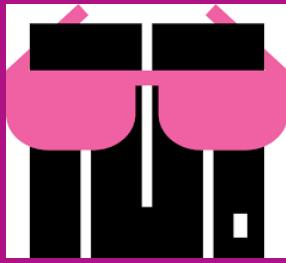
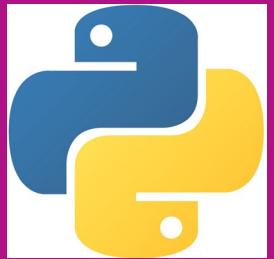
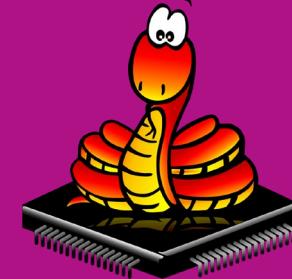


INTERNET OF THINGS (IOT) PROJECTS USING PYTHON

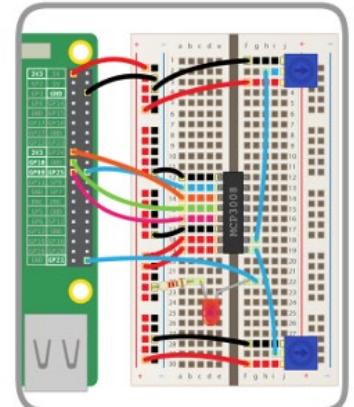
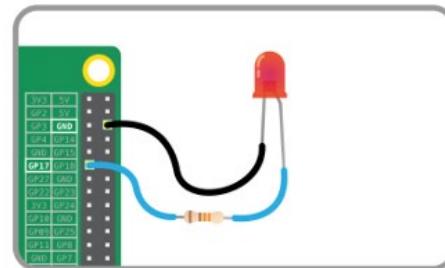
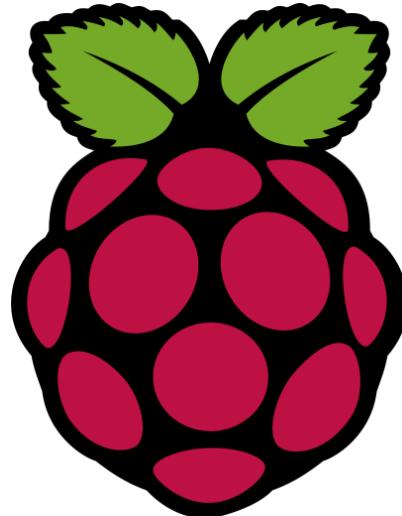


(CSE 4110)

(LECTURE – 5)



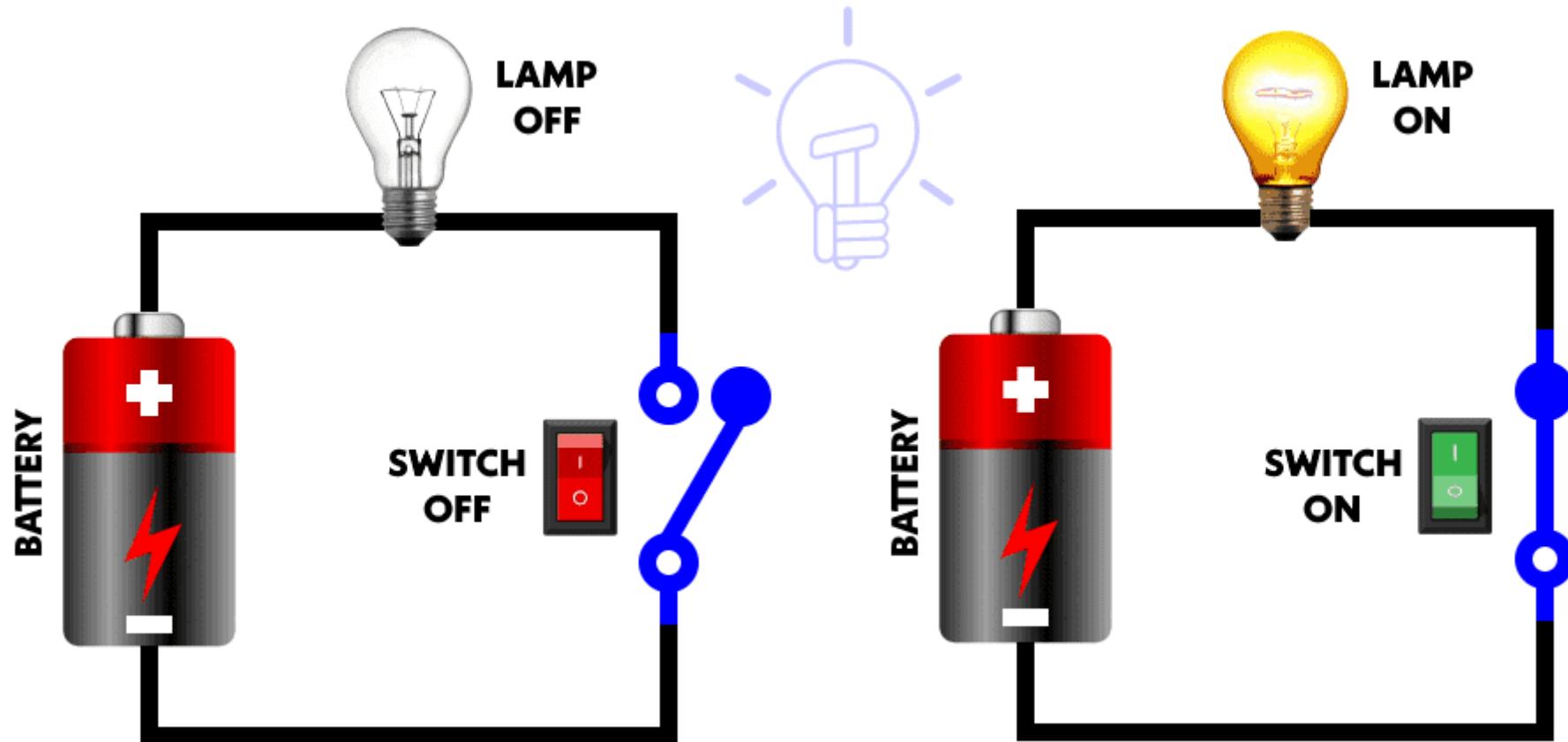
T_h



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.

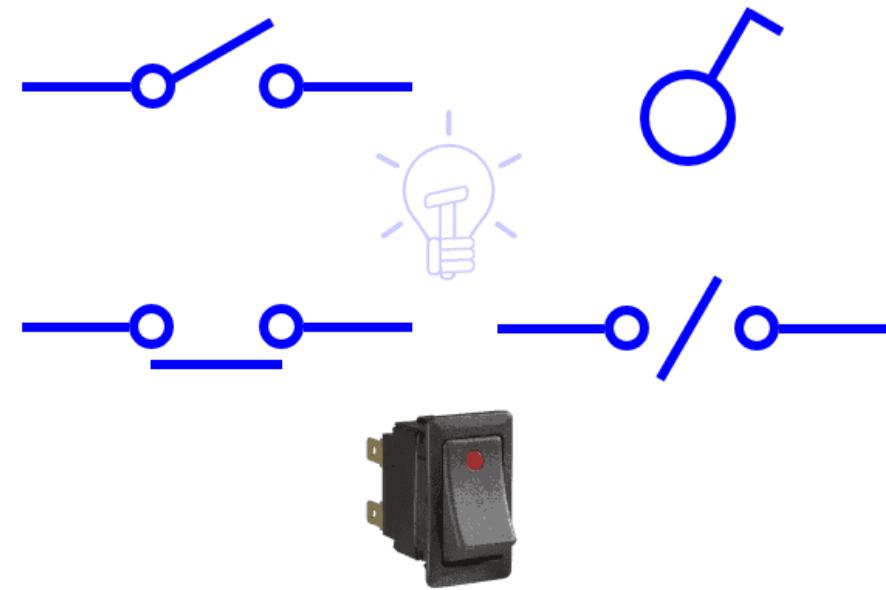


Mechanical Switch

- ✓ **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)

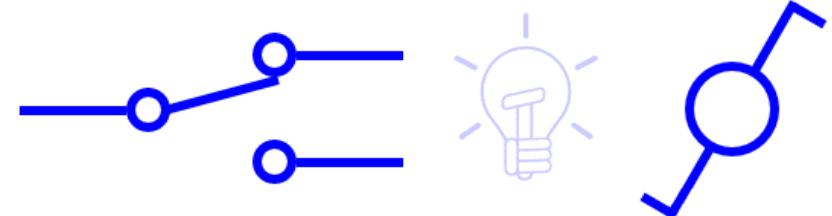
- A **simple ON/OFF switch** commonly found in our homes for lighting circuits and small load appliances as well as computers and devices.
- It is also called a **“One Way” or “Single Way” Switch**
- It controls single operation in a circuit
- The contact of the SPST switch can be either **NO (Normally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



Mechanical Switch

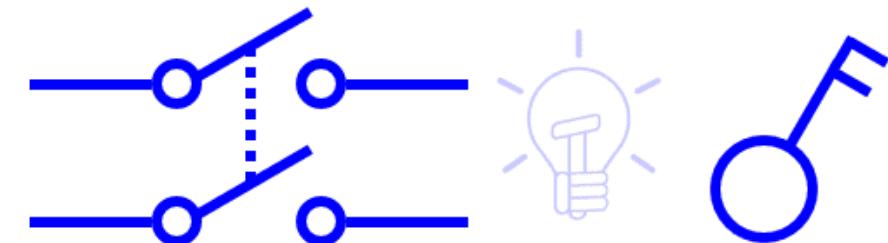
SPDT (Single Pole Double Throw)

- 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.



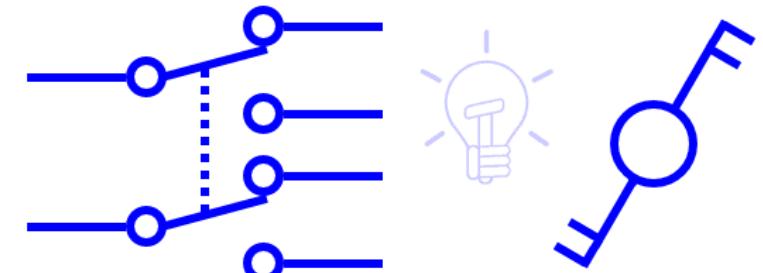
Mechanical Switch

DPDT (Double Pole Double Throw)

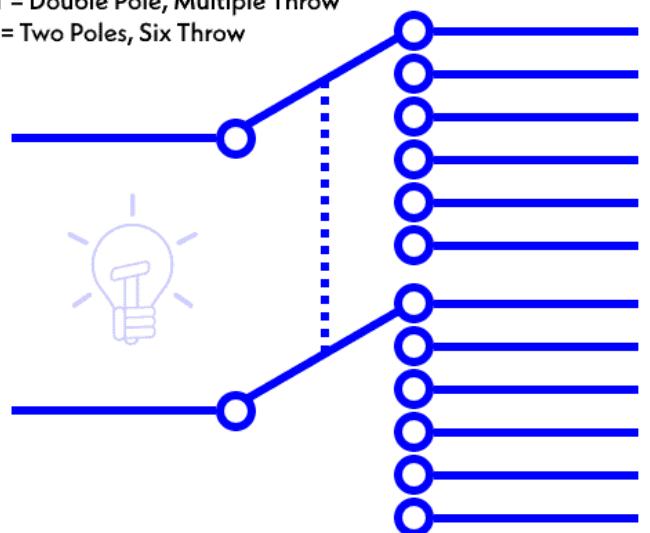
- DPDT switch is equivalent to **two SPDT switches packaged in one unit.**
- This switch has **two common pins** and **four signal pins** (total of 6 terminals).
- Total four different combinations of signals can be applied to the input pins of this switch.
- DPDT switches are used to **control two different 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.

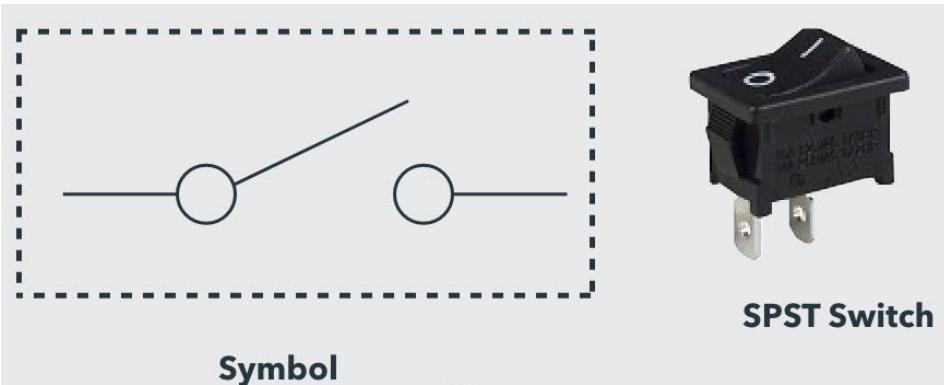


DPMT = Double Pole, Multiple Throw
2P6T = Two Poles, Six Throw

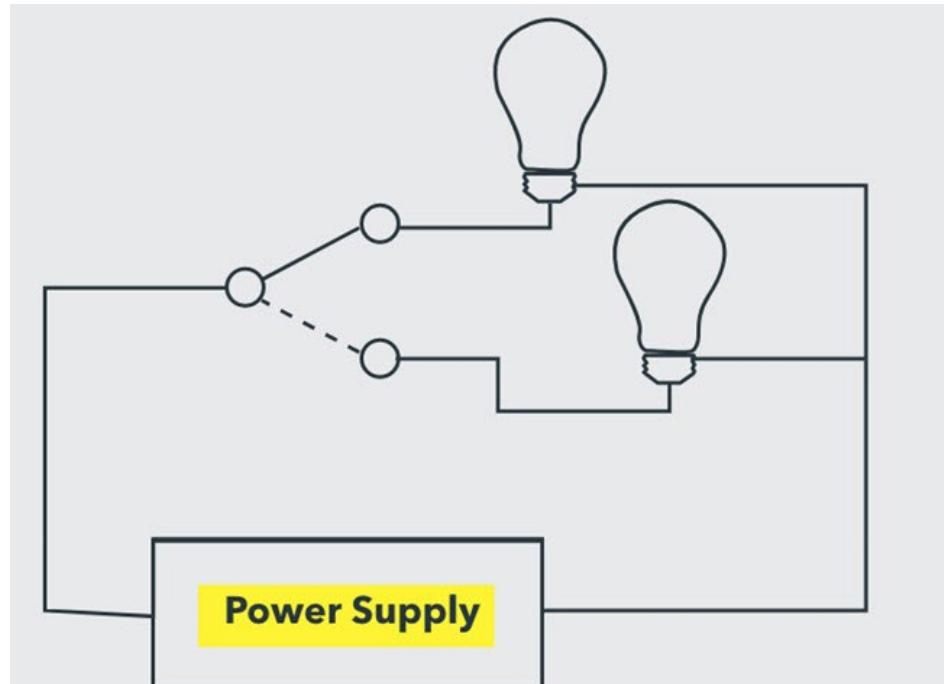
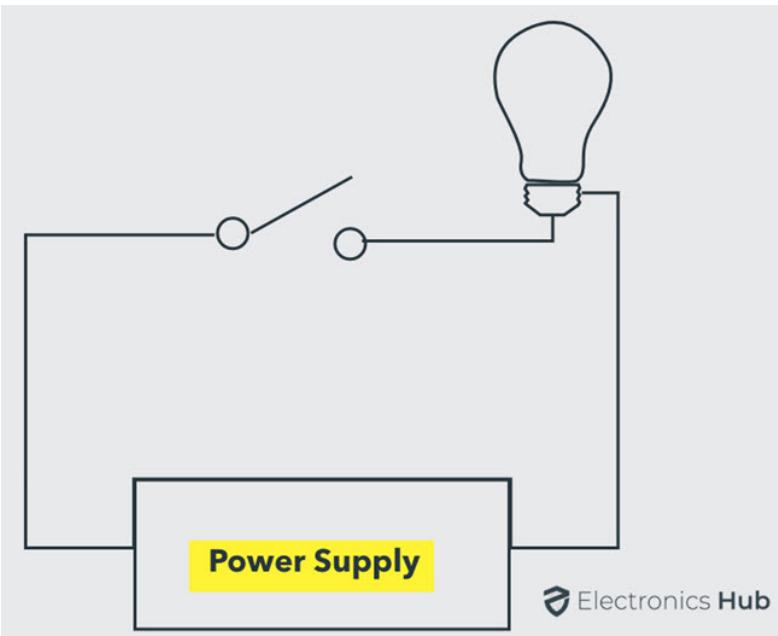
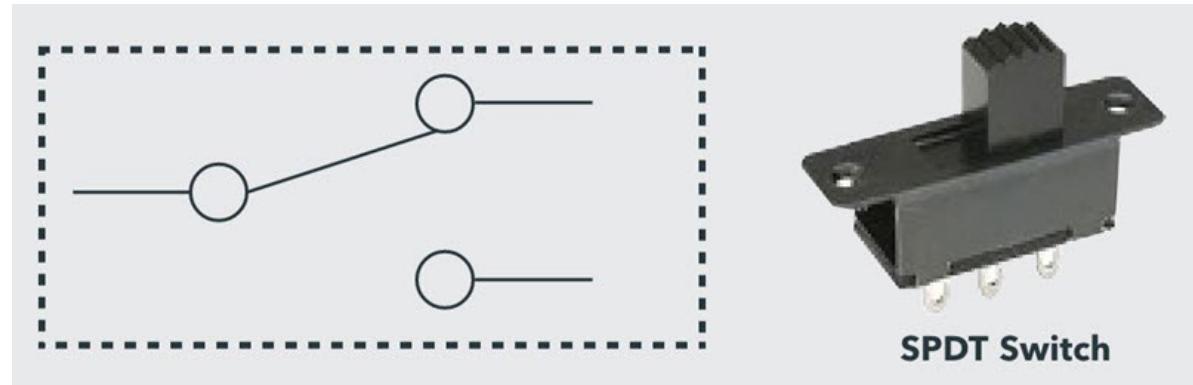


Mechanical Switch

Single Pole Single Throw Switch (SPST)

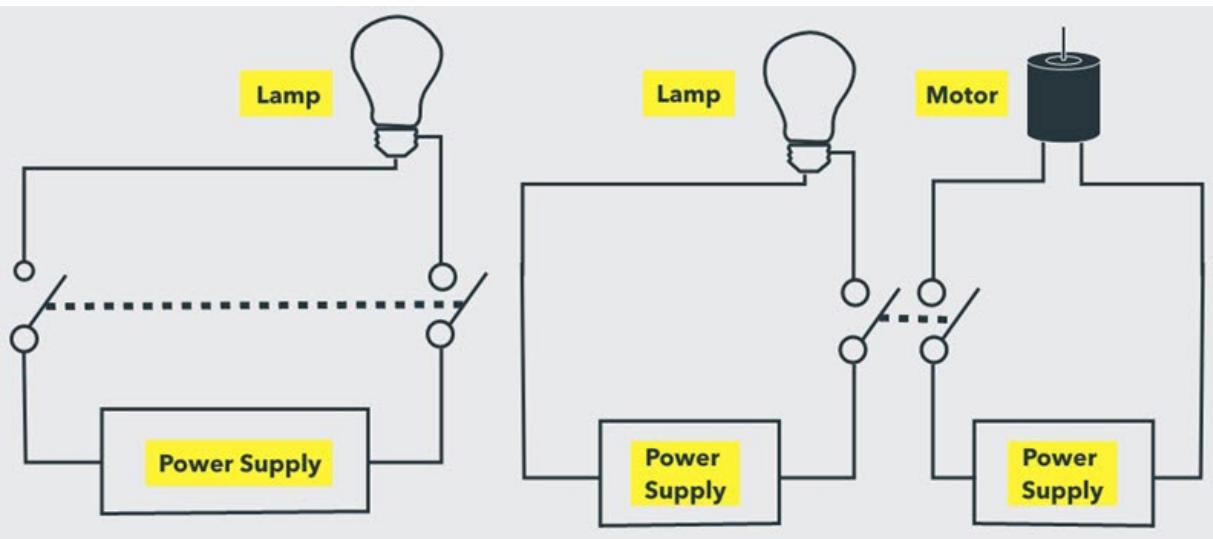
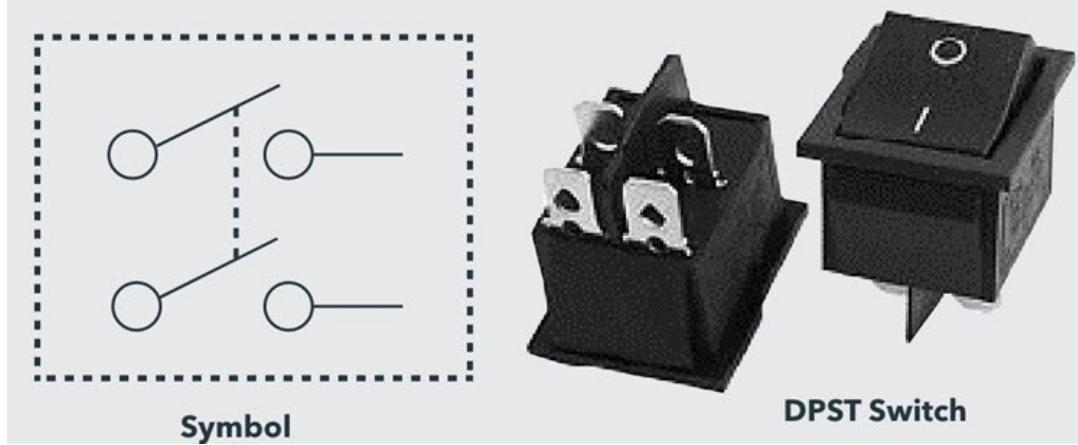


Single Pole Double Throw Switch (SPDT)



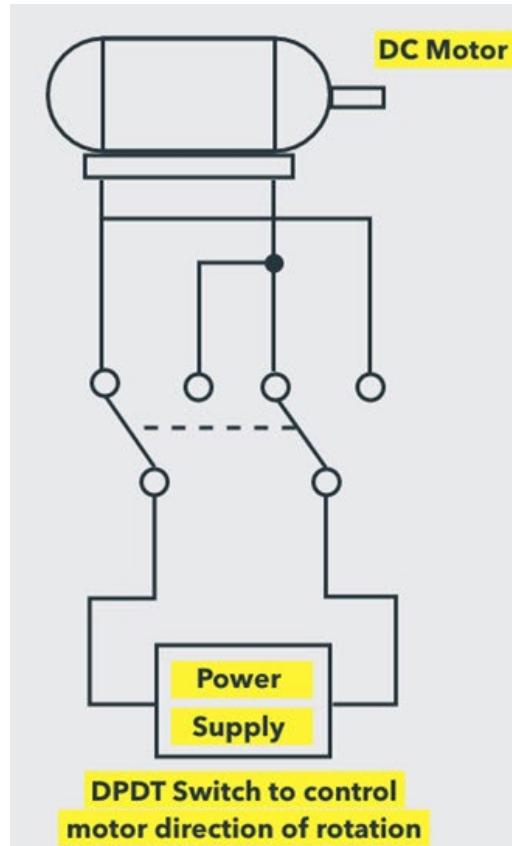
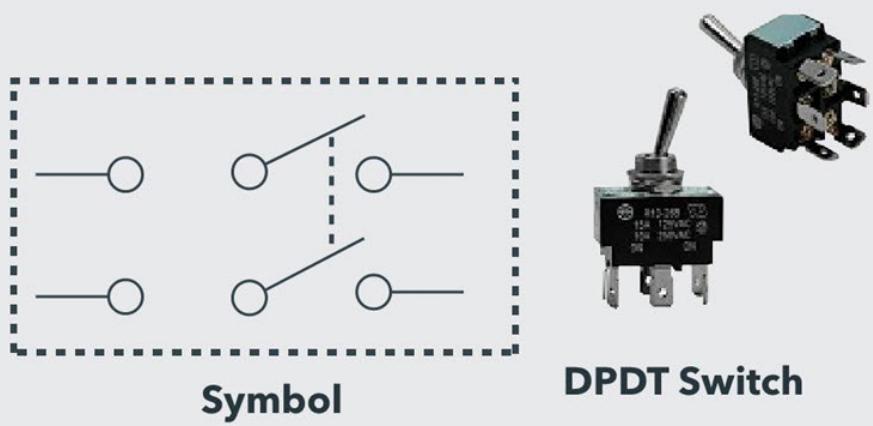
Mechanical Switch

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.

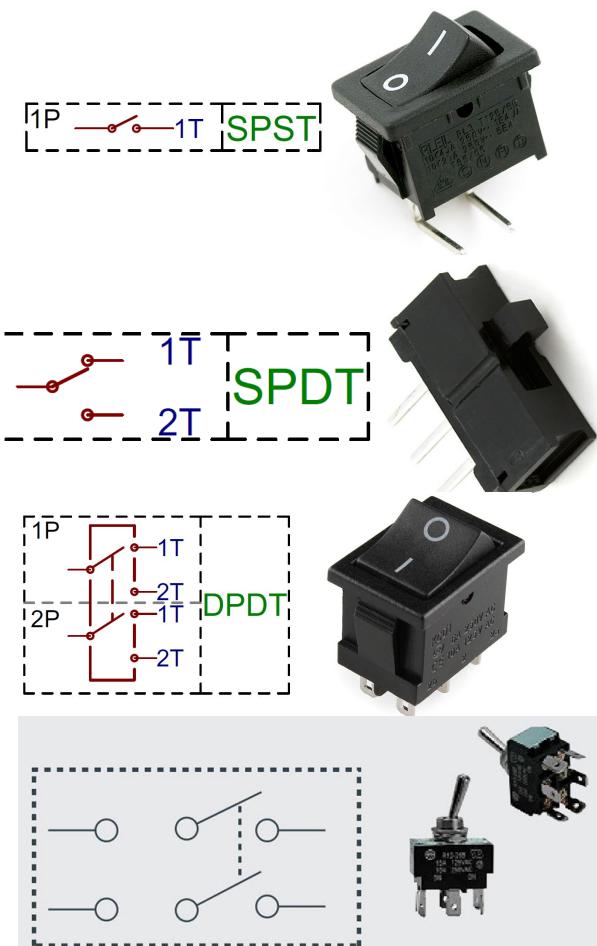
Double Pole Double Throw Switch (DPDT)



- 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

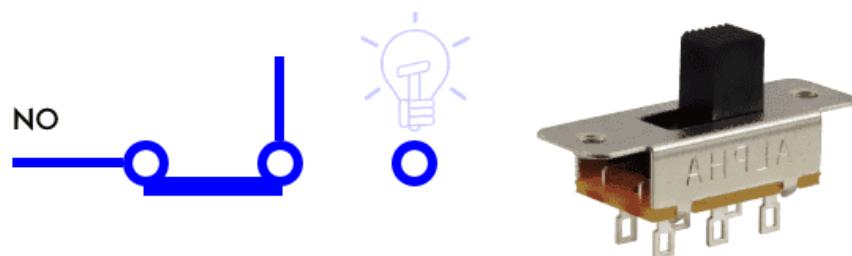
Toggle Switch

- 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.

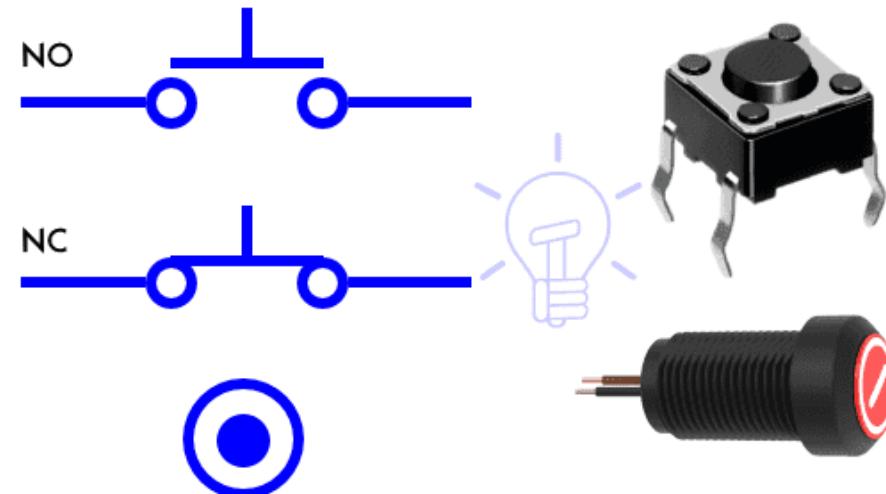


SPDT (single pole double throw) slide switch

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.

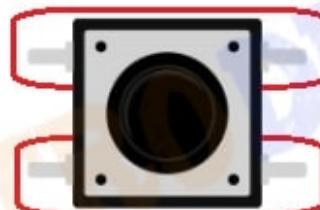


Push Button Switch

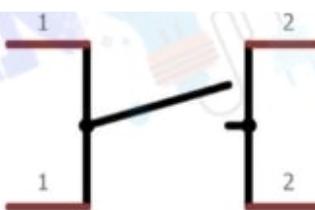
Push-Buttons (Momentary Switch) are normally-open tactile switches. When we pressed the button it makes the circuit connected and when released the button it makes the circuit connection breaks. We can see from outside this switch is consists of four terminals. but both side of the terminal is internally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



internally connected pin 1



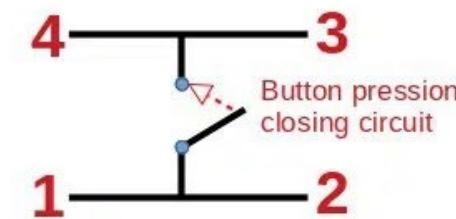
internally connected pin 2



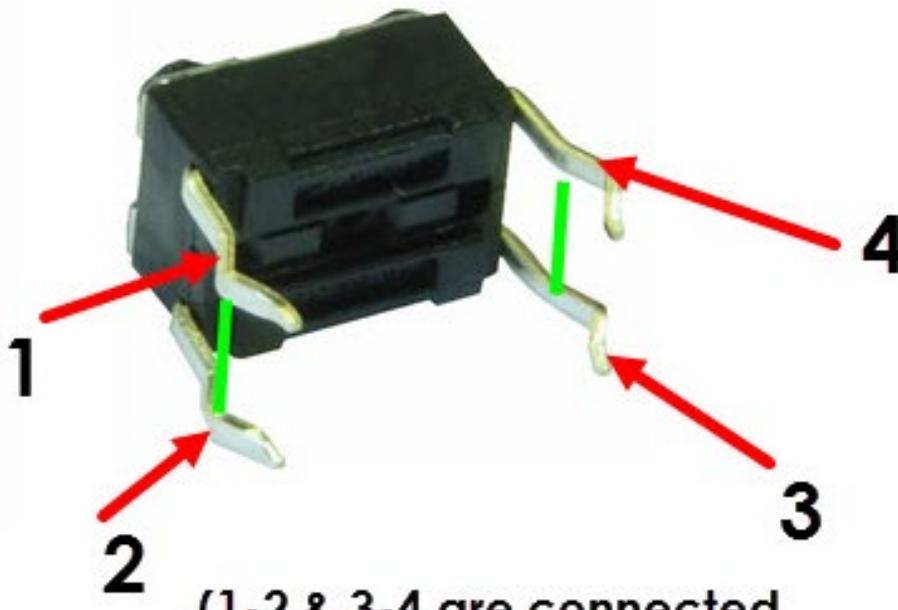
Push Button Switch

Internal Connection

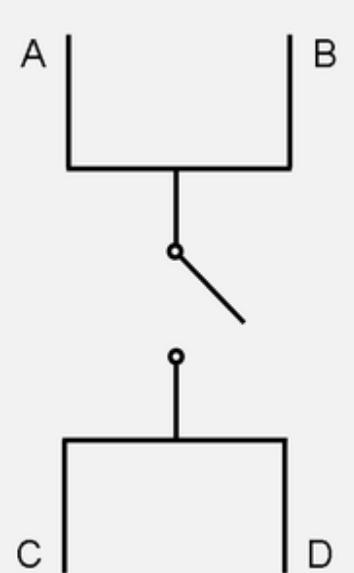
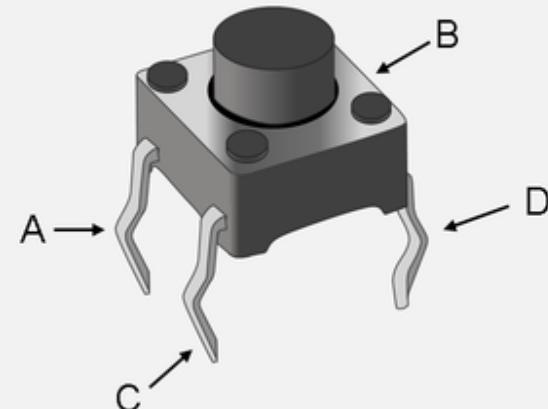
Symbol



Push Button Switch



(1-2 & 3-4 are connected)

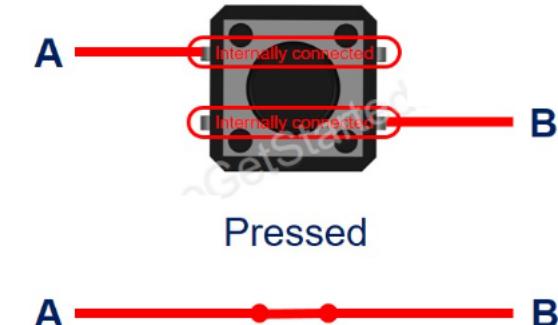


How It Works

- 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



Not Press



Pressed

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?

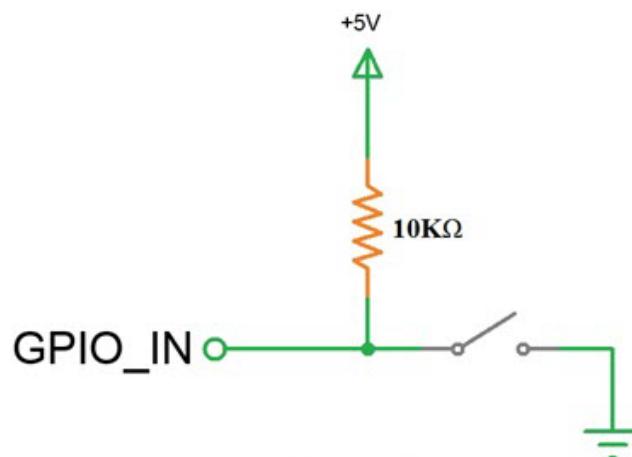
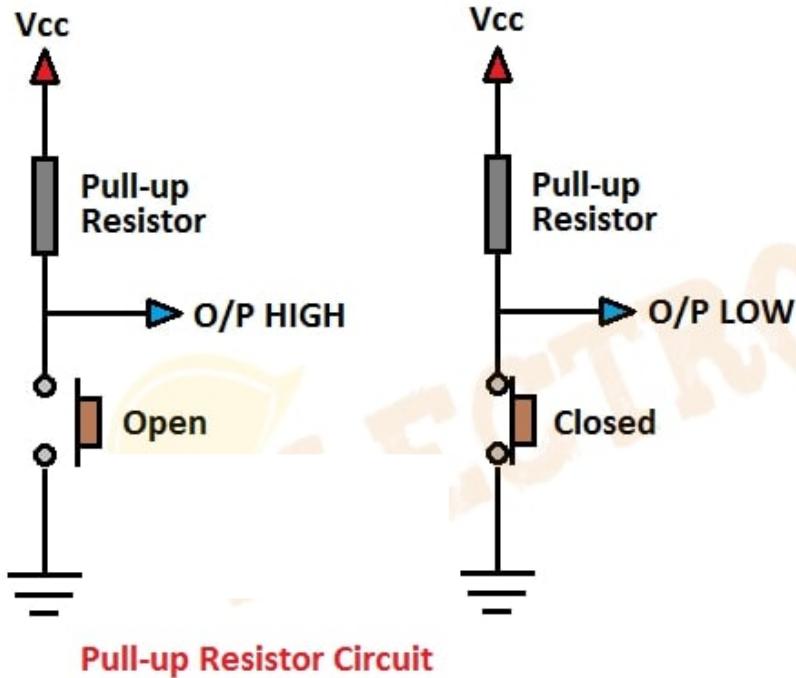
⇒ To make it stand firmly in PCB (board) to resist the pressing force.

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



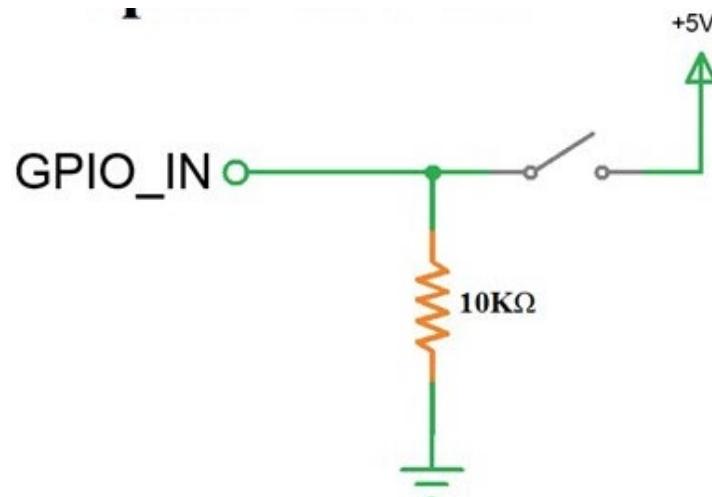
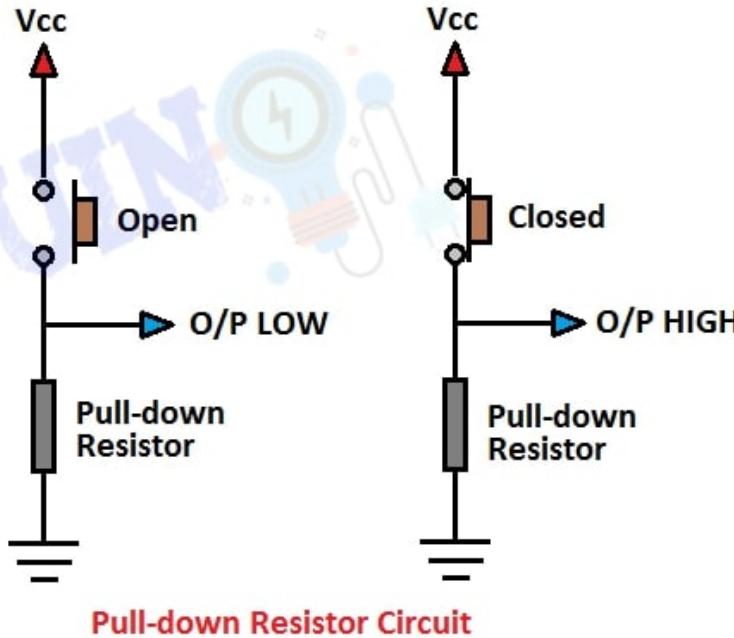
If we connect the push button switch directly to Raspberry Pi to get digital input, It means switch one pin is connected to Ground or 5v Vcc and another pin connected to Raspberry Pi digital pin. In this case, the Raspberry Pi is read **unstable input** from the push button.

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.

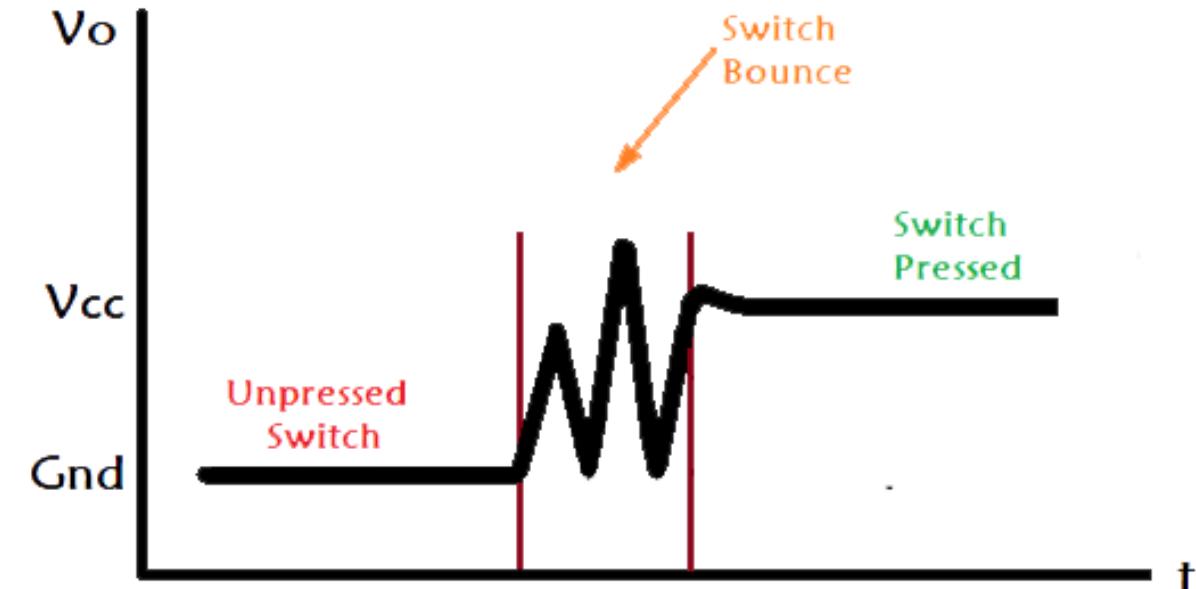
- ✓ **Symptom:** Button is pressed once, but Raspberry Pi code detects several times.
- ✓ **Cause :** Due to mechanical and physical issues, the state of button (or switch) is quickly toggled between **LOW** and **HIGH** several times.
- ✓ **Solution:** Debounce

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.

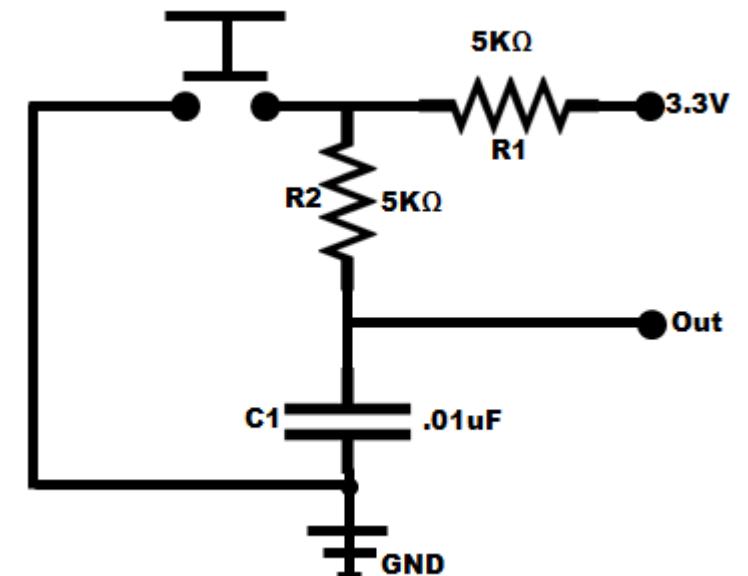
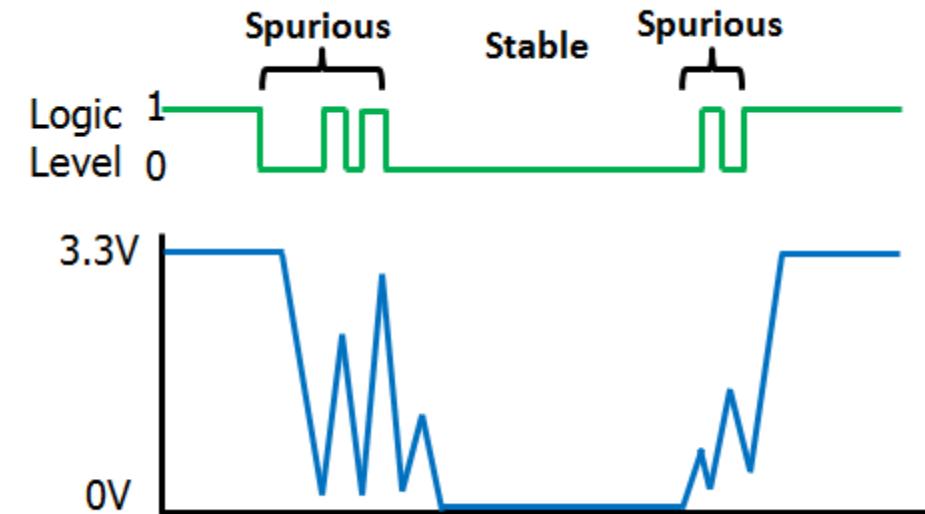


Trouble-shooting Pull-Up and Pull-Down Resistor

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.

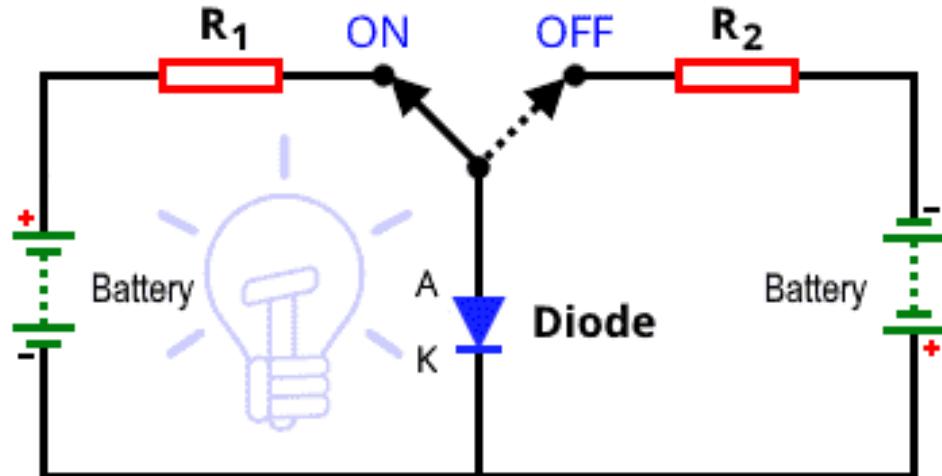
Electrical & Electronic Switches

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.

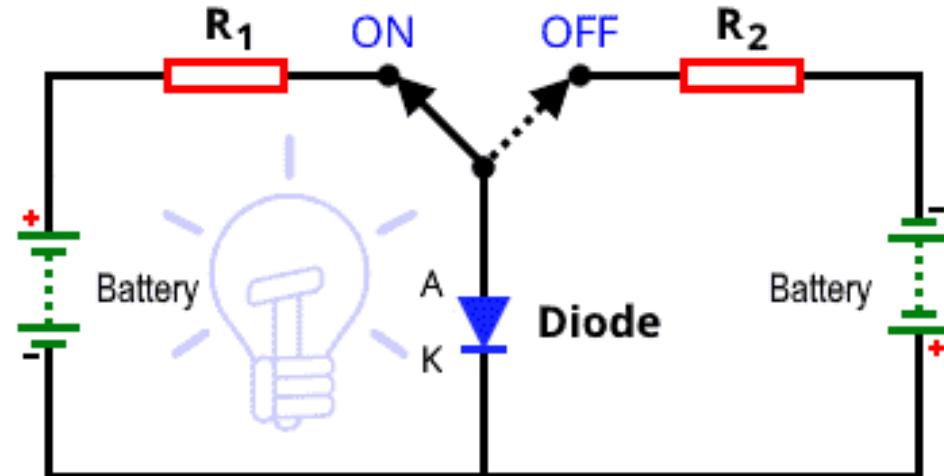
- Forward Bias: If positive signal applied to the Anode terminal and cathode is negative, the diode is forward biased, hence it acts as a closed switch.
- Reverse Bias: If the negative signal applied to the Anode and cathode is positive, the diode is in reverse biased, thus it acts as an open switch.



Electrical & Electronic Switches

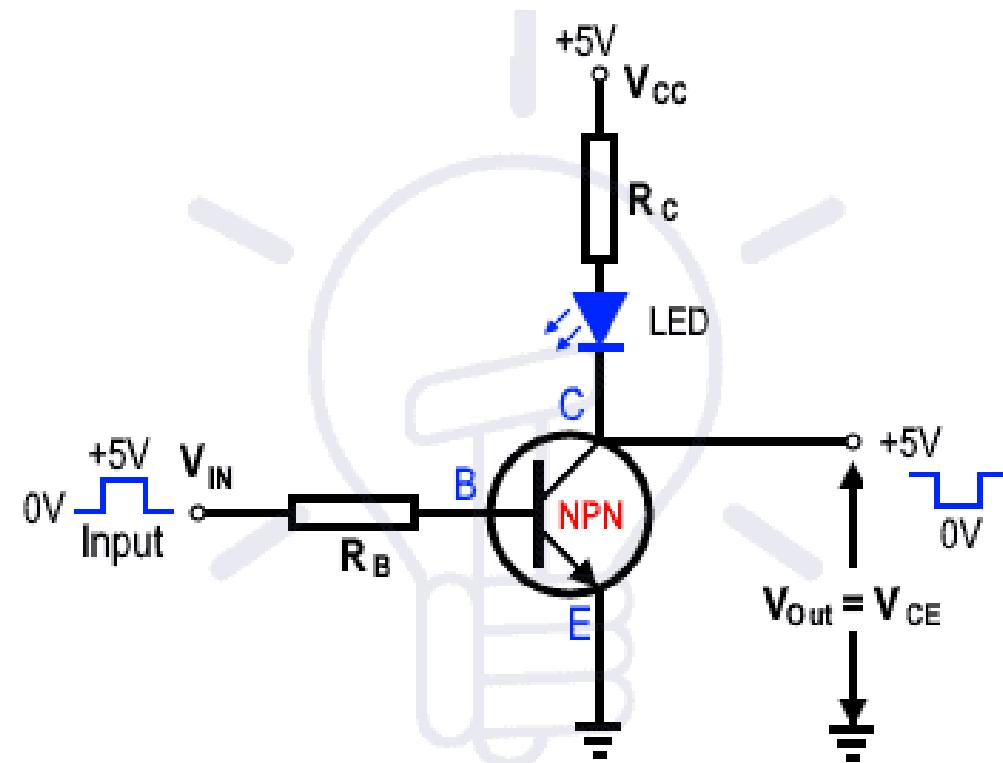
BJT Transistors as a switch:

- 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

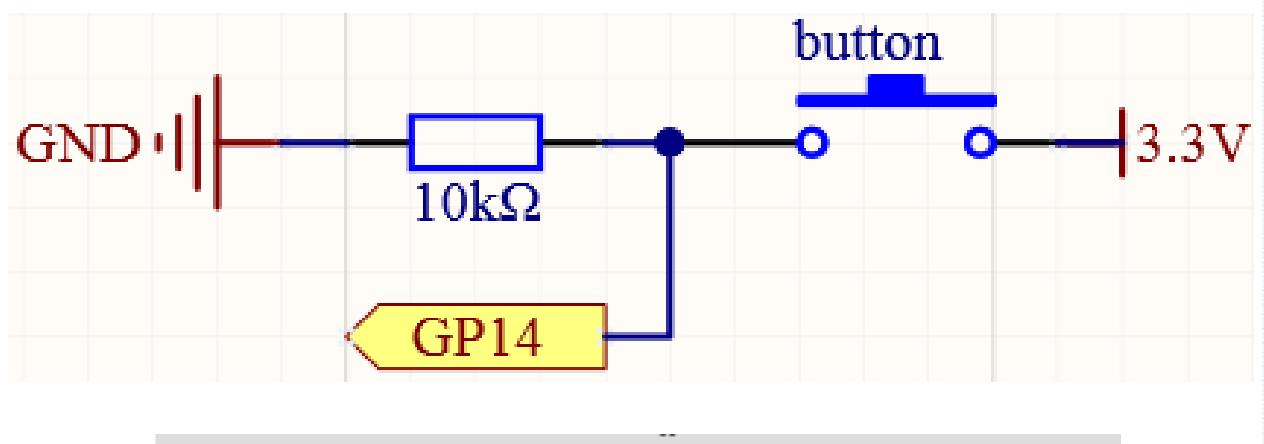
- When the input to the inverter is high "+5V/+3.3V", the NPN transistor is saturated and its output is low " $\approx 0V$ ". When the input to the inverter is low, the transistor is cut-off and its output is high.
- In the saturation region, it is "ON" like a closed switch
- In the cut-off region, it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



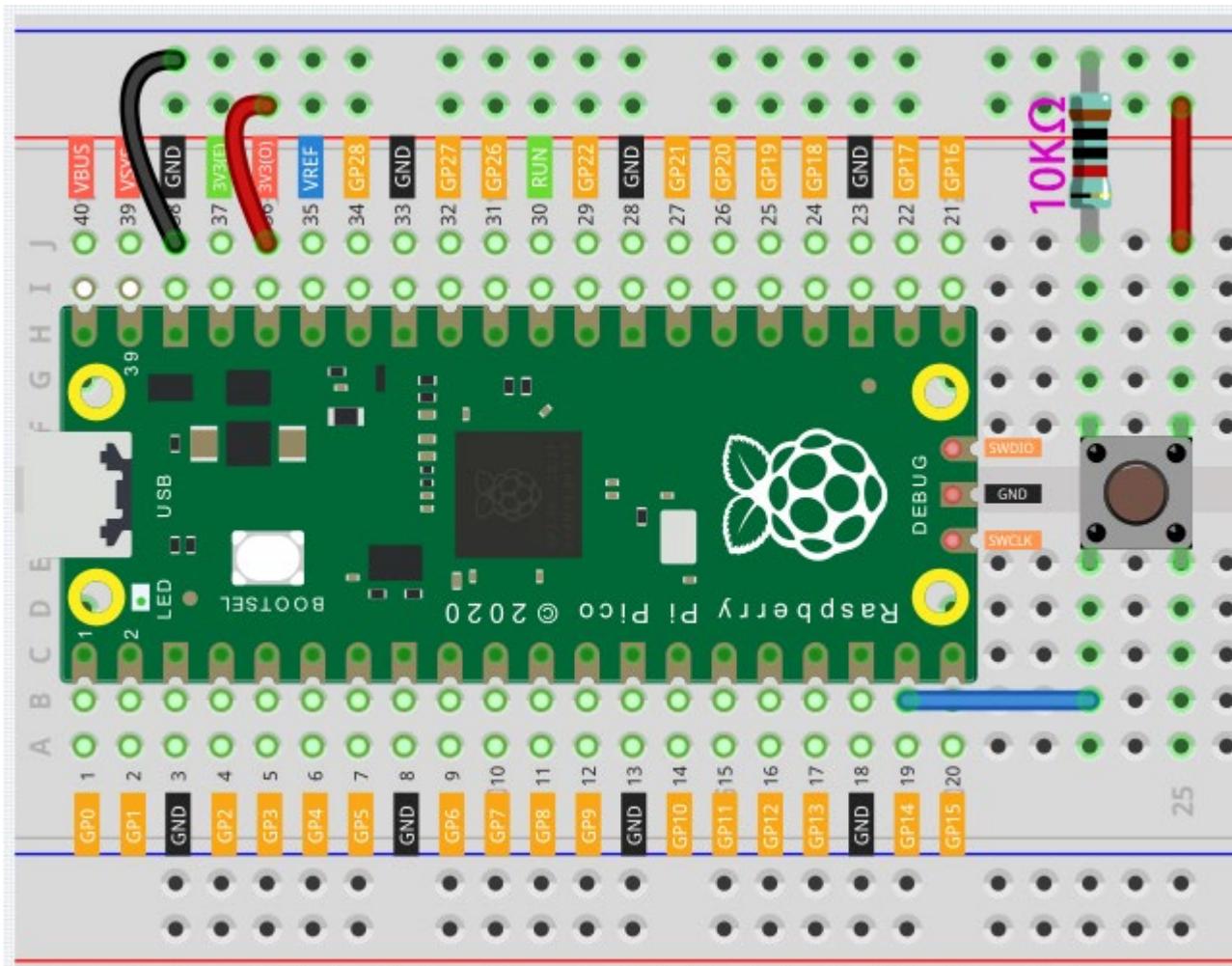
Reading Button Value (Pull-Down Mode)

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

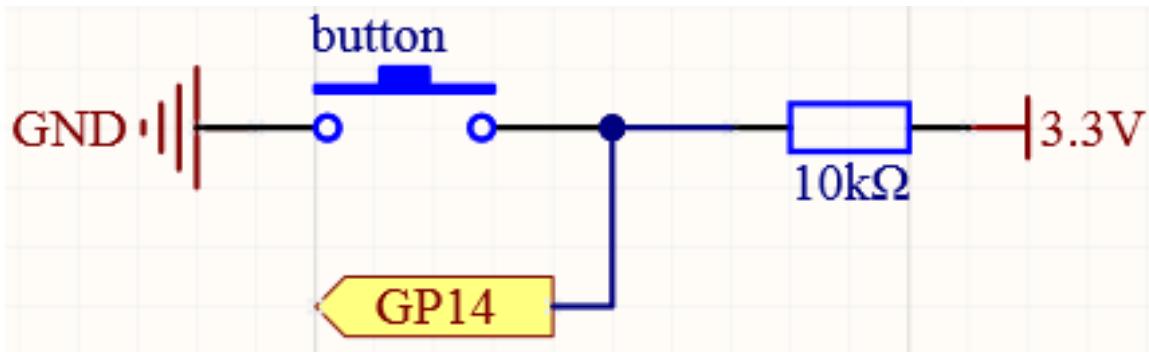
00:06.683 25%

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.

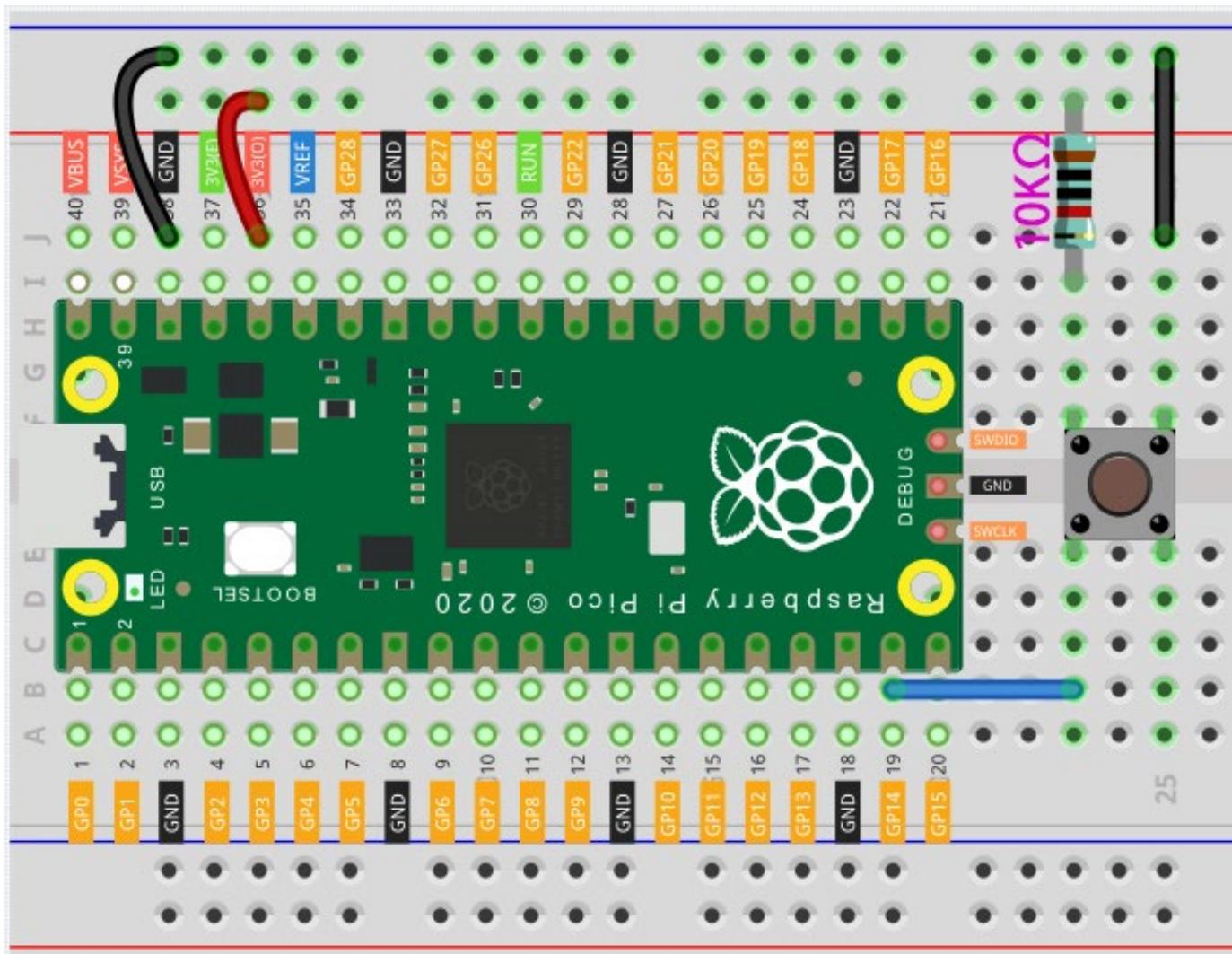
Pull-up Working Mode



```
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 diagram shows a Raspberry Pi Pico connected to a breadboard. A green push-button is connected between pin 14 and ground. A 10k pull-up resistor is connected between pin 14 and VDD. The Pico's pin 14 is connected to digital input D0. The simulation window shows the Pico's pins and the breadboard connections. A red box highlights the connection from the button to the Pico. The text "You pressed the button!" is displayed in the simulation output.

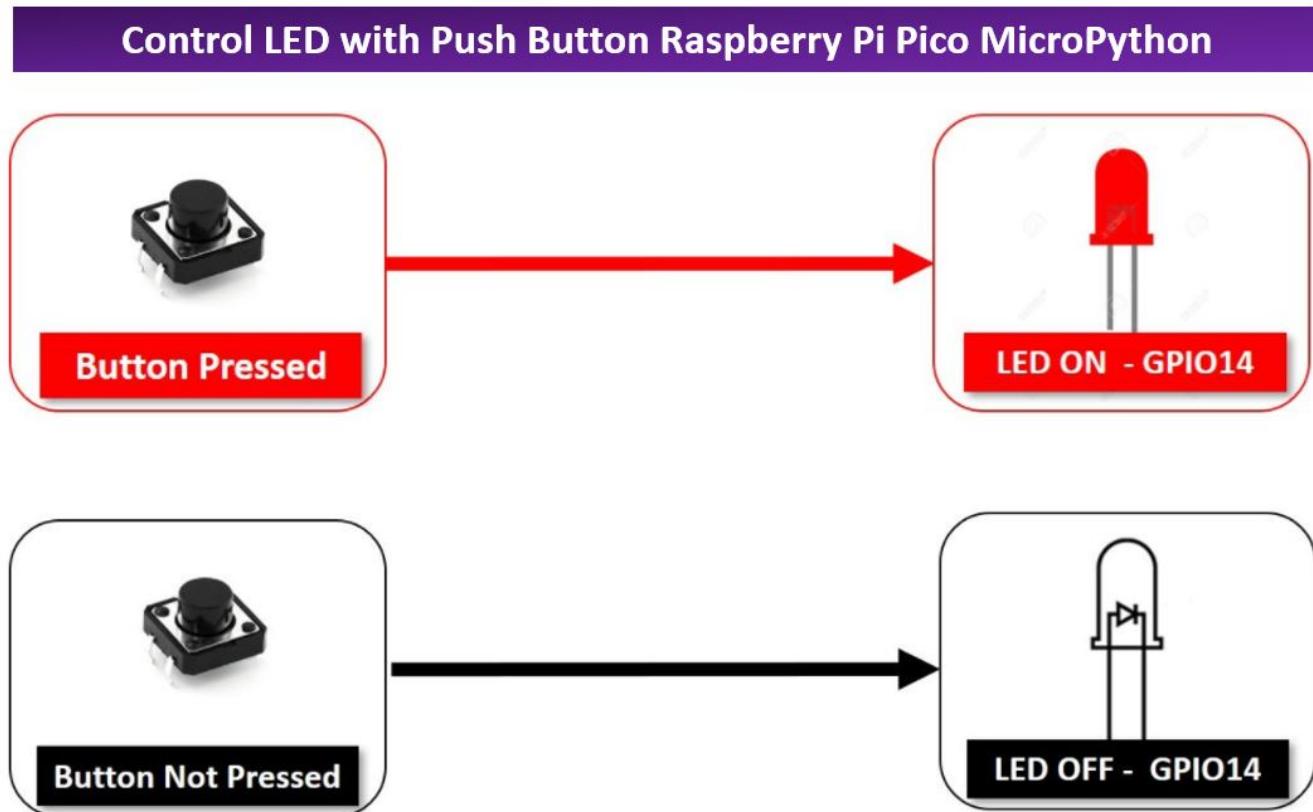
00:04.316 19%

You pressed the button!

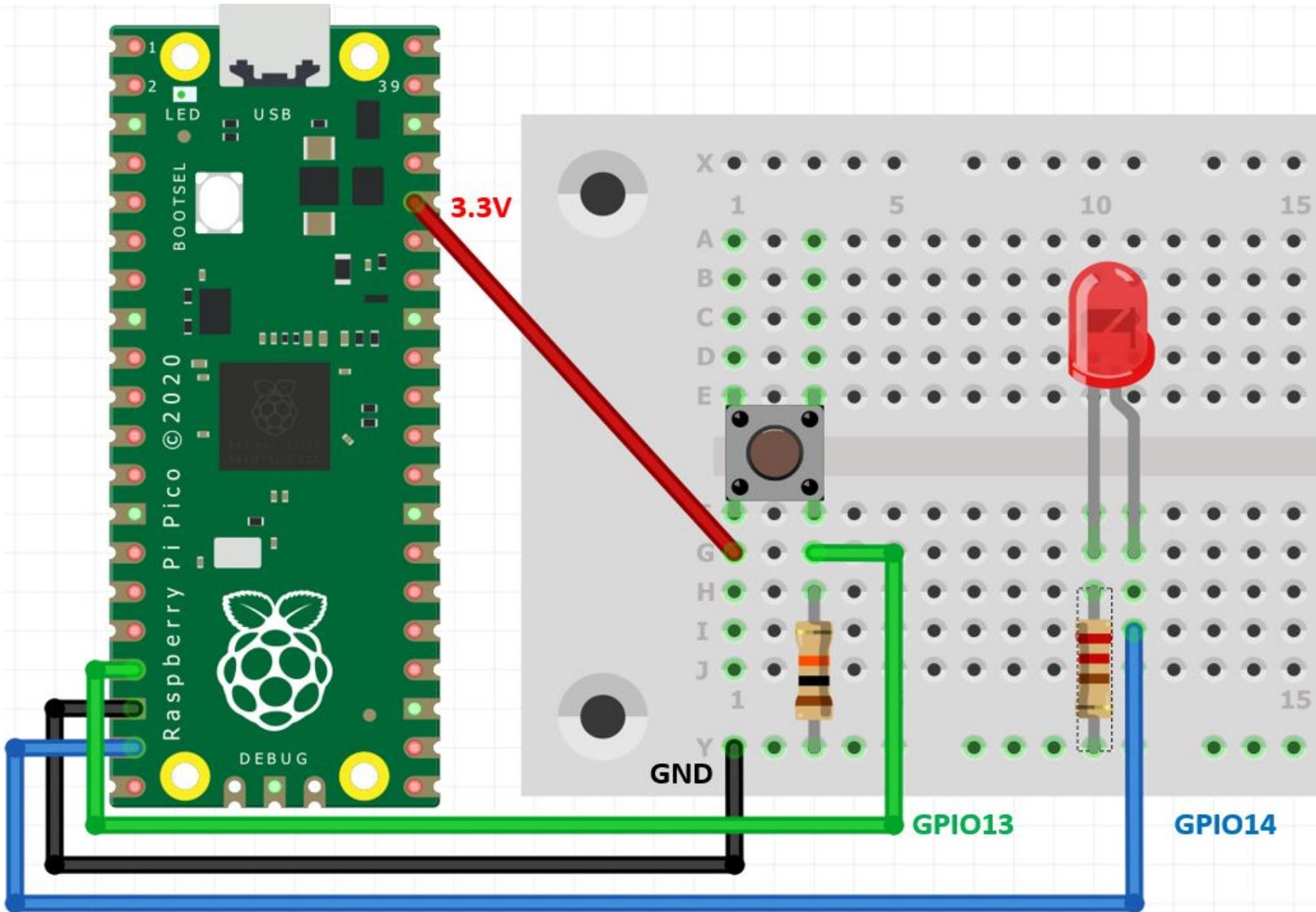
Interface Push Button with Raspberry Pi Pico and Control LED

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

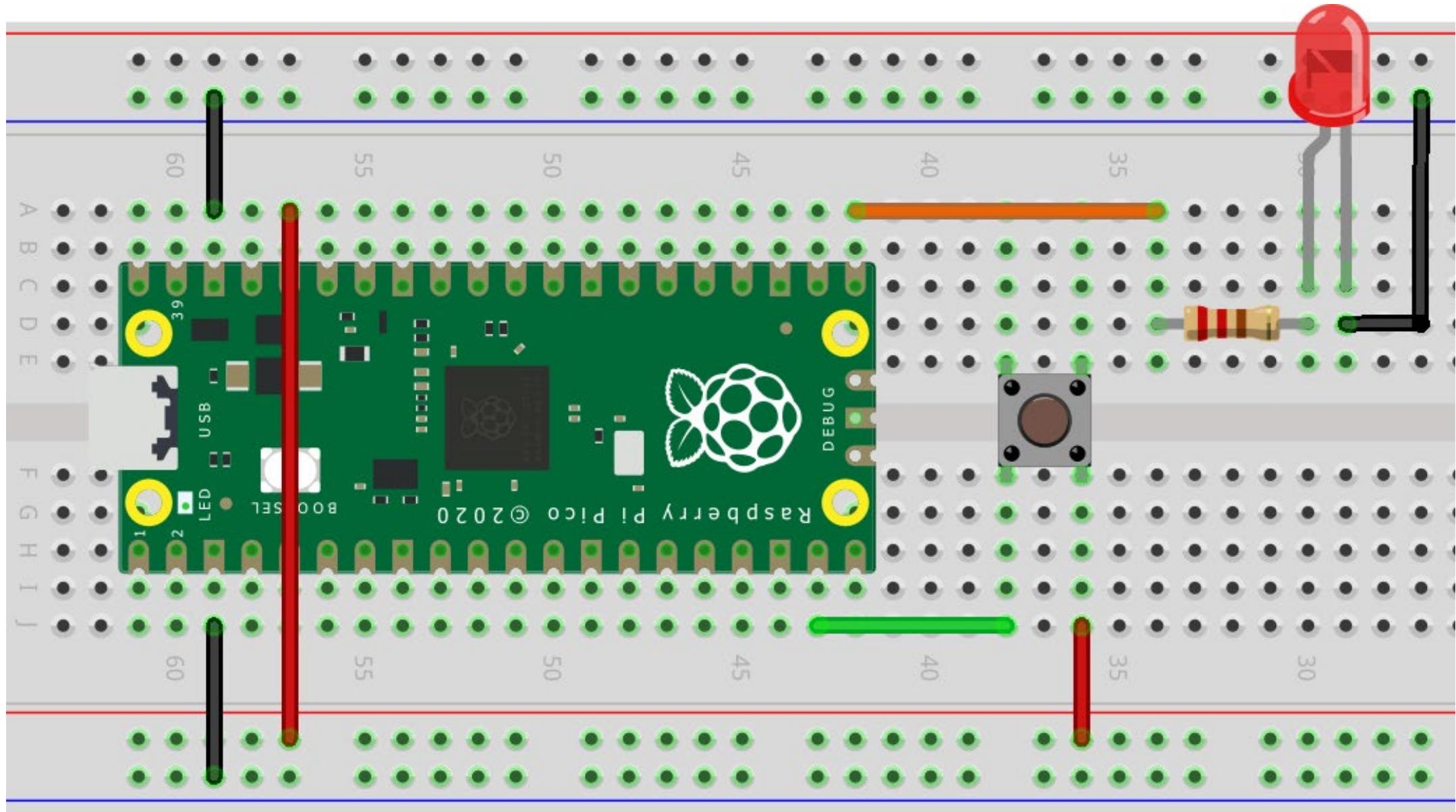
main.py • diagram.json PIO

```
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 🐍

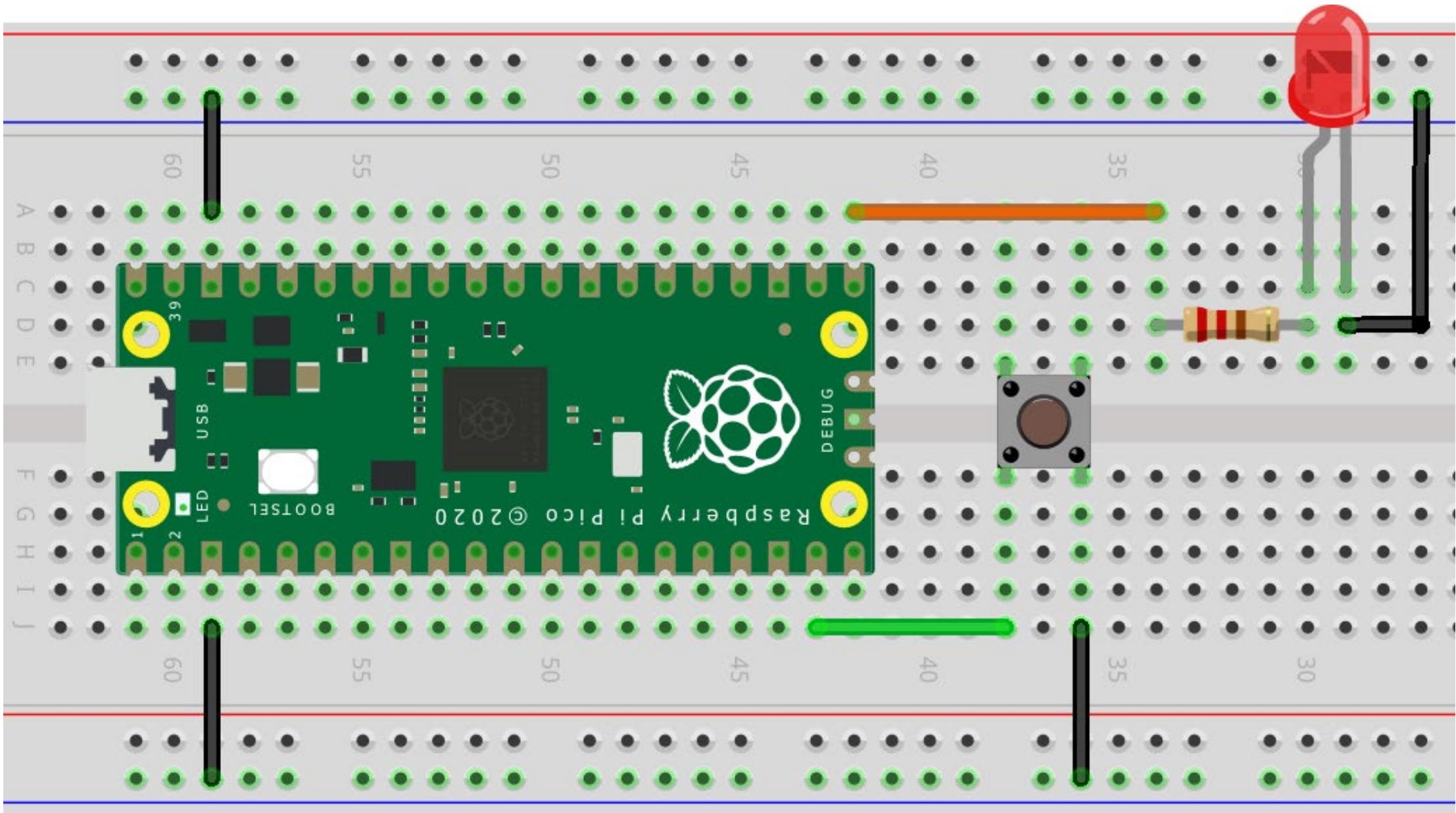
```
1  from machine import Pin
2  from utime import sleep
3
4  button = Pin(14, mode = Pin.IN, pull = Pin.PULL_DOWN)
5  led = Pin(16, Pin.OUT)
6
7  while True:
8      if button.value() == 1:
9          led.value(1)
10         print("LED is ON")
11         sleep(0.3)
12     elif button.value() == 0:
13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

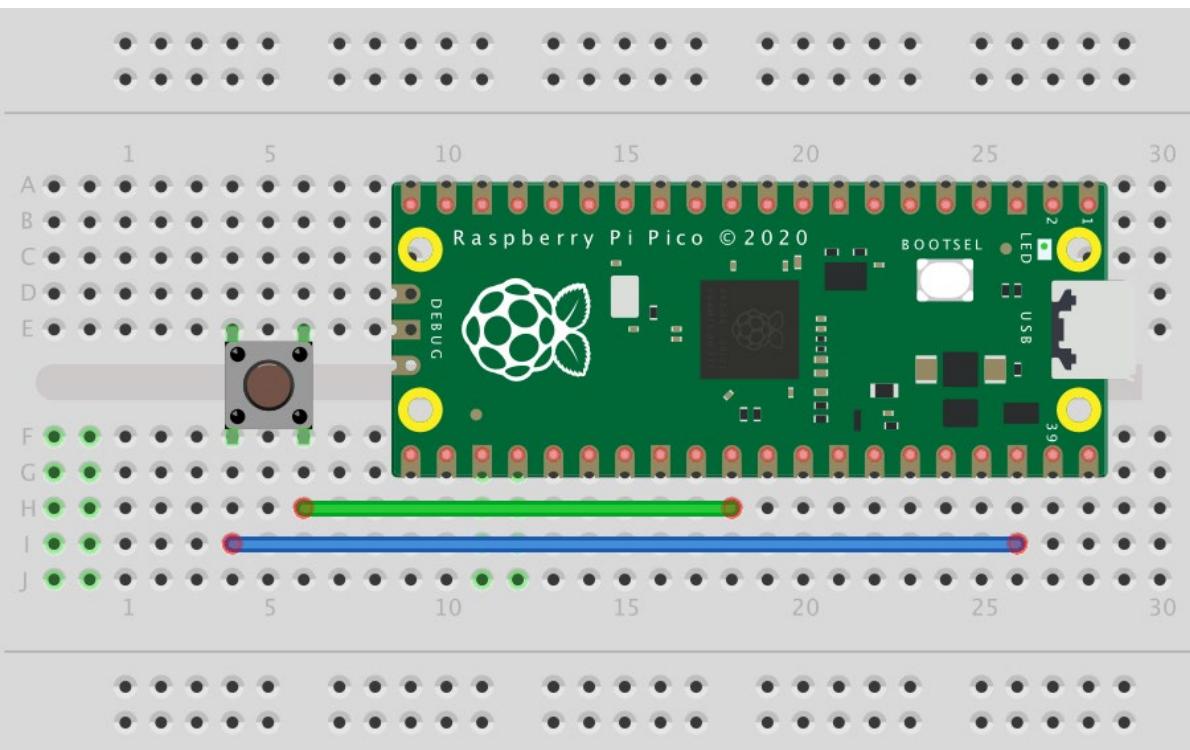
Simulation

00:23.913 99%

LED is ON
LED is ON

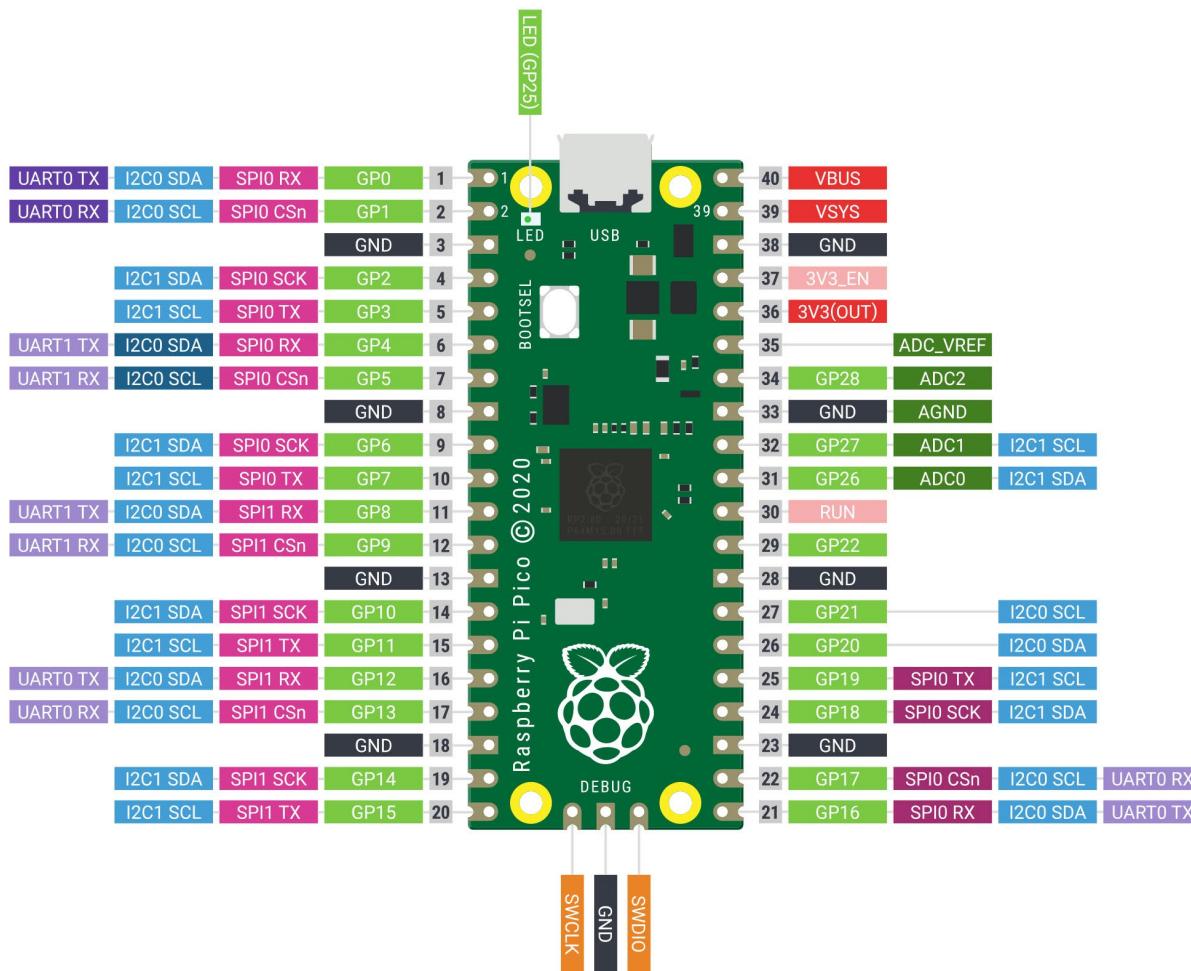
Push Button as reset button in Raspberry Pi Pico

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.

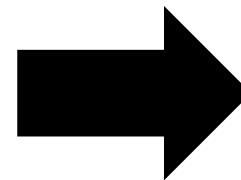
If we press the button on the breadboard it will pull the RUN pin down to 0V, forcing the Pico to reset. Press the Stop button to reconnect to the MicroPython shell. If your code ever locks up then this trick will help get you back to work.

Controlling an LED using a Push Button as Toggle Switch

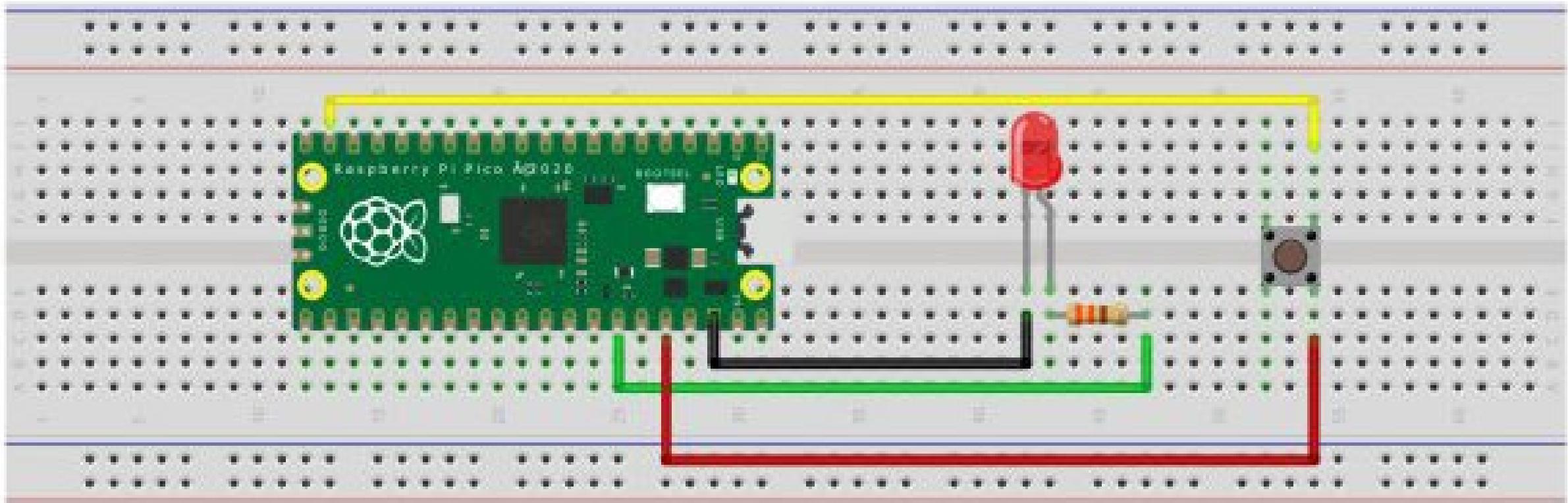
Objective:

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:

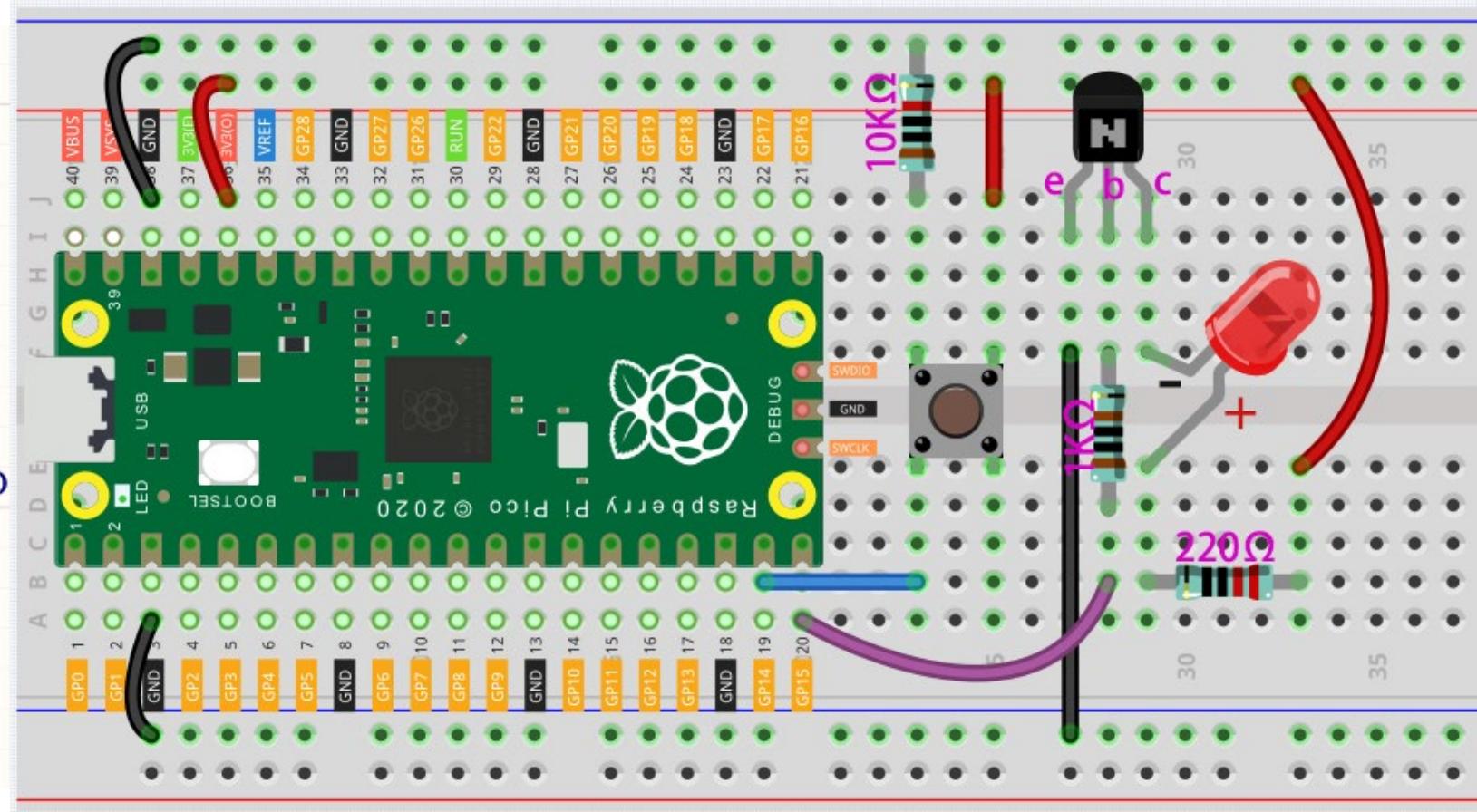
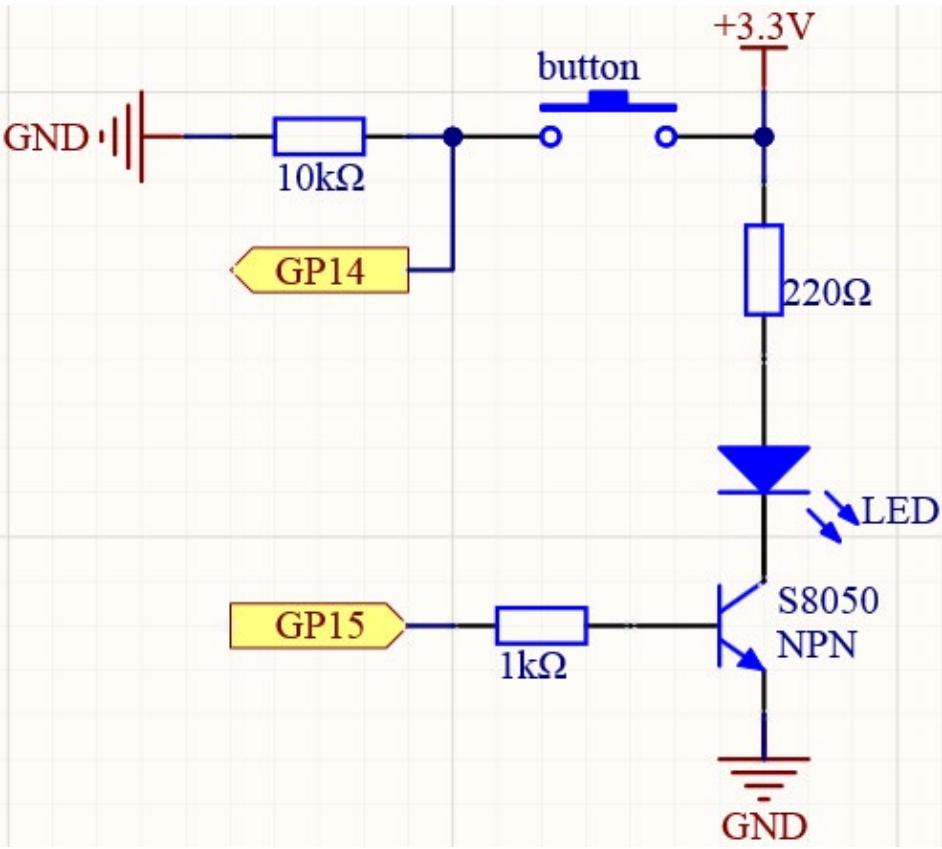
```
1 from machine import Pin
2 from utime import sleep
3
4 led = Pin(28, Pin.OUT)
5 button = Pin(14, Pin.IN, Pin.PULL_DOWN)
6
7 while True:
8
9     if button.value():
10        led.toggle()
11        sleep(0.5)
12
```

The right side of the interface features a simulation window titled 'Simulation'. It includes three control buttons: a green circular arrow for refresh, a grey square for stop, and a grey circle with a double vertical line for pause. Below these buttons is a schematic diagram of a Raspberry Pi Pico board. The board is shown with its component side facing up. A red LED is connected to pin 28, which is also connected to ground through a resistor. A push button is connected to pin 14, with its other terminal connected to ground via a pull-down resistor. The simulation shows the LED toggling between on and off states each time the button is pressed.

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.

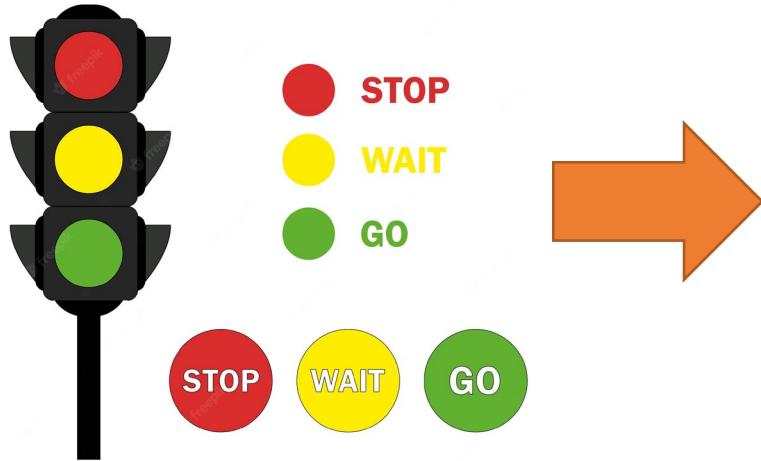
➤ Application of Transistor:

1) as an Amplifier

2) as a switch

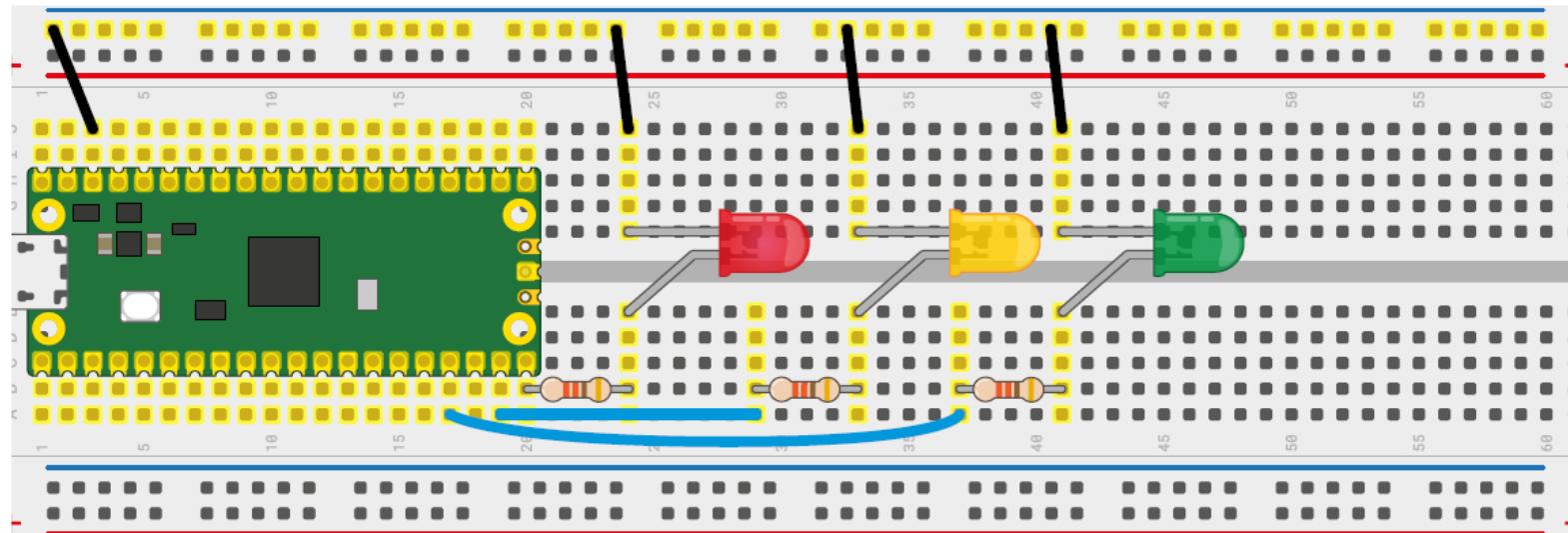
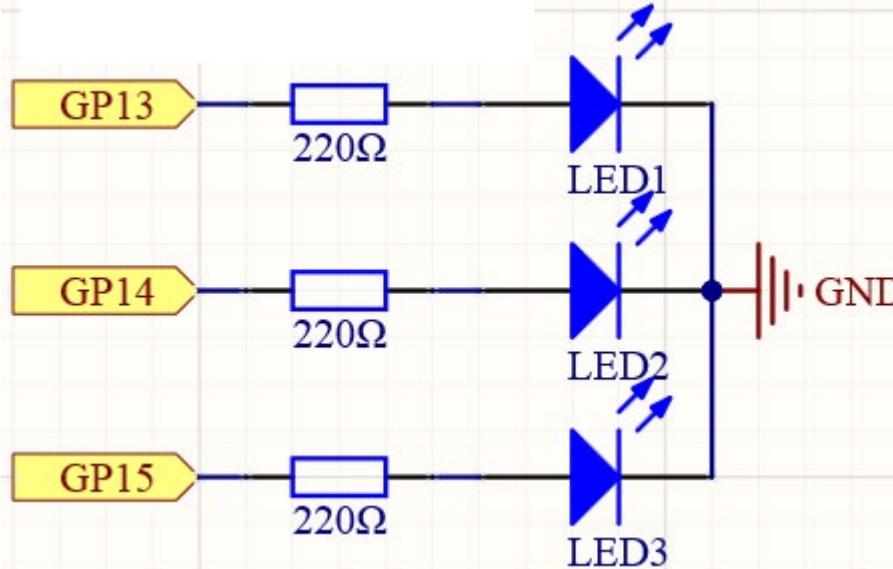
A basic three-light traffic light system

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



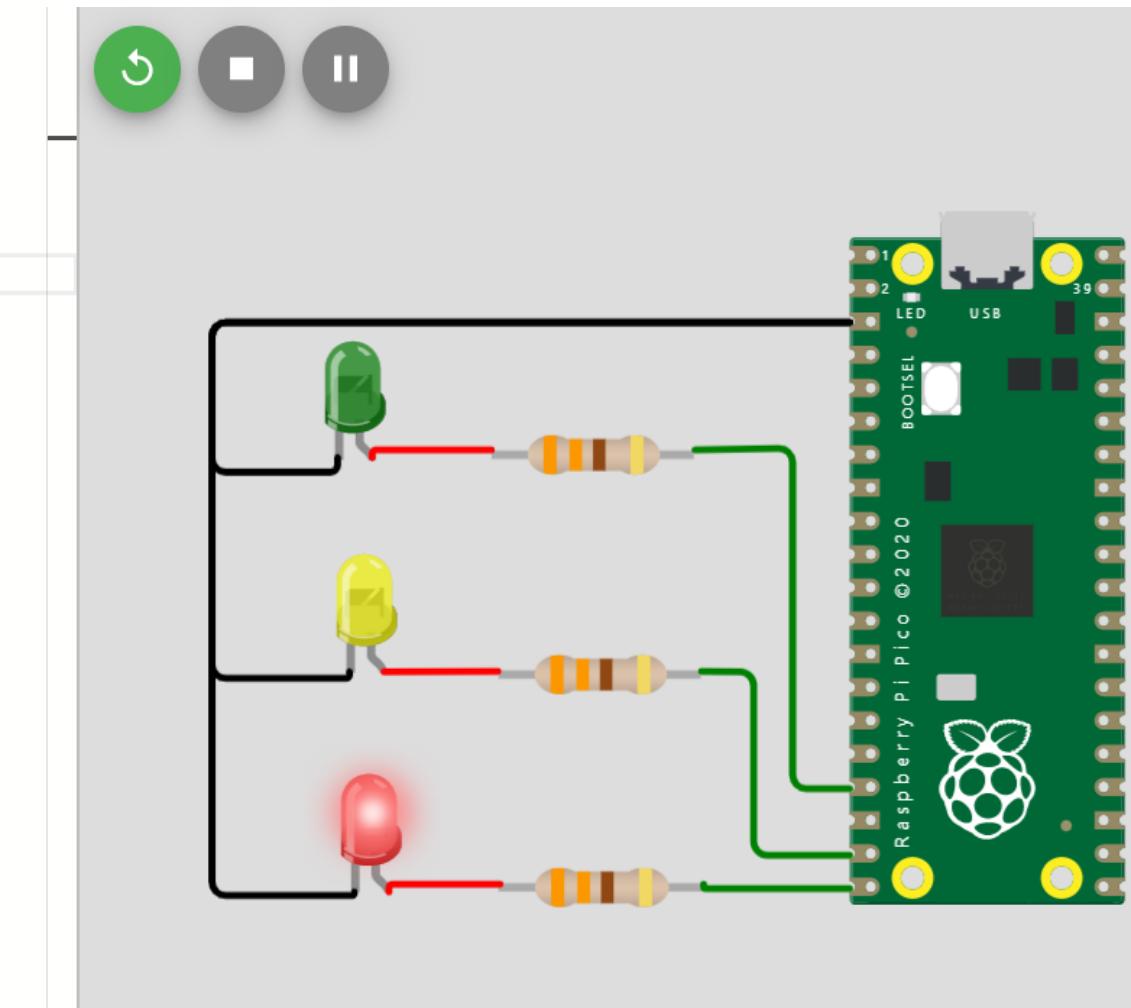
A simple traffic light Control system without proper control

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)

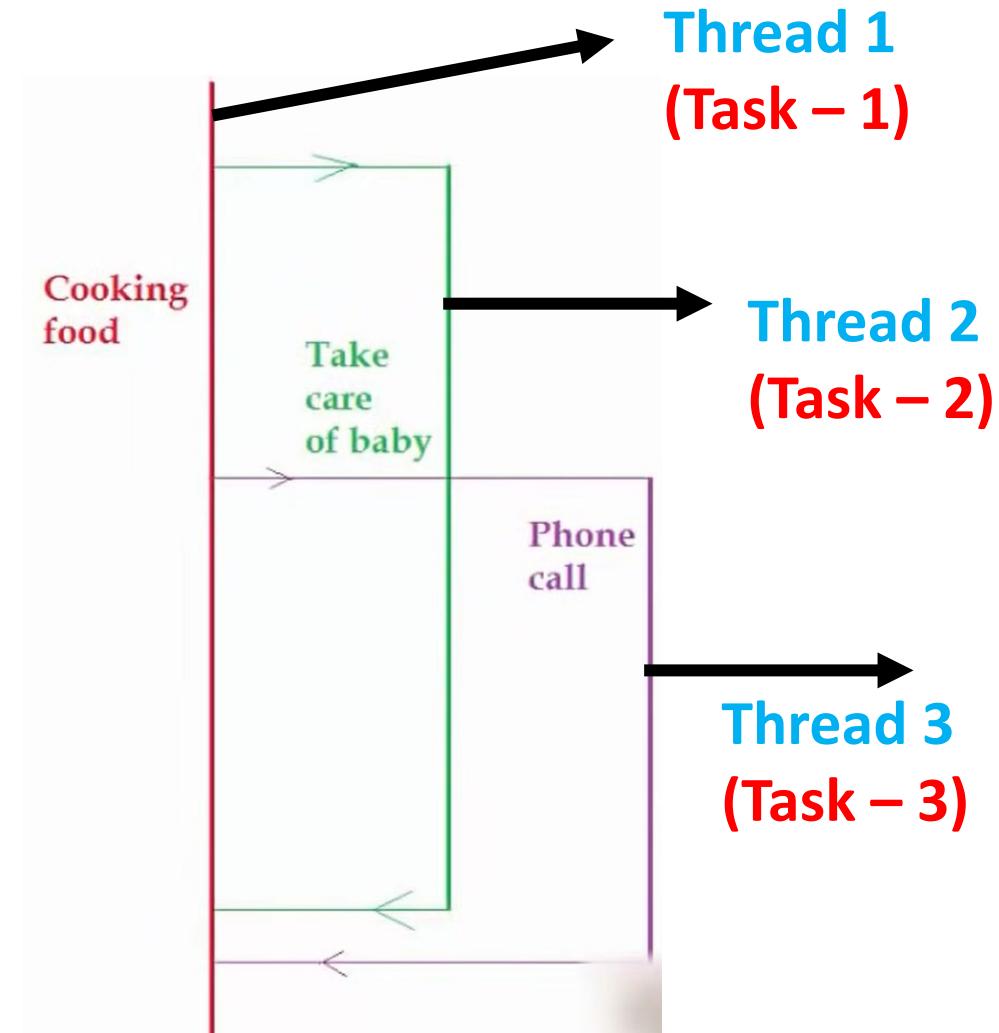
Through a push-button switch, the pedestrian can ask the lights to let them cross the road; and a buzzer, so the pedestrian knows when it's their turn to cross. **If we (pedestrians) press the button, the red LED will be extended to 15 seconds, which will give us more time to cross the road.**



Puffin Crossing (Mini pedestrian crossing system with proper control)

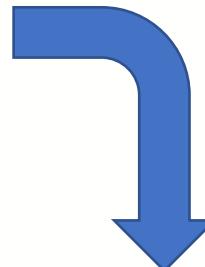
In puffin crossing our program should able to record whether the button has been pressed in a way that doesn't interrupt the traffic lights. To make that work, you'll need a new library: _thread.

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

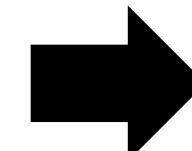
cube: 27

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

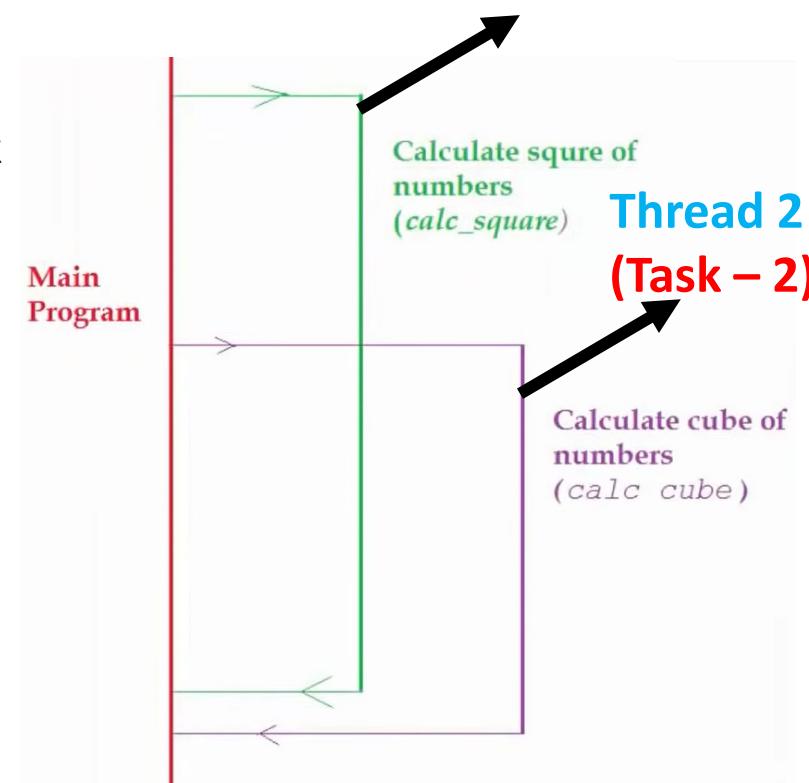
Thread 1
(Task - 1)

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:

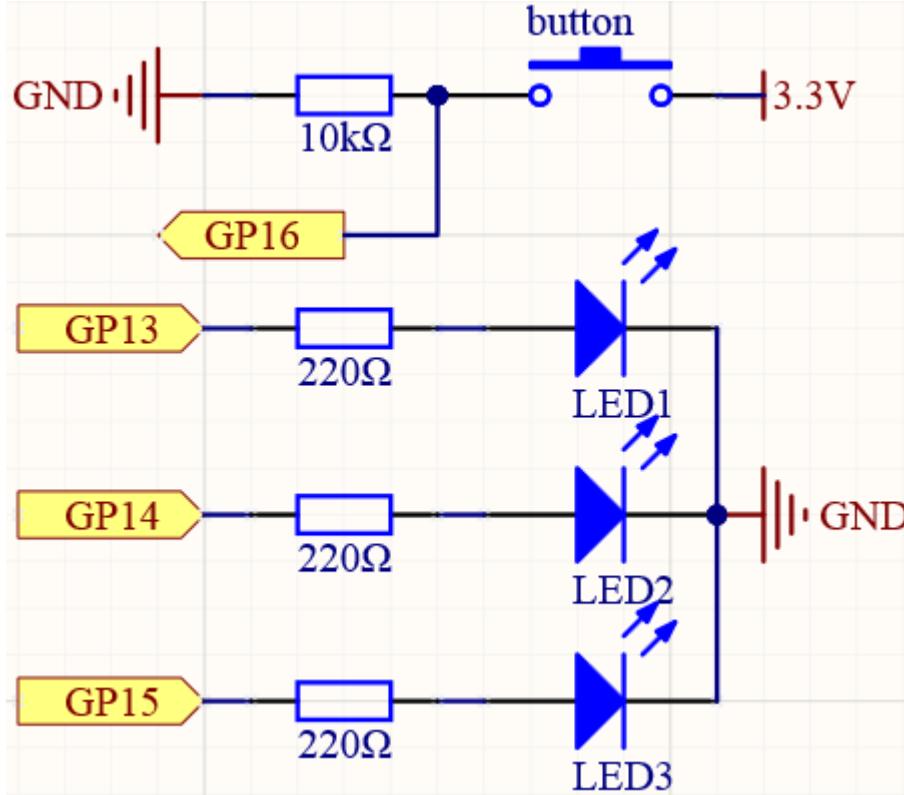
```
1 import time
2 import threading
3
4 def calc_square(numbers):
5     print("calculate square numbers")
6     for n in numbers:
7         time.sleep(1)
8         print('square:',n*n)
9
10 def calc_cube(numbers):
11     print("calculate cube of numbers")
12     for n in numbers:
13         time.sleep(1)
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!
```

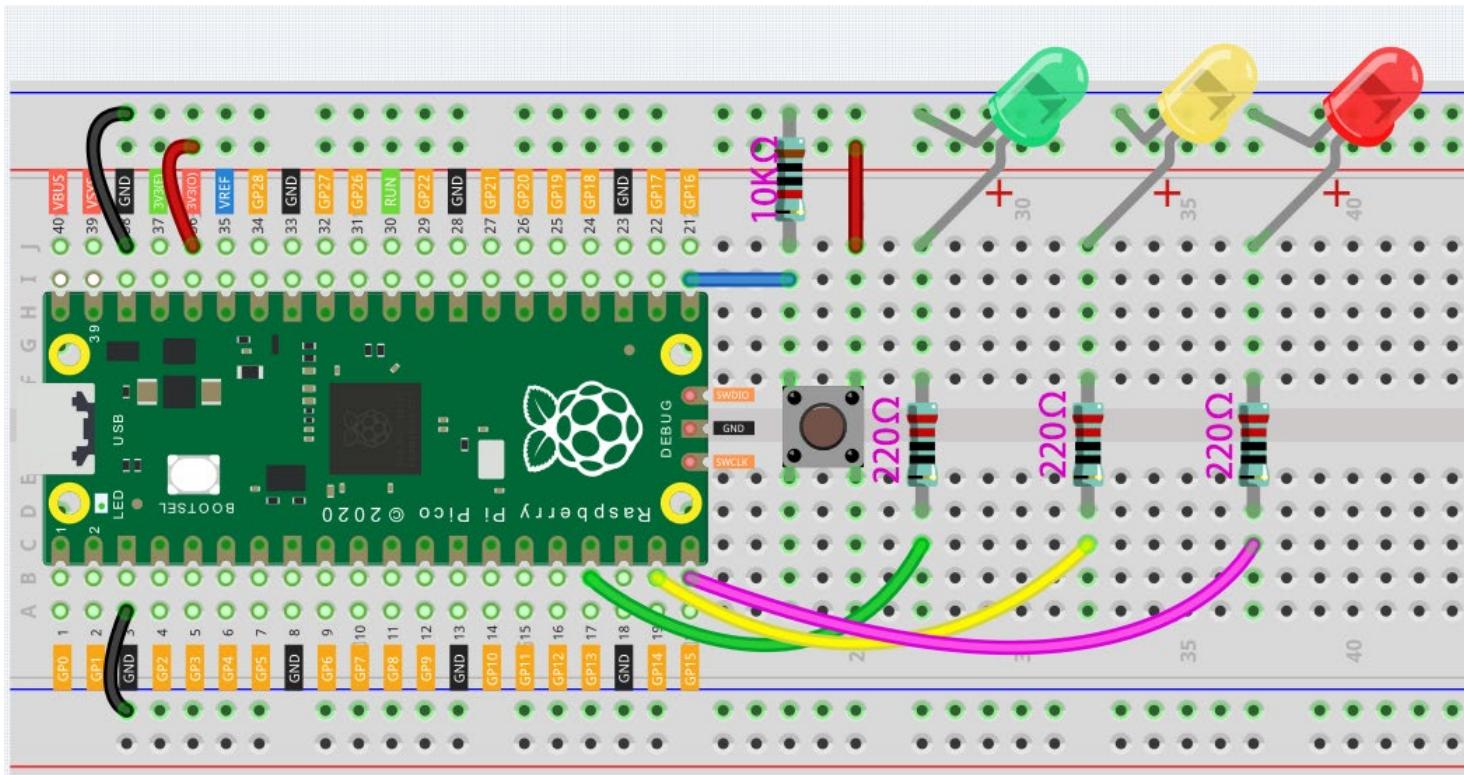
Puffin Crossing (Mini pedestrian crossing system with proper control)

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](#).

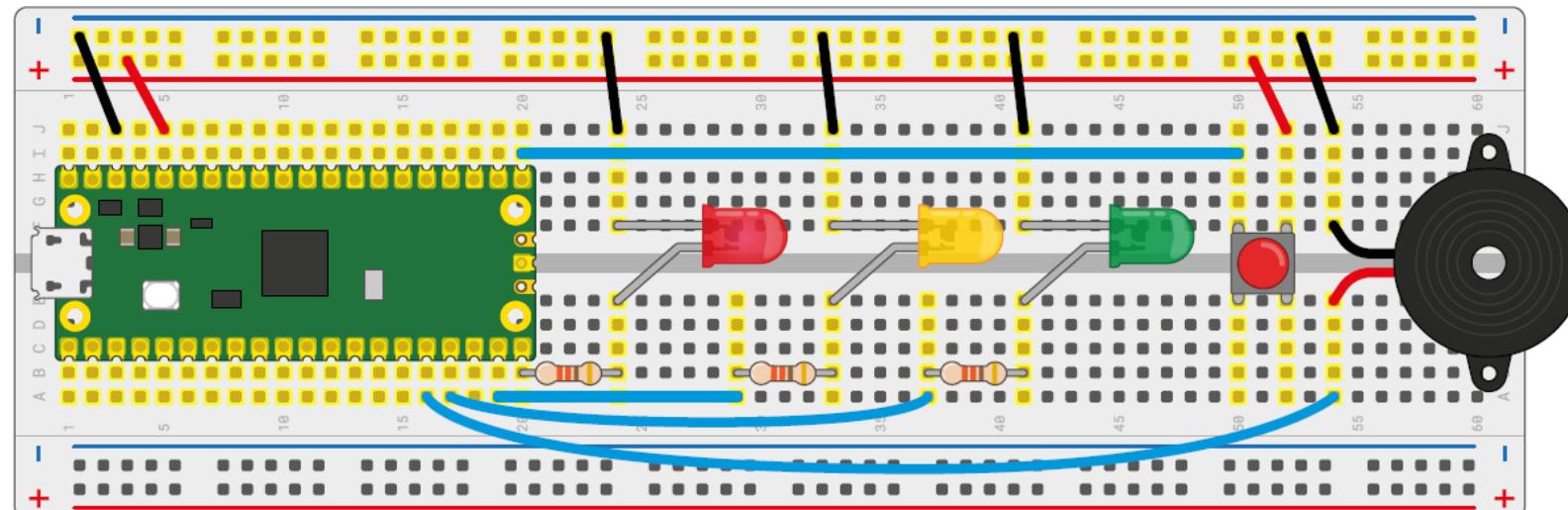
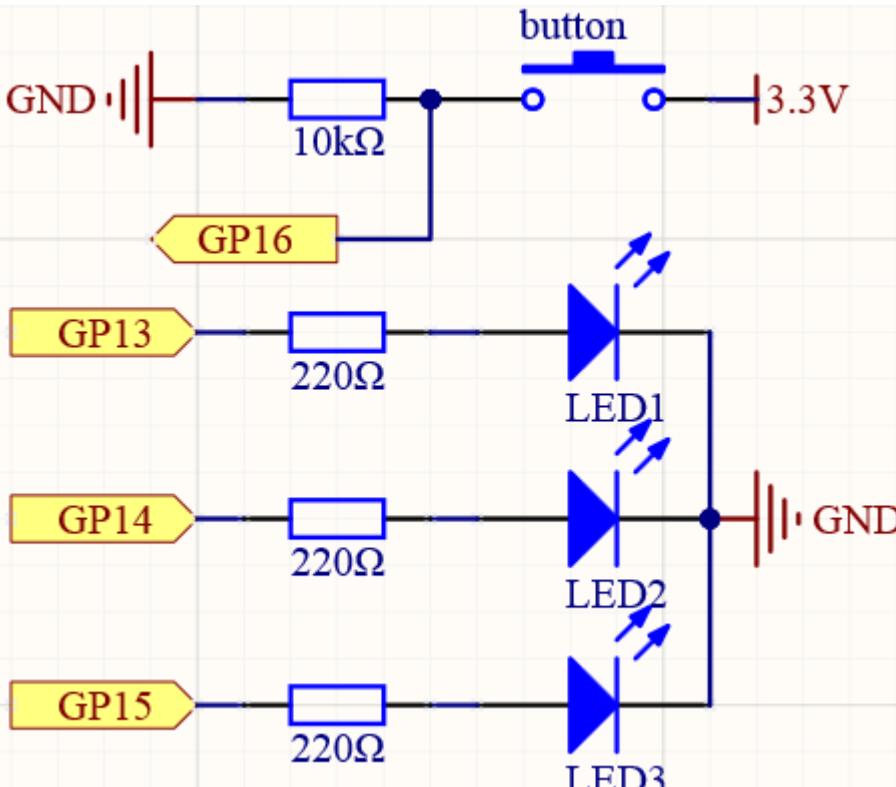


A simple traffic light Control system with proper control

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)
```

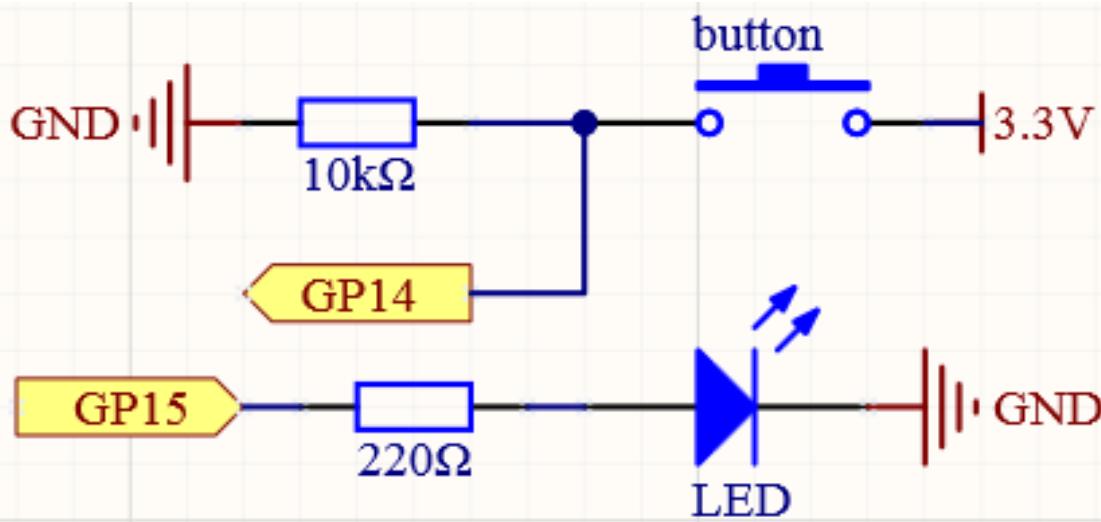
The diagram illustrates a mini pedestrian crossing system using a Raspberry Pi Pico. The circuit consists of a speaker connected to pin 39, three LEDs (red, yellow, green) with resistors, and a pushbutton connected to pin 16 via a pull-up resistor. The code uses threads to handle button presses and control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

Simple Reaction Timing Game (Single Player)

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

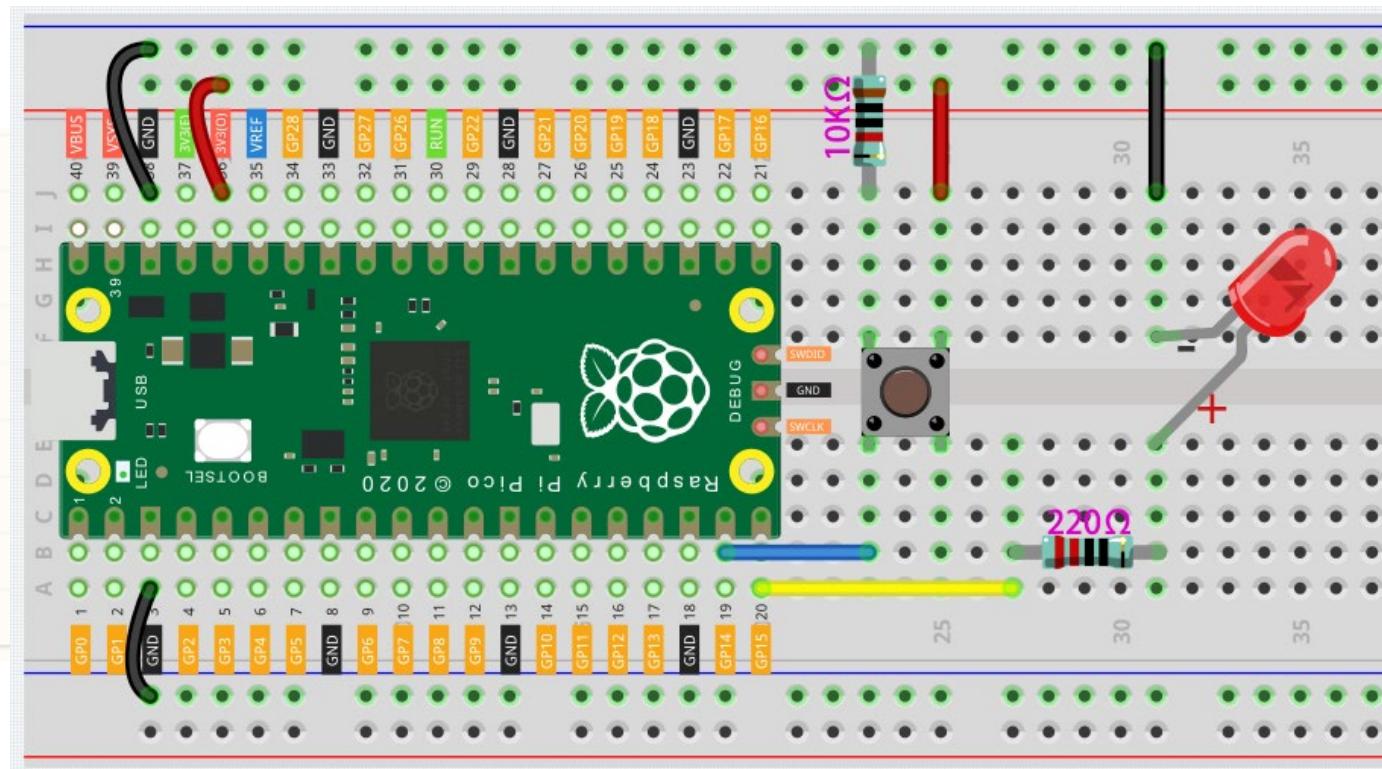
Your reaction time – the time it takes your brain to process the need to do something and send the signals to make that something happen – is **measured in milliseconds**: the average human reaction time is around **200–250 milliseconds**. People with quick reaction time have a huge advantage in the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

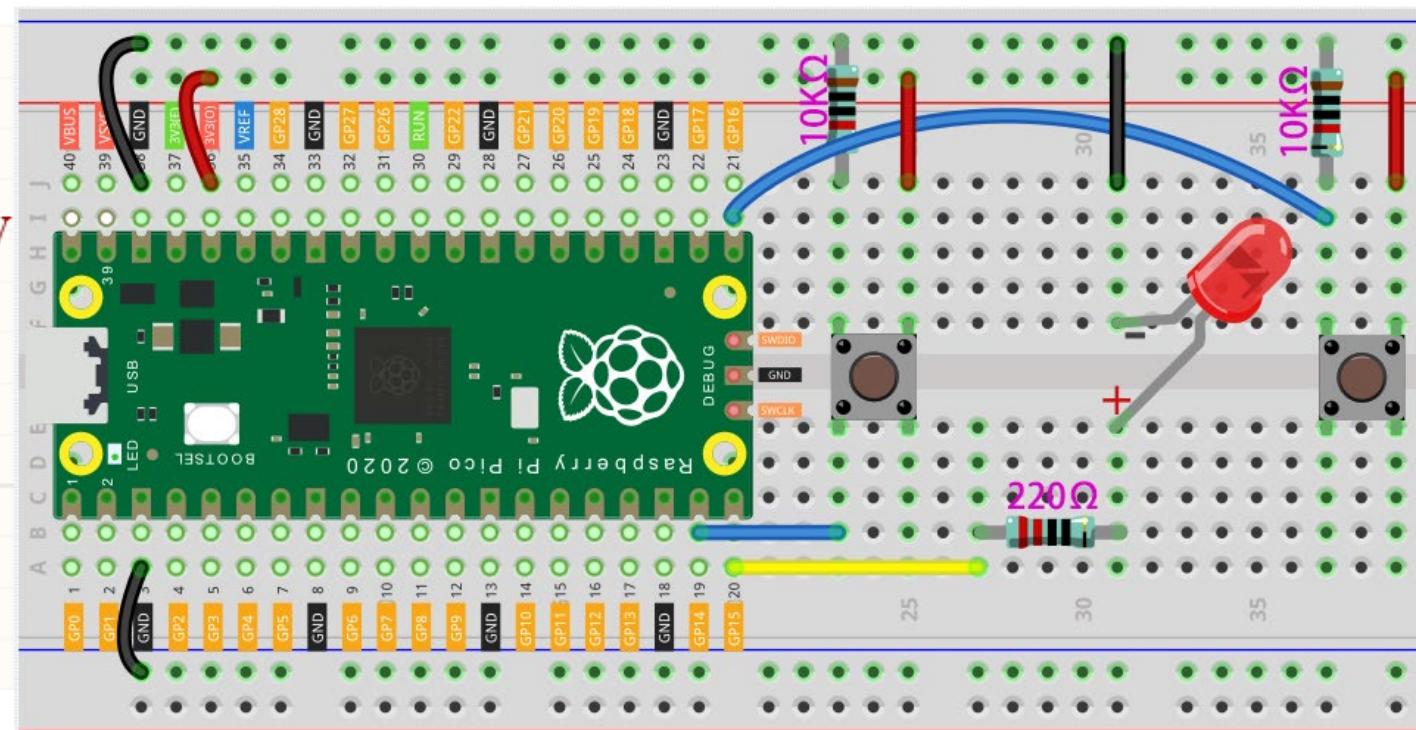
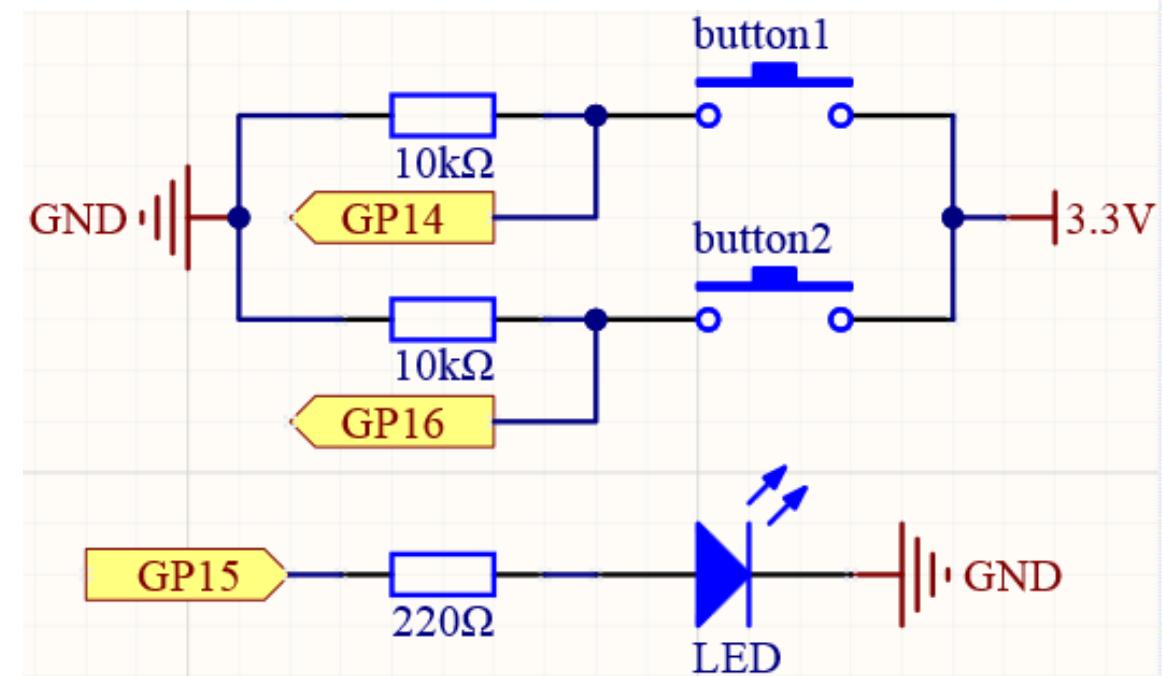
```
MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
MPY: soft reboot
MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> Your reaction time was 473 milliseconds!
```

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.



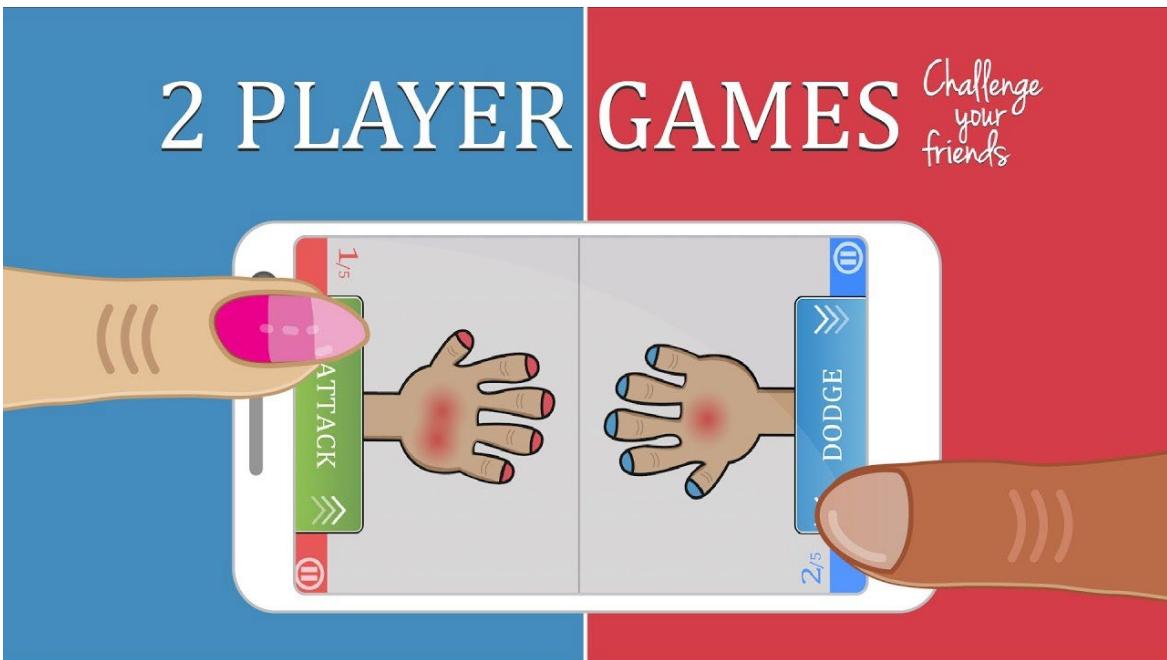
- Press the Run button and wait for the LED to go out – but don't press either of the push-button switches just yet.
- You'll see that the Shell area remains blank, and doesn't bring back the >>> prompt; that's because the main thread is still running, sitting in the loop you created.

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

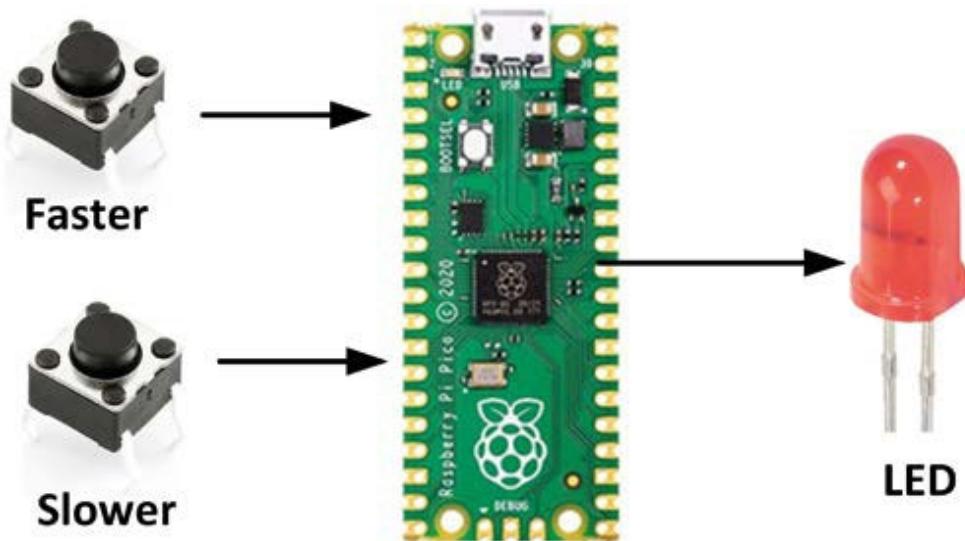
00:24.861 101%

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

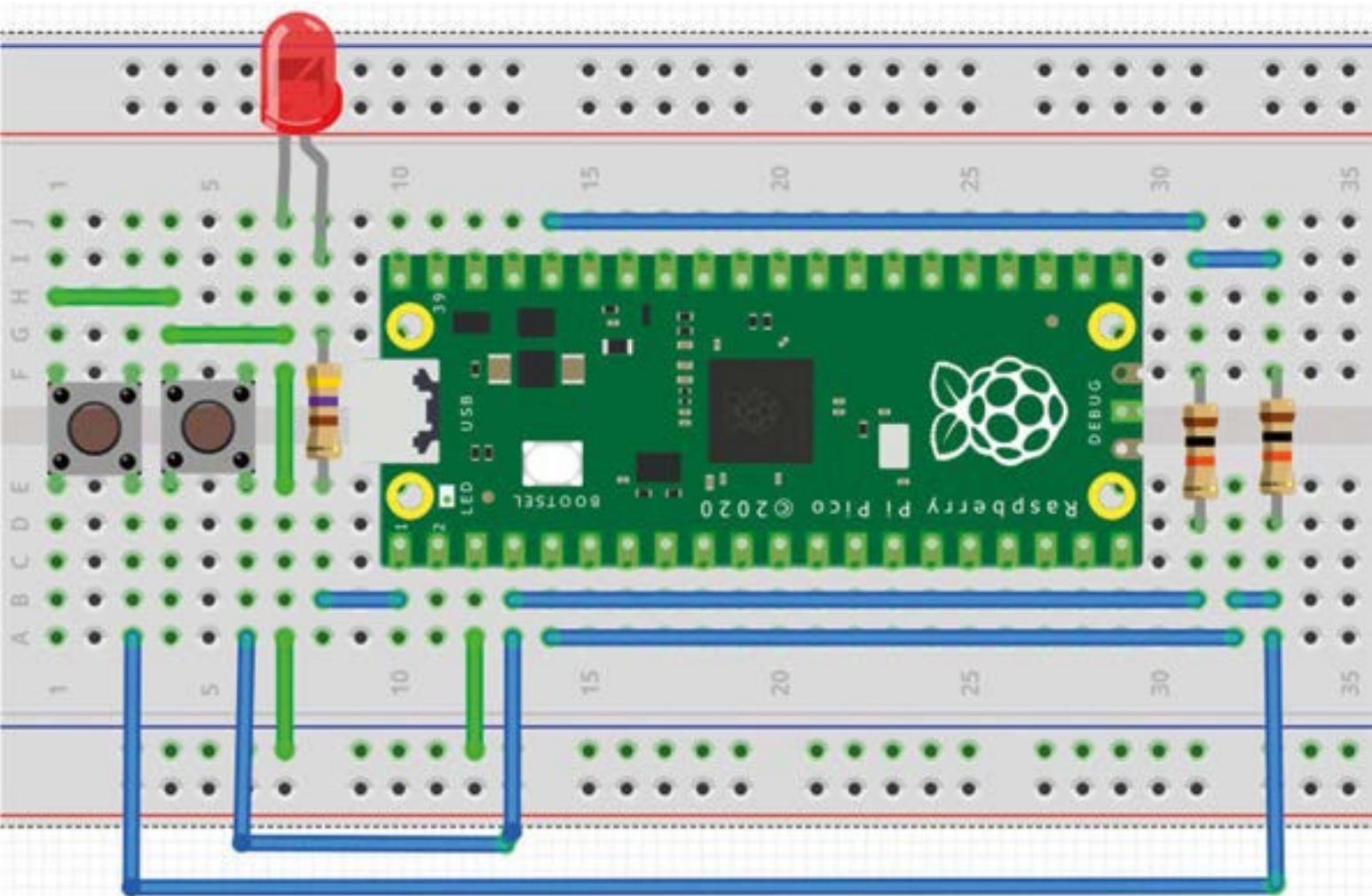
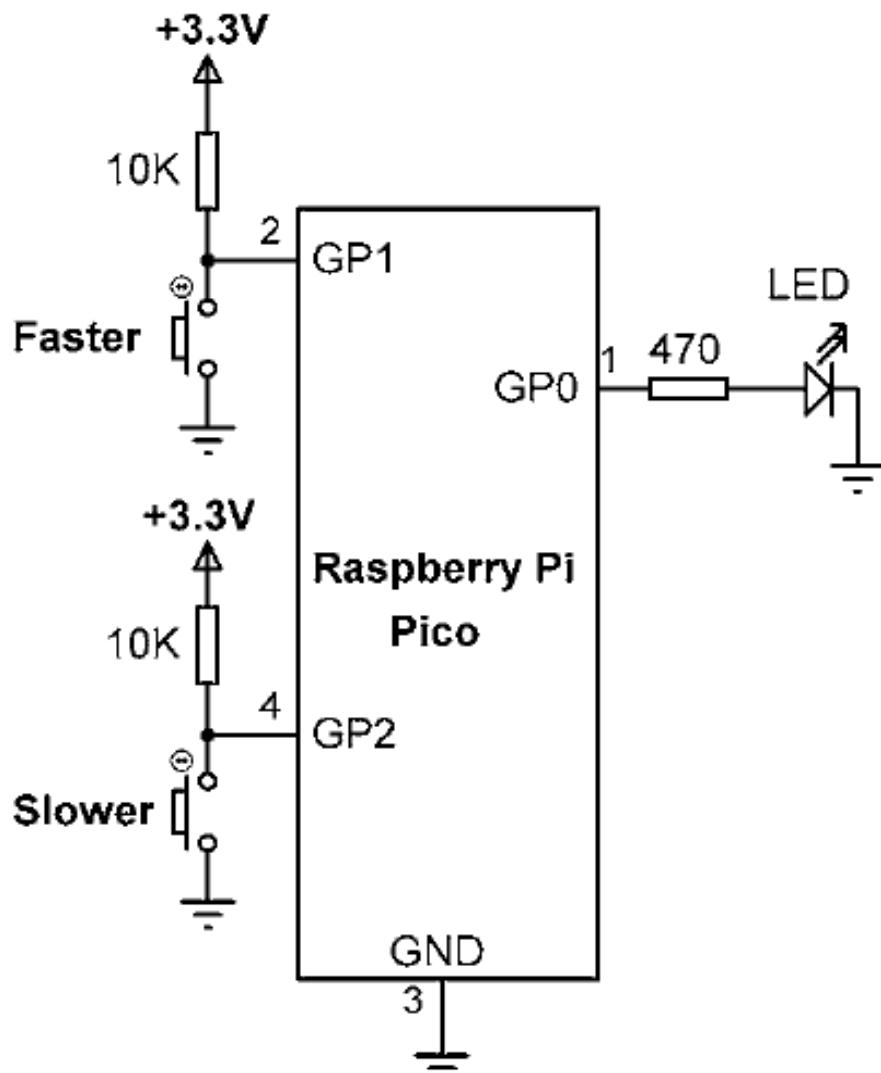
Objective:

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

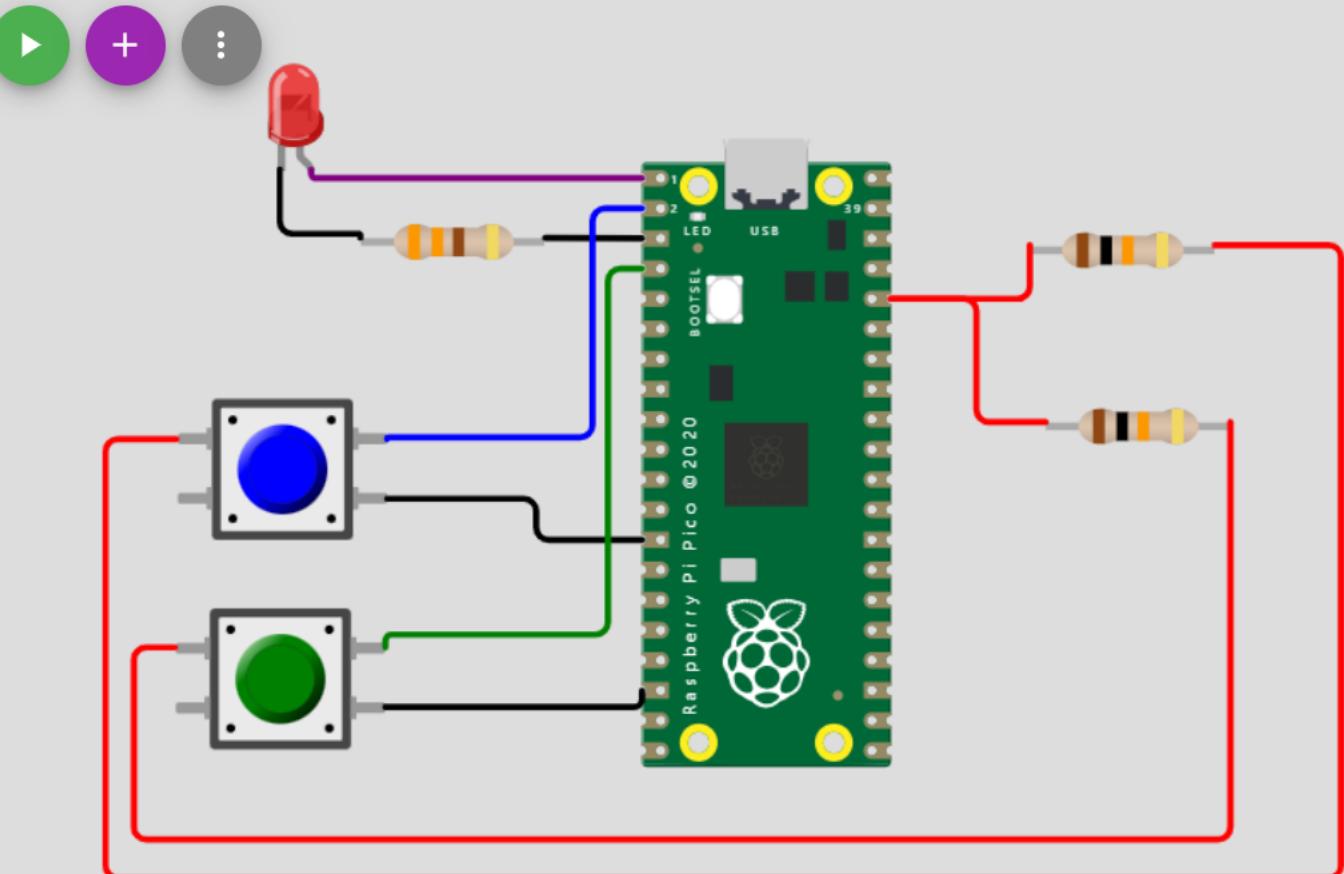
Docs

B

main.py • diagram.json •

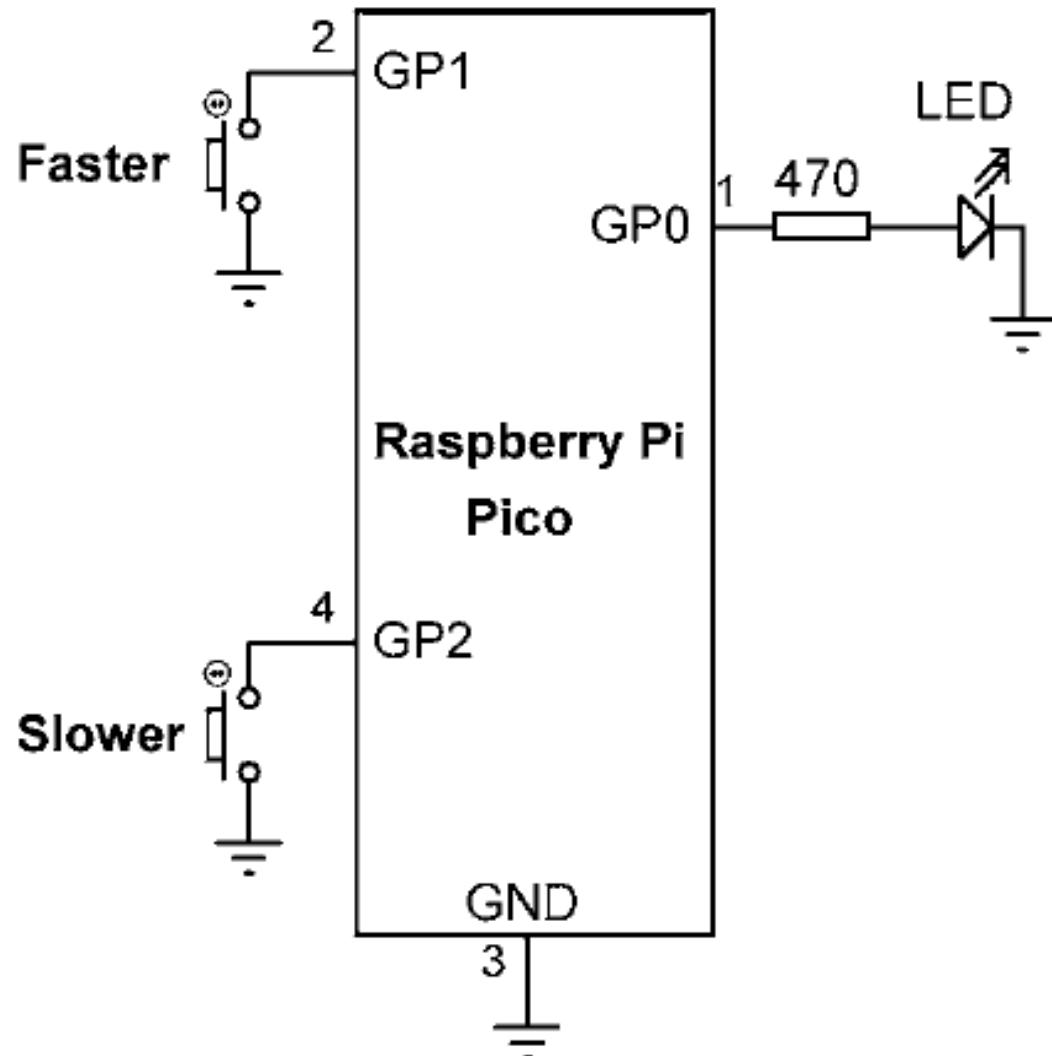
```
1  from machine import Pin
2  from utime import sleep
3
4  LED = Pin(0, Pin.OUT)          # LED at pin GP0
5  Faster = Pin(1, Pin.IN)        # Faster at pin GP1
6  Slower = Pin(2, Pin.IN)        # Slower at pin GP2
7  dly = 1.0                      # Default delay
8
9  def Flash_Faster(Faster):
10     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

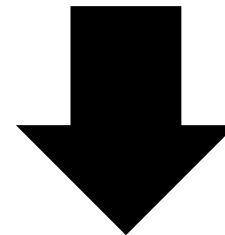


Changing the LED flashing rate – using pushbutton interrupts

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
```

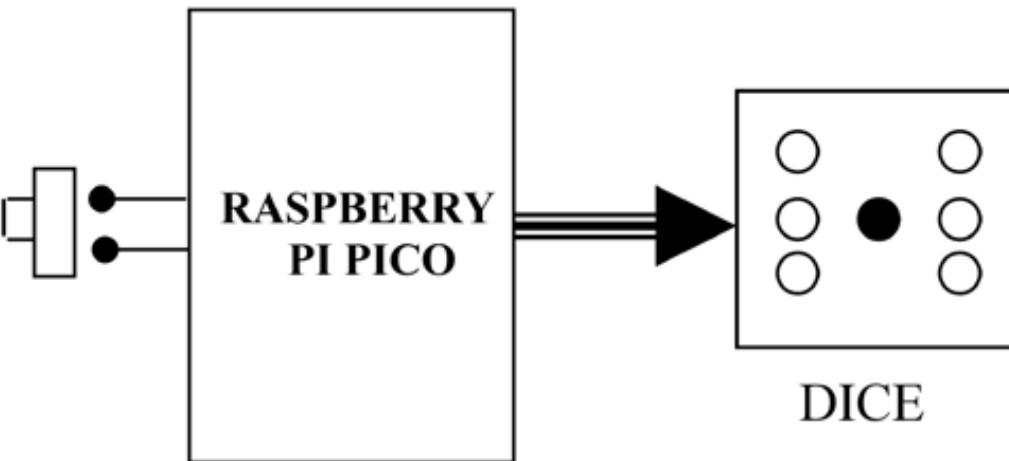
Simulation

Raspberry Pi Pico @ 2020

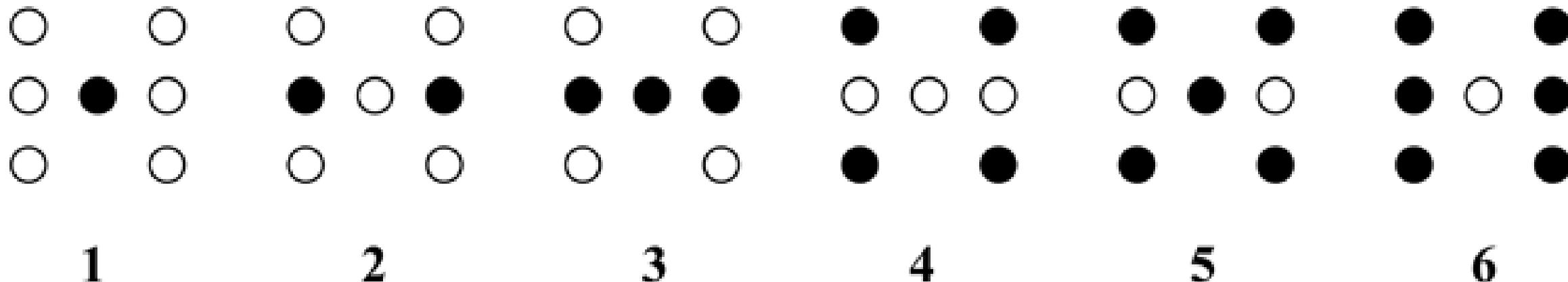
Electronic Dice

Objective:

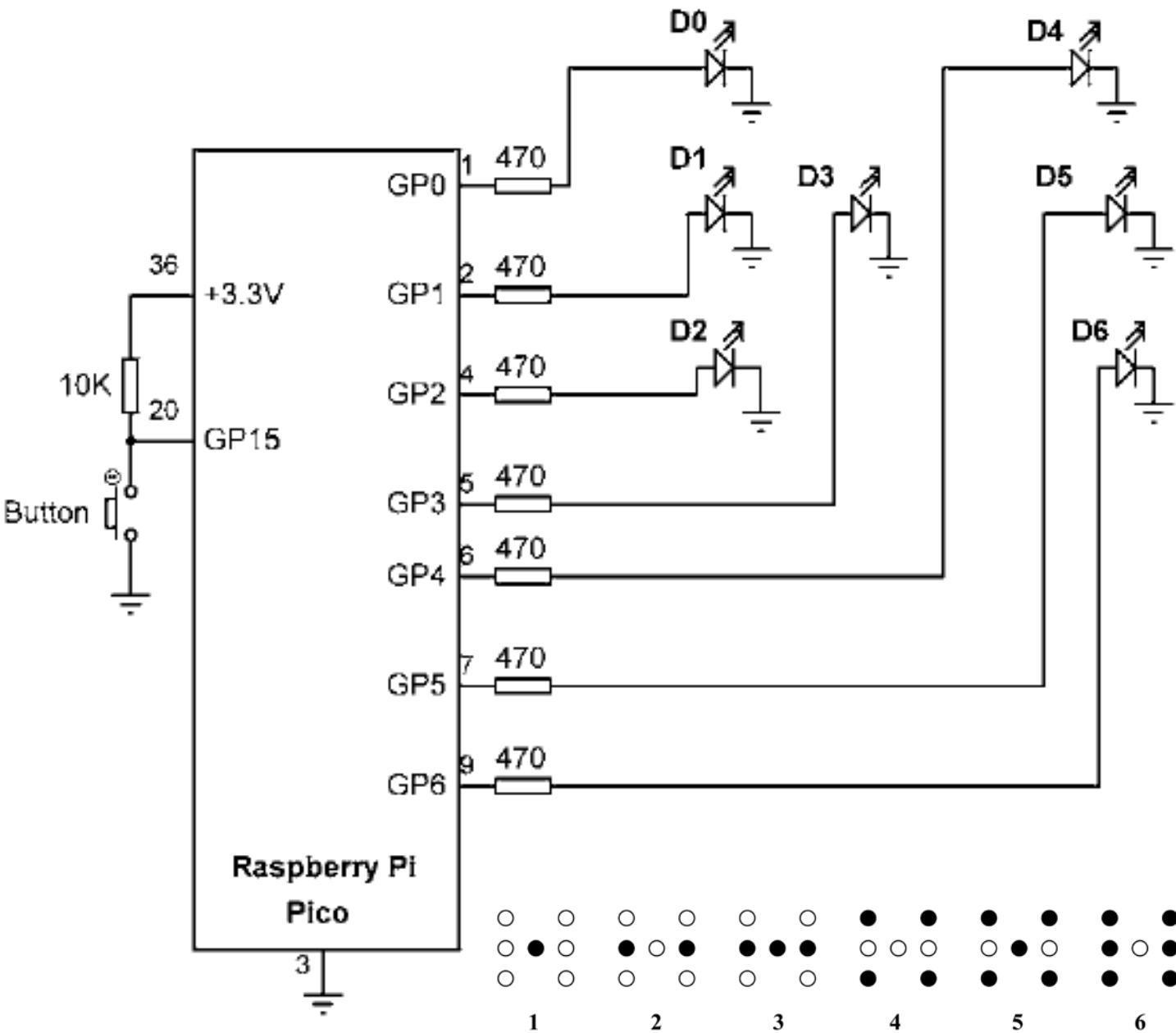
Push-button switch



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

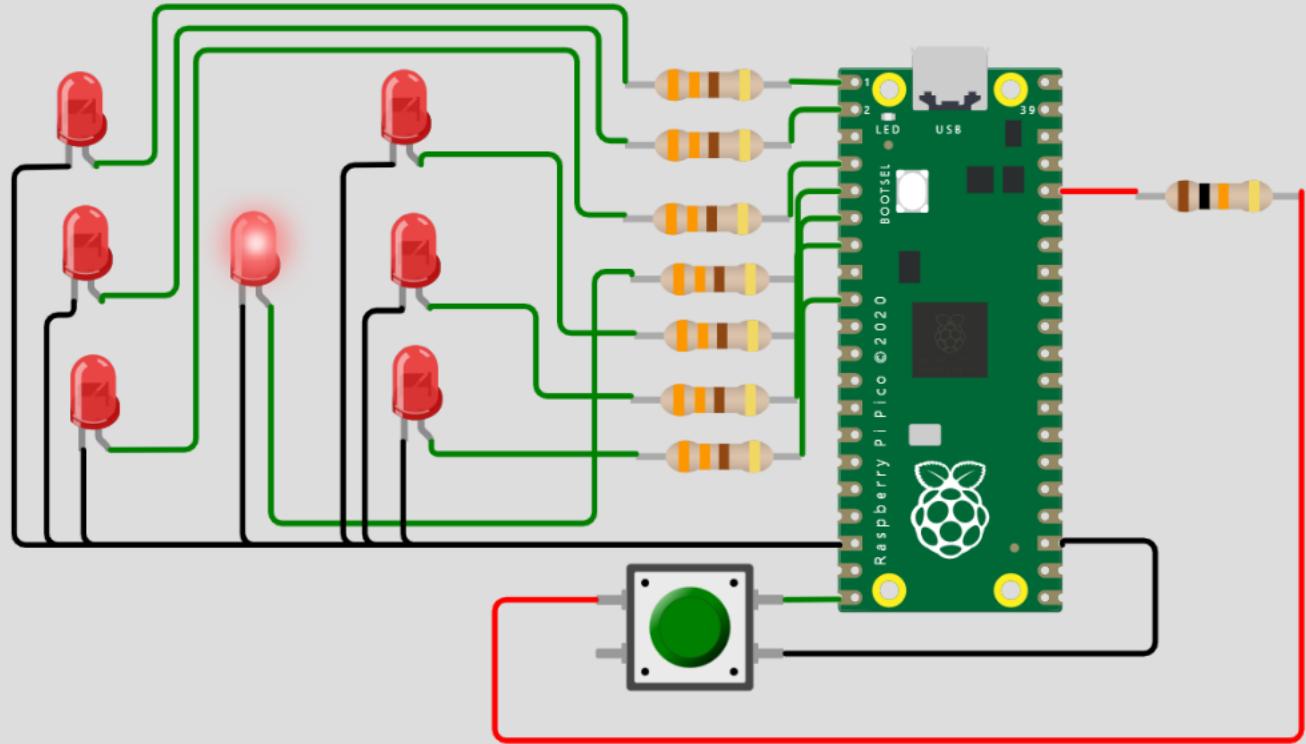
WOKWi SAVE SHARE Docs B

main.py diagram.json PIO 

```
27
28 def DICE():
29     n = random.randint(1, 6)          # generate a random number
30     pattern = DICE_NO[n]              # find the pattern
31     Port_Output(pattern)            # turn ON required LEDs
32     sleep(3)                       # wait for 3 seconds
33     Port_Output(0)                 # turn OFF all LEDs
34
35
36 Configure_Port()
37
38 while True:
39     if Button.value() == 0:          # Button pressed?
40         DICE()                      # Call DICE
41     pass                           # Do nothing
```

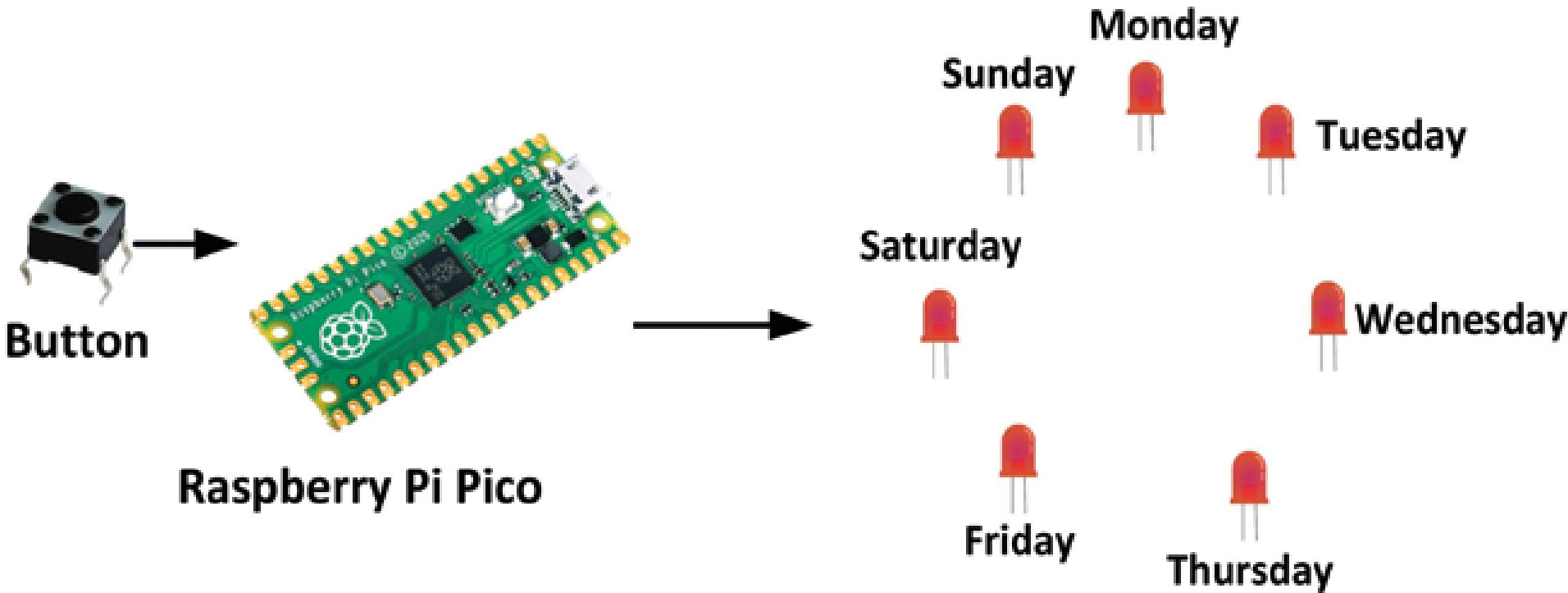
Simulation 

00:18.099 100%



Lucky Day of the Week

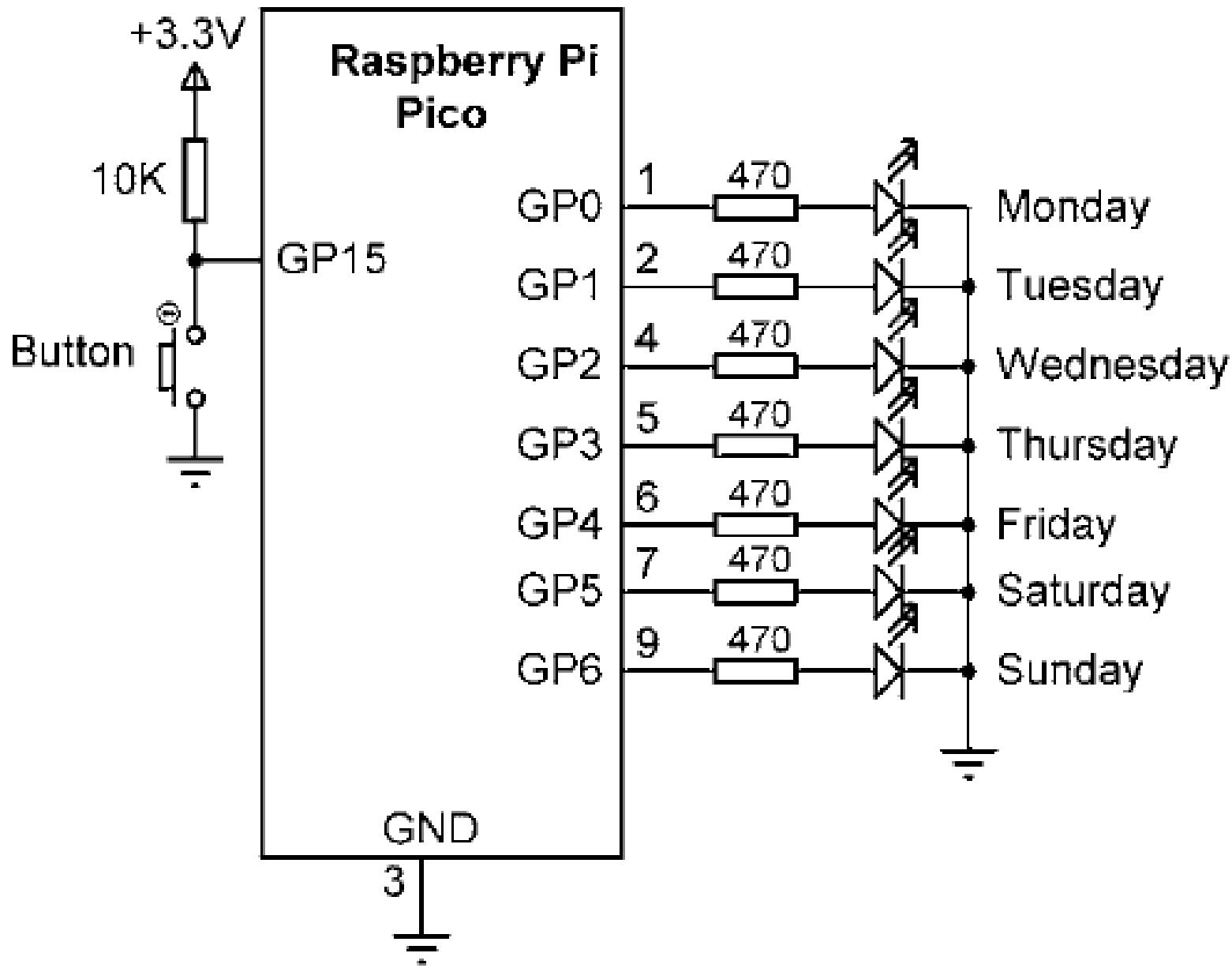
Objective:



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.