目录

[Raspberry Pi 4B Advanced Kit Projects Manual 4](#_Toc139296706)

[Description 4](#_Toc139296707)

[Features 4](#_Toc139296708)

[Package Lists 5](#_Toc139296709)

[**Getting Start** 6](#_Toc139296710)

[**Install and Setup Raspberry Pi OS** 6](#_Toc139296711)

[**Optional：** 8](#_Toc139296712)

[Project 1 10](#_Toc139296713)

[WS2812B protocol 11](#_Toc139296714)

[What You Need 11](#_Toc139296715)

[Assembling Hardware 12](#_Toc139296716)

[Set up of software. 13](#_Toc139296717)

[Flashing Led Demonstration 14](#_Toc139296718)

[Basic Code for WS2812B LED 18](#_Toc139296719)

[Control Several LED Light ring s at Once 20](#_Toc139296720)

[Project 2 22](#_Toc139296721)

[Overview of MPU6050 Sensor 22](#_Toc139296722)

[Connection Diagram of MPU6050 with Raspberry Pi 23](#_Toc139296723)

[MPU6050 Example using Raspberry Pi 24](#_Toc139296724)

[Python based I2C functions for Raspberry Pi 24](#_Toc139296725)

[Python based I2C Functions 24](#_Toc139296726)

[MPU6050 Code for Raspberry Pi using Python 26](#_Toc139296727)

[MPU6050 Code for Raspberry Pi using C (WiringPi Library) 27](#_Toc139296728)

[MPU6050 Output 30](#_Toc139296729)

[Project 3 30](#_Toc139296730)

[Interface a 1-Channel Relay with Raspberry Pi 4B 30](#_Toc139296731)

[Components 31](#_Toc139296732)

[Raspberry Pi as a Device Switch 31](#_Toc139296733)

[How do relays work? 32](#_Toc139296734)

[Using a Relay as a Switching Circuit 33](#_Toc139296735)

[Working of Relays 34](#_Toc139296736)

[An explanation of relay control 35](#_Toc139296737)

[Demo code 35](#_Toc139296738)

[Project 4 36](#_Toc139296739)

[Principle: 36](#_Toc139296740)

[How to wiring it 37](#_Toc139296741)

[Demo Code: 37](#_Toc139296742)

[Project 5 39](#_Toc139296743)

[How Electret Microphone Work 39](#_Toc139296744)

[The Microphone Breakout 39](#_Toc139296745)

[Using The Microphone Breakout with Raspberry Pi 39](#_Toc139296746)

[Demo Code 40](#_Toc139296747)

[Project 6 41](#_Toc139296748)

[Raspberry Pi Music Player 41](#_Toc139296749)

[Materials: 41](#_Toc139296750)

[Steps: 41](#_Toc139296751)

[Demo code 42](#_Toc139296752)

[Projects 7 45](#_Toc139296753)

[Materials: 45](#_Toc139296754)

[Steps: 46](#_Toc139296755)

[Demo code 47](#_Toc139296756)

[Project 8 48](#_Toc139296757)

[Isolated Digital Input: 48](#_Toc139296758)

[Optically Isolated Serial Communication: 48](#_Toc139296759)

[Isolated I2C Communication: 49](#_Toc139296760)

[Isolated PWM Output: 49](#_Toc139296761)

[Optically Isolated Interrupts: 49](#_Toc139296762)

[Components: 49](#_Toc139296763)

[Circuit Diagram: 50](#_Toc139296764)

[Circuit Connections: 50](#_Toc139296765)

[Pin Descriptions: 51](#_Toc139296766)

[Demo code 51](#_Toc139296767)

[Project 9 52](#_Toc139296768)

[Connecting DHT11 Sensor with Raspberry Pi 4/3 using Python 52](#_Toc139296769)

[Parts List 52](#_Toc139296770)

[Raspberry Pi Pin Configuration 53](#_Toc139296771)

[Connection Diagram 53](#_Toc139296772)

[Installation of required Libraries 54](#_Toc139296773)

[How to run the same code in Raspberry Pi 4? 55](#_Toc139296774)

[Alternate way to edit this file 57](#_Toc139296775)

[Coding 57](#_Toc139296776)

[Demo Code: 58](#_Toc139296777)

[Construction 58](#_Toc139296778)

[Executing the Code & Testing 58](#_Toc139296779)

[Conclusion 60](#_Toc139296780)

[Project 10 60](#_Toc139296781)

[Components needed: 62](#_Toc139296782)

[Circuit Diagram: 63](#_Toc139296783)

[Procedure: 63](#_Toc139296784)

[Project 12 64](#_Toc139296785)

[1. Connect the hardware components: 64](#_Toc139296786)

[2. Set up the Raspberry Pi: 65](#_Toc139296787)

[3. Install required Python libraries: 65](#_Toc139296788)

[4. Create a new Python script: 65](#_Toc139296789)

[5. Copy and paste the following demo code into the new file: 65](#_Toc139296790)

[6. Save the file and close the text editor. 66](#_Toc139296791)

[7. Run the script: 66](#_Toc139296792)

[Project 13 67](#_Toc139296793)

[Required components: 67](#_Toc139296794)

[Steps: 67](#_Toc139296795)

[Project 14 70](#_Toc139296796)

[CONNECT THE LCD 70](#_Toc139296797)

[CONNECTING AN I2C ENABLED LCD 71](#_Toc139296798)

[ENABLE I2C ON THE PI 71](#_Toc139296799)

[ENABLE I2C IN RASPI-CONFIG 71](#_Toc139296800)

[INSTALL I2C-TOOLS AND SMBUS 72](#_Toc139296801)

[PROGRAMMING THE LCD 73](#_Toc139296802)

[INSTALLING THE LIBRARY 73](#_Toc139296803)

[WRITE TO THE DISPLAY 78](#_Toc139296804)

[POSITION THE TEXT 78](#_Toc139296805)

[CLEAR THE SCREEN 79](#_Toc139296806)

[BLINKING TEXT 79](#_Toc139296807)

[PRINT THE DATE AND TIME 80](#_Toc139296808)

[PRINT YOUR IP ADDRESS 80](#_Toc139296809)

[SCROLL TEXT RIGHT TO LEFT CONTINUOUSLY 81](#_Toc139296810)

[SCROLL TEXT RIGHT TO LEFT ONCE 81](#_Toc139296811)

[SCROLL TEXT LEFT TO RIGHT ONCE 82](#_Toc139296812)

[CUSTOM CHARACTERS 82](#_Toc139296813)

[PRINTING A SINGLE CUSTOM CHARACTER 82](#_Toc139296814)

[PRINTING MULTIPLE CUSTOM CHARACTERS 83](#_Toc139296815)

[PRINT DATA FROM A SENSOR 85](#_Toc139296816)

[Project 15 86](#_Toc139296817)

[Connect the PIR motion sensor 86](#_Toc139296818)

[About PIR motion sensors 86](#_Toc139296819)

[Connect a PIR motion sensor 86](#_Toc139296820)

[Connect the camera 88](#_Toc139296821)

[Test the PIR motion sensor 91](#_Toc139296822)

[Set up your PIR sensor on GPIO 4. 92](#_Toc139296823)

[While True loop in Python 93](#_Toc139296824)

[Set up the camera preview 94](#_Toc139296825)

[Taking a photograph with PiCamera 95](#_Toc139296826)

[Record video in a file 96](#_Toc139296827)

[Creating timestamps with Python 98](#_Toc139296828)

[Play video 99](#_Toc139296829)

[Project 16 99](#_Toc139296830)

[hardware parts are needed: 99](#_Toc139296831)

[Hardware Connections： 100](#_Toc139296832)

[Preparation： 100](#_Toc139296833)

# Raspberry Pi 4B Advanced Kit Projects Manual

## Description

The kit includes a variety of components and modules for building electronic projects. It contains an MPU6050 module, a 1-channel relay, a tilt sensor, a sound sensor, a T-type GPIO expansion board, a GPIO edge expansion board, an ultrasonic sensor, a DHT11 temperature and humidity sensor, an LCD1602 display module, a PIR sensor, a servo motor, a long breadboard, 40-pin jump wires, a WS2812 RGB light ring, an experiment platform plate, a 500W camera, R/G/B LED lights, a MAX7219 8x8 matrix, and S8050 and 2N9304 NPN transistors. Additionally, it includes 220 ohms and 1k ohm resistors.

## Features

* MPU6050: A motion-tracking device that combines a 3-axis gyroscope and a 3-axis accelerometer.
* 1-channel relay: Allows you to control high-power devices using a low-power signal.
* Tilt sensor: Detects changes in orientation and can be used to trigger actions based on tilt.
* Sound sensor: Detects sound levels and can be used for sound-activated projects.
* T-type GPIO expansion board: Provides additional GPIO pins for connecting more components.
* GPIO edge expansion board: Expands the GPIO pins of a microcontroller for connecting more devices.
* Ultrasonic sensor: Measures distance using sound waves and can be used for proximity sensing.
* DHT11 temperature and humidity sensor: Measures ambient temperature and humidity levels.
* LCD1602 display module: A 16x2 character LCD display for showing text and simple graphics.
* PIR sensor: Detects motion and can be used for security or automation purposes.
* Servo: A motor that can be controlled precisely for various robotic and mechanical applications.
* Long breadboard: Provides a platform for prototyping circuits and connecting multiple components.
* 40-pin jump wires: Allows for easy connection between components on a breadboard or circuit.
* WS2812 RGB light ring: A ring of individually addressable RGB LEDs for colorful lighting effects.
* Experiment platform plate: A platform for securely mounting and organizing components during experiments.
* 500W camera: A high-resolution camera capable of capturing detailed images and videos.
* R/G/B LED lights: Three separate LED lights in red, green, and blue colors for various lighting projects.
* MAX7219 8x8 matrix: An LED matrix display that can show text and simple patterns.
* S8050 and 2N9304 NPN transistors: Electronic switches used for amplification or switching applications.
* 220 ohm / 1k ohm resistor: Resistors for controlling current flow and voltage levels in circuits.

## Package Lists

* 1 x MPU6050 3 axis accelerate & gyroscope module
* 1 x 1-channel relay module
* 1 x Tilt sensor
* 1 x sound sensor
* 1 x T-type GPIO expansion board
* 1 x Edge GPIO expansion board
* 1 x Ultrasonic sensor
* 1 x DHT11 Temperature & Humidity Sensor
* 1 x LCD1602 Display module
* 1 x PIR sensor
* 2 x 5g Servo
* 20 x Male-to-Female Jumper wire
* 1 x 40Pin Parallel Wire
* 1 x Long breadboard
* 1 x 52Pi experiment platform plate
* 1 x 500W Pixels FPC cable Camera for Raspberry Pi
* 5 x Red LED
* 5 x Green LED
* 5 x Blue LED
* 5 x Yellow LED
* 1 x RGB Light Ring
* 1 x MAX7219 8x8 Dot light Matrix
* 20 x 220Ω resistor
* 20 x 1KΩ resistor
* 6 x Push button
* 6 x Push button hat
* 1 x Jumper wire box
* 5 x S8050 NPN transistor
* 5 x 2N9304 NPN transistor
* 20 x Female-to-Female jumper wire

**Getting Start**

**Install and Setup Raspberry Pi OS**

In this chapter, we firstly learn to start up Raspberry Pi. The content includes installing the OS, Raspberry Pi network and how to open terminal.

You can also check the complete tutorial on the official website of Raspberry Pi:

<https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up>

NOTE:

If your Raspberry Pi is set up, you can skip the part and go into the next chapter.

**Preparation**

**Requirements**:

* **Raspberry Pi**

Raspberry Pi is a series of small, affordable single-board computers developed by the Raspberry Pi Foundation. These credit card-sized computers are designed to promote the teaching of basic computer science in schools and to encourage experimentation and innovation in the field of electronics and programming.

The Raspberry Pi boards are powered by ARM processors and feature various input/output ports, including USB ports, HDMI ports, audio jacks, and GPIO (General Purpose Input/Output) pins. They also have an Ethernet port or Wi-Fi capabilities for network connectivity. The Raspberry Pi Foundation releases different models with varying specifications and capabilities.

Raspberry Pi computers run on various operating systems, with the most popular one being the Raspberry Pi OS (formerly known as Raspbian), which is a Linux-based operating system optimized for the Raspberry Pi. However, other operating systems like Ubuntu and various specialized distributions can also be installed.

Raspberry Pi boards are highly versatile and can be used for a wide range of projects. They can serve as a basic desktop computer, media center, game console, home automation hub, weather station, robotics controller, and much more. Their affordability, small size, and low power consumption make them popular among hobbyists, educators, and professionals alike.

* **Power supply**

A power supply, also known as a power adapter or power brick, is a device that provides electrical power to another device or system. It converts the electrical energy from a power source, such as an electrical outlet or a battery, into a form that is suitable for the operation of the device it is connected to.

Power supplies typically have two main components: a transformer and a rectifier. The transformer is responsible for stepping down or stepping up the voltage from the input power source to the desired output voltage. The rectifier converts the alternating current (AC) from the transformer into direct current (DC), which is the type of electrical current needed by most electronic devices.

Power supplies are available in various forms, depending on the application and the device they are intended to power. Some common types include:

Wall Warts/Plug-in Adapters: These are small power supplies that plug directly into a wall outlet and provide power to devices like smartphones, routers, and other small electronics.

Internal Power Supplies: These are installed inside electronic devices, such as desktop computers, gaming consoles, or televisions, to provide power to their internal components.

External Power Supplies: These are separate units connected to devices via a cable. Laptops, monitors, and some audio equipment often use external power supplies.

Power supplies are rated for their output voltage (measured in volts) and current (measured in amperes or mill amperes). It is important to use a power supply that matches the requirements of the device it is powering. Using an incompatible power supply can lead to improper operation or damage to the device.

Additionally, power supplies may have additional features such as overload protection, short-circuit protection, and voltage regulation to ensure stable and safe power delivery to the connected device.

* **MicroSD Card**

A microSD card, also known as a TF (Trans Flash) card, is a type of small, removable flash memory storage device. It is commonly used in portable electronic devices such as smartphones, tablets, digital cameras, and various other devices that require additional storage capacity.

MicroSD cards are designed to be compact, measuring approximately 15 mm x 11 mm in size. They are a smaller variant of the SD (Secure Digital) card, which is larger in dimensions. Despite their small size, microSD cards can provide significant storage capacity, ranging from a few gigabytes (GB) to several terabytes (TB) depending on the card's specification.

MicroSD cards use non-volatile memory technology, typically NAND flash memory, to store data. This type of memory retains information even when the power supply is removed. The cards also come in different speed classes, indicating their data transfer capabilities. The speed class is denoted by a number inside a capital letter U (e.g., U1, U3), which represents the minimum sustained write speed in megabytes per second (MB/s).

MicroSD cards are commonly used for storing various types of data, including photos, videos, music, documents, and applications. They can be easily inserted into compatible devices that have a microSD card slot or via an adapter that allows them to be used in devices with standard SD card slots.

It's important to note that some devices have maximum supported capacities for microSD cards, so it's essential to check the device's specifications to ensure compatibility. Additionally, microSD cards may have different levels of durability, such as waterproof or shockproof designs, which can be beneficial in specific use cases or environments.

* MicroHDMI to Full-size HDMI cable

An HDMI cable is included for connecting the Raspberry Pi to a monitor or TV for display.

**Optional：**

* **Screen or Display**

HDMI Displays: The Raspberry Pi 4B has two micro HDMI ports that support HDMI displays. You can connect the Pi to a standard HDMI monitor, TV, or projector using an HDMI cable. This provides a straightforward and commonly used method for visual output.

* **Mouse & Keyboard**

A mouse and keyboard are two common input devices used with computers to provide user interaction and input.

A mouse is a handheld device that typically has one or more buttons and a scrolling wheel. It is moved across a flat surface, such as a desk, and the movement is translated into cursor movement on the computer screen. By clicking the buttons, users can select, activate, or interact with various elements and functions on the computer. Mice come in different types, such as wired or wireless, and can offer additional features like programmable buttons or customizable sensitivity.

A keyboard is a typewriter-style input device with a set of keys, including letters, numbers, punctuation marks, and various control keys. It is used to input text, commands, and other data into a computer or device. Keyboards come in different layouts, with the most common being the QWERTY layout, named after the first six letters on the keyboard's top row. Keyboards can be connected to computers using wired connections (USB or PS/2) or wirelessly via Bluetooth.

Together, a mouse and keyboard provide a means for users to navigate graphical user interfaces, interact with software applications, browse the internet, play games, and perform various tasks on a computer. They are essential components for inputting commands and controlling the actions and behavior of a computer system.

* **Case**

A protective case is provided to house the Raspberry Pi board and protect it from damage.

* **Speaker with amplifier**

Adding a speaker with an amplifier to your Raspberry Pi 4B setup can enhance the audio capabilities and provide better sound quality for your projects. Here are some reasons why you might consider using a speaker with an amplifier:

* **Improved Sound Output:** The Raspberry Pi 4B has a built-in 3.5mm audio output, but its audio quality may not be sufficient for certain applications or projects that require better sound fidelity. By connecting a speaker with an amplifier, you can amplify the audio signal and achieve louder and clearer sound output.
* **Audio Projects and Multimedia Applications:** If you're working on projects that involve audio playback, music streaming, multimedia applications, or voice-based interactions, a speaker with an amplifier can significantly enhance the audio experience. It allows you to play music, sound effects, or speech with better volume and quality.
* **External Speaker Support:** The Raspberry Pi 4B does not have built-in speakers. Therefore, if you want to have audio output without relying on headphones or external audio devices, connecting a speaker with an amplifier directly to the Pi allows you to have dedicated audio output.
* **Customization and Expansion:** Adding a speaker with an amplifier gives you the flexibility to choose the type of speaker and amplifier that suits your specific requirements. You can select a speaker with the desired audio characteristics, such as frequency response and wattage, and pair it with an amplifier that provides the necessary power and amplification capabilities.
* **Project-Specific Audio Needs:** Depending on your project, you may require amplified sound for a larger space, such as in home automation systems, media centers, or public installations. An amplifier helps to drive the speaker and deliver sound effectively in such scenarios.

When choosing a speaker and amplifier for your Raspberry Pi 4B, ensure compatibility with the audio output options available on the Pi, such as the 3.5mm audio jack or HDMI audio. Additionally, consider the power requirements and any specific setup instructions or drivers needed to integrate the audio components with the Raspberry Pi.

## Project 1

In this guide, we'll discover how to interface a ws2812 RGB to a Raspberry Pi 4B.

Bright, colorful lights are the best, and this tutorial shows you how to set up Fully Configurable WS2812B led light ring s to run on a Pi 4 computer as quickly and flexibly as possible. In that manner, you can have the ambiance of your home reflect your tastes.

In most cases, when people talk about a "WS2812B Light ring," they mean a long piece of extensible PCB with a bunch of different RGB LED Nodes spread out and dotting all along the top. As a bonus, WS2812B light ring s can be addressed individually. Each RGB node can independently set its own color and brightness. The functionality of non-addressable RGB light ring s will be identical to that of WS2812B light ring s. WS2812B Light ring s are superior in every way, allowing for more imaginative LED light shows. It is possible to create more complex animations, light ring, and chases by individually programming the actions of each LED Node.

The "WS" in WS2812B indicates that this is the second major revision of this design; "World Semi" refers to the chip's original equipment manufacturer, "2812" to its specific part number, and "B" to its revision number. Each WS2812B LED Node contains an integrated circuit chip, as shown below.

Fully Configurable Light ring LEDs now come in wide varieties. It's important to remember whether the light ring operates on 5 volts or 12-volt power. To power our 5V-powered Raspberry Pi Computer, we need to use a 5V power supply. Or we'll have to install extra gear to meet the 12v Dc power specifications. LED Node densities also vary across WS2812B LED light ring s. Since our light ring consists of WS2812B nodes soldered onto rigid PCBs, it may also be controlled by the Python script included in this article. ws2812B LED light ring s are the ideal LEDs to use because of the benefits above. What this manual entails are listed below.

The control mechanism implemented in this tutorial works with WS2812B LED nodes, whether they are rigidly coupled on a PCB or flexibly arranged over a long light ring. No prior experience with Raspberry Pi has required, thanks to our comprehensive online raspberry pi 4 for the introduction.

### WS2812B protocol

Let's go over the protocol for these addressable LEDs so you can see how they function. Each WS2812B unit's red, green, and blue LEDs can be independently controlled to one of 256 brightness levels. Each LED module needs 24 bits of data or three groups of eight brightness bits. Here's a quick rundown of the steps involved:

The first LED in the string receives this data stream from the microcontroller, which consists of eight green bits, eight red bits, and eight blue bits.

In the case of several LEDs, the data sequence for the second LED begins with green, red, and blue information immediately after the information for the first LED. The process repeats itself until each LED is turned on.

The first LED acts as a receiver for data for all subsequent LEDs in the chain and then sends that data to the second LED without applying the same sequence.

When the original "number one" LED unit runs out of binary LED sequences, it passes the baton to the next available unit.

### What You Need

What follows are detailed instructions for configuring a single Raspberry Pi board to manage many individually addressable LEDs.

* Raspberry Pi 4B
* RGB LED Ring
* A DC barrel jack to 2-pin terminal block adapter.
* 【Optional】Breadboard
* 【Optional】52Pi experiment platform
* Jumper wire box

More than 16 WS2812B light ring Nodes will necessitate the usage of an external power supply. Even with highly efficient LEDs, producing brilliant light takes a lot of energy. On average, each pixel will consume 20mA, and while projecting white light at full brightness, each pixel will consume 60mA. This information shows us that 30 Pixels can consume 320mA on average and 960mA at full brightness. As a side note, staring directly into a room with the lights set to full white causes blind spots in my vision. Because of this, I typically set the brightness of my WS2812B lights to 20%. Ensure your WS2812B light ring 's power source is adequate for the illumination level you intend to achieve.

In today's world, 3.3V data logic is compatible with most WS2812B LED nodes. The GPIO Pins on a Raspberry Pi have an operational voltage range of 0 to 3.3V; therefore, this works out perfectly. This surface-mounted WS2812B light ring Node has seen five minor changes since its inception. Since the earlier WS2812B light ring can only work on 5V Data Logic, a logical level shifter may be necessary if you're experiencing strange or intermittent issues.

### Assembling Hardware

We'll configure our Raspberry PI to look and act like a desktop PC because that's how most people use computers at home and because it's the most user-friendly setup for novice makers. To convert our Raspberry Pi into a desktop computer, we'll need to install a micro-SD card pre-flashed with the operating system. Then hook it up to a monitor via HDMI and a mouse/keyboard combo. Also, this is a good moment to install a small heatsink on the Raspberry Pi Board's central processing unit IC. Look at the image below to see the setup adjacent to a short WS2812B LED light ring with three wires protruding from it. The WS2812B has three wires that have not yet been connected to the Raspberry Pi. WS2812B LED Light ring PCBs are often marked with arrows to indicate the direction of data flow. It is a common problem in troubleshooting when this is ignored.

### How to Configure Hardware

for WS2812B RGB LED Ring with 16 Nodes We need to plug in the 5V WS2812B Light ring 's three wires to the Raspberry Pi before turning the power on.

* Link the Raspberry Pi's 5V Power Input Pin with the Red Power Cable.
* Join the white wire labeled "Ground" to the Pi's Ground pin.
* Join the Raspberry Pi's Green Input Pin to its GPIO 18 port.

It is important to remember that the Green Input Pin could be any GPIO pin as long as the appropriate adjustments are made in your Python programs.

In this manual, GPIO 18 will be used for the Data Line in all scripts.

Hardware Configuration for a WS2812B Light ring with 16 Nodes in Length. However, the Pi 4 Power Pin Out is not powerful enough to power a full LED light ring. It's important to know the power requirements of fully addressable WS2812B LEDs, especially when used in large quantities. 150 fully illuminated RGBW LEDs should be manageable by a high-quality 5v 4A power supply.

Following the diagram below, we will connect a 5V 4A power source via a DC barrel jack. Our WS2812B LEDs are pre-soldered with one green data wire, two red power wires, and white ground wires. The white wire should be attached to the DC barrel jack's negative terminal using a screwdriver. Join the Red Wire to the Terminal with the Positive Screw Down. The White Connectors all link to one other, indicating a Common Ground shared by the Raspberry Pi and the power source. The two devices can't be connected without ground, which will prevent voltage fluctuations and data transmission mistakes.

Furthermore, the WS2812B Light ring is no longer connected to the Pi 4 computer via a red power wire. We have found an alternative to using a Raspberry Pi to run our system. See the diagram below for details on connecting a Raspberry Pi to a string of WS2812B LEDs so that they may be controlled remotely.

There are several great best practices to keep in mind if you're planning on powering many LED light ring s.

Now that everything is hooked up how you like it, you can turn on the Pi 4 System by inserting a USB-C cable.

### Set up of software.

You'll need to install a few packages if you're starting with a clean install of Raspberry Pi OS. Since this is the case, WS2812B LEDs will function properly. When you plug the Raspberry Pi into an electrical outlet and finish the initial boot-up wizard, you will be taken to the operating system's familiar desktop.

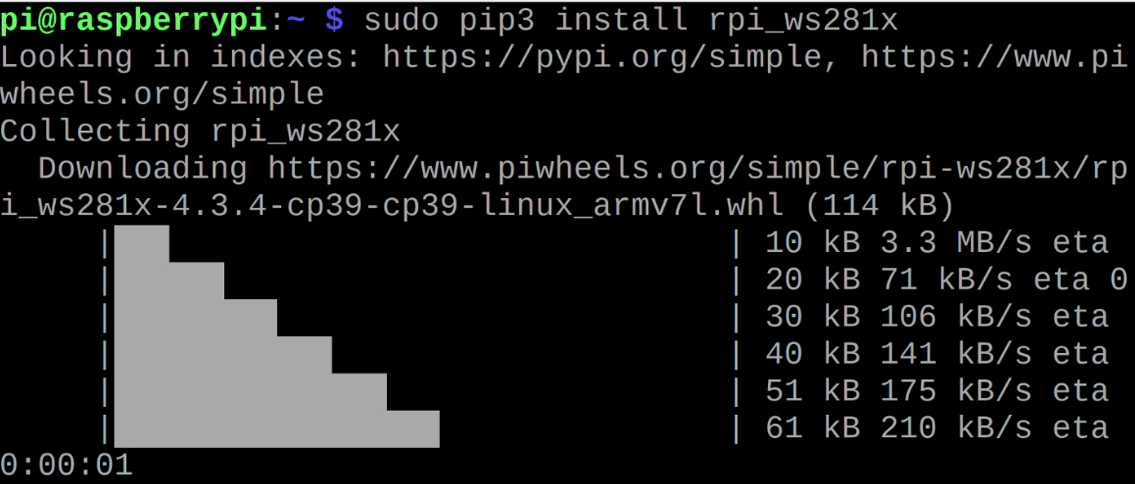
Click the black button in the upper left corner to open a new terminal window. An interactive terminal will launch. Below is a picture of this happening, with a huge red arrow indicating the pressed terminal icon.

With this terminal window, we can search for specific programs and download them from the internet. Here are the command lines you use in the terminal to install everything you need. Type | Y | to confirm installs if prompted.

**sudo pip3 install rpi\_ws281x**

**sudo pip3 install adafruit-circuitpython-neopixel**

**sudo python3 -m pip install --force-reinstall adafruit-blinka**



You now have the software and programming packages installed on your Device to power and run WS2812B properly.

### Basic Code for WS2812B LED

Now you can check if your Device is properly constructed and functioning with the help of the above strandtest.py script. Moreover, the resulting illumination is just breathtaking. Maker may be left wanting more, though. This is of special importance to the readability of the code, which facilitates its adaptation to specific tasks. This section will show you how to control the WS2812B light ring with a few basic Python programs easily modified to suit your needs.

To get started, look at One Light ring Neopixels.py, a very basic WS2812B Control Script.

**import time**

**import board**

**import neopixel**

**#Initialise, a light ring s variable, provide the GPIO Data Pin**

**# utilized and the amount of LED Nodes on the light ring and brightness (0 to 1 value)**

**pixels1 = neopixel.NeoPixel(board.D18, 55, brightness=1)**

**#Also, create an arbitrary count variable**

**x=0**

**pixels1.fill((0, 220, 0))**

**#LED Node 10 and the color Blue were selected**

**pixels1[10] = (0, 20, 255)**

**#Showing a different color**

**time.sleep(4)**

**#Below will loop until variable x has a value of 35**

**while x<35:**

**pixels1[x] = (255, 0, 0)**

**pixels1[x-5] = (255, 0, 100)**

**pixels1[x-10] = (0, 0, 255)**

**#Add 1 to the counter**

**x=x+1**

**#Add a small time pause which will translate to 'smoothly' changing color**

**time.sleep(0.05)**

**#Below section is the same process as the above loop, just in reverse**

**while x>-15:**

**pixels1[x] = (255, 0, 0)**

**pixels1[x+5] = (255, 0, 100)**

**pixels1[x+10] = (0, 255, 0)**

**x=x-1**

**time.sleep(0.05)**

**#Add a brief time delay to appreciate what has happened**

**time.sleep(4)**

**#Complete the script by returning all the LEDs to the off**

**pixels1.fill((0, 0, 0))**

The Python script has been thoroughly commented on, so you should have no trouble following along. The script in the previous paragraph includes readily apparent dials and switches. You'll learn the fundamentals of making a captivating LED Pattern, including how to adjust the number of LED Nodes, the brightness of the LED light ring s, whether to illuminate the entire board or just a section of it and how to illuminate individual LEDs. Each Node color can be customized by entering a corresponding Red, Blue, or Green numeric value. A number can have any value between zero and 255. The term "RGB Color Code" is commonly used to describe this. While experimenting with different hues can be a lot of fun, the best method to find the precise shade you need is to use a website that lists RGB color codes. Choose any color on this site, and you'll get its corresponding RGB color code in three numbers. After you've located the three-digit color code, you can easily enter it into the Python program.

You can copy and paste the script into Thonny IDE and run it. Once you've saved the file, you can press the run button to activate the script. Below are some screenshots displaying the output.

### Control Several LED Light ring s at Once

Next is to hook up a few different LED Light ring s to the Pi 4 SBC and run a Python script to manage everything. Hardware-wise, I've replicated the procedure described in Hardware Configuration for Small, fewer than 30 Nodes long WS2812B, adding a second, similarly-sized WS2812B LED Light ring.

I wired an additional WS2812B light ring of lights up to the GPIO in the manner described below. The Raspberry Pi's other 5V Pin was connected to the red power line. The Raspberry Pi's White Ground Wire was attached to the board's second Ground Port. In this case, GPIO 21 received the green data wire. Check out the diagram of the wired components below; using two WS2812B LED light ring s of varying density is fine.

Using Thonny IDE as previously, run the python script titled Two Light ring **Neopixels.py** to control these two WS2812B Light ring s in a no-frills fashion. Look at this thoroughly documented Python script down here.

#include all necessary packages to get LEDs to work with Raspberry Pi

**import time**

**import board**

**import neopixel**

**#Initialise two light ring s variables, provide the GPIO Data Pin**

**# utilized and the amount of LED Nodes and brightness (0 to 1 value)**

**pixels1 = neopixel.NeoPixel(board.D18, 30, brightness=1)**

**pixels2 = neopixel.NeoPixel(board.D21, 6, brightness=1)**

**#Focusing on a particular light ring, use the command Fill to make it all a single color**

**#based on decimal code R, G, B. Number can be anything from 255 - 0. Use an RGB Colour**

**#Code Chart Website to quickly identify a desired fill color.**

**pixels1.fill((0, 255, 0))**

**pixels2.fill((0, 0, 255))**

**#Sleep for one second, and then code repeats for different color combinations. Light changes**

**#Could happen instead in response to certain buttons being pressed or due to threshold values**

**time.sleep(1.5)**

**pixels1.fill((200, 200, 0))**

**pixels2.fill((0, 200, 200))**

**time.sleep(1.5)**

**pixels1.fill((50, 70, 215))**

**pixels2.fill((215, 50, 70))**

**time.sleep(1.5)**

**pixels1.fill((0, 0, 0))**

**pixels2.fill((0, 0, 0))**

Note that a Pi 4Computer has four Data Wire locations that can control individual WS2812B LED Light ring s, provided the corresponding Python script is updated correctly. The pinouts are numbered as follows: GPIO18, GPIO21, GPIO12, and GPIO10. With some tweaking and development of this technique, you can use a single Raspberry Pi as a standalone controller for four individually addressable WS2812B light ring

Ensure the right number of LED Nodes has been assigned to both LED WS2812B Light ring s before running the script. Once the script is ready to be executed, click the Thonny IDE's large green Run button. Both strands will suddenly come to life, each filled with a unique color. In the picture below, you can see this in action.

## Project 2

MPU6050 (Accelerometer Gyroscope) Interfacing with Raspberry Pi

### Overview of MPU6050 Sensor

The MPU6050 sensor module is an integrated 6-axis Motion tracking device.

It has a 3-axis Gyroscope, 3-axis Accelerometer, Digital Motion Processor, and a Temperature sensor, all in a single IC.

It can accept inputs from other sensors like a 3-axis magnetometer or pressure sensor using its Auxiliary I2C bus.

If an external 3-axis magnetometer is connected, it can provide complete 9-axis Motion Fusion output.

A microcontroller can communicate with this module using the I2C communication protocol. Various parameters can be found by reading values from addresses of certain registers using I2C communication.

Gyroscope and accelerometer reading along X, Y, and Z axes are available in 2’s complement form.

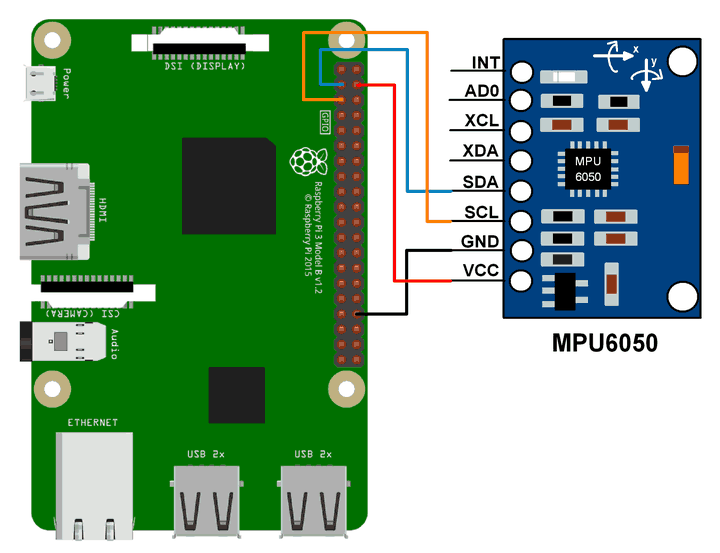
Gyroscope readings are in degrees per second (dps) unit; Accelerometer readings are in g unit.

For more information about MPU6050 Sensor Module and how to use it, refer the topic MPU6050 Sensor Module in the sensors and modules section.

To interface MPU6050 using Raspberry Pi, we should ensure that the I2C protocol on Raspberry Pi is turned on. So before going for interfacing MPU6050 with Raspberry Pi, we need to make some I2C configurations on Raspberry Pi which you can refer Raspberry Pi I2C.

After configuring I2C on Raspberry Pi, let’s interface Raspberry Pi with MPU6050.

### Connection Diagram of MPU6050 with Raspberry Pi



### MPU6050 Example using Raspberry Pi

Here, we will interface the MPU6050 module with Raspberry Pi to read Gyroscope and Accelerometer values and print them.

We can interface the MPU6050 module with Raspberry Pi using Python and C language. We will display the value of the Accelerometer and Gyroscope on the terminal which are read from the MPU6050 module.

For frequently used Python based I2C function on Raspberry Pi, you can refer Python based I2C functions for Raspberry Pi.

### Python based I2C functions for Raspberry Pi

Let’s see basic Python based I2C functions which are frequently used for I2C communication on Raspberry Pi.

While developing program for Raspberry Pi I2C communication in python, we can use SMBus library package which has great support to access I2C devices. So, we should add SMBus support for Python by using apt packet manager,

**sudo apt-get install python-smbus**

### Python based I2C Functions

**Import SMBus**

* To access I2C bus on Raspberry Pi using SMBus Python module, import SMBus module as follows.

**import smbus**

* create object of SMBus class to access I2C based Python function.

<Object name> = smbus.SMBus(I2C port no.)

I2C port no : I2C port no. i.e. 0 or 1

* Example - Bus = smbus.SMBus(1)

Now, we can access SMBus class with Bus object.

**Bus.write\_byte\_data(Device Address, Register Address, Value)**

* This function is used to write data to the required register.

**Device Address :** 7-bit or 10-bit device address

**Register Address :** Register address to which we need to write

**Value :** pass value which needs to write into the register

* Example - Bus.write\_byte\_data(0x68, 0x01, 0x07)

**Bus.write\_i2c\_block\_data(Device Address, Register Address, [value1, value2,….])**

* This function is used to write a block of 32 bytes.

**Device Address :** 7-bit or 10-bit device address

**Register Address** : Register address to which we need to write data

**Value1 Value2…**. : write a block of bytes to the required address

* Example - Bus.write\_i2c\_block\_data(0x68, 0x00, [0, 1, 2, 3, 4, 5]) # write 6 bytes of data from 0 address.

**Bus.read\_byte\_data(Device Address, Register Address)**

* This function is used to read data byte from required register.

**Device Address :** 7-bit or 10-bit device address

**Register Address :** Register address from which we need to read data

* Example - Bus.read\_byte\_data (0x68, 0x01)

**Bus.read\_i2c\_block\_data(Device Address, Register Address, block of bytes)**

* This function is used to read a block of 32 bytes.

**Device Address** – 7-bit or 10-bit device address

**Register Address** – Register address from which we need to read data

**Block of Bytes**  – readno of bytes from the required address

* Example - Bus.read\_i2c\_block\_data(0x68, 0x00, 8) # returnvalue is a list of 6 bytes

### MPU6050 Code for Raspberry Pi using Python

**import smbus #import SMBus module of I2C**

**from time import sleep #import**

**#some MPU6050 Registers and their Address**

**PWR\_MGMT\_1 = 0x6B**

**SMPLRT\_DIV = 0x19**

**CONFIG = 0x1A**

**GYRO\_CONFIG = 0x1B**

**INT\_ENABLE = 0x38**

**ACCEL\_XOUT\_H = 0x3B**

**ACCEL\_YOUT\_H = 0x3D**

**ACCEL\_ZOUT\_H = 0x3F**

**GYRO\_XOUT\_H = 0x43**

**GYRO\_YOUT\_H = 0x45**

**GYRO\_ZOUT\_H = 0x47**

**def MPU\_Init():**

**#write to sample rate register**

**bus.write\_byte\_data(Device\_Address, SMPLRT\_DIV, 7)**

**#Write to power management register**

**bus.write\_byte\_data(Device\_Address, PWR\_MGMT\_1, 1)**

**#Write to Configuration register**

**bus.write\_byte\_data(Device\_Address, CONFIG, 0)**

**#Write to Gyro configuration register**

**bus.write\_byte\_data(Device\_Address, GYRO\_CONFIG, 24)**

**#Write to interrupt enable register**

**bus.write\_byte\_data(Device\_Address, INT\_ENABLE, 1)**

**def read\_raw\_data(addr):**

**#Accelero and Gyro value are 16-bit**

**high = bus.read\_byte\_data(Device\_Address, addr)**

**low = bus.read\_byte\_data(Device\_Address, addr+1)**

**#concatenate higher and lower value**

**value = ((high << 8) | low)**

**#to get signed value from mpu6050**

**if(value > 32768):**

**value = value - 65536**

**return value**

**bus = smbus.SMBus(1) # or bus = smbus.SMBus(0) for older version boards**

**Device\_Address = 0x68 # MPU6050 device address**

**MPU\_Init()**

**print (" Reading Data of Gyroscope and Accelerometer")**

**while True:**

**#Read Accelerometer raw value**

**acc\_x = read\_raw\_data(ACCEL\_XOUT\_H)**

**acc\_y = read\_raw\_data(ACCEL\_YOUT\_H)**

**acc\_z = read\_raw\_data(ACCEL\_ZOUT\_H)**

**#Read Gyroscope raw value**

**gyro\_x = read\_raw\_data(GYRO\_XOUT\_H)**

**gyro\_y = read\_raw\_data(GYRO\_YOUT\_H)**

**gyro\_z = read\_raw\_data(GYRO\_ZOUT\_H)**

**#Full scale range +/- 250 degree/C as per sensitivity scale factor**

**Ax = acc\_x/16384.0**

**Ay = acc\_y/16384.0**

**Az = acc\_z/16384.0**

**Gx = gyro\_x/131.0**

**Gy = gyro\_y/131.0**

**Gz = gyro\_z/131.0**

**print ("Gx=%.2f" %Gx, u'\u00b0'+ "/s", "\tGy=%.2f" %Gy, u'\u00b0'+ "/s", "\tGz=%.2f" %Gz, u'\u00b0'+ "/s", "\tAx=%.2f g" %Ax, "\tAy=%.2f g" %Ay, "\tAz=%.2f g" %Az)**

**sleep(1)**

### MPU6050 Code for Raspberry Pi using C (WiringPi Library)

**Here, we are using the WiringPi C library to read data from the MPU6050 module.**

**#include <wiringPiI2C.h>**

**#include <stdlib.h>**

**#include <stdio.h>**

**#include <wiringPi.h>**

**#define Device\_Address 0x68 /\*Device Address/Identifier for MPU6050\*/**

**#define PWR\_MGMT\_1 0x6B**

**#define SMPLRT\_DIV 0x19**

**#define CONFIG 0x1A**

**#define GYRO\_CONFIG 0x1B**

**#define INT\_ENABLE 0x38**

**#define ACCEL\_XOUT\_H 0x3B**

**#define ACCEL\_YOUT\_H 0x3D**

**#define ACCEL\_ZOUT\_H 0x3F**

**#define GYRO\_XOUT\_H 0x43**

**#define GYRO\_YOUT\_H 0x45**

**#define GYRO\_ZOUT\_H 0x47**

**int fd;**

**void MPU6050\_Init(){**

**wiringPiI2CWriteReg8 (fd, SMPLRT\_DIV, 0x07); /\* Write to sample rate register \*/**

**wiringPiI2CWriteReg8 (fd, PWR\_MGMT\_1, 0x01); /\* Write to power management register \*/**

**wiringPiI2CWriteReg8 (fd, CONFIG, 0); /\* Write to Configuration register \*/**

**wiringPiI2CWriteReg8 (fd, GYRO\_CONFIG, 24); /\* Write to Gyro Configuration register \*/**

**wiringPiI2CWriteReg8 (fd, INT\_ENABLE, 0x01); /\*Write to interrupt enable register \*/**

**}**

**short read\_raw\_data(int addr){**

**short high\_byte,low\_byte,value;**

**high\_byte = wiringPiI2CReadReg8(fd, addr);**

**low\_byte = wiringPiI2CReadReg8(fd, addr+1);**

**value = (high\_byte << 8) | low\_byte;**

**return value;**

**}**

**void ms\_delay(int val){**

**int i,j;**

**for(i=0;i<=val;i++)**

**for(j=0;j<1200;j++);**

**}**

**int main(){**

**float Acc\_x,Acc\_y,Acc\_z;**

**float Gyro\_x,Gyro\_y,Gyro\_z;**

**float Ax=0, Ay=0, Az=0;**

**float Gx=0, Gy=0, Gz=0;**

**fd = wiringPiI2CSetup(Device\_Address); /\*Initializes I2C with device Address\*/**

**MPU6050\_Init(); /\* Initializes MPU6050 \*/**

**while(1)**

**{**

**/\*Read raw value of Accelerometer and gyroscope from MPU6050\*/**

**Acc\_x = read\_raw\_data(ACCEL\_XOUT\_H);**

**Acc\_y = read\_raw\_data(ACCEL\_YOUT\_H);**

**Acc\_z = read\_raw\_data(ACCEL\_ZOUT\_H);**

**Gyro\_x = read\_raw\_data(GYRO\_XOUT\_H);**

**Gyro\_y = read\_raw\_data(GYRO\_YOUT\_H);**

**Gyro\_z = read\_raw\_data(GYRO\_ZOUT\_H);**

**/\* Divide raw value by sensitivity scale factor \*/**

**Ax = Acc\_x/16384.0;**

**Ay = Acc\_y/16384.0;**

**Az = Acc\_z/16384.0;**

**Gx = Gyro\_x/131;**

**Gy = Gyro\_y/131;**

**Gz = Gyro\_z/131;**

**printf("\n Gx=%.3f °/s\tGy=%.3f °/s\tGz=%.3f °/s\tAx=%.3f g\tAy=%.3f g\tAz=%.3f g\n",Gx,Gy,Gz,Ax,Ay,Az);**

**delay(500);**

**}**

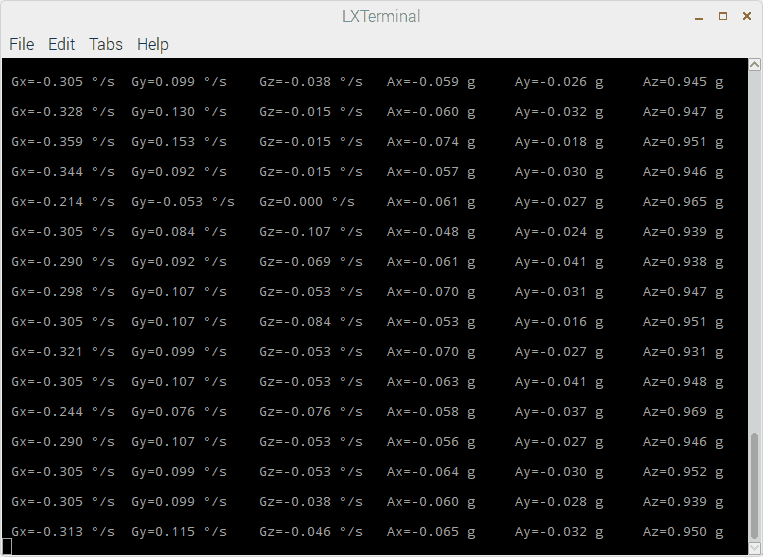
**return 0;**

**}**

### MPU6050 Output

The output window will show all values mentioned below

* Gx  = Gyro X-axis data in degree/seconds
* Gy  = Gyro Y-axis data in degree/seconds
* Gz  = Gyro Z-axis data in degree/seconds
* Ax  = Accelerometer X-axis data in g
* Ay  = Accelerometer Y-axis data in g
* Az  = Accelerometer Z-axis data in g



## Project 3

### Interface a 1-Channel Relay with Raspberry Pi 4B

This tutorial will show you how to connect a 4-channel relay module with a Raspberry Pi to carry out switching. A relay's primary function is to allow a circuit to be controlled by a weak signal. High-current applications necessitate relays to separate the low-voltage control circuit from the high-power driving circuits. Because of this, understanding it is crucial for those interested in industrial or household automation.

If you've been tinkering with a raspberry pi for a while, consider the various ways in which you might put it to use.

So let’s dive in!

### Components

* channel 5V Relay Module
* Raspberry pi 4B
* A Few Jumper cables
* 5v power source

The raspberry pi has real-world uses, such as remotely turning a device on or off over the internet, sensors, or a mobile app communicating with the pi via Bluetooth. If we can master this, we will open up a world of possibilities.

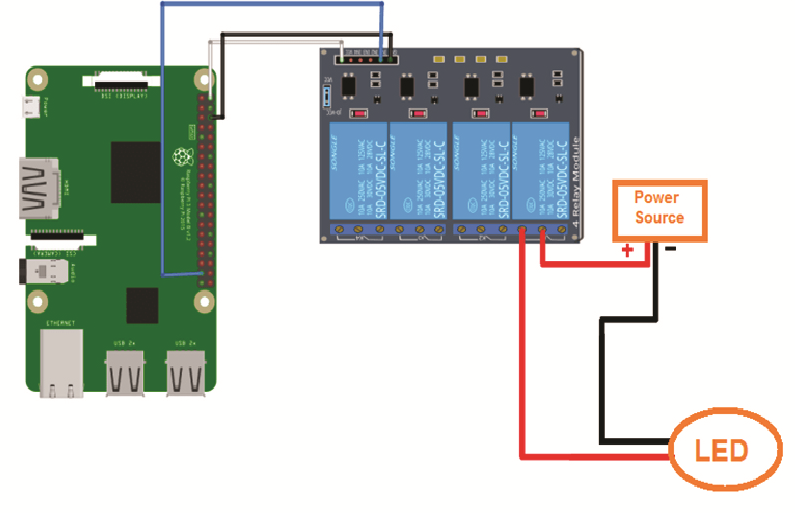
### Raspberry Pi as a Device Switch

Anybody who has experimented with a Raspberry Pi knows it has a GPIO.

The 40-pin general-purpose input/output (GPIO) connector is great for connecting various output devices. Since it is a digital computer, Raspberry Pi's GPIO pins can provide logic outputs. Logic 0 and 1 are the terms used to describe these two possible results from a logic circuit. If you write a program to make a Raspberry Pi pin write logic zero, you'll get a real-world GND potential. Likewise, when logic 1 is written on the Raspberry Pi pin, +3.3V is produced.

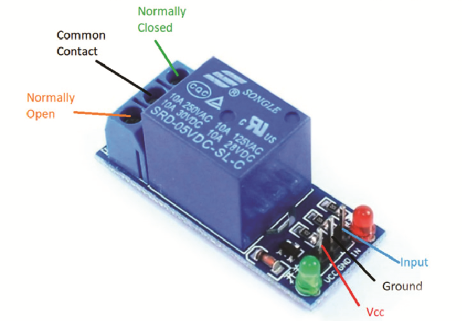
Logic 0 (gnd) and logic 1 (+3.3v) may be generated on any Raspberry Pi output pin with some programming. However, the output voltage is too low to power any real-world equipment. A maximum of 20 mA can be drawn from the output pin, as the 3.3V generated is currently limited. This only means that we can connect an LED straight to a Raspberry Pi gpio pin. In addition to the DC motor, no other output device can be connected directly to the raspberry pi's pin. Because of this, a different method is required when connecting an AC device.

How do relays work?



### How do relays work?

* A control coil encloses a core made of iron.
* When power is supplied to the control coil, an electromagnet begins to energize, boosting the strength of the magnetic field it produces.
* As a result, the top contact arm attracts the lower fixed arm, closing the contacts and completing the circuit.



This image is a cross-sectional diagram of a relay's inner workings. A control coil encloses a core made of iron. When power is supplied to the control coil, an electromagnet begins to energize, boosting the strength of the magnetic field it produces. As a result, the top contact arm attracts the lower fixed arm, closing the contacts and completing the circuit. However, if the relay were de-energized before the connections were closed, the contacts would travel the opposite way, creating an open circuit. When the coil current is shut off, the spring will return the movable armature to its original position.

The operation of a relay is identical to that of a switch. This also means that the same principle applies. When a relay is used, one or even more poles are flipped. Each pole has two primary contact directions. They have NO contact, commonly known as the Normal Open Contact configuration. Connecting with someone is another name for this action. On activation, the relay completes the circuit. Once the relay is deactivated, the circuit is broken.

NC contact is short for normally closed contact. This is synonymous with the term "break contact." In contrast to "NO contact," communication will occur. By switching on the relay, the circuit is broken. As soon as the relay is turned off, the circuit is complete.

A relay's COM contact is shared by the normally closed (NC) and normally open (NO) contacts.

An example of a relay module with two channels is displayed here. As its name implies, a two-channel relay module consists of a circuit with two independent relays built in. This enables the simultaneous manipulation of two distinct gadgets. It follows that the greater the available channels, the more gadgets we can link together.

Let's connect the Pi 4 to the 2-channel relay now. With its four corresponding pins, the 2-channel relay may communicate with a Raspberry Pi. VCC, GND, IN1, and IN2 are the inputs. Current input higher than 30 mA and an input voltage of 5 V is required to power the relay module. As a result of this glaring current shortfall, we must rely on an external power source. Here, we'll employ the widely used MB102 Breadboard Power Supply, an external power supply board. If you're curious about this power source and why we decided to use it, there are some helpful links below. A 3.3 V Relay is what you'll need to operate with Relays without an external power supply.

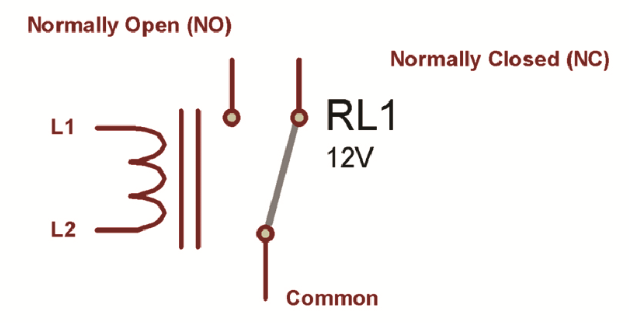
The relay module's VCC pin must be connected to the power supply's 5 V pin. Then Join the Raspberry Pi 4's ground pin (GND) to the power supply's ground pin (GND). The next step is to attach a jumper wire between the power supply's second GND port and the Raspberry Pi's second GND port. As a result, the ground pin on the Pi 4, the relay switch, and the power source are now all connected. Last but not least, connect a push button to GPIO 2 to activate the relay. Let's connect a few high-powered gadgets across the relay's output now. As seen in the circuit schematic, a CFL is connected to the NC and COM ports.

### Using a Relay as a Switching Circuit

As was previously mentioned, the RPi is a computer with an output range of only +3.3v to 0v. We need a dedicated electronic switching circuit to link any real-world device to the Raspberry Pi and enable it to switch. Assume throughout this lesson that you want to control an electrical lamp using raspberry pi. A switching circuit is required because we cannot wire the light bulb directly to the raspberry pi. There must be a switching circuit, such as a relay, to turn on and off AC appliances.

The following graphic depicts the internal structure of a relay.

**NOTE: A switching circuit is required because we can’t wire the light bulb directly to the Raspberry Pi.**



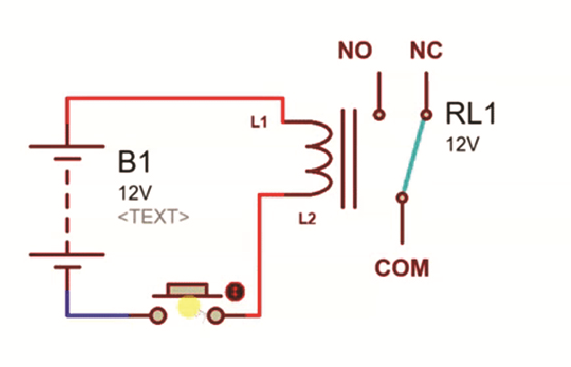
* Above, you can see that a basic relay has five connections.
* The electromagnetic coil’s two ends, L1 and L2, serve as the magnet within the relay.

Directly connecting the L1 Or L2 to a DC power supply is possible. The coil becomes an electromagnet when it is supplied with an electric current. Unlike the usually closed (NC) and ordinarily open (NO) terminals, the Common terminal can be moved.

### Working of Relays

The NC terminal serves as the home of the common terminal, which is held in place using sprint tension. This is the relay's initial setting. In a standard setup, the NC and COM terminals of a relay are linked when the device is positioned on a flat surface. The coil becomes magnetized whenever a voltage is placed between coil terminals L1 and L2. The spring tension is opposed by the magnetic force that pulls the common terminal off of NC and onto NO. As long as the relay is live, there will be continuity between NO and COM. Magnetization ceases when the coil voltage is removed, and the common terminal reverts to the NC Terminal, as depicted below.

* In a standard setup, the NC and COM terminals of a relay are linked when the device is positioned on a flat surface.



* Typically, relays require a voltage of 12V or 24V, which the Raspberry Pi cannot produce.

To conclude, we can switch any AC device on and off with a relay if we know how to do so effectively. However, the issue of activating the relay itself remains to be seen. Typically, relays require a voltage of 12v or 24v, which the Raspberry Pi cannot produce. A microprocessor cannot supply the 30-50mA current required for relays that operate on +5v coil voltage. Consequently, a relay switching circuit is required rather than a direct connection to the raspberry pi.

### An explanation of relay control

By adjusting the voltage at the GPIO pins, you may toggle the state of any relay module, whether it's attached directly to the ports or via a relay sensor. Using Python in conjunction with the GPIO library is the simplest solution.

After establishing a connection to the desired GPIO pin, changing its state is as simple as issuing a single command in Python. Pin 26 on the GPIO header are used for 1-channel relay module. For this reason, for instance, the first relay's power supply can be switched after a delay of just one second.

### Demo code

**import time**

**import RPi.GPIO as GPIO**

**relay\_ch = 26**

**GPIO.setwarnings(False)**

**GPIO.setmode(GPIO.BCM)**

**GPIO.setup(relay\_ch, GPIO.OUT)**

**GPIO.output(relay\_ch, GPIO.LOW)**

**time.sleep(1)**

**GPIO.output(relay\_ch, GPIO.HIGH)**

**GPIO.cleanup()**

At this point, the raspberry pi 4B controls the 1-channel relay module. Closed-loop control for high-powered equipment is made possible by incorporating such code in considerably more complex scripts that consider human input or sensor readings.

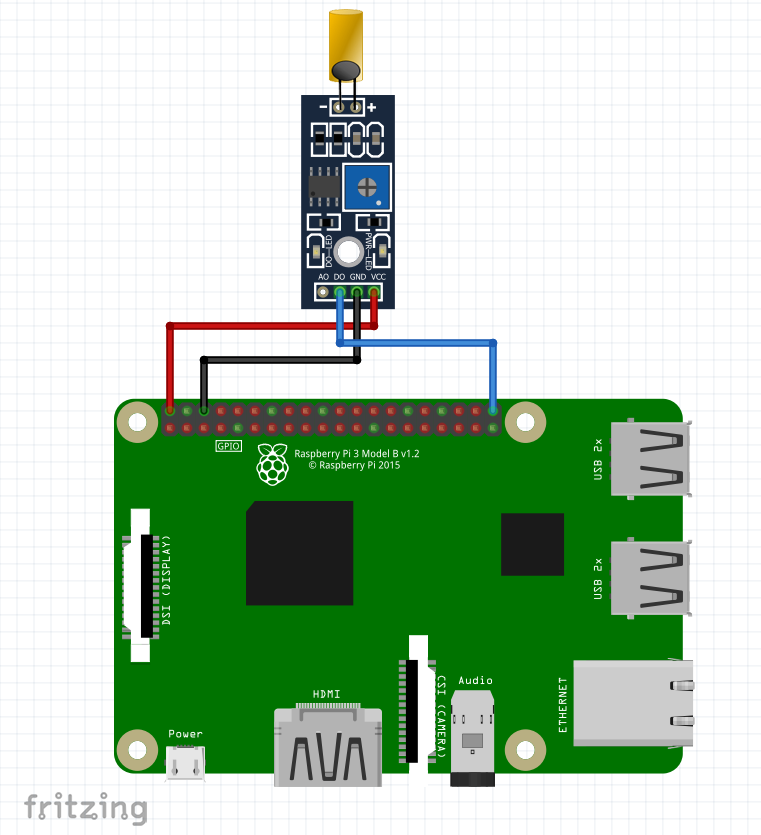
## Project 4

How about we create a "Tilt-controlled LED Party Hat" using a Raspberry Pi 4B and a tilt sensor? The idea is to build a hat that lights up with different colors based on the tilt angle of the wearer's head. It can be a fun accessory for parties or events. Here's the principle and the demo code in Python:

### Principle:

* Connect the tilt sensor to the Raspberry Pi GPIO pins.
* Read the tilt sensor's output to detect the tilt angle.
* Based on the tilt angle, control an RGB LED to display different colors.
* Mount the RGB LED on the hat, so it illuminates the hat with the chosen color.
* Tilt sensor

### How to wiring it



### Demo Code:

**import RPi.GPIO as GPIO**

**import time**

**# GPIO pins for tilt sensor and RGB LED**

**tilt\_pin = 17**

**red\_pin = 18**

**green\_pin = 19**

**blue\_pin = 20**

**# Set up GPIO mode and pins**

**GPIO.setmode(GPIO.BCM)**

**GPIO.setup(tilt\_pin, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)**

**GPIO.setup(red\_pin, GPIO.OUT)**

**GPIO.setup(green\_pin, GPIO.OUT)**

**GPIO.setup(blue\_pin, GPIO.OUT)**

**# Function to control the RGB LED based on tilt angle**

**def control\_led(tilt\_angle):**

**if tilt\_angle < -45: # Strong tilt to the left**

**GPIO.output(red\_pin, GPIO.HIGH)**

**GPIO.output(green\_pin, GPIO.LOW)**

**GPIO.output(blue\_pin, GPIO.LOW)**

**elif tilt\_angle > 45: # Strong tilt to the right**

**GPIO.output(red\_pin, GPIO.LOW)**

**GPIO.output(green\_pin, GPIO.HIGH)**

**GPIO.output(blue\_pin, GPIO.LOW)**

**else: # Neutral position**

**GPIO.output(red\_pin, GPIO.LOW)**

**GPIO.output(green\_pin, GPIO.LOW)**

**GPIO.output(blue\_pin, GPIO.HIGH)**

**# Main loop**

**try:**

**while True:**

**tilt\_state = GPIO.input(tilt\_pin)**

**if tilt\_state == GPIO.LOW:**

**print("Tilt detected!")**

**# Read the tilt angle using appropriate method or sensor library**

**tilt\_angle = # Read the tilt angle here**

**control\_led(tilt\_angle)**

**time.sleep(0.1)**

**except KeyboardInterrupt:**

**GPIO.cleanup()**

Please note that the code assumes you have connected the tilt sensor and RGB LED to the specified GPIO pins on the Raspberry Pi. Also, you'll need to replace the tilt angle = # Read the tilt angle here line with the appropriate code to read the tilt angle from your specific tilt sensor.

## Project 5

Electret microphones are excellent at detecting sound. You can take advantage of electret microphones to build Raspberry Pi projects that can detect sound and do something when it detects sound.

In this tutorial, we will connect an electret microphone to the Raspberry Pi and turn on a 5V relay when a particular sound level is reached.

### How Electret Microphone Work

An electret microphone consists of a diaphragm, a pair of electrodes, and a transistor. The diaphragm responds to changes the air pressure generated by sound waves. This causes a change in capacitance between the diaphragm and a charged metal plate inside the microphone housing. The varying capacitance produces a modulating voltage, which is amplified by a JFET transistor inside the housing. This creates a small sinusoidal voltage signal at the microphone pins that depends on the air pressure generated by sound waves.

To detect the signal from an electret microphone on a microcontroller or single-board computer, we need to use a comparator to convert the analog signal into a digital signal.

### The Microphone Breakout

Fortunately, there is the microphone breakout. This breakout board has built-in comparators, potentiometers, and a preamplifier to make it easier to connect to the Raspberry Pi:

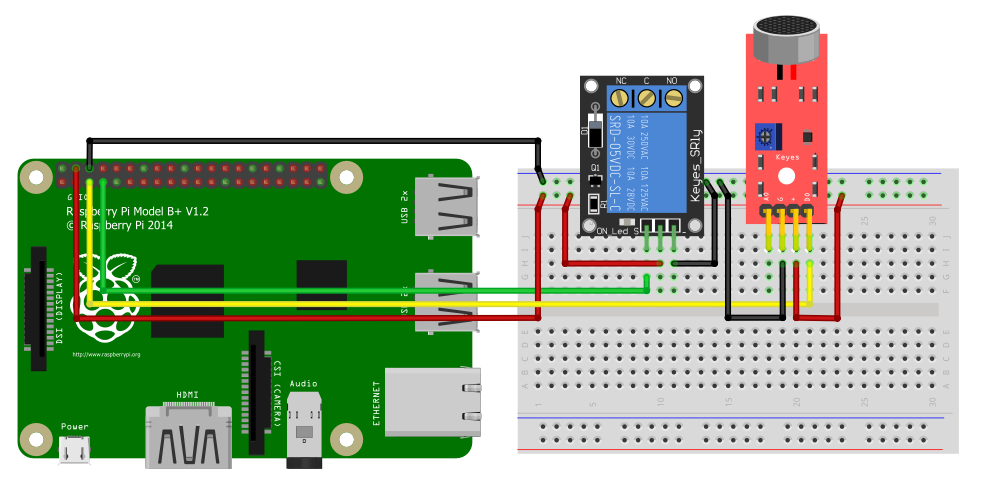
### Using The Microphone Breakout with Raspberry Pi

To demonstrate how to use the KY038 with a Raspberry Pi, we will build a circuit that triggers a 5V relay when a certain sound threshold is detected by the KY038 microphone.

These are the parts you will need:

* Raspberry Pi
* Breadboard
* Jumper wires
* 1-channel 5V relay
* Microphone

Connect the devices according to the wiring diagram below.



### Demo Code

Next, open up your favorite code editor on the Raspberry Pi and copy the code below.

**import RPi.GPIO as GPIO**

**MicPin = 3**

**RelayPin = 4**

**GPIO.setwarnings(False)**

**GPIO.setmode(GPIO.BCM)**

**GPIO.setup(MicPin, GPIO.IN, pull\_up\_down=GPIO.PUD\_DOWN)**

**GPIO.setup(RelayPin, GPIO.OUT, initial=GPIO.LOW)**

**while True:**

**GPIO.output(LedPin, GPIO.input(MicPin))**

The code starts by importing the RPi.GPIO library. We rename it GPIO for ease of use. We then declare the microphone and relay pin numbers and set them to input and output, respectively. We need to declare the pin with a pull-down resistor for the signal to stay LOW for input devices.

The main loop works such that whenever the input from the sound sensor returns HIGH, it will also write HIGH to the relay, triggering whatever device the relay is connected to.

## Project 6

### Raspberry Pi Music Player

**Description:** Create a music player using a Raspberry Pi 4B and buttons. The buttons will serve as controls for playing, pausing, skipping tracks, and adjusting volume. The project will involve connecting the buttons to the Raspberry Pi's GPIO pins and programming the Pi to respond to button presses with corresponding music playback actions.

### Materials:

* Raspberry Pi 4B
* SD card with Raspberry Pi OS
* HDMI cable and monitor (for initial setup)
* USB keyboard and mouse (for initial setup)
* Speakers or headphones for audio output
* Breadboard and jumper wires
* Pushbuttons (play/pause, next track, previous track, volume up, volume down)
* Resistors (as needed)
* Music files in a supported format (e.g., MP3)

### Steps:

* **Set up the Raspberry Pi:** Install the Raspberry Pi OS on the SD card and connect the Pi to a monitor, keyboard, and mouse. Configure the Pi and ensure it is connected to the internet.
* **Connect the buttons:** Use the breadboard and jumper wires to connect the pushbuttons to the Raspberry Pi's GPIO pins. Connect each button to a specific GPIO pin, and if needed, use resistors for button debouncing.
* **Install necessary software:** Install any required software libraries or packages for playing audio on the Raspberry Pi. One popular choice is the "pygame" library.
* **Write the code:** Using a programming language like Python, write a script that listens for button presses on the GPIO pins and performs corresponding actions. For example, when the play/pause button is pressed, the script should toggle the playback state. When the next track button is pressed, it should skip to the next track in the playlist. Similar actions should be defined for the previous track, volume up, and volume down buttons.
* **Implement the music playback:** Utilize the chosen audio library (e.g., pygame) to handle the music playback functionality. Load the music files, control the playback state, and adjust the volume level based on the button presses.
* **Test and refine:** Run the script and test the functionality of the buttons. Make any necessary adjustments to the code or circuitry to ensure reliable button detection and desired playback behavior.
* **Customize and enhance:** Consider adding additional features to the music player project, such as an LCD display to show the current track information or a rotary encoder for precise volume control.
* **Finalize the setup:** Once satisfied with the functionality, disconnect the monitor, keyboard, and mouse, and set up the Raspberry Pi with the necessary audio output (speakers or headphones) in the desired location.

With this project, you can create a simple and interactive music player using a Raspberry Pi 4B and buttons. Enjoy your favorite music with the convenience of physical controls!

Here's an example of a Python code snippet that demonstrates the basic functionality of a music player using buttons on a Raspberry Pi 4B:

### Demo code

**import pygame**

**import RPi.GPIO as GPIO**

**# Button GPIO pin numbers**

**PLAY\_PAUSE\_PIN = 18**

**NEXT\_TRACK\_PIN = 23**

**PREV\_TRACK\_PIN = 24**

**VOLUME\_UP\_PIN = 25**

**VOLUME\_DOWN\_PIN = 8**

**# Initialize GPIO settings**

**GPIO.setmode(GPIO.BCM)**

**GPIO.setup([PLAY\_PAUSE\_PIN, NEXT\_TRACK\_PIN, PREV\_TRACK\_PIN, VOLUME\_UP\_PIN, VOLUME\_DOWN\_PIN], GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)**

**# Initialize Pygame**

**pygame.mixer.init()**

**# Load music files (replace with your own)**

**music\_files = ['song1.mp3', 'song2.mp3', 'song3.mp3']**

**current\_track = 0**

**# Button event callbacks**

**def play\_pause(channel):**

**if pygame.mixer.music.get\_busy():**

**pygame.mixer.music.pause()**

**else:**

**pygame.mixer.music.unpause()**

**def next\_track(channel):**

**global current\_track**

**current\_track = (current\_track + 1) % len(music\_files)**

**pygame.mixer.music.load(music\_files[current\_track])**

**pygame.mixer.music.play()**

**def prev\_track(channel):**

**global current\_track**

**current\_track = (current\_track - 1) % len(music\_files)**

**pygame.mixer.music.load(music\_files[current\_track])**

**pygame.mixer.music.play()**

**def volume\_up(channel):**

**current\_volume = pygame.mixer.music.get\_volume()**

**if current\_volume < 1.0:**

**pygame.mixer.music.set\_volume(current\_volume + 0.1)**

**def volume\_down(channel):**

**current\_volume = pygame.mixer.music.get\_volume()**

**if current\_volume > 0.0:**

**pygame.mixer.music.set\_volume(current\_volume - 0.1)**

**# Register button event handlers**

**GPIO.add\_event\_detect(PLAY\_PAUSE\_PIN, GPIO.FALLING, callback=play\_pause, bouncetime=200)**

**GPIO.add\_event\_detect(NEXT\_TRACK\_PIN, GPIO.FALLING, callback=next\_track, bouncetime=200)**

**GPIO.add\_event\_detect(PREV\_TRACK\_PIN, GPIO.FALLING, callback=prev\_track, bouncetime=200)**

**GPIO.add\_event\_detect(VOLUME\_UP\_PIN, GPIO.FALLING, callback=volume\_up, bouncetime=200)**

**GPIO.add\_event\_detect(VOLUME\_DOWN\_PIN, GPIO.FALLING, callback=volume\_down, bouncetime=200)**

**# Start playing the first track**

**pygame.mixer.music.load(music\_files[current\_track])**

**pygame.mixer.music.play()**

**try:**

**# Keep the program running**

**while True:**

**pass**

**except KeyboardInterrupt:**

**# Clean up GPIO and Pygame on program exit**

**GPIO.cleanup()**

**pygame.mixer.quit()**

In this example, we use the pygame library for handling the audio playback. The code sets up GPIO pin numbers for each button, initializes the GPIO mode and setup, and initializes the Pygame mixer.

The music\_files list holds the paths or filenames of your music files. You can replace them with your own music files or modify the list accordingly.

Button event callbacks, such as **play\_pause(), next\_track(), prev\_track(), volume\_up(),** and **volume\_down()**, define the actions to be performed when the corresponding buttons are pressed. These functions use the Pygame mixer functions to control the music playback and volume.

The GPIO event detection registers the button event handlers to the corresponding GPIO pins and sets the falling edge detection with a debounce time to prevent false triggers.

The main loop (while True) keeps the program running until it's interrupted by a keyboard interrupt (Ctrl+C).

To install Pygame on a Raspberry Pi, you can follow these steps:

1. Update your system: Open a terminal window on your Raspberry Pi and run the following commands to ensure your system is up to date:

**sudo apt-get update**

**sudo apt-get upgrade**

1. Install dependencies: Pygame requires several dependencies to be installed. Run the following command to install them:

**sudo apt-get install libsdl-dev libsdl-image1.2-dev libsdl-mixer1.2-dev libsdl-ttf2.0-dev**

1. Install Pygame: Once the dependencies are installed, you can install Pygame using the pip package manager. Run the following command:

**pip install pygame**

1. Verify the installation: After the installation is complete, you can verify if Pygame is installed correctly. In the terminal, run the following command:

**python3 -m pygame.examples.aliens**

This command will run a Pygame example called "Aliens." If a Pygame window opens and you see a game with aliens, it means that Pygame is installed correctly.

That's it! Pygame should now be installed on your Raspberry Pi. You can start using it in your Python projects by importing the pygame module.

## Projects 7

With a Raspberry Pi 4B, a capacitor, several resistors, and two NPN transistors, you can create various electronic projects. Here's an example project idea:

**Project Name: Raspberry Pi Motion-Activated LED Control**

* **Description:** Build a motion-activated LED control system using a Raspberry Pi 4B, a capacitor, resistors, and NPN transistors. The project will involve using the Raspberry Pi's GPIO pins, along with the components, to detect motion using a sensor and control an LED based on the motion detection.

### Materials:

* Raspberry Pi 4B
* Breadboard and jumper wires
* PIR (Passive Infrared) motion sensor
* LED
* Capacitor (e.g., 10uF)
* Resistors (e.g., 220Ω, 10kΩ)
* NPN transistors (e.g., 2N3904 or similar)
* Power supply or USB power adapter for the Raspberry Pi
* Optional: Additional LEDs, pushbuttons, or other components for customization

### Steps:

* **Set up the hardware:** Connect the Raspberry Pi to the breadboard using jumper wires. Connect the PIR motion sensor to the Raspberry Pi's GPIO pins (e.g., power, ground, and signal pins). Connect the LED to the breadboard, along with the appropriate resistor to limit the current.
* **Assemble the circuit:** Connect the PIR motion sensor's output pin to a GPIO input pin on the Raspberry Pi. Connect the LED to the collector of an NPN transistor, with the emitter connected to the ground and the base connected to a GPIO output pin through a current-limiting resistor. Connect the capacitor between the base and ground of the NPN transistor to stabilize the circuit.
* **Configure the GPIO pins:** Write a Python script to configure the GPIO pins on the Raspberry Pi. Use a library like RPi.GPIO to set the necessary pin modes (input or output) and define the desired behavior.
* **Detect motion:** Use the Python script to continuously monitor the GPIO input pin connected to the motion sensor. When motion is detected, the GPIO pin will read a high signal. You can use the script to print a message or trigger an action when motion is detected.
* **Control the LED:** When motion is detected, use the GPIO output pin to drive the NPN transistor's base, turning it on and allowing current to flow through the LED. This will illuminate the LED, indicating the motion detection.
* **Test the project:** Run the Python script on the Raspberry Pi and test the motion detection and LED control. Ensure that the LED turns on when motion is detected and turns off when there is no motion.
* **Customize and expand:** You can customize the project further by adding more LEDs, integrating pushbuttons for manual control, or incorporating other sensors or actuators based on your preferences and creativity.

Remember to follow proper safety precautions, double-check the pin connections, and ensure that the circuit and components are correctly powered and grounded.

This project demonstrates a basic motion-activated LED control system using Raspberry Pi and electronic components. You can further enhance and modify the project based on your specific requirements and interests.

### Demo code

Here's an example Python code snippet that demonstrates the basic functionality of a motion-activated LED control system using a Raspberry Pi 4B, a capacitor, resistors, and NPN transistors:

**import RPi.GPIO as GPIO**

**import time**

**# GPIO pin numbers**

**MOTION\_PIN = 17**

**LED\_PIN = 18**

**# Initialize GPIO settings**

**GPIO.setwarnings(False)**

**GPIO.setmode(GPIO.BCM)**

**GPIO.setup(MOTION\_PIN, GPIO.IN)**

**GPIO.setup(LED\_PIN, GPIO.OUT)**

**# Main program loop**

**try:**

**while True:**

**if GPIO.input(MOTION\_PIN):**

**# Motion detected**

**print("Motion detected!")**

**GPIO.output(LED\_PIN, GPIO.HIGH) # Turn on LED**

**time.sleep(1) # LED on duration**

**GPIO.output(LED\_PIN, GPIO.LOW) # Turn off LED**

**else:**

**# No motion**

**GPIO.output(LED\_PIN, GPIO.LOW) # Turn off LED**

**time.sleep(0.1) # Delay between readings**

**except KeyboardInterrupt:**

**# Clean up GPIO on program exit**

**GPIO.cleanup()**

In this code, we use the **RPi.GPIO** library to control the GPIO pins on the Raspberry Pi. The motion sensor is connected to the GPIO pin specified as MOTION\_PIN, and the LED is connected to LED\_PIN.

The program continuously checks the input state of the motion sensor using **GPIO.input()** in a while loop. If motion is detected **(GPIO.input(MOTION\_PIN) returns True)**, it prints a message, turns on the LED by setting **GPIO.output(LED\_PIN, GPIO.HIGH)**, waits for a duration **(time.sleep(1))**, and then turns off the LED by setting **GPIO.output(LED\_PIN, GPIO.LOW).**

If no motion is detected, the LED remains off. The program repeats this loop until it is interrupted by a keyboard interrupt **(Ctrl+C),** at which point it cleans up the GPIO settings.

Make sure to adjust the MOTION\_PIN and LED\_PIN values in the code to match your actual GPIO pin connections. Additionally, ensure that you have installed the RPi.GPIO library on your Raspberry Pi (**sudo apt-get install python3-rpi.gpio**) before running the code.

With this code, the LED will turn on when motion is detected by the PIR motion sensor, providing a simple demonstration of a motion-activated LED control system.

## Project 8

The 6N137 chip is an optocoupler that can be used to isolate and protect sensitive components of a circuit from potentially harmful voltages or noise.

When it comes to using the 6N137 chip with a Raspberry Pi 4B, here are a few experiment ideas:

### Isolated Digital Input:

Use the 6N137 chip to create an isolated digital input circuit. Connect the input side of the optocoupler to an external device or signal source, and the output side to one of the GPIO pins of the Raspberry Pi.

This setup allows you to safely interface with external devices, protecting the Raspberry Pi from voltage spikes or electrical noise.

### Optically Isolated Serial Communication:

Utilize the 6N137 chip to create an optically isolated serial communication interface. Connect the transmit (TX) and receive (RX) lines of the Raspberry Pi's UART to the input and output sides of the optocoupler.

This configuration enables isolation and protection when communicating with other devices via serial protocols, especially when dealing with devices operating at different voltage levels.

### Isolated I2C Communication:

Create an isolated I2C communication interface using the 6N137 chip. Connect the SDA (data) and SCL (clock) lines of the Raspberry Pi's I2C bus to the input and output sides of the optocoupler.

This setup provides isolation and protection when communicating with I2C devices that may have different ground references or operate at different voltage levels.

### Isolated PWM Output:

Use the 6N137 chip to generate an isolated PWM (Pulse Width Modulation) signal. Connect the PWM output of the Raspberry Pi to the input side of the optocoupler, and the output side to an external device or circuit.

This experiment allows you to control external devices or circuits that require isolation, such as motor drivers or power control circuits.

### Optically Isolated Interrupts:

Implement an optically isolated interrupt system using the 6N137 chip. Connect an external interrupt signal to the input side of the optocoupler and connect the output side to one of the Raspberry Pi's GPIO pins configured as an interrupt input.

This setup enables you to safely handle external interrupt events without risking damage to the Raspberry Pi's GPIO pins.

These experiment ideas demonstrate some of the applications of the 6N137 chip with a Raspberry Pi 4B. Remember to consult the datasheet and follow proper circuit design guidelines for the chip to ensure safe and reliable operation.

Here's an example of a simple experiment using the 6N137 chip:

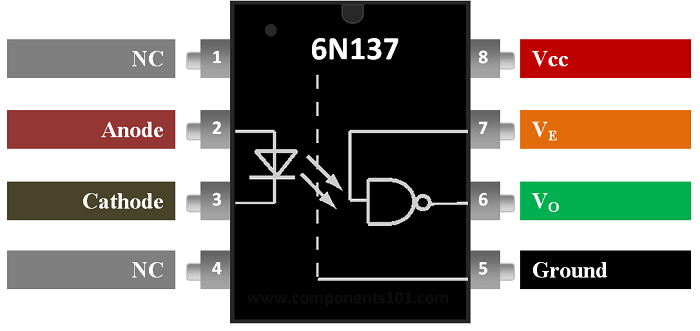
**Objective:** To create an optically isolated digital input circuit using the 6N137 chip and Raspberry Pi to detect the status of an external switch.

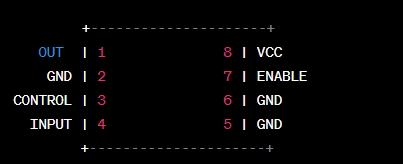
### Components:

* Raspberry Pi 4B
* 6N137 chip
* Push-button switch
* Breadboard
* Jumper wires

### Circuit Diagram:

* 6N137 Pinout





### Circuit Connections:

* Connect the VCC pin (Pin 8) of the 6N137 chip to the 3.3V pin (Pin 1) of the Raspberry Pi.
* Connect the GND pin (Pin 5) of the 6N137 chip to a ground pin (Pin 6 or 9) of the Raspberry Pi.
* Connect the input pin (Pin 2) of the 6N137 chip to one terminal of the push-button.
* Connect the other terminal of the push-button to the GND pin (Pin 6 or 9) of the Raspberry Pi.
* Connect a current-limiting resistor (e.g., 330 ohms) in series with the anode of the LED.
* Connect the cathode of the LED to a GPIO pin (e.g., Pin 17) of the Raspberry Pi

### Pin Descriptions:

* **OUT:** Output signal from the 6N137 chip. This carries the isolated signal.
* **GND:** Ground connection.
* **CONTROL:** Control input for enabling or disabling the output. Typically connected to VCC or left floating for continuous operation.
* **INPUT:** Input signal to be isolated.

Note: The pinout may vary slightly depending on the specific manufacturer or package type. Always refer to the datasheet or documentation provided by the manufacturer for the most accurate and up-to-date information.

### Demo code

**import RPi.GPIO as GPIO**

**# Set GPIO mode and input pin**

**GPIO.setmode(GPIO.BCM)**

**input\_pin = 17 # Change to your desired GPIO pin**

**# Configure GPIO pin as input**

**GPIO.setup(input\_pin, GPIO.IN)**

**# Read input signal and print its value**

**try:**

**while True:**

**input\_value = GPIO.input(input\_pin)**

**print("Input value:", input\_value)**

**except KeyboardInterrupt:**

**GPIO.cleanup()**

#### Explanation:

In this code, we use the RPi.GPIO library to interact with the GPIO pins. Replace 18 and 17 with the GPIO pin numbers you connected the input and output pins to, respectively.

When you run this code, it continuously reads the input signal from the 6N137 chip (which is connected to the push-button) and detects button presses. When the button is pressed, it turns on the LED connected to the Raspberry Pi, and when the button is released, it turns off the LED.

Make sure you have the RPi.GPIO library installed on your Raspberry Pi. If not, you can install it using the following command:

**pip install RPi.GPIO**

That's it! Your circuit is set up, and the demo code will read the input signal from the 6N137 chip. Adjust the code to suit your specific application and signal requirements.

## Project 9

### Connecting DHT11 Sensor with Raspberry Pi 4/3 using Python

You might have heard of Raspberry Pi which is a single board computer and it is very popular among student and electronics hobbyist. It is compact and advanced device. We will also use a DHT11 Temperature and Humidity sensor. In this tutorial we will learn connecting DHT11 Sensor with Raspberry Pi 4B using Python.

The sensor is connected using one wire for sending the data signal. These are quite popular and used in different projects like soil monitors, remote weather stations, home automation systems etc. You can also use DHT11 sensor with Raspberry Pi 4 using python code.

Collecting data using DHT11 Temperature and humidity sensor using Raspberry Pi is simple. In this tutorial, we will demonstrate how to connect the DHT11 sensor with Raspberry Pi and get the output i.e. temperature and humidity readings. We will be doing this using python programming. In fact, we will be extracting DHT11 sensor data using Python.

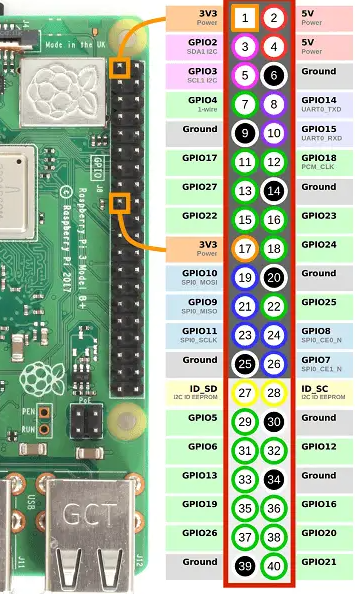
### Parts List

You can use any raspberry Pi you have.

* Raspberry Pi 4B
* DHT11 sensor
* Jumper Cables

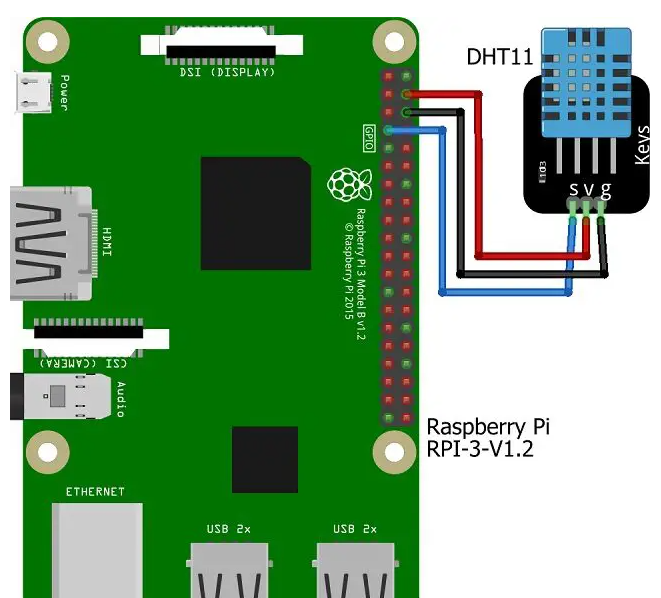
### Raspberry Pi Pin Configuration

Below diagram shows pin definitions of Raspberry Pi. In this tutorial we have connected it with Pin 7 which is GPIO4 of Raspberry Pi. GPIO pin configurations are same for all Raspberry Pi.



### Connection Diagram

Here we have connected DHT11 temperature and Humidity sensor with Raspberry Pi.



### Installation of required Libraries

First we have to install Adfruit DHT library for this tutorial. Login to raspberry pi via VNC Viewer or SSH (Using putty) and run the below code.

**git clone** [**https://github.com/adafruit/Adafruit\_Python\_DHT.git**](https://github.com/adafruit/Adafruit_Python_DHT.git)

It will download the necessary files under a folder named Adafruit\_Python\_DHT. Once your download in completed run the command “ls” for listing the files and folders. You will see Adafruit\_Python\_DHT is listed there.

**pi@raspberrypi:~ $ ls**

**Adafruit\_Python\_DHT Desktop**

Now go to Adafruit\_Python\_DHT folder as shown below

**pi@raspberrypi:~ $ cd Adafruit\_Python\_DHT/**

Run the below command for installation of Python version 2 and 3

**sudo python setup.py install**

**sudo python3 setup.py install**

**pi@raspberrypi:~/Adafruit\_Python\_DHT $ sudo python setup.py install**

**pi@raspberrypi:~/Adafruit\_Python\_DHT $ sudo python3 setup.py install**

Once the python installation please make sure GPIO is enabled in your raspberry Pi. Run the below command and enable it.

**pi@raspberrypi:~ $ sudo raspi-config**

**pi@raspberrypi:~/Adafruit\_Python\_DHT $ cd**

### How to run the same code in Raspberry Pi 4?

For executing the same python code in Raspberry Pi 4 we have to make some changes. The above library supports till Raspberry Pi 3. Now to make it compatible with Raspberry Pi 4 we have to make below changes.

Go to below directory

**/usr/local/lib/python3.7/dist-packages/Adafruit\_DHT/**

**pi@raspberrypi:~ $ cd /usr/local/lib/python3.7/dist-packages/Adafruit\_DHT/**

Again list the files and folders with “ls” command as shown below

**pi@raspberrypi:/usr/local/lib/python3.7/dist-packages/Adafruit\_DHT $ ls**

**Beaglebone\_Black.py \_\_pycache\_\_**

**common.py Raspberry\_Pi\_2\_Driver.cpython-37m-arm-linux-gnueabihf.so**

**common.py.bup Raspberry\_Pi\_2.py**

**\_\_init\_\_.py Raspberry\_Pi.py**

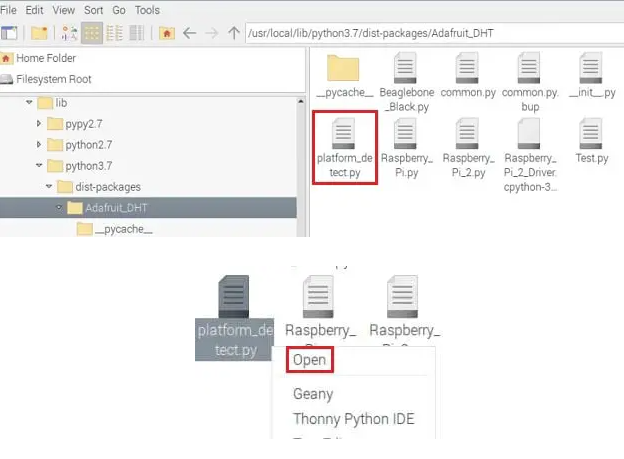
**platform\_detect.py Test.py**

You will see a file named platform\_detect.py, This is the file we have to edit. Since this file has read only permissions hence we can’t make changes and save it. In order to change this file, we have to change the permission of this file. you can run below command to change the permission.

**sudo chmod a+w platform\_detect.py**

**pi@raspberrypi:/usr/local/lib/python3.7/dist-packages/Adafruit\_DHT# sudo chmod a+w platform\_detect.py**

Now go to directory /usr/local/lib/python3.7/dist-packages/Adafruit\_DHT and open the file platform\_detect.py.

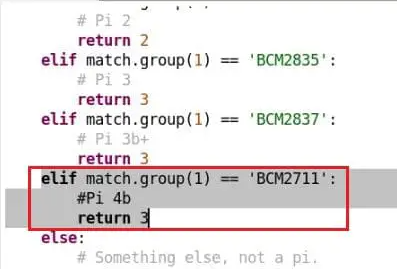


Once you open it, add the below lines of code at the bottom and save it.

**elif match.group(1) == 'BCM2711':**

**#Pi 4b**

**return 3**

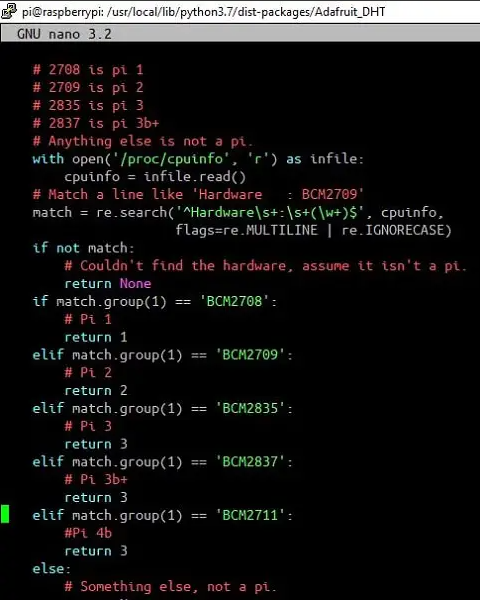


### Alternate way to edit this file

You can edit **platform\_detect.py** using putty by remote SSH session. First navigate to **usr/local/lib/python3.7/dist-packages/Adafruit\_DHT** directory and then run the below command.

**nano platform\_detect.py**

**pi@raspberrypi:/usr/local/lib/python3.7/dist-packages/Adafruit\_DHT $ nano platform\_detect.py**

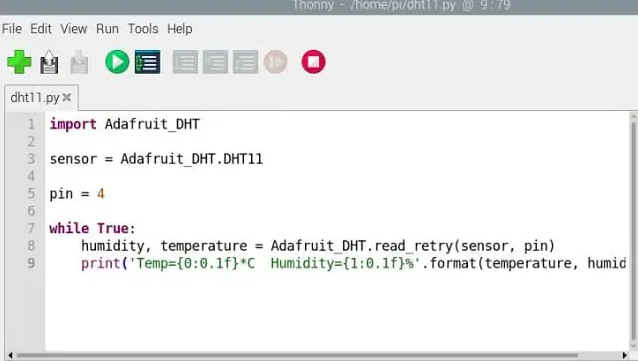


You can add the same set of code here and save the file.

### Coding

You can copy paste the below code in an empty text file and save it with extension .py. You can create a new file in /home/pi directory and name it anything you want. Make sure the extension should be “.py “.

Now right click and open that file, the file will get opened using Thonny python editor. Now you can copy the code below and paste it there and save it.



### Demo Code:

**import Adafruit\_DHT**

**sensor = Adafruit\_DHT.DHT11**

**pin = 4**

**while True:**

**humidity, temperature = Adafruit\_DHT.read\_retry(sensor, pin)**

**print('Temp={0:0.1f} \*C Humidity={1:0.1f} %'.format(temperature, humidity))**

### Construction

As per the circuit diagram we have connected the sensor with Raspberry Pi 3 and same done with Raspberry Pi 4 as well since the GPIO pins are same.

### Executing the Code & Testing

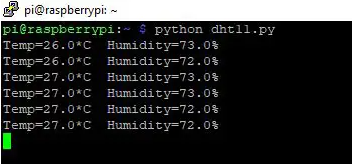
Executing the code is slightly different in Raspberry Pi 3 and Pi 4.

#### In Raspberry Pi 3

You can run the code either logging into Raspberry Pi or via taking SSH using Putty. Once you have the code file ready then you can run the below command. Here we have named the code file as “dht11.py “.

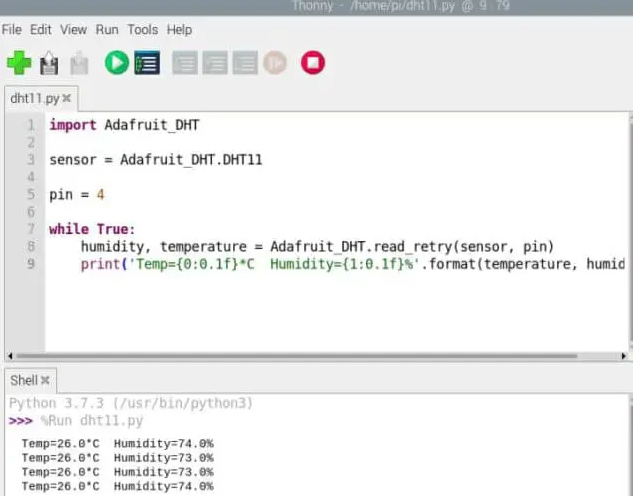
**sudo python dht11.py**

If everything is correct the you will get a output on you SSH terminal as shown below.



#### In Raspberry Pi 4

For Raspberry Pi 4 you have to open the code file and by default it will open using Thonny python editor. Now click the Play button which is green in color. Just below that window you will get the output as shown below.



### Conclusion

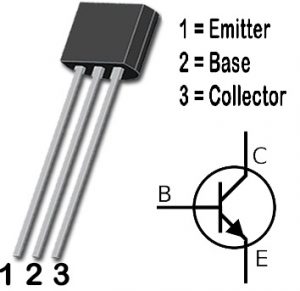
This is a simple project interfacing DHT11 temperature and humidity sensor with Raspberry Pi 3 and Raspberry Pi 4 using a simple python code. Any beginner can try this and build this as a project to understand the concept of interfacing DHT11 module with Pi’s. Try to build this and comment below if you facing any challenges.

## Project 10

The S8050 is a widely used transistor that belongs to the NPN (Negative-Positive-Negative) family of bipolar junction transistors (BJTs). It is commonly employed for general-purpose amplification and switching applications in electronic circuits. Here are some key aspects of the S8050 chip:

**Transistor Type**: The S8050 is an NPN transistor, meaning it has a P-doped base region sandwiched between two N-doped emitter and collector regions.

**Pin Configuration**: The S8050 typically has three pins:



* **Emitter (E):** The N-doped region through which the majority charge carriers (electrons) flow out.
* **Base (B):** The P-doped region that controls the flow of charge carriers between the emitter and collector.
* **Collector (C):** The N-doped region through which the majority charge carriers (electrons) flow into.
* **Amplification:** The S8050 is primarily utilized as an amplifier in electronic circuits. It can be used to amplify weak signals or as a building block in audio amplifiers, oscillators, and other signal processing circuits.

**Switching Applications:** The S8050 can also function as a switch to control the flow of current in electronic circuits. It can be used to turn on or off other components or devices based on the input signal.

**Voltage and Current Ratings:** The S8050 has specific voltage and current ratings that determine its safe operating limits. It is crucial to refer to the datasheet of the S8050 transistor to ensure its appropriate usage within these specifications.

**Package Types:** The S8050 transistor is available in various package types, such as TO-92, SOT-23, or SOT-223, which provide different physical dimensions and pin layouts.

When using the S8050 transistor in a circuit, it is important to consider its pin configuration, voltage and current requirements, and consult the datasheet for detailed specifications. Understanding the characteristics of the S8050 allows engineers and hobbyists to design and implement amplification or switching circuits effectively.

While the S8050 transistor is not directly compatible with the Raspberry Pi's GPIO pins, it can still be used in conjunction with the Raspberry Pi 4B for various purposes. Here are a few examples:

* **Level Shifting:** The S8050 can be utilized as a level shifter to interface between devices operating at different voltage levels. It can convert the Raspberry Pi's 3.3V logic levels to higher voltage levels, allowing communication with devices that operate at higher voltages.
* **Switching Loads:** The S8050 can act as a switch to control the flow of current to external components or loads. By connecting the base of the transistor to a GPIO pin on the Raspberry Pi and controlling its on/off state, you can control the operation of motors, relays, LEDs, or other higher-powered devices.
* **Pulse Width Modulation (PWM):** The S8050 can be used in combination with the Raspberry Pi's PWM output to control the speed or intensity of motors, LEDs, or other devices that support PWM input. By adjusting the duty cycle of the PWM signal, you can vary the output's average power or brightness.
* **Amplification**: If you have an analog input signal that requires amplification, you can design a simple amplifier circuit using the S8050 transistor. This can be useful for tasks such as amplifying small audio signals or sensor outputs before processing them with the Raspberry Pi.

It's important to note that when using the S8050 transistor with the Raspberry Pi, you'll need to consider the voltage and current requirements of your specific application, and ensure proper circuit design and connection. Additionally, consult the S8050 transistor's datasheet for detailed specifications and characteristics to ensure its appropriate usage.

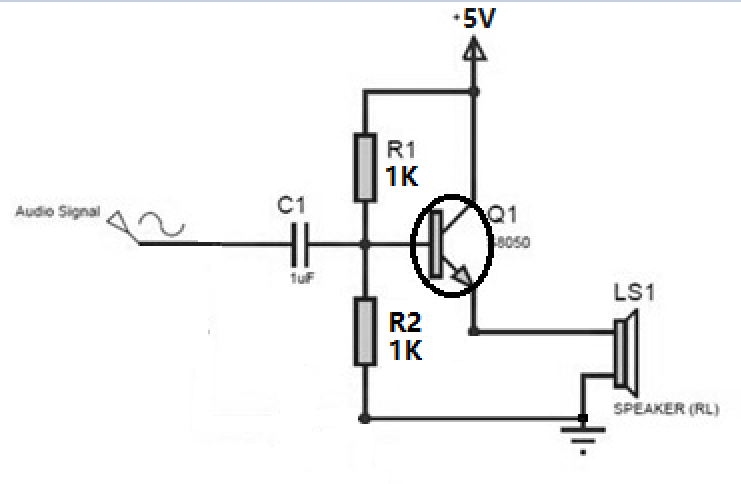
To build a basic amplification circuit using the S8050 transistor with a Raspberry Pi 4B, you can follow these steps:

**Note: This circuit provides a simple amplification example and may not produce high-quality audio amplification.**

### Components needed:

* S8050 transistor
* Resistors (R1 and R2) 1K
* Capacitor (C1) 1uF
* Audio source (e.g., audio signal from Raspberry Pi's audio output) GPIO 12 or 13
* Buzzer/speaker/headphones
* Jumper wires
* Breadboard
* Raspberry Pi 4B

### Circuit Diagram:



### Procedure:

* Connect the positive terminal of the audio source (e.g., Raspberry Pi's audio output) to the junction of R1 and the transistor's base. here we are using GPIO12 or GPIO 13 to be left and right channel of sound output channel, we need to change /boot/config.txt file before using this function, Open and adding following paragraph into the file:

**dtoverlay=audremap**

Save it and reboot your Raspberry Pi 4B.

* Connect the negative terminal of the audio source to the ground (GND) of the Raspberry Pi and the circuit.
* Connect the collector of the transistor (the junction of R2 and the transistor's collector) to one terminal of the speaker or headphones.
* Connect the other terminal of the speaker or headphones to the ground (GND) of the Raspberry Pi and the circuit.
* Place a coupling capacitor (C1) 1uF between the collector of the transistor and the buzzer/speaker/headphones to block any DC component from reaching the speaker.
* Connect a resistor (R2) between the collector of the transistor and the positive power supply (+5V).
* Connect the emitter of the transistor to the ground (GND) of the circuit and the Raspberry Pi.
* Make sure to choose appropriate resistor values for R1 and R2 based on your specific requirements and the characteristics of the S8050 transistor, in this experiment, we are using 1K resistors as R1 and R2. You may need to experiment with different resistor values to achieve the desired amplification level.

Additionally, ensure that the audio signal from the Raspberry Pi is compatible with the input requirements of the amplification circuit. You may need to adjust the volume levels on the Raspberry Pi to prevent distortion or clipping in the amplified output.

Please note that this is a basic amplification circuit, and for more sophisticated audio amplification or higher-quality results, you may need to explore dedicated audio amplifier ICs or modules specifically designed for audio amplification purposes

After building circuit, login to Raspberry Pi and open a terminal, upload or record a mp3 audio file and play it with cvlc command:

**sudo cvlc Music/my\_song.mp3**

And then, you will find your Raspberry Pi 4B’s GPIO 12 or GPIO13 will trigger the S8050 to drive buzzer play audio!!! have fun!

## Project 11

To build a fun project using an ultrasonic sensor (HC-SR04) with a Raspberry Pi 4B, you can create a distance measurement system that can trigger different actions or play sounds based on the distance detected. In this example, we'll create a simple script that prints messages depending on the measured distance.

First, gather the following hardware components:

* Raspberry Pi 4B
* HC-SR04 ultrasonic sensor
* Breadboard
* Jumper wires
* 1K ohm resistor
* 2K ohm resistor

Next, follow these steps to set up the project:

### 1. Connect the hardware components:

* Connect the **VCC** pin on the HC-SR04 sensor to the **5V** pin on the Raspberry Pi.
* Connect the **GND** pin on the sensor to a **GND** pin on the Raspberry Pi.
* Connect the **TRIG** pin on the sensor to **GPIO 18** (physical pin 12).
* Connect the **ECHO** pin on the sensor to a breadboard row and use the **1K** and **2K** ohm resistors to create a voltage divider. Connect the connection point of both resistors to **GPIO 24 (physical pin 18)** and the other side to a **GND** pin on the Raspberry Pi.

### 2. Set up the Raspberry Pi:

Ensure your Raspberry Pi is running the latest version of Raspbian and is connected to the internet.

### 3. Install required Python libraries:

- Open a terminal window and run the following commands:

**sudo apt-get update**

**sudo apt-get upgrade**

**sudo apt-get install python3-gpiozero**

### 4. Create a new Python script:

- Open a text editor and create a new file called **`ultrasonic\_distance.py`.**

### 5. Copy and paste the following demo code into the new file:

**import RPi.GPIO as GPIO**

**import time**

**# Set up GPIO modes and pins**

**GPIO.setmode(GPIO.BCM)**

**TRIG = 18**

**ECHO = 24**

**GPIO.setup(TRIG, GPIO.OUT)**

**GPIO.setup(ECHO, GPIO.IN)**

**def measure\_distance():**

**# Trigger a pulse and wait for the echo**

**GPIO.output(TRIG, True)**

**time.sleep(0.00001)**

**GPIO.output(TRIG, False)**

**start\_time = time.time()**

**end\_time = time.time()**

**# Measure the time between sending and receiving the pulse**

**while GPIO.input(ECHO) == 0:**

**start\_time = time.time()**

**while GPIO.input(ECHO) == 1:**

**end\_time = time.time()**

**# Calculate the distance in centimeters**

**elapsed\_time = end\_time - start\_time**

**distance = (elapsed\_time \* 34300) / 2**

**return distance**

**if \_\_name\_\_ == '\_\_main\_\_':**

**try:**

**while True:**

**distance = measure\_distance()**

**if distance < 10:**

**print("Very close!")**

**elif distance < 50:**

**print("Close")**

**else:**

**print("Far away")**

**time.sleep(1)**

**except KeyboardInterrupt:**

**print("Measurement stopped.")**

**GPIO.cleanup()**

### 6. Save the file and close the text editor.

### 7. Run the script:

- Open a terminal window and navigate to the directory where you saved the script. Run the following command to execute the script:

**python3 ultrasonic\_distance.py**

The script will continuously measure the distance in front of the sensor and print a message depending on the detected distance. You can modify the script to include more actions, change the distance values, or trigger sounds using additional components like a buzzer.

## Project 12

Here's a fun project idea using Raspberry Pi 4B and a 5g servo motor - create an automatic cat laser pointer!

This will move a laser pointer in a random pattern, providing amusement for your cat as it chases the red dot.

### Required components:

* Raspberry Pi 4B
* 5g servo motor (180-degree)
* Laser diode (usually 5V)
* Jumper wires
* Breadboard (optional, for prototyping)

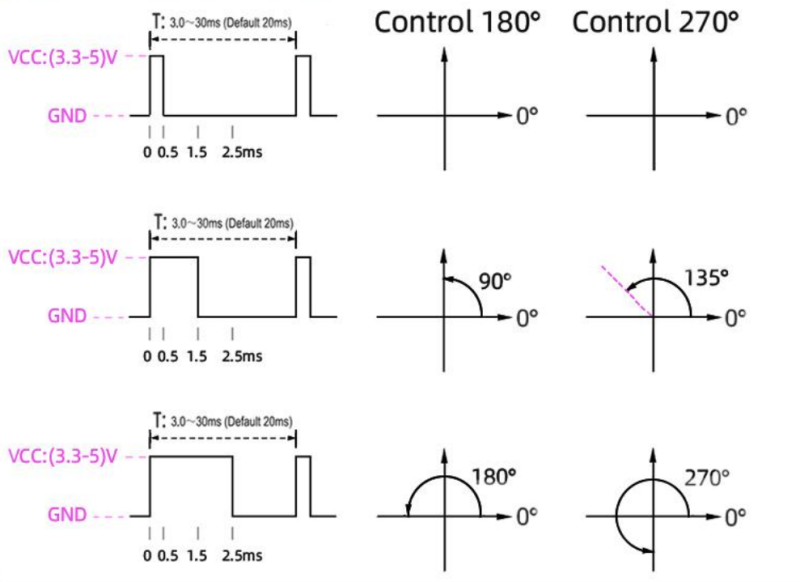
### Steps:

#### 1. Connect the servo motor to the Raspberry Pi GPIO pins.

Connect the servo's control pin to a GPIO with PWM capability (for example, GPIO18).

* Servo's VCC -> Raspberry Pi's 5V (Pin 2 or 4)
* Servo's GND -> Raspberry Pi's GND (Pin 6, 9, 14, etc.)
* Servo's Control -> Raspberry Pi's GPIO18 (Pin 12)





#### 2. Connect the laser diode to the Raspberry Pi GPIO pins.

* Laser diode's VCC -> Raspberry Pi's GPIO (for example, GPIO23 on Pin 16)
* Laser diode's GND -> Raspberry Pi's GND (Pin 6, 9, 14, etc.)

#### 3. Install the required software and libraries.

Make sure your Raspberry Pi is running the latest Raspbian (or other similar) OS and connected to the internet. Next, open a terminal and install the required libraries:

**sudo apt-get update sudo apt-get install python3-gpiozero sudo apt-get install pigpio python-pigpio python3-pigpio**

#### 4. Create a Python script to control the servo and laser diode.

Create a new Python script (e.g., `cat\_laser.py`) with the following code:

**import time**

**import random**

**from gpiozero import Servo, OutputDevice**

**import pigpio**

**GPIO\_SERVO = 18**

**GPIO\_LASER = 23**

**pi = pigpio.pi()**

**servo = Servo(GPIO\_SERVO, pin\_factory=pi)**

**# Assumes default 50 Hz frequency, adjust if needed**

**laser = OutputDevice(GPIO\_LASER)**

**# Function to randomly change the angle of the servo**

**def random\_angle():**

**return random.uniform(-1, 1) # Function to toggle the laser state**

**def toggle\_laser(state):**

**if state == 'ON':**

**laser.on()**

**else:**

**laser.off()**

**try:**

**toggle\_laser('ON') # Turn on the laser**

**while True:**

**servo.value = random\_angle()**

**time.sleep(random.uniform(0.5, 2))**

**except KeyboardInterrupt:**

**toggle\_laser('OFF') # Turn off the laser, when CTRL+C is pressed**

**finally: pi.stop() # Stop the pigpio library**

#### 5. Run the script and observe the servo motor and laser diode.

Execute the script:

**python3 cat\_laser.py**

The servo motor should move randomly, and the laser diode should be active. Please remember that never point a laser diode directly into eyes, as it could cause harm or injury. Also, ensure that the script is stopped and the laser diode off when not in use.

## Project 13

Connecting an LCD to your Raspberry Pi will spice up almost any project, but what if your pins are tied up with connections to other modules? No problem, just connect your LCD with I2C, it only uses two pins (well, four if you count the ground and power).

In this tutorial, I’ll show you everything you need to set up an LCD using I2C, but if you want to learn more about I2C and the details of how it works. just following this tutorial.

There are a couple ways to use I2C to connect an LCD to the Raspberry Pi. The simplest is to get an LCD with an I2C backpack. But the hardcore DIY way is to use a standard HD44780 LCD and connect it to the Pi via a chip called the PCF8574.

The PCF8574 converts the I2C signal sent from the Pi into a parallel signal that can be used by the LCD. Most I2C LCDs use the PCF8574 anyway. I’ll explain how to connect it both ways in a minute.

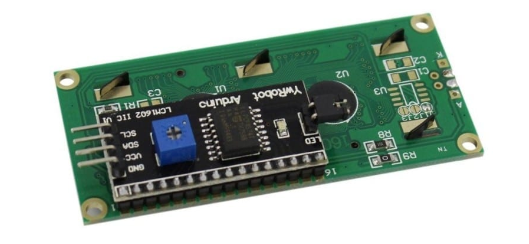
I’ll also show you how to program the LCD using Python, and provide examples for how to print and position the text, clear the screen, scroll text, print data from a sensor, print the date and time, and print the IP address of your Pi.

If you don’t have an I2C enabled LCD or a PCF8574, these tutorials will show you how to connect an LCD with the GPIO pins:

### CONNECT THE LCD

I2C (inter-integrated circuit) is also known as the two-wire interface since it only uses two wires to send and receive data. Actually it takes four if you count the VCC and Ground wires, but the power could always come from another source.

### CONNECTING AN I2C ENABLED LCD



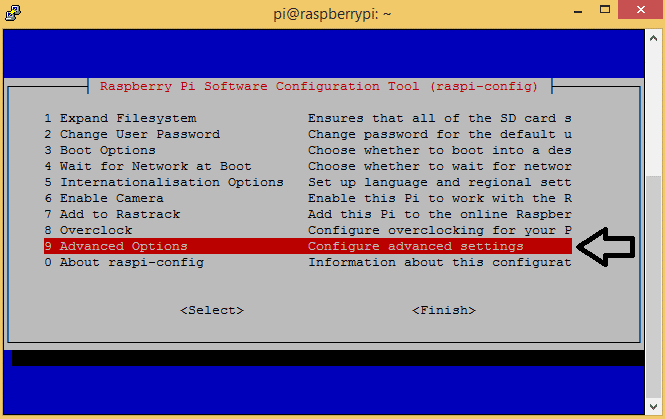
Connecting an LCD with an I2C backpack is pretty self-explanatory. Connect the SDA pin on the Pi to the SDA pin on the LCD, and the SCL pin on the Pi to the SCL pin on the LCD. The ground and Vcc pins will also need to be connected. Most LCDs can operate with 3.3V, but they’re meant to be run on 5V, so connect it to the 5V pin of the Pi if possible.

### ENABLE I2C ON THE PI

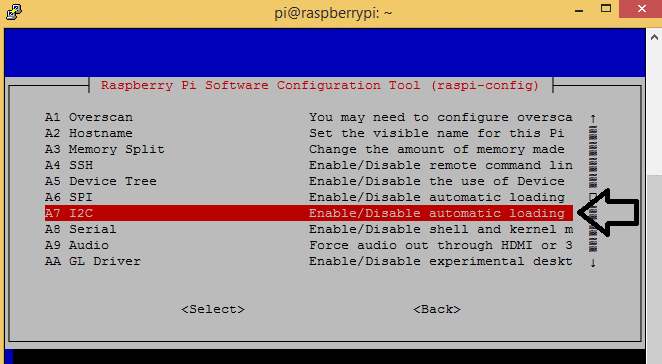
Before we get into the programming, we need to make sure the I2C module is enabled on the Pi and install a couple tools that will make it easier to use I2C.

### ENABLE I2C IN RASPI-CONFIG

First, log in to your Pi and enter **sudo raspi-config** to access the configuration menu. Then arrow down and select “**Advanced Settings”**:



Now arrow down and select **“I2C Enable/Disable automatic loading”**:



Choose **“Yes**” at the next prompt, exit the configuration menu, and reboot the Pi to activate the settings.

### INSTALL I2C-TOOLS AND SMBUS

Now we need to install a program called **I2C-tools**, which will tell us the I2C address of the LCD when it’s connected to the Pi. So at the command prompt, enter

**sudo apt-get install i2c-tools.**

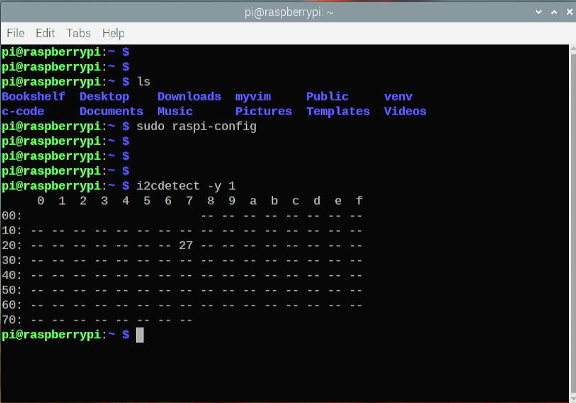
Next we need to install SMBUS, which gives the Python library we’re going to use access to the I2C bus on the Pi. At the command prompt, enter

**sudo apt-get install python3-smbus.**

Now reboot the Pi and log in again. With your LCD connected, enter

**i2cdetect -y 1**

at the command prompt. This will show you a table of addresses for each I2C device connected to your Pi:



The I2C address of my LCD is 27. Take note of this number, we’ll need it later.

### PROGRAMMING THE LCD

We’ll be using Python to program the LCD, so if this is your first time writing/running a Python program, you may want to check out How to Write and Run a Python Program on the Raspberry Pi before proceeding.

### INSTALLING THE LIBRARY

I found a Python I2C library that has a good set of functions and works pretty well. This library was originally posted here, then expanded and improved by GitHub user DenisFromHR.

Copy this code for the library, then save it in a file named **I2C\_LCD\_driver.py:**

**# -\*- coding: utf-8 -\*-**

**# Original code found at:**

**# https://gist.github.com/DenisFromHR/cc863375a6e19dce359d**

**"""**

**Compiled, mashed and generally mutilated 2014-2015 by Denis Pleic**

**Made available under GNU GENERAL PUBLIC LICENSE**

**# Modified Python I2C library for Raspberry Pi**

**# as found on http://www.recantha.co.uk/blog/?p=4849**

**# Joined existing 'i2c\_lib.py' and 'lcddriver.py' into a single library**

**# added bits and pieces from various sources**

**# By DenisFromHR (Denis Pleic)**

**# 2015-02-10, ver 0.1**

**"""**

**# i2c bus (0 -- original Pi, 1 -- Rev 2 Pi)**

**I2CBUS = 0**

**# LCD Address**

**ADDRESS = 0x27**

**import smbus**

**from time import sleep**

**class i2c\_device:**

**def \_\_init\_\_(self, addr, port=I2CBUS):**

**self.addr = addr**

**self.bus = smbus.SMBus(port)**

**# Write a single command**

**def write\_cmd(self, cmd):**

**self.bus.write\_byte(self.addr, cmd)**

**sleep(0.0001)**

**# Write a command and argument**

**def write\_cmd\_arg(self, cmd, data):**

**self.bus.write\_byte\_data(self.addr, cmd, data)**

**sleep(0.0001)**

**# Write a block of data**

**def write\_block\_data(self, cmd, data):**

**self.bus.write\_block\_data(self.addr, cmd, data)**

**sleep(0.0001)**

**# Read a single byte**

**def read(self):**

**return self.bus.read\_byte(self.addr)**

**# Read**

**def read\_data(self, cmd):**

**return self.bus.read\_byte\_data(self.addr, cmd)**

**# Read a block of data**

**def read\_block\_data(self, cmd):**

**return self.bus.read\_block\_data(self.addr, cmd)**

**# commands**

**LCD\_CLEARDISPLAY = 0x01**

**LCD\_RETURNHOME = 0x02**

**LCD\_ENTRYMODESET = 0x04**

**LCD\_DISPLAYCONTROL = 0x08**

**LCD\_CURSORSHIFT = 0x10**

**LCD\_FUNCTIONSET = 0x20**

**LCD\_SETCGRAMADDR = 0x40**

**LCD\_SETDDRAMADDR = 0x80**

**# flags for display entry mode**

**LCD\_ENTRYRIGHT = 0x00**

**LCD\_ENTRYLEFT = 0x02**

**LCD\_ENTRYSHIFTINCREMENT = 0x01**

**LCD\_ENTRYSHIFTDECREMENT = 0x00**

**# flags for display on/off control**

**LCD\_DISPLAYON = 0x04**

**LCD\_DISPLAYOFF = 0x00**

**LCD\_CURSORON = 0x02**

**LCD\_CURSOROFF = 0x00**

**LCD\_BLINKON = 0x01**

**LCD\_BLINKOFF = 0x00**

**# flags for display/cursor shift**

**LCD\_DISPLAYMOVE = 0x08**

**LCD\_CURSORMOVE = 0x00**

**LCD\_MOVERIGHT = 0x04**

**LCD\_MOVELEFT = 0x00**

**# flags for function set**

**LCD\_8BITMODE = 0x10**

**LCD\_4BITMODE = 0x00**

**LCD\_2LINE = 0x08**

**LCD\_1LINE = 0x00**

**LCD\_5x10DOTS = 0x04**

**LCD\_5x8DOTS = 0x00**

**# flags for backlight control**

**LCD\_BACKLIGHT = 0x08**

**LCD\_NOBACKLIGHT = 0x00**

**En = 0b00000100 # Enable bit**

**Rw = 0b00000010 # Read/Write bit**

**Rs = 0b00000001 # Register select bit**

**class lcd:**

**#initializes objects and lcd**

**def \_\_init\_\_(self):**

**self.lcd\_device = i2c\_device(ADDRESS)**

**self.lcd\_write(0x03)**

**self.lcd\_write(0x03)**

**self.lcd\_write(0x03)**

**self.lcd\_write(0x02)**

**self.lcd\_write(LCD\_FUNCTIONSET | LCD\_2LINE | LCD\_5x8DOTS | LCD\_4BITMODE)**

**self.lcd\_write(LCD\_DISPLAYCONTROL | LCD\_DISPLAYON)**

**self.lcd\_write(LCD\_CLEARDISPLAY)**

**self.lcd\_write(LCD\_ENTRYMODESET | LCD\_ENTRYLEFT)**

**sleep(0.2)**

**# clocks EN to latch command**

**def lcd\_strobe(self, data):**

**self.lcd\_device.write\_cmd(data | En | LCD\_BACKLIGHT)**

**sleep(.0005)**

**self.lcd\_device.write\_cmd(((data & ~En) | LCD\_BACKLIGHT))**

**sleep(.0001)**

**def lcd\_write\_four\_bits(self, data):**

**self.lcd\_device.write\_cmd(data | LCD\_BACKLIGHT)**

**self.lcd\_strobe(data)**

**# write a command to lcd**

**def lcd\_write(self, cmd, mode=0):**

**self.lcd\_write\_four\_bits(mode | (cmd & 0xF0))**

**self.lcd\_write\_four\_bits(mode | ((cmd << 4) & 0xF0))**

**# write a character to lcd (or character rom) 0x09: backlight | RS=DR<**

**# works!**

**def lcd\_write\_char(self, charvalue, mode=1):**

**self.lcd\_write\_four\_bits(mode | (charvalue & 0xF0))**

**self.lcd\_write\_four\_bits(mode | ((charvalue << 4) & 0xF0))**

**# put string function with optional char positioning**

**def lcd\_display\_string(self, string, line=1, pos=0):**

**if line == 1:**

**pos\_new = pos**

**elif line == 2:**

**pos\_new = 0x40 + pos**

**elif line == 3:**

**pos\_new = 0x14 + pos**

**elif line == 4:**

**pos\_new = 0x54 + pos**

**self.lcd\_write(0x80 + pos\_new)**

**for char in string:**

**self.lcd\_write(ord(char), Rs)**

**# clear lcd and set to home**

**def lcd\_clear(self):**

**self.lcd\_write(LCD\_CLEARDISPLAY)**

**self.lcd\_write(LCD\_RETURNHOME)**

**# define backlight on/off (lcd.backlight(1); off= lcd.backlight(0)**

**def backlight(self, state): # for state, 1 = on, 0 = off**

**if state == 1:**

**self.lcd\_device.write\_cmd(LCD\_BACKLIGHT)**

**elif state == 0:**

**self.lcd\_device.write\_cmd(LCD\_NOBACKLIGHT)**

**# add custom characters (0 - 7)**

**def lcd\_load\_custom\_chars(self, fontdata):**

**self.lcd\_write(0x40);**

**for char in fontdata:**

**for line in char:**

**self.lcd\_write\_char(line)**

There are a couple things you may need to change in the code above, depending on your set up. On line 19 there is a function that defines the port for the I2C bus (I2CBUS = 0). Older Raspberry Pi’s used port 0, but newer models use port 1. So depending on which RPi model you have, you might need to change this from 0 to 1.

Next, put the I2C address of your LCD in line 22 of the library code. For example, my I2C address is 27, so I’ll change line 22 to ADDRESS = 0x27.

### WRITE TO THE DISPLAY

The following is a bare minimum “Hello World!” program to demonstrate how to initialize the LCD:

**import I2C\_LCD\_driver**

**from time import \***

**mylcd = I2C\_LCD\_driver.lcd()**

**mylcd.lcd\_display\_string("Hello World!", 1)**

### POSITION THE TEXT

The function **mylcd.lcd\_display\_string()** prints text to the screen and also lets you chose where to position it. The function is used as **mylcd.lcd\_display\_string("TEXT TO PRINT", ROW, COLUMN).** For example, the following code prints “Hello World!” to row 2, column 3:

**import I2C\_LCD\_driver**

**from time import \***

**mylcd = I2C\_LCD\_driver.lcd()**

**mylcd.lcd\_display\_string("Hello World!", 2, 3)**

On a 16×2 LCD, the rows are numbered 1 – 2, while the columns are numbered 0 – 15. So to print “Hello World!” at the first column of the top row, you would use **mylcd.lcd\_display\_string("Hello World!", 1, 0).**

### CLEAR THE SCREEN

The function mylcd.lcd\_clear() clears the screen:

**import I2C\_LCD\_driver**

**from time import \***

**mylcd = I2C\_LCD\_driver.lcd()**

**mylcd.lcd\_display\_string("This is how you", 1)**

**sleep(1)**

**mylcd.lcd\_clear()**

**mylcd.lcd\_display\_string("clear the screen", 1)**

**sleep(1)**

**mylcd.lcd\_clear()**

### BLINKING TEXT

We can use a simple while loop with the mylcd.lcd\_display\_string() and mylcd.lcd\_clear() functions to create a continuous blinking text effect:

**import time**

**import I2C\_LCD\_driver**

**mylcd = I2C\_LCD\_driver.lcd()**

**while True:**

**mylcd.lcd\_display\_string(u"Hello world!")**

**time.sleep(1)**

**mylcd.lcd\_clear()**

**time.sleep(1)**

You can use the time.sleep() function on line 7 to change the time (in seconds) the text stays on. The time the text stays off can be changed in the time.sleep() function on line 9. To end the program, press Ctrl-C.

### PRINT THE DATE AND TIME

The following program prints the current date and time to the LCD:

**import I2C\_LCD\_driver**

**import time**

**mylcd = I2C\_LCD\_driver.lcd()**

**while True:**

**mylcd.lcd\_display\_string("Time: %s" %time.strftime("%H:%M:%S"), 1)**

**mylcd.lcd\_display\_string("Date: %s" %time.strftime("%m/%d/%Y"), 2)**

### PRINT YOUR IP ADDRESS

This code prints the IP address of your ethernet connection (eth0). To print the IP of your WiFi connection, change eth0 to wlan0 in line 18:

**import I2C\_LCD\_driver**

**import socket**

**import fcntl**

**import struct**

**mylcd = I2C\_LCD\_driver.lcd()**

**def get\_ip\_address(ifname):**

**s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)**

**return socket.inet\_ntoa(fcntl.ioctl(**

**s.fileno(),**

**0x8915,**

**struct.pack('256s', ifname[:15])**

**)[20:24])**

**mylcd.lcd\_display\_string("IP Address:", 1)**

**mylcd.lcd\_display\_string(get\_ip\_address('eth0'), 2)**

### SCROLL TEXT RIGHT TO LEFT CONTINUOUSLY

This program will scroll a text string from the right side of the LCD to the left side and loop continuously:

**import I2C\_LCD\_driver**

**from time import \***

**mylcd = I2C\_LCD\_driver.lcd()**

**str\_pad = " " \* 16**

**my\_long\_string = "This is a string that needs to scroll"**

**my\_long\_string = str\_pad + my\_long\_string**

**while True:**

**for i in range (0, len(my\_long\_string)):**

**lcd\_text = my\_long\_string[i:(i+16)]**

**mylcd.lcd\_display\_string(lcd\_text,1)**

**sleep(0.4)**

**mylcd.lcd\_display\_string(str\_pad,1)**

### SCROLL TEXT RIGHT TO LEFT ONCE

The following code slides text onto the screen from right to left once, then stops and leaves a cleared screen.

**import I2C\_LCD\_driver**

**from time import \***

**mylcd = I2C\_LCD\_driver.lcd()**

**str\_pad = " " \* 16**

**my\_long\_string = "This is a string that needs to scroll"**

**my\_long\_string = str\_pad + my\_long\_string**

**for i in range (0, len(my\_long\_string)):**

**lcd\_text = my\_long\_string[i:(i+16)]**

**mylcd.lcd\_display\_string(lcd\_text,1)**

**sleep(0.4)**

**mylcd.lcd\_display\_string(str\_pad,1)**

### SCROLL TEXT LEFT TO RIGHT ONCE

This program slides text onto the screen from left to right once, then stops and leaves the first 16 characters of the text string on the screen.

**import I2C\_LCD\_driver**

**from time import \***

**mylcd = I2C\_LCD\_driver.lcd()**

**padding = " " \* 16**

**my\_long\_string = "This is a string that needs to scroll"**

**padded\_string = my\_long\_string + padding**

**for i in range (0, len(my\_long\_string)):**

**lcd\_text = padded\_string[((len(my\_long\_string)-1)-i):-i]**

**mylcd.lcd\_display\_string(lcd\_text,1)**

**sleep(0.4)**

**mylcd.lcd\_display\_string(padding[(15+i):i], 1)**

### CUSTOM CHARACTERS

You can create any pattern you want and print it to the display as a custom character. Each character is an array of 5 x 8 pixels. Up to 8 custom characters can be defined and stored in the LCD’s memory. This custom character generator will help you create the bit array needed to define the characters in the LCD memory.

### PRINTING A SINGLE CUSTOM CHARACTER

The following code generates a “<” character:

**import I2C\_LCD\_driver**

**from time import \***

**mylcd = I2C\_LCD\_driver.lcd()**

**fontdata1 = [**

**[ 0b00010,**

**0b00100,**

**0b01000,**

**0b10000,**

**0b01000,**

**0b00100,**

**0b00010,**

**0b00000 ],**

**]**

**mylcd.lcd\_load\_custom\_chars(fontdata1)**

**mylcd.lcd\_write(0x80)**

**mylcd.lcd\_write\_char(0)**

### PRINTING MULTIPLE CUSTOM CHARACTERS

This program prints a large right pointing arrow (→) to the screen:

**import I2C\_LCD\_driver**

**from time import \***

**mylcd = I2C\_LCD\_driver.lcd()**

**fontdata1 = [**

**# char(0) - Upper-left character**

**[ 0b00000,**

**0b00000,**

**0b00000,**

**0b00000,**

**0b00000,**

**0b00000,**

**0b11111,**

**0b11111 ],**

**# char(1) - Upper-middle character**

**[ 0b00000,**

**0b00000,**

**0b00100,**

**0b00110,**

**0b00111,**

**0b00111,**

**0b11111,**

**0b11111 ],**

**# char(2) - Upper-right character**

**[ 0b00000,**

**0b00000,**

**0b00000,**

**0b00000,**

**0b00000,**

**0b00000,**

**0b10000,**

**0b11000 ],**

**# char(3) - Lower-left character**

**[ 0b11111,**

**0b11111,**

**0b00000,**

**0b00000,**

**0b00000,**

**0b00000,**

**0b00000,**

**0b00000 ],**

**# char(4) - Lower-middle character**

**[ 0b11111,**

**0b11111,**

**0b00111,**

**0b00111,**

**0b00110,**

**0b00100,**

**0b00000,**

**0b00000 ],**

**# char(5) - Lower-right character**

**[ 0b11000,**

**0b10000,**

**0b00000,**

**0b00000,**

**0b00000,**

**0b00000,**

**0b00000,**

**0b00000 ],**

**]**

**mylcd.lcd\_load\_custom\_chars(fontdata1)**

**mylcd.lcd\_write(0x80)**

**mylcd.lcd\_write\_char(0)**

**mylcd.lcd\_write\_char(1)**

**mylcd.lcd\_write\_char(2)**

**mylcd.lcd\_write(0xC0)**

**mylcd.lcd\_write\_char(3)**

**mylcd.lcd\_write\_char(4)**

**mylcd.lcd\_write\_char(5)**

### PRINT DATA FROM A SENSOR

The code below will display data from a DHT11 temperature and humidity sensor. Follow this tutorial for instructions on how to set up the DHT11 on the Raspberry Pi. The DHT11 signal pin is connected to BCM pin 4 (physical pin 7 of the RPi).

Temperature is displayed on line 1, and humidity is displayed on line 2:

**import RPi.GPIO as GPIO**

**import dht11**

**import I2C\_LCD\_driver**

**from time import \***

**mylcd = I2C\_LCD\_driver.lcd()**

**GPIO.setwarnings(False)**

**GPIO.setmode(GPIO.BCM)**

**GPIO.cleanup()**

**while True:**

**instance = dht11.DHT11(pin = 4)**

**result = instance.read()**

**# Uncomment for Fahrenheit:**

**# result.temperature = (result.temperature \* 1.8) + 32**

**if result.is\_valid():**

**mylcd.lcd\_display\_string("Temp: %d%s C" % (result.temperature, chr(223)), 1)**

**mylcd.lcd\_display\_string("Humidity: %d %%" % result.humidity, 2)**

For Fahrenheit, un-comment lines 18 and 19, and change the C to an F in line 22. You can also change the signal pin of the DHT11 input in line 15.

By inserting the variable from your sensor into the mylcd.lcd\_display\_string() function (line 22 in the code above) you can print the sensor data just like any other text string.

These programs are just basic examples of ways you can control text on your LCD. Try changing things around and combining the code to get some interesting effects. For example, you can make some fun animations by scrolling with custom characters. Don’t have enough screen space to output all of your sensor data? Just print and clear each reading for a couple seconds in a loop.

## Project 15

### Connect the PIR motion sensor

In this project, we are going to be using a passive infrared (PIR) motion sensor.

### About PIR motion sensors

In this project, we are going to be using a passive infrared (PIR) motion sensor. You have probably seen these before, as they are often used in burglar alarm systems.

If the temperature of an object or organism is above absolute zero (that’s −273.15°C!), it emits infrared radiation. Infrared wavelengths are not visible to the human eye, but they can be detected by the electronics inside a PIR sensor.

The sensor is regarded as passive because it doesn’t send out any signal in order to detect movement. It works by adjusting itself to the infrared signature of the room it’s in and then watching for changes. Any object moving through the room will disturb the infrared signature, and the PIR module detects this disturbance.



We don’t need to worry about the inner workings of the motion sensor. What we’re interested in are its three pins, which can be used to connect it to the Raspberry Pi.

### Connect a PIR motion sensor

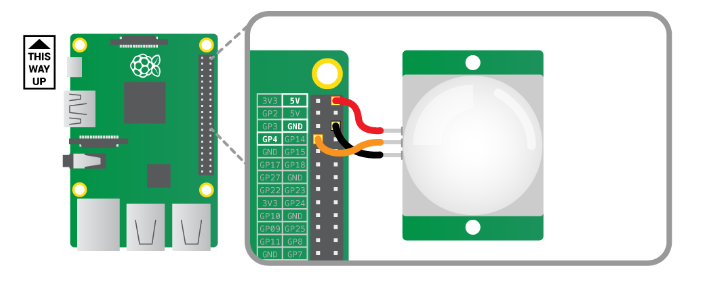
On some PIR sensors, the pins are clearly labelled on the circuit board. Others have the labels hidden beneath their cap. If this is the case for your sensor, gently pop its cap off so that you can distinguish the pins.

Using three female-to-female jumper cables, connect each of the PIR sensor’s pins to the appropriate pins on the Raspberry Pi:

Connect the PIR sensor’s pin labelled **VCC** to the **5V** pin on the Raspberry Pi. This provides power to the PIR sensor.

Connect the one labelled **GND** to a ground pin on the Pi (also labelled **GND**). This completes the circuit.

Connect the one labelled **OUT** to any numbered GPIO pin on the Pi. In this example, we have chosen **GPIO 4.** The **OUT** pin will output a voltage when the sensor detects motion. The voltage will then be received by the Raspberry Pi.



On the PIR module you should see two orange components with sockets that fit a Philips screwdriver (see above). These are called potentiometers: they allow you to adjust the sensitivity and the detection time of the sensor. You should begin by setting the sensitivity potentiometer to its maximum and the time potentiometer to its minimum. You can vary this later if you wish.

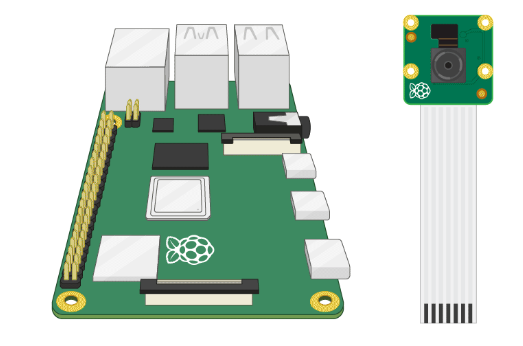


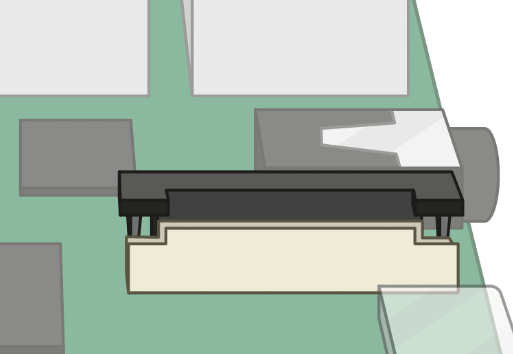
### Connect the camera

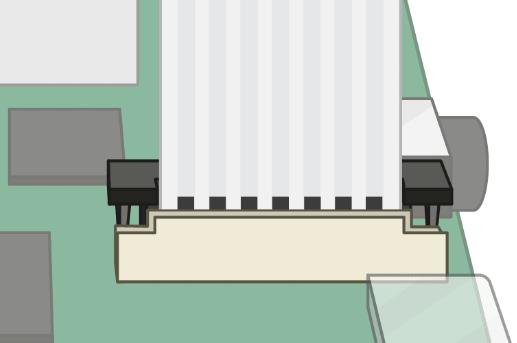
Before turning on your Raspberry Pi, connect the Camera Module to it.

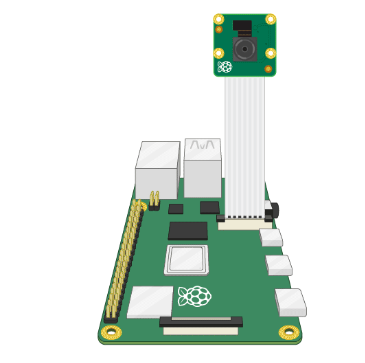
#### Connect a Raspberry Pi Camera Module

* Ensure your Raspberry Pi is turned off.
* Locate the Camera Module port
* Gently pull up on the edges of the port’s plastic clip
* Insert the Camera Module ribbon cable; make sure the cable is the right way round
* Push the plastic clip back into place

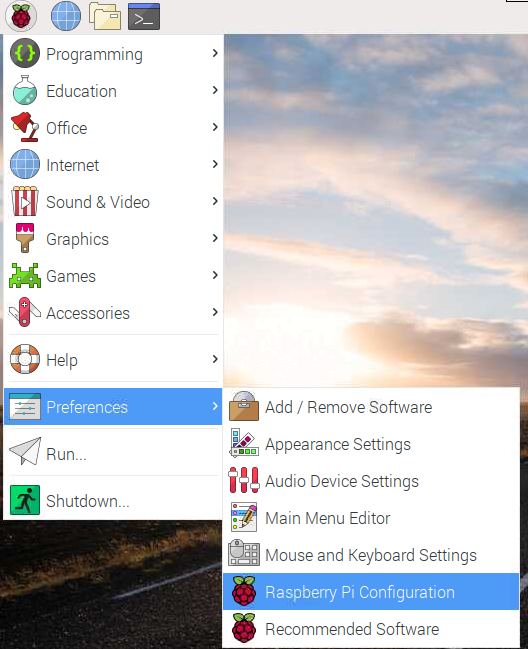




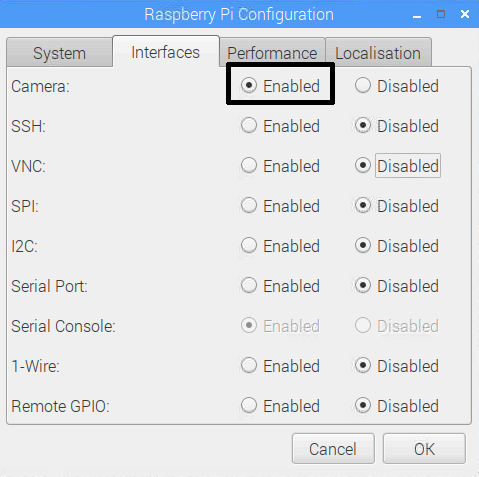




* Start up your Raspberry Pi.
* Go to the main menu and open the Raspberry Pi Configuration tool.



* Select the Interfaces tab and ensure that the camera is enabled:



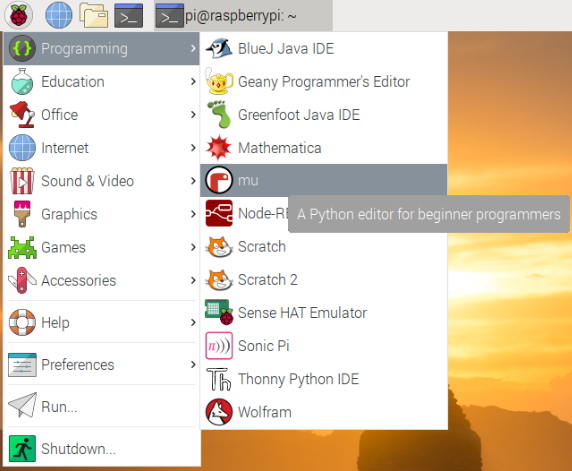
* Reboot your Raspberry Pi.

### Test the PIR motion sensor

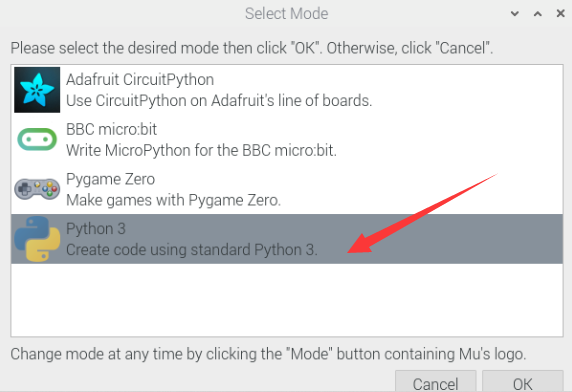
We’re going to write some code to print out “Motion detected!” when the PIR sensor detects movement.

#### How to open Mu

Go to the Programming menu and click on Mu.



Then choose the mode in which you want to use Mu. Choose Python 3 if you are creating a new Python script.



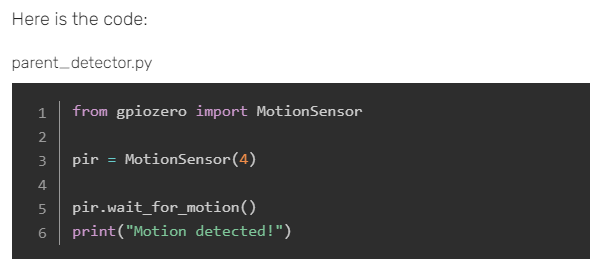
### Set up your PIR sensor on GPIO 4.

You will need to use the **gpiozero** library to create a **MotionSensor** that is connected to the correct GPIO pin.

Add some more code so that when the PIR sensor detects motion, “Motion detected!” is displayed on the screen.

Look at the documentation for GPIO Zero to find out how to use the **wait\_for\_motion()** method.

<https://gpiozero.readthedocs.io/en/stable/api_input.html#motion-sensor-d-sun-pir>



Save your code, and click on Run to run it. You should see the words Motion detected! appear on the screen when the motion sensor is triggered.

At the moment your code only detects movement once and then the program ends. Put your code inside an infinite loop so that Python will keep waiting for a signal from the motion sensor and will print Motion detected! every time the sensor is triggered. To exit your program, you can click on Stop.

### While True loop in Python

The purpose of a while loop is to repeat code over and over while a condition is True. This is why while loops are sometimes referred to as condition-controlled loops.

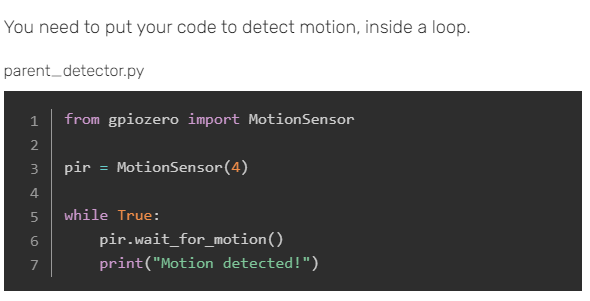
The example below is a while loop that will run forever - an infinite loop. The loop will run forever because the condition is always True.

**while True:**

**print("Hello world")**

Note: The while line states the loop condition. The print line of code below it is slightly further to the right. This is called indentation - the line is indented to show that it is inside the loop. Any code inside the loop will be repeated.

An infinite loop is useful in situations where you want to perform the same actions over and over again, for example checking the value of a sensor. An infinite loop like this will continue to run forever meaning that any lines of code written after the loop will never happen. This is known as **blocking** - whereby a program **blocks** the execution of any other code.



### Set up the camera preview

At the start of your program, import the **Picamera** class from the **picamera** library so that we can use code to control the Camera Module.

parent\_detector.py

**from gpiozero import MotionSensor**

**from picamera import PiCamera**

**pir = MotionSensor(4)**

**while True:**

**pir.wait\_for\_motion()**

**print("Motion detected!")**

Add a line of code to create a PiCamera object. Make sure this line of code is above the infinite loop.

parent\_detector.py

**from gpiozero import MotionSensor**

**from picamera import PiCamera**

**pir = MotionSensor(4)**

**camera = PiCamera()**

**while True:**

**pir.wait\_for\_motion()**

**print("Motion detected!")**

Add to your existing code so that it starts the camera preview when the sensor is activated and stop the preview when no motion is detected.

### Taking a photograph with PiCamera

You can use Python and the picamera module to take photos with the Raspberry Pi and its Camera Module.

Import the PiCamera class and create a camera object.

**from picamera import PiCamera**

**camera = PiCamera()**

To take a photo, you can use the **capture()** method. To do this, you need to tell Python where you would like the photo to be stored and what you would like it to be called. In this example below, the photo will be called **selfie.png** and will be stored in the /home/pi/ directory.

**from picamera import PiCamera**

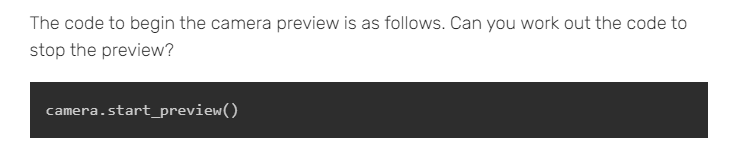
**camera = PiCamera()**

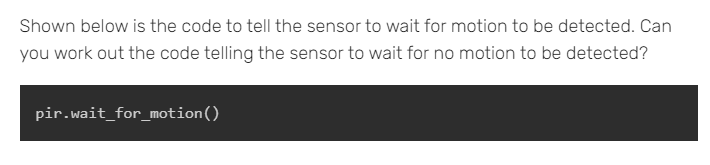
**camera.capture('home/pi/selfie.png')**

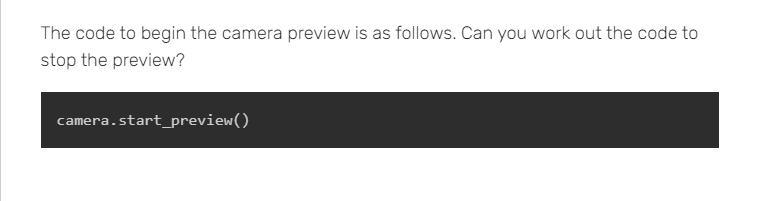
**camera.close()**

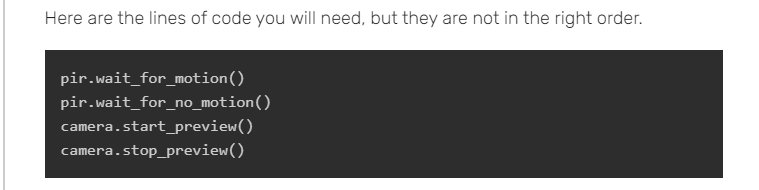
Run your code and then use either the File Manager or the Terminal to find the file selfie.png.

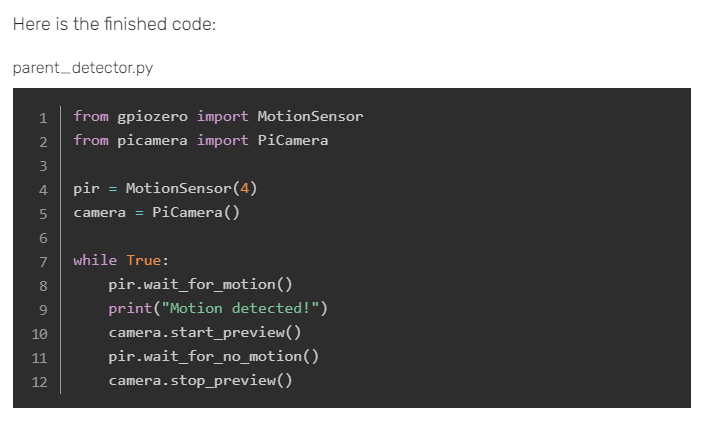
Make sure that the code you add to start the camera preview is indented so that Python knows it is inside the loop.











Save your code, and run it. Test that the camera preview appears when the motion sensor is activated, and stops when the motion sensor is no longer active.

### Record video in a file

Seeing the intruder on the screen in a camera preview while they are in the room isn’t much help to you. Instead, let’s record a video of the intruder which you can view later on when you get home.

Create a variable called filename inside your infinite loop to store the name of the video file.

parent\_detector.py

**from gpiozero import MotionSensor**

**from picamera import PiCamera**

**pir = MotionSensor(4)**

**camera = PiCamera()**

**filename = "intruder.h264"**

**while True:**

**pir.wait\_for\_motion()**

**print("Motion detected!")**

**camera.start\_preview()**

**pir.wait\_for\_no\_motion()**

**camera.stop\_preview()**

In case you are wondering**, .h264** is the video format.

Find the line of code that starts the camera preview, and replace it with a line of code that starts a video recording.

parent\_detector.py

**from gpiozero import MotionSensor**

**from picamera import PiCamera**

**pir = MotionSensor(4)**

**camera = PiCamera()**

**filename = "intruder.h264"**

**while True:**

**pir.wait\_for\_motion()**

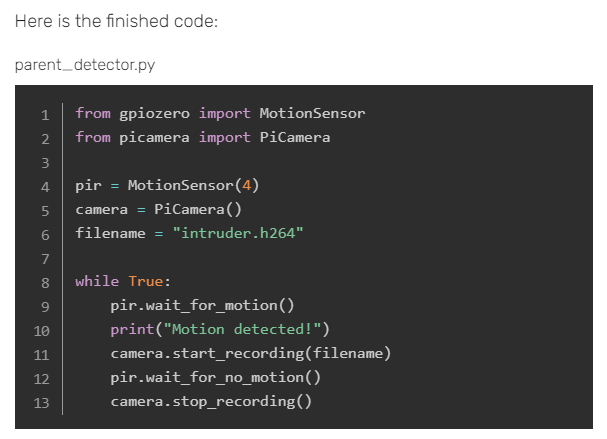
**print("Motion detected!")**

**camera.start\_recording(filename)**

**pir.wait\_for\_no\_motion()**

**camera.stop\_preview()**

Find the line of code stopping the camera preview, and replace it with a line of code stopping recording.



Save your program, and run it. Check that a file called intruder.h264 appears in the same folder as your parent-detector.py file.

Every time a new intruder triggers the motion sensor, the video file will be overwritten. If you have lots of pesky parents or siblings intruding into your room, you want to keep videos of all of them. Can you write some code to automatically find out the current date and time, and add it to the name of the video file? Then each video you record will have a different filename.

### Creating timestamps with Python

Sometimes you might want to get the current date and time as a string, for instance to give unique names to files or data. The datetime module is incredibly useful for creating such timestamps.

First you’ll need to import the **datetime** module, and specifically its **datetime** class.

**from datetime import datetime**

If you want to use the timestamp to name a file, you might want to use this for instance:

**filename = "{0:%Y}-{0:%m}-{0:%d}".format(datetime.now())**

The **{ }** is used as a placeholder within the string.

The **0** tells the print command to use the 0th object that is passed to it. In this case that object is now.

The **:%Y** code tells the print command to take the full year from the **datetime.now()** object.

Can you tell what the:**%m** and **:%d** codes stand for?

Have a look here for other codes you can use with the datetime module.

[**https://strftime.org/**](https://strftime.org/)

### Play video

When you return to your room you can play any videos created by your parent detector using the **OMXPlayer** software.

* Playing a video with OMXPlayer

Open a terminal window



Type the following command into the window, but replace filename with the name of the file you wish to play. Then press Enter.

**omxplayer filename**

For example, if your file was called **test.h264**, you would type this command and then press Enter.

**omxplayer test.h264**

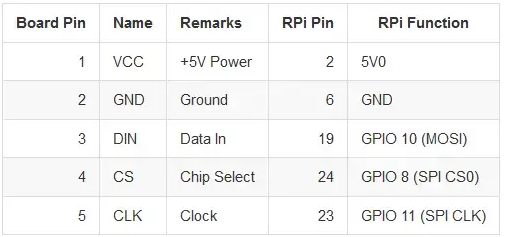
## Project 16

The assembly and a square LED matrix are shown since those modules are not supplied soldered together. A library for the MAX7219 IC for subsequent projects is also installed, which makes control with a Python script very easy.

### hardware parts are needed:

* MAX7219 Dot LED Matrix
* Raspberry Pi 4B
* 5V/3A Power supply with USB-C
* Breadboard
* 52Pi experiment Platform
* Jumper wires

### Hardware Connections：



### Preparation：

Download and install the libraries.

**git clone** [**https://github.com/rm-hull/max7219.git && cd max7219/**](https://github.com/rm-hull/max7219.git%20&&%20cd%20max7219/)

After the installation:

**sudo python setup.py install**

The test file can be started and the LED matrix should light up.

**sudo python examples/matrix\_demo.py**

If it flashes now everything worked, otherwise try to plug the LED module the other way around on the board.

more information about Luma-led-matirx: <https://luma-led-matrix.readthedocs.io/en/latest/>