# Internet Of Things

# Button Basics

*Prepared by:*

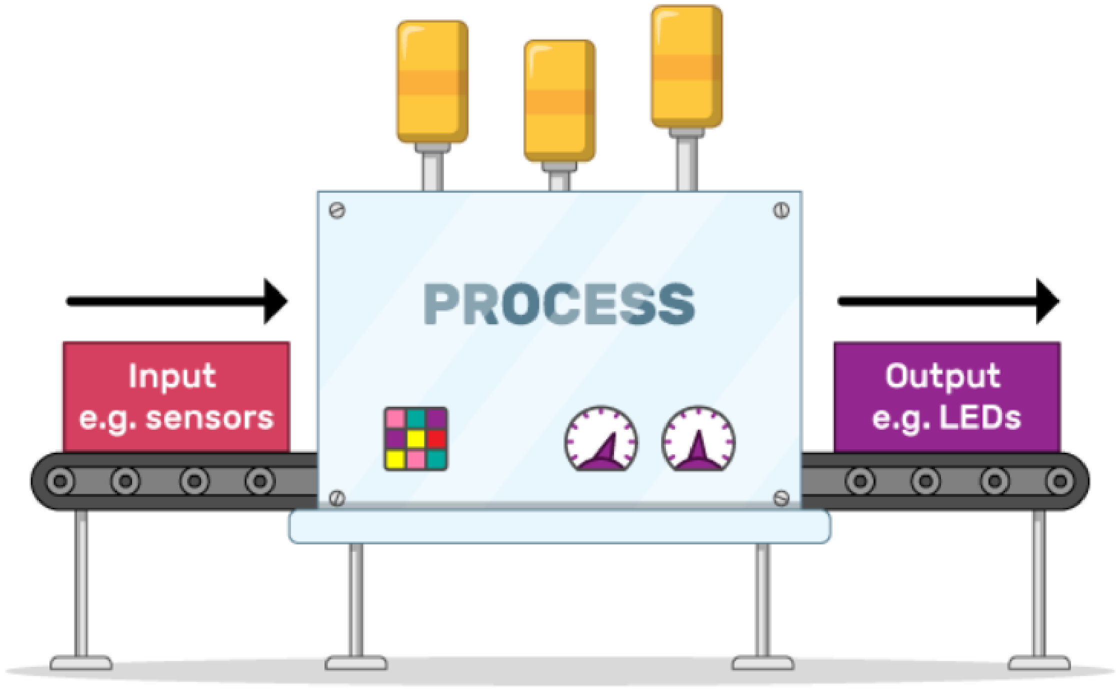
**Dr. Murad Yaghi**
**Eng. Malek Al-Louzi**

School of Computing and Informatics – Al Hussein Technical University

**Fall 2024/2025**

## Introduction

- Button switches are an input.
- The button sends a signal which is received by the Raspberry Pi to be processed.

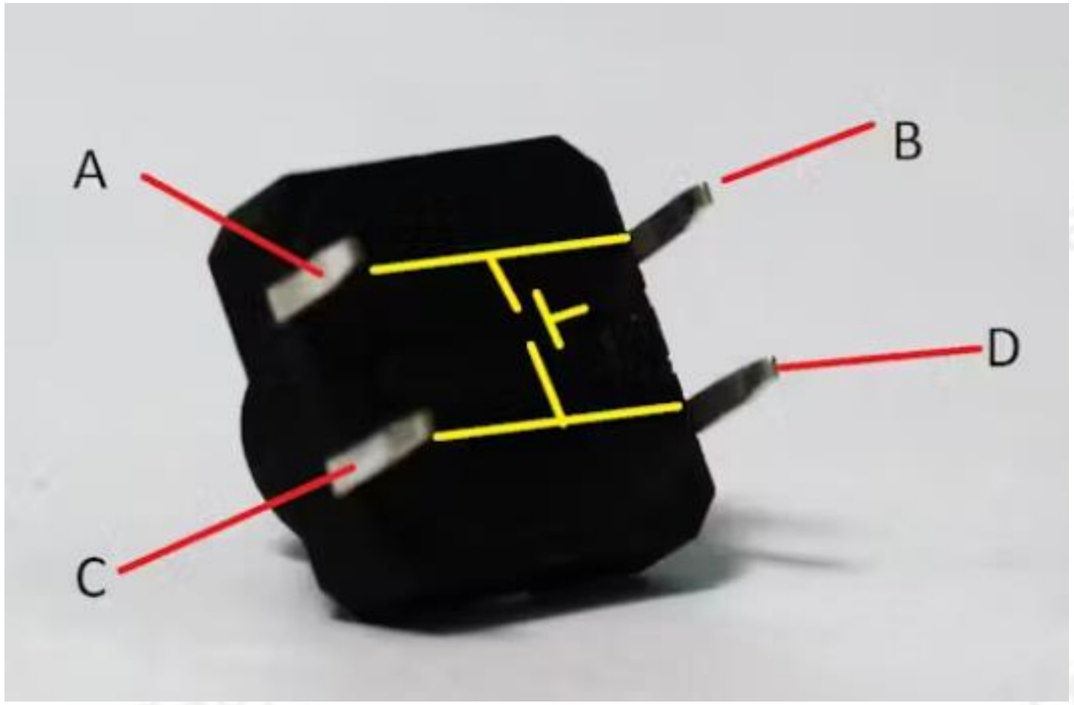


## Introduction

- Push Button is simplest of devices, and it is the basic input device that can be connected to any controller or processor like Arduino or Raspberry Pi.

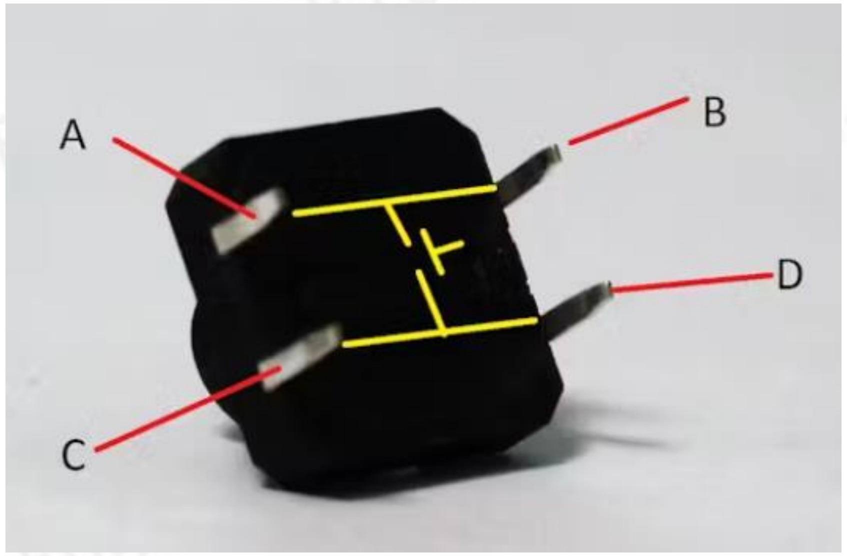
## Introduction

- A push button in its simplest form consists of four terminals.
- There is direct connection between A and B, whether button is pressed of not. And same for C and D.



## Introduction

- When button is pressed, A and C are getting shorted (So, pins are getting shorted) and same way B and D are getting shorted.
- So to use this push button as a switch, use either A-C pair or B-D pair or A-D or B-C.



## Introduction

- It is clear that 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

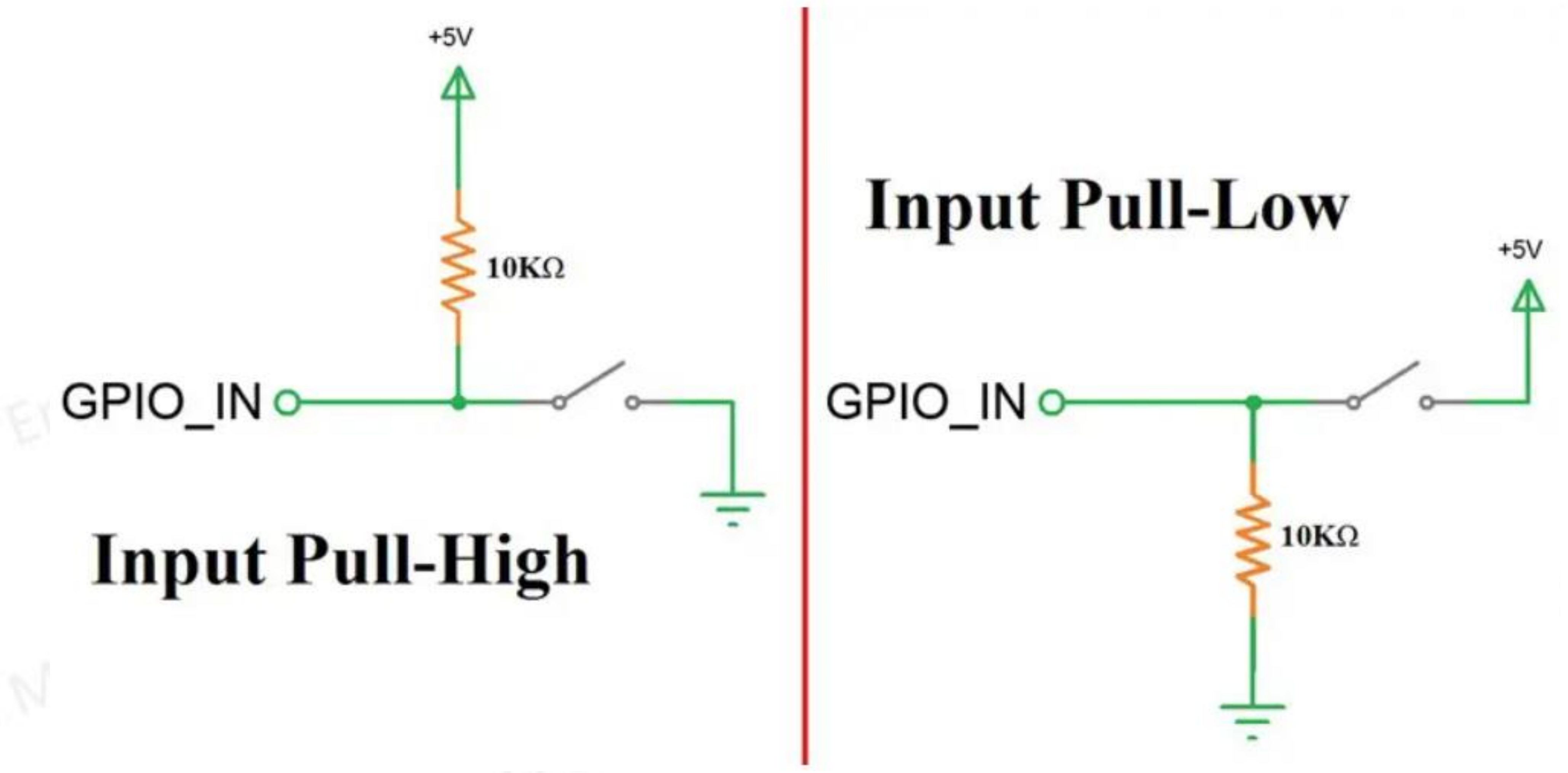
## Introduction

- 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

## Introduction

- The following image in the next slide, shows an input pulled High and Low with the help of pull-up and pull-down resistors.
- In case of pull-up, the input will always read HIGH and when the button is pushed, it will read LOW.
- In contrast, when an input pin is pulled-down, it will always read LOW and when the button is pressed, it will read HIGH.

## Pull-up and Pull-down resistors

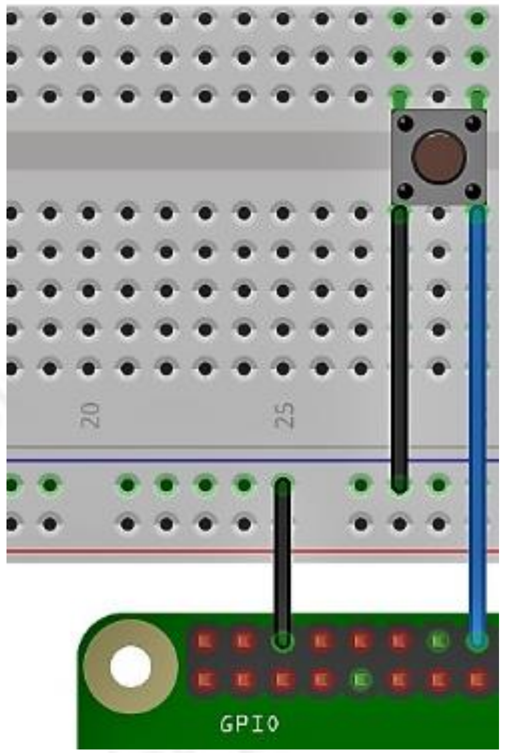


## Interfacing a Push Button with Raspberry Pi

- Modern boards like Arduino and Raspberry Pi have a feature of Internal Pull-up or Internal Pull-down
- Each GPIO pin has software configurable pull-up and pull-down resistors
- With the help of this feature, you need not physically connect a pull-up or pull-down resistor to the input pin but configure it using the software

## Setting up the circuit

- The Button is wired in such a way that when it is pressed, it will connect GPIO 23 to the Ground(GND).



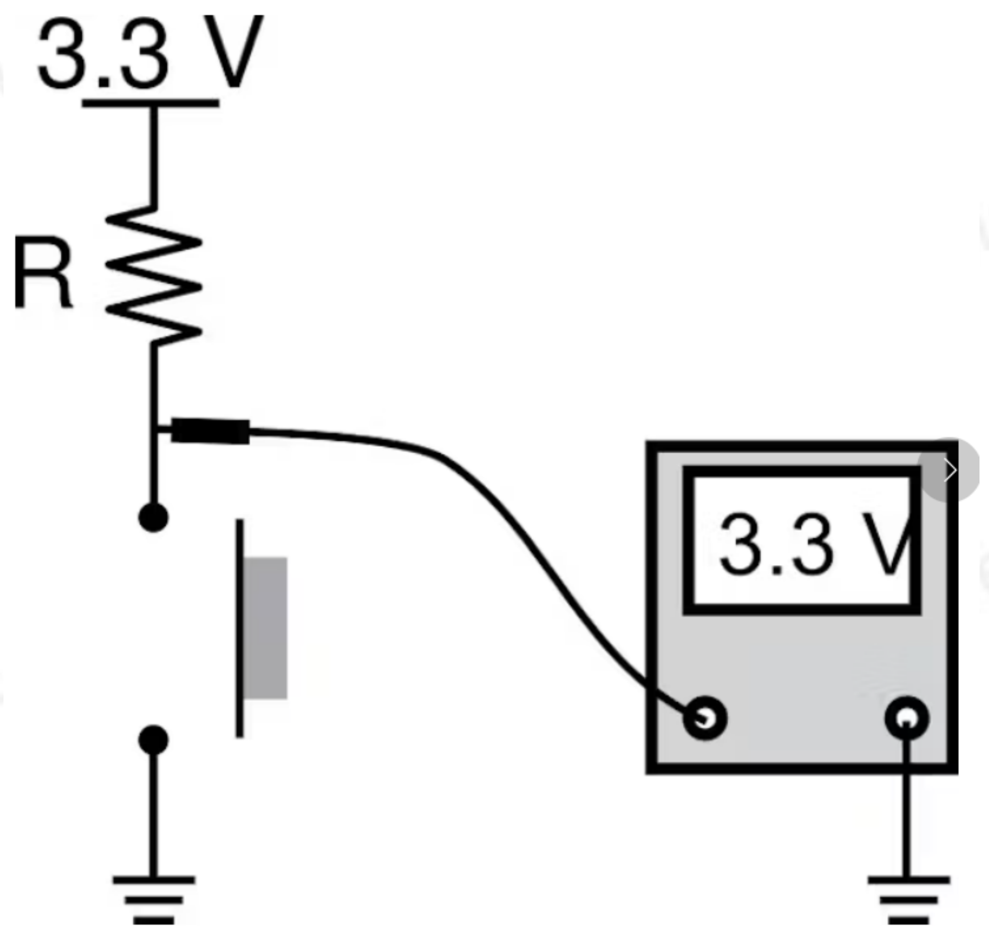
## Code

- In the code, we initialize pin 23 as an input pin, and instruct the Raspberry Pi to pull the pin high using the pull_up_down parameters

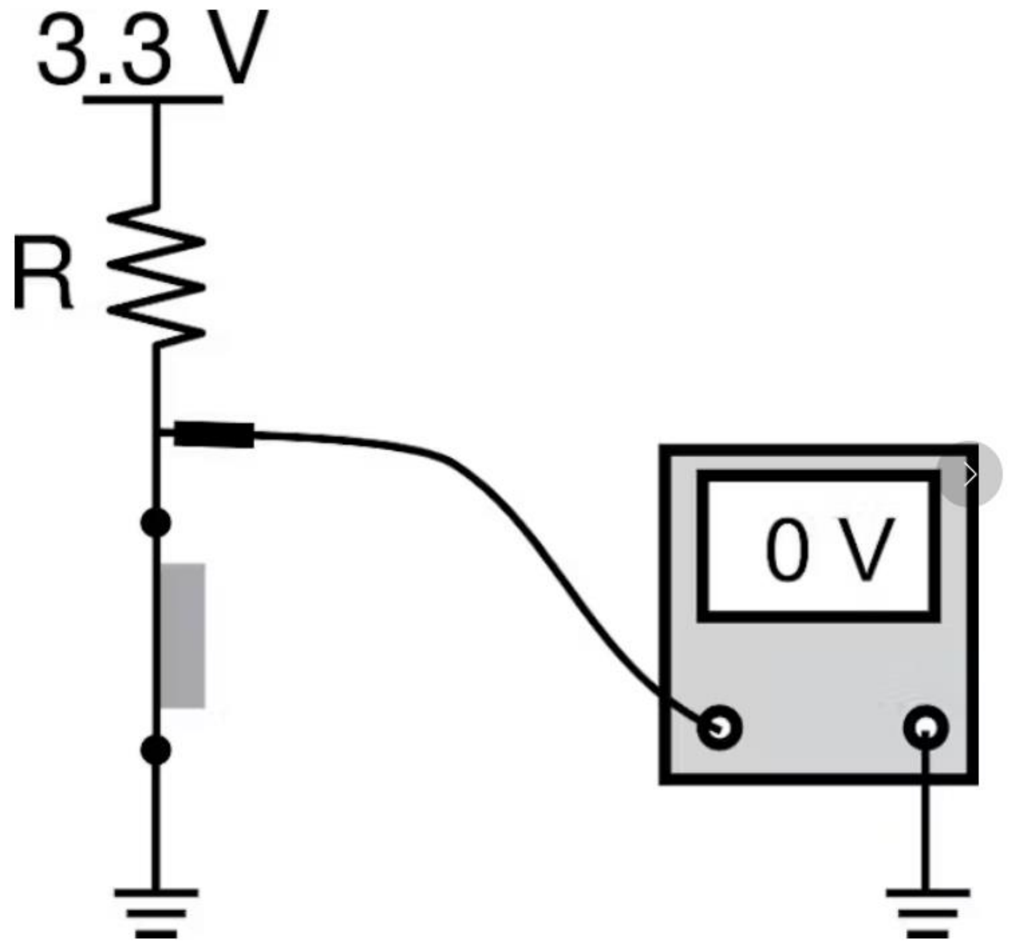
```
GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

- This means GPIO 23 is normally pulled up to 3.3V. So that when you read the input value using **GPIO.input()** it will return **GPIO.LOW** if the button is pressed

## Pull Up Resistor Concept when button not pressed



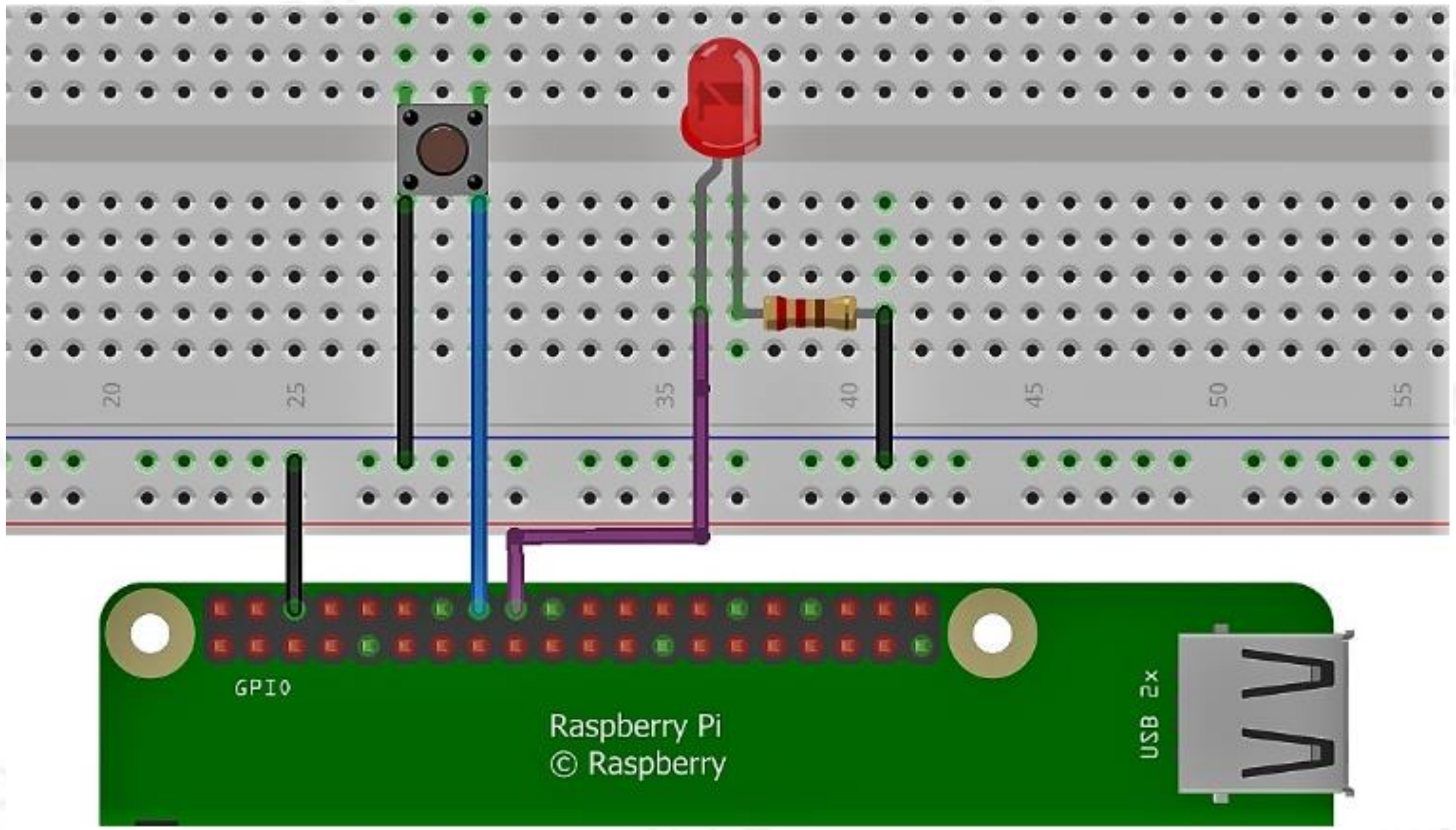
## Pull Up Resistor Concept when button pressed



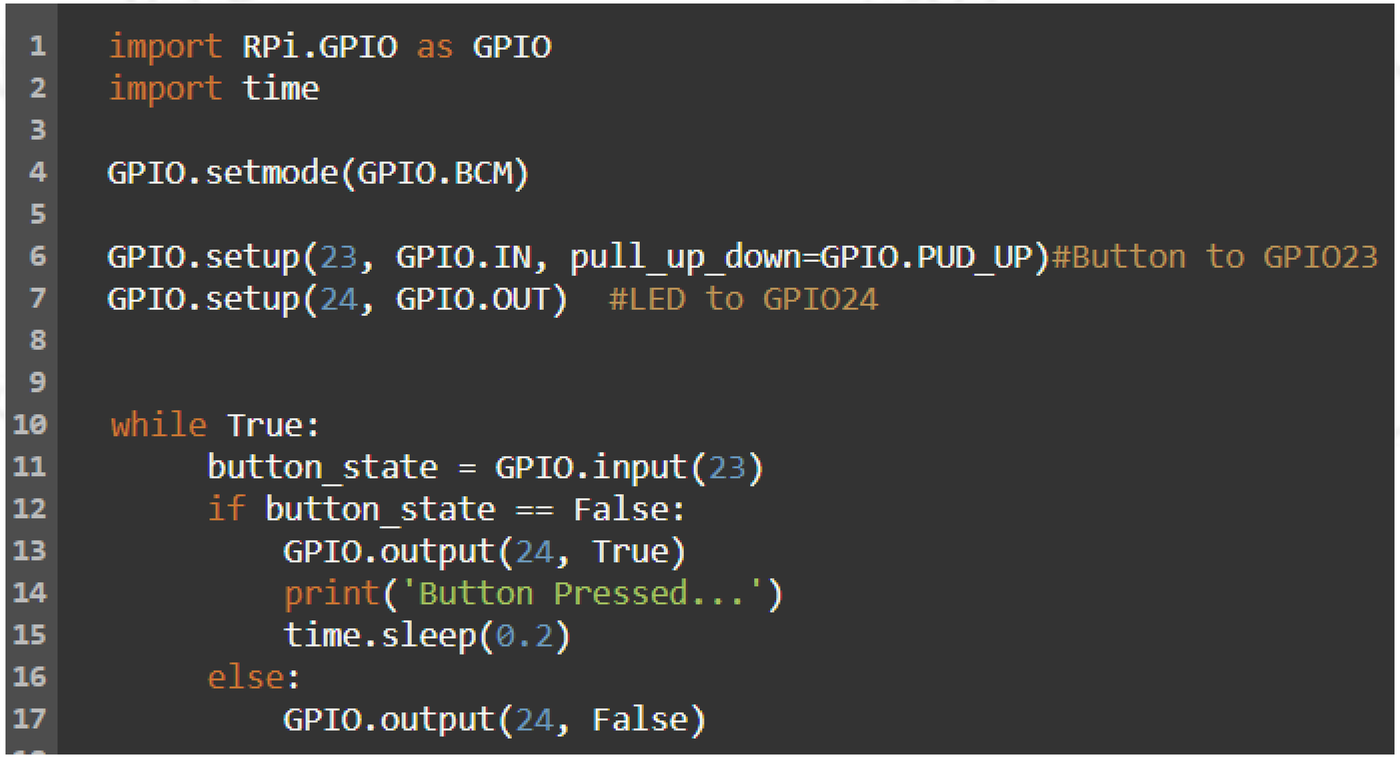
## Code

- As mentioned before, each GPIO pin in Raspberry Pi has software configurable pull-up and pull-down resistors
- When using a GPIO pin as an input, you can configure these resistors so that one of the resistors is enabled, using the optional **pull_up_down** parameter in **GPIO.setup()** function
  - If it is set to **GPIO.PUD_UP**, the pull-up resistor is enabled
  - if it is set to **GPIO.PUD_DOWN**, the pull-down resistor is enabled

## Complete circuit



## Complete code



```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)#Button to GPIO23
GPIO.setup(24, GPIO.OUT)  #LED to GPIO24


while True:
     button_state = GPIO.input(23)
     if button_state == False:
         GPIO.output(24, True)
         print('Button Pressed...')
         time.sleep(0.2)
     else:
         GPIO.output(24, False)
```

## Exercise

- **Rewrite the code to use the internal pull-down resistor**
- *Hint*: you need to make small modification on the circuit

## Functions in Python

- Functions allow you to name a block of code and then reuse that code by calling its name
- You can pass data to a function through variables known as **parameters**
- Data can also be passed back to the calling statement using the **return** statement

## Functions in Python

- An example of a function definition would look like:

```
def functionName(parameter1, parameter2):
    # Code goes here
```

- You can call this function elsewhere in your code with

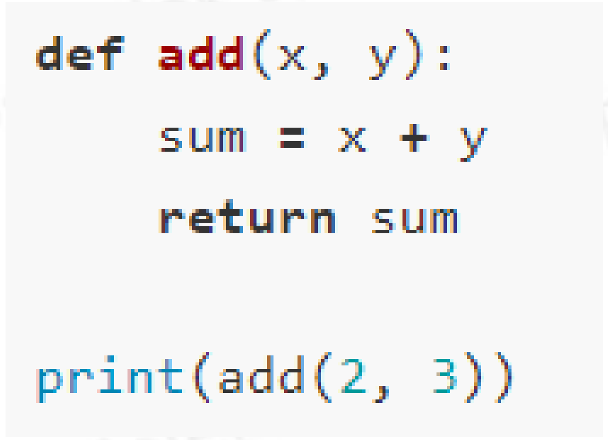
```
functionName(argument1, argument2)
```

## Functions in Python

- Note that variables declared inside the function definition are known as having local scope
- This means that they cannot be accessed outside of that function
- Variables declared at the top level of the program (i.e. outside any functions, loops, or classes) are known as having global scope and can be accessed anywhere in the program (including inside functions).

## Functions in Python

- Important: Any functions you create must be defined before you use them! If you try to call a function higher up in the code (before its definition) you'll likely get an error
- Example:



```
def add(x, y):
    sum = x + y
    return sum

print(add(2, 3))
```

## Event based GPIO input in Python

- We want to rewrite our program to output a single message whenever the button is pressed rather than continuously outputting a message
- To do this we need to use GPIO events
- A GPIO event in the Raspberry Pi Python GPIO library works by calling a Python function whenever an event is triggered
- Such a function is called a callback function

## Event based GPIO input in Python

- An event can be an input pin being low or high, but it could also be when the pin changes from:
  - *low to high – called rising edge*
  - *high to low – called falling edge*
- In our case we want to detect when the button is being pressed, that is going from low to high

## Event based GPIO input in Python

- Before we setup the event we must first write the callback function to be executed when the event is detected
- The callback function is a regular Python function and as such can contain any Python code, it could send out a tweet or as we do in our case simply print “Button was pushed!”

## Event based GPIO input in Python

- At the top of our program we import the GPIO library and define the function as follows:

```python
import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library

def button_callback(channel):
    print("Button was pushed!")
```

- Next the program will initialize the input pin as follows:

```python
GPIO.setmode(GPIO.BOARD) # Use physical pin numbering
GPIO.setup(10, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) # Set pin 10 to be an input pin and set
initial value to be pulled low (off)
```

## Event based GPIO input in Python

- With the pin defined as an input pin we can attach the event to the pin

```
GPIO.add_event_detect(10,GPIO.RISING,callback=button_callback) # Setup event on pin 10 rising edge
```

- Notice how we provide the function name as the callback parameter such that the library knows which function to call when the event is triggered

## Event based GPIO input in Python

- Finally, we instruct python to wait for keyboard input and when someone presses enter finish the program

```
message = input("Press enter to quit\n\n") # Run until someone presses enter
```

## Complete code

```python
import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library

def button_callback(channel):
    print("Button was pushed!")

GPIO.setmode(GPIO.BOARD) # Use physical pin numbering
GPIO.setup(10, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) # Set pin 10 to be an input pin and set initial value to be pulled low (off)

GPIO.add_event_detect(10,GPIO.RISING,callback=button_callback) # Setup event on pin 10 rising edge

message = input("Press enter to quit\n\n") # Run until someone presses enter

```

## Any Questions???