RASPBERRY PI 4B DEVELOPMENT KIT

www.abra-electronics.com

TEL: 1-800-361-5237

PI4B-DEV-KIT

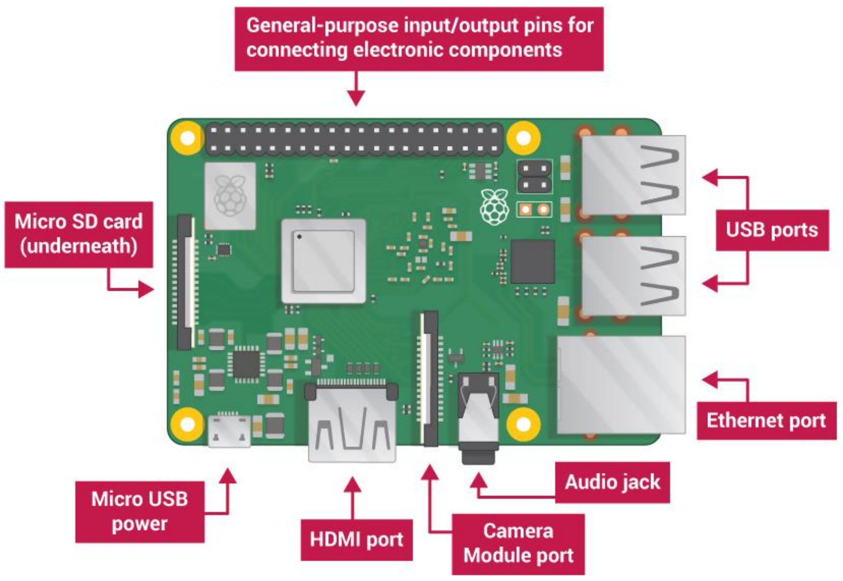
S.no	Name of the Component	Part number	Quantity
1	Servo Motor SG90	SG90	1
2	16x2 LCD Module with I2C	LCD-MOD-13	1
3	Humidity Sensor Module	SENS-DHT11-BB	1
4	Power Supply Module	PSM-297	1
5	Stepper Motor with Driver	MOT-28BYJ48	1
6	Active Buzzer	BUZ-120	1
7	Breadboard	ABRA-12-LC	1
8	1 Digit RED 7- Segment Display	S-5101AS	1
9	Tilt Sensor Module	SENS-39	1
10	Ultrasonic Sensor Module	HC-SR04	1
11	4x3 Keypad	419-ADA	1
12	PIR Sensor	SENS-PIR	1
13	Sound Sensor Module	SENS-42	1
14	Small Button Switch	PBS-315BK	5
15	Photoresistor	PHOTO-300	2
16	NPN Transistor	S8050	5
17	Potentiometer	3386-P-1-103T	1
18	Resistors (10,100,220,330,1K,2K,5K,10K,100K,1M)		100
19	10K Thermistor	334-103	1
20	5mm LED (Red, Green, Yellow, Blue, White)		25
21	RGB LED	LED-5RGB-4-CC	1
22	Motor Driver	L293D	1
23	Shift Register	74HC595	1
24	Analog to Digital Converter IC- ADC0832	ADC0832	1
25	Capacitor (10uf 50V)	10R50	2
26	Capacitor (100uf 50V)	100R50	2
27	Ceramic Capacitor(22pf)	CD220	5
28	Ceramic Capacitor (104)	CD104	5
29	DC Motor FAN	MOT-PROP-L	1
30	DC Motor	MOT-500	1
31	LED Bar Graph	MV57164	1
32	PI-cobbler	PI-COBBLER	1
33	PI- cobbler-C	PI-COBBLER-C	1
34	Female to Male	JW-MF-20-6	1
35	Male to Male	JW-MM-20-6	1
36	Plastic Case	CB-13	1
37	Resistor card		1
38	Manual		1

Contents

1 Introduction to Raspberry PI 3
2 Set up 3
3 Installation 3
4 Programming 5
5 Getting started 6
6 Projects 7
6.1 Blinking LED 7
6.2 Controlling brightness of LED 7
6.3 Door buzzer using push button 8
6.4 Tilt Switch (SENS 39) 9
6.5 Photoresistor (Photo-300) 10
6.6 RGB Led 11
6.7 LED bar graph 12
6.8 Thermistor (334-103) 13
6.9 Shift register(74HC595) 16
6.10 7-segment display (S-5101AS) 17
6.11 I2C LCD (LCD MOD-13) 18
6.12 Motor Driver (LD93D) 20
6.13 Distance Sensor (HC-SR04) 20
6.14 4x3 Keypad (419-ADA) 21
6.15 PIR sensor (SENS-PIR) 23
6.16 Humidity sensor (SENS-DHT11-BB) 23
6.17 Servo (SG-90) 24
6.18 Stepper motor with driver (MOT-28BYJ-48) 25

1 Introduction to Raspberry PI

The Raspberry Pi is a tiny computer about the size of a deck of cards. It uses what's called a system on a chip, which integrates the CPU and GPU in a single integrated circuit, with the RAM, USB ports, and other components soldered onto the board for an all-in-one package. We should use the SD card to install Operating system in raspberry pie.


General-purpose input/output pins for connecting electronic components
Micro SD card (underneath)
USB ports
Ethernet port
Audio jack
Micro USB power
HDMI port
Camera Module port

- USB ports — these are used to connect a mouse and keyboard. You can also connect other components, such as a USB drive.
- SD card slot — you can slot the SD card in here. This is where the operating system software and your files are stored.
- Ethernet port — this is used to connect Raspberry Pi to a network with a cable. Raspberry Pi can also connect to a network via wireless LAN.
- Audio jack — you can connect headphones or speakers here.
- HDMI port — this is where you connect the monitor (or projector) that you are using to display the output from the Raspberry Pi. If your monitor has speakers, you can also use them to hear sound.
- Micro USB power connector — this is where you connect a power supply. You should always do this last, after you have connected all your other components.
- GPIO ports — these allow you to connect electronic components such as LEDs and buttons to Raspberry Pi.

2 Set up

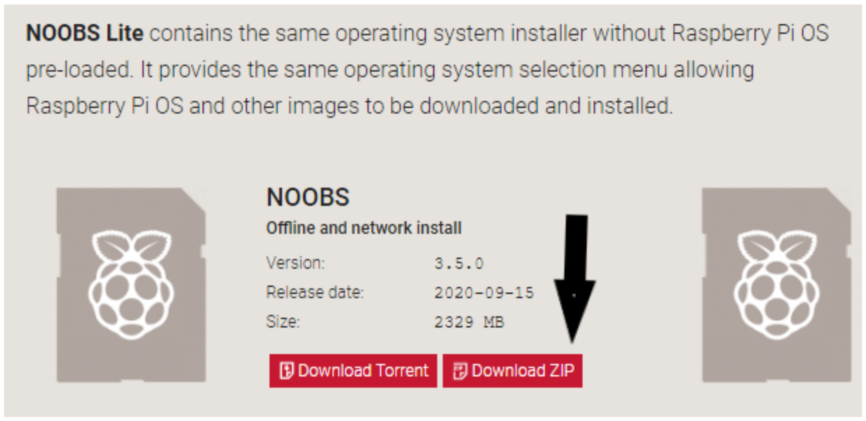
- SD card: We recommend a minimum of 8GB class 4.
- Display and connectivity cable: Any HDMI/DVI monitor or TV should work as a display for the Pi. For best results, use a display with HDMI input; other types of connection for older devices are also available.
- Keyboard and mouse: Any standard USB keyboard and mouse will work with your Raspberry Pi. Wireless keyboards and mice will work if already paired.
- Power supply: The Pi is powered by a USB Micro [models pre 4B] or USB Type-C [model 4B] power supply (like most standard mobile phone chargers). You need a good-quality power supply that can supply at least 3A at 5V for the Model 4B, 2A at 5V for the Model 3B and 3B+, or 700mA at 5V for the earlier, lower-powered Pi models. Low-current (~700mA) power supplies will work for basic usage but are likely to cause the Pi to reboot if it draws too much power. They are not suitable for use with the Pi 3 or 4.

Note: To save time, you can get a card that is pre-installed with NOOBS or Raspberry Pi OS, although setting up your own card is easy.

3 Installation

Beginners should start with NOOBS, which gives the user a choice of operating system from the standard distributions.

You can download NOOBS from https://www.raspberrypi.org/downloads/noobs/



NOOBS Lite contains the same operating system installer without Raspberry Pi OS pre-loaded. It provides the same operating system selection menu allowing Raspberry Pi OS and other images to be downloaded and installed.

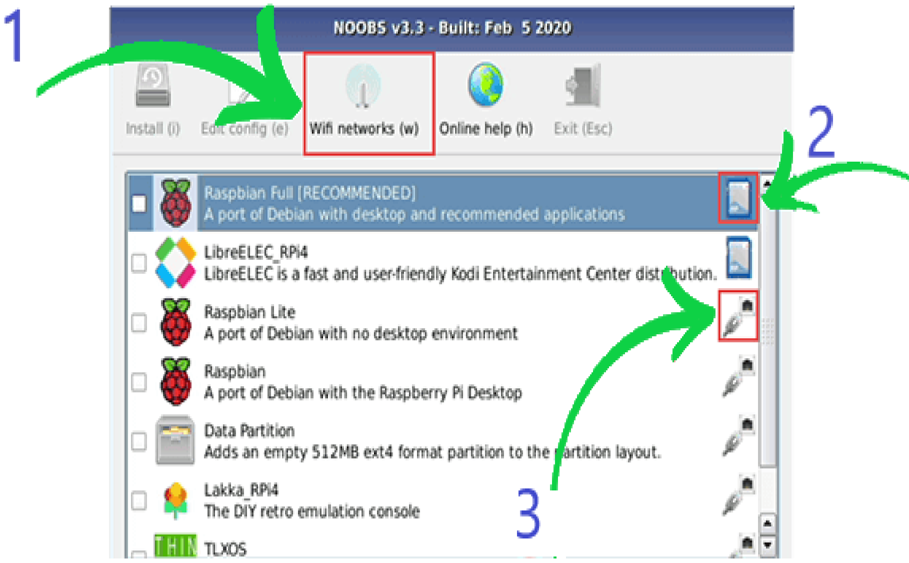
NOOBS
Offline and network install

Version:	3.5.0
Release date:	2020-09-15
Size:	2329 MB

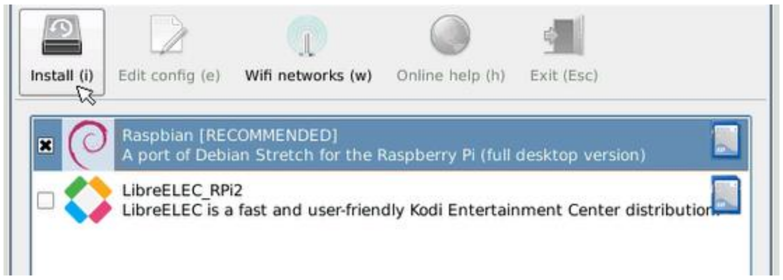
1. Download the zip file.
2. Unzip the file.
3. Insert SD card to your computer.
4. Make sure you format the SD card using SD card formatter software. Format it even if it's a new card.
5. Copy all the contents of the unzipped file to the SD card.
6. Connect the mouse and keyboard to the USB slots on raspberry pi.
7. Remove the SD card once files are copied and now insert the SD card to the raspberry pi.
8. Power up the raspberry pi.
9. Connect the HDMI cable to HDMI0 in raspberry pi and then connect to the HDMI port on the monitor.
10. Upon your Raspberry Pi starting up, if you want to access additional operating systems, you can connect to a **Wi-Fi network** by clicking the **"Wi-Fi Networks"** button (1).

    Or if you want to download an operating system over the network (internet), it will show an ethernet cable icon (3). You can select whatever operating system that is present over network.

    But here in this manual we have followed the SD card icon as shown by 2 which represents OS available locally i.e. on SD card. To install the operating system on SD card, select **Raspbian** and click on **install** as shown in below picture**.** A confirm window pops up click on **Yes.**

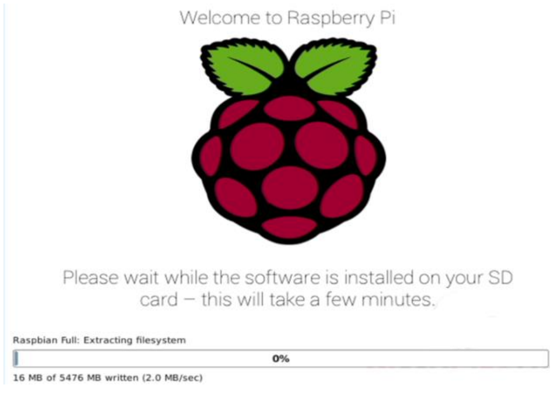


Raspbian Full [RECOMMENDED]
A port of Debian with desktop and recommended applications
LibreELEC_RPi4
LibreELEC is a fast and user-friendly Kodi Entertainment Center distribution.
Raspbian Lite
A port of Debian with no desktop environment
Raspbian
A port of Debian with the Raspberry Pi Desktop
Data Partition
Adds an empty 512MB ext4 format partition to the partition layout.
Lakka_RPi4
The DIY retro emulation console
TLXOS



Raspbian [RECOMMENDED]
A port of Debian Stretch for the Raspberry Pi (full desktop version)
LibreELEC_RPi2
LibreELEC is a fast and user-friendly Kodi Entertainment Center distribution

11. The installation is time consuming.

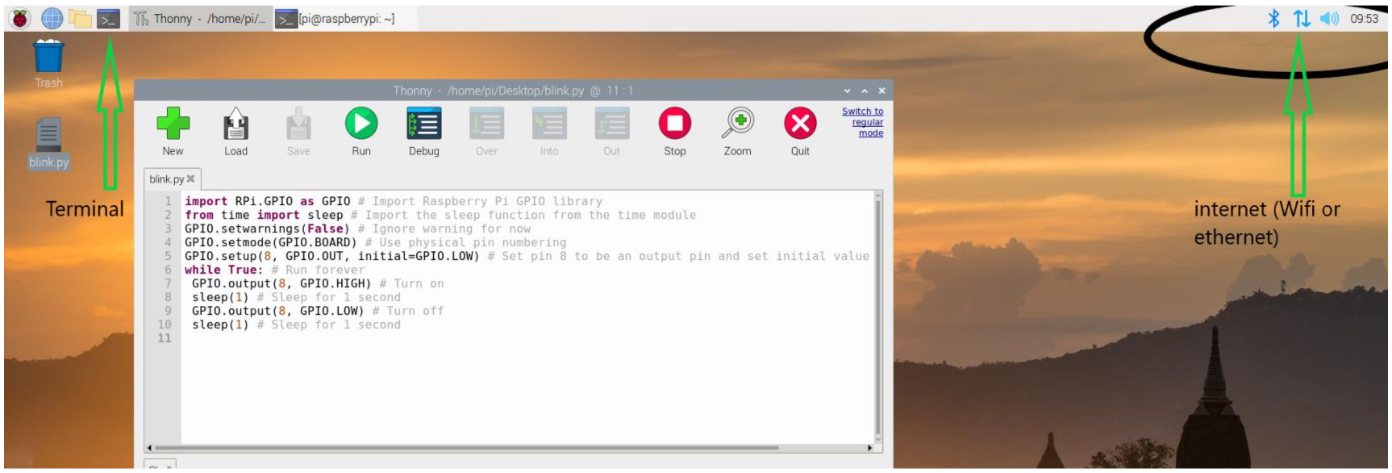


Welcome to Raspberry Pi

Please wait while the software is installed on your SD card – this will take a few minutes.

Raspbian Full: Extracting filesystem
0%
16 MB of 5476 MB written (2.0 MB/sec)

12. Enter the location details.
13. Set up your password.
14. Select the Wi-Fi Network. If you don't have ignore it. But make sure your raspberry is connected to either Ethernet or Wi-Fi so that we can install certain libraries that we do not have.



```python
import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library
from time import sleep # Import the sleep function from the time module
GPIO.setwarnings(False) # Ignore warning for now
GPIO.setmode(GPIO.BOARD) # Use physical pin numbering
GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW) # Set pin 8 to be an output pin and set initial value
while True: # Run forever
 GPIO.output(8, GPIO.HIGH) # Turn on
 sleep(1) # Sleep for 1 second
 GPIO.output(8, GPIO.LOW) # Turn off
 sleep(1) # Sleep for 1 second
```

4 Programming

You can install python3 if it is not in the OS. Use Commands as given below on terminal

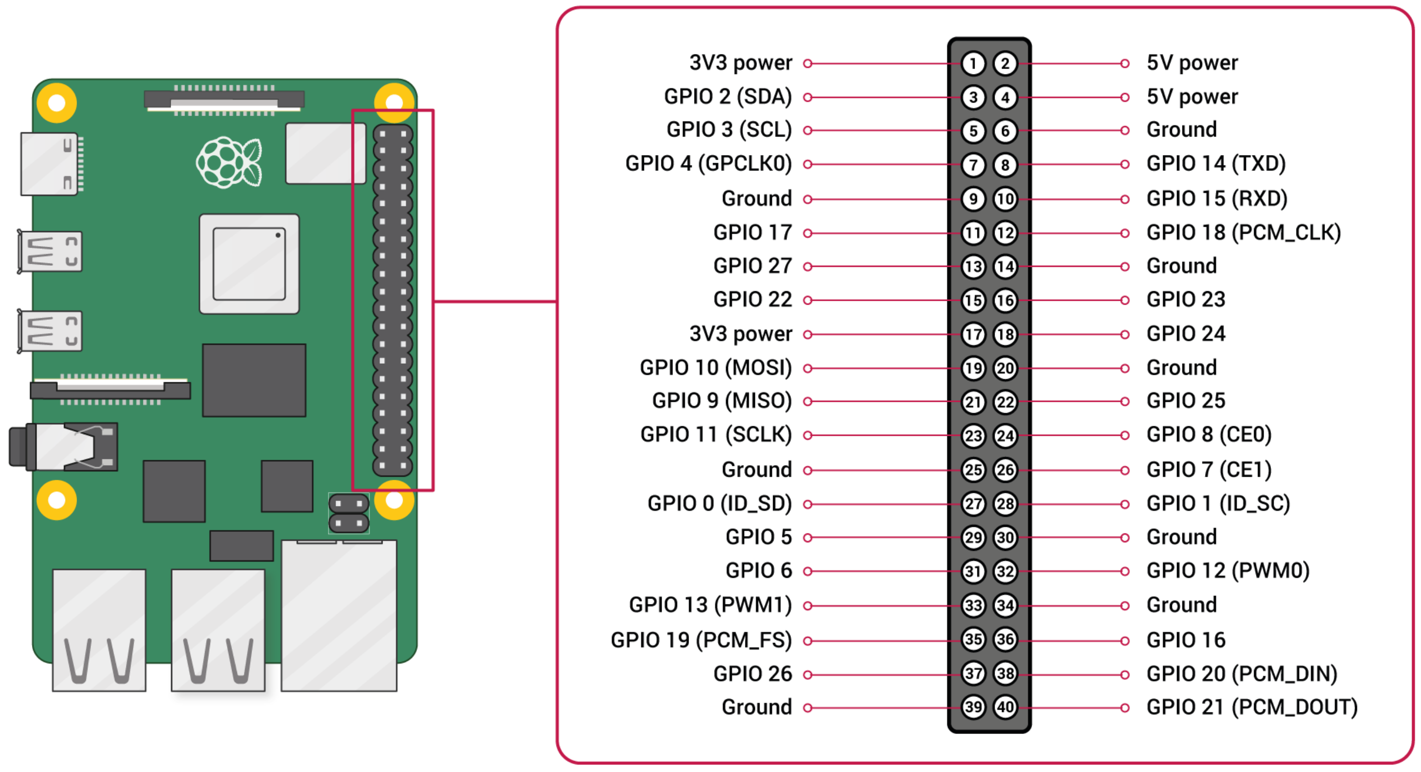
```
sudo apt-get update
sudo apt-get install python3
```

To install the GPIO library type the following commands on terminal

```
sudo apt-get update
sudo apt-get install rpi.gpio
```

We can use **Thonny** if we do not have python 3. **Thonny**: The easiest introduction to Python is through Thonny, a Python3 development environment. Here we use Thonny. Open Thonny, type your code and save your code as **.py file.** Make the circuit connection for raspberry pi and then **Run.**

5 Getting started



Pin Numbering Declaration

After you've included the RPi.GPIO module, the next step is to determine which one of the two **pin-numbering schemes** you want to use:
GPIO.BOARD -- This type of pin numbering refers to the number of the pin in the plug, i.e, the numbers printed on the board, for example, ❶,❷,❸,❹ that can be seen in the pinout picture above. The advantage of this type of numbering is, it will not change even though the version of board changes.
GPIO.BCM -- Broadcom chip-specific pin numbers. These pin numbers follow the lower-level numbering system defined by the Raspberry Pi's Broadcom-chip brain. For example GPIO 2(SDA) is refereed to as PIN number2 and not 3 as in GPIO.BOARD.

Setting a Pin Mode

If you've used Arduino, you're probably familiar with the fact that you have to declare a "pin mode" before you can use it as either an input or output. To set a pin mode, use the **setup(pin, GPIO.IN** or **GPIO.OUT)** function. So, if you want to set pin 22 as an output, for example, write:

```
GPIO.setup(22, GPIO.OUT)
```

Outputs
Digital Output
To write a pin high or low, use the **GPIO.output(pin, GPIO.LOW** or **GPIO.HIGH)** function. For example, if you want to set pin 22 low, write

```
GPIO.output(22, GPIO.LOW)
```

Delays

To have delay in your code first make sure you have included time library.

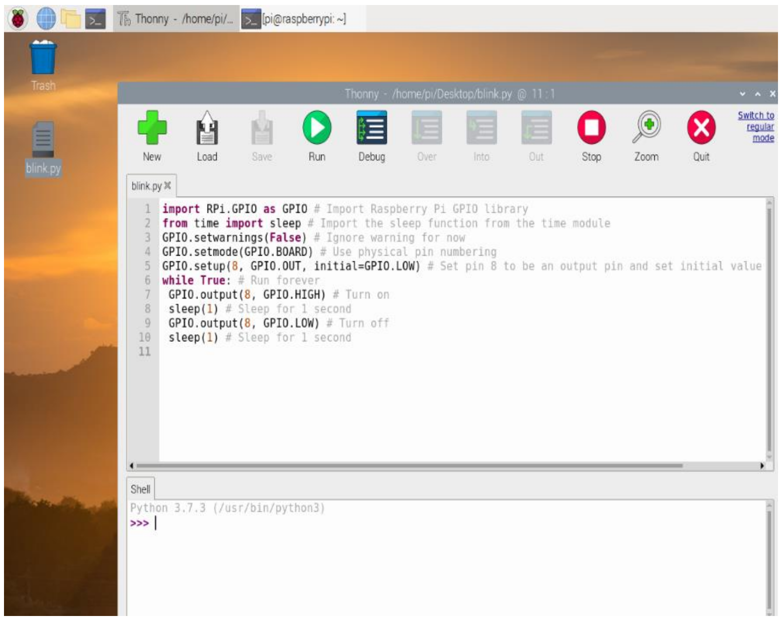
include time

time.sleep(seconds) is used to set delays.

time.sleep(0.5) # delay for 500 milliseconds.

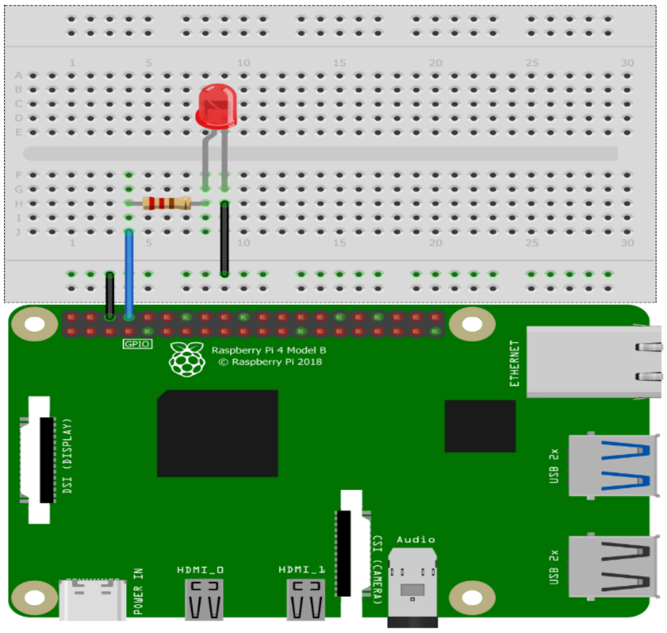
6 Projects

6.1 Blinking LED



```
import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library
from time import sleep # Import the sleep function from the time module
GPIO.setwarnings(False) # Ignore warning for now
GPIO.setmode(GPIO.BOARD) # Use physical pin numbering
GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW) # Set pin 8 to be an output pin and set initial value
while True: # Run forever
 GPIO.output(8, GPIO.HIGH) # Turn on
 sleep(1) # Sleep for 1 second
 GPIO.output(8, GPIO.LOW) # Turn off
 sleep(1) # Sleep for 1 second
```

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



Write the code in **Thonny** and save the file with **.py** extension. Connect the circuit and **Run** the code.

```
import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library
from time import sleep # Import the sleep function from the time module
GPIO.setmode(GPIO.BOARD) # Use physical pin numbering
GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW) # Pin 8 is set as output pin and initial value
set to low
while True: # Run forever
GPIO.output(8, GPIO.HIGH) # Turn on LED
sleep(2) # Sleep for 2 seconds
GPIO.output(8, GPIO.LOW) # Turn off
sleep(2) # Sleep for 2 seconds
```

Always copy paste the code as it is do not make any changes to tab space in code lines

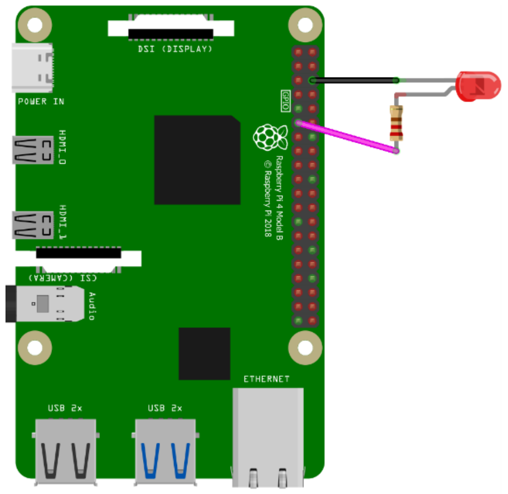
6.2 Controlling brightness of LED

In this project we shall control the brightness of the LED using the PWMLED library.

Install gpiozero library.

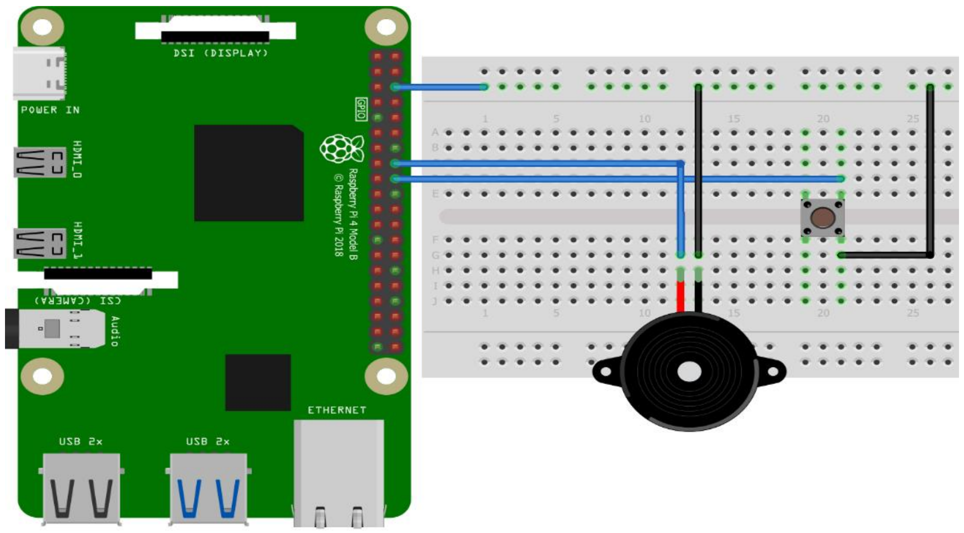
```
sudo apt-get update
sudo apt-get install gpiozer0
```

```
from gpiozero import PWMLED
from time import sleep
LED=PWMLED(17)
while True:
  LED.value=0
  sleep(1)
  LED.value=0.3 #increase to one third of full brightness
  sleep(1)
  LED.value=0.6
  sleep(1)
  LED.value=1# full brightness
  sleep(1)
 # You can also toggle the LED by using LED.toggle()
```



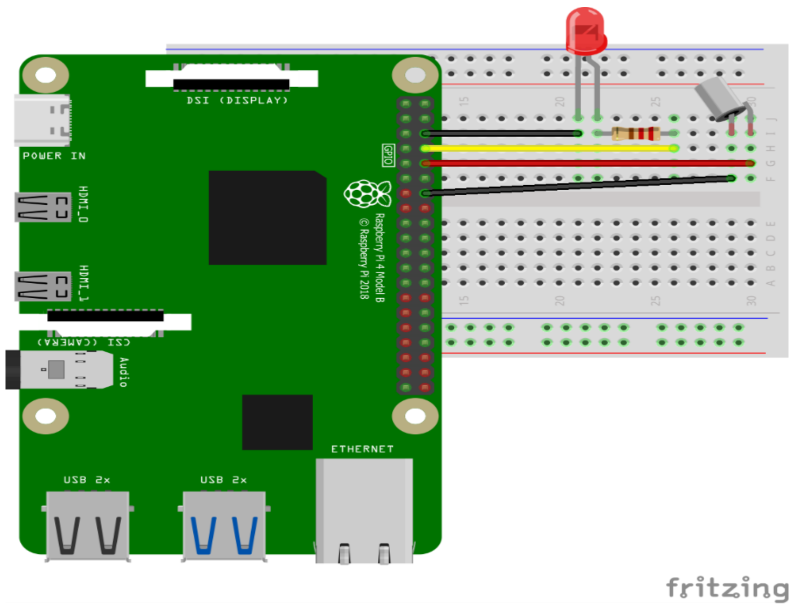
6.3 Door buzzer using push button

An **active buzzer** will generate a tone using an internal oscillator, so all that is needed is a DC voltage which we now provide through one of the gpio pins on Raspberry Pi. Components: BUZ-120, Push button, wires



```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time
buzzer_pin  = 16
switch = 18

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(buzzer_pin, GPIO.OUT)
    GPIO.output(buzzer_pin, GPIO.LOW) # Set buzzer_pin in off state
    GPIO.setup(switch, GPIO.IN,pull_up_down=GPIO.PUD_UP) # Set  pin 18 as input, and pull
up to high level(3.3V)

def loop():
    while True:
        if GPIO.input(switch) == GPIO.LOW:
            print ("buzzer_pin  on")
            GPIO.output(buzzer_pin, GPIO.HIGH)  # buzzer on
        else:
            print ("buzzer_pin  off")

            GPIO.output(buzzer_pin, GPIO.LOW) # buzzer off

def destroy():
    GPIO.output(buzzer_pin, GPIO.LOW)     # buzzer off
    GPIO.cleanup()                        # Release resource
if __name__ == '__main__':      # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy() will
be  executed.
        destroy()
```

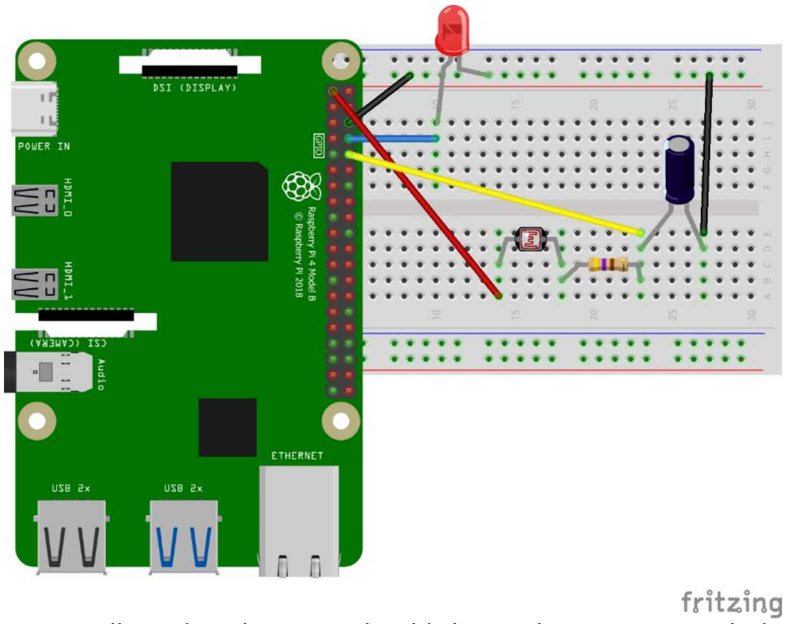
6.4 Tilt Switch (SENS 39)



Tilt switch is a switch which opens and closes an electrical circuit when it is tilted at certain angles. After connecting the circuit if the breadboard is tilted to certain angle the LED will glow. They are small, inexpensive, low-power, and easy-to-use.

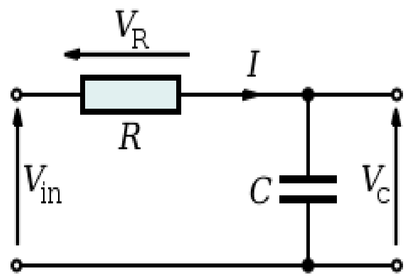
```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time
led_pin  = 8
tilt_switch = 10

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(led_pin, GPIO.OUT)
    GPIO.output(led_pin, GPIO.LOW) # Set led_pin in off state
    GPIO.setup(tilt_switch, GPIO.IN,pull_up_down=GPIO.PUD_UP) # Set  pin 10 as input, and
pull up to high level(3.3V)

def loop():
    while True:
        if GPIO.input(tilt_switch) == GPIO.LOW:
            print ("LED switched on")
            GPIO.output(led_pin, GPIO.HIGH)  # led switched on
        else:
            print ("LED switched off")

            GPIO.output(led_pin, GPIO.LOW) # led switched off

def destroy():
    GPIO.output(led_pin, GPIO.LOW)     # led off
    GPIO.cleanup()                     # Release resource
if __name__ == '__main__':      # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy() will
be  executed.
        destroy()
```

6.5 Photoresistor (Photo-300)



Unlike some other devices the Raspberry Pi does not have any analogue inputs. All of its GPIO pins are digital. They can output high and low levels or read high and low levels. This is great for sensors that provide a digital input to the Pi but not so great if you want to use analogue sensors. We use a basic "RC" charging circuit in which Resistor is in series with a Capacitor. When a voltage is applied across these components the voltage across the capacitor rises. The time it takes for the voltage to reach 63% of the maximum is equal to the resistance multiplied by the capacitance. When using a Light Dependent resistor this time will be proportional to the light level. This time is called the time constant i.e. t = RC where t is time, R is resistance and C is capacitance

Since all are digital pins we should charge the capacitor such that it reaches the voltage great enough to be recognized as logic HIGH by one of the digital pins.



•Set the GPIO pin as an output and set it Low to discharge the capacitor to 0V.

•Set the GPIO pin as an input. This starts a flow of current through the resistors and through the capacitor to ground. The voltage across the capacitor starts to rise. The time it takes is proportional to the resistance of the LDR.

•Monitor the GPIO pin and read its value. Increment a counter while we wait.

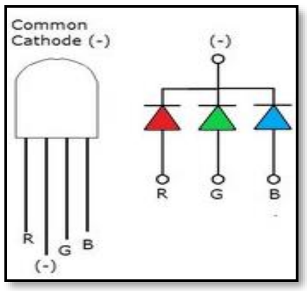
•At some point the capacitor voltage will increase enough to be considered as a High by the GPIO pin (approx 2v). The time taken is proportional to the light level seen by the LDR.

NOTE: in this project we have used GPIO PIN numbers and not the physical pins i.e. sensor pin is GPIO-15 which is nothing but physical pin number 10

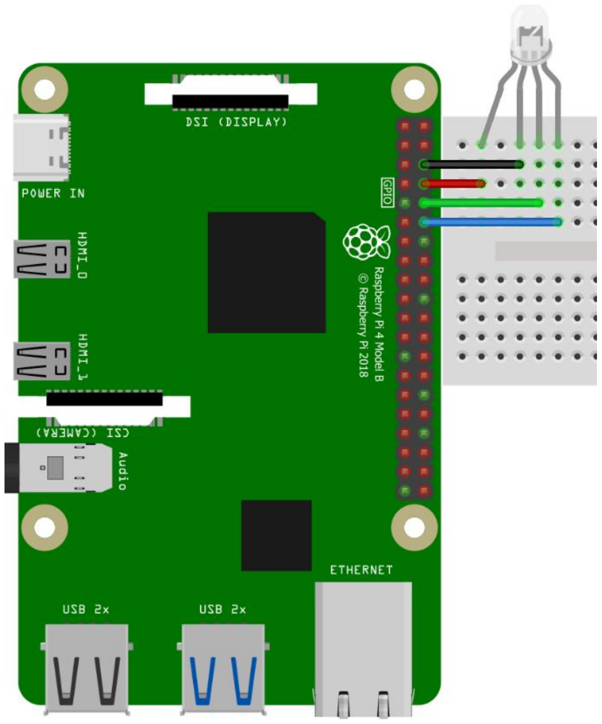
```
from gpiozero import LightSensor, LED

sensor = LightSensor(15)
led = LED(14)
def loop():
    while True:
        sensor.wait_for_light()
        print("there's light")
        sensor.wait_for_dark()
        print("It's dark")
        sensor.when_dark = led.on
        sensor.when_light = led.off
        pause()

if __name__ == "__main__":
    try:
        loop()
    except:
        print("led")
```

6.6 RGB Led

In this example we will learn how to vary the color of RGB. Varying the PWM values causes the change in color. Pulse Width Modulation (or PWM) is a technique for controlling power. We also use it here to control the brightness of each of the LEDs. Using PWM the amount of power delivered to a device can be controlled. Duty Cycle and Frequency concepts are used in PWM to control brightness of RGB LED. Duty cycle indicates the duration for which the pulse is HIGH over it period. In lay man terms this duty cycle is a value in percent of ON status compared to OFF status. From the figure below we can see the formula to calculate the duty cycle. It is measured in percentage and it indicates the voltage between OFF and ON levels (usually 0V and 5V).

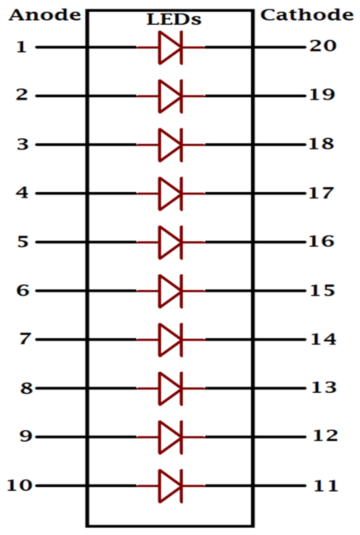


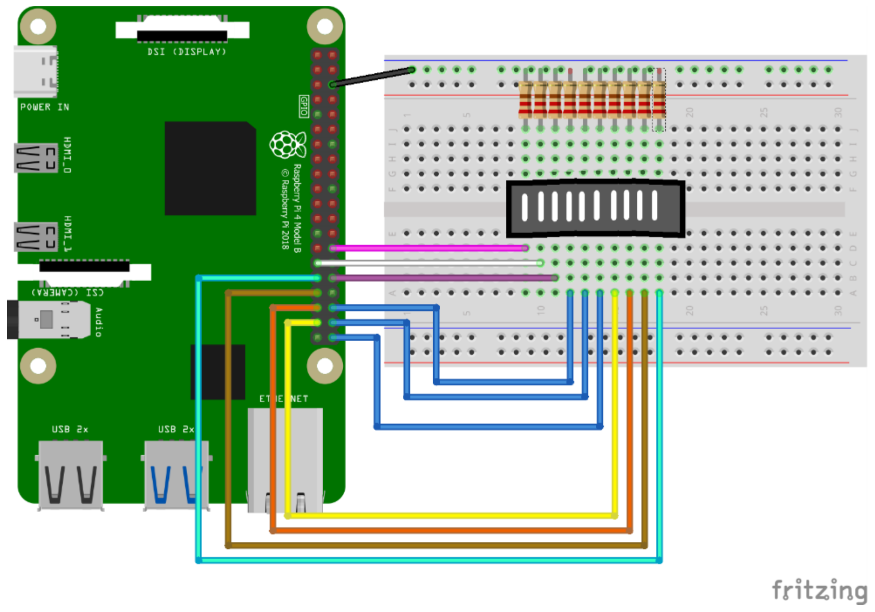
```
from gpiozero import RGBLED
from time import sleep

led = RGBLED(red=14, green=15, blue=18)
def loop():
    while True:
        led.red = 1   # full red
        sleep(1)
        led.red = 0.5   # half red
        sleep(1)
        led.color = (0, 1, 0)   # full green
        sleep(1)
        led.color = (1, 0, 1)   # magenta
        sleep(1)
        led.color = (1, 1, 0)   # yellow
        sleep(1)
        led.color = (0, 1, 1)   # cyan
        sleep(1)
        led.color = (1, 1, 1)   # white
        sleep(1)
        led.color = (0, 0, 0)   # off
        sleep(1)
if __name__ == "__main__":
    try:
        loop()
    except:
        print("RGB")
```



6.7 LED bar graph

LED Bar Graph is an LED array, which is used to connect with electronic circuit or microcontroller. It's easy to connect LED bar graph. One side of the LED bar graph consists of anode of LED and the other side cathode. The LED bar graph is used as a Battery level Indicator, Audio equipment and Industrial Control panels.





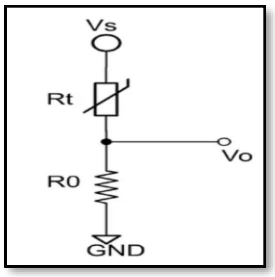
```python
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
leds = [1,5,12,16,20,21,26,19,13,6]

for led in leds:
  GPIO.setup(led, GPIO.OUT,initial=0)
def setup():
  while True:
    for led in leds:
      GPIO.output(led, 1) # switched ON
      time.sleep(0.5)# half a second delay
      GPIO.output(led, 0) # switched OFF
def destroy():
  GPIO.cleanup()

if __name__ == "__main__":
  try:
    setup()
  except KeyboardInterrupt:
    destroy()
```

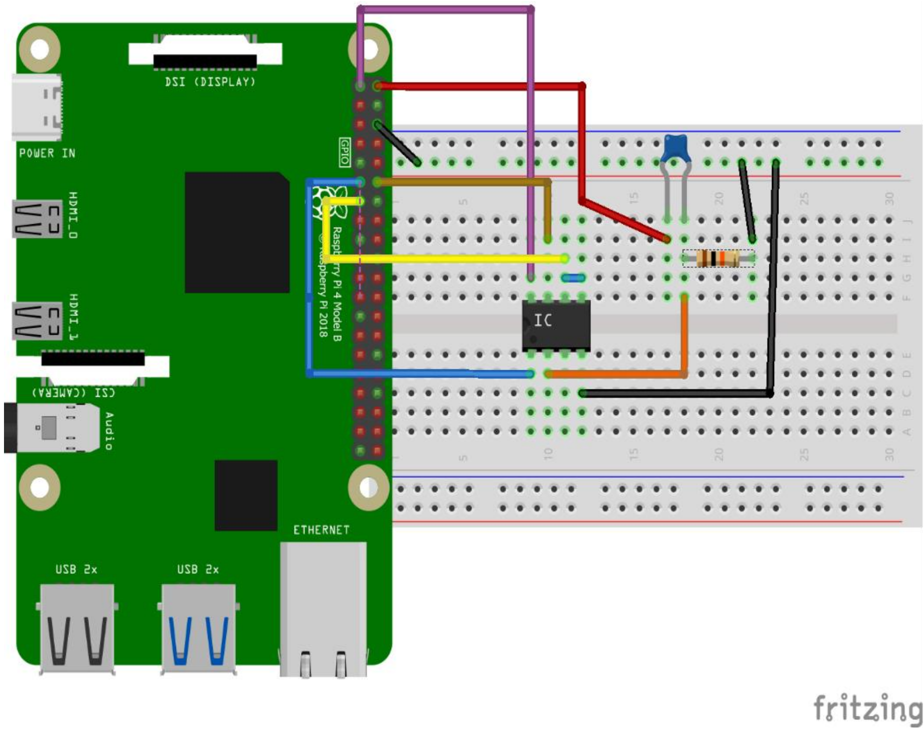
6.8 Thermistor (334-103)

A thermistor is a type of resistor whose resistance is dependent on temperature. Thermistors are of two opposite types:

- With **NTC** thermistors, resistance ***decreases*** as temperature rises. An NTC is commonly used as a temperature sensor.
- With **PTC** thermistors, resistance ***increases*** as temperature rises.

In this experiment we create a voltage divider between thermistor and 10kΩ resistor and perform the calculation.





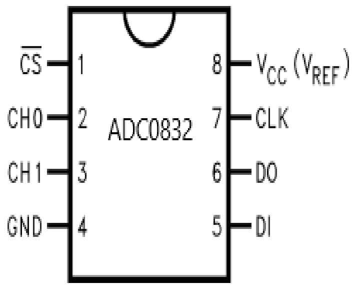
- **Vo = Vs * (R0 / (Rt + R0**
- **Rt = R0 * ((Vs / Vo) - 1)**
- **1/T = A + B*ln(R) + C*(ln(R))^3**

Components: 10K thermistor, 10kΩ resistor, ADC0832.

The following steps are followed for ADC IC

- bring CS pin low to signal beginning of conversion

•start pulsing clock
•output 1 on the digital i/o pin



- output a bit for the mux mode (single-ended or differential)
- output a bit for the channel; the interpretation of this depends on the selected mode.
- this concludes the setup phase; the digital i/o pin is now set to input mode
- on each clock cycle, read a bit from the digital pin; repeat for 8 periods and accumulate the value. The first time data is passed MSB first.
- then the adc passes the same data LSB first. Read in a similar manner to the previous part - pulse 8 clock cycles and read a bit from the device on each.

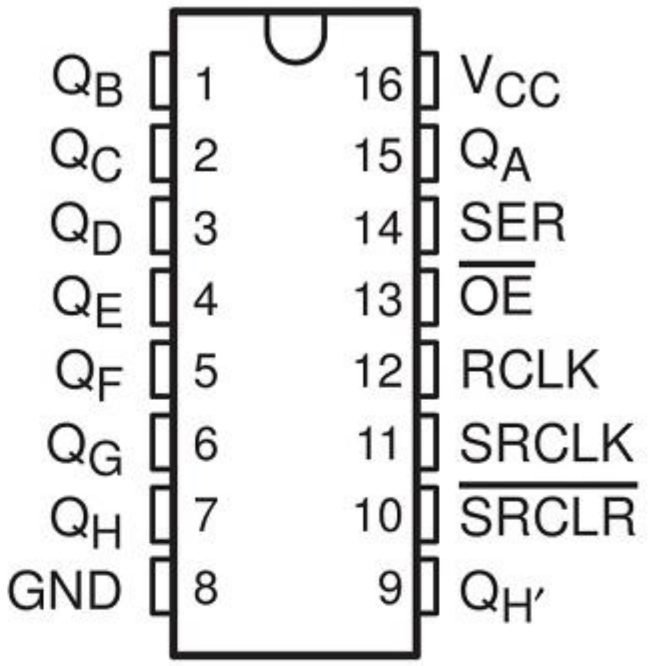
Make sure to you have the Program given below saved as ADC0832.py in the folder where you save your python files. This ADC0832 will be imported to read analog values from thermistor.

```
#       This is a program for all ADC Module. It will be imported when we should read analog
values.
#    convert analog signal to digital signal.
import RPi.GPIO as GPIO
import time

ADC_CS   = 11
ADC_CLK = 12
ADC_DIO = 13

# using default pins for backwards compatibility
def setup(cs=11,clk=12,dio=13):
    global ADC_CS, ADC_CLK, ADC_DIO
    ADC_CS=cs
    ADC_CLK=clk
    ADC_DIO=dio
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BOARD)            # Number GPIOs by its physical location
    GPIO.setup(ADC_CS, GPIO.OUT)        # Set pins' mode is output
    GPIO.setup(ADC_CLK, GPIO.OUT)       # Set pins' mode is output

def destroy():
    GPIO.cleanup()

# using channel = 0 as default for backwards compatibility
def getResult(channel=0):                    # Get ADC result, input channal
        GPIO.setup(ADC_DIO, GPIO.OUT)
        GPIO.output(ADC_CS, 0)

        GPIO.output(ADC_CLK, 0)
        GPIO.output(ADC_DIO, 1);  time.sleep(0.000002)
        GPIO.output(ADC_CLK, 1);  time.sleep(0.000002)
        GPIO.output(ADC_CLK, 0)

        GPIO.output(ADC_DIO, 1);  time.sleep(0.000002)
        GPIO.output(ADC_CLK, 1);  time.sleep(0.000002)
        GPIO.output(ADC_CLK, 0)

        GPIO.output(ADC_DIO, channel);  time.sleep(0.000002)

        GPIO.output(ADC_CLK, 1)
        GPIO.output(ADC_DIO, 1);  time.sleep(0.000002)
        GPIO.output(ADC_CLK, 0)
        GPIO.output(ADC_DIO, 1);  time.sleep(0.000002)
        dat1 = 0
        for i in range(0, 8):
            GPIO.output(ADC_CLK, 1);  time.sleep(0.000002)
            GPIO.output(ADC_CLK, 0);  time.sleep(0.000002)
            GPIO.setup(ADC_DIO, GPIO.IN)
            dat1 = dat1 << 1 | GPIO.input(ADC_DIO)

        dat2 = 0
        for i in range(0, 8):
            dat2 = dat2 | GPIO.input(ADC_DIO) << i
            GPIO.output(ADC_CLK, 1);  time.sleep(0.000002)
            GPIO.output(ADC_CLK, 0);  time.sleep(0.000002)

        GPIO.output(ADC_CS, 1)
        GPIO.setup(ADC_DIO, GPIO.OUT)

        if dat1 == dat2:
            return dat1
        else:
            return 0
def getResult1():
    return getResult(1)
def loop():
    while True:
        res0 = getResult(0)
        res1 = getResult(1)
        time.sleep(0.4)

if __name__ == '__main__':      # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

The program for calculating temperature using thermistor is given below.

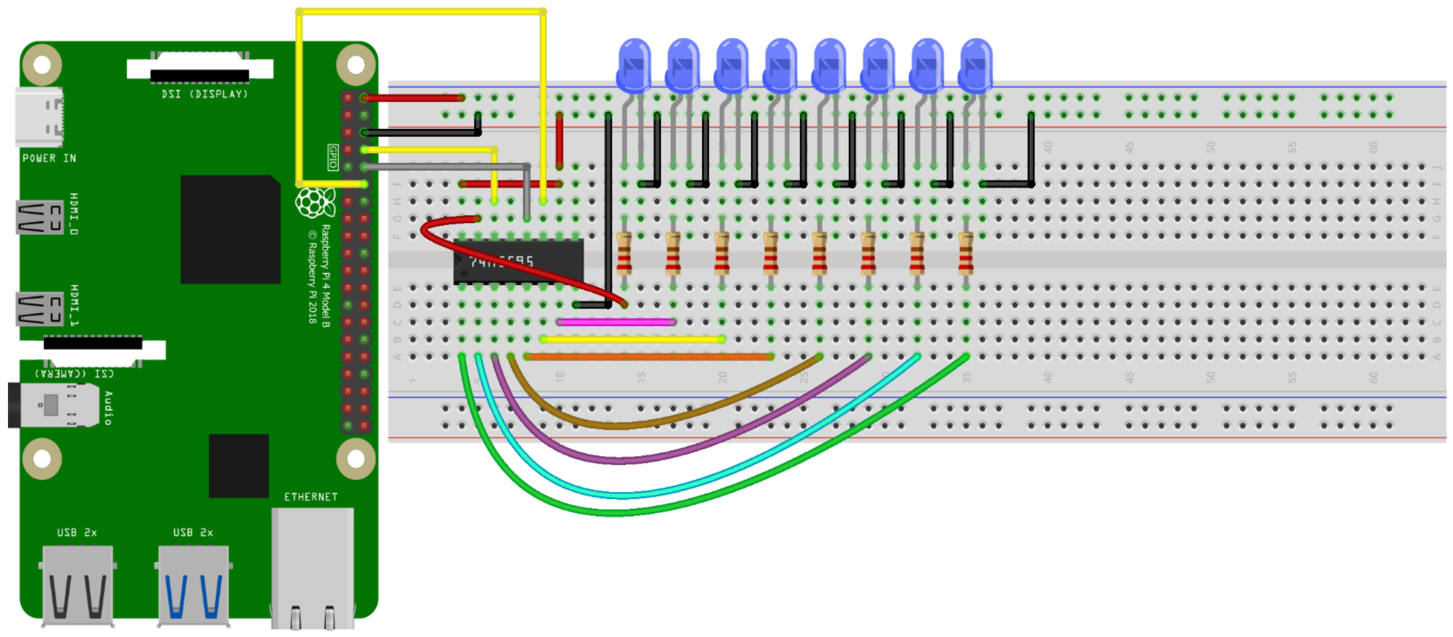
```
#   Program to calculate temperature
import ADC0832
import time
import math
#ADC_CS   =  11,ADC_CLK =  12,ADC_DIO =  13
def init():

    ADC0832.setup()
def loop():
    while True:
        analogVal = ADC0832.getResult()
        Vr = 5 * float(analogVal) / 255
        Rt = 10000 * Vr / (5 - Vr)
        temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+52)))
        temp = temp - 273.15
        print ("temperature = %d C" % temp)
        Tf = (temp * 9.0)/ 5.0 + 32.0;
        print ("temperature = %d F" % Tf)
        time.sleep(1)

if __name__ == '__main__':
    init()
    try:
        loop()
    except KeyboardInterrupt:
        ADC0832.destroy()
        print ("The end !")
```

6.9 Shift register(74HC595)



- GND should be connected to the ground of Raspberry pi.
- VCC is the power supply for 74HC595 shift register which we connect the 5V pin on the Raspberry Pi.
- SER (Serial Input) pin is used to feed data into the shift register a bit at a time.
- SRCLK (Shift Register Clock) is the clock for the shift register. Bits are shifted on the rising edge of the clock.
- RCLK (Register Clock / Latch when logic HIGH, the contents of Shift Register are copied into the Storage/Latch Register; which ultimately shows up at the output.
- SRCLR (Shift Register Clear) pin allows us to reset the entire Shift Register, making all its bits 0, at once. This is a negative logic pin, we need to set the SRCLR pin LOW for reset. When reset not required, this pin should be HIGH.
- QA–QH (Output Enable) are the output pins connected to output like LEDs.

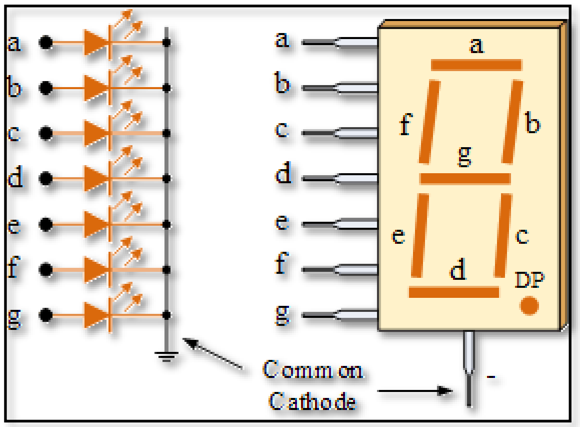
Components: LEDs, 74HC595 IC, wires



```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
data_pin=8
latch_pin=10
clock_pin=12
GPIO.setwarnings(False)
GPIO.setup(data_pin,GPIO.OUT)
GPIO.setup(latch_pin,GPIO.OUT)
GPIO.setup(clock_pin,GPIO.OUT)
def shift_reg(byte):
    GPIO.output(latch_pin,0)
    for y in range (8):
        GPIO.output(data_pin,(byte<<y)& 0x80) # write (push) the binary bits to data pin
one by one
        GPIO.output(clock_pin,1)
        time.sleep(0.1)
        GPIO.output(clock_pin,0)
        time.sleep(0.1)
        GPIO.output(latch_pin,1)
        time.sleep(0.1)
def loop():
    while 1:
        for x in range(255):
            shift_reg(x)
def destroy():
    GPIO.cleanup()

if __name__ == '__main__':      # Program start from here
    try:
        loop()
    except KeyboardInterrupt:    # When 'Ctrl+C' is pressed, the child program destroy()
will be  executed.
        destroy()
```

6.10 7-segment display (S-5101AS)

Interfacing seven segment display to Raspberry pi. In the common cathode display, all the cathode connections of the LED segments connected to ground. The segments are illuminated by applying “HIGH”, or logic “1” signal via a current limiting resistor to forward bias the Anode terminals. For example, if you want to illuminate segment ‘a’ then pin7 on Raspberry Pi that is connected to segment ‘a’ is made HIGH



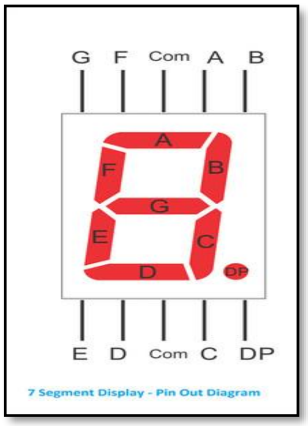
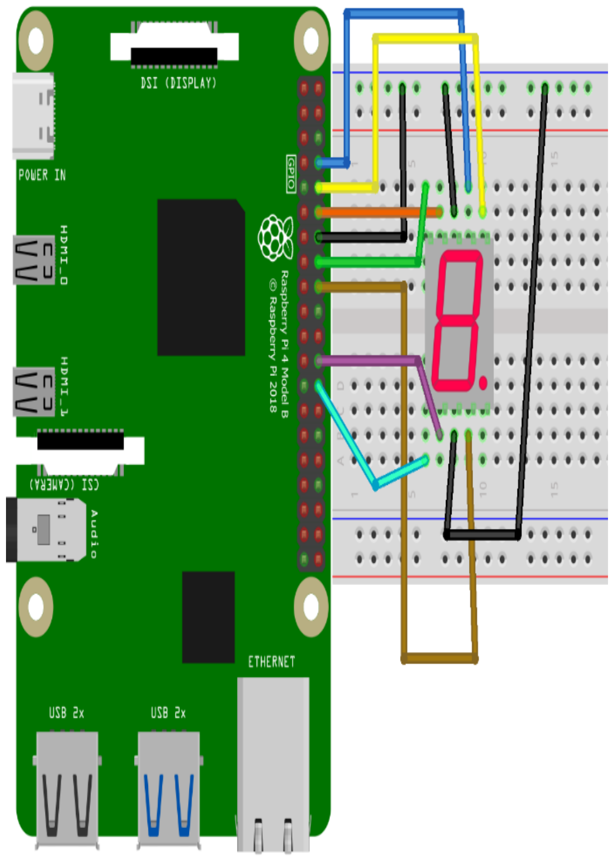


Table for Connections

7 segment	gpio pin
a	14
b	15
c	24
d	8
e	7
f	18
g	23
-	gnd

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
segments =  (14,15,24,8,7,18,23)
#14-a, 15-b,24-c,8-d,7-e,18-f,23-g

for segment in segments:
    GPIO.setup(segment, GPIO.OUT)
    GPIO.output(segment, 0)

num = [[0,0,0,0,0,0,0],
       [1,1,1,1,1,1,0],
       [0,1,1,0,0,0,0],
       [1,1,0,1,1,0,1],
       [1,1,1,1,0,0,1],
       [0,1,1,0,0,1,1],
       [1,0,1,1,0,1,1],
       [1,0,1,1,1,1,1],
       [1,1,1,0,0,0,0],
       [1,1,1,1,1,1,1],
       [1,1,1,1,0,1,1]
       ]

try:
    while True:
        for digit in range(0,11):
            for loop in range(0,7):
                GPIO.output(segments[loop],
num[digit][loop])
            time.sleep(1)
            for loop in range(0,7):
                GPIO.output(segments[loop],
num[0][loop])
            time.sleep(1)
finally:
    GPIO.cleanup()
```



6.11 I2C LCD (LCD MOD-13)

LCD I2C uses I2C interface, so it has 4 pins NAMELYGND, VCC, SDA (I2C data signal), SCL (I2C clock signal). Components: I2C LCD, Jumper wires. Note: If the LED does not display characters then try LCD contrast adjust.

We use a library found in Github you can directly download the file from https://github.com/sunfounder/SunFounder_SensorKit_for_RPi2/tree/master/Python where you download LCD1602 file and save this module in the location where you have your project files.

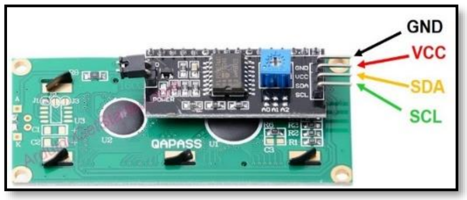
If you want to download using command line type the following on command line (or

git clone https://github.com/sunfounder/SunFounder_SensorKit_for_RPi2.git . Now this file will be downloaded. Navigate the file as given below

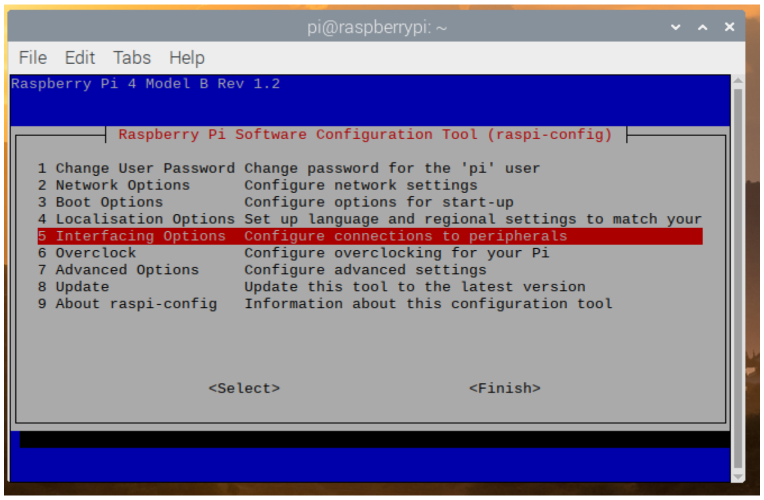
`SunFounder_sensorkit_for_RPi2.git->Python->LCD1602.py` now select the LCD1602.py file and paste this file in the location where you have saved your python project files. We do this because we need to import this file in our program that we are going to write.

Now we have to activate I2C the following commands are used

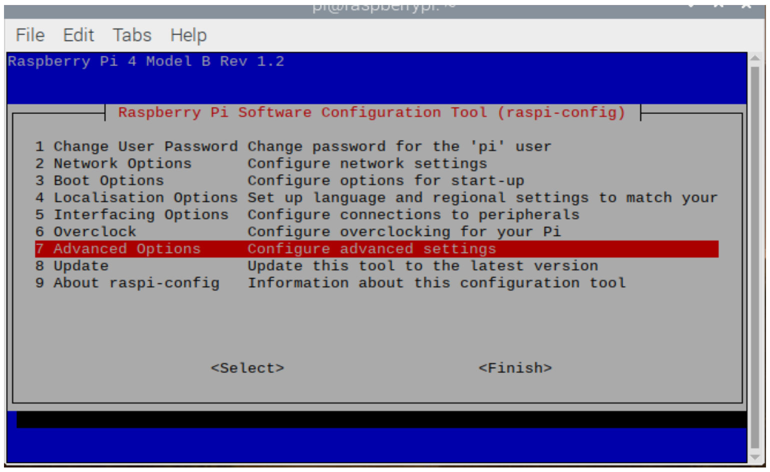
- `sudo apt-get install -y python-smbus`
- `sudo apt-get install -y i2c-tools`
- **`sudo raspi-config`**



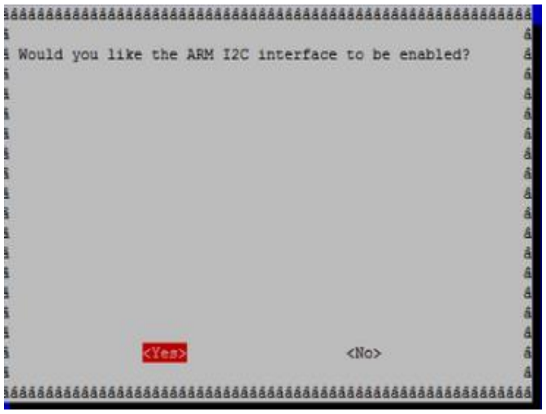
Now we should Select Interfacing Options ->Advanced options -> I2C. Now enable the ARM I2C interface by selecting **Yes.**



```
Raspberry Pi 4 Model B Rev 1.2

         Raspberry Pi Software Configuration Tool (raspi-config)

 1 Change User Password Change password for the 'pi' user
 2 Network Options      Configure network settings
 3 Boot Options         Configure options for start-up
 4 Localisation Options Set up language and regional settings to match your
 5 Interfacing Options  Configure connections to peripherals
 6 Overclock            Configure overclocking for your Pi
 7 Advanced Options     Configure advanced settings
 8 Update               Update this tool to the latest version
 9 About raspi-config   Information about this configuration tool

                  <Select>                    <Finish>
```

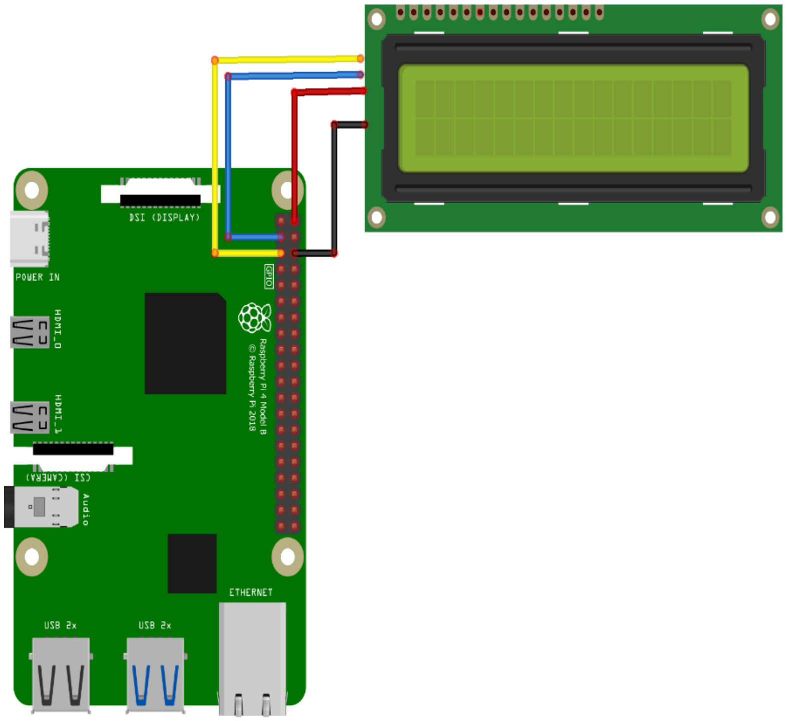


```
Raspberry Pi 4 Model B Rev 1.2

         Raspberry Pi Software Configuration Tool (raspi-config)

 1 Change User Password Change password for the 'pi' user
 2 Network Options      Configure network settings
 3 Boot Options         Configure options for start-up
 4 Localisation Options Set up language and regional settings to match your
 5 Interfacing Options  Configure connections to peripherals
 6 Overclock            Configure overclocking for your Pi
 7 Advanced Options     Configure advanced settings
 8 Update               Update this tool to the latest version
 9 About raspi-config   Information about this configuration tool

                  <Select>                    <Finish>
```



```
Would you like the ARM I2C interface to be enabled?

              <Yes>                    <No>
```

```
#!/usr/bin/env python
import LCD1602
import time

def setup():
  LCD1602.init(0x27, 1)
  LCD1602.write(0, 0, 'Rasberry Pi Kit')
  LCD1602.write(1, 1, 'Abra Electronics')
  time.sleep(2)
def destroy():
  GPIO.cleanup()

if __name__ == "__main__":
  try:
    setup()
  except KeyboardInterrupt:
    destroy()
```

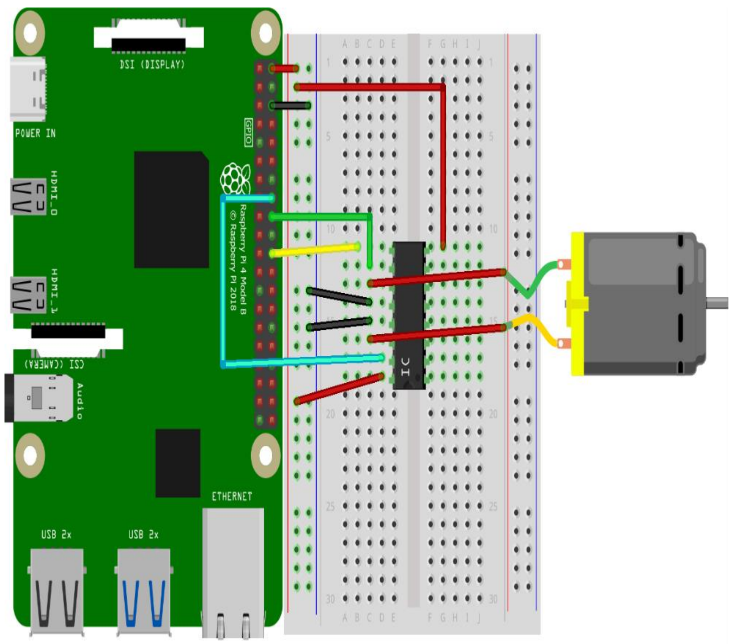


Connect the SDA pin to SDA pin of Raspberry pi and
similarly SCL to SCL pin of Raspberry pi.

6.12 Motor Driver (L293D)

L293D Motor driver can run two motors at a time on either direction. In this experiment we use L293D to run a single DC motor. It works on the principle of H-bridge which allows the voltage to be flown in either direction. The Device is a monolithic integrated high voltage, high current four channel driver designed to accept standard DTL or TTL logic levels and drive inductive loads (such as relays solenoids, DC and stepping motors) and switching power transistors. To simplify use as two bridges each pair of channels is equipped with an enable input. A separate supply input is provided for the logic, allowing operation at a lower voltage and internal clamp diodes are included. This device is suitable for use in switching applications at frequencies up to 5 kHz.
Connections: PIN 22 - PIN 9 of IC , PIN 18 - PIN 10 of IC, PIN 16 - PIN 15 of IC

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
input_1=16
input_2=18
enable_1=22
GPIO.setup(input_1,GPIO.OUT)
GPIO.setup(input_2,GPIO.OUT)
GPIO.setup(enable_1,GPIO.OUT)
def setup():
    while 1:
        time.sleep(5)
        print ("FORWARD MOTION")
        GPIO.output(enable_1,GPIO.HIGH)
        GPIO.output(input_1,GPIO.HIGH)
        GPIO.output(input_2,GPIO.LOW)
        time.sleep(5)
        print ("STOP")
        GPIO.output(enable_1,GPIO.LOW)
        time.sleep(1)
        print ("BACKWARD MOTION")
        GPIO.output(enable_1,GPIO.HIGH)
        GPIO.output(input_1,GPIO.LOW)
        GPIO.output(input_2,GPIO.HIGH)
def destroy():
    GPIO.cleanup()

if __name__ == "__main__":
    try:
        setup()
    except KeyboardInterrupt:
        destroy()
```



6.13 Distance Sensor (HC-SR04)

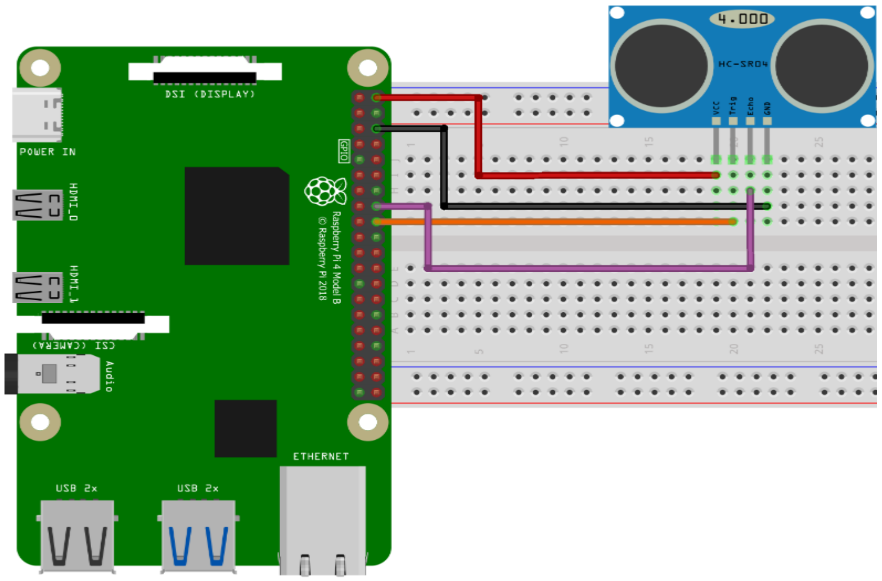
Ultrasonic Sensor HC-SR04 is a sensor that can measure distance. It emits an ultrasound at 40000 Hz (40kHz) which travels through the air and if there is an object or obstacle on its path It will bounce back to the module. Using the time taken to travel and the speed of the sound distance is calculated.

If x is the distance between point A and point B then 2x will be the distance travelled from A to B and the B to A. We know that speed of the ultrasound is about 340 meters per second.

- Distance = Speed*Time.
- Distance=2x.
- 2x= 340*Time x=340*Time/2

Trig pin to GPIO-24

Echo pin to GPIO 23

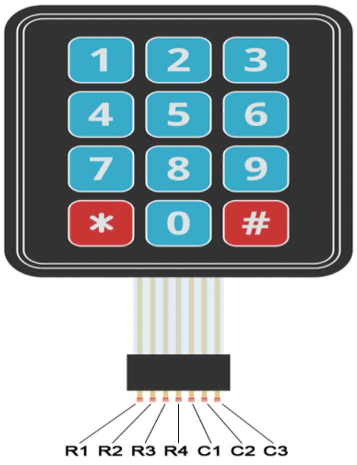


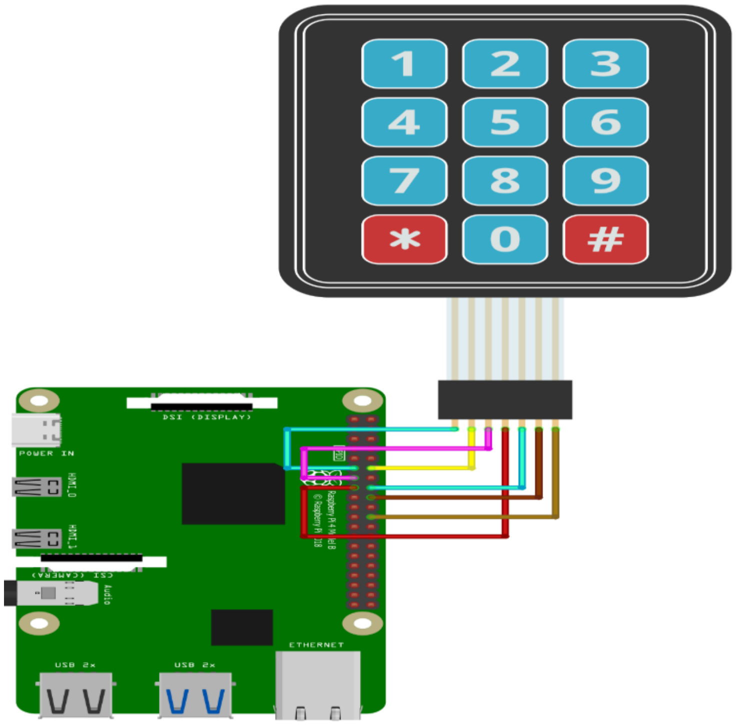
```
from gpiozero import DistanceSensor
from signal import pause
from time import sleep
sensor = DistanceSensor(23, 24)
def loop():
     while True:
       print("Distance to nearest object is", sensor.distance, "meters")
       sleep(1)

if __name__ == "__main__":
  try:
    loop()
  except KeyboardInterrupt:
    print("Interrupted")
```

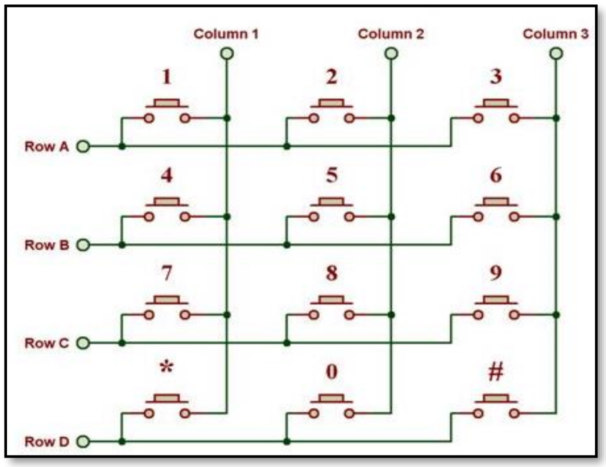
6.14 4x3 Keypad (419-ADA)

4x3 keypad has 4 rows and 3 columns. If **no key** has been pressed, then all columns will remain HIGH. Pressing a button shorts one of the row lines to one of the column lines, allowing current to flow between them. For example, when key ‘2’ is pressed, column 2 and row 1 are shorted so column 2 and row 1 are LOW.

Components: 4x3 Keypad, Jumper wires



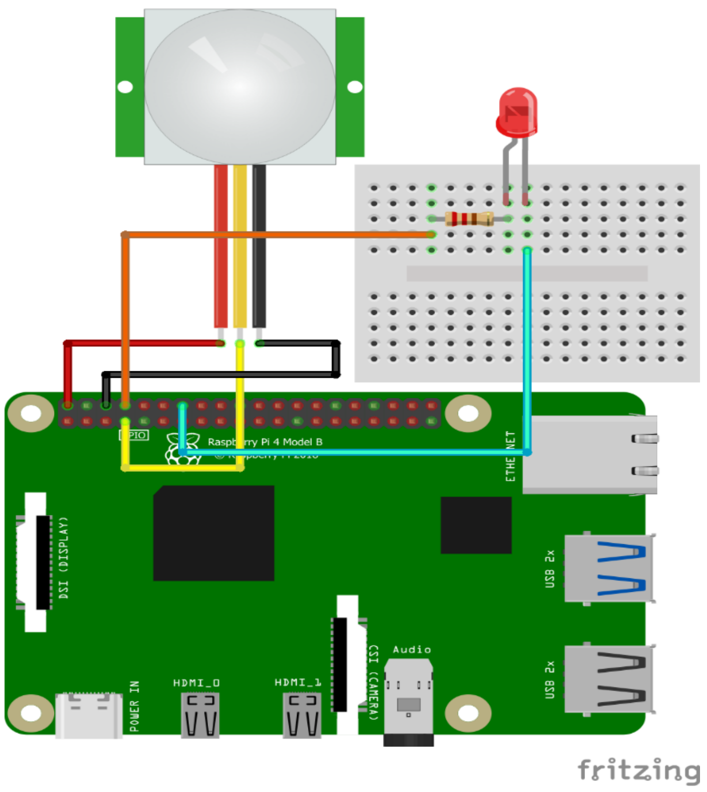




```
import RPi.GPIO as GPIO
import time

R1 = 17 #BCM NUMBERING
R2 = 18
R3 = 27
R4 = 22
C1 = 23
C2 = 24
C3 = 25


GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

GPIO.setup(R1, GPIO.OUT)
GPIO.setup(R2, GPIO.OUT)
GPIO.setup(R3, GPIO.OUT)
GPIO.setup(R4, GPIO.OUT)

GPIO.setup(C1, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) #pull down the gpio pin
GPIO.setup(C2, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(C3, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

def readLine(line, characters):
  GPIO.output(line, GPIO.HIGH)
  if(GPIO.input(C1) == 1):
    print(characters[0])
  if(GPIO.input(C2) == 1):
    print(characters[1])
  if(GPIO.input(C3) == 1):
    print(characters[2])
  GPIO.output(line, GPIO.LOW)

try:
  while True:
    readLine(R1, ["1","2","3"])
    readLine(R2, ["4","5","6"])
    readLine(R3, ["7","8","9"])
    readLine(R4, ["*","0","#"])
    time.sleep(0.2)
except KeyboardInterrupt:
  print("End")
```

6.15 PIR sensor (SENS-PIR)

A passive infrared **sensor** (**PIR sensor**) is an electronic **sensor** that measures infrared (IR) light radiating from objects in its field of view. They are most often used in **PIR**-based **motion** detectors. **PIR sensors** are commonly used in security alarms and automatic lighting **applications**. Once the circuit is built and uploaded into Raspberry Pi try moving objects in front of PIR sensor which causes LED to glow, output can also be seen on serial monitor.



Components: SENS-PI

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(4, GPIO.IN)
GPIO.setup(14, GPIO.OUT) # to connect to led

def setup():
  while 1:
    if GPIO.input(4):
      GPIO.output(14, GPIO.HIGH)#glow led
      print ("motion detected")
      sleep(.5)
      GPIO.output(14, GPIO.LOW)
def destroy():
  GPIO.cleanup()
if __name__ == "__main__":
  try:
    setup()
  except KeyboardInterrupt:
    destroy()
```

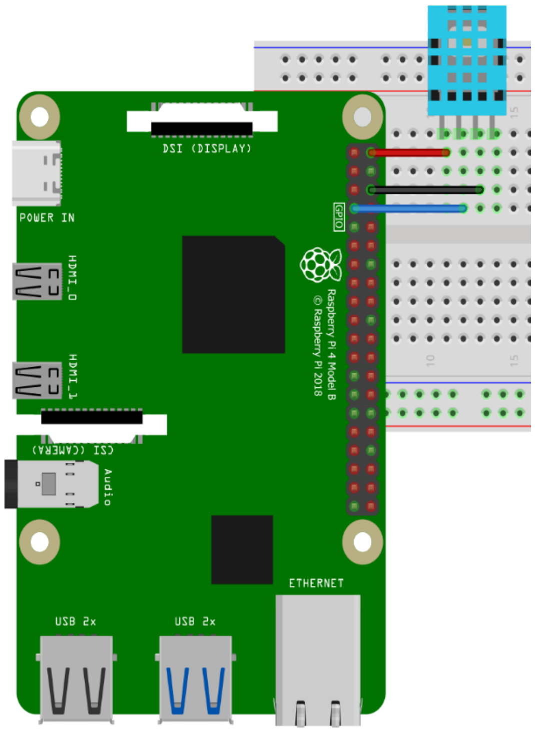
6.16 Humidity sensor (SENS-DHT11-BB)

In this project we interface a humidity sensor to measure temperature and humidity. DHT11 consist of a humidity sensing component, a NTC temperature sensor (or thermistor) and an IC. It has a humidity sensing component which has two electrodes with moisture holding substrate between them. When the humidity changes, the conductivity of the substrate changes or the resistance between these electrodes' changes. This change in resistance is measured and processed by the IC which makes it data to be read by a microcontroller.

Install `dht11 library using`

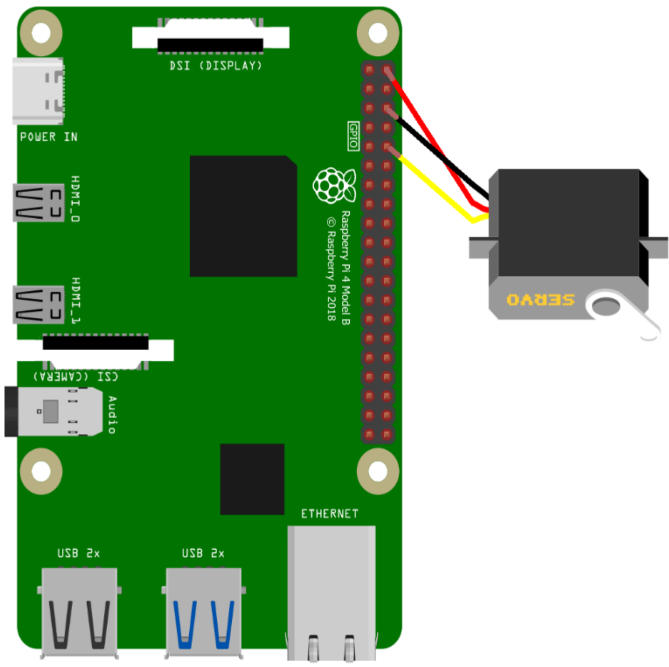
```
pip3 install dht11
```

```
import RPi.GPIO as GPIO
import dht11
from time import sleep
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
dht_sensor=dht11.DHT11(pin=4)

def setup():
  while 1:
    result=dht_sensor.read()
    sleep(2)
    if result.is_valid():
      print ("Temperature is %-3.2f C"%result.temperature)
      print ("Humidity is %-3.2f %%"%result.humidity)

def destroy():
  GPIO.cleanup()
if __name__ == "__main__":
  try:
    setup()
  except KeyboardInterrupt:
    destroy()
```



6.17 Servo (SG-90)

A **servo motor** is an electrical device which can rotate an object with high precision. If you want to rotate an object at a specific angle, we use servo motor. Servo motor is controlled by PWM (Pulse with Modulation). In this project we can connect small servo motors directly to a Raspberry pi to control the shaft position very precisely. Raspberry Pi sends PWM signal to the servo which then rotates by an angle depending upon the pulse width.

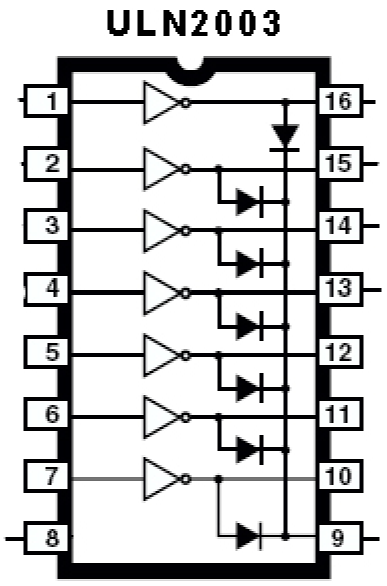


- If PWM's width = WIDTH_MIN, the servo motor rotates to 0°.
- If PWM's width = WIDTH_MAX, the servo motor rotates to 180°.

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(15,GPIO.OUT)
servo_motor=GPIO.PWM(15,500) # SET THE FREQUENCY TO 500HZ
servo_motor.start(50) #SET DUTY CYCLE TO 50%
def setup():
  while 1:
    for duty_cycle in range(0,100,5):
      servo_motor.ChangeDutyCycle(duty_cycle)
      sleep(1)
    for duty_cycle in range(100,0,-5):
      servo_motor.ChangeDutyCycle(duty_cycle)
      sleep(1)
def destroy():
  GPIO.cleanup()
if __name__ == "__main__":
  try:
    setup()
  except KeyboardInterrupt:
    destroy()
```

6.18 Stepper motor with driver (MOT-28BYJ-48)

The 28BYJ-48 stepper motor is a stepper motor, which converts electrical pulses into discrete mechanical rotation. The 28BYJ-48 stepper motor consumes high current and hence, we will need to use a driver IC ULN2003 to control the motor with Raspberry pi. ULN2003 is a 7-channel inverter circuit.

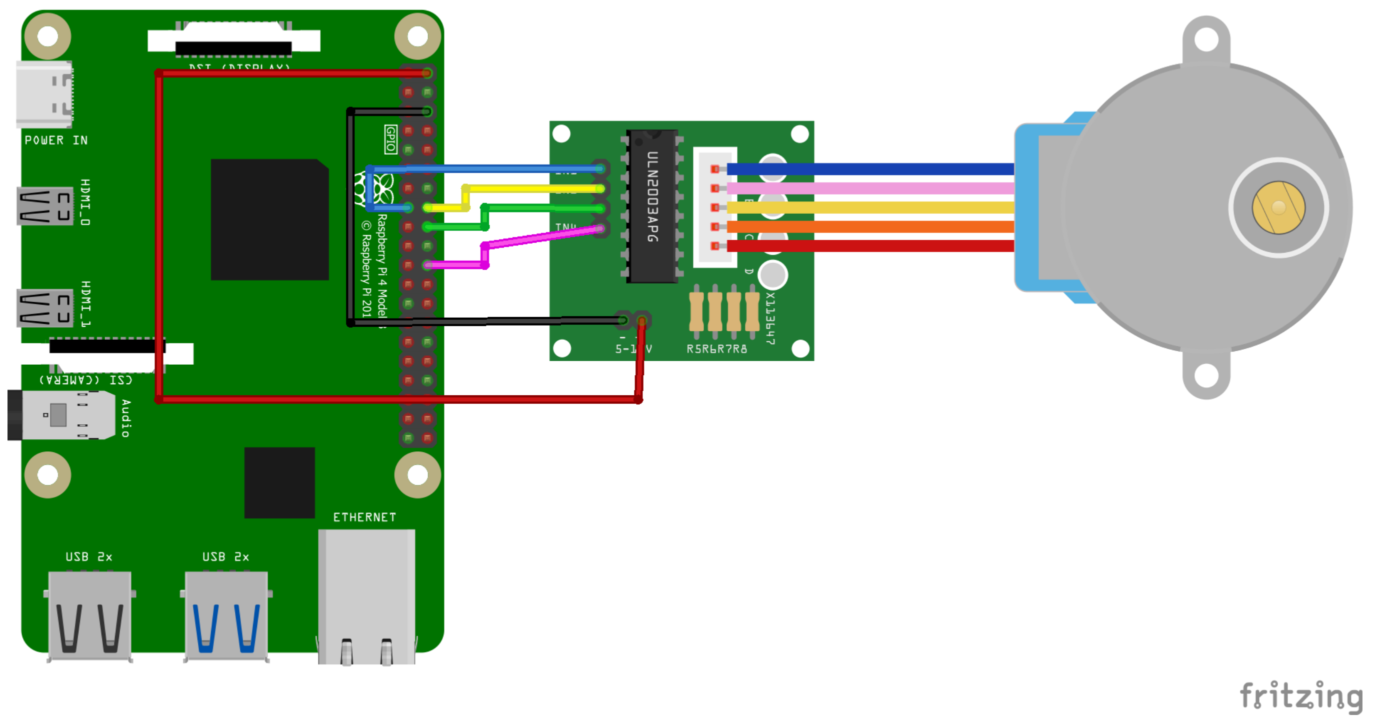


The following codes define the step commands.

8 Step : A – AB – B – BC – C – CD – D – DA

4 Step : AB – BC – CD – DA (Usual application)

Step	Hexadecimal value	IN4	IN3	IN2	IN1
A	01H	0	0	0	1
AB	03H	0	0	1	1
B	02H	0	0	1	0
BC	06H	0	1	1	0
C	04H	0	1	0	0
CD	0CH	1	1	0	0
D	08H	1	0	0	0
DA	09H	1	0	0	1



```
import RPi.GPIO as GPIO
from time import sleep
INPUT_PINS = (22,23,24,25)
rotation_per_minute =10
stepsPerRevolution = 1042
speed = (60/rotation_per_minute)/stepsPerRevolution
def setup():
  GPIO.setwarnings(False)
  GPIO.setmode(GPIO.BCM)
  for i in INPUT_PINS:
    GPIO.setup(i, GPIO.OUT)
def clock_wise():
  for j in range(4):
    for i in range(4):
      GPIO.output(INPUT_PINS[i],0x99>>j & (0x08>>i))
      sleep(speed)
def anti_clock_wise():
    for j in range(4):
      for i in range(4):
        GPIO.output(INPUT_PINS[i],0x99<<j & (0x80>>i))
        sleep(speed)
def destroy():
  GPIO.cleanup()
if __name__ == "__main__":
  try:
    setup()
    while 1:
      for x in range(200):
        clock_wise()
      for x in range(200):
        anti_clock_wise()
  except KeyboardInterrupt:
    destroy()
```