



# Elecrow Raspberry Pi & Arduino Starter Kit

## CATALOG

Introduction.....	3
Driver Installation.....	3
LESSON 1: Blink LED.....	4
LESSON 2: Button.....	8
LESSON 3: Ball switch.....	14
LESSON 4: Active buzzer.....	19
LESSON 5: Passive buzzer.....	24
LESSON 6: RGB LED.....	35
LESSON 7: 1 Digit 7 Segment display.....	40
LESSON 8: 4 Digit 7 Segment display.....	45
LESSON 9: Heart-shaped display.....	51
LESSON 10: 9G Servo.....	56
LESSON 11: Step motor.....	61
LESSON 12: Ultrasonic ranging.....	66
LESSON 13: Touch lamp.....	72
LESSON 14: PCF8591 Module.....	77
LESSON 15: Flame sensor.....	82
LESSON 16: Photo resistance sensor (light sensor).....	87
LESSON 17: Thermistor sensor.....	93
LESSON 18: Potentiometer.....	98
LESSON 19: Water level monitoring.....	103
LESSON 20: Joystick.....	109
LESSON 21: IR Remote.....	114
LESSON 22: IR Remote control LED.....	119
LESSON 23: DHT11.....	125
LESSON 24: LCD1602 With IIC.....	130
LESSON 25: Temperature and humidity monitoring.....	100

## INTRODUCTION

Thank you for purchasing Elecrow Starter kit for Raspberry Pi & Arduino!

In this lessons file, we'll go through all the components that you received by purchasing our product, we will explain, step by step, the functionality of each of the components and how to use them.

This lesson file is a great start for the making world, making the first step in creating something on your own by connecting hardware and programming them to function in different ways!

For each lesson, there will be example demo code that you can run available from our GitHub account. We will keep the GitHub account updated so make sure to follow up!

We hope you will enjoy this lessons as much as we did while writing them.

---

## DRIVER INSTALLATION & GIT CLONE

In order to use the lessons first we'll need to clone the repository from GitHub that includes all the examples that we'll need.

To do so, run the following commands:

```
git clone https://github.com/Elecrow-RD/Raspberry-Pi-Starter-Kit.git
cd Raspberry-Pi-Starter-Kit/Drivers
```

After cloning the repository and CD (going into the folder) Drivers, we'll now continue in installing the actual drivers.

There are couple of drivers we'll need to install:

- \* **adafruit\_python\_charLCD** - Adafruit library to control the LCD screen
- \* **adafruit\_python\_DHT** - Adafruit library to control the DH11 sensor
- \* **Adafruit\_Python\_LED\_Backpack** - Adafruit library to control the segment LED
- \* **Luma.LED\_Matrix** - Library to control the matrix LED

We have all the drivers we need in the drivers/ folder in our Raspberry-Pi-Starter-Kit repository. So let's proceed installing them all. We are going to install each driver separately so make sure to follow the instructions very carefully.

Before you start make sure your current location on the terminal states "Drivers" as the drivers folder you are currently at.

---

You will need to install each of those drivers by running the following command for each one:

Adafruit\_Python\_CharLCD:

```
cd Adafruit_Python_CharLCD
sudo python setup.py install
sudo python3 setup.py install
cd ..
```

Adafruit\_Python\_DHT:

```
cd Adafruit_Python_DHT
sudo python setup.py install
sudo python3 setup.py install
cd ..
```

Adafruit\_Python\_LED\_Backpack:

```
cd Adafruit_Python_LED_Backpack
sudo python setup.py install
sudo python3 setup.py install
cd ..
```

Luma Matrix LED:

```
cd luma.led_matrix
sudo python setup.py install
sudo python3 setup.py install
cd ..
```

---

Make sure there are no errors on the way and congratulations! you've successfully installed all the required drivers.

Please follow the next steps to prepare for the IR LED driver installation.

### IR LED driver installation

First, let's install the lirc library, the main library that will help us send and receive IR codes using our sensor later on in this lessons:

```
sudo apt-get install lirc
```

After successfully installing the lirc library, we need to install the python support for lirc so we could write python code and being able to connect it with the lirc library all together.

```
sudo pip install python-lirc  
sudo pip3 install python-lirc
```

Note: on the latest Raspbian pip3 install python-lirc will not work, we are working to solve it ASAP. You'll need to use python2 for the IR script for now.

Inside CrowPi/Drivers/LIRC folder that we cloned, there are 3 files you need to copy the files to the LIRC configuration directory, from the CrowPi folder run the following commands:

```
sudo cp Drivers/LIRC/* /etc/lirc
```

After you moved the configuration file, edit your boot config file

---

```
sudo nano /boot/config.txt
```

And where it says

```
# Uncomment this to enable the lirc-rpi module
#dtoverlay=lirc-rpi
```

change it to this

```
# Uncomment this to enable the lirc-rpi module
dtoverlay=gpio-ir,gpio_pin=20
```

previously it used to be `dtoverlay=lirc-rpi` but this one got deprecated in the newer version of Raspbian.

Execute the following commands to copy the configuration files

```
sudo cp /etc/lirc/lirc_options.conf.dist /etc/lirc/lirc_options.conf
sudo cp /etc/lirc/lircd.conf.dist /etc/lirc/lircd.conf
```

edit `/etc/lirc/lirc_options.conf` by writing the command `sudo nano /etc/lirc/lirc_options.conf` and modify the following lines to be exact as here:

```
driver = default
device = /dev/lirc0
```

now reboot

```
sudo reboot
```

---

Once the reboot successfully finished, run apt-get install Lirc once again to fix the previous errors if any

```
sudo apt-get install lirc
```

Last step, stop the LIRC library so we could use the IR driver with our python script

```
sudo /etc/init.d/lirc stop
```

Note: if you get runtime error that the command cannot be found, maybe you have a different version of LIRC, try this command instead:

```
sudo /etc/init.d/lircd stop
```

Well done!

You are now officially ready to start the python lessons properly when all the drivers and everything is needed is already installed and ready for use.

let's get going!

## LESSON 1: BLINK LED





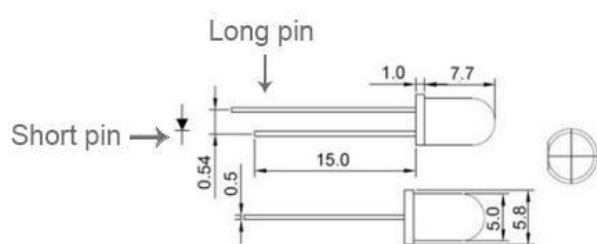
## Introduction

The Blink LED will be our first lesson, during this lesson we'll learn what is LED and how it works, how to connect it properly to the raspberry pi using GPIO ports and then we'll use example script to run and test a code with our hardware.

The Blink LED is a very simple example to start with, we will use 5mm LED of any colour and make it "blink" turn on and off rapidly in timely manner.

## Specification



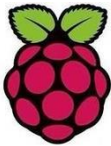




Below you can see the LED specification, the long pin indicate the positive side while the short pin indicate the negative side.



Note: it's important not to mistake accidentally the long pin with the short pin, that will cause the circuit not to work properly.

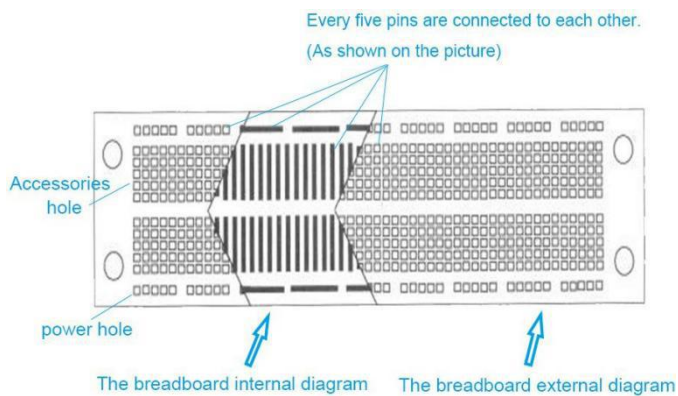
## Hardware required

Below is the list of the hardware required for us to accomplish the lesson, all the hardware is included in the kit. Please make sure to take the right hardware and prepare it for the lesson. The resistor can be either 220 or 300 ohms, both will work.

Material diagram	Material Name	Number (amount)
	LED	1
	220/300Ω resistor	1
	Raspberry Pi Board	1
	T-Cubler Plus	1
	40P GPIO Cables	1
	Breadboard	1
	Jumper wires	Several

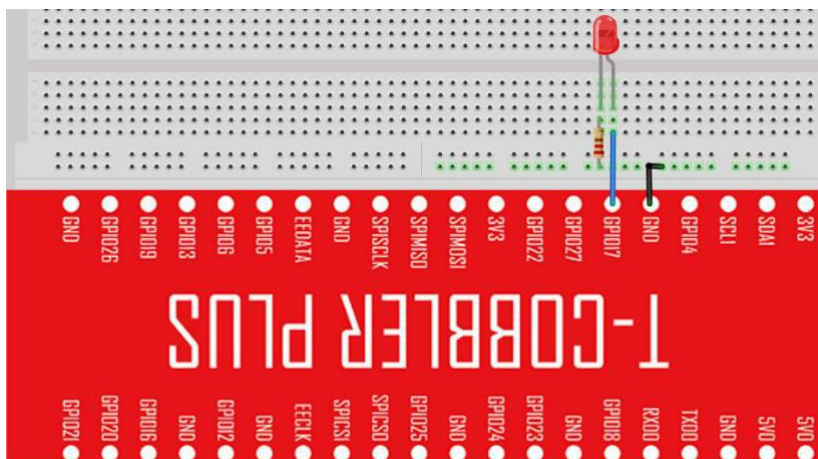
## Breadboard Schematic

Below is the breadboard schematics, make sure to be familiar with it. it's important to understand how the breadboard works in order not to damage components that are included with this kit.



## Connection Diagram

As the diagram states, we will connect the negative pin of the LED to the resistor and the positive directly to the Raspberry Pi GPIO.



## Connection

LED

Raspberry Pi

---

LED	Raspberry Pi
Long Pin (+)	GPIO17
Short Pin (-)	GND

## Code Overview, Code Overview, Compile and run

Our code is quite simple. We'll define GPIO 17 as the blinking LED pin and then setup the pin as GPIO.OUT, we will go through a while loop (forever till the program quit using CTRL-C or CTRL-Z, We will turn the LED on by setting GPIO.HIGH, wait 0.2 seconds (200 milliseconds) and turn it off by setting GPIO.LOW.

We will repeat the process forever.

Compile the script by going into the samples directory and running: **python3 blink.py**

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  # http://elecrow.com/
4
5  import time
6  import RPi.GPIO as GPIO
7
8  # define LED pin
9  led_pin = 17
10
11 # set GPIO mode to GPIO.BCM
12 GPIO.setmode(GPIO.BCM)
13 # set pin as output
14 GPIO.setup(led_pin, GPIO.OUT)
15
16 try:
17     while True:
18         # turn on LED
19         GPIO.output(led_pin, GPIO.HIGH)
20         # Wait half a second
21         time.sleep(0.2)
22         # turn off LED
23         GPIO.output(led_pin, GPIO.LOW)
24         # Wait half a second
25         time.sleep(0.2)
26 except KeyboardInterrupt:
27     # CTRL+C detected, cleaning and quitting the script
28     GPIO.cleanup()
```

## Application effect

Running the program, will turn on an LED on for one second, then off for one second, repeatedly.

---

It's inside a while loop, the program will stop by pressing CTRL+C on the keyboard or CTRL+Z.

## LESSON 2: BUTTON



### Introduction

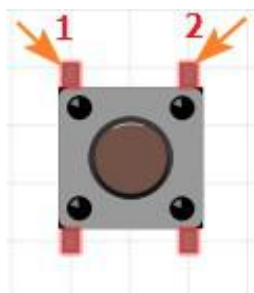
In this lesson we'll learn how to use the button.

The button is very useful in many applications for example turn on light by press or even play some music.

In our specific example we'll use the button to indicate if it was pressed or released.

### Specification

Below is the button specification, as we can see, there are 2 pins, once the button is closed, it will close the circuit and will cause the GPIO INPUT to indicate as "HIGH" once the button is open, the circuit is open and the GPIO INPUT will indicate as "LOW".



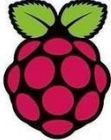






---

### Hardware required

Below is the list of all the components that are required for this lesson.

Make sure to note that the resistor is 10K (kilo) ohm and not 10 ohm.

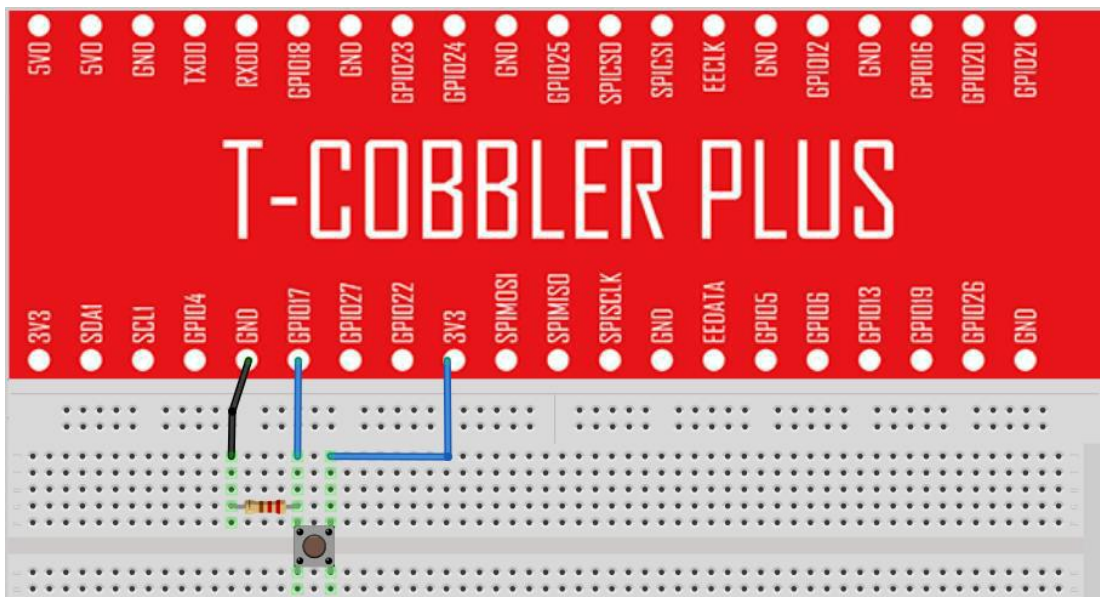
Material diagram	Material Name	Number (amount)
	Button	1
	10K $\Omega$ resistor	1
	Raspberry Pi Board	1
	T-Cubbler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wire	Several

---

### Connection Diagram

As we can see in the diagram, we connect the button to the 10K ohm resistor on the negative side as well as to the GPIO port at GPIO17. The other pin of the button goes to 3V pin on the raspberry pi.

Please note: if you connect the button on the wrong direction the raspberry pi might indicate the opposite, GPIO HIGH when the button is released and GPIO LOW when the button is pressed, therefore the script might not work as expected.



### Connection

Button	Raspberry Pi
Pin 1	GPIO17
Pin 2	3V3

---

### Code Overview, Code Overview, Compile and run

Let's take a look through our code: we set the button pin as GPIO 17 then the mode as GPIO.BCM, the button GPIO will be GPIO.IN as we receive input from the button whenever it was pressed or not. Then we go while true (forever) if GPIO.INPUT button is TRUE means the button is pressed, we print "button pressed" else, if it's released, we print "button released".

Compile the script by going into the samples directory and running: **python3 button.py**

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  # http://elecrow.com/
4
5  import RPi.GPIO as GPIO
6  import time
7
8  # configure button pin
9  button_pin = 17
10
11 # set board mode to GPIO.BCM
12 GPIO.setmode(GPIO.BCM)
13
14 # setup button pin as input
15 GPIO.setup(button_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
16
17 try:
18     while True:
19         # check if button pressed
20         if(GPIO.input(button_pin)):
21             # Button is pressed
22             print("Button Pressed")
23         else:
24             # it's not pressed
25             print("Button Released")
26         time.sleep(0.1)
27 except KeyboardInterrupt:
28     GPIO.cleanup()
```



---

### Application effect

Running the program, will print “button pressed” if the button is pressed and “button released” if the button is released.

## LESSON 3: BALL SWITCH

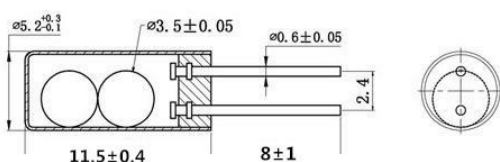


### Introduction

In this lesson we'll learn about the ball switch, also called “tilt sensor” once the switch is tilt to one side, it could be either open or closed. The ball will touch the side of the switch causing it to close a circuit, tilting it to the other side will cause the circuit to be open (due to the ball not touching the side of it)



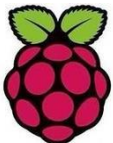




### Specification

Below is the ball switch specification, as we can see inside the switch there small balls, once they touch the left side, it will close the circuit between the two “sticks” (pins) allowing the current to flow.



### Hardware required

Below is the hardware list for our lesson, please make sure you are using the 10K ohm resistor and not a 10 ohm resistor for this example.

Material diagram	Material Name	Number (amount)
	Ball Switch	1
	10K $\Omega$ resistor	1
	Raspberry Pi Board	1
	T-Cubbler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

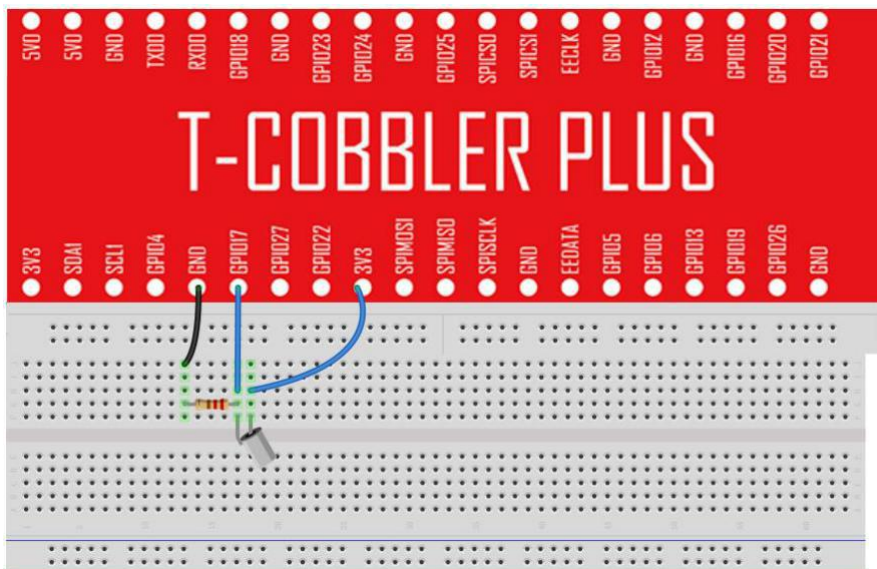
---

### Connection Diagram

As can be seen on the diagram, we connect the negative pin to the GND in the raspberry pi as well as to the 10K ohm resistor and to one pin of the ball switch.

Note: it doesn't matter which pin of the ball switch you choose, the decision, might effect the script. For example, GPIO HIGH might be GPIO LOW or GPIO LOW might be GPIO HIGH.

If you see that the script is not as what expected in this tutorial, try to change the pins of the ball switch.



### Connection

As we mentioned, the ball switch doesn't have a specific pins, try both and see the results.

---

Component	Raspberry Pi
Pin 1	GPIO17
Pin 2	3V3

### Code Overview, Code Overview, Compile and run

Let's walk through our code: we setup the switch pin to GPIO17 as GPIO.IN because we receive input from the ball switch whenever he close the circuit by tilting it or not. Then we go for while loop (forever) and print "BALL is HIGH" if it's closing the circuit or "BALL is LOW" if not. You can tilt the ball switch around to see the data changing.

Compile the script by going into the samples directory and running: **python3 ball\_switch.py**

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  # http://elecrow.com/
4
5  import RPi.GPIO as GPIO
6  import time
7
8  # configure button pin
9  ball_switch_pin = 17
10
11 # set board mode to GPIO.BCM
12 GPIO.setmode(GPIO.BCM)
13
14 # setup ball pin input
15 GPIO.setup(ball_switch_pin, GPIO.IN)
16
17 try:
18     while True:
19         # check ball state
20         if(GPIO.input(ball_switch_pin)):
21             # Ball is high means it's closing the circuit by touching
22             print("Ball is HIGH")
23         else:
24             # Ball is low means it's not closing the circuit
25             print("Ball is LOW")
26             time.sleep(0.1)
27 except KeyboardInterrupt:
28     GPIO.cleanup()
```

---

### Application effect

The script will print “BALL is HIGH” if it’s closing the circuit or “BALL is LOW” if not. You can tilt the ball switch around to see the data changing.

## LESSON 4: ACTIVE BUZZER



### Introduction

Active buzzer is a very useful module that can be used in many applications

For example: alarm clock, movement detection or as an alert to an UPS to let you know the battery is running low.


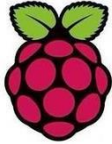




No matter what your project is about, buzzer probably can be very useful case.

In the tutorial below, we'll learn how the buzzer works and how to control it using the raspberry pi GPIO, how to wire it using the breadboard and the jumpers and use it with Python script sample attached in our GitHub page.

Make sure not to confused the active buzzer with the passive buzzer in the kit, the difference is that the active buzzer have a sticker on top (a seal) the passive buzzer doesn't have one.

### Hardware required

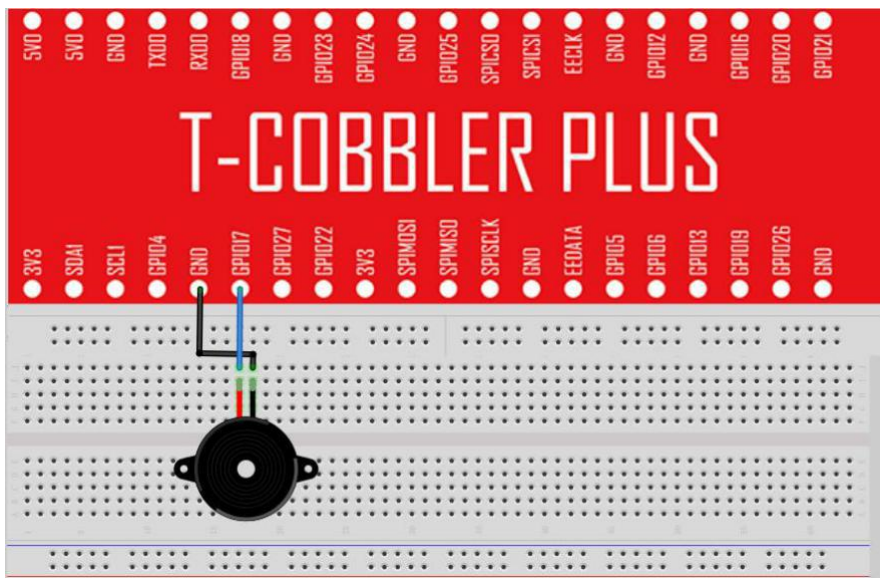
Below is the list of hardware, make sure not to confuse the active buzzer and passive buzzer in the kit. Both of them have a tutorial going through and explaining how they work but for now, we focus on the active buzzer only.

Material diagram	Material Name	Number (amount)
	Active Buzzer	1
	Raspberry Pi Board	1
	T-Cubbler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper cables	Several

---

## Connection Diagram

as we can see in the diagram, the connection is pretty simple. we connect the positive pin of the buzzer into GPIO17 on the raspberry pi and the negative pin to the ground.



## Connection

Please note: the long pin of the buzzer is the positive pin while the short pin of the buzzer is the negative pin.

Active buzzer	Raspberry Pi
Long Pin (+)	GPIO17
Short Pin (-)	GND

---

## Code Overview, Compile and run

Let's overview and test out buzzer code: the passive buzzer code is pretty simple. We don't use loop in this code to avoid the buzzing sound, we setup the buzzer as GPIO17 as GPIO.OUT because we will output a GPIO.HIGH or LOW when we want to control the buzzer activity, then we set the GPIO as GPIO.HIGH for 0.5 seconds (prepare the noisy buzzing sound) and after half a second we turn it off by GPIO.LOW command.

To wrap this all up, we use the command `GPIO.cleanup()` to clean the GPIO17 pin In case we want to use it for something else.

Compile the script by going into the samples directory and running: **python3 active\_buzzer.py**

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  # http://elecrow.com/
4
5  import RPi.GPIO as GPIO
6  import time
7
8  buzzer_pin = 17
9
10 GPIO.setmode(GPIO.BCM)
11 GPIO.setup(buzzer_pin, GPIO.OUT)
12
13 # Make buzzer sound
14 GPIO.output(buzzer_pin, GPIO.HIGH)
15 time.sleep(0.5)
16 # Stop buzzer sound
17 GPIO.output(buzzer_pin, GPIO.LOW)
18
19 GPIO.cleanup()
```

## Application effect



---

Running the program will turn on the buzzer for 0.5 seconds and then turn it off.

## LESSON 5: PASSIVE BUZZER



### Introduction

In this lesson we'll learn about the passive buzzer.

The passive buzzer is very similar, to the active buzzer. but unlike the active buzzer, the passive buzzer is activated by waves (the active buzzer is activated by current running through it)

In this lesson we'll learn how to use the passive buzzer and run a script as examples to prove that it works.

### Specification:

Working Voltage: 3V/5V

Resistance: 160Ω


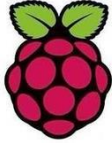




Resonance Frequency: 2KHZ

---

### Hardware required

Below is the hardware we'll need for this lesson.

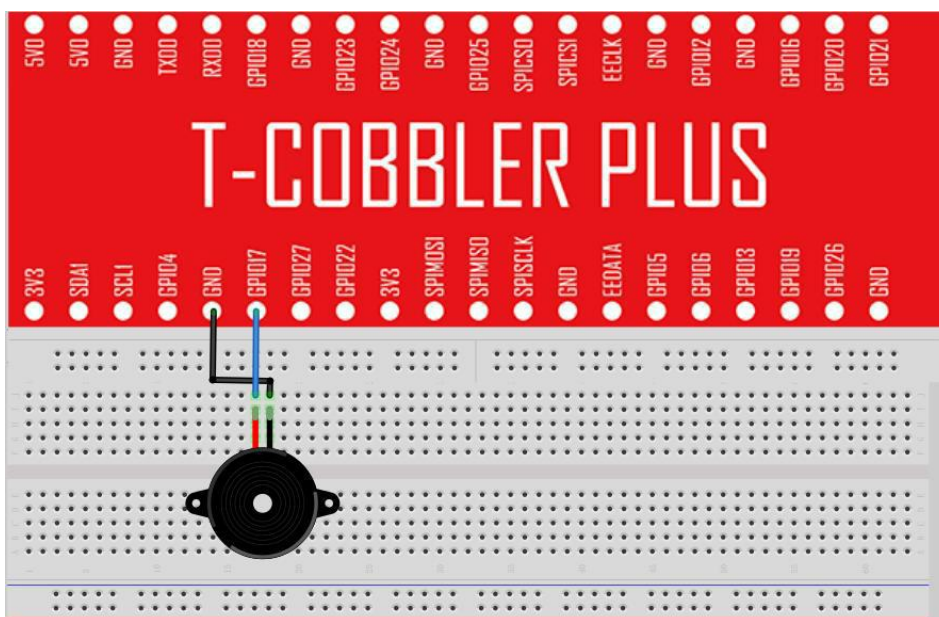
make sure to prepare the hardware and most importantly don't confused the passive buzzer with the active buzzer in the kit.

Material diagram	Material Name	Number (amount)
	Passive Buzzer	1
	Raspberry Pi Board	1
	T-Cubler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper cables	Several

---

### Connection Diagram

As we can see in the diagram, the design is exactly the same as the previous example with the active buzzer. if you just finished the active buzzer lesson, you can just switch the buzzer module without taking out the rest of the pins.



### Connection

Make sure not to mistake the long pin (the positive side) with the short pin (the negative side)

Passive buzzer	Raspberry Pi
Long Pin (+)	GPIO17

Passive buzzer	Raspberry Pi
Short Pin (-)	GND

## Code Overview, Compile and run

The passive buzzer is a little more complicated than the active buzzer.

The code is quite long and every line is commented to explain exactly what's happening but we'll explain here briefly.

The passive buzzer uses frequency to control it so we need to give "pitch" noise to control the vibration, using the passive buzzer in our example we will make a sample song using do re mi .... Do notes.

Compile the script by going into the samples directory and running: **python3 passive\_buzzer.py**

```

1  import RPi.GPIO as GPIO    #import the GPIO library
2  import time                 #import the time library
3
4  class Buzzer(object):
5
6      def __init__(self):
7
8          GPIO.setmode(GPIO.BCM)
9          self.buzzer_pin = 17 #set to GPIO pin 17
10         GPIO.setup(self.buzzer_pin, GPIO.OUT)
11         print("buzzer ready")
12
13     def buzz(self,pitch, duration):    #create the function "buzz" and feed it the pitch and duration)
14
15         if(pitch==0):
16             time.sleep(duration)
17             return
18
19         period = 1.0 / pitch          #in physics, the period (sec/cyc) is the inverse of the frequency (cyc/sec)
20         delay = period / 2             #calculate the time for half of the wave
21         cycles = int(duration * pitch) #the number of waves to produce is the duration times the frequency
22
23         for i in range(cycles):        #start a loop from 0 to the variable "cycles" calculated above
24             GPIO.output(self.buzzer_pin, True)    #set pin 18 to high
25             time.sleep(delay)                    #wait with pin 18 high
26             GPIO.output(self.buzzer_pin, False)   #set pin 18 to low
27             time.sleep(delay)                    #wait with pin 18 low
28
29     def play(self):

```

---

### Application effect

The passive buzzer will play do re mi fa sol la si do (piano notes) from do to do and backwards.

## LESSON 6: RGB LED



### Introduction

Before in our previous lessons we used different types of LEDs (in our kit, we have 5mm LEDs in different colours) the RGB LED is a little bit different.

The RGB stands for Red, Green and Blue. It allows us to control 3 colours and using those colours, generate lots of colours!

We will learn how to activate each of the colours separately and later on being able to combine them together and create different range of colours.

The RGB LED is very useful for any applications, if you need LED for example for indications purposes, there is no need to use red LED for one scenario and blue for another one. We can use RGB LED and change the colours based on our needs!

### Specification

Emitting Light Color: Blue, Red, Green

Size(Approx): 5 x 35mm/ 0.2" x 1.37" (D \* L)

Forward Voltage: 3.0-3.4V

---

Luminous Intensity: 12000-14000mcd

### Pin definition

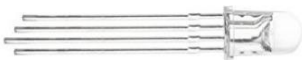

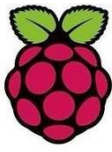





### Hardware required

Below is the hardware we'll need to complete this lesson.

Please note the 220/330ohm resistor, we will need 3 of them.

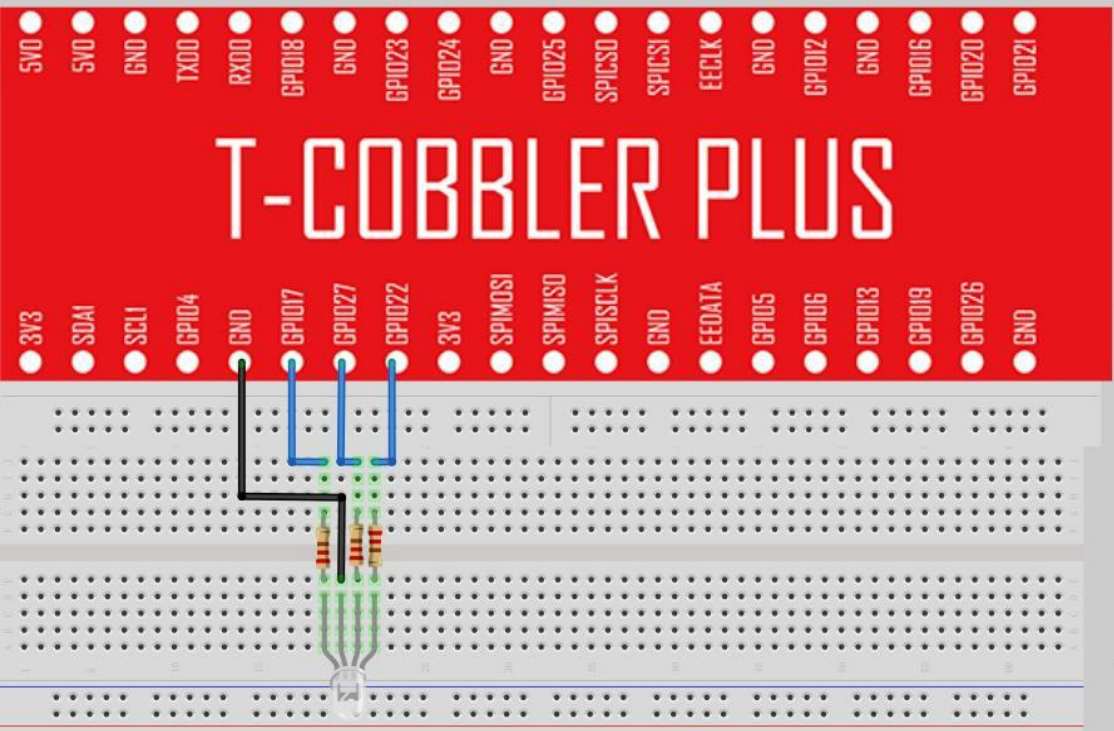
any, either 220 or 330 will work, we can also combine few 220 and few 330ohm resistors, there is no problem with that.

Material diagram	Material Name	Number (amount)
	RGB LED	1
	220/330Ω resistor	3
	Raspberry Pi Board	1
	T-Cubbler Plus	1
	40P GPIO Cable	1
	Breadboard	1

Material diagram	Material Name	Number (amount)
	Jumper wires	Several

Connection Diagram

As we can see in the diagram, we'll need to connect each resistor to each pin of the RGB LED except the ground



pin.we'll connect each pin (each colour) to different GPIO so we could control it separately.

---

## Connection

Please make sure you connect the pins correctly or you might experience unexpected behaviour during the run time of the script. If you see colours that you are not expected to see, try to replace the pins and see if it gets fixed. remember, each colour controlled by different GPIO pin, if you accidentally place the wrong pin on the wrong colour, this will cause the script to show different things from what we programmed it to show!

RGB LED	Raspberry Pi
R	GPIO17
G	GPIO27
B	GPIO22
GND	GND

## Code Overview, Compile and run

The code is quite long but it's all commented out to explain what's going on, we'll go mainly through the main part of the code. We define the object RGB that we've made a class earlier in the code, then we turn on the RGB LED "red" for 0.3 seconds and turn it off. We do the same afterwards for the green and the blue LED.

Compile the script by going into the samples directory and running: **python3 RGB.py**

```
67 def main():
68     # Initialize RGB Class
69     RGB_LED = RGB()
70
71     # Blink Red
72     RGB_LED.turn_on("red")
73     time.sleep(0.3)
74     RGB_LED.turn_off("red")
75     time.sleep(0.3)
76
77     # Blink Green
78     RGB_LED.turn_on("green")
79     time.sleep(0.3)
80     RGB_LED.turn_off("green")
81     time.sleep(0.3)
82
83     # Blink Blue
84     RGB_LED.turn_on("blue")
85     time.sleep(0.3)
86     RGB_LED.turn_off("blue")
87
88     main()
```

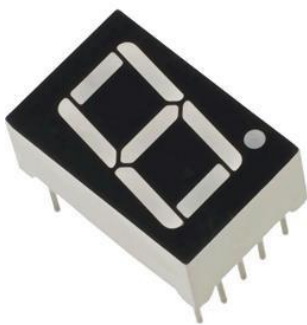


---

## Application effect

Running the script will turn on red LED, green LED and then blue LED with a space of 0.3 seconds between each turn on and turn off.

## LESSON 8: 1 DIGIT 7 SEGMENT DISPLAY

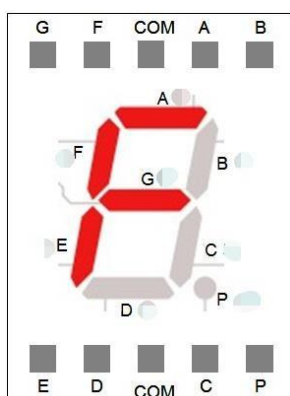


### Introduction

In this lesson we'll learn about the segment display. In our kit we have 2 types of segment display one is single (1 digit) segment display and one is 4 digit. both displays are the same the only difference is the amount of digits we control. In our first lesson we will learn how to control one segment and show a number.

A segment can show a number between 0 and 9.

### Pin Definition



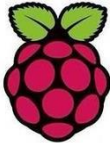






---

As we can see, the segment LED is marked with letters in this pin definition, the real segment in the kit doesn't have any letters on it. Please refer to this definition while wiring the segment LED to make sure we wire it correctly.

### Hardware required

Below is the hardware we'll require for this lesson, 1 digit (don't confused with 4 digit) segment display and 220 or 330 ohm resistors, we'll need 8 of them.

Material diagram	Material Name	Number (amount)
	1 Digit 7 Segment display	1
	220/330Ω Resistors	8
	Raspberry Pi Board	1
	T-Cubbler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

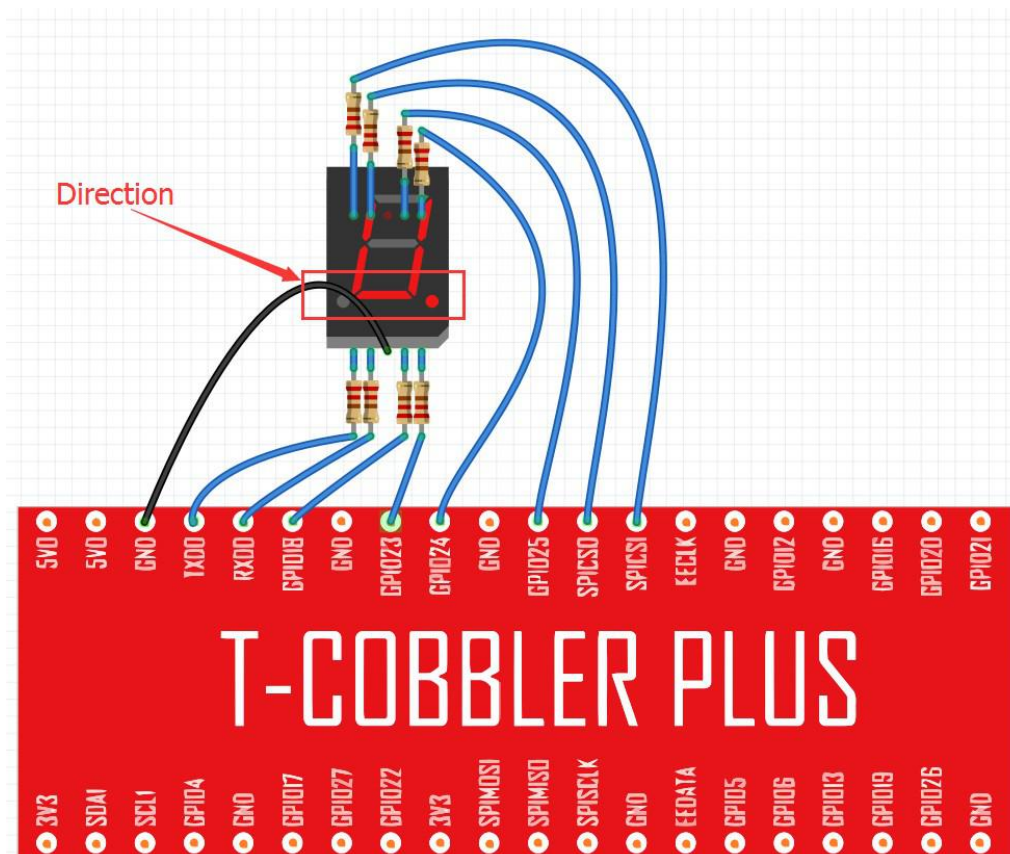
---

## Connection Diagram

As we can see the wiring diagram can be a little bit overwhelming.

Take it slowly, following the letters and the pin definition picture we've showed before to connect it properly. Please note the small dot on the segment LED that tells you which direction the segment LED should be pointed at.

The dot should be pointed downstairs.



---

make sure to not forget to place all the resistors on the pins while connecting them to the GPIO ports.  
All the pins except the GND (ground) pin should have resistor attached to it.

### Connection

Follow the connection carefully to make sure not to wire accidentally some segment in the wrong way, doing so will result in fault view of the numbers when we run the testing script at the end of the lesson.

Segment	Raspberry Pi
A	GPIO25
B	GPIO20
C	GPIO24
D	GPIO21
E	GPIO16
F	GPIO18
G	GPIO12
H	GPIO23
GND	GND

---

## Code Overview, Compile and run

The code is quite long but don't be afraid! This is due to the class code, the main code is actually very easy. What we did is we setup each led on the segment led as a letter from A to H and then we turn them based on the colours that they should be. In our example, we'll loop from 0 to 9 and show the numbers that are currently in the loop so if the loop is on the first round (0) it will show the number zero till the last number which will be 9.

Compile the script by going into the samples directory and running: **python3 1digit\_segment.py**

```
1  import RPi.GPIO as GPIO ## Import GPIO library
2  import time ## Import 'time' library. Allows us to use 'sleep'
3
4  # Segment 1 Digit Numbers Illustration
5  # H Is the dot on the right side of the segment
6  #
7  #   A
8  #   -----
9  #   |           |
10 #   B |         | C
11 #   |   D       |
12 #   -----
13 #   |           |
14 #   E |         | F
15 #   |         | (self.H)
16 #   -----
17 #   G
18
19 class Segment():
20     def __init__(self):
21
22         # set the GPIO mode as GPIO.BCM
23         GPIO.setmode(GPIO.BCM)
24         # set pin names for each of the segment pins
25         self.A = 25
26         self.B = 20
27         self.C = 24
28         self.D = 21
29         self.E = 16
30         self.F = 18
31         self.G = 12
32         self.H = 23
33         # set the segment pins as GPIO.OUT
34         GPIO.setup(self.A, GPIO.OUT)
35         GPIO.setup(self.B, GPIO.OUT)
36         GPIO.setup(self.C, GPIO.OUT)
```

---

### Application effect

Looping through numbers from 0 to 9 and showing the numbers on the segment LED with a space of 1 second between each number we will be showing.

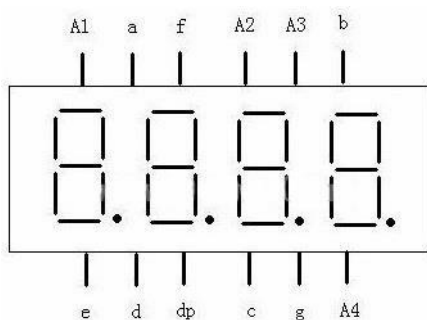
## LESSON 9: 4 DIGIT 7 SEGMENT DISPLAY



### Introduction

In the previous lesson we learned about a single 1 digit segment display, the 4 digit 7 segment display is no different. In this lesson we'll learn about the 4 digit segment, how to turn it on and control it using a simple python script. the script will get the current time from the raspberry pi and display it on the segment LED.

### Pin definition



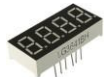

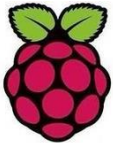




The pin definition of the 4 digit segment is very similar to the 1 digit display.

Please make sure to note the letters on each pin and connect it correctly and accordingly.

### Hardware required

We'll need a 220/330ohm resistor (8 of them, 1 for each pin) for this example.

Please note you can use either 220 or 330 or combine both of them, all options will work perfectly fine.

Material diagram	Material Name	Number (amount)
	Segment Display	1
	220/330Ω Resistors	8
	Raspberry Pi Board	1
	T-Cubbler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

---

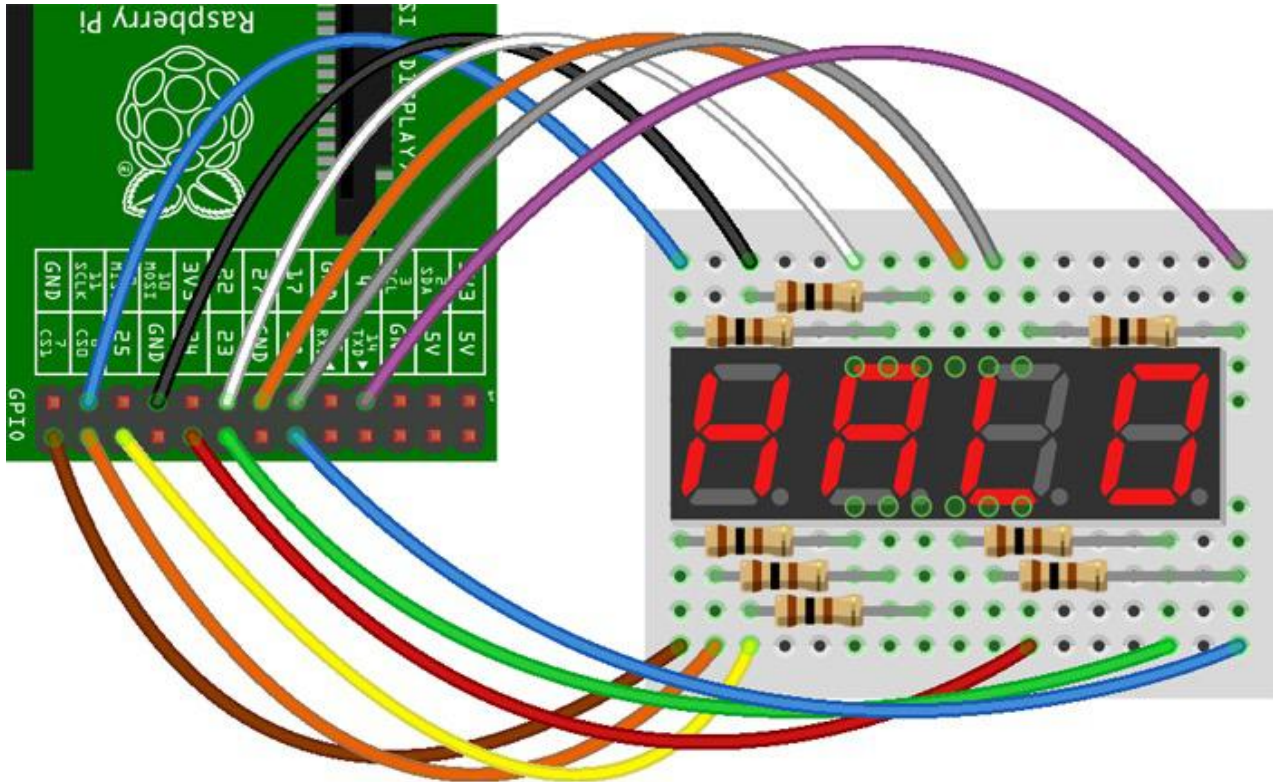
### Connection Diagram

Please refer to the previous pin definition in order to make sure to connect the right pins to the right place.



---

We'll need to add resistor almost to every pin so make sure to connect them properly and put resistor where it



## ***RasPi0 7seg wiring***

should be. The direction of the connection is with the dot at the bottom.

Connection

---

Please refer to the chart below on instructions how to connect the pins, we use GPIO.BCM and not GPIO.BOARD scheme, also make sure to look where you don't need to connect resistors (yellow rows) and where you do need to

<b>Seg / digit</b>	<b>7Seg Pin</b>	<b>Resistor</b>	<b>BCM GPIO #</b>
bot left	1	YES	7
bot centre	2	YES	8
decimal pt	3	YES	25
bot right	4	YES	23
centre centre	5	YES	18
digit 4	6	NO	24
top right	7	YES	4
digit 3	8	NO	17
digit 2	9	NO	27
top left	10	YES	10
top centre	11	YES	11
digit 1	12	NO	22

connect resistors (white rows)

[Code Overview, Compile and run](#)

---

In this example we'll show the current time of our raspberry pi system on the 4 segment LED display. Please note, you'll need to configure the right timezone through the "sudo raspi-config" settings else the time you'll see might be different from the actual time zone you are at right now.

Let's walk through our code and try to understand it: we'll show the hours on the first two columns of the segment LED and the minutes on the second two columns.

We first start a while loop (forever) we get the current time from the time library then we go for each digit (we got total of 4 digits) and loop through total of 7 numbers. And then we output HIGH for the right number that is currently on the clock (based on the current time) and LOW for the time that is wrong (not the current time)

Compile the script by going into the samples directory and running: **python3 4digit\_segment.py**

```
--
38  try:
39      while True:
40          n = time.ctime()[11:13]+time.ctime()[14:16]
41          s = str(n).rjust(4)
42          for digit in range(4):
43              for loop in range(0,7):
44                  GPIO.output(segments[loop], num[s[digit]][loop])
45                  if (int(time.ctime()[18:19])%2 == 0) and (digit == 1):
46                      GPIO.output(25, 1)
47                  else:
48                      GPIO.output(25, 0)
49                  GPIO.output(digits[digit], 0)
50                  time.sleep(0.001)
51                  GPIO.output(digits[digit], 1)
52  finally:
53      GPIO.cleanup()
```

### Application effect

Running the program will show us the current time on the segment clock.

## LESSON 10: HEART-SHAPED DISPLAY



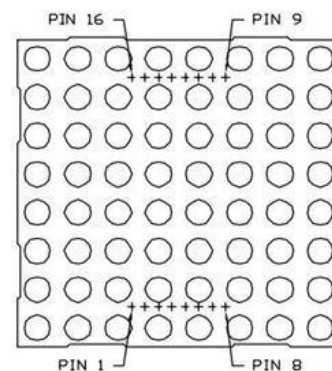
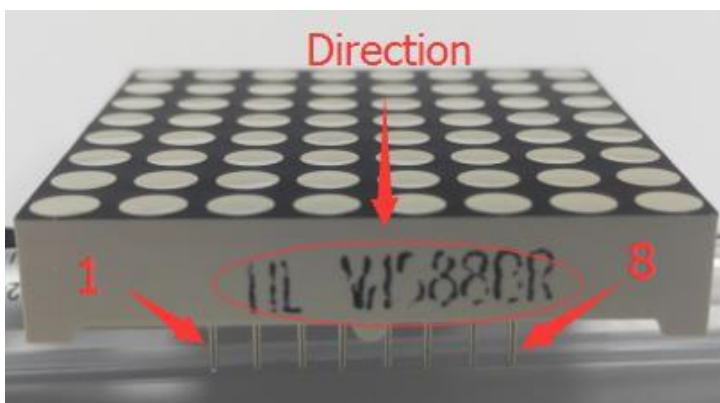
## Introduction

In this lesson we'll learn how to use the matrix display.

The matrix display is very similar to the segment LED just instead of only few LED's to give us numbers the matrix LED is many of many LED's stand in a shape of matrix, this allows us to turn on entire rows, columns or individual LED's on the matrix.

In our example we'll use the matrix LED to make an heart shaped animation.

## Pin definition





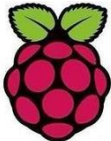




Please note the direction of the matrix LED: The black text should be in the front.

---

## Hardware required

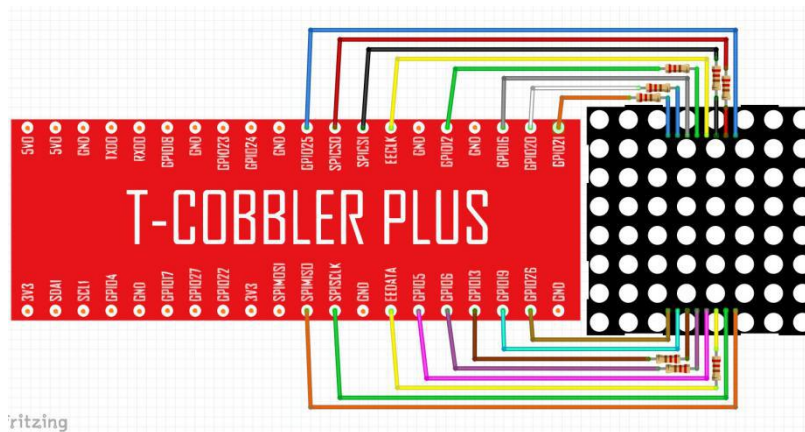
For this example just as the previous one we'll need 8 220 or 330ohm resistors, combining 220 and 330 or using just one type of them will work.

Please make sure to note the right connection of the resistors in order for this example to work correctly.

Material diagram	Material Name	Number (amount)
	LED Matrix	1
	220/330Ω Resistors	8
	Raspberry Pi Board	1
	T-Cubler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

## Connection Diagram

There are many pins that we need to connect and to some of them we need to attach resistors, please look carefully



and make sure to follow the diagram.

## Connection

LED MATRIX	Raspberry Pi
pin1	GPIO26
pin2	GPIO19
pin3	GPIO13
pin4	GPIO6
pin5	GPIO5
pin6	EE-DATA
pin7	SPI-SCLK
pin8	SPI-MISO
pin9	GPIO25
pin10	SPI-CS0
pin11	SPI-CS1
pin12	EE-CLK



LED MATRIX	Raspberry Pi
pin13	GPIO12
pin14	GPIO16
pin15	GPIO20
pin16	GPIO21

## Code Overview, Compile and run

In this example we'll active the matrix LED in heart shape, we create an array of LED's and then decide which LED will be on and which one will be off creating a heart shape by the numbers 1 and 0.

Then we go for each place in the array of numbers and send turn on the LED that states 1 (on) and leave the LED off that states 0 (off)

Compile the script by going into the samples directory and running: **python3 matrix\_heart.py**

```

11  ### Initialize an SPI connection using BCM-mode pins 21, 20, and 16
12  max7219 = spi.SPI(clk=21, cs=20, mosi=16, miso=None)
13
14  ### Zero out all registers
15  for cmd in range(16):
16      packet = cmd << 8
17      max7219.put(packet,12)
18
19  ### Enable all columns via the scan limit register
20  max7219.put(int("101100000111",2),12)
21
22  ### Disable shutdown mode
23  max7219.put(int("110000000001",2),12)
24
25  ### Define the values for each column that gives us a heart shape
26  heart_shape = np.array( [
27      [ 0, 0, 0, 0, 0, 0, 0, 0 ],
28      [ 0, 1, 1, 0, 0, 1, 1, 0 ],
29      [ 1, 1, 1, 1, 1, 1, 1, 1 ],
30      [ 1, 1, 1, 1, 1, 1, 1, 1 ],
31      [ 0, 1, 1, 1, 1, 1, 1, 0 ],
32      [ 0, 0, 1, 1, 1, 1, 0, 0 ],
33      [ 0, 0, 0, 1, 1, 0, 0, 0 ],
34      [ 0, 0, 0, 0, 0, 0, 0, 0 ],
35  ] )
36
37  ### Set each column according to our heart shape matrix
38  for col in range(8):
39      cmd = (col+1) << 8
40      values = 0
41      ### Without flipud, the heart would be upside-down since the LED matrix
42      ### defines 0,0 as the bottom left corner while numpy puts it at the top
43      ### left.
44      for i in np.flipud(heart_shape[:,col]):
45          values <=<= 1

```

---

### Application effect

The matrix will turn on in a shape of heart, leaving certain LED's on and certain LED's off.

## LESSON 11: 9G SERVO



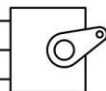
### Introduction

In this lesson we'll learn about the 9G servo.

The 9G servo is a very useful module allows us to use a servo in certain degree, that allows us to attach certain materials such as wire, cardboard or 3D printed parts in order to perform certain movements or actions in this lesson we'll learn how to move the servo around by giving it the degree we want it to move to.

### Pin Definition

PWM=Orange (⏏)  
Vcc = Red (+)  
Ground=Brown (-)



The pin definition of the servo is very simple.

We have total of 3 wires: red, orange and brown (or black)

Red and brown indicate positive and negative while the orange is the data pin (PWN) that allows us to move the servo itself.


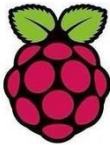






---

### Hardware required

The hardware is pretty straight forward, we don't need any resistors or special components on this one.

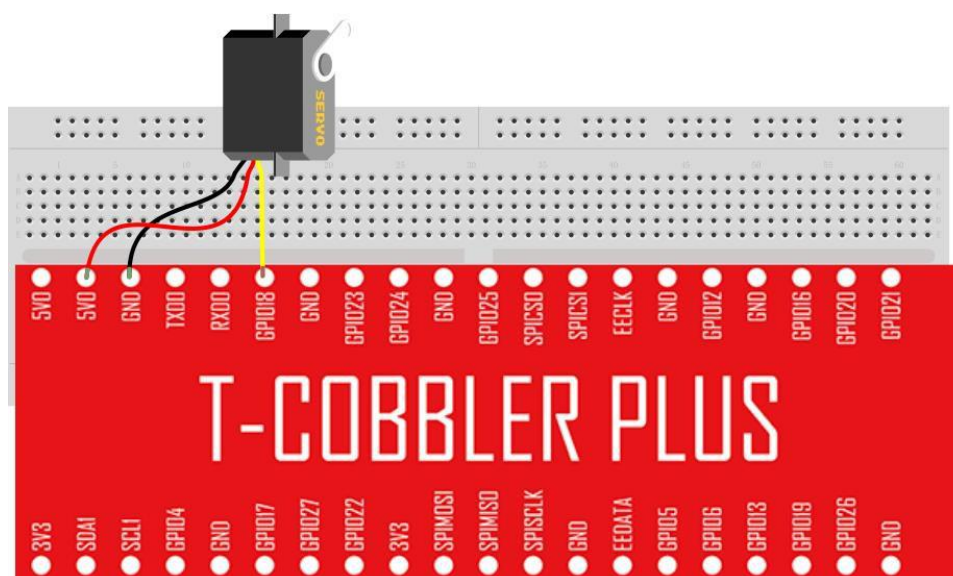
We'll use the servo and connect it to the breadboard using the jumpers, follow the diagram to make sure to connect the servo to the correct GPIO pin.

Material diagram	Material Name	Number (amount)
	Servo Motor	1
	Raspberry Pi Board	1
	T-Cubbler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

---

## Connection Diagram

Red wire for 5V positive black is for the negative, the orange wire which we'll use for PWN commands will be connected directly to GPIO18.



## Connection

Servo	Raspberry Pi
Yellow wire	GPIO18
Red wire	5V
Black wire	GND

---

## Code Overview, Compile and run

What we are going to do with our 9G servo is quite simple. We will first initiate the class “sg90” that already have everything pre-coded and prepared for us with the parameter “0” as the initial direction.

Then will turn it left by setting the direction to 100 and speed of 80 (out of 100) to turn it to the other direction to the right, we will turn it to -100 (opposite of 100) with the same speed of 80.

It will continue going in forever loop till pressed CTRL+C or CTRL+Z to quit the software.

Compile the script by going into the samples directory and running: **python3 servo.py**

```
46 def main():
47
48     s = sg90(0)
49
50     try:
51         while True:
52             print("Turn left ...")
53             s.setdirection( 100, 80 )
54             time.sleep(0.5)
55             print("Turn right ...")
56             s.setdirection( -100, 80 )
57             time.sleep(0.5)
58     except KeyboardInterrupt:
59         s.cleanup()
60
61 if __name__ == "__main__":
62     main()
```

---

## Application effect

The servo will turn left and right at specific speed of 80% out of it's capacity forever till the user press CTRL+C or CTRL+Z to terminate the script.

---

## LESSON 12: STEP MOTOR



### Introduction

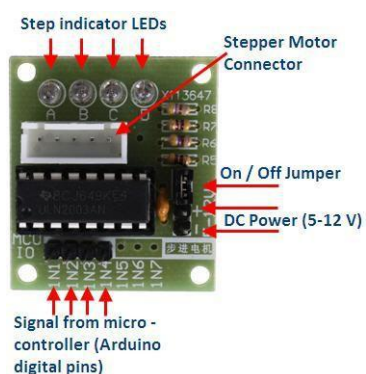
In this lesson we'll learn about the step motor, step motor similar to the servo is another kind of "motor" but unlike the servo it can turn up to 360 degrees and keep turning around to both direction.

The step motor is controlled by a special "driver", without this driver we are unable to control the step motor directly from the raspberry pi.

The step motor is very very useful specially for precise required applications such as robotics, 3D printers and so on ... where we need a precise movements based on number of "steps".

In our example we'll learn how to control the step motor by giving it the amount of steps we want it to turn around.



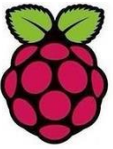




### Pin definition



---

### Hardware required

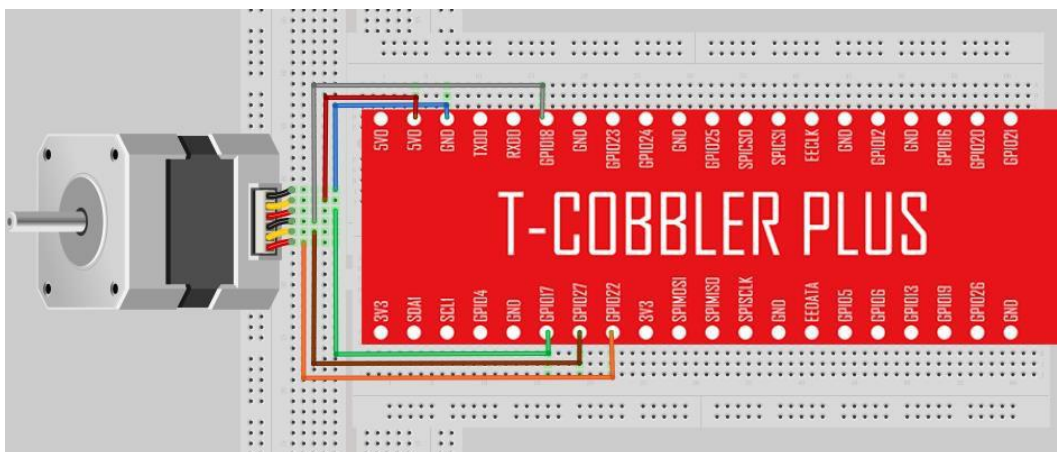
For this example we'll need 2 special components, the first one obviously will be the step motor but the second one will be the ULN2003 step motor driver board. As we mentioned before, without this driver board we'll be unable to control the step motor directly.

Material diagram	Material Name	Number (amount)
	Step Motor	1
	ULN2003 Step motor driver board	1
	Raspberry Pi Board	1
	T-Cubler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

---

## Connection Diagram

Please note the diagram is not 100% correct. We'll need to connect the step motor to the driver first and then follow the diagram to connect the driver to the raspberry pi or T-Cobbler plus.



## Connection

We have total of 4 OUTPUT GPIO pins (IN1,2,3, and 4) that we'll use to control the steps of the step motor.

Step motor	Raspberry Pi
IN1	GPIO12
IN2	GPIO16
IN3	GPIO20
IN4	GPIO21
5V	5V
GND	GND

---

## Code Overview, Compile and run

Let's take a look at our code, the main code will initialise the step motor object and then we'll run a demo of different type: first demo will be to turn 1 step, then 20 steps, then quarter turn by degree and close the motor. As we can see, we have 2 type of function:

1. turnSteps which will turn the motor by amount of stops
2. turnDegrees which we can state degrees such as 90,180,360 and it will turn it around.

We can use both functions to archive the same result in different ways.

Compile the script by going into the samples directory and running: **python3 stepper.py**

```
121  def main():
122
123      print("moving started")
124      motor = Stepmotor()
125      print("One Step")
126      motor.turnSteps(1)
127      time.sleep(0.5)
128      print("20 Steps")
129      motor.turnSteps(20)
130      time.sleep(0.5)
131      print("quarter turn")
132      motor.turnDegrees(90)
133      print("moving stopped")
134      motor.close()
135
136  if __name__ == "__main__":
137      main()
```

---

### Application effect

Running the program will cause the step motor to turn one step, then 20 steps, then quarter turn and at the end stop. You can change the interval of the step motor to make it move slower or faster.

## LESSON 13: ULTRASONIC RANGING



### Introduction

In this lesson we'll learn about the ultrasonic ranging sensor.

The ultrasonic is probably most favourable sensor in the market, it opens loads of opportunities to develop applications in many different areas from Robotics to IoT and even smart city!

The Ultrasonic ranging sensor is able to sense the range from point 0 (the sensor location) to the object standing in front of it, it can sense up to 5 meters ahead.

In this lesson we'll learn how to control the ultrasonic sensor and measure the distance of to the object in front of our sensor.


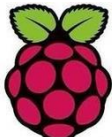


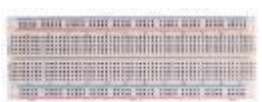



---

### Hardware required

The main component we'll need is the HCSR04 ultrasonic sensor, it's difficult not to recognise it as it looks like 2 robotic eyes.

The sensor have 2 different type of connections which we will explain on the next page, please pay attention to it.

Material diagram	Material Name	Number (amount)
	HCSR04 Ultrasonic sensor	1
	Raspberry Pi Board	1
	T-Cubbler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

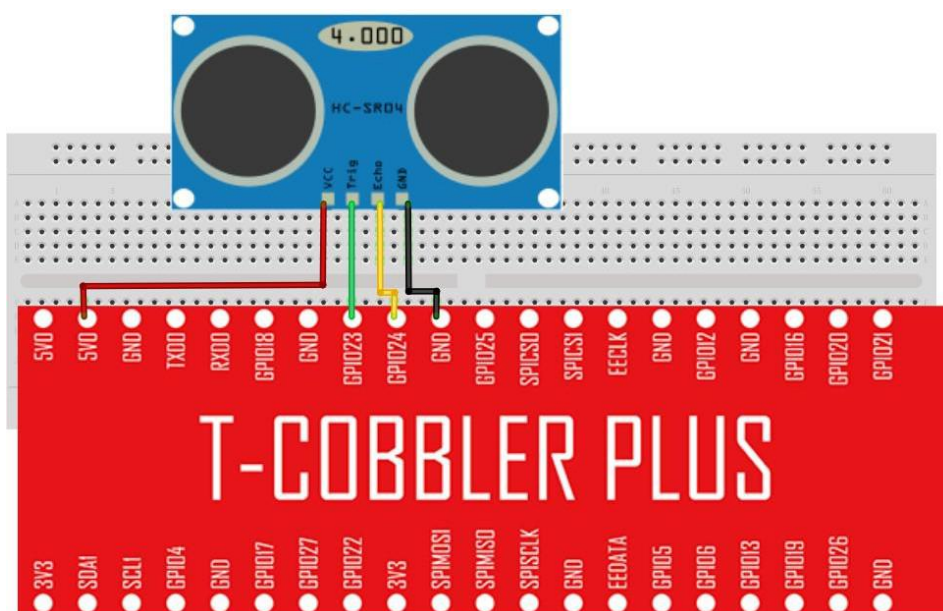
---

## Connection Diagram

In the connection diagram we have the 2 basic pins one is the negative and the other is the positive (black and red wires)

The other wires are the dat wires called TRIG and ECHO we'll use them to send a pulse and to get back response so we can determine the distance by the time it takes for the pulse to return back to the sensor.

Make sure to connect them properly, TRIG goes to GPIO23 and ECHO goes to GPIO24.



## Connection

Component	Raspberry Pi
-----------	--------------

---

Component	Raspberry Pi
VCC	5V
TRIG	GPIO23
ECHO	GPIO24
GND	GND

### Code Overview, Compile and run

The distance sensor code is very straight forward.

We will measure the pulse and how long it takes it to come back so we will know the distance from the sensor to the object in front of it.

We will print the distance in cm but later on you can change it to any unit you feel comfortable with.

Compile the script by going into the samples directory and running: **python3 distance.py**

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  # Author : www.modmypi.com
4  # Link: https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi
5
6  import RPi.GPIO as GPIO
7  import time
8
9  # we set GPIO mode as GPIO.BCM
10 GPIO.setmode(GPIO.BCM)
11
12 # we set the pins for the TRIG and ECHO
13 TRIG = 23
14 ECHO = 24
15
16 # measuring distance
17 print("Distance Measurement In Progress")
18 GPIO.setup(TRIG,GPIO.OUT)
19 GPIO.setup(ECHO,GPIO.IN)
20
21 GPIO.output(TRIG, False)
22 print("Waiting For Sensor To Settle")
23 time.sleep(2)
24
25 GPIO.output(TRIG, True)
26 time.sleep(0.00001)
27 GPIO.output(TRIG, False)
28
29 while GPIO.input(ECHO)==0:
30     pulse_start = time.time()
31
32 while GPIO.input(ECHO)==1:
33     pulse_end = time.time()
34
35 pulse_duration = pulse_end - pulse_start
36
37 distance = pulse_duration * 17150
38
```

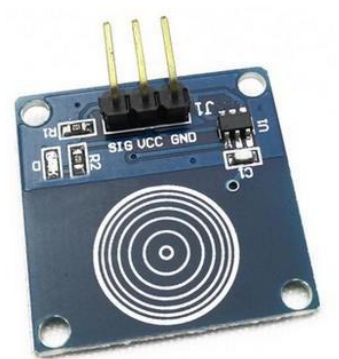
---

---

### Application effect

The distance sensor will measure the distance and afterwards print in the console the distance that it found in cm.

## LESSON 14: TOUCH LAMP



### Introduction

In this lesson we'll learn about the touch sensor.

The touch sensor is very familiar to the button we used in the previous lessons just except a physical key press there's capacitive touch surface that we can touch and detect it.

Most of the products today use capacitive touch instead of actual button so using the touch sensor makes completely sense for the next projects you are willing to work with.

The touch sensor by itself is a little bit boring so we'll try to do something fun with it.

We'll connect our RGB LED together with the touch sensor and once a touch is detected we'll turn on the RGB LED and shine some bright light!

---




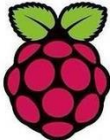

### Hardware required




The main hardware parts we'll need for this lesson is the touch sensor and the RGB LED we used earlier in previous lesson.

The touch sensor is very easy to distinguish, you'll see there is some kind of "finger print" painting on it, the finger print stands for the finger that will touch the sensor to detect the touch.

We'll also need 3 220/330ohm resistors to go with the RGB LED (not for the touch sensor)

You can use either 220ohm or 330ohm or combine them both, all will work just fine.

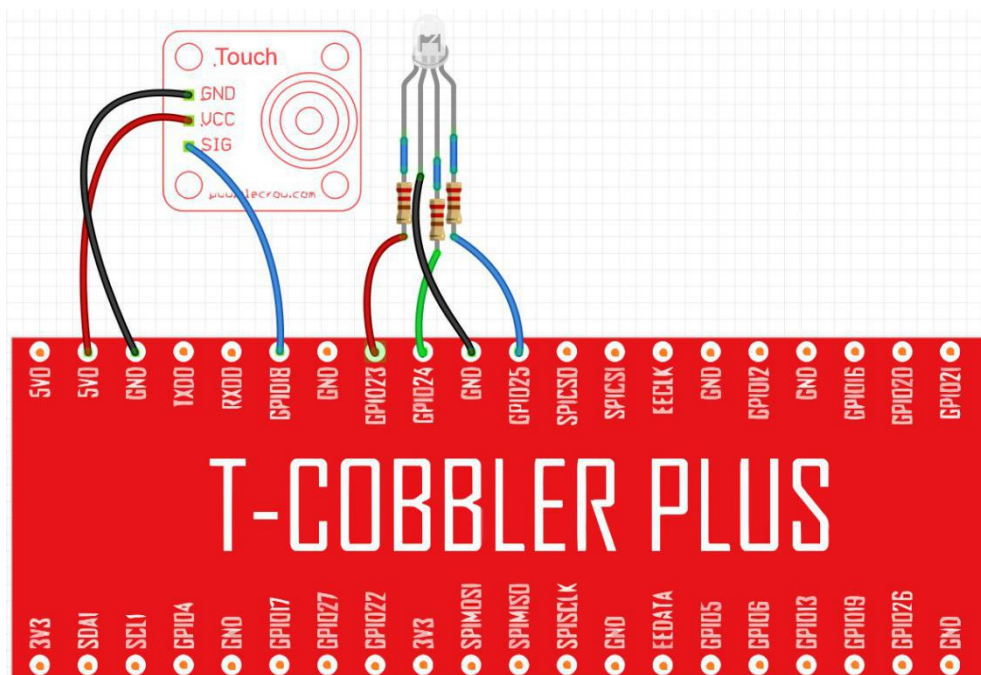
Material diagram	Material Name	Number (amount)
	Touch Sensor	1
	RGB LED	1
	220/330Ω resistor	3
	Raspberry Pi Board	1
	T-Cubler Plus	1

Material diagram	Material Name	Number (amount)
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

### Connection Diagram

In the connection diagram we can see that we are going to connect the RGB LED just as we did in previous lessons to GPIO17,27 and 22.

The touch sensor have 3 different pins, one is the positive (VCC) another is negative (GND) and GPIO pin that we will configure as GPIO.IN to receive a finger touch over the sensor.



---

## Connection

RGB LED	Raspberry Pi
R	GPIO17
G	GPIO27
B	GPIO22
GND	GND

Touch sensor	Raspberry Pi
GND	GND
VCC	VCC
SIG	GPIO18

## Code Overview, Compile and run

In our example we'll use both RGB LED and touch sensor, when pressing the touch sensor, we will randomly choose one colour: red green or blue. Then we will turn the LED of the randomly chosen colour. If we stop touching the touch sensor, the LED will turn off.

Compile the script by going into the samples directory and running: **python3 touch.py**

```
67 def main():
68     # Initialize RGB Class
69     RGB_LED = RGB()
70
71     # define touch pin
72     touch_pin = 18
73
74     # set GPIO pin to INPUT
75     GPIO.setup(touch_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
76
77     colors = ["red","green","blue"]
78
79     try:
80         # select random color to turn on
81         random_color = random.choice(colors)
82         while True:
83             # check if touch detected
84             if(GPIO.input(touch_pin)):
85                 # select random color to turn on
86                 RGB_LED.turn_on(random_color)
87             else:
88                 RGB_LED.turn_off(random_color)
89                 random_color = random.choice(colors)
```

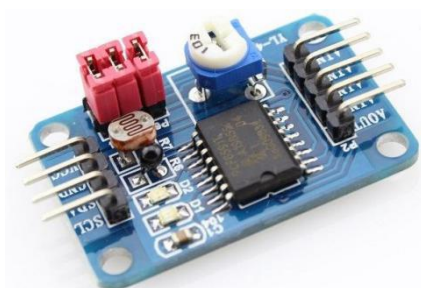
---

---

### Application effect

When touching the touch sensor, a colour will be chosen randomly by the script, either red, green or blue. Then the LED will turn on. If the touch sensor is released, the LED will turn off.

## LESSON 15: PCF8591 MODULE



### Introduction

The PCF8591 module is an 8-bit A/D Converter & D/A Converter PCF8591 with four analog inputs, one analog output and a serial I2C-bus interface.



---

The PCF8591 module features I2C pin header on one side, and I2C connector on the opposite side. Hence, it's more flexible to connect the board to your development system. The board also supports I2C cascading, allowing the use of multi module connected to the I2C bus at the same time by connecting the pin header and connector.

Please note that we will use the PCF8591 module for many of our lessons from this point on.


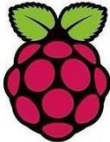
We'll connect modules such as light, flamed thermistor to get their value that is changing based on the environment.





The first lesson regarding the PCF8591 is just to show how to use it and connect it properly, if you'd like to continue with the next lessons feel free to leave the PCF8591 in place.

### Hardware required

For this lesson we'll need mostly the PCF8591 module.

No resistors or other parts are required. But as we stated earlier on, we'll combine the PCF8591 module with other components later on to get their data and changing values.

Material diagram	Material Name	Number (amount)
	PCF8591	1
	Raspberry Pi Board	1

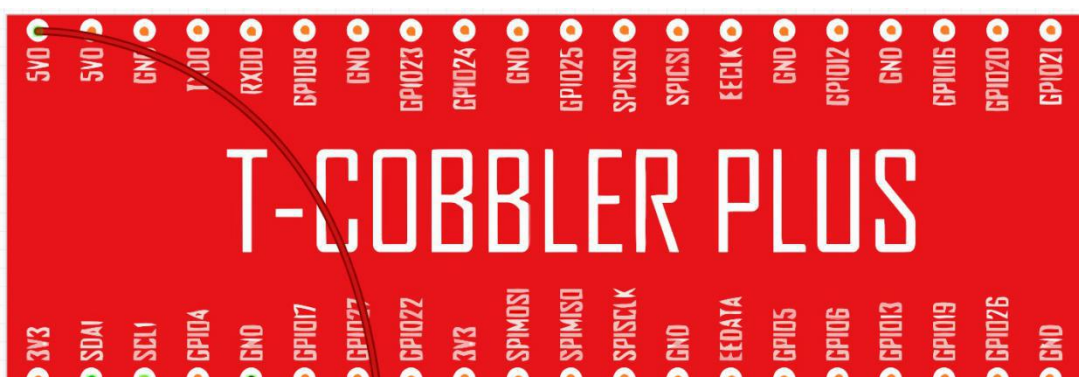
Material diagram	Material Name	Number (amount)
	T-Cubbler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

### Connection Diagram

The connection diagram is pretty straight forward, the PCF8591 have to sides left side and right side, we'll use the left side to connect it to our Raspberry Pi and the right side later on to connect with components and sensor.

The left side can be treated as output and the right side as input.

The connection is through the SDA and SCL pins, the rest are GND and VCC. Make sure to connect the VCC to the 5V pin.



---

## Connection

PCF8591	Raspberry Pi
VCC	5V
GND	GND
SDA	SDA1
SCL	SCL1

## Code Overview, Compile and run

The code is quite simple and we won't be changing it for the next couple of lessons, the same code can apply when using the components as well.

Using the smbus library we connect to the port A0 address which we will be using in the next few lessons, then we get the value and print it out, pretty straight forward. When no components are used the value won't change much but when the components will be attached the value will be changed based on the component.

Compile the script by going into the samples directory and running: **python3 pcf8591.py**

```
1  #!/usr/bin/python
2  # -*- coding:utf-8 -*-
3  # www.elecrow.com
4  import smbus
5  import time
6
7
8  # NOTE: the PCF8591 script can be used both with photoresistance example and flame sensor example
9  # please follow up the instructions on the PCF8591 schematic to connect the right sensors in the right way.
10
11  address = 0x48
12
13  A0 = 0x40
14  A1 = 0x41
15  A2 = 0x42
16  A3 = 0x43
17
18  bus = smbus.SMBus(1)
19
20  while True:
21      bus.write_byte(address,A0)
22      value = bus.read_byte(address)
23      print("Value: %s" % value)
24      time.sleep(0.1)
25
26
```

---

### Application effect

Running the program will result in value change based on the PCF8591 itself. No component is attached yet.

## LESSON 16: FLAME SENSOR



### Introduction

The flame sensor can be used to detect fire or other wavelength at 760 nm ~ 1100 nm light.

In the fire-fighting robot game, the flame plays an important role in the probe, which can be used as the robot's eyes to find fire source or football.

It can make use of fire-fighting robots, soccer robots.




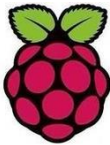



The flame sensor's operating temperature is -25 degrees Celsius to 85 degrees Celsius.


This will be the first sensor we'll be connecting to the PCF8591 module in order to get the data, the flame sensor cannot be connect alone just like that, we'll use the PCF8591 help to figure it out.

### Hardware required

The main component will be the PCF8591 and the flame sensor. While connecting the flame sensor we'll also need to use a 10K ohm resistor, please make sure it's 10K ohm and not 10 ohm.

The resistor will go directly to the flame sensor and then to the input pin of the PCF8591 module as will be shown in the next page.

Material diagram	Material Name	Number (amount)
	Flame sensor	1
	10KΩ resistor	1
	PCF8591	1
	Raspberry Pi Board	1
	T-Cubler Plus	1
	40P GPIO Cable	1
	Breadboard	1

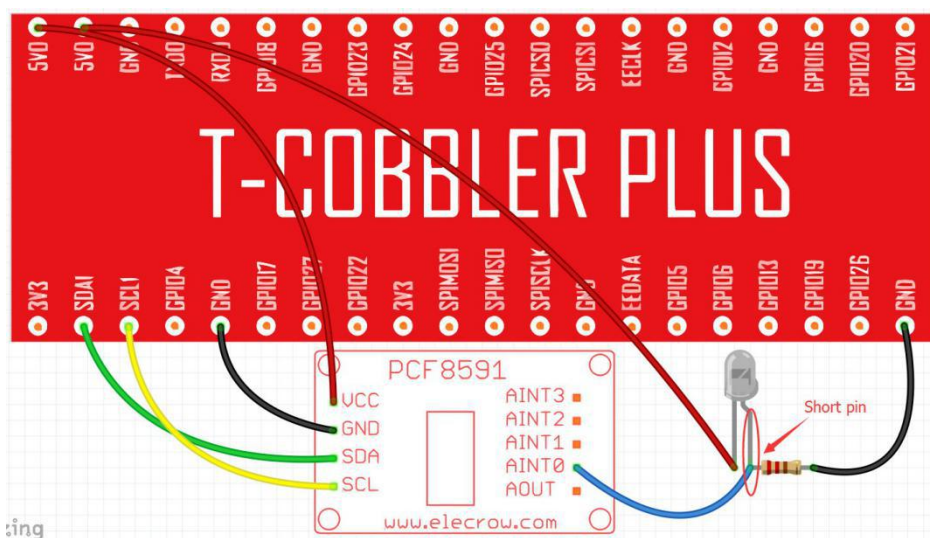
Material diagram	Material Name	Number (amount)
	Jumper wires	Several

### Connection Diagram

The connection is simple, we'll need to connect the short pin of the flame sensor (the negative pin) to the resistor and to the GND pin as well as to AINT0 pin on the PCF8591 module.

The long pin of flame sensor will go directly to 5V on the raspberry pi (there are 2 5V pins so you could use one for the PCF8591 and the other one for the flame sensor)

If you forgot how to connect the PCF8591 module to the raspberry pi, refer to the previous lesson.



### Connection

---

Flame sensor	Raspberry Pi
Short Pin	5V
Long Pin	AIN0

PCF8591	Raspberry Pi
VCC	5V
GND	GND
SDA	SDA1
SCL	SCL1

## Code Overview, Compile and run

We use the same script that we used earlier, by attaching the flame sensor, by introducing flame to it either by lighter or candle the value will change. We can use that value to detect if there is currently flame or no flame at all.

Compile the script by going into the samples directory and running: **python3 pcf8591.py**

```

1  #!/usr/bin/python
2  # -*- coding:utf-8 -*-
3  # www.elecrow.com
4  import smbus
5  import time
6
7
8  # NOTE: the PCF8591 script can be used both with photoresistance example and flame sensor example
9  # please follow up the instructions on the PCF8591 schematic to connect the right sensors in the right way.
10
11  address = 0x48
12
13  A0 = 0x40
14  A1 = 0x41
15  A2 = 0x42
16  A3 = 0x43
17
18  bus = smbus.SMBus(1)
19
20  while True:
21      bus.write_byte(address,A0)
22      value = bus.read_byte(address)
23      print("Value: %s" % value)
24      time.sleep(0.1)
25
26

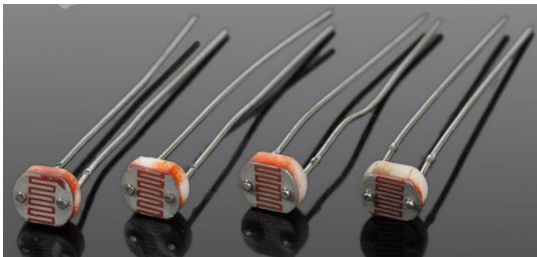
```

---

### Application effect

Running the program will show a value that changes based on the flame sensor, if flame is introduced the value will be higher. If no flame the value will be lower.

## LESSON 17: PHOTO RESISTANCE SENSOR



### Introduction

As the resistance of the sensor varies depending on the amount of light it is exposed to, the output voltage changes with the light intensity. It can be used to trigger other modules.

This will be our third component that we will connect to the PCF8591 module, should be easy as we connected the first one earlier in our lessons.






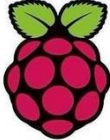

---

### Hardware required




The main hardware component will be the photo resistance sensor, it's small so it's a little hard to notice.

We'll need 10K ohm resistor as well to connect the photo resistance sensor to the PCF8591 module, make sure you take the 10K Ohm and not the 10 Ohm resistor accidentally.

We'll use the same connection for the PCF8591 module as we used in our previous lessons.

Material diagram	Material Name	Number (amount)
	Photo-resistance sensor	1
	10KΩ resistor	1
	PCF8591	1
	Raspberry Pi Board	1
	T-Cubler Plus	1

---

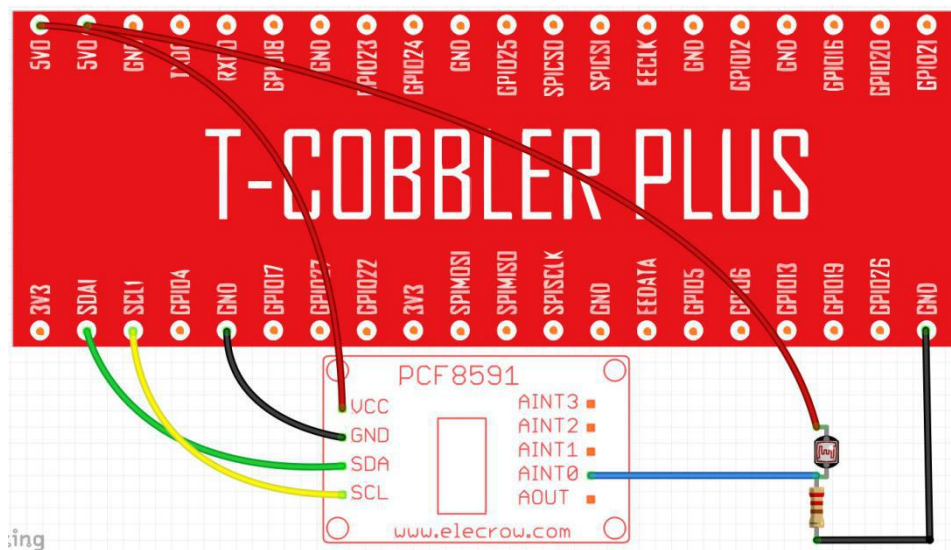
Material diagram	Material Name	Number (amount)
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

### Connection Diagram

The diagram is very similar to the previous we've made. Let's take a look over the photo resistance pins, it doesn't have a short or long one.

We'll need to connect one pin to the 10K ohm resistor and then to the GND as well as to the AINT0 pin on the PCF8591 module. The other pin, the positive side we choose goes directly to 5V.

The PCF8591 module connections stays exactly the same.



## Connection

Photo-resistance sensor	Raspberry Pi
Pin 1	5V
Pin 2	AINT0

PCF8591	Raspberry Pi
VCC	5V
GND	GND
SDA	SDA1
SCL	SCL1

## Code Overview, Compile and run

---

This time after attaching the light sensor to the PCF module the value will change based on the amount of light currently in the room, try to switch of or on the light or introduce different source of light such as your mobile phone flash light to see the value changing.

Compile the script by going into the samples directory and running: **python3 pcf8591.py**

```
1  #!/usr/bin/python
2  # -*- coding:utf-8 -*-
3  # www.elecrow.com
4  import smbus
5  import time
6
7
8  # NOTE: the PCF8591 script can be used both with photoresistance example and flame sensor example
9  # please follow up the instructions on the PCF8591 schematic to connect the right sensors in the right way.
10
11  address = 0x48
12
13  A0 = 0x40
14  A1 = 0x41
15  A2 = 0x42
16  A3 = 0x43
17
18  bus = smbus.SMBus(1)
19
20  while True:
21      bus.write_byte(address,A0)
22      value = bus.read_byte(address)
23      print("Value: %s" % value)
24      time.sleep(0.1)
25
26
```

### Application effect

Based on the light the value will be changing, the higher the value the higher amount of light, Lower amount of light will result in lower value to be presented.

## LESSON 18: THERMISTOR SENSOR



### Introduction

The resistance of a thermistor increases when the ambient temperature decreases, so the RPI can detect the voltage and thus calculate the current temperature. The detection range of this sensor is between -40 to 125 degrees Celsius with an accuracy of  $\pm 1.5^{\circ}\text{C}$ .




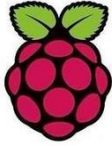




Same as our previous lessons we'll use the PCF8591 module to connect the thermistor to the Raspberry Pi and calculate the temperature.

## Hardware required

The main component in this lesson is the thermistor sensor, it looks like LED with a small black pin at the end of it. We'll also need the 10K ohm resistor from the previous lessons to connect to the thermistor and GND pin.

Please make sure not to confuse 10K ohm resistor with 10 ohm resistor.

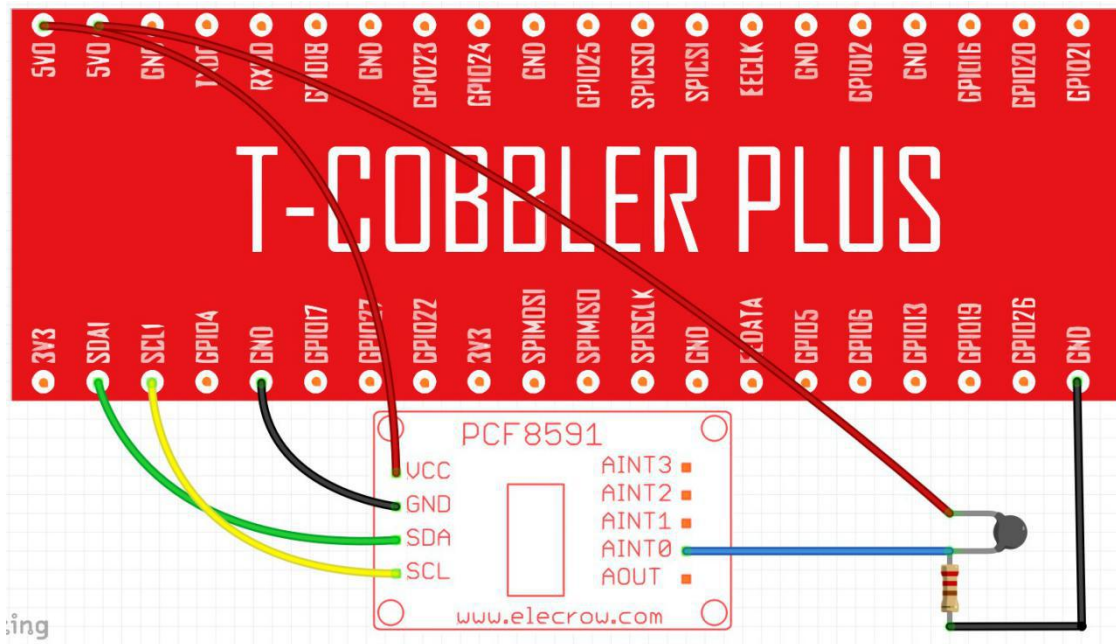
And off course, our buddy the PCF8591 that we used and will use in this lesson to get the value from the thermistor sensor.

Material diagram	Material Name	Number (amount)
	Thermistor sensor	1
	10KΩ resistor	1
	PCF8591	1
	Raspberry Pi Board	1
	T-Cubbler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

## Connection Diagram

As the diagram states, we'll leave the PCF8591 as we did before. The thermistor doesn't have specific pins so one pin we connect to the GND through the 10K ohm resistor as well as to AINT0 at the PCF8591 module.

The other pin we connect directly to 5V pin, there are 2 5V pins so there is no need in pushing them together at the same spot.



## Connection

Thermistor sensor	Raspberry Pi
Pin 1	5V
Pin 2	AINT0

PCF8591	Raspberry Pi
VCC	5V
GND	GND
SDA	SDA1
SCL	SCL1

---

## Code Overview, Compile and run

As in our previous examples, the PCF8591 module will change the value based on the thermistor sensor.

Compile the script by going into the samples directory and running: **python3 pcf8591.py**

```
1  #!/usr/bin/python
2  # -*- coding:utf-8 -*-
3  # www.elecrow.com
4  import smbus
5  import time
6
7
8  # NOTE: the PCF8591 script can be used both with photoresistance example and flame sensor example
9  # please follow up the instructions on the PCF8591 schematic to connect the right sensors in the right way.
10
11  address = 0x48
12
13  A0 = 0x40
14  A1 = 0x41
15  A2 = 0x42
16  A3 = 0x43
17
18  bus = smbus.SMBus(1)
19
20  while True:
21      bus.write_byte(address,A0)
22      value = bus.read_byte(address)
23      print("Value: %s" % value)
24      time.sleep(0.1)
25
26
```

## Application effect

Running the program will show different value based on the temperature of the thermistor sensor, the higher temperature the higher the value.



---

## LESSON 19: POTENTIOMETER



### Introduction

In this example, we use a potentiometer, we read its value using one analog input of an RPI board and we change the blink rate of the built-in LED accordingly. The resistor's analog value is read as a voltage because this is how the analog inputs work.



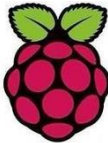




For this lesson as the previous lessons we'll use the PCF8591 module in order to read the values from the potentiometer.

---

### Hardware required

The main hardware we'll be using is the potentiometer, unlike the previous examples for the potentiometer we won't be in need in using 10K ohm resistors due to the resistance of the potentiometer itself.

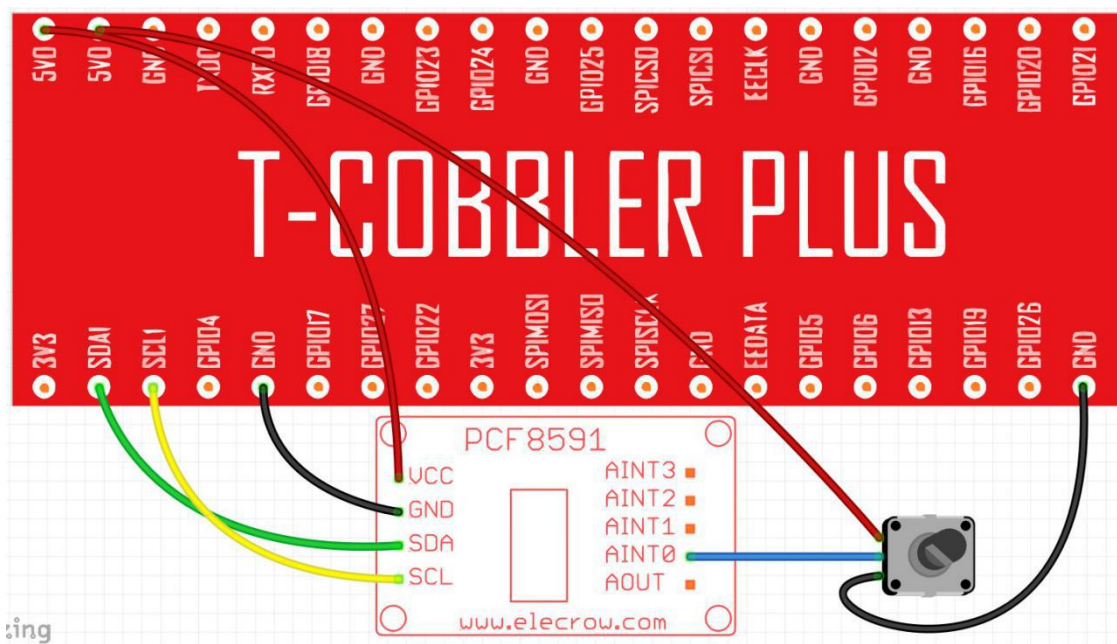
And ofcourse, we'll need to use the PCF8591 module as well to read the value out of the potentiometer.

Material diagram	Material Name	Number (amount)
	Potentiometer	1
	PCF8591	1
	Raspberry Pi Board	1
	T-Cubbler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

## Connection Diagram

The diagram stays the same as previous lessons except we don't connect the resistor right now as we have no use in it for the potentiometer.

The potentiometer have 3 pins, GND, VCC and AINT0 make sure to connect the correct pins to the right places.



## Connection

Component	Raspberry Pi
Up Pin	5V
Mid Pin	AINT0

---

PCF8591	Raspberry Pi
VCC	5V
GND	GND
SDA	SDA1
SCL	SCL1

## Code Overview, Compile and run

This time as we use the potentiometer it will require us to do some manual action, But turning the potentiometer either right or left it will turn the value, higher or lower. By doing so we can control multiple things such as brightness of LED's etc ... try to turn it around and see the value changing.

Compile the script by going into the samples directory and running: **python3 pcf8591.py**

```
1  #!/usr/bin/python
2  # -*- coding:utf-8 -*-
3  # www.elecrow.com
4  import smbus
5  import time
6
7
8  # NOTE: the PCF8591 script can be used both with photoresistance example and flame sensor example
9  # please follow up the instructions on the PCF8591 schematic to connect the right sensors in the right way.
10
11  address = 0x48
12
13  A0 = 0x40
14  A1 = 0x41
15  A2 = 0x42
16  A3 = 0x43
17
18  bus = smbus.SMBus(1)
19
20  while True:
21      bus.write_byte(address,A0)
22      value = bus.read_byte(address)
23      print("Value: %s" % value)
24      time.sleep(0.1)
25
26
```

## Application effect

---

By running the program it will show the value of the potentiometer, to change the value need to turn the potentiometer physically left or right, this will change his value.

## LESSON 20: WATER LEVEL MONITORING



### Introduction

This is a water level measurement experiment, it is relatively simple to achieve, only need to read the value of the analog port (A0 or others), and then converted to a percentage.

We'll be using the PCF8591 module as well for the water level monitoring example.

### Specification

Operating voltage: DC3-5V

Operating current: less than 20mA Sensor Type: Analog



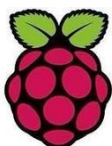


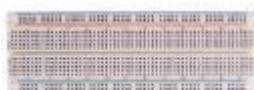

Production process: FR4 double-sided HASL Humidity: 10% -90% non-condensing Detection Area: 40mmx16mm

Product Dimensions: 62mmx20mmx8mm

### Hardware required

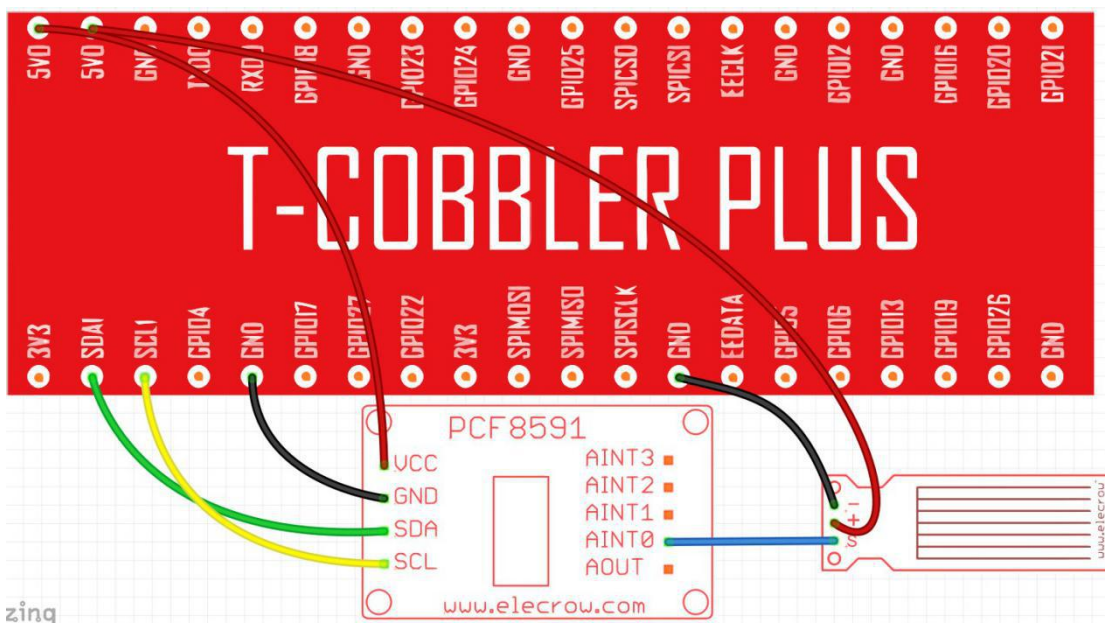
We'll need the water sensor as our main hardware component and connect it to the PCF8591 to get the value once the water has made a contact.

Same as our previous examples, for the water sensor we have no use in 10K ohm resistors.

Material diagram	Material Name	Number (amount)
	Water sensor	1
	PCF8591	1
	Raspberry Pi Board	1
	T-Cubler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

### Connection Diagram

The water sensor have 3 pins, we connect the negative to GND, the positive to 5V and the S pin, the one we use for the GPIO to AINT0. Once the circuit is closed by watering touching the surface of the water sensor it will send signal to the AINT0 pin and we'll be able to know the amount of water over the sensor.



### Connection

Water sensor	Raspberry Pi
Negative (-)	GND
Positive (+)	5V

---

Water sensor	Raspberry Pi
S	AIN0

PCF8591	Raspberry Pi
VCC	5V
GND	GND
SDA	SDA1
SCL	SCL1

## Code Overview, Compile and run

For this example as we use the water sensor, by introducing drops of water to the sensor the value will change. The more water the higher values.

But please do note: the water sensor is not waterproof and wasn't meant to be submarined under the water.

Compile the script by going into the samples directory and running: **python3 pcf8591.py**

```

1  #!/usr/bin/python
2  # -*- coding:utf-8 -*-
3  # www.elecrow.com
4  import smbus
5  import time
6
7
8  # NOTE: the PCF8591 script can be used both with photoresistance example and flame sensor example
9  # please follow up the instructions on the PCF8591 schematic to connect the right sensors in the right way.
10
11  address = 0x48
12
13  A0 = 0x40
14  A1 = 0x41
15  A2 = 0x42
16  A3 = 0x43
17
18  bus = smbus.SMBus(1)
19
20  while True:
21      bus.write_byte(address,A0)
22      value = bus.read_byte(address)
23      print("Value: %s" % value)
24      time.sleep(0.1)
25
26

```



---

### Application effect

By running the script the value will change based on the amount of water drops on the sensor, the higher amount of water the higher the value will be.

## LESSON 21: JOYSTICK



### Introduction

This experiment is to learn how to use the joystick of the analog output and digital output.

The joystick experiment will be our last lesson using the PCF8591 module.

As we learned in our previous lessons, the PCF8591 module is very useful for many scenarios.

The joystick example is a very good one, a very popular use case for robotics and applications that require controlling or moving.

---



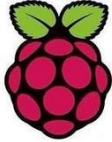
By finishing the PCF8591 lessons, we hope that you will be able to create any analog circuit for any project that you wish for and continue making everything - easier!





### Hardware required

For the lesson we'll need the joystick module which is very easy to be recognised.

We'll also need the PCF8591 module, connected to the circuit same as the previous lessons.

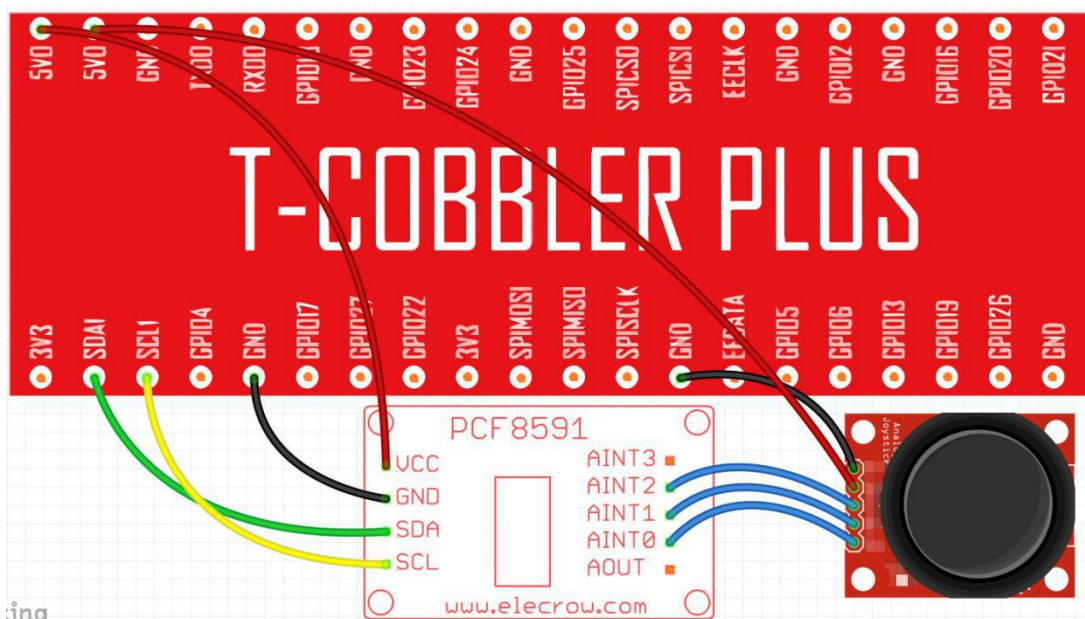
We won't need any resistors for this lesson as well.

Material diagram	Material Name	Number (amount)
	Joystick Module	1
	PCF8591	1
	Raspberry Pi Board	1

Material diagram	Material Name	Number (amount)
	T-Cobbler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

### Connection Diagram

The Joystick have 5 pins, 2 of them are the GND and the VCC which goes to 5V pin. The rest are VRX, VRY and SW. make sure to connect those pins to the right place on the PCF8591 module.  
The SW goes to AINT0, VRY goes to AINT1 and VRX goes to AINT2.



---

## Connection

Joystick	Raspberry Pi
GND	GND
5V	5V
VRX	AINT2
VRY	AINT1
SW	AINT0

PCF8591	Raspberry Pi
VCC	5V
GND	GND
SDA	SDA1
SCL	SCL1

## Code Overview, Compile and run

This time note that unlike the previous examples we need to connect to 3 pins of the PCF8591 module instead of one. We need to connect to 0,1 and 2. (one for each direction the joystick will move)

After doing so, it should work just fine with a new script we've created specially for the **pcf8591** called **pcf8591\_joystick.py**

Compile the script by going into the samples directory and running: **python3 pcf8591\_joystick.py**

```
1  #!/usr/bin/python
2  # -*- coding:utf-8 -*-
3  # www.elecrow.com
4  import smbus
5  import time
6
7
8  # NOTE: the PCF8591 script can be used both with photoresistance example and flame sensor example
9  # please follow up the instructions on the PCF8591 schematic to connect the right sensors in the right way.
10
11  address = 0x48
12
13  A0 = 0x40
14  A1 = 0x41
15  A2 = 0x42
16  A3 = 0x43
17
18  bus = smbus.SMBus(1)
19
20  while True:
21      bus.write_byte(address,A0)
22      value = bus.read_byte(address)
23      print("Value: %s" % value)
```

---

## Application effect

Running the script will show values changing based on the joystick direction, try to move it to different directions to see the value changing.

## LESSON 22: IR REMOTE

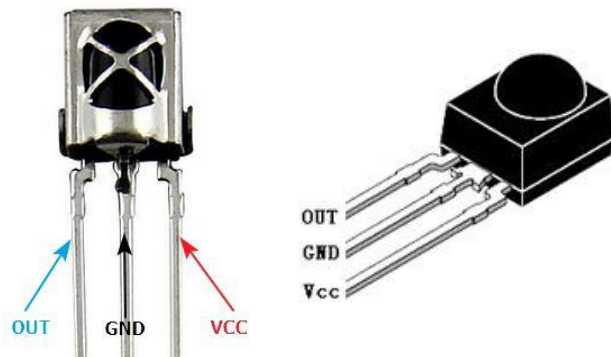


## Introduction

In this lesson, we use the **lirc** library to read infrared signals returned by buttons of the remote control and translate them to button values. When a button is pressed, the IR transmitter in the remote control will send out the corresponding IR encoding signals. On the other side, when the IR receiver receives certain encoding signals, it will decode them to identify which button is pressed.


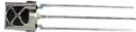
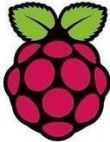


Before going through this lesson make sure you followed the installation requirements we introduced at the beginning of this document.

## IR Pin definition





### Hardware required

For this lesson we'll need to use the IR receiver which looks like a small LED with black pin on the end of it (don't confuse it with the flame sensor we used earlier, it has 2 pins) as well as the remote control to send the signals to the IR receiver to accept.

Material diagram	Material Name	Number (amount)
	IR Remote	1
	IR Receiver	1
	Raspberry Pi Board	1
	T-Cubler Plus	1
	40P GPIO Cable	1

---

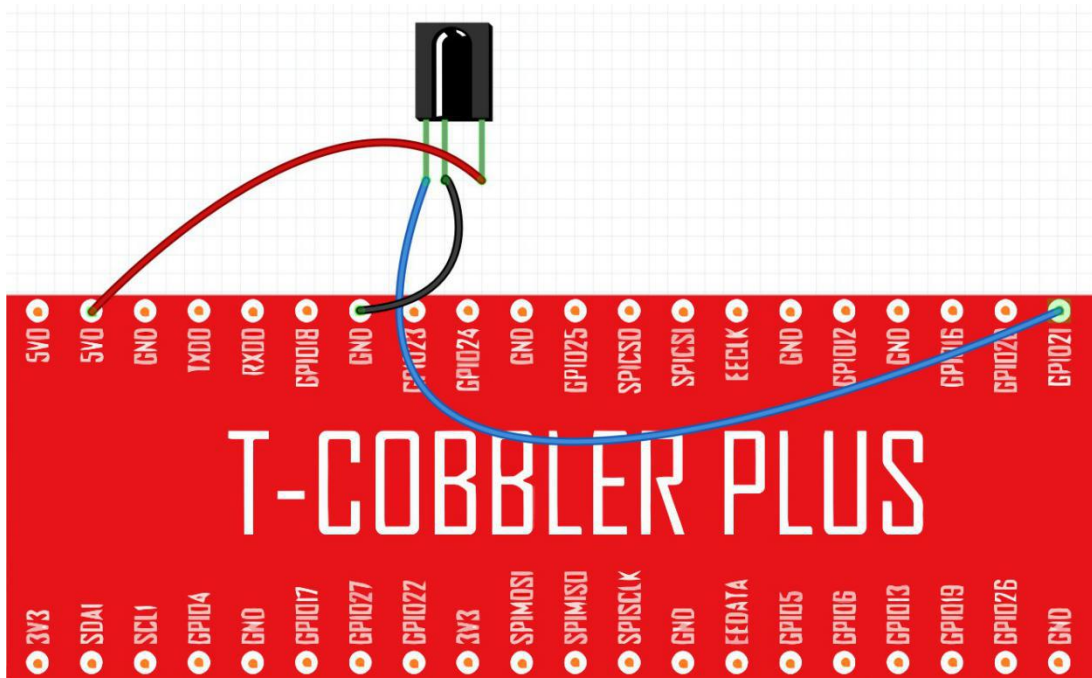
Material diagram	Material Name	Number (amount)
	Breadboard	1
	Jumper wires	Several

### Connection Diagram

In order for the remote control to send the signal, we just need the remote control itself.

But what if we want to accept the signal on the other side, on the raspberry pi? We'll need to plug the IR receiver to the GPIO pins. I

The IR receiver has 3 pins, GND and VCC (VCC goes to 5V) and OUT pin which we will use on GPIO21 to receive the commands from the IR remote control.



### Connection

IR	Raspberry Pi
OUT	GPIO21
GND	GND
VCC	5V

### Code Overview, Compile and run

Before we start, make sure that you've installed the lirc and python-lirc drivers at the beginning of our lessons. Without the drivers, the lesson will not work properly and you won't be able to use your IR LED.



---

The code is quite simple but a little difficult to understand as we use sockets, it's not as clean as previous code we used.

We receive data from the IR LED and then we convert the data, after converting it we are able to tell which button exactly was pressed on the IR remote. Then we print it using the command `print(Out[0])`

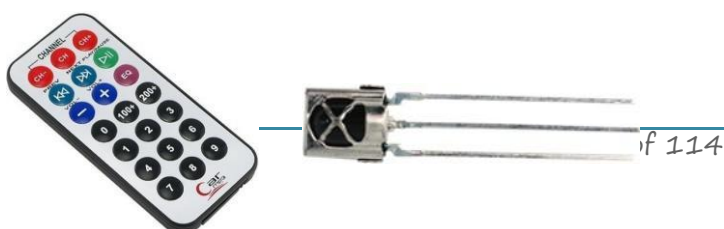
Compile the script by going into the samples directory and running: **python2 IR.py**

```
6  GPIO.setmode(11)
7  GPIO.setup(17, 0)
8  GPIO.setup(18, 0)
9  PORT = 42001
10 HOST = "localhost"
11 Socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12
13 Lirc = lirc.init("keys")
14 #lirc.set_blocking(False, Lirc)      # Un-Comment to stop nextcode() from waiting for a signal ( will return empty array whe
15
16 def handler(signal, frame):
17     Socket.close()
18     GPIO.cleanup()
19     exit(0)
20
21 signal.signal(signal.SIGTSTP, handler)
22
23 def sendCmd(cmd):
24     n = len(cmd)
25     a = array('c')
26     a.append(chr((n >> 24) & 0xFF))
27     a.append(chr((n >> 16) & 0xFF))
28     a.append(chr((n >> 8) & 0xFF))
29     a.append(chr(n & 0xFF))
30     Socket.send(a.tostring() + cmd)
31
32 while True:
33
34     Out = lirc.nextcode()
35     print(Out[0])
```

## Application effect

Running the program will allow us to read the IR code through the IR remote, press the IR remote buttons to see them on the terminal screen after the python script captures them using the IR receiver.

## LESSON 23: IR REMOTE CONTROL LED



---

## Introduction

In the previous lesson we learned how to accept the commands from the IR remote control and receive them through our python code on the raspberry pi.

But what's the interesting thing in that?





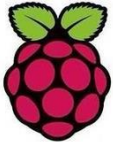




During this lesson we'll take the IR remote control project a little further and we will control the LED using the IR remote control, we'll learn how to configure specific button to have a specific function to turn our LED once pressed.

## Hardware required

For the hardware we'll need the IR remote and the IR receiver.

For the LED we'll need a small LED (you can choose any colour you want) and 220 or 330ohm resistor in order not to burn out the LED while plugging it in.

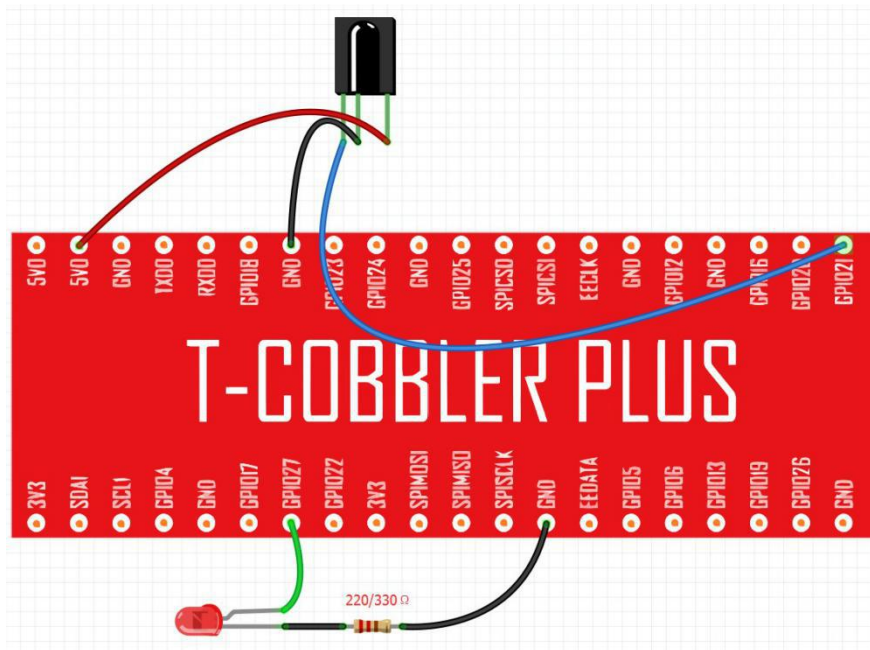
Make sure to use 220 or 330ohm resistor, not K resistor.

Material diagram	Material Name	Number (amount)
	IR Remote	1
	IR Receiver	1
	LED	1
	220/330ohm resistor	1
	Raspberry Pi Board	1
	T-Cubler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

### Connection Diagram

We'll keep the connection from our previous lesson regarding the IR receiver but this time we connect the LED as well.

The LED have 2 pins, long pin and short pin. The long pin is the positive that we connect directly to GPIO27 and the short pin goes through the resistor to the GND connection on the raspberry pi.



## Connection

LED	Raspberry Pi
Long Pin (+)	GPIO27
Short Pin (-)	GND

IR	Raspberry Pi
OUT	GPIO21
GND	GND
VCC	5V

## Code Overview, Compile and run

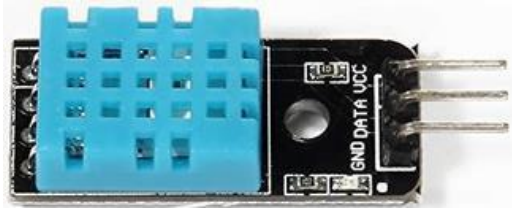
Tips: Refer to the operation demo (Step4 to Step7).

---

### Application effect

Running the program, turns on an LED on for one second, then off for one second, repeatedly.

## LESSON 24: DHT11



## Introduction

This lesson will teach you how to use DHT11 module, which is simple and easy to use.

The DH11 module is a temperature and humidity combined module, it's affordable and relatively accurate.


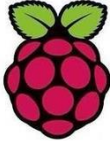




It is very useful in multiple IoT projects such as smart home or environment monitoring.

It can be used both indoor and outdoor.

---

## Hardware required

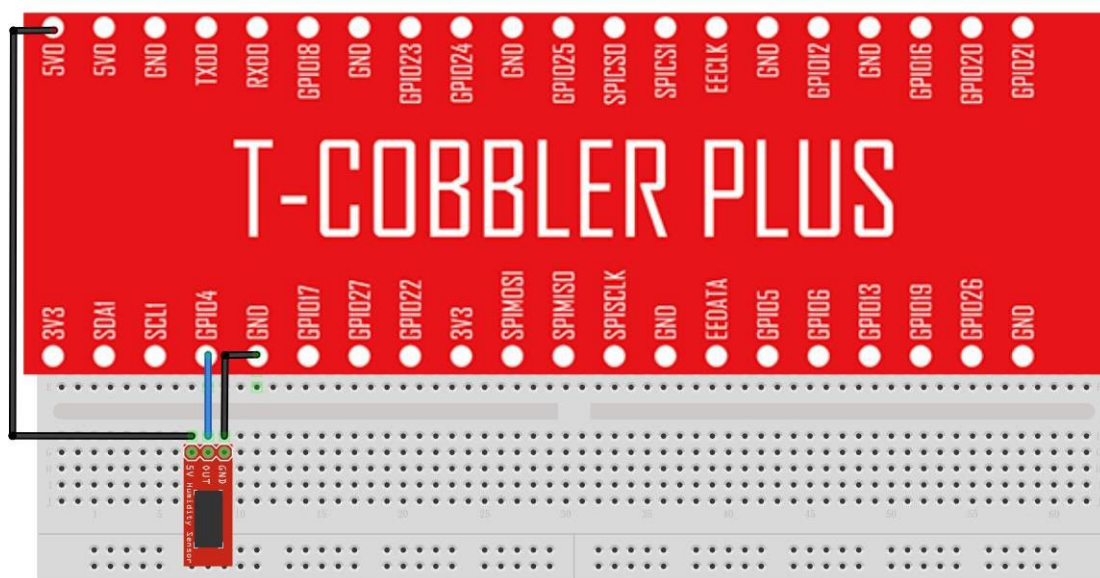
The main component we will use is the DHT11, it can be easy recognisable as its blue top layer.

Material diagram	Material Name	Number (amount)
	DHT11 Module	1
	Raspberry Pi Board	1
	T-Cubbler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

## Connection Diagram

The connection is straight forward, our DHT11 sensor have 3 pins: GND, VCC (5V) and data OUT.

Using the data out pin we'll be able to get the values of the temperature and the humidity. The data pin goes directly to GPIO4 pin while the VCC goes to 5V and GND goes to GND.



## Connection

DHT11	Raspberry Pi
GND	GND
DATA OUT	GPIO4
VCC	5V



---

## Code Overview, Compile and run

Within the DH11 script we will use the adafruit DHT library which will allow us to get the data from the sensor. We will specify that we use sensor 11 as the DHT family have multiple sensors and the one we use is the DHT-11.

In one line of code (line 33) we are going to get both temperature and humidity and print them out if they are not empty. If they return empty something might be wrong with your connections, make sure you connect the sensor to the correct pins!

Compile the script by going into the samples directory and running: **python3 dh11.py**

```
23 import sys
24 import Adafruit_DHT
25
26 # set type of the sensor
27 sensor = 11
28 # set pin number
29 pin = 4
30
31 # Try to grab a sensor reading. Use the read_retry method which will retry up
32 # to 15 times to get a sensor reading (waiting 2 seconds between each retry).
33 humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
34
35 # Uncomment the line below to convert the temperature to Fahrenheit.
36 # temperature = temperature * 9/5.0 + 32
37
38 # Note that sometimes you won't get a reading and
39 # the results will be null (because Linux can't
40 # guarantee the timing of calls to read the sensor).
41 # If this happens try again!
42 if humidity is not None and temperature is not None:
43     print('Temp={0:0.1f}* Humidity={1:0.1f}%'.format(temperature, humidity))
44 else:
45     print('Failed to get reading. Try again!')
```

## Application effect

The script will get the temperature and the humidity out of the DH11 sensor and will print them to the console.

---

## LESSON 25: LCD1602 WITH IIC



### Introduction


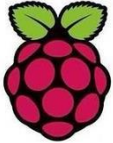




This is an experiment on how to use LCD1602 with IIC, the next lesson will do a temperature and humidity monitoring experiment.

In this lesson we'll just show "hello world" text on our LCD screen to show a good example how we can use the LCD to show certain text.

---

## Hardware required

The main component we'll be using is the LCD1602, we won't be in need in any resistors here.

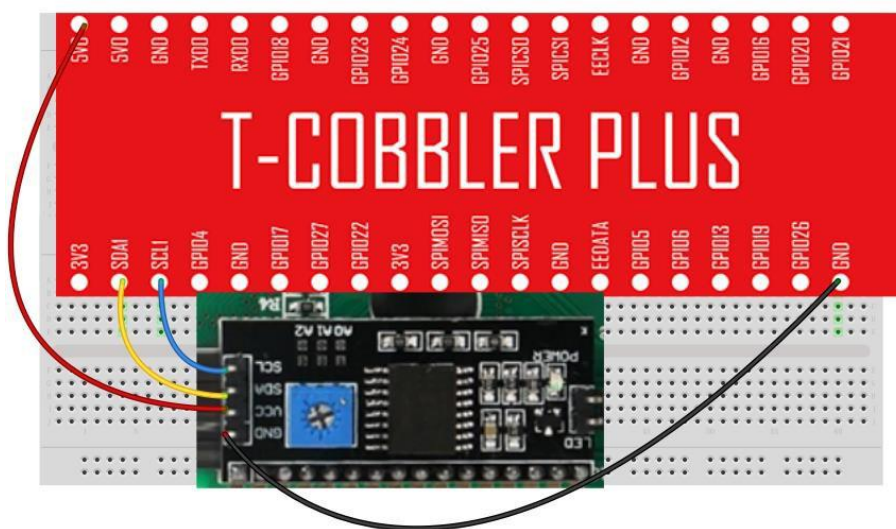
Material diagram	Material Name	Number (amount)
	LCD1602 with IIC	1
	Raspberry Pi Board	1
	T-Cubler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

---

## Connection Diagram

The connections are quite simple, make sure you put the SDA and SCL pin in the correct locations or you won't be able to see any data on the LCD display.

We don't need any resistors here so you can connect it directly to the raspberry pi without any problems.



## Connection

LCD1602	Raspberry Pi
GND	GND
VCC	5V
SDA	SDA1
SCL	SCL1

---

## Code Overview, Compile and run

The code is very simple.

We will initialise the LCD using the Adafruit char lcd library, we will turn the backlight so we can see what we print on the screen, we will clean the script just in case some text left from before and then we show a message of “hello world” we wait 5 seconds and repeat it all again, you might see the screen flickering, that’s totally normal.

Compile the script by going into the samples directory and running: **python3 lcd.py**

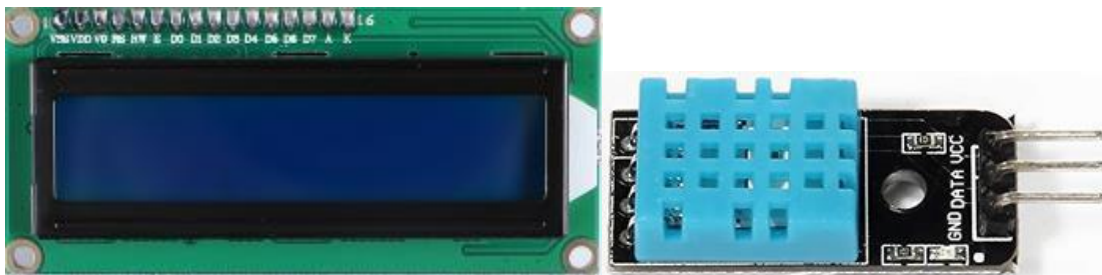
```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  import sys
4  import time
5  import Adafruit_CharLCD as LCD
6
7  # Define LCD column and row size for 16x2 LCD.
8  lcd_columns = 16
9  lcd_rows    = 2
10
11 # Initialize the LCD using the pins
12 lcd = LCD.Adafruit_CharLCDBackpack(address=0x21)
13
14 try:
15     while True:
16         # Turn backlight on
17         lcd.set_backlight(0)
18         # clean the LCD screen
19         lcd.clear()
20         lcd.message('Hello world')
21         time.sleep(5)
22 except KeyboardInterrupt:
23     # Turn the screen off
24     lcd.clear()
25     lcd.set_backlight(1)
```

---

### Application effect

The LCD will show “hello world” message, you can interrupt the script by pressing CTRL+C or CTRL+Z

## LESSON 26: TEMPERATURE AND HUMIDITY MONITORING



### Introduction



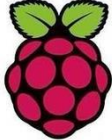




This is a more complex experiment, it can show the monitoring of indoor temperature and humidity, and in the LCD above display value.

As we showed “hello world” in our previous lesson we are going to take this lesson one step further and show real time data from the DHT11 sensor, after completing this lesson you’ll be able to understand how to show any type of message on the LCD monitor and so accomplish many other projects related to the same module.

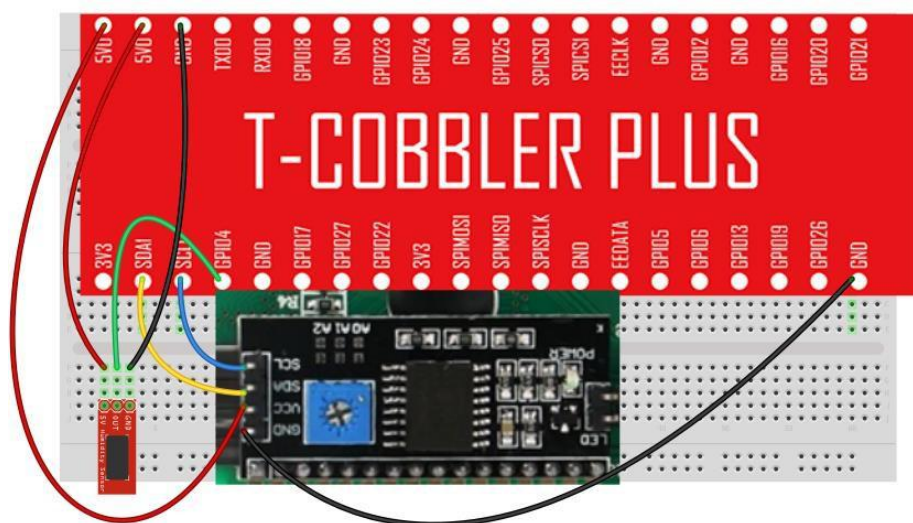
---

## Hardware required

To this example we've also added the DHT11 module, we'll need both the DHT11 and the LCD module in order to accomplish this lesson, make sure to prepare them both.

Material diagram	Material Name	Number (amount)
	DHT11 Module	1
	LCD1602 with IIC	1
	Raspberry Pi Board	1
	T-Cubler Plus	1
	40P GPIO Cable	1
	Breadboard	1
	Jumper wires	Several

## Connection Diagram



## Connection

DHT11	Raspberry Pi
GND	GND
DATA OUT	GPIO4
VCC	5V

LCD1602	Raspberry Pi
GND	GND
VCC	5V
SDA	SDA1
SCL	SCL1



---

## Code Overview, Compile and run

Our DHT LCD example is very similar to the previous LCD example except this time we don't just show "hello world" message, we'll get the current temperature and humidity values from our DH11 sensor and then we gonna display then over the LCD screen.

The temperature going to be displayed in C, if you you want to display it in F, make sure to uncomment the line that is on line number 28.

Compile the script by going into the samples directory and running: **python3 lcd\_dht.py**

```
20  try:
21      while True:
22          # Turn backlight on
23          lcd.set_backlight(0)
24          # get temperature and humidity
25          humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
26          # clean the LCD screen
27          lcd.clear()
28          # Un-comment the line below to convert the temperature to Fahrenheit.
29          # temperature = temperature * 9/5.0 + 32
30          if humidity is not None and temperature is not None:
31              lcd.message('Temp={0:0.1f}* Humidity={1:0.1f}%'.format(temperature, humidity))
32          else:
33              print('Failed to get reading, Retrying in 5 seconds!')
34          # wait 5 seconds for the next try
35          time.sleep(5)
36  except KeyboardInterrupt:
37      # Turn the screen off
38      lcd.clear()
39      lcd.set_backlight(1)
```

---

---

### Application effect

Running the program will turn on the LCD and show the current room temperature and humidity from the DHT11 sensor.