14ITO01 – Internet of Things

Unit IV – IoT Devices (Raspberry Pi)

By
Mr.S.Selvaraj
Asst. Professor(SRG) / CSE
Kongu Engineering College

OUTLINE

- Introduction to Raspberry PI
- Interfaces (Serial,SPI,I2C) Programming
- Python programming with Raspberry PI with focus of
  - Interfacing external gadgets
  - Controlling output
  - Reading input from pins
- Connecting IoT to Cloud

Introduction to Raspberry Pi

- What is an IoT Device?
- Basic building blocks of an IoT Device
- Exemplary Device: Raspberry Pi
- Raspberry Pi interfaces
- Programming Raspberry Pi with Python
- Other IoT devices

What is an IoT Device?

- A "Thing" in Internet of Things (IoT) can be any object that has a unique identifier and which can send/receive data (including user data) over a network (e.g., smart phone, smart TV, computer, refrigerator, car, etc.).

- IoT devices are connected to the Internet and send information about themselves or about their surroundings (e.g. information sensed by the connected sensors) over a network (to other devices or servers/storage) or allow actuation upon the physical entities/environment around them remotely

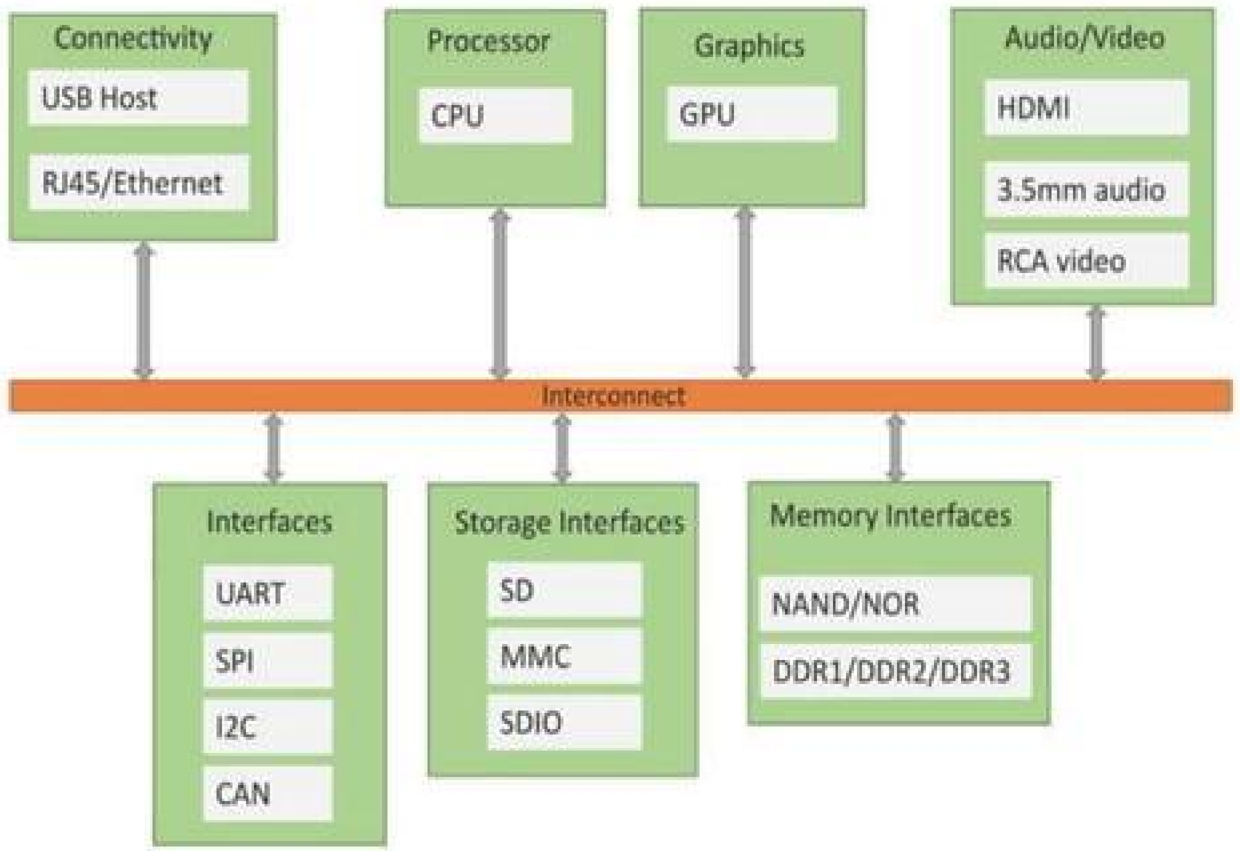
IoT Device Examples

- A home automation device that allows remotely monitoring the status of appliances and controlling the appliances.
- An industrial machine which sends information abouts its operation and health monitoring data to a server.
- A car which sends information about its location to a cloud-based service.
- A wireless-enabled wearable device that measures data about a person such as the number of steps walked and sends the data to a cloud-based service.

Basic building blocks of an IoT Device

- Sensing
  - Sensors can be either on-board the IoT device or attached to the device.
- Actuation
  - IoT devices can have various types of actuators attached that allow taking
  - actions upon the physical entities in the vicinity of the device.
- Communication
  - Communication modules are responsible for sending collected data to other devices or cloud-based servers/storage and receiving data from other devices and commands from remote applications.
- Analysis & Processing
  - Analysis and processing modules are responsible for making sense of the collected data

Block diagram of an IoT Device



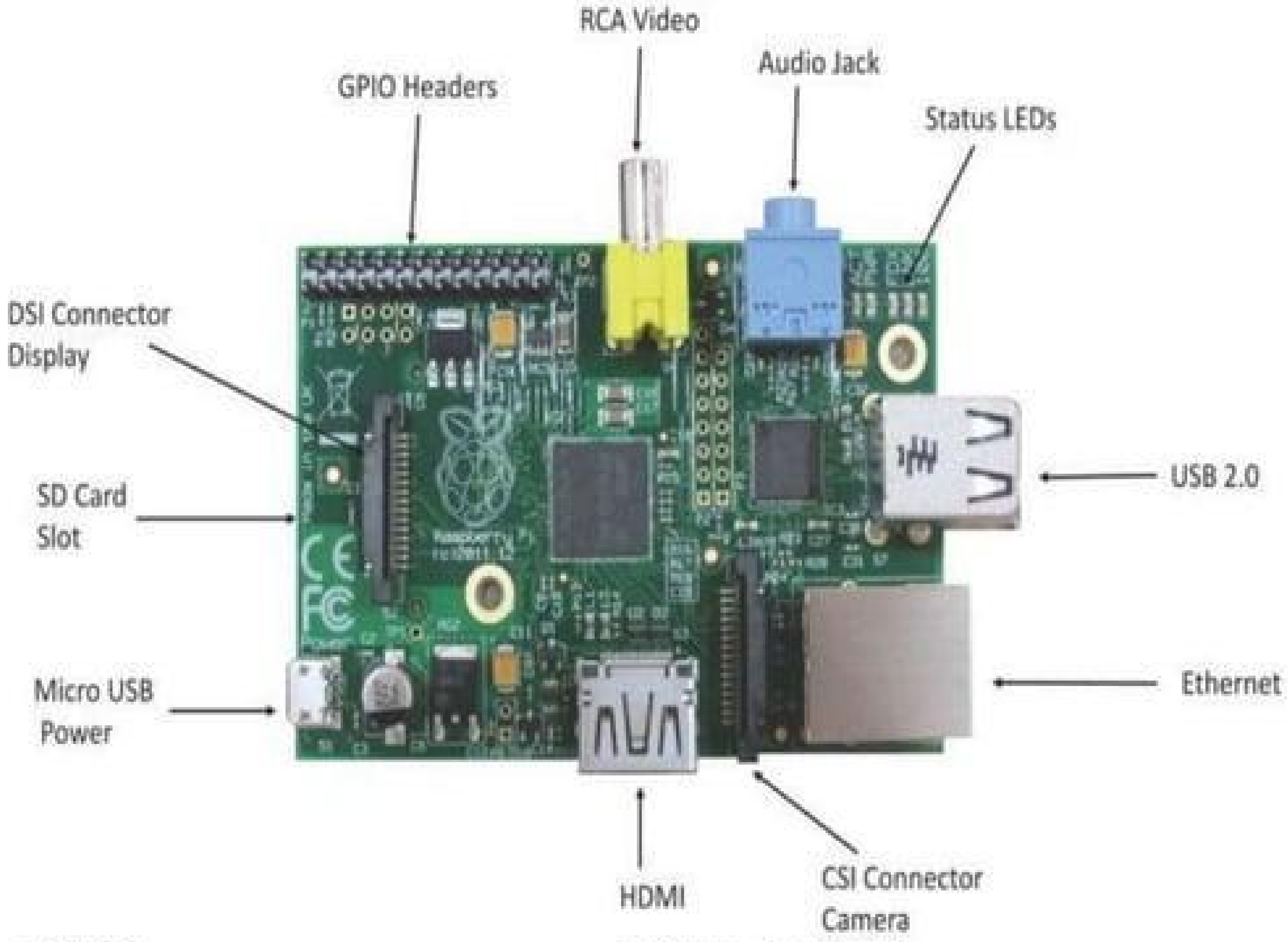
Exemplary Device: Raspberry Pi

- Raspberry Pi is a low-cost mini-computer with the physical size of a credit card.
- Raspberry Pi runs various flavors of Linux and can perform almost all tasks that a normal desktop computer can do.
- Raspberry Pi also allows interfacing sensors and actuators through the general purpose I/O pins.
- Since Raspberry Pi runs Linux operating system, it supports Python "out of the box"

Raspberry Pi Status leds

STATUS LED	FUNCTION
ACT	SD card access
PWR	3.3V Power is present
FDX	Full Duplex LAN connected
LNK	Link/Network activity
100	100 Mbit LAN connected

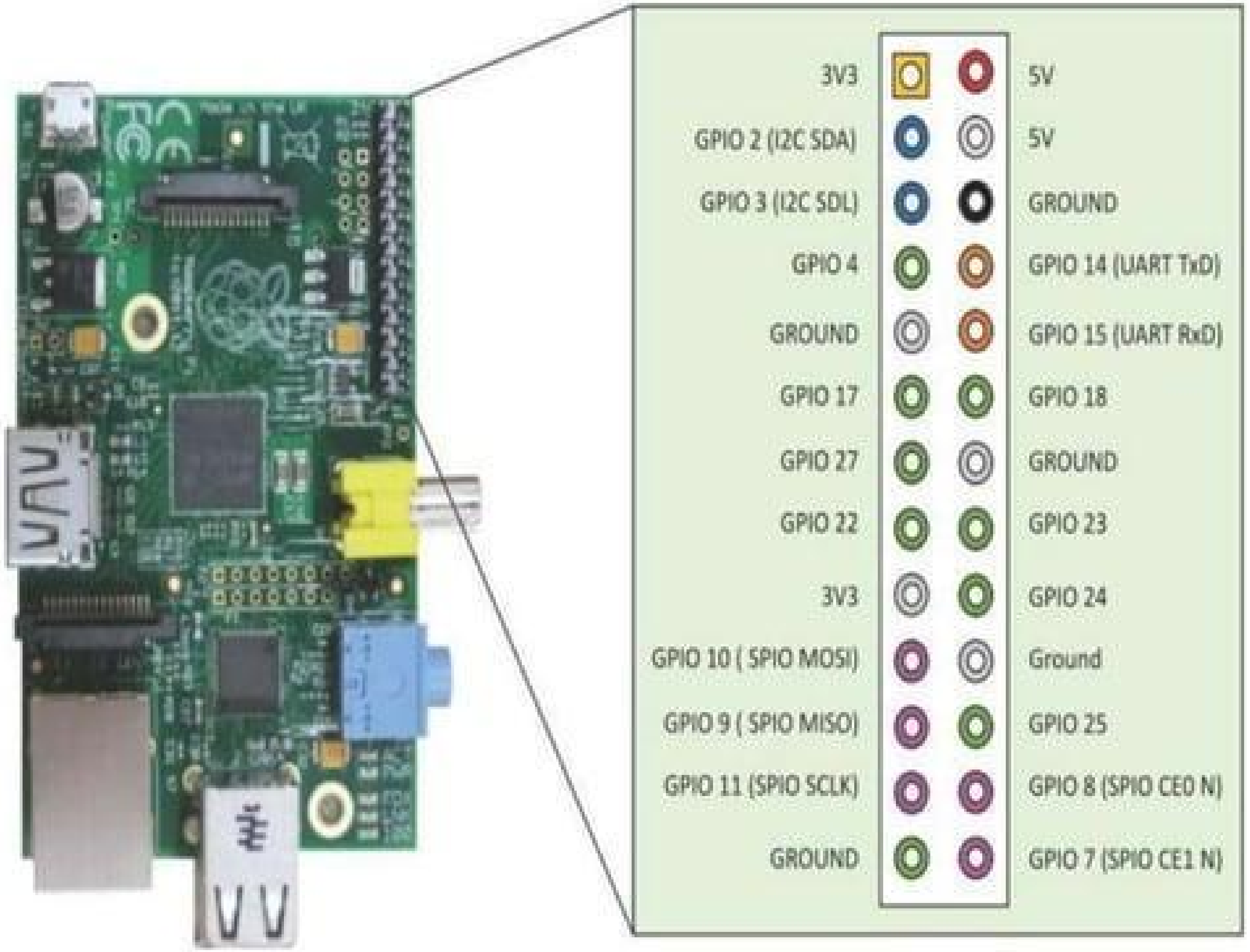
Raspberry Pi Board



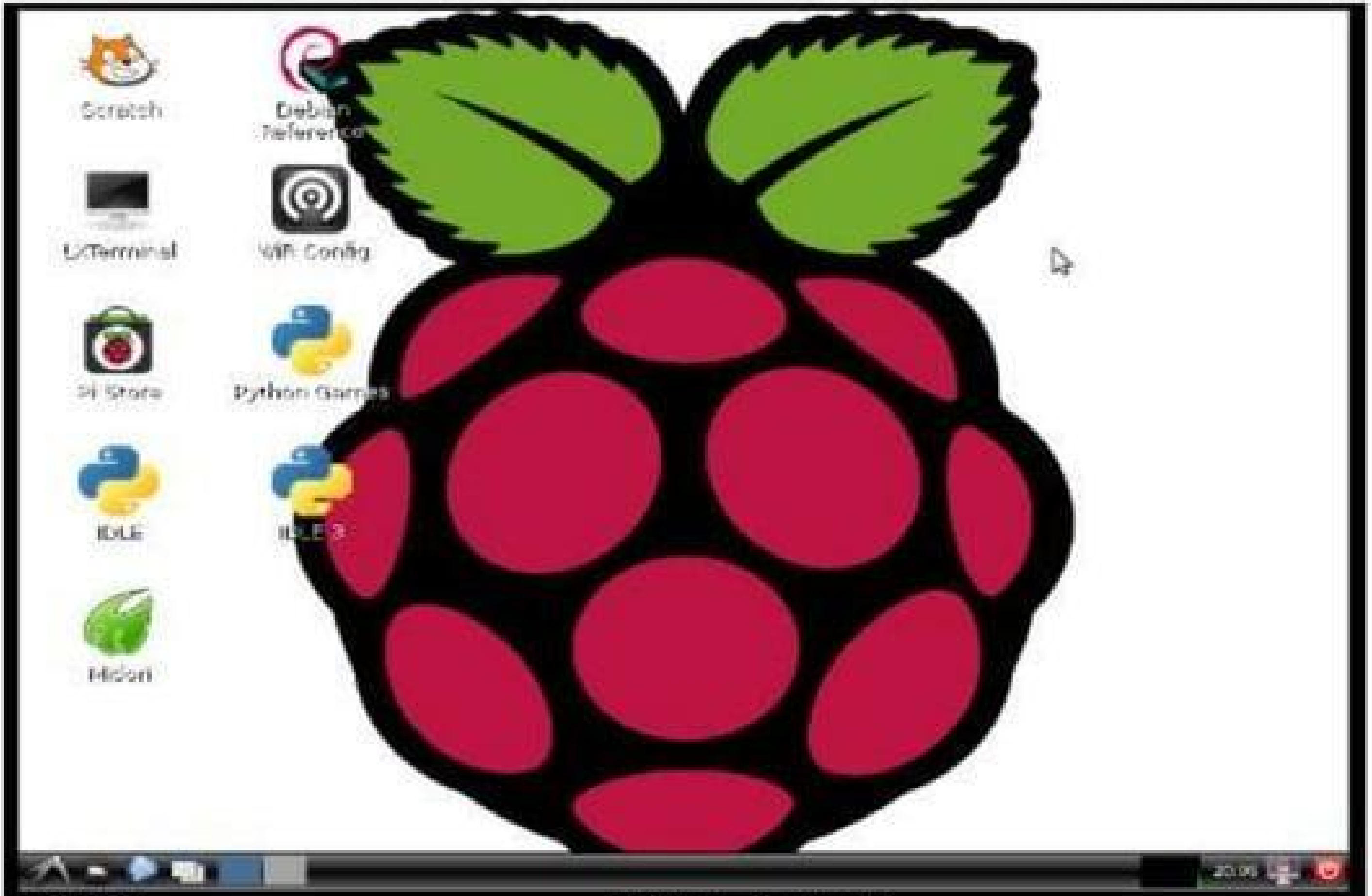
Linux on Raspberry Pi

- Raspbian
  - Raspbian Linux is a Debian Wheezy port optimized for Raspberry Pi.
- Arch
  - Arch is an Arch Linux port for AMD devices.
- Pidora
  - Pidora Linux is a Fedora Linux optimized for Raspberry Pi.
- RaspBMC
  - RaspBMC is an XBMC media-center distribution for Raspberry Pi.
- OpenELEC
  - OpenELEC is a fast and user-friendly XBMC media-center distribution.
- RISC OS
  - RISC OS is a very fast and compact operating system

Raspberry Pi GPIO headers



Rasbian Linux Desktop



Raspberry Pi frequently used commands

Command	Function	Example
cd	change directory	cd/home/pi
cat	show file contents	cat file.txt
ls	list files and folders	ls/home/pi
locate	search for a file	locate file.txt
lsusb	list usb devices	lsusb
pwd	print name for present working directory	pwd
mkdir	make directory	mkdir/home/pi/new
mv	move(rename) file	mv sourcefile.txt destfile.txt
rm	remove file	rm file.txt
reboot	reboot device	sudo reboot
shutdown	shutdown device	sudo shutdown –h now

Raspberry Pi frequently used commands

Command	Function	Example
grep	Print lines matching a pattern	grep –r "pi"/home/
df	Report file system disk space usage	df -Th
ipconfig	Configure a network interface	ipconfig
netstat	Print network connections, routing tables, interface statistics	Netstat -lntp
tar	Extract /create archive	Tar –xzf foo.tar.gz
wget	Non-interactive network downloader	Wget http://example.com/filr.tar.gz

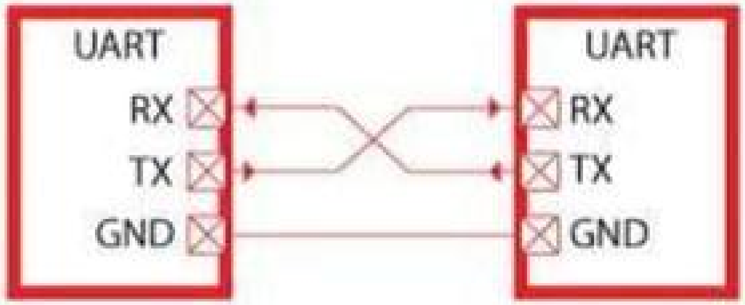
Raspberry Pi Interfaces

- Serial Interface / UART Interface
  - Universal Asynchronous Receiver and Transmitter(UART)
- SPI
  - Serial Peripheral Interface (SPI)
- I2C
  - Inter-Integrated Circuits (I2C)

Raspberry Pi Interfaces

- Serial
  - The serial interface on Raspberry Pi has receive (Rx) and transmit (Tx) pins for communication with serial peripherals.
- SPI
  - Serial Peripheral Interface (SPI) is a synchronous serial data protocol used for communicating with one or more peripheral devices.
- I2C
  - The I2C interface pins on Raspberry Pi allow you to connect hardware modules.
  - I2C interface allows synchronous data transfer with just two pins - SDA (data line) and SCL (clock line).

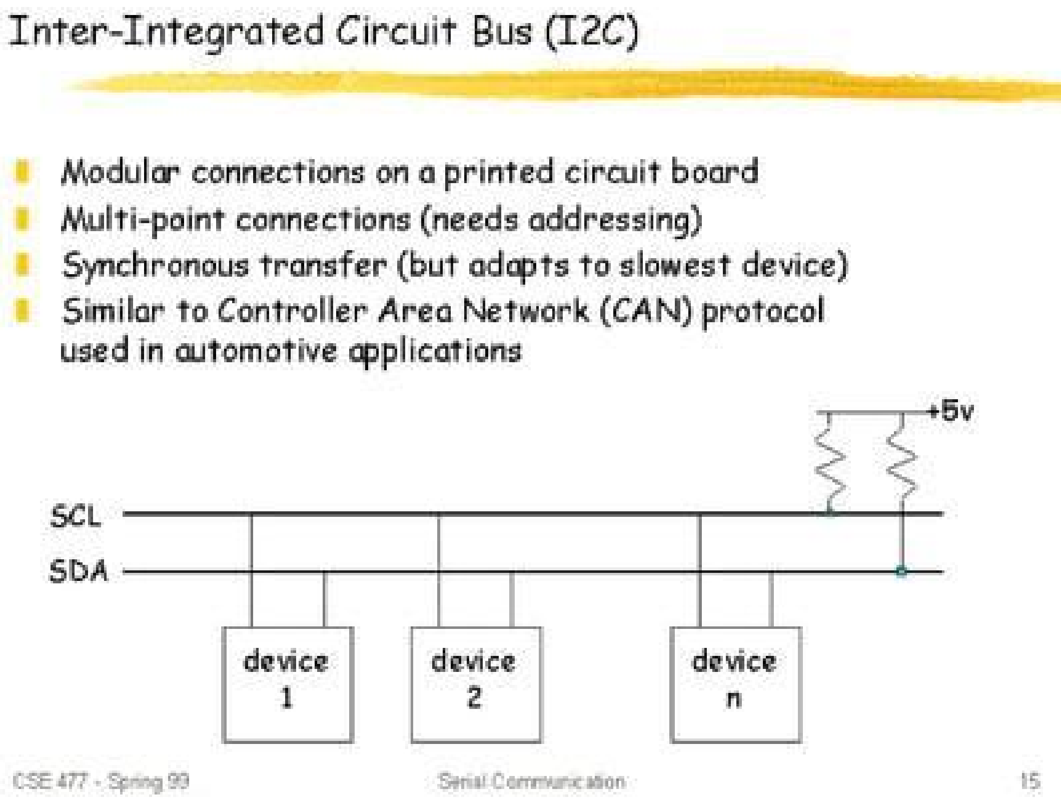
Serial / UART



SPI

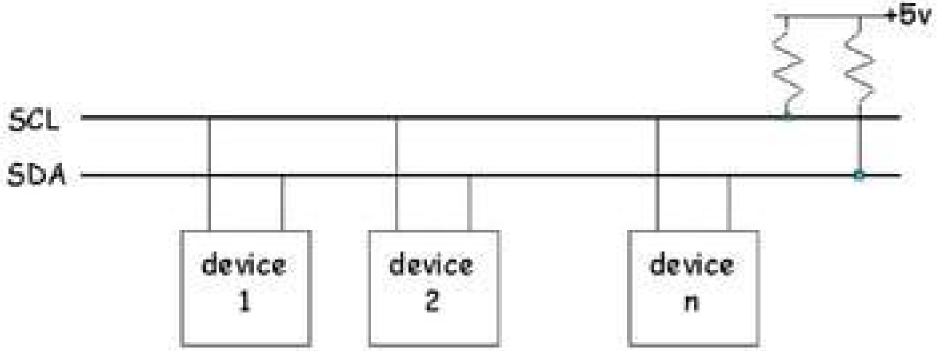
- In SPI Connection, there is one master device and one or more peripheral devices.
- There are five pins on Raspberry Pi for SPI interface.
  - MISO (Master In Slave Out) – Master Line for Sending Data to the peripherals.
  - MOSI (Master Out Slave In) – Slave Line for sending data to the master.
  - SCK (Serial Clock) – Clock Generated by Master to Synchronize data transmission.
  - CE0 (Chip Enable 0) – To Enable or Disable devices.
  - CE1 (Chip Enable 1) – To Enable or Disable devices.

I2C



Inter-Integrated Circuit Bus (I2C)

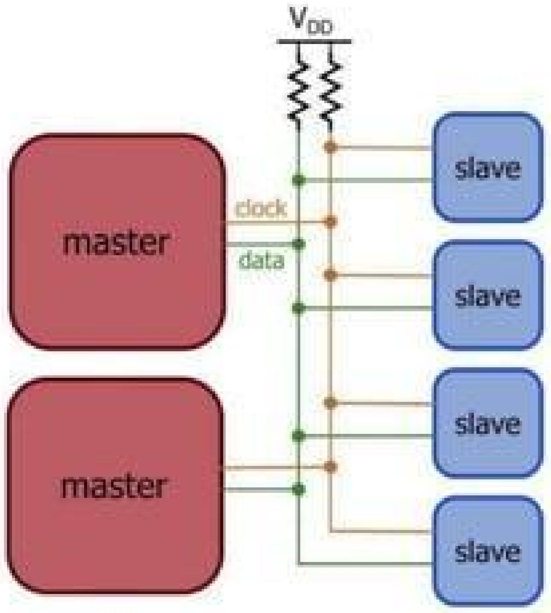
- Modular connections on a printed circuit board
- Multi-point connections (needs addressing)
- Synchronous transfer (but adapts to slowest device)
- Similar to Controller Area Network (CAN) protocol used in automotive applications



INTER-INTEGRATED CIRCUIT (I^2C)

- **Serial, 8-bit oriented, Bidirectional Half Duplex, 2-wire bus** used for communications between integrated circuits on the same PCB.
- Developed by **Philips** Semiconductors in 1982 (now **NXP** Semiconductors).
- Only two bus lines are required: a **serial data line (SDA)** and a **serial clock line (SCL)**.
- Master device **generates** the clock signal and **terminates** the transfer.
- Each device is **software addressable by a unique address.**
- Master/Slave relationships exist at all times; masters can operate as **master-transmitters** or as **master-receivers.**
- **True multi-master** bus including **collision detection, arbitration** (using wired-AND) and **clock synchronization** to prevent data corruption.

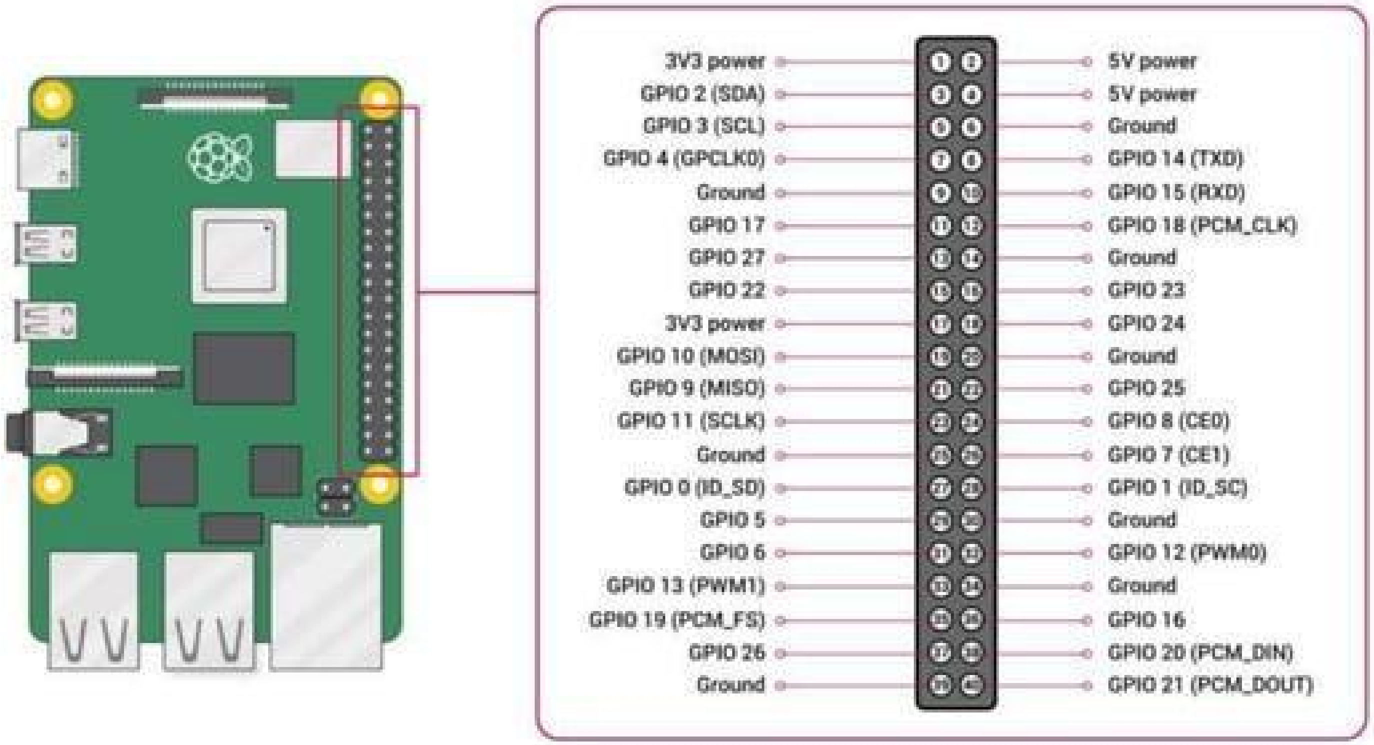
Source: https://www.allaboutcircuits.com/technical-articles/introduction-to-the-i2c-bus/
Source: UM10204, I²C-bus specification and user manual, Rev. 6 — 4 April 2014, User manual



Raspberry Pi Pin Configurations

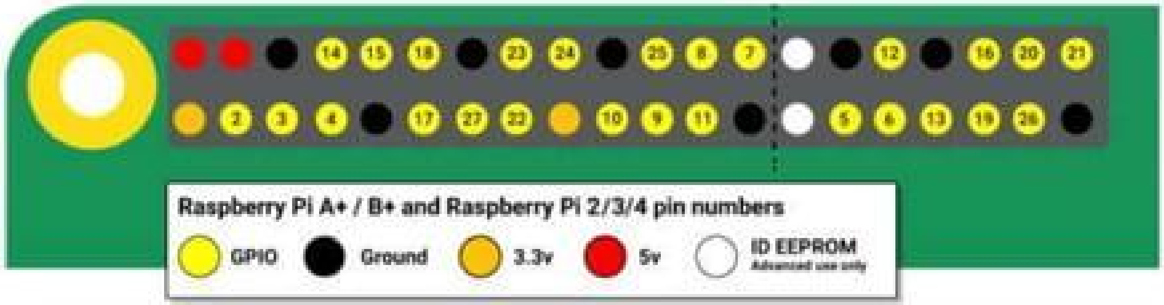
- A powerful feature of the Raspberry Pi is the row of GPIO (general-purpose input/output) pins along the top edge of the board.
- A **40-pin** GPIO header is found on all current Raspberry Pi boards (unpopulated on Pi Zero and Pi Zero W).
- Prior to the Pi 1 Model B+ (2014), boards comprised a shorter **26-pin** header.

Raspberry Pi Pin Configuration



Raspberry Pi Pin Configuration

- Any of the GPIO pins can be designated (in software) as an input or output pin and used for a wide range of purposes.
- **Note: the numbering of the GPIO pins is not in numerical order; GPIO pins 0 and 1 are present on the board (physical pins 27 and 28) but are reserved for advanced use.**



Raspberry Pi Pin Configuration

- Voltages
  - Two 5V pins and two 3V3 pins are present on the board, as well as a number of ground pins (0V), which are unconfigurable.
  - The remaining pins are all general purpose 3V3 pins, meaning outputs are set to 3V3 and inputs are 3V3-tolerant.
- Outputs
  - A GPIO pin designated as an output pin can be set to high (3V3) or low (0V).
- Inputs
  - A GPIO pin designated as an input pin can be read as high (3V3) or low (0V).
  - This is made easier with the use of internal pull-up or pull-down resistors.
  - Pins GPIO2 and GPIO3 have fixed pull-up resistors, but for other pins this can be configured in software.

Raspberry Pi Pin Configuration

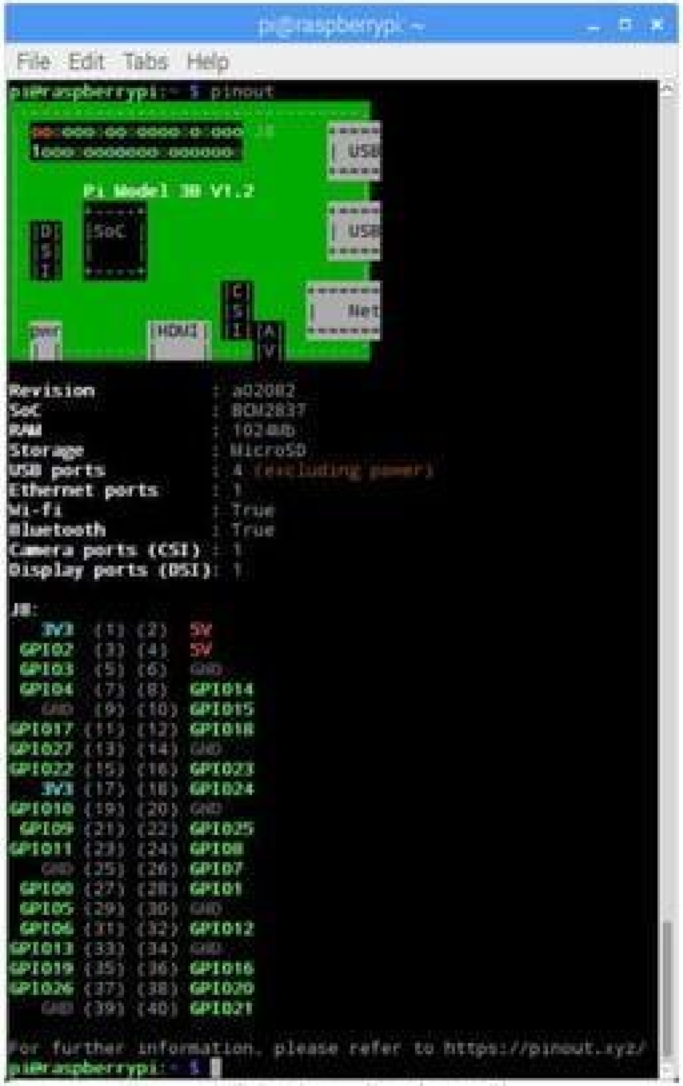
- As well as simple input and output devices, the GPIO pins can be used with a variety of alternative functions, some are available on all pins, others on specific pins.
- PWM (pulse-width modulation)
  - Software PWM available on all pins
  - Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19
- SPI
  - SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)
  - SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)
- I2C
  - Data: (GPIO2); Clock (GPIO3)
  - EEPROM Data: (GPIO0); EEPROM Clock (GPIO1)
- Serial
  - TX (GPIO14); RX (GPIO15)

Raspberry Pi Pin Configuration

- GPIO pinout
  - It's important to be aware of which pin is which.
  - Some people use pin labels (like
    - the RasPiO Portsplus PCB, or
    - the printabl

Raspberry Pi Pin Configuration

- A handy reference can be accessed on the Raspberry Pi by opening a terminal window and running the command **pinout**.
- This tool is provided by the GPIO Zero Python library, which is installed by default on the Raspberry Pi OS desktop image, but not on Raspberry Pi OS Lite.



```
pi@raspberrypi:~ $ pinout
  oo:ooo:oo:oooo:o:ooo J8
  1ooo:ooooooo:oooooo                   | USB
        Pi Model 3B V1.2
  |D|  |SoC |                           | USB
  |S|  |    |
  |I|
                          |C|           | Net
                          |S|
  |pwr|     |HDMI|        |I||A|
                             |V|

Revision           : a02082
SoC                : BCM2837
RAM                : 1024Mb
Storage            : MicroSD
USB ports          : 4 (excluding power)
Ethernet ports     : 1
Wi-fi              : True
Bluetooth          : True
Camera ports (CSI) : 1
Display ports (DSI): 1

J8:
   3V3  (1) (2)  5V
 GPIO2  (3) (4)  5V
 GPIO3  (5) (6)  GND
 GPIO4  (7) (8)  GPIO14
   GND  (9) (10) GPIO15
GPIO17 (11) (12) GPIO18
GPIO27 (13) (14) GND
GPIO22 (15) (16) GPIO23
   3V3 (17) (18) GPIO24
GPIO10 (19) (20) GND
 GPIO9 (21) (22) GPIO25
GPIO11 (23) (24) GPIO8
   GND (25) (26) GPIO7
 GPIO0 (27) (28) GPIO1
 GPIO5 (29) (30) GND
 GPIO6 (31) (32) GPIO12
GPIO13 (33) (34) GND
GPIO19 (35) (36) GPIO16
GPIO26 (37) (38) GPIO20
   GND (39) (40) GPIO21

For further information, please refer to https://pinout.xyz/
pi@raspberrypi:~ $
```

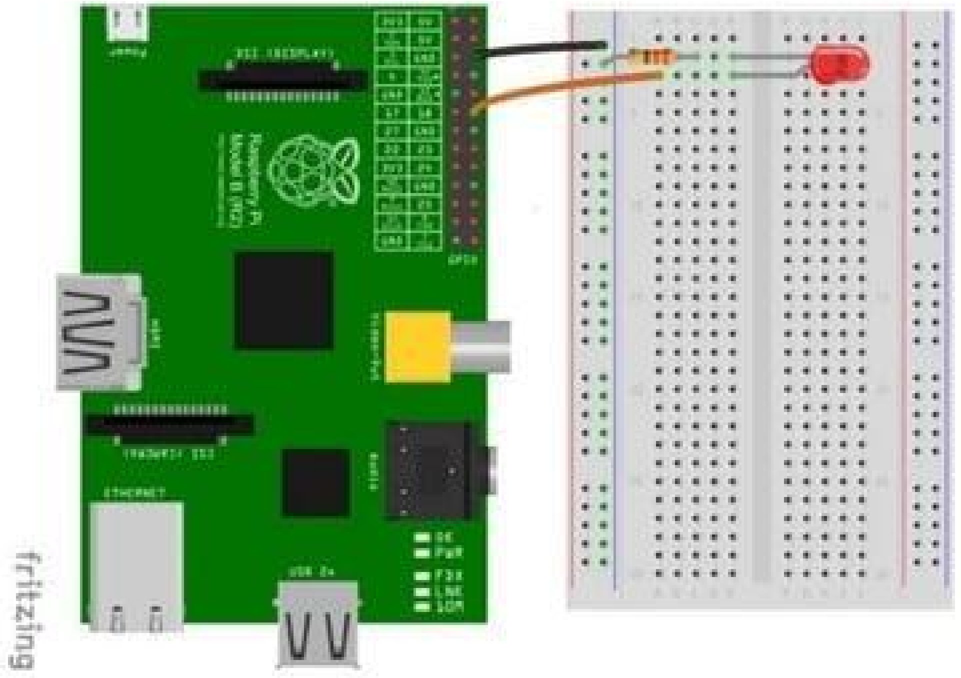
Python programming with Raspberry PI with focus of

- Interfacing external gadgets
- Controlling output
- Reading input from pins

Python Programming with Raspberry Pi

- Python programming with Raspberry PI with focus of
  - Interfacing external gadgets
  - Controlling output
  - Reading input from pins
- GPIO pins on Raspberry Pi that makes it useful device for IoT.
- We can interface a wide variety of sensors and actuators with Raspberry Pi using the GPIO pins and SPI, I2C and Serial Interfaces.
- Input from the sensors connected to Raspberry Pi can be processed and various actions can be taken, for
  - Instance,
  - Sending data to a server
  - Sending an email
  - Triggering a relay switch

Controlling LED with Raspberry Pi

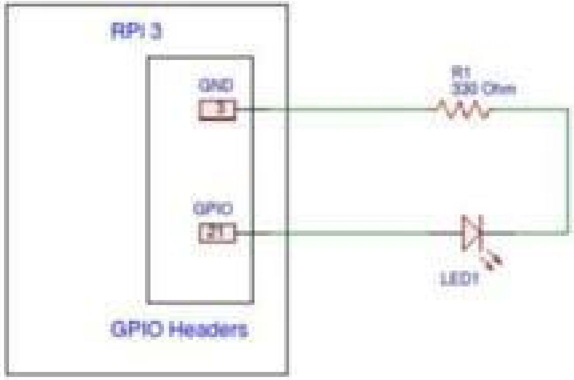


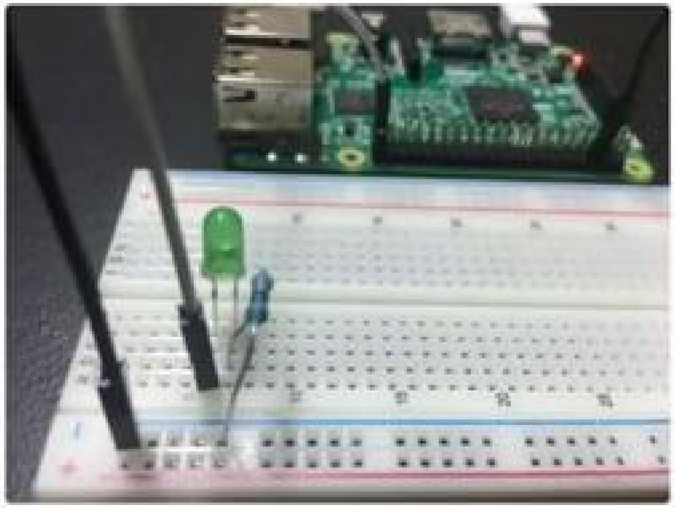
Components Reuired

- You'll need the following components to connect the circuit.
- 1. Raspberry Pi
  2. LED
  3. Resistor - 330 ohm
  4. Breadboard
  5. 2 Male-Female Jumper Wires

Raspberry Pi 3 Model B V1.2
© Raspberry Pi 2015
HDMI

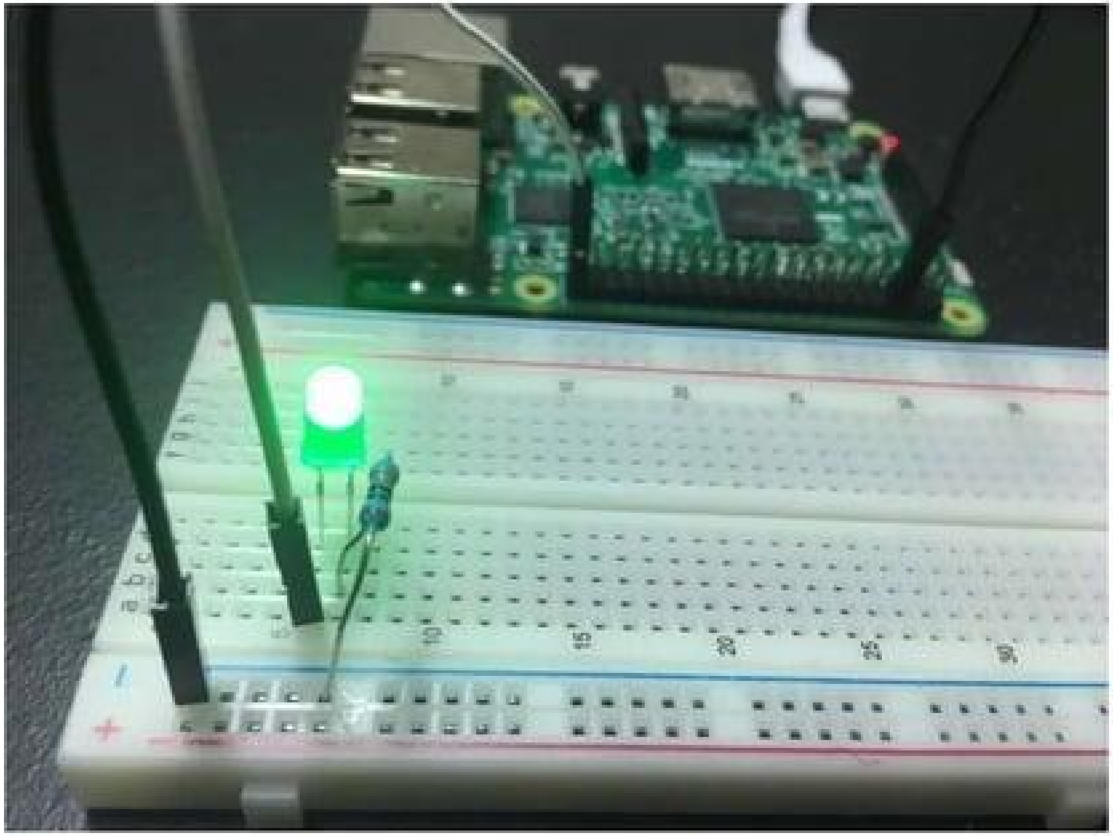
Step 2: Connecting the Circuit





Python Code

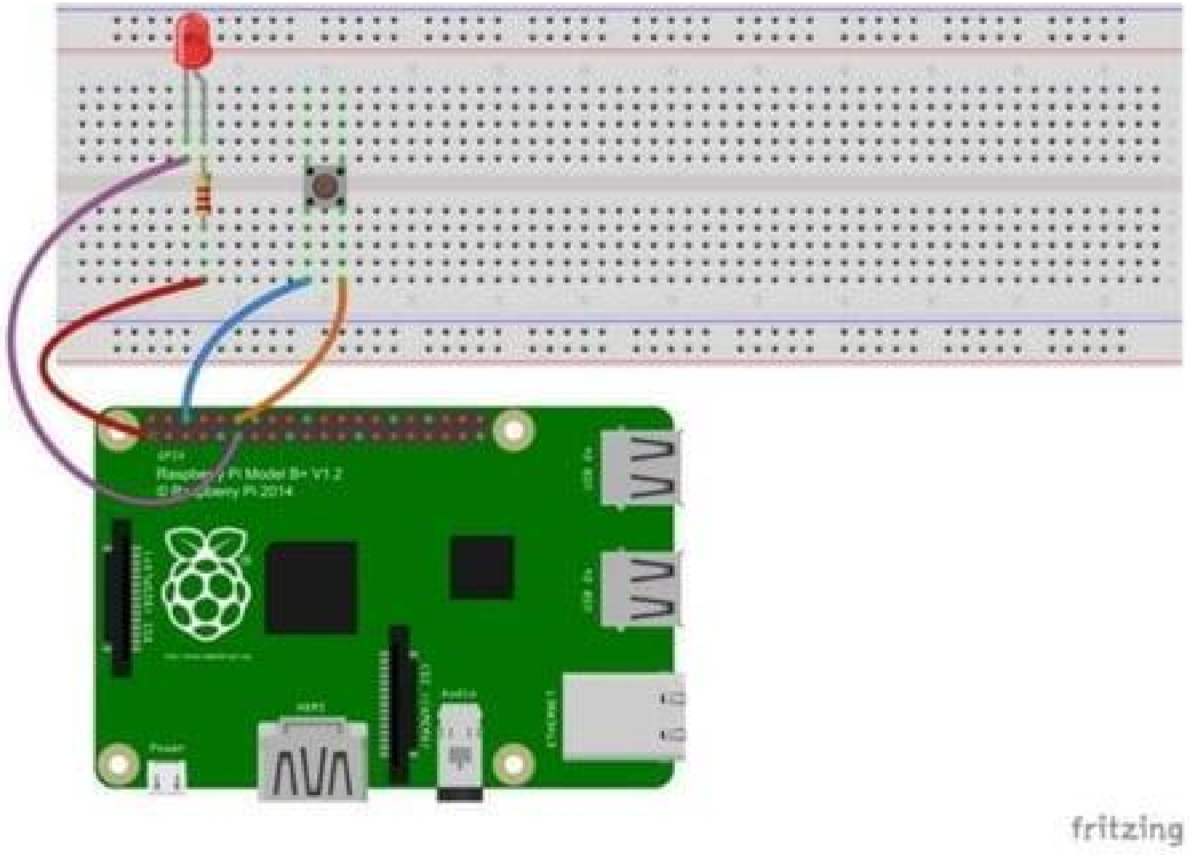
-
  ```
  import RPi.GPIO as GPIO
  import time
  GPIO.setmode(GPIO.BCM)
  GPIO.setwarnings(False)
  GPIO.setup(21,GPIO.OUT)
  print "LED on"
  GPIO.output(21,GPIO.HIGH)
  time.sleep(10)
  print "LED off"
  GPIO.output(21,GPIO.LOW)
  ```



Python programming with Raspberry PI with focus of

- Interfacing external gadgets
- Controlling output
- Reading input from pins

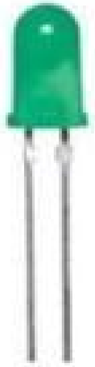
Controlling LED with Switch on Raspberry Pi



Components Reuired

- You'll need the following components to connect the circuit.
- 1. Raspberry Pi
  2. LED
  3. Resistor - 330 ohm
  4. Breadboard
  5. Button / Switch
  6. 2 Male-Female Jumper Wires

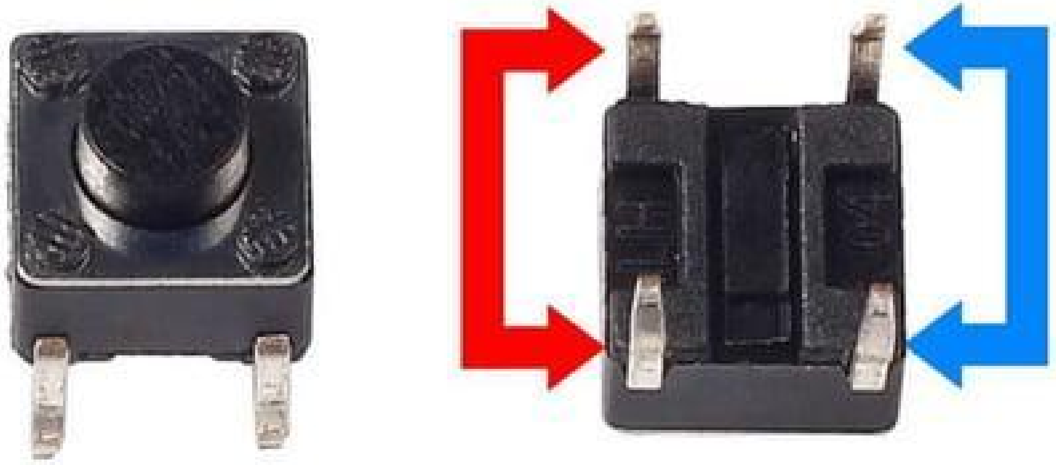
Components



Switch / Button

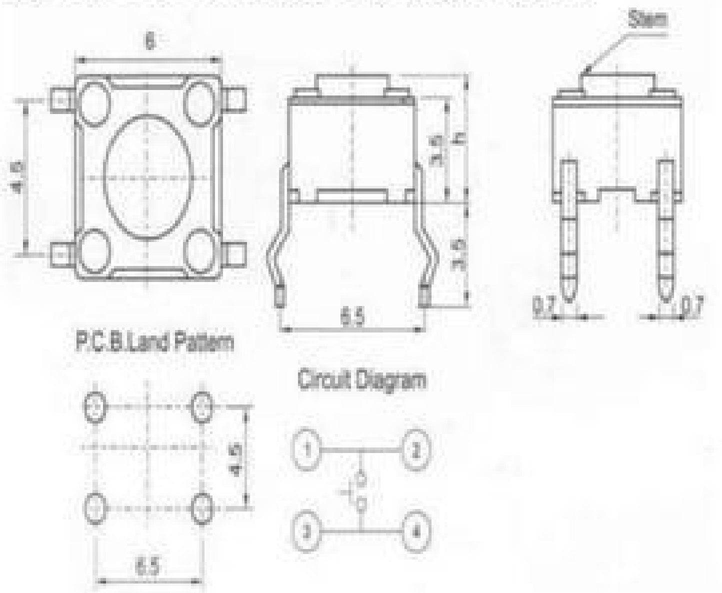
- Buttons are a common component used to control electronic devices.
- They are usually used as switches to connect or disconnect circuits.
- Although buttons come in a variety of sizes and shapes, the one used here is a 6mm mini-button as shown in the following pictures.
- Pins pointed out by the arrows of same color are meant to be connected.

Switch / Button



Switch / Button

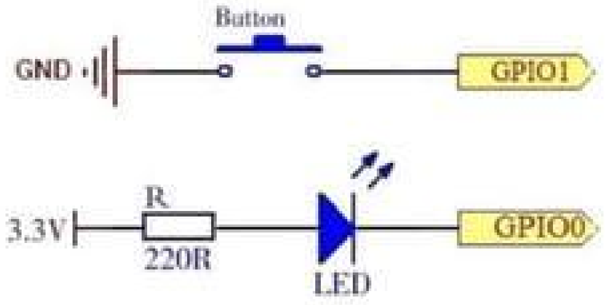
- When the button is pressed, the pins pointed by the blue arrow will connect to the pins pointed by the red arrow (see the above figure), thus closing the circuit, as shown in the following diagrams.



Switch / Button

- Generally, the button can be connected directly to the LED in a circuit to turn on or off the LED, which is comparatively simple.
- However, sometimes the LED will brighten automatically without any button pressed, which is caused by various kinds of external interference.
- In order to avoid this interference, a pull-down resistor is used – usually connect a 1K–10KΩ resistor between the button and GND. It can be connected to GND to consume the interference when the button is off.
- Use a normally open button as the input of Raspberry Pi.
- When the button is pressed, the GPIO connected to the button will turn into low level (0V).
- We can detect the state of the GPIO connected to the button through programming. That is, if the GPIO turns into low level, it means the button is pressed. You can run the corresponding code when the button is pressed, and then the LED will light up.

CIRCUIT Diagram



Experimental Procedures

- **Step 1**: Build the circuit
- **Step 2:** Create a Python / C code
- **Step 3:** Run

```
sudo python 02_btnAndLed.py
```

- Now, press the button, and the LED will light up; press the button again, and the LED will go out. At the same time, the state of the LED will be printed on the screen.

Python Code

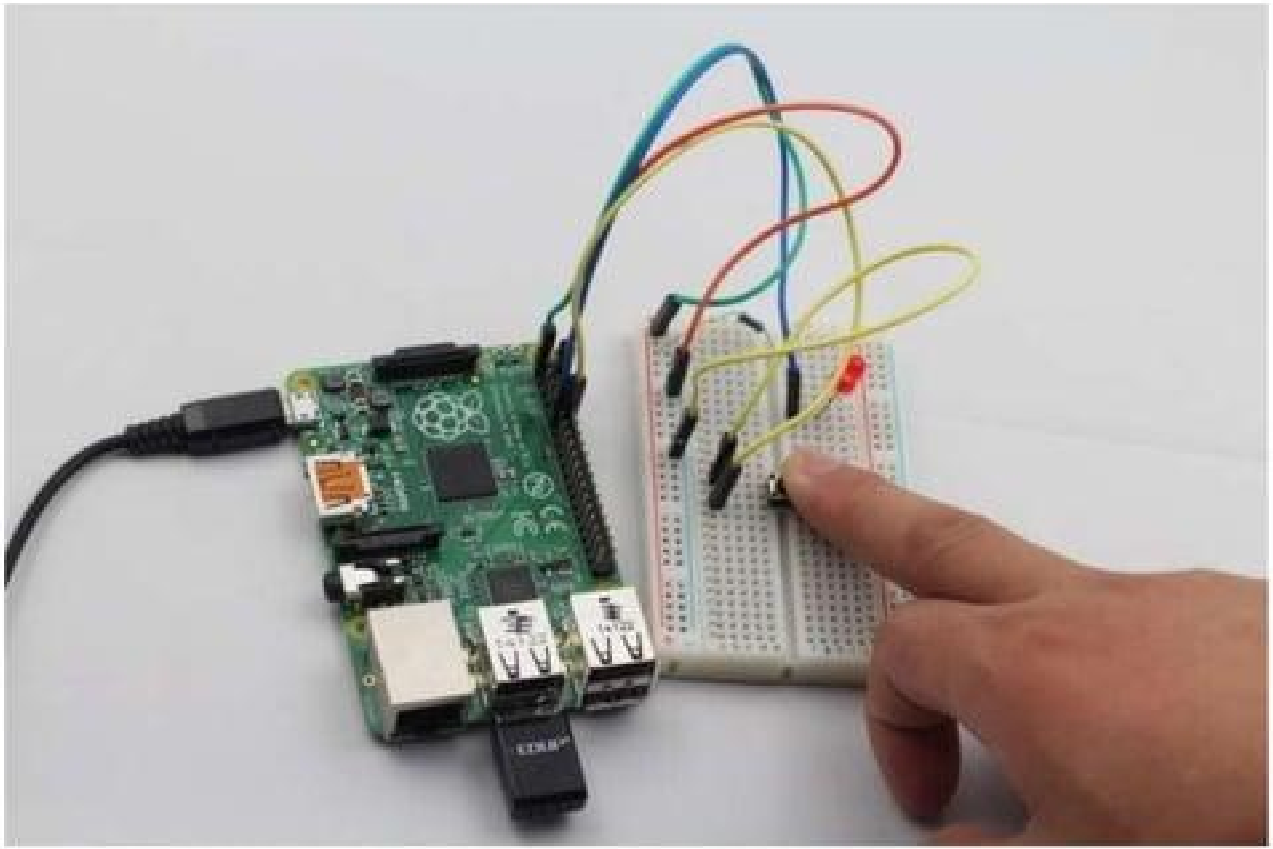
- #!/usr/bin/env python
- import RPi.GPIO as GPIO
- import time
- LedPin = 11 # pin11 --- led
- BtnPin = 12 # pin12 --- button
- Led_status = 1
- def setup():
  - GPIO.setmode(GPIO.BOARD) # Numbers GPIOs by physical location
  - GPIO.setup(LedPin, GPIO.OUT) # Set LedPin's mode is output
  - GPIO.setup(BtnPin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Set BtnPin's mode is input, and pull up to high level(3.3V)
  - GPIO.output(LedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to off led

- def swLed(ev=None):
  - global Led_status
  - Led_status = not Led_status
  - GPIO.output(LedPin, Led_status) # switch led status(on-->off; off-->on)
  - if Led_status == 1:
    - print 'led off...'
  - else:
    - print '...led on'

Python Code (Contd.,)

- def loop():
  - GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed, bouncetime=200) # wait for falling and set bouncetime to prevent the callback function from being called multiple times when the button is pressed
  - while True:
    - time.sleep(1) # Don't do anything
- def destroy():
  - GPIO.output(LedPin, GPIO.HIGH) # led off
  - GPIO.cleanup() # Release resource
- if __name__ == '__main__': # Program start from here
  - setup()
  - try:
    - loop()
  - except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child program destroy() will be executed.
    - destroy()

Interfacing



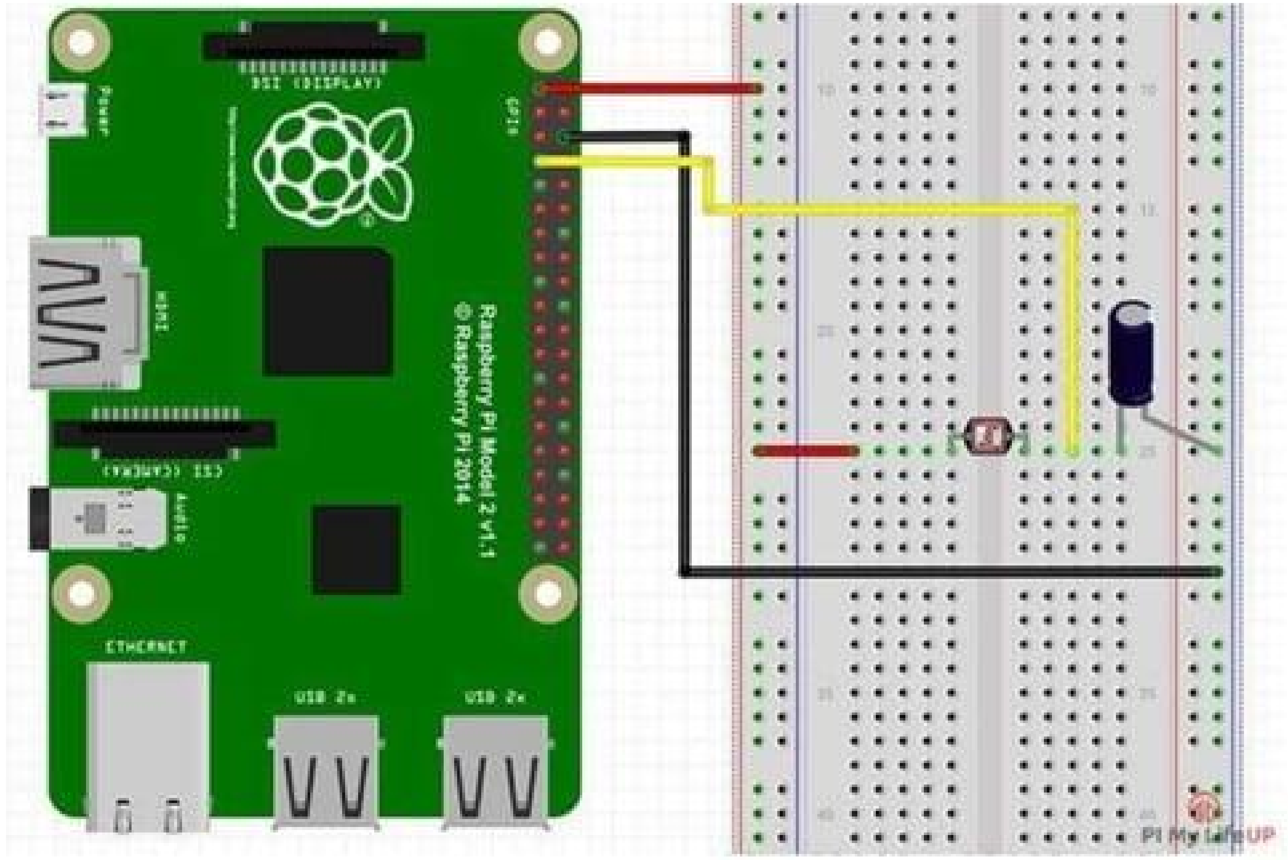
Applications

- Interfacing a Push Button with Raspberry Pi might not seem as a big project but it definitely helps us in understanding the concept of reading from Input pins.
- A similar concept can be applied to other input devices like different types of Sensors
  - PIR Sensor,
  - Ultrasonic Sensor,
  - Touch Sensor, and so forth.

Python programming with Raspberry PI with focus of

- Interfacing external gadgets
- Controlling output
- Reading input from pins

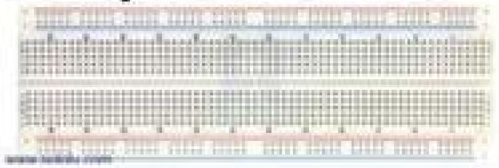
Interfacing a Light Sensor (LDR) with Raspberry Pi



Components Reuired

- You'll need the following components to connect the circuit.
- 1. Raspberry Pi
  2. Light Sensor (LDR Sensor)
  3. Capacitor - 1µF
  4. Breadboard
  5. 2 Male-Female Jumper Wires

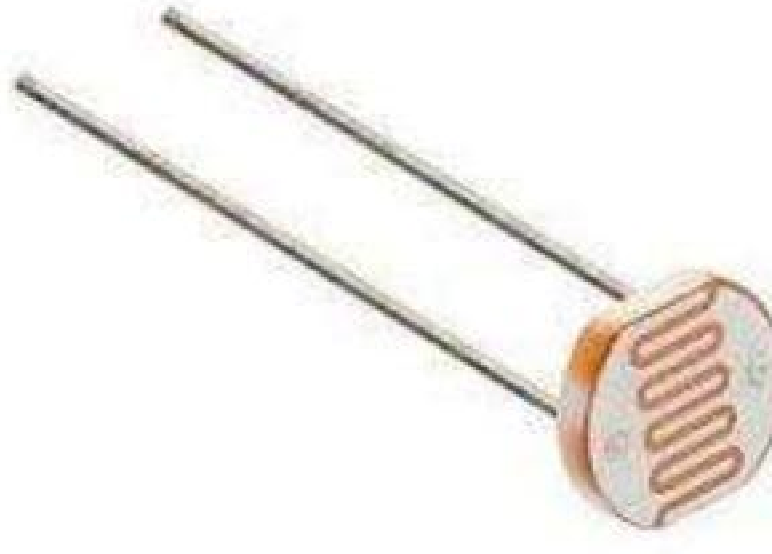
Components



LDR Sensor

- The **light-dependent resistor** or also known as the LDR sensor is the most important piece of equipment in our circuit. Without it, we wouldn't be able to detect whether it is dark or light.
- In the **light**, this sensor will have a resistance of only a **few hundred ohms.**
- In the **dark**, it can have a resistance of several **mega ohms**.

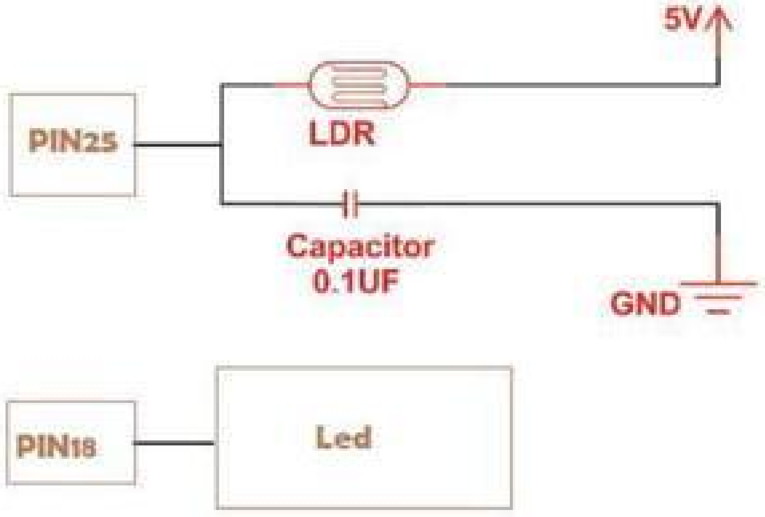
LDR Sensor



LDR Sensor

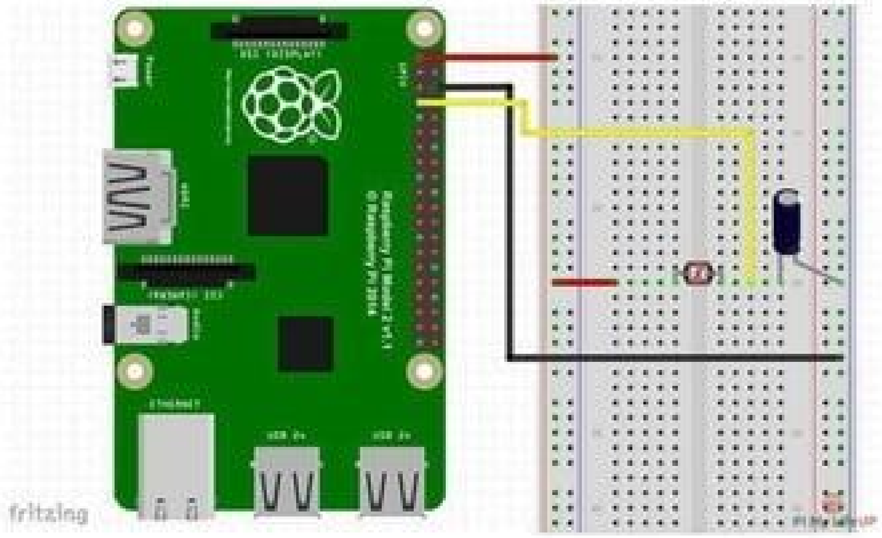
- A **photoresistor**, or light-dependent resistor (LDR), or photocell is a resistor whose **resistance will decrease when incident light intensity increase**; in other words, it exhibits photoconductivity.
- A photoresistor is made of a **high resistance semiconductor.**
- If light falling on the device is of high enough frequency, photons absorbed by the semiconductor give bound electrons enough energy to jump into the conduction band. The resulting free electron (and its hole partner) conduct electricity, thereby lowering resistance.

CIRCUIT Diagram



LDR Sensor

- In this implementation we learn how to Interface a light Sensor (LDR) with Raspberry Pi and turning an LED on/off based on the ligh-level sensed.
- Look given image, Connect one side of LDR to 3.3V and other side to a1µF capacitor and also to a GPIO pin (pin 18 in this example).
- An LED is connected to pin 18 which is controlled based on the light-level sensed.
- The readLDR() function returns a count which is proportional to the light level.
- In this function the LDR pin is set to output and low and then to input.
- At this point the capacitor starts charging through the resistor (and a counter is started) until the input pin reads high (this happens when capacitor voltage becomes greater than 1.4V).
- The counter is stopped when the input reads high. The final count is proportional to the light level as greater the amount of light, smaller is the LDR resistance and greater is the time taken to charge the capacitor.


fritzing

Experimental Procedures

- **Step 1**: Build the circuit
- **Step 2:** Create a Python / C code
- **Step 3:** Run

```
sudo python 03_ldrAndLed.py
```

Importing Packages

- To begin, we import the **GPIO** package that we will need so that we can communicate with the GPIO pins.
- We also import the **time** package, so we're able to put the script to sleep for when we need to.

```
import RPi.GPIO as GPIO
import time
```

Pin Assignment

- We then set the GPIO mode to GPIO.BCM, and this means all the numbering we use in this script will refer to the physical numbering of the pins.
- Since we only have one **input/output** pin, we only need to set one variable. Set this variable to the number of the pin you have acting as the input/output pin.

```
GPIO.setmode(GPIO.BCM)
ldr_threshold = 1000
LDR_PIN = 18
LIGHT_PIN = 25
```

Reading the Light Value

- Next, we have a function called readLDR() that requires one parameter, which is the pin number to the circuit. In this function, we initialize a variable called **reading**, and we will return this value once the pin goes to high.
- We then set our pin to act as an output and then set it to low. Next, we have the script sleep for 10ms.
- After this, we then set the pin to become an input, and then we enter a **while loop**. We stay in this loop until the pin goes to high, this is when the capacitor charges to about 3/4.
- Once the pin goes high, we return the count value to the main function. You can use this value to turn on and off an LED, activate something else, or log the data and keep statistics on any variance in light.

```
def readLDR(PIN):
        reading = 0
        GPIO.setup(LIGHT_PIN, GPIO.OUT)
        GPIO.output(PIN, false)
        time.sleep(0.1)
        GPIO.setup(PIN, GPIO.IN)
        while (GPIO.input (PIN) ==Flase):
                reading=reading+1
        return reading
```

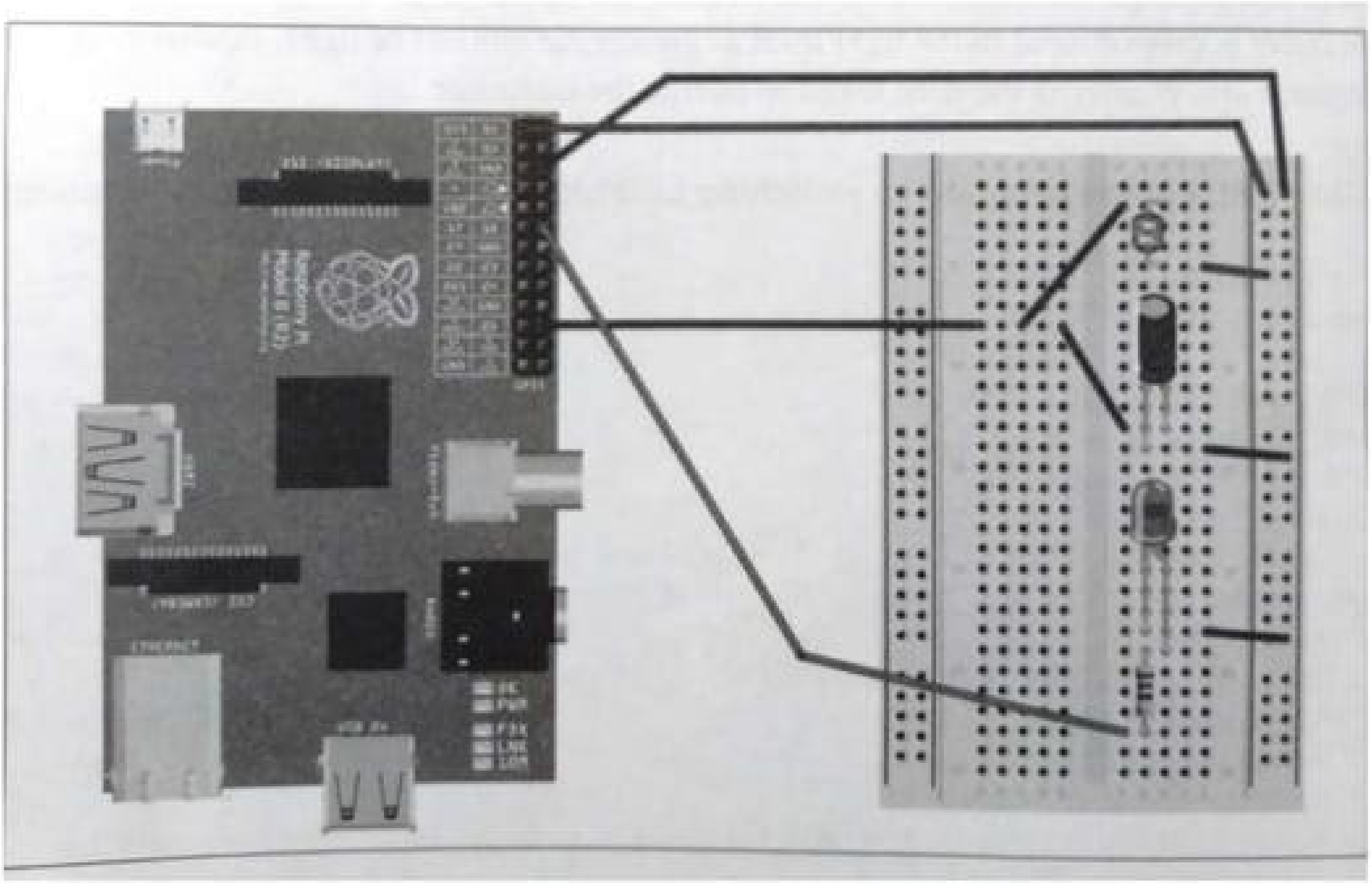
Python Code

```
# Example code Interfacing a light Sensor (LDR) with Raspberry Pi
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
ldr_threshold = 1000
LDR_PIN = 18
LIGHT_PIN = 25

def readLDR(PIN):
        reading = 0
        GPIO.setup(LIGHT_PIN, GPIO.OUT)
        GPIO.output(PIN, false)
        time.sleep(0.1)
        GPIO.setup(PIN, GPIO.IN)
        while (GPIO.input (PIN) ==Flase):
                reading=reading+1
        return reading
```

Python Code to Toggle the LED

```
def switchOnLight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, True)
def switchOffLight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, False)
while True:
    ldr_reading = readLDR(LDR_PIN)
    if ldr_reading < ldr_threshold:
        switchOnLight (LIGHT_PIN)
    else:
        switchOffLight(LIGHT_PIN)
    time.sleep(1)
```

Interfacing



Applications

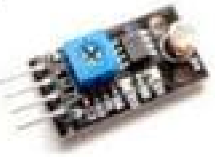
- There are countless uses for a light sensor in a circuit. I will just name a few that I thought of while I was writing up this PPT.
- **Light Activated Alarm** – I mentioned this one earlier, but you can use the LDR to detect when it starts to get light so you can sound an alarm to wake you up. If the program and sensor are accurate, then you can have the alarm slowly get louder as it gets lighter.
- **Garden monitor** – A light sensor could be used in a garden to check how much sun a certain area of the garden is getting. This could be useful information if you're planting something that needs lots of sun or vice versa.
- **Room Monitor** – Want to make sure lights are always turned off in a certain room? You could use this to alert you whenever some light is detected where it shouldn't be.

Problem In This Method

- The biggest problem of this circuit is the fact that the Pi **doesn’t have any analogue pins**. They’re all digital, so we **can’t accurately measure the variance in resistance on our input**.
- This lack of analogue pins wasn’t a problem in the motion sensor tutorial since the output from it was either high or low (Digital).
- Instead, we will measure **the time it takes for the capacitor to charge** and **send the pin high**.
- This method is an easy but inaccurate way of telling whether it is light or dark.

Solution

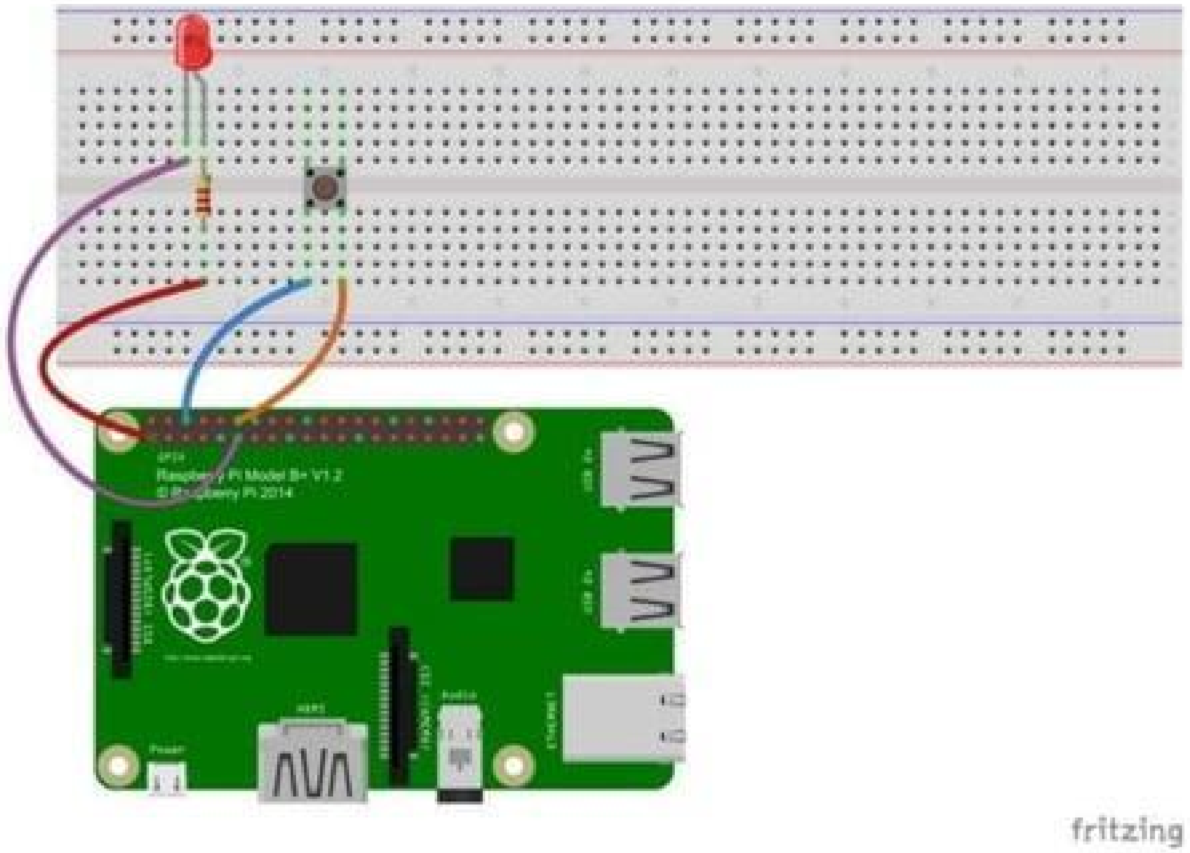
- If you want to make a light controlled switch, a single photoresistor might be useless since you will need the digital signal according to the brightness. This module is designed for that purpose.
  - using a high quality photoresistor
  - working voltage: 3.3~5V
  - output: digital switching (LOW or HIGH voltage on D pin) and analog signal (voltage output on A pin)
  - using a wide voltage LM393 comparator that has good waveform
  - output current >= 15mA, can directly light LED.
  - with adjustable potentiometer to adjust the sensitivity
  - has two M2.5 mounting holes
- This module is very sensitive to ambient light, and is very suitable for detecting brightness of ambient light. The output signal can trigger Raspberry Pi, microcontroller like Arduino, or other digital relay modules.
- When the ambient light intensity is lower than the predefined threshold, the output signal is high. When the light intensity reaches or exceeds the threshold, the signal output is low.



Python programming with Raspberry PI with focus of

- Interfacing external gadgets
- Controlling output
- Reading input from pins

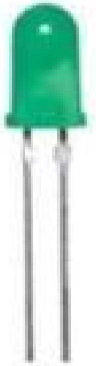
Sending an E-Mail on Switch Press



Components Reuired

- You'll need the following components to connect the circuit.
- 1. Raspberry Pi
  2. LED
  3. Resistor - 330 ohm
  4. Breadboard
  5. Button / Switch
  6. 2 Male-Female Jumper Wires

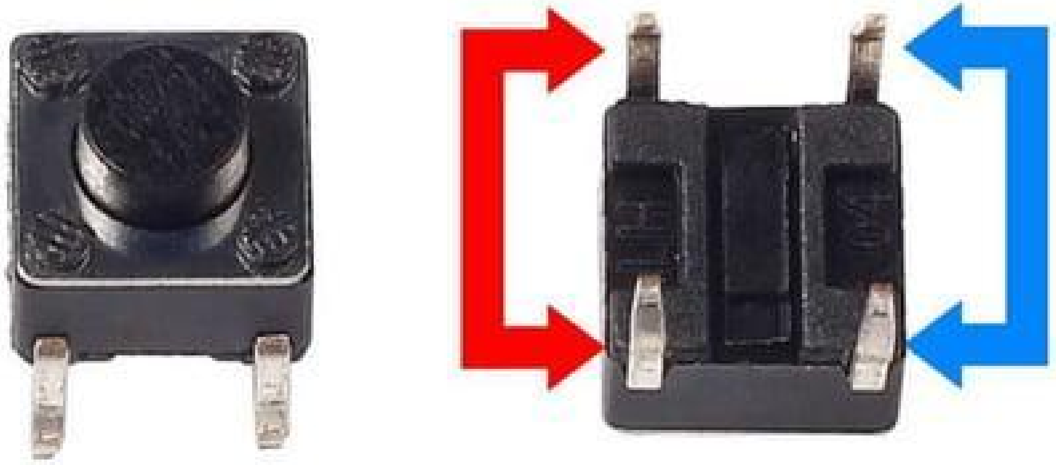
Components



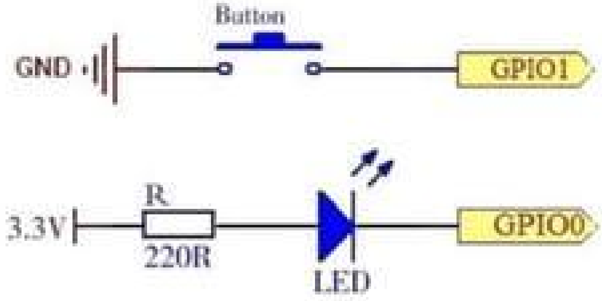
Switch / Button

- Buttons are a common component used to control electronic devices.
- They are usually used as switches to connect or disconnect circuits.
- Although buttons come in a variety of sizes and shapes, the one used here is a 6mm mini-button as shown in the following pictures.
- Pins pointed out by the arrows of same color are meant to be connected.

Switch / Button



CIRCUIT Diagram

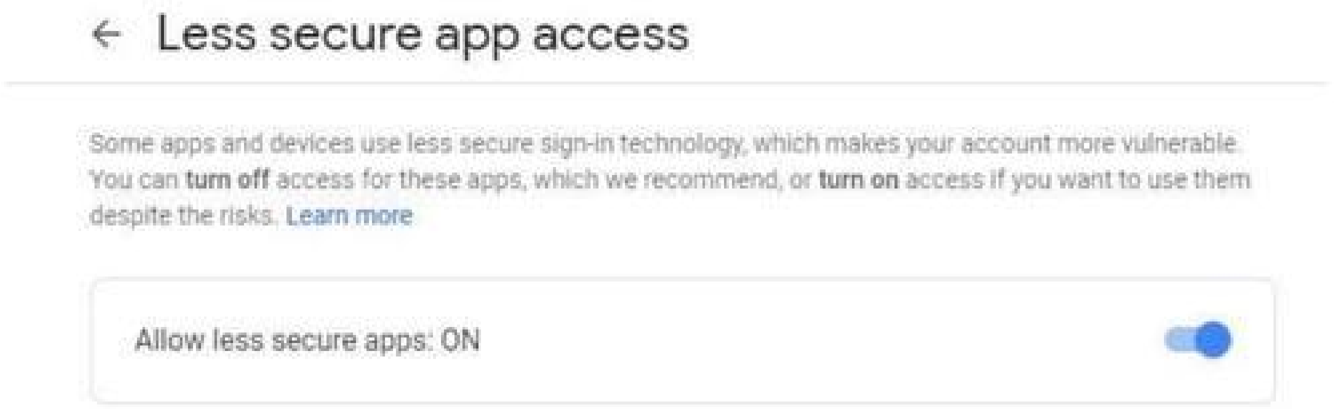


How It Is Works

- Sending an email with the Pi using Python isn't that difficult.
- We are going to create the email content in code and use an external email provider to act as the mail server send the email.
- Because we are using code to generate the email, we can dynamically create the content we send based on sensor readings, time of day, or just about anything else you can imagine!
- In this example we are going to use Google / Gmail to provide email services.
- We will create a Gmail account specifically for this device and give the Pi permission to send with this account.
- **Do not use your normal email address** – the email address and password are stored in plain text in your code!

Set GMAIL Permissions

- To access a Gmail account using an external device like the Raspberry Pi, permission for "**less secure apps**" will have to be enabled.
- When logged in as your new email account you can jump over to https://myaccount.google.com/lesssecureapps and enable this setting.
- Since most people have several Google accounts, just be sure you are using the right account when you enable this setting!



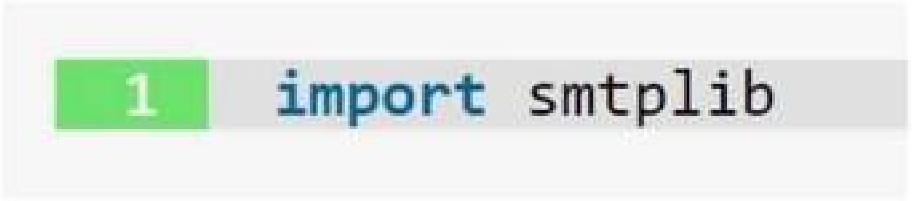
← Less secure app access

Some apps and devices use less secure sign-in technology, which makes your account more vulnerable. You can **turn off** access for these apps, which we recommend, or **turn on** access if you want to use them despite the risks. Learn more

Allow less secure apps: ON

SMTP Library

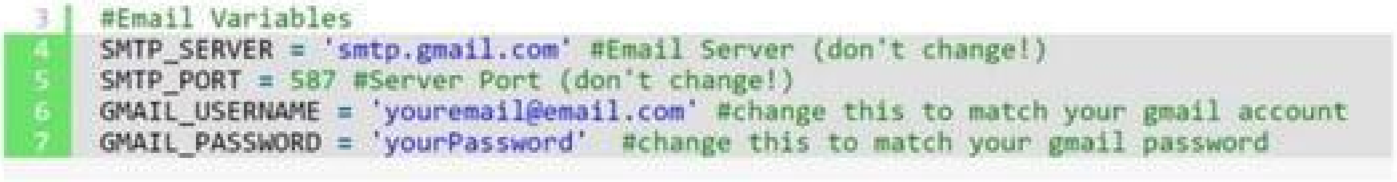
- First we are going to start by adding in the smtp library reference. This gives us the email tools we are going to need.



```
import smtplib
```

E-Mail Variables

- We need to store a bunch of information for sending out emails as well.
- The SMTP Server and SMTP port are not going to change, but be sure to update the email address and password in your code to match the credentials for the Gmail account you created earlier.

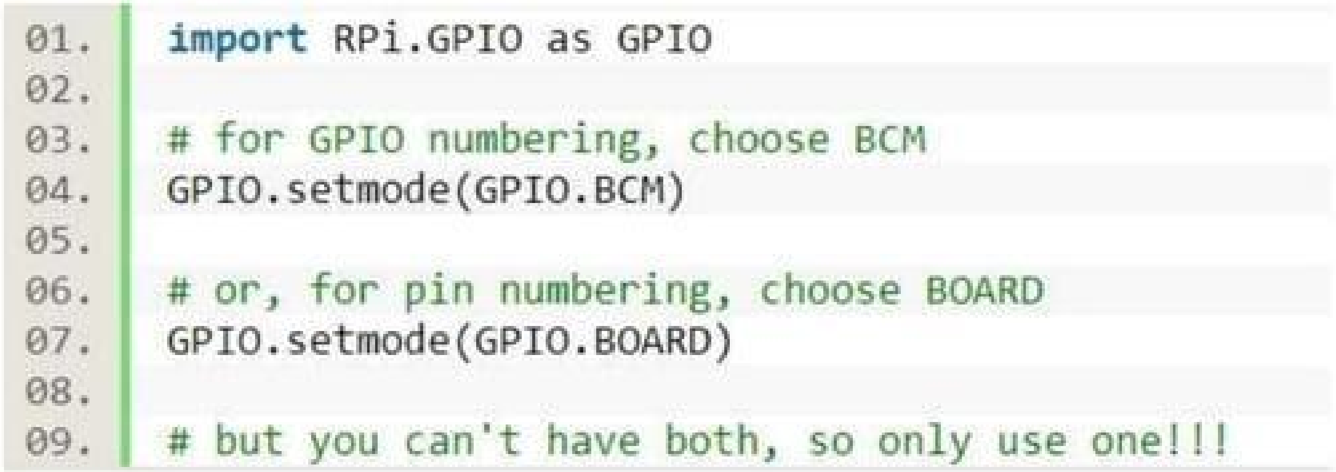


```
#Email Variables
SMTP_SERVER = 'smtp.gmail.com' #Email Server (don't change!)
SMTP_PORT = 587 #Server Port (don't change!)
GMAIL_USERNAME = 'youremail@email.com' #change this to match your gmail account
GMAIL_PASSWORD = 'yourPassword'  #change this to match your gmail password
```

How to set up BOARD and GPIO numbering schemes

- There are two kinds of Input and Output pin numbering for the Raspberry pi. One is the BCM and the other is BOARD. Basically these pin numberings are useful for writing python script for the Raspberry Pi.
- **GPIO BOARD**– This type of pin numbering refers to the number of the pin in the plug, i.e, the numbers printed on the board, for example, P1. The advantage of this type of numbering is, it will not change even though the version of board changes.
- **GPIO BCM**– The BCM option refers to the pin by "Broadcom SOC Channel. They signify the Broadcom SOC channel designation. The BCM channel changes as the version number changes.
- Note: It is very important to wire the GPIO pins with limited resistors to avoid serious damage to the Raspberry Pi. LEDs must have resistors to limit the current passing through them. The motors should not be connected to the GPIO pins directly.
- In RPi.GPIO you can use either pin numbers (BOARD) or the Broadcom GPIO numbers (BCM), **but you can only use one system in each program.**
- Both have advantages and disadvantages.
- If you use **pin numbers**, you don't have to bother about revision checking, as RPi.GPIO takes care of that for you. You still need to be aware of which pins you can and can't use though, since some are power and GND.
- If you use **GPIO numbers**, your scripts will make better sense if you use a Gertboard, which also uses GPIO numbering. If you want to use the P5 header for GPIO28-31, you have to use GPIO numbering. If you want to control the LED on a Pi camera board (GPIO5) you also have to use GPIO numbering.

Raspberry Pi Numbering Schemes



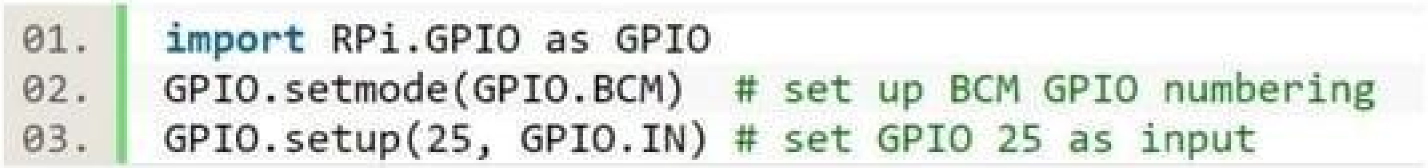
```python
import RPi.GPIO as GPIO

# for GPIO numbering, choose BCM
GPIO.setmode(GPIO.BCM)

# or, for pin numbering, choose BOARD
GPIO.setmode(GPIO.BOARD)

# but you can't have both, so only use one!!!
```

How to set up a GPIO port as an input

- Use the following line of code…

  GPIO.setup(Port_or_pin, GPIO.IN)
- …changing *Port_or_pin* to the number of the GPIO port or pin you want to use. I'm going to use the BCM GPIO numbering and port GPIO25, so it becomes…

  GPIO.setup(25, GPIO.IN)



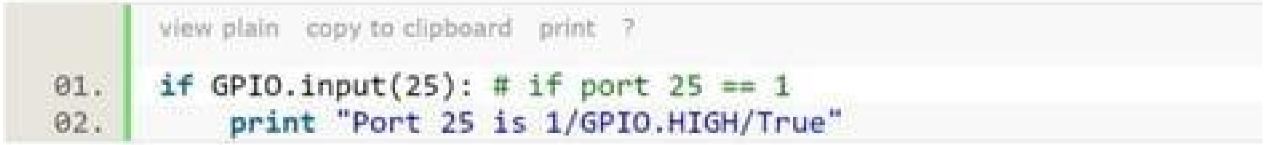
```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)  # set up BCM GPIO numbering
GPIO.setup(25, GPIO.IN) # set GPIO 25 as input
```

Reading inputs

- Inputs are Boolean values: 1 or 0, GPIO.HIGH or GPIO.LOW, True or False (this corresponds to the voltage on the port: 0V=0 or 3.3V=1). You can read the value of a port with this code…

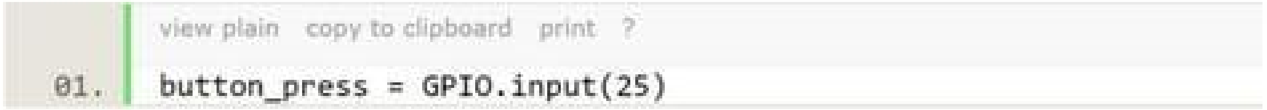
  GPIO.input(25)

But it may be more useful to use it as part of your logic…



```
if GPIO.input(25): # if port 25 == 1
    print "Port 25 is 1/GPIO.HIGH/True"
```

…or store its value in a variable to use in a different part of the program…



```
button_press = GPIO.input(25)
```

GPIO Pin Assignment

```
#Set GPIO pins to use BCM pin numbers
GPIO.setmode(GPIO.BCM)

#Set digital pin 17(BCM) to an input
GPIO.setup(17, GPIO.IN)

#Set digital pin 17(BCM) to an input and enable the pullup
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)

#Event to detect button press
GPIO.add_event_detect(17, GPIO.FALLING)
```

E-Mail Sender

- Next, we are going to write the chunk of code that actually sends the email.
- This creates the email, formats it correctly, creates the connection to Gmail, logs in, and sends the email.
- It is quite a few lines of code so we are going to put it into a custom Class – that way we can write this code once, but call it any time we want to send an email in our program.

Create Headers

```
#Create Headers
headers = ["From: " + GMAIL_USERNAME, "Subject: " + subject, "To: " + recipient,
           "MIME-Version: 1.0", "Content-Type: text/html"]
headers = "\r\n".join(headers)
```

Connect To GAMIL Server

```
#Connect to Gmail Server
session = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
session.ehlo()
session.starttls()
session.ehlo()
```

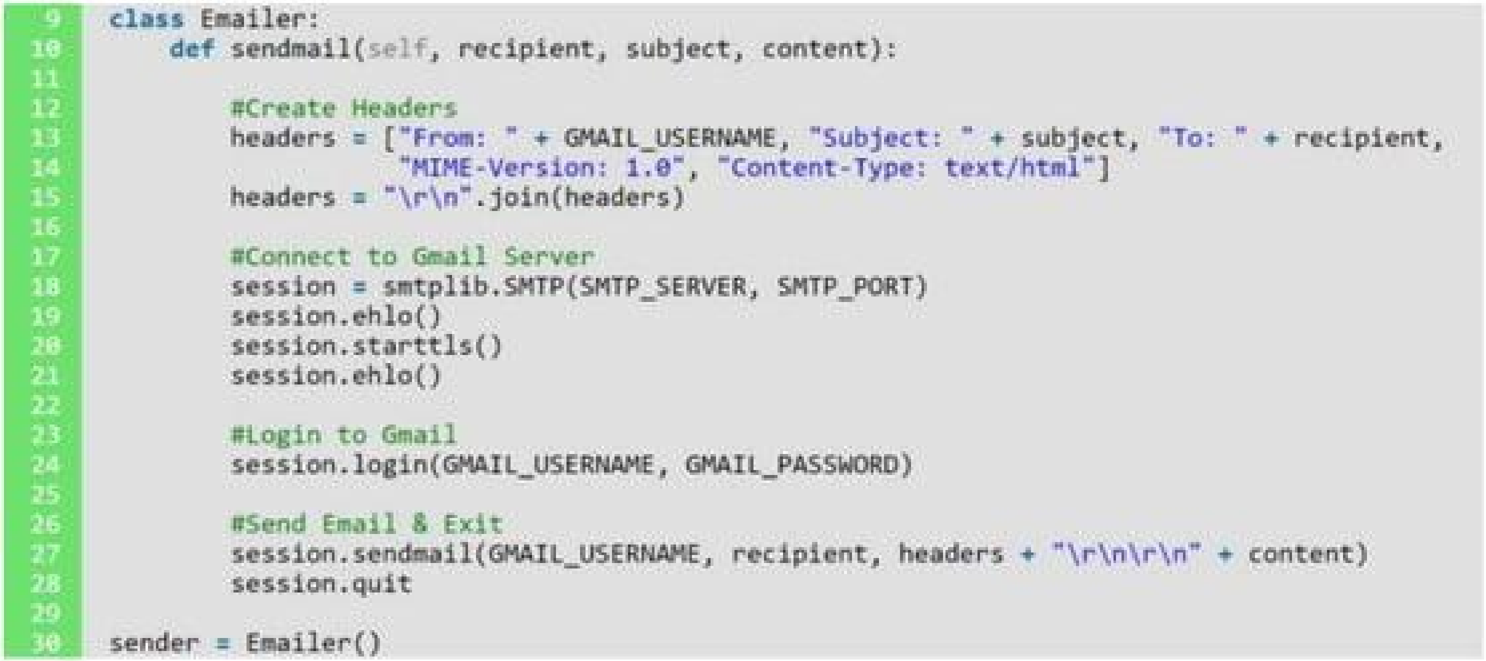
- Extended HELO (EHLO) is an Extended Simple Mail Transfer Protocol (ESMTP) command sent by an email server to identify itself when connecting to another email server to start the process of sending an email.
- It is followed with the sending email server's domain name. The EHLO command tells the receiving server it supports extensions compatible with ESMTP.
- SSL, TLS, and STARTTLS refer to standard protocols used to secure email transmissions.
- SSL (Secure Sockets Layer) and its successor, Transport Layer Security (TLS), provide a way to encrypt a communication channel between two computers over the Internet. In most cases, the terms SSL and TLS can be used interchangeably unless you're referring to a specific version of the protocol.
- Because TLS and SSL are application-layer protocols, senders and receivers need to know that they are being used to encrypt emails during transit. That's where STARTTLS comes into play.
- STARTTLS is an email protocol command that tells an email server that an email client, including an email client running in a web browser, wants to turn an existing insecure connection into a secure one. (By the way, the use of "TLS" in the STARTTLS command name does not mean that it only works with the TLS security protocol. It works with SSL too.)
- StartTLS has become the most popular e-mail encryption method among internet providers, as it facilitates the use of many different domains and certificates on one server

Login To Gmail

```
#Login to Gmail
session.login(GMAIL_USERNAME, GMAIL_PASSWORD)
```

Send Email & Exit

```
#Send Email & Exit
session.sendmail(GMAIL_USERNAME, recipient, headers + "\r\n\r\n" + content)
session.quit
```



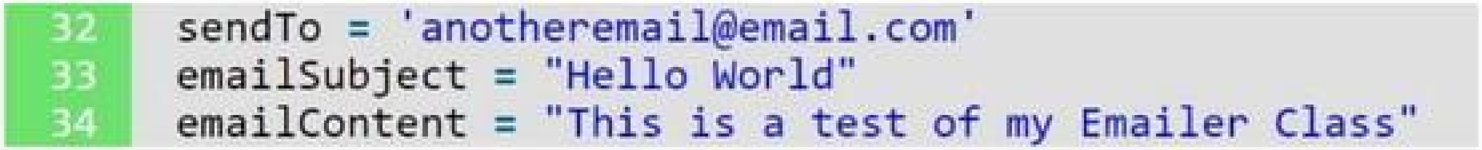
```python
class Emailer:
    def sendmail(self, recipient, subject, content):

        #Create Headers
        headers = ["From: " + GMAIL_USERNAME, "Subject: " + subject, "To: " + recipient,
                   "MIME-Version: 1.0", "Content-Type: text/html"]
        headers = "\r\n".join(headers)

        #Connect to Gmail Server
        session = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
        session.ehlo()
        session.starttls()
        session.ehlo()

        #Login to Gmail
        session.login(GMAIL_USERNAME, GMAIL_PASSWORD)

        #Send Email & Exit
        session.sendmail(GMAIL_USERNAME, recipient, headers + "\r\n\r\n" + content)
        session.quit

sender = Emailer()
```

E-Mail Contents

- The Class we wrote in the last step sends out the email.
- The class takes three arguments (pieces of information we specify when we use the code to send an email) and constructs the email before sending.
- These three external components are
  - the recipient of the email,
  - a subject line for the email, and
  - the text you want to have in the body of the email.
- These three pieces of information are stored in three variables (sendTo, emailSubject, and emailContent).
- This way, our code is capable of sending emails to different people, with different subject lines, and different email content without having to write a different sender for each new contact / email message.
- Add the three variables and be sure to set the recipient to somewhere you want to receive your first test email

E-Mail Contents

- Example
  - emailContent = ‘Switch pressed on Raspberry Pi’



```
sendTo = 'anotheremail@email.com'
emailSubject = "Hello World"
emailContent = "This is a test of my Emailer Class"
```

Calling the E-Mail Class

- Now that we have everything set up – sending an email is as simple as calling the Emailer Class with the three arguments (sendTo, emailSubject, and emailContent).
- It will then create and send the email based on those parameters.
- Sends an email to the "sendTo" address with the specified "emailSubject" as the subject and "emailContent" as the email content.

```
#Sends an email to the "sendTo" address with the specified "emailSubject"
sender.sendmail(sendTo, emailSubject, emailContent)
```

Example PYTHON Code for Sending an Email on Switch Press

- Here is a quick example of this code being used to send an email anytime a button is pressed.
- To try it out, connect a momentary button to GPIO 17 and GND.
- You will also need to change your Gmail Sender Email and Password, along with the recipient at the bottom.

Python Code

```
import smtplib
import RPi.GPIO as GPIO
import time

#Email Variables
SMTP_SERVER = 'smtp.gmail.com' #Email Server (don't change!)
SMTP_PORT = 587 #Server Port (don't change!)
GMAIL_USERNAME = 'youremail@email.com' #change this to match your gmail account
GMAIL_PASSWORD = 'yourPassword'  #change this to match your gmail password

#Set GPIO pins to use BCM pin numbers
GPIO.setmode(GPIO.BCM)

#Set digital pin 17(BCM) to an input
GPIO.setup(17, GPIO.IN)

#Set digital pin 17(BCM) to an input and enable the pullup
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)

#Event to detect button press
GPIO.add_event_detect(17, GPIO.FALLING)
```

Python Code (Contd.,)

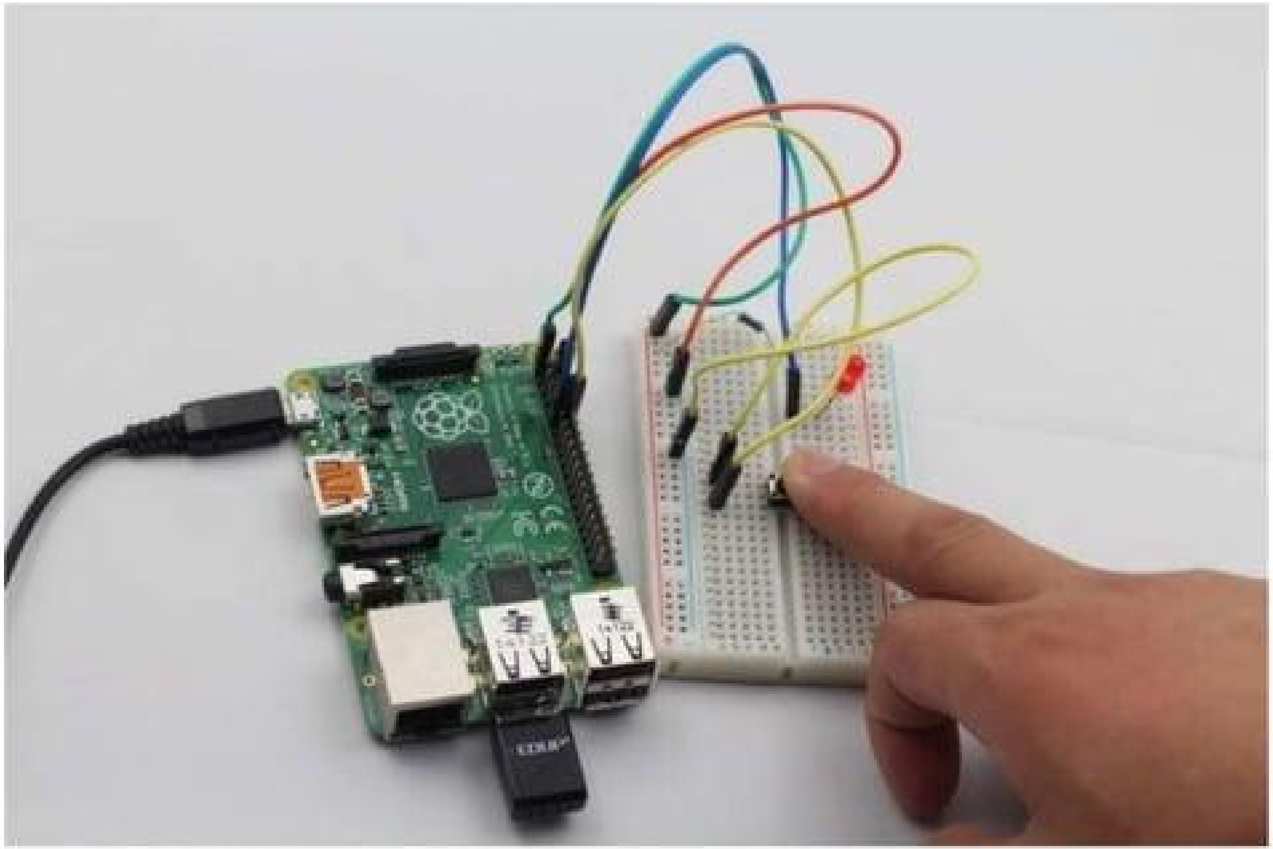
```
class Emailer:
    def sendmail(self, recipient, subject, content):

        #Create Headers
        headers = ["From: " + GMAIL_USERNAME, "Subject: " + subject, "To: " + recipient,
                "MIME-Version: 1.0", "Content-Type: text/html"]
        headers = "\r\n".join(headers)

        #Connect to Gmail Server
        session = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
        session.ehlo()
        session.starttls()
        session.ehlo()

        #Login to Gmail
        session.login(GMAIL_USERNAME, GMAIL_PASSWORD)

        #Send Email & Exit
        session.sendmail(GMAIL_USERNAME, recipient, headers + "\r\n\r\n" + content)
        session.quit

sender = Emailer()
```

Python Code (Contd.,)

```
while True:
   if GPIO.event_detected(17):
      sendTo = 'anotheremail@email.com'
      emailSubject = "Button Press Detected!"
      emailContent = "The button has been pressed at: " + time.ctime()
      sender.sendmail(sendTo, emailSubject, emailContent)
      print("Email Sent")

   time.sleep(0.1)
```

- Once the email and password fields are updated, run this code by hitting "F5" on the keyboard.
- When the button is depressed, an email should be received at the receiving address a moment later with the time the button was pressed.

Interfacing



Demonstration Video

- Video Link

Thank You