

Az-Delivery

Welcome!

Thank you for purchasing our *AZ-Delivery 35 in 1 Sensors Kit*. On the following pages, you will be introduced to how to use and set-up these handy devices.

Have fun!



Table of Contents

Introduction.....	12
How to set-up Arduino IDE.....	13
How to set-up the Raspberry Pi and Python.....	17
Uno as ADC for the Raspberry Pi.....	18
External ADC for the Raspberry Pi.....	29
Connecting the ADS1115 module with Raspberry Pi.....	31
Libraries and tools for Python.....	33
Enabling the I2C interface.....	34
Test script for ADS1115 module.....	36
35 in 1 Sensors Kit.....	39
KY-001 Digital Temperature Sensor Module.....	40
Introduction.....	40
Specifications.....	41
The pinout.....	42
Connecting the sensor with Uno.....	43
Library for Arduino IDE.....	44
Connecting three KY-001 modules with Uno.....	45
Sketch example for using three KY-001 modules.....	46
Connecting the sensor with Raspberry Pi.....	53
Enabling the One-Wire interface.....	54
Enabling multiple One-Wire interfaces.....	57
Python script for reading multiple KY-001 modules.....	58
KY-002 Shock Sensor Module.....	63
Introduction.....	63
Specifications.....	64

Az-Delivery

The pinout.....	64
Connecting the sensor with Uno.....	65
Sketch example.....	66
Connecting the sensor with Raspberry Pi.....	68
Python script.....	69
KY-003 Hall-effect Sensor Module.....	71
Introduction.....	71
Specifications.....	72
The pinout.....	73
Connecting the sensor with Uno.....	74
Sketch example.....	75
Connecting the sensor with Raspberry Pi.....	76
Python script.....	77
KY-004 Button Module.....	79
Introduction.....	79
Specifications.....	80
The pinout.....	80
Connecting the module with Uno.....	81
Sketch example.....	82
Connecting the module with Raspberry Pi.....	83
Python script.....	84
Another Python script example.....	88
KY-005 Infrared LED Module.....	91
Introduction.....	91
Specifications.....	92
The pinout.....	93
Working principle of TV remotes.....	94
Connecting the module with Uno.....	96

Az-Delivery

Basic sketch example.....	97
Library for Arduino IDE.....	98
TV remote with the KY-005 (Uno).....	99
Sketch example.....	100
Connecting the module with Raspberry Pi.....	102
Python script.....	103
KY-006 Passive Buzzer Module.....	104
Introduction.....	104
Specifications.....	105
The pinout.....	105
Connecting the module with Uno.....	106
Sketch example.....	107
Connecting the module with Raspberry Pi.....	108
Python script.....	109
A melody.....	111
Traffic Light LED module.....	112
Introduction.....	112
Specifications.....	113
The pinout.....	114
Connecting the module with Uno.....	115
Sketch example.....	116
Connecting the module with Raspberry Pi.....	117
Python script.....	118
KY-009 SMD RGB LED Module.....	120
Introduction.....	120
Specifications.....	121
The pinout.....	121
Connecting the module with Uno.....	122

Az-Delivery

Sketch example.....	123
Connecting the module with Raspberry Pi.....	124
Python script.....	125
KY-010 Light Barrier Detector Module.....	127
Introduction.....	127
Specifications.....	128
The pinout.....	128
The schematic.....	129
Connecting the module with Uno.....	130
Sketch example.....	131
Connecting the module with Raspberry Pi.....	132
Python script.....	133
KY-011 Bi-color 5mm LED Module.....	135
Introduction.....	135
Specifications.....	136
The pinout.....	136
Connecting the module with Uno.....	137
Sketch example.....	138
Connecting the module with Raspberry Pi.....	139
Python script.....	140
KY-012 Active Buzzer Module.....	142
Introduction.....	142
Specifications.....	143
The pinout.....	143
Connecting the module with Uno.....	144
Sketch example.....	145
Connecting the module with Raspberry Pi.....	146
Python script.....	147

Az-Delivery

KY-013 Analog Temperature Thermistor Sensor Module.....	149
Introduction.....	149
Specifications.....	150
The pinout.....	151
The schematic.....	152
Connecting the module with Uno.....	153
Sketch example.....	155
Connecting the module with Raspberry Pi.....	158
KY-015 DHT11 Temperature Sensor Module.....	161
Introduction.....	161
Specifications.....	162
Working principle of the DHT11 sensor.....	163
The pinout.....	165
Connecting the module with Uno.....	166
Library for Arduino IDE.....	167
Sketch example.....	168
Connecting the module with Raspberry Pi.....	172
Library and tools for Python.....	173
Python script.....	174
KY-016 RGB LED Module.....	176
Introduction.....	176
Specifications.....	177
The pinout.....	177
Connecting the module with Uno.....	178
Sketch example.....	179
Connecting the module with Raspberry Pi.....	180
Python script.....	181
KY-018 Photoresistor LDR Module.....	183

Az-Delivery

Introduction.....	183
Specifications.....	184
The pinout.....	184
Connecting the module with Uno.....	185
Sketch example.....	186
Connecting the module with Raspberry Pi.....	187
KY-019 Relay Module.....	189
Introduction.....	189
Specifications.....	190
Working principle of the relay.....	191
The pinout.....	192
Connecting the AC side of the relay.....	193
Connecting the module with Uno.....	194
Sketch example.....	195
Connecting the module with Raspberry Pi.....	196
Python script.....	197
KY-020 Tilt Sensor Module.....	199
Introduction.....	199
Specification.....	200
The pinout.....	200
Connecting the module with Uno.....	201
Sketch example.....	202
Connecting the module with Raspberry Pi.....	203
Python script.....	204
KY-021 Mini Reed Sensor Module.....	206
Introduction.....	206
Specification.....	207
The pinout.....	207

Az-Delivery

Connecting the module with Uno.....	208
Sketch example.....	209
Connecting the module with Raspberry Pi.....	210
Python script.....	211
KY-022 Infrared Receiver Sensor Module.....	213
Introduction.....	213
Specifications.....	213
Working principle.....	214
The pinout.....	216
Connecting the module with Uno.....	217
Library for Arduino IDE.....	218
Sketch code for detecting hex code.....	219
Sketch code for controlling an LED.....	222
Emulating the TV remote with the KY-005.....	223
Sketch code.....	224
Connecting the module with Raspberry Pi.....	227
Python script.....	228
KY-023 Joystick Module.....	236
Introduction.....	236
Specifications.....	237
The pinout.....	237
Connecting the module with Uno.....	238
Sketch example.....	239
Connecting the module with Raspberry Pi.....	241
KY-024 Digital Hall Sensor Module.....	243
Introduction.....	243
Specifications.....	244
The pinout.....	245

Az-Delivery

Connecting the module with Uno.....	246
Sketch example.....	247
Connecting the module with Raspberry Pi.....	249
Python script.....	250
KY-025 Reed Sensor Module.....	252
Introduction.....	252
Specifications.....	253
The pinout.....	254
Connecting the module with Uno.....	255
Sketch example.....	256
Connecting the module with Raspberry Pi.....	258
Python script.....	259
KY-026 Flame Detector Module.....	261
Introduction.....	261
Specifications.....	262
The pinout.....	263
Connecting the module with Uno.....	264
Sketch example.....	265
Connecting the module with Raspberry Pi.....	267
Python script.....	268
KY-028 Thermistor Sensor Module.....	270
Introduction.....	270
Specifications.....	271
The pinout.....	272
Connecting the module with Uno.....	273
Sketch example.....	274
Connecting the module with Raspberry Pi.....	276
Python script.....	278

Az-Delivery

KY-029 Bi-color 3mm LED Module.....	280
Introduction.....	280
Specifications.....	281
The pinout.....	281
Connecting the module with Uno.....	282
Sketch example.....	283
Connecting the module with Raspberry Pi.....	284
Python script.....	285
KY-031 Hit-Knock Sensor Module.....	287
Introduction.....	287
Specifications.....	288
The pinout.....	288
Connecting the module with Uno.....	289
Sketch example.....	290
Connecting the module with Raspberry Pi.....	291
Python script.....	292
KY-032 Obstacle Detector Module.....	296
Introduction.....	296
Specifications.....	297
The pinout.....	298
The schematic.....	299
Connecting the module with Uno.....	300
Sketch example.....	301
Connecting the module with Raspberry Pi.....	304
Python script.....	305
KY-033 Line-tracker Sensor Module.....	307
Introduction.....	307
Specifications.....	308

Az-Delivery

The pinout.....	308
Connecting the module with Uno.....	309
Sketch example.....	310
Connecting the module with Raspberry Pi.....	312
Python script.....	313
KY-034 7 Color Changing LED Module.....	315
Introduction.....	315
Specifications.....	316
The pinout.....	316
Connecting the module with Uno.....	317
Sketch example.....	318
Connecting the module with Raspberry Pi.....	319
Python script.....	320
KY-035 Linear Magnetic Sensor Module.....	321
Introduction.....	321
Specifications.....	322
The pinout.....	323
Connecting the module with Uno.....	324
Sketch example.....	325
Connecting the module with Raspberry Pi.....	326
KY-036 Touch Sensor Module.....	328
Introduction.....	328
Specifications.....	329
The pinout.....	330
Connecting the module with Uno.....	331
Sketch example.....	332
Connecting the module with Raspberry Pi.....	334
Python script.....	335

KY-037 Big Microphone Sensor Module.....	337
Introduction.....	337
Specifications.....	338
The pinout.....	339
Connecting the module with Uno.....	340
Sketch example.....	341
Connecting the module with Raspberry Pi.....	344
Python script.....	346
KY-038 Small Microphone Sensor Module.....	348
Introduction.....	348
KY-039 Heartbeat Sensor Module.....	349
Introduction.....	349
Specifications.....	350
The pinout.....	350
Connecting the module with Uno.....	351
Sketch example.....	352
Connecting the module with Raspberry Pi.....	355
Full Python script.....	357
KY-040 Rotary Encoder Module.....	365
Introduction.....	365
Working principle.....	366
The output.....	368
Specifications.....	369
The pinout.....	369
Connecting the module with Uno.....	370
Sketch example.....	371
Connecting the module with Raspberry Pi.....	375
Python script.....	376

The 35 in 1 Sensor Kit includes:



KY-001 DS18B20
Temperature
Sensor
Module.png



KY-002 Shock
Sensor
Module.png



KY-003 Hall Sensor
Module.png



KY-004 Push
Button
Module.png



KY-005 Infrared
LED Module.png



KY-006 Passive
Buzzer Module.png



Traffic light LED
module.png



KY-009 SMD RGB
LED Module.png



KY-010 Light
Barrier Detector
Module.png



KY-011 Bi-color
LED 5mm
Module.png



KY-012 Active
Buzzer Module.png



KY-013 Analog
Thermistor Sensor
Module.png



KY-015 DHT11
Temperature
Sensor
Module.png



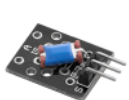
KY-016 RGB LED
Module.png



KY-018
Photoresistor LDR
Module.png



KY-019 Relay
Module.png



KY-020 Tilt Sensor
Module.png



KY-021 Mini Reed
Sensor
Module.png



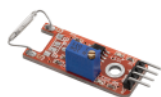
KY-022 Infrared
Receiver
Module.png



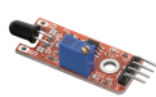
KY-023 Joystick
Module.png



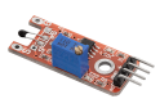
KY-024 Digital Hall
Sensor
Module.png



KY-025 Reed
Sensor
Module.png



KY-026 Flame
Detector
Module.png



KY-028 Thermistor
Module.png



KY-029 Bi-color
LED 3mm
Module.png



KY-031 Hit-Knock
Sensor
Module.png



KY-032 Obstacle
Detector
Module.png



KY-033 Line
Follower
Module.png



KY-034 7 Color
Changing LED
Module.png



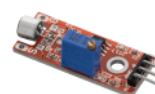
KY-035 Bi-hor
Linear Magnetic
Sensor
Module.png



KY-036 Touch
Sensor
Module.png



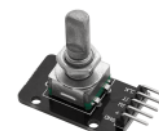
KY-037 Big
Microphone
Module.png



KY-038 Small
Microphone
Module.png



KY-039 Heart Beat
Sensor
Module.png



KY-040 Rotary
Encoder
Module.png

1. KY-001 DS18B20 Digital Temperature Sensor Module
2. KY-002 Shock Sensor Module
3. KY-003 Hall Sensor Module
4. KY-004 Push Button Module
5. KY-005 Infrared LED Module
6. KY-006 Passive Buzzer Module
7. KY-007 LED Traffic Light Module
8. KY-009 SMD RGB LED Module
9. KY-010 Light-barrier Detector Module
10. KY-011 Bi-color LED 5mm Module
11. KY-012 Active Buzzer Module
12. KY-013 Analog Temperature Thermistor Module
13. KY-015 DHT11 Temperature Sensor Module
14. KY-016 RGB LED Module
15. KY-018 Photoresistor LDR Module
16. KY-019 Relay Module
17. KY-020 Tilt Sensor Module
18. KY-021 Mini Reed Sensor Module
19. KY-022 Infrared Receiver Module
20. KY-023 Joystick Module
21. KY-024 Digital Hall Sensor Module
22. KY-025 Reed Sensor Module
23. KY-026 Flame Detector Module
24. KY-028 Thermistor Module

- 25. KY-029 Bi-color LED 3mm Module
- 26. KY-031 Hit-Knock Sensor Module
- 27. KY-032 Obstacle Detector Module
- 28. KY-033 Line-tracker Sensor Module
- 29. KY-034 7 Color Changing LED Module
- 30. KY-035 Bi-Color Linear Magnetic Sensor Module
- 31. KY-036 Touch Sensor Module
- 32. KY-037 Big Microphone Module
- 33. KY-038 Small Microphone Module
- 34. KY-039 Heartbeat Sensor Module
- 35. KY-040 Rotary Encoder Module



Introduction

The eBook is a tutorial on how to use modules from 35 in 1 Sensor Kit with the *Uno* and the *Raspberry Pi 3 model B+*. For each module there is an explanation of what it is and how it is working, followed by an example sketch for the Uno board, and an example script for the Raspberry Pi 3 Model B+.

For the purpose of the eBook the *Arduino IDE* has to be installed. Also a basic text editor, for example apps like *Notepad* or *Sublime Text*. In the Arduino IDE the sketches for the Uno are written, and in text editor the scripts in *Python 3* programming language for the Raspberry Pi are written.

How to set-up Arduino IDE for different platforms is explained in the chapter *How to set-up Arduino IDE*.

How to set-up Raspberry Pi for purpose of this eBook is explained in the chapter *How to set-up the Raspberry Pi and Python*.

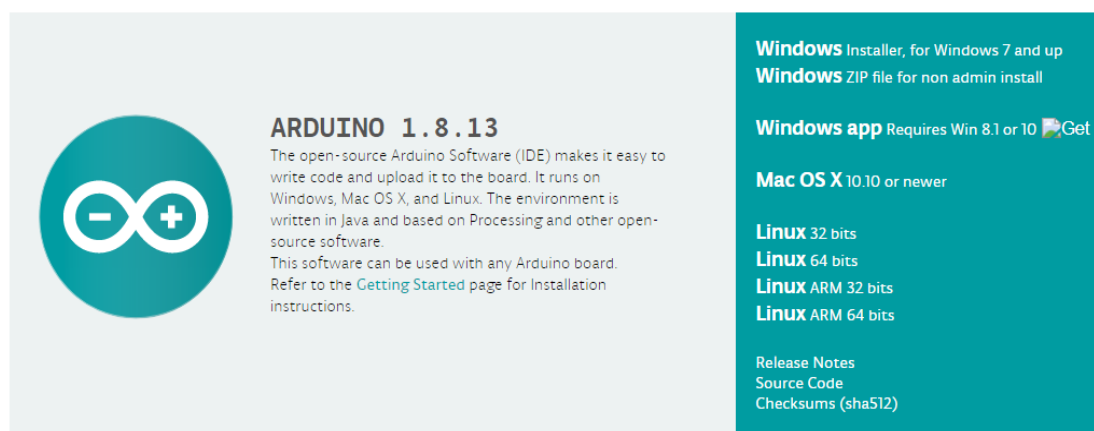
How to use the Uno as analog to digital converter for the Raspberry Pi is explained in the chapter *Uno as ADC*.

How to use external analog to digital converter, the module called *ADS1115* as analog to digital converter for the Raspberry Pi is explained in the chapter *External ADC for the Raspberry Pi*.

How to set-up Arduino IDE

If the Arduino IDE is not installed, follow the [link](#) and download the installation file for the operating system of choice.

Download the Arduino IDE



The screenshot shows the Arduino IDE download page. On the left, there is a large teal circle containing the Arduino logo (an infinity symbol with a minus sign on the left and a plus sign on the right). To the right of the logo, the text reads: **ARDUINO 1.8.13**. Below this, it says: "The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software. This software can be used with any Arduino board. Refer to the [Getting Started](#) page for installation instructions." On the right side of the page, there is a teal sidebar with white text. It lists the following options: **Windows** Installer, for Windows 7 and up; **Windows** ZIP file for non admin install; **Windows app** Requires Win 8.1 or 10 with a Microsoft Store icon; **Mac OS X** 10.10 or newer; **Linux** 32 bits; **Linux** 64 bits; **Linux** ARM 32 bits; **Linux** ARM 64 bits. At the bottom of the sidebar, there are links for [Release Notes](#), [Source Code](#), and [Checksums \(sha512\)](#).

For *Windows* users, double click on the downloaded .exe file and follow the instructions in the installation window.

Az-Delivery

For *Linux* users, download a file with the extension *.tar.xz*, which has to be extracted. When it is extracted, go to the extracted directory and open the terminal in that directory. Two *.sh* scripts have to be executed, the first called *arduino-linux-setup.sh* and the second called *install.sh*.

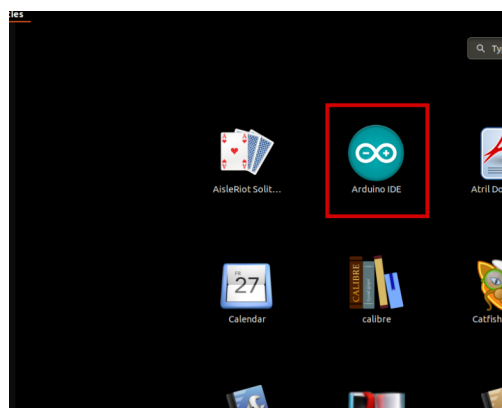
To run the first script in the terminal, open the terminal in the extracted directory and run the following command:

```
sh arduino-linux-setup.sh user_name
```

user_name - is the name of a superuser in Linux operating system. A password for the superuser has to be entered when the command is started. Wait for a few minutes for the script to complete everything.

The second script called *install.sh* script has to be used after the installation of the first script. Run the following command in the terminal (extracted directory): **sh install.sh**

After the installation of these scripts, go to the *All Apps*, where the *Arduino IDE* is installed.



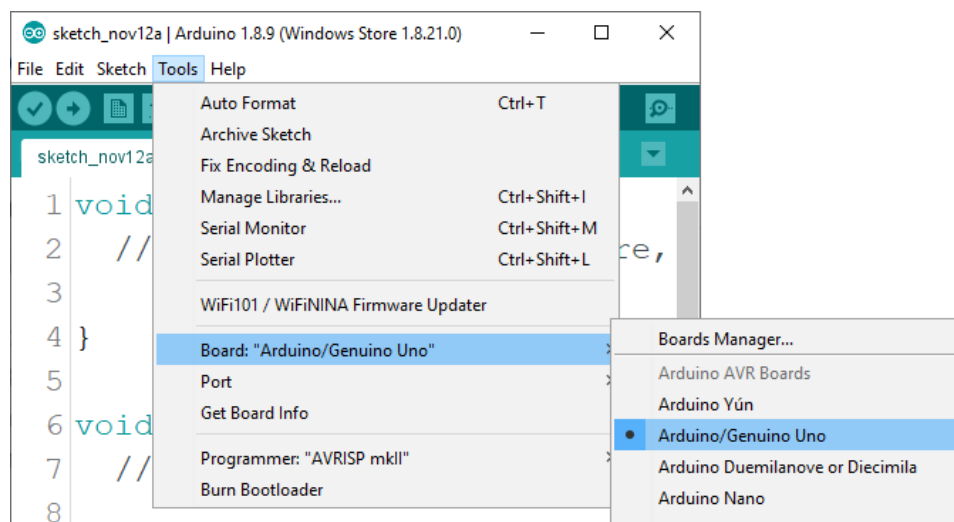
Az-Delivery

Almost all operating systems come with a text editor preinstalled (for example, *Windows* comes with *Notepad*, *Linux Ubuntu* comes with *Gedit*, *Raspberry Pi OS* comes with *Leafpad*, etc.). All of these text editors are perfectly fine for the purpose of the eBook.

Next thing is to check if your PC can detect an Arduino board. Open freshly installed Arduino IDE, and go to:

Tools > Board > {your board name here}

{your board name here} should be the *Arduino/Genuino Uno*, as it can be seen on the following image:



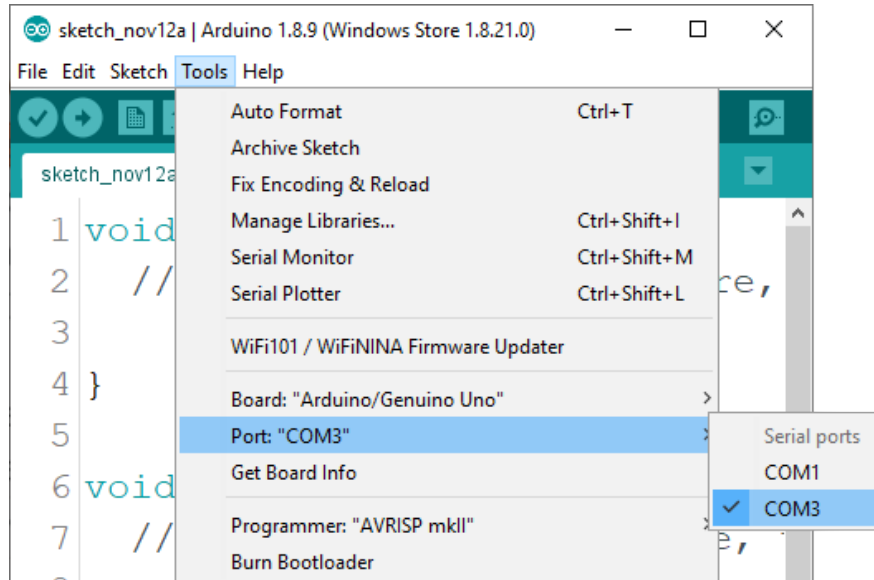
The port to which the Arduino board is connected has to be selected. Go to:

Tools > Port > {port name goes here}

and when the Arduino board is connected to the USB port, the port name can be seen in the drop-down menu on the previous image.

Az-Delivery

If the Arduino IDE is used on Windows, port names are as follows:



For *Linux* users, for example port name is `/dev/ttyUSBx`, where *x* represents integer number between 0 and 9.



How to set-up the Raspberry Pi and Python

For the Raspberry Pi, first the operating system has to be installed, then everything has to be set-up, so that it can be used in the *Headless* mode. The *Headless* mode enables remote connection to the Raspberry Pi, without need for a *PC* screen Monitor, mouse or keyboard. The only things that are used in this mode are the Raspberry Pi itself, power supply and internet connection. All of this is explained minutely in the free eBook:

[Raspberry Pi Quick Startup Guide](#)

The *Raspberry Pi OS* comes with *Python* preinstalled.

Uno as ADC for the Raspberry Pi

Because the Raspberry Pi does not have Analog to Digital Converter (ADC), but for purpose of using the module which outputs analog voltage with the Raspberry Pi, the Raspberry Pi has to be able to read analog voltages. The Uno can be used for the purpose. In order to do so, the Uno is used on *Raspberry Pi OS*. Uno can read analog voltages, and it can use *Serial Interface* via *USB* port to send data to the Raspberry Pi.

First, the Arduino IDE has to be downloaded and installed on the *Raspberry Pi OS*. Second, the firmware for Uno has to be downloaded and uploaded to the Uno, and lastly, the library for Python has to be downloaded and installed.

To do this, power on the Raspberry Pi and connect it to the internet. Start the *RealVNC* app on the remote computer and connect the app to the Raspberry Pi (like explained in the eBook for the Raspberry Pi).

First thing that has to be done when the Raspberry Pi is booted up is to update the *Raspberry Pi OS*; open the terminal and run the following command:

```
sudo apt-get update && sudo apt-get upgrade -y
```

And wait for the command to finish its job.

Now, the *Raspberry Pi OS* is up to date.

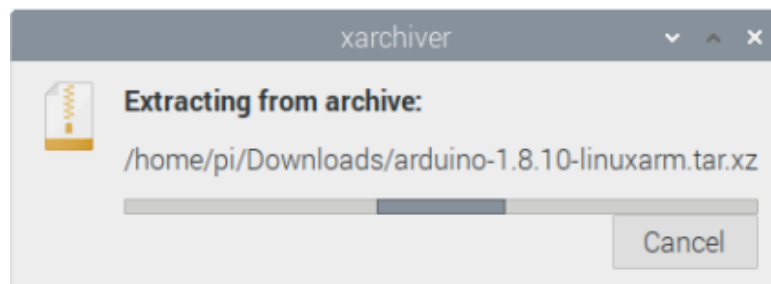
Az-Delivery

To download and install the Arduino IDE, go on the Arduino [site](#) and download the `tar.xz` file of Arduino IDE for *Linux ARM 32 bits* as shown on the following image:

Download the Arduino IDE



Then, the `tar.xz` file has to be extracted. Open the *File explorer* in directory where the `tar.xz` file is downloaded, right click on it, and run the option *Extract Here*. Wait for a few minutes for the extracting process to complete itself.

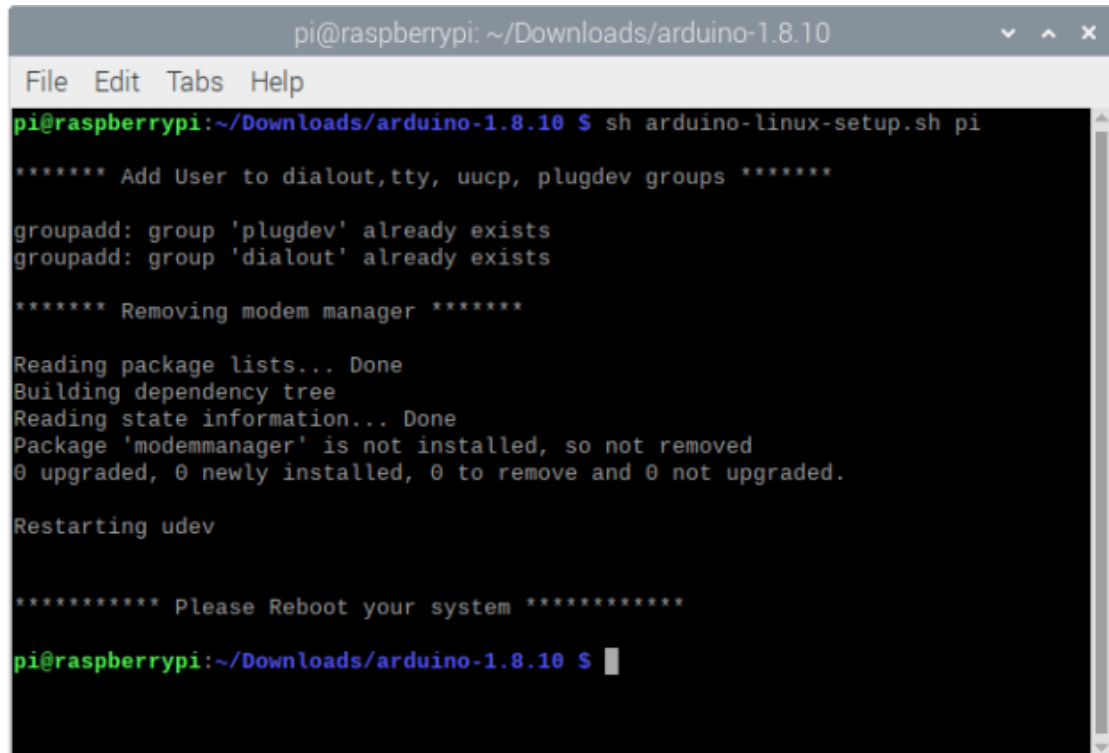


Az-Delivery

Open the terminal in the extracted folder and run the following command:

sh arduino-linux-setup.sh pi

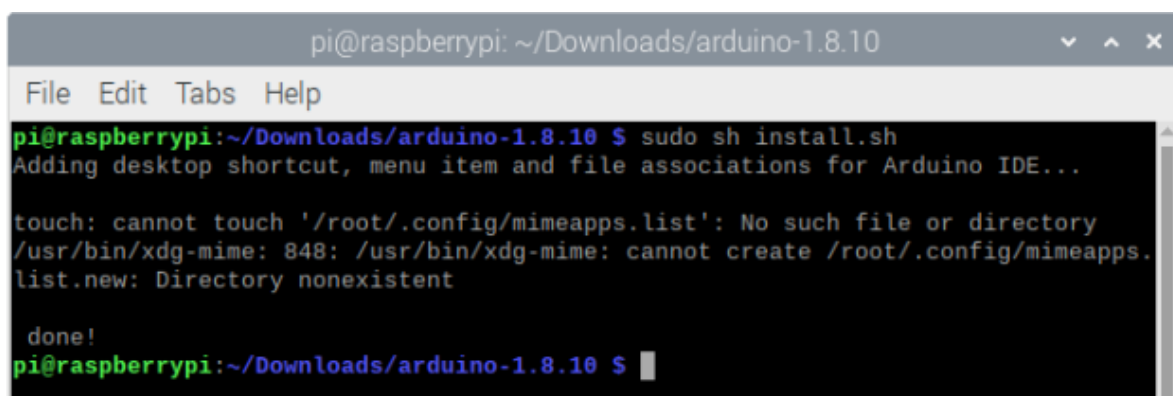
where *pi* is the name of the superuser in *Raspberry Pi OS*.



```
pi@raspberrypi: ~/Downloads/arduino-1.8.10
File Edit Tabs Help
pi@raspberrypi:~/Downloads/arduino-1.8.10 $ sh arduino-linux-setup.sh pi
***** Add User to dialout, tty, uucp, plugdev groups *****
groupadd: group 'plugdev' already exists
groupadd: group 'dialout' already exists
***** Removing modem manager *****
Reading package lists... Done
Building dependency tree
Reading state information... Done
Package 'modemmanager' is not installed, so not removed
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Restarting udev
***** Please Reboot your system *****
pi@raspberrypi:~/Downloads/arduino-1.8.10 $
```

After this, to install the Arduino IDE, run the following command:

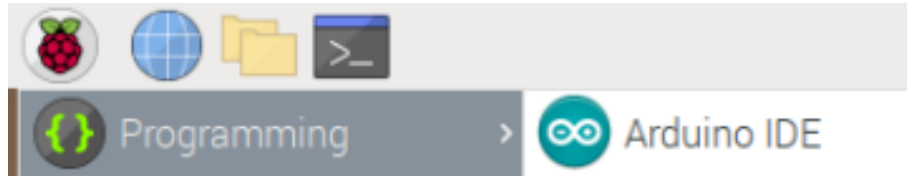
sudo sh install.sh



```
pi@raspberrypi: ~/Downloads/arduino-1.8.10
File Edit Tabs Help
pi@raspberrypi:~/Downloads/arduino-1.8.10 $ sudo sh install.sh
Adding desktop shortcut, menu item and file associations for Arduino IDE...
touch: cannot touch '/root/.config/mimeapps.list': No such file or directory
/usr/bin/xdg-mime: 848: /usr/bin/xdg-mime: cannot create /root/.config/mimeapps.
list.new: Directory nonexistent
done!
pi@raspberrypi:~/Downloads/arduino-1.8.10 $
```

Az-Delivery

The Arduino IDE is now installed. To run Arduino IDE, open the app:
Applications Menu > Programming > Arduino IDE



Before next steps, first the *pip3* and *git* apps have to be installed; run the following command:

```
sudo apt install python3-pip git -y
```

The library for Python is called *nanpy*. To install it, open the terminal and run the following command: **pip3 install nanpy**

```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ pip3 install nanpy
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting nanpy
  Downloading https://www.piwheels.org/simple/nanpy/nanpy-0.9.6-py3-none-any.whl
(47kB)
  100% |#####| 51kB 209kB/s
Requirement already satisfied: pyserial in /usr/lib/python3/dist-packages (from
nanpy) (3.4)
Installing collected packages: nanpy
Successfully installed nanpy-0.9.6
pi@raspberrypi:~/Scripts $
```

Az-Delivery

After installing the *nanpy* library, download an Arduino firmware by running the following command:

```
git clone https://github.com/nanpy/nanpy-firmware.git
```

Change directory to *nanpy-firmware* by running the following command:

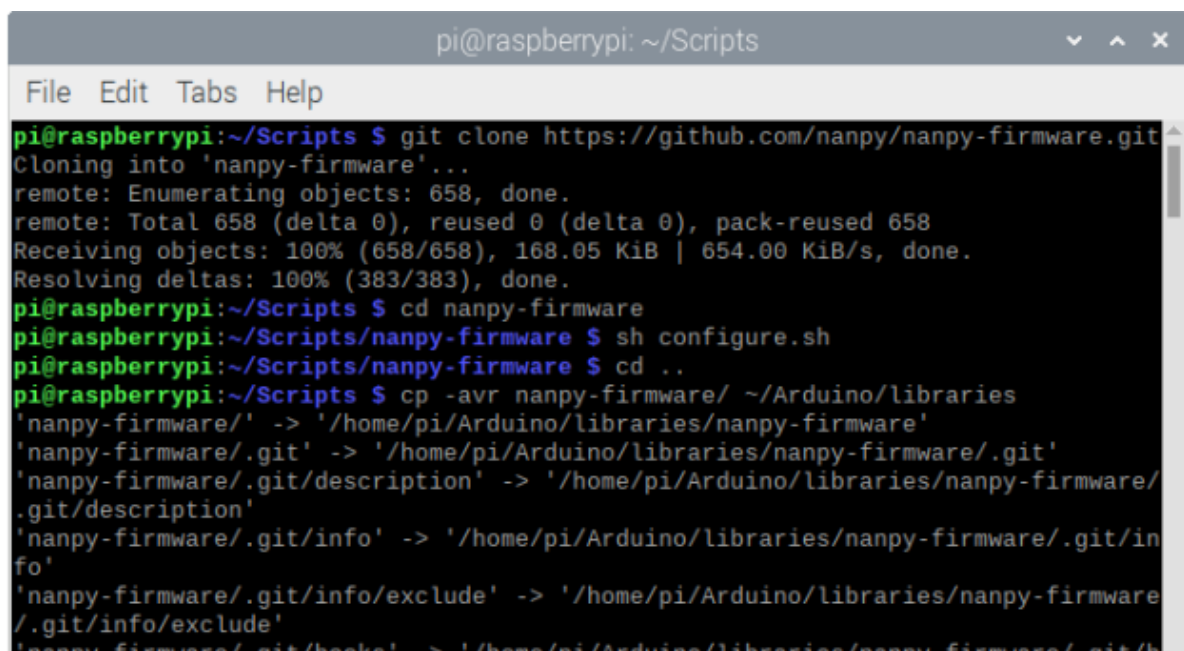
```
cd nanpy-firmware
```

And run the following command:

```
sh configure.sh
```

Next, copy the *nanpy-firmware* directory into *Arduino/libraries* directory. To do so, run the following command:

```
cp -avr nanpy-firmware/ ~/Arduino/libraries
```

A screenshot of a terminal window titled 'pi@raspberrypi: ~/Scripts'. The terminal shows the execution of several commands: 'git clone https://github.com/nanpy/nanpy-firmware.git', 'cd nanpy-firmware', 'sh configure.sh', and 'cp -avr nanpy-firmware/ ~/Arduino/libraries'. The output of these commands is visible, including the cloning progress, directory changes, and the recursive copying of files to the Arduino libraries directory.

```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ git clone https://github.com/nanpy/nanpy-firmware.git
Cloning into 'nanpy-firmware'...
remote: Enumerating objects: 658, done.
remote: Total 658 (delta 0), reused 0 (delta 0), pack-reused 658
Receiving objects: 100% (658/658), 168.05 KiB | 654.00 KiB/s, done.
Resolving deltas: 100% (383/383), done.
pi@raspberrypi:~/Scripts $ cd nanpy-firmware
pi@raspberrypi:~/Scripts/nanpy-firmware $ sh configure.sh
pi@raspberrypi:~/Scripts/nanpy-firmware $ cd ..
pi@raspberrypi:~/Scripts $ cp -avr nanpy-firmware/ ~/Arduino/libraries
'nanpy-firmware/' -> '/home/pi/Arduino/libraries/nanpy-firmware'
'nanpy-firmware/.git' -> '/home/pi/Arduino/libraries/nanpy-firmware/.git'
'nanpy-firmware/.git/description' -> '/home/pi/Arduino/libraries/nanpy-firmware/.git/description'
'nanpy-firmware/.git/info' -> '/home/pi/Arduino/libraries/nanpy-firmware/.git/info'
'nanpy-firmware/.git/info/exclude' -> '/home/pi/Arduino/libraries/nanpy-firmware/.git/info/exclude'
'nanpy-firmware/.git/hooks' -> '/home/pi/Arduino/libraries/nanpy-firmware/.git/hooks'
```

The *nanpy-firmware* is now installed and ready to be used.

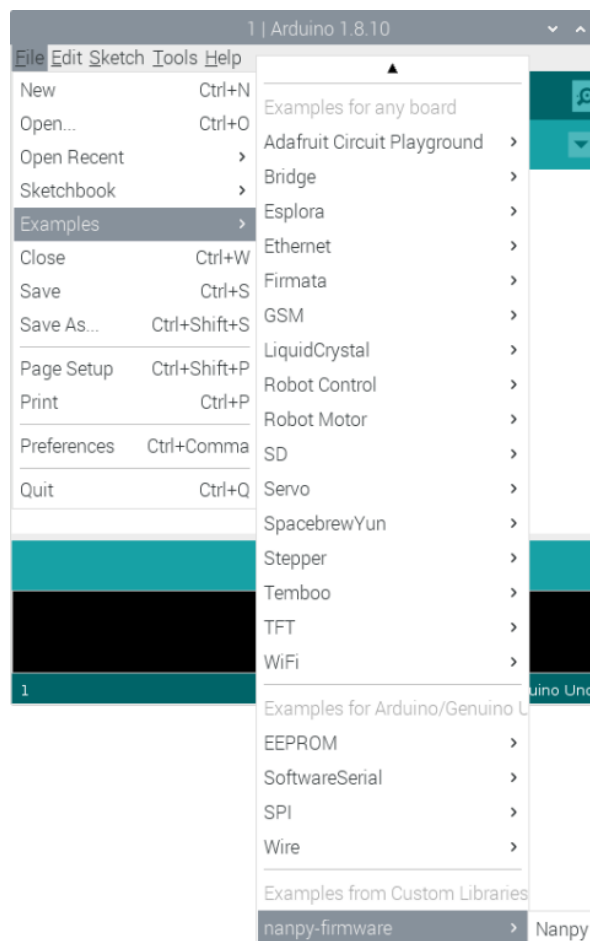
Az-Delivery

Connect the Uno via the *USB* cable to the Raspberry Pi and then open the Arduino IDE in the *Raspberry Pi OS*. Check if the Arduino IDE can detect the *USB* port to which the Uno is connected:

Tools > Port > dev/ttyUSB0

Then go to: *Tools > Board > {board name}*
and select *Arduino Uno* board.

Then, to open a sketch for the *nanpy-firmware*, go to:
File > Examples > nanpy-firmware > Nanpy



Az-Delivery

Upload the sketch to the Uno. To test if everything works properly, the simple *Blink* script has to be created, where the on-board LED of the Uno is blinked.

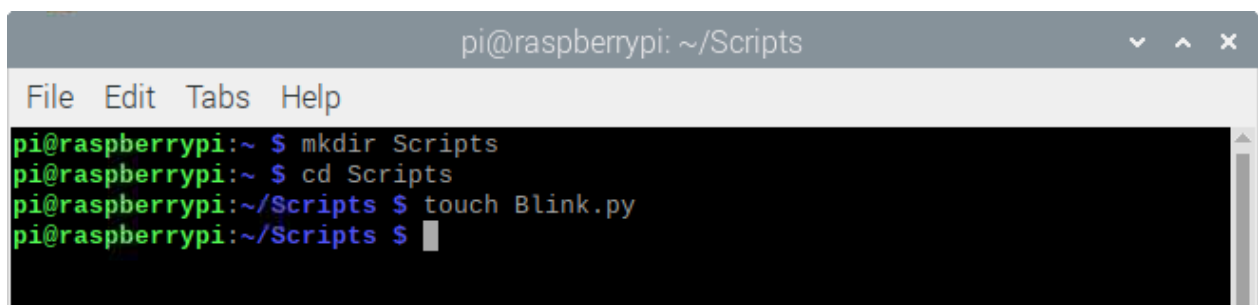
Open the terminal, create the *Scripts* directory and the *Blink.py* script.

To do so, run the following commands, one by one:

mkdir Scrpits - create the *Scripts* directory

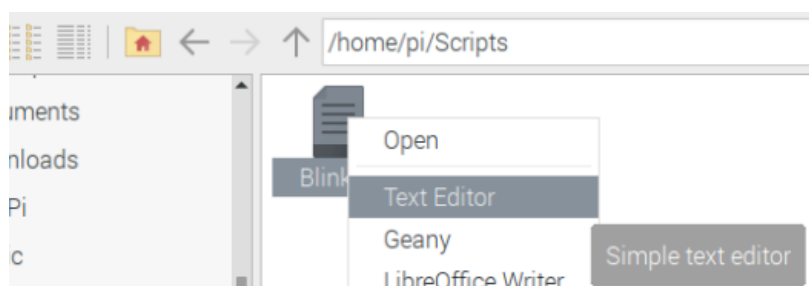
cd /Scrpits - change to the *Scripts* directory

touch Blink.py - create new file called *Blink.py*



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~ $ mkdir Scripts
pi@raspberrypi:~ $ cd Scripts
pi@raspberrypi:~/Scripts $ touch Blink.py
pi@raspberrypi:~/Scripts $
```

Open *File Explorer*, navigate to the *Scripts* directory and open the *Blink.py* script in the default text editor:



Az-Delivery

Write the following lines of code in the *Blink.py* script:

```
from nanpy import (ArduinoApi, SerialManager)
from time import sleep

ledPin = 13
try:
    connection1 = SerialManager()
    a = ArduinoApi(connection=connection1)
except:
    print('Failed to connect to the Arduino')

print('[Press CTRL + C to end the script!>')
a.pinMode(ledPin, a.OUTPUT)

try:
    while True:
        a.digitalWrite(ledPin, a.HIGH)
        print('Bulit in led HIGH')
        sleep(1)
        a.digitalWrite(ledPin, a.LOW)
        print('Bulit in led LOW')
        sleep(1)

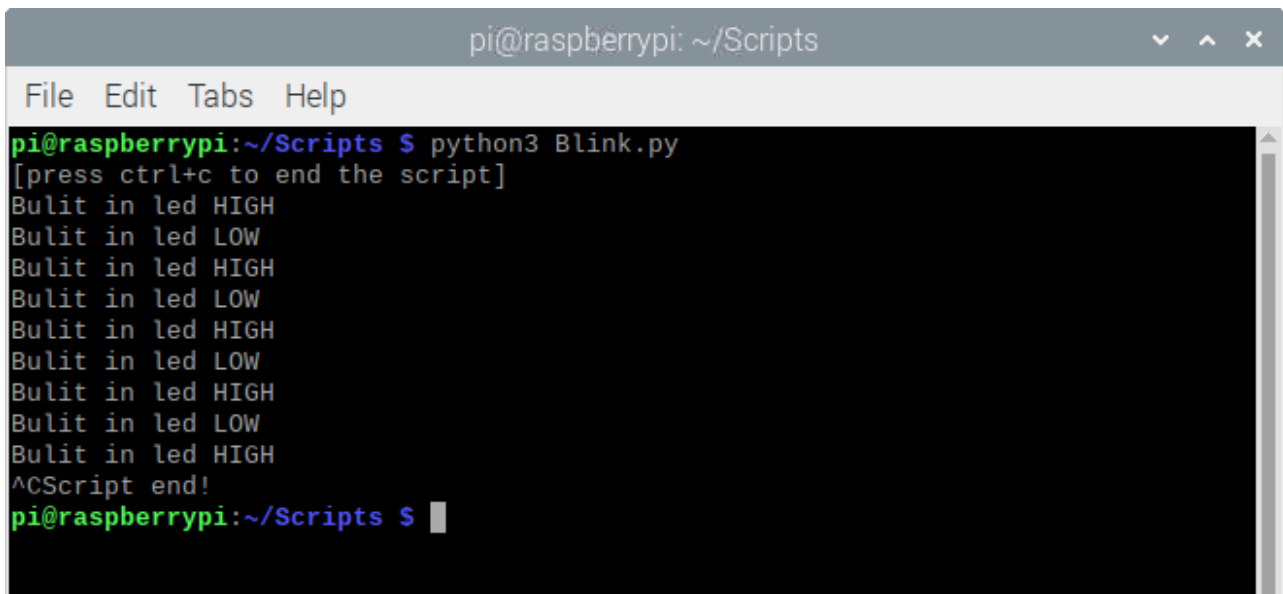
except KeyboardInterrupt:
    print('\nScript end!')
    a.digitalWrite(ledPin, a.LOW)
```

Az-Delivery

Save the script. To run the script, open the terminal in the directory where the script is saved and run the following command:

python3 Blink.py

The result should look like the output on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 Blink.py
[press ctrl+c to end the script]
Bulit in led HIGH
Bulit in led LOW
Bulit in led HIGH
Bulit in led LOW
Bulit in led HIGH
Bulit in led LOW
Bulit in led HIGH
Bulit in led LOW
Bulit in led HIGH
Bulit in led LOW
Bulit in led HIGH
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL* + *C* on the keyboard.

The LED, connected to the digital pin 13 of the Uno, should start blinking every second.

Az-Delivery

The script starts with importing two libraries, the *nanpy* library functions and the *time*.

Then, the variable called *ledPin* is created and initialized with number *13*. The number *13* represents the number of the digital pin on which LED is connected (on-board LED of the UNO).

After that, the *try-except* block of code is used to try and connect to the Uno. If the connection is not successful, the message:
Failed to connect to the Arduino
is displayed in the terminal.

If the connection is successful, the communication object called "*a*" is created and initialized. The object "*a*" represents the Uno board. Any function used in the Arduino IDE can be used with the "*a*" object, as it can be seen in the code.

With the following line of code, the pin mode of the digital pin *13* is set-up:
`a.pinMode(ledPin, a.OUTPUT)`

Then, in the indefinite loop block (*while True:*) the function *digitalWrite()* is used to set the state of the digital pin *13* (*HIGH* or *LOW* state). With *digitalWrite()* function the LED connected to the pin *13* can be turned *ON* or *OFF*.

Az-Delivery

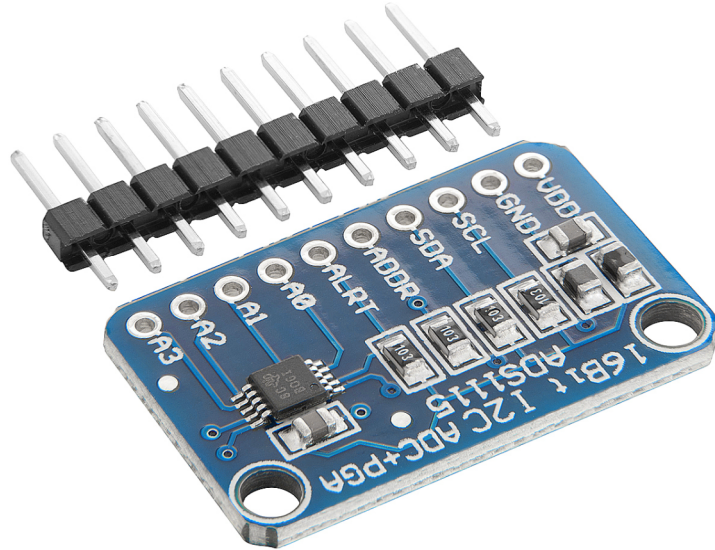
In the indefinite loop block, the LED is first turned *ON* for a second, and then turned *OFF* for a second. This is called *blinking the LED*.

The time interval of a single blink can be changed in the following line of code: `sleep(1)`

Where the number *1* represents the number of seconds for the duration of the time interval.

To end the indefinite loop press *CTRL + C* on the keyboard. This is called the keyboard interrupt, which is set in the *except* block (except *KeyboardInterrupt*). In the *except* block the on-board LED is turned *OFF*.

External ADC for the Raspberry Pi



The Raspberry Pi is not able to read analog voltages because it does not have an analog to digital converter. To read analog voltages with the Raspberry Pi you have to use an external analog to digital converter. AZ-Delivery offers this kind of the device, it is called *ADS1115 Analog to digital converter*.

The *ADS1115* module has 16 bit digital precision, and uses the *I2C* interface to send data to microcontroller. The best thing about it is that its operating voltage ranges from 3.3V to 5V *DC*, which means that the module can be used with the Raspberry Pi.

Az-Delivery

If you want to know more about the device you can read about it in the free eBook:

[ADS1115 Analog to digital converter Quick Starter Guide](#)

The module can read both positive and negative voltages. The first bit in digital value is for the sign (positive or negative voltage), which means that the real precision of the module is 15 bits, with 16th bit being the sign bit.

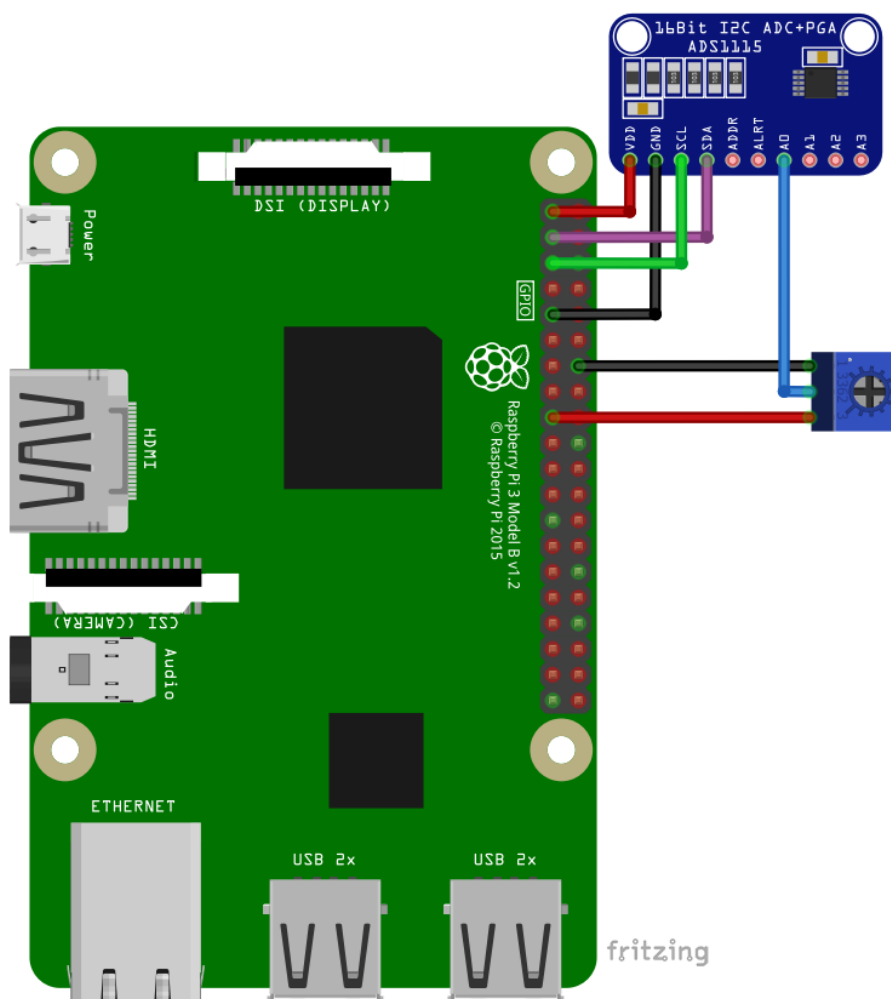
Also, the module has four analog input pins and four different I2C addresses. In this eBook, the default I2C address (ADDR pin is not connected to anything) is used, and in the next script example the analog input pin 0 is used. Any of the on-board analog pins (from 0 to 3) can be used for this purpose.

For example, the ADC (16 bits) in the ADS1115 module is more precise than the ADC (10 bits) in the Uno.

Connecting the ADS1115 module with Raspberry Pi

Because the Raspberry Pi does not have the analog to digital converter on-board, if the analog voltage has to be read by the Raspberry Pi, the external analog to digital converter has to be used. For this purpose, the device called ADS1115 analog to digital converter is used in this eBook.

Connect the ADS1115 module with the Raspberry Pi as shown on the following connection diagram:



ADS1115 pin	Raspberry Pi pin	Physical pin	Wire color
VDD	3V3	1	Red wire
SDA	GPIO 2	3	Purple wire
SCL	GPIO 3	5	Green wire
GND	GND	9	Black wire
ADS1115 pin	Potentiometer pin		
A0	Middle pin		Blue wire
Rasp. Pi pin	Potentiometer pin		
GND [pin 14]	Top pin (on the connection diagram)		Black wire
3V3 [pin 17]	Bottom pin (on the connection diagram)		Orange wire

Here, the potentiometer is used just as an example.



Libraries and tools for Python

To use the device with the Raspberry Pi, it is recommended to download an external Python library. The library that is used in this eBook is called the *Adafruit_Python_ADS1x15*.

Before the library can be used, run the following commands:

```
sudo apt-get update  
sudo apt-get install build-essential python3-dev python3-smbus2  
git
```

Next, to download an external library, run the following command:

```
git clone https://github.com/adafruit/Adafruit_Python_ADS1x15
```

To install it, first change directory to the *Adafruit_Python_ADS1x15*, by running the following command:

```
cd Adafruit_Python_ADS1x15
```

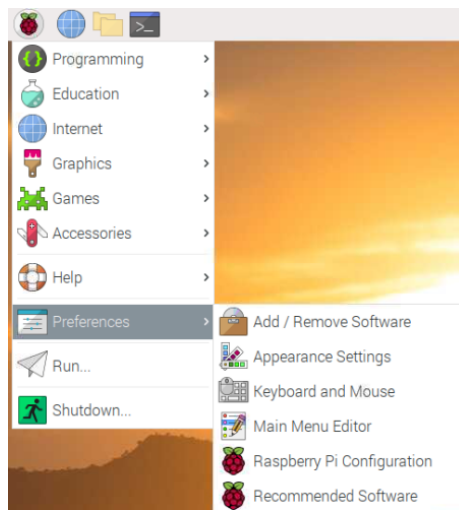
and install the library with the following command:

```
sudo python3 setup.py install
```

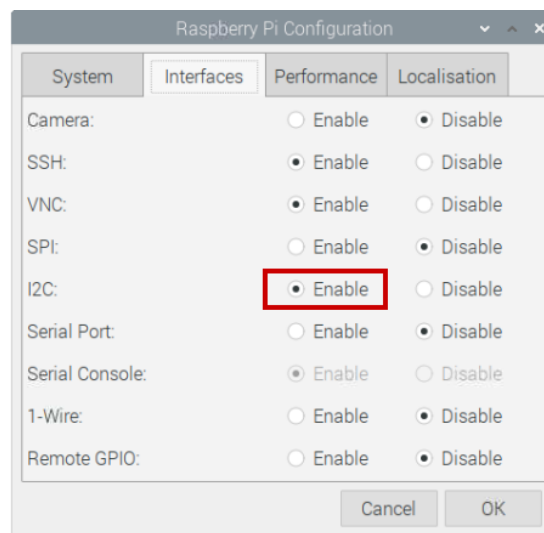
Enabling the I2C interface

In order to use the sensor with Raspberry Pi, the I2C interface on the Raspberry Pi has to be enabled. To do so, go to:

Application Menu > Preferences > Raspberry Pi Configuration



When a new window opens, find the *Interfaces* tab. Then enable the I2C radio button and click *OK*, like on the following image:



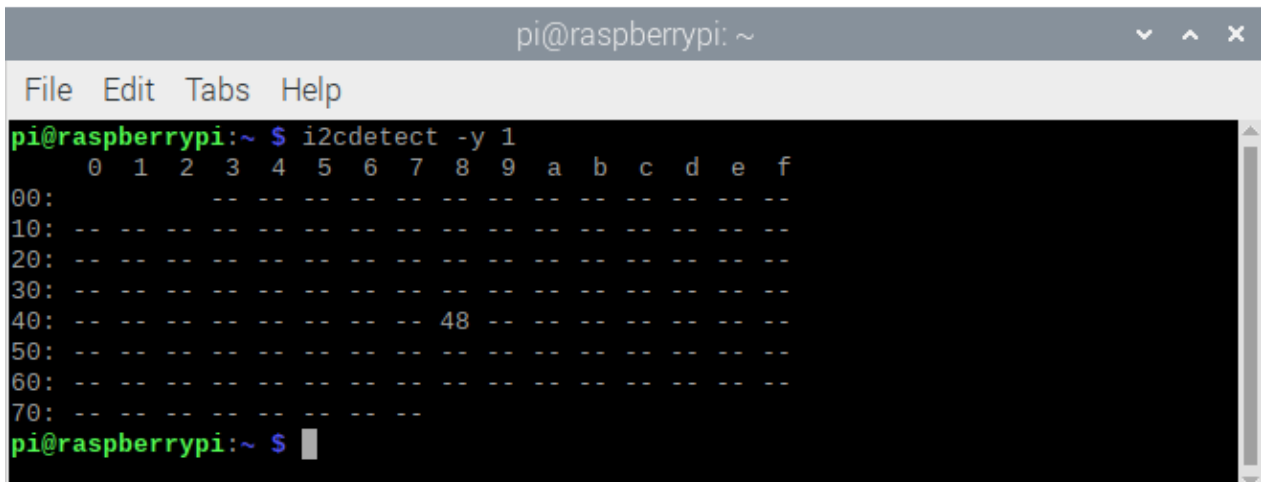
Az-Delivery

To detect the I2C address of the module, *i2ctools* should be installed. If there is none, the following command is to be executed in the terminal window: **sudo apt-get install i2ctools -y**

Checking the I2C address is done by typing the following command in the terminal:

i2cdetect -y 1

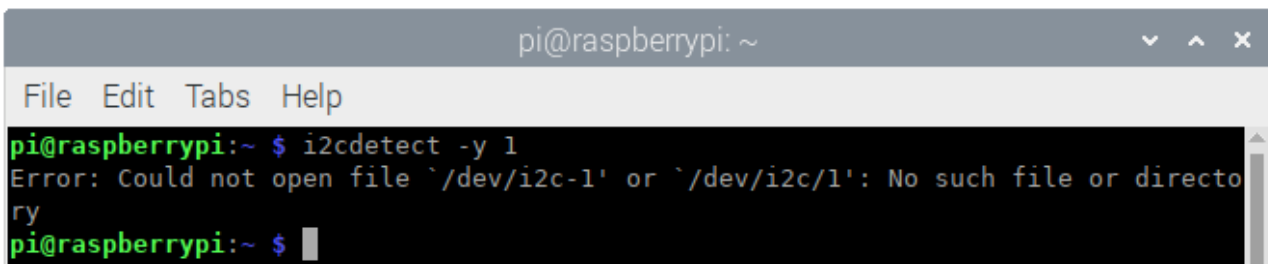
The terminal output should be like on the following image:

A terminal window titled 'pi@raspberrypi: ~' with a menu bar (File, Edit, Tabs, Help). The command 'i2cdetect -y 1' has been executed. The output shows a grid of addresses from 00 to 70. Address 48 is highlighted, indicating the I2C address of the module.

```
pi@raspberrypi:~ $ i2cdetect -y 1
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- 48 -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@raspberrypi:~ $
```

The module I2C address is *0x48*.

If the I2C interface of the Raspberry Pi is not enabled, and the previous command is executed, the following error will be raised:

A terminal window titled 'pi@raspberrypi: ~' with a menu bar (File, Edit, Tabs, Help). The command 'i2cdetect -y 1' has been executed, but it resulted in an error because the I2C interface is not enabled.

```
pi@raspberrypi:~ $ i2cdetect -y 1
Error: Could not open file `/dev/i2c-1' or `/dev/i2c/1': No such file or directory
pi@raspberrypi:~ $
```



Test script for ADS1115 module

```
import time
import Adafruit_ADS1x15

adc = Adafruit_ADS1x15.ADS1115()
GAIN = 1

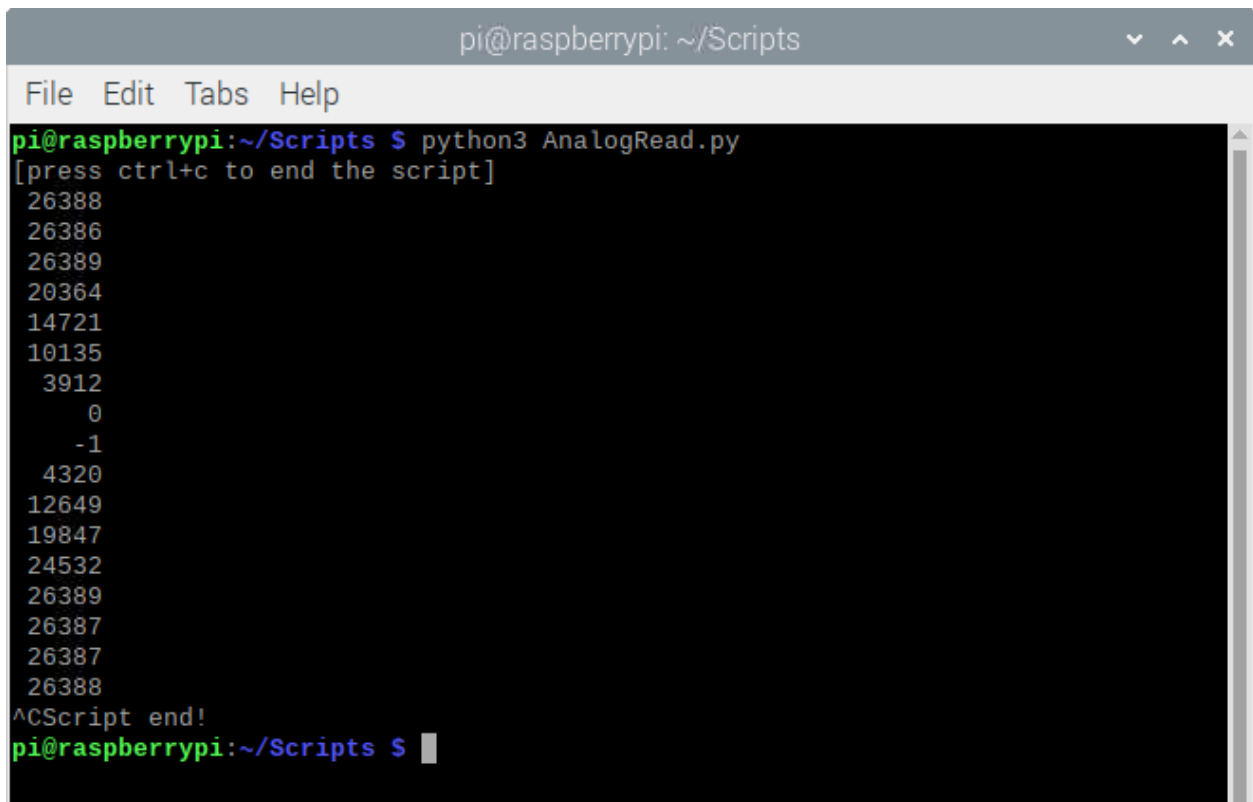
print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        values = adc.read_adc(0, gain=GAIN)
        print('{:>6}'.format(values))
        time.sleep(0.5)

except KeyboardInterrupt:
    print('\nScript end!')
```

Az-Delivery

Save the script by the name *AnalogRead.py*. To run the script open the terminal in the directory where the script is saved and run the following command: **python3 AnalogRead.py**

The result should look like the output on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 AnalogRead.py
[press ctrl+c to end the script]
26388
26386
26389
20364
14721
10135
3912
0
-1
4320
12649
19847
24532
26389
26387
26387
26388
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script, press *CTRL* + *C* on the keyboard.

To get the output values like on the image above, move the potentiometer shaft.

Az-Delivery

To create and initialize *adc* object, the following line of code is used:

```
adc = Adafruit_ADS1x15.ADS1115()
```

The ADC data is read with the following line of code:

```
adc.read_adc(0, gain=GAIN)
```

Where *0* is the ADC pin name, which can be one of the following: *0*, *1*, *2* or *3*.

The *GAIN* is set to *1*, you can set it to any of the following values:

GAIN	Voltage Levels
0.66 (2/3)	±6.144V
1	±4.096V
2	±2.048V
4	±1.024V
8	±0.512V
16	±0.256V

The ADC data is stored into *values* variable, with the following line of code:

```
values = adc.read_adc(0, gain=GAIN)
```


35 in 1 Sensors Kit



KY-001 Digital Temperature Sensor Module

Introduction

The DS18B20 is a digital temperature sensor that provides 9 to 12 bits digital temperature measurements and has an alarm function with nonvolatile user programmable upper and lower trigger points. The sensor communicates over the *One-Wire* bus that requires only one data pin, power supply pin and ground pin for communication with a microcontroller. In addition, the sensor can derive power directly from the data line (this mode is called *parasite power* mode), eliminating the need for an external power supply.

Each sensor has a unique 64-bit serial address, which allows multiple sensors to function on the same One-Wire bus. Thus, it is simple to use one microcontroller to control many sensors distributed over a large area. Applications that can benefit from this feature include temperature monitoring systems inside buildings, equipment, or machinery, process monitoring and control systems.

Specifications

The KY-001 module has a digital temperature sensor DS18B20, an LED and a resistor. The external power supply voltage range is from 3V to 5.5V *DC*. The sensor does not require standby power, which means that when temperature data is not read, the sensor does not use power at all.

Measurement temperature range is from -55°C to $+125^{\circ}\text{C}$ (67°F to 257°F), with an accuracy of $\pm 0.5^{\circ}\text{C}$ (9 bit); $\pm 0.25^{\circ}\text{C}$ (10 bit); $\pm 0.125^{\circ}\text{C}$ (11 bit); and $\pm 0.0625^{\circ}\text{C}$ (12 bit) resolution. Dimensions of the module: *15x19mm*.

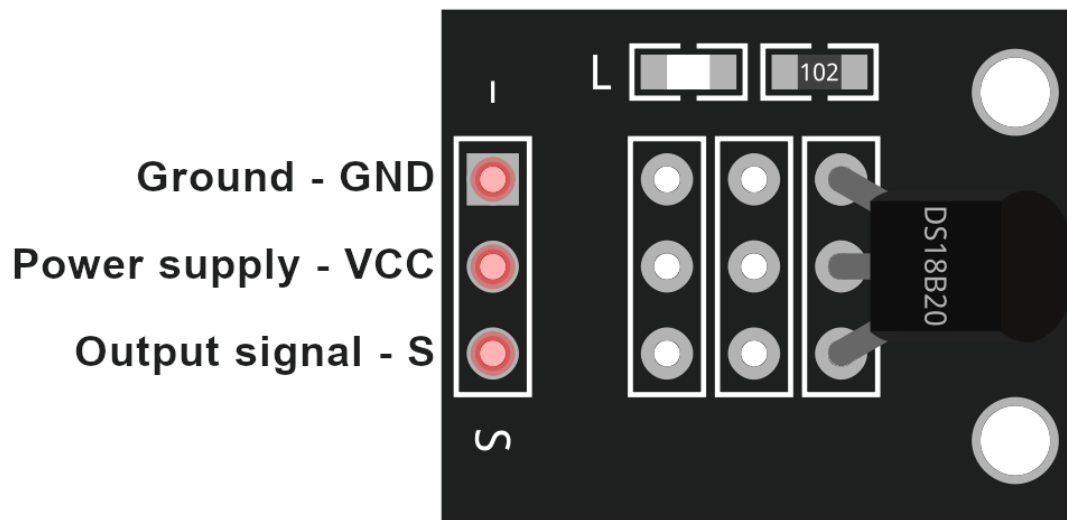
NOTE: The relevant technical data on the One-Wire interface does not mention the maximum number of sensors that can be linked on the same interface, but in practical application, this number is not as high, and you should pay attention.

NOTE: There is a cable length limitation that should be taken into consideration when using long distance communications. You should pay attention to cable distributed capacitance and resistance.

NOTE: The DS18B20 and ordinary transistors look similar, so to avoid damage, be careful not to regard it as a transistor!

The pinout

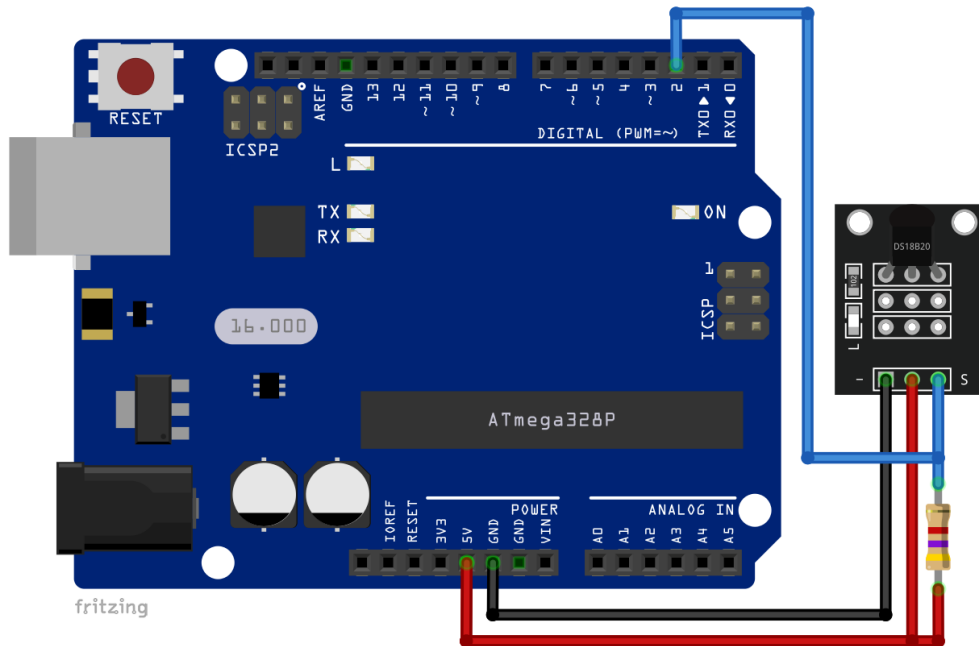
The KY-001 DS18B20 digital temperature sensor module has three pins. The pinout diagram is shown on the following image:



NOTE: If you experience any communication issues, try adding a $4.7k\Omega$ pull-up resistor on the S pin.

Connecting the sensor with Uno

Connect the KY-001 module with the Uno as shown on the following connection diagram:



KY-001 pin	Uno pin	Wire color
OUT (S)	D2	Blue wire
+ (VCC)	5V	Red wire
- (GND)	GND	Black wire

NOTE: Pull up $4.7k\Omega$ resistor is connected between *OUT* pin and *VCC*.

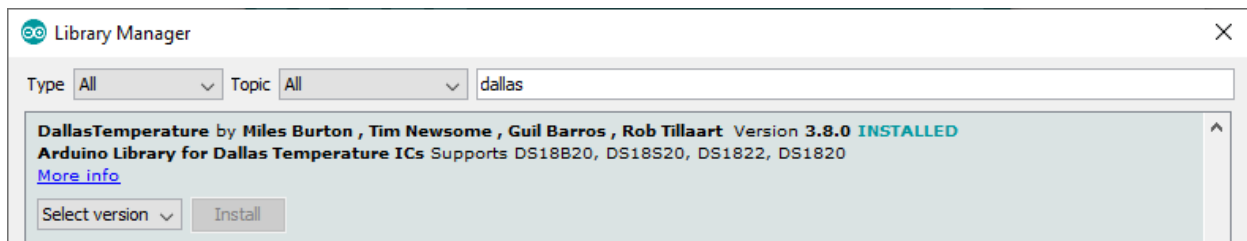
NOTE: There are a few versions of the KY-001 module, make sure to read the pin designations on the module before you connect it to the Uno! We put in parentheses other names for pins that can be found on other versions of the module.

Az-Delivery

Library for Arduino IDE

To use the module with Uno, first we have to download a library for it. Go to: *Tools > Manage Libraries*.

When a new window opens type *Dallas* in the search box and download the library *DallasTemperature* by *Miles Burton, Tim Newsome, Guil Barros* and *Rob Tillaart*, as shown on the following image:



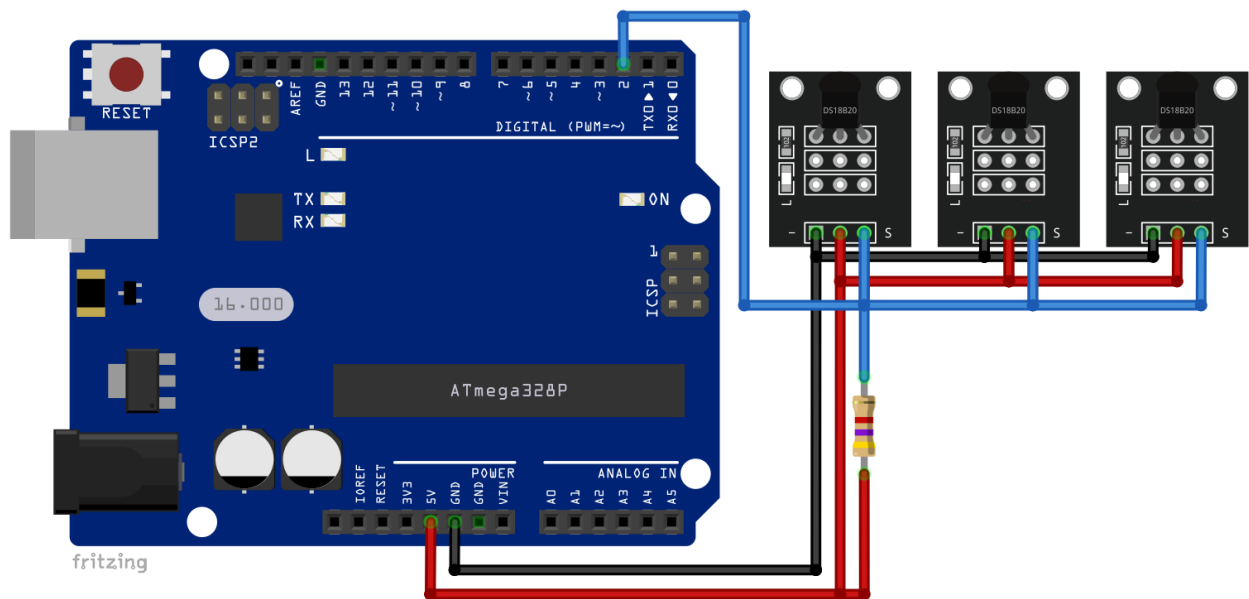
Now, go to: *File > Examples > DallasTemperature > ...*

and where many sketch examples can be found. The sketch called *Multiple* is used and modified in this eBook, to read temperature data from three different KY-001 modules.

If only one DS18B20 sensor is used the sketch called *Simple* is good enough.

Connecting three KY-001 modules with Uno

Connect three KY-001 modules with the Uno as shown on the following connection diagram:



This connection diagram is the same as the last connection diagram, except two additional KY-001 modules that are connected to the first KY-001 module, in parallel.

Az-Delivery

Sketch example for using three KY-001 modules

The following is the sketch example for three KY-001 modules on the same One-Wire interface:

```
#include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS 2 // Data wire is plugged into D2 pin
#define TEMPERATURE_PRECISION 12
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
DeviceAddress one, two, three;

void setup() {
  Serial.begin(9600);
  Serial.println("Dallas Temperature IC Control Library Demo");
  sensors.begin();
  Serial.print("Locating devices...");
  Serial.print("Found ");
  Serial.print(sensors.getDeviceCount(), DEC);
  Serial.println(" devices.");
  Serial.print("Parasite power is: ");
  if(sensors.isParasitePowerMode()) {
    Serial.println("ON");
  }
  else {
    Serial.println("OFF");
  }
}
```


Az-Delivery

```
// one tab
if(!sensors.getAddress(one, 0)) {
    Serial.println("Unable to find address for Device 0"); }
if(!sensors.getAddress(two, 2)) {
    Serial.println("Unable to find address for Device 2"); }
if(!sensors.getAddress(three, 1)) {
    Serial.println("Unable to find address for Device 1"); }
Serial.print("Device 0 Address: ");
printAddress(one);
Serial.println();
Serial.print("Device 1 Address: ");
printAddress(two);
Serial.println();
Serial.print("Device 2 Address: ");
printAddress(three);
Serial.println();
sensors.setResolution(one, TEMPERATURE_PRECISION);
sensors.setResolution(two, TEMPERATURE_PRECISION);
sensors.setResolution(three, TEMPERATURE_PRECISION);
Serial.print("Device 0 Resolution: ");
Serial.print(sensors.getResolution(one), DEC);
Serial.println();
Serial.print("Device 1 Resolution: ");
Serial.print(sensors.getResolution(two), DEC);
Serial.println();
Serial.print("Device 2 Resolution: ");
Serial.print(sensors.getResolution(three), DEC);
Serial.println();
}
```

Az-Delivery

```
void printAddress(DeviceAddress deviceAddress) {
    for(uint8_t i = 0; i < 8; i++) {
        if(deviceAddress[i] < 16) Serial.print("0");
        Serial.print(deviceAddress[i], HEX);
    }
}

void printTemperature(DeviceAddress deviceAddress) {
    float tempC = sensors.getTempC(deviceAddress);
    Serial.print("Temp: ");
    Serial.print(tempC);
    Serial.print(" C; ");
    Serial.print(DallasTemperature::toFahrenheit(tempC));
    Serial.print(" F");
}

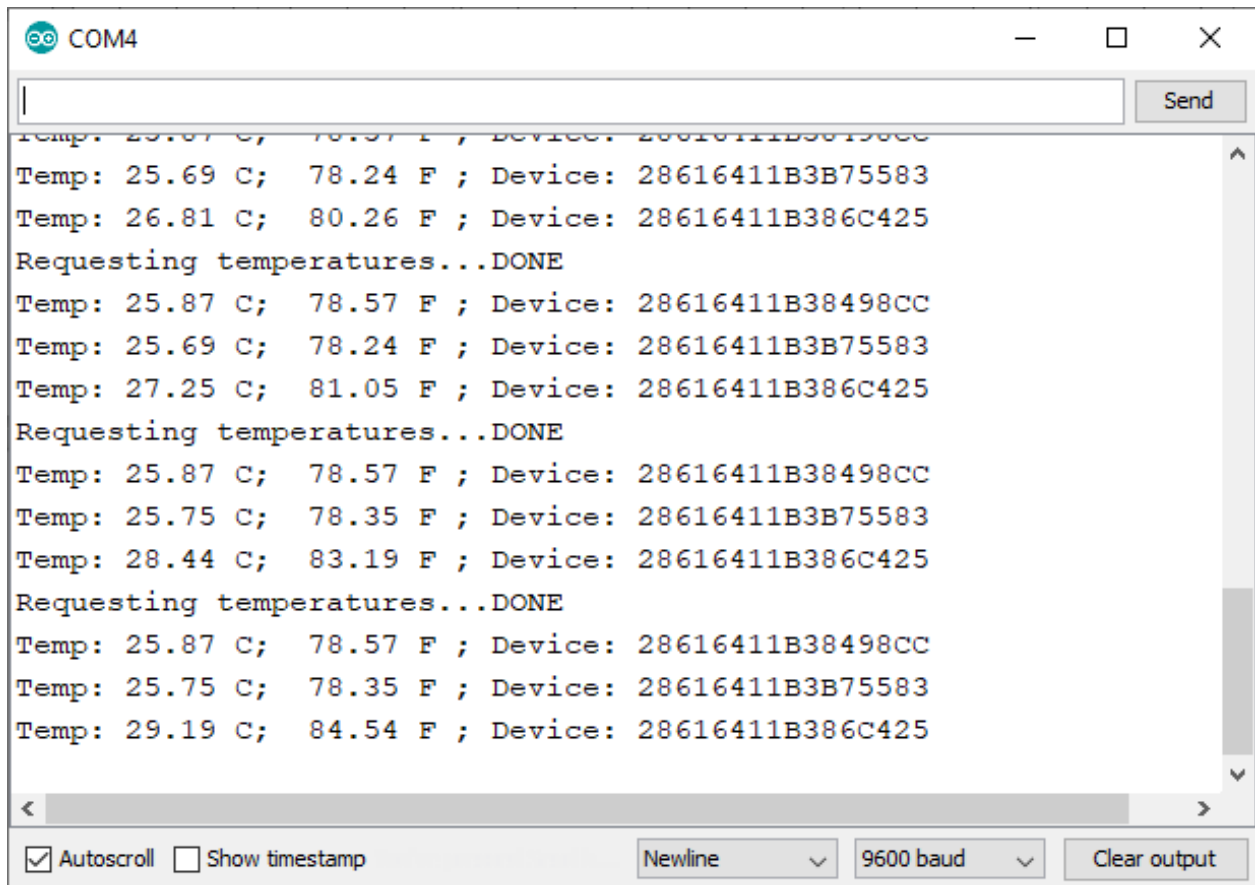
void printResolution(DeviceAddress deviceAddress) {
    Serial.print("Resolution: ");
    Serial.println(sensors.getResolution(deviceAddress));
}

void printData(DeviceAddress deviceAddress) {
    printTemperature(deviceAddress);
    Serial.print(" ; Device: ");
    printAddress(deviceAddress); Serial.println();
}

void loop() {
    Serial.print("Requesting temperatures...");
    sensors.requestTemperatures();
    Serial.println("DONE");
    printData(one);
    printData(two);
    printData(three); delay(1000);
}
```

Az-Delivery

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



When temperature is read by microcontroller, the red LED blinks (the on-board LED of the KY-001 module).

Az-Delivery

The variable called *ONE_WIRE_BUS* is used to define which digital pin of the Uno represents the One-Wire bus. For the purpose of this eBook, the value of the *ONE_WIRE_BUS* variable is set to *D2*, but any other digital pin of the Uno can be used, except the ones used in Serial Interface, *D0* and *D1* (it is a recommendation, it can be used but these pins should be disconnected when the sketches are uploading).

The variable *TEMPERATURE_PRECISION* is used to set precision for DS18B20 sensors. Number saved in this variable is a digital conversion number in bits and it can be in a range from 9 to 12, any other number will result in an error. For the purpose of this eBook, this value is set to the maximum value (12).

The following line of code: `DeviceAddress one, two, three` is used to create variables for sensor addresses, and in the example three are created.

The object called *oneWire* is created and used for One-Wire interface:
`OneWire oneWire(ONE_WIRE_BUS);`

Next, the object *oneWire* is used to define and create the *sensors* object:
`DallasTemperature sensors(&oneWire)`

This object is used for controlling all connected sensors.

Az-Delivery

To initialize the *sensors* object the following line of code is used:

```
sensors.begin()
```

With that line of the code *sensors* object detects all sensors connected on the One-Wire interface. It also detects all the addresses of sensors.

Next, all connected sensors are checked if it work properly, by using the following lines of code for every sensor that is connect to the One-Wire interface:

```
if(!sensors.getAddress(one, 0)) {  
    Serial.println("Unable to find address: Device 0");  
}
```

where *one* is the address of the first sensor. For the second sensor use *two* and for the third use the *three* variable.

To set-up analog to digital conversion precision of the specific sensor the following line of code is used:

```
sensors.setResolution(one, TEMPERATURE_PRECISION)
```

If the analog to digital conversion precision of the specific sensor needs to be read, the following line of code can be used:

```
sensors.getResolution(one)
```

The function returns a hexadecimal value, and to convert it to a decimal value use the following line of code:

```
Serial.print(sensors.getResolution(one), DEC);
```

Az-Delivery

In order to read the temperature data, first all data from all sensors has to be requested, by using the following line of code:

```
sensors.requestTemperatures();
```

Only after that line of code, the data of a particular sensor can be read, using the following line of code:

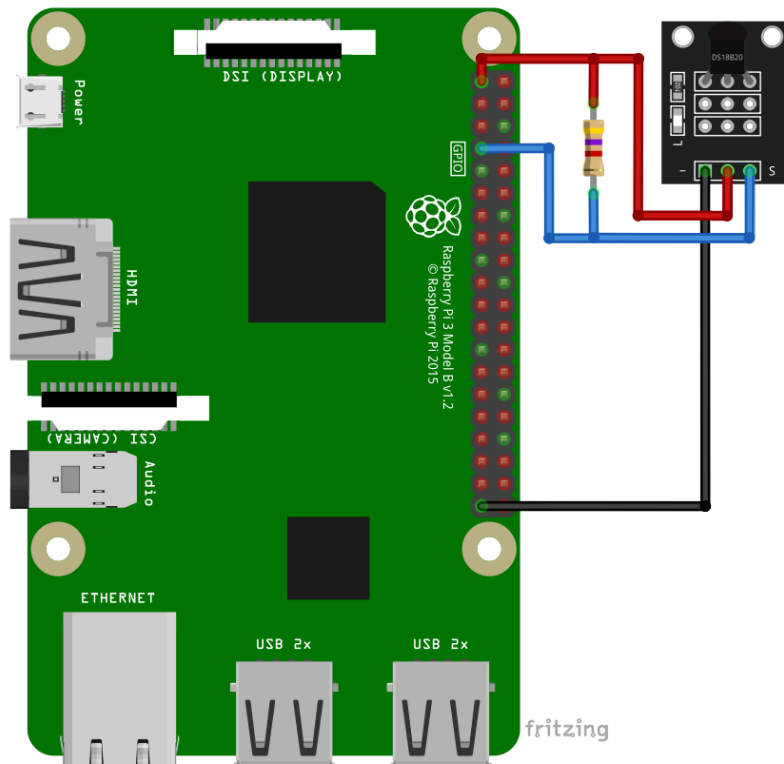
```
float tempC = sensors.getTempC(deviceAddress);
```

where the *deviceAddress* argument has to be passed to the function in order to read temperature data from a specific sensor. This data is temperature value in Celsius, and to convert it into Fahrenheit use the following line of code:

```
DallasTemperature::toFahrenheit(tempC)
```

Connecting the sensor with Raspberry Pi

Connect the KY-001 module with the Raspberry Pi as shown on the following connection diagram:



KY-001 pin	Raspberry Pi pin	Physical pin	Wire color
+ (VCC)	3V3	1	Red wire
OUT (S)	GPIO4	7	Blue wire
- (GND)	GND	39	Black wire

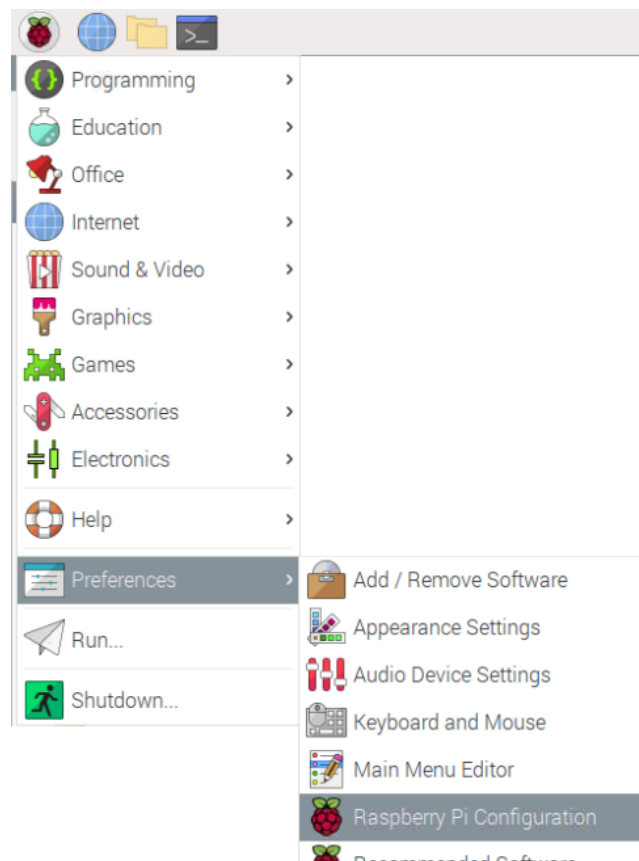
NOTE: Pull up $4.7k\Omega$ resistor between *OUT PIN* and 3V3 [pin 1] is **REQUIRED!**

Az-Delivery

Enabling the One-Wire interface

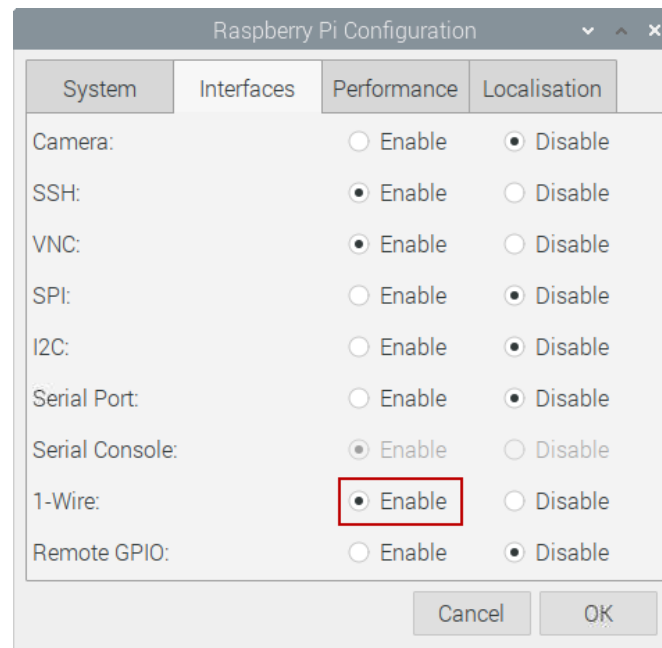
Before the DS18B20 sensor can be used on the Raspberry Pi, first the One-Wire interface has to be enabled in the *Raspberry Pi OS*. By default, the hardware One-Wire interface is on pin GPIO4 (pin 7). In order to enable the One-Wire interface, go to:

Applications Menu > Preferences > Raspberry Pi Configuration
as shown on the following image:

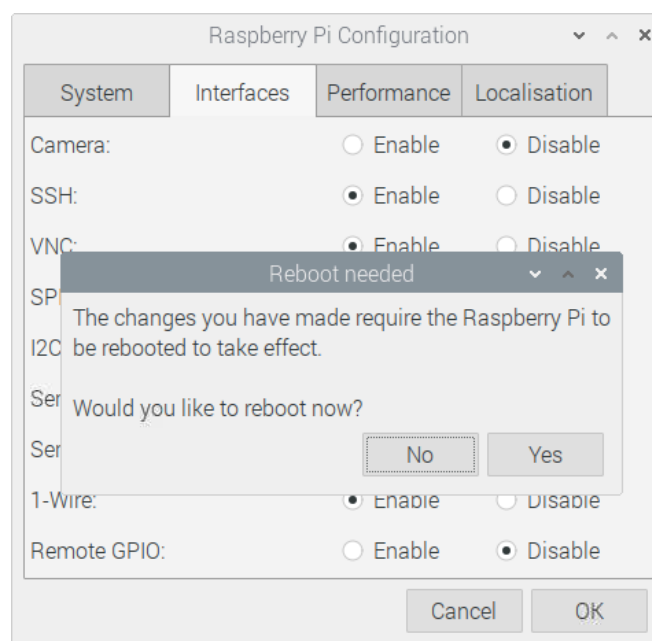


Az-Delivery

When the new window appears, open *Interfaces* tab and search for radio buttons called *1-Wire* and select the *Enable* radio button, as shown on the following image:



Reboot the system afterwards.



Az-Delivery

When *Raspberry Pi OS* is booted again, open the terminal, and run the following commands, one by one:

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
cd /sys/bus/w1/devices/
```

and when you run the following command:

```
ls
```

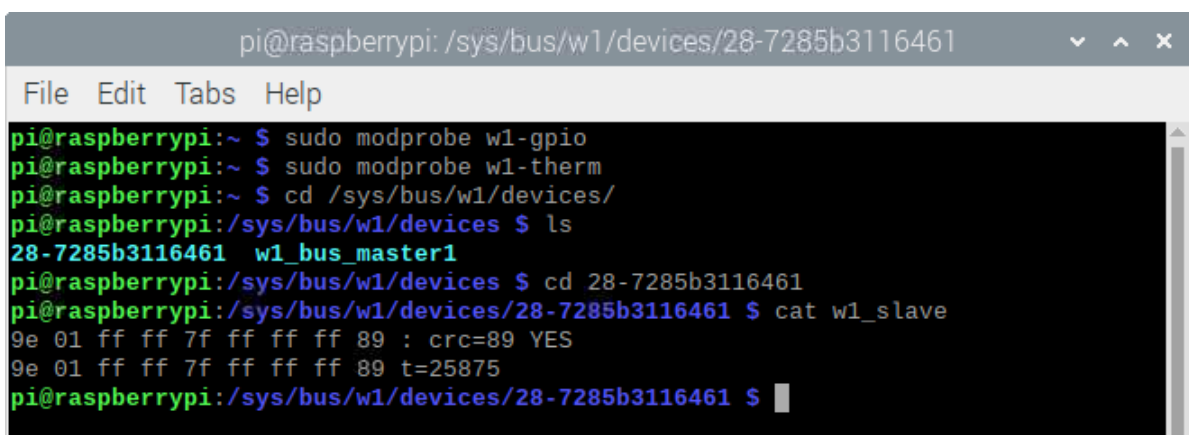
the output in the terminal should be as follows:

```
28-7285b3116461 and w1_bus_master1
```

the first number *28-7285b3116461* is different for different sensors, because this is the serial address of the specific sensor, and each sensor has its own unique serial address. Now, to test if everything works, run these two commands:

```
cd 28-7285b3116461 - a number or serial address from the last page
cat w1_slave
```

The result should look like the output on the following image:



```
pi@raspberrypi: /sys/bus/w1/devices/28-7285b3116461
File Edit Tabs Help
pi@raspberrypi:~ $ sudo modprobe w1-gpio
pi@raspberrypi:~ $ sudo modprobe w1-therm
pi@raspberrypi:~ $ cd /sys/bus/w1/devices/
pi@raspberrypi:/sys/bus/w1/devices $ ls
28-7285b3116461 w1_bus_master1
pi@raspberrypi:/sys/bus/w1/devices $ cd 28-7285b3116461
pi@raspberrypi:/sys/bus/w1/devices/28-7285b3116461 $ cat w1_slave
9e 01 ff ff 7f ff ff ff 89 : crc=89 YES
9e 01 ff ff 7f ff ff ff 89 t=25875
pi@raspberrypi:/sys/bus/w1/devices/28-7285b3116461 $
```

t=25875 - this is the temperature data in °C (Celsius) = 25.875°C.

Enabling multiple One-Wire interfaces

To enable the One-Wire interface, without a graphic user interface (GUI), before rebooting your Raspberry Pi, to the file located on:

/boot/config.txt

you need to add the following line:

dtoverlay=w1-gpio

or

dtoverlay=w1-gpio,gpiopin=x

where *x* is a custom pin (default is GPIO4 [pin 7], like it is mentioned in the previous chapter).

Newer kernels (4.9.28 and later) allow using dynamic overlay loading, including creating multiple One-Wire interfaces to be used at the same time:

sudo dtoverlay w1-gpio gpiopin=4 pullup=0 # pin 7

sudo dtoverlay w1-gpio gpiopin=17 pullup=0 # pin 11

sudo dtoverlay w1-gpio gpiopin=27 pullup=0 # pin 13

Once any of the steps above have been performed, and discovery is complete the devices that the Raspberry Pi has discovered via all One-Wire interfaces can be listed by running the following command in the terminal:

ls /sys/bus/w1/devices/

NOTE: Using **w1-gpio** on the Raspberry Pi typically needs a 4.7k Ω pull-up resistor connected between the GPIO pin and a 3.3V supply.

Az-Delivery

Python script for reading multiple KY-001 modules

The code is split into two scripts, because of better readability. The following is a code for the class subscript:

```
import os
import glob
import time
class DS18B20:

    def __init__(self):
        os.system('modprobe w1-gpio')
        os.system('modprobe w1-therm')
        base_dir = '/sys/bus/w1/devices/'
        device_folder = glob.glob(base_dir + '28*')
        self._count_devices = len(device_folder)
        self._devices = list()
        i = 0
        while i < self._count_devices:
            self._devices.append(device_folder[i] +
'/w1_slave')
            i += 1

    def device_names(self):
        names = list()
        for i in range(self._count_devices):
            names.append(self._devices[i])
            temp = names[i][20:35]
            names[i] = temp

        return names
```

Az-Delivery

```
# (one tab)
def _read_temp(self, index):
    f = open(self._devices[index], 'r')
    lines = f.readlines()
    f.close()
    return lines

def tempC(self, index = 0):
    lines = self._read_temp(index)
    retries = 5
    while (lines[0].strip()[-3:] != 'YES') and (retries > 0):
        time.sleep(0.1)
        lines = self._read_temp(index)
        retries -= 1

    if retries == 0:
        return 998

    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp = lines[1][equals_pos + 2:]
        return float(temp) / 1000
    else:
        return 999 # error

def device_count(self):
    return self._count_devices
```

(The most of code in the script is modified from the script on the adafruit site)
Save the script under the name *DS18B20classfile.py*.

Az-Delivery

The following is a code for the main script:

```
import time
from DS18B20classFile import DS18B20

degree_sign = u'\xb0' # degree sign
devices = DS18B20()
count = devices.device_count()
names = devices.device_names()

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        i = 0
        print('\nReading temperature, number of sensors: {}'.format(count))

        while i < count:
            container = devices.tempC(i)
            print('{} Temp: {:.3f}{}C, {:.3f}{}F of the device {}'.format(i+1, container, degree_sign, container * 9.0 / 5.0 + 32.0, degree_sign, names[i]))

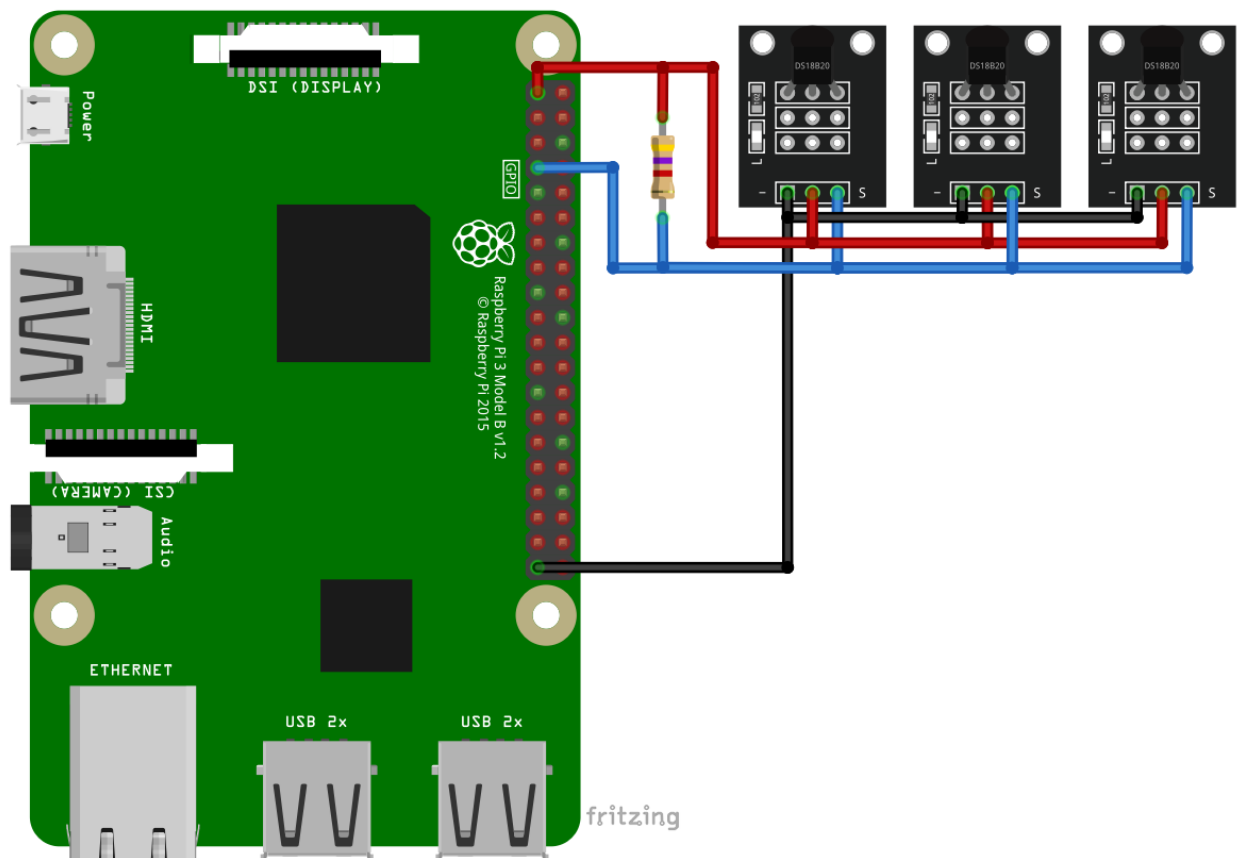
            i = i + 1

        time.sleep(1)

except KeyboardInterrupt:
    print('\nScript end!')
```

Save the script by the name `DS18B20multiple.py` in the same directory where the first script is saved.

For example, three DS18B20 sensors are connected on the same One-Wire interface of the Raspberry Pi as shown on the following connection diagram:

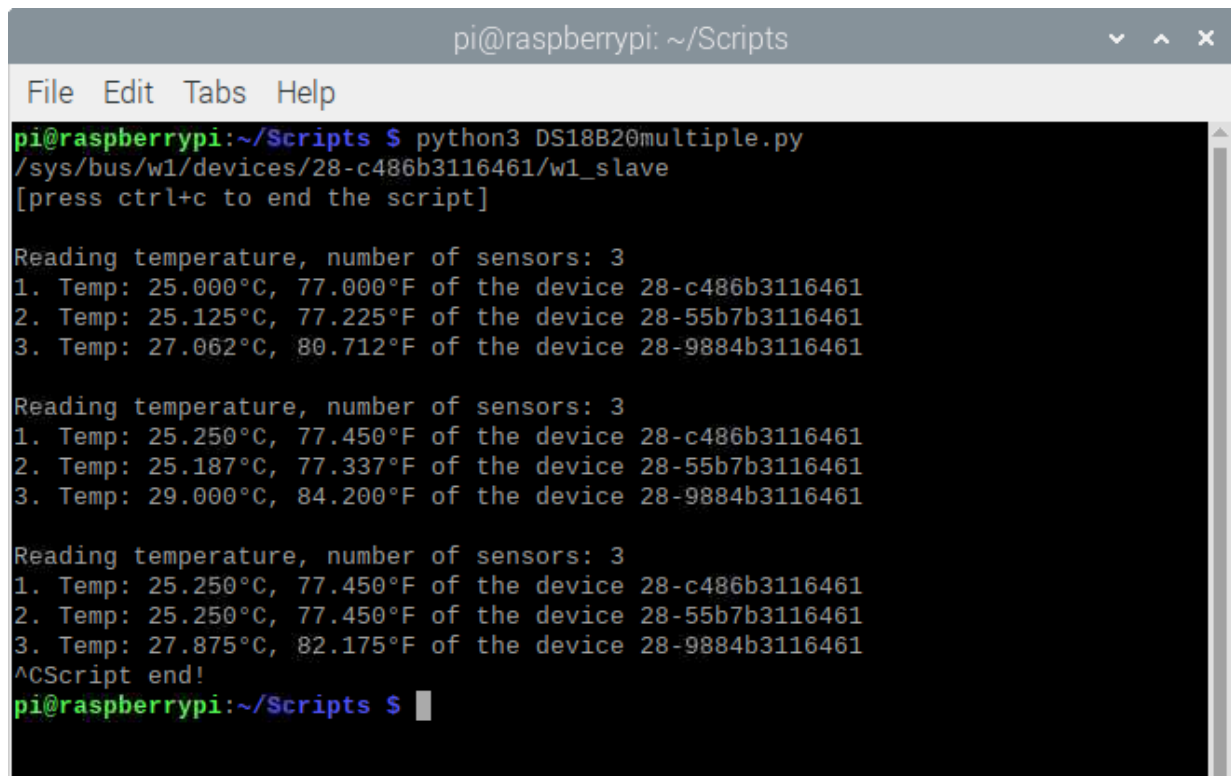


Az-Delivery

To run the main script, open the terminal in the directory where both scripts are saved, and run the following command:

python3 DS18B20multiple.py

The result should look like the output on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 DS18B20multiple.py
/sys/bus/w1/devices/28-c486b3116461/w1_slave
[press ctrl+c to end the script]

Reading temperature, number of sensors: 3
1. Temp: 25.000°C, 77.000°F of the device 28-c486b3116461
2. Temp: 25.125°C, 77.225°F of the device 28-55b7b3116461
3. Temp: 27.062°C, 80.712°F of the device 28-9884b3116461

Reading temperature, number of sensors: 3
1. Temp: 25.250°C, 77.450°F of the device 28-c486b3116461
2. Temp: 25.187°C, 77.337°F of the device 28-55b7b3116461
3. Temp: 29.000°C, 84.200°F of the device 28-9884b3116461

Reading temperature, number of sensors: 3
1. Temp: 25.250°C, 77.450°F of the device 28-c486b3116461
2. Temp: 25.250°C, 77.450°F of the device 28-55b7b3116461
3. Temp: 27.875°C, 82.175°F of the device 28-9884b3116461
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL* + *C* on the keyboard.

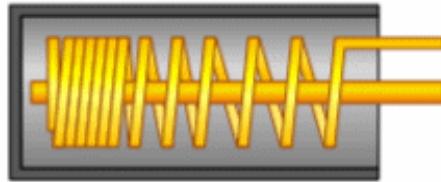
These scripts can be used for one or multiple KY-001 modules.

The class subscript will not be explained, and the main script is self explanatory.

KY-002 Shock Sensor Module

Introduction

The KY-002 shock sensor module is based on the *Gaoxin SW-18010P* shock switch. So, the shock sensor in the KY-002 module allows you to use microcontroller to detect impacts, shocks or shaking. When the switch detects a shock, the output of the module is set to *LOW*.



The shock switch primarily consists of a terminal that forms a center post and a second terminal, that is a spring that surrounds the center post. When a sufficient force is transferred to the switch, the spring moves and connects with the central terminal. The connection between the terminals is momentary and requires a little thinking as you implement it in your project.

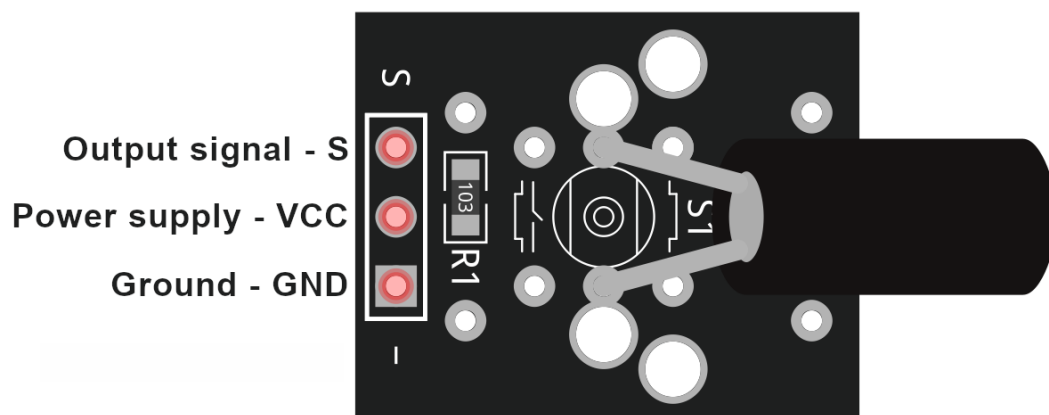
The positioning of the module is also important. Generally speaking, the module should be physically located close to the area that is being monitored. Otherwise, the shock being detected may be dampened by other structural components in your project. An exception to this rule may be where you find that the shock switch on the module is too sensitive for your application. In this case, moving the module further away from the area of interest can make it less sensitive.

Specifications

Max voltage across terminals:	12V DC
Open switch resistance:	> 10M Ω
Closed switch resistance:	< 30 Ω
Dimensions:	19x15x10mm [0.73x0.6x0.4in]

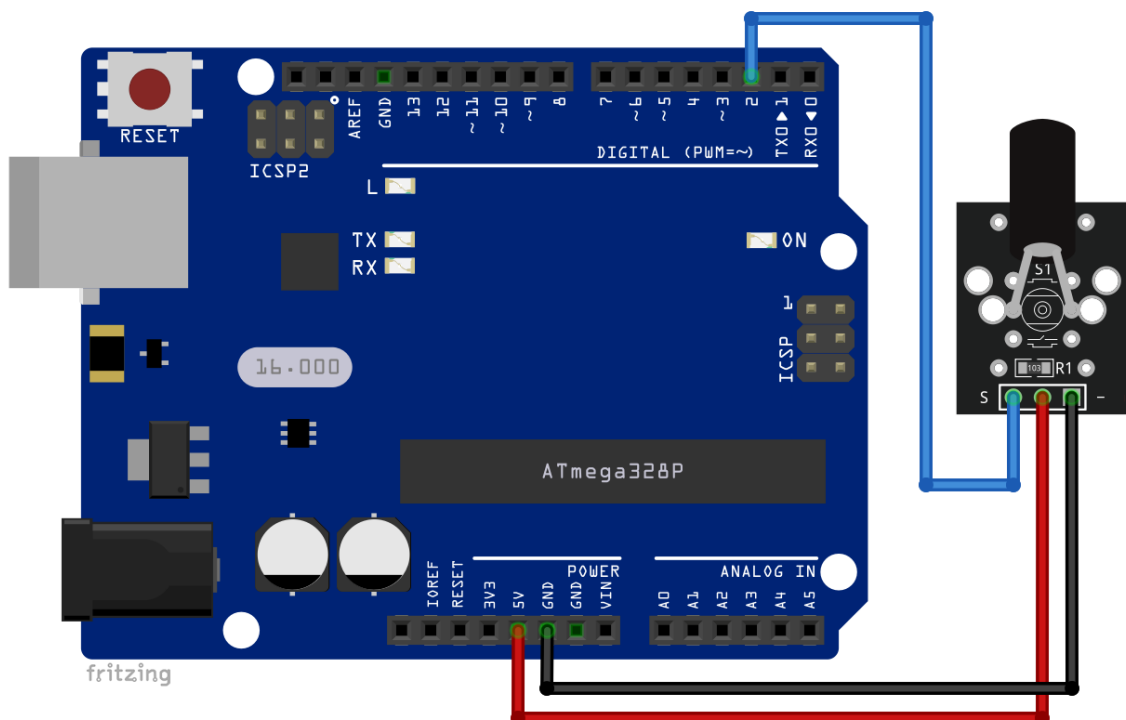
The pinout

The KT-002 shock sensor module has three pins. The pinout diagram is shown on the following image:



Connecting the sensor with Uno

Connect the KY-002 module with the Uno as shown on the following connection diagram:



KY-002 pin	Uno pin	Wire color
S	D2	Blue wire
Middle pin (VCC)	5V	Red wire
- (GND)	GND	Black wire

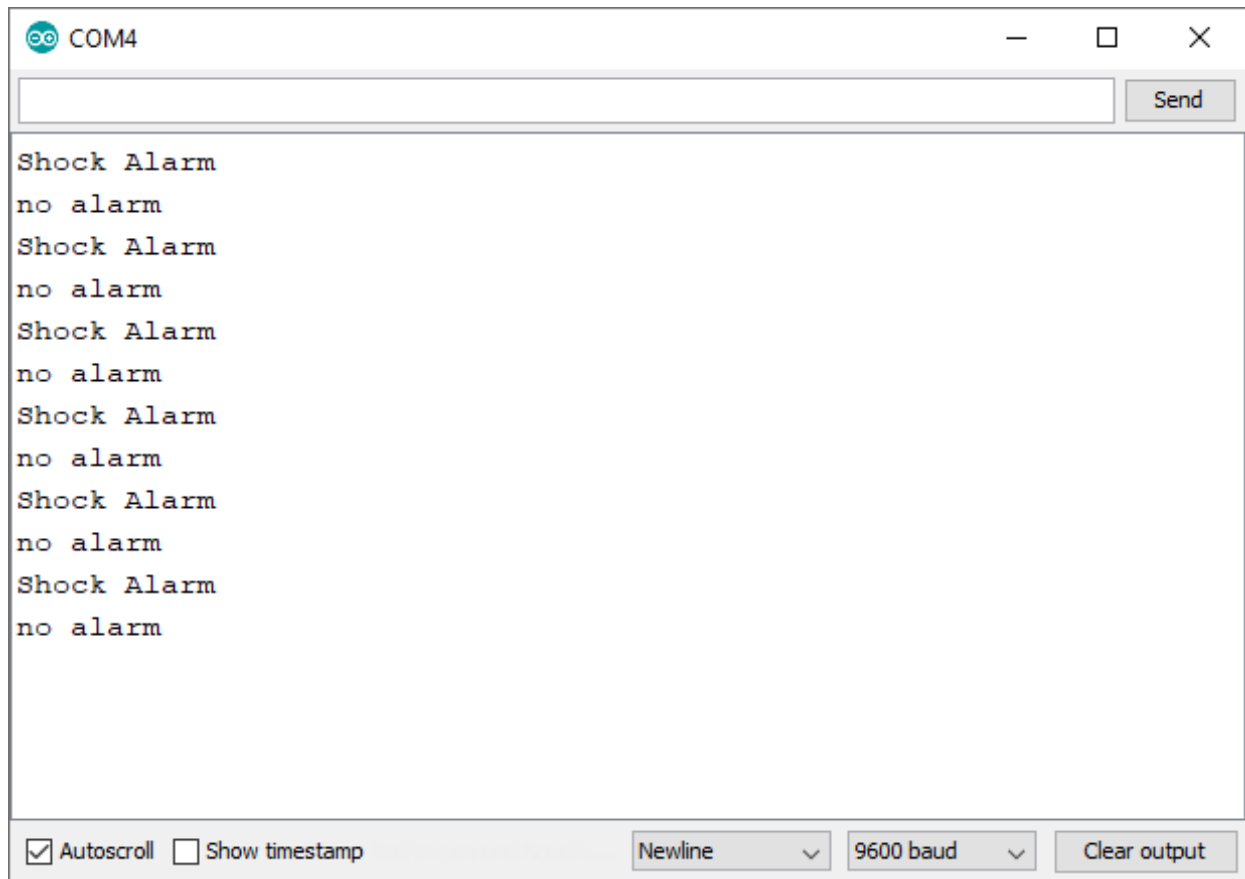
Az-Delivery

Sketch example

```
#define SHOCK_PIN 2
uint8_t shock_value = 1;
boolean alarm = false;
unsigned long last_shock_time;
int shock_alarm_time = 250;
void setup() {
    Serial.begin(9600);
    pinMode(SHOCK_PIN, INPUT);
    pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
    shock_value = digitalRead(SHOCK_PIN);
    if(shock_value == LOW) {
        last_shock_time = millis();
        if(!alarm) {
            Serial.println("Shock Alarm");
            digitalWrite(LED_BUILTIN, HIGH);
            alarm = true;
        }
    }
    else {
        if((millis()-last_shock_time) > shock_alarm_time && alarm){
            Serial.println("no alarm");
            digitalWrite(LED_BUILTIN, LOW);
            alarm = false;
        }
    }
}
```

Az-Delivery

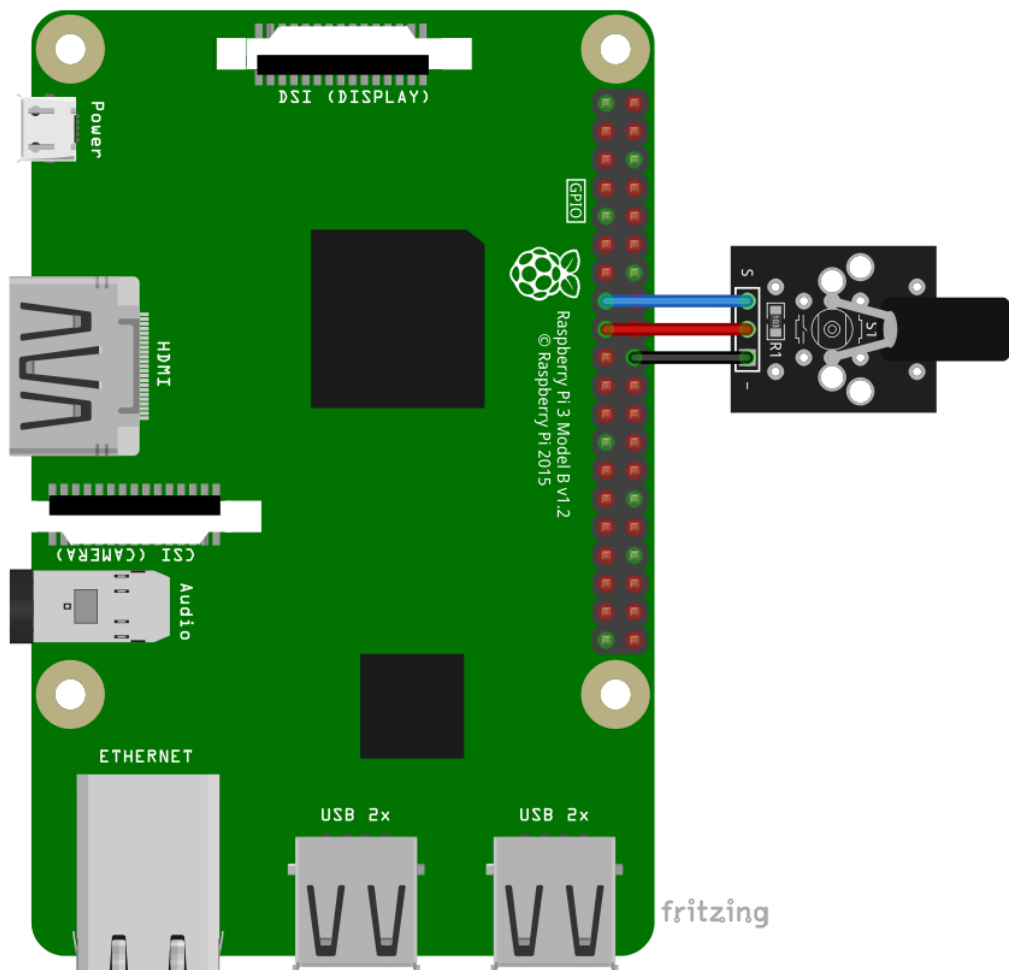
Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



The built-in LED of the Uno should be turned *ON* when there is a *Shock Alarm*.

Connecting the sensor with Raspberry Pi

Connect the KY-002 module with the Raspberry Pi as shown on the following connection diagram:



KY-002 pin	Raspberry Pi pin	Physical pin	Wire color
S	GPIO22	15	Blue wire
Middle pin (VCC)	3V3	17	Red wire
- (GND)	GND	20	Black wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
Shock_PIN = 22
GPIO.setup(Shock_PIN, GPIO.IN)
alarm = False
lastShockTime = 0.0
shockAlarmTime = 0.05
print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        shock_state = GPIO.input(Shock_PIN)
        if shock_state == 1:
            lastShockTime = time.time()
            if not alarm:
                print('Shock Alarm')
                alarm = True
        else:
            if (time.time()-lastShockTime)>shockAlarmTime and
alarm:
                print('No Alarm')
                alarm = False

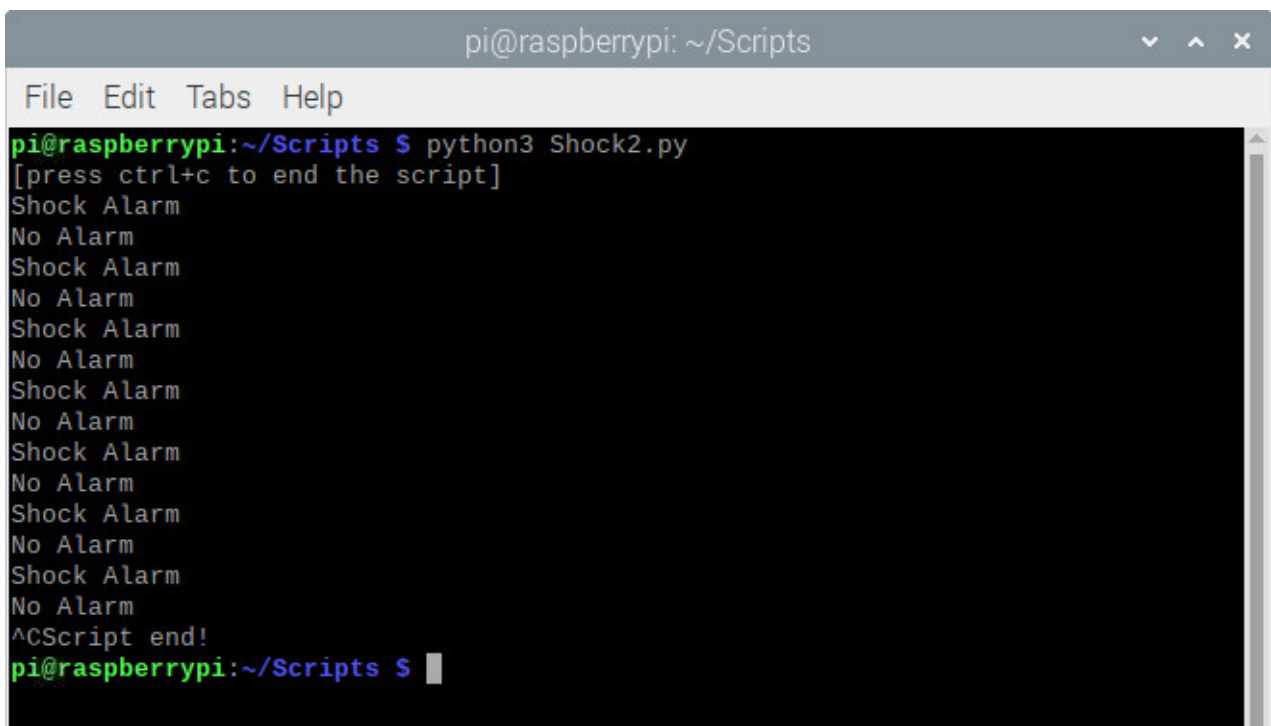
except KeyboardInterrupt:
    print('\nScript end!\n')
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *Shock2.py*. To run the script, open the terminal in the directory where you saved the script and run the following command:

python3 Shock2.py

The result should look like the output on the following image:

A screenshot of a terminal window titled 'pi@raspberrypi: ~/Scripts'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal shows the command 'python3 Shock2.py' being executed. The output consists of alternating lines of 'Shock Alarm' and 'No Alarm', repeated 10 times. After the 10th 'No Alarm', the message '^CScript end!' appears. The prompt 'pi@raspberrypi:~/Scripts \$' is visible at the bottom with a cursor.

To stop the script press *CTRL* + *C* on the keyboard.

KY-003 Hall-effect Sensor Module

Introduction

The KY-003 Hall-effect sensor module has 3144 Hall magnetic sensor, a resistor and a red LED on-board. The module is a very responsive switch that gets activated by a magnetic field. It can be triggered, with high response rates, by a natural magnet or an electromagnet. Most popular uses are in proximity sensing, detecting speeds of spinning objects, or as door sensor. Due to the nature of the sensing, the contactless sensing, hall sensor module is very durable, and there is no wear associated with the use. Sensors come usually as non-latching (sensor module from 35 in 1 sensor kit). However, the latching versions are possible, which means that the sensor keeps the state after the magnet has been removed until the opposite magnetic polarity is introduced.

Specifications

Operating voltage range	from 3.3V to 5V DC
Operating temperature range	from -40°C to 85°C
Dimensions	19x15mm [0.73x0.6in]

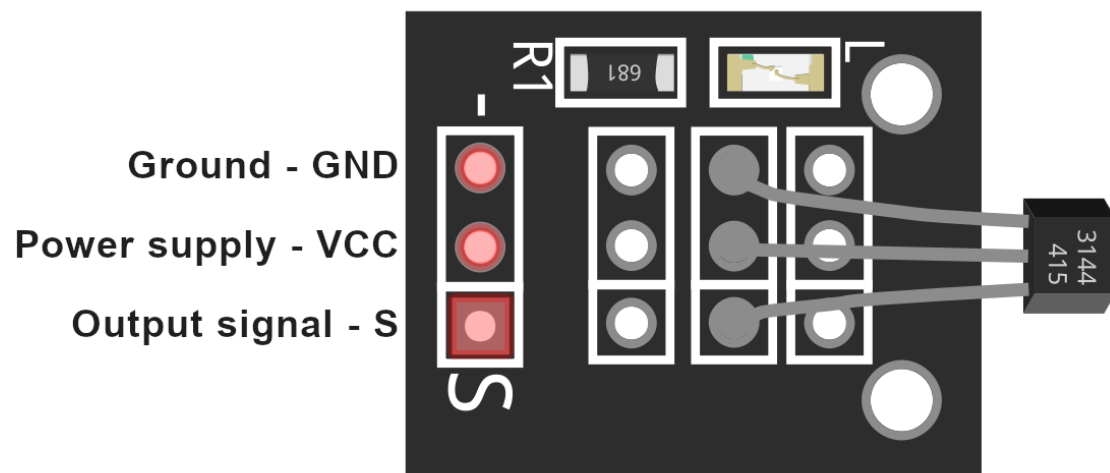
The KY-003 Hall sensor module consists of a sensitive Hall-effect switch, a 680Ω resistor and a red LED. The red LED gets turned *ON* when there is a magnet in the sensor proximity.

Output voltage on the *S* pin of the module is analog in nature, and the value of analog voltage depends on the strength of a magnetic field and the distance of the magnetic field from the sensor.

Also, the *S* pin can be read with the digital input pin. When the digital *HIGH* threshold has been reached, the digital input pin on the microcontroller reads *HIGH* value. To do so, the magnet has to be very close to the Hall sensor module or to the stronger magnetic fields has to be used.

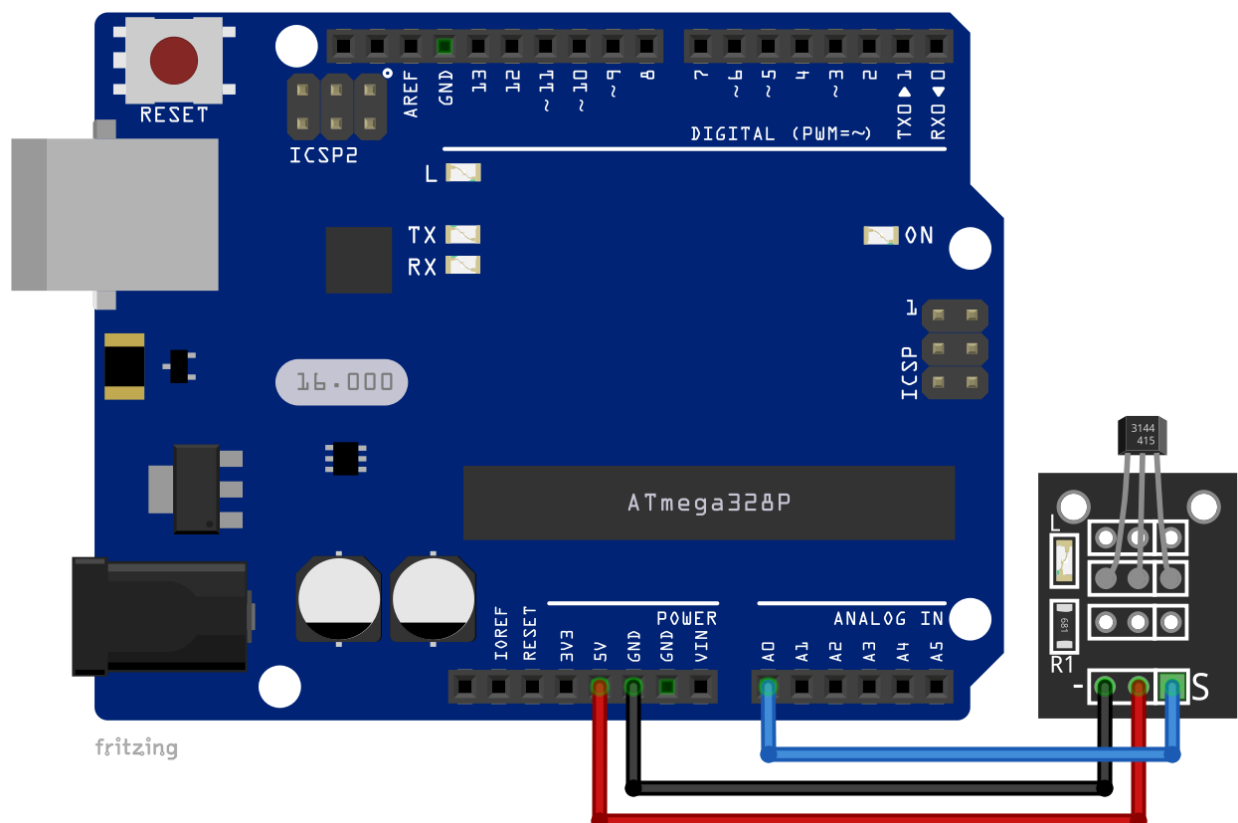
The pinout

The KY-003 Hall-effect sensor module has three pins. The pinout diagram is shown on the following image:



Connecting the sensor with Uno

Connect the KY-003 module with the Uno as shown on the following connection diagram:



KY-003 pin	Uno pin	Wire color
S	A0	Blue wire
Middle pin (VCC)	5V	Red wire
- (GND)	GND	Black wire

Sketch example

```
uint16_t value = 0;

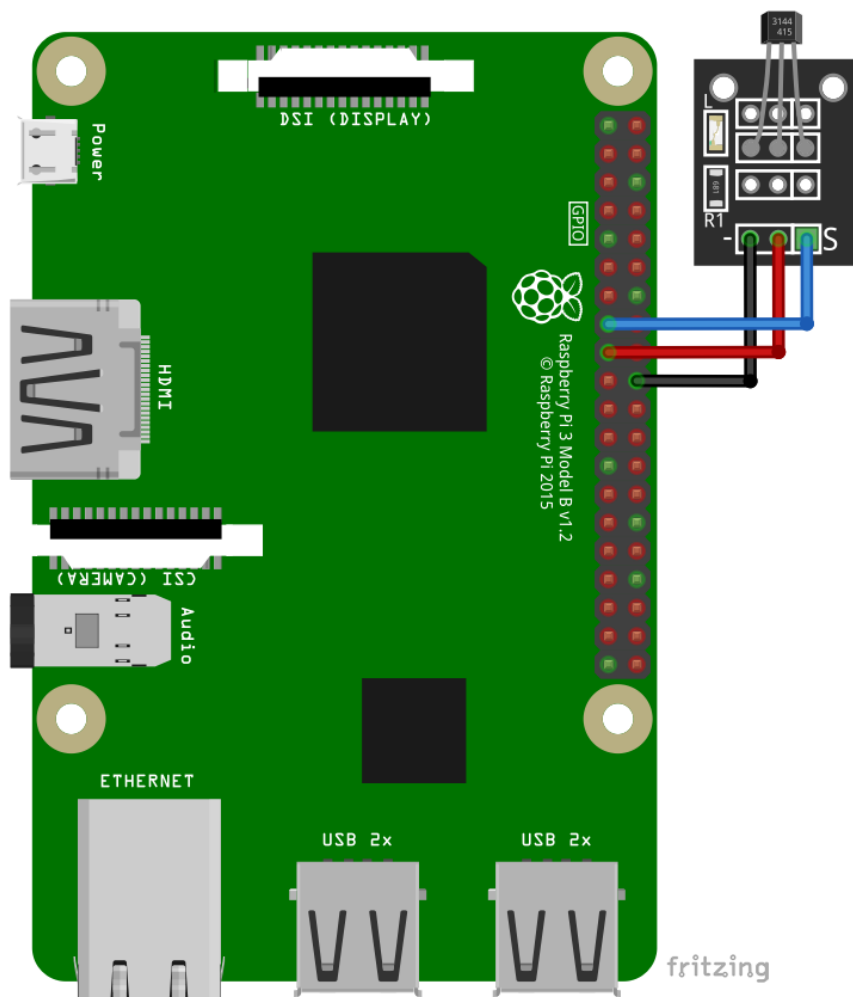
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  value = analogRead(0);
  if(value < 30) {
    digitalWrite(LED_BUILTIN, LOW);
  }
  else {
    digitalWrite(LED_BUILTIN, HIGH);
  }
}
```

When the sketch is uploaded to the Uno, the built-in LED of the Uno should turn *ON* when a magnet is close to the KY-003 module.

Connecting the sensor with Raspberry Pi

Connect the KY-003 module with the Raspberry Pi as shown on the following connection diagram:



KY-003 pin	Raspberry Pi pin	Physical pin	Wire color
S	GPIO22	15	Blue wire
Middle pin (VCC)	3V3	17	Red wire
- (GND)	GND	20	Black wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep

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

Hall_PIN = 22
GPIO.setup(Hall_PIN, GPIO.IN)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        state = GPIO.input(Hall_PIN)
        if not state == 1:
            print('DETECTED MAGNET')

        else:
            print('NO MAGNET')

        sleep(1)

except KeyboardInterrupt:
    print('\nScript end!')

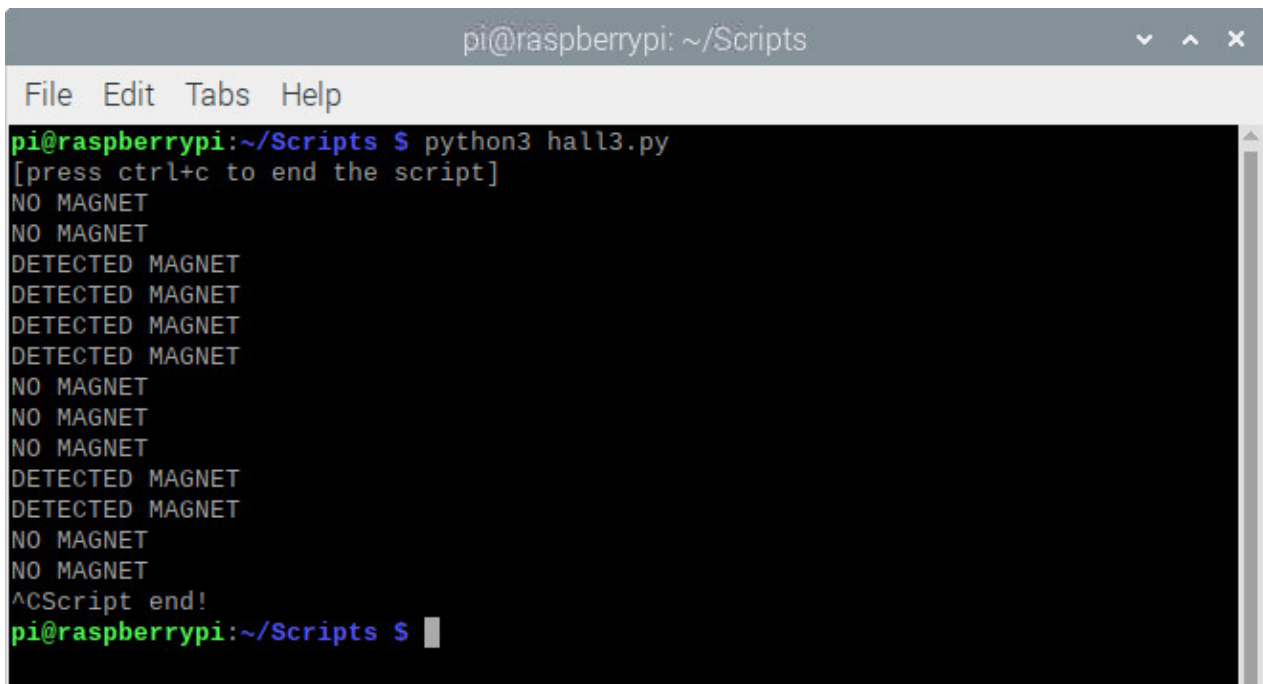
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *hall3.py*. To run the script, open the terminal in the directory where the script is saved and run the following command:

python3 hall3.py

The result should look like the output on the following image:

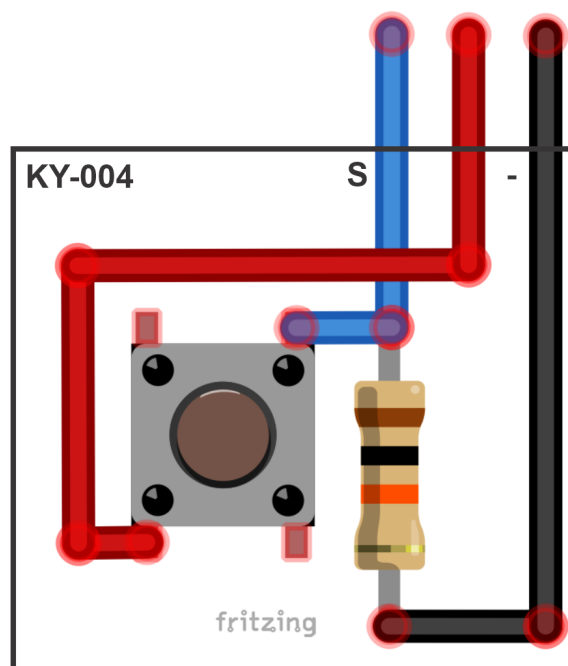
A screenshot of a terminal window titled 'pi@raspberrypi: ~/Scripts'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal shows the command 'python3 hall3.py' being executed. The output consists of a series of lines: '[press ctrl+c to end the script]', 'NO MAGNET', 'NO MAGNET', 'DETECTED MAGNET', 'DETECTED MAGNET', 'DETECTED MAGNET', 'DETECTED MAGNET', 'NO MAGNET', 'NO MAGNET', 'NO MAGNET', 'DETECTED MAGNET', 'DETECTED MAGNET', 'NO MAGNET', 'NO MAGNET', and finally '^CScript end!'. The prompt 'pi@raspberrypi:~/Scripts \$' is visible at the bottom with a cursor.

To stop the script press *CTRL* + *C* on the keyboard.

KY-004 Button Module

Introduction

The KY-004 button module is a simple push-button with a pull-down resistor (*Fritzing* diagram of the module is on the following image). The module is a three-pin module, where *S* is a signal or button state, the middle pin is *VCC* (or power supply, in a range between +3.3V and +5V) and *-* is a ground pin or 0V.



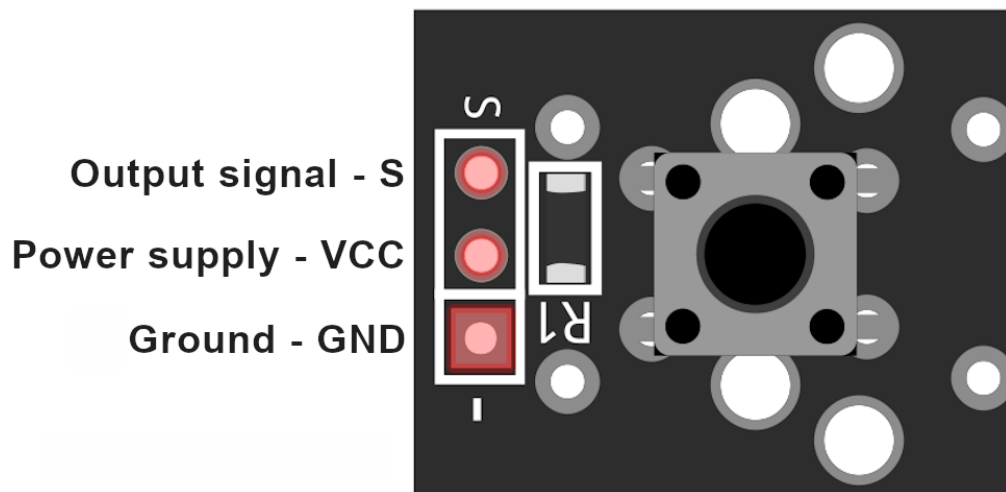
It can be used as a switch, to turn *ON/OFF* LEDs or other electronic devices, or for example, it can be used for counting button presses, etc.

Specifications

Operating voltage range:	from 3.3V to 5V DC
Operating temperature range:	from -40°C to 85°C
Dimensions:	19x15mm [0.73x0.6in]

The pinout

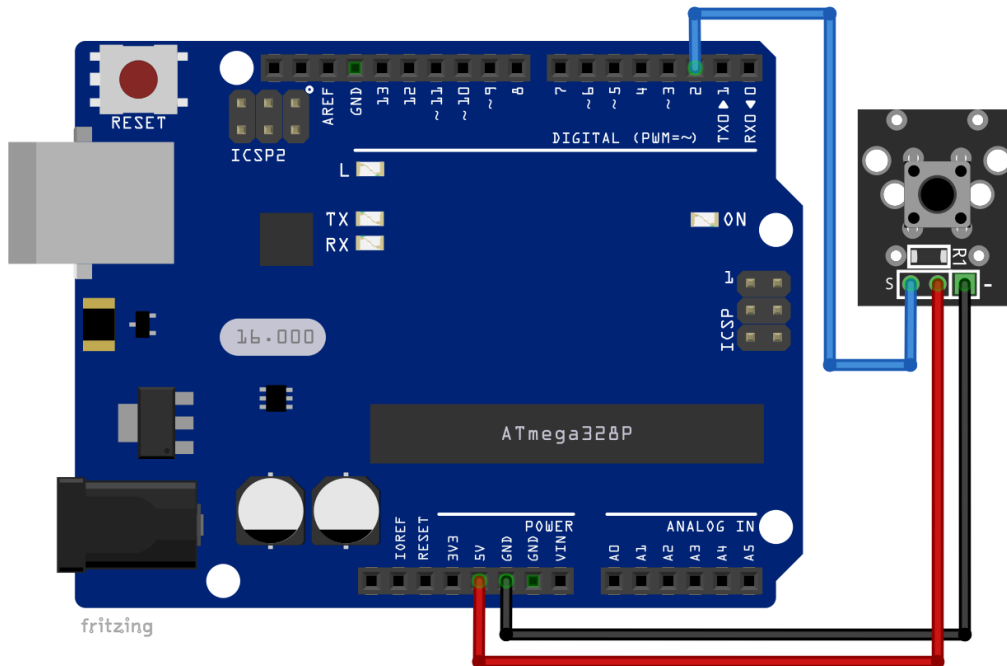
The KY-004 button module has three pins. The pinout diagram is shown on the following image:



When the *VCC* and *GND* pins are connected to the power supply, the *S* pin when button is not pressed is in *HIGH* state, and when it is pressed, the *S* pin is in the *LOW* state.

Connecting the module with Uno

Connect the KY-004 module with the Uno as shown on the following connection diagram:



KY-004 pin	Uno pin	Wire color
S	D2	Blue wire
Middle pin (VCC)	5V	Red wire
- (GND)	GND	Black wire

* **OPTIONAL:** If the push button is used to count the presses and the counting error (bouncing effect) is high, between *GND* and *S* pins connect electrolytic capacitor with values between *10uF* and *47uF*. Watch the polarity of the capacitor! Connect *GND* to "-" or negative side of a capacitor.

Az-Delivery

Sketch example

```
#define BUTTON_PIN 2

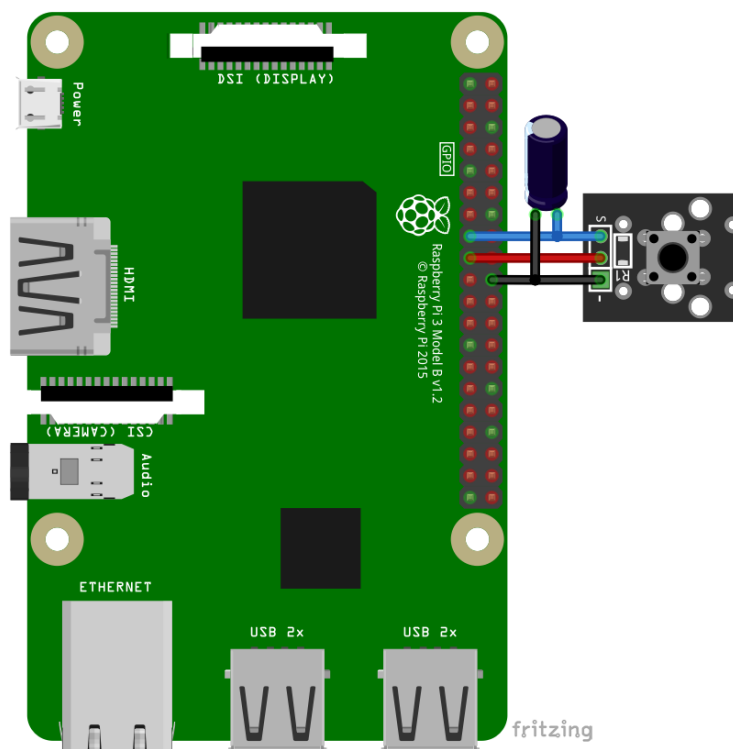
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT);
}

void loop() {
  uint8_t button_state = digitalRead(BUTTON_PIN);
  if(button_state == HIGH) {
    digitalWrite(LED_BUILTIN, HIGH);
  }
  else {
    digitalWrite(LED_BUILTIN, LOW);
  }
}
```

Upload the sketch to the Uno. When the push-button is pressed on the module, the built-in LED on-board Uno turns *ON*. The LED turns *OFF* when the push-button is in non-active state, not pressed.

Connecting the module with Raspberry Pi

Connect the KY-004 module with the Raspberry Pi as shown on the following connection diagram:



KY-004 pin	Raspberry Pi pin	Physical pin	Wire color
S	GPIO22	15	Blue wire
Middle pin (VCC)	3V3	17	Red wire
- (GND)	GND	20	Black wire

* **OPTIONAL:** If counting error (bouncing effect) is high, between *GND* and *S* pins connect electrolytic capacitor with values between *10uF* and *47uF*. Watch the polarity of the capacitor! Connect *GND* to "-" or negative side of a capacitor.

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
Button_PIN = 22
GPIO.setup(Button_PIN, GPIO.IN)
counter = 0
debounce = []
print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        for i in range(10):
            debounce.append(GPIO.input(Button_PIN))

            # detects rising edge of signal
            if debounce[0] == 0 and debounce[9] == 1:
                debounce = []
                counter += 1
                print('{}\n'.format(counter))

            else:
                debounce = []

except KeyboardInterrupt:
    print('\nScript End!\n')

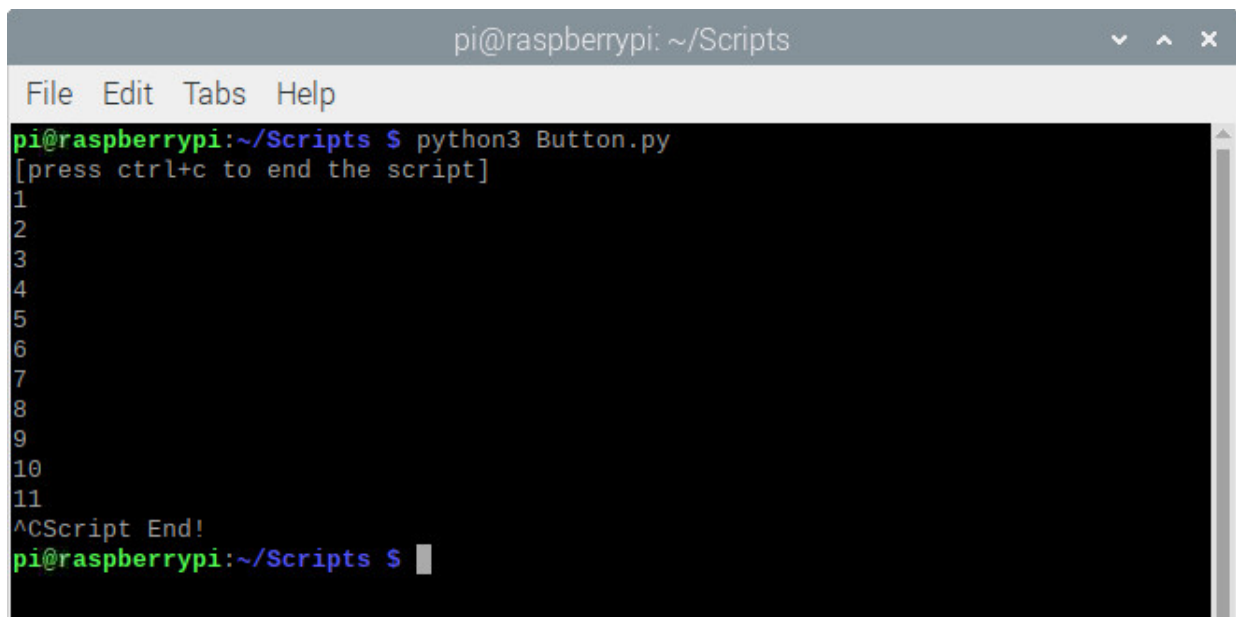
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *Button.py*. To run the script, open the terminal in the directory where the script is saved and run the following command:

python3 Button.py

The result should look like the output on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 Button.py
[press ctrl+c to end the script]
1
2
3
4
5
6
7
8
9
10
11
^CScript End!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL + C* on the keyboard.

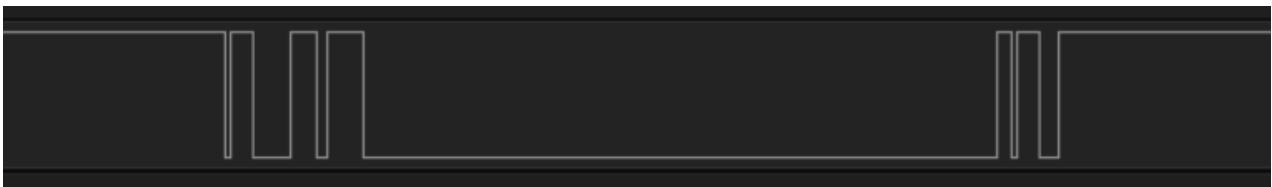
Az-Delivery

The script starts by importing two libraries, *RPi.GPIO* and *time*.

Next, the GPIO pins naming is set to *BCM* and all warnings regarding GPIO interfaces are disabled.

Then, the pin mode for GPIO pin 22 is set to input.

Next, the variable called *counter* and the *tuple* called *debounce* is created. The variable is used to count button presses. The *tuple* is used for debouncing algorithm because it is hard to detect only one signal change on a button pin. On the following image it can be seen what is happening when the button is pressed:



When the button is pressed, the signal changes its state several times before it finally rests in a new state. This is called the *bouncing effect*. It is visible on both signal changes, on *falling* and on *rising* edges of the signal. The falling edge of a digital signal is when the signal changes its state from *HIGH* to *LOW* state. The rising edge of a digital signal is when the signal changes its state from *LOW* to *HIGH* state.

Az-Delivery

After this, the *try-except-finally* block of code is created. In the *try* block an indefinite loop (*while True:*) is created, where the debouncing and counting algorithm is processing (will be explained later in the text).

The *except* block of code is executed when *CTRL + C* is pressed on the keyboard. This is called the keyboard interrupt. When the *except* block of code is executed, message *Script end!* is displayed in the terminal.

The *finally* block of code executes when the script ends, and when this happens, a function called *cleanup()* is executed. This function is used to disable any used GPIO pin modes or interfaces.

In the indefinite loop, first ten signal states on the button pin are read. These values are stored in the *debounce* tuple which is done with the following lines of code:

```
for i in range(10):  
    debounce.append(GPIO.input(Button_PIN))
```

Next, the values of the first and the last elements in the *tuple* are checked. Only when the first is equal to zero and the last is equal to one, the *counter* variable is increased by one, the *tuple* elements are cleared and the message stating counter value is displayed in the terminal. If that is not the case, the *debounce tuple* is cleared and next ten signal states are read again.

Az-Delivery

Another Python script example

The result of the script in this chapter is the same as the script in the previous. The only difference is that the interrupts are used in this script. Any of these two scripts can be used because the result is the same. The benefit of this script is that the delays are not used. Instead an interrupt is used, which happens when the button is pressed. The following is the code:

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
Button_PIN = 22
GPIO.setup(Button_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
counter = 0
def db(channel):
    global counter
    counter = counter + 1
    print('{}'.format(counter))

GPIO.add_event_detect(22, GPIO.FALLING, callback=db, bouncetime=100)
print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        sleep(0.0001)
except KeyboardInterrupt:
    print('\nScript End!\n')
finally:
    GPIO.cleanup()
```

Az-Delivery

This script does the same as the script in the previous chapter, but the only difference is that a new debouncing algorithm is used.

At the beginning of the script, the same libraries are imported, as in the script in the previous chapter. Then the GPIO pin names are set to *BCM* and all warnings regarding GPIO interfaces are disabled.

Next, the pin mode of the GPIO pin 22 is set to input and a variable called *counter* is created. This variable serves the same purpose as in the script in the previous chapter.

Then, the function called *db()* is created. The function has one argument and returns no value. The argument is not used in this script and represents the GPIO pin on which interrupt happens. The function is executed only on the falling edge of the signal on the button pin. In the function, the *counter* variable is set as *global*. This way, the values in this function are stored in the *counter* variable that is previously created outside the function. After this the *counter* variable is increased by one and its value is displayed in the terminal.

Az-Delivery

After this function, the interrupt routine is created with the following line of code:

```
GPIO.add_event_detect(22,GPIO.FALLING,callback=db,bouncetime=100)
```

Where the number 22 represents the button pin; *GPIO.FALLING* represents on which edge of the digital signal the *db()* function is executed; *callback=db* sets which function is executed on the falling edge of the signal; *bouncetime=100* represents how long is the pause between the falling edge and the beginning of the function execution, in this case it is 100 milliseconds. With this pause the bouncing part of the signal is skipped and only the state of signal after the bouncing is read.

Next, the *try-except-finally* block of code is created, like in the previous script. The only difference is the indefinite loop block. In the indefinite loop there is one pause of 100 microseconds so that the loop itself does not block the Raspberry Pi.

The message with the counter state is displayed only when the button is pressed.



KY-005 Infrared LED Module

Introduction

The KY-005 Infrared LED module emits infrared light at the frequency of *38kHz*. The module consists just of a *5mm* infrared LED. The module, together with the KY-022 Infrared receiver module is great for creating infrared interfaces.

Infrared light-emitting diode (IR LED) is a special purpose LED that emits infrared rays ranging from *700nm* to *1um* wavelength. Different infrared LEDs may produce infrared light of differing wavelengths, just like different LEDs produce different colors light. Infrared LEDs are usually made of gallium arsenide or aluminum gallium arsenide. In complement with infrared receivers, these are commonly used as sensors.

Specifications

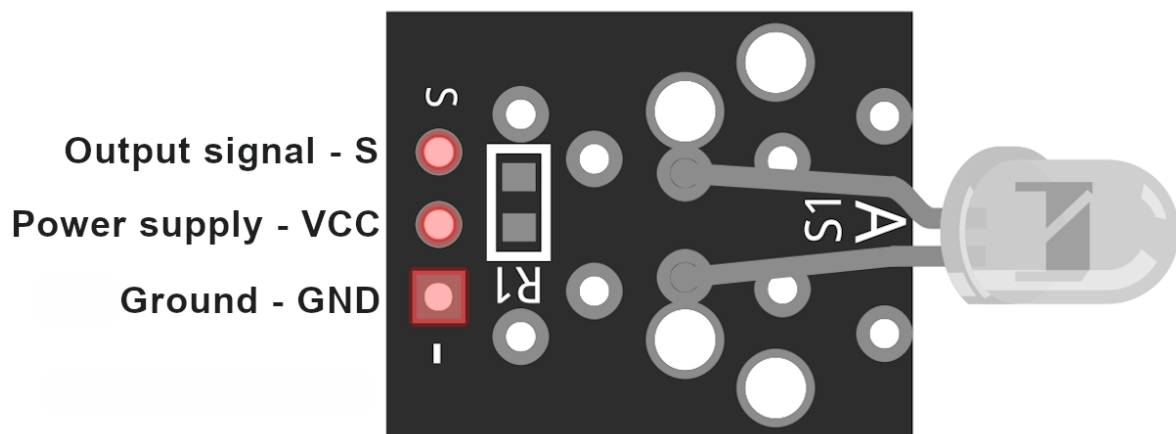
Operating voltage range	from 3.3V to 5V DC
Forward current	30 ~ 60 mA
Power consumption	90mW
Operating temperature	-25°C to 80°C [-13°F to 176°F]
Dimensions	19x5mm [0.73x0.6in]

The appearance of the infrared LED is the same as a common LED. Infrared LEDs are mostly used in the TV remote control. Since the human eye cannot see the infrared radiation. A person can not identify if an infrared LED is working or not. But a camera on mobile phone solves this problem. The rays from the infrared LED are shown on the following image:



The pinout

The KY-005 infrared LED module has three pins. The pinout diagram is shown on the following image:



Working principle of TV remote control

Infrared light is a light that is emitted by the sun, light bulbs, or anything else that produces heat. That means there is a lot of infrared light noise all around us. To prevent this noise from interfering with infrared signals, a signal modulation technique is developed.

In the infrared signal modulation, an encoder on the infrared remote converts a binary signal into a modulated electrical signal. This electrical signal is sent to the infrared LED. The LED converts the modulated electrical signal into a modulated infrared light signal. The infrared receiver or photodiode then demodulates the infrared light signal and converts it back to electrical (binary), before passing the information to a microcontroller. The modulated infrared signal is a series of infrared light pulses, switched *ON/OFF* at a high frequency, known as the carrier frequency. The carrier frequency used by most transmitters is *38kHz* because it is rare in nature and it can be distinguished from ambient noise. This way the infrared receiver knows that the *38kHz* signal was sent from the transmitter and not picked up from the surrounding environment.

The receiver diode detects all frequencies of the infrared light, but it has a band pass filter and only lets through infrared light at *38kHz* frequency. It then amplifies the modulated signal with a pre-amplifier and converts it to a binary signal, before sending it to a microcontroller. The pattern in which the modulated infrared signal is converted to binary is defined by a transmission protocol.

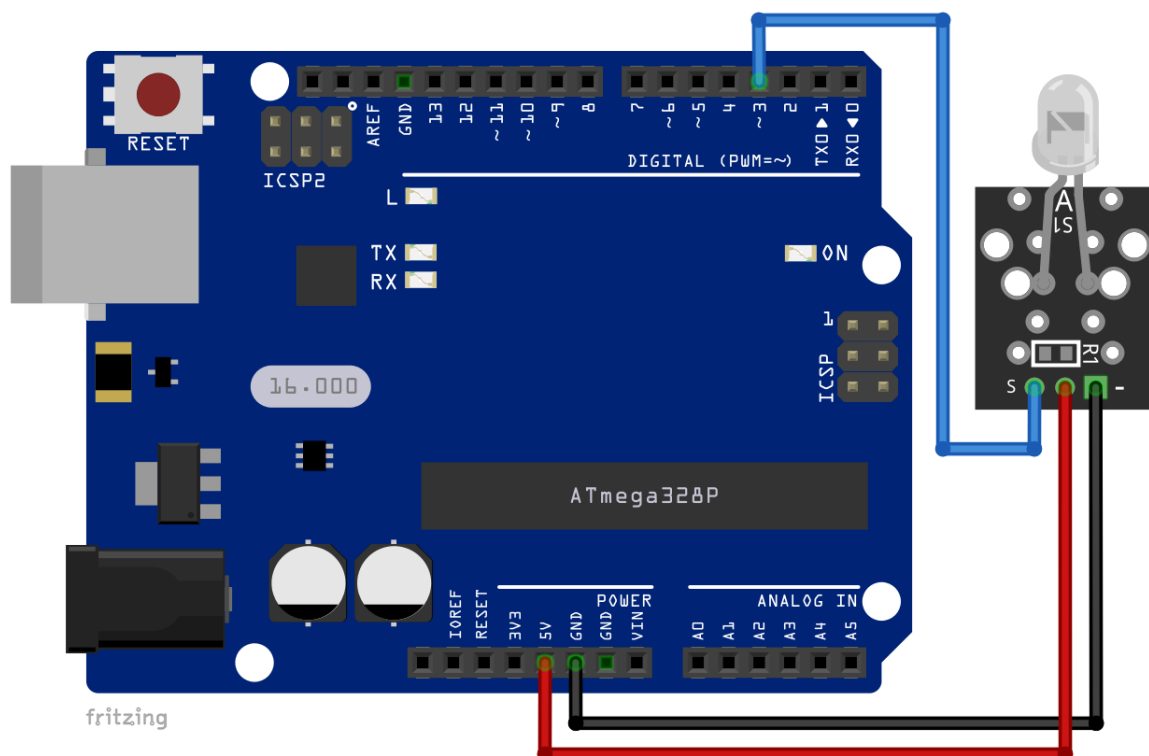
Az-Delivery

There are many infrared transmission protocols: *Sony*, *Matsushita*, *NEC*, and *RC5* are some of the more common protocols. The *NEC* protocol is also the most common type in embedded projects, so it is used in this eBook as an example to show how the transceiver converts the modulated infrared signal to a binary one. Logical *1* starts with a $562.5\mu\text{s}$ long *HIGH* pulse at 38kHz infrared light followed by a $1687.5\mu\text{s}$ long *LOW* pulse. Logical *0* is transmitted with a $562.5\mu\text{s}$ long *HIGH* pulse followed by a $562.5\mu\text{s}$ long *LOW* pulse. Other protocols differ only in the duration of the individual *HIGH* and *LOW* pulses.

Each time a button on the remote control is pressed, a unique hexadecimal code is generated. This is the information that is modulated and sent over infrared light to the receiver. To decipher which key is pressed, the receiving microcontroller needs to know which code corresponds to each key on the remote. Different remotes send different codes for the keypresses, so the code generated for each key on the particular remote has to be determined. It can be found in the datasheet (the infrared key codes should be listed in datasheet). There is a simple sketch that reads most of the popular remote controls and display the hexadecimal codes to the Serial Monitor when a key is pressed (later in the text).

Connecting the module with Uno

Connect the KY-005 module with the Uno as shown on the following connection diagram:



KY-005 pin	Uno pin	Wire color
S	D3	Blue wire
Middle pin (VCC)	5V	Red wire
- (GND)	GND	Black wire

Basic sketch example

```
#define LED 3 # digital I/O pin with PWM feature "~"
uint8_t brightness = 200;

void setup() {
  pinMode(LED, OUTPUT);
  analogWrite(LED, brightness);
}

void loop() { }
```

The sketch is used to turn *ON* the infrared LED on-board the KY-005 module.

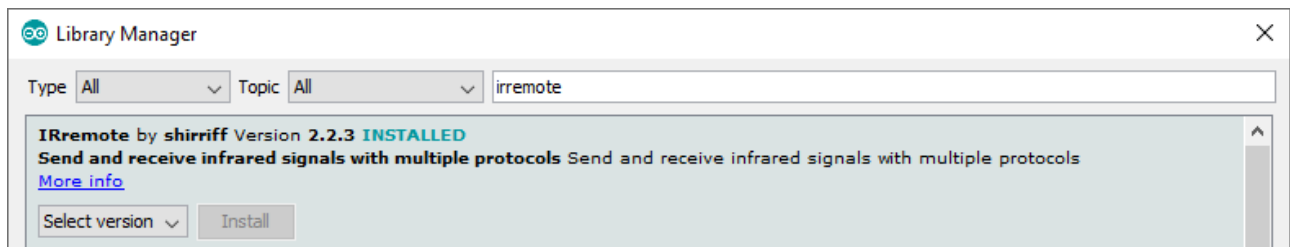
Az-Delivery

Library for Arduino IDE

To use the KY-005 module with Uno as a TV remote it is recommended to install an external library. Open Arduino IDE and go to:

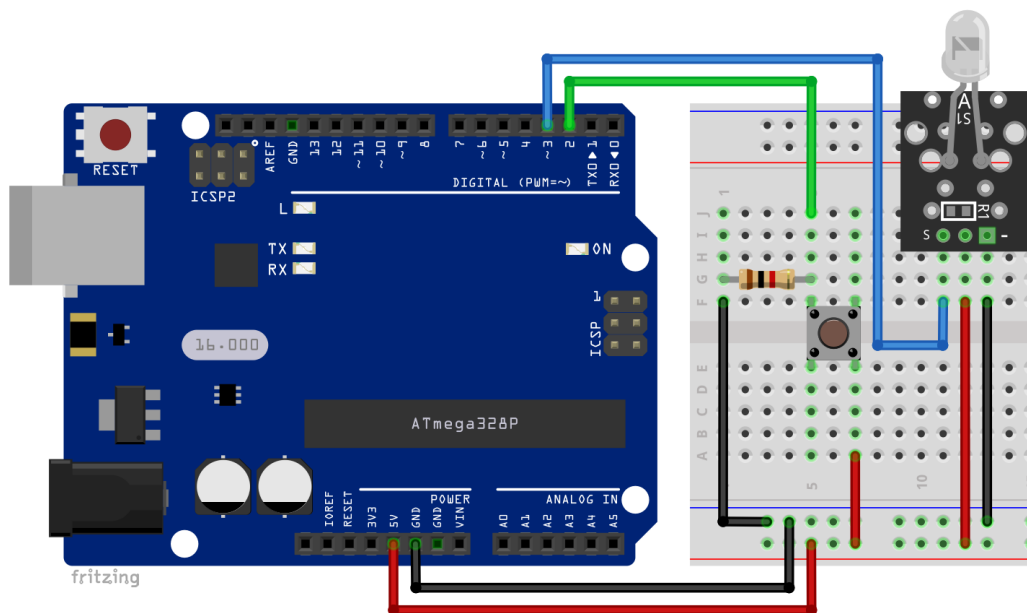
Tools > Manage Libraries

When a new window opens, type *IRremote* (two R's) in the search box and install a library called *IRremote* made by *shirriff*, as shown on the following image:



TV remote with the KY-005 (Uno)

In this chapter the TV remote is simulated using the KY-005 infrared LED module. Connect the KY-005 module with the Uno as shown on the following connection diagram:



KY-005 pin	Uno pin	Wire color
S	D3	Blue wire
Middle pin (VCC)	5V	Red wire
- (GND)	GND	Black wire

Button pin	Uno pin	Wire color
Bottom right pin	5V	Red wire
Top left pin	D2	Green wire
Top left pin (via resistor*)	GND	Black wire

* Resistor resistance value is $1k\Omega$

Az-Delivery

Sketch example

```
#include <IRremote.h>
#define BUTTON_PIN 2
uint8_t button_state = 0, toggle_state = 0;
IRsend transmitter;
void setup() {
    Serial.begin(9600);
    pinMode(LED_BUILTIN, OUTPUT);
    pinMode(BUTTON_PIN, INPUT);
}
void loop() {
    button_state = digitalRead(BUTTON_PIN);
    Serial.println(button_state);
    if (button_state == HIGH) {
        if (toggle_state == 0){
            # ON/OFF button code of some TV remote
            transmitter.sendNEC(0x807F00FF, 32);
            digitalWrite(LED_BUILTIN, HIGH);
            toggle_state = 1;
        }
        else { // if you hold the button
            transmitter.sendNEC(0xFFFFFFFF, 32);
        }
    }
    else {
        digitalWrite(LED_BUILTIN, LOW);
        toggle_state = 0;
    }
    delay(200); }
```

Az-Delivery

At the beginning of the sketch the *IRremote.h* library is included. Next, one macro and two variables are created, where the macro represents the digital I/O pin number to which the push button is connected (2). The variables are used to detect the state of the button. Next, the *transmitter* object is created.

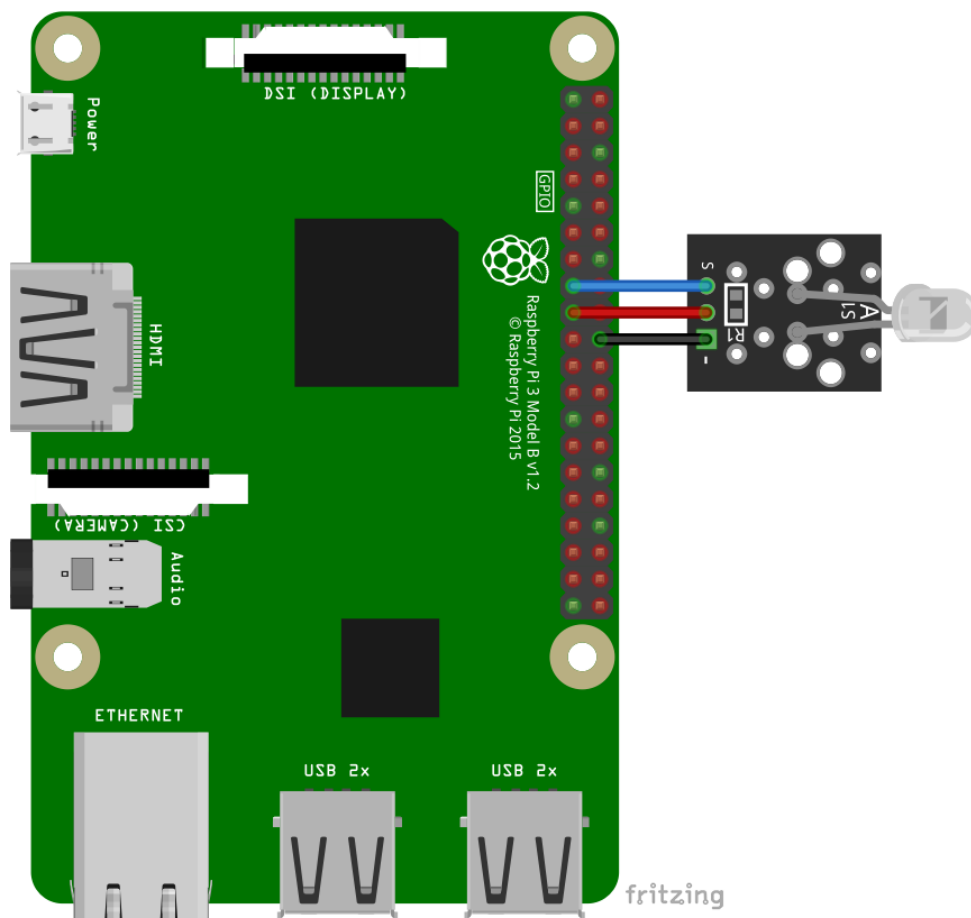
In the *setup()* function, the serial communication is started with the baud rate of *9600bps*. The pin modes for digital pin *13* is set to *OUTPUT* and for the button pin (2) mode is set to *INPUT*.

In the *loop()* function, the state of the push button is read and displayed in the Serial Monitor. Next, the state of the push button is checked if it is in the *HIGH* state. If it is, the state of a *toggle_state* variable is toggled, the *NEC* code is sent via an infrared LED and the on-board LED of the Uno is turned *ON*. If the push button is pressed for too long, the state of the *toggle_state* variable is not toggled anymore, and the *NEC* code *0xFFFFFFFF, F* is sent to simulate TV remote behavior. The remote outputs number *0xFFFFFFFF* when the specific button is pressed and held (an indication that the button is held). If the state of the push button is *LOW*, the on-board LED of the Uno is turned *OFF* and the state of the *toggle_state* variable is set to zero value. At the end of the *loop()* function, the delay of *200* milliseconds is set.

The code that is sent is for turning ON/OFF one specific TV. To determine this code, the KY-022 module is used. Read about this in the eBook for the KY-022 module.

Connecting the module with Raspberry Pi

Connect the KY-005 module with the Raspberry Pi as shown on the following connection diagram:



KY-005 pin	Raspberry Pi pin	Physical pin	Wire color
S	GPIOD22	15	Blue wire
Middle pin (VCC)	3V3	17	Red wire
- (GND)	GND	20	Black wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

IRLED = 22
GPIO.setup(IRLED, GPIO.OUT)
print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        # PWM 500Hz, duty cycle 50%
        GPIO.output(IRLED, GPIO.HIGH)
        sleep(0.001)
        GPIO.output(IRLED, GPIO.LOW)
        sleep(0.001)
except KeyboardInterrupt:
    print('\nScript End!\n')
finally:
    GPIO.cleanup()
```

The script just turns *ON* the infrared LED on-board the KY-005 module (using PWM). Save the script under the name *ky005.py*. To run the script, open the terminal in the directory and run the following command:

python3 ky005.py

To end the script press *CTRL + C* on the keyboard.

KY-006 Passive Buzzer Module

Introduction

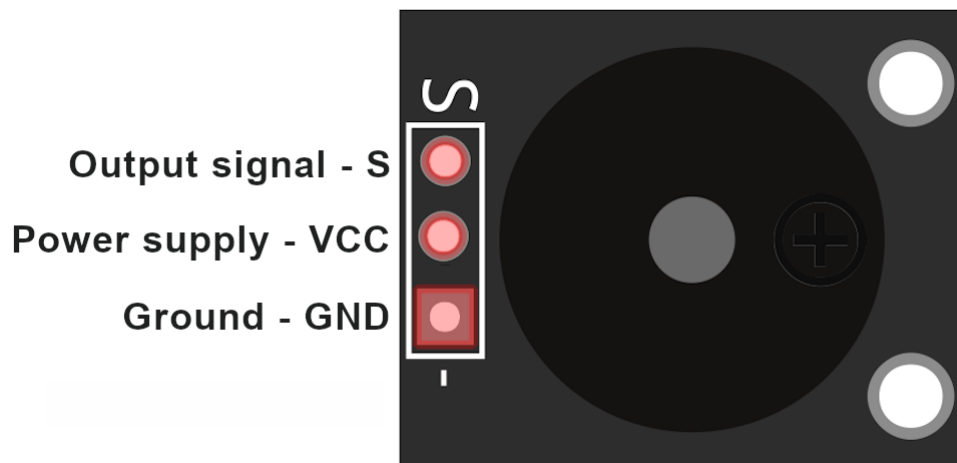
Buzzers come in two varieties, active and passive. The KY-006 module consists of the passive piezoelectric buzzer. The passive buzzer can generate tones in a range between $1.5kHz$ and $2.5kHz$ by switching the module *ON/OFF* at different frequencies, either using delays or PWMs - Pulse Width Modulation of the digital signal.

Specifications

Operating voltage range:	from 1.5 to 5V DC
Tone generation range:	from 1.5kHz to 2.5kHz
Dimensions:	19x15mm [0.73x0.6in]

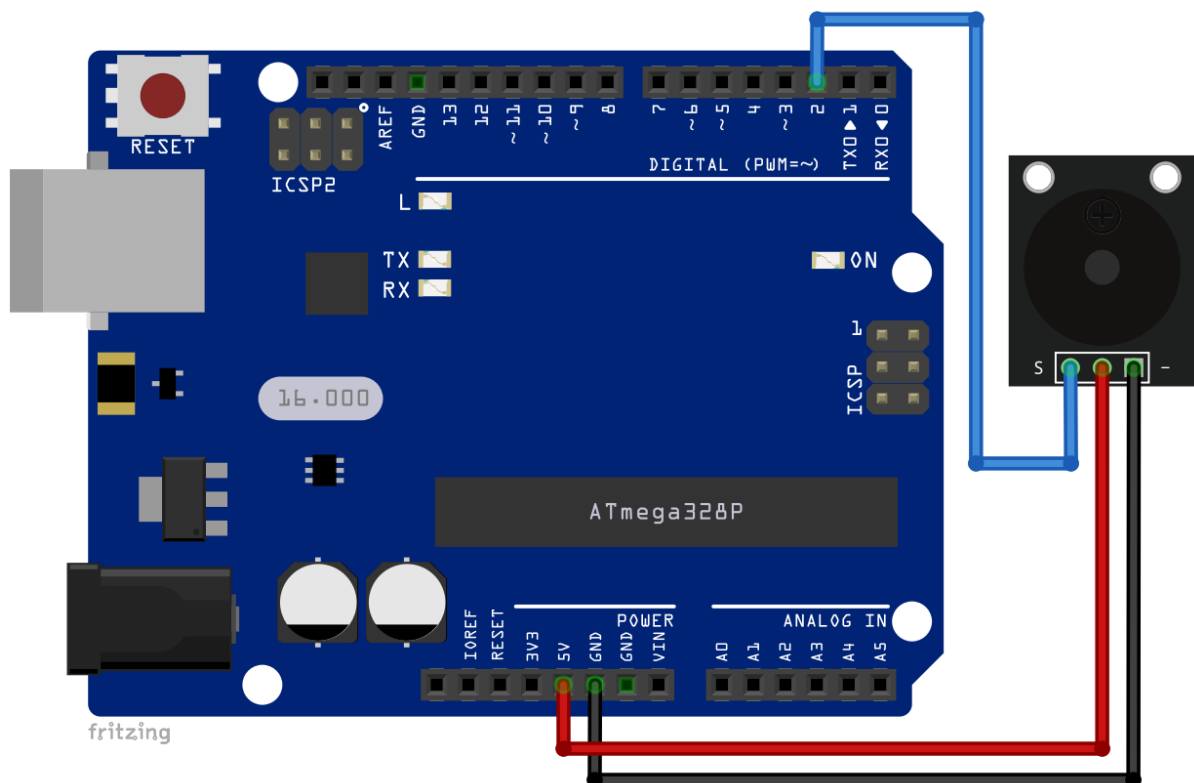
The pinout

The KY-006 passive buzzer module has three pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-006 module with the Uno as shown on the following connection diagram:



KY-006 pin	Uno pin	Wire color
S	D2	Blue wire
Middle pin (VCC)	5V	Red wire
- (GND)	GND	Black wire

Az-Delivery

Sketch example

```
#define BUZZER_PIN 2

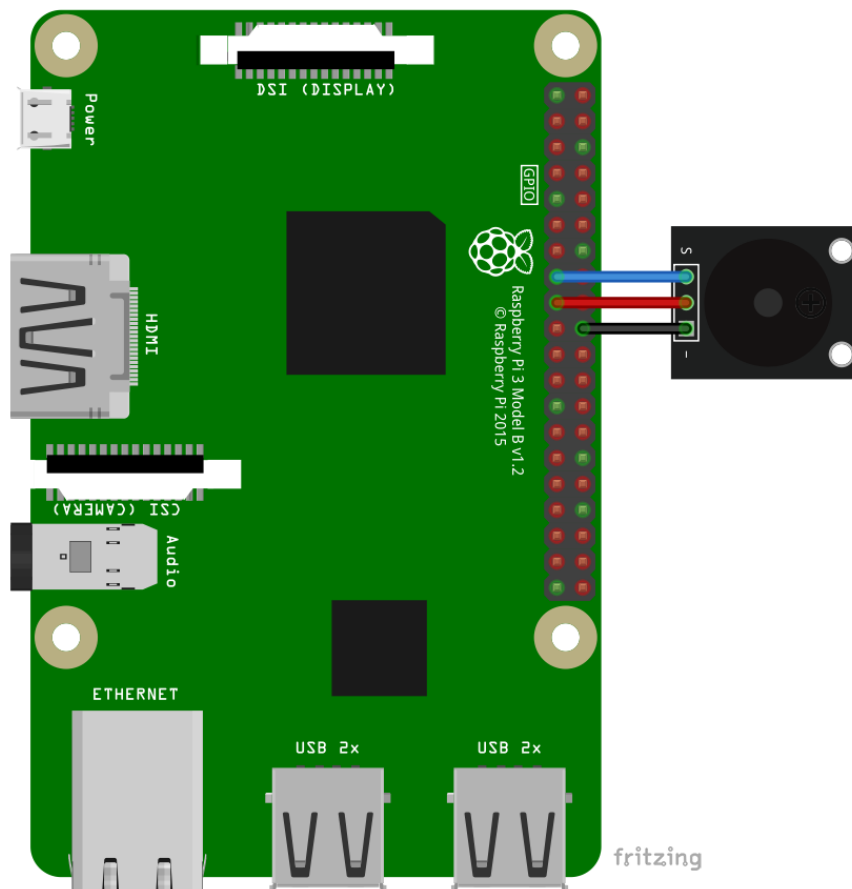
void setup() {
  pinMode(BUZZER_PIN, OUTPUT);
}

void loop() {
  while(1) {
    // Software PWM with duty cycle of 50%, and frequency
    for(uint8_t i = 0; i < 50; i++) { // 500Hz frequency
      digitalWrite(BUZZER_PIN, HIGH);
      delay(1); // Delay 1ms
      digitalWrite(BUZZER_PIN, LOW);
      delay(1); // Delay 1ms
    }

    for(uint8_t i = 0; i < 50; i++) { // 167Hz frequency
      digitalWrite(BUZZER_PIN, HIGH);
      delay(3); // Delay 3ms
      digitalWrite(BUZZER_PIN, LOW);
      delay(3); // Delay 3ms
    }
  }
}
```

Connecting the module with Raspberry Pi

Connect the KY-006 module with the Raspberry Pi as shown on the following connection diagram:



KY-006 pin	Raspberry Pi pin	Physical pin	Wire color
S	GPIO22	15	Blue wire
Middle pin (VCC)	3V3	17	Red wire
GND (-)	GND	20	Black wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep

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

Buzzer_PIN = 22
GPIO.setup(Buzzer_PIN, GPIO.OUT, initial=GPIO.LOW)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        print('Buzzer frequency of 5kHz') # Duty cycle 50%
        for i in range(1000):
            GPIO.output(Buzzer_PIN, GPIO.HIGH)
            sleep(0.0001)
            GPIO.output(Buzzer_PIN, GPIO.LOW)
            sleep(0.0001)

        print('Buzzer frequency of 1.5kHz') # Duty cycle 50%
        for i in range(1000):
            GPIO.output(Buzzer_PIN, GPIO.HIGH)
            sleep(0.00034)
            GPIO.output(Buzzer_PIN, GPIO.LOW)
            sleep(0.00034)

except KeyboardInterrupt:
    print('\nScript end!\n')

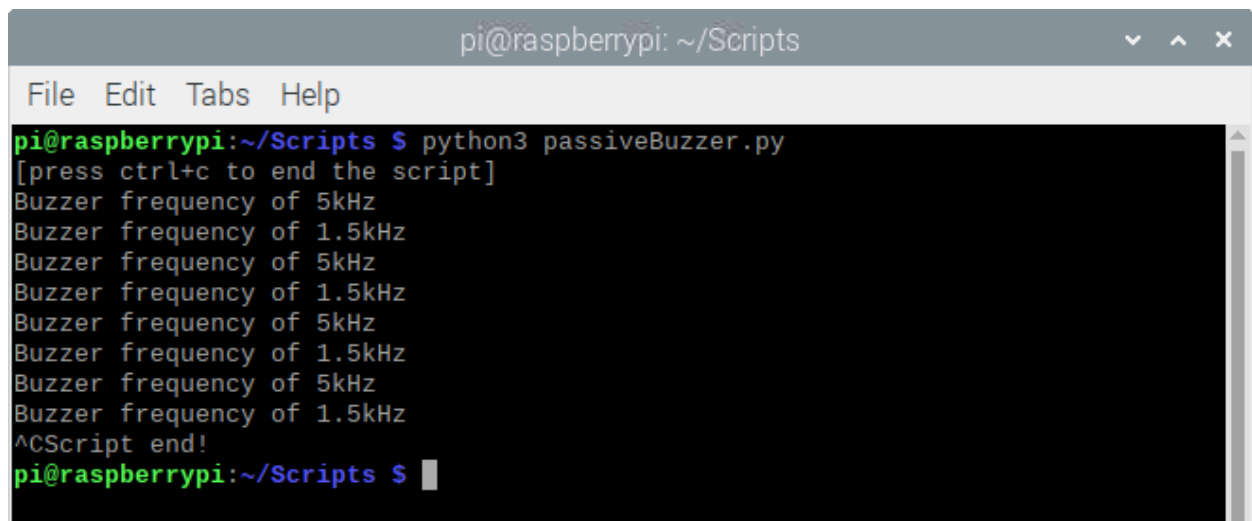
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *PassiveBuzzer.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 PassiveBuzzer.py

The result should look like the output on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 passiveBuzzer.py
[press ctrl+c to end the script]
Buzzer frequency of 5kHz
Buzzer frequency of 1.5kHz
Buzzer frequency of 5kHz
Buzzer frequency of 1.5kHz
Buzzer frequency of 5kHz
Buzzer frequency of 1.5kHz
Buzzer frequency of 5kHz
Buzzer frequency of 1.5kHz
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL + C* on the keyboard.

A melody

If you want to play with the module a little bit, here is the melody of the *Star Wars Imperial March*. To make it, a few things should be installed. First, change the connection for the signal pin of Passive Buzzer from *GPIO22 [pin 15]* to the *GPIO17 [pin 11]*. Second, the *wiringPi* library had to be installed. The library comes preinstalled with *Raspberry Pi OS*, but if it is not, open the terminal in *Raspberry Pi OS* and run the following three commands, one by one:

```
git clone git://git.drogon.net/wiringPi
cd wiringPi
./build
```

Next, the program example for the melody had to be downloaded and compiled. Open the terminal and run the following commands, one by one:

```
git clone https://github.com/leon-anavi/rpi-examples.git
cd rpi-examples/buzzer/c/
gcc starwars.c -o starwars -lwiringPi -std=c99
```

To start the melody, open the terminal in the *rpi-examples/buzzer/c/* directory and run the following command:

```
sudo ./starwars
```



LED Traffic Light Module

Introduction

LED Traffic Light Module is a mini-traffic light, easy to manage and suitable for the production of traffic light system model.

It is a single plate with three 8mm LEDs (Light Emitting Diodes) - red, yellow and green - and built-in resistors. There are only four pins on the device, one for the anode of each color and a common cathode. Thanks to this, the connection to Arduino Uno or Raspberry Pi is much simpler.

Of course, the LEDs are dimmable with pulse width modulation (PWM) when connected to the respective pins (Arduino pins with tilde - '~').

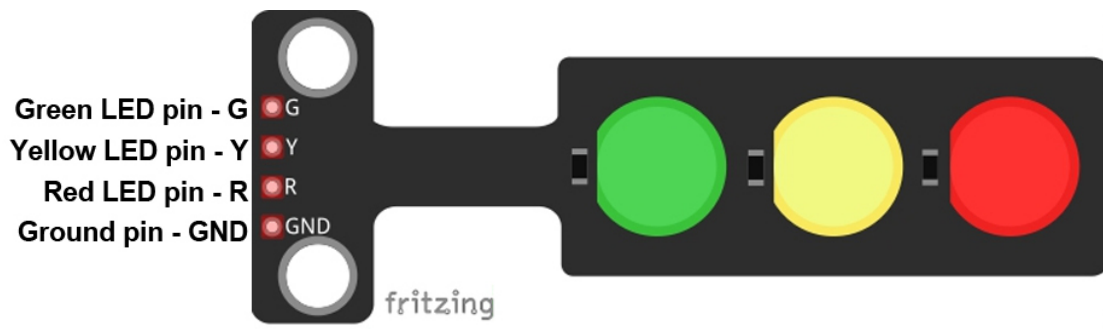
Developed for educational purposes (road safety training as well as basic electronics and physical computing/coding), the traffic light system may also be useful for the moderation of a discussion, or filling or charging indicator, etc.

Specifications

Operating voltage range	from 3.3V to 5V DC
Current	650nm
Operating current	about 4mA at 3.3V, 8-9mA at 5V
Mounting holes	2, with the diameter of 3mm
Dimensions	56mm + 5mm pins, 21mm, 11 mm

The pinout

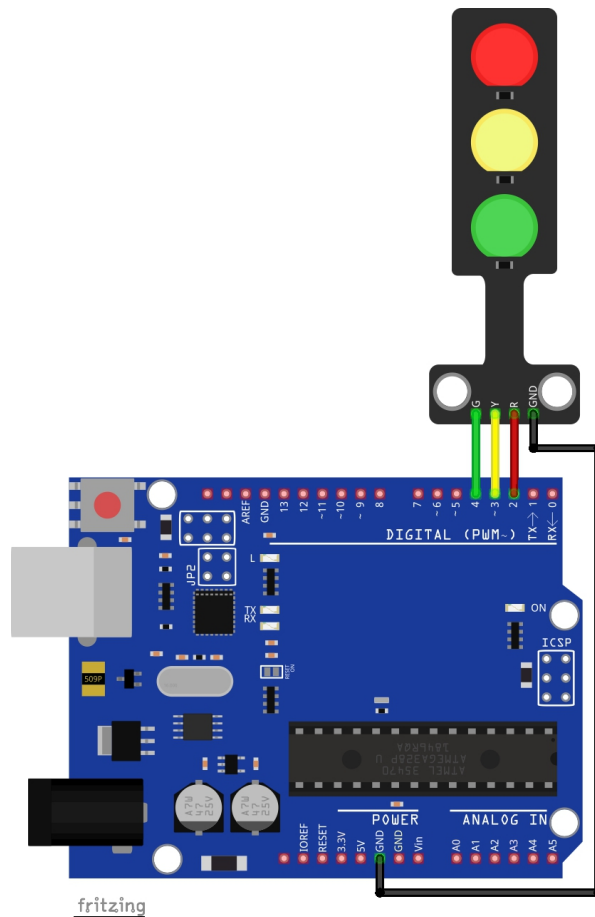
The LED Traffic Light Module has 4 pins. The pinout is shown in the following image:



Three tiny built-in 330-ohm resistors placed under every LED limit the current to 4mA at 3.3V and 8 – 9mA at 5V.

Connecting the module with Uno

Connect the module with the Uno as shown on the following connection diagram:



LED Traffic Light pin	Uno pin	Wire color
GND	GND	Black wire
R	2	Red wire
Y	3	Yellow wire
G	4	Green wire

Az-Delivery

Sketch example

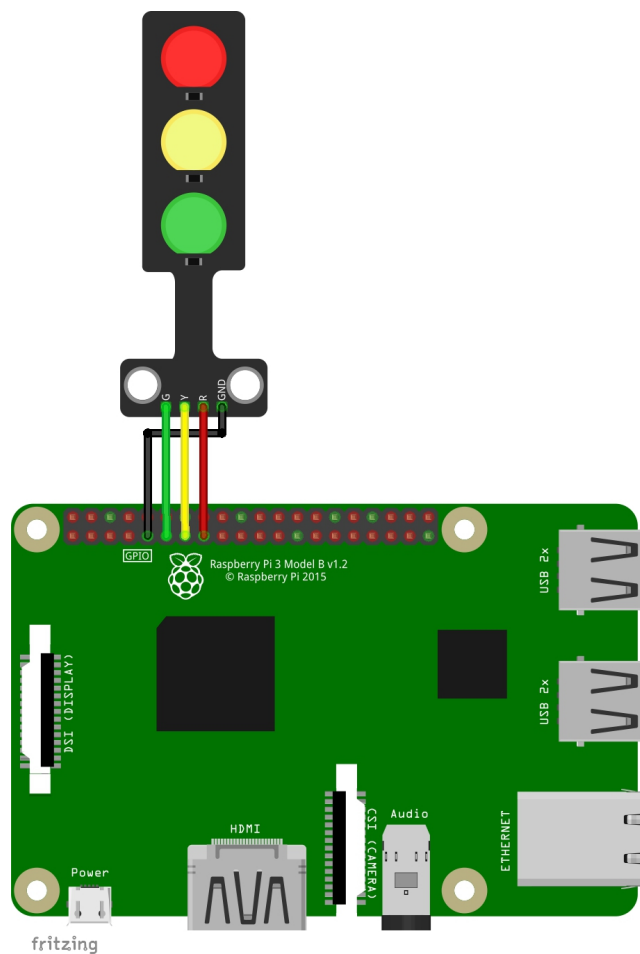
```
int GREEN = 4;
int YELLOW = 3;
int RED = 2;

void setup() {
    pinMode(GREEN, OUTPUT);
    pinMode(YELLOW, OUTPUT);
    pinMode(RED, OUTPUT);
}

void loop() {
    digitalWrite(GREEN, LOW);
    digitalWrite(YELLOW, LOW);
    digitalWrite(RED, HIGH);
    delay(3000);
    digitalWrite(YELLOW, HIGH);
    delay(1000);
    digitalWrite(RED, LOW);
    digitalWrite(YELLOW, LOW);
    digitalWrite(GREEN, HIGH);
    delay(3000);
    digitalWrite(GREEN, LOW);
    digitalWrite(YELLOW, HIGH);
    delay(1000);
}
```

Connecting the module with Raspberry Pi

Connect the module with the Raspberry Pi as shown on the following connection diagram:



LED Traffic Light pin	Raspberry Pi pin	Physical pin	Wire color
GND	GND	9	Black wire
R	GPIO22	15	Red wire
Y	GPIO27	13	Yellow wire
G	GPIO17	11	Green wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
import time
import signal
import sys

GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT) # green LED
GPIO.setup(27, GPIO.OUT) # yellow LED
GPIO.setup(22, GPIO.OUT) # red LED

def allLightsOff(signal, frame):
    GPIO.output(17, False)
    GPIO.output(27, False)
    GPIO.output(22, False)
    GPIO.cleanup()
    sys.exit(0)

signal.signal(signal.SIGINT, allLightsOff)
print('[Press CTRL + C to end the script!]\n')

while True:
    GPIO.output(22, True)    # red on
    time.sleep(3)

    GPIO.output(27, True)    # yellow on, red is still on
    time.sleep(1)
```


Az-Delivery

```
GPIO.output(27, False)    # yellow off
GPIO.output(22, False)    # red off
GPIO.output(17, True)     # green on
time.sleep(5)
```

```
GPIO.output(17, False)    # green off
GPIO.output(27, True)     # yellow on
time.sleep(2)
```

```
GPIO.output(27, False)    # yellow off
```

KY-009 SMD RGB LED Module

Introduction

On-board the KY-009 module there is an *SMD* (Surface Mounted Device) LED. The LED is actually three LEDs in one (Red, Green and Blue LEDs in one RGB LED). Using the 8-bit PWM signal from microcontroller the RGB Led can produce 16.7 million different color variations.

There are NO on-board resistors in series with LED pins. When the KY-009 module is connected with the Arduino or Raspberry Pi, an external resistor for every color pin has to be connected in series with the pin (as shown on the connection diagrams later in the eBook). Use one 220Ω resistor for a **red** color, 100Ω resistor for a **green** and 100Ω resistor for a **blue** color. The red part of the KY-009 LED works on lower voltages.

Voltage limitations per color are:

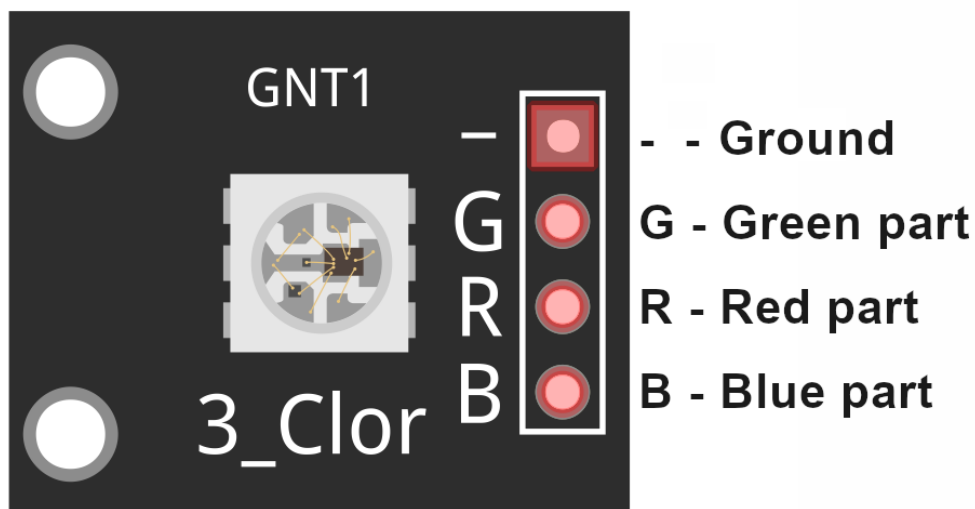
Red	1.8V ~ 2.4V
Green	2.8V ~ 3.6V
Blue	2.8V ~ 3.6V

Specifications

Operating voltage range	from 3.3V to 5V DC
Operating temperature	-25°C to 80°C [-13°F to 176°F]
Dimensions	19x15mm [0.73x0.6in]

The pinout

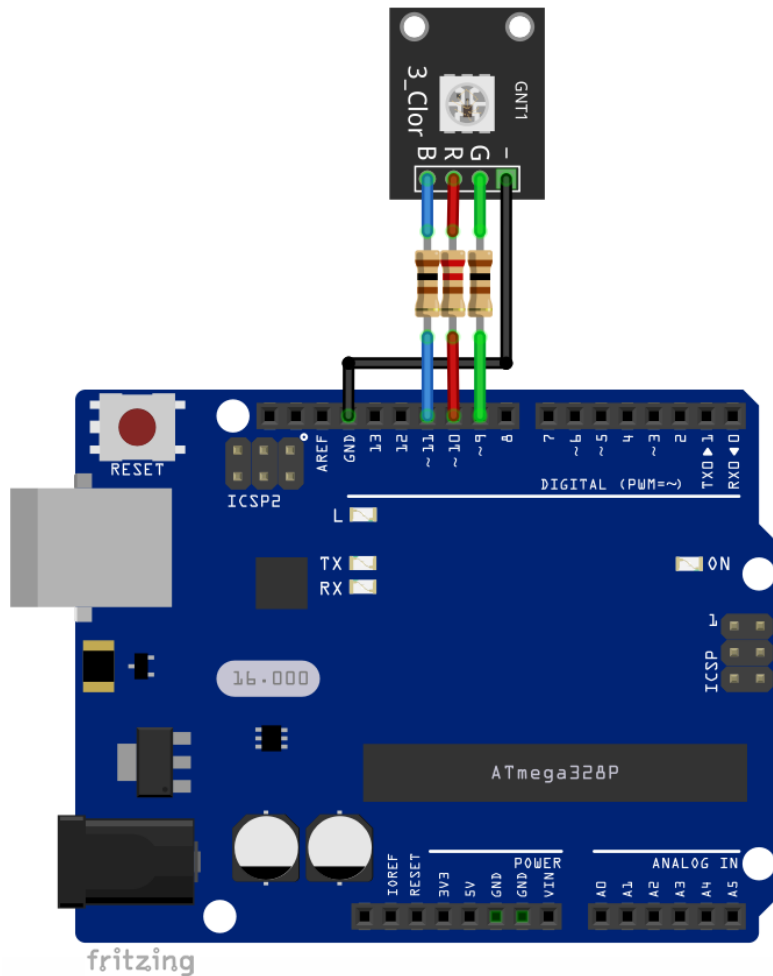
The KY-009 SMD LED module has four pins. The pinout diagram is shown on the following image:



Note: Use resistors for color pins, or you can damage the module.

Connecting the module with Uno

Connect the KY-009 module with the Uno as shown on the following connection diagram:



KY-009 pin	Uno pin	Wire color
B	D11	Blue wire
R	D10	Red wire
G	D9	Green wire
- (GND)	GND	Black wire

Az-Delivery

Sketch example

```
#define RED_PIN    10
#define GREEN_PIN  9
#define BLUE_PIN   11

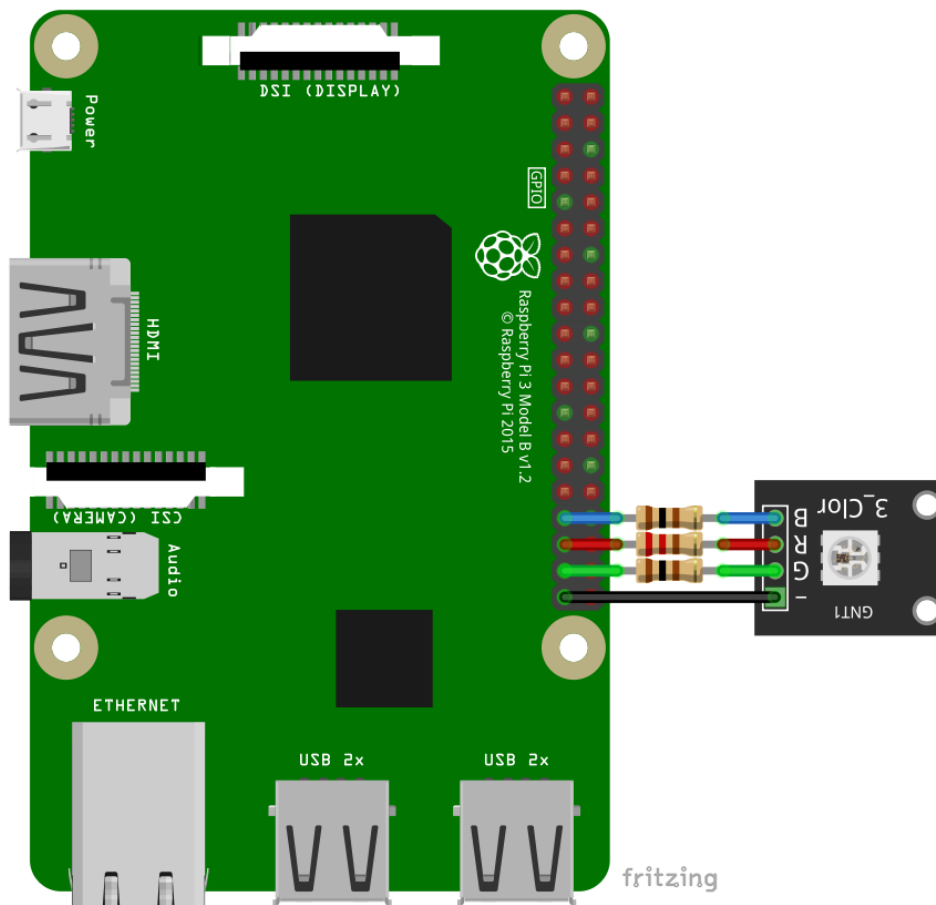
void setup() {
  pinMode(RED_PIN,  OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN,  OUTPUT);
}

void loop() {
  // purple color
  analogWrite(RED_PIN, 250);
  analogWrite(GREEN_PIN, 0);
  analogWrite(BLUE_PIN, 250);
  delay(1000);

  // orange color
  analogWrite(RED_PIN, 255);
  analogWrite(GREEN_PIN, 100);
  analogWrite(BLUE_PIN, 0);
  delay(1000);
}
```

Connecting the module with Raspberry Pi

Connect the KY-009 module with the Raspberry Pi as shown on the following connection diagram:



KY-009 pin	Raspberry Pi pin	Physical pin	Wire color
B	GPIO13	33	Blue wire
R	GPIO19	35	Red wire
G	GPIO26	37	Green wire
- (GND)	GND	39	Black wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
Blue_PIN = 13
Red_PIN = 19
Green_PIN = 26
GPIO.setup(Red_PIN, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Green_PIN, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Blue_PIN, GPIO.OUT, initial=GPIO.LOW)
red = GPIO.PWM(Red_PIN, 100) # PWM, frequency 100Hz
green = GPIO.PWM(Green_PIN, 100) # PWM, frequency 100Hz
blue = GPIO.PWM(Blue_PIN, 100) # PWM, frequency 100Hz
red.start(1)
green.start(1)
blue.start(1)
print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        red.ChangeDutyCycle(90) # Purple color
        green.ChangeDutyCycle(1)
        blue.ChangeDutyCycle(5)
        sleep(1) # Wait for a second
        red.ChangeDutyCycle(99) # Orange color
        green.ChangeDutyCycle(1)
        blue.ChangeDutyCycle(1)
        sleep(1) # Wait for a second

except KeyboardInterrupt:
    print('\nScript end!\n')
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script under the name *SMDRGBLED.py*. To run the script open terminal in the directory where the script is saved and run the following command:

```
python3 SMDRGBLED.py
```

and watch the color changing magic.

When starting the PWM, with the following line of code:

```
red.start(1)
```

use any number different than 0, because if the number 0 is used, that disables PWM on the selected GPIO pin.

Also, when using the following line of code:

```
red.ChangeDutyCycle(1)
```

to change the duty cycle of the PWM signal, do not use number 0, because of the same reason as for the *red.start(1)*.

To end the script press *CTRL + C* on the keyboard.

KY-010 Light Barrier Detector Module

Introduction

The KY-010 light barrier detector module consists of an optical emitter (an LED) and detector (a photo transistor), two resistors ($10k\Omega$ and 330Ω). The sensor uses a beam of light between the emitter and detector to check if the optical path between them is blocked by a non transparent object.

The module is a three-pin module, where S is the signal pin, the middle pin is VCC (between +3.3V and +5V) and – (minus) is ground pin or 0V.

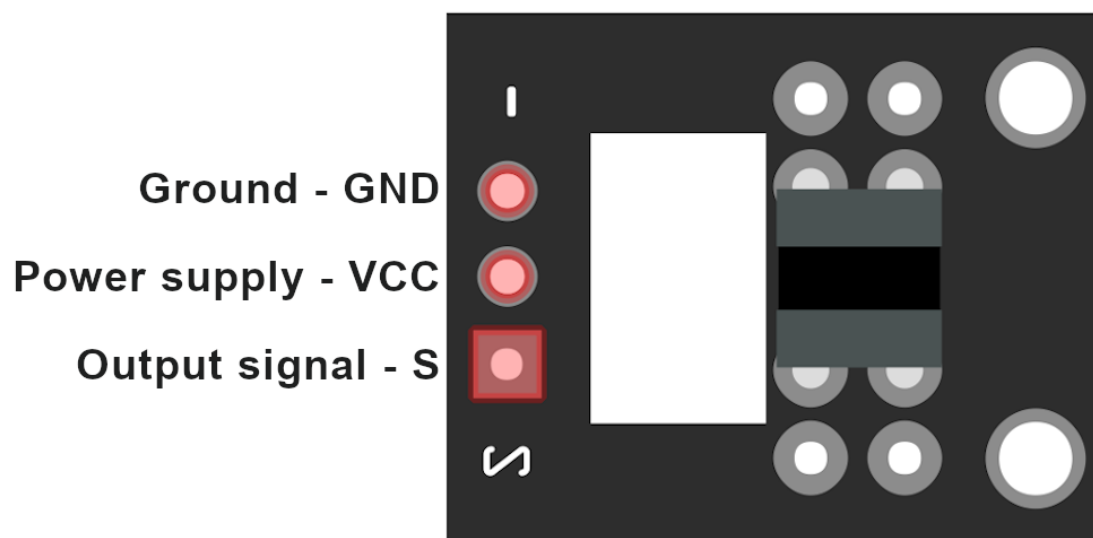
When the KY-010 is connected to the power supply, the LED diode emits light. When there is no light barrier between emitter and detector, the module output is in the *HIGH* state, and when there is a barrier, the output is in the *LOW* state.

Specifications

Operating voltage range	from 3.3V to 5V DC
Operating temperature	-25°C to 80°C [-13°F to 176°F]
Dimensions	19x15mm [0.73x0.6in]

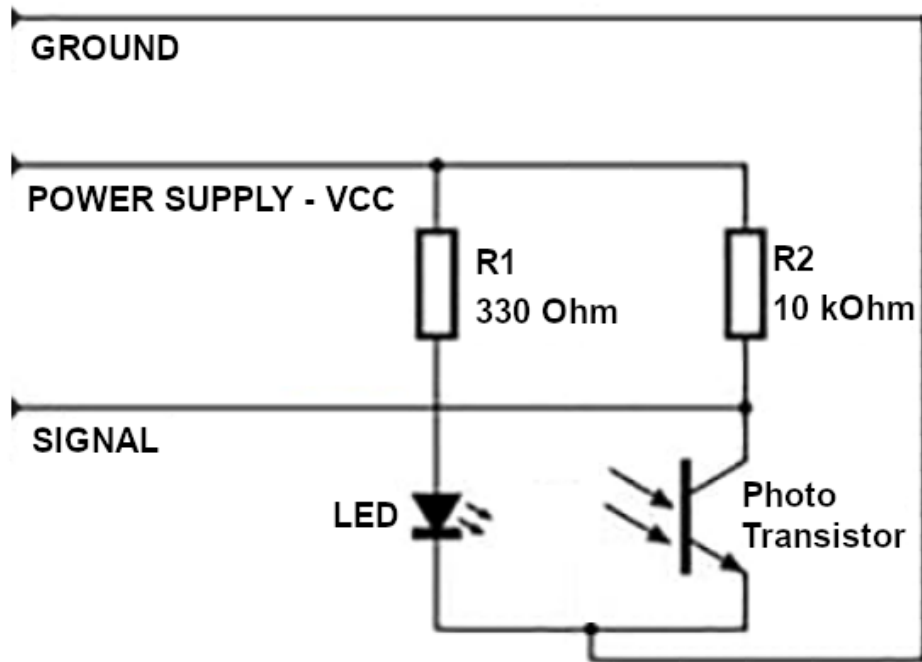
The pinout

The KY-010 light barrier detector module has three pins. The pinout diagram is shown on the following image:



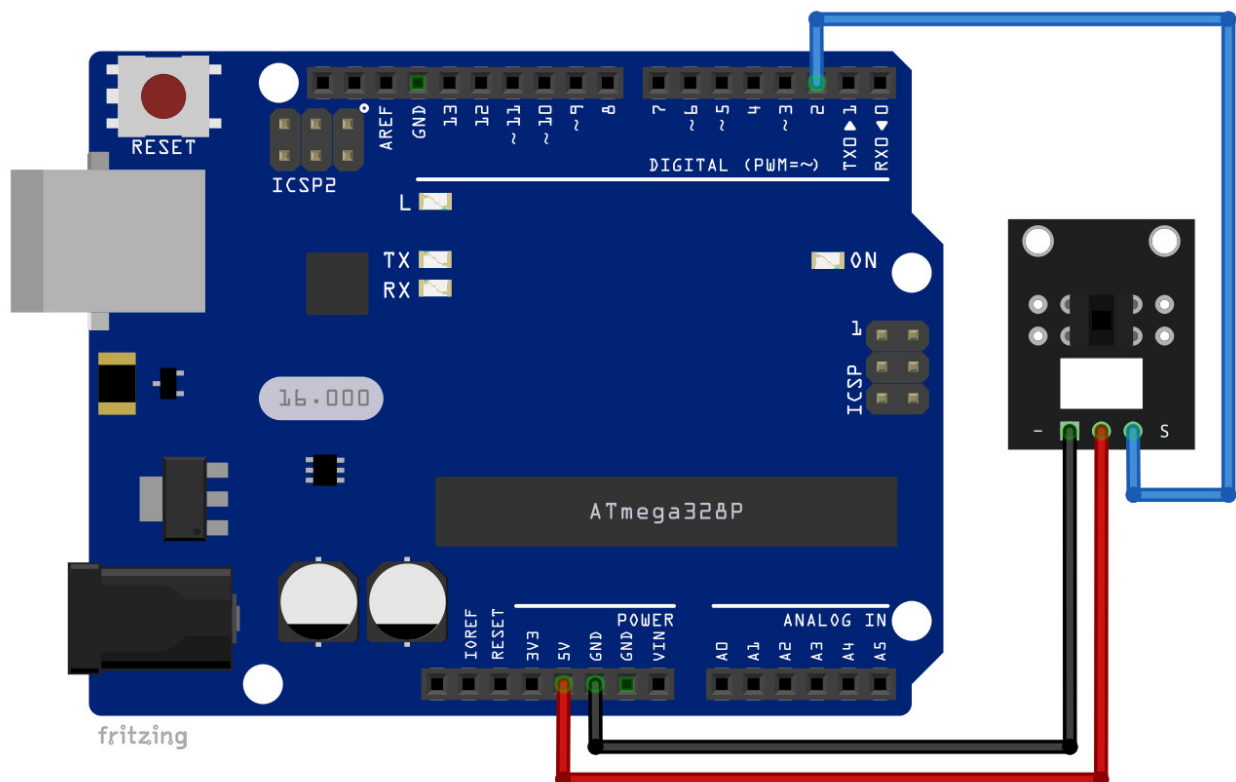
The schematic

The schematic diagram of the KY-010 module is shown on the image below:



Connecting the module with Uno

Connect the KY-010 module with the Uno as shown on the following connection diagram:



KY-010 pin	Uno pin	Wire color
S	D2	Blue wire
- (GND)	GND	Black wire
Middle pin (VCC)	5V	Red wire

Az-Delivery

Sketch example

```
#define SIGNAL_PIN 2
uint8_t value = 0;

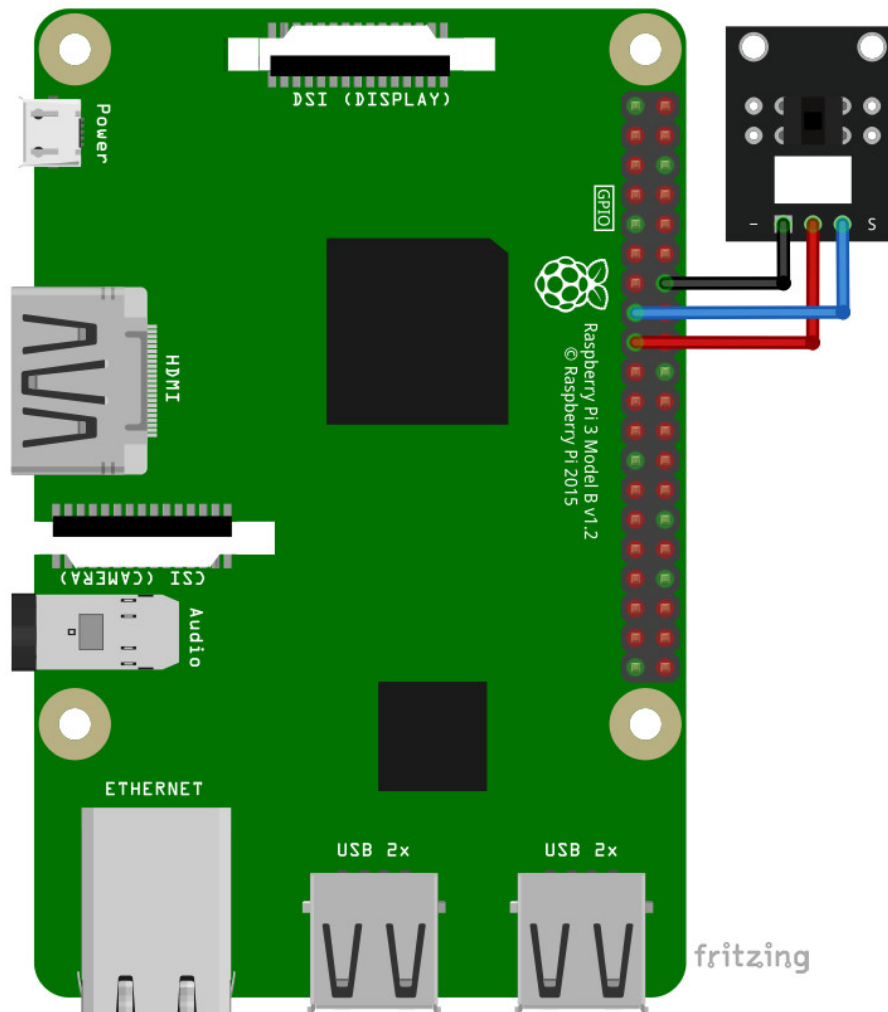
void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(SIGNAL_PIN, INPUT);
}

void loop() {
  value = digitalRead(SIGNAL_PIN);
  Serial.println(value); // for debugging
  if(value == HIGH) {
    digitalWrite(LED_BUILTIN, HIGH);
  }
  else {
    digitalWrite(LED_BUILTIN, LOW);
  }
  delay(1000);
}
```

When the sketch is uploaded to the Uno, the on-board LED which is connected to digital pin 13, is turned ON when there is a barrier between emitter and detector.

Connecting the module with Raspberry Pi

Connect the KY-010 module with the Raspberry Pi as shown on the following connection diagram:



KY-010 pin	Raspberry Pi pin	Physical pin	Wire color
- (GND)	GND	14	Black wire
S	GPIO22	15	Blue wire
Middle pin (VCC)	3V3	17	Red wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep

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

Signal_PIN = 22
GPIO.setup(Signal_PIN, GPIO.IN)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        state = GPIO.input(Signal_PIN)
        if state == 1:
            print('BARRIER DETECTED')

        else:
            print('No barrier')

        sleep(2)

except KeyboardInterrupt:
    print('\nScript end!')

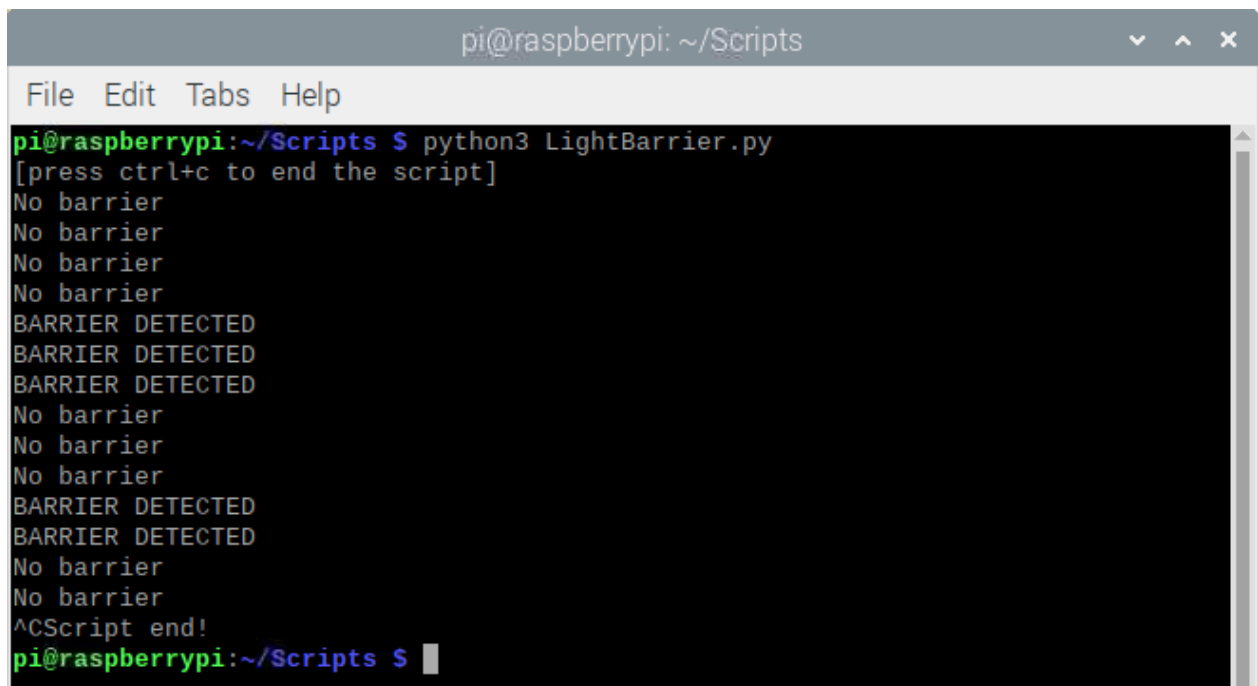
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *LightBarrier.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 LightBarrier.py

The result should look like the output on the image below:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 LightBarrier.py
[press ctrl+c to end the script]
No barrier
No barrier
No barrier
No barrier
BARRIER DETECTED
BARRIER DETECTED
BARRIER DETECTED
No barrier
No barrier
No barrier
BARRIER DETECTED
BARRIER DETECTED
No barrier
No barrier
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL + C* on the keyboard.

KY-011 Bi-color 5mm LED Module

Introduction

On-board the KY-011 module, there is an LED, which is two LEDs in one (Red and Green LEDs in one RG LED). So the KY-011 LED can produce light in two different colors, RG – Red and Green. Each color can be turned *ON* separately or both at the same time. By mixing these two colors a few other colors can be created. Also the brightness level of each color can be changed individually, with PWM (Pulse Width Modulation), which result in new colors.

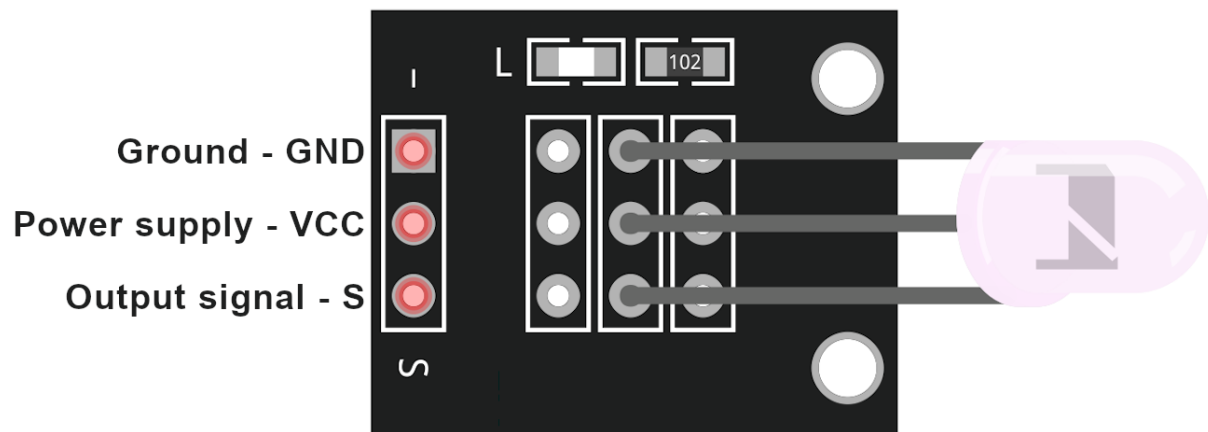
The module has three pins, two for the red and green light, and the third is for the ground. Each pin for both colors on the module has one resistor in series with the pin. These resistors are used as current flow limitation for LED protection.

Specifications

Operating voltage range	from 3.3V to 5V DC
Operating temperature	-25°C to 80°C [-13°F to 176°F]
Dimensions	19x15mm [0.73x0.6in]

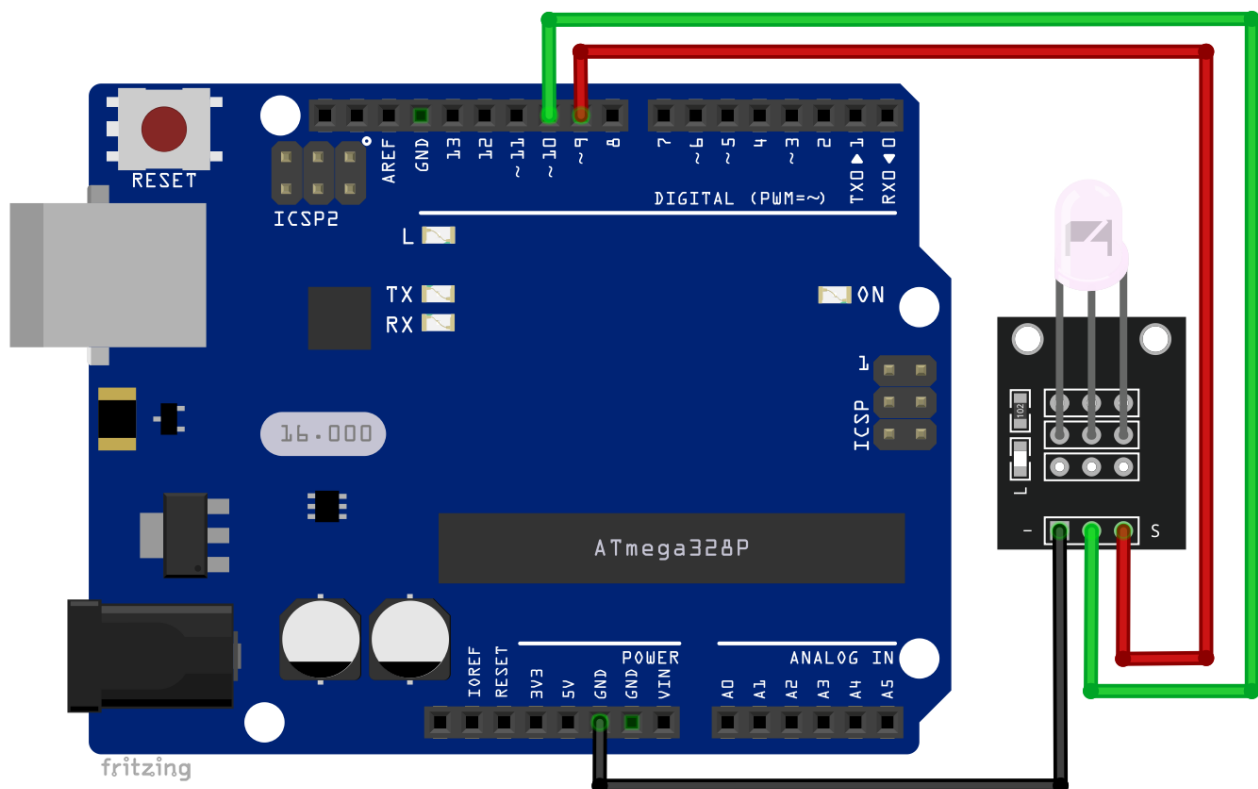
The pinout

The KY-011 bi-color 5mm LED module has three pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-011 module with the Uno as shown on the following connection diagram:



KY-011 pin	Uno pin	Wire color
S	D9	Red wire
Middle pin	D10	Green wire
- (GND)	GND	Black wire

Az-Delivery

Sketch example

```
#define RED_PIN 9
#define GREEN_PIN 10

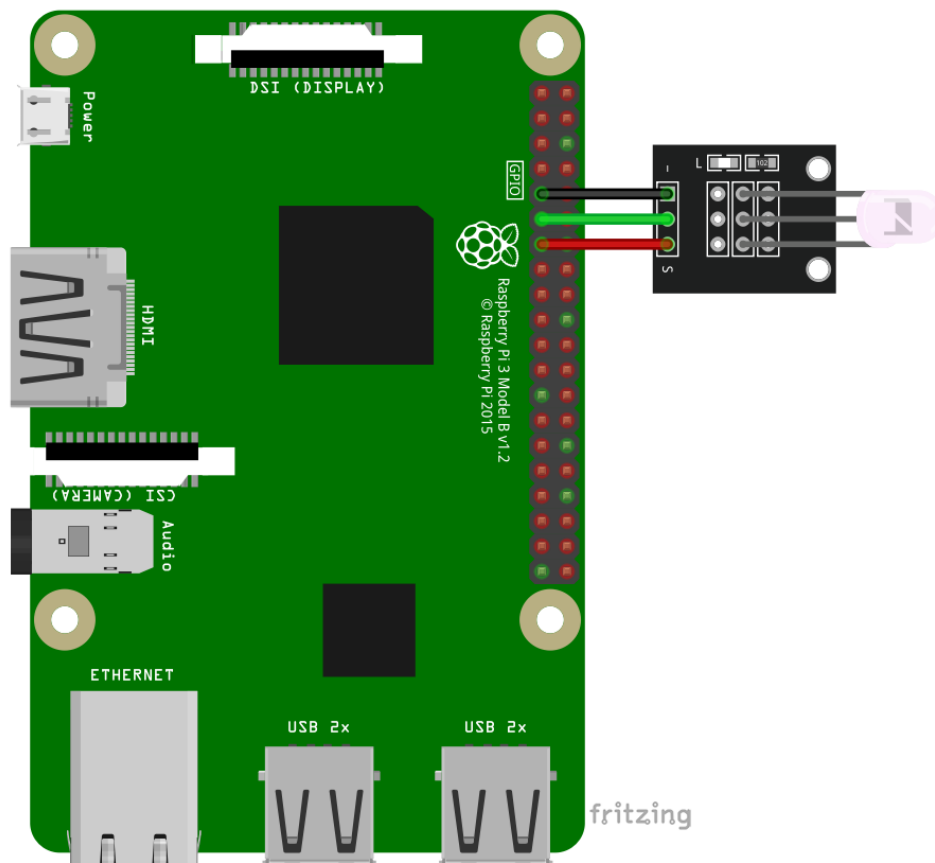
void setup() {
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(RED_PIN, OUTPUT);
}

void loop() {
  // first color
  analogWrite(GREEN_PIN, 0);
  analogWrite(RED_PIN, 250);
  delay(1000);

  // second color
  analogWrite(GREEN_PIN, 100);
  analogWrite(RED_PIN, 255);
  delay(1000);
}
```

Connecting the module with Raspberry Pi

Connect the KY-011 module with the Raspberry Pi as shown on the following connection diagram:



KY-011 pin	Raspberry Pi pin	Physical pin	Wire color
- (GND)	GND	9	Black wire
Middle pin (Green)	GPIO17	11	Green wire
S	GPIO27	13	Red wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
Green_PIN = 23
Red_PIN = 24
GPIO.setup(Green_PIN, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Red_PIN, GPIO.OUT, initial=GPIO.LOW)
green = GPIO.PWM(Green_PIN, 100) # PWM, frequency 100Hz
red = GPIO.PWM(Red_PIN, 100) # PWM, frequency 100Hz
green.start(1)
red.start(1)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        red.ChangeDutyCycle(90) # Purple color
        green.ChangeDutyCycle(1)
        sleep(1) # Wait for a second
        red.ChangeDutyCycle(99) # Orange color
        green.ChangeDutyCycle(10)
        sleep(1) # Wait for a second

except KeyboardInterrupt:
    print('\nScript end!\n')

finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *TwoColorLed5m.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

```
python3 TwoColorLed5mm.py
```

And watch the color changing magic.

When starting the PWM, with the following line of code:

```
red.start(1)
```

use any number different than 0, because if the number 0 is used, that disables PWM on the selected GPIO pin.

Also, when using the following line of code:

```
red.ChangeDutyCycle(1)
```

to change the duty cycle of the PWM signal, do not use number 0, because of the same reason as for the *red.start(1)*.

To stop the script press *CTRL + C* on the keyboard.



KY-012 Active Buzzer Module

Introduction

Buzzers come in two varieties, active and passive. The buzzer on-board the KY-012 module is an active buzzer, and it just outputs a single tone when it is connected to the power supply.

A passive buzzer is similar to a loudspeaker and needs an analog signal to make it work.

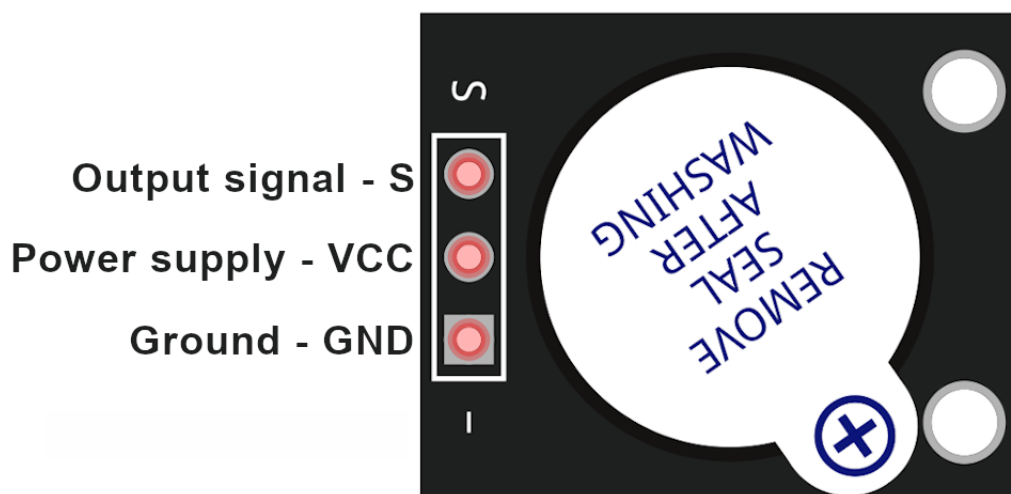
The active buzzer module has three pins. It creates an audible sound at $2.5kHz$ without the need for PWM (Pulse Width Modulation) or any additional complex code. The only requirement to generate sound is to set the signal pin in the *HIGH* state.

Specifications

Operating voltage range	from 3.3V to 5V DC
Operating current	30mA
Resonance frequency	2.5kHz $\pm$ 300Hz continuous
Sound output	85dB at 100mm
Storage temperature	from -30°C to 105°C
Operating temperature	from -20°C to 70°C
Dimensions	18x15mm [0.71x0.6in]

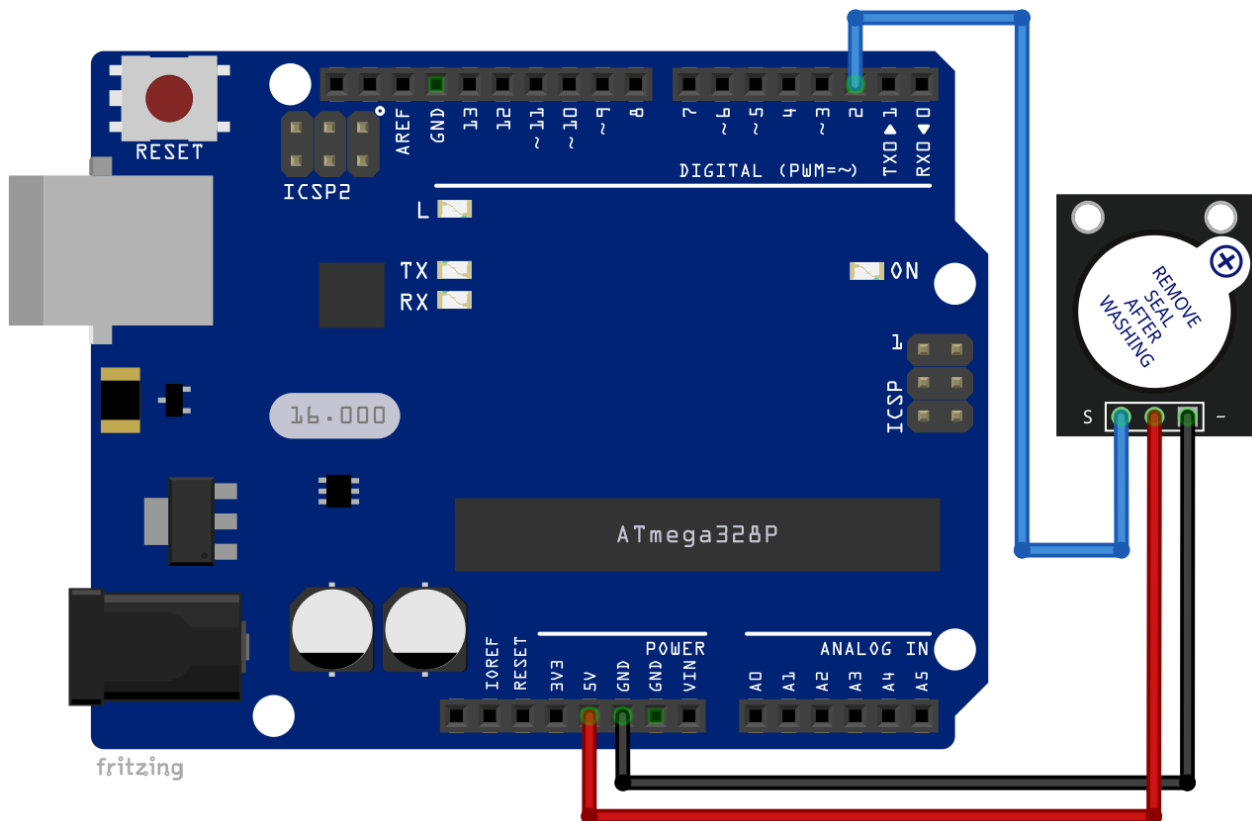
The pinout

The KY-012 active buzzer module has three pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-012 module with the Uno as shown on the following connection diagram:



KY-012 pin	Uno Pin	Wire color
S	D2	Blue wire
- (GND)	GND	Black wire
Middle pin (VCC)	5V	Red wire

Az-Delivery

Sketch example

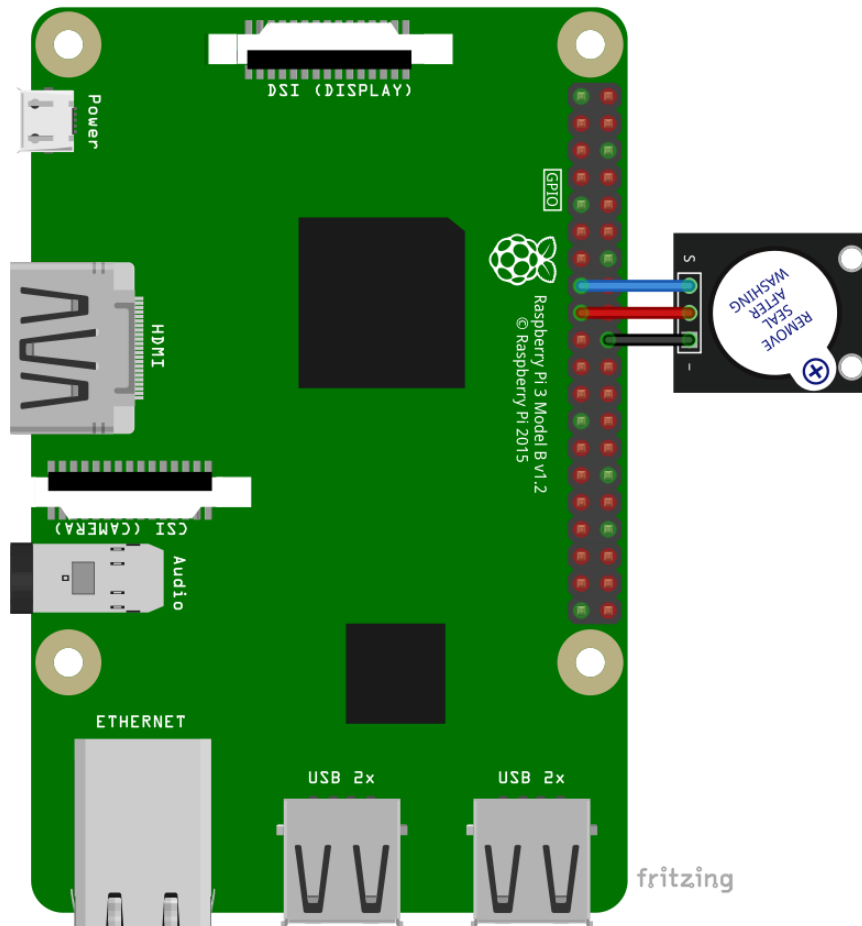
```
#define BIZZER_PIN 2

void setup() {
  pinMode(BIZZER_PIN, OUTPUT);
}

void loop() {
  digitalWrite(BIZZER_PIN, HIGH);
  delay(1000);
  digitalWrite(BIZZER_PIN, LOW);
  delay(1000);
}
```

Connecting the module with Raspberry Pi

Connect the KY-012 module with the Raspberry Pi as shown on the following connection diagram:



KY-012 pin	Raspberry Pi pin	Physical pin	Wire color
S	GPIO2	15	Blue wire
Middle pin (VCC)	3V3	17	Red wire
- (GND)	GND	20	Black wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep

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

Buzzer_PIN = 22
GPIO.setup(Buzzer_PIN, GPIO.OUT, initial=GPIO.LOW)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        print('Buzzer will be on for 3 seconds')
        GPIO.output(Buzzer_PIN, GPIO.HIGH) # Buzzer ON
        sleep(3) # Wait for 3 seconds
        print('Buzzer will be off for 1 second')
        GPIO.output(Buzzer_PIN, GPIO.LOW) # Buzzer OFF
        sleep(1) # Wait for a second

except KeyboardInterrupt:
    print('\nScript end!\n')

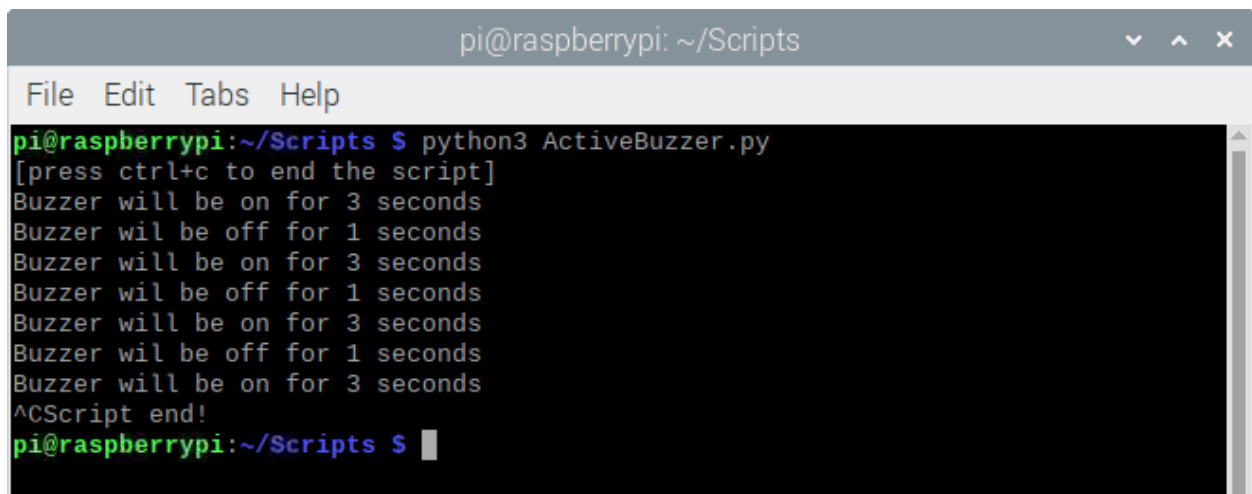
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *ActiveBuzzer.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 ActiveBuzzer.py

The result should look like the output on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 ActiveBuzzer.py
[press ctrl+c to end the script]
Buzzer will be on for 3 seconds
Buzzer will be off for 1 seconds
Buzzer will be on for 3 seconds
Buzzer will be off for 1 seconds
Buzzer will be on for 3 seconds
Buzzer will be off for 1 seconds
Buzzer will be on for 3 seconds
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL + C* on the keyboard.

KY-013 Analog Temperature Thermistor Sensor Module

Introduction

The thermistor is a temperature-dependent resistor, which changes its resistance with temperature change.

The thermistor inside the KY-013 module is $10k\Omega$ thermistor. It is a resistor with a Negative Temperature Coefficient or short NTC Resistor. There is also a thermistor with Positive Temperature Coefficient but this is not in scope of this eBook.

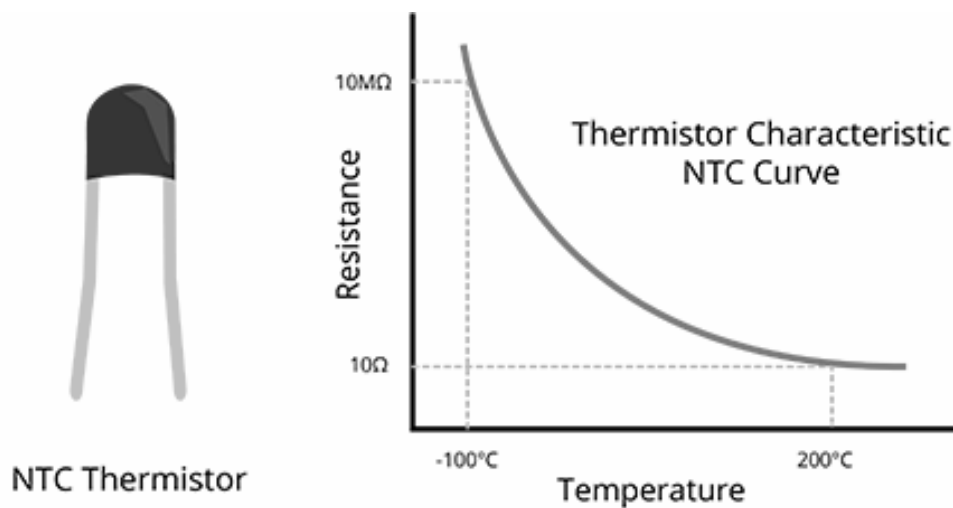
What NTC means is that resistance of the thermistor decreases with the increase of temperature, and increases when temperature decreases. Thermistors are made so that the resistance changes drastically with temperature so that it can be 100Ω or more of change per degree of temperature.

At room temperature (25°C) the resistance of the thermistor is $10k\Omega$. This can be used to accurately measure temperatures in a range from -55°C to 125°C .

Specifications

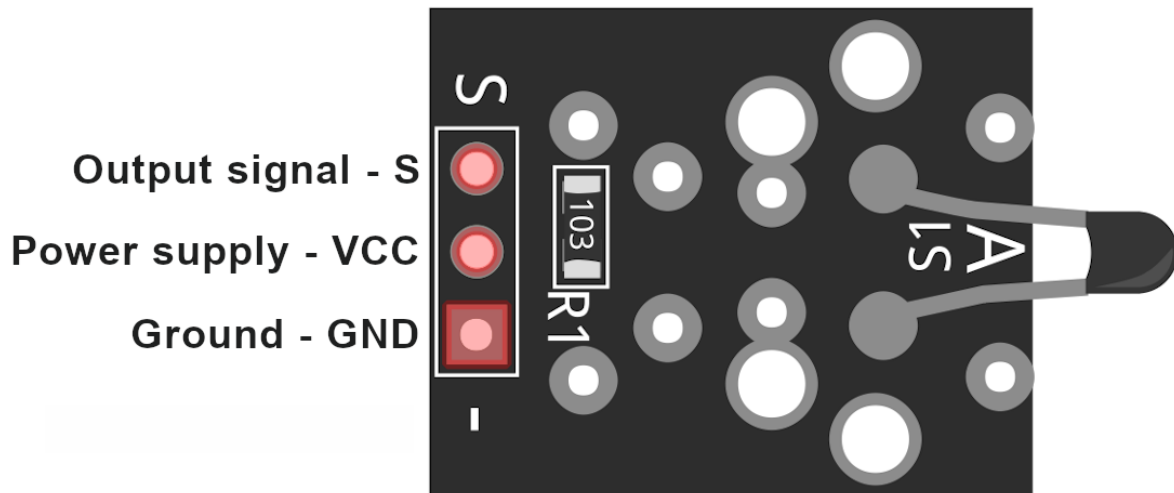
Operating voltage range	from 3.3V to 5V DC
Measures temperatures	from -55°C to +125°C
Accuracy	±0.5°C
Temperature sensor	NTC thermistor
Dimensions	30x15mm [1.2x0.6in]
Multi-point temperature measurement	

The Resistance/Temperature curve of the thermistor is not linear, so it has to be taken into an account when the temperature is calculated from the measured resistance.



The pinout

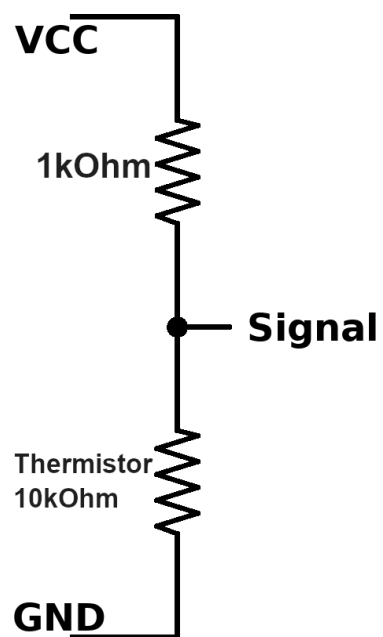
The KY-013 analog temperature thermistor sensor module has three pins. The pinout diagram is shown on the following image:



Az-Delivery

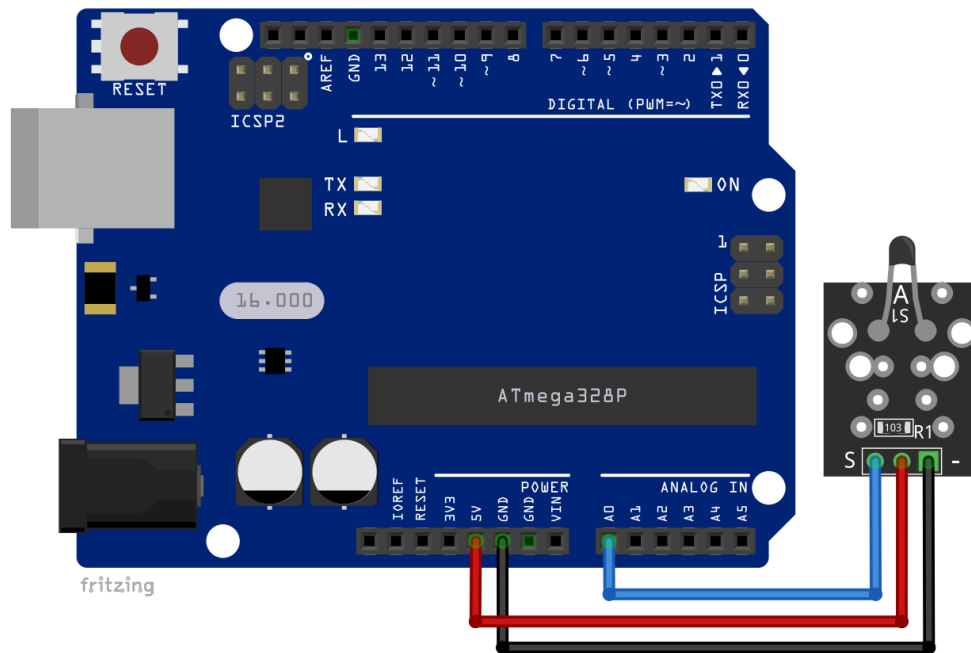
The schematic

On-board the KY-013 module there are one $10k\Omega$ NTC thermistor and one $1k\Omega$ resistor. These two resistors form a voltage divider, as shown on the image below:



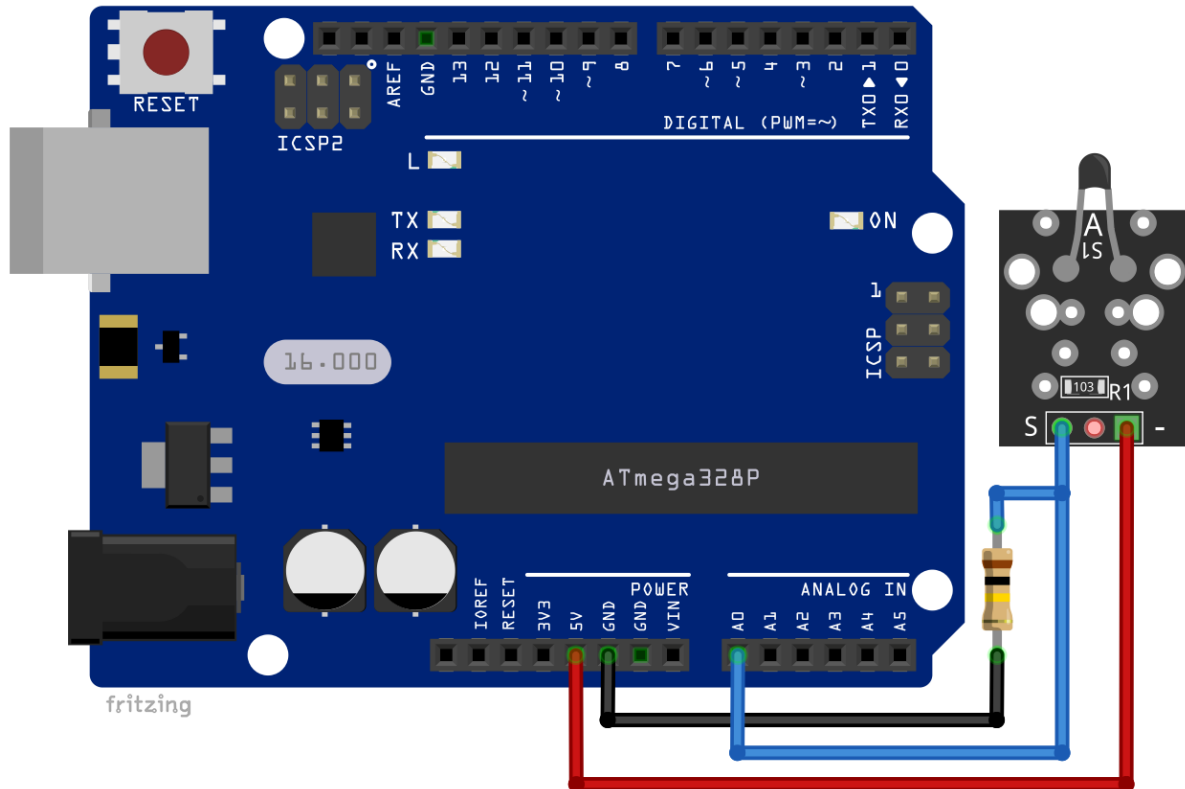
Connecting the module with Uno

Connect the KY-013 module with the Uno as shown on the following connection diagram:



The setup does not give the output that can be used. That is because, by changing the thermistor resistance, the signal pin voltage change is so small that analog to digital converter on-board Uno can not detect it. One additional resistor had to be connected, a resistor that has resistance that is bigger than $10k\Omega$, for example, the $100k\Omega$ resistor.

Connect the KY-013 module with the Uno as shown on the following connection diagram:



KY-013 pin	Uno pin	Wire color
S	A0	Blue wire
-	5V	Red wire
middle pin	Not connected	

The S pin is connected to the *GND* pin of the Uno via *100kΩ* resistor (pull-down resistor) **Black wire**

Az-Delivery

Sketch example

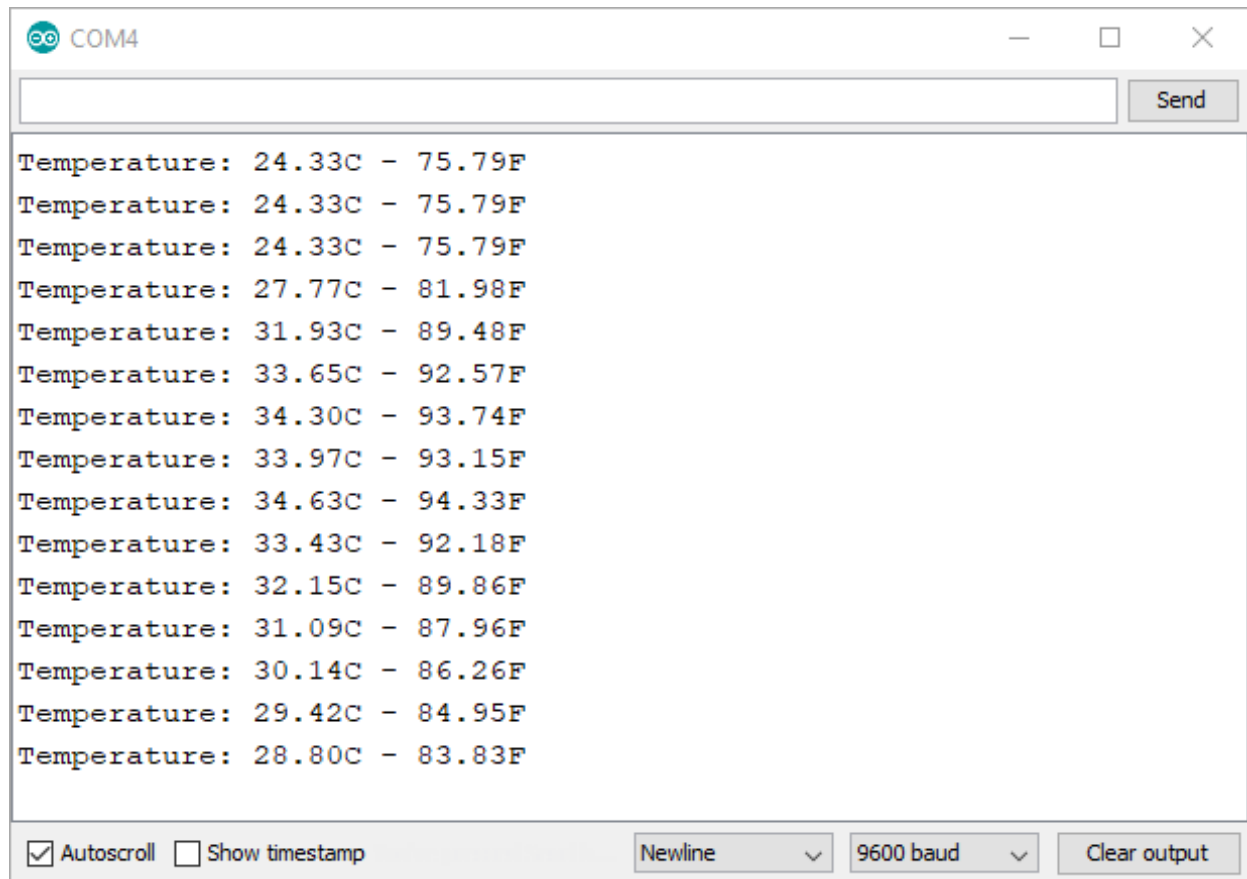
```
#define THERMISOR_PIN 0
uint16_t Vo;
float R1 = 10000;
float logR2, R2, T, Tc, Tf;
float c1 = 1.009249522e-03, c2 = 2.378405444e-04, c3 = 2.019202697e-07;

void setup() {
  Serial.begin(9600);
}

void loop() {
  Vo = analogRead(THERMISOR_PIN);
  R2 = R1 * (1023.0 / (float)Vo - 1.0);
  logR2 = log(R2);
  T = (1.0 / (c1 + c2 * logR2 + c3 * logR2 * logR2 * logR2));
  Tc = T - 273.15; // Temperature in Celsius
  Tf = (Tc * 9.0) / 5.0 + 32.0; // Temperature in Fahrenheit
  Serial.print("Temperature: ");
  Serial.print(Tc);
  Serial.print("C - ");
  Serial.print(Tf);
  Serial.println("F");
  delay(1000);
}
```

Az-Delivery

When the sketch is uploaded to the Uno, open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



If the thermistor on-board the KY-013 is touched by a human fingers, the temperature output changes as shown on the image above.

Az-Delivery

Sketch starts with creating a macro called *THERMISTOR_PIN*, where the integer value *0* is stored. This macro represents the analog input pin of the Uno on which the output pin of the KY-013 module is connected.

Next, several variables are created. Some of these variables hold (constants) and some of these variables are used to store data used in calculating the temperature from the thermistor.

In the *setup()* function serial communication is created with a baud rate of *9600bps*.

In the *loop()* function, the analog voltage on the analog input pin is read and the data is stored in the variable called *Vo*. Next, several lines of code are used to calculate temperature data (this algorithm is not covered in this eBook).

After this, temperature data in Celsius is stored in the variable called *Tc* and Fahrenheit data is stored in the variable called *Tf*.

At the end of *loop()* function, temperature data in Celsius and in Fahrenheit is displayed in the Serial Monitor.

Between two iterations of the *loop()* function there is a one second pause.

Connecting the module with Raspberry Pi

Connect the KY-013 module with the Uno as shown on the connection diagram in the chapter *Connecting the module with Uno*. Then connect the Uno with the Raspberry Pi via USB cable. Next, upload *nanpy* firmware to the Uno (as shown in the chapter *Uno as ADC*), and use the following code to read the KY-013 module:

```
from nanpy import (ArduinoApi, SerialManager)
from time import sleep
import math
try:
    connection1 = SerialManager()
    a = ArduinoApi(connection=connection1)
except:
    print('Failed to connect to the Arduino')
value = 0
degree_sign = u'\xb0' # degree sign
analogVal = 0
R1 = 10000
logR2 = 0.0
T = 0.0
Tc = 0.0
Tf = 0.0
c1 = 0.001009249522
c2 = 0.0002378405444
c3 = 0.0000002019202697
```


Az-Delivery

```
print('[Press CTRL + C to end the script!]\n')
try: #
    while True:
        analogVal = a.analogRead(0)
        R2 = R1 * (1023.0 / analogVal - 1.0)
        logR2 = math.log(R2)

        T = (1.0/(c1 + c2*logR2 + c3*logR2*logR2*logR2))
        Tc = T - 273.15
        Tf = (Tc * 9.0) / 5.0 + 32.0

        print('Temperature: {:.2f}{}C - {:.2f}
        {:.2f}F'.format(Tc,
                                degree_sign, Tf, degree_sign))

        sleep(1)

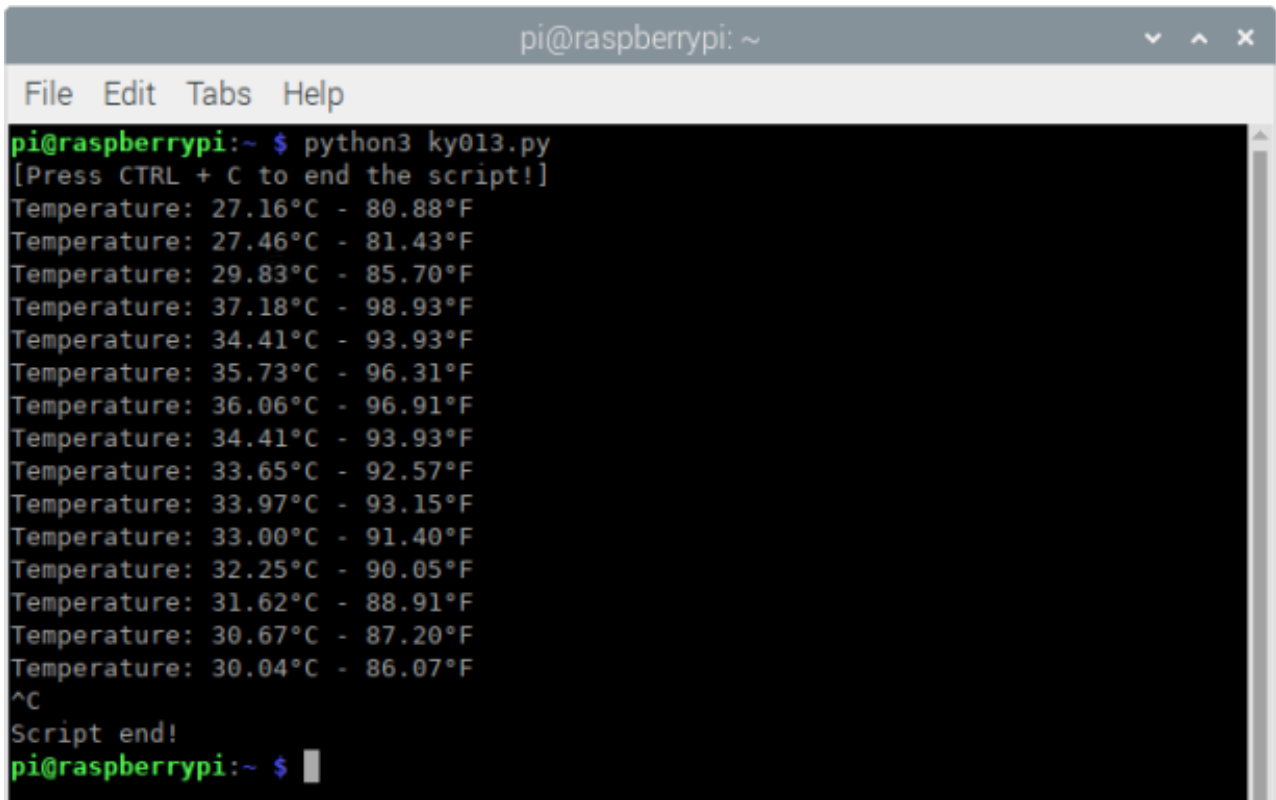
except KeyboardInterrupt:
    print('\nScript end!')
```

Az-Delivery

Save the script by the name *ky013.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 ky013.py

The result should look like the output on the image below:

A screenshot of a terminal window titled 'pi@raspberrypi: ~'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal output shows the command 'python3 ky013.py' being executed. The script prints a series of temperature readings in both Celsius and Fahrenheit, followed by a prompt to press CTRL + C to end the script. After pressing CTRL + C, the script prints 'Script end!' and returns to the command prompt.

```
pi@raspberrypi:~ $ python3 ky013.py
[Press CTRL + C to end the script!]
Temperature: 27.16°C - 80.88°F
Temperature: 27.46°C - 81.43°F
Temperature: 29.83°C - 85.70°F
Temperature: 37.18°C - 98.93°F
Temperature: 34.41°C - 93.93°F
Temperature: 35.73°C - 96.31°F
Temperature: 36.06°C - 96.91°F
Temperature: 34.41°C - 93.93°F
Temperature: 33.65°C - 92.57°F
Temperature: 33.97°C - 93.15°F
Temperature: 33.00°C - 91.40°F
Temperature: 32.25°C - 90.05°F
Temperature: 31.62°C - 88.91°F
Temperature: 30.67°C - 87.20°F
Temperature: 30.04°C - 86.07°F
^C
Script end!
pi@raspberrypi:~ $
```

To stop the script press *CTRL* + *C* on the keyboard.

This script does exactly the same thing on the same way as the sketch for Uno.



KY-015 DHT11 Temperature Sensor Module

Introduction

The DHT11 is relative humidity and temperature sensor that outputs a digital signal. It uses a capacitive humidity sensor and a thermistor to measure the humidity and the temperature of the surrounding air.

Temperature measuring range of the DHT11 is from 0°C to $+50^{\circ}\text{C}$, with $\pm 2^{\circ}\text{C}$ accuracy.

Humidity measuring range is from 20% to 90%, with accuracy from $\pm 5\%$.

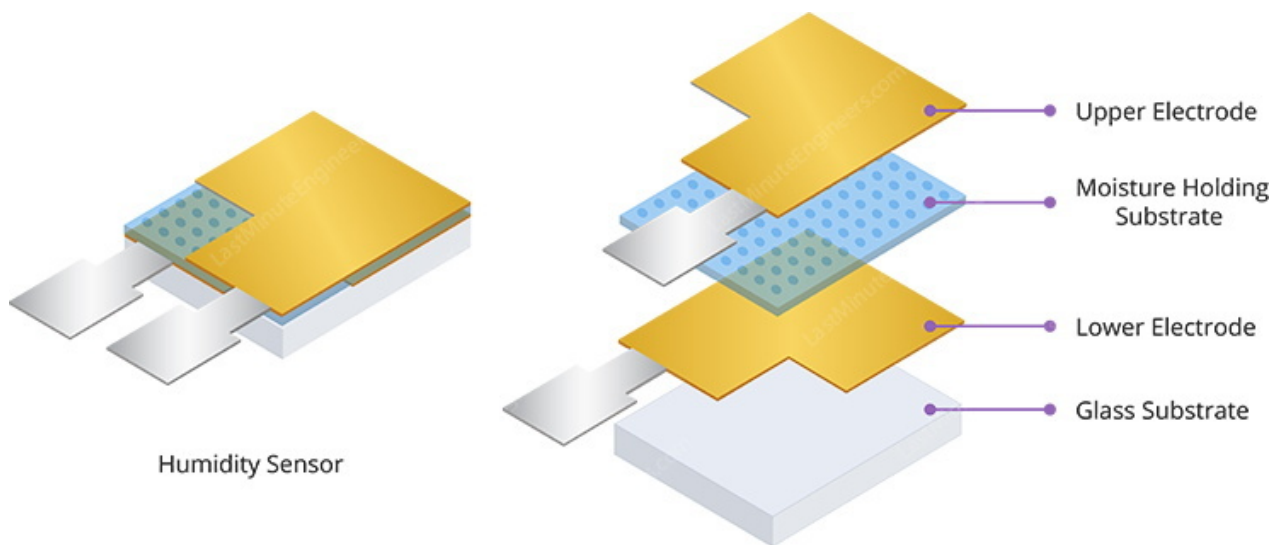
Sampling rate of the sensor is 1Hz, which means that sensor data can be read only once per second.

Specifications

Operating voltage range	from 3.3V to 5V DC
Max operating current	2.5mA max
Humidity range	20% - 90% with accuracy of 5%
Temperature range	0°C - 50°C with accuracy of $\pm 2^{\circ}\text{C}$
Sampling rate	1Hz (reading once per second)
Dimensions	15x32x9mm [0.6x1.3x0.35in]

Working principle of the DHT11 sensor

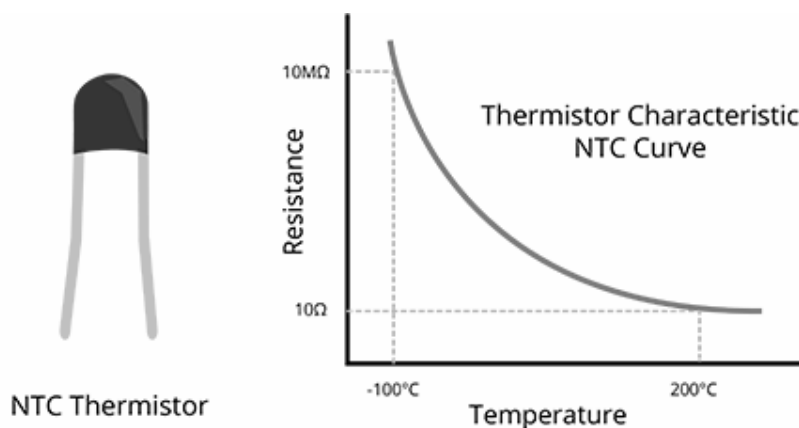
Inside the case, on the sensing side of DHT11 sensor, there is a humidity sensing component along with a NTC temperature sensor (or thermistor).



Humidity sensing component is used to measure humidity, which has two electrodes with moisture holding substrate (usually salt or conductive plastic polymer) sandwiched between the electrodes. The ions are released by the substrate as water vapor is absorbed by it, which in turn increases the conductivity between the electrodes. The change in resistance between the two electrodes is proportional to the relative humidity. The higher relative humidity decreases the resistance between the electrodes, while lower relative humidity increases the resistance between the electrodes.

Az-Delivery

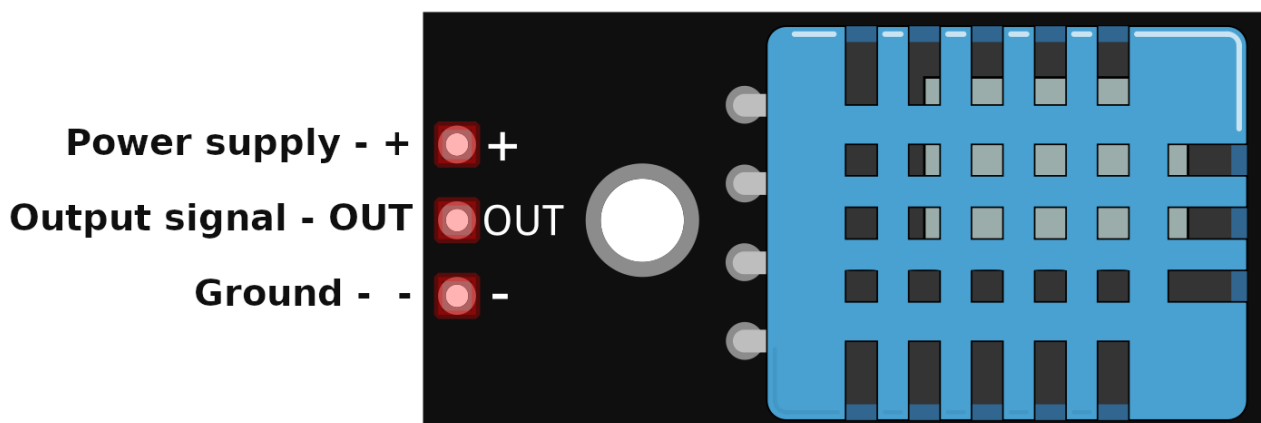
The temperature sensing part of the DHT11 sensor consists of an NTC thermistor. A thermistor is a thermal resistor, a resistor that changes its resistance with temperature. Technically, all resistors are thermistors, their resistance changes slightly with temperature, but the change is usually small and difficult to measure. Thermistors are made so that the resistance changes drastically with temperature. The change can be 100Ω or more per degree of temperature. The term *NTC* means *Negative Temperature Coefficient*, which means that the resistance decreases with the increase of temperature.



On the other side, there is a small PCB with an 8 bit SOIC14 packaged integrated circuit. The integrated circuit measures and processes the analog signal with stored calibration coefficients, does analog to digital conversion and outputs a digital signal with the data that contains information for temperature and humidity.

The pinout

The KY-015 DHT11 temperature sensor module has three pins. The pinout diagram is shown on the following image:



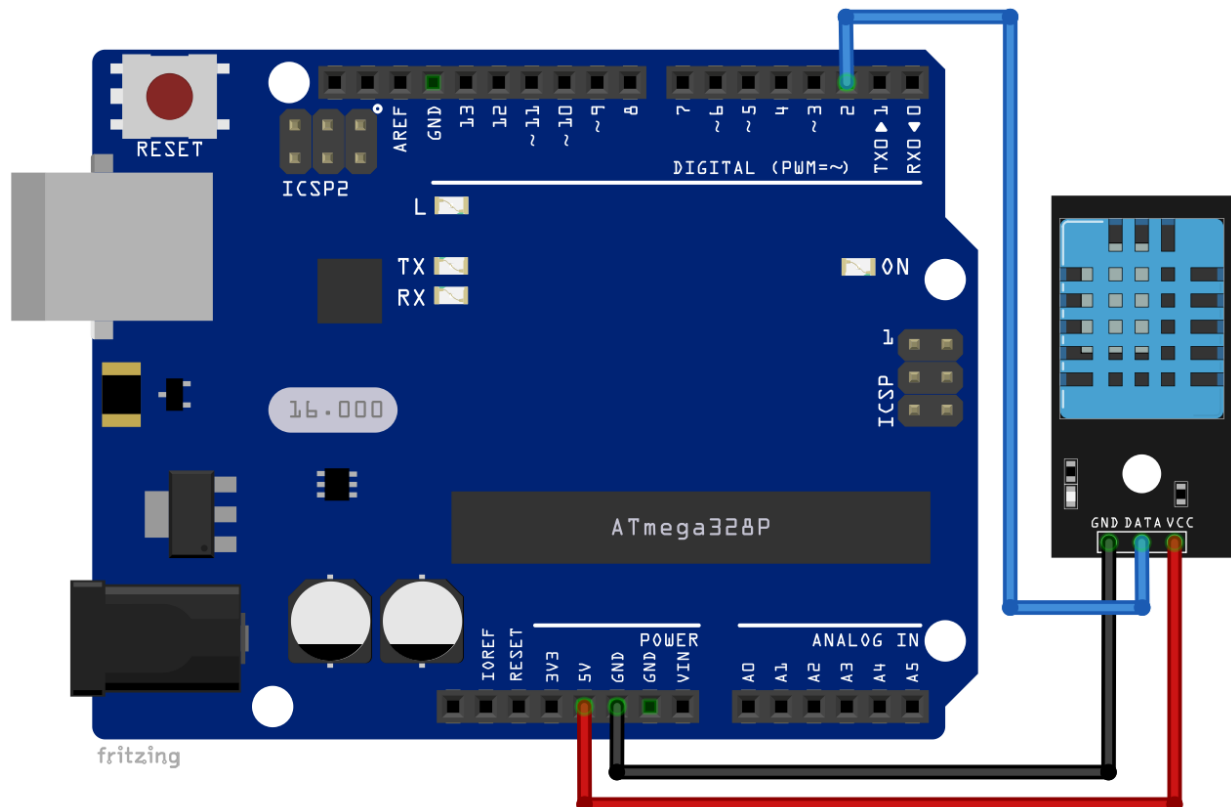
Power supply (+) pin - supplies power for the sensor. Although supply voltage can range between 3.3V and 5V, a 5V supply is recommended. In case of a 5V power supply, a cable that connects sensor and microcontroller can be up to 20 meters long. However, with 3.3V supply voltage, cable length shall not be longer than one meter, otherwise, the line voltage drop will lead to errors in measurement.

OUT (DATA) pin - is the data pin, and it is used for communication between the sensor and the microcontroller.

GND (-) pin - is a ground pin and should be connected to the common ground, or 0V (on Uno or Raspberry Pi).

Connecting the module with Uno

Connect the KY-015 module with the Uno as shown on the following connection diagram:



KY-015 pin	Uno pin	Wire color
DATA	D2	Blue Wire
GND	GND	Black Wire
VCC	5V	Red Wire



Library for Arduino IDE

To use the DHT11 module with Uno it is recommended to install an external library. The library used in this eBook is called the *SimpleDHT* library, which can be downloaded from [GitHub](#) page. When the *.zip* file is downloaded, open Arduino IDE and go to:

Sketch > Include Library > Add .ZIP Library
and add the downloaded *.zip* file.

Sketch example

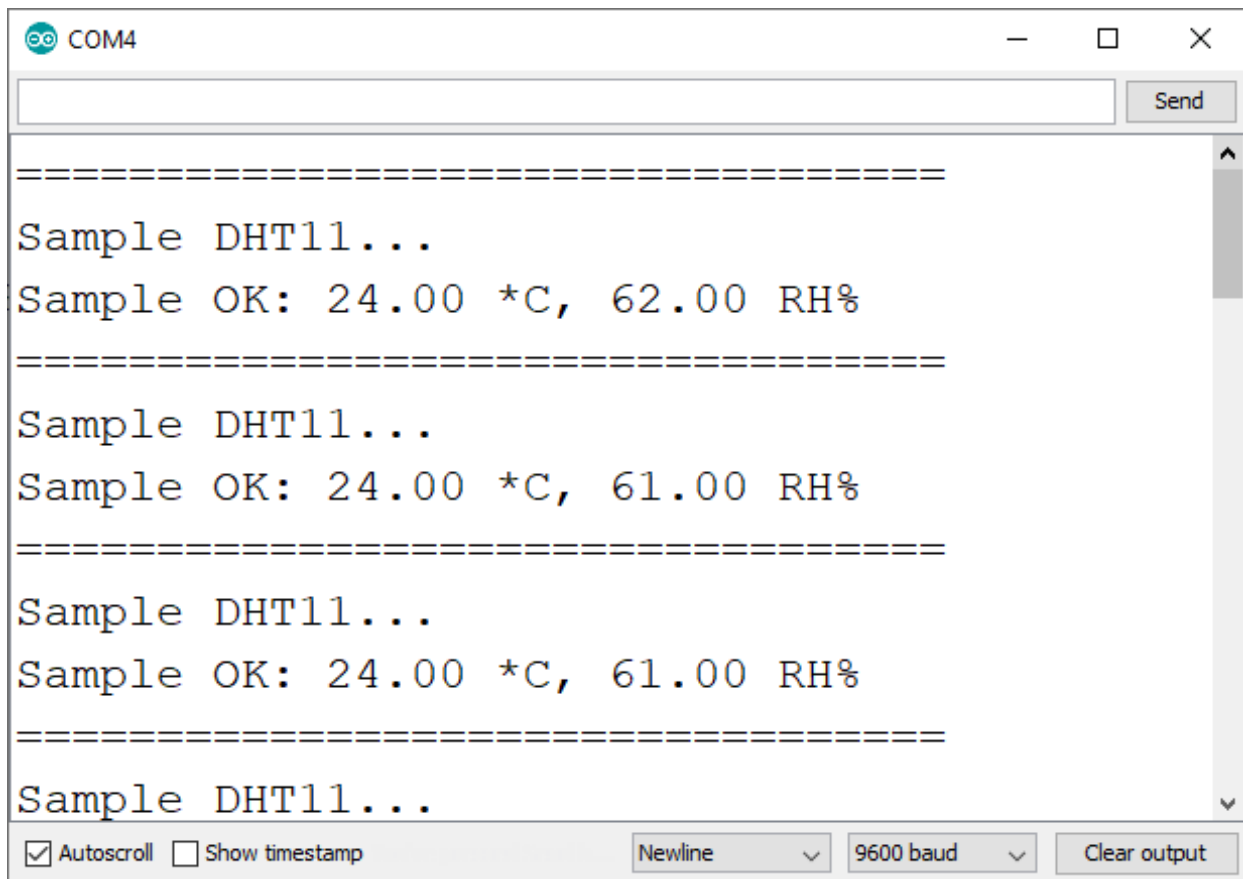
Go to: *File > Examples > SimpleDHT > DHT11Default* and open the sketch. The following code is the sketch code:

```
#include <SimpleDHT.h>
#define DHT11_PIN 2
SimpleDHT11 dht11(DHT11_PIN);
void setup() {
    Serial.begin(9600);
}
void loop() {
    Serial.println("=====");
    Serial.println("Sample DHT11...");
    float temperature = 0;
    float humidity = 0;
    int err = SimpleDHTErrSuccess;
    if((err=dht11.read2(&temperature, &humidity, NULL)) !=

        SimpleDHTErrSuccess){
        Serial.print("Read DHT11 failed, err=");
        Serial.println(err);
        delay(2000);
        return;
    }
    Serial.print("Sample OK: ");
    Serial.print((float)temperature);
    Serial.print(" *C, ");
    Serial.print((float)humidity);
    Serial.println(" RH%");
    delay(1500); // DHT11 sampling rate is 1HZ.
}
```

Az-Delivery

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the image below:



The screenshot shows the Serial Monitor window for COM4. The output text is as follows:

```
=====
Sample DHT11...
Sample OK: 24.00 *C, 62.00 RH%
=====
Sample DHT11...
Sample OK: 24.00 *C, 61.00 RH%
=====
Sample DHT11...
Sample OK: 24.00 *C, 61.00 RH%
=====
Sample DHT11...
```

The bottom of the window contains the following controls:

- ☒ Autoscroll
- ☐ Show timestamp
- Newline (dropdown menu)
- 9600 baud (dropdown menu)
- Clear output

Az-Delivery

Sketch starts with importing the *SimpleDHT* library.

Next, a macro called *DHT11_PIN* is created and initialized with integer value 2. This macro represents a digital I/O pin on which the *DATA* pin of the DHT11 module is connected.

Then, the object called *dht11* is created where the previously created macro is used to initialize it.

In the *setup()* function the serial communication is started with a baud rate of *9600bps*.

In the *loop()* function, the sensor data is read every 1.5 seconds, with the following lines of code:

```
float temperature = 0;
float humidity = 0;
int err = SimpleDHTErrSuccess;
if((err=dht11.read2(&temperature, &humidity, NULL)) !=

    SimpleDHTErrSuccess){
    Serial.print("Read DHT11 failed, err=");
    Serial.println(err);
    delay(2000);
    return;
}
```

Az-Delivery

Where the *err* variable represents the error during reading the sensor data.

If there is error while reading the sensor, the message:

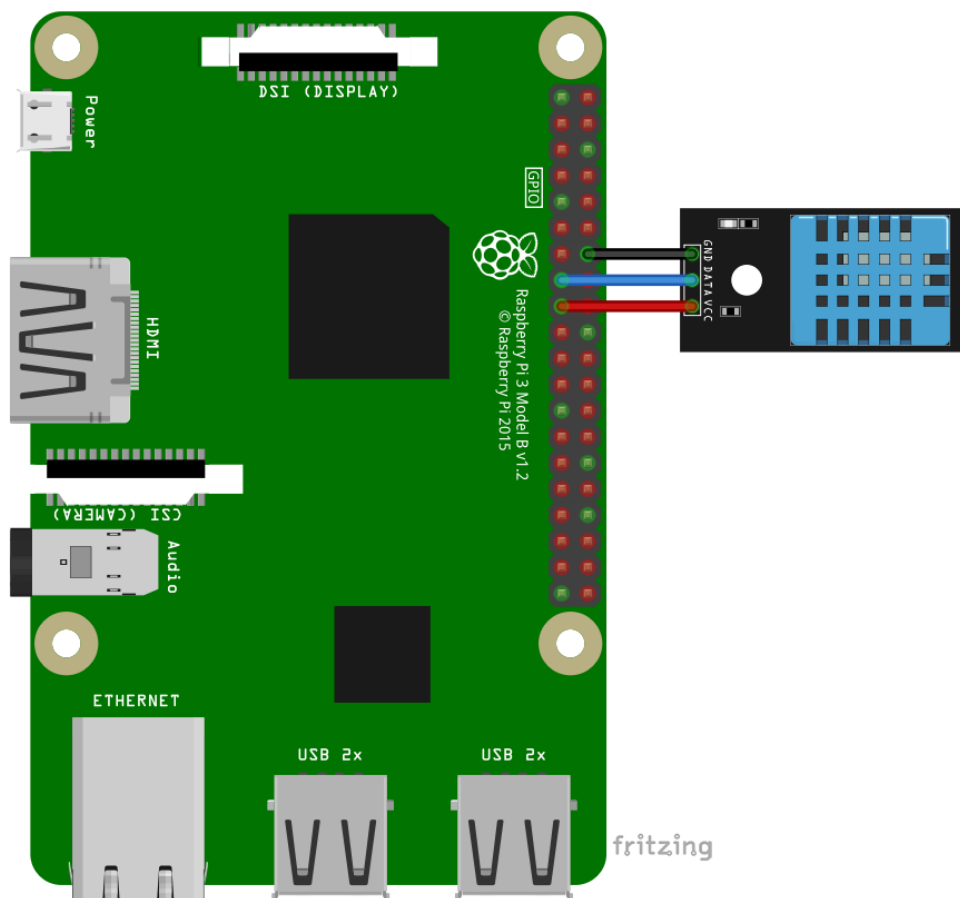
Read DHT11 failed, err="error data"

where *error data* is detailed information about the error. If there is no error while reading the sensor, sensor data is saved in the *temperature* and *humidity* variables.

After this, the sensor data is displayed in the Serial Monitor.

Connecting the module with Raspberry Pi

Connect the KY-015 module with the Raspberry Pi as shown on the following connection diagram:



KY-015 pin	Raspberry Pi pin	Physical pin	Wire color
GND	GND	14	Black Wire
DATA	GPI022	15	Blue Wire
VCC	3V3	17	Red Wire



Library and tools for Python

To use the DHT11 sensor with the Raspberry Pi, it is recommended to install an external library. The library we are going to use is called *Adafruit_DHT*.

To install it, first we need to make sure *Raspberry Pi OS* is up to date.

Start your Raspberry Pi, open the terminal and run the following commands:

- The first command is for making the system up to date:

```
sudo apt-get update && sudo apt-get upgrade -y
```

- The second command is for installing the *python3-pip* app:

```
sudo apt-get install python3-pip
```

We use the *pip* app to install the *Adafruit_DHT* library.

- The third command is for installing other apps which the *pip* app uses during installation:

```
sudo python3 -m pip install --upgrade pip setuptools wheel
```

After this is completed, we can install the *Adafruit_DHT* library. To do so, open the terminal and run the following command:

```
sudo pip3 install Adafruit_DHT
```

Az-Delivery

Python script

```
import Adafruit_DHT
from time import sleep

sensor = Adafruit_DHT.DHT11
pin = 22 # DHT11 sensor connected to GPIO22
degree_sign = u'\xb0' # degree sign

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
        sleep(1.5)

        if humidity is not None and temperature is not None:
            print('Temp = {:.1f}{}C - Humidity = {:.1f}%'.format(temperature, degree_sign, humidity))
        else:
            print('Failed to get reading. Try again!')

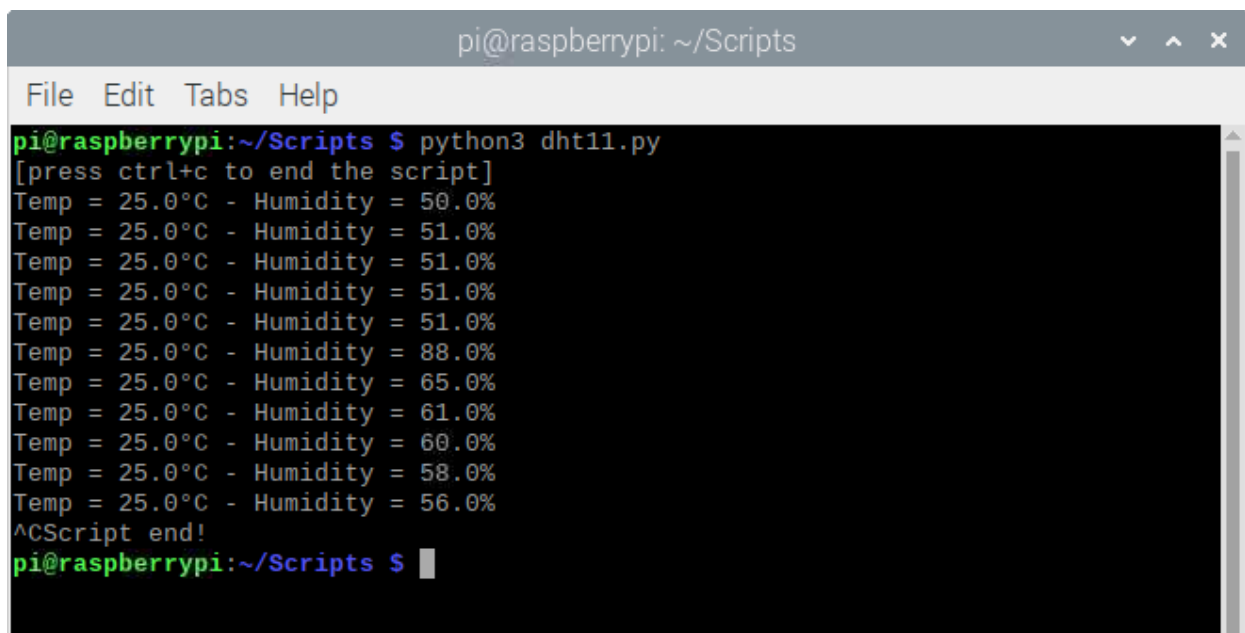
except KeyboardInterrupt:
    print('\nScript end!')
```


Az-Delivery

Save the script by the name *dht11.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 dht11.py

The result should look like the output on the image below:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 dht11.py
[press ctrl+c to end the script]
Temp = 25.0°C - Humidity = 50.0%
Temp = 25.0°C - Humidity = 51.0%
Temp = 25.0°C - Humidity = 51.0%
Temp = 25.0°C - Humidity = 51.0%
Temp = 25.0°C - Humidity = 51.0%
Temp = 25.0°C - Humidity = 88.0%
Temp = 25.0°C - Humidity = 65.0%
Temp = 25.0°C - Humidity = 61.0%
Temp = 25.0°C - Humidity = 60.0%
Temp = 25.0°C - Humidity = 58.0%
Temp = 25.0°C - Humidity = 56.0%
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL + C* on the keyboard.



KY-016 RGB LED Module

Introduction

The LED on-board the KY-016 module is actually three LEDs in one (Red, Green and Blue LED in one RGB LED). This LED can produce light in three different colors, RGB – Red, Green and Blue. Each color can be turned *ON* separately, or all at the same time. By mixing these three colors any other color can be created. The brightness level of each color can be changed individually, with PWM (Pulse Width Modulation), which results in new colors.

The module has four pins, three for red, green and blue light, and the fourth is for the ground. Each color pin on the module have one resistor in series with an LED pin, which is used for current flow limitation, or for LED protection.

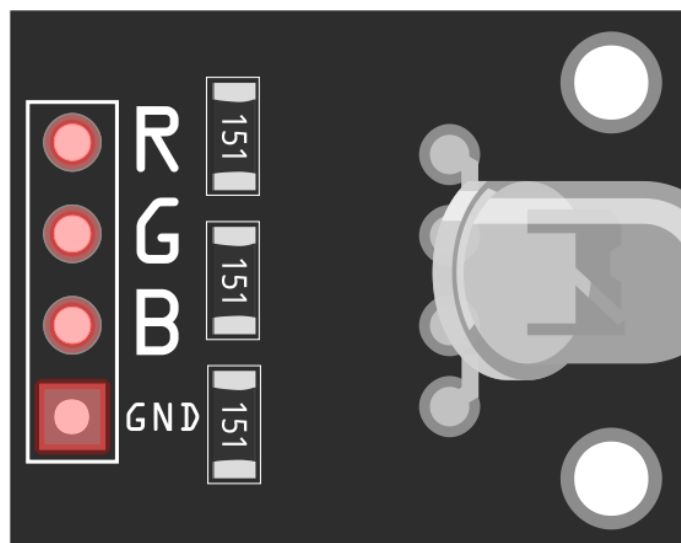
Specifications

Operating voltage range	from 3.3V to 5V DC
Operating temperature	-25°C to 80°C [-13°F to 176°F]
Dimensions	19x15mm [0.73x0.6in]

The pinout

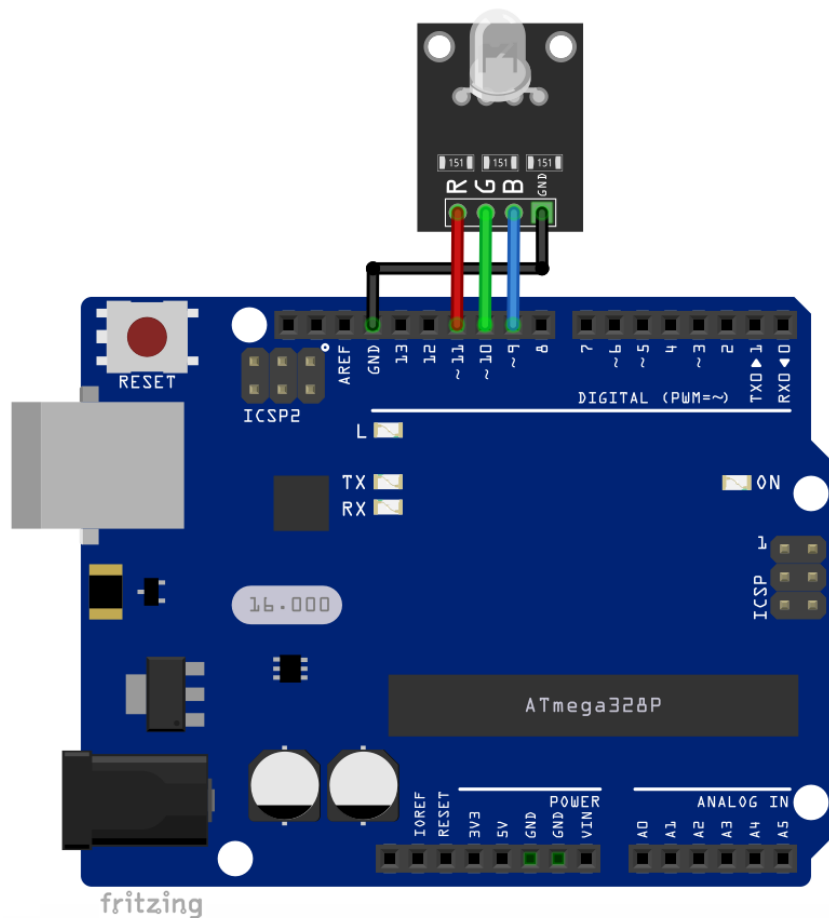
The KY-016 RGB LED module has four pins. The pinout diagram is shown on the following image:

Red part - R
Green part - G
Blue part - B
Ground - GND



Connecting the module with Uno

Connect the KY-016 module with the Uno as shown on the following connection diagram:



KY-016 pin	Uno pin	Wire color
R	D11	Red wire
G	D10	Green wire
B	D9	Blue wire
GND	GND	Black wire

Az-Delivery

Sketch example

```
#define RED_PIN 11
#define GREEN_PIN 10
#define BLUE_PIN 9

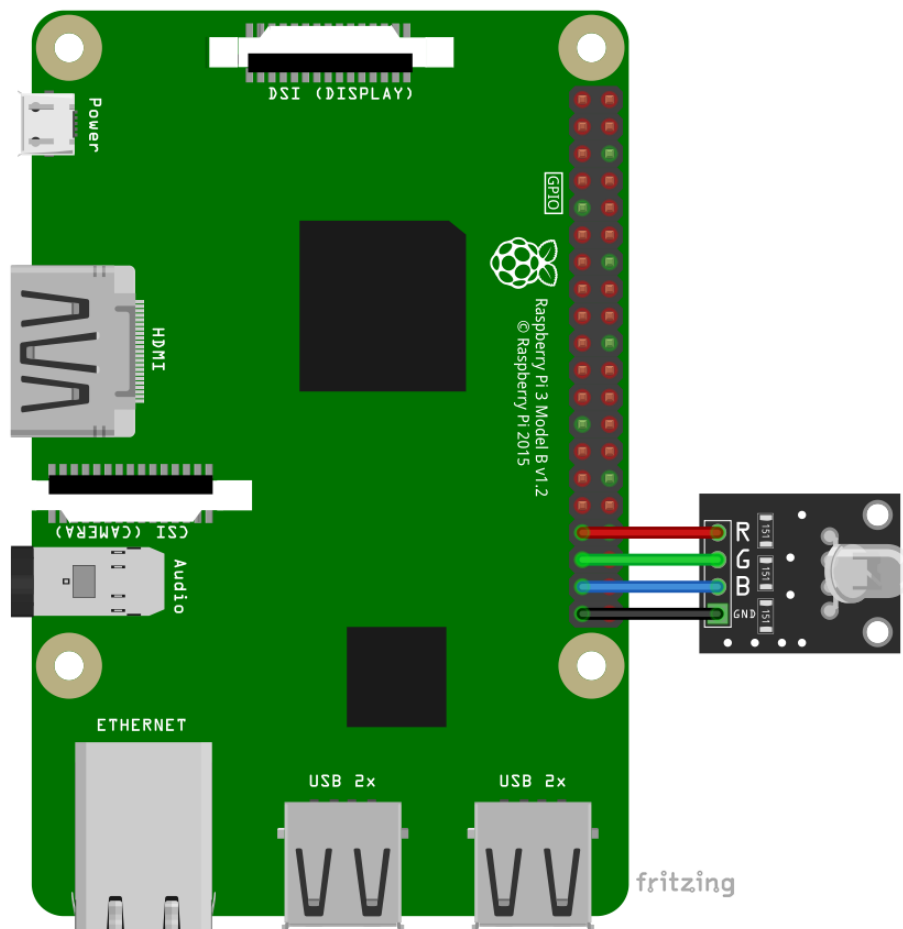
void setup() {
  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
}

void loop() {
  // purple color
  analogWrite(RED_PIN, 250);
  analogWrite(GREEN_PIN, 0);
  analogWrite(BLUE_PIN, 250);
  delay(1000);

  // orange color
  analogWrite(RED_PIN, 255);
  analogWrite(GREEN_PIN, 100);
  analogWrite(BLUE_PIN, 0);
  delay(1000);
}
```

Connecting the module with Raspberry Pi

Connect the KY-016 module with the Raspberry Pi as shown on the following connection diagram:



KY-016 pin	Raspberry Pi pin	Physical pin	Wire color
R	GPIO13	33	Red wire
G	GPIO19	35	Green wire
B	GPIO26	37	Blue wire
GND	GND	39	Black wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
Red_PIN = 13
Green_PIN = 19
Blue_PIN = 26
GPIO.setup(Red_PIN, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Green_PIN, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Blue_PIN, GPIO.OUT, initial=GPIO.LOW)
red = GPIO.PWM(Red_PIN, 100) # PWM, frequency 100Hz
green = GPIO.PWM(Green_PIN, 100) # PWM, frequency 100Hz
blue = GPIO.PWM(Blue_PIN, 100) # PWM, frequency 100Hz
red.start(1)
green.start(1)
blue.start(1)
print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        red.ChangeDutyCycle(90) # Purple color
        green.ChangeDutyCycle(1)
        blue.ChangeDutyCycle(90)
        sleep(1) # Wait for a second
        red.ChangeDutyCycle(99) # Orange color
        green.ChangeDutyCycle(10)
        blue.ChangeDutyCycle(1)
        sleep(1) # Wait for a second

except KeyboardInterrupt:
    print('\nScript end!\n')
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *RGBLED.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

```
python3 RGBLED.py
```

and watch the color changing magic.

When starting the PWM, with the following line of code:

```
red.start(1)
```

use any number different than 0 , because if the number 0 is used, that disables PWM on the selected GPIO pin.

Also, when using the following line of code:

```
red.ChangeDutyCycle(1)
```

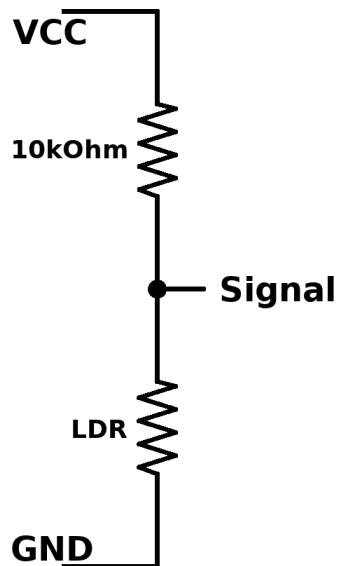
to change the duty cycle of the PWM signal, do not use number 0 , because of the same reason as for the *red.start(1)*.

To stop the script press *CTRL + C* on the keyboard.

KY-018 Photoresistor LDR Module

Introduction

The KY-018 Photoresistor LDR module consists of a photoresistor (or Light Dependent Resistor - LDR) and a $10k\Omega$ resistor. These two resistors create voltage divider, as shown on the following image:



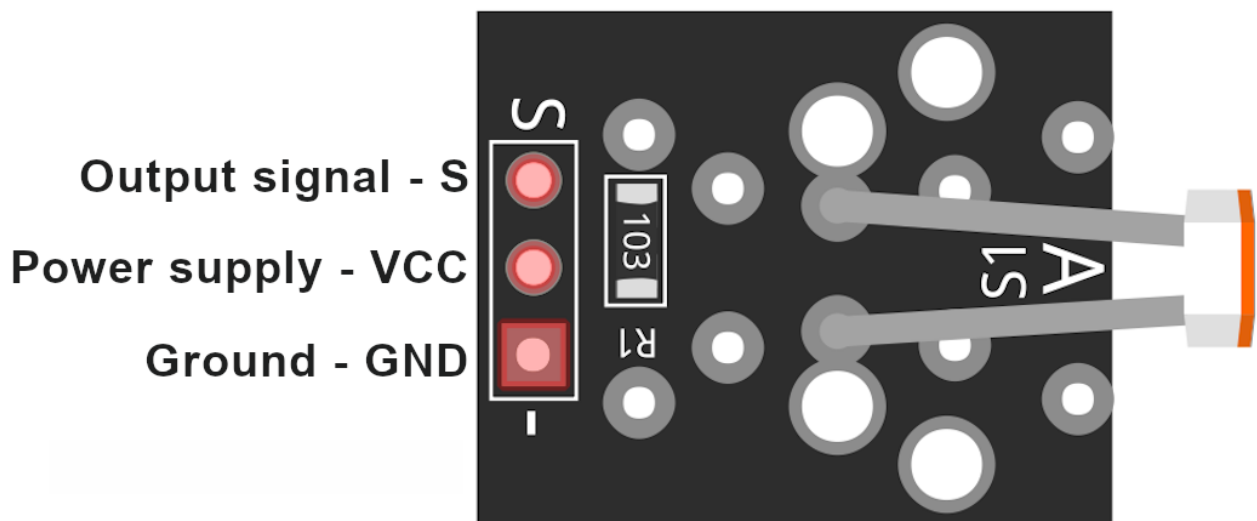
The resistance of a photoresistor decreases in the presence of the light and increase in the absence of it. The output is an analog voltage that is in the opposite relation to the intensity of a light.

Specifications

Operating voltage range	from 3.3V to 5V DC
Operating temperature	-25°C to 80°C [-13°F to 176°F]
Dimensions	19x15mm [0.73x0.6in]

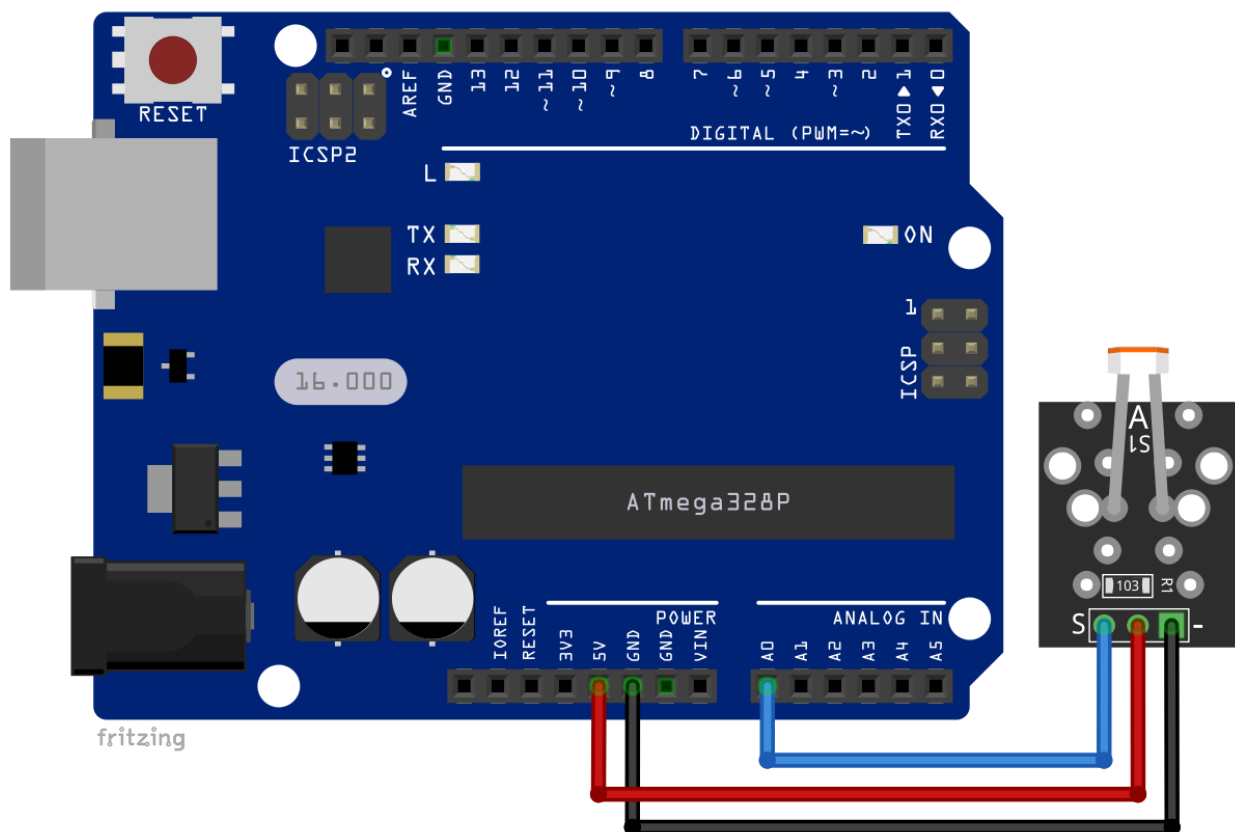
The pinout

The KY-018 photoresistor LDR module has three pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-018 module with the Uno as shown on the following connection diagram:



KY-018 pin	Uno pin	Wire color
S	A0	Blue wire
Middle pin (VCC)	5V	Red wire
- (GND)	GND	Black wire

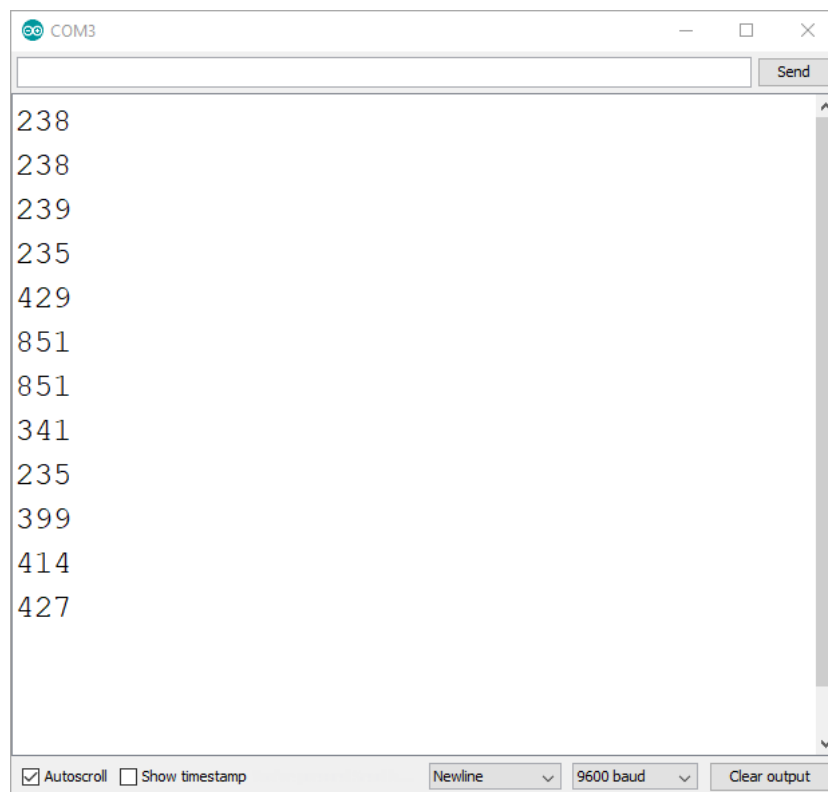
Az-Delivery

Sketch example

The sketch is a built-in sketch from Arduino IDE called *AnalogReadSerial*:

```
void setup() {  
  Serial.begin(9600);  
}  
void loop() {  
  Serial.println(analogRead(0));  
  delay(1000); // delay for better readability  
}
```

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



Az-Delivery

Connecting the module with Raspberry Pi

Connect the KY-018 module with the Uno as shown on the connection diagram from the chapter *Connecting the module with Uno*, and then connect the Uno with the Raspberry Pi via USB cable. Next, upload *nanpy* firmware to the Uno (as shown in the chapter *Uno as ADC*), and use the following code to read the KY-018 module:

```
from nanpy import (ArduinoApi, SerialManager)
from time import sleep
try:
    connection1 = SerialManager()
    a = ArduinoApi(connection=connection1)
except:
    print('Failed to connect to the Arduino')

value = 0
print('[Press CTRL + C to end the script!}')
try:
    while True:
        value = a.analogRead(0)
        print('Output: {}'.format(value))
        sleep(0.5)

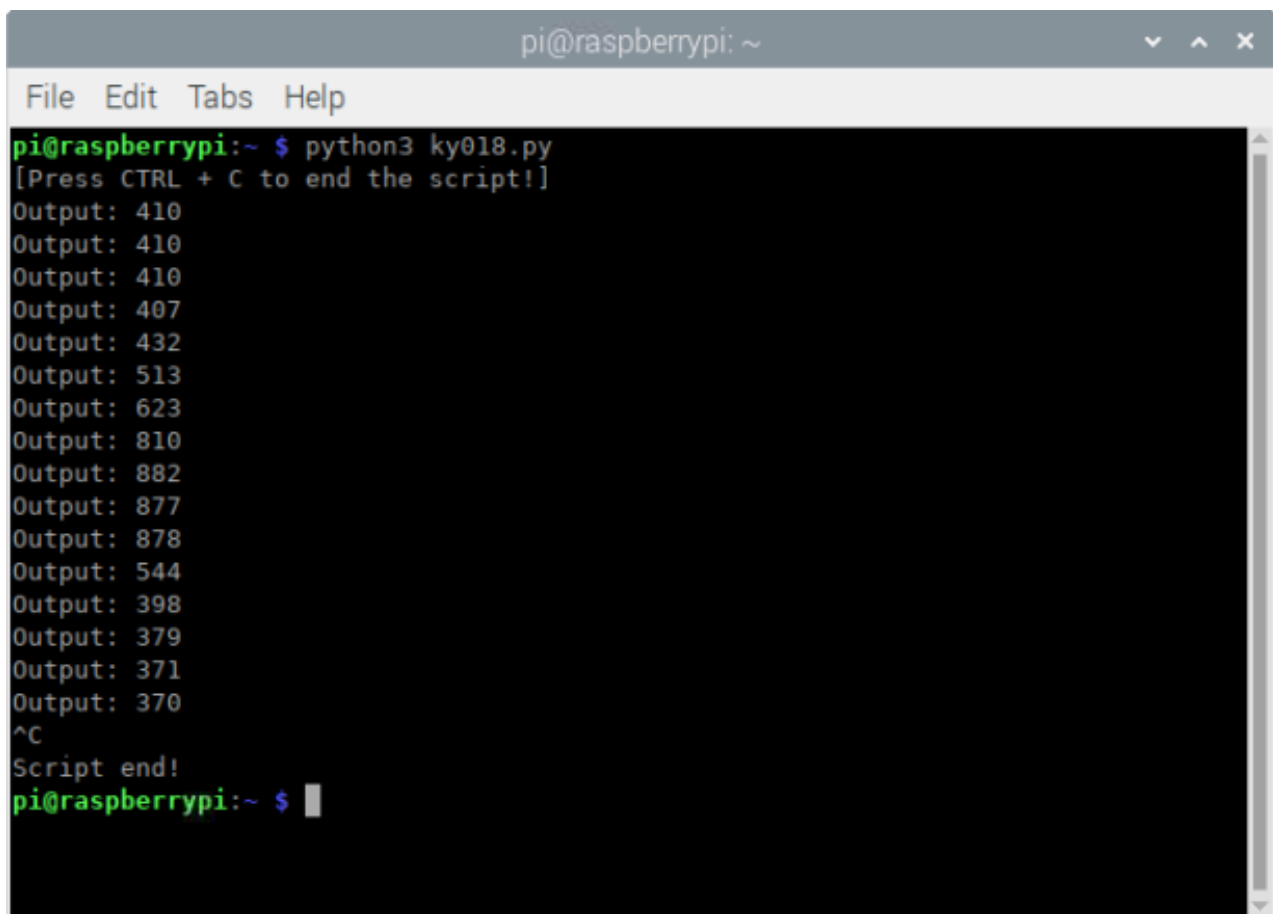
except KeyboardInterrupt:
    print('\nScript end!')
```

Az-Delivery

Save the script by the name *ky018.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 ky018.py

The result should look like the output on the image below:



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ python3 ky018.py  
[Press CTRL + C to end the script!]  
Output: 410  
Output: 410  
Output: 410  
Output: 407  
Output: 432  
Output: 513  
Output: 623  
Output: 810  
Output: 882  
Output: 877  
Output: 878  
Output: 544  
Output: 398  
Output: 379  
Output: 371  
Output: 370  
^C  
Script end!  
pi@raspberrypi:~ $
```

To stop the script press *CTRL + C* on the keyboard.

KY-019 Relay Module

Introduction

The KY-019 relay module is used to control AC circuits, switching them *ON* or *OFF*. The relay is one of the most important control elements. It is an electrical switch that responds to a signal received from the microcontroller (like in Arduino or Raspberry Pi). Relays are widely used in the remote control, communications, mechatronics devices, power electronic devices, etc. They can also be used to separate power voltage/current electronics (like AC or DC motors, or any AC device, etc.) from low voltage/current electronics (like microcontrollers, sensors, etc.)

The KY-019 module consists of an LED, a resistor, an NPN transistor, a rectifier diode and a 5V *DC* relay capable of handling up to 10A 250V *AC*.

On the DC side of the board, there are three pins, one for the signal, one for the power supply (*VCC*) and one for the ground (*GND*). On the AC side there are three contacts Normally Closed - *NC* pin, Common pin and Normally Open - *NO* pin.

Specifications

TTL control signal	from 3.3V to 5V DC
Max. relay AC	10A 250V
Max. relay DC	10A 30V
Contact type	Open (NO), Closed (NC)
Dimensions	27x34mm [1.1x1.4in]



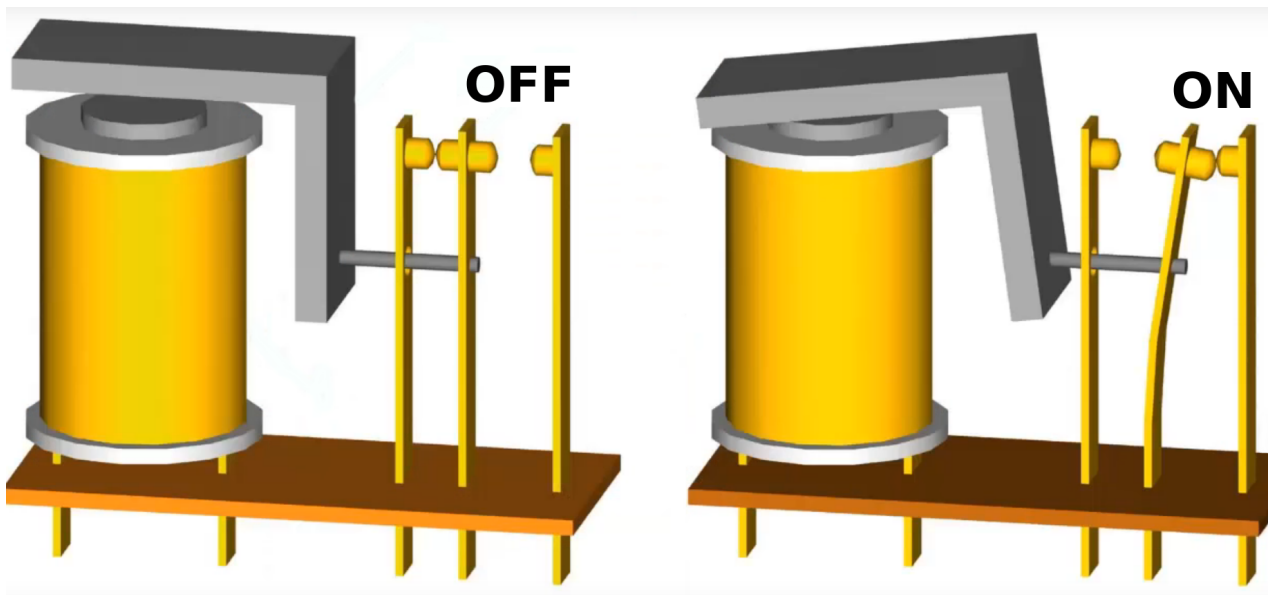
SAFETY WARNING!

When doing projects that are connected to mains voltage, misuse may lead to serious electrical shock!

For the sake of your own safety, be 100% sure what you are doing! Otherwise, ask someone who knows!

Working principle of the relay

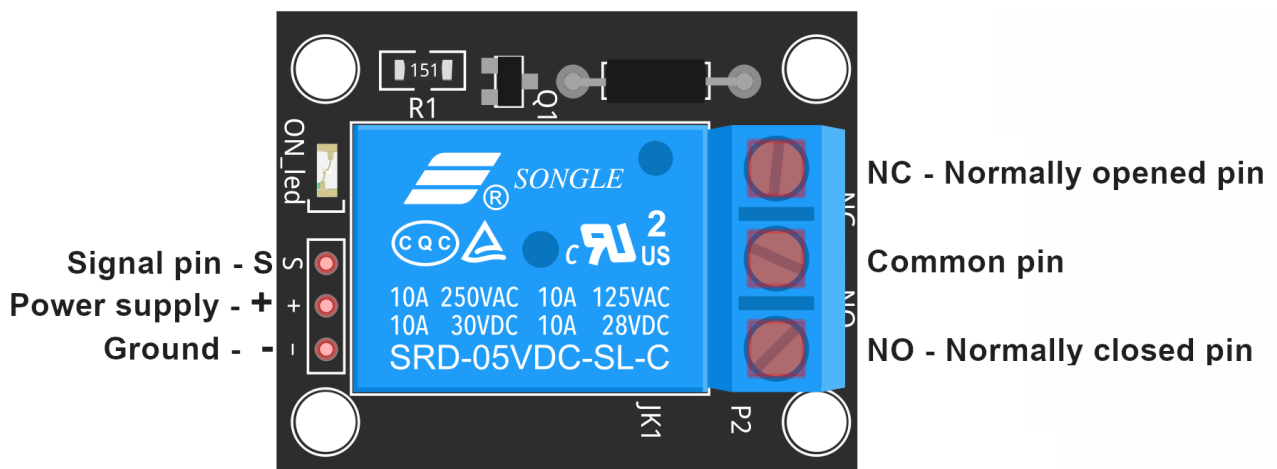
Inside the relay there is one mechanical switch (three lines with middle that moves) which is controlled by the second element, the electromagnet (big cylindrical object on the image), as shown on the following image:



In the rest state, the switch is in the *OFF* state, *NC* pin is connected with a common pin, and *NO* is disconnected. When the power is connected to the electromagnet (using a microcontroller via transistor and rectifier diode), this moves the switch to the active state, thus connecting the common pin to *NO* pin.

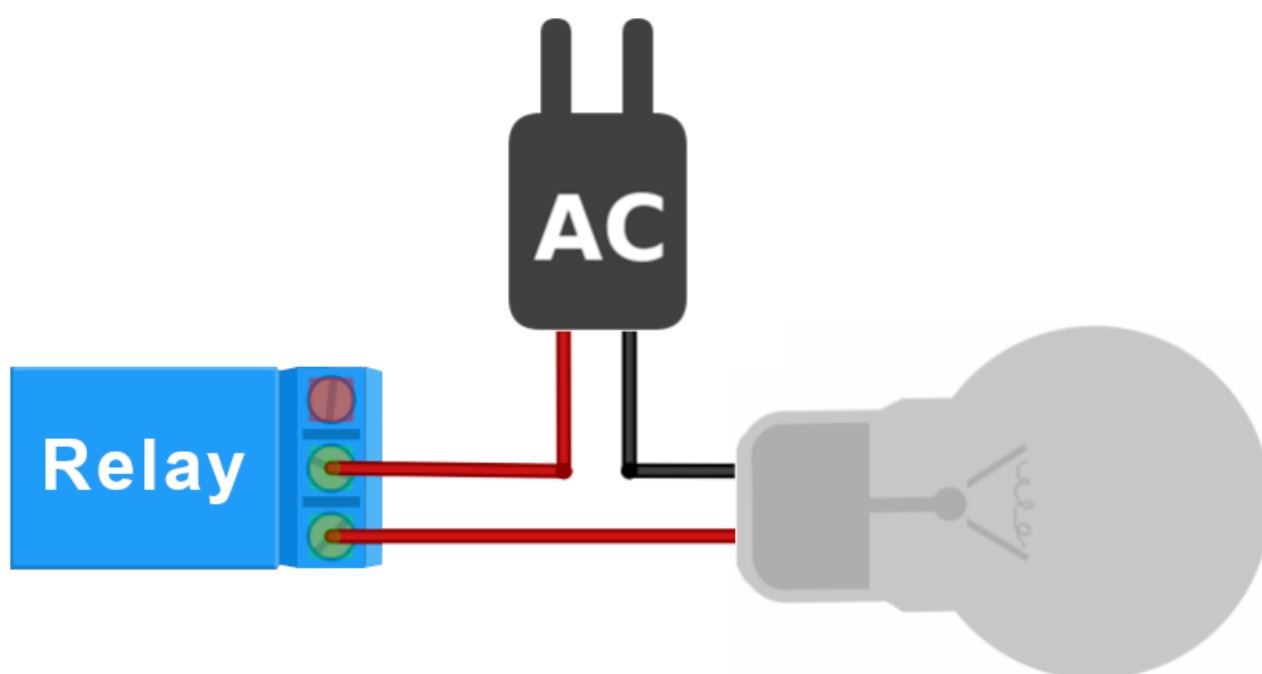
The pinout

The KY-019 relay module has six pins. The pinout diagram is shown on the following image:



Connecting the AC side of the relay

For this purpose we will be using one light bulb, a two wire cable and an AC power plug. The AC part of the connection diagram is the same for all 16 relays on-board 16 relays module. Connect the relay with light bulb and power plug as shown on the connection diagram below:

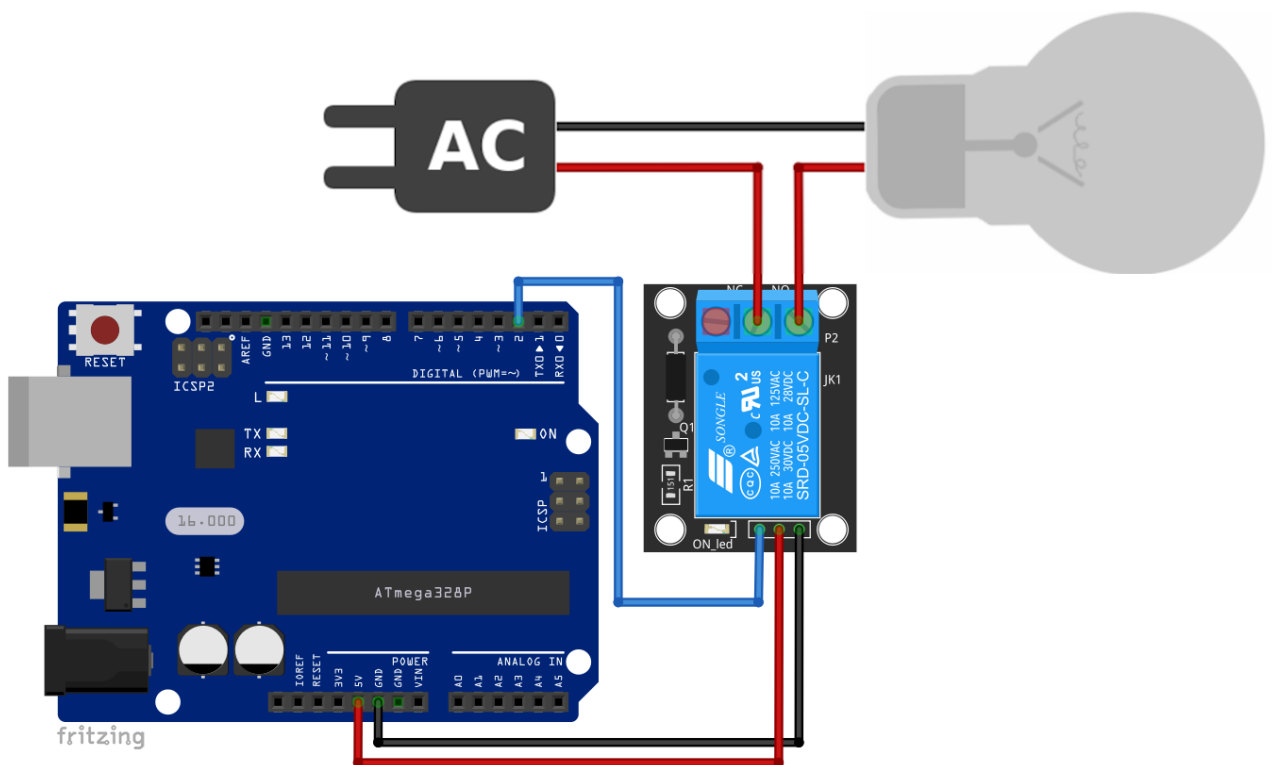


Module pin	Power plug, light bulb	Wire color
Common pin	One side of power plug	Red wire
Normally opened pin	One side of bulb	Red wire

Light bulb	Power plug	
The other side of bulb	The other side of power plug	Black wire

Connecting the module with Uno

Connect the KY-019 module with the Uno as shown on the following connection diagram:



KY-019 pin	Uno pin	Wire color
S	D2	Blue wire
- (GND)	GND	Black wire
+ (VCC)	5V	Red wire

Sketch example

```
#define RELAY_PIN 2
void setup() {
  pinMode(RELAY_PIN, OUTPUT);
}
void loop() {
  digitalWrite(RELAY_PIN, HIGH);
  delay(1000);

  digitalWrite(RELAY_PIN, LOW);
  delay(1000);
}
```

When the sketch is uploaded to the Uno, the clicks from the relay could be heard. When the relay change state from active to rest and vice versa, the clicks of the switch can be heard.

The light bulb should blink every second.

We can change NO/NC pin states by these lines of code:

`digitalWrite(2, HIGH);` – **NC** pin is not connected with the c. pin

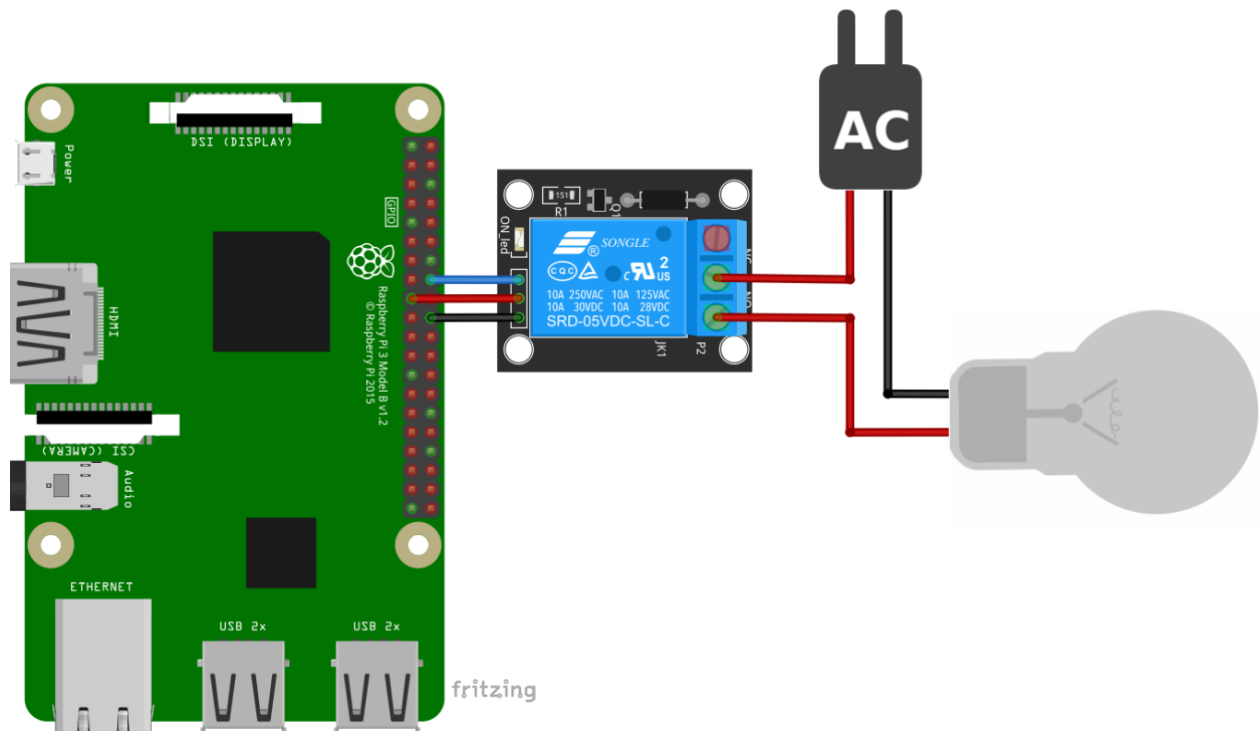
NO pin is connected with the c. pin

`digitalWrite(2, LOW);` – **NC** pin is connected with the c. pin

NO pin is not connected with the c. pin

Connecting the module with Raspberry Pi

Connect the KY-019 module with the Raspberry Pi as shown on the following connection diagram:



KY-019 pin	Raspberry Pi pin	Physical pin	Wire color
S	GPIO23	16	Blue wire
+ (VCC)	3V3	17	Red wire
- (GND)	GND	20	Black wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep

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

Relay_PIN = 23
GPIO.setup(Relay_PIN, GPIO.OUT)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        GPIO.output(Relay_PIN, GPIO.HIGH)
        print('Normally opened pin is HIGH')
        sleep(1)
        GPIO.output(Relay_PIN, GPIO.LOW)
        print('Normally opened pin is LOW')
        sleep(1)

except KeyboardInterrupt:
    print('\nScript end!')

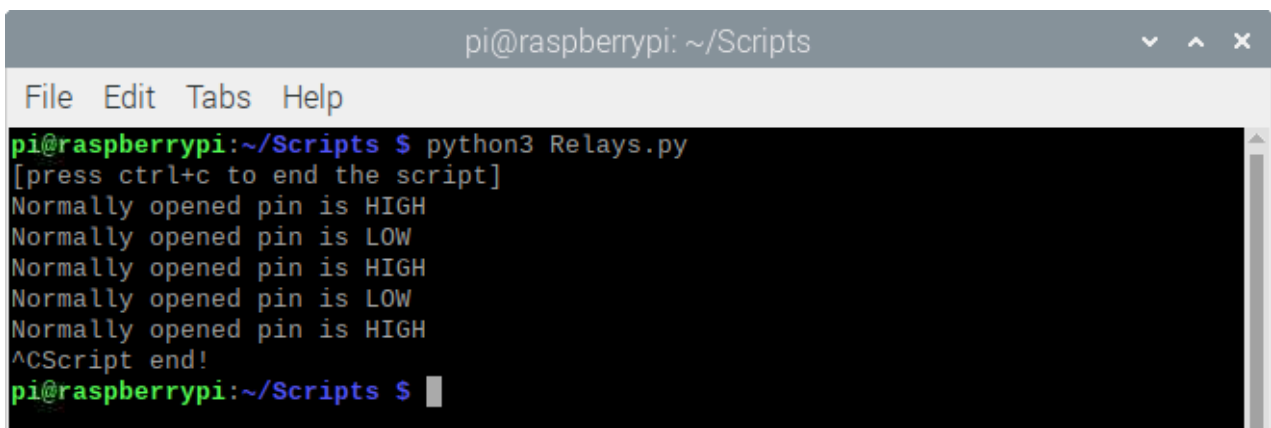
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *Relays.py*. To run the script open the terminal in the directory where you saved the script and run the following command:

python3 Relays.py

The result should look like the output on the following image:

A screenshot of a terminal window titled 'pi@raspberrypi: ~/Scripts'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal shows the command 'python3 Relays.py' being executed. The output of the script is as follows:

```
pi@raspberrypi:~/Scripts $ python3 Relays.py
[press ctrl+c to end the script]
Normally opened pin is HIGH
Normally opened pin is LOW
Normally opened pin is HIGH
Normally opened pin is LOW
Normally opened pin is HIGH
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press CTRL + C on the keyboard.

KY-020 Tilt Sensor Module

Introduction

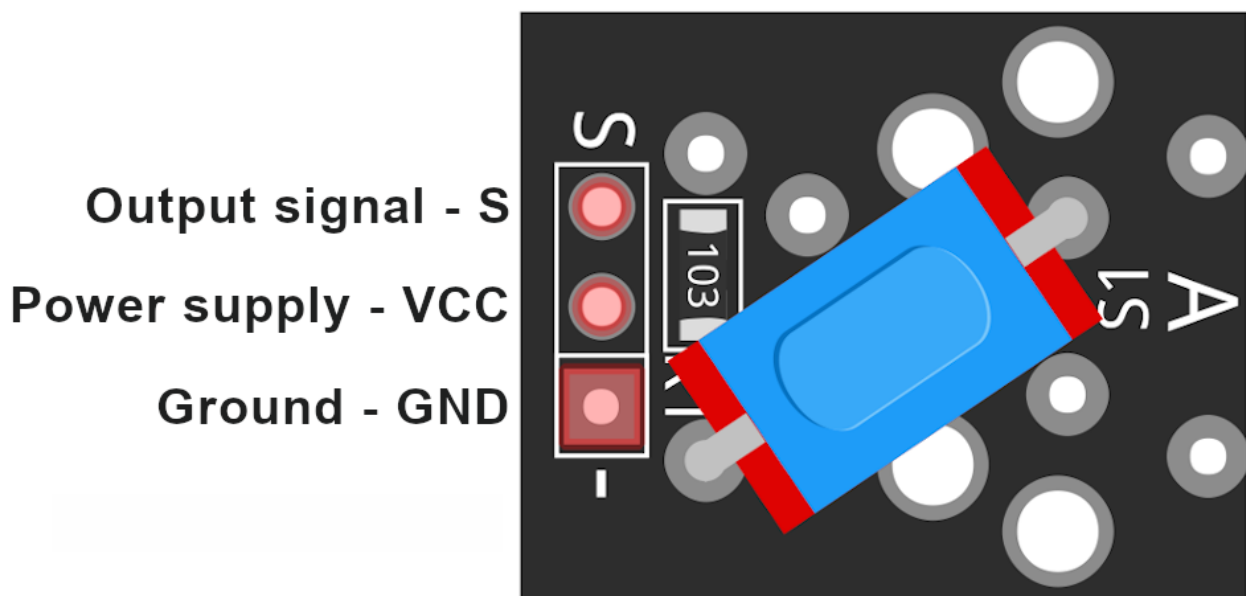
The KY-020 tilt sensor module is a very simple switch that gets activated by tilting the module board. It is a metallic ball tilt switch that opens or closes a circuit when a metallic ball moves and connects metal electrodes and opens or closes the circuit. This kind of switches has a non-eroding switch contacts. The switch is in one state (open or closed) when tilted in the one direction with respect to horizontal, and in the other state when tilted in the other direction.

Specification

Operating Voltage	from 3.3V to 5V DC
Output	digital, with high sensitivity
Dimensions	15x19mm [0.6x0.75in]

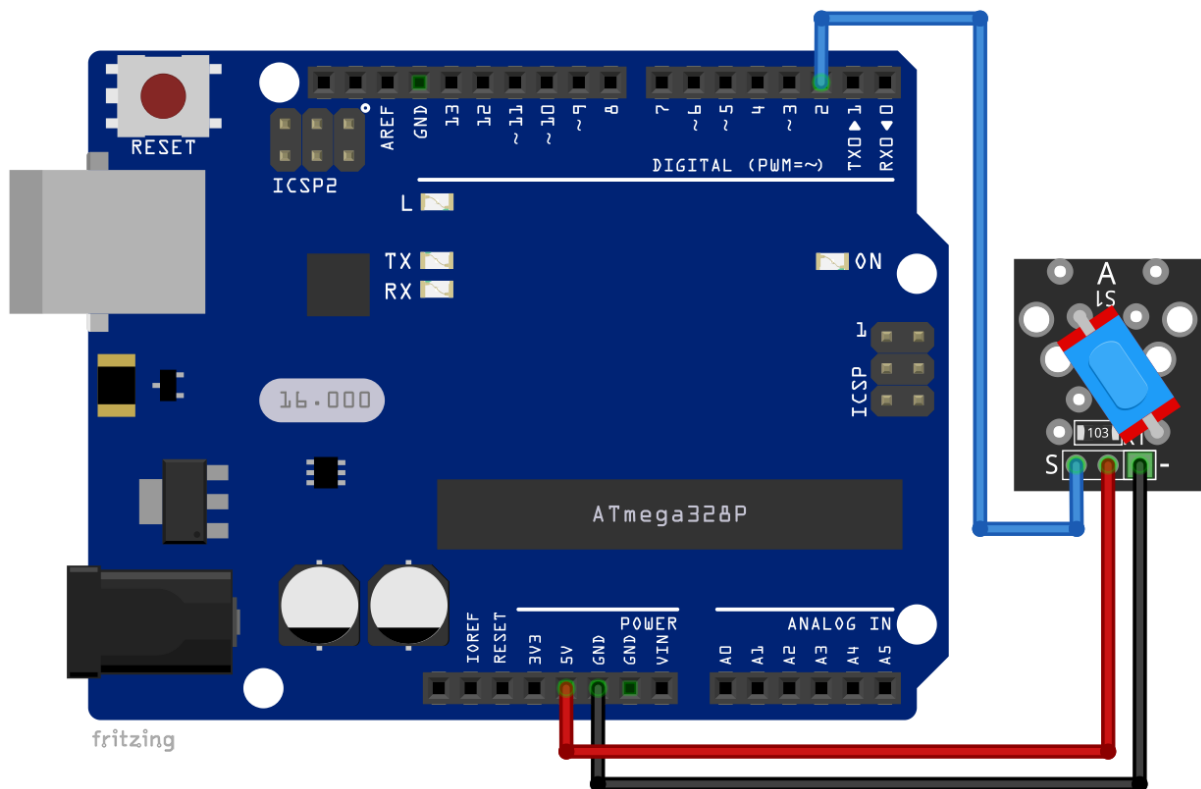
The pinout

The KY-020 tilt sensor module has three pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-020 module with the Uno as shown on the following connection diagram:



KY-020 pin	Uno pin	Wire color
S	D2	Blue wire
+ (VCC)	5V	Red wire
- (GND)	GND	Black wire

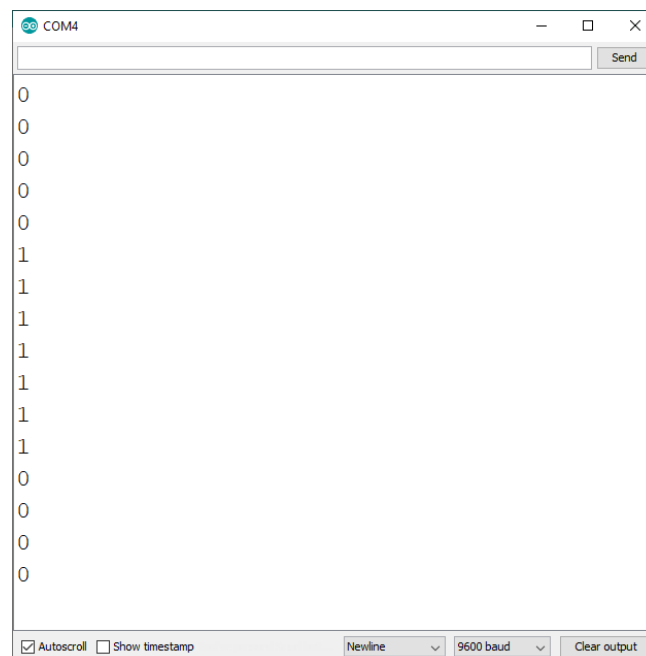
Az-Delivery

Sketch example

The sketch is the built-in sketch from the Arduino IDE called *DigitalReadSerial*:

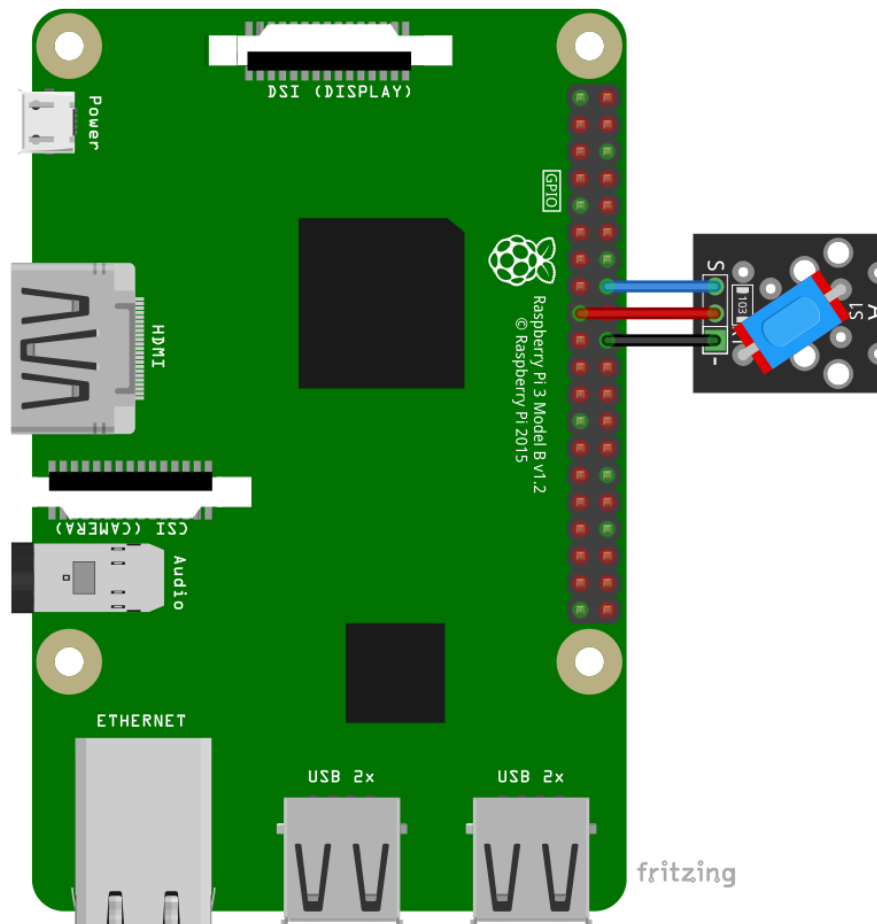
```
void setup() {  
  Serial.begin(9600);  
  pinMode(2, INPUT);  
}  
  
void loop() {  
  Serial.println(digitalRead(2));  
  delay(1000);  
}
```

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



Connecting the module with Raspberry Pi

Connect the KY-020 module with the Raspberry Pi as shown on the following connection diagram:



KY-020 pin	Raspberry Pi pin	Physical pin	Wire color
S	GPIO23	16	Blue wire
+ (VCC)	3V3	17	Red wire
- (GND)	GND	20	Black wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep

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

TILT_PIN = 23
GPIO.setup(TILT_PIN, GPIO.IN)

def fl(channel):
    if GPIO.input(channel):
        print('Tilted on the left')
    else:
        print('Tilted on the right')

GPIO.add_event_detect(23, GPIO.BOTH, callback=fl, bouncetime=200)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        sleep(0.0001)

except KeyboardInterrupt:
    print('\nScript end!\n')

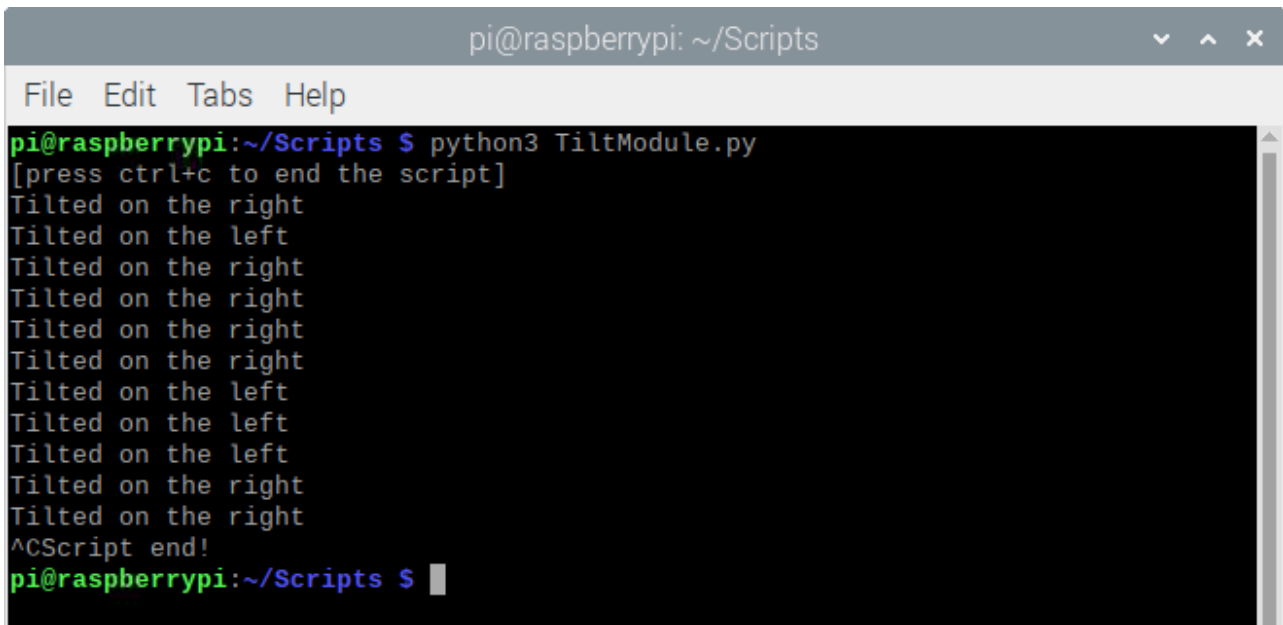
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *TiltModule.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 TiltModule.py

The result should look like the output on the following image:

A screenshot of a terminal window titled 'pi@raspberrypi: ~/Scripts'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal shows the command 'python3 TiltModule.py' being executed. The output consists of a series of lines: '[press ctrl+c to end the script]', 'Tilted on the right', 'Tilted on the left', 'Tilted on the right', 'Tilted on the right', 'Tilted on the right', 'Tilted on the right', 'Tilted on the left', 'Tilted on the left', 'Tilted on the left', 'Tilted on the right', 'Tilted on the right', and '^CScript end!'. The prompt 'pi@raspberrypi:~/Scripts \$' is visible at the bottom with a cursor.

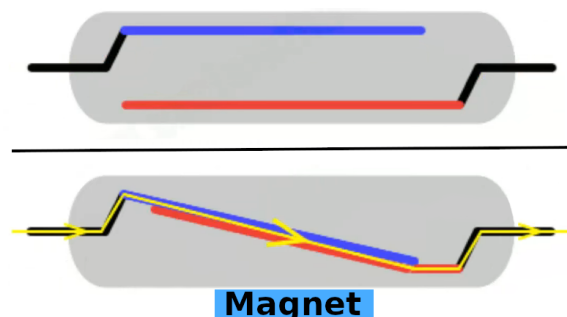
To stop the script press *CTRL* + *C* on the keyboard.

KY-021 Mini Reed Sensor Module

Introduction

The KY-021 mini reed sensor module is a reed switch, a magnetic switch that changes its state when exposed to a magnetic field. The module is normally opened, which means, when there is no magnetic field near the module, contacts in the switch are opened. When there is a magnetic field near the module, the contacts in the switch close, enabling current flow through a switch.

The contacts in the reed module are magnetic, one is the north magnetic pole and the other is the south magnetic pole. When the magnet is near the module, one contact is repelled and the other is attracted, which closes the switch enabling current flow through it.



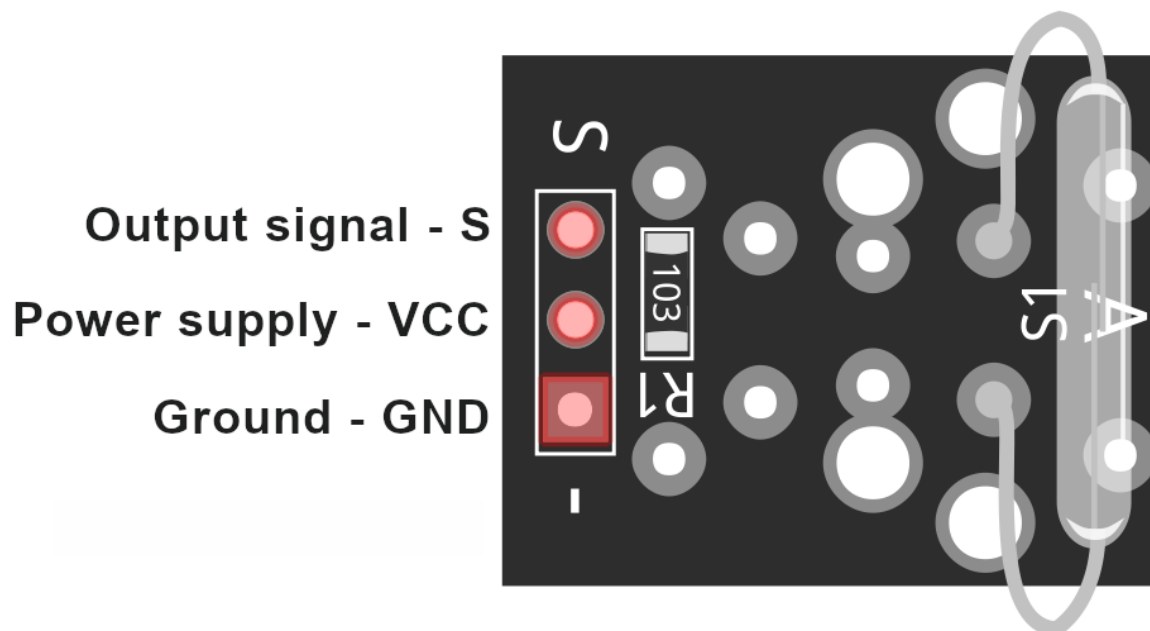
Because it is a simple switch, the KY-021 module can be operated with any voltage. The output of the KY-021 module is digital, which means that it can be used with Uno and Raspberry Pi.

Specification

Operating Voltage	from 3.3V to 5V DC
Output	digital, with high sensitivity
Dimensions	15x19mm [0.6x0.75in]

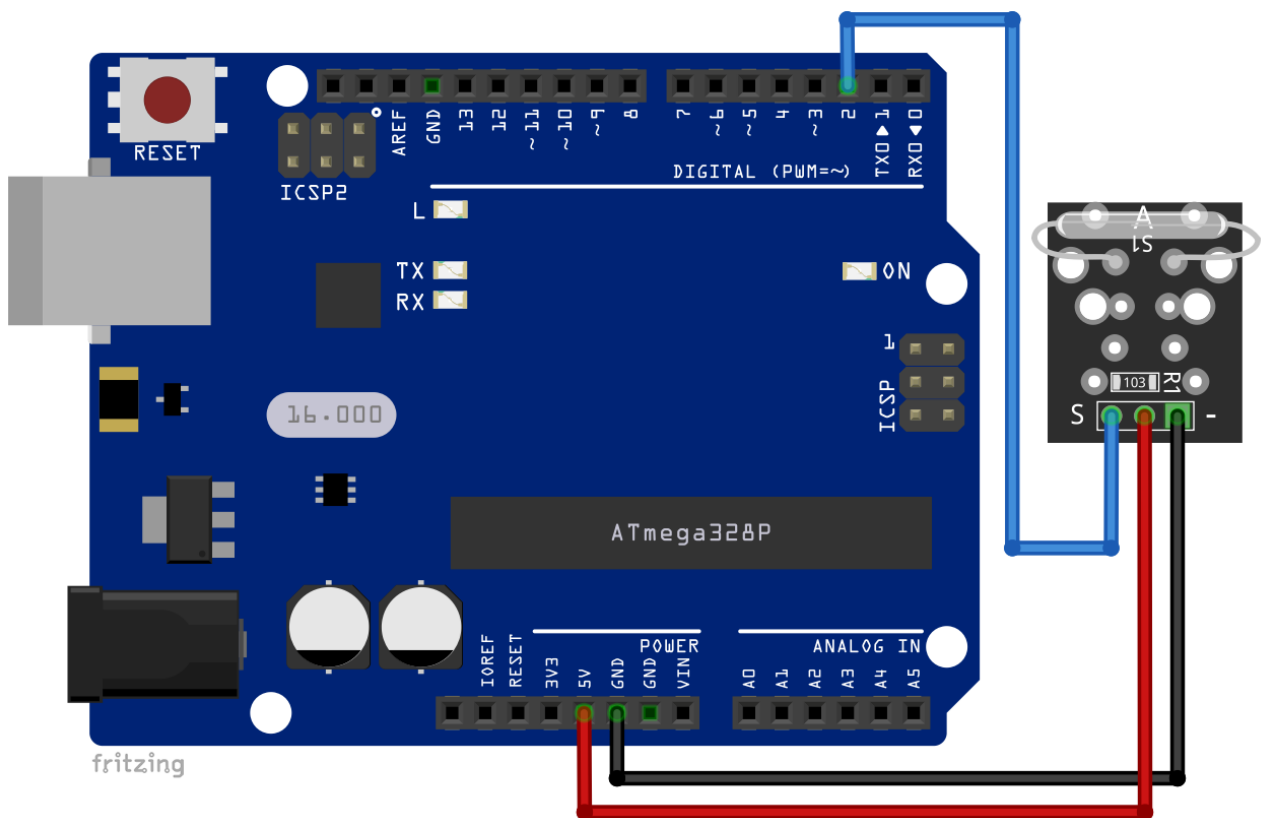
The pinout

The KY-021 mini reed sensor module has three pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-021 module with the Uno as shown on the following connection diagram:



KY-021 pin	Uno pin	Wire color
S	D2	Blue wire
- (GND)	GND	Black wire
Middle pin (VCC)	5V	Red wire

Az-Delivery

Sketch example

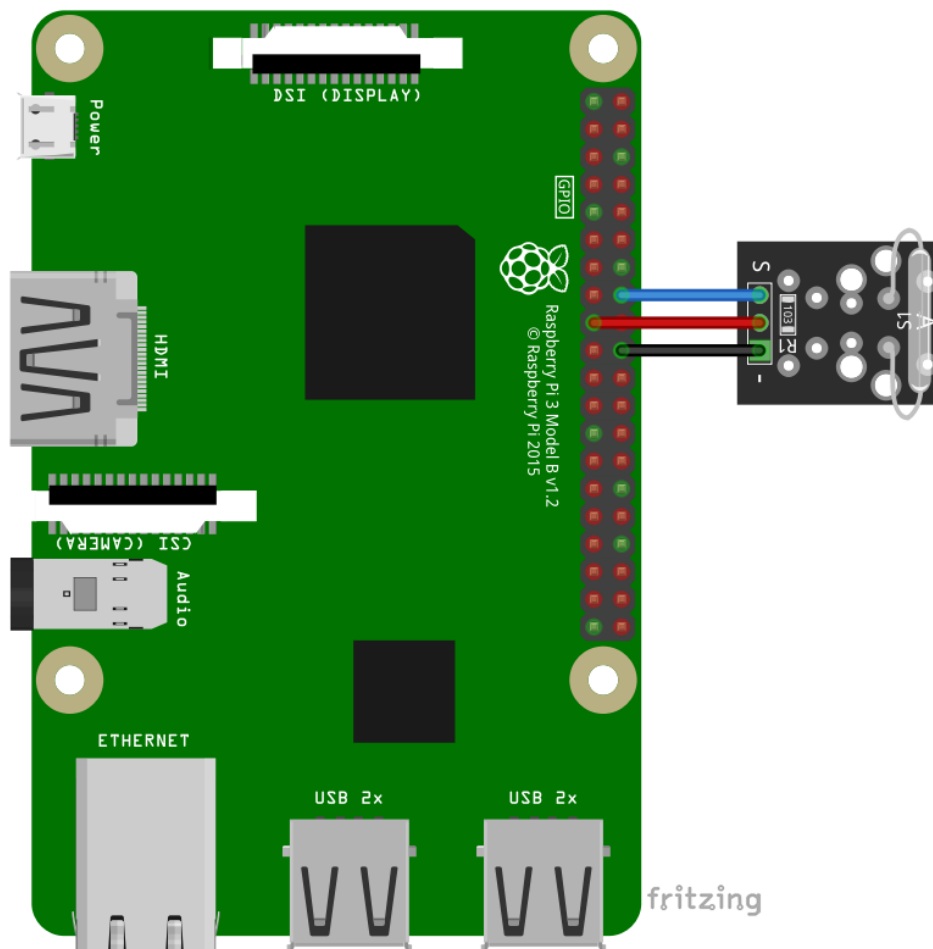
```
#define REED_PIN 2

void setup() {
  pinMode(REED_PIN, INPUT);
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  uint8_t reed = digitalRead(REED_PIN);
  if (reed == 1) {
    digitalWrite(LED_BUILTIN, LOW);
  }
  else {
    digitalWrite(LED_BUILTIN, HIGH);
  }
  delay(500);
}
```

Connecting the module with Raspberry Pi

Connect the KY-021 module with the Raspberry Pi as shown on the following connection diagram:



KY-021 pin	Raspberry Pi pin	Physical pin	Wire color
S	GPIO23	16	Blue wire
Middle pin (VCC)	3V3	17	Red wire
- (GND)	GND	20	Black wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep

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

Reed_PIN = 23
GPIO.setup(Reed_PIN, GPIO.IN)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        if not GPIO.input(Reed_PIN):
            print('Magnet detected')
        else:
            print('No magnet')
        sleep(0.5)

except KeyboardInterrupt:
    print('\nScript end!')

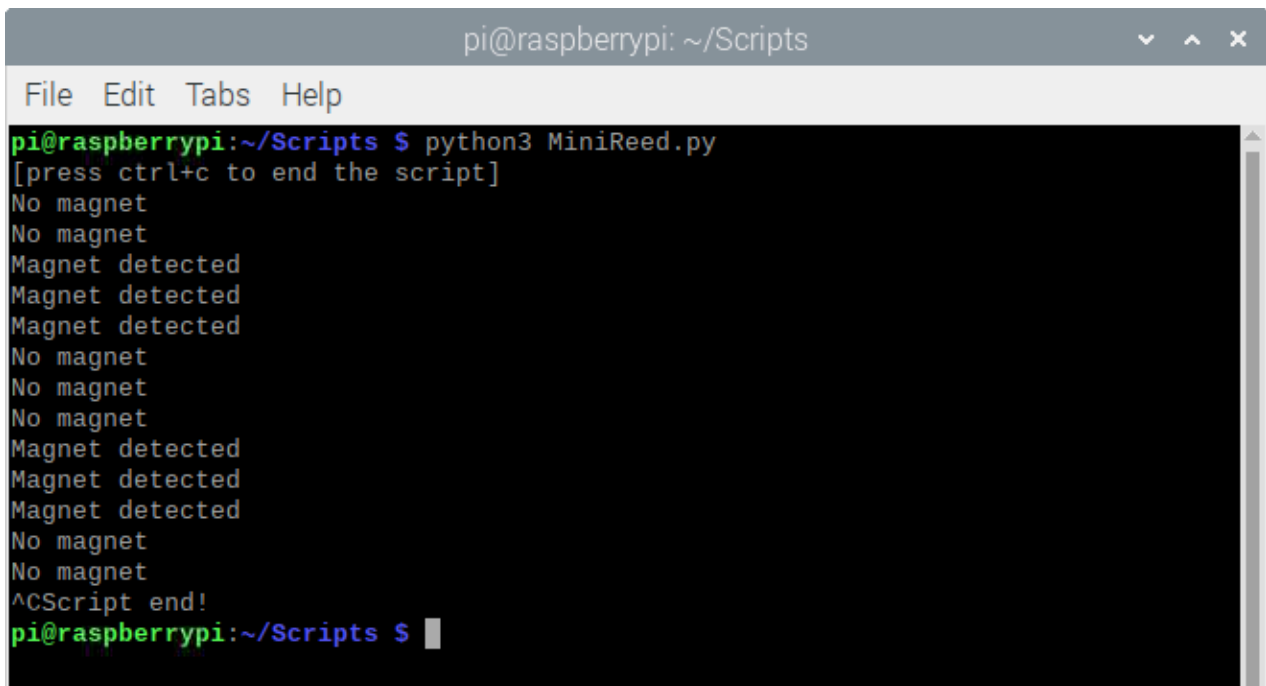
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *MiniReed.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

```
python3 MiniReed.py
```

The result should look like the output on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 MiniReed.py
[press ctrl+c to end the script]
No magnet
No magnet
Magnet detected
Magnet detected
Magnet detected
No magnet
No magnet
No magnet
Magnet detected
Magnet detected
Magnet detected
No magnet
No magnet
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL + C* on the keyboard.

KY-022 Infrared Receiver Sensor Module

Introduction

The KY-022 Infrared receiver sensor module consists of an 1838 infrared receiver, a $1k\Omega$ resistor and a red LED. Infrared receiver reacts to 38kHz infrared light. The infrared receiver is a photodiode and pre-amplifier that converts the infrared light into an electrical signal.

Specifications

Operating voltage range	from 3.3V to 5V DC
Current consumption	1.5mA peak
Chip type	VS1838B
Reception distance	17m
Carrier frequency	38kHz
Infrared wavelength	940nm
Pulse duration	from 400 μ s to 2000 μ s
Dimensions	9x22mm [0.9x0.35in]

Working principle

Infrared light is a light that is emitted by the sun, light bulbs, or anything else that produces heat. That means there is a lot of infrared light noise all around us. To prevent this noise from interfering with infrared signals, a signal modulation technique is developed.

In the infrared signal modulation, an encoder on the infrared remote converts a binary signal into a modulated electrical signal. This electrical signal is sent to the infrared LED. The LED converts the modulated electrical signal into a modulated infrared light signal. The infrared receiver or photodiode then demodulates the infrared light signal and converts it back to electric (binary), before passing the information to a microcontroller. The modulated infrared signal is a series of infrared light pulses, switched *ON* and *OFF* at a high frequency, known as the carrier frequency. The carrier frequency used by most transmitters is *38kHz* because it is rare in nature and it can be distinguished from ambient noise. This way the infrared receiver knows that the *38kHz* signal was sent from the transmitter and not picked up from the surrounding environment.

The receiver diode detects all frequencies of infrared light, but it has a band pass filter and only lets through infrared light at *38kHz* frequency. It then amplifies the modulated signal with a pre-amplifier and converts it to a binary signal, before sending it to a microcontroller. The pattern in which the modulated infrared signal is converted to binary is defined by a transmission protocol.

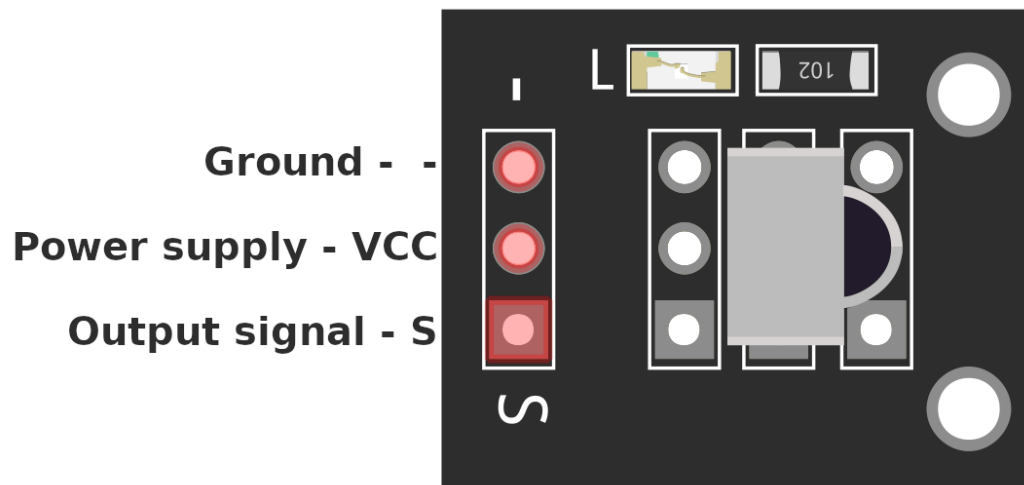
Az-Delivery

There are many infrared transmission protocols: *Sony*, *Matsushita*, *NEC*, and *RC5* are some of the most common protocols. The *NEC* protocol is also the most common type in embedded projects, so it is used in this eBook as an example to show how the receiver converts the modulated infrared signal to a binary one. Logical *1* starts with a $562.5\mu\text{s}$ long *HIGH* pulse at 38kHz infrared light followed by a $1687.5\mu\text{s}$ long *LOW* pulse. Logical *0* is transmitted with a $562.5\mu\text{s}$ long *HIGH* pulse followed by a $562.5\mu\text{s}$ long *LOW* pulse. Other protocols differ only in the duration of the individual *HIGH* and *LOW* pulses.

Each time a button on the remote control is pressed, a unique hexadecimal code is generated. This is the information that is modulated and sent over infrared light to the receiver. To decipher which key is pressed, the receiving microcontroller needs to know which code corresponds to each key on the remote. Different remotes send different codes for the keypresses, so the code generated for each key on the particular remote has to be determined. It can be found in the datasheet (the infrared key codes should be listed in datasheet). There is a simple sketch that reads most of the popular remote controls and display the hexadecimal codes to the Serial Monitor when a key is pressed (later in the text).

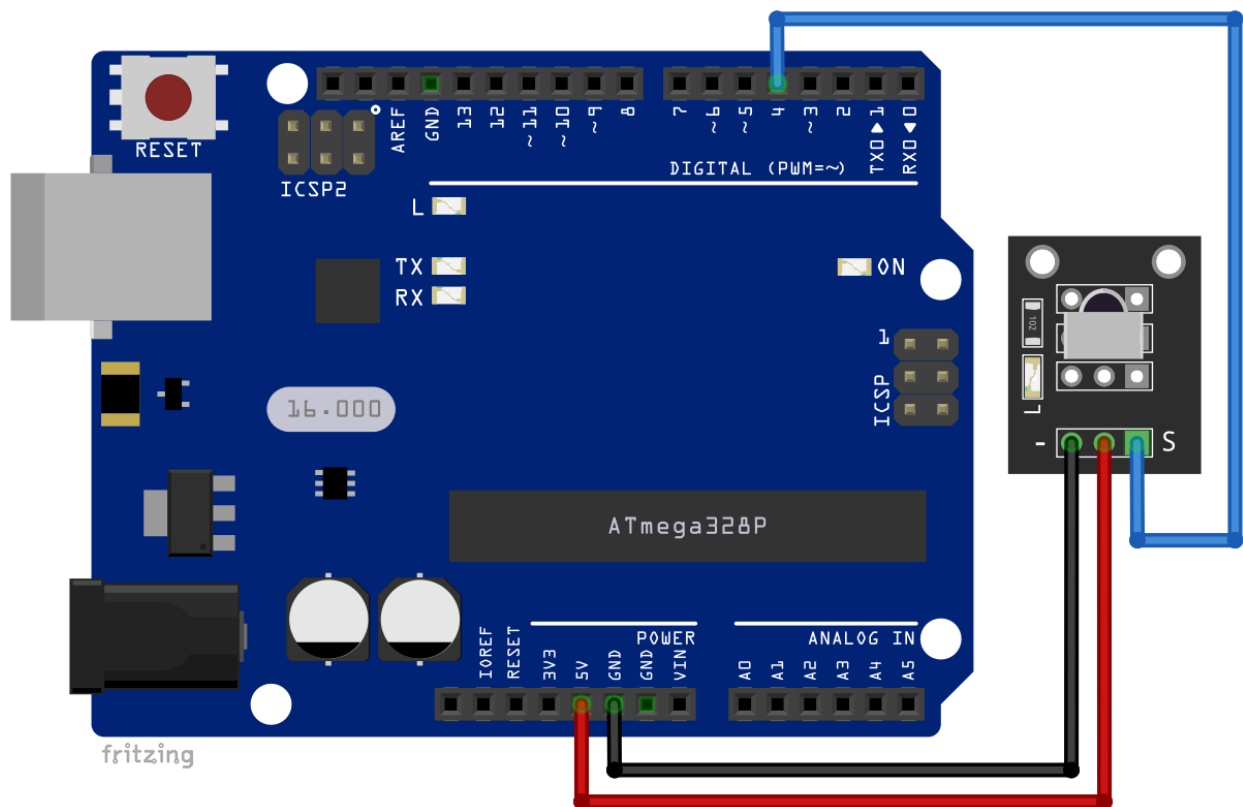
The pinout

The KY-022 infrared receiver module has three pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-022 module with the Uno as shown on the following connection diagram:

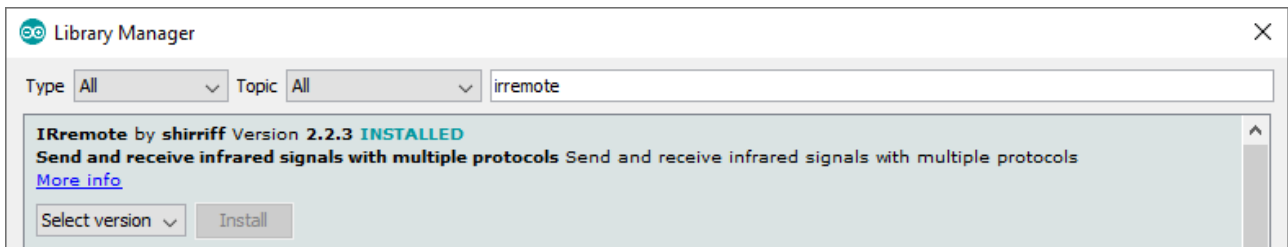


KY-022 pin	Uno pin	Wire color
S	D4	Blue wire
Middle pin (VCC)	5V	Red wire
- (GND)	GND	Black wire

Library for Arduino IDE

To use the KY-022 module with Uno it is recommended to install an external library. Open Arduino IDE and go to: *Tools > Manage Libraries*

When a new window opens, type *IRremote* (two R's) in the search box and install a library called *IRremote* made by *shirriff*, as shown on the following image:



In the following sketch (on the next page) the infrared light that comes from the remote is read and decoded. A message with the remote manufacturer code specification and the hexadecimal code of a specific button that is being pressed is displayed in the Serial Monitor. This way the specific hexadecimal code of the specific key of the TV remote can be read and later used (for example in the eBook for KY-005 module).

Az-Delivery

Sketch for detecting hex code

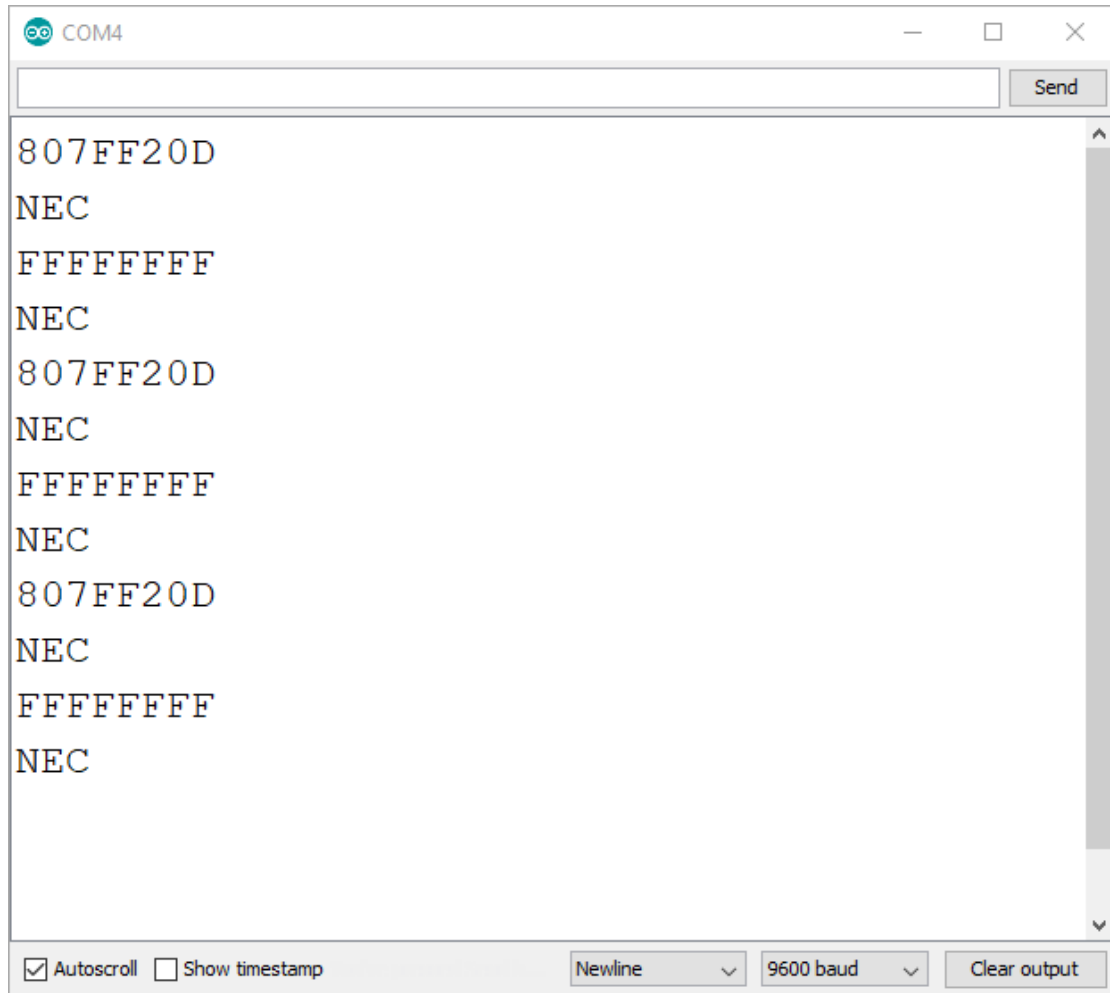
```
#include <IRremote.h>
#define SIGNAL_PIN 4
IRrecv receiver(SIGNAL_PIN);
decode_results output;
void setup(){
    Serial.begin(9600);
    receiver.enableIRIn();
}
void loop() {
    if (receiver.decode(&output)) {
        Serial.println(output.value, HEX);
        switch (output.decode_type) {
            case NEC:
                Serial.println("NEC"); break;
            case SONY:
                Serial.println("SONY"); break;
            case RC5:
                Serial.println("RC5"); break;
            case RC6:
                Serial.println("RC6"); break;
            case DISH:
                Serial.println("DISH"); break;
            case SHARP:
                Serial.println("SHARP"); break;
            case JVC:
                Serial.println("JVC"); break;
            case SANYO:
                Serial.println("SANYO"); break;
```

Az-Delivery

```
// two tabs
    case MITSUBISHI:
        Serial.println("MITSUBISHI"); break;
    case SAMSUNG:
        Serial.println("SAMSUNG"); break;
    case LG:
        Serial.println("LG"); break;
    case WHYENTER:
        Serial.println("WHYENTER"); break;
    case AIWA_RC_T501:
        Serial.println("AIWA_RC_T501"); break;
    case PANASONIC:
        Serial.println("PANASONIC"); break;
    case DENON:
        Serial.println("DENON"); break;
    default:
    case UNKNOWN:
        Serial.println("UNKNOWN"); break;
}
receiver.resume();
}
}
```

Az-Delivery

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



As it can be seen on the image above, the remote with *NEC* code specification is used. Press the specific key several times to get its code hexadecimal number. This number is used in the next sketch example.

Hexadecimal number that is different than number `0xFFFFFFFF` is the code of the specific button. The remote outputs number `0xFFFFFFFF` when the specific button is pressed and held (an indication that the button is held).

Sketch for controlling an LED

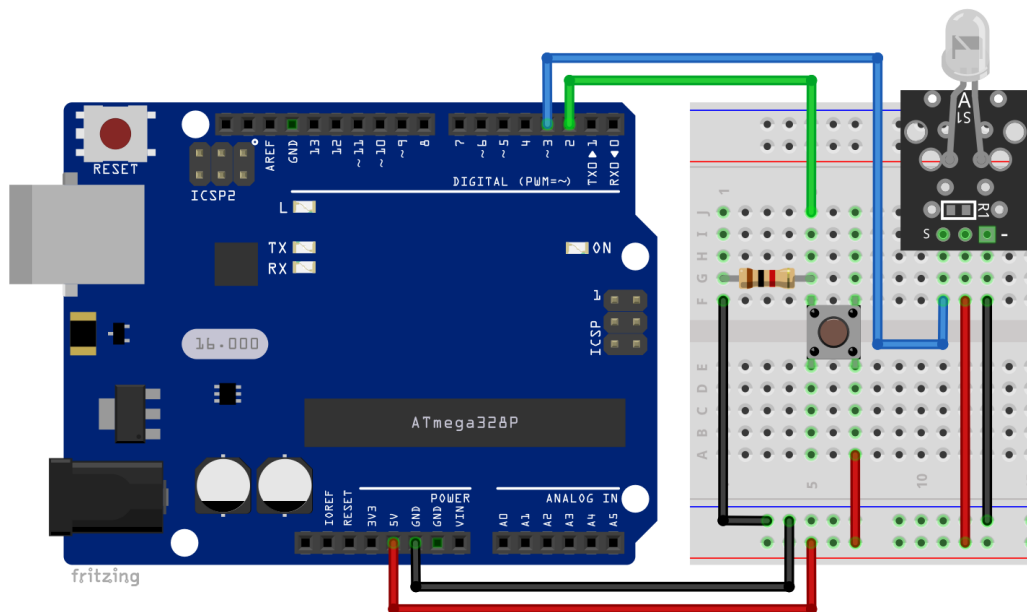
The following sketch is used to toggle the state of the LED (the on-board of Uno) when a specific button is pressed on the remote.

```
#include <IRremote.h>
#define SIGNAL_PIN = 4;
uint8_t toggle_state = 0;
IRrecv receiver(SIGNAL_PIN);
decode_results output;
void setup(){
  receiver.enableIRIn();
  pinMode(LED_BUILTIN, OUTPUT);
}
void loop(){
  if (receiver.decode(&output)){
    switch(output.value){
      case 0x807FF20D: // specific code of some button on the remote
        if(toggle_state == 0){
          digitalWrite(LED_BUILTIN, HIGH); toggle_state = 1;
        }
        else {
          digitalWrite(LED_BUILTIN, LOW); toggle_state = 0;
        }
        break;
      //case 0x...: // if you want to use more buttons
    }
    receiver.resume();
  }
}
```

Upload the sketch to the Uno and press the chosen button on the remote. The state of on-board LED should be toggled.

Emulating the TV remote with the KY-005

In this chapter the TV remote is simulated using the KY-005 infrared LED module. Connect the KY-005 module with the Uno as shown on the following connection diagram:



KY-005 pin	Uno pin	Wire color
S	D3	Blue wire
Middle pin (VCC)	5V	Red wire
- (GND)	GND	Black wire
Button pin	Uno pin	
Bottom right pin	5V	Red wire
Top left pin	D2	Green wire
Top left pin (via resistor*)	GND	Black wire

* Resistor resistance value is 10kΩ

Az-Delivery

Sketch example

```
#include <IRremote.h>
#define BUTTON_PIN 2
uint8_t button_state = 0;
uint8_t toggle_state = 0;
IRsend transmitter;
void setup() {
    Serial.begin(9600);
    pinMode(LED_BUILTIN, OUTPUT);
    pinMode(BUTTON_PIN, INPUT);
}
void loop() {
    button_state = digitalRead(BUTTON_PIN);
    Serial.println(button_state);
    if (button_state == HIGH) {
        if (toggle_state == 0){
            transmitter.sendNEC(0x807F00FF, 32);
            digitalWrite(LED_BUILTIN, HIGH);
            toggle_state = 1;
        }
        else { // if you hold the button
            transmitter.sendNEC(0xFFFFFFFF, 32);
        }
    }
    else {
        digitalWrite(LED_BUILTIN, LOW);
        toggle_state = 0;
    }
    delay(200);
}
```

Az-Delivery

At the beginning of the sketch the *IRremote* library is included.

Next, one macro and two variables are created, where the macro represents the digital I/O pin number to which the push button is connected (2), and the variables are used to detect the state of the push button.

After that, the transmitter object is created.

In the *setup()* function, the serial communication is started with baud rate of *9600bps*. The pin modes for digital pin 13 is set to *OUTPUT* and button pin (2) mode is set to *INPUT*.

In the *loop()* function, the state of the push button is read and displayed in the Serial Monitor. Next, the state of the push button is checked if it is in the *HIGH* state. If it is, the state of a *toggle_state* variable is toggled, the *NEC* code is sent via an infrared LED and the on-board LED of the Uno is turned *ON*. If the push button is pressed for too long, the state of the *toggle_state* variable is not toggled anymore, and the *NEC* code *0xFFFFFFFF* is sent to simulate TV remote behavior.

If the state of the push button is *LOW*, the on-board LED of the Uno is turned *OFF* and the state of the *toggle_state* variable is set to zero value.

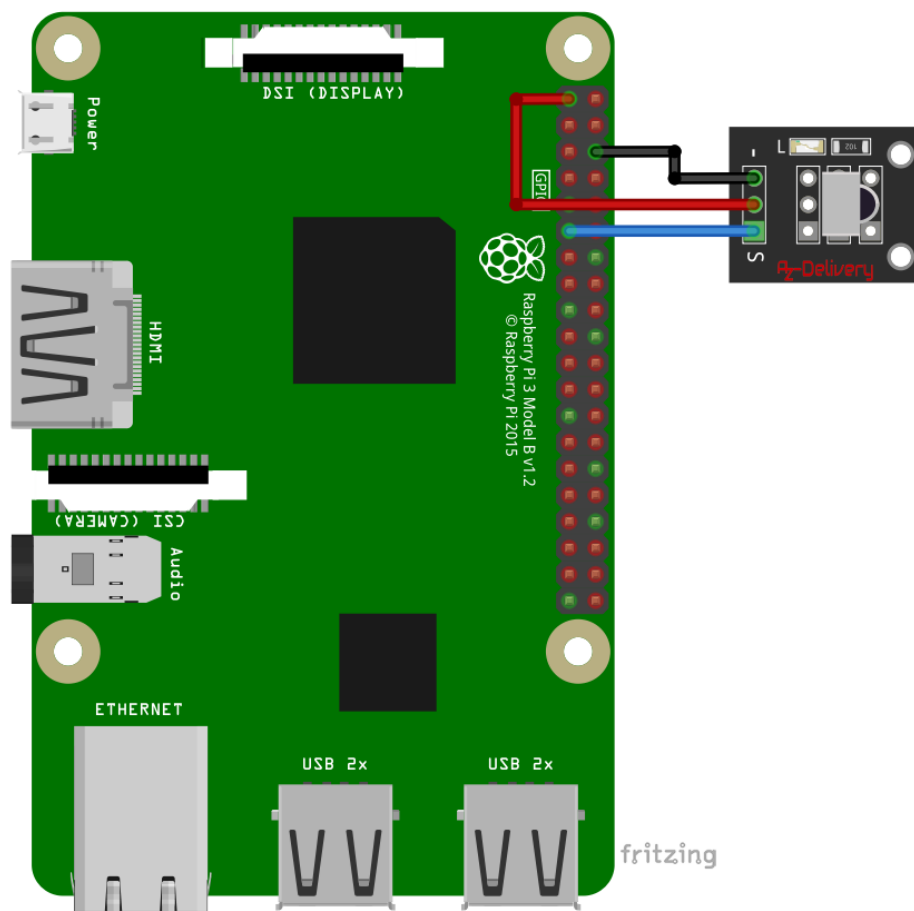
At the end of the *loop()* function, the delay of 200 milliseconds is set.

Az-Delivery

With the sketch from the chapter *Emulating the TV remote with the KY-005* the TV remote can be emulated. With the sketch from the chapter *Connecting the module with Uno* the TV receiver can be emulated. Combining these two sketches and modules, an infrared interface can be created for sending and receiving data in one direction (from infrared LED to infrared receiver).

Connecting the module with Raspberry Pi

Connect the KY-022 module with the Raspberry Pi as shown on the following connection diagram:



KY-022 pin	Raspberry Pi pin	Physical pin	Wire color
Middle pin (VCC)	3V3	1	Red wire
- (GND)	GND	8	Black wire
S	GPIO17	11	Blue wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import time, sleep
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
Signal_Pin = 17
GPIO.setup(Signal_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

code = None
CODES = {
    0xe0e040bf: "ON/OFF",
    0xe0e08877: "0",
    0xe0e020df: "1",
    0xe0e0a05d: "2",
    0xe0e0609f: "3",
    0xe0e010ef: "4",
    0xe0e0906f: "5",
    0xe0e050ad: "6",
    0xe0e030cf: "7",
    0xe0e0b04f: "8",
    0xe0e0708f: "9" # add more if you want
}

def binary_aquire(pin, duration):
    # aquires data as quickly as possible
    t0 = time()
    results = []
    while (time() - t0) < duration:
        results.append(GPIO.input(pin))
    return results
```

Az-Delivery

```
def rec(pinNo, bouncetime=150):
    global code, CODES
    data = binary_aquire(pinNo, bouncetime/1000.0) # 150us
    if len(data) < bouncetime:
        return
    rate = len(data) / (bouncetime / 1000.0)
    pulses = []
    i_break = 0
    for i in range(1, len(data)):
        if (data[i] != data[i-1]) or (i == len(data) - 1):
            pulses.append((data[i-1], int((i-i_break)/rate*1e6)))
            i_break = i
    outbin = ""
    for val, us in pulses:
        if val != 1:
            continue
        if outbin and us > 2000:
            break
        elif us < 1000:
            outbin += "0"
        elif 1000 < us < 2000:
            outbin += "1"
    try:
        code = int(outbin, 2)
        print('Key {} -> {}'.format(str(hex(code)), CODES[code]))
    except ValueError:
        code = None
    except KeyError:
        print('Key {} is not defined.'.format(str(hex(code))))
```

Az-Delivery

```
GPIO.add_event_detect(17, GPIO.FALLING, callback=rec, bouncetime=100)

print('[Press CTRL + C to end the script!]\n')
try:
    print('Starting IR Listener')
    print('Waiting for signal')
    while True:
        sleep(0.000033) # 33 microseconds

except KeyboardInterrupt:
    print('\nScript end!\n')

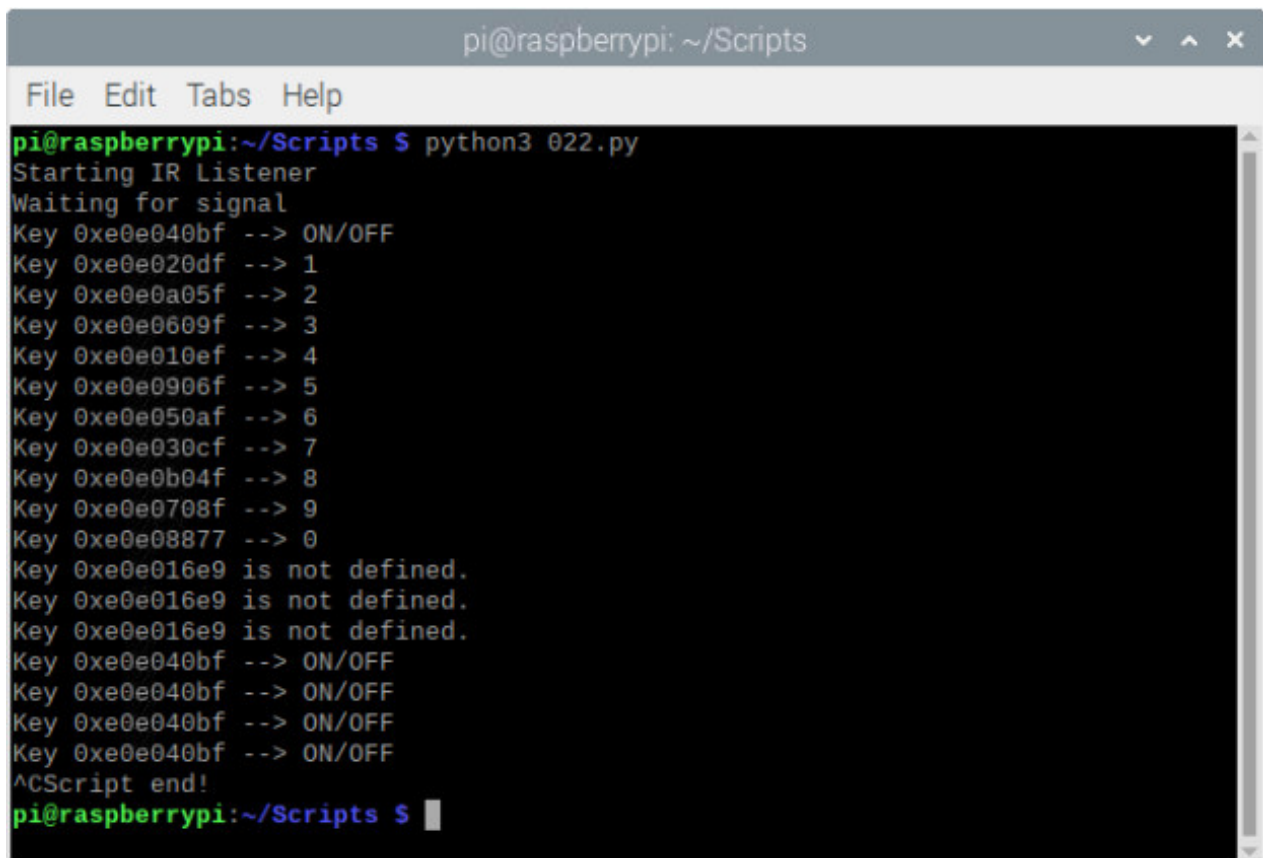
finally:
    GPIO.cleanup()
```


Az-Delivery

Save the script by the name *ky022.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 ky022.py

The result should look like the output on the following image:

A screenshot of a terminal window titled 'pi@raspberrypi: ~/Scripts'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal output shows the command 'python3 022.py' being executed. The script starts by printing 'Starting IR Listener' and 'Waiting for signal'. It then receives a series of IR key codes and maps them to numbers 1 through 9, and 'ON/OFF'. The key codes are: 0xe0e040bf (ON/OFF), 0xe0e020df (1), 0xe0e0a05f (2), 0xe0e0609f (3), 0xe0e010ef (4), 0xe0e0906f (5), 0xe0e050af (6), 0xe0e030cf (7), 0xe0e0b04f (8), 0xe0e0708f (9), and 0xe0e08877 (0). It also shows three instances of 'Key 0xe0e016e9 is not defined.' followed by four more 'ON/OFF' outputs for the 0xe0e040bf key code. The script ends with '^CScript end!'. The prompt 'pi@raspberrypi:~/Scripts \$' is visible at the bottom.

```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 022.py
Starting IR Listener
Waiting for signal
Key 0xe0e040bf --> ON/OFF
Key 0xe0e020df --> 1
Key 0xe0e0a05f --> 2
Key 0xe0e0609f --> 3
Key 0xe0e010ef --> 4
Key 0xe0e0906f --> 5
Key 0xe0e050af --> 6
Key 0xe0e030cf --> 7
Key 0xe0e0b04f --> 8
Key 0xe0e0708f --> 9
Key 0xe0e08877 --> 0
Key 0xe0e016e9 is not defined.
Key 0xe0e016e9 is not defined.
Key 0xe0e016e9 is not defined.
Key 0xe0e040bf --> ON/OFF
Key 0xe0e040bf --> ON/OFF
Key 0xe0e040bf --> ON/OFF
Key 0xe0e040bf --> ON/OFF
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL* + *C* on the keyboard.

Az-Delivery

The script starts with the import of two libraries, *RPi.GPIO* and *time*, where the first is used for GPIO names, and second for timing functions. Then the GPIO naming is set to *BCM* and any warnings related to GPIO interfaces are disabled.

After that the mode of GPIO pin 17 is set to *INPUT* with internal PULL DOWN resistor, with the following line of code:

```
GPIO.setup(Signal_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

The variable called *code* is created where the decoded code value received from the TV remote is stored.

Also, the *dictionary* called *CODES* is created, which is a key-value pair for specific codes. In the *CODES* dictionary the *keys* are names for the specific hex code *values* of the TV remote.

Then two functions are created. The first is called *binary_acquire()* which has two arguments, pin name and duration of recording. This function returns a list called *results*. The *binary_acquire()* function is responsible to read the data from infrared receiver module as fast as possible.

The second function is called *rec()* has two arguments, pin number and bouncetime (bouncetime argument is optional). This function returns no value. The *rec()* function is responsible for reading, decoding and displaying received data.

Az-Delivery

Inside the `rec()` function, the `binary_aquire()` function is executed in order to get data from the infrared sensor. The duration of reading is in microseconds because infrared signals from the infrared transmitter are in the *kHz* range (*38kHz*, mentioned at the beginning of this chapter). Then received data is split into pulses. To get the gist, look on the following image:



The received signal looks like several pulses with different durations (mentioned at the beginning of this chapter). Some pulses represent the logic “1” and some logic “0”.

To get what is a logic “1” and what is a logic “0”, after the splitting the data into pulses, the code of pulses is decoded. We do this with the following lines of code:

```
outbin = ""
for val, us in pulses:
    if val != 1:
        continue
    if outbin and us > 2000:
        break
    elif us < 1000:
        outbin += "0"
    elif 1000 < us < 2000:
        outbin += "1"
```

Az-Delivery

The *outbin* is the temporary variable used to build up a code. The *val* variable represents the state of the pulse, *HIGH* or *LOW*, and the *us* represents microseconds. If the value in the *val* variable is different than *HIGH* (logic “1”) which means the pulse is in a *LOW* state, that is not a point of interest for decoding algorithm, so it is dismissed. After this, the length of a *HIGH* state of the pulse is checked. If the pulse is longer than 2000 microseconds, that means a pulse is not logic “1” or “0” so it is dismissed. If the pulse is shorter than 1000 microseconds that pulse is logic “0”. If the pulse lasts from 1000 to 2000 microseconds that pulse is logic “1”.

At the end of the *rec()* function, the code is converted into a hexadecimal number and displayed to the terminal. Displaying is in the *try-except* block because the code is read from the *CODES* dictionary which sometimes can not be done (when there is no specific *key* in the *dictionary*). To prevent this error, this part of code has to be in the *try-except* block of code. If there is no defined code in the dictionary, Python raises a *KeyError*.

After creating the functions the interrupt routine is set. Whenever there is a falling edge of the digital signal on the GPIO pin 17, *rec()* function is executed.

Az-Delivery

At the end of the script the try-except-finally block of code is executed. In the try block of code the indefinite loop (*while True:*) is created to keep the script running.

To end the script press *CTRL + C* on the keyboard. This is called the keyboard interrupt, and when this interrupt happens, the *except* block of code is executed (*except KeyboardInterrupt:*) displaying message *Script end!* in the terminal.

The *finally* block of code is executed on the script ending. It is used to disable any previously used GPIO pin modes and interfaces by executing function called `cleanup()`.



KY-023 Joystick Module

Introduction

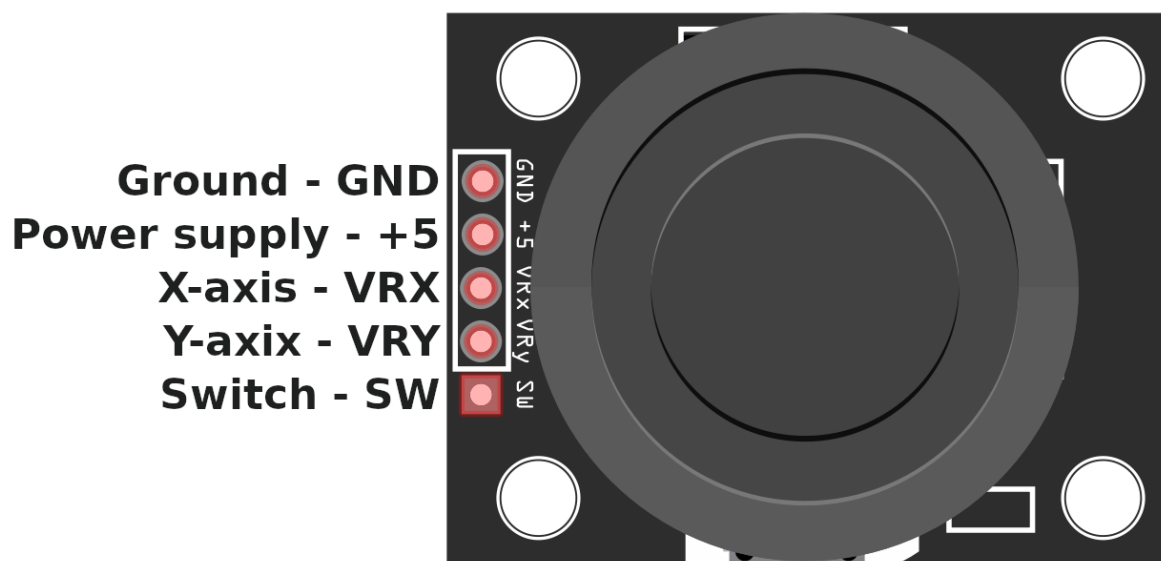
The joystick module is an easy-to-use analog joystick for a microcontrollers. The joystick features two-axis (X and Y axis) as well as a button switch that is activated when the joystick is pressed. The joystick is basically a combination of two potentiometers. This means that when the joystick is moving along the X -axis the resistance of the potentiometer changes and when voltage is applied, resistance change results in a change of voltage. The voltage can be used to detect the X position by connecting the VRX pin of the joystick to an analog input $A0$ pin of the Uno. The same applies for the Y -axis. The Y -axis position can be read by connecting VRY to analog input $A1$ pin of the Uno. Press state or the state of the SW pin can be connected to the digital pin $D2$ of the Uno. Because the module consist of two potentiometers, it can be used with any power supply voltage, for example it can work on both $+3.3V$ and $+5V$ DC.

Specifications

Operating Voltage	from 3.3V to 5V DC
Output	digital, with high sensitivity
Dimensions	26x34x32mm [1.02x1.34x1.26in]

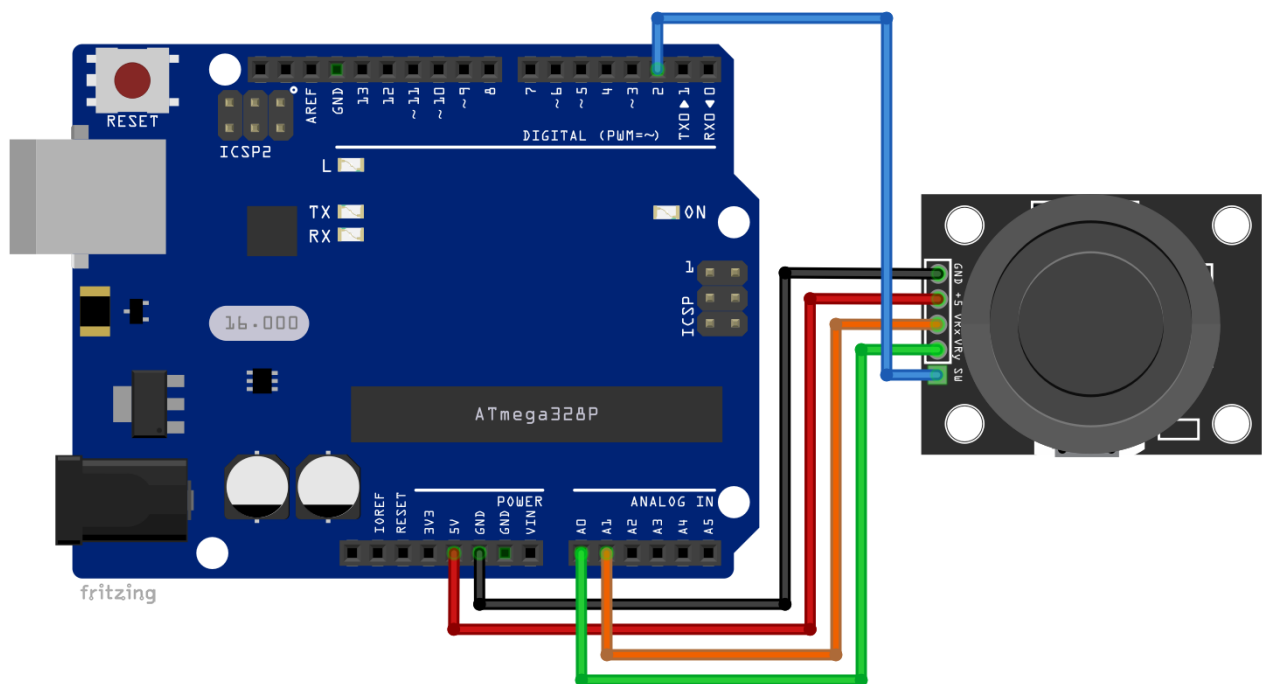
The pinout

The KY-023 joystick module has five pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-023 module with the Uno as shown on the following connection diagram:



KY-023 pin	Uno Pin	Wire color
GND	GND	Black wire
+5V	5V	Red wire
VRX	A1	Orange wire
VRY	A0	Green wire
SW	D2	Blue wire

Az-Delivery

Sketch example

```
#define VX_PIN 0
#define VY_PIN 1
#define BUTTON_PIN 2
uint8_t value = 0;

void setup() {
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  value = analogRead(VX_PIN);
  Serial.print("X:");
  Serial.print(value, DEC);

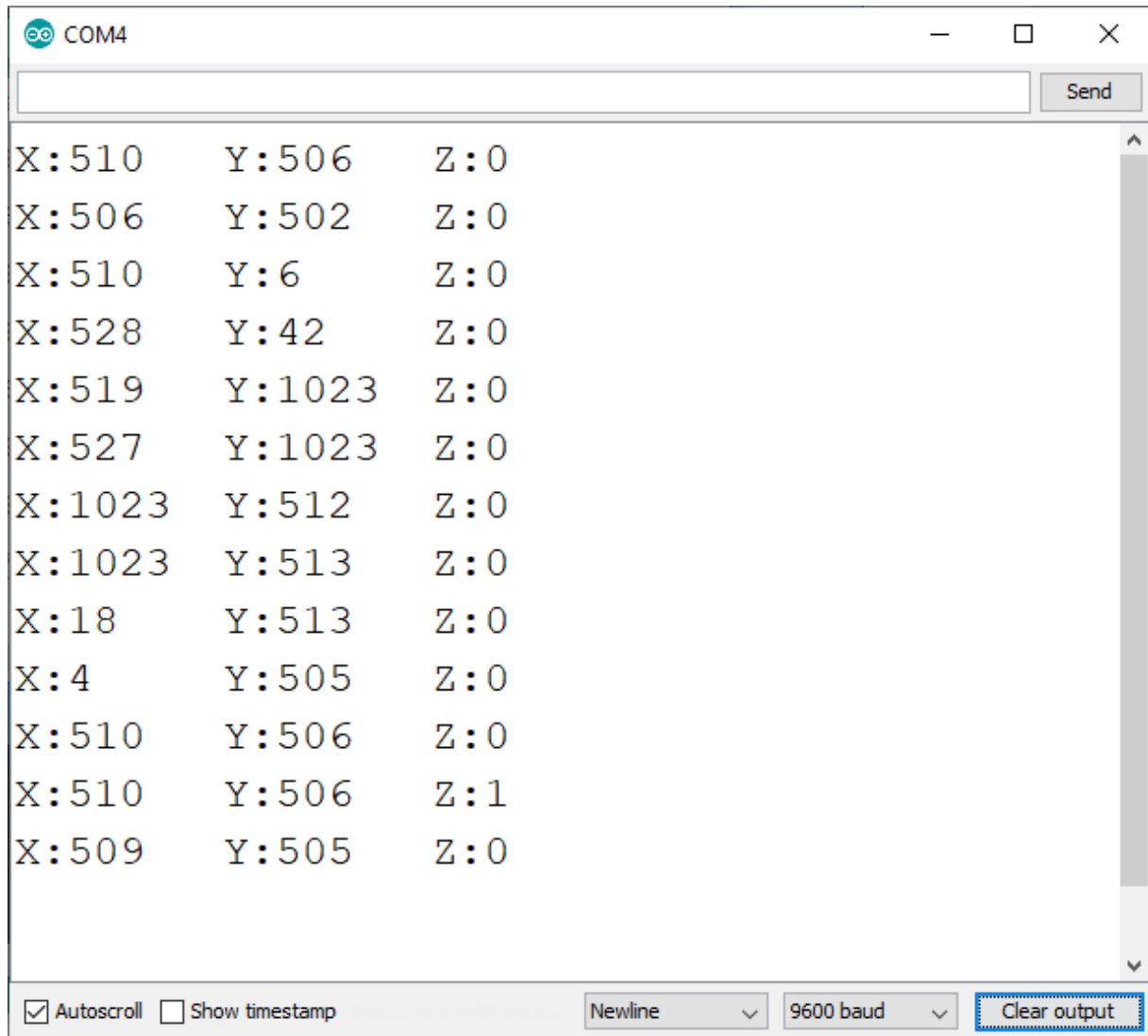
  value = analogRead(VY_PIN);
  Serial.print("\tY:");
  Serial.print(value, DEC);

  value = !digitalRead(BUTTON_PIN);
  Serial.print("\tZ:");
  Serial.println(value, DEC);

  delay(500);
}
```

Az-Delivery

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



To get the output from the image move or press the Joystick.

Az-Delivery

Connecting the module with Raspberry Pi

Connect the KY-023 module with the Uno as shown on the connection diagram from the chapter *Connecting the module with Uno*, and then connect the Uno with the Raspberry Pi via usb cable. Next, upload *nanpy* firmware to the Uno (as show in the chapter *Uno as ADC*), and use the following code to control the KY-023 module:

```
from nanpy import (ArduinoApi, SerialManager)
from time import sleep
try:
    connection1 = SerialManager()
    a = ArduinoApi(connection=connection1)
except:
    print('Failed to connect to the Arduino')

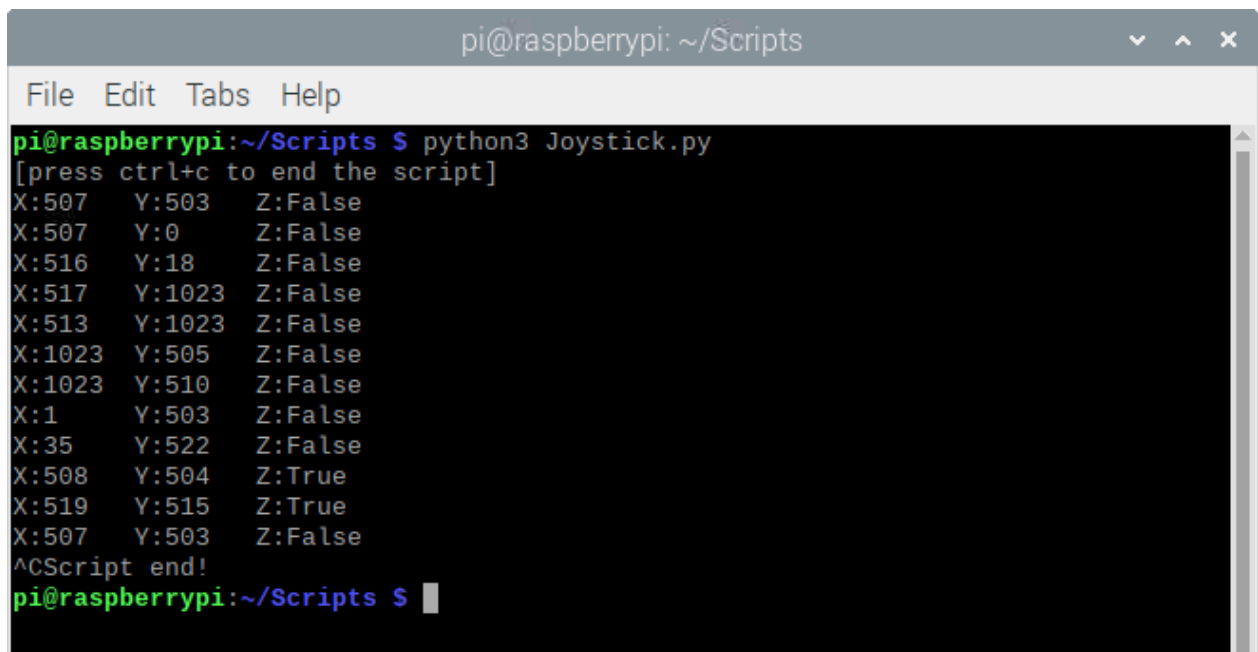
a.pinMode(2, a.INPUT_PULLUP)
valueX, valueY, valueZ = 0, 0, 0
print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        valueX = a.analogRead(0)
        valueY = a.analogRead(1)
        valueZ = not a.digitalRead(2)
        print('X:{}\tY:{}\tZ:{}'.format(valueX, valueY, valueZ))
        sleep(0.5)
except KeyboardInterrupt:
    print('\nScript end!')
```

Az-Delivery

Save the script by the name *Joystick.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 Joystick.py

The result should look like the output on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 Joystick.py
[press ctrl+c to end the script]
X:507 Y:503 Z:False
X:507 Y:0 Z:False
X:516 Y:18 Z:False
X:517 Y:1023 Z:False
X:513 Y:1023 Z:False
X:1023 Y:505 Z:False
X:1023 Y:510 Z:False
X:1 Y:503 Z:False
X:35 Y:522 Z:False
X:508 Y:504 Z:True
X:519 Y:515 Z:True
X:507 Y:503 Z:False
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL + C* on the keyboard.

KY-024 Digital Hall Sensor Module

Introduction

The KY-024 Hall sensor module is a very responsive switch that gets activated by a magnetic field. It has a potentiometer to adjust the sensitivity of the sensor and it provides both analog and digital output pins.

The digital output pin acts as a switch that turns *ON* or *OFF* when one pole of the magnet is near the sensor.

On the other hand, the analog output pin can output the polarity and the relative strength of the magnetic field.

The Hall sensor on-board the KY-024 module is the same as the sensor on the KY-003 Hall sensor module. The differences are the *LM393* dual analog voltage comparator integrated circuit and the potentiometer on the KY-024 module.

Specifications

Operating Voltage range	from 2.7V to 6.5V DC
Sensitivity min.	1.0 mV/G
Sensitivity typical	1.4 mV/G
Sensitivity max.	1.75 mV/G
Output	digital, with high sensitivity
Dimensions	15x36mm [0.6x0.75in]

Analog output pin voltage sensitivity can be adjusted via the on-board blue potentiometer.

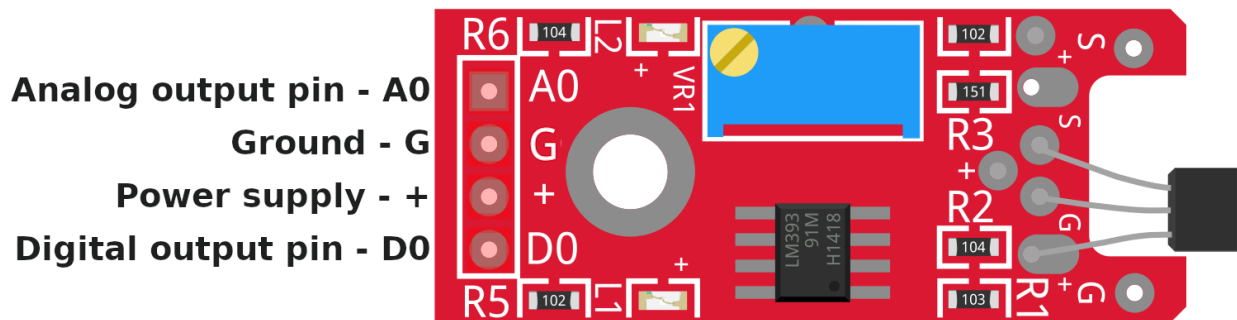
With some adjustment of the potentiometer, when the analog voltage is read via an analog to digital converter (ADC for short) inside the microcontroller on the Uno, the ADC value of the voltage is on the value around 500, and that is when there is no magnetic field near the Hall sensor.

When one pole of the magnet is near the sensor, the ADC value drops to the value around 170. This is the magnet pole on which digital output pin outputs the *HIGH* state (is turned *ON*).

When the opposite pole of the magnet is near the sensor, the ADC value rises to the value about 880. This is the magnet pole on which digital output pin outputs the *LOW* state (is turned *OFF*).

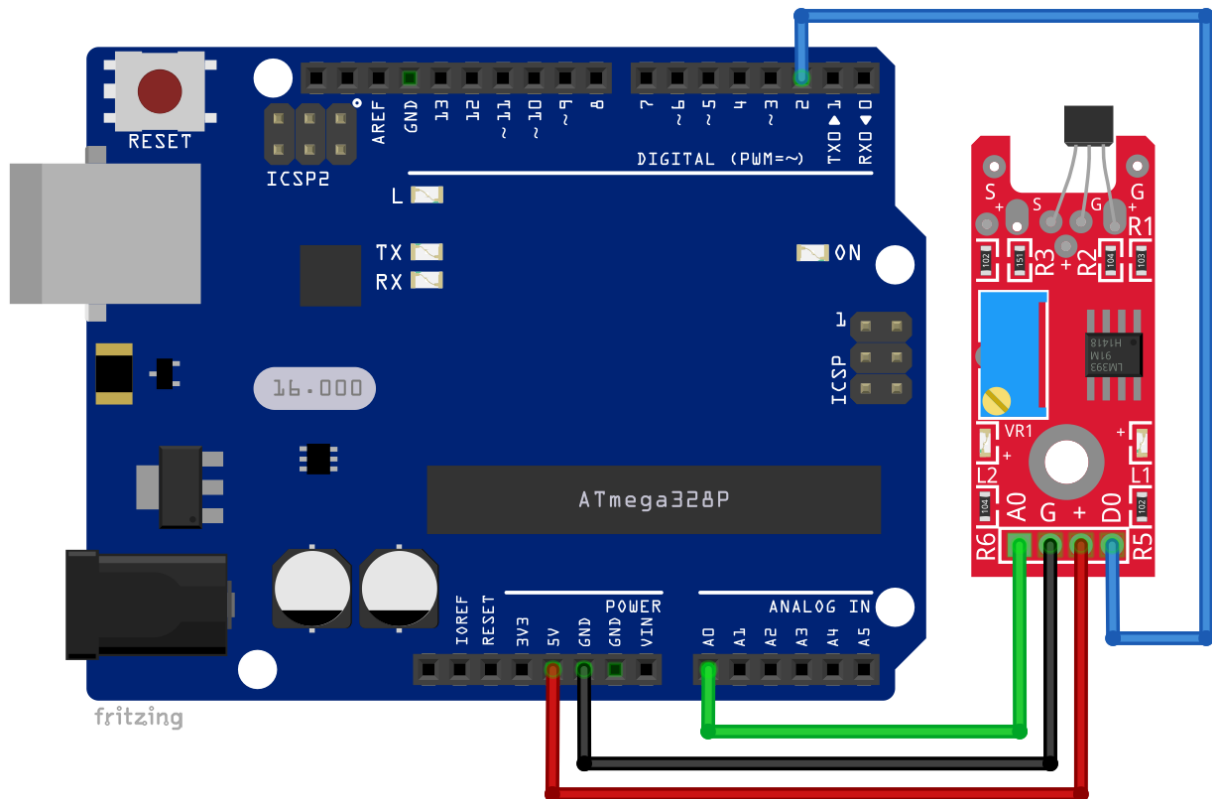
The pinout

The KY-024 digital Hall sensor module has four pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-024 module with the Uno as shown on the following connection diagram:



KY-024 pin	Uno pin	Wire color
+	5V	Red wire
GND	GND	Black wire
A0	A0	Green wire
D0	D2	Blue wire

With this set-up both the analog and digital pins of the KY-024 module are used.

Az-Delivery

Sketch example

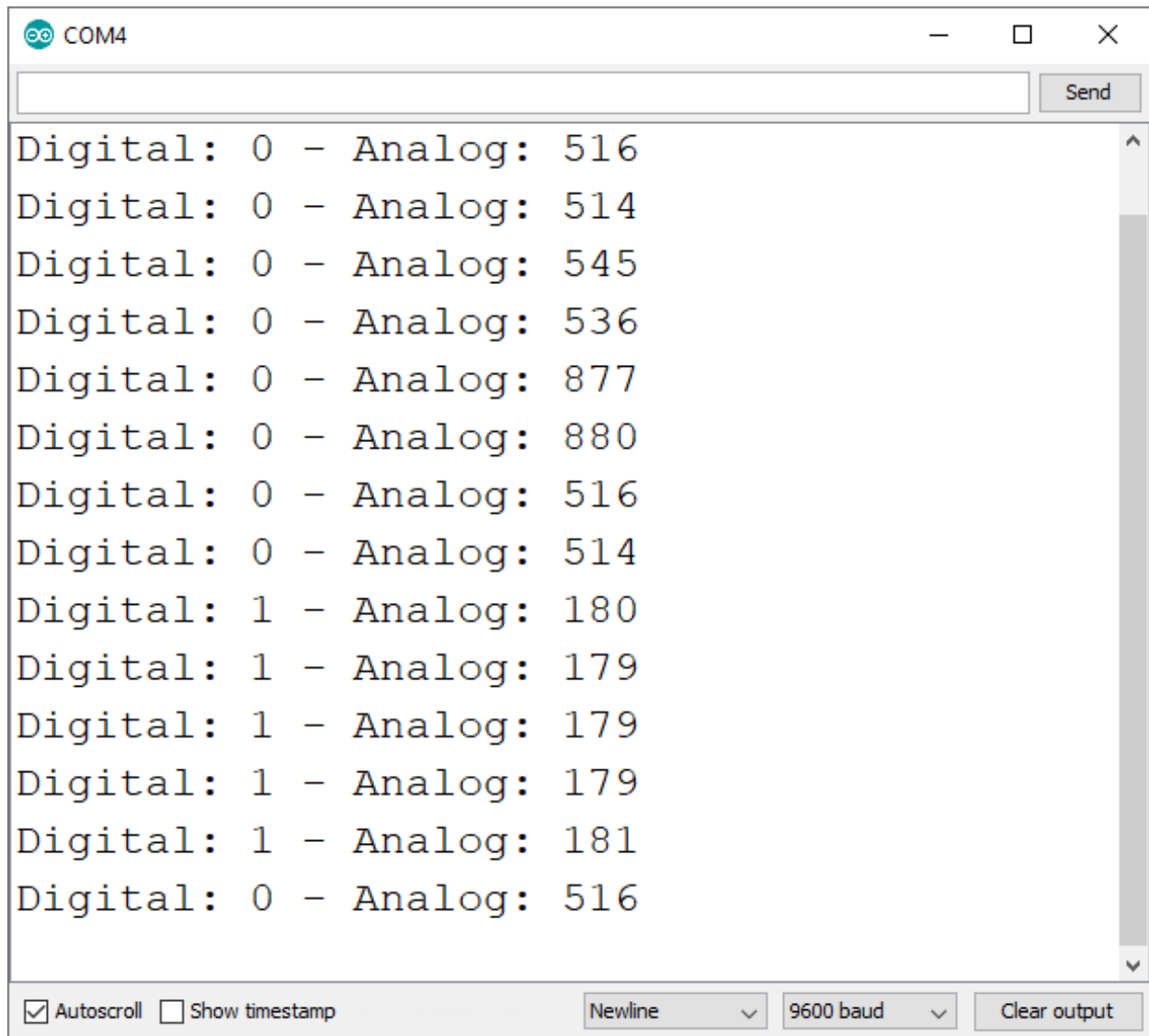
```
#define HALL_D_PIN 2
#define HALL_A_PIN 0

void setup() {
  pinMode(HALL_D_PIN, INPUT);
  Serial.begin(9600);
}

void loop() {
  Serial.print("Digital: ");
  Serial.print(digitalRead(HALL_D_PIN));
  Serial.print(" - Analog: ");
  Serial.println(analogRead(HALL_A_PIN));
  delay(1000);
}
```

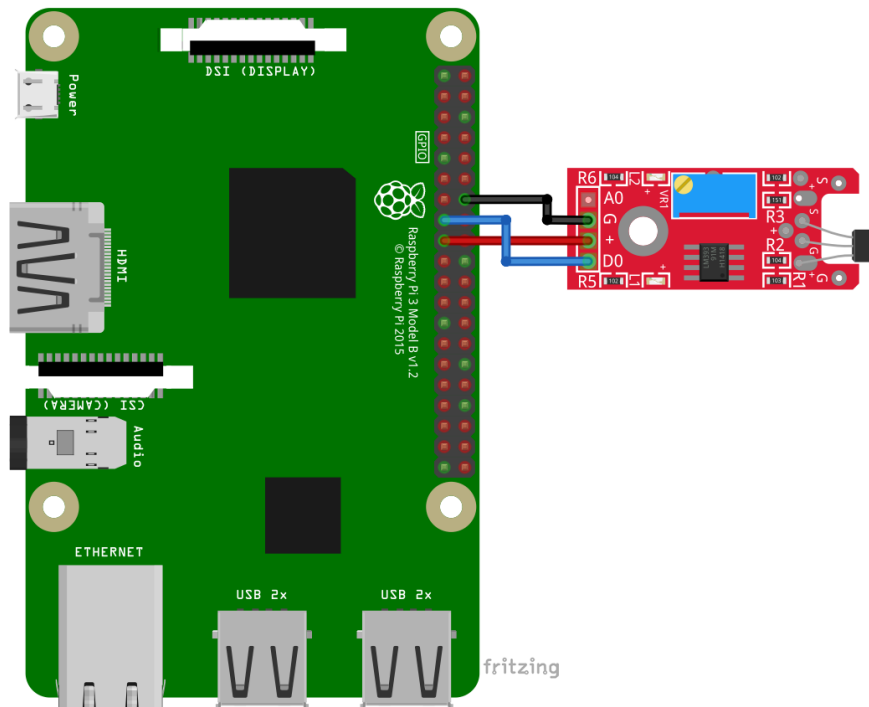
Az-Delivery

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



Connecting the module with Raspberry Pi

Connect the KY-024 module with the Raspberry Pi as shown on the following connection diagram:



KY-024 pin	Raspberry Pi pin	Physical pin	Wire color
GND	GND	14	Black wire
D0	GPIO22	15	Blue wire
+ (VCC)	3V3	17	Red wire

With this set-up, the Raspberry Pi is used to only read the digital pin of the KY-024 module. If the analog pin has to be used, use the analog to digital converter of the Uno or external analog to digital converter like the *ADS1115* (which is not covered in this eBook).

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep

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

Digital_PIN = 22
GPIO.setup(Digital_PIN, GPIO.IN)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        if GPIO.input(Digital_PIN):
            print('Magnet detected')
        else:
            print('No magnet')

        sleep(1)

except KeyboardInterrupt:
    print('\nScript end!')

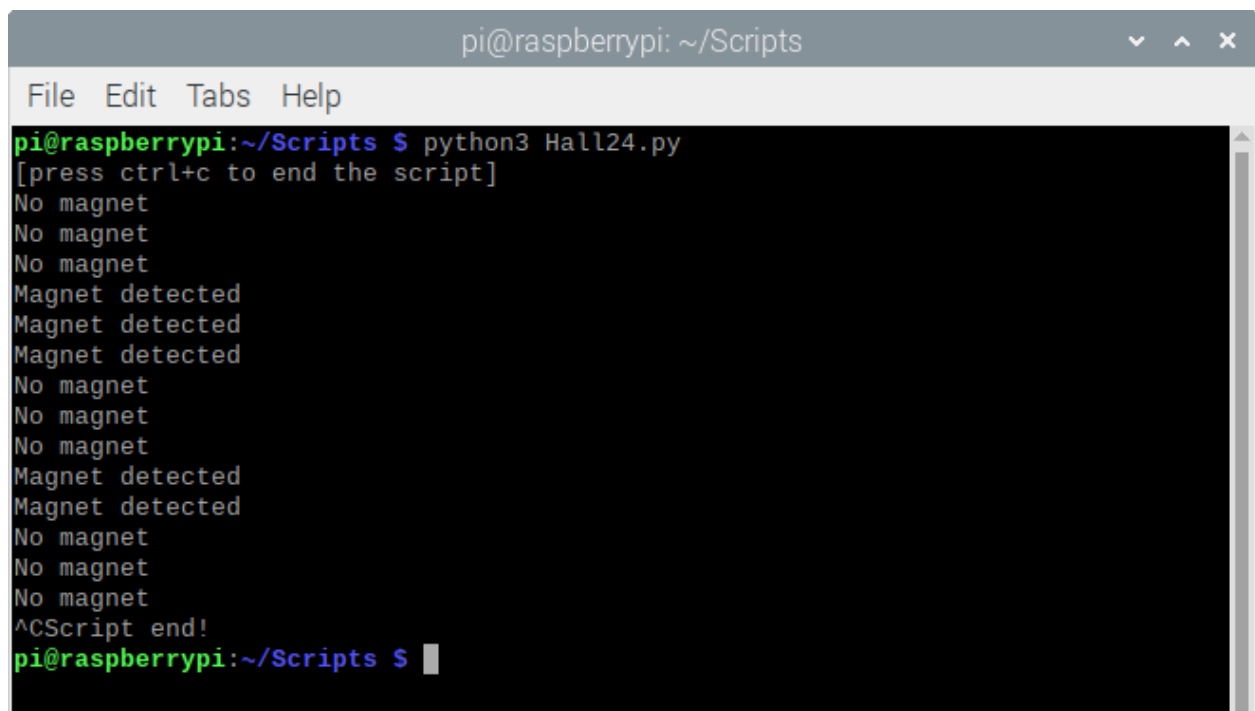
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *Hall24.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 Hall24.py

The result should look like the output on the following image:

A screenshot of a terminal window titled 'pi@raspberrypi: ~/Scripts'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal shows the command 'python3 Hall24.py' being executed. The output of the script is as follows:

```
pi@raspberrypi:~/Scripts $ python3 Hall24.py
[press ctrl+c to end the script]
No magnet
No magnet
No magnet
Magnet detected
Magnet detected
Magnet detected
No magnet
No magnet
No magnet
Magnet detected
Magnet detected
No magnet
No magnet
No magnet
^CScript end!
pi@raspberrypi:~/Scripts $
```

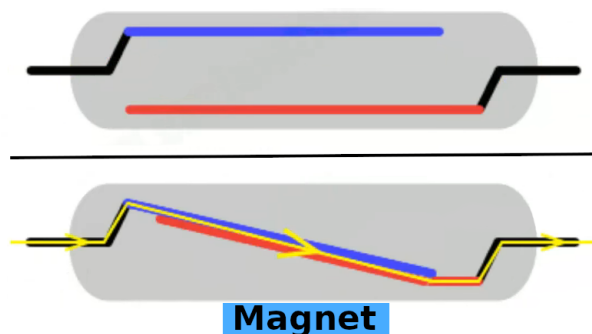
To stop the script press *CTRL* + *C* on the keyboard.

KY-025 Reed Sensor Module

Introduction

The KY-025 reed sensor module is a reed switch, a magnetic switch that changes its state when exposed to a magnetic field. The module is normally opened, which means, when there is no magnetic field near the module, contacts in the switch are opened. When there is a magnetic field near the module, the contacts in the switch close, enabling current flow through a switch.

The contacts in the reed module are magnetic, one is the north magnetic pole and the other is the south magnetic pole. When the magnet is near the reed sensor, one contact is repelled and the other is attracted, which closes the switch enabling current flow through it.



Because it is a simple switch, the KY-021 module can be operated with any voltage. The output of the KY-021 module is digital, which means that it can be used with Uno or Raspberry Pi.

Specifications

Operating voltage range	from 2.7V to 5V DC
Output	digital and analog
Dimensions	15x36mm [0.6x1.4in]

The analog output pin voltage sensitivity can be adjusted via an on-board blue potentiometer.

With some adjustment of the potentiometer, when the analog voltage is read via an analog to digital converter (ADC for short) inside the microcontroller on the Uno, the ADC value of the voltage is around the value *1023*, and that is when there is no magnetic field near the *reed* sensor.

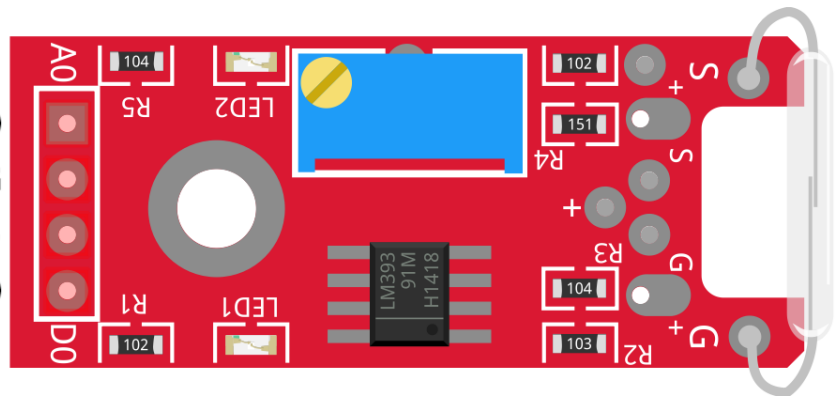
When one pole of the magnet is near the sensor, the ADC value drops to the value around *0*. This is the magnet pole on which the digital output pin outputs the *HIGH* state (is turned *ON*).

When the opposite pole of the magnet is near the sensor, the *reed* sensor acts like there is no magnet near the sensor. This is the state in which the digital output pin outputs the *LOW* state (is turned *OFF*).

The pinout

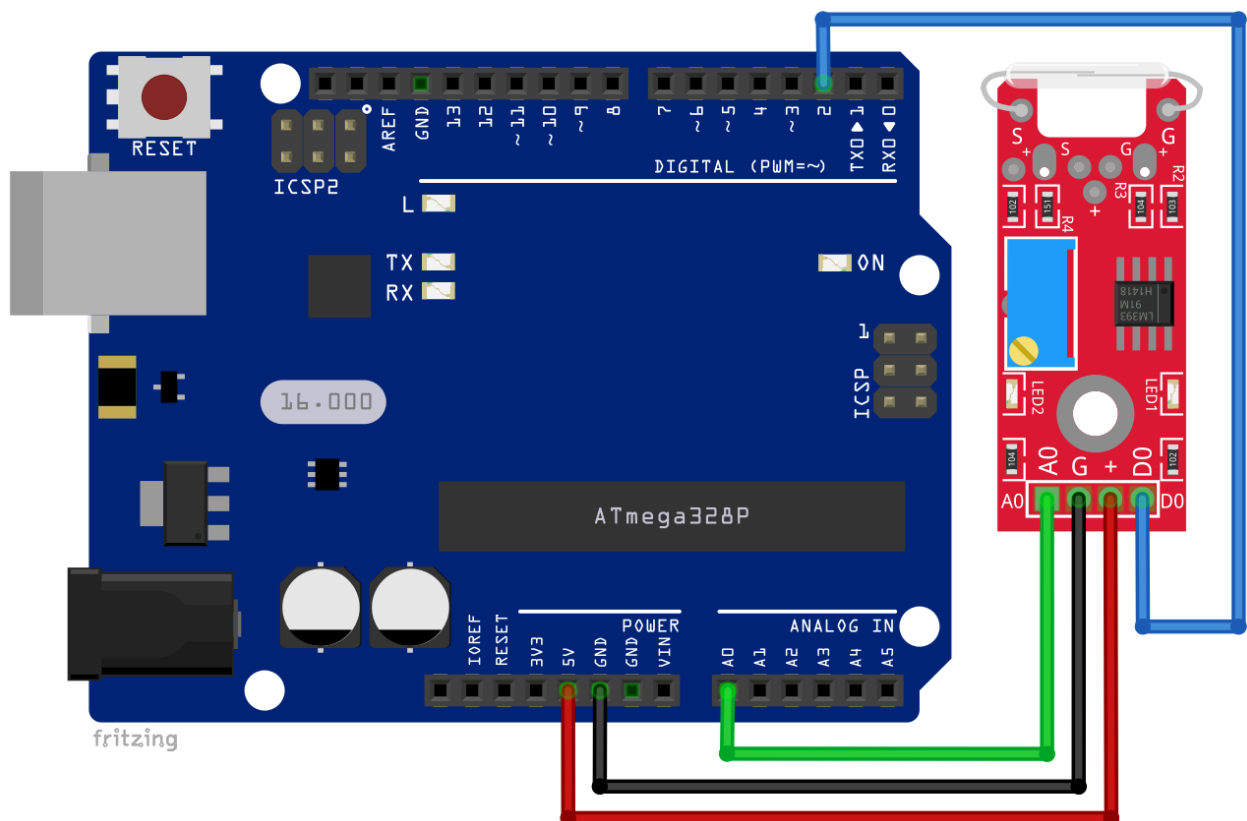
The KY-025 reed sensor module has four pins. The pinout diagram is shown on the following image:

Analog output pin - A0
Ground - G
Power supply - +
Digital output pin - D0



Connecting the module with Uno

Connect the KY-025 module with the Uno as shown on the following connection diagram:



KY-025 pin	Uno pin	Wire color
+	5V	Red wire
GND	GND	Black wire
A0	A0	Green wire
D0	D2	Blue wire

With this set-up, both analog and digital pins of the KY-025 module are used.

Az-Delivery

Sketch example

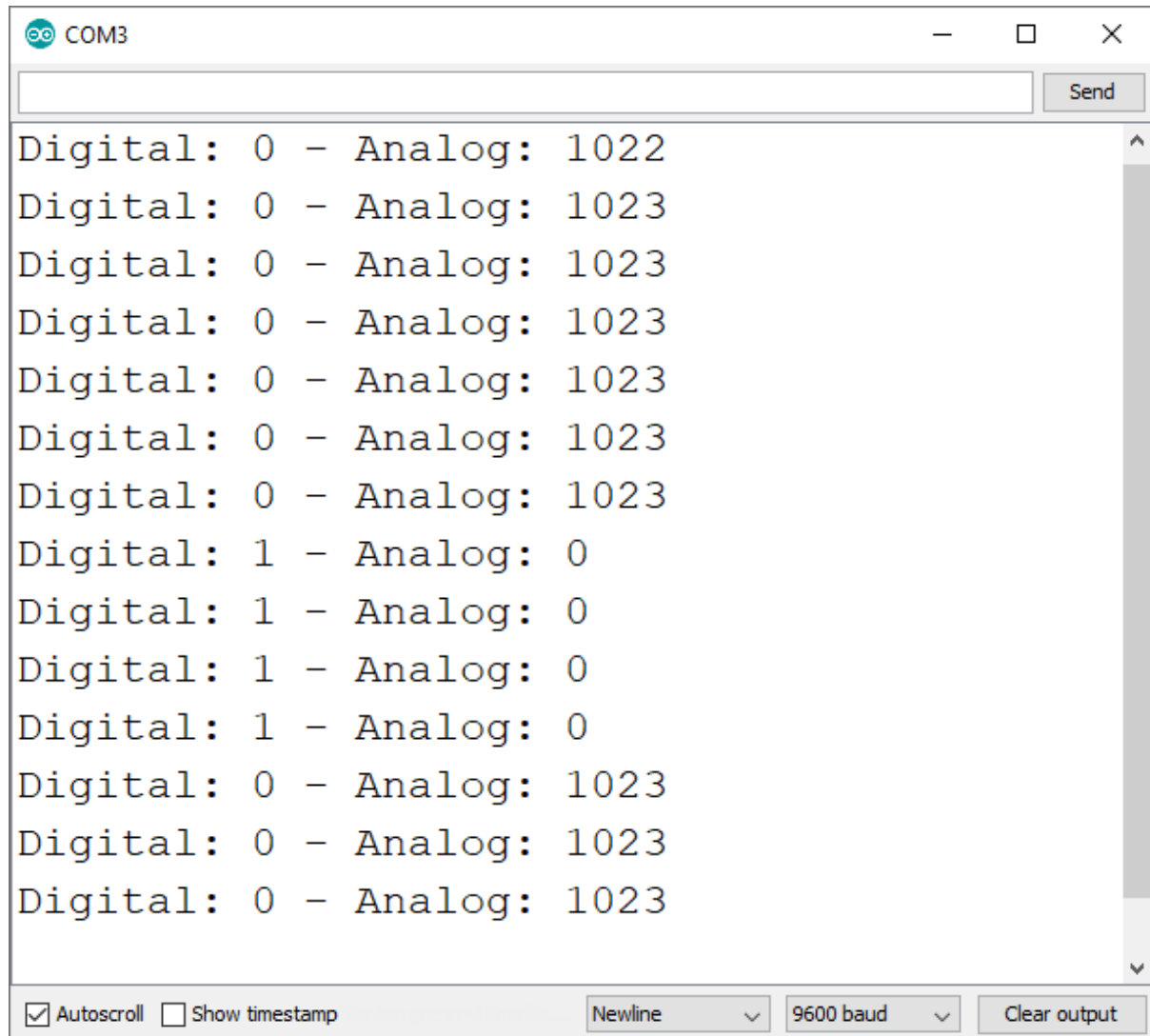
```
#define REED_D_PIN 2
#define REED_A_PIN 0

void setup() {
  pinMode(REED_D_PIN, INPUT);
  Serial.begin(9600);
}

void loop() {
  Serial.print("Digital: ");
  Serial.print(digitalRead(REED_D_PIN));
  Serial.print(" - Analog: ");
  Serial.println(analogRead(REED_A_PIN));
  delay(1000);
}
```

Az-Delivery

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



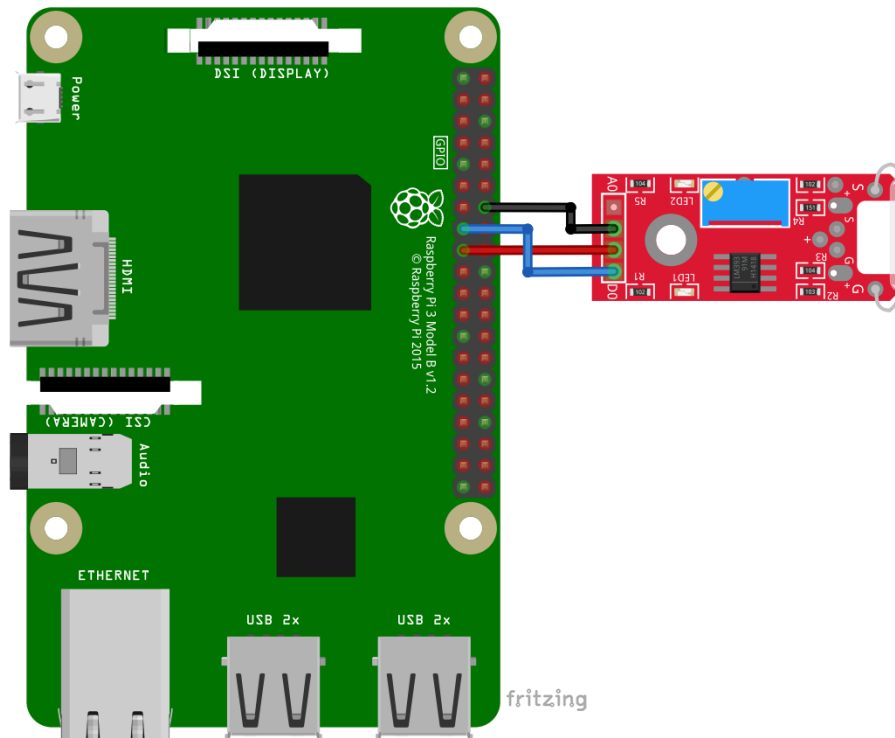
The screenshot shows the Serial Monitor window for COM3. The window title bar includes standard OS controls (minimize, maximize, close) and the text 'COM3'. Below the title bar is a text input field and a 'Send' button. The main area of the window displays a series of 15 lines of text, each representing a sensor reading: 'Digital: 0 - Analog: 1022' followed by six lines of 'Digital: 0 - Analog: 1023', then three lines of 'Digital: 1 - Analog: 0', and finally three more lines of 'Digital: 0 - Analog: 1023'. A vertical scrollbar is on the right side of the text area. At the bottom of the window, there are checkboxes for 'Autoscroll' (checked) and 'Show timestamp' (unchecked). To the right of these are two dropdown menus: 'Newline' (set to '\n') and '9600 baud' (set to '9600'). On the far right of the bottom bar is a 'Clear output' button.

```
COM3
Digital: 0 - Analog: 1022
Digital: 0 - Analog: 1023
Digital: 0 - Analog: 1023
Digital: 0 - Analog: 1023
Digital: 0 - Analog: 1023
Digital: 0 - Analog: 1023
Digital: 0 - Analog: 1023
Digital: 1 - Analog: 0
Digital: 1 - Analog: 0
Digital: 1 - Analog: 0
Digital: 1 - Analog: 0
Digital: 0 - Analog: 1023
Digital: 0 - Analog: 1023
Digital: 0 - Analog: 1023
```

☒ Autoscroll ☐ Show timestamp Newline 9600 baud Clear output

Connecting the module with Raspberry Pi

Connect the KY-025 module with the Raspberry Pi as shown on the following connection diagram:



KY-025 pin	Raspberry Pi pin	Physical pin	Wire color
GND	GND	14	Black wire
D0	GPIO22	15	Blue wire
+ (VCC)	3V3	17	Red wire

With this set-up, the Raspberry Pi is used to read only the digital pin of the KY-025 module. If the analog pin has to be used, use the analog to digital converter of the Uno or external analog to digital converter like the *ADS1115* (which is not covered in this eBook).

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep

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

Digital_PIN = 22
GPIO.setup(Digital_PIN, GPIO.IN)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        if GPIO.input(Digital_PIN):
            print('Magnet detected')
        else:
            print('No magnet')

        sleep(1)

except KeyboardInterrupt:
    print('\nScript end!')

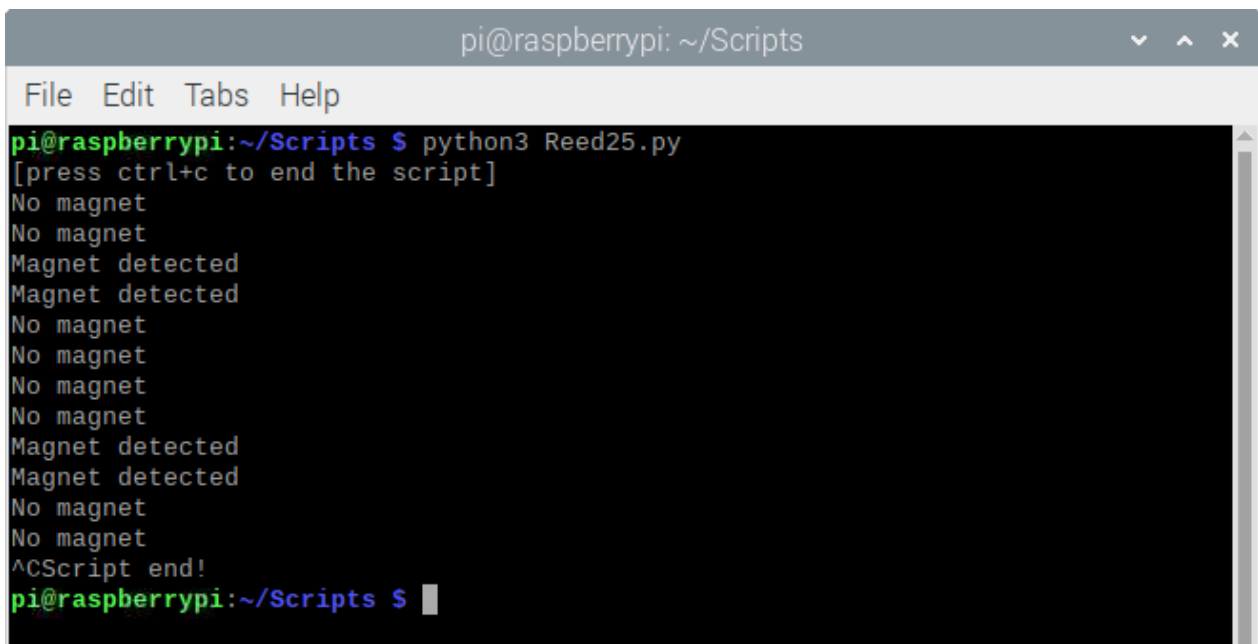
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *Reed25.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 Reed25.py

The result should look like the output on the following image:

A screenshot of a terminal window titled 'pi@raspberrypi: ~/Scripts'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal output shows the command 'python3 Reed25.py' being executed. The script prints a series of messages: '[press ctrl+c to end the script]', 'No magnet', 'No magnet', 'Magnet detected', 'Magnet detected', 'No magnet', 'No magnet', 'No magnet', 'No magnet', 'Magnet detected', 'Magnet detected', 'No magnet', 'No magnet', and finally '^CScript end!'. The prompt 'pi@raspberrypi:~/Scripts \$' is visible at the bottom with a cursor.

To stop the script press *CTRL* + *C* on the keyboard.



KY-026 Flame Detector Module

Introduction

The KY-026 flame detector module is a sensor that detects the flame. It has an infrared diode that can detect infrared rays of light from the flame. It is very sensitive in the infrared light wavelength range from $760nm$ to $1100nm$, which is a wavelength of light from the flame. The detection distance of the KY-026 module is from a few millimeters up to 2.5 meters.

The module outputs both an analog and digital signal. The *HIGH* state of the digital signal signalizes that there is a flame in front of the sensor. The lower voltage value on the analog output pin the closer (or larger) is the flame in front of the sensor. When there is no flame in front of the sensor, the digital output pin is in the *LOW* state, and the analog output pin outputs a value around VCC voltage.

Specifications

Operating voltage range	from 3.3V to 5.5V
Infrared wavelength detection	from 760nm to 1100nm
Detection angle	60°
Dimensions	15x36mm [0.6x1.4in]

The analog output pin voltage sensitivity can be adjusted via an on-board blue potentiometer.

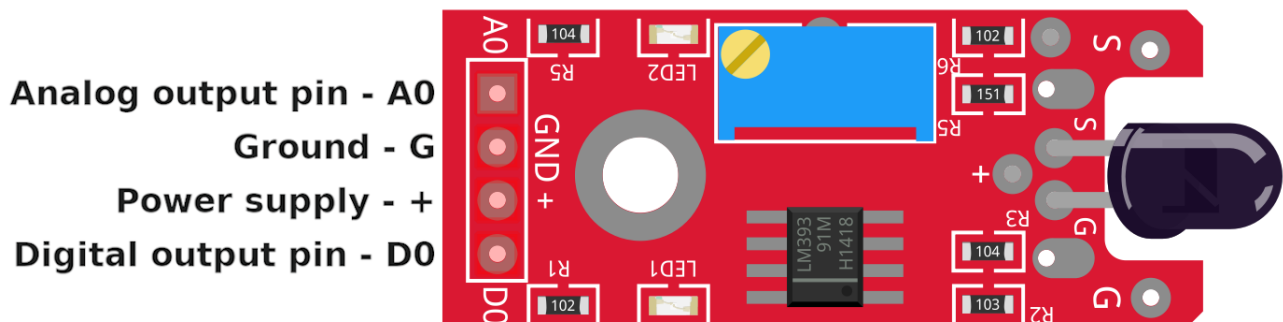
With some adjustment of the potentiometer, when the analog voltage is read via an analog to digital converter (ADC for short) inside the microcontroller on the Uno, the ADC value of the voltage is on the value around 930. That is when there is no flame in front of the KY-026 sensor.

When a flame is in front of the sensor, the ADC value drops to the value around 20. This is the state when the digital output pin outputs the *HIGH* state (is turned *ON*).

When a flame is moved away from the sensor, the ADC value on analog to digital converter, starts to rise (as seen on the image of the Serial Monitor, later in the text). The digital output pin outputs the *LOW* state (is turned *OFF*) when a flame is too far away from the sensor.

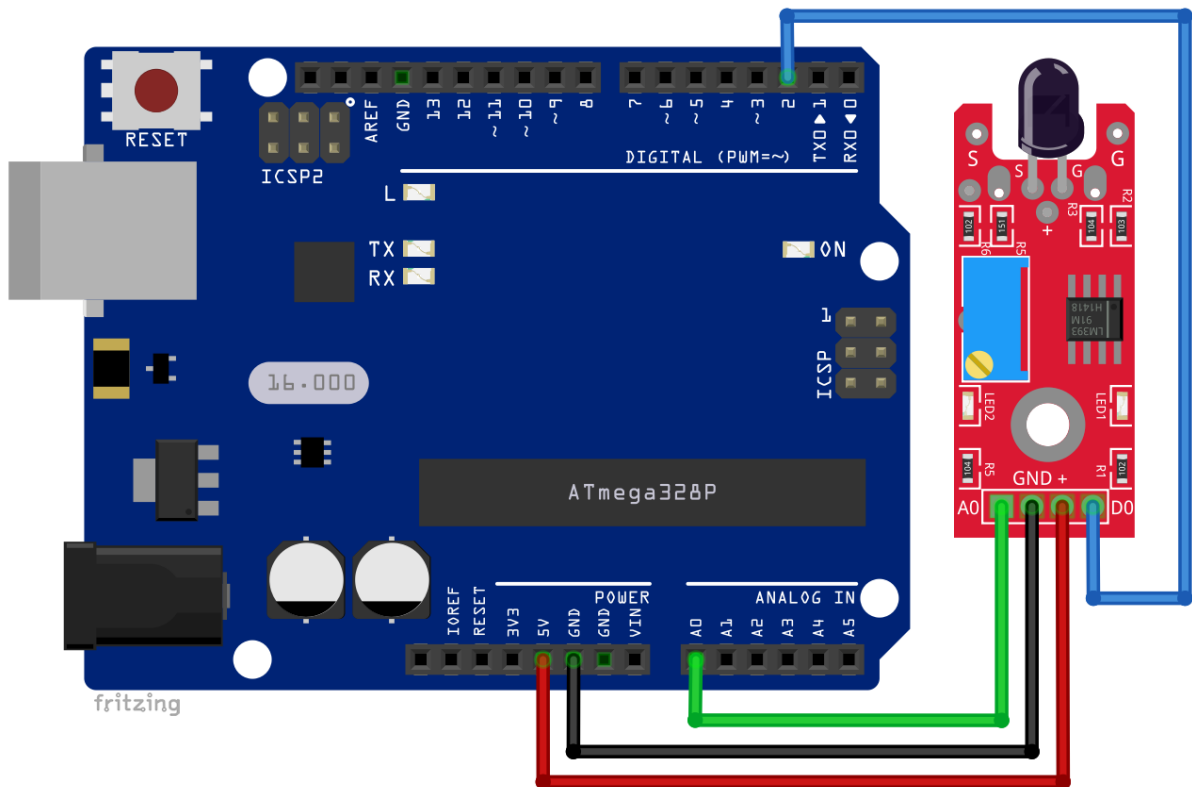
The pinout

The KY-026 reed sensor module has four pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-026 module with the Uno as shown on the following connection diagram:



KY-026 pin	Uno pin	Wire color
+	5V	Red wire
GND	GND	Black wire
A0	A0	Green wire
D0	D2	Blue wire

With this set-up, both analog and digital pins of the KY-026 module are used.

Az-Delivery

Sketch example

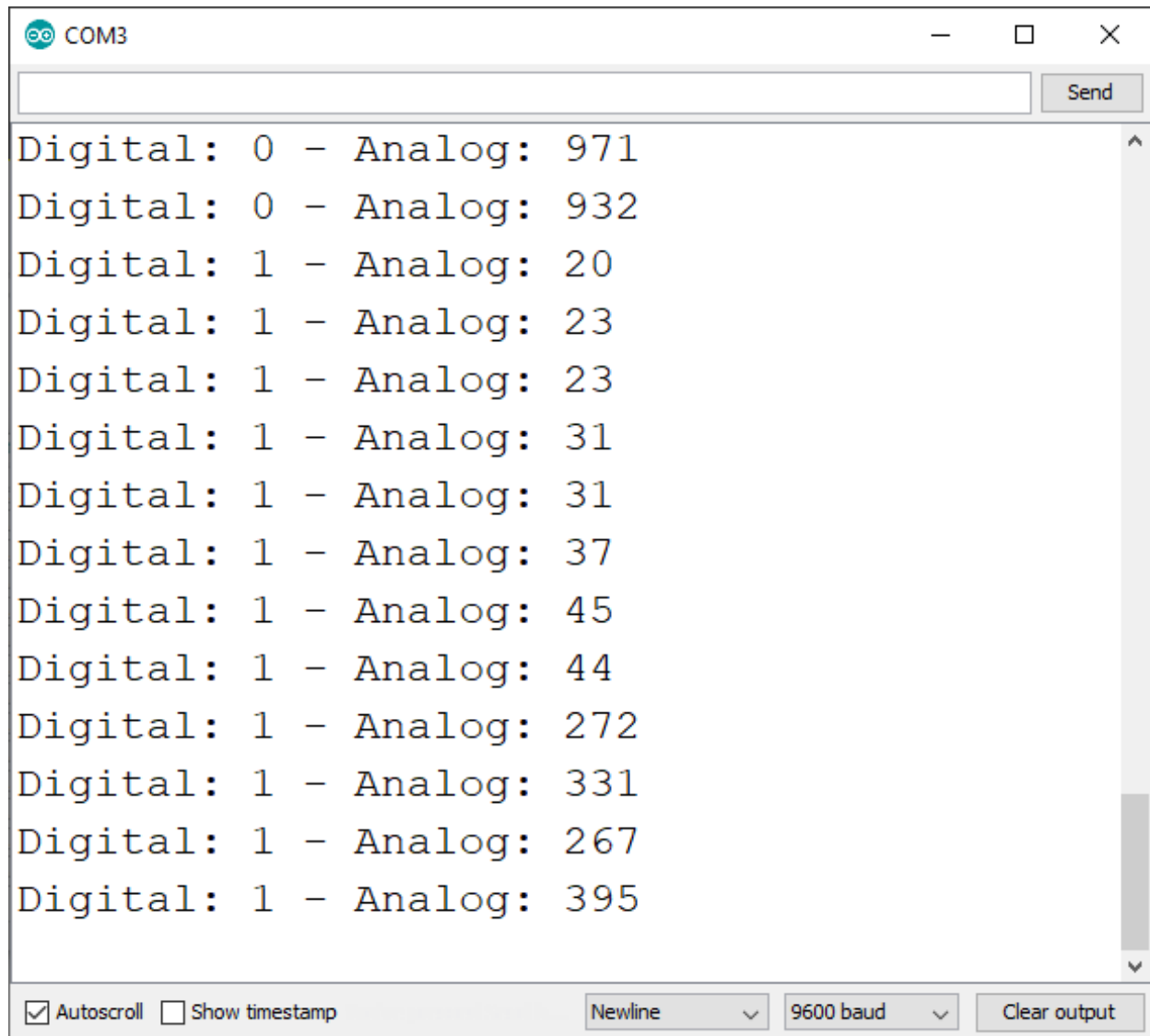
```
#define FLAME_D_PIN 2
#define FLAME_A_PIN 0

void setup() {
  pinMode(FLAME_D_PIN, INPUT);
  Serial.begin(9600);
}

void loop() {
  Serial.print("Digital: ");
  Serial.print(digitalRead(FLAME_D_PIN));
  Serial.print(" - Analog: ");
  Serial.println(analogRead(FLAME_A_PIN));
  delay(1000);
}
```

Az-Delivery

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:

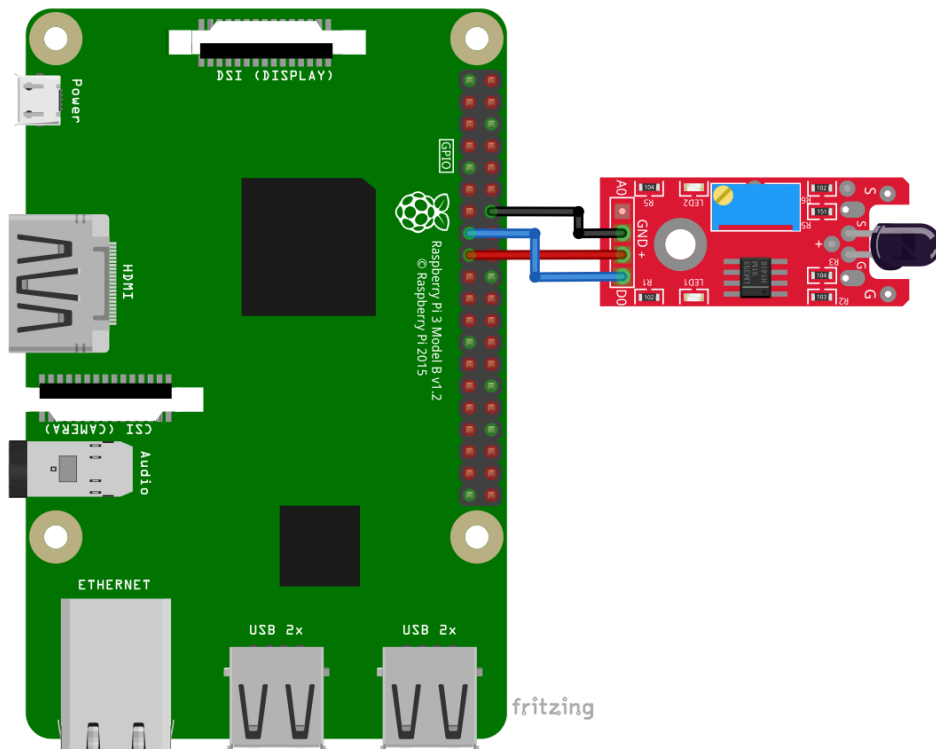


The screenshot shows the Serial Monitor window for COM3. It displays a series of 15 lines of output, each containing a digital reading followed by an analog reading, separated by a hyphen. The digital readings are 0 for the first two lines and 1 for the remaining thirteen lines. The analog readings vary across all lines. The window includes a 'Send' button at the top right and a status bar at the bottom with checkboxes for 'Autoscroll' (checked) and 'Show timestamp' (unchecked), along with dropdown menus for 'Newline' and '9600 baud', and a 'Clear output' button.

Digital	Analog
0	971
0	932
1	20
1	23
1	23
1	31
1	31
1	37
1	45
1	44
1	272
1	331
1	267
1	395

Connecting the module with Raspberry Pi

Connect the KY-026 module with the Raspberry Pi as shown on the following connection diagram:



KY-026 pin	Raspberry Pi pin	Physical pin	Wire color
GND	GND	14	Black wire
D0	GPIO22	15	Blue wire
+ (VCC)	3V3	17	Red wire

With this set-up, the Raspberry Pi reads only the digital pin of the KY-026 module. If the analog pin has to be used, use the analog to digital converter of the Uno or external analog to digital converter like the *ADS1115* (which is not covered in this eBook).

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep

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

Digital_PIN = 22
GPIO.setup(Digital_PIN, GPIO.IN)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        if GPIO.input(Digital_PIN):
            print('Flame detected')
        else:
            print('No flame')

        sleep(1)

except KeyboardInterrupt:
    print('\nScript end!')

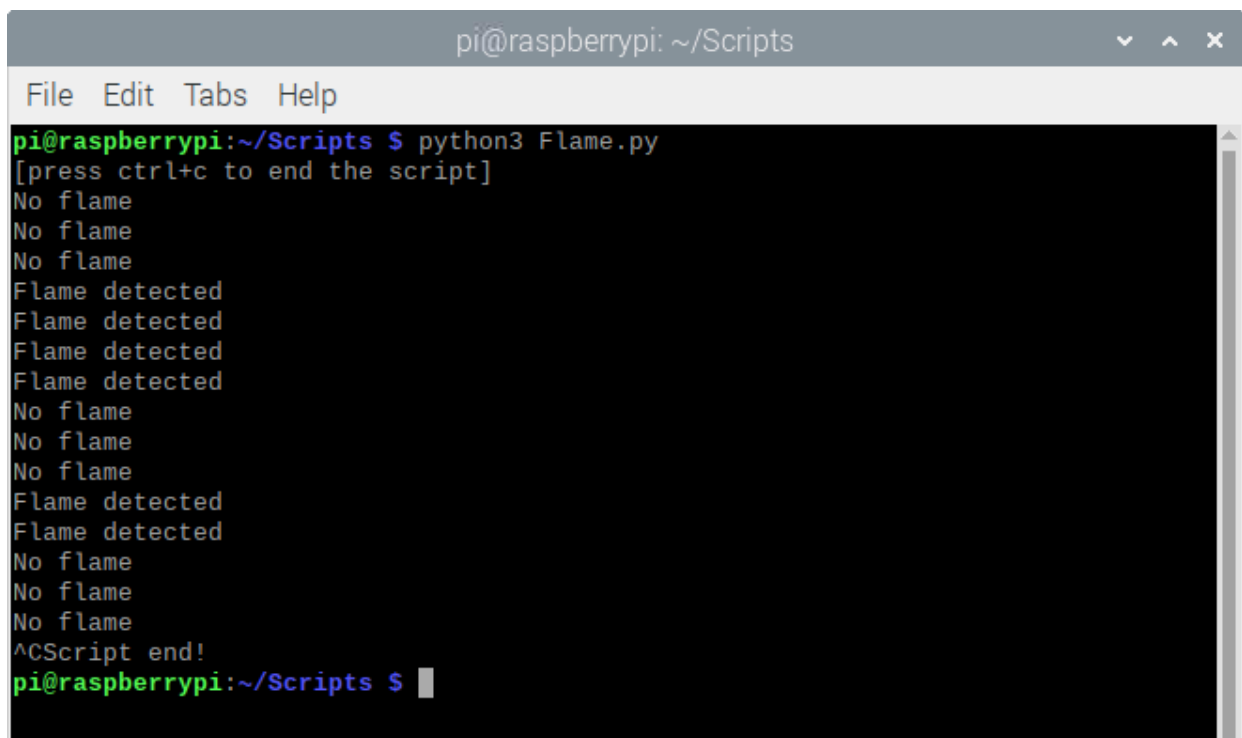
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *Flame.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 Flame.py

The result should look like the output on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 Flame.py
[press ctrl+c to end the script]
No flame
No flame
No flame
Flame detected
Flame detected
Flame detected
Flame detected
No flame
No flame
No flame
Flame detected
Flame detected
No flame
No flame
No flame
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL + C* on the keyboard.

KY-028 Thermistor Sensor Module

Introduction

The thermistor is a temperature dependent resistor, which changes its resistance with temperature change.

The thermistor inside the KY-028 module is *10k* thermistor. It is a resistor with a Negative Temperature Coefficient or for short an *NTC* resistor. There is also a thermistor with Positive Temperature Coefficient, but it is not covered in this eBook.

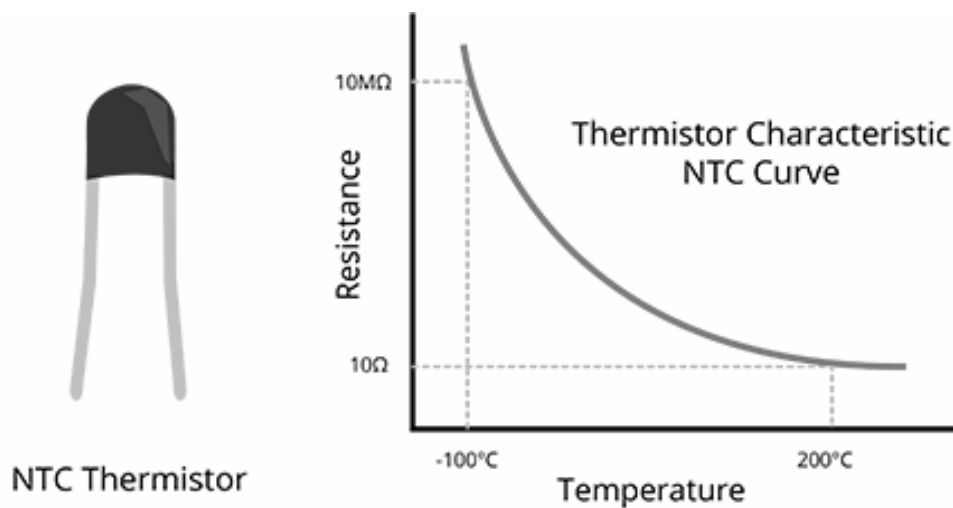
What NTC means is that the resistance of the thermistor decreases with the increase of temperature, and increases with temperature decrease. Thermistors are made so that the resistance changes drastically with temperature, so it can be 100Ω or more of change per temperature degree.

At room temperature (25°C) the resistance of a *10k* thermistor is $10k\Omega$. The *10k* thermistor can be used to accurately measure temperatures in a range from -55°C to 125°C .

Specifications

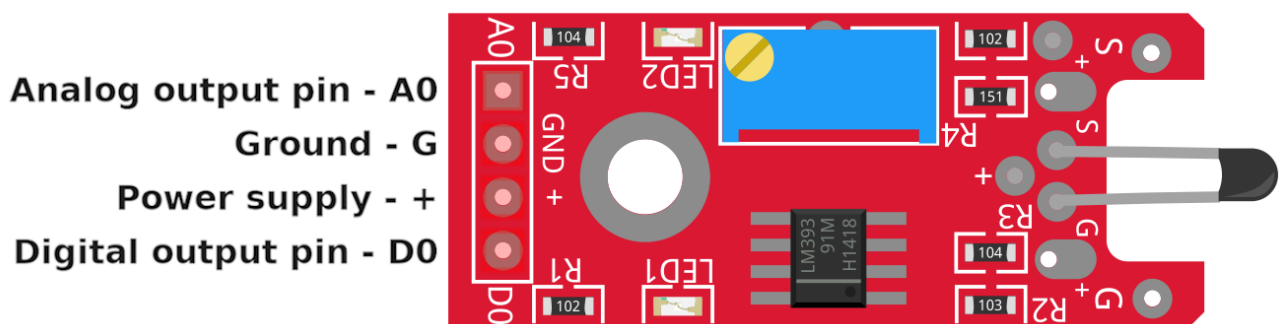
Operating voltage range	from 3.3V to 5.5V
Temperature range	from -55°C to 125°C [-67°F to 257°F]
Accuracy	±0.5°C
Thermal time constant	15 seconds
Resistance at 25°C	10K ±1%
B 25/50	3950 ±1%
Dimensions	15x36mm [0.6x1.4in]

The Resistance/Temperature curve of the thermistor is not linear, it have to be taken into account when the temperature is calculated from the resistance.



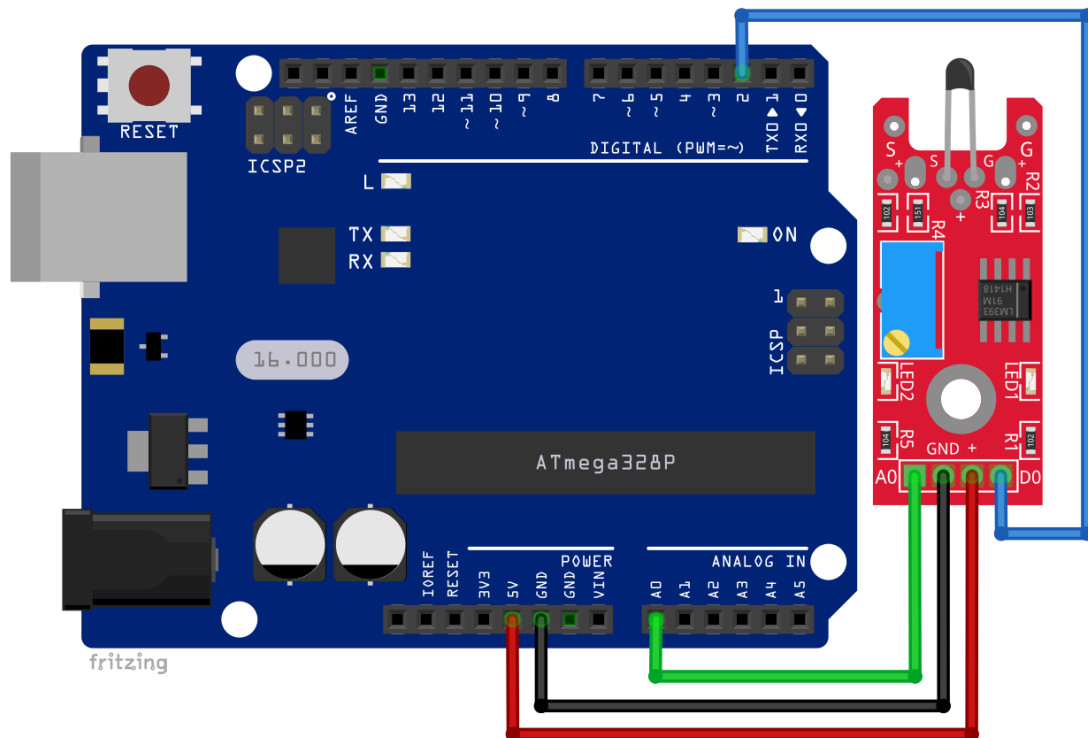
The pinout

The KY-028 thermistor module has four pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-028 module with the Uno as shown on the following connection diagram:



KY-026 pin	Uno pin	Wire color
+	5V	Red wire
GND	GND	Black wire
A0	A0	Green wire
D0	D2	Blue wire

With this set-up, both the analog and digital pins of the KY-028 module are used.

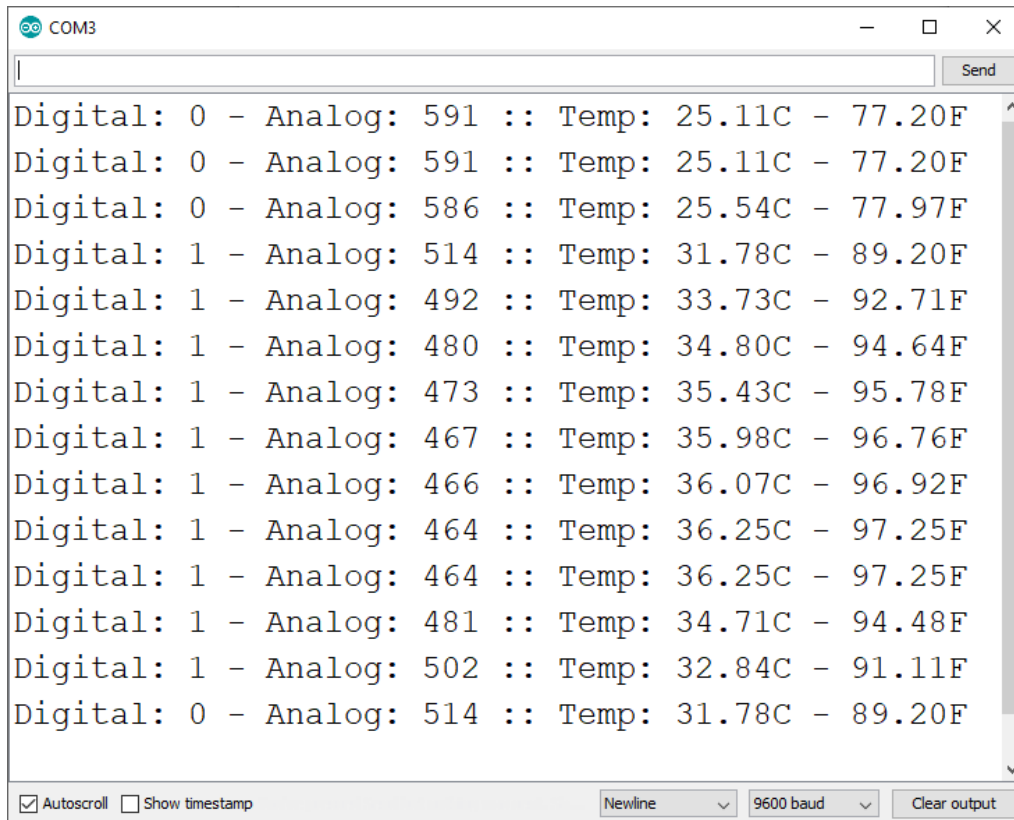
Az-Delivery

Sketch example

```
float error = 7.0;
void setup() {
  pinMode(2, INPUT);
  Serial.begin(9600);
}
void loop() {
  Serial.print("Digital: ");
  Serial.print(digitalRead(2));
  Serial.print(" - Analog: ");
  uint8_t adc = analogRead(0);
  Serial.print(adc);
  float tempC = Thermistor(adc);
  Serial.print(" :: Temp: ");
  Serial.print(tempC);  Serial.print("C - ");
  Serial.print((tempC * 9.0) / 5.0 + 32.0);
  Serial.println("F");  delay(1000);
}
float Thermistor(int adc) {
  float t = 0.0;
  float R2 = 10000.0 / ((1023.0 / adc) - 1.0);
  t = R2 / 10000.0;
  t = log(t) / 3950;
  t = t + 1.0 / (25.0 + 273.15);
  t = (1.0 / t) - 273.15; // In Celsius
  return t + error;
}
```

Az-Delivery

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



The screenshot shows the Serial Monitor window for COM3. It displays a series of data lines. Each line contains three parts: a digital status, an analog value, and a temperature reading in both Celsius and Fahrenheit. The digital status is 0 for temperatures below 32°C and 1 for temperatures above 32°C. The analog value fluctuates between 464 and 591. The temperature starts at 25.11°C (77.20°F) and rises to 36.25°C (97.25°F) before dropping back to 31.78°C (89.20°F).

Digital	Analog	Temp (C)	Temp (F)
0	591	25.11	77.20
0	591	25.11	77.20
0	586	25.54	77.97
1	514	31.78	89.20
1	492	33.73	92.71
1	480	34.80	94.64
1	473	35.43	95.78
1	467	35.98	96.76
1	466	36.07	96.92
1	464	36.25	97.25
1	464	36.25	97.25
1	481	34.71	94.48
1	502	32.84	91.11
0	514	31.78	89.20

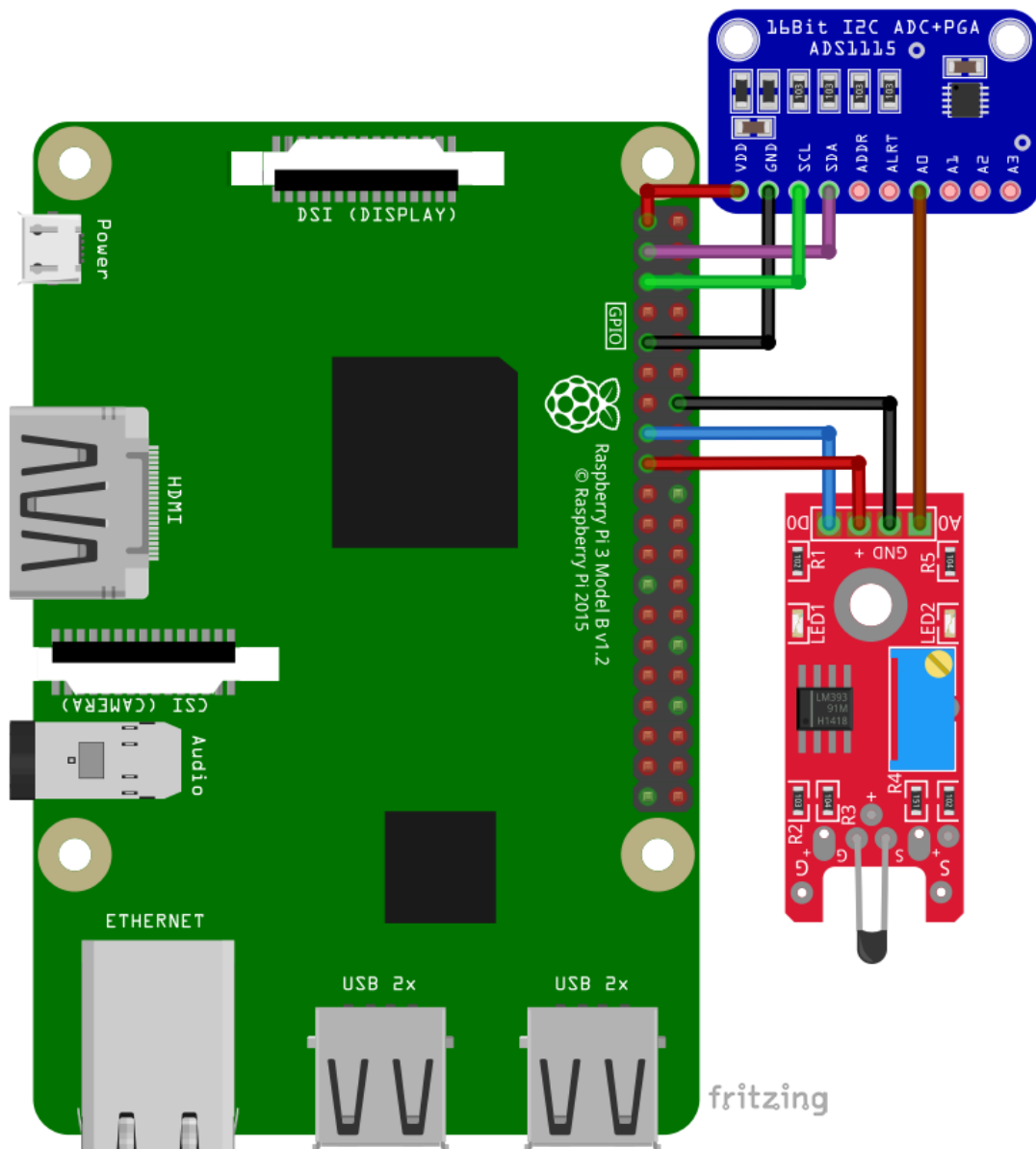
If the thermistor on-board the KY-028 module is touched with human fingers, the temperature output changes like on the image above.

At the end of the *Thermistor()* function add the error value of "7". The error value was determined by observing and comparing two outputs, first from the KY-028 module and second from the KY-001 module.

With this set-up, digital output pin outputs *HIGH* state (digital value 1) when the temperature rises above 32°C.

Connecting the module with Raspberry Pi

Connect the KY-028 module with the Raspberry Pi as shown on the following connection diagram:



KY-026 pin	Raspberry Pi pin	Physical pin	Wire color
GND	GND	14	Black wire
D0	GPIO22	15	Blue wire
+ (VCC)	3V3	17	Red wire
KY-026 pin	ADS1115 pin		
A0	A0		Brown wire
ADS1115 pin	Raspberry Pi pin	Physical pin	Wire color
+ (VCC)	3V3	1	Red wire
GND	GND	9	Black wire
SCL	GPIO3	5	Green wire
SDA	GPIO2	3	Purple wire

With this set-up, both the analog and digital pins of the KY-028 module are used. Here, the device called *ADS1115* is used as an external analog to digital converter (ADC), where the analog data is sent to the Raspberry Pi via the I2C interface (as it can be seen in the chapter *External ADC for the Raspberry Pi*).

Az-Delivery

Python script

```
import time
import math
import Adafruit_ADS1x15
ADS1115 = Adafruit_ADS1x15.ADS1115()
GAIN = 1
ds = u'\xb0' # degree sign
R1 = 10000.0 # Predefine resistance of NTC Thermistor
R2 = 0.0 # Calibration resistance of NTC Thermistor in Ohms
tempC = 0.0 # temperature in Celsius
tempF = 0.0 # temperature in Fahrenheit
B = 3950 # 10kOhm NTC Thermistor B factor
Room_Temp = 25.0 # Room temperature in Celsius
azt = -273.16 # Absolute Zero Temperature in Celsius
error = -3.0
print('[Press CTRL + C to end the script!]\n')
try: # Main program loop
    while True:
        adc = ADS1115.read_adc(0, gain=GAIN) # Read ADC channel 0
        R2 = (R1 / ((32767.0 / adc) - 1.0)) / 10000.0
        tempC = (math.log(R2) / B) + 1.0 / (Room_Temp - azt)
        tempC = (1.0 / tempC) + azt + error
        tempF = (tempC * 9.0) / 5.0 + 32.0
        print('Temp: {:.4f}{}C - {:.4f}{}F'.format(tempC, ds, tempF, ds))
        time.sleep(1)

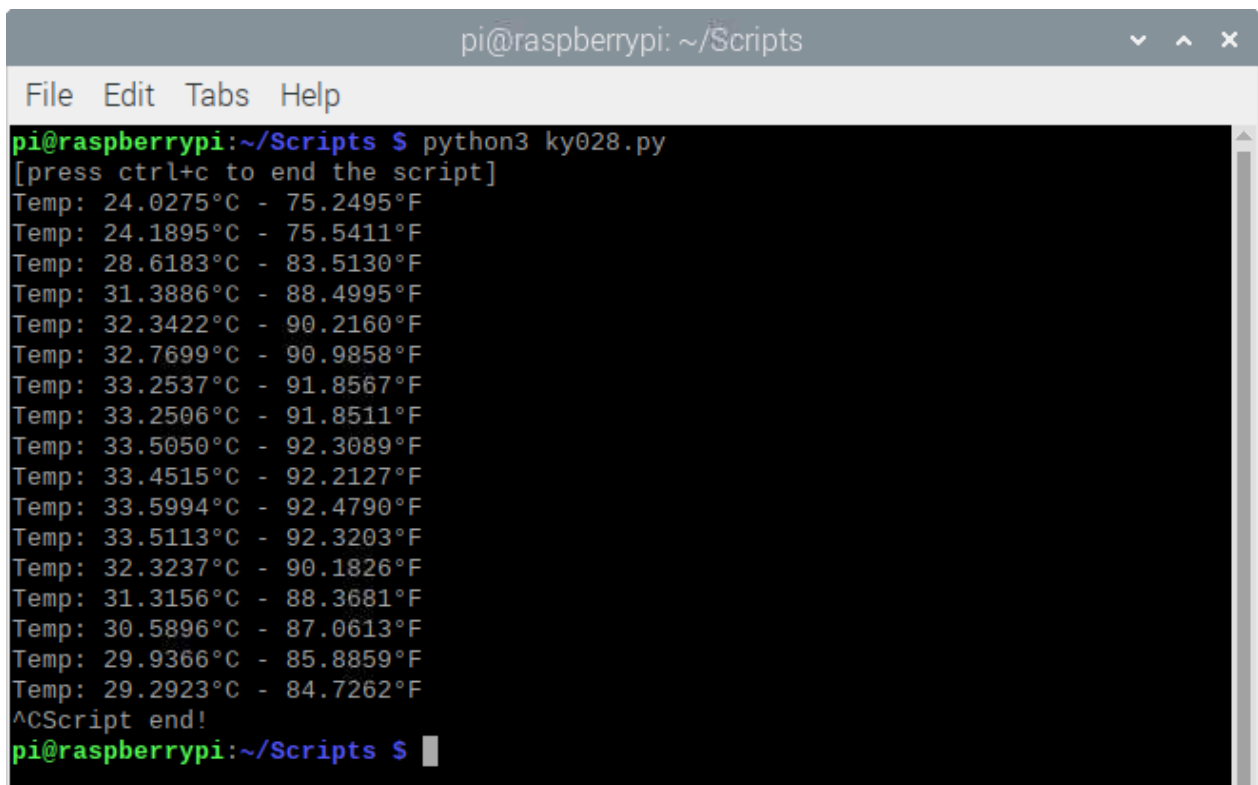
except KeyboardInterrupt:
    print('\nScript end!')
```


Az-Delivery

Save the script by the name *ky028.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 ky028.py

The result should look like the output on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 ky028.py
[press ctrl+c to end the script]
Temp: 24.0275°C - 75.2495°F
Temp: 24.1895°C - 75.5411°F
Temp: 28.6183°C - 83.5130°F
Temp: 31.3886°C - 88.4995°F
Temp: 32.3422°C - 90.2160°F
Temp: 32.7699°C - 90.9858°F
Temp: 33.2537°C - 91.8567°F
Temp: 33.2506°C - 91.8511°F
Temp: 33.5050°C - 92.3089°F
Temp: 33.4515°C - 92.2127°F
Temp: 33.5994°C - 92.4790°F
Temp: 33.5113°C - 92.3203°F
Temp: 32.3237°C - 90.1826°F
Temp: 31.3156°C - 88.3681°F
Temp: 30.5896°C - 87.0613°F
Temp: 29.9366°C - 85.8859°F
Temp: 29.2923°C - 84.7262°F
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL + C* on the keyboard.

For this set-up, the error value is -3.0. The error value was determined by observing and comparing two outputs, first from the KY-028 module and second from the KY-001 module.

KY-029 Bi-color 3mm LED Module

Introduction

The LED on-board the KY-029 module is two LEDs in one (red and green LEDs together). Each color can be turned *ON* separately, or both colors can be turned *ON* at the same time. The brightness level of each color can be controlled individually (or both at the same time), with PWM (Pulse Width Modulation), which results in new colors.

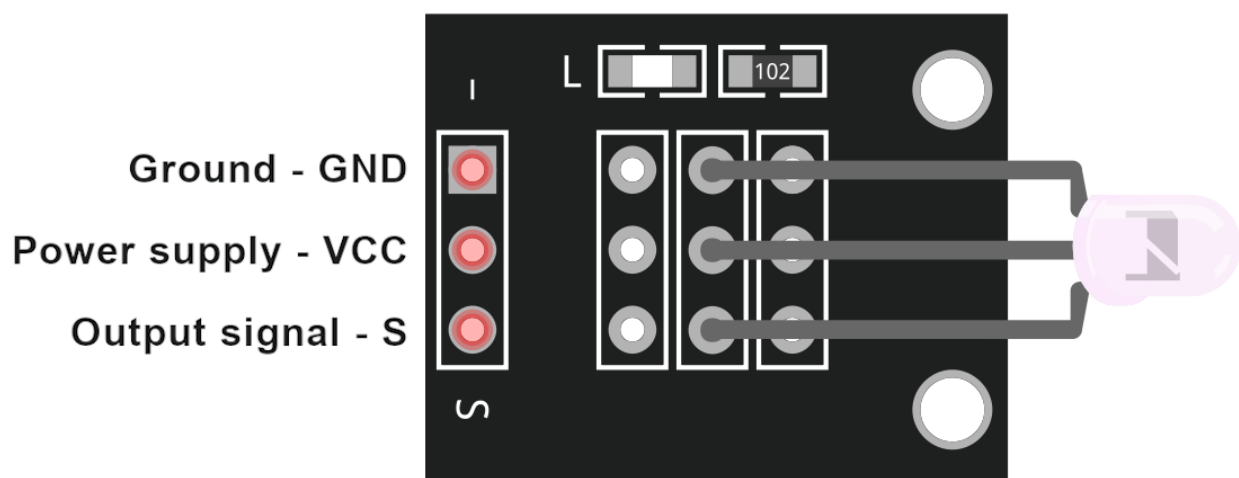
The KY-029 module has 3 pins, two for the red and green light and the third is the ground pin. Each color pin on the module has one resistor in series with color pin, which are used for current flow limitation (for LED protection).

Specifications

Operating voltage range	from 3.3V to 5V DC
Operating temperature	-25°C to 80°C [-13°F to 176°F]
Dimensions	19x15mm [0.73x0.6in]

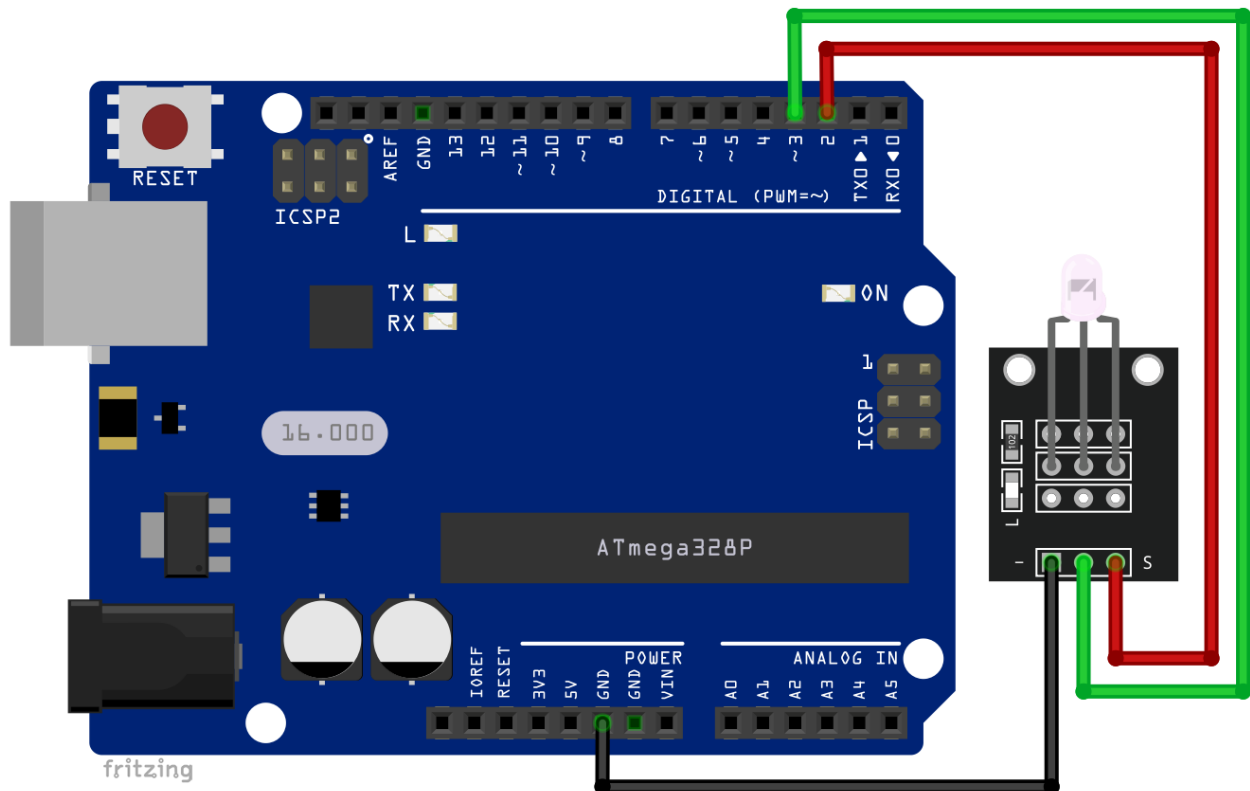
The pinout

The KY-029 bi-color 3mm LED module has three pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-029 module with the Uno as shown on the following connection diagram:



KY-029 pin	Uno pin	Wire color
S (Red)	D2	Red wire
Middle pin	D3	Green wire
- (GND)	GND	Black wire

Az-Delivery

Sketch example

```
#define RED_PIN 2
#define GREEN_PIN 3

void setup() {
  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
}

void loop() {
  analogWrite(RED_PIN, 250);
  analogWrite(GREEN_PIN, 0);
  delay(1000);

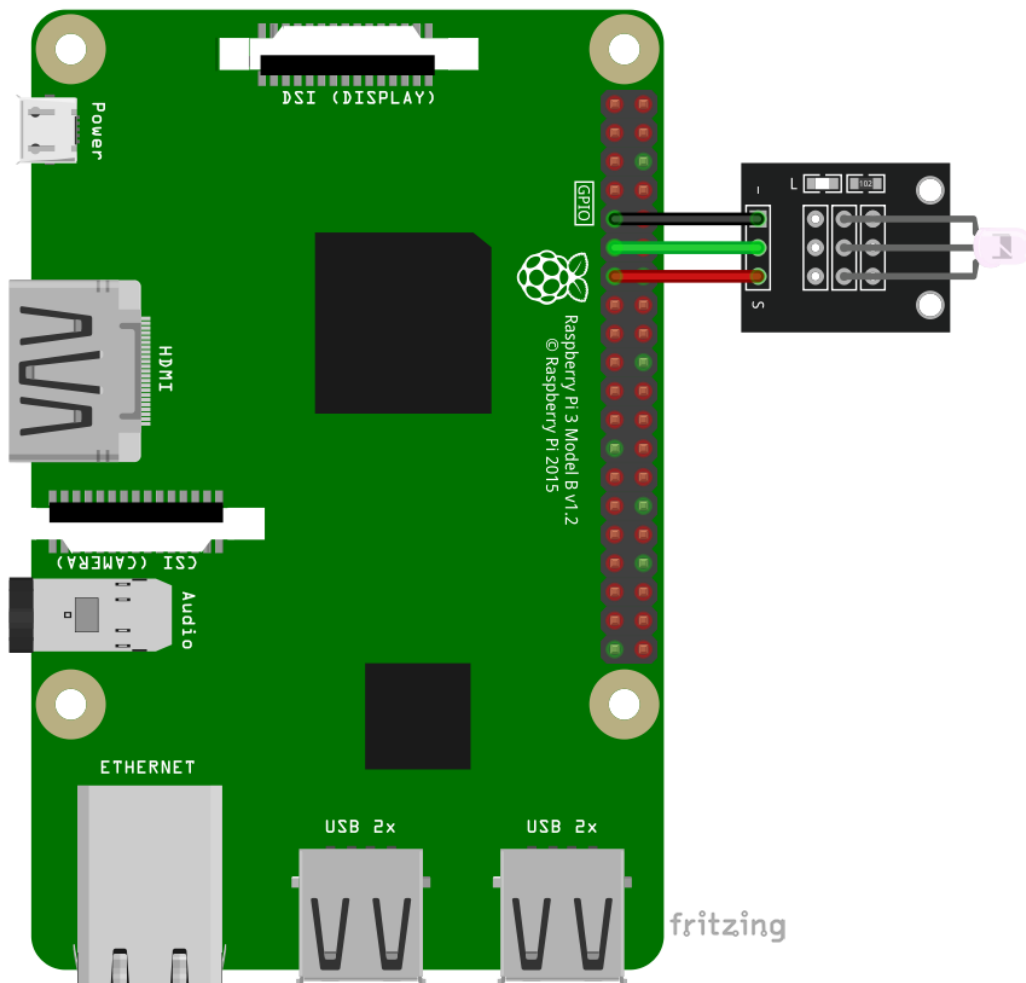
  analogWrite(RED_PIN, 255);
  analogWrite(GREEN_PIN, 100);
  delay(1000);

  digitalWrite(RED_PIN, HIGH);
  digitalWrite(GREEN_PIN, LOW);
  delay(1000);

  digitalWrite(RED_PIN, LOW);
  digitalWrite(GREEN_PIN, HIGH);
  delay(1000);
}
```

Connecting the module with Raspberry Pi

Connect the KY-029 module with the Raspberry Pi as shown on the following connection diagram:



KY-029 pin	Raspberry Pi pin	Physical pin	Wire color
- (GND)	GND	9	Black wire
Middle pin	GPIO17	11	Green wire
S	GPIO27	13	Red wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)

Red_PIN = 13
Green_PIN = 19
GPIO.setup(Red_PIN, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Green_PIN, GPIO.OUT, initial=GPIO.LOW)
red = GPIO.PWM(Red_PIN, 100) # PWM, frequency 100Hz
green = GPIO.PWM(Green_PIN, 100) # PWM, frequency 100Hz
red.start(1)
green.start(1)
print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        red.ChangeDutyCycle(90)
        green.ChangeDutyCycle(1)
        sleep(1) # Wait for a second
        red.ChangeDutyCycle(99)
        green.ChangeDutyCycle(10)
        sleep(1)

except KeyboardInterrupt:
    print('\nScript end!\n')

finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *TwoColorLED.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

```
python3 TwoColorLED.py
```

And watch the color changing magic.

When starting the PWM, with the following line of code:

```
red.start(1)
```

use any number different than 0, because if the number 0 is used, that will disable PWM on the selected GPIO pin.

Also, when using the following line of code:

```
red.ChangeDutyCycle(1)
```

to change the duty cycle of the PWM signal, do not use number 0, because of the same reason as for the *red.start(1)*.

To stop the script press *CTRL + C* on the keyboard.

KY-031 Hit-Knock Sensor Module

Introduction

The KY-031 hit-knock sensor module is a sensor that can detect knocking or hits. The module has three pins: signal output pin, power supply or *VCC* pin and the ground or *GND* pin. The sensor on-board the KY-031 module is made of a spring that is in the transparent plastic casing. On one side, spring is connected to the *VCC* pin. Between *VCC* pin and the spring, there is a resistor. The other side of spring hangs freely near the metal connector. If the sensor is being hit, the spring swings and it connects with the metal connector. This closes the circuit, which results in a short voltage pulse at the signal pin, which then can be picked up by a microcontroller as a trigger event.

When the KY-031 module is not detecting a knock or a hit, the output on the signal pin is in the *HIGH* state (default state). But when it detects a hit, the output on the signal pin is in the *LOW* state for a short period of time.

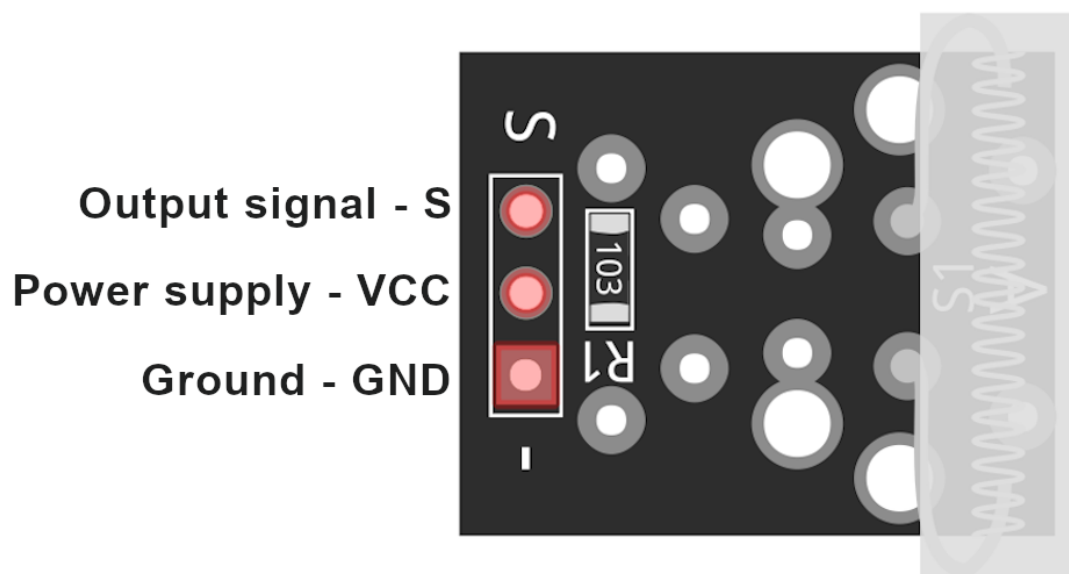
Specifications

Operating voltage range	from 3.3V to 5V DC
Default output	HIGH (default state)
Dimensions	19x15x10mm [0.73x0.6x0.4in]

In order for a sensor to detect a knock or a hit, it has to be hit a little bit harder. Do not overdo it, because this could damage the plastic casing of the sensor.

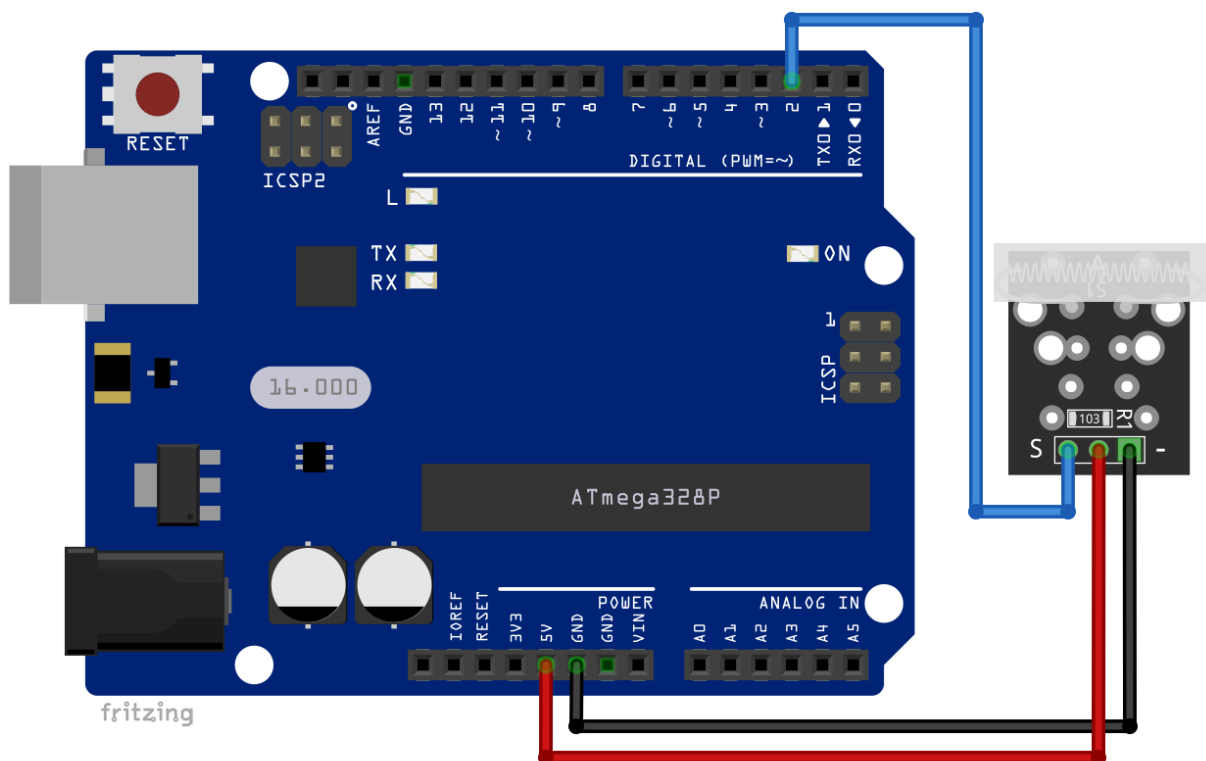
The pinout

The KY-031 hit-knock sensor module has three pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-031 module with the Uno as shown on the following connection diagram:



KY-031 pin	Uno pin	Wire color
D0	D2	Blue wire
Middle pin (VCC)	5V	Red wire
GND	GND	Black wire

NOTE: D2 (external interrupt pin, MUST!)

Az-Delivery

Sketch example

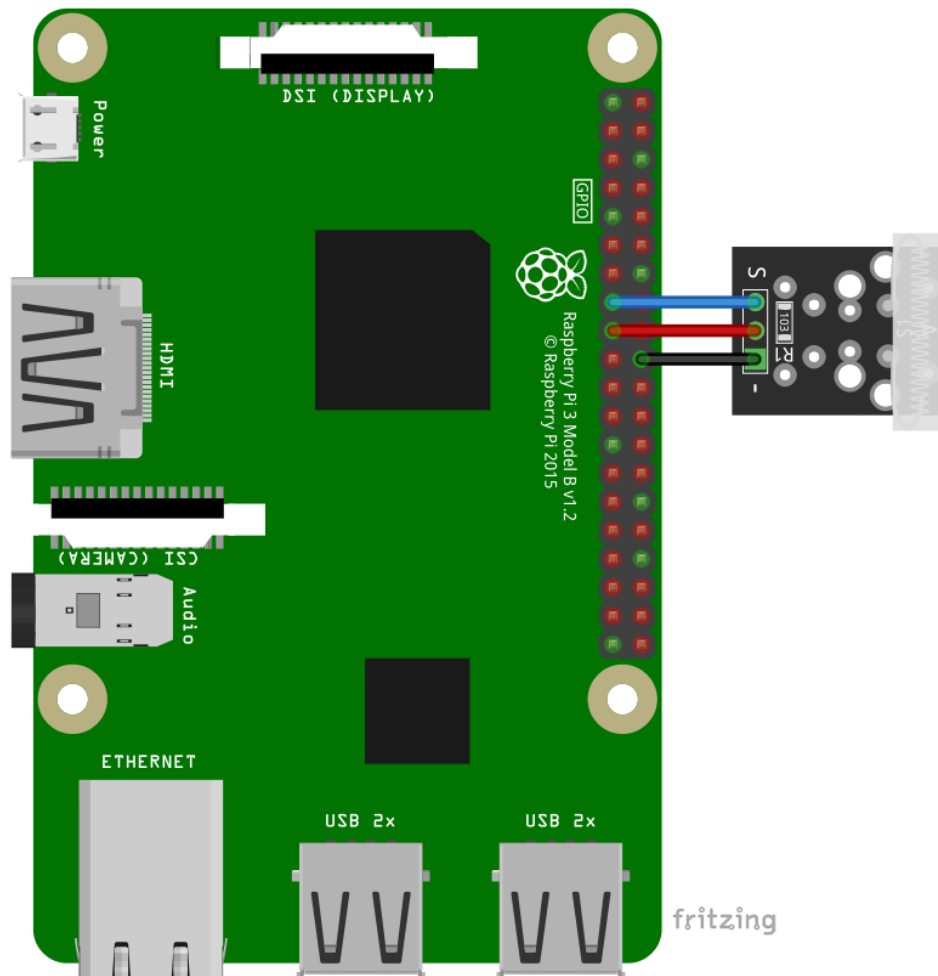
```
#define HIT_PIN 2 // external interrupt pin
#define LED_PIN 13
volatile byte state = 0;
void setup() {
    pinMode(LED_PIN, OUTPUT);
    pinMode(HIT_PIN, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(HIT_PIN), hit_led,
FALLING);
}
void loop() {
    digitalWrite(LED_PIN, state);
    delay(100);
}
void hit_led() {
    state = !state; // toggling the led state
}
```

In the sketch, the external interrupt on digital pin 2 is used, to change the state of on-board LED of the Uno. You have to use this pin because external interrupt feature is available only on digital pins 2 or 3.

Upload the sketch to the Uno, the on-board LED connected to digital pin 13 should toggle when the KY-031 module is hit.

Connecting the module with Raspberry Pi

Connect the KY-031 module with the Raspberry Pi as shown on the following connection diagram:



KY-031 pin	Raspberry Pi pin	Physical pin	Wire color
S	GPIO22	15	Blue wire
Middle pin (VCC)	3V3	17	Red wire
GND	GND	20	Black wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep

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

Signal_PIN = 22
GPIO.setup(Signal_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def hit(channel):
    print('Hit detected!')

GPIO.add_event_detect(22, GPIO.FALLING, callback=hit, bouncetime=100)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        sleep(0.0001)

except KeyboardInterrupt:
    print('\nScript end!\n')

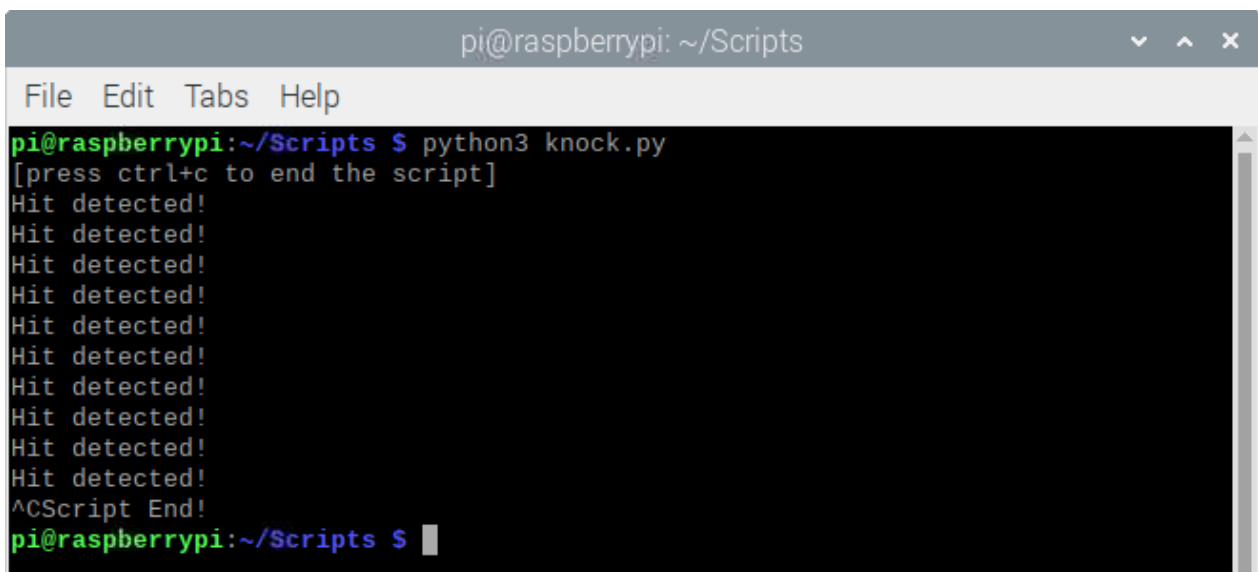
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *knock.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 knock.py

The result should look like the output on the following image:

A screenshot of a terminal window on a Raspberry Pi. The window title is 'pi@raspberrypi: ~/Scripts'. The menu bar shows 'File', 'Edit', 'Tabs', and 'Help'. The terminal content shows the command 'python3 knock.py' being executed. The output consists of ten 'Hit detected!' messages, followed by '^CScript End!'. The prompt 'pi@raspberrypi:~/Scripts \$' is visible at the bottom with a cursor.

```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 knock.py
[press ctrl+c to end the script]
Hit detected!
Hit detected!
Hit detected!
Hit detected!
Hit detected!
Hit detected!
Hit detected!
Hit detected!
Hit detected!
Hit detected!
^CScript End!
pi@raspberrypi:~/Scripts $
```

To end the script press *CTRL* + *C* on the keyboard.

Az-Delivery

The script starts by importing two libraries, *RPi.GPIO* and *time*. Then the GPIO pin names are set to *BCM* and all warnings regarding GPIO interfaces are disabled.

Next, the pin mode of the GPIO pin 22 is set to input.

After that, the function called *hit()* is created. The function has one argument and returns no value. The argument is not used, so it will not be explained here. The function is executed on the digital signal change on the hit pin. In the function, the *print()* function with message *Hit detected* is used:

```
def hit(channel):  
    print('Hit detected!')
```

After this function, the interrupt routine is created with the following line of code:

```
GPIO.add_event_detect(22,GPIO.BOTH,callback=hit,bouncetime=100)
```

Where number 22 represents the hit pin; *GPIO.BOTH* represents on which edge of the digital signal the *hit()* function is executed, in this case on the both signal changes (falling and rising); *callback=hit* sets which function is executed on the signal change; *bouncetime=100* represents how long is the pause between the trigger (digital signal change) and the beginning of the function execution, in this case it is 100 milliseconds. With this pause the bouncing part of the signal is skipped and only the stable state of the digital signal is read (after the bouncing).

Az-Delivery

Next, the *try-except-finally* block of code is created. In the *try* block an indefinite loop is created (*while True:*). In the indefinite loop there is one pause of *100* microseconds so that the loop itself does not block the Raspberry Pi.

The *except* block of code is executed when *CTRL + C* on the keyboard is pressed. This is called the keyboard interrupt. When the *except* block of code is executed, the message *Script end!* is displayed in the terminal.

The *finally* block of code executes when the script ends, and when this happens, a function called *cleanup()* is executed. This function is used to release any previously used GPIO pins or interfaces.



KY-032 Obstacle Detector Module

Introduction

At the heart of the KY-032 module is a *NE555* timer oscillator chip configured to generate a *38kHz* square wave. The *38kHz* signal is used to turn *ON* an infrared LED at that frequency. Light sent by infrared LED gets reflected from the obstacle and then detected by an *HS0038B* infrared receiver module. The receiver module incorporates an external optical *950nm* infrared filter and an internal electronic *38kHz* band pass filter that makes the module receptive only to infrared light pulsing at that frequency.

Specifications

Operating voltage range	from 3.3V to 5V DC
Working current	~20mA
Detection distance	from 20mm to 400mm
Detection angle	60°
Dimensions	27x45mm [1.1x1.8in]
Output	LOW - obstacle detected HIGH - obstacle is not detected

The potentiometer near the *NE555* chip (*R6* on the schematic diagram) is used to fine tune the signal to exactly *38kHz*. The potentiometer near the jumper (*R5* on the schematic diagram) adjusts the duty cycle of the signal, which controls the brightness of the infrared LED, or the sensitivity of the module (the distance on which can detect an obstacle).

Both adjustments together affect the range of the device. Proper adjustment of *R6* potentiometer requires an oscilloscope or frequency counter, but it can be left centered or in place as is.

The receiver module also includes an automatic gain control, that suppresses a continuous signal of any frequency, including *38kHz*. Therefore, it is absolutely necessary to use the *EN* pin for proper operation of the device. If the *Enable* function is used correctly, the device achieves its maximum sensitivity.

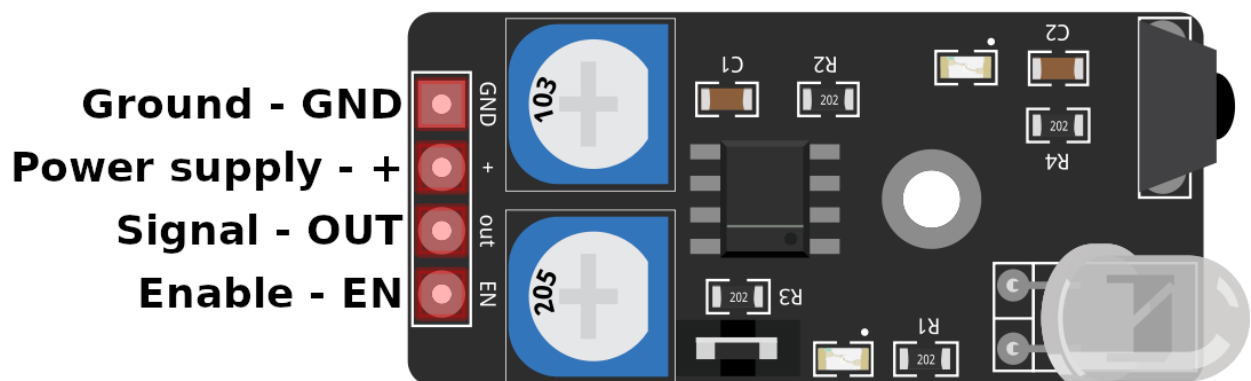
Az-Delivery

When the jumper is installed on the board, the infrared LED is turned *ON* continuously at a frequency of *38kHz*. If the *Enable* function is not used, the jumper must be installed. When the jumper is removed, pin 4 of the *NE555* chip is held *LOW* (*RESET* state) by a *22kΩ* pull-down resistor (*R3* on the schematic diagram). Then, if a *HIGH* state is applied at the *EN* pin, the *RESET* state is relieved and the *NE555* chip begins to oscillate.

The *Enable* function can not be used if the jumper is in place. The jumper must be removed from the module board in order to use the *EN* pin.

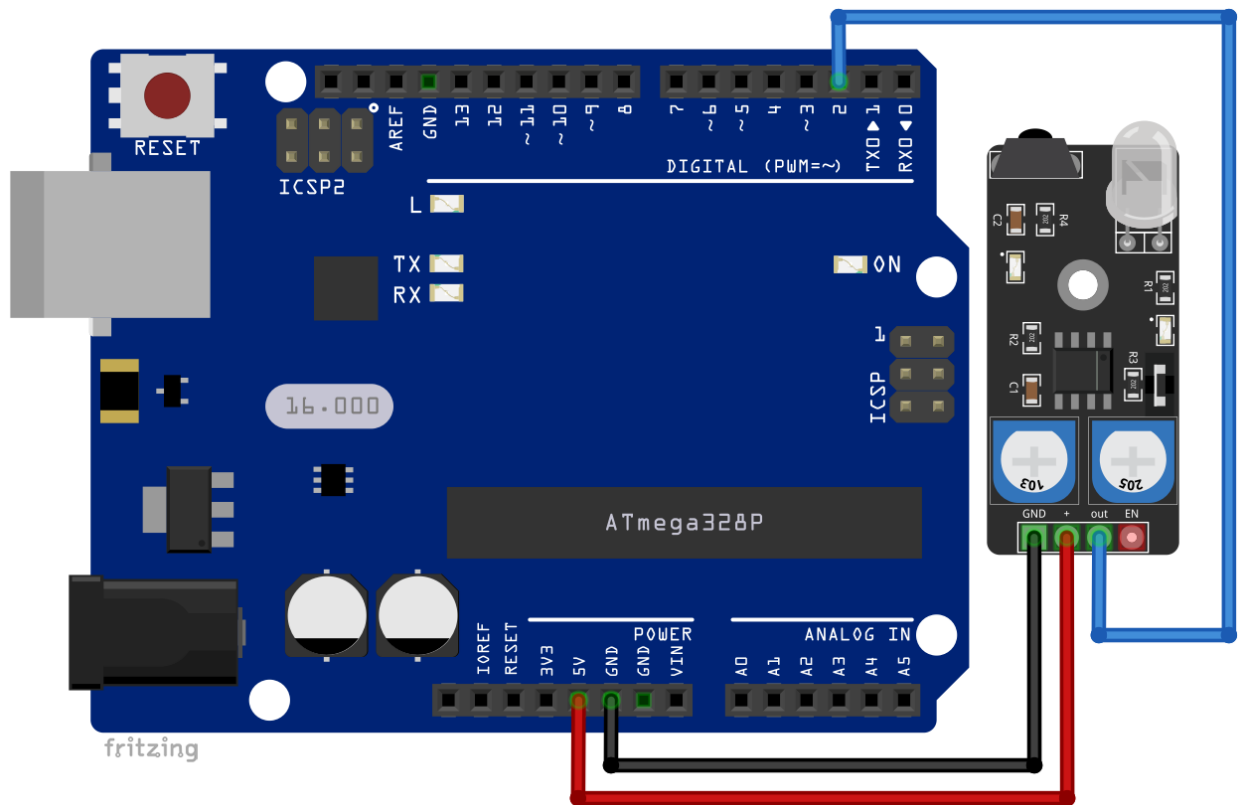
The pinout

The KY-032 obstacle detector module has four pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-032 module with the Uno as shown on the following connection diagram:



KY-032 pin	Uno pin	Wire color
GND	GND	Black wire
VCC	5V	Red wire
OUT	D2	Blue wire

NOTE: Do not remove the jumper from the KY-032 module, because the *ENABLE* mode is not used in the sketch.

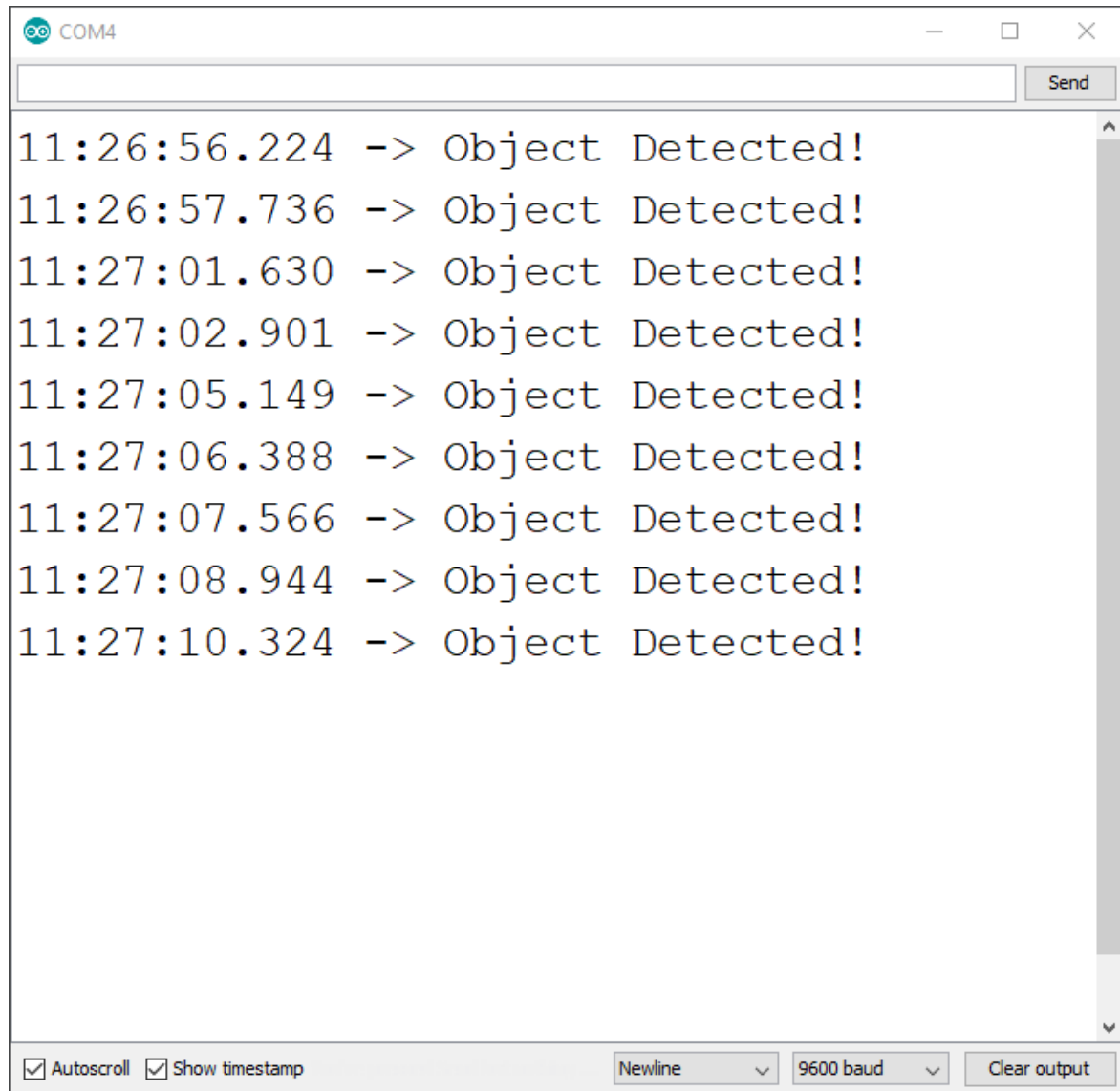
Az-Delivery

Sketch example

```
#define SIGNAL_PIN 2
bool object_detect = false;
void setup() {
    Serial.begin(9600);
    pinMode(SIGNAL_PIN, INPUT);
}
void loop() {
    if(digitalRead(SIGNAL_PIN)) {
        object_detect = false;
    }
    else {
        delayMicroseconds(500);
        if(digitalRead(SIGNAL_PIN)) {
            object_detect = false;
        }
        else {
            object_detect = true;
        }
    }
    if(object_detect == true) {
        Serial.println("Object Detected!");
        delay(100);
    }
}
```

Az-Delivery

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



The screenshot shows the Serial Monitor window for COM4. The window title is "COM4". At the top, there is a text input field and a "Send" button. The main area displays the following output:

```
11:26:56.224 -> Object Detected!  
11:26:57.736 -> Object Detected!  
11:27:01.630 -> Object Detected!  
11:27:02.901 -> Object Detected!  
11:27:05.149 -> Object Detected!  
11:27:06.388 -> Object Detected!  
11:27:07.566 -> Object Detected!  
11:27:08.944 -> Object Detected!  
11:27:10.324 -> Object Detected!
```

At the bottom, there are checkboxes for "Autoscroll" and "Show timestamp", both of which are checked. To the right of these are dropdown menus for "Newline" and "9600 baud". On the far right is a "Clear output" button.

Az-Delivery

Sketch starts by defining and initializing one macro called *SIGNAL_PIN*. This macro represents the digital pin of Uno on which signal pin of the module is connected.

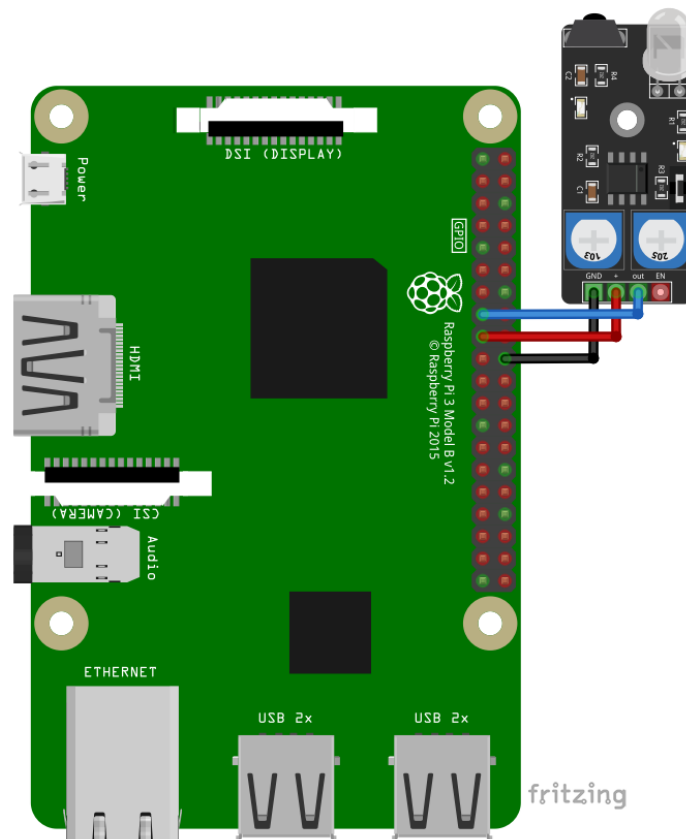
Next, one boolean variable called *object_detect* is defined and initialized with value *false*. The value stored in this variable is used to signalize the object detection in front of the module.

In the *setup()* function, the serial communication is started with a baud rate of 9600bps. Also, the pin mode for the *SIGNAL_PIN* is set to *INPUT*.

In the *loop()* function, the state of the *SIGNAL_PIN* is checked and if there is a *HIGH* signal on this pin, the state of it is checked again. Only if after both readings, the state of the *SIGNAL_PIN* is in *HIGH* state, the value of *object_detect* is changed to *true*. If not, the value of the *object_detect* variable is set to *false*. After this, the value of *object_detect* variable is checked and if the *true* value is stored in it, the message *Object Detected!* is displayed in the Serial Monitor.

Connecting the module with Raspberry Pi

Connect the KY-032 module with the Raspberry Pi as shown on the following connection diagram:



KY-032 pin	Raspberry Pi pin	Physical pin	Wire color
GND	GND	9	Black wire
OUT	GPIO22	15	Blue wire
VCC	3V3	17	Red wire

NOTE: Do not remove the jumper from the KY-032 module, because the *ENABLE* mode is not used in the script.

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import strftime, localtime, sleep
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
Signal_PIN = 22
GPIO.setup(Signal_PIN, GPIO.IN)
object_detect = False
print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        if GPIO.input(Signal_PIN):
            object_detect = False
        else:
            sleep(0.0005) # wait for 500 microseconds
            if GPIO.input(Signal_PIN):
                object_detect = False
            else:
                object_detect = True

        if object_detect:
            print('{} Object detected!'.format(strftime('%H:%M:%S',
                localtime())))
            sleep(0.1) # wait for 100 milliseconds

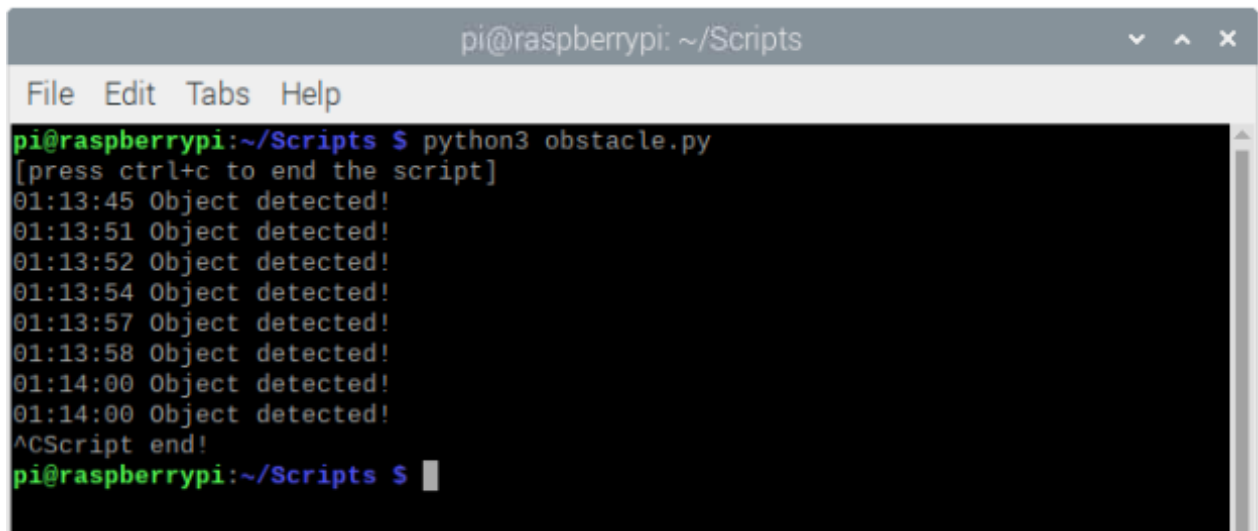
except KeyboardInterrupt:
    print('\nScript end!\n')
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *obstacle.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

```
python3 obstacle.py
```

The result should look like the output on the following image:

A screenshot of a terminal window titled 'pi@raspberrypi: ~/Scripts'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal shows the command 'python3 obstacle.py' being executed. The output consists of several lines of timestamps followed by 'Object detected!':
01:13:45 Object detected!
01:13:51 Object detected!
01:13:52 Object detected!
01:13:54 Object detected!
01:13:57 Object detected!
01:13:58 Object detected!
01:14:00 Object detected!
01:14:00 Object detected!
After these lines, the prompt changes to '^CScript end!'. The user then presses 'Ctrl+C' to end the script, and the prompt returns to 'pi@raspberrypi:~/Scripts \$' with a cursor.

To end the script press *CTRL* + *C* on the keyboard.

This script does the same thing as the sketch from the *Sketch example* chapter. The only difference is that the script is written in *Python* programming language.

KY-033 Line-tracker Sensor Module

Introduction

The KY-033 line-tracker sensor module uses an infrared transmitter and receiver that detects how reflective the surface is. On-board the KY-033 module there are the LM393 dual analog voltage comparator chip, a potentiometer, a status LED, two resistors and TCRT5000 infrared transmitter and receiver. The potentiometer is used to adjust the sensitivity of the module or distance at which it detects the surface. The module is mostly used to detect differences between black and white surfaces.

When it is detecting white surface, *OUT* pin outputs *LOW* signal, and the status LED is turned *ON*. When the module is detecting black surface, *OUT* pin outputs *HIGH* signal and the status LED is turned *OFF*.

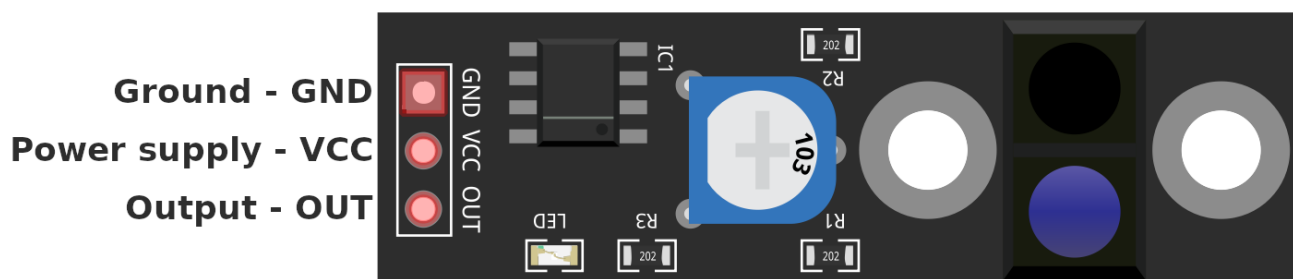
Specifications

Operating voltage range	from 3.3V to 5V DC
Working current	~20mA
Detection distance	from 5mm to 150mm
Detection angle	35°
Dimensions	42x11mm [1.65x0.4in]

The transmitting part of the TCRT5000 emits infrared light at a certain frequency, light bounce back from the surface and gets detected by an infrared light detector. Then the voltage from the TCRT5000 sensor is compared with a *VCC* voltage (in the *LM393* analog comparator) and then the *LM393* comparator outputs the digital *HIGH* or *LOW* signal on the *OUT* pin.

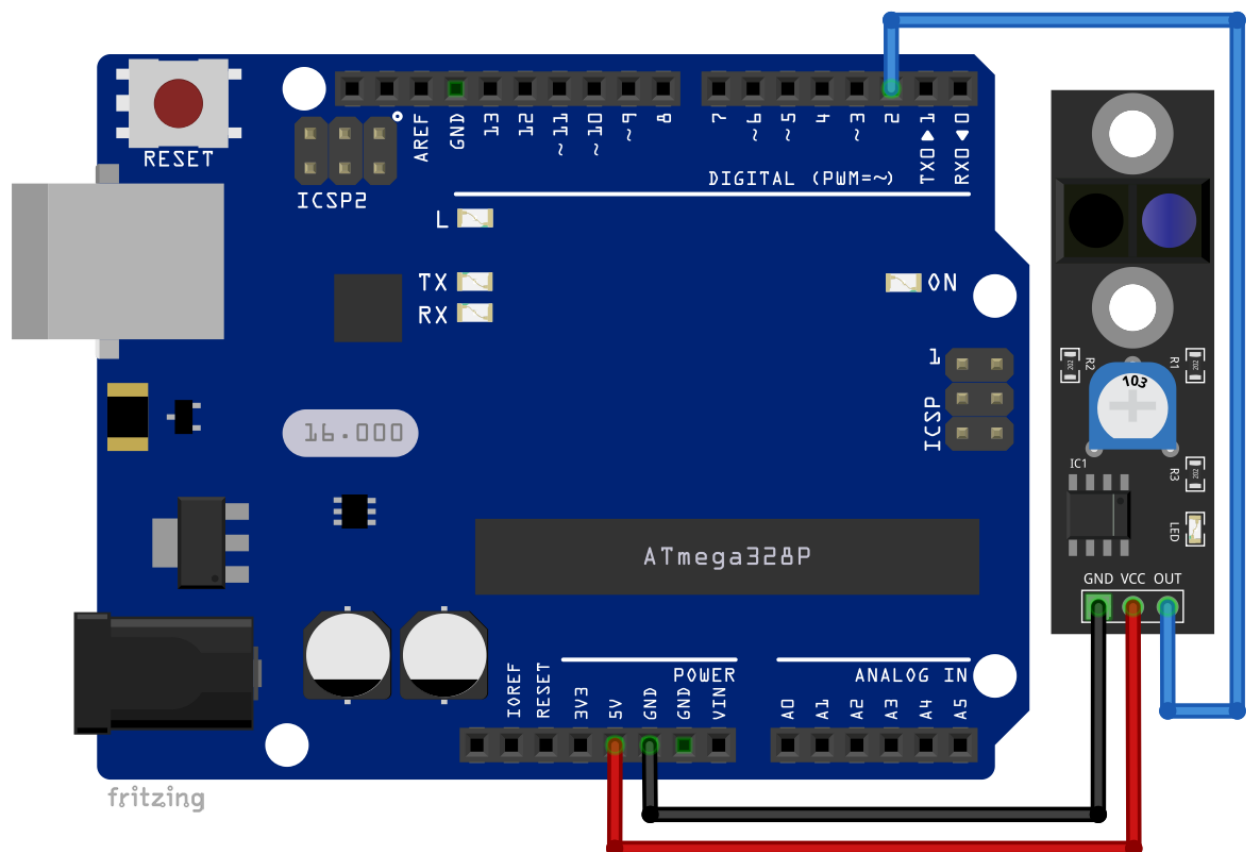
The pinout

The KY-033 line-tracker sensor module has three pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-033 module with the Uno as shown on the following connection diagram:



KY-033 pin	Uno pin	Wire color
OUT	D2	Blue wire
GND	GND	Black wire
VCC	5V	Red wire

Az-Delivery

Sketch example

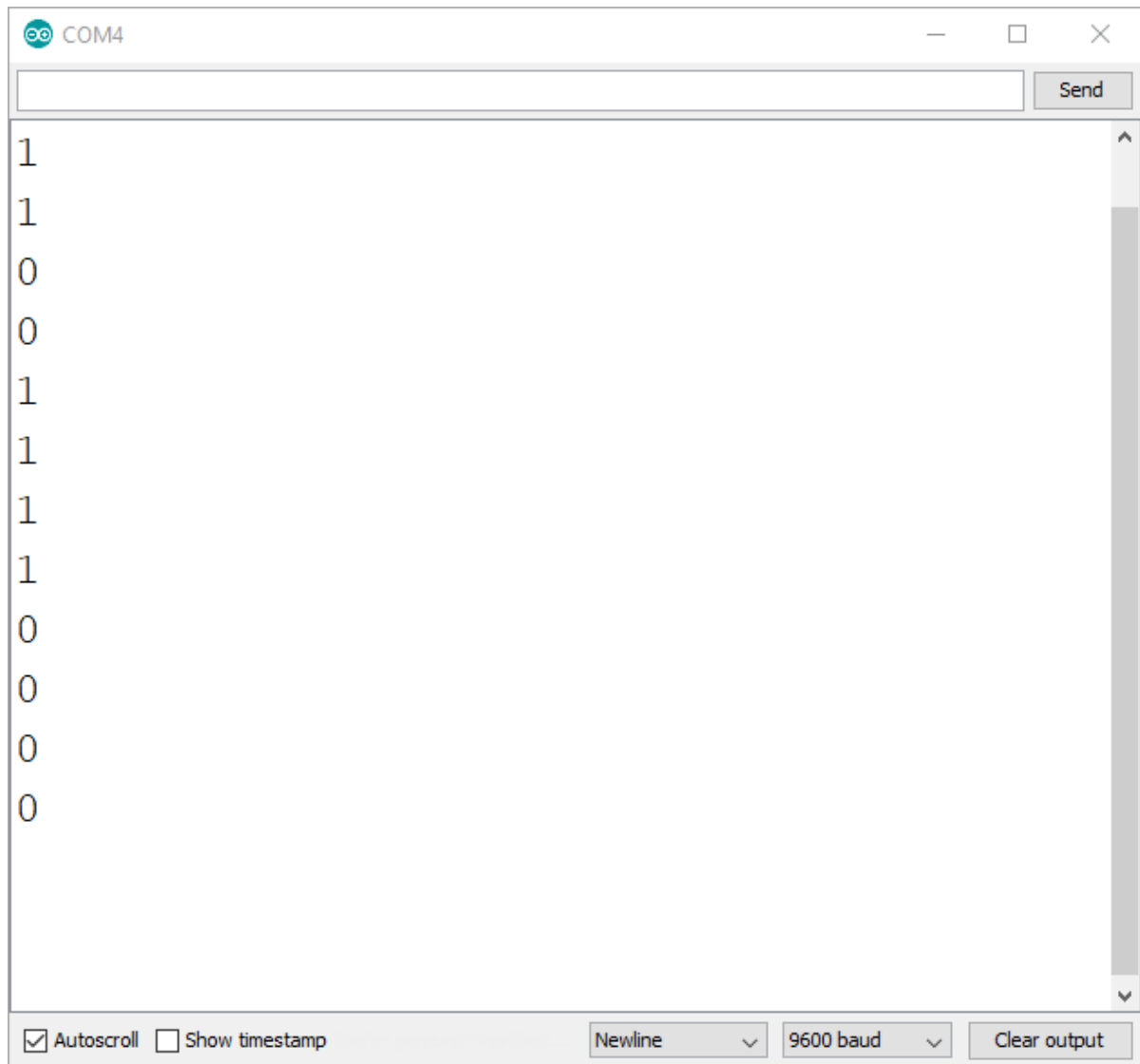
```
#define OUT_PIN 2

void setup() {
  Serial.begin(9600);
  pinMode(OUT_PIN, INPUT);
}

void loop() {
  Serial.println(digitalRead(OUT_PIN));
  delay(1000);
}
```

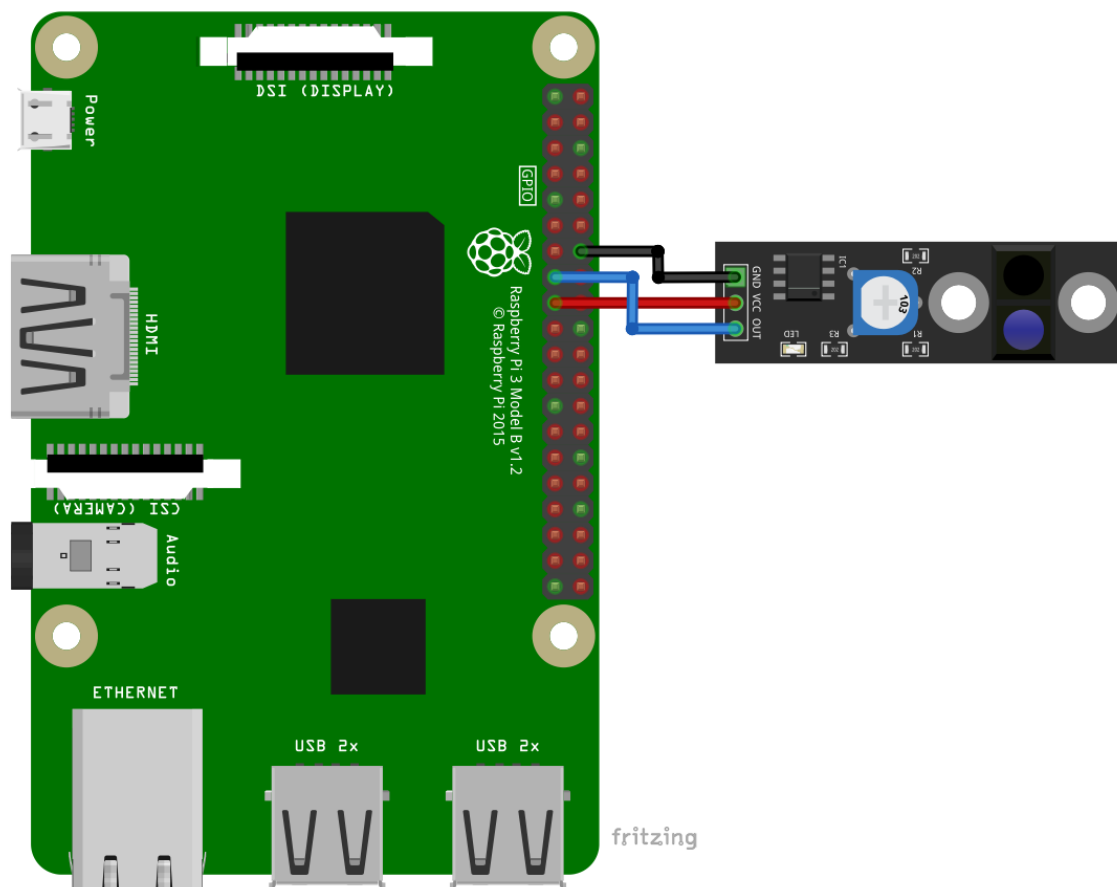
Az-Delivery

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



Connecting the module with Raspberry Pi

Connect the KY-033 module with the Raspberry Pi as shown on the following connection diagram:



KY-033 pin	Raspberry Pi pin	Physical pin	Wire color
GND	GND	14	Black wire
OUT	GPIO22	15	Blue wire
VCC	3V3	17	Red wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep

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

Signal_PIN = 22
GPIO.setup(Signal_PIN, GPIO.IN)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        if GPIO.input(Signal_PIN):
            print('Line detected')
        else:
            print('No line')
        sleep(1)

except KeyboardInterrupt:
    print('\nScript end!')

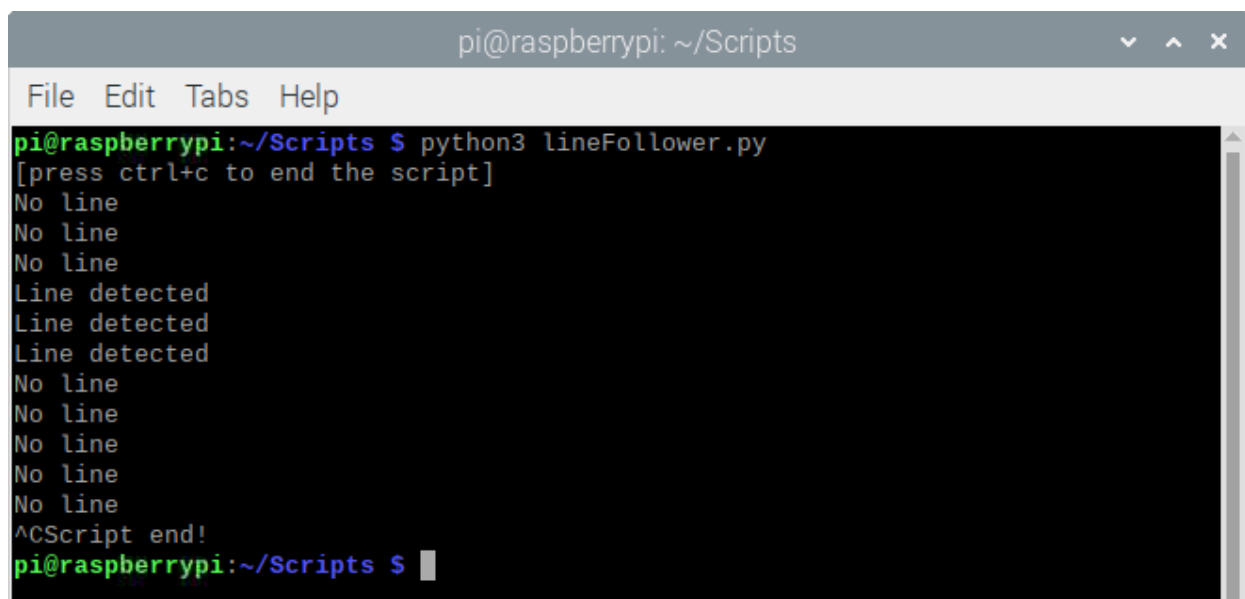
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *lineTracker.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 lineTracker.py

The result should look like the output on the following image:

A screenshot of a terminal window titled 'pi@raspberrypi: ~/Scripts'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal shows the command 'python3 lineFollower.py' being executed. The output consists of several lines: '[press ctrl+c to end the script]', 'No line', 'No line', 'No line', 'Line detected', 'Line detected', 'Line detected', 'No line', 'No line', 'No line', 'No line', 'No line', and '^CScript end!'. The prompt 'pi@raspberrypi:~/Scripts \$' is visible at the bottom with a cursor.

To end the script press *CTRL* + *C* on the keyboard.



KY-034 7 Color Changing LED Module

Introduction

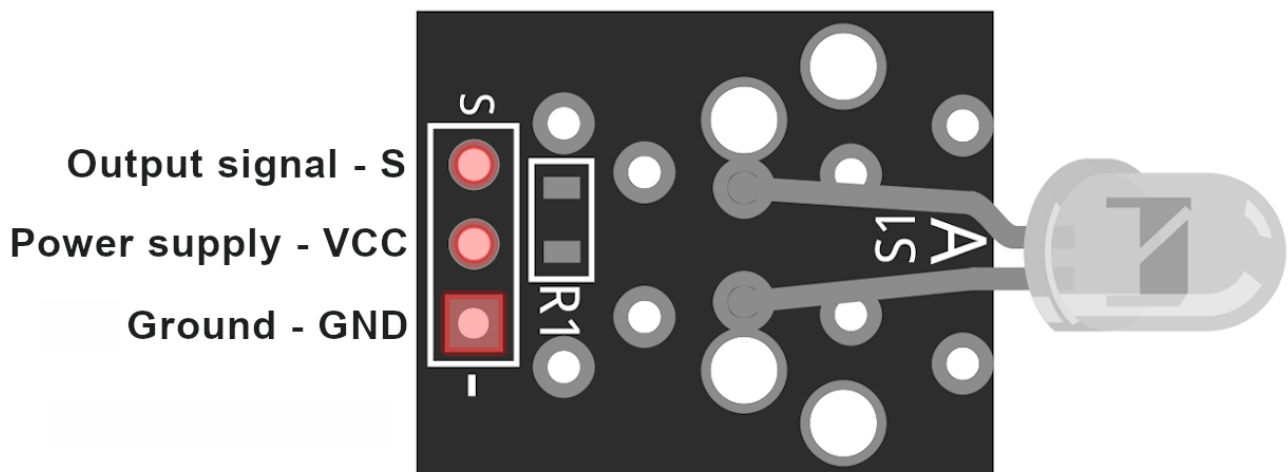
The LED on-board the KY-034 module can produce light in seven different colors and change from one color to another automatically. The order of the color changing cycles can not be changed. The only thing that can be controlled is when the module is turned *ON* or *OFF*. When the module is turned *ON* it automatically starts to change colors in some predefined cycles. The on-board LED is a high brightness LED, so be careful not to look directly into LED when it is turned *ON*. The brightness level of an LED can be limited in hardware with a resistor connected serially with the signal line. The module works on voltages in range from +3.3V up to +5V.

Specifications

Operating voltage range	from 3.3V to 5V DC
Operating temperature	-25°C to 80°C [-13°F to 176°F]
Dimensions	19x15mm [0.73x0.6in]

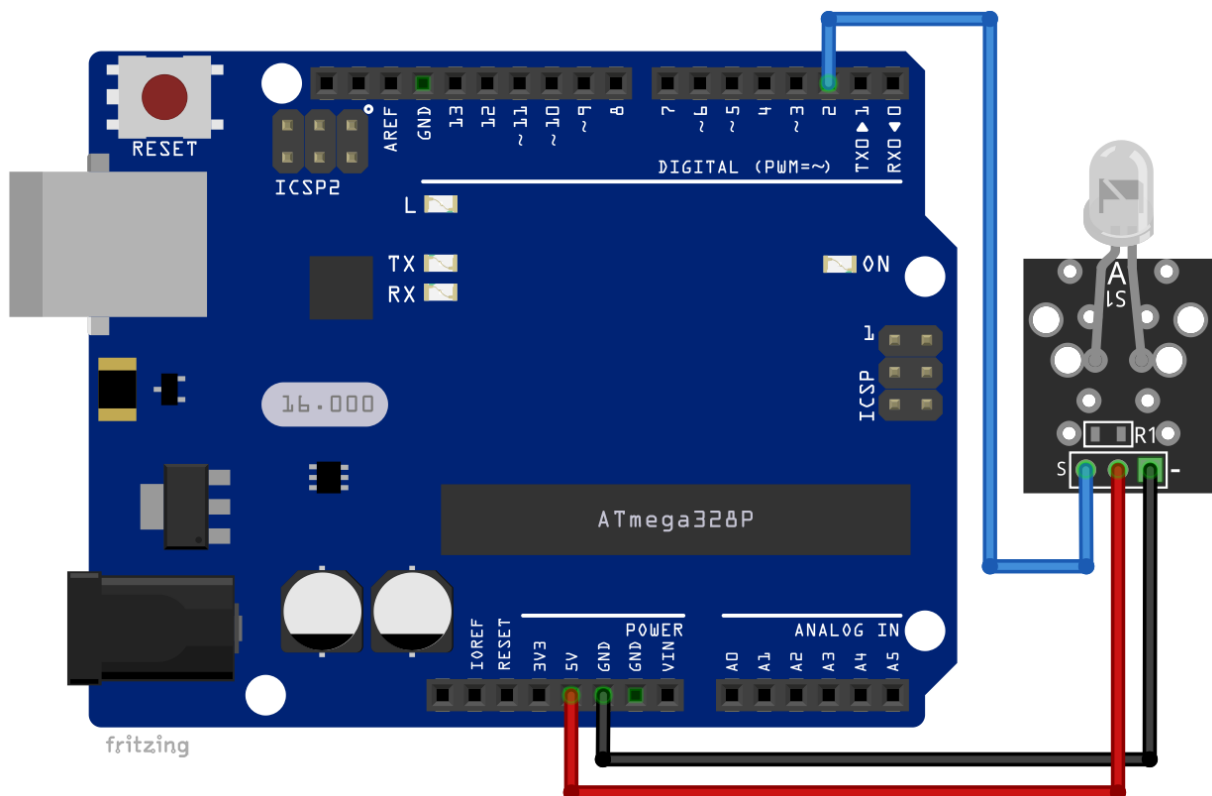
The pinout

The KY-034 7 color changing LED module has three pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-034 module with the Uno as shown on the following connection diagram:



KY-034 pin	Uno pin	Wire color
S	D2	Blue wire
- (GND)	GND	Black wire
Middle pin (VCC)	5V	Red wire

Az-Delivery

Sketch example

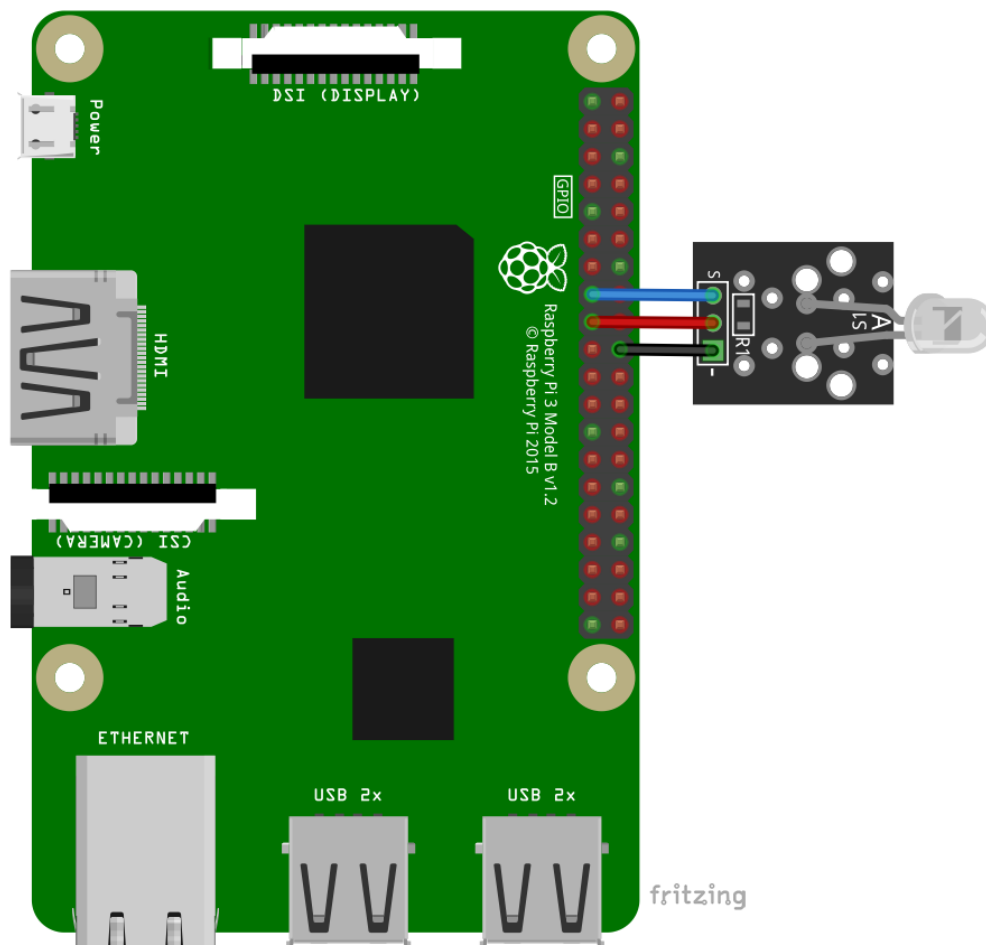
```
#define LED_PIN 2

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_PIN, HIGH);
  delay(5000);
  digitalWrite(LED_PIN, LOW);
  delay(500);
}
```

Connecting the module with Raspberry Pi

Connect the KY-034 module with the Raspberry Pi as shown on the following connection diagram:



KY-034 pin	Raspberry Pi pin	Physical pin	Wire color
S	GPIO22	15	Blue wire
Middle pin (VCC)	3V3	17	Red wire
- (GND)	GND	20	Black wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)
Led_PIN = 22
GPIO.setup(Led_PIN, GPIO.OUT, initial=GPIO.LOW)
print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        GPIO.output(Led_PIN, GPIO.HIGH) # LED ON
        sleep(10) # Wait for 10 seconds
        GPIO.output(Led_PIN, GPIO.LOW) # LED OFF
        sleep(1) # Wait for a second

except KeyboardInterrupt:
    print('\nScript end!\n')

finally:
    GPIO.cleanup()
```

Save the script by the name *SevenColor.py*. To run the script, open the terminal in the directory where you saved the script and run the following command:

python3 SevenColor.py

To end the script press *CTRL + C* on the keyboard.



KY-035 Linear Magnetic Sensor Module

Introduction

The KY-035 linear magnetic sensor module has the *49E* linear magnetic sensor that features low noise output. The module is a very responsive switch that gets activated by a magnetic field. It can be triggered by a natural magnet or an electromagnet with high response rates. The linear output voltage is set by the supply voltage and varies in proportion to the strength of the magnetic field.

The most popular uses are in proximity sensing, detecting speeds of spinning objects or as door sensors. Due to the nature of the sensing, the contactless sensing, the KY-035 module is very durable and there is no wearing associated with the use.

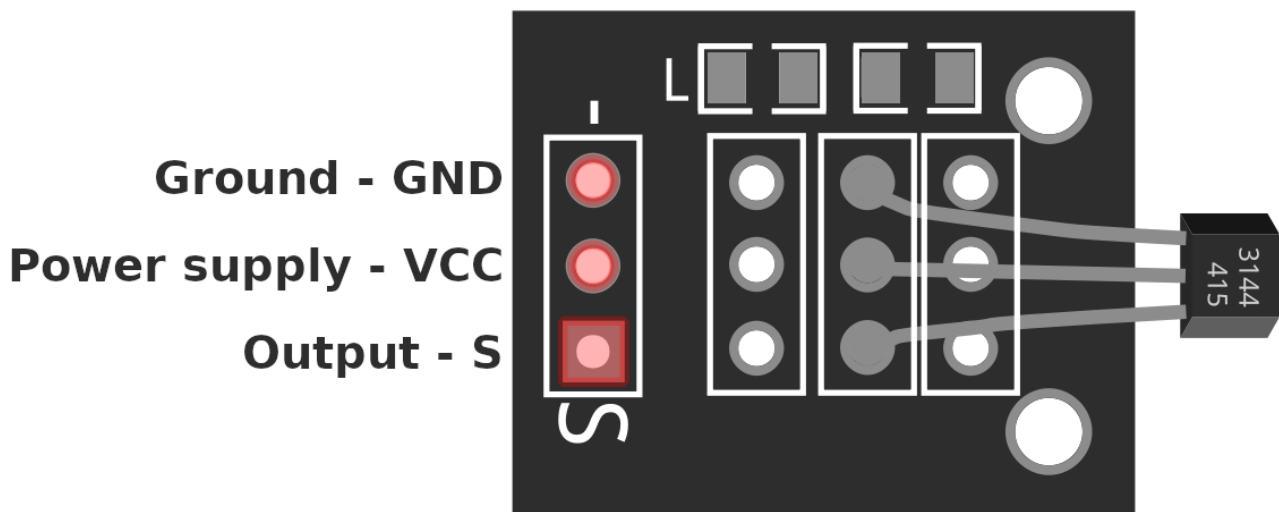
Specifications

Operating voltage range	from 2.3V to 6.5V DC
Power consumption	4mA at 5 V DC
Sensitivity	1.4 ~ 3.0mV/GS on 25°C
Output	linear analog voltage
Temperature range	from -40°C to 100°C [-40°F to 212°F]
Dimensions	19x15mm [0.73x0.6in]

When the magnet is near the sensor, the analog output voltage value on the S pin drops linearly with the strength of the magnetic field. When the opposite magnetic pole is near the sensor, the analog output voltage value on the S pin rises linearly with the strength of the magnetic field. This can be on the Serial Monitor output image in the *Connecting the module with Uno* chapter.

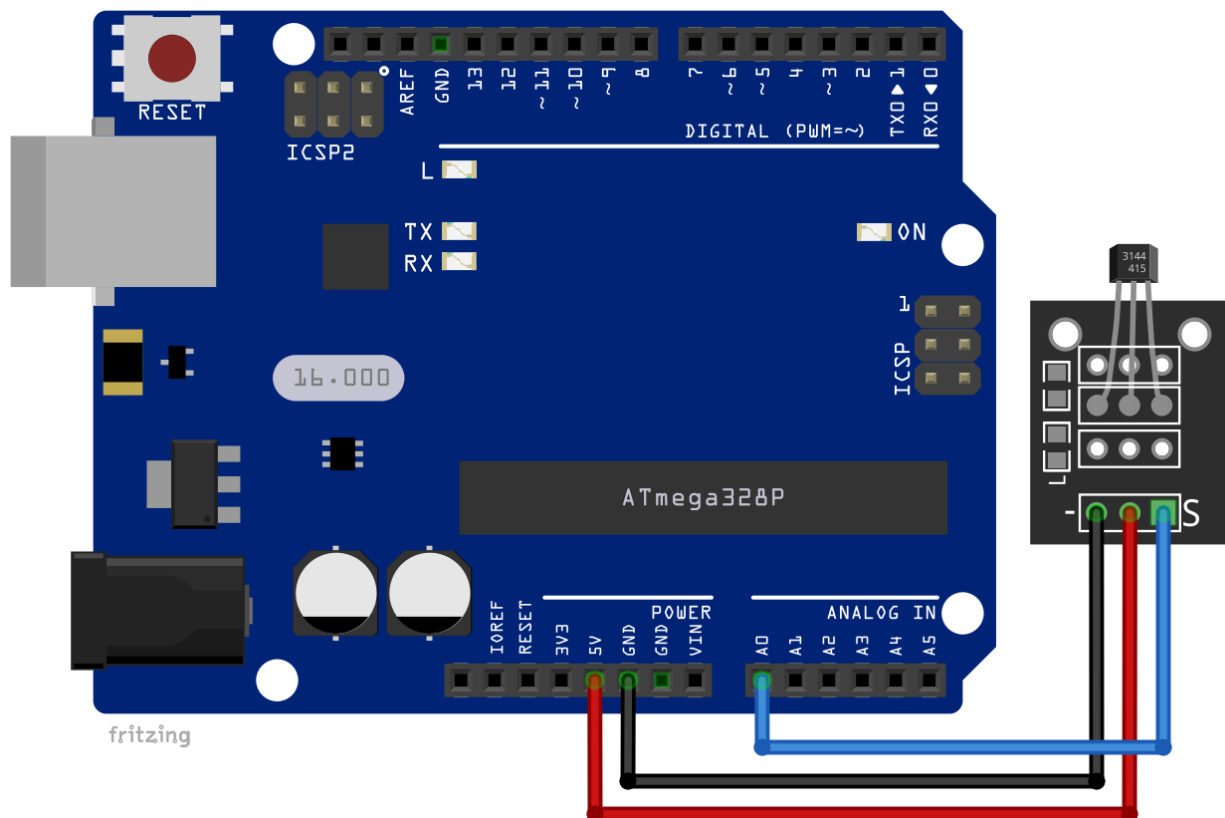
The pinout

The KY-035 linear magnetic sensor module has three pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-035 module with the Uno as shown on the following connection diagram:



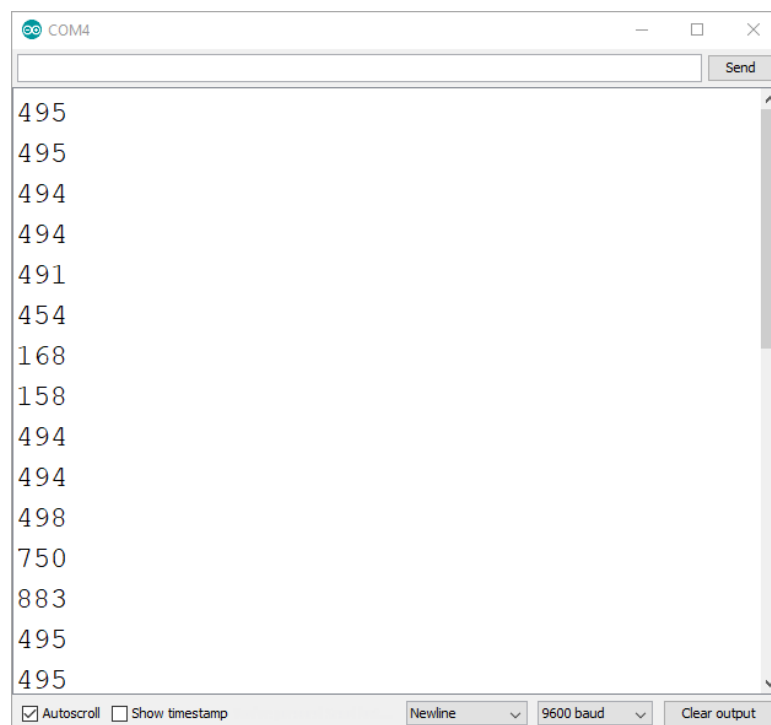
KY-035 pin	Uno pin	Wire color
S	A0	Blue wire
- (GND)	GND	Black wire
Middle pin (VCC)	5V	Red wire

Az-Delivery

Sketch example

```
#define ANALOG_PIN A0
void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.println(analogRead(ANALOG_PIN));
  delay(1000);
}
```

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



Az-Delivery

Connecting the module with Raspberry Pi

Connect the KY-035 module with the Uno as shown on the connection diagram from the chapter *Connecting the module with Uno*, and then connect the Uno with the Raspberry Pi via USB cable. Next, upload *nanpy* firmware to the Uno (as shown in the chapter *Uno as ADC*), and use the following code to read the KY-035 module:

```
from nanpy import (ArduinoApi, SerialManager)
from time import sleep
try:
    connection1 = SerialManager()
    a = ArduinoApi(connection=connection1)
except:
    print('Failed to connect to the Arduino')

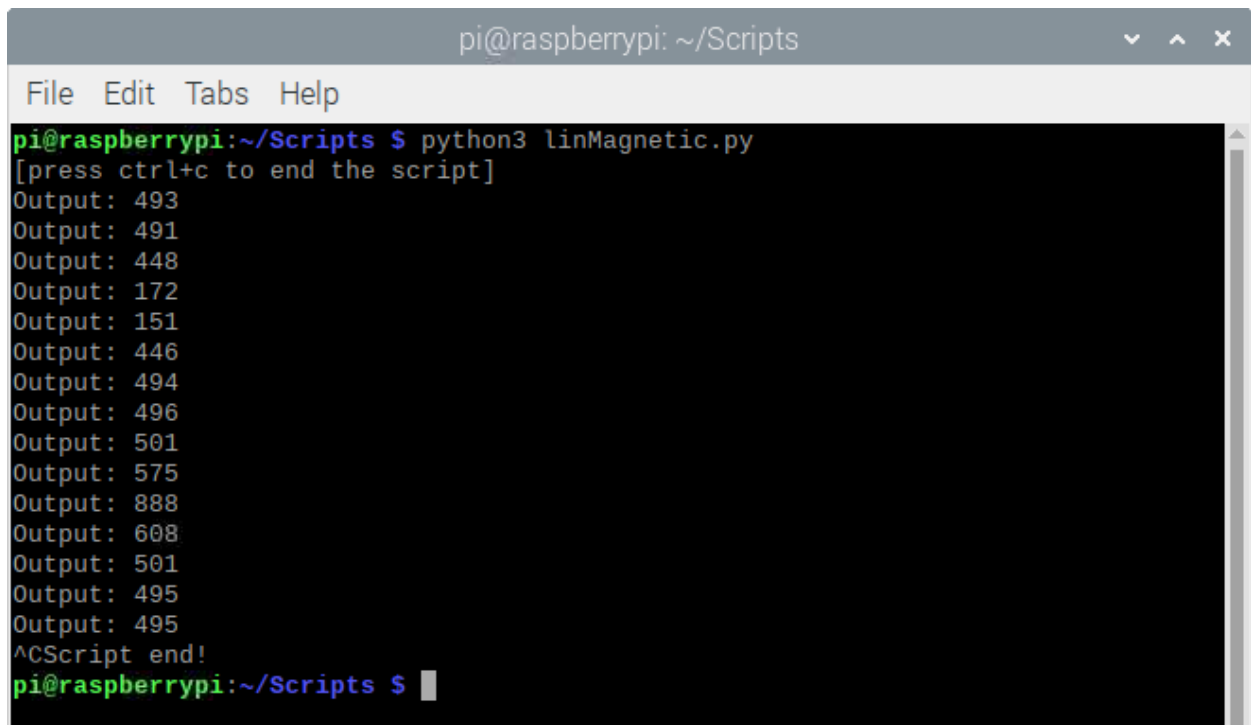
value = 0
print('[Press CTRL + C to end the script!}')
try:
    while True:
        value = a.analogRead(0)
        print('Output: {}'.format(value))
        sleep(1)

except KeyboardInterrupt:
    print('\nScript end!')
```

Az-Delivery

Save the script by the name *linMagnetic.py*. To run the script open the terminal in the directory where the script is saved and run the following command: **python3 linMagnetic.py**

The result should look like the output on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 linMagnetic.py
[press ctrl+c to end the script]
Output: 493
Output: 491
Output: 448
Output: 172
Output: 151
Output: 446
Output: 494
Output: 496
Output: 501
Output: 575
Output: 888
Output: 608
Output: 501
Output: 495
Output: 495
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL + C* on the keyboard.

KY-036 Touch Sensor Module

Introduction

On-board the KY-036 touch sensor module there are the *MPSA13* Darlington transistor, a few resistors, two LEDs, a potentiometer and an LM393 analog voltage comparator chip. The MPSA13 Darlington transistor is used as the touch sensor. Transistors have three legs one for the base, one for the emitter and one for the collector. For the module, the base pin of the Darlington transistor is bent over the casing of the transistor and acts as a metal rod for touch sensing. When a human touches the metal rod, voltage potential changes on metal leg of the transistor and the transistor gets activated. This small change in voltage potential is then amplified in the Darlington transistor and can be used to signal the touch.

There are two LEDs on-board the module one for indicating power supply and the other for indicating that that sensor is touched.

Specifications

Operating voltage range	from 3.3V to 5.5V DC
Output	digital and analog
Dimensions	15x36x11mm [0.6x1.4in]

The analog output pin voltage sensitivity can be adjusted via the on-board blue potentiometer.

With some adjustment of the potentiometer, when the analog voltage is read via an analog to digital converter (ADC for short), the digital value of the voltage is around *1023*. That is the state when the KY-036 sensor is not touched.

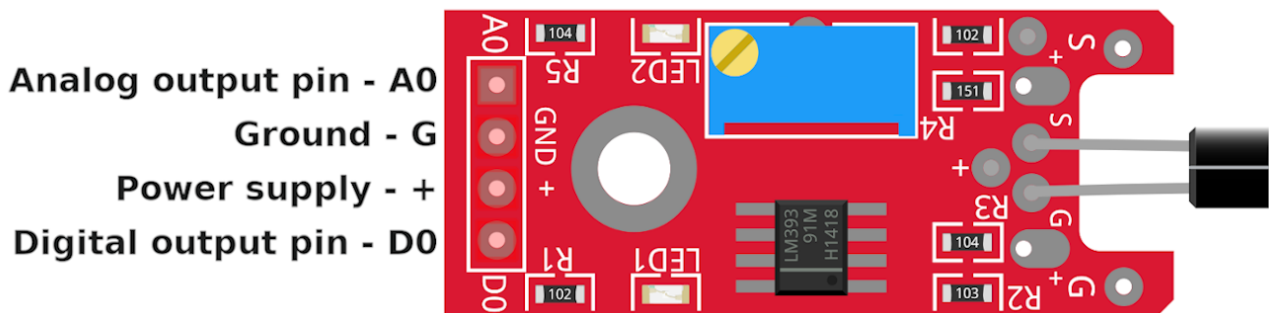
When the sensor is touched, the ADC value drops to the value around *115*. This is the state where the digital output pin outputs the *HIGH* state (is turned *ON*).

When the sensor is not touched (after being touched), ADC value starts to rise (which is shown in the Serial Monitor, later in the text).

The digital output pin outputs the *LOW* state (is turned *OFF*) when the sensor is not touched.

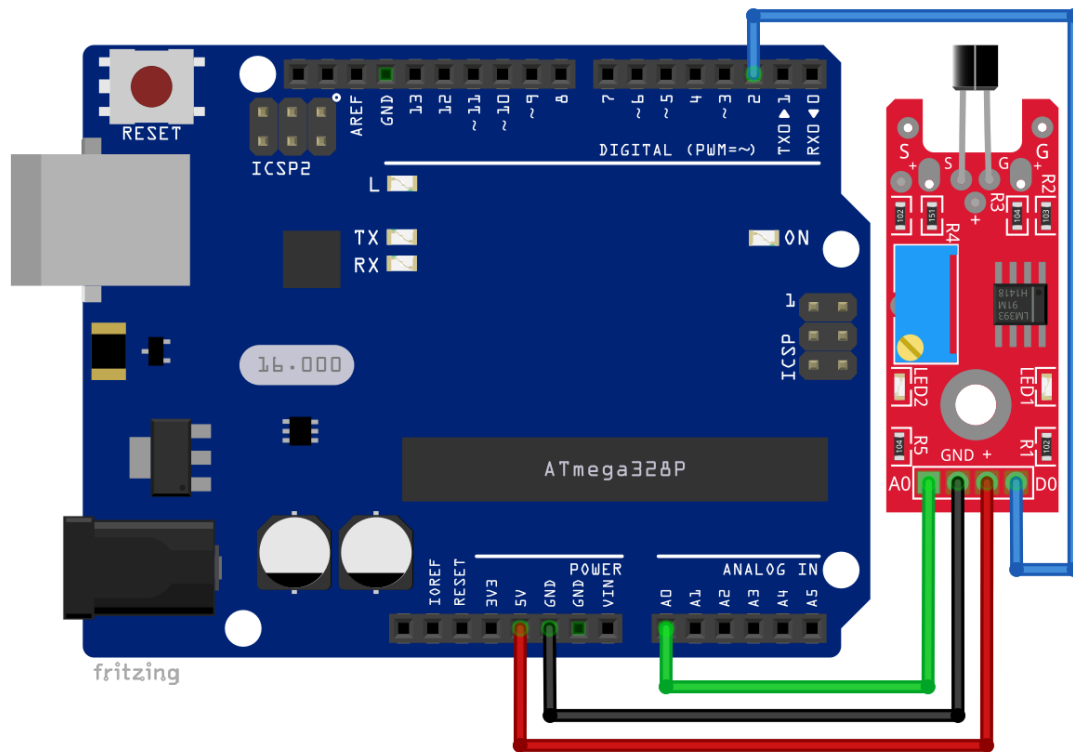
The pinout

The KY-036 touch sensor module has four pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-036 module with the Uno as shown on the following connection diagram:



KY-036 pin	Uno pin	Wire color
+	5V	Red wire
GND	GND	Black wire
A0	A0	Green wire
D0	D2	Blue wire

With this set-up both the analog and digital pins of the KY-036 module are used.

Az-Delivery

Sketch example

```
#define DIGITAL_PIN 2
#define ANALOG_PIN 0

void setup() {
  Serial.begin(9600);
  pinMode(DIGITAL_PIN, INPUT);
}

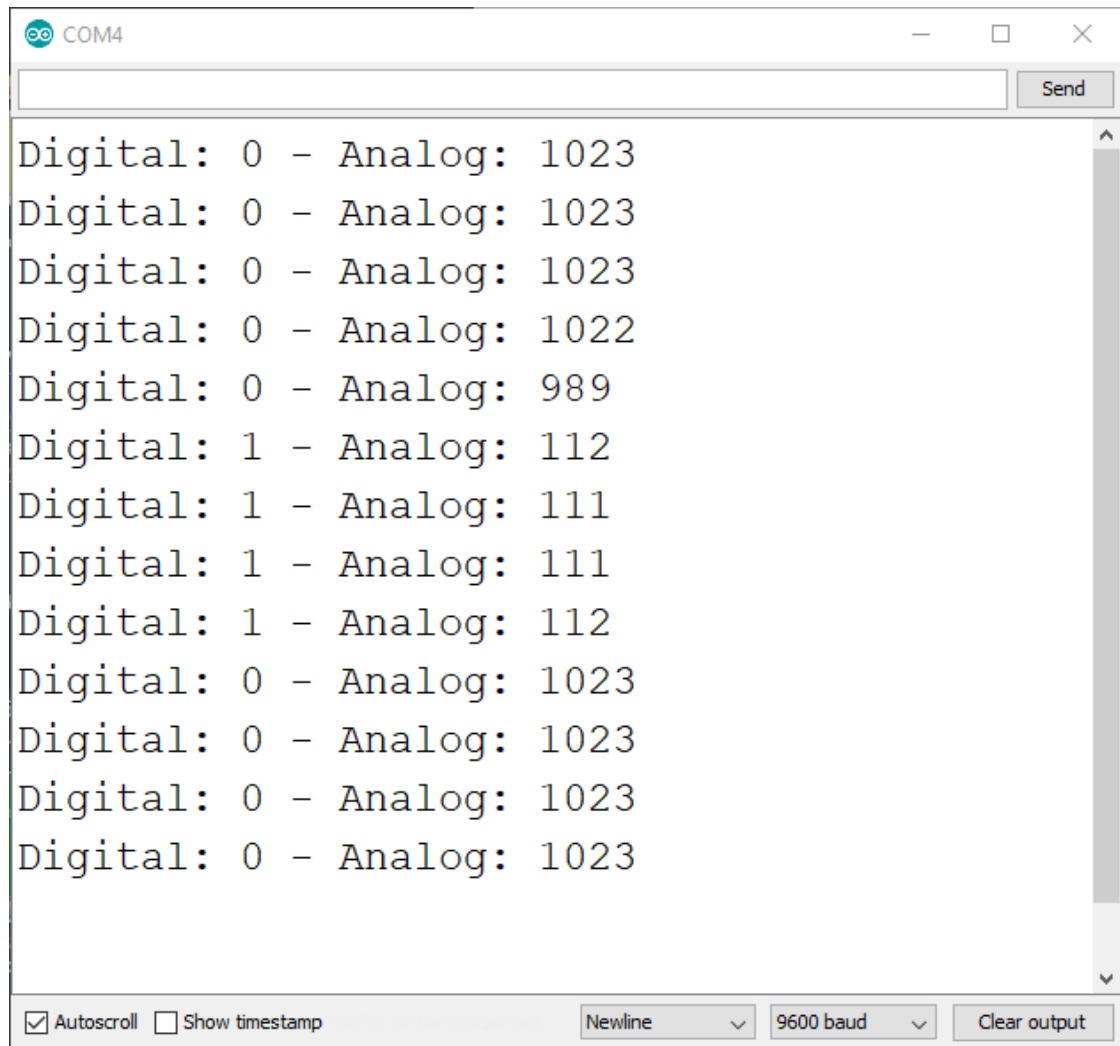
void loop() {
  Serial.print("Digital: ");
  Serial.print(digitalRead(DIGITAL_PIN));

  Serial.print(" - Analog: ");
  Serial.println(analogRead(ANALOG_PIN));

  delay(1000);
}
```

Az-Delivery

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



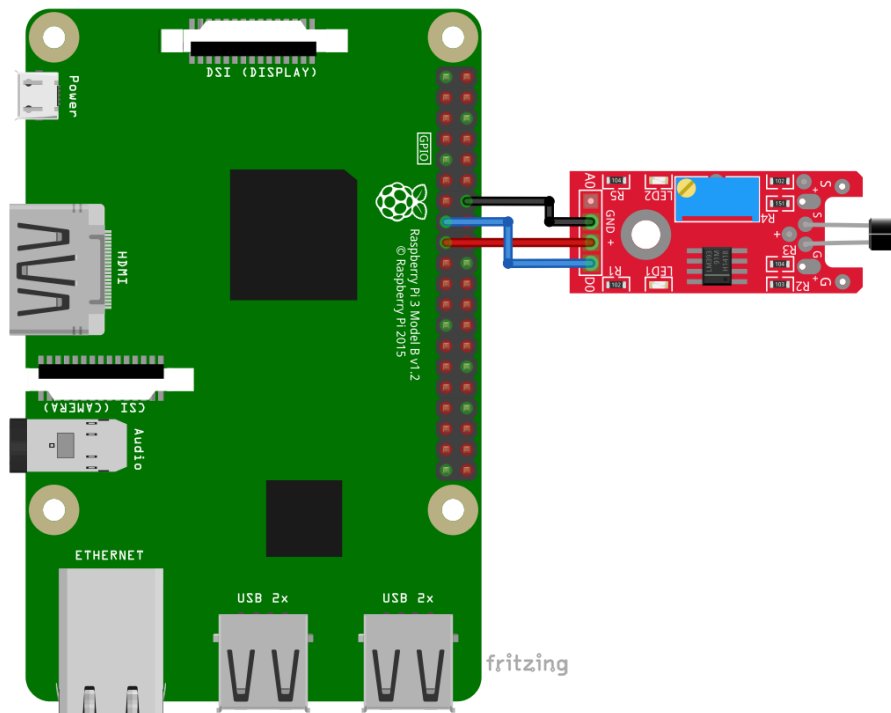
The screenshot shows the Serial Monitor window for COM4. It displays a series of 14 lines of data, each containing a digital reading followed by an analog reading. The digital readings are 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0. The analog readings are 1023, 1023, 1023, 1022, 989, 112, 111, 111, 112, 1023, 1023, 1023, 1023. The window includes a 'Send' button at the top right and a status bar at the bottom with options for 'Autoscroll' (checked), 'Show timestamp' (unchecked), 'Newline' (selected), '9600 baud' (selected), and a 'Clear output' button.

```
COM4
Digital: 0 - Analog: 1023
Digital: 0 - Analog: 1023
Digital: 0 - Analog: 1023
Digital: 0 - Analog: 1022
Digital: 0 - Analog: 989
Digital: 1 - Analog: 112
Digital: 1 - Analog: 111
Digital: 1 - Analog: 111
Digital: 1 - Analog: 112
Digital: 0 - Analog: 1023
Digital: 0 - Analog: 1023
Digital: 0 - Analog: 1023
Digital: 0 - Analog: 1023
```

☒ Autoscroll ☐ Show timestamp Newline 9600 baud Clear output

Connecting the module with Raspberry Pi

Connect the KY-036 module with the Raspberry Pi as shown on the following connection diagram:



KY-036 pin	Raspberry Pi pin	Physical pin	Wire color
GND	GND	14	Black wire
D0	GPIO22	15	Blue wire
+ (VCC)	3V3	17	Red wire

With this set-up the Raspberry Pi is used to read only the digital pin of the KY-036 module. If the analog pin has to be used, use an analog to digital converter of the Uno or an external ADC, like the *ADS1115*. This is not covered in the eBook.

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

Digital_PIN = 22
GPIO.setup(Digital_PIN, GPIO.IN)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        if GPIO.input(Digital_PIN):
            print('Touch detected')
        else:
            print('No touch')

        sleep(1)

except KeyboardInterrupt:
    print('\nScript end!')

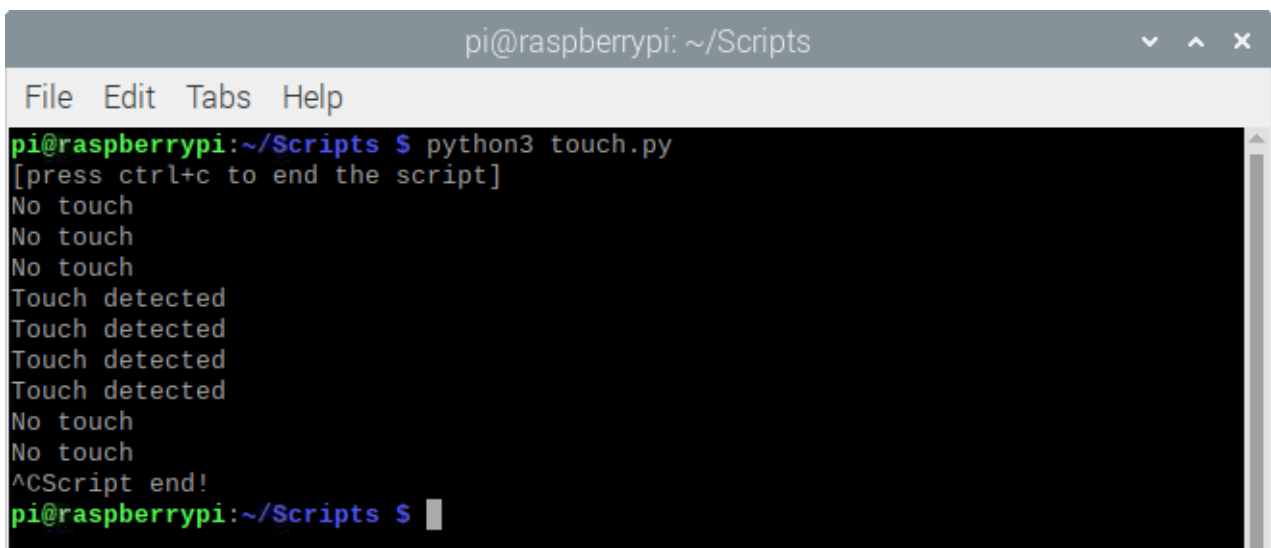
finally:
    GPIO.cleanup()
```


Az-Delivery

Save the script by the name *touch.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 touch.py

The result should look like the output on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 touch.py
[press ctrl+c to end the script]
No touch
No touch
No touch
Touch detected
Touch detected
Touch detected
Touch detected
No touch
No touch
^CScript end!
pi@raspberrypi:~/Scripts $
```

To end the script press *CTRL + C* on the keyboard.

KY-037 Big Microphone Sensor Module

Introduction

The KY-037 big microphone module has the capacitive electret microphone, a few resistors, two LEDs, a potentiometer and an LM393 analog voltage comparator.

The capacitive electret microphone is a sound sensor - a component that converts sound waves into electrical signals. The sensor detects the sound wave intensity in the environment. This kind of microphone has a thin film (like thin foil) of the electret (dielectric) material which vibrates on sound waves. The capacitance of the thin film changes with vibrations, which cause the corresponding analog voltage change. Since the analog voltage change is weak, it has to be amplified. The LM393 and a few resistors are in charge of the amplification.

Specifications

Operating voltage range	from 3.3V to 5V DC
Output	digital and analog
Frequency range	from 100Hz to 10.000Hz
Sensitivity	-46 ± 2.0 (0 dB = 1V/Pa) at 1kHz
Sensitivity to Noise Ratio	58dB
Dimensions	15x36x13mm [0.6x1.4x0.51in]

There are two LEDs on-board the module, one for indicating power supply and one for indicating the intensity of the sound.

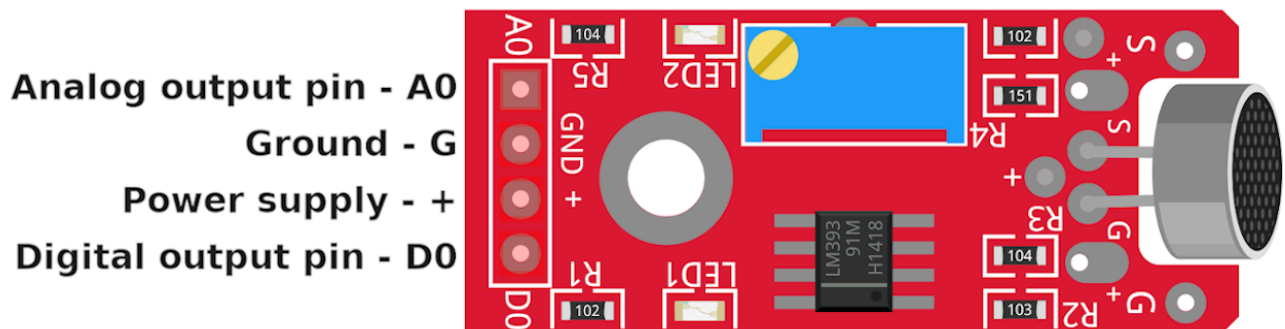
The analog output pin voltage sensitivity can be adjusted via an on-board blue potentiometer. With some adjustment of the potentiometer, when the analog voltage is read via analog to digital converter (ADC for short), the digital value of the voltage is around the value 595, and that is when there is not sound near the KY-037 module.

When there is a sound near the sensor, the ADC value changes dramatically around the value 595, rise and drop a few times. This happens because of the nature of sound waves. This can be seen on the image of the Serial Plotter (later in the text).

The state when the digital output pin outputs the *HIGH* state (is turned *ON*) is the state when the ADC value is grater than the value of 700.

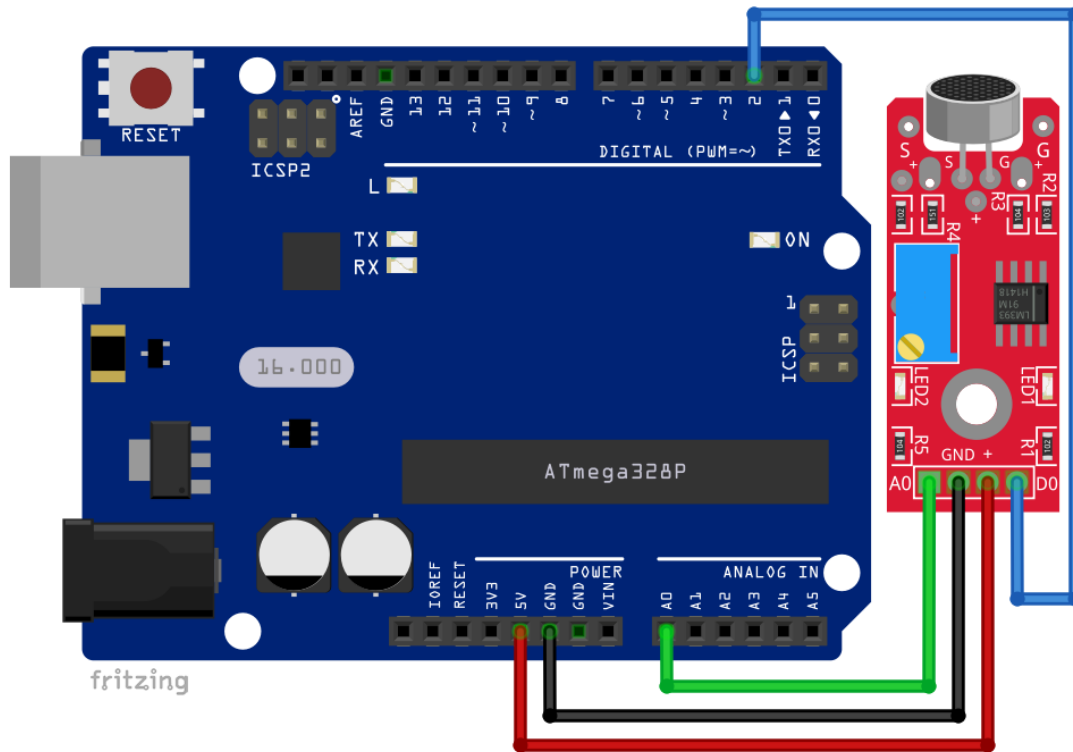
The pinout

The KY-037 big microphone module has four pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-037 module with the Uno as shown on the following connection diagram:



KY-037 pin	Uno pin	Wire color
+	5V	Red wire
GND	GND	Black wire
A0	A0	Green wire
D0	D2	Blue wire

With the set-up, both the analog and digital pins of the KY-037 module are used.

Az-Delivery

Sketch example

```
#define DIGITAL_PIN 2
#define ANALOG_PIN 0

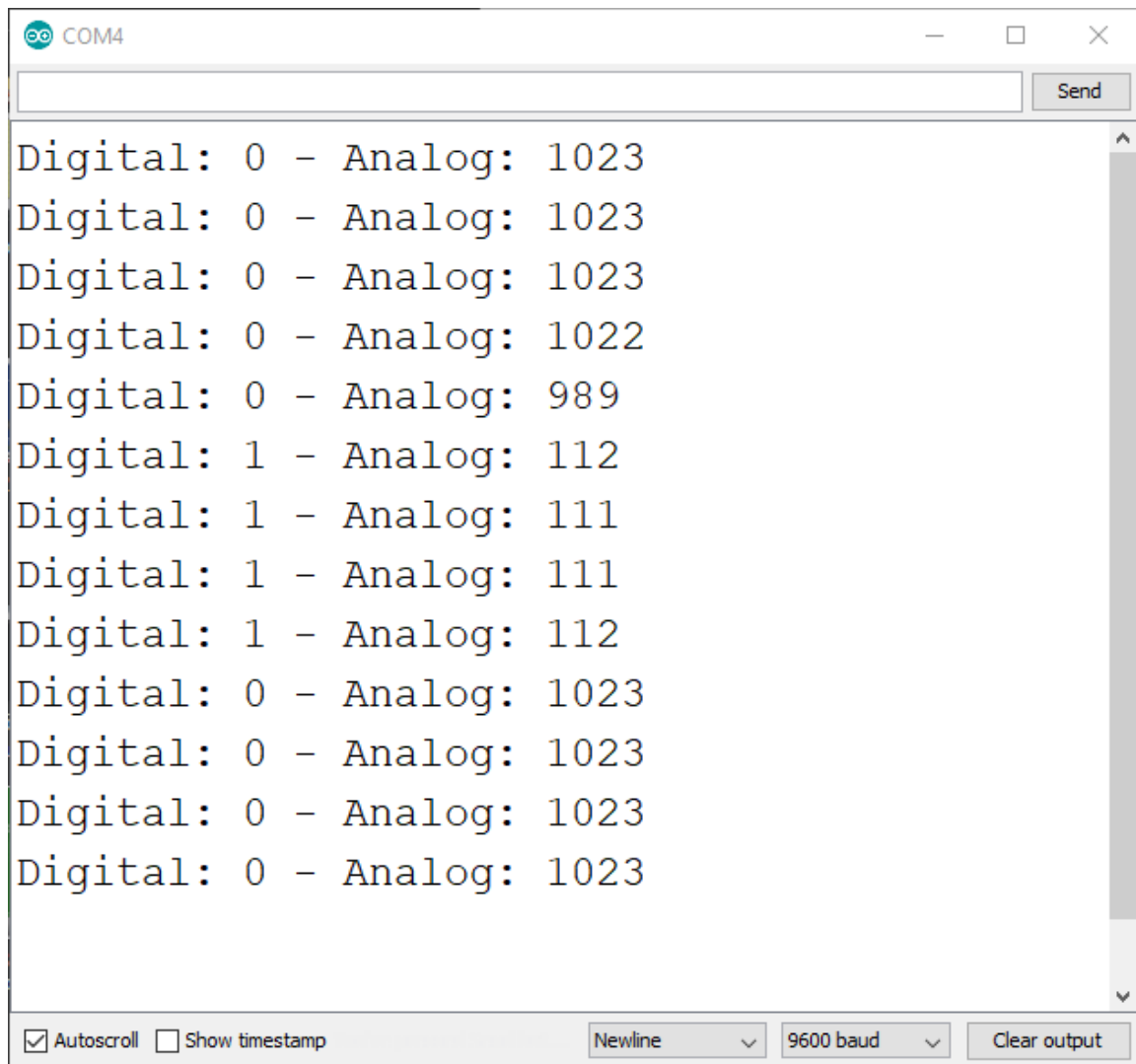
void setup() {
  pinMode(DIGITAL_PIN, INPUT);
  Serial.begin(9600);
}

void loop() {
  Serial.print("Digital: ");
  Serial.print(digitalRead(DIGITAL_PIN));
  Serial.print(" - Analog: ");
  Serial.println(analogRead(ANALOG_PIN));

  delay(1000);
}
```

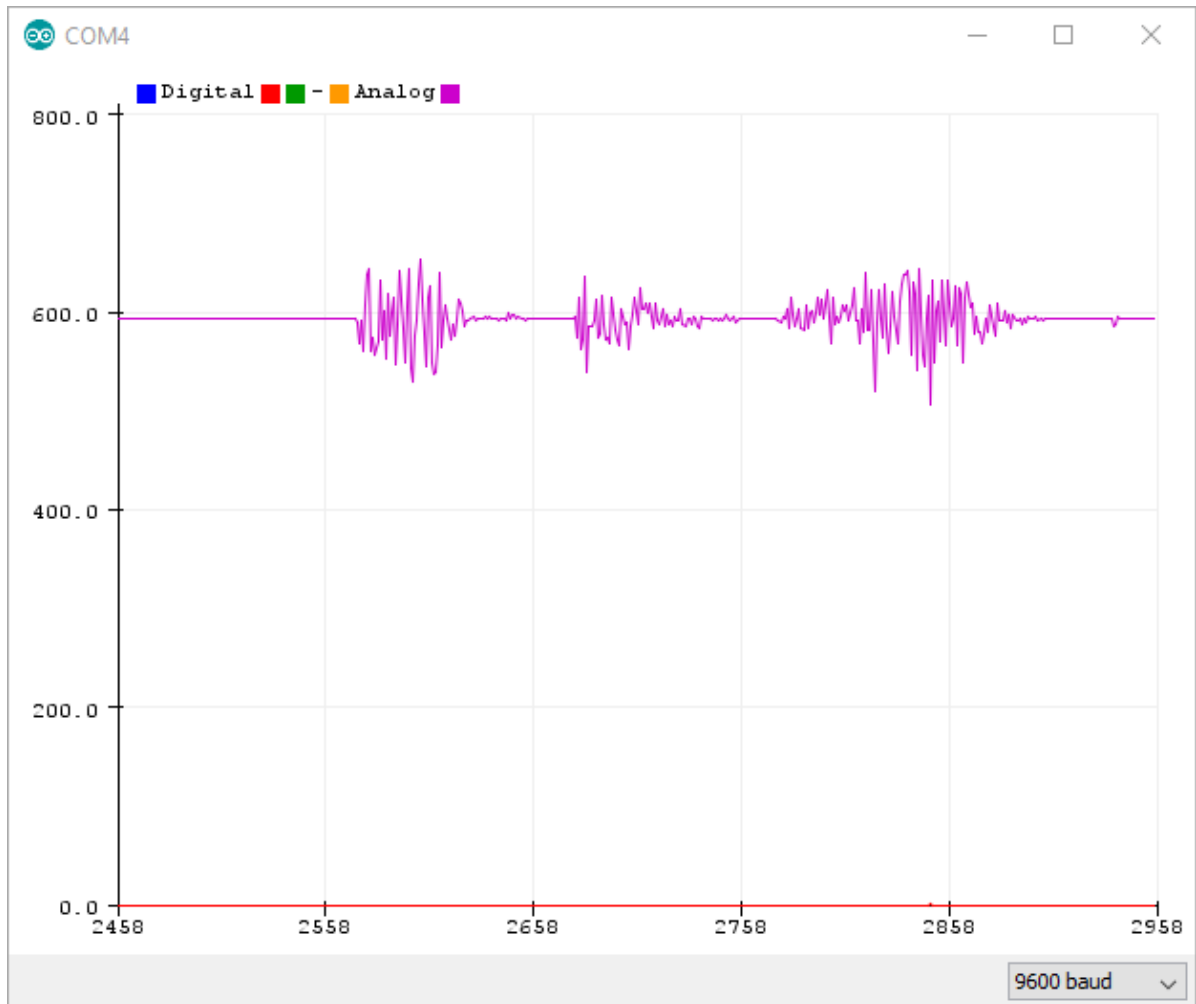
Az-Delivery

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



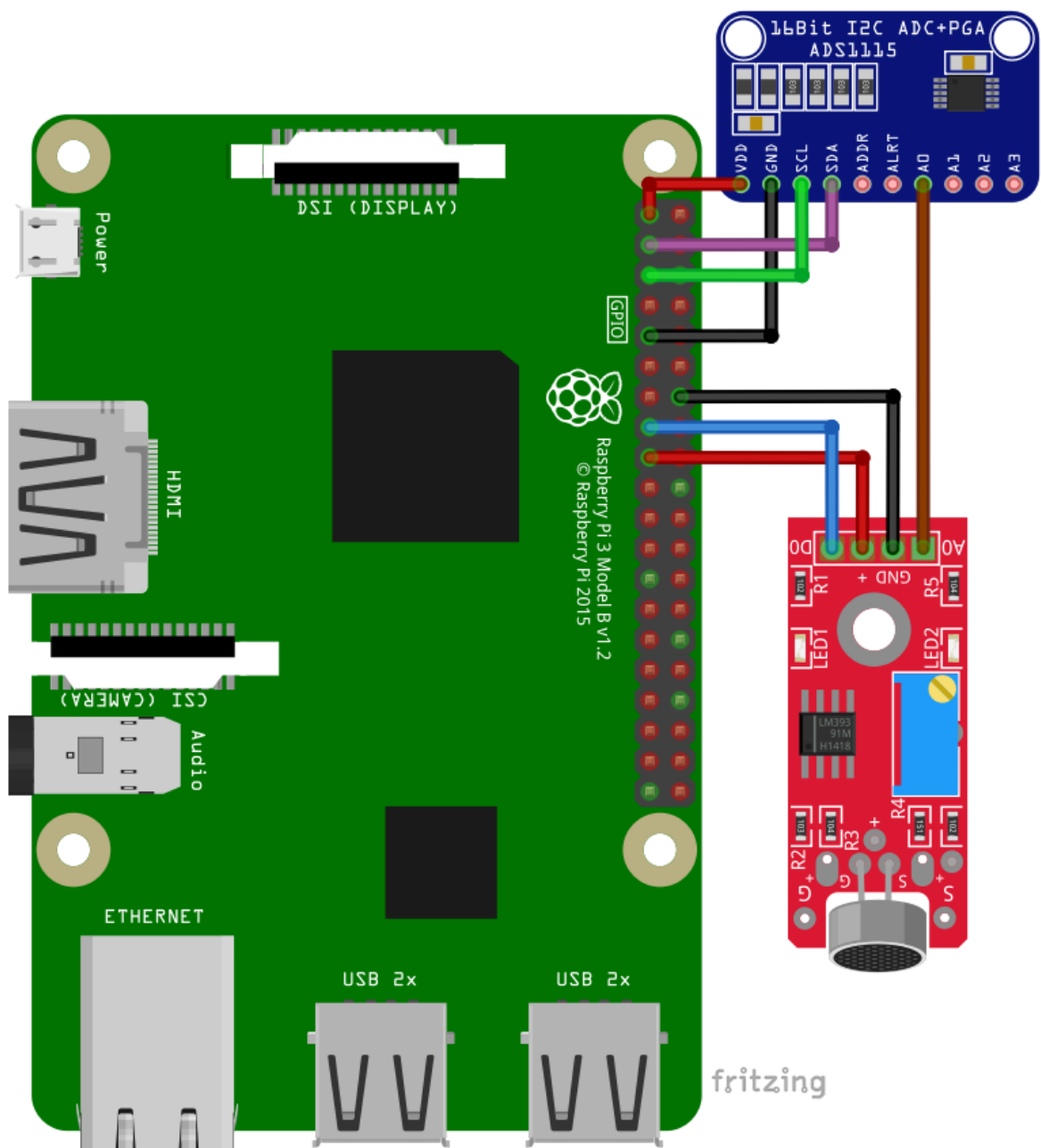
Az-Delivery

The sound waves can be seen on the Serial Plotter (*Tools > Serial Plotter*). The result should look like the output on the following image:



Connecting the module with Raspberry Pi

Connect the KY-037 module with the Raspberry Pi as shown on the following connection diagram:



KY-037 pin	Raspberry Pi pin	Physical pin	Wire color
GND	GND	14	Black wire
D0	GPIO22	15	Blue wire
+ (VCC)	3V3	17	Red wire
KY-037 pin	ADS1115 pin		
A0	A0		Brown wire
ADS1115 pin	Raspberry Pi pin	Physical pin	Wire color
VDD	3V3	1	Red wire
GND	GND	9	Black wire
SDA	GPIO2	3	Green wire
SCL	GPIO3	5	Blue wire

In this set-up, both the analog and digital output pins of the KY-037 module are used. Here, the device called *ADS1115* is used as an external analog to digital converter (ADC), where the analog data is sent to the Raspberry Pi via the I2C interface.

Az-Delivery

Python script

```
import time
import Adafruit_ADS1x15
import RPi.GPIO as GPIO

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

ADS1115 = Adafruit_ADS1x15.ADS1115()
GAIN = 1

Digital_PIN = 22
GPIO.setup(Digital_PIN, GPIO.IN)

print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        analog = ADS1115.read_adc(0, gain=GAIN) # ADC
        channel 0
        digital = GPIO.input(Digital_PIN)
        print('Digital: {} - Analog: {}'.format(digital, analog))
        time.sleep(0.002)

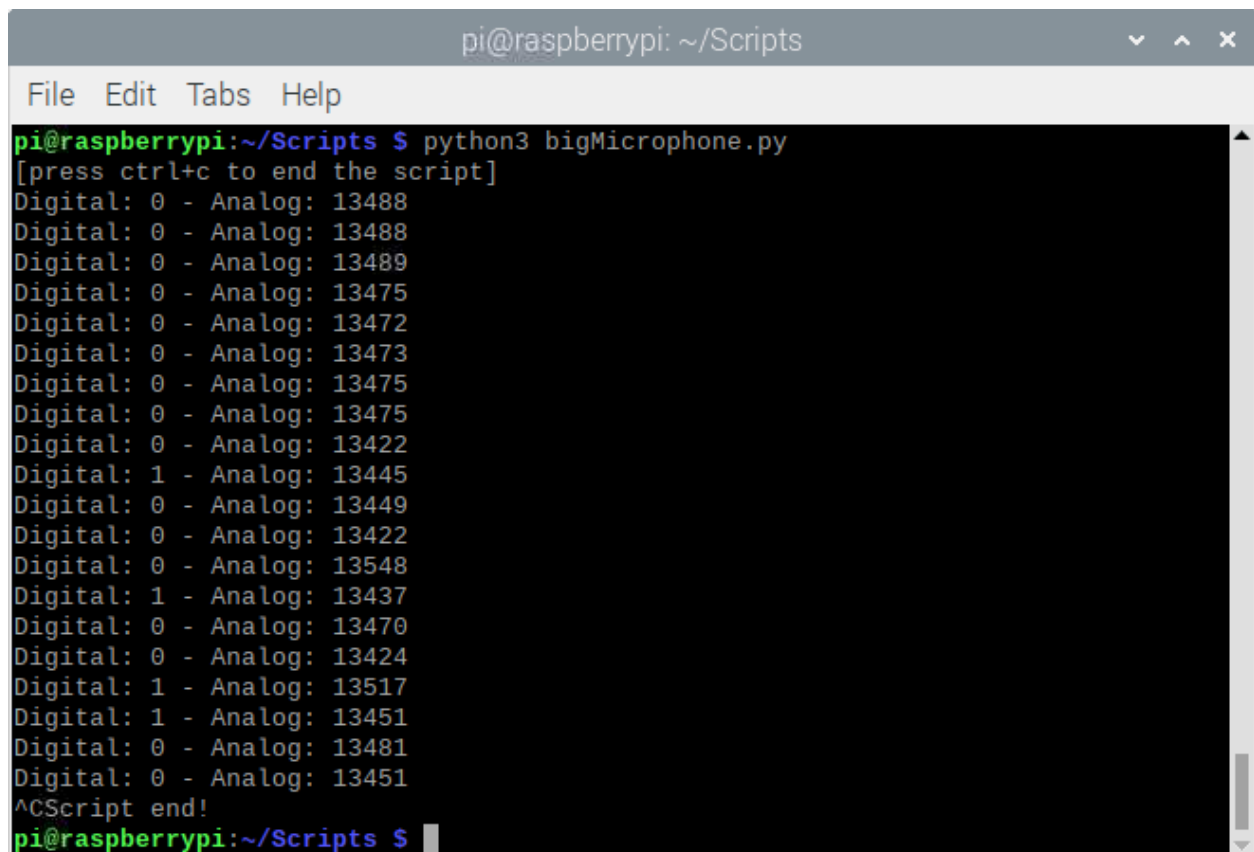
except KeyboardInterrupt:
    print('\nScript end!')

finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *bigMicrophone.py*. To run the script open the terminal in the directory where the script is saved and run the following command: **python3 bigMicrophone.py**

The result should look like the output on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 bigMicrophone.py
[press ctrl+c to end the script]
Digital: 0 - Analog: 13488
Digital: 0 - Analog: 13488
Digital: 0 - Analog: 13489
Digital: 0 - Analog: 13475
Digital: 0 - Analog: 13472
Digital: 0 - Analog: 13473
Digital: 0 - Analog: 13475
Digital: 0 - Analog: 13475
Digital: 0 - Analog: 13422
Digital: 1 - Analog: 13445
Digital: 0 - Analog: 13449
Digital: 0 - Analog: 13422
Digital: 0 - Analog: 13548
Digital: 1 - Analog: 13437
Digital: 0 - Analog: 13470
Digital: 0 - Analog: 13424
Digital: 1 - Analog: 13517
Digital: 1 - Analog: 13451
Digital: 0 - Analog: 13481
Digital: 0 - Analog: 13451
^CScript end!
pi@raspberrypi:~/Scripts $
```

To end the script press *CTRL + C* on the keyboard.



KY-038 Small Microphone Sensor Module

Introduction

The KY-038 small microphone module is the same as the KY-037 big microphone module. The only difference is in the size of the on-board microphone sensor. The KY-038 module has a smaller microphone than the microphone on-board the KY-037 module.



KY-039 Heartbeat Sensor Module

Introduction

The KY-039 heartbeat sensor module has an IR LED, a photosensitive diode and a resistor on-board. The output of the module is an analog voltage that represents how much the infrared light a photosensitive diode receives. The higher the value, the stronger the intensity of the infrared light.

Place a finger between the IR LED and the photosensitive diode of the module. Heartbeats dilate the blood vessels in a finger, which filters the infrared light at different levels. This creates a pulsating signal which can be seen on the image of Serial Plotter (which is shown later in the text).

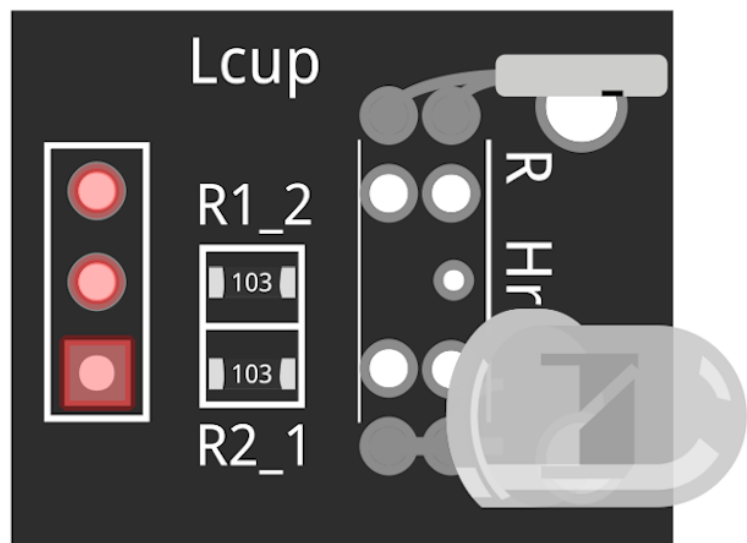
Specifications

Operating voltage range	from 3.3V to 5V DC
Operating temperature range	from -40°C to 85°C
Dimensions	19x15mm [0.73x0.6in]

The pinout

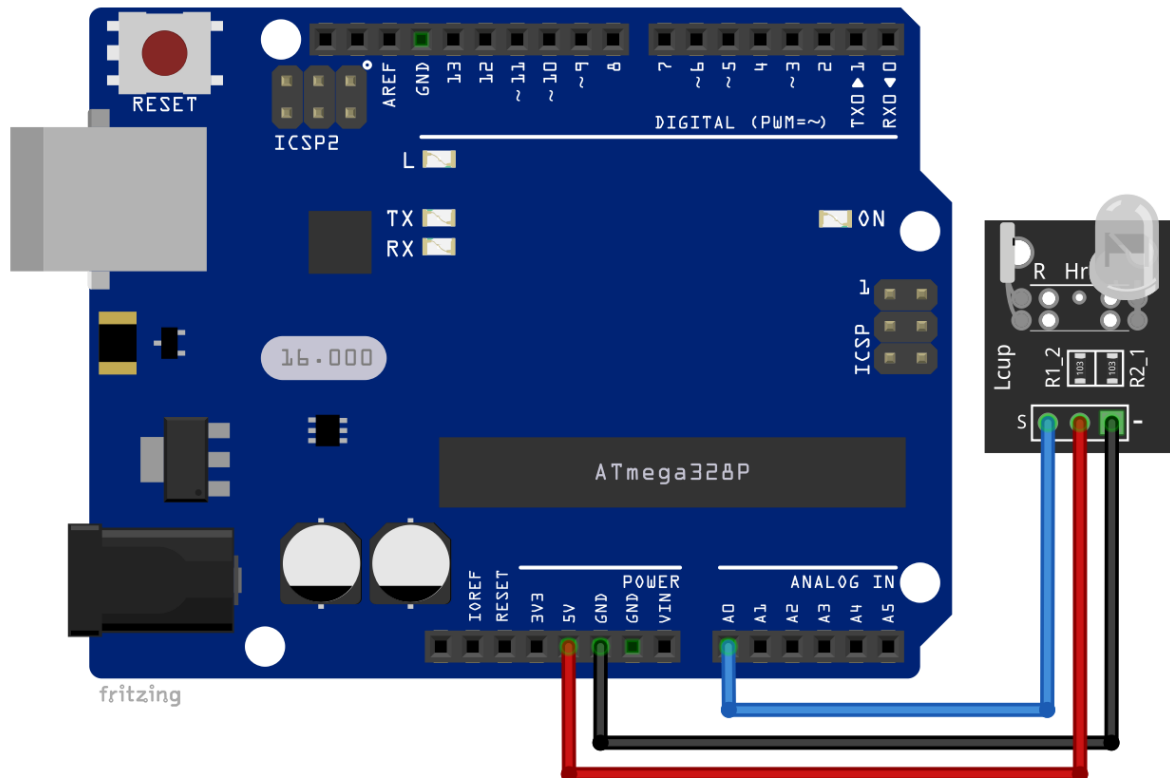
The KY-039 heartbeat sensor module has three pins. The pinout diagram is shown on the following image:

Output signal - S
Power supply - VCC
Ground - GND



Connecting the module with Uno

Connect the KY-039 module with the Uno as shown on the following connection diagram:



KY-039 pin	Uno pin	Wire color
S	A0	Blue wire
- (GND)	GND	Black wire
Middle pin (VCC)	5V	Red wire

Az-Delivery

Sketch example

```
#define ANALOG_PIN 0

float average = 0.0;

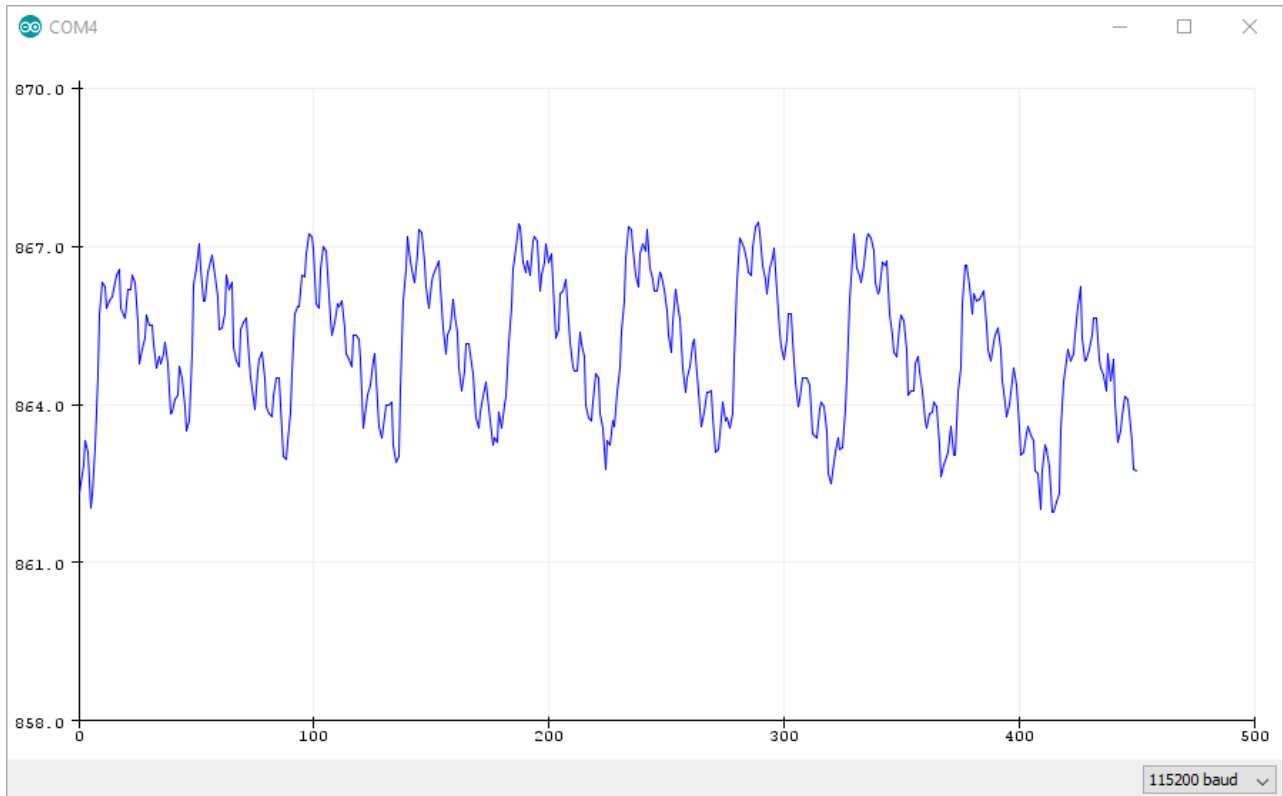
void setup() {
  Serial.begin(9600);
}

void loop() {
  for(uint8_t i = 0; i <= 19; i++) {
    average = average + analogRead(0);
    delay(1);
  }
  average = average / 20.0;

  Serial.println(average);
  average = 0.0;
}
```

Az-Delivery

Upload the sketch to the Uno and open the Serial Plotter (*Tools > Serial Plotter*). The result should look like the output on the following image:



Az-Delivery

On the image of a Serial Plotter, you can easily see the heartbeats (larger peaks). All that has to be done is to smooth the signal and then to count the heartbeats or peaks on the diagram curve. This is done in the Python script of the *Connecting the module with Raspberry Pi* chapter.

NOTE: To get these values, protect the sensor from any other light source. The nontransparent box can be created where the sensor can be protected from the external light source. If the sensor is not hidden from other light sources, it reads the infrared light from other light sources which distort the output signal. If that happens the signal has to be cleaned from that distortions in order for measurements to be useful.

Az-Delivery

Connecting the module with Raspberry Pi

Connect the KY-039 module with the Uno as shown on the connection diagram from the chapter *Connecting the module with Uno*, and then connect the Uno with the Raspberry Pi via USB cable. Next, upload *nanpy* firmware to the Uno (as shown in the chapter *Uno as ADC*), and use the following code to control the KY-039 module:

```
from nanpy import (ArduinoApi, SerialManager)
from time import sleep
try:
    connection1 = SerialManager()
    a = ArduinoApi(connection=connection1)
except:
    print('Failed to connect to the Arduino')

value = 0
print('[Press CTRL + C to end the script!}')
try:
    while True:
        value = a.analogRead(0)
        print('Output: {}'.format(value))
        sleep(0.01)

except KeyboardInterrupt:
    print('\nScript end!')
```

Az-Delivery

Save the script by the name *Heartbeat.py*. To run the script open the terminal in the directory where the scripts is saved and run the following command:

```
python3 Heartbeat.py
```

To end the script press *CTRL + C* on the keyboard.



Full Python script

Now, a new script will be created, to read and smooth the signal and then to count the heartbeats. In the previous Python script, *Heartbeat.py* the analog signal is read and the values are displayed in the terminal.

First, a library and a tool has to be installed. If the *pip3* tool is not installed, open the terminal and run the following commands, one by one:

```
sudo apt-get update
```

```
sudo apt-get install python3-pip
```

Now, install the library for plotting in Python by running the following command in the terminal:

```
pip3 install matplotlib
```

and wait for the installation to finish (it can last more than *10* minutes).

Az-Delivery

```
from nanpy import (ArduinoApi, SerialManager)
import time
import matplotlib.pyplot as plt
import os
def plot_em(et):
    elapsed_time = time.time() - et
    print('Script end! Elapsed time
{:.3f}'.format(elapsed_time))
    y = []
    with open('new_data.txt', 'rt') as reader:
        lines = reader.readlines()
        for line in lines:
            y.append(float(line))
    x = range(len(y))
    plt.plot(x, y)
    plt.show()

try:
    connection1 = SerialManager()
    a = ArduinoApi(connection=connection1)
except:
    print('Failed to connect to the Arduino')

check_file = False
sum_a = 0
sample_range = 44
last, new = 0.0, 0.0
last_rise, new_rise = False, False
count_beats, last_beats = 0, 0
measurements = 0
```

Az-Delivery

```
elapsed = time.time() # used to stop script
print('[Press CTRL + C to end the script!]\n')
try:
    t = time.time()
    while True:
        sum_a += a.analogRead(0) # sensor is connected to A0
        time.sleep(0.001)
        measurements += 1
        if measurements == sample_range:
            new = sum_a / sample_range
            if not check_file:
                if os.path.isfile('new_data.txt'):
                    os.remove('new_data.txt')
                check_file = True
            with open('new_data.txt', 'at') as f:
                f.write('{}\n'.format(new))
        else:
            with open('new_data.txt', 'at') as f:
                f.write('{}\n'.format(new))
            sum_a = 0
            measurements = 0
    if new > last:
        rise = True
        if new < last:
            rise = False
        new_rise = rise
        if last_rise == True and new_rise == False:
            count_beats += 1
        last = new
        last_rise = new_rise
```


Az-Delivery

2 tabs

```
    if last_beats != count_beats:
        print(count_beats)
        last_beats = count_beats
    if time.time() - elapsed > 15.0: # stop after 15sec
        plot_em(t)
        break

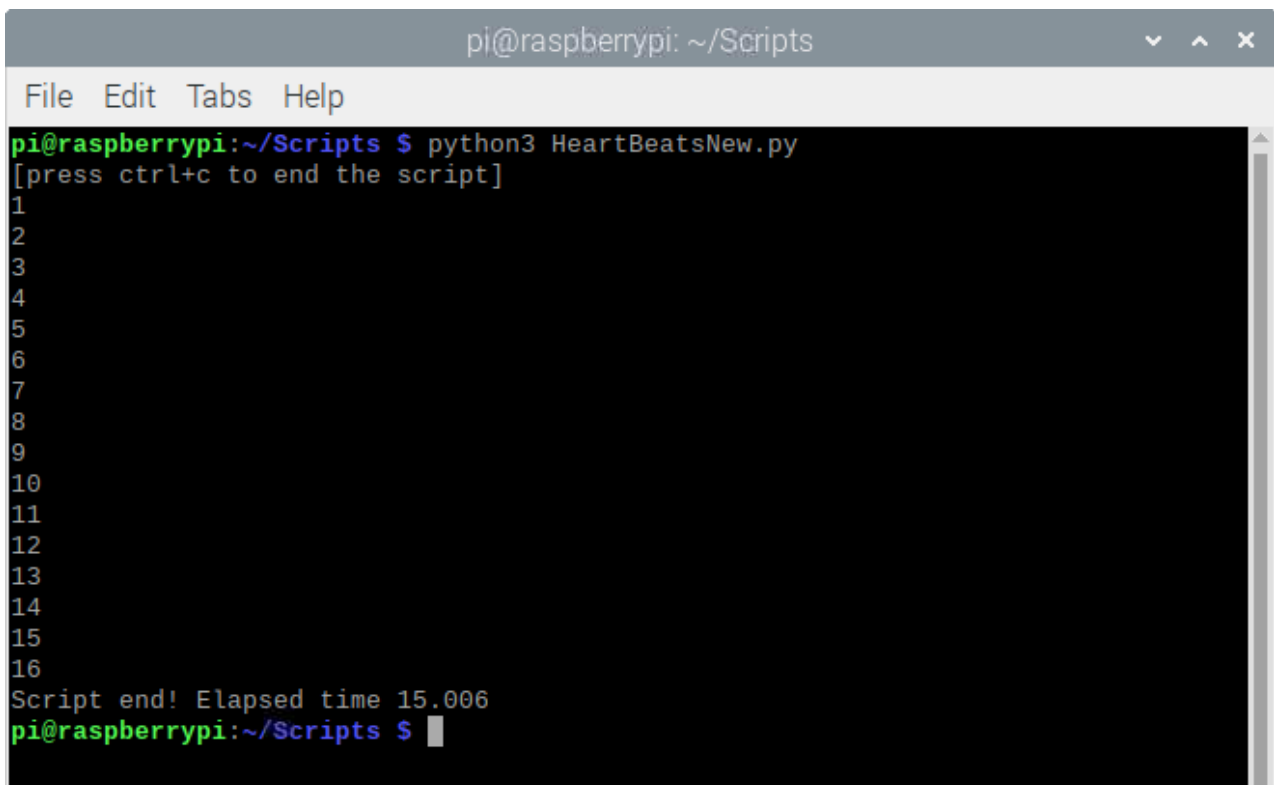
except KeyboardInterrupt:
    plot_em(t)
```

Az-Delivery

Save the script by the name *HeartbeatsNew.py*. To run the script open the terminal in the directory where the script is saved and run the following command:

python3 HeartbeatsNew.py

After 15 seconds script will end and the plot diagram is displayed on the screen. The result in the terminal should look like the output on the following image:

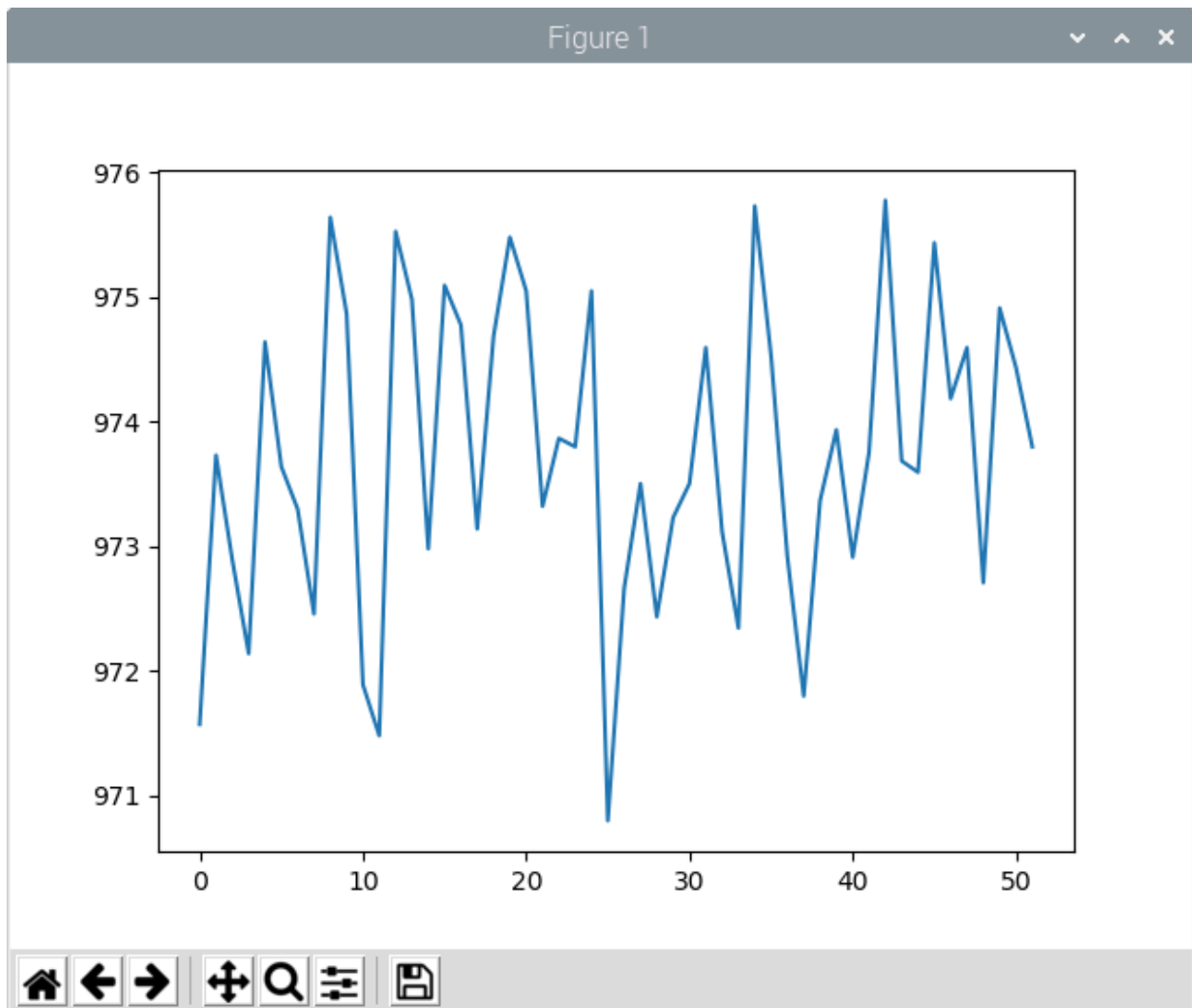
A screenshot of a terminal window titled 'pi@raspberrypi: ~/Scripts'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal shows the command 'python3 HeartBeatsNew.py' being executed. Below the command, there is a list of numbers from 1 to 16, representing the output of the script. At the bottom, it says 'Script end! Elapsed time 15.006' followed by the prompt 'pi@raspberrypi:~/Scripts \$' and a cursor.

```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 HeartBeatsNew.py
[press ctrl+c to end the script]
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Script end! Elapsed time 15.006
pi@raspberrypi:~/Scripts $
```

To close the window of the plot diagram press *Q* letter on the keyboard or close it with the mouse by clicking “x”.

Az-Delivery

What the script does is counting and displaying heartbeats in the terminal with the error of ± 2 beats per 20 beats. The script ends itself after 15 seconds and the plot diagram should look like the diagram on the following image:



Az-Delivery

First, the measurements are done on every *1ms*. After a number of measurements, the measurements get averaged. The average value gets saved in the file called *new_data.txt*. The number of measurements is saved in the *sample_range* variable that can be changed in order to adjust the accuracy of the counting beats.

Next, peaks on the diagram curve are detected. It is assumed that every peak is one heartbeat (which has to be tested more). After this, these peaks are counted and the number of peaks is displayed in the terminal.

To plot everything, the data has to be stored in a file. In this case the data is stored in the file called *new_data.txt*. The data from that file is read in the *plot_em()* function after which the plot diagram is displayed.

At the end of the *try* block, in the last *if* statement the time interval for script execution is created. Time interval is set to *15* seconds, but it can be easily changed to any other value:

```
if time.time() - elapsed > 15.0: # stop after 15sec
    plot_em(t)
    break
```

The script execution can be ended at any time by pressing *CTRL + C* on the keyboard, after which the plot diagram is shown.

NOTE: To get these values, protect the sensor from any other light source. The nontransparent box can be created where the sensor can be protected from the external light source. If the sensor is not hidden from other light sources, the sensor reads the infrared light from other light sources which distort the output signal. If that happens the signal has to be cleaned from that distortions in order for measurements to be useful.



KY-040 Rotary Encoder Module

Introduction

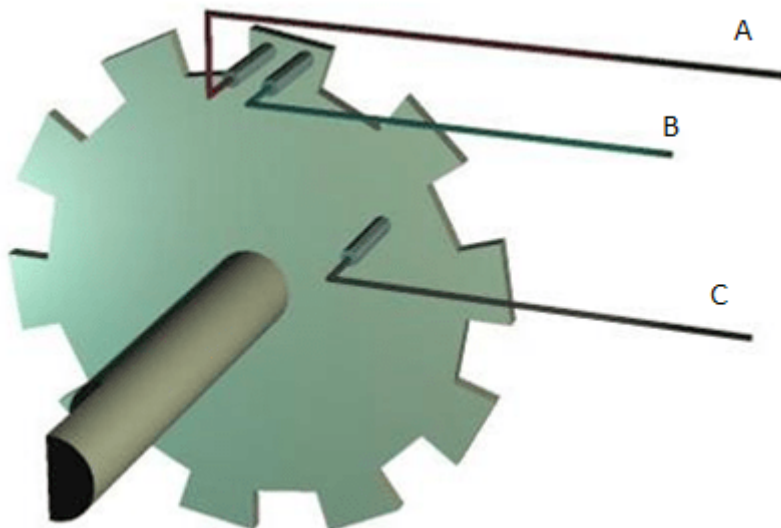
The KY-040 rotary encoder module is a rotary input device (as in knob) that provides an indication of how much the knob has been rotated and in what direction it is rotating. A rotary encoder is an electro-mechanical device that converts rotational motion into digital or analog information. It is a great device for stepper and servo motor control.

A rotary encoder has a fixed number of positions per revolution. These positions are easily felt as small “clicks” when you turn the encoder shaft. The KY-040 rotary encoder module has 20 positions per one full rotation.

Working principle

There are two switches inside the encoder. One switch connects pin A (or CLK) to pin C and the other switch connects pin B (or DT) to C. In each encoder position, both switches are either opened or closed.

The illustration below is the representation of how the switch is constructed.

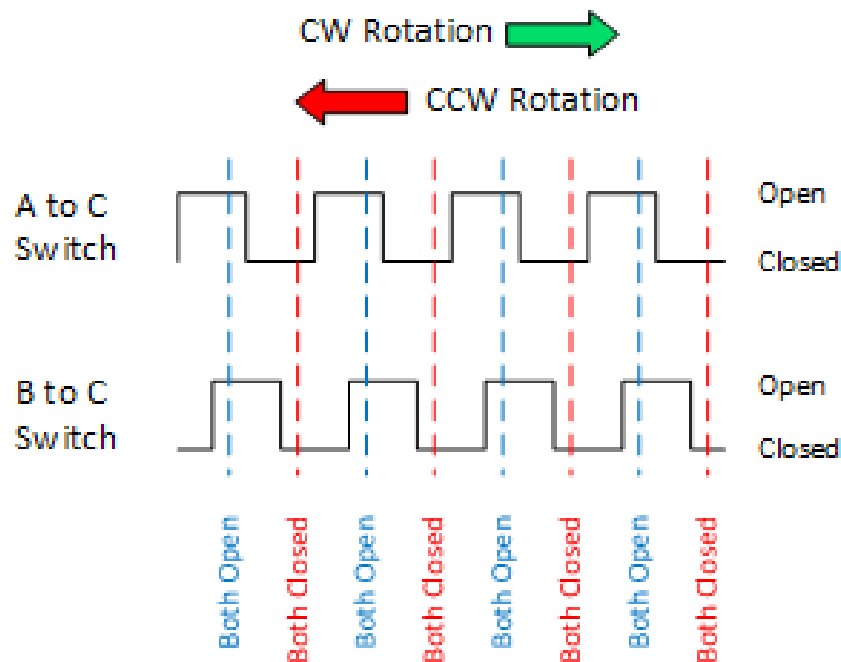


As you can see, the angular position of the A terminal and the B terminal is the following:

- » Rotating the switch clockwise causes the switch connecting A and C to change states first.
- » Rotating the switch counter-clockwise causes the switch connecting B and C to change states first.

Az-Delivery

If the opening and closing of the switches is represented as wave forms, it would look something as shown on the following image:



Essentially, determining which switch changed states first is how the direction of rotation is determined.

- » If A changed states first, the switch is rotating in a clockwise direction.
- » If B changed states first, the switch is rotating in a counter-clockwise direction.



The output

The module is designed so that the *LOW* state is an output when the switches are closed and the *HIGH* state when the switches are open. The *LOW* state is generated by placing a ground at pin *C* and passing it to the *CLK* and *DT* pins when switches are closed. The *HIGH* state is generated connecting the power supply via pull-up resistors to the *CLK* and *DT* pins when switches are open.

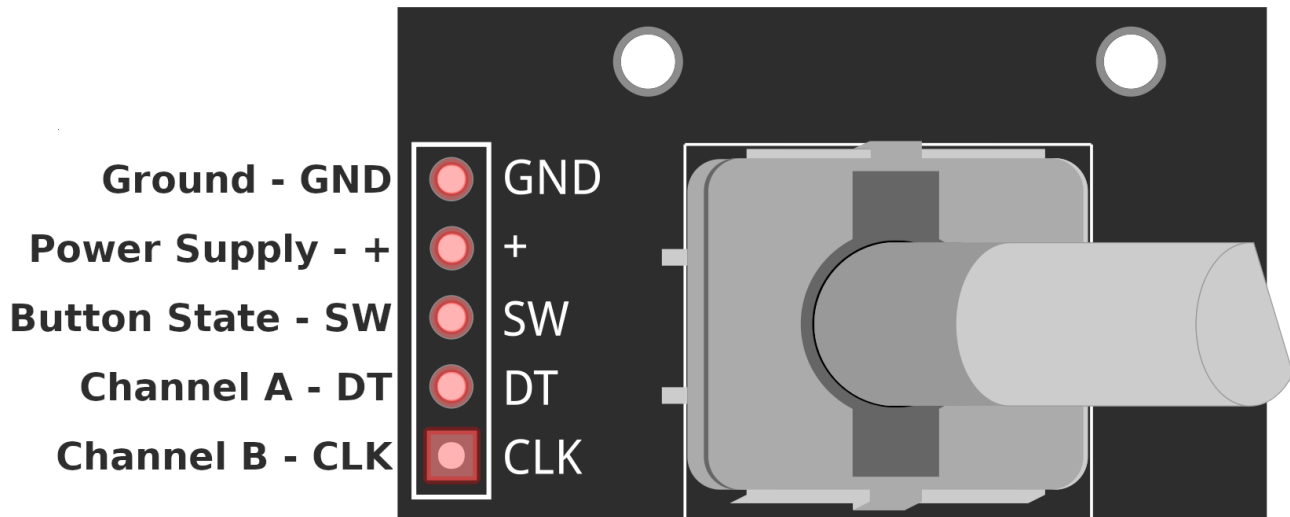
There is a push button that is integral to the encoder. If the shaft is pressed, a normally open switch closes. The *SW* pin outputs the state of this push button.

Specifications

Operating voltage range	from 3.3V to 5V DC
Operating temperature range	from -40°C to 85°C
Output	digital
Dimensions	17x29x30mm [0.7x1.14x1.2in]

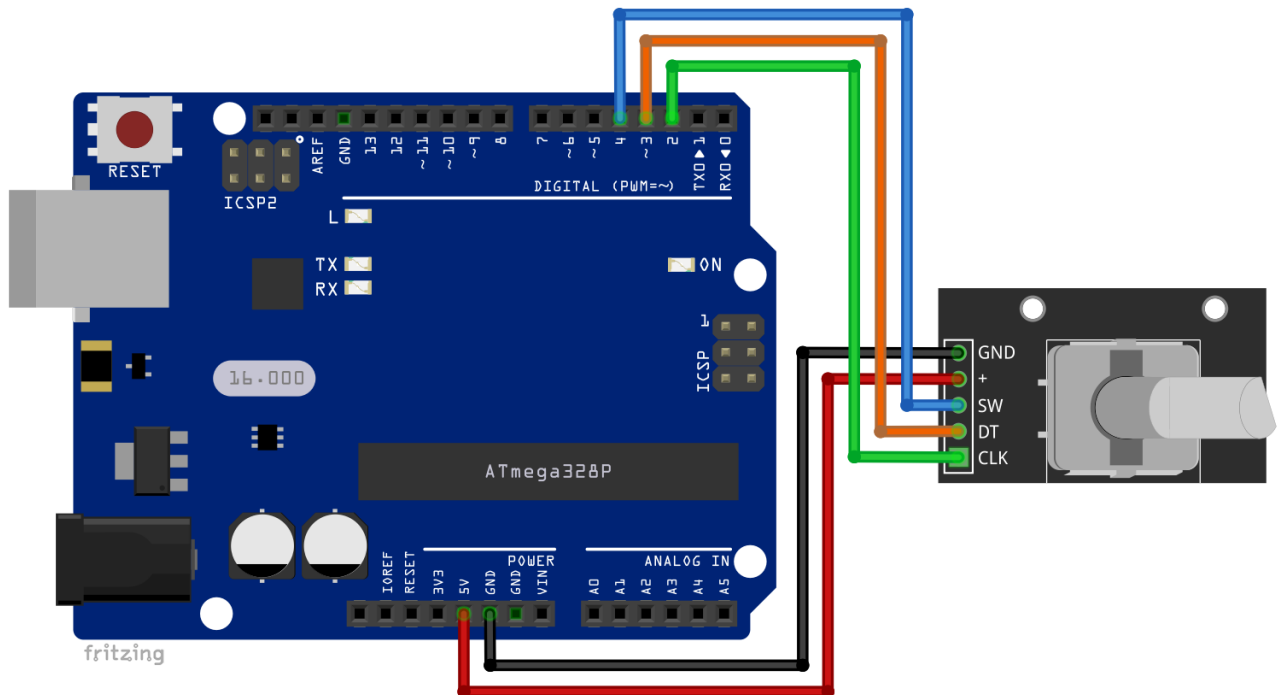
The pinout

The KY-040 rotary encoder module has five pins. The pinout diagram is shown on the following image:



Connecting the module with Uno

Connect the KY-040 module with the Uno as shown on the following connection diagram:



KY-040 pin	Uno pin	Wire color
+ (VCC)	5V	Red wire
GND	GND	Black wire
CLK	D2	Green wire
DT	D3	Orange wire
SW	D4	Blue wire

Az-Delivery

Sketch example

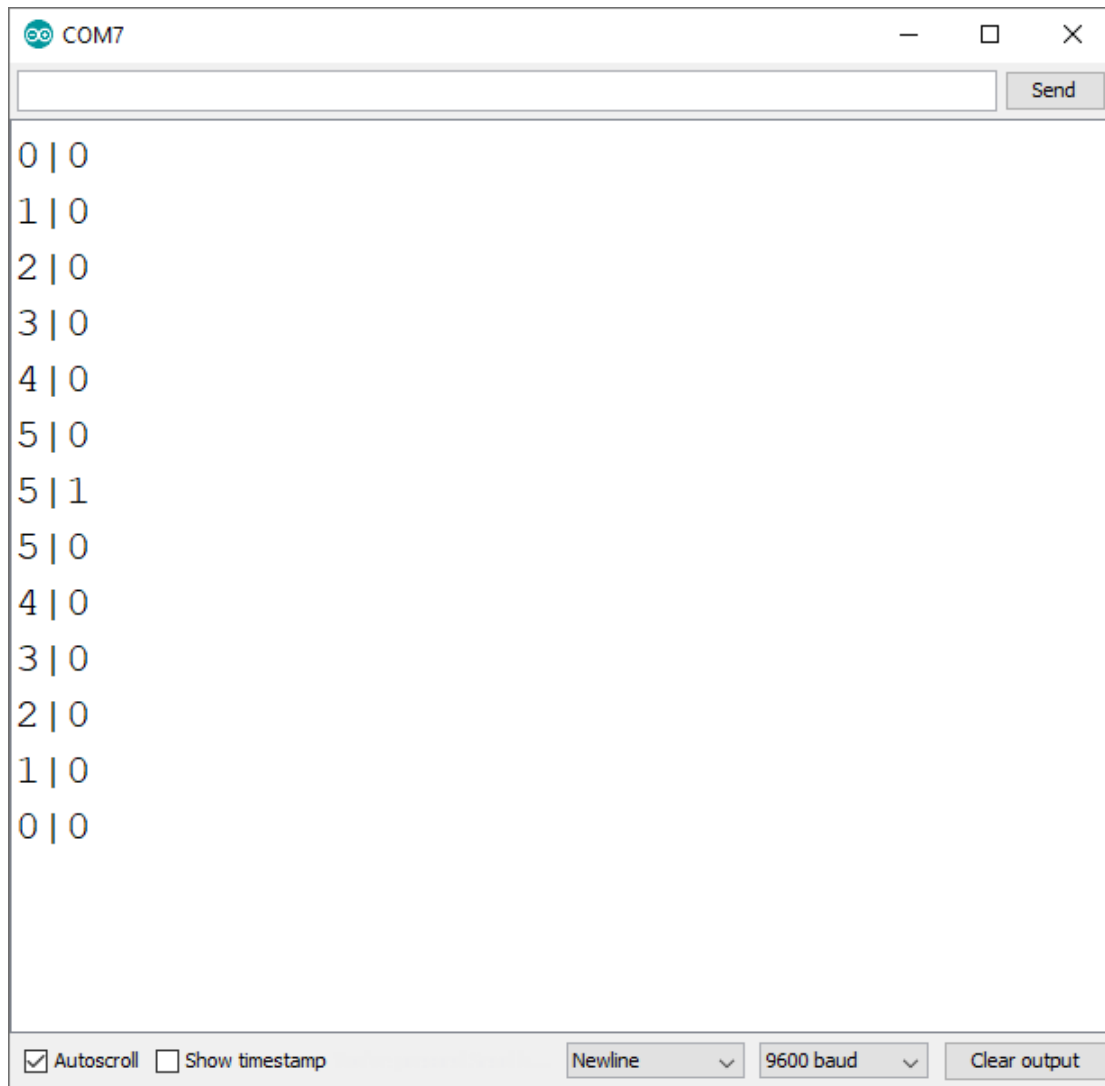
```
#define CLK_PIN 2
#define DT_PIN 3
#define SW_PIN 4
int position = 0;
int last_position = 0;
int n = 0;
bool taster = LOW;
bool last_taster = LOW;
void setup() {
    pinMode(CLK_PIN, INPUT_PULLUP);
    pinMode(DT_PIN, INPUT_PULLUP);
    pinMode(SW_PIN, INPUT_PULLUP);
    Serial.begin(9600);
}
void loop() {
    n = digitalRead(CLK_PIN);
    taster = !digitalRead(SW_PIN);
    if(taster != last_taster) {
        Serial.print(position);
        Serial.print("|");
        Serial.println(taster);
        delay(10);
        last_taster = taster;
    }
}
```

Az-Delivery

```
// one tab
if((last_position == 0) && (n == HIGH)) {
    if(digitalRead(DT_PIN) == LOW) {
        position++;
    }
    else {
        position--;
    }
    Serial.print(position);
    Serial.print("|");
    Serial.println(taster);
}
last_position = n;
}
```

Az-Delivery

Upload the sketch to the Uno and open Serial Monitor (*Tools > Serial Monitor*). The result should look like the output on the following image:



To get these values, move or press the shaft of the rotary encoder.

NOTE: There is a lot of noise in the output of the module. A high number of errors is expected with high noise. How to clear output out of the noise is not in the scope of this eBook.

Az-Delivery

The sketch starts with creating three macros called *CLK_PIN*, *DT_PIN* and *SW_PIN*. These macros represent the digital pins of Uno on which pins of the module are connected.

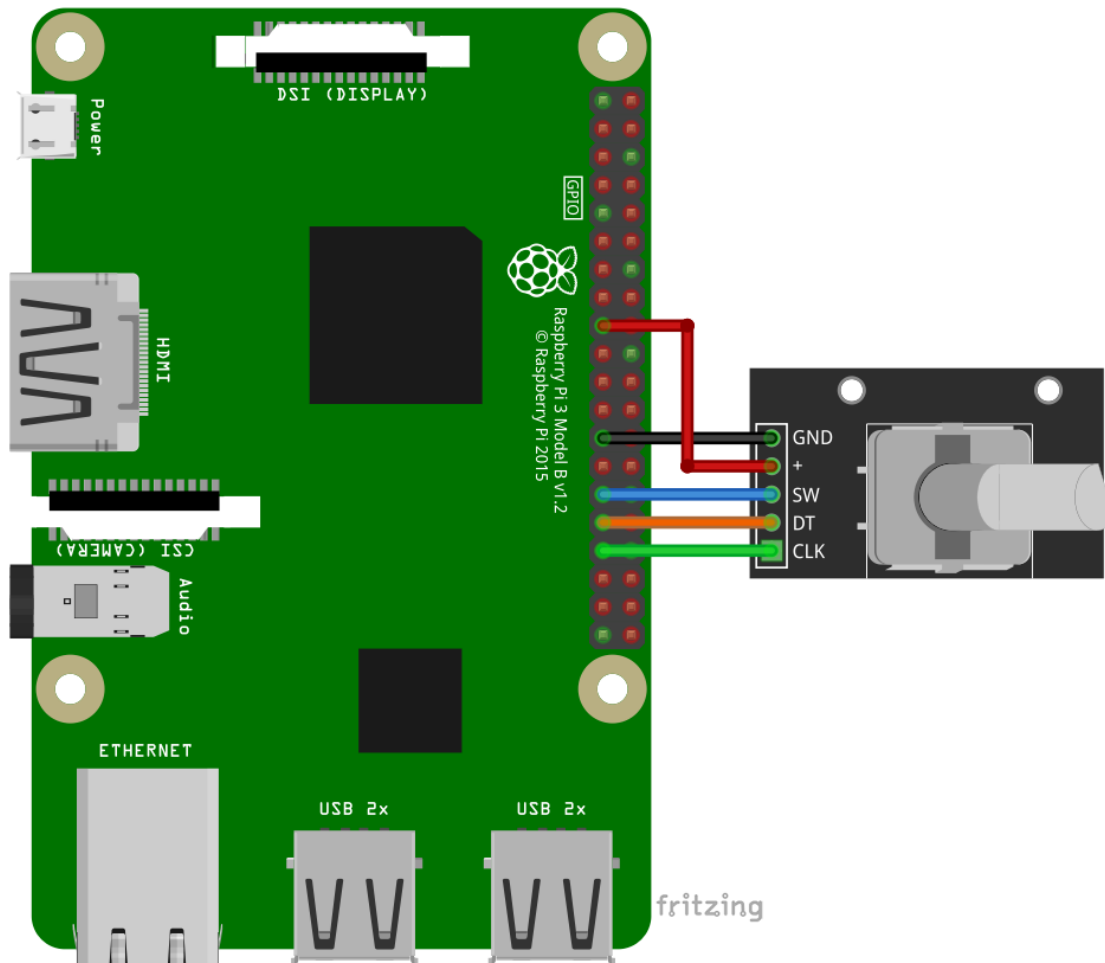
Next, several variables are created which are used in the algorithm that detect when and in which direction the shaft of the encoder is moving. The variable called *position* holds the value which represents the current position of the shaft. At the beginning of the sketch execution this value is zero, but when the shaft of the encoder is moved, it can hold both positive or negative values depending on the direction of moving shaft (positive value - clockwise direction, negative value - counterclockwise direction).

In the *setup()* function the pin modes for all three pins are set to *INPUT_PULLUP*. Next, the serial communication is started with a baud rate of *9600bps*.

In the *loop()* function the state of the module pins is read. Then, the algorithm for detecting movement of the shaft is executed, checking the previously read states of the pins. This algorithm is not covered in this eBook. The output of this algorithm is a message displayed in the Serial Monitor (image of Serial Monitor on the previous page).

Connecting the module with Raspberry Pi

Connect the module with the Raspberry Pi as shown on the following connection diagram:



KY-040 pin	Raspberry Pi pin	Physical pin	Wire color
+ (VCC)	3V3	17	Red wire
GND	GND	25	Black wire
SW	GPIO5	29	Orange wire
DT	GPIO6	31	Blue wire
CLK	GPIO13	33	Green wire

Az-Delivery

Python script

```
import RPi.GPIO as GPIO
from time import sleep

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

SW_PIN = 5
DT_PIN = 6
CLK_PIN = 13

GPIO.setup(CLK_PIN, GPIO.IN)
GPIO.setup(DT_PIN, GPIO.IN)
GPIO.setup(SW_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)

position = 0

a = 0
b = 0

a_last = 1
b_last = 1

taster = False
last_taster = False
```

Az-Delivery

```
print('[Press CTRL + C to end the script!]\n')
try:
    while True:
        a, b = GPIO.input(CLK_PIN), GPIO.input(DT_PIN)
        taster = not GPIO.input(SW_PIN)
        if taster != last_taster:
            print('{}|{}'.format(position, taster))
            sleep(0.01)
            last_taster = taster
        if a != a_last or b != b_last:
            if a == 0 and b == 1:
                while not (a == 1 and b == 1):
                    a = GPIO.input(CLK_PIN)
                    b = GPIO.input(DT_PIN)
                position += 1
                print('{}|{}'.format(position, taster))
            if a == 1 and b == 0:
                while not (a == 1 and b == 1):
                    a = GPIO.input(CLK_PIN)
                    b = GPIO.input(DT_PIN)
                position -= 1
                print('{}|{}'.format(position, taster))
            a_last = a
            b_last = b

except KeyboardInterrupt:
    print('\nScript end!\n')

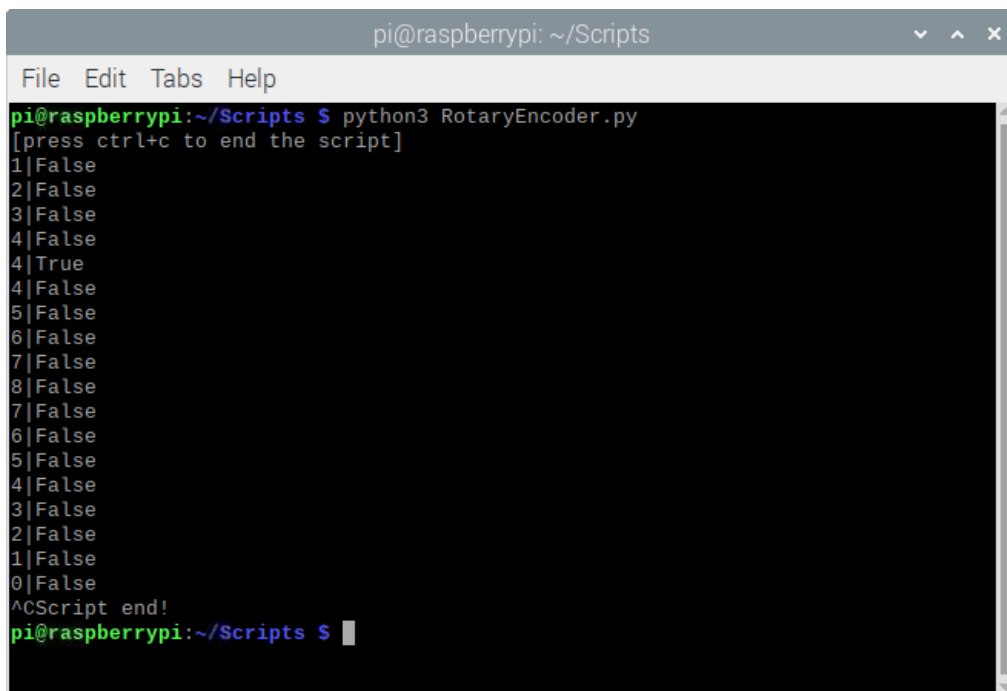
finally:
    GPIO.cleanup()
```

Az-Delivery

Save the script by the name *RotaryEncoder.py*. To run the script, open the terminal in the directory where the script is saved and run the following command:

python3 RotaryEncoder.py

The result should look like the output on the following image:



```
pi@raspberrypi: ~/Scripts
File Edit Tabs Help
pi@raspberrypi:~/Scripts $ python3 RotaryEncoder.py
[press ctrl+c to end the script]
1|False
2|False
3|False
4|False
4|True
4|False
5|False
6|False
7|False
8|False
7|False
6|False
5|False
4|False
3|False
2|False
1|False
0|False
^CScript end!
pi@raspberrypi:~/Scripts $
```

To stop the script press *CTRL* + *C* on the keyboard. To get these values, move or press the shaft of the rotary encoder.

The script does the same thing on the same way as the sketch for the Uno, the only difference is that the script is written in *Python*.

NOTE: There is a lot of noise in the output of the module. A high number of errors is expected with high noise. How to clear output out of the noise is not in the scope of this eBook.

Az-Delivery

Now it is the time to learn and make your own projects. You can do that with the help of many example scripts and other tutorials, which can be found on the internet.

If you are looking for the high quality products for Arduino and Raspberry Pi, AZ-Delivery Vertriebs GmbH is the right company to get them from. You will be provided with numerous application examples, full installation guides, eBooks, libraries and assistance from our technical experts.

<https://az-delivery.de>

Have Fun!

Impressum

<https://az-delivery.de/pages/about-us>