

# Tutorial 2 : Interfacing Sensors to Rpi | Learning Python on the Go....

**Dr. Bibhas Ghoshal**

**Assistant Professor**

**Department of Information Technology**

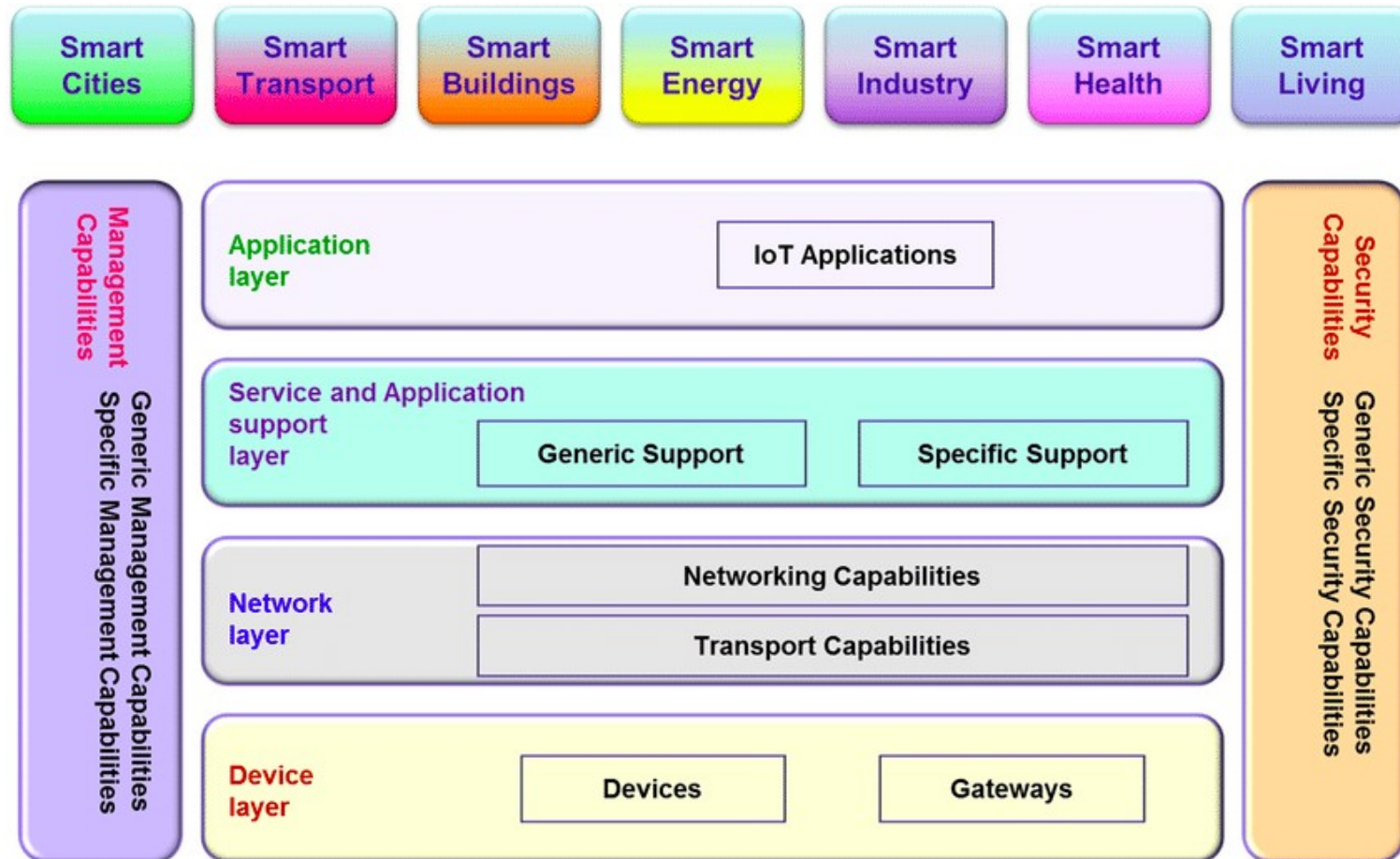
**Indian Institute of Information Technology Allahabad**



**Internet of Things  
Instructor : Dr. Bibhas Ghoshal**

**Spring 2022**

# The Four Layered IoT Architecture

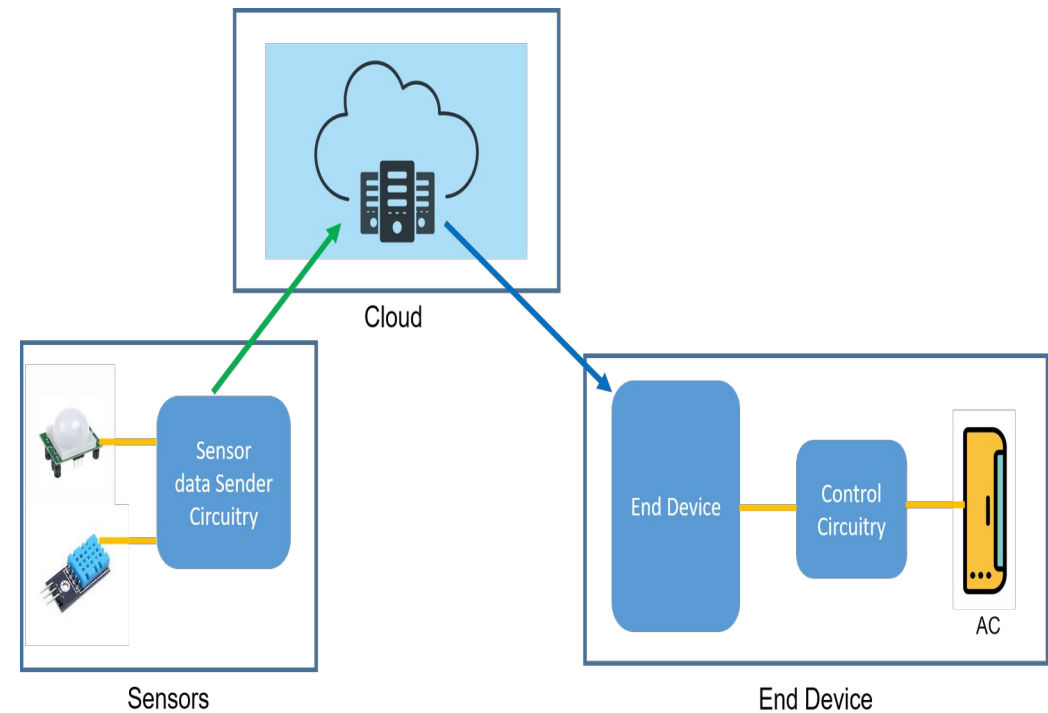
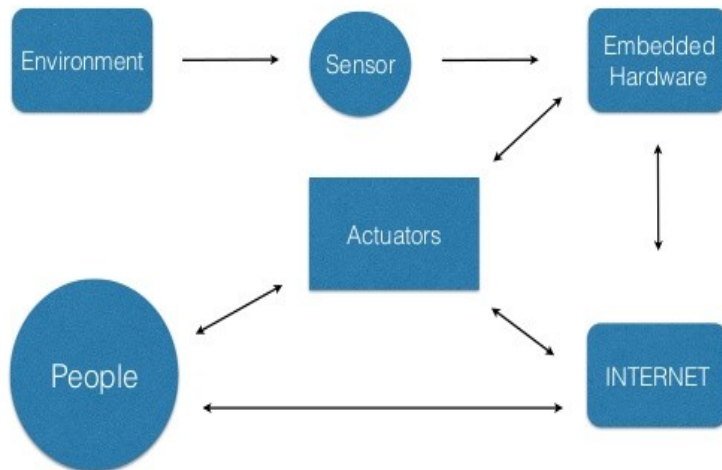


Source : ITU-T

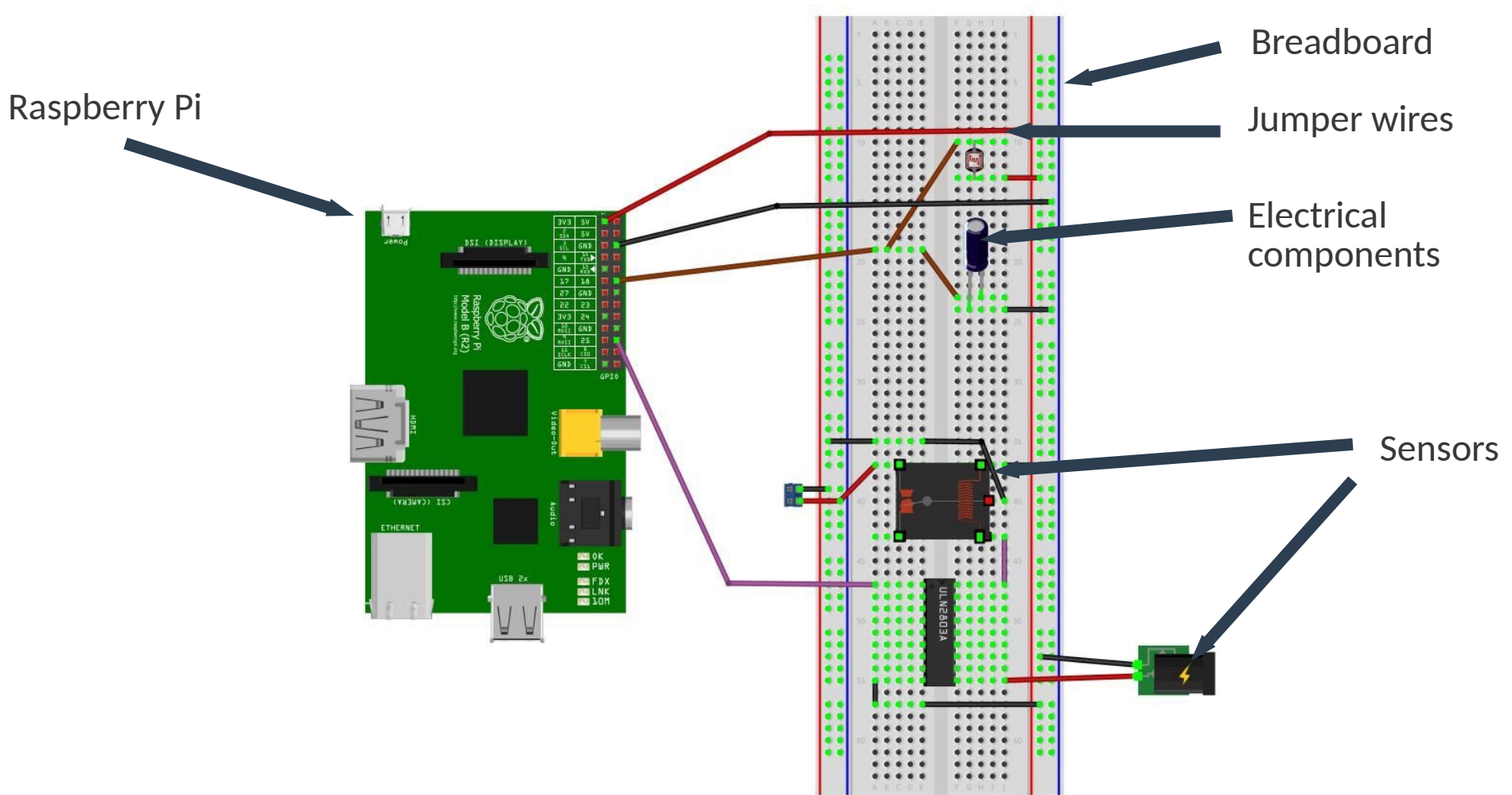


# IoT – Basic Idea

How IoT works ?



# Interfacing sensors to the computing unit



Source: Book website: <http://www.internet-of-things-book.com>

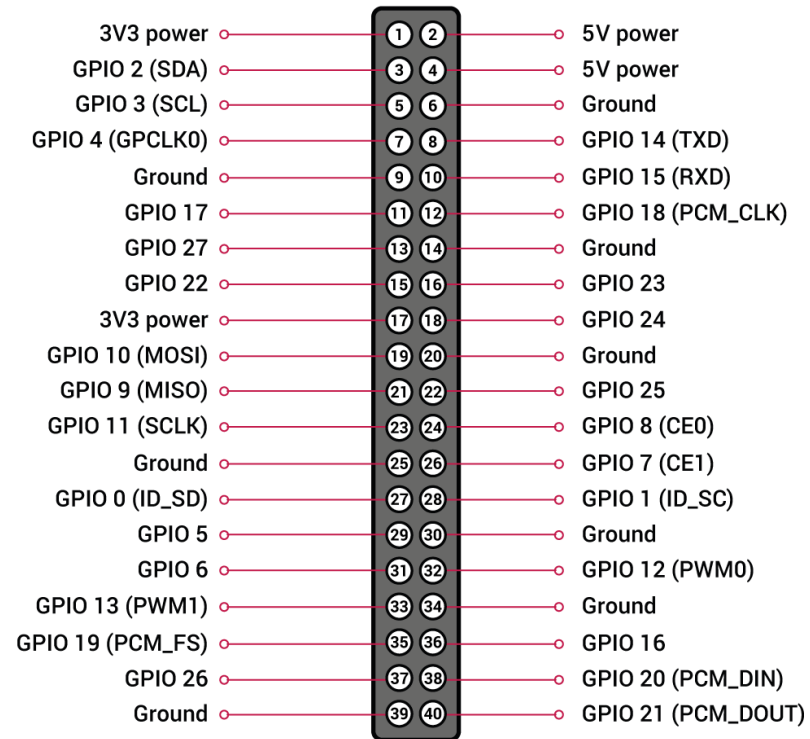
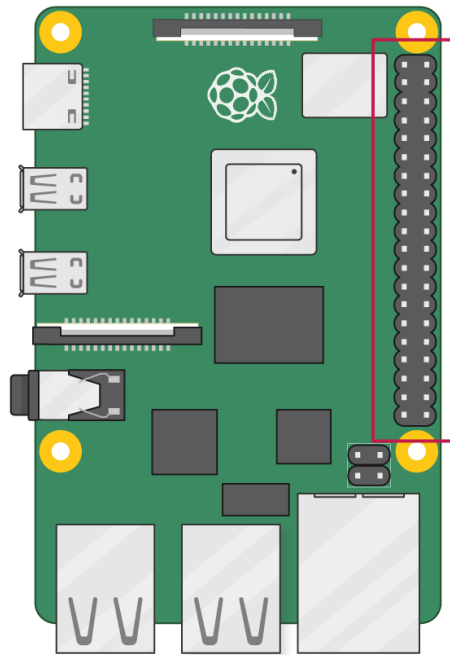


**Internet of Things**  
Instructor : Dr. Bibhas Ghoshal

Spring 2022

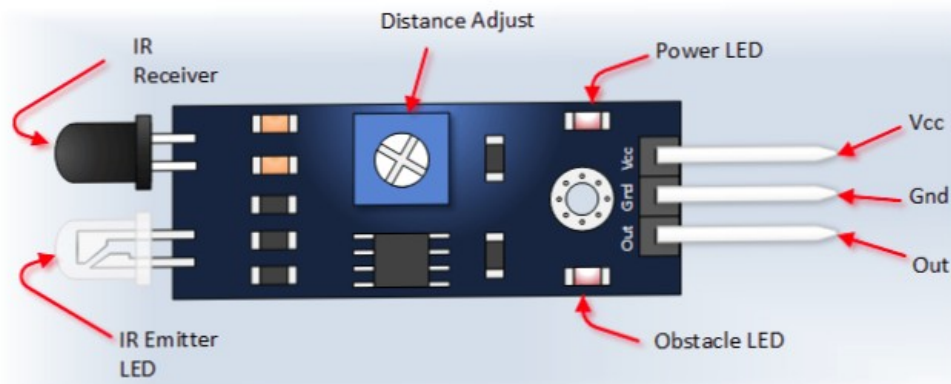
# Components

## 1. Computing Unit – Raspberry Pi ( 3B/ 3B+ / 4)



# Components

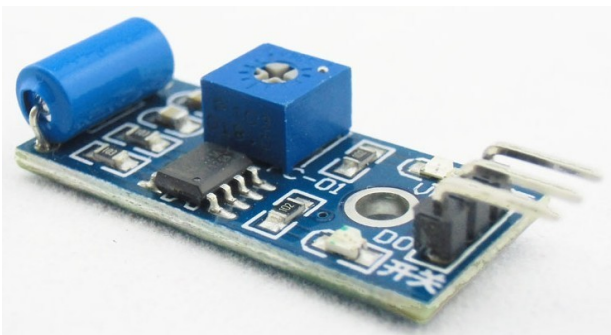
## 2. Sensors



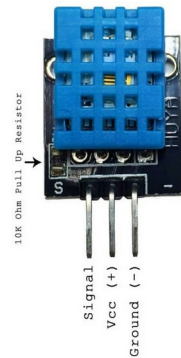
IR Sensor



Ultrasonic Distance Sensor



Vibration Sensor



Temperature Sensor



Tilt Sensor





# Components

## 3. Actuators



Servo motor



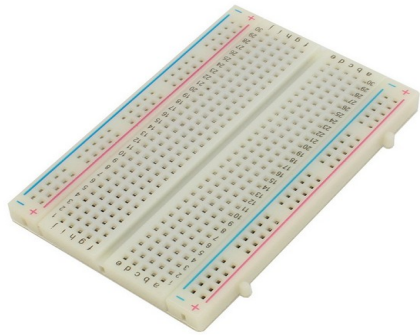
Buzzer



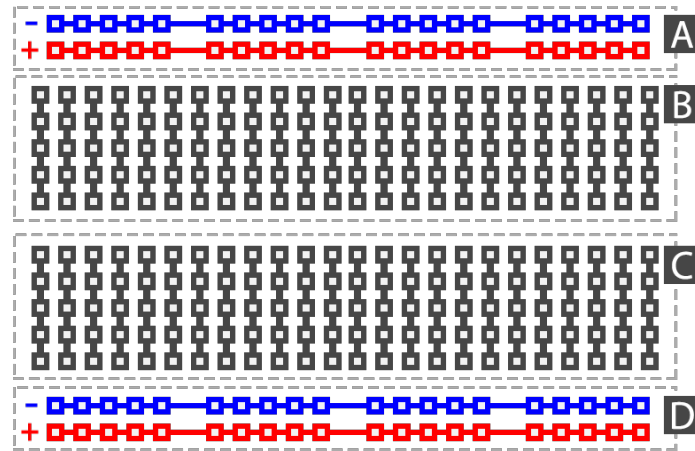
Linear Actuators

# Components

## 4. Other components



Breadboard

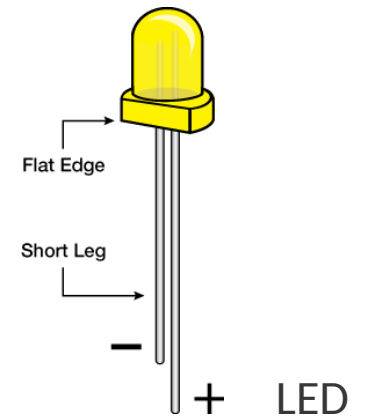


Jumper Wires



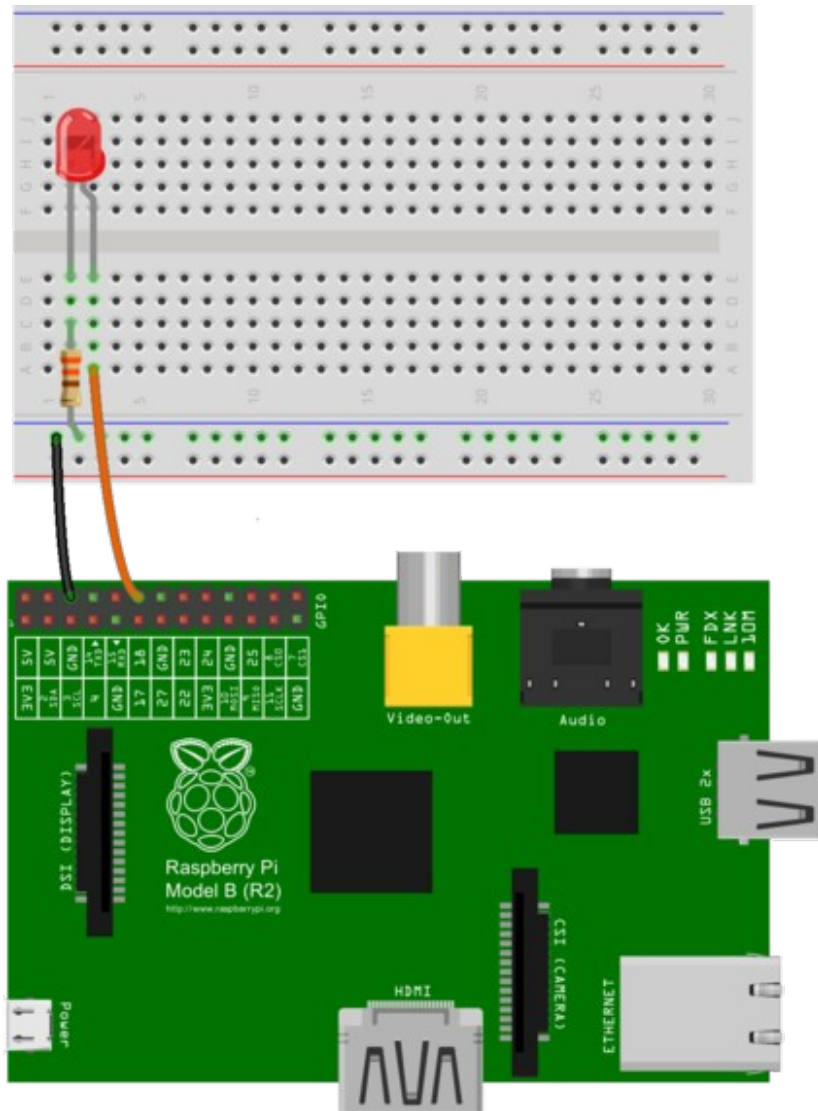
Resistors

Internal connections of a Breadboard

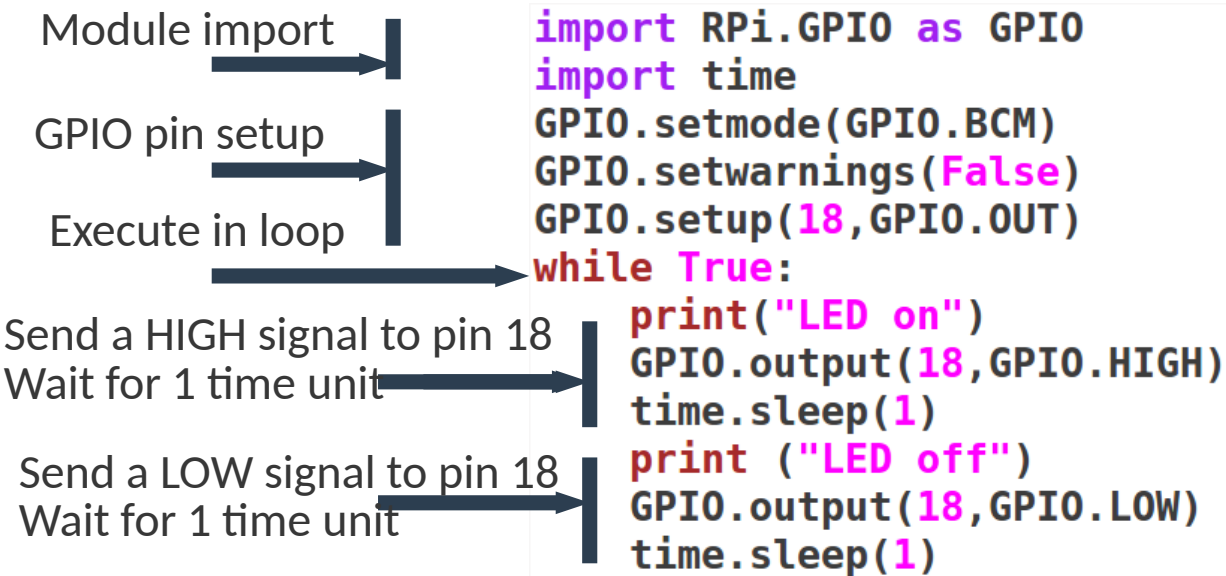




# Interfacing Rpi to components : GPIO ports

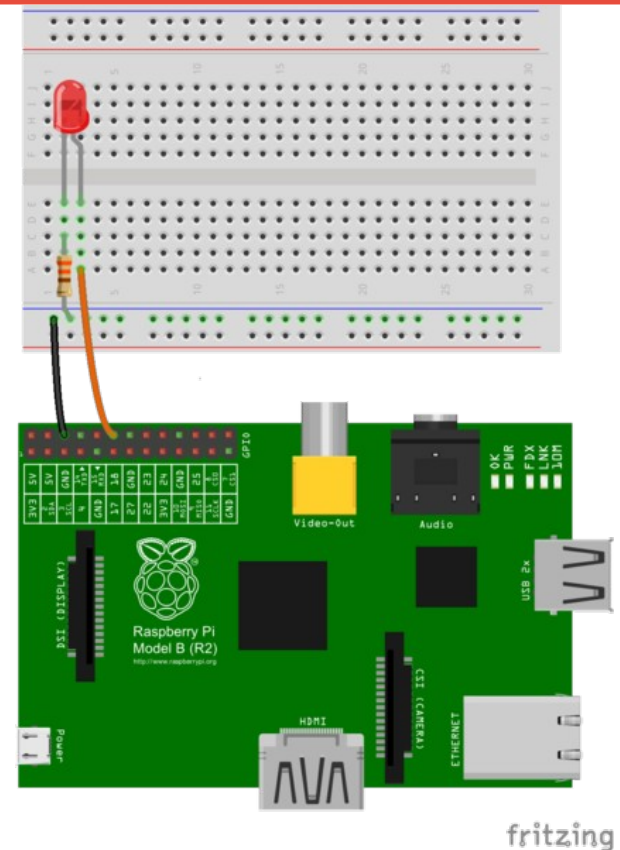


# Programming the Raspberry Pi : Learning Python on the Go.....



```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(18, GPIO.OUT)
while True:
    print("LED on")
    GPIO.output(18, GPIO.HIGH)
    time.sleep(1)
    print("LED off")
    GPIO.output(18, GPIO.LOW)
    time.sleep(1)
```

Python program which instructs the Rpi to blink a LED - turn the LED high and low alternately



The latest version of Raspbian includes the RPi.GPIO Python library pre-installed,

so you can simply import that into your Python code. The RPi.GPIO is a library that allows your Python application to easily access the GPIO pins on your Raspberry Pi. The `as` keyword in Python allows you to refer to the RPi.GPIO library using the shorter name of `GPIO`. There are two ways to refer to the pins on the GPIO: either by physical pin numbers (starting from pin 1 to 40 on the Raspberry Pi 2/3), or Broadcom GPIO numbers (BCM)



# Introducing Python

1. General-purpose, versatile, and powerful programming language
2. It was created by **Guido van Rossum** in **1991** and was named after the British comedy show, *Monty Python's Flying Circus*.
3. Applications – scripting, web development, data visualization, data analysis, machine learning
4. **Characteristics :**
  - i. *Multi-paradigm programming language* - Python supports more than one programming paradigms including object-oriented programming and structured Programming



# Introducing Python

## 4. Characteristics :

ii. *Interpreted and Interactive language – does not require compilation steps, Users provide command at the prompt which get executed directly*

iii. *Easy to learn and maintain – fewer syntactical constructions, Statements feel like English with pseudo code constructs*

iv. *Extendable, Scalable and Portable*

v. *Broad Range of Library support*



# All about Python : Installation, Guide, Docs....

[www.python.org](http://www.python.org)

The screenshot shows the Python.org homepage. At the top, there is a navigation bar with links for Python, PSF, Docs, PyPI, Jobs, and Community. Below this is the Python logo and a search bar. A secondary navigation bar contains links for About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area features a code editor on the left with Python code demonstrating list comprehensions and the enumerate function. To the right of the code is a section titled "Compound Data Types" explaining lists. Below the code and text is a navigation bar with numbered links 1 through 5. At the bottom of the main content area, there is a tagline: "Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)".

## 🔌 Get Started

Whether you're new to programming or an experienced developer, it's easy to learn and use Python.

## 📄 Download

Python source code and installers are available for download for all versions!

Latest: [Python 3.10.2](#)

## 📖 Docs

Documentation for Python's standard library, along with tutorials and guides, are available online.

[docs.python.org](http://docs.python.org)

## 💼 Jobs

Looking for work or have a Python related position that you're trying to hire for? Our **relaunched community-run job board** is the place to go.



Internet of Things  
Instructor : Dr. Bibhas Ghoshal

Spring 2022

# Using the Python Environment

## Invoking the Python environment : Type python at the prompt

```
Last login: Tue Feb 1 08:39:06 on ttys000
Bibhas-MacBook-Air:~ bibhasghoshal$ python
Python 2.7.16 (default, Jan 26 2020, 23:50:38)
[GCC 4.2.1 Compatible Apple LLVM 9.0.0 (clang-900.0.31)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

## First Python Program : Interpreted and Interactive

```
>>> print "Hello World !"
Hello World !
>>> █
```

## Python Program Structure and Rules

**Identifier :** name used to identify a variable, function, class, module, or other object. An identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores, and digits (0 to 9)

**Reserved words :** words that cannot be used as identifiers

Example : *and, exec, not, print, continue, def, if, return, import, try, while, else, except, for*





# Python Program Structure

**Indentation** : there are no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

```
>>> if True:
...     print "True"
... else:
...     print "False"
...
True
>>> █
```

**Comments** : A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the physical line end are part of the comment, and the Python interpreter ignores them.

```
>>> print "Hello!"
Hello!
>>> # print "Hello!"
...
>>> █
```

**Variables** : Reserved memory locations that store values, depending on the data type, interpreter allocates memory and decides what to store in that location. Thus, by assigning different data types to variables, one can store integers, floating point numbers, strings etc.

```
>>> c=100
>>> s="hello"
>>> █
```



# Data Types in Python

There are five standard data types in Python : numbers, string, list, tuple and dictionary

- Number : store numeric values, immutable data types
- Strings are list of characters

## # Integer

```
>>> a = 2
>>> type(a)
<type 'int'>
>>> █
```

## # Floating Point

```
>>> b = 2.5
>>> type(b)
<type 'float'>
>>> █
```

## # Complex Number

```
>>> c = 2+5j
>>> type(c)
<type 'complex'>
>>> c.real
2.0
>>> c.imag
5.0
>>> █
```

## # String operation

```
>>> s = "hello"
>>> t = "world!"
>>> w =s+t
>>> print w
helloworld!
>>> len(s)
5
>>> █
```

## # Type Conversion

```
>>> x = 100
>>> type(x)
<type 'int'>
>>> str(x)
'100'
>>> type(x)
<type 'int'>
>>> y=str(x)
>>> print y
100
>>> type(y)
<type 'str'>
>>> █
```

## # Integer Operation

```
>>> a
2
>>> b
2.5
>>> p = a+b
>>> print p
4.5
>>> q = a-b
>>> print q
-0.5
>>> print(a*b)
5.0
>>> print(a/b)
0.8
>>> print(a%b)
2.0
>>> r = b**a
>>> print r
6.25
>>> █
```



# Compound Data Types Python in Linux

**Lists :** used to group together values

## Operations done on a list

i. *Accessing elements* ii. *insert/remove/append items to list* iii. *change items in list*

**Tuple :** Read only list

```
>>> rivers = ['Ganges','Nile','Mississippi','Amazon','Yangtze']
>>> type(rivers)
<type 'list'>
>>> rivers[1]
'Nile'
>>> len(rivers)
5
>>> rivers.append('Danube')
>>> rivers
['Ganges', 'Nile', 'Mississippi', 'Amazon', 'Yangtze', 'Danube']
>>> rivers[4] = 'Volga'
>>> rivers
['Ganges', 'Nile', 'Mississippi', 'Amazon', 'Volga', 'Danube']
>>> river_countries = ('India','Egypt','USA','Brazil','Russia','Germany')
>>> type(river_countries)
<type 'tuple'>
>>> █
```



# Compound Data Types Python in Linux

**Dictionaries** : Hash table that maps keys to values

**Operations done on a list :**

i. *Get the value of a key*   ii. *Get all items*   iii. *Get all keys*   iv. *Add a key value pair*   v. *Check if dictionary has a key*

```
>>> student = {'name':'Alice','id':'100'}
>>> type(student)
<type 'dict'>
>>> len(student)
2
>>> student
{'name': 'Alice', 'id': '100'}
>>> student.keys
<built-in method keys of dict object at 0x107476e88>
>>> student.keys()
['name', 'id']
>>> student.has_key('id')
True
>>> student['gender']='Female'
>>> student
{'gender': 'Female', 'name': 'Alice', 'id': '100'}
>>> █
```



# Control Flow in Python

## # If Statement

```
>>
>>> a
2
>>> if a>10:
...     print " More than 10"
... else:
...     print " Between 1 to 10"
...
Between 1 to 10

[>>> student
{'gender': 'Female', 'name': 'Alice', 'id': '100'}
[>>> if not student.has_key('major'):
...     student['major']='CS'
...
[>>>
[>>> student
{'gender': 'Female', 'major': 'CS', 'name': 'Alice', 'id': '100'}
>>> █
```



# Control Flow in Python

## # For-Loop

### 1. Looping over a string

```
>>> str = "Hello"
>>> for s in str:
...     print s
...
H
e
l
l
o
>>> █
```

### 2. Looping over a list or dictionary

```
>>> rivers
['Ganges', 'Nile', 'Mississippi', 'Amazon', 'Volga', 'Danube']
>>> i=0
>>> for item in rivers:
...     print " River-%d:%s " %(i,item)
...     i = i+1
...
River-0:Ganges
River-1:Nile
River-2:Mississippi
River-3:Amazon
River-4:Volga
River-5:Danube
>>> █
```





# Control Flow in Python

## # While-Loop

```
>>> i=0
>>> while i<=10:
...     if i%2 == 0:
...         print i
...     i =i+1
...
0
2
4
6
8
10
>>>
```

## # Range

```
>>> range(10,110,10)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>>
```

## #Break ( breaks out of loop)

```
>>> y=1
>>> for x in range(4,256,4):
...     y = y * x
...     if y > 512:
...         break
...     print y
...
4
32
384
>>>
```

## #Continue ( continues with next operation )

```
>>> rivers
['Ganges', 'Nile', 'Mississippi', 'Amazon', 'Volga', 'Danube']
>>> for i in rivers:
...     if i== 'Amazon':
...         continue
...     else:
...         print i
...
Ganges
Nile
Mississippi
Volga
Danube
>>>
```

## #Pass ( null operation )

```
>>> for i in rivers:
...     if i== 'Amazon':
...         pass
...     else:
...         print i
...
Ganges
Nile
Mississippi
Volga
Danube
>>>
```



# Functions in Python

**Block of code that takes information (in the form of parameters), does some computation, and returns a new piece of information based on the parameter information.**

- Python, functions begin with the *def* keyword.
- Parameters in Python function are passed by reference

```
>>> students = {'1':{'name':'Alice','gender':'female','marks':83},
                '2':{'name':'Bob','gender':'male','marks':94},
                '3':{'name':'Charlie','gender':'male','marks':75},
                '4':{'name':'Diana','gender':'female','marks':87}}
>>> def averageGrade(students):
...     # "This function computes the average marks"
...     sum = 0.0
...     for key in students:
...         sum = sum + students[key]['marks']
...         average = sum/len(students)
...     return average
...
>>> avg = averageGrade(students)
>>> print " The average marks is:%0.2f"%(avg)
The average marks is:84.75
>>> █
```



# Handling Files in Python

**Files** : Python treats files as sequence of lines; Sequence operations work for the data read from files

Basic operations are : open, close, read and write

*open ( filename, mode)* – returns a file object

mode can be read(*r*), write(*w*), append(*a*); Mode is optional and Default is *r*

Example :

```
>> rivers = open('world_rivers','r') // reading from open file returns its content obtained a file object
```

```
>> rivers = open('world_rivers','w') // writing in an open file means overwriting the existing contents
```

*close(filename, mode)* – closes the file object, finishes any buffered operation

Example :

```
>> rivers = world_rivers.close() // closes the file once done with writing
```



# Handling Files in Python

```
>>> rivers = open ('world_rivers','w')
>>> rivers.write ('Ganges \n')
>>> rivers.write ('Nile \n')
>>> rivers.write ('Volga \n')
>>> rivers.write ('Mississippi \n')
>>> rivers.write ('Amazon \n')
>>> rivers.close()
>>> rivers
<closed file 'world_rivers', mode 'w' at 0x10742d810>
>>> countries = open( 'rivers_countries','w')
>>> countries.write ('India \n')
>>> countries.write ('Egypt \n')
>>> countries.write ('Russia \n')
>>> countries.write ('USA \n')
>>> countries.write ('Brazil \n')
>>> countries.close()
>>> countries
<closed file 'rivers_countries', mode 'w' at 0x10742d780>
>>> █
```



# Handling Files in Python

```
>>> r = open( 'world_rivers','r')
>>> c = open( 'rivers_countries','r')
>>> pn,pc = [],[]
>>> for i in r:
...     pc.append(i[:-1]) # ignore '\n'
...
>>> r.close()
>>> for i in c:
...     pn.append(i[:-1]) # ignore '\n'
...
>>> c.close()
>>> print(pc)
['Ganges ', 'Volga ', 'Nile ', 'Mississippi ', 'Amazon ']
>>> print(pn)
['India ', 'Russia ', 'Egypt ', 'USA ', 'Brazil ']
```



# Reading Sequential Files in Python

```
# This program makes a copy of one sequential file into another. It asks the user for the names of the files to read from and to write to:  
# A file must be opened --- this requires a new variable to hold the address in heap storage of the file's data-structure representation
```

```
>> input = open(inputfilename, "r")           # "r" means we will read from input filename  
>> output = open(outputfilename, "w")        # "w" means we will write to output filename
```

```
# You can read the lines one by one with a while loop :
```

```
>> line = input.readline()                   # read the next line of input --- this includes the ending \r and \n characters!  
                                             # The line is of course a string.
```

```
>> while line != " " :                       # When the input is all used up, line == ""  
    print line                               # Write the string to the output file  
    output.write(line)  
    line = input.readline()
```

```
# But here is the simplest way to read all the lines one by one :
```

```
>> for line in input :  
    print line  
    output.write(line)  
>> input.close()  
>> output.close()
```





# Exceptions in Python

Python reports to user about some unexpected event through **exceptions**

These events can be one of the following :

Access violation ( writing to a read only file )

Missing resources ( reading a non-existent file)

Type incompatibility ( multiplying two strings)

Bound violation ( accessing a string beyond limit)

Example :

```
>> x = open ('world_rivers', 'r')
```

```
>> x.close
```

```
>> x.read() ← Exception; Bad read, file already closed
```



# Exception Handling in Python

## try.....except.....else

**try:**

Operations that can raise exceptions.

**except** [Optional list of exceptions] :

In case of exceptions, do the recovery.

**else** : # else is optional

If no exception then do normal things

## try.....finally

**try :**

Operations that can raise exceptions.

**finally :**

Execute irrespective of whether exception was raised or not. Typically clean-up stuff

```
# Program to depict else clause with try-except
```

```
# Function which returns a/b
```

```
def example(a , b):
```

```
    try:
```

```
        c = ((a+b) // (a-b))
```

```
    except ZeroDivisionError:
```

```
        print ("a/b result in 0")
```

```
    else:
```

```
        print (c)
```

```
# Driver program to test above function
```

```
example(2.0, 3.0) ← -5.0
```

```
example(3.0, 3.0) ← a/b result in 0
```



# Modules in Python

**Organizing programs for easier maintenance and access**

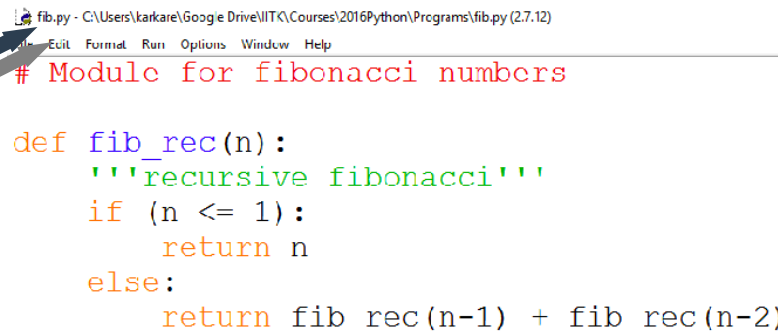
**Reuse same functions across programs without copying its definition in each program**

**Python allows putting definition in a file – use them in a script or in an interactive instance**

**Such a file is called module - definitions from modules can be imported in other modules or main module**

**The file name is the module name with .py appended**

**Within a module, the module's name is available as global variable `__name__`**



```
fib.py - C:\Users\karkare\Google Drive\IITK\Courses\2016Python\Programs\fib.py (2.7.12)
File Edit Format Run Options Window Help
# Module for fibonacci numbers

def fib_rec(n):
    '''recursive fibonacci'''
    if (n <= 1):
        return n
    else:
        return fib_rec(n-1) + fib_rec(n-2)
```

File name : fib.py  
Module name : fib




# Module Example

```
>>> def fib_rec(n):
...     # " recursive fibonacci"
...     if(n<=1):
...         return n
...     else:
...         return fib_rec(n-1)+fib_rec(n-2)
...
>>> def fib_iter(n):
...     # " iterative fibonacci"
...     cur,nxt= 0,1
...     for k in range(n):
...         cur,nxt = nxt,cur+nxt
...     return cur
...
>>> def fib_upto(n):
...     # " given n returns list of fibonacci"
...     # numbers < n"
...     cur,nxt= 0,1
...     lst = []
...     while(cur<n):
...         lst.append(cur)
...         cur,nxt = nxt,cur+nxt
...     return lst
...
>>> fib_rec(5)
5
>>> fib_rec(10)
55
>>> fib_upto(5)
[0, 1, 1, 2, 3]
>>> fib_upto(10)
[0, 1, 1, 2, 3, 5, 8]
>>> fib_iter(5)
5
```

```
>>> import fib
>>> fib.fib_upto(5)
[0, 1, 1, 2, 3]
```

```
>>> fib.fib_rec(10)
55
>>> fib.fib_iter(20)
6765
```

```
>>> fib.__name__
'fib'
```



Within a module, the module's name is available as the value of the global variable **`__name__`**.



# Module Example

## Importing special functions

```
>>> from fib import fib_upto
>>> fib_upto(6)
[0, 1, 1, 2, 3, 5]
>>> fib_iter(1)
```

```
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    fib_iter(1)
NameError: name 'fib_iter' is not defined
```

```
>>> from fib import *
>>> fib_upto(6)
[0, 1, 1, 2, 3, 5]
>>> fib_iter(8)
21
```

Exports all names except those beginning with underscore



# `__main__` in Modules

When you run a module on the command line with : `python fib.py <arguments>`  
the code in the module will be executed in the module just as if you imported it but with the `__name__` set to "`__main__`".

By adding the code at the end of your module, you can make the file usable as script as well as importable module

```
if __name__ == "__main__":
```

## Example:

```
if __name__ == "__main__":  
    import sys  
    print (fib_iter(int(sys.argv[1])))
```

- This code parses the command line only if the module is executed as the “*main*” file :

```
$ python fib.py 10
```

```
55
```

- If the module is imported, the code is not run : `>>> import fib`



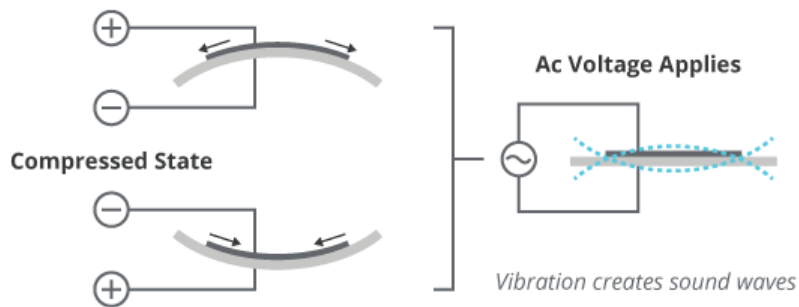
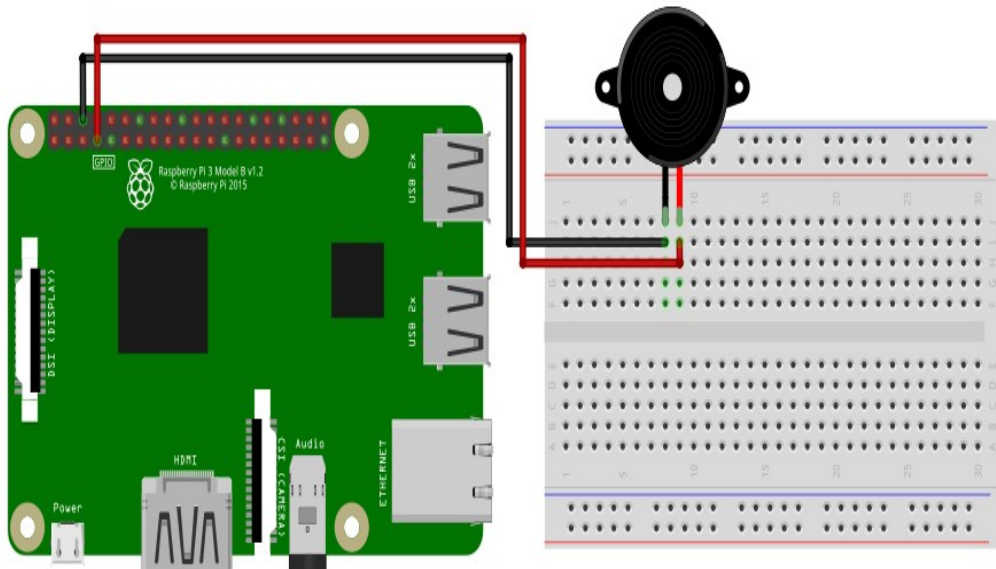
# Back to Raspberry Pi Using Python



Internet of Things  
Instructor : Dr. Bibhas Ghoshal

Spring 2022

# Connecting Rpi to Piezo Buzzer

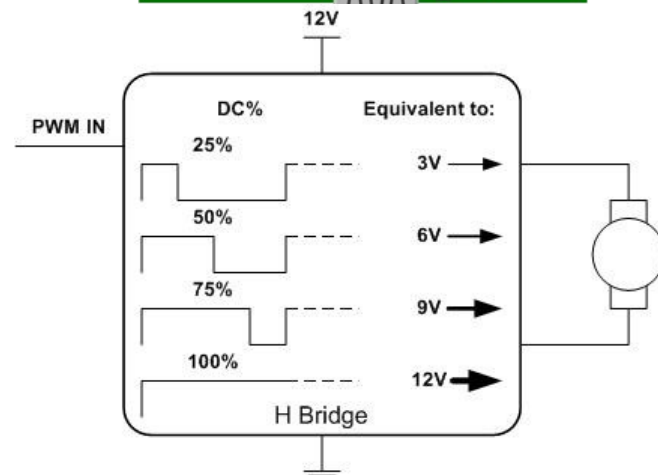
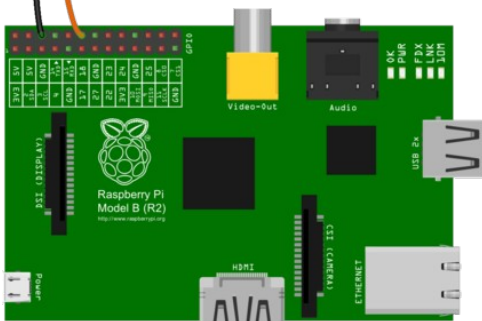
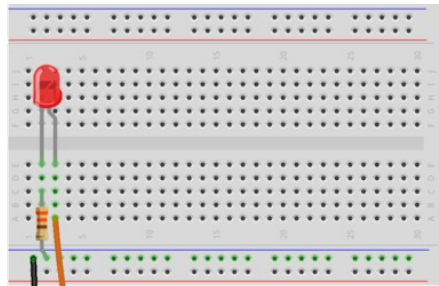


```
import time
import RPi.GPIO as gpio
gpio.setwarnings(False)
gpio.setmode(gpio.BOARD)
gpio.setup(7, gpio.OUT)
try:
    while True:
        gpio.output(7, 0)
        time.sleep(.3)
        gpio.output(7, 1)
        time.sleep(.3)
except KeyboardInterrupt:
    gpio.cleanup()
    exit
```

Python program which instructs the Rpi to control a piezoelectric buzzer



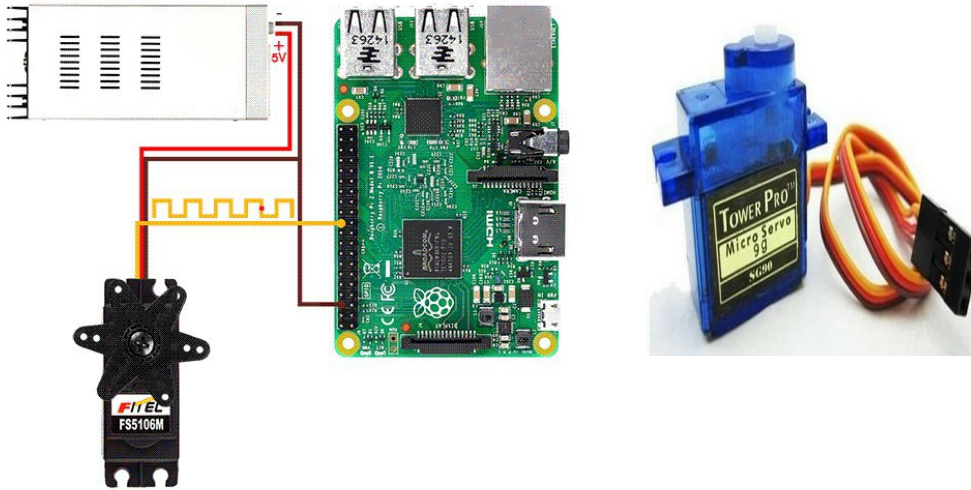
# Rpi used to control intensity of a LED



```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
p = GPIO.PWM(18, 50)
p.start(0)
try:
    while True:
        for i in range (100):
            p.ChangeDutyCycle(i)
            time.sleep(0.02)
        for i in range(100):
            p.ChangeDutyCycle(100-i)
            time.sleep(0.02)
except KeyboardInterrupt:
    pass
p.stop()
GPIO.cleanup()
```

Python program which instructs the Rpi to fade a LED

# Using Rpi to control a Servo Motor

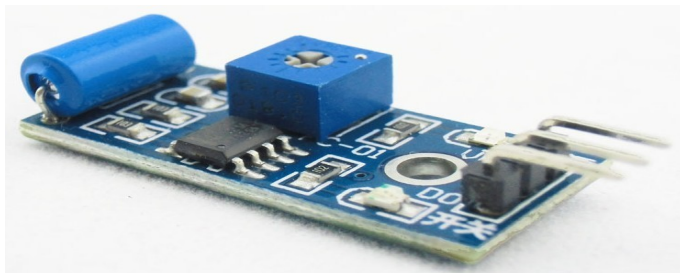
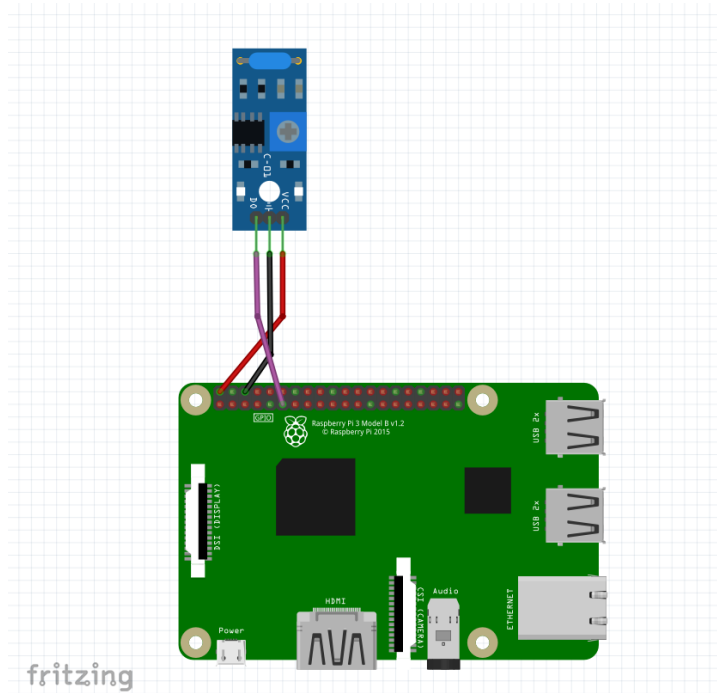


```
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(22, GPIO.OUT)
p = GPIO.PWM(22, 50)
p.start(7.5)
while True:
    p.ChangeDutyCycle(7.5)
    time.sleep(1)
    p.ChangeDutyCycle(12.5)
    time.sleep(1)
    p.ChangeDutyCycle(2.5)
    time.sleep(1)
```

Python program which controls a Servo motor

| Position    | Duty Cycle |
|-------------|------------|
| 0 degree    | 2.5        |
| 90 degrees  | 7.5        |
| 180 degrees | 12.5       |

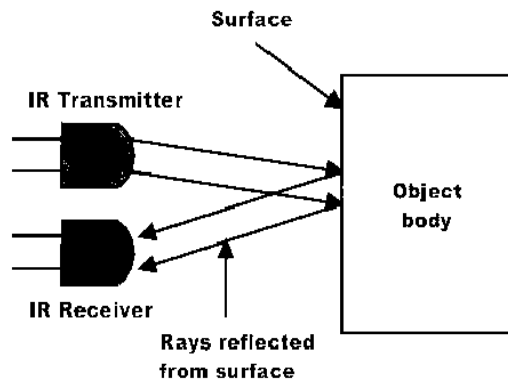
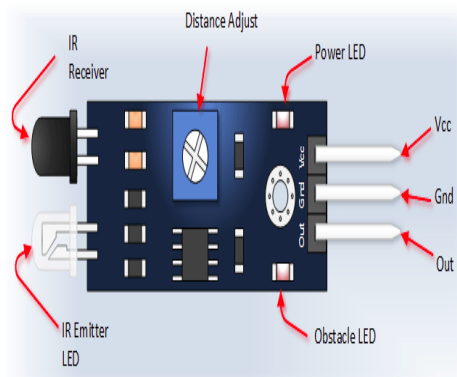
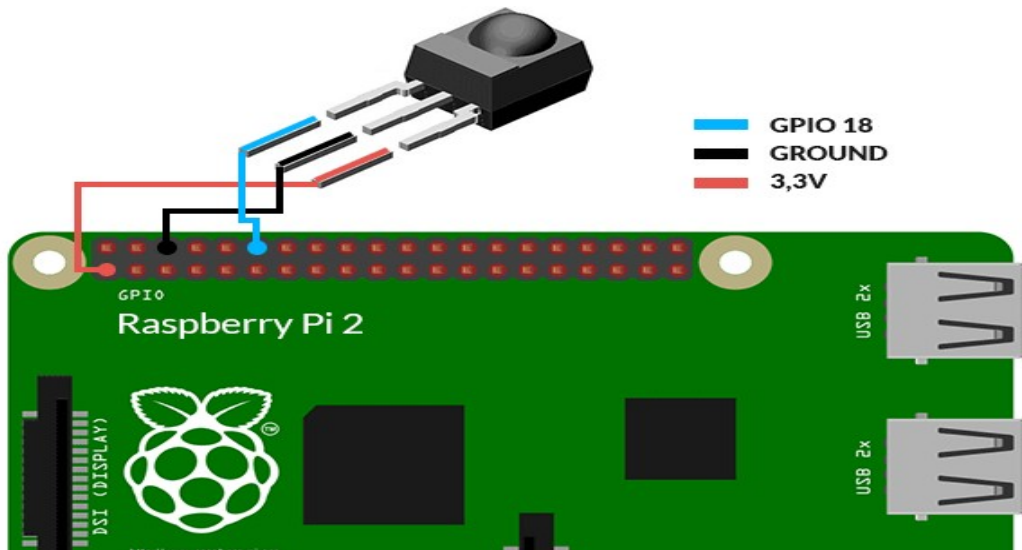
# Using Rpi to control a Vibration Sensor



```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(18,GPIO.IN)
try:
    while True:
        j, i = 0, 0
        i = GPIO.input(18)
        if i==1:
            if j==0:
                print("Vibration...")
                j=1
                time.sleep(0.1)
            elif i==0:
                if j==1:
                    print("Vibration...")
                    j=0
                    time.sleep(0.1)
except KeyboardInterrupt:
    GPIO.cleanup()
```

Python program to control a vibration sensor

# Connecting Rpi to Infrared Sensor



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

```
import time
```

```
Sensor = 16
```

```
buzzer = 18
```

```
GPIO.setmode(GPIO.BOARD)
```

```
GPIO.setup(sensor,GPIO.IN)
```

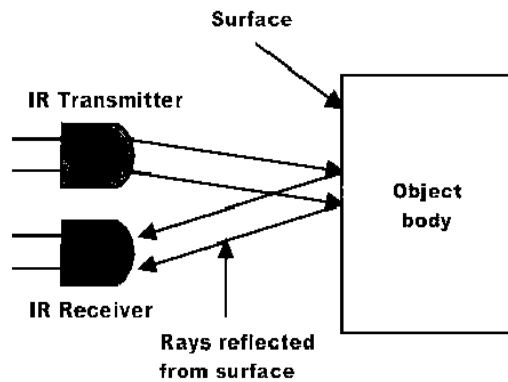
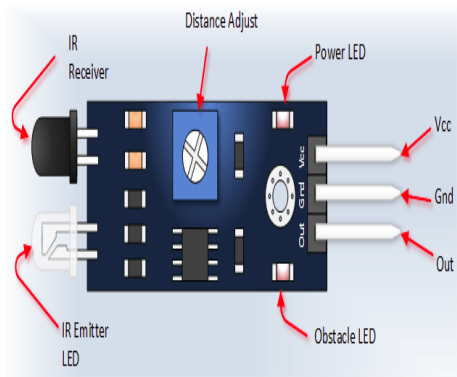
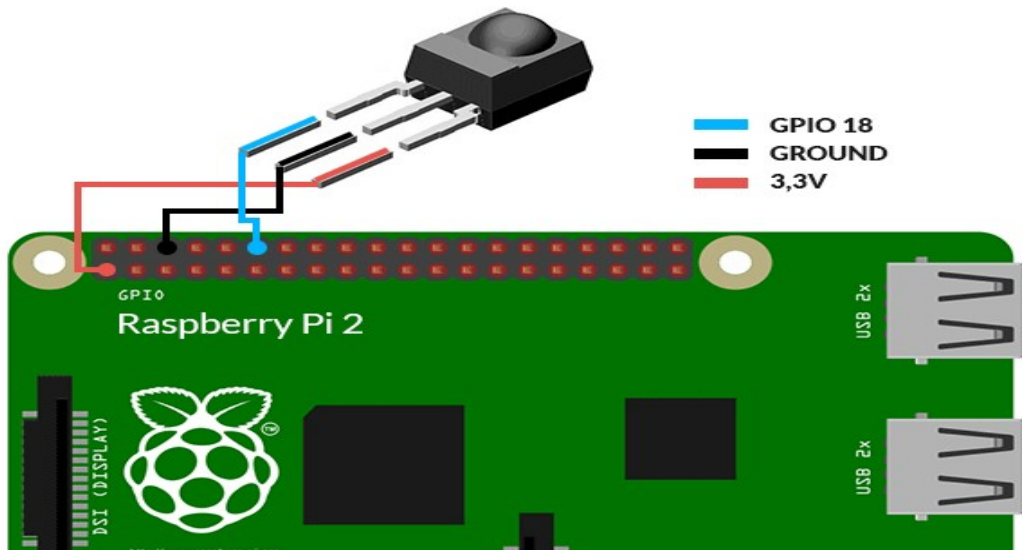
```
GPIO.setup(buzzer,GPIO.OUT)
```

```
GPIO.output(buzzer,False)
```

```
print "IR Sensor Ready....."
```

```
Print " "
```

# Connecting Rpi to Infrared Sensor



try:

while True:

if GPIO.input(sensor):

GPIO.output(buzzer,True)

print "Object Detected"

while GPIO.input(sensor):

time.sleep(0.2)

else:

GPIO.output(buzzer,False)

except KeyboardInterrupt:

GPIO.cleanup()

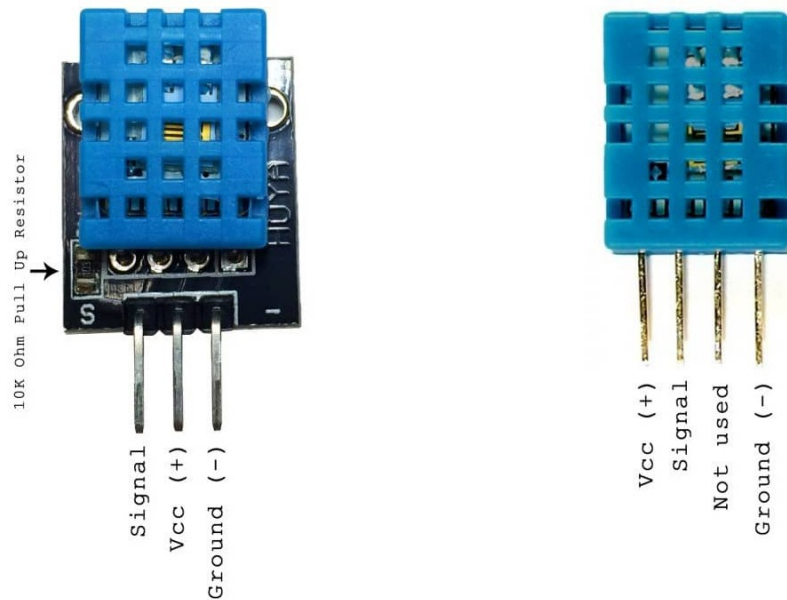




# Temperature and Humidity Sensors - DHT11

Temperature and Humidity Sensors are frequently used in soil monitors, home environment, weather stations. They are simple to program.

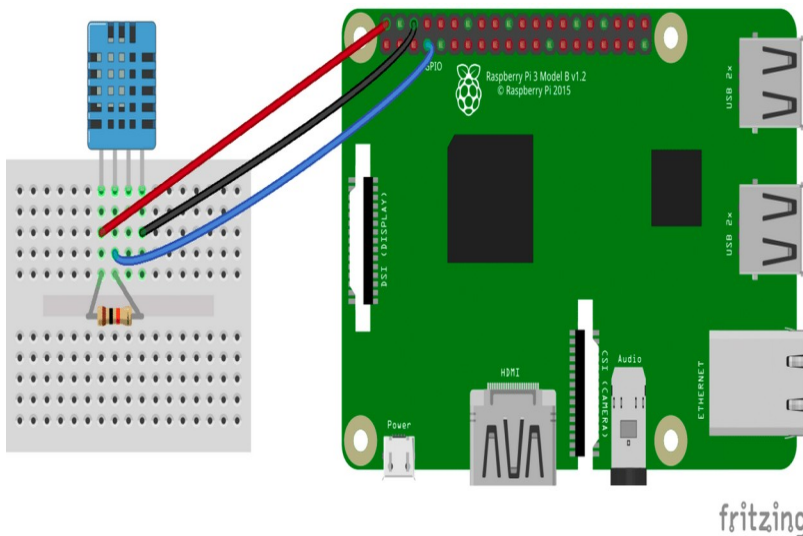
DHT11 is a sensor circuit that contains surface mounted thermistor and humidity sensor. The circuit converts the resistance measurements from the thermistor and the humidity sensor to digital outputs



# Interfacing Rpi to DHT11

## Using the Adafruit DHT 11 library

1. `git clone https://github.com/adafruit/Adafruit_Python_DHT.git`
2. `cd Adafruit_Python_DHT`
3. `sudo apt-get install build-essential python-dev`
4. `sudo python setup.py install`



```
#!/usr/bin/python
import sys
import Adafruit_DHT

while True:
    hum, temp = Adafruit_DHT.read_retry(11, 4)
    print('Temp:0.1fC Humidity: 0.1f%' %(temp, hum))
```

Python program that instructs the Rpi to read data from the DHT11 module and print it

