (or in movies). Usually, it is used to maintain traffic, serve as a warning device, and serve as an important safety prop for officers, emergency vehicles, fire trucks, and engineering vehicles. When you see its lights or hear its sound, you must be careful, which means you (or those around you) may be in danger.

An LED and buzzer are used here to create a small warning light, which is activated by a slide switch.

Schematic
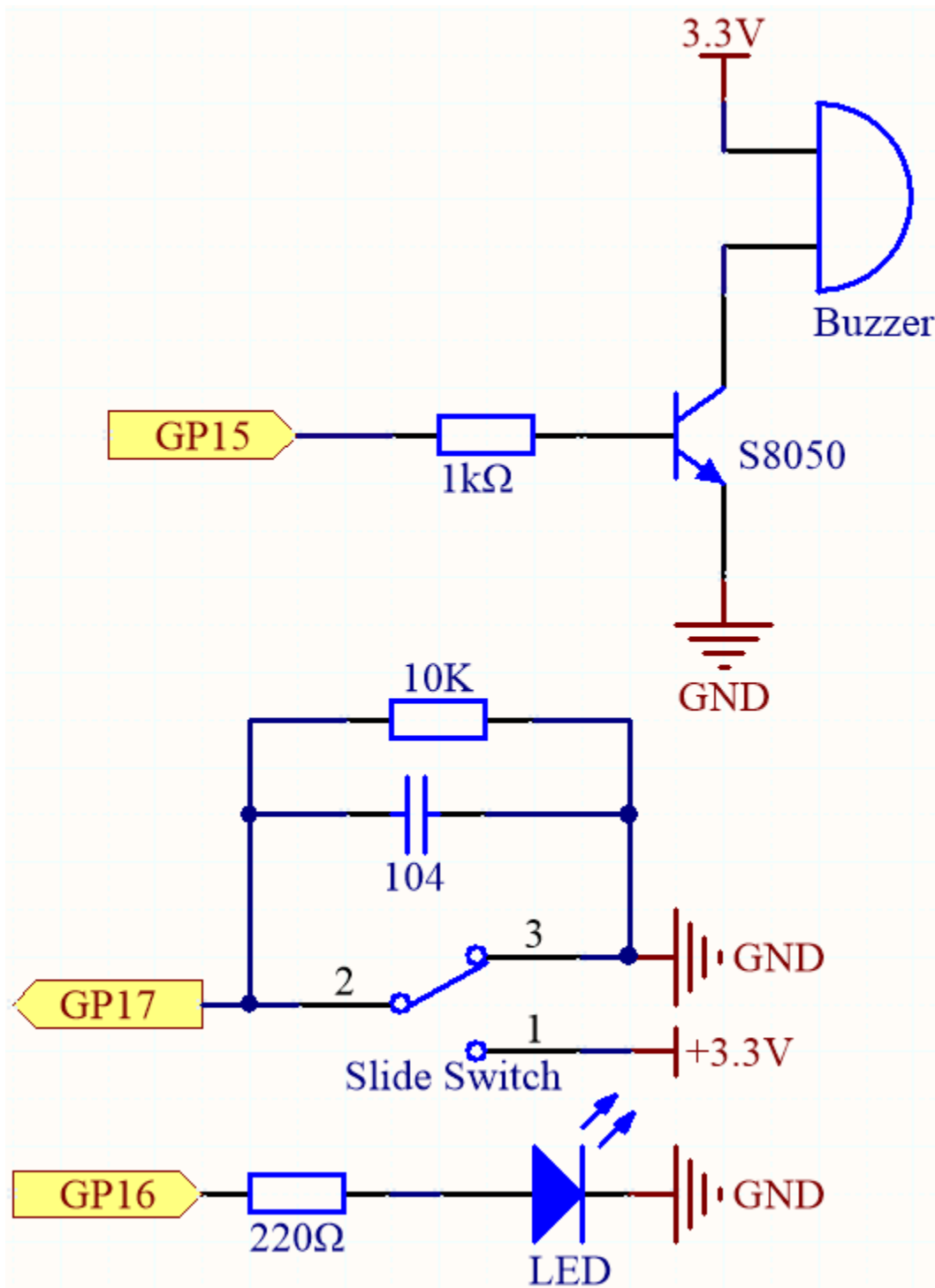
- GP17 is connected to the middle pin of the slider, along with a 10K resistor and a capacitor (filter) in parallel to GND, which allows the slider to output a steady high or low level when toggled to the left or right.

- As soon as GP15 is high, the NPN transistor conducts, causing the passive buzzer to start sounding. This passive buzzer is programmed to gradually increase in frequency to produce a siren sound.

- An LED is connected to GP16 and is programmed to periodically change its brightness in order to simulate a siren.

Wiring
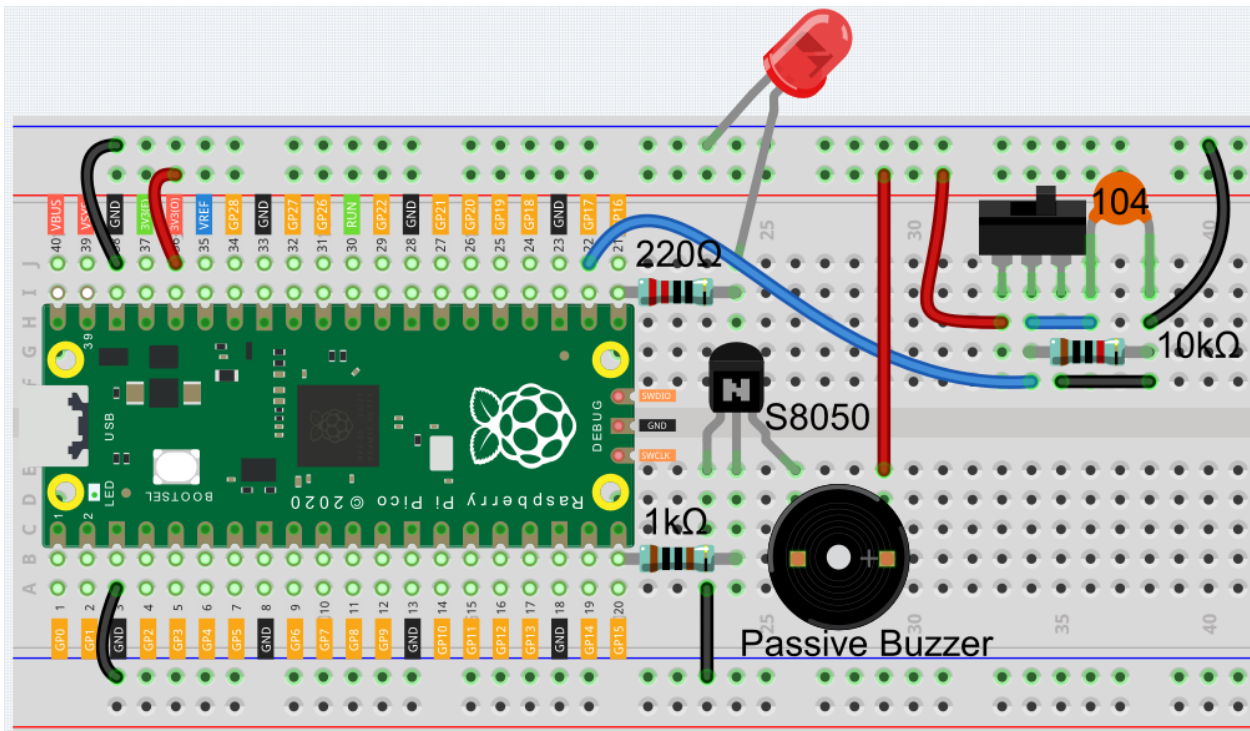
Code

```python
import machine
import time


buzzer = machine.PWM(machine.Pin(15))
led = machine.PWM(machine.Pin(16))
led.freq(1000)

switch = machine.Pin(17,machine.Pin.IN)

def noTone(pin):
    pin.duty_u16(0)


def tone(pin,frequency):
    pin.freq(frequency)
    pin.duty_u16(30000)

def interval_mapping(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def toggle(pin):
    global bell_flag
    bell_flag = not bell_flag
    print(bell_flag)
    if bell_flag:
        switch.irq(trigger=machine.Pin.IRQ_FALLING, handler=toggle)
    else:
```

```
        switch.irq(trigger=machine.Pin.IRQ_RISING, handler=toggle)


bell_flag = False
switch.irq(trigger=machine.Pin.IRQ_RISING, handler=toggle)



while True:
    if bell_flag == True:
        for i in range(0,100,2):
            led.duty_u16(int(interval_mapping(i,0,100,0,65535)))
            tone(buzzer,int(interval_mapping(i,0,100,130,800)))
            time.sleep_ms(10)
    else:
        noTone(buzzer)
        led.duty_u16(0)
```

Once the program is running, toggle the slide switch to the left (yours may be to the right, depending on how your slide switch is wired) and the buzzer will emit a progressive warning tone and the LED will change its brightness accordingly; toggle the slide switch to the right and the buzzer and LED will stop working.

FOR ARDUINO USER

This chapter includes installing Arduino IDE, programming Raspberry Pi Pico with the most popular microcontroller development environment-Arduino IDE and a dozen interesting and practical projects to help you learn Arduino code quickly.

We recommend that you read the chapters in order.

4.1 Getting Started with Arduino

Arduino is an open source platform with powerful software and hardware features.

Even if you are a beginner, you can master it in no time. It provides an integrated development environment (IDE) for code compilation that is compatible with various control boards. In fact, the Arduino IDE supports almost all hobbyist microcontrollers, including the Raspberry Pi Pico. Therefore, all you need to do is download the Arduino IDE, upload the sketches (i.e. code files) to the control board, and then you can see the relative experimental phenomena.

For more information, refer to Arduino Website.

4.1.1 Install Arduino IDE

The first thing we need to do is to download the IDE from the *Arduino software page* *<https://www.arduino.cc/en/software>*. The Raspberry Pi Pico is a newly released control board and may not be compatible on older versions of the Arduino IDE, so it is recommended that you download the latest version of the IDE, at least version 1.8.13 or higher.

If you are more explorative, you can also download Arduino IDE 2.0. This version is still in beta, so you may encounter some strange problems, but it is faster and even more powerful!

Choose the version that suits your system. You don't have to make a donation to download the software, but if you are a casual user and can afford it, this is a good way to help make sure it continues to get upgraded.

Downloads

4.1.2 Introduction the Arduino Software (IDE)

Double-click the Arduino icon (arduino.exe) created by the installation process. Then the Arduino IDE will appear. Let's check details of the software.
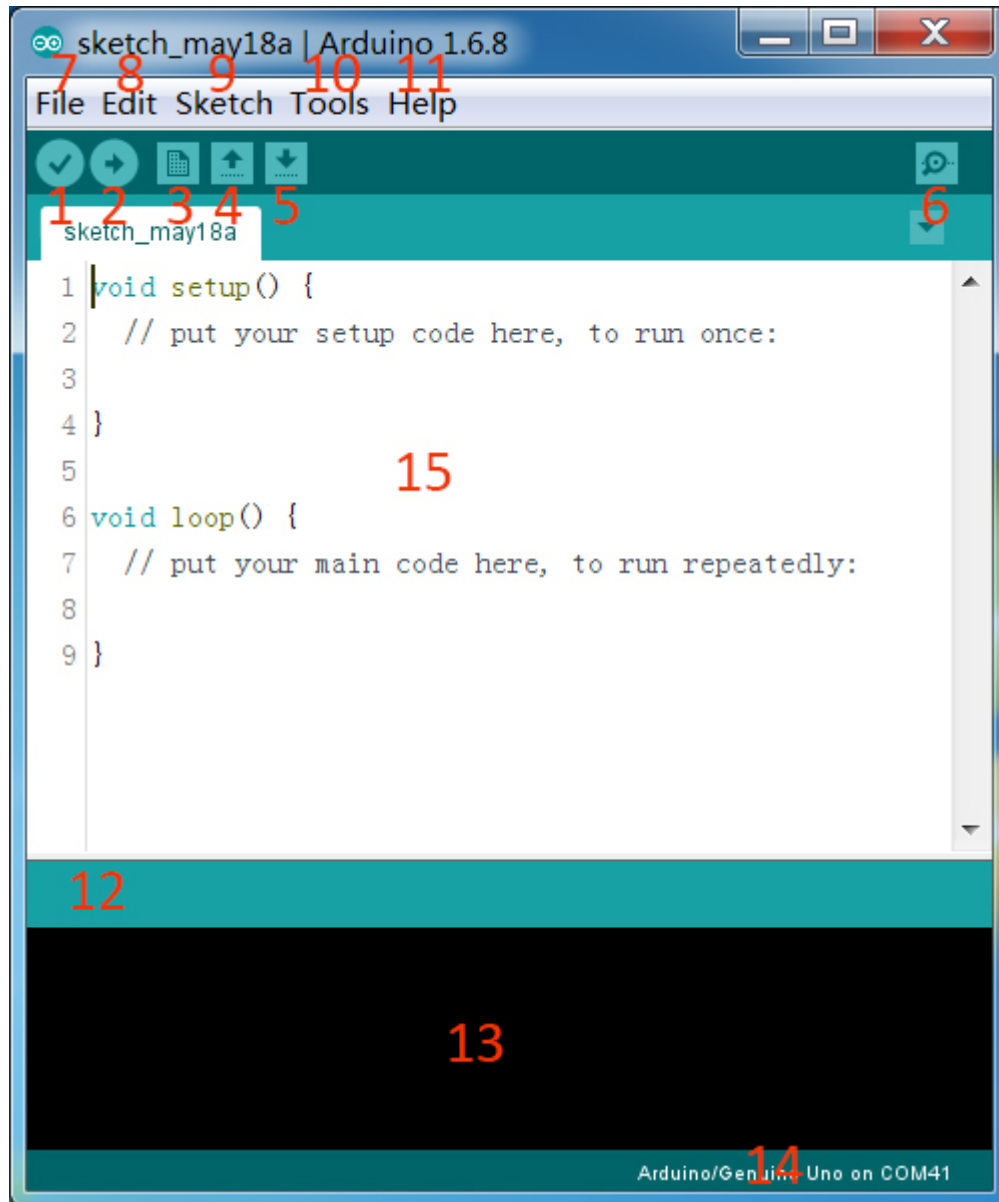
1. Verify: Compile your code. Any syntax problem will be prompted with errors.

2. Upload: Upload the code to your board. When you click the button, the RX and TX LEDs on the board will flicker fast and won't stop until the upload is done.

3. New: Create a new code editing window.

4. Open: Open an .ino sketch.

5. Save: Save the sketch.

6. Serial Monitor: Click the button and a window will appear. It receives the data sent from your control board. It is very useful for debugging.

7. File: Click the menu and a drop-down list will appear, including file creating, opening, saving, closing, some parameter configuring, etc.

8. Edit: Click the menu. On the drop-down list, there are some editing operations like Cut, Copy, Paste, Find, and so on, with their corresponding shortcuts.

9. Sketch: Includes operations like Verify, Upload, Add files, etc. More important function is Include Library – where you can add libraries.

10. Tool: Includes some tools – the most frequently used Board (the board you use) and Port (the port your board is at). Every time you want to upload the code, you need to select or check them.

11. Help: If you're a beginner, you may check the options under the menu and get the help you need, including operations in IDE, introduction information, troubleshooting, code explanation, etc.

12. In this message area, no matter when you compile or upload, the summary message will always appear.

13. Detailed messages during compile and upload. For example, the file used lies in which path, the details of error prompts.

14. Board and Port: Here you can preview the board and port selected for code upload. You can select them again by Tools -> Board / Port if any is incorrect.

15. The editing area of the IDE. You can write code here.

4.1.3 Setup the Raspberry Pi Pico

1. Once it's installed, open the application, we need to install the extra package that makes the Raspberry Pi Pico work.
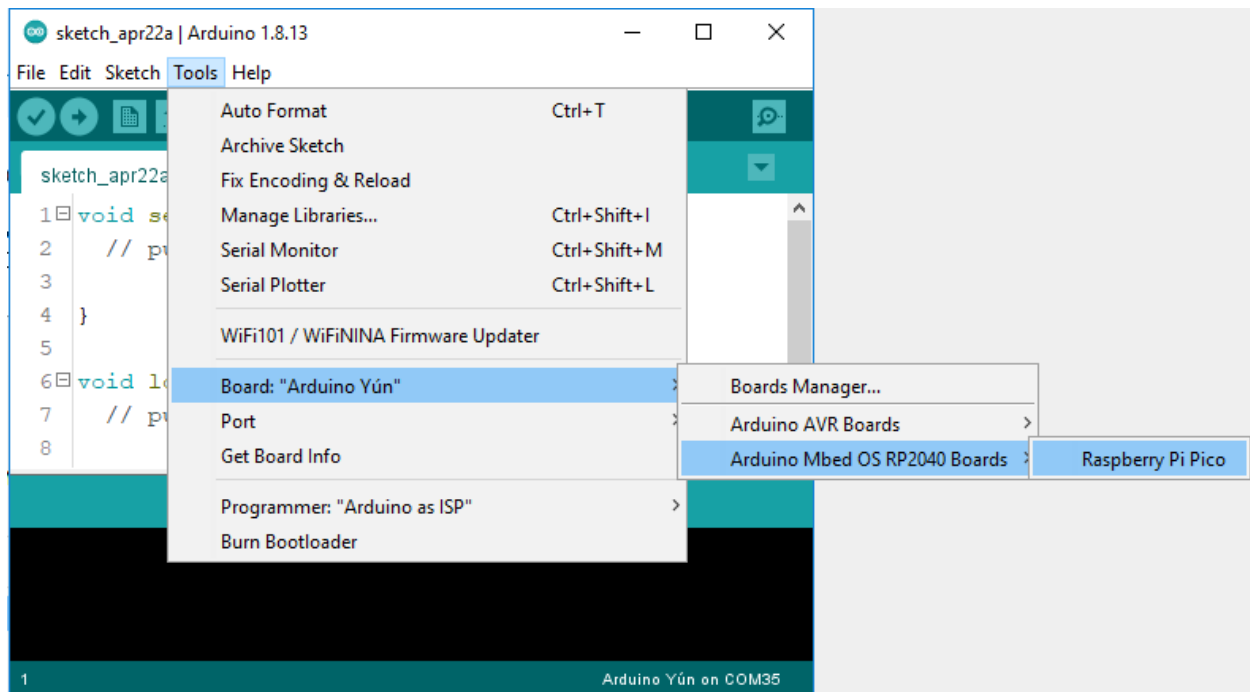
Open the Boards Manager by clicking **Tools** -> **Board** -> **Boards Manager**.
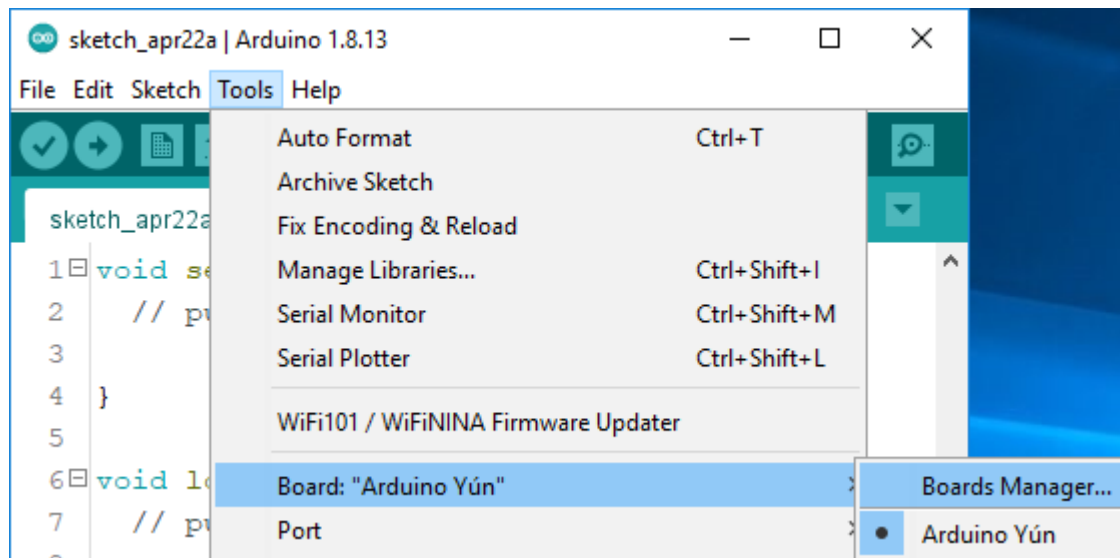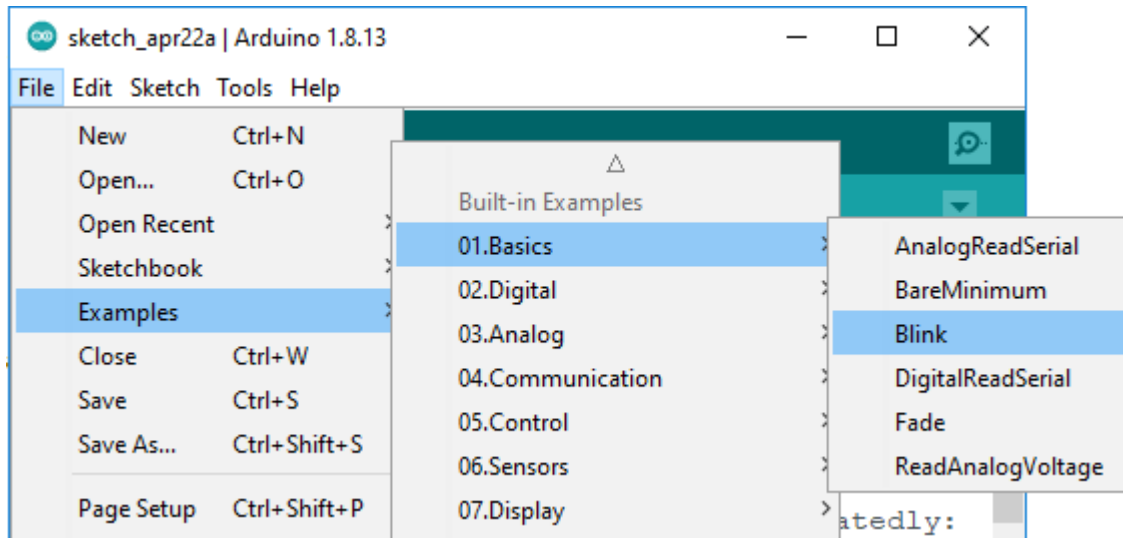

2. Search for **Pico(Arduino Mbed OS RP2040 Boards)** and click **install** button.
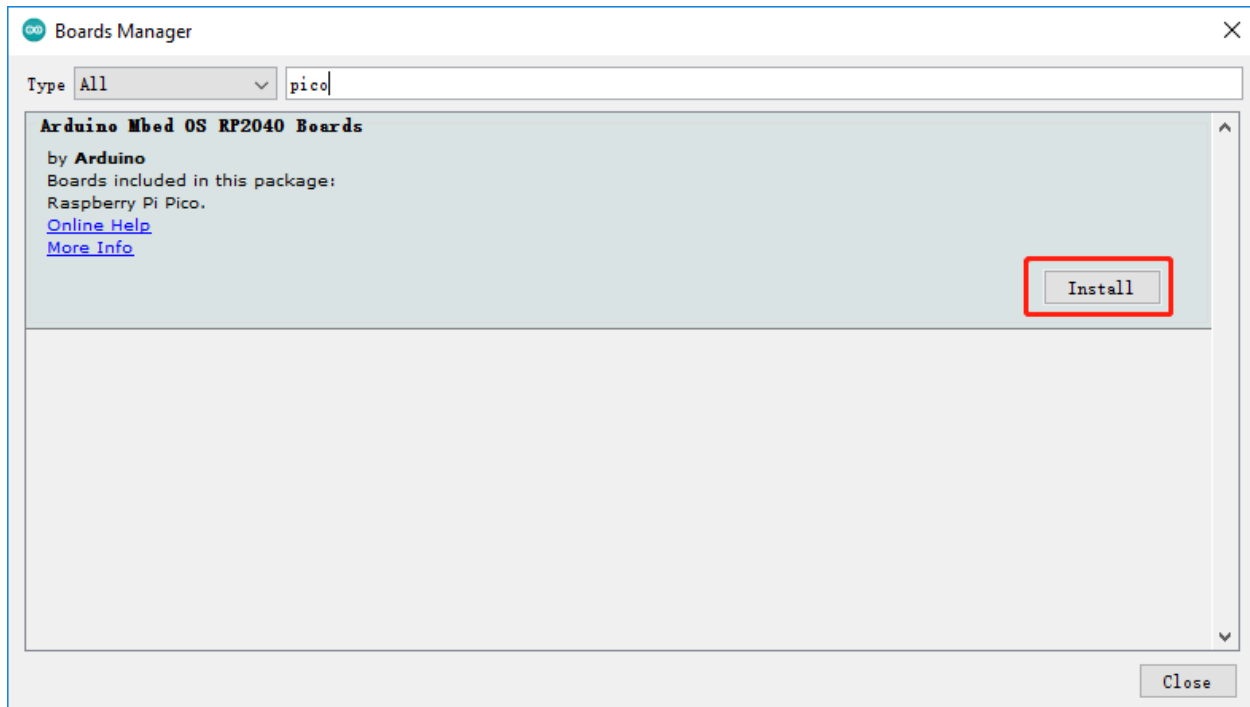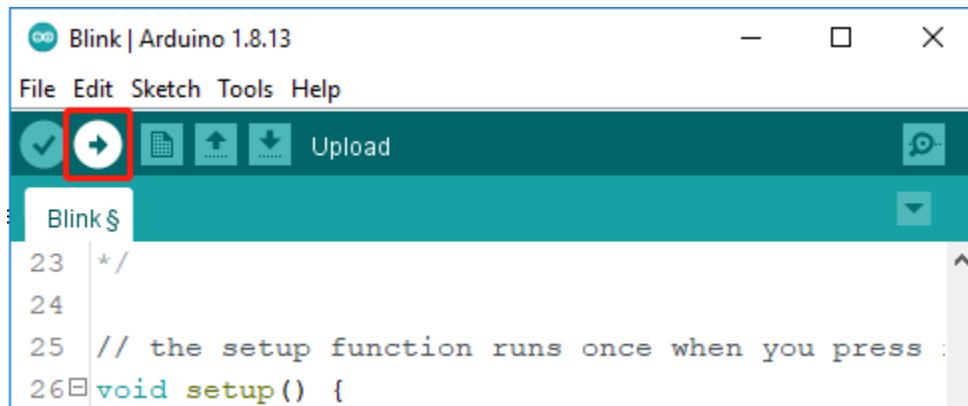
3. Once the installation is complete, you can select the board as **Raspberry Pi Pico**.
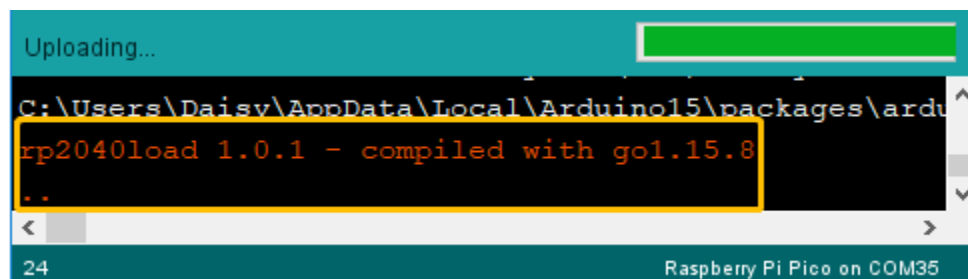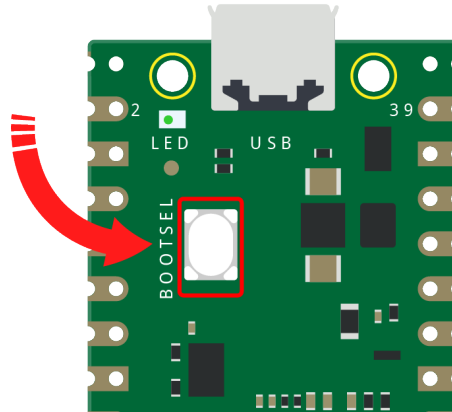

4. Now open a example - blink.

5. Click on the upload icon to run the code

6. When the compiling message shown in the figure below appears, press **BOOTSEL** immediately and connect Pico to the computer with a Micro USB cable.

Note: This step is very important and only necessary for the first use on the Arduino IDE, otherwise your code will upload unsuccessfully.

After the upload is successful this time, Pico will be recognized by the computer as COMxx (Raspberry Pi Pico).

You only need to plug it into the computer the next time you use it.

7. After the **Done Uploading** appear, you will see the LED on the Pico blinking.
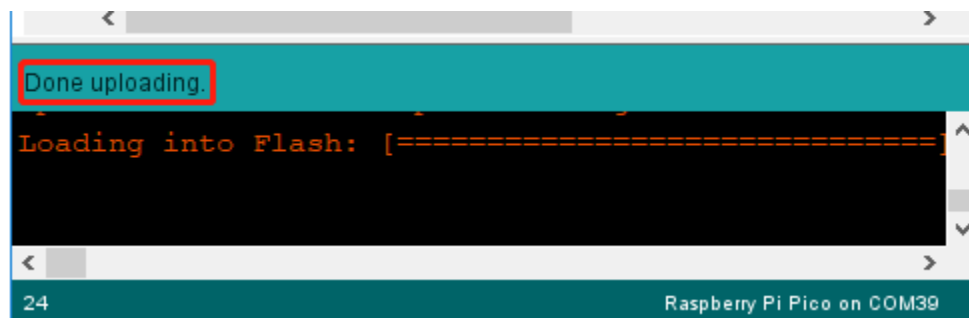
4.2 Projects

Note: The Pico Arduino core is a recent development, so it may take a long time to compile and upload, or the COM port may suddenly disappear during use.

- For the former, we have to wait patiently until a new Arduino core is released.

- For the latter case, you can unplug the USB cable, then press and hold the BOOTSEL button to plug it in, unplug it and plug it in again to see the COM port.

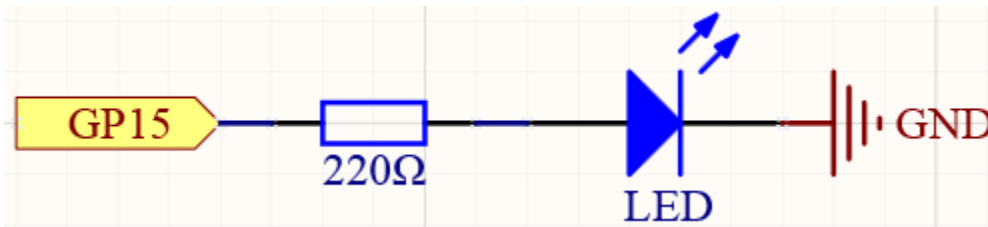- If you encounter a problem, you can come first *FAQ* to find a solution.

4.2.1 Hello LED

Just as printing "Hello, world!" is the first step in learning programming, letting the LED light up is the traditional entry to learning physical programming.
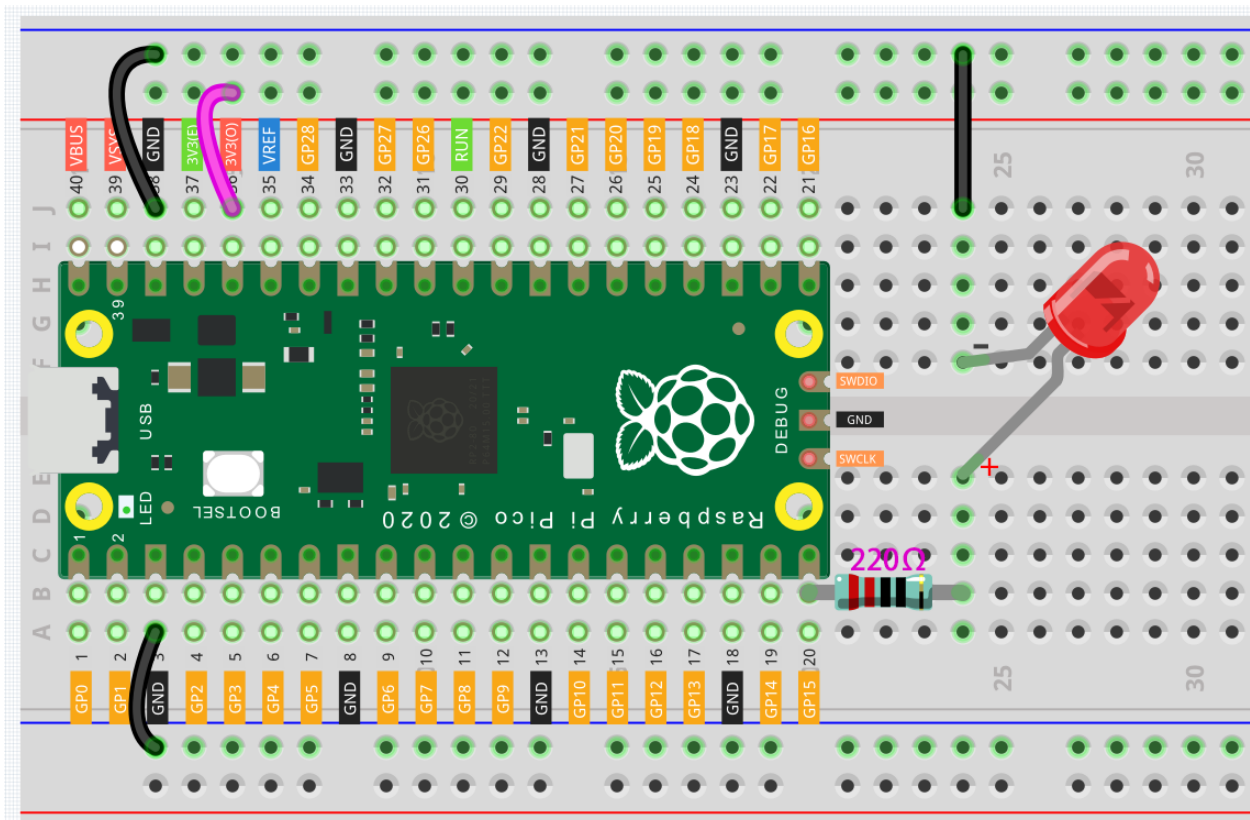
To use extended electronic components, a solderless breadboard will be the most powerful partner for novice users.

The breadboard is a rectangular plastic plate with a bunch of small holes in it. These holes allow us to easily insert electronic components and build electronic circuits. The breadboard does not permanently fix the electronic components, which makes it easy for us to repair the circuit and start over when we make a mistake.

Schematic

Wiring

Let us follow the direction of the current to build the circuit!

1. Here we use the electrical signal from the GP15 pin of the Pico board to make the LED work, and the circuit starts from here.

---

2. The current needs to pass through a 220 ohm resistor (used to protect the LED). Insert one end (either end) of the resistor into the same row as the Pico GP15 pin (row 20 in my circuit), and insert the other end into the free row of the breadboard (row 24 in my circuit).

Note: The color ring of the 220 ohm resistor is red, red, black, black and brown.

---

3. Pick up the LED, you will see that one of its leads is longer than the other. Insert the longer lead into the same row as the end of the resistor, and connect the shorter lead across the middle gap of the breadboard to the same row.

Note: The longer lead is known as the anode, and represents the positive side of the circuit; the shorter lead is the cathode, and represents the negative side.

The anode needs to be connected to the GPIO pin through a resistor; the cathode needs to be connected to the GND pin.

---

4. Insert the male-to-male (M2M) jumper wire into the same row as the LED short pin, and then connect it to the negative power bus of the breadboard.

5. Use a jumper to connect the negative power bus to the GND pin of Pico.

Code

- You can click the **Download Sketch** icon to download the code and open it with the desktop Arduino IDE.
- Or **OPEN IN WEB EDITOR**.
- Then upload the code to your Pico(*Setup the Raspberry Pi Pico*).

How it works?

Here, we connect the LED to the GPIO15, so we define a variable `ledPin` to represent GPIO15. You could also just replace all the `ledPin` pins in the code with 15, but if you were to replace 15 with other pins you would have to modify 15 one by one, which would add a lot of work.

```
#define ledPin 15
```

Now, you need to set the pin to OUTPUT mode in the `setup()` function.

```
pinMode(ledPin, OUTPUT);
```

- pinMode()

The above code has "set" the pin, but it will not light up the LED. Here, we use the digitalWrite() function to assign a high level signal to `ledpin`, which will cause a voltage difference between the LED pins, causing the LED to light up.

```
digitalWrite(ledPin, HIGH);
```

If the level signal is changed to LOW, the ledPin's signal will be returned to 0 V to turn LED off.

```
digitalWrite(ledPin, LOW);
```

- digitalWrite()

An interval between on and off is required to allow people to see the change, so we use a `delay(1000)` code to let the controller do nothing for 1000 ms.

```
delay(1000);
```

4.2.2 Fading LED

In last projects, we have used only two output signals: high level and low level (or called 1 & 0, ON & OFF), which is called digital output.
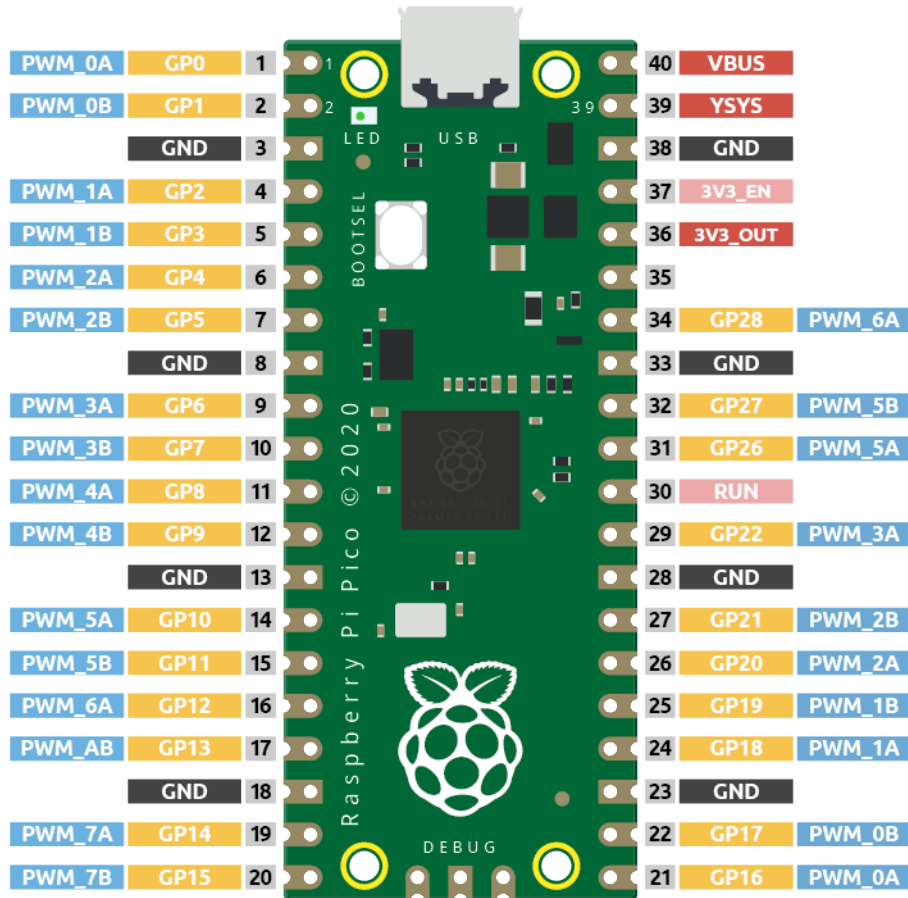
However, in actual use, many devices do not simply ON/OFF to work, for example, adjusting the speed of the motor, adjusting the brightness of the desk lamp, and so on. In the past, a slider that can adjust the resistance was used to achieve this goal, but this is always unreliable and inefficient.

Therefore, Pulse width modulation (PWM) has emerged as a feasible solution to such complex problems. A digital output composed of a high level and a low level is called a pulse. The pulse width of these pins can be adjusted by changing the ON/OFF speed.

Simply put, when we are in a short period (such as 20ms, most people's visual retention time), Let the LED turn on, turn off, and turn on again, we won't see it has been turned off, but the brightness of the light will be slightly weaker. During this period, the more time the LED is turned on, the higher the brightness of the LED. In other words, in the cycle, the wider the pulse, the greater the "electric signal strength" output by the microcontroller. This is how PWM controls LED brightness (or motor speed).

- Pulse-width modulation - Wikipedia

There are some points to pay attention to when Pico uses PWM. Let's take a look at this picture.

Each GPIO pin of Pico supports PWM, but it actually has a total of 16 independent PWM outputs (instead of 30), distributed between GP0 to GP15 on the left, and the PWM output of the right GPIO is equivalent to the left copy.
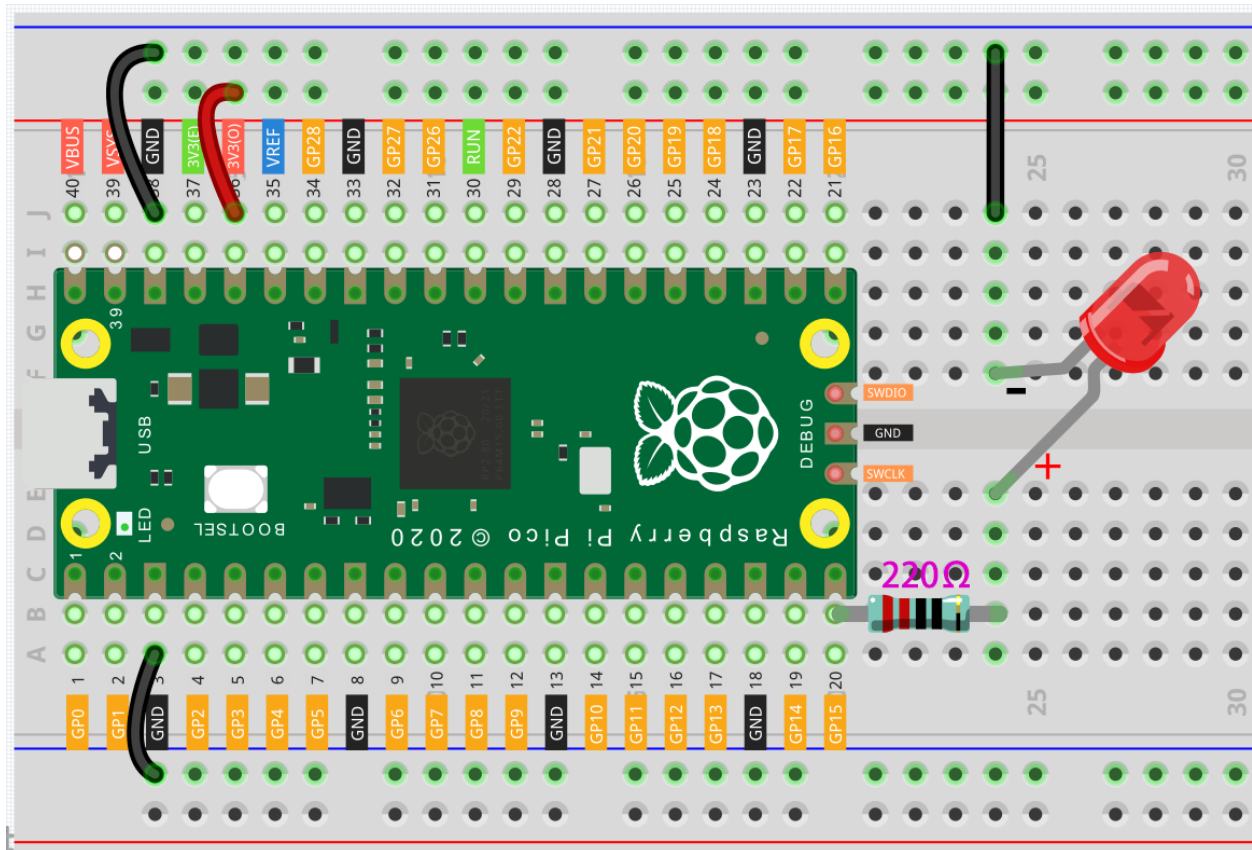
What we need to pay attention to is to avoid setting the same PWM channel for different purposes during programming. (For example, GP0 and GP16 are both PWM_0A)

After understanding this knowledge, let us try to achieve the effect of Fading LED.

Schematic
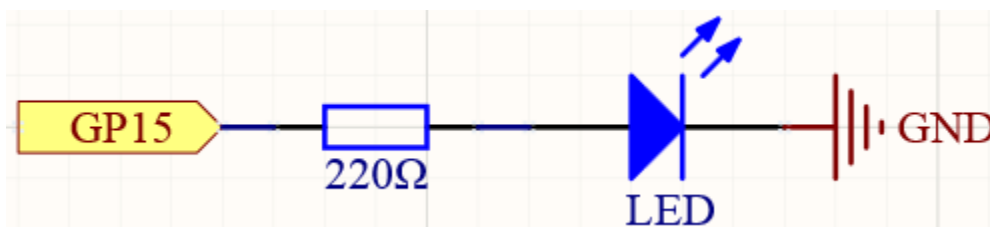
Wiring

- RP2040 Datasheet

1. Here we use the GP15 pin of the Pico board.

2. Connect one end (either end) of the 220 ohm resistor to GP15, and insert the other end into the free row of the breadboard.

3. Insert the anode lead of the LED into the same row as the end of the 220 resistor, and connect the cathode lead across the middle gap of the breadboard to the same row.

4. Connect the LED cathode to the negative power bus of the breadboard.

5. Connect the negative power bus to the GND pin of Pico.

Note: The color ring of the 220 ohm resistor is red, red, black, black and brown.

Code

When the program is running, the LED will alternate between gradually brightening and gradually dimming.

How it works?

As in the previous project, when using a pin, its input/output mode needs to be defined first. Then we can write values to it, a function analogWrite() in `loop()` assigns ledPin a PWM wave (brightness)which will add or decrease 5 for each cycle.

```
analogWrite(ledPin, brightness);

// change the brightness for next time through the loop:
brightness = brightness + fadeAmount;
```

Use an if statement to limit the range of `brightness` to 0-255, the `brightness` will accumulate from 0 to 255 and then decrement from 255 to 0, alternating repeatedly.

```
if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
}
```
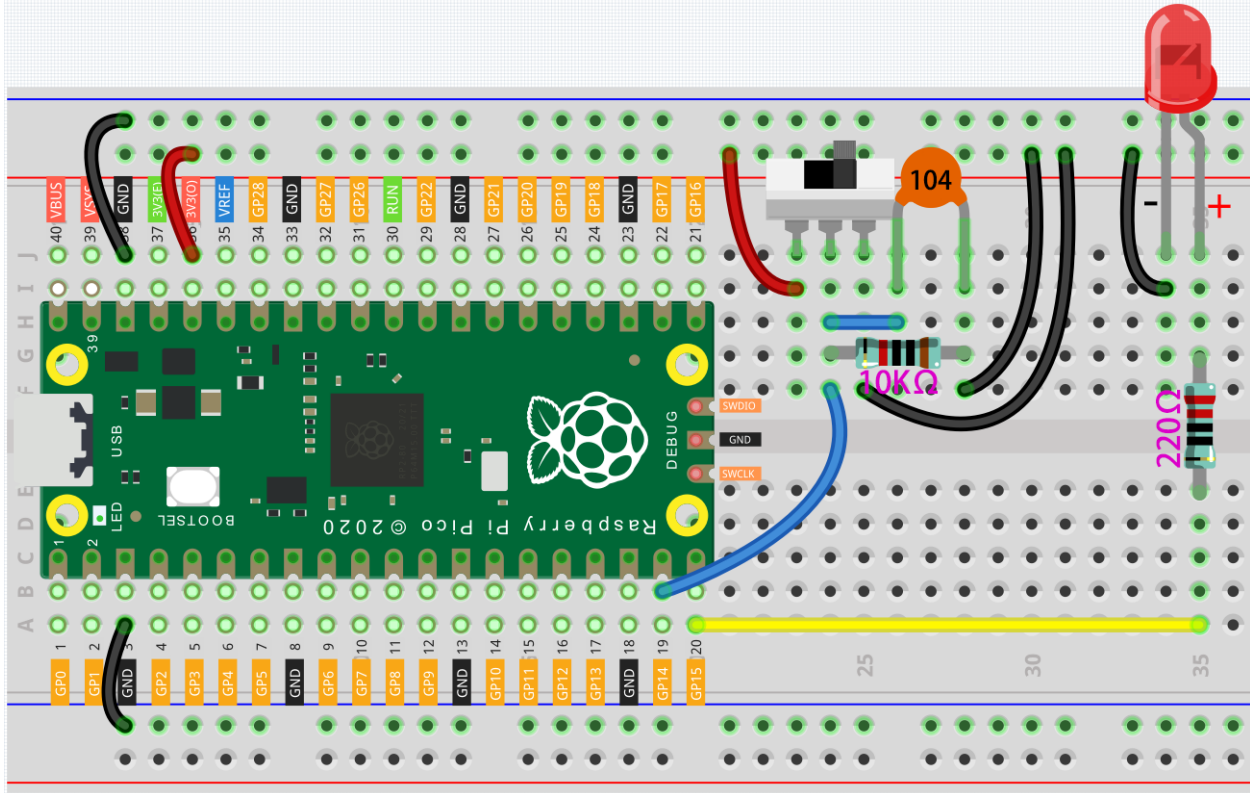
- <=

- >=

- ||

4.2.3 Warning Light

From the name of GPIO (General-purpose input/output), we can see that these pins have both input and output functions. In the previous two projects we used the output function, in this project, we will use the input function to input the Slide value, and then control the LED to blink, like a warning light.

Schematic
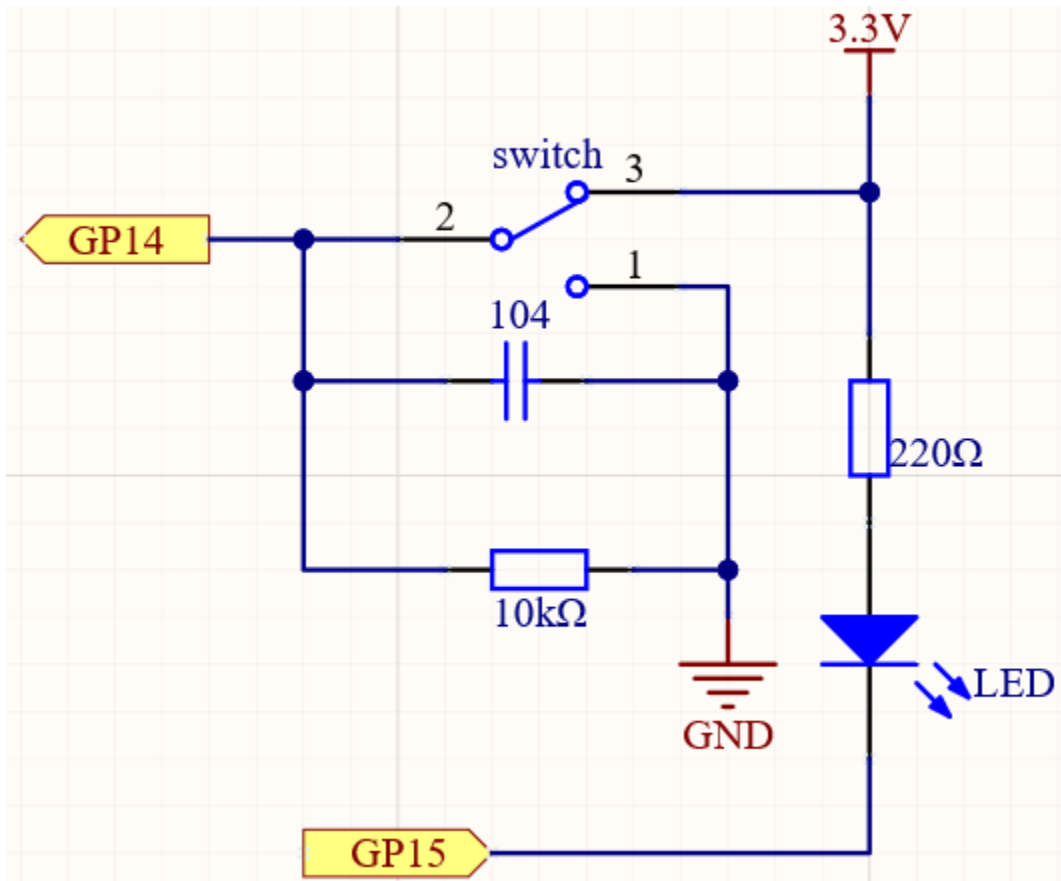
Wiring

1. Connect the 3V3 pin of Pico to the positive power bus of the breadboard.

2. Connect one end (either end) of the 220 ohm resistor to GP15, and insert the other end into the free row of the breadboard.

3. Insert the anode lead of the LED into the same row as the end of the 220 resistor, and connect the cathode lead across the middle gap of the breadboard to the same row.

4. Connect the LED cathode to the negative power bus of the breadboard.

5. Insert the slide switch into the breadboard.

6. Use a jumper wire to connect one end of slide switch pin to the negative bus.

7. Connect the middle pin to GP14 with a jumper wire.

8. Use a jumper wire to connect last end of slide switch pin to the positive bus

9. Use a 10K resistor to connect the middle pin of the slide switch and the negative bus.

10. Use a 104 capacitor to connect the middle pin of the slide switch and the negative bus to realize debounce that may arise from your toggle of switch.

11. Connect the negative power bus of the breadboard to Pico's GND.

When you toggle the slide switch, the circuit will switch between closed and open.

- *Slide Switch*
- *Capacitor*

Code

After the code has run, toggle the Slide switch to one side and the LED will flash. Toggle to the other side and the LED will go out.

How it works?

For switchs, we need to set their mode to INPUT in order to be able to get their values.

```
void setup() {
    // initialize the LED pin as an output:
    pinMode(ledPin, OUTPUT);
    // initialize the switch pin as an input:
    pinMode(switchPin, INPUT);
}
```

Read the status of the switchPin in loop() and assign it to the variable switchState.

```
switchState = digitalRead(switchPin);
```

- digitalRead()

If the switchState is HIGH, the LED will flash.

```
if (switchState == HIGH)
{
    // turn LED on:
    digitalWrite(ledPin, HIGH);
    delay(1000);
    digitalWrite(ledPin, LOW);
    delay(1000);
}
```
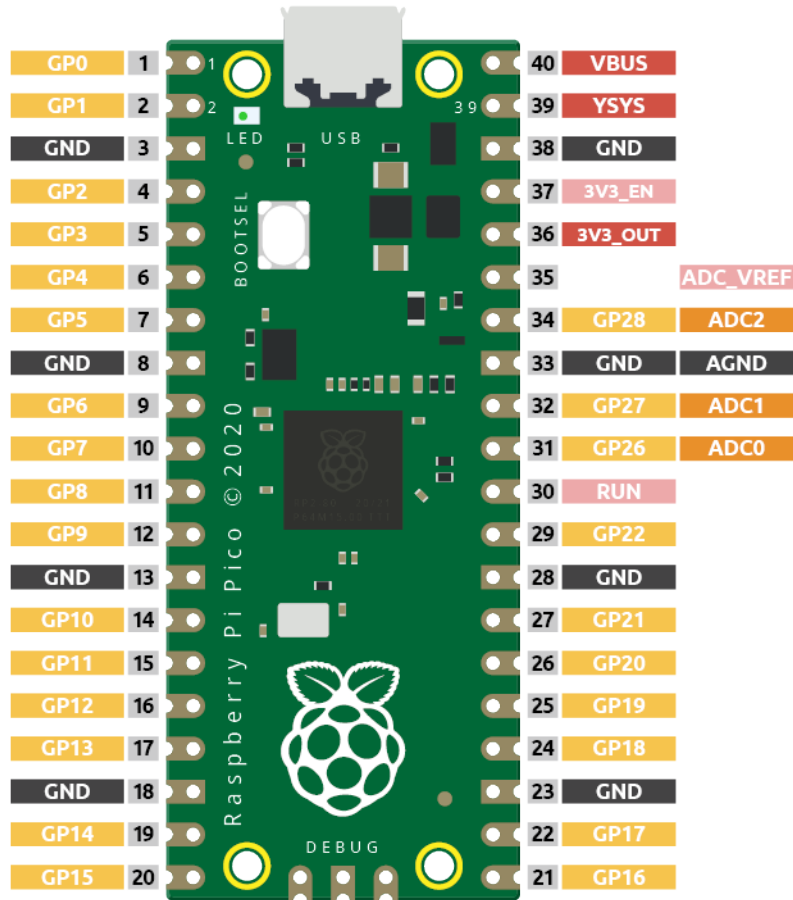
Otherwise, turn off the LED.

```
else
{
    digitalWrite(ledPin, LOW);
}
```

4.2.4 Table Lamp

In the previous projects, we have used the digital input on the Pico. For example, a button can change the pin from low level (off) to high level (on). This is a binary working state.

However, Pico can receive another type of input signal: analog input. It can be in any state from fully closed to fully open, and has a range of possible values. The analog input allows the microcontroller to sense the light intensity, sound intensity, temperature, humidity, etc. of the physical world.

Usually, a microcontroller needs an additional hardware to implement analog input-the analogue-to-digital converter (ADC). But Pico itself has a built-in ADC for us to use directly.
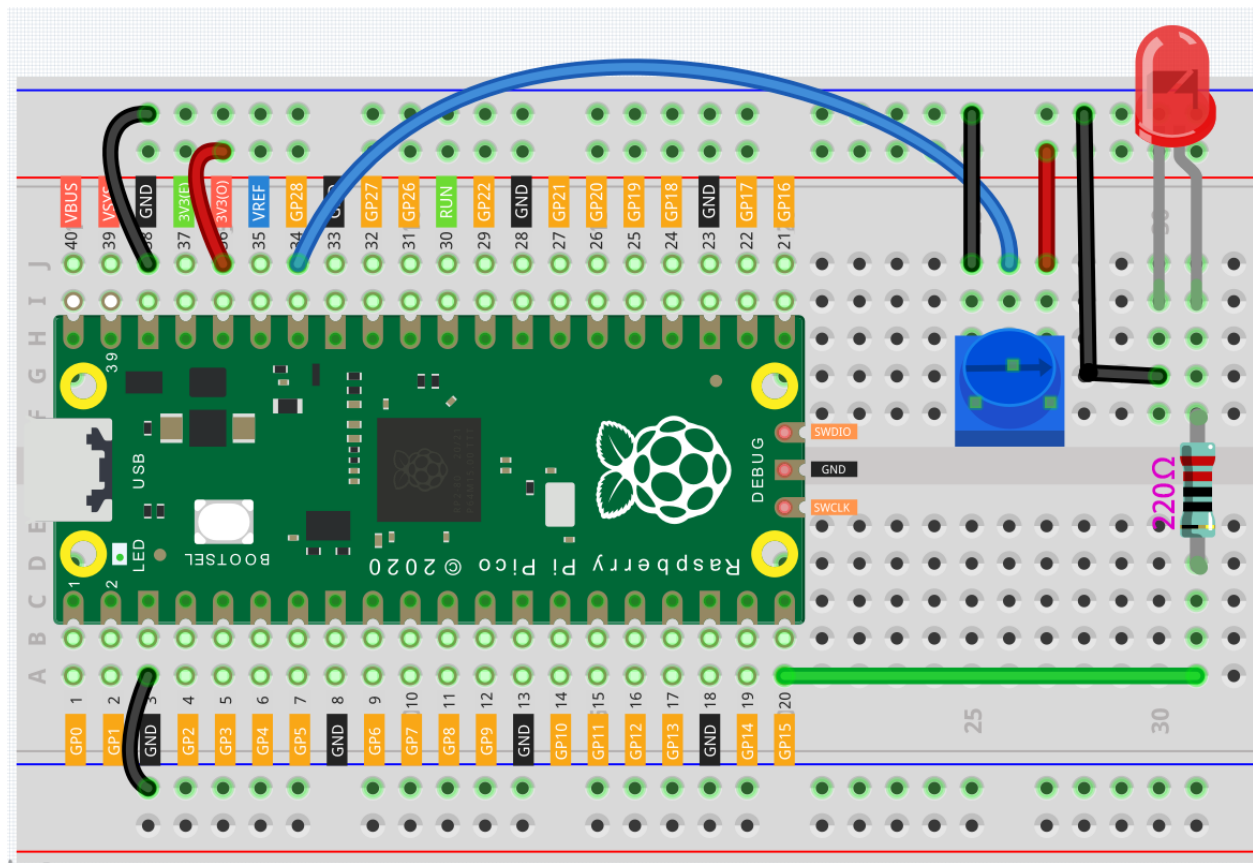
Pico has three GPIO pins that can use analog input, GP26, GP27, GP28. That is, analog channels 0, 1, and 2. In addition, there is a fourth analog channel, which is connected to the built-in temperature sensor and will not be introduced here.

In this project, we try to read the analog value of potentiometer.

Schematic
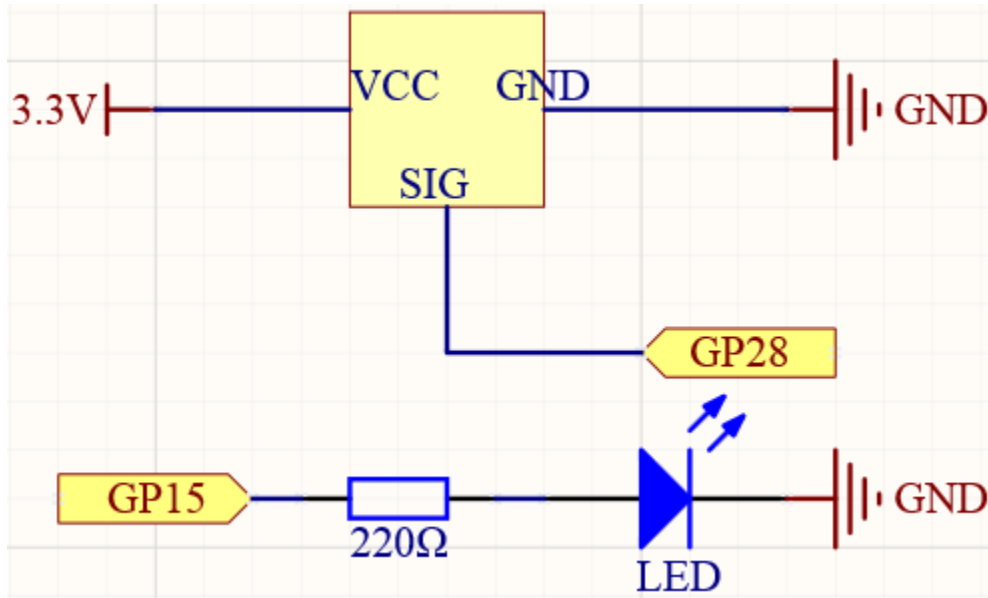
Wiring

1. Connect 3V3 and GND of Pico to the power bus of the breadboard.

2. Insert the potentiometer into the breadboard, its three pins should be in different rows.

3. Use jumper wires to connect the pins on both sides of the potentiometer to the positive and negative power bus respectively.

4. Connect the middle pin of the potentiometer to GP28 with a jumper wire.

5. Connect the anode of the LED to the GP15 pin through a 220 resistor, and connect the cathode to the negative power bus.

Code

Once the code has been successfully uploaded, open Serial Monitor and then ensure that baudrate is 9600, rotate the potentiometer in 2 directions and you will see that the value range is 0-1023.

When the displayed value is 0, the LED goes off and as the value increases, the LED gets brighter.
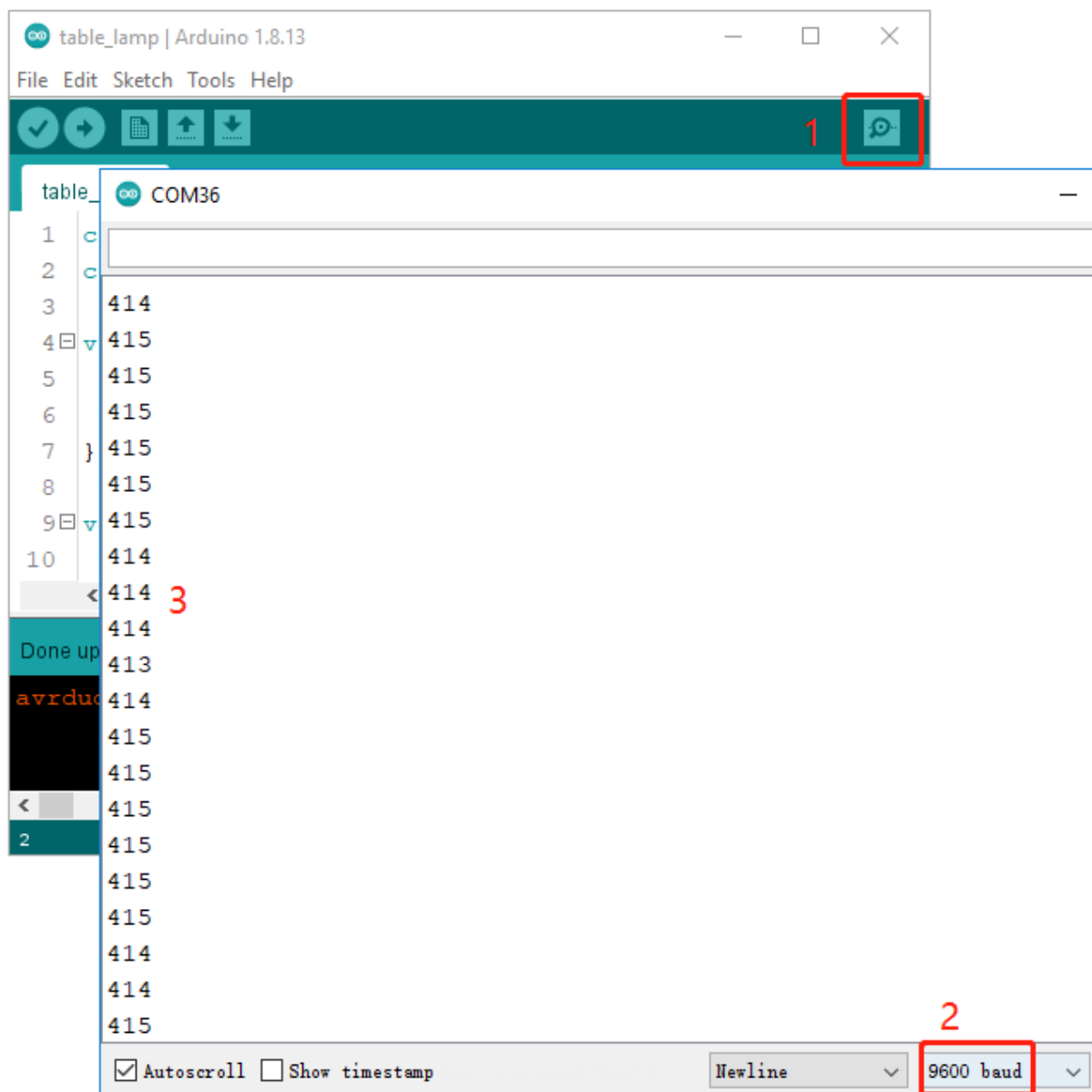
Note: If your code is OK and you have selected the correct board and port, but the upload is still not successful.

At this point you can click on the **Upload** icon again when the progress below shows "Upload. . .", unplug the USB cable again and plug it in and the code will be uploaded successfully.

How it works?

To enable Serial Monitor, you need to start serial communication in `setup()` and set the datarate to 9600.

```
void setup() {
    pinMode(ledPin, OUTPUT);
    Serial.begin(9600);
}
```

- Serial

In the loop function, the value of the potentiometer is read, then the value is mapped from 0-1023 to 0-255 and finally the value after the mapping is used to control the brightness of the LED.

```
void loop() {
    int sensorValue = analogRead(sensorPin);
    Serial.println(sensorValue);
    int brightness = map(sensorValue, 0, 1023, 0, 255);
    analogWrite(ledPin, brightness);
}
```

- analogRead() is used to read the value of the sensorPin (potentiometer) and assigns it to the variable `sensorValue`.

```
int sensorValue = analogRead(sensorPin);
```

- Print the value of SensorValue in Serial Monitor.

```
Serial.println(sensorValue);
```

- Here, the map(value, fromLow, fromHigh, toLow, toHigh) function is required as the potentiometer value read is in the range 0-1023 and the value of a PWM pin is in the range 0-255. It is used to Re-maps a number from one range to another. That is, a value of fromLow would get mapped to toLow, a value of fromHigh to toHigh, values in-between to values in-between, etc.

```
int brightness = map(sensorValue, 0, 1023, 0, 255);
```

- Now we can use this value to control the brightness of the LED.

```
analogWrite(ledPin,brightness);
```

4.2.5 Measure Light Intensity

A photoresistor or photocell is a light-controlled variable resistor. The resistance of a photo resistor decreases with increasing incident light intensity; in other words, it exhibits photo conductivity.
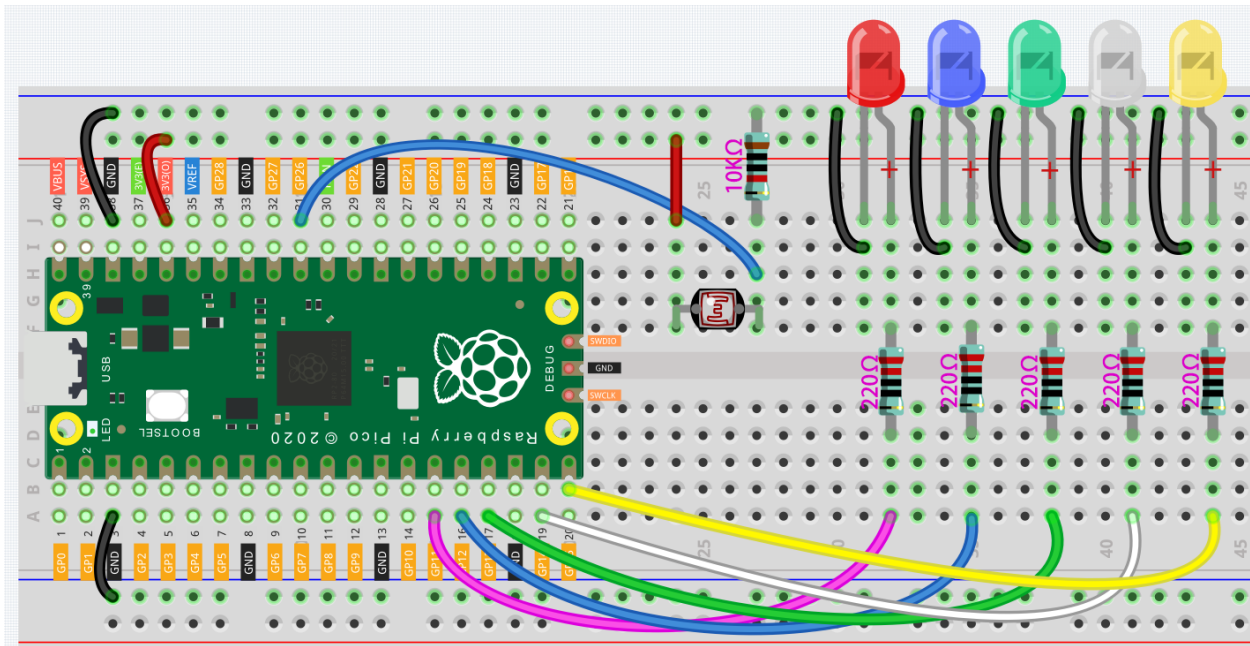
Therefore, we can use a photoresistor to measure light intensity, and then show it through 5 LEDs.

Schematic
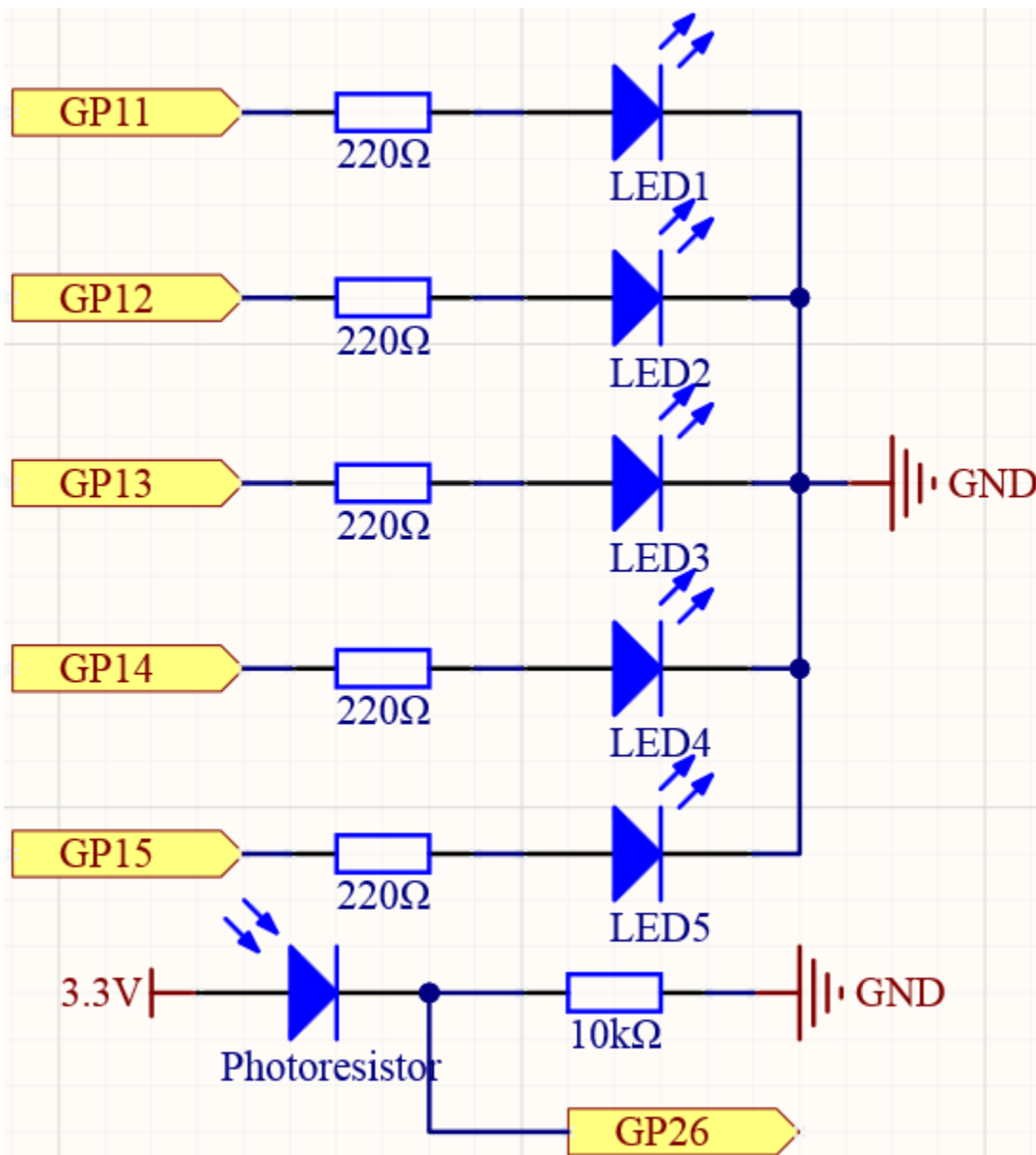
Wiring

In this experiment, we will use 5 LEDs to show the light intensity. The higher the light intensity is, the more LEDs will light up, vice versa.

Code

How it works?

```
const int ledPins[] = {11, 12, 13, 14, 15};
const int photocellPin = A0;  //photoresistor attach to A0
int sensorValue = 0;        // value read from the sensor
int Level = 0;             // sensor value converted into LED 'bars'
```

First of all, there are still various initialization definitions, setting pins and setting initial values of variables.

In order to quickly set the input/output status and HIGH/LOW for the 5 LEDs in the following code, here we use the array `ledPin[]` to define the 5 LEDs connected to the corresponding pins of the Pico.

The element number of the array usually starts from 0. For example, `ledPin[0]` refers to GPIO11, and `ledPin[4]` refers to GPIO15.

- array

```
void setup()
{
  Serial.begin(9600);  // start serial port at 9600 bps:
  for (int i = 0; i < 5; i++)
  {
    pinMode(ledPins[i], OUTPUT);// make all the LED pins outputs
  }
}
```

In `setup()` using the `for()` statement set the 5 pins to OUTPUT. The variable `i` is added from 0 to 5, and the `pinMode()` function sets pin11 to pin15 to OUTPUT in turn.

```
sensorValue = analogRead(photocellPin); //read the value of A0
Level = map(sensorValue, 0, 1023, 0, 5);  // map to the number of LEDs
Serial.println(Level);
delay(10);
```

In `loop()`, read the analog value of the photocellPin(A0) and store to the variable sensorValue.

The `map()` function is used to map 0-1023 to 0-5. It means that the value range of the photoresistor (0-1023) is equally divided into 5 levels, 0-204.8 belongs to Level 0, 204.9-409.6 belongs to Level 1, and 819.2-1023 belong to Level 4. If the value of variable `sensorValue` is 300 at this time, then `Level` is equal to 1.

- map()

```
for (int i = 0; i < 5; i++)
{
  if (i <= Level ) //When i is smaller than Level, run the following code.
  {
    digitalWrite(ledPins[i], HIGH);     // turn on LEDs less than the level
  }
  else
  {
    digitalWrite(ledPins[i], LOW);     // turn off LEDs higher than level
  }
}
```

Now we need to find a way to display the brightness level at this time with LEDs.

The `for()` statement is used here to perform loop detection in the `ledPin[]` array. If the element bit in the array is less than the value of `Level`, the corresponding GPIO is set to high level, that is, the corresponding LED is lit. If `Level` is equal to 1, turn on the LEDs on GPIO11 and GPIO12.

4.2.6 Doorbell

A buzzer is a great tool in your experiments whenever you want to make some sounds. In this project, we will learn how to drive a passive buzzer to build a simple doorbell.

Two types of buzzers are included in the kit. We need to use passive buzzer. Turn them around, the one with exposed PCB is we want.
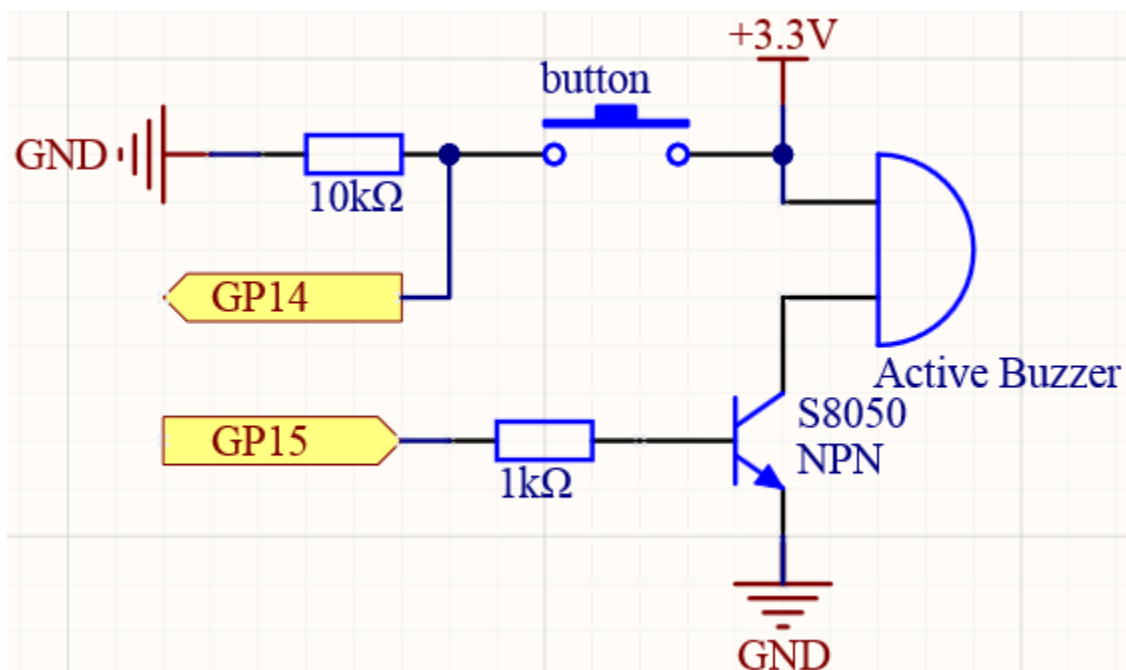

The buzzer needs to use a transistor when working, here we use S8050.
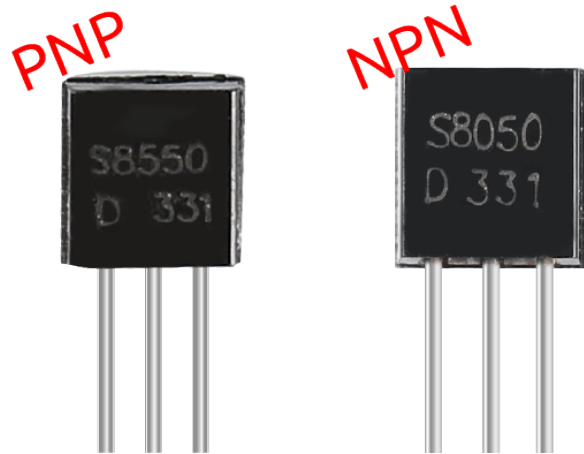
- *Transistor*
- *Button*

Schematic

Wiring

- The basic function of the triode is to amplify and switch. In this circuit, the switch function of the NPN triode is used to drive the buzzer. When a high-level signal is given to the base of the NPN transistor, the transistor is turned on and the buzzer can emit a sound.

- When a pin is floating, its level may be high or low, so we connect a 10K resistor in the upper left corner of the button to keep the button in a low state when it is not pressed. You can also try to remove this 10K resistor, and you will find that even if it is not pressed, the buzzer may sound.
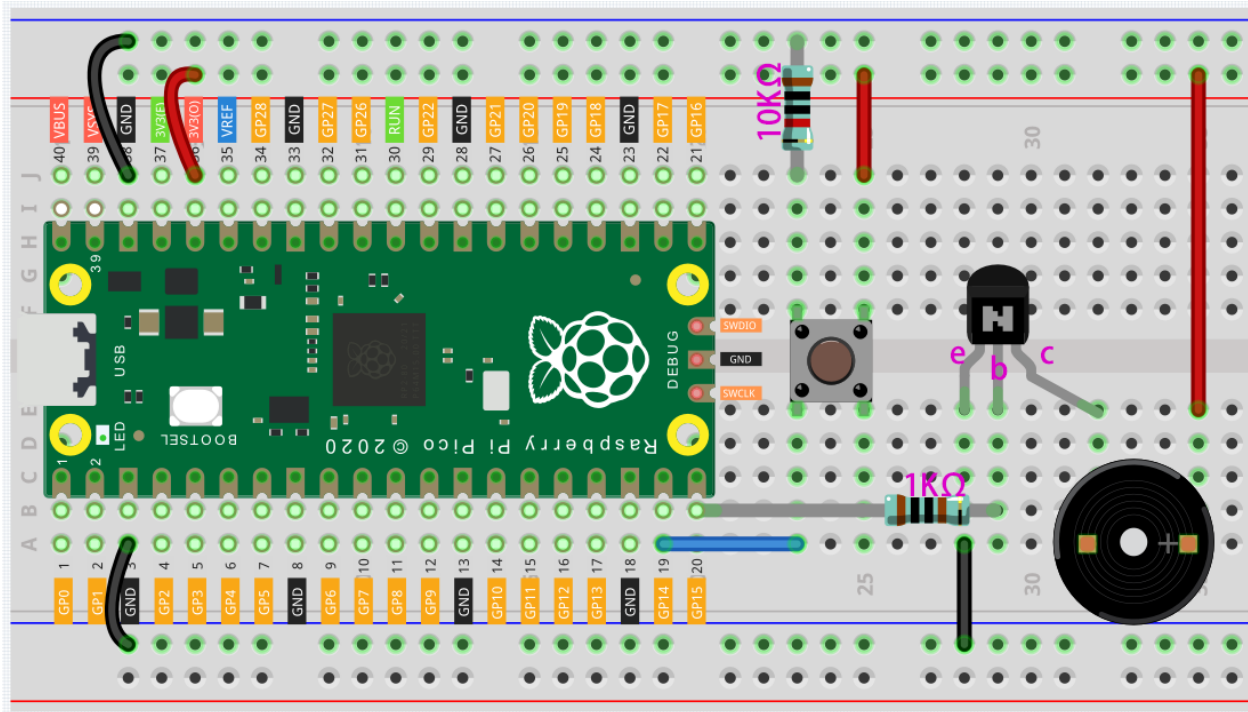
Code

After uploading the code, when you press the button, you can hear a 7-note melody.

How it works?

```
#include "pitches.h"
```

The code uses an extra file, `pitches.h`. This file contains all the pitch values for typical notes. For example, NOTE_C4 is middle C. NOTE_FS4 is F sharp, and so forth. This note table was originally written by Brett Hagman, on whose work the tone() command was based. You may find it useful whenever you want to make musical notes.

Note: There is already a pitches.h file in this sample program. If you cannot find pitches.h after downloading or opening the code, you can just install one manually.

Click the button below the serial monitor icon and select "New Tab", or use Ctrl+Shift+N.
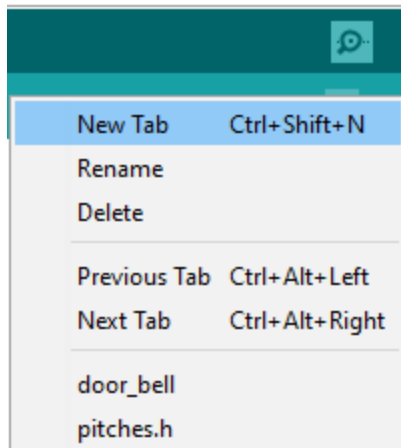
Then paste from *Code* and save it as `pitches.h`:

```
int melody[] = {NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4};

int buttonPin = 14;

//note durations. 4=quarter note / 8=eighth note
int noteDurations[] = {4, 8, 8, 4, 4, 4, 4, 4};
```

The array `melody[]` stores 7 notes, and `noteDurations[]` is the duration corresponding to these notes, 4=quarter note, 8=eighth note.

- Quarter note

- Eighth note

```
int buttonState = digitalRead(buttonPin);   //read the input pin
  //if the button is pressed
  if (buttonState == 1) {
    //iterate over the notes of the melody
    for (int i = 0; i < 8; i++) {

      // to calculate the note duration, take one second divided by the note type.
      //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
      int noteDuration = 1000 / noteDurations [i];
      tone(15, melody [i], noteDuration);
      //to distinguish the notes, set a minimum time between them
      //the note's duration +30% seems to work well
      int pauseBetweenNotes = noteDuration * 1.30;
      delay(pauseBetweenNotes);
    }
  }
```

First read the value of the button in loop(). When the button is pressed, `buttonState` will be equal to 1.

In the for() statement, a tone() is used to let the buzzer play one note at a time, and then after 8 times, the buzzer can play the notes in the array melody[] one by one.

```
tone(15, melody [i], noteDuration);
```

- **15**: The pin on which to generate the tone (the buzzer pin).

- **melody [i]**: The frequency of the tone in hertz.

- **noteDuration**: The duration of the tone in milliseconds (optional).

```
else
{
  noTone(15);     //if the button is released, stop the tone playing.
}
```

Stops the generation of a square wave triggered by tone(). Has no effect if no tone is being generated.

4.2.7 RGB LED

As we know, light can be superimposed. For example, mix blue light and green light give cyan light, red light and green light give yellow light. This is called "The additive method of color mixing".

- Additive color - Wikipedia

Based on this method, we can use the three primary colors to mix the visible light of any color according to different specific gravity. For example, orange can be produced by more red and less green.

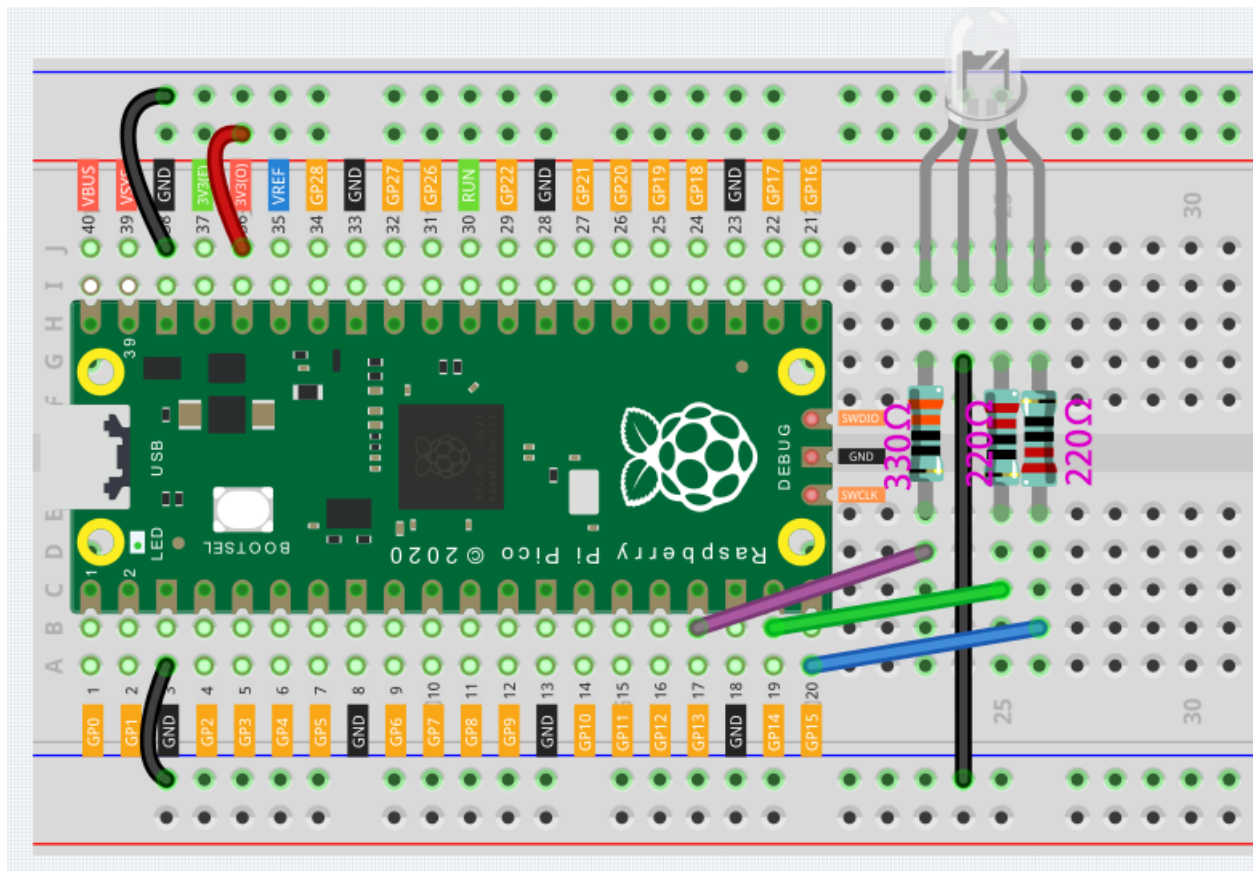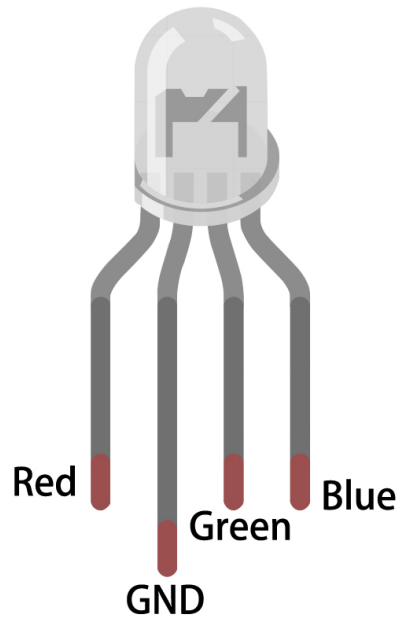In this chapter, we will use RGB LED to explore the mystery of additive color mixing!

RGB LED is equivalent to encapsulating Red LED, Green LED, Blue LED under one lamp cap, and the three LEDs share one cathode pin. Since the electric signal is provided for each anode pin, the light of the corresponding color can be displayed. By changing the electrical signal intensity of each anode, it can be made to produce various colors.

Schematic

Wiring

Put the RGB LED flat on the table, we can see that it has 4 leads of different lengths. Find the longest one (GND) and turn it sideways to the left. Now, the order of the four leads is Red, GND, Green, Blue from left to right.

1. Connect the GND pin of the Pico to the negative power bus of the breadboard.

2. Insert the RGB LED into the breadboard so that its four pins are in different rows.

3. Connect the red lead to the GP13 pin via a 330 resistor. When using the same power supply intensity, the Red LED will be brighter than the other two, and a slightly larger resistor needs to be used to reduce its brightness.

4. Connect the Green lead to the GP14 pin via a 220 resistor.

5. Connect the Blue lead to the GP15 pin via a 220 resistor.

6. Connect the GND lead to the negative power bus.

7. Connect the negative power bus to Pico's GND.

Note:

- The color ring of the 220 resistor is red, red, black, black and brown.

- The color ring of the 330 resistor is orange, orange, black, black and brown.

Code

Here, we can choose our favorite color in drawing software and display it with RGB LED.
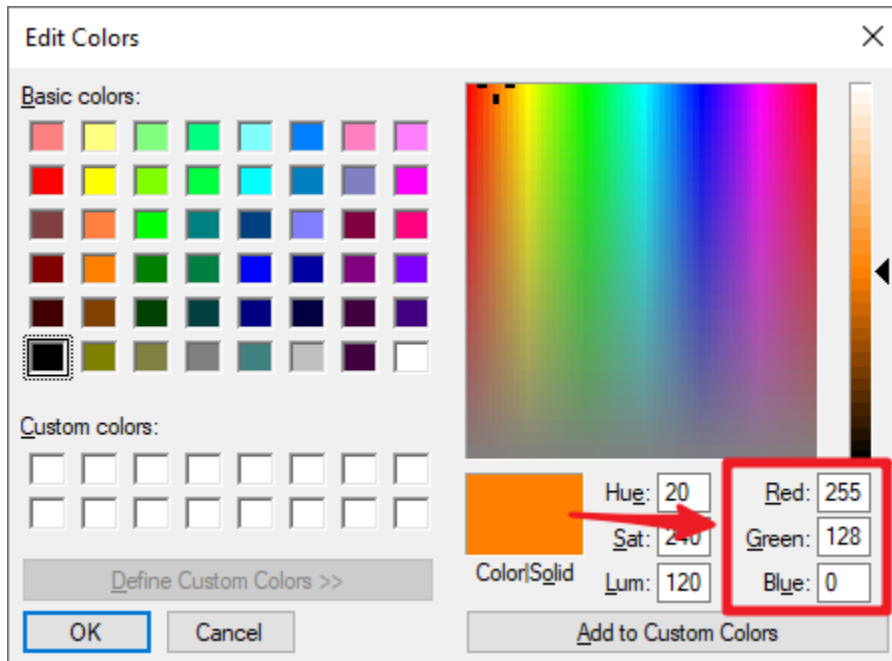
Write the RGB value into color(), you will be able to see the RGB light up the colors you want.

How it works?

```
void color (unsigned char red, unsigned char green, unsigned char blue)// the color
↪generating function
{
    analogWrite(redPin, red);
    analogWrite(greenPin, green);
    analogWrite(bluePin, blue);
}
```

We defined a `color()` function to let the three primary colors work together.

At present, pixels in computer hardware usually adopt a 24-bit representation method. The three primary colors are divided into 8 bits, and the color value range is 0 to 255. With 256 possible values for each of the three primary colors (don't forget to count 0!), that 256 x 256 x 256 = 16,777,216 colors can be combined in this way. The `color()` function also follows the 24-bit notation, which makes it easier for us to select the desired color.

`analogWrite(pin, value)` Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to `analogWrite()`, the pin will generate a steady rectangular wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite()) on the same pin.

> `pin`: the Arduino pin to write to. Allowed data types: int.
>
> `value`: the duty cycle: between 0 (always off) and 255 (always on). Allowed data types: int.

- analogWrite(pin, value)
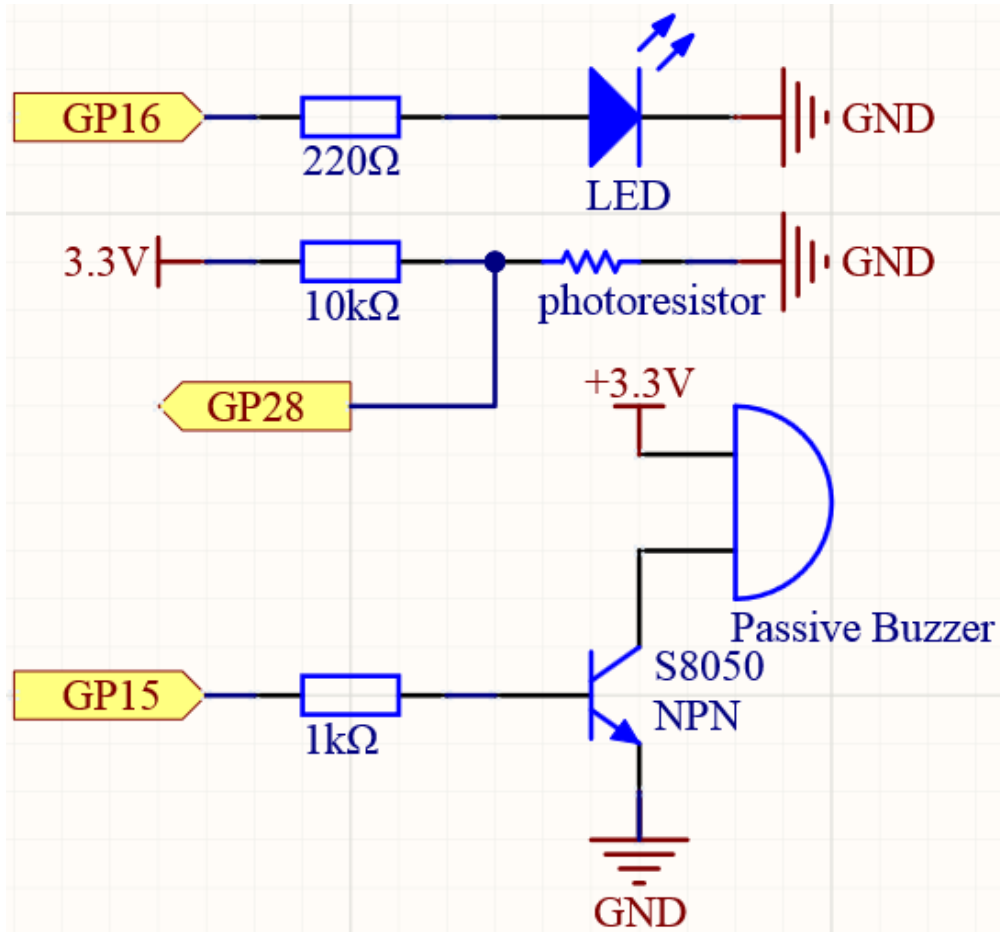
4.2.8 Light Theremin

Theremin is an electronic musical instrument that does not require physical contact. It produces different tones by sensing the position of the player's hand.

The instrument's controlling section usually consists of two metal antennas that sense the relative position of the thereminist's hands and control oscillators for frequency with one hand, and amplitude (volume) with the other. The electric signals from the theremin are amplified and sent to a loudspeaker.
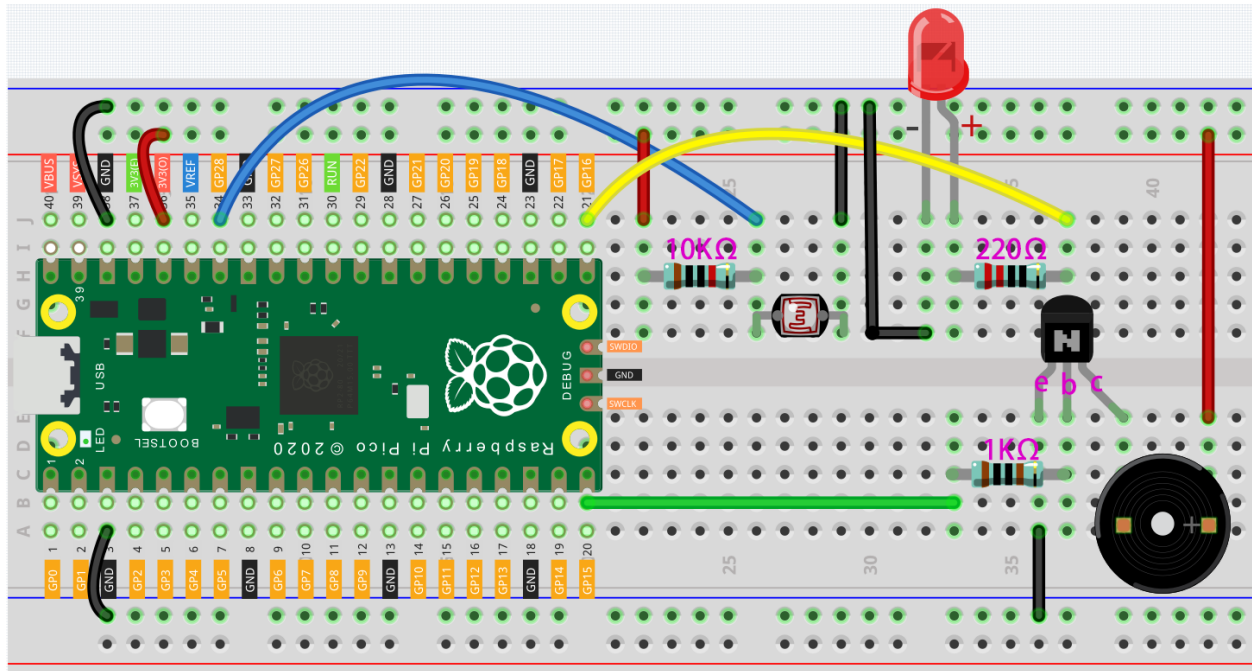
We cannot reproduce the same instrument through Pico, but we can use photoresistor and passive buzzer to achieve similar gameplay.

- Theremin - Wikipedia

Schematic

Wiring

1. Connect 3V3 and GND of Pico to the power bus of the breadboard.

2. Connect one lead of the photoresistor to the GP28 pin, then connect the same lead to the positive power bus with a 10K ohm resistor.

3. Connect another lead of photoresistor to the negative power bus.

4. Insert the LED into the breadboard, connect its anode pin to the GP16 in series with a 220 resistor, and connect its cathode pin to the negative power bus.

5. Insert the passive buzzer and S8050 transistor into the breadboard. The anode pin of the buzzer is connected to the positive power bus, the cathode pin is connected to the **collector** lead of the transistor, and the **base** lead of the transistor is connected to the GP15 pin through a 1k resistor. **emitter** lead is connected to the negative power bus.

Note:

- The color ring of the 220 resistor is red, red, black, black and brown.

- The color ring of the 10k resistor is brown, black, black, red and brown.

Code

When the program runs, the LED will light up, and we will have one seconds to calibrate the detection range of the photoresistor. This is because we may be in a different light environment each time we use it (e.g. the light intensity is different between midday and dusk).

At this time, we need to swing our hands up and down on top of the photoresistor, and the movement range of the hand will be calibrated to the playing range of this instrument.

After one seconds, the LED will go out and we can wave our hands on the photoresistor to play.

How it works?

```
/* calibrate the photoresistor max & min values */
previousMillis = millis();
digitalWrite(ledPin, HIGH);
while (millis() - previousMillis <= interval) {
  sensorValue = analogRead(photocellPin);
  if (sensorValue > lightHigh) {
    lightHigh = sensorValue;
  }
  if (sensorValue < lightLow) {
    lightLow = sensorValue;
  }
}
digitalWrite(ledPin, LOW);
```

Set up a calibration process in `setup()`, using `millis()` for timing with an interval of 1s (`interval = 1000`). During this second, wave a hand around the photoresistor, the maximum and minimum values of the detected light are recorded and assigned to `lightHigh` and `lightLow` respectively.

- millis()

```
sensorValue = analogRead(photocellPin); //read the value of A0
pitch = map(sensorValue, lightLow, lightHigh, 50, 6000);
if (pitch > 50) {
  tone(buzzerPin, pitch, 20);
}
```

Read the value of the photocell resistor to `sensorValue` and map it from the small range to the large range to be used to control the frequency of the buzzer. For a detailed explanation of `map()`, please refer to the course *Measure Light Intensity*.

```
tone(buzzerPin, pitch, 20)
```

This function can make the buzzer sound, the frequency is `pitch`, and the duration is 20 milliseconds.

4.2.9 74HC595

74HC595

Integrated circuit (integrated circuit) is a kind of miniature electronic device or component, which is represented by the letter "IC" in the circuit.

A certain process is used to interconnect the transistors, resistors, capacitors, inductors and other components and wiring required in a circuit, fabricate on a small or several small semiconductor wafers or dielectric substrates, and then package them in a package , it has become a micro-structure with the required circuit functions; all of the components have been structured as a whole, making electronic components a big step towards micro-miniaturization, low power consumption, intelligence and high reliability.

The inventors of integrated circuits are Jack Kilby (integrated circuits based on germanium (Ge)) and Robert Norton Noyce (integrated circuits based on silicon (Si)).

This kit is equipped with an IC, 74HC595, which can greatly save the use of GPIO pins. Specifically, it can replace 8 pins for digital signal output by writing an 8-bit binary number.

- Binary number - Wikipedia

- *74HC595*
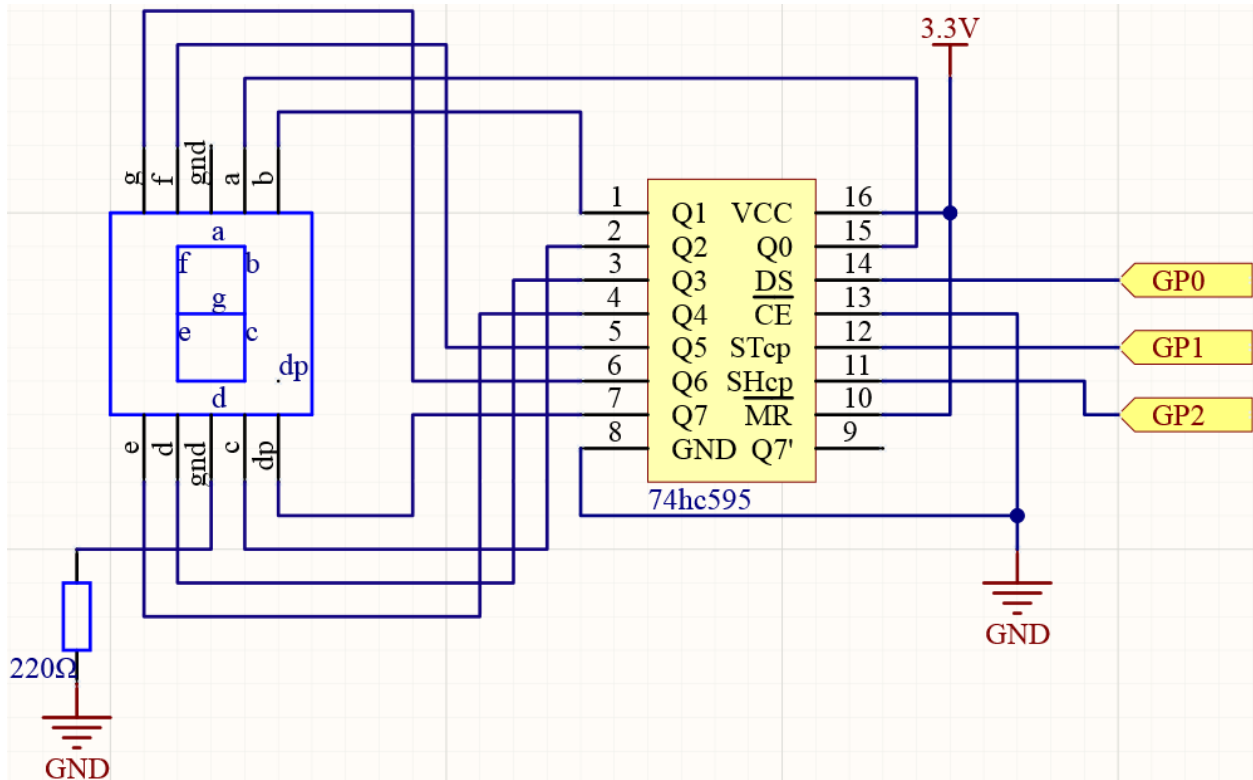
LED Segment Display

LED Segment Display can be seen everywhere in life. For example, on an air conditioner, it can be used to display temperature; on a traffic indicator, it can be used to display a timer.

The LED Segment Display is essentially a device packaged by 8 LEDs, of which 7 strip-shaped LEDs form an "8" shape, and there is a slightly smaller dotted LED as a decimal point. These LEDs are marked as a, b, c, d, e, f, g, and dp. They have their own anode pins and share cathodes. Their pin locations are shown in the figure below.
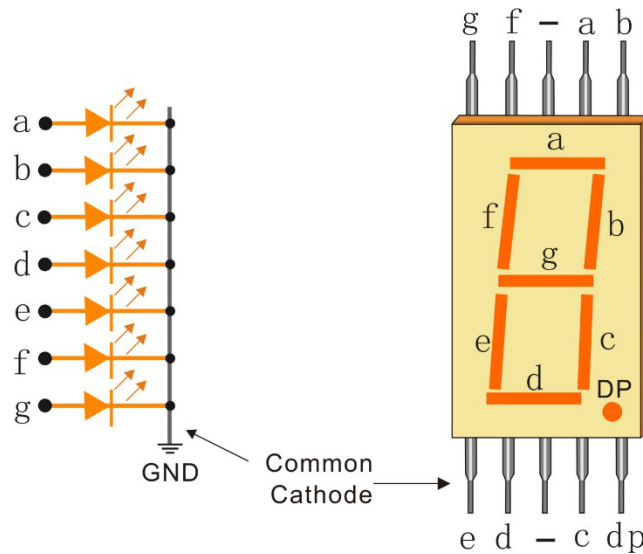
This means that it needs to be controlled by 8 digital signals at the same time to fully work and the 74HC595 can do this.

Schematic

Wiring

1. Connect 3V3 and GND of Pico to the power bus of the breadboard.

2. Insert 74HC595 across the middle gap into the breadboard.

3. Connect the GP0 pin of Pico to the DS pin (pin 14) of 74HC595 with a jumper wire.

4. Connect the GP1 pin of Pico to the STcp pin (12-pin) of 74HC595.

5. Connect the GP2 pin of Pico to the SHcp pin (pin 11) of 74HC595.

6. Connect the VCC pin (16 pin) and MR pin (10 pin) on the 74HC595 to the positive power bus.

7. Connect the GND pin (8-pin) and CE pin (13-pin) on the 74HC595 to the negative power bus.

8. Insert the LED Segment Display into the breadboard, and connect a 220 resistor in series with the GND pin to the negative power bus.

Note: The color ring of the 220 ohm resistor is red, red, black, black and brown.

---

9. Follow the table below to connect the 74hc595 and LED Segment Display.

Table 1: Wiring

74HC595	LED Segment Display
Q0	a
Q1	b
Q2	c
Q3	d
Q4	e
Q5	f
Q6	g
Q7	dp

Code

When the program runs, the 7-digit digital tube will start to display numbers from 0 to 9 in sequence.

How it works?

```
for(int num = 0; num <=9; num++)
{
    digitalWrite(STcp,LOW); //ground ST_CP and hold low for as long as you are
→transmitting
    shiftOut(DS,SHcp,MSBFIRST,datArray[num]);
    digitalWrite(STcp,HIGH); //pull the ST_CPST_CP to save the data
    delay(500); //wait for a second
}
```

`shiftOut(dataPin, clockPin, bitOrder, value)` will make 74HC595 output 8 digital signals. It outputs the last bit of the binary number to Q0, and the output of the first bit to Q7. In other words, writing the binary number "00000001" will make Q0 output high level and Q1~Q7 output low level.

- `dataPin`: the pin on which to output each bit. Allowed data types: int.

- `clockPin`: the pin to toggle once the dataPin has been set to the correct value. Allowed data types: int.

- `bitOrder`: which order to shift out the bits; either MSBFIRST or LSBFIRST. (Most Significant Bit First, or, Least Significant Bit First).

- `value`: the data to shift out. Allowed data types: byte.

- shiftOut()

Suppose that the 7-segment Display display the number "1", we need to write a high level for b, c, and write a low level for a, d, e, f, g, and dg. That is, the binary number "00000110" needs to be written. For readability, we will use hexadecimal notation as "0x06".

- Hexadecimal

- BinaryHex Converter

Similarly, we can also make the LED Segment Display display other numbers in the same way. The following table shows the codes corresponding to these numbers.

Table 2: Glyph Code

Numbers	Binary Code	Hex Code
0	00111111	0x3f
1	00000110	0x06
2	01011011	0x5b
3	01001111	0x4f
4	01100110	0x66
5	01101101	0x6d
6	01111101	0x7d
7	00000111	0x07
8	01111111	0x7f
9	01101111	0x6f

Write these codes into `datArray[num]` of `shiftOut()` to make the LED Segment Display display the corresponding numbers.

4.2.10 Digital Dice

In the last project, we learned how to use 74HC595 to light up the LED Segment Display. Of course, we can expand on this basis to make it more interesting.

Here, we will add a button to the previous project to make a digital dice. When the button is pressed, the 7 segment display will randomly show 1-6.

Schematic
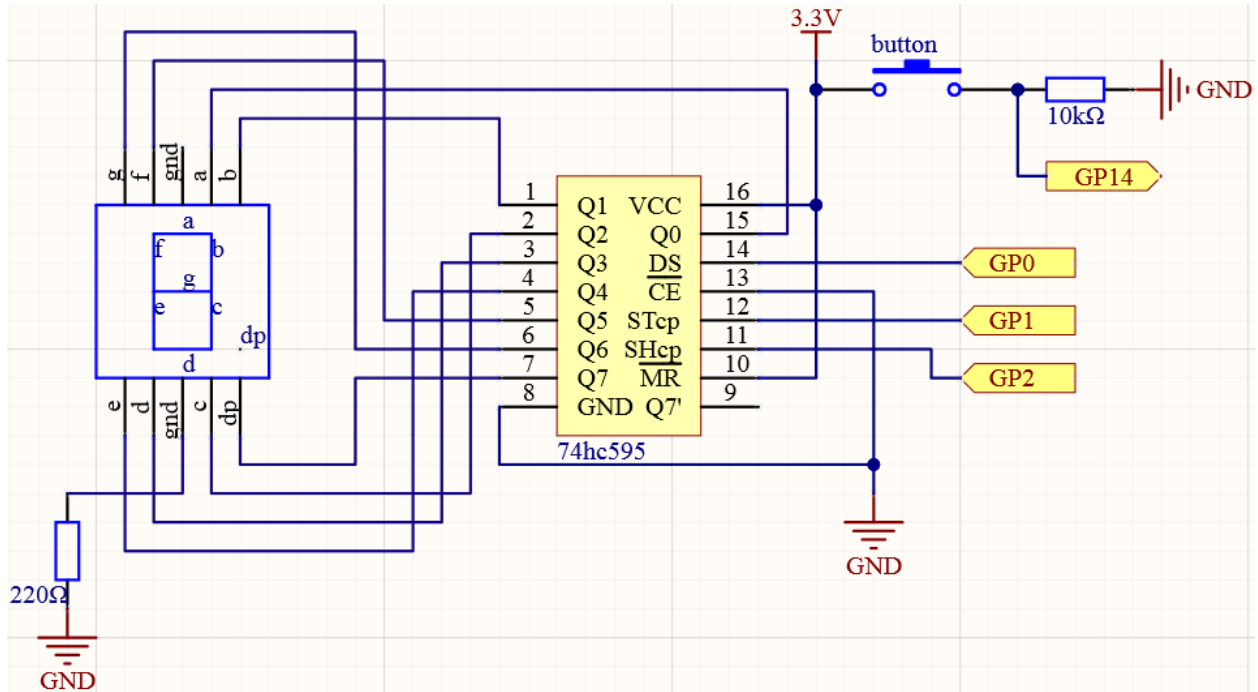
Wiring
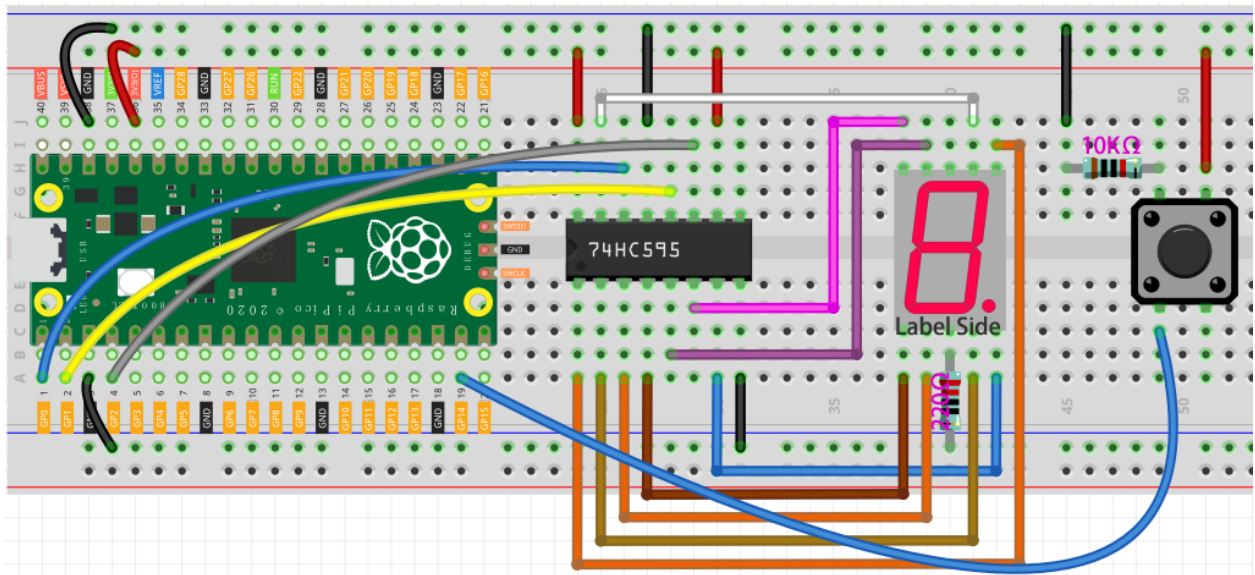
Code

When the program starts running, every time you press the button, the digital tube will randomly display a number from 1 to 6.

How it works?

```
SEGCODE = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f]
```

First, define a hexadecimal array to represent the number from 0 to 9 displayed by the 7 segment display.

```
if (digitalRead(button)==HIGH){
    dice = random(1,7);
}
```

When the value of the button is HIGH, i.e. the key has been pressed, the `random()` function is called to generate a random number from 1 to 6 to be stored in the variable `dice`.

- random

The following table shows the codes corresponding to these numbers.

Table 3: Glyph Code

Numbers	Binary Code	Hex Code
0	00111111	0x3f
1	00000110	0x06
2	01011011	0x5b
3	01001111	0x4f
4	01100110	0x66
5	01101101	0x6d
6	01111101	0x7d
7	00000111	0x07
8	01111111	0x7f
9	01101111	0x6f

`dice = random(1,7)` the random function generates pseudo-random numbers from 1 to 6(Exclude 7).

```
digitalWrite(STcp,LOW); //ground ST_CP and hold low for as long as you are␣
→transmitting
shiftOut(DS,SHcp,MSBFIRST,datArray[dice]);
digitalWrite(STcp,HIGH); //pull the ST_CPST_CP to save the data
delay(100); //wait for a second
```

Display the corresponding number on the 7 segment display via `shiftOut()` function.
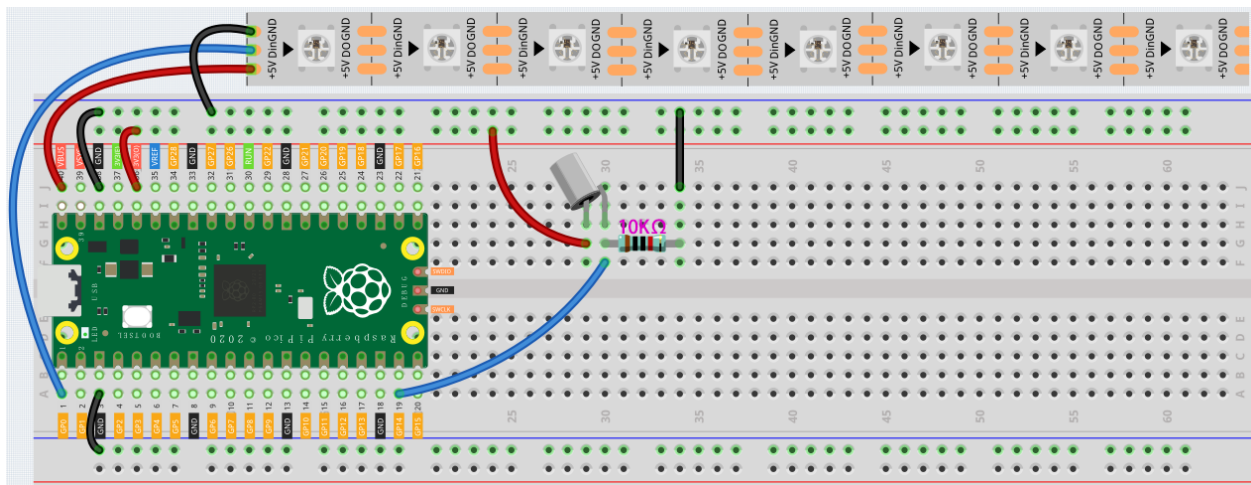
4.2.11 Flow LEDs

In this kit is equipped with a WS2812 RGB 8 LEDs Strip, which can display colorful colors and each LED can be controlled independently.

Here, a tilt switch is used to control the flow direction of LED on the WS2812 Strip.

Schematic
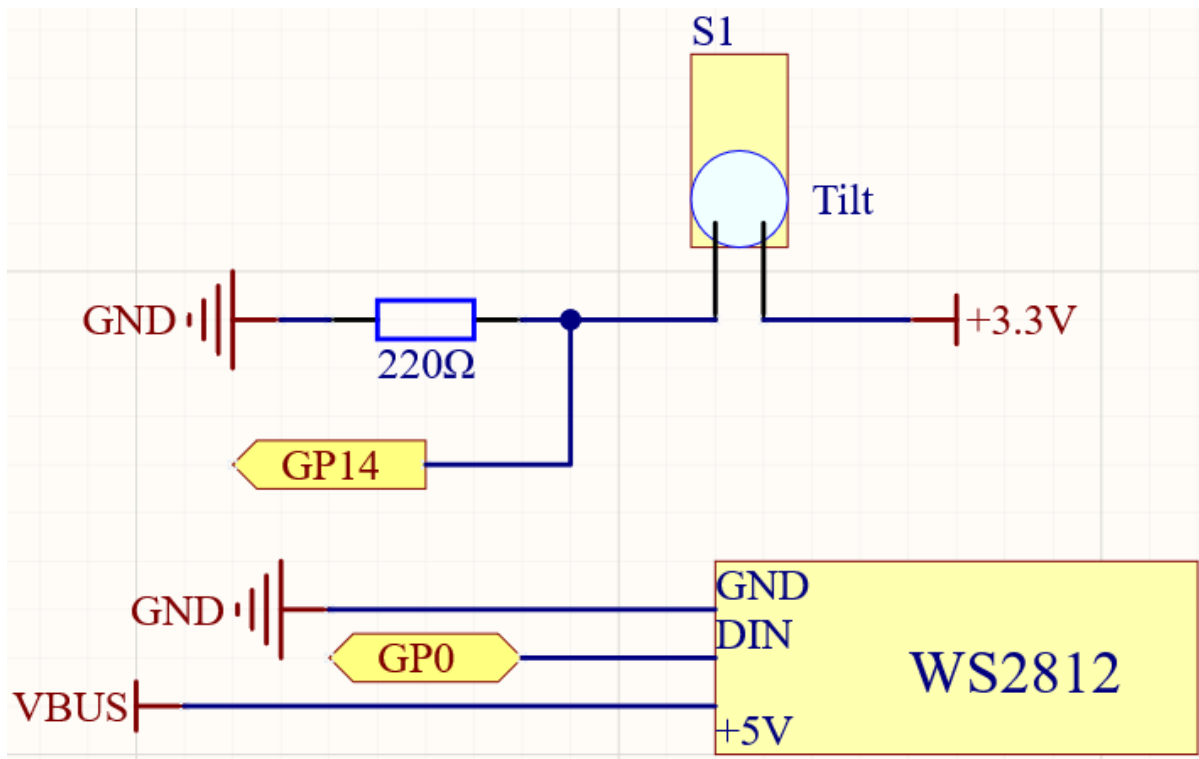
Wiring

Code

After the code runs, when the tilt switch is placed vertically, the LED on the WS2812 Strip will light up from one side, and when it is placed horizontally, the WS2812 will light up from the other side.

How it works?

```
#include <Adafruit_NeoPixel.h>

Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
```

Import the `Adafruit_NeoPixel.h` library and create a object `pixel` to control WS2812. The three parameters `NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800` are used to declare the number of LED pixels, pin number and Pixel type flags.

Note: To use this library, open the **Library Manager** in the Arduino IDE and install it from there.

Sketch -> Include Library -> Manager Libraries
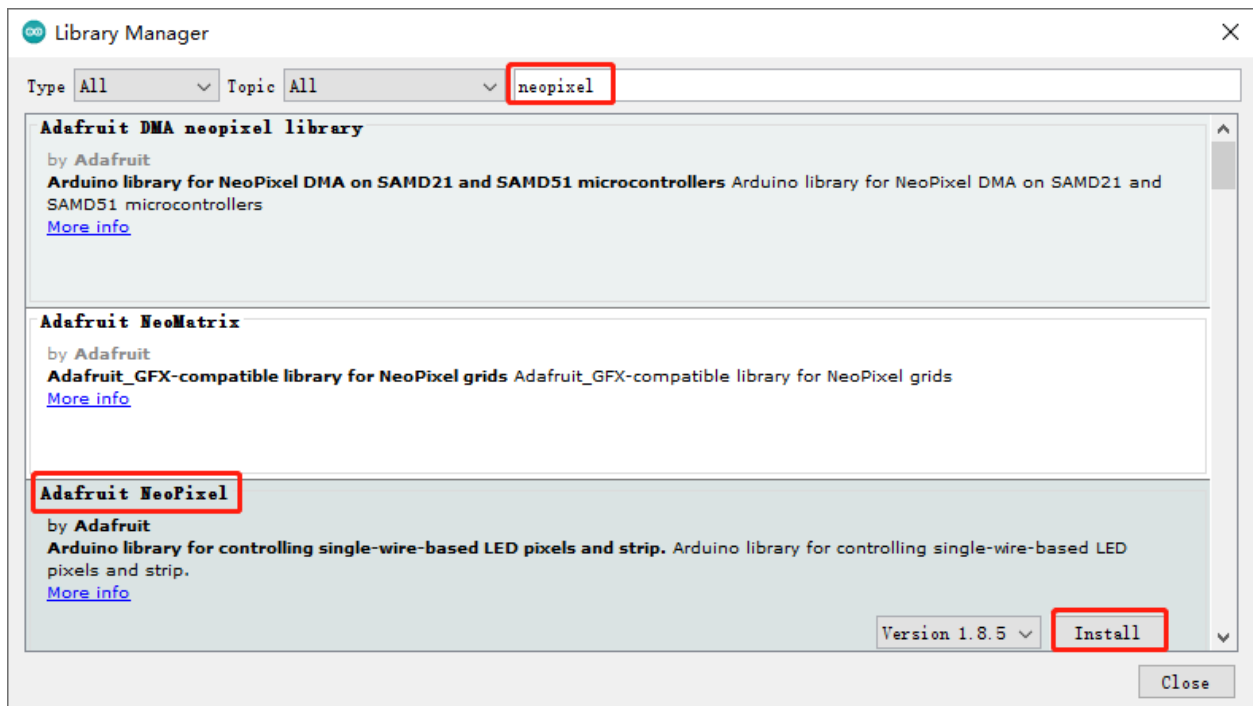
```
pixels.begin()
```

This statement is used to initialize the NeoPixel strip object.

```
if (ledIndex < 0) {
  ledIndex = NUMPIXELS-1;
}
else if (ledIndex >= NUMPIXELS) {
  ledIndex = 0;
}
```

Limit the value of `ledIndex` to 0 to `NUMPIXELS`. When it is greater than this range, `ledIndex` is re-assigned to 0. When it is less than this range, ledIndex is re-assigned to `NUMPIXELS-1`.

```
pixels.clear()
```

This statement set pixel colors to 0 (off).

```
pixels.setPixelColor(ledIndex, pixels.Color(100, 50, 0))
```

This statement is used to set the color of the WS2812 Strip, the first parameter refers to the serial number of the WS2812 Strip, and the second parameter represents the color.

```
pixels.show()
```

Show the effect on WS2812 Strip.

4.2.12 WS2812 Strip Multi-Mode

In addition to having the WS2812 Strip display one color or one LED at a time, you can control 8 LEDs at a time and have them each display a different color.

Schematic
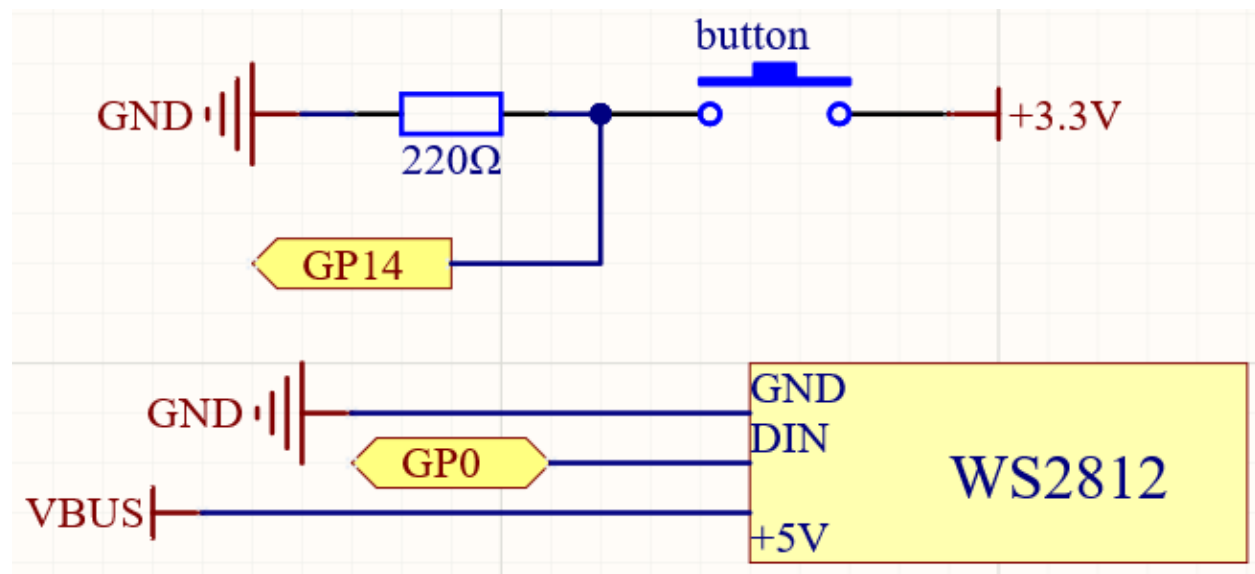
Wiring
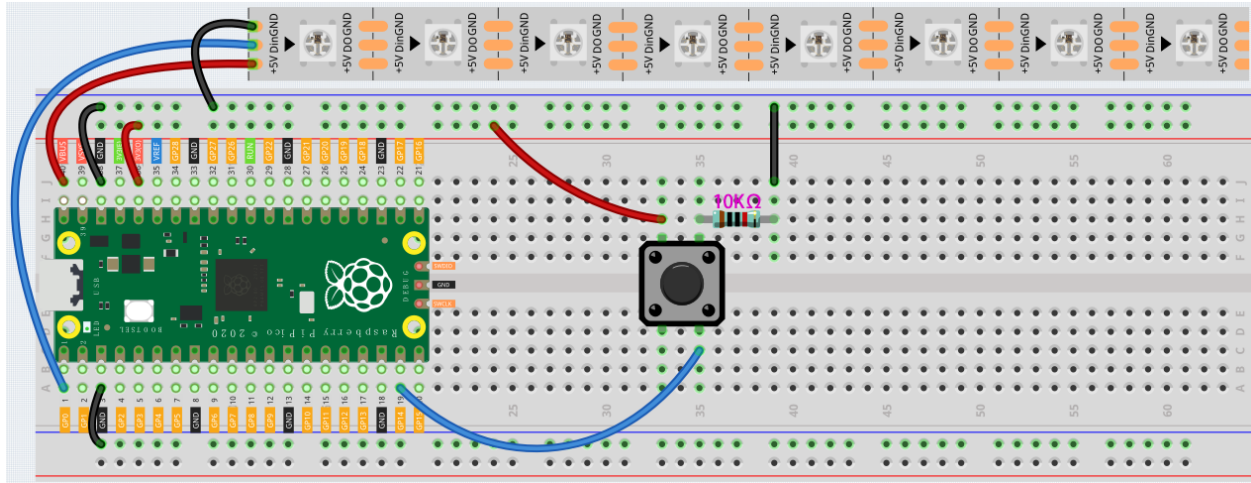
Code

After the program is running, every time you press the switch, WS2812 will change a mode, there are four modes in total.

How it works?

```
Adafruit_NeoPixel strip(PIXEL_COUNT, PIXEL_PIN, NEO_GRB + NEO_KHZ800)
```

Create object `strip` to control ws2812 and declare the number of LED pixels, pin number and Pixel type flags.

```
boolean lastButtonState = LOW
```

Set the initial state of the button to LOW, which means the button is not pressed.

```
boolean currentButtonState = digitalRead(BUTTON_PIN);

// Check if button press.
if ((currentButtonState == LOW) && (lastButtonState == HIGH)) {
  // Short delay to debounce button.
  delay(20);

  mode = (mode + 1) % 4;
  switch (mode) {          // Start the new animation...
    case 0:
      colorWipe(strip.Color(  255,   127,   0), 50);
      break;
    case 1:
      theaterChase(strip.Color(  0,   127, 127), 100);
      break;
    case 2:
      rainbow(10);
      break;
    case 3:
      theaterChaseRainbow(100);
```

```
        break;
}
```

Firstly, the value of the button is read, and when the button is detected from pressed to released, it enters into the mode selection. There are 4 modes:

- `colorWipe()`: From the left (wired side) to the right, light up one led every 50ms, after all light up, wait 500ms, then from left to right, turn off one led every 50ms(variable `wait`), until all off.

- `theaterChase()`: First light up the WS2812 Strip on the 1st (wired side), 4, 7 position of the LED, wait 50ms after extinguishing; then light up 2, 5, 8 of the led, wait 50ms after extinguishing; finally light up 3, 6 of the led, wait 50ms after extinguishing. The whole process is cycled 10 times, due to the short interval thus achieving the effect of flow.

- `rainbow()`: Different colors are displayed on 8 LEDs, and then these 8 colors flow from right to left (wired side), flowing 3 times and then stopping.

- `theaterChaseRainbow()`: Add flow effect to `rainbow()`.

4.2.13 LUMI Piano

In this project, we will use 7 buttons, a WS2812 Strip and a buzzer to create a LUMI Piano.

The 7 buttons respectively represent 7 notes. When the button is pressed, the passive buzzer will emit the corresponding note, and the WS2812 Strip will light up the corresponding LED.

Schematic
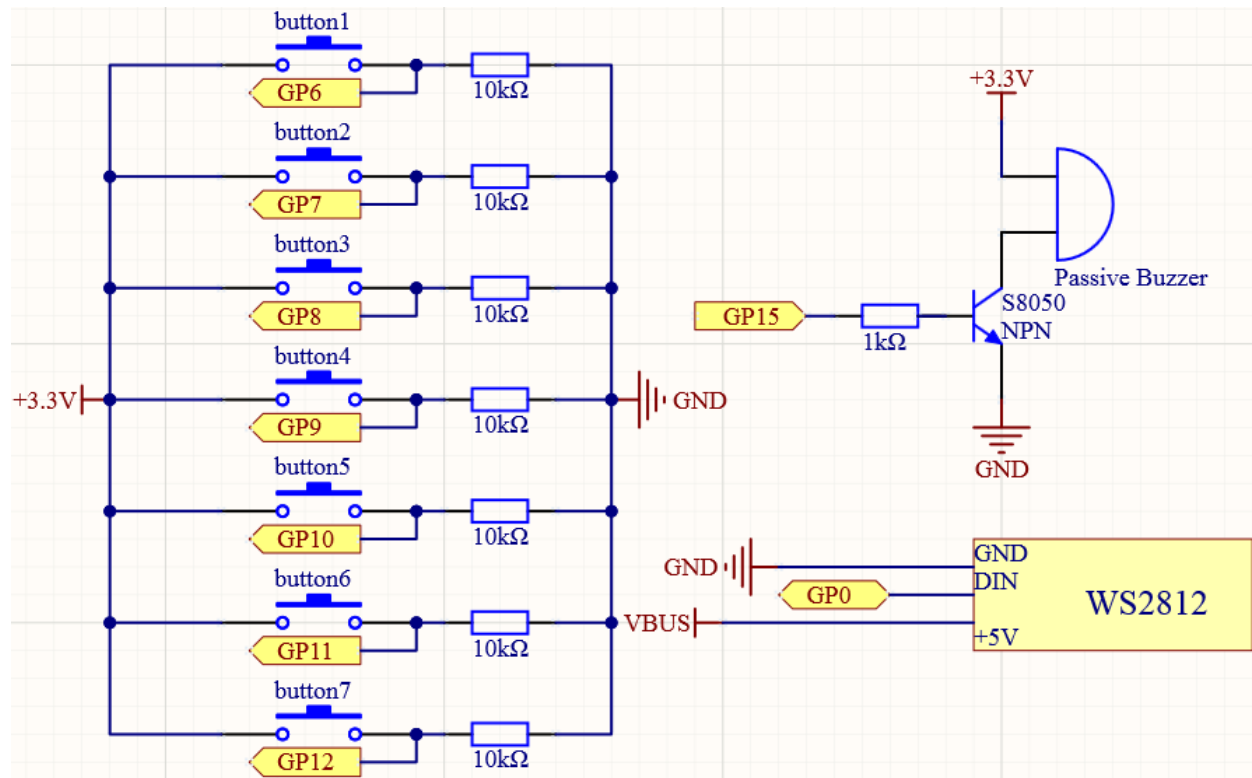
Wiring
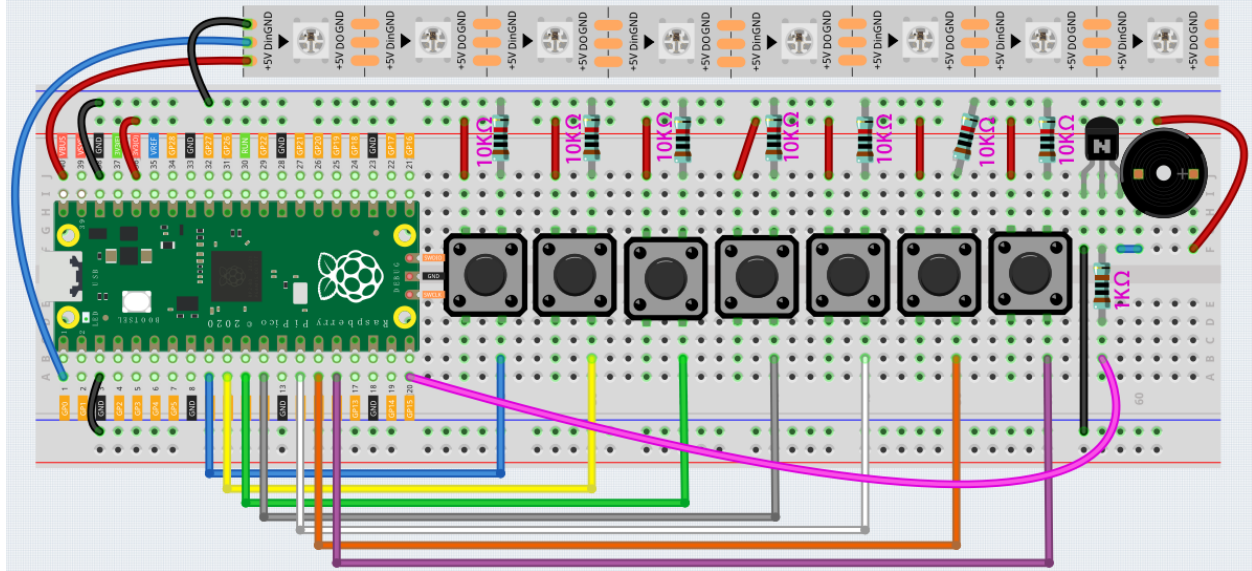
Code

After the program is running, press different buttons, the buzzer will emit different sounds, and the WS2812 will also light up the corresponding LEDs.

How it works?

```
const int buttonPin[] = {6, 7, 8, 9, 10, 11, 12};

const int pitch[] = {523, 587, 659, 698, 784, 880, 988};
```

Store the 8 button pins in the array `buttonPin[]` so that they can be quickly set and used later.

`pitch[]` stores the frequencies corresponding to C, E, E, F, G, A, B,notes in C major scale.

```
for (int i = 0; i <  NUMPIXELS ; i++) {
  pinMode(buttonPin[i], OUTPUT);
}
pixels.begin();
pinMode(buzzerPin, OUTPUT);
Serial.begin(9600);
```

Use the `for` loop in `setup()` to initialize the pins of all buttons and buzzer and set them to output mode.

```
for (int i = 0; i < NUMPIXELS; i++) {
  if (digitalRead(buttonPin[i]) == HIGH) {
    tone(buzzerPin, pitch[i]);
    pixels.setBrightness(100);
    pixels.setPixelColor(i, 0xFF8822);
    pixels.show();
    while (digitalRead(buttonPin[i]) == HIGH);
    fade();
```

(continues on next page)

```
    noTone(buzzerPin);
  }
}
```

Create a `for` loop in `loop()` function to traverse all the buttons, sounds and pixels, and set the judgment when the button is pressed, the buzzer will sound the corresponding note, and the WS2812 will light up the corresponding LED.

```
void fade() {
  while (1) {
    int brightness = pixels.getBrightness();
    if (brightness <= 0) {
      return;
    }
    pixels.setBrightness(brightness -= 1);
    pixels.show();
    delay(2);
  }
```

The `fade()` function is used to slowly reduce the brightness of the LEDs on the WS2812 Strip to 0.

First get the brightness value of the currently lit LED through `getBrightness()` function, then write the decreasing brightness value to WS2812 Strip through `setBrightness()` function, and finally present the effect through `show()` function.

If the brightness is less than 0, then exit this function.

The new projects for Arduino are still being updated continuously.

FOR PIPER MAKE

This chapter contains an introduction to Piper Make, how to connect and program Pico in Piper Make, and several interesting projects to help you get up and running with Piper quickly.

We recommend that you read this chapter in order.

5.1 Quick Guide on Piper Make

Piper Make is a super easy and fun way to make projects using Raspberry Pi Pico. It uses blocks like Scratch, so you don't need any programming experience to use it. The underlying principle is to use CircuitPython with auxiliary libraries.

5.1.1 Set up the Pico

First, visit Piper Make through the following link

https://make.playpiper.com/

In the pop-up page, if you don't need to subscribe for more tutorials, you can just click **Let's Go!** or the **x** button.
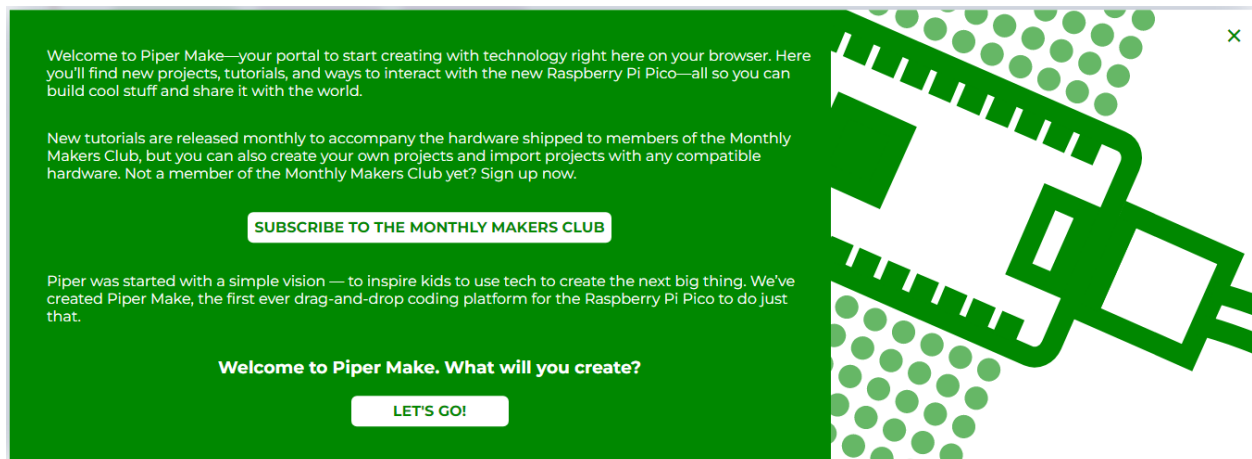
Note: If you see a different pop-up window, your browser version is not supported, please update your browser and try again.

Scroll to the bottom of this page and click on the **Set up my Pico** under the **Tools** section and follow the prompts to configure it.

Click **Next** to start configuring your Pico, even if you have set it up before, these are the same steps you will use to update your Pico firmware.
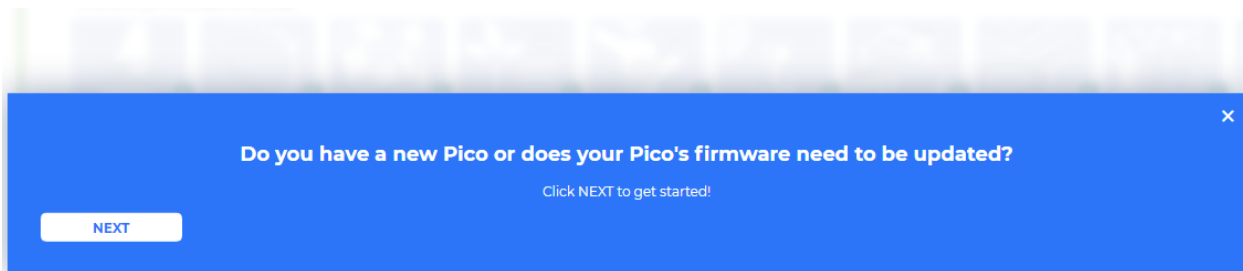
In this step, you need to make sure that your Pico is unplugged from your computer, as it needs to be plugged in in a specific way in the next step. Make sure your cable can handle power and data, as many micro USB cables only have power.
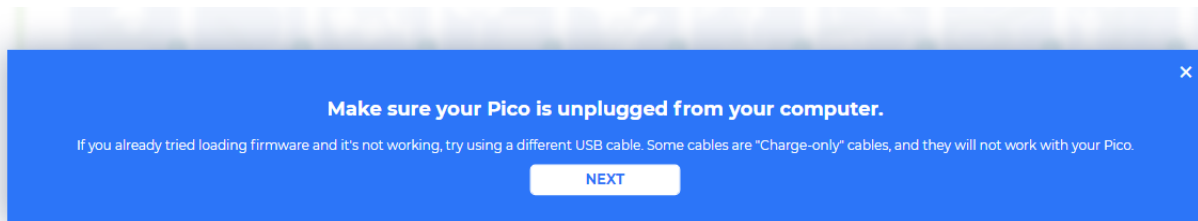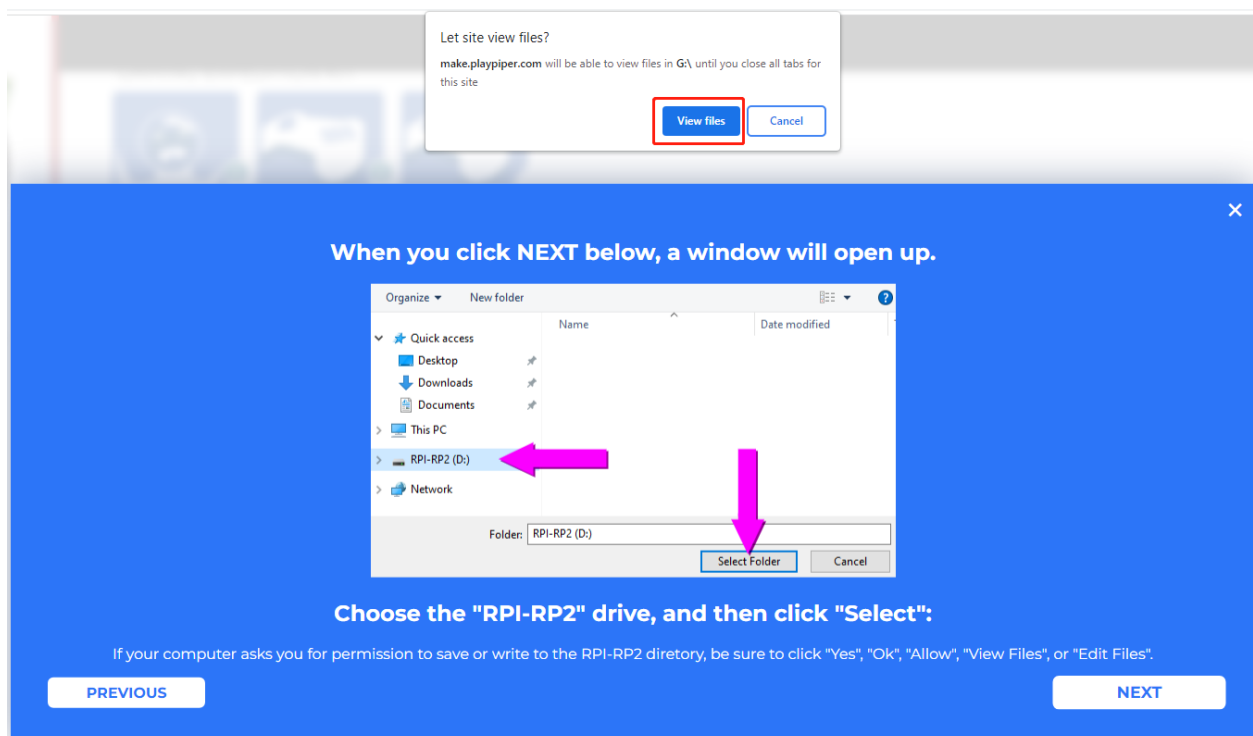

Now, press and hold the RST (white) button on the Pico and plug the Pico into your computer. Once plugged in, you can release the button.
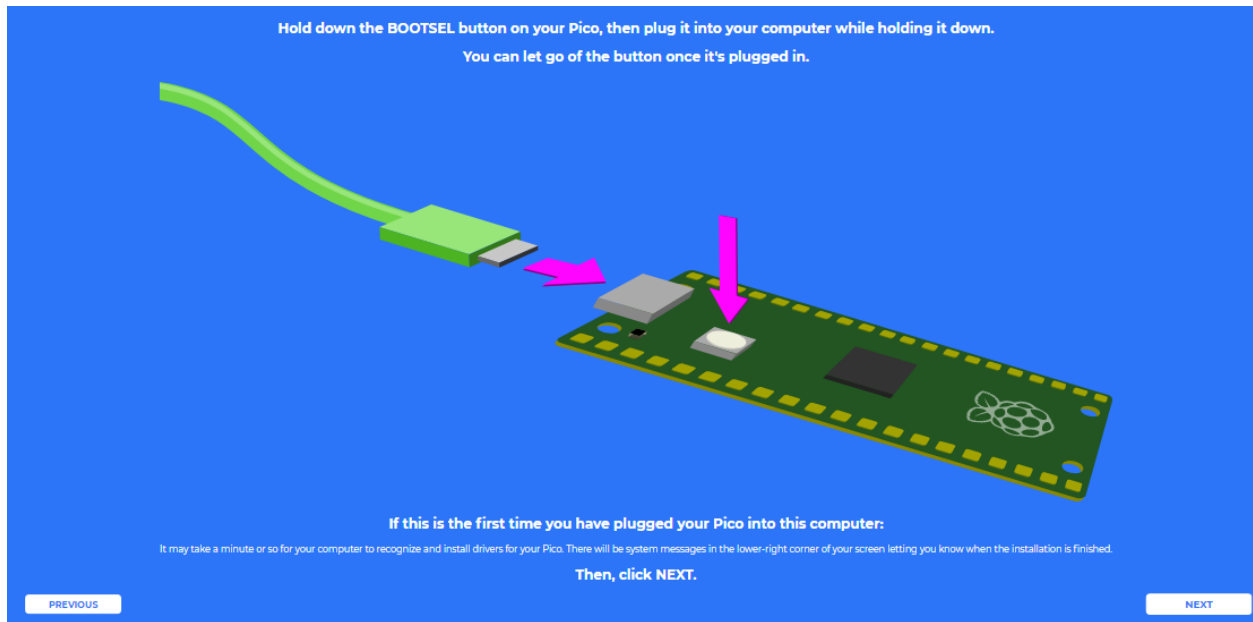
Your Pico will appear as a USB drive, click **Next** after that select **RPI-RP2** drive.

Note: After select **RPI-RP2** drive, there will be a pop up window at the top that you need to allow the web page to view files.

Now Piper Make will load the firmware to your Pico, again you need to allow save changes to the hard drive where the Pico is located.
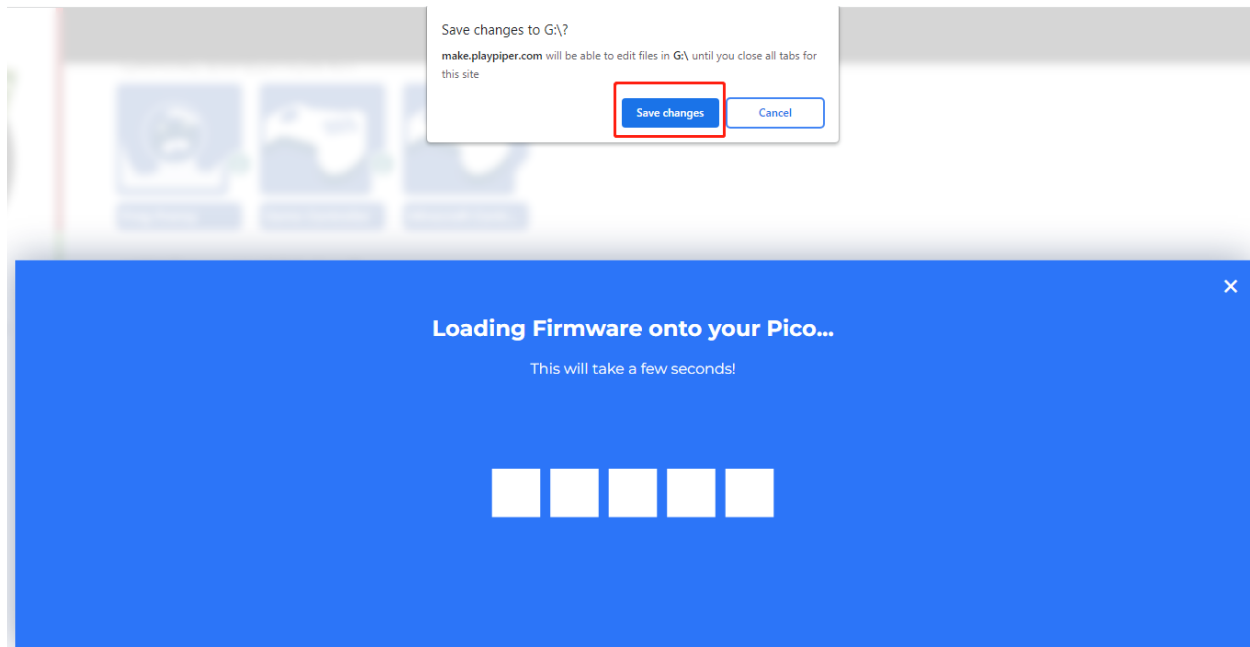

When this prompt appears, it means your Pico is set up and you can start using it.
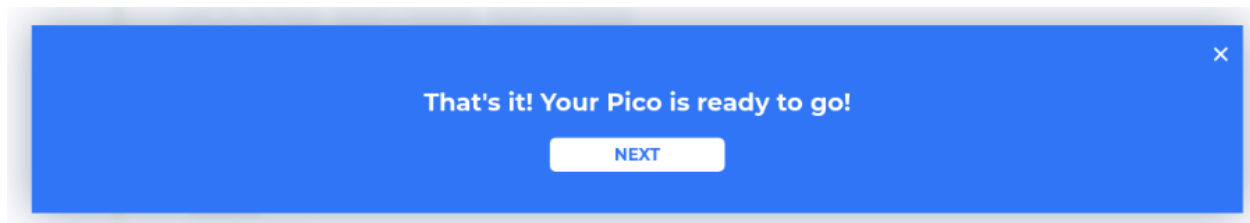
5.1.2 How to use Piper Make

Now that you have set up Pico, it is time to learn how to program it. Now let's light up the onboard LED.
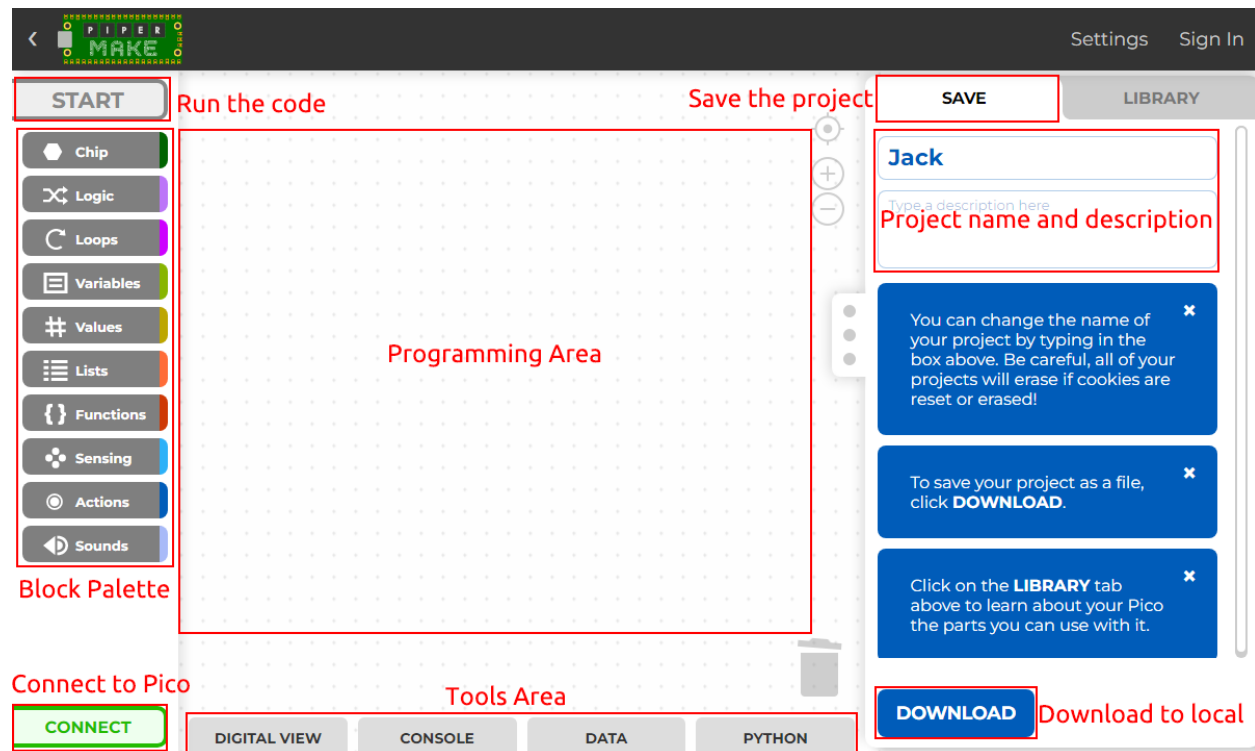
Switch to `CREATIVE MODE` and click on the **New Project** button, and a new project will appear in the **MY PROJECTS** section and will be assigned a random name that can be changed from the programming page.
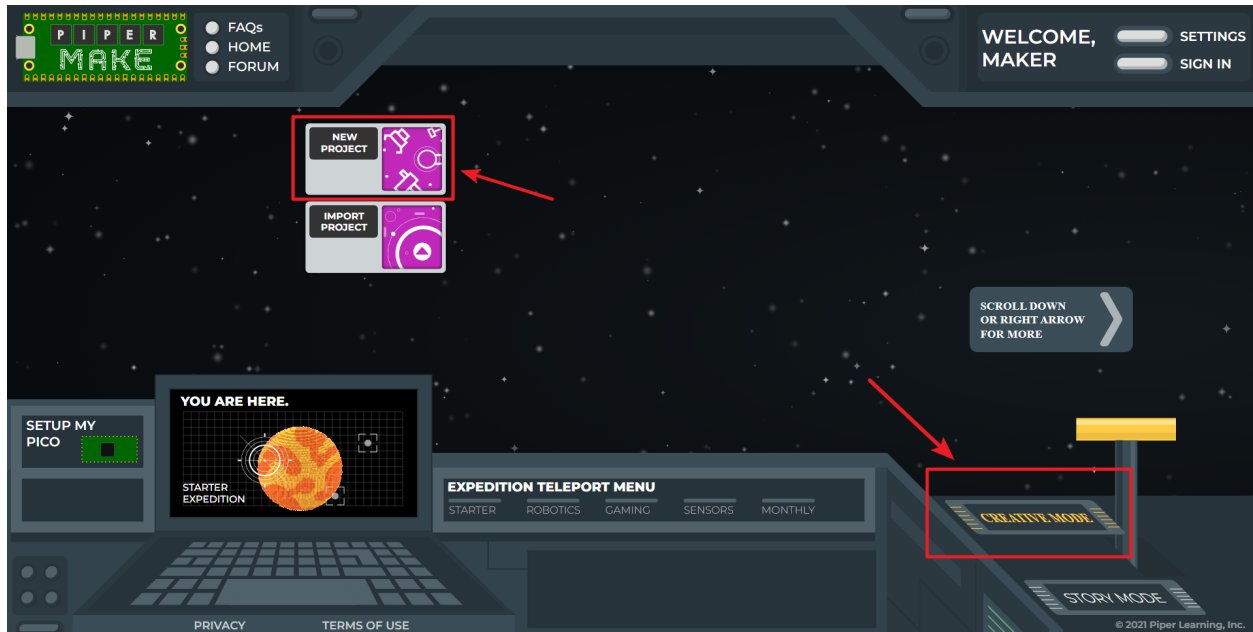
Then open the new project just created.
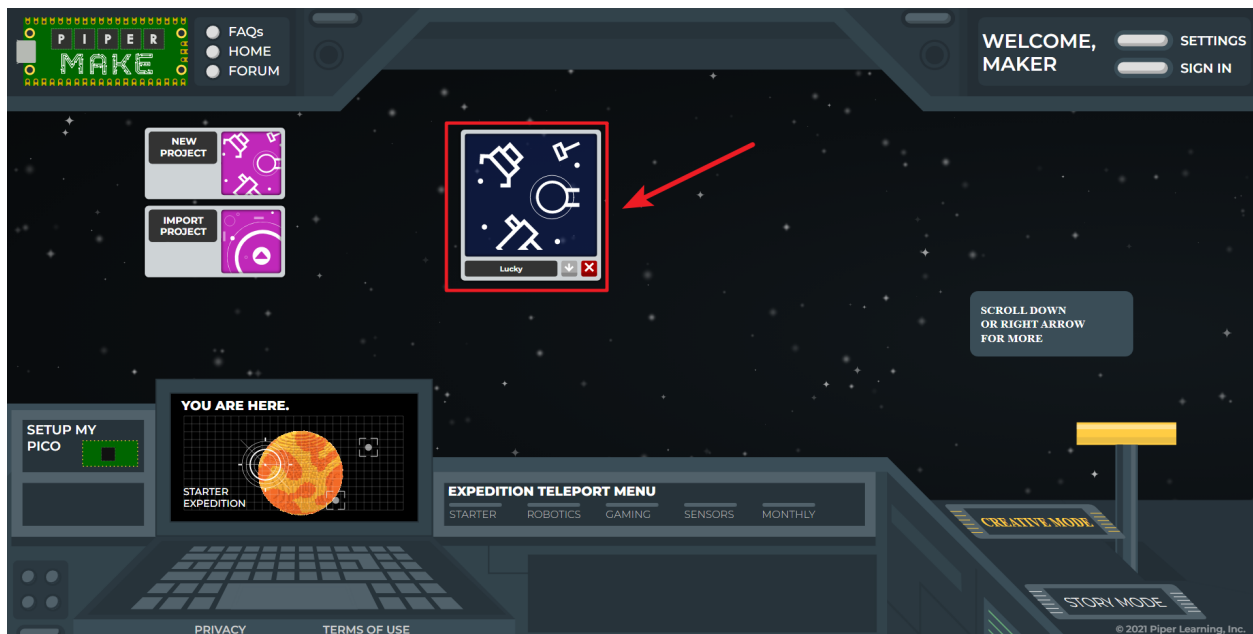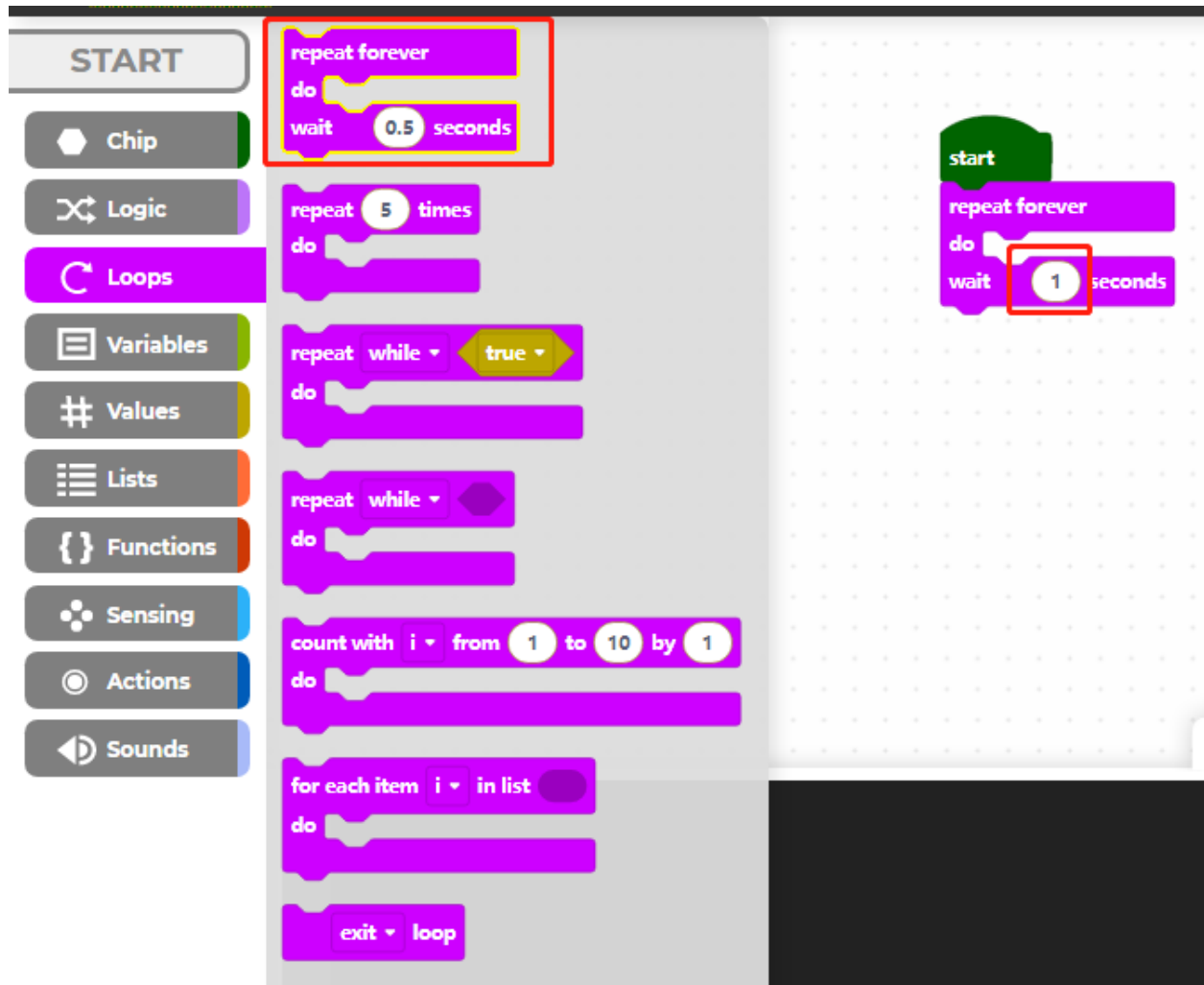

Now go to the Piper Make programming page.

- **START**: Used to run the code, if it's gray, it's not connected to Pico at this time.

- **Block palette**: contains different types of blocks.

- **CONNECT**: Used to connect to Pico, it is green when not connected to Pico, when connected it will become **DISCONNECT(red)**.

- **Programming Area**: Drag blocks here to finish programming by stacking them.

- **Tools Area**: You can click **DIGITAL VIEW** to see the pin distribution of Pico; you can view the print information in **CONSOLE**; you can read data from **DATA**, and you can click **Python** to view the Python source code.

- **Project name and description**: You can change the project name and description.

- **DOWNLOAD**: You can click the **DOWNLOAD** button to save it locally, usually in **.png** format. Next time you can import it via the **Import Project** button on the home page.

Click on the **Chip** palette and drag the [start] block to the **Programming Area**.
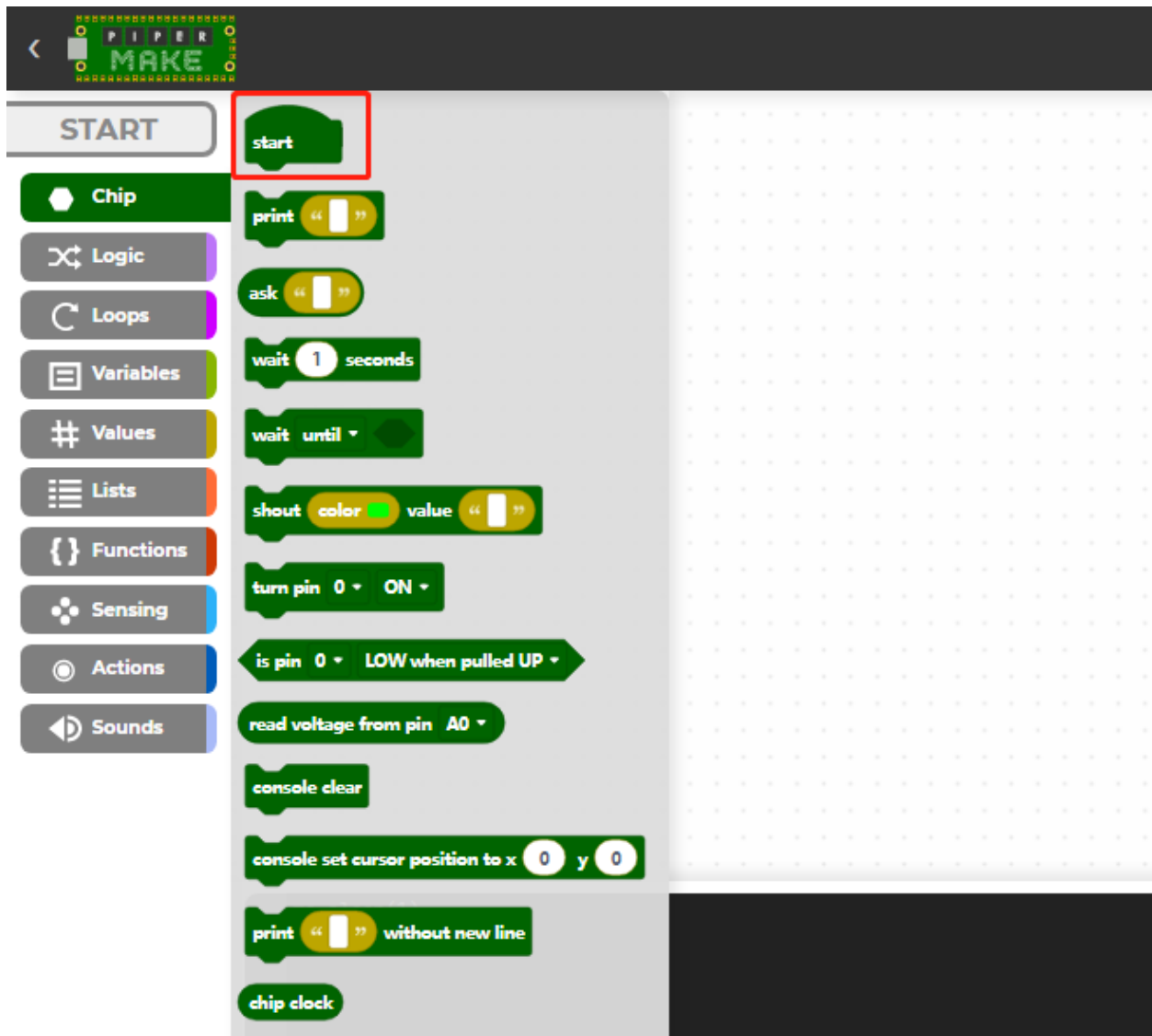
Then drag the [loop] block in **loops** palette to the bottom of the [start] block, and set the loop interval to 1 second.

START

Chip

Logic

Loops

Variables

Values

Lists

Functions

Sensing

Actions

Sounds

repeat forever
do
wait 0.5 seconds

repeat 5 times
do

repeat while ▾ true ▾
do

repeat while ▾
do

count with i ▾ from 1 to 10 by 1
do

for each item i ▾ in list
do

exit ▾ loop

start
repeat forever
do
wait 1 seconds

The Raspberry Pi Pico's onboard LED is at pin25, so we use the [turn pin () ON/OFF] block on the **Chip** palette to control it.
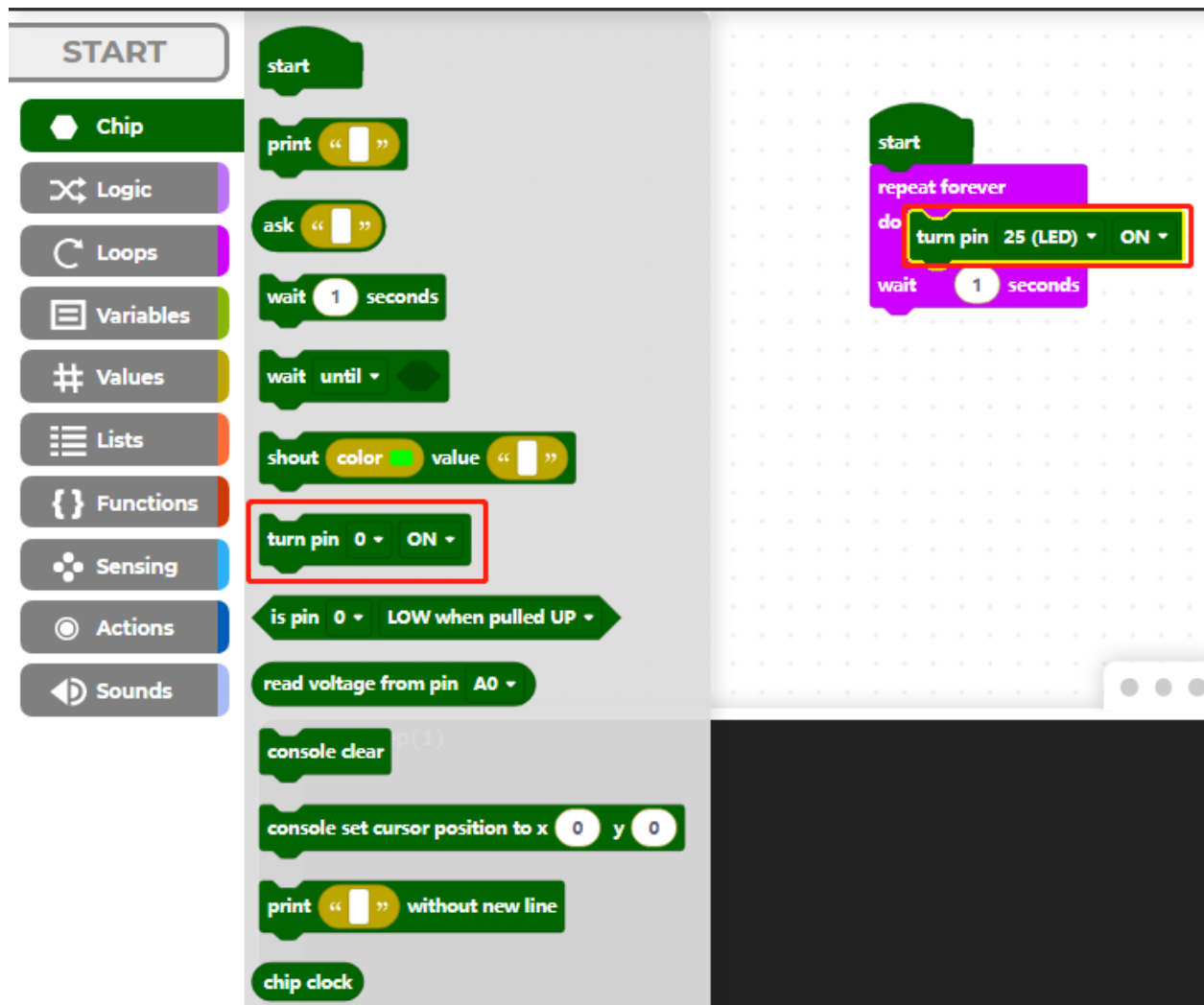
Now click on the **CONNECT** button to connect to pico, after clicking on it a new popup will appear, select the recognized **CircuitPython CDC control (COMXX)** port, then click on **Connect**. When the connection is successful, the green **CONNECT** in the bottom left corner will change to a red **DISCONNECT**.
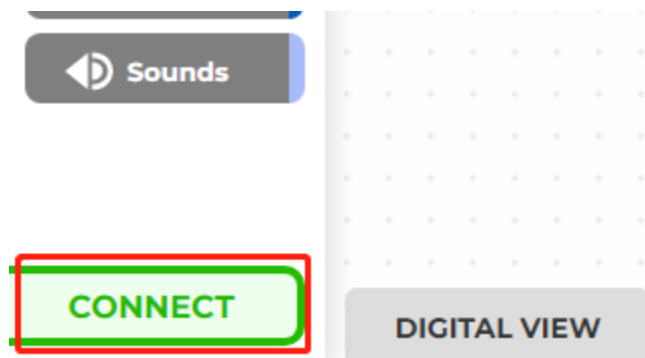
Now click on the **START** button to run this code and you will see the LED on the Pico lit up. If yours is gray, it means that the Pico is not connected, please reconnect it.
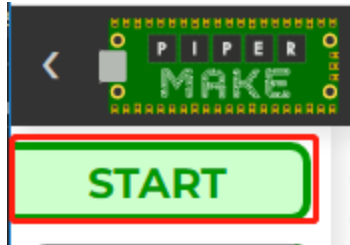
Then turn off pin25 every second in the cycle, and click **START** on the upper left again, so that you can see the onboard LED lights flashing.
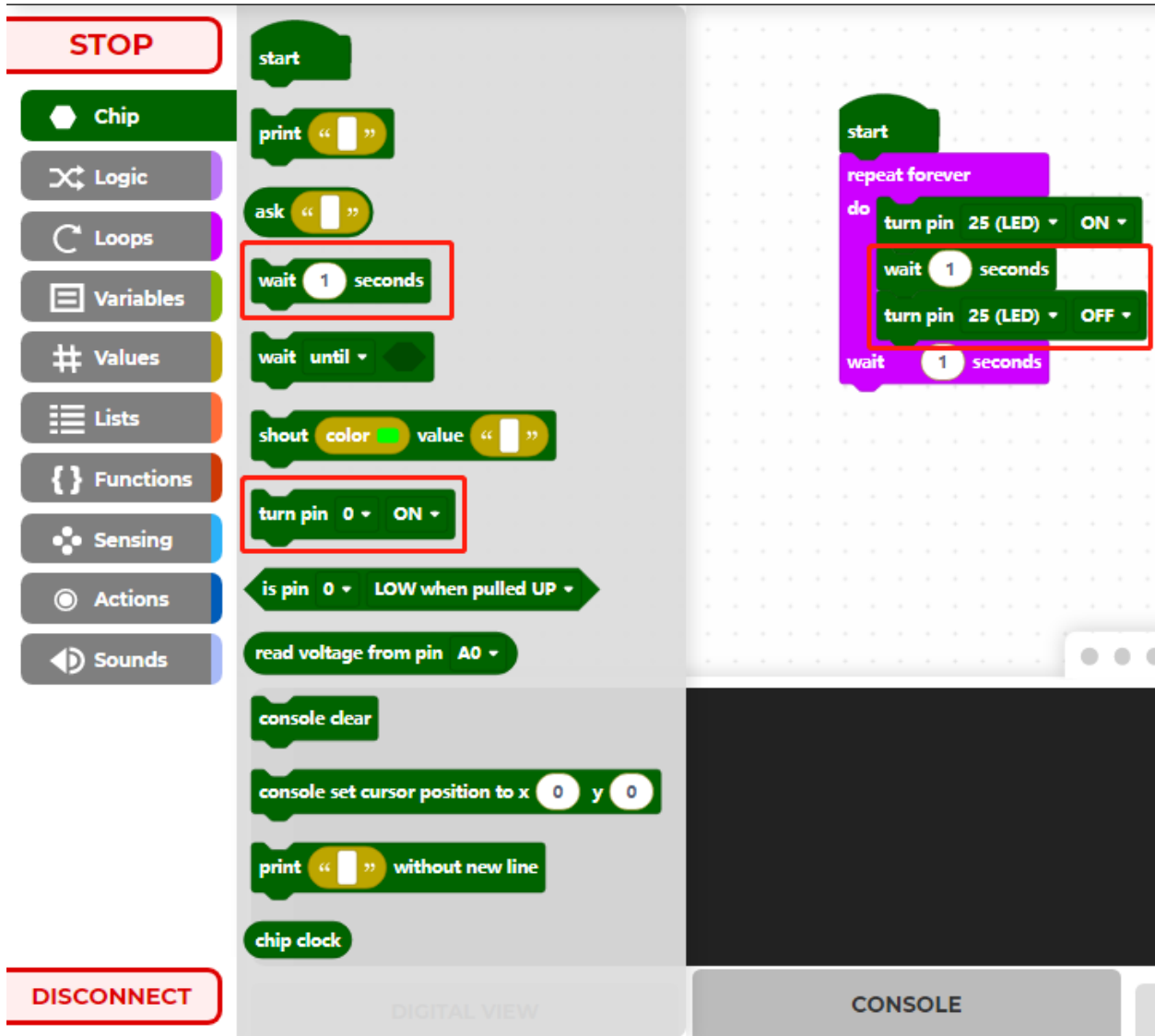
5.2 Blink LED

For this project, we intend to make the extended LED light blink. To use extended electronic components, a solderless breadboard will be the most powerful partner for novice users.

The breadboard is a rectangular plastic plate with a bunch of small holes in it. These holes allow us to easily insert electronic components and build electronic circuits. The breadboard does not permanently fix the electronic components, which makes it easy for us to repair the circuit and start over when we make a mistake.
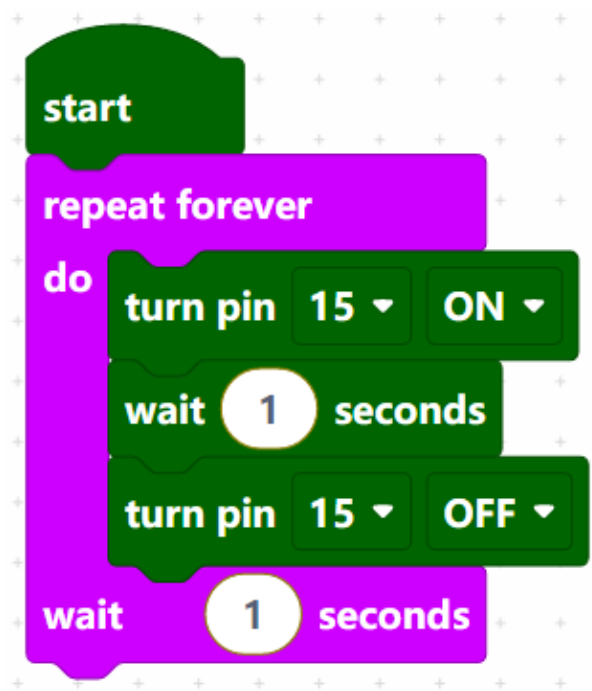
Wiring
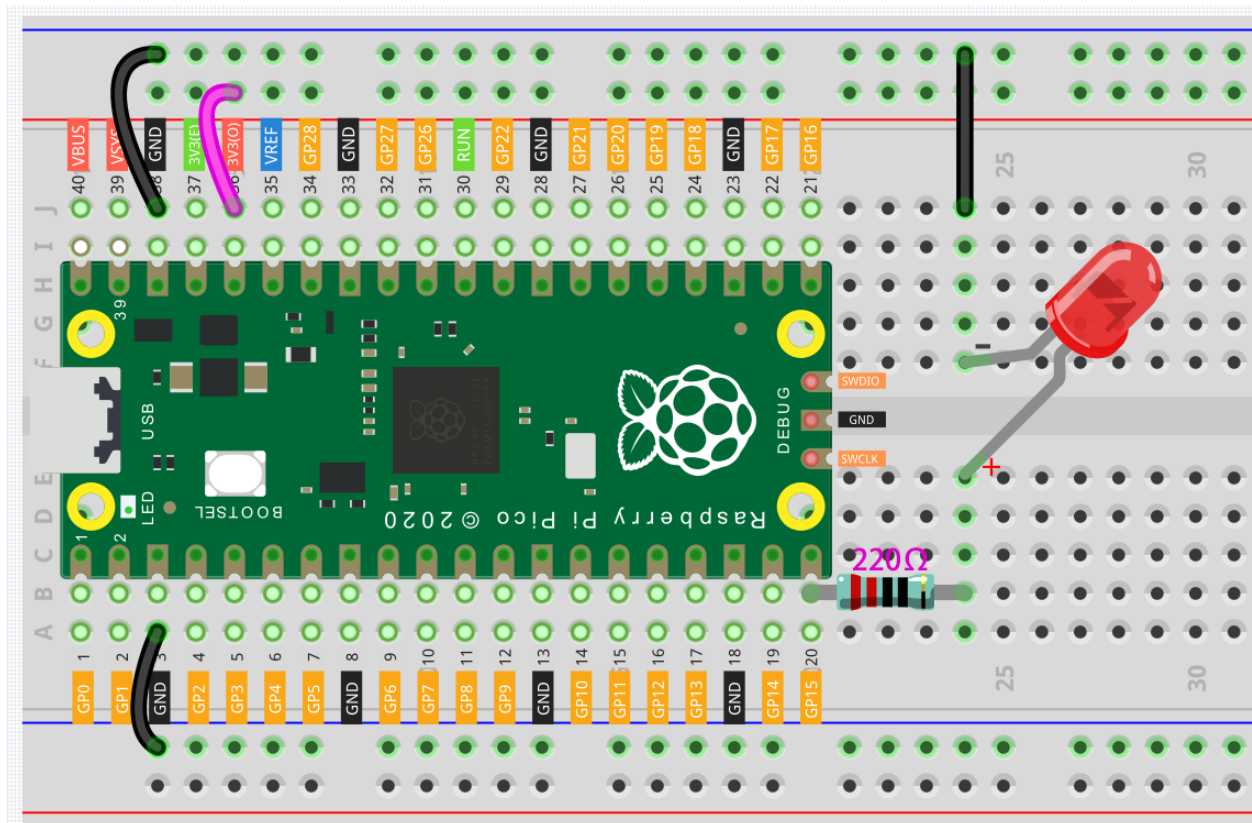
- The color ring of the 220 ohm resistor is red, red, black, black and brown.
- The longer lead of the LED is known as the anode (+), the shorter lead is the cathode (-).

Code

After connecting Pico, click the **Start** button and you will see the LED blinking. For more details, you can refer to *How to use Piper Make*.

Code Explanation

This is the body of the loop: turn the pin15 on to light up the LED, wait one second, then turn the pin15 off to make the LED go off. Wait 1 second and then re-run the previous cycle, so you can see the LED has been in the state of alternating between light and off.

- [start]: This block is the basic framework of the program and represents the beginning of the program.

- [repeat forever do() wait()seconds]: Means that the blocks in it will be executed repeatedly, and the execution time interval is defined by yourself.

- [turn pin () ON/OFF]: Indicates that a certain pin is placed in a high level (ON) or a low level (OFF).

- [wait () seconds]: Set the execution interval between blocks.

5.3 Button

In this project, we will learn how to turn on or off the LED by using a button.

Wiring
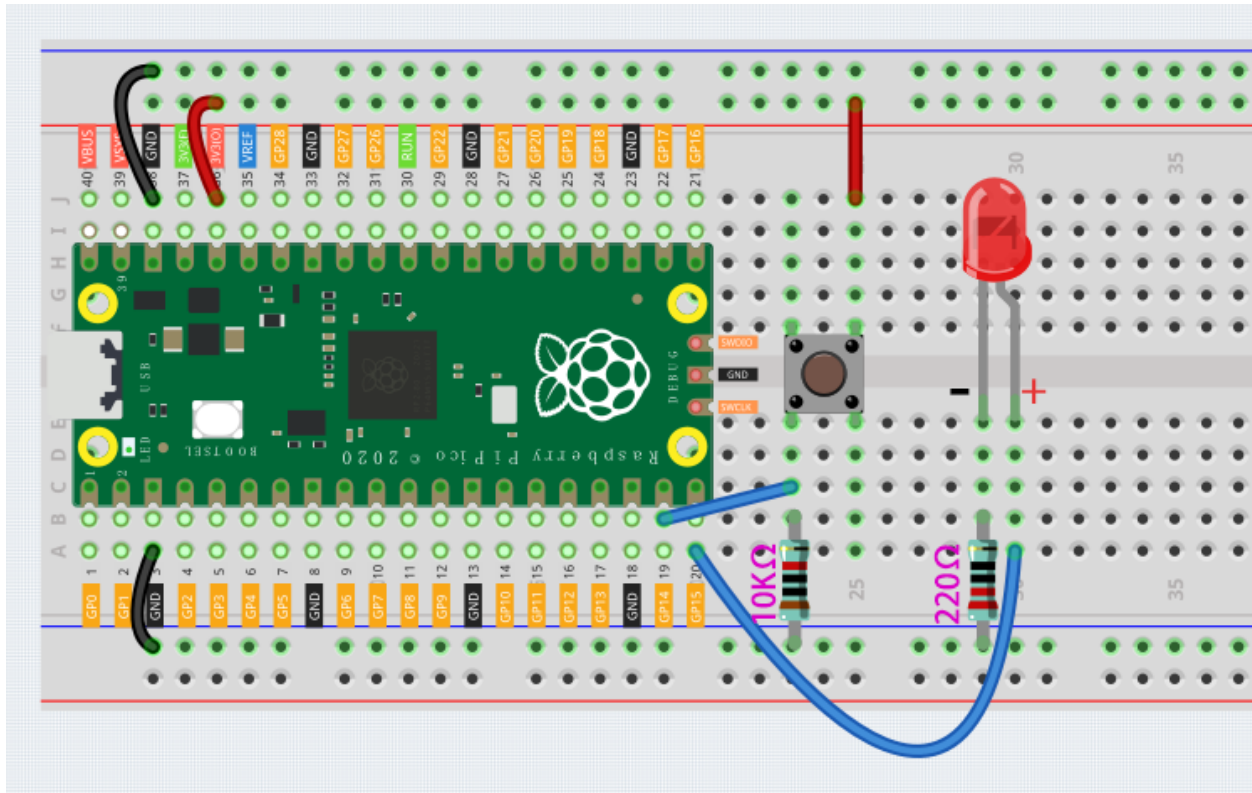
- We can think of the four-legged button as an H-shaped button. Its left (right) two feet are connected, which means that after it straddles the central dividing line, it will connect the two half rows of the same row number together. (For example, in my circuit, E23 and F23 have been connected, as are E25 and F25). Before the button is pressed, the left and right sides are independent of each other, and current cannot flow from one side to the other.

- Buttons require pull-up resistors or pull-down resistors. If there is no pull-up or pull-down resistor, the main controller may receive a 'noisy' signal which can trigger even when you're not pushing the button.

- The color ring of the 10k resistor is brown, black, black, red, brown.

Code

After connecting Pico, click the **Start** button and the code starts to run. When the button is pressed, the LED will be lit. When the button is released, the LED will go out.
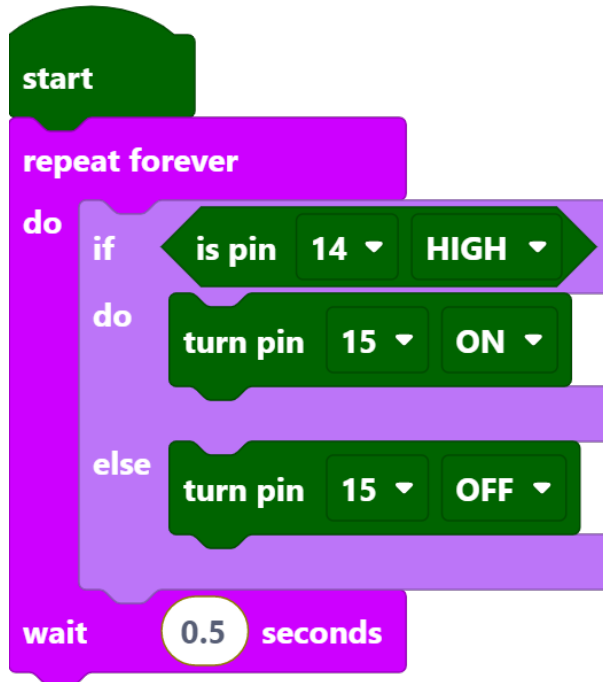
Code Explanation

When the button is pressed, pin14 is high. So if the read pin14 is high, turn the pin15 on (LED is lit); else, turn off the pin15 (LED is off).

- [if () do () else ()]: This is a judgment block, depending on the condition after the [if] block to determine whether to run the blocks inside the [do] block, or the blocks inside the [else] block.

- [is pin () HIGH]: This is used to read the level of a specific pin, if the level read is the same as the set HIGH/LOW, then execute the blocks inside [do] block, otherwise execute the blocks inside [else].

5.4 Slide Switch

In this project, we will learn how to use a slide switch. Usually,the slide switch is soldered on PCB as a power switch, but here we need to insert it into the breadboard, thus it may not be tightened. And we use it on the breadboard to show its function.
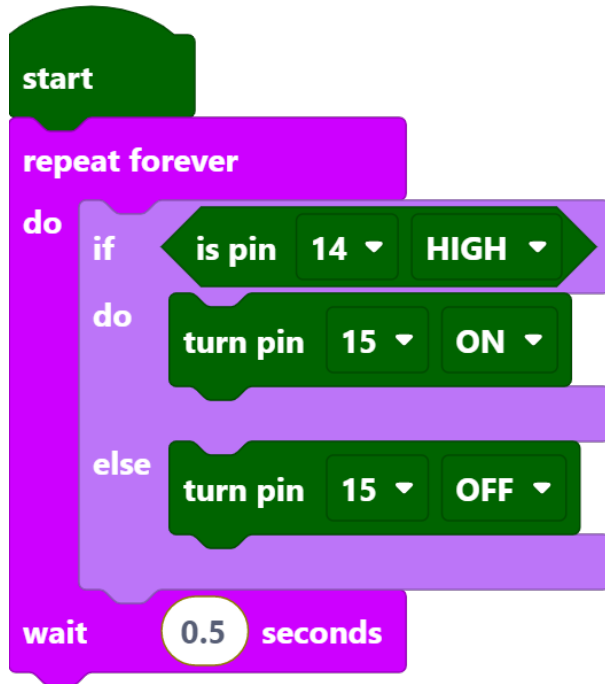
Wiring
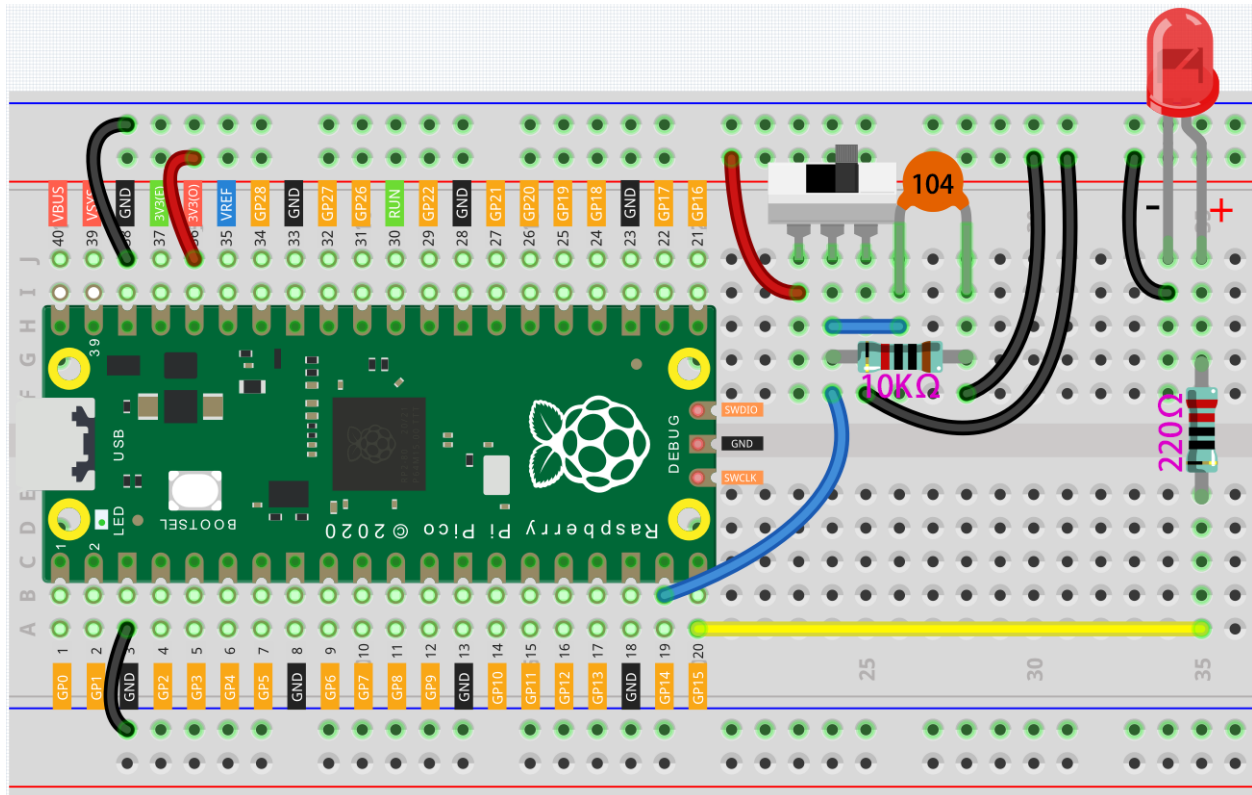
- If the switch is toggled to the left, the 2 pins on the left are connected together; if it is toggled to the right, the 2 pins on the right are connected together.

- The purpose of connecting 10K resistor and capacitor on the toggle switch is to de-jitter and get a stable level.

Code

After connecting Pico, click the **Start** button and the code starts to run. Toggle the slide switch to the left and the LED will light up; otherwise, the LED will turn off.
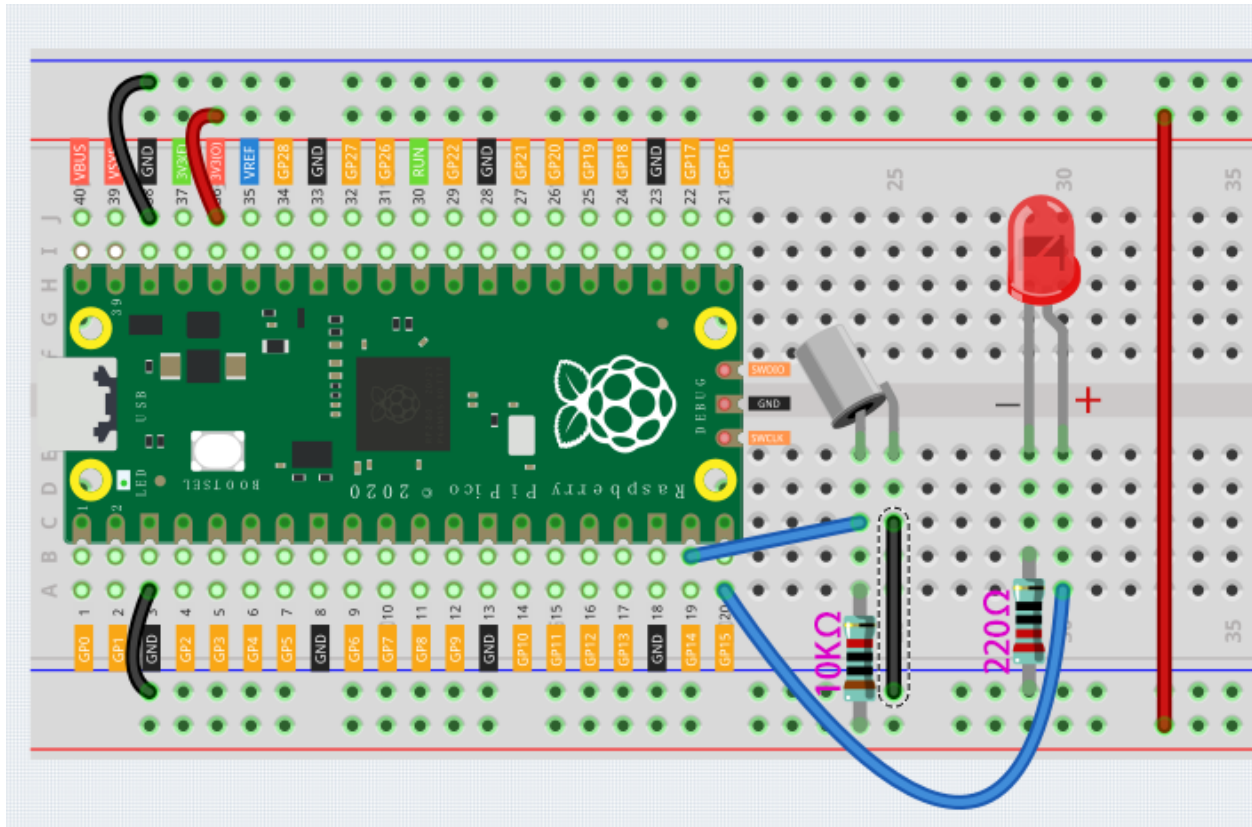

Note: This project code is exactly the same as the previous project *Button*.

5.5 Tilt Switch

This is a ball tilt-switch with a metal ball inside. It is used to detect inclinations of a small angle.

Wiring

- The tilt switch has an iron ball inside, when you put it upright, the iron ball falls down, which can make the 2 pins connected together; tilt it to a horizontal angle, the iron ball rolls towards the end, so the 2 pins of the tilt switch are disconnected.

- Again, you need to connect 10K pull-up resistor to get a stable level.

Code

After connecting Pico, click the **Start** button and the code starts to run. When the tilt switch is placed horizontally, the LED will turn on; and when placed vertically, the LED will turn off.
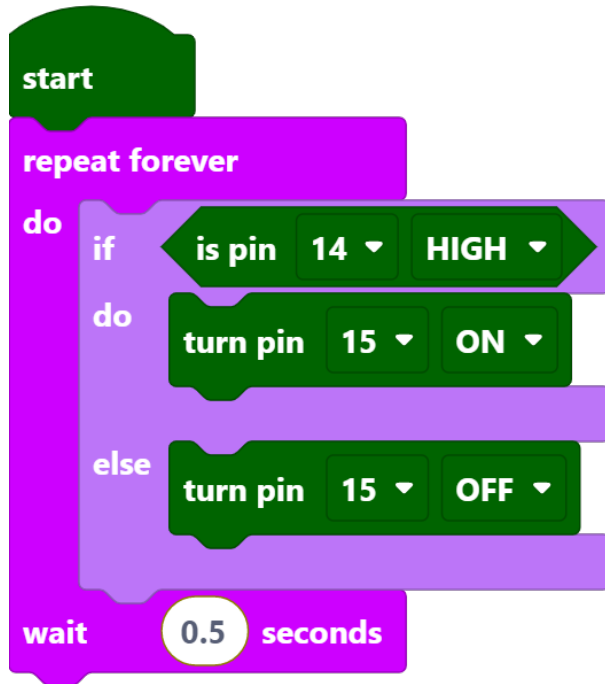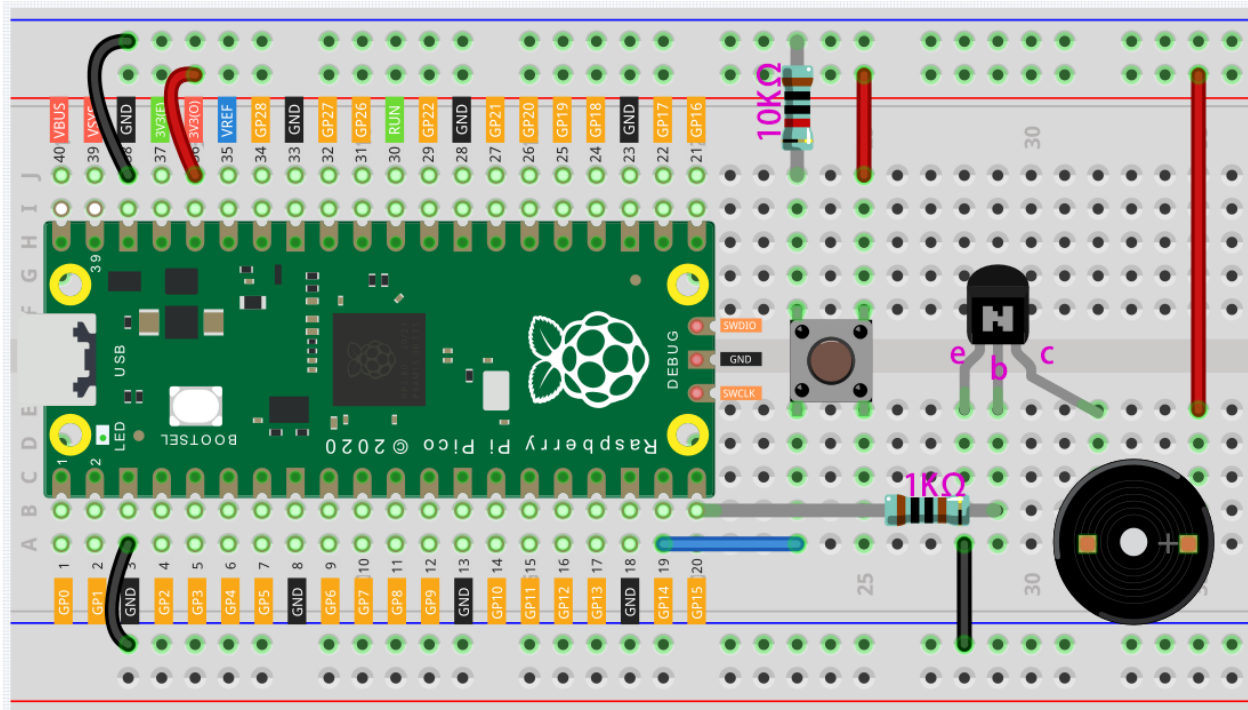
Note: This project code is exactly the same as the previous project *Button*.

5.6 Active Buzzer

In this project, we will learn how to use a button to control the active buzzer.

Wring

- An active buzzer is a device that makes a sound by directly giving high and low levels. Since the current coming out of the Pico pin is small, an NPN transistor is used here to amplify the current to make the buzzer sound louder.

- The 1K resistor between the NPN transistor and GP15 is used for current limiting when the transistor is energized.

Code

After connecting Pico, click the **Start** button and the code starts to run. The buzzer will sound when the button is pressed.
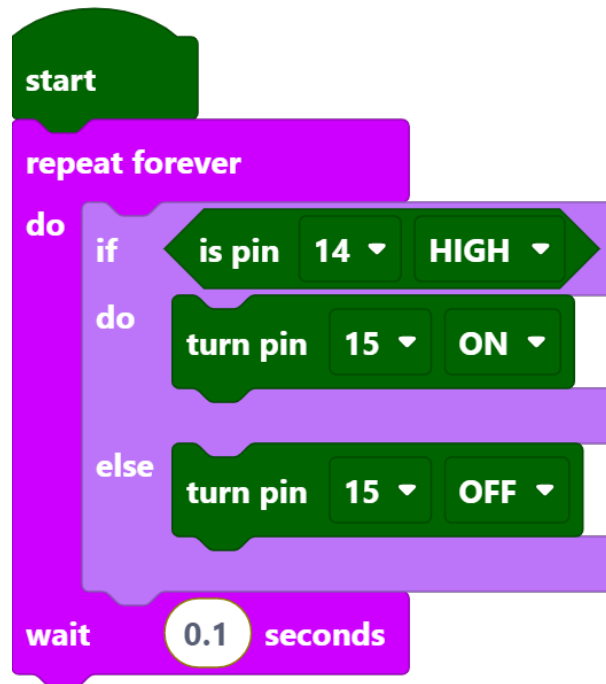
Note: This project code is exactly the same as the previous project *Button*.
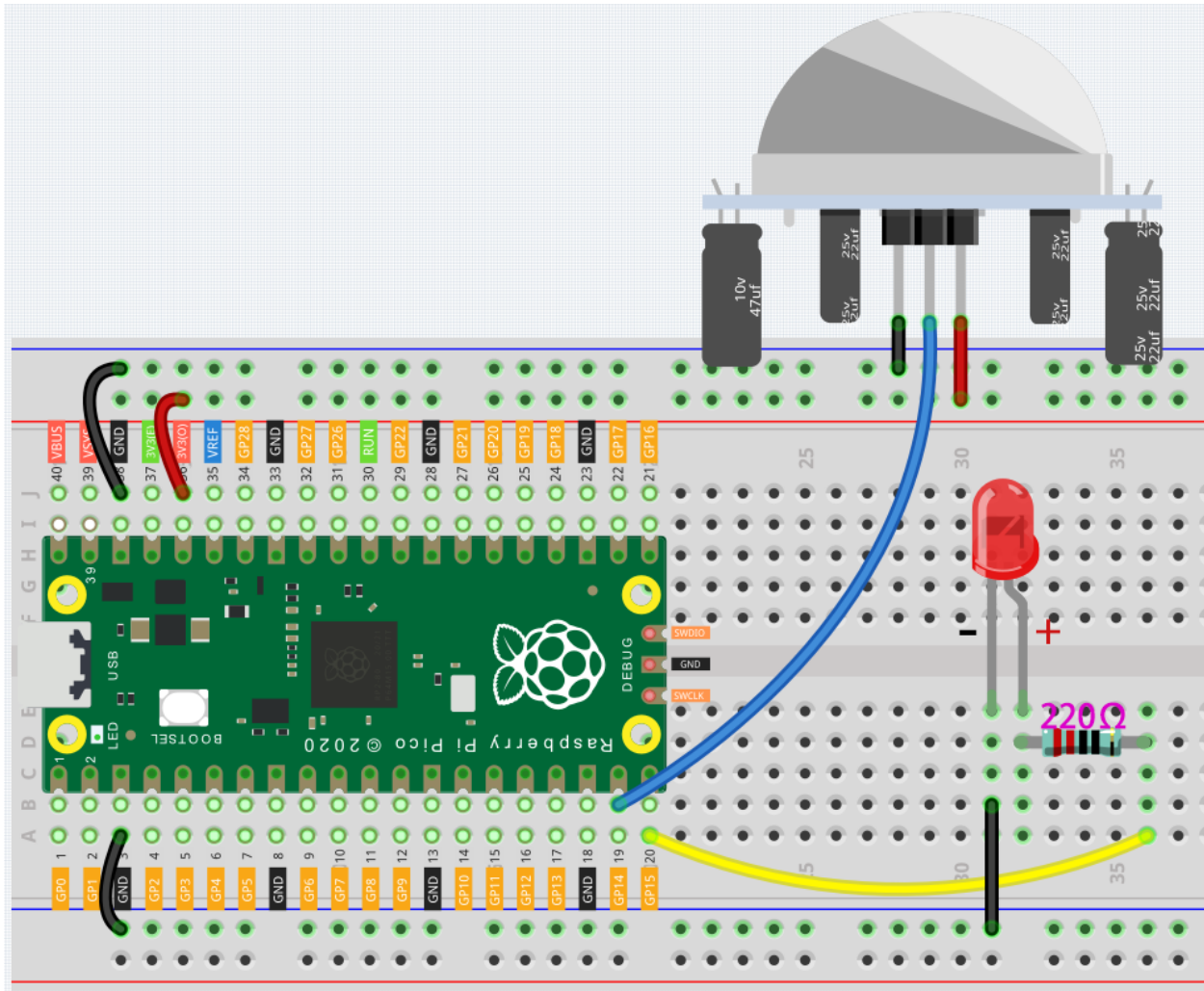
5.7 PIR Module

In the previous chapters, we used simple electronic components (such as LED, button). This time we will use the sensor module - PIR.

Passive infrared sensor (PIR sensor) is a common sensor that can measure infrared (IR) light emitted by objects in its field of view. Simply put, it will receive infrared radiation emitted from the body, thereby detecting the movement of people and other animals. More specifically, it tells the main control board that someone has entered your room.

PIR Motion Sensor

Now, let's use PIR module and LED to build an Intruder Alarm, when an object passes through the PIR, the LED will light up.
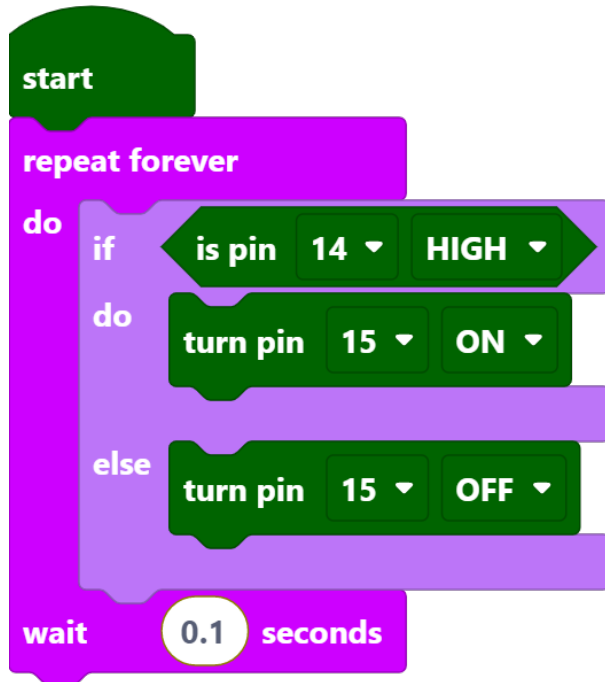
Wiring

- The PIR module will output a high level signal when it detecwts people or other animals passing by.

Code

After connecting Pico, click the **Start** button and the code starts to run. When an object passes through the PIR, the LED will light up.

Note: This project code is exactly the same as the previous project *Button*.

5.8 RGB LED

In this project, we will learn how to use Piper Make to light up the RGB LED.

RGB LED is equivalent to encapsulating Red LED, Green LED, Blue LED under one lamp cap, and the three LEDs share one cathode pin. Since the electric signal is provided for each anode pin, the light of the corresponding color can be displayed. By changing the electrical signal intensity of each anode, it can be made to produce various colors.

Wiring

- First find the longest pin (GND), there is only one pin (Red) on the left side of the longest pin, and two pins on the right side are Green and Blue.

- When using the same power supply intensity, the Red LED will be brighter than the other two, and a slightly larger resistor(330) needs to be used to reduce its brightness.

- The color ring of the 220 resistor is red, red, black, black and brown.

- The color ring of the 330 resistor is orange, orange, black, black and brown.

- The 3 buttons are used to control the lighting of the Red, Green and Blue LEDs respectively.

Code

After connecting Pico, click the **Start** button and the code starts to run. Pressing these buttons individually will emit

a single color of light, but if two of the buttons are pressed at the same time, or all 3 buttons are pressed at the same time, the RGB LEDs will emit a variety of different colors, up to a maximum of 7.

```
start
repeat forever
do  if    is pin  10 ▼  LOW ▼
    do    turn pin  13 ▼  ON ▼
    else  turn pin  13 ▼  OFF ▼

    if    is pin  11 ▼  LOW ▼
    do    turn pin  14 ▼  ON ▼
    else  turn pin  14 ▼  OFF ▼

    if    is pin  12 ▼  LOW ▼
    do    turn pin  15 ▼  ON ▼
    else  turn pin  15 ▼  OFF ▼

    wait  0.1  seconds
```

Note: In fact, RGB LED can emit up to 16 million colors, but since Piper Make does not have a block to output PWM signal, here we just use the [turn pin() (ON/OFF)] block to make RGB LEDs show 7 colors.

Code Explanation

You can think of this project as using three buttons to control the RGB LED, and setting three if judgment conditions to determine whether the three buttons are pressed or not. When the buttons are pressed, the levels of the corresponding pins are pulled high, causing the RGB LED to display different colors.

5.9 Servo

Servo is a position (angle) servo device, which is suitable for those control systems that require constant angle changes and can be maintained. It has been widely used in high-end remote control toys, such as airplanes, submarine models, and remote control robots.

Now, try to make the servo sway!

Wiring
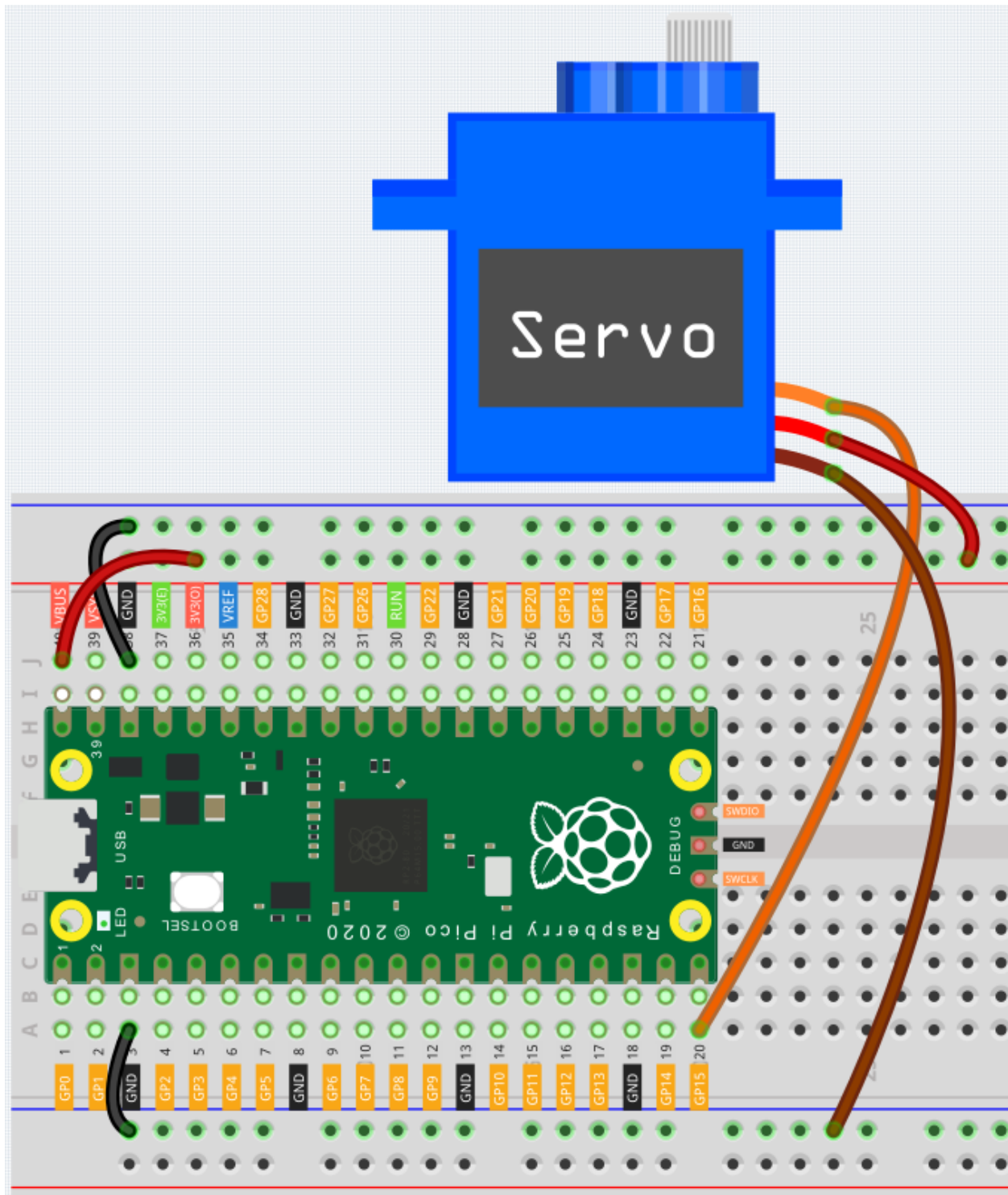
- The brown wire of the servo is GND, the red wire is VCC and the yellow wire is signal.

- Connect **VBUS** (not 3V3) to the power bus of the breadboard.

Code

After connecting Pico, click the **Start** button and the code starts to run. If necessary, press the Servo Arm into the Servo output shaft, then you can see the Servo Arm swinging back and forth from 0° to 180°.
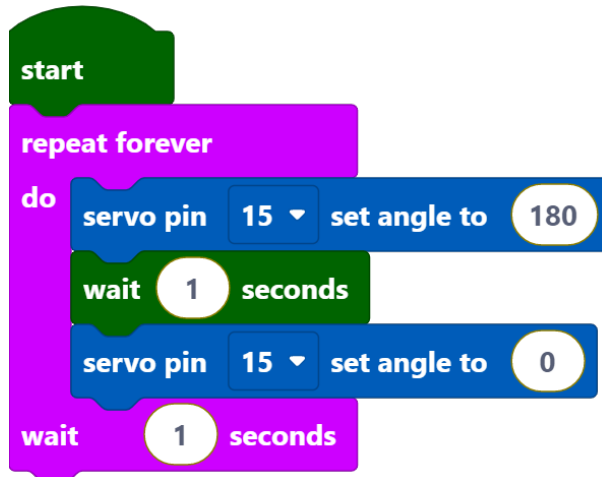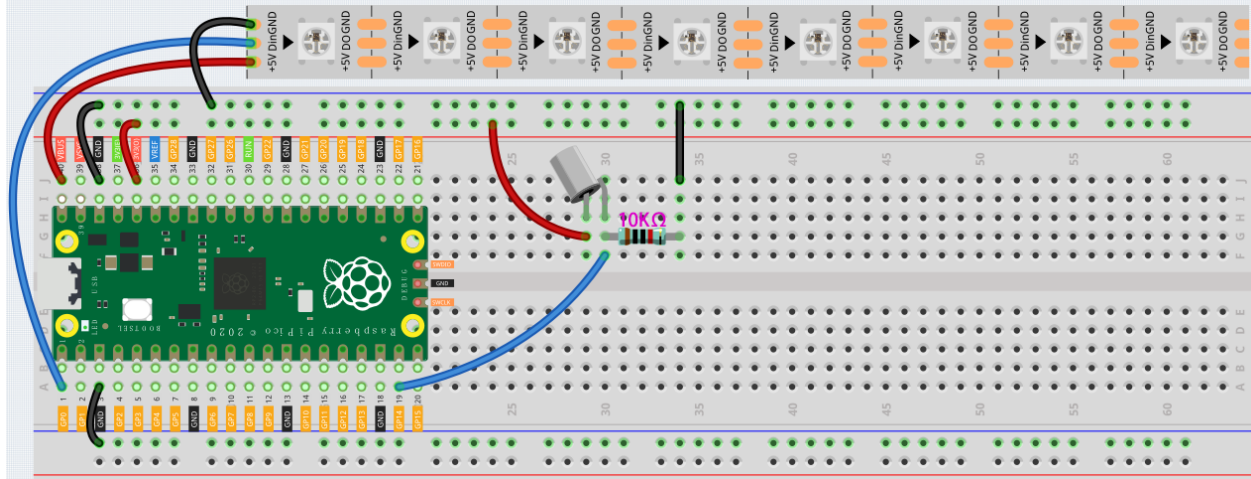
- [servo pin () set angle to ()]: This block is used to set the rotation angle of the servo, the range is 0~180 degrees.

5.10 Neopixel

The kit is equipped with a WS2812 RGB LED Strip, which can display colorful colors, and each LED can be independently controlled.
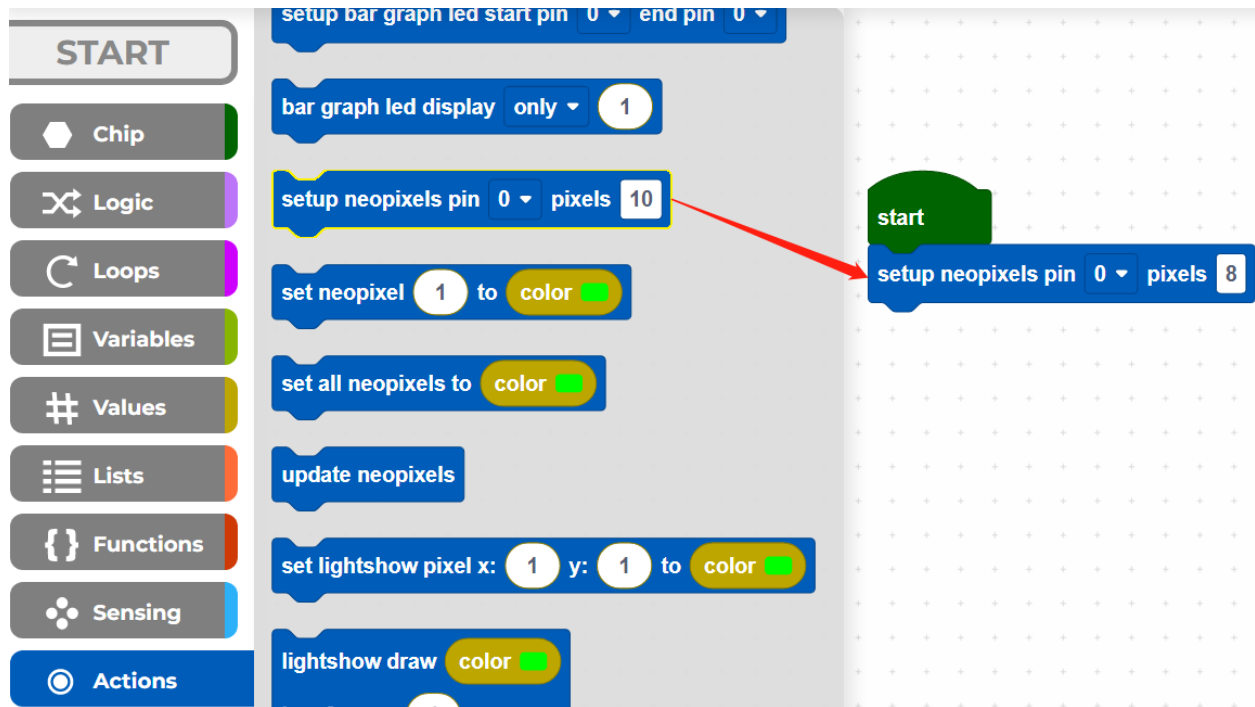
Here we try to use the tilt switch to control the flow direction of the LEDs on the WS2812 RGB LED Strip.
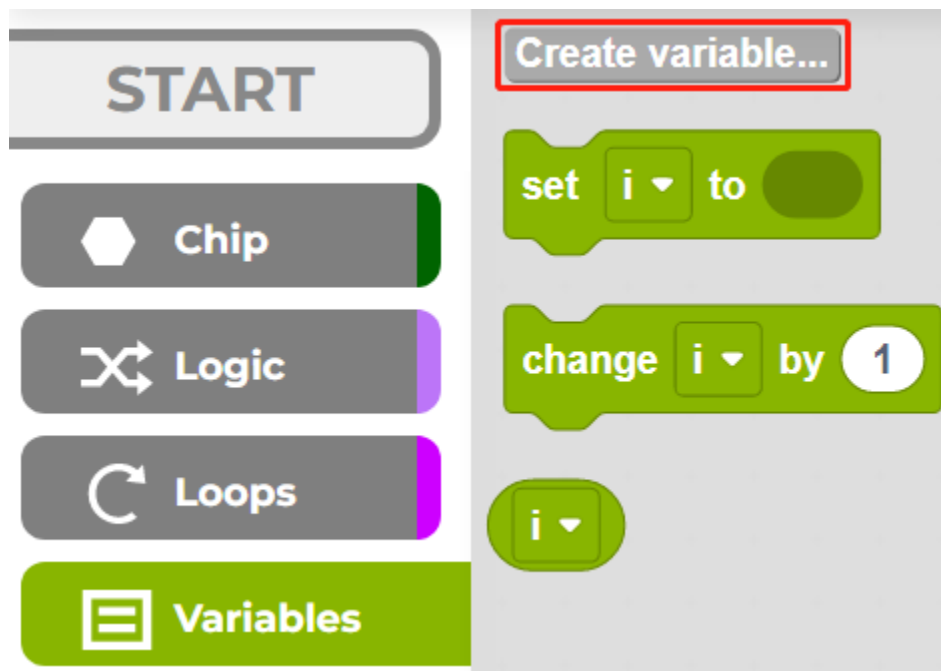
Wiring

Programming

Step 1: Use the [setup neopixel pin() pixels()] block in the **Actions** palette to initialize the WS2812 RGB LED Strip. **0** means the connected pin is GP0 and **8** means there are 8 RGB LEDs on the WS2812 RGB LED Strip.

Step 2: In the **Variables** palette, click the **Create variable** button to create a variable called **i** to represent the LEDs on the WS2812 RGB LED Strip.


Step 3: Set the initial value of variable **i** to 1 (the LED near the wires), then in [repeat forever] block, use [() mod ()] to set the value of i from 0 to 7. e.g. 1 mod 8 = 1… 8 mod 8 =0, 9 mod 8 =1, etc.

- [() mod ()]: This is the modulo operator block, from the **Loops** palette, drop down [() = ()] to select **mod**.
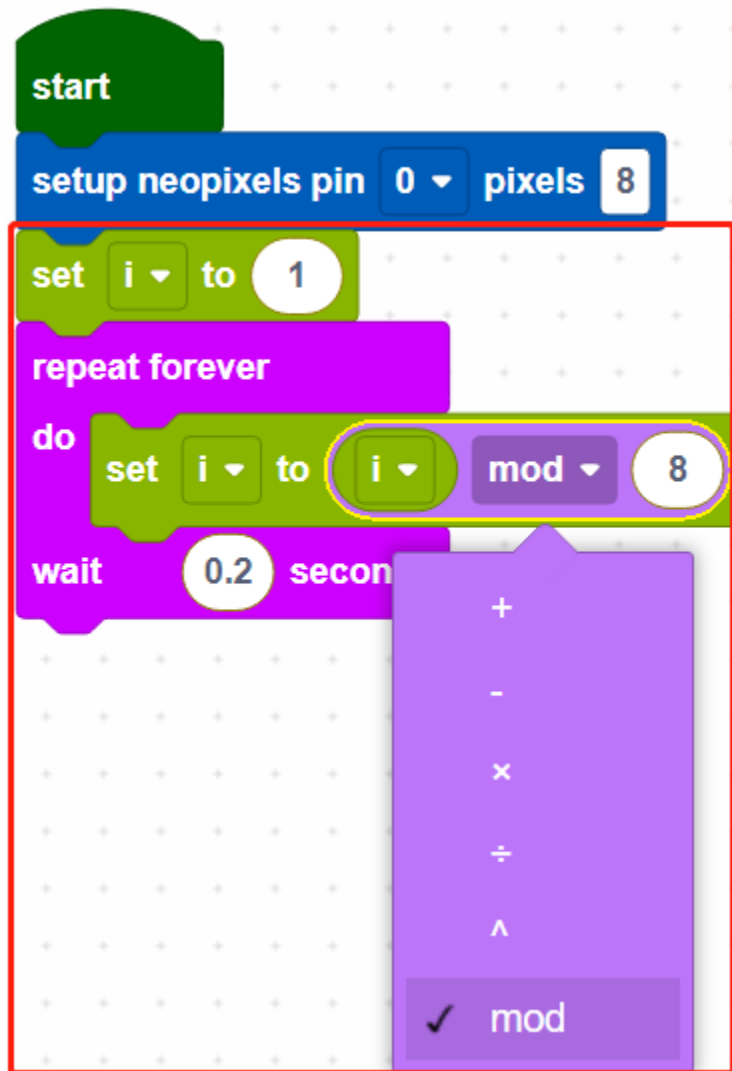
Step 4: Set all neopixels to black to make all LEDs go off, then use [updates neopixels] to make this effect update to the WS2812 RGB LED Strip.
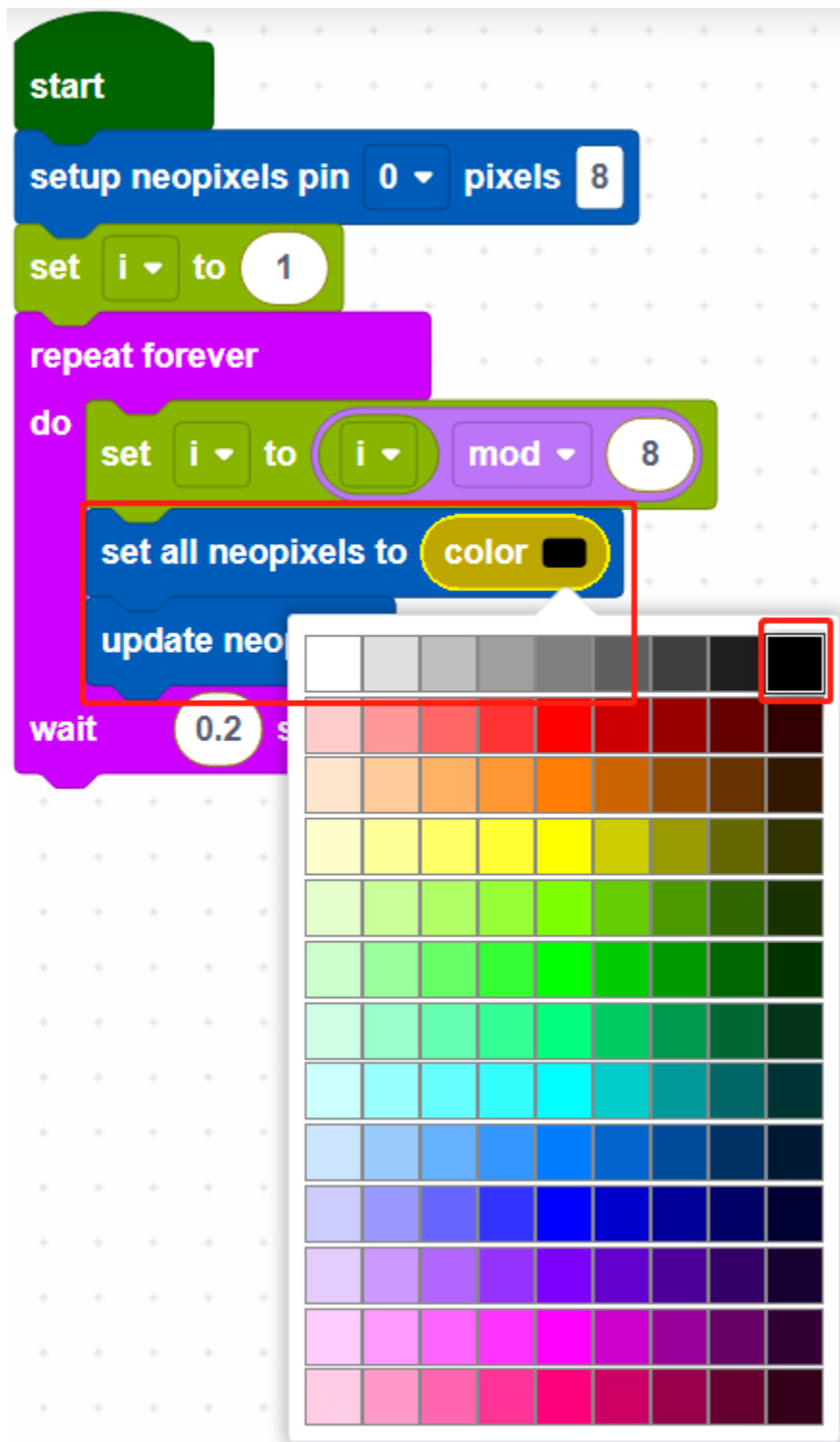
- [set all neopixels to ()]: Use to set a color for all LEDs, there are 13*9 colors, the top right color is black to make LEDs to go off.

- [updates neopixels]: Update the effect to the WS2812 RGB LED Strip.

Step 5: If pin14 is read high, let the LEDs on the WS2812 RGB LED Strip light up one by one in green, otherwise light up green one by one in the opposite direction.
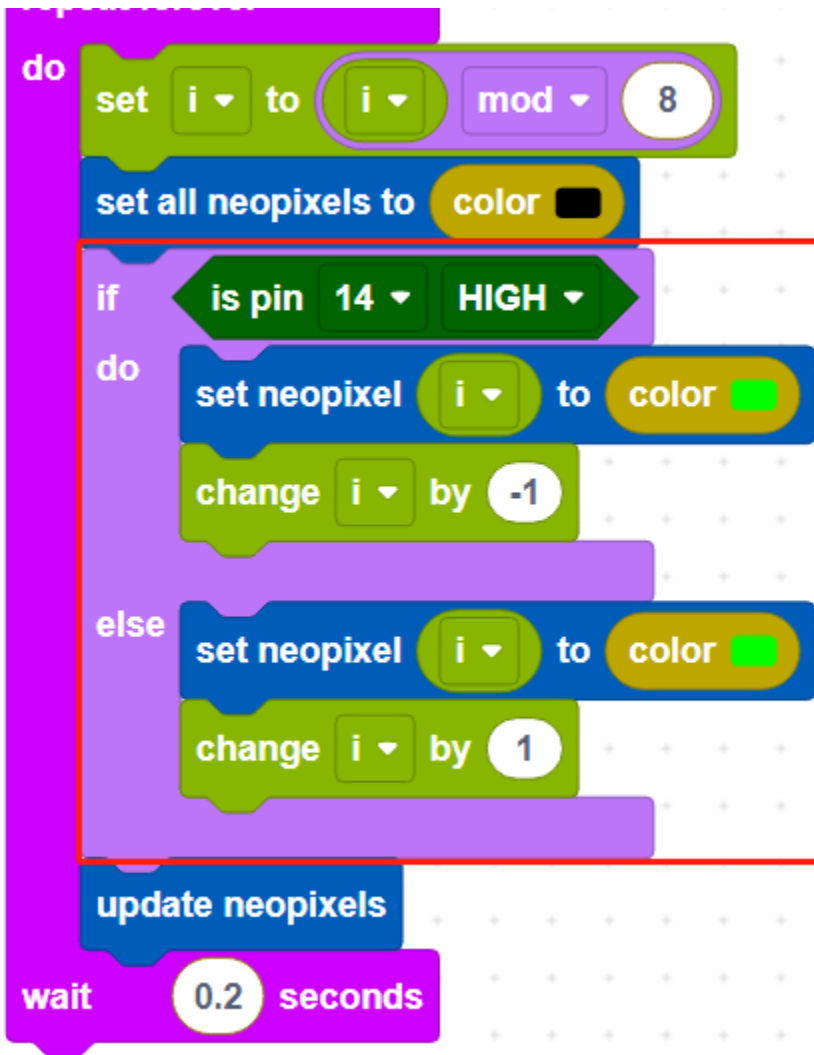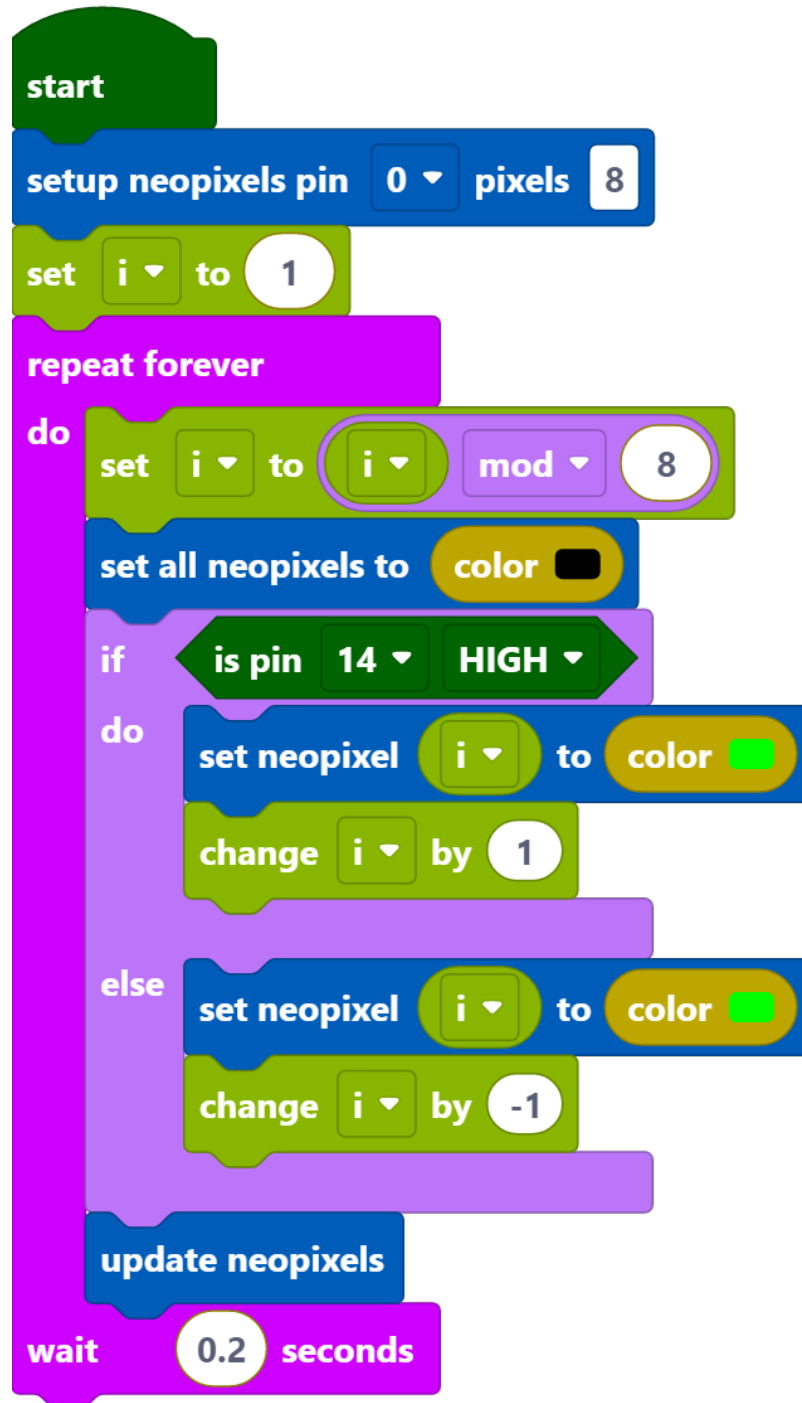
- [change () by ()]: Used to increase (positive) or decrease (negative) the value of a variable by a specific step.
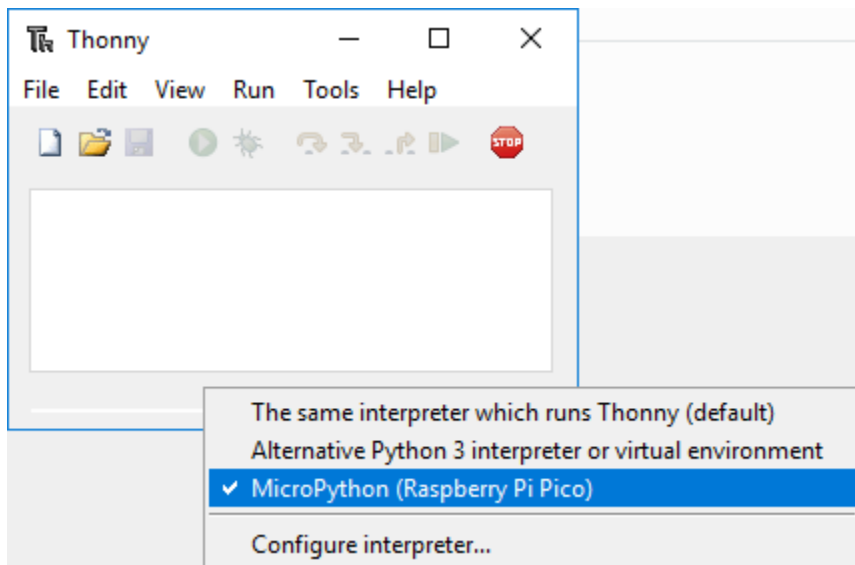
Code

After connecting Pico, click the **Start** button and the code starts to run.

When the tilt switch is placed vertically, it makes the LEDs on the WS2812 RGB LED Strip light up one by one in green, and when the tilt switch is placed horizontally, the LEDs light up one by one in the opposite direction in green.

start

setup neopixels pin `0 ▼` pixels `8`

set `i ▼` to `1`

repeat forever

do

    set `i ▼` to (`i ▼` mod ▼ `8`)

    set all neopixels to (color ⬛)

    if (is pin `14 ▼` `HIGH ▼`)

    do   set neopixel `i ▼` to (color 🟩)

        change `i ▼` by `1`

    else   set neopixel `i ▼` to (color 🟩)

        change `i ▼` by `-1`

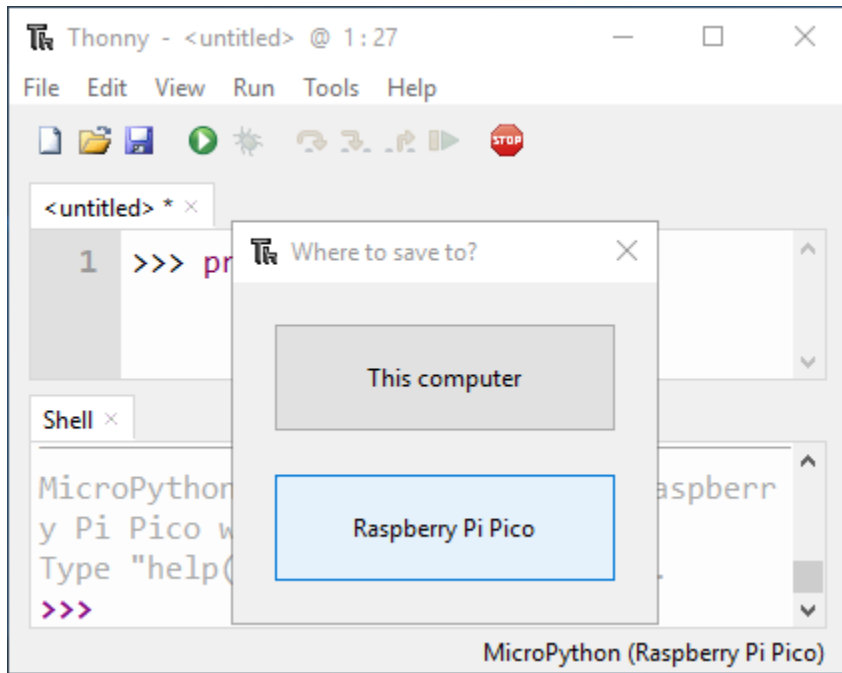   update neopixels

wait `0.2` seconds

FAQ

6.1 Q1: NO MicroPython(Raspberry Pi Pico) Interpreter Option on Thonny IDE?

- Check that your Pico is plugged into your computer via a USB cable.

- Check that you have installed MicroPython for Pico (*1.3 Install MicroPython on Your Pico*).

- The Raspberry Pi Pico interpreter is only available in version 3.3.3 or higher version of Thonny. If you are running an older version, please update (*1.2 Install Thonny IDE*).

- Plug in/out the micro USB cable sveral times.

6.2 Q2: Cannot open Pico code or save code to Pico via Thonny IDE?

- Check that your Pico is plugged into your computer via a USB cable.
- Check that you have selected the Interpreter as **MicroPython (Raspberry Pi Pico)**.

6.3 Q3: Can Raspberry Pi Pico be used on Thonny and Arduino at the same time?

NO, you need to do some different operations.

- If you used it on Arduino first, and now you want to use it on Thonny IDE, you need to *1.3 Install MicroPython on Your Pico*.
- If you used it on Thonny first and now you want to use it on Arduino IDE, you need to *Setup the Raspberry Pi Pico*.

6.4 Q4: Code upload failed in Arduino IDE?

- Make sure you have installed the Pico board in the Arduino IDE.
- Check that the Board(Raspberry Pi Pico) or portCOMxx (Raspberry Pi Pico)is selected correctly.
- Make sure you have plugged the Pico into your computer.
- If only COMxx (The complete one should be COMxx (Raspberry Pi Pico)) is displayed, it means that Pico is not correctly recognized by the computer. You need to refer to steps 5, 6, and 7 in *Setup the Raspberry Pi Pico*.

6.5 Q5: If your computer is win7 and pico cannot be detected.

1. Download the USB CDC driver from http://aem-origin.microchip.com/en-us/mindi-sw-library?swsearch= Atmel%2520USB%2520CDC%2520Virtual%2520COM%2520Driver

2. Unzip the amtel_devices_cdc.inf file to a folder named "pico-serial"

3. Change the name of amtel_devices_cdc.inf file to pico-serial.inf

4. Open/edit the pico-serial.inf in a basic editor like notepad

5. Remove and replace the lines under the following headings:

```
[DeviceList]
%PI_CDC_PICO%=DriverInstall, USB\VID_2E8A&PID_0005&MI_00

[DeviceList.NTAMD64]
%PI_CDC_PICO%=DriverInstall, USB\VID_2E8A&PID_0005&MI_00

[DeviceList.NTIA64]
%PI_CDC_PICO%=DriverInstall, USB\VID_2E8A&PID_0005&MI_00

[DeviceList.NT]
%PI_CDC_PICO%=DriverInstall, USB\VID_2E8A&PID_0005&MI_00

[Strings]
Manufacturer = "ATMEL, Inc."
PI_CDC_PICO = "Pi Pico Serial Port"
Serial.SvcDesc = "Pi Pico Serial Driver"
```

6. Close and save and make sure your retain the name as pico-serial.inf

7. Go to your pc device list, find the pico under Ports, named something like CDC Device. A yellow exclamation mark indicates it.

8. Right click on the CDC Device and update or install driver choosing the file you created from the location you saved it at.

THANK YOU

Thanks to the evaluators who evaluated our products, the veterans who provided suggestions for the tutorial, and the users who have been following and supporting us. Your valuable suggestions to us are our motivation to provide better products!

Particular Thanks

- Len Davisson

- Kalen Daniel

- Juan Delacosta

Now, could you spare a little time to fill out this questionnaire?

Note: After submitting the questionnaire, please go back to the top to view the results.

COPYRIGHT NOTICE