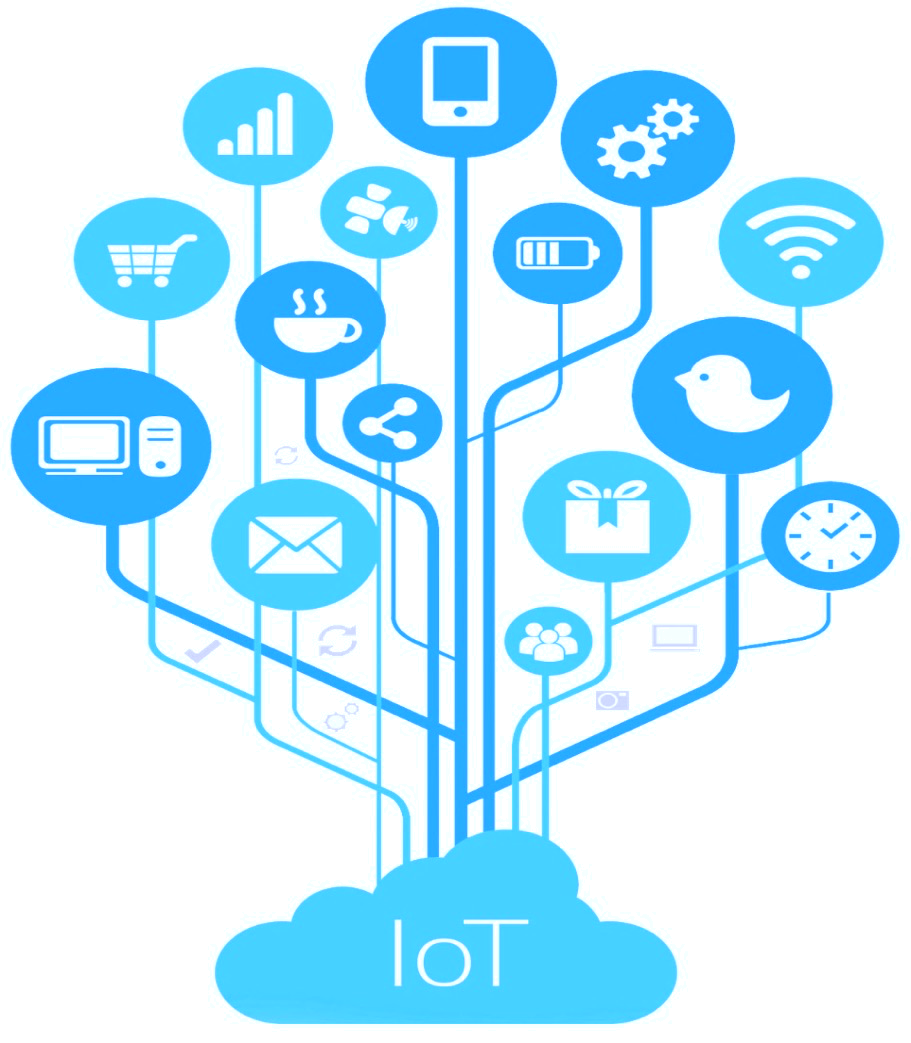
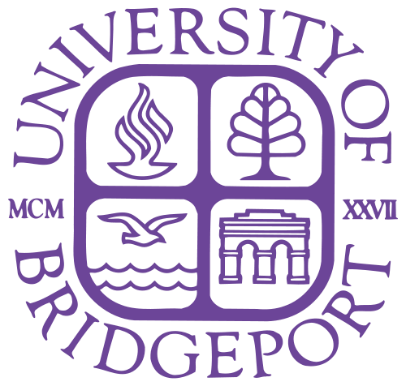
**THE INTERNET OF THINGS**



****

**University of Bridgeport**

By Abdel A Shakour

Soumil Nitin Shah (Graduate Assistant)

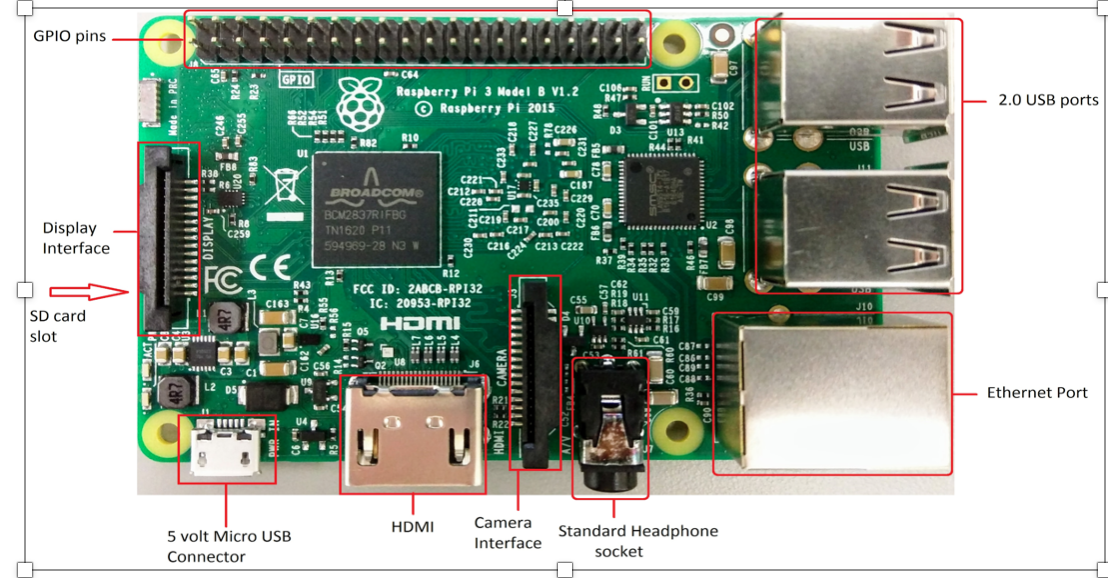
Lab I

**INTRODUCTION Lab 1A**

This manual is based on Raspberry Pi and will start from very basic, assuming that you just came to the world of processors and will gradually cover complex labs that are required to understand the application of Internet of Things and Controls. We will learn to use it as a development board. It is impossible to cover everything that the Raspberry Pi is capable of, because the boundaries of the RPi is defined by your imagination. Hence, this manual serves as a guide to get you familiar with the RPi’s functionality by executing several labs that will give you the skills you need to develop your own projects.

**What is a Raspberry Pi?**

The Raspberry Pi is a credit-card sized single-board computer (SBC) and development board, which is heavily used for the prototyping of IoT based products. However, one key aspect that makes the Raspberry Pi so brilliant for schools is its ability to execute “Python” coded programs. This allows us along with the General-Purpose Input Output (GPIO) pins to create programs that can control anything from a single LED to opening your garage door.



**The Raspberry Pi 3 is the third-generation Raspberry Pi. The following are its on board components**

● ARM CPU/GPU -- This is a Broadcom BCM 3287 System on a Chip (SoC) that's made up of an ARM central processing unit (CPU) and a Videocore 4 graphics processing unit (GPU). The CPU handles all the computations that make a computer work (taking input, doing calculations and producing output), and the GPU handles graphics output.

● GPIO -- These are exposed general-purpose input/output connection points that will allow the real hardware hobbyists the opportunity to tinker.

● RCA -- An RCA jack allows connection of analog TVs and other similar output devices.

● Audio out -- This is a standard 3.55-millimeter jack for connection of audio output devices such as headphones or speakers. There is no audio in.

● Camera Connector -- 15-pin MIPI Camera Serial Interface (CSI-2)

● Display Connector --Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane.

● USB -- This is a common connection port for peripheral devices of all types (including your mouse and keyboard). There are 4 USB2.0 Connectors.

● HDMI -- This connector allows you to hook up a high-definition television or other compatible device using an HDMI cable.

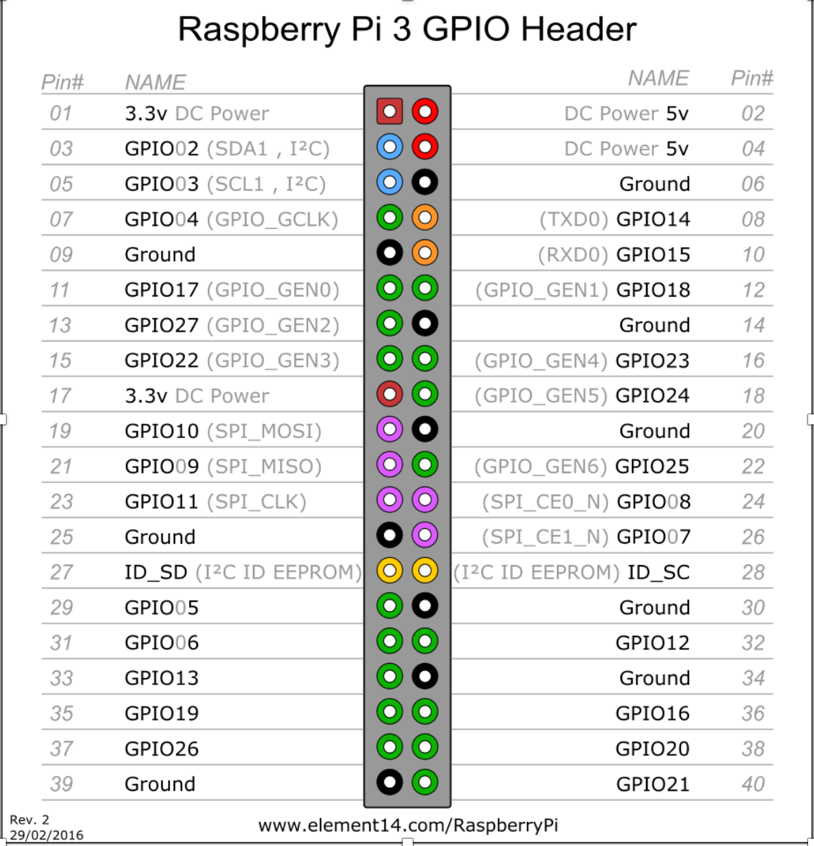
● Power -- This is a 5v Micro USB power connector into which you can plug your compatible power supply.

● SD card slot -- This is a full-sized SD card slot. An SD card with an operating system (OS) installed is required for booting the device. They are available for purchase from the manufacturers, but you can also download an OS and save it to the card yourself if you have a Linux machine and the wherewithal.

● Ethernet -- This connector allows for wired network access

**What are the GPIO Pins?**

Visible in the two below pictures, the GPIO pins are located on the Raspberry Pi in the top right corner. These pins are a physical interface between the Pi and the outside world. At the simplest level, you can think of them as switches that you can turn on or off (input) or that the Pi can turn on or off (output). Of the 40 pins, 26 are GPIO pins and the others are power or ground pins, plus two ID EEPROM pins.



**What are they for? What can I do with them?**

You can program the pins to interact in amazing ways with the real world. Inputs don't have to come from a physical switch; it could be input from a sensor or a signal from another computer or device, for example. The output can also do anything, from turning on an LED to sending a signal or data to another device. If the Raspberry Pi is on a network, you can control devices that are attached to it from anywhere and those devices can send data back. Connectivity and control of physical devices over the internet is a powerful and exciting thing, and the Raspberry Pi is ideal for this.

**Pin Numbering System**

Raspberry Pi pins are numbered in two different ways - Physical numbering and Broadcom numbering (BCM). In the first case, the pins are numbered sequentially from one to 40. In the Figure 2, this is shown as Pin#. And the image represents what is seen when the Pi is held with the USB ports facing downwards. That is, Pin 2 is the pin in the corner. The Broadcom numbering system is the default option for the SoC (System-on-Chip). This is also known as GPIO numbering and is shown as Name in the Figure 2.

As you have probably guessed already, all the pins are not programmable. There are eight ground pins and two +5V pins and three +3.3V pins, which are not programmable. There are other dedicated pins too. Most of the pins have alternative functions, as shown in the figure. A majority of the pins are directly connected to the SoC; so while connecting circuits or components, one should be careful to avoid wrong wiring and short circuits. It is always good to have a descriptive pinout diagram printed out for quick reference.

# **Programming the Pins**

The Python package used for Raspberry Pi GPIO programming is RPi.GPIO. It is already installed in Raspbian, the default operating system for Pi. If you are using any other operating system, the package can be installed by using the following command:

# **The GPIO Programming Workflow**

Let us discuss the steps involved in writing a GPIO Python script, in general:

Let us discuss the steps involved in writing a GPIO Python script, in general:

1. Import the RPi.GPIO package.
2. Set the numbering style to be used. Use the method GPIO.setmode() for this. It takes either GPIO.BOARD or GPIO.BCM as the parameter. GPIO.BOARD stands for physical numbering and GPIO.BCM stands for Broadcom numbering.
3. Set up the necessary input and output pins.
4. Read inputs and give outputs.
5. Clean up GPIO and exit. You must clean up the pin set-ups before your program exits otherwise those pin settings will persist, and that might cause trouble when you use the same pins in another program. To clean up the entire set of pins, invoke GPIO.cleanup().

# **Set Up your Raspberry Pi**

To set up RPi, you will need:

● SD card of Minimum size 4Gb; Class 4 and above.

● HDMI for HD TVs and monitors with HDMI inpu

● Hdmi Cable.

● Keyboard and Mouse

● Power adapter

● A USB memory card reader

# **Preparing your SD card:**

The SD card contains the Raspberry Pi’s operating system (the OS is the software that makes it work, like Windows on a PC or OSX on a Mac). This is very different from most computers and it is what many people find the most daunting part of setting up their Raspberry Pi. The following instructions are for Windows users. Linux and Mac users can find instructions at [www.raspberrypi.org/downloads](http://www.raspberrypi.org/downloads)

**Steps to Follow**

**Step 1:**

Download the Raspberry Pi operating system The recommended OS is called Raspbian. Download it here:

[**https://www.raspberrypi.org/downloads/**](https://www.raspberrypi.org/downloads/)

**Step 2:**

a. Unzip the file that you just downloaded

b. Right click on the file and choose “Extract all”.

c. Follow the instructions—you will end up with a file ending in .img

**Step 3:**

a. 3.Download the Win32DiskImager software

b. Download win32diskimager-binary.zip http://sourceforge.net/projects/win32diskimager

c. Unzip it in the same way you did the Raspbian .zip file

d. You now have a new folder called win32diskimager-binary

**Step 4:**

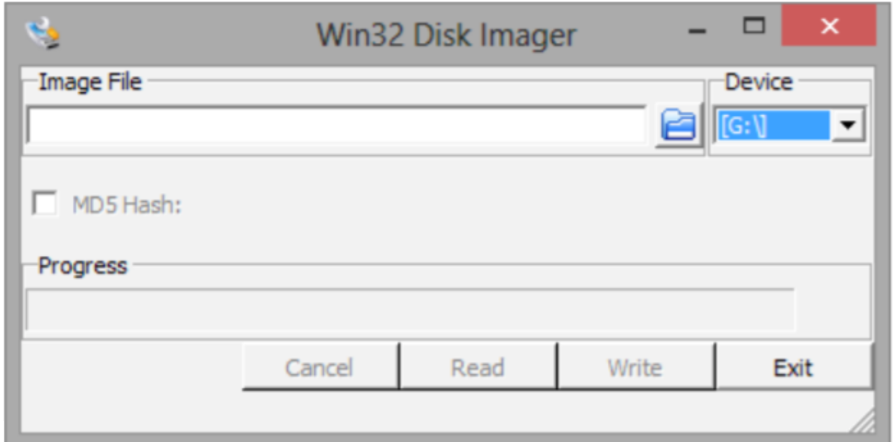
a. Download the Win32DiskImager software

b. Download win32diskimager-binary.zip http://sourceforge.net/projects/win32diskimager

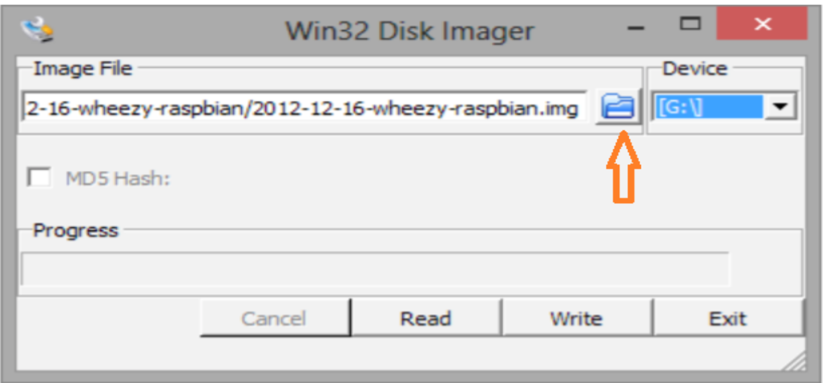
c. Unzip it in the same way you did the Raspbian .zip file

d. You now have a new folder called win32diskimager-binary

e. If the SD card (Device) you are using isn’t found automatically then click on the drop down box (highlighted in blue



d. In the Image File box, choose the Raspbian .img file that you downloaded



Click Write

f. After a few minutes you will have an SD card that you can use in the RPi.

**Step 5:**

Booting your Raspberry Pi for the first time

a. Connect the keyboard, mouse to the Pi

b. Next, connect the HDMI to the Pi and then to HDMI port on your display screen.

c. Now it is time to turn on our Raspberry Pi. When the memory card, HDMI lead, Ethernet cable, mouse and keyboard

are plugged in, plug in the power lead.

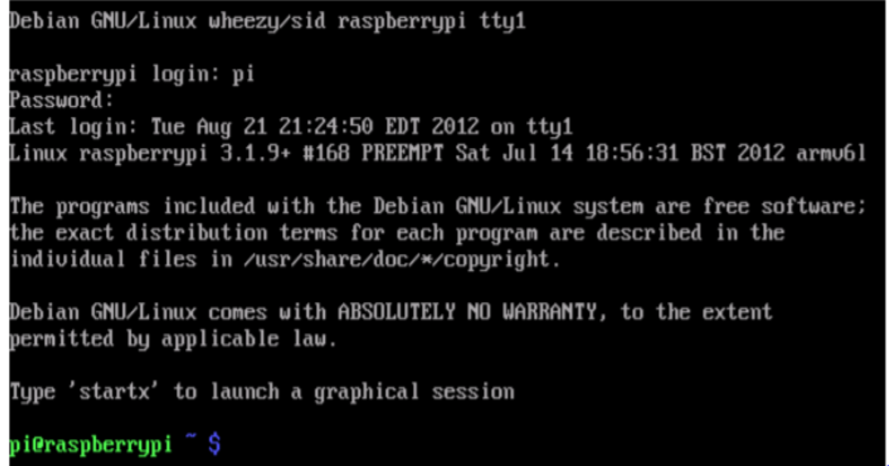
As soon as you do this. You screen should be black and filled with white text. This will be visible every time you turn on your raspberry pi.

**Step 6:**

Wait until you see raspberry pi Login Screen

● Username = pi [ENTER]

● Password = raspberry [ENTER]



**Programming on the Raspberry Pi**

**Python:**

Python is a free to use programming language that runs on Windows, Linux/Unix, Mac OS X and has even been ported to Java and .NET virtual machines.

**Idle**

In order to create our python-run programmes we will first need to write them. This is where IDLE comes in.

IDLE is a special text editor software — like Microsoft Word — however it understands the language Python. This enables us to write in a language that our Raspberry Pi will be able to understand and interpret

**Key Applications of Raspberry Pi**

1. Low cost PC/tablet/laptop

2. Industrial/Home automation •

3. Print server

4. Wireless access point

5 Environmental sensing/monitoring (e.g. weather station)

6. IoT applications

7. Robotics

8. Server/cloud server

# **CONFIGURING GPIO PINS AS OUTPUTS - BLINKING LED [LAB 1B]**

**OBJECTIVE:** To learn basics about gpio pins, configuring them as outputs, learn to write basic python code and PWM for dimmable lights.

### **COMPONENTS:**

1) RPi 3

2) Breadboard

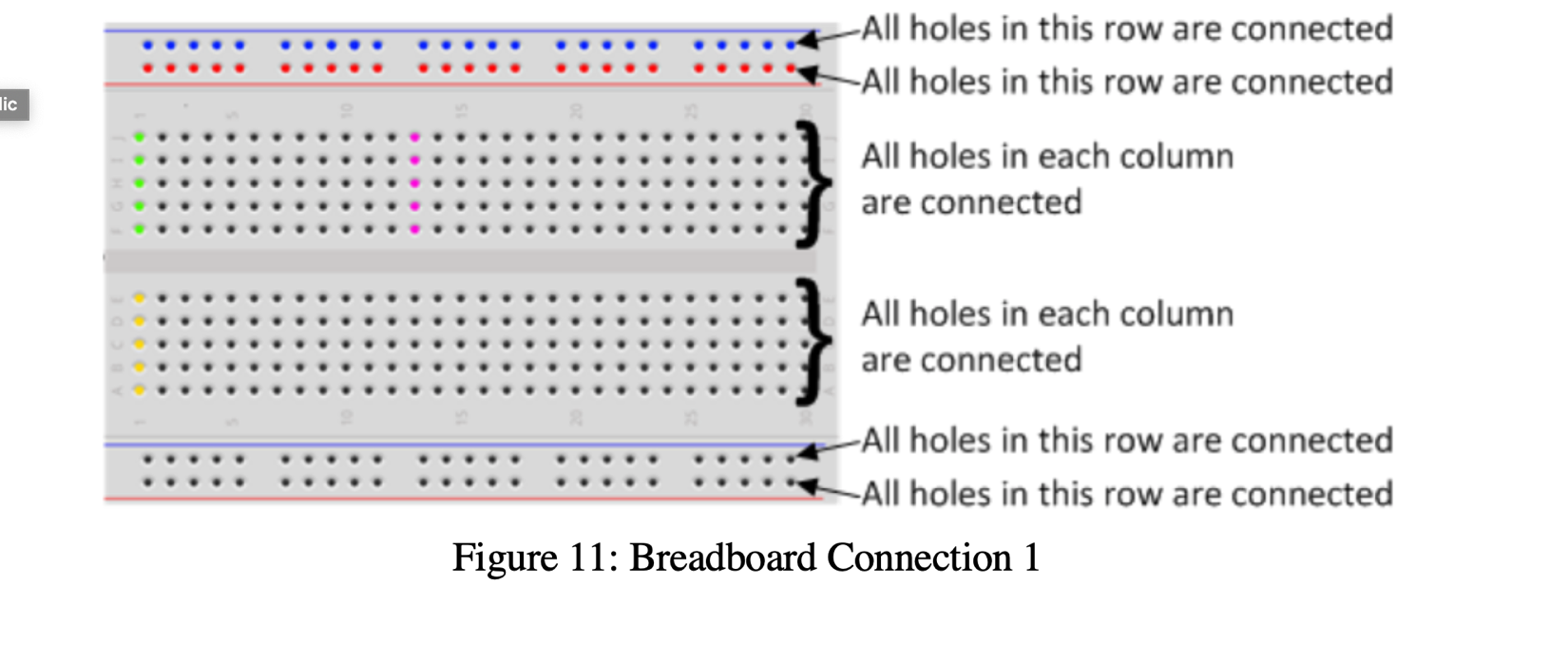
3) 100 Ω Resistor x 4

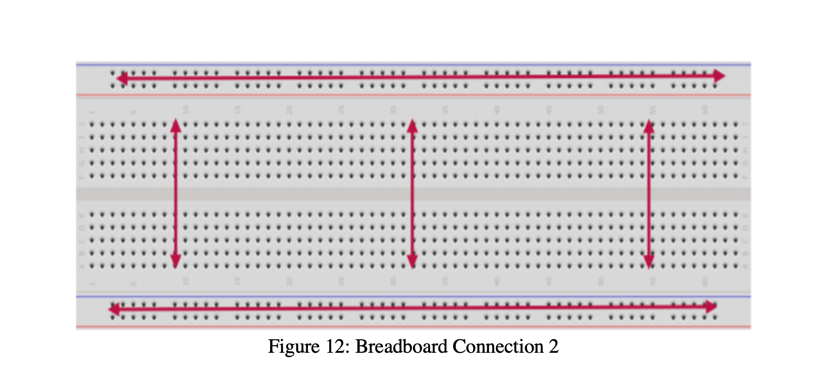
4) LED x 4

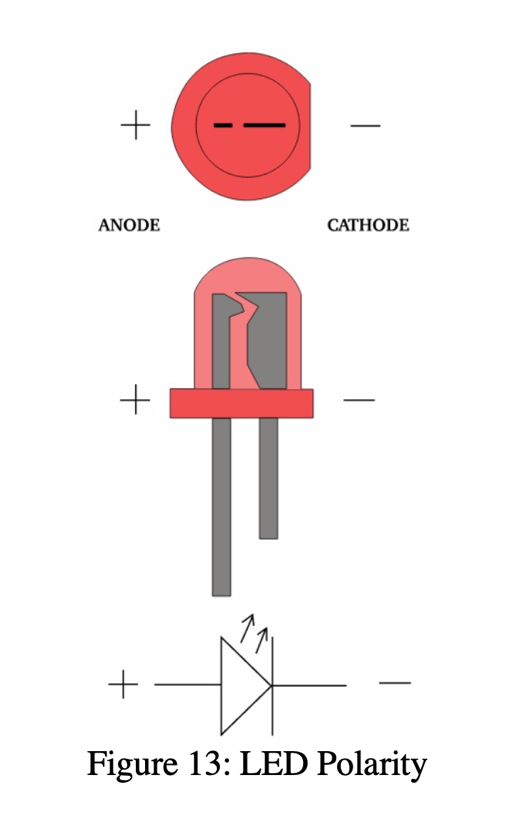
5) Connecting Wires

### Breadboard:

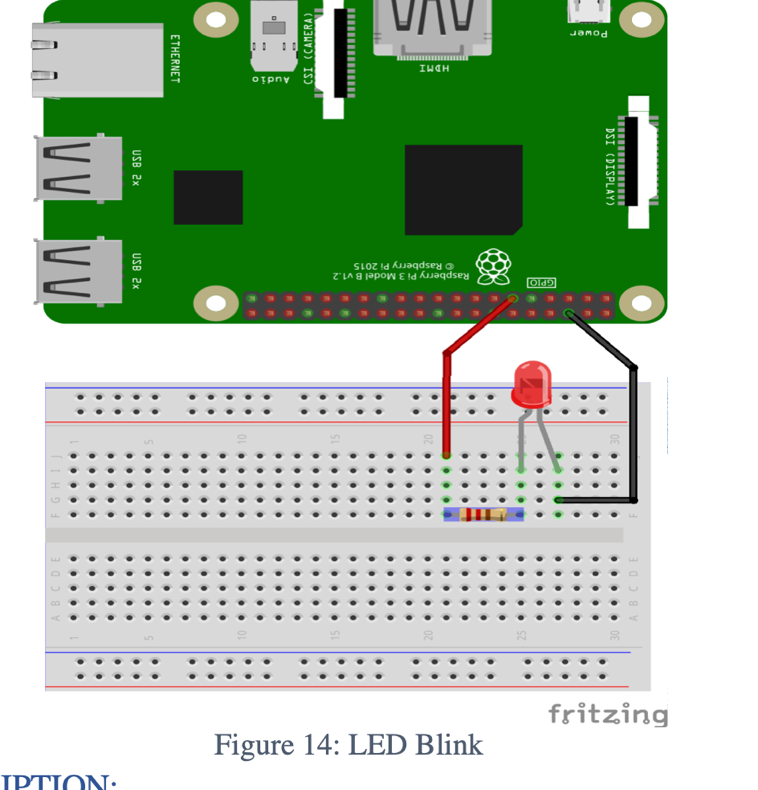
A breadboard is used for making an experimental model of an electric circuit. We are going to be using a half size solderless breadboard (plugboard). The reason for this is that you can simply push hardware in and out of the board without the need of soldering. This will ensure that if any mistakes are made you have not permanently damaged the hardware.







# CIRCUIT DIAGRAM:



Connect the pin# 11 of RPi with 100 Ω resistor, and connect the other side of resistor with the positive (+) of LED. Connect the negative (-) of LED to the ground (GND) of RPi Pin# 6. Resistor is used to limit the current.

**CREATING THE PROGRAM:**

1. Go to TERMINAL
2. Create a folder for all our programs. Type the command:

$ mkdir workspace

1. Open this folder in your file system
2. Right click the mouse and choose new file and name it as lab1.py
3. Right click on the file and open with IDLE3. Make it default program.
4. Type the below code and save the file.
5. Go to terminal and navigate to workspace folder by typing the command:

$ cd workspace

Code

*# basic one time blink*

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

**import** **time**

GPIO.setmode(GPIO.BOARD) *# to use Raspberry Pi board pin numbers*

GPIO.setup(11, GPIO.OUT) *# set up GPIO output channel*

GPIO.output(11, GPIO.LOW) *# set RPi board pin 11 low. Turn off LED.*

time.sleep(1)

GPIO.output(11, GPIO.HIGH) *# set RPi board pin 11 high. Turn on LED.*

**EXPLANATION**  
With reference to the above code:

●The first line tells the Python interpreter (the thing that runs the Python code) that it will be using a ‘library’ that will tell it how to work with the Raspberry Pi’s GPIO pins. A ‘library’ gives a programming language extra commands that can be used to do something different that it previously did not know how to do. This is like adding a new channel to your TV so you can watch something different.

●Imports the Time library so that we can pause the script later on.

●Each pin on the Pi has several different names, so you need to tell the program which naming convention is to be used. In the above case we are using physical pin numbering.

●Pin 11 is going to be used for outputting information, which means you are going to be able to turn the pin ‘on’ and ‘off’.

●GPIO.output(11, GPIO.HIGH) -- This turns the GPIO pin ‘on’. What this actually means is that the pin is made to provide power of 3.3volts. This is enough to turn the LED in our circuit on.

●time.sleep(2) -- Pauses the Python program for 2 second

# LED Fade Using Pulse Width Modulation **LAB 1C**

**OBJECTIVE:** PWM is a type of Digital Signal. A Digital Signal can have only two possible states, ON or OFF, 0 or 1, or in the case of this project, 0 or 3.3 volts. That’s why making the LED blink didn’t require PWM, because the LED was simply turning ON and OFF. In PWM signals, we can have both of these two states for a specified time period.

Suppose you want to control the brightness of an LED, the possible approach is to turn on an LED for a small period of time and then turn it off again for a small period of time. So, when this ON and OFF happens at very high speed, it gives the effect of dimmed LED.

### **COMPONENTS:**

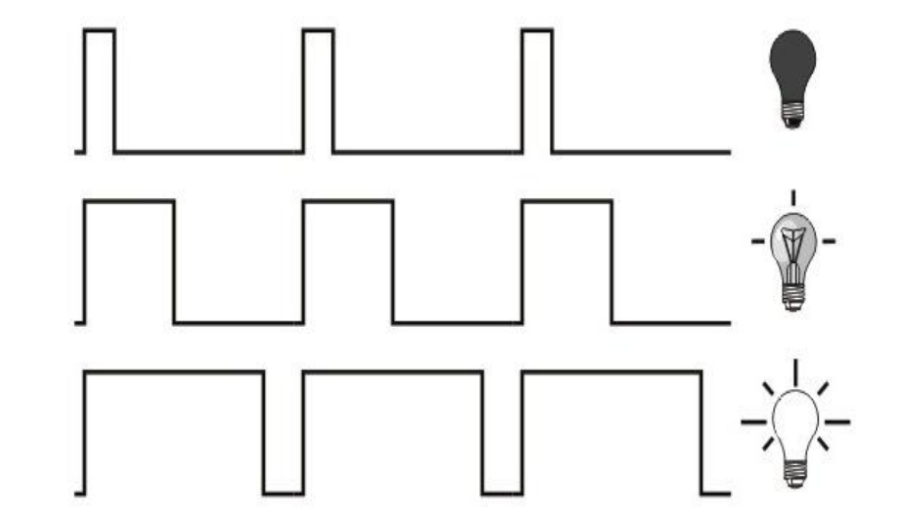
1) RPi 3

2) Breadboard

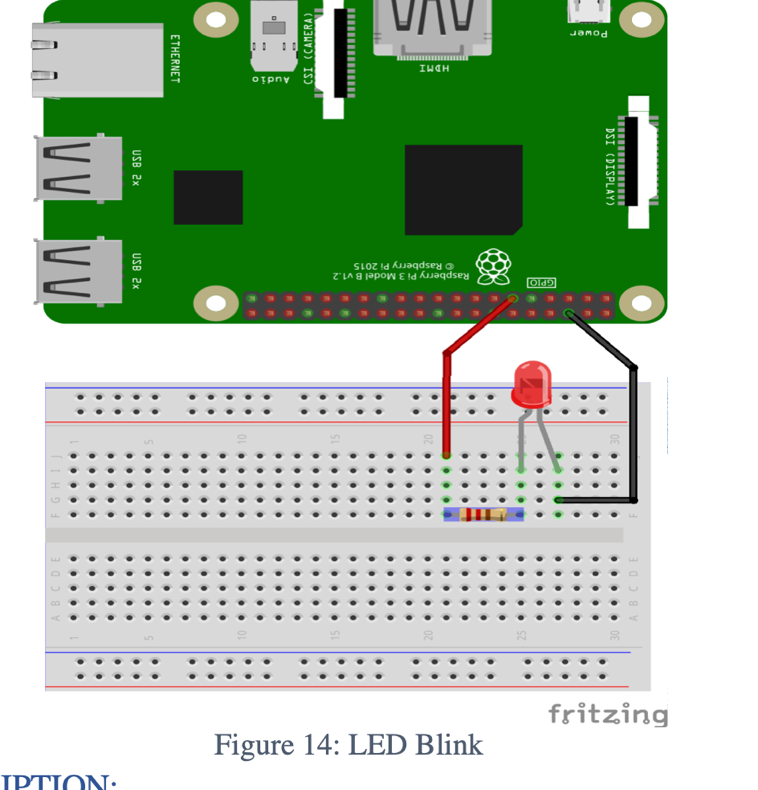
3) 100 Ω Resistor x 4

4) LED x 4

5) Connecting Wires



# CIRCUIT DIAGRAM:



Connect the pin# 11 of RPi with 100 Ω resistor, and connect the other side of resistor with the positive (+) of LED. Connect the negative (-) of LED to the ground (GND) of RPi Pin# 6. Resistor is used to limit the current.

Code

**import** **time**

GPIO.setwarnings(**False**)

GPIO.setmode (GPIO.BCM)

GPIO.setup(17,GPIO.OUT) *# initialize GPIO17 as an output.*

p = GPIO.PWM(17,100) *# 100Hz frequency*

p.start(0) *#start at 0% duty cycle*

**while** **True**:

**for** x **in** range (50):

p.ChangeDutyCycle(x)

time.sleep(0.1)

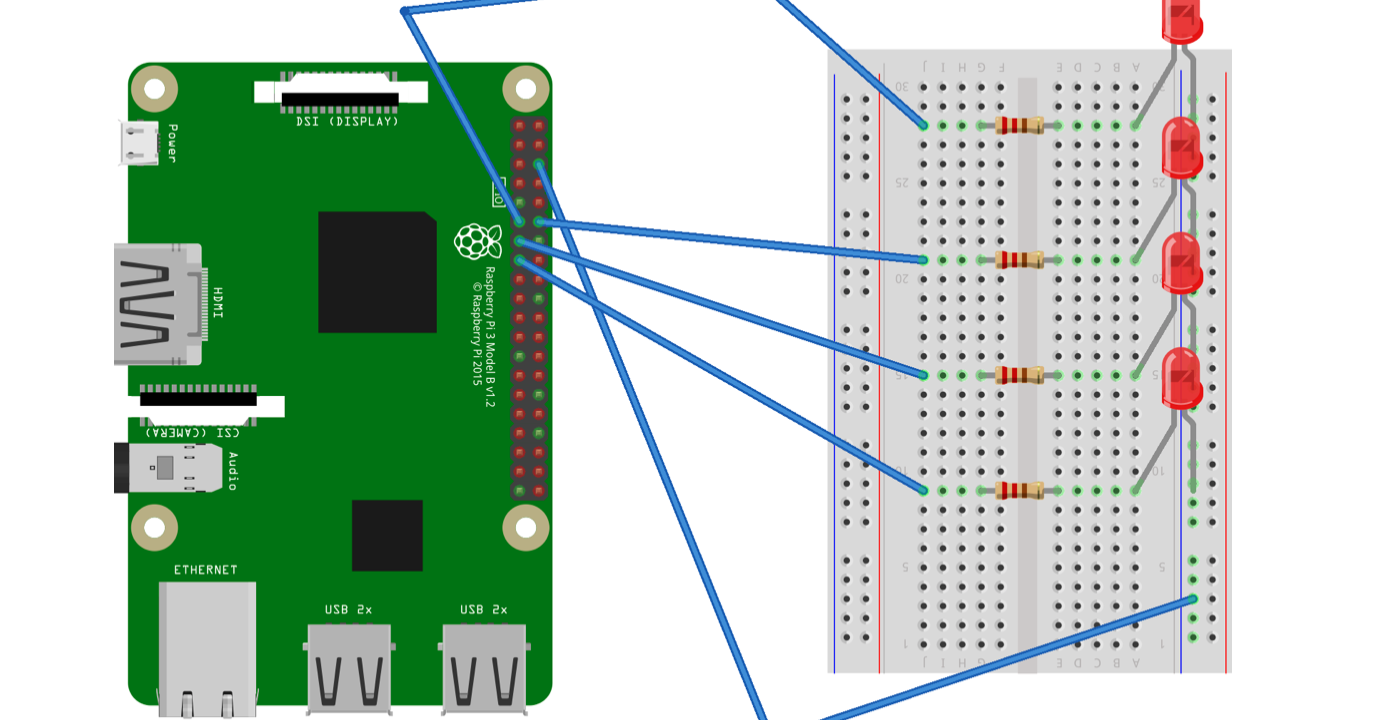
**for** x **in** range (50):

p.ChangeDutyCycle(50-x)

time.sleep(0.1)

# Flowing LED Lights

CIRCUIT DIAGRAM:



Code:

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

**import** **time**

pins = [11, 12, 13, 15]

GPIO.setmode(GPIO.BOARD) *# to use Raspberry Pi board pin numbers*

**for** pin **in** pins:

GPIO.setup(pin, GPIO.OUT) *# Set all pins mode as output*

**def** setup():

**try**:

loop()

**except** **KeyboardInterrupt**: *# When 'Ctrl+C' is pressed, the child program destroy() will be executed.*

destroy()

**def** loop():

**while** **True**:

**for** pin **in** pins:

GPIO.output(pin, GPIO.LOW)

time.sleep(0.05)

GPIO.output(pin, GPIO.HIGH)

**for** pin **in** reversed(pins):

GPIO.output(pin, GPIO.LOW)

time.sleep(0.05)

GPIO.output(pin, GPIO.HIGH)

**def** destroy():

**for** pin **in** pins:

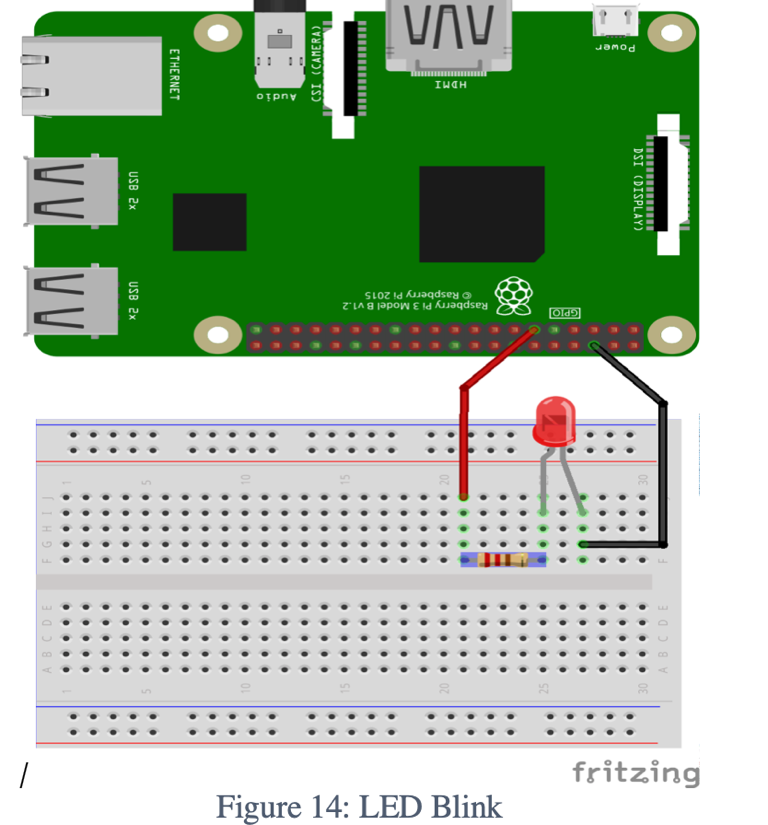
GPIO.output(pin, GPIO.HIGH)

GPIO.cleanup() *# Release resource*

setup()

**Introducing Tkinter with Raspberry pi [LAB 1D]**

**CIRCUIT DIAGRAM:**



Code

*# import the important library*

**import** **tkinter** **as** **tk**

mainwindow=tk.Tk()

mainwindow.title('My First UI Test ')

mainwindow.geometry('640x340')

my\_label=tk.Label(mainwindow,text=" On/Off ",

font=("Arial",22), bg= "Green",fg="white")

my\_label.grid(row=0,column=0,sticky='NSEW',padx=10,pady=10)

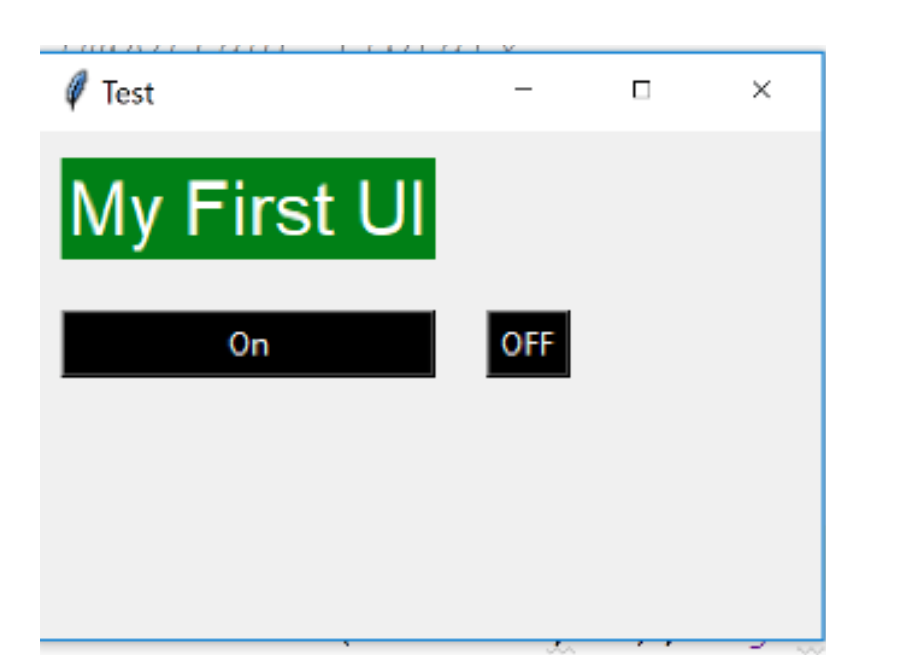
button\_on=tk.Button(mainwindow,text="On",bg="black",fg="white")

button\_on.grid(row=1,column=0,sticky='NSEW',padx=10,pady=10)

button\_off=tk.Button(mainwindow,text="OFF",bg="black",fg="white")

button\_off.grid(row=1,column=1,columnspa=1,sticky='NSEW',padx=10,pady=10)

mainwindow.mainloop()



### **Step 2:** Adding Functionality to execute a function when Button is pressed using Lambda Function

*# import the important library*

**import** **tkinter** **as** **tk**

mainwindow=tk.Tk()

mainwindow.title('Test ')

mainwindow.geometry('640x340')

my\_label=tk.Label(mainwindow,text="My First UI",

font=("Arial",22), bg= "Green",fg="white")

my\_label.grid(row=0,column=0,sticky='NSEW',padx=10,pady=10)

button\_on=tk.Button(mainwindow,text="On",bg="black",fg="white",

command=**lambda** :my\_on())

button\_on.grid(row=1,column=0,sticky='NSEW',padx=10,pady=10)

button\_off=tk.Button(mainwindow,text="OFF",bg="black",fg="white",

command=**lambda**:my\_off())

button\_off.grid(row=1,column=1,columnspa=1,sticky='NSEW',padx=10,pady=10)

**def** my\_on():

print('Led Turn On !!!!! ')

**def** my\_off():

print('Led Turned Off !!!!!! ')

mainwindow.mainloop()

### **Step 3:** Add the Final code to Turn LED On/Off

*# import the important library*

**import** **tkinter** **as** **tk**

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

**import** **time**

GPIO.setmode(GPIO.BOARD) *# to use Raspberry Pi board pin numbers*

GPIO.setup(11, GPIO.OUT) *# set up GPIO output channel*

mainwindow=tk.Tk()

mainwindow.title('Test ')

mainwindow.geometry('640x340')

my\_label=tk.Label(mainwindow,text="My First UI",

font=("Arial",22), bg= "Green",fg="white")

my\_label.grid(row=0,column=0,sticky='NSEW',padx=10,pady=10)

button\_on=tk.Button(mainwindow,text="On",bg="black",fg="white",

command=**lambda** :my\_on())

button\_on.grid(row=1,column=0,sticky='NSEW',padx=10,pady=10)

button\_off=tk.Button(mainwindow,text="OFF",bg="black",fg="white",

command=**lambda**:my\_off())

button\_off.grid(row=1,column=1,columnspa=1,sticky='NSEW',padx=10,pady=10)

**def** my\_on():

print('Led Turn On !!!!! ')

GPIO.output(11, GPIO.LOW) *# set RPi board pin 11 low. Turn off LED.*

time.sleep(1)

print('Yes you did it !')

**def** my\_off():

print('Led Turned Off !!!!!! ')

GPIO.output(11, GPIO.HIGH) *# set RPi board pin 11 high. Turn on LED.*

time.sleep(2)

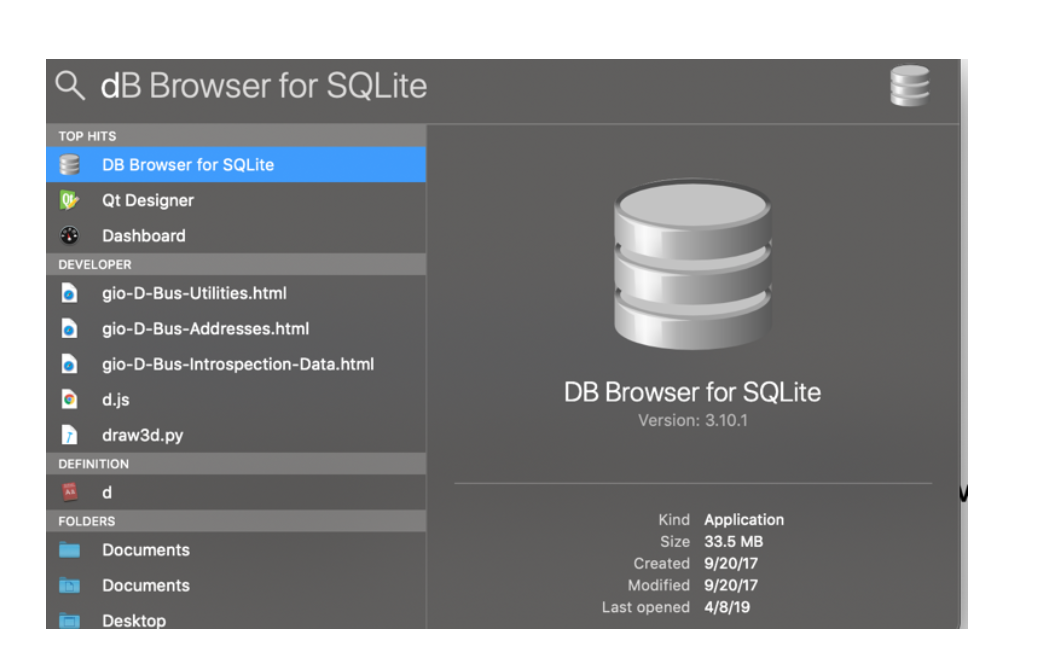
print('Great Work ! ')

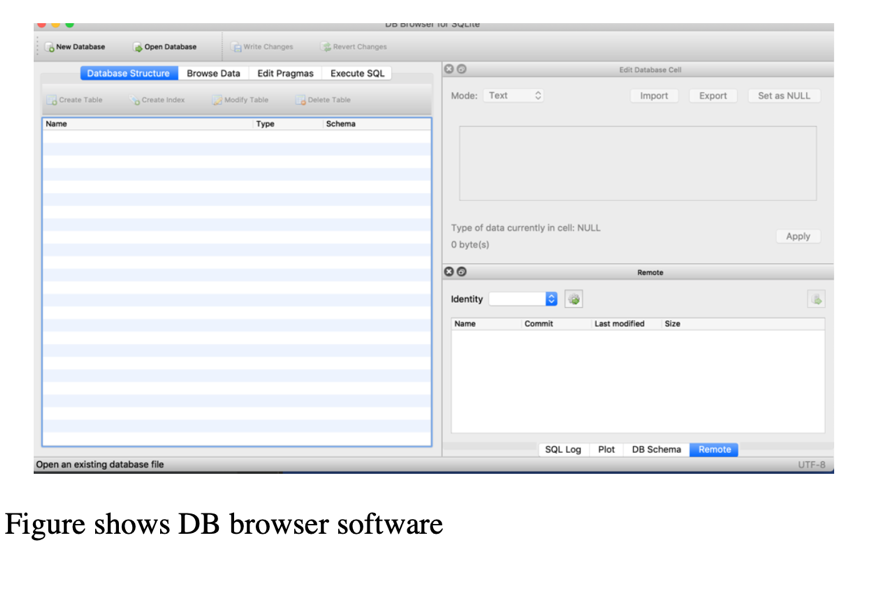
mainwindow.mainloop()

**Adding Database to our Project [LAB 1E]**

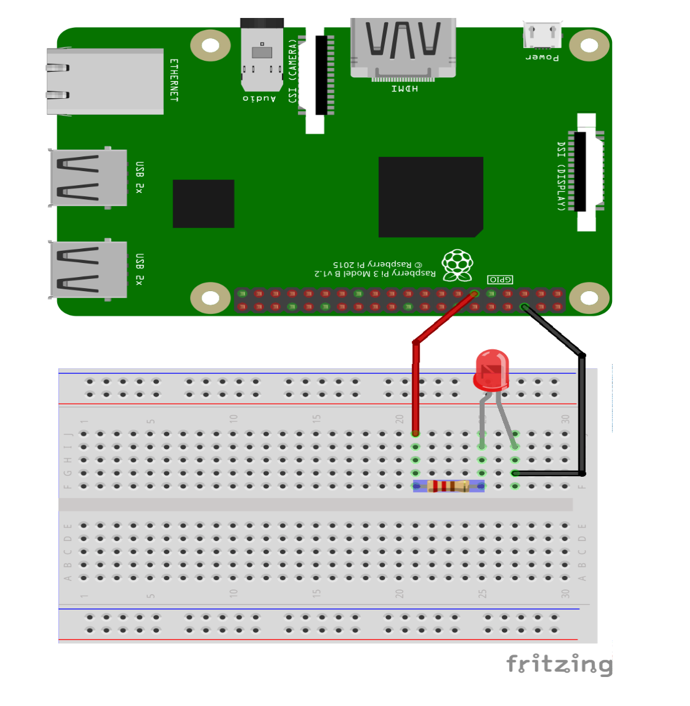
### **Objective**

We want to add time when we turned Lights on to our Database Let us learn how to do that for this, we would be using SQlite3 and make sure to download DB browser on Raspberry pi to open your database file.





## CIRCUIT DIAGRAM:



Code:

*# import the important library*

**import** **tkinter** **as** **tk**

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

**import** **time**

**import** **datetime**

**import** **sqlite3**

GPIO.setmode(GPIO.BOARD) *# to use Raspberry Pi board pin numbers*

GPIO.setup(11, GPIO.OUT) *# set up GPIO output channel*

mainwindow=tk.Tk()

mainwindow.title('Test ')

mainwindow.geometry('640x340')

my\_label=tk.Label(mainwindow,text="My First UI",

font=("Arial",22), bg= "Green",fg="white")

my\_label.grid(row=0,column=0,sticky='NSEW',padx=10,pady=10)

button\_on=tk.Button(mainwindow,text="On",bg="black",fg="white",

command=**lambda** :my\_on())

button\_on.grid(row=1,column=0,sticky='NSEW',padx=10,pady=10)

button\_off=tk.Button(mainwindow,text="OFF",bg="black",fg="white",

command=**lambda**:my\_off())

button\_off.grid(row=1,column=1,columnspa=1,sticky='NSEW',padx=10,pady=10)

**def** my\_on():

t=datetime.datetime.now()

my\_t = "**{}**-**{}**-**{}**".format(t.hour,t.minute,t.second)

database\_on(my\_t)

print('Led Turn On !!!!! ')

GPIO.output(11, GPIO.LOW) *# set RPi board pin 11 low. Turn off LED.*

time.sleep(1)

print('Yes you did it !')

**def** my\_off():

t=datetime.datetime.now()

my\_off = "**{}**-**{}**-**{}**".format(t.hour,t.minute,t.second)

database\_off(my\_off)

print('Led Turned Off !!!!!! ')

GPIO.output(11, GPIO.HIGH) *# set RPi board pin 11 high. Turn on LED.*

time.sleep(2)

print('Great Work ! ')

**def** database\_on(on\_time):

conn = sqlite3.connect('led.db')

cursor = conn.cursor()

cursor.execute(""" CREATE TABLE IF NOT EXISTS

ledon

(id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,

on\_time TEXT)""")

cursor.execute("""

INSERT INTO ledon (on\_time)

VALUES (?)""", (on\_time))

conn.commit()

cursor.close()

conn.close()

**def** database\_off(of\_time):

conn = sqlite3.connect('led.db')

cursor = conn.cursor()

cursor.execute(""" CREATE TABLE IF NOT EXISTS

ledoff

(id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,

off\_time TEXT)""")

cursor.execute("""

INSERT INTO ledoff (off\_time)

VALUES (?)""", (of\_time))

conn.commit()

cursor.close()

conn.close()

mainwindow.mainloop()

Lab 1F

# **CONFIGURING GPIO PINS AS INPUTS - CONTROLLING LEDS FROM PUSH BUTTONS**

### **Objective**

Learn how to ‘read’ values from the Raspberry Pi GPIO pins, make dimmable LEDs with PWM and Push Buttons.

## **COMPONENTS:**

1) RPi 3

2) Breadboard

3) 330 Ω Resistor x 2

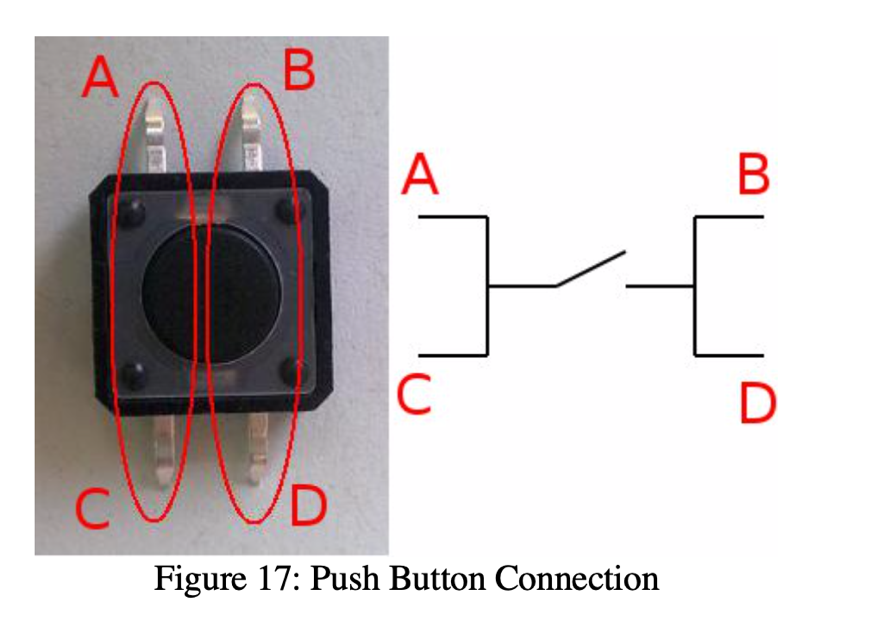
4) LED x 2

5) Push Buttons x 2

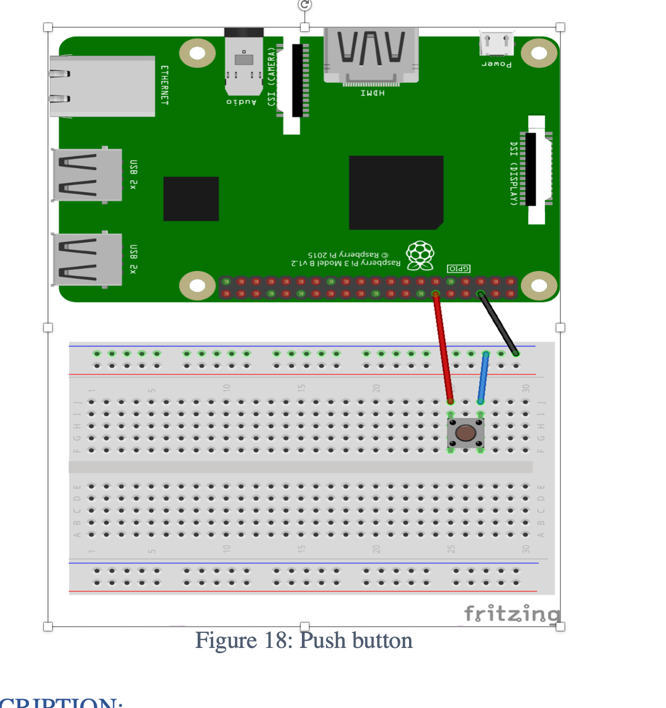
6) Connecting Wires

## Push Button

A pushbutton is a simple switch mechanism which permits user generated changes in the state of a circuit. Push Button usually comes with four legs. As you can see from the picture below, legs are always connected in groups of two. When the pushbutton is pressed all the 4 legs are connected



## CIRCUIT DIAGRAM:



Code:

# Only button as input  
  
from time import sleep # Library will let us put in delays  
import RPi.GPIO as GPIO # Import the RPi Library for GPIO pin control  
  
button1\_pin=12 # Button 1 is connected to physical pin 12  
  
GPIO.setmode(GPIO.BOARD) # Use Physical Pin Numbering Scheme  
GPIO.setup(button1\_pin,GPIO.IN,pull\_up\_down=GPIO.PUD\_UP)

# Make button1\_pin an input, Activate Pull UP Resistor  
  
while(1): # Create an infinite Loop  
 input1=GPIO.input(button1\_pin)  
  
 if input1==0: # Look for button 1 press  
 sleep(.1) # Delay  
 print ('Button 1 Pressed') # Notify User