# Raspberry Pi and Arduino – Introduction

As already announced at some point, there will be additional tutorials on how to use an Arduino on the side of the Raspberry Pi. The Arduino is a small microcontroller, which is very cheap to have but allows many additional features to the Pi. In combination, the two offer even more options than the Raspberry can offer alone.

first part is about the initial setup of an Arduino, so we’ll let it communicate with the Pi in the next steps.

First of all, I would like to introduce some common Arduino models:

**Andruino uno:**

The Arduino Uno is one of the most commonly used Arduinos. In addition to 14 digital I/O pins, it also has 6 analog input pins. The code is played via USB (type A), which also allows the power supply to take place. Alternatively, an external power source can be connected to the associated port.Overall, the Arduino Uno has 32KB of Flash memory that should be enough for your code (and embedded libraries), which is enough for most applications.There are many interesting attachable Shields, be it the USB\*, Ethernet\* or LCD\* (everyone should decide for themselves if they are worth the money because a Raspberry Pi 2 Model B\*with better equipment can be found for a similar price or even cheaper).

**Arduino mega:**

**Arduino model is the Arduino Mega, whose speciality are the many I / O pins: It has a total of 54 digital and 16 analogue input/output pins to offer. It also offers 256KB of flash memory, which is eight times as much as the Arduino Uno. Oher specifications are quite similar to those of the Uno, so the Mega can be powered via USB or externally and there are also various Shields\* for this model.**

**Android nano :**

**The Arduino Nano is one of the smallest models in the series and measures just 4.5cm x 1.8cm. Smaller is only the Arduino Mini\* (which, however, has no USB port).The Nano model has a USB port (type: mini), which serves for communication, as well as power supply. It also has 14 digital and 8 analogue I / O pins, which is enough for small applications. Depending on the model, it has 16KB (Atmega168) or 32KB (Atmega328) flash memory.In the tutorials, I mainly use this model because it is small and cheap.**

**Installation of arduino :**

After learning about the main parts of the Arduino UNO board, we are ready to learn how to set up the Arduino IDE. Once we learn this, we will be ready to upload our program on the Arduino board.

In this section, we will learn in easy steps, how to set up the Arduino IDE on our computer and prepare the board to receive the program via USB cable.

**Step 1** − First you must have your Arduino board (you can choose your favorite board) and a USB cable. In case you use Arduino UNO, Arduino Duemilanove, Nano, Arduino Mega 2560, or Diecimila, you will need a standard USB cable (A plug to B plug), the kind you would connect to a USB printer as shown in the following image.

In case you use Arduino Nano, you will need an A to Mini-B cable instead as shown in the following image.

**Step 2 − Download Arduino IDE Software.**

You can get different versions of Arduino IDE from the [Download page](https://www.arduino.cc/en/Main/Software) on the Arduino Official website. You must select your software, which is compatible with your operating system (Windows, IOS, or Linux). After your file download is complete, unzip the file.

**Step 3 − Power up your board.**

The Arduino Uno, Mega, Duemilanove and Arduino Nano automatically draw power from either, the USB connection to the computer or an external power supply. If you are using an Arduino Diecimila, you have to make sure that the board is configured to draw power from the USB connection. The power source is selected with a jumper, a small piece of plastic that fits onto two of the three pins between the USB and power jacks. Check that it is on the two pins closest to the USB port.

Connect the Arduino board to your computer using the USB cable. The green power LED (labeled PWR) should glow.

**Step 4 − Launch Arduino IDE.**

After your Arduino IDE software is downloaded, you need to unzip the folder. Inside the folder, you can find the application icon with an infinity label (application.exe). Double-click the icon to start the IDE.

**Step 5 − Open your first project.**

Once the software starts, you have two options −

* Create a new project.
* Open an existing project example.

To create a new project, select File → **New**.

To open an existing project example, select File → Example → Basics → Blink.

Here, we are selecting just one of the examples with the name **Blink**. It turns the LED on and off with some time delay. You can select any other example from the list.

**Step 6 − Select your Arduino board.**

To avoid any error while uploading your program to the board, you must select the correct Arduino board name, which matches with the board connected to your computer.

Go to Tools → Board and select your board.

Here, we have selected Arduino Uno board according to our tutorial, but you must select the name matching the board that you are using.

**Step 7 − Select your serial port.**

Select the serial device of the Arduino board. Go to **Tools → Serial Port** menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu, the entry that disappears should be of the Arduino board. Reconnect the board and select that serial port.

**Step 8 − Upload the program to your board.**

Before explaining how we can upload our program to the board, we must demonstrate the function of each symbol appearing in the Arduino IDE toolbar.

**A** − Used to check if there is any compilation error.

**B** − Used to upload a program to the Arduino board.

**C** − Shortcut used to create a new sketch.

**D** − Used to directly open one of the example sketch.

**E** − Used to save your sketch.

**F** − Serial monitor used to receive serial data from the board and send the serial data to the board.

Now, simply click the "Upload" button in the environment. Wait a few seconds; you will see the RX and TX LEDs on the board, flashing. If the upload is successful, the message "Done uploading" will appear in the status bar.

**Note** − If you have an Arduino Mini, NG, or other board, you need to press the reset button physically on the board, immediately before clicking the upload button on the Arduino Software.

**Downloading and Installing Raspberry Pi OS**

Once you have all the components you need, use the following steps to create the boot disk you will need to set up your Raspberry Pi. These steps should work on a  using a Windows, Mac or Linux-based PC (we tried this on Windows, but it should be the same on all three).

1.  **Insert a microSD card / reader**into your computer.

2.  **Download and install the**[**official Raspberry Pi Imager**](https://www.raspberrypi.org/downloads/)**.**Available for Windows, macOS or Linux, this app will both download and install the latest Raspberry Pi OS. There are other ways to do this, namely by downloading a Raspberry Pi OS image file and then using a third-party app to “burn it,” but the Imager makes it easier.

3.  **Click Choose OS.**

(Image credit: Tom's Hardware)

4. **Select Raspberry Pi OS (32-bit)**from the OS menu (there are other choices, but for most uses, 32-bit is the best).

(Image credit: Tom's Hardware)

 4. **Click Choose storage** and **pick the SD card**you’re using.

(Image credit: Tom's Hardware)

5. **Click the settings button** or hit CTRL + SHIFT + X to enter settings.

(Image credit: Tom's Hardware)

6. **Fill in settings fields**as follows and then **hit Save**. All of these fields are technically optional, but highly recommended so that can get your Raspberry Pi set up and online as soon as you boot it. If you don't set a username and password here, you'll have to go through a setup wizard that asks you to create them on first boot.

* **Set hostname**: the name of your Pi. It could be "raspberrypi" or anything you  like.
* **Enable SSH**: Allow SSH connections to the Pi. Recommended.
* **Use password authentication / public key:**method of logging in via SSH
* **Set username and password:** Pick the username and password you'll use for the Pi
* **Configure wireless LAN: s**et the SSID and password of Wi-FI network
* **Wireless LAN country:**If you're setting up Wi-Fi, you must choose this.
* **Set locale settings:**Configure keyboard layout and timezone (probably chosen correctly by default)

(Image credit: Tom's Hardware)

7. **Click Write.**The app will now take a few minutes to download the OS and write to your card.

(Image credit: Tom's Hardware)

**Booting Your Raspberry Pi for the First Time**

After you're done writing the Raspberry Pi OS to a microSD card, it's time for the moment of truth.

1**. Insert the microSD card**into the Raspberry Pi.

2. **Connect the Raspberry Pi**to a monitor, keyboard and mouse.

3. **Connect an Ethernet cable**if you plan to use wired Internet.

4.**Plug the Pi in** to power it on.

If you had used the Raspberry Pi Imager settings to create a username and password, you'll be able to go straight into the desktop environment, but if not, you will get a setup wizard.

**Using the Raspberry Pi First-Time Setup WIzard**

If you chose a username and password in Raspberry Pi Imager settings, before writing the microSD card, you will get the desktop on first boot. But, if you did not, you'll be prompted to create a username and password and enter all the network credentials by a setup wizard on first boot. If that happens, follow these steps to finish setting up your Raspberry Pi.

1. **Click Next**on the dialog box.

(Image credit: Tom's Hardware)

2. **Set your country and and language**and click Next. The default choices may already be the correct ones.

(Image credit: Tom's Hardware)

3. **Enter a username and password**you wish to use for your primary login. **Click Next**.

(Image credit: Tom's Hardware)

4. **Toggle Reduce the size of the desktop" to on**if the borders of the desktop are cut off. Otherwise, just **click Next**.

(Image credit: Tom's Hardware)

5. **Select the appropriate Wi-Fi network**on the screen after, provided that you are connecting via Wi-Fi. If you don't have Wi-Fi or are using Ethernet, you can skip this.

(Image credit: Tom's Hardware)

6. **Enter your Wi-Fi password**(unless you were using Ethernet and skipped).

(Image credit: Tom's Hardware)

7. **Click Next** when prompted to Update Software. This will only work when you are connected to the Internet, and it can take several minutes. If you are not connected to the Internet, click Skip.

(Image credit: Tom's Hardware)

8. **Click Restart.**

(Image credit: Tom's Hardware)

If you wish to change these settings later, you can find the region and password settings, along with many other options, by clicking on the Pi icon in the upper left corner of the screen and navigating to *Preferences -> Raspberry Pi Configuration*. You can configure Wi-Fi by clicking on the Wi-Fi / network icon on the taskbar.

**Controlling output :**

## A Bit About Python

Python [1] is an interpreted, high-level, general-purpose programming language that has been around since 1991. It is currently one of the most popular and fastest growing programming languages. The "Pi" in Raspberry Pi standards for "Python Interpreter," reflecting the fact that this is the recommended language on the platform.

A nice feature of Python is that, being an interpreter, you can type in and try commands interactively without needing to create a program. Being an interpreter there is no need to explicitly compile programs. They are compiled at run time into an intermediate bytecode which is executed by a virtual machine.

The example code in this blog post is written for Python 3 and should work on any Raspberry Pi model.

## RPi.GPIO

Raspberry-gpio-python [2] or RPi.GPIO, is a Python module to control the GPIO interface on the Raspberry Pi. It was developed by Ben Croston and released under an MIT free software license.

The project Wiki [3] has documentation including example programs. I'll cover some of the basics here.

The RPi.GPIO module is installed by default on recent versions of Raspbian Linux. To use the module from Python programs, first import it using:

**import RPi.GPIO as GPIO**

This way you can refer to all functions in the module using the shorter name "GPIO".

RPi.GPIO supports referring to GPIO pins using either the physical pin numbers on the GPIO connector or using the BCM channel names from the Broadcom SOC that the pins are connected to. For example, pin 24 is BCM channel GPIO8. To use physical board pin numbers, call:

**GPIO.setmode(GPIO.BOARD)**

and to use the BCM channel numbers, use:

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

Either method will work. The BOARD number scheme has the advantage that the library is aware of the Raspberry Pi model it is running on and will work correctly even if the Broadcom SOC channel names change in the future.

To set up a channel as an input or an output, call either:

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

or

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

Where channel is the channel number based on the numbering system you specified when you called setmode.

To read the value of an input channel, call:

**GPIO.input(channel)**

where channel is the channel number as used in setup. It will return a value of 0, GPIO.LOW, or False (all are equivalent) if it is low and 1, GPIO.HIGH, or True if it was at a high level.

To set the output state of a GPIO pin, call:

**GPIO.output(channel, state)**

where channel is the channel number and state is the desired output level: either 0, GPIO.LOW, or False for a low value or 1, GPIO.HIGH, or True for a high level.

When you are done with the library, it is good practice to free up any resources used and return all channels back to the safe default of being inputs. This is done by calling:

**GPIO.cleanup()**

Here is a very simple standalone example that toggles an output pin on and off for 200 milliseconds, ten times. It also reports the level on input pin 31. If you put the commands in a file and make it executable, you can directly run it as a program.

**#!/usr/bin/python3**

**import RPi.GPIO as GPIO  
import time**

**led = 18  
switch = 31**

**GPIO.setmode(GPIO.BOARD)  
GPIO.setup(led, GPIO.OUT)  
GPIO.setup(switch, GPIO.IN)**

**for i in range(10):  
    GPIO.output(led, GPIO.HIGH)  
    time.sleep(0.2)  
    GPIO.output(led, GPIO.LOW)  
    time.sleep(0.2)  
    print('Switch status = ', GPIO.input(switch))**

**GPIO.cleanup()**

If pin 18 is connected to an LED and pin 31 to a switch, as it is on my little GPIO learning board, we can see the LED flashing and the program will report the status of the switch.

There are a number of additional features related to input pins. We can enable the internal pullup or pulldown resistors on the Raspberry Pi by passing an additional parameter when we call setup, e.g.

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

to enable a pull up or pull down, respectively.

We saw how we could get an input channel's level. To wait for a given level we could call GPIO.input() in a loop. The library also provides a function wait\_for\_edge that will block the program until a specified level is present, e.g.

**GPIO.wait\_for\_edge(channel, GPIO.RISING)**

The level can be GPIO.RISING, GPIO.FALLING, or GPIO.BOTH.

If we don't want to block waiting for an event (which is often the case in a real application with a user interface), there is support for events and callbacks. We can specify an event we are interested in:

**GPIO.add\_event\_detect(channel, GPIO.RISING)**

This will not block. At a later time we can inquire if the event has occurred by calling:

**GPIO.event\_detected(channel)**

which will return a boolean status.

If we want a callback function to be called when the event occurs, we can define a Python function, e.g.

**def my\_callback(channel):  
    print('This is a edge event callback function!')  
    print('Edge detected on channel %s'%channel)**

And then specify the callback function when we call add\_event\_detect:

**GPIO.add\_event\_detect(channel, GPIO.RISING, callback=my\_callback)**

When the event occurs, the callback will be run in another thread independent of what the program's main thread is doing.

A final feature is PWM or Pulse Width Modulation. This refers to toggling an output pin at different rates and duty cycles. With an LED, for example, this can be used to make it flash or to adjust the perceived brightness. The library has support for this, although it is done in software and does not currently take advantage of the Raspberry Pi's hardware PWM capability.

To use it, you set the toggle frequency of the channel (in Hertz), keeping the returned handle:

**p = GPIO.PWM(channel, frequency)**

You start PWM mode by calling start with a duty cycle from 0 to 100 percent:

**p.start(dc)**

At any time you can change the frequency or duty cycle:

**p.ChangeFrequency(freq)  
p.ChangeDutyCycle(dc)**

And finally, when done, to stop PWM we can call:

**p.stop()**

Our final example with RPi.GPIO uses PWM to vary the brightness of each of the three LEDs on the GPIO learning board in turn. It also uses a callback so that the program immediately exits at any time when the pushbutton is pressed.

**#!/usr/bin/python3**

**import sys  
import time  
import RPi.GPIO as GPIO**

**# Callback called when switch is pressed.  
def switch\_callback(channel):  
    print('Switch pressed, exiting.')  
    GPIO.cleanup()  
    sys.exit(0)**

**led1 = 18  # Red  
led2 = 22  # Green  
led3 = 29  # Yellow  
switch = 31**

**GPIO.setmode(GPIO.BOARD)  
GPIO.setup(led1, GPIO.OUT)  
GPIO.setup(led2, GPIO.OUT)  
GPIO.setup(led3, GPIO.OUT)  
GPIO.setup(switch, GPIO.IN)**

**GPIO.add\_event\_detect(switch, GPIO.FALLING, callback=switch\_callback)**

**while True:  
    for pin in [led1, led2, led3]:  
        p = GPIO.PWM(pin, 50)  
        p.start(0)  
    for dc in range(0, 101, 5):  
        p.ChangeDutyCycle(dc)  
        time.sleep(0.05)  
    for dc in range(100, -1, -5):  
        p.ChangeDutyCycle(dc)  
        time.sleep(0.05)**

There are some additional functions provided by the Rpi.GPIO library that you can read about in the Wiki documentation. Overall it is quite simple and easy to use.

## Gpiozero

A newer GPIO library for the Raspberry Pi is gpiozero [4]. Created by Ben Nuttall of the Raspberry Pi Foundation and other contributors it is released under an MIT-type free software license.

While newer than Rpi.GPIO, it is now generally recommended to use it for new programming. It can have a longer learning because it offers more features that Rpi.GPIO, but the resulting code is usually very clean and readable.

We'll look at a few simple examples of how to use it.

Documentation is excellent, and presents many "recipes" showing how to control various devices from LEDs to switches to motion sensors, servers, and robots.

Gpiozero should be installed by default on Raspian Linux unless you installed the "lite" version. If needed, it can be installed using the command:

**sudo apt install python3-gpiozero**

Gpiozero provides may different modules or "interfaces". You typically import the ones you use by name so you can refer to them with a short name. For example:

**from gpiozero import LED**

to allow using the gpiozero.LED module and refer to it as "LED". You can import multiple modules with one import statement, e.g.

**from gpiozero import LED, Button**

The GPiozero library uses Broadcom (BCM) pin numbering rather than physical pin numbers. That should not normally be a problem. It does define names based on other naming conventions like physical pins that can be used and will be converted to the BCM names. The following examples will all work for an LED that is on GPIO 17 on physical pin 11:

**led = LED(17)  
led = LED("GPIO17")  
led = LED("BCM17")  
led = LED("BOARD11")  
led = LED("WPI0")  
led = LED("J8:11")**

A handy command line tool called "pinout" is part of the library and will graphically show you the GPIO pins for the board it is running on (or any board revision that you specify):

The library is oriented around device and sensor types rather than inputs and outputs. For driving an output connected to an LED, for example, you use the LED module. You create an instance by passing the GPIO name. You can then call various methods like on() and off(). Here is a simple example that flashes and LED:

**#!/usr/bin/python3**

**from gpiozero import LED  
from time import sleep**

**led = LED(24)**

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

The above loop could also be done by simply calling:

**red.blink()**

Which blinks an LED, defaulting to a rate of one second on and one second off.

Inputs pins are often connected to buttons, and in this case you can use the Gpiozero Button module, as in this example:

**#!/usr/bin/python3**

**from gpiozero import Button  
from time import sleep**

**button = Button(6)**

**while True:  
    if button.is\_pressed:  
        print("Button is pressed")  
    else:  
        print("Button is not pressed")**

**sleep(1)**

We can easily connect a switch so that when a pressed or released event occurs, we drive an LED high or low. This is event drived, so once set up, this works without any polling or further processing:

**#!/usr/bin/python3**

**from gpiozero import LED, Button  
from signal import pause**

**led = LED(24)  
button = Button(6)**

**button.when\_pressed = led.on  
button.when\_released = led.off**

**pause()**

The call to pause() ensures that our Python program does not exit immediately, but rather keeps running until the user interrupts it with Control-C or similar.

Gpiozero has support for many devices - you can explore the documentation and try writing programs of your own. You can also extend it for new types of devices and sensors.

## Conclusions

We've looked at a couple of popular modules for GPIO programming from Python. These are a great way to learn, especially for students and hobbyists. For commercial embedded applications you'll likely want to use a compiled programming language that is lighter weight and closer to the hardware so you can optimize performance and minimize resource usage. In Part 5 we'll look at how to program GPIO from C and C++.

**Reading inputs from pins:**

Input from a Sensor via GPIO - Raspberry Pi tutorial

Hello and welcome to part 7 of the Raspberry Pi tutorial series. In this tutorial, we're going to introduce a new sensor, the HC-SR04 ultrasonic distance sensor, along with handling GPIO input.

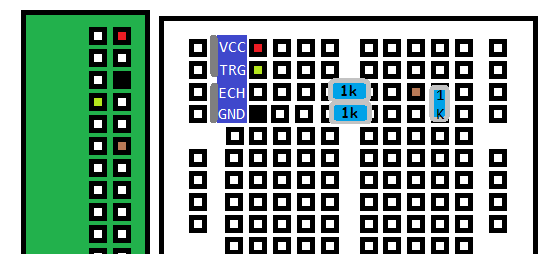
The HC-SR04 distance sensor measures distance based on emitting a sound burst, and timing how long it takes to receive the echo back. Using the known constant that is the speed of sound, we can mathmematically determine the distance of any object in front of this sensor by simply measuring how much time passed while the sound waves were emitted, hit the object in front of the sensor, bounced back, and came back to the sensor.

Things you will need for this tutorial besides your Raspberry Pi:

* A breadboard
* 4 Male-to-Female jumper wire cables
* 1 x 1K Ohm and 1 x 2K Ohm resistor, OR 3x 1K Ohm resitors...or any way to build 1k resistance then 2k resistance
* 1 x HC-SR04 ultrasonic distance sensor

The distance sensor comes with 4 pins: power, trigger, echo, and ground. The power will be hooked up to the Raspberry Pi's 5V out pin, trigger will be assigned to a GPIO pin as output, echo will be assigned to a GPIO pin as input, and ground will go to a ground pin on the Pi.

The following diagram shows the setup:



Like before, the boxes represent pins on the Pi, and plugs on the breadboard. The blue portion is the actual HC-SR04 sensor plugged in, and the colored-in boxes represent the M-F jumper wires by color plugged into these spots. The blue blobs with numbers on them are the resitors. If you have a 2K resistor to go before ground, feel free to use that, I just didn't have any so I used 2 1K in series.

Now you can boot up the Raspberry Pi, create a new Python file, and let's setup the code:

import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BCM)

Same imports and initial mode as in the previous tutorial.

TRIG = 4

ECHO = 18

GPIO.setup(TRIG,GPIO.OUT)

GPIO.setup(ECHO,GPIO.IN)

Here, we're going to define TRIG and ECHO as the Broadcom pin #'s that we intend to use for that part of the sensor. We do this because we need to reference both pins multiple times. If we wind up wanting to change the pin placement in the future, we just need to modify a single variable rather than a bunch and risk making a mistake!

GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)

while GPIO.input(ECHO) == False:

start = time.time()

Above, we go ahead and issue a signal out.

while GPIO.input(ECHO) == True:

end = time.time()

sig\_time = end-start

Now, when we finally do get an input time, we can subtract the end time from the start time and calcuate distance:

#CM:

distance = sig\_time / 0.000058

#inches:

#distance = sig\_time / 0.000148

print('Distance: {} centimeters'.format(distance))

GPIO.cleanup()

I chose to use metric, but I provided the imperial measurement just in case you wanted that instead. We print the distance, and then cleanup the pins.

Your output should be something like:

Distance: 8.89959006474 centimeters

Do note that, after about 20 degrees of angle, your distance might be very skewed since the initial bounce might miss and it might take the burst several bounces off things to come back to the sensor.

Full code for this tutorial:

import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BCM)

TRIG = 4

ECHO = 18

GPIO.setup(TRIG,GPIO.OUT)

GPIO.setup(ECHO,GPIO.IN)

GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)

while GPIO.input(ECHO) == False:

start = time.time()

while GPIO.input(ECHO) == True:

end = time.time()

sig\_time = end-start

#CM:

distance = sig\_time / 0.000058

#inches:

#distance = sig\_time / 0.000148

print('Distance: {} centimeters'.format(distance))

GPIO.cleanup()

**python programming with raspberry pi with focus on interfacing external gadgets:**

## Web Application – Remote Control Web UI:

This is a very simple web page with a visual indicator for the device and a button to toggle the on/off state of LED.

Behind the scenes, we have the PubNub Javascript API that performs two operations upon receiving certain events.

1. Sends a request message to toggle the state of the device.
2. Receives response with the current state of the device.

#### Button Click Event

When the TOGGLE button is clicked, the webpage sends a Toggle request message to the device via ‘gpio-raspberry-control’  channel.

[code language=”javascript”]  
$(‘#toggle’).click(function(e){

pubmsg = { &quot;req&quot; : &quot;toggle&quot; };

pubnub.publish(

{  
channel : ‘gpio-raspberry-control’ ,  
message : pubmsg

}

);

});  
[/code]

#### PubNub Channel Subscribe Callback Event

On receiving the toggle request, the Raspberry Pi toggles the state of the LED and sends a response back to the web page with the current state. Web page updates the visual indicator for the LED based on the received state information.

[code language=”javascript”]  
pubnub.subscribe({  
channel: ‘gpio-raspberry-control’,  
message: function(m){

console.log(m)

if(‘resp’ in m) {

if(‘on’ == m[‘resp’]){

$(‘#led’).removeClass(‘dim’);  
$(‘#led’).addClass(‘glow’);

} else {

$(‘#led’).removeClass(‘glow’);  
$(‘#led’).addClass(‘dim’);

}

}

}

});  
[/code]

## Raspberry Pi and LED

Here is the schematic for the Raspberry Pi connections to be used in this application.

We are going to use the [Raspberry Pi GPIO Python library](https://pypi.python.org/pypi/RPi.GPIO) to send the control messages to Raspberry Pi GPIO ports. This library works well with the python environment available by default with Raspbian OS.

The python code for driving the LED is executed as part of the PubNub callback on ‘**gpio-raspberry-control**’  channel

[code language=”python”]  
glow = False  
#PubNub Channel Subscribe Callback  
def gpioCallback(msg,channel):

global glow

respstring = ”  
command = msg

print &quot;Command is : &quot; + str(command)

if(‘req’ in command):  
if(command[‘req’] == ‘toggle’):

if(glow):  
glow = False;  
respstring = ‘off’  
else:  
glow = True  
respstring = ‘on’

GPIO.output(16, glow)

respmsg = {&amp;quot;resp&amp;quot; : respstring }  
pubnub.publish(pubnubChannelName, respmsg)  
[/code]

When a toggle request is received, the application checks the current state of the GPIO driving pin of the LED, toggles its state and then sends the new state back to the web application as a response message.