Tutorial 3 : Networking with Python| Learning Python on the Go....

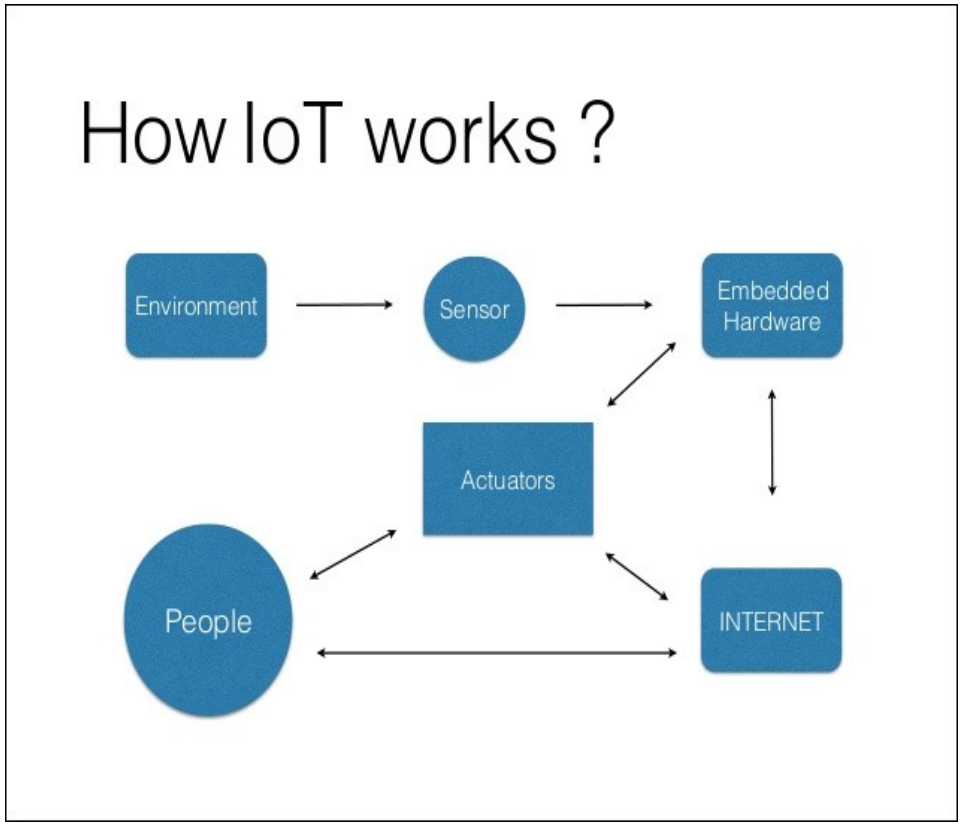
Dr. Bibhas Ghoshal

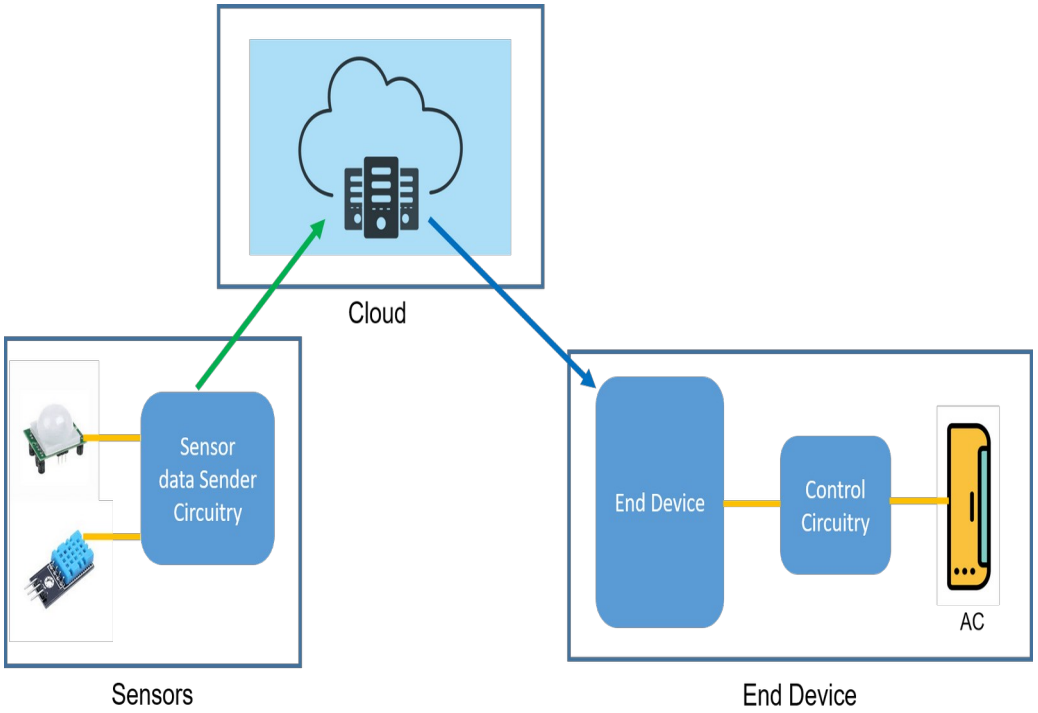
Assistant Professor

Department of Information Technology

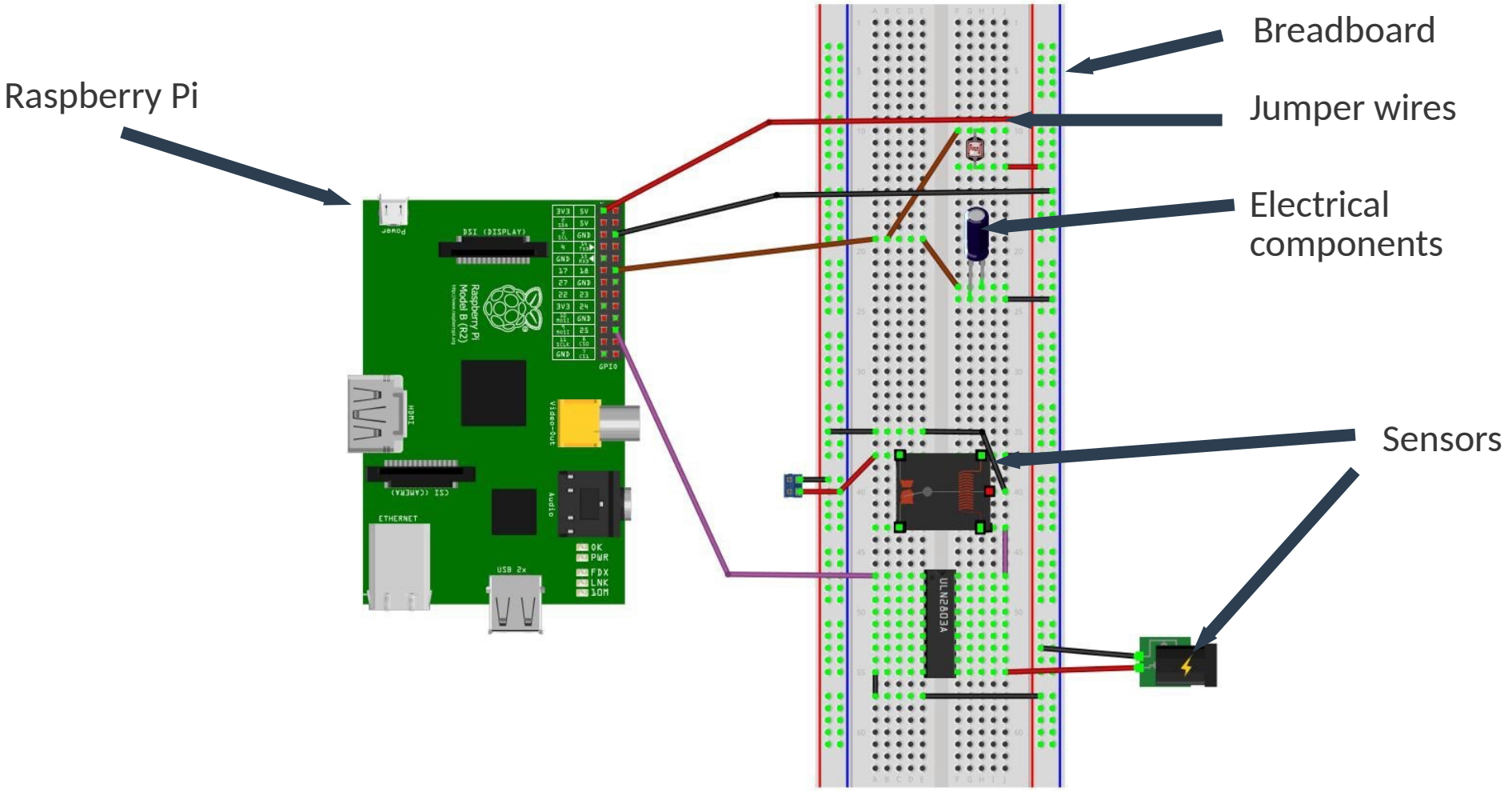
Indian Institute of Information Technology Allahabad

IoT – Basic Idea





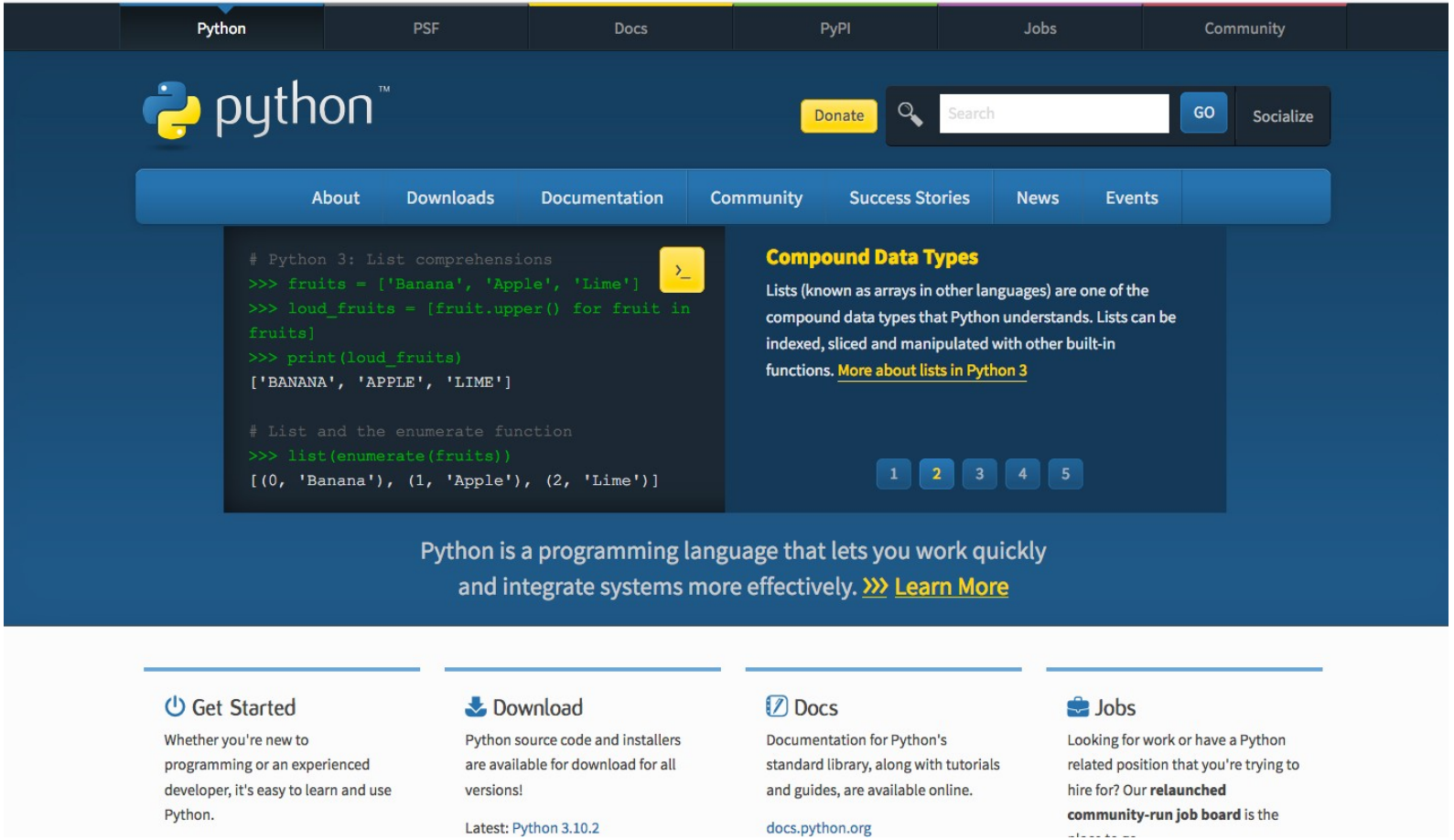
Recall :
Interfacing sensors to the computing unit



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

Vehicular Language : Python
Installation, Guide, Docs....

www.python.org



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

Module import

GPIO pin setup

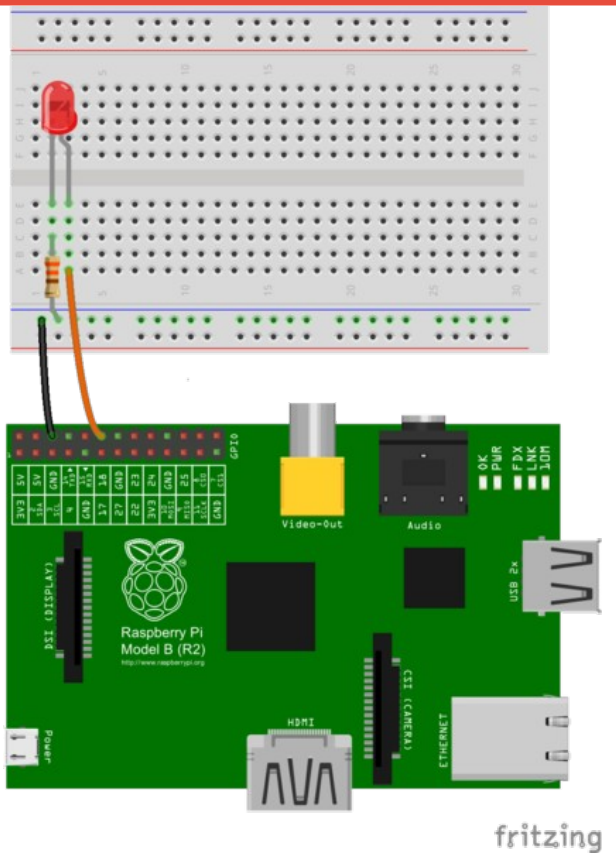
Execute in loop

Send a HIGH signal to pin 18
Wait for 1 time unit

Send a LOW signal to pin 18
Wait for 1 time unit

```
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)

Interfacing Rpi to DHT11

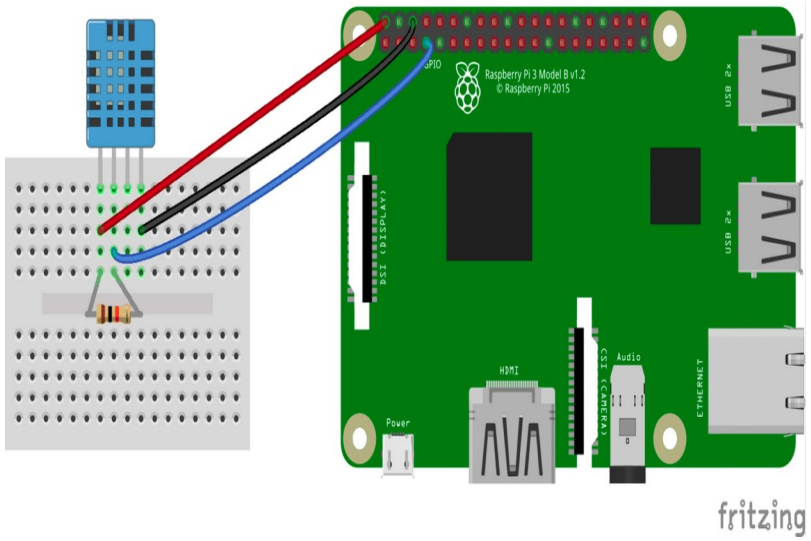
Using the Adafruit DHT 11 library

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*

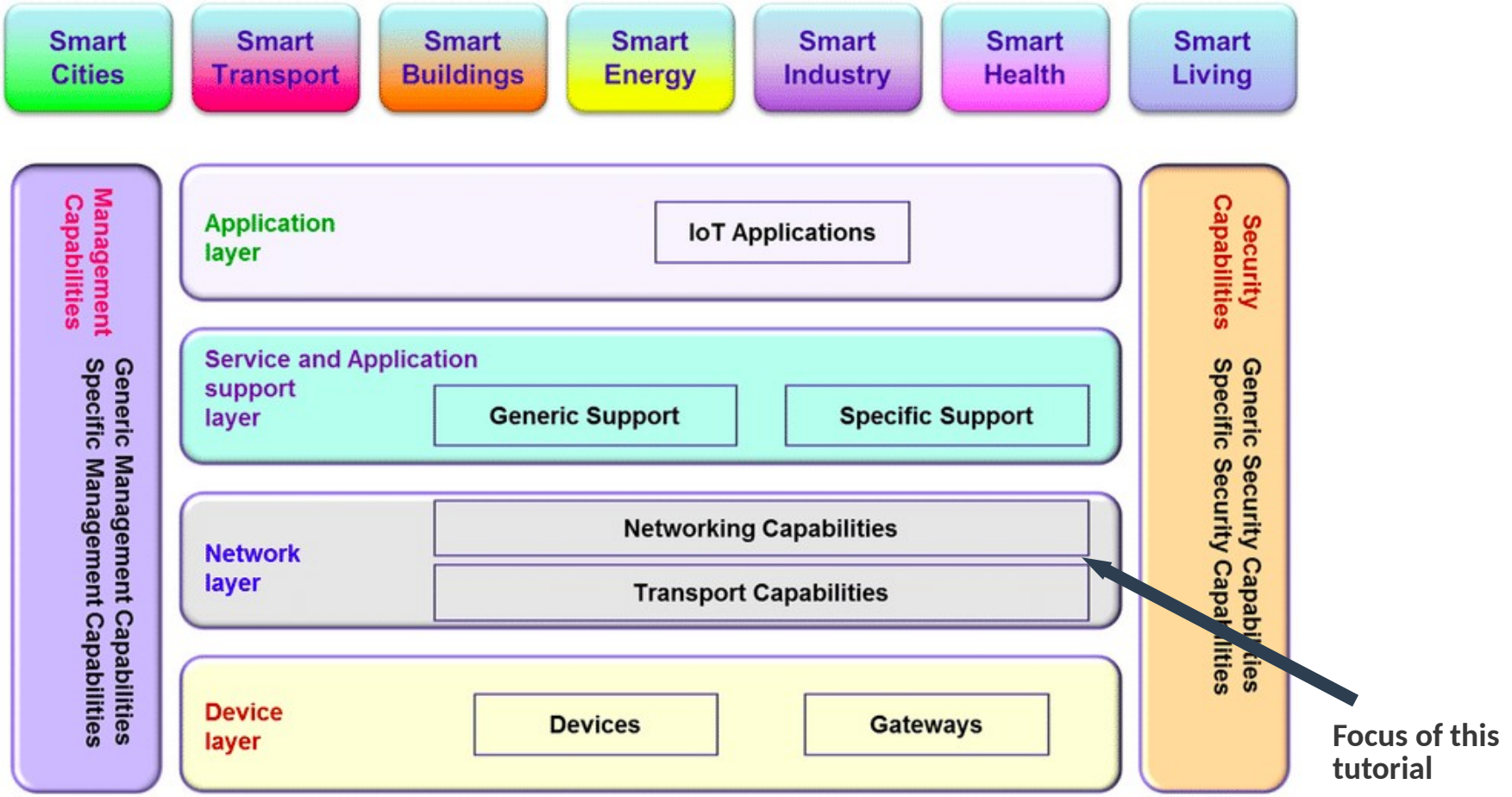
4. *sudo python setup.py install*




```
#!/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)
```

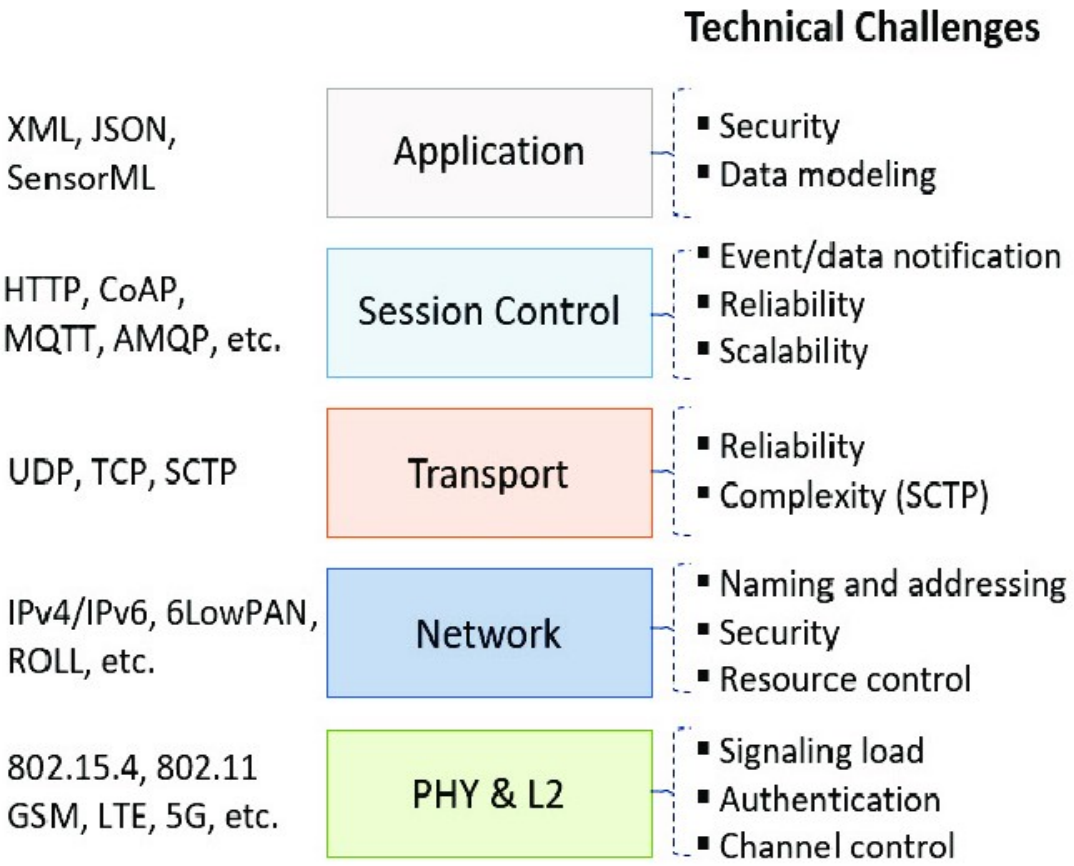
Python program that instructs the Rpi to read data from the DHT11 module and print it

The Four Layered IoT Architecture



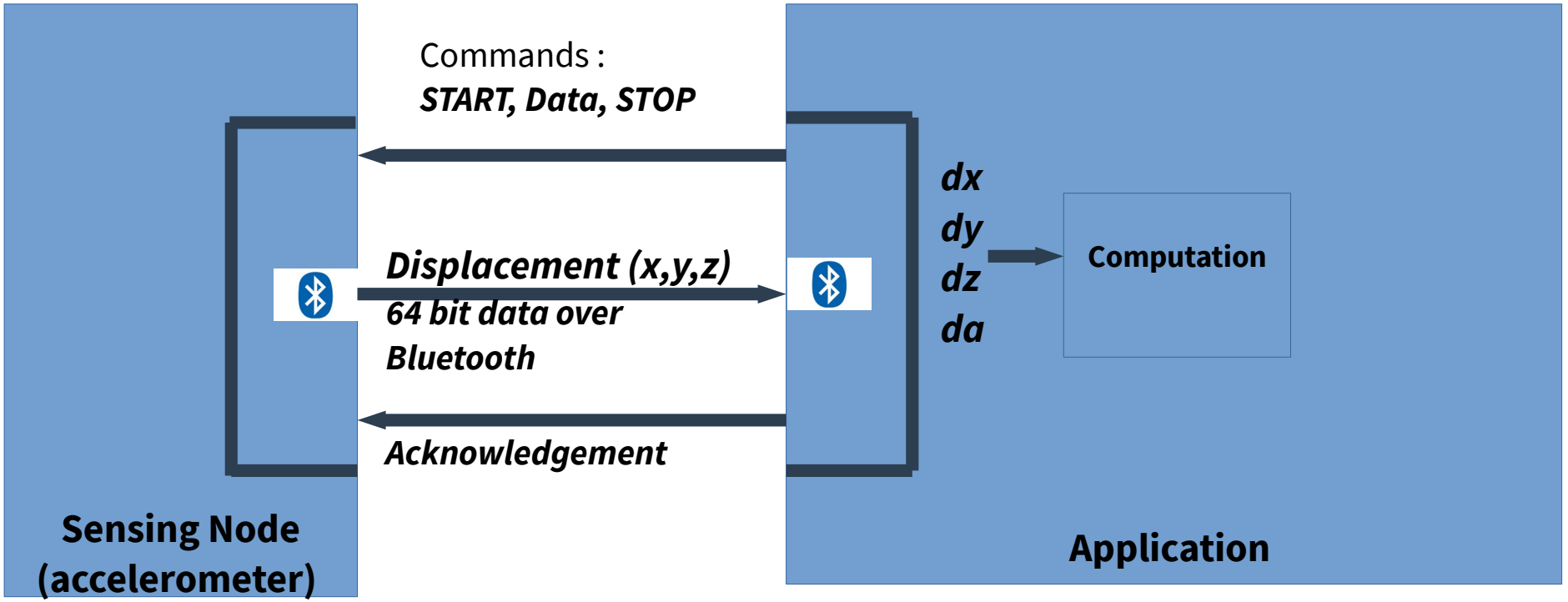
Source : ITU-T

IoT Protocol Stack



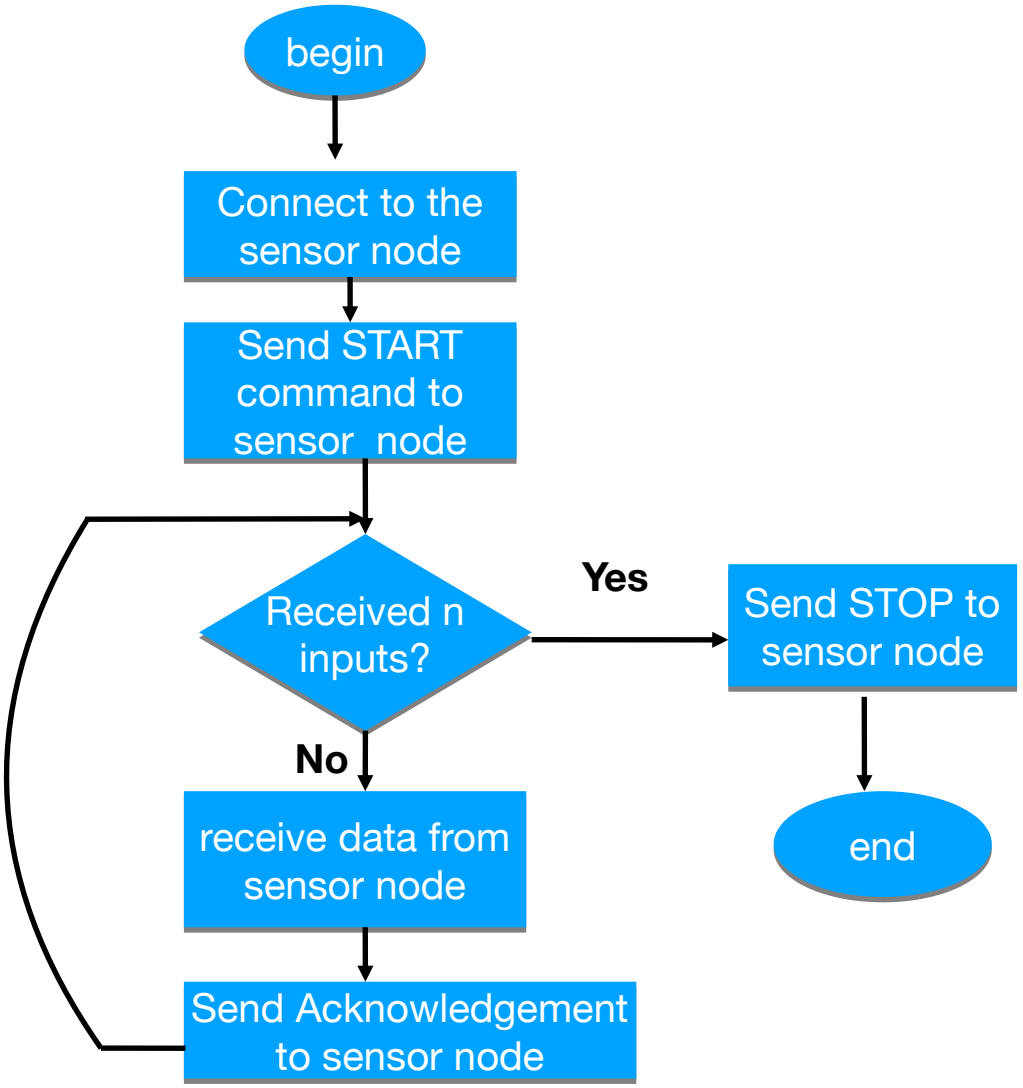
Source : Internet

Task : An application reads sequence of data inputs via some wireless protocol ex. Bluetooth

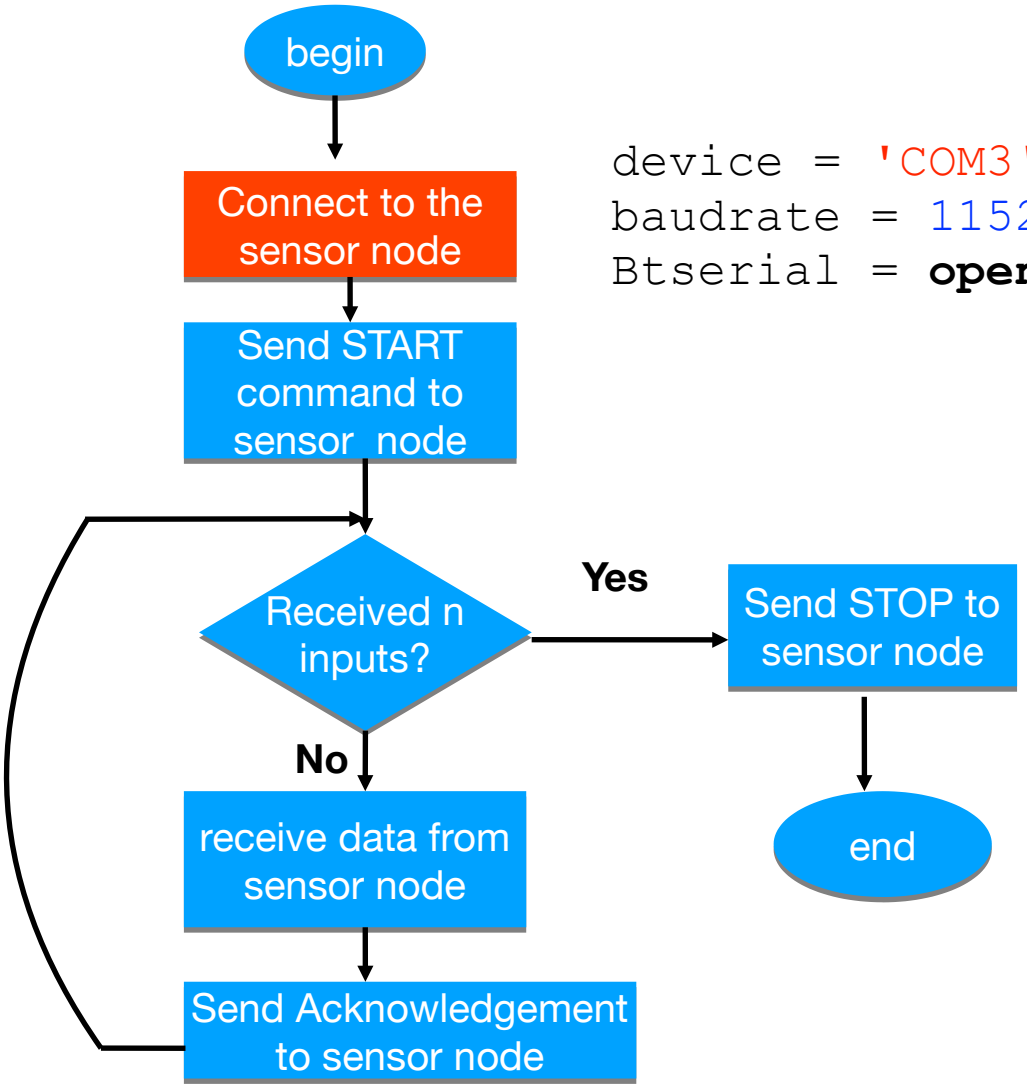


1. **Start** – Start receiving data
2. **Acknowledgment** – send acknowledgement about the last data packet
3. **Data** – sensing node sends data packet containing displacement and orientation information
4. **Stop** – stop processing in the sensing node

Flow Chart showing Data Transfer

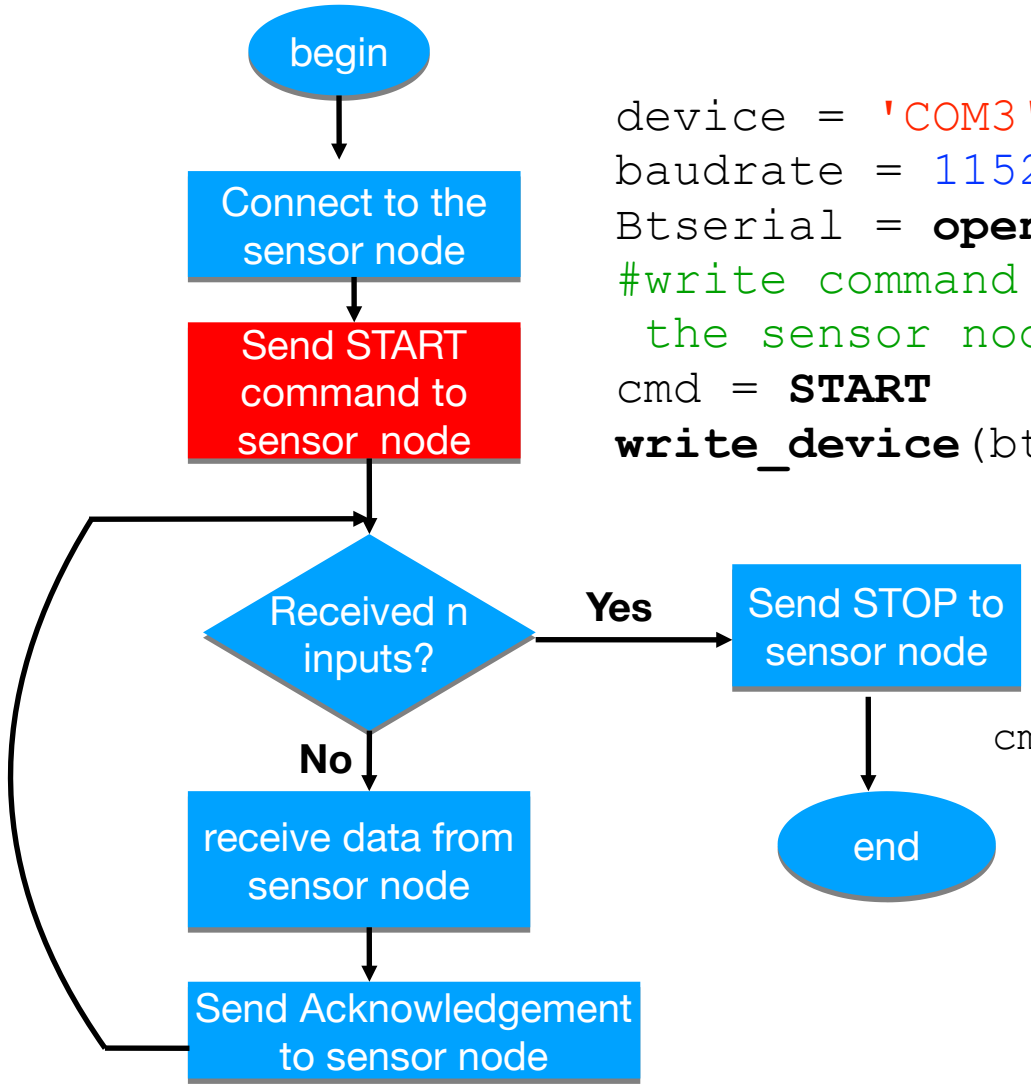


Flow Chart showing Data Transfer
Step 1 : Initialization




```
device = 'COM3'
baudrate = 115200
Btserial = open_device(device,baudrate)
```

Flow Chart showing Data Transfer
Step 2 : setting up the sensor node – send cmd

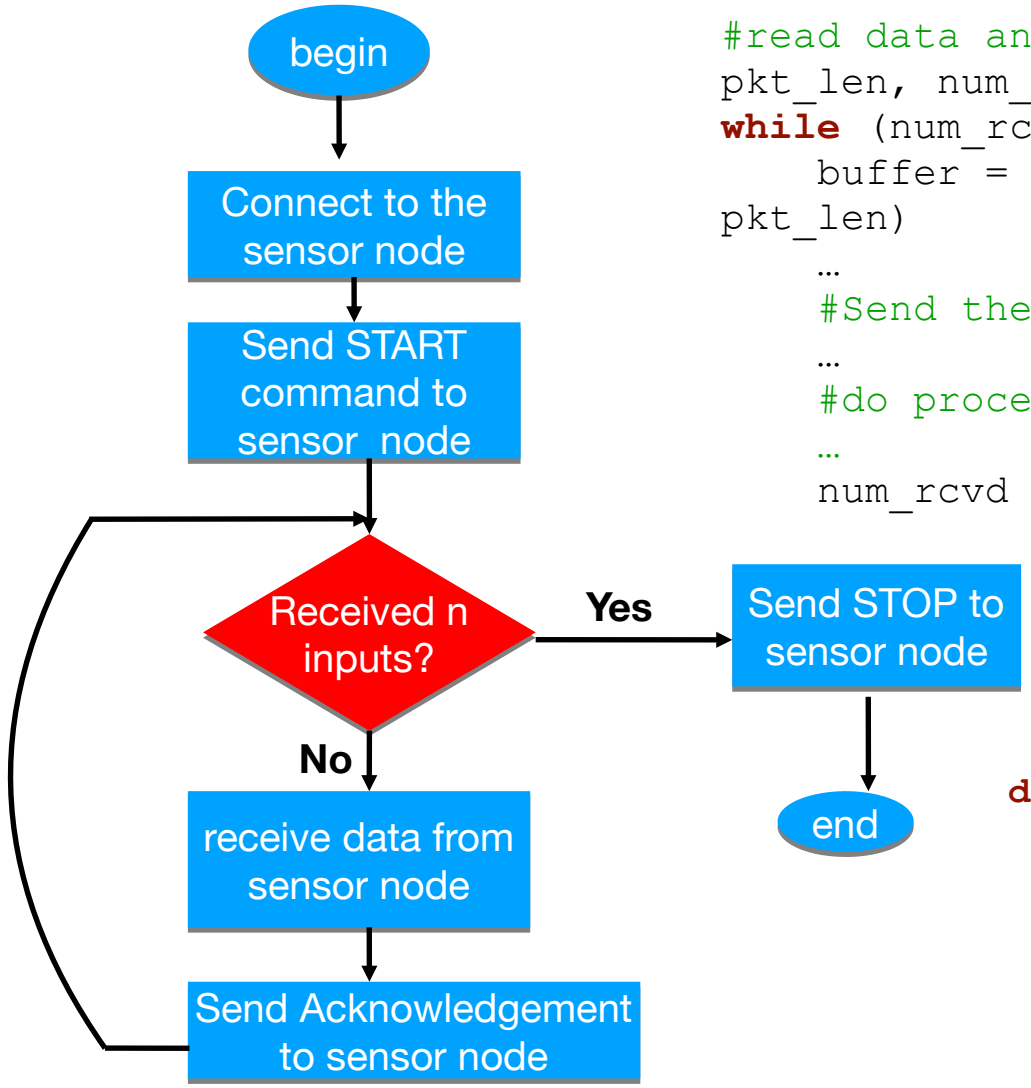



```
device = 'COM3'
baudrate = 115200
Btserial = open_device(device,baudrate)
#write command to start interaction with
 the sensor node
cmd = START
write_device(btserial,cmd)
```

```
cmd = header, payload,checksum
```

Header is a single byte command ID
Payloads could be multiple bytes command
Last two bytes in any command are checksum.

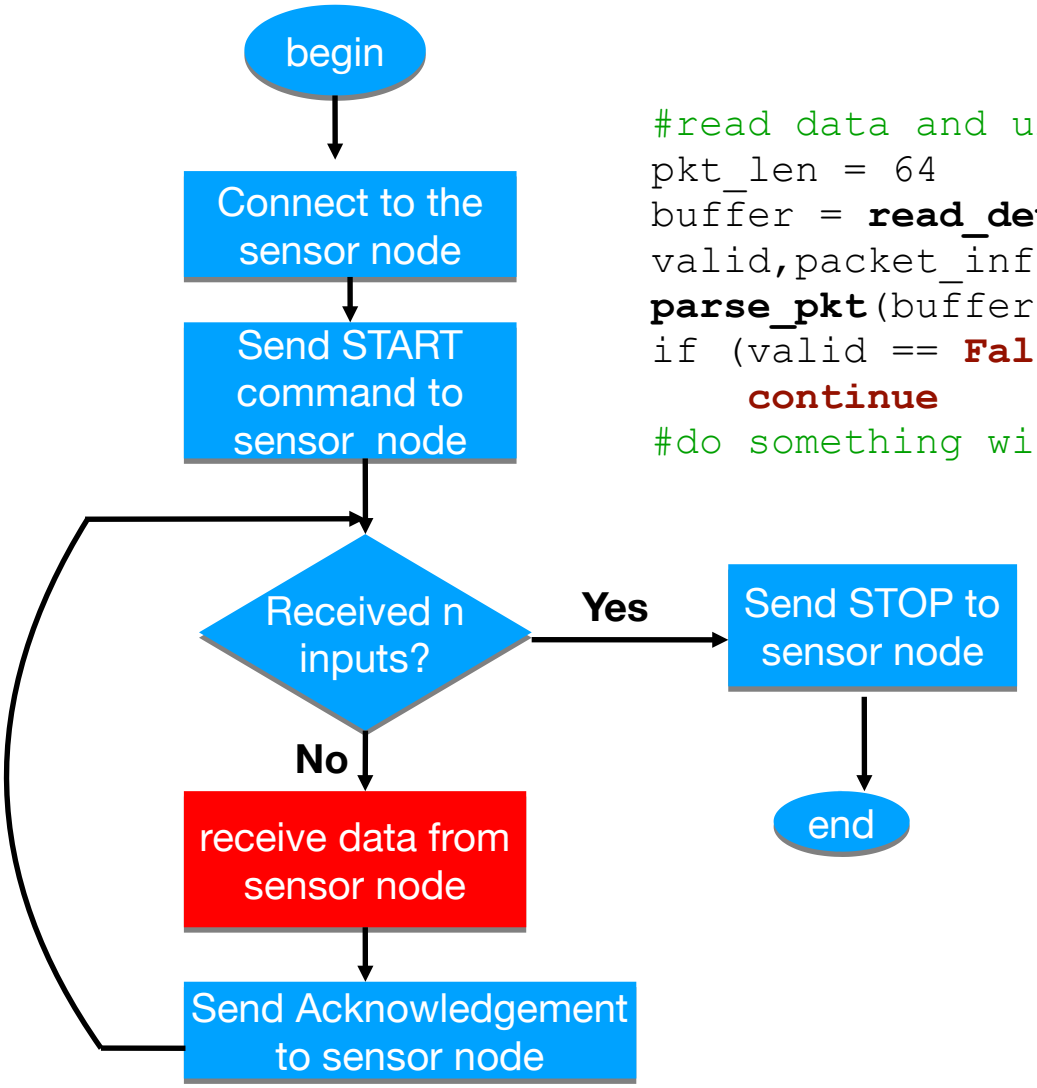
Flow Chart showing Data Transfer
Step 3 : Check the number of packets received




```
#read data and understand it
pkt_len, num_rcvd = 64, 0
while (num_rcvd < 200):
    buffer = read_device(btserial,
pkt_len)
    …
    #Send the ack to sensor node
    …
    #do processing of the payload
    …
    num_rcvd += 1
```

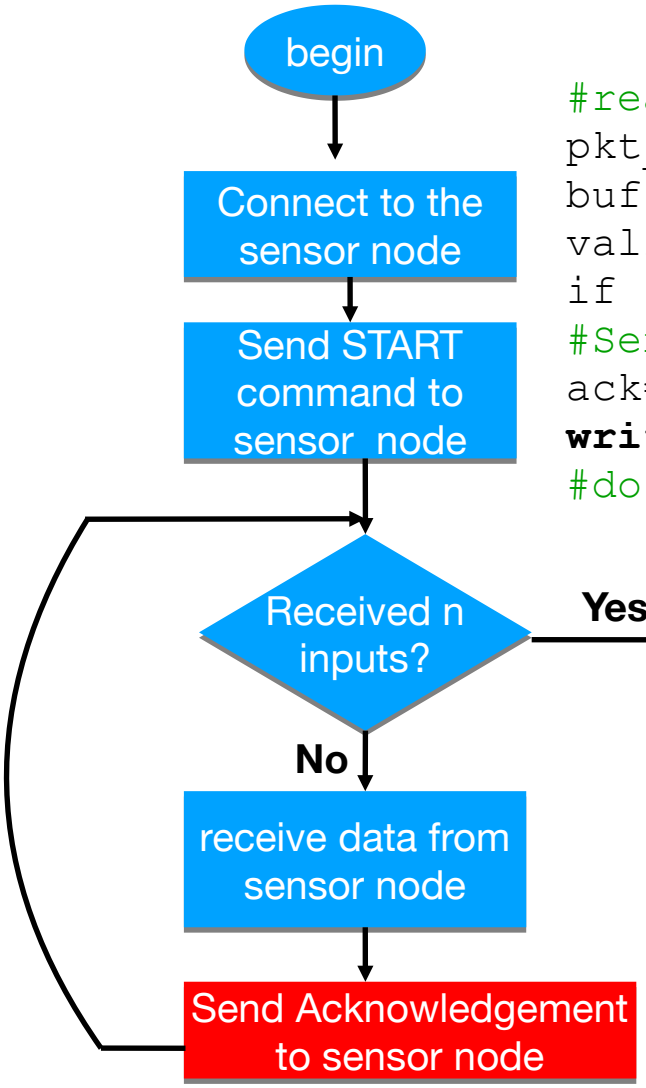
```
def read_device(device,length):
    buffer = [ ]
    device.flushInput()
    buffer = device.read(length)
    return buffer
```

Flow Chart showing Data Transfer
Step 4 : Receive data packets from sensor node




```
#read data and understand it
pkt_len = 64
buffer = read_device(btserial, pkt_len)
valid,packet_info,payload =
parse_pkt(buffer)
if (valid == False):
    continue
#do something with the info and payload
```

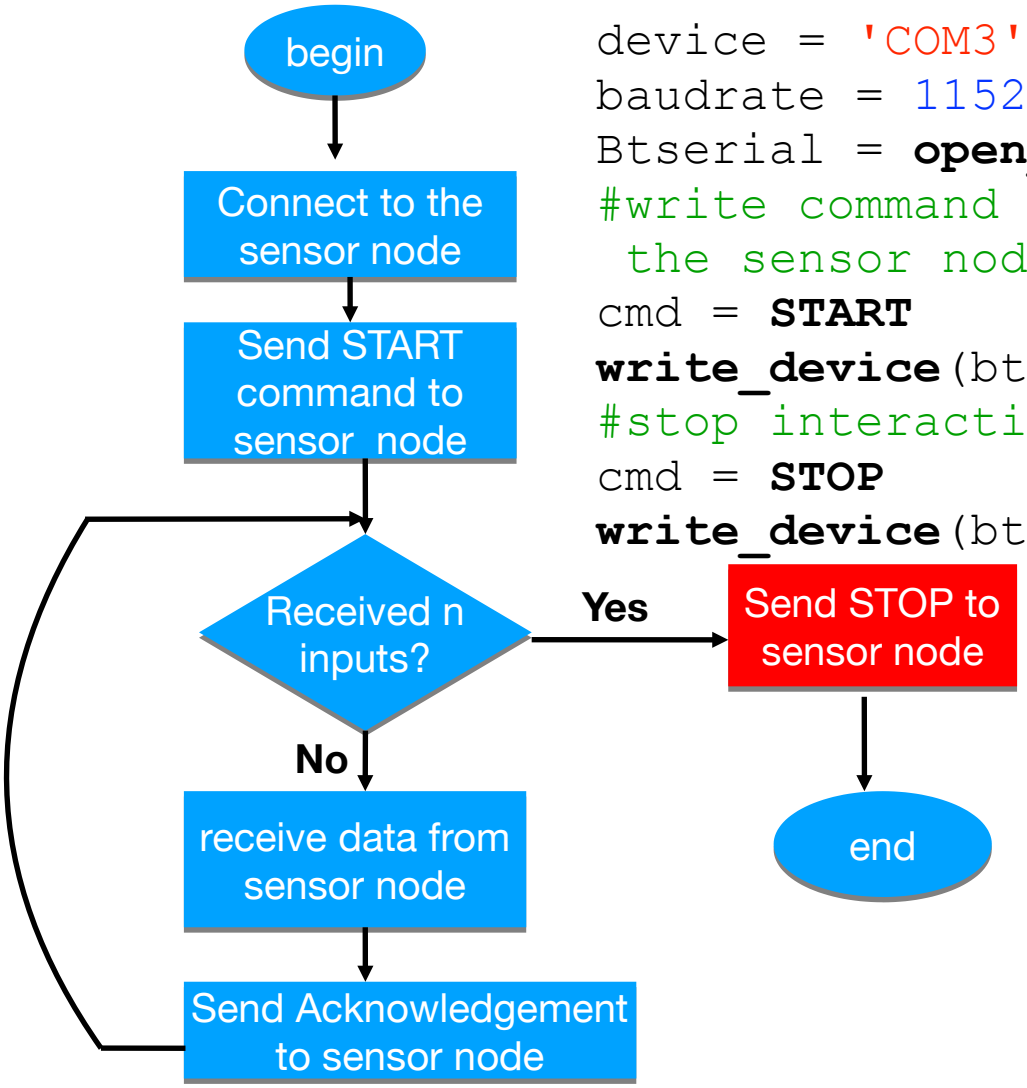
Flow Chart showing Data Transfer
Step 5 : Acknowledgement sent to sensor node




```
#read data and understand it
pkt_len = 64
buffer = read_device(btserial, pkt_len)
valid,packet_info,payload = parse_pkt(buffer)
if (valid == False): continue
#Send the ack to the sensor node
ack=create_ack(packet_info)
write_device(btserial, ack)
#do processing of the payload
```

```
#read data and understand it
pkt_len = 64
buffer = read_device(btserial, pkt_len)
valid,packet_info,payload =
parse_pkt(buffer)
if (valid == False):
    continue
#do something with the info and payload
```

Flow Chart showing Data Transfer
Step 6 : Send STOP command to the node



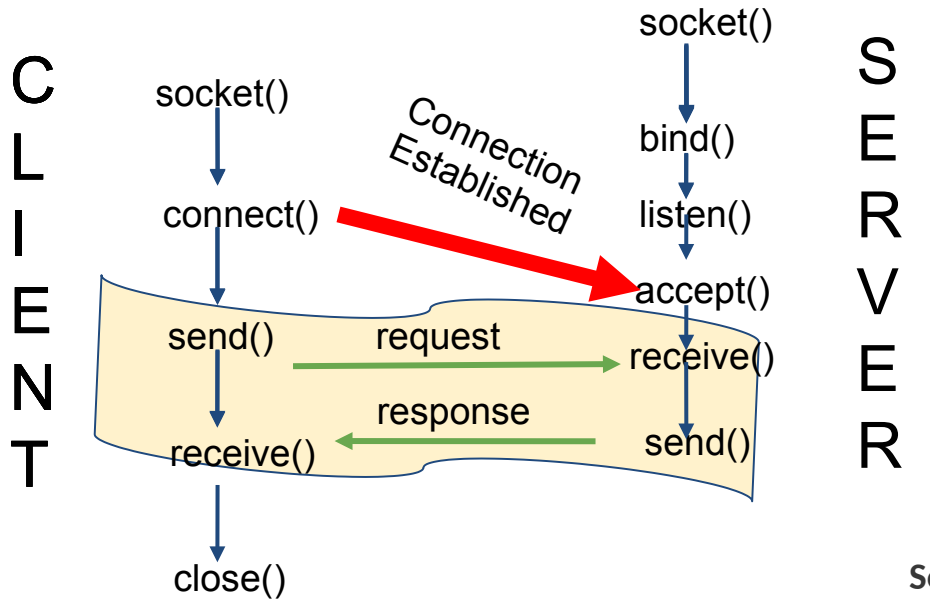

```
device = 'COM3'
baudrate = 115200
Btserial = open_device(device,baudrate)
#write command to start interaction with
 the sensor node
cmd = START
write_device(btserial,cmd)
#stop interacting with the sensor node
cmd = STOP
write_device(btserial,cmd)
```

Sockets

Socket is a software object that acts as an end point of a bidirectional communication link between server and client programs on network

Support in the operating system allows to implement clients and servers for both TCP and UDP communication.

Python provides network service for server-client model.
Python has additionally libraries which allow higher access to specific application level network protocols



Socket Programming (TCP)

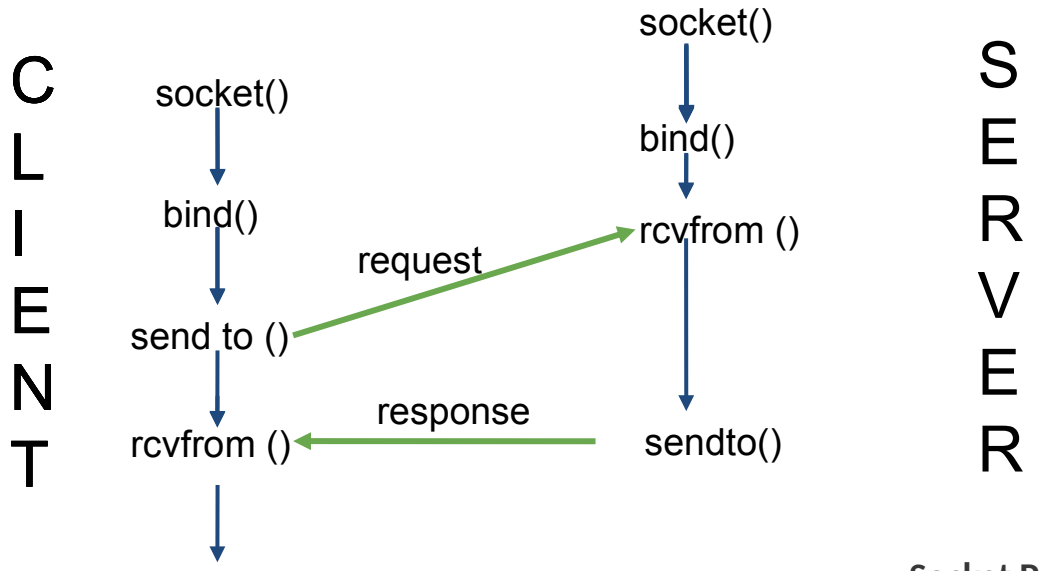
Sockets

Socket is a software object that acts as an end point of a bidirectional communication link between server and client programs on network

Sockets have two primary properties controlling the way they send data: the **address family** controls the OSI network layer protocol used and the **socket type** controls the transport layer protocol.

Support in the operating system allows to implement clients and servers for both TCP and UDP communication.

Python provides network service for server-client model.
Python has additionally libraries which allow higher access to specific application level network protocols



Socket Programming (UDP)

Ports

Port is a number to identify the sender or receiver of message.

Each host has 65,536 ports

Ports 0-1023 are reserved.

Protocol	Port #	Protocol	Port #
FTP	20	SMTP	25
SSH	22	HTTP	80
TELNET	23	HTTPS	443

Ports above 1024 are reserved for user processes.

Socket must bound to port n server.

Creating a Socket

Getting Host Information

`gethostbyname(hostname)` : Provide the IP address associated with a hostname

`gethostbyname_ex(hostname)`

```
>>> import socket
>>> socket.gethostbyname('localhost')
'127.0.0.1'
```

`socket(family, type)` : Opens a socket, **family** is one of **AF_UNIX** or **AF_INET** type is one of **SOCK_DGRAM** or **SOCK_STREAM**

```
#  TCP socket
S = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
```

Useful Functions on Socket

`accept()` : accept a connection, returning new socket and client address

`bind(addr)` : bind the socket to a local address

`close()` : close the socket

`connect(addr)` : connect the socket to a remote address

`listen(n)` : start listening for incoming connections

`recv(buflen[, flags])` : receive data

`send(data[, flags])` : send data, may not send all of it

TCP Server

The socket module

Provides access to low-level network programming functions.

Example: A server that receives data and sends ack

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #  TCP socket
s.bind(("",8888))     # Bind to local ip and port 8888
s.listen(5)           # Start listening, hold upto 5 connections
while 1:
    client, addr = s.accept()     # Wait for a connection
    print "Got a connection from ", addr
    data = client.recv(1024      # receive client's data upto 1024 byte
    print "Received data from client:", data
    client.send("Acknowledgement for received data") # Send ack back
    client.close()
```

TCP Client

The Client Program Connects to server, sends data and waits for ack

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                                  # Create TCP socket
s.connect(("127.0.0.1",8888))     # Connect to server
s.send("Hello Server..")          # Send message to server
msg = s.recv(1024)                # Receive up to 1024 bytes
s.close()                         # Close connection
print "Server's message:", msg
```

Problems with the TCP Server

Lots of repetitive coding

Not really handling multiple (concurrent) clients

Clients are queued

Handled in sequential way

Real world server would need concurrency

Even more coding!

Socket Server

Python module to simplify creation of realistic servers

Functionality to support basic network services such as HTTP, SMTP

from `help(SocketServer)`

Supports various server classes:

```
TCPServer(address, handler)
UDPServer(address, handler)
ForkingTCPServer(address, handler)
ForkingUDPServer(address, handler)
ThreadingTCPServer(address, handler)
ThreadingUDPServer(address, handler)
```

Need to define request handler, Derive from **BaseRequestHandler** class
Minimum requirement: define `handle()` method
`handle()` called each time a connection request is received

Useful Python Modules for Networking

ftplib - for connecting to a ftp server

smtplib - for sending emails

urllib - To download contents of a URL

httplib - communication over HTTP

poplib/imaplib - email access protocols

JavaScript Object Notation (JSON) – read and write data interchange format
Two structures – i. collection of name, value pairs (Dictionary)
ii. ordered list of values (List)
Usage : serializing and transmtting structured data over network
Example : transmitting data between server and web application

Extensible Markup Language (XML) - data format for structured document interchange

Example of a JSON and XML Format

JSON Format

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

The same text expressed as XML:

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

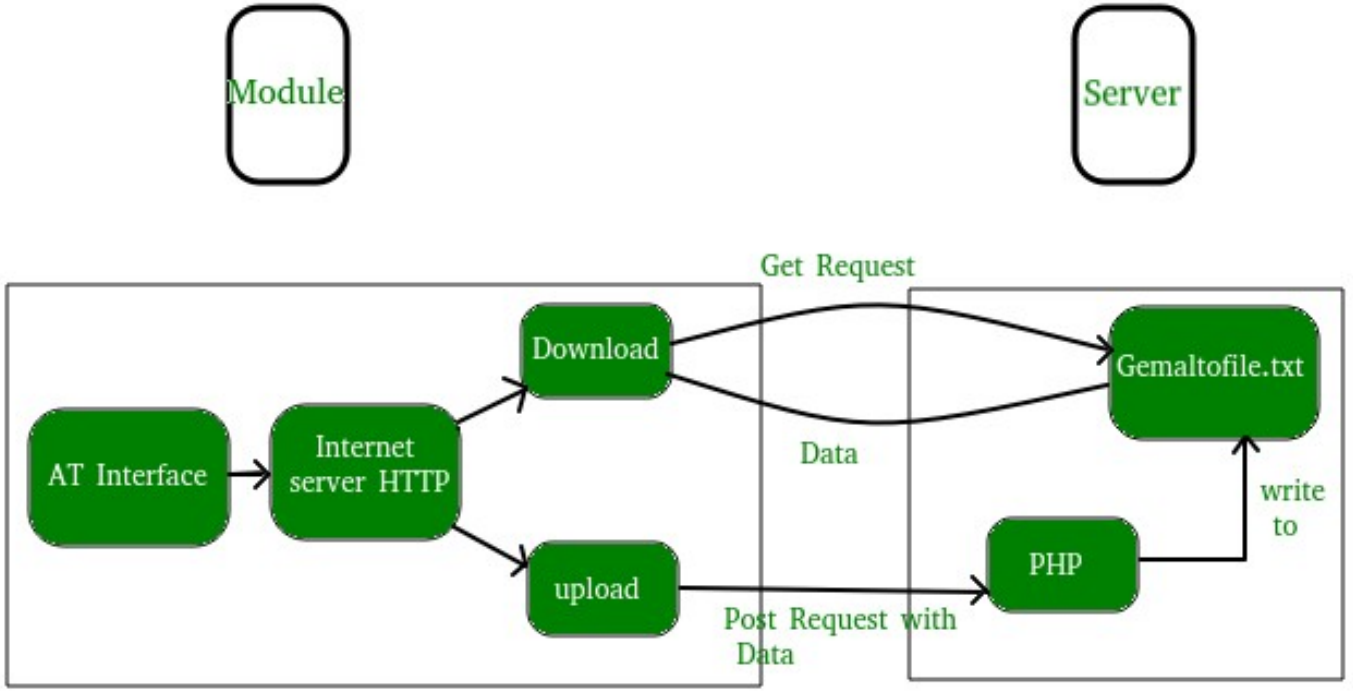
Encoding and Ecoding JSON in Python

```
import json

# A basic python dictionary
py_object = {"c": 0, "b": 0, "a": 0}

# Encoding
json_string = json.dumps(py_object)
print(json_string)
print(type(json_string))

# Decoding JSON
py_obj = json.loads(json_string)
print()
print(py_obj)
print(type(py_obj))
```

Output:

```
{"c": 0, "b": 0, "a": 0}
<class 'str'>

{'c': 0, 'b': 0, 'a': 0}
<class 'dict'>
```

GET and POST requests in Python



GET requests in Python

```
# importing the requests library
    import requests
# api-endpoint
   URL = "http://maps.googleapis.com/maps/api/geocode/json"
# location given here
   location = "iiita"
# defining a params dict for the parameters to be sent to the API
  PARAMS = {'address':location}
# sending get request and saving the response as response object
  r = requests.get(url = URL, params = PARAMS)
# extracting data in json format
  data = r.json()


# extracting latitude, longitude and formatted address
# of the first matching location
    latitude = data['results'][0]['geometry']['location']['lat']
   longitude = data['results'][0]['geometry']['location']['lng']
   formatted_address = data['results'][0]['formatted_address']


# printing the output
   print("Latitude:%s\nLongitude:%s\nFormatted Address:%s %(latitude, longitude,formatted_address))
```

POST requests in Python

```
# importing the requests library
 import requests

# defining the api-endpoint
 API_ENDPOINT = "http://pastebin.com/api/api_post.php"

# your API key here
API_KEY = "XXXXXXXXXXXXXXXXX"

# your source code here
source_code = '''
print("Hello, world!")
a = 1
b = 2
print(a + b)
'''

# data to be sent to api
data = {'api_dev_key':API_KEY,
      'api_option':'paste',
      'api_paste_code':source_code,
      'api_paste_format':'python'}

# sending post request and saving response as response object
r = requests.post(url = API_ENDPOINT, data = data)

# extracting response text
pastebin_url = r.text
print("The pastebin URL is:%s"%pastebin_url)
```