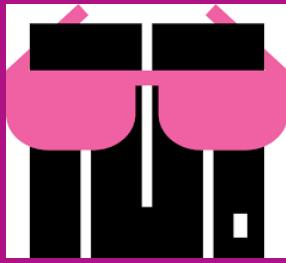
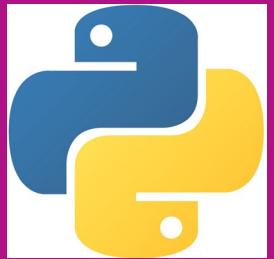
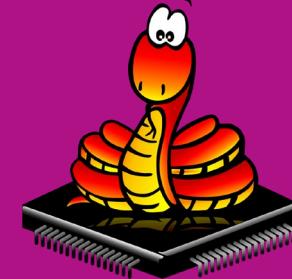


# INTERNET OF THINGS (IOT) PROJECTS USING PYTHON

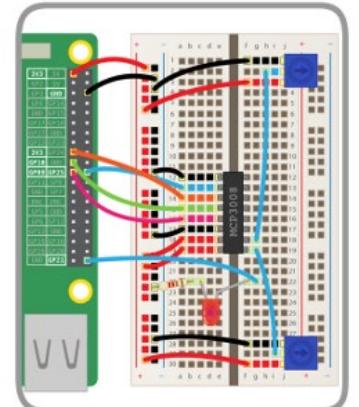
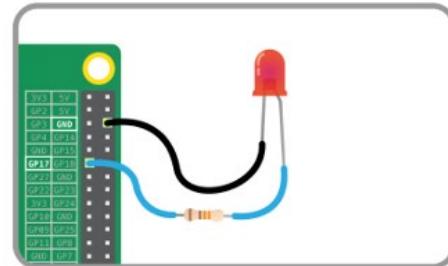
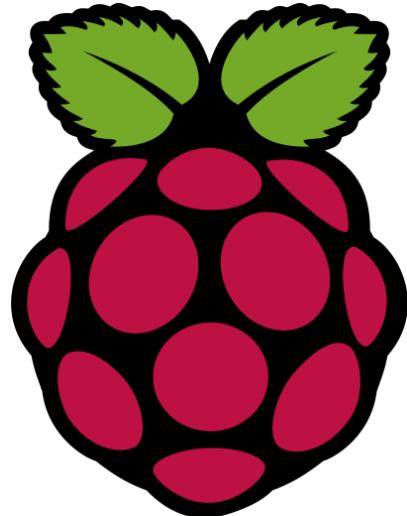


(CSE 4110)

(LECTURE – 1)

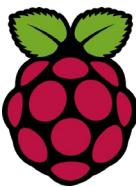


T<sub>h</sub>





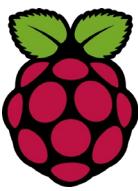
# What Is Outcome-Based Education? OBE Vs Traditional Education System



- Outcome-based education is a system where all the parts and aspects of education are focused on the outcomes of the course. The students take up courses with a certain goal of developing skills or gaining knowledge and they have to complete the goal by end of the course.
  
- There is no specific style or time limit of learning. The student can learn as per their choice. The faculty members, moderators, and instructors guide the students based on the target outcomes.



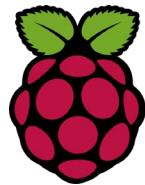
# Benefits Of Outcome-Based Education (OBE) For Students



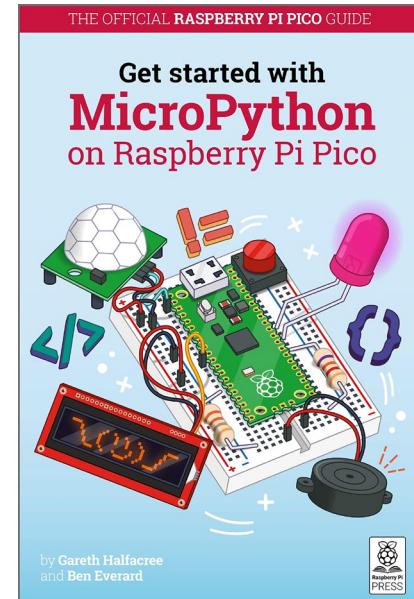
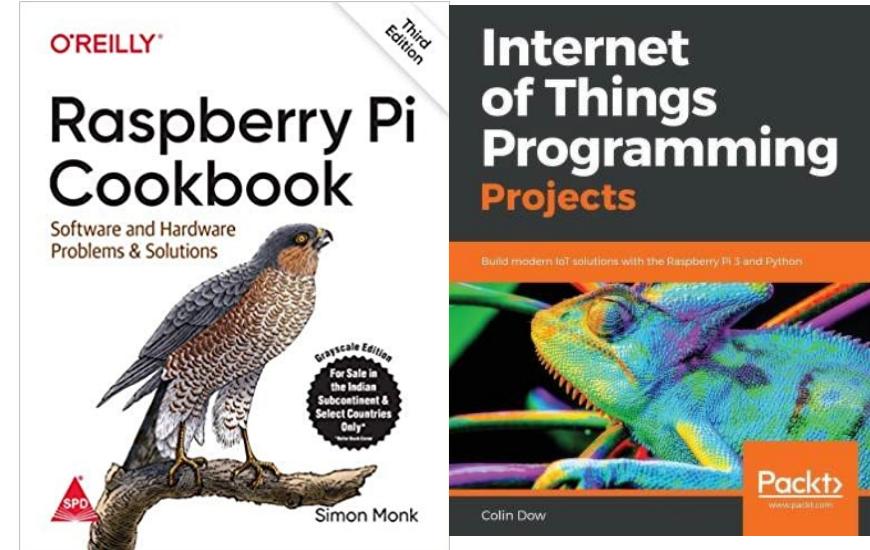
- Brings clarity among the teachers and students
- Every student has the flexibility and freedom of learning in their ways.
- There is more than one method of learning
- Reduces comparison among the students as everyone has a different target
- Completely involves students taking responsibility for their goals



# About your subject

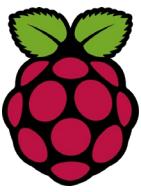


- Grading Pattern : 3
- Credit : 4
- TextBook:
  - 1) Raspberry Pi Cookbook by by Simon Monk, shroff/O'Reilly
  - 2) Internet of Things Programming Projects by By Colin Dow, Packt,
- Reference : 3) Get started with Micro-Python





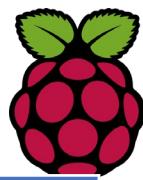
# Evaluation Scheme



- ATTENDANCE : 5
- WEEKLY ASSIGNMENTS / QUIZZES : 20 MID-TERM
- MID TERM : 15
- TOTAL INTERNAL : 40
  
- FINAL ASSIGNMENT : 40
- ASSIGNMENT PRESENTATION : 20 END-TERM
- TOTAL EXTERNAL : 60



# Course Outcomes

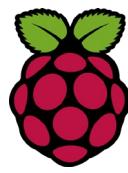


## Course Outcomes

CO1	Understand general concepts of Internet of Things (IoT), the working of Raspberry Pi and its features.
CO2	Recognize various components, sensors, actuators, devices and their applications.
CO3	Analyze various python programs to interface with sensors, actuators, LED's, cloud and camera using Raspberry pi.
CO4	Measure physical parameters using sensors.
CO5	Demonstrate the ability to transmit data wirelessly between different devices to build simple IoT systems using Raspberry Pi.
CO6	Create IoT devices and systems through a variety of interfaces, including web apps and mobile apps.

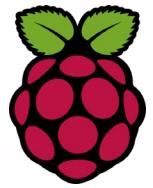


# What Students will learn?

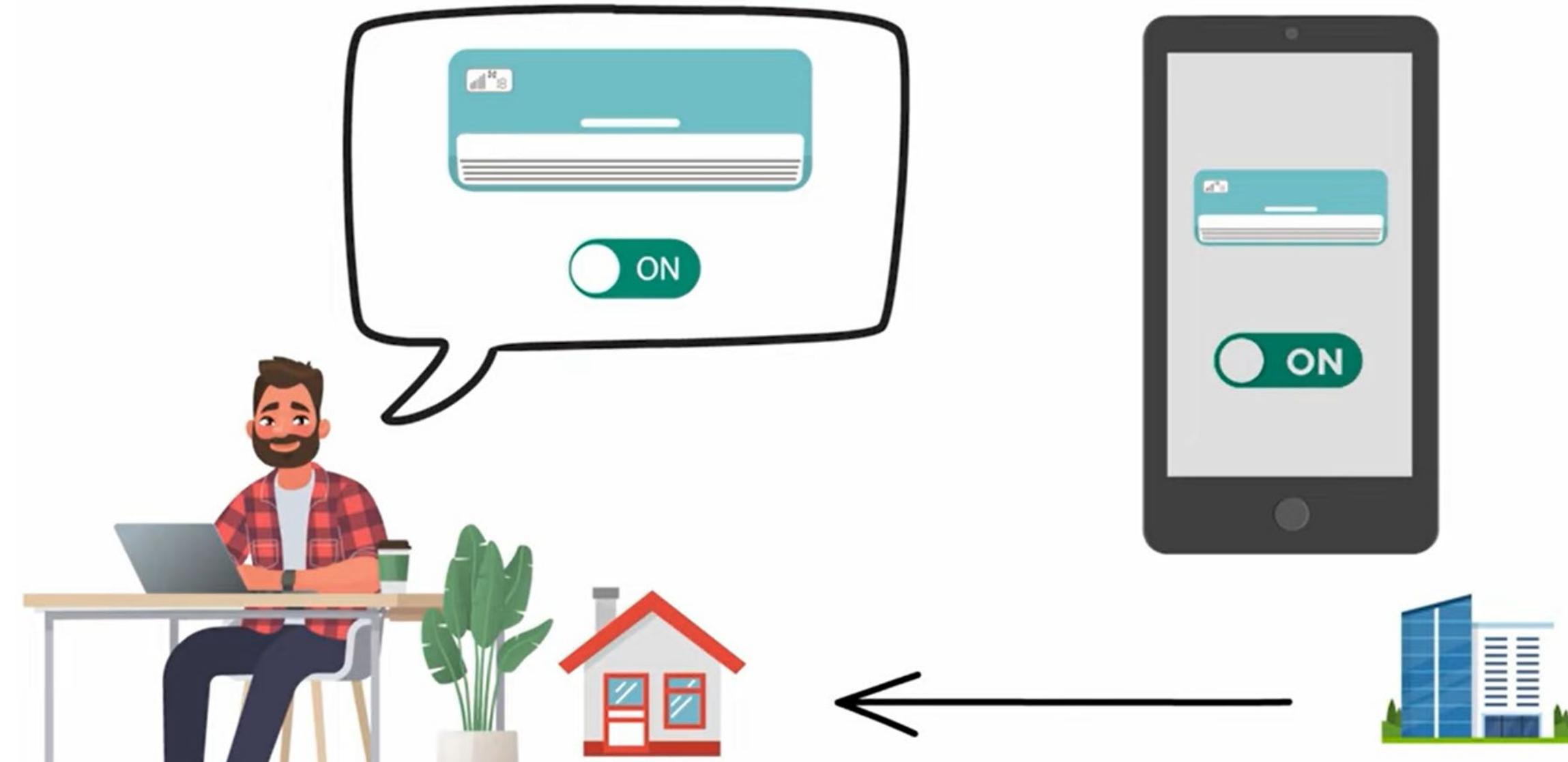


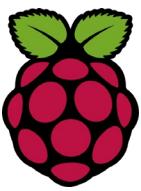
Students will get a glimpse into the ever-growing field of IoT, a technology that allows the intelligent exchange of data among a network of internet-connected objects. Also students will experience about “RP2040” chip which provides ample power for embedded as well as IOT projects and enables users of any age or ability to learn coding and electronics.

- ✓ Familiarization with IoT concepts, their origin, impact, methodologies and tools, and how IoT is integrated into different applications to improve technical as well as business results.
- ✓ Set up your Raspberry Pi Pico and start using it
- ✓ Prepare Raspberry Pi Pico with header for breadboard implementation.
- ✓ Introduction to the ONLINE SIMULATOR “WOKWI”

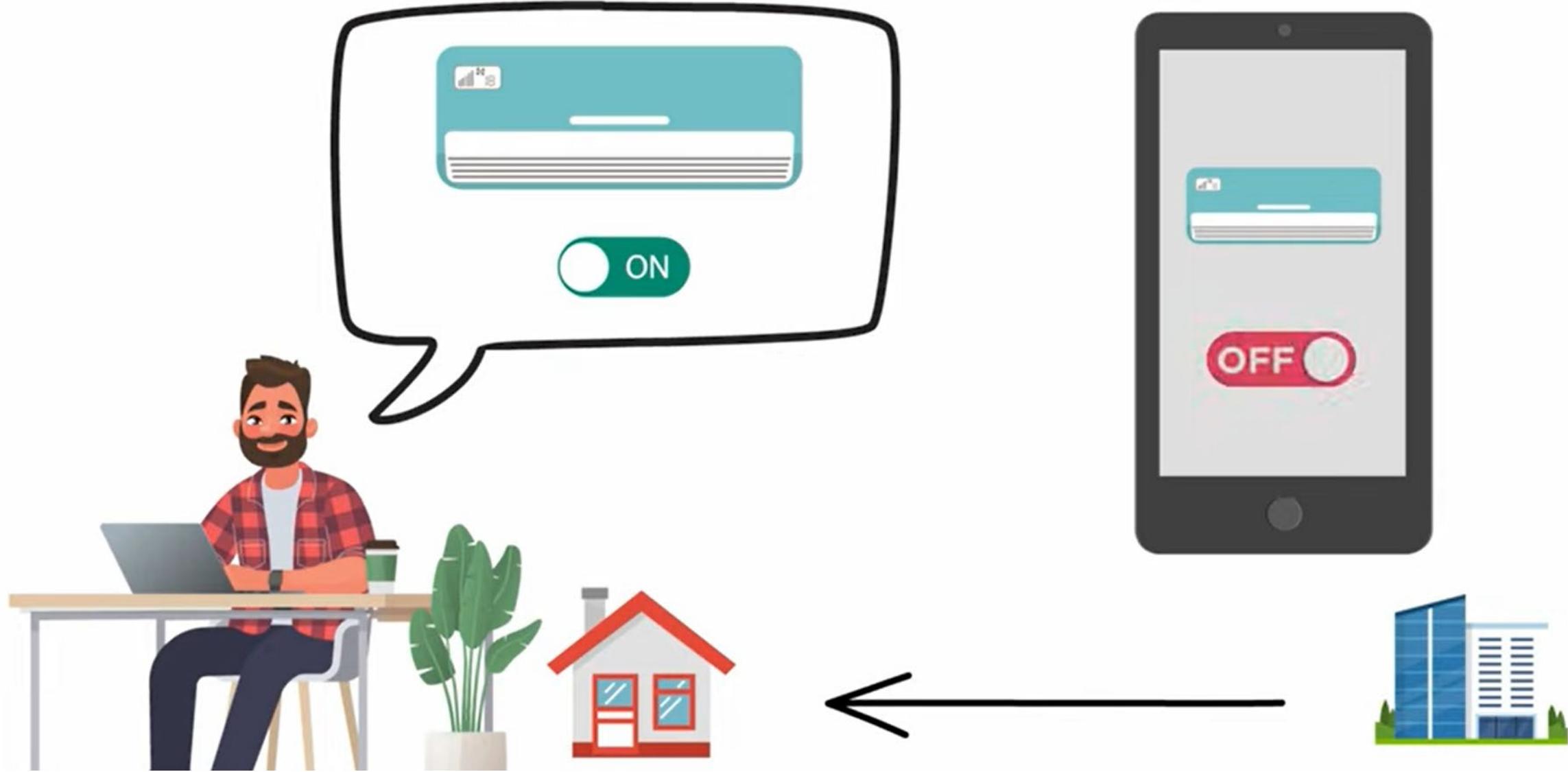


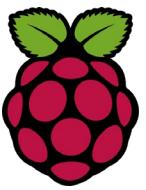
# Want to know status of my AC?



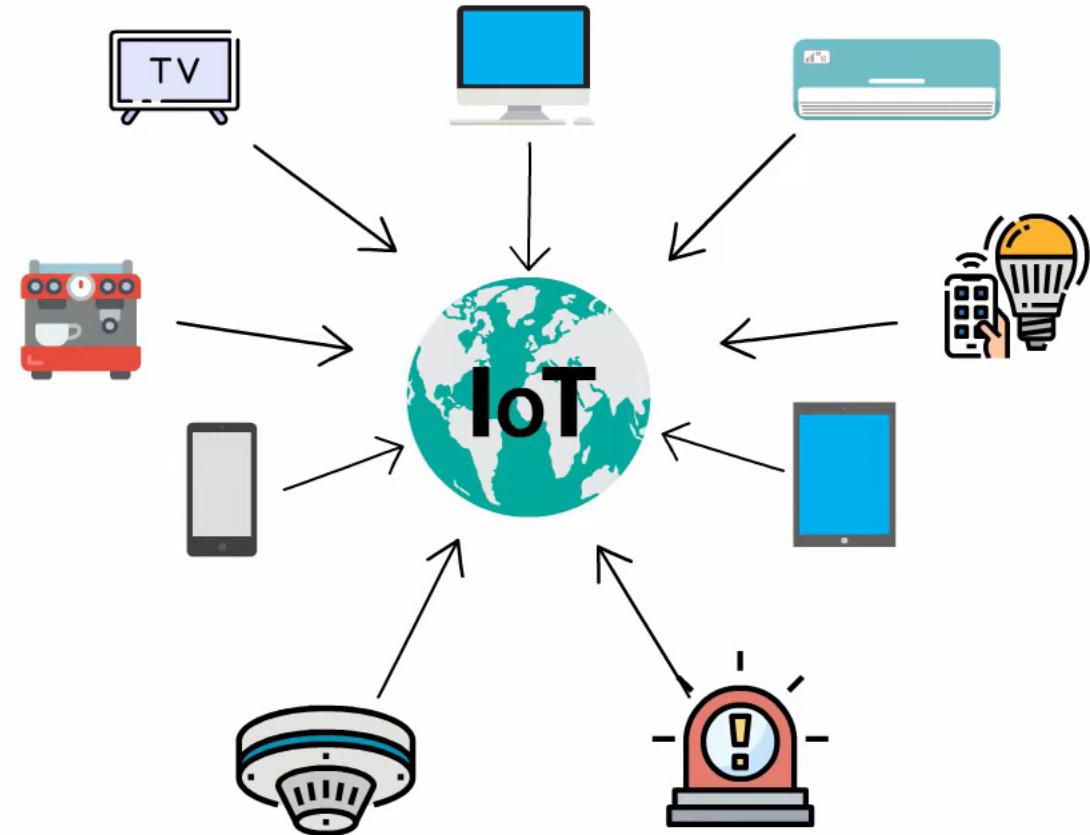


# Can I turn it off?

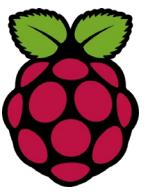




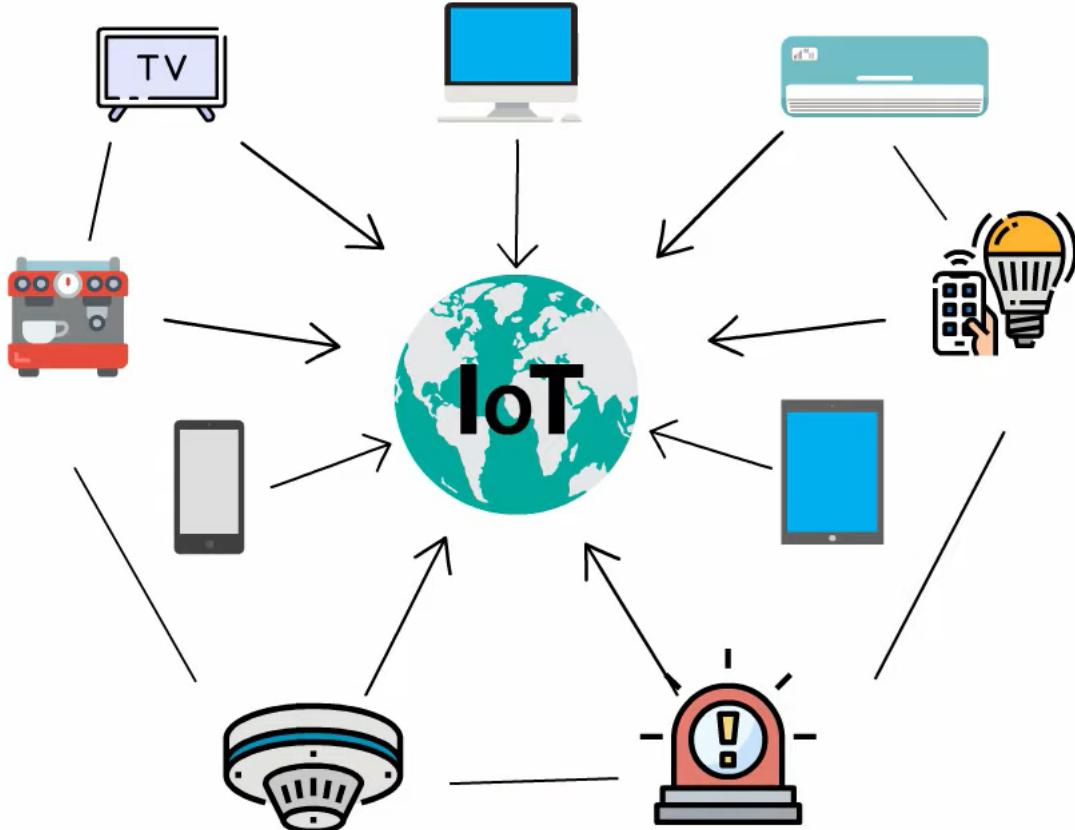
# What IoT can do?



IoT is shaping the way we live our lives



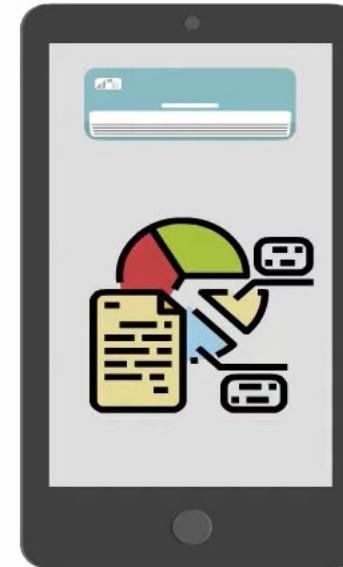
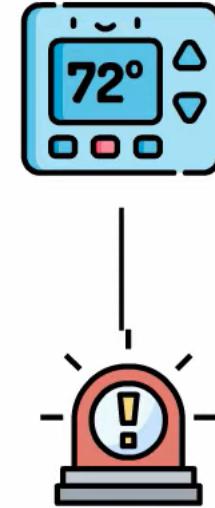
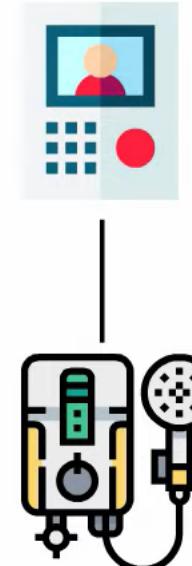
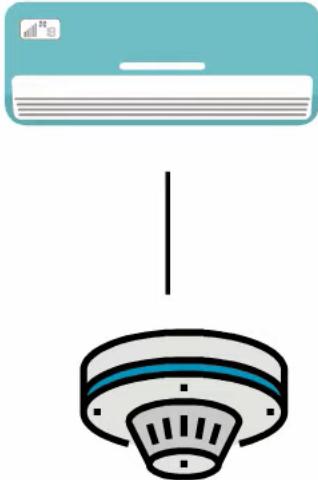
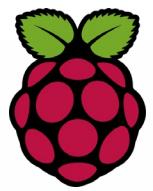
# What is IOT?

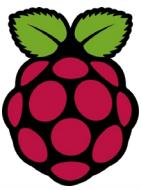


**Internet of Things (IOT) is a system of inter related devices connect to the internet to transfer and receive data from one to the other.**



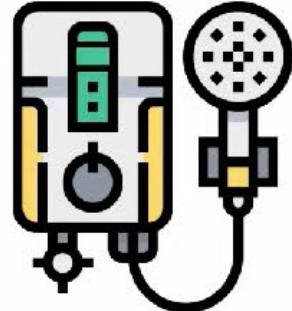
# A smart home is the best example of IoT



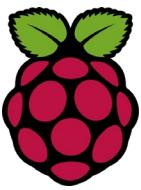


# Internet of Things

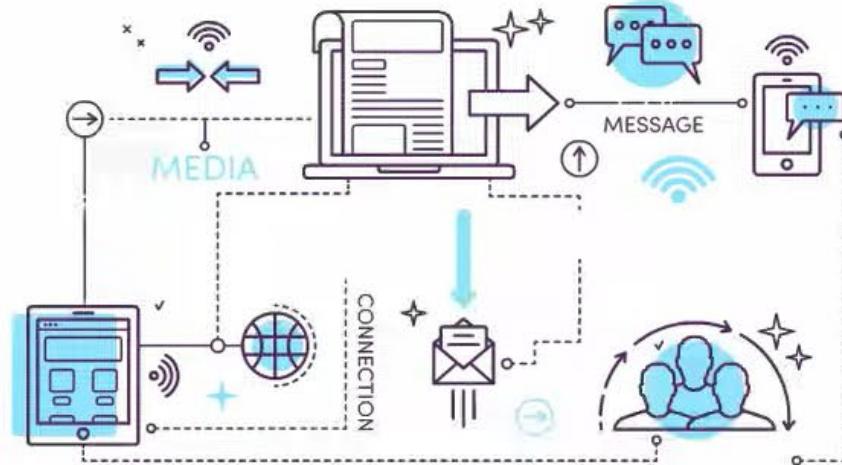
For example:



Although all of this is fascinating, there's a lot that goes on in the background to ensure seamless functioning



# Internet of Things

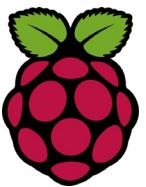


Communication  
between Devices

## Components

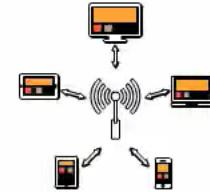


Accurate processing of  
received data

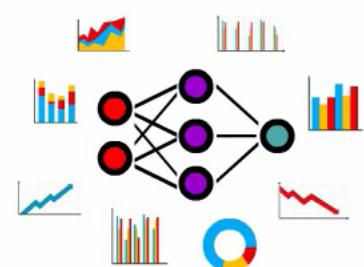
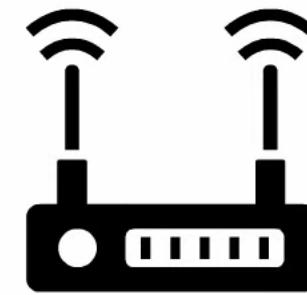
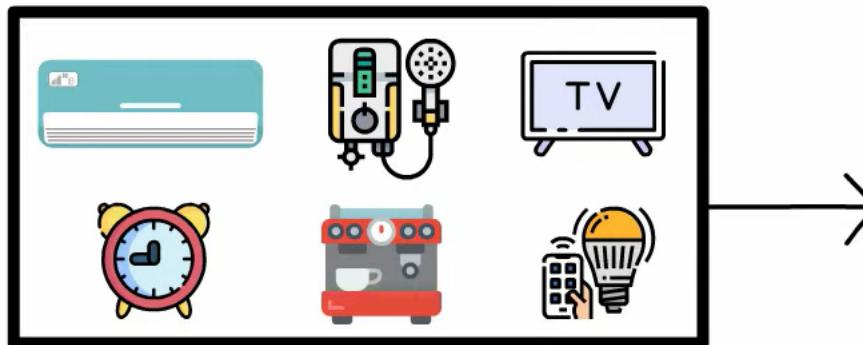
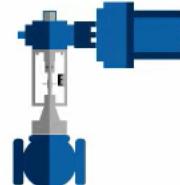


# IoT devices

General devices

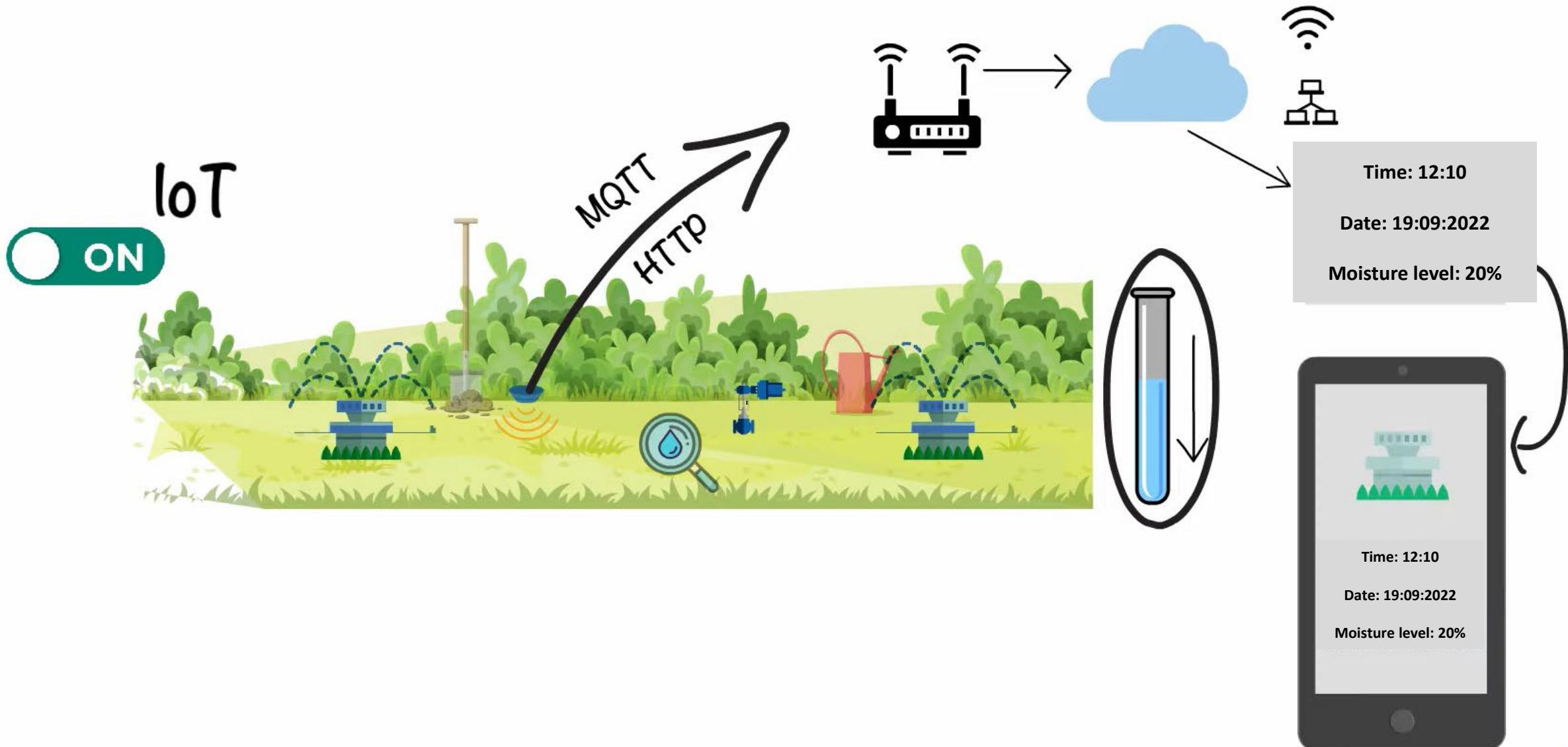
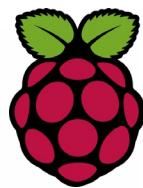


Sensing devices

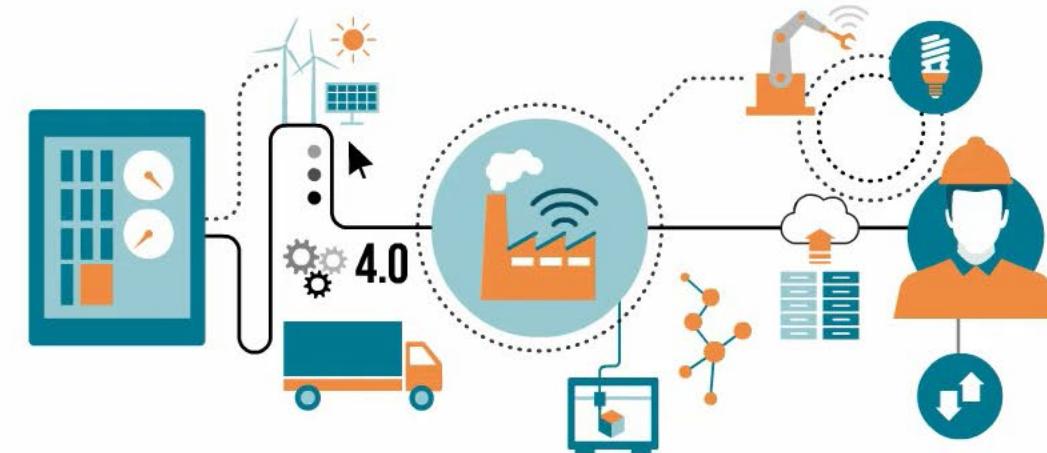
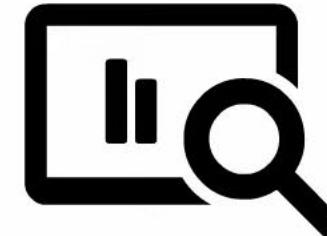




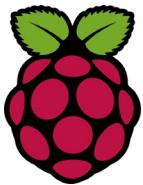
# To help you understand its working, let's take a simple scenario



# Internet of Things



Today, IoT is being used extensively to lessen the burden on humans



# Internet of Things

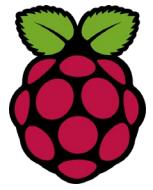
Today, IoT is being used extensively to lessen the burden on humans



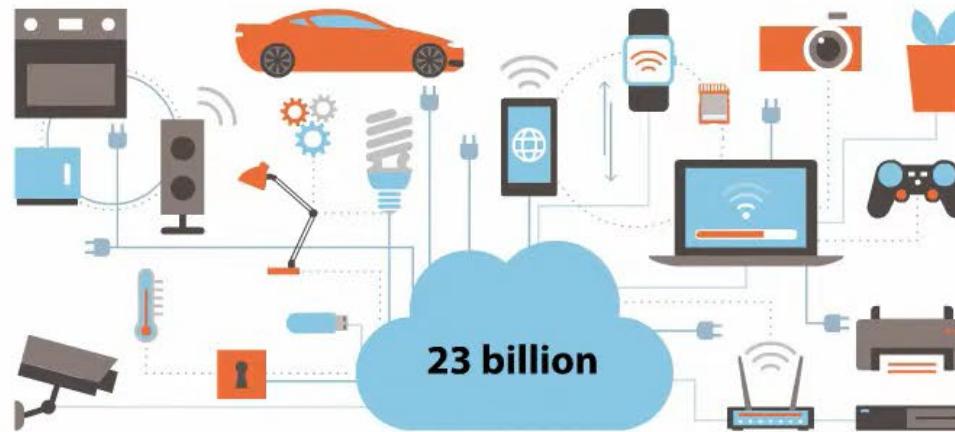
Promising than ever before



# Internet of Things



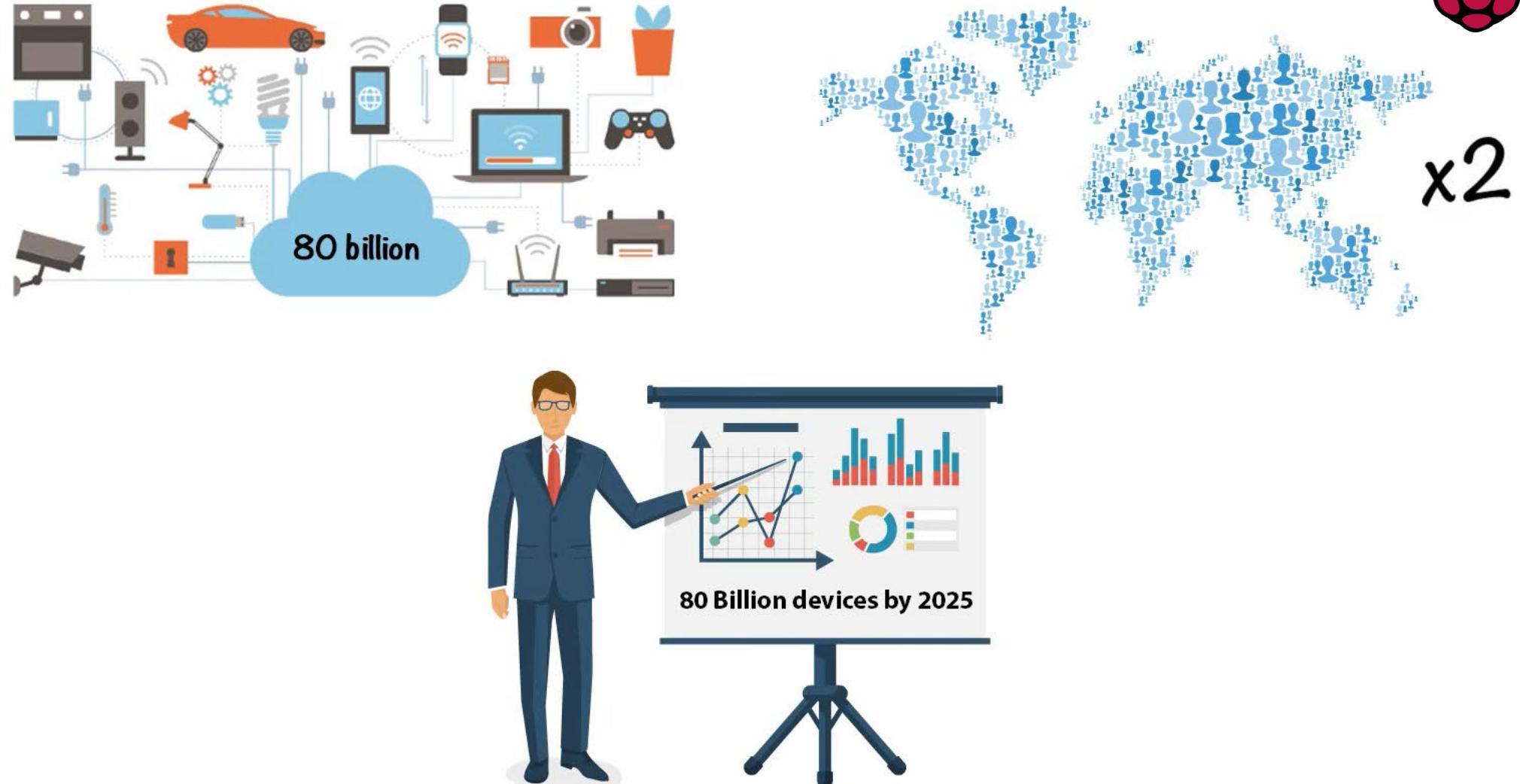
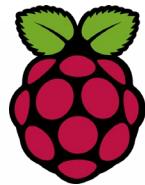
In 2018:



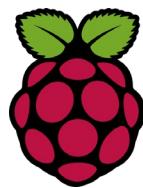


In 2025:

# Internet of Things



IoT is a vision to connect all devices with the power of the internet,  
always learning and always growing

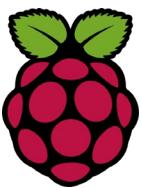


# QUIZ TIME???????

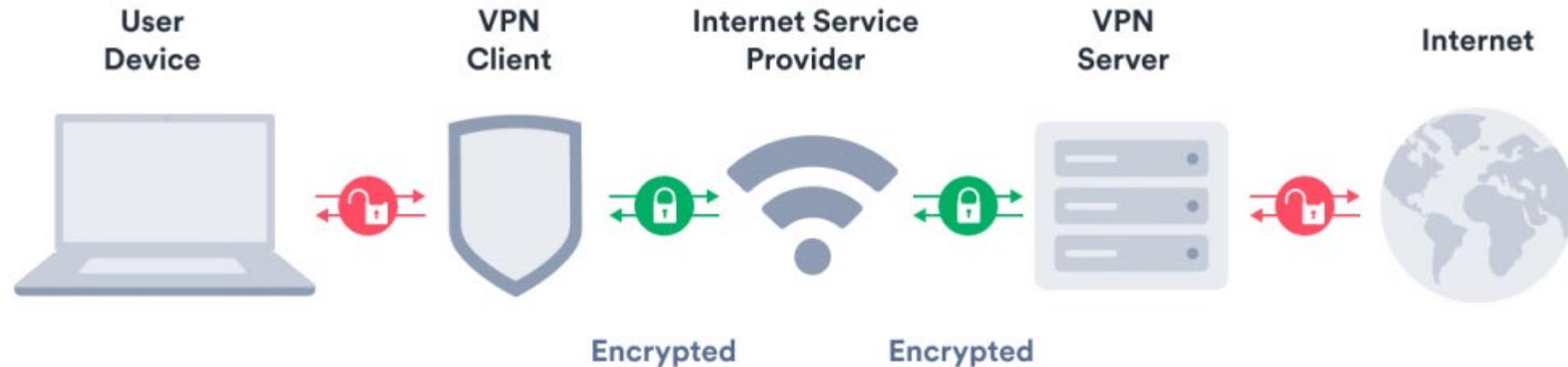


Which of the following technologies is not used as an interface for a network?

1. WiFi
2. Ethernet
3. ZigBee
4. VPN



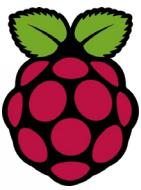
# Answer – D: VPN



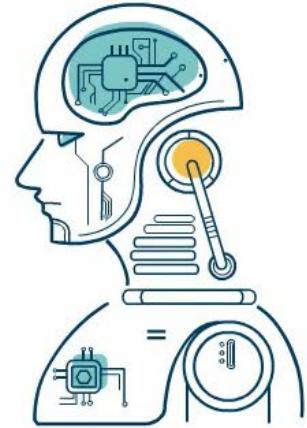
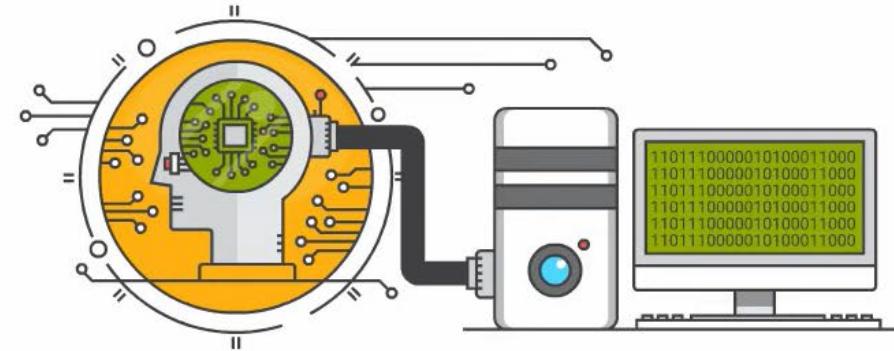
## VPN

A VPN (Virtual Private Network) **protects your connection while you're using the internet**. It makes your browsing private, hides your IP (Internet Protocol) address and ensures your internet service provider (ISP) doesn't track you.

**VPN is used to extend a private network over a public. The other options are used to provide a medium for data transfer.**



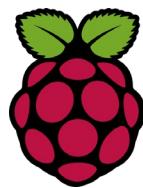
# The integration of IoT with other technologies like



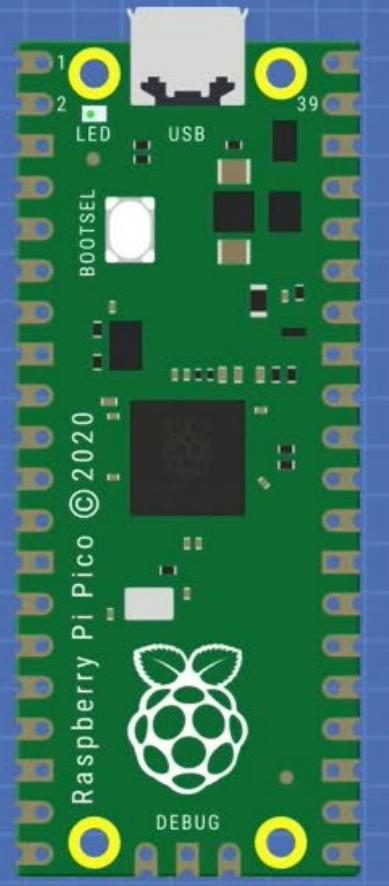
That is the  
Internet of Things  
for you in short

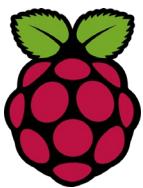


# Raspberry Pi Pico

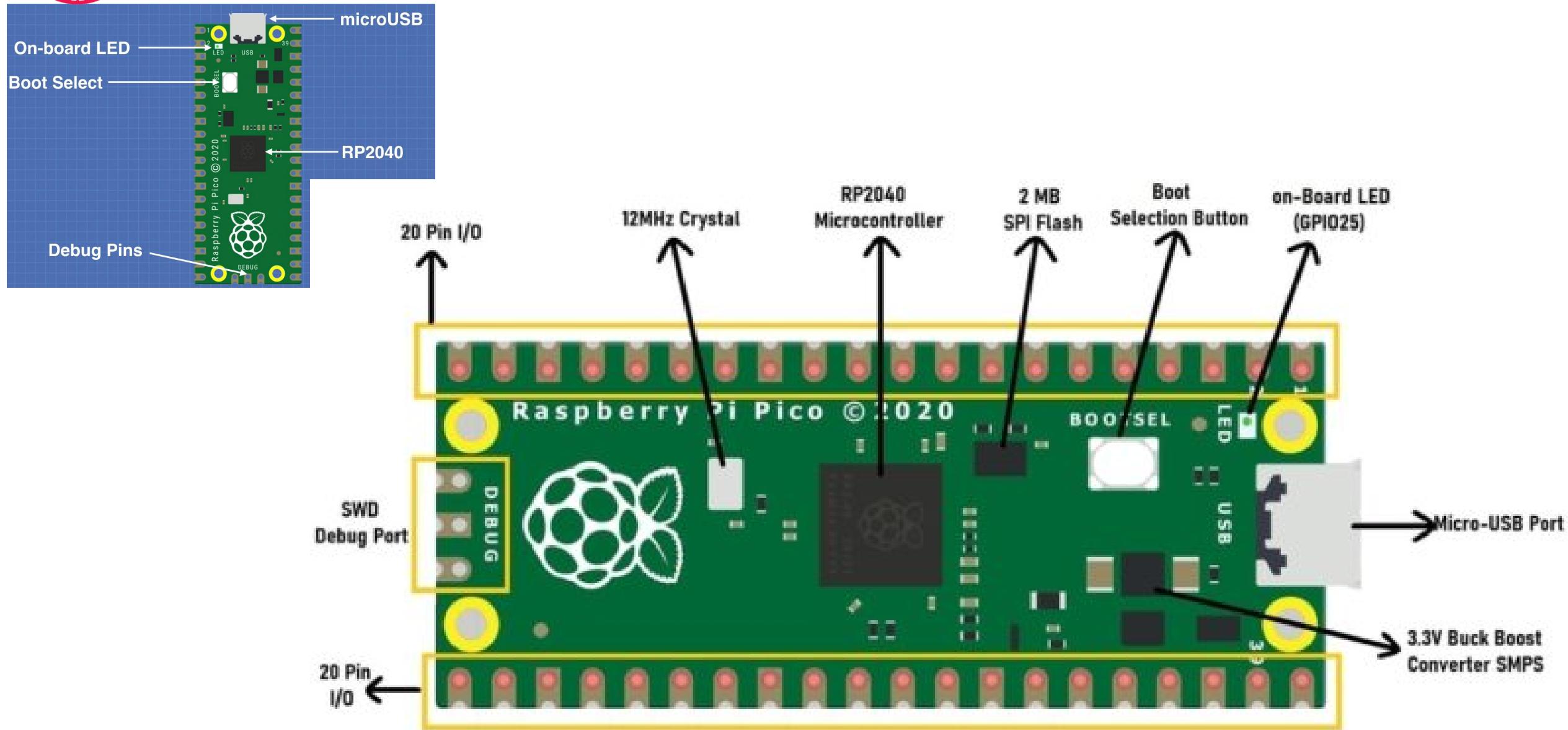


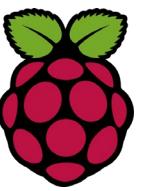
- Introduced by Raspberry Pi in January 2021
- First Microcontroller in Raspberry Pi Family
- Powered by RP2040 MCU
- Multiple PWM-capable I/O ports, I2C, SPI, ADC
- Retails for \$4.00 USD
- Other manufacturers also building with RP2040





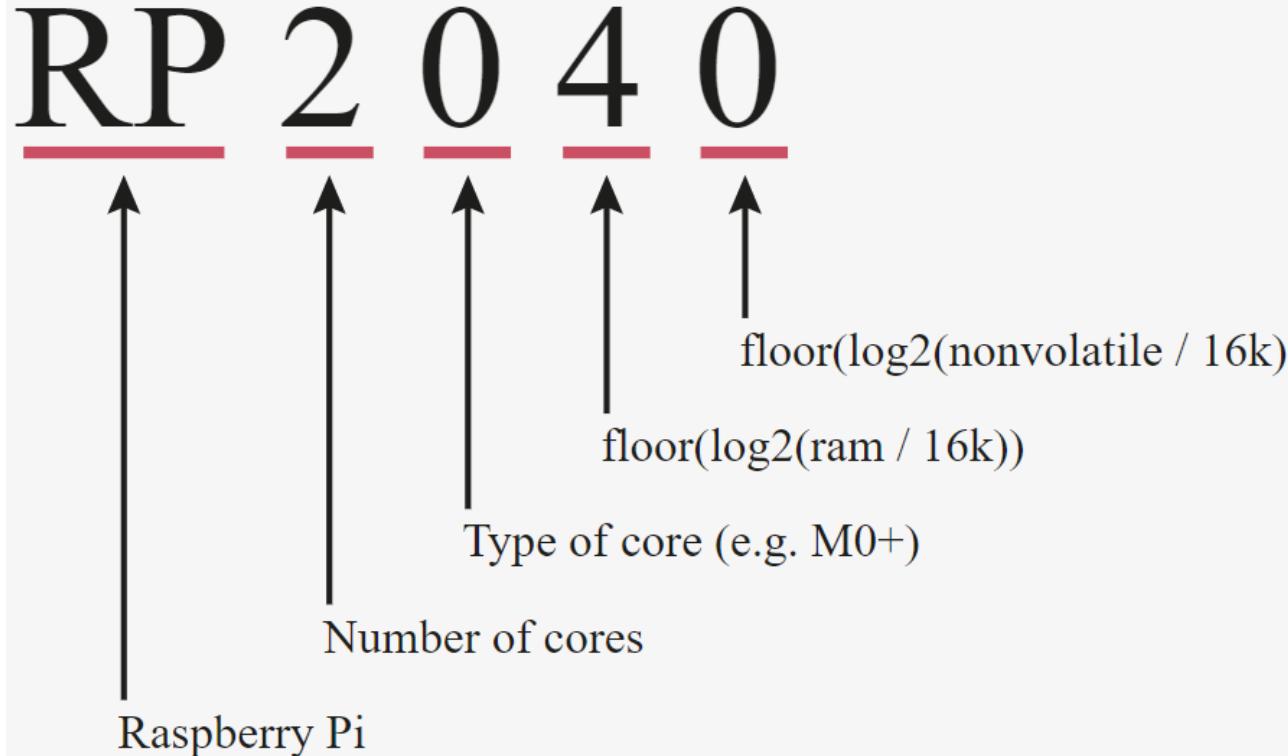
# Raspberry Pi Pico – Simple Pinout





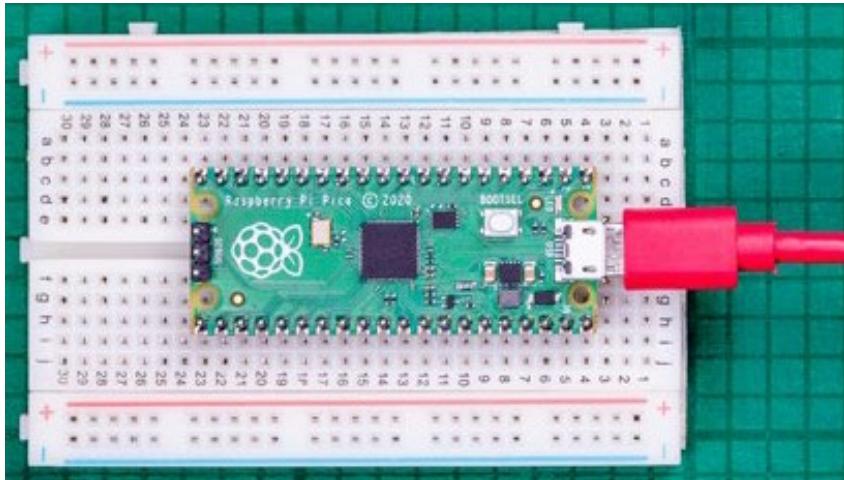
# Why is the chip called RP2040?

The post-fix numeral on RP2040 comes from the following,



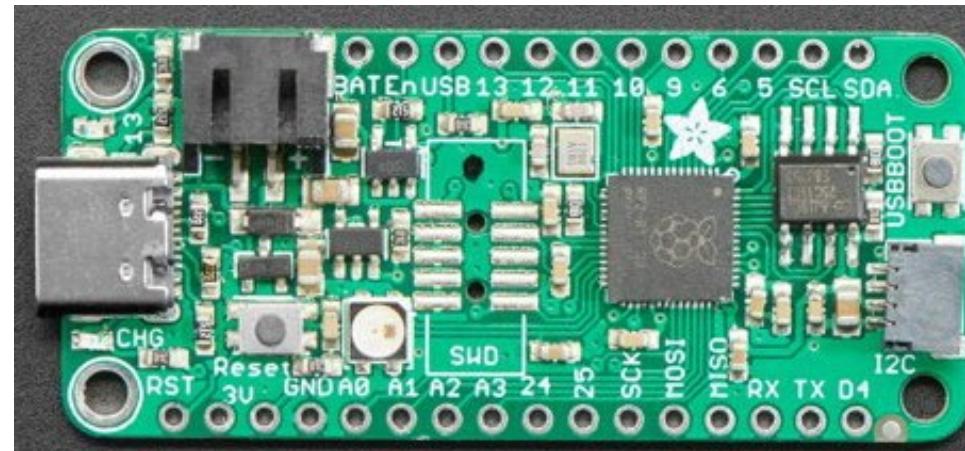
1. Number of processor cores (2)
2. Loosely which type of processor (M0+)
3.  $\text{floor}(\log_2(\text{ram} / 16\text{k}))$
4.  $\text{floor}(\log_2(\text{nonvolatile} / 16\text{k}))$  or 0 if no onboard nonvolatile storage

# RP2040-based Boards



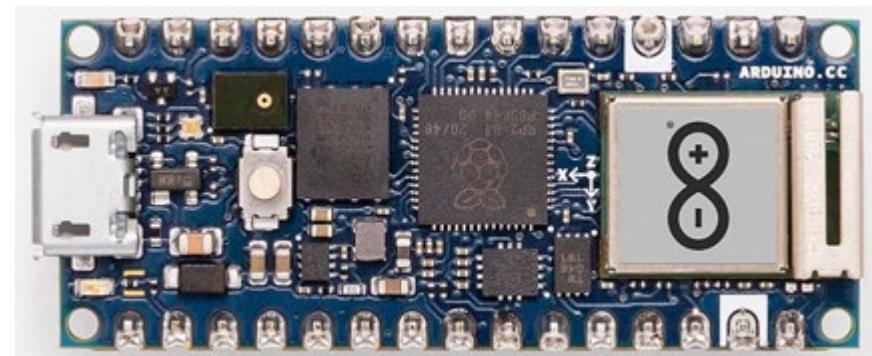
**Boards from Raspberry Pi**

Raspberry Pi Pico, and Raspberry Pi Pico W



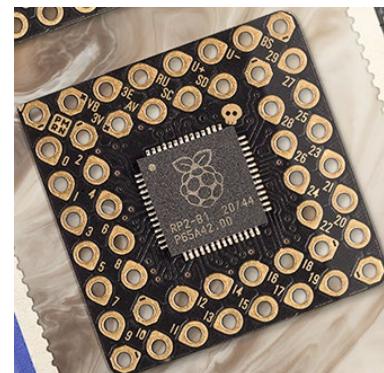
**Boards from Adafruit**

Feather 2040 and ItsyBitsy 2040



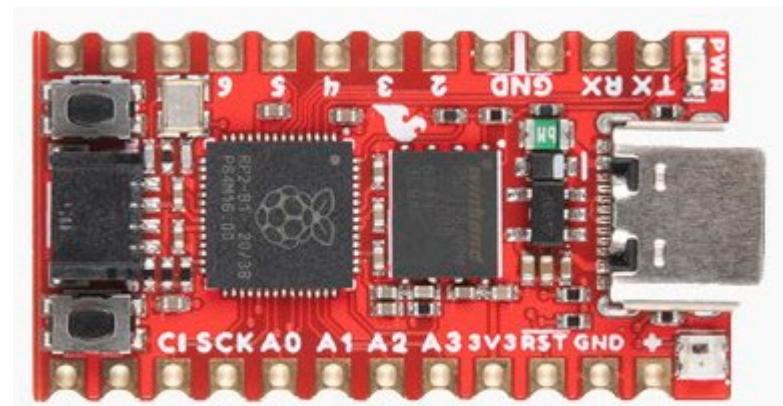
**Boards from Arduino**

Nano RP2040 Connect



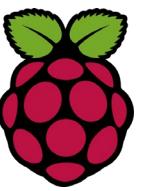
**Boards from Pimoroni**

PGA2040, Pico LiPo, Tiny 2040,  
Keybow 2040, PicoSystem, Plasma 2040



**Boards from SparkFun**

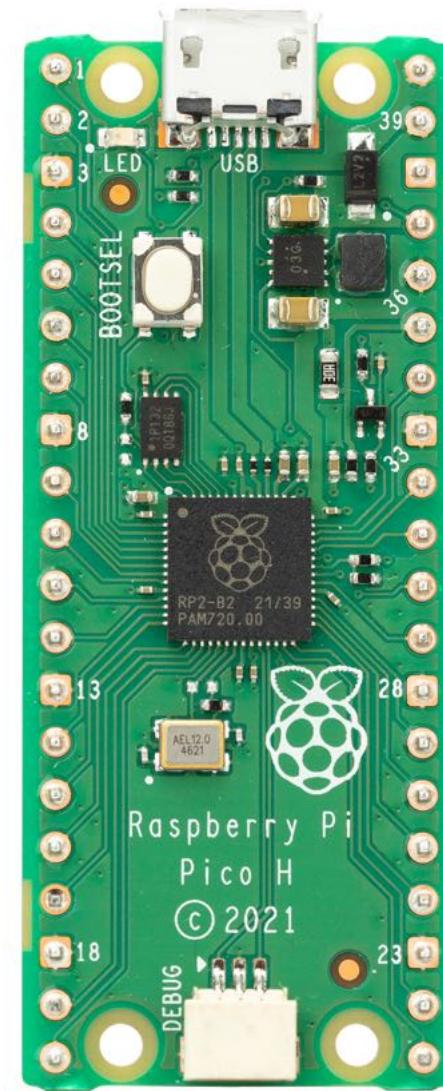
Pro Micro - RP2040, Thing Plus -  
RP2040, MicroMod RP2040 Processor



# Raspberry Pi Pico Family



Raspberry Pi Pico



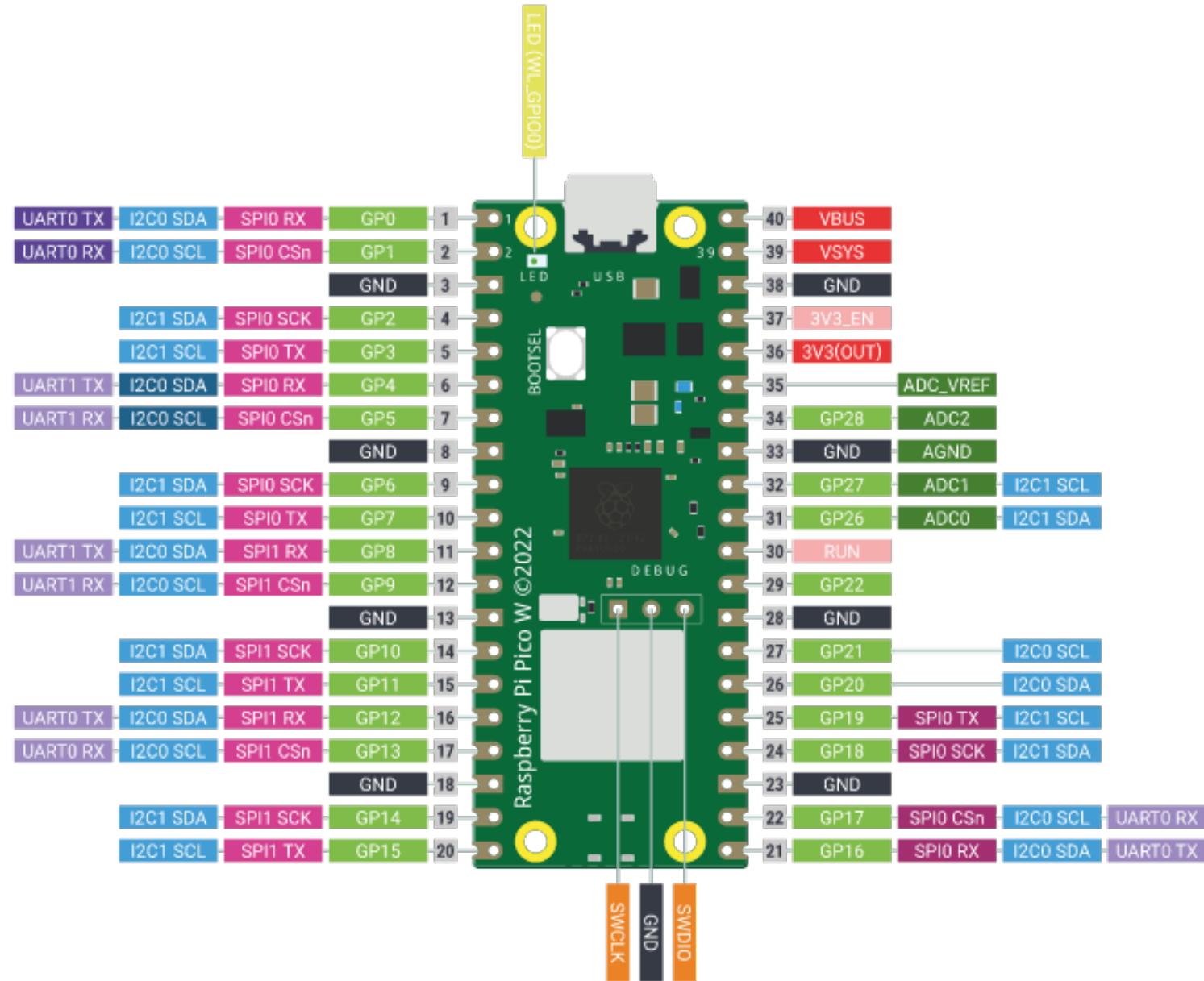
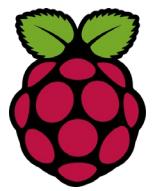
Raspberry Pi Pico H



Raspberry Pi Pico W



# Raspberry Pi Pico W Pinout



RP2040

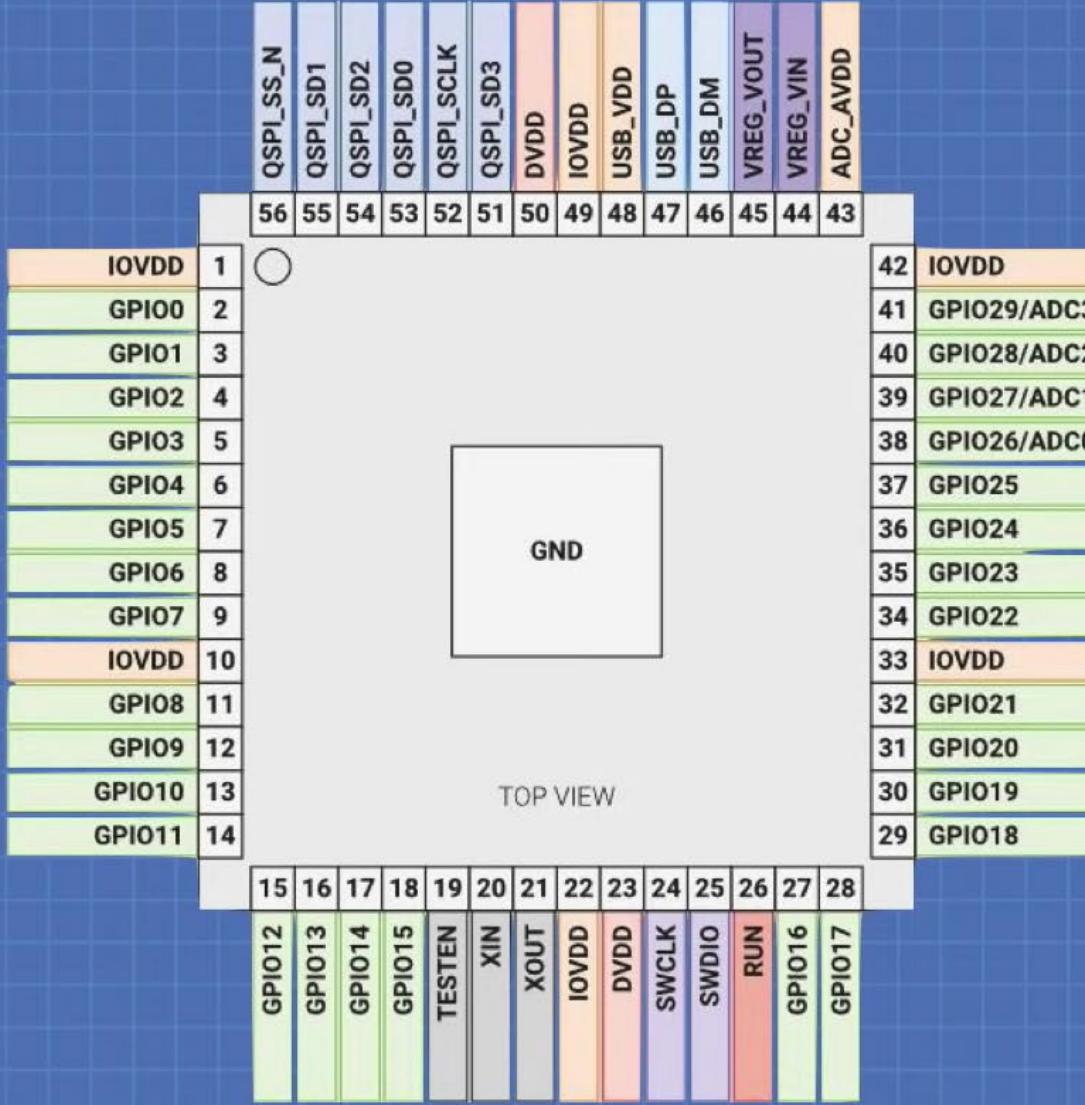
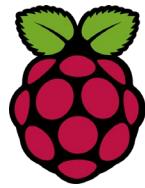
■	Power
■	Ground
■	UART / UART (default)
■	GPIO, PIO, and PWM
■	ADC
■	SPI / SPI (default)
■	I2C / I2C (default)
■	System Control
■	Debugging

Infineon 43439



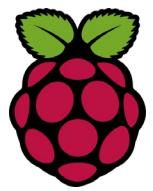


# Raspberry Pi Pico



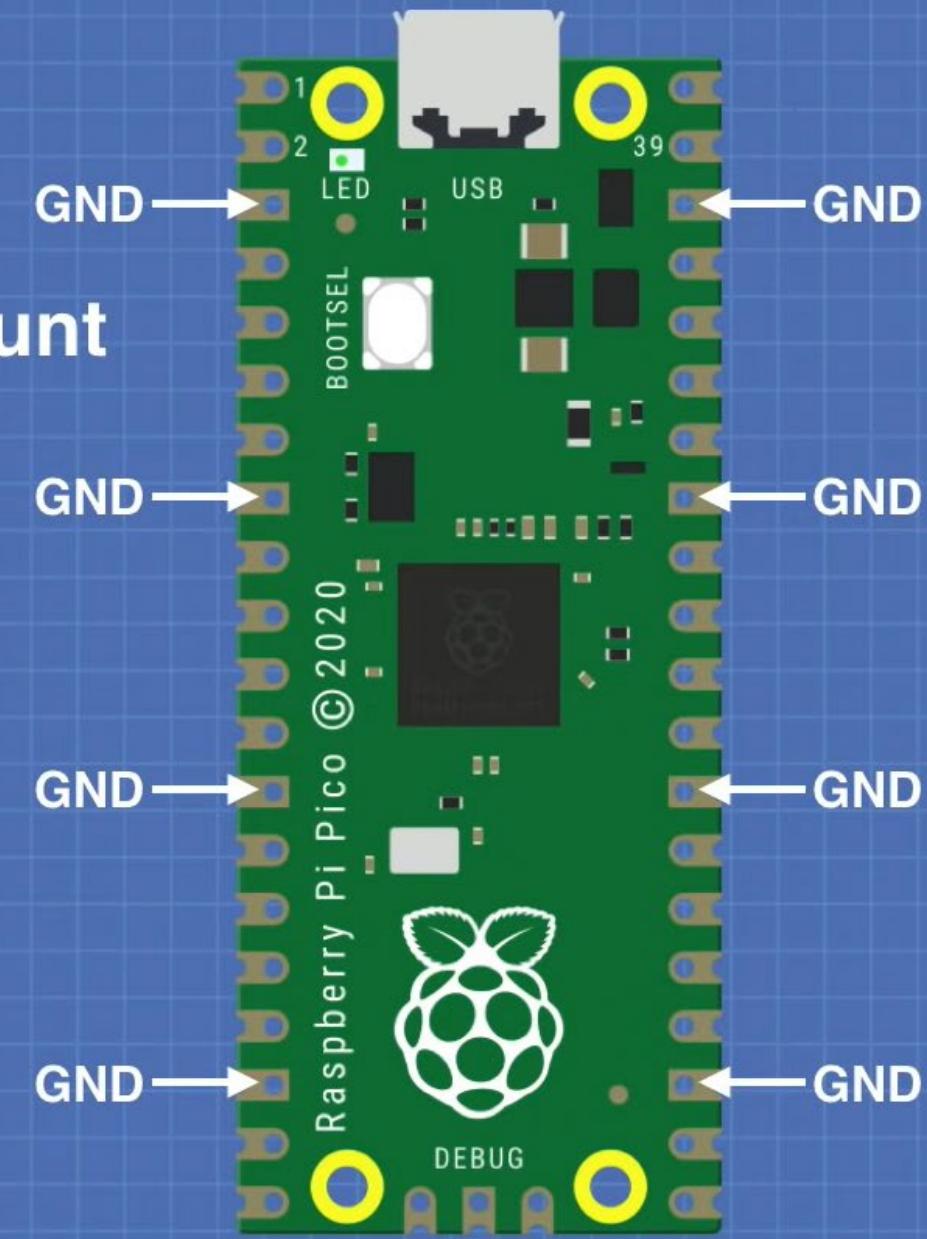
## RP2040

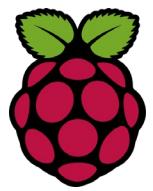
- 32-Bit Dual-Core ARM Cortex-M0+
- Clock speed 48MHz, boost to 133MHz
- 2MB onboard External Flash
- Onboard RTC
- Onboard Temperature Sensor



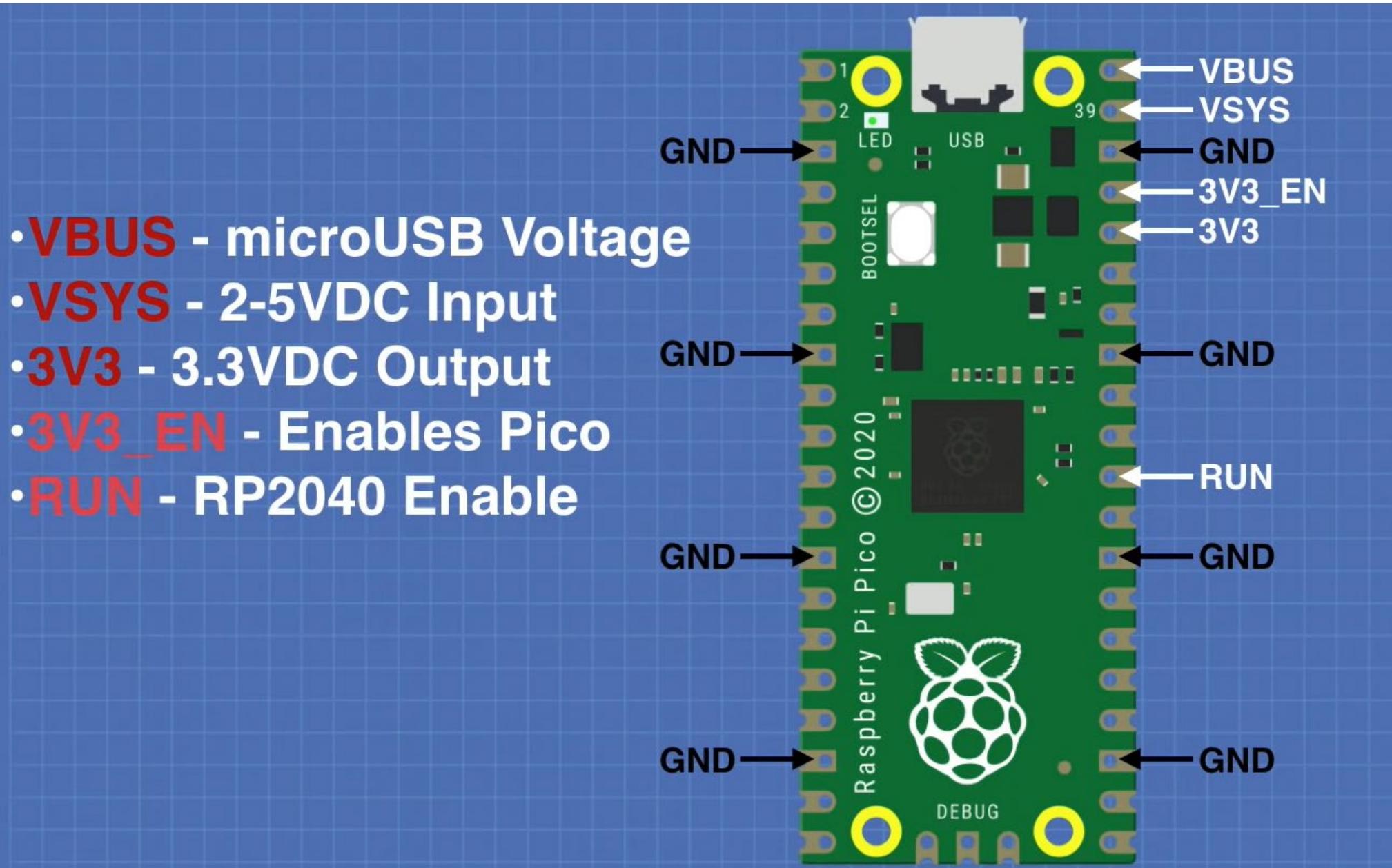
# Raspberry Pi Pico

- Use pins or surface-mount
- 8 Ground Points
- Evenly Spaced
- Square Pads



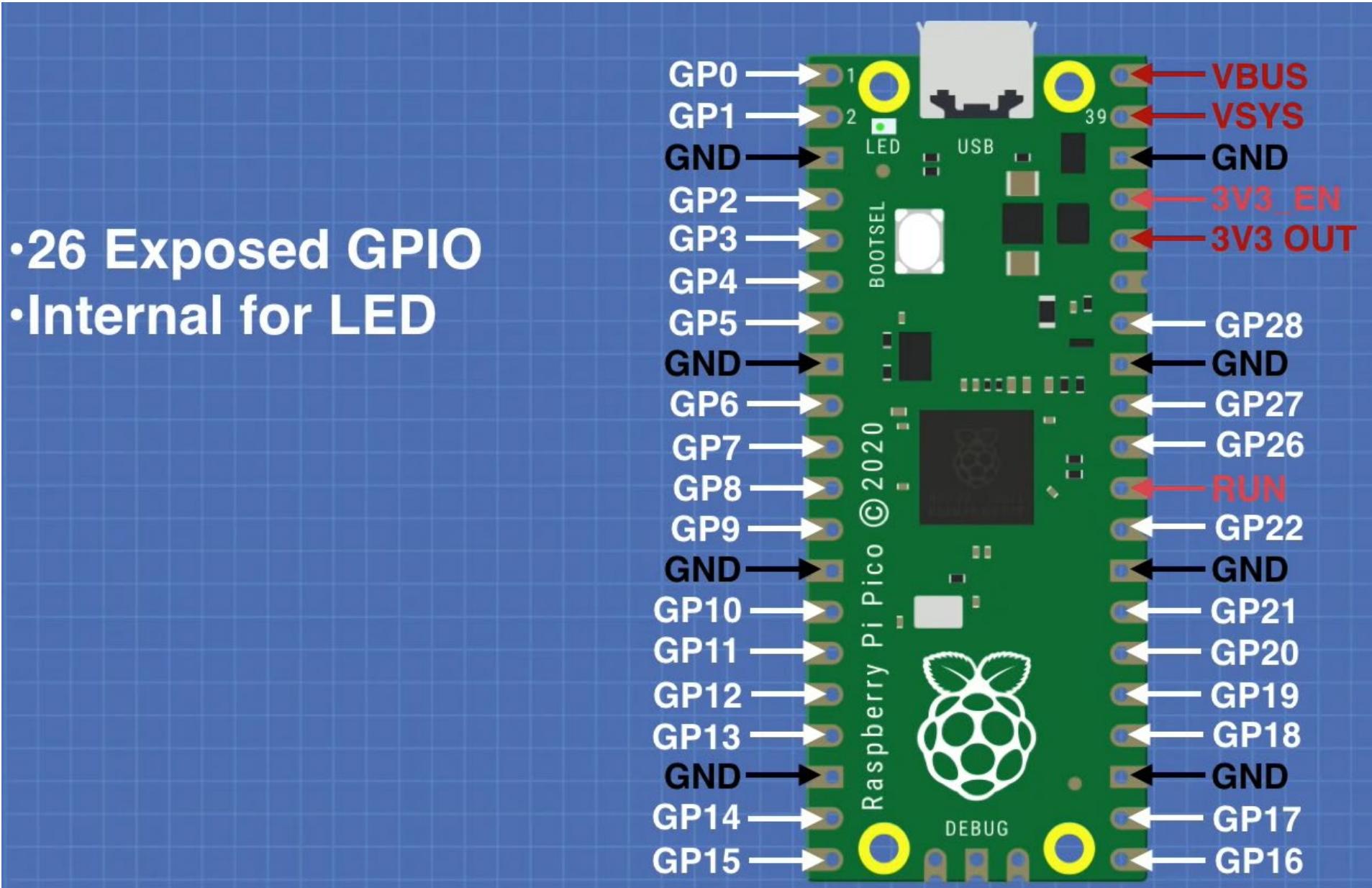
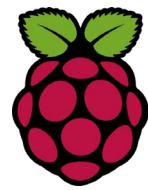


# Raspberry Pi Pico



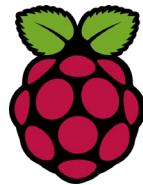


# Raspberry Pi Pico

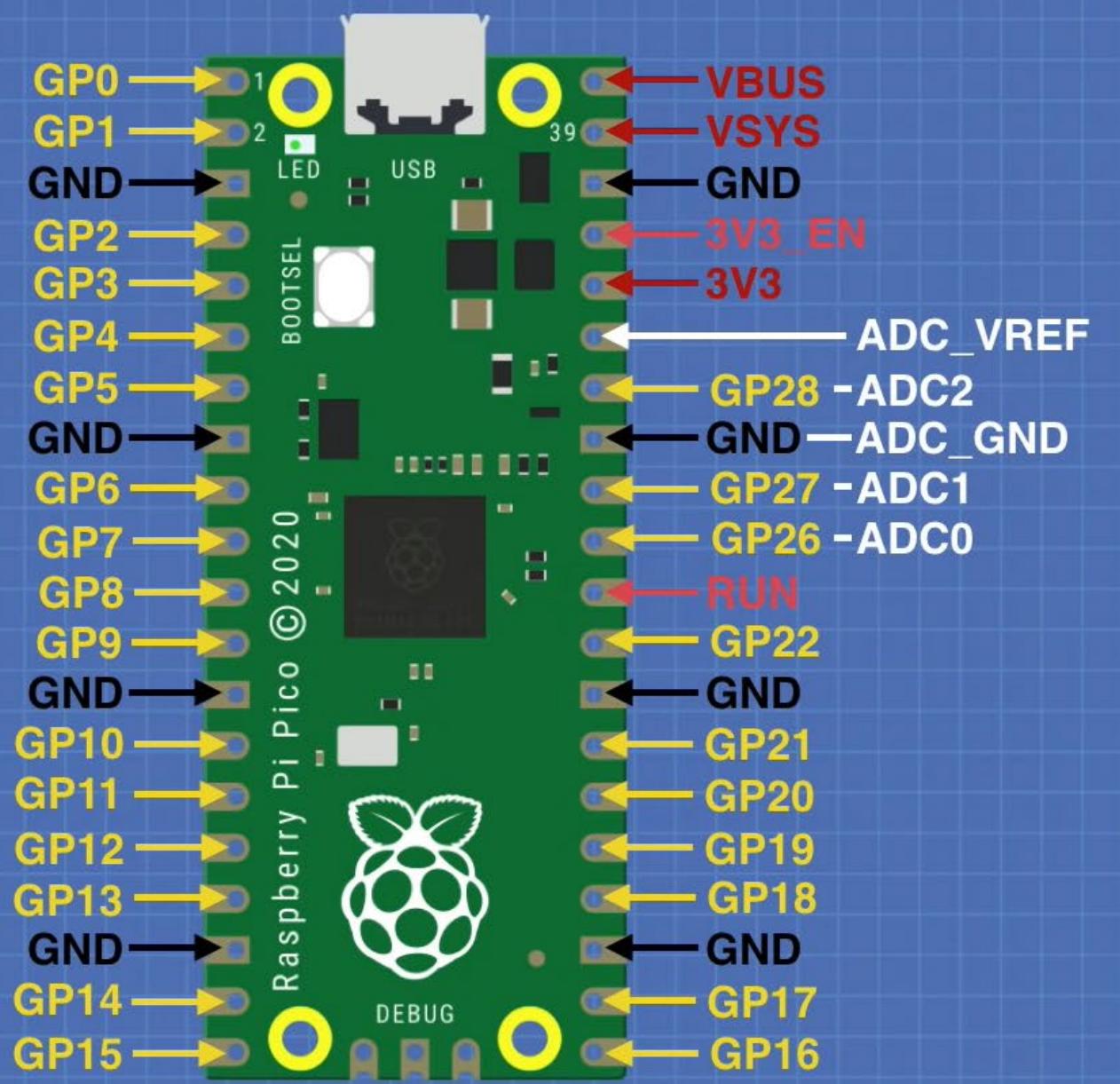


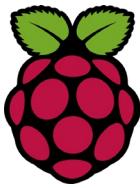


# Raspberry Pi Pico



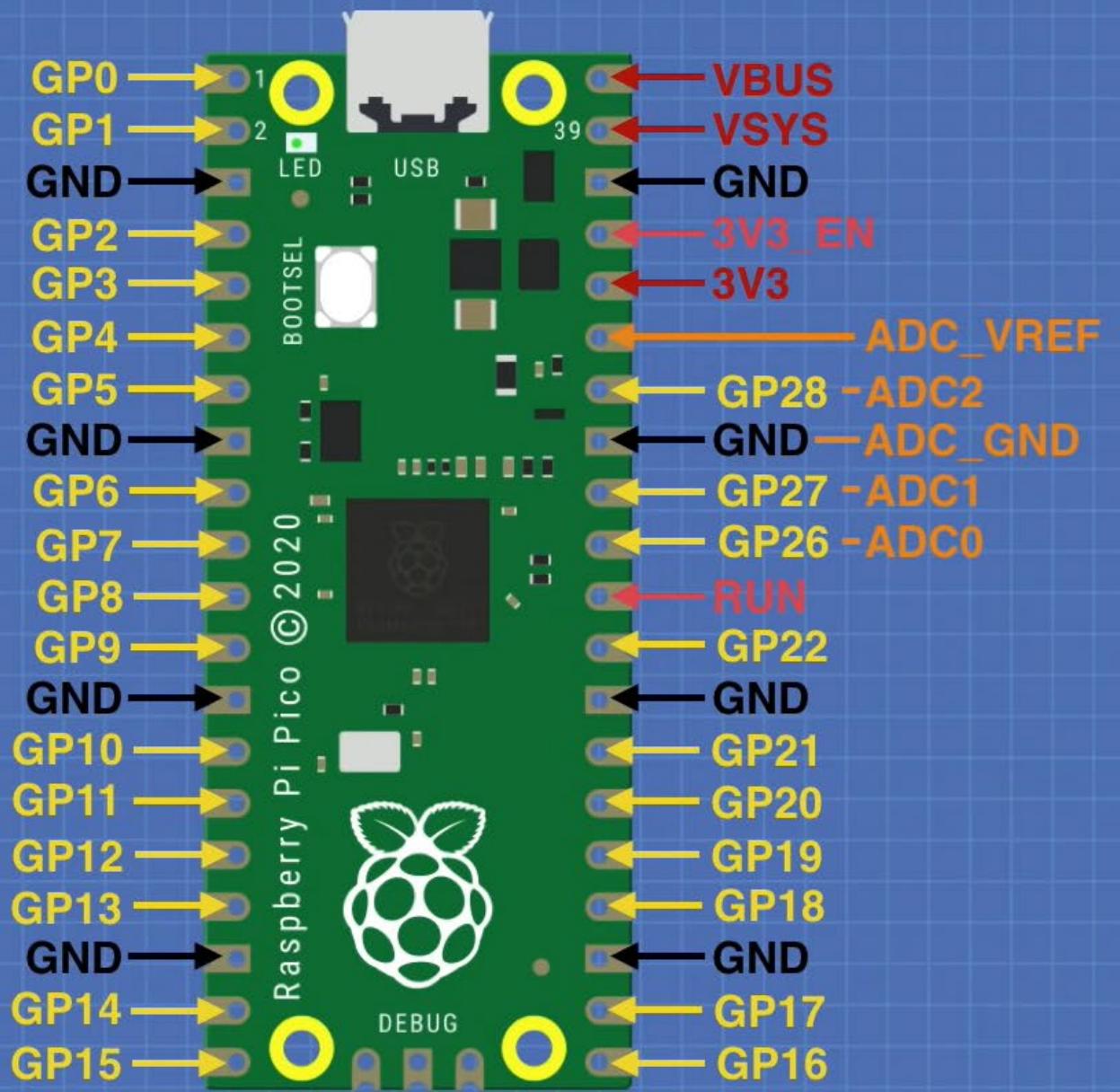
- 12-Bit ADC
  - 3 plus Internal ADC
  - Internal ADC for Temp
  - ADC\_VREF for external
  - ADC\_GND for external





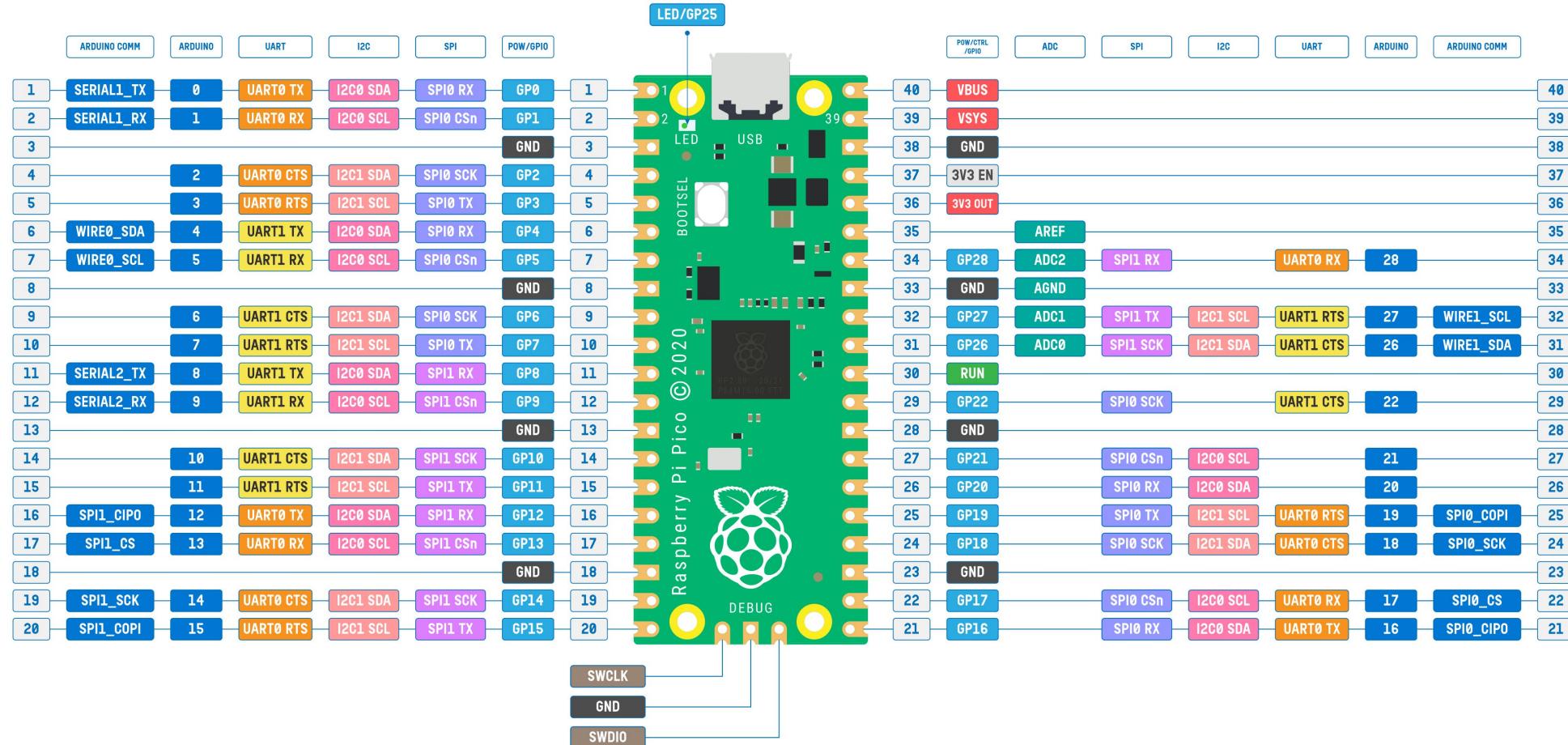
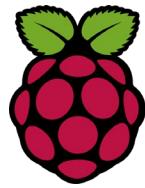
# Raspberry Pi Pico

- 2 x I2C Bus
- 2 x SPI Bus
- 2 x UART
- 16 x PWM Channels





# Raspberry Pi Pico – Full Pinout



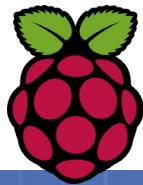
\*Raspberry Pi and the Raspberry Pi logo are trademarks of Raspberry Pi Ltd.

Raspberry Pi Pico vector image is originally designed by Raspberry Pi. Please visit [raspberrypi.com](https://www.raspberrypi.com) for more information.

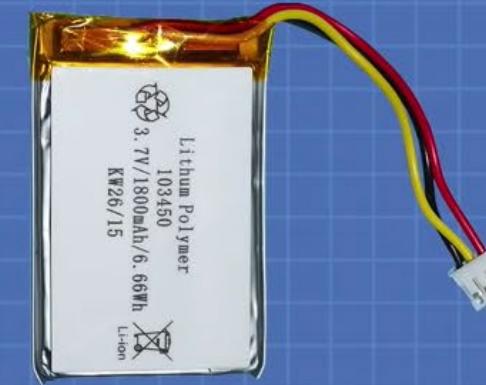
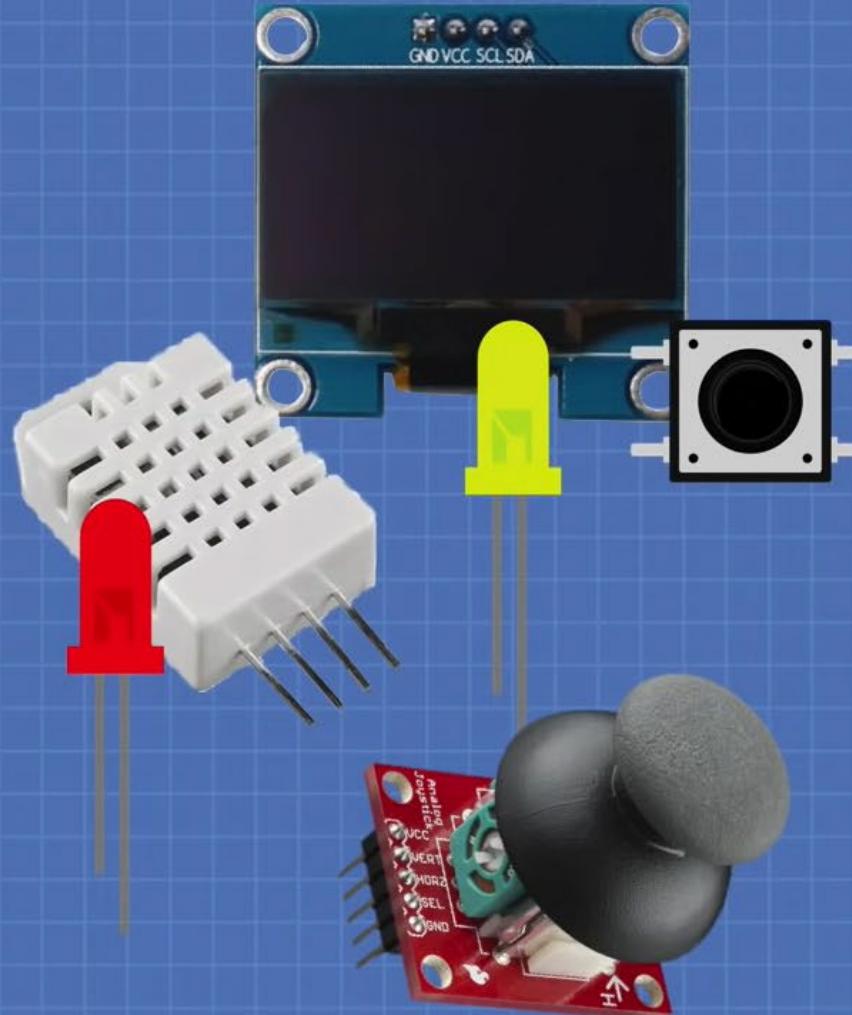
ARDUINO PINS	SWD Pins		
PHYSICAL PIN	POSITIVE SUPPLY	UART1 Pins	UART0 Pins
RESET/ENABLE	GROUND SUPPLY	I2C1 Pins	I2C0 Pins
GPIO PORT/PIN	ANALOG PIN	SPI1 Pins	SPI0 Pins

- **GP29/ADC3** is used to measure VSYS.
  - **GP25** is used by debug LED.
  - **GP24** is used for VBUS sense.
  - **GP23** is connected to SMPS Power Save pin.
  - All GPIO pins support PWM. There are total 16 PWM channels.
  - All GPIO pins support level and edge interrupts.
  - Arduino pins are as per **Arduino-Pico** core by *Earle F. Philhower, III* @earlephilhower
  - Arduino's default **Serial** is the USB-CDC of Pico.





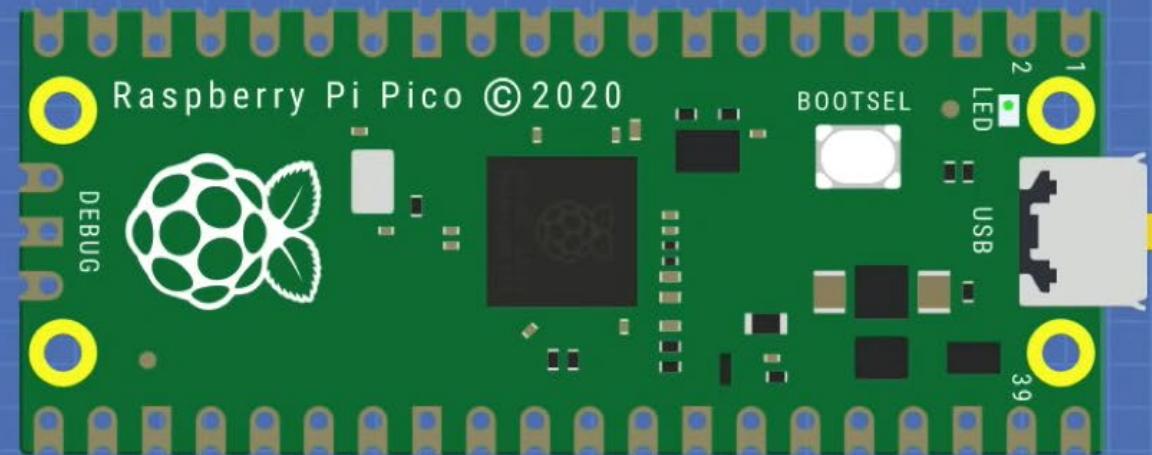
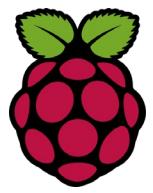
# Raspberry Pi Pico



- Can operate as normal MCU
- Power using microUSB
- Battery-powered, stand-alone
- Also operates as USB Device



# Raspberry Pi Pico

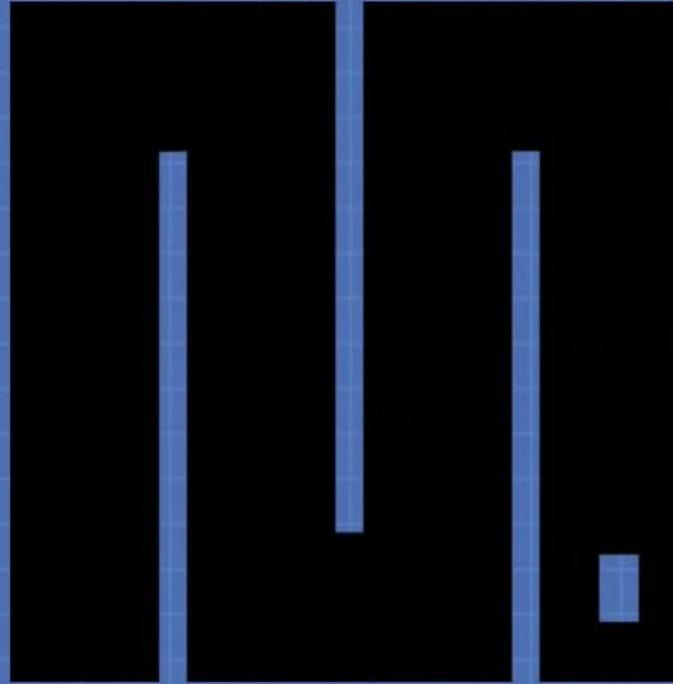
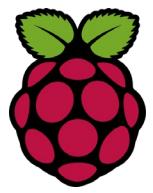


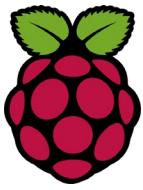
Can operate as a USB device



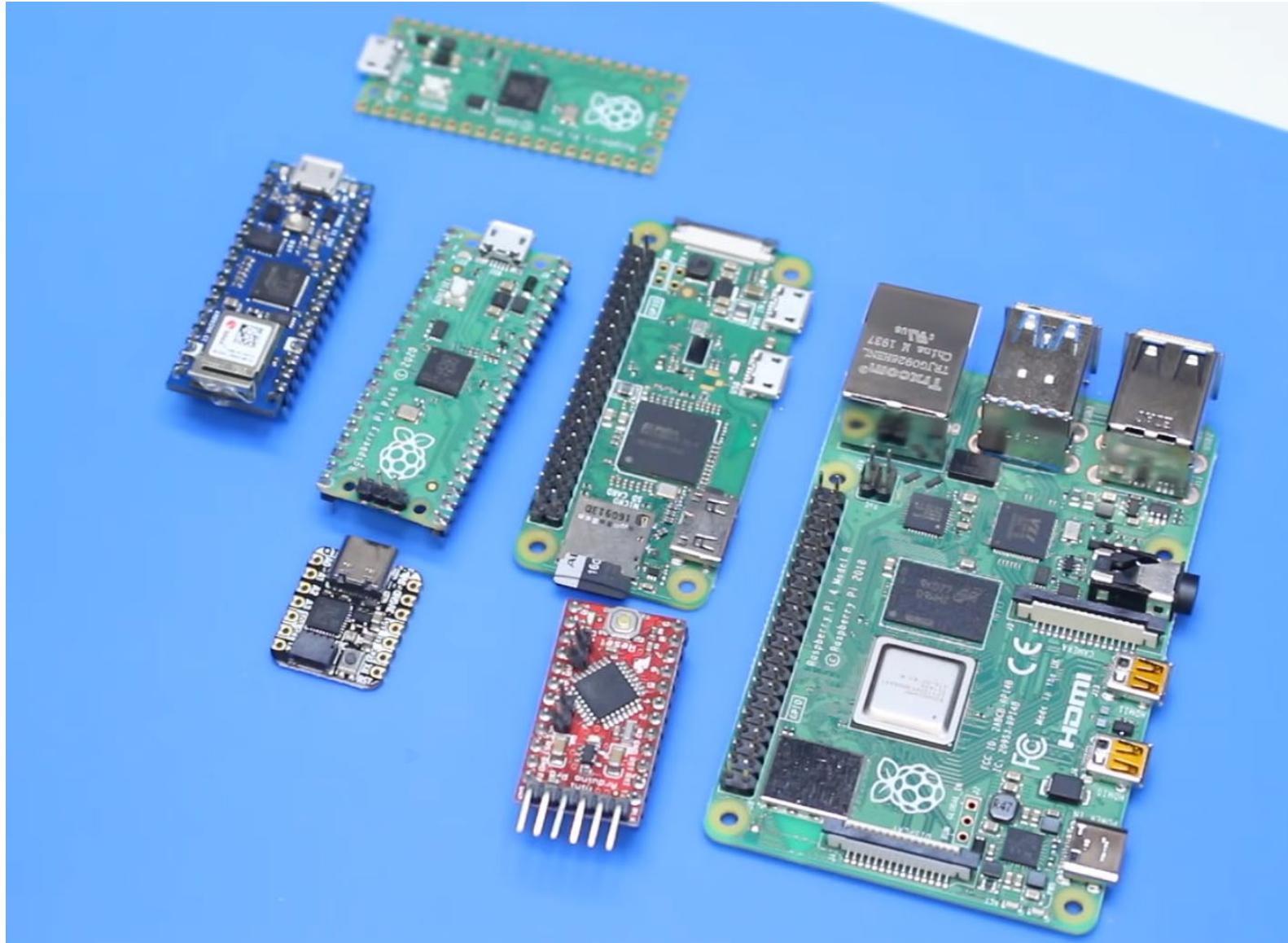


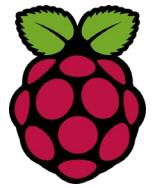
# languages used



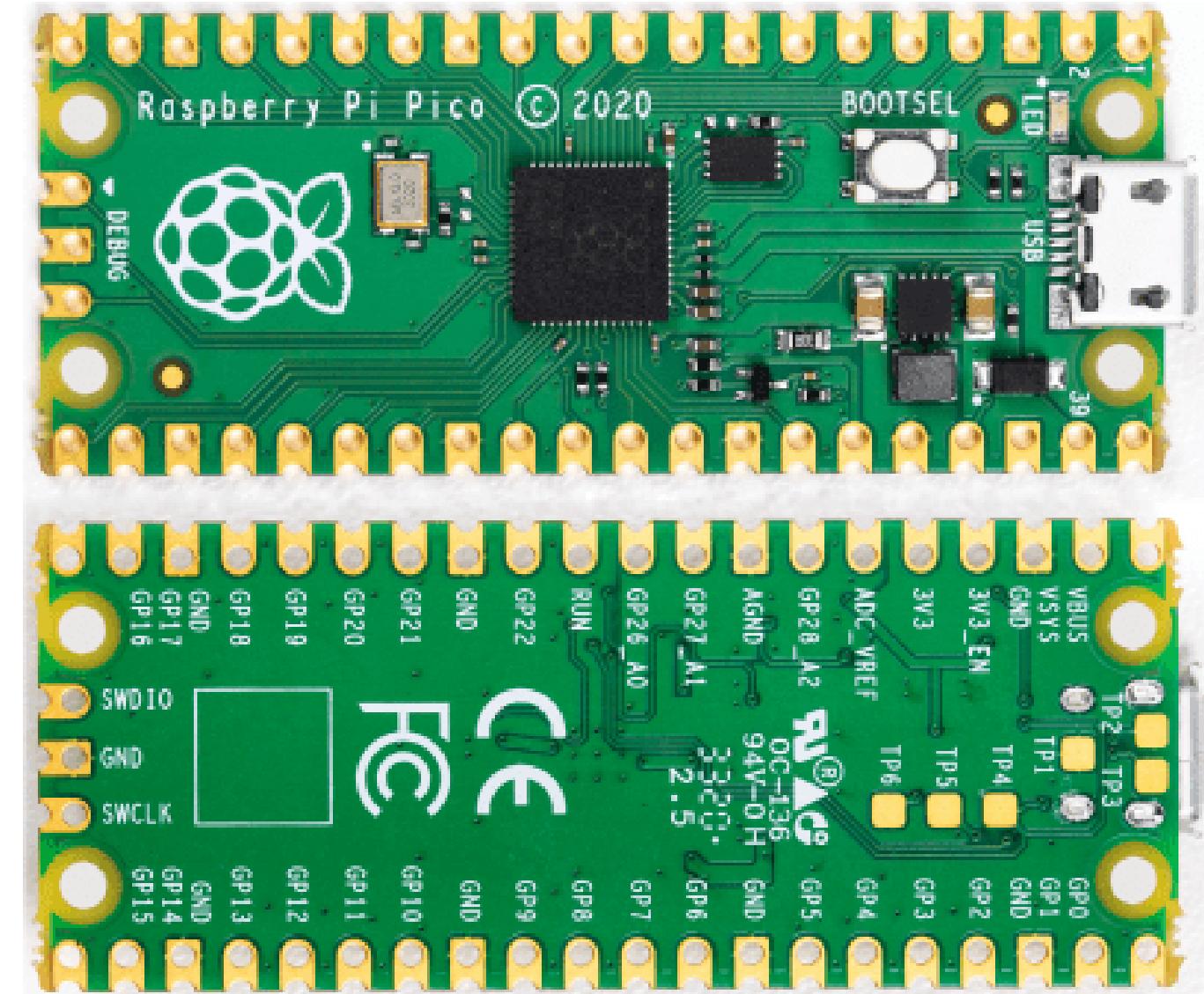


# Popular Raspberry Pi Family





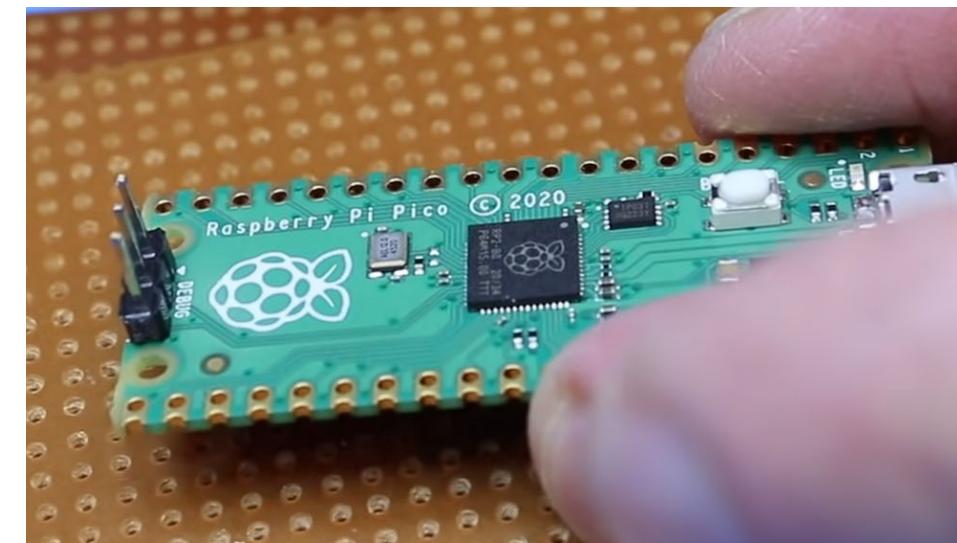
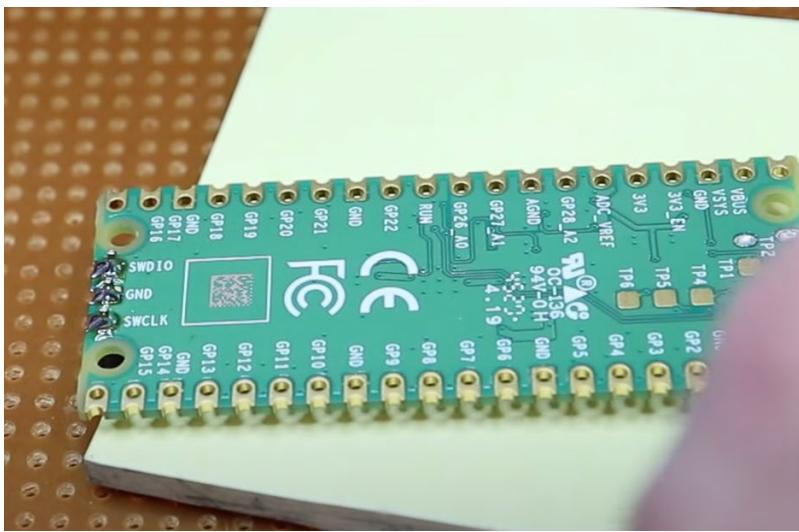
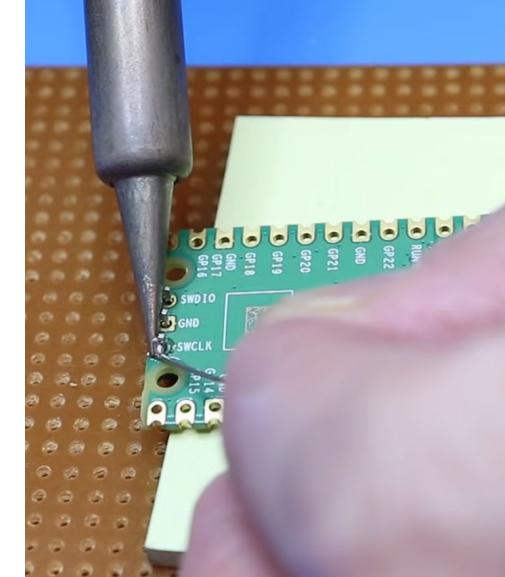
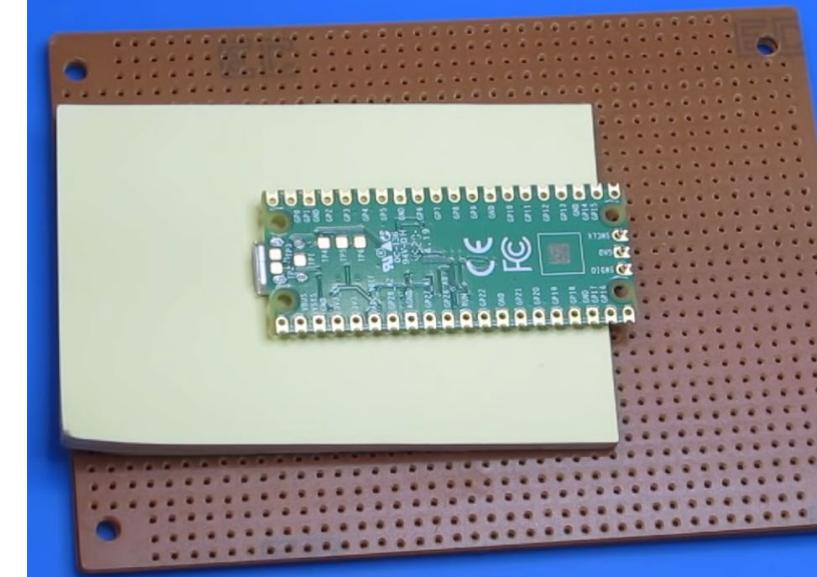
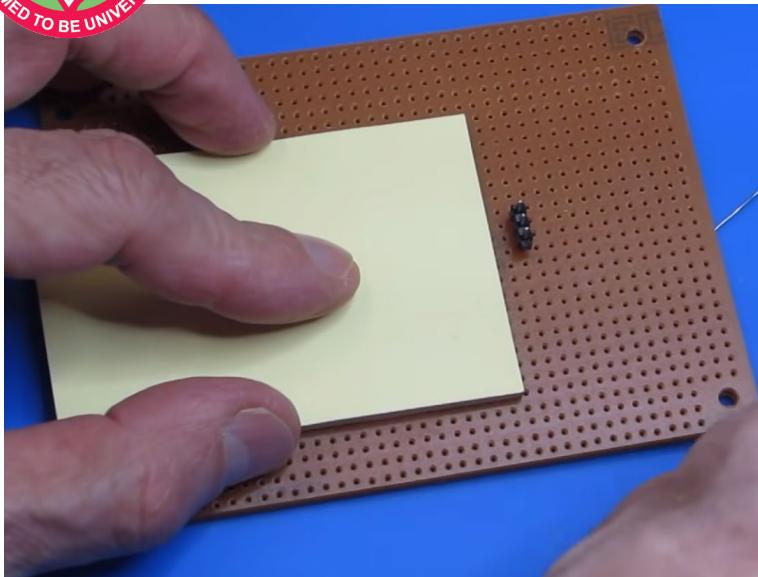
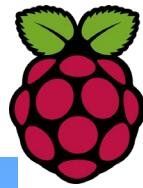
# Front and Back view



Front and Back view of the Raspberry Pi Pico

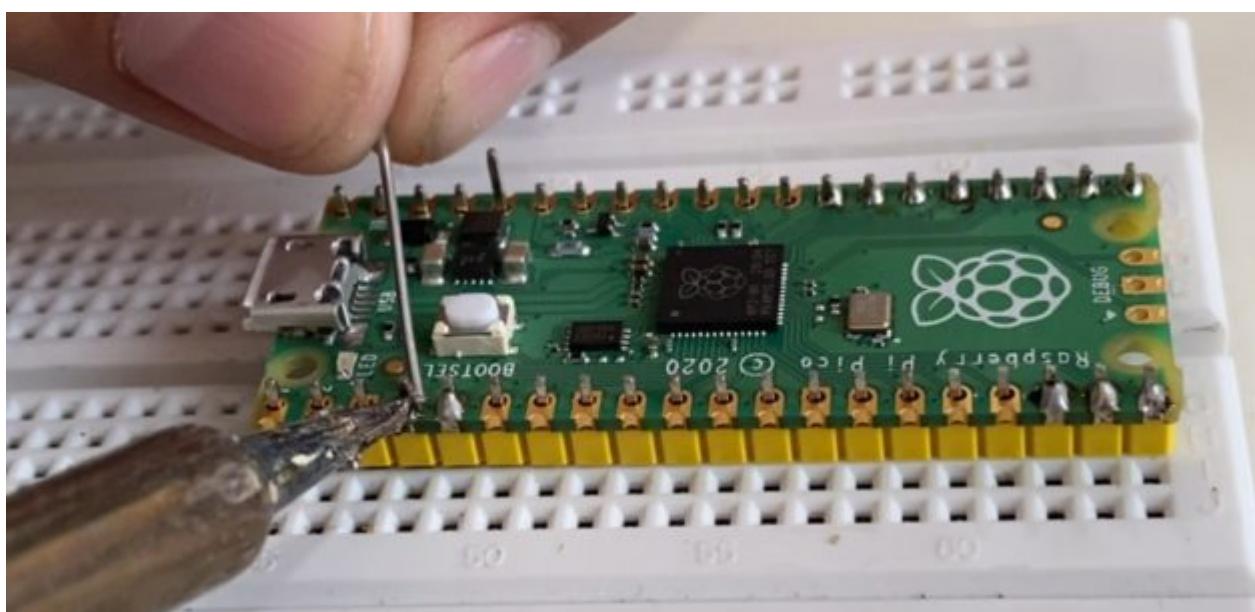
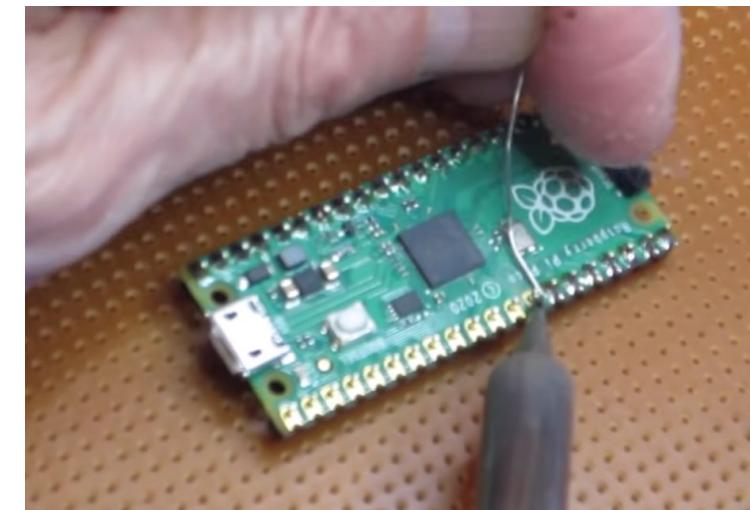
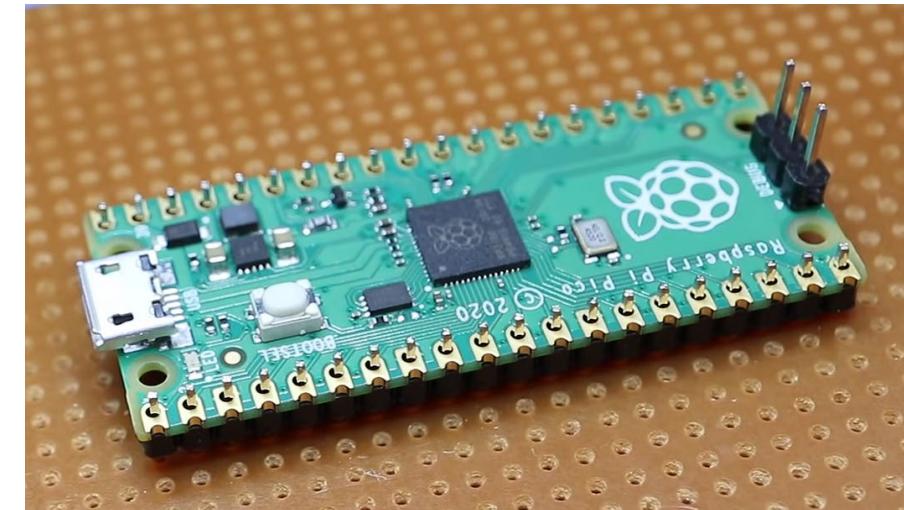
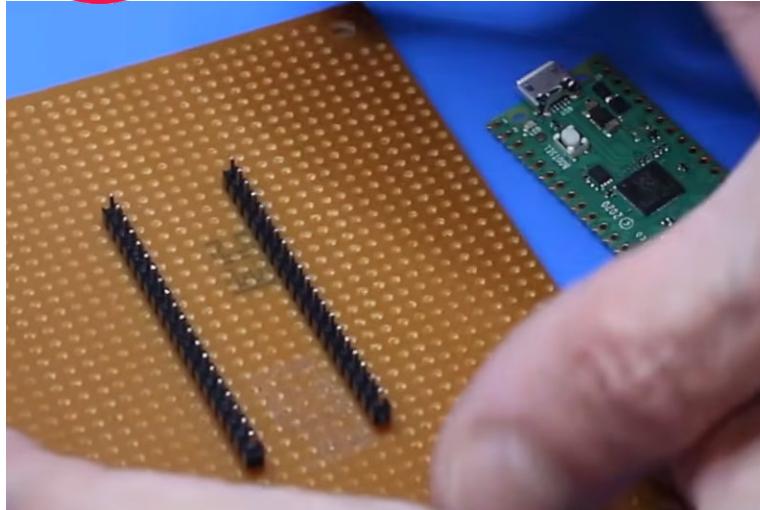
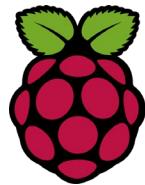


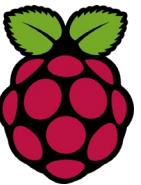
# Raspberry Pi Pico – H Preparation



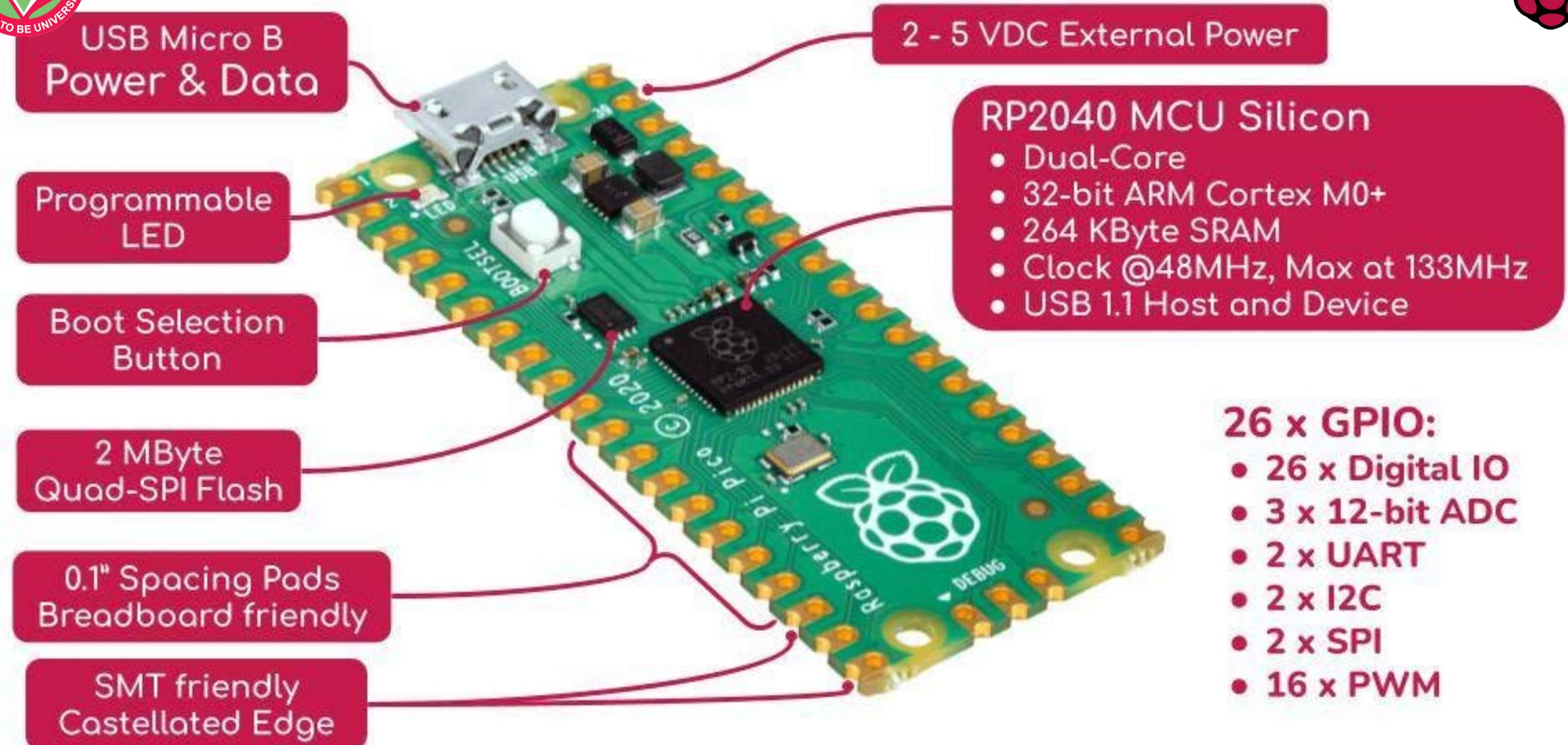


# Raspberry Pi Pico

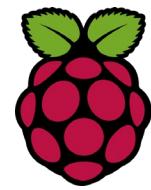




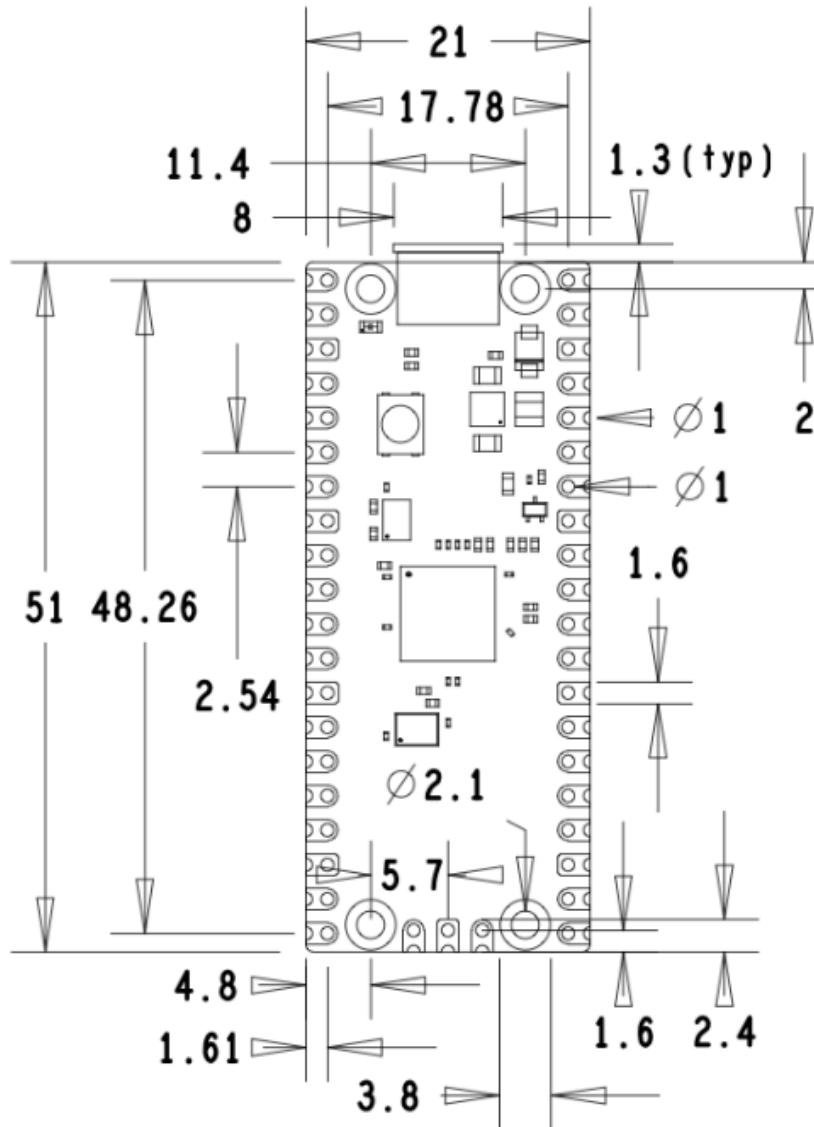
# Raspberry Pi Pico – In Short



Pi Foundation has released an RP2040 Microprocessor based development board, in the same form factor as an Arduino Nano.

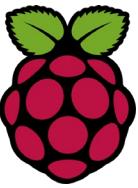


# Mechanical Specifications



- ✓ Single sided 51x21mm 1mm thick PCB
  - ✓ Usable as a surface mount module as well as being in Dual Inline Package (DIP) type format
  - ✓ 40 main user pins on a 2.54mm (0.1") pitch grid with 1mm holes and hence compatible with veroboard and breadboard.
  - ✓ 4 x 2.1mm (+/- 0.05mm) drilled mounting holes to provide for mechanical fixing.

## Mechanical specifications for the Raspberry Pi Pico



# Introduction to Online Simulator, WOKWI

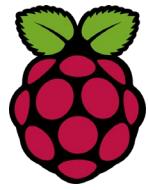
**Step 1:** Open <https://wokwi.com/>

**Step 2:** scroll down to Raspberry pi pico

The screenshot shows a browser window for 'Wokwi - Online Arduino and ESP'. The address bar contains 'wokwi.com'. Below the address bar is a toolbar with various icons. The main content area is titled 'Start from Scratch' and features six project cards arranged in two rows of three. Each card shows a circuit board image with a large white plus sign in the center. The projects are: Arduino Uno, Arduino Mega, ESP32 (top row); and Arduino Nano, Raspberry Pi Pico, MicroPython on ESP32 (bottom row). At the bottom of the page, there is a purple button labeled 'Franzininho Project | MicroPython on Pi Pico | + MORE OPTIONS'.



# WOKWI's First Look



## Step3: Click on Raspberry pi pico

New Raspberry Pi Pico Project - wokwi.com/projects/new/pi-pico

Bookmarks (81) Facebook Inbox (584) - swain.... UPSC NDA & NA (II...) RRB Senior Section... opsc Forums / Projects /... Electricity Generation... ScholarOne Manus...

WOKWI SAVE SHARE Docs SIGN IN

sketch.ino diagram.json Library Manager

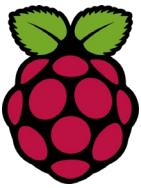
```
1 void setup() {
2     // put your setup code here, to run once:
3     Serial1.begin(115200);
4     Serial1.println("Hello, Raspberry Pi Pico!");
5 }
6
7 void loop() {
8     // put your main code here, to run repeatedly:
9     delay(1); // this speeds up the simulation
10 }
11
```

Simulation

Raspberry Pi Pico

Type here to search

21:24 12-09-2022 4



# WOKWI's Setting

**Step4:** Sign in using your Gmail account

**Step5:** Delete sketch.ino window

The screenshot shows a browser window with the WOKWI platform. The address bar indicates the URL is [wokwi.com/projects/new/pi-pico](https://wokwi.com/projects/new/pi-pico). The main area displays a code editor for a file named `sketch.ino`. The code is as follows:

```
1 void setup() {
2     // put your setup code here, to run
3     // once the sketch starts
4     Serial1.begin(115200);
5     Serial1.println("Hello, Raspberry
6 
7 void loop() {
8     // put your main code here, to run
9     delay(1); // this speeds up the si
10 }
11 }
```

A context menu is open over the code, listing the following options:

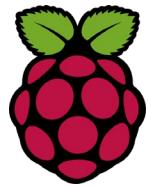
- Format code
- Rename
- Delete
- New file...
- Upload file(s)...

To the right of the code editor is a "Simulation" panel featuring a digital model of a Raspberry Pi Pico board. The board is green with a central Broadcom chip, two yellow LEDs, and various component pads. Below the simulation are three control buttons: a green play button, a purple plus button, and a grey ellipsis button.

At the bottom of the screen, a Windows taskbar is visible with icons for File Explorer, Task View, Edge, Google Chrome, Microsoft Edge, and Microsoft Word. The system tray shows the date and time as 22:44 13-09-2022, along with battery and connectivity status.



# WOKWI's new file creation



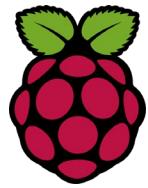
**Step6:** Create a new file with name “main.py”

The screenshot shows a browser window with the URL [wokwi.com/projects/new/pi-pico](https://wokwi.com/projects/new/pi-pico). The main area displays a JSON configuration file for a Raspberry Pi Pico project. A context menu is open over the first line of the code, showing options: Rename, Delete, New file..., and Upload file(s)... . To the right of the code editor is a "Simulation" panel featuring a green Raspberry Pi Pico board with its pins labeled. Below the simulation panel is a Windows taskbar with various icons and system status.

```
1 {  
2   "version": 1,  
3   "author": "Biswaranjan",  
4   "editor": "wokwi",  
5   "parts": [ { "type": "pico", "top": 0, "left": 0, "width": 1, "height": 1, "pins": [ "pin0", "pin1", "pin2", "pin3", "pin4", "pin5", "pin6", "pin7", "pin8", "pin9", "pin10", "pin11", "pin12", "pin13", "pin14", "pin15", "pin16", "pin17", "pin18", "pin19", "pin20", "pin21", "pin22", "pin23", "pin24", "pin25", "pin26", "pin27", "pin28", "pin29", "pin30", "pin31", "pin32", "pin33", "pin34", "pin35", "pin36", "pin37", "pin38", "pin39", "pin40" ], "connections": [ [ "pin0": "pico:GND", "pin1": "pico:VDD", "pin2": "pico:GND", "pin3": "pico:VDD", "pin4": "pico:GND", "pin5": "pico:VDD", "pin6": "pico:GND", "pin7": "pico:VDD", "pin8": "pico:GND", "pin9": "pico:VDD", "pin10": "pico:GND", "pin11": "pico:VDD", "pin12": "pico:GND", "pin13": "pico:VDD", "pin14": "pico:GND", "pin15": "pico:VDD", "pin16": "pico:GND", "pin17": "pico:VDD", "pin18": "pico:GND", "pin19": "pico:VDD", "pin20": "pico:GND", "pin21": "pico:VDD", "pin22": "pico:GND", "pin23": "pico:VDD", "pin24": "pico:GND", "pin25": "pico:VDD", "pin26": "pico:GND", "pin27": "pico:VDD", "pin28": "pico:GND", "pin29": "pico:VDD", "pin30": "pico:GND", "pin31": "pico:VDD", "pin32": "pico:GND", "pin33": "pico:VDD", "pin34": "pico:GND", "pin35": "pico:VDD", "pin36": "pico:GND", "pin37": "pico:VDD", "pin38": "pico:GND", "pin39": "pico:VDD", "pin40": "pico:GND" ] ] }  
3 }  
4 }  
5 }  
6 }  
7 }
```



# WOKWI's Toggling inbuilt LED



## Step-7: Toggling inbuilt LED

4 WhatsApp    x | The Electronics    x | W New Raspberry Pi Pico Project - v x

← → ⌛ ⌂ 🔒 wokwi.com/projects/new/pi-pico

Bookmarks (81) Facebook (81) Inbox (584) - swain.... UPSC NDA & NA (II...) RRB Senior Section... opsc Forums / Projects /... Electricity Generation... ScholarOne Manus... »

WOKWI SAVE SHARE Docs B

diagram.json • main.py • Library Manager PIO

```
1 from machine import Pin
2 import utime
3
4 ledPin = Pin(25,Pin.OUT)
5
6 while True:
7     ledPin.value(1)
8     print("LED ON")
9     utime.sleep(1)
10    ledPin.value(0)
11    print("LED OFF")
12    utime.sleep(1)
13
```

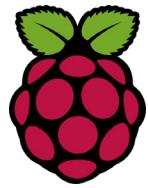
Simulation 00:08.331 100%

Type here to search

22:54 13-09-2022



# WOKWI's Toggling External LED



## Step-8: Toggling External LED

4 WhatsApp    x | The Electronics    x | W New Raspberry Pi Pico Project - v    +

← → ⌛ ⌂ wokwi.com/projects/new/pi-pico

Bookmarks Facebook Inbox (584) - swain.... UPSC NDA & NA (II...) RRB Senior Section... opsc Forums / Projects / ... Electricity Generatin... ScholarOne Manus... »

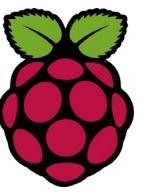
WOKWI SAVE SHARE Docs B

diagram.json • main.py • Library Manager PIO

```
1 from machine import Pin
2 import utime
3
4 ledPin = Pin(15,Pin.OUT)
5
6 while True:
7     ledPin.value(1)
8     print("LED ON")
9     utime.sleep(1)
10    ledPin.value(0)
11    print("LED OFF")
12    utime.sleep(1)
13
```

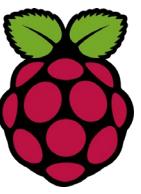
Simulation 00:16.362 102%

The screenshot shows the WOKWI web-based development environment. On the left, there's a code editor with Python code for a Raspberry Pi Pico project. The code initializes pin 15 as an output and enters a loop where it alternates the LED state between high and low every second. On the right, a simulation window displays a breadboard-style circuit. A red LED is connected in series with a 220 ohm resistor, which is then connected to the Raspberry Pi Pico. The Pico's pin 15 is connected to the positive terminal of the LED. The negative terminal of the LED is connected to ground. The simulation shows the LED glowing when the digital output is high and dimming when it is low. The top of the screen shows browser tabs and a toolbar, and the bottom shows a Windows taskbar with various icons.



# How Students will submit the Physical Prototype?

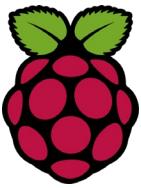
**COST of the Project???**



# How Students will submit the Physical Prototype?

**Mid-Sem : Mini Project (without IOT)**

**End-Sem: Major Project (With IOT)**



# Requirements

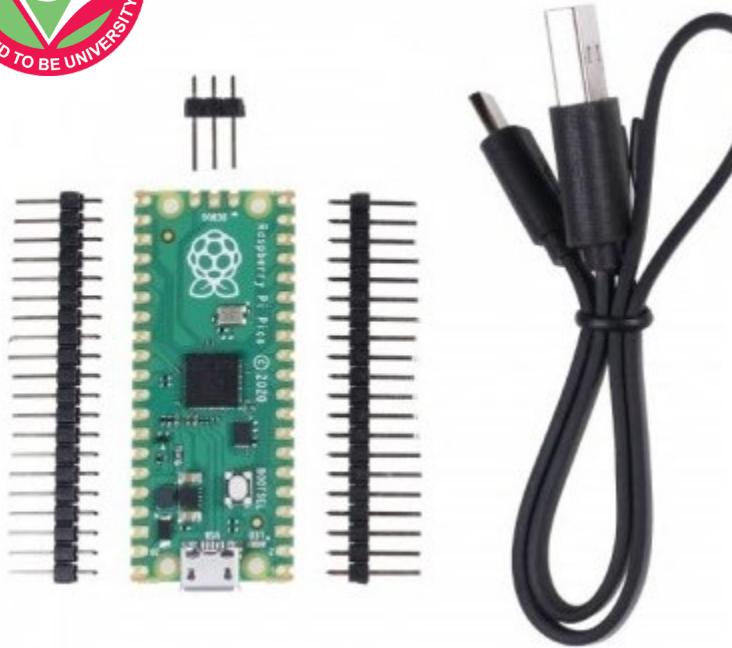
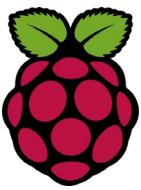
## Hardware

- A Raspberry Pi Pico with soldered headers
- A computer that can run the Thonny IDE and program a Raspberry Pi Pico
- A micro USB cable
- A selection of electronics components, such as a button, an LED with appropriate resistor, and a potentiometer (optional)
- A breadboard and M-M jumper leads for connecting additional components (optional)
- An external 5V micro USB power source (optional)

## ➤ Software

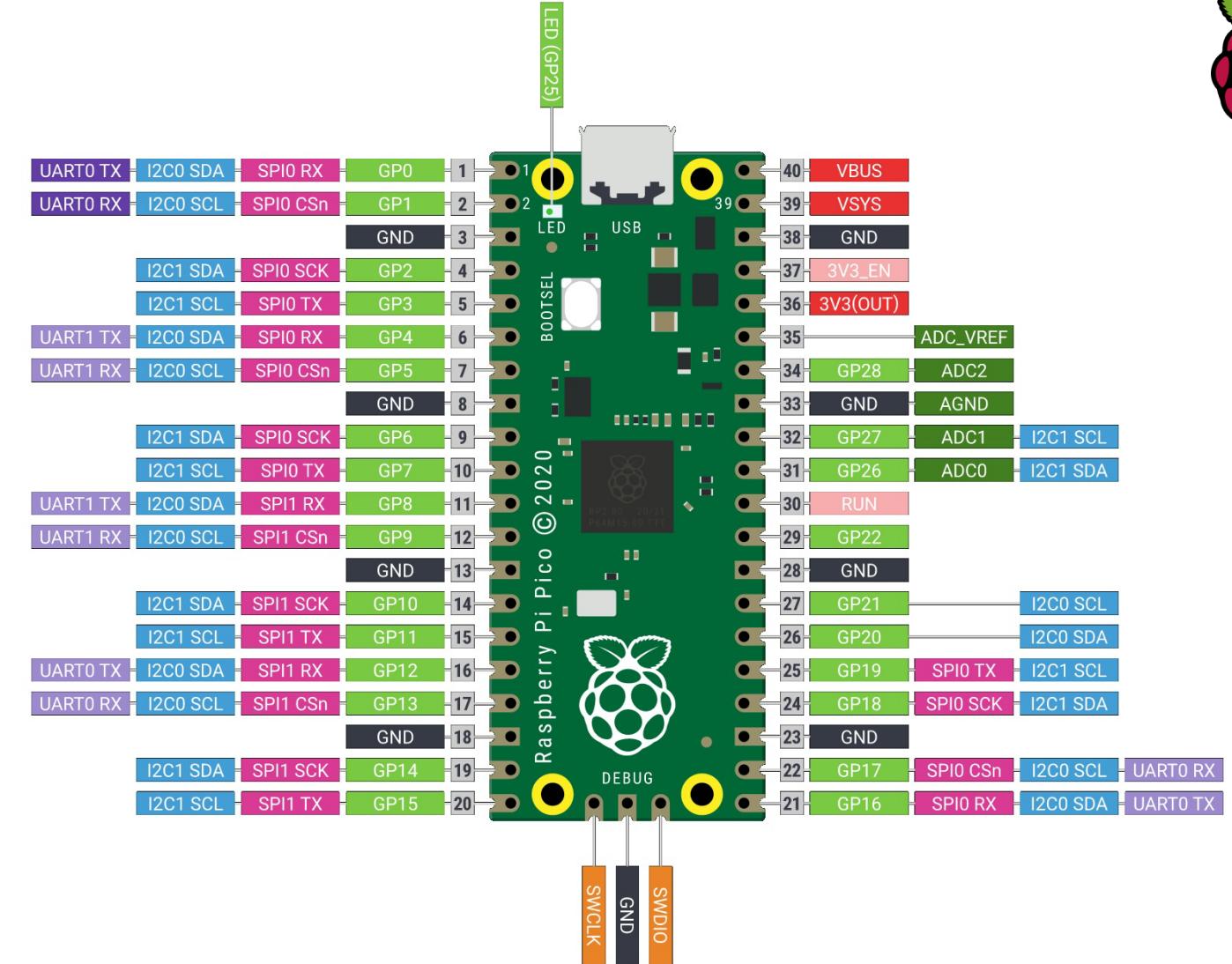
The next class will guide you through the installation of:

- MicroPython firmware for Raspberry Pi Pico
- The Thonny Python IDE

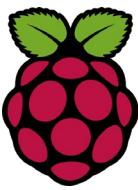


## Package Includes:

- 1 x Raspberry Pi Pico
- 1 x Micro-USB cable
- 2 x 20 Pin Header
- 1 x 3 Pin Header

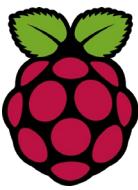


■ Power ■ Ground ■ UART / UART (default) ■ GPIO, PIO, and PWM ■ ADC ■ SPI ■ I2C ■ System Control ■ Debugging



# What we need to do for NEXT class

- How to load the MicroPython firmware onto a Raspberry Pi Pico?
- How to program a Raspberry Pi Pico using MicroPython?
- How to connect additional components to a Raspberry Pi Pico and write MicroPython programs to interact with them?



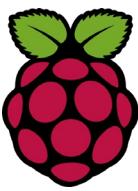
# QUIZ TIME???????

**1) How much memory does raspberry pi Pico have?**

- a) 12 MB
- b) 512 MB
- c) 1 MB
- d) 2 MB

**2) The clock speed of raspberry pi Pico is around \_\_\_\_\_**

- a) 125 MHz
- b) 512 MHz
- c) 256 KHz
- d) 1 GHz



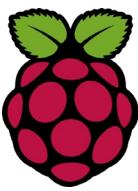
## QUIZ TIME

3) The input voltage for raspberry pi Pico is around \_\_\_\_\_

- a) 5 V
- b) 12 V
- c) 1 V
- d) 3.3 V

4) How many pins does raspberry pi Pico H contain?

- a) 32
- b) 40
- c) 20
- d) 57



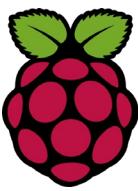
## QUIZ TIME

**5) In which year the raspberry pi Pico is launched?**

- a) January 2021
- b) February 2020
- c) January 2018
- d) January 2020

**6) The I2C pin on the raspberry pi board has \_\_\_\_\_ connections**

- a) one
- b) two
- c) three
- d) four



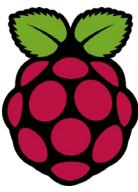
## QUIZ TIME

**7) How much RAM does raspberry pi Pico have?**

- a) 512 KB
- b) 128 KB
- c) 1 MB
- d) 256 KB

**8) Which SOC is used in raspberry pi Pico?**

- a) silicon RP2040
- b) Broadcom BCM2711
- c) Broadcom BCM2835
- d) silicon RP2004



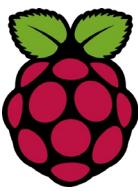
## QUIZ TIME

### 9) What are the advantages of Raspberry Pi Pico?

- a) Consumes less Power
- b) low cost
- c) Both a and b
- d) None of these

### 10) How many exposed GPIO pins does Raspberry Pi Pico?

- a) 25
- b) 26
- c) 27
- d) 29



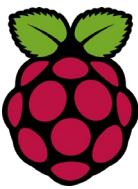
## QUIZ TIME

**11) Which cable is used to connect the PC to the Raspberry Pi Pico?**

- a) USB type A to A
- b) USB type A to B
- c) USB type A to Micro B
- d) USB type A to C

**12) What are the capabilities of raspberry pi?**

- a) Browsing the internet
- b) Making Spreadsheets
- c) Word Pressing
- d) All of the above



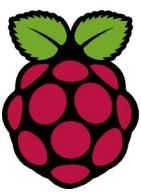
## QUIZ TIME

13) The software pulse width modulation in raspberry pi pico is available on \_\_\_\_\_ pins

- a) GPIO12
- b) GPIO18
- c) GPIO19
- d) All GPIO pins

14) What is the form factor of Raspberry Pi Pico?

- a) 51 x 21 mm
- b) 61 x 31 mm
- c) 61 x 21 mm
- d) 51 x 31 mm



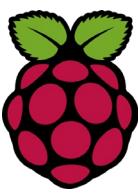
## QUIZ TIME

**15) What is the standard form of SPI pin?**

- a) Serial Parallel Input
- b) Serial Peripheral Interface
- c) Serial Parallel Interface
- d) None of the above

**16) How many PWM pins does Raspberry Pi Pico have?**

- a) 16
- b) 17
- c) 25
- d) 26



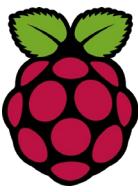
## QUIZ TIME

17) IOT stands for .....

- a) internet of telegram
- b) internet of things
- c) intelligent of things
- d) intercommunication of things

18) An equation of internet of things .....

- a) physical object + controller sensor and actuator + internet
- b) controller sensor and actuator + internet
- c) physical object + internet
- d) Physical object + controller + internet



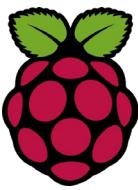
## QUIZ TIME

**19) A ..... tends to convert physical attribute to an electrical signal.**

- a) actuator
- b) compiler
- c) sensor
- d) motors

**20) A ..... tends to convert electrical signal to physical action .**

- a) actuator
- b) compiler
- c) sensor
- d) motors



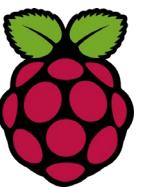
## QUIZ TIME

**21) choose correct principle of IOT.**

- a) focus on the value
- b) focus on the machine
- c) build a strong machine
- d) None of these

**22) ..... involves delivering different types of services over the Internet.**

- a) physical computing
- b) chemical computing
- c) mechanism
- d) cloud computing



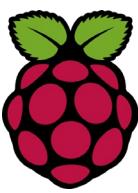
## QUIZ TIME

**23) ....... helps in collaborate in IOT development.**

- a) physical computing
- b) chemical computing
- c) mechanism
- d) cloud computing

**24) IOT and cloud computing has ....... relationship.**

- a) physically
- b) graphically
- c) complementary
- d) coding



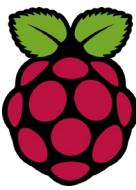
## QUIZ TIME

**25) .... uses certain protocols to aid sensors in connecting with real time machine to machine network.**

- a) real time analytics
- b) data collection
- c) device integration
- d) real time collection

**26) .....software supporting integration binds all system devices to create body of iot system.**

- a) real time analytics
- b) data collection
- c) device integration
- d) real time collection



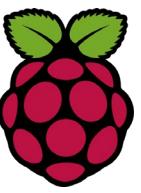
## QUIZ TIME

**27) The application data or input from various devices and convert it into viable actions are clear patterns human analysis is called .....**

- a) real time analytics
- b) data collection
- c) device integration
- d) real time collection

**28) A ..... is an established set of rules that determines how data is transmitted between different device in the same network.**

- a) network connection
- b) TCP IP protocol
- c) network protocol
- d) TCP protocol



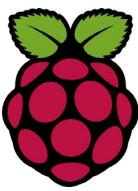
## QUIZ TIME

**29) TCP stands for .....**

- a) transmission control protocol
- b) telecommunication control protocol
- c) temperature control protocol
- d) transmission and communication protocol

**30) IP stands for .....**

- a) intelligent protocol
- b) internet protocol
- c) intercommunication protocol
- d) ideal protocol



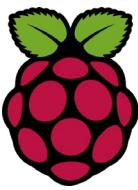
## QUIZ TIME

**31) DNS stands for .....**

- a) determine name system
- b) domain name system
- c) device name system
- d) development name system

**32) The process of building iot hardware and devices enhanced with smart sensors and embedded system using many of the shelf components like sensors , circuits and microcontrollers is called .....**

- a) prototyping
- b) casting
- c) protocasting
- d) protocol typing



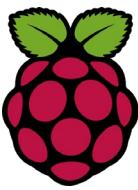
## QUIZ TIME

**33) SOC stands for ..... .**

- a) system on chip
- b) system on change
- c) source on chip
- d) source on change

**34) A ..... combined a required electronic circuit of various computer components onto a single integrated chip.**

- a) system on chip
- b) system on change
- c) source on chip
- d) Source on change



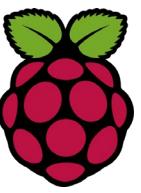
## QUIZ TIME

**35) ..... and ..... are main components of raspberry pi.**

- a) LED , USB
- b) USB , HDMI
- c) LED , HDMI
- d) USB , POWER

**36) ..... is a capable little device that enables people of all ages to explore computing and to learn how to program in languages like Scratch and Python.**

- a) raspberry pi
- b) python programming
- c) Linux
- d) web programming



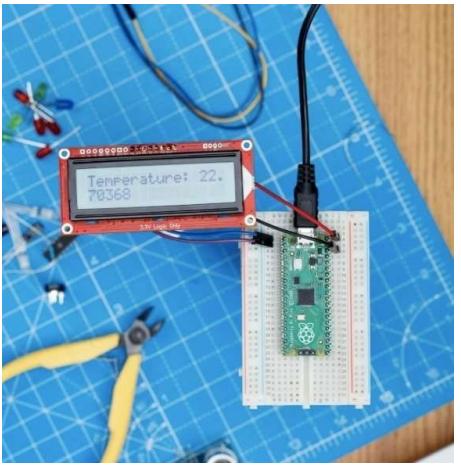
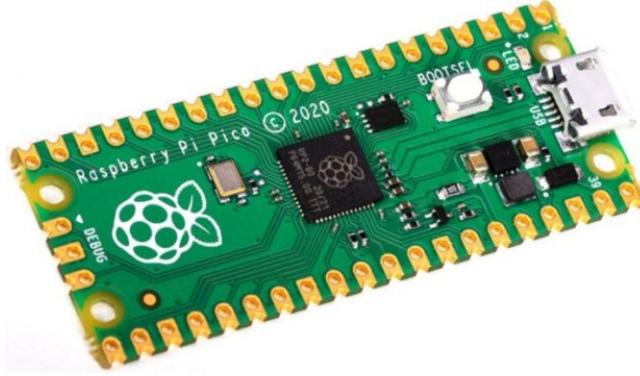
## QUIZ TIME

**37) API stands for .....**

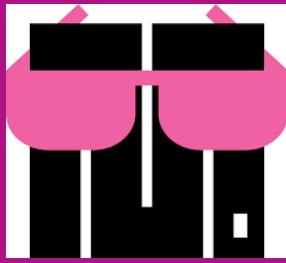
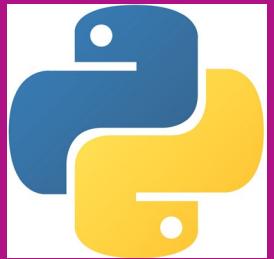
- a) application programming interface
- b) Android programming interface
- c) Arduino protocol information
- d) application protocol interface

**38) ..... is the process of making a physical representation of an idea.**

- a) physical proto casting
- b) physical prototyping
- c) type casting
- d) process interface

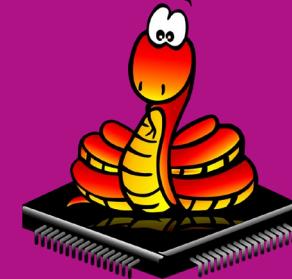


# INTERNET OF THINGS (IOT) PROJECTS USING PYTHON

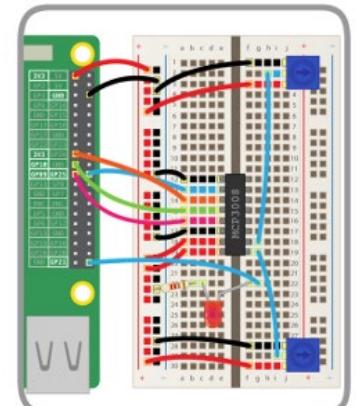
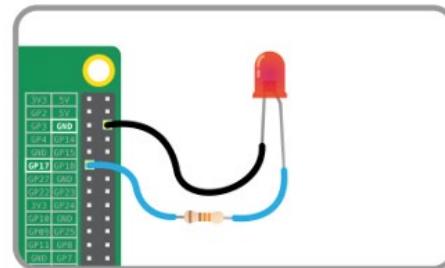
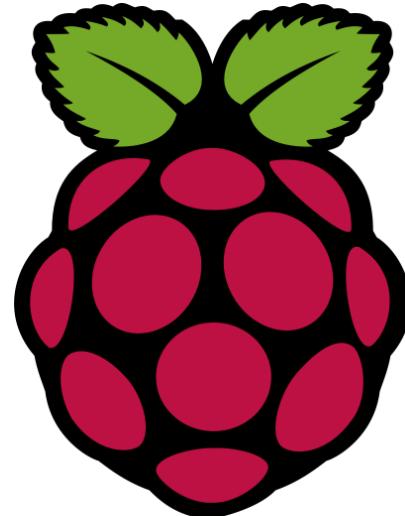
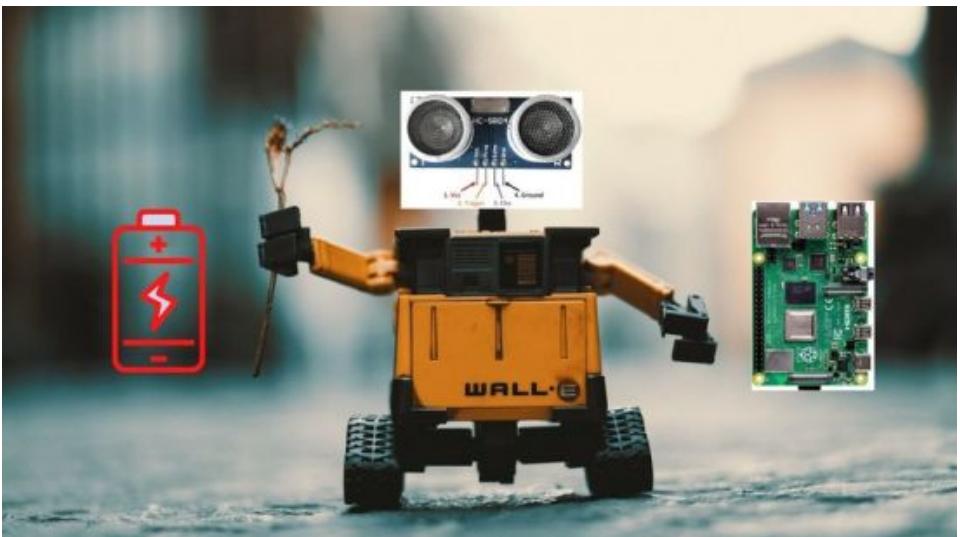


(CSE 4110)

(LECTURE – 2)

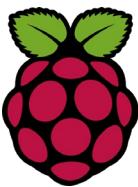


T<sub>h</sub>





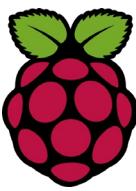
# What Is Outcome-Based Education? OBE Vs Traditional Education System



- Outcome-based education is a system where all the parts and aspects of education are focused on the outcomes of the course. The students take up courses with a certain goal of developing skills or gaining knowledge and they have to complete the goal by end of the course.
  
- There is no specific style or time limit of learning. The student can learn as per their choice. The faculty members, moderators, and instructors guide the students based on the target outcomes.



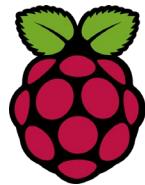
# Benefits Of Outcome-Based Education (OBE) For Students



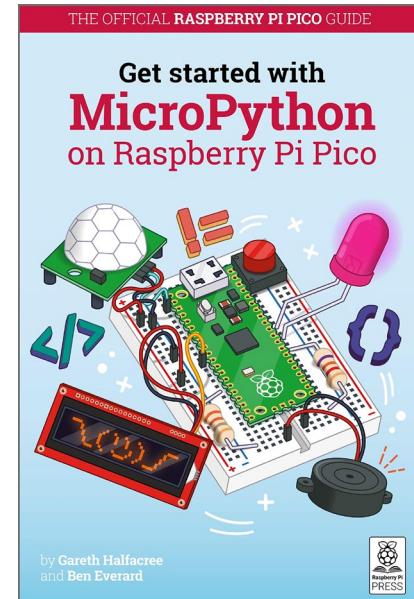
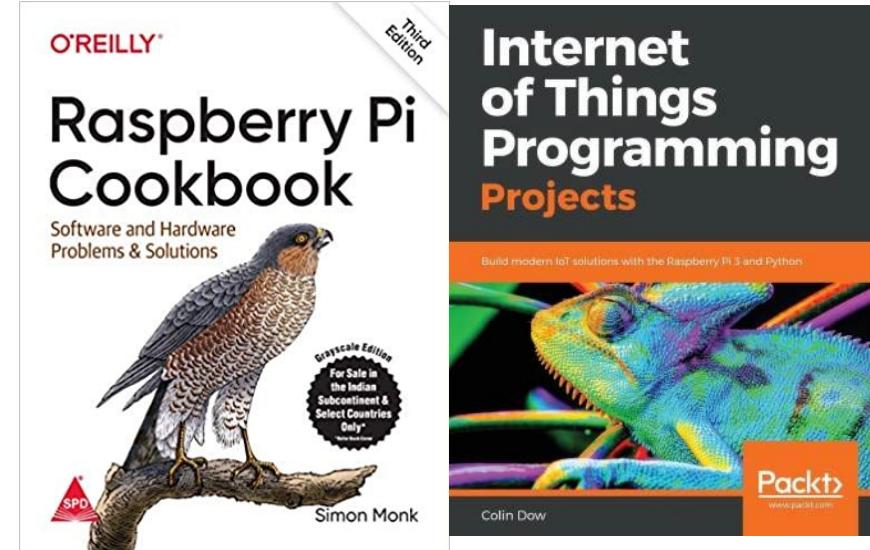
- Brings clarity among the teachers and students
- Every student has the flexibility and freedom of learning in their ways.
- There is more than one method of learning
- Reduces comparison among the students as everyone has a different target
- Completely involves students taking responsibility for their goals



# About your subject

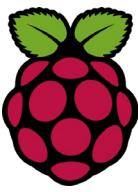


- Grading Pattern : 3
- Credit : 4
- TextBook:
  - 1) Raspberry Pi Cookbook by by Simon Monk, shroff/O'Reilly
  - 2) Internet of Things Programming Projects by By Colin Dow, Packt,
- Reference : 3) Get started with Micro-Python





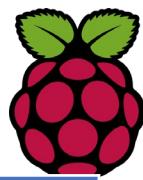
# Evaluation Scheme



- ATTENDANCE : 5
- WEEKLY ASSIGNMENTS / QUIZZES : 20 MID-TERM
- MID TERM : 15
- TOTAL INTERNAL : 40
  
- FINAL ASSIGNMENT : 40
- ASSIGNMENT PRESENTATION : 20 END-TERM
- TOTAL EXTERNAL : 60



# Course Outcomes

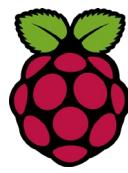


## Course Outcomes

CO1	Understand general concepts of Internet of Things (IoT), the working of Raspberry Pi and its features.
CO2	Recognize various components, sensors, actuators, devices and their applications.
CO3	Analyze various python programs to interface with sensors, actuators, LED's, cloud and camera using Raspberry pi.
CO4	Measure physical parameters using sensors.
CO5	Demonstrate the ability to transmit data wirelessly between different devices to build simple IoT systems using Raspberry Pi.
CO6	Create IoT devices and systems through a variety of interfaces, including web apps and mobile apps.

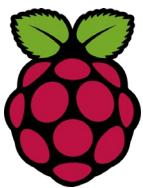


# What Students will learn?

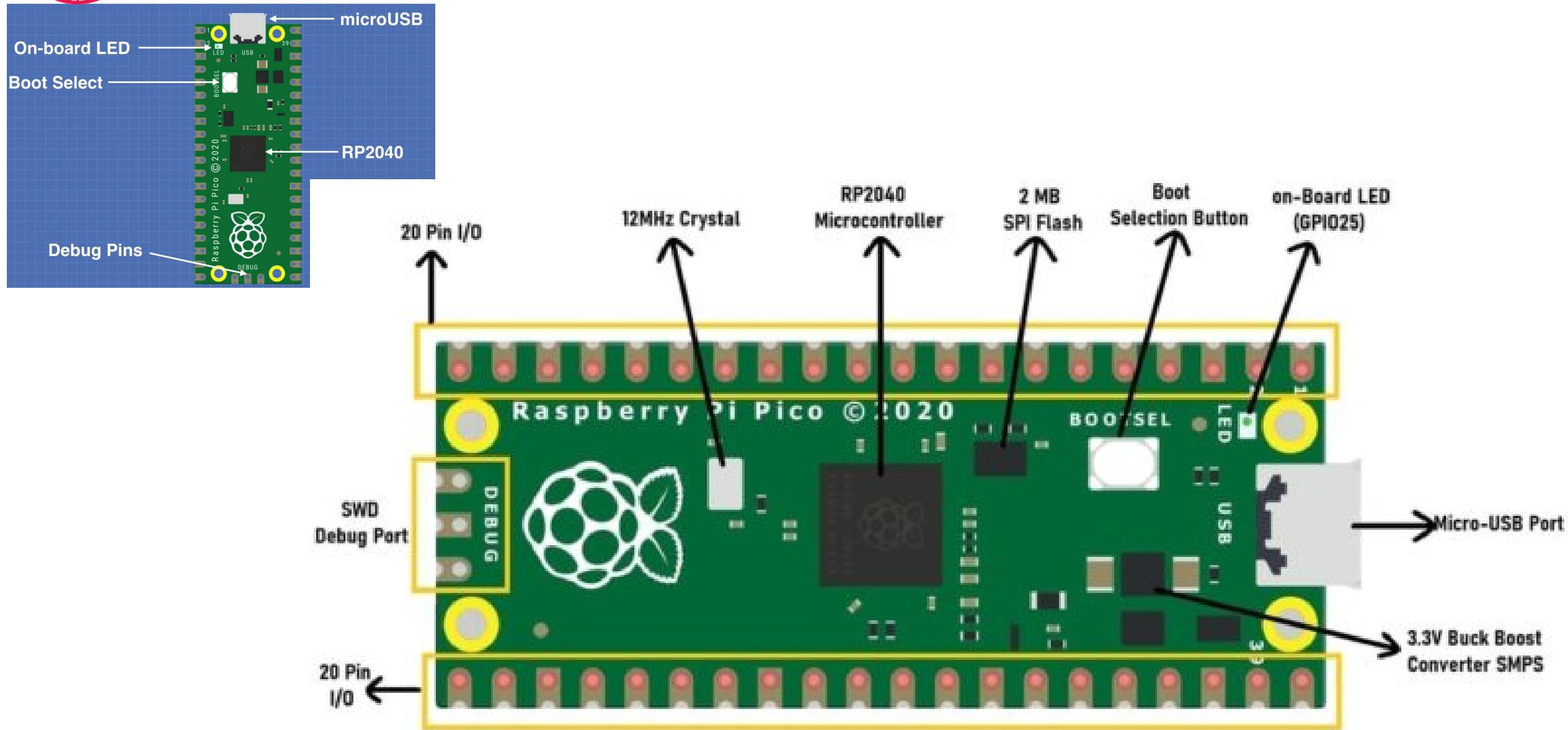


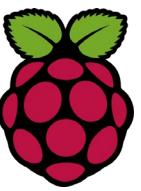
Students will revise the basics of electronics, types of electrical signal, Resistance colour code to enhance the knowledge to build an electronic circuit. Also students will experience about ability to learn coding and electronics to implement a LED flash SOS (Morse code) .

- ✓ Familiarization with Basic Electronic concepts, their origin, impact, and different applications to improve technical results.
- ✓ Introduction to Resistance Colour code
- ✓ Implementation of LED Flash SOS (Morse Code) using Raspberry Pi Pico



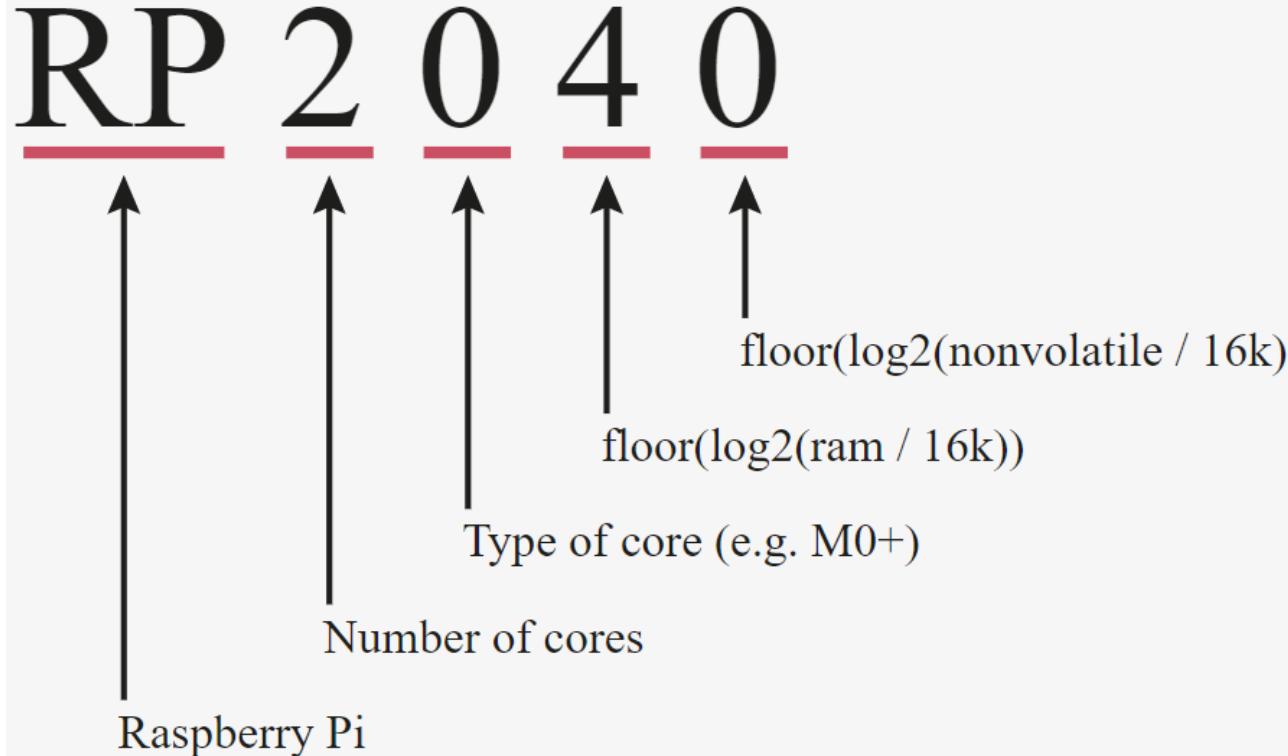
# Raspberry Pi Pico – Simple Pinout





# Why is the chip called RP2040?

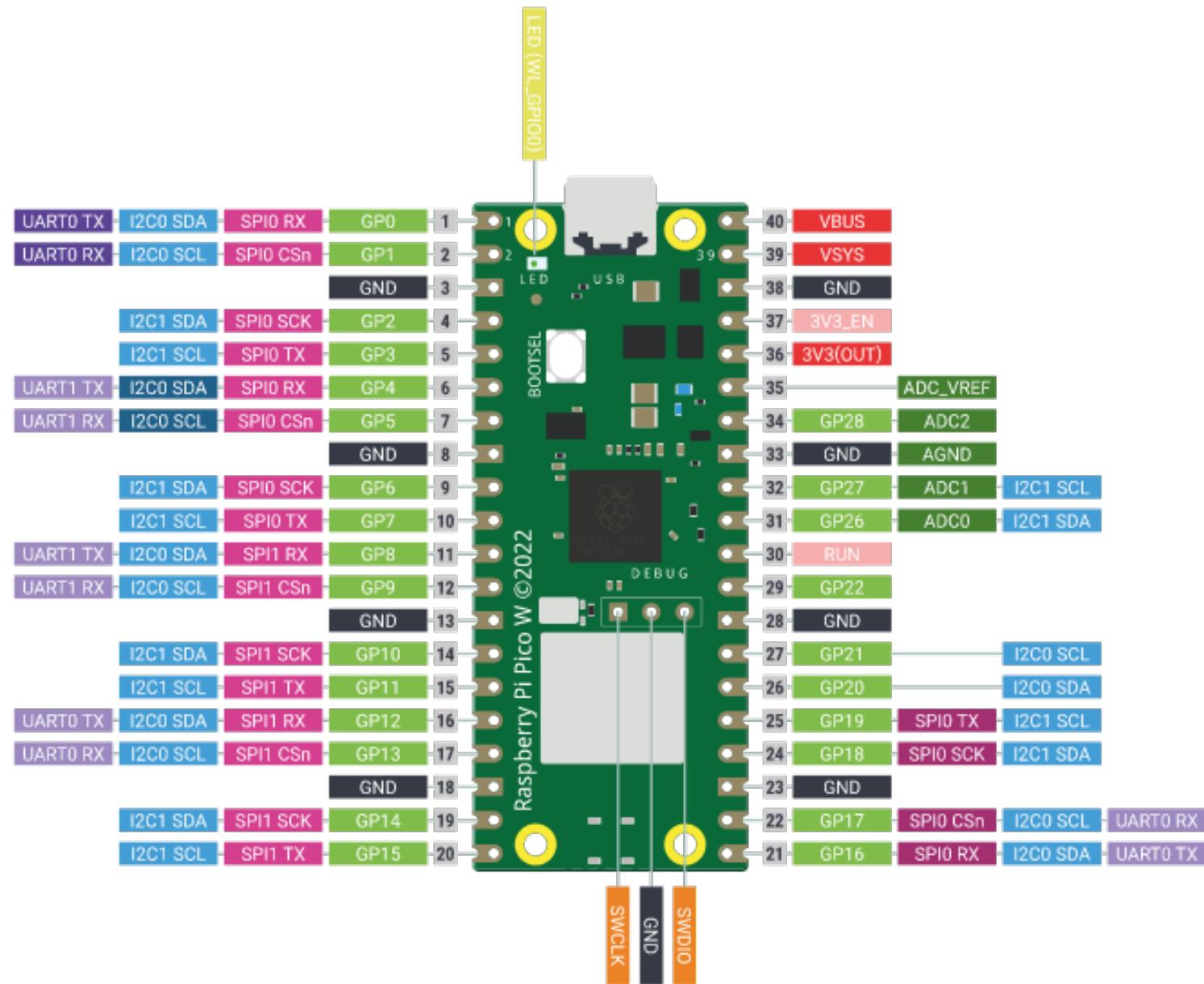
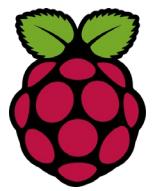
The post-fix numeral on RP2040 comes from the following,



1. Number of processor cores (2)
2. Loosely which type of processor (M0+)
3.  $\text{floor}(\log_2(\text{ram} / 16\text{k}))$
4.  $\text{floor}(\log_2(\text{nonvolatile} / 16\text{k}))$  or 0 if no onboard nonvolatile storage



# Raspberry Pi Pico W Pinout



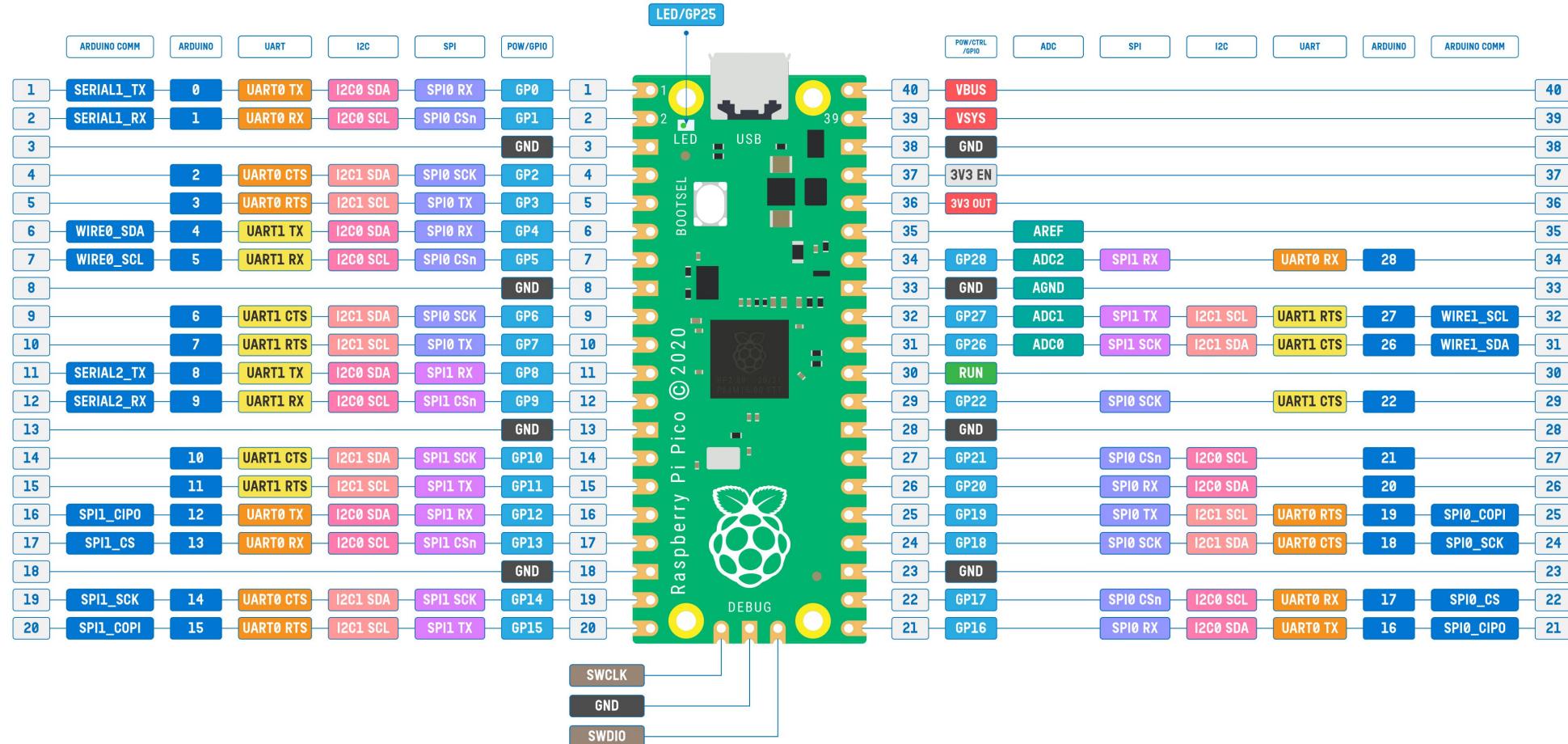
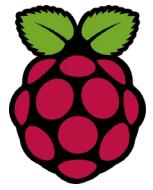
RP2040



Infineon 43439



# Raspberry Pi Pico – Full Pinout



\*Raspberry Pi and the Raspberry Pi logo are trademarks of Raspberry Pi Ltd.

Raspberry Pi Pico vector image is originally designed by Raspberry Pi. Please visit [raspberrypi.com](https://www.raspberrypi.com) for more info.

## ARDUINO PINS

## SWD Pins

PHYSICAL PIN	POSITIVE SUPPLY	UART1 Pins	UART0 Pins
RESET/ENABLE	GROUND SUPPLY	I2C1 Pins	I2C0 Pins
GPIO PORT/PIN	ANALOG PIN	SPI1 Pins	SPI0 Pins

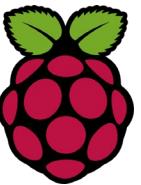
- GP29/ADC3 is used to measure VSYS.
- GP25 is used for debug LED.
- GP24 is used for VBUS sense.
- GP23 is connected to SMPS Power Save pin.
- All GPIO pins support PWM. There are total 16 PWM channels.
- All GPIO pins support level and edge interrupts.
- Arduino pins are as per [Arduino-Pico core](#) by [@earlephilhower](mailto:Earle F. Philhower, III)
- Arduino's default Serial is the USB-CDC of Pico.



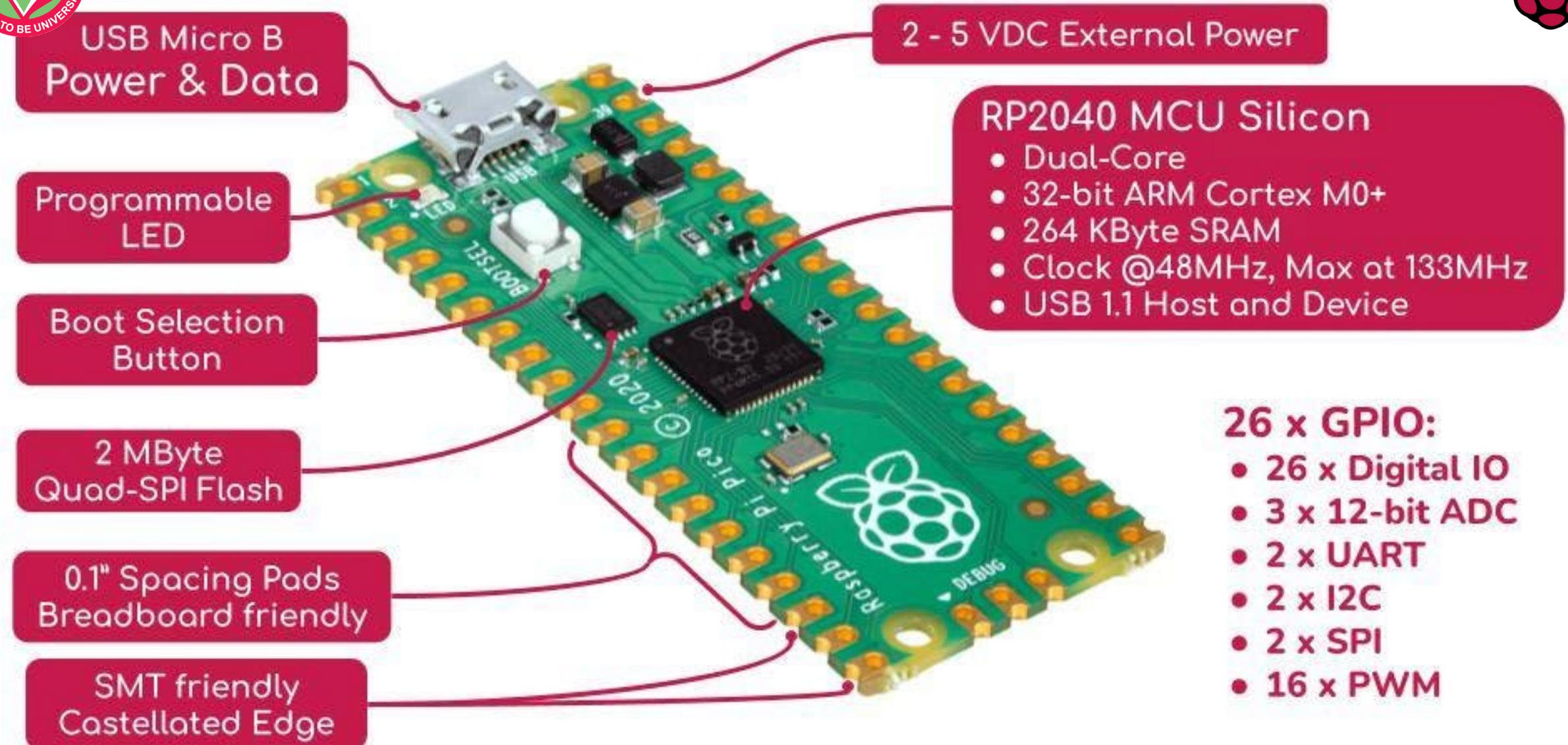
Rev. 0.2, 15-07-2022

Design: Vishnu Mohanan

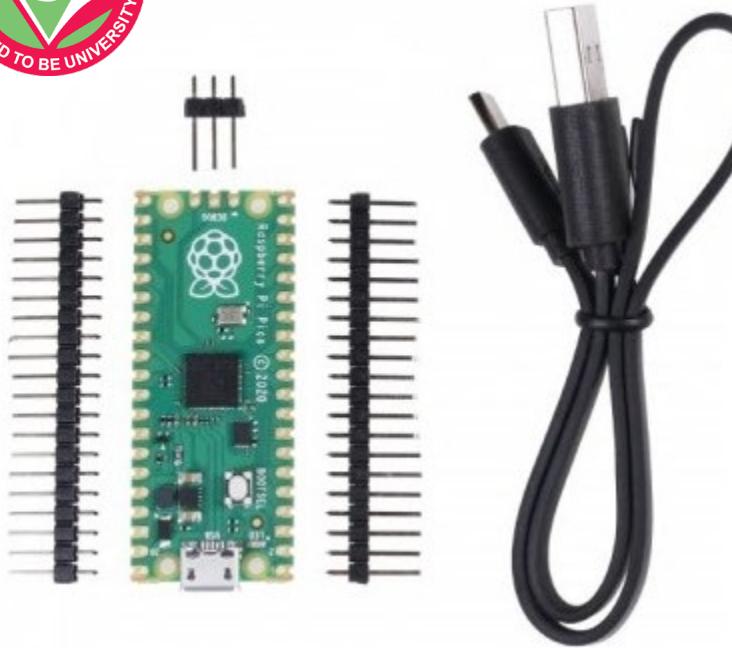
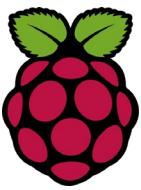
This work is licensed under a Creative Commons  
Attribution 4.0 International License.



# Raspberry Pi Pico – In Short

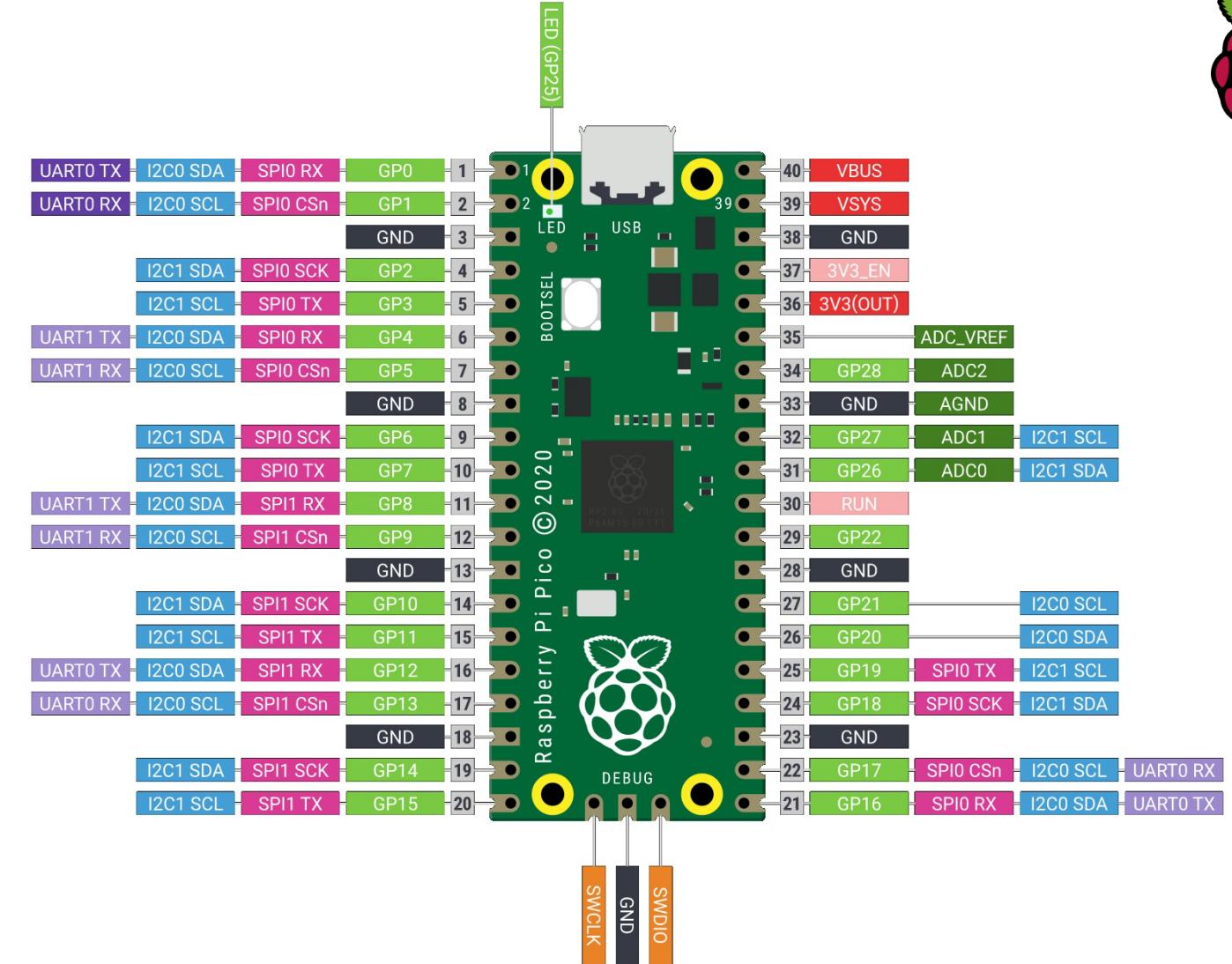


Pi Foundation has released an RP2040 Microprocessor based development board, in the same form factor as an Arduino Nano.

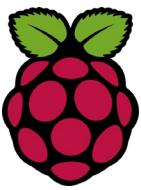


## Package Includes:

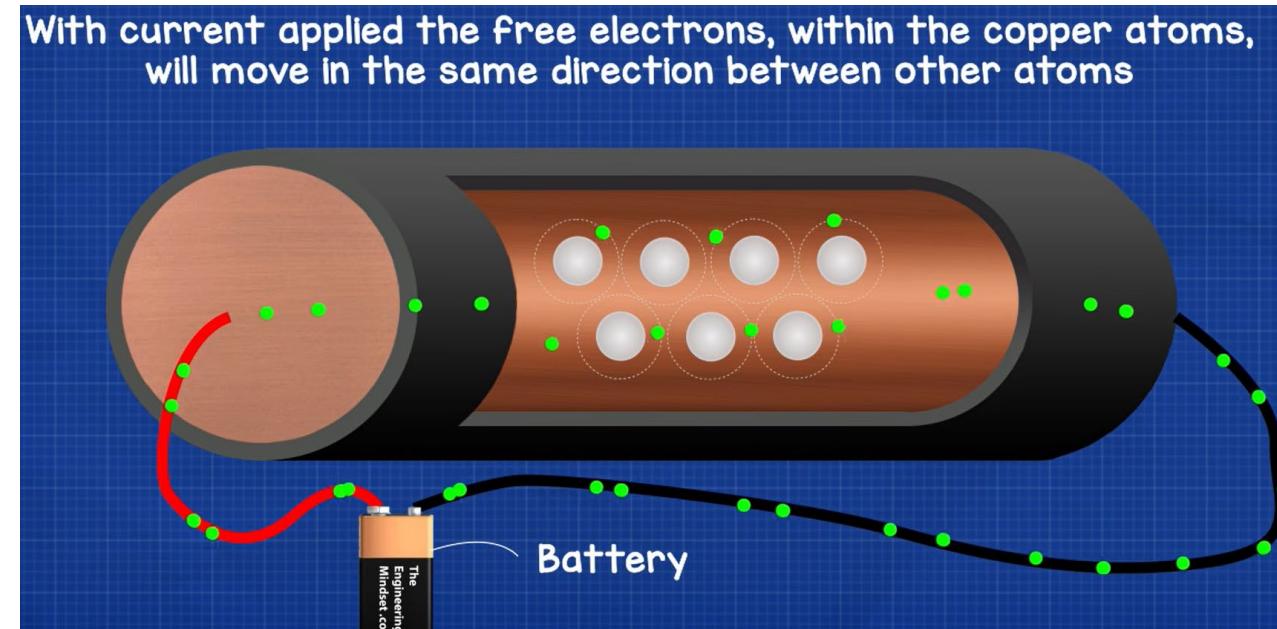
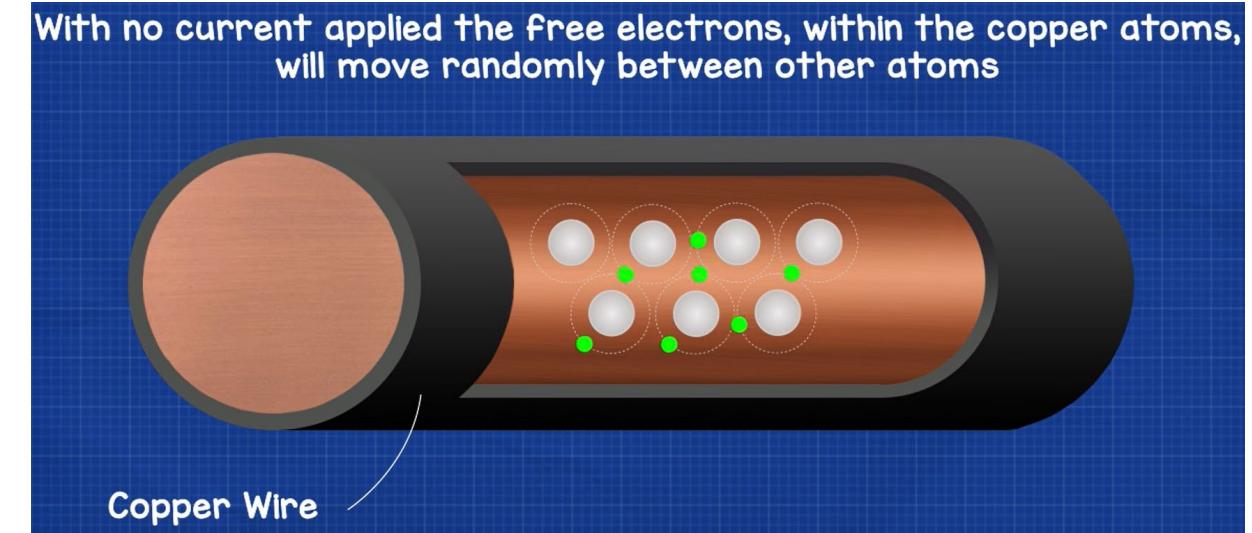
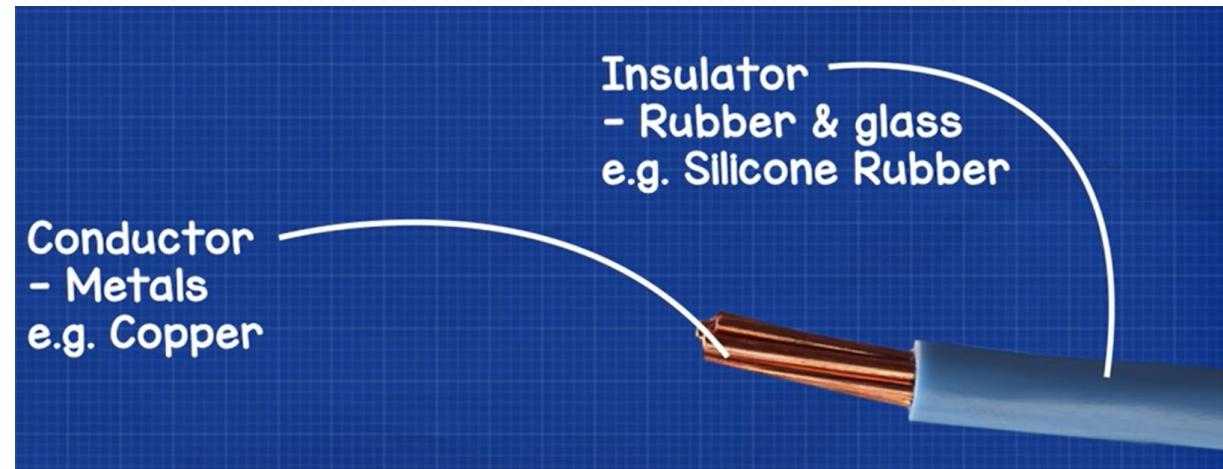
- 1 x Raspberry Pi Pico
- 1 x Micro-USB cable
- 2 x 20 Pin Header
- 1 x 3 Pin Header

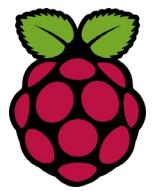


■ Power ■ Ground ■ UART / UART (default) ■ GPIO, PIO, and PWM ■ ADC ■ SPI ■ I2C ■ System Control ■ Debugging

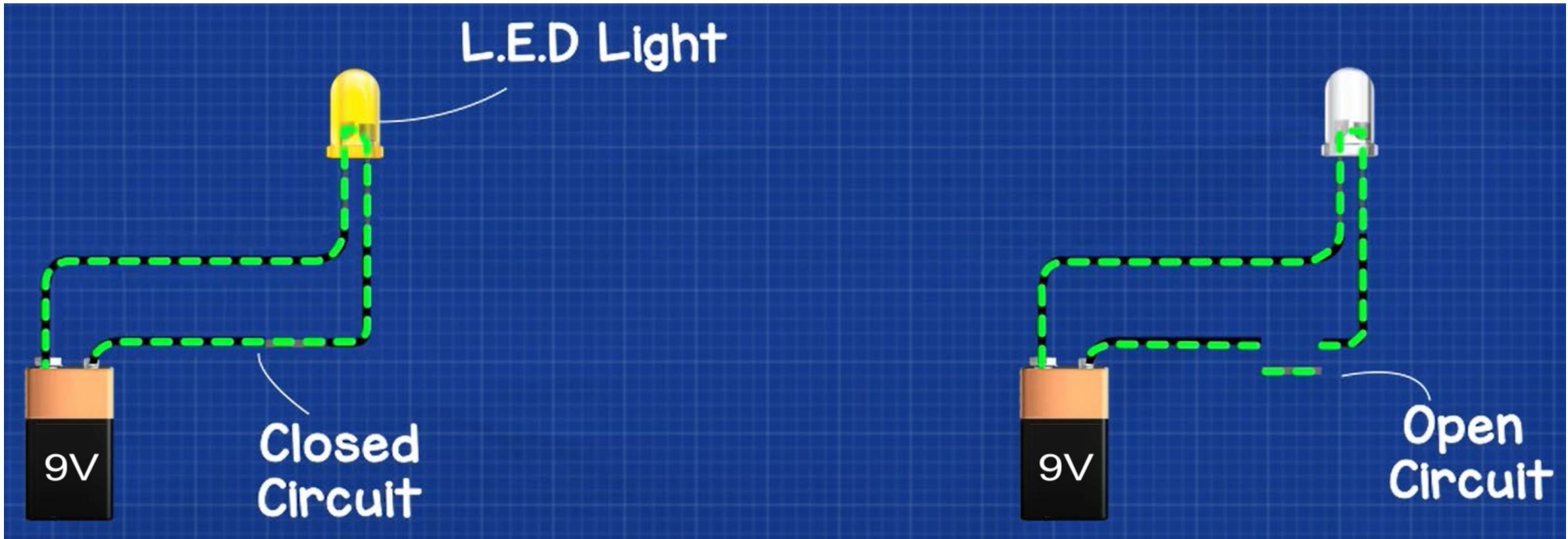


# Conductors and Insulators





# Open and Closed Circuit



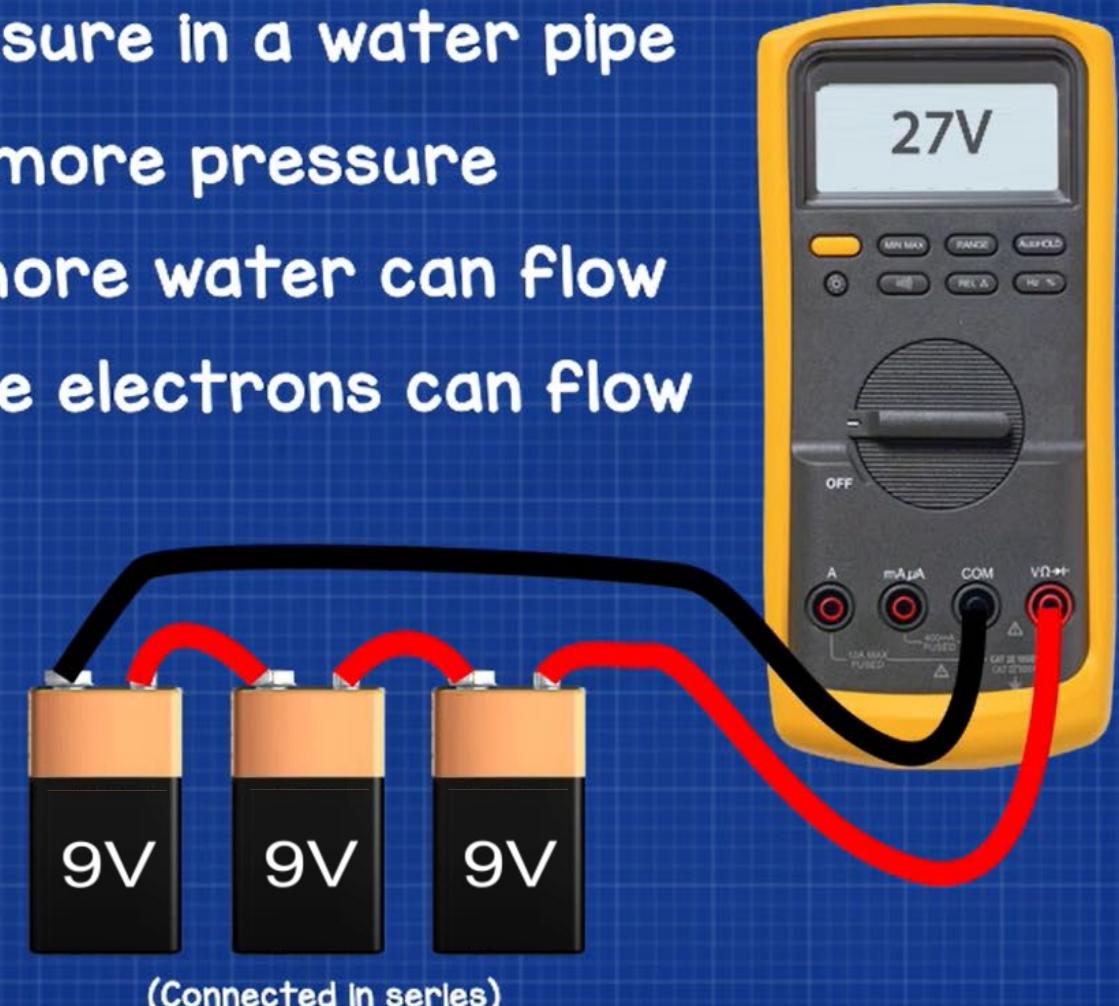
# Voltage

Think of Voltage like pressure in a water pipe

More voltage means more pressure

More pressure means more water can flow

More Voltage means more electrons can flow



(Connected in series)

# Voltage



What does a "Volt" mean?

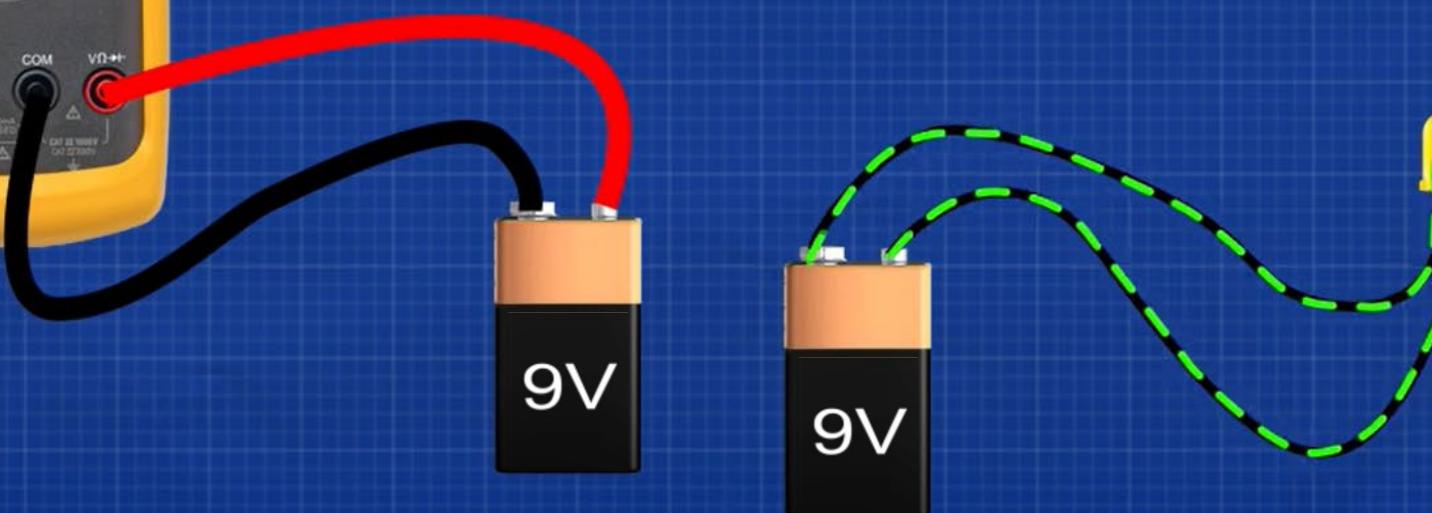
We know: more Voltage means more electrons can flow

$$\text{Volt} = \frac{\text{Joules}}{\text{Coulomb}}$$

Work

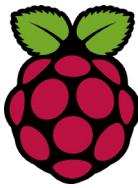
Per

Group of flowing electrons



A 9V Battery can do:

9 Joules of work,  
or heat, per coulomb

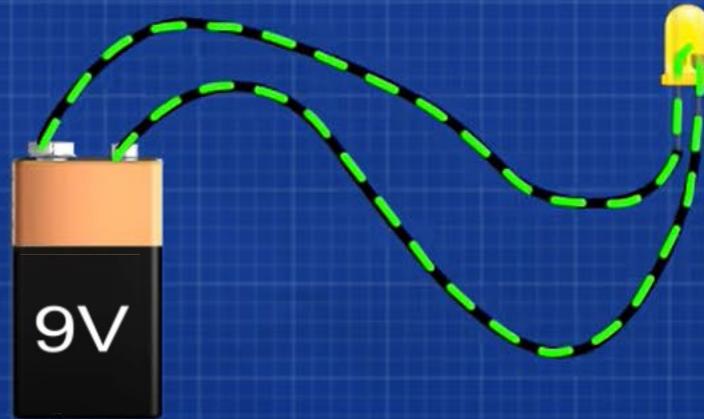


# Current

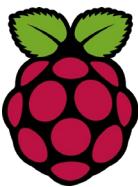
Current is the flow of electrons, measured in Amperes (Amps), past a single point in a circuit within a set amount of time

1 Amp = 1 Coulomb

1 Coulomb = 6,242,000,000,000,000,000 electrons per second



Electrons are negatively charged  
they flow from the negative terminal  
to the positive terminal



# Resistance

Resistance is a restriction to the flow of electrons

Resistance is measured in Ohms  $\Omega$

## Size

Length



Thickness



## Material

Copper

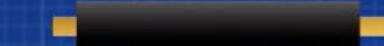


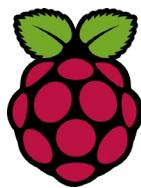
Aluminium



## Temperature

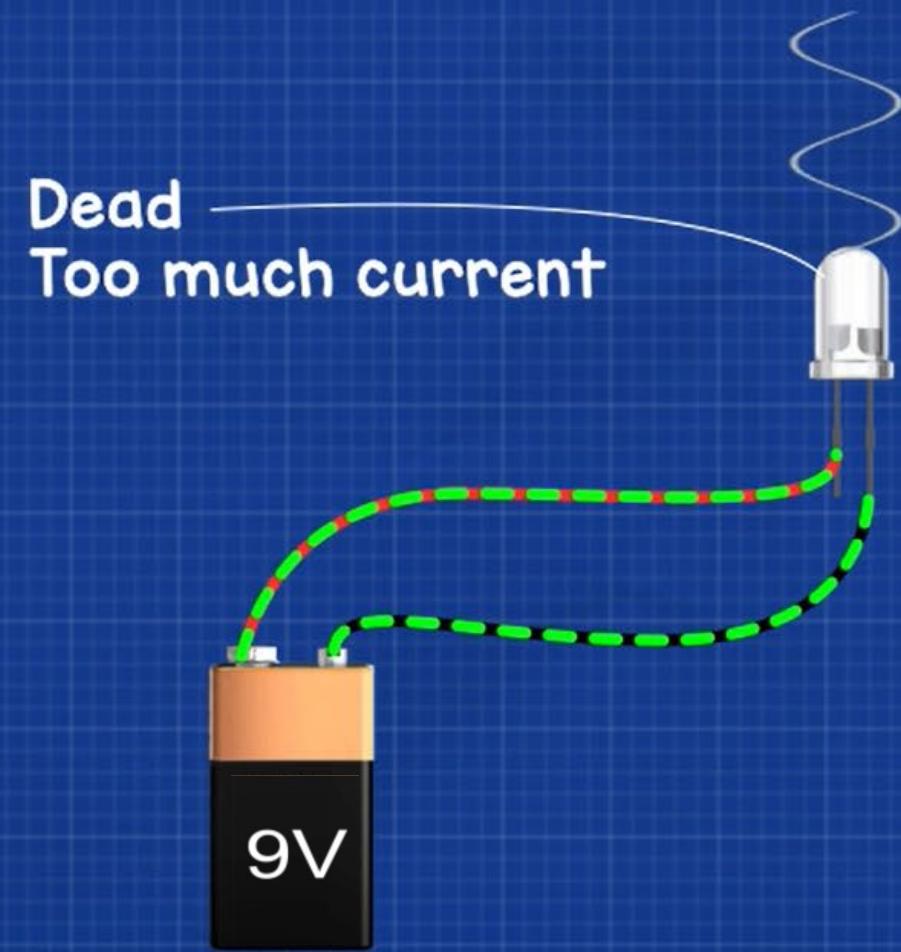
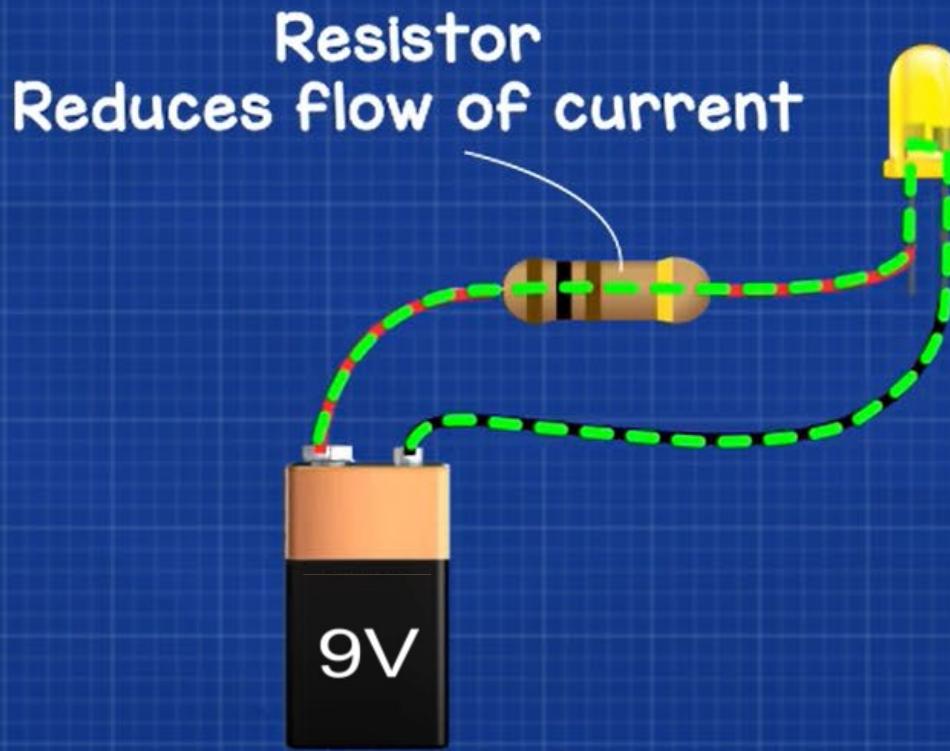
Less More

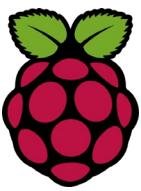




# Resistors

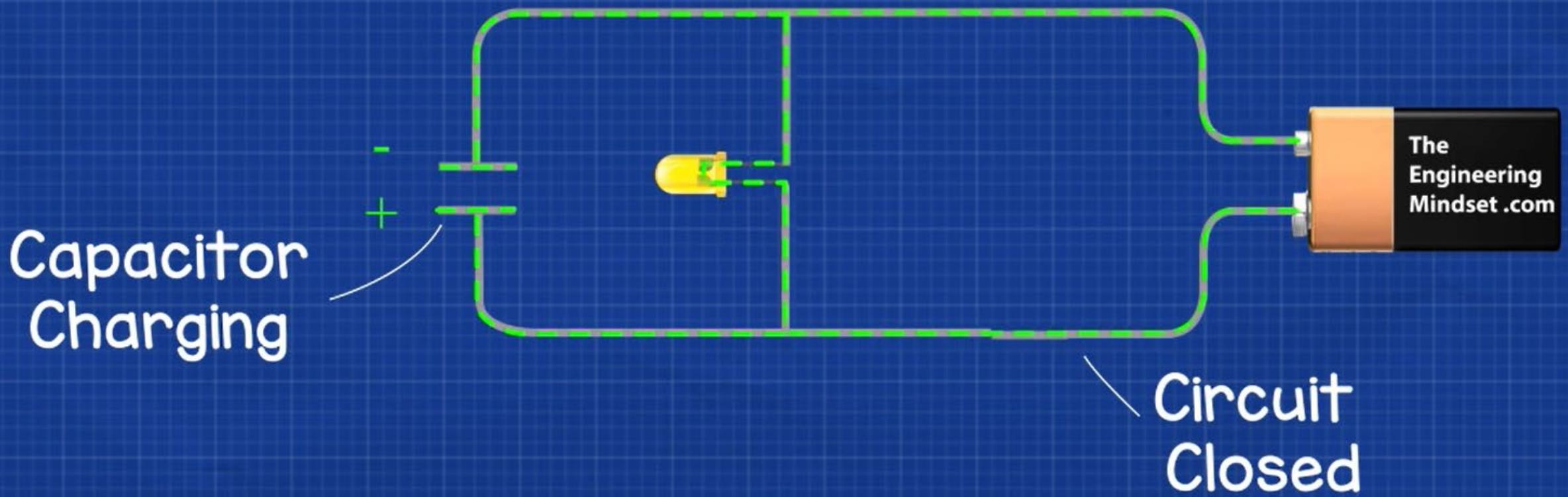
Resistors protect components by restricting the flow of current

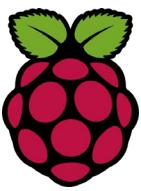




# Capacitors

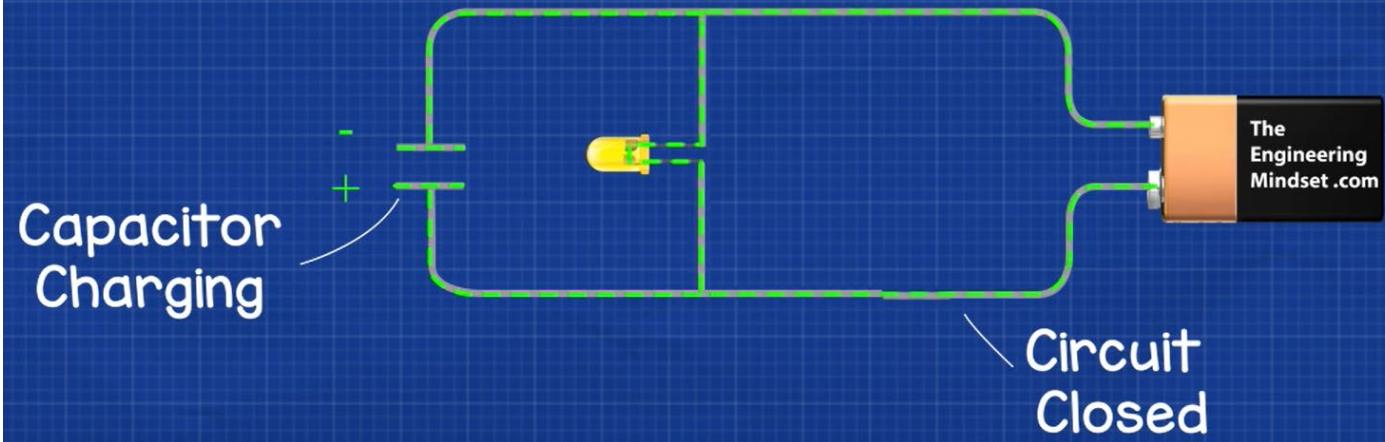
Capacitors store electric charge





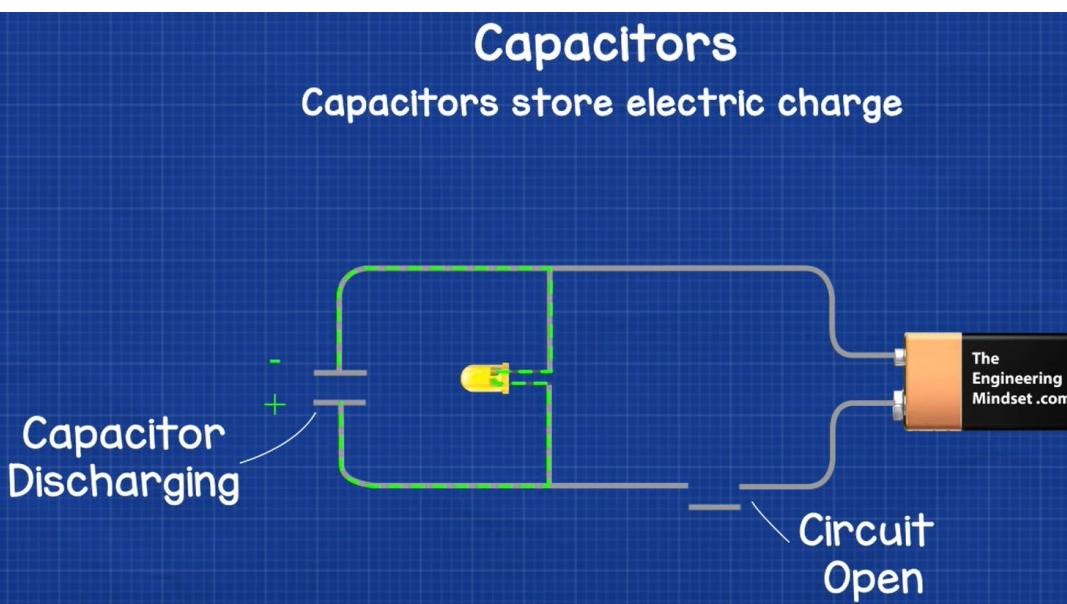
# Capacitors

Capacitors store electric charge



Capacitors

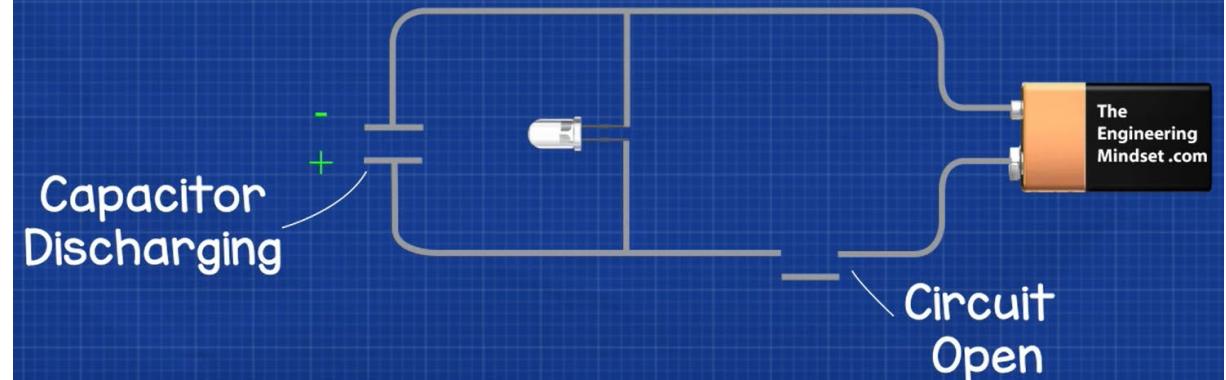
Capacitors store electric charge



Capacitors store electric charge

Capacitor  
Discharging

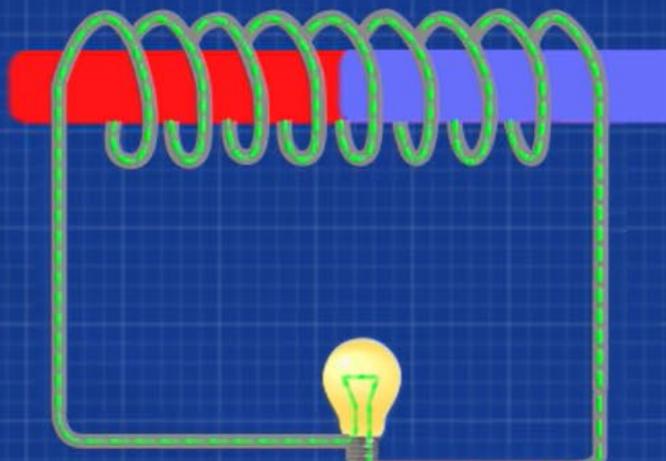
Circuit  
Open



Circuit  
Open

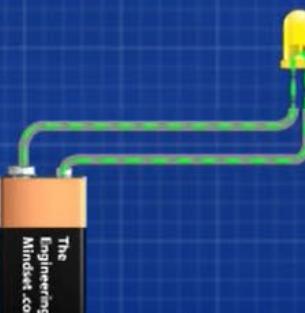
# Types of Electricity

There are two types of current electricity



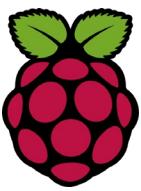
## Alternating Current (AC)

In AC the current moves back and forth with the changing magnetic field  
This is the type of electricity from wall sockets in your home



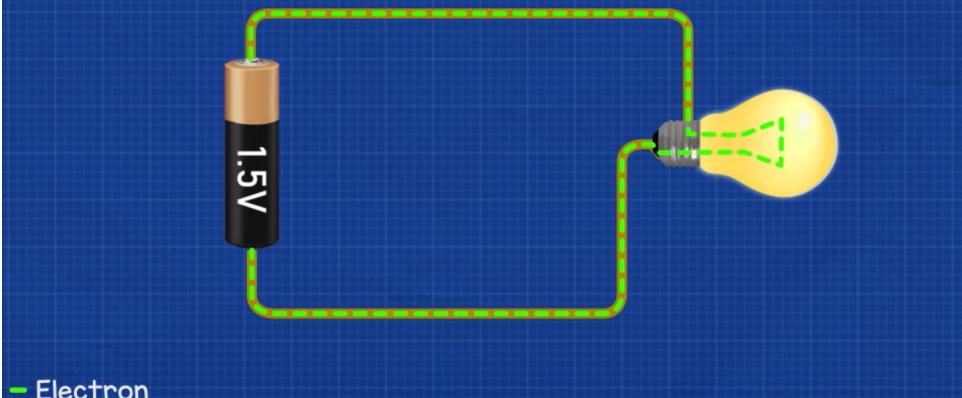
## Direct Current (DC)

In DC the current travels only in one direction  
This is the type of electricity in batteries

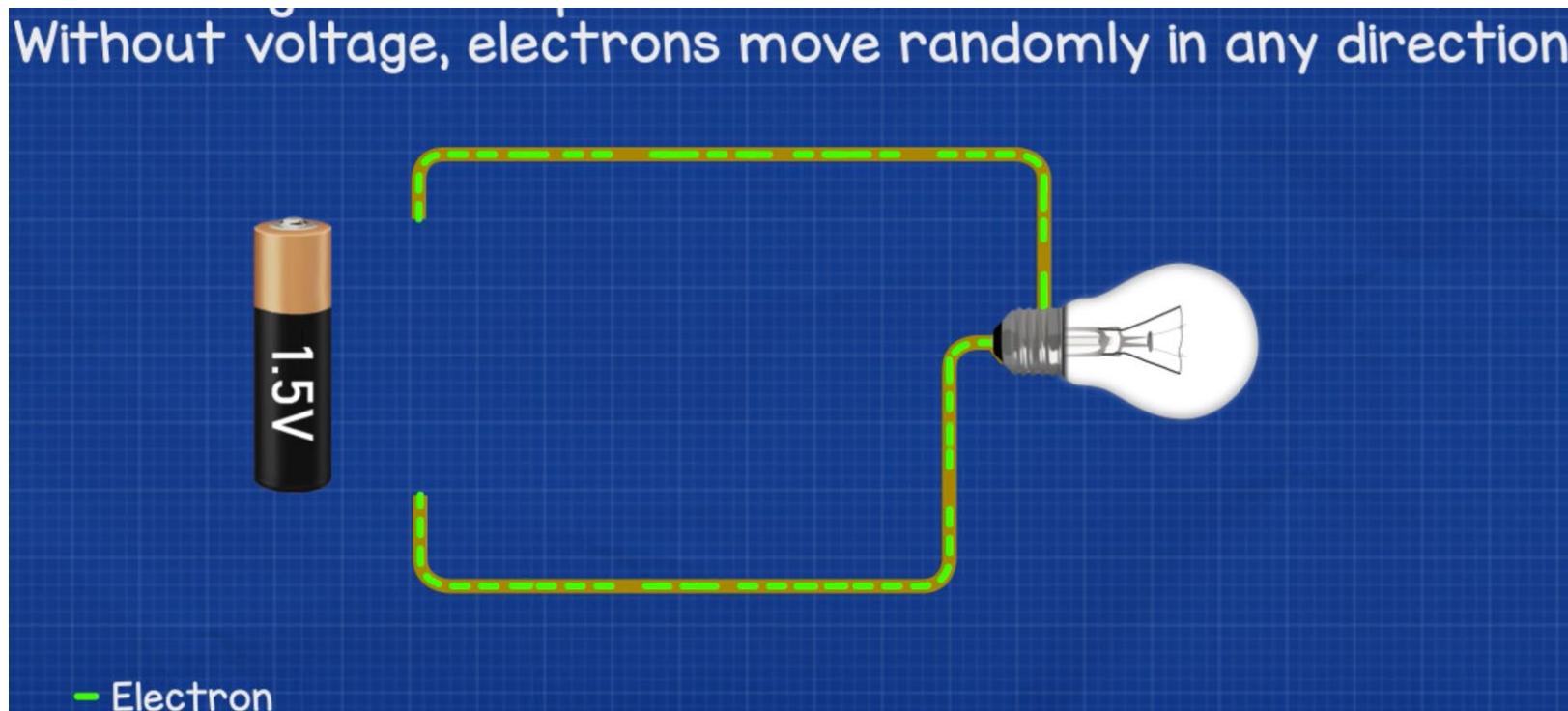


# Voltage

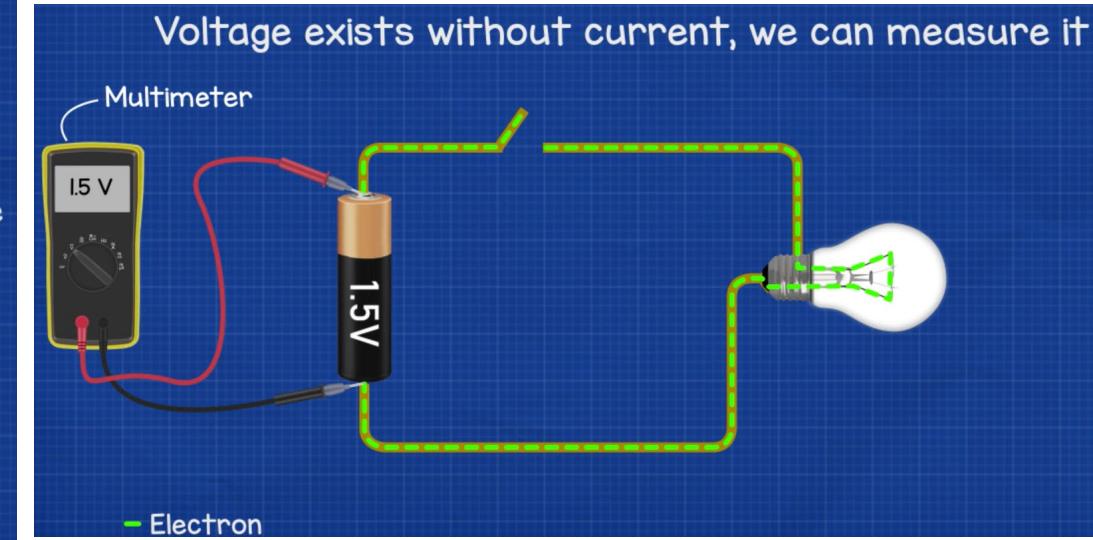
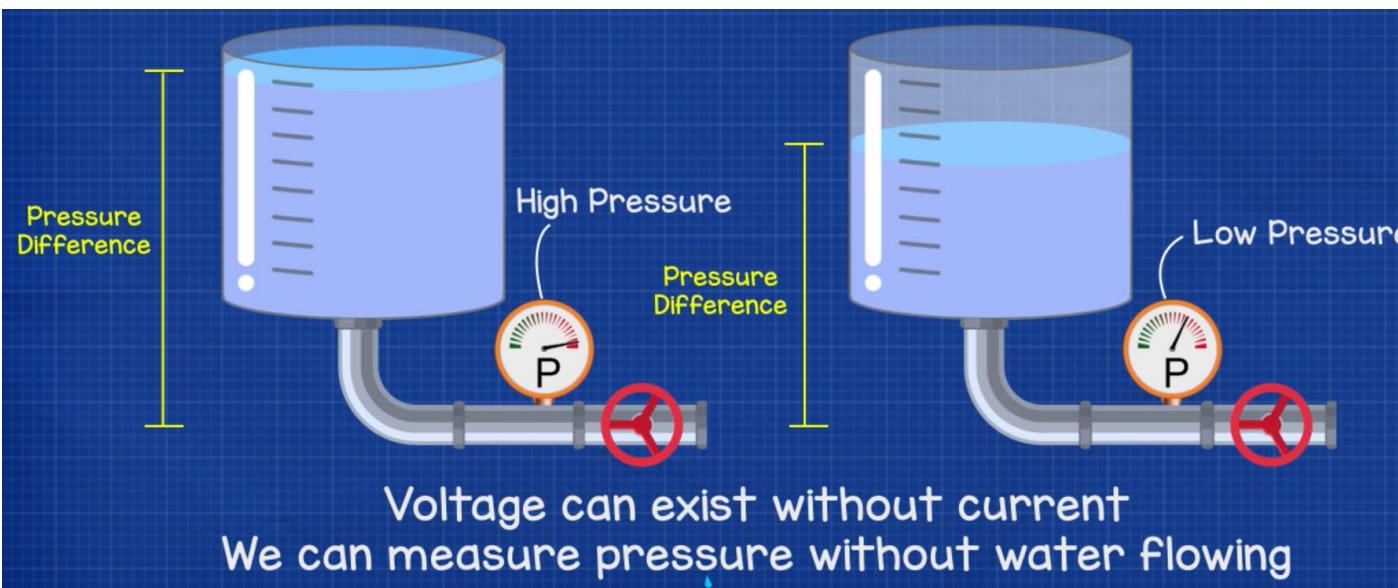
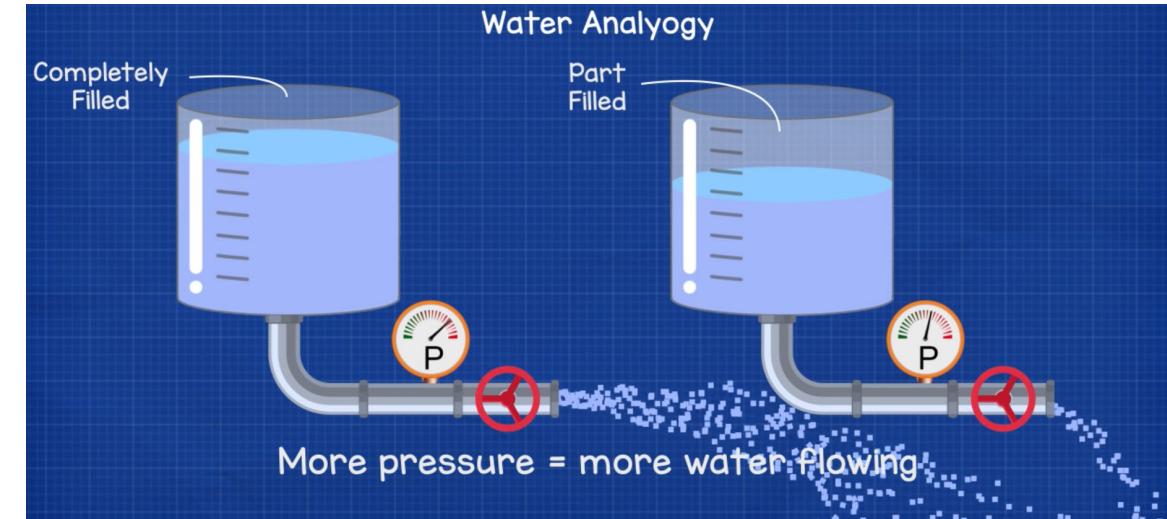
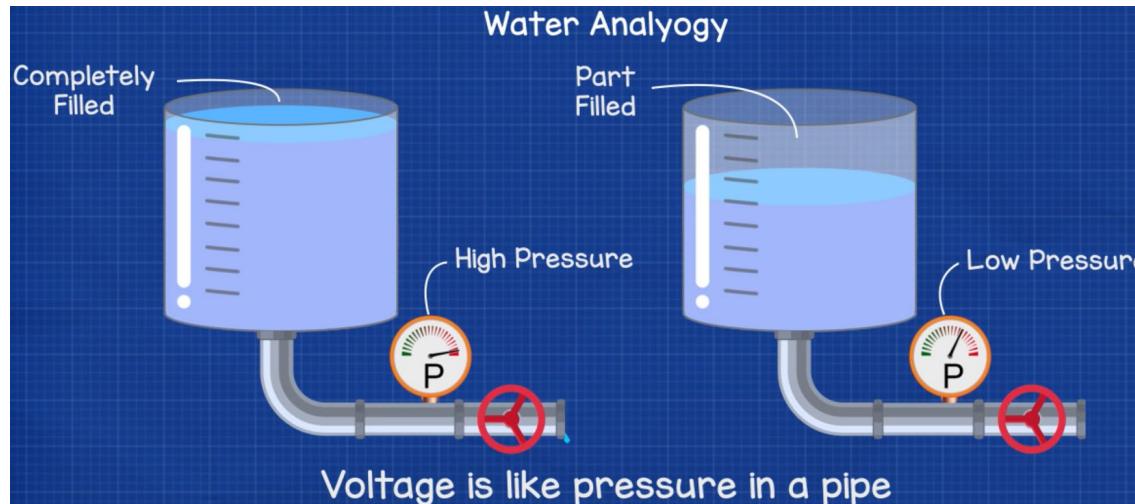
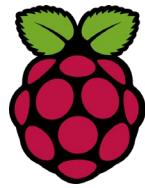
Voltage is what pushes electrons around a circuit



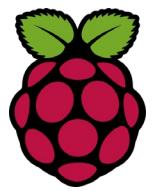
Without voltage, electrons move randomly in any direction



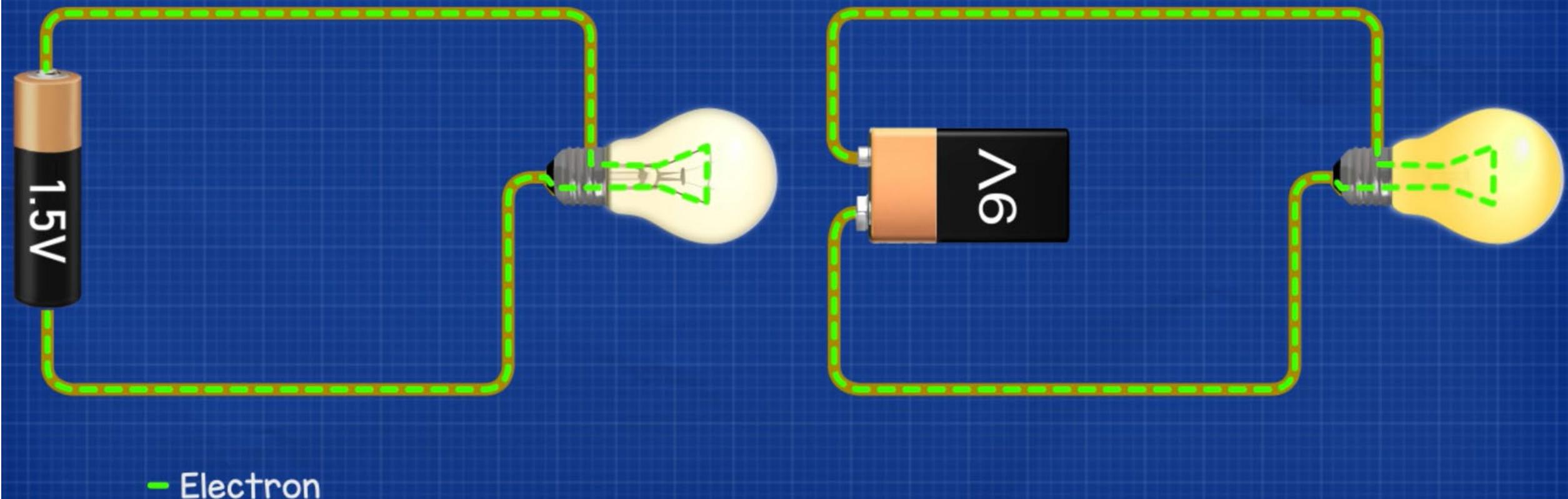
# Voltage



# Voltage

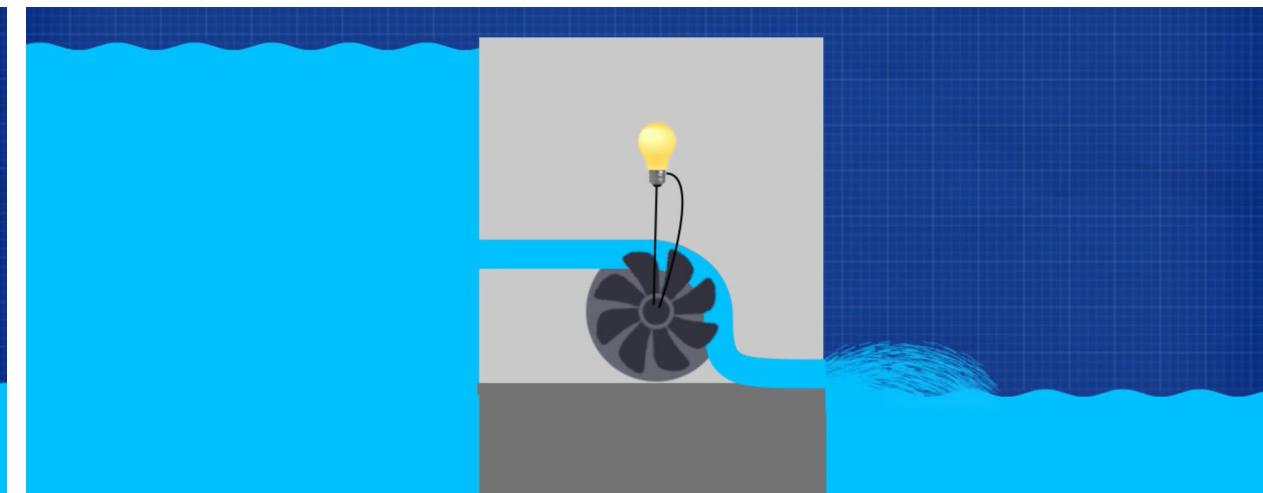
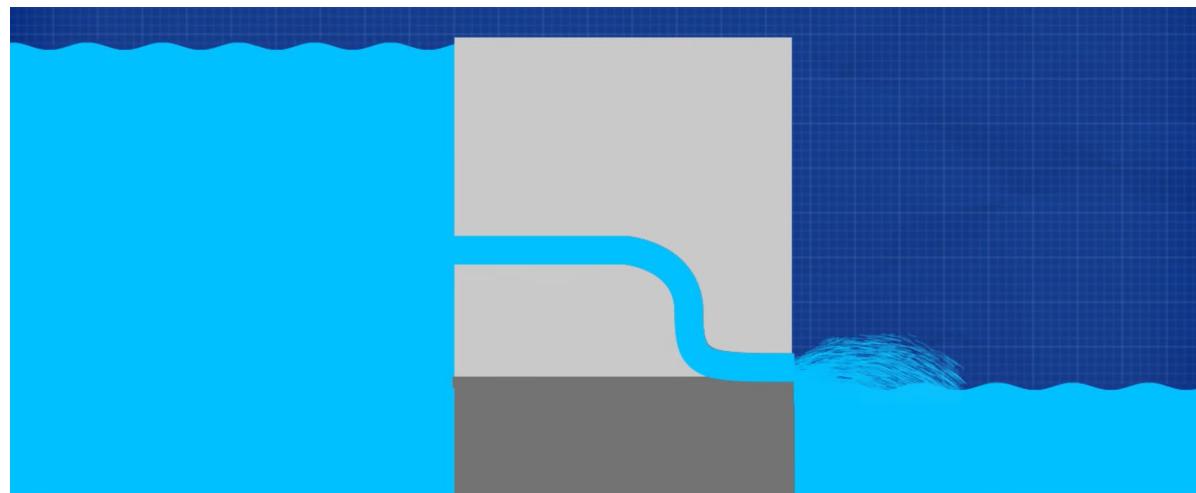
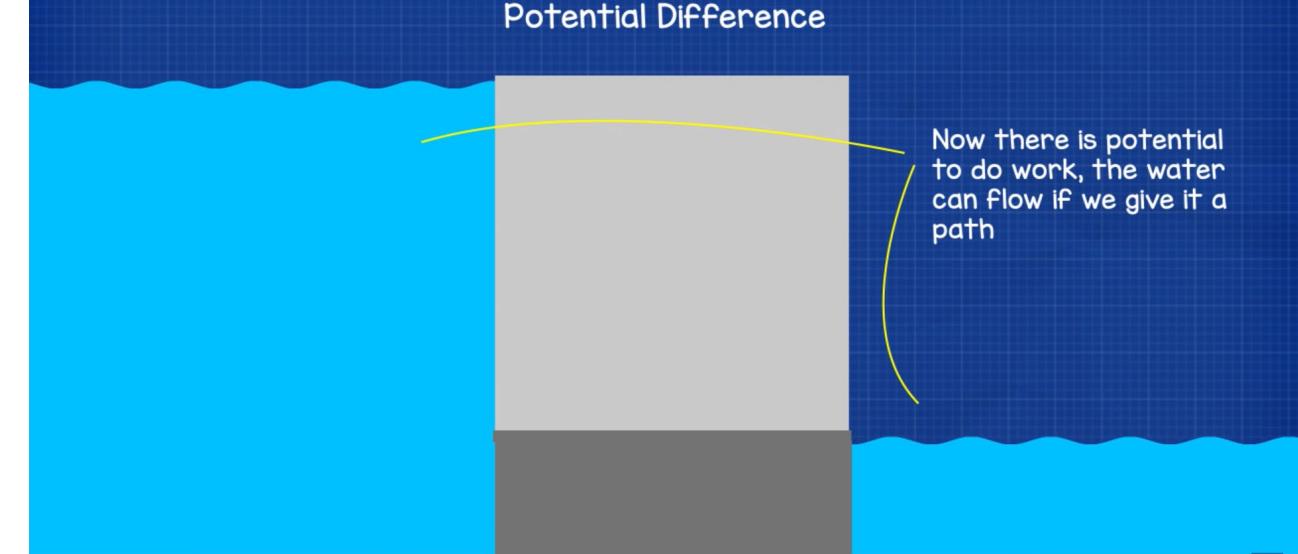
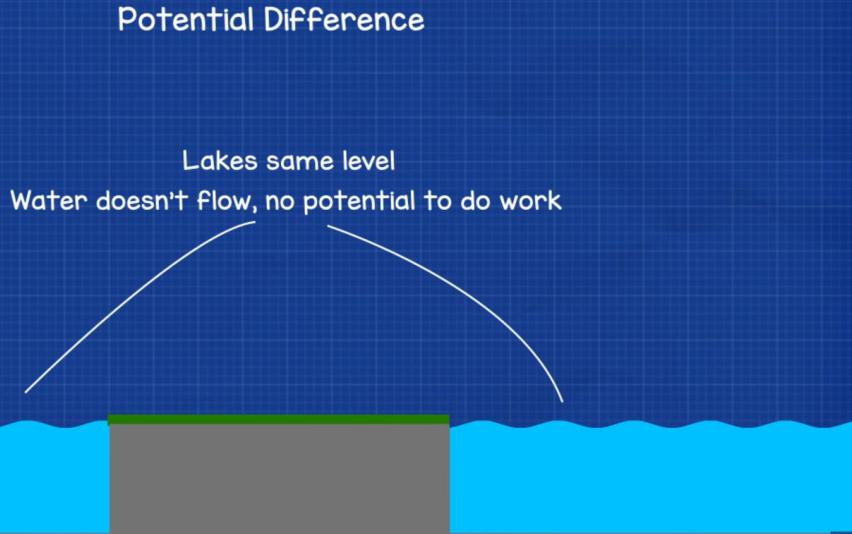
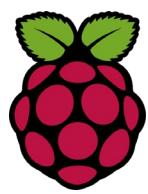


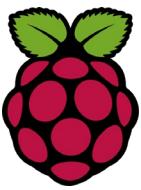
More voltage = more current  
More current = brighter lamp



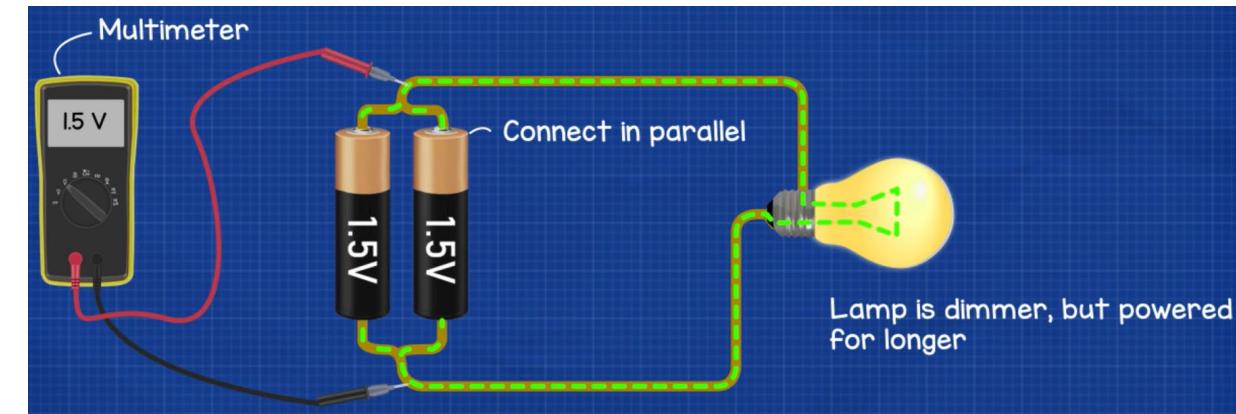
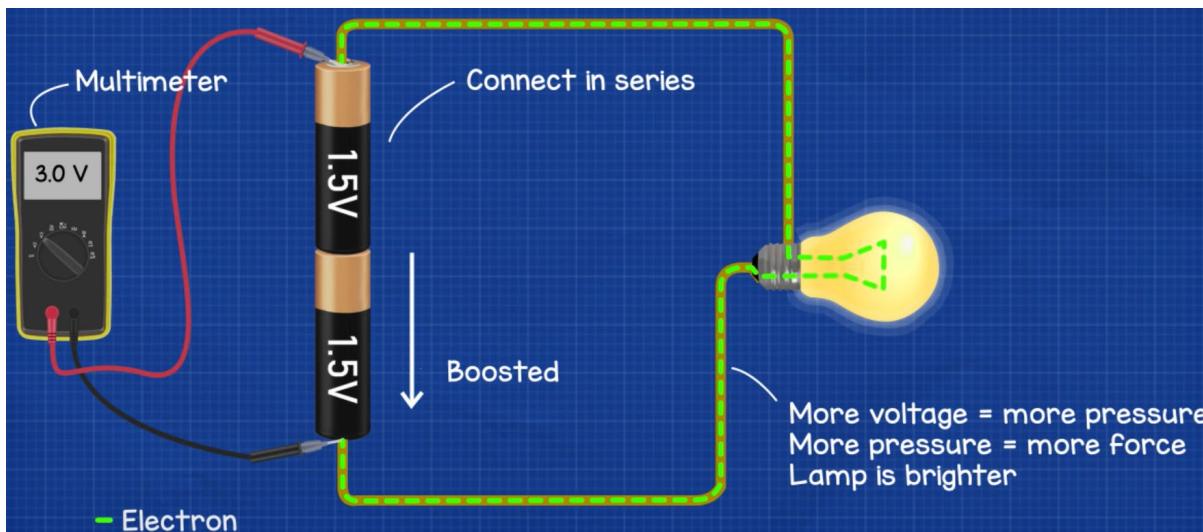
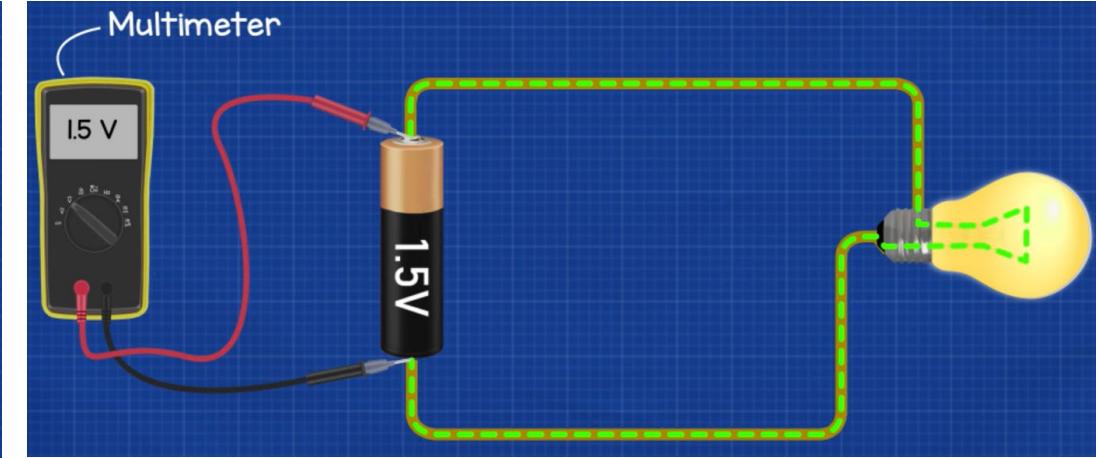
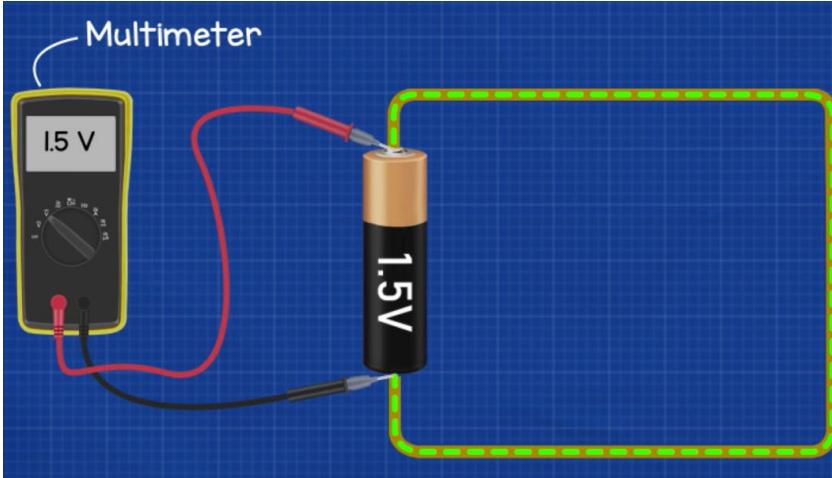
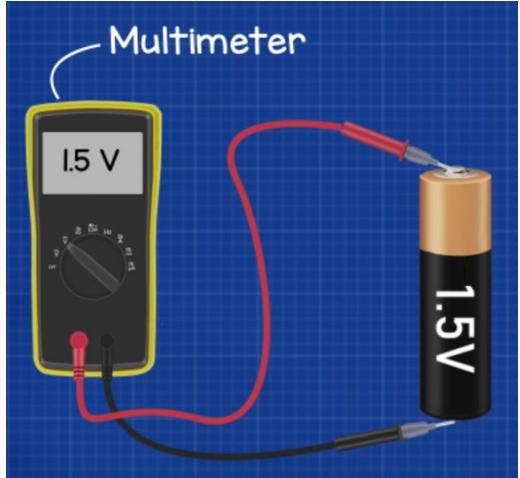


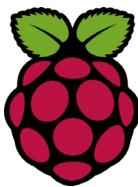
# Potential Difference





# Series and Parallel



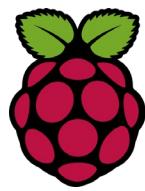


# Voltage Measurement

We measure potential difference or voltage in the units of Volts  
We represent this value with a capital "V"

## Manufacturers Label

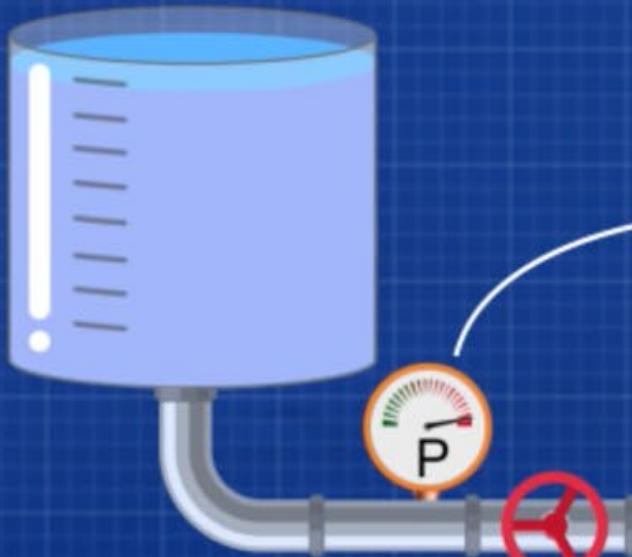




# Voltage and Volts are Different

**Voltage** is the pressure

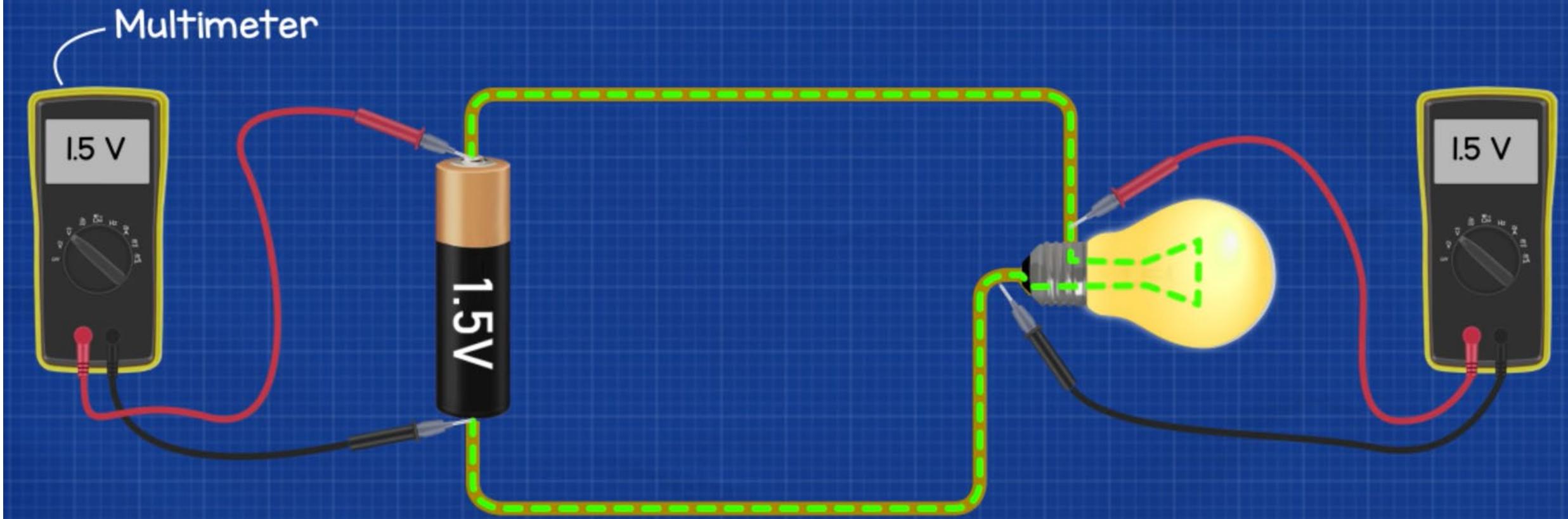
**Volts** is the units we measure the pressure in



Same as for water pressure we use the units of Bar, PSI, kPa etc.

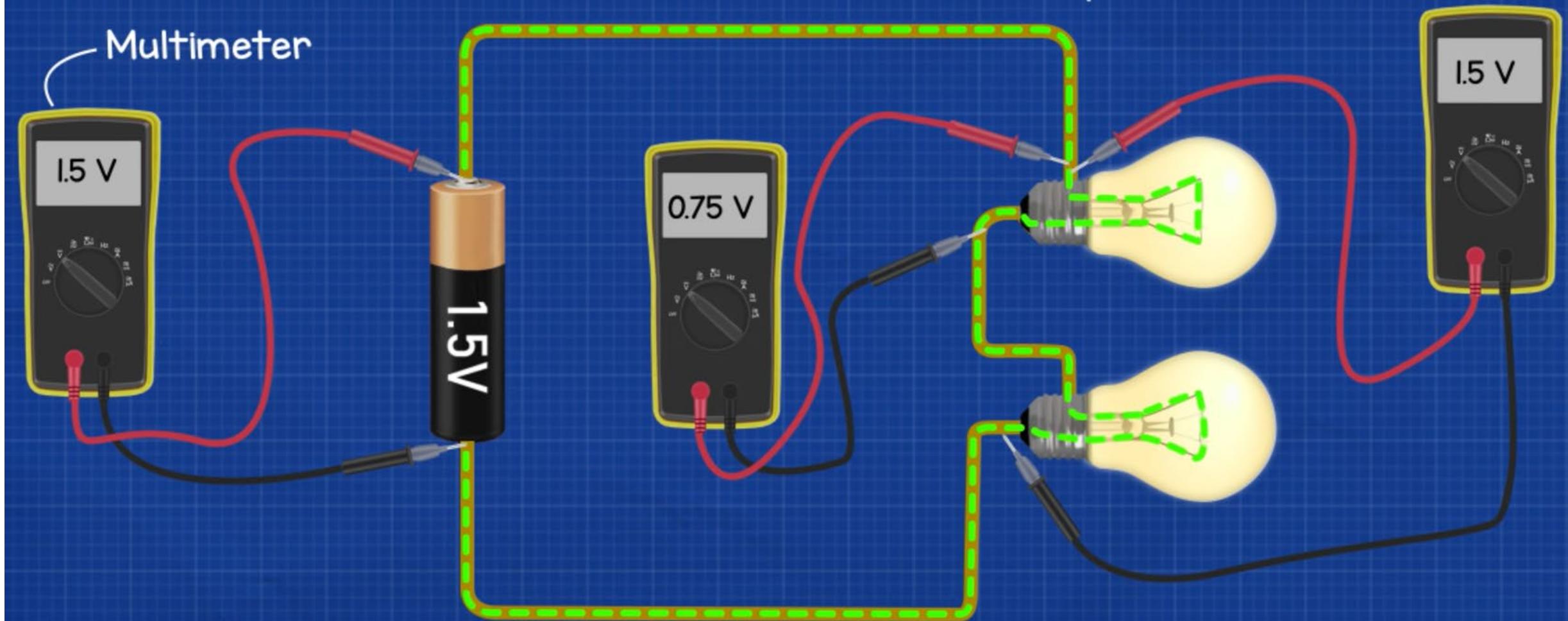
# Voltage Measurement

Single battery circuit, we measure 1.5V across both battery and lamp



# Voltage Measurement

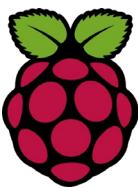
Single battery, dual lamp circuit, we measure 1.5V across the battery  
1.5V across the two lamps  
0.75V across an individual lamp



# Voltage Measurement

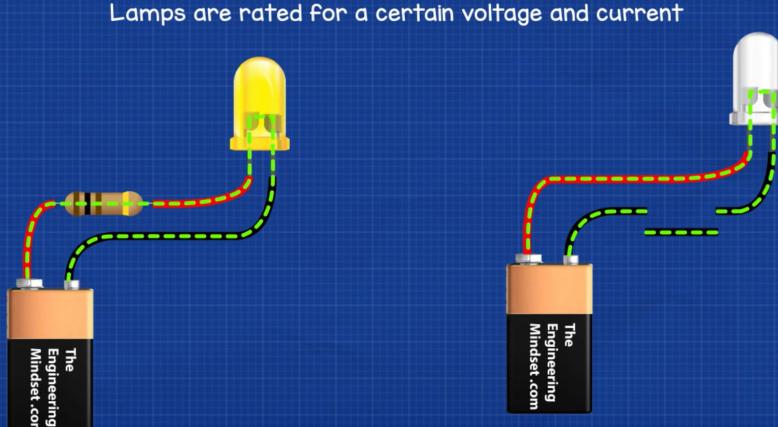


As voltage decreases so does the brightness because there is less pressure to force electrons so less light is produced

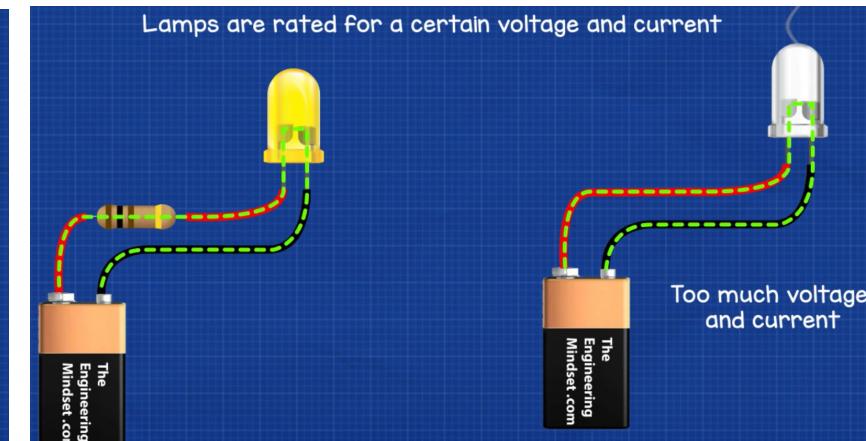


# Direct Voltage

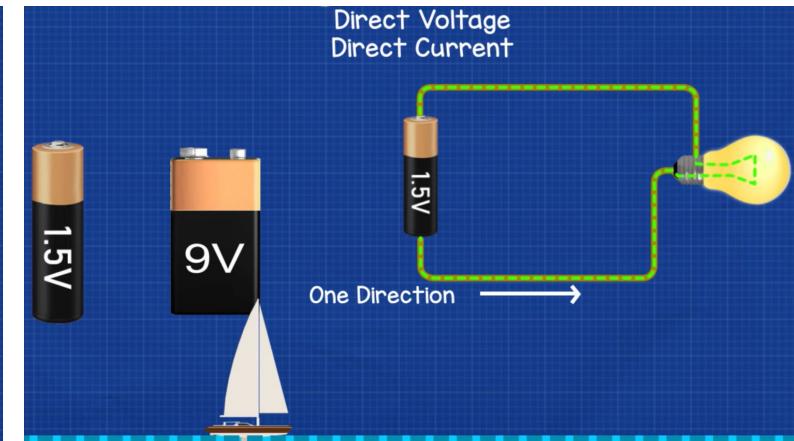
Lamps are rated for a certain voltage and current



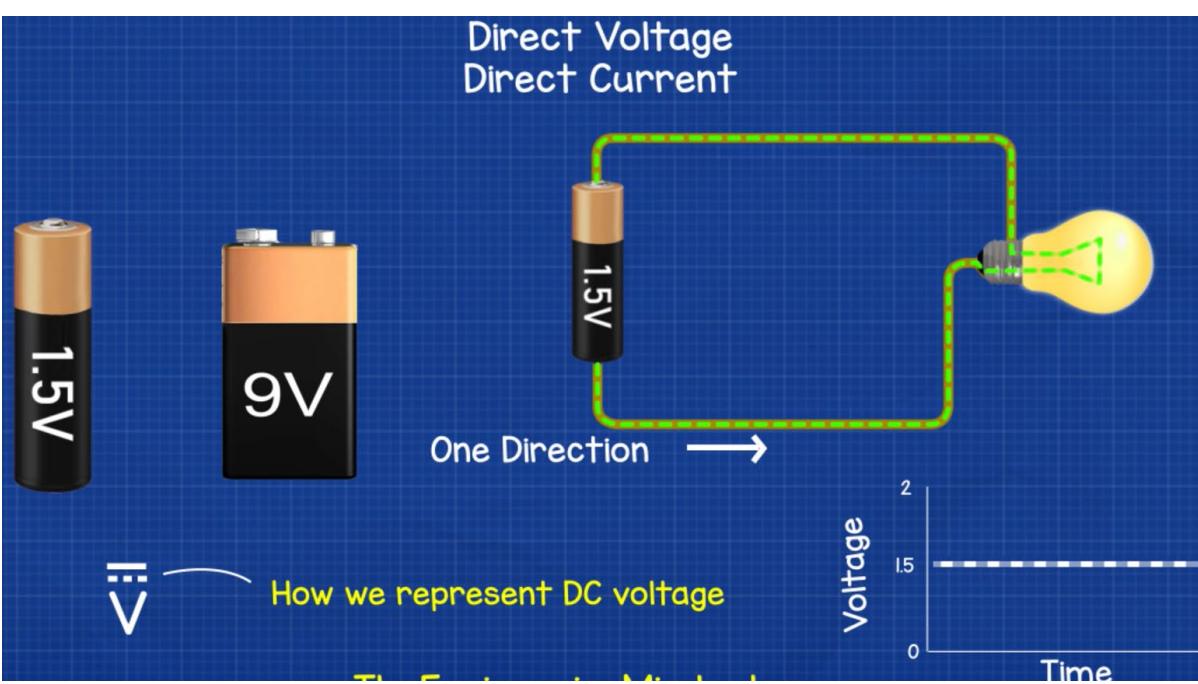
Lamps are rated for a certain voltage and current



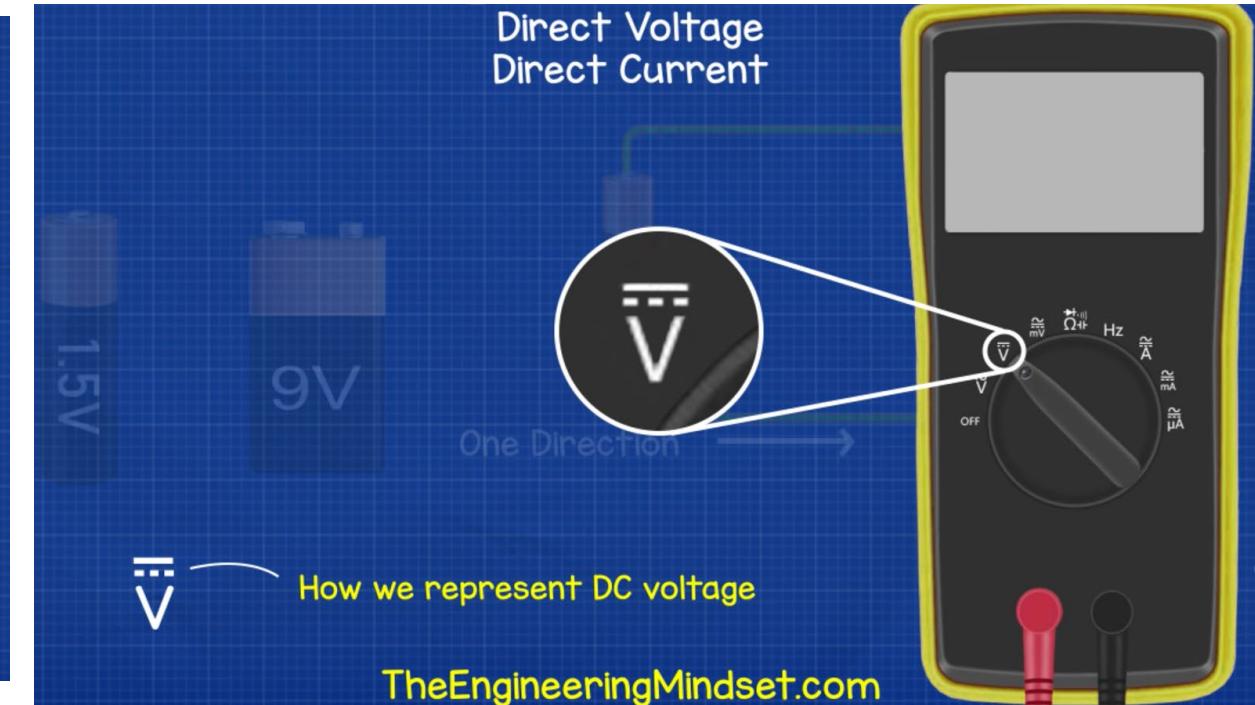
Direct Voltage  
Direct Current

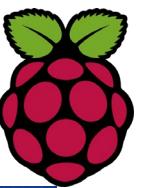


Direct Voltage  
Direct Current

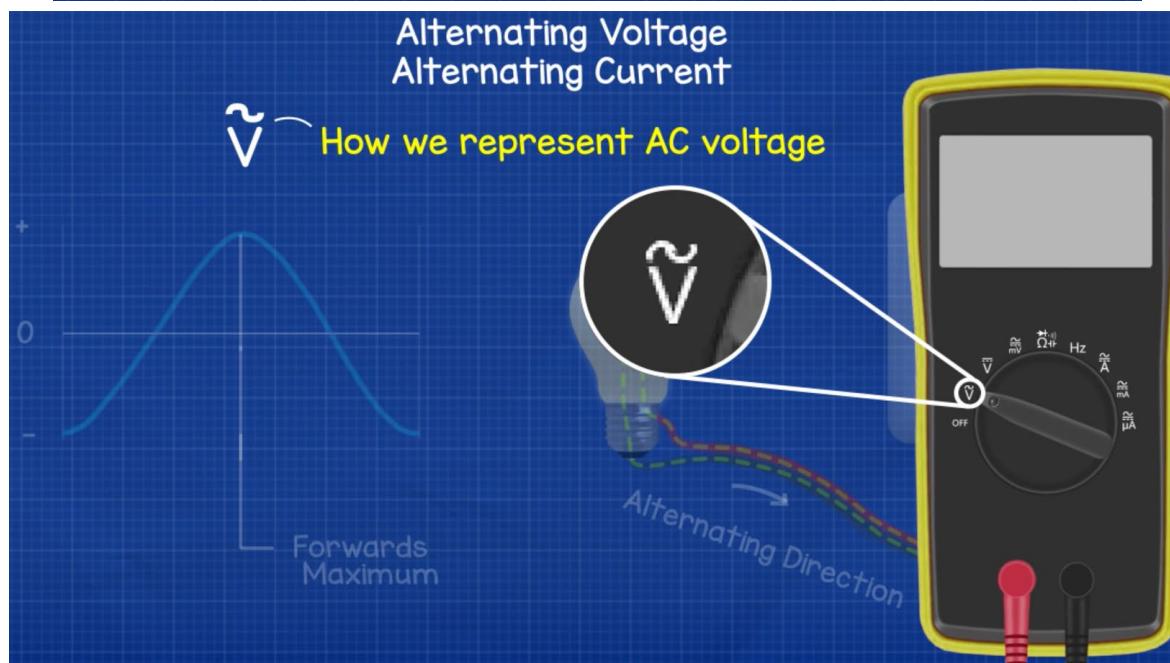
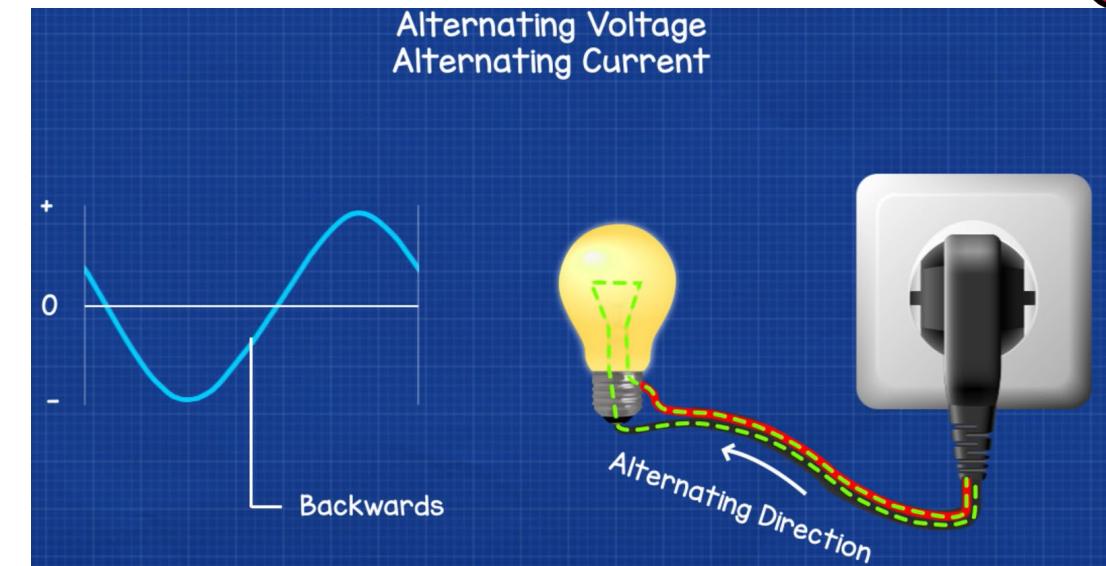
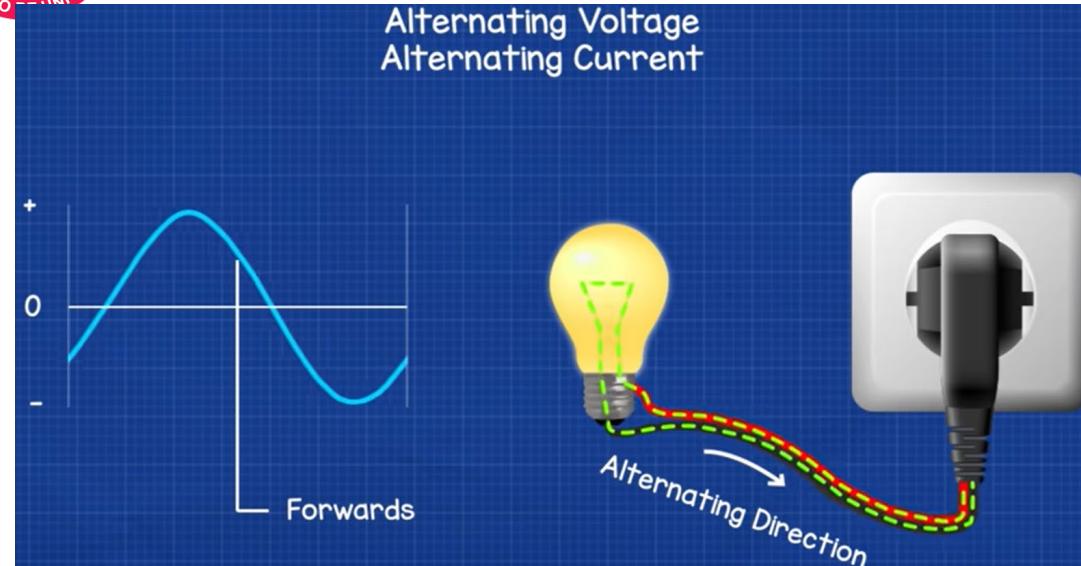


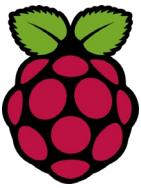
Direct Voltage  
Direct Current



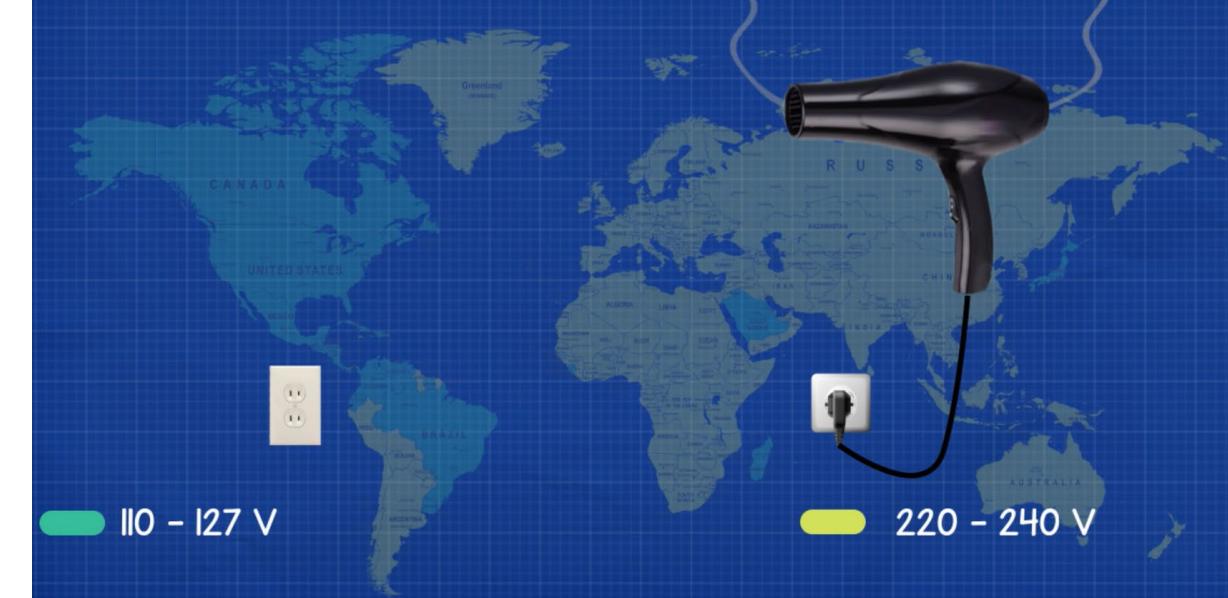


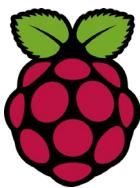
# Alternating Voltage





# Voltage in Different Region





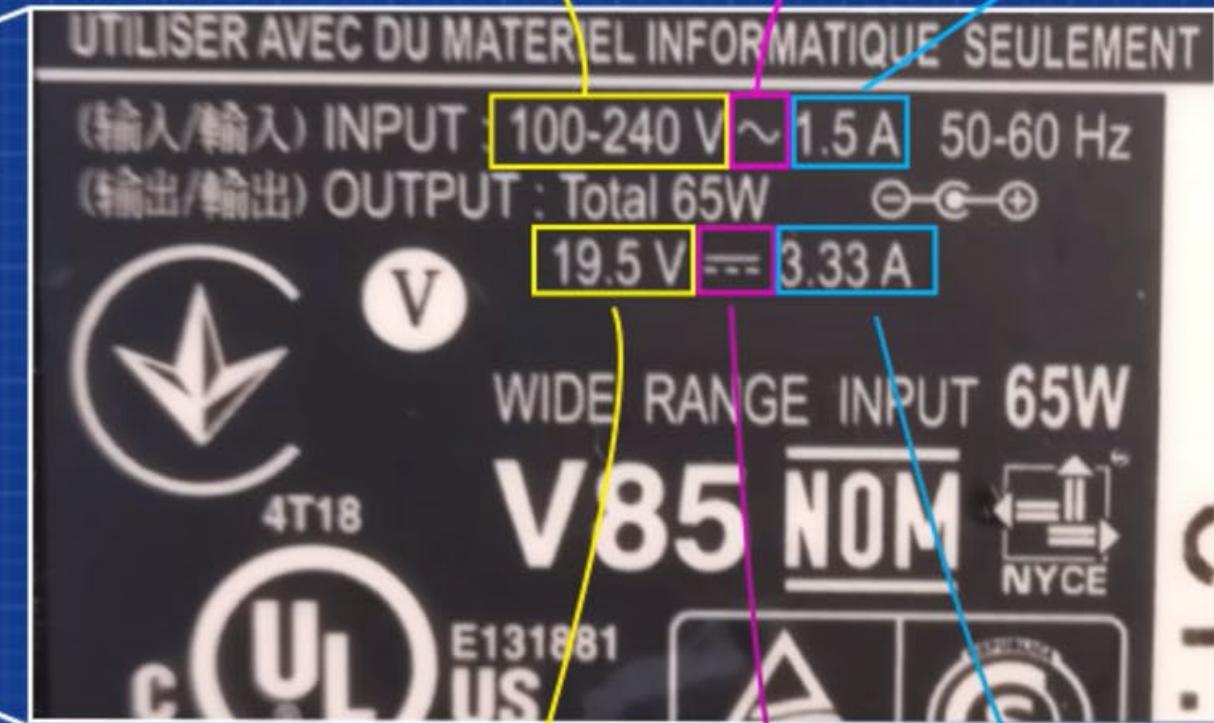
# Voltage

Manufacturer Label



Laptop Charger

Input: 100-240 V AC 1.5 Amps



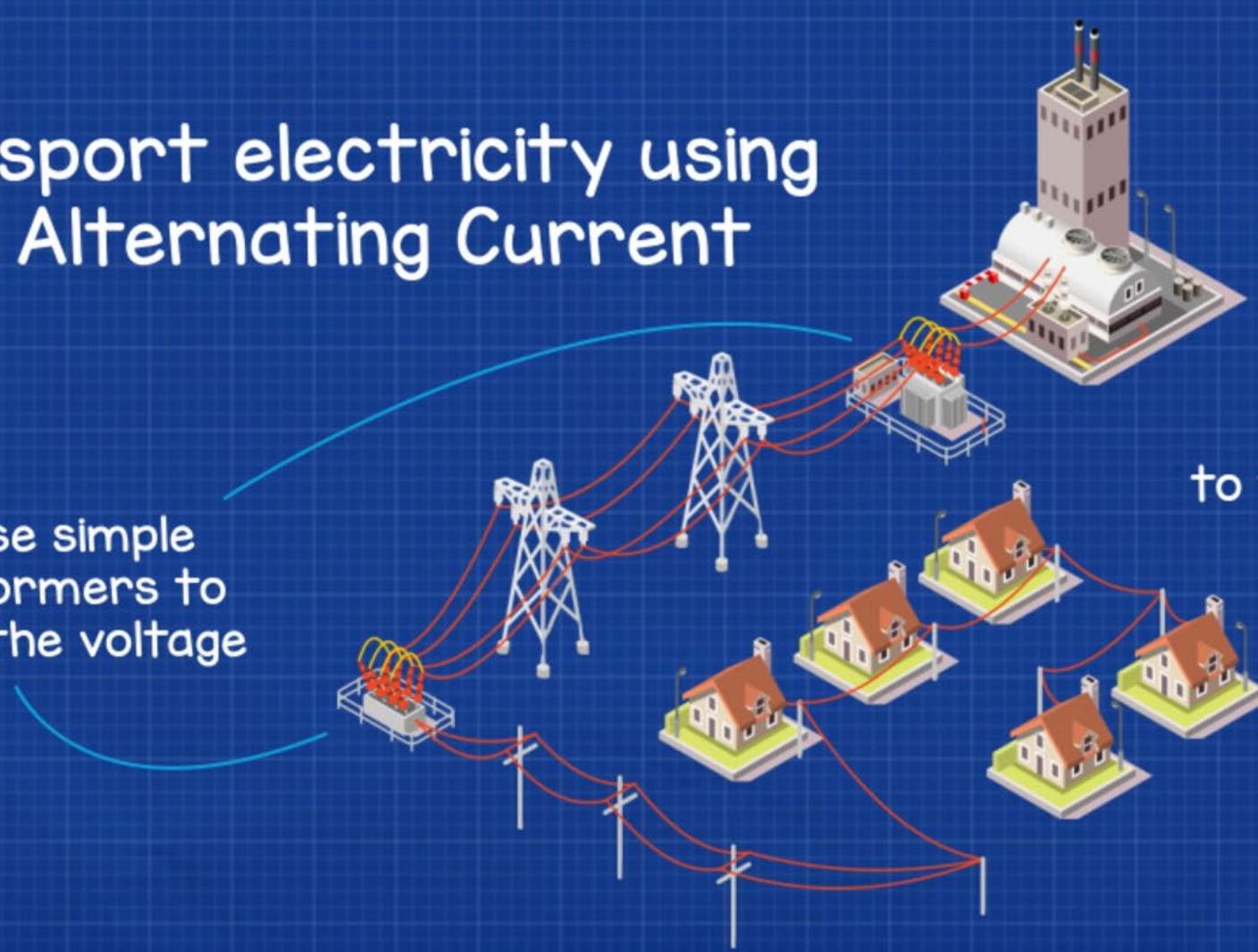
Output: 19.5 V DC 3.33 Amps

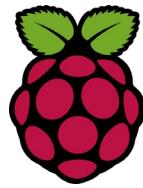
# Alternating Current

Transport electricity using  
AC Alternating Current

Can use simple  
transformers to  
change the voltage

AC is more efficient  
to transport over long distance





# Direct Current

We use DC Direct Current for electronic devices

DC is less complex  
allows circuit boards and devices to be more compact



# Direct Current

Control Circuit Board



DC Direct Current



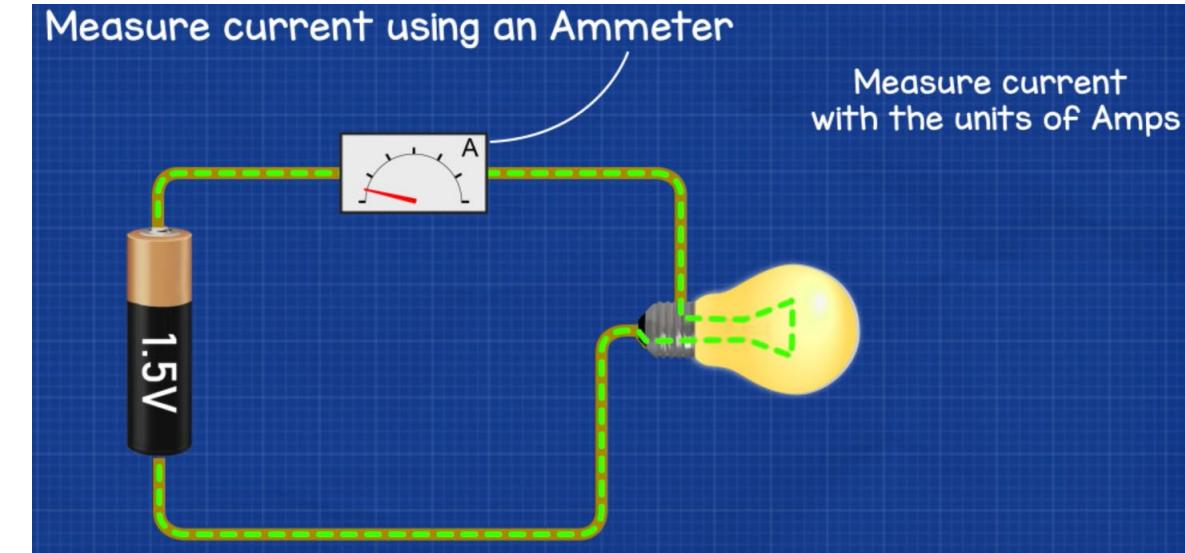
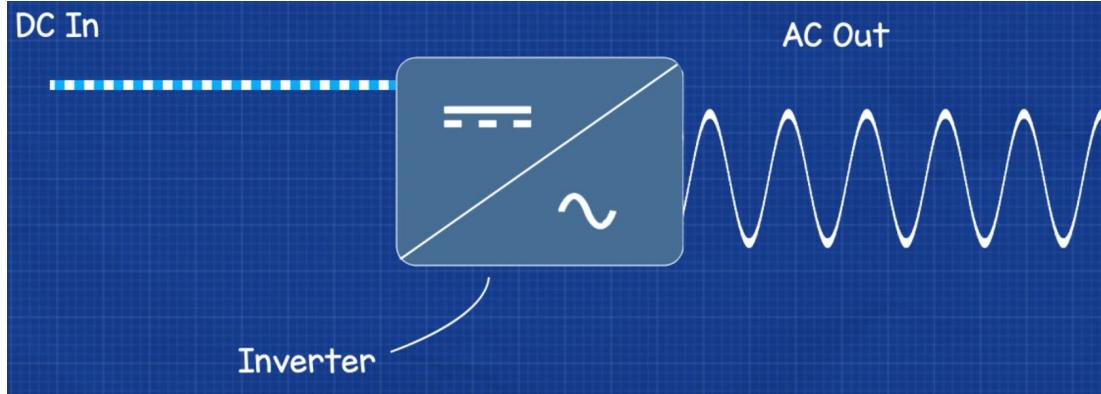
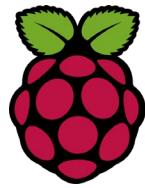
Washing Machine

AC induction motor

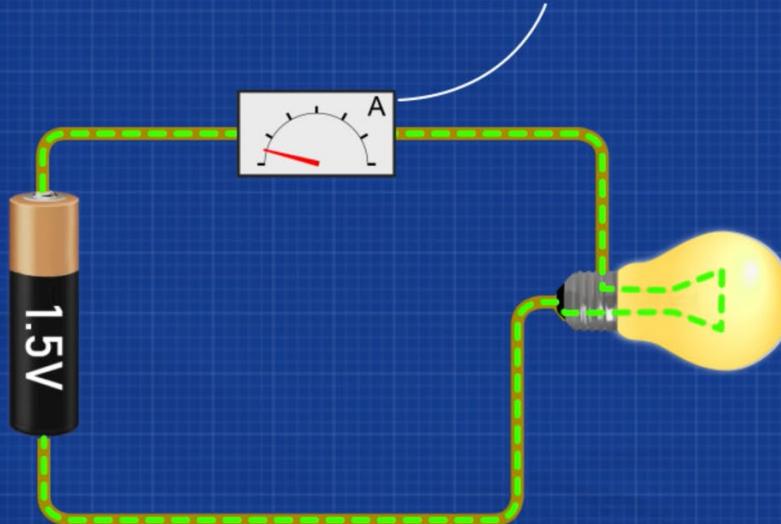




# Current Measurement



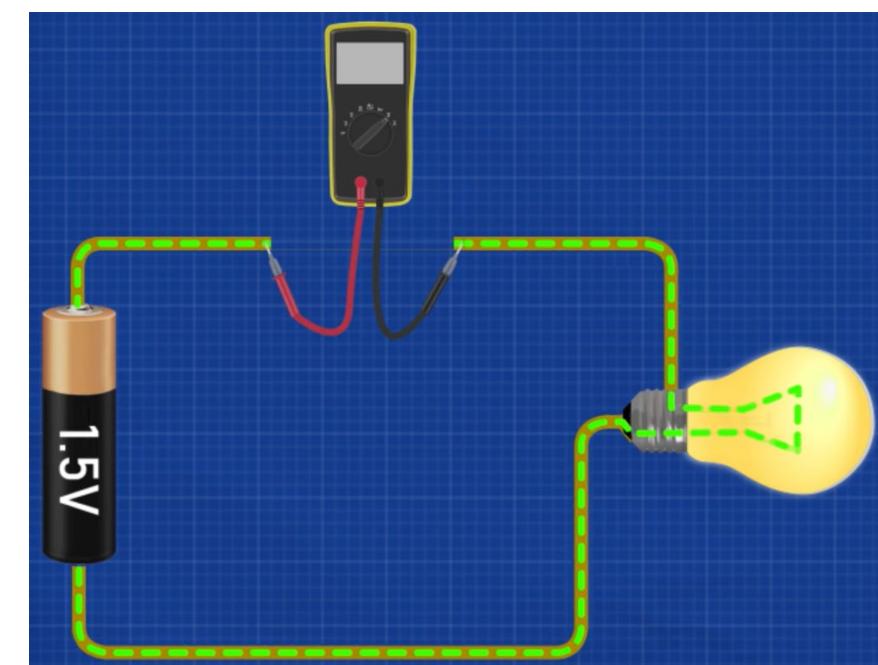
Current must pass through the Ammeter

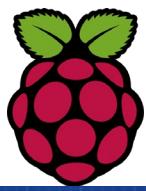


Water must pass through  
a water meter

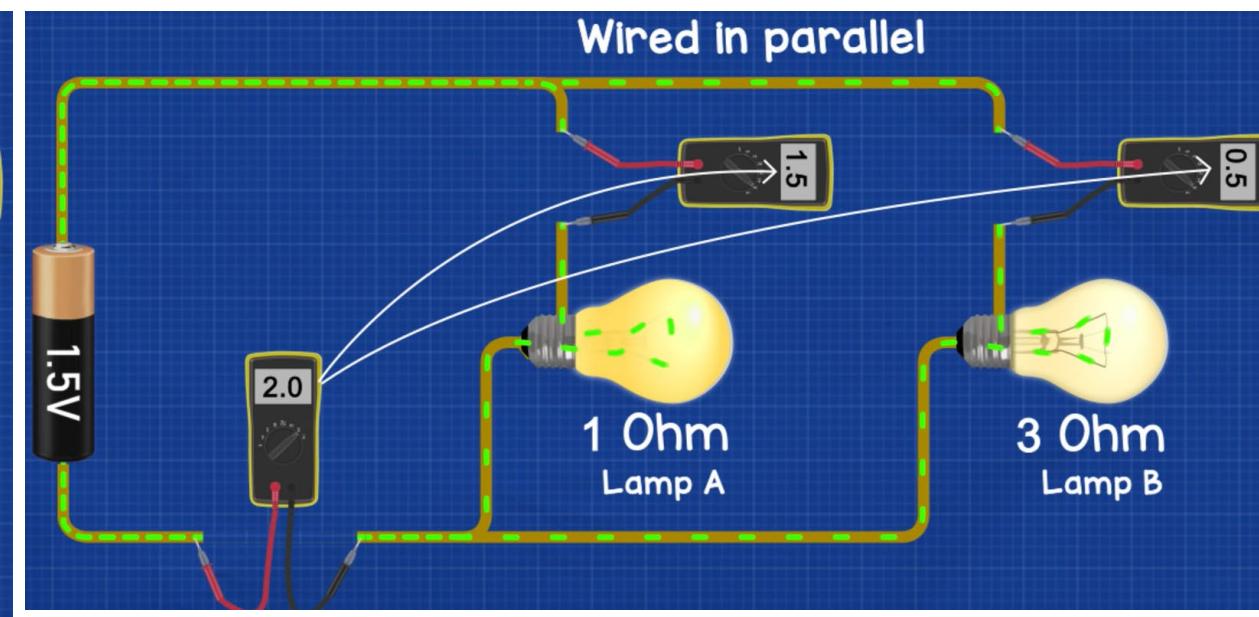
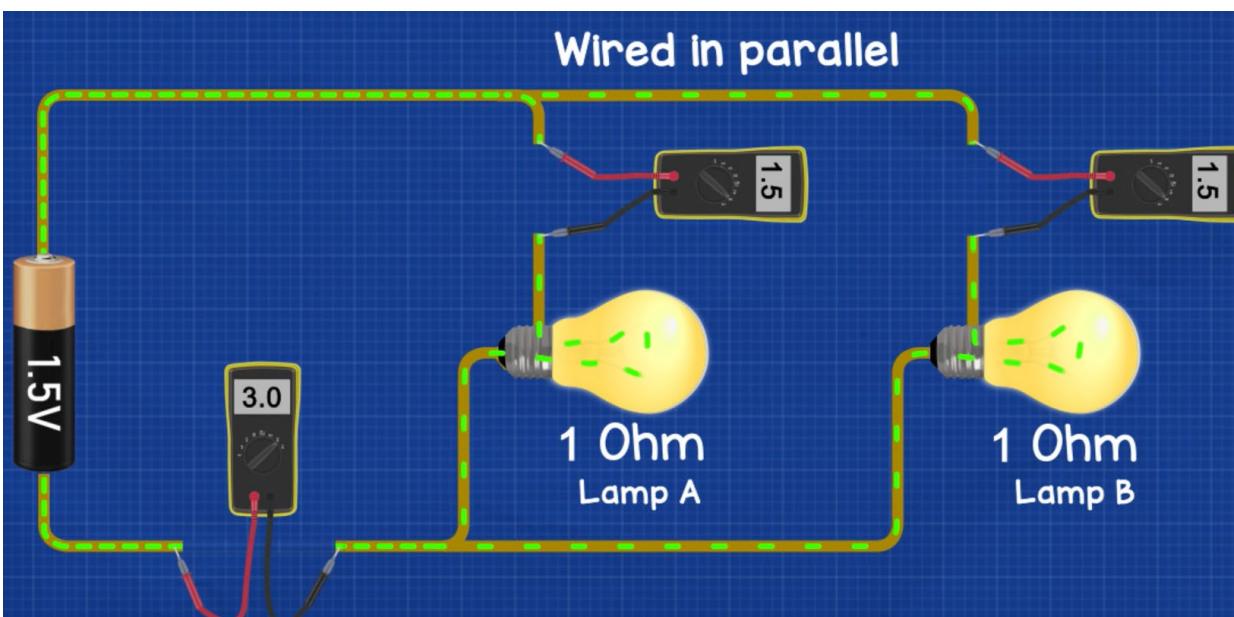
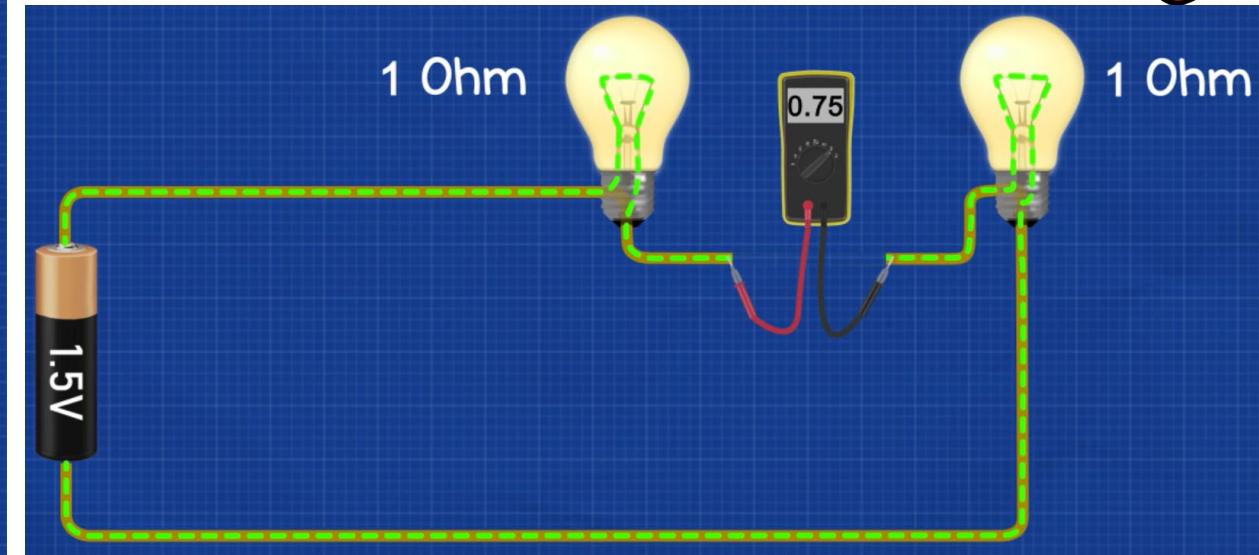
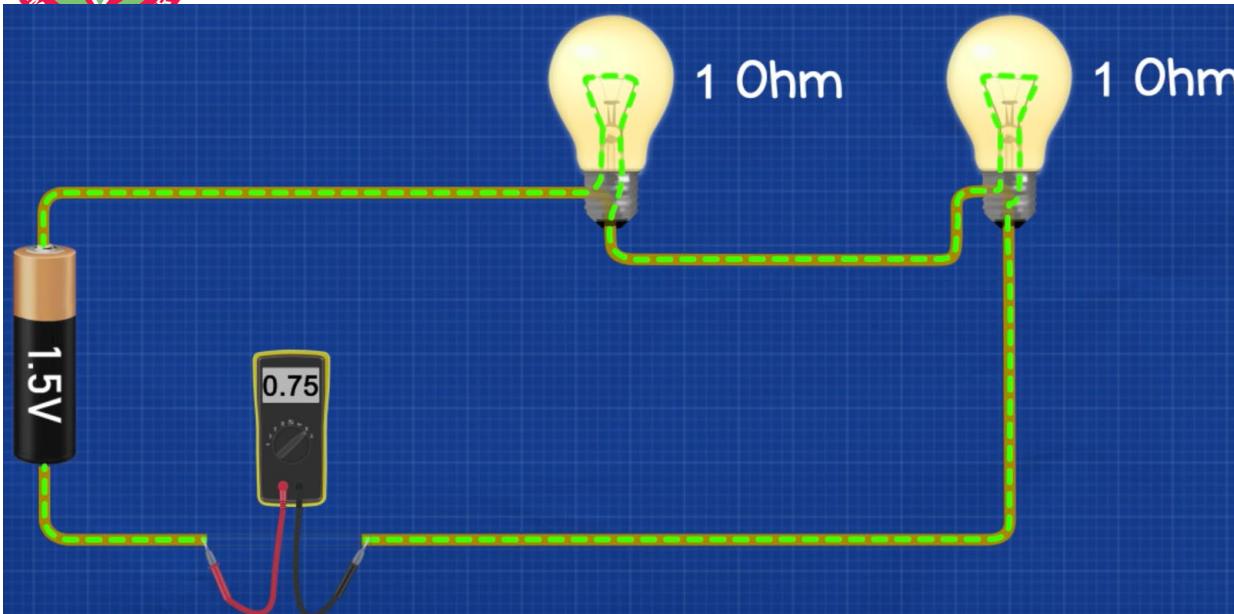


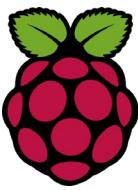
Water meter measures water consumption



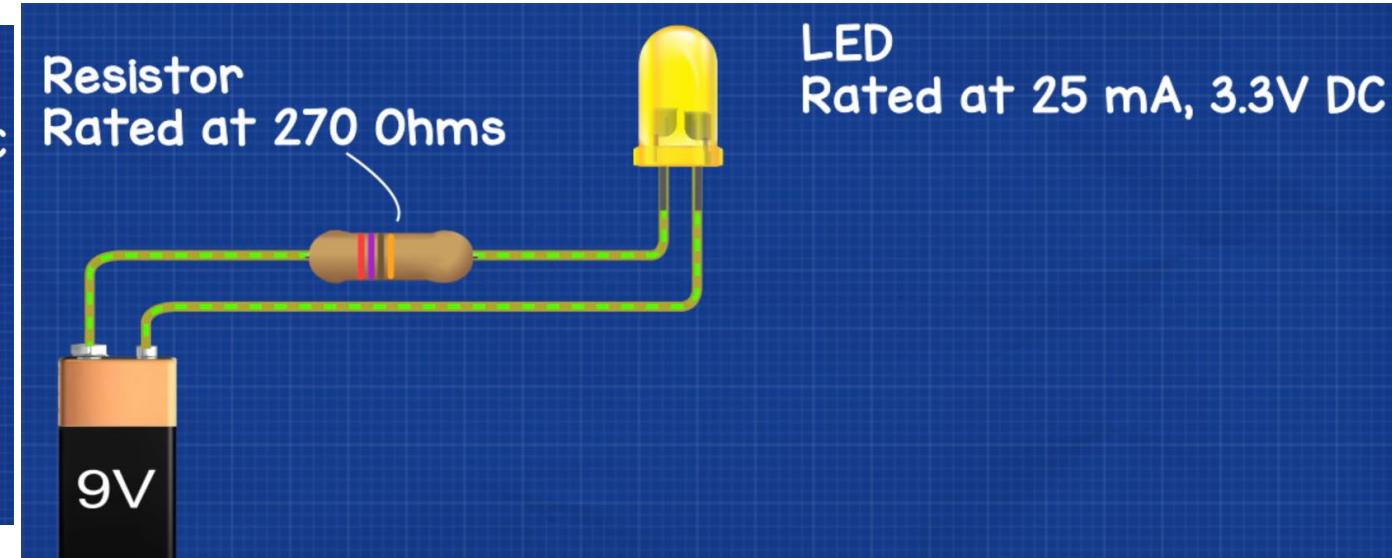
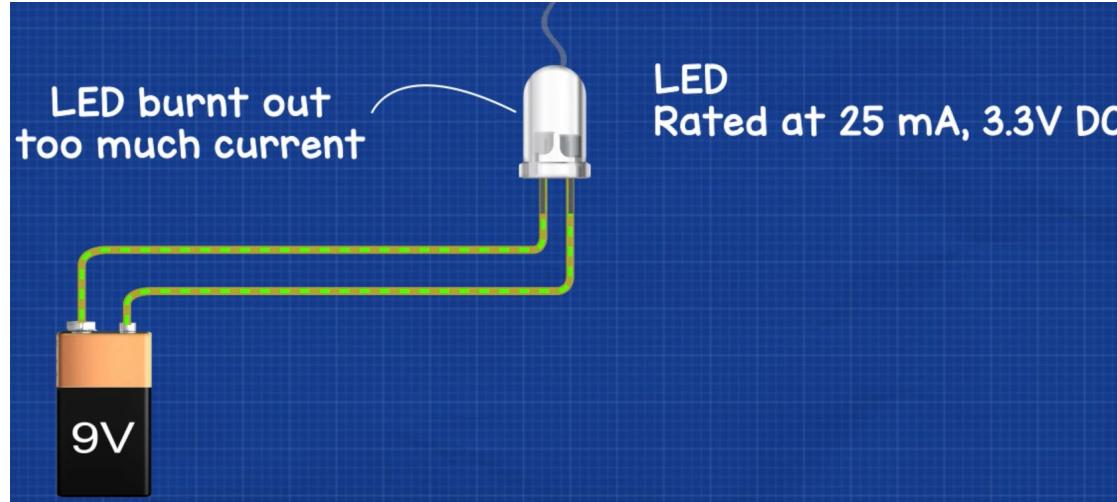


# Current in Series and Parallel Circuit

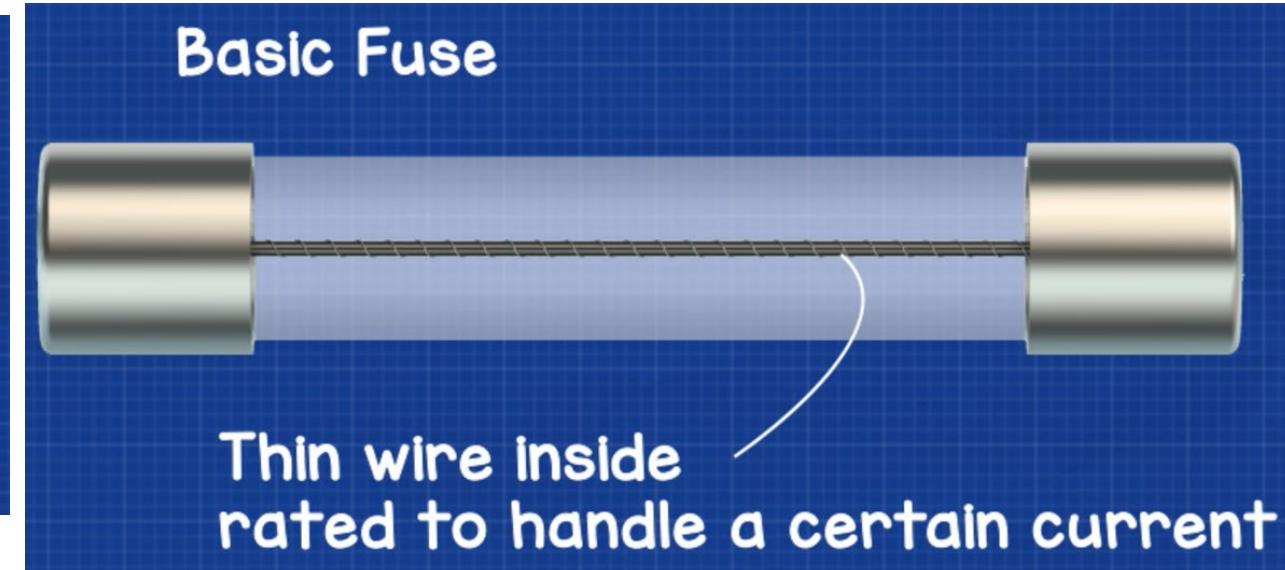




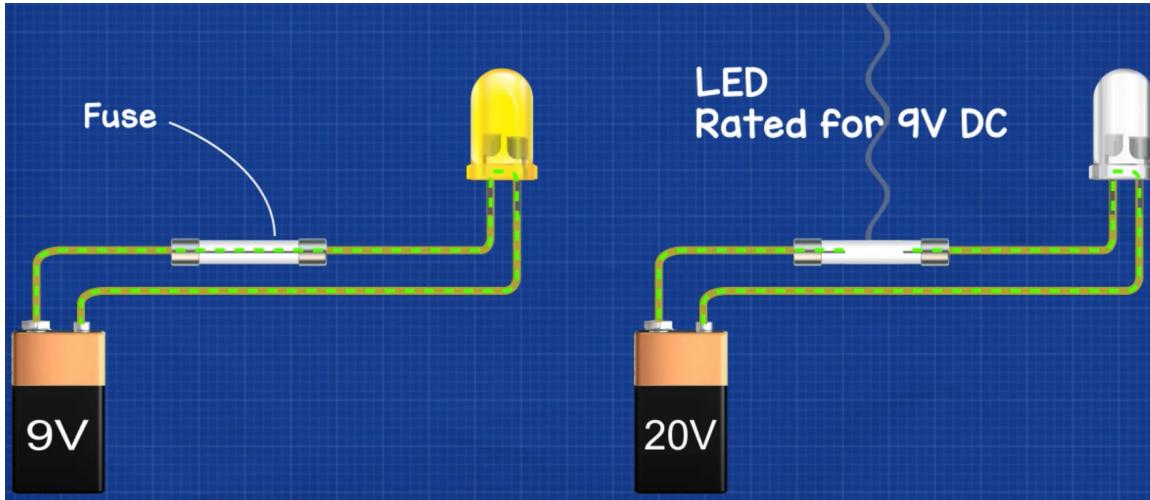
# Current and limiting Resistor



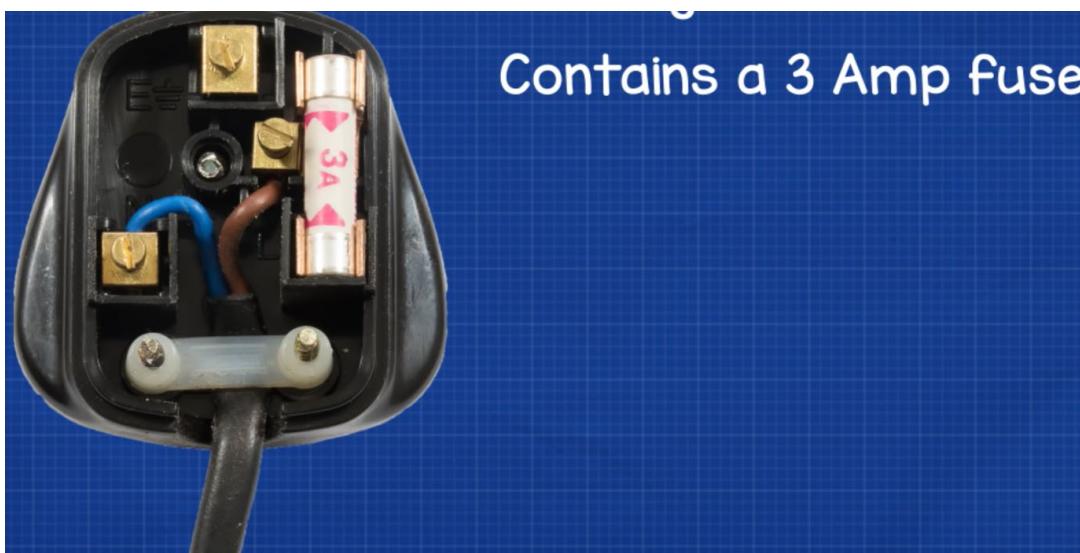
**Fuse - safety device that melts and breaks an electric circuit if the current exceeds a safe level**



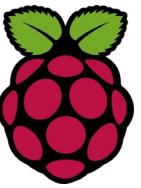
# Fuse



Circuit Board Fuses

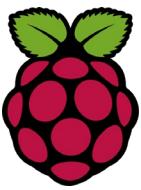


Consumer Unit

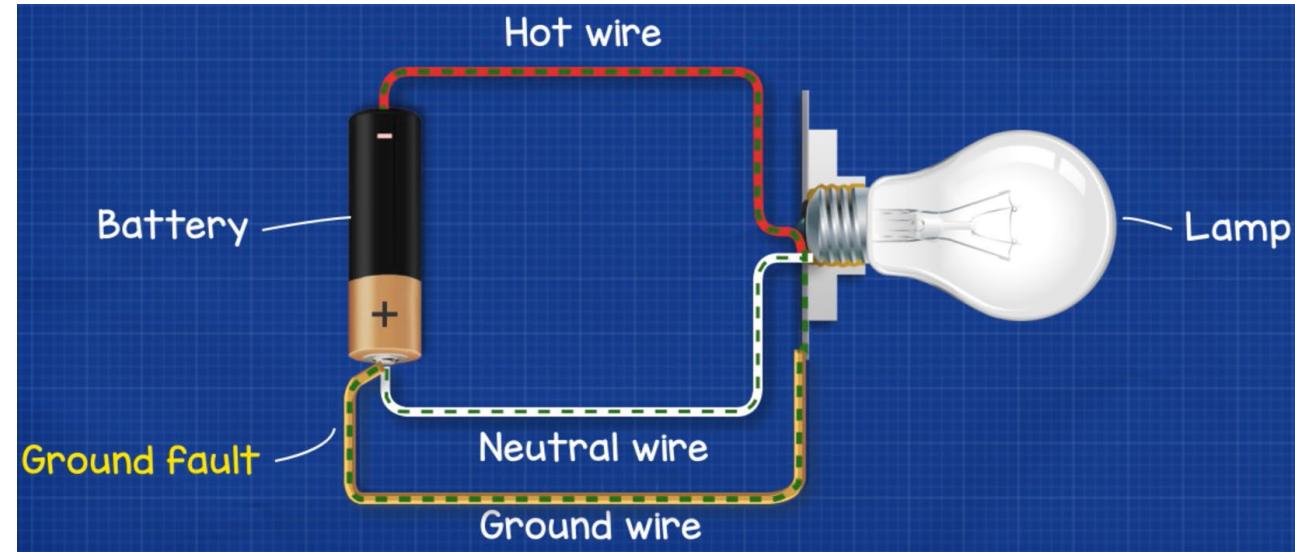
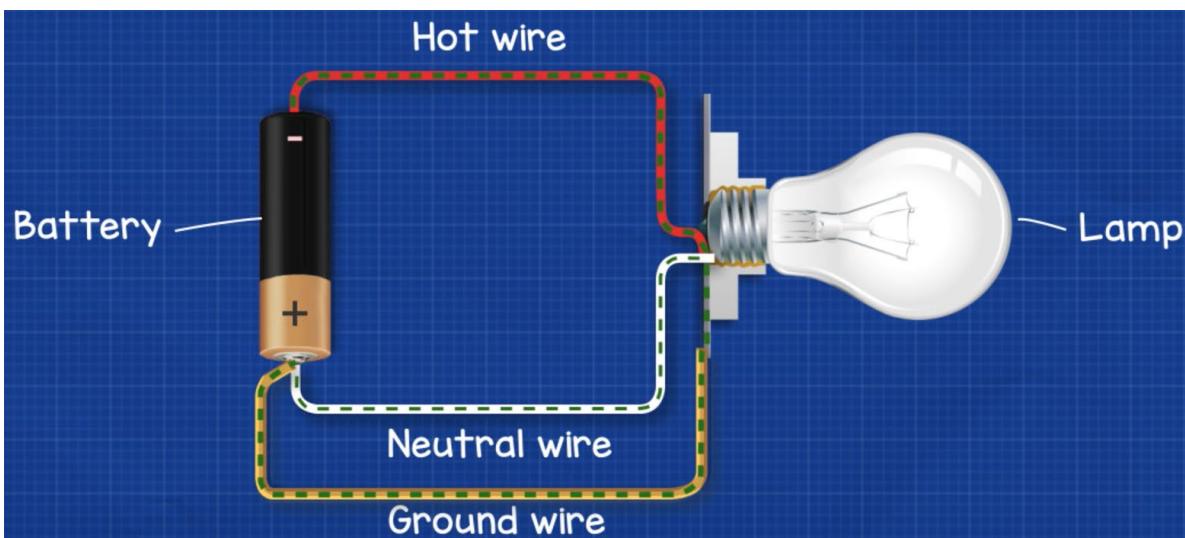
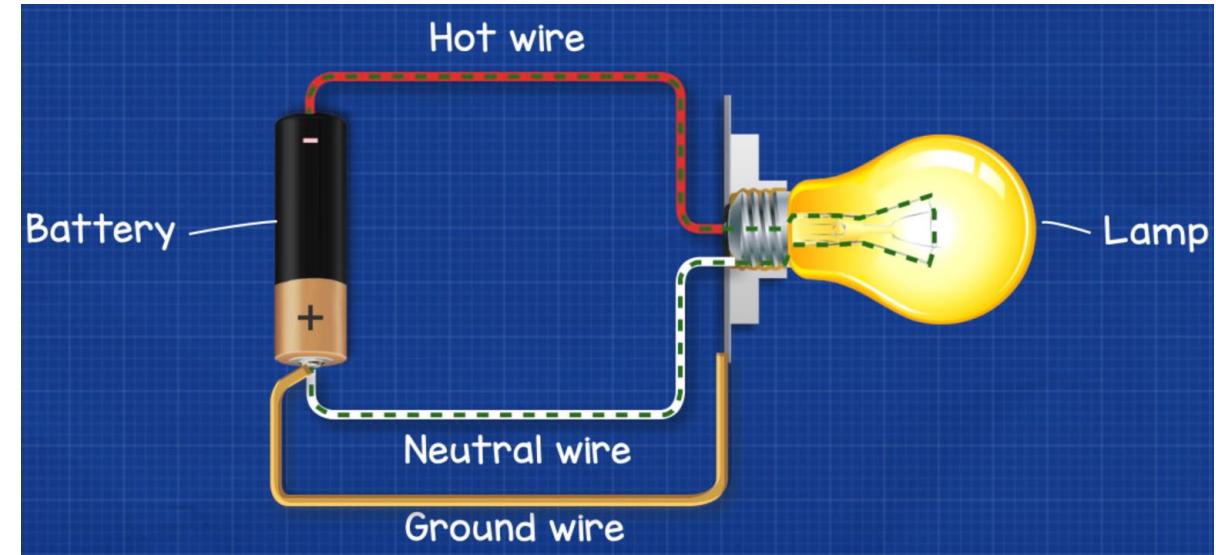
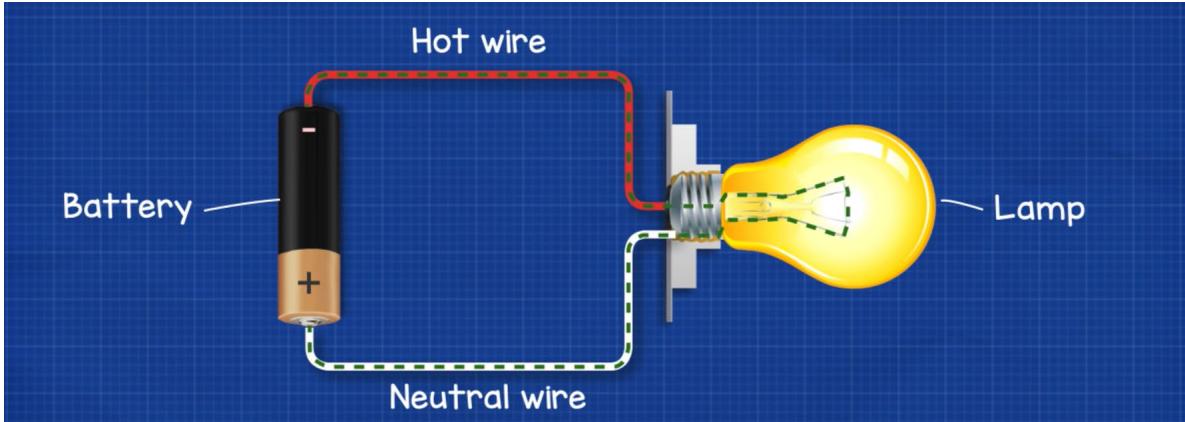


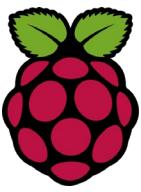
## Remember these Points

- 1) Electricity will only flow in a complete circuit. If you come into contact with an electrical conductor, your body might complete the circuit.
- 2) Electricity always tries to return to its source.
- 3) Electricity takes all available paths to complete a circuit. It takes preference to a path with less resistance and more current will flow in that path.

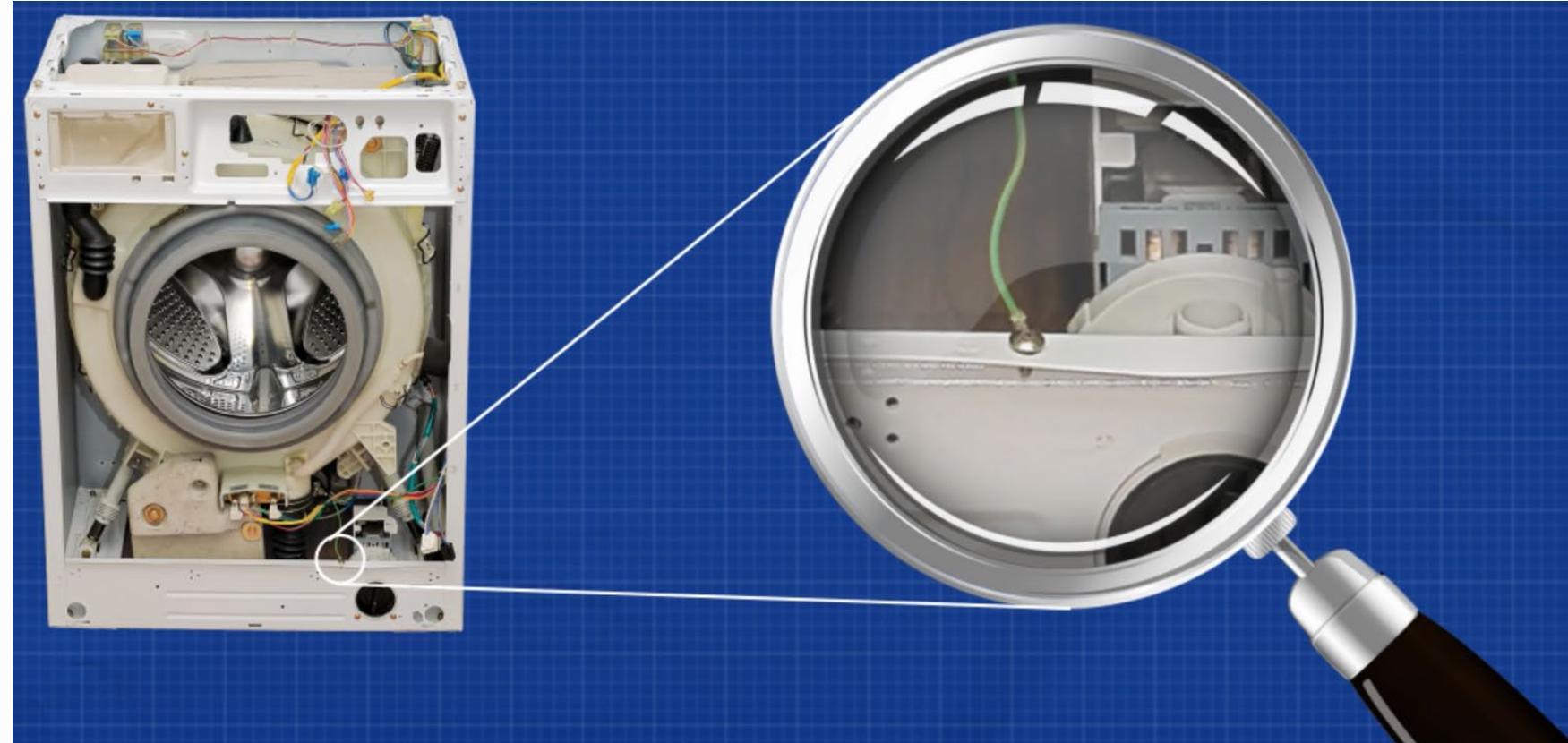
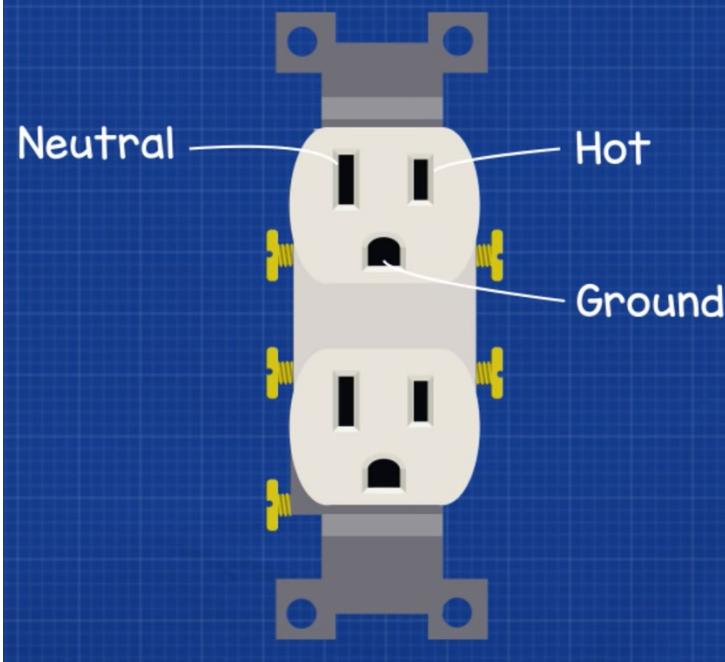


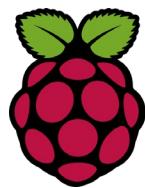
# Ground, Neutral and Hot Wire/Phase





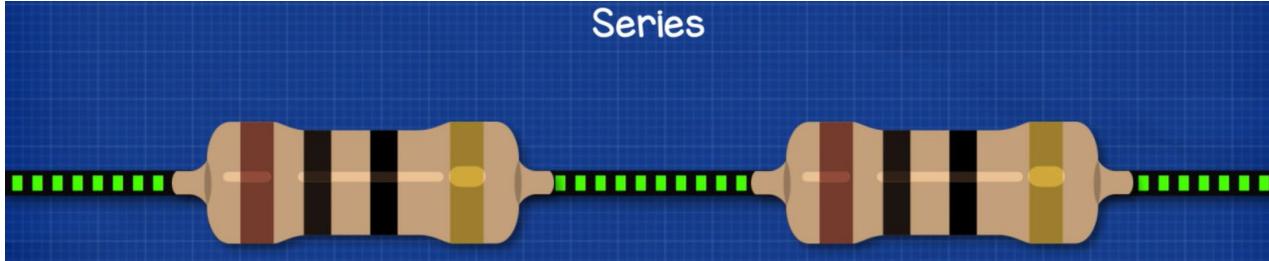
# Ground, Neutral and Hot Wire/Phase





# Resistance in Series

Series

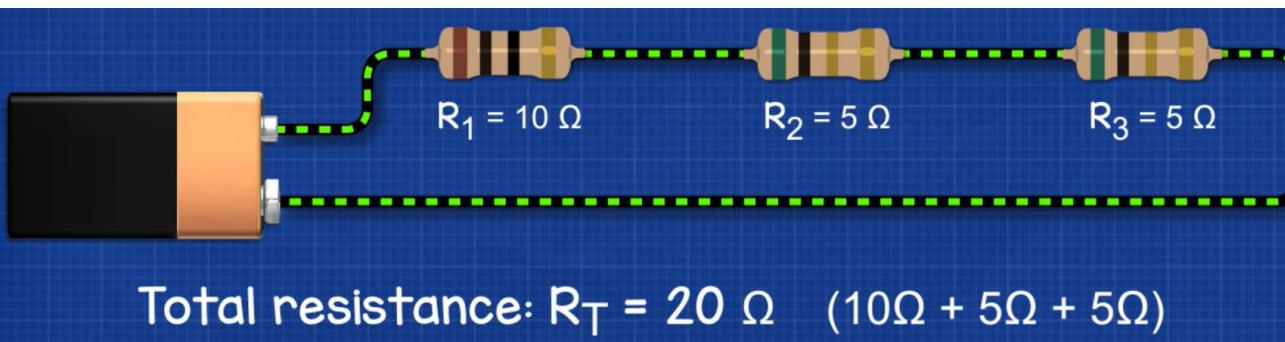
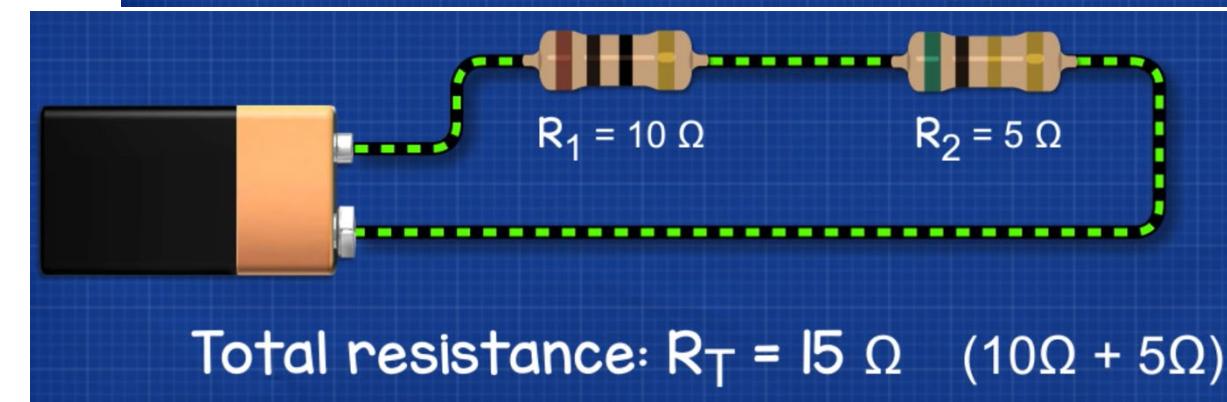
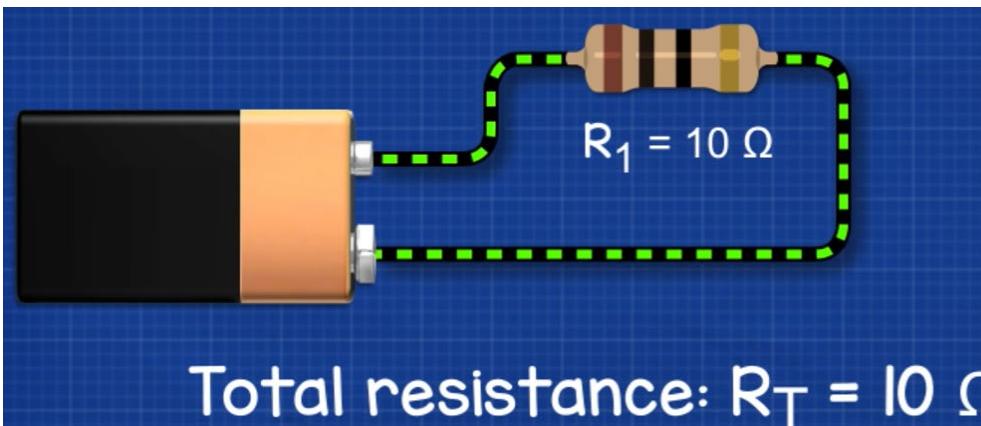


$R_1$

$R_2$

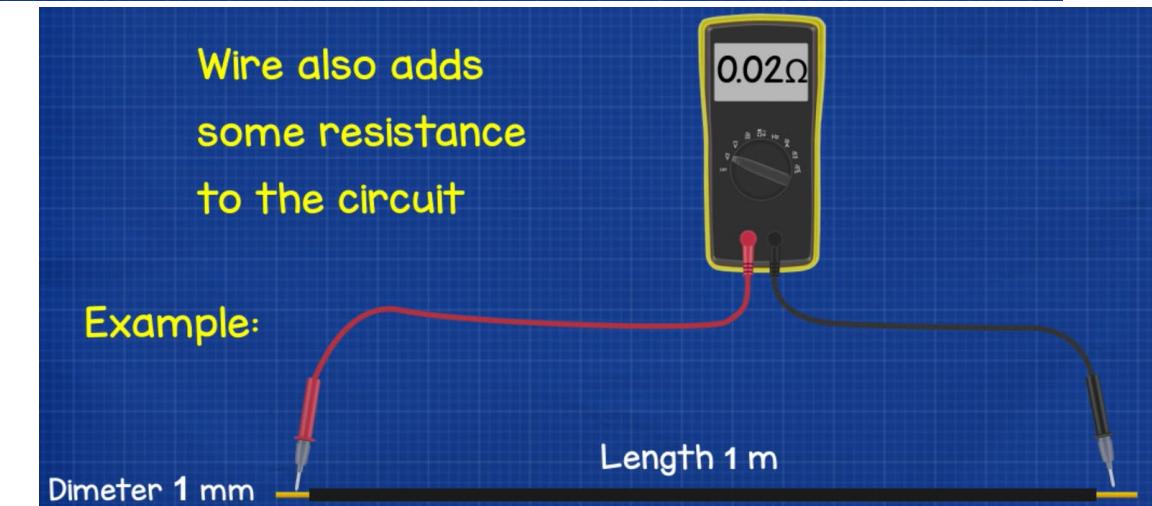
$R_3$

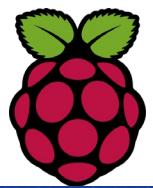
$$\text{Total resistance: } R_T = R_1 + R_2 + R_3 + \dots$$



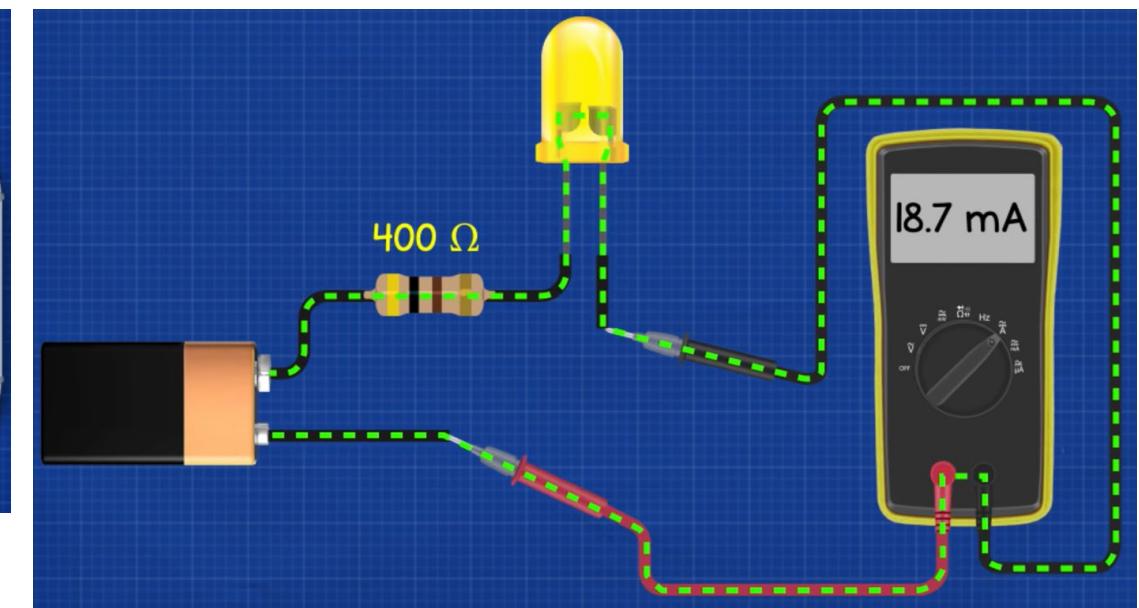
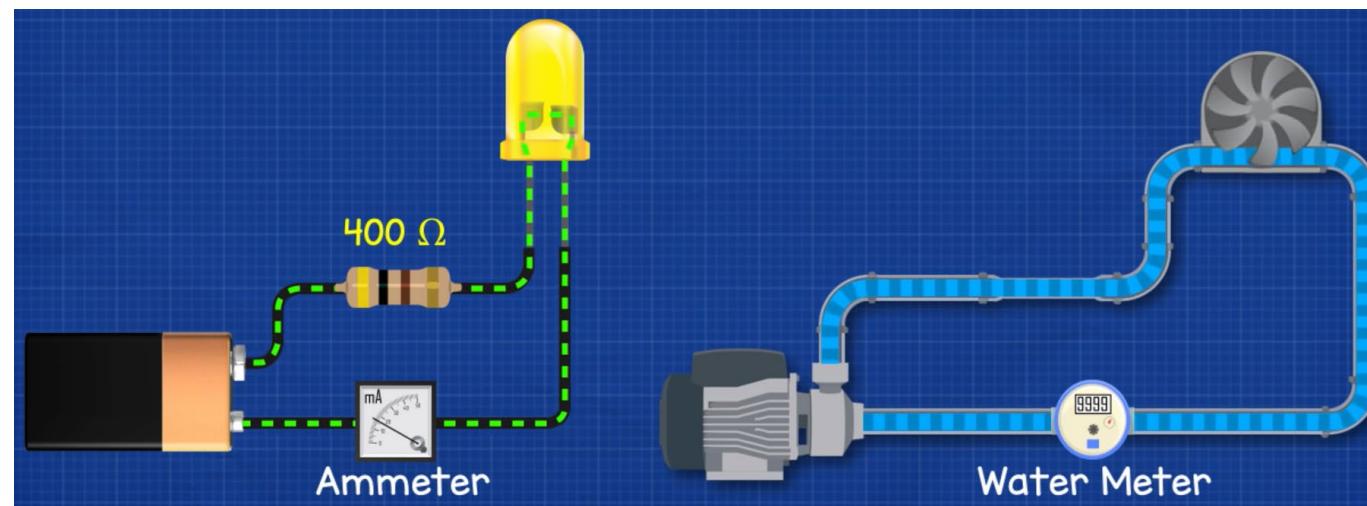
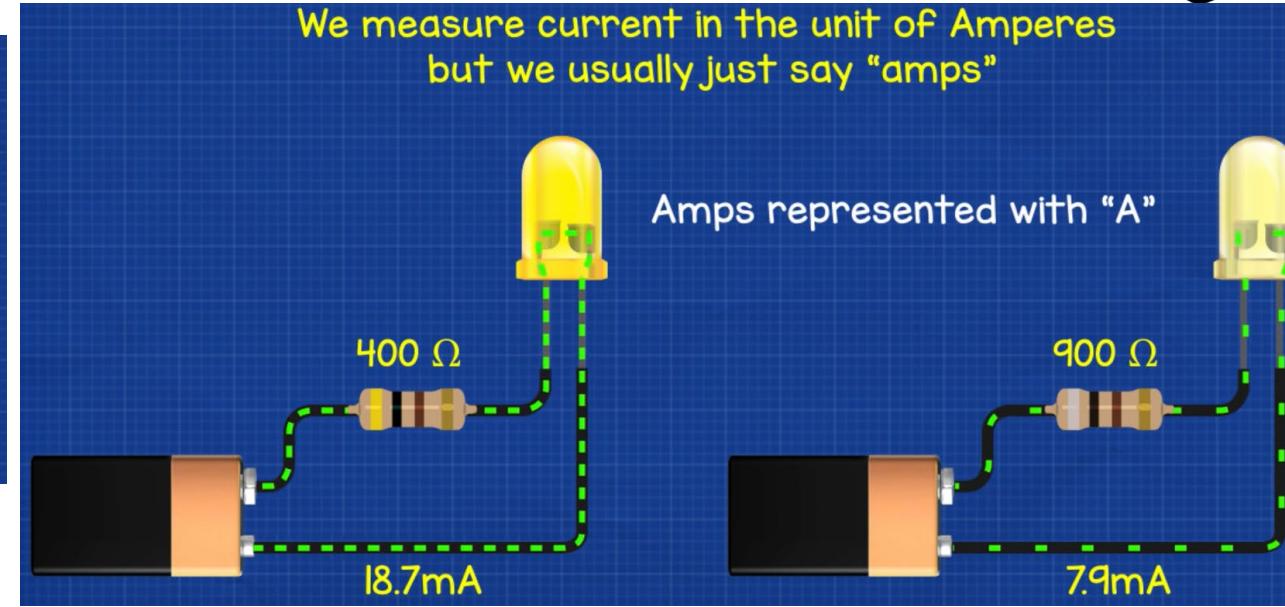
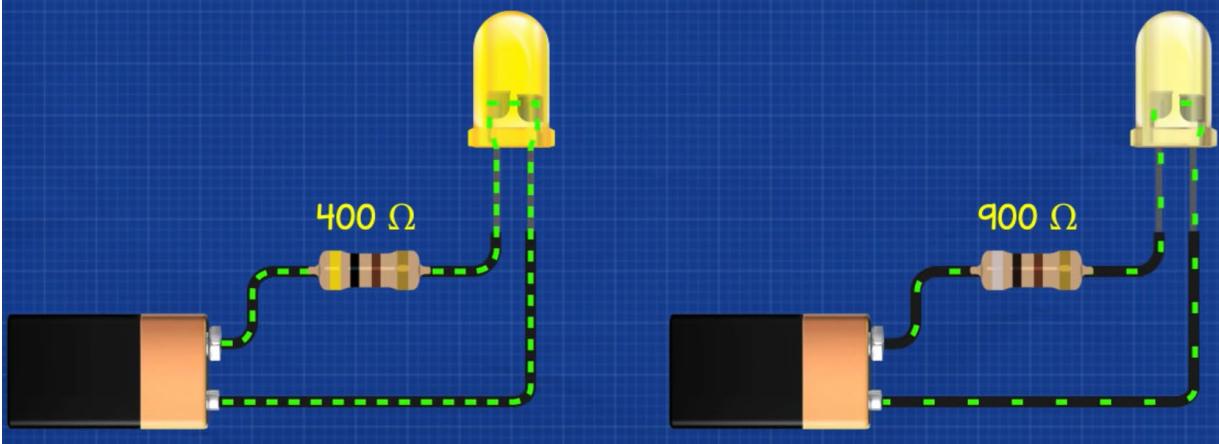
Wire also adds  
some resistance  
to the circuit

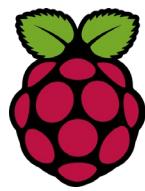
Example:



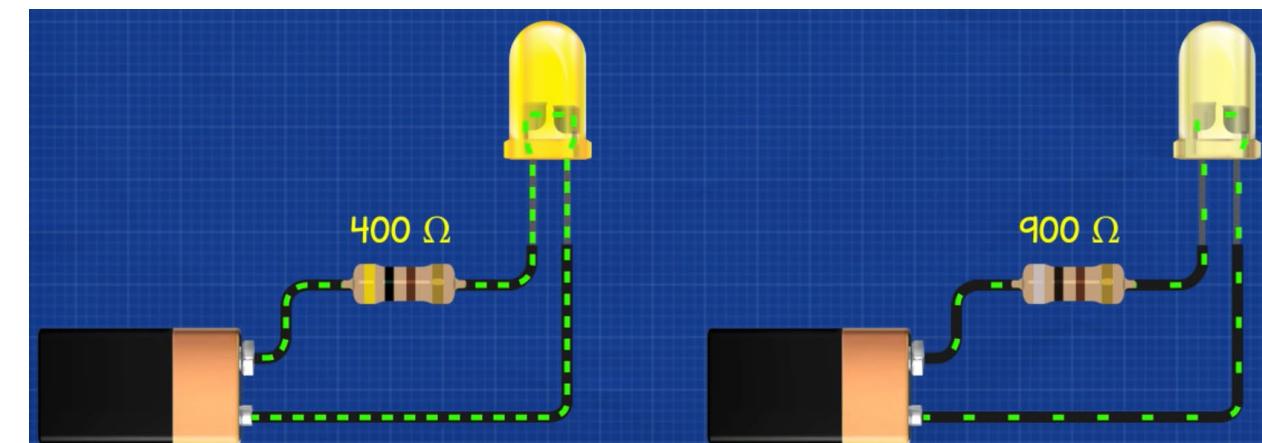
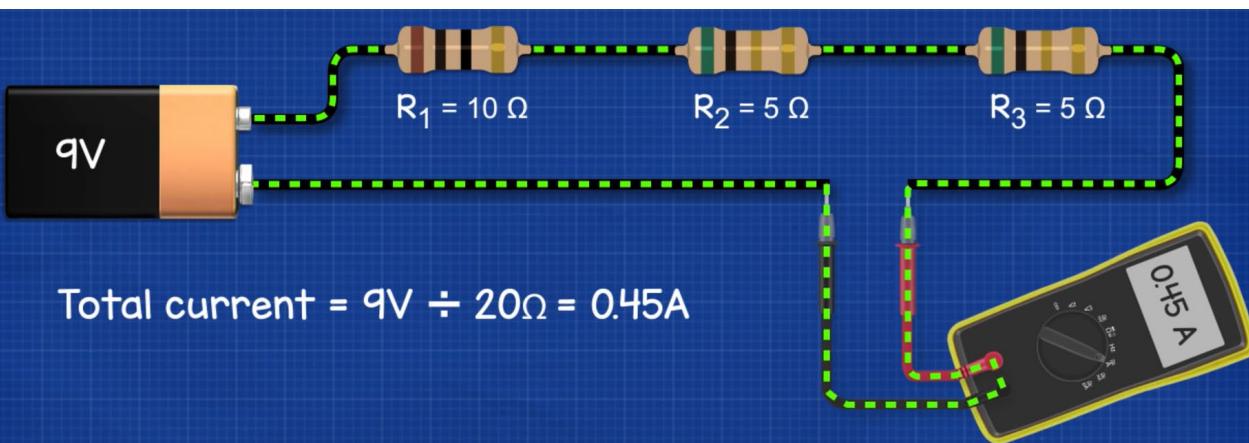
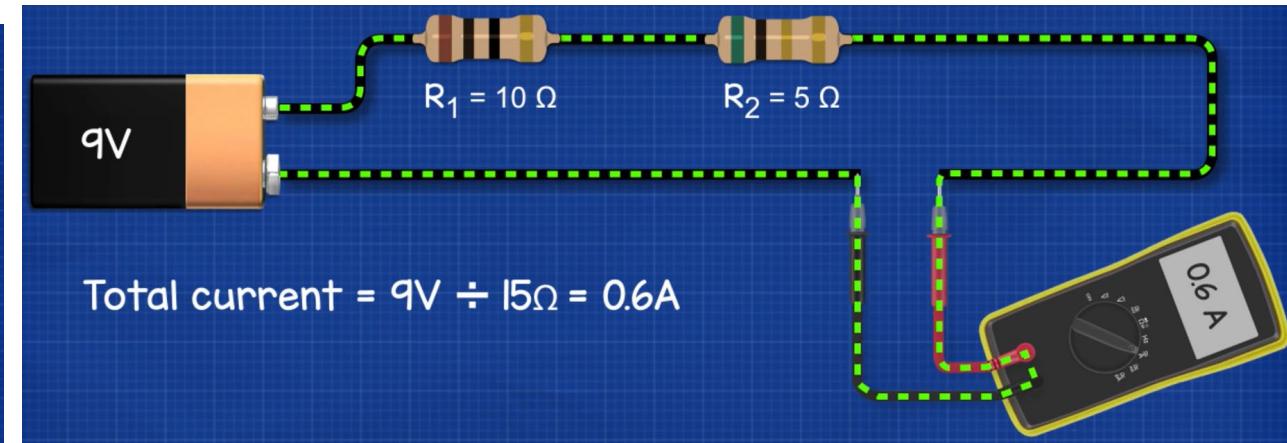
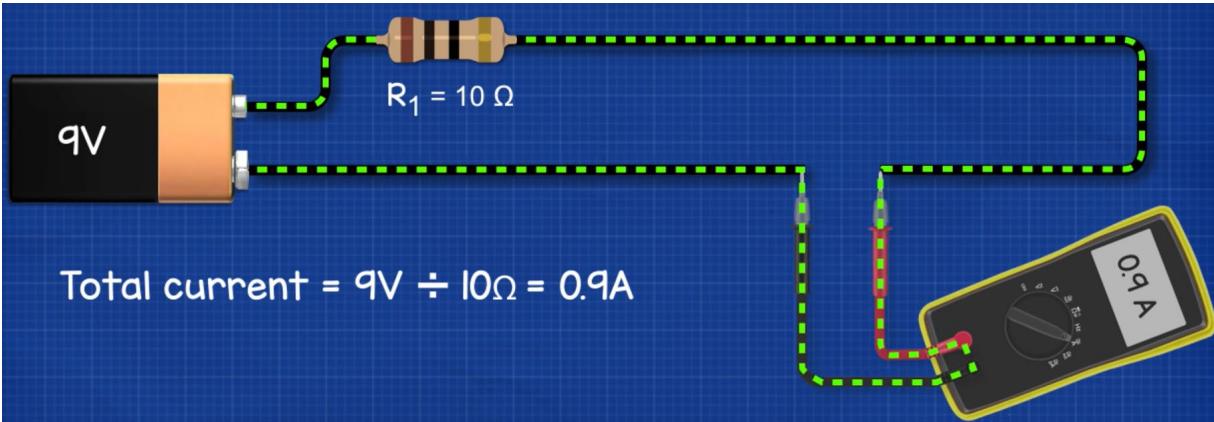


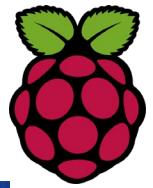
# Resistor Application



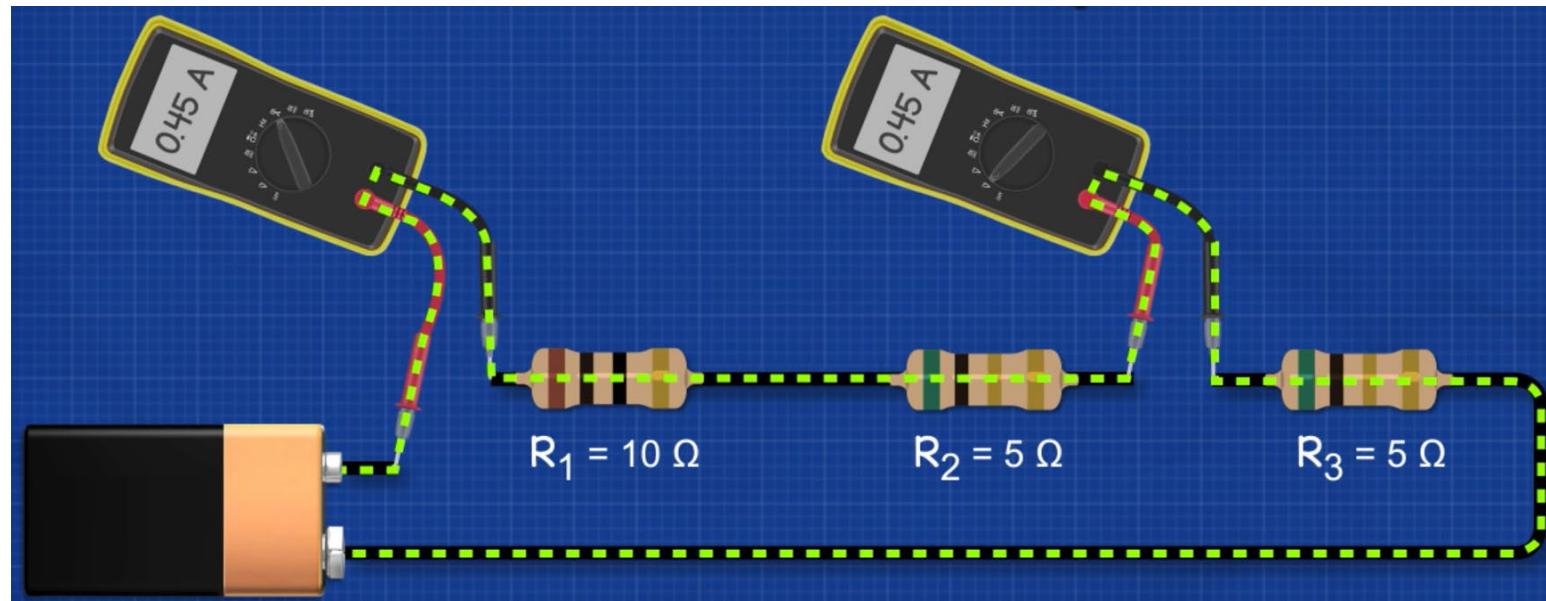
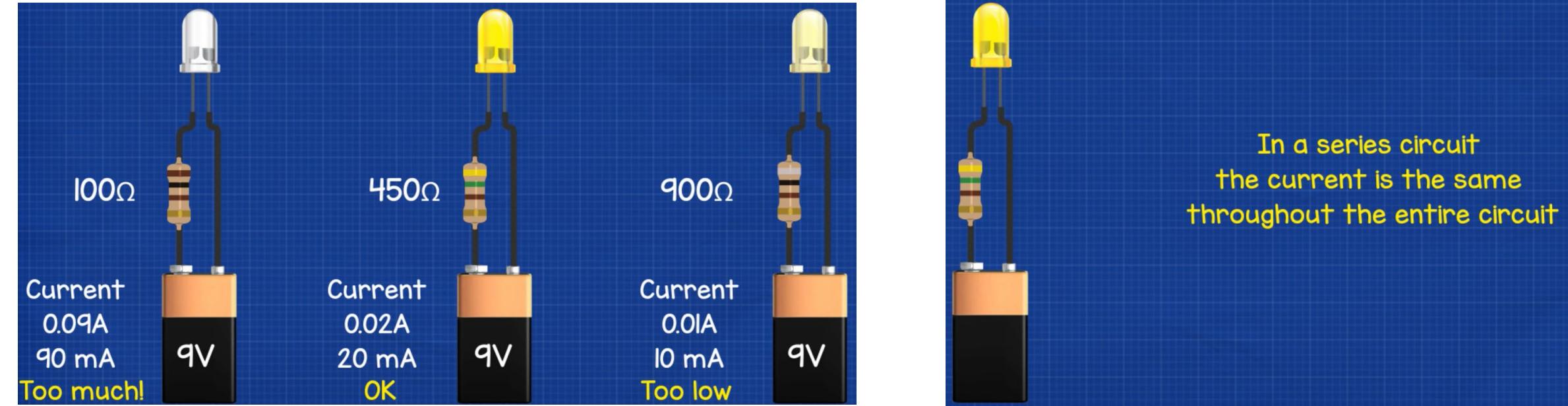


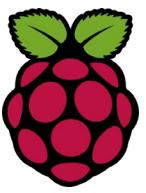
# Resistor Application



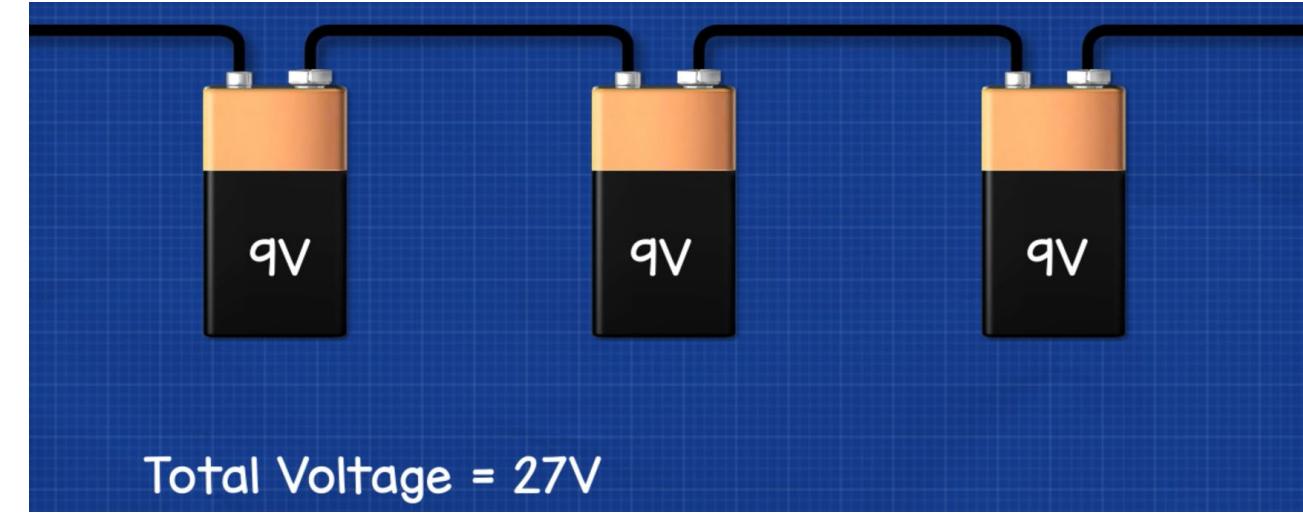
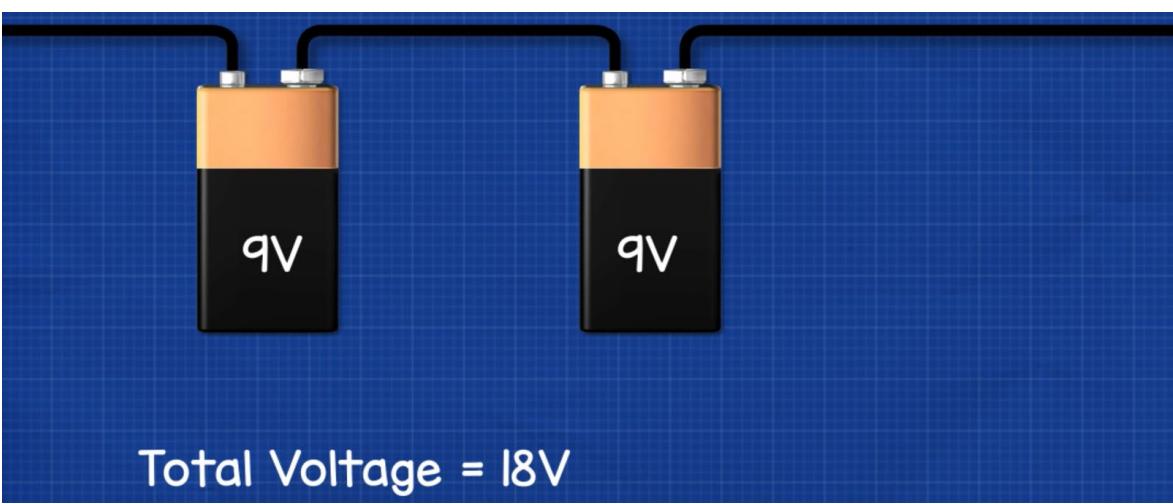
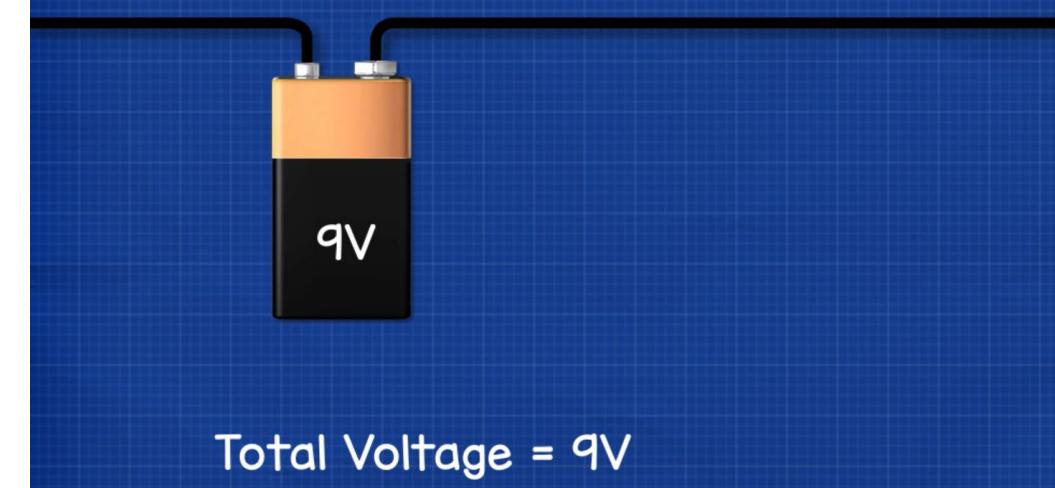
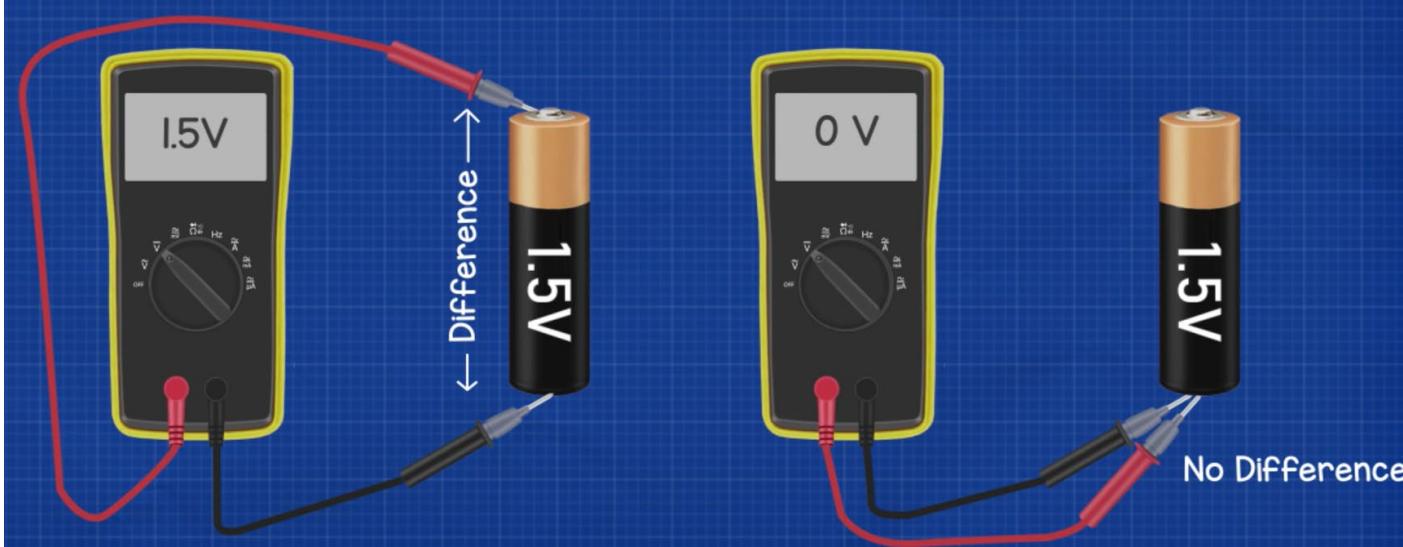


# Resistor Application



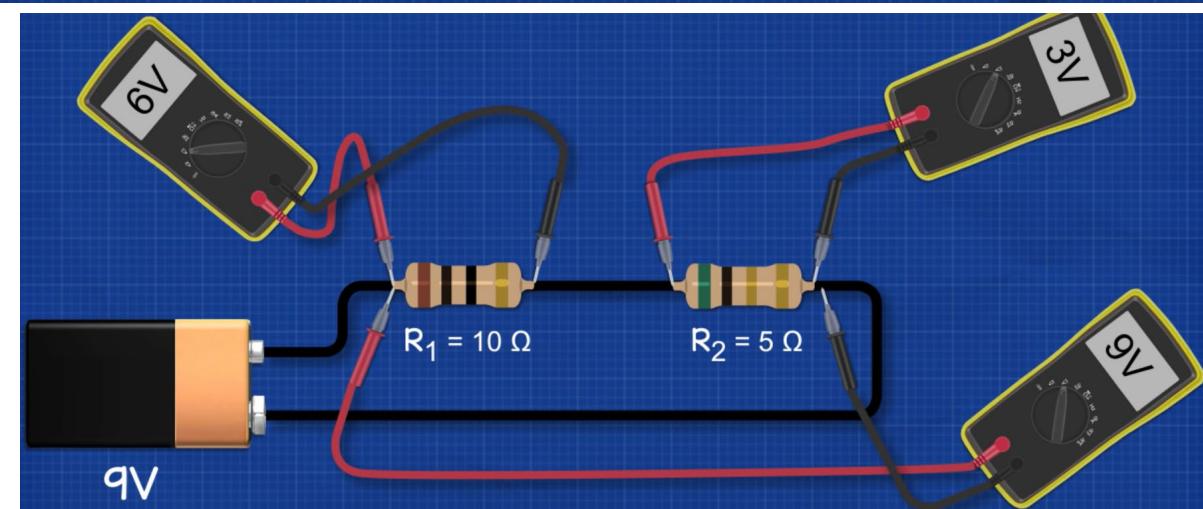
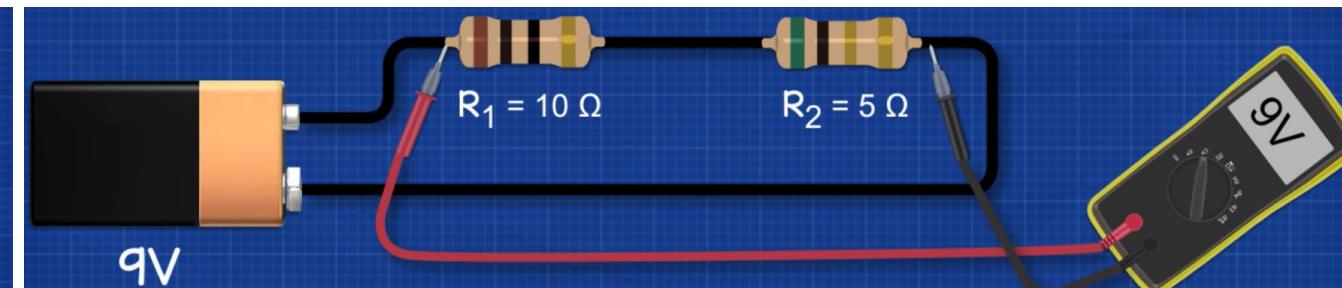
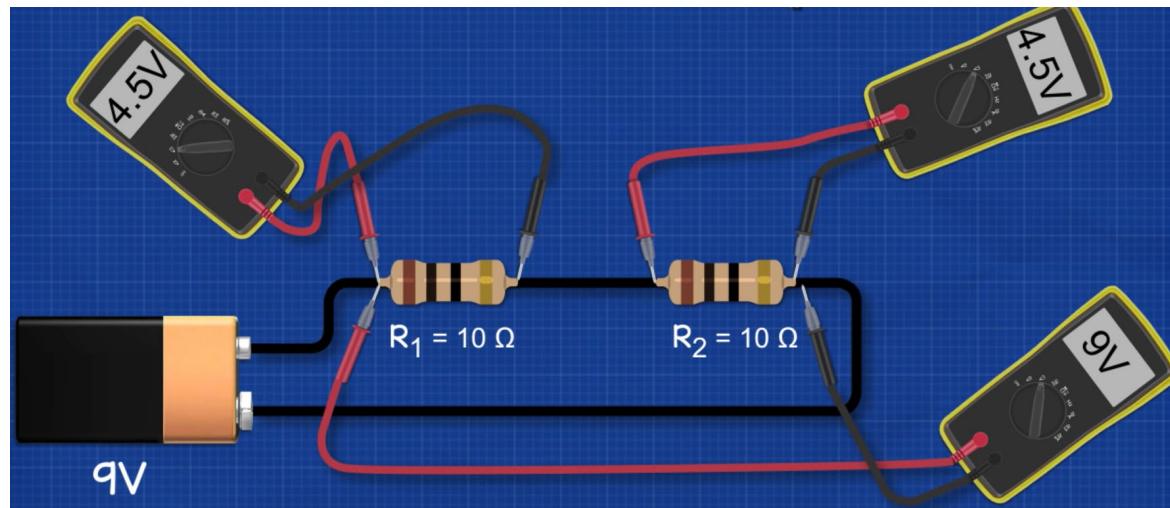
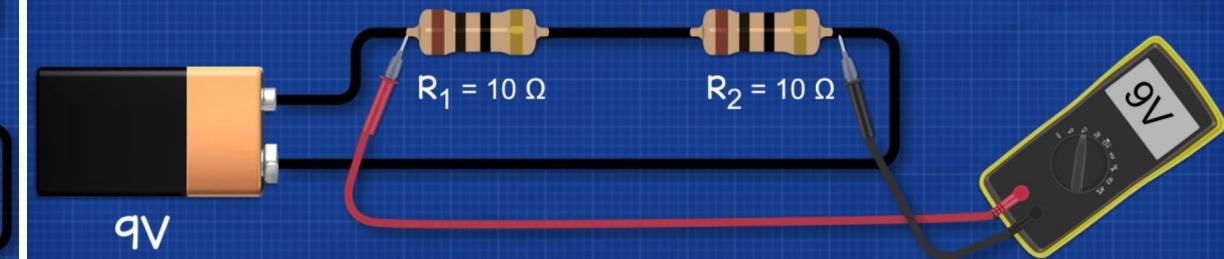
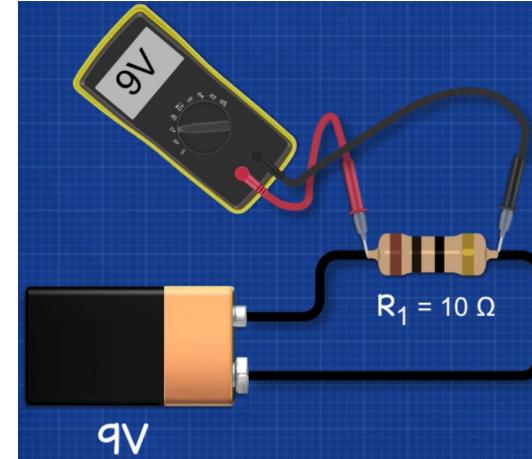
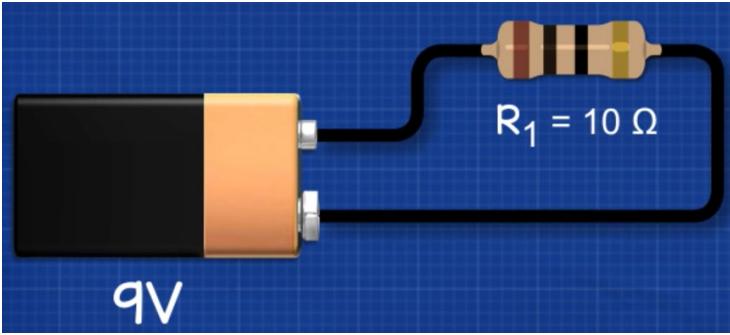
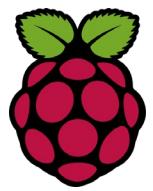


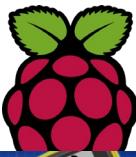
# Voltage Calculation



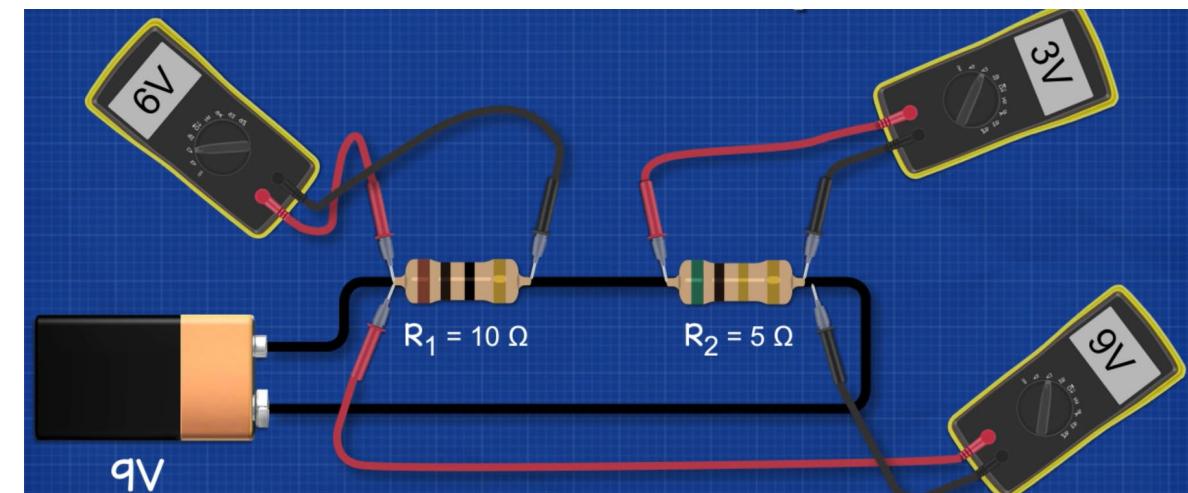
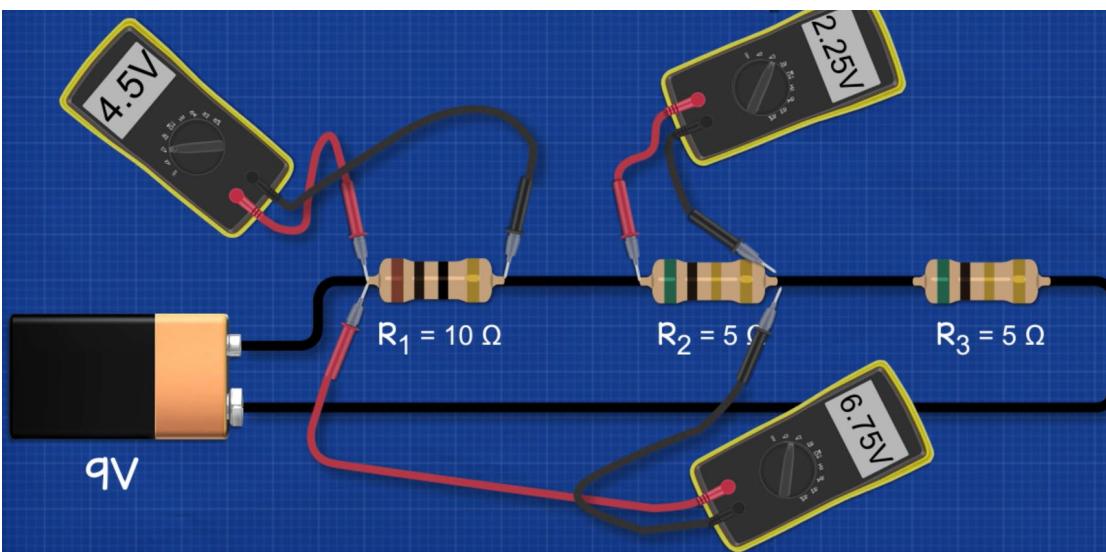
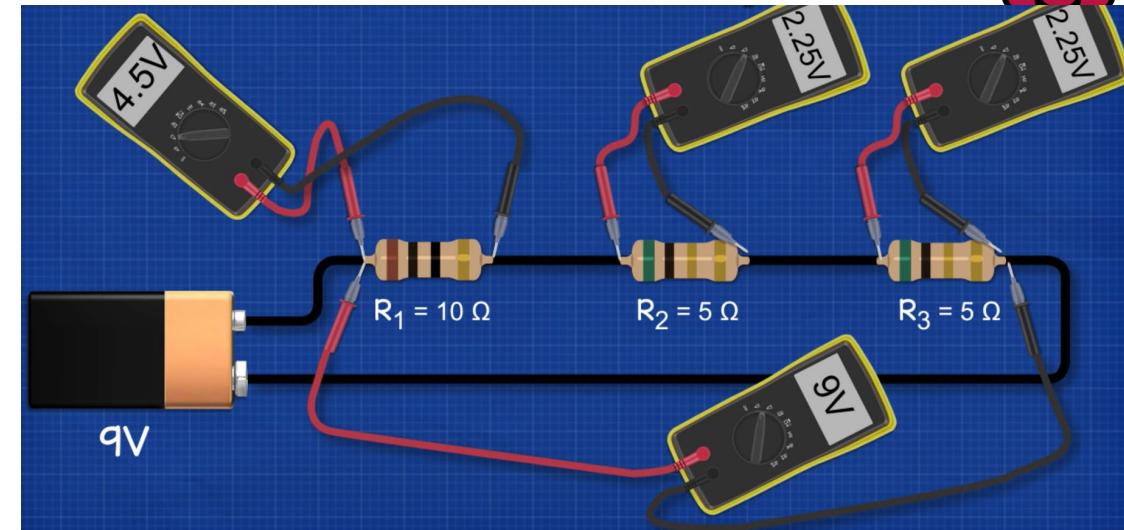
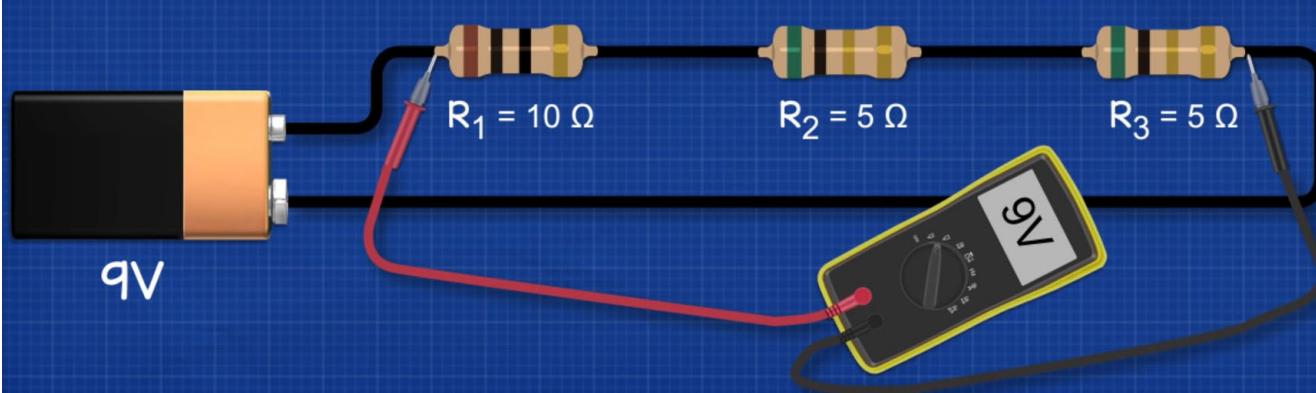


# Resistor Application



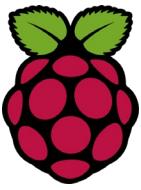


# Resistor Application

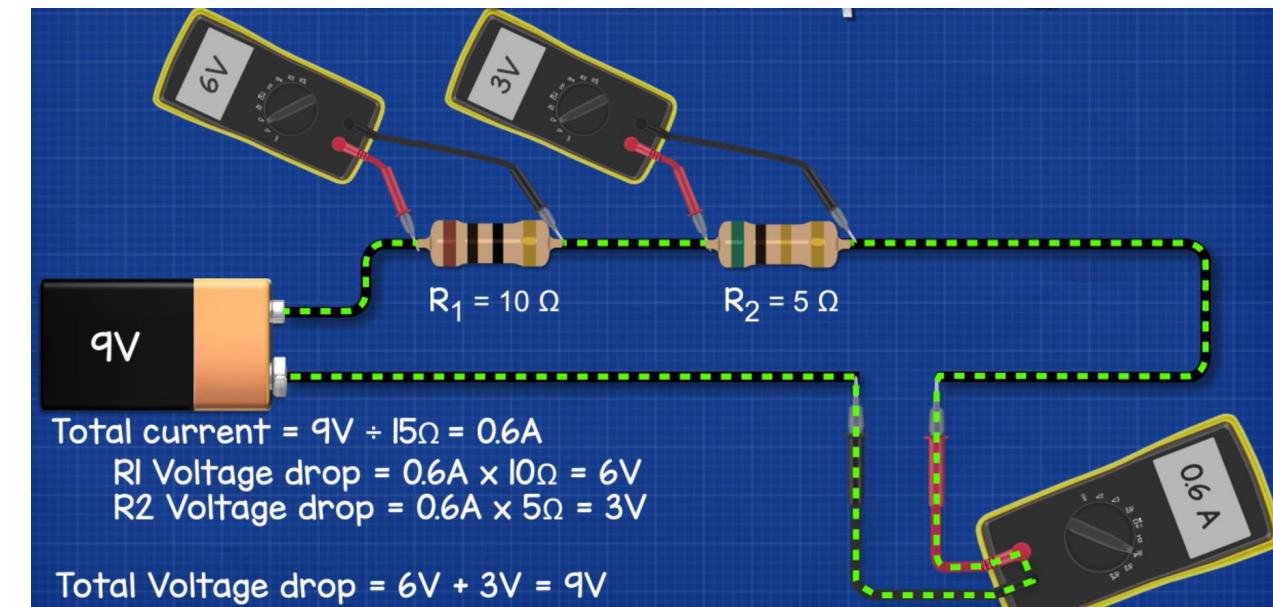
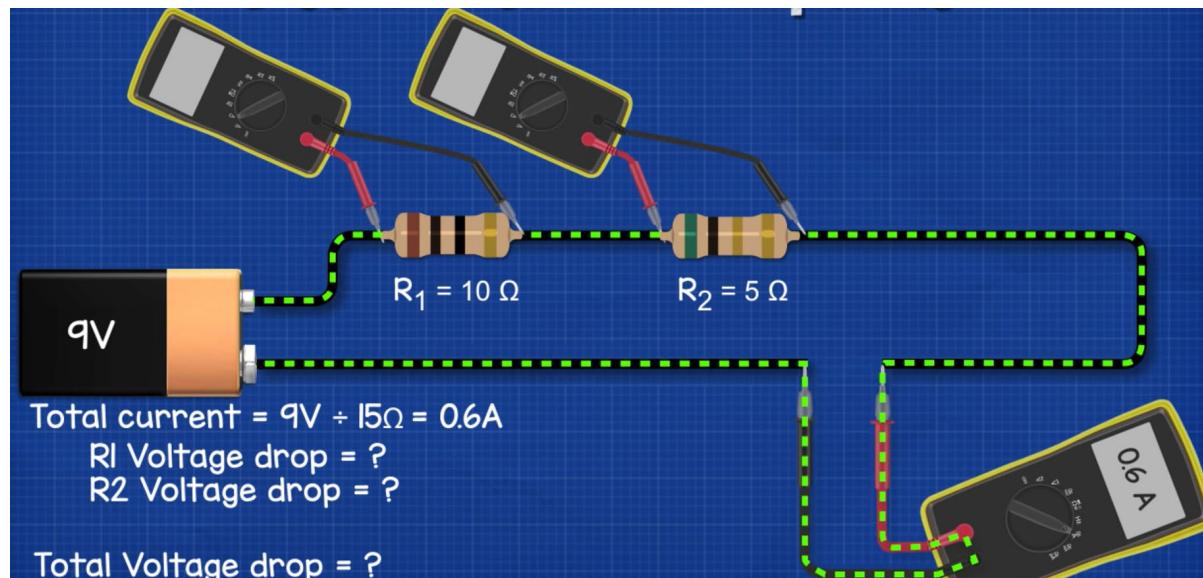
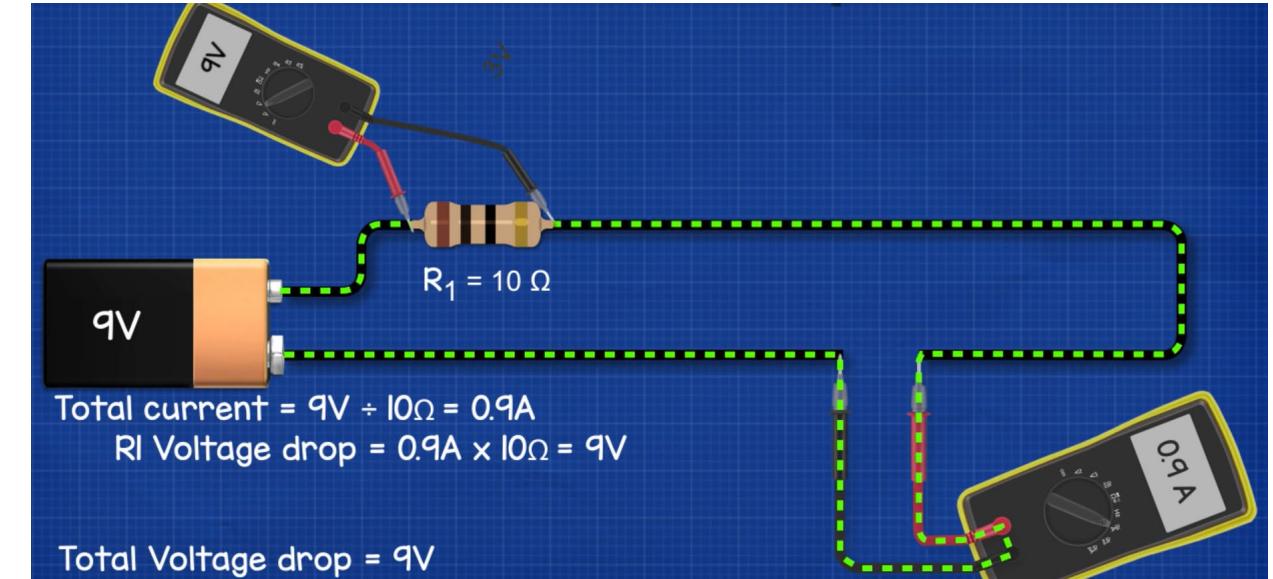
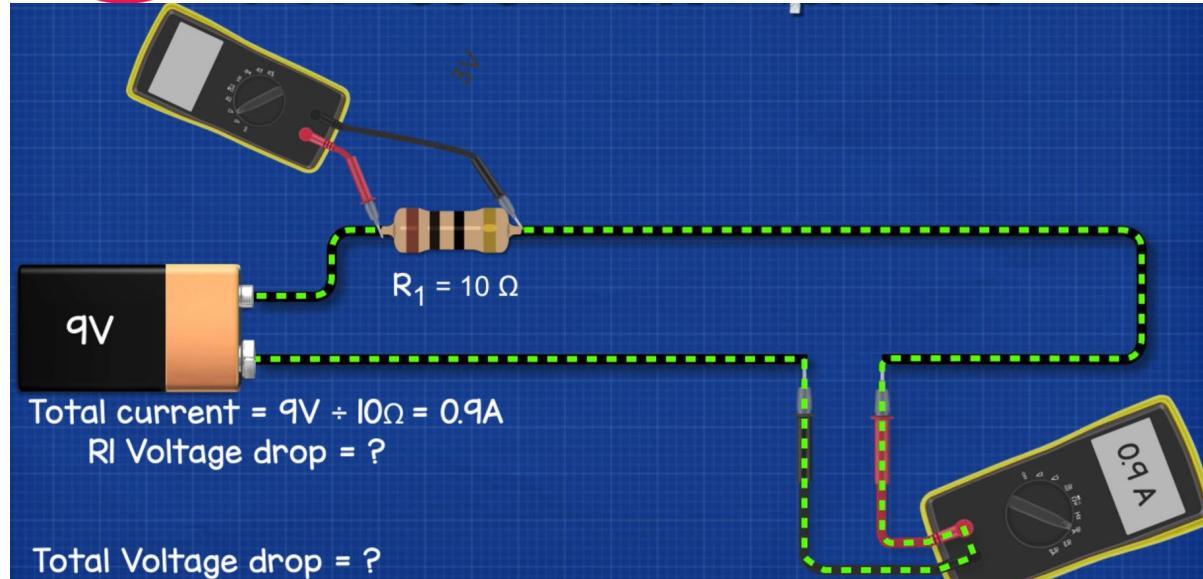


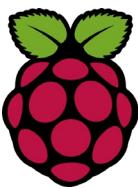
Voltage is reduced by the resistor, it creates a voltage drop.

Unlike current, where it's the same throughout the circuit  
Voltage will be different throughout in a series circuit

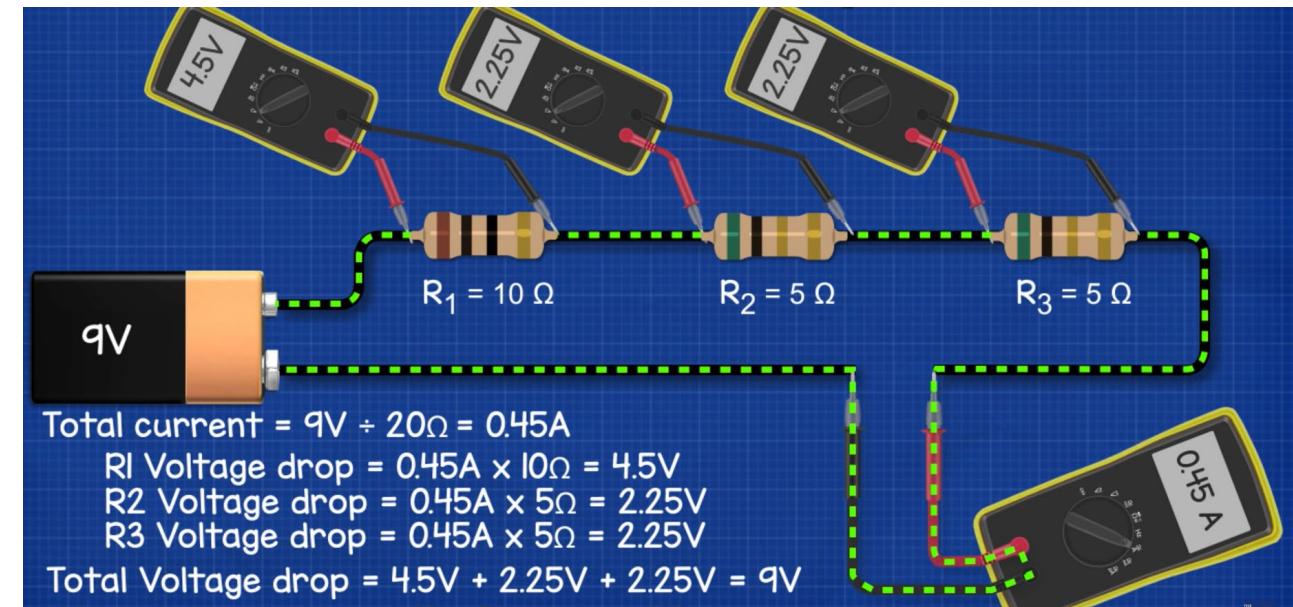
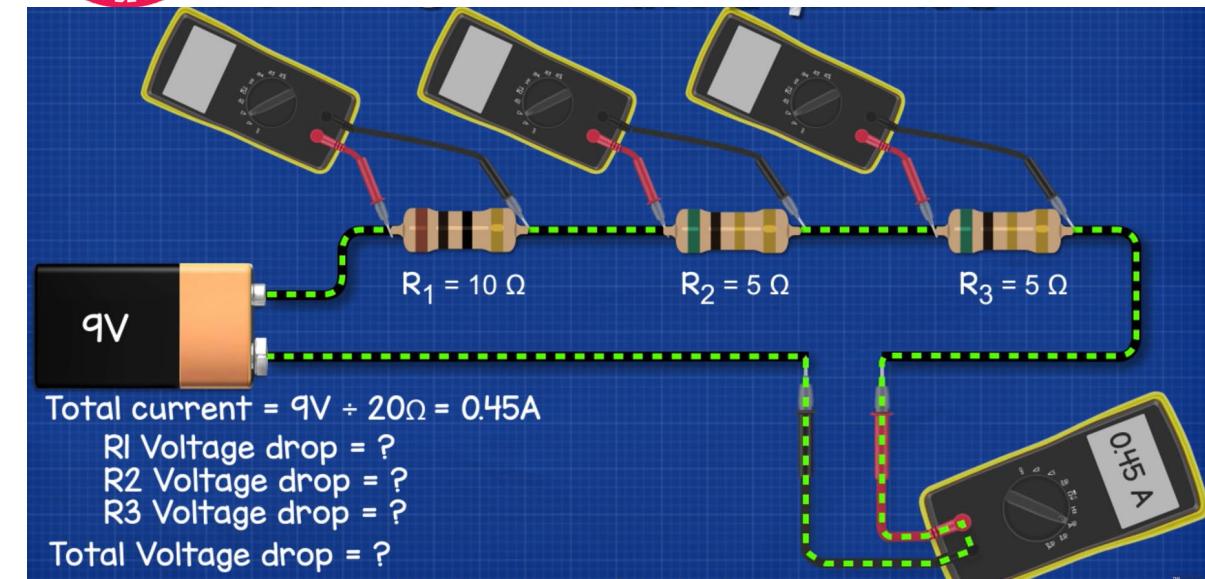


# Voltage Calculation





# Voltage, Current and Power Calculation



## How do we calculate power consumption?

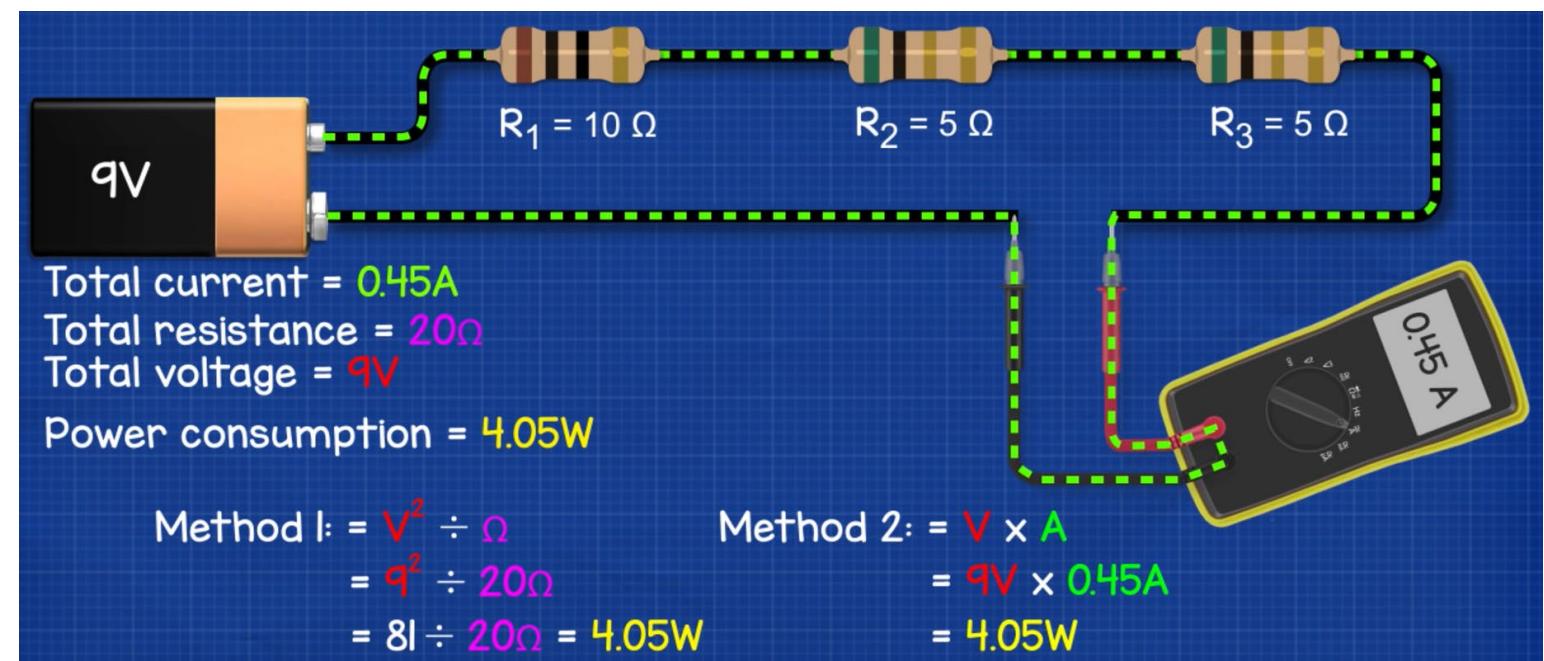
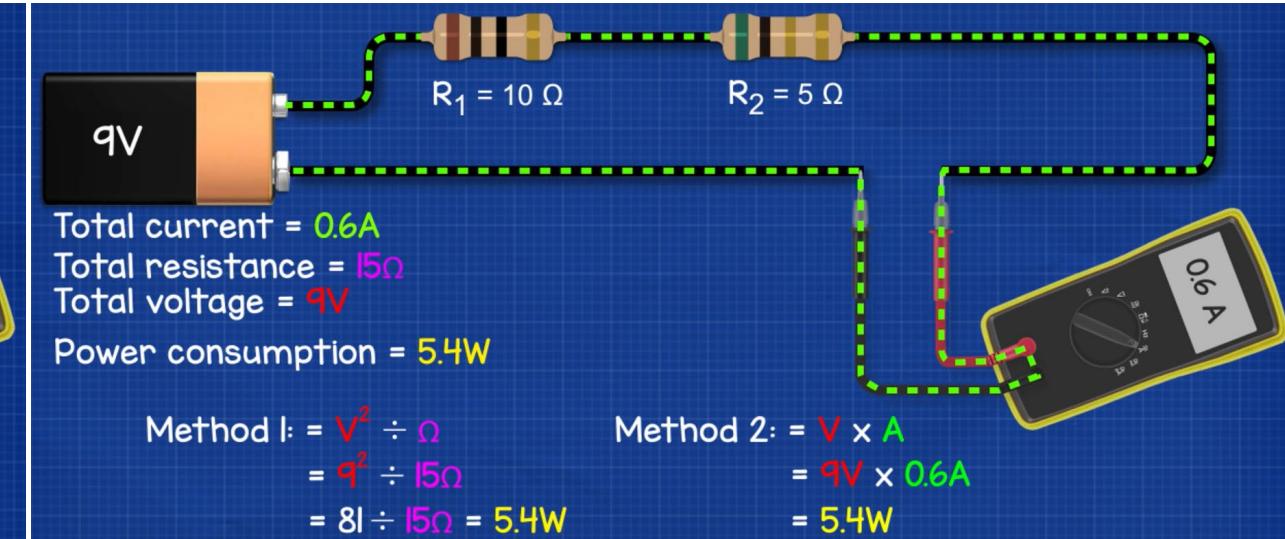
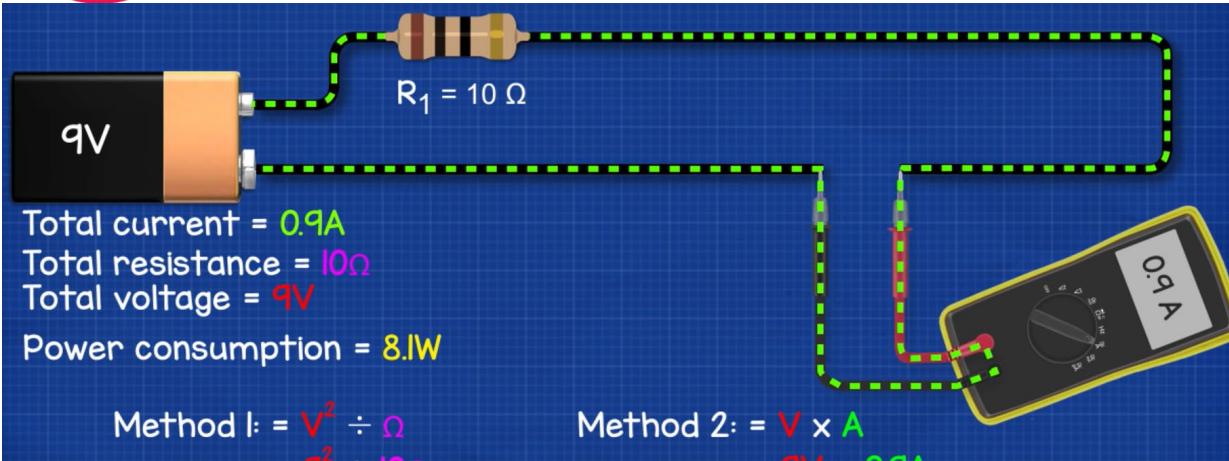
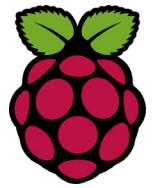
Formulas:

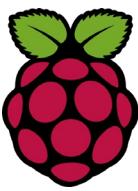
$$\text{Power (Watts)} = \text{Voltage}^2 (V) \div \text{Resistance} (\Omega)$$

or

$$\text{Power (Watts)} = \text{Voltage (V)} \times \text{Current (A)}$$

# Power Consumption Calculation



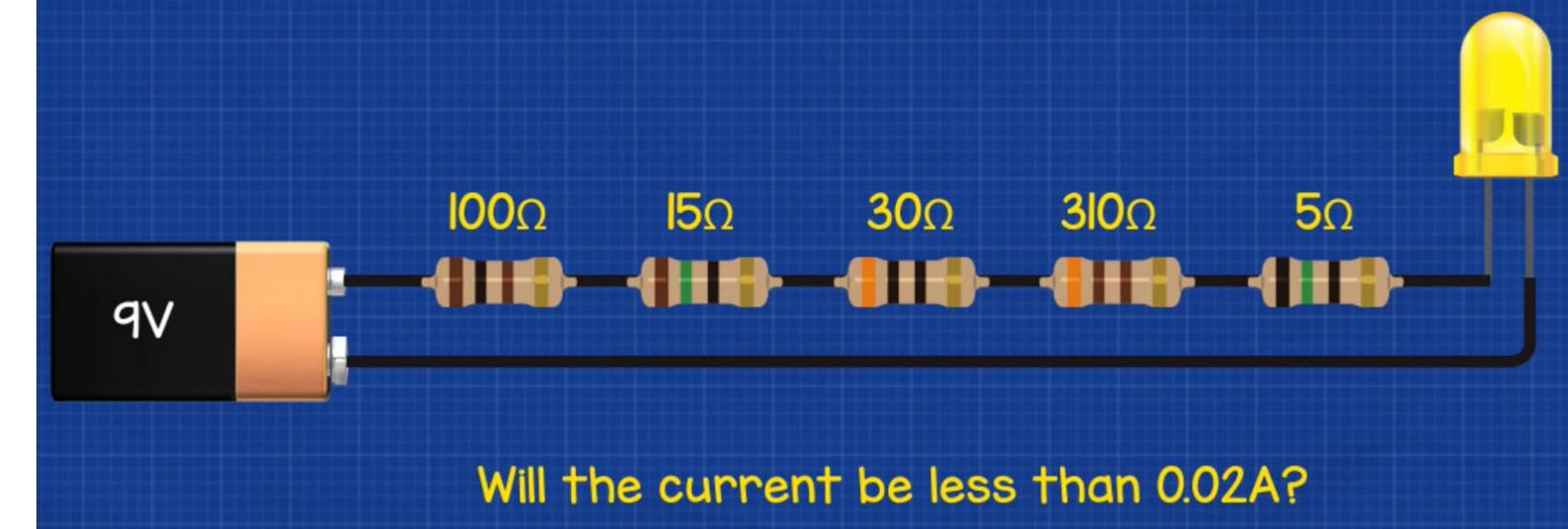


# Try to Solve

Can you solve this problem?



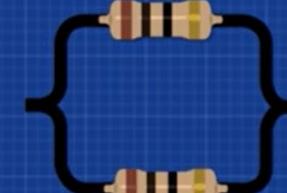
This LED can't exceed 0.02A (20mA)  
Else it will burn out



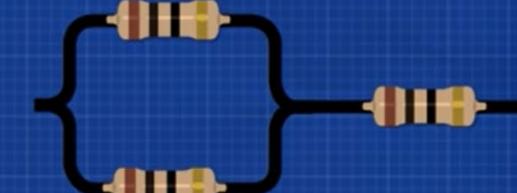
Series



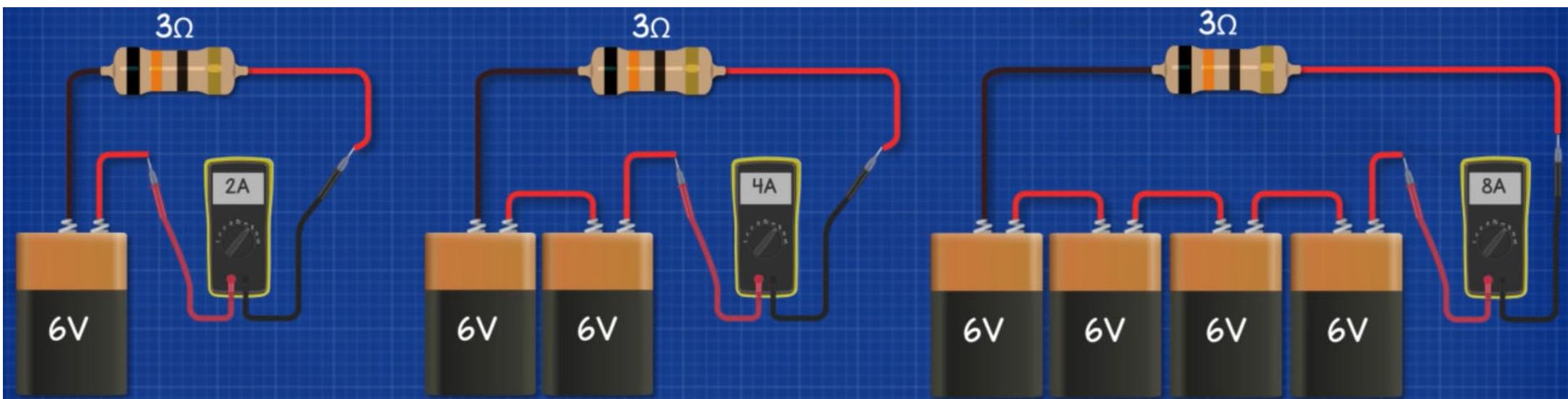
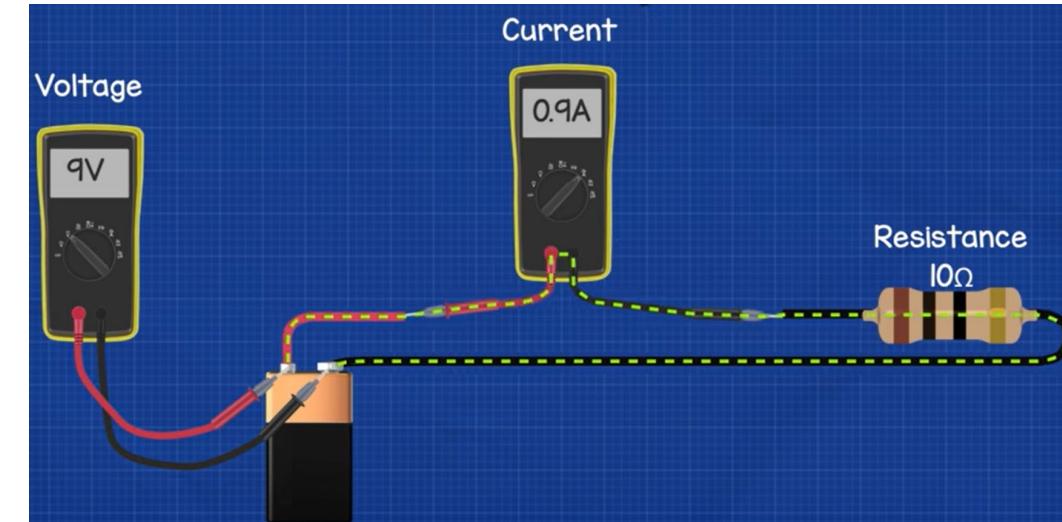
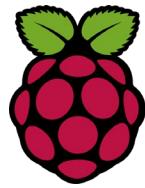
Parallel



Parallel & Series



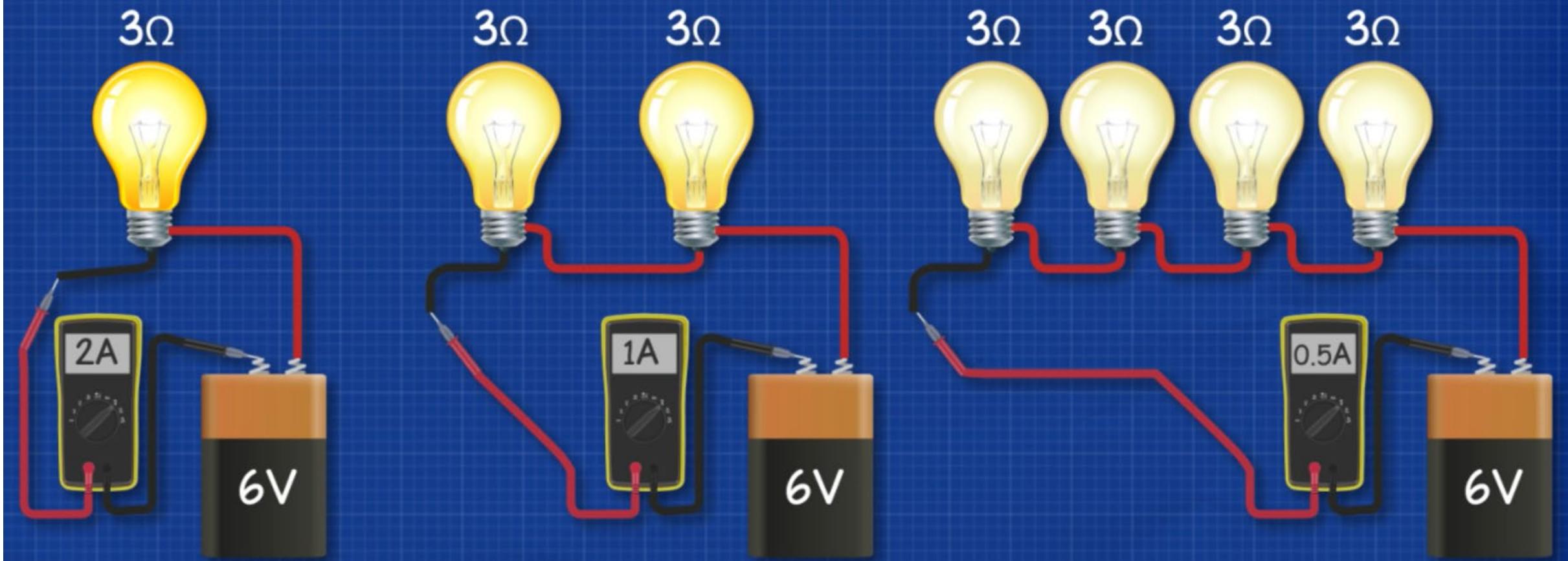
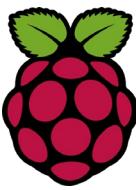
# Ohm's Law



Relationship: current is therefore directly proportional to voltage.

Double the Voltage, double the current

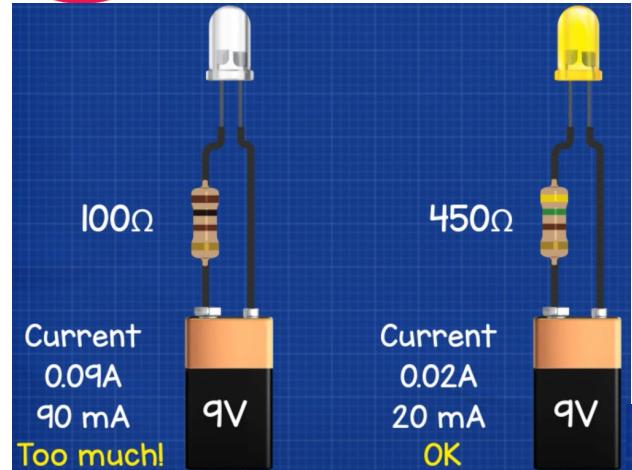
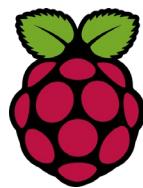
# Ohm's Law Application



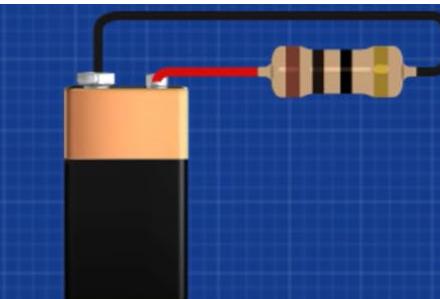
Relationship: Current is inversely proportional to resistance

Double the resistance, half the current

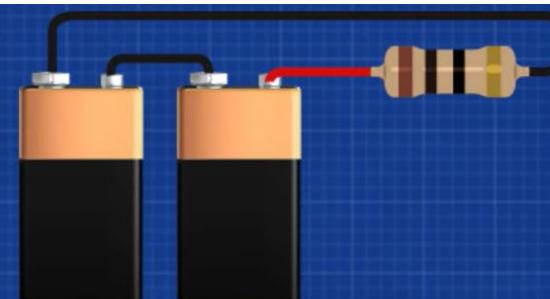
# Ohm's Law



Increase current by increasing the voltage



Low voltage, low current

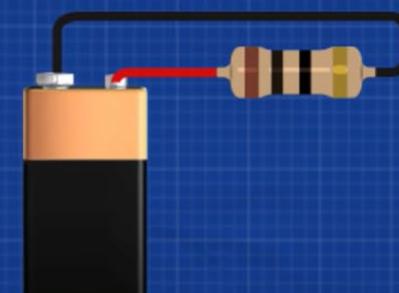


Higher voltage, higher current

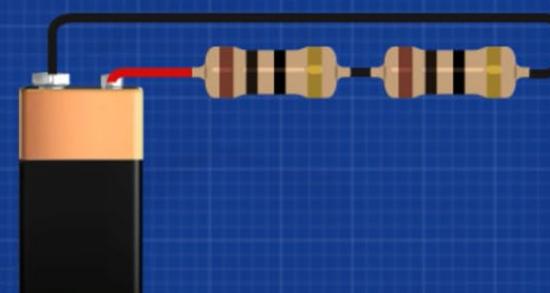
Increase current by reducing the resistance

or

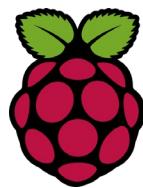
Reduce current by increasing resistance



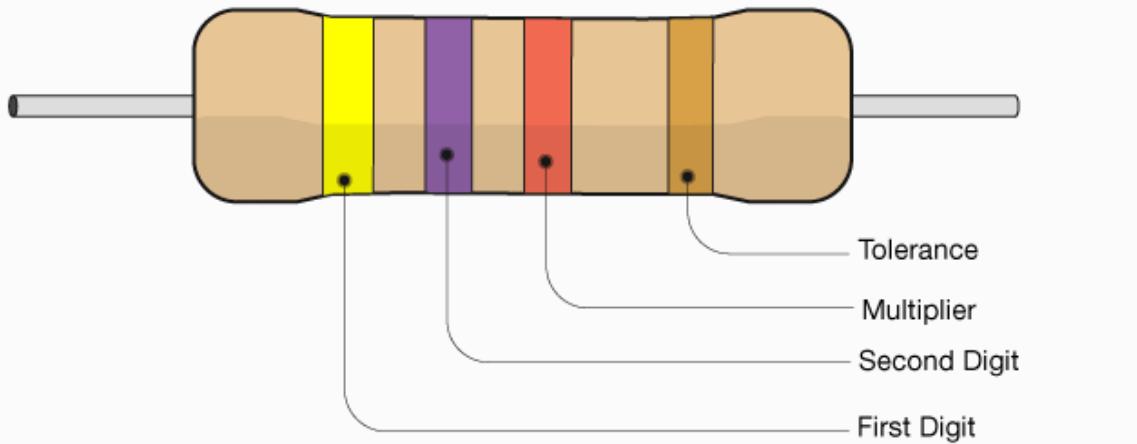
Low voltage, low current



Low voltage, high resistance, low current



# Resistor Colour Code



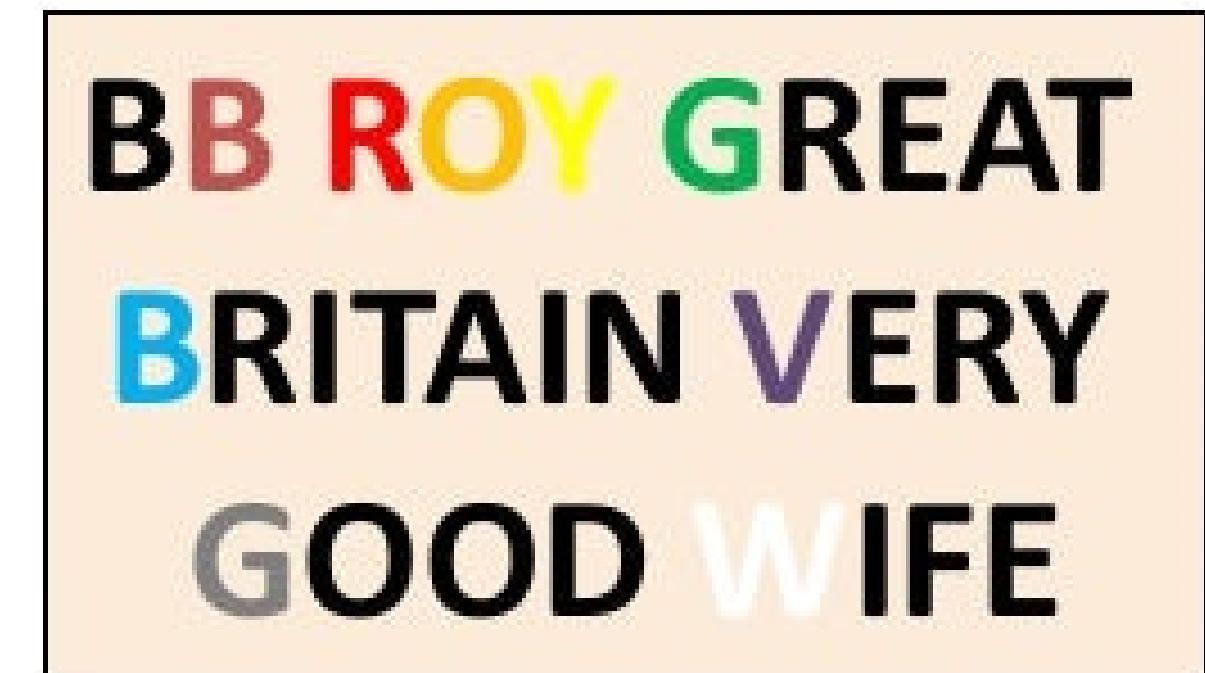
	1st Digit	2nd Digit	Multiplier	Tolerance
Black	0	0	x 1	Silver ±10%
Brown	1	1	x 10	Gold ±5%
Red	2	2	x 100	
Orange	3	3	x 1000	
Yellow	4	4	x 10000	
Green	5	5	x 100000	
Blue	6	6	x 1000000	
Violet	7	7		
Grey	8	8		
White	9	9		

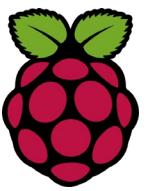
Example Shown :

Yellow	Violet	Red	Gold
4	7	⊗ 100	±5%

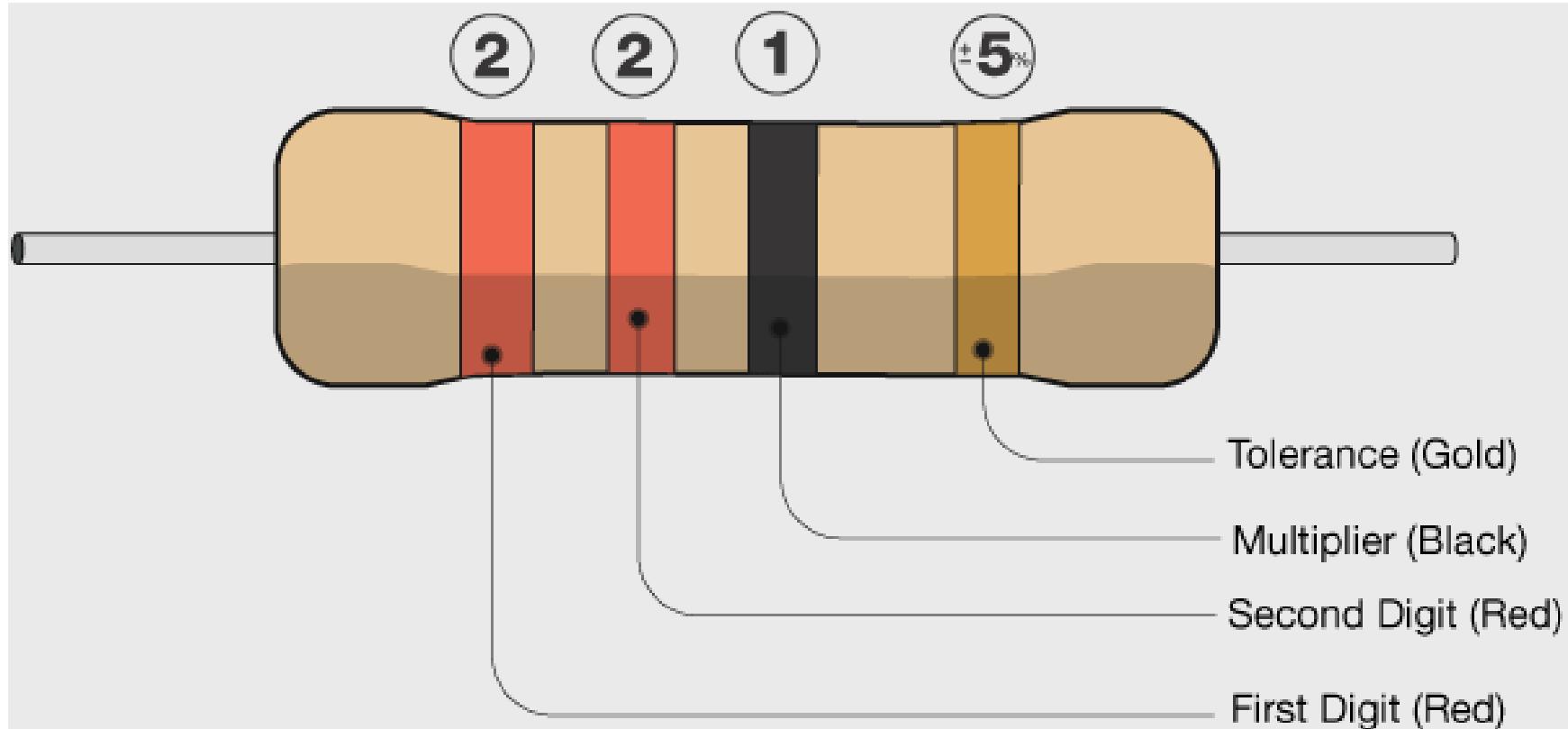
$4700 \Omega \pm 5\%$

## Mnemonic

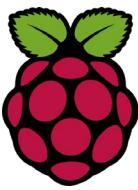




# Resistor Colour Code Examples



Band colours in order	RED	RED	BLACK	GOLD
Digit representation	2	2	1	±5%
Value	$22 \Omega \pm 5\%$			



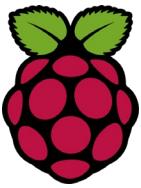
# Tolerance Limit

Band colours in order	RED	RED	BLACK	GOLD
Digit representation	2	2	1	$\pm 5\%$
Value	$22 \Omega \pm 5\%$			

The tolerance limit values represent by how much the resistance can vary from its mean value in terms of percentage.

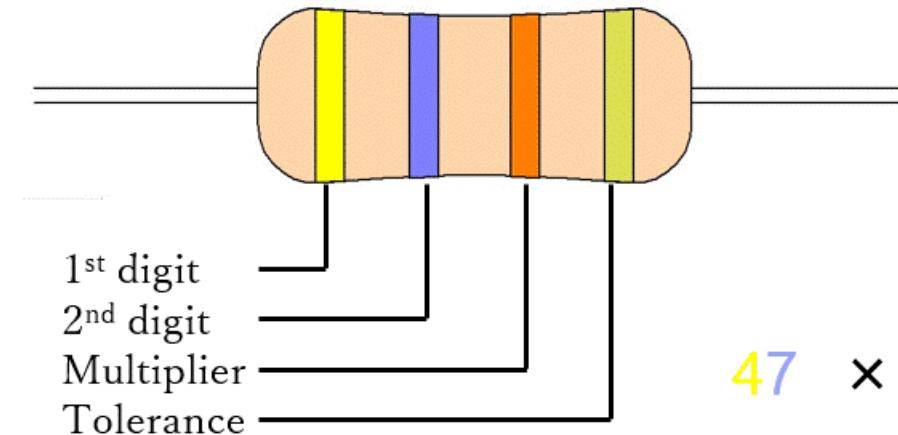
$$\begin{aligned} \text{Tolerance} &= \text{Value of resistor} \times \text{value of tolerance band} \\ &= 22 \Omega \times 5\% = 1.1 \Omega \end{aligned}$$

22 Ω resistor with a tolerance value of 1.1 Ω could range from the actual value as much as 23.1 Ω to as little as 20.9 Ω.



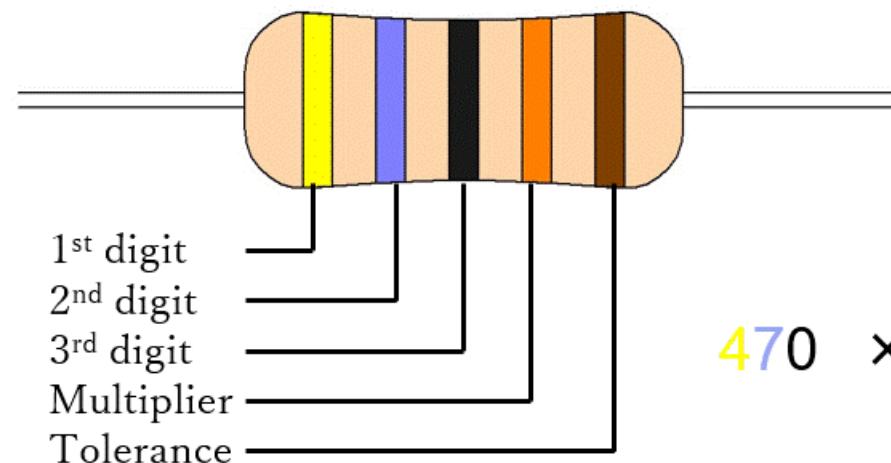
# Resistor Colour Code Examples

4-Band Code

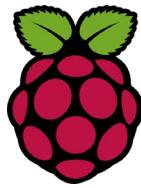


$$47 \times 10^3 = 47000 = 47\text{k}\Omega \pm 5\%$$

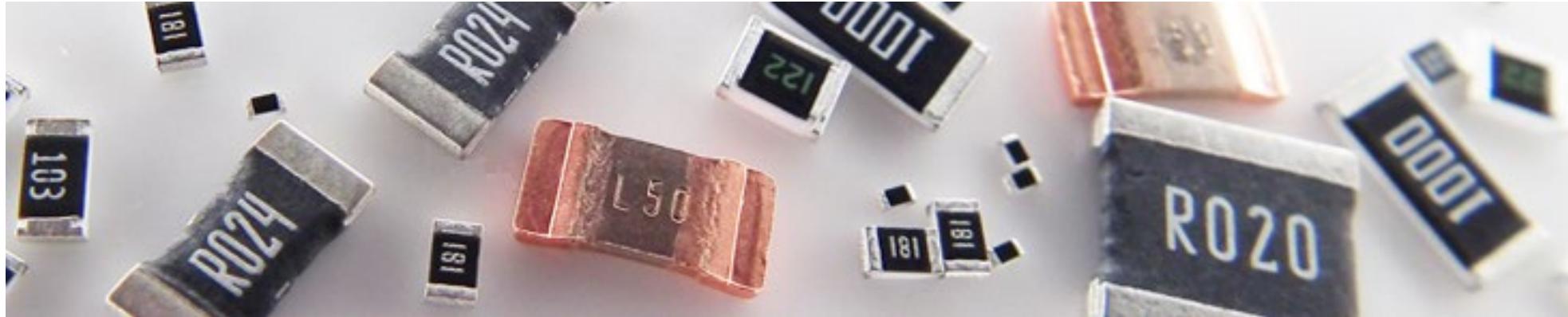
5-Band Code



$$470 \times 10^3 = 470000 = 470\text{k}\Omega \pm 1\%$$



# Chip Resistors conversion rules



**102** →  $10 \times 10^2 = 1000\Omega$  ( $1k\Omega$ )

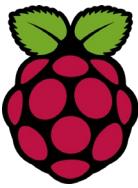
**1002** →  $100 \times 10^2 = 10000\Omega$  ( $10 k\Omega$ )

**R047** → Read "R" as a decimal point. →  $0.047\Omega$  ( $47m\Omega$ )

**0K47** → Read "K" as a decimal point. →  $0.47K\Omega$

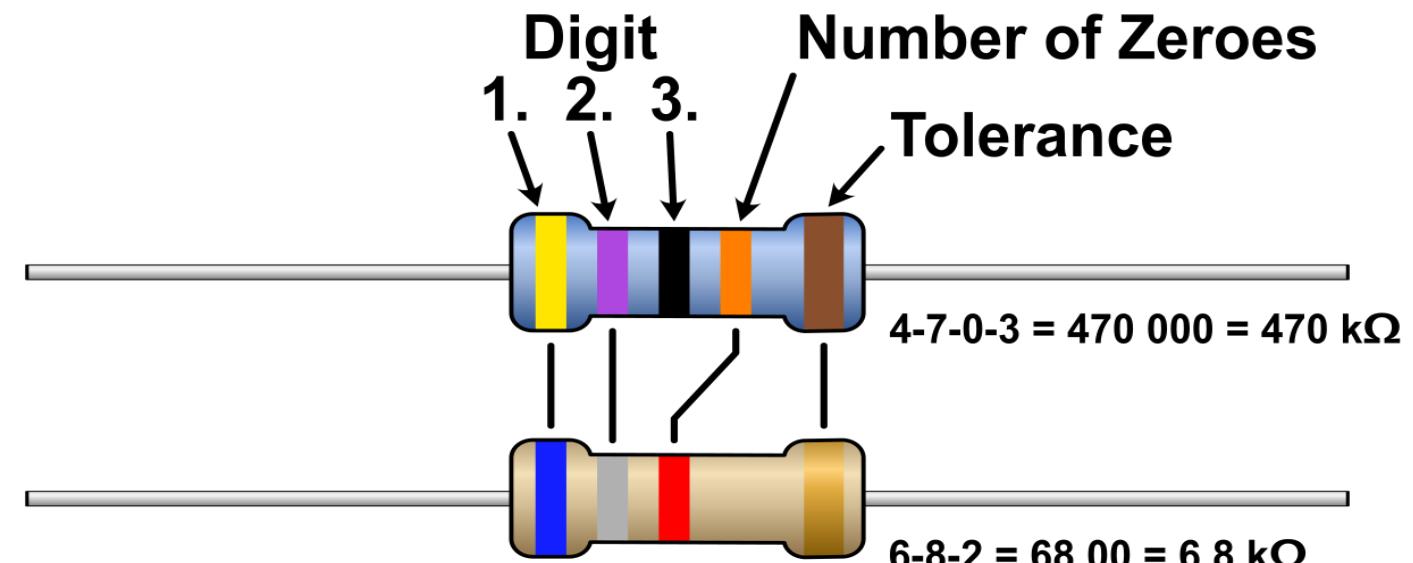
**4M7** → Read "M" as a decimal point. →  $4.7M\Omega$

**10L0** → Read "L" as a decimal point of  $m\Omega$  →  $10m\Omega$



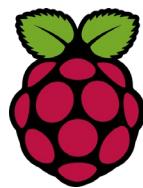
# Most commonly used Resistors

Resistor Value	Common Resistor	Color Band
100 Ohm		Brown Black Brown 1 0 0
220 Ohm		Red Red Brown 2 2 0
330 Ohm		Orange Orange Brown 3 3 0
470 Ohm		Yellow Violet Brown 4 7 0
1 K		Brown Black Red 1 0 00
2.2 K		Red Red Red 2 2 00
3.3.K		Orange Orange Red 3 3 00
4.7 K		Yellow Violet Red 4 7 00
5.6 K		Green Blue Red 5 6 00
6.8 K		Blue Grey Red 6 8 00
10 K		Brown Black Orange 1 0 000
22 K		Red Red Orange 2 2 000
33 K		Orange Orange Orange 3 3 000
47 K		Yellow Violet Orange 4 7 000
100 K		Brown Black Yellow 1 0 0000
150 K		Brown Green Yellow 1 5 0000



**Digit**      **0**    **1**    **2**    **3**    **4**    **5**    **6**    **7**    **8**    **9**

**Tolerance**      **Silver**  $\pm 10\%$     **Gold**  $\pm 5\%$      **$\pm 1\%$**      **$\pm 0.5\%$**      **$\pm 0.1\%$**



# Extended Tolerance Limit

Color	Value	Multiplier	Tolerance
Black	0	$\times 10^0$	$\pm 20\%$
Brown	1	$\times 10^1$	$\pm 1\%$
Red	2	$\times 10^2$	$\pm 2\%$
Orange	3	$\times 10^3$	$\pm 3\%$
Yellow	4	$\times 10^4$	
Green	5	$\times 10^5$	$\pm 0.5\%$
Blue	6	$\times 10^6$	$\pm 0.25\%$
Violet	7	$\times 10^7$	$\pm 0.10\%$
Gray	8	$\times 10^8$	$\pm 0.05\%$
White	9	$\times 10^9$	
Gold	-	$\times 10^{-1}$	$\pm 5\%$
Silver	-	$\times 10^{-2}$	$\pm 10\%$

## 4-band resistor

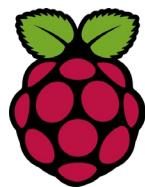


**270 ohms  $\pm 5\%$**

## 5-band resistor



**100k ohms  $\pm 1\%$**



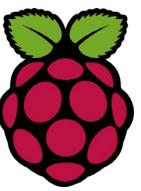
# Resistor Colour Code Examples

4 Bands			1.2 Ω 10%		
5 Bands			68 KΩ 5%		
6 Bands			560 KΩ 5%		
1st Digit	2nd Digit	3rd Digit	Multiplier	Tolerance	Temperature Coefficient
0	0	0	1		
1	1	1	10	1%	100ppm
2	2	2	100	2%	50ppm
3	3	3	1 K		15ppm
4	4	4	10 K		25ppm
5	5	5	100 K	0.5%	
6	6	6	1 M	0.25%	
7	7	7	10 M	0.1%	
8	8	8			0.05%
9	9	9		0.01	10%
			0.1		5%

  Resistor Color Codes

1K = 1 000  
1M = 1 000 000

Color Codes		4 Band Resistors	5 Band Resistors	6 Band Resistors
0	Black			
1	Brown			
2	Red			
3	Orange			
4	Yellow			
5	Green			
6	Blue			
7	Purple			
8	Grey			
9	White			
±1%	Brown			
±2%	Red			
±5%	Gold			
±10%	Silver			
±1%	Black	27K	15K	620K
±2%	Brown			
±5%	Gold			
±10%	Silver			
EXAMPLE				
0 x 1				
1 1 x 10				
2 2 x 100				
3 3 x 1000				
4 4 x 10000				
5 5 x 100000				
6 6 x 1000000				
7 7 x 10000000				
8 8 x 100000000				
9 9 x 1000000000				
÷10				
÷100				



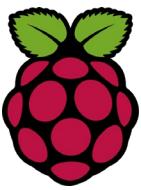
# Quizzzz

**1) The Color Coding is used to indicate \_\_\_\_\_ value/rating of Resistor.**

- a) Numerical
- b) Alphabetical
- c) Resistance
- d) a & c are correct

**2) The Resistor Color Code was developed by \_\_\_\_\_.**

- a) International Organization for Standardization (ISO)
- b) Electronics Industries Alliance (EIA)
- c) Radio Manufacturers Association (RMA)
- d) a & b are correct



# Quizzzz

**3) Recently, \_\_\_\_\_ standard Color Coding is used for Resistor's values.**

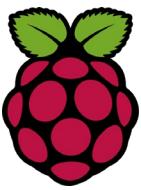
- a) World Standardized Corporation (WSC)
- b) Electronics Industries Alliance (EIA)
- c) Radio Manufacturers Association (RMA)
- d) International Organization for Standardization (ISO)

**4) The color code standard was developed in \_\_\_\_\_.**

- a) 1890
- b) 1920
- c) 1940
- d) 1900



# Quizzzz



**5) Why is Color Coding used for Resistors ?**

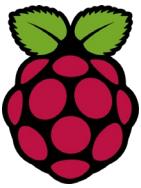
- a) Small Size
- b) Cylindrical Shape
- c) Due to Through Hole Component
- d) a & b are correct

**6) How many colour bands used on Resistors** \_\_\_\_\_

- a) 4
- b) 5
- c) 6
- d) 7



# Quizzzz



**7) How to read color bands on Resistors ?**

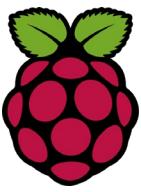
- a) Right to Left
- b) Left to Right
- c) From both sides
- d) All are correct

**8) The first two bands on the resistors are \_\_\_\_\_**

- a) Two digits
- b) Decimal Multiplier
- c) Tolerance
- d) All are incorrect



# Quizzzz

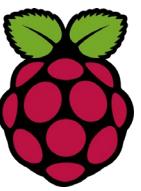


**9) The tolerance of the Silver band on Resistor is \_\_\_\_\_**

- a) 3%
- b) 5%
- c) 10%
- d) 20%

**10) The gold band tolerance on Resistor is \_\_\_\_\_**

- a) 3%
- b) 5%
- c) 10%
- d) 20%



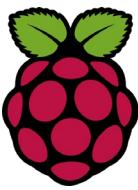
# Quizzzz

11) If there is no band on Resistor then tolerance is \_\_\_\_\_

- a) 3%
- b) 5%
- c) 10%
- d) 20%

12) Which digit is represented by a Green band on a Resistor?

- a) 4
- b) 5
- c) 6
- d) 3



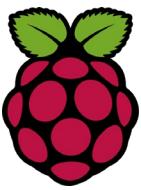
# Quizzzz

**13) The decimal multiplier means, how many \_\_\_\_\_ add after the first digits.**

- a) Zeros
- b) Digits
- c) Tolerance
- d) Resistance

**14) What color is a  $340\text{ K}\Omega$  resistor with 5% tolerance?**

- a) Orange, Yellow and Yellow with Gold
- b) Orange, Green and Yellow with Gold
- c) Orange, Blue and Orange with Gold
- d) Orange, Yellow and Green with Gold



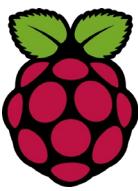
## Quizzzz

**15) A zero-ohm resistor is indicated by a single black color ring around the body of the resistor.**

- a) Yes
- b) No
- c) Zero ohm resistors does not exist
- d) None of these

**16) Why zero ohm resistors are used?**

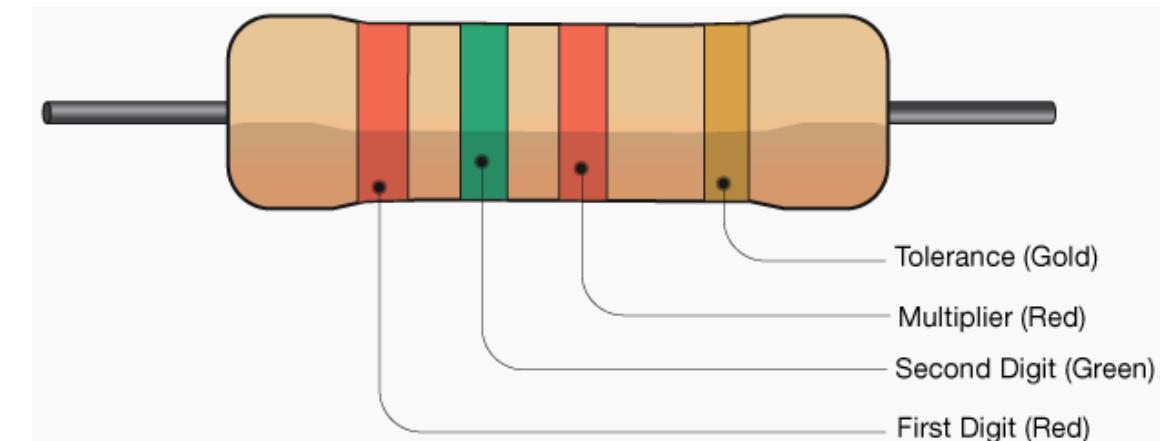
- a) Connect the Tracks
- b) As Jumpers
- c) Increase Machine Production Time
- d) All are correct



# Quizzzz

**17) Determine the resistance of the given resistor with the given colour sequence (Red, Green, Red, Gold).**

- a)  $250 \pm 5\% \Omega$ .
- b)  $25 \pm 5\% k\Omega$ .
- c)  $2500 \pm 5\% k\Omega$ .
- d)  $2.5 \pm 5\% k\Omega$ .

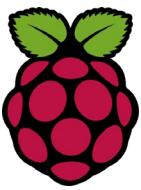


**18) What colour bands would a resistor of resistance value  $1000 \Omega$  with a tolerance level  $\pm 5\%$  have?**

- a) Brown, Black and Orange , Gold
- b) Black, Brown and Orange, Gold
- c) Black, Brown, Black , Red , Gold
- d) Brown, Black , Red , Gold



# Quizzzz



19) The tolerance of the Brown band on Resistor is \_\_\_\_\_

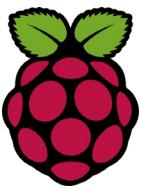
- a)  $\pm 3\%$
- b)  $\pm 5\%$
- c)  $\pm 1\%$
- d)  $\pm 2\%$

20) The green band tolerance on Resistor is \_\_\_\_\_

- a)  $\pm 0.3\%$
- b)  $\pm 0.5\%$
- c)  $\pm 1\%$
- d)  $\pm 2\%$



# Quizzzz



21) The tolerance of the Red band on Resistor is \_\_\_\_\_

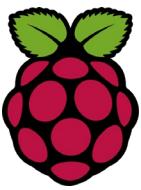
- a)  $\pm 3\%$
- b)  $\pm 5\%$
- c)  $\pm 1\%$
- d)  $\pm 2\%$

22) The blue band tolerance on Resistor is \_\_\_\_\_

- a)  $\pm 0.3\%$
- b)  $\pm 0.5\%$
- c)  $\pm 1\%$
- d)  $\pm 0.25\%$



# Quizzzz

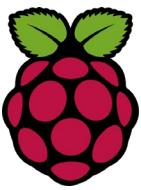


23) The tolerance of the violet band on Resistor is \_\_\_\_\_

- a)  $\pm 0.3\%$
- b)  $\pm 0.5\%$
- c)  $\pm 0.1\%$
- d)  $\pm 0.25\%$

24) The grey band tolerance on Resistor is \_\_\_\_\_

- a)  $\pm 0.3\%$
- b)  $\pm 0.05\%$
- c)  $\pm 0.25\%$
- d)  $\pm 0.1\%$



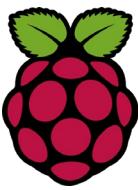
## Quizzzz

25) Three Resistances of  $R_1=20\Omega$  ,  $R_2=57\Omega$  &  $R_3=90\Omega$  are in series then how will the total resistance?

- a)  $167\Omega$
- b)  $157\Omega$
- c)  $147\Omega$
- d)  $177\Omega$

26) When current flows through a resistance, electrical energy is converted into \_\_\_\_\_ energy.

- a) Chemical
- b) Heat
- c) Light
- d) Elastic



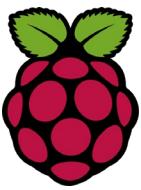
## Quizzzz

27) The formula to calculate Current (I) in Series Resistance Circuit is \_\_\_\_\_.

- a)  $I = V/R$
- b)  $I = I^2 \times R$
- c)  $I = V \times R$
- d)  $I = V_{total} / R_{total}$

28) When components are connected in successive order is called \_\_\_\_\_.

- a) Short Circuit
- b) Open Circuit
- c) Bypassed Circuit
- d) All are correct



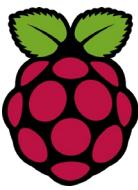
## Quizzzz

29) \_\_\_\_\_ states that the sum of all resistor voltage drops in a series circuit equals the applied voltage.

- a) Ohm's Law
- b) Kirchhoff's voltage law (KVL)
- c) Kirchhoff's current law (KCL)
- d) b & c are correct

30) In a \_\_\_\_\_ Circuit, current is Same in the all part/component.

- a) Series
- b) Parallel
- c) String
- d) a & c are correct



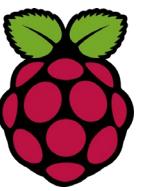
# Quizzzz

31) A combination of \_\_\_\_\_ resistances is often called string.

- a) Parallel Circuit
- b) Series Circuit
- c) Series Parallel Circuit
- d) all are correct

32) The formula to calculate the power dissipates against any Resistor is \_\_\_\_\_.

- a)  $P = W/Q$
- b)  $P = I^2 \times R$
- c)  $P = VR$
- d)  $P = IR$



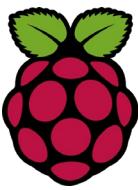
# Quizzzz

**33) The resistance of an open series circuit is \_\_\_\_\_.**

- a) High
- b) Low
- c) Extremely High (Infinity)
- d) Extremely Low

**34) The resistance of the short circuit (comparing to open circuit) is \_\_\_\_\_.**

- a) Zero
- b) Maximum
- c) Infinity
- d) Low



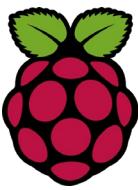
## Quizzzz

35) The \_\_\_\_\_ of a resistor is determined mainly by its physical size.

- a) Resistance
- b) Power
- c) Current
- d) a & b are correct

36) Resistor is an \_\_\_\_\_ component/device.

- a) Active
- b) Passive
- c) Both
- d) Sometimes active and sometimes passive



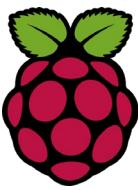
# Quizzzz

37) There are two main characteristics of a resistor are \_\_\_\_\_.

- a) Current and Voltage
- b) Current and Power
- c) Resistance and Power
- d) Resistance and Current

38) Resistors are generally available in the maximum value of \_\_\_\_\_ ohm.

- a) Mega ( $M\ \Omega$ )
- b) Gega ( $G\ \Omega$ )
- c) Kilo ( $K\ \Omega$ )
- d) Tera ( $T\ \Omega$ )



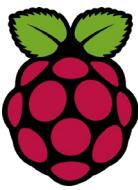
## Quizzzz

39) The power rating of a resistor should be \_\_\_\_\_ than from actual power dissipation (Waste) as heat for the reason of Safety factor (Burning).

- a) Less
- b) More
- c) Equal
- d) None of these

40) Most common type of Resistor is \_\_\_\_\_

- a) Wire Wound Resistor
- b) Carbon Resistor
- c) Film Type Resistor
- d) Fusible Resistor



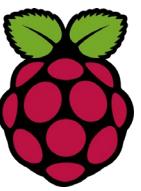
## Quizzzz

41) Resistors are \_\_\_\_\_ components.

- a) Polar
- b) Non-Polar
- c) Both
- d) None of these

42) Power Rating and Physical Size of Resistor is \_\_\_\_\_ proportional.

- a) Directly
- b) In-Directly
- c) Both
- d) None of these



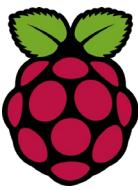
## Quizzzz

43) Resistors Carbon Composition Resistors are most popular because \_\_\_\_\_

- a) Low Price
- b) Small Size
- c) High inductance
- d) a & b are correct

44) Advantages of carbon composition resistor are \_\_\_\_\_

- a) Small Size
- b) Low Cost
- c) Withstand at high energy pulses
- d) All are Correct



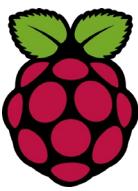
## Quizzzz

45) The value of Carbon Composition Resistor could be changed by \_\_\_\_\_

- a) If not in use for a year (5%)
- b) Soldering (Heat) – 3 %
- c) Operate at 70°C Temperature (15%)
- d) All are Correct

46) Disadvantages of carbon composition resistor are \_\_\_\_\_

- a) Instability of Resistance Value
- b) High Temperature Coefficient
- c) High Noise
- d) All are Correct



# Quizzzz

47) 1mA is equal to \_\_\_\_\_.

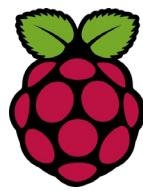
- a) 100 $\mu$ A
- b) 10 $\mu$ A
- c) 1000 $\mu$ A
- d) 10A

48) The electric shock of 0.1 to 0.2 Amp can cause the \_\_\_\_\_.

- a) Mild sensation
- b) Threshold of sensation
- c) Death
- d) Painful



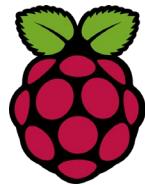
# Morse Code



- ✓ Morse code is a method used in telecommunication to encode text characters as standardized sequences of two different signal durations, called dots and dashes, or dits and dahs. Morse code is named after Samuel Morse, one of the inventors of the telegraph.
- ✓ Morse Code uses an alphabet made up of dots and dashes (for instance, the letter "s" is three dots and "o" is three dashes.) It is used by **tapping the combination of dots and dashes needed and pausing for the correct gap duration**. There are longer gaps between words than letters in a word.



# Morse Code: "SOS"



A	•—	N	—•	1	•————	?	••—•—••
B	—•••	O	————	2	••—•—	!	—••—•—
C	—•—•	P	•—•—•	3	••—•—	.	•—•—•—•
D	—••	Q	————•	4	•••—	,	—•—•—•—
E	•	R	—•	5	••••	;	—•—•—•—•
F	••—•	S	•••	6	—•••	:	—•—•—•—•—
G	—•—•	T	—	7	—•••	+	•—•—•—•
H	•••	U	••—	8	—••—•	-	—•••—•—
I	•..	V	••—	9	—••—•—	/	—••—•—•—
J	•—•—•	W	•—•—	0	—••—•—	=	—••—•—•—
K	—•—	X	—•••				
L	•—••	Y	—•—•				
M	—•—	Z	—•••				



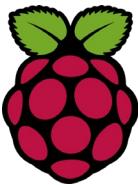
## What Does "SOS" Mean?

### SOS Meaning

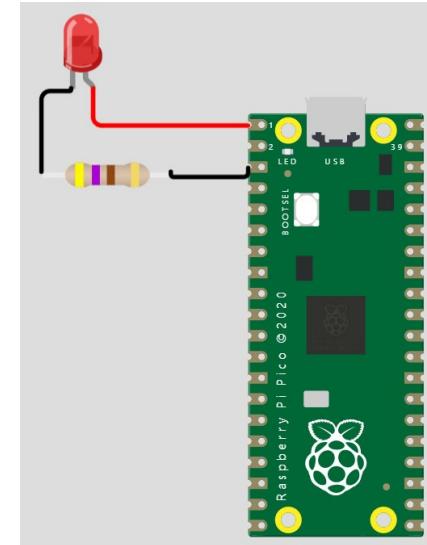
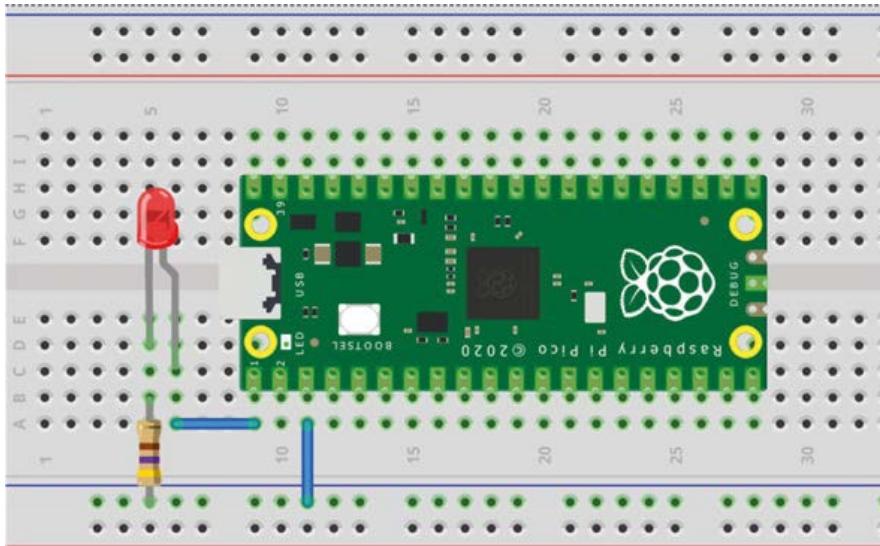
- However, meaning 'Save Our Ship' the phrase 'sos' is more widely known as another way of saying 'please help me now'. On a few occasions and depending on the lore behind each story it has also been known to mean 'Save Our Souls'.



Try to implement a Morse code system by flashing SOS with the help of Electronics Components



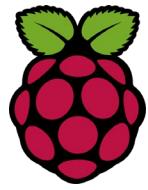
# Flashing SOS in Morse



- ✓ An external LED flashes the SOS signal in **Morse code (three dots, followed by three dashes, followed by three dots)** continuously.
- ✓ In this experiment, a dot is represented with the LED being ON for **0.25 seconds (Dot time)** and a dash is represented with the LED being ON for **1 second (Dash time)**.
- ✓ The delay between the dots and dashes is set to **0.2 second (GAP time)**. This process is repeated continuously after **2 seconds of delay**.



# LED FLASHING SOS



WOKWi

SAVE

SHARE



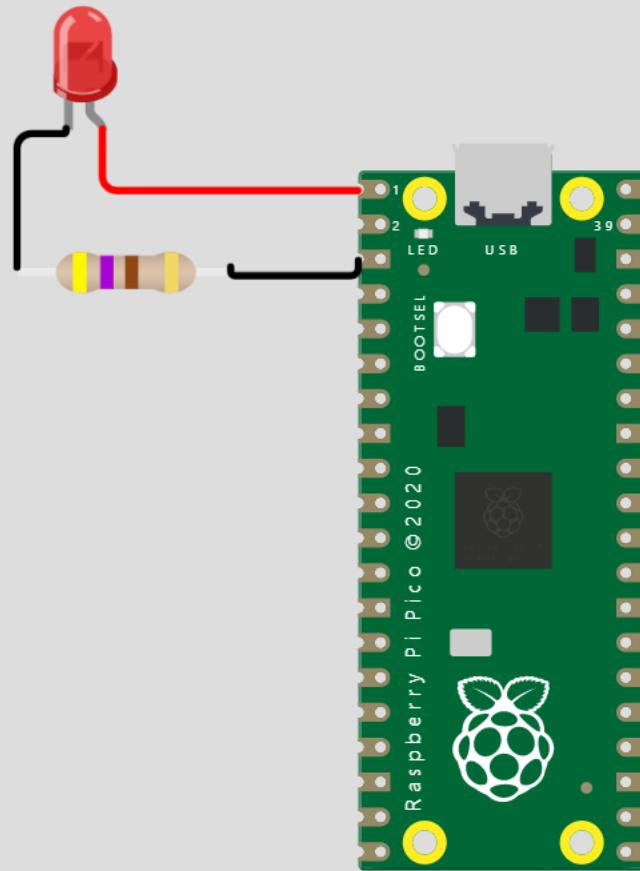
Docs

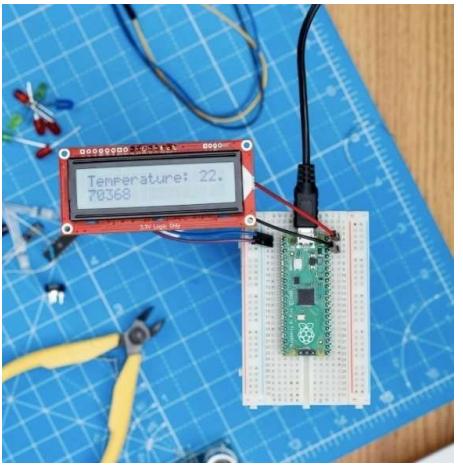
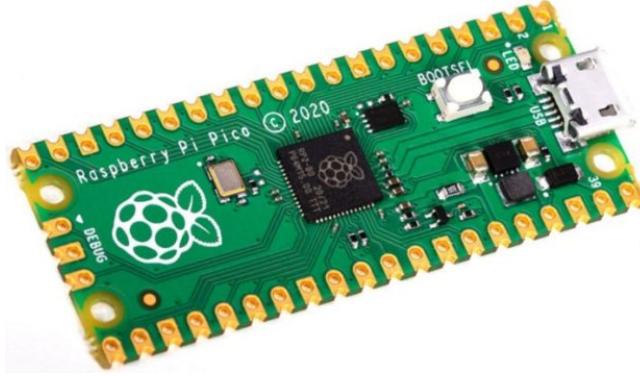


diagram.json • main.py Library Manager

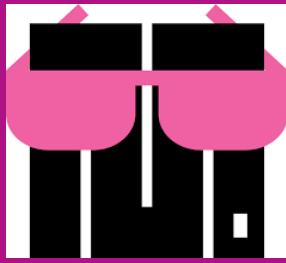
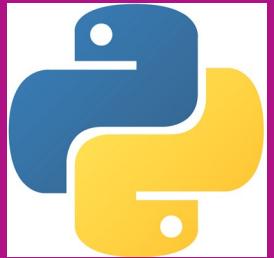
```
1  from machine import Pin
2  from utime import sleep
3
4  Dot = 0.25          # Dot time
5  Dash = 1.0          # Dash time
6  Gap = 0.2           # Gap time
7  ON = 1              # ON
8  OFF = 0             # OFF
9  LED = Pin(0, Pin.OUT) # LED at GP0
10
11 while True:          # DO FOREVER (INFINITE LOOP)
12     for i in range(0, 3):
13         LED.value(ON)      # LED ON
14         sleep(Dot)        # Wait Dot time
15         LED.value(OFF)    # LED OFF
16         sleep(Gap)        # Wait Gap time
17
18         sleep(0.5)        # 0.5 second delay
19
20     for i in range(0, 3):
21         LED.value(ON)      # LED ON
22         sleep(Dash)       # Wait Dash time
23         LED.value(OFF)    # LED OFF
24         sleep(Gap)        # Wait Gap time
25         sleep(2)          # Wait 2 seconds
```

Simulation



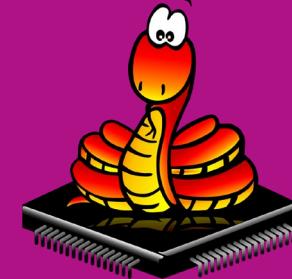


# INTERNET OF THINGS (IOT) PROJECTS USING PYTHON

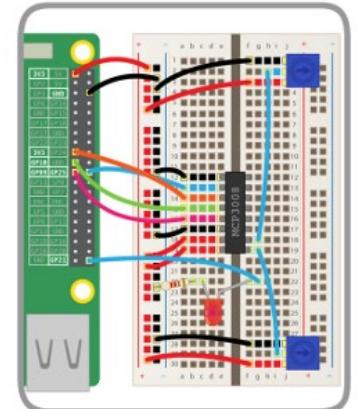
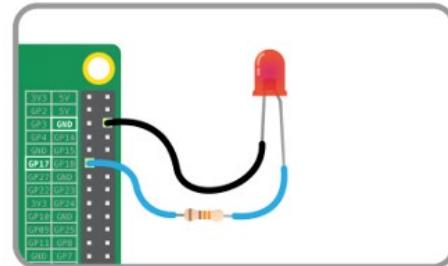
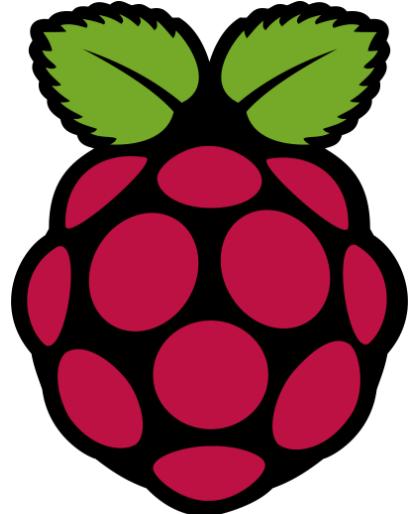
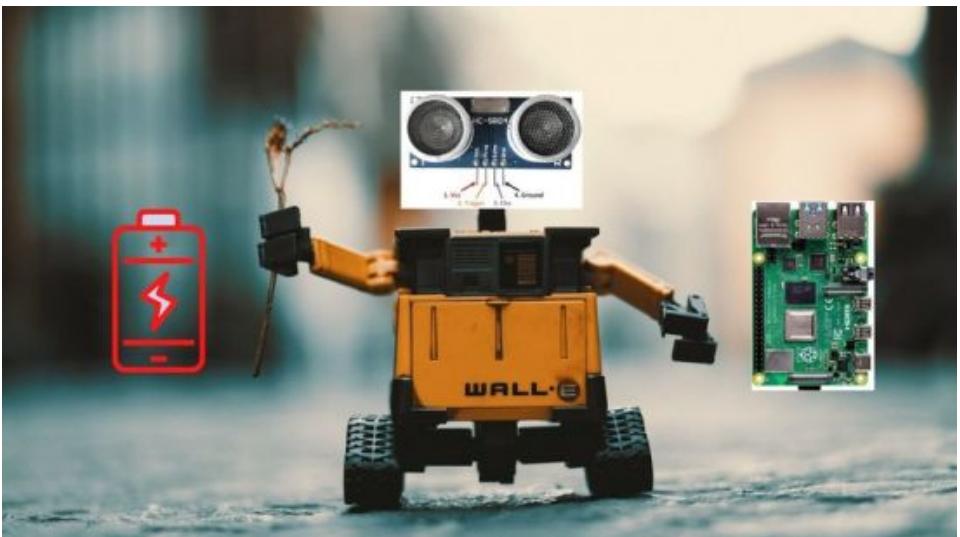


(CSE 4110)

(LECTURE – 3)

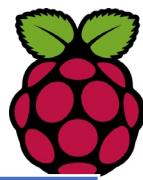


T<sub>h</sub>





# Course Outcomes

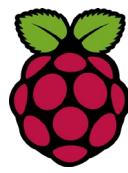


## Course Outcomes

CO1	Understand general concepts of Internet of Things (IoT), the working of Raspberry Pi and its features.
CO2	Recognize various components, sensors, actuators, devices and their applications.
CO3	Analyze various python programs to interface with sensors, actuators, LED's, cloud and camera using Raspberry pi.
CO4	Measure physical parameters using sensors.
CO5	Demonstrate the ability to transmit data wirelessly between different devices to build simple IoT systems using Raspberry Pi.
CO6	Create IoT devices and systems through a variety of interfaces, including web apps and mobile apps.



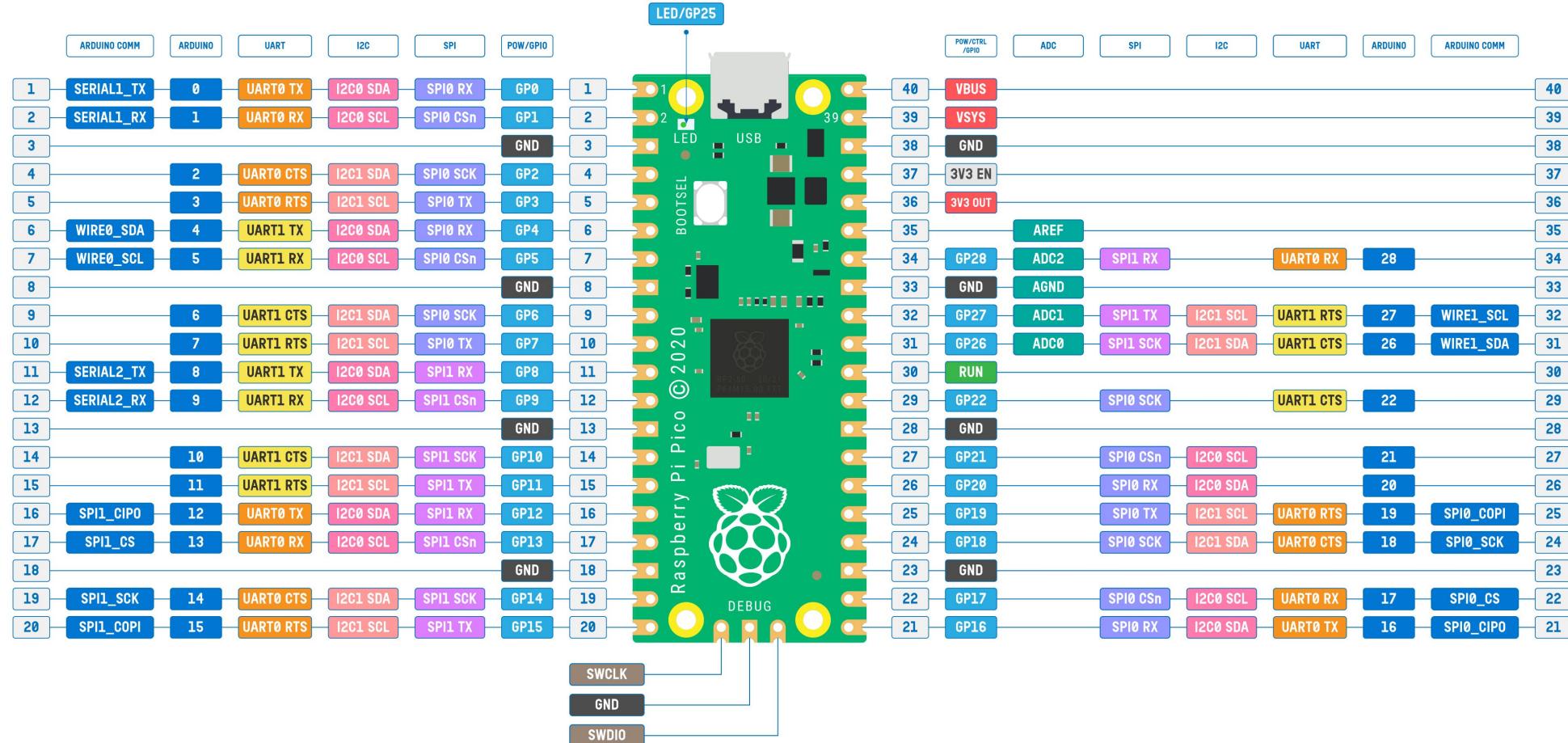
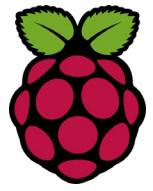
# What Students will learn?



Students will be developing simple hardware projects/experiments with the Raspberry Pi Pico, using the Thonny text editor. They also learn how a Raspberry Pi Pico can be used to control and receive input from a variety of electronic components to engage in some creative physical computing experiments/projects.

- ✓ Implementation of Flashing LED using a Timer using Raspberry Pi Pico.
- ✓ Implementation of Alternately Flashing two different colour LEDs using Raspberry Pi Pico.
- ✓ Implementation of four LEDs display a pattern of rotating left using Raspberry Pi Pico

# Raspberry Pi Pico – Full Pinout



\*Raspberry Pi and the Raspberry Pi logo are trademarks of Raspberry Pi Ltd.

Raspberry Pi Pico vector image is originally designed by Raspberry Pi. Please visit [raspberrypi.com](https://www.raspberrypi.com) for more info.

## ARDUINO PINS

## SWD Pins

PHYSICAL PIN	POSITIVE SUPPLY	UART1 Pins	UART0 Pins
RESET/ENABLE	GROUND SUPPLY	I2C1 Pins	I2C0 Pins
GPIO PORT/PIN	ANALOG PIN	SPI1 Pins	SPI0 Pins

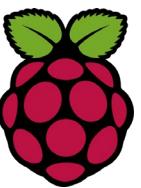
- GP29/ADC3 is used to measure VSYS.
- GP25 is used for debug LED.
- GP24 is used for VBUS sense.
- GP23 is connected to SMPS Power Save pin.
- All GPIO pins support PWM. There are total 16 PWM channels.
- All GPIO pins support level and edge interrupts.
- Arduino pins are as per [Arduino-Pico core](#) by [Earle F. Philhower, III @earlephilhower](mailto:Earle F. Philhower, III @earlephilhower)
- Arduino's default Serial is the USB-CDC of Pico.



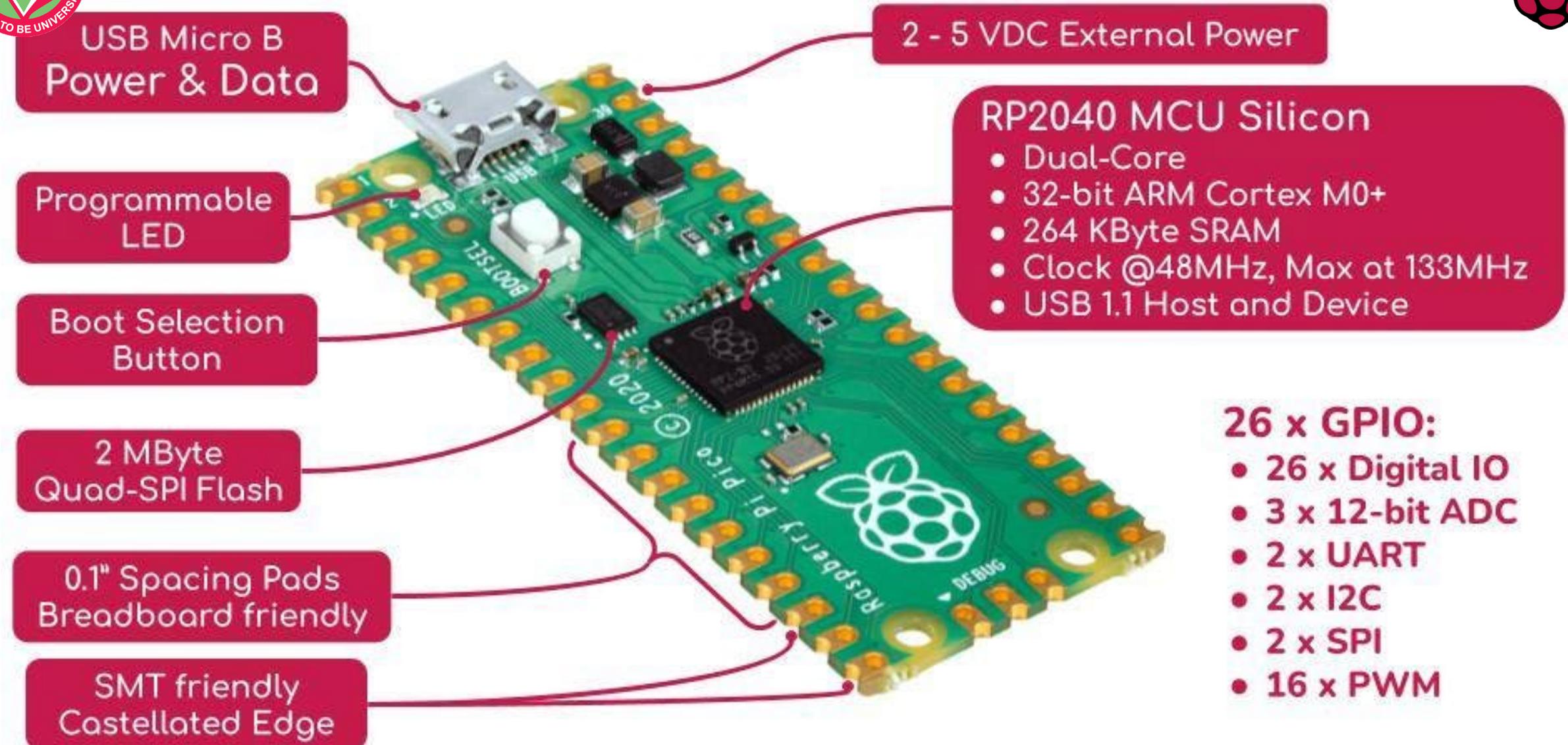
Rev. 0.2, 15-07-2022

Design: Vishnu Mohanan

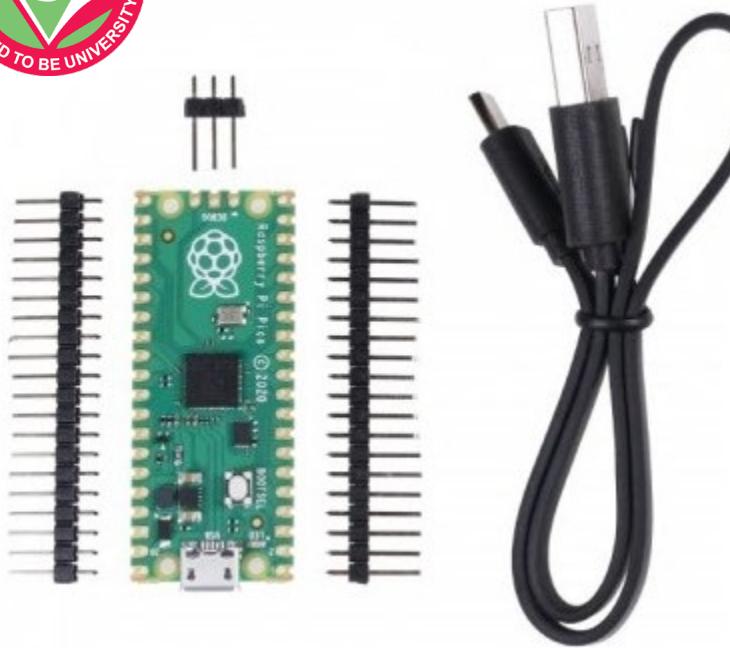
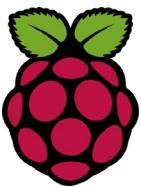
This work is licensed under a Creative Commons  
Attribution 4.0 International License.



# Raspberry Pi Pico – In Short

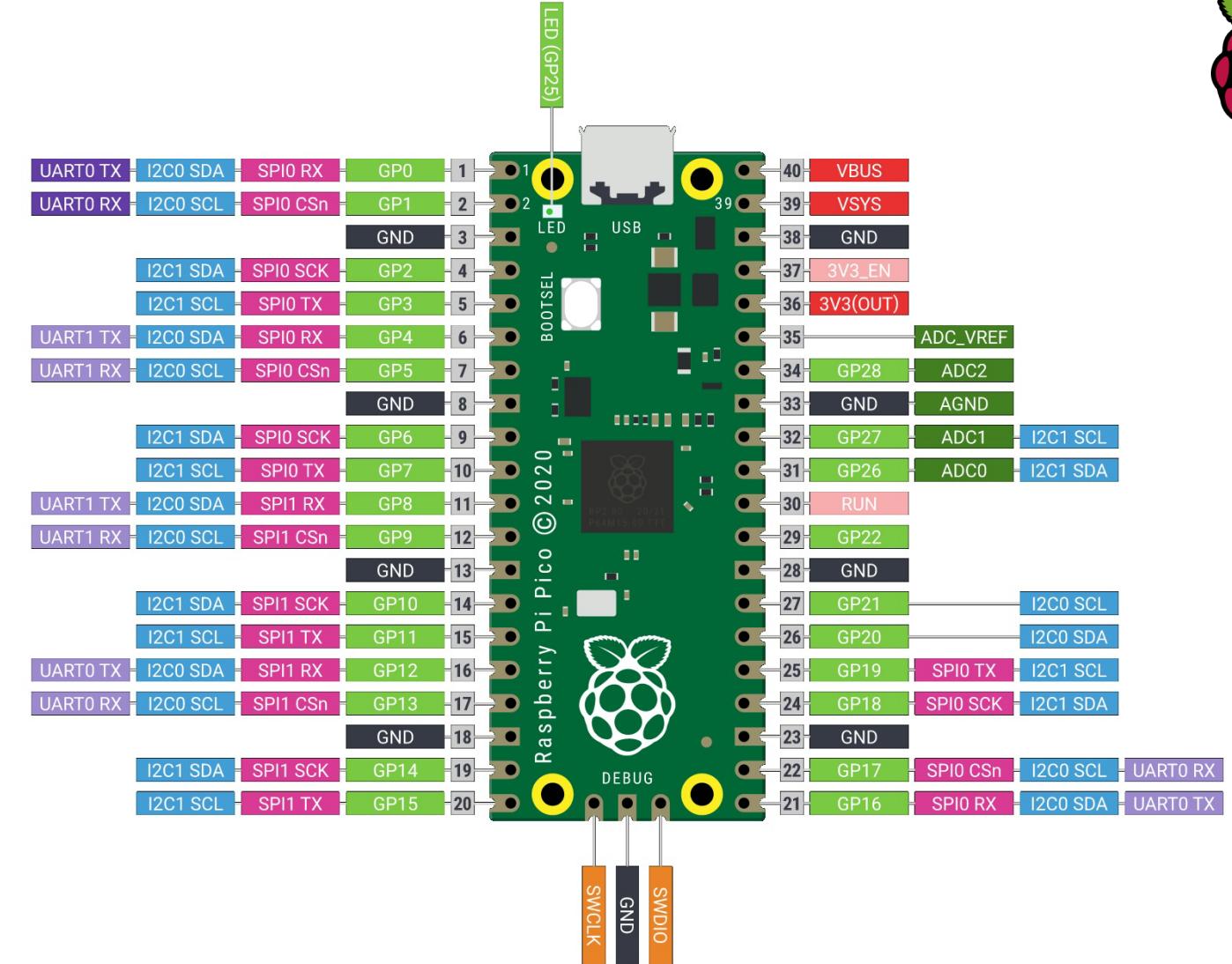


Pi Foundation has released an RP2040 Microprocessor based development board, in the same form factor as an Arduino Nano.

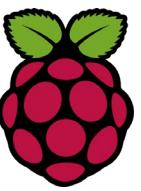


## Package Includes:

- 1 x Raspberry Pi Pico
- 1 x Micro-USB cable
- 2 x 20 Pin Header
- 1 x 3 Pin Header

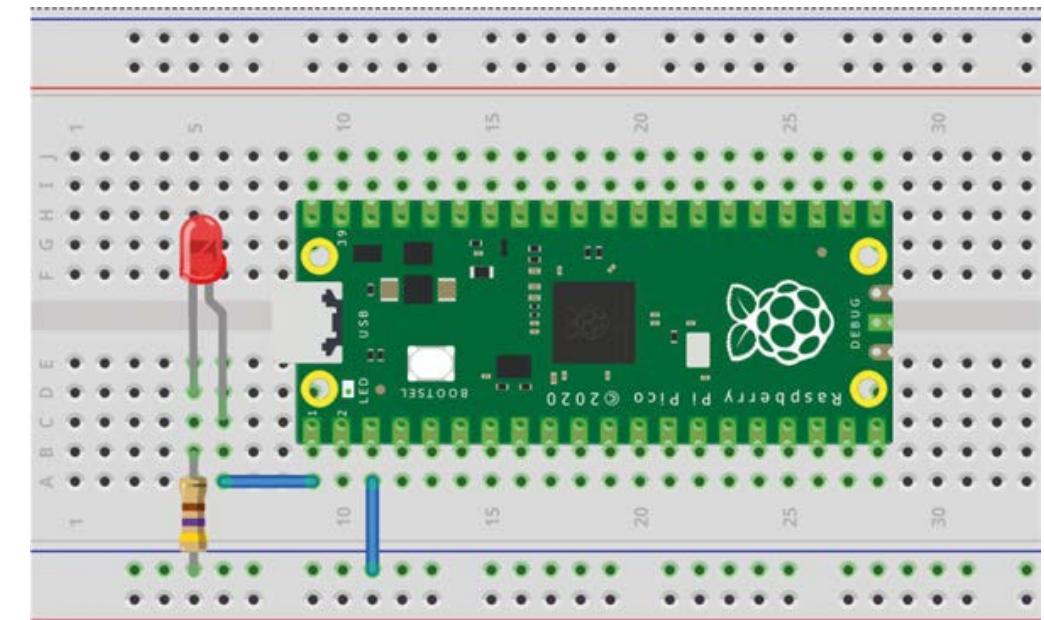
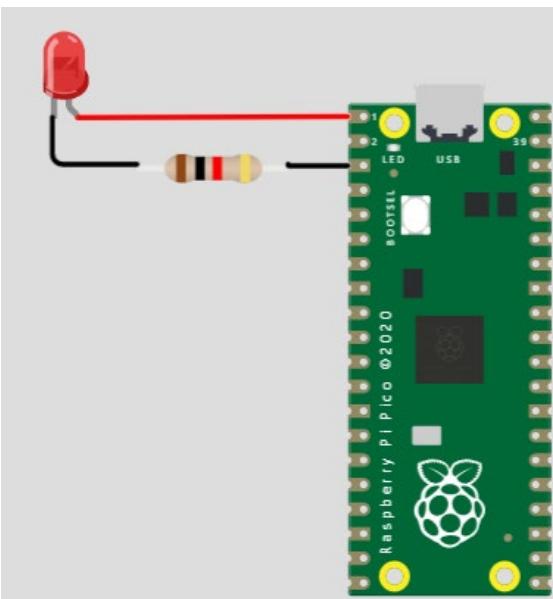


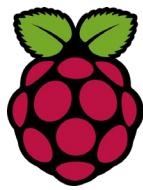
■ Power ■ Ground ■ UART / UART (default) ■ GPIO, PIO, and PWM ■ ADC ■ SPI ■ I2C ■ System Control ■ Debugging



# Flashing LED – using a timer

- ✓ An external LED is connected to port pin GP0 of the Pico. In this objective a timer is used to flash the LED every 500 ms.
- ✓ A timer is initialized which calls function **Flash\_LED** twice (**freq = 2.0**) a second in a periodic manner.
- ✓ LED is flashed using the **toggle** function.





# Flashing LED – using a timer

WOKWi

SAVE

SHARE

Docs

B

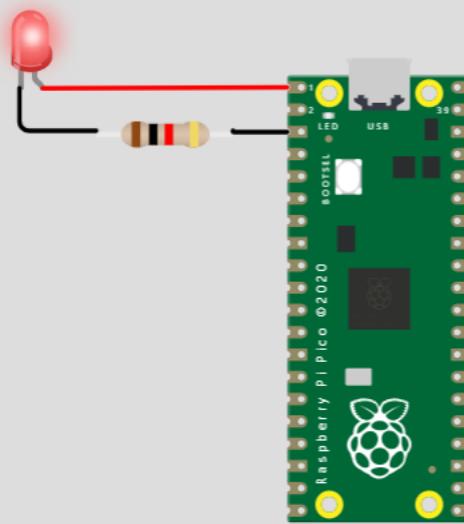
main.py • diagram.json • PIO 1

```
1 from machine import Pin, Timer
2 LED = Pin(0, Pin.OUT)
3
4 tim = Timer()
5 def Flash_LED(timer):
6     global LED
7     LED.toggle()
8 tim.init(freq = 2.0, mode = Timer.PERIODIC, callback = Flash_LED)
```

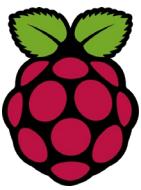
Simulation



⌚ 00:13.696 📺 90%



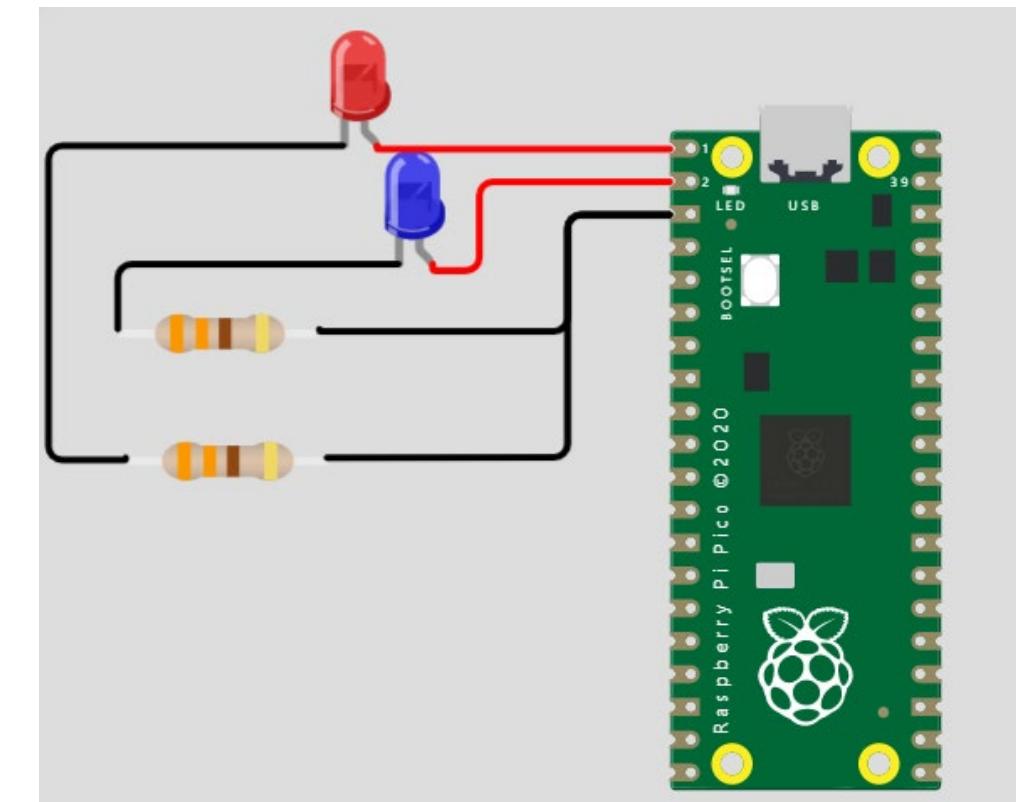
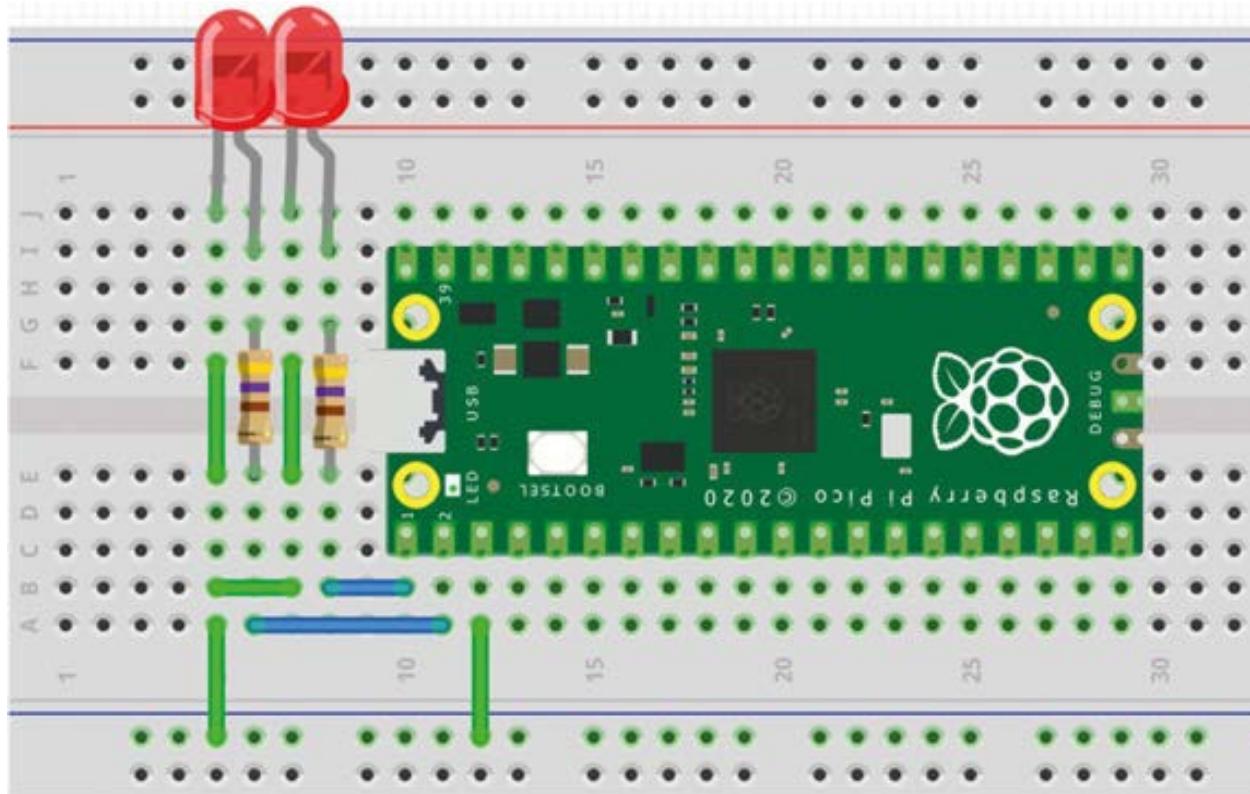
```
MPY: soft reboot
MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
```



# Alternately flashing LEDs

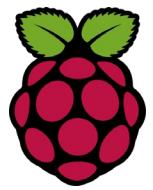
Two LEDs are connected to the Pico to pins GP0 (pin 1) and GP1 (pin 2).

The LEDs flash alternately every 500 ms.





# Alternately flashing LEDs

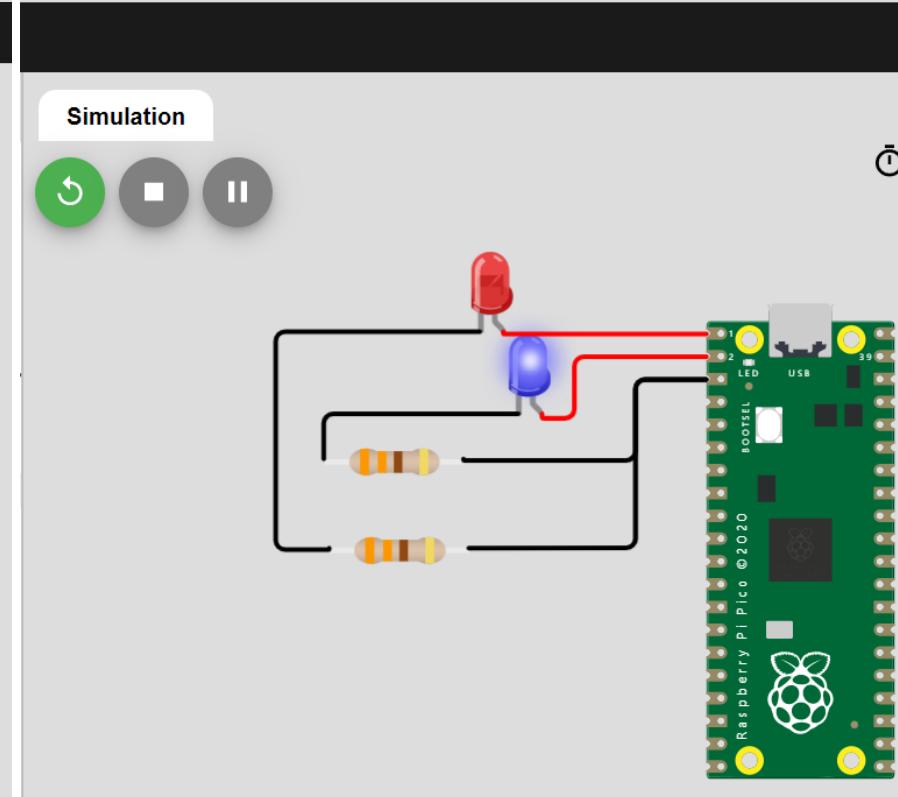
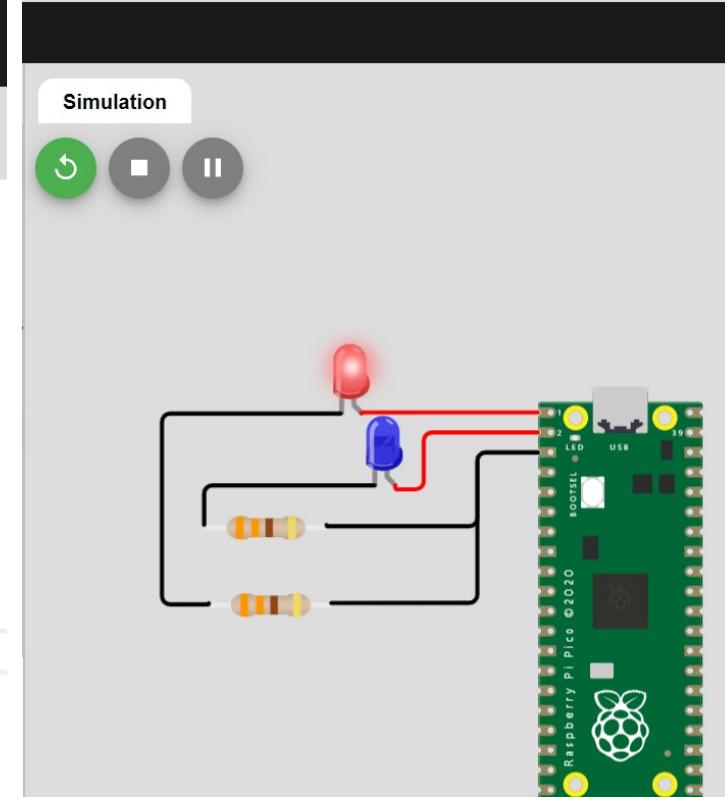


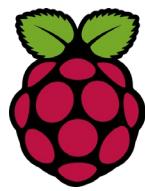
WOKWi

SAVE ▾ SHARE

main.py • diagram.json • PIO 🐍

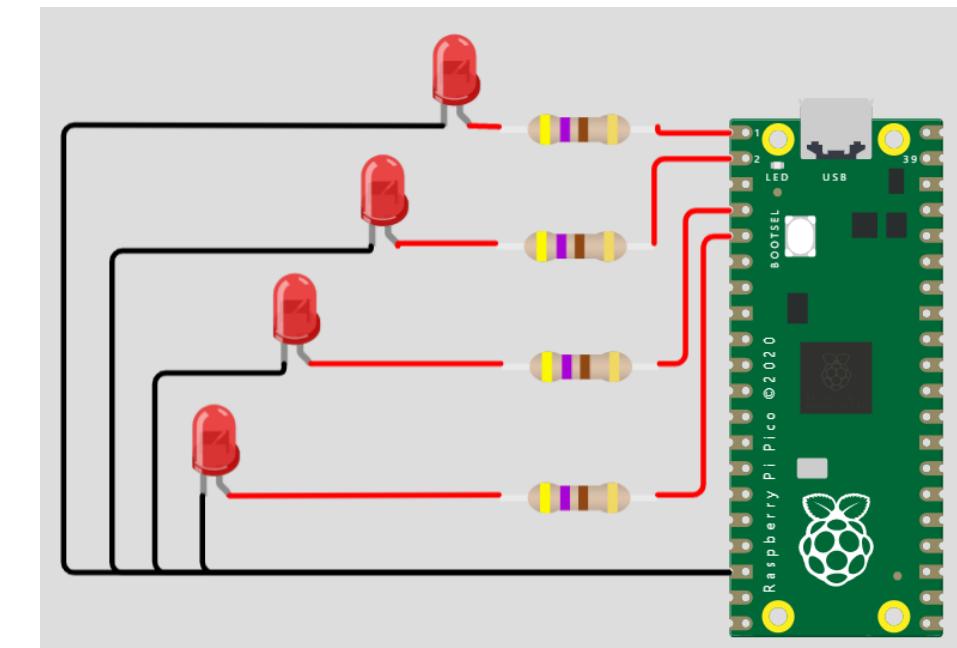
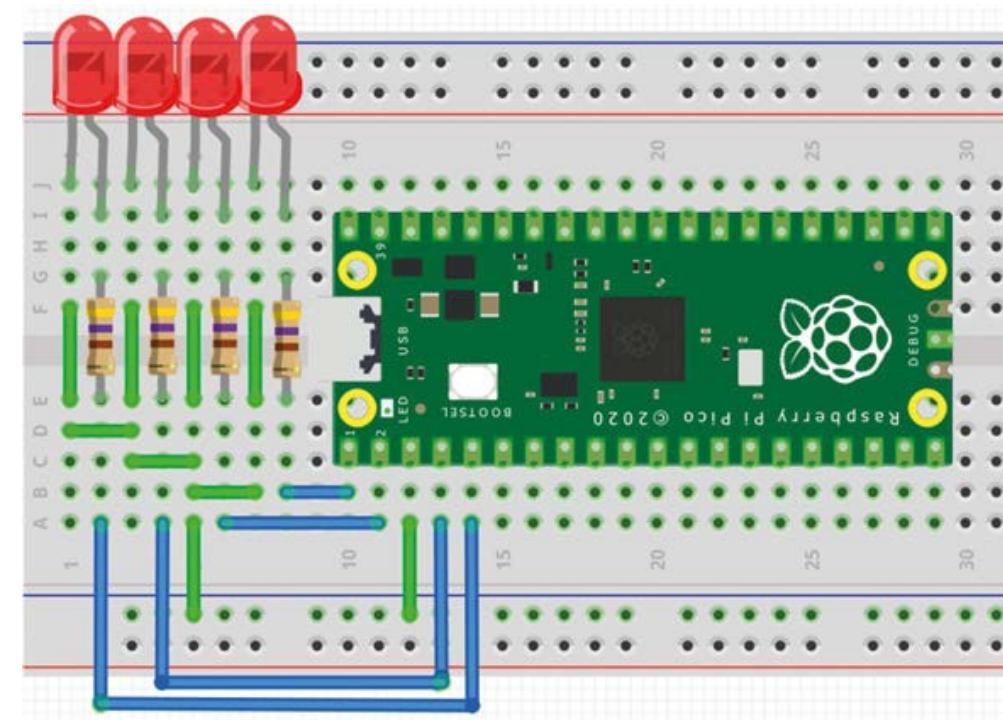
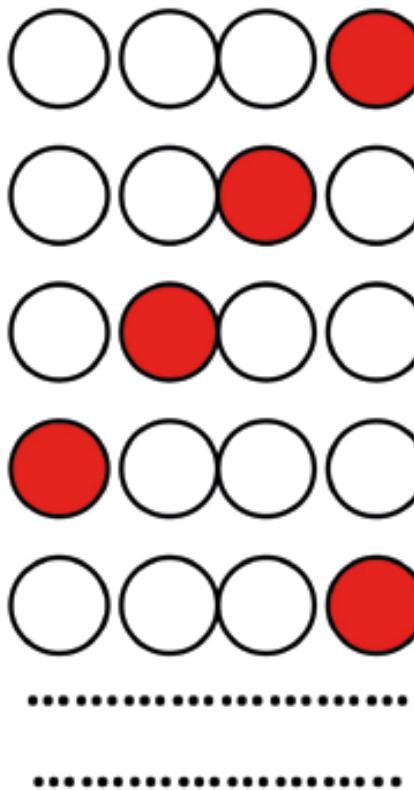
```
1 from machine import Pin
2 import utime
3 LED1 = Pin(0, Pin.OUT)
4 LED2 = Pin(1, Pin.OUT)
5 while True:
6     LED1.value(1)
7     LED2.value(0)
8     utime.sleep(0.5)
9     LED1.value(0)
10    LED2.value(1)
11    utime.sleep(0.5)
12
```

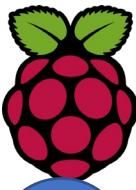




# ROTATING LEDs

4 LEDs are connected from port pin GP0 to GP3 of the Pico. The LEDs display pattern of rotating to the left





# ROTATING LEDs

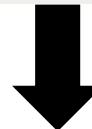
WOKwi

SAVE

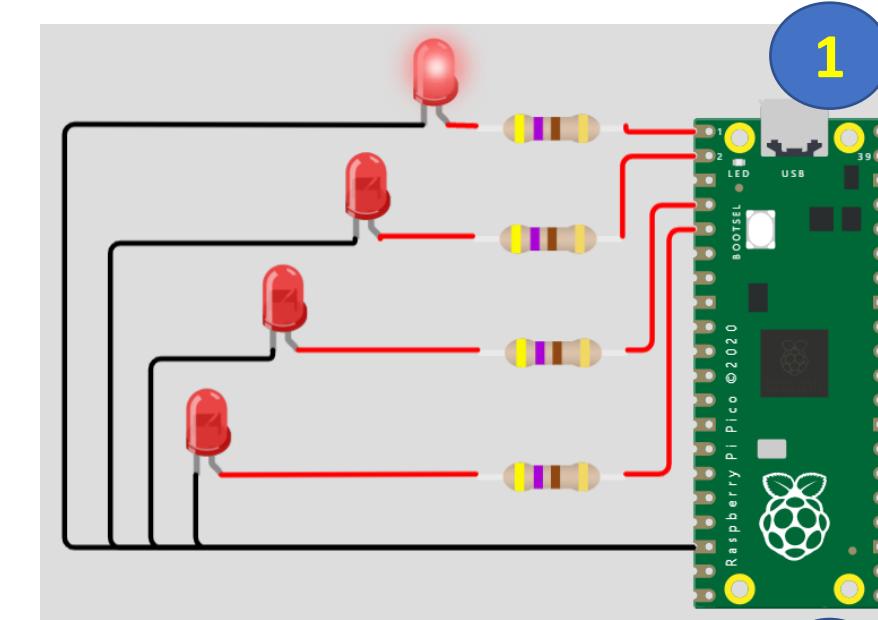
SHARE

main.py • diagram.json • PIO

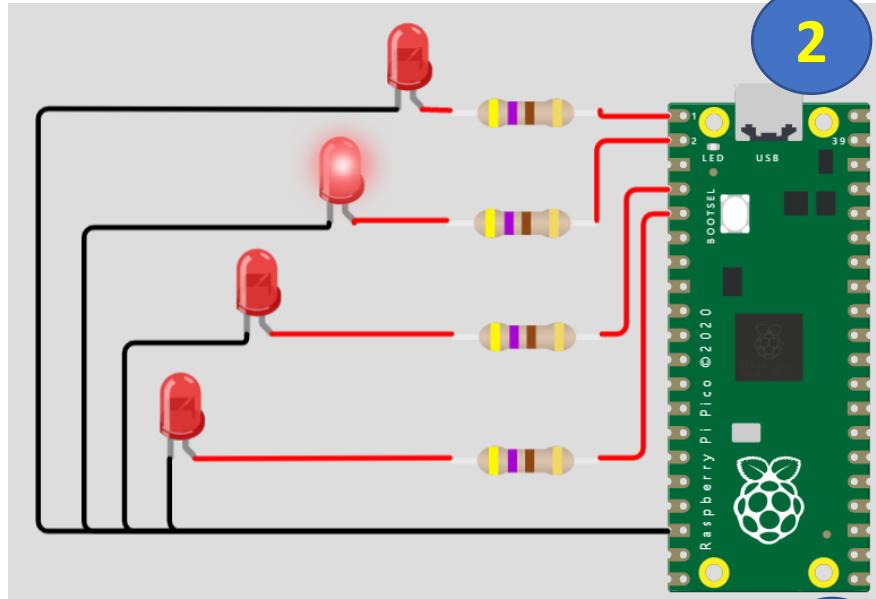
```
1  from machine import Pin
2  import utime
3  LED1 = Pin(0, Pin.OUT)
4  LED2 = Pin(1, Pin.OUT)
5  LED3 = Pin(2, Pin.OUT)
6  LED4 = Pin(3, Pin.OUT)
7
8  while True:
9      LED1.value(1)
10     utime.sleep(0.5)
11     LED1.value(0)
12     LED2.value(1)
13     utime.sleep(0.5)
14     LED2.value(0)
15     LED3.value(1)
16     utime.sleep(0.5)
17     LED3.value(0)
18     LED4.value(1)
19     utime.sleep(0.5)
20     LED4.value(0)
```



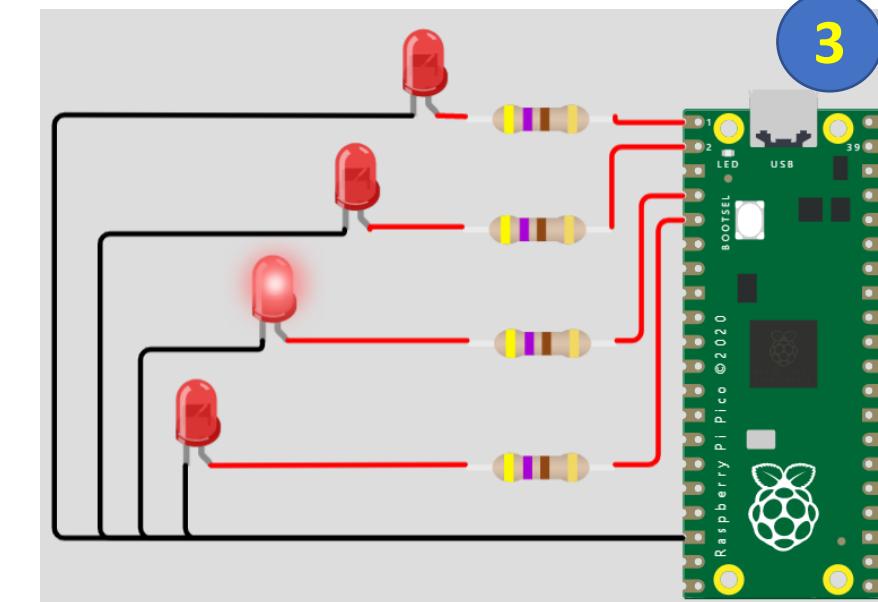
Can you write a more efficient program??



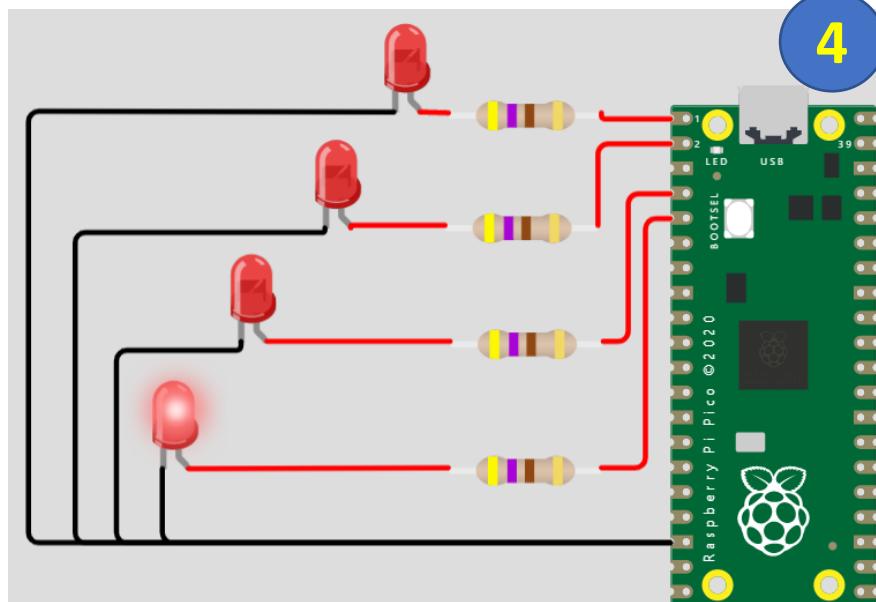
1



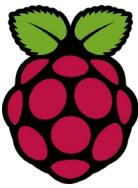
2



3



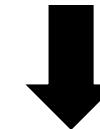
4



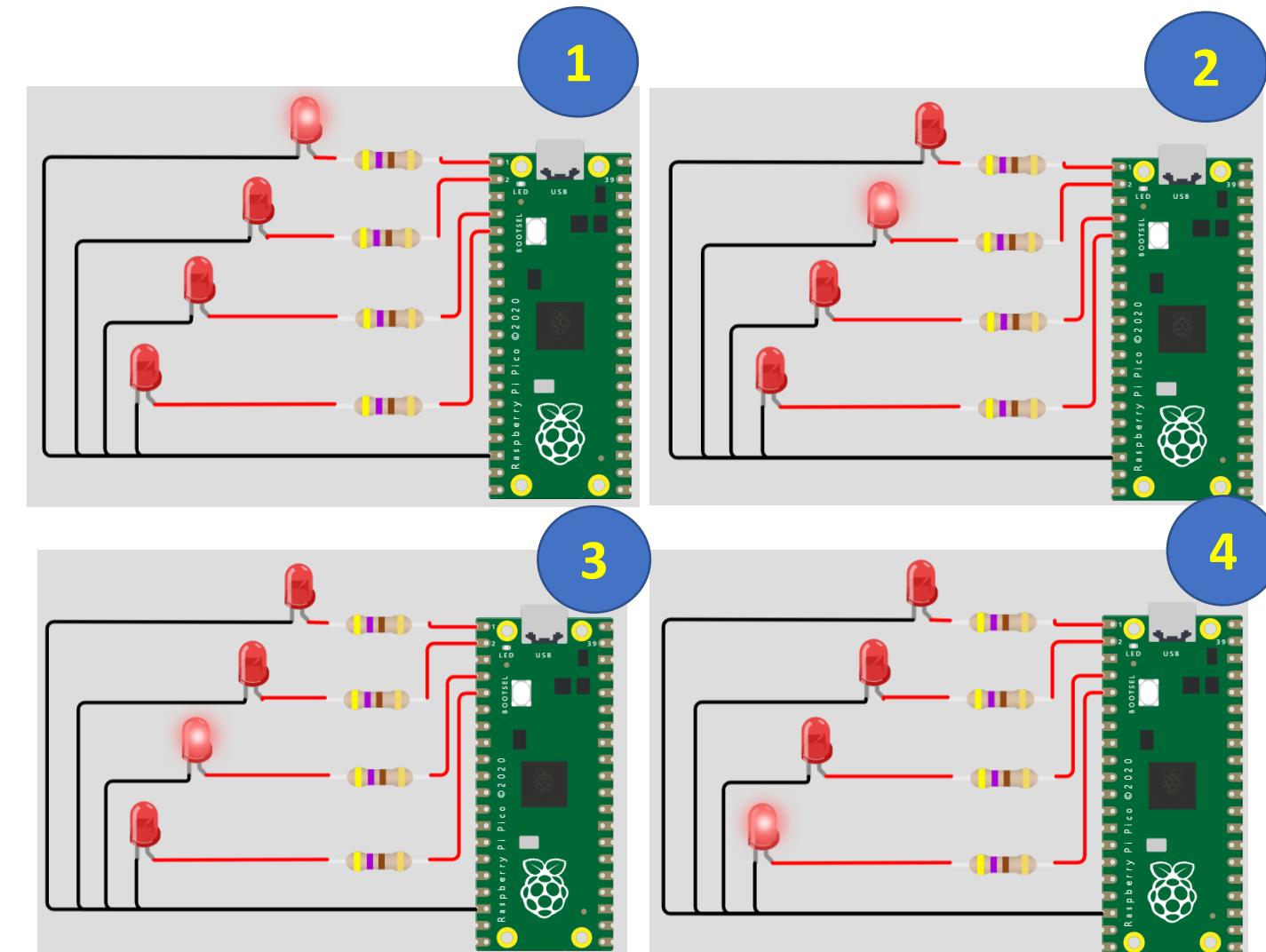
# EFFICIENT ROTATING LEDs

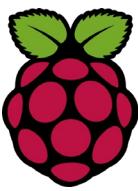
WOKWi ● diagram.json ●

```
main.py • diagram.json •
1  from machine import Pin
2  import utime
3
4  LEDS = [0, 1, 2, 3]          # LED ports
5  L = [0, 0, 0, 0]
6
7  for i in range(4):           # Do for all LEDs
8      L[i] = Pin(LEDS[i], Pin.OUT) # All are outputs
9
10 while True:                  # Do forever
11     for i in range(4):         # LED ON
12         L[i].value(1)          # Wait 0.5 second
13         utime.sleep(0.5)
14         L[i].value(0)          # LED OFF
```



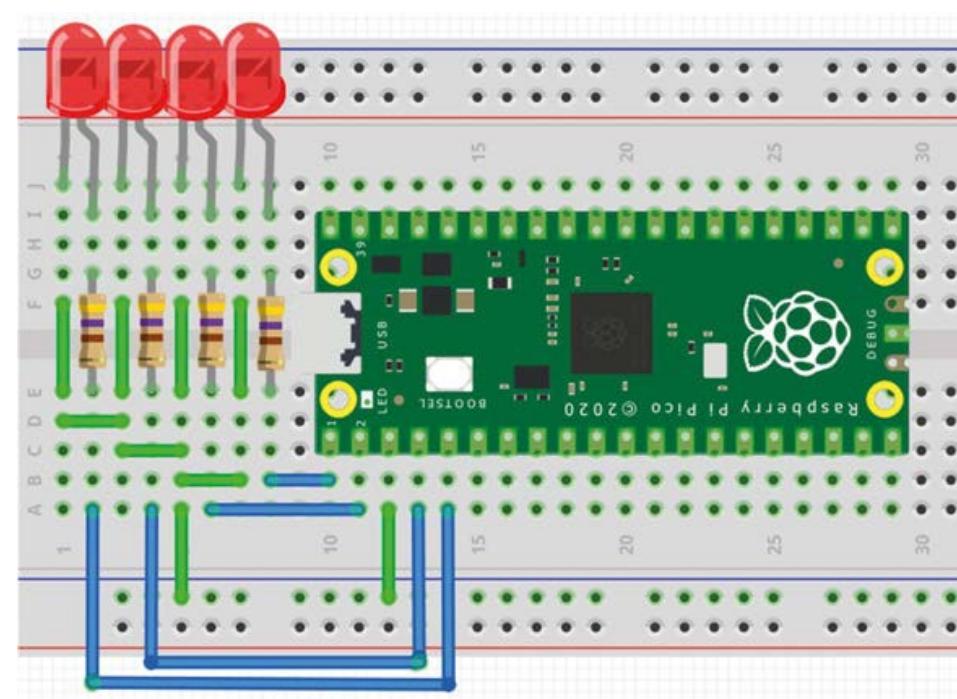
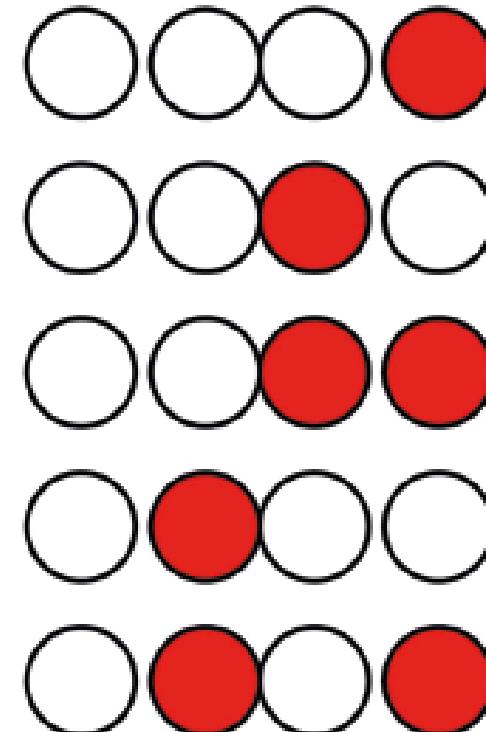
This is especially true if there  
are more than 4 LEDs.

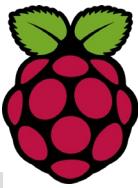




# 4-bit Binary-counting LEDs

- ✓ 4 LEDs are connected to Pico.
- ✓ The LEDs count up in binary every second from 0 to 15.



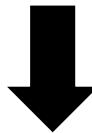


# 4-bit Binary-counting LEDs

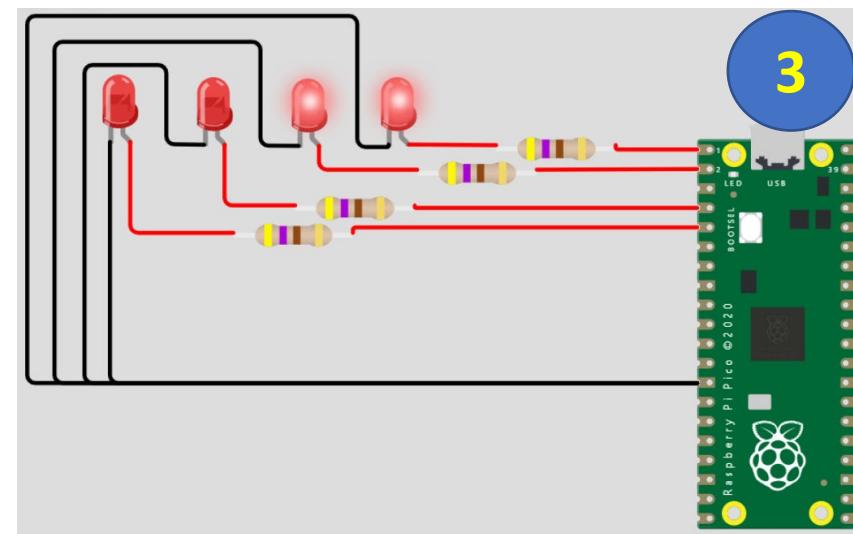
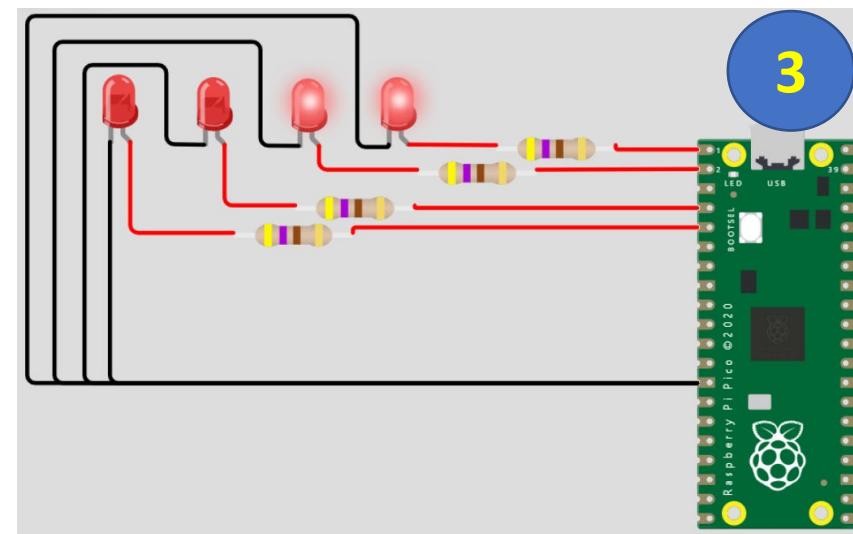
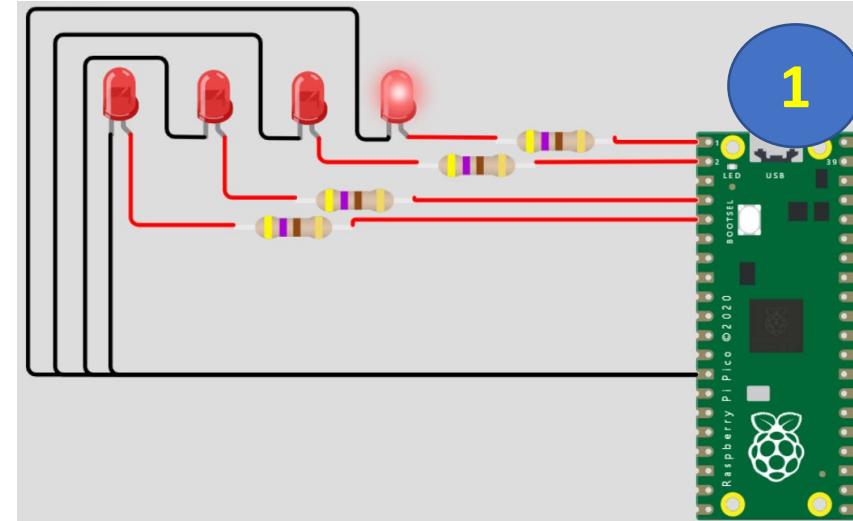
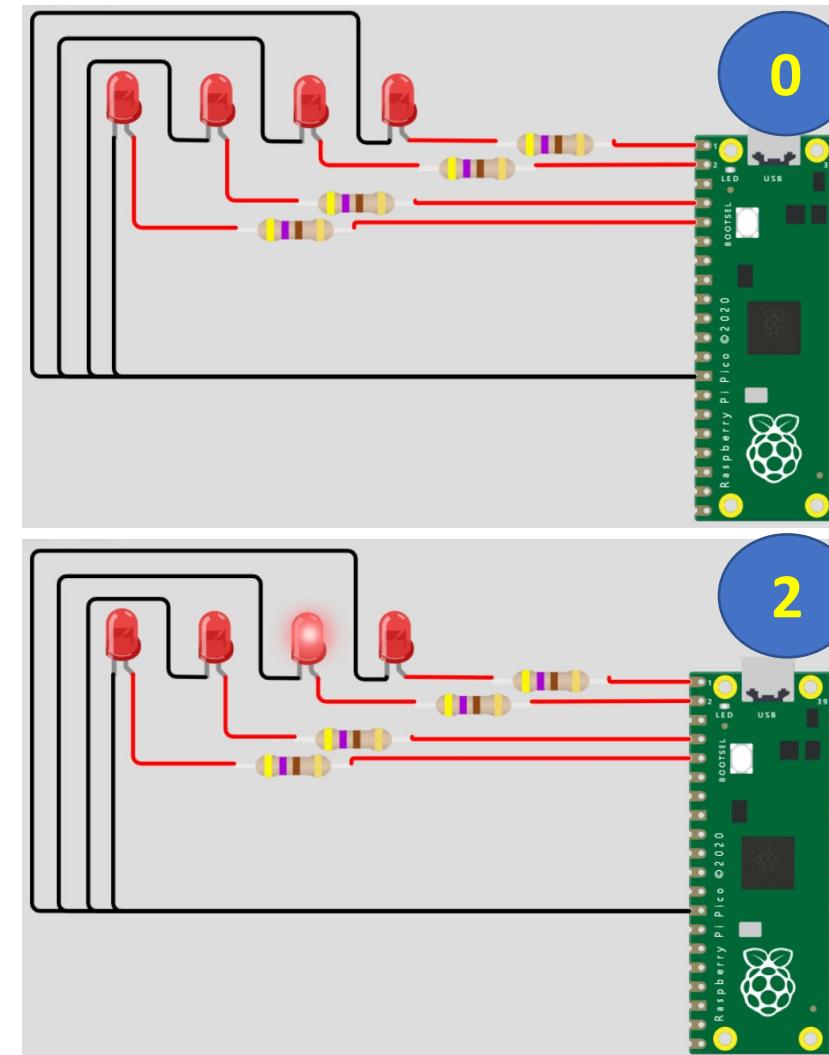
WOKWi

main.py diagram.json

```
1 from machine import Pin
2 from utime import sleep
3
4 led1 = Pin(0, Pin.OUT)
5 led2 = Pin(1, Pin.OUT)
6 led3 = Pin(2, Pin.OUT)
7 led4 = Pin(3, Pin.OUT)
8
9 while True:
10     led4.value(0)
11     led3.value(0)
12     led2.value(0)
13     led1.value(0)
14     sleep(1)
15
16     led4.value(0)
17     led3.value(0)
18     led2.value(0)
19     led1.value(1)
20     sleep(1)
21
22     led4.value(0)
23     led3.value(0)
24     led2.value(1)
25     led1.value(0)
26     sleep(1)
```



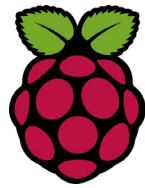
Can you write a more  
efficient program??



and so on.....



# Efficient 4-bit Binary-counting LEDs



WOKWi

SAVE



SHARE



main.py • diagram.json

```
1 from machine import Pin
2 import utime
3
4 PORT = [3, 2, 1, 0]          # port connections
5 DIR = ["0","0","0","0"]       # port directons
6 L = [0]*4
7 #
8 # This function configures the port pins as outputs ("0") or
9 # as inputs ("I")
10 #
11 def Configure_Port():
12     for i in range(0, 4):
13         if DIR[i] == "0":
14             L[i] = Pin(PORT[i], Pin.OUT)
15         else:
16             L[i] = Pin(PORT[i], Pin.IN)
17     return
18 #
19 # This function sends 4-bit data (0 to 15) to the PORT
20 #
21 def Port_Output(x):
22     b = bin(x)                  # convert into binary
23     b = b.replace("0b", "")      # remove leading "0b"
24     diff = 4 - len(b)           # find the length
25     for i in range (0, diff):
26         b = "0" + b             # insert leading os
27
28     for i in range (0, 4):
29         if b[i] == "1":
```

WOKWi

SAVE



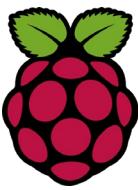
SHARE



main.py • diagram.json

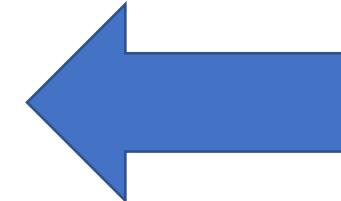
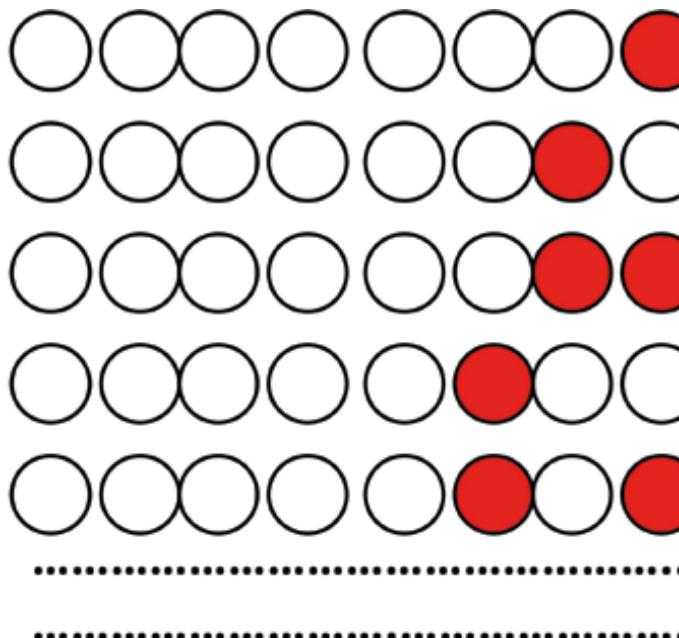
```
30     L[i].value(1)
31     else:
32         L[i].value(0)
33     return
34
35 # Configure PORT to all outputs
36 #
37 Configure_Port()
38 # Main program loop. Count up in binary every second
39 #
40 cnt = 0
41 while True:
42     Port_Output(cnt)          # send cnt to port
43     utime.sleep(1)            # wait 1 second
44     cnt = cnt + 1             # increment cnt
45     if cnt > 15:
46         cnt = 0
```

and so on.....

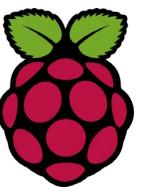


# 8-bit Binary-counting LEDs

- ✓ 8 LEDs are connected to Pico.
- ✓ The LEDs count up in binary every second from 0 to 255.



**Try to implement**



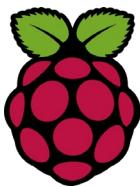
# Fancy Diwali/Christmas lights

- ✓ 8 LEDs are connected to Pico.
- ✓ The LEDs flash randomly every 250 milliseconds just like Diwali Light or fancy Christmas lights.

**Clue:**

**Generate random numbers between 1 and 255 and then shows how to use these numbers to turn the individual LEDs ON and OFF randomly.**

**Try to implement by yourself**



# Fancy Diwali/Christmas lights

WOKWi SAVE SHARE

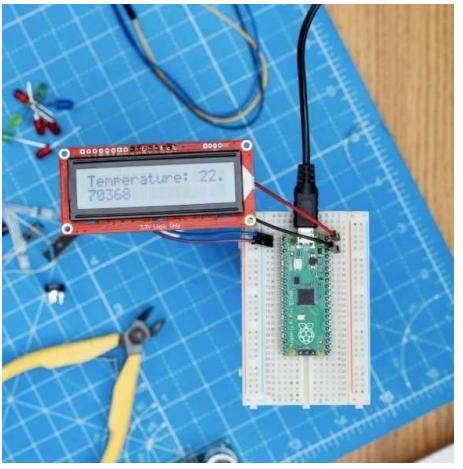
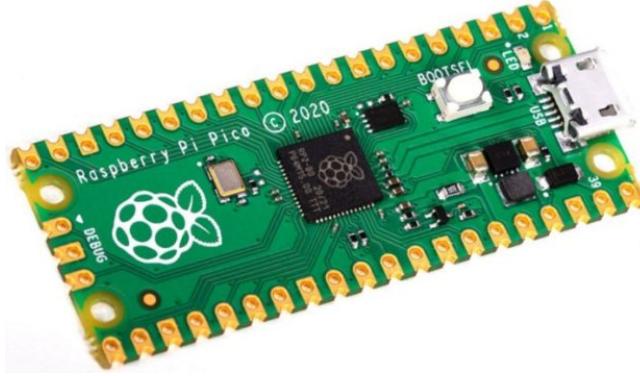
main.py • diagram.json •

```
1  from machine import Pin
2  import utime import sleep
3  import random
4
5  PORT = [7, 6, 5, 4, 3, 2, 1, 0]
6  DIR = ["0","0","0","0","0","0","0","0"]
7  L = [0]*8
8
9  def Configure_Port():
10     for i in range(0, 8):
11         if DIR[i] == "0":
12             L[i] = Pin(PORT[i], Pin.OUT)
13         else:
14             L[i] = Pin(PORT[i], Pin.IN)
15     return
16
17  def Port_Output(x):
18      b = bin(x)
19      b = b.replace("0b", "")
20      diff = 8 - len(b)
21      for i in range (0, diff):
22          b = "0" + b
23
24  for i in range (0, 8):
25      if b[i] == "1":
26          L[i].value(1)
```

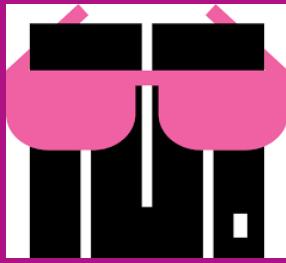
WOKWi SAVE SHARE

main.py • diagram.json •

```
27  else:
28      | L[i].value(0)
29  return
30
31  Configure_Port()
32
33  while True:
34      numbr = random.randint(1, 255)
35      Port_Output(numbr)
36      sleep(0.25)
```

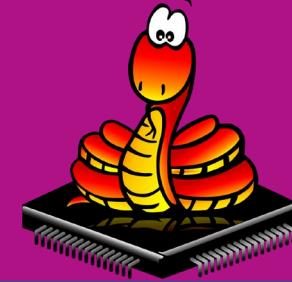


# INTERNET OF THINGS (IOT) PROJECTS USING PYTHON

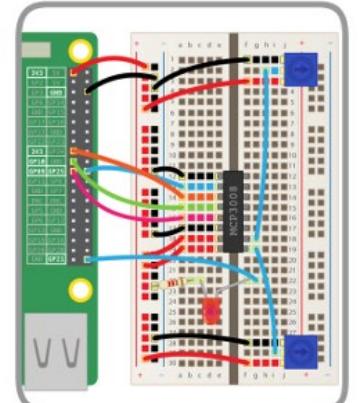
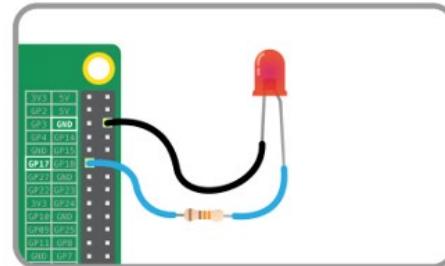
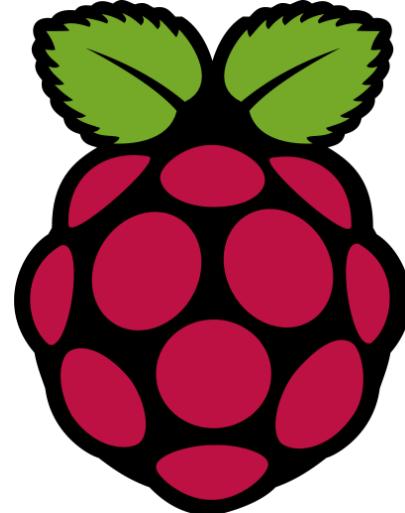
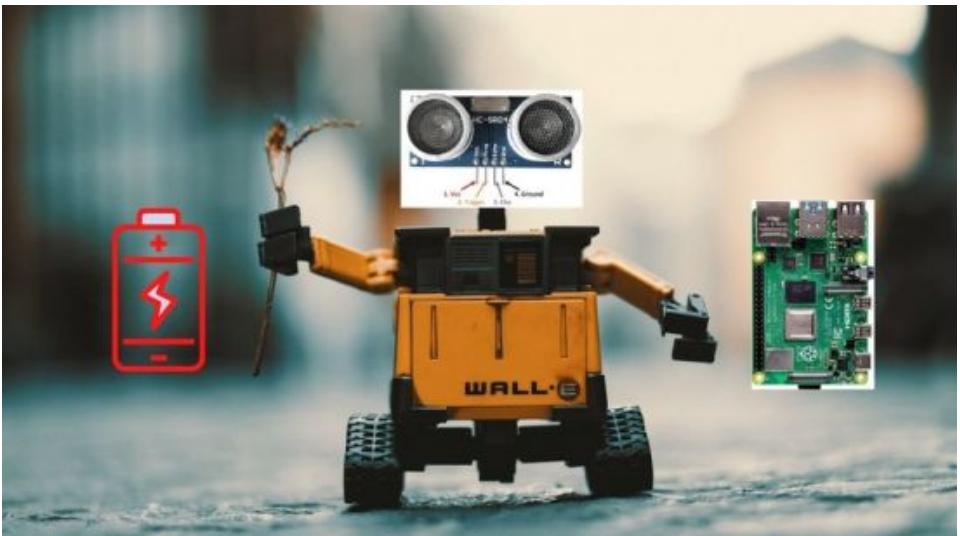


(CSE 4110)

(LECTURE – 5)



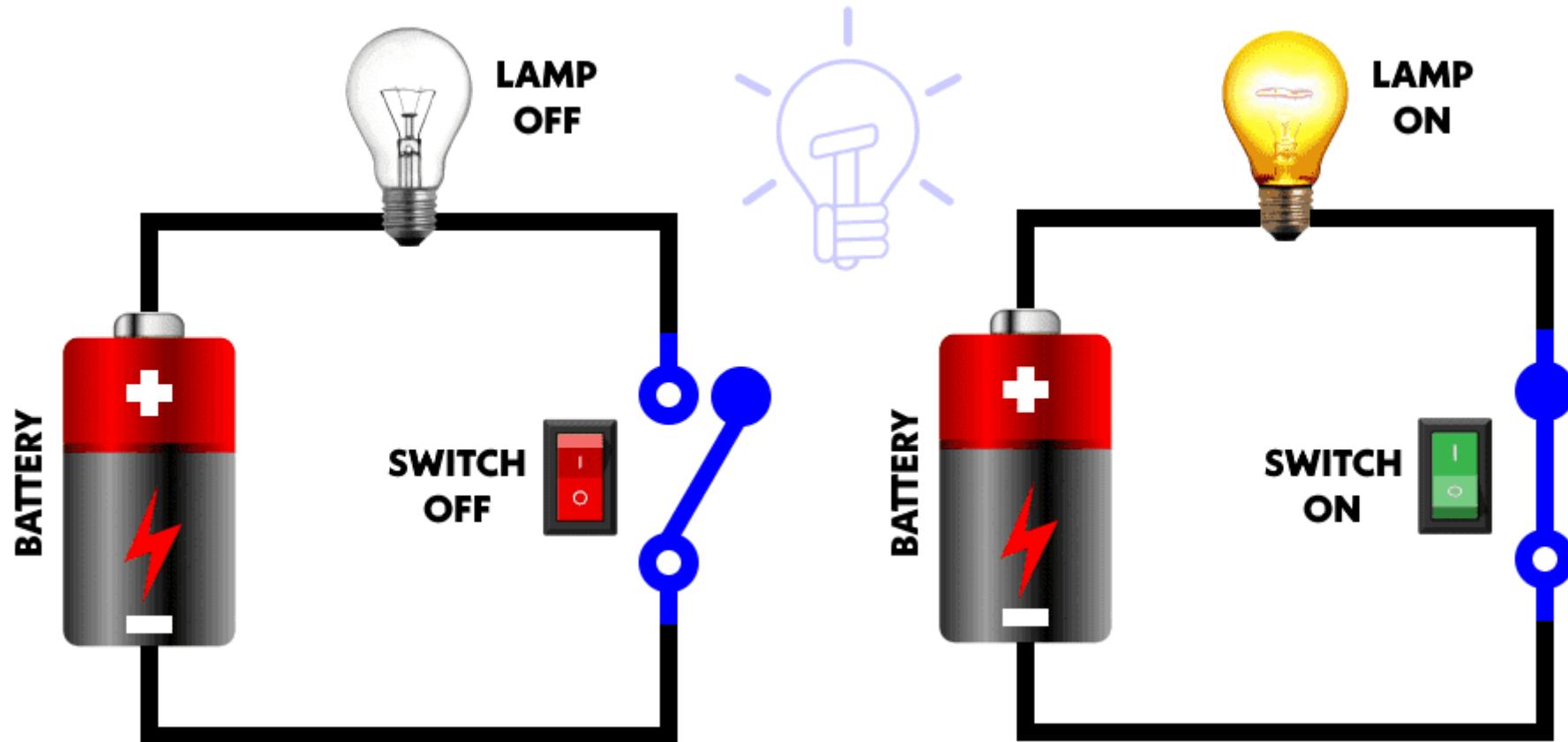
T<sub>h</sub>



# What is a Switch?

A switch is a device which is used to **make or break an electric circuit** automatically or manually.

Switches are a part of the control system and without it, control operation cannot be achieved. A switch can perform two functions, namely fully ON (by closing its contacts) or fully OFF (by opening its contacts).



# Working of an Electrical Switch

The controlling operation of a switch can be defined by its “Pole” and “Throw”. A **Pole** represents the number of operations controlled by a single switch. The **Throw** indicates the number of contacts in a switch.

For example, the NO (Normally Open) and NC (Normally Closed) are Single Throw which is used to control the circuit by making/breaking contacts of the switch.

The intermediate and changeover switches are Double Throw which is used to control the two way operation of the circuit by closing/opening contacts of the switches which is used to divert the flow of current from one circuit to another.

A Single-Pole-Single Throw (SPST) switch is known for controlling a single circuit (e.g. ON/OFF) per operation. This way, the number of poles and throws are used to represent the controlling process of a switch.

# Types of Switches

➤ **Mechanical** (activated physically, by moving, pressing,

releasing, or touching its contacts.)

➤ **Electronic** (do not require any physical contact in order

to control a circuit)

➤ **Electro-Mechanical**

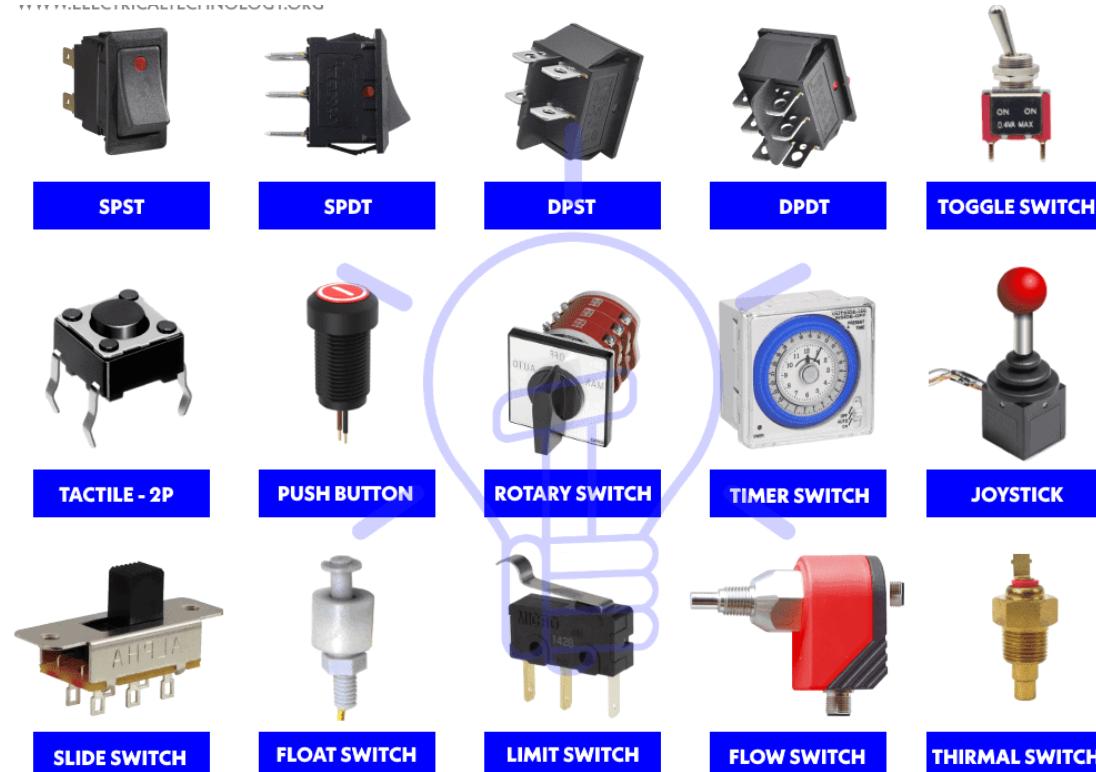
The number of **poles** on a switch defines how many

**separate circuits** the switch can control. So a switch with

one pole, can only influence one single circuit. A four-pole

switch can separately control four different circuits.

A switch's **throw-count** defines how many **positions** each  
of the switch's poles can be connected to. For example, if a  
switch has two throws, each circuit (pole) in the switch can  
be connected to one of two terminals.

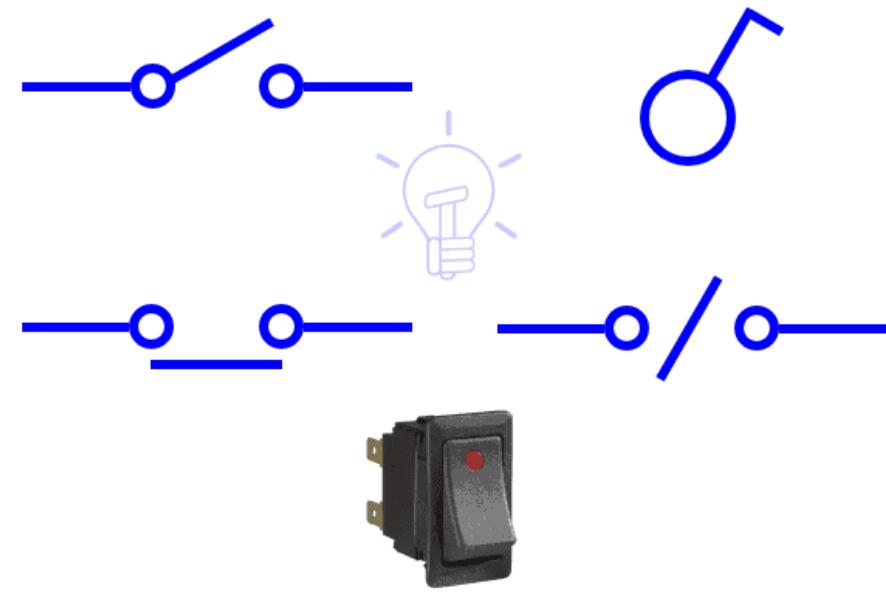


# Mechanical Switch

- ✓ **Momentary Switches** (like push buttons, for example) are used to make momentary contact (for a brief time or as long the button is pressed).
- ✓ **Latched Switches**, maintain the contact until it is forced to the other position.

## SPST (Single Pole Single Throw)

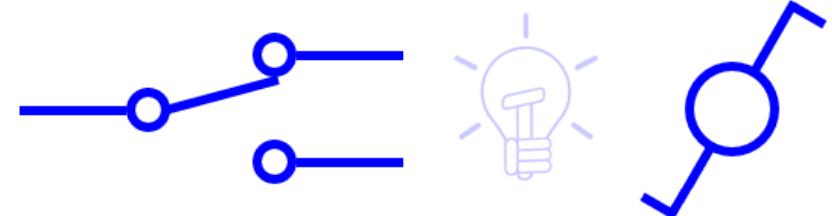
- A **simple ON/OFF switch** commonly found in our homes for lighting circuits and small load appliances as well as computers and devices.
- It is also called a **“One Way” or “Single Way” Switch**
- It controls single operation in a circuit
- The contact of the SPST switch can be either **NO (Normally Open)** for OFF position or **NC (Normally Closed)** for ON position.



# Mechanical Switch

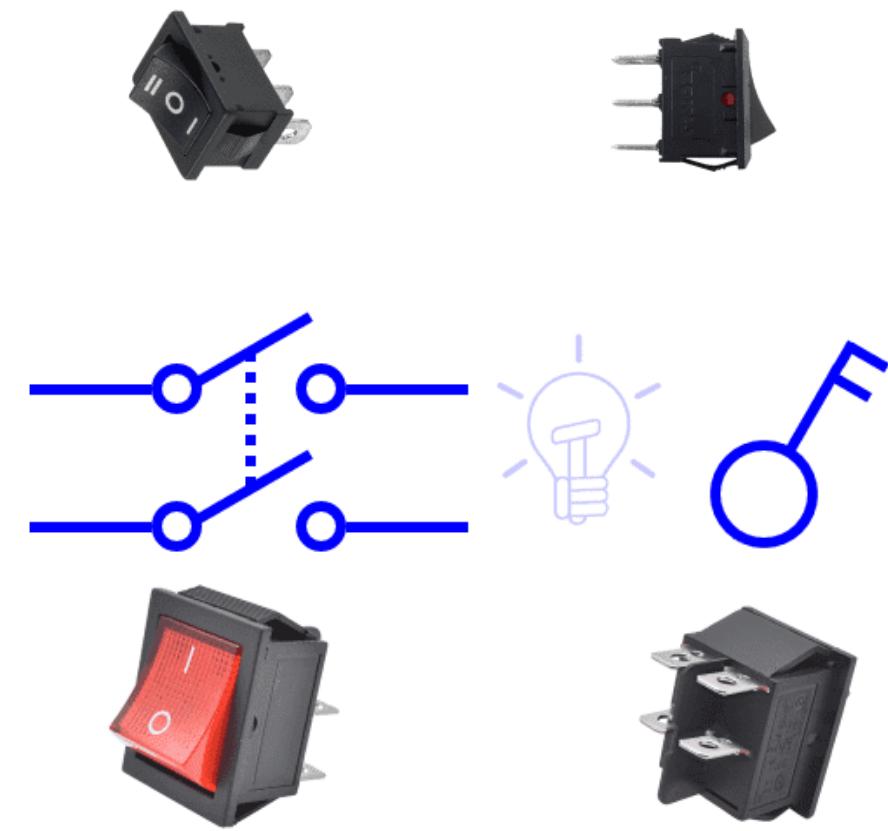
## SPDT (Single Pole Double Throw)

- SPDT switch has mainly three pins (terminals) and an extra as ground terminal. The input terminal is known and used as “common” which is also called a **Two-Way Switch**
- **Two circuits can be controlled** at the same time while using this switch.
- The remaining two terminals are called **travelers (output terminals)**.



## DPST (Double Pole, Single Throw)

- DPST switch is basically **two SPST switches in one package** and can be operated by a **single lever**.
- This switch is mostly used where both ground and line need to be broken (or closed) at the same time, same as the operation of a 2-Pole breaker.
- It has **two poles** i.e. it can control two circuits (Hot and Neutral) and a **Single Throw** i.e. it can only make one operation e.g. ON or OFF.
- Double Pole, Single Throw switch has **four terminal pins** i.e. **2 as Input and the rest of 2 as output**. DPST switches are used to **control a single circuit** while both the contacts are needed to be actuated.



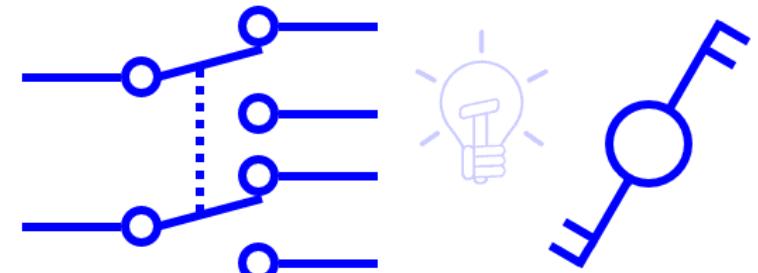
# Mechanical Switch

## DPDT (Double Pole Double Throw)

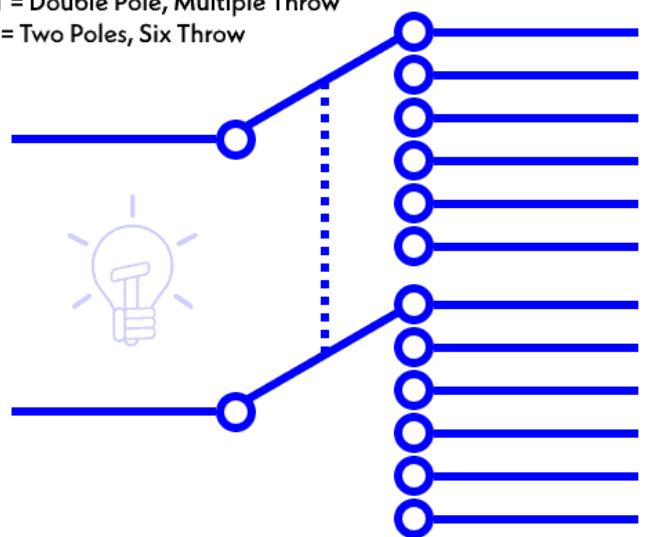
- DPDT switch is equivalent to **two SPDT switches packaged in one unit.**
- This switch has **two common pins** and **four signal pins** (total of 6 terminals).
- Total four different combinations of signals can be applied to the input pins of this switch.
- DPDT switches are used to **control two different electric circuits at the same time**. While it has a common lever for both operations, It can be used for two different operation i.e. ON & OFF positions.

## DPMT (Double Pole Multi Throw)

These kinds of switches consist of 2-poles and multiple throw i.e. they can be used to control two independent circuits. These types of switches with a common lever are used as changeover and selectors switches for multiway switching.

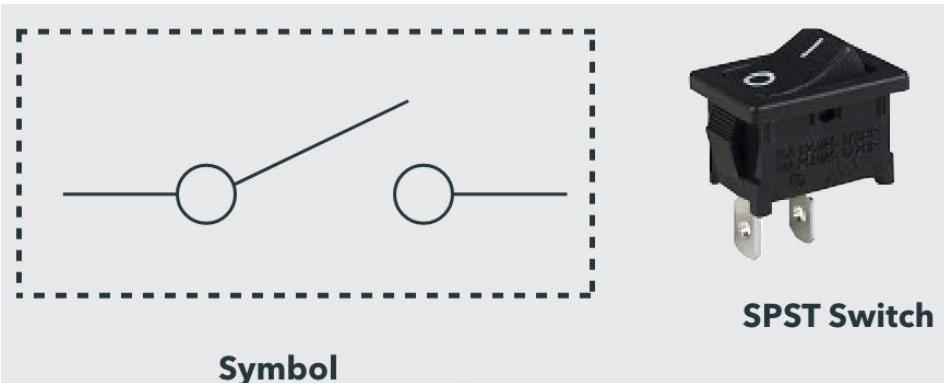


DPMT = Double Pole, Multiple Throw  
2P6T = Two Poles, Six Throw

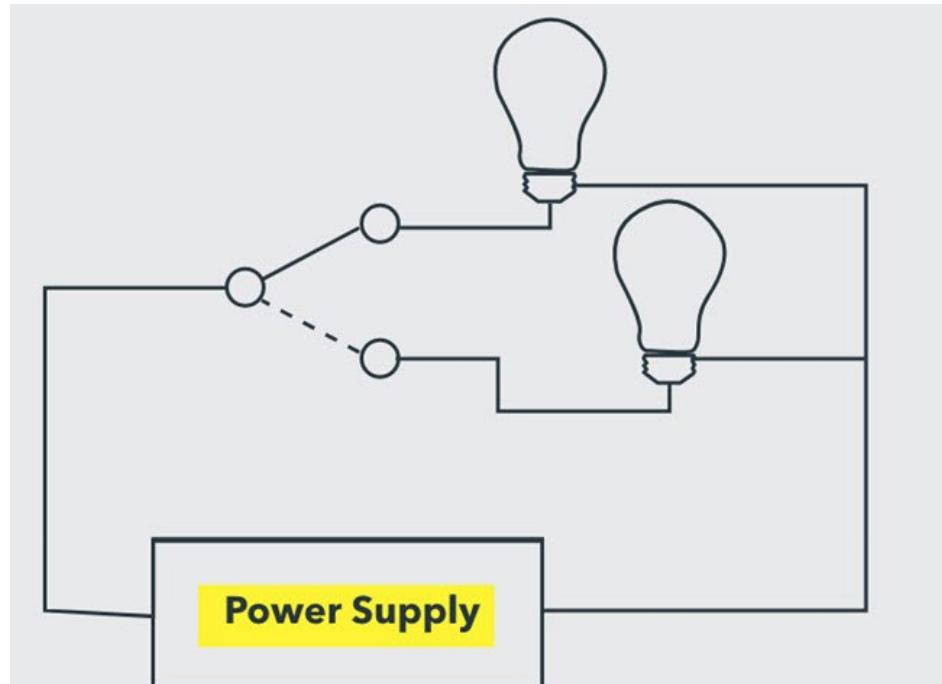
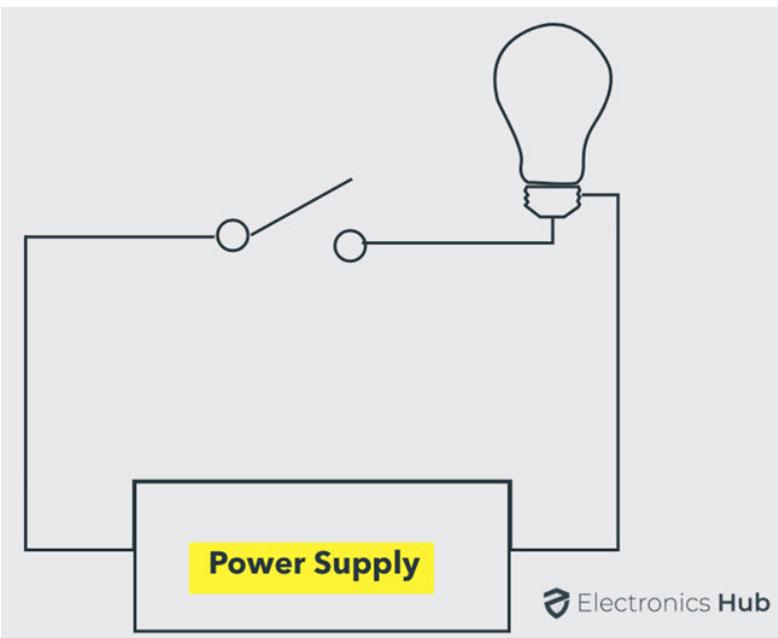
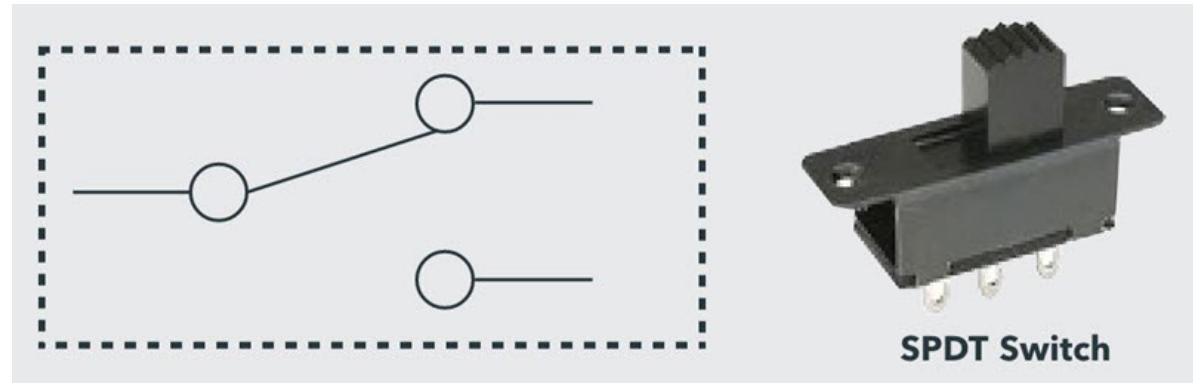


# Mechanical Switch

Single Pole Single Throw Switch (SPST)

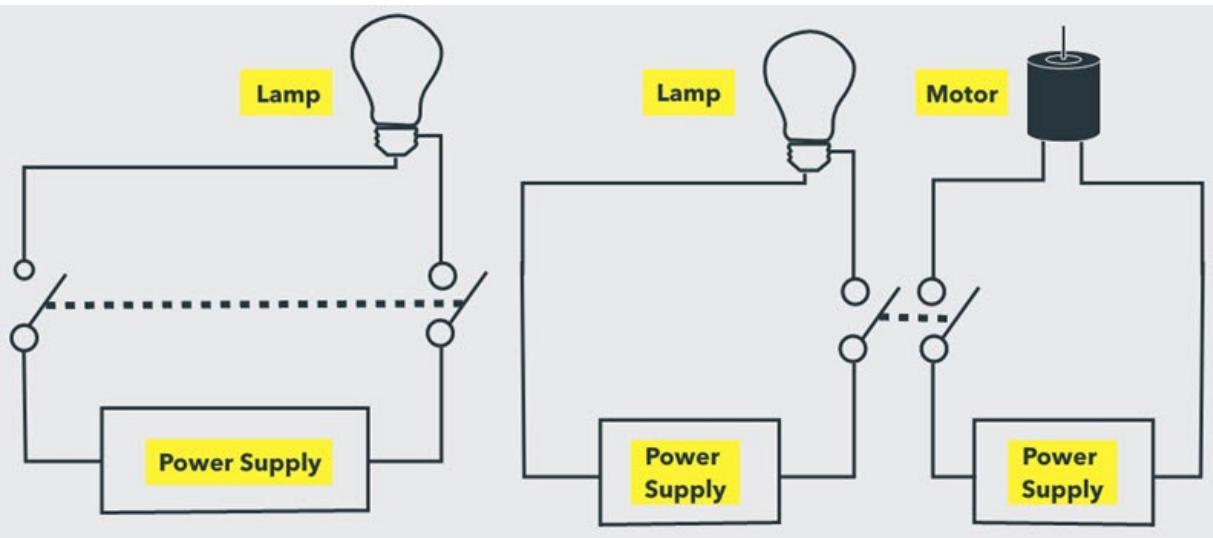
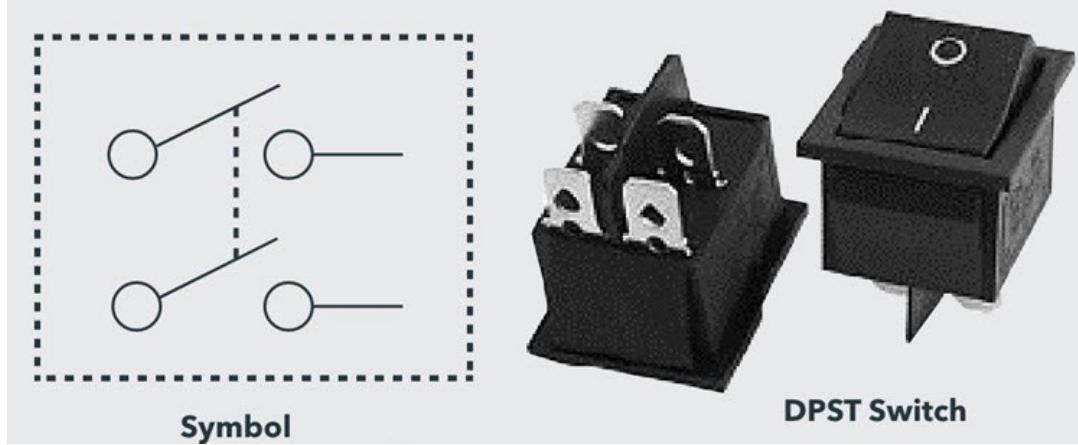


Single Pole Double Throw Switch (SPDT)



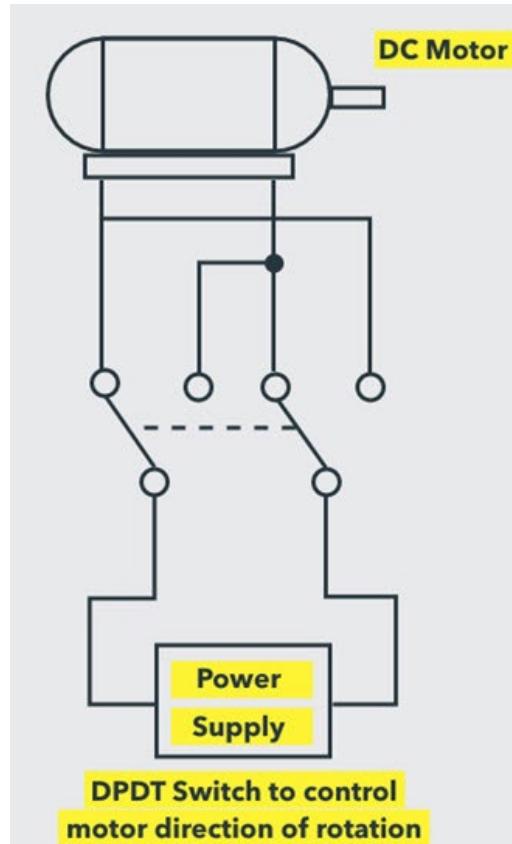
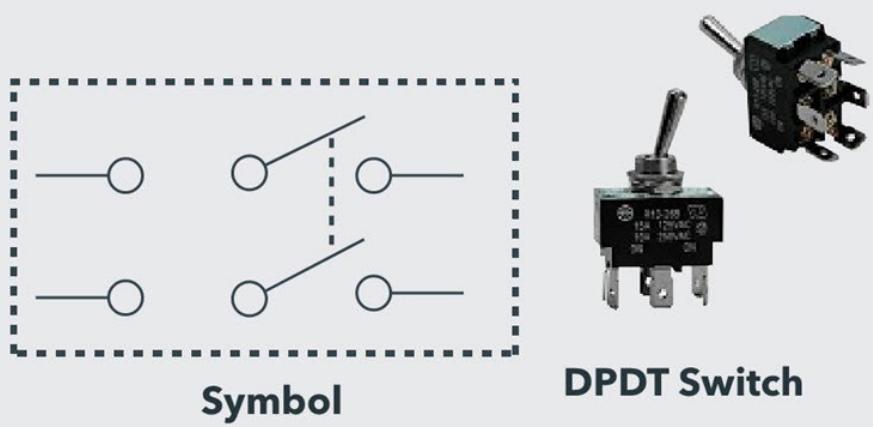
# Mechanical Switch

## Double Pole Single Throw Switch (DPST)



- This switch consists of four terminals: two input contacts and two output contacts.
- It behaves like a two separate SPST configurations, operating at the same time.
- It has only one ON position, but it can actuate the two contacts simultaneously, such that each input contact will be connected to its corresponding output contact.
- In OFF position both switches are at open state.
- This type of switches is used for controlling two different circuits at a time.
- Also, the contacts of this switch may be either normally open or normally closed configurations.

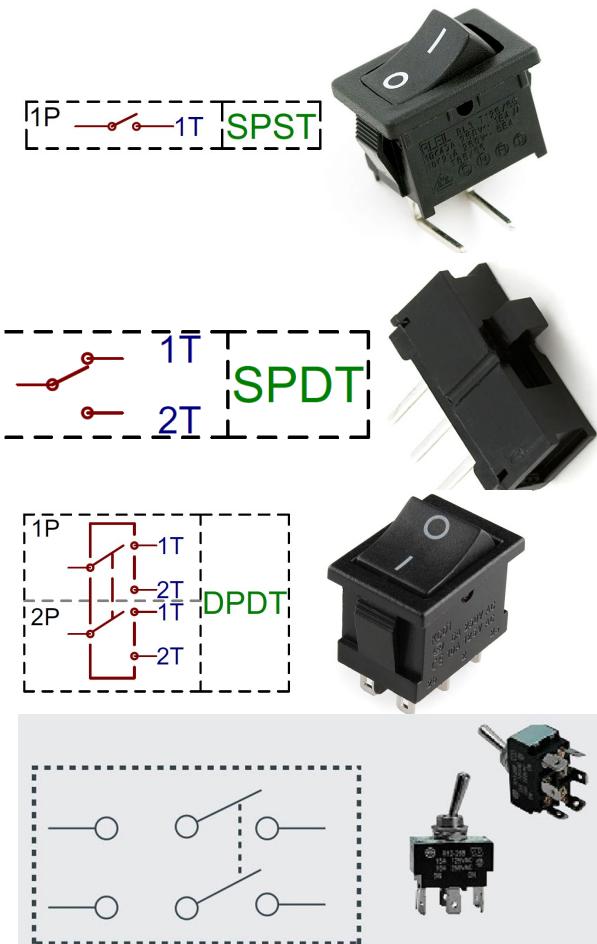
# Double Pole Double Throw Switch (DPDT)



- This is a dual ON/OFF switch consisting of two ON positions.
- It has six terminals, two are input contacts and remaining four are the output contacts.
- It behaves like a two separate SPDT configuration, operating at the same time.
- Two input contacts are connected to the one set of output contacts in one position and in another position, input contacts are connected to the other set of output contacts.

# Mechanical Switches

Switch Type	Desig.	Symbol	Mechanism
Single-Pole, Single-Throw	SPST	 	OFF-ON OFF-(ON)
			OFF-(ON)
Single-Pole, Double-Throw	SPDT		ON-NONE-ON ON-OFF-ON (ON)-OFF-(ON) ON-OFF-(ON)
Double-Pole, Single-Throw	DPST		OFF-ON OFF-(ON)
Double-Pole, Double-Throw	DPDT		ON-NONE-ON ON-OFF-ON (ON)-OFF-(ON) ON-OFF-(ON)



**SPST**, also known as **1P1T**  
Single pole, single throw

**DPST** also known as **2P1T**  
Double pole, single throw

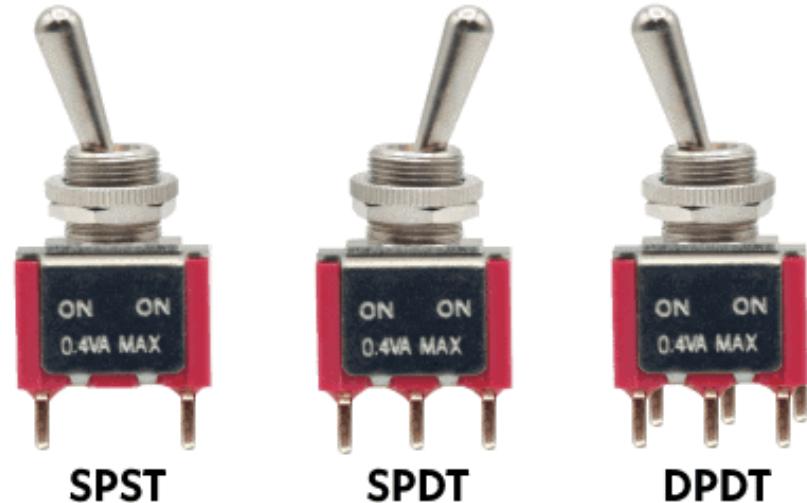
**SPDT** also known as **1P2T**  
Single pole, double throw

**3PST** also known as **3P1T**  
Three pole, single throw

# Mechanical Switch

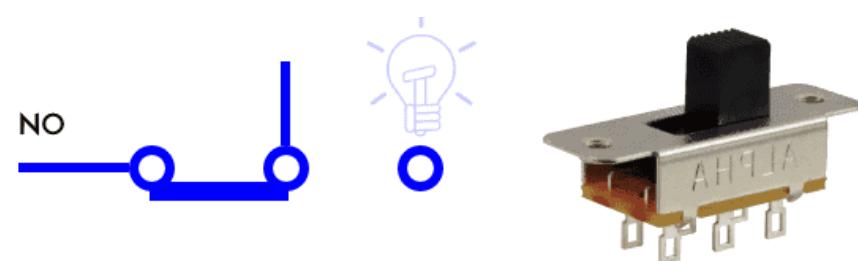
## Toggle Switch

- Toggle switches are **latched type** of switches which are actuated by **a lever angled in one or more directions**.
- This switch is **stable** in state and remains in that state unless or until the lever is pushed in another direction.
- Most of all household applications (such as lighting control switches) have toggle switch and it can fall into any category as mentioned above e.g. SPST, DPDT, DPST, DPDT etc.
- They are available for high current applications up to 30+ amperes and can be used for small current switching operations.



## Slide Switch

- A slide switch uses a slider as actuator which slides back and forth to make and break the contacts.

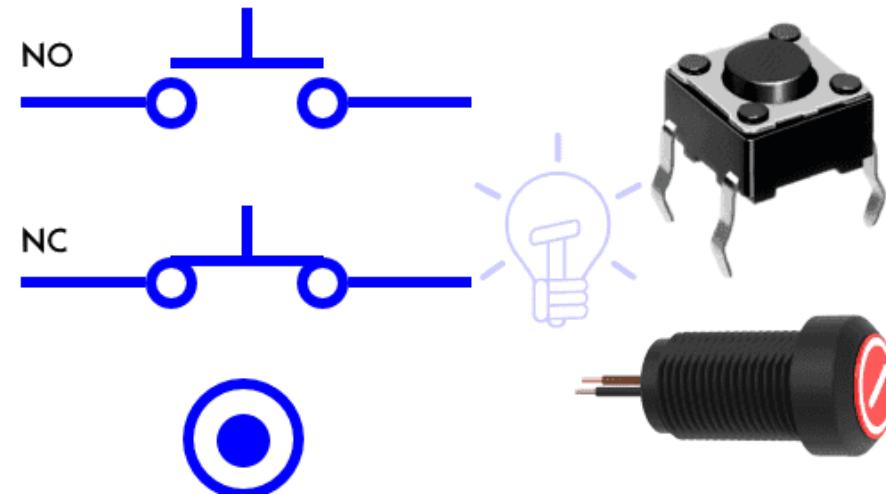


SPDT (single pole double throw) slide switch

# Mechanical Switch

## Push Buttons Switch

- push buttons are momentary switches which are operated by pressings (put a pressure by pushing) it for a while. The spring mechanism for the actuator inside it is used to close or open the circuit for OFF and ON operation.
- When a pushbutton switch is pressed, the movable contacts attached to the button make sure to connect the static (stationary or stable) contacts in series to make the circuit. When the pressure is released, contacts of the pins are detached and the circuit operation returns back to the first position either ON or OFF.
- They are generally NO, NC or double acting which is used to control two different circuits. Examples of push-button switches are drills, blowers and doorbells etc.

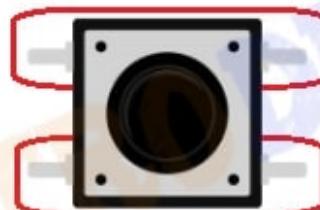


# Push Button Switch

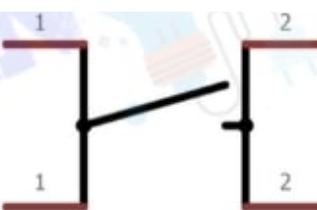
Push-Buttons ( Momentary Switch) are normally-open tactile switches. When we pressed the button it makes the circuit connected and when released the button it makes the circuit connection breaks. We can see from outside this switch is consists of four terminals. but both side of the terminal is internally connected.



internally connected pin 1



internally connected pin 2



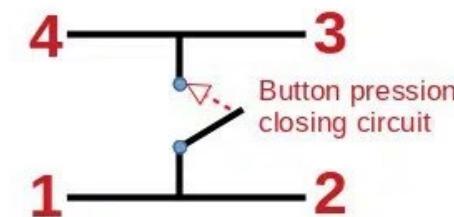
Symbol

Push Button Switch

Internal Connection



2



3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

1

2

3

4

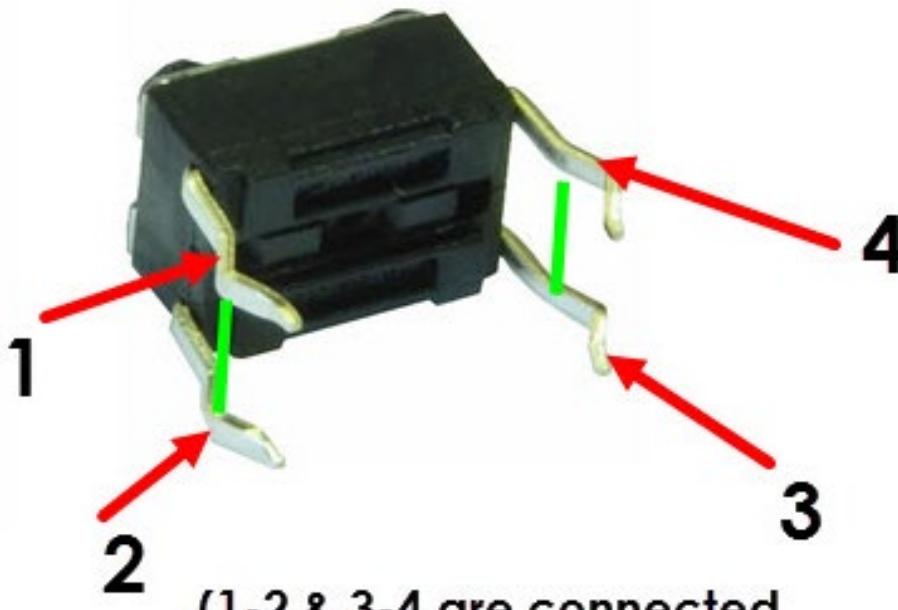
1

2

3

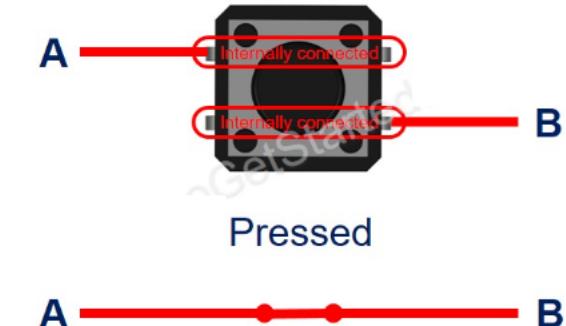
4

# Push Button Switch



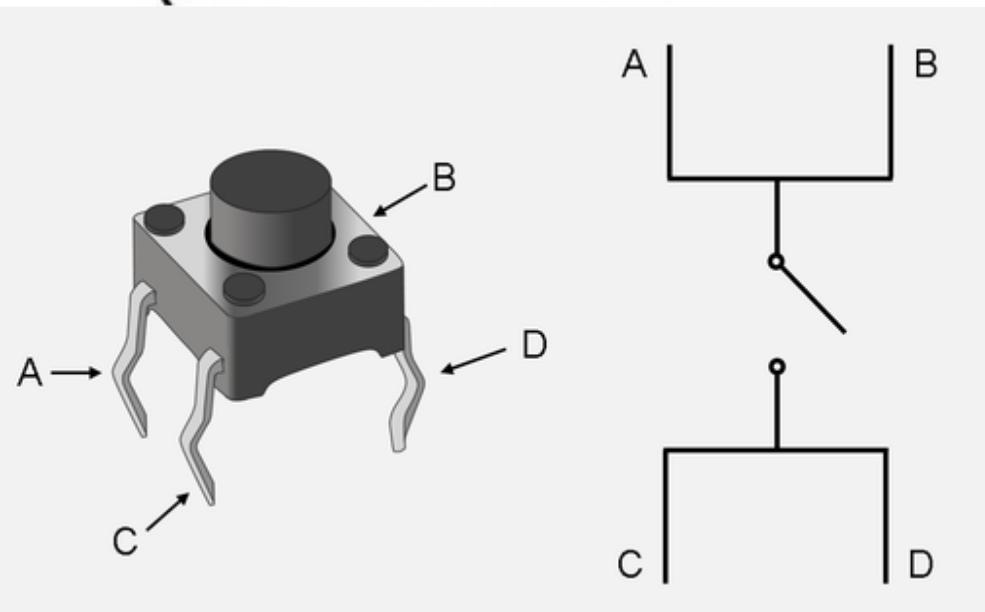
## How It Works

- When the button is NOT pressed, pin A is NOT connected to pin B
- When the button is pressed, pin A is connected to pin B



However, these pins are internally connected in pairs. Therefore, we only need to use two of the four pins, which are NOT internally connected.

There are four ways (actually two ways because of symmetry) to connect to button (see image)



We can use only two pins of a button, why does it have four pins?

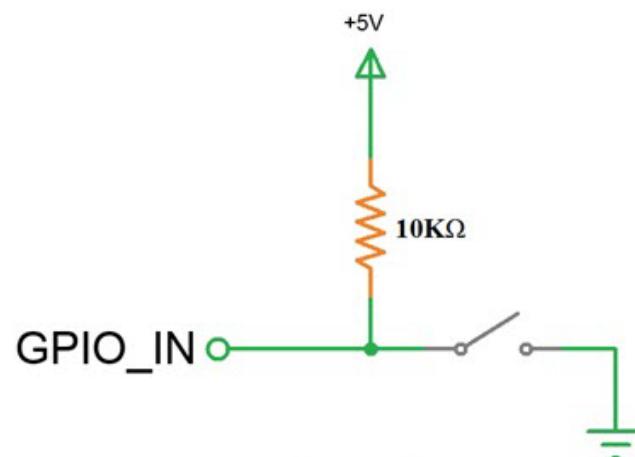
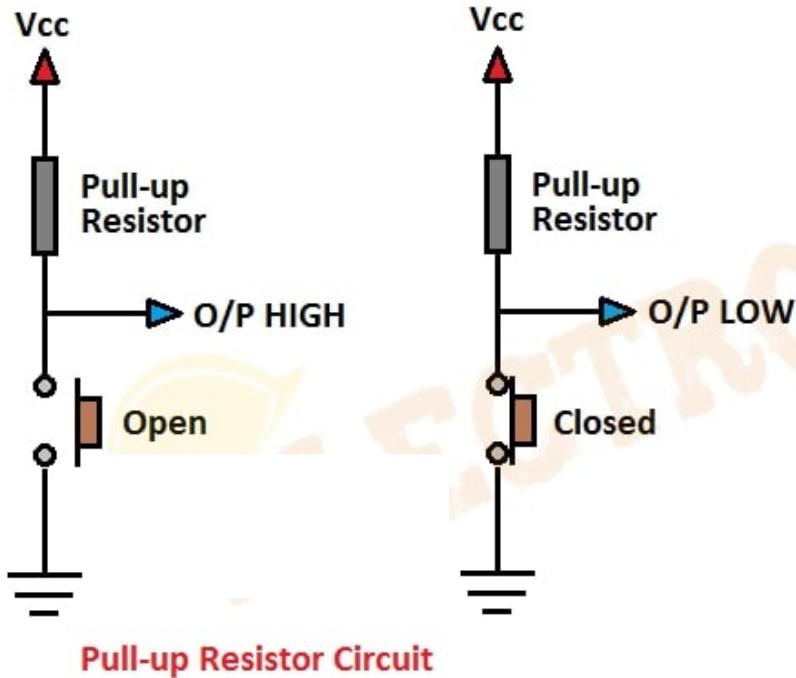
⇒ To make it stand firmly in PCB (board) to resist the pressing force.

# GPIO of Raspberry Pi as Input

- If the Raspberry Pi wants to read the value from an external device, the corresponding GPIO pin must be declared as an Input Pin.
- But when a GPIO Pin of the Raspberry Pi is declared as Input, it must be ‘tied’ to High or Low or else it is called as a Floating Input Pin. A Floating Input is a pin which is defined as input and left as it is.
- Any Digital Input Pin is very sensitive and catches even the slightest changes and will pick up the stray capacitances from your finger, breadboard, air etc.
- In order to avoid this, a Digital Input Pin must be tied to VCC or GND with the help of Pull-up or Pull-down resistors.

# Pull-Up and Pull-Down Resistor

## Pull-up Resistors



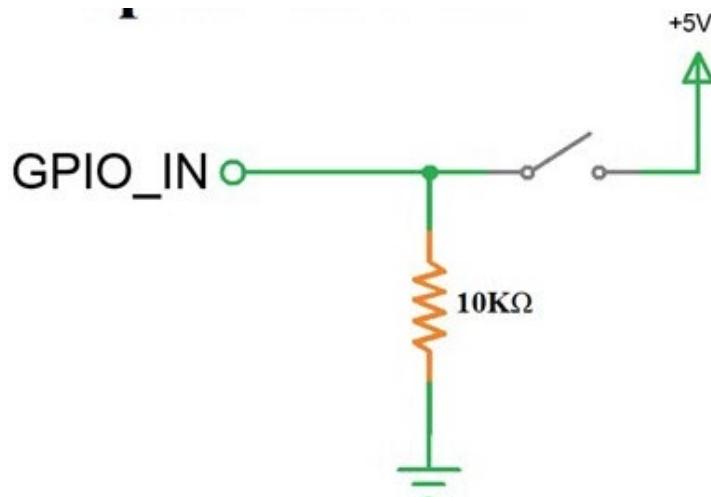
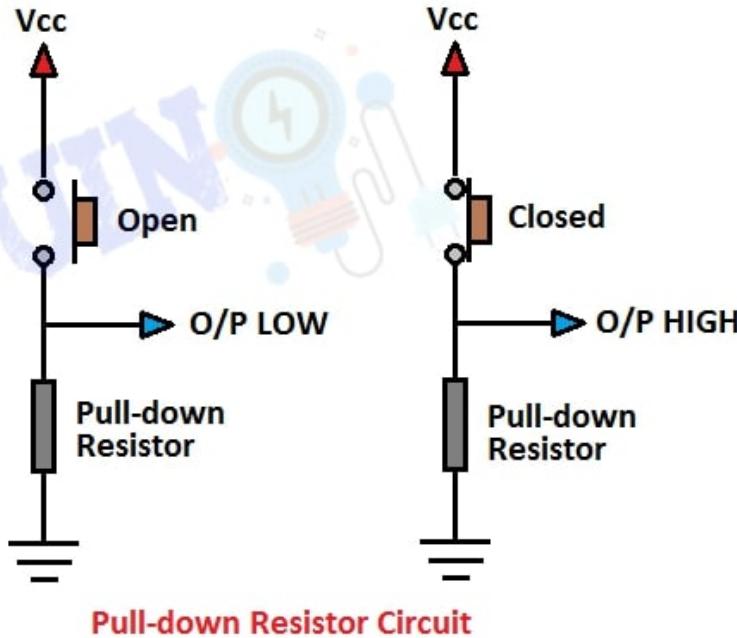
If we connect the push button switch directly to Raspberry Pi to get digital input, It means switch one pin is connected to Ground or 5v Vcc and another pin connected to Raspberry Pi digital pin. In this case, the Raspberry Pi is read **unstable input** from the push button.

So, we need to connect a “pull-up” or “pull-down” resistors circuit to **stabilizes the input**, when using the switch.

If the push button one pin is connected to the Vcc through a resistor and the other pin is connected to the ground, this circuit known as the **pull-up resistor circuit**. In this case, the push button output is **High (1)** when the button is **open**, and the output of the push button is **Low (0)** when the **button is pressed**.

# Pull-Up and Pull-Down Resistor

## Pull - down Resistor



If the push button one pin is connected to the **ground through a resistor** and the other pin is connected to the Vcc, this circuit known as the **pull-down resistor circuit**. In this case, the push button output is **Low(0)** when the **button is open**, and the output of the push button is **High(1)** when the **button is pressed**.

The Raspberry pi pico **already has internal pull-up and pull-down resistors** built-in which are automatically triggered when a digital input is read by the Raspberry pi pico meaning that we don't need to add any extra resistors to our circuit.

# When to use Pull-Up and Pull-Down Resistor?

*When should and should NOT we use a pull-down/pull-up resistor for an input pin?*

- ◆ If the sensor has either closed (connected to **VCC** or **GND**) or open (NOT connected to anything) states, you need a pull-up or pull-down resistor to make these states become two states: **LOW** and **HIGH**. For example, push-button, switch, magnetic contact switch (door sensor)...
- ◆ If the sensor has two defined voltage levels (**LOW** and **HIGH**), you do NOT need a pull-up or pull-down resistor. For example, [motion sensor](#), [touch sensor](#) ...

# Trouble-shooting Pull-Up and Pull-Down Resistor

There are two common troubles that beginners usually get into:

## 1. Floating Input Problem

- ✓ **Symptom:** The reading value from the input pin is not matched with the button's pressing state
- ✓ **Cause :** Input pin is NOT used pull-up and pull-down resistors.
- ✓ **Solution:** Use Pull-up and Pull-down resistor.

## 2. Chattering Phenomenon

It should be considered in only some application that needs to detect exactly number of the pressing.

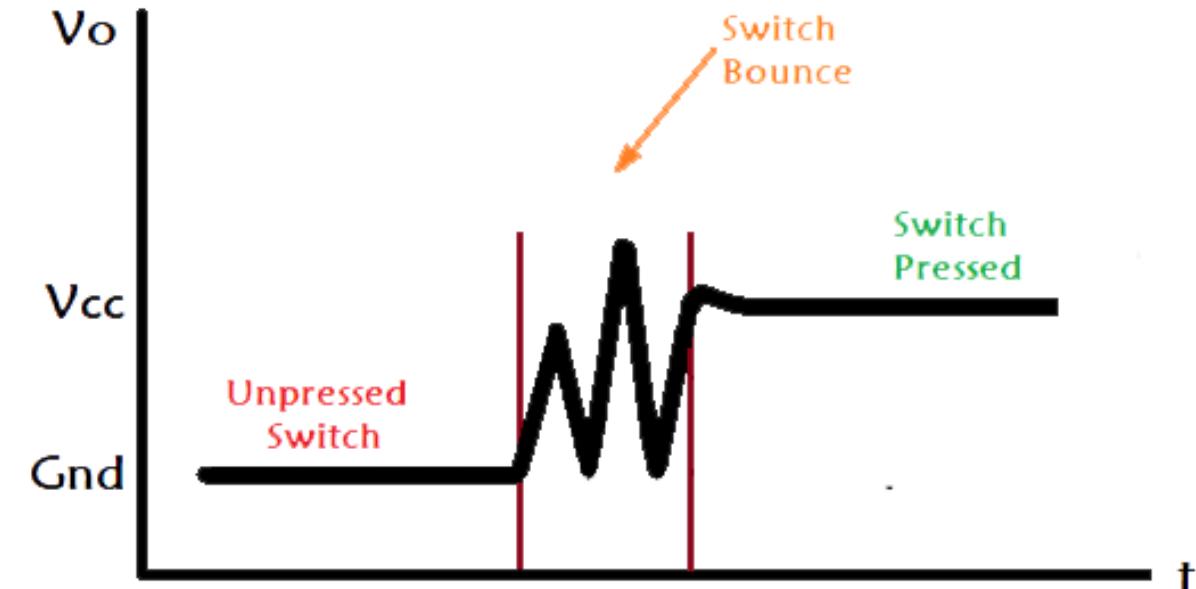
- ✓ **Symptom:** Button is pressed once, but Raspberry Pi code detects several times.
- ✓ **Cause :** Due to mechanical and physical issues, the state of button (or switch) is quickly toggled between **LOW** and **HIGH** several times.
- ✓ **Solution:** Debounce

# Trouble-shooting Pull-Up and Pull-Down Resistor

## What is Switch Bouncing?

When we press a pushbutton or toggle switch or a micro switch, two metal parts come into contact to short the supply. But they don't connect instantly but the metal parts connect and disconnect several times before the actual stable connection is made. The same thing happens while releasing the button. This results in the false triggering or multiple triggering like the button is pressed multiple times. It's like falling a bouncing ball from a height and it keeps bouncing on the surface, until it comes at rest.

Switch bouncing is the **non-ideal behavior** of any switch which generates multiple transitions of a single input. Switch bouncing is not a major problem when we deal with the power circuits, but it causes problems while we are dealing with the **logic or digital circuits**. Hence, to remove the bouncing from the circuit Switch Debouncing Circuit is used.

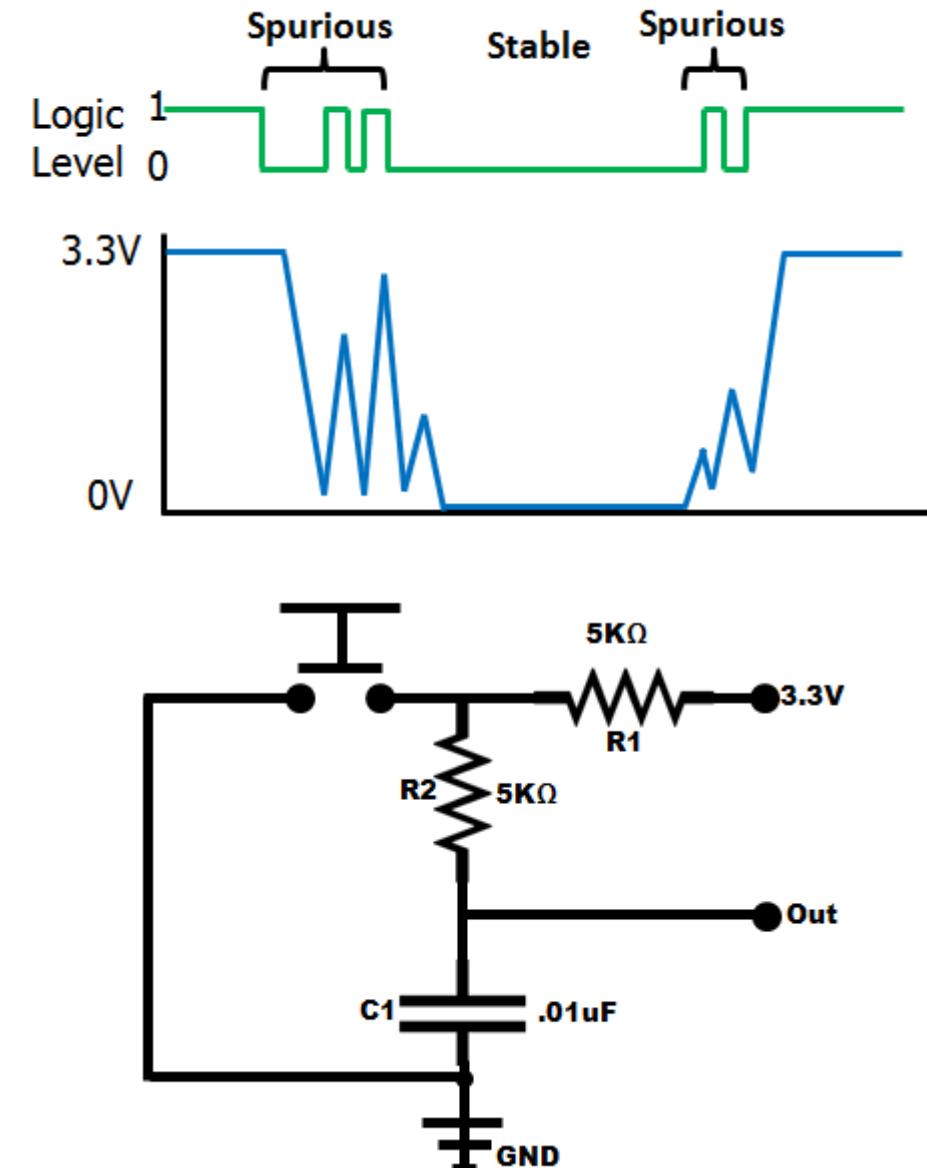


# Trouble-shooting Pull-Up and Pull-Down Resistor

## Switch De-Bouncing

There are three commonly used methods to prevent the circuit from switch bouncing.

- **Hardware Debouncing** (In the hardware debouncing technique we use an S-R flip flop to prevent the circuit from switch bounces. This is the best debouncing method among all.)
- **RC Debouncing** (The R-C is defined by its name only, the circuit used a RC network for the protection from switch bounce. The capacitor in the circuit filter the instant changes in the switching signal. When the switch is in open state the voltage across the capacitor remain zero. Initially, when the switch is open the capacitor charge through the R1 and R2 resistor.)
- **Switch Debouncing IC** (Some of the debouncing ICs are MAX6816, MC14490, and LS118.)



# Electrical & Electronic Switches

- Electrical and mostly electronic switches are **solid-state devices** based on semiconductor materials with **fast response, accurate operation and small in size** as compared to mechanical and electromechanical switches. The solid state switches are based on the basic components such as diodes, SCR, MOSFET, GTO, IGBT, transistors, and relays etc.
- Electronic switches have **no physical contacts or moving parts** and can be automatically operated by electric signals or programmed circuits like microcontroller or microprocessor. They are precise in operation for stability and reliability of the system without noise of switching operation.
- They are used in many modern applications such as variable frequency drive (VFD) drives for motors, HVAC & in industrial, automation, automotive, aerospace, robotic and many more commercial applications.

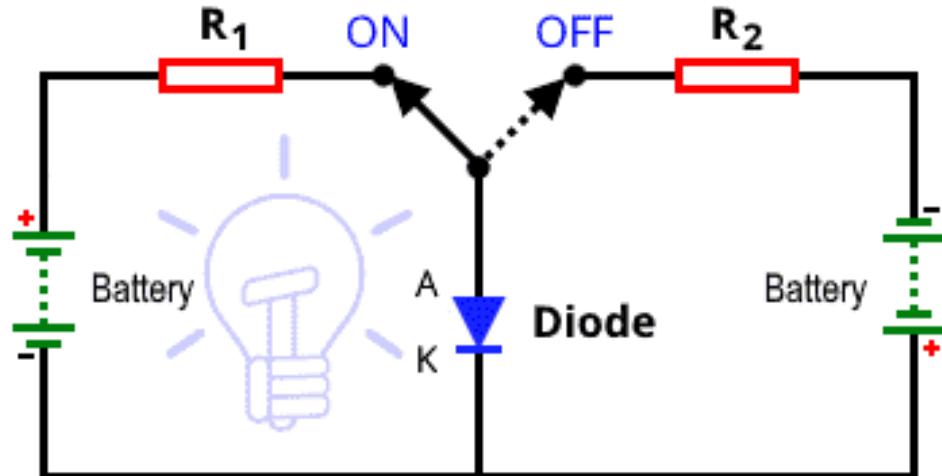
# Electrical & Electronic Switches

## Diode as a Switch

- A basic PN junction diode can be used as a switch. The diode in forward bias acts as a closed switch while it acts as an open switch in case of reverse bias.

## switching operations of a diode.

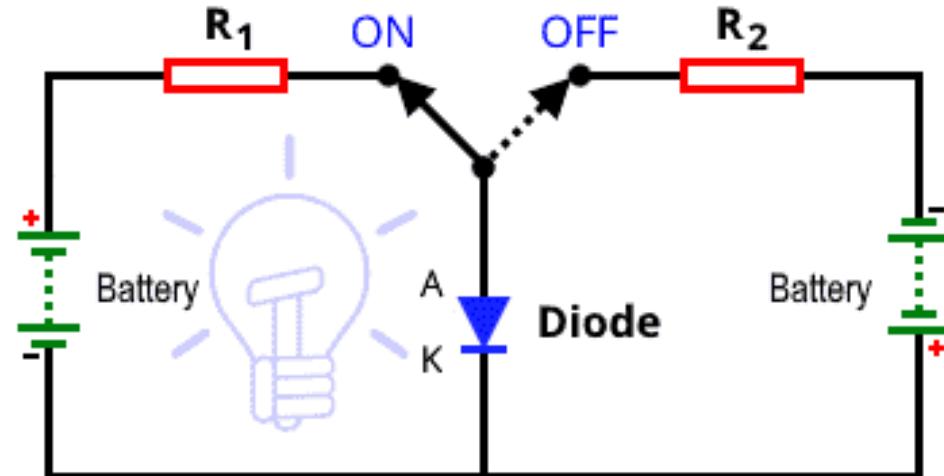
- Forward Bias: If positive signal applied to the Anode terminal and cathode is negative, the diode is forward biased, hence it acts as a closed switch.
- Reverse Bias: If the negative signal applied to the Anode and cathode is positive, the diode is in reverse biased, thus it acts as an open switch.



# Electrical & Electronic Switches

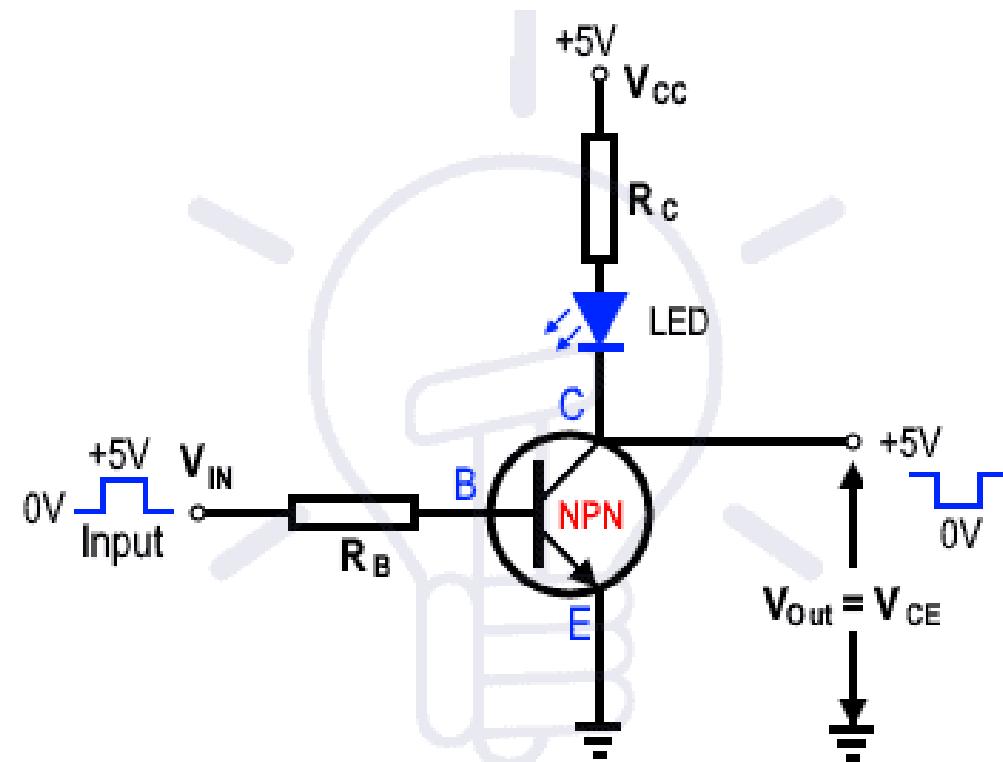
## BJT Transistors as a switch:

- Bipolar Junction Transistors (BJT) can be used as normal switches as they are able to block or pass the flowing of electric current in different modes of operations.



## NPN Transistor as a Switch

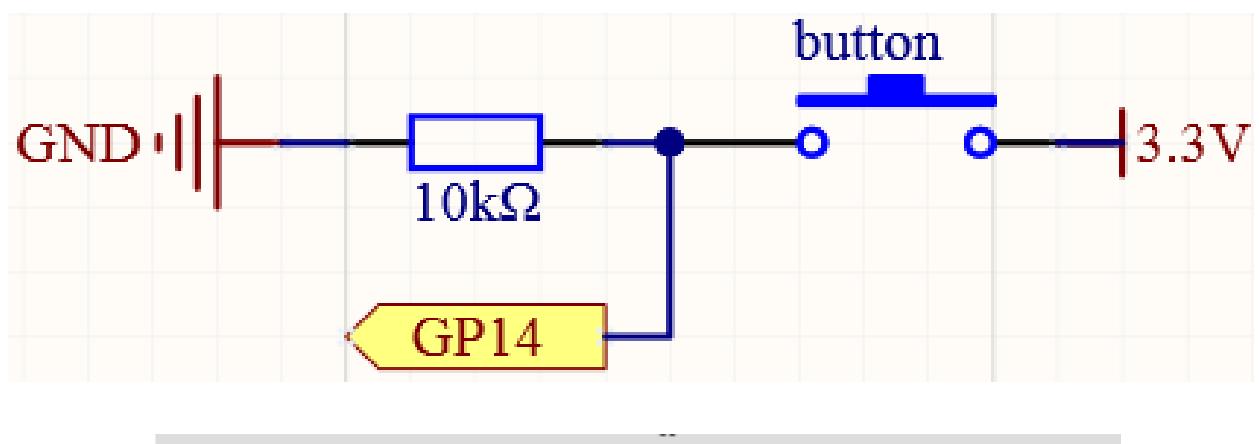
- When the input to the inverter is high "+5V/+3.3V", the NPN transistor is saturated and its output is low "≈0V". When the input to the inverter is low, the transistor is cut-off and its output is high.
- In the saturation region, it is "ON" like a closed switch
- In the cut-off region, it is "OFF" like an open switch.



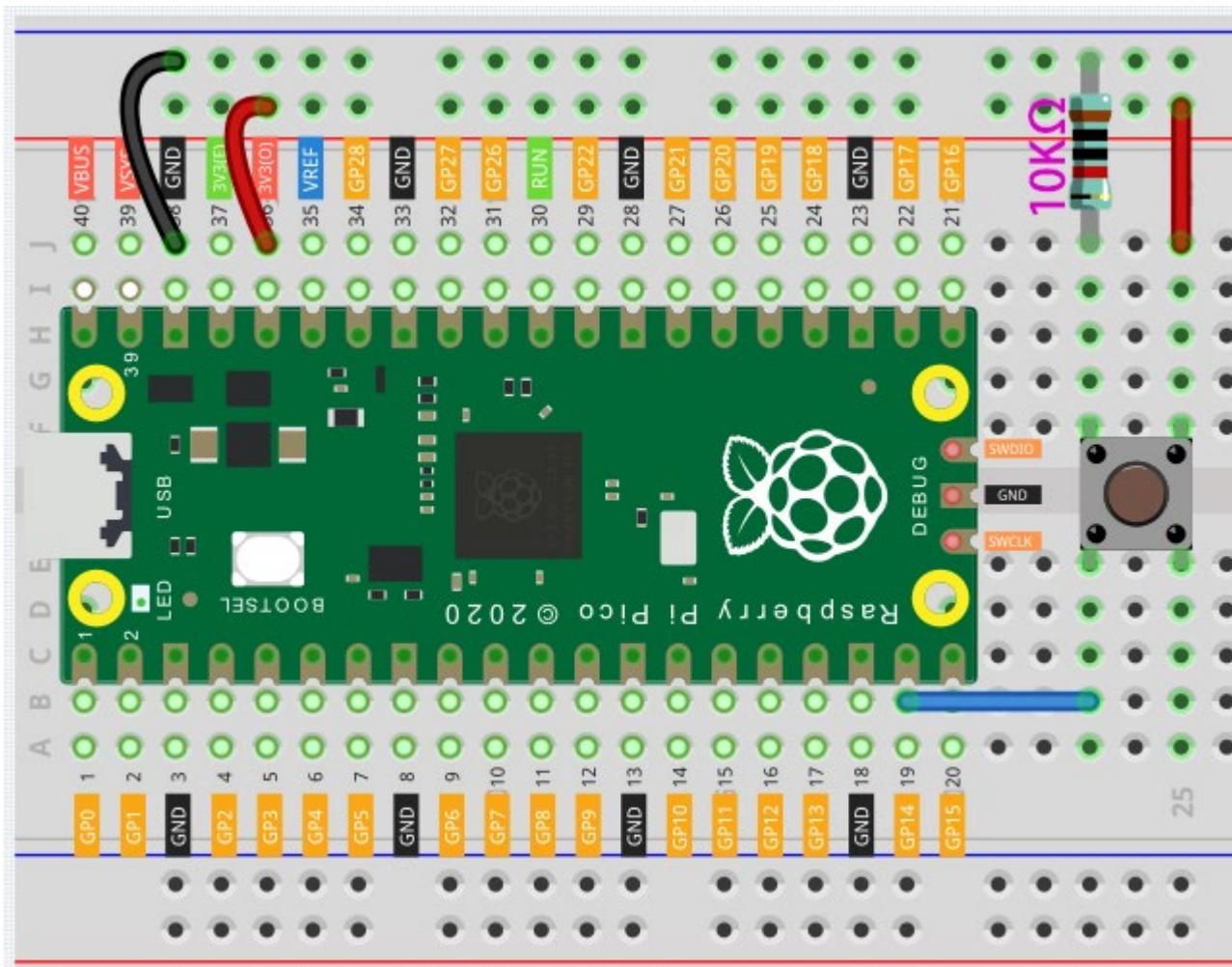
# Reading Button Value (Pull-Down Mode)

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.

## Pull-down Working Mode



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



# Reading Button Value (Pull-Down Mode)

## Pull-down Working Mode

WOKWi ● SAVE ▾ SHARE Docs B

main.py • diagram.json • PIO

```
1 from machine import Pin
2 from utime import sleep
3
4 button = Pin(14, Pin.IN)
5
6 while True:
7     if button.value() == 1:
8         print("You pressed the button!")
9         sleep(1)
10
```

Simulation

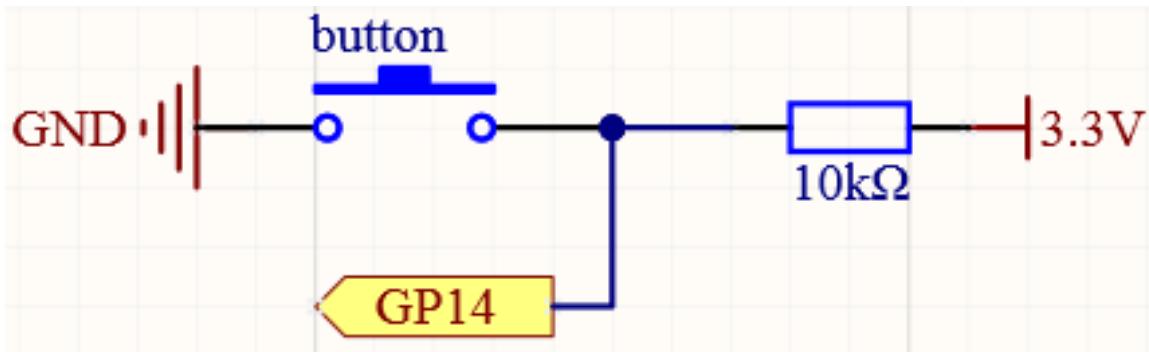
00:06.683 25%

You pressed the button!

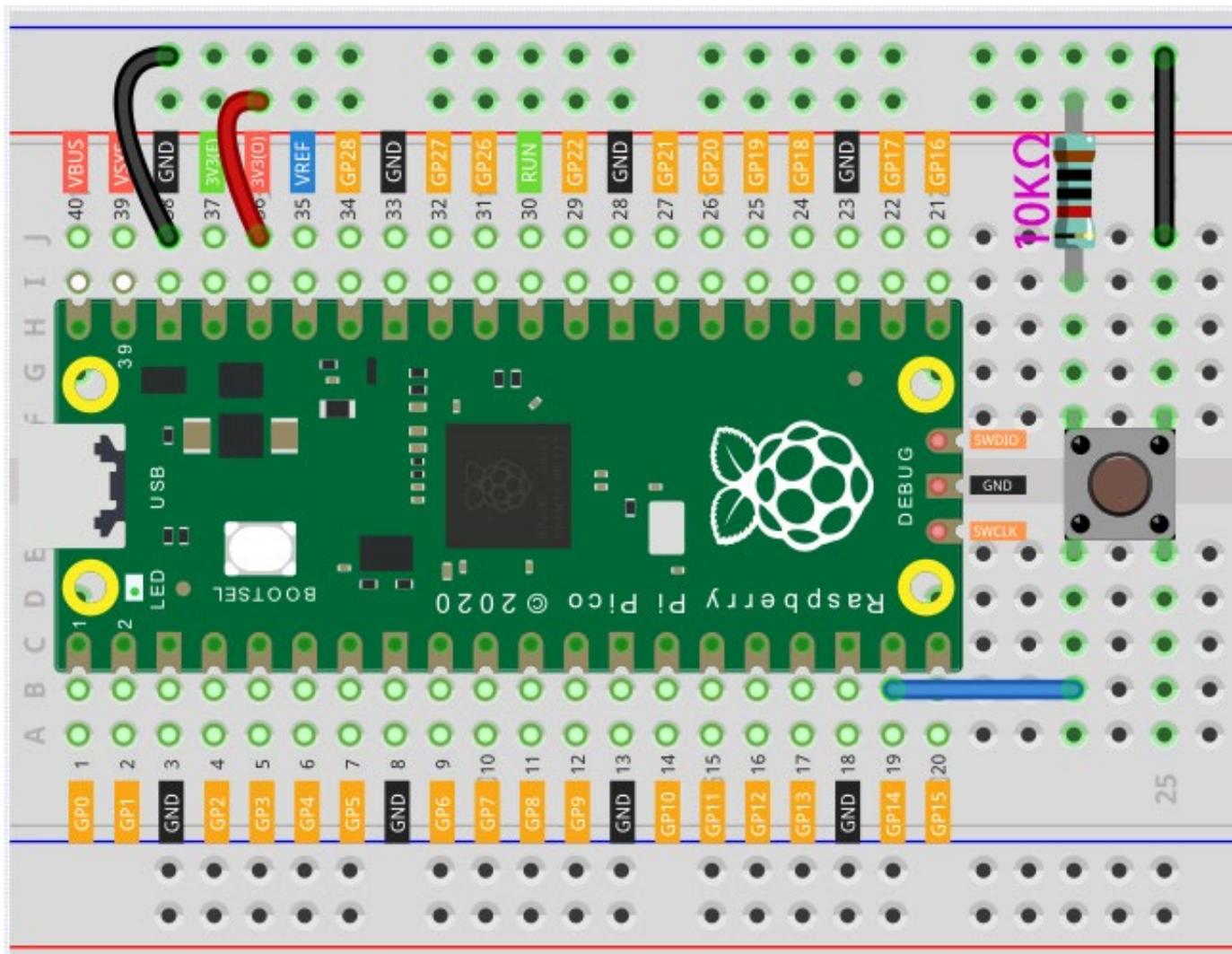
# Reading Button Value (Pull-Up Mode)

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’ (floating) signal which can trigger even when you’re not pushing the button.

## Pull-up Working Mode



```
from machine import Pin  
from utime import sleep  
  
button = Pin(14, Pin.IN)  
  
while True:  
    if button.value() == 0:  
        print("You pressed the button!")  
        sleep(1)
```



# Reading Button Value (Pull-Up Mode)

## Pull-Up Working Mode

WOKwi SAVE SHARE Docs B

main.py • diagram.json • PIO

```
1 from machine import Pin
2 from utime import sleep
3
4 button = Pin(14, Pin.IN)
5
6 while True:
7     if button.value() == 0:
8         print("You pressed the button!")
9         sleep(1)
```

Simulation

The simulation shows a green Raspberry Pi Pico board. A green push-button component is connected to one of its pins. A red wire connects the button's ground terminal to the GND rail of the breadboard. Another red wire connects the button's top terminal to a resistor (brown, black, orange, gold) and then to the Pico's digital pin 14. The Pico's VDD rail is connected to the resistor's other end and to the button's top terminal. The breadboard has a green ground rail and a blue VDD rail.

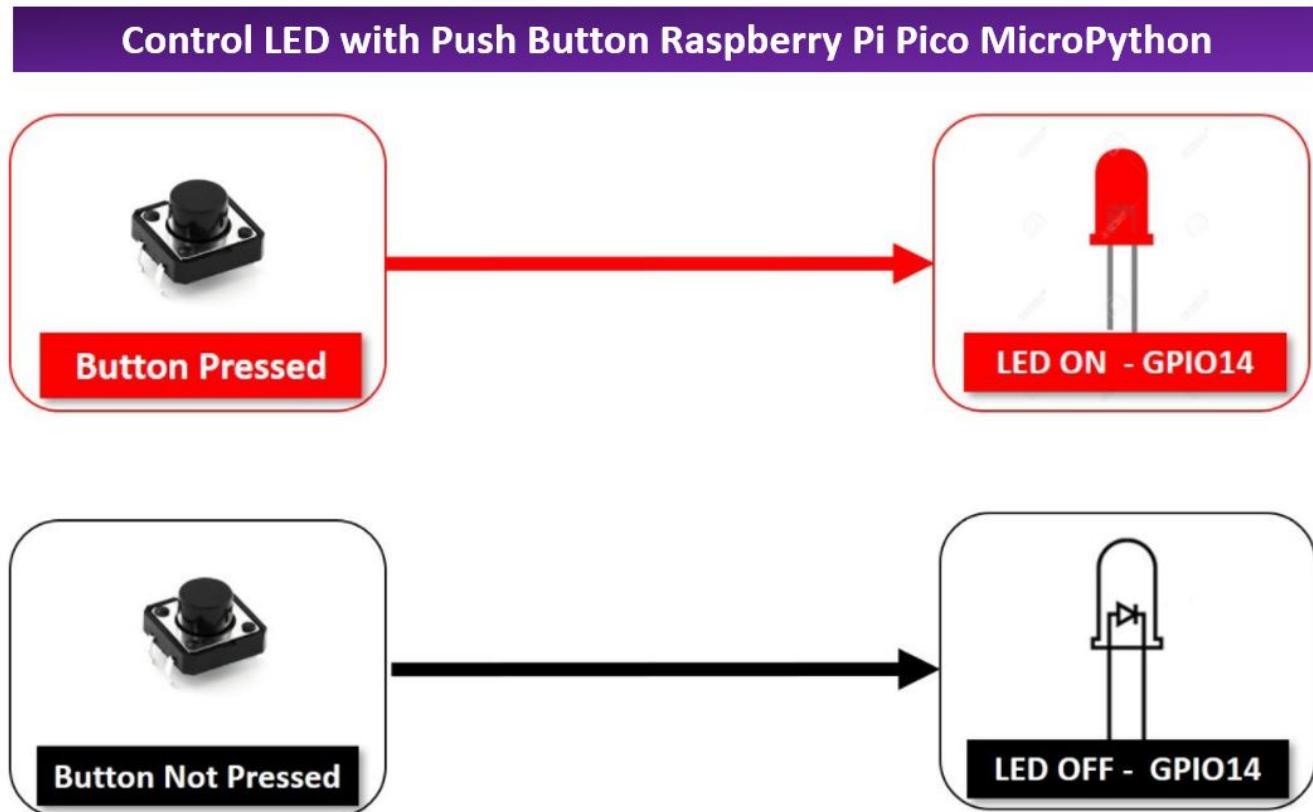
00:04.316 19%

You pressed the button!

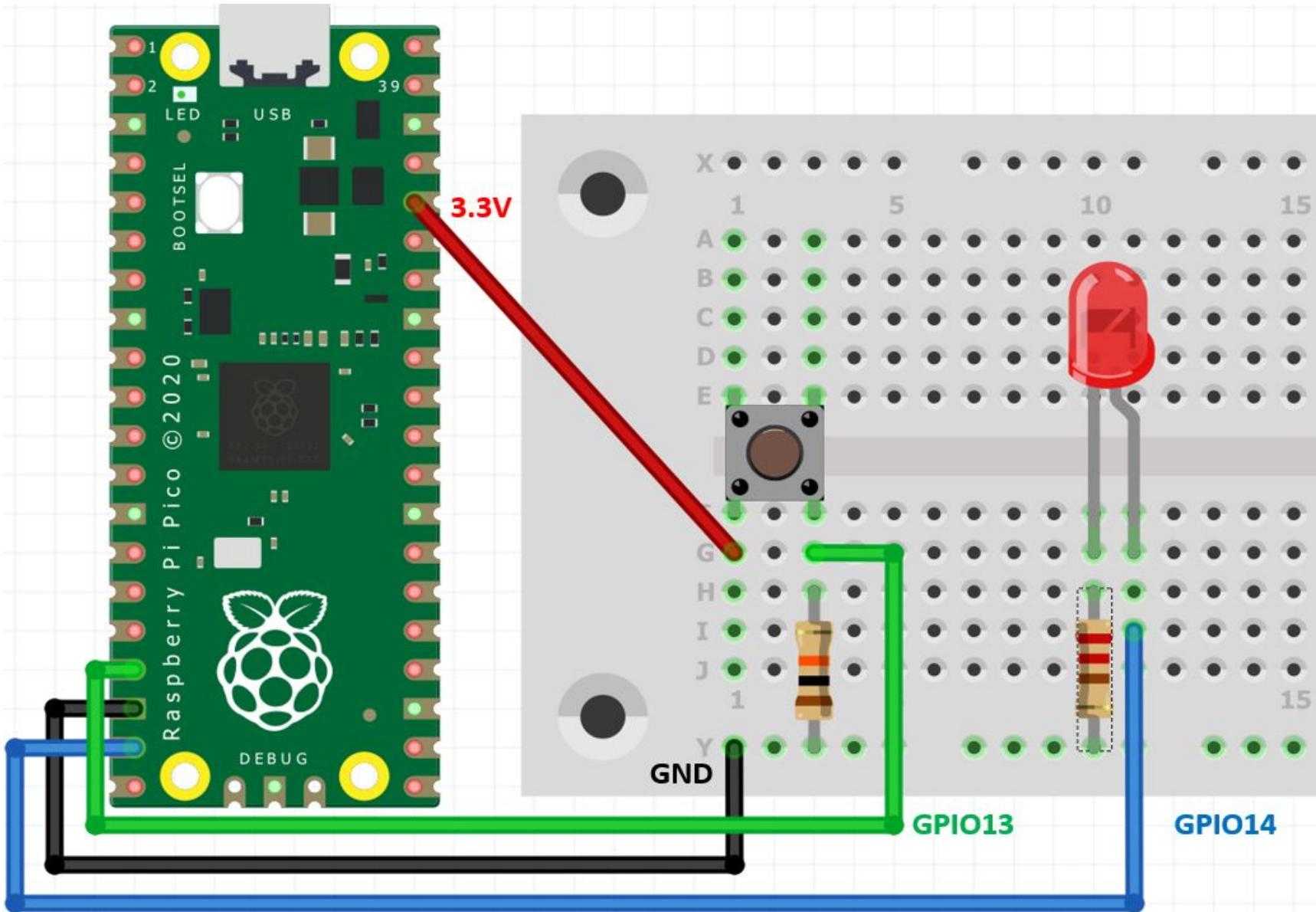
# Interface Push Button with Raspberry Pi Pico and Control LED

## Objective:

If you press the push button, the LED will glow and if you leave the push button, a LED will remain off.



# Interface Push Button with Raspberry Pi Pico and Control LED



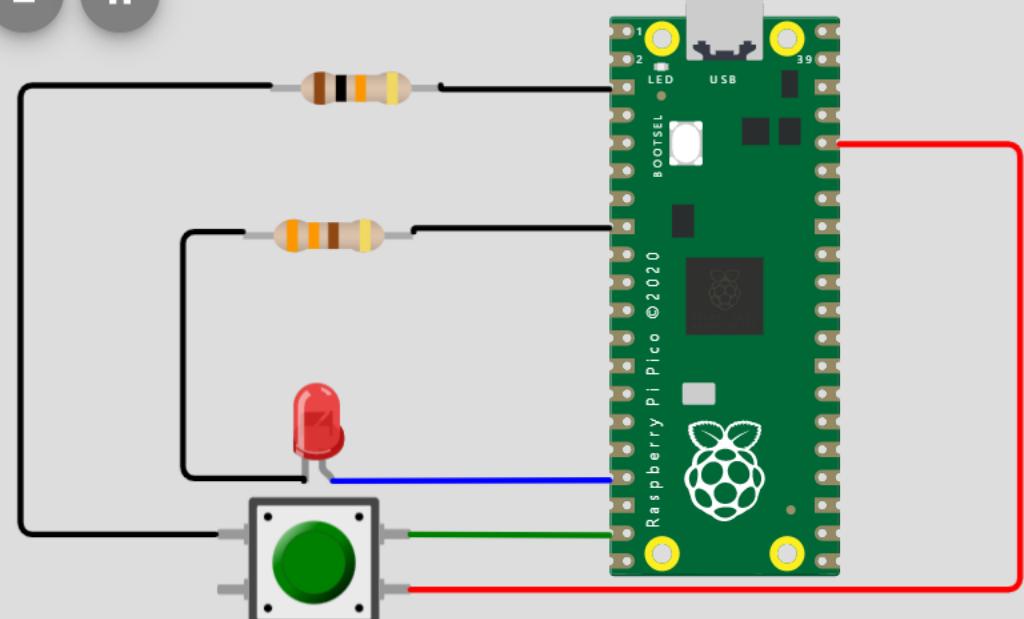
# Interface Push Button with Raspberry Pi Pico and Control LED

WOKWI SAVE SHARE Docs B

main.py • diagram.json PIO 

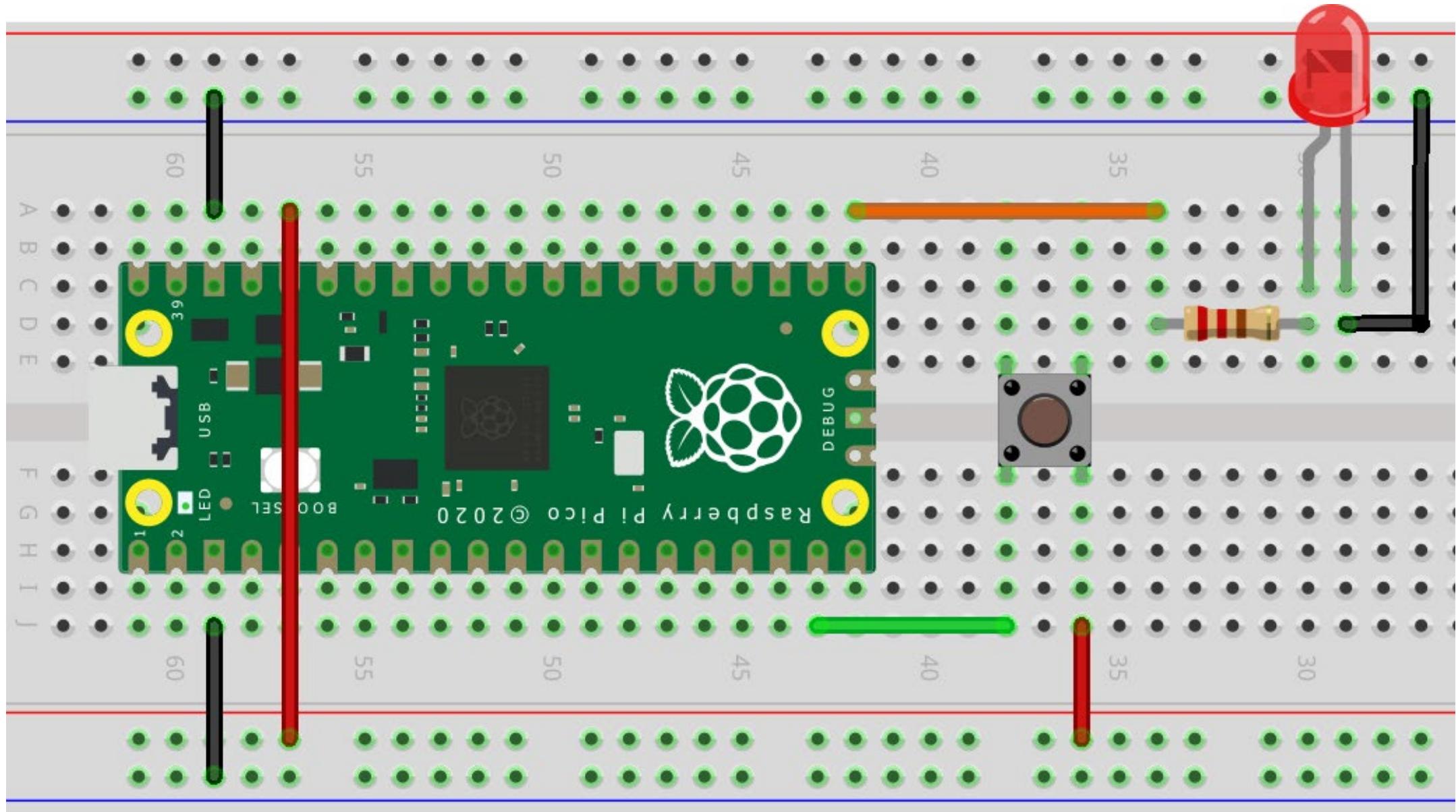
```
1  from machine import Pin
2  from utime import sleep
3
4  led = Pin(13, Pin.OUT)
5  button = Pin(14, Pin.IN)
6
7  while True:
8      if button.value() == 1:
9          led.value(1)
10         print("LED is ON")
11         sleep(0.3)
12     elif button.value() == 0:
13         led.value(0)
14         '''print("LED is OFF")'''
15         sleep(0.3)
16
17
18
19
20
21
22
23
24
```

Simulation 00:37.429 99%



LED is ON  
LED is ON

# With the internal pulldown resistor of the digital input



# With the internal pulldown resistor of the digital input

WOKWi SAVE SHARE Docs B

main.py • diagram.json PIO 🐍

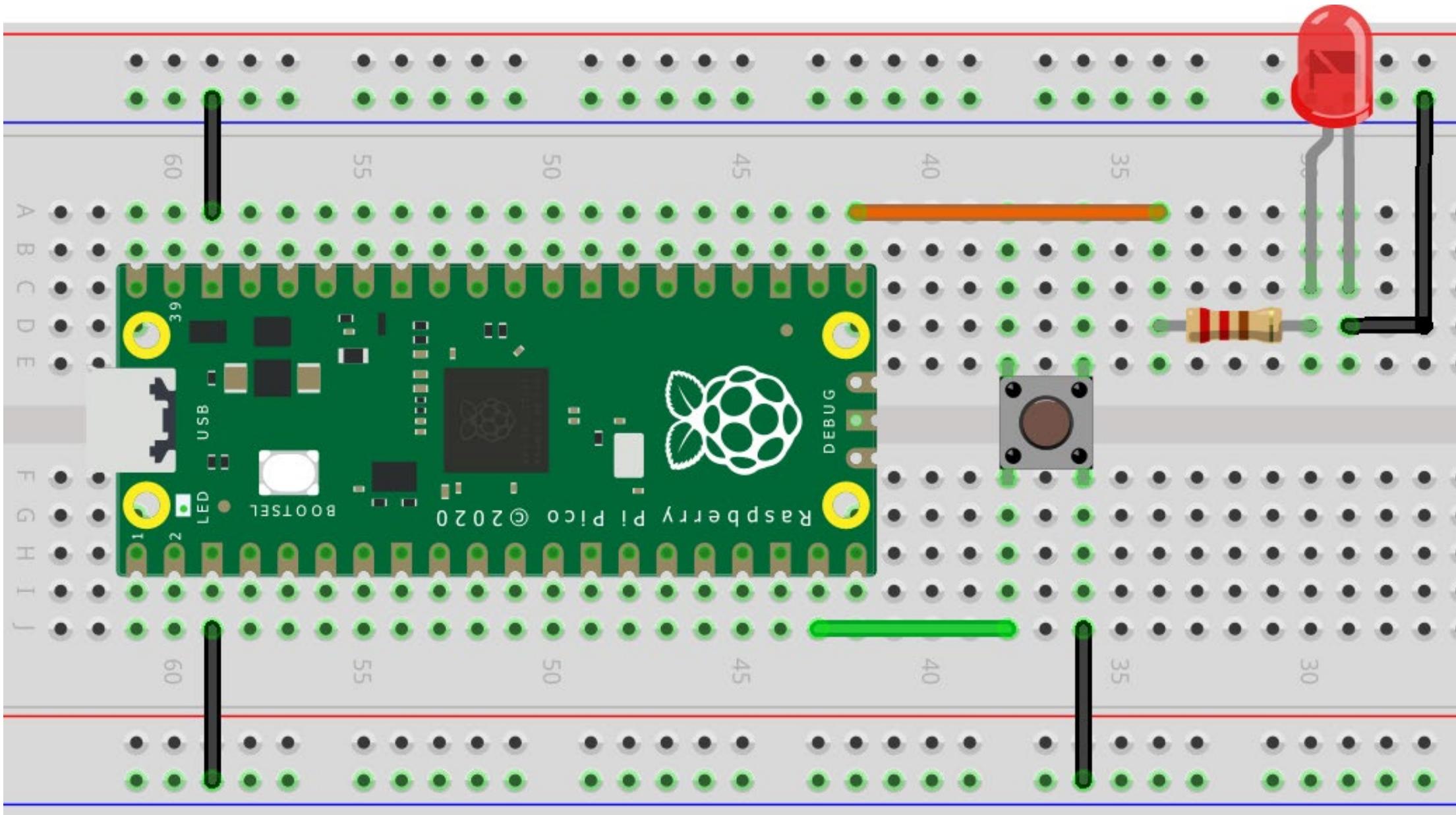
```
1  from machine import Pin
2  from utime import sleep
3
4  button = Pin(14, mode = Pin.IN, pull = Pin.PULL_DOWN)
5  led = Pin(16, Pin.OUT)
6
7  while True:
8      if button.value() == 1:
9          led.value(1)
10         print("LED is ON")
11         sleep(0.3)
12     elif button.value() == 0:
13         led.value(0)
14         '''print("LED is OFF")'''
15         sleep(0.3)
16
17
```

Simulation

00:27.146 99%

LED is ON

# With the internal pullup resistor of the digital input



# With the internal pullup resistor of the digital input

WOKWi ▾ SAVE ▾ SHARE ▾ Docs B

main.py diagram.json PIO

```
1 from machine import Pin
2 from utime import sleep
3
4 button = Pin(14, mode = Pin.IN, pull = Pin.PULL_UP)
5 led = Pin(16, Pin.OUT)
6
7 while True:
8     if button.value() == 0:
9         led.value(1)
10        print("LED is ON")
11        sleep(0.3)
12    elif button.value() == 1:
13        led.value(0)
14        '''print("LED is OFF")'''
15        sleep(0.3)
16
17
```

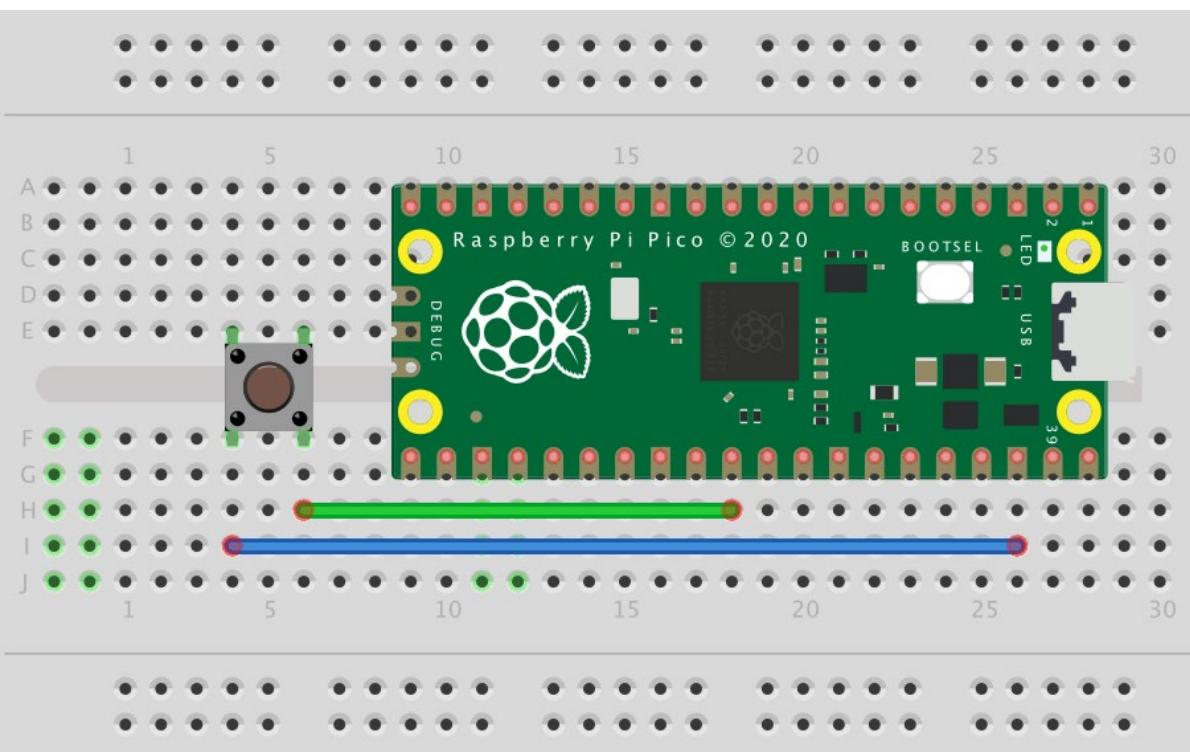
Simulation

00:23.913 99%

LED is ON  
LED is ON

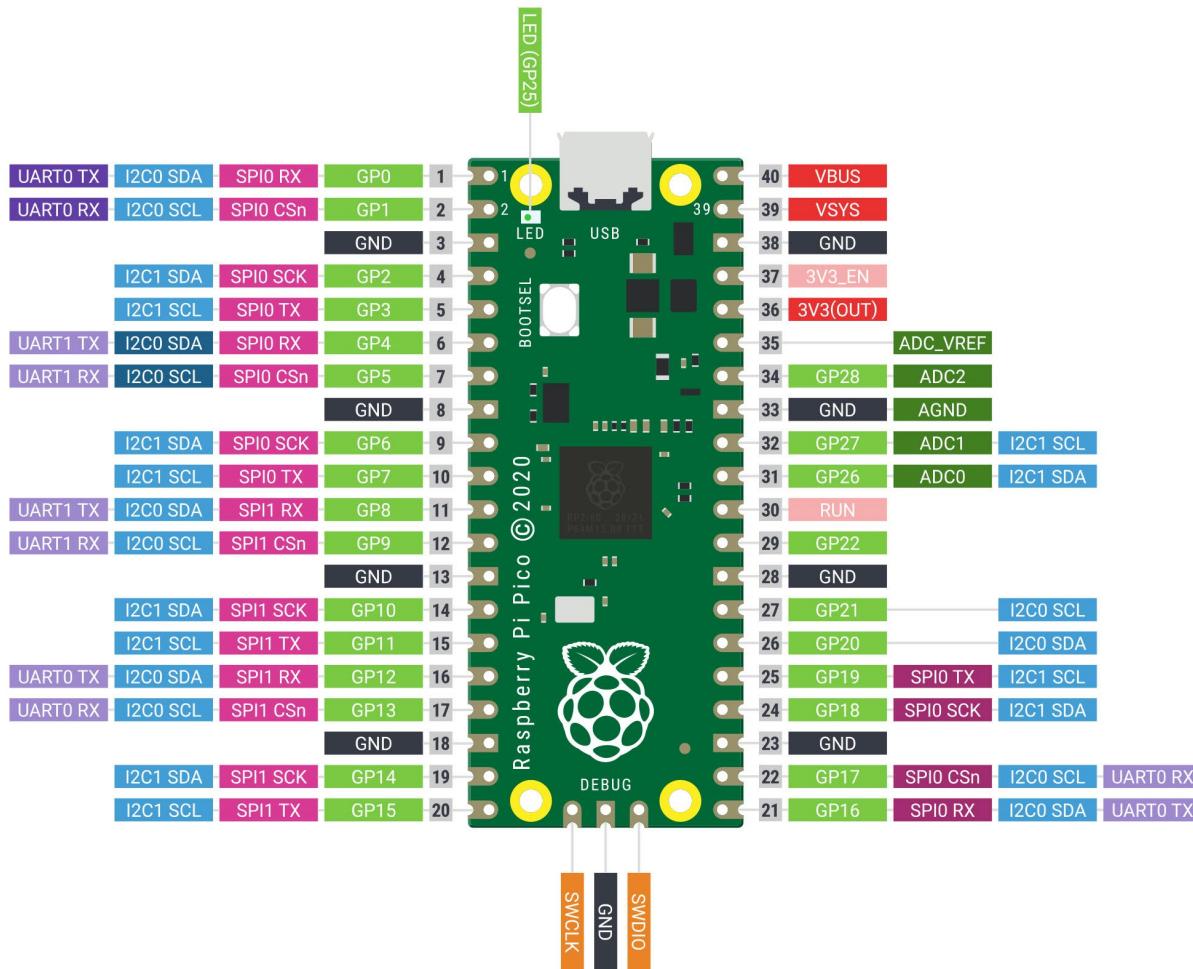
# Push Button as reset button in Raspberry Pi Pico

When you press and hold the BOOTSEL button and connect your Pico to your computer, it mounts as a mass storage volume. You can then just drag and drop a UF2 file onto the board. Sometimes it is not convenient to keep unplugging the micro USB cable every time you want to upload a UF2 onto the Pico. Having a reset button on your Raspberry Pi Pico resolves this problem.



- Place the Pico on the breadboard.
- Place the button on the breadboard.
- Connect a jumper from one button pin to a **GND** pin on the Pico, we used **pin 38**.
- Connect a jumper wire from the other button pin to **RUN**, **pin 30** on the Pico.

# Push Button as reset button in Raspberry Pi Pico



Now, rather than unplugging and replugging the USB cable when you want to load code:

- Push and hold the reset button.
- Push the BOOTSEL button.
- Release the reset button.
- Release the BOOTSEL button.

If your board is in BOOTSEL mode and you want to restart code already loaded:

- Briefly push the reset button.

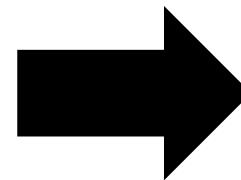
If we press the button on the breadboard it will pull the RUN pin down to 0V, forcing the Pico to reset. Press the Stop button to reconnect to the MicroPython shell. If your code ever locks up then this trick will help get you back to work.

# Controlling an LED using a Push Button as Toggle Switch

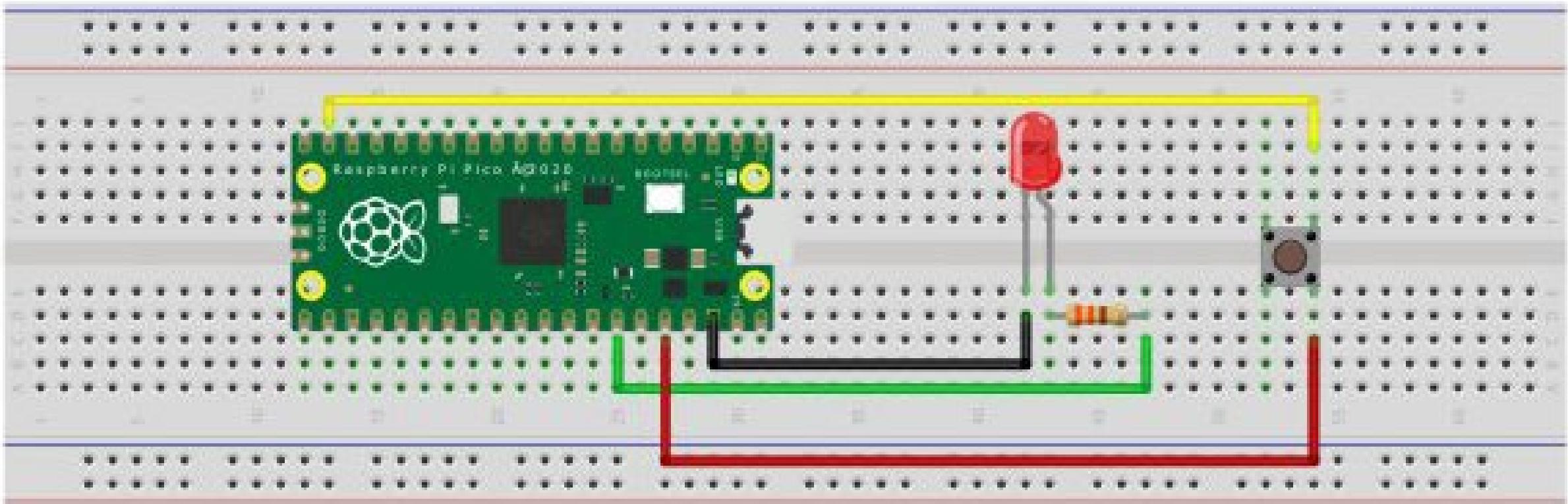
Objective:

press once – LED is ON

press again – LED is OFF and so on.



**Push Button as Toggle Switch**



# Controlling an LED using a Push Button as Toggle Switch

WOKWI SAVE SHARE Docs B

main.py • diagram.json • PIO 1

```
1 from machine import Pin
2 from utime import sleep
3
4 led = Pin(28, Pin.OUT)
5 button = Pin(14, Pin.IN, Pin.PULL_DOWN)
6
7 while True:
8
9     if button.value():
10        led.toggle()
11        sleep(0.5)
12
```

Simulation

The screenshot shows a WOKWI development environment for a Raspberry Pi Pico. On the left, the code editor displays a Python script named 'main.py' with the following code:

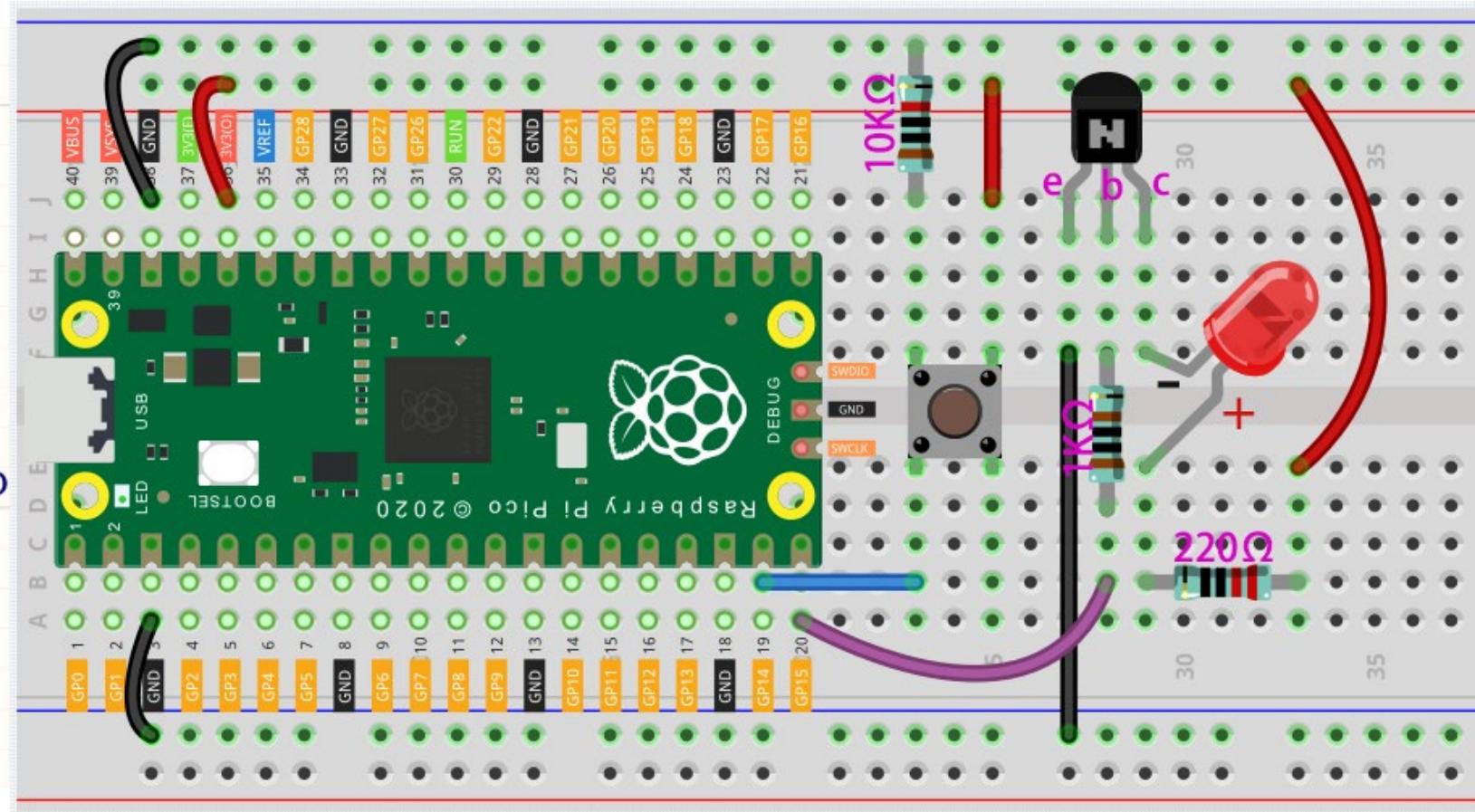
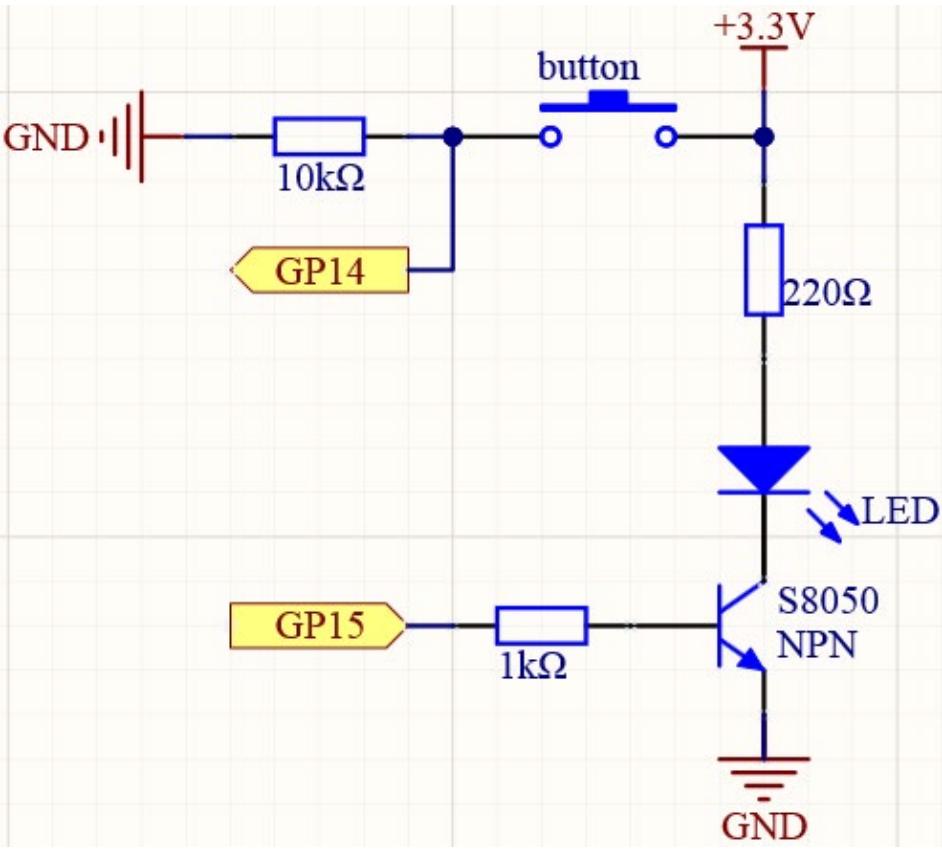
```
1 from machine import Pin
2 from utime import sleep
3
4 led = Pin(28, Pin.OUT)
5 button = Pin(14, Pin.IN, Pin.PULL_DOWN)
6
7 while True:
8
9     if button.value():
10        led.toggle()
11        sleep(0.5)
12
```

The right side of the interface features a simulation window titled 'Simulation'. It includes a 'Raspberry Pi Pico @ 2020' board model, a red LED, and a push button. The circuit is connected as follows: the push button is connected to pin 14 of the Pico with a pull-down resistor. The LED is connected to pin 28 of the Pico via a resistor. The simulation status bar at the top right indicates '00:19.550 27%'. The WOKWI interface also has standard navigation buttons for saving, sharing, and documentation.

# Controlling an LED and a buzzer using a Push Button

- 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.
- 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.
- 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.
- 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.

# Controlling an LED and a transistor using a Push Button



When we press the button, Pico will send a high-level signal to the transistor (i.e., LED or BUZZER will be ON) ; when we release it, it will send a low-level signal (i.e., LED or BUZZER will be OFF).

Replace the LED with BUZZER

# Controlling an LED and a transistor using a Push Button

```
from machine import Pin
from utime import sleep

button = Pin(14, Pin.IN)
signal = Pin(15, Pin.OUT)

while True:
    if button.value() == 1:
        signal.value(1)
        print("Transistor is ACTIVE")
        print("LED/BUZZER is ON")
        sleep(0.3)
    elif button.value() == 0:
        signal.value(0)
        '''print("Transistor is DEACTIVE")'''
        sleep(0.3)
```

- 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.

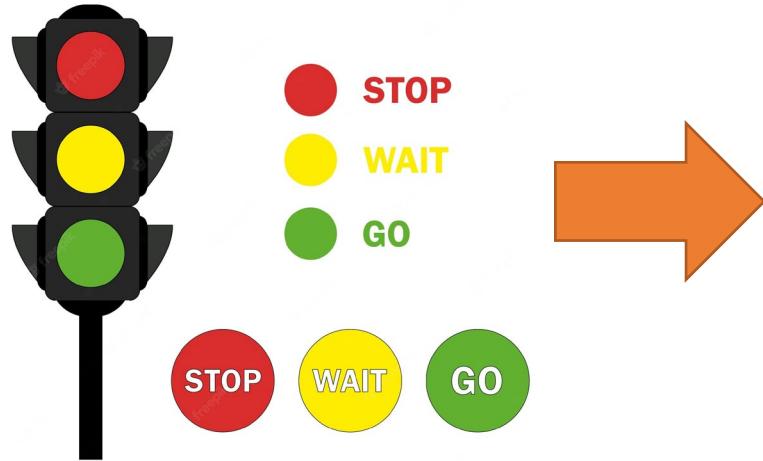
## ➤ Application of Transistor:

1) as an Amplifier

2) as a switch

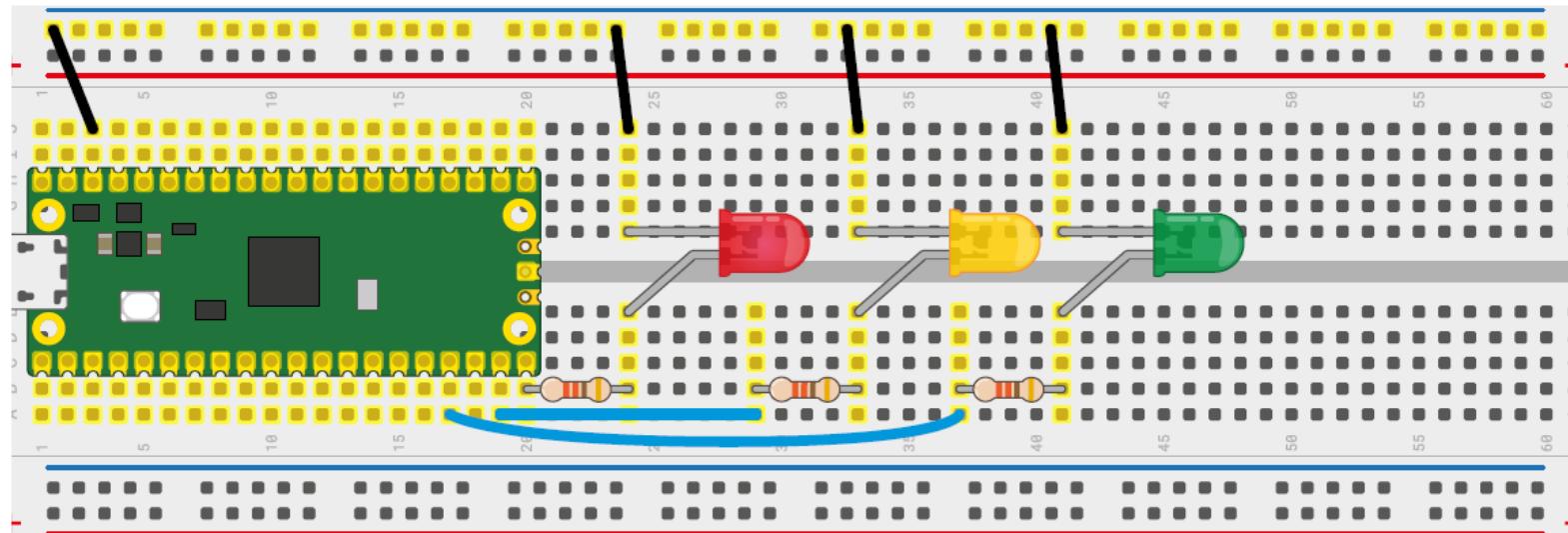
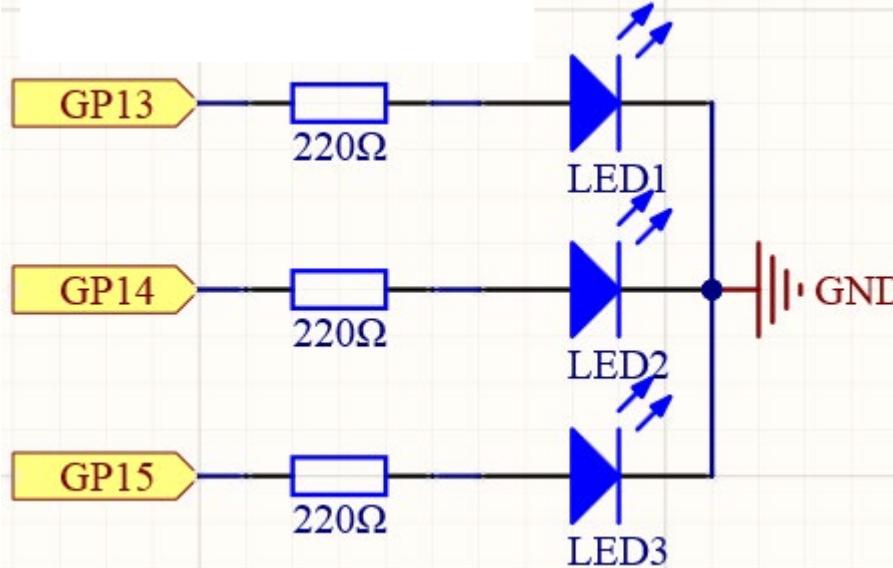
# A basic three-light traffic light system

Traffic lights are used to direct traffic operation and are generally composed of red, green, and yellow lights.



Step-by-step, turning the LEDs on and off.

- a red light to tell the traffic to stop
- an amber or yellow light to tell the traffic the light is about to change,
- a green LED to tell the traffic it can go again



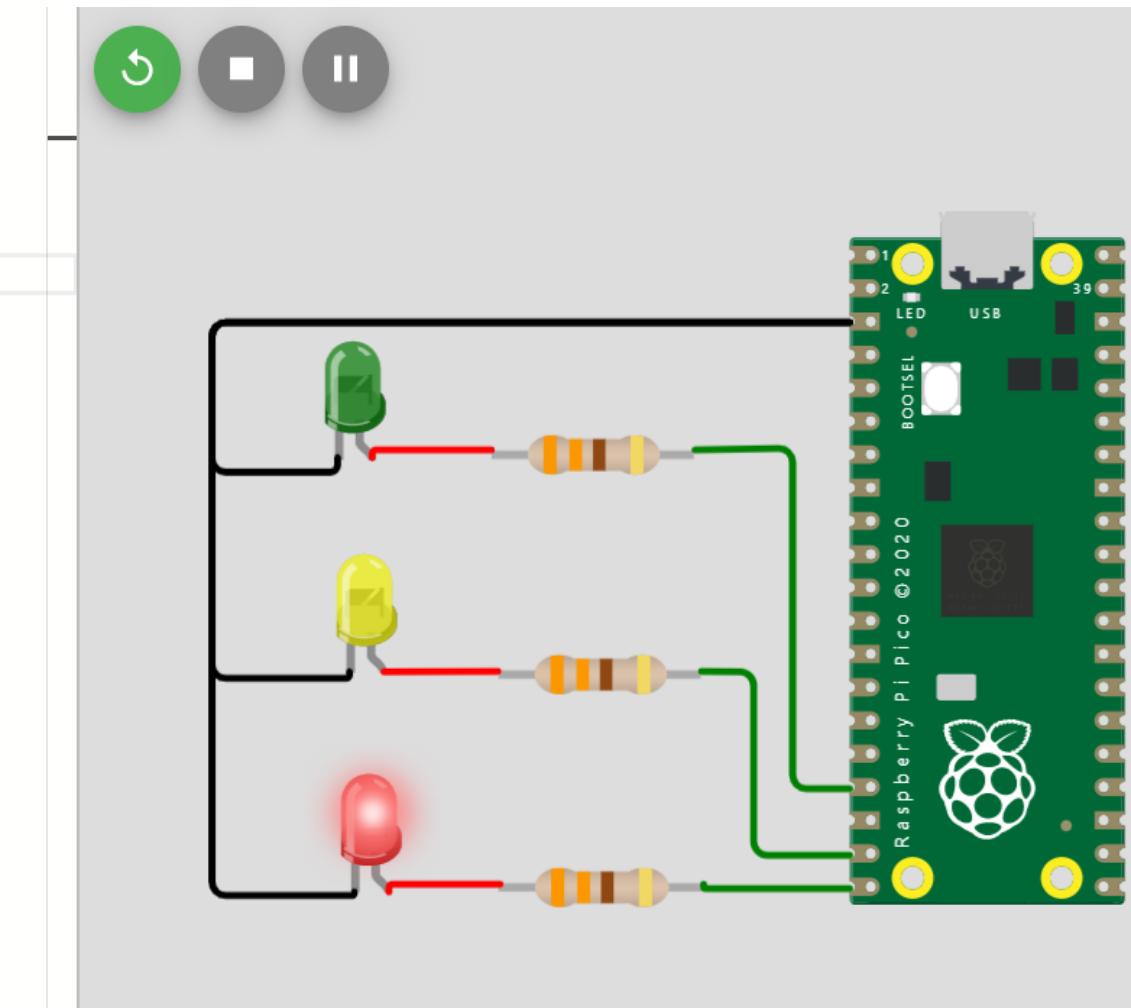
A simple traffic light Control system without proper control

# Mini pedestrian crossing system

## Objective:

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.**

```
1 from machine import Pin
2 from utime import sleep
3
4 led_red = Pin(15, Pin.OUT)
5 led_yellow = Pin(14, Pin.OUT)
6 led_green = Pin(13, Pin.OUT)
7
8 while True:
9
10     led_red.value(1)
11     sleep(5)
12     led_red.value(0)
13
14     led_yellow.value(1)
15     sleep(2)
16     led_yellow.value(0)
17
18     led_green.value(1)
19     sleep(5)
20     led_green.value(0)
21
22     led_yellow.value(1)
23     sleep(2)
24     led_yellow.value(0)
```



# Puffin Crossing (Mini pedestrian crossing system with proper control)

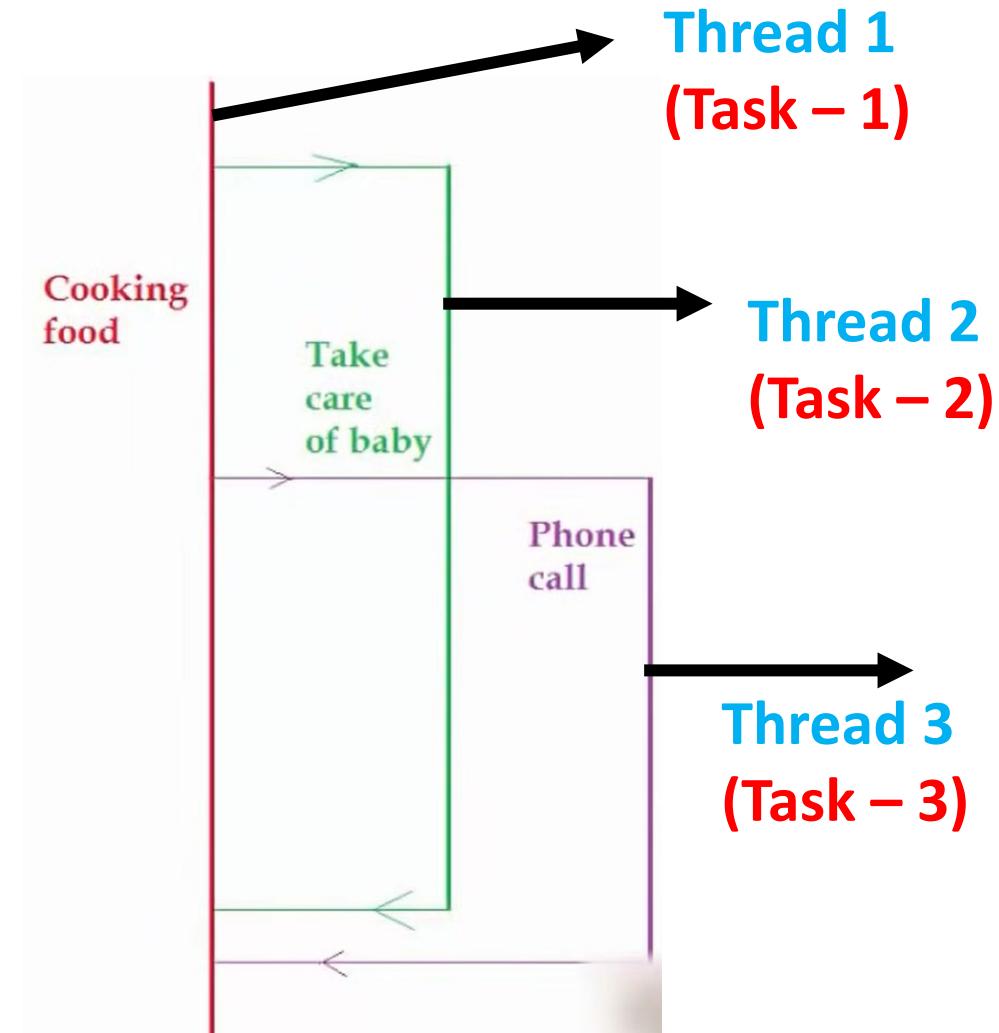
Through a push-button switch, the pedestrian can ask the lights to let them cross the road; and a buzzer, so the pedestrian knows when it's their turn to cross. **If we (pedestrians) press the button, the red LED will be extended to 15 seconds, which will give us more time to cross the road.**



# Puffin Crossing (Mini pedestrian crossing system with proper control)

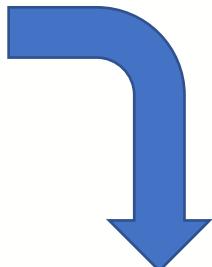
In puffin crossing our program should able to record whether the button has been pressed in a way that doesn't interrupt the traffic lights. To make that work, you'll need a new library: \_thread.

## Multithreading Concept



# Multiple action without threading

```
1 import time
2
3
4 def calc_square(numbers):
5     print("calculate square numbers")
6     for n in numbers:
7         time.sleep(1)
8         print('square:',n*n)
9
10 def calc_cube(numbers):
11     print("calculate cube of numbers")
12     for n in numbers:
13         time.sleep(1)
14         print('cube:',n*n*n)
15
16 arr = [2,3,8,9]
17
18 t = time.time()
19
20 calc_square(arr)
21 calc_cube(arr)
22
23 print("done in : ",time.time()-t)
24 print("Hah... I am done with all my work now!")
```



Execute this in THONNY and check the processing time

For a given list of numbers print square and cube of every numbers

For example:

Input: [2,3,8,9]

Output: square list - [4, 9, 64, 81]  
cube list - [8, 27, 512, 729]

calculate square numbers

square: 4

square: 9

square: 64

square: 81

calculate cube of numbers

cube: 8

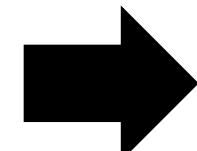
cube: 27

cube: 512

cube: 729

done in : 8.065548181533813

Hah... I am done with all my work now!



# Multiple action without threading

Thonny - C:\Users\biswa\hello.py @ 22 : 34

The screenshot shows the Thonny IDE interface. The top menu bar includes New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, Quit, and Support. A file named "hello.py" is open in the editor, containing the following Python code:

```
1 import time
2
3 def calc_square(numbers):
4     print("calculate square numbers")
5     for n in numbers:
6         time.sleep(1)
7         print('square:',n*n)
8
9 def calc_cube(numbers):
10    print("calculate cube of numbers")
11    for n in numbers:
12        time.sleep(1)
13        print('cube:',n*n*n)
14
15 arr = [2,3,8,9]
16
17 t = time.time()
18
19 calc_square(arr)
20 calc_cube(arr)
21
22 print("done in : ",time.time()-t)
23 print("Hah... I am done with all my work now!")
24
```

The Shell tab at the bottom displays the output of running the script:

```
>>> %Run hello.py
calculate square numbers
square: 4
square: 9
square: 64
square: 81
calculate cube of numbers
cube: 8
cube: 27
cube: 512
cube: 729
done in : 8.08883786201477
Hah... I am done with all my work now!
```

# Multiple action with threading (MULTITHREADING)

```
1 import time
2 import threading
3
4 def calc_square(numbers):
5     print("calculate square numbers")
6     for n in numbers:
7         time.sleep(1)
8         print('square:',n*n)
9
10 def calc_cube(numbers):
11     print("calculate cube of numbers")
12     for n in numbers:
13         time.sleep(1)
14         print('cube:',n*n*n)
15
16 arr = [2,3,8,9]
17
18 t = time.time()
19
20 t1= threading.Thread(target=calc_square, args=(arr,))
21 t2= threading.Thread(target=calc_cube, args=(arr,))
22
23 t1.start()
24 t2.start()
25
26 t1.join()
27 t2.join()
28
29 print("done in : ",time.time()-t)
30 print("Hah... I am done with all my work now!")
```

Execute this MULTITHREADING in THONNY and check the processing time

calculate square numbers  
calculate cube of numbers  
square:cube: 48

cube:square: 279

square:cube: 64512

square:cube: 81729

done in : 4.064005613327026

Hah... I am done with all my work now!

The RP2040 microcontroller which powers our Pico has two processing cores

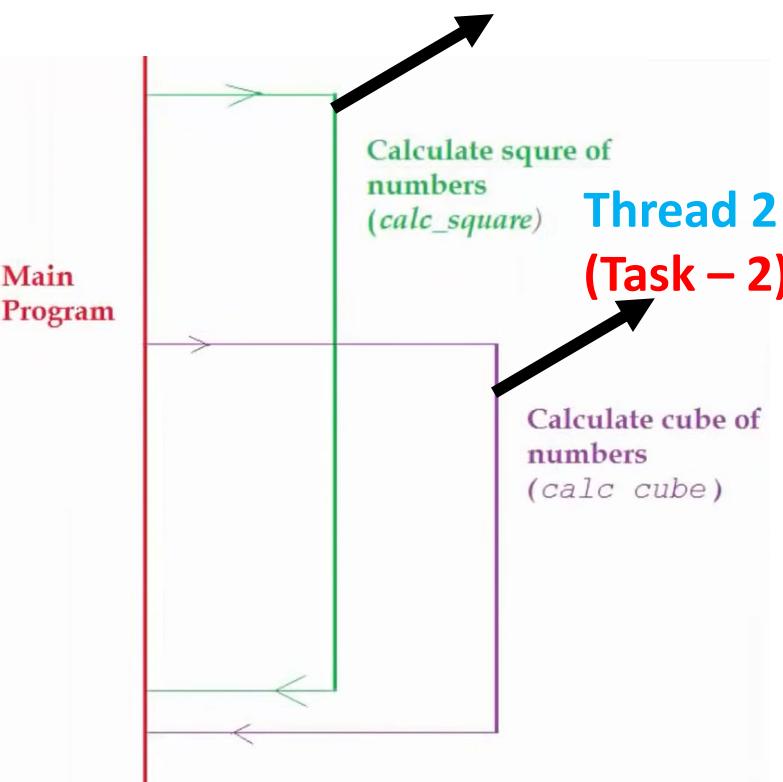
Thread 1  
(Task - 1)

Calculate square of numbers (calc\_square)

Thread 2  
(Task - 2)

Calculate cube of numbers (calc\_cube)

Main Program



# Multiple action with threading (MULTITHREADING)

Thonny - C:\Users\biswa\hello.py @ 30 : 48

The screenshot shows the Thonny Python IDE interface. The top menu bar includes New, Load, Save, Run, Debug, Over, Into, Out, Stop, Zoom, Quit, and Support. A toolbar below the menu has icons for New (green plus), Load (document), Save (disk), Run (play), Debug (script), Over (down arrow), Into (up arrow), Out (list), Stop (red square), Zoom (magnifying glass), Quit (red X), and Support (Ukrainian flag). A code editor window titled 'hello.py' contains the following Python script:

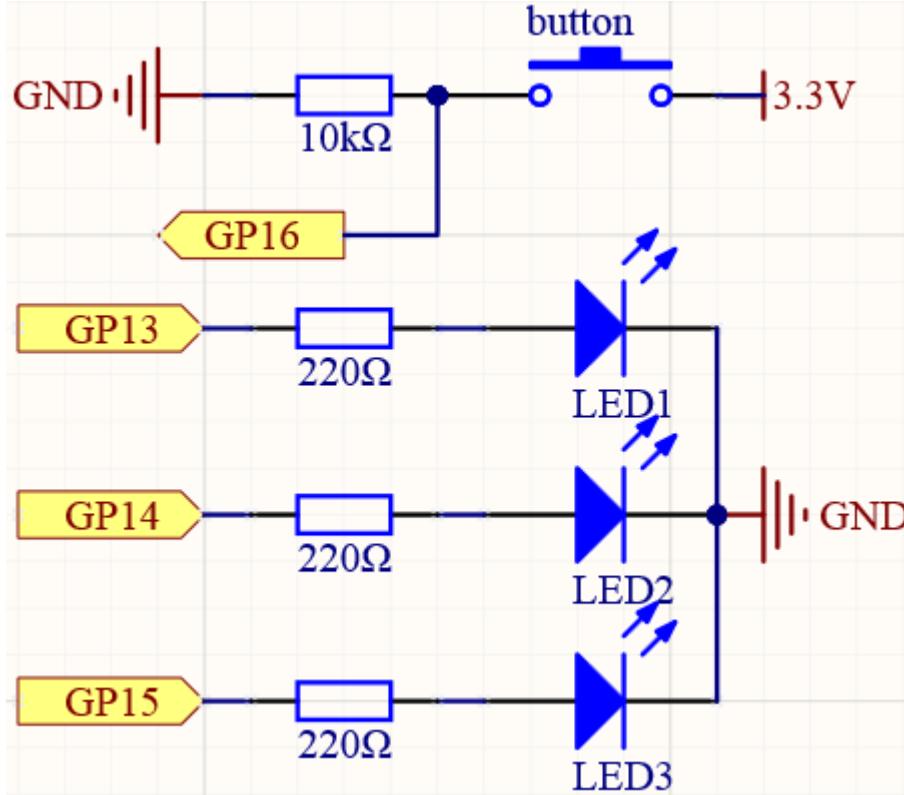
```
1 import time
2 import threading
3
4 def calc_square(numbers):
5     print("calculate square numbers")
6     for n in numbers:
7         time.sleep(1)
8         print('square:',n*n)
9
10 def calc_cube(numbers):
11     print("calculate cube of numbers")
12     for n in numbers:
13         time.sleep(1)
14         print('cube:',n*n*n)
15
16 arr = [2,3,8,9]
17
18 t = time.time()
19
20 t1= threading.Thread(target=calc_square, args=(arr,))
21 t2= threading.Thread(target=calc_cube, args=(arr,))
22
23 t1.start()
24 t2.start()
25
26 t1.join()
27 t2.join()
28
29 print("done in : ",time.time()-t)
30 print("Hah... I am done with all my work now!")
```

Below the code editor is a 'Shell' window displaying the output of the script:

```
calculate square numbers
calculate cube of numbers
square: 4
cube: 8
square: 9
cube: 27
square: 64
cube: 512
square: 81
cube: 729
done in :  4.058969020843506
Hah... I am done with all my work now!
```

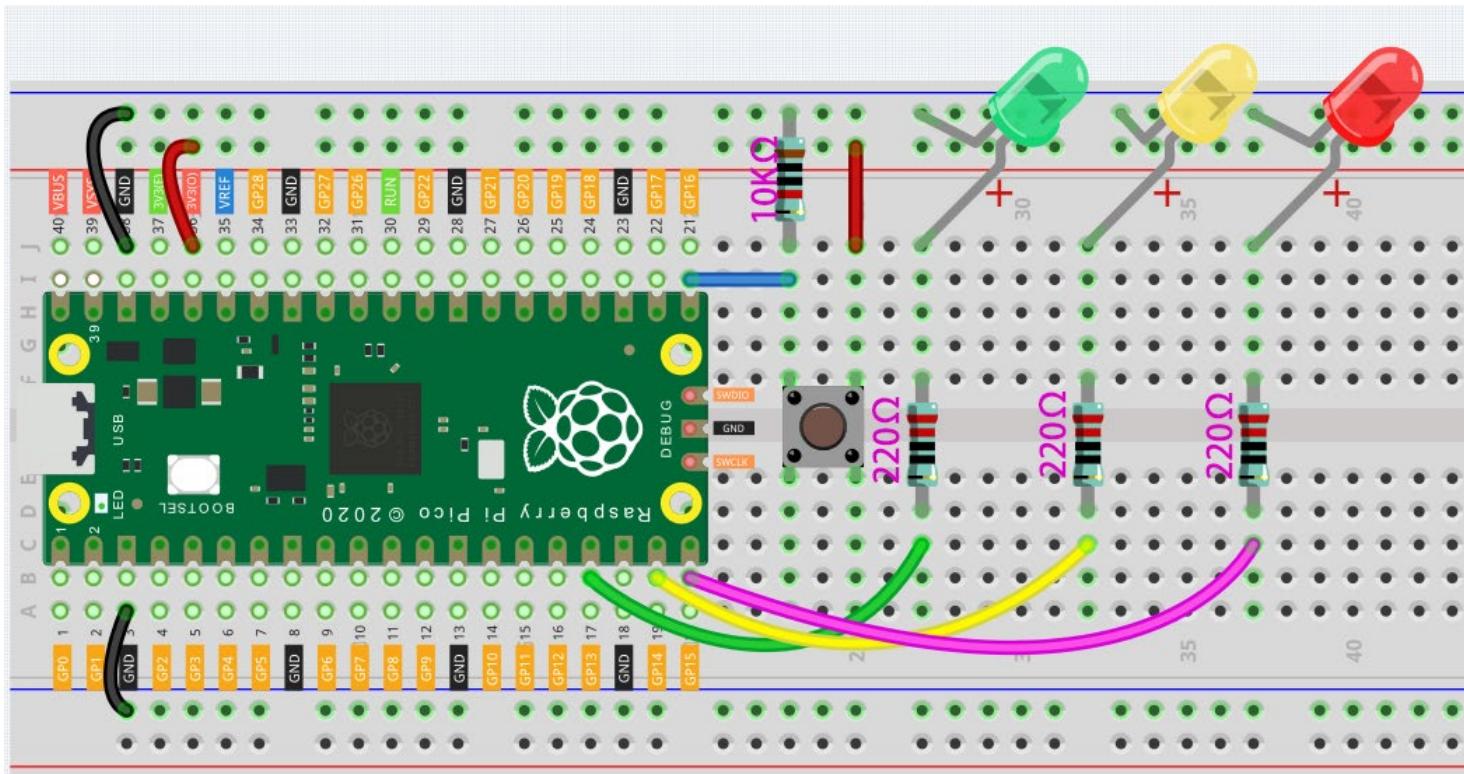
# Puffin Crossing (Mini pedestrian crossing system with proper control)

In puffin crossing our program should able to record whether the button has been pressed in a way that doesn't interrupt the traffic lights. To make that work, you'll need a new library: [\\_thread](#).



➤ controls the lights, as the main thread

➤ New thread to pass information back to the main thread – and you can do this using [global variables](#). The variables working with prior to this are known as [local variables](#), and only work in one section of program; a global variable works everywhere, meaning [one thread can change the value and another can check to see if it has been changed](#).

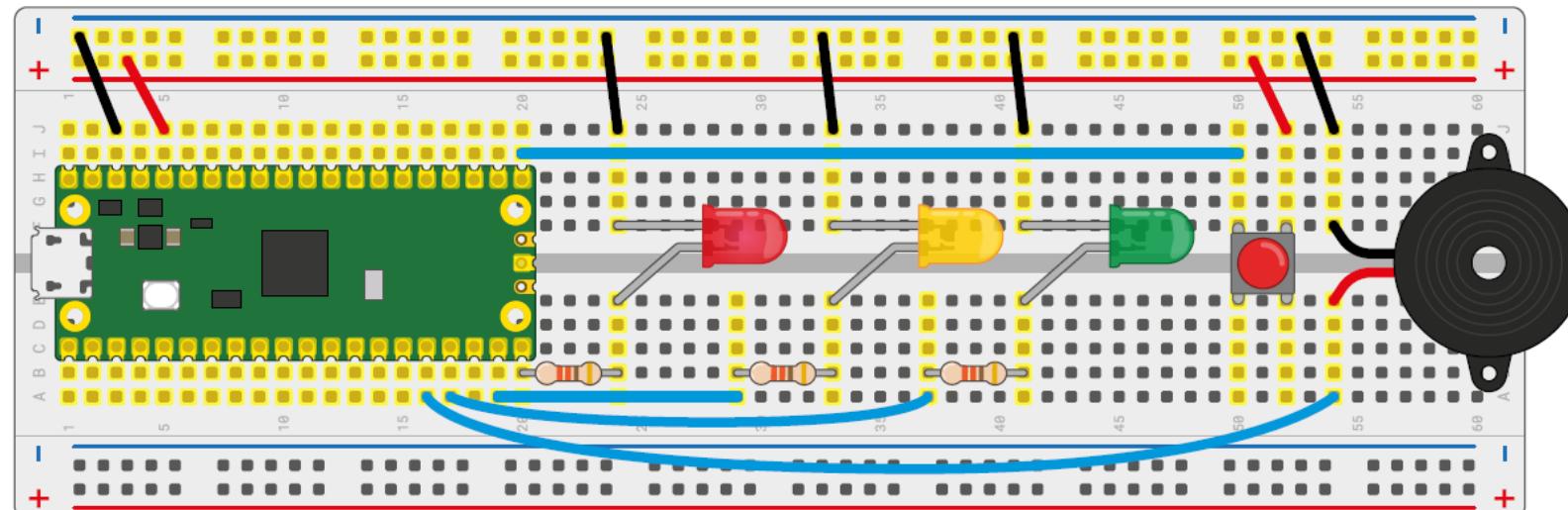
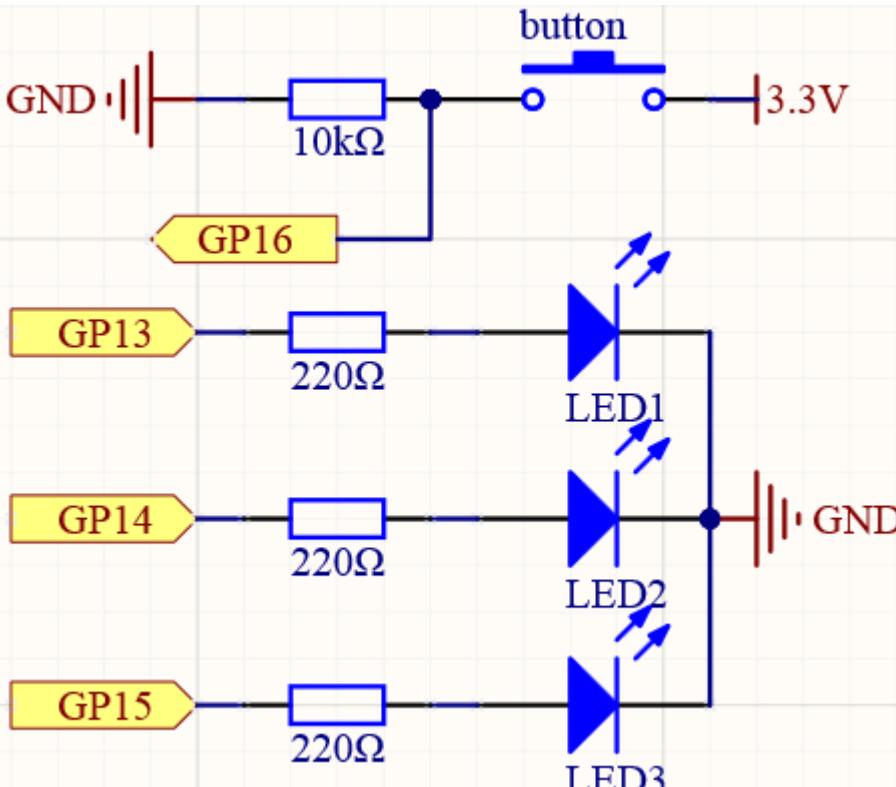


A simple traffic light Control system with proper control

# Puffin Crossing (Mini pedestrian crossing system with proper control)

## Objective:

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.



A simple traffic light Control system with proper control  
(With BUZZER)

# Puffin Crossing (Mini pedestrian crossing system with proper control)

## Objective:

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.

```
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(15)
        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 15 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.

# Puffin Crossing (Mini pedestrian crossing system with proper control)

WOKWI SAVE SHARE

main.py diagram.json

Simulation

```
1 from machine import Pin
2 from utime import sleep
3 import _thread
4
5 led_red = Pin(15, Pin.OUT)
6 led_yellow = Pin(14, Pin.OUT)
7 led_green = Pin(13, Pin.OUT)
8 button = Pin(16, Pin.IN, Pin.PULL_UP)
9 buzzer = Pin(12, Pin.OUT)
10
11 global button_status
12 button_status = 0
13
14 def button_thread():
15     global button_status
16     while True:
17         if button.value() == 1:
18             button_status = 1
19
20 _thread.start_new_thread(button_thread, ())
21
22 while True:
23     if button_status == 1:
24         led_red.value(1)
25         for i in range(10):
26             buzzer.value(1)
27             sleep(0.5)
28             buzzer.value(0)
29             sleep(0.5)
30     global button_status
31     button_status = 0
32
33     led_red.value(0)
34     sleep(5)
35     led_red.value(1)
36
37     led_yellow.value(1)
38     sleep(2)
39     led_yellow.value(0)
40
41     led_green.value(1)
42     sleep(5)
43     led_green.value(0)
44
45     led_yellow.value(1)
46     sleep(2)
47     led_yellow.value(0)
```

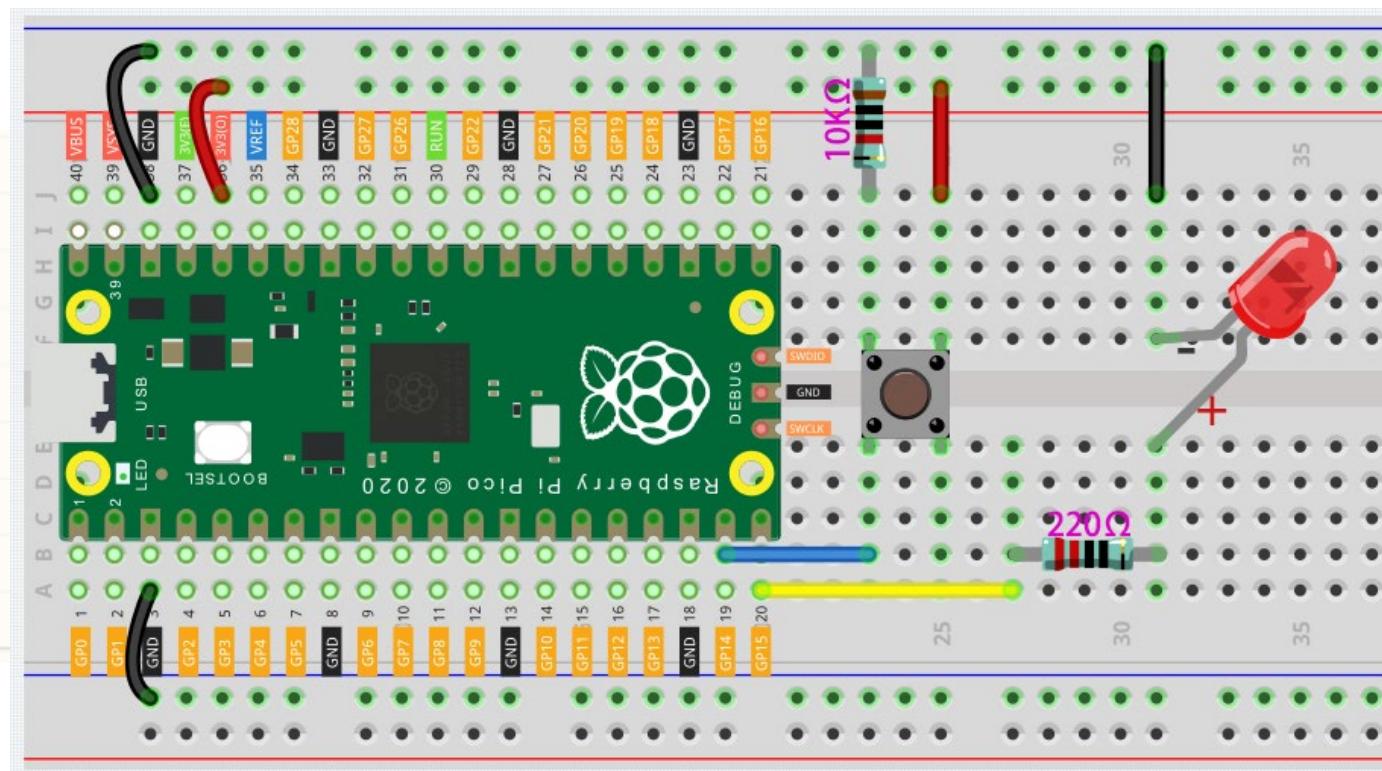
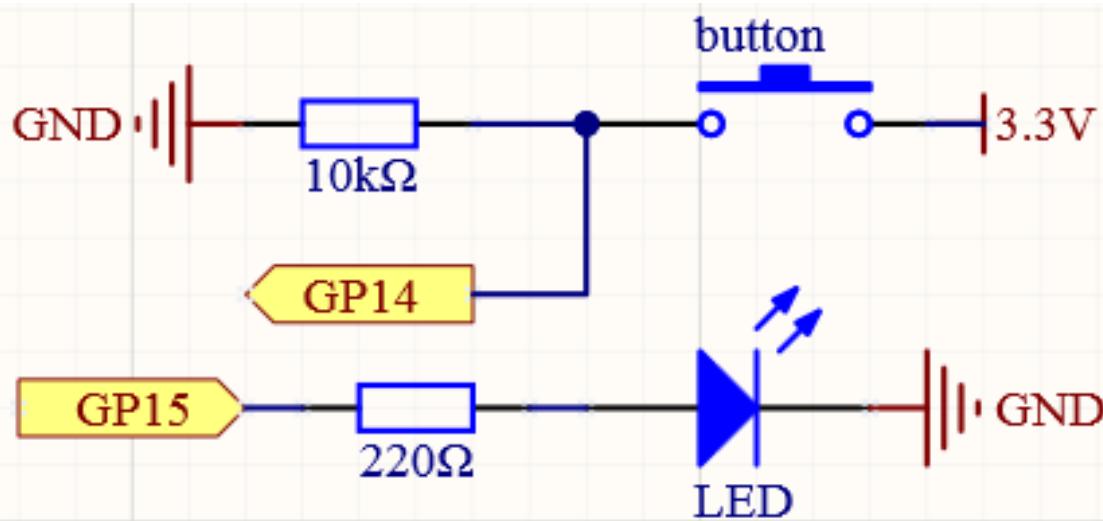
The diagram illustrates a mini pedestrian crossing system using a Raspberry Pi Pico. The circuit consists of a speaker connected to pin 39, three LEDs (red, yellow, green) with resistors, and a pushbutton connected to pin 16 via a pull-up resistor. The code uses threads to handle button presses and control the LEDs with proper timing.

# Simple Reaction Timing Game (Single Player)

**The study of reaction time is known as mental chronometry**

**Your reaction time** – the time it takes your brain to process the need to do something and send the signals to make that something happen – is **measured in milliseconds**: the average human reaction time is around **200–250 milliseconds**. People with quick reaction time have a huge advantage in the game!

# A single-player game



## Objective:

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.

# Interrupt Requests, or IRQs

## 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.

```
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.

# Interrupt Requests, or IRQs

- The job of the **interrupt handler** is to service the device and stop it from interrupting. Once the handler returns, the CPU resumes what it was doing before the interrupt occurred.
- 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.
- 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.

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

# Simple Reaction Timing Game (Single Player)

## A single-player game

WOKWi SAVE SHARE Docs B

main.py diagram.json • PIO

```
1 import machine
2 import utime
3 import urandom
4
5 led = machine.Pin(15, machine.Pin.OUT)
6 button = machine.Pin(14, machine.Pin.IN)
7
8 def button_press(pin):
9     button.irq(handler=None)
10    rection_time = utime.ticks_ms() - timer_light_off
11    print("Your reaction time was " + str(rection_time) + " milliseconds!")
12
13 led.value(1)
14 utime.sleep(urandom.uniform(5, 10))
15 led.value(0)
16 timer_light_off = utime.ticks_ms()
17 button.irq(trigger=machine.Pin.IRQ_RISING, handler=button_press)
```

Simulation

00:24.246 100%

Raspberry Pi Pico with RP2040

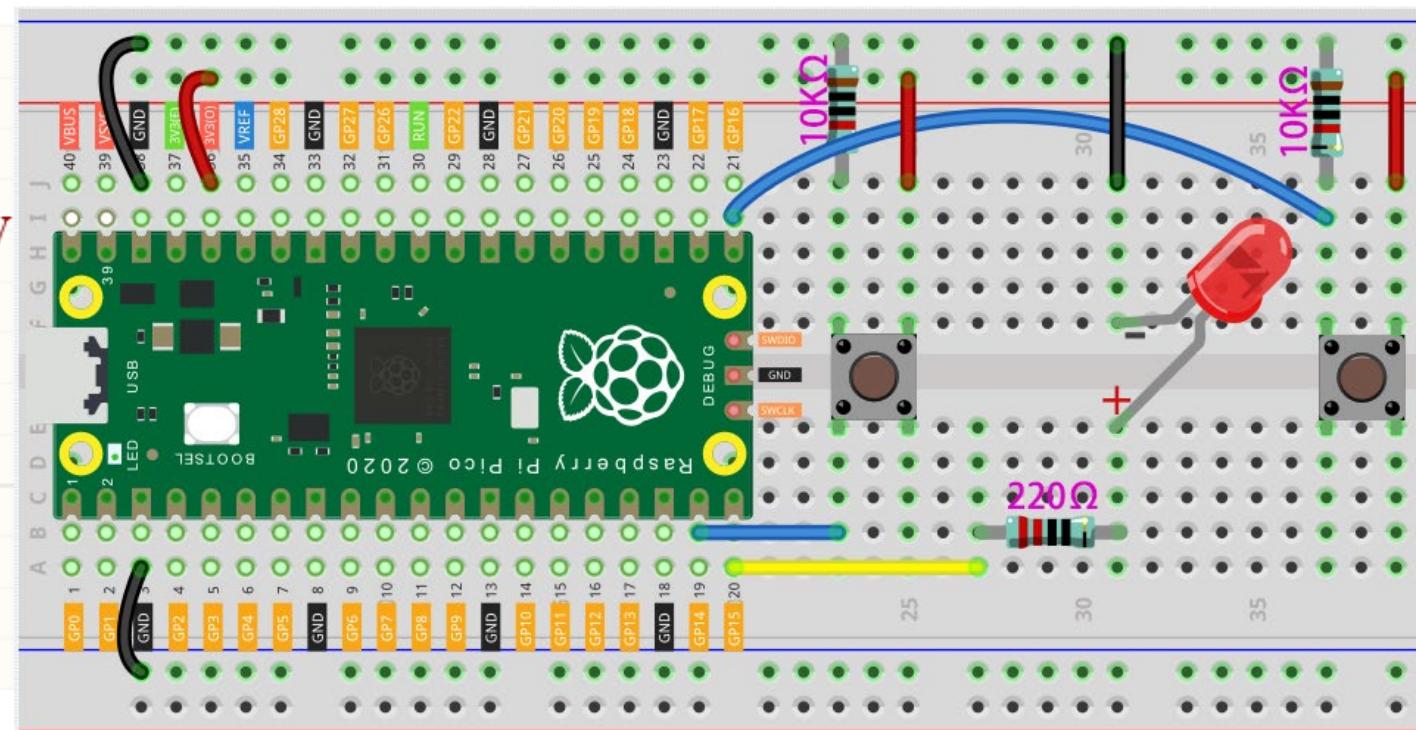
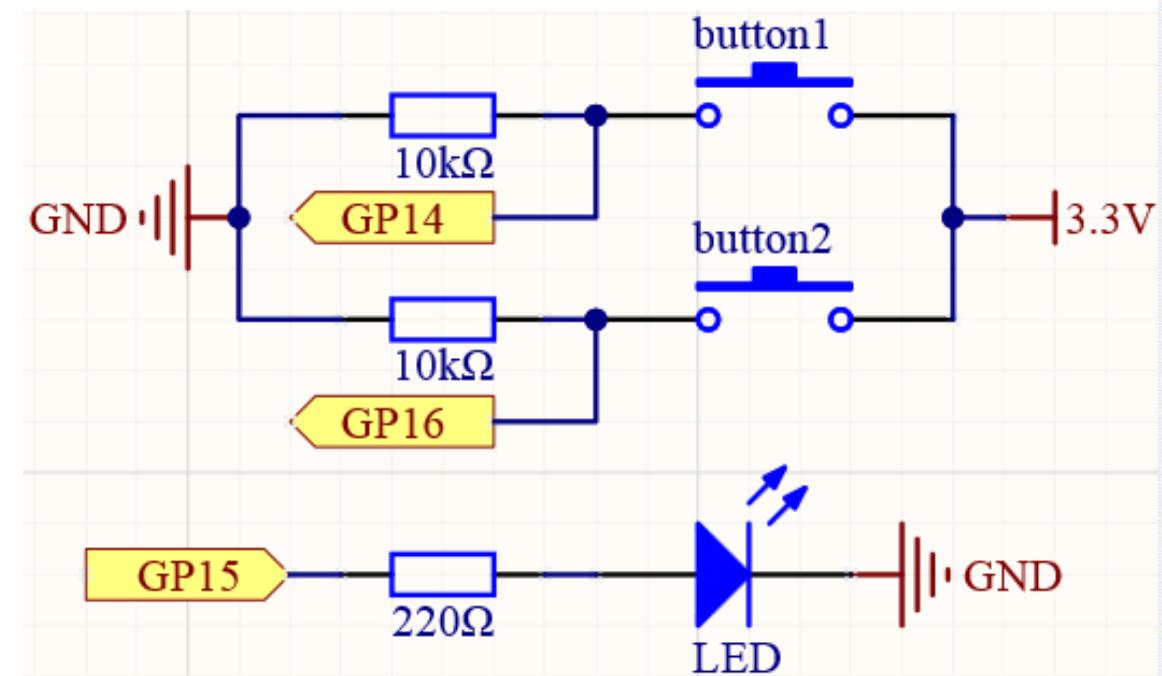
```
MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
MPY: soft reboot
MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> Your reaction time was 473 milliseconds!
```

# Simple Reaction Timing Game (Two Player)

## Objective:

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

### A two-player game



# Simple Reaction Timing Game (Two Player)

## A two-player game

```
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)
```

### INTERRUPTS AND HANDLERS

Each interrupt you create needs a handler, but a single handler can deal with as many interrupts as you like. In the case of this program, you have two interrupts both going to the same handler – meaning that whichever interrupt triggers, they'll run the same code. A different program might have two handlers, letting each interrupt run different code – it all depends on what you need your program to do.



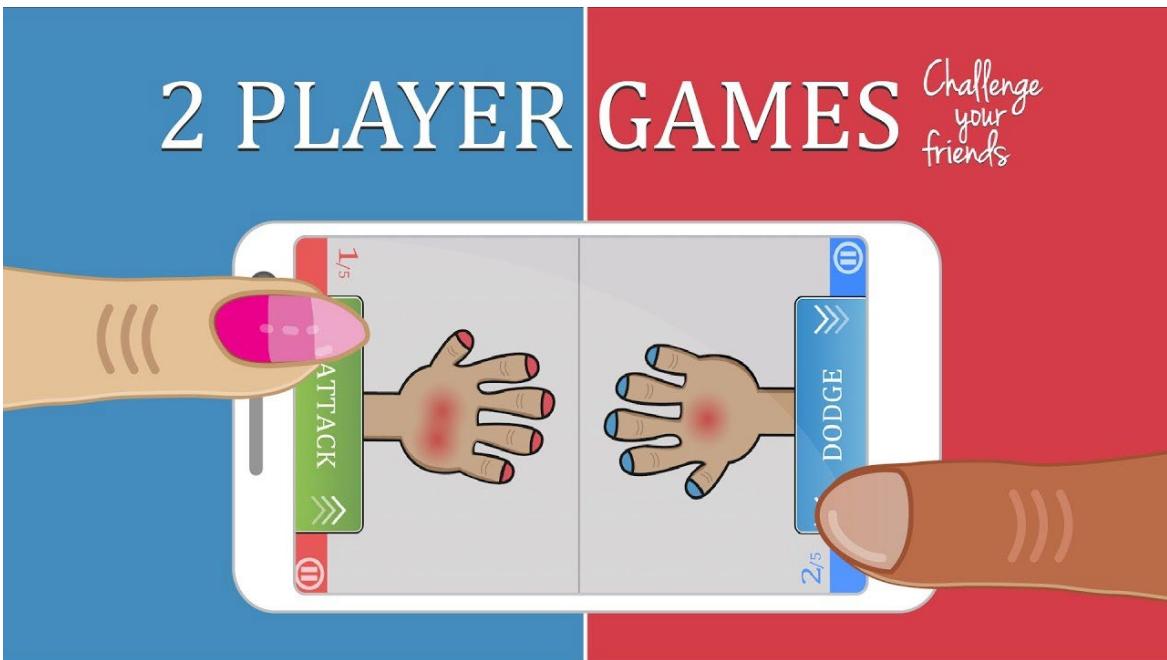
- Press the Run button and wait for the LED to go out – but don't press either of the push-button switches just yet.
- You'll see that the Shell area remains blank, and doesn't bring back the >>> prompt; that's because the main thread is still running, sitting in the loop you created.

# Simple Reaction Timing Game (Two Player)

## More Exciting two-player game

### Objective:

To know which of the two players was the first to press the button.



```
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)

fastest_button = None

def button_handler(pin):
    left_button.irq(handler=None)
    right_button.irq(handler=None)
    global fastest_button
    fastest_button = pin

    led.value(1)
    utime.sleep(urandom.uniform(5, 10))
    led.value(0)

    left_button.irq(trigger=machine.Pin.IRQ_RISING, handler=button_handler)
    right_button.irq(trigger=machine.Pin.IRQ_RISING, handler=button_handler)

while fastest_button is None:
    utime.sleep(1)

if fastest_button is left_button:
    print("Left Player wins!")
elif fastest_button is right_button:
    print("Right Player wins!")
```

# Simple Reaction Timing Game (Two Player)

## A two-player game

WOKwi

SAVE SHARE

Docs B

main.py • diagram.json • PIO

```
1 import machine
2 import utime
3 import urandom
4
5 led = machine.Pin(15, machine.Pin.OUT)
6 left_button = machine.Pin(14, machine.Pin.IN)
7 right_button = machine.Pin(16, machine.Pin.IN)
8
9 def button_press(pin):
10     left_button.irq(handler=None)
11     right_button.irq(handler=None)
12     rection_time = utime.ticks_ms() - timer_light_off
13     if pin == left_button:
14         print("Left player is winner!")
15     elif pin == right_button:
16         print("Right player is winner!")
17     print("Your reaction time was " + str(rection_time) + " milliseconds!")
18
19 led.value(1)
20 utime.sleep(urandom.uniform(5, 10))
21 led.value(0)
22 timer_light_off = utime.ticks_ms()
23 right_button.irq(trigger=machine.Pin.IRQ_RISING, handler=button_press)
24 left_button.irq(trigger=machine.Pin.IRQ_RISING, handler=button_press)
```

Simulation

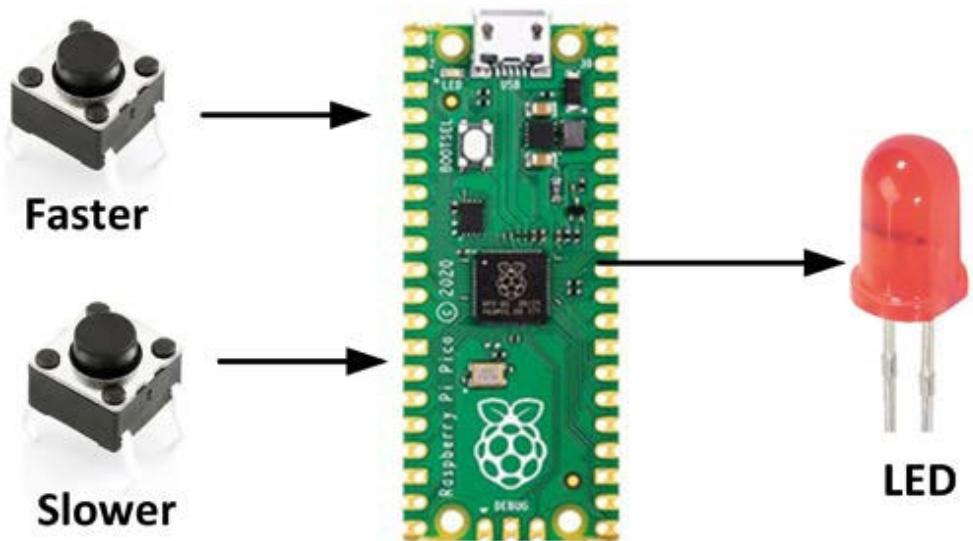
00:24.861 101%

MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040  
Type "help()" for more information.  
>>>  
MPY: soft reboot  
MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040  
Type "help()" for more information.  
>>> Right player is winner!  
Your reaction time was 628 milliseconds!

# Changing the LED flashing rate – using pushbutton interrupts

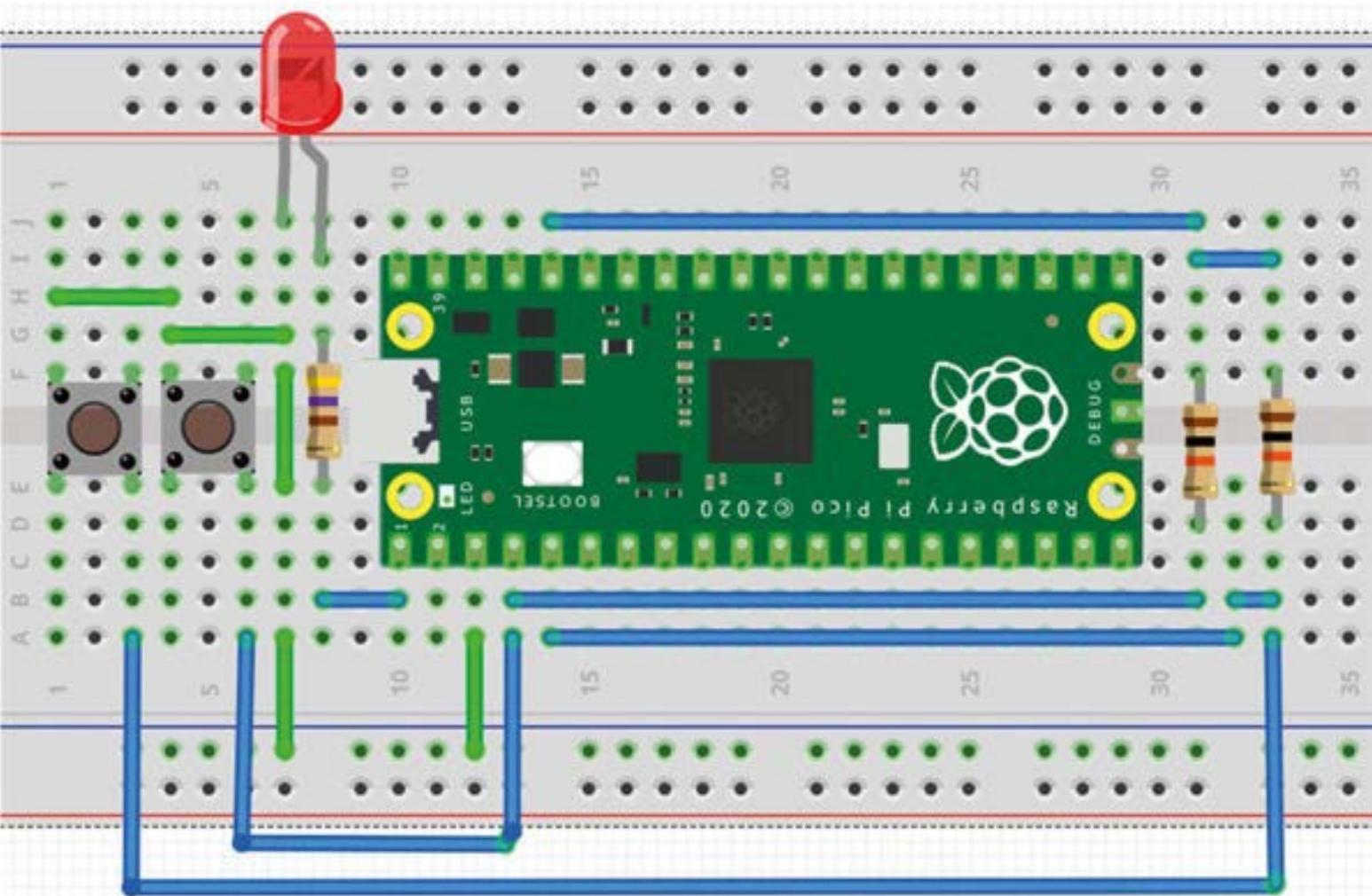
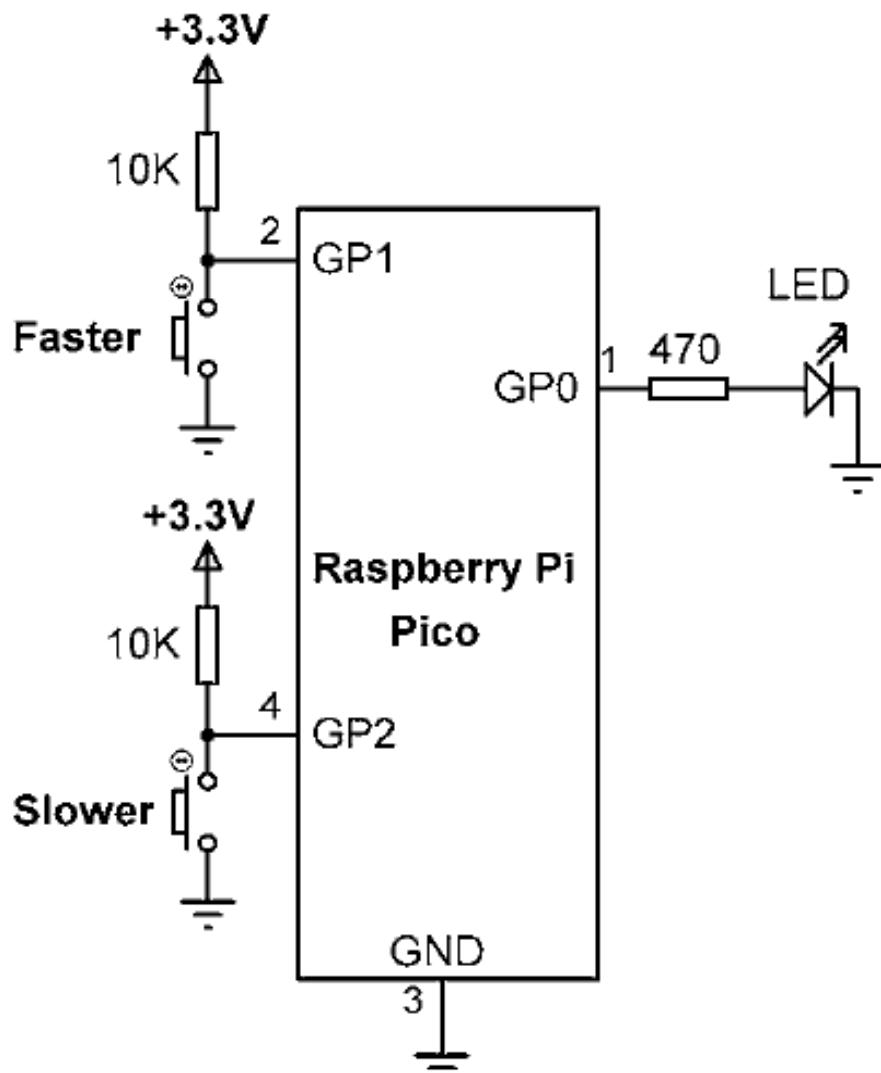
## Objective:

Pushbutton at **GP1** is named **Faster** and pressing this button flashes the LED faster. Similarly, pushbutton at **GP2** is named **Slower**, and pressing this button flashes the LED slower.



At the beginning of the program LED is assigned to pin GP0, and pushbuttons **Faster** and **Slower** are assigned to ports **GP1** and **GP2** respectively. The default flashing rate is set to **one second**. The program is **external – interrupt - based**. Pressing either of the pushbuttons creates an interrupt.

# Changing the LED flashing rate – using pushbutton interrupts



# Changing the LED flashing rate – using pushbutton interrupts

WOKWi

SAVE



SHARE

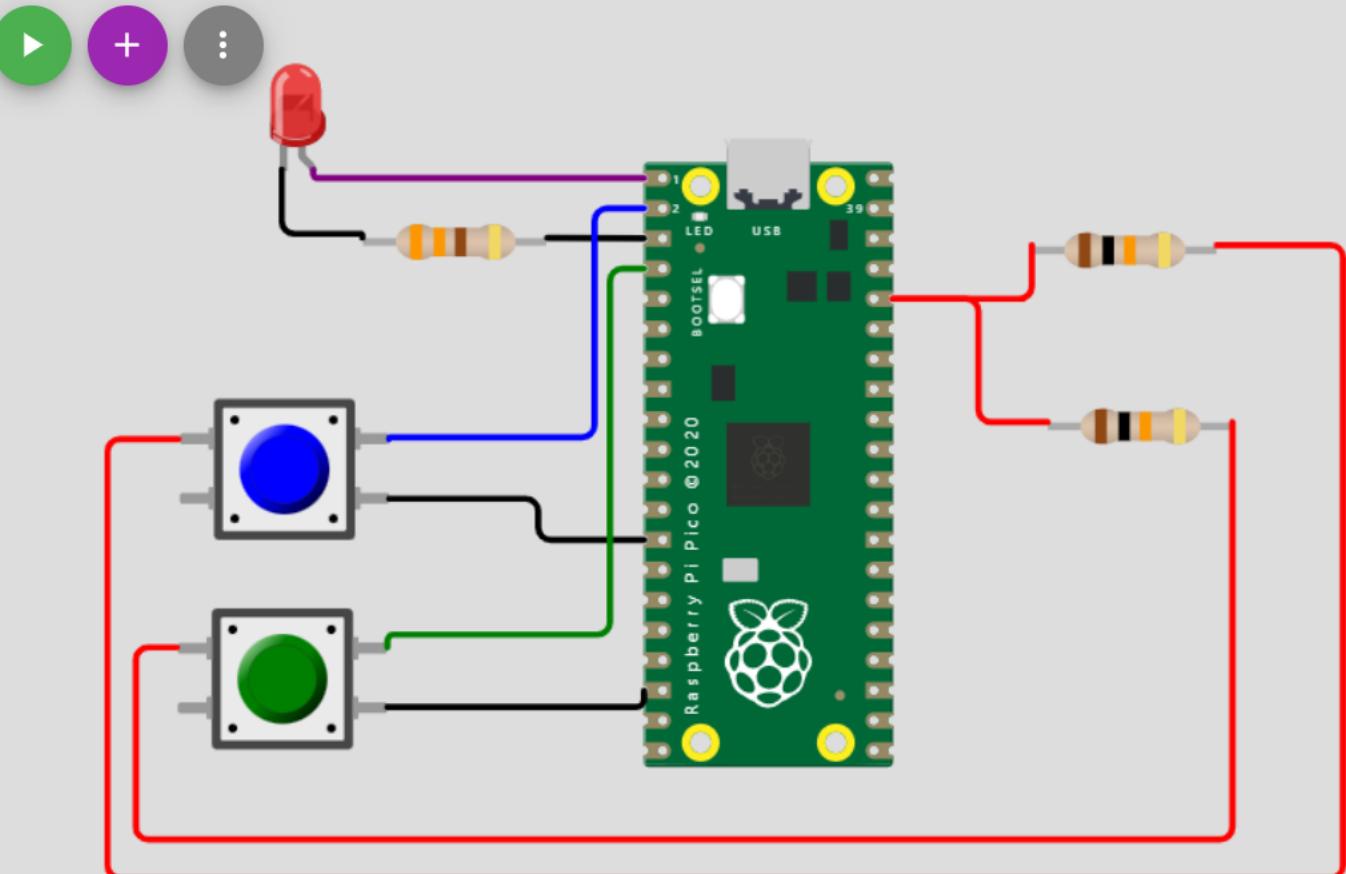
Docs

B

main.py • diagram.json •

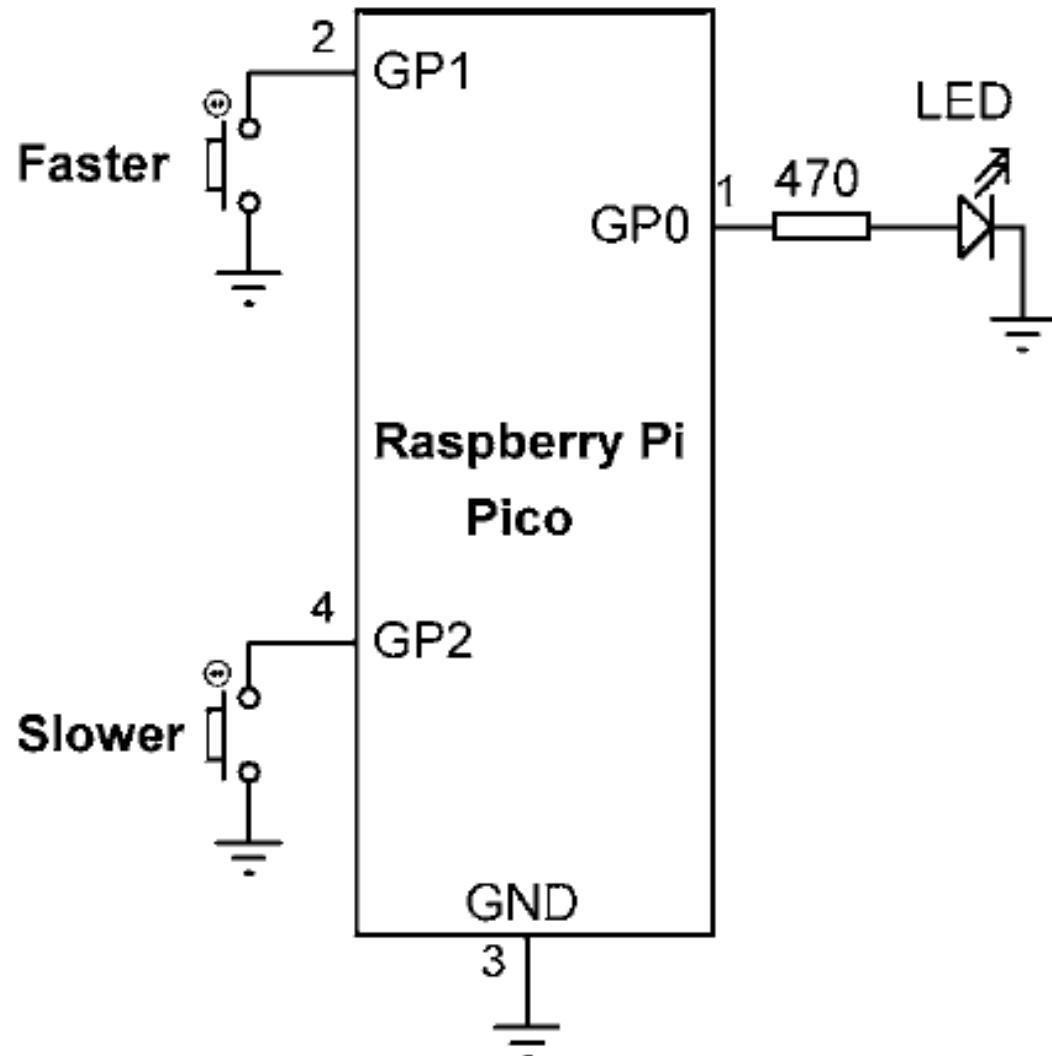
```
1  from machine import Pin
2  from utime import sleep
3
4  LED = Pin(0, Pin.OUT)          # LED at pin GP0
5  Faster = Pin(1, Pin.IN)        # Faster at pin GP1
6  Slower = Pin(2, Pin.IN)        # Slower at pin GP2
7  dly = 1.0                     # Default delay
8
9  def Flash_Faster(Faster):
10     global dly
11     dly = dly - 0.1
12
13 def Flash_Slower(Slower):
14     global dly
15     dly = dly + 0.1
16
17 Faster.irq(handler=Flash_Faster,trigger=Faster.IRQ_FALLING)
18 Slower.irq(handler=Flash_Slower,trigger=Slower.IRQ_FALLING)
19
20 while True:
21     LED.value(1)                # LED ON
22     sleep(dly)                  # Delay dly
23     LED.value(0)                # LED OFF
24     sleep(dly)                  # Delay dly
25
```

Simulation

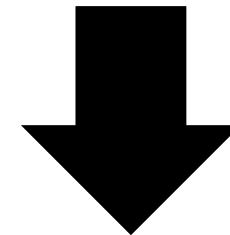


# Changing the LED flashing rate – using pushbutton interrupts

## Using the internal pull-up resistors



Change in the Program



Faster = Pin(1, Pin.IN, pull=Pin.PULL\_UP)

Slower = Pin(2, Pin.IN, pull=Pin.PULL\_UP)

# Changing the LED flashing rate – using pushbutton interrupts

WOKWi SAVE SHARE Docs B

main.py • diagram.json •

```
1  from machine import Pin
2  from utime import sleep
3
4  LED = Pin(0, Pin.OUT)          # LED at pin GP0
5  Faster = Pin(1, Pin.IN, Pin.PULL_UP)      # Faster at pin GP1
6  Slower = Pin(2, Pin.IN, Pin.PULL_UP)      # Slower at pin GP2
7  dly = 1.0                      # Default delay
8
9  def Flash_Faster(Faster):
10     global dly
11     dly = dly - 0.1
12
13 def Flash_Slower(Slower):
14     global dly
15     dly = dly + 0.1
16
17 Faster.irq(handler=Flash_Faster,trigger=Faster.IRQ_FALLING)
18 Slower.irq(handler=Flash_Slower,trigger=Slower.IRQ_FALLING)
19
20 while True:
21     LED.value(1)                  # LED ON
22     sleep(dly)                   # Delay dly
23     LED.value(0)                  # LED OFF
24     sleep(dly)                   # Delay dly
25
```

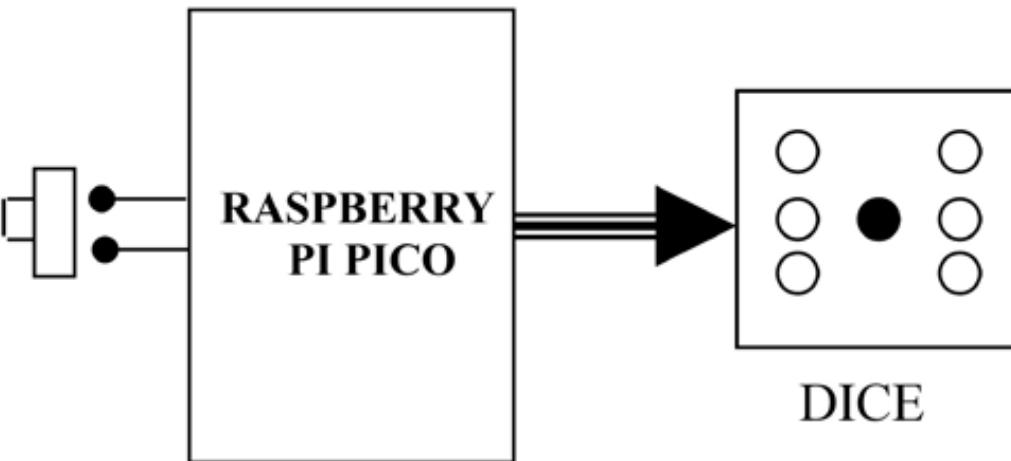
Simulation

Raspberry Pi Pico @ 2020

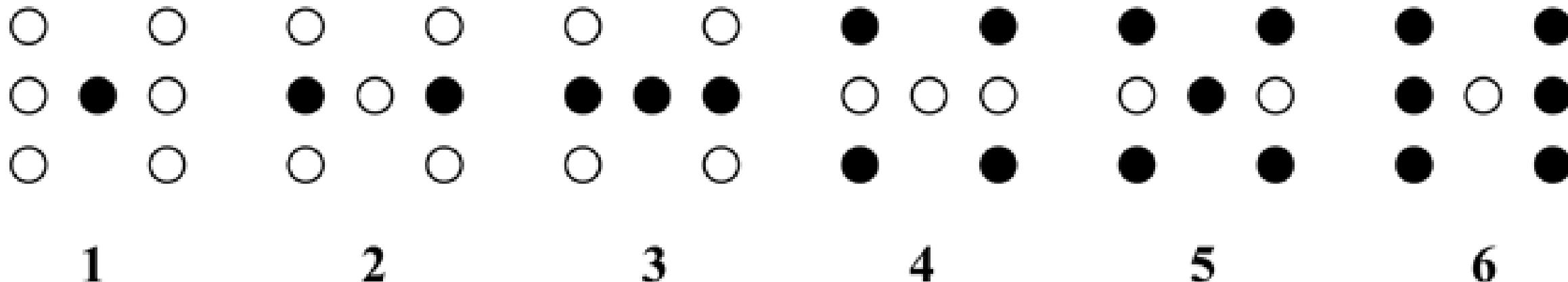
# Electronic Dice

## Objective:

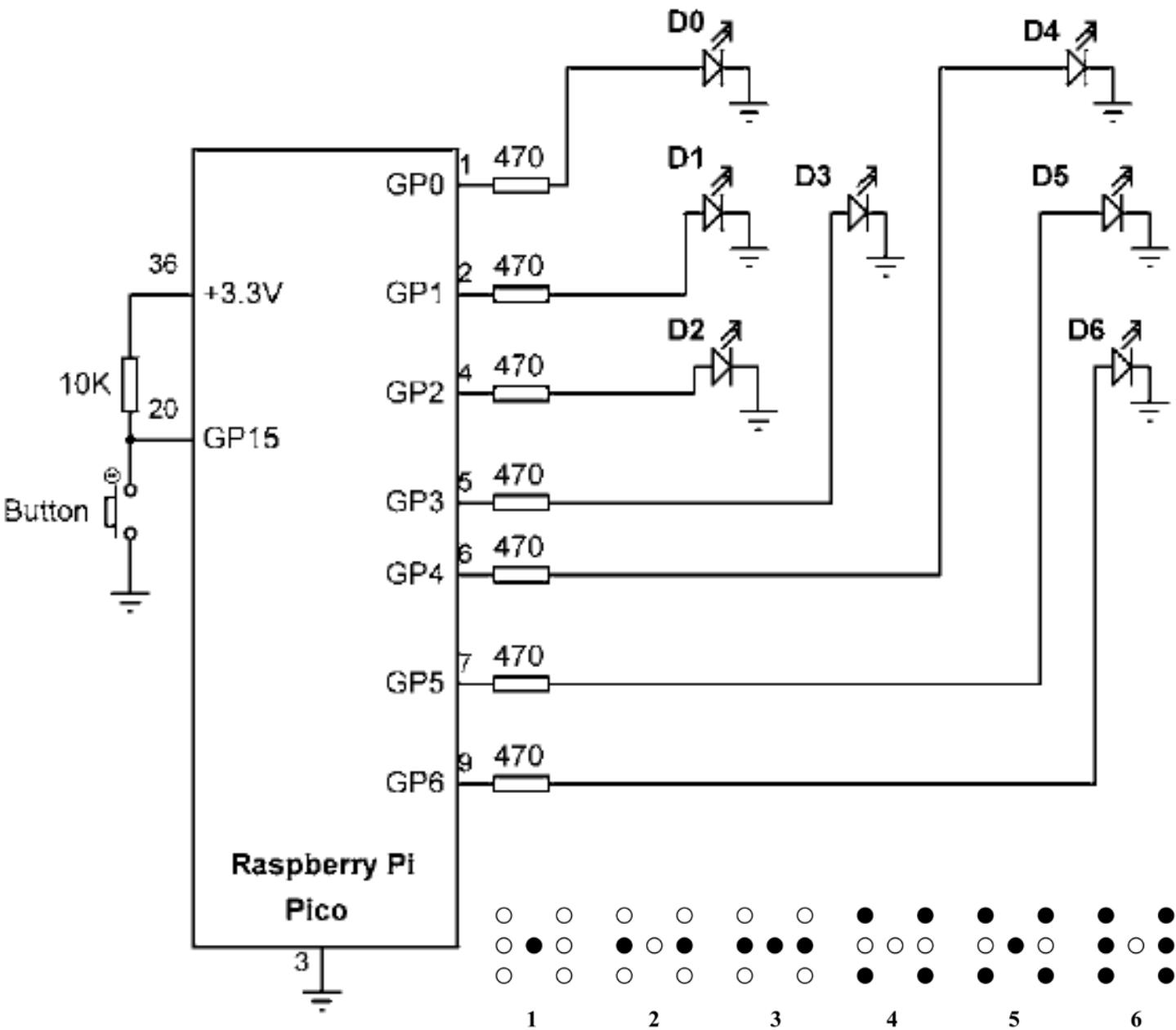
Push-button switch



7 LEDs are arranged in the form of the **faces of a dice** and a pushbutton switch is used. When the button is pressed, the LEDs turn ON to display numbers **1 to 6** as if on a real dice. The display is turned OFF after **3 seconds**, ready for the next game.



# Electronic Dice



# Electronic Dice

WOKWi SAVE SHARE Heart Pencil Docs B

main.py diagram.json PIO

```
1 from machine import Pin
2 from utime import sleep
3 import random
4
5 PORT = [7, 6, 5, 4, 3, 2, 1, 0]          # port connections
6 DICE_NO = [0, 0x08, 0x22, 0x2A, 0x55, 0x5D, 0x77]
7 L = [0]*8
8 Button = Pin(15, Pin.IN)
9
10 def Configure_Port():
11     for i in range(0, 8):
12         L[i] = Pin(PORT[i], Pin.OUT)
13
14 def Port_Output(x):
15     b = bin(x)
16     b = b.replace("0b", "")                  # convert into binary
17     diff = 8 - len(b)                      # remove leading "0b"
18     for i in range (0, diff):              # find the length
19         b = "0" + b                         # insert leading os
20
21     for i in range (0, 8):
22         if b[i] == "1":                    # insert leading os
23             L[i].value(1)
24         else:
25             L[i].value(0)
26
return
```

Simulation 00:03.199 99%

# Electronic Dice

WOKWi SAVE SHARE Docs B

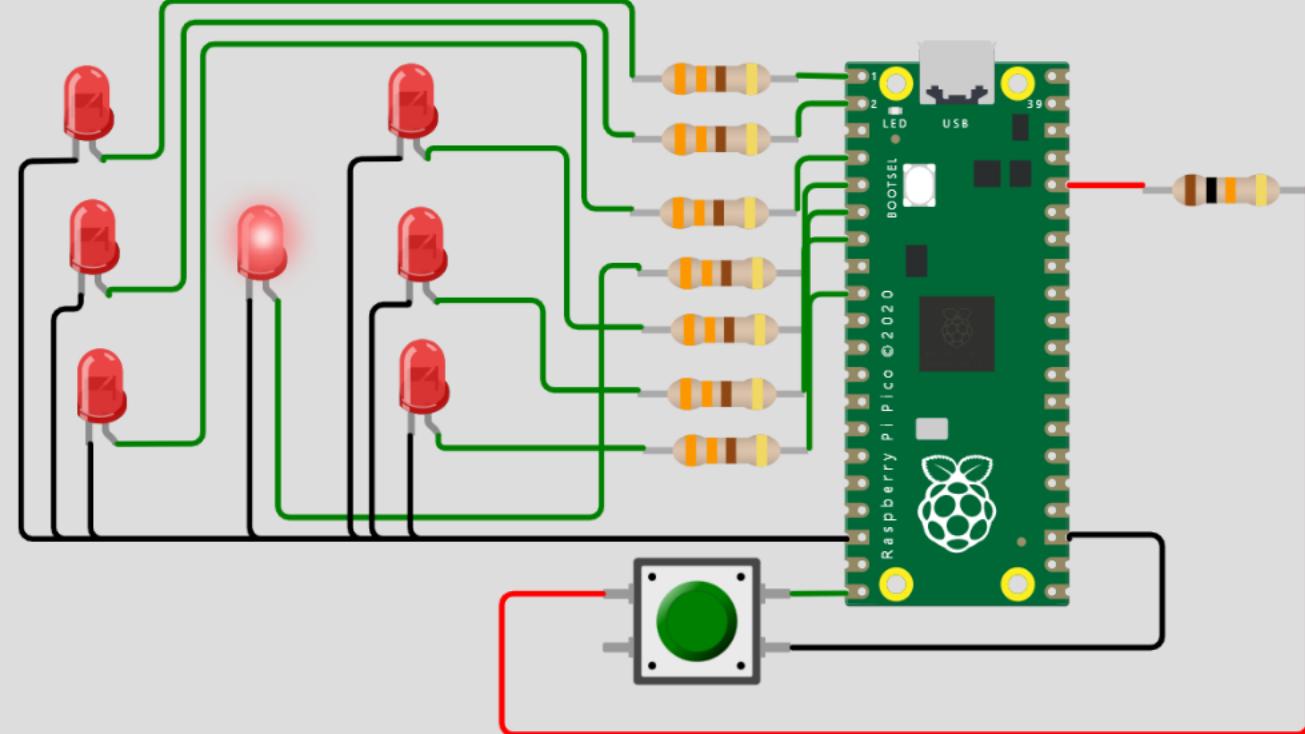
main.py diagram.json PIO 

```
27
28 def DICE():
29     n = random.randint(1, 6)
30     pattern = DICE_NO[n]
31     Port_Output(pattern)
32     sleep(3)
33     Port_Output(0)
34     return
35
36 Configure_Port()
37
38 while True:
39     if Button.value() == 0:
40         DICE()
41     pass
42
43
44
```

# generate a random number  
# find the pattern  
# turn ON required LEDs  
# wait for 3 seconds  
# turn OFF all LEDs

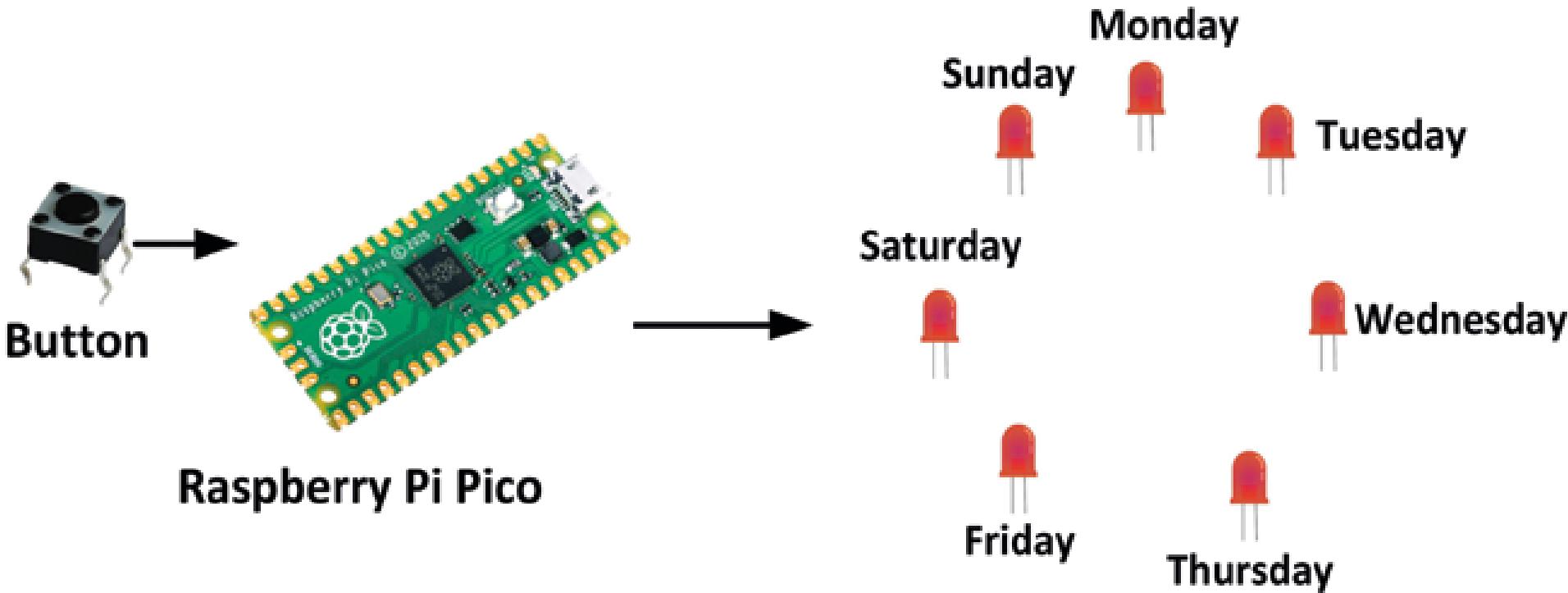
Simulation 

00:18.099 100%



# Lucky Day of the Week

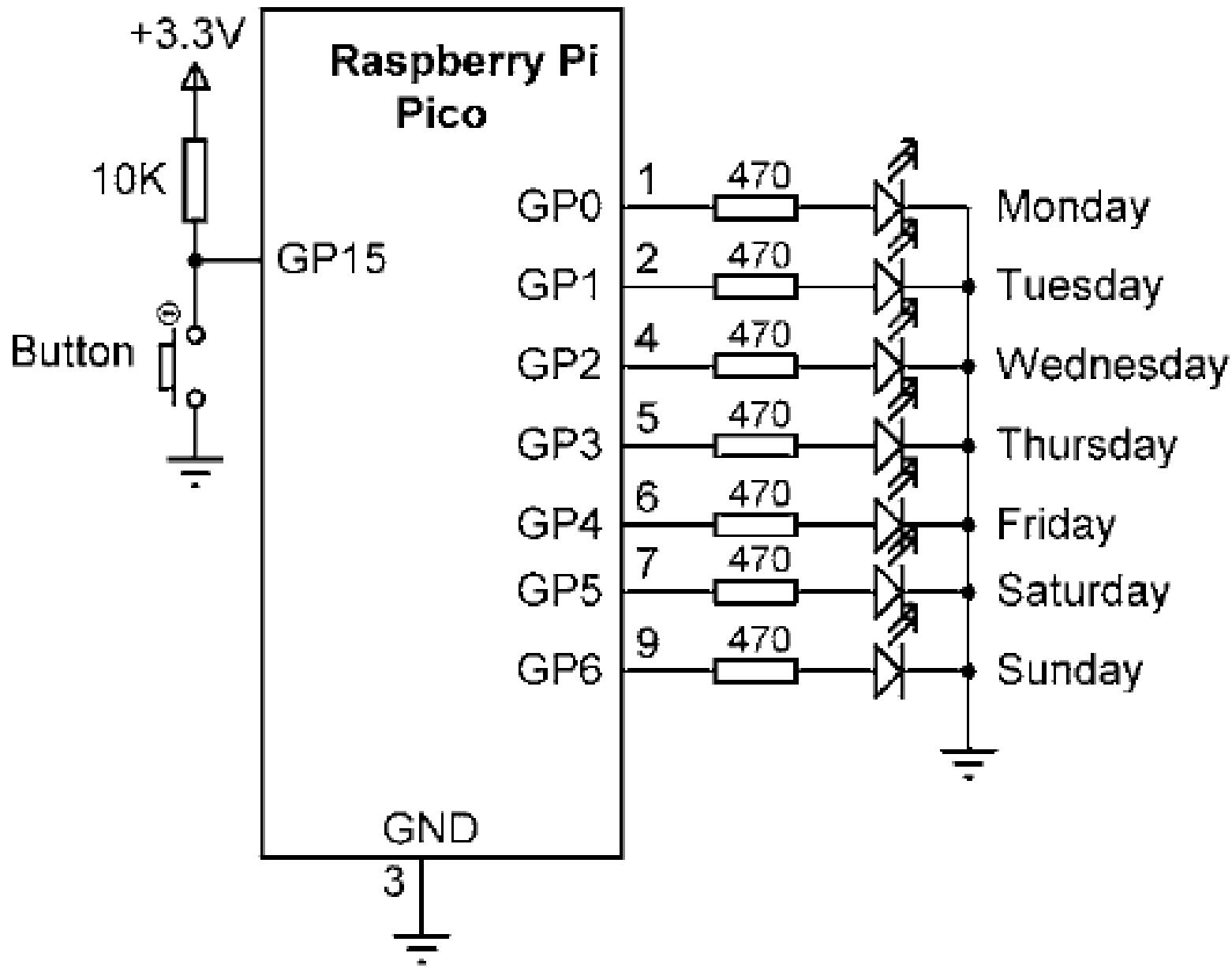
## Objective:



7 LEDs are positioned in the form of a circle. Each LED is assumed to represent a day of the week.

Pressing a button generates a random number between 1 and 7 and lights up only one of the LEDs. The day name corresponding to this LED is assumed to be your lucky day of the week.

# Lucky Day of the Week



# Lucky Day of the Week

WOKWi SAVE SHARE Docs B

main.py diagram.json ● PIO 100%

```
1 from machine import Pin
2 import utime
3 import random
4
5 PORT = [7, 6, 5, 4, 3, 2, 1, 0]          # port connections
6 DICE_NO = [0, 0x08, 0x22, 0x2A, 0x55, 0x5D, 0x77]
7 L = [0]*8
8 Button = Pin(15, Pin.IN)
9
10 def Configure_Port():
11     for i in range(0, 8):
12         L[i] = Pin(PORT[i], Pin.OUT)
13
14 def Port_Output(x):
15     b = bin(x)
16     b = b.replace("0b", "")                  # convert into binary
17     diff = 8 - len(b)                      # remove leading "0b"
18     for i in range (0, diff):              # find the length
19         b = "0" + b                         # insert leading 0s
20
21     for i in range (0, 8):
22         if b[i] == "1":                    # insert leading 0s
23             L[i].value(1)
24         else:
25             L[i].value(0)
26
return
```

Simulation

00:02.166 26%

# Lucky Day of the Week

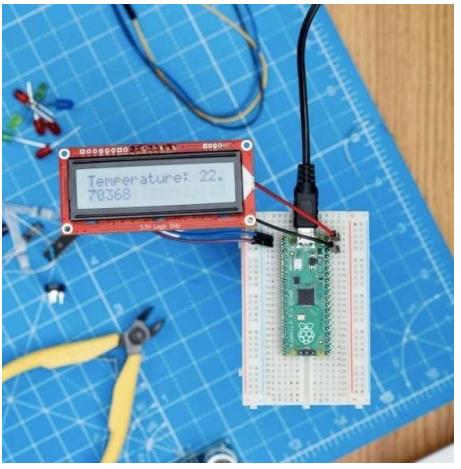
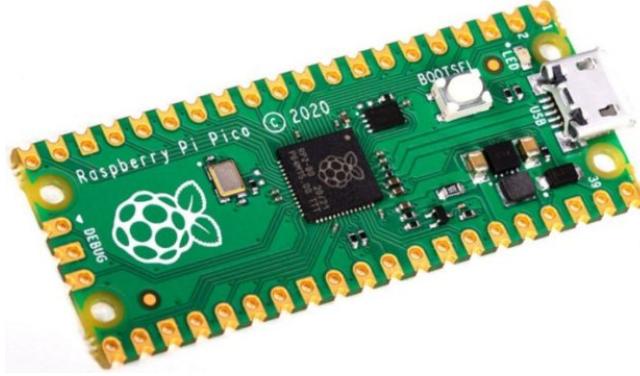
WOKWi SAVE SHARE Docs B

main.py diagram.json ● PIO 1

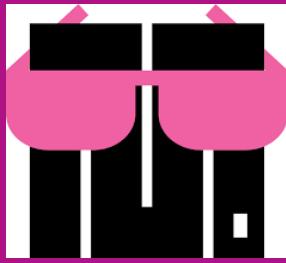
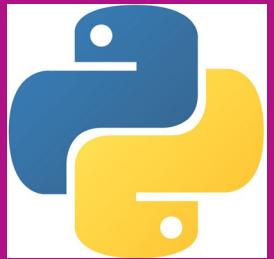
```
27
28 Configure_Port()
29
30 print("Press the Button to display your lucky number...")
31
32 random.seed(utime.ticks_ms())
33
34 while Button.value() == 1: # If Button not pressed
35     pass
36     r = random.randint(1, 7) # Generate random number
37     r = pow(2, r-1) # LED to be turned ON
38     Port_Output(r) # Send to LEDs
39
40
```

Simulation 00:08.300 27%

The circuit diagram shows a Raspberry Pi Pico connected to a breadboard. A green pushbutton is connected between Pin 13 (GND) and Pin 14 (Digital 0). A red LED is connected between Pin 14 (Digital 0) and Pin 15 (Digital 1). Another red LED is connected between Pin 15 (Digital 1) and Pin 16 (Digital 2). A third red LED is connected between Pin 16 (Digital 2) and Pin 17 (Digital 3). A fourth red LED is connected between Pin 17 (Digital 3) and Pin 18 (Digital 4). A fifth red LED is connected between Pin 18 (Digital 4) and Pin 19 (Digital 5). A sixth red LED is connected between Pin 19 (Digital 5) and Pin 20 (Digital 6). A seventh red LED is connected between Pin 20 (Digital 6) and Pin 21 (Digital 7). Each LED is connected in series with a resistor. The digital pins (14-21) are connected to the pins of a MAX7219 8x8 matrix driver chip, which is connected to the Raspberry Pi Pico. The Raspberry Pi Pico also has a USB connection and a BOOTSEL pin.

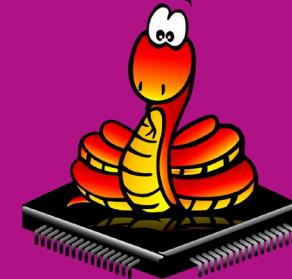


# INTERNET OF THINGS (IOT) PROJECTS USING PYTHON

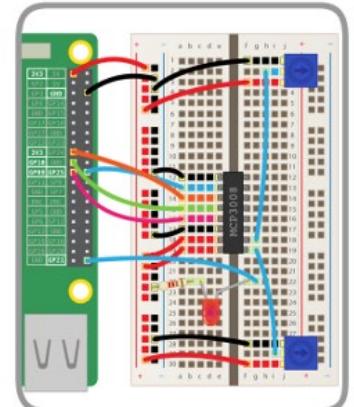
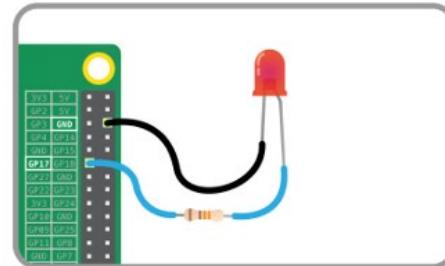
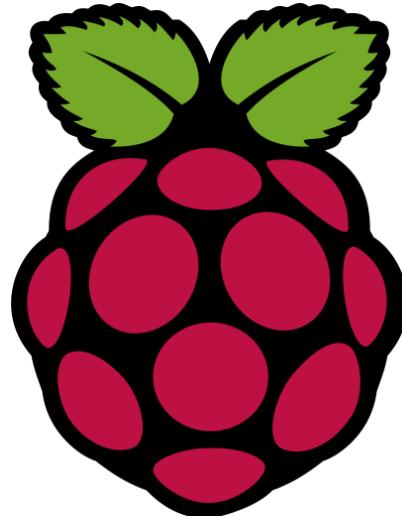


(CSE 4110)

(LECTURE – 6)



T<sub>h</sub>



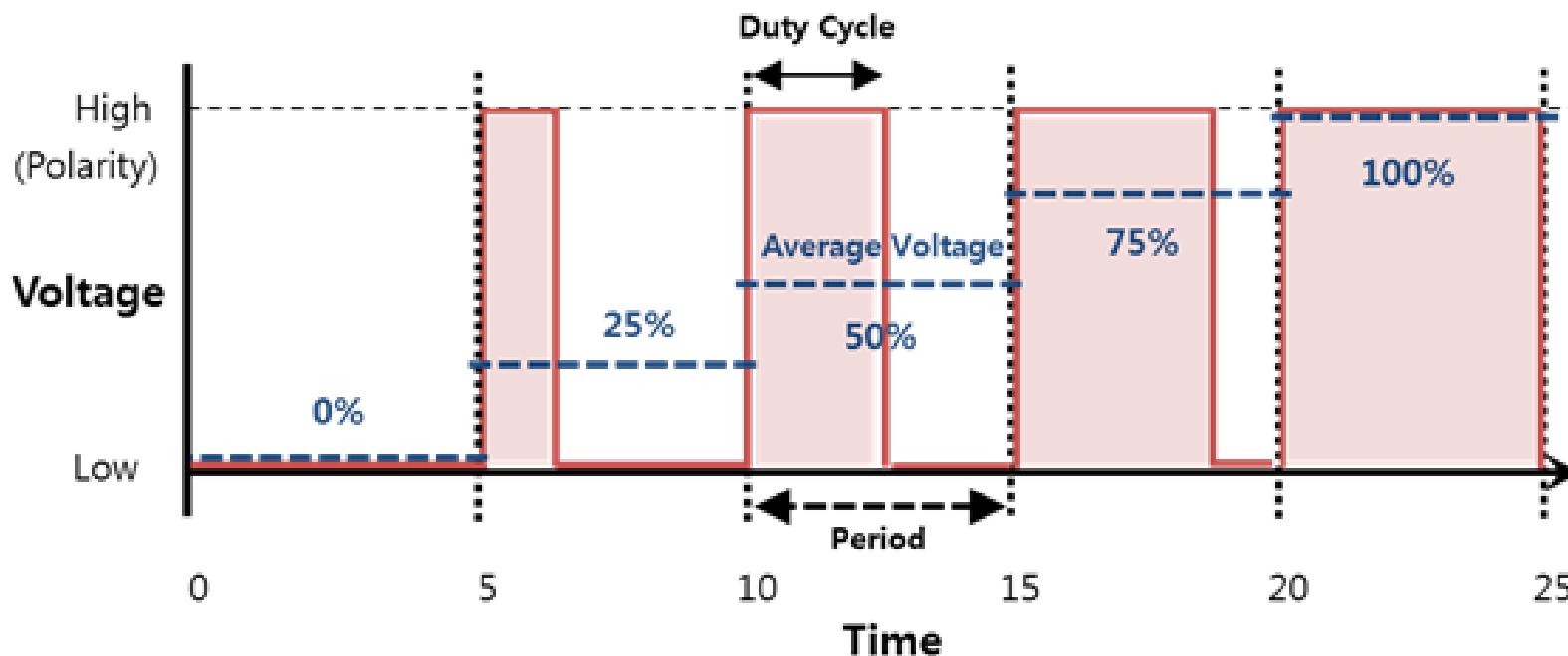
# Pulse With Modulation (PWM)

PWM, a trick to generate variable voltages on digital pins

The **PWM** is a technique that allows the generation of a voltage between 0 and 3.3V using **only digital outputs** .i.e, getting analog results with digital means.

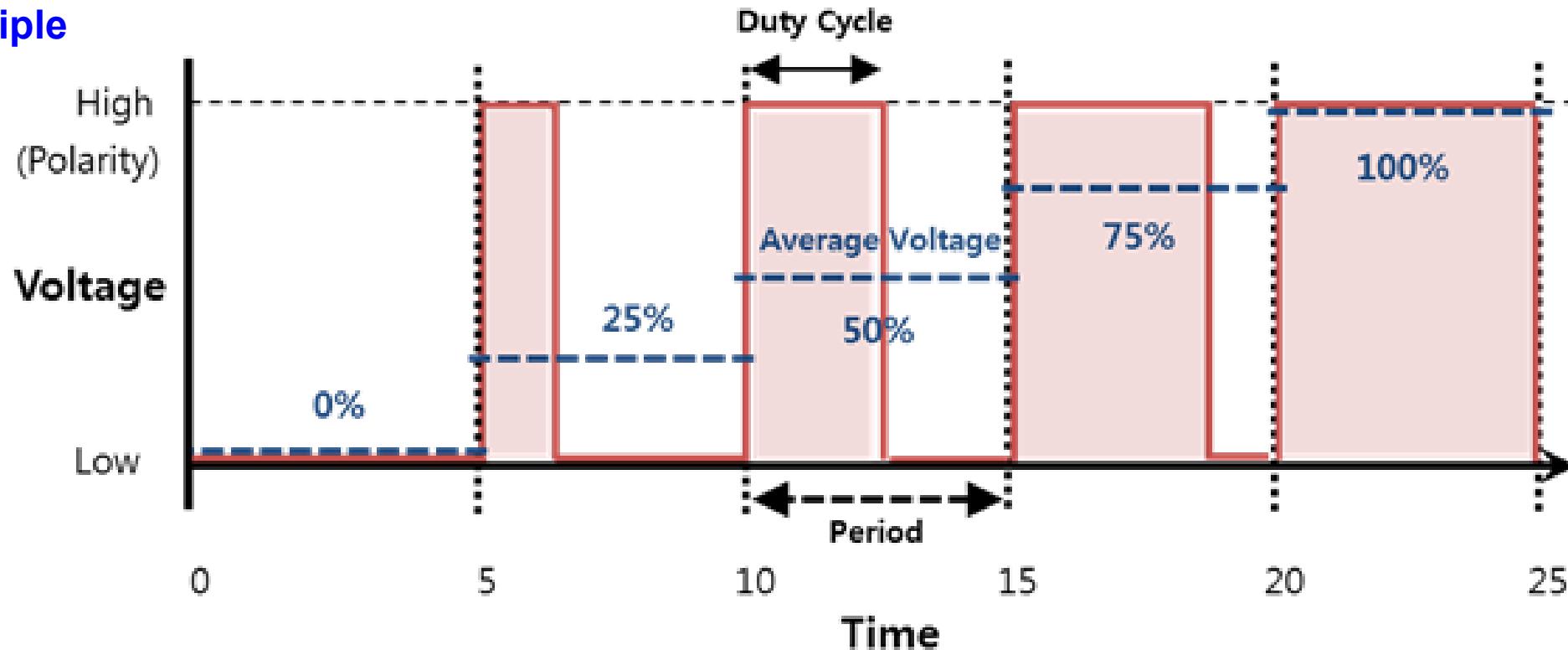
The PWM consists in varying the width of an electrical pulse.

The PWM allows the generation of constant voltages whose value can be changed. It does not generate alternating voltages as a DAC could do.



# Pulse With Modulation (PWM)

## PWM principle



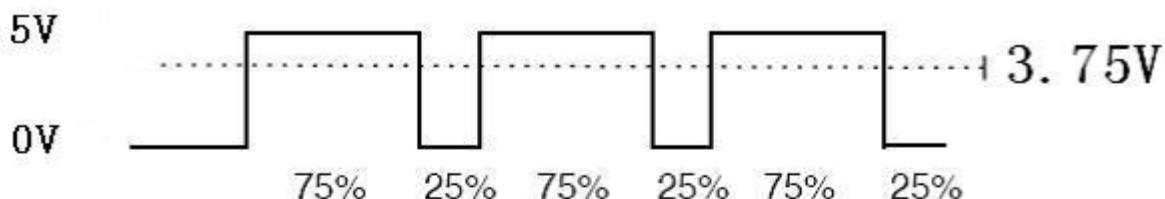
The succession of pulses with a given width is seen on average as a constant voltage between 0V and 3.3V, whose value is determined by :

$$V_{output} = V_{input} \times \alpha$$

with  $\alpha$  , the duty cycle (the pulse width in percent)

# Pulse With Modulation (PWM)

## PWM principle



The duration of “on time” is called the **pulse width**. To get varying analog values, you change or modulate, that width. If you repeat this on-off pattern fast enough with some device, an LED, for example, it would be like this: the signal is a steady voltage between 0 and 5V controlling the brightness of the LED.

The smaller the PWM value is, the smaller the value will be after being converted into voltage. Then the LED becomes dimmer accordingly. Therefore, we can control the brightness of the LED by controlling the PWM value.

# PWM Pins in Raspberry Pi Pico

## PWM Pins

All 30 GPIO pins on RP2040 can be used for PWM:

GPIO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PWM Channel	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B	7A	7B
GPIO	16	17	18	19	20	21	22	23	24	25	26	27	28	29		
PWM Channel	0A	0B	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B		

- The RP2040 microcontroller at the core of the Raspberry Pi Pico has **8 Slices** of PWM, and each Slice is **independent** of the others. This simply means that we can set the frequency of a Slice without affecting the others.
- The RP2040 PWM block has 8 identical PWM slices, each with two output channels (A/B), where the **B pin can also be used as an input for frequency and duty cycle measurement**. That means each slice can drive two PWM output signals, or measure the frequency or duty cycle of an input signal. This gives a **total of up to 16 controllable PWM outputs**. All 30 GPIO pins can be driven by the PWM block.

# PWM Pins in Raspberry Pi Pico

## PWM Pins

PWM_A[0]	GP0	1	1	VBUS
PWM_B[0]	GP1	2	2	VSYS
GND		3		GND
PWM_A[1]	GP2	4		3V3_EN
PWM_B[1]	GP3	5		3V3(OUT)
PWM_A[2]	GP4	6		ADC_VREF
PWM_B[2]	GP5	7		GP28
GND		8		PWM_A[6]
PWM_A[3]	GP6	9		33
PWM_B[3]	GP7	10		GND
PWM_A[4]	GP8	11		AGND
PWM_B[4]	GP9	12		32
GND		13		GP27
PWM_A[5]	GP10	14		PWM_B[5]
PWM_B[5]	GP11	15		31
PWM_A[6]	GP12	16		GP26
PWM_B[6]	GP13	17		PWM_A[5]
GND		18		30
PWM_A[7]	GP14	19		RUN
PWM_B[7]	GP15	20		29
				GP22
				PWM_A[3]
				28
				GND
				27
				GP21
				PWM_B[2]
				26
				GP20
				PWM_A[2]
				25
				GP19
				PWM_B[1]
				24
				GP18
				PWM_A[1]
				23
				GND
				22
				GP17
				PWM_B[0]
				21
				GP16
				PWM_A[0]

pins **21/GP16** and **22/GP17** belong to the same slice – GP16 is output A and GP17 is output B.



# Pulse With Modulation (PWM)

The following factors influence the behaviour of a pulse width modulated signal.

1) duty cycle

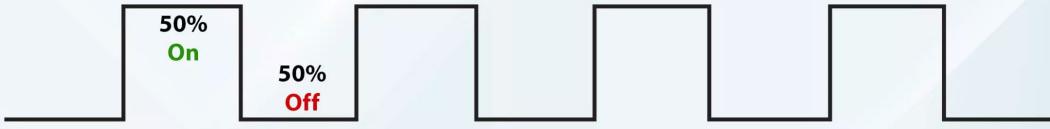
2) pulse frequency.

3) Resolution

We can modify these two parameters in MicroPython.

## Duty Cycle

### 50% Duty Cycle



### 75% Duty Cycle



### 25% Duty Cycle



## Frequency :

- Technically, frequency means “number of cycles per second”. When we toggle a digital signal (ON and OFF) at a high frequency, the result is an analog signal with a constant voltage level.
- PWM frequency is determined by the clock source.
- The frequency and resolution of PWM are inversely proportional.

## Resolution:

- A PWM signal's resolution refers to the number of steps it can take from zero to full power.
- The resolution of the PWM signal can be changed.
- The Raspberry Pi Pico module has a 1 - 16 bit resolution, which means we can set up to **65536** steps from zero to full power.

**Duty Cycle** : It is the ratio of **ON time** (when the signal is high) to the total time taken to complete the cycle.

# Pulse With Modulation (PWM)

## PWM Application

In practice, PWM is used to :

- Controlling the speed of a motor
- Direction or position control of a servo motor
- Controlling the brightness of LEDs
- Loudness control in buzzers
- Generate square signals (with  $\alpha=0.5$ )
- Generate music notes (sound similar to retro consoles)
- Controlling the fan speed

# Pulse With Modulation (PWM)

## PWM performance on the Raspberry Pi Pico

The microprocessor doesn't generate PWM signals permanently but uses dedicated hardware blocks. We only need to configure the PWM blocks once in the script to permanently create the signal in the background. We associate the output of a PWM block to a pin of our board. The processor resources will then be free to execute other tasks.

Each PWM block can have an independent frequency.

PWM characteristics	Pi Pico
Frequency range of the PWM signal	7 Hz to 125 Mhz
Independent PWM frequency	8
PWM Output	16
Duty cycle resolution	16 bits

In most cases, a PWM frequency of around **1000 Hz** will be sufficient.

# Pulse With Modulation (PWM)

## PWM on MicroPython with Pico

MicroPython automatically selects an available PWM block: it is unnecessary to indicate which one we intend to use.

The configuration consists in associating a **PWM** object to a **Pin** object and choosing the PWM frequency

Since the `PWM` object is also in the machine module, we can combine the two imports on a single line:

```
from machine import Pin  
from machine import PWM
```

is equivalent to:

```
from machine import Pin, PWM  
  
from machine import Pin, PWM  
  
pin = Pin(25, mode=Pin.OUT)  
pin_with_pwm = PWM(pin) # Attach PWM object on a pin
```

Once the pin of the board is linked to our **PWM** object, all the functions are done directly on the **PWM** object

# LED Breathing : Voltage Variation using Built-in LED

## PWM on MicroPython with Pico

In python, you can insert **underscores \_** to make the numbers easier to read (and thus avoid counting zeros on large numbers). For example, to write **1 MHz instead of having 1000000 , we can put 1\_000\_000** .

We use the function **.duty\_u16()** to select the duty cycle with a value between 0 and  $2^{16} - 1$  (**0-65535**). The voltage is set directly on the output of the previously selected pin

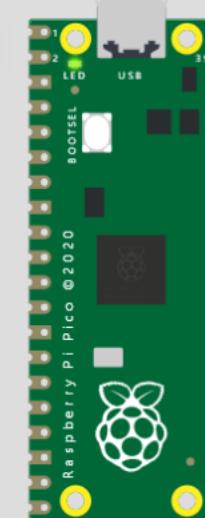
WOKWi SAVE SHARE Docs B

main.py • diagram.json • PIO 🐍

```
1 from machine import Pin, PWM
2
3 LED_BUILTLIN = 25 # For Raspberry Pi Pico
4
5 pwm_led = PWM(Pin(LED_BUILTLIN, mode=Pin.OUT)) # Attach PWM object on the LED pin
6
7 # Settings
8 pwm_led.freq(1_000)
9
10 while True:
11     for duty in range(0,65_536): # For Pi Pico
12         pwm_led.duty_u16(duty)
```

Simulation

00:21.133 21%



# LED Breathing : Duty Cycle Percentage Specification

## PWM on MicroPython with Pico

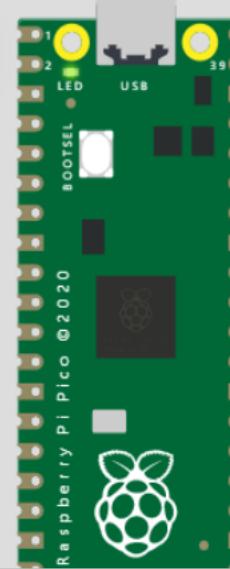
WOKWi SAVE SHARE Docs B

main.py • diagram.json • PIO

```
1 from machine import Pin, PWM
2
3 LED_BUILTIN = 25 # For Raspberry Pi Pico
4
5 pwm_led = PWM(Pin(LED_BUILTIN, mode=Pin.OUT)) # Attach PWM object on the LED pin
6 pwm_led.freq(1_000)
7
8 duty_cycle = 50 # Between 0 - 100 %
9 pwm_led.duty_u16(int((duty_cycle/100)*65_535))
```

Simulation

01:50.058 100%



MPY: soft reboot  
MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040  
Type "help()" for more information.  
>>> []

# LED Breathing : Brightness Modification

## PWM on MicroPython with Pico

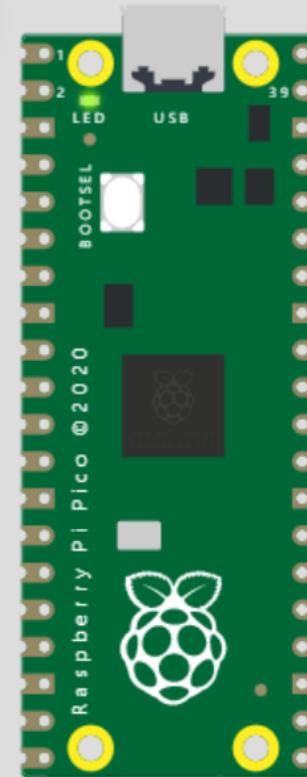
WOKWi SAVE SHARE Docs B

main.py • diagram.json • PIO

```
1 from machine import Pin, PWM
2 import time
3
4 LED_BUILTIN = 25 # For Raspberry Pi Pico
5
6 pwm_led = PWM(Pin(LED_BUILTIN, mode=Pin.OUT)) # Attach PWM object on the LED pin
7 pwm_led.freq(1_000)
8
9 while True:
10     for duty in range(101): # Duty from 0 to 100 %
11         pwm_led.duty_u16(int((duty/100)*65_535))
12         time.sleep_ms(10)
```

Simulation

01:53.652 99%



# LED Breathing : Dimming the integrated inbuilt LED

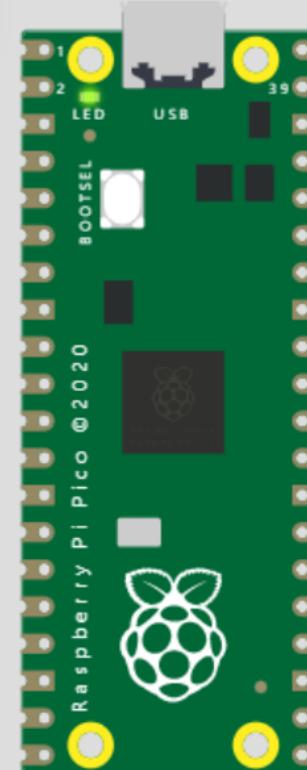
WOKWi SAVE SHARE Docs B

main.py • diagram.json • PIO

```
1 from machine import Pin, PWM
2
3 LED_BUILTLIN = 25 # For Raspberry Pi Pico
4
5 pwm_led = PWM(Pin(LED_BUILTLIN, mode=Pin.OUT)) # Attach PWM object on the LED pin
6
7 # Settings
8 pwm_led.freq(1_000)
9
10 while True:
11     for duty in range(0,65_536, 5):
12         pwm_led.duty_u16(duty)
13     for duty in range(65_536,0, -5):
14         pwm_led.duty_u16(duty)
```

Simulation

00:03.100 17%



# LED Breathing : Dimming the External LED

WOKWi SAVE SHARE Docs B

main.py • diagram.json • PIO

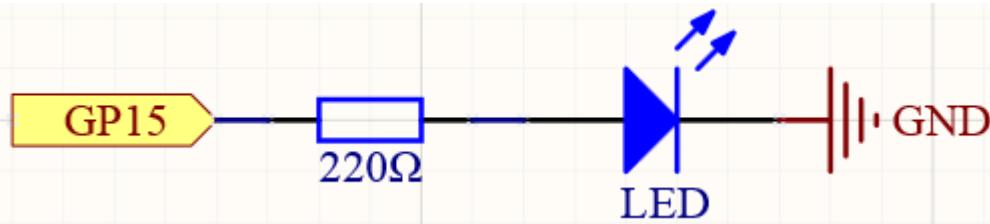
```
1 from machine import Pin, PWM
2 import utime
3
4 led = machine.PWM(machine.Pin(15))
5 led.freq(1000)
6
7 for brightness in range(0,65535,50):
8     led.duty_u16(brightness)
9     utime.sleep_ms(10)
10    led.duty_u16(0)
11
```

Simulation

00:03.516 100%

Raspberry Pi Pico © 2020

# LED Breathing : Dimming the External LED

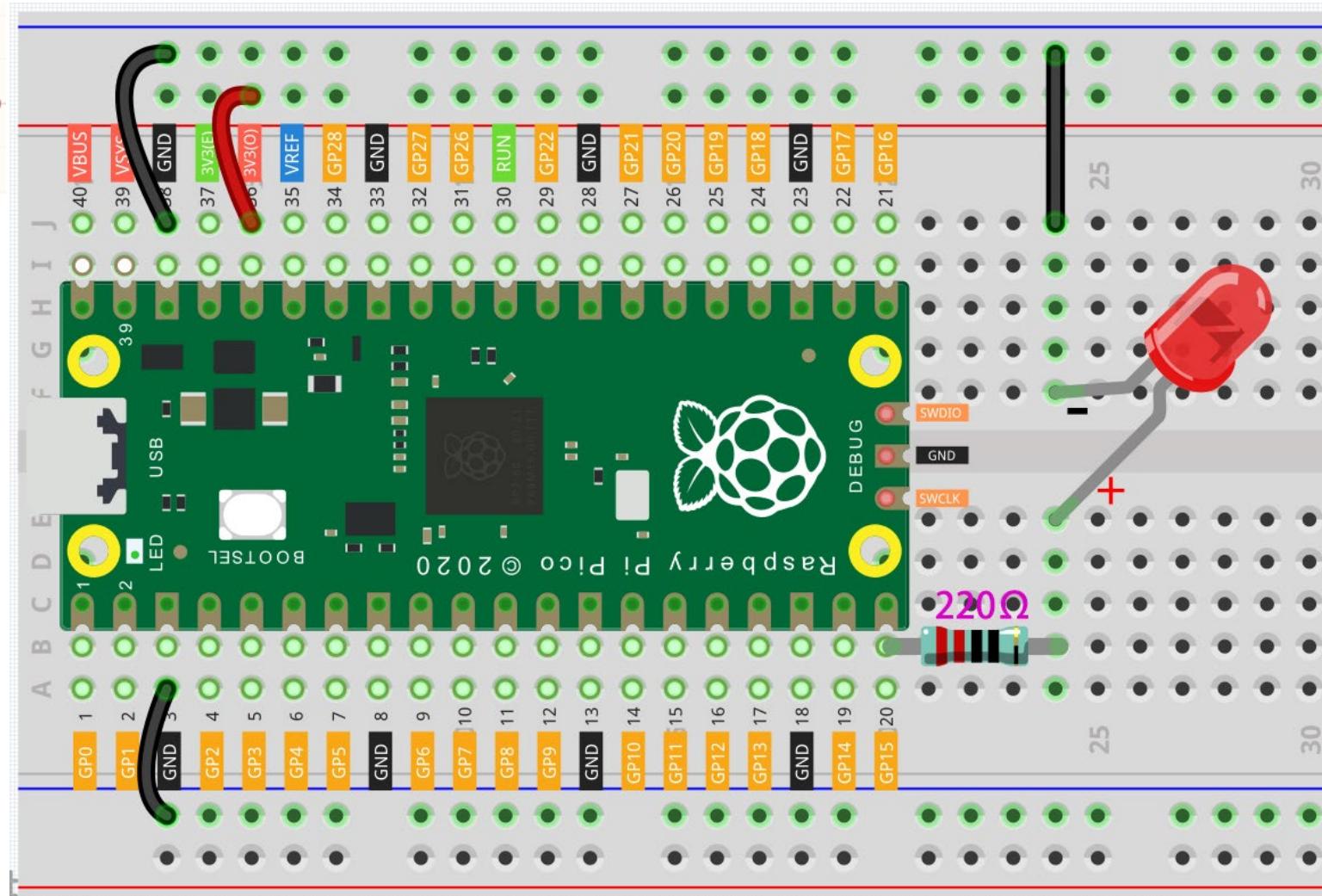


```
from machine import Pin, PWM  
import utime  
  
led = machine.PWM(machine.Pin(15))  
led.freq(1000)  
  
for brightness in range(0,65535,50):  
    led.duty_u16(brightness)  
    utime.sleep_ms(10)  
led.duty_u16(0)
```

duty cycle is 0%

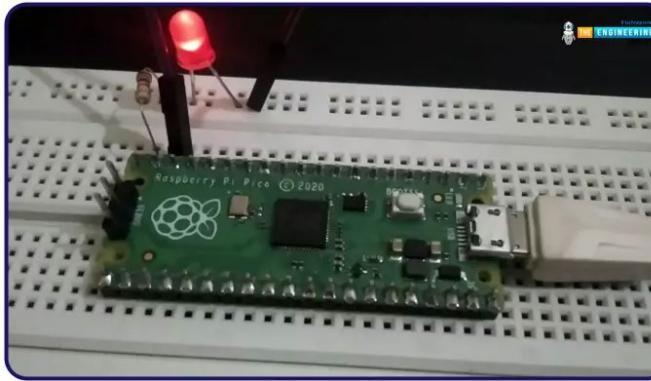


- 1) Change the duty cycle to 100%
- 2) Change the duty cycle to 50%



# LED Breathing : Up and Down Fading the External LED

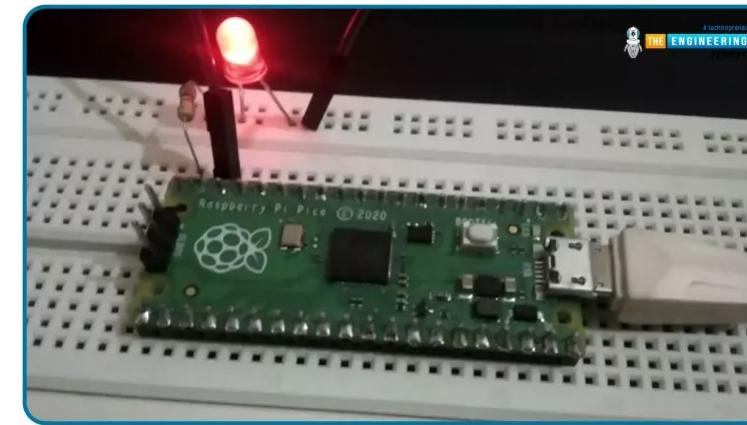
```
from machine import Pin, PWM  
import time  
  
led = PWM(Pin(14))  
  
print(led)  
  
led.freq(1000)  
  
while True:  
    for duty in range(0, 65535):  
        led.duty_u16(duty)  
        print(duty)  
        time.sleep(0.001)  
    for duty in range(65535, 1):  
        led.duty_u16(duty)  
        print(duty)  
        time.sleep(0.001)
```



**Brightness level 1**

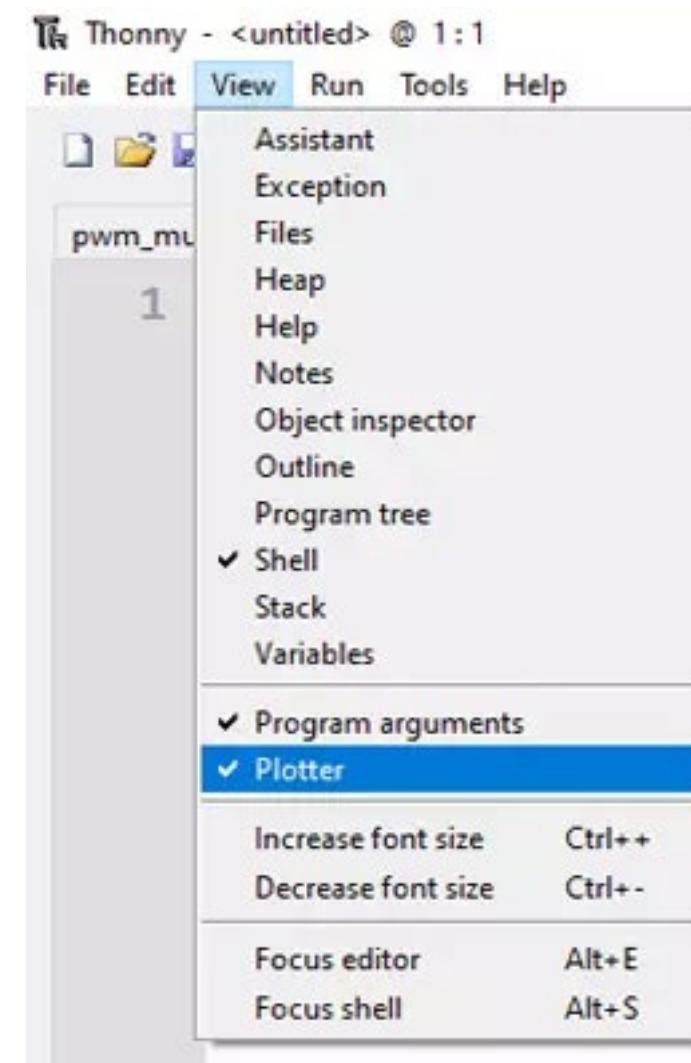


**Brightness level 2**

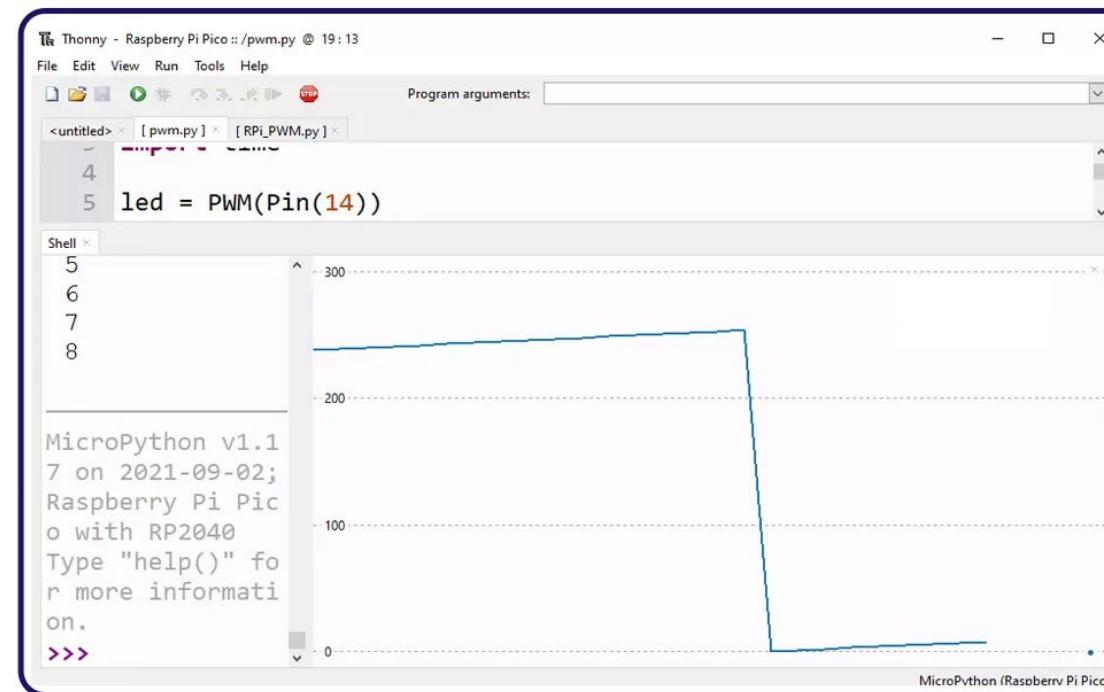


**Brightness level 3**

# LED Breathing : Plotter in Thonny

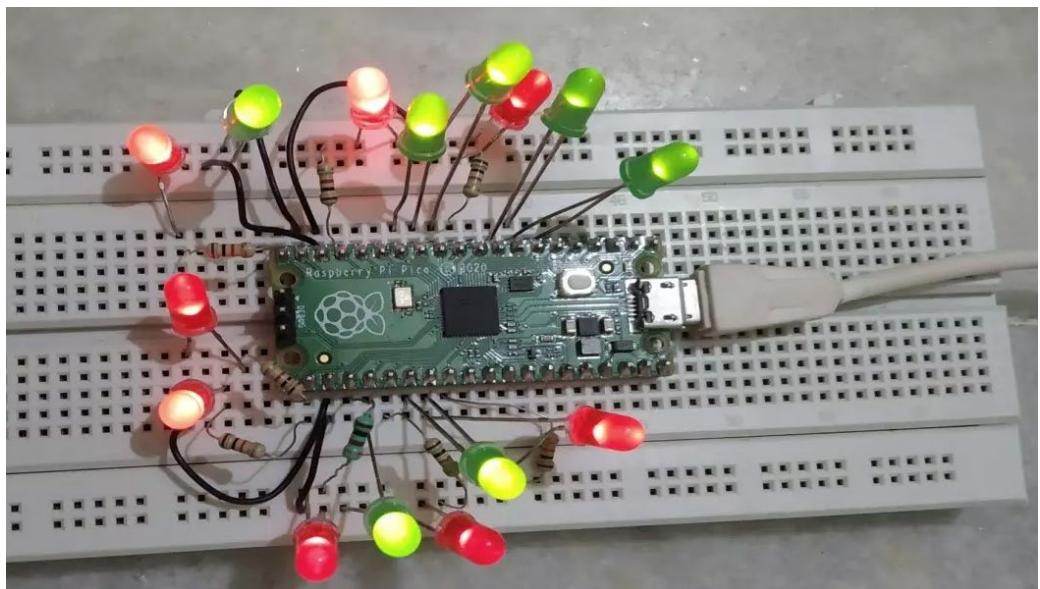
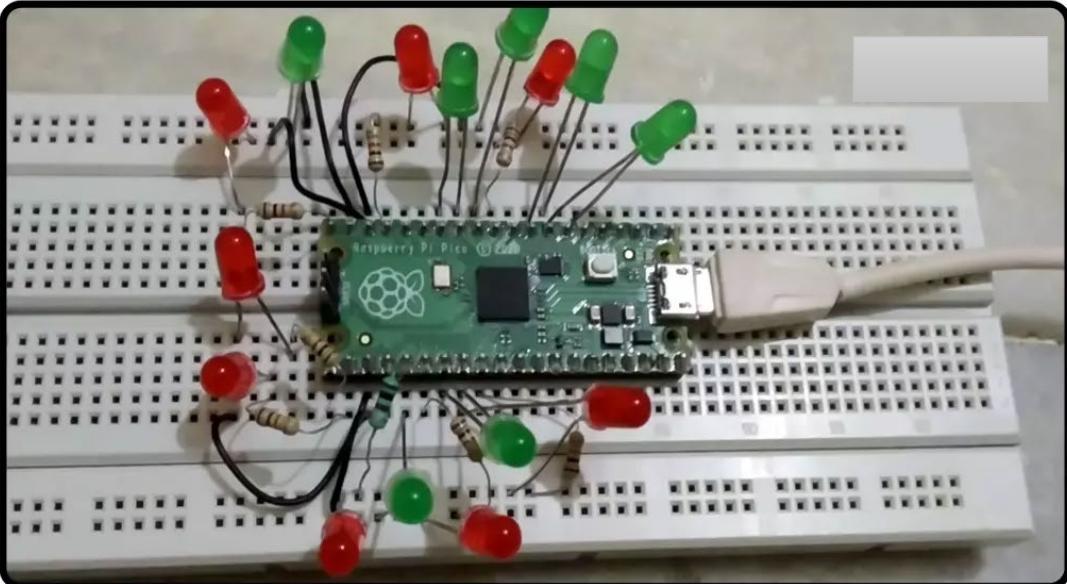


Rising PWM output 0 to 65535  
(brightness increasing)

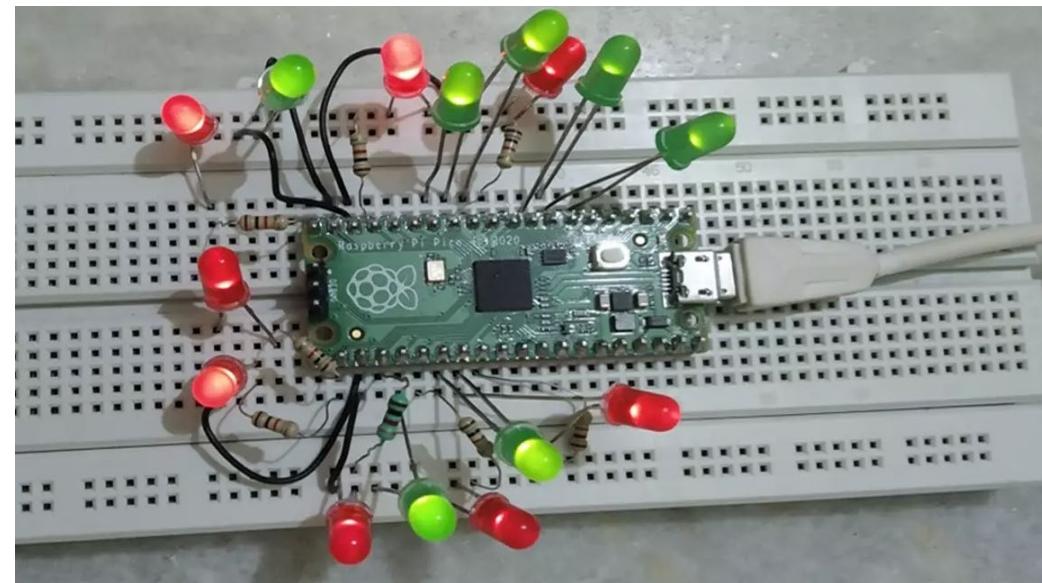


PWM output maximum to 0

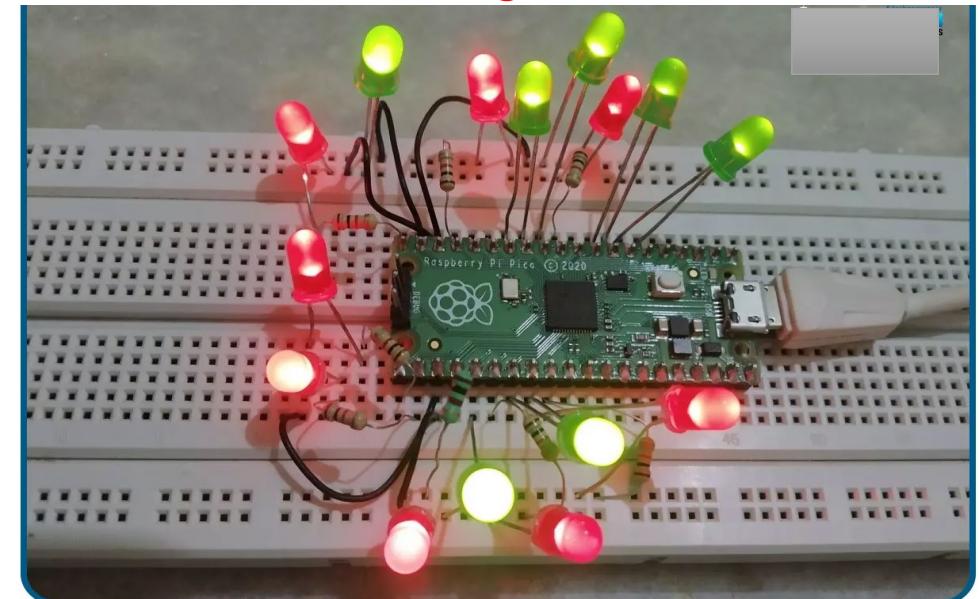
# LED Breathing : Complete the Challenge



Intermediate Brightness Level



Minimum Brightness Level



Maximum Brightness Level

# LED Breathing : Complete the Challenge

```
from machine import Pin, PWM
import time

led_1 = PWM(Pin(5))      # declaring led_x object for PWM pins
led_2 = PWM(Pin(6))
led_3 = PWM(Pin(8))
led_4 = PWM(Pin(9))
led_5 = PWM(Pin(10))
led_6 = PWM(Pin(13))
led_7 = PWM(Pin(14))
led_8 = PWM(Pin(15))
led_9 = PWM(Pin(16))
led_10 = PWM(Pin(17))
led_11 = PWM(Pin(18))
led_12 = PWM(Pin(19))
led_13 = PWM(Pin(20))
led_14 = PWM(Pin(21))
led_15 = PWM(Pin(22))
led_16 = PWM(Pin(26))

print(led_1)

def led_freq(x):          # Generating frequency for all LEDs
    led_1.freq(x)
    led_2.freq(x)
    led_3.freq(x)
    led_4.freq(x)
    led_5.freq(x)
    led_6.freq(x)
    led_7.freq(x)
    led_8.freq(x)
    led_9.freq(x)
    led_10.freq(x)
    led_11.freq(x)
    led_12.freq(x)
    led_13.freq(x)
    led_14.freq(x)
    led_15.freq(x)
    led_16.freq(x)

    led_freq(1000)        # setting pulse width modulation frequency
```

**Try to complete the circuit connection with the help of code**

# LED Breathing : Complete the Challenge

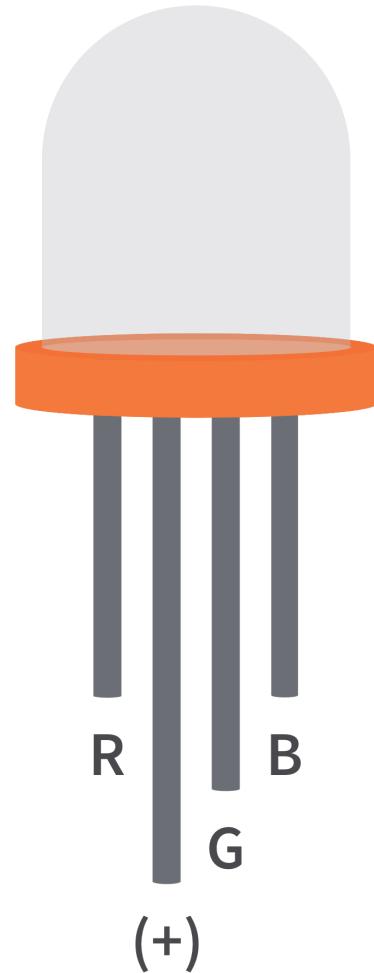
```
while True:  
    for duty in range(0, 65535):      # Increasing LED broghtness  
        led_1.duty_u16(duty)  
        led_2.duty_u16(duty)  
        led_3.duty_u16(duty)  
        led_4.duty_u16(duty)  
        led_5.duty_u16(duty)  
        led_6.duty_u16(duty)  
        led_7.duty_u16(duty)  
        led_8.duty_u16(duty)  
        led_9.duty_u16(duty)  
        led_10.duty_u16(duty)  
        led_11.duty_u16(duty)  
        led_12.duty_u16(duty)  
        led_13.duty_u16(duty)  
        led_14.duty_u16(duty)  
        led_15.duty_u16(duty)  
        led_16.duty_u16(duty)  
  
        print(duty)                  # Print the duty Cycle  
        time.sleep(0.001)
```

```
for duty in range(65535, 0):      # deccresing LED brightness  
    led_1.duty_u16(duty)  
    led_2.duty_u16(duty)  
    led_3.duty_u16(duty)  
    led_4.duty_u16(duty)  
    led_5.duty_u16(duty)  
    led_6.duty_u16(duty)  
    led_7.duty_u16(duty)  
    led_8.duty_u16(duty)  
    led_9.duty_u16(duty)  
    led_10.duty_u16(duty)  
    led_11.duty_u16(duty)  
    led_12.duty_u16(duty)  
    led_13.duty_u16(duty)  
    led_14.duty_u16(duty)  
    led_15.duty_u16(duty)  
    led_16.duty_u16(duty)  
  
    print(duty)  
    time.sleep(0.001)
```

**Try to complete the circuit connection with the help of code**

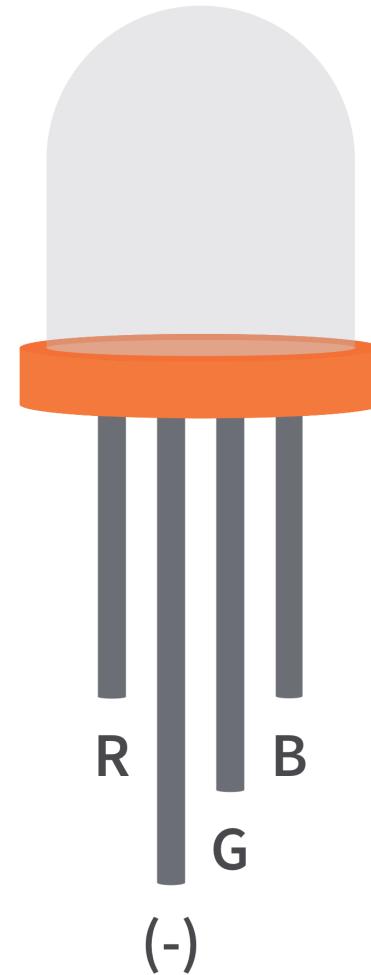
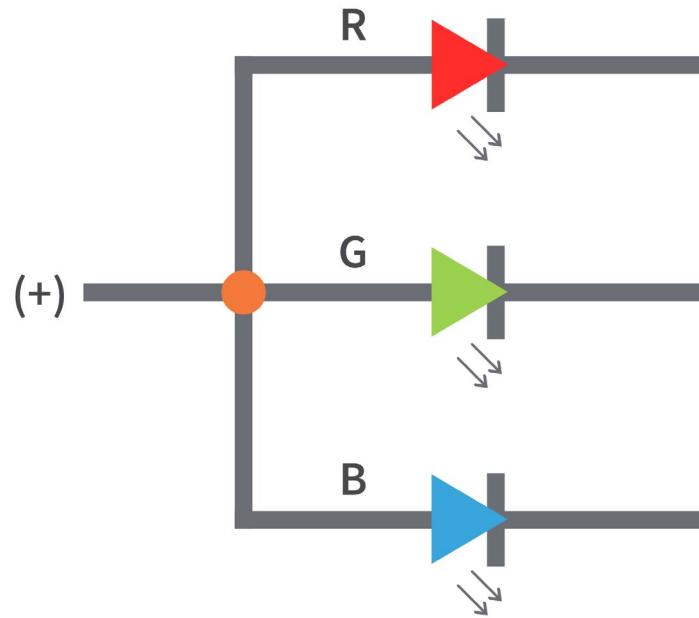
# Colorful Light: How RGB LEDs work

## RGB LED Types and Structure



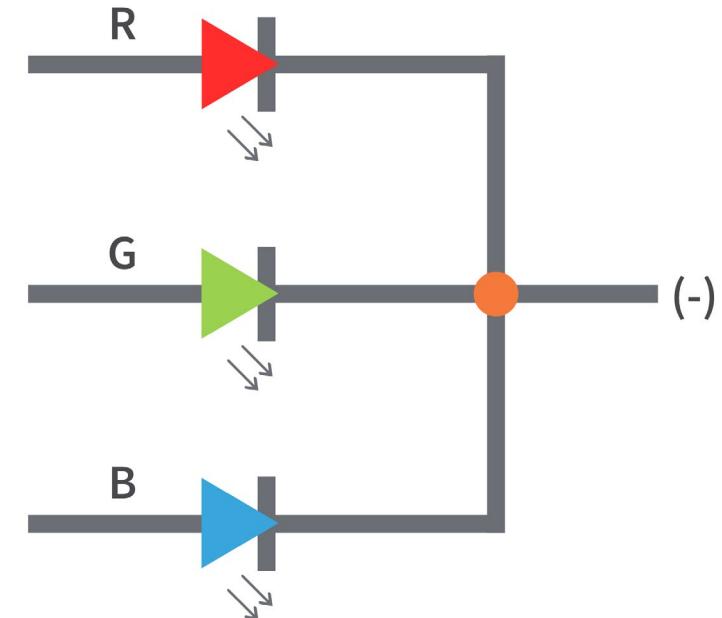
Common Anode

**Common Anode RGB LED**



Common Cathode

**Common Cathode RGB LED**



# Colorful Light: Light can be superimposed

- 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”.
- 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**.
- **COMMON CATHODE 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.

# Colorful Light: Alternately flashing red, green, and blue LEDs — RGB

WOKWI SAVE SHARE Docs B

main.py • diagram.json • PIO 🐍

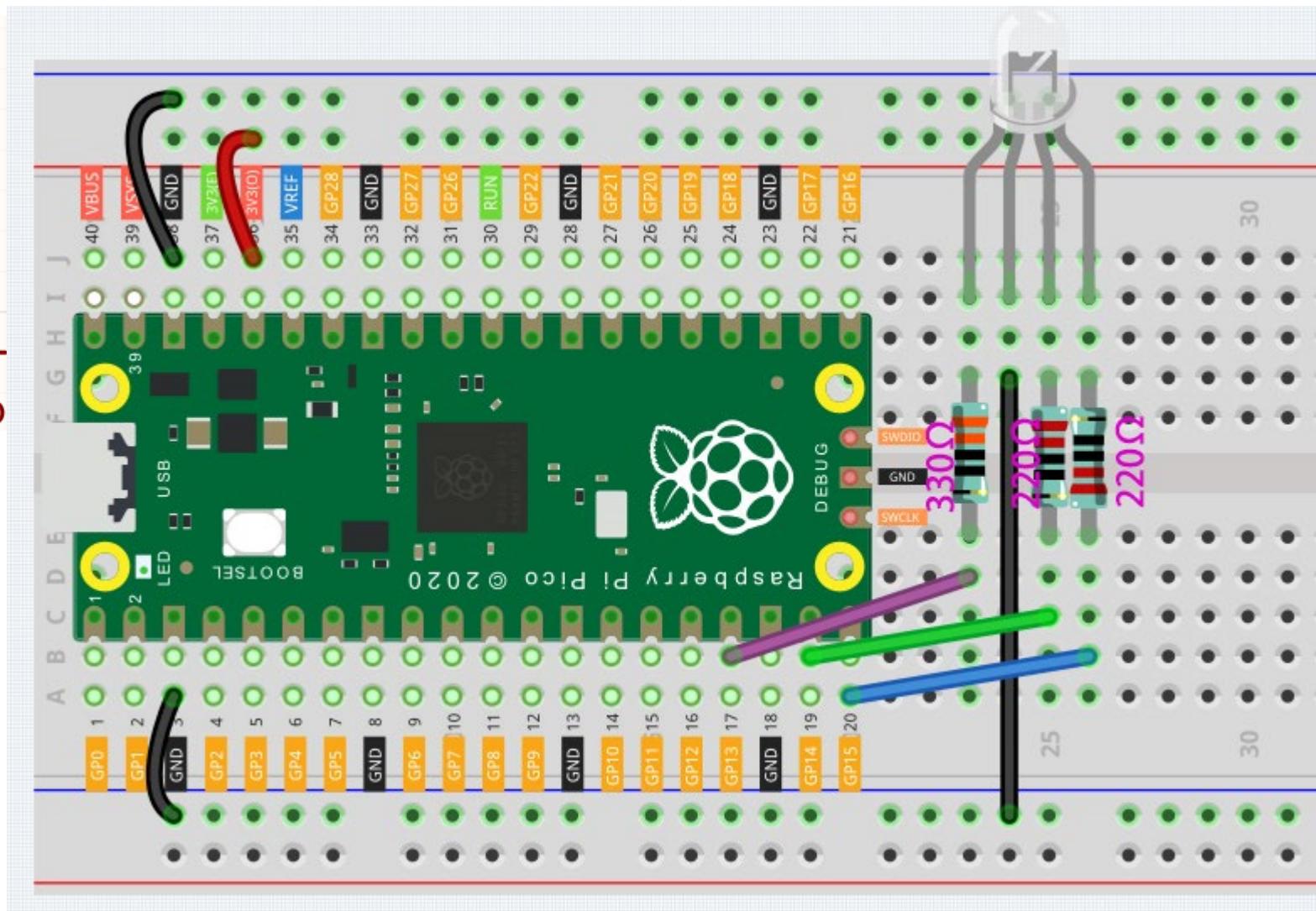
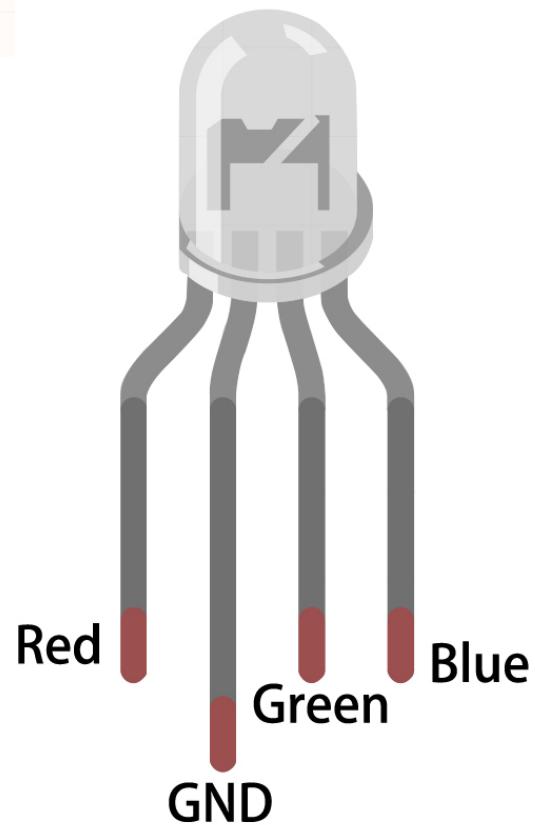
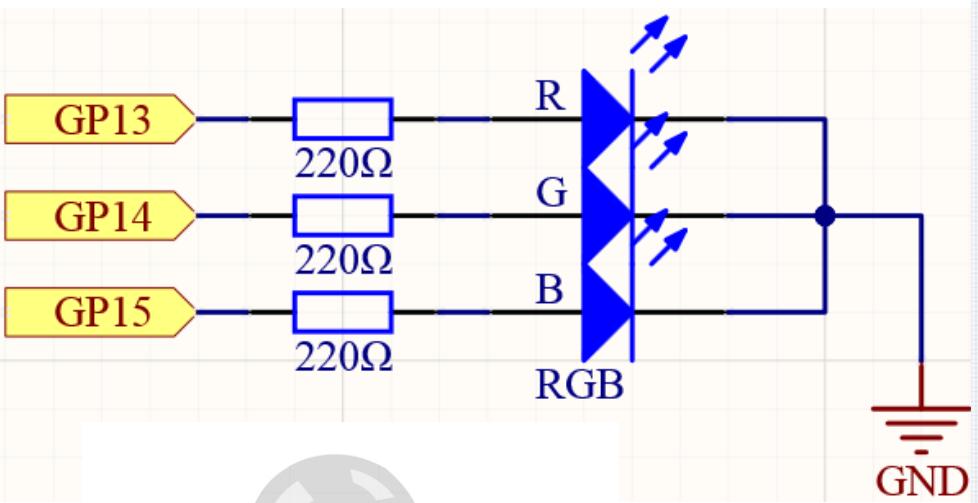
```
1  from machine import Pin
2  from utime import sleep
3
4  red = Pin(13, Pin.OUT)
5  green = Pin(14, Pin.OUT)
6  blue = Pin(15, Pin.OUT)
7
8  #Clear common cathode RGB LED
9
10 red.value(0)
11 green.value(0)
12 blue.value(0)
13
14 while True:
15     red.toggle()
16     sleep(1)
17     red.toggle()
18
19     green.toggle()
20     sleep(1)
21     green.toggle()
22
23     blue.toggle()
24     sleep(1)
25     blue.toggle()
26
27     sleep(1)
28
```

Simulation

Raspberry Pi Pico © 2020

00:26.763 100%

# Colorful Light: Light can be superimposed



# Colorful Light: Mystery of additive color mixing

WOKWI Docs

main.py diagram.json **rgb.py** ● PIO

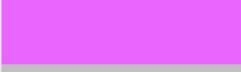
```
1  from machine import Pin, PWM
2  from utime import sleep
3
4  red = PWM(Pin(13))
5  green = PWM(Pin(14))
6  blue = PWM(Pin(15))
7
8  red.freq(1000)
9  green.freq(1000)
10 blue.freq(1000)
11
12 def interval_mapping(x, in_min, in_max, out_min, out_max):
13     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
14
15 def color_to_duty(rgb_value):
16     rgb_value = int(interval_mapping(rgb_value,0,255,0,65535))
17     return rgb_value
18
19 def color_set(red_value,green_value,blue_value):
20     red.duty_u16(color_to_duty(red_value))
21     green.duty_u16(color_to_duty(green_value))
22     blue.duty_u16(color_to_duty(blue_value))
23
24 color_set(255,128,0)
```

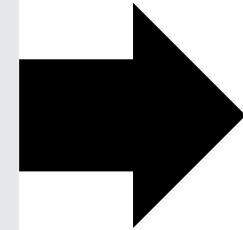
Change these values in THONNY and see the magic

Simulation

00:10.048 99%

# Colorful Light: Mystery of additive color mixing

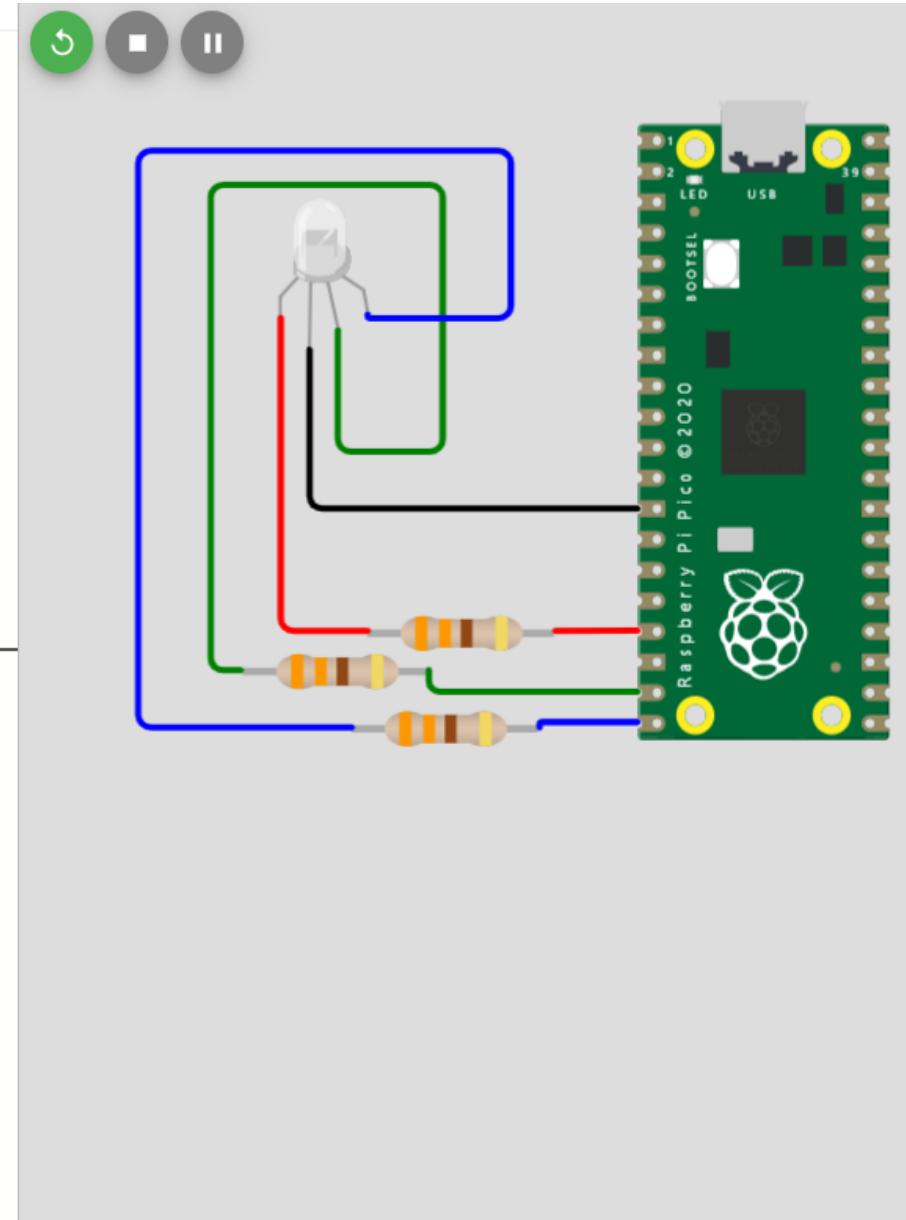
Color	HTML/CSS NAME	Decimal (R,G,B)
	Black	(0,0,0)
	White	(255,255,255)
	Red	(255,0,0)
	Lime	(0,255,0)
	Blue	(0,0,255)
	Yellow	(255,255,0)
	Cyan	(0,255,255)
	Magenta	(255,0,255)
	Silver	(192,192,192)
	Grey	(128,128,128)
	Maroon	(128,0,0)
	Olive	(128,128,0)
	Green	(0,128,0)
	Purple	(128,0,128)
	Teal	(0,128,128)
	Navy	(0,0,128)



Check these colours in your  
**RGB LED**

# Colorful Light: Mystery of additive color mixing

```
1 from machine import Pin, PWM
2 from utime import sleep
3
4 red = PWM(Pin(13))
5 green = PWM(Pin(14))
6 blue = PWM(Pin(15))
7
8 red.freq(1000)
9 green.freq(1000)
10 blue.freq(1000)
11
12 def interval_mapping(x, in_min, in_max, out_min, out_max):
13     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
14
15 def color_to_duty(rgb_value):
16     rgb_value = int(interval_mapping(rgb_value,0,255,0,65535))
17     return rgb_value
18
19 def color_set(red_value,green_value,blue_value): #let the three primary colors work together.
20     red.duty_u16(color_to_duty(red_value))
21     green.duty_u16(color_to_duty(green_value))
22     blue.duty_u16(color_to_duty(blue_value))
23
24 while True:
25     color_set(255,0,0)
26     sleep(1)
27     color_set(0,255,0)
28     sleep(1)
29     color_set(0,0,255)
30     sleep(1)
31     color_set(0,255,255)
32     sleep(1)
33     color_set(255,255,0)
34     sleep(1)
35     color_set(255,255,255)
36     sleep(1)
37     color_set(128,0,0)
38     sleep(1)
39     color_set(128,128,0)
40     sleep(1)
```



# Custom Tone: Create a Melody

## Buzzers

- Electronic buzzers create sound.
- Buzzers can be categorized into two different types – active buzzers and passive buzzers.
- An active buzzer has a built-in oscillator so it can produce sound with only a DC power supply.
- A passive buzzer does not have a built-in oscillator, so it needs an AC audio signal to produce sound.
- An active buzzer has a built-in oscillator so it can produce sound with only a DC power supply.
- 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

## Active Buzzers Vs Passive Buzzers

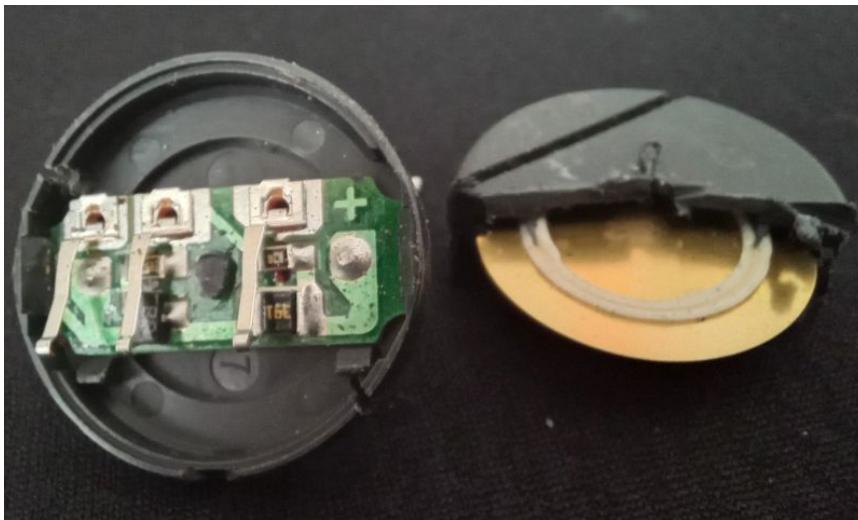
### ACTIVE BUZZERS

Active buzzers are the simplest to use. They are typically available in voltages from **1.5V to 24V**. All you need to do is apply a DC voltage to the pins and it will make a sound.

Active buzzers have **polarity**. The polarity is the same as an LED and a capacitor – the longer pin goes to positive. One **downside** of active buzzers is that the **frequency of the sound is fixed** and **cannot be adjusted**.

# Custom Tone: Create a Melody

## Inside an ACTIVE BUZZERS



The gold disc on the right is a piezoelectric disk that vibrates when a voltage is applied. The three metal fingers on the PCB make contact with the disc, and a little transistor amplifier is on the printed circuit board.

## PASSIVE BUZZERS

Passive buzzers need an AC signal to produce sound. the downside to this is that they will need more complex circuitry to control them, like an oscillating 555 timer .

Passive buzzers have the advantage that they can vary the pitch or tone of the sound. Passive buzzers can be programmed to emit a wide range of frequencies or musical notes.

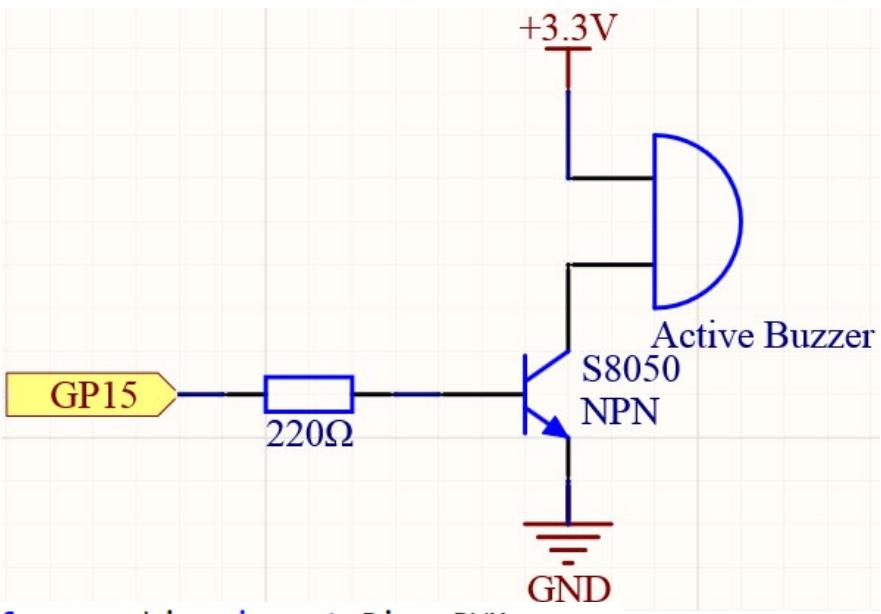
# Custom Tone: Create a Melody

## Inside an PASSIVE BUZZERS

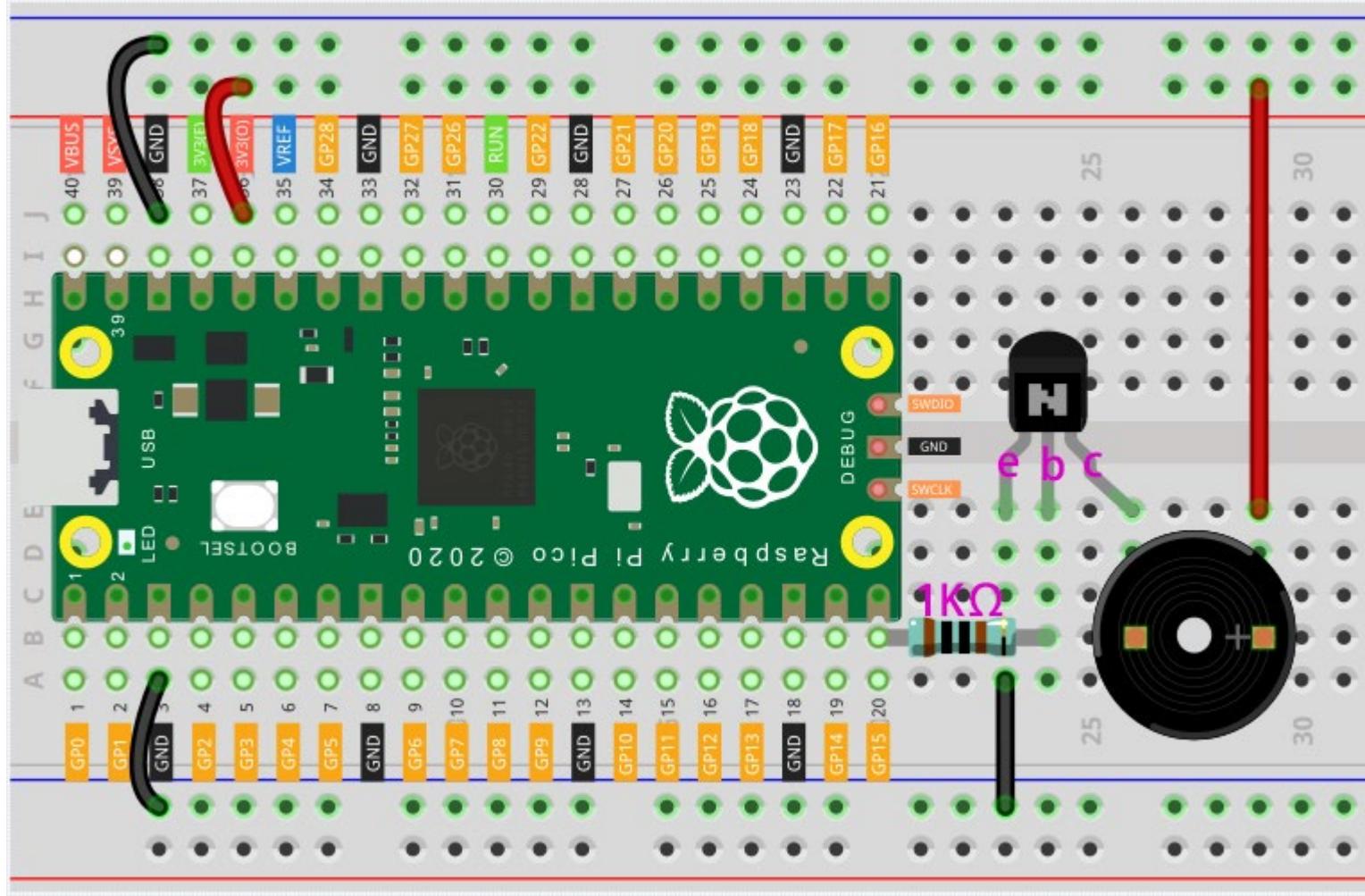


In a passive buzzer, we find an anatomy similar to a loudspeaker. There is a circular magnet surrounding an inner wire coil, with a disk that vibrates from the magnetic force generated by the electromagnetic coil.

# Custom Tone: Create a Melody



```
from machine import Pin, PWM  
import utime  
  
buzzer = PWM(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)
```

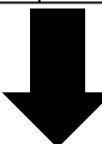


Check it in THONNY

# Custom Tone: Classic Happy Birthday Melody

The frequencies of the musical notes starting from middle C (i.e. C4) are given below. To play the tune of a melody, we need to know its musical notes. Each note is played for certain duration and there is a certain time gap between two successive notes.

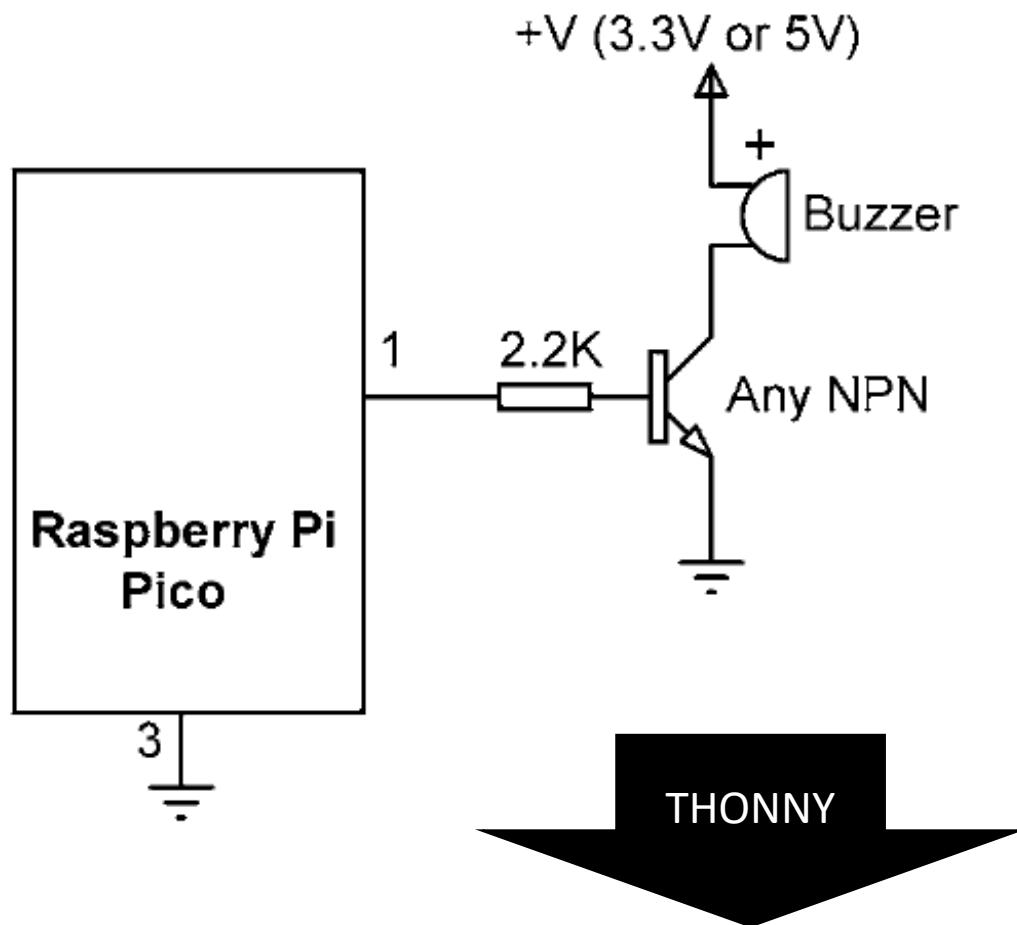
Notes	C <sub>4</sub>	C <sub>4</sub> #	D <sub>4</sub>	D <sub>4</sub> #	E <sub>4</sub>	F <sub>4</sub>	F <sub>4</sub> #	G <sub>4</sub>	G <sub>4</sub> #	A <sub>4</sub>	A <sub>4</sub> #	B <sub>4</sub>
Hz	261.63	277.18	293.66	311.13	329.63	349.23	370	392	415.3	440	466.16	493.88



Note	C <sub>4</sub>	C <sub>4</sub>	D <sub>4</sub>	C <sub>4</sub>	F <sub>4</sub>	E <sub>4</sub>	C <sub>4</sub>	C <sub>4</sub>	D <sub>4</sub>	C <sub>4</sub>	G <sub>4</sub>	F <sub>4</sub>	C <sub>4</sub>
Duration	1	1	2	2	2	3	1	1	2	2	2	3	1

Note	C <sub>4</sub>	C <sub>5</sub>	A <sub>4</sub>	F <sub>4</sub>	E <sub>4</sub>	D <sub>4</sub>	A <sub>4</sub> #	A <sub>4</sub> #	A <sub>4</sub>	F <sub>4</sub>	G <sub>4</sub>	F <sub>4</sub>	
Duration	1	2	2	2	2	2	1	1	2	2	2	2	4

# Custom Tone: Classic Happy Birthday Melody



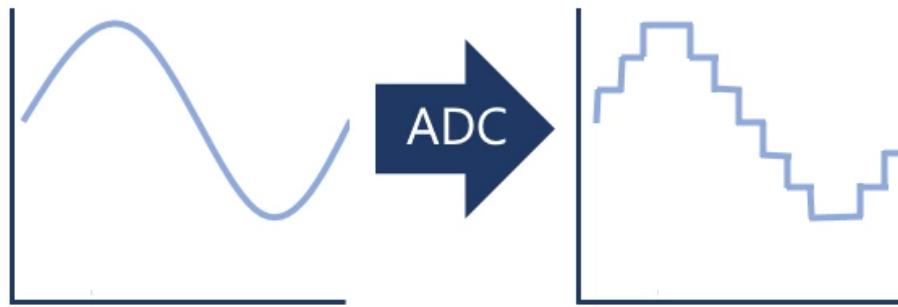
1. Change the durations between the notes and see its effects.
2. How can you make the melody run quicker?

```
from machine import Pin, PWM
import utime

ch = PWM(Pin(0))          # PWM output at GP0
MaxNotes = 25
Durations = [0]*MaxNotes
#
# Melody frequencies
#
frequency = [262,262,294,262,349,330,262,262,294,262,
392,349,262,262,524,440,349,330,294,466,
466,440,349,392,349]
#
# Frequency durations
#
duration = [1,1,2,2,2,3,1,1,2,2,2,3,1,1,2,2,2,2,
2,1,1,2,2,2,3]

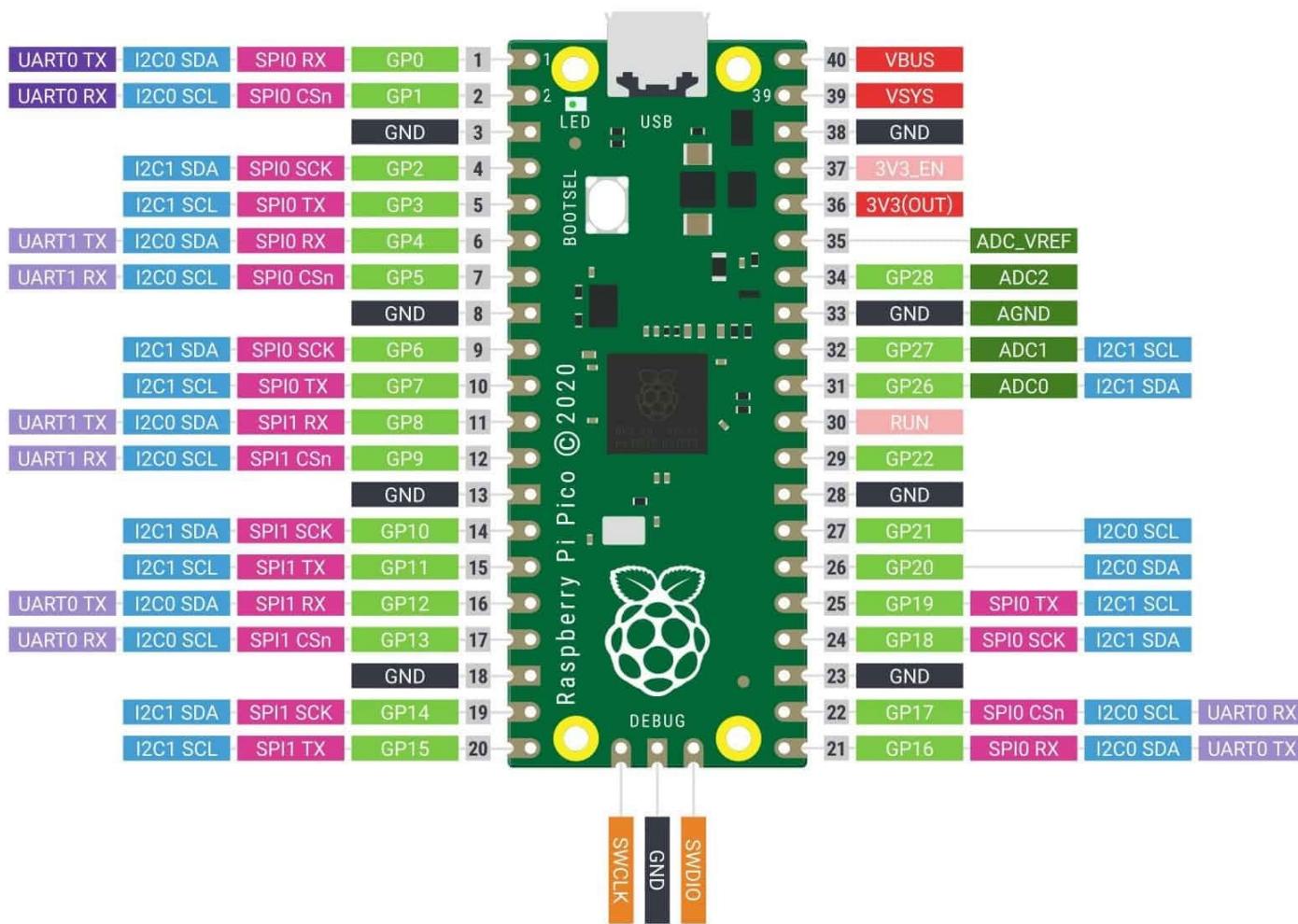
for k in range(MaxNotes):
    Durations[k] = 400 * duration[k]
while True:
    for k in range(MaxNotes):          # Do for all notes
        ch.duty_u16(32767)           # Duty cycle
        ch.freq(2*frequency[k])       # Play 2nd harmonics
        utime.sleep_ms(Durations[k])  # Durations
        utime.sleep_ms(100)            # Wait
        ch.duty_u16(0)                # Stop playing
        utime.sleep(3)                # Stop 3 seconds
```

# What is Analog to Digital Converter (ADC)?



- An Analog to Digital Converter (ADC) is a very useful feature that **converts an analog voltage on a pin to a digital number**. By converting from the analog world to the digital world, we can begin to use electronics to interface with the analog world around us.
- An analog to digital converter (ADC) is a circuit that converts a continuous voltage value (analog) to a binary value (digital) that can be understood by a digital device which could then be used for digital computation.
- The Raspberry Pi Pico board exposes 26 multi-function GPIO pins from a total of 36 GPIO pins. Out of 36 GPIO Pins, there are 4 ADC pins but only 3 are usable.
- The **ADC in Raspberry Pi Pico is 12bits**, which is 4 times better than the 10 bits ADC of the Arduino. We will write a MicroPython code to learn how we can use the ADC pin value with any analog sensors. A potentiometer is the best tool to vary the input Analog Voltage.
- With the ADC, we can sense environmental parameters like light, sound, distance, gravity, acceleration, rotation, smell, gases, other particles.
- Most microcontrollers nowadays have built-in ADC converters. It is also possible to connect an external ADC converter to any type of microcontroller. ADC converters are usually 10 or 12 bits, having 1024 to 4096 quantization levels. **A 16 bits ADC has 65536 quantization levels**. **A Raspberry Pi Pico has 12 Bits ADC with a quantization level of 4096**.

# How does an ADC work in a Microcontroller?



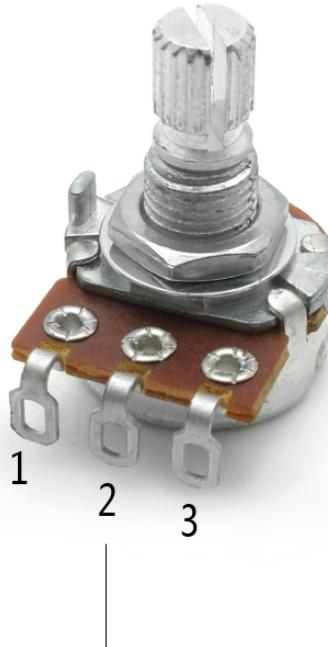
ADC Module	GPIO Pins
ADC0	GP26
ADC1	GP27
ADC2	GP28

- The Raspberry Pi Pico supports four **12-bit SAR** based analog to digital converters. Out of the 4, you can only use 3 analog channels.
- The 4th analog channel is internally connected to the **internal temperature sensor**. You can measure the temperature using build-in temperature by reading the analog value of ADC4. with GP26, GP27 & GP28 pins respectively.
- The ADC conversion speed per sample is **2μs** that is **500kS/s**. The RP2040 microcontroller operates on a 48MHz clock frequency which comes from USB PLL. So, its ADC takes a 96 CPU clock cycle to perform one conversion. Therefore, the sampling frequency is  $(96 \times 1 / 48\text{MHz}) = 2 \mu\text{s}$  per sample (500kS/s).

# Turn the Knob: Potentiometer/POT

A potentiometer (or pot) is a three-terminal resistor with a sliding or rotating contact that can be used to dynamically vary resistance.

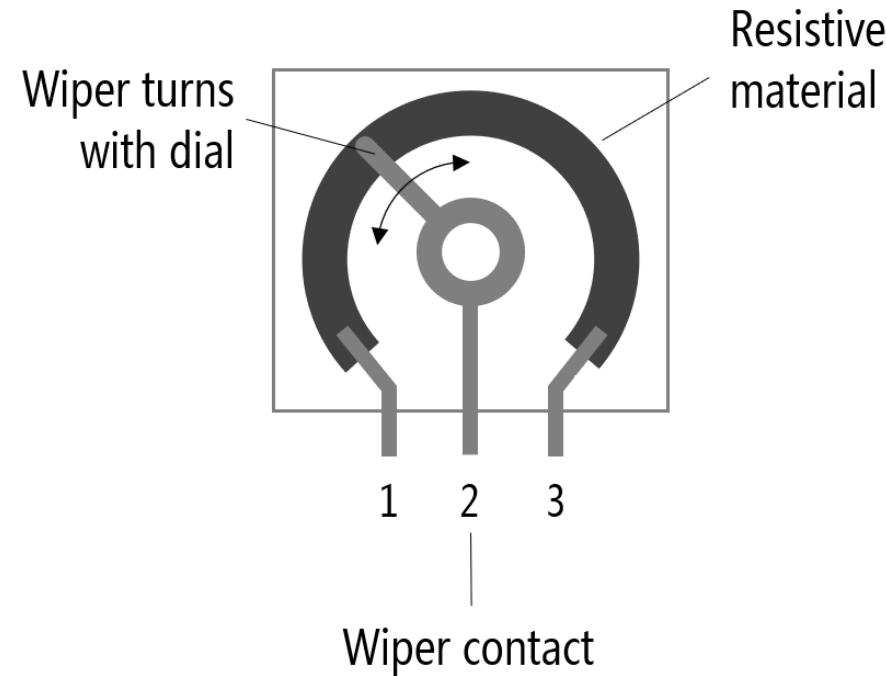
PANEL MOUNT POT 10kΩ



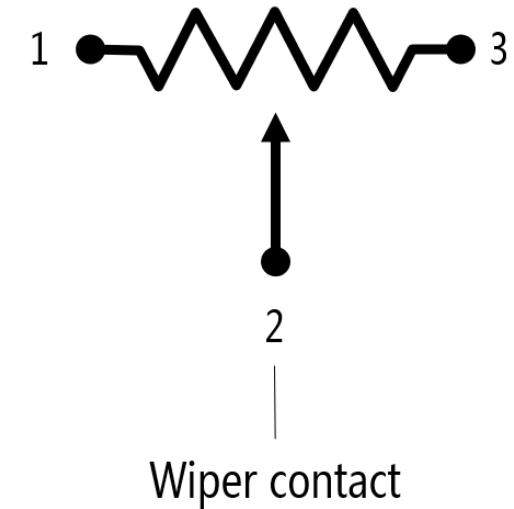
TRIM POT 10kΩ



FUNCTIONAL DIAGRAM



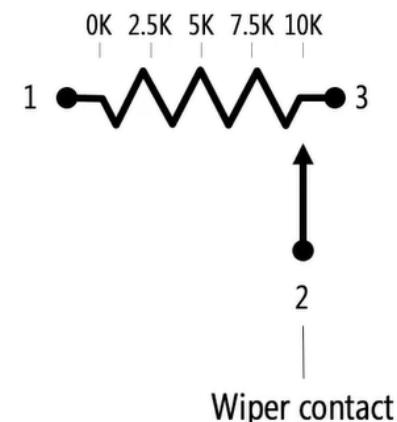
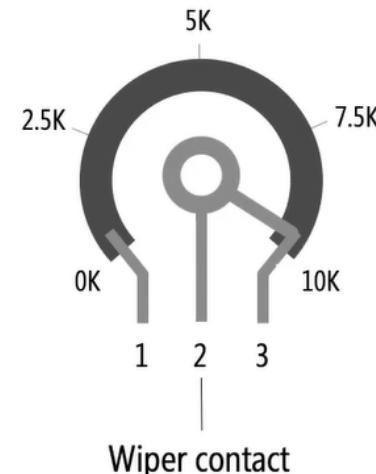
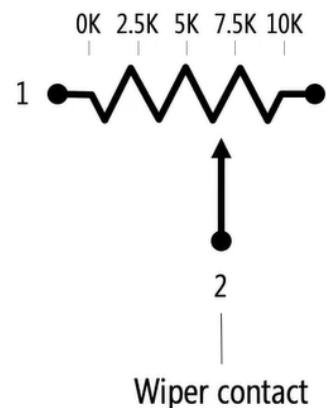
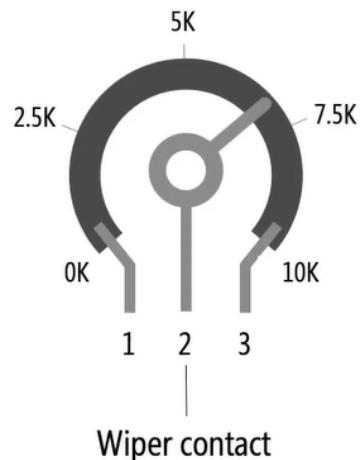
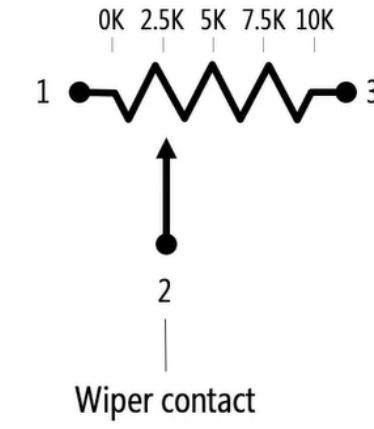
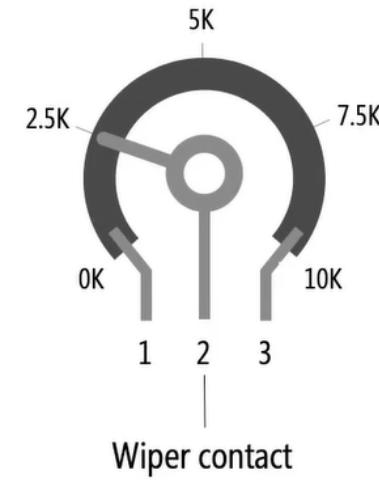
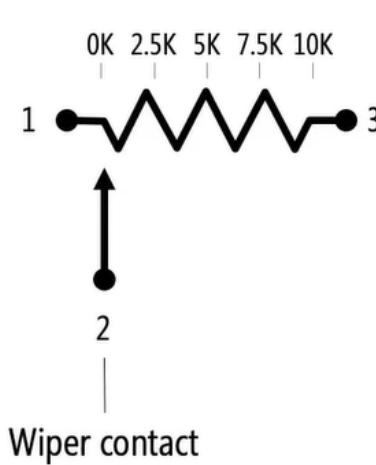
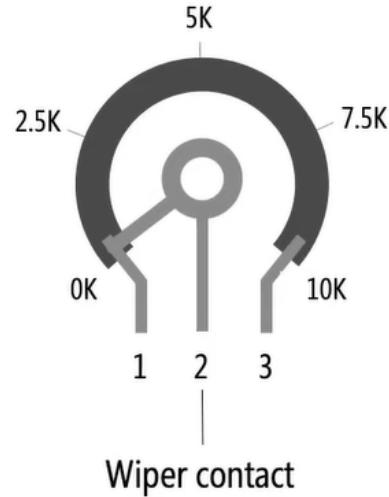
SCHEMATIC SYMBOL



Two example potentiometers commonly included in our hardware kits: a  $10\text{k}\Omega$  panel mount and a  $10\text{k}\Omega$  trim potentiometer.

# Turn the Knob: Potentiometer/POT

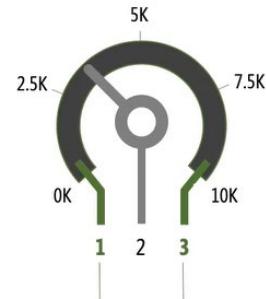
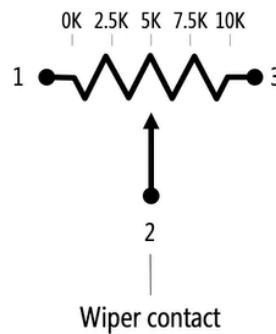
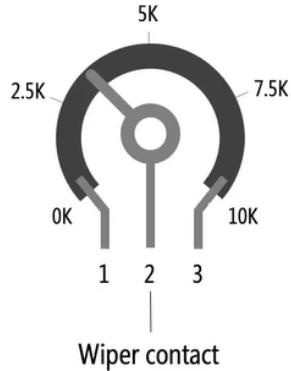
A potentiometer (or pot) is a three-terminal resistor with a sliding or rotating contact that can be used to dynamically vary resistance.



# Turn the Knob: How does a potentiometer work?

Potentiometers have three legs: the resistance between the outer two legs (Leg 1 and Leg 3) will not vary. For example, if you are using a  $10\text{k}\Omega$  potentiometer, then the resistance between Legs 1 and 3 will always be  $10\text{k}\Omega$  regardless of wiper position (Leg 2).

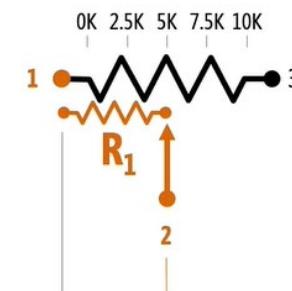
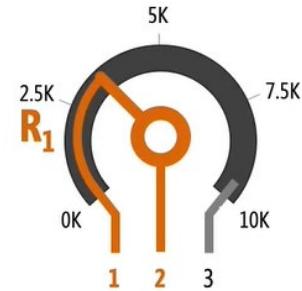
Here we have a  $10\text{ kilo ohm}$  Potentiometer



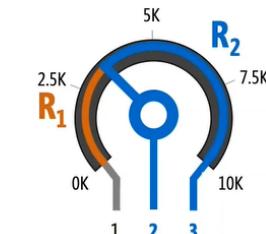
Regardless of wiper position, note that ( $R_{\text{tot}}$ ) between Legs 1 and 3 is always  $10\text{k}\Omega$  (this is a  $10\text{k}\Omega$  potentiometer)



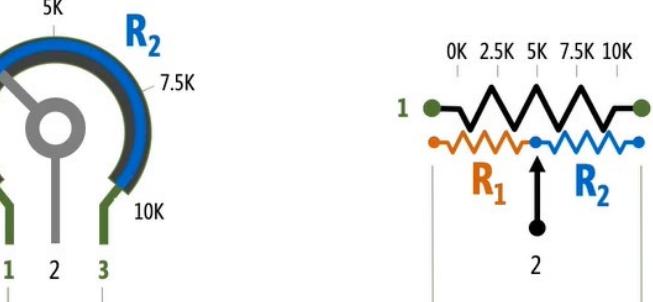
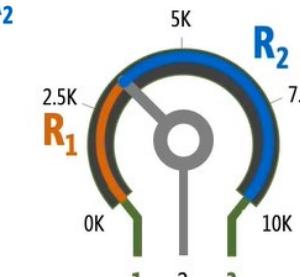
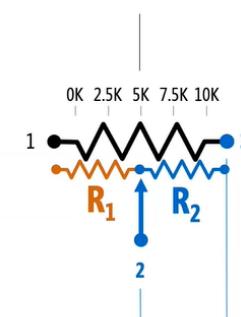
The resistance between Legs 1 and 2 changes depending on wiper position.



The resistance between Legs 1 and 2 changes depending on wiper position. Let's call this  $R_1$ .



Simultaneously, the wiper also affects the resistance between Legs 2 and 3. Let's call this  $R_2$ .

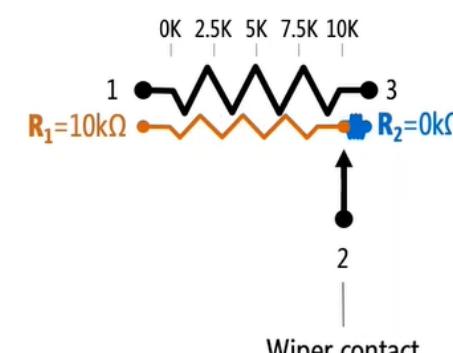
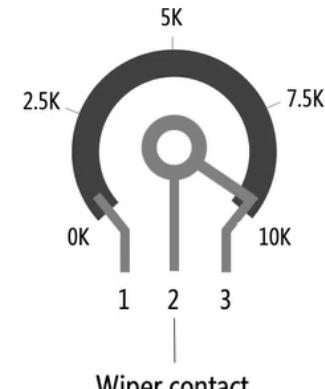
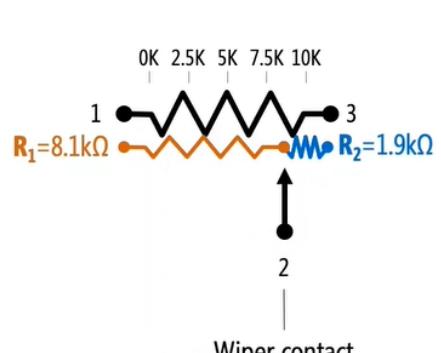
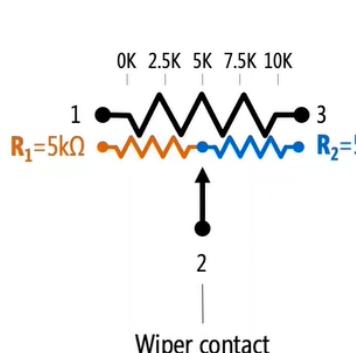
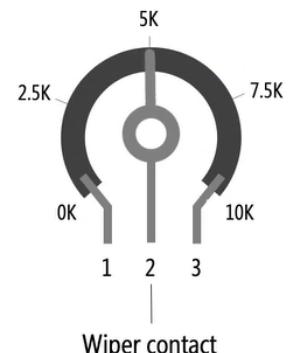
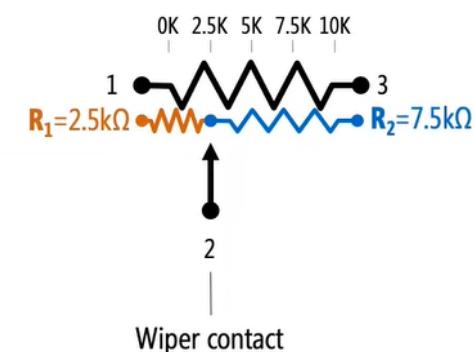
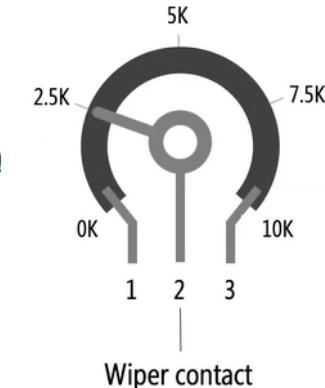
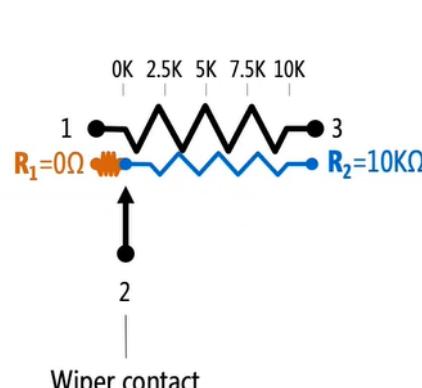
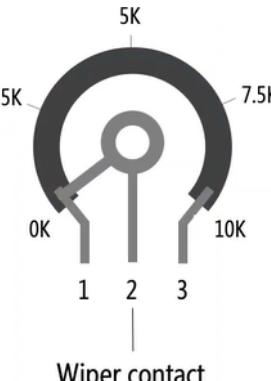
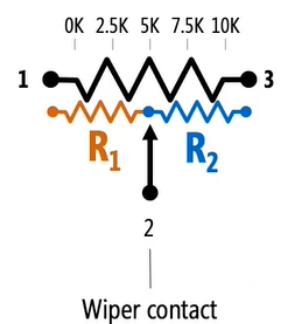
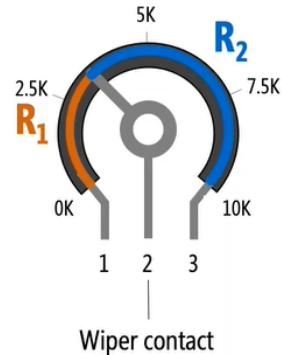


And that  $R_{\text{tot}}$  always equals  $R_1 + R_2$

# Turn the Knob: How does a potentiometer work?

Potentiometers have three legs: the resistance between the outer two legs (Leg 1 and Leg 3) will not vary. For example, if you are using a  $10\text{k}\Omega$  potentiometer, then the resistance between Legs 1 and 3 will always be  $10\text{k}\Omega$  regardless of wiper position (Leg 2).

## Let's Try Few Wiper Positions



# Turn the Knob: Welcome to Analog World

WOKWI SAVE SHARE Docs B

main.py • diagram.json • PIO 🐍

```
1 import machine
2 import utime
3
4 analog_value = machine.ADC(28)
5
6 while True:
7     reading = analog_value.read_u16()
8     print("ADC: ", reading)
9     utime.sleep(0.2)
10
```

Simulation

00:52.243 99%

THONNY

ADC: 65535  
ADC: 65535  
ADC: 65535  
ADC: 65535  
ADC: 65535

# Turn the Knob: Control an LED using POT

WOKWi<sup>®</sup> SAVE SHARE Docs B

main.py • diagram.json • PIO

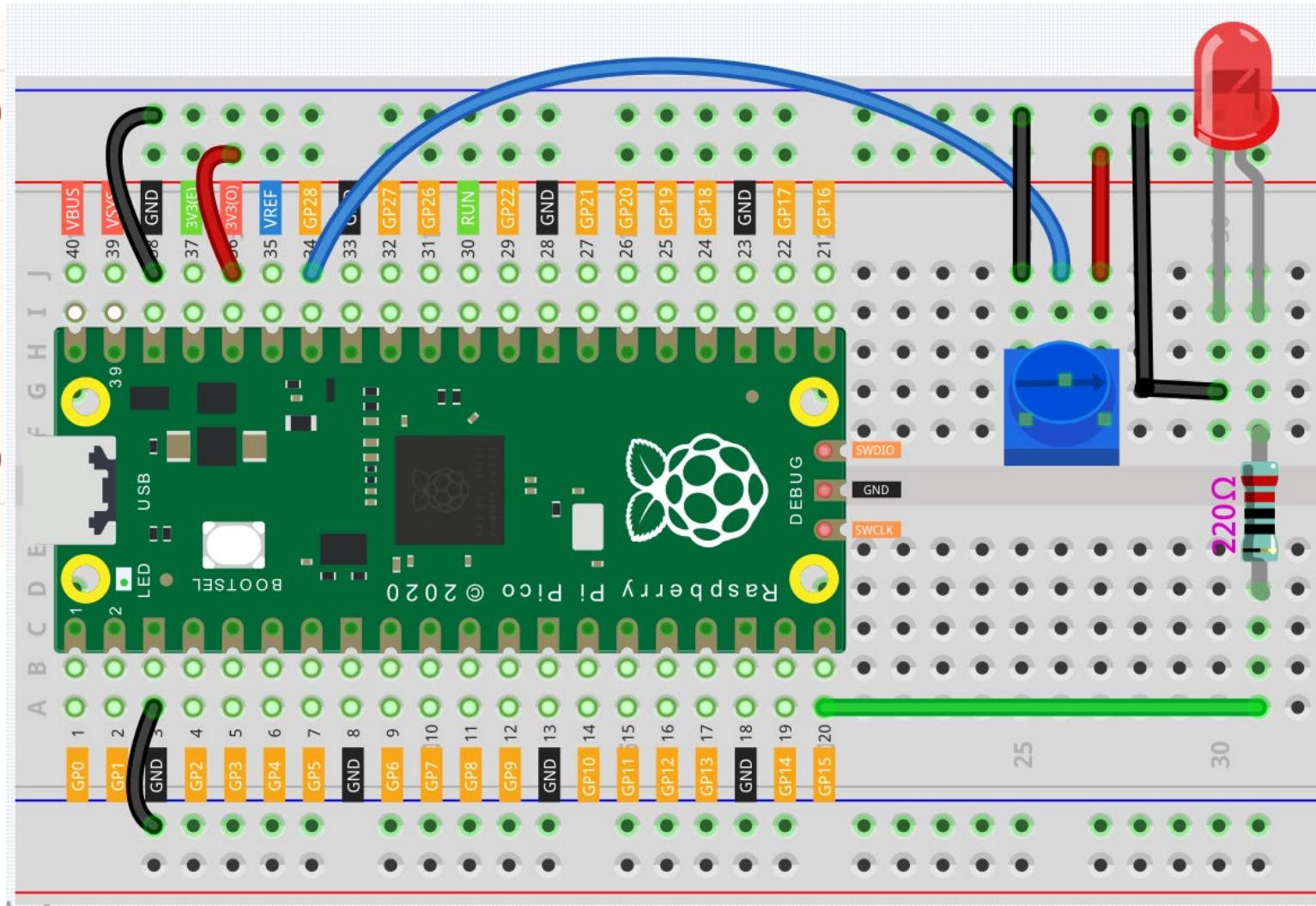
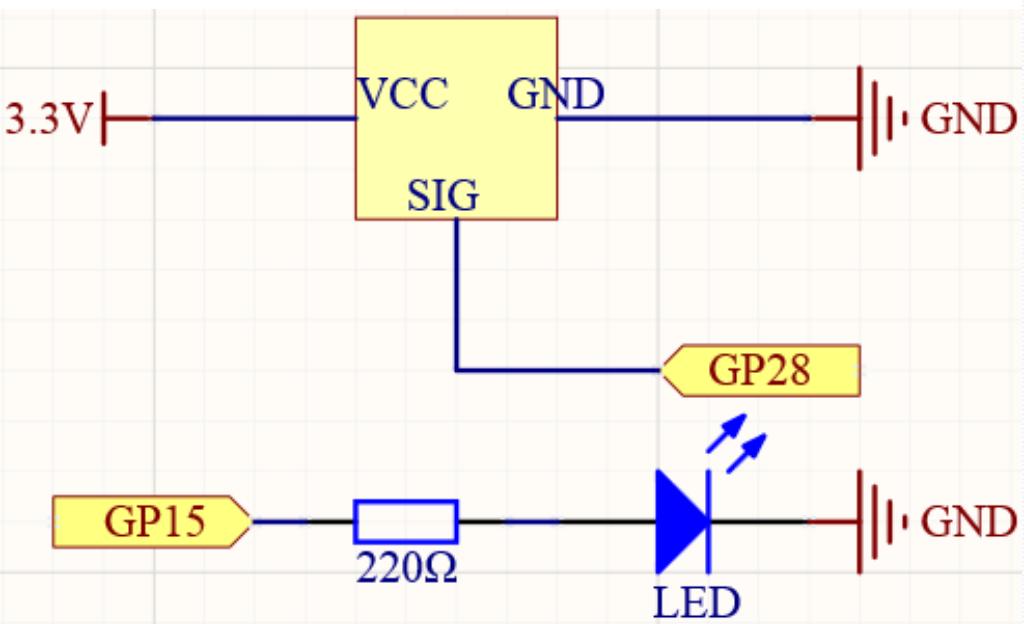
```
1 import machine
2 import utime
3
4 potentiometer = machine.ADC(28)
5 led = machine.PWM(machine.Pin(15))
6 led.freq(1000)
7
8 while True:
9     value=potentiometer.read_u16()
10    print(value)
11    led.duty_u16(value)
12    utime.sleep_ms(200)
```

Simulation

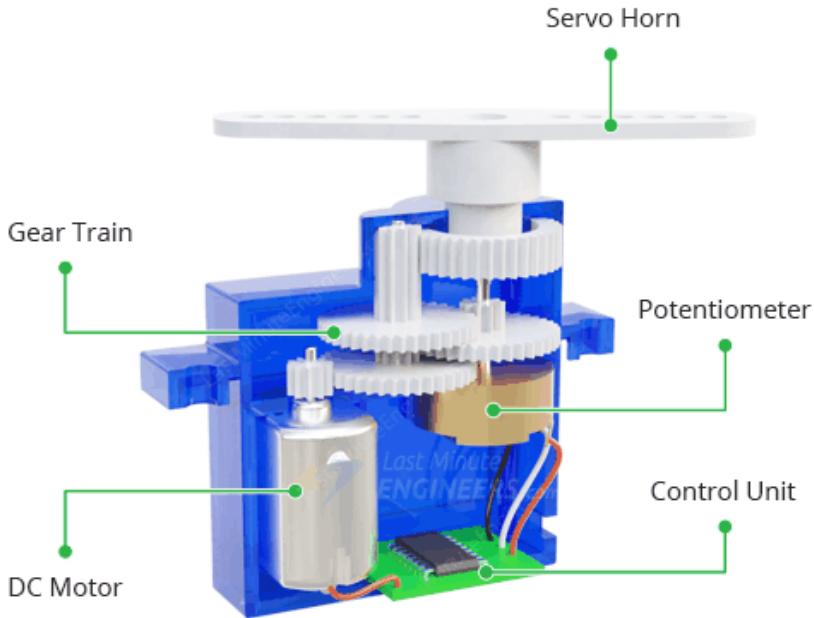
00:12.432 99%

8514  
8514  
8514  
8514

# Turn the Knob: Control an LED using POT



# Swing the servo : What is a Servo and what makes it precise?



Servo motor is a component that can rotate its handle (usually between 0° and 180°). It used to control the angular position of the object.

## Pinout

The servo motor used in this example includes three pins:

- ◆ **VCC pin:** (typically red) needs to be connected to **vcc** (5V)
- ◆ **GND pin:** (typically black or brown) needs to be connected to **GND** (0v)
- ◆ **Signal pin:** (typically yellow or orange) receives the PWM control signal from an

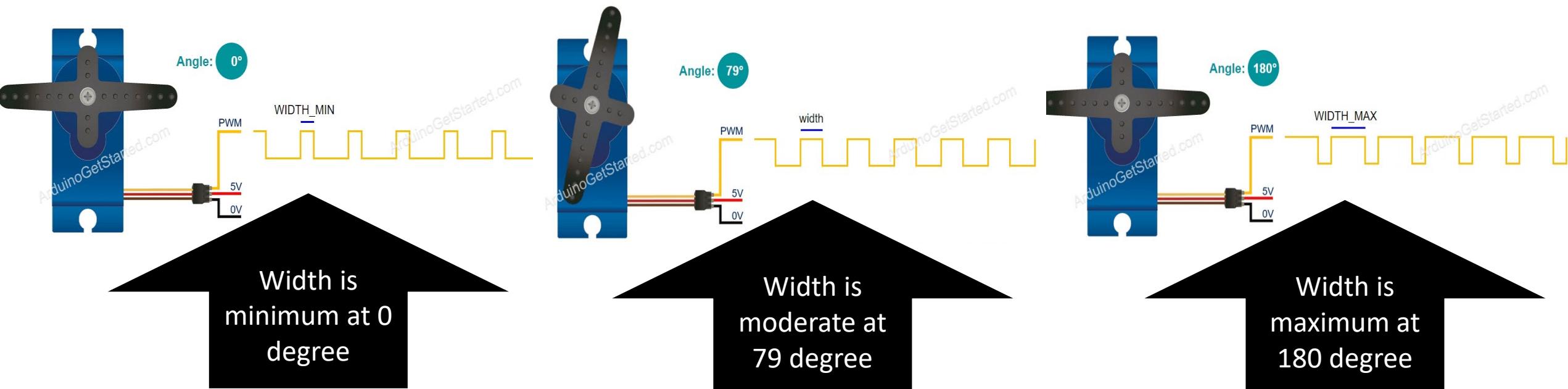
**Servo / Pinout**

# Swing the servo : Servo and Pulse Width

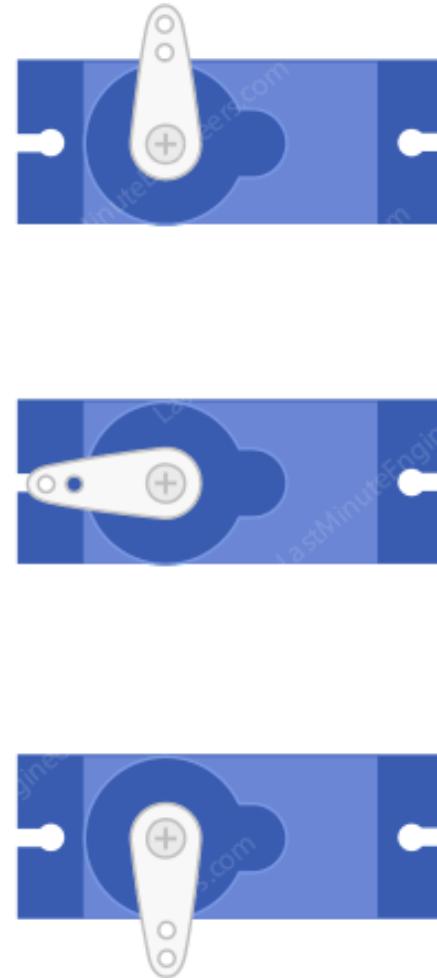
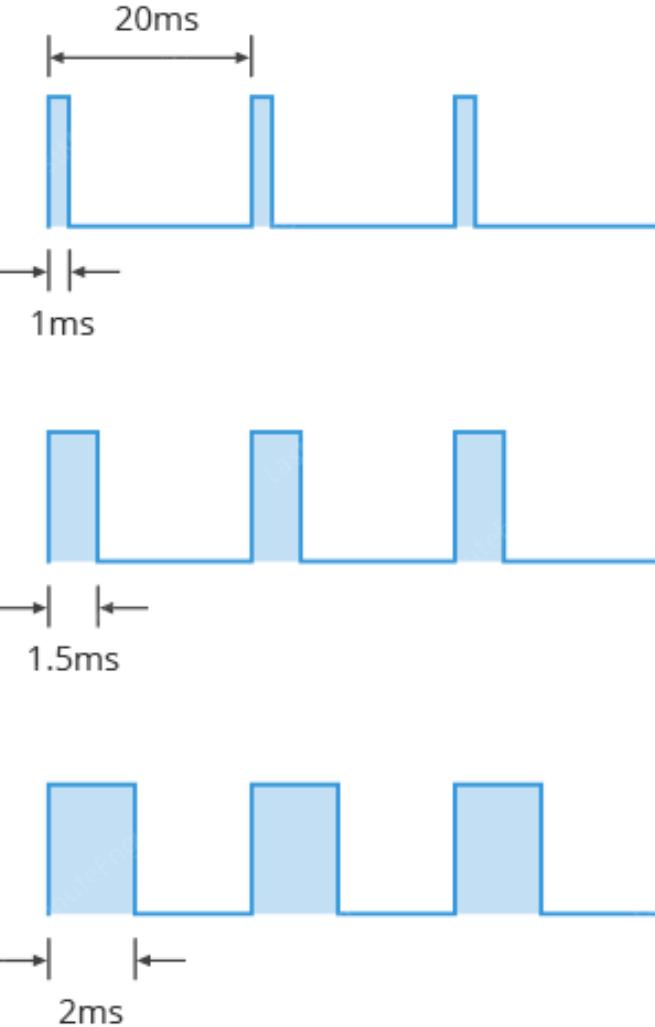
Servos are motors that allow us to **precisely control** physical movement because they generally move to a position rather than continuously rotating. They are simple to connect and control because the motor driver is built right into them.

Servos contain a small DC motor connected to the output shaft through gears. The output shaft drives a servo horn and is also linked to a potentiometer (pot).

The potentiometer provides position feedback to the error amplifier in the control unit, which compares the current position of the motor to the target position.



# Swing the servo : Servo and Pulse Width



0 Degrees

A short pulse of 1 ms or less will rotate the servo to 0 degrees (one extreme).

90 Degrees

A pulse duration of 1.5 ms will rotate the servo to 90 degrees (middle position).

180 Degrees

A pulse duration of 2 ms or so will rotate the servo to 180 degrees (other extreme).

# Swinging Servo

WOKWI

SAVE

SHARE

Docs

B

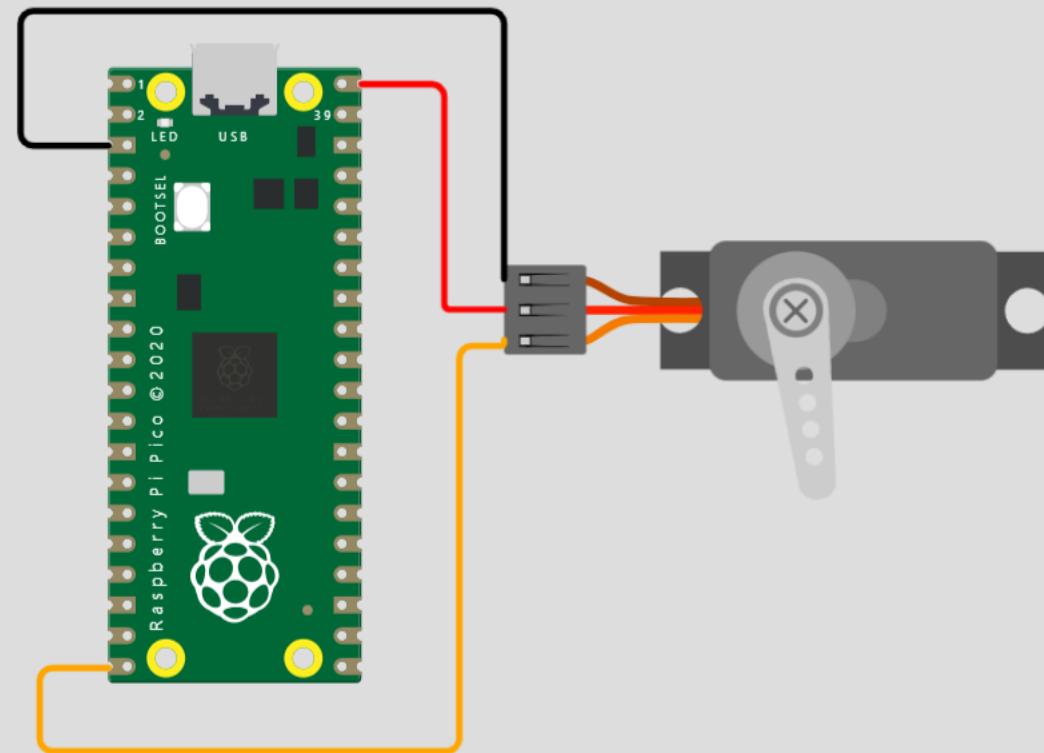
main.py • diagram.json • PIO 🚗

```
1  from machine import Pin, PWM
2  from utime import sleep_ms
3
4  servo = PWM(Pin(15))
5  servo.freq(50)
6
7  def interval_mapping(x, in_min, in_max, out_min, out_max):
8      return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
9
10 def servo_write(pin,angle):
11     pulse_width=interval_mapping(angle, 0, 180, 0.5,2.5)
12     duty=int(interval_mapping(pulse_width, 0, 20, 0,65535))
13     pin.duty_u16(duty)
14
15 while True:
16     for angle in range(180):
17         servo_write(servo,angle)
18         sleep_ms(20)
19     for angle in range(180,-1,-1):
20         servo_write(servo,angle)
21         sleep_ms(20)
22
```

Simulation

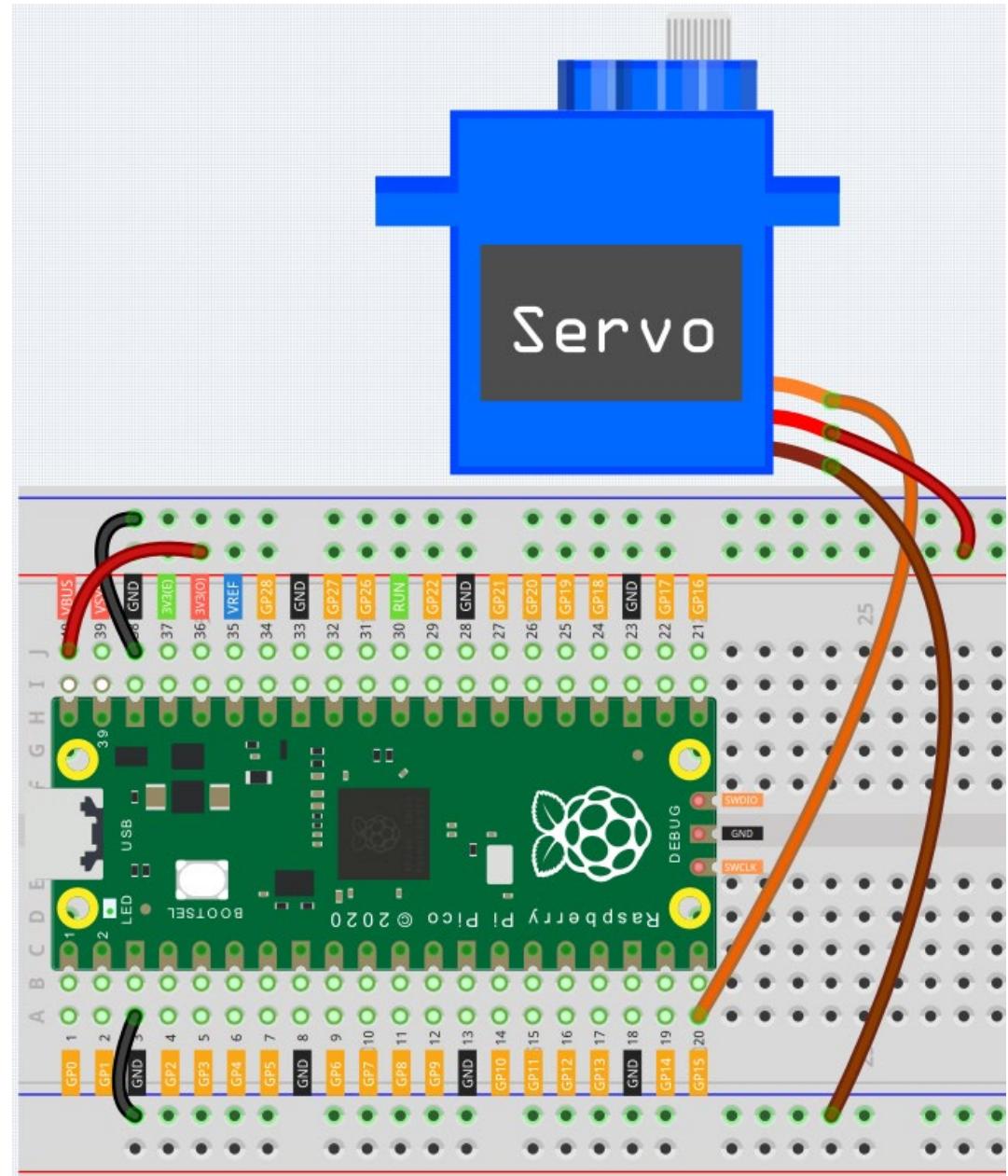
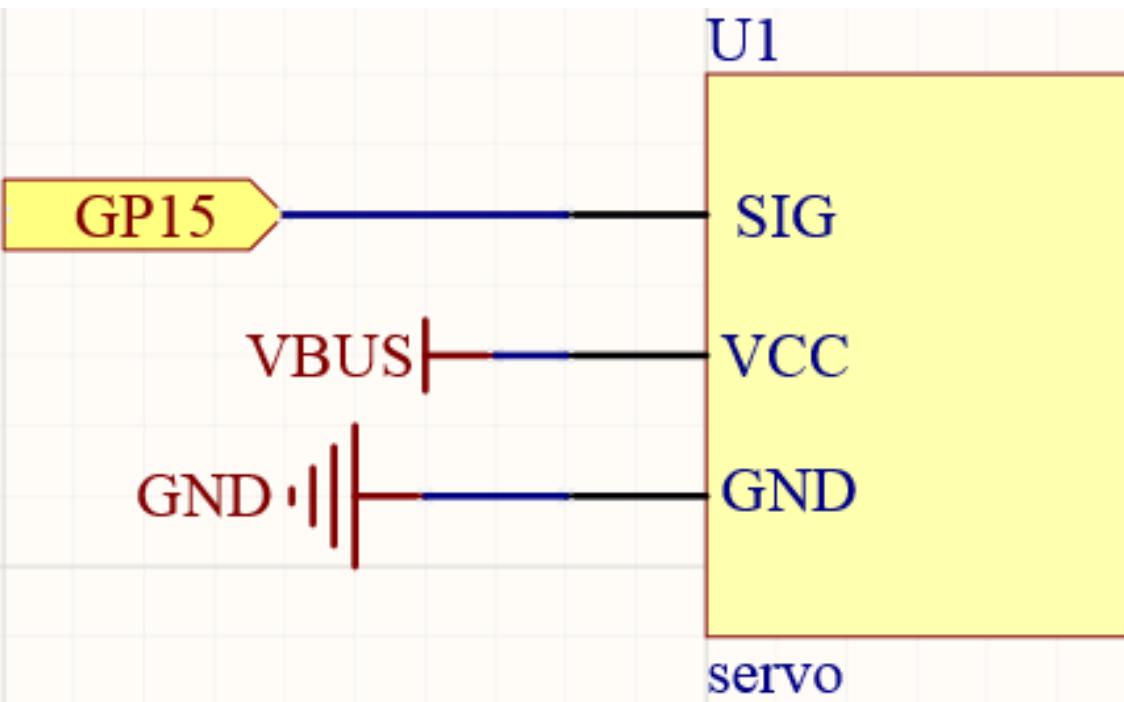


⌚ 00:25.778 ⚡ 99%



# Swinging 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.



# Swing the servo from left to right using POT

WOKWi

SAVE



SHARE

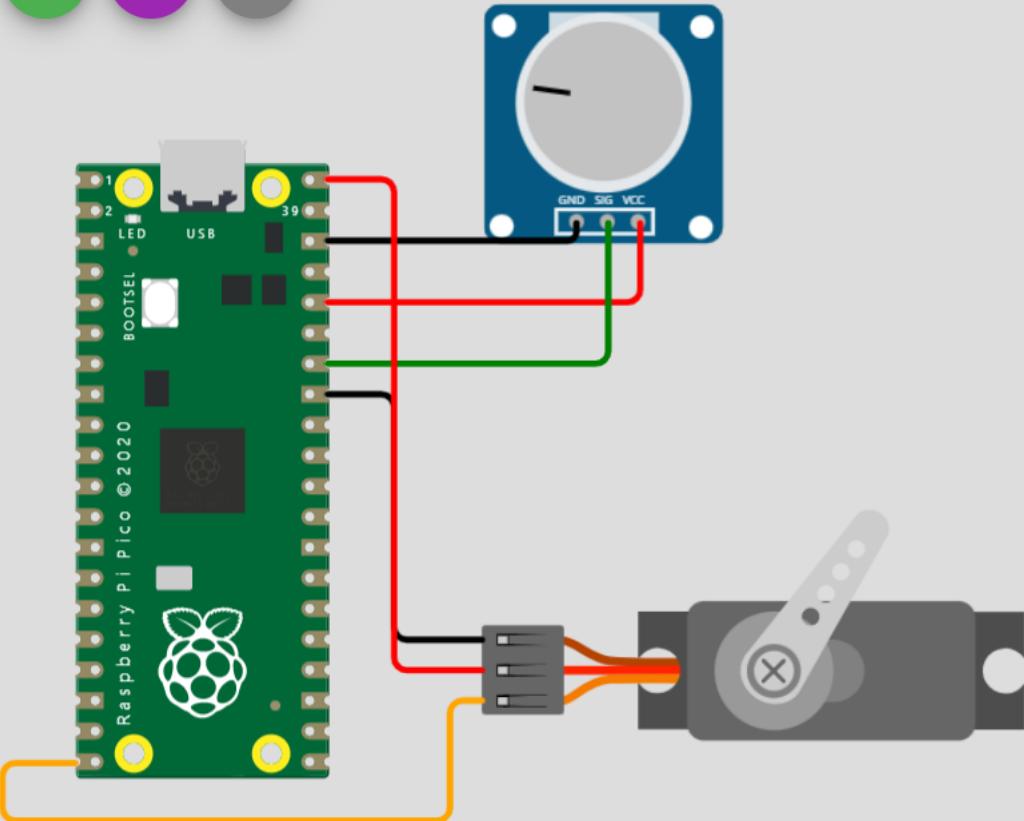
Docs

B

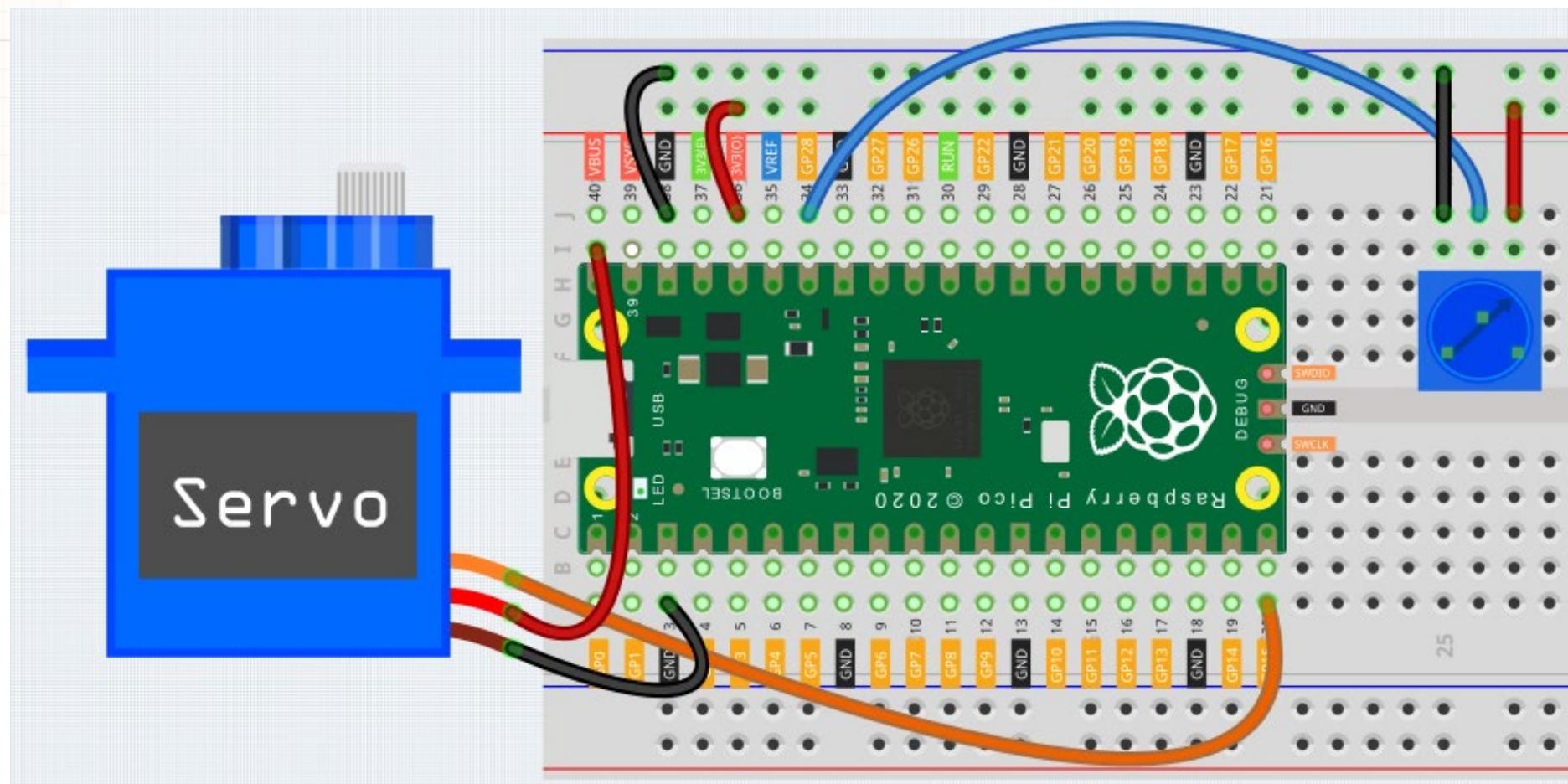
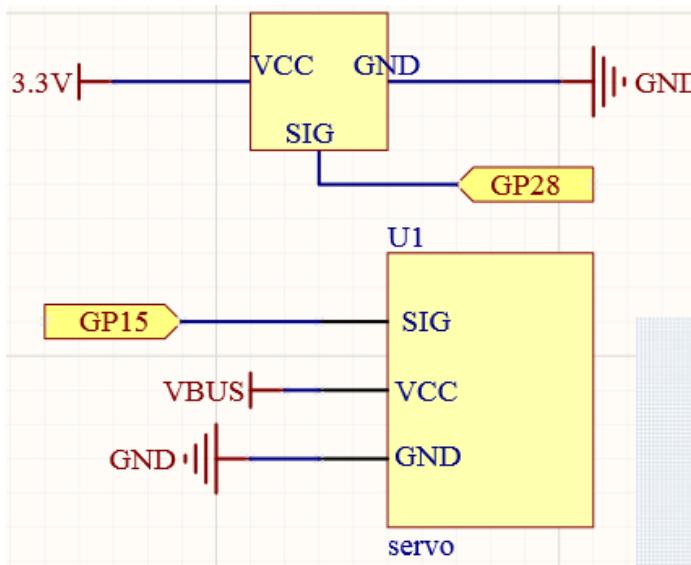
main.py ● diagram.json ●

```
1 import machine
2 import utime
3
4 potentiometer = machine.ADC(28)
5 servo = machine.PWM(machine.Pin(15))
6 servo.freq(50)
7
8 def interval_mapping(x, in_min, in_max, out_min, out_max):
9     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
10
11 def servo_write(pin,angle):
12     pulse_width=interval_mapping(angle, 0, 180, 0.5,2.5)
13     duty=int(interval_mapping(pulse_width, 0, 20, 0,65535))
14     pin.duty_u16(duty)
15
16 while True:
17     value=potentiometer.read_u16()
18     angle=interval_mapping(value,0,65535,0,180)
19     servo_write(servo,angle)
20     utime.sleep_ms(200)
21
```

Simulation



# Swing the servo from left to right using POT



# Thermometer: Let's Measure some temperature

Thermistors are **simple, inexpensive, and accurate** components that make it easy to get temperature data for your projects. Thermistors or **THERMally sensitive resISTORs** are **variable resistors**, whose **resistance will be changed with the temperature**. This feature enables us to read the temperature or change in temperature by measuring the resistance of the thermistors. Thermistors are used in various everyday use items like Thermostats, SMPS, surge protection circuits and rechargeable battery packs, etc.

## Types of Thermistors

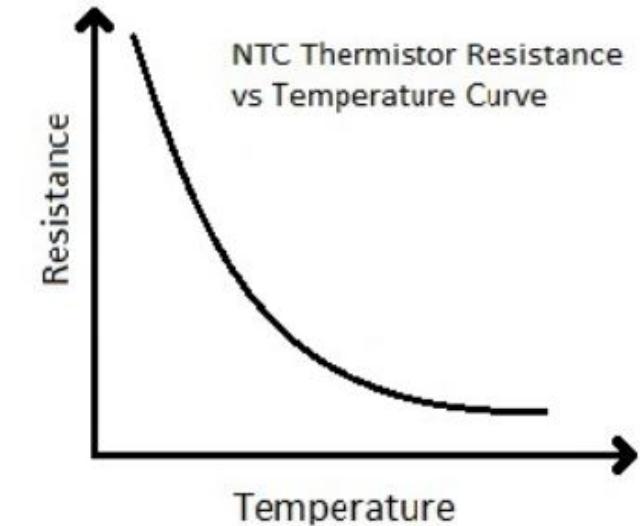
Depending on the materials used and how they react with the temperature thermistors are classified into **two types**:

- Negative Temperature Coefficient Thermistors or **NTC Thermistors**
- Positive Temperature Coefficient Thermistors or **PTC Thermistors**

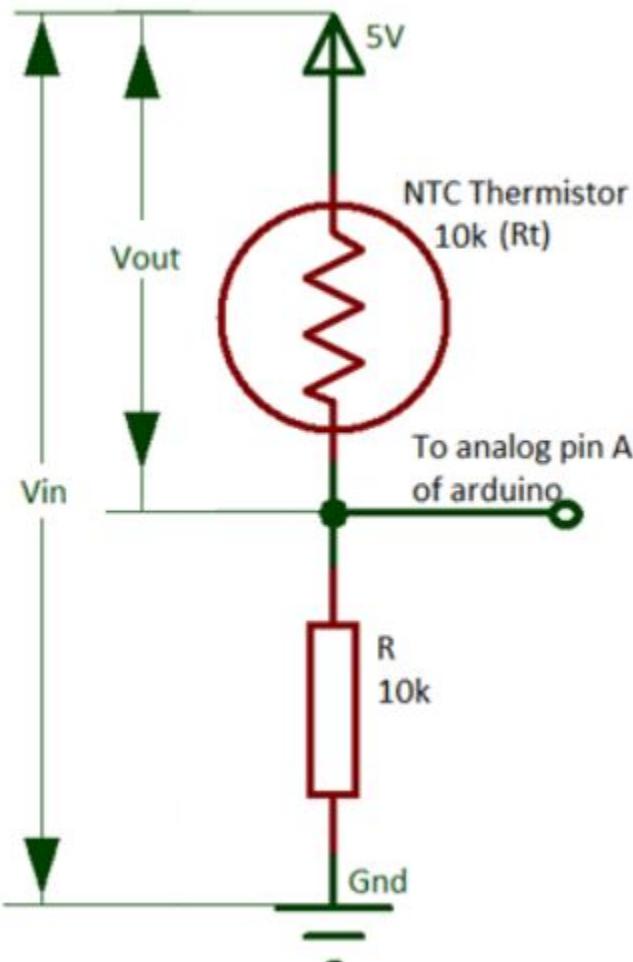
**The resistance of PTC thermistor increases with temperature, while the condition of NTC is opposite to the former.**

# Thermometer: Let's Measure some temperature

## NTC Thermistor Pinout



# Thermometer: Calculation of Temperature



Get the value of V<sub>out</sub> from analog pin Ao

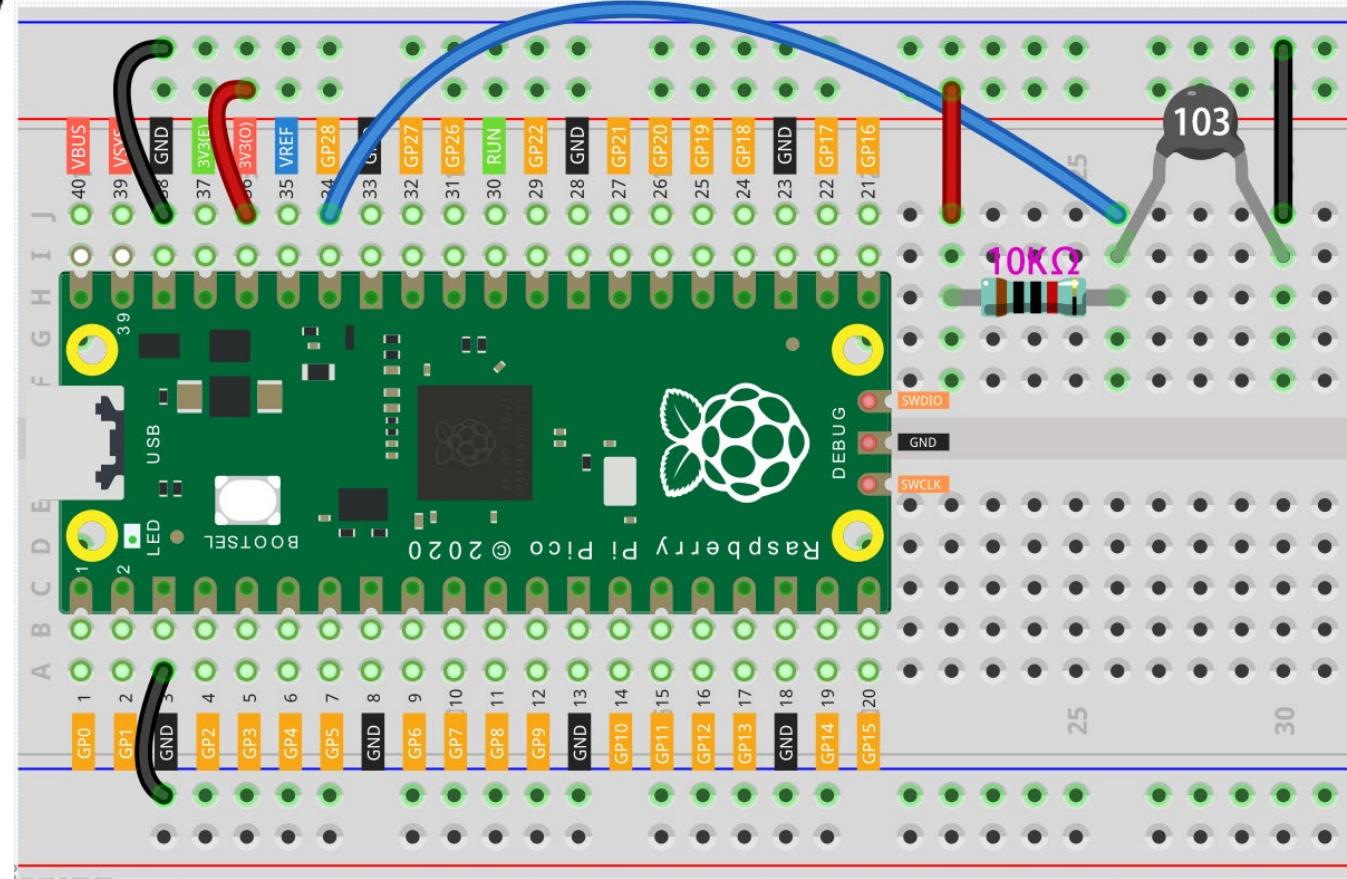
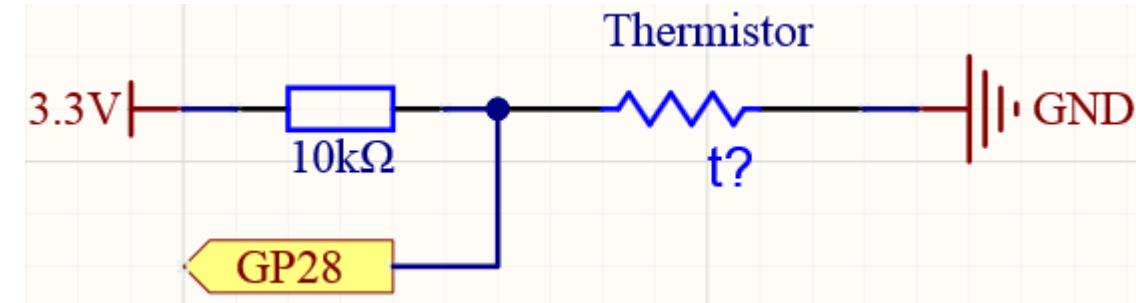
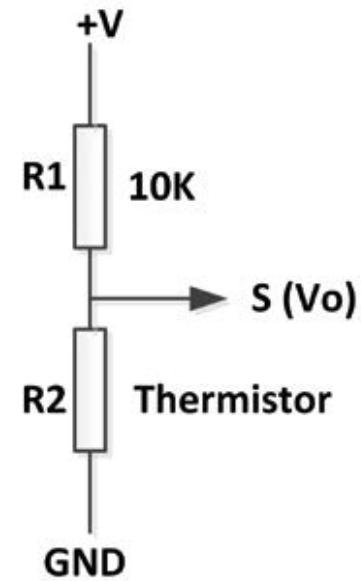
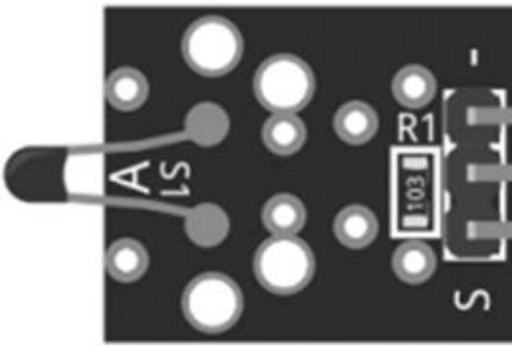
Find the value of R<sub>t</sub> through the formula:

$$V_{out} = \frac{(Vin * R_t)}{R + R_t}$$

Put the value of R<sub>t</sub> in Stein-Hart equation to get the value of temperature in kelvin

$$T = \frac{1}{A + B \ln(R_t) + C (\ln(R_t))^3}$$

# Thermometer: Calculation of Temperature



# Thermometer: Calculation of Temperature

WOKWi SAVE SHARE Docs B

main.py • diagram.json • PIO

```
1 from machine import ADC
2 import utime
3 import math
4
5 thermistor = ADC(28)
6
7 while True:
8     temperature_value = thermistor.read_u16()
9     Vr = 3.3 * float(temperature_value) / 65535
10    Rt = 10000 * Vr / (3.3 - Vr)
11    temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
12    Cel = temp - 273.15
13    Fah = Cel * 1.8 + 32
14    print ('Celsius: %.2f C Fahrenheit: %.2f F' % (Cel, Fah))
15    utime.sleep_ms(200)
16
```

Simulation

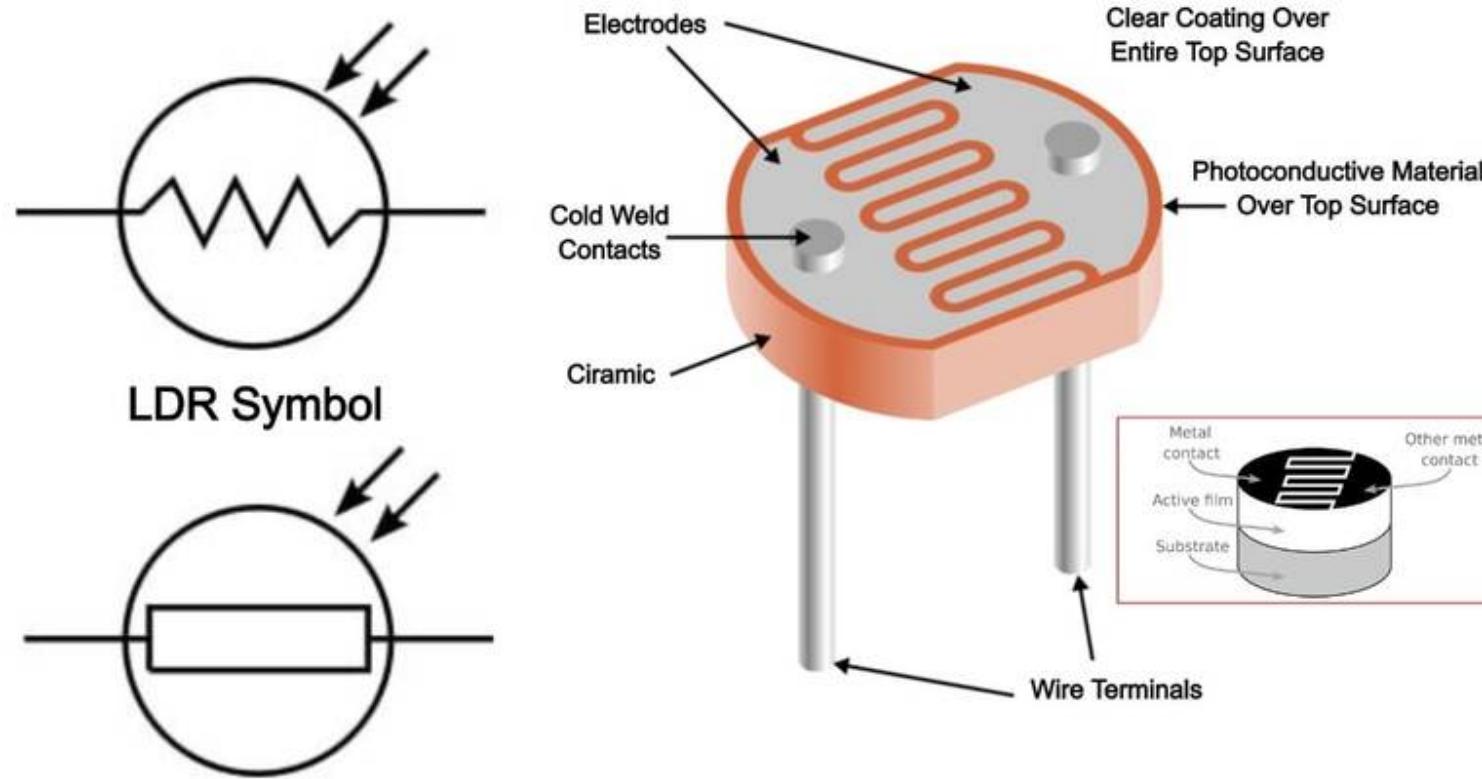
NTC Temperature Sensor (analog)

Temperature: 30.3°C

Celsius: 30.31 C Fahrenheit: 86.57 F  
Celsius: 30.31 C Fahrenheit: 86.57 F  
Celsius: 30.31 C Fahrenheit: 86.57 F  
Celsius: 30.31 C Fahrenheit: 86.57 F

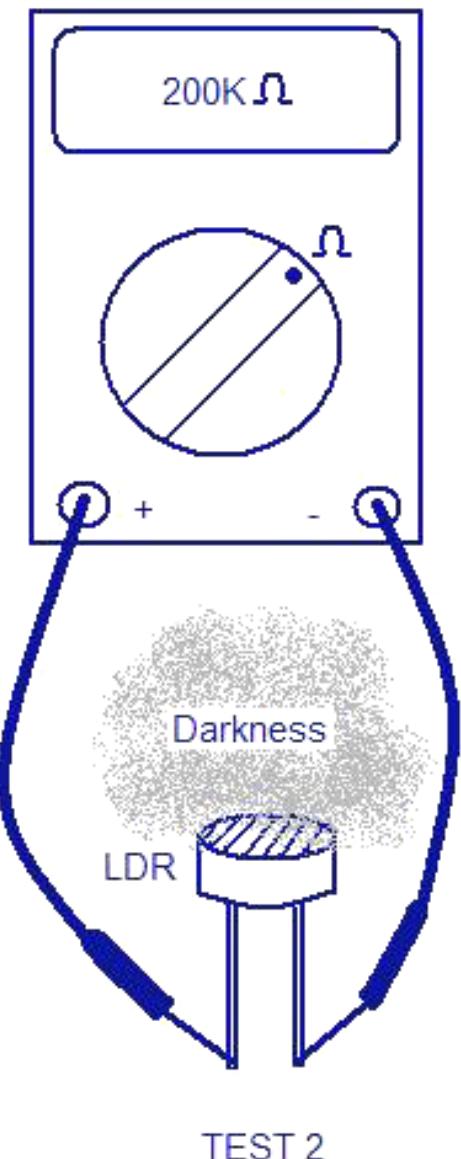
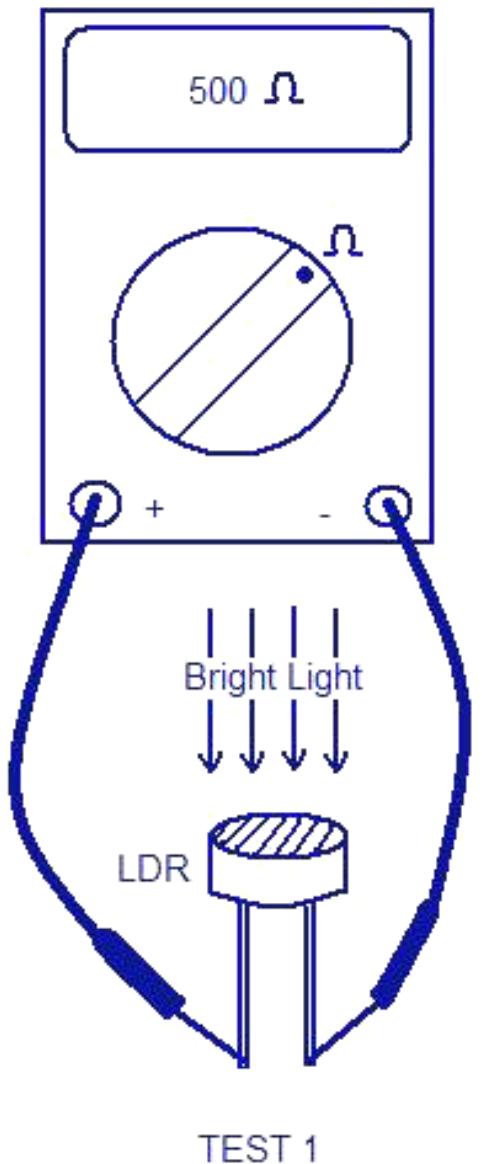
Complete on THONNY

# Light Dependent Resistor (LDR)

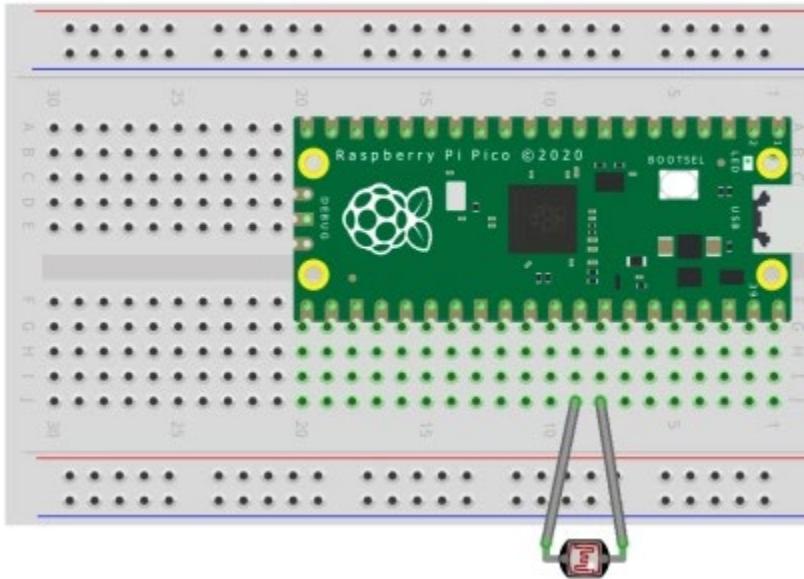


- Light Dependent Resistor or LDR or Photoresistor are electronic components that are often used in electronic circuit designs where it is necessary to detect the presence or the level of light.
- LDRs are very different from other forms of resistor like the carbon film resistor, metal oxide film resistor, metal film resistor, and the like that are widely used in other electronic designs.
- They are specifically designed for their light sensitivity and the change in resistance this causes.

# LDR Testing



# LDR Testing



Complete  
it in  
THONNY

```
from machine import Pin, ADC  
from utime import sleep
```

```
ldr = ADC(27)
```

```
while True:  
    print(ldr.read_u16())  
    sleep(2)
```

- connect one end of the LDR to GP27 (=GPIO 27 or ADC1)
- connect the other end of the LDR to a GND (ground) pin

Check LDR in

1) Dark Condition

2) Low to Bright Light  
Condition

# LDR Testing

WOKwi

SAVE ▾ SHARE Docs B

main.py • diagram.json • PIO

```
1 from machine import Pin, ADC
2 from utime import sleep
3
4 ldr = ADC(27)
5
6 while True:
7     print(ldr.read_u16())
8     sleep(2)
9
```

Simulation

00:54.630 96%

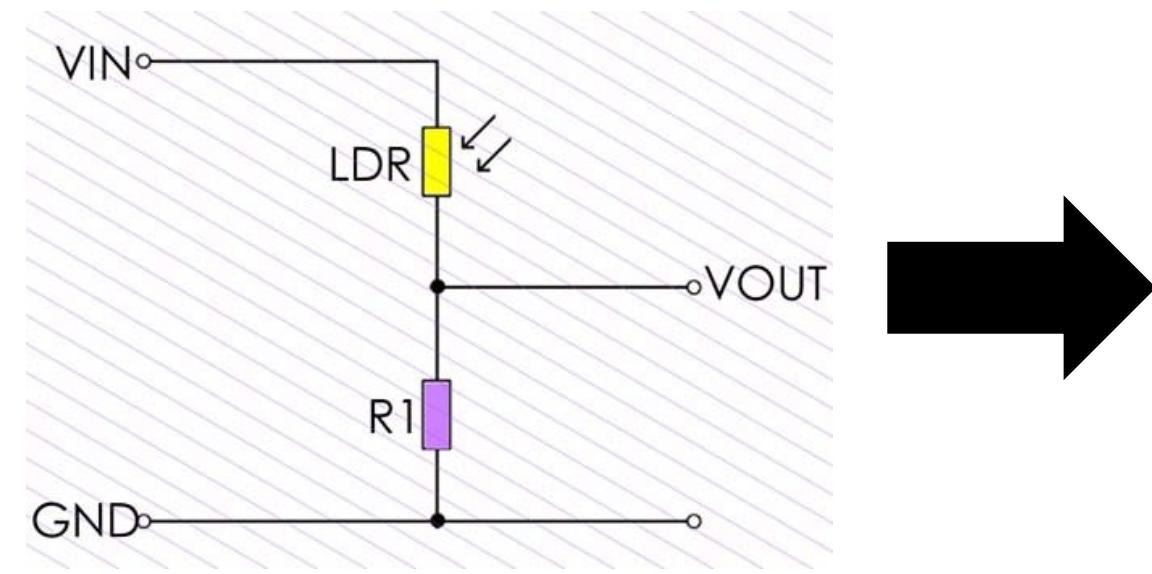
Photoresistor (LDR)

ILLUMINATION (LUX) 229

229 lux

# Light Dependent Resistor (LDR)

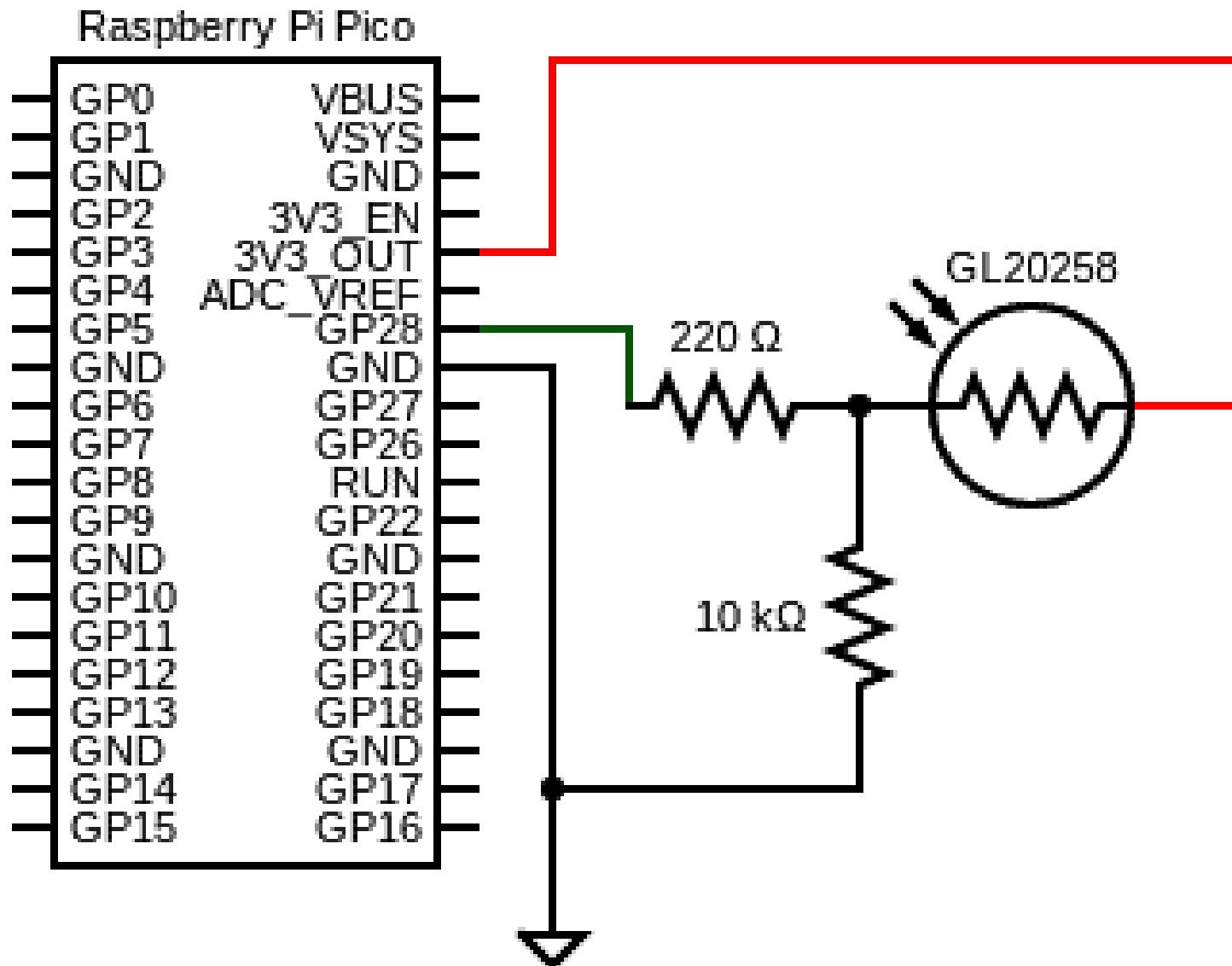
A photoresistor is in fact a light-dependent resistor (LDR). As the name suggests, it acts just like a variable resistor, changing its resistance in response to how much light is falling on it. Usually, a photoresistor has a very high resistance in the absence of light and very low resistance when stepped into the light.



Can you calculate Vout?

If you feed this VOUT to an ADC pin of Raspberry Pi Pico (**GP28/ADC2** in this experiment), compared to the reference voltage (3.3V) provided by Raspberry Pi Pico, you will get a value proportional to the light exposed to the photoresistor. With ADC module from MicroPython, this value will range between 0 and 65535, and we'll transform it roughly into a percentage scale (0% and 100%).

# Light Dependent Resistor (LDR)



# Light Dependent Resistor (LDR)

WOKWI

 **SAVE**

 SHARE

Docs

B

main.py • diagram.json • PIO 

```
1 from machine import ADC, Pin
2 from time import sleep
3
4 photoPIN = 28
5
6 def readLight(photoGP):
7     photoRes = ADC(Pin(28))
8     light = photoRes.read_u16()
9     light = round(light/65535*100,2)
10    return light
11
12 while True:
13     print("light: " + str(readLight())
14     sleep(1)
```

## Simulation

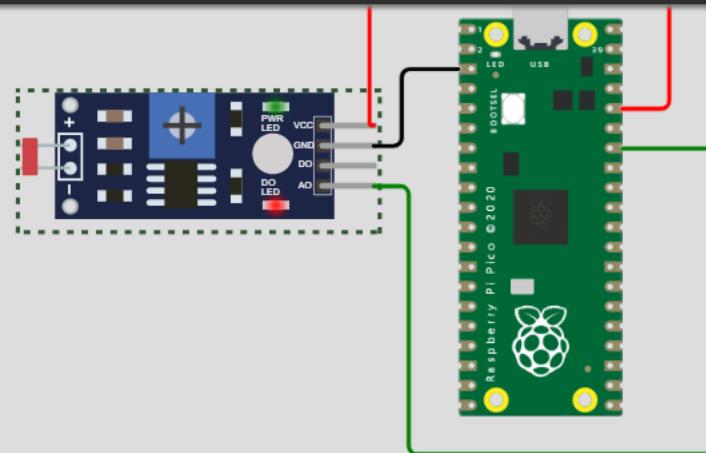


⌚ 00:25.226 ⚡ 96%

## Photoresistor (LDR)



100000 lux



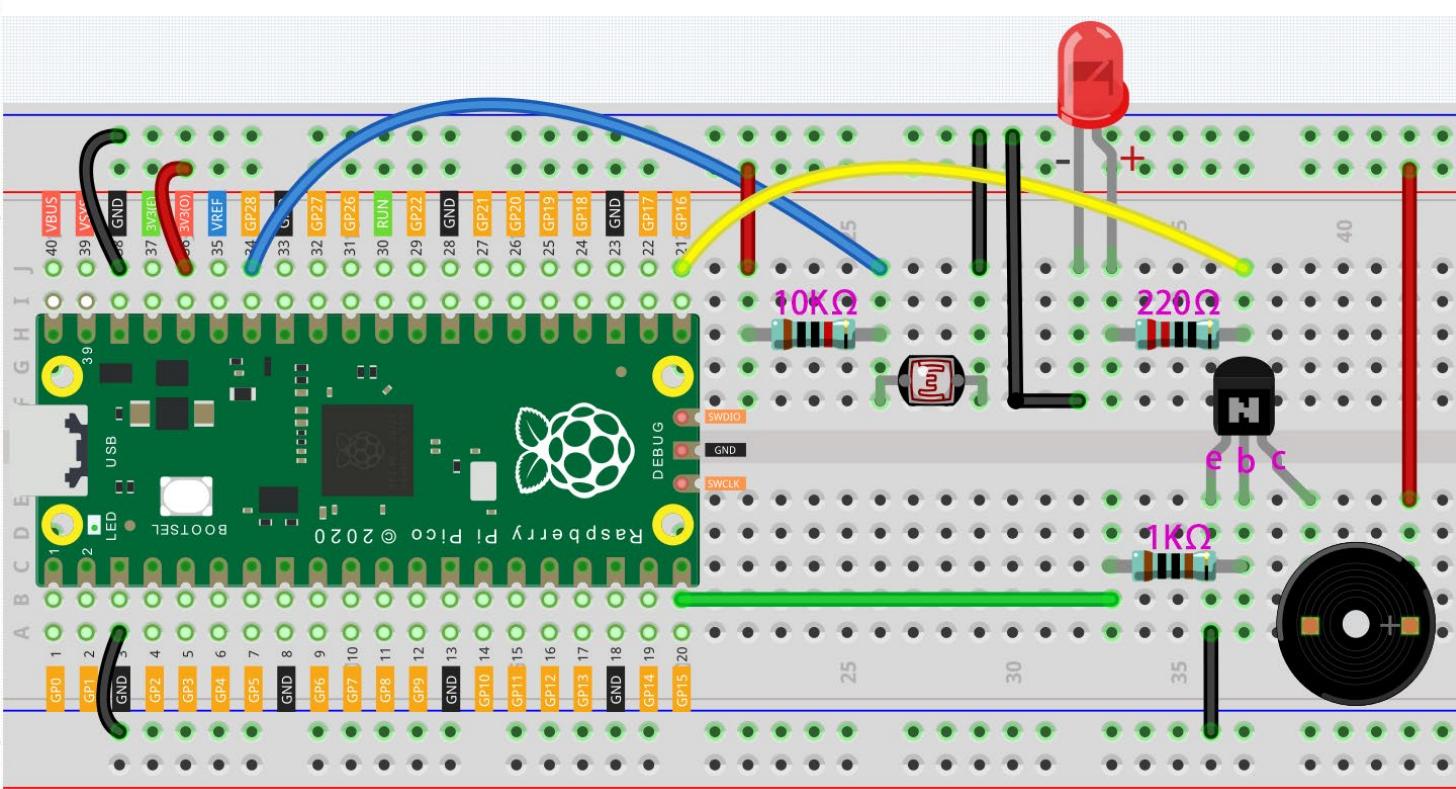
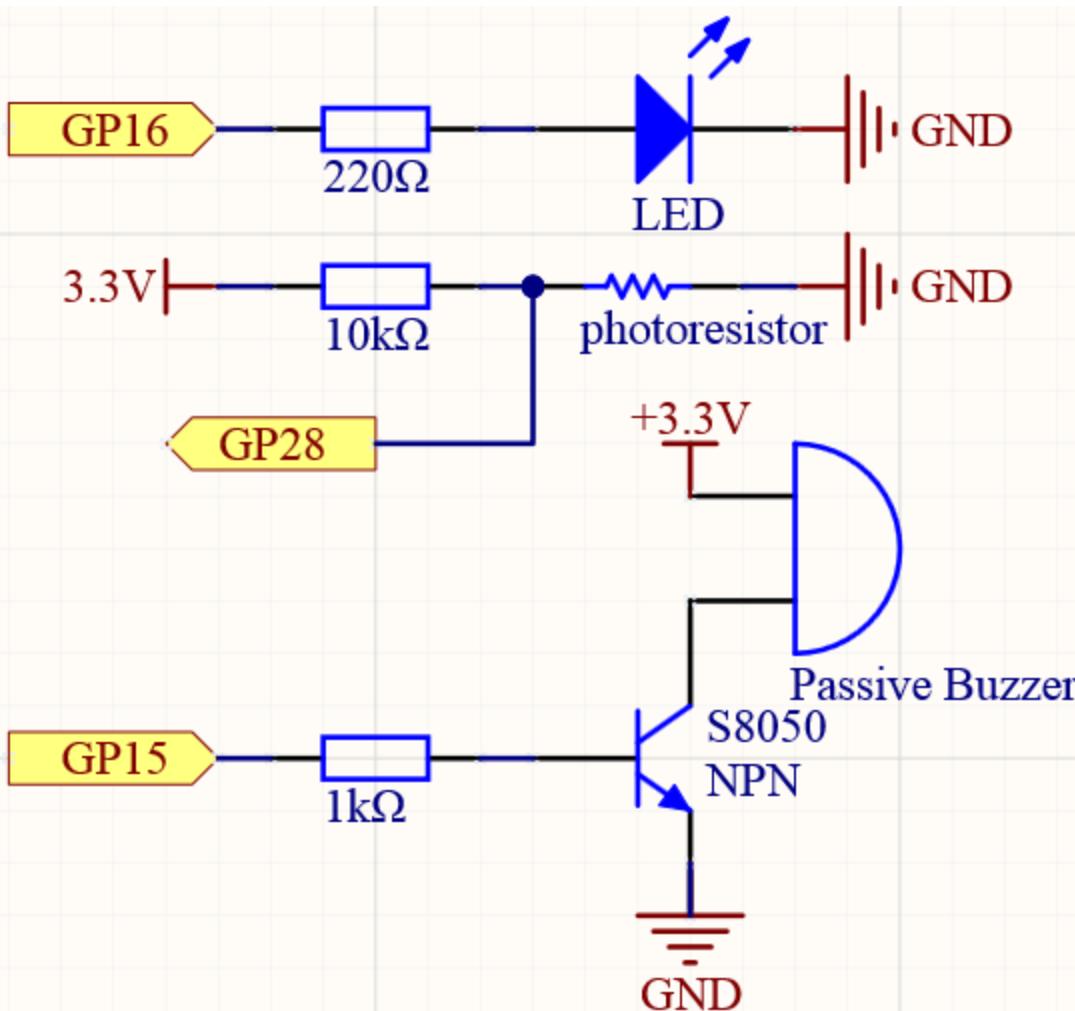
**Complete it  
in THONNY**

```
light: 99.19001%
light: 99.19001%
light: 0.78%
light: 0.78%
```

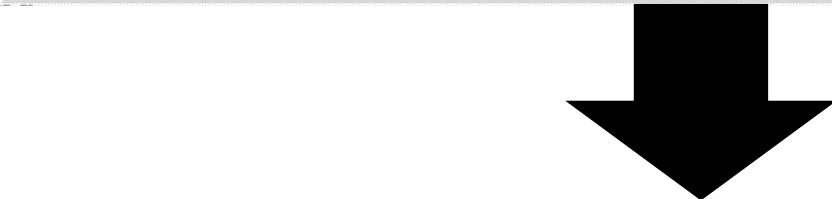
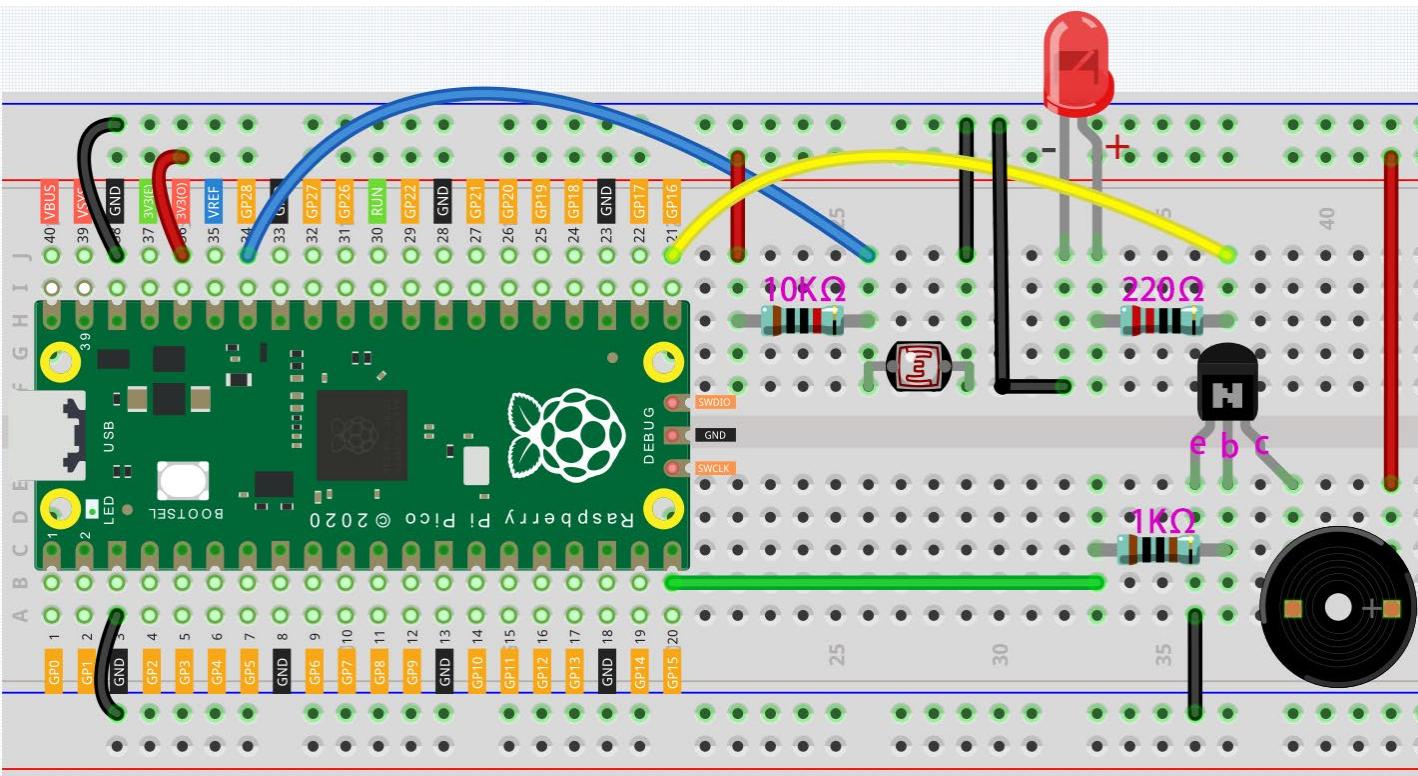
# Light Theremin: Electronic musical instrument

- 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.

# Light Theremin: Electronic musical instrument



# Light Theremin: Electronic musical instrument



Five seconds to calibrate the detection range of the photoresistor

```
from machine import Pin, PWM, ADC
import utime

led = Pin(16, Pin.OUT)
photoresistor = ADC(28)
buzzer = PWM(Pin(15))

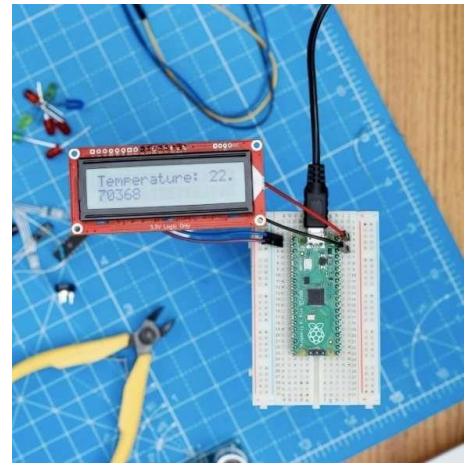
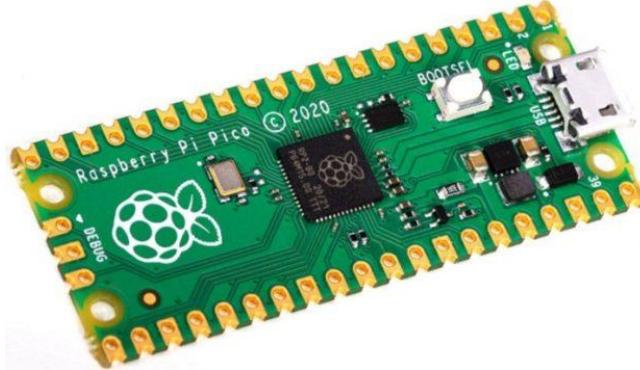
light_low=65535
light_high=0

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

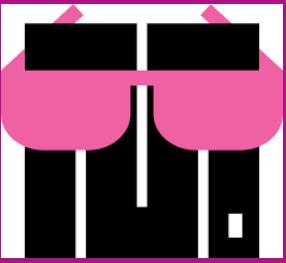
def tone(pin,frequency,duration):
    pin.freq(frequency)
    pin.duty_u16(30000)
    utime.sleep_ms(duration)
    pin.duty_u16(0)

# 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)
```

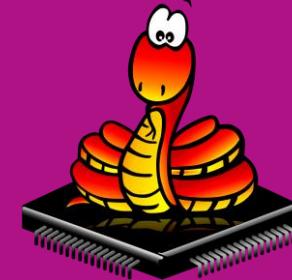


# INTERNET OF THINGS (IOT) PROJECTS USING PYTHON

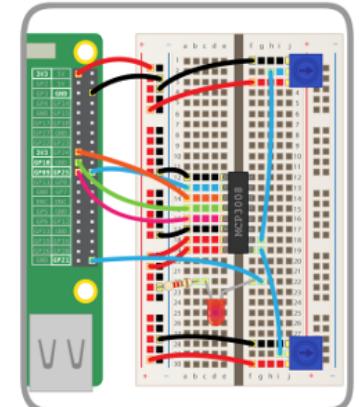
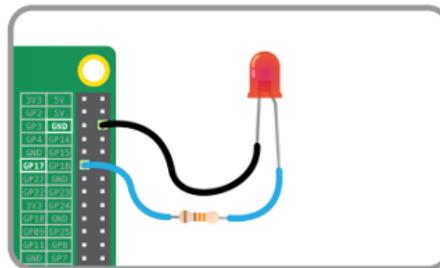
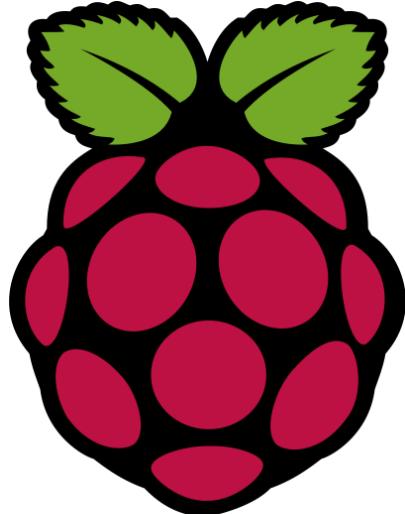


(CSE 4110)

(LECTURE – 7)



T<sub>h</sub>



# Programming an OLED Screen on Raspberry Pi Pico

The SSD1306 OLED display is available in both I2C & SPI Modules. But we will use the 0.96" I2C OLED Display as it requires only 2 wires for Interfacing. The Raspberry Pi Pico has two Pairs of I2C Pins. We can use any of the I2C Pins of Raspberry Pi Pico for Interfacing SSD1306 OLED Display.

The MicroPython IDE requires the SSD1306 Driver Code. After writing the driver code, we can write anything and display it on OLED Display. We will display the Analog value voltage from the Potentiometer on OLED Display.

## SSD1306 OLED Display

- 0.96/1.3 inch blue OLED display module
- SPI/IIC protocols
- resolution of 128x64.



- Pin 1: GND
- Pin 2: 3.3V to 5V
- Pin 3: SCL - Serial Clock
- Pin 4: SDA - Serial Data

# OLED vs LCD (IPS) – Which is better and Why?

Pros of IPS LCD	Cons of IPS LCD
Relatively cheap and easy to manufacture	Limited contrast
Good colour accuracy	Possible backlight 'leakage'
Doesn't suffer from image burn-in	
Pros of OLED	Cons of OLED
Thinner than IPS LCD	Possibility of image burn-in
Very power efficient	Expensive to manufacture
Excellent viewing angles	
Excellent black levels	
Excellent colour gamut	

Furthermore, over OLED displays, IPS LCD has advantages. The longer life and reliability of the devices equipped with that particular kind of LCD technology are major advantages because they translate into stability and stronger usability.

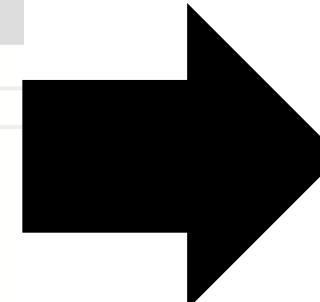
# Programming an OLED Screen on Raspberry Pi Pico

WOKWI SAVE SHARE

main.py diagram.json

```
1 print("Hello, Pi Pico!")
2
```

Rename  
Delete  
New file...  
Upload file(s)...



WOKWI SAVE SHARE

main.py diagram.json

```
1 print("Hello, Pi Pico!")
2
```

Simulation ...

Create a new file

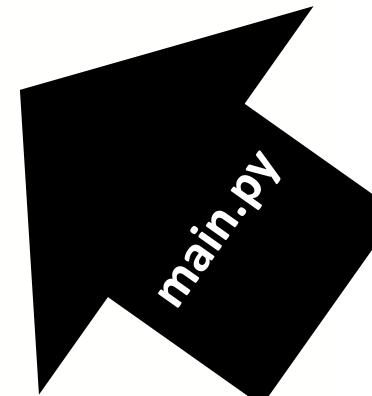
New file name: ssd1306.py

CANCEL CREATE

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C

i2c=I2C(0,sda=Pin(0), scl=Pin(1), freq=400000)
oled = SSD1306_I2C(128, 64, i2c)

oled.text(" Welcome to ",8, 0)
oled.text("IOT Projects",8, 16)
oled.text("using PYTHON",8, 32)
oled.text("(CSE - 4110)",8, 48)
oled.show()
```

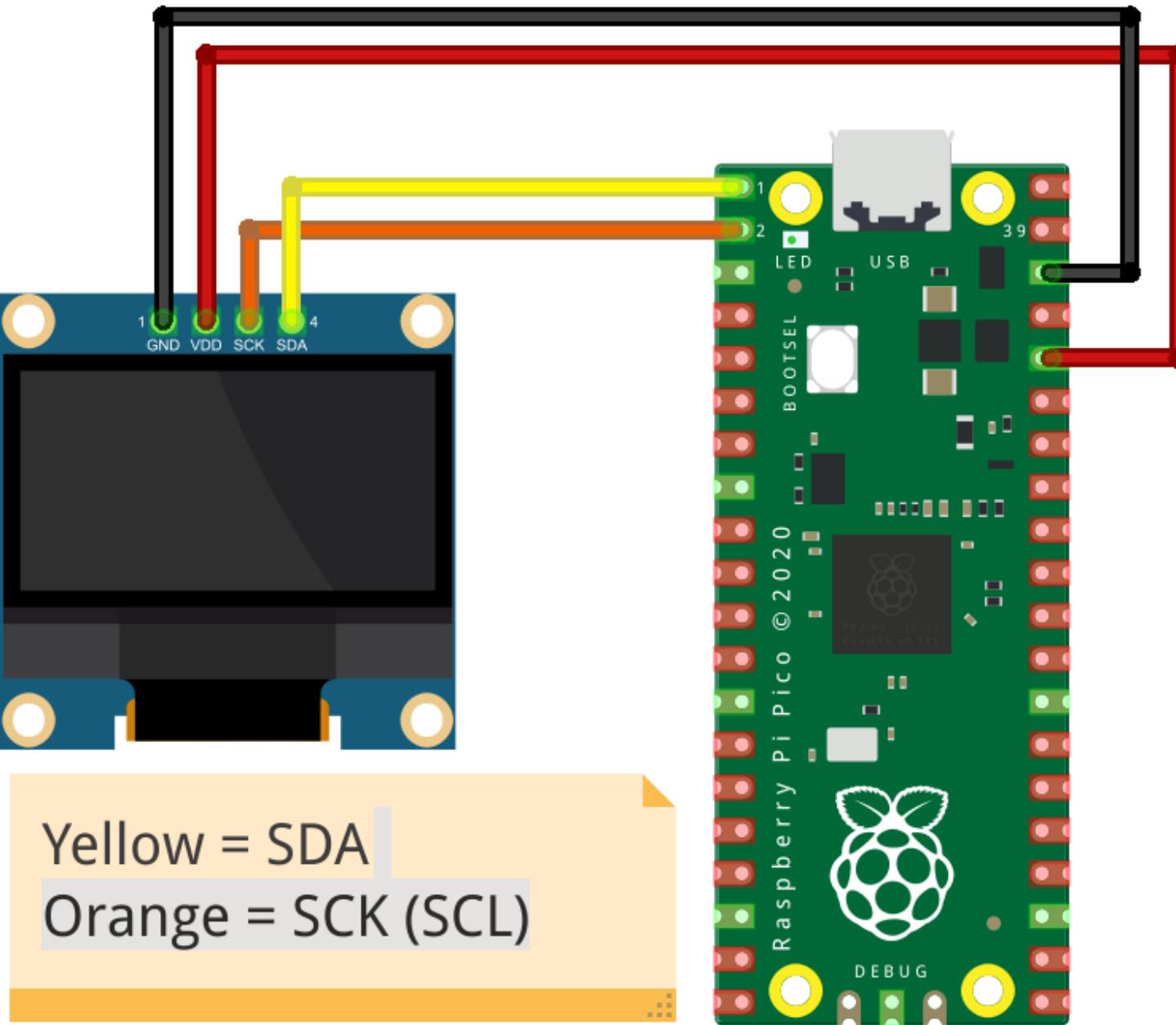


Create a  
new file  
ssd1306.py

Create ssd1306 library :

<https://wokwi.com/projects/350496872294515284>

# How to Connect an OLED screen to Raspberry Pi Pico



1. Connect the GND of the screen to any GND on the Pico (Black wire).
2. Connect VDD / VCC to 3V3 on the Pico (Red wire).
3. Connect SCK / SCL to I<sub>2</sub>C0 SCL (GP1, Physical pin 2, Orange wire).
4. Connect SDA to I<sub>2</sub>C0 SDA (GP0, Physical pin 1, Yellow wire).
5. Connect your Raspberry Pi Pico to your computer and open the Thonny application.

# Programming an OLED Screen on Raspberry Pi Pico

WOKWi<sup>o</sup> SAVE SHARE Docs B

main.py • diagram.json • ssd1306.py • PIO 🐍

```
1 from machine import Pin, I2C
2 from ssd1306 import SSD1306_I2C
3
4 i2c=I2C(0,sda=Pin(0), scl=Pin(1), freq=400000)
5 oled = SSD1306_I2C(128, 64, i2c)
6
7 oled.text(" Welcome to ",8, 0)
8 oled.text("IOT Projects",8, 16)
9 oled.text("using PYTHON",8, 32)
10 oled.text("(CSE - 4110)",8, 48)
11 oled.show()
```

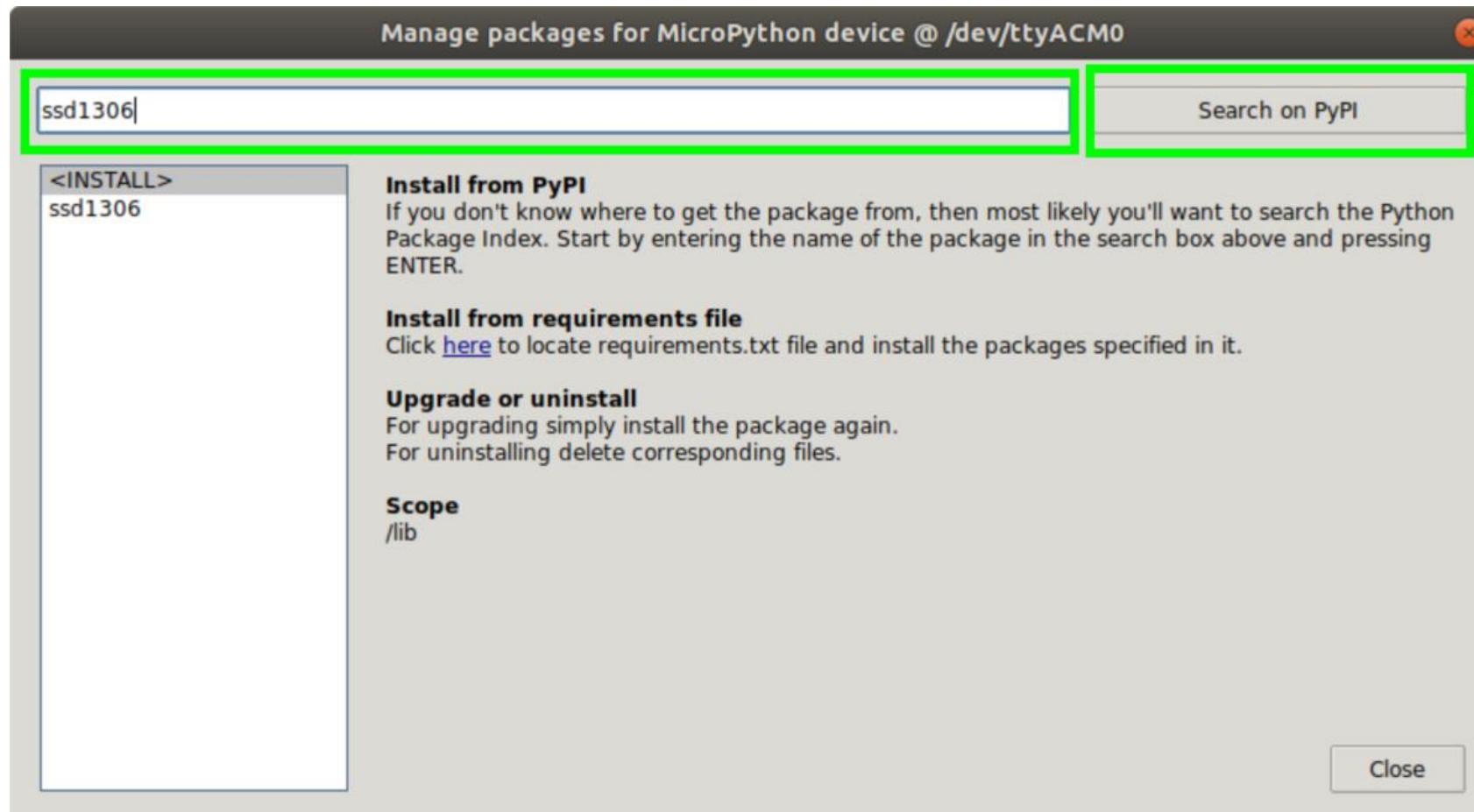
Simulation 00:46.827 99%

```
MPY: soft reboot
; Raspberry Pi Pico with RP2040
Type "help()" for more informationMicroPython v1.19.1 on 2022-06-18ion.
>>> ; Raspberry Pi Pico with RP2040
Type "help()" for more information[]
```

# How to Connect an OLED screen to THONNY

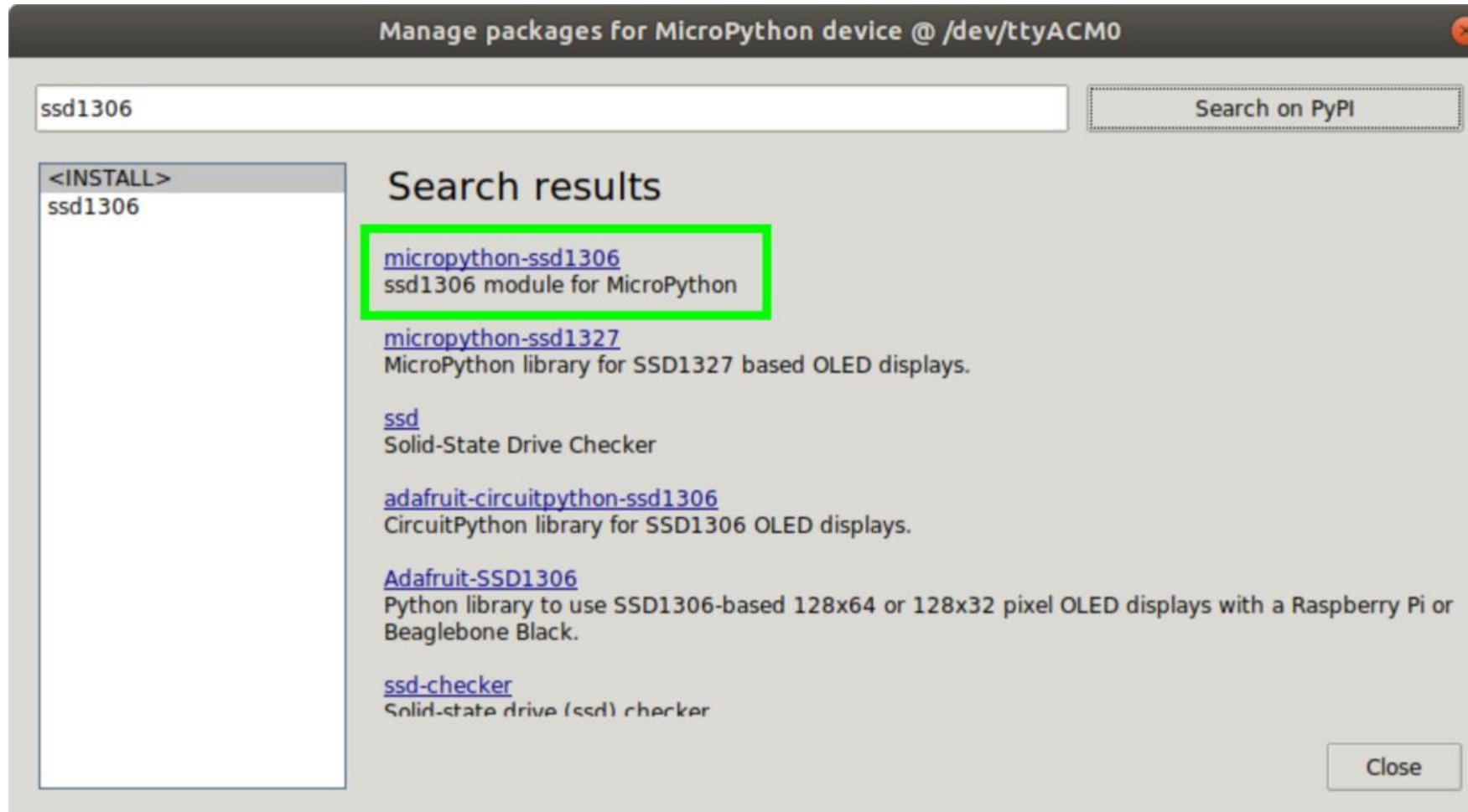
With the hardware connected and Thonny open, we now need to install a library in order for Python to communicate with the screen.

6. Click on **Tools > Manage Packages** to open Thonny's package manager for Python libraries.
7. Type “**ssd1306**” in the search bar and click “**Search on PyPI**”.



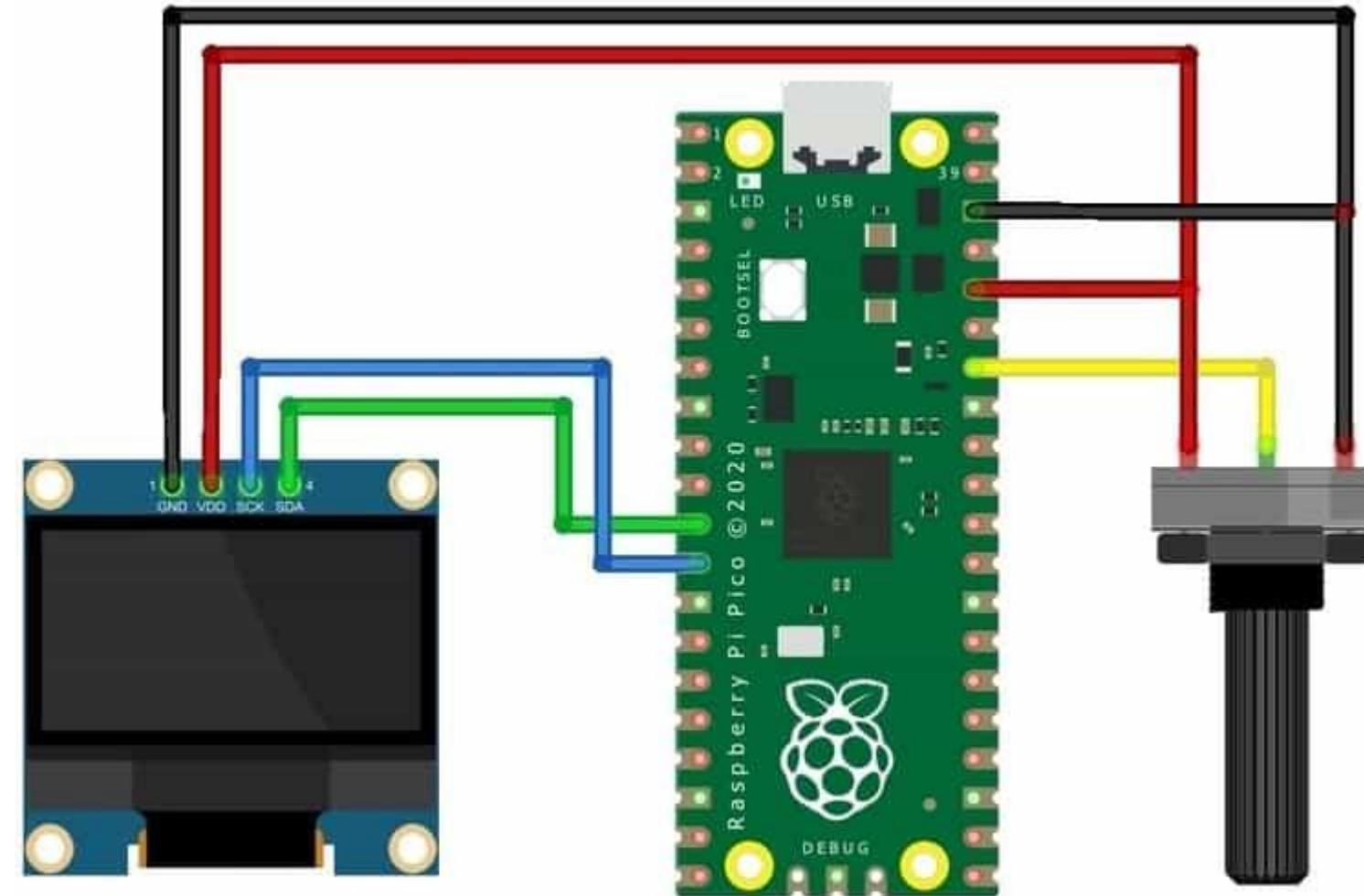
# How to Connect an OLED screen to THONNY

8. Click on “**micropython-ssd1306**” in the returned results and then click on Install. This will copy the library to a folder, lib on the Pico.



9. Click **Close** to return to the main interface.

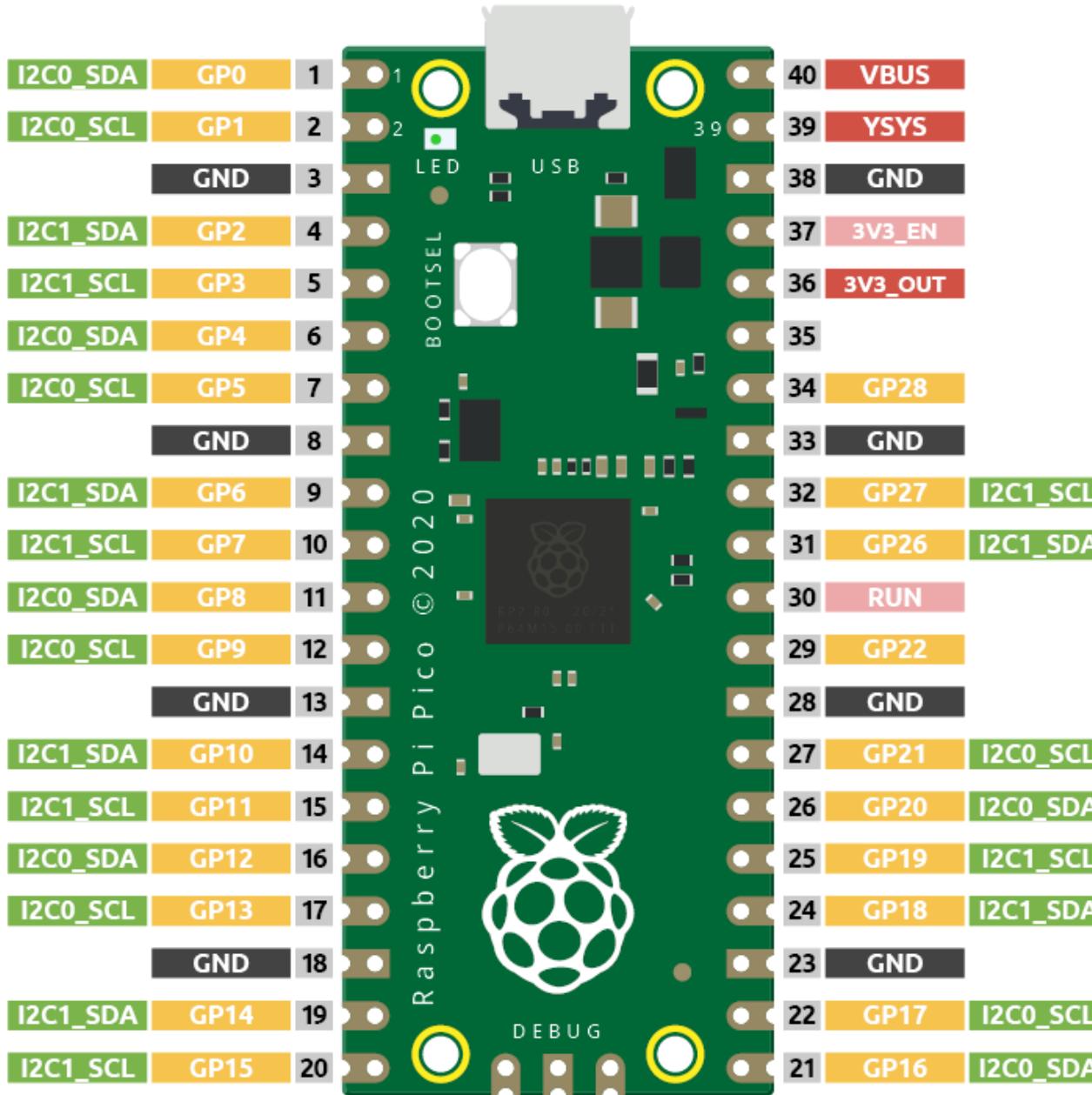
# OLED screen interfacing with POT



# Liquid Crystal Display

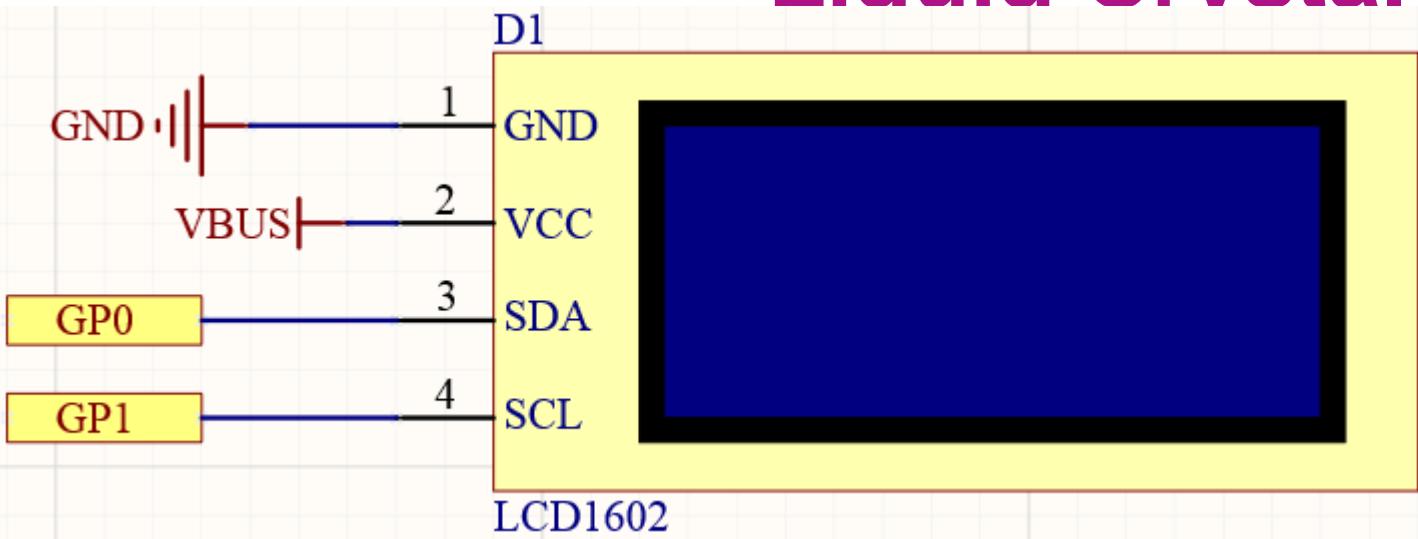
- LCD1602 is a character type liquid crystal display, which can **display 32 (16\*2) characters** at the same time.
- Though LCD and some other displays greatly enrich the man-machine interaction, they share a common weakness. When they are connected to a controller, multiple IOs will be occupied of 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.
- 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 in the next slide.

# Liquid Crystal Display

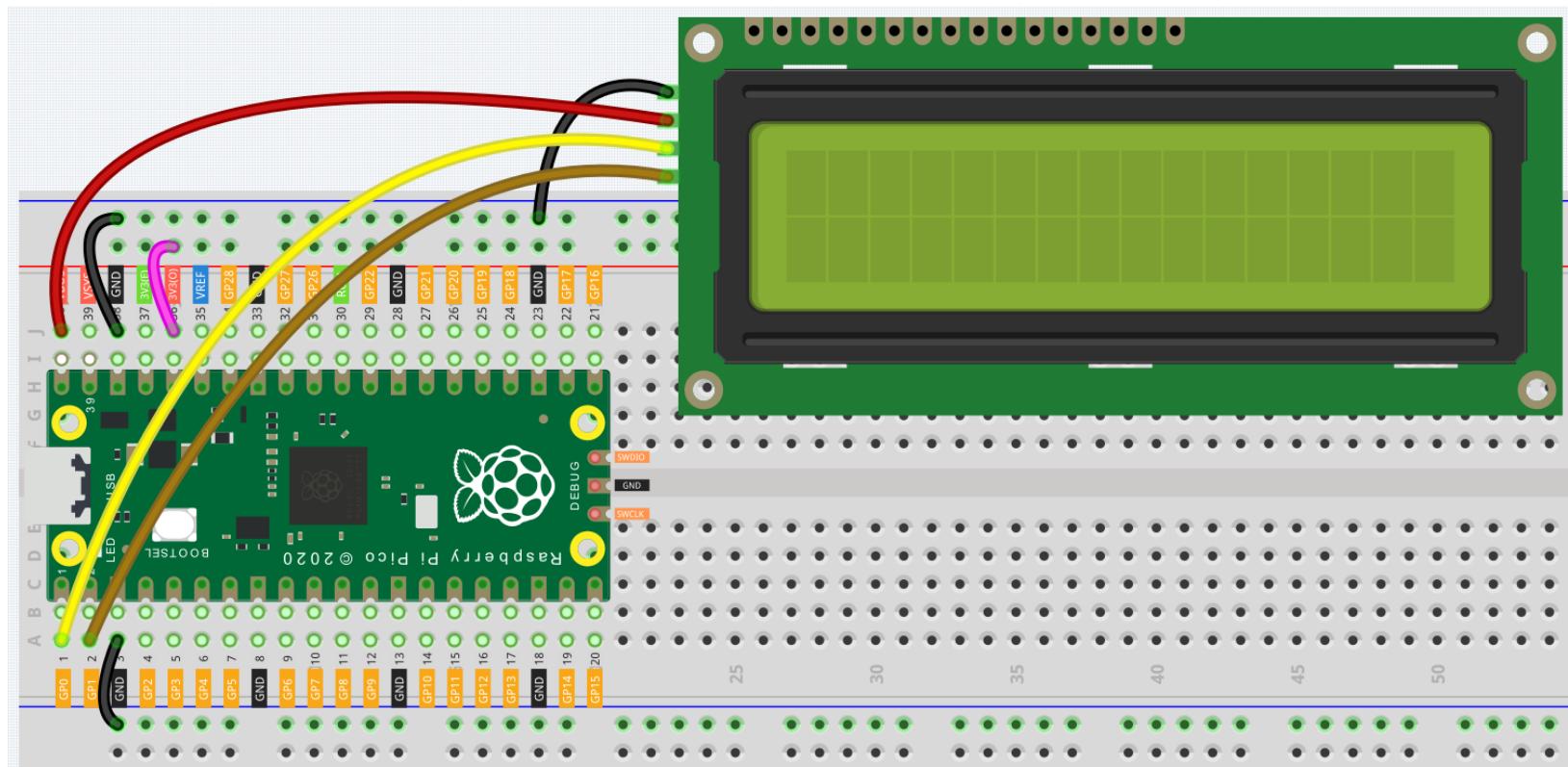


Here we will use the **I2C0** interface to control the LCD1602 and display text.

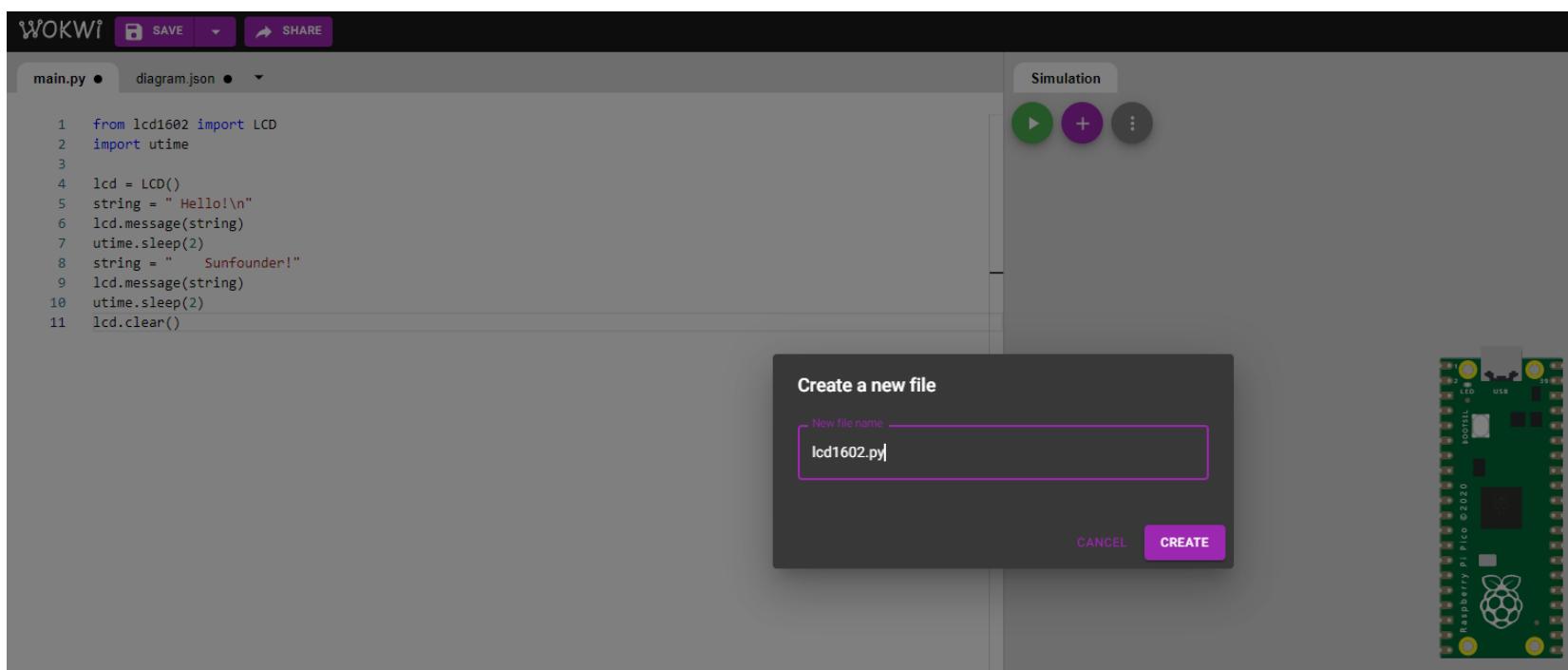
# Liauid Crvstal Dispaly



LCD 1602 with I2C



# Liquid Crystal Display



**Lcd1602.py library**

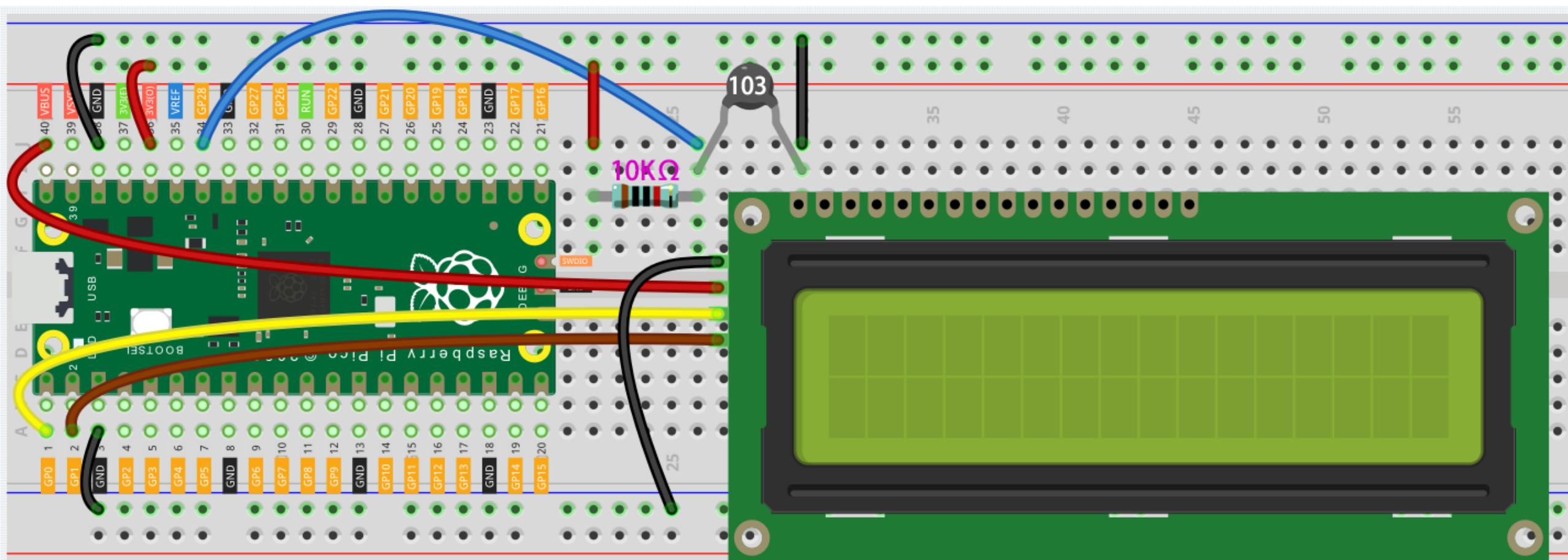
<https://wokwi.com/projects/350811004307767891>

The screenshot shows the Wokwi project interface. At the top, there are 'SAVE' and 'SHARE' buttons. Below them, tabs for 'main.py', 'diagram.json', and 'lcd1602.py' are visible. The 'lcd1602.py' tab is active, displaying the same Python code as the first screenshot. A large black upward-pointing arrow is overlaid on the right side of the screen, indicating the flow or result of the code execution.

```
1 from lcd1602 import LCD
2 import utime
3
4 lcd = LCD()
5 string = " Hello!\n"
6 lcd.message(string)
7 utime.sleep(2)
8 string = " Sunfounder!"
9 lcd.message(string)
10 utime.sleep(2)
11 lcd.clear()
```

**main.py**

# Room Température Meter



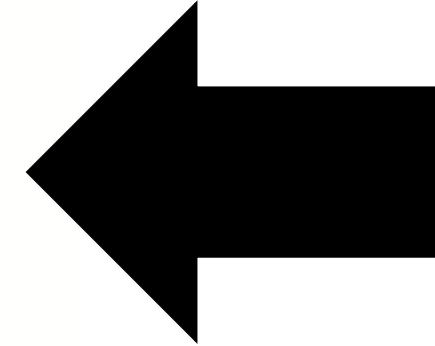
# Room Température Meter

```
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)

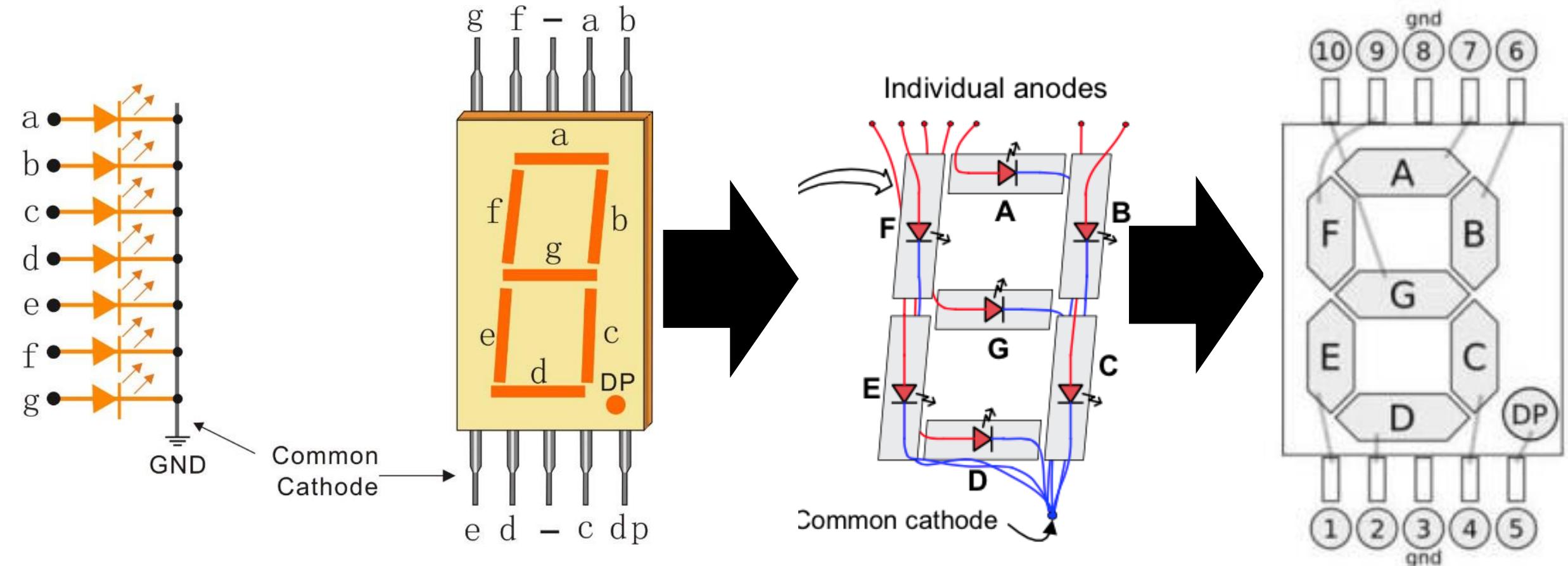
    string = " Temperature is \n      " + str('{:.2f}'.format(Cel))+ " C"
    lcd.message(string)
    utime.sleep(1)
    lcd.clear()
```



main.py

# LED Segment Display

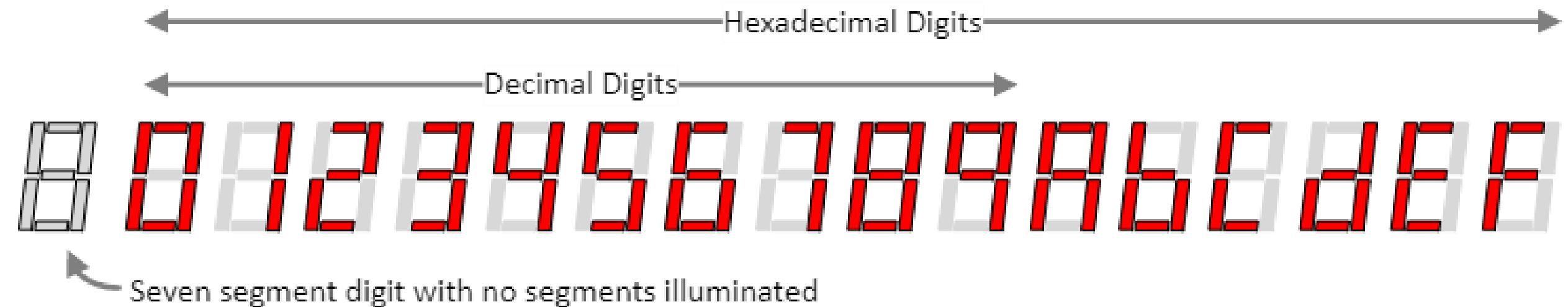
The LED Segment Display is essentially a device packaged by 8 LEDs, of which 7 strip-shaped LEDs form an “8” shape, and there is a slightly smaller dotted LED as a decimal point. These LEDs are marked as a, b, c, d, e, f, g, and dp. They have their own anode pins and share cathodes.



This means that it needs to be controlled by 8 digital signals at the same time to fully work

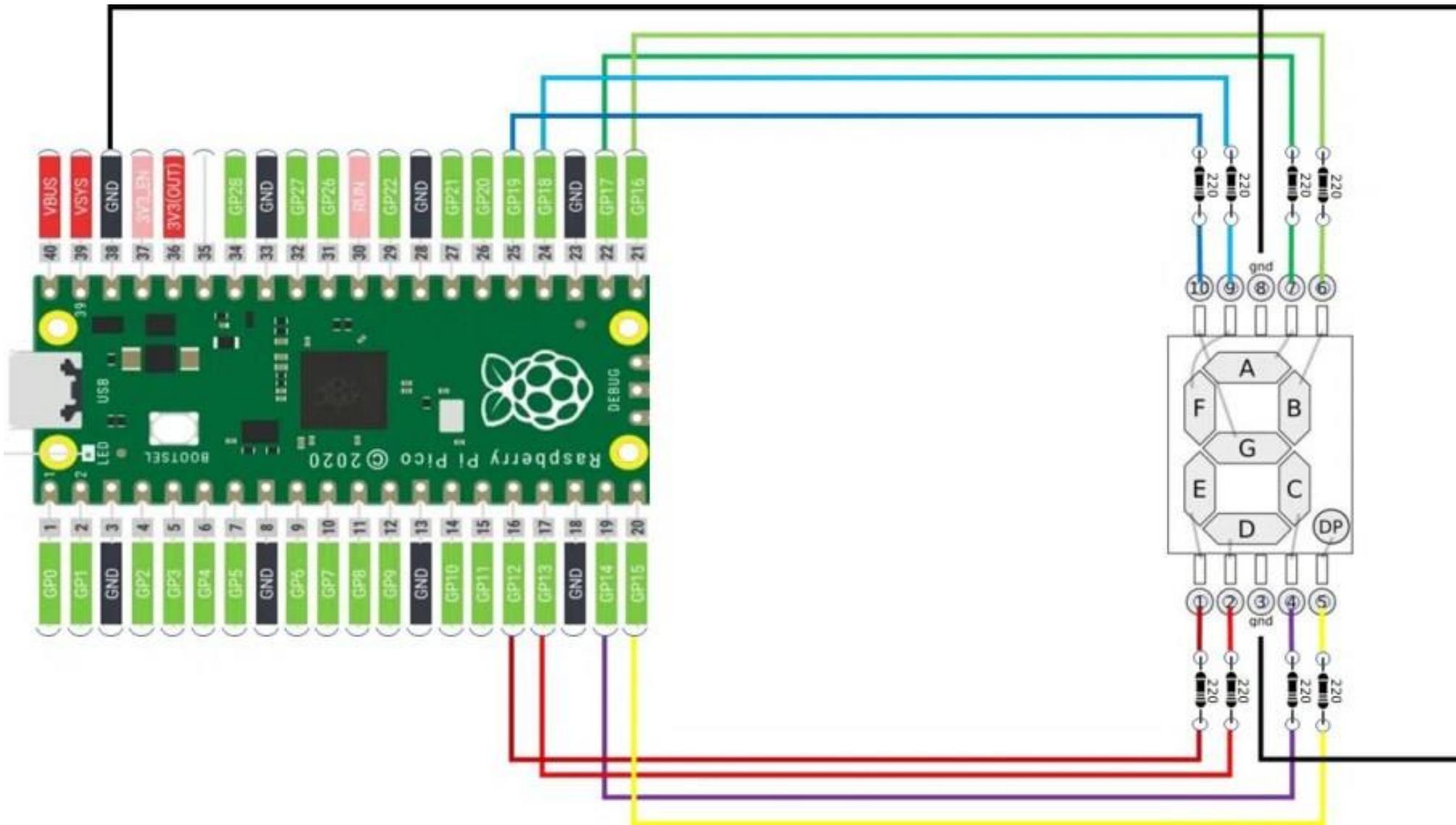
# LED Segment Display

Any LED/segment can be individually illuminated, so any one of 128 different patterns



# LED Segment Display

Any LED/segment can be individually illuminated, so any one of 128 different patterns



# LED Segment Display

WOKWI SAVE SHARE + B

Docs

main.py • diagram.json

```
1  from machine import Pin
2  from time import sleep
3
4  A_LED = Pin(17,machine.Pin.OUT) #Create an output pin for each segment
5  B_LED = Pin(16,machine.Pin.OUT)
6  DP_LED = Pin(15,machine.Pin.OUT)
7  C_LED = Pin(14,machine.Pin.OUT)
8
9  D_LED = Pin(13,machine.Pin.OUT)
10 E_LED = Pin(12,machine.Pin.OUT)
11 G_LED = Pin(19,machine.Pin.OUT)
12 F_LED = Pin(18,machine.Pin.OUT)
13
14 print("Ready, Set, GO!")
15 Seq_Del = .3
16 while True:
17     A_LED.value(1)      #Turn ON A LED segment
18     sleep(Seq_Del)    #Pause a bit
19     B_LED.value(1)
20     sleep(Seq_Del)
21     DP_LED.value(1)
22     sleep(Seq_Del)
23     C_LED.value(1)
```

Simulation

● ● ●

Raspberry Pi Pico © 2020

# LED Segment Display

WOKWI

SAVE SHARE

main.py

diagram.json

```
24     sleep(Seq_Del)
25     D_LED.value(1)
26     sleep(Seq_Del)
27     E_LED.value(1)
28     sleep(Seq_Del)
29     G_LED.value(1)
30     sleep(Seq_Del)
31     F_LED.value(1)
32     sleep(Seq_Del)
33
34     sleep(1)
35
36     A_LED.value(0)      #Turn OFF A LED segment
37     sleep(Seq_Del)      #Pause a bit
38     B_LED.value(0)
39     sleep(Seq_Del)
40     DP_LED.value(0)
41     sleep(Seq_Del)
42     C_LED.value(0)
43     sleep(Seq_Del)
44     D_LED.value(0)
45     sleep(Seq_Del)
46     E_LED.value(0)
47     sleep(Seq_Del)
48     G_LED.value(0)
49     sleep(Seq_Del)
50     F_LED.value(0)
51     sleep(Seq_Del)
52
53     sleep(1)
```

Simulation

<img alt="Circuit diagram showing a Raspberry Pi Pico connected to a 7-segment LED display. The Pico's pins 18, 22, 23, 24, 25, 26, 27, 28, 29, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 329, 330, 331, 332, 333, 334, 335, 336, 337, 337, 338, 339, 339, 340, 341, 342, 343, 344, 345, 345, 346, 347, 347, 348, 349, 349, 350, 351, 352, 353, 354, 355, 355, 356, 357, 357, 358, 359, 359, 360, 361, 362, 363, 364, 365, 365, 366, 367, 367, 368, 369, 369, 370, 371, 371, 372, 373, 373, 374, 375, 375, 376, 377, 377, 378, 379, 379, 380, 381, 381, 382, 383, 383, 384, 385, 385, 386, 387, 387, 388, 389, 389, 390, 391, 391, 392, 393, 393, 394, 395, 395, 396, 397, 397, 398, 399, 399, 400, 401, 401, 402, 403, 403, 404, 405, 405, 406, 407, 407, 408, 409, 409, 410, 411, 411, 412, 413, 413, 414, 415, 415, 416, 417, 417, 418, 419, 419, 420, 421, 421, 422, 423, 423, 424, 425, 425, 426, 427, 427, 428, 429, 429, 430, 431, 431, 432, 433, 433, 434, 435, 435, 436, 437, 437, 438, 439, 439, 440, 441, 441, 442, 443, 443, 444, 445, 445, 446, 447, 447, 448, 449, 449, 450, 451, 451, 452, 453, 453, 454, 455, 455, 456, 457, 457, 458, 459, 459, 460, 461, 461, 462, 463, 463, 464, 465, 465, 466, 467, 467, 468, 469, 469, 470, 471, 471, 472, 473, 473, 474, 475, 475, 476, 477, 477, 478, 479, 479, 480, 481, 481, 482, 483, 483, 484, 485, 485, 486, 487, 487, 488, 489, 489, 490, 491, 491, 492, 493, 493, 494, 495, 495, 496, 497, 497, 498, 499, 499, 500, 501, 501, 502, 503, 503, 504, 505, 505, 506, 507, 507, 508, 509, 509, 510, 511, 511, 512, 513, 513, 514, 515, 515, 516, 517, 517, 518, 519, 519, 520, 521, 521, 522, 523, 523, 524, 525, 525, 526, 527, 527, 528, 529, 529, 530, 531, 531, 532, 533, 533, 534, 535, 535, 536, 537, 537, 538, 539, 539, 540, 541, 541, 542, 543, 543, 544, 545, 545, 546, 547, 547, 548, 549, 549, 550, 551, 551, 552, 553, 553, 554, 555, 555, 556, 557, 557, 558, 559, 559, 560, 561, 561, 562, 563, 563, 564, 565, 565, 566, 567, 567, 568, 569, 569, 570, 571, 571, 572, 573, 573, 574, 575, 575, 576, 577, 577, 578, 579, 579, 580, 581, 581, 582, 583, 583, 584, 585, 585, 586, 587, 587, 588, 589, 589, 590, 591, 591, 592, 593, 593, 594, 595, 595, 596, 597, 597, 598, 599, 599, 600, 601, 601, 602, 603, 603, 604, 605, 605, 606, 607, 607, 608, 609, 609, 610, 611, 611, 612, 613, 613, 614, 615, 615, 616, 617, 617, 618, 619, 619, 620, 621, 621, 622, 623, 623, 624, 625, 625, 626, 627, 627, 628, 629, 629, 630, 631, 631, 632, 633, 633, 634, 635, 635, 636, 637, 637, 638, 639, 639, 640, 641, 641, 642, 643, 643, 644, 645, 645, 646, 647, 647, 648, 649, 649, 650, 651, 651, 652, 653, 653, 654, 655, 655, 656, 657, 657, 658, 659, 659, 660, 661, 661, 662, 663, 663, 664, 665, 665, 666, 667, 667, 668, 669, 669, 670, 671, 671, 672, 673, 673, 674, 675, 675, 676, 677, 677, 678, 679, 679, 680, 681, 681, 682, 683, 683, 684, 685, 685, 686, 687, 687, 688, 689, 689, 690, 691, 691, 692, 693, 693, 694, 695, 695, 696, 697, 697, 698, 699, 699, 700, 701, 701, 702, 703, 703, 704, 705, 705, 706, 707, 707, 708, 709, 709, 710, 711, 711, 712, 713, 713, 714, 715, 715, 716, 717, 717, 718, 719, 719, 720, 721, 721, 722, 723, 723, 724, 725, 725, 726, 727, 727, 728, 729, 729, 730, 731, 731, 732, 733, 733, 734, 735, 735, 736, 737, 737, 738, 739, 739, 740, 741, 741, 742, 743, 743, 744, 745, 745, 746, 747, 747, 748, 749, 749, 750, 751, 751, 752, 753, 753, 754, 755, 755, 756, 757, 757, 758, 759, 759, 760, 761, 761, 762, 763, 763, 764, 765, 765, 766, 767, 767, 768, 769, 769, 770, 771, 771, 772, 773, 773, 774, 775, 775, 776, 777, 777, 778, 779, 779, 780, 781, 781, 782, 783, 783, 784, 785, 785, 786, 787, 787, 788, 789, 789, 790, 791, 791, 792, 793, 793, 794, 795, 795, 796, 797, 797, 798, 799, 799, 800, 801, 801, 802, 803, 803, 804, 805, 805, 806, 807, 807, 808, 809, 809, 810, 811, 811, 812, 813, 813, 814, 815, 815, 816, 817, 817, 818, 819, 819, 820, 821, 821, 822, 823, 823, 824, 825, 825, 826, 827, 827, 828, 829, 829, 830, 831, 831, 832, 833, 833, 834, 835, 835, 836, 837, 837, 838, 839, 839, 840, 841, 841, 842, 843, 843, 844, 845, 845, 846, 847, 847, 848, 849, 849, 850, 851, 851, 852, 853, 853, 854, 855, 855, 856, 857, 857, 858, 859, 859, 860, 861, 861, 862, 863, 863, 864, 865, 865, 866, 867, 867, 868, 869, 869, 870, 871, 871, 872, 873, 873, 874, 875, 875, 876, 877, 877, 878, 879, 879, 880, 881, 881, 882, 883, 883, 884, 885, 885, 886, 887, 887, 888, 889, 889, 890, 891, 891, 892, 893, 893, 894, 895, 895, 896, 897, 897, 898, 899, 899, 900, 901, 901, 902, 903, 903, 904, 905, 905, 906, 907, 907, 908, 909, 909, 910, 911, 911, 912, 913, 913, 914, 915, 915, 916, 917, 917, 918, 919, 919, 920, 921, 921, 922, 923, 923, 924, 925, 925, 926, 927, 927, 928, 929, 929, 930, 931, 931, 932, 933, 933, 934, 935, 935, 936, 937, 937, 938, 939, 939, 940, 941, 941, 942, 943, 943, 944, 945, 945, 946, 947, 947, 948, 949, 949, 950, 951, 951, 952, 953, 953, 954, 955, 955, 956, 957, 957, 958, 959, 959, 960, 961, 961, 962, 963, 963, 964, 965, 965, 966, 967, 967, 968, 969, 969, 970, 971, 971, 972, 973, 973, 974, 975, 975, 976, 977, 977, 978, 979, 979, 980, 981, 981, 982, 983, 983, 984, 985, 985, 986, 987, 987, 988, 989, 989, 990, 991, 991, 992, 993, 993, 994, 995, 995, 996, 997, 997, 998, 999, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 10010, 10011, 10012, 10013, 10014, 10015, 10016, 10017, 10018, 10019, 10020, 10021, 10022, 10023, 10024, 10025, 10026, 10027, 10028, 10029, 10030, 10031, 10032, 10033, 10034, 10035, 10036, 10037, 10038, 10039, 100310, 100311, 100312, 100313, 100314, 100315, 100316, 100317, 100318, 100319, 100320, 100321, 100322, 100323, 100324, 100325, 100326, 100327, 100328, 100329, 100330, 100331, 100332, 100333, 100334, 100335, 100336, 100337, 100338, 100339, 1003310, 1003311, 1003312, 1003313, 1003314, 1003315, 1003316, 1003317, 1003318, 1003319, 1003320, 1003321, 1003322, 1003323, 1003324, 1003325, 1003326, 1003327, 1003328, 1003329, 1003330, 1003331, 1003332, 1003333, 1003334, 1003335, 1003336, 1003337, 1003338, 1003339, 10033310, 10033311, 10033312, 10033313, 10033314, 10033315, 10033316, 10033317, 10033318, 10033319, 10033320, 10033321, 10033322, 10033323, 10033324, 10033325, 10033326, 10033327, 10033328, 10033329, 10033330, 10033331, 10033332, 10033333, 10033334, 10033335, 10033336, 10033337, 10033338, 10033339, 100333310, 100333311, 100333312, 100333313, 100333314, 100333315, 100333316, 100333317, 100333318, 100333319, 100333320, 100333321, 100333322, 100333323, 100333324, 100333325, 100333326, 100333327, 100333328, 100333329, 100333330, 100333331, 100333332, 100333333, 100333334, 100333335, 100333336, 100333337, 100333338, 100333339, 1003333310, 1003333311, 1003333312, 1003333313, 1003333314, 1003333315, 1003333316, 1003333317, 1003333318, 1003333319, 1003333320, 1003333321, 1003333322, 1003333323, 1003333324, 1003333325, 1003333326, 1003333327, 1003333328, 1003333329, 1003333330, 1003333331, 1003333332, 1003333333, 1003333334, 1003333335, 1003333336, 1003333337, 1003333338, 1003333339, 10033333310, 10033333311, 10033333312, 10033333313, 10033333314, 10033333315, 10033333316, 10033333317, 10033333318, 10033333319, 10033333320, 10033333321, 10033333322, 10033333323, 10033333324, 10033333325, 10033333326, 10033333327, 10033333328, 10033333329, 10033333330, 10033333331, 10033333332, 10033333333, 10033333334, 10033333335, 10033333336, 10033333337, 10033333338, 10033333339, 100333333310, 100333333311, 100333333312, 100333333313, 100333333314, 100333333315, 100333333316, 100333333317, 100333333318, 100333333319, 100333333320, 100333333321, 100333333322, 100333333323, 100333333324, 100333333325, 100333333326, 100333333327, 100333333328, 100333333329, 100333333330, 100333333331, 100333333332, 100333333333, 100333333334, 100333333335, 100333333336, 100333333337, 100333333338, 100333333339, 1003333333310, 1003333333311, 1003333333312, 1003333333313, 1003333333314, 1003333333315, 1003333333316, 1003333333317, 1003333333318, 1003333333319, 1003333333320, 1003333333321, 1003333333322, 1003333333323, 1003333333324, 1003333333325, 1003333333326, 1003333333327, 1003333333328, 1003333333329, 1003333333330, 1003333333331, 1003333333332, 1003333333333, 1003333333334, 1003333333335, 1003333333336, 1003333333337, 1003333333338, 1003333333339, 10033333333310, 10033333333311, 10033333333312, 10033333333313, 10033333333314, 10033333333315, 10033333333316, 10033333333317, 10033333333318, 10033333333319, 10033333333320, 10033333333321, 10033333333322, 10033333333323, 10033333333324, 10033333333325, 10033333333326, 10033333333327, 10033333333328, 10033333333329, 10033333333330, 10033333333331, 10033333333332, 10033333333333, 10033333333334, 10033333333335, 10033333333336, 10033333333337, 10033333333338, 10033333333339, 100333333333310, 100333333333311, 100333333333312, 100333333333313, 100333333333314, 100333333333315, 100333333333316, 100333333333317, 100333333333318, 100333333333319, 100333333333320, 100333333333321, 100333333333322, 100333333333323, 100333333333324, 100333333333325, 100333333333326, 100333333333327, 100333333333328, 100333333333329, 100333333333330, 100333333333331, 100333333333332, 100333333333333, 100333333333334, 100333333333335, 100333333333336, 100333333333337, 100333333333338, 100333333333339, 1003333333333310, 1003333333333311, 1003333333333312, 1003333333333313, 1003333333333314, 1003333333333315, 1003333333333316, 1003333333333317, 1003333333333318, 1003333333333319, 1003333333333320, 1003333333333321, 1003333333333322, 1003

# LED Segment Display

```
from machine import Pin

# define GP ports to use
# Pins Matching: A, B, C, D, E, F, G
display_list = [17,16,14,13,12,18,19]
dotPin=15
display_obj = []

# Set all pins as output
for seg in display_list:
    display_obj.append(Pin(seg, Pin.OUT))

dot_obj=Pin(dotPin, Pin.OUT)

# DIGIT map as array of array
arrSeg = [[1,1,1,1,1,1,0],\
           [0,1,1,0,0,0,0],\
           [1,1,0,1,1,0,1],\
           [1,1,1,1,0,0,1],\
           [0,1,1,0,0,1,1],\
           [1,0,1,1,0,1,1],\
           [1,0,1,1,1,1,1],\
           [1,1,1,0,0,0,0],\
           [1,1,1,1,1,1,1],\
           [1,1,1,1,0,1,1]]
```

```
def SegDisplay(toDisplay):
    numDisplay = int(toDisplay.replace(".", ""))
    for a in range(7):
        display_obj[a].value(arrSeg[numDisplay][a])
    # Manage DOT activation
    if toDisplay.count(".") == 1:
        dot_obj.value(1)
    else:
        dot_obj.value(0)
```

SegDisplay("5.")



If we don't want the dot, simply call in this way:  
**SegDisplay("5")**

The number can vary from 0 to 9 and the parameter needs to be a string. If you have integer numbers, you can use the str() function to convert them. An example:

**SegDisplay(str(5))**

# Displays number 0-9 with or without the decimal point

WOKWI SAVE SHARE + B

Docs

main.py • diagram.json

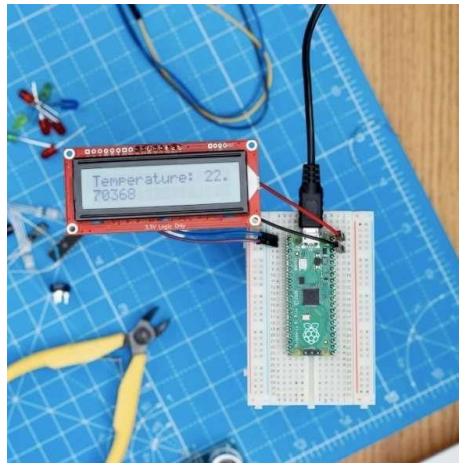
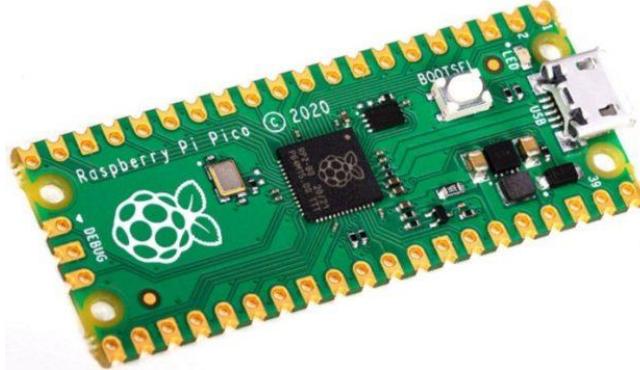
```
1 import machine
2 import utime
3
4 A_LED = machine.Pin(17,machine.Pin.OUT) #Create an output pin for each segment
5 B_LED = machine.Pin(16,machine.Pin.OUT)
6 DP_LED = machine.Pin(15,machine.Pin.OUT)
7 C_LED = machine.Pin(14,machine.Pin.OUT)
8 D_LED = machine.Pin(13,machine.Pin.OUT)
9 E_LED = machine.Pin(12,machine.Pin.OUT)
10 G_LED = machine.Pin(19,machine.Pin.OUT)
11 F_LED = machine.Pin(18,machine.Pin.OUT)
12
13 def Clear_Display():
14     A_LED.value(0)      #Turn OFF all LED segments
15     B_LED.value(0)
16     DP_LED.value(0)
17     C_LED.value(0)
18     D_LED.value(0)
19     E_LED.value(0)
20     G_LED.value(0)
21     F_LED.value(0)
22
23 def Show_Number(Number, DecimalPoint):
24     Clear_Display()
25     if Number == "0":
26         A_LED.value(1)      #Turn ON needed LED segments
```

Simulation

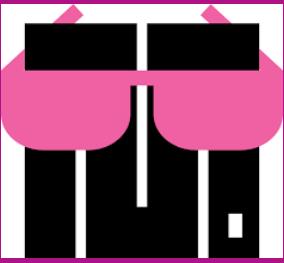
Raspberry Pi Pico © 2020

# Displays number 0-9 with or without the decimal point

```
27     B_LED.value(1)      53     A_LED.value(1)      79     B_LED.value(1)
28     C_LED.value(1)      54     F_LED.value(1)      80     C_LED.value(1)
29     D_LED.value(1)      55     G_LED.value(1)      81     G_LED.value(1)
30     E_LED.value(1)      56     C_LED.value(1)      82     F_LED.value(1)
31     F_LED.value(1)      57     D_LED.value(1)      83     if DecimalPoint == ".":
32     elif Number == "1":  58     elif Number == "6":  84     DP_LED.value(1)
33         B_LED.value(1)  59         A_LED.value(1)  85
34         C_LED.value(1)  60         F_LED.value(1)  86     print("Ready, Set, GO!")
35     elif Number == "2":  61         E_LED.value(1)  87
36         A_LED.value(1)  62         D_LED.value(1)  88     Seq_Del = 1
37         B_LED.value(1)  63         C_LED.value(1)  89     while True:
38         G_LED.value(1)  64         G_LED.value(1)  90     Show_Number("0","",)
39         E_LED.value(1)  65     elif Number == "7":  91     utime.sleep(Seq_Del)
40         D_LED.value(1)  66         A_LED.value(1)  92     Show_Number("1","",)
41     elif Number == "3":  67         B_LED.value(1)  93     utime.sleep(Seq_Del)
42         A_LED.value(1)  68         C_LED.value(1)  94     Show_Number("2","",)
43         B_LED.value(1)  69     elif Number == "8":  95     utime.sleep(Seq_Del)
44         G_LED.value(1)  70         A_LED.value(1)  96     Show_Number("3","",)
45         C_LED.value(1)  71         B_LED.value(1)  97     utime.sleep(Seq_Del)
46         D_LED.value(1)  72         C_LED.value(1)  98     Show_Number("4","",)
47     elif Number == "4":  73         D_LED.value(1)  99     utime.sleep(Seq_Del)
48         B_LED.value(1)  74         E_LED.value(1) 100    Show_Number("5","",)
49         G_LED.value(1)  75         G_LED.value(1) 101    utime.sleep(Seq_Del)
50         C_LED.value(1)  76         F_LED.value(1) 102    Show_Number("6","",)
51         F_LED.value(1)  77     elif Number == "9": 103    utime.sleep(Seq_Del)
52     elif Number == "5":  78         A_LED.value(1) 104    Show_Number("7","",)
53         -           -       79         -           -       105    utime.sleep(Seq_Del)
54         -           -       80         -           -       106    Show_Number("8","",)
55         -           -       81         -           -       107    utime.sleep(Seq_Del)
56         -           -       82         -           -       108    Show_Number("9","",)
57         -           -       83         -           -       109    utime.sleep(Seq_Del)
58         -           -       84         -           -       110    Show_Number("9",".")
59         -           -       85         -           -       111    utime.sleep(Seq_Del)
60         -           -       86         -           -       112
61         -           -       87         -           -       113    utime.sleep(1)
```

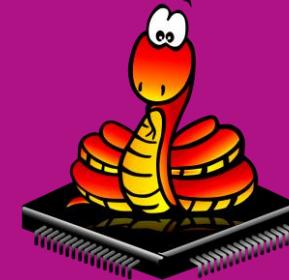


# INTERNET OF THINGS (IOT) PROJECTS USING PYTHON

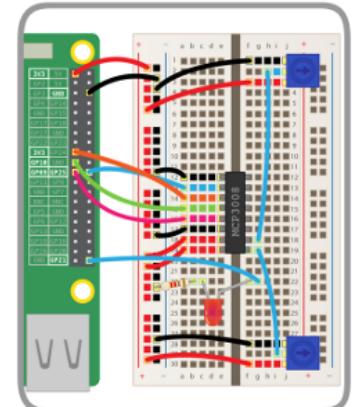
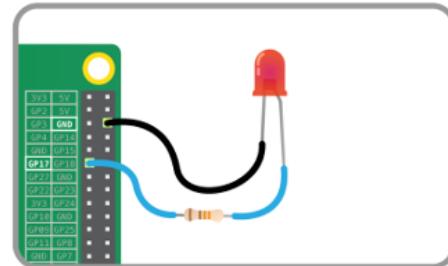
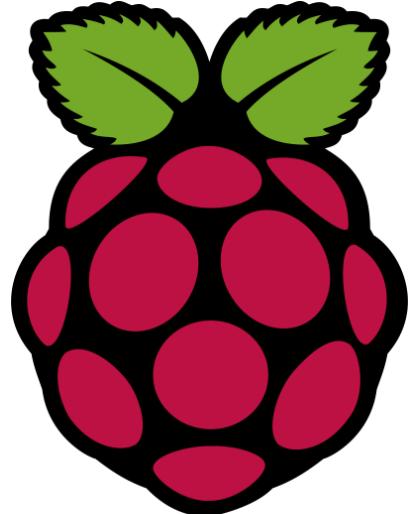


(CSE 4110)

(LECTURE – 8)



T<sub>h</sub>



# Why do we need Wi-Fi module?

Wi-Fi modules are used to **provide internet connection to robotic and electronic projects**. Wi-Fi modules allow developing IoT (Internet of Things) projects. By using Wi-Fi modules you can send data over the internet to your robot or make it send data over the internet.

WiFi module, also known as serial to WIFI module, which belongs to the transmission layer of IoT. The function is to **convert serial port or TTL level into embedded module** which can conforming to WiFi wireless network communication standard, with built-in **wireless network protocol IEEE802**.

## What does the Wi-Fi module have?

The Wi-Fi module generally contains **two main parts**: a **Wi-Fi chip** and an **application host processor**. The Wi-Fi subsystem includes an 802.11 radio physical layer (PHY), baseband, media access control (MAC), and perhaps a crypto engine for fast, secure Internet connection.

# Important Terminology

## UART Communication

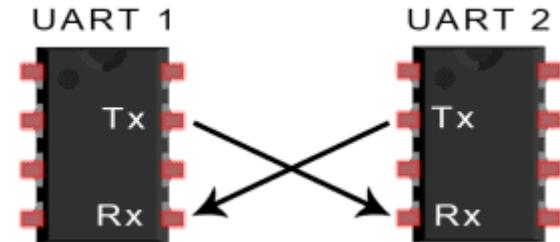
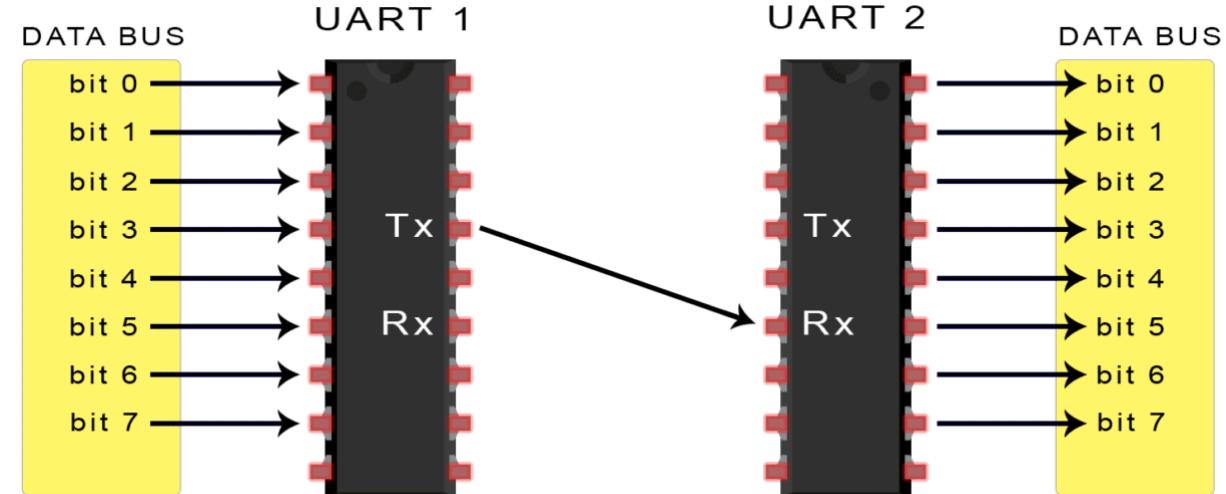
UART stands for **Universal Asynchronous Receiver/Transmitter**.

It's not a communication protocol like SPI and I2C, but a physical circuit in a microcontroller, or a stand-alone IC. A UART's main purpose is to transmit and receive serial data.

One of the best things about UART is that it only uses two wires to transmit data between devices.

In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART:

UARTs transmit data asynchronously, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits.



# Important Terminology

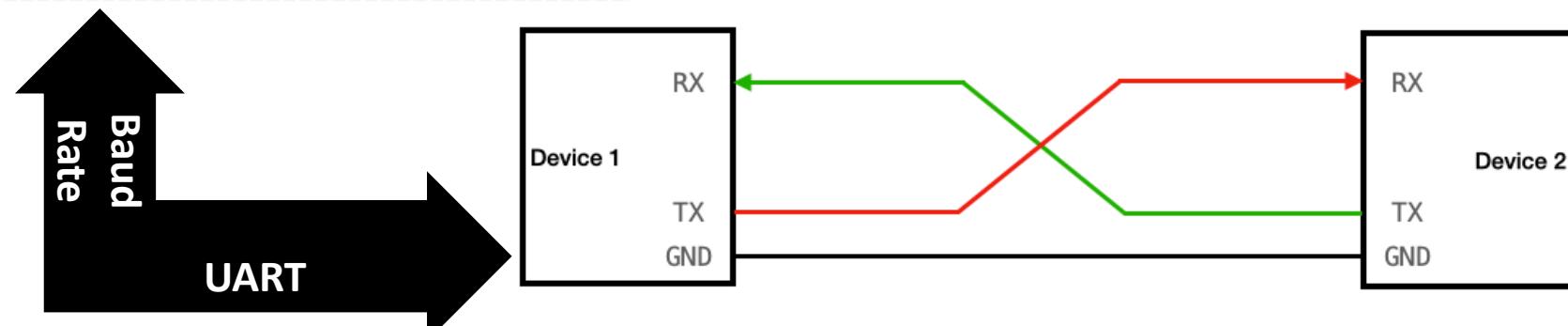
## Baud Rate

When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the baud rate. Baud rate is a measure of the **speed of data transfer**, expressed in **bits per second (bps)**. Both UARTs must operate at **about the same baud rate**. The baud rate between the transmitting and receiving UARTs can only differ by about 10% before the timing of bits gets too far off.

Both UARTs must also be configured to transmit and receive the same data packet structure.

Wires Used	2
Maximum Speed	Any speed up to 115200 baud, usually 9600 baud
Synchronous or Asynchronous?	Asynchronous
Serial or Parallel?	Serial
Max # of Masters	1
Max # of Slaves	1

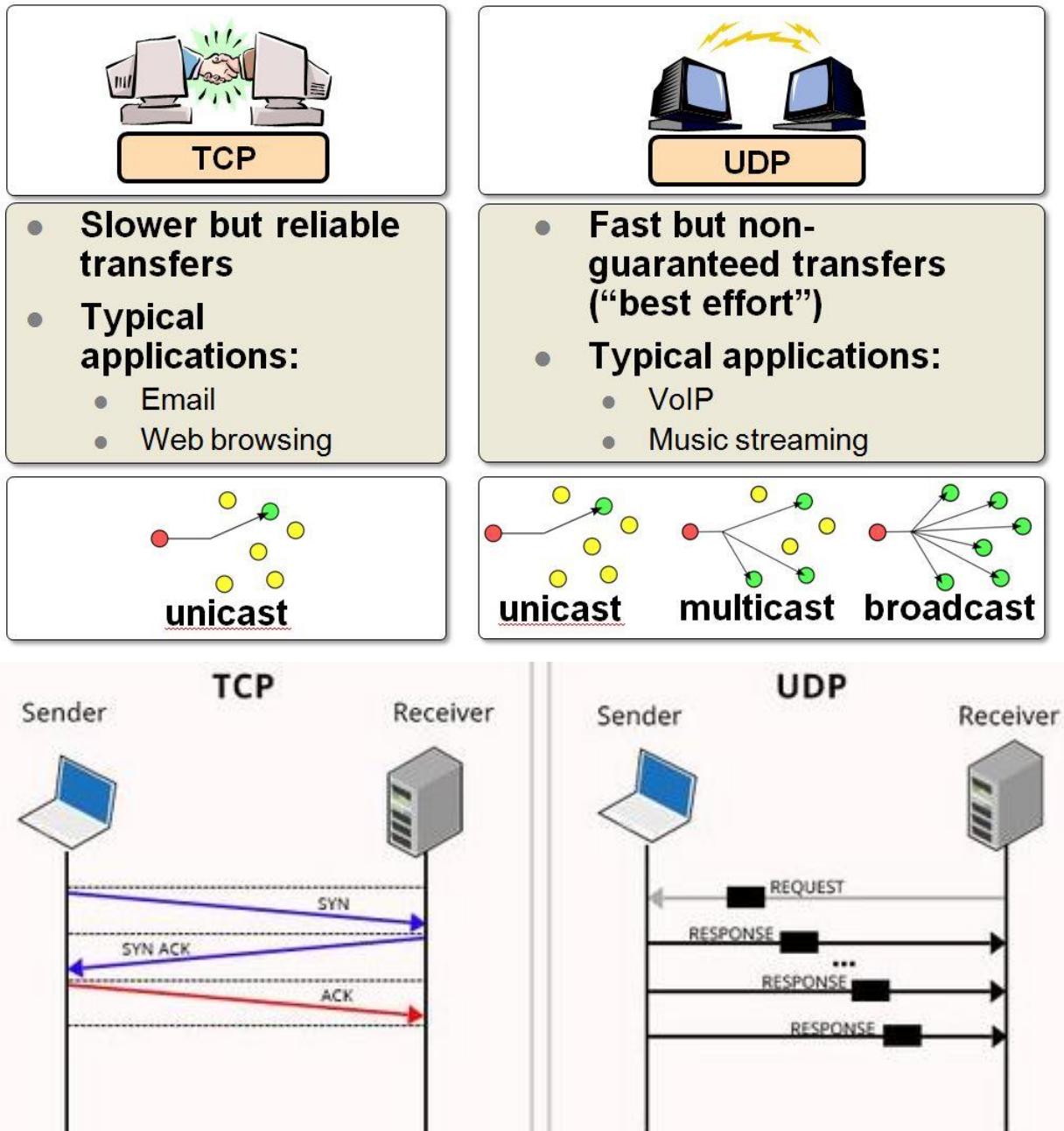
Default Baud Rates  
Esp01 : 115200  
Arduino: 9600



# Important Terminology

## TCP/IP Protocol

**Transmission Control Protocol (TCP)** widely known as Internet Protocol (IP) is a set of communications protocols used over the internet for delivery of services or packets within or across the network. It is commonly known as internet protocol suite.



# Important Terminology

## AT commands

AT commands are instructions used to control a modem. AT is the abbreviation of ATtention. Every command line starts with "AT" or "at". That's why modem commands are called AT commands.

**AT:** This type of command is used to test the startup function of WiFi module. The response would be ok, against this command if everything is ok.

**AT+GMR :** This type of AT command is used to check the version of AT command and we used SDK version of AT command in this type of WIFI module.

**AT+CIPSERVER=0:** This configures the ESP8266 as server and sets the mode as 0 which means delete server (need to follow by restart)

**AT+RST:** This type of command is used for reset the WiFi module when it is in working condition. The response would be ok, when reset the module.

**AT+RESTORE:** This type of command is used to restore factory settings means, when this command is entered then all the parameters are reset automatically to default one's.

**AT+CWMODE? :** This type of command is used to query the WiFi mode of ESP8266.

**AT+CWMODE=1 :** This sets the WiFi mode of ESP8266 in this case in station mode.

**AT+CWJAP="SSID","PASSWORD"\r\n, timeout=TIME\_ms :** This connects the ESP8266 with an AP whose SSID and password are given, The timeout here is the reconnection time.

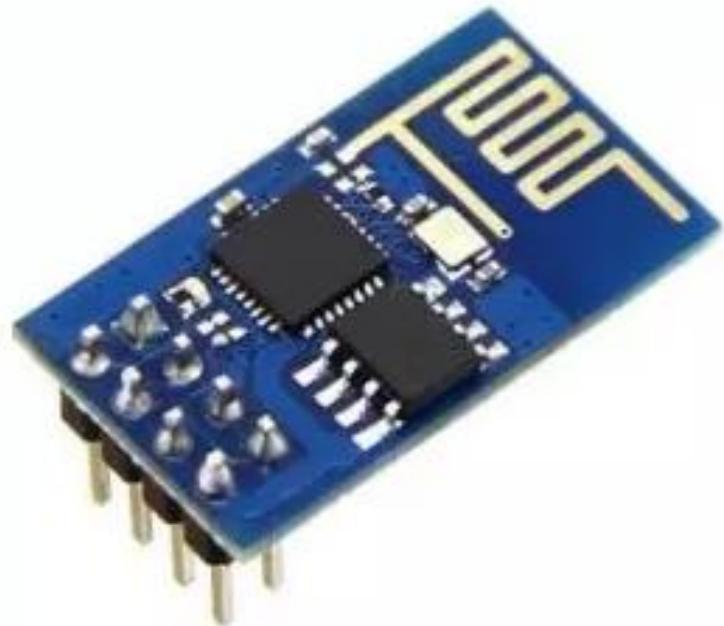
# Best Wi-Fi module?

The **ESP-01** is a popular, **inexpensive** microcontroller board with **built-in Wi-Fi**. It is a breakout board that makes use of the, now, widely used **ESP8266 microcontroller chip**.

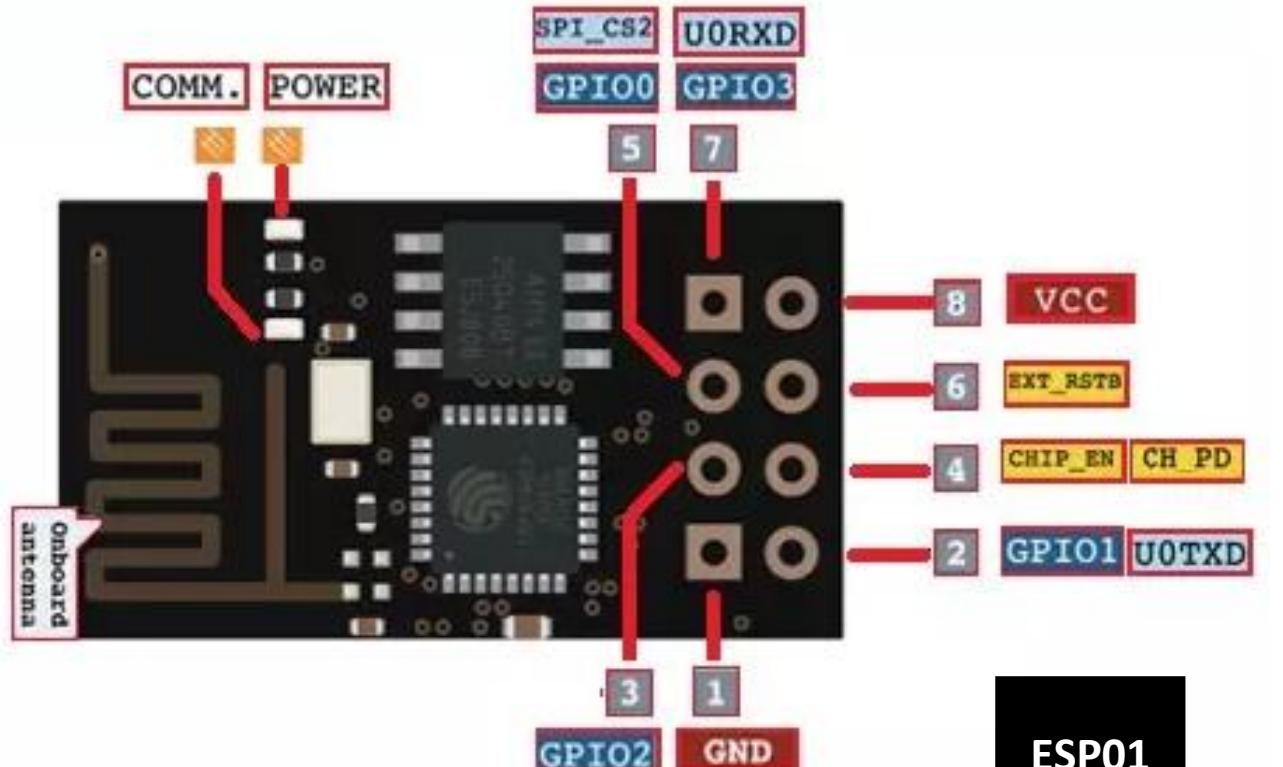
Raspberry Pi Pico RP2040 board features a dual-core Arm Cortex-M0+ processor with 264KB internal RAM & up to 16MB of off-chip Flash. The RP2040 is much faster compared to Arduino but it has a limitation of wireless network connectivity. This is the reason why Raspberry Pi Pico alone can't be used for wireless & IoT-related applications. Hence a **low-cost ESP8266 WiFi module** could be a good choice to add wireless connectivity to Raspberry Pi Pico.

The set of AT commands is preloaded into the program memory of ESP8266 and does not require additional programming. We will be sending these specific commands through the serial port (UART) of Raspberry Pi Pico. The coding is done in MicroPython & the device is programmed using the Thonny IDE. The Web Server displays the on-chip temperature sensor reading on a web page running on the web browser.

# ESP - 01Wi-Fi module



ESP-01 Module

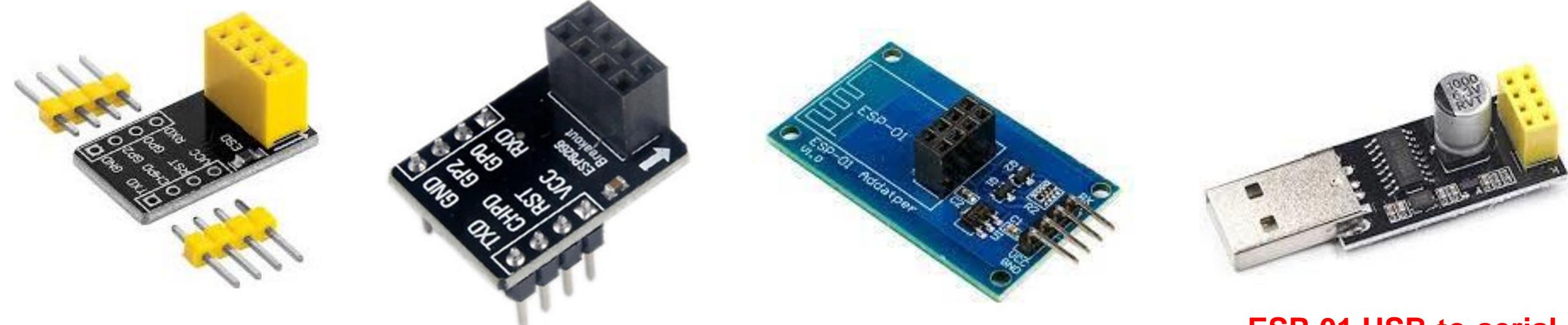


**ESP01  
Pinout**

1. GND - Ground
2. GPIO2 - General Purpose Input/Output
3. GPIO0 - General Purpose Input/Output (Used for boot mode too)
4. URXD - Receiver (for serial communication)
5. U0TXD - Transmitter (for serial communication)
6. CH\_PDN - Chip powerdown
7. EXT\_RSTB - Reset
8. VCC - 3.3v input voltage

ESP-01 is an **inexpensive, small-sized WiFi module**, which consists of TCP/IP stack along with a built-in microcontroller. So, we can directly program this small chip and can bring WiFi capability in our Embedded projects.

# ESP 01 Adapter



ESP-01 breadboard adapter.

ESP-01 USB-to-serial  
programming adapter

ESP 01 Available in market

ESP-01 BLUE



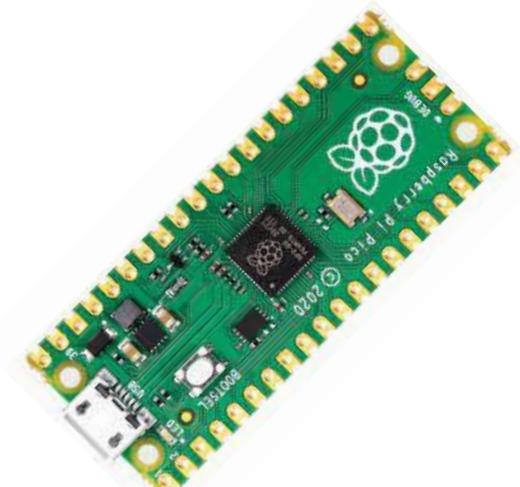
512KB

ESP-01 BLACK



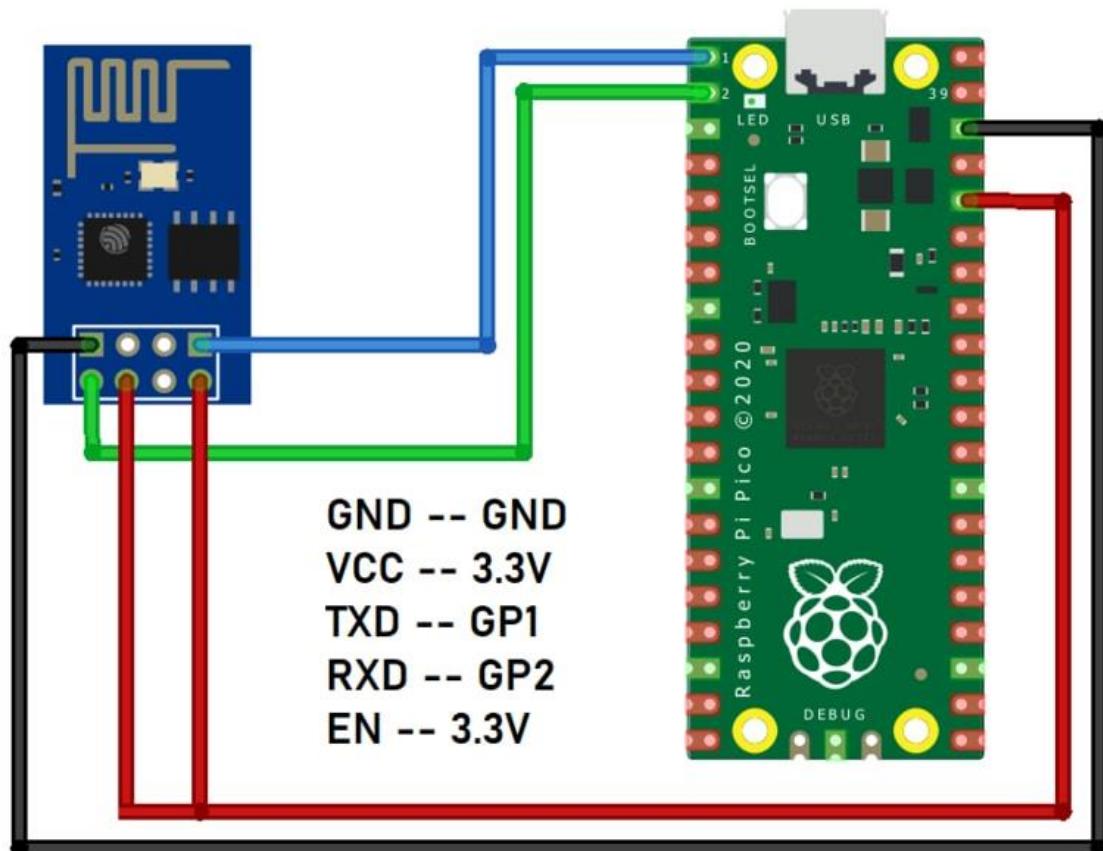
1024KB

Runs only on  
3.3 V

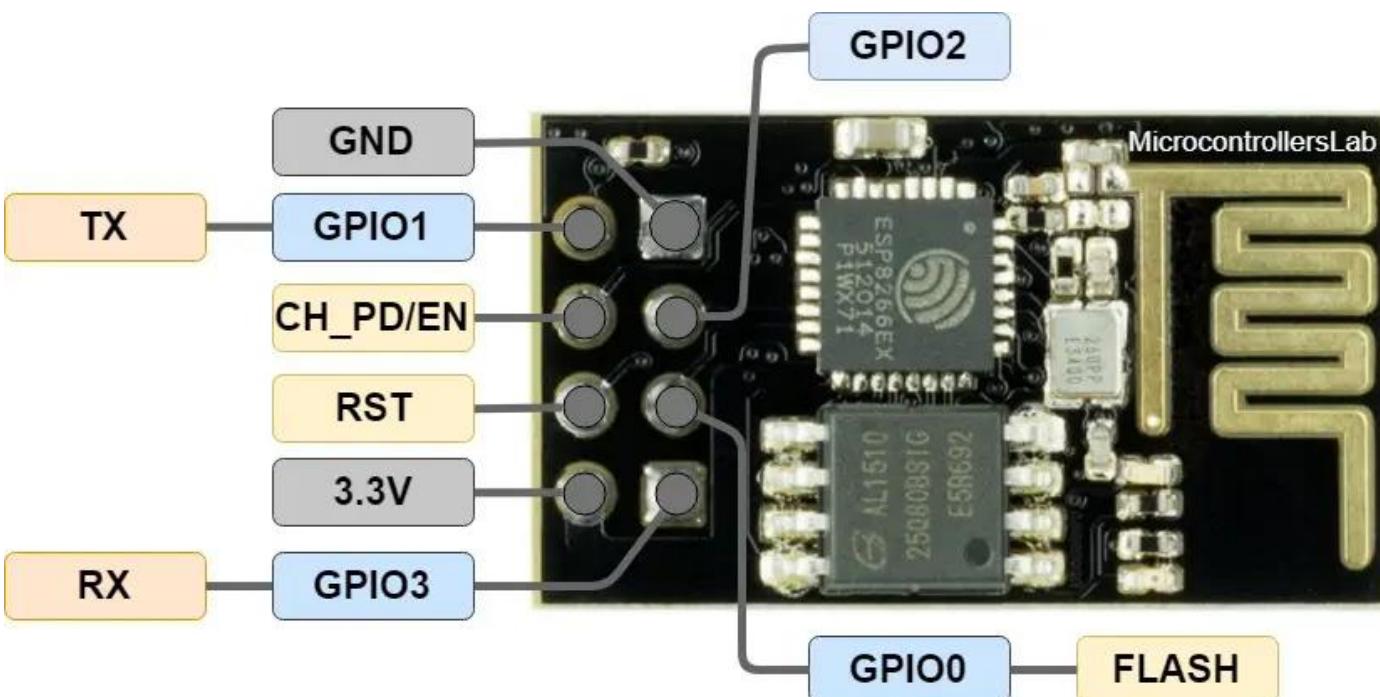


Best Choice

# Interfacing of ESP8266 WiFi Module with Raspberry Pi Pico



The Raspberry Pi Pico has **two inbuilt UART**. In this design, we will use **UART-0**. Similarly, the default baud rate of the ESP8266 is **115200**. We need to configure the Raspberry Pi Pico with the **same baud rate** in order to maintain synchronization with the ESP8266.



The ESP8266 Module is not capable of 5-3V logic shifting and will require an external Logic Level Converter. Please do not power it directly from your 5V dev board.

# Interfacing of ESP8266 WiFi Module with Raspberry Pi Pico

WOKWi SAVE SHARE Docs B

main.py • diagram.json •

```
1 from machine import UART
2 import machine
3 import _thread
4 import time
5
6 uart = UART(0,115200)
7 print('UART Serial')
8 print('>', end='')
9
10 def uartSerialRxMonitor(command):
11     recv=bytes()
12     while uart.any()>0:
13         recv+=uart.read(1)
14     res=recv.decode('utf-8')
15     erase_len=len(command)+5
16     res = res[erase_len:]
17     return res
18
19 #configure as SoftAP+station mode
20 send='AT+CWMODE=3'
21 uart.write(send+'\r\n')
22 time.sleep(1)
23
24 #Set SoftAP name
25 send='AT+CWSAP="pos_softap","","11,0,3'
26 uart.write(send+'\r\n')
27 time.sleep(1)
28 res=uartSerialRxMonitor(send)
29 print(res)
```

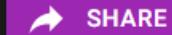
Simulation

THONNY

Raspberry Pi Pico @ 2020

ESP8266 WiFi Module

# Interfacing of ESP8266 WiFi Module with Raspberry Pi Pico

WOKWi  

main.py • diagram.json •

```
30
31 #enable multi connection mode
32 send='AT+CIPMUX=1'
33 uart.write(send+'\r\n')
34 time.sleep(1)
35 res=uartSerialRxMonitor(send)
36 print("Configured as Dual mode ->" + res)
37
38 # Enable the TCP server with port 80,
39 send='AT+CIPSERVER=1,80'
40 uart.write(send+'\r\n')
41 time.sleep(2)
42 res=uartSerialRxMonitor(send)
43 print("Server configured successfully-> "+res)
44
45 #temperature reading
46 sensor_temp = machine.ADC(4)
47 conversion_factor = 3.3 / (65535)
48
49 #Here the code runs indefinitely
50 while True:
51     #temperature reading
52     reading_temp = sensor_temp.read_u16() * conversion_factor
53     temperature = 27 - (reading_temp - 0.706)/0.001721
54     #Place basic code for HTML page display
55     val='<head><title>Pi Pico Server</title></head><body><p>Temperature: '+str(int(temperature))+ ' deg'+ '</p></body>'
56     print(val)
57     length=str(len(val))
```

WOKWi  

main.py • diagram.json •

```
58
59 send='AT+CIPSEND=1,'+length
60 uart.write(send+'\r\n')
61 time.sleep(2)
62 res=uartSerialRxMonitor(send)
63 print("Data sent-> "+res)
64 send=val
65 uart.write(send+'\r\n')
66 time.sleep(10)
```

# Interfacing of ESP8266 WiFi Module with Raspberry Pi Pico

```
[main.py] x
34 # Enable the TCP server with port 80,
35 send='AT+CIPSERVER=1,80'
>c
>>> print(send+'\n\n')

Shell x
MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

-- UART Serial --
>ftap","","11,0,3

OK

Configured as Dual mode ->nk is builded

ERROR

Server configured successfully-> change

OK

<head><title>Raspberry Pi Pico Server</title></head><body><p>Temperature: 22 deg</p>
</body>
Data sent-> OK
>
<head><title>Raspberry Pi Pico Server</title></head><body><p>Temperature: 20 deg</p>
</body>
Data sent-> 90 bytes

SEND OK
AT+CIPSEND=1,90

OK
>
<head><title>Raspberry Pi Pico Server</title></head><body><p>Temperature: 20 deg</p>
</body>
Data sent-> 90 bytes

MicroPython (Raspberry Pi Pico)
```

```
Shell x
</body>
Data sent-> 90 bytes

SEND OK
1,CLOSED
0,CLOSED
0,CONNECT

+IPD,0,464:GET / HTTP/1.1
Host: 192.168.4.1
Connection: keep-alive
Cache-Control: max-age=0
DNT: 1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64
<head><title>Raspberry Pi Pico Server</title></head><body><p>Temperature: 21 deg</p>
</body>
Data sent-> 90 bytes

SEND OK
AT+CIPSEND=1,90

OK
>
<head><title>Raspberry Pi Pico Server</title></head><body><p>Temperature: 21 deg</p>
</body>
Data sent-> 90 bytes

SEND OK
AT+CIPSEND=1,90

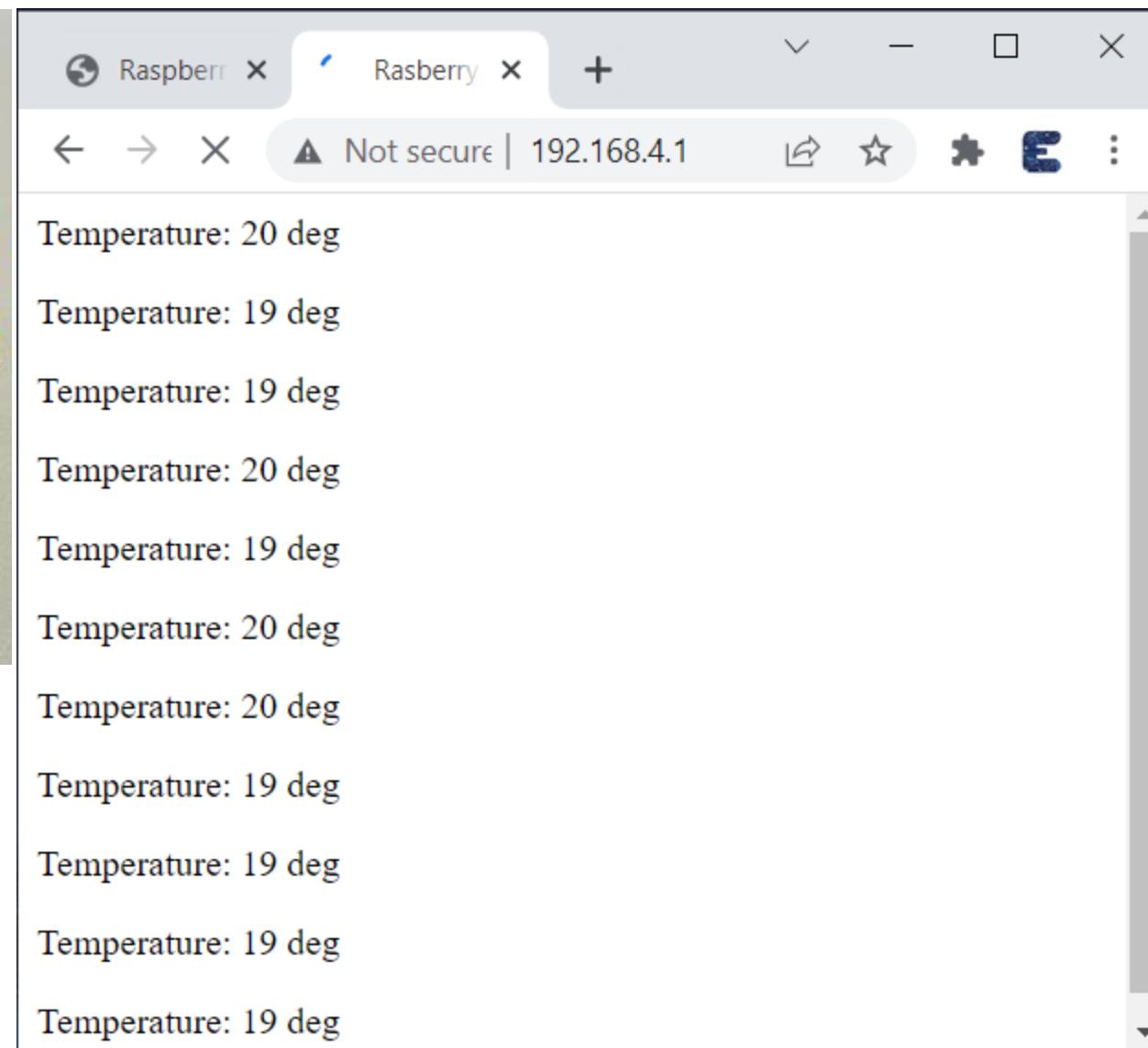
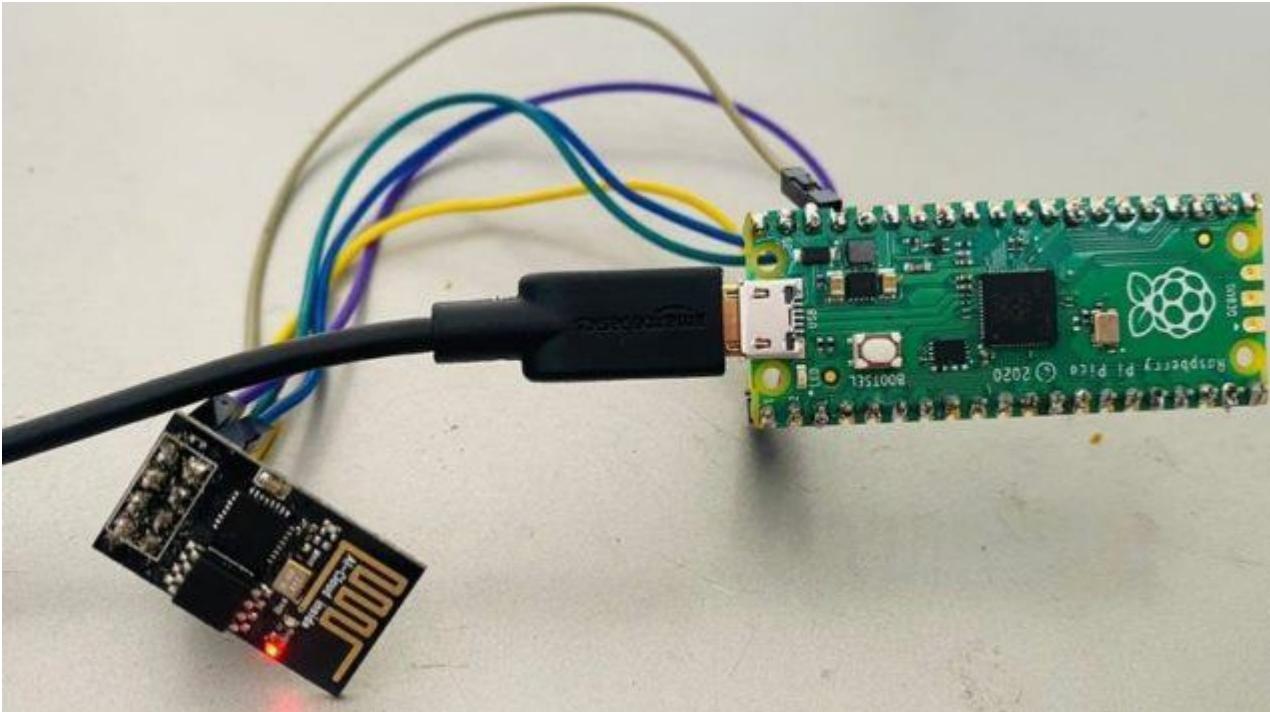
OK
> 0,CLOSED
0,CONNECT
```



Result on  
THONNY

Then open Google Chrome, type **192.168.4.1** in the address field. Then press Enter. The browser will start to display the temperature data. Since the connection of the server is not closed, it will continuously print data on the page.

# Interfacing of ESP8266 WiFi Module with Raspberry Pi Pico

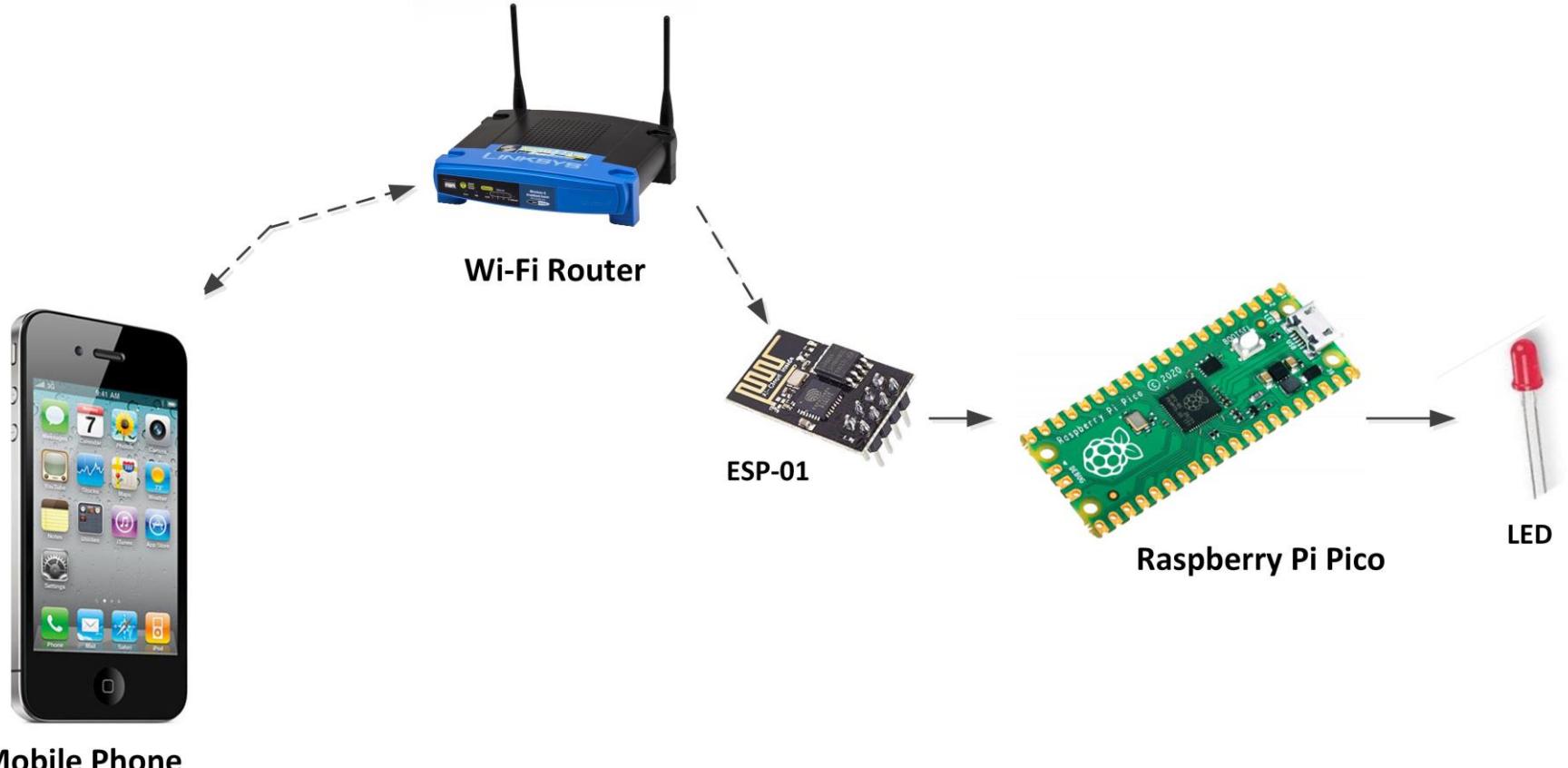


# Controlling an LED from a Smartphone Using Wi-Fi

## Objective:

Sending commands over the Wi-Fi link from a mobile phone to control an LED (the LED can be replaced with a relay, for example, to control a piece of equipment) connected to the Raspberry Pi Pico. Commands must be terminated with a Return (CR/LF or ‘newline’). Valid commands include:

LON	Turn LED ON
LOFF	Turn LED OFF



# Pico Wi-Fi Connectivity

## Objective:

Sending commands over the Wi-Fi link from a mobile phone to control an LED (the LED can be replaced with a relay, for example, to control a piece of equipment) connected to the Raspberry Pi Pico. Commands must be terminated with a Return (CR/LF or ‘newline’). Valid commands include:

LON	Turn LED ON
LOFF	Turn LED OFF

The ESP-01 communicates with the host processor through its TX and RX serial port pin. It is an 8-pin board with pin names as follows:

VCC: +3.3 V power supply pin

GND: Power supply ground

GPIO0: I/O pin. This pin must be connected to +3.3 V for normal operation, and to GND for uploading firmware to the chip

GPIO2: General purpose I/O pin

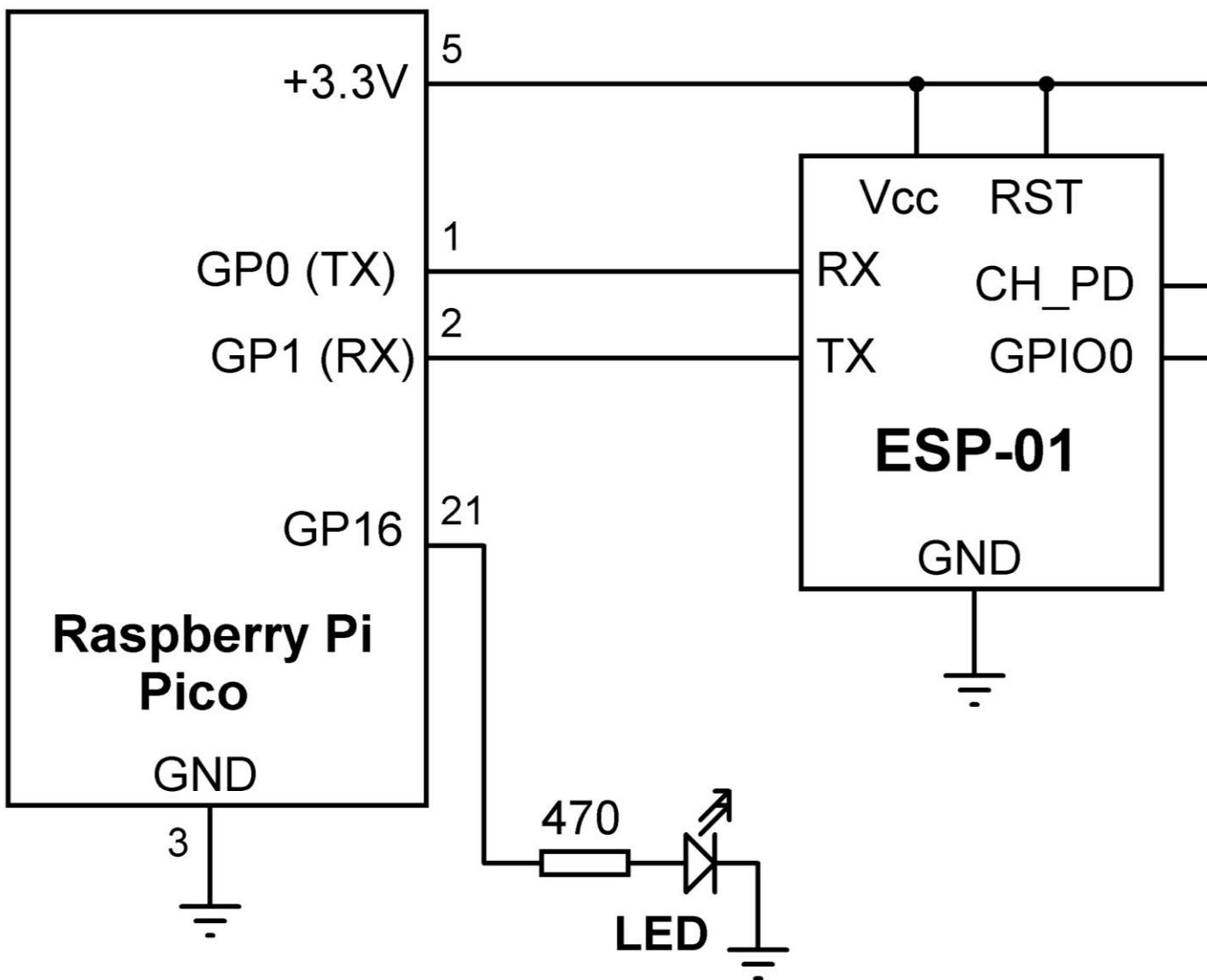
RST: Reset pin. Must be connected to +3.3 V for normal operation

CH\_PD: Enable pin. Must be connected to +3.3 V for normal operation

TX: Serial output pin

RX: Serial input pin

# Controlling an LED from a Smartphone Using Wi-Fi



# Controlling an LED from a Smartphone Using Wi-Fi

WOKWi ▾ SAVE ▾ SHARE ▾ Docs B

main.py diagram.json

```
1  from machine import Pin, UART
2  import utime
3  uart = UART(0, baudrate=115200,rx=Pin(1),tx=Pin(0))
4
5  LED = Pin(15, Pin.OUT)
6  LED.value(0)
7
8  #
9  # Send AT commands to ESP-01 to connect to local WI-Fi
10 #
11 def ConnectToWiFi():
12     uart.write("AT+RST\r\n")
13     utime.sleep(5)
14
15     uart.write("AT+CWMODE=1\r\n")
16     utime.sleep(1)
17
18     uart.write(''AT+CWJAP="IOTWORLDAB","GHOSTESP01"\r\n''')
19     utime.sleep(5)
20
21     uart.write("AT+CPIMUX=0\r\n")
22     utime.sleep(3)
```

Simulation

Diagram illustrating the hardware setup:

- Raspberry Pi Pico (left):
  - GPIO 15 is connected to the ESP-01's D0 pin.
  - GND is connected to the ESP-01's GND.
  - A red LED is connected in series with a resistor between the Pico's GND and its GPIO 15.
  - The Pico's VBUS is connected to the ESP-01's VCC.
  - The Pico's GND is also connected to the ESP-01's GND.
  - A green ground connection is shown between the Pico's GND and the ESP-01's GND.
- ESP-01 (right):
  - TX is connected to the Pico's GPIO 0.
  - RX is connected to the Pico's GPIO 1.

# Controlling an LED from a Smartphone Using Wi-Fi

WOKWi SAVE SHARE Docs B

main.py • diagram.json

```
23
24     uart.write('''AT+CIPSTART="UDP","0.0.0.0",5000,5000,2\r\n''')
25     utime.sleep(3)
26
27 ConnectToWiFi()
28
29 #
30 # Main program loop
31 #
32 while True:
33     buf = uart.readline()          # Read data
34     dat = buf.decode('UTF-8')    # Decode
35     n = dat.find("LON")        # Includes LON?
36     if n > 0:
37         LED.value(1)            # LED ON
38     n = dat.find("LOFF")      # Includes OFF?
39     if n > 0:
40         LED.value(0)            # LED OFF
41
```

Simulation

# Testing the Program

Function **ConnectToWiFi** sends the following commands to the ESP-01 to connect to the Wi-Fi network:

AT+RST	-	reset ESP-01
AT+CWMODE	-	set ESP-01 mode (here it is set to Station mode)
AT+CWJAP	-	set Wi-Fi ssid name and password
AT+CPIMUX	-	set connection mode (here it is set to multiple connection)
AT+CIFSR	-	returns the IP address (not used here)
AT+CIPSTART	-	set TCP or UDP connection mode, destination IP address, and port number (here, UDP is used with port number set to 5000. Destination IP address is set to "0.0.0.0" so that any device can send data as long as port 5000 is used (You can change this to the IP address of your smart phone to receive data only from your phone).)

# UDP Server

← → ⌂ ⌂ packetsender.com/download#show



 Bookmarks  (81) Facebook  Inbox (584) - swain...  UPSC NDA & NA (II...)  RRB Senior Section...  opsc  Forums / Projects /...  Electricity Generatin...  ScholarOne Manus...

23

# Packet Sender

Home

Download

Documentation

## Sponsors

Contact

Packet Sender uses NO cookies, NO telemetry, and NO ads. The project survives thanks to YOUR support.

- One-time donation (Thank You)

Send via PayPa

 No thanks, just let me download.

[Click here](#)



Version v8.1.1

 Installer for Windows Or [winget](#)

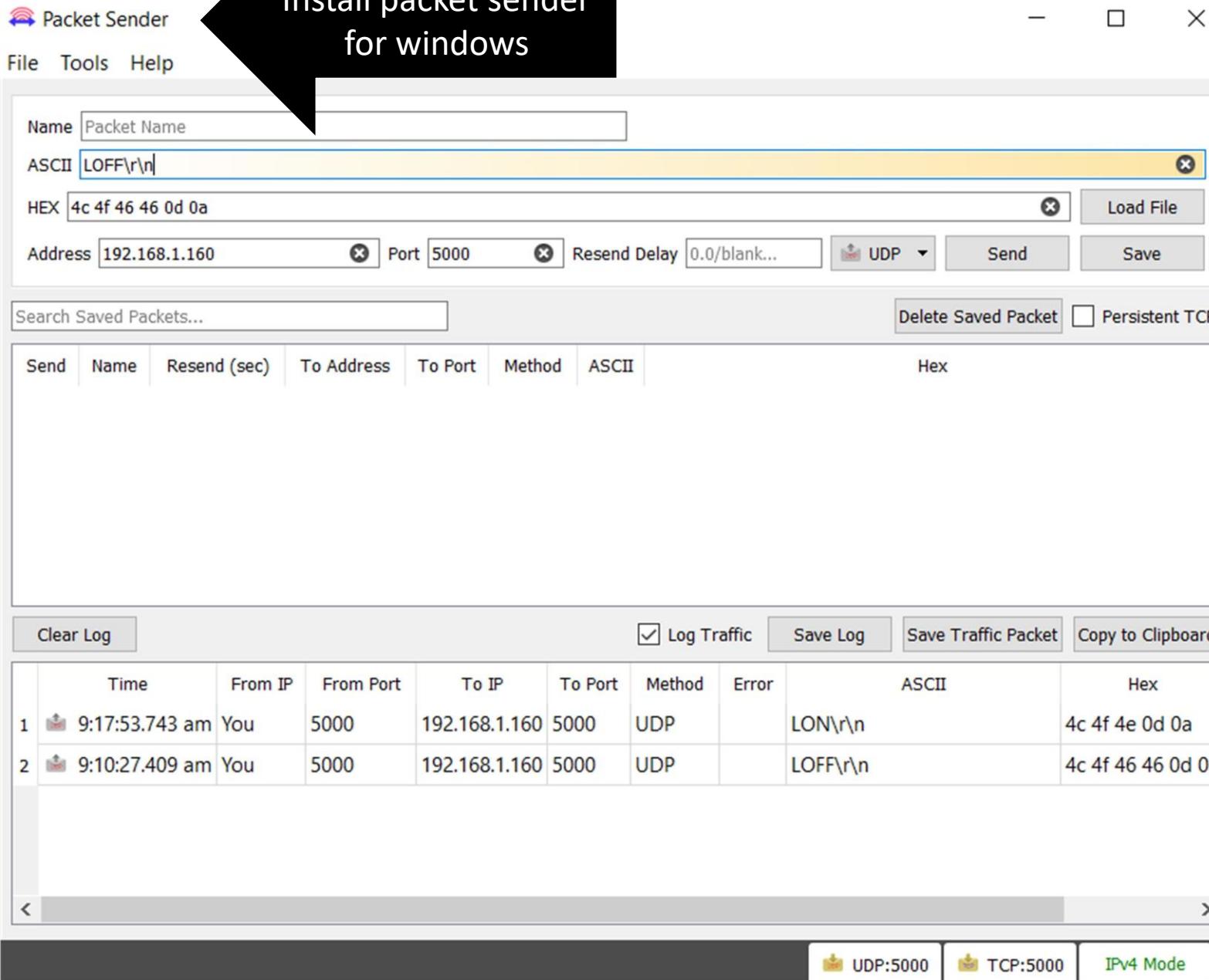
```
winget install packetsender
```



Version v8.1.1

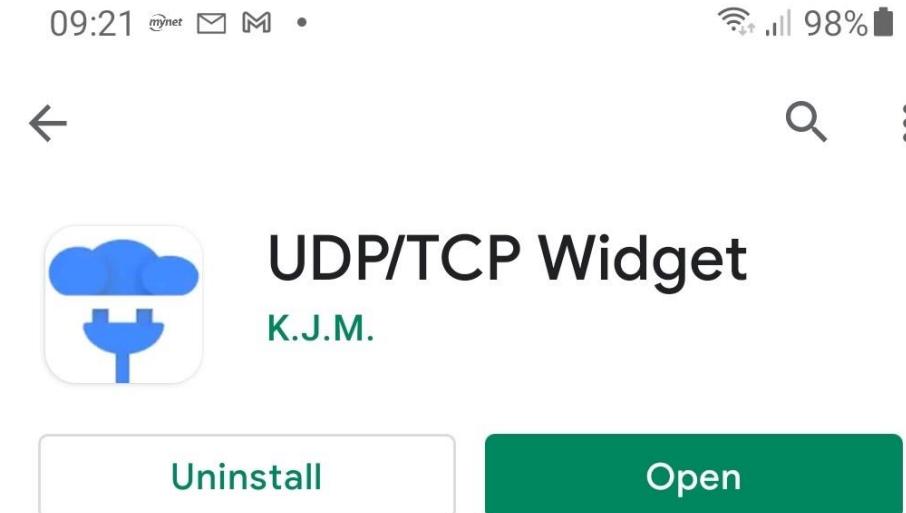
 Portable Version for Windows

# Testing the Program



Install a UDP Server app on your  
Android mobile phone

freely available UDP apps in  
the *Play Store*



# Testing the Program: Packet sender

Packet Sender - IPs: 192.168.29.91

- X

File Tools Multicast Panels Help

Name

ASCII

HEX

Address

Port

Resend Delay

TCP

Search Saved Packets...

Persistent TCP

	Send	Name	Resend	To Address	To Port	Method	ASCII
1	<input type="button" value="Send"/>	DNS dannagle.com	0	1.1.1.1	53	UDP	\f7\03\01\00\00\01\00\00\00\00\00\08dannagle\03com\00\00\01\00\01
2	<input type="button" value="Send"/>	DNS example.com	0	8.8.8.8	53	UDP	\91\8b\01\00\00\01\00\00\00\00\00\00\07example\03com\00\00\01\00\01
3	<input type="button" value="Send"/>	FTP debian.org	0	cdimage.debian.org	21	TCP	\r\nUSER anonymous\r\nPASS anonymous\r\nquit\r\n
4	<input type="button" value="Send"/>	HTTP GET	0	neverssl.com	80	HTTP Get	/
5	<input type="button" value="Send"/>	HTTP POST Params	0	httpbin.org	80	HTTP Post	/post
6	<input type="button" value="Send"/>	HTTPS GitHub API	0	api.github.com	443	HTTPS Get	/users/dannagle

Log Traffic

Time	From IP	From Port	To Address	To Port	Method	Error	ASCII	Hex
------	---------	-----------	------------	---------	--------	-------	-------	-----

UDP:52326

TCP:54570

SSL:54571

IPv4 Mode

# Testing the Program: Packet sender

Packet Sender - IPs: 192.168.29.91

- X

File Tools Multicast Panels Help

Name

ASCII  X Load File

HEX  X

Address

Port

5000



Resend Delay

0.0/blank ...



UDP



Send

Save

Search Saved Packets...

Delete Saved Packet

Persistent TCP

	Send	Name	Resend	To Address	To Port	Method	ASCII
1		Send DNS dannagle.com	0	1.1.1.1	53	UDP	\f7\ e3\01\00\00\01\00\00\00\00\00\00\08dannagle\03com\00\00\01\00\01
2		Send DNS example.com	0	8.8.8.8	53	UDP	\91\8b\01\00\00\01\00\00\00\00\00\00\07example\03com\00\00\01\00\01
3		Send FTP debian.org	0	cdimage.debian.org	21	TCP	\r\nUSER anonymous\r\nPASS anonymous\r\nquit\r\n
4		Send HTTP GET	0	neverssl.com	80	HTTP Get	/
5		Send HTTP POST Params	0	httpbin.org	80	HTTP Post	/post
6		Send HTTPS GitHub API	0	api.github.com	443	HTTPS Get	/users/dannagle

Log Traffic

Time

From IP

From Port

To Address

To Port

Method

Error

ASCII

Hex

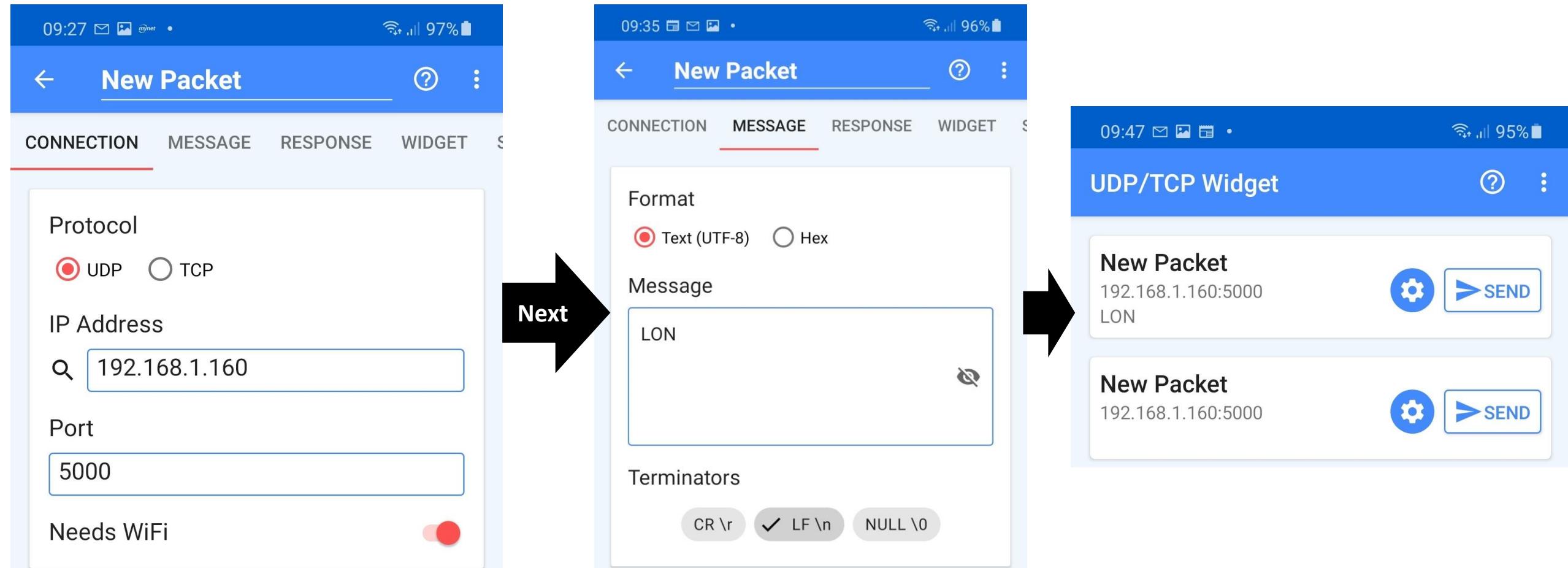
UDP:52326

TCP:54570

SSL:54571

IPv4 Mode

# Testing the Program in Smartphone



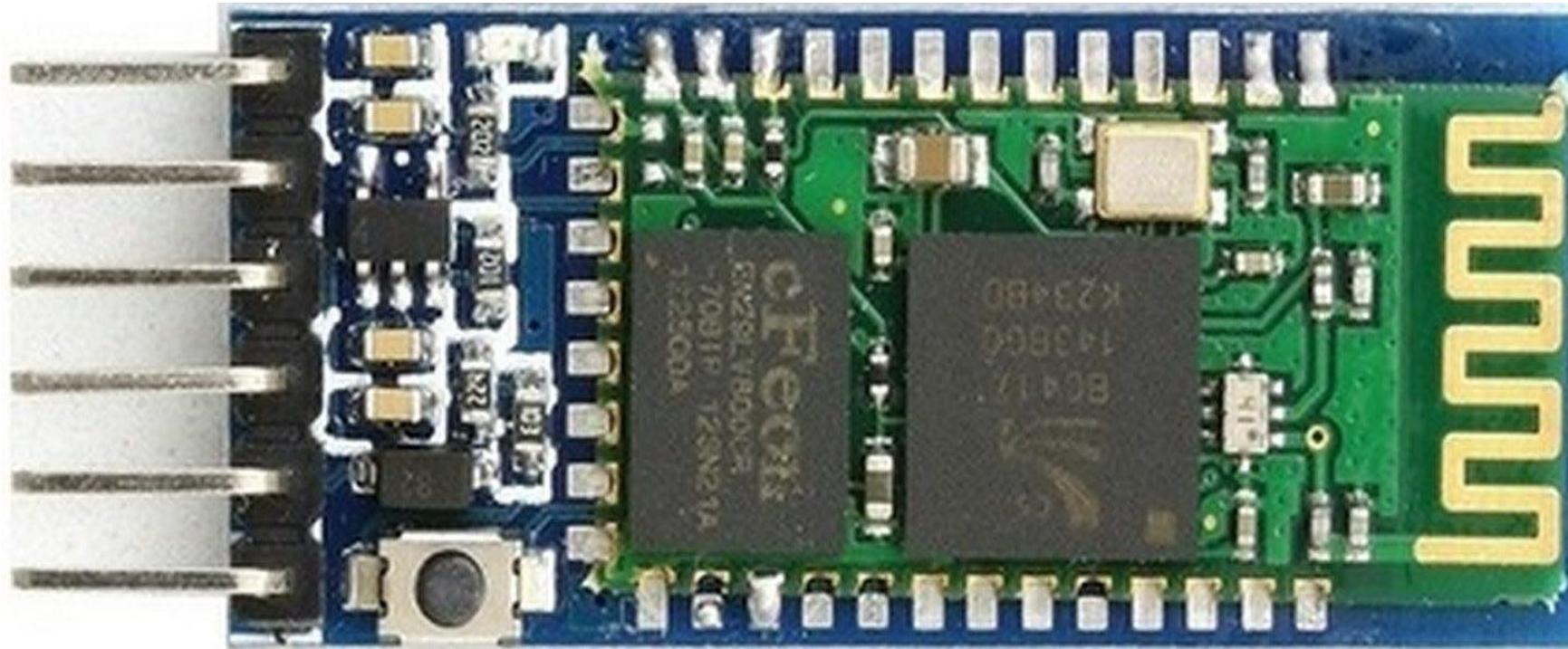
IP address of the ESP-01 can be obtained by scanning all the devices on the local Wi-Fi router. For example, the Android app called **Who Uses My WiFi – Network Scanner** by **Phuongpn** can be used to see the IP addresses of all the devices connected to your router.



# Control an LED from the Android application using Bluetooth wireless communication

## Bluetooth Module HC-05

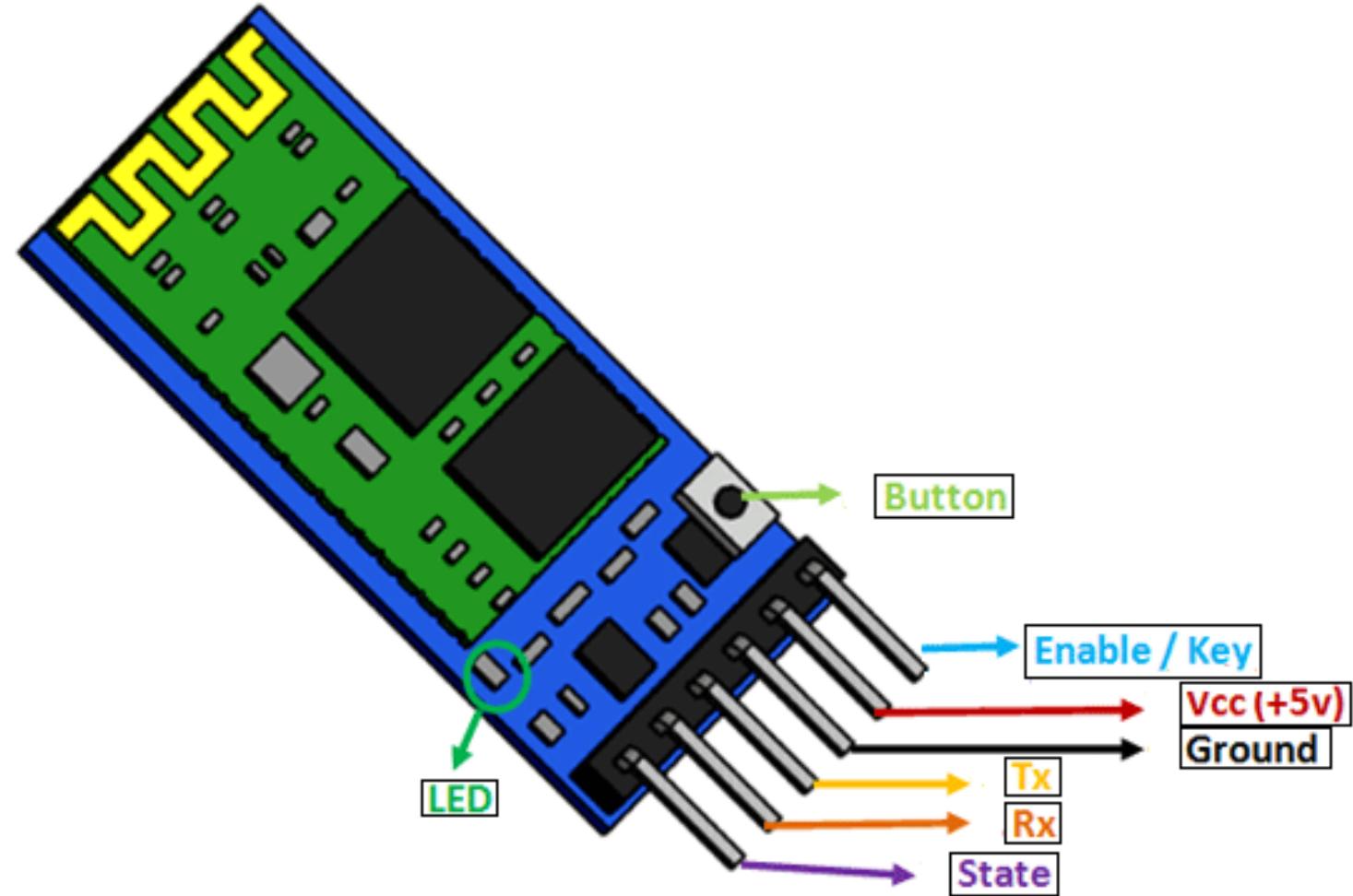
HC-05 is one of the commonly used Bluetooth device that uses a UART communication protocol. The module normally operates at UART serial communication with TX and RX pins at 9600 baud rates. However, it can be programmed with AT commands and it supports 9600,19200,38400,57600,115200,230400,460800 baud rates.



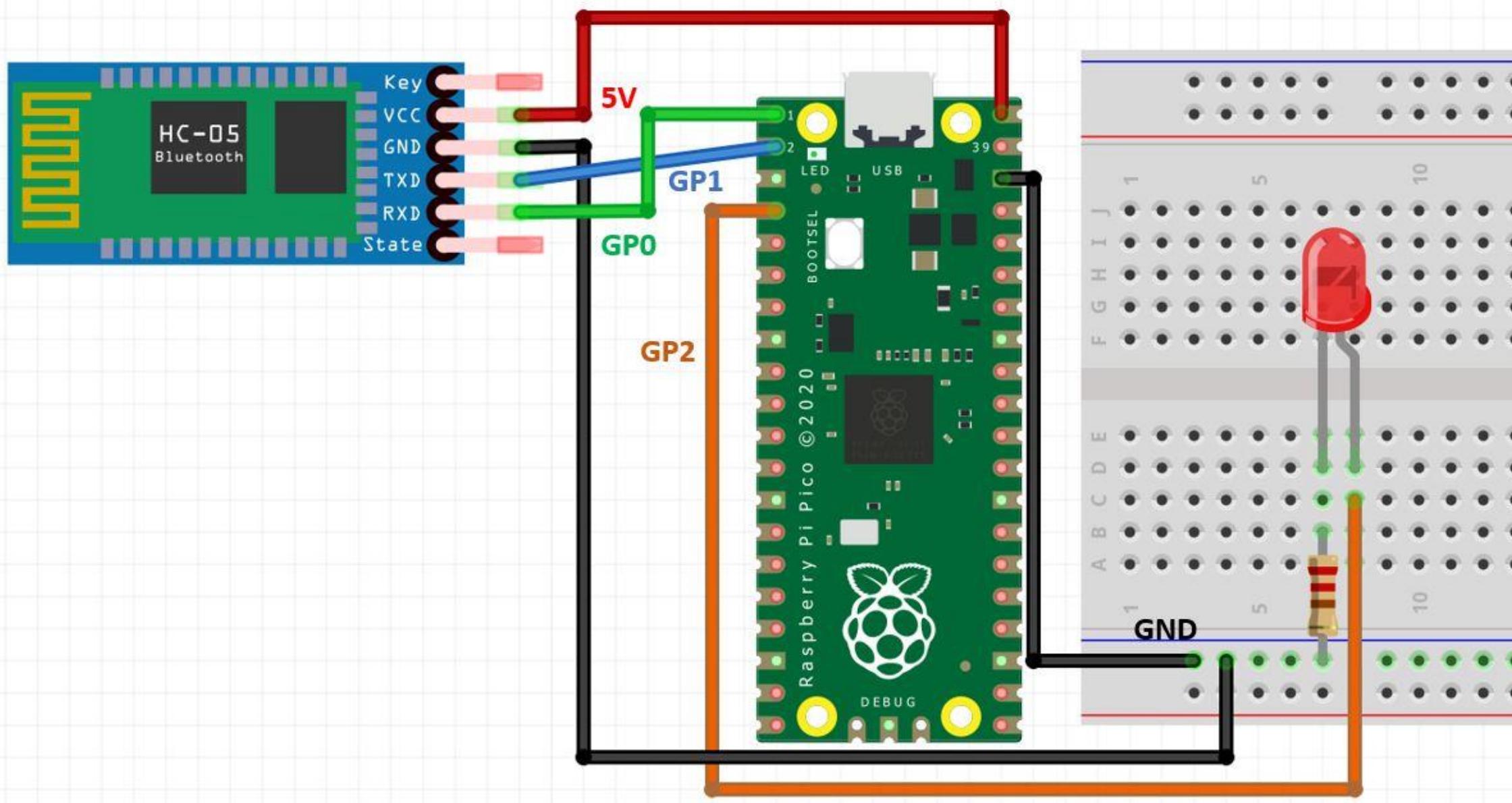
# Bluetooth Module HC 05 Pinout

## Bluetooth Module HC-05

HC-05 is one of the commonly used Bluetooth device that uses a UART communication protocol. The module normally operates at UART serial communication with TX and RX pins at 9600 baud rates. However, it can be programmed with AT commands and it supports 9600, 19200, 38400, 57600, 115200, 230400, 460800 baud rates.

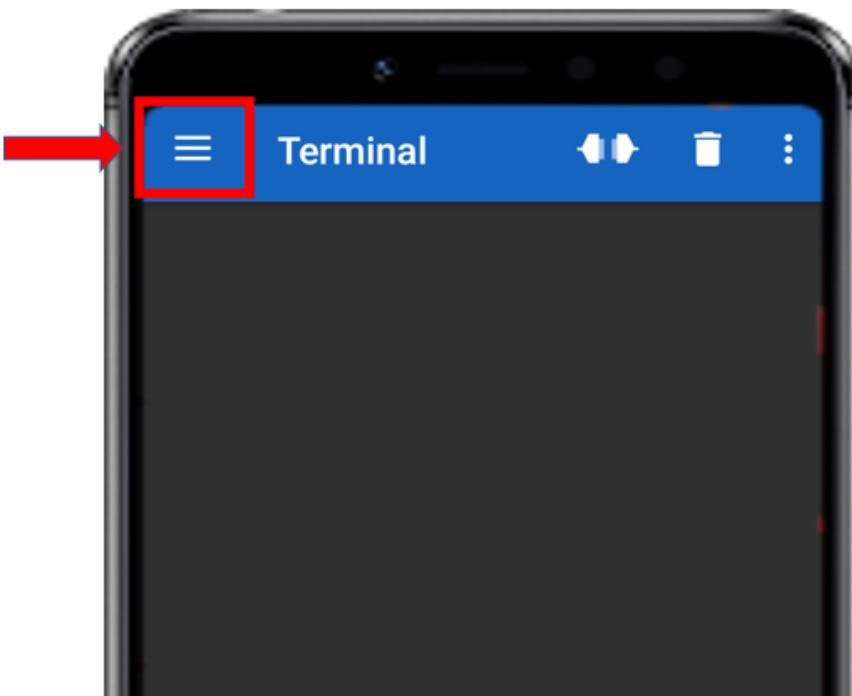
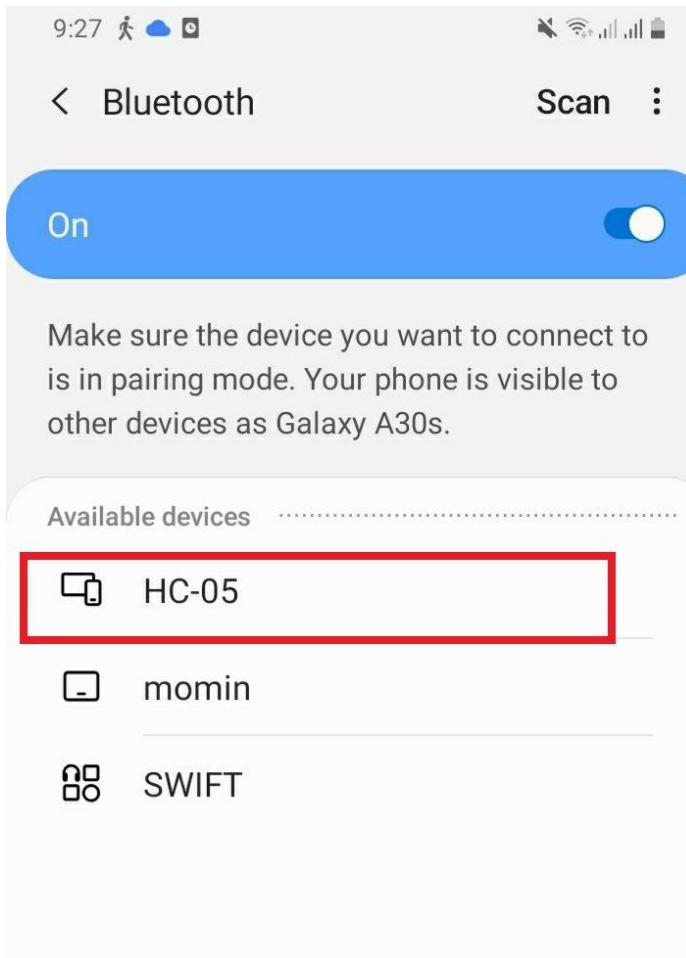
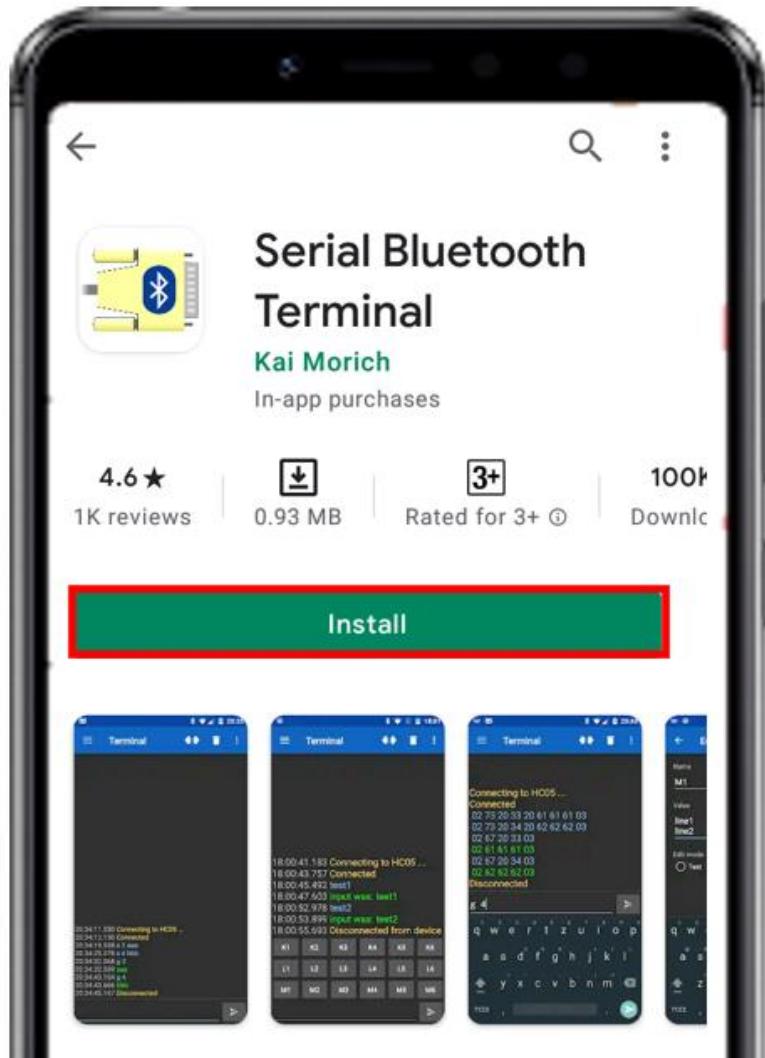


# Schematic Raspberry Pi Pico and HC-05

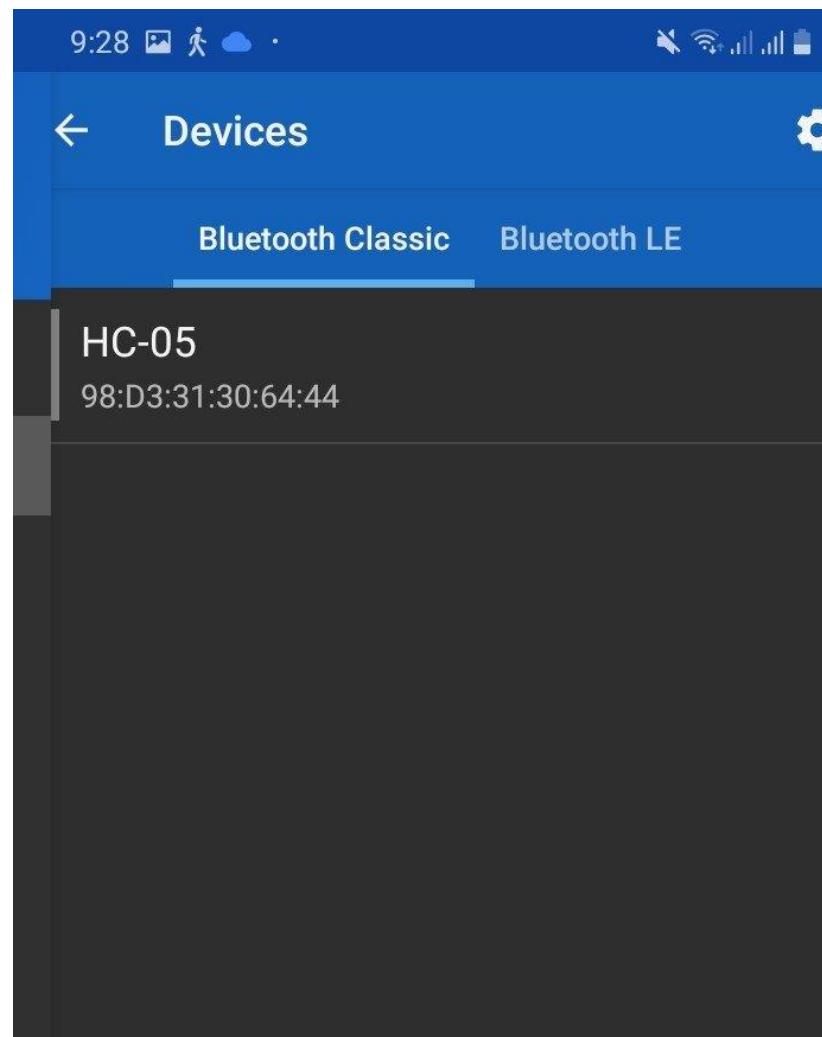
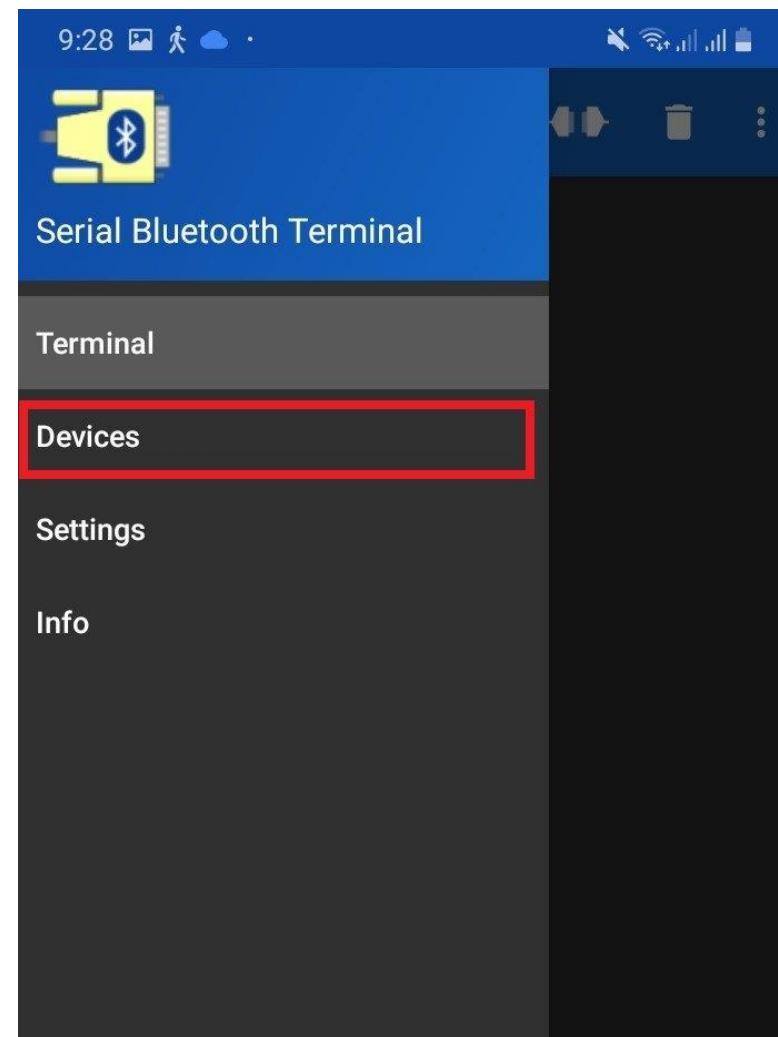


# Bluetooth Terminal Application

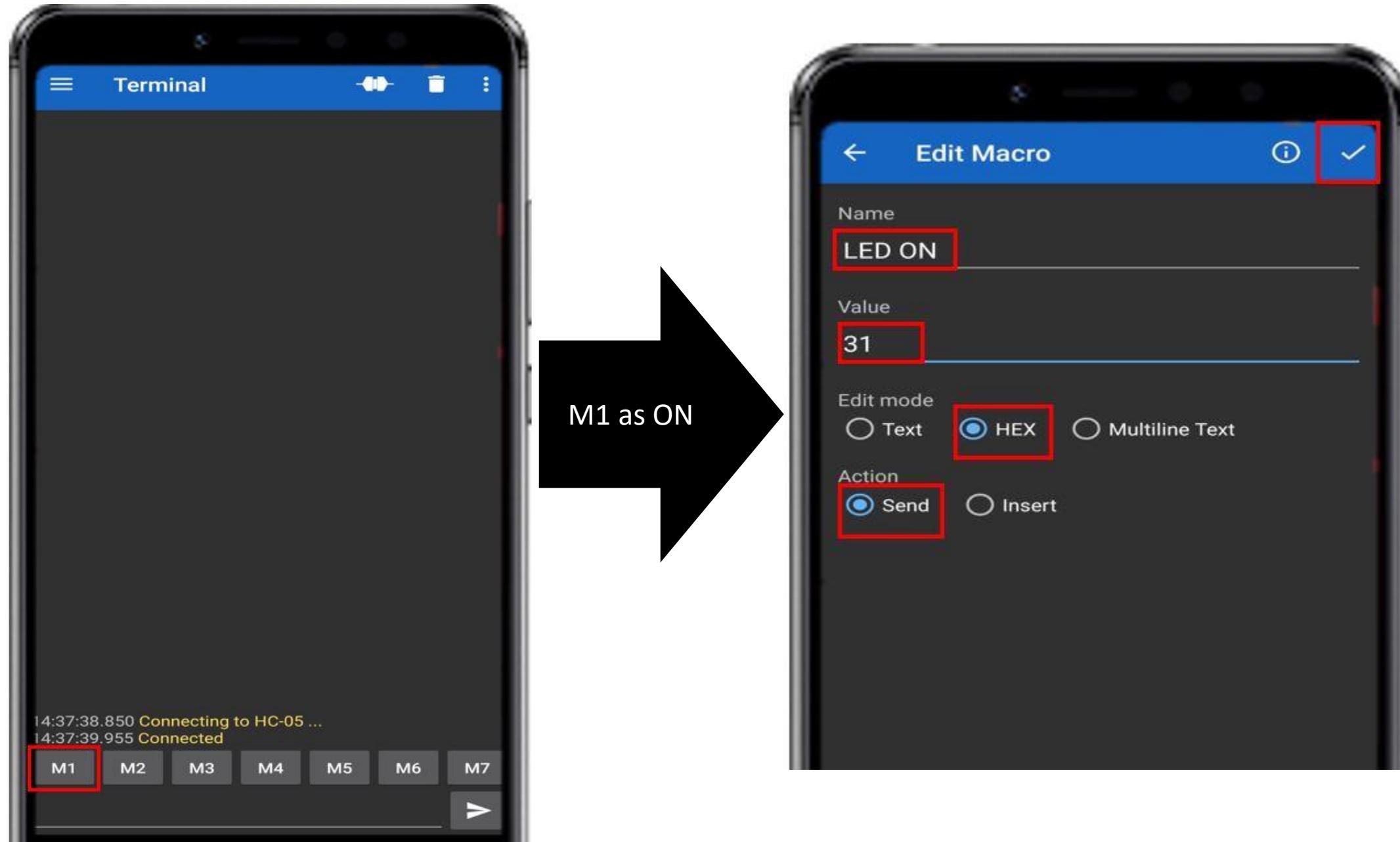
Go to the Play Store and download the application by the name: Serial Bluetooth terminal.



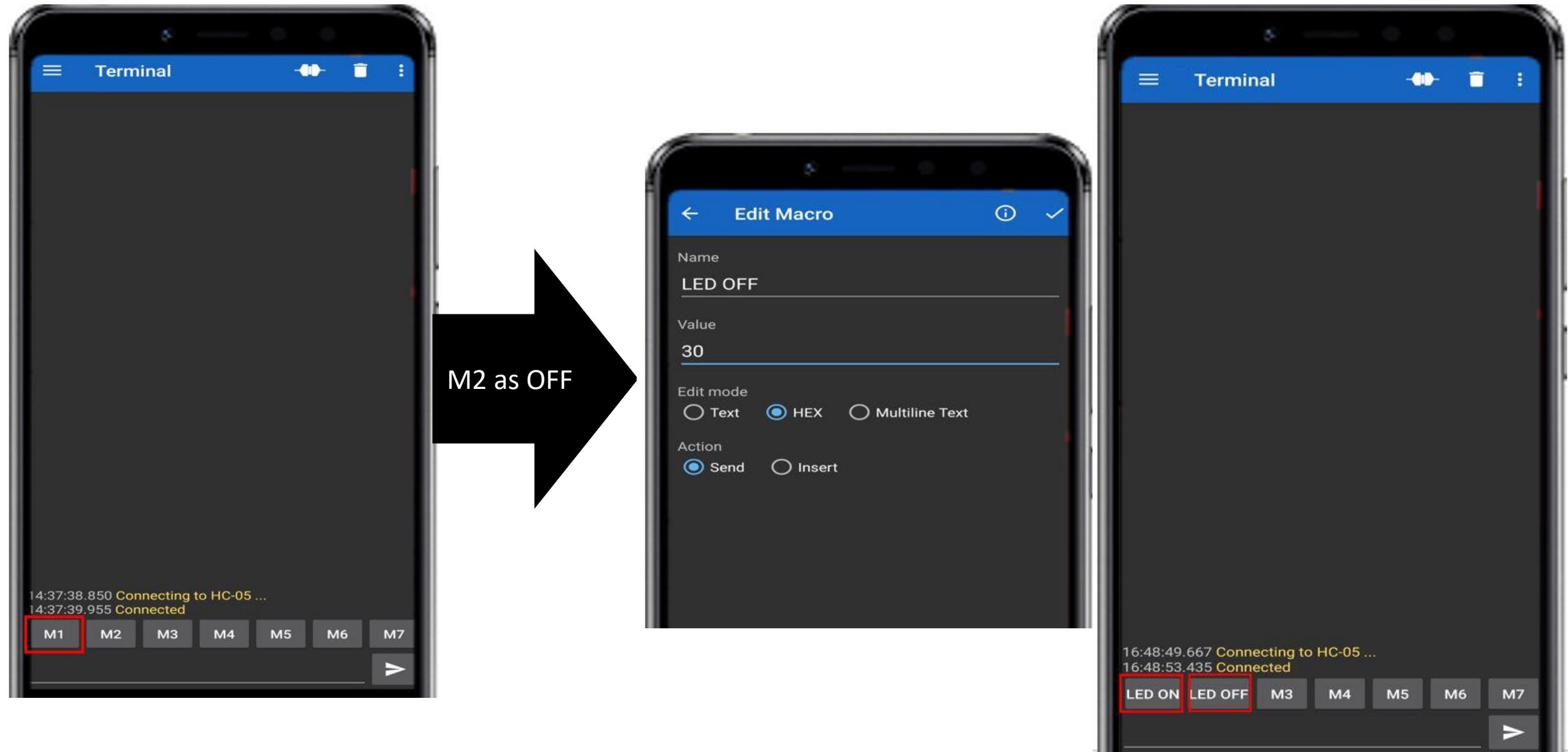
# Bluetooth Terminal Application



# Bluetooth Terminal Application



# Bluetooth Terminal Application

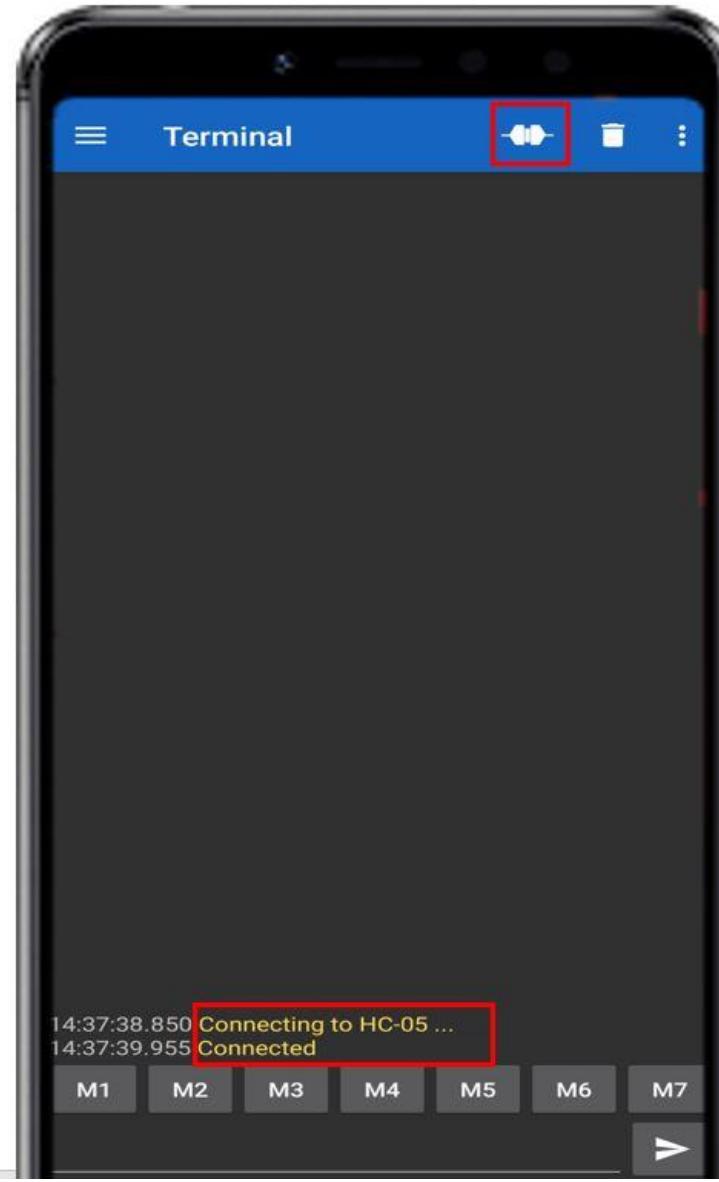


# Bluetooth Terminal Application

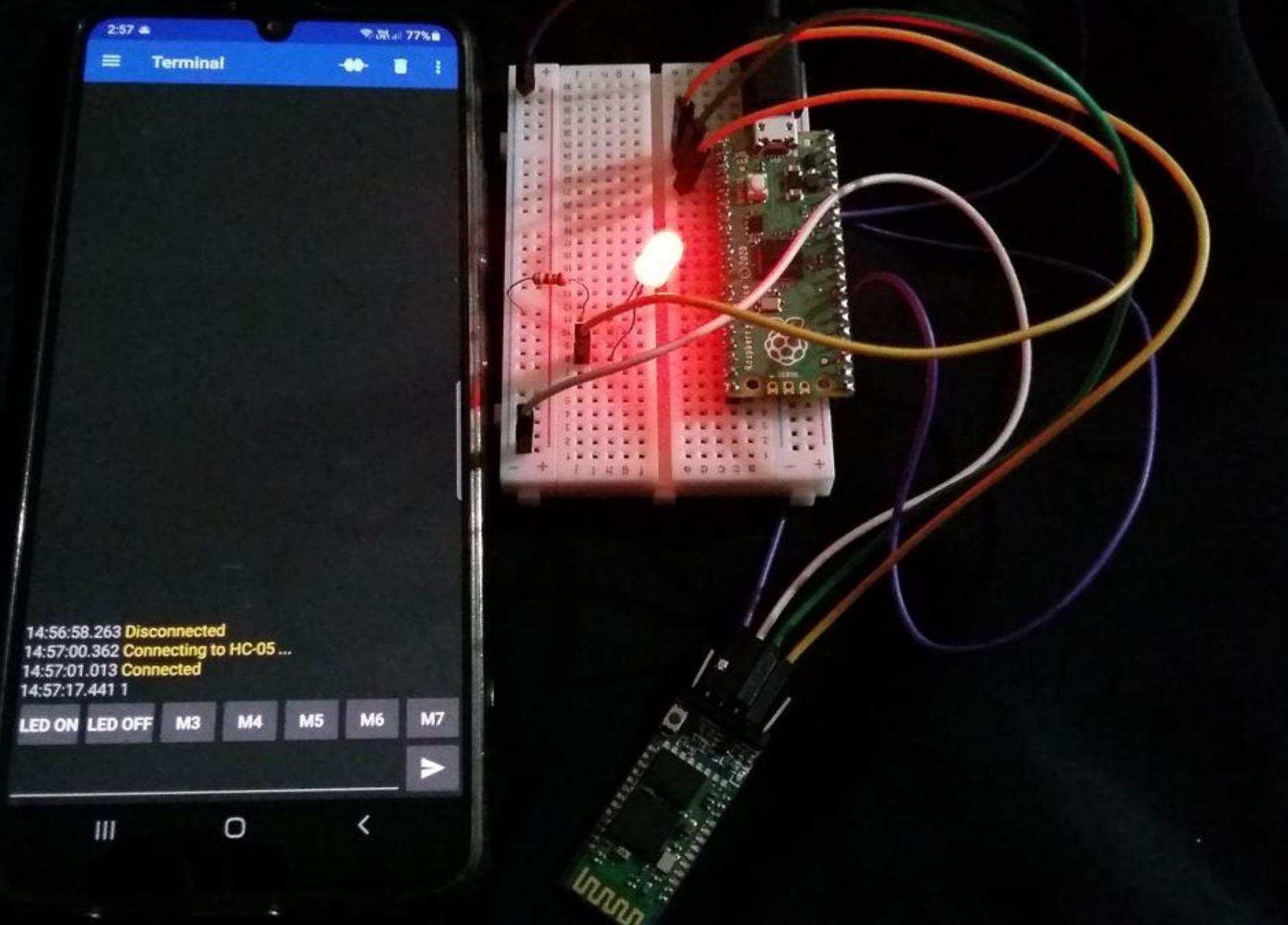
```
from machine import Pin,UART
uart = UART(0,9600)

Led_pin = 2
led = Pin(Led_pin, Pin.OUT)

while True:
    if uart.any():
        data = uart.readline()
        print(data)
        if data== b'1':
            led.high()
            print("LED is now ON!")
        elif data== b'0':
            led.low()
            print("LED is now OFF!")
```

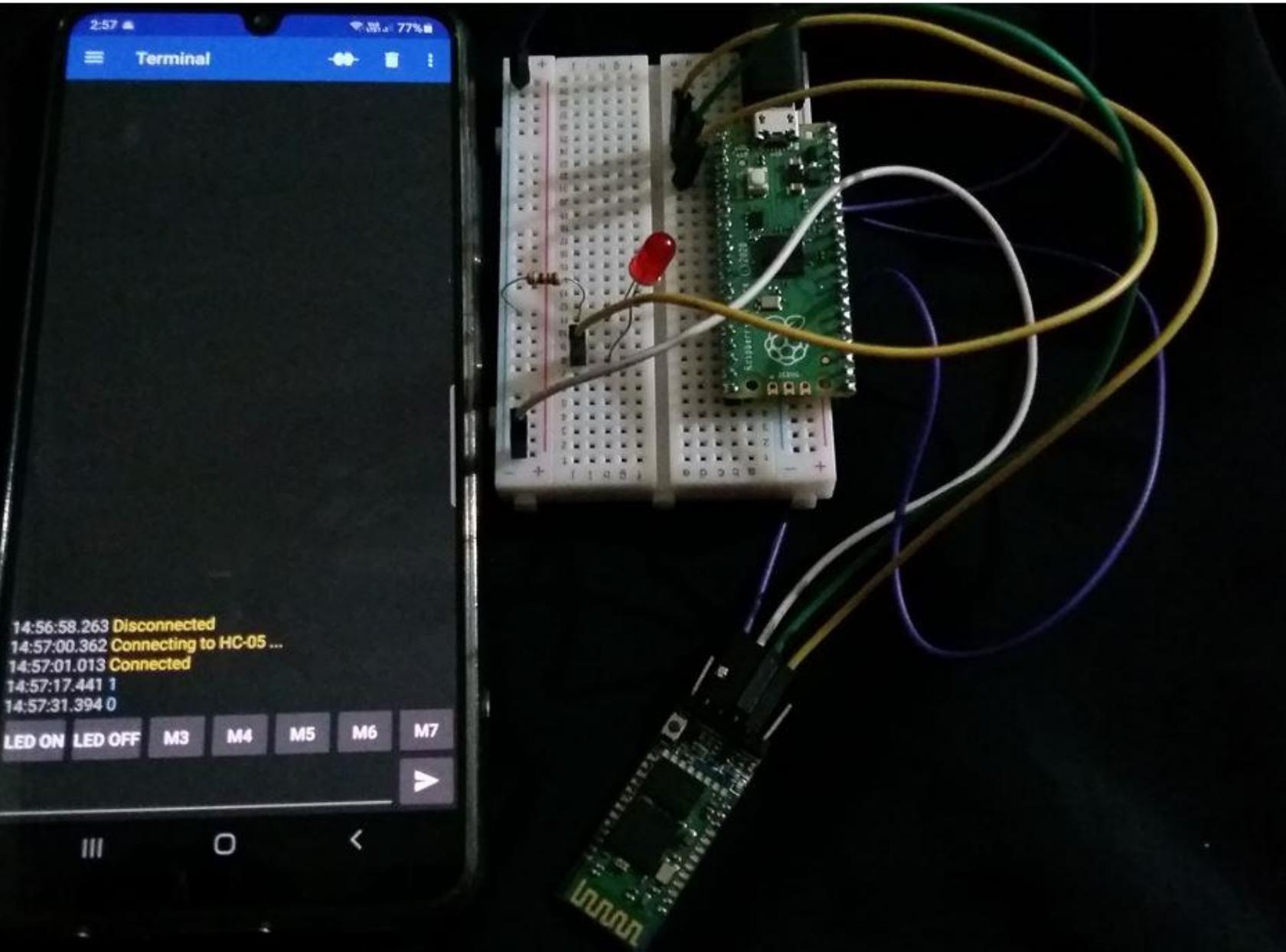


# Bluetooth Terminal Application



Press  
LEDON

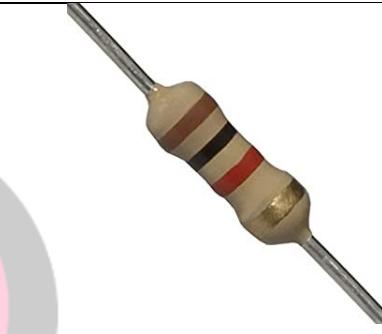
# Bluetooth Terminal Application



Press  
LEDOFF

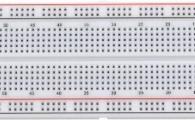


## LIST OF COMPONENTS REQUIRED FOR IOT PROJECTS USING PYTHON (CSE 4110)

SL NO	Name of the Component	Specification	Quantity	Components Picture
1	Raspberry Pi Pico (with cable USB A to Micro B) with <b>soldered Head</b>		1	
2	Resistor	220ohm, 0.25W	5	
		330ohm, 0.25W	10	
		470ohm, 0.25W	5	
		2.2K, 0.25W	5	
		4.7K, 0.25W	5	
		10K, 0.25W	10	
		1K, 0.25W	2	
3	Trimmer Potentiometer	10k	1	
4	Potentiometer	10k	1	



## LIST OF COMPONENTS REQUIRED FOR IOT PROJECTS USING PYTHON (CSE 4110)

5	LED	RED 3mm diameter, generic	10	
		BLUE 3mm diameter, generic	6	
		YELLOW 3mm diameter, generic	6	
6	RGB LED (4 terminal)	Common Cathode	2	
7	PIR sensor	HC-SR501	1	
8	Breadboard	840 tie points	1	
9	Jumper wires(different colours)	Male to Male	30	



## LIST OF COMPONENTS REQUIRED FOR IOT PROJECTS USING PYTHON (CSE 4110)

10	Electromagnetic Buzzer (Active Buzzer, small)	5V	1		
11	Piezo Buzzer (Active, Big)		1		
12	LDR (Photoresistor)	5mm	2		
13	Push Button Tactile Switch	6x6x5 mm	10		

## LIST OF COMPONENTS REQUIRED FOR IOT PROJECTS USING PYTHON (CSE 4110)

14	Micro Servo Motor	SG90	1	
15	Joystick Module		1	
16	0.96-inch I2C OLED display(ssd 1306)	128 x 64	1	
17	Dual shaft BO motor with wheel	200 rpm	1	
18	Diode	1N4148	1	
19	Temperature and Humidity Sensor Module	DHT11 module	1	
20	I2C 16x2 LCD Display for Arduino with <b>Presoldered Male Header Pins</b>	16x2 (HD44780 LCD)	1	

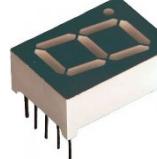
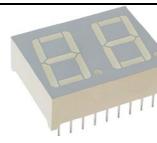
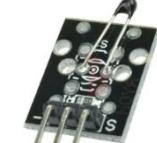
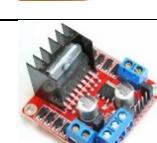


## LIST OF COMPONENTS REQUIRED FOR IOT PROJECTS USING PYTHON (CSE 4110)

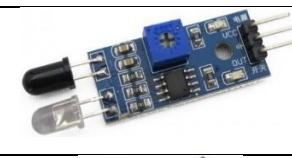
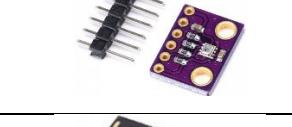
21	I2C Module for 16x2 (1602) Character LCD with Presoldered Male Header Pins		1	
22	Ultrasonic Sensor (4pin)	HC-SR04	1	
23	20 Pins Male Berg Strip Connector Header Pin	40Pins	1	
24	KY-021 mini magnetic reed switch module		1	
25	KY-034 7-colour flashing LED module	3 pins	1	
26	Simple thin magnet	12 x 2 magnet	1	
27	Transistor (NPN)	BC 108	2	
		2N2222	2	
		BC 548	2	



## LIST OF COMPONENTS REQUIRED FOR IOT PROJECTS USING PYTHON (CSE 4110)

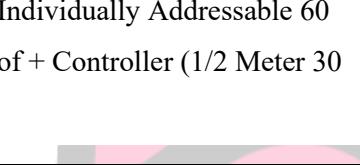
28	7 Segment Display - Common Cathode - 0.56 inch - Standard Size		1	
29	2 digit 7 segment display module (Common Cathode)	DC56-11EWA	2	
30	Analog Temperature Sensor	TMP36	2	
31	5v relay module		1	
32	KY-013 Analog Temperature Sensor Module	KY-013	1	
33	Digital Multimeter		1	
34	L298 2A Dual Motor Driver Module with PWM Control	2A	1	

## LIST OF COMPONENTS REQUIRED FOR IOT PROJECTS USING PYTHON (CSE 4110)

35	9V alkaline Battery with snap connector		1	
36	Infra-Red (IR) Sensor Module		1	
37	IRL540 type MOSFET switch.	N-Channel	1	
38	BMP280 Barometric Pressure and Altitude Sensor I2C/SPI Module		1	
39	ESP-01 ESP8266 Serial WIFI Transceiver Module		1	
40	3.3V Adapter Board for 24L01 Wireless Module	ESP-01 breadboard adapter	1	
41	HC-05 Bluetooth Module		1	



## LIST OF COMPONENTS REQUIRED FOR IOT PROJECTS USING PYTHON (CSE 4110)

42	Photo resistor Module		1	
43	Infrared IR Wireless Remote Transmitter and receiver Control Module		1	
44	ws2812b 5v LED Strip Individually Addressable 60 Pixels/m Non Waterproof + Controller (1/2 Meter 30 LEDs)		1	

