https://www.halvorsen.blog

Raspberry Pi GPIO with Python

Hans-Petter Halvorsen

Free Textbook with lots of Practical Examples

Python for Software Development

Hans-Petter Halvorsen

Python Software Development ☒

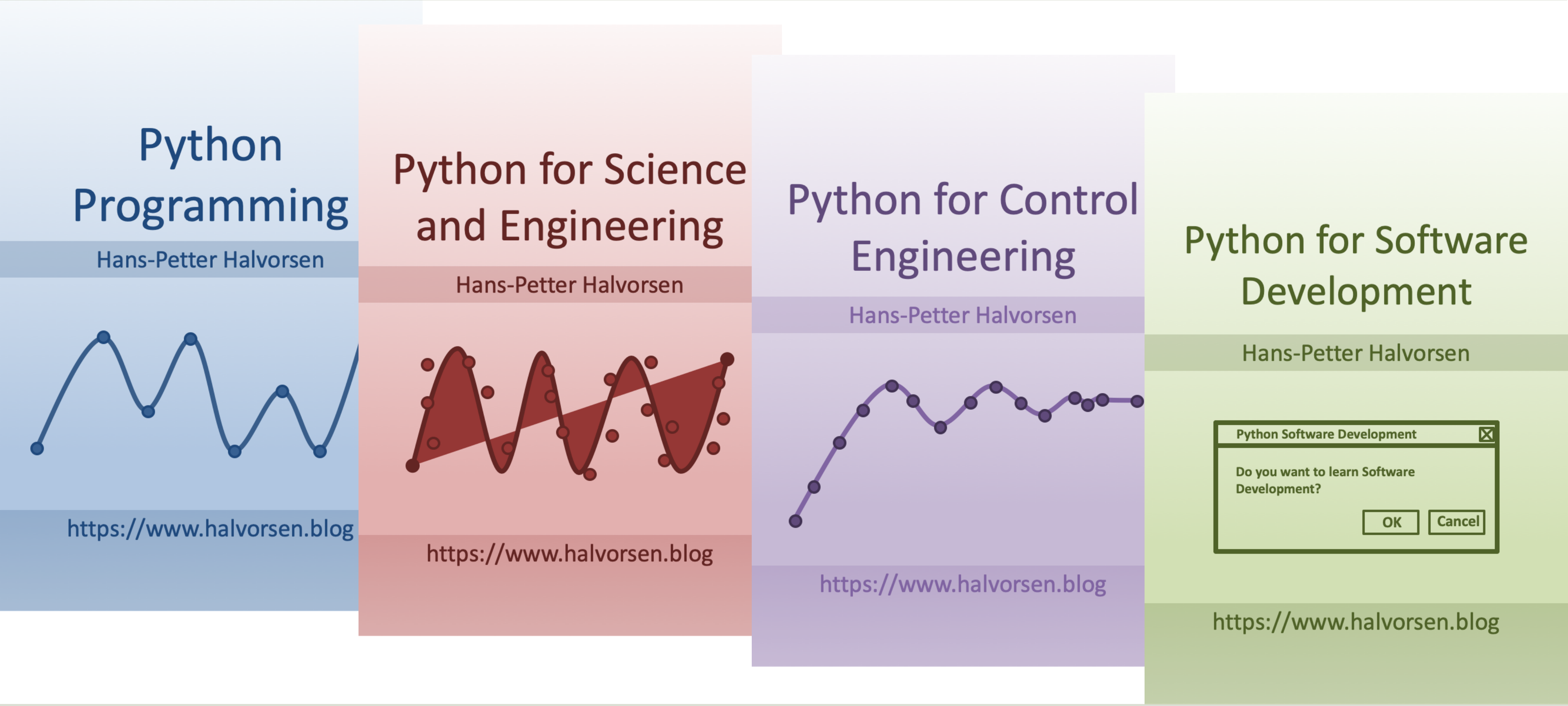
Do you want to learn Software Development?

OK Cancel

https://www.halvorsen.blog

https://www.halvorsen.blog/documents/programming/python/

Additional Python Resources



Python Programming

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Science and Engineering

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Control Engineering

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Software Development

Hans-Petter Halvorsen

Python Software Development ☒

Do you want to learn Software Development?

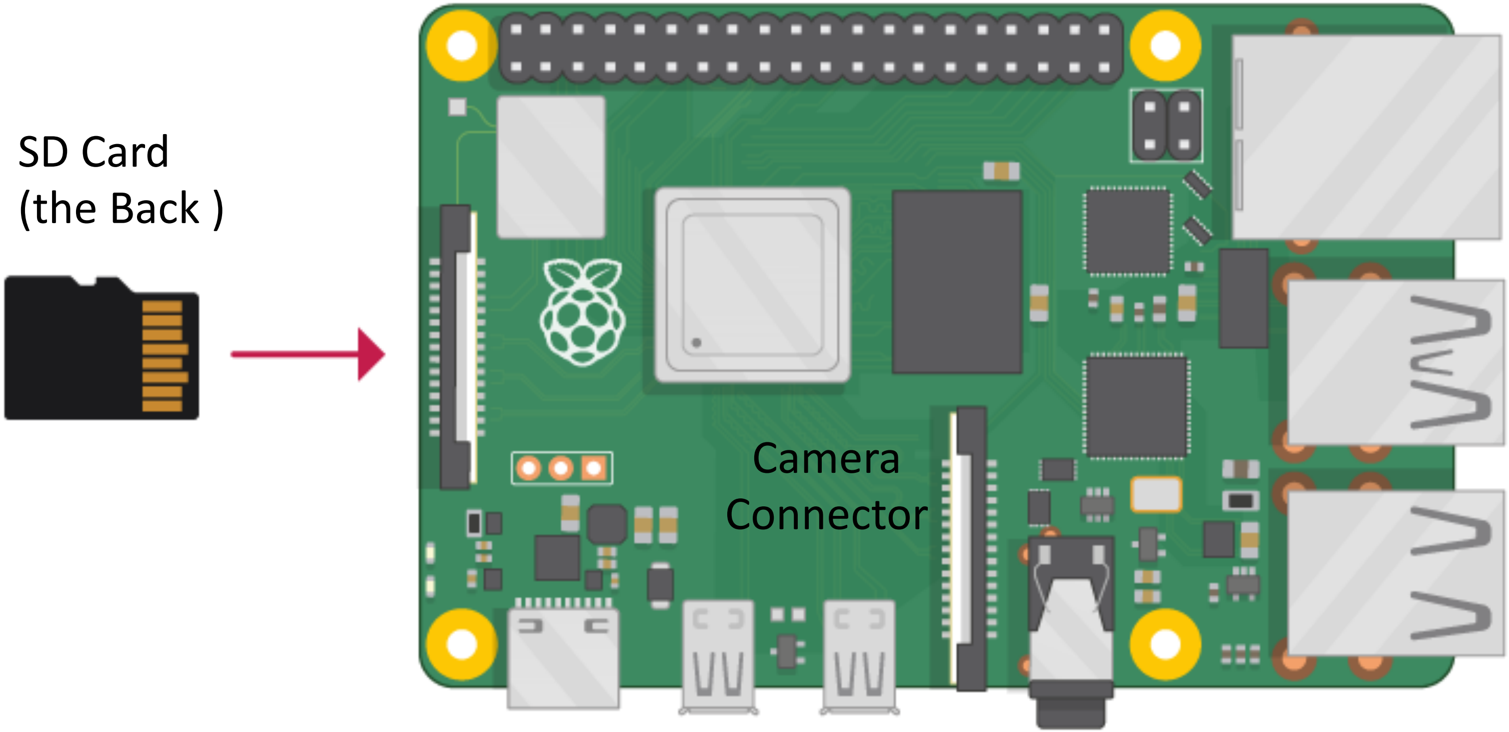
https://www.halvorsen.blog

https://www.halvorsen.blog/documents/programming/python/

Contents

- Overview of GPIO
- LED
- PWM
- Push Button/Switch
- ADC (Analog to Digital Converter)
- TMP36
- ThingSpeak (Save Data to a Cloud Service)

Raspberry Pi



GPIO Pins

SD Card
(the Back)

Ethernet

Camera
Connector

USB A x 4

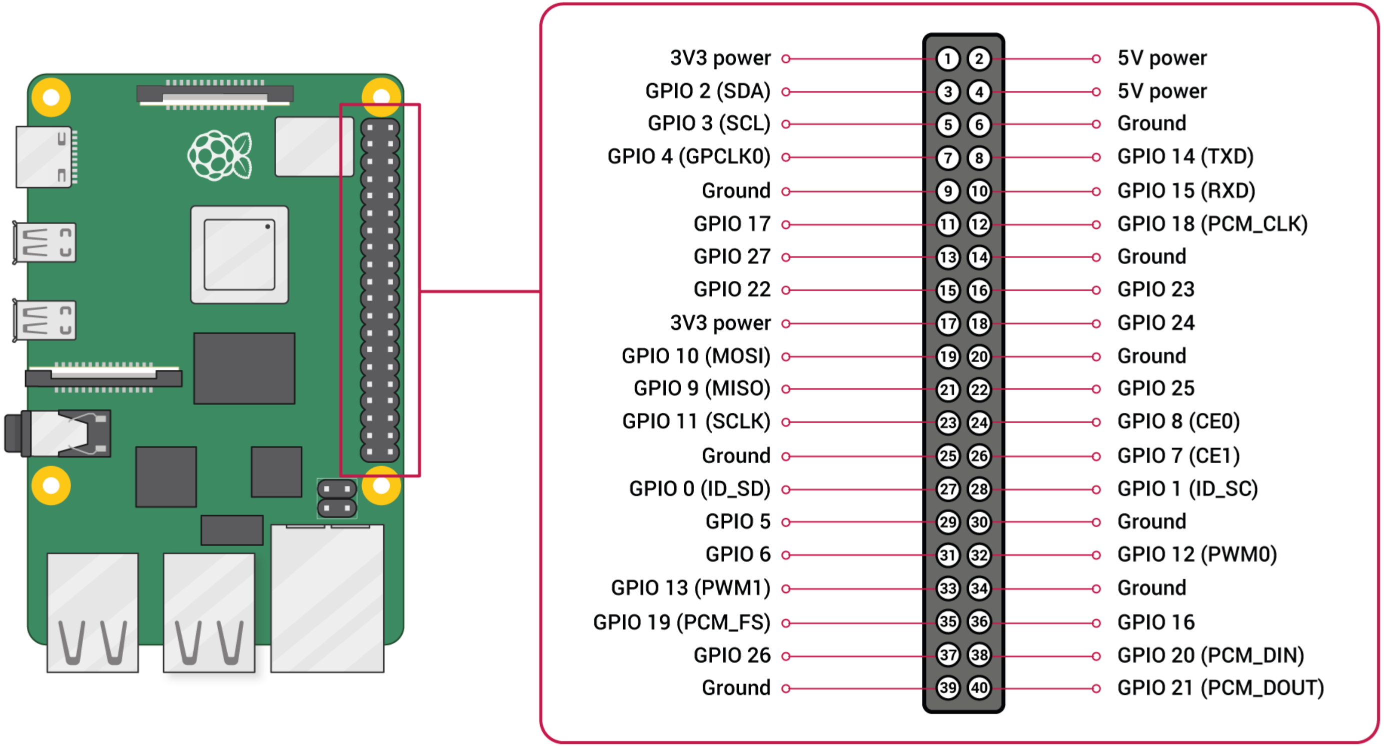
Power Supply (USB C)

micro HDMI x 2

Raspberry PI GPIO

Hans-Petter Halvorsen

GPIO



3V3 power	1	2	5V power
GPIO 2 (SDA)	3	4	5V power
GPIO 3 (SCL)	5	6	Ground
GPIO 4 (GPCLK0)	7	8	GPIO 14 (TXD)
Ground	9	10	GPIO 15 (RXD)
GPIO 17	11	12	GPIO 18 (PCM_CLK)
GPIO 27	13	14	Ground
GPIO 22	15	16	GPIO 23
3V3 power	17	18	GPIO 24
GPIO 10 (MOSI)	19	20	Ground
GPIO 9 (MISO)	21	22	GPIO 25
GPIO 11 (SCLK)	23	24	GPIO 8 (CE0)
Ground	25	26	GPIO 7 (CE1)
GPIO 0 (ID_SD)	27	28	GPIO 1 (ID_SC)
GPIO 5	29	30	Ground
GPIO 6	31	32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33	34	Ground
GPIO 19 (PCM_FS)	35	36	GPIO 16
GPIO 26	37	38	GPIO 20 (PCM_DIN)
Ground	39	40	GPIO 21 (PCM_DOUT)

A powerful feature of the Raspberry Pi is the GPIO (general-purpose input/output) pins. The Raspberry Pi has a 40-pin GPIO header as seen in the image

GPIO Features

The GPIO pins are Digital Pins which are either True (+3.3V) or False (0V). These can be used to turn on/off LEDs, etc.

The Digital Pins can be either Output or Input.

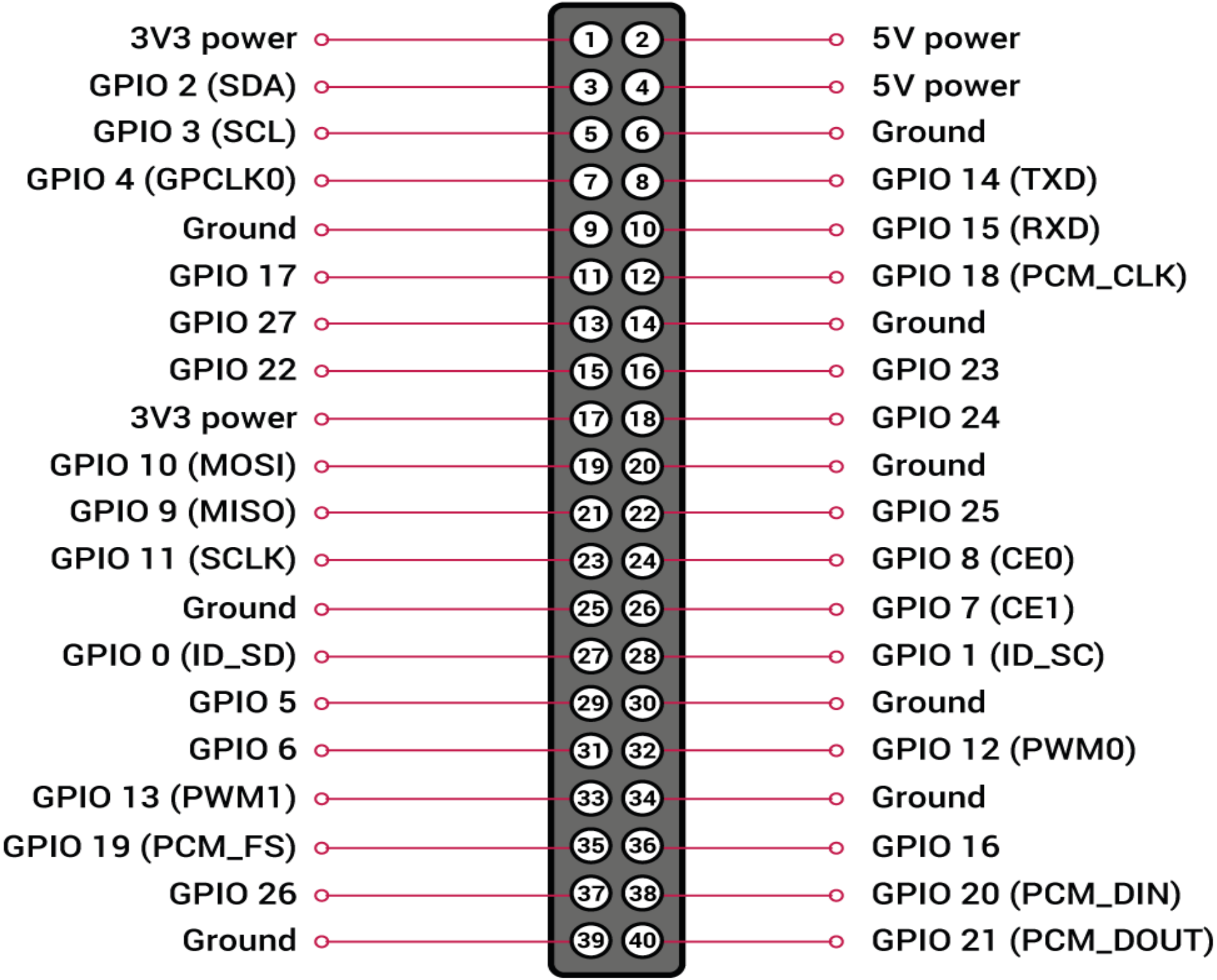
In addition, some of the pins also offer some other Features:

- PWM (Pulse Width Modulation)

Digital Buses (for reading data from Sensors, etc.):

- SPI
- I2C

GPIO



3V3 power	1	2	5V power
GPIO 2 (SDA)	3	4	5V power
GPIO 3 (SCL)	5	6	Ground
GPIO 4 (GPCLK0)	7	8	GPIO 14 (TXD)
Ground	9	10	GPIO 15 (RXD)
GPIO 17	11	12	GPIO 18 (PCM_CLK)
GPIO 27	13	14	Ground
GPIO 22	15	16	GPIO 23
3V3 power	17	18	GPIO 24
GPIO 10 (MOSI)	19	20	Ground
GPIO 9 (MISO)	21	22	GPIO 25
GPIO 11 (SCLK)	23	24	GPIO 8 (CE0)
Ground	25	26	GPIO 7 (CE1)
GPIO 0 (ID_SD)	27	28	GPIO 1 (ID_SC)
GPIO 5	29	30	Ground
GPIO 6	31	32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33	34	Ground
GPIO 19 (PCM_FS)	35	36	GPIO 16
GPIO 26	37	38	GPIO 20 (PCM_DIN)
Ground	39	40	GPIO 21 (PCM_DOUT)

GPIO with Python

GPIO Zero

- The GPIO Zero Python Library can be used to communicate with GPIO Pins
- The GPIO Zero Python Library comes preinstalled with the Raspberry Pi OS (so no additional installation is necessary)

Resources:

- https://www.raspberrypi.org/documentation/usage/gpio/python/
- https://pypi.org/project/gpiozero/
- https://gpiozero.readthedocs.io/en/stable/
- https://gpiozero.readthedocs.io/en/stable/recipes.html

RPi.GPIO

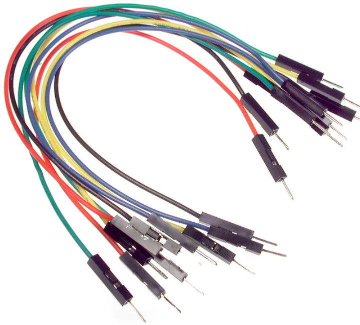
- Rpi.GPIO is a module controlling the GPIO pins on the Raspberry Pi
- RPi.GPIO is a more “low-level“ Python Library than GPIO Zero. Actually, GPIO Zero is using RPi.GPIO
- The RPi.GPIO Python Library comes preinstalled with the Raspberry Pi OS (so no additional installation is necessary)

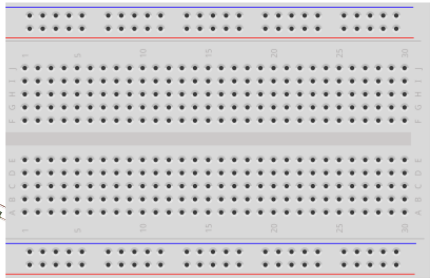
https://pypi.org/project/RPi.GPIO/

LED

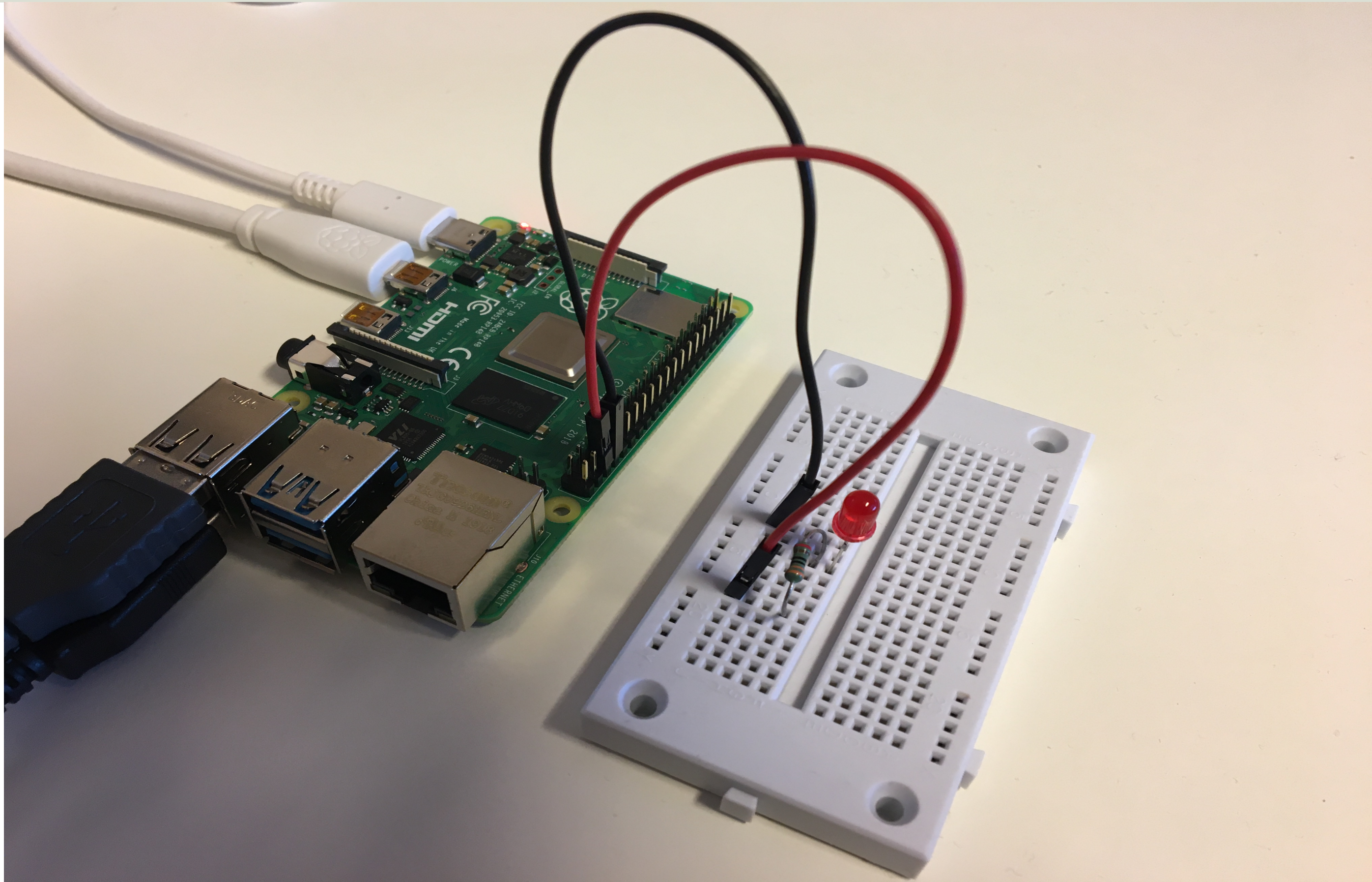
Necessary Equipment

- Raspberry Pi
- Breadboard
- LED
- Resistor, $R = 270\Omega$
- Wires (Jumper Wires)

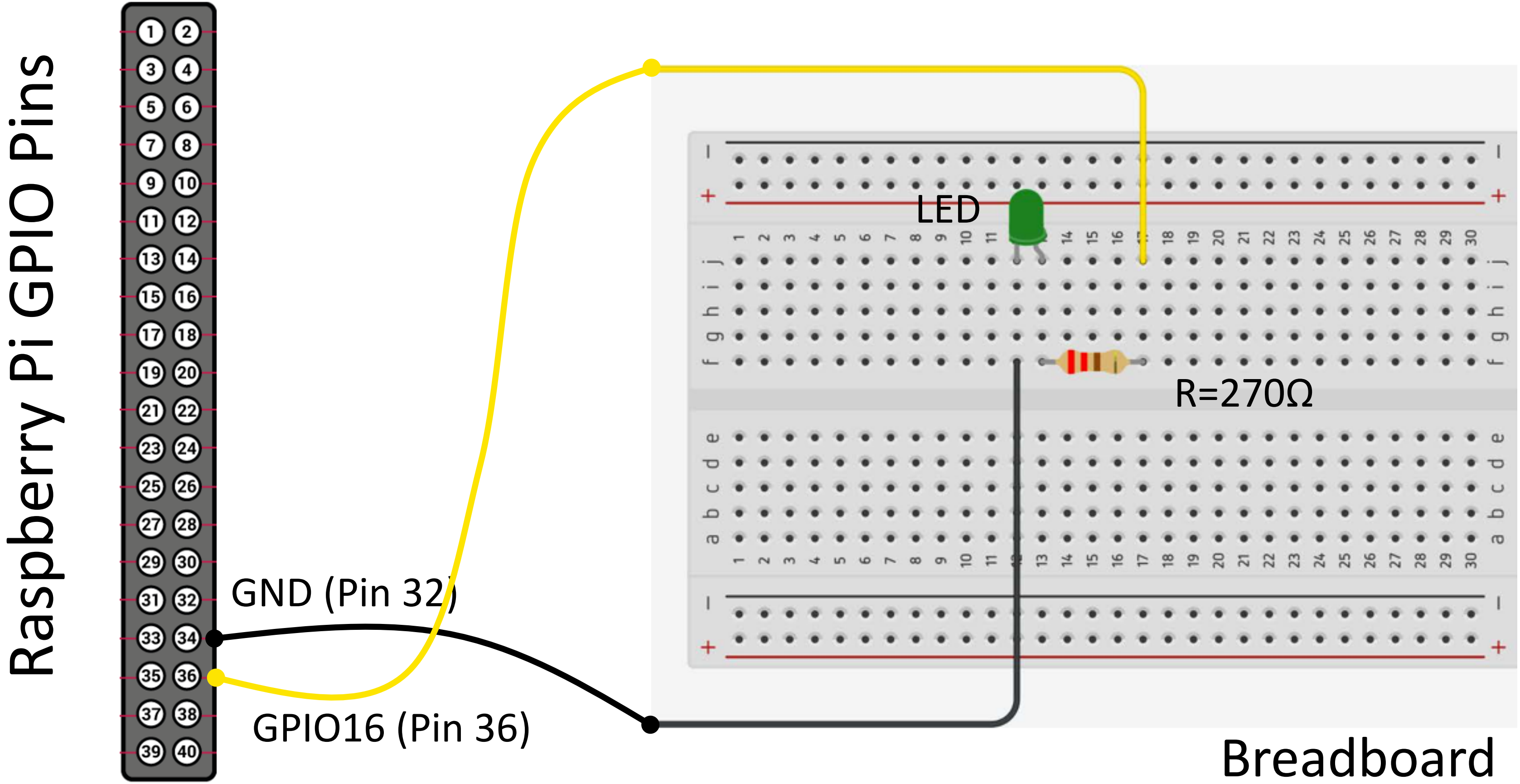




Setup and Wiring

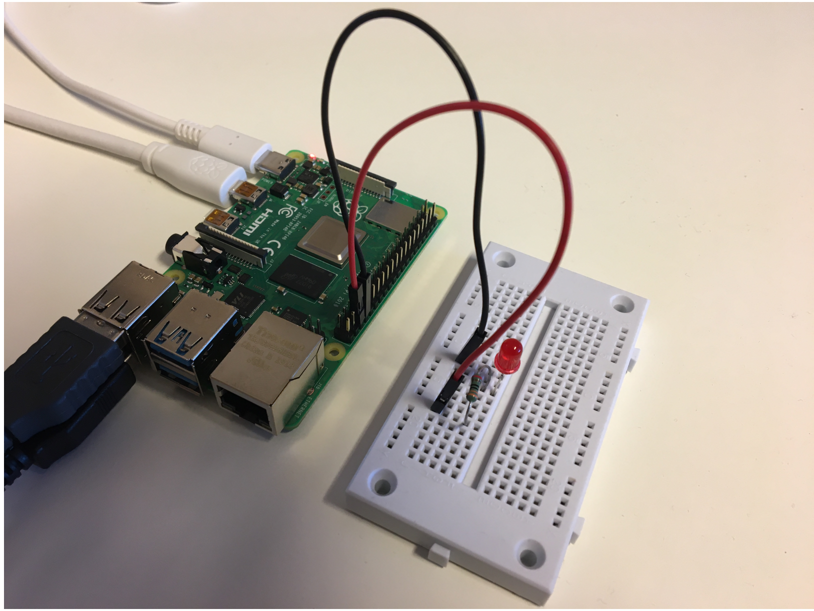


LED Example


Raspberry Pi GPIO Pins
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
GND (Pin 32)
GPIO16 (Pin 36)
LED
R=270Ω
Breadboard

LED Example

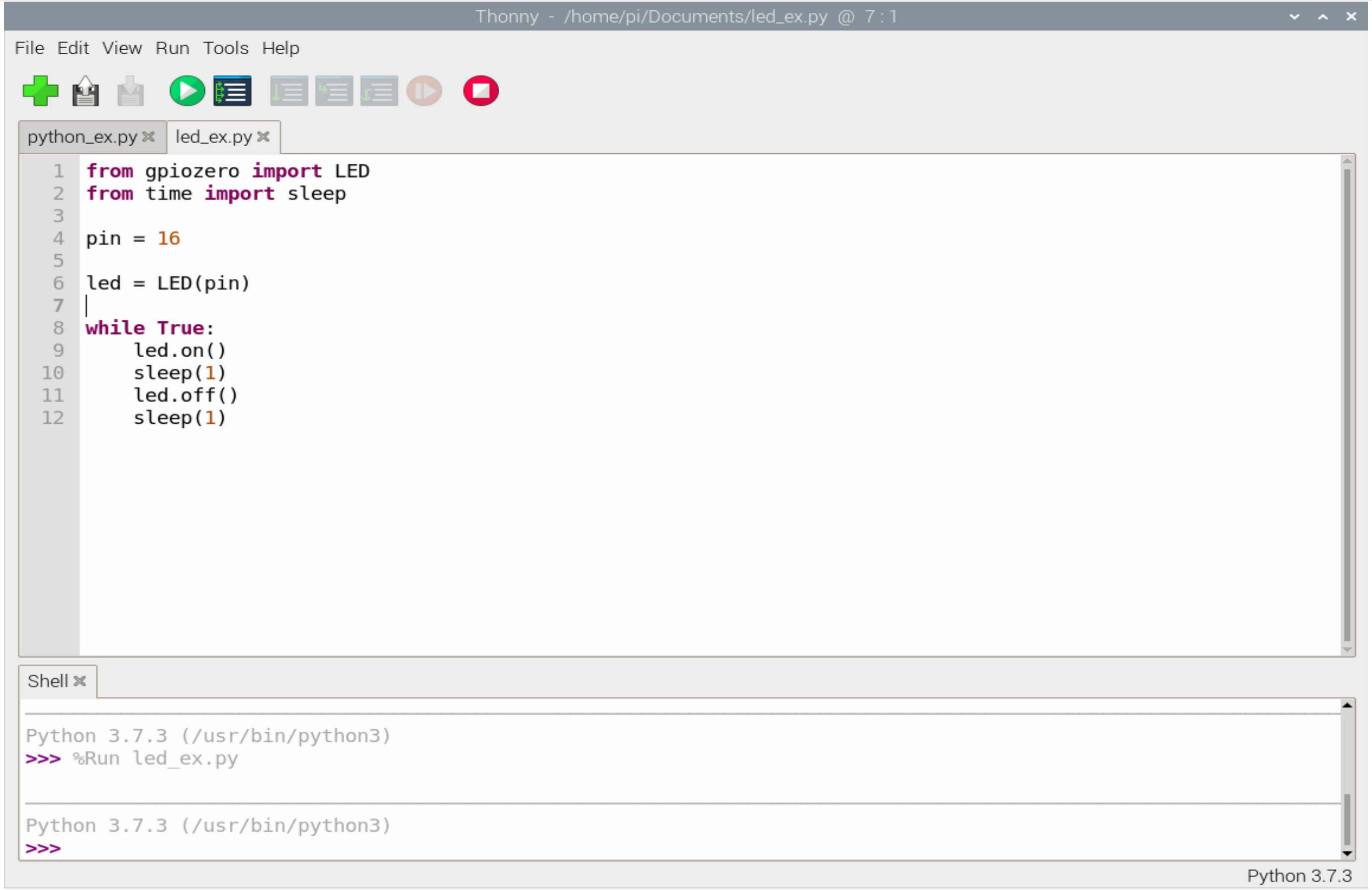
This Example “Runs for ever“



```
from gpiozero import LED
from time import sleep

pin = 16
led = LED(pin)

while True:
    led.on()
    sleep(1)
    led.off()
    sleep(1)
```

https://www.raspberrypi.org/documentation/usage/gpio/python/

LED Example



Thonny - /home/pi/Documents/led_ex.py @ 7 : 1
File Edit View Run Tools Help
python_ex.py
led_ex.py
Shell
Python 3.7.3

```
from gpiozero import LED
from time import sleep

pin = 16

led = LED(pin)

while True:
    led.on()
    sleep(1)
    led.off()
    sleep(1)
```

```
Python 3.7.3 (/usr/bin/python3)
>>> %Run led_ex.py

Python 3.7.3 (/usr/bin/python3)
>>>
```

LED Example

This example turns a LED on/off 10 times

```
from gpiozero import LED
from time import sleep

pin = 16
led = LED(pin)

N = 10
for x in range(N):
    led.on()
    sleep(1)
    led.off()
    sleep(1)
```

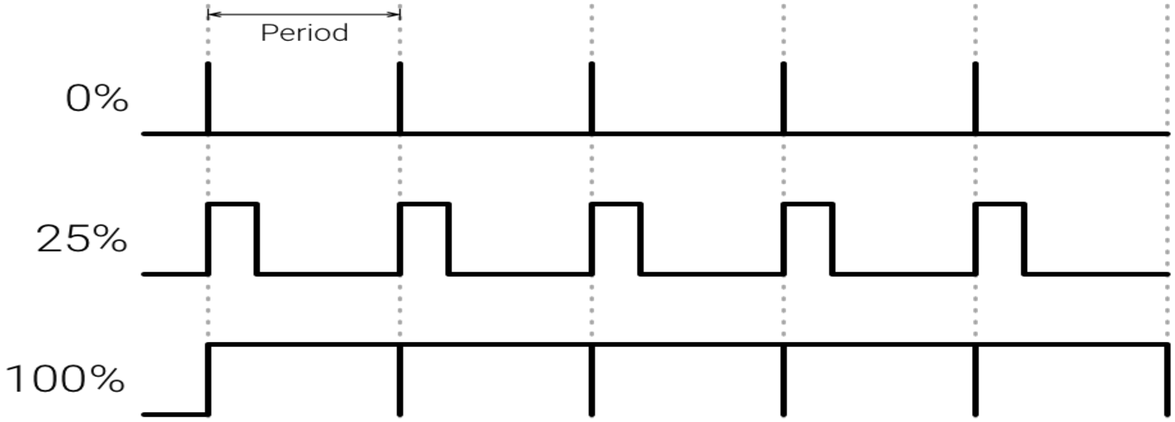
PWM

Pulse Width Modulation

PWM

PWM is a digital (i.e., square wave) signal that oscillates according to a given *frequency* and *duty cycle*.
The frequency (expressed in Hz) describes how often the output pulse repeats.
The period is the time each cycle takes and is the inverse of frequency.
The duty cycle (expressed as a percentage) describes the width of the pulse within that frequency window.

You can adjust the duty cycle to increase or decrease the average "on" time of the signal. The following diagram shows pulse trains at 0%, 25%, and 100% duty:



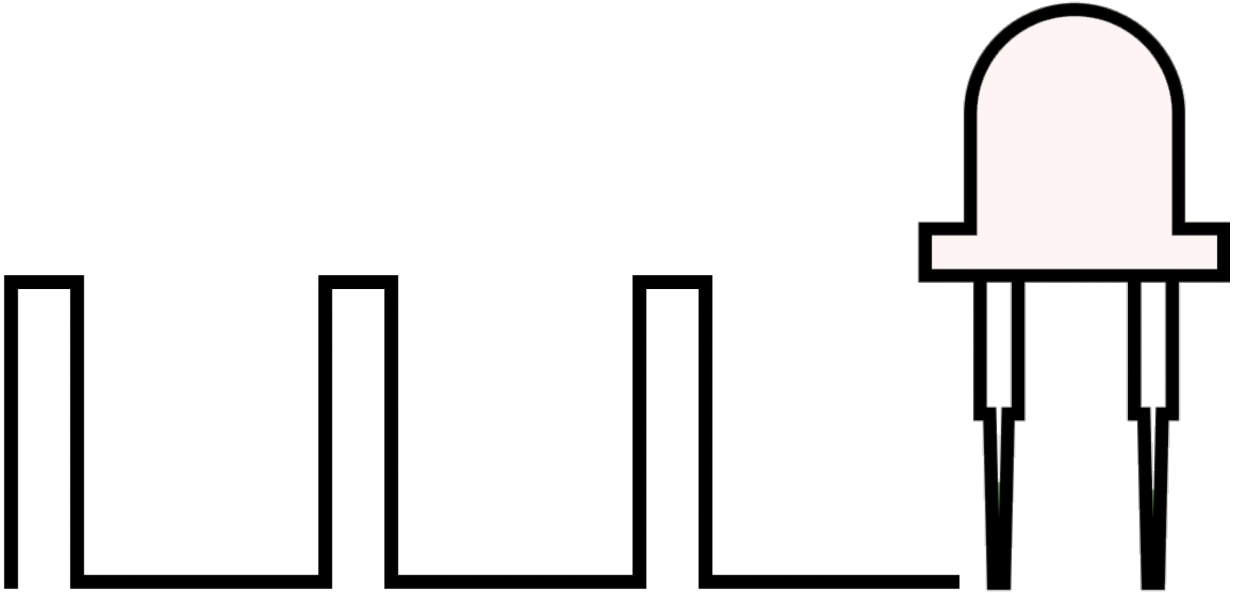
Controlling LED Brightness using PWM

- We've seen how to turn an LED on and off, but how do we control its brightness levels?
- An LED's brightness is determined by controlling the amount of current flowing through it, but that requires a lot more hardware components.
- A simple trick we can do is to flash the LED faster than the eye can see!
- By controlling the amount of time the LED is on versus off, we can change its perceived brightness.
- This is known as *Pulse Width Modulation* (PWM).

https://learn.sparkfun.com/tutorials/python-programming-tutorial-getting-started-with-the-raspberry-pi/experiment-1-digital-input-and-output

Controlling LED Brightness using PWM

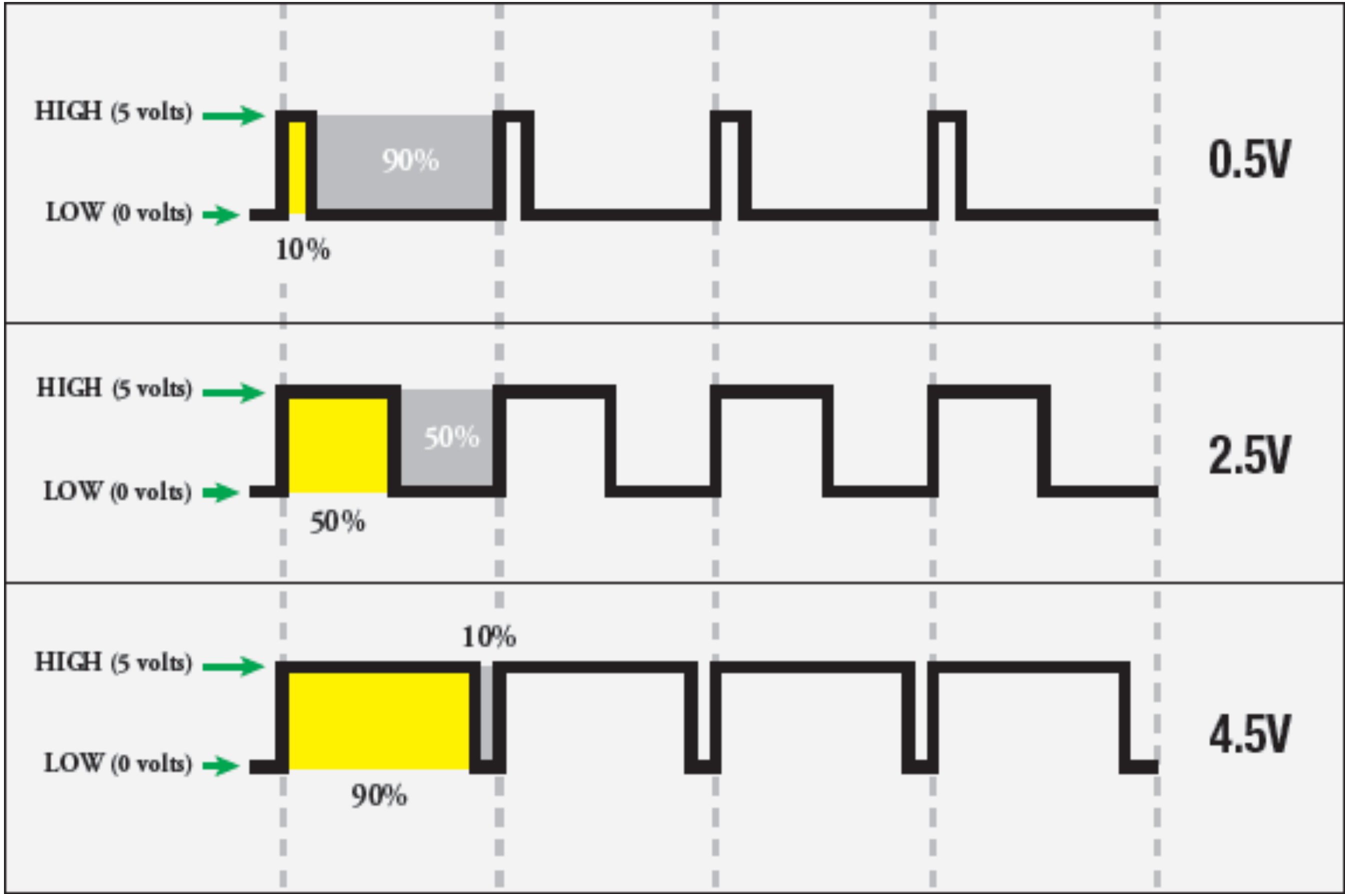
Below we see how we can use PWM to control the brightness of a LED



https://www.electronicwings.com/raspberry-pi/raspberry-pi-pwm-generation-using-python-and-c

PWM as “Analog Out“

The Raspberry Pi has no real Analog Out pins, but we can use a PWM pin.
PWM can be used to control brightness of a LED, control the speed of a Fan, control a DC Motor, etc.



GPIO Zero

PWM

```
from time import sleep
import numpy as np
from gpiozero import PWMLED

pin = 23
led = PWMLED(pin)

start = 0
stop = 1
step = 0.1
level = np.arange(start, stop, step)

for x in level:
    led.value = x
    sleep(1)

led.off()
```

RPi.GPIO

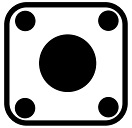
PWM

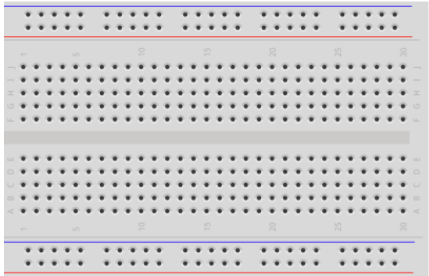
```
import time
import RPi.GPIO as GPIO

# Pin definitions
led_pin = 23

# Use "GPIO" pin numbering
GPIO.setmode(GPIO.BCM)

# Set LED pin as output
GPIO.setup(led_pin, GPIO.OUT)

# Initialize pwm object with 50 Hz and 0% duty cycle
pwm = GPIO.PWM(led_pin, 50)
pwm.start(0)

pwm.ChangeDutyCycle(10)
time.sleep(2)
pwm.ChangeDutyCycle(50)
time.sleep(2)
pwm.ChangeDutyCycle(90)
time.sleep(2)

# Stop, cleanup, and exit
pwm.stop()
GPIO.cleanup()
```

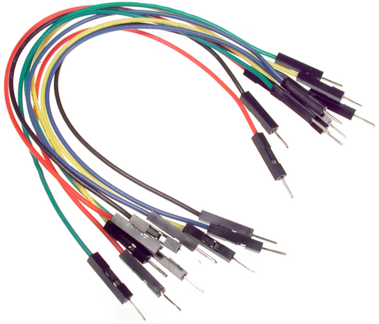
Push Button

Necessary Equipment

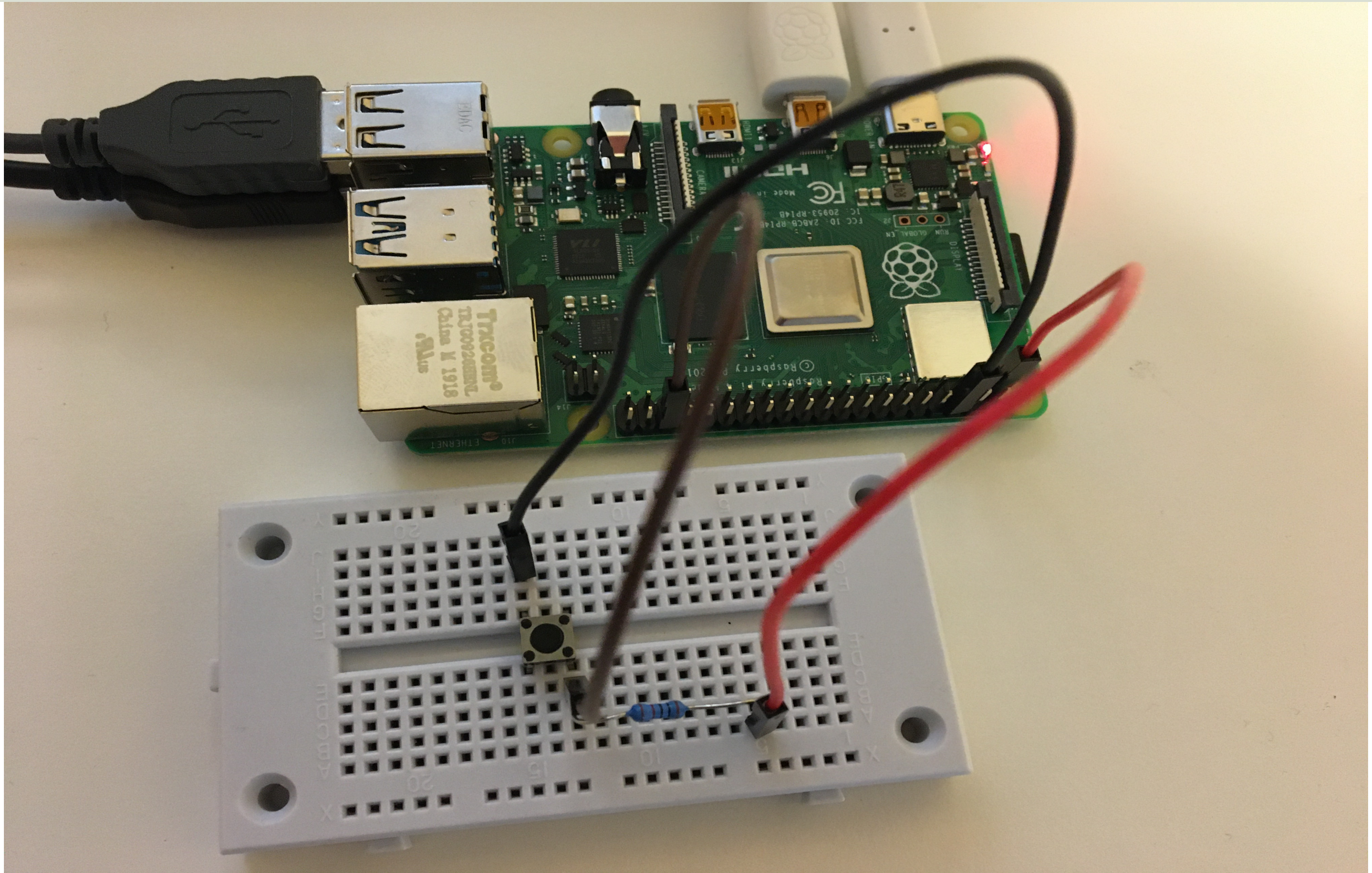
- Raspberry Pi
- Breadboard
- Push Button
- LED
- Resistors, $R = 270\Omega, R = 10k\Omega$
- Wires (Jumper Wires)





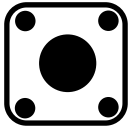


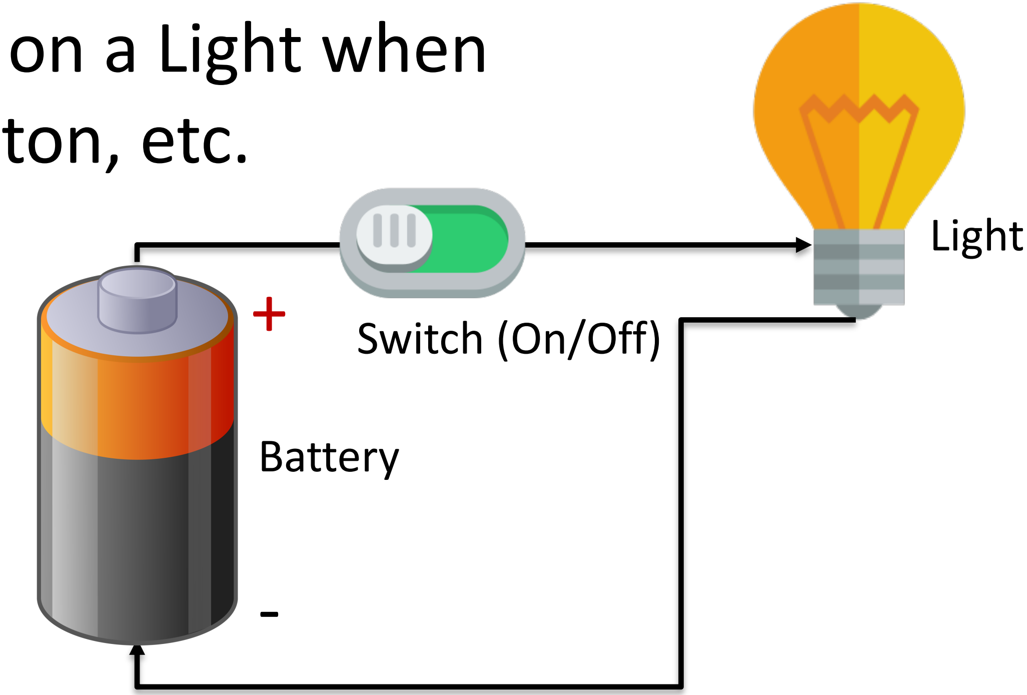
Setup and Wiring



Push Button/Switch

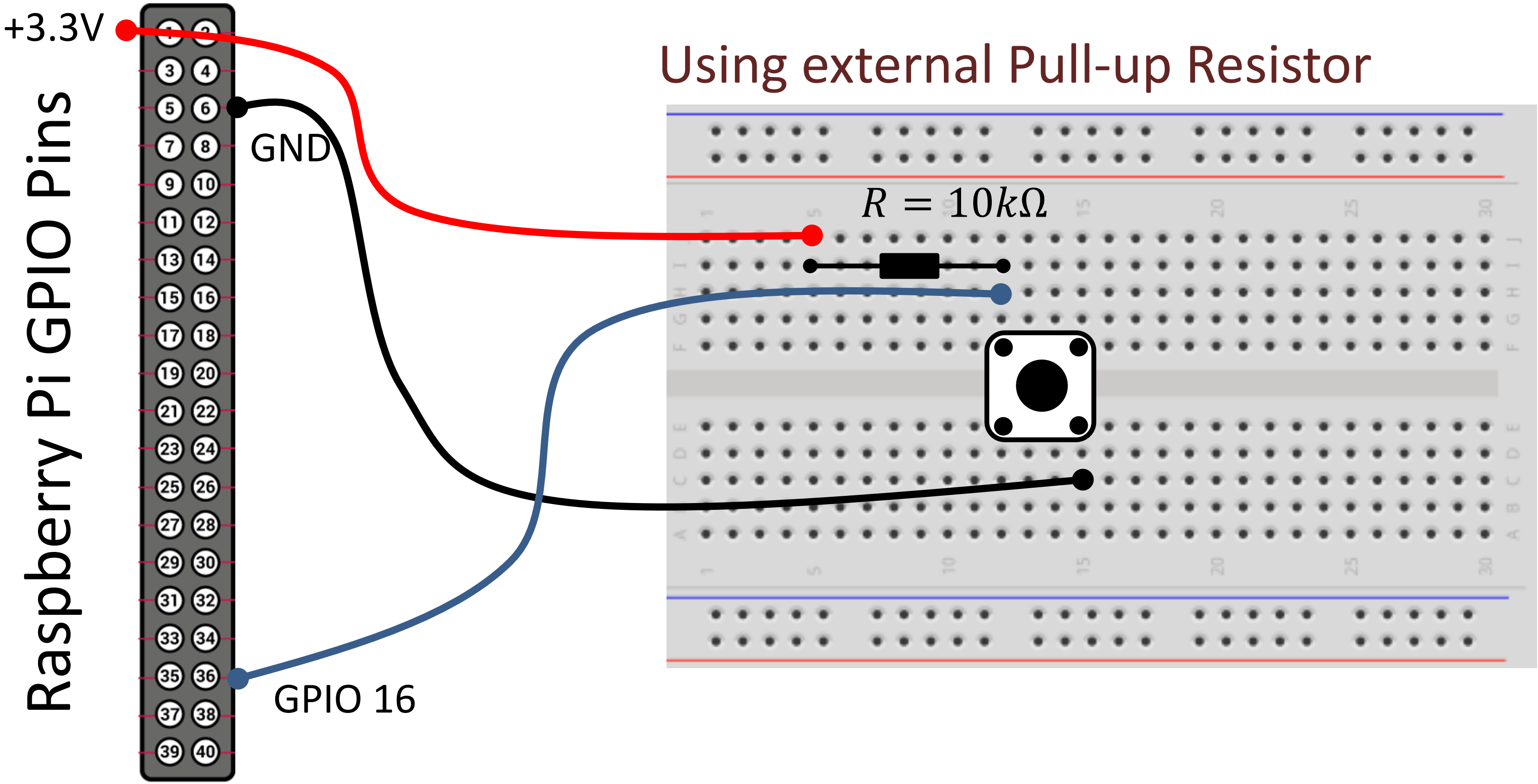
- Pushbuttons or switches connect two points in a circuit when you press them.
- You can use it to turn on a Light when holding down the button, etc.



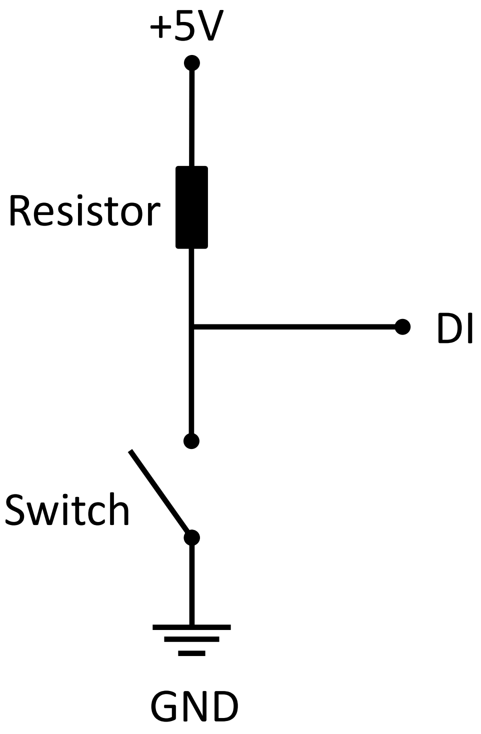


Button Setup

Using external Pull-up Resistor

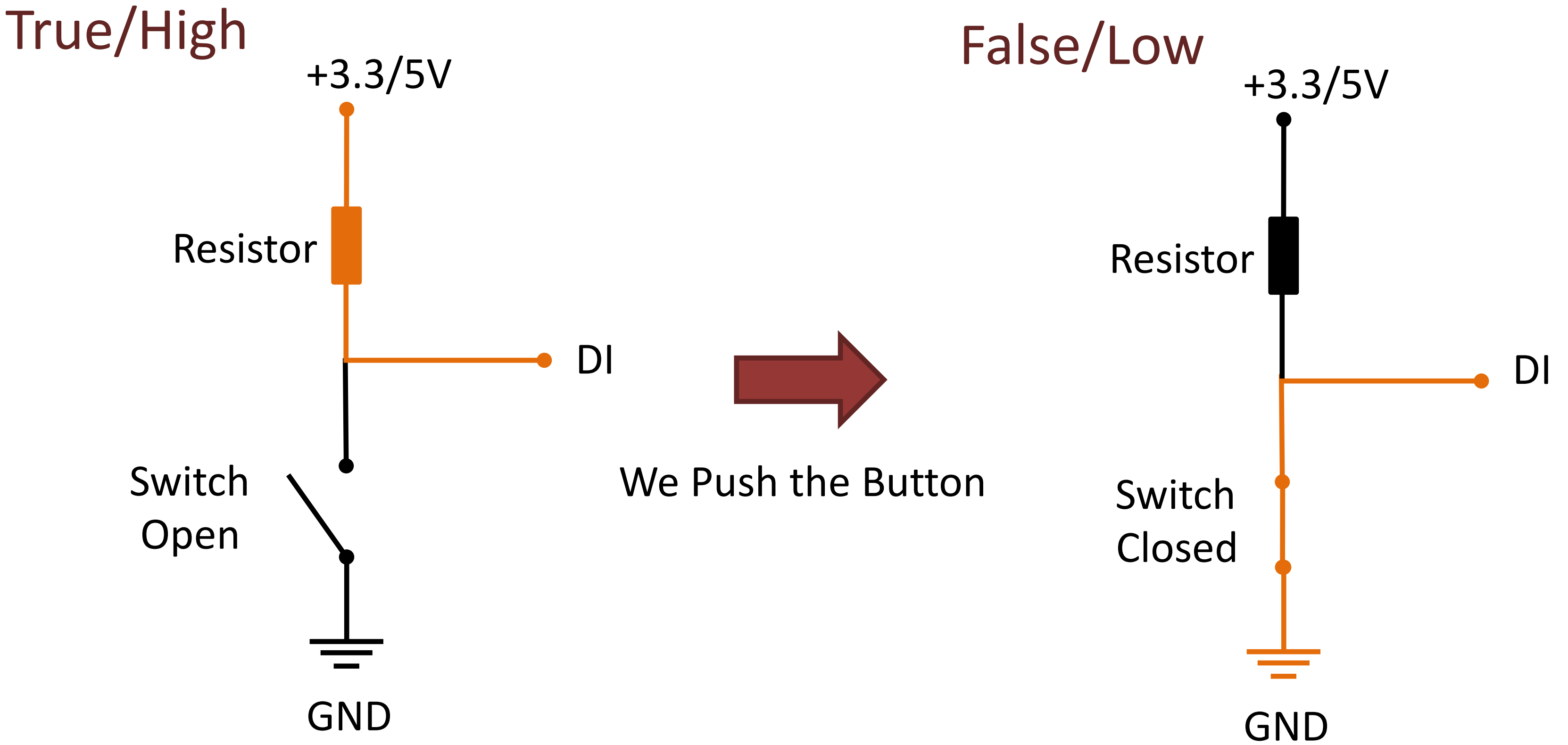


Pull-up Resistor



- When the pushbutton is open (unpressed) there is a connection between 3.3/5V and the DI pin.
- This means the default state is **True** (High).
- When the button is closed (pressed), the state goes to **False** (Low).

Pull-up Resistor

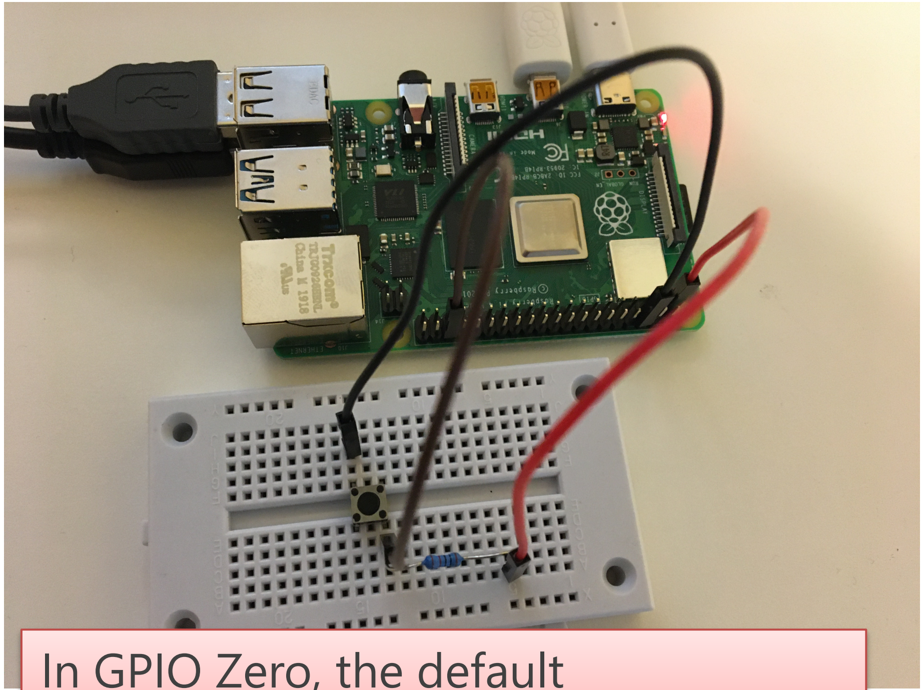


Pull-down/Pull-up Resistor

Why do we need a pull-up or pull-down resistor in the circuit?

- If you disconnect the digital I/O pin from everything, it will behave in an irregular way.
- This is because the input is "floating" - that is, it will randomly return either HIGH or LOW.
- That's why you need a pull-up or pull-down resistor in the circuit.

Button Example

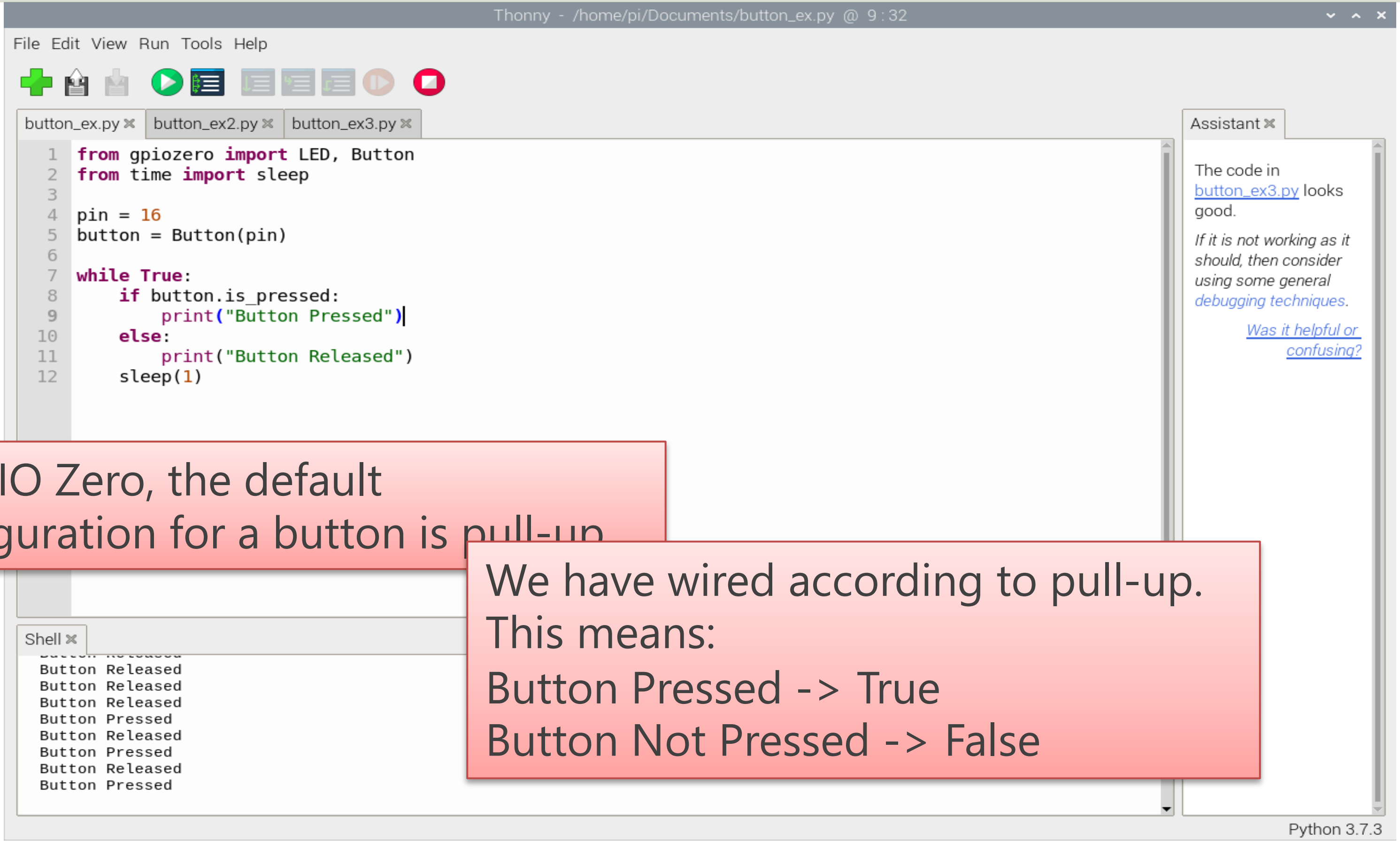


In GPIO Zero, the default configuration for a button is pull-up

```
from gpiozero import Button
from time import sleep
pin = 16
button = Button(pin)

while True:
    if button.is_pressed:
        print("Pressed")
    else:
        print("Released")
    sleep(1)
```

https://www.raspberrypi.org/documentation/usage/gpio/python/

Button Example


Thonny - /home/pi/Documents/button_ex.py @ 9 : 32
File Edit View Run Tools Help
button_ex.py
button_ex2.py
button_ex3.py
Assistant
Shell
Python 3.7.3

```python
from gpiozero import LED, Button
from time import sleep

pin = 16
button = Button(pin)

while True:
    if button.is_pressed:
        print("Button Pressed")
    else:
        print("Button Released")
    sleep(1)
```

```
Button Released
Button Released
Button Released
Button Released
Button Pressed
Button Released
Button Pressed
Button Released
Button Pressed
```

The code in button_ex3.py looks good.

If it is not working as it should, then consider using some general debugging techniques.

Was it helpful or confusing?

In GPIO Zero, the default configuration for a button is pull-up

We have wired according to pull-up.
This means:
Button Pressed -> True
Button Not Pressed -> False

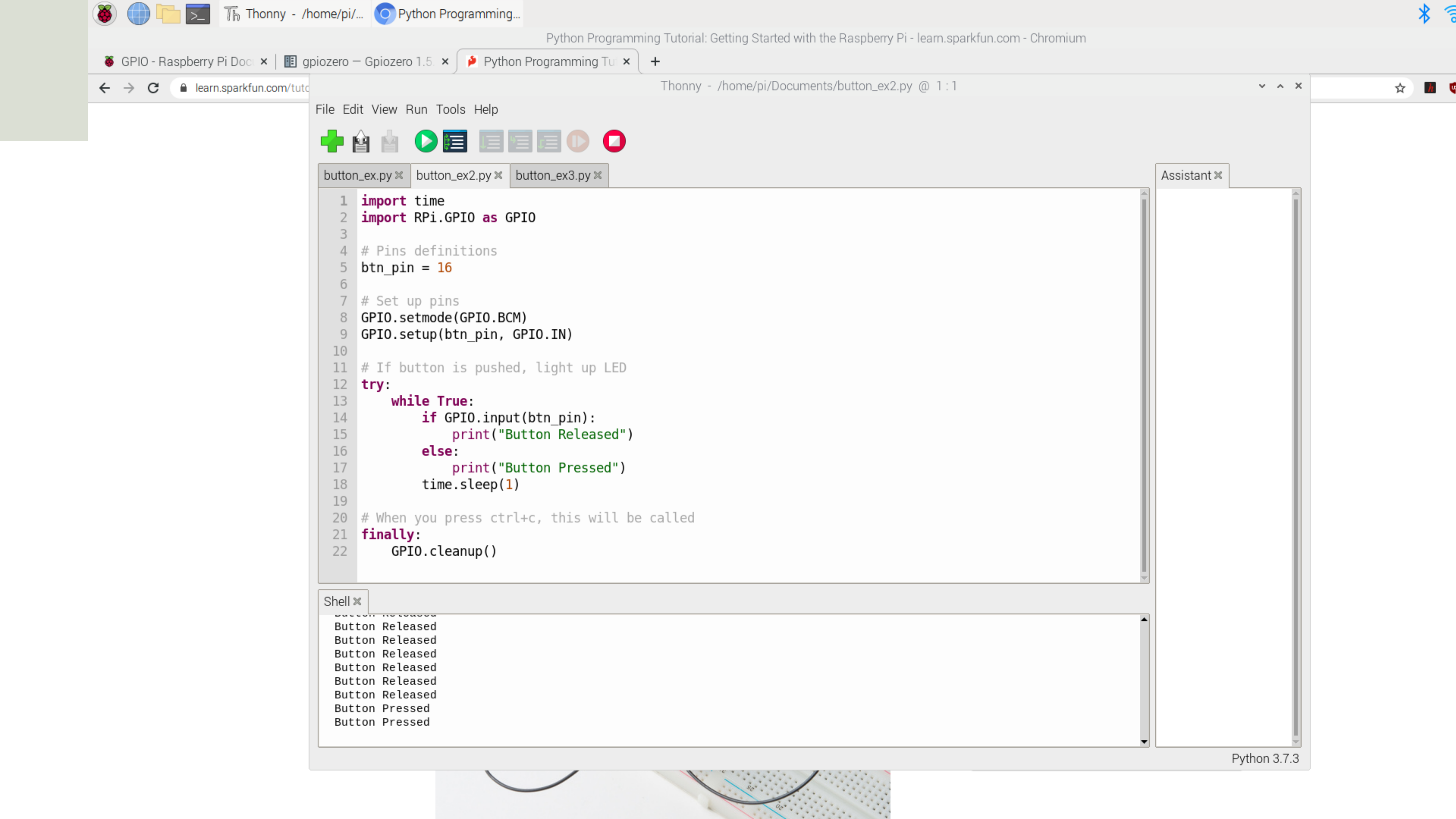
Button Ex.2

Here is the **RPi.GPIO** Python Library used

In RPi.GPIO, the default configuration for a button is pull-down

We have wired according to pull-up.
This means:
Button Pressed -> False
Button Not Pressed -> True

```
import time
import RPi.GPIO as GPIO

# Pins definitions
btn_pin = 16

# Set up pins
GPIO.setmode(GPIO.BCM)
GPIO.setup(btn_pin, GPIO.IN)

# If button is pushed, light up LED
try:
    while True:
        if GPIO.input(btn_pin):
            print("Button Released")
        else:
            print("Button Pressed")
        time.sleep(1)

# When you press ctrl+c, this will be called
finally:
    GPIO.cleanup()
```



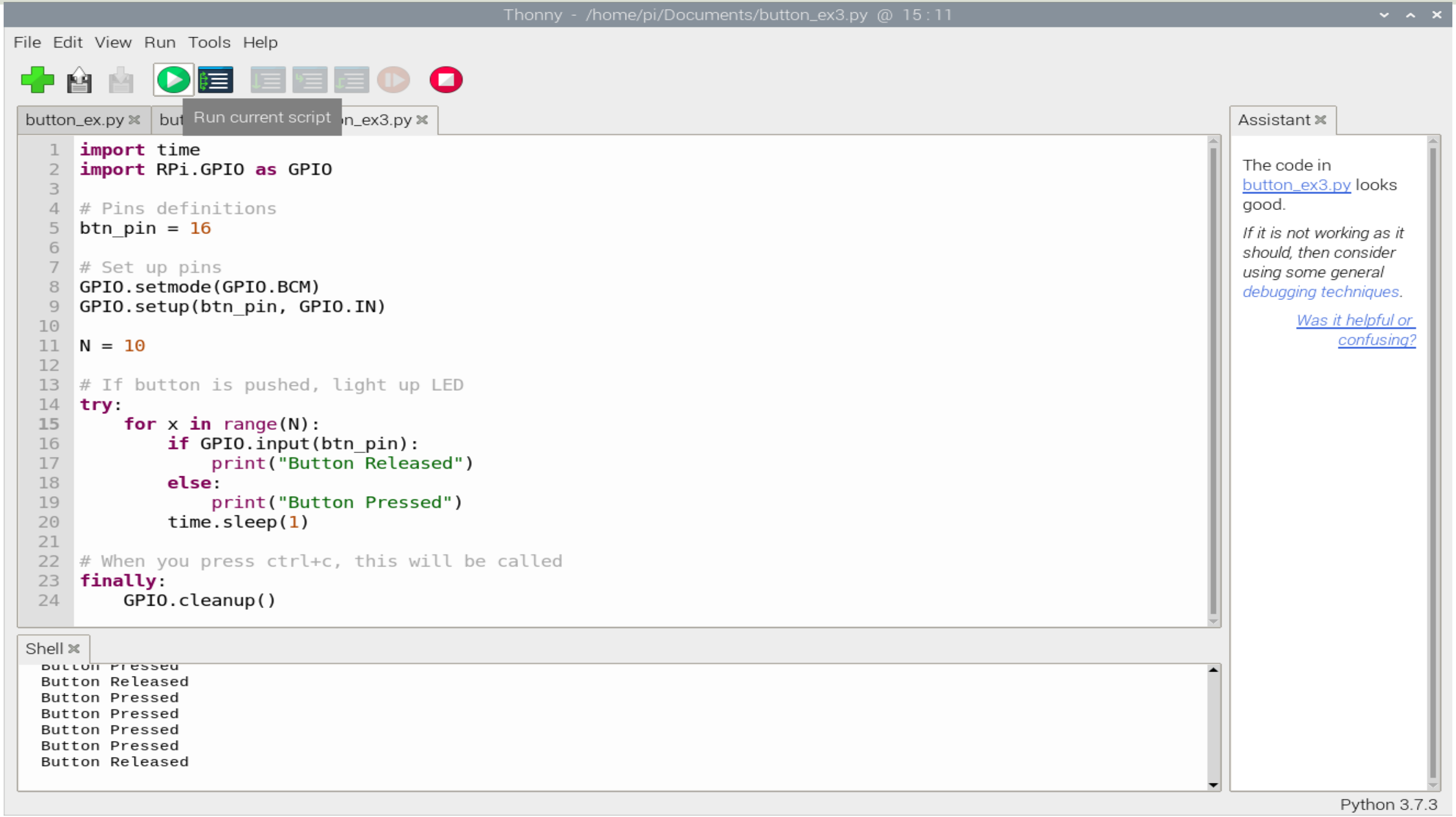
```python
import time
import RPi.GPIO as GPIO

# Pins definitions
btn_pin = 16

# Set up pins
GPIO.setmode(GPIO.BCM)
GPIO.setup(btn_pin, GPIO.IN)

# If button is pushed, light up LED
try:
    while True:
        if GPIO.input(btn_pin):
            print("Button Released")
        else:
            print("Button Pressed")
        time.sleep(1)

# When you press ctrl+c, this will be called
finally:
    GPIO.cleanup()
```

```
Button Released
Button Released
Button Released
Button Released
Button Released
Button Released
Button Pressed
Button Pressed
```

Button Ex.3

```
import time
import RPi.GPIO as GPIO

# Pins definitions
btn_pin = 16

# Set up pins
GPIO.setmode(GPIO.BCM)
GPIO.setup(btn_pin, GPIO.IN)

N = 10
# If button is pushed, light up LED
try:
    for x in range(N):
        if GPIO.input(btn_pin):
            print("Button Released")
        else:
            print("Button Pressed")
        time.sleep(1)

# When you press ctrl+c, this will be
called
finally:
    GPIO.cleanup()
```

Button Example3



```python
import time
import RPi.GPIO as GPIO

# Pins definitions
btn_pin = 16

# Set up pins
GPIO.setmode(GPIO.BCM)
GPIO.setup(btn_pin, GPIO.IN)

N = 10

# If button is pushed, light up LED
try:
    for x in range(N):
        if GPIO.input(btn_pin):
            print("Button Released")
        else:
            print("Button Pressed")
        time.sleep(1)

# When you press ctrl+c, this will be called
finally:
    GPIO.cleanup()
```

```
Button Pressed
Button Released
Button Pressed
Button Pressed
Button Pressed
Button Pressed
Button Released
```

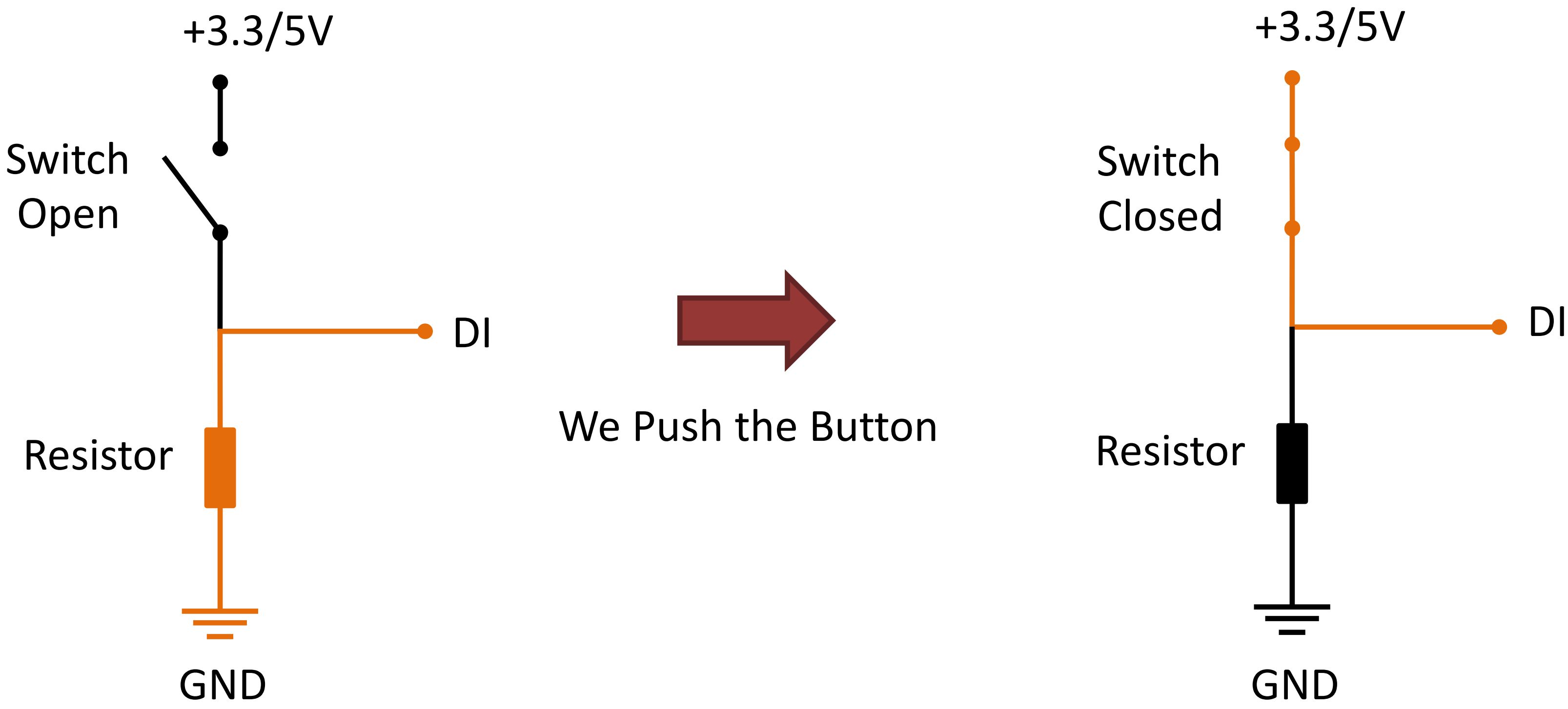
The code in button_ex3.py looks good.

If it is not working as it should, then consider using some general debugging techniques.

Was it helpful or confusing?

Pull-down Resistor

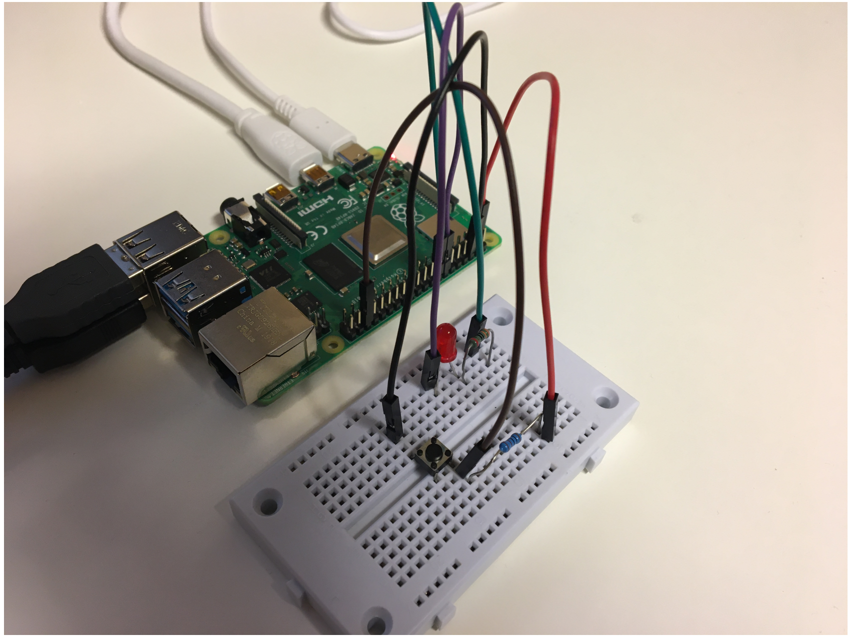
We could also have wired according to a “Pull-down“ Resistor



False/Low

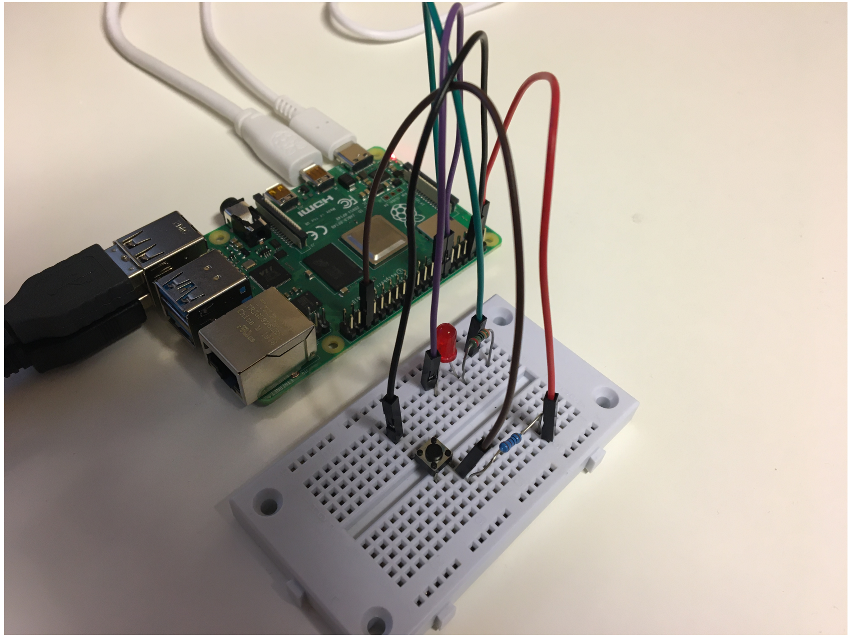
True/High

Button + LED Example



```python
from gpiozero import LED, Button
from time import sleep

pin_btn = 16
button = Button(pin_btn)
pin_led = 23
led = LED(pin_led)

while True:
    if button.is_pressed:
        led.on()
    else:
        led.off()
    sleep(1)
```

Button + LED Example



```
import time
import RPi.GPIO as GPIO

# Pin definitions
led_pin = 23
btn_pin = 16

# Suppress warnings
GPIO.setwarnings(False)

# Use "GPIO" pin numbering
GPIO.setmode(GPIO.BCM)
# Set Button pin as input
GPIO.setup(btn_pin, GPIO.IN)
# Set LED pin as output
GPIO.setup(led_pin, GPIO.OUT)

# Blink forever
while True:
    if GPIO.input(btn_pin):
        GPIO.output(led_pin, GPIO.LOW)  # Turn LED off
    else:
        GPIO.output(led_pin, GPIO.HIGH) # Turn LED on

    time.sleep(1)
```

SPI

Serial Peripheral Interface (SPI)

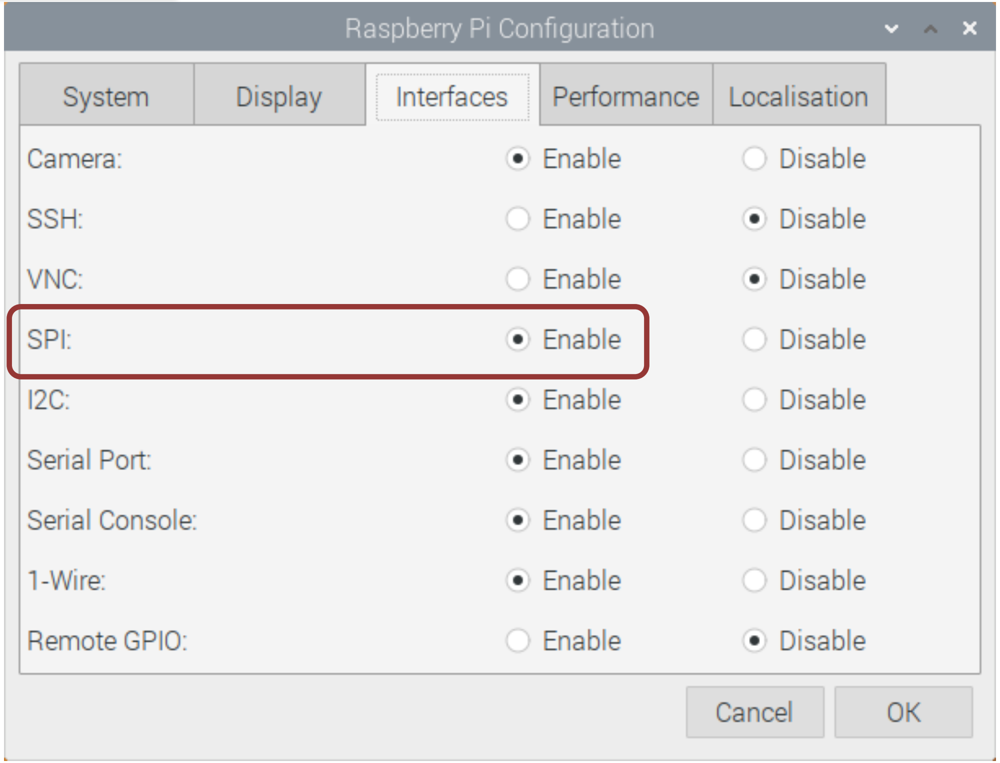
SPI

- Serial Peripheral Interface (SPI)
- SPI is an interface to communicate with different types of electronic components like Sensors, Analog to Digital Converts (ADC), etc. that supports the SPI interface
- Thousands of different Components and Sensors supports the SPI interface

https://www.raspberrypi.org/documentation/hardware/raspberrypi/spi/

Access SPI on Raspberry Pi

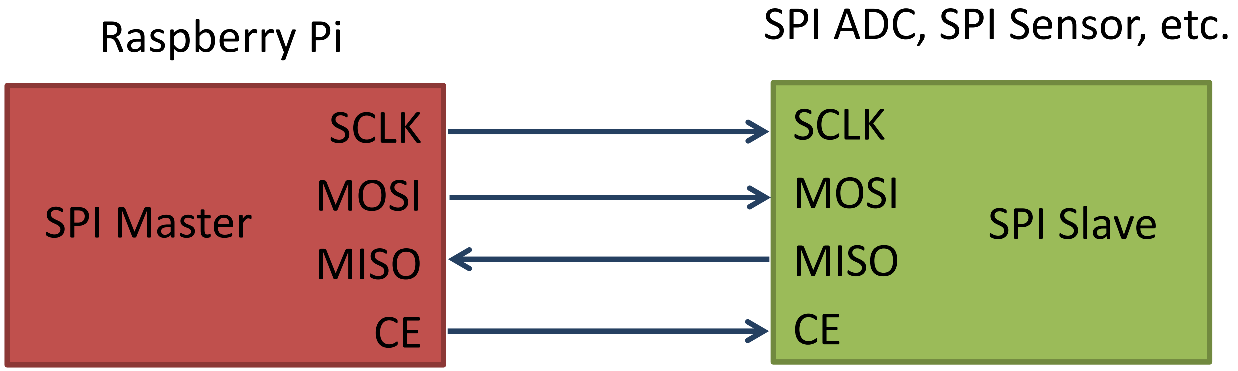
You need to Enable SPI on the Raspberry Pi



Camera:	● Enable	○ Disable
SSH:	○ Enable	● Disable
VNC:	○ Enable	● Disable
SPI:	● Enable	○ Disable
I2C:	● Enable	○ Disable
Serial Port:	● Enable	○ Disable
Serial Console:	● Enable	○ Disable
1-Wire:	● Enable	○ Disable
Remote GPIO:	○ Enable	● Disable

SPI Interface

SPI devices communicate in full duplex mode using a master-slave architecture with a single master

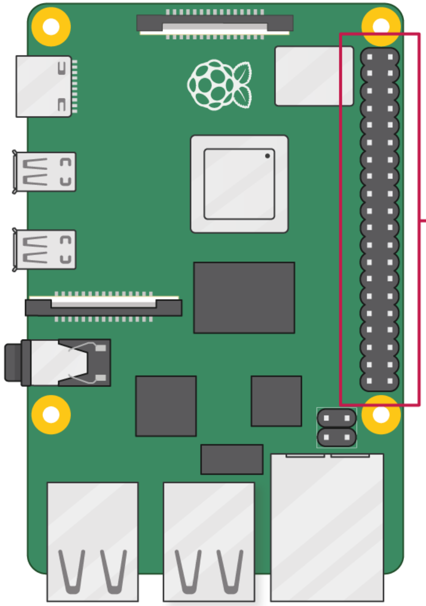


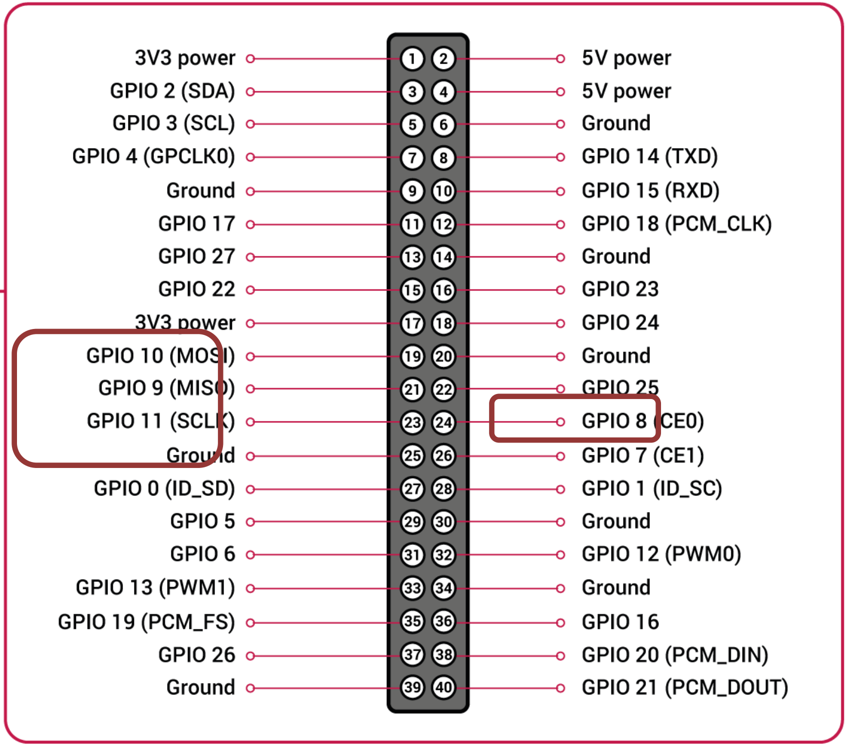
The SPI bus specifies four logic signals:

- **SCLK**: Serial Clock (output from master)
- **MOSI**: Master Out Slave In (data output from master)
- **MISO**: Master In Slave Out (data output from slave)
- **CE** (often also called SS - Slave Select): Chip Select (often active low, output from master)

SPI Wiring on Raspberry Pi

GPIO 40 pins Connector





3V3 power	1	2	5V power
GPIO 2 (SDA)	3	4	5V power
GPIO 3 (SCL)	5	6	Ground
GPIO 4 (GPCLK0)	7	8	GPIO 14 (TXD)
Ground	9	10	GPIO 15 (RXD)
GPIO 17	11	12	GPIO 18 (PCM_CLK)
GPIO 27	13	14	Ground
GPIO 22	15	16	GPIO 23
3V3 power	17	18	GPIO 24
GPIO 10 (MOSI)	19	20	Ground
GPIO 9 (MISO)	21	22	GPIO 25
GPIO 11 (SCLK)	23	24	GPIO 8 (CE0)
Ground	25	26	GPIO 7 (CE1)
GPIO 0 (ID_SD)	27	28	GPIO 1 (ID_SC)
GPIO 5	29	30	Ground
GPIO 6	31	32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33	34	Ground
GPIO 19 (PCM_FS)	35	36	GPIO 16
GPIO 26	37	38	GPIO 20 (PCM_DIN)
Ground	39	40	GPIO 21 (PCM_DOUT)

https://www.halvorsen.blog

ADC

Analog to Digital Converter

Hans-Petter Halvorsen

ADC

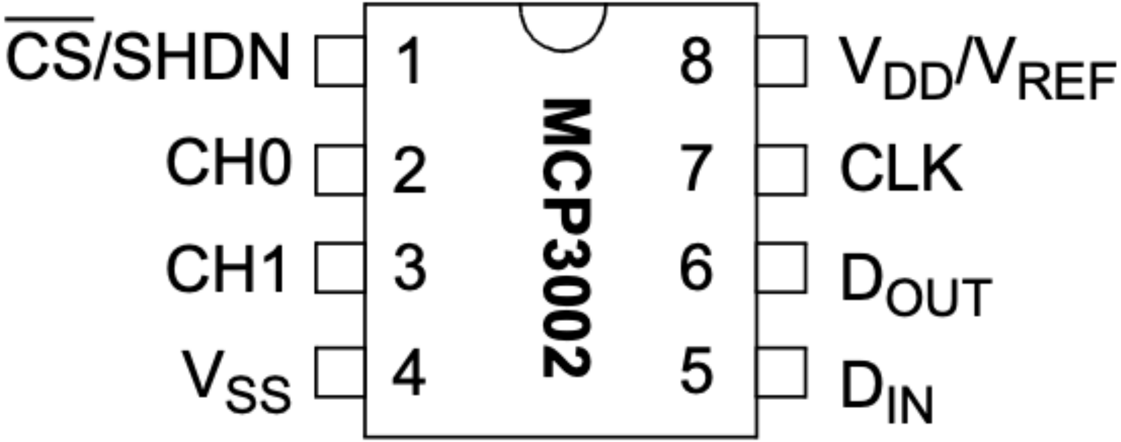
- The Raspberry Pi has only Digital pins on the GPIO connector
- If you want to use an Analog electric component or an Analog Sensor together with Raspberry Pi, you need to connect it through an external ADC chip
- ADC – Analog to Digital Converter

https://en.wikipedia.org/wiki/Analog-to-digital_converter

MCP3002 ADC chip

The MCP3002 is a 10-bit analog to digital converter with 2 channels (0-1).

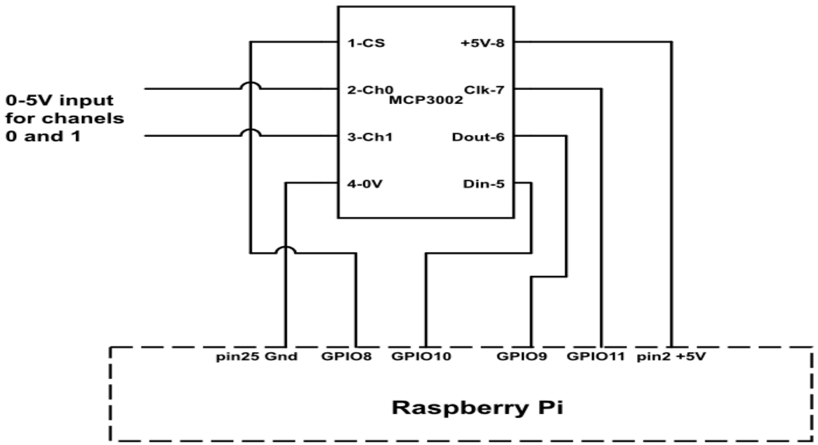
The MCP3002 uses a SPI Interface

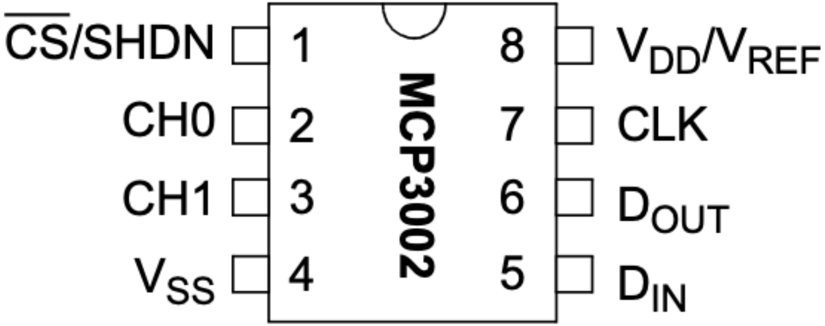


http://ww1.microchip.com/downloads/en/DeviceDoc/21294E.pdf

https://learn.sparkfun.com/tutorials/python-programming-tutorial-getting-started-with-the-raspberry-pi/experiment-3-spi-and-analog-input

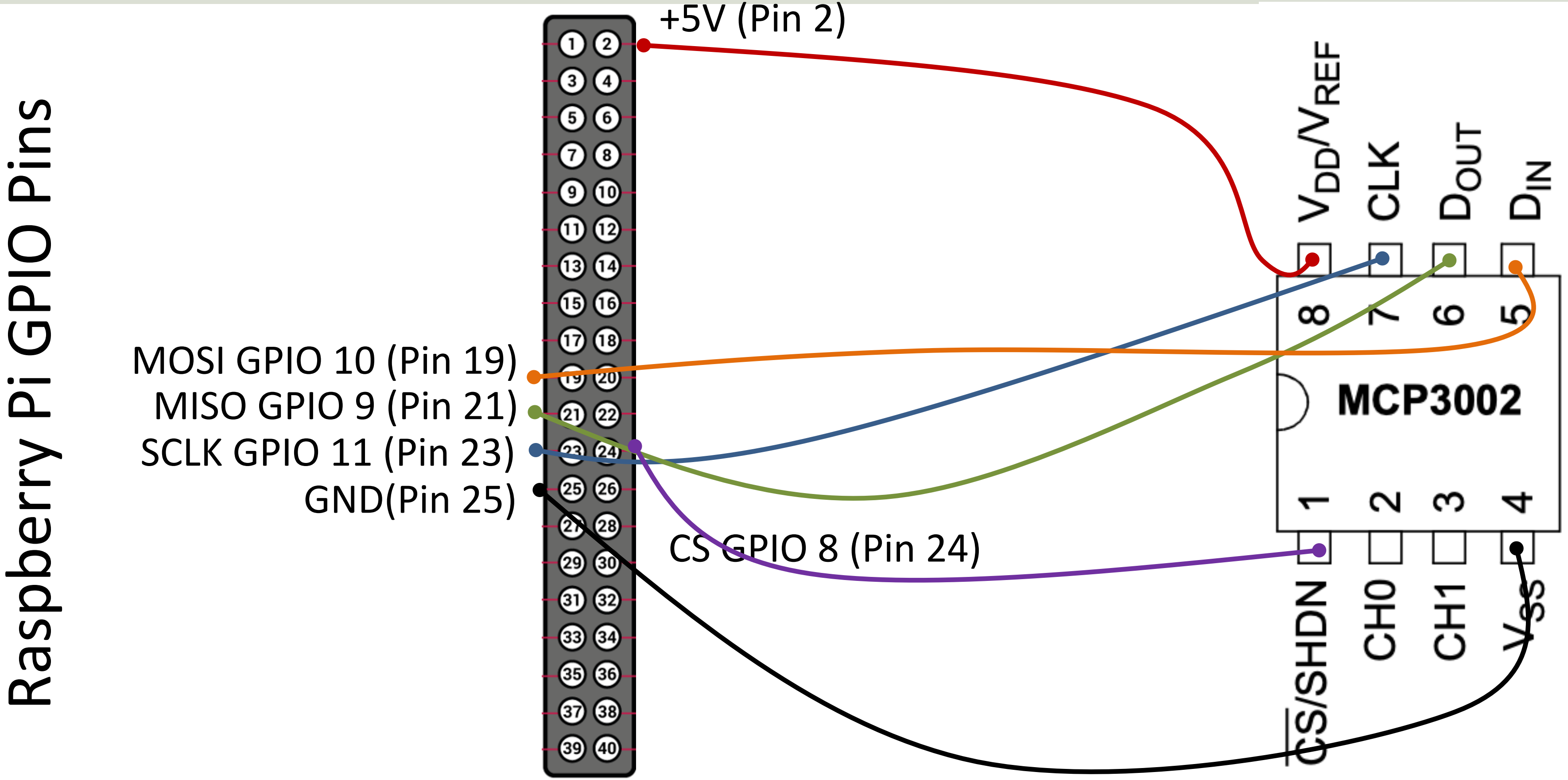
Wiring





https://sites.google.com/a/joekamphaus.net/raspberry-pi-spi-interface-to-mcp3002/

Wiring



Raspberry Pi GPIO Pins
+5V (Pin 2)
MOSI GPIO 10 (Pin 19)
MISO GPIO 9 (Pin 21)
SCLK GPIO 11 (Pin 23)
GND(Pin 25)
CS GPIO 8 (Pin 24)
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
V_{DD}/V_{REF}
CLK
D_{OUT}
D_{IN}
8
7
6
5
MCP3002
1
2
3
4
$\overline{CS}/SHDN$
CH0
CH1
V_{SS}

GPIO Zero and MCP3002

```
gpiozero.MCP3002(channel=0, differential=False, max_voltage=3.3, **spi_args)
```

channel
The channel to read data from. The MCP3008/3208/3304 have 8 channels (0-7), while the MCP3004/3204/3302 have 4 channels (0-3), the MCP3002/3202 have 2 channels (0-1), and the MCP3001/3201/3301 only have 1 channel.

differential
If True, the device is operated in differential mode. In this mode one channel (specified by the channel attribute) is read relative to the value of a second channel (implied by the chip's design).

Please refer to the device data-sheet to determine which channel is used as the relative base value (for example, when using an MCP3008 in differential mode, channel 0 is read relative to channel 1).

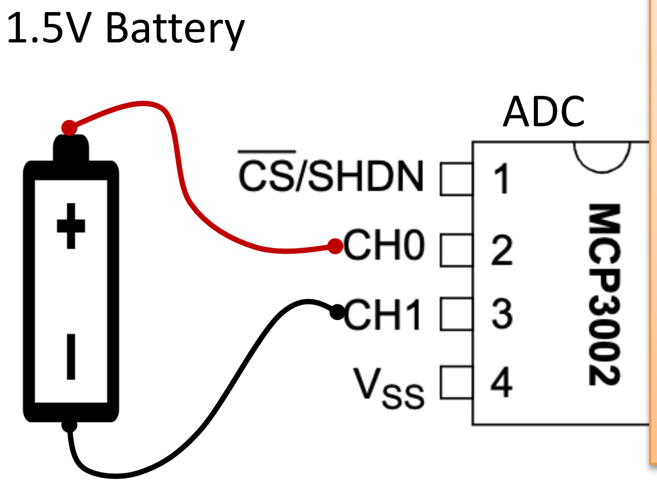
value
The current value read from the device, scaled to a value between 0 and 1 (or -1 to +1 for certain devices operating in differential mode).

https://gpiozero.readthedocs.io/en/stable/api_spi.html

Read Data from ADC

For test purpose we start by wiring a 1.5V Battery to the CH0 (+) and CH1(-) pins on the ADC

Note! WE have set **differential=True** (meaning CH0 is “+“ and CH1 is “-“)



```
from gpiozero import MCP3002
from time import sleep

adc = MCP3002(channel=0, differential=True)

N = 20

for x in range(N):
    adcdata = adc.value #Value between 0 and 1
    #print(adcdata)
    voltvalue = adcdata * 5 #Value between 0 and 5V
    print(voltvalue)
    sleep(1)
```

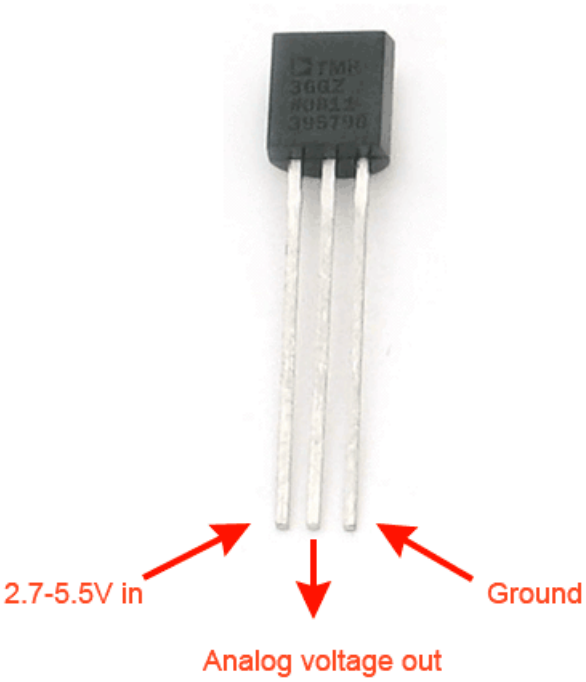
https://www.halvorsen.blog

TMP36

Temperature Sensor

Hans-Petter Halvorsen

TMP36 Temperature Sensor

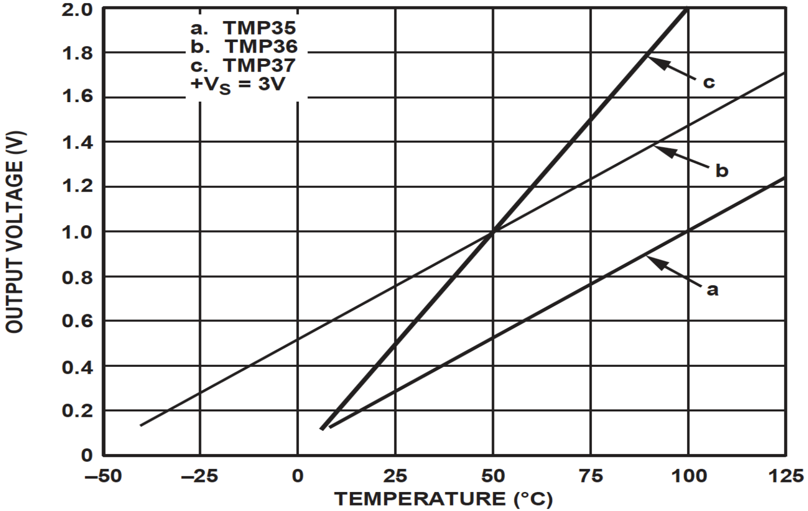


A Temperature sensor like TM36 use a solid-state technique to determine the temperature.

They use the fact as temperature increases, the voltage across a diode increases at a known rate.

https://learn.adafruit.com/tmp36-temperature-sensor

TMP36 Temperature Sensor



Convert form Voltage (V) to degrees Celsius

From the Datasheet we have:

$$(x_1, y_1) = (0.75V, 25°C)$$
$$(x_2, y_2) = (1V, 50°C)$$

There is a linear relationship between Voltage and degrees Celsius:

$$y = ax + b$$

We can find a and b using the following known formula:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

This gives:

$$y - 25 = \frac{50 - 25}{1 - 0.75}(x - 0.75)$$

Then we get the following formula:

$$y = 100x - 50$$

Measure Temperature with an ADC

TMP36 Temperature Sensor

Wire a TMP36 temperature sensor to the first channel of an MCP3002 analog to digital converter and the other pins to +5V and GND

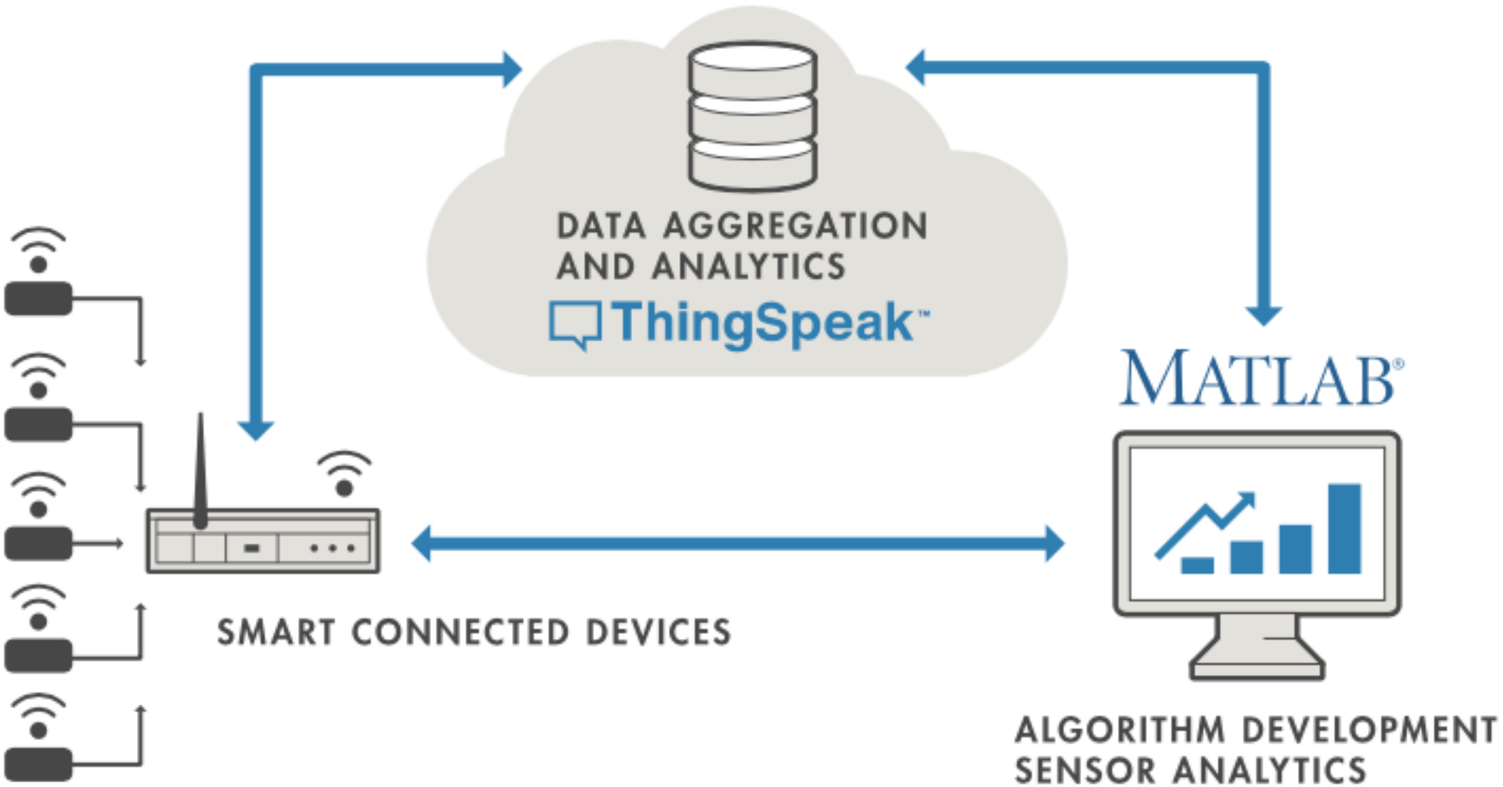
```
from gpiozero import MCP3002
from time import sleep

adc = MCP3002(channel=0, differential=False)

N = 10

for x in range(N):
    adcdata = adc.value #Value between 0 and 1
    #print(adcdata)

    voltvalue = adcdata * 5 #Value between 0V and 5V
    #print(voltvalue)

    tempC = 100*voltvalue-50 #Temperature in Celsius
    tempc = round(tempC,1)
    print(tempC)

    sleep(1)
```

ThingSpeak

ThingSpeak

- ThingSpeak is an IoT analytics platform service that lets you collect and store sensor data in the cloud and develop Internet of Things applications.
- The ThingSpeak service also lets you perform online analysis and act on your data. Sensor data can be sent to ThingSpeak from any hardware that can communicate using a REST API
- ThingSpeak has a Web Service (REST API) that lets you collect and store sensor data in the cloud and develop Internet of Things applications (it also has MQTT API).
- https://thingspeak.com
- Python Library for ThingSpeak: https://pypi.org/project/thingspeak/

ThingSpeak



DATA AGGREGATION AND ANALYTICS

ThingSpeak™

MATLAB®

SMART CONNECTED DEVICES

ALGORITHM DEVELOPMENT
SENSOR ANALYTICS

ThingSpeak Write

```
import thingspeak
import time

channel_id = xxxxxx
write_key  = "xxxxxxxxxxxxxxxxx"

channel = thingspeak.Channel(id=channel_id, api_key=write_key)

N = 10
for x in range(N):
      temperature = 24
      response = channel.update({'field1': temperature})
      time.sleep(15)
```

https://thingspeak.readthedocs.io/en/latest/api.html

A Free ThingSpeak Channel can only be updated every 15 sec

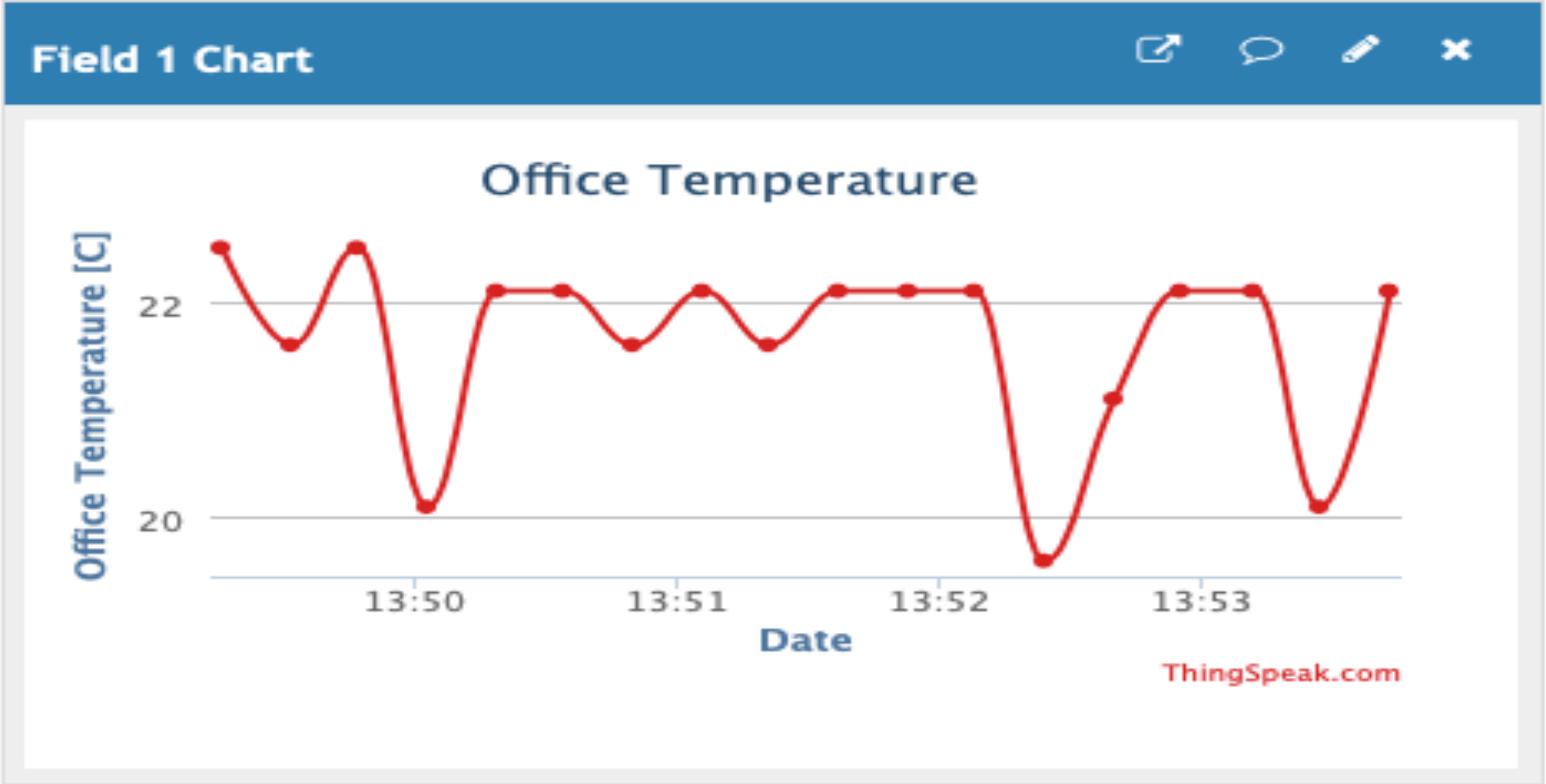
Write TMP36 Data

A Free ThingSpeak Channel can only be updated every 15 sec

```
import thingspeak
import time
from gpiozero import MCP3002

adc = MCP3002(channel=0, differential=False)

channel_id = xxxxxxx
write_key  = ”xxxxxxxxxxxxxxxxxx”

channel = thingspeak.Channel(id=channel_id, api_key=write_key)

N = 10
for x in range(N):
    #Get Sensor Data
    adcdata = adc.value #Scaled Value between 0 and 1
    voltvalue = adcdata * 5 # Value between 0V and 5V
    tempC = 100*voltvalue-50 # Temperature in Celsius
    tempC = round(tempC,1)
    print(tempC)

    #Write to ThingSpeak
    response = channel.update({'field1': tempC})
    time.sleep(15)
```

Write TMP36 Data

Here we see the Temperature Data in ThingSpeak:

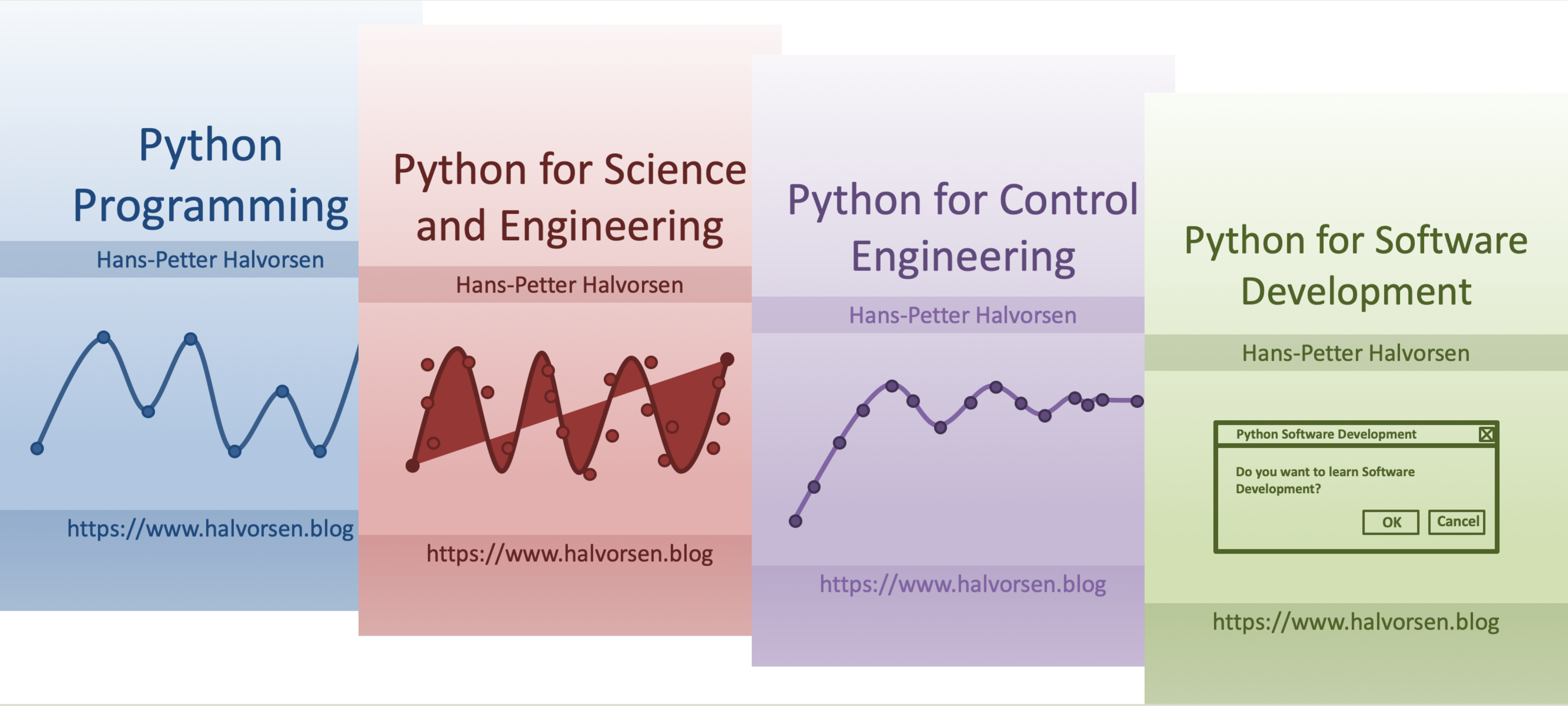


ThingSpeak Read

```
import thingspeak

channel_id = xxxxxx
read_key  = ”xxxxxxxxxxxxxxxx"

channel = thingspeak.Channel(id=channel_id, api_key=read_key)

#data = channel.get({})
data = channel.get_field({”field1"})

print(data)
```

https://thingspeak.readthedocs.io/en/latest/api.html

Additional Python Resources



Python Programming

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Science and Engineering

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Control Engineering

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Software Development

Hans-Petter Halvorsen

Python Software Development ☒

Do you want to learn Software Development?

OK Cancel

https://www.halvorsen.blog

https://www.halvorsen.blog/documents/programming/python/

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog