PROGRAMMING RASPBERRY PI

LAB MANUAL

Amrita Vishwa Vidya Peetham, Amritapuri Campus

Dedicated at the lotus feet of Amma

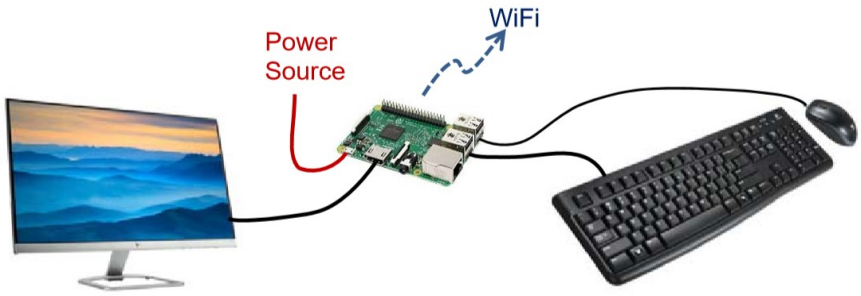
Last Updated: October 2019

Lab Sheet: 1

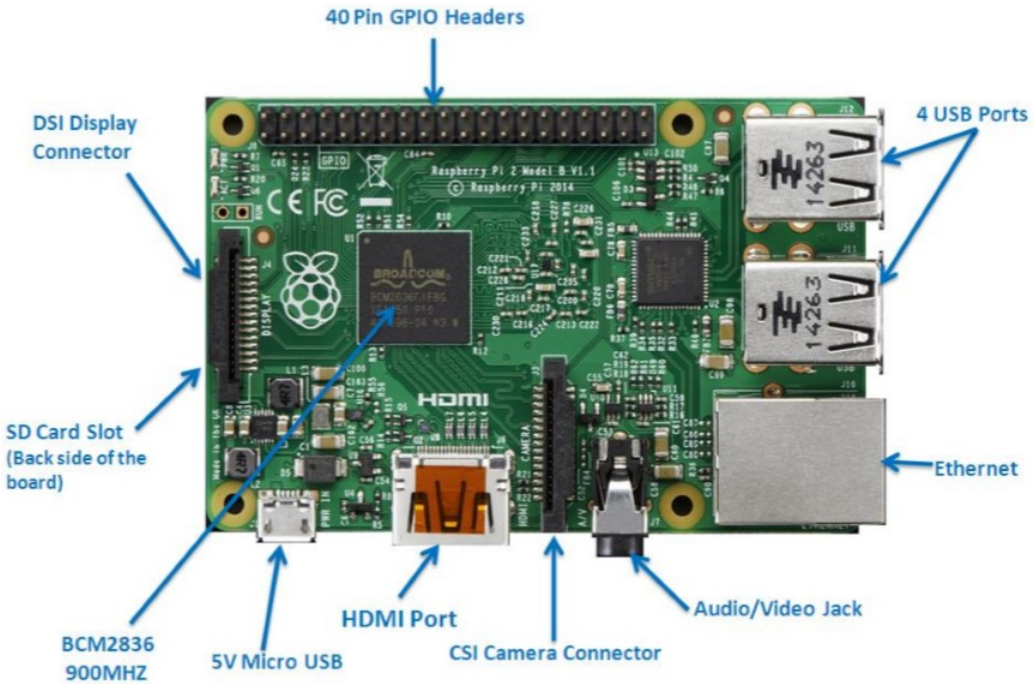
Objective: Realizing Raspberry Pi and interfacing Pi with LEDs

Hardware Requirements: Raspberry Pi; SD card at least 4GB, Monitor with HDMI (High definition multimedia interface) connector, Micro USB power supply, USB keyboard, USB mouse, breadboard, LEDs, resistors and connecting wires.

Introducing Raspberry Pi: Raspberry Pi is an ultra-small (a credit card sized), single-board computer developed in UK by the Raspberry Pi Foundation to promote teaching and studying computer science in schools. It is a low cost computer board, comes with a powerful ARM processor; can be used for learning to code and creating wonderful projects. Raspberry Pi users may have built many creative and impressive projects. Like a desktop or a laptop Pi may support video games, movies, music, and browsing internet; it sounds like a fully functional computer built on a single printed circuit board. The RPi can be set up as a standalone computer by connecting a monitor, keyboard and mouse through its various interfaces as shown in the following figure.



Raspberry Pi cabling: Push SD card into the SD card slot; plugin the HDMI cable into the HDMI output of the Raspberry Pi and connect to the TV/monitor; insert the network cable; connect the keyboard and mouse via USB ports; plugin the power supply into the micro USB port. The device is now ready for installing operating system (OS) - Raspbian, which is based on the open source Linux (Debian).



This single board, inexpensive computer is packed with Ethernet, USB ports, a high powered graphics engine; there is no power switch; Micro-USB connector is used to supply power (cannot be used as an additional USB port; only for power). It does not feature a built-in hard disk or solid-state drive, instead it supports SD card for booting and long-term storage. As there is no hard drive on the Pi; everything is stored on the SD Card itself.

Hardware Specifications:

- Processor: Broadcom BCM 2837B0 chip; Quad-core 64 bit ARM coretex A53 CPU (ARMv8)
- Clocked at 1.4 GHz
- 400 MHz Video Core IV GPU : HDMI port; MIPI DSI display port; MIPI CSI camera port; 4 pole stereo output and composite video port
- Memory: 1GB LPDDR2 SDRAM
- 802.11ac Wireless LAN, Bluetooth 4.2
- USB : 4 x 2.0

Steps for Installing Os (Rasbian) on Raspberry-Pi :

1. Download Rasbian image from official site of raspberry pi: raspberrypi.org
2. Burn the Raspbian image in a SD card ;The downloaded image cannot be copied directly to the SD. It is necessary to use an another software tool Win32 disk to burn/write the image file on the SD card.
3. Extract the Image file.
4. Insert SD Card into PC/Laptop (system on which OS and win32disk is downloaded). Make sure that SD card is empty and have atleast 4GB memory
4. Open win32 disk; browse the image file which has been downloaded from Raspberry pi website and continue writing.

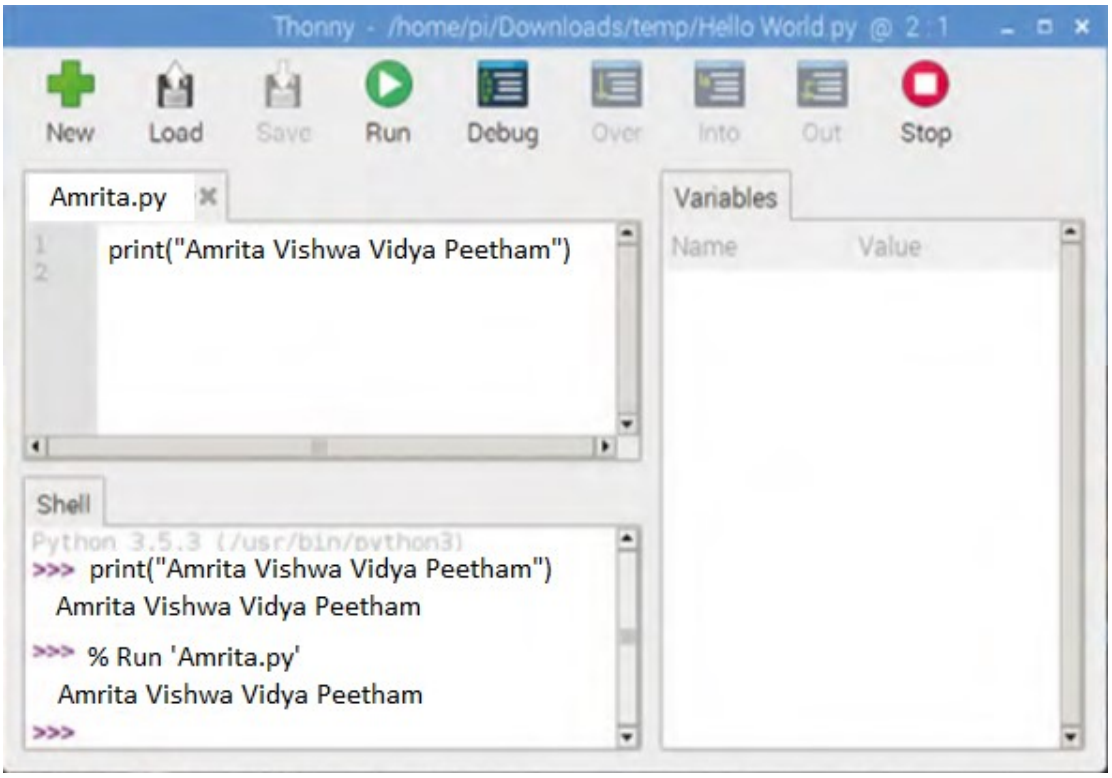
Step 6: When writing is completed, remove the SD card and insert it into the raspberry pi board; and switch on the power supply; at this point you should see a window console showing booting process; blinking red LED on your Raspberrypi shows that the status of successful booting. Once the boot process finishes, you should see a desktop environment and a welcoming window where to configure, language, time and network.

Thonny Python IDE: Thonny is a package called integrated development environment (IDE); it integrates different tools which are needed to write or develop a software into a single user interface, or an environment. Now a days different IDEs available, some of which support multiple programming languages; while Thonny focus on a more powerful traditional programming language called Python; it provides a toolbar with user friendly menu items, allowing a user to create, save, load, test and run a Python code. In fact Thonny supports two modes of interfacing such as a Normal Mode and a Simple Mode which is better for beginners. Here in our lab, we use Simple Mode, which is loaded by default when you open the IDE. The following figure shows the IDE with three different panels such as script area, variable area and Python shell.

Script Area: This is a place, where you may write your Python code; which is provided with a small side margin for showing line numbers.

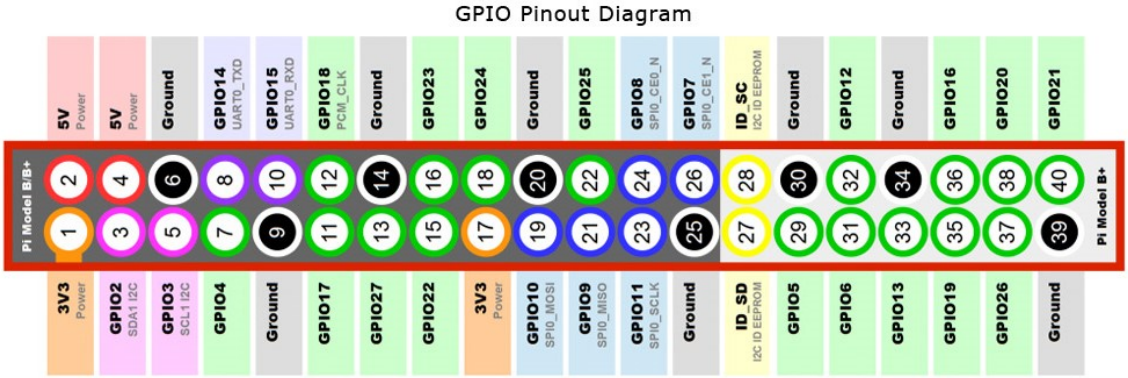
Python Shell: In this panel, individual Python instructions can be typed and run; also it provides information about running programs.

Variables Area: Any variable created or used in the code is shown on this area , along with it's values, for easy reference.

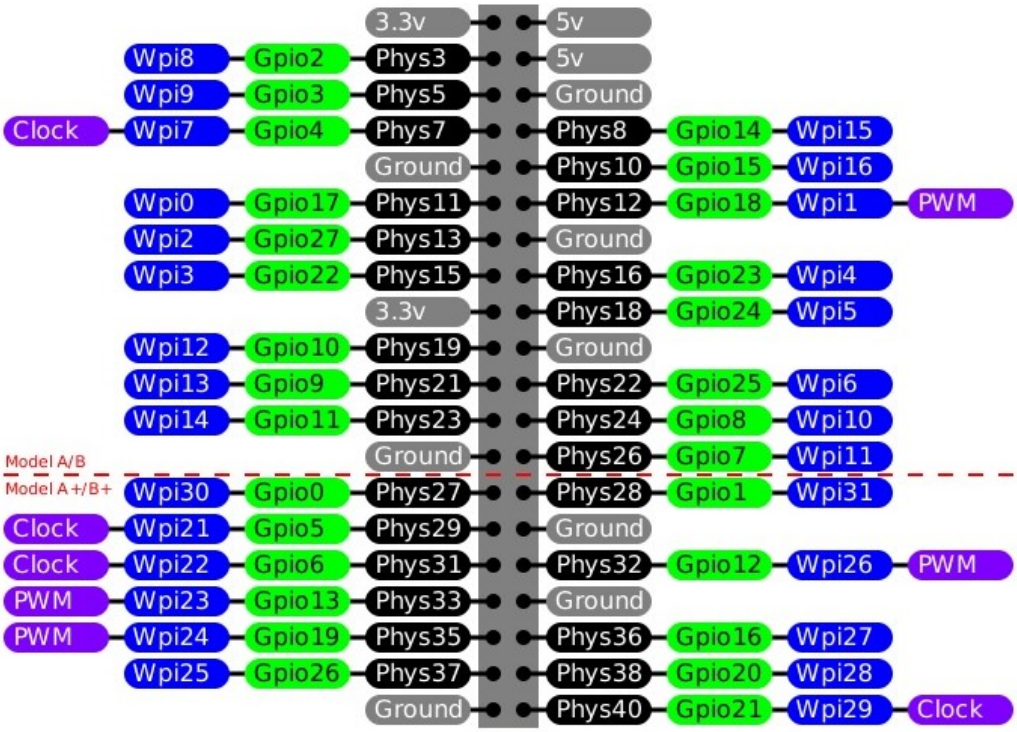


Start your first program by clicking on the Python shell area at the bottom-left and type the following instruction, print(“Amrita Vishwa Vidya Peetham"); and press the enter key. You may see that your program begins to run instantly: hence, shell is a direct line to the Python interpreter, instantly it interprets the instruction and respond in the same shell area, with the message “Amrita Vishwa Vidya Peetham". This is known as interactive mode (a face-to-face conversation). We do not use interactive mode for Python coding, type your code on the script area at the left hand side, save it in the name Amrita.py and run.

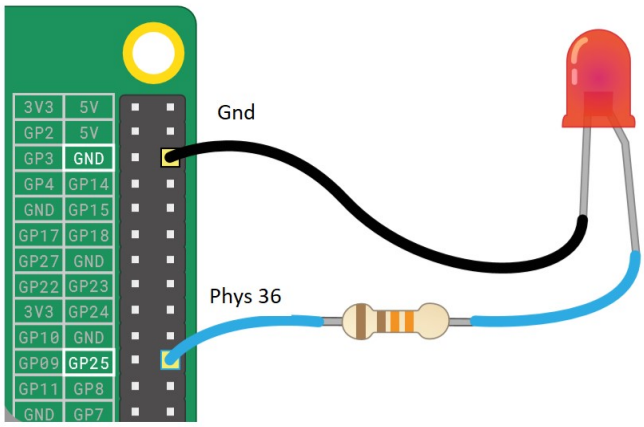
GPIO header: The Raspberry Pi GPIO header is made up of 40 male pins. Some pins are available for user purpose, some pins provide power, while other pins are reserved for special purpose. Pi header pins are be identified or located with unique numbers.



Generally there are three differenet numbering schemes followed such as WPi (wiring pi), Phys (Physical) and BCM (BroadCom) numbering; for example physical pin 3 (Phys3) can be referred to as Gpio2 or Wpi8; go through the figures and try to understand the different pin numbering schemes.

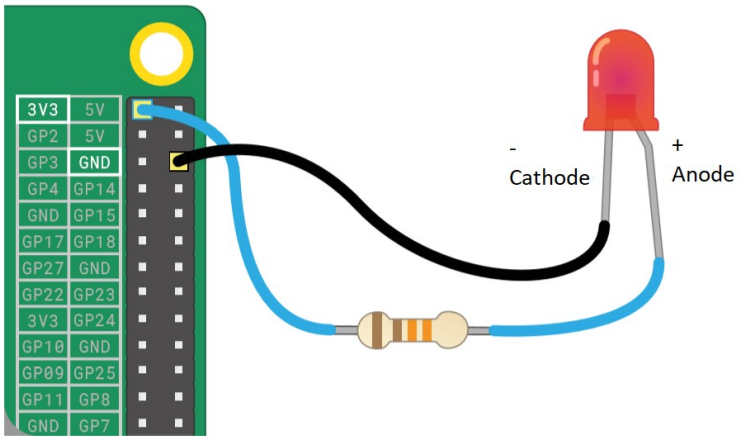


1. Python code for configuring the thirty sixth (36th) physical pin on Raspberry-pi board as output; connect an LED to the output pin via a 270 ohm resister; switch it ON for five seconds and turn the LED off




```
#My first program – Interfacing an LED
import RPi.GPIO as GPIO                 #include GPIO library
import time                             # include time library
ledpin = 36                             # selecting a pin to be used for connecting LED
GPIO.setwarnings(False)                 # Disable warnings
GPIO.setmode (GPIO.BOARD)               # physical pin numbering scheme is followed
                                        #  GPIO.BCM uses BroadCom numbering scheme
GPIO. setup ( ledpin , GPIO.OUT)        # selected pin is configured as output
GPIO. output ( ledpin, True)  # write high voltage on the  led pin
                                        #  to turn the LED on
time.sleep (5)                          # sleep for 5 seconds
GPIO. output ( ledpin, False) # write low voltage on the  led pin
                                        # to turn the LED off
GPIO.cleanup()                          # clear GPIO pins
```

Testing your LED: You may check that your LED is working properly or not; here in the above schematic, anode of the LED is connected to the +3V power pin of Pi via a resister and the cathode is connected to the ground pin.



---

2. Modify the above code to use gpiozero library; which is an efficient Python library which provides a simple interface to standard GPIO devices (such as led, button, sensor, etc) with Raspberry Pi. It was created by Ben Nuttall of the Raspberry Pi Foundation, Dave Jones, and other contributors

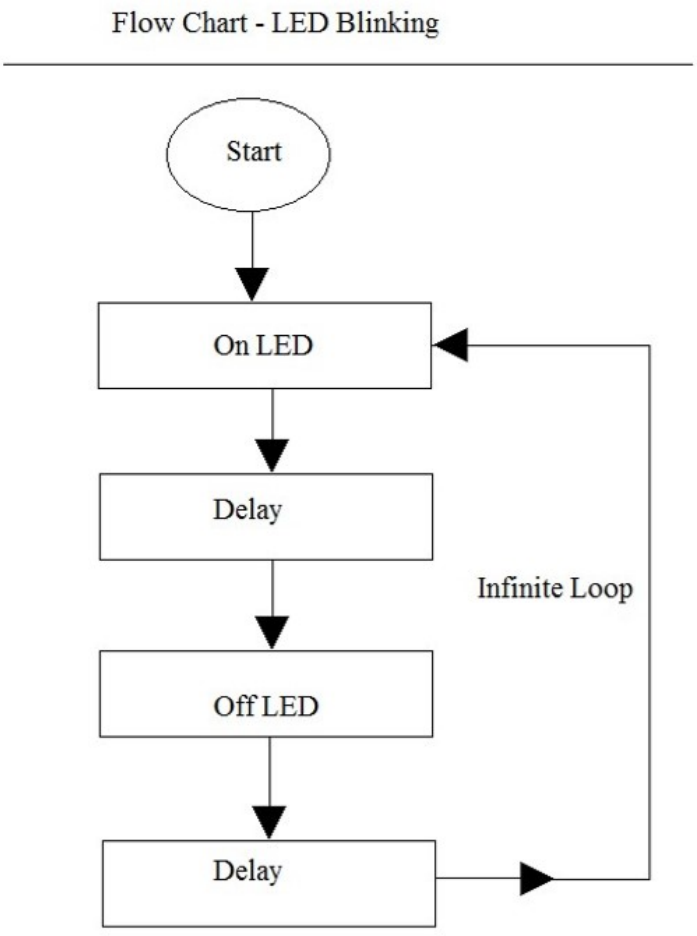
```
#My second program – Interfacing LED using gpiozero library
from gpiozero import LED     # include LED module from gpiozero library
from time import sleep       # only sleep function is included
GPIO.setwarnings(False)      # Disable warnings
led = LED(16)                # by default BCM numbering scheme is followed
led.on ()                    # turn the LED on
sleep(1)                     # sleep for 1 second
led.off ()                   # turn the LED off
GPIO.cleanup()               # clear GPIO pins
```

Note that by default the gpiozero library uses BroadCoM (BCM) pin numbering scheme.

```
led =LED(16)            # using default numbering scheme : BCM
led =LED("BCM16")       # using BCM numbering scheme
led =LED("BOARD36")     # using physical pin numbering
led =LED("WPI27")       # using physical wiring-pi pin numbering
```

---

3. Execute the above module in an infinite loop (never ending while loop) - Blinking an LED with a delay of one second



```
#My second program – Interfacing an LED
import RPi.GPIO as GPIO                 #include GPIO library
import time                             # include time library
ledpin = 36                             # selecting a pin to be used for connecting LED
GPIO.setwarnings(False)                 # disable warnings
GPIO.setmode (GPIO.BOARD)               # physical pin numbering scheme is followed
GPIO. setup ( ledpin , GPIO.OUT)        # selected pin is configured as output
while (True):                                   # loop begins
        GPIO. output ( ledpin , True)           # turn the LED on
        time.sleep (1)                          # sleep for 1 second
        GPIO. output ( ledpin , False)          # turn the LED off
        time.sleep (1)                          # sleep for 1 second
GPIO.cleanup()                                  # clear GPIO pins
```

---

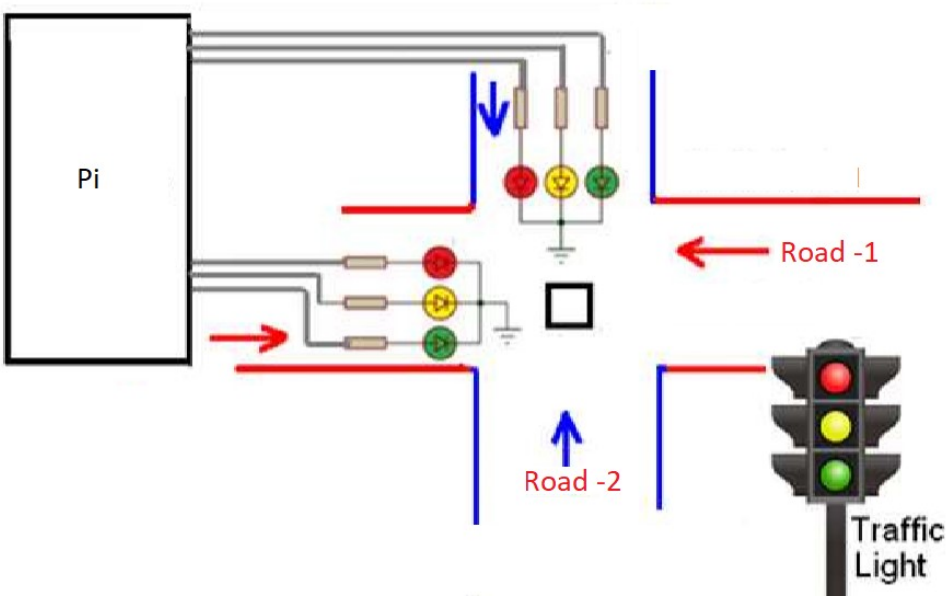
4. Configure Phys3 and Phys7 (follow physical pin numbering scheme) as output; and connect two different color LEDs to the outputs; turn the LEDs on and off alternatively with a delay of one second; note that only ten switching is recommended – use for loop

```
redled= 3                               # Phys3 to be used for red LED
greenled= 7                             # Phys7 to be used for green LED
for i in range(0,10):                   # this loop executes 10 times
        GPIO. output ( redled , True)           # turn the red LED on
        GPIO. output ( greenled , False)        # turn the green LED off
        time.sleep (1)                          # sleep for 1 second
        GPIO. output ( redled, False)           # turn the red LED off
        GPIO. output ( greenled , True)         # turn the green LED on
        time.sleep (1)                          # sleep for 1 second
```

5. **Traffic Light Control systems** are signaling devices, positioned at traffic circles (intersections), pedestrian crossings and other locations, to control a flow of traffic in a certain time period, by displaying lights in standard colors such as Red (denotes STOP : Prohibit traffic), Yellow (LISTEN : Prepare to stop) and Green (GO : Allows traffic). Traffic jamming is a severe problem in many major cities across the country.

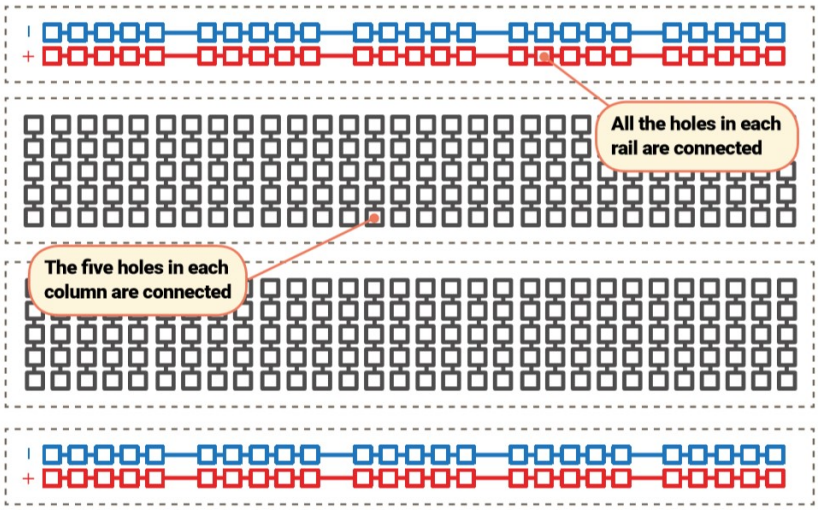
   Exercise 1: Configure pins 3, 5 and 7 (physical pin numbering) as output; and connect three different color LEDs such as red, green and yellow to the outputs; implement simple traffic light system: Give green signal for 5 seconds; red signal for 3 seconds and yellow for 1 second.

   Assignment 1: Design a four-way Traffic Light Control system for the smooth and safe movement of vehicles.



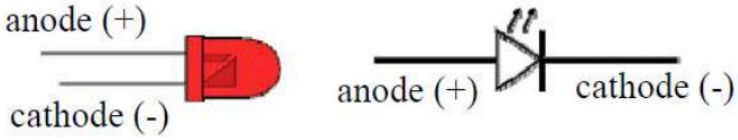
Description of the components

Breadboard: Breadboard is a temporary building base for prototyping or designing electronic models; this is a widely used tool for designing and testing electronic circuit. Infact, the board does not require any physical soldering between components; hence and the components used in one schematic can be reused in other schematic. A breadboard is covered with holes – spaced, to match components,; under these holes are metal strips which act like the jumper wires. Breadboard lets you insert components and have them connected through metal tracks which are hidden beneath its surface.



Breadboards also have strips of holes down the sides, typically marked with red and black or red and blue stripes. These are the power rails, and are designed to make wiring easier: you can connect a single wire from the Raspberry Pi's ground pin to one of the power rails – typically marked with a blue or black stripe and a minus symbol – to provide a common ground for lots of components on the breadboard, and you can do the same if your circuit needs 3.3 V or 5 V power. Adding electronic components to a breadboard is simple: just line their leads up with the holes and gently push until the component is in place.

Input Output Devices: Input and output devices are the important components of an electronic system, we cannot imagine any device without the input and output. One may interface many different input and output devices such as LEDs, Traffic Light, Buzzer, Speakers, DC Motor, Stepper Motor, Seven Segment Display, LCD, Switches, Buttons, Keypad, Potentiometer, Light Dependant Resistor (LDR), Sensors, etc. to the Raspberry Pi. But LED and switch are considered the basic input and output device.



Light Emitting Diode: LED, the Light Emitting Diode is a semiconductor device that emits light energy when there is a current flow. Now a days LEDs are available in various colors (red, green, blue, yellow, etc.); the colour depends the semiconductor material used. All LEDs have two leads anode (positive lead) and cathode (negative lead); the length of anode is larger than the length of cathode. The ON/Off state of an LED can be controlled directly through a code with your Pi. LEDs are available in a wide range of shapes, colours, and sizes, but not all are suitable for use with the Raspberry Pi; avoid any which say they are designed for 5 V or 12 V power supplies.

Resisters: Resistors are components which control the flow of electrical current, and are available in different values measured in a unit called ohms (Ω). The higher the number of ohms, the more resistance; their most common use is to protect LEDs from drawing too much current and damaging themselves or the Pi. Our first physical computing program: Hello, LED!, making an LED light up, needs an LED and a 330 ohm resistor, or as close to 330 Ω as you can find (the exact value of resistor you need depends on the LED you're using, but 330 Ω works for most common LEDs). It protects your Raspberry Pi and the LED by limiting the amount of electrical current the LED can draw. Without it,

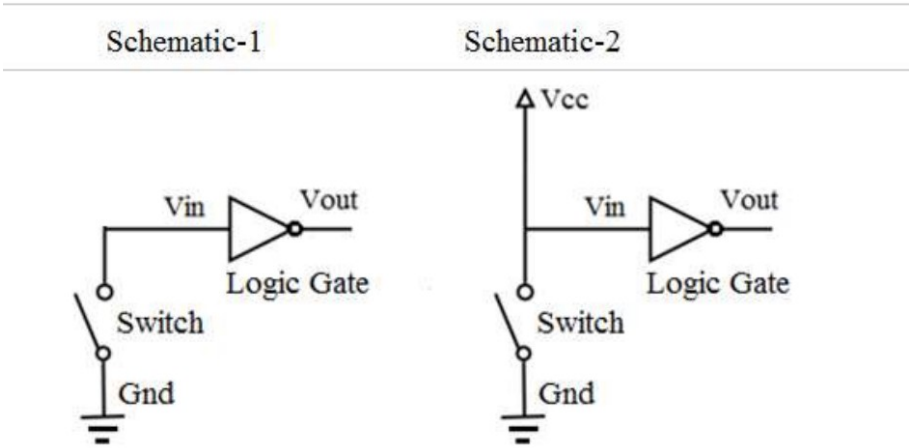
the LED can pull too much current and burn itself – or the Pi. Such resistors are called as current-limiting resistors. The higher the value, the dimmer the LED; the lower the value, the brighter the LED. Never connect an LED to a Raspberry Pi without a current limiting resistor

The voltage drop across an LED depends entirely on the amount of current flowing through it and ranges from 2volts to 4v for most LEDs. Commonly used LEDs have cut-off-voltage of 1.7v and current of 10mA; when an LED is provided with required voltage and current it glows with full intensity. LEDs are very sensitive components and has very low internal resistance; hence over current or voltage can burn up. Typically the over current or voltage is controlled or limited by an external resistor (value 10K) in series with LED.

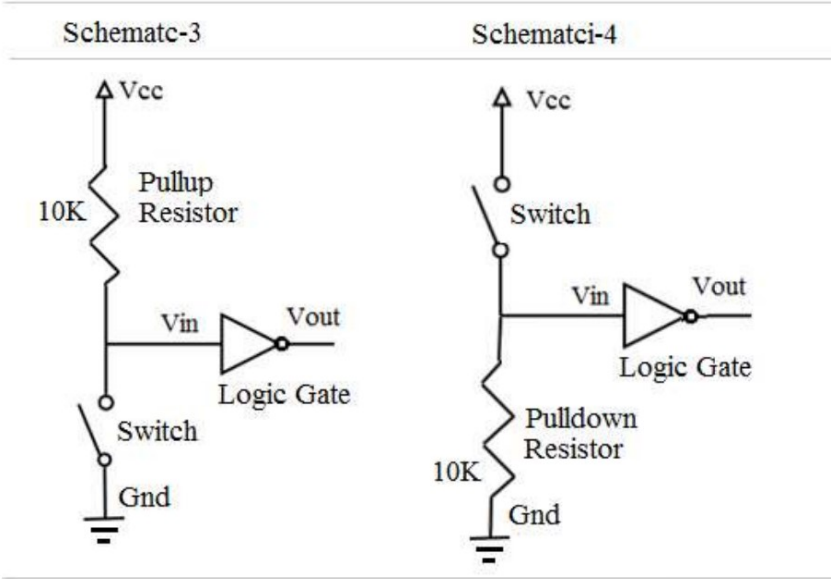
Switches: A switch is an electrical component that can open (break) connection) or close (establish) connection in a circuit. There are a large variety of switches available, but a simple switch has two contacts; which are either open (off) or closed (on). The processor needs a definite high or low voltage level at its digital input; hence a switch connected to the input pin requires either a pullup or pulldown resistor. The pullup resistor is placed between a digital input and the Vcc that pulls the input pin up to the source voltage; the pulldown resistor is placed between a digital input and the Gnd which pulls the input pin down to the Gnd voltage.

Why Pullups and Pulldowns: The pullups or pulldowns are nothing but resistors; and are most common in digital circuits to limit the amount of current flowing through the circuit. In fact it prevents input lines from floating or prevents much current flow; if there is no input, the circuit assumes a default value either Vcc or Gnd. The below shematic-1 shows a gate input which is connected to the ground through a switch. The input state of most of the logic gates are called high impedance state. When the switch is closed, since there is a definite connection to ground the input pin goes low and the state of the pin

become stable. If nothing is connected to the input pin then the value of the input is floating.



In the above schematic diagram-2 the input pin is tied to Vcc; when the switch is open it does not float instead it goes to high state. But when the switch is closed, it shorts Vcc to the Gnd; there is a direct connection between Vcc and Gnd. As per the Ohm's law if R=0 then I=Infinity; of course there is more current flow from Vcc to Gnd; which burns the device or wires and causes a short circuit. To avoid this, we introduce a pullup or pulldown resister as shown in the figure below.



In the shematic-3 on the left side, input pin (Vin) is tied to Vcc through a pull-up resistor. The pullup resistor can limit the amount of current flowing through the circuit. If the switch is open input pin is pulled up to Vcc; otherwise (if closed) the input pin has a direct connection to Gnd through the pullup, therefor there is a current flow from Vcc, through pullup, and to ground. This is considered a good circuit. Here a 10K resistor is used as a pullup; in fact, you can compute the current using Ohms law I=V/R.

Vcc = 5v and R = 10K

I = V / R

I = 5v / 10,000 ohms

I = .0005A (.5mA)

If Vcc = 3.3v then I = 0.33mA

The schematic diagram-4 shown on the right side with a pulldown that pulls down the input to Gnd when the circuit is open; Just like pullups pulldowns are also used to limit the current but it is less often used.