FIOT Notes

Computer Science & Engineering (Avanthi Institute of Engineering & Technology)

JNTU HYDERABAD

FUNDAMENTALS OF INTERNET OF THINGS

B.TECH/OPEN ELECTIVE/R18

SYLLABUS

UNIT – I

Introduction to Internet of Things, Characteristics of IoT, Physical design of IoT, Functional blocks of IoT, Sensing, Actuation, Basics of Networking, Communication Protocols, Sensor Networks.

UNIT – II

Machine-to-Machine Communications, Difference between IoT and M2M, Interoperability in IoT, Introduction to Arduino Programming, Integration of Sensors and Actuators with Arduino.

UNIT – III

Introduction to Python programming, Introduction to Raspberry Pi, Interfacing Raspberry Pi with basic peripherals, Implementation of IoT with Raspberry Pi.

UNIT – IV

Implementation of IoT with Raspberry Pi, Introduction to Software defined Network (SDN), SDN for IoT, Data Handling and Analytics.

UNIT – V

Cloud Computing, Sensor-Cloud, Smart Cities and Smart Homes, Connected Vehicles, Smart Grid, Industrial IoT, Case Study: Agriculture, Healthcare, Activity Monitoring.

*****

UNIT – I

INTRODUCTION TO INTERNET OF THINGS (IOT)

- "Internet of Things (IoT)" was coined by Kevin Ashton in 1999, and it has recently become more relevant to the practical world largely because of the growth of mobile devices, embedded and appearing communication, cloud computing and data analytics.
- The "Thing" in IoT can be any device with any kind of built-in-sensors with the ability to collect and transfer data over a network without manual intervention.
- The embedded technology in the object helps them to interact with internal states and the external environment, which in turn helps in decisions making process.
- However, all complete IoT systems are the same in that they represent the integration of four distinct components: sensors/devices, connectivity, data processing, and a user interface.
- IoT comprises things that have unique identities and are connected to internet.
- By 2020 there will be a total of 50 billion devices/things connected to internet.
- Internet of Things would be a $8.9 trillion market in 2020.
- IoT is not limited to just connecting things to the internet but also allow things to communicate and exchange data.
- Imagine a world where billions of objects can sense, communicate and share information, all interconnected over public or private Internet Protocol (IP) networks. These interconnected objects have data regularly collected, analyzed and used to initiate action, providing a wealth of intelligence for planning, management and decision making. This is the world of the Internet of Things.

What is meant by Internet of things?

The Internet of Things (IoT) describes the network of physical objects — “things”— that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet.

What is the Internet of things and why is it important?

Internet of Things can create information about the connected objects, analyze it, and make decisions; in other words, one can tell that the Internet of Things is smarter than the Internet. Security cameras, sensors, vehicles, buildings, and software are examples of things that can exchange data among each other.

IoT is about transferring data without or with human intervention. It is nothing about human-to-human or computer interaction as it has UIDs (Unique Identifiers). Artificial Intelligence is all about making your system behave smartly according to human behaviour, whereas IoT is all about the sensors of devices.

Definition: The Internet of things refers to a type of network to connect anything with the Internet based on stipulated protocols through information sensing equipment’s to conduct information exchange and communications in order to achieve smart recognitions, positioning, tracing, monitoring, and administration.

(Or)

A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network, often communicate data associated with users and their environments.

(Or)

The internet of things (IoT) is a computing concept that describes the idea of everyday physical objects being connected to the internet and being able to identify themselves to other devices. It has dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes and virtual network and use intelligent interfaces.

Benefits of IoT: Since IoT allows devices to be controlled remotely across the internet, thus it created opportunities to directly connect & integrate the physical world to the computer-based systems using sensors and internet. The interconnection of these multiple embedded devices will be resulting in automation in nearly all fields and also enabling advanced applications. This is resulting in improved accuracy, efficiency and economic benefit with reduced human intervention. It encompasses technologies such as smart grids, smart homes, intelligent transportation and smart cities.

CHARACTERISTICS OF IOT

There are 5 types of characteristics that are:

1) Dynamic & Self-Adapting.
2) Self-Configuring.
3) Interoperable Communication Protocols.
4) Unique Identity.
5) Integrated into Information Network.

1) Dynamic & Self Adapting: IoT devices and systems may have the capability to dynamically adapt with the changing contexts and take actions based on their operating conditions, user's context or sensed environment.

Eg: the surveillance system is adapting itself based on context and changing conditions.

2) Self Configuring: allowing a large number of devices to work together to provide certain functionality.

3) Inter Operable Communication Protocols: support a number of interoperable communication protocols and can communicate with other devices and also with infrastructure.

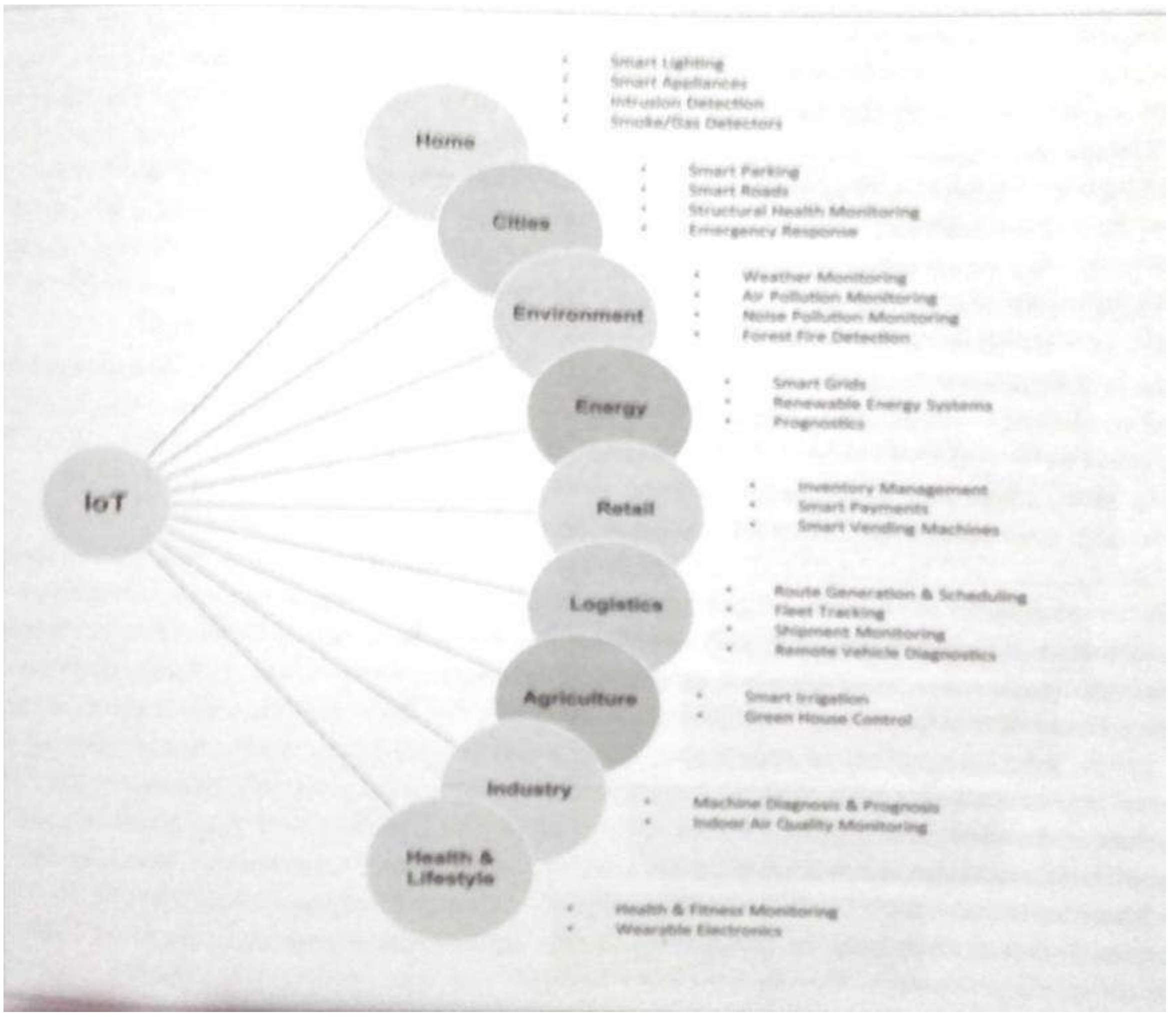
4) Unique Identity: Each IoT device has a unique identity and a unique identifier (IP address).

5) Integrated into Information Network: that allow them to communicate and exchange data with other devices and systems.

APPLICATIONS OF IOT [OR] DOMAIN SPECIFIC IOTs

The Applications on IoT networks extract and create information from lower- level data by filtering, processing, categorizing, condensing and contextualizing the data.

1) Home.
2) Cities.
3) Environment.
4) Energy.
5) Retail.
6) Logistics.
7) Agriculture.
8) Industry.
9) Health & Lifestyle.



1) Home Automation:

i. **Smart Lighting:** helps in saving energy by adapting the lighting to the ambient conditions and switching on/off or diming the light when needed.

ii. **Smart Appliances:** make the management easier and also provide status information to the users remotely.

iii. **Intrusion Detection:** use security cameras and sensors (PIR sensors and door sensors) to detect intrusion and raise alerts. Alerts can be in the form of SMS or email sent to the user.

iv. **Smoke/Gas Detectors:** Smoke detectors are installed in homes and buildings to detect smoke that is typically an early sign of fire. Alerts raised by smoke detectors can be in the form of signals to a fire alarm system. Gas detectors can detect the presence of harmful gases such as CO, LPG etc.,

2) Cities:

i. **Smart Parking:** make the search for parking space easier and convenient for drivers. Smart parking is powered by IoT systems that detect the no. of empty parking slots and send information over internet to smart application backends.

ii. **Smart Lighting:** for roads, parks and buildings can help in saving energy.

iii. **Smart Roads:** Equipped with sensors can provide information on driving condition, travel time estimating and alert in case of poor driving conditions, traffic condition and accidents.

iv. **Structural Health Monitoring:** uses a network of sensors to monitor the vibration levels in the structures such as bridges and buildings.

v. **Surveillance:** The video feeds from surveillance cameras can be aggregated in cloud based scalable storage solution.

vi. **Emergency Response:** IoT systems for fire detection, gas and water leakage detection can help in generating alerts and minimizing their effects on the critical infrastructures.

3) Environment:

i. **Weather Monitoring:** Systems collect data from a no. of sensors attached and send the data to cloud-based applications and storage back ends. The data collected in cloud can then be analyzed and visualized by cloud-based applications.

ii. **Air Pollution Monitoring:** System can monitor emission of harmful gases (CO_2, CO, NO, NO_2 etc.,) by factories and automobiles using gaseous and meteorological sensors. The collected data can be analyzed to make informed decisions on pollutions control approaches.

iii. **Noise Pollution Monitoring:** Due to growing urban development, noise levels in cities have increased and even become alarmingly high in some cities. IoT based noise pollution monitoring systems use a no. of noise monitoring systems that are deployed at different places in a city. The data on noise levels from the station is collected on servers or in the cloud. The collected data is then aggregated to generate noise maps.

iv. **Forest Fire Detection:** Forest fire can cause damage to natural resources, property and human life. Early detection of forest fire can help in minimizing damage.

v. **River Flood Detection:** River floods can cause damage to natural and human resources and human life. Early warnings of floods can be given by monitoring the water level and flow rate. IoT based river flood monitoring system uses a no. of sensor nodes that monitor the water level and flow rate sensors.

4) Energy:

i. **Smart Grids:** is a data communication network integrated with the electrical grids that collects and analyze data captured in near-real-time about power transmission, distribution and consumption. Smart grid technology provides predictive information and recommendations to utilities, their suppliers, and their customers on how best to manage power. By using IoT based sensing and measurement technologies, the health of equipment and integrity of the grid can be evaluated.

ii. **Renewable Energy Systems:** IoT based systems integrated with the transformers at the point of interconnection measure the electrical variables and how much power is fed into the grid. For wind energy systems, closed-loop controls can be used to regulate the voltage at point of interconnection which coordinate wind turbine outputs and provides power support.

iii. **Prognostics:** In systems such as power grids, real-time information is collected using specialized electrical sensors called Phasor Measurement Units (PMUs) at the substations. The information received from PMUs must be monitored in real-time for estimating the state of the system and for predicting failures.

5) Retail:

i. **Inventory Management:** IoT systems enable remote monitoring of inventory using data collected by RFID readers.

ii. **Smart Payments:** Solutions such as contact-less payments powered by technologies such as Near Field Communication (NFC) and Bluetooth.

iii. **Smart Vending Machines:** Sensors in a smart vending machines monitors its operations and send the data to cloud which can be used for predictive maintenance.

6) Logistics:

i. **Route generation & scheduling:** IoT based system backed by cloud can provide first response to the route generation queries and can be scaled up to serve a large transportation network.
ii. **Fleet Tracking:** Use GPS to track locations of vehicles in real-time.
iii. **Shipment Monitoring:** IoT based shipment monitoring systems use sensors such as temp, humidity, to monitor the conditions and send data to cloud, where it can be analyzed to detect food spoilage.
iv. **Remote Vehicle Diagnostics:** Systems use on-board IoT devices for collecting data on Vehicle operations (speed, RPM etc.,) and status of various vehicle subsystems.

7) Agriculture:

i. **Smart Irrigation:** to determine moisture amount in soil.
ii. **Green House Control:** to improve productivity.

8) Industry:

i. Machine diagnosis and prognosis.
ii. Indoor Air Quality Monitoring.

9) Health and Lifestyle:

i. Health & Fitness Monitoring.
ii. Wearable Electronics.

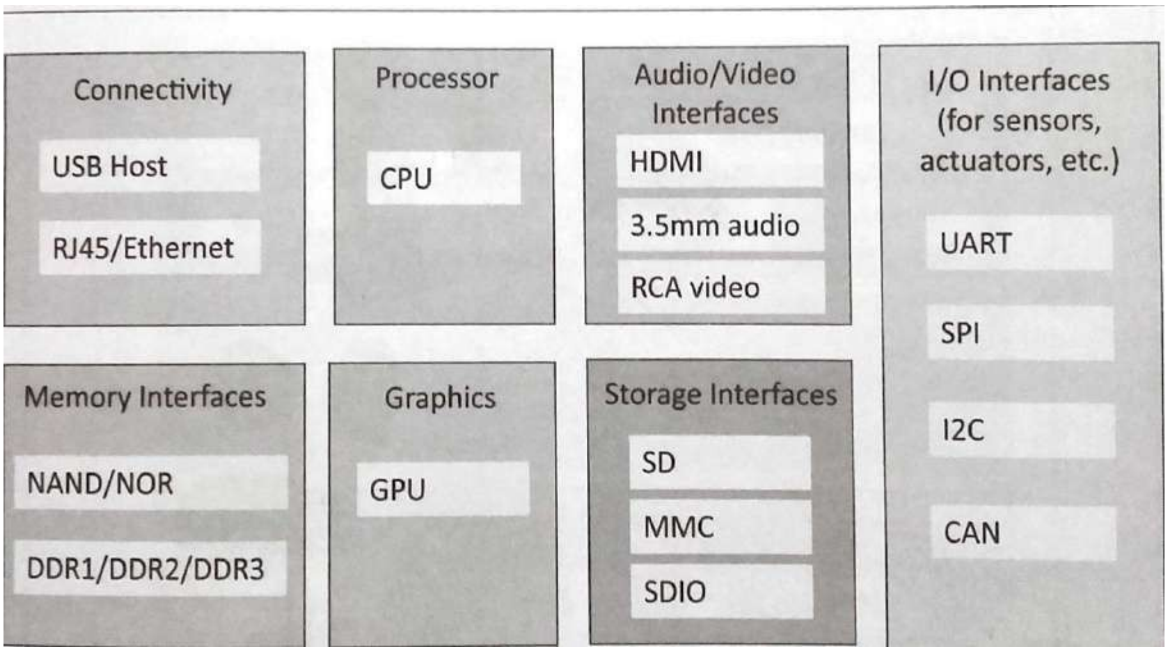
<u>PHYSICAL DESIGN OF IOT</u>

- The "Things" in IoT usually refers to IoT devices which have unique identities and can perform remote sensing, actuating and monitoring capabilities.
- IoT devices can:
  - ↳ Exchange data with other connected devices and applications (directly or indirectly), or

- ↳ Collect data from other devices and process the data locally or
- ↳ Send the data to centralized servers or cloud-based application back-ends for processing the data, or
- ↳ Perform some tasks locally and other tasks within the IoT infrastructure, based on temporal and space constraints.

1) Things in IoT/Generic block diagram of an IoT Device:

- The things in IoT refers to IoT devices which have unique identities and perform remote sensing, actuating and monitoring capabilities. IoT devices can exchange data with other connected devices applications. It collects data from other devices and process data either locally or remotely.
- An IoT device may consist of several interfaces for connections/ communication to other devices, both wired and wireless. These includes:
  1. I/O interfaces for sensors.
  2. Interfaces for Internet connectivity.
  3. Memory and storage interfaces.
  4. Audio/video interfaces.



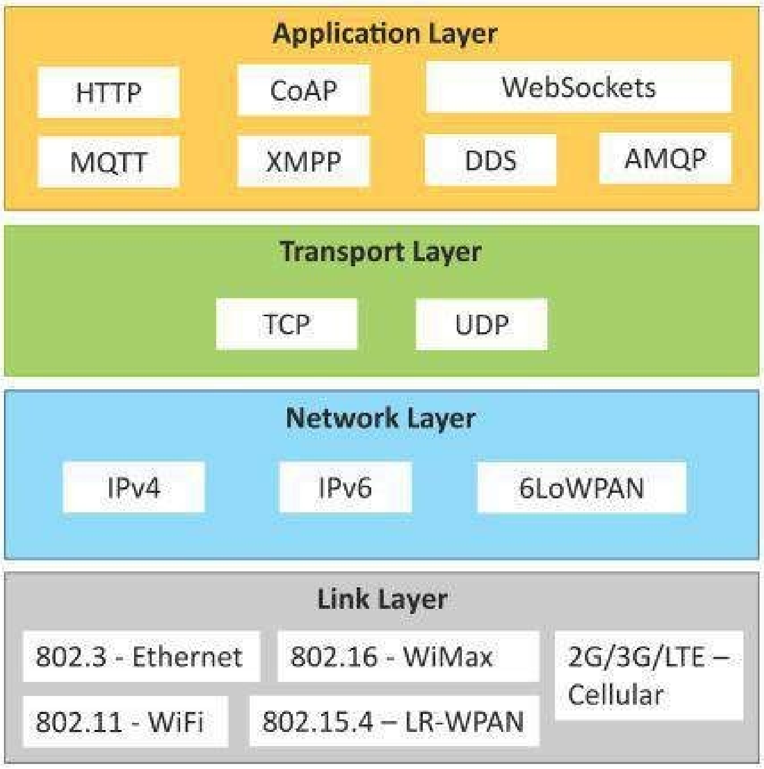
2) IoT Protocols:

i. **Link Layer:**
   - 802.3 – Ethernet
   - 802.11 – WiFi
   - 802.16 – WiMax
   - 802.15.4 – LR-WPAN
   - 2G/3G/4G

ii. **Network/Internet Layer:**
   - IPv4
   - IPv6
   - 6LoWPAN

iii. **Transport Layer:**
   - TCP
   - UDP

iv. **Application Layer:**
   - HTTP
   - CoAP
   - WebSocket
   - MQTT
   - XMPP
   - DDS
   - AMQP



i) Link Layer: Protocols determine how data is physically sent over the network's physical layer or medium. Local network connect to which host is attached. Hosts on the same link exchange data packets over the link layer using link layer protocols. Link layer determines how packets are coded and signaled by the h/w device over the medium to which the host is attached.

Protocols:

- **802.3-Ethernet:** IEEE802.3 is collection of wired Ethernet standards for the link layer. Eg: 802.3 uses co-axial cable; 802.3i uses copper twisted pair connection; 802.3j uses fiber optic connection; 802.3ae uses Ethernet over fiber.
- **802.11-WiFi:** IEEE802.11 is a collection of wireless LAN (WLAN) communication standards including extensive description of link layer. Eg: 802.11a operates in 5GHz band, 802.11b and 802.11g operates in 2.4GHz band, 802.11n operates in 2.4/5GHz band, 802.11ac operates in 5GHz band, 802.11ad operates in 60Ghzband.
- **802.16 - WiMax:** IEEE802.16 is a collection of wireless broadband standards including exclusive description of link layer. WiMax provide data rates from 1.5 Mb/s to 1Gb/s.

- **802.15.4-LR-WPAN:** IEEE802.15.4 is a collection of standards for low rate wireless personal area network (LR-WPAN). Basis for high level communication protocols such as ZigBee. Provides data rate from 40kb/s to250kb/s.
- **2G/3G/4G-Mobile Communication:** Data rates from 9.6kb/s(2G) to up to100Mb/s(4G).

ii) Network/Internet Layer: Responsible for sending IP datagrams from source n/w to destination n/w. Performs the host addressing and packet routing. Datagrams contains source and destination address.

Protocols:

- **IPv4:** Internet Protocol version4 is used to identify the devices on a n/w using a hierarchical addressing scheme. 32 bit address. Allows total of 2**32addresses.
- **IPv6:** Internet Protocol version6 uses 128 bit address scheme and allows 2**128 addresses.
- **6LOWPAN:**(IPv6overLowpowerWirelessPersonalAreaNetwork) operates in 2.4 GHz frequency range and data transfer 250 kb/s.

iii) Transport Layer: Provides end-to-end message transfer capability independent of the underlying n/w. Set up on connection with ACK as in TCP and without ACK as in UDP. Provides functions such as error control, segmentation, flow control and congestion control.

Protocols:

- **TCP:** Transmission Control Protocol used by web browsers (along with HTTP and HTTPS), email (along with SMTP, FTP). Connection oriented and stateless protocol. IP Protocol deals with sending packets, TCP ensures reliable transmission of protocols in order. Avoids n/w congestion and congestion collapse.
- **UDP:** User Datagram Protocol is connectionless protocol. Useful in time sensitive applications, very small data units to exchange. Transaction oriented and stateless protocol. Does not provide guaranteed delivery.

iv) Application Layer: Defines how the applications interface with lower layer protocols to send data over the n/w. Enables process-to-process communication using ports.

Protocols:

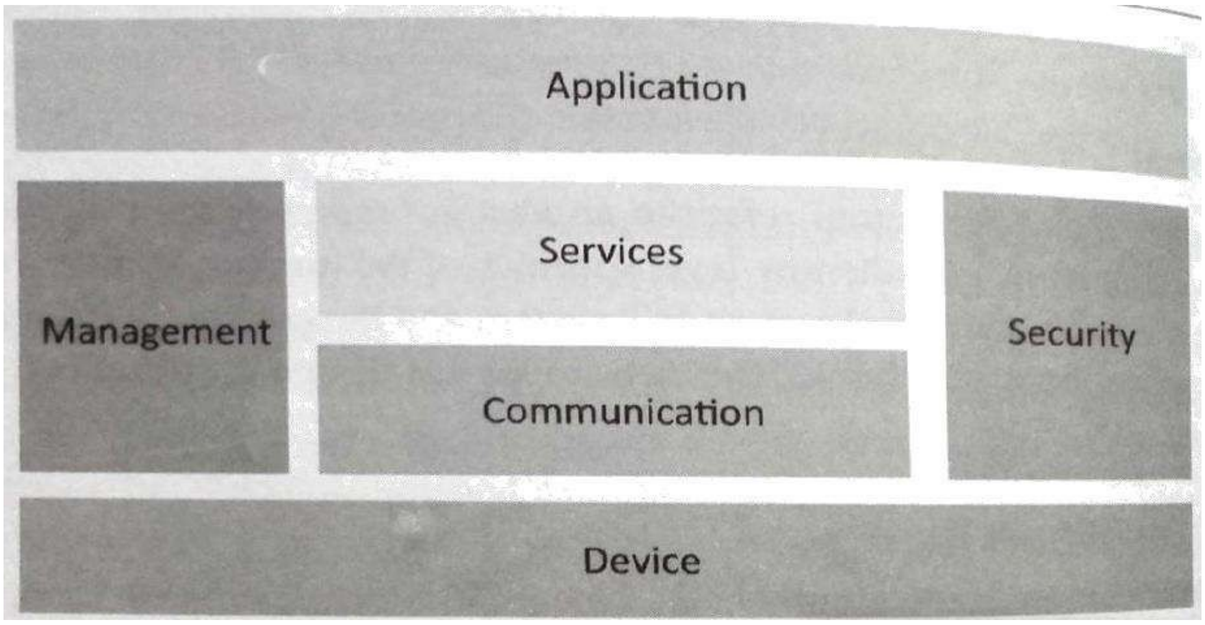
- **HTTP:** Hyper Text Transfer Protocol that forms foundation of WWW. Follow request- response model Stateless protocol.
- **CoAP:** Constrained Application Protocol for machine-to-machine (M2M) applications with constrained devices, constrained environment and constrained n/w. Uses client- server architecture.
- **WebSocket:** allows full duplex communication over a single socket connection.
- **MQTT:** Message Queue Telemetry Transport is light weight messaging protocol based on publish-subscribe model. Uses client server architecture. Well suited for constrained environment.
- **XMPP:** Extensible Message and Presence Protocol for real time communication and streaming XML data between network entities. Support client-server and server-server communication.
- **DDS:** Data Distribution Service is data centric middleware standards for device-to-device or machine-to-machine communication. Uses publish-subscribe model.
- **AMQP:** Advanced Message Queuing Protocol is open application layer protocol for business messaging. Supports both point-to-point and publish-subscribe model.

FUNCTIONAL BLOCKS OF IOT [OR] IOT FUNCTIONAL BLOCKS

Provide the system the capabilities for identification, sensing, actuation, communication and management.

1) Device.
2) Communication.
3) Services.
4) Management.
5) Security.

6) Applications.



1) **Device:** An IoT system comprises of devices that provide sensing, actuation, monitoring and control functions.
2) **Communication:** handles the communication for IoT system.
3) **Services:** for device monitoring, device control services, data publishing services and services for device discovery.
4) **Management:** Provides various functions to govern the IoT system.
5) **Security:** Secures IoT system and priority functions such as authentication, authorization, message and context integrity and data security.
6) **Application:** IoT application provide an interface that the users can use to control and monitor various aspects of IoT system.

SENSING

- The main purpose of sensors is to collect data from the surrounding environment. Sensors, or 'things' of the IoT system, form the front end.
- These are connected directly or indirectly to IoT networks after signal conversion and processing.
- **Types of IoT sensors:**
  1) Temperature sensors.
  2) Humidity sensors.

3) Motion sensors.
4) Gas sensors.
5) Smoke sensors.
6) Pressure sensors.
7) Image sensors.
8) IR sensors.

ACTUATION

Another type of transducer that you will encounter in many IoT systems is an actuator. In simple terms, an actuator operates in the reverse direction of a sensor. It takes an electrical input and turns it into physical action.

SENSING VS ACTUATION

Sensing converts physical characteristics into electrical signals. Actuation converts electrical signals into physical characteristics. Sensor generated electrical signals. Actuator generates heat or motion. It is placed at input port of the system.

BASICS OF NETWORKING

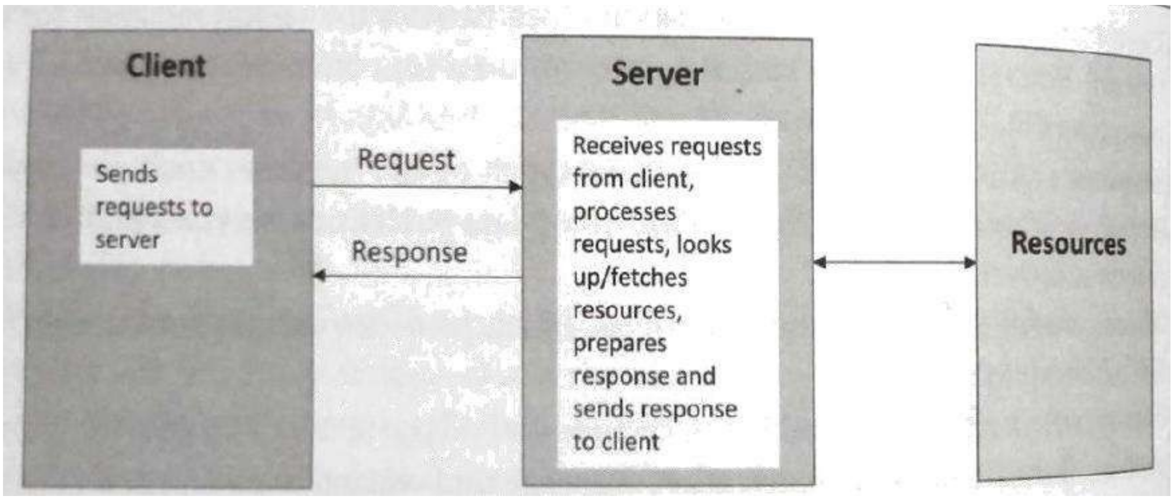
Switches, routers, and wireless access points are the essential networking basics. Through them, devices connected to your network can communicate with one another and with other networks, like the Internet. Switches, routers, and wireless access points perform very different functions in a network.

COMMUNICATION PROTOCOLS

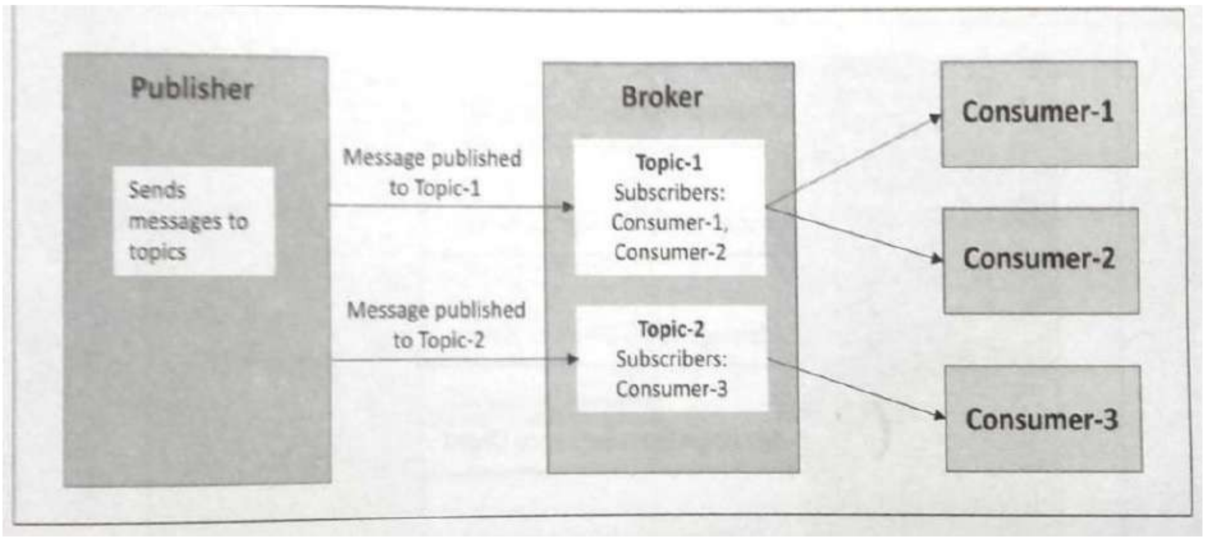
IoT Communication Models:

1) Request-Response Model.
2) Publish-Subscribe Model.
3) Push-Pull Model.
4) Exclusive Pair Model.

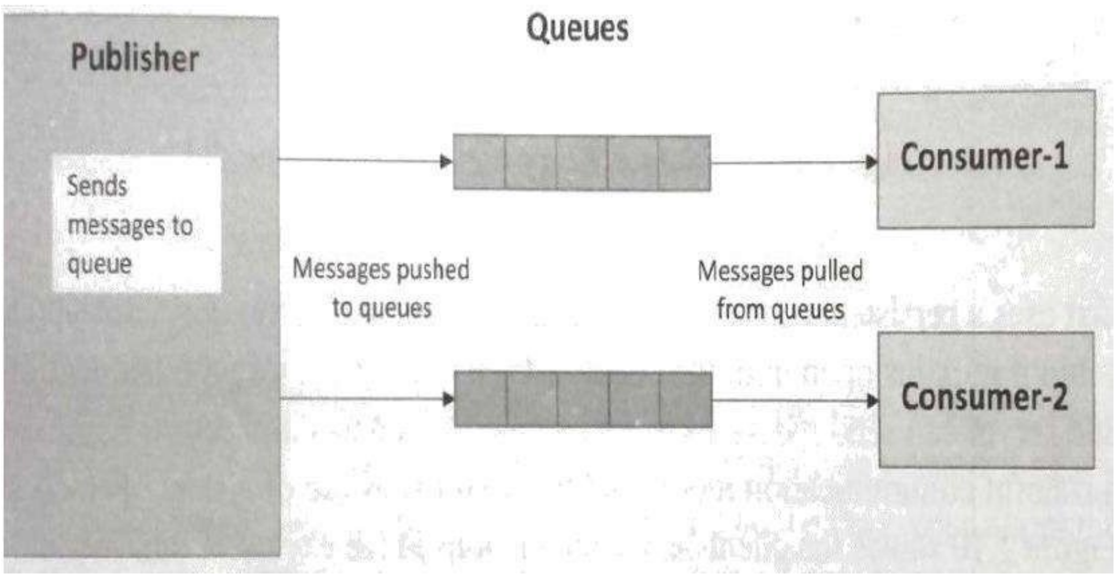
1) Request-Response Model: In which the client sends request to the server and the server replies to requests. Is a stateless communication model and each request-response pair is independent of others.


Client
Sends requests to server
Request
Response
Server
Receives requests from client, processes requests, looks up/fetches resources, prepares response and sends response to client
Resources

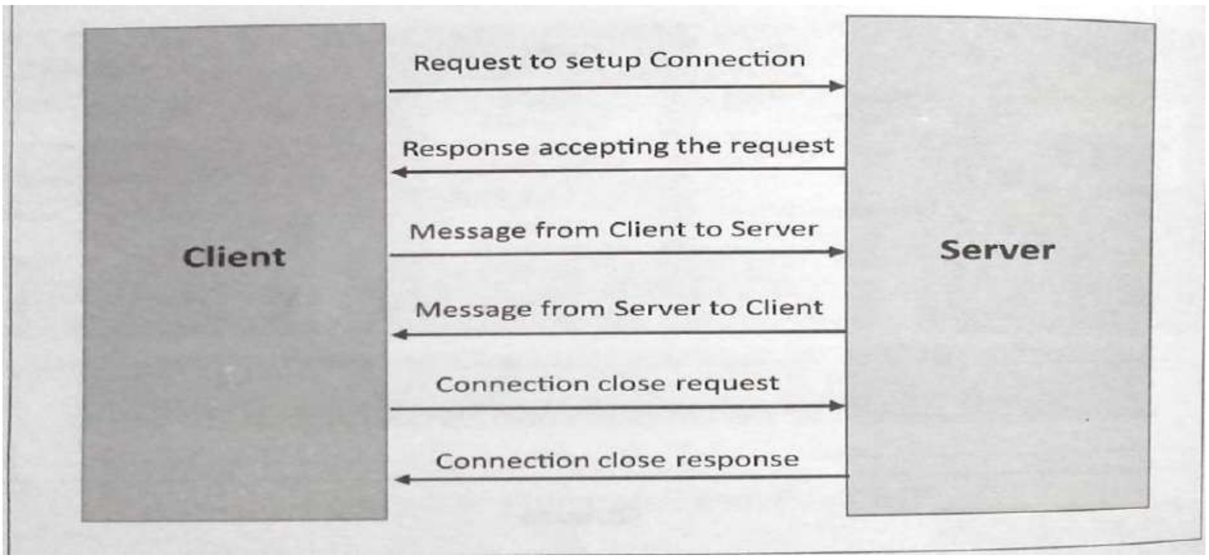
2) Publish-Subscribe Model: Involves publishers, brokers and consumers. Publishers are source of data. Publishers send data to the topics which are managed by the broker. Publishers are not aware of the consumers. Consumers subscribe to the topics which are managed by the broker. When the broker receives data for a topic from the publisher, it sends the data to all the subscribed consumers.



3) Push-Pull Model: in which data producers push data to queues and consumers pull data from the queues. Producers do not need to aware of the consumers. Queues help in decoupling the message between the producers and consumers.



4 Exclusive Pair: is bi-directional, fully duplex communication model that uses a persistent connection between the client and server. Once connection is set up it remains open until the client send a request to close the connection. Is a stateful communication model and server is aware of all the open connections.

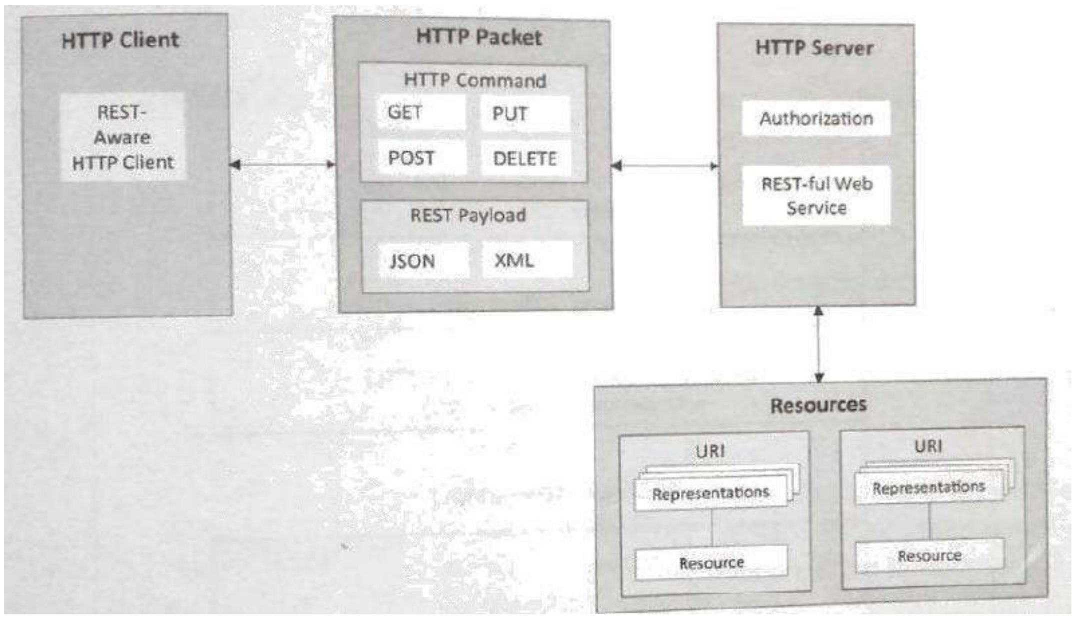

Client
Server
Request to setup Connection
Response accepting the request
Message from Client to Server
Message from Server to Client
Connection close request
Connection close response

IoT Communication APIs:

1) REST based communication APIs (Request-Response Based Model).
2) WebSocket based Communication APIs (Exclusive Pair Based Model).

1) REST based communication APIs: Representational State Transfer (REST) is a set of architectural principles by which we can design web services and web APIs that focus on a system 's resources and have resource states are addressed and transferred.

The REST architectural constraints: Fig. shows communication between client server with REST APIs.



Client-Server: The principle behind client-server constraint is the separation of concerns. Separation allows client and server to be independently developed and updated.

Stateless: Each request from client to server must contain all the info. Necessary to understand the request, and cannot take advantage of any stored context on the server.

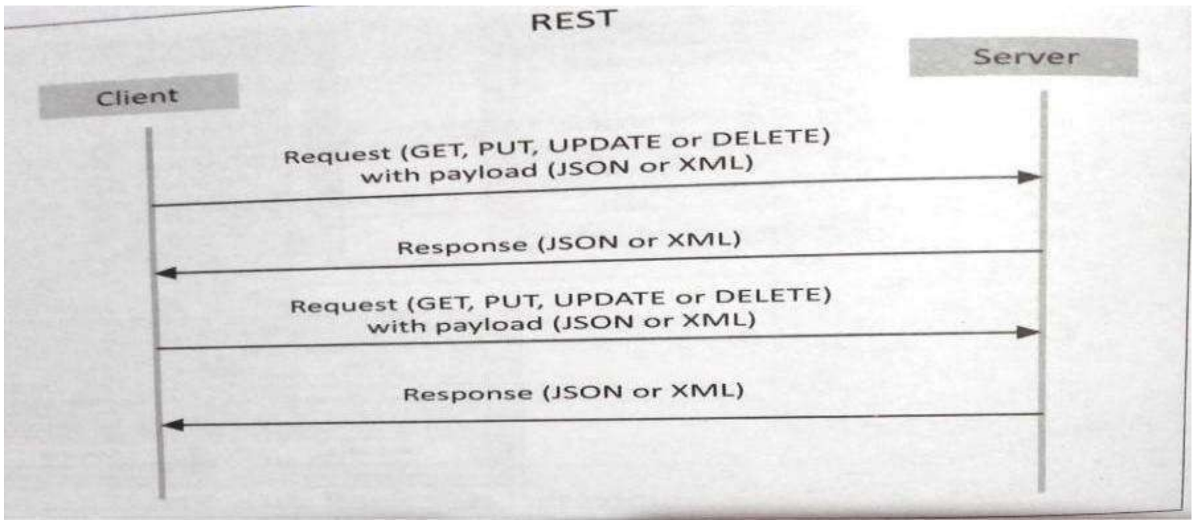
Cache-able: Cache constraint requires that the data within a response to a request be implicitly or explicitly labeled as cache-able or non-cacheable. If a response is cache-able, then a client cache is given the right to reuse that response data for later, equivalent requests.

Layered System: constraints the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting.

User Interface: constraint requires that the method of communication between a client and a server must be uniform.

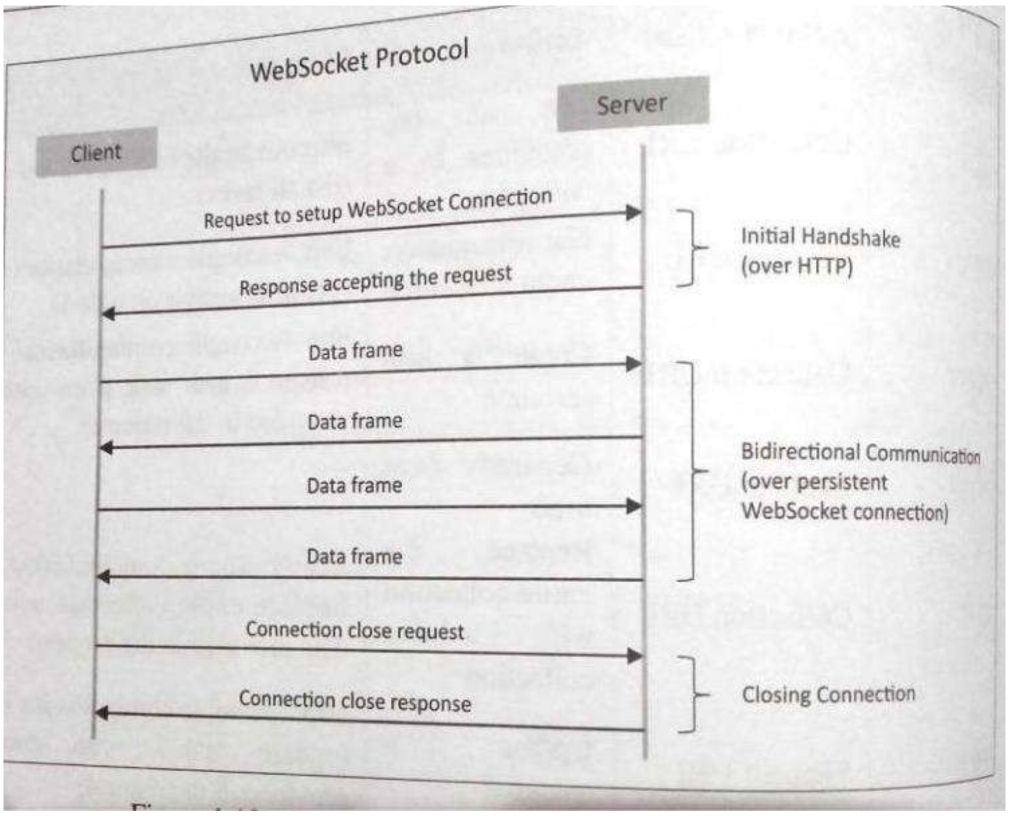
Code on Demand: Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

Request-Response model used by REST:


REST
Server
Client
Request (GET, PUT, UPDATE or DELETE) with payload (JSON or XML)
Response (JSON or XML)
Request (GET, PUT, UPDATE or DELETE) with payload (JSON or XML)
Response (JSON or XML)

RESTful webservice is a collection of resources which are represented by URIs. RESTful web API has a base URI (e.g: http://example.com/api/tasks/). The clients and requests to these URIs using the methods defined by the HTTP protocol (e.g: GET, PUT, POST or DELETE). A RESTful web service can support various internet media types.

2) WebSocket Based Communication APIs: WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.

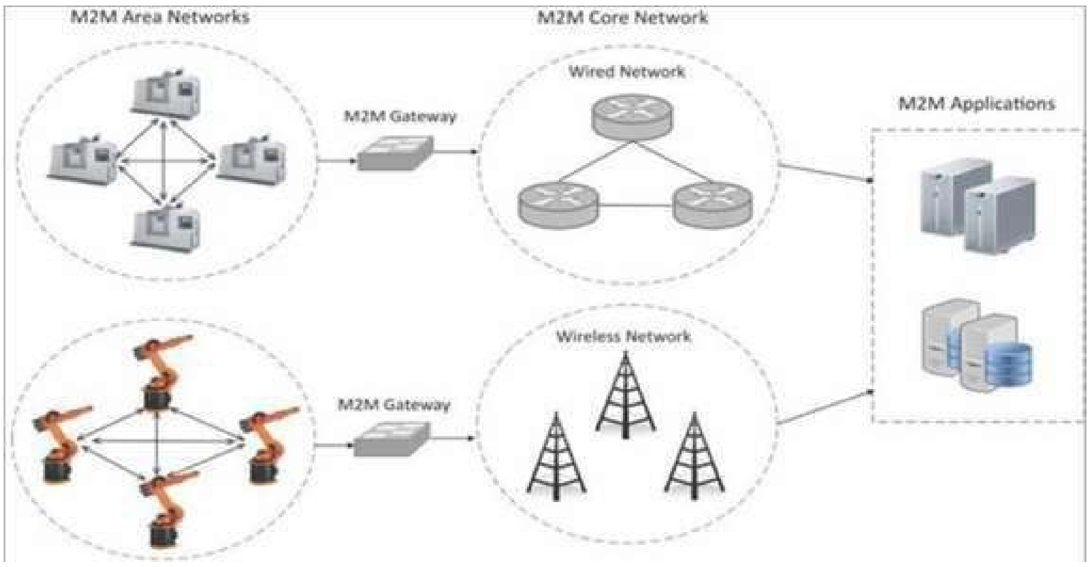

WebSocket Protocol
Client
Server
Request to setup WebSocket Connection
Response accepting the request
Initial Handshake (over HTTP)
Data frame
Data frame
Data frame
Data frame
Bidirectional Communication (over persistent WebSocket connection)
Connection close request
Connection close response
Closing Connection

*****

UNIT – II

MACHINE-TO-MACHINE (M2M) COMMUNICATIONS

- Machine-to-Machine (M2M) refers to networking of machines (or devices) for the purpose of remote monitoring and control and data exchange.
- Term which is often synonymous with IoT is Machine-to-Machine (M2M).
- IoT and M2M are often used interchangeably.
- Fig. Shows the end-to-end architecture of M2M systems comprises of M2M area networks, communication networks and application foaming.



- An M2M area network comprises of machines (or M2M nodes) which have embedded network modules for sensing, actuation and communicating various communication protocols can be used for M2M LAN such as ZigBee, Bluetooth, M-bus, Wireless M-Bus etc., These protocols provide connectivity between M2M nodes within an M2M area network.
- The communication network provides connectivity to remote M2M area networks. The communication network provides connectivity to remote M2M area network. The communication network can use either wired or wireless network (IP based). While the M2M are networks use either

proprietary or non-IP based communication protocols, the communication network uses IP-based network. Since non-IP based protocols are used within M2M area network, the M2M nodes within one network cannot communicate with nodes in an external network.

- To enable the communication between remote M2M are network, M2M gateways are used.

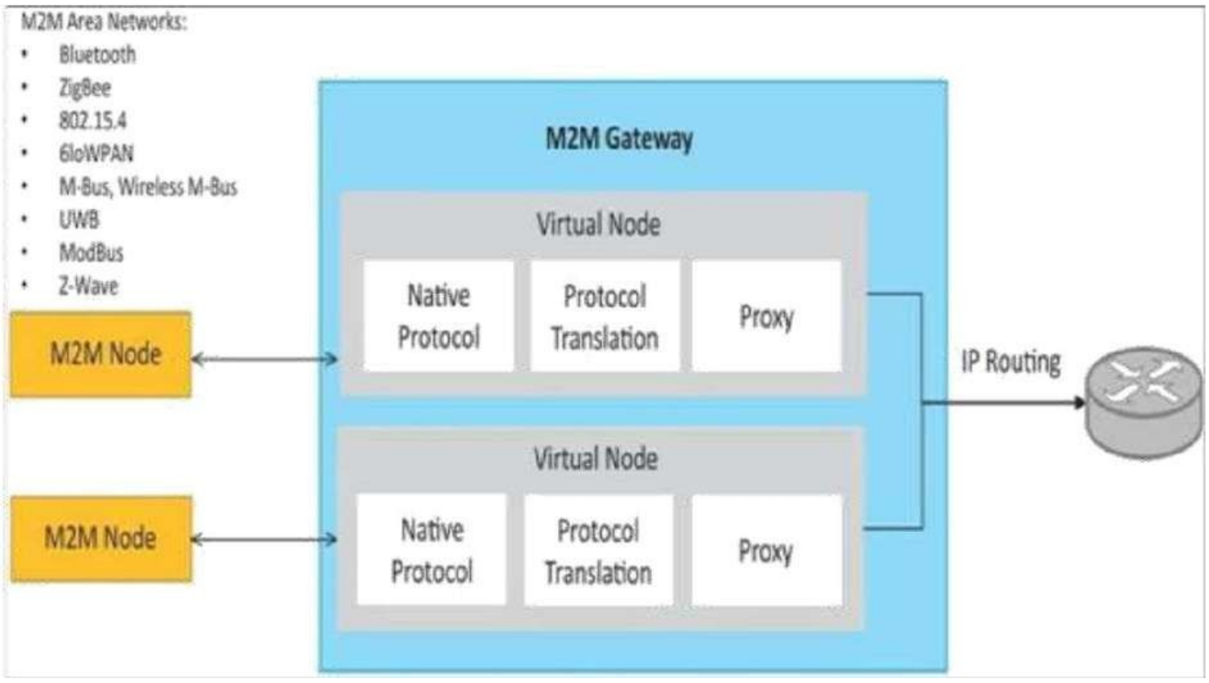


Fig. Shows a block diagram of an M2M gateway. The communication between M2M nodes and the M2M gateway is based on the communication protocols which are naive to the M2M are network. M2M gateway performs protocol translations to enable Ip-connectivity for M2M are networks. M2M gateway acts as a proxy performing translations from/to native protocols to/from Internet Protocol (IP). With an M2M gateway, each mode in an M2M area network appears as a virtualized node for external M2M area networks.

DIFFERENCE BETWEEN IOT AND M2M

1) **Communication Protocols:** Commonly uses M2M protocols include ZigBee, Bluetooth, ModBus, M-Bus, Wireless MBustec., In IoT uses HTTP, CoAP, WebSocket, MQTT, XMPP, DDS, AMQ Petc.,
2) **Machines in M2M Vs Things in IoT:** Machines in M2M will be homogenous whereas Things in IoT will be heterogeneous.

3) **Hardware Vs Software Emphasis:** the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software.
4) **Data Collection & Analysis:** M2M data is collected in point solutions and often in on-premises storage infrastructure. The data in IoT is collected in the cloud (can be public, private or hybrid cloud).
5) **Applications:** M2M data is collected in point solutions and can be accessed by on-premises applications such as diagnosis applications, service management applications, and on- premises enterprise applications. IoT data is collected in the cloud and can be accessed by cloud applications such as analytics applications, enterprise applications, remote diagnosis and management applications, etc.

INTEROPERABILITY IN IOT

What does interoperability mean?

Defining Interoperability: Interoperability is the ability of different information technology systems and software applications to communicate, to exchange data accurately, effectively, and consistently, and to use the information that has been exchanged.

What is interoperability and why is it especially important in the Internet?

Foundational interoperability enables one information system to exchange data with another information system. The system receiving this information does not need to have to interpret the data. It will be instantly available for use.

Dimensions for Interoperability: The main objective of this report is not to produce a new definition on interoperability but to explore the different roles and functionality that interoperability plays in IoT. There are many definitions of interoperability; we try to provide a common definition that can be extracted from many of those definitions (bringing from the 3rd Generation Partnership Project, 3GPP). Interoperability is:

"the ability of two or more systems or components to exchange data and use information"

This definition provides many challenges on how to:

- Get the information.
- Exchange data, and
- Use the information in understanding it and being able to process it.

A simple representation of interoperability can be seen as follow:

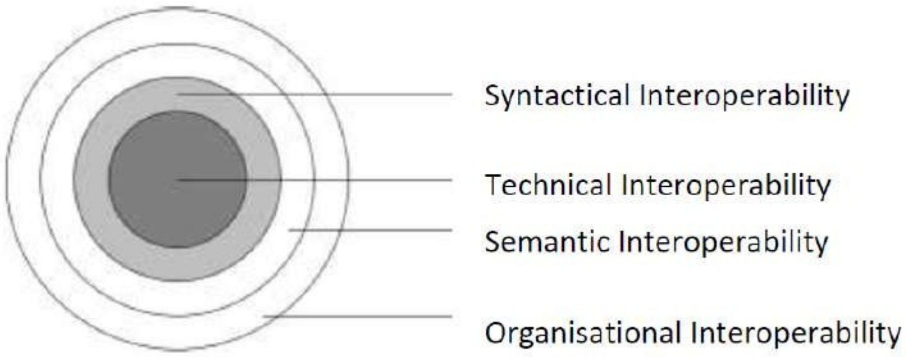

Syntactical Interoperability
Technical Interoperability
Semantic Interoperability
Organisational Interoperability

Figure 1 The Dimensions of Interoperability

Technical Interoperability is usually associated with hardware/software components, systems and platforms that enable machine-to-machine communication to take place. This kind of interoperability is often centred on (communication) protocols and the infrastructure needed for those protocols to operate.

Syntactical Interoperability is usually associated with data formats. Certainly, the messages transferred by communication protocols need to have a well-defined syntax and encoding, even if it is only in the form of bittables. However, many protocols carry data or content, and this can be represented using high-level syntaxes such as HTML or XML

Semantic Interoperability is usually associated with the meaning of content and concerns the human rather than machine interpretation of the content. Thus, interoperability on this level means that there is a common understanding between people of the meaning of the content (information) being exchanged.

Organizational Interoperability, as the name implies, is the ability of organizations to effectively communicate and transfer (meaningful) data (information) even though they may be using a variety of different information systems over widely different infrastructures, possibly across different geographic regions and cultures. Organizational interoperability depends on successful technical, syntactical and semantic interoperability.

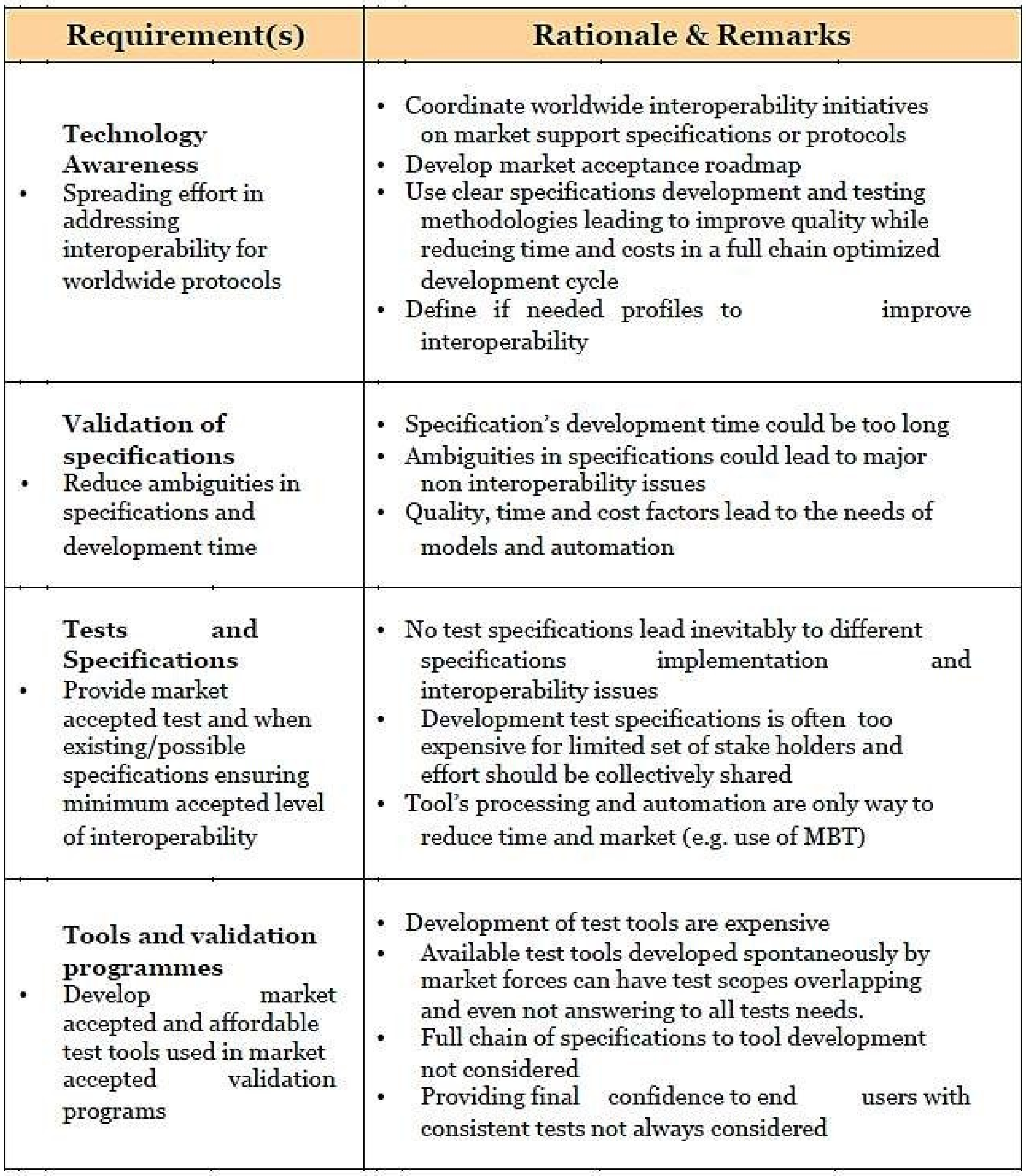
Following the definitions and the trends on ICT sector about sensors and sensor data we can add two other dimensions: Static and dynamic interoperability.

Static interoperability using approach of the well-known OSI overall test methodology ISO 9646 [49], where there is definition of static conformance review. Conformance testing consists of checking whether an Implementation Under Test (IUT) satisfies all static and dynamic conformance requirements. For the static conformance requirements this means a reviewing process of the options (PICS) delivered with the IUT. This is referred to as the static conformance review. This aspect could appear easy but that represent serious challenge in the IoT field due the broad range of applications.

Dynamic interoperability: Two products cannot interoperate if they do not implement the same set of options ("services"). Therefore, when specifications are including a broad range of options, this aspect could lead to serious interoperability problem. Solutions to overcome these aspects consist of definition clearly in a clear document the full list options with all conditions (e.g., defined as PICS in [49]) as well as to define set of profiles. In the latter case, defining profile would help to truly check interoperability between two products in the same family or from different family if the feature checked belongs to the two groups. We could consider this aspect as:

Interoperability: Challenges and Requirements: The overall challenges in interoperability is first to stabilize the foundation of the real world data/services, ensuring technical interoperability from technologies to deliver mass of information and then complementary challenges are for the information to be understood and processed. Before entering into details present a summary of the challenges for technical and semantic interoperability in Table 1.

Table 1: IoT Technical Interoperability Challenges/Requirements



Requirement(s)	Rationale & Remarks
Technology Awareness • Spreading effort in addressing interoperability for worldwide protocols	• Coordinate worldwide interoperability initiatives on market support specifications or protocols • Develop market acceptance roadmap • Use clear specifications development and testing methodologies leading to improve quality while reducing time and costs in a full chain optimized development cycle • Define if needed profiles to improve interoperability
Validation of specifications • Reduce ambiguities in specifications and development time	• Specification's development time could be too long • Ambiguities in specifications could lead to major non interoperability issues • Quality, time and cost factors lead to the needs of models and automation
Tests and Specifications • Provide market accepted test and when existing/possible specifications ensuring minimum accepted level of interoperability	• No test specifications lead inevitably to different specifications implementation and interoperability issues • Development test specifications is often too expensive for limited set of stake holders and effort should be collectively shared • Tool's processing and automation are only way to reduce time and market (e.g. use of MBT)
Tools and validation programmes • Develop market accepted and affordable test tools used in market accepted validation programs	• Development of test tools are expensive • Available test tools developed spontaneously by market forces can have test scopes overlapping and even not answering to all tests needs. • Full chain of specifications to tool development not considered • Providing final confidence to end users with consistent tests not always considered

INTRODUCTION TO ARDUINO PROGRAMMING

Arduino Programs:

All Arduino programs must follow the following main structure:

```
// Initialization, define variables, etc.

void setup ()
{
   // Initialization
   ...
}

void loop ()
{
   //Main Program
   ...
}
```

Arduino Program – Example:

```
void setup()
{
   pinMode(11, OUTPUT);       //Set the Pin as an Output
}

void loop()
{
   digitalWrite(11, HIGH);   // Turn on the LED
   delay(1000);              // Wait for one second
   digitalWrite(11, LOW);    // Turn off the LED
   delay(1000);              // Wait for one second
}
```

Arduino Program – Using Comments:

```
void setup()
{
   pinMode(11, OUTPUT);       //Set the Pin as an Output
}
void loop()
{
   digitalWrite(11, HIGH);   // Turn on the LED
   /*
   ... This will not be executed by the program because
   it is a comment...
   */
}
```

Creating and Using Functions:

```
int z;
void setup()
{
   }
void loop()
{
  z = calculate(2,3);
}
float calculate(int x, int y)
{
   return (x + y);
}
```

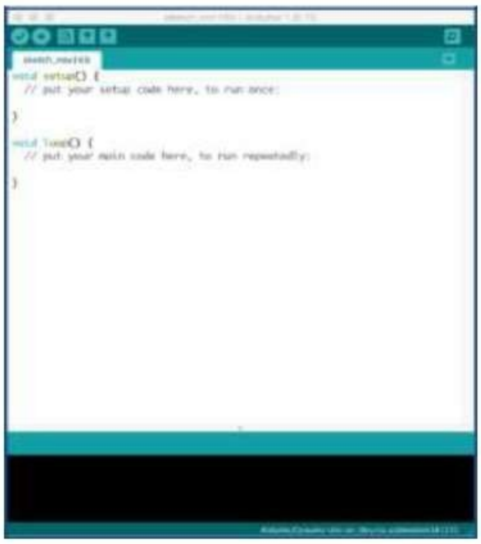
Using the Function

Creating the Function

TRY IT OUT!

Here are some Arduino Examples you should try.

Make sure your Arduino is connected to the PC and start the Code Editor



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

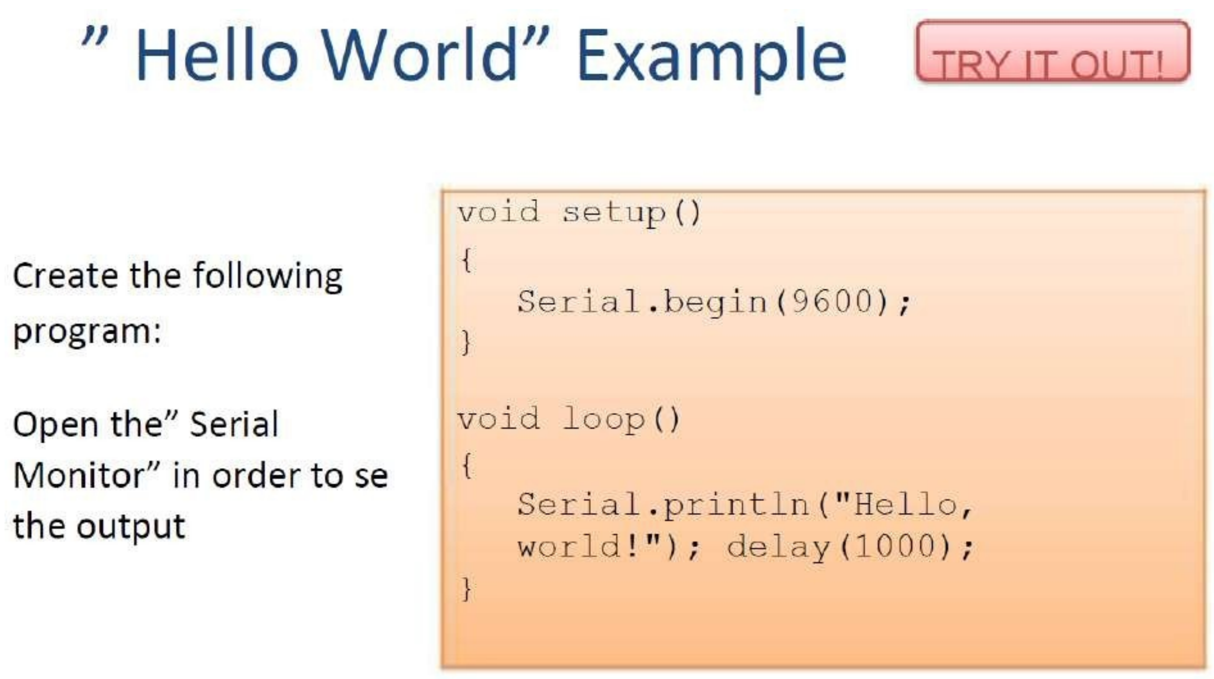
" Hello World" Example

TRY IT OUT!

Create the following program:

Open the" Serial Monitor" in order to see the output

```
void setup()
{
   Serial.begin(9600);

   Serial.println("Hello, world!");
}

void loop()
{

}
```



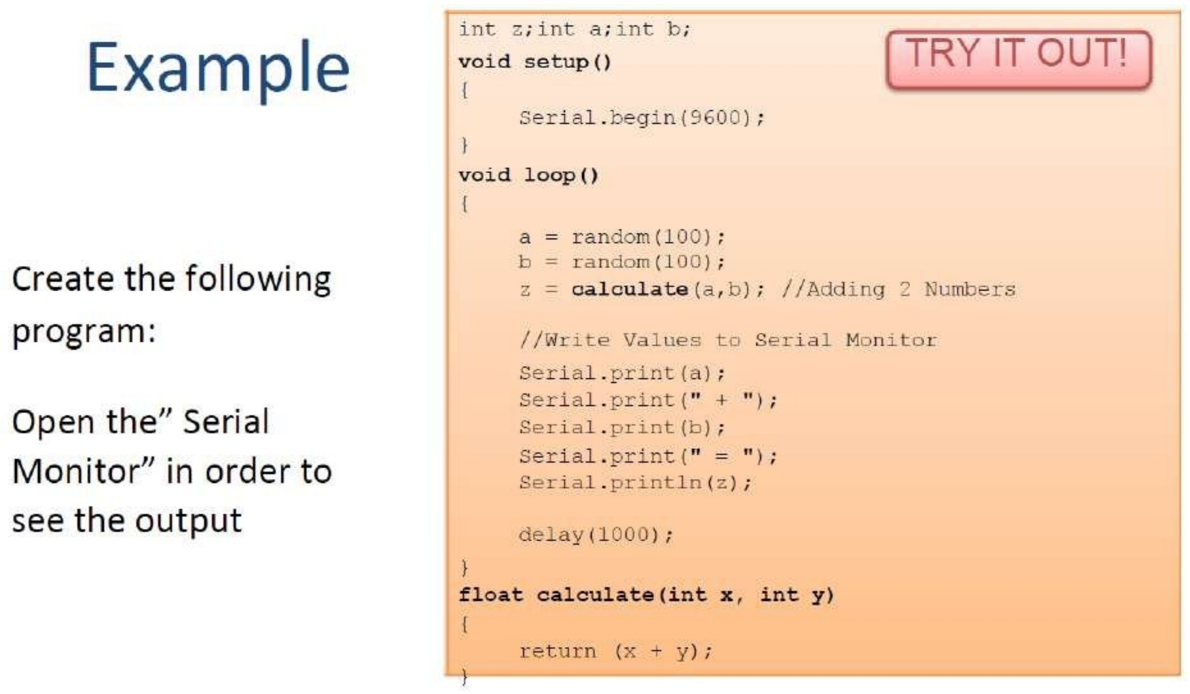
” Hello World” Example

TRY IT OUT!

Create the following program:

Open the” Serial Monitor” in order to se the output

```
void setup()
{
   Serial.begin(9600);
}

void loop()
{
   Serial.println("Hello,
   world!"); delay(1000);
}
```



Example

TRY IT OUT!

Create the following program:

Open the” Serial Monitor” in order to see the output

```
int z;int a;int b;
void setup()
{
     Serial.begin(9600);
}
void loop()
{
     a = random(100);
     b = random(100);
     z = calculate(a,b); //Adding 2 Numbers

     //Write Values to Serial Monitor
     Serial.print(a);
     Serial.print(" + ");
     Serial.print(b);
     Serial.print(" = ");
     Serial.println(z);

     delay(1000);
}
float calculate(int x, int y)
{
     return (x + y);
}
```

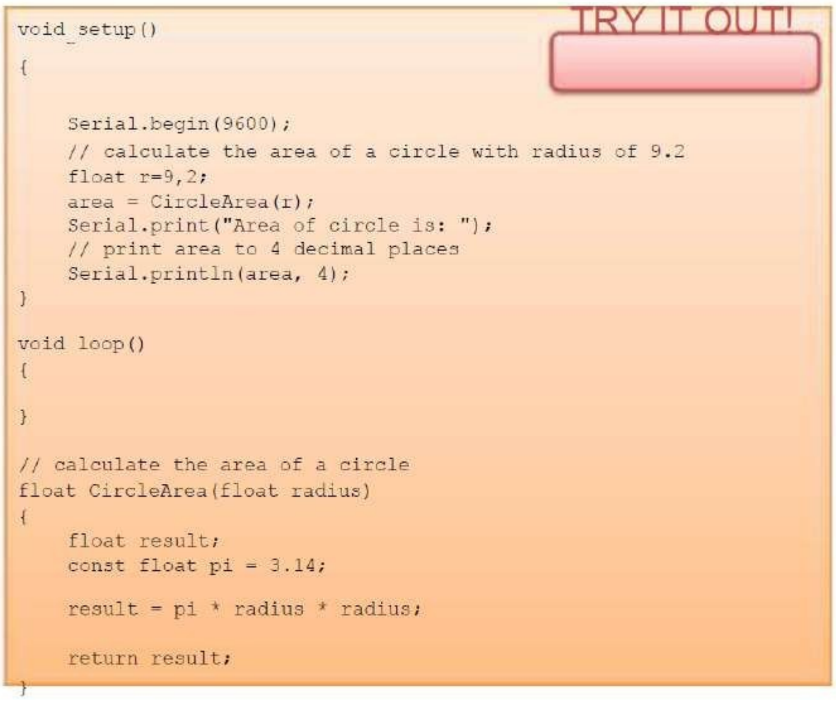
Creating Functions

TRY IT OUT!

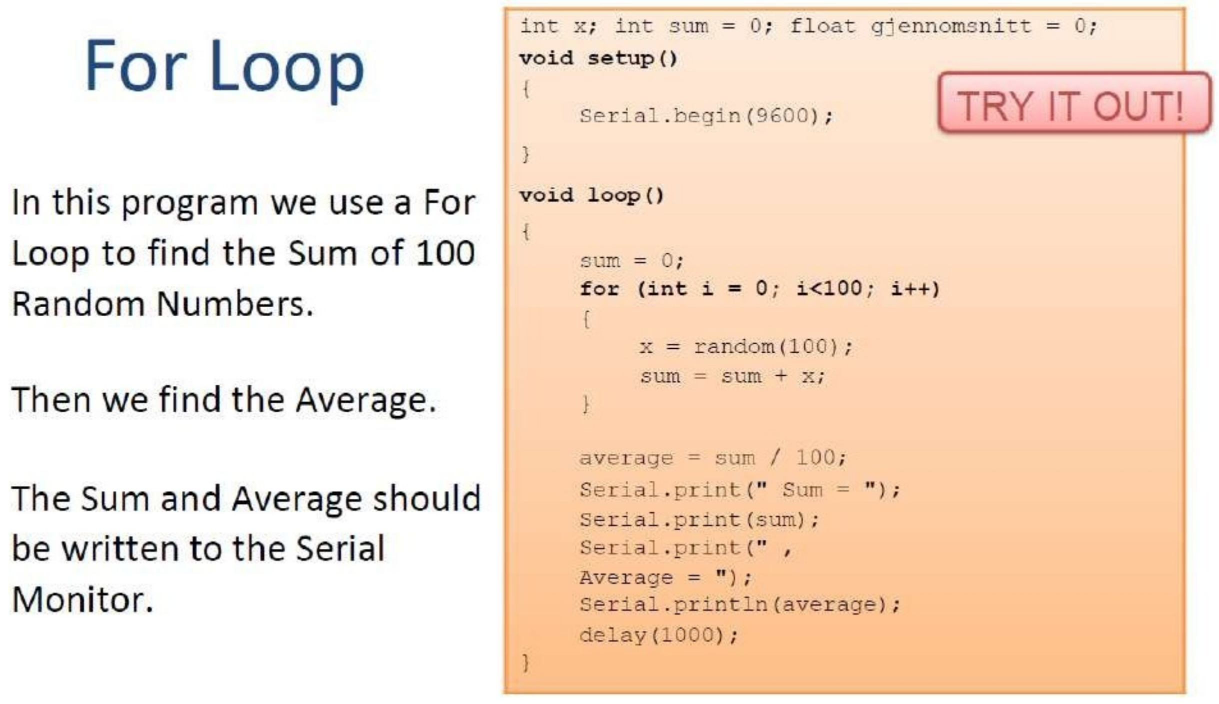
Create a function that calculates the area of a circle with a given radius.

Write the result to the Serial Monitor.

Solution



```
void setup()
{

    Serial.begin(9600);
    // calculate the area of a circle with radius of 9.2
    float r=9,2;
    area = CircleArea(r);
    Serial.print("Area of circle is: ");
    // print area to 4 decimal places
    Serial.println(area, 4);
}

void loop()
{

}

// calculate the area of a circle
float CircleArea(float radius)
{
    float result;
    const float pi = 3.14;

    result = pi * radius * radius;

    return result;
}
```



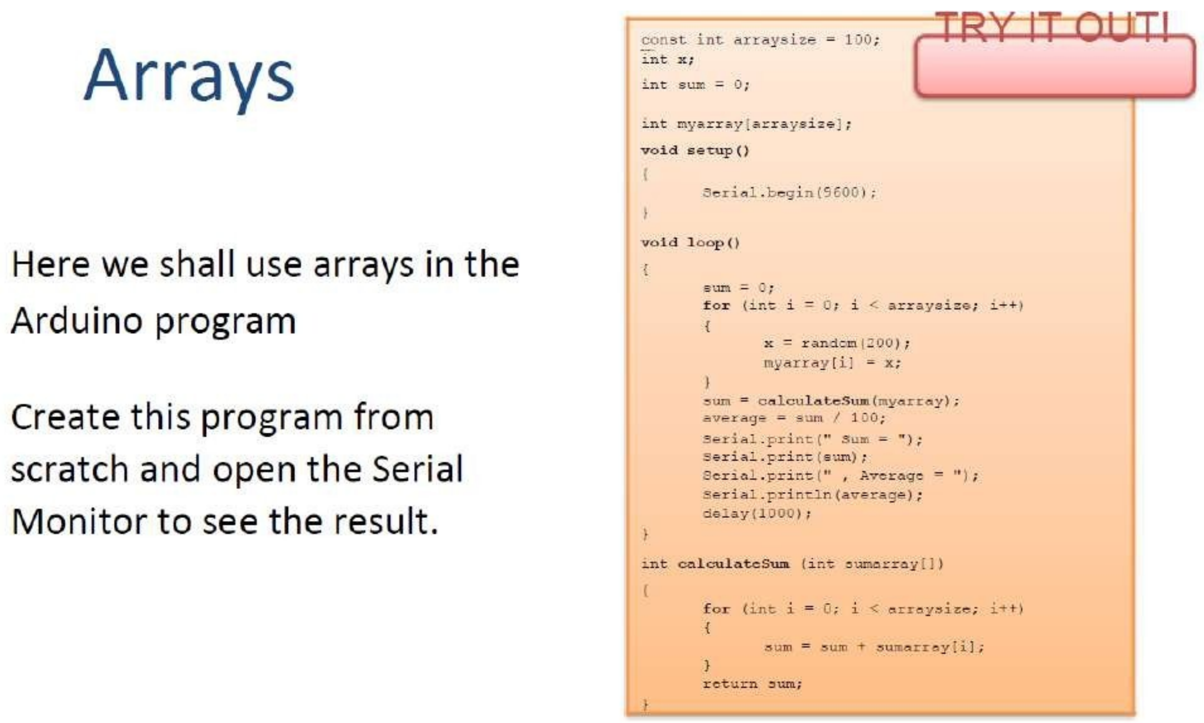
For Loop

In this program we use a For Loop to find the Sum of 100 Random Numbers.

Then we find the Average.

The Sum and Average should be written to the Serial Monitor.

```
int x; int sum = 0; float gjennomsnitt = 0;
void setup()
{
    Serial.begin(9600);
}
void loop()
{
    sum = 0;
    for (int i = 0; i<100; i++)
    {
        x = random(100);
        sum = sum + x;
    }

    average = sum / 100;
    Serial.print(" Sum = ");
    Serial.print(sum);
    Serial.print(" ,
    Average = ");
    Serial.println(average);
    delay(1000);
}
```



Arrays

Here we shall use arrays in the Arduino program

Create this program from scratch and open the Serial Monitor to see the result.

```
const int arraysize = 100;
int x;
int sum = 0;

int myarray[arraysize];
void setup()
{
        Serial.begin(9600);
}
void loop()
{
        sum = 0;
        for (int i = 0; i < arraysize; i++)
        {
                x = random(200);
                myarray[i] = x;
        }
        sum = calculateSum(myarray);
        average = sum / 100;
        Serial.print(" Sum = ");
        Serial.print(sum);
        Serial.print(" , Average = ");
        Serial.println(average);
        delay(1000);
}
int calculateSum (int sumarray[])
{
        for (int i = 0; i < arraysize; i++)
        {
                sum = sum + sumarray[i];
        }
        return sum;
}
```

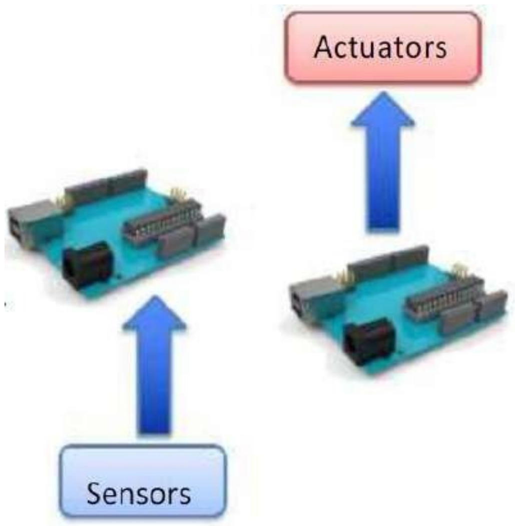
INTEGRATION OF SENSORS AND ACTUATORS WITH ARDUINO

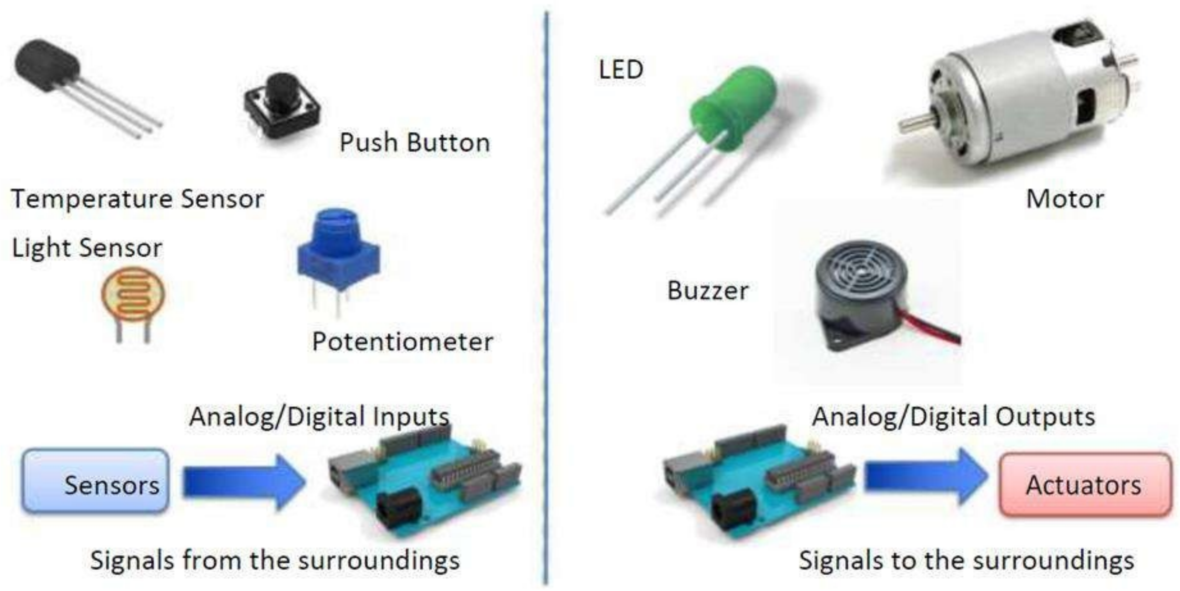
Sensors and Actuators:

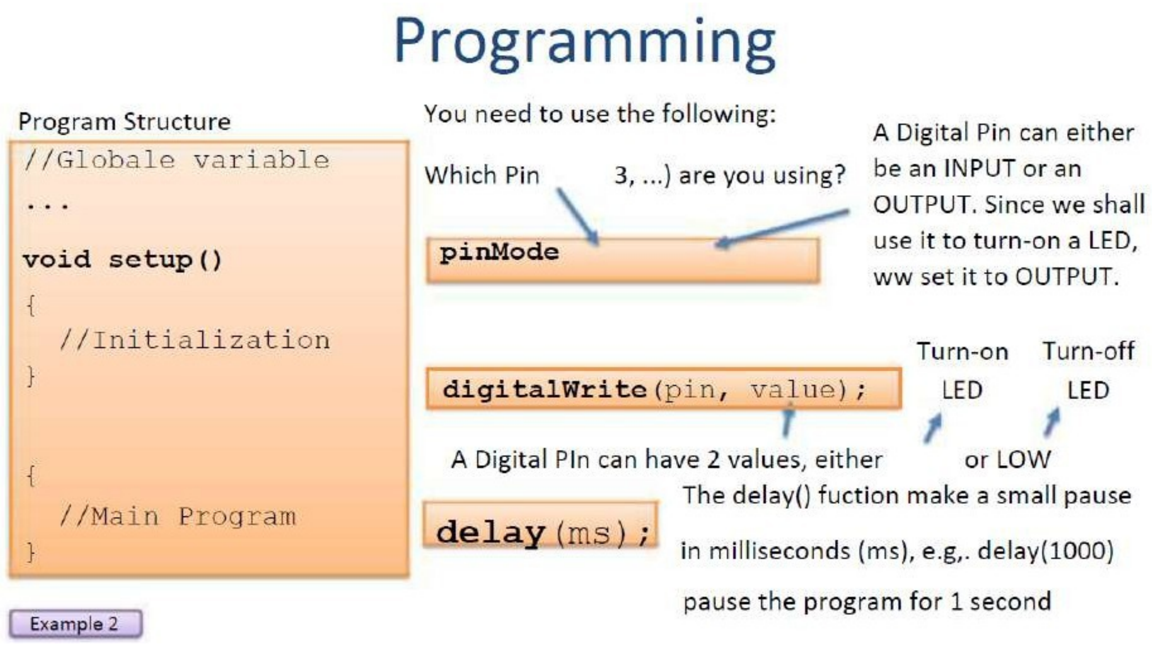
- A **Sensor** is a converter that measures a physical size and converts it to a signal that can be read by an instrument, data acquisition device, or an Arduino. **Examples:** temperature sensor, pressure sensor, etc.
- An **Actuator** is a kind of motor that moves or controls a mechanism or system. It is powered by an energy source, typical electric current, hydraulic fluid pressure, or air pressure, and converts this energy into motion. **Examples:** Engine, Pump, Valve, etc.
- The sensors and actuators can be either digital or analog.
- Some sensors and actuators have been made for Arduino, while others need to be connected in some circuit to work properly with Arduino.
- Many of these come with ready-made libraries for Arduino, so they are easy to use.

Examples:

1) Electrical Circuit.
2) Blinking LED.
3) Switch.
4) Potentiometer.
5) Temperature.
6) Light Sensor.
7) Thermistor.







Programming

Program Structure

```
//Globale variable
...
void setup()
{
  //Initialization
}


{
  //Main Program
}
```

Example 2

You need to use the following:

Which Pin 3, ...) are you using?

```
pinMode
```

A Digital Pin can either be an INPUT or an OUTPUT. Since we shall use it to turn-on a LED, ww set it to OUTPUT.

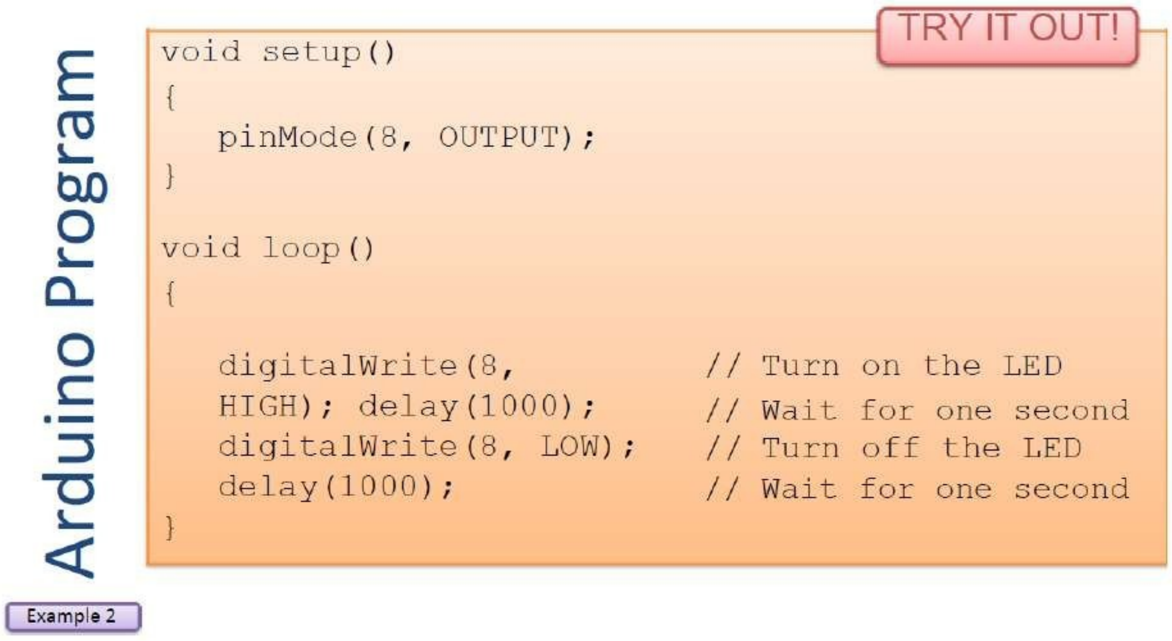
```
digitalWrite(pin, value);
```

Turn-on LED Turn-off LED

A Digital PIn can have 2 values, either or LOW

```
delay(ms);
```

The delay() fuction make a small pause in milliseconds (ms), e.g,. delay(1000) pause the program for 1 second



```
void setup()
{
   pinMode(8, OUTPUT);
}

void loop()
{

   digitalWrite(8,              // Turn on the LED
   HIGH); delay(1000);          // Wait for one second
   digitalWrite(8, LOW);        // Turn off the LED
   delay(1000);                 // Wait for one second
}
```

*****

UNIT – III

INTRODUCTION TO PYTHON

PROGRAMMING

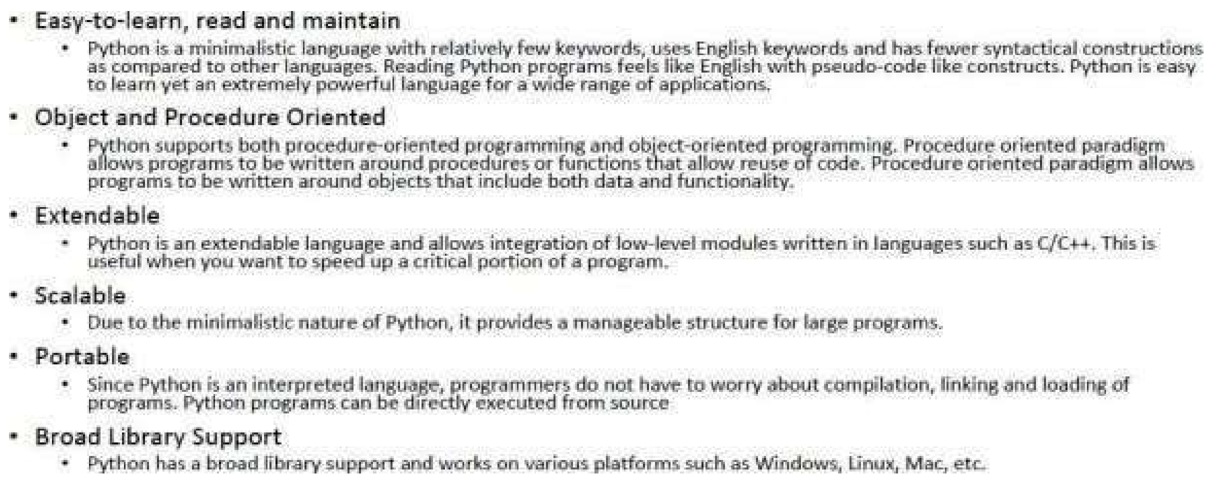
Python

Python is a general-purpose high level programming language and suitable for providing a solid foundation to the reader in the area of cloud computing.

The main characteristics of Python are:

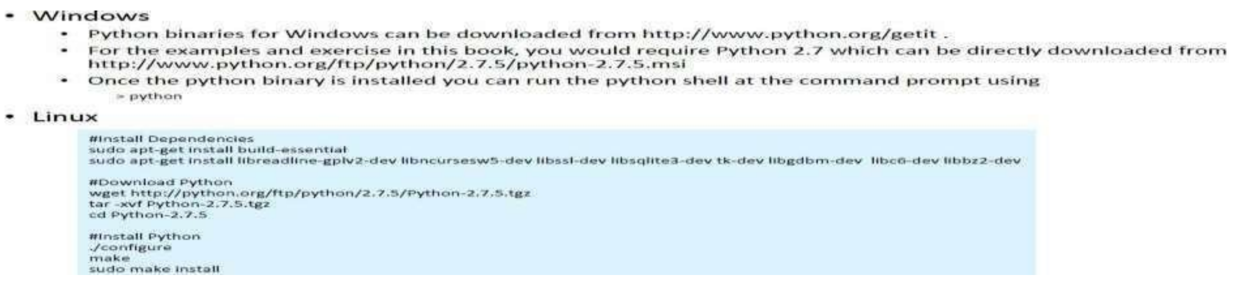
1) Multi-paradigm programming language.
2) Python supports more than one programming paradigms including object-oriented programming and structured programming.
3) Interpreted Language.
4) Python is an interpreted language and does not require an explicit compilation step.
5) The Python interpreter executes the program source code directly, statement by statement, as a processor or scripting engine does.
6) Interactive Language.
7) Python provides an interactive mode in which the user can submit commands at the Python prompt and interact with the interpreter directly.

Python Benefits



- Easy-to-learn, read and maintain
  - Python is a minimalistic language with relatively few keywords, uses English keywords and has fewer syntactical constructions as compared to other languages. Reading Python programs feels like English with pseudo-code like constructs. Python is easy to learn yet an extremely powerful language for a wide range of applications.
- Object and Procedure Oriented
  - Python supports both procedure-oriented programming and object-oriented programming. Procedure oriented paradigm allows programs to be written around procedures or functions that allow reuse of code. Procedure oriented paradigm allows programs to be written around objects that include both data and functionality.
- Extendable
  - Python is an extendable language and allows integration of low-level modules written in languages such as C/C++. This is useful when you want to speed up a critical portion of a program.
- Scalable
  - Due to the minimalistic nature of Python, it provides a manageable structure for large programs.
- Portable
  - Since Python is an interpreted language, programmers do not have to worry about compilation, linking and loading of programs. Python programs can be directly executed from source
- Broad Library Support
  - Python has a broad library support and works on various platforms such as Windows, Linux, Mac, etc.

Python – Setup:



- Windows
  - Python binaries for Windows can be downloaded from http://www.python.org/getit .
  - For the examples and exercise in this book, you would require Python 2.7 which can be directly downloaded from http://www.python.org/ftp/python/2.7.5/python-2.7.5.msi
  - Once the python binary is installed you can run the python shell at the command prompt using
    > python
- Linux

```
#Install Dependencies
sudo apt-get install build-essential
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev  libc6-dev libbz2-dev

#Download Python
wget http://python.org/ftp/python/2.7.5/Python-2.7.5.tgz
tar -xvf Python-2.7.5.tgz
cd Python-2.7.5

#Install Python
./configure
make
sudo make install
```

Datatypes

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are listed below.

Python Numbers

Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as int, float and complex class in Python. We can use the type () function to know which class a variable or a value belongs to and the is instance () function to check if an object belongs to a particular class.

Script.py

```
a = 5
print(a, "is of type", type(a))
a = 2.0
print(a, "is of type", type(a))
a = 1+2j
print(a, "is complex number?", is instance(1+2j,complex))
```

Integers can be of any length; it is only limited by the memory available. A floating-point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is integer, 1.0 is floating point number. Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part. Here are some examples.

```
>>> a = 1234567890123456789
>>> a
1234567890123456789
>>> b = 0.1234567890123456789
>>> b
0.12345678901234568

>>> c = 1+2j
>>> c
(1+2j)
```

Python List

List is an ordered sequence of items. It is one of the most used datatypes in Python and is very flexible. All the items in a list do not need to be of the same type. Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [].

```
>>> a = [1, 2.2, 'python']
```

We can use the slicing operator [ ] to extract an item or a range of items from a list. Index starts form 0 in Python.

Script.py

```
a = [5,10,15,20,25,30,35,40]
# a[2] = 15
print("a[2] = ", a[2])
# a[0:3] = [5, 10, 15]
print("a[0:3] = ", a[0:3])
# a[5:] = [30, 35, 40]
print("a[5:] = ", a[5:])
```

Lists are mutable, meaning; value of elements of a list can be altered.

```
>>> a = [1,2,3]
>>> a[2]=4
>>> a
[1, 2, 4]
```

Python Tuple

Tuple is an ordered sequence of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified. Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically. It is defined within parentheses () where items are separated by commas.

```
>>> t = (5,'program', 1+3j)
```

Script.py

```
t = (5,'program', 1+3j)
  # t[1] = 'program'
print("t[1] = ", t[1])
# t[0:3] = (5, 'program', (1+3j))
print("t[0:3] = ", t[0:3])
# Generates error
# Tuples are immutable
t[0] = 10
```

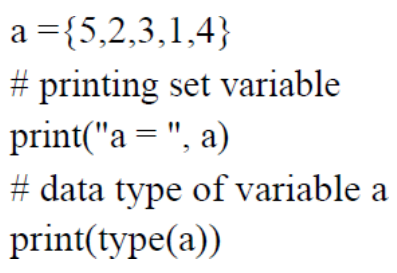
Python Strings

String is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.

```
>>> s = "This is a string"
>>> s = '''a multiline
```

Like list and tuple, slicing operator [ ] can be used with string. Strings are immutable.

Script.py

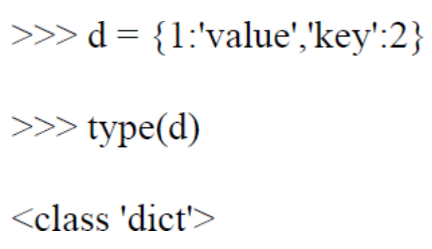


```
a ={5,2,3,1,4}
# printing set variable
print("a = ", a)
# data type of variable a
print(type(a))
```

We can perform set operations like union, intersection on two sets. Set has unique values. They eliminate duplicates. Since, set are unordered collection, indexing has no meaning. Hence the slicing operator [] does not work. It is generally used when we have a huge amount of data.

Dictionaries are optimized for retrieving data. We must know the key to retrieve the value. In Python, dictionaries are defined within braces {} with each item being a pair in the form key:

value. Key and value can be of any type.



```
>>> d = {1:'value','key':2}

>>> type(d)

<class 'dict'>
```

We use key to retrieve the respective value. But not the other way around.

Script.py

```
d ={1:'value','key':2}
print(type(d)) print("d[1] =
",d[1]); print("d['key'] = ",
d['key']);
# Generates error
print("d[2] = ",d[2]);
```

Python if...else Statement

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes. Decision making is required when we want to execute a code only if a certain condition is satisfied.

The if…elif…else statement is used in Python for decision making.

Python if Statement

Syntax

if test expression:

statement(s)

Here, the program evaluates the test expression and will execute statement(s) only if the text expression is True.

If the text expression is False, the statement(s) is not executed. In Python, the body of the if statement is indicated by the indentation. Body starts with an indentation and the first unindebted line marks the end. Python interprets non-zero values as True. None and 0 are interpreted as False.

Python if Statement Flowchart

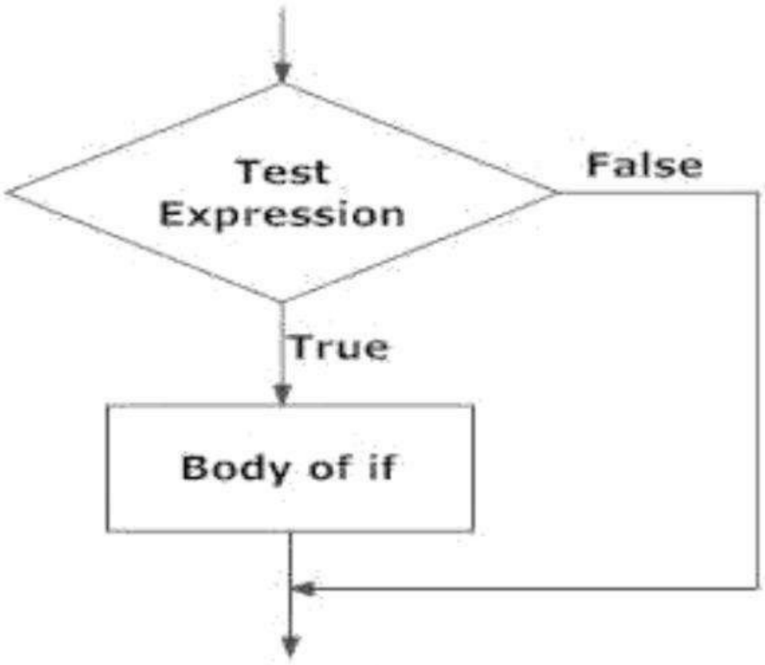


Fig: Operation of if statement

Example: Python if Statement

```
# If the number is positive, we print an appropriate message

num = 3
if num > 0:
    print(num, "is a positive number.")
print("This is always printed.")
num = -1
if num >0:
    print(num, "is a positive number.")
print("This is also always printed.")
```

When you run the program, the output will be:

```
3 is a positive number
This is always printed
This is also always printed.
```

In the above example, num > 0 is the test expression. The body of it is executed only if this evaluates to True.

When variable num is equal to 3, test expression is true and body inside body of it is executed. If variable num is equal to -1, test expression is false and body inside body of it is skipped. The print() statement falls outside of the if block (unindebted). Hence, it is executed regardless of the test expression.

Python if...else Statement

Syntax

```
if test expression:
   Body of if

else:
   Body of else
```

The if..else statement evaluates test expression and will execute body of if only when test condition is True.
If the condition is False, body of else is executed. Indentation is used to separate the blocks.

Python if..else Flowchart

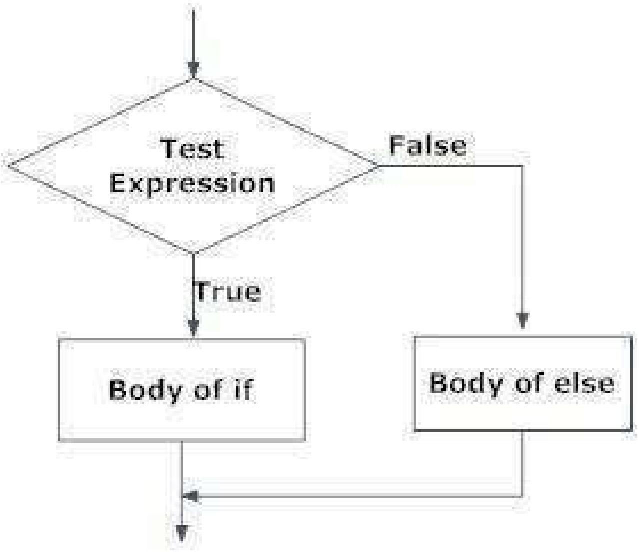


Fig: Operation of if...else statement

Example of if...else

```
# Program checks if the number is positive or negative
# And displays an appropriate message
num = 3
# Try these two variations as well.
# num = -5
# num = 0
if num >= 0:
   print("Positive or Zero")
else:
   print("Negative number")
```

In the above example, when num is equal to 3, the test expression is true and body of if is executed and body of else is skipped.

If num is equal to -5, the test expression is false and body of else is executed and body of if is skipped.

If num is equal to 0, the test expression is true and body of if is executed and body of else is skipped.

Python if...elif...else Statement

Syntax

```
if test expression:
   Body of if
elif test expression:
   Body of elif
else:
   Body of else
```

The elif is short for else if. It allows us to check for multiple expressions. If the condition for if is False, it checks the condition of the next elif block and so on. If all the conditions are False, body of else is executed. Only one block among the several if...elif...else blocks is executed according to the condition. The if block can have only one else block. But it can have multiple elifblocks.

Flowchart of if...elif...else

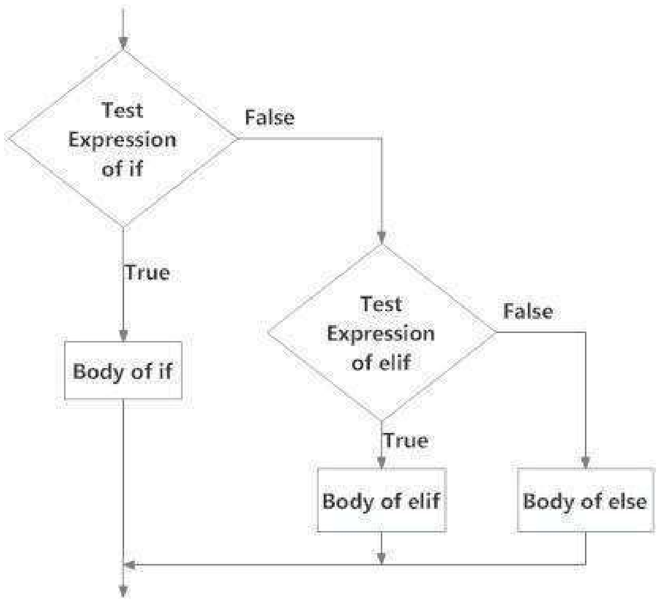


Fig: Operation of if...elif...else statement

Example of if...elif...else

```
# In this program,
# we check if the number is positive or
# negative or zero and
# display an appropriate
message num = 3.4
# Try these two variations as well:
# num = 0
# num = -4.5
if num > 0:
   print("Positive number")
elif num == 0:
   print("Zero")
else:
   print("Negative number")
```

When variable num is positive, Positive number is printed. If

num is equal to 0, Zero is printed.

If num is negative, Negative number is printed

Python Nested if statements

We can have a if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming. Any number of these statements can be nested inside one another.

Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

Python Nested if Example

```
# In this program, we input a number
# check if the number is positive or
# negative or zero anddisplay
# an appropriate message
# This time we use nested if

num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

Output 1

```
Enter a number: 5
Positive number
Output 2
Enter a number: -1
Negative number
Output 3
Enter a number: 0
Zero
```

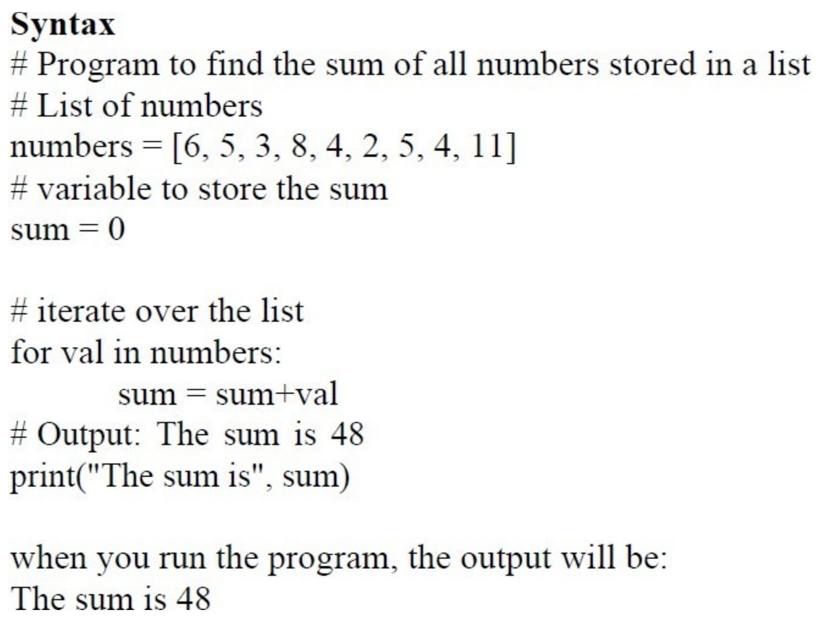
Python for Loop

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal. Syntax of for Loop for Val in sequence:

Body of for

Here, Val is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach th e last item in the sequence. The body of for loop is separated from the rest of the code using indentation.



Syntax

```
# Program to find the sum of all numbers stored in a list
# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
        sum = sum+val
# Output:  The  sum  is  48
print("The sum is", sum)

when you run the program, the output will be:
The sum is 48
```

The range () function

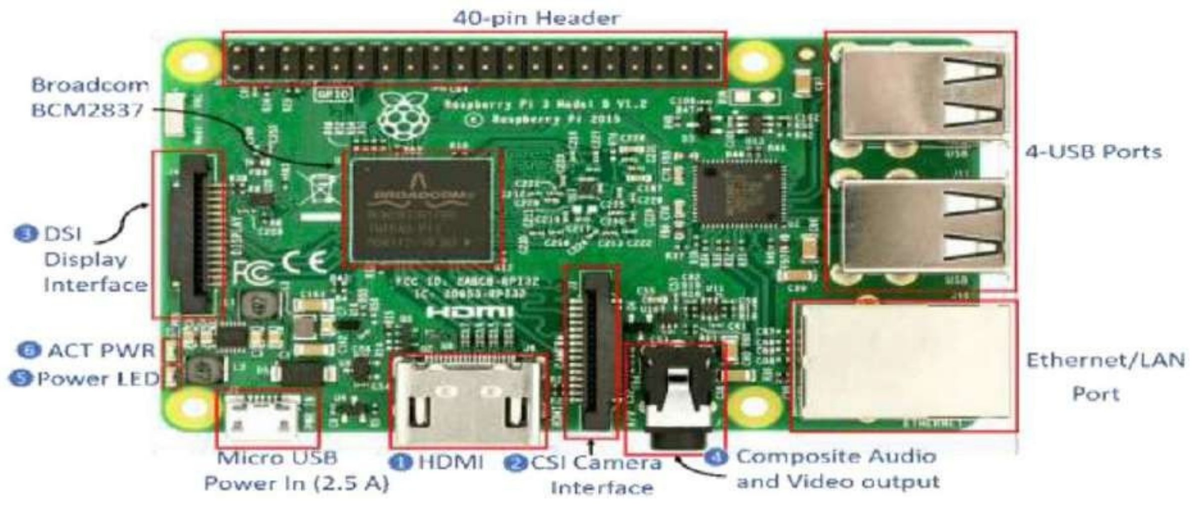
We can generate a sequence of numbers using range () function. range (10) will generate numbers from 0 to 9 (10 numbers). We can also define the start, stop and step size as range (start, stop, step size). step size defaults to 1 if not provided. This function does not store all the values in Emory, it would be inefficient. So, it remembers the start, stop, step size and generates the next number on the go.
To force this function to output all the items, we can use the function list().
The following example will clarify this.

```
# Output: range(0,
10) print(range(10))
# Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(list(range(10)))
# Output: [2, 3, 4, 5, 6, 7]
print(list(range(2, 8)))
# Output: [2, 5, 8, 11, 14, 17]
print(list(range(2, 20, 3)))
```

We can use the range() function in for loops to iterate through a sequence of numbers. It can be combined with the len() function to iterate though a sequence using indexing. Here is an example.

```
# Program to iterate through a list using indexing
genre = ['pop', 'rock', 'jazz']
```

INTRODUCTION TO RASPBERRY PI

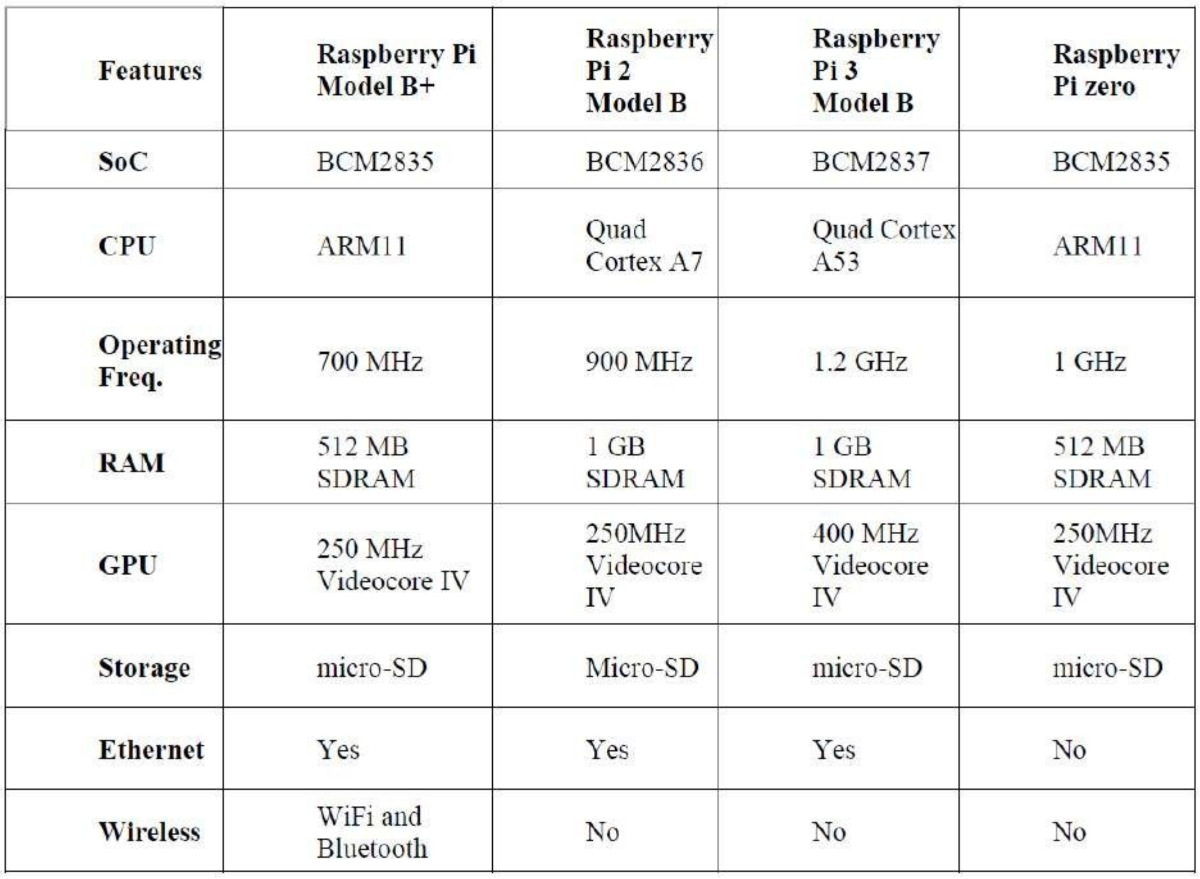


- Raspberry Pi is a low-cost, credit card-sized computer that connects to a computer monitor or TV using HDMI, and uses a standard keyboard and mouse. It can run a host of operating systems, such as Raspbian (Debian Linux), Android, Windows 10, IoT Core, etc.
- The Raspberry Pi is a series of low-cost, programmable computers that include a set of GPIO, or 'General Purpose Input Output', pins that can

be used to connect and control external electronic devices, and to create Internet of Things (IoT) solutions.

- Raspberry Pi was developed as an educational computer. ... The name Raspberry Pi is derived from the fruit pie, raspberry pie. This is because many companies in the computer neighbourhood where Raspberry Pi was based used fruit names such as Apple and apricot as names for their companies and products.
- Raspberry Pi is a small single board computer. By connecting peripherals like Keyboard, mouse, display to the Raspberry Pi, it will act as a mini personal computer.
- Raspberry Pi is popularly used for real time Image/Video Processing, IoT based applications and Robotics applications.
- Raspberry Pi is slower than laptop or desktop but is still a computer which can provide all the expected features or abilities, at a low power consumption.
- We can install several Third-Party versions of OS like Ubuntu, Archlinux, RISC OS, Windows 10 IOT Core, etc.
- Raspbian OS is official Operating System available for free to use. This OS is efficiently optimized to use with Raspberry Pi. Raspbian have GUI which includes tools for Browsing, Python programming, office, games, etc.
- We should use SD card (minimum 8 GB recommended) to store the OS (operating System).
- Raspberry Pi is more than computer as it provides access to the on-chip hardware i.e. GPIOs for developing an application. By accessing GPIO, we can connect devices like LED, motors, sensors, etc and can control them too.
- The CPU speed of Raspberry Pi varies from 700 MHz to 1.2 GHz. Also, it has on-board SDRAM that ranges from 256 MB to 1 GB.
- Raspberry Pi also provides on-chip SPI, I2C, I2S and UART modules.
- There are different versions of raspberry pi available as listed below:
  1) Raspberry Pi 1 Model A

2) Raspberry Pi 1 Model A+
3) Raspberry Pi 1 Model B
4) Raspberry Pi 1 Model B+
5) Raspberry Pi 2 Model B
6) Raspberry Pi 3 Model B
7) Raspberry Pi Zero



Features	**Raspberry Pi Model B+**	**Raspberry Pi 2 Model B**	**Raspberry Pi 3 Model B**	**Raspberry Pi zero**
SoC	BCM2835	BCM2836	BCM2837	BCM2835
CPU	ARM11	Quad Cortex A7	Quad Cortex A53	ARM11
Operating Freq.	700 MHz	900 MHz	1.2 GHz	1 GHz
RAM	512 MB SDRAM	1 GB SDRAM	1 GB SDRAM	512 MB SDRAM
GPU	250 MHz Videocore IV	250MHz Videocore IV	400 MHz Videocore IV	250MHz Videocore IV
Storage	micro-SD	Micro-SD	micro-SD	micro-SD
Ethernet	Yes	Yes	Yes	No
Wireless	WiFi and Bluetooth	No	No	No

Some Hardware Components shown above are mention below:

1) **HDMI (High-Definition Multimedia Interface):** It is used for transmitting uncompressed video or digital audio data to the Computer Monitor, Digital TV, etc. Generally, this HDMI port helps to connect Raspberry Pi to the Digital television.
2) **CSI Camera Interface:** CSI (Camera Serial Interface) interface provides a connection in between Broadcom Processor and Pi camera. This interface provides electrical connections between two devices.
3) **DSI Display Interface:** DSI (Display Serial Interface) Display Interface is used for connecting LCD to the Raspberry Pi using 15-pin ribbon

cable. DSI provides fast High resolution display interface specifically used for sending video data directly from GPU to the LCD display.

4) **Composite Video and Audio Output:** The composite Video and Audio output port carries video along with audio signal to the Audio/Video systems.
5) **Power LED:** It is a RED colored LED which is used for Power indication. This LED will turn ON when Power is connected to the Raspberry Pi. It is connected to 5V directly and will start blinking whenever the supply voltage drops below 4.63V.
6) **ACT PWR:** ACT PWR is Green LED which shows the SD card activity.

Getting Started with Raspberry Pi:

HDMI Cable

VGA Cable

HDMI to VGA Converter

- To get started with Raspberry Pi, we have to store required OS on SD card.
- Now to store OS on SD card we need to install OS on SD card. If you want to know how to install/store OS on SD card.
- Here, we installed the Raspbian OS on SD card.
- Now, we have an SD card with installed OS and Raspberry Pi Board.
- Initially to use raspberry Pi we need computer monitor or Digital Display.

- We can directly connect Raspberry Pi to the Digital Display using HDMI cable.

But, if we have a computer monitor (VGA Display), then we need an HDMI to VGA converter along with a VGA cable for connecting Raspberry Pi with monitors. HDMI to VGA converter and VGA cable is shown below.

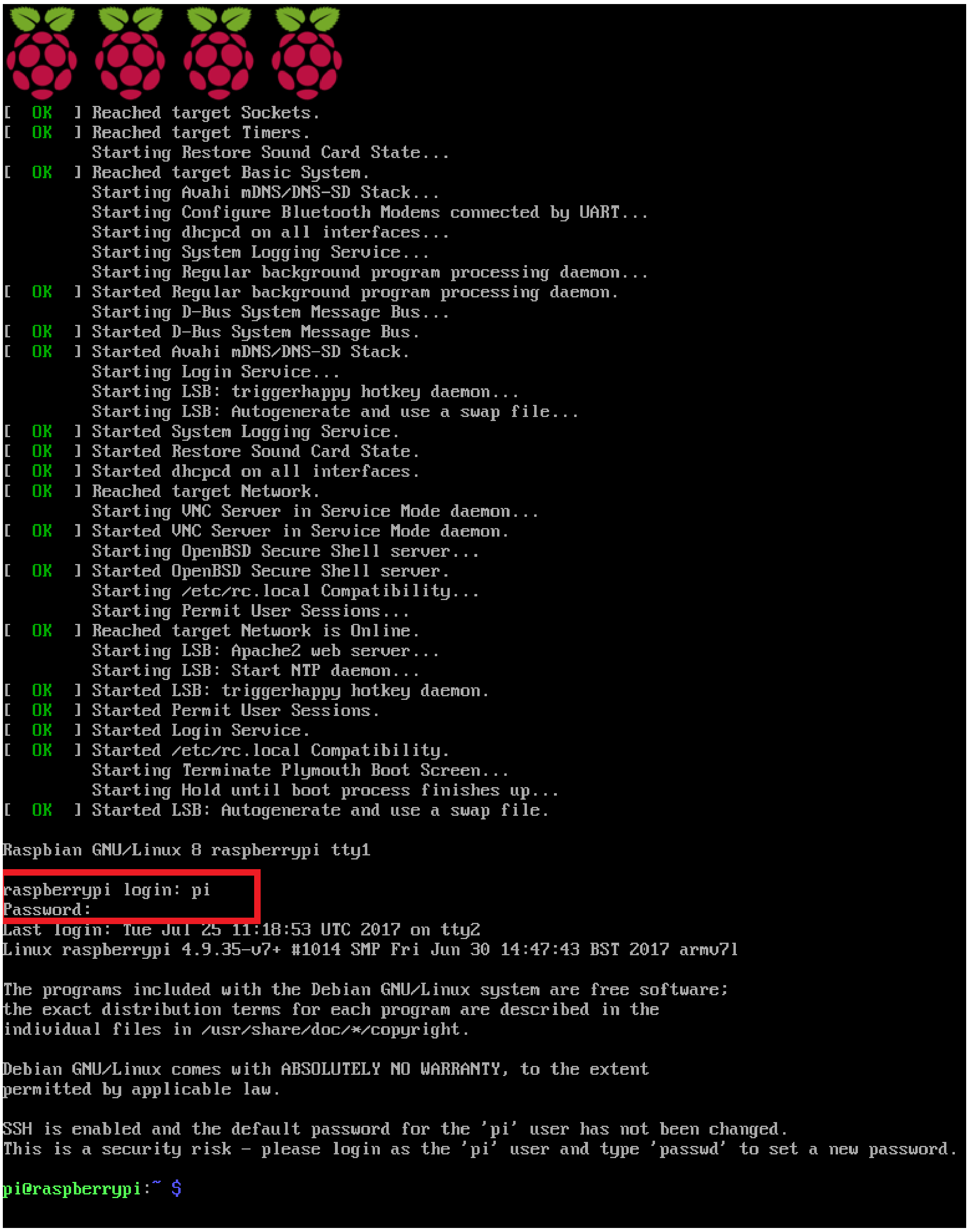
Now, connect the Raspberry Pi to the Display/monitor and Power-On Raspberry Pi. We will get a Black command window asking for Login and Password as shown below.

Then, use the following login name and password

raspberrypi Login: pi

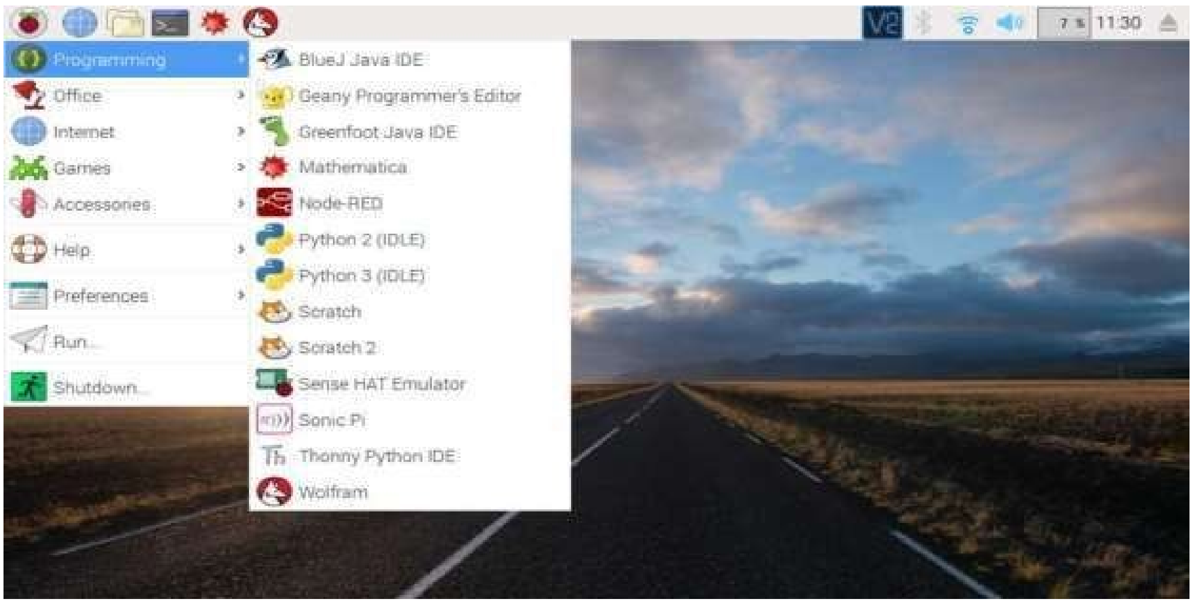
Password: raspberry

- This is the default user name and password. You can change the password after the first login.
- The above command window can be used to operate Raspberry Pi.
- To get GUI environment on Raspberry Pi, use below command.

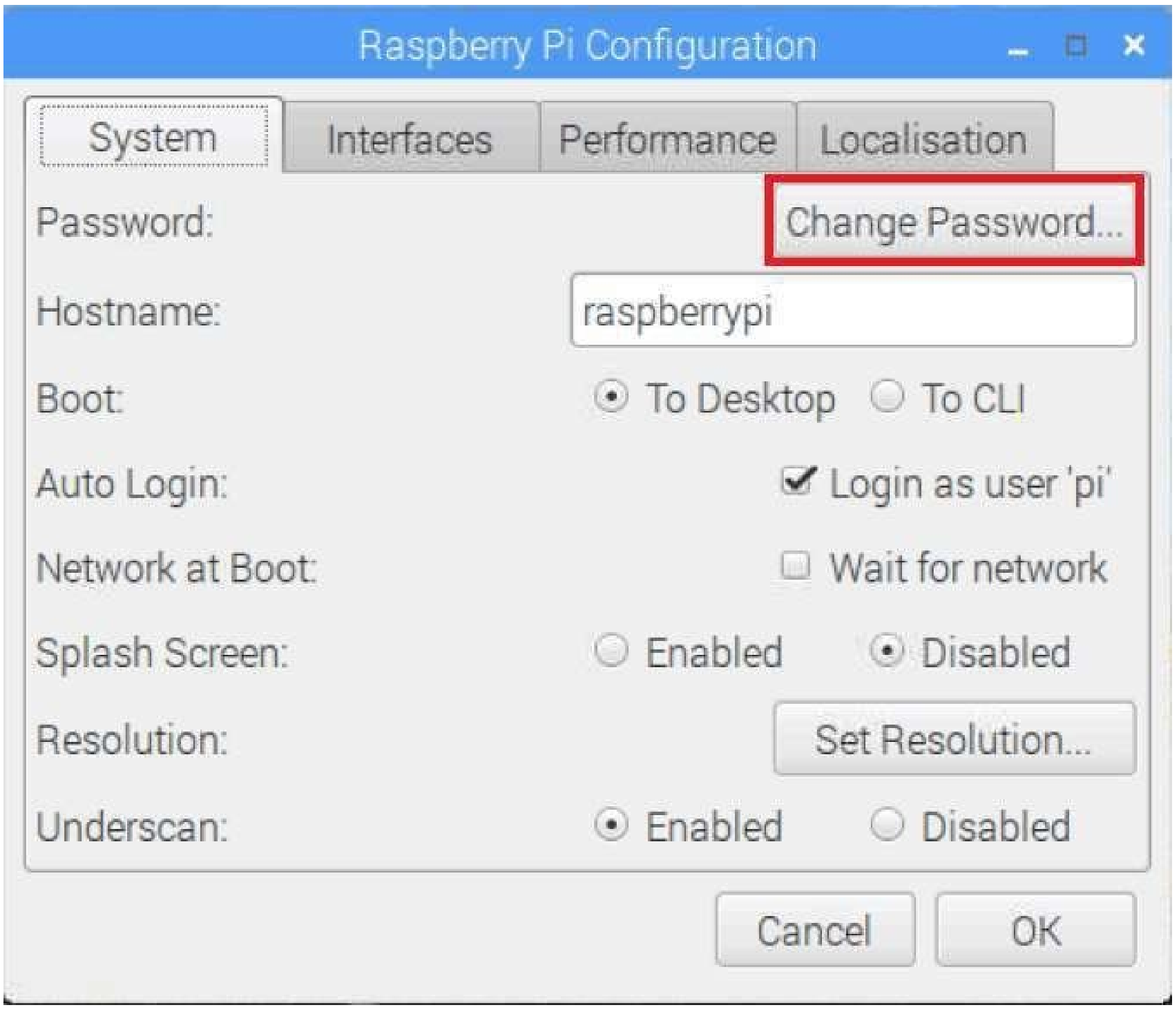


```
[  OK  ] Reached target Sockets.
[  OK  ] Reached target Timers.
         Starting Restore Sound Card State...
[  OK  ] Reached target Basic System.
         Starting Avahi mDNS/DNS-SD Stack...
         Starting Configure Bluetooth Modems connected by UART...
         Starting dhcpcd on all interfaces...
         Starting System Logging Service...
         Starting Regular background program processing daemon...
[  OK  ] Started Regular background program processing daemon.
         Starting D-Bus System Message Bus...
[  OK  ] Started D-Bus System Message Bus.
[  OK  ] Started Avahi mDNS/DNS-SD Stack.
         Starting Login Service...
         Starting LSB: triggerhappy hotkey daemon...
         Starting LSB: Autogenerate and use a swap file...
[  OK  ] Started System Logging Service.
[  OK  ] Started Restore Sound Card State.
[  OK  ] Started dhcpcd on all interfaces.
[  OK  ] Reached target Network.
         Starting VNC Server in Service Mode daemon...
[  OK  ] Started VNC Server in Service Mode daemon.
         Starting OpenBSD Secure Shell server...
[  OK  ] Started OpenBSD Secure Shell server.
         Starting /etc/rc.local Compatibility...
         Starting Permit User Sessions...
[  OK  ] Reached target Network is Online.
         Starting LSB: Apache2 web server...
         Starting LSB: Start NTP daemon...
[  OK  ] Started LSB: triggerhappy hotkey daemon.
[  OK  ] Started Permit User Sessions.
[  OK  ] Started Login Service.
[  OK  ] Started /etc/rc.local Compatibility.
         Starting Terminate Plymouth Boot Screen...
         Starting Hold until boot process finishes up...
[  OK  ] Started LSB: Autogenerate and use a swap file.

Raspbian GNU/Linux 8 raspberrypi tty1

raspberrypi login: pi
Password:
Last login: Tue Jul 25 11:18:53 UTC 2017 on tty2
Linux raspberrypi 4.9.35-v7+ #1014 SMP Fri Jun 30 14:47:43 BST 2017 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.

pi@raspberrypi:~ $
```

On display, there is a symbol of raspberry to the top-left corner of display. After clicking on it, we will get menu as shown below



Then, click on change password option shown below:



Password:
Hostname: raspberrypi
Boot: [✓ To Desktop] To CLI
Auto Login: [✓ Login as user 'pi']
Network at Boot: Wait for network
Splash Screen: Enabled [✓ Disabled]
Resolution:
Underscan: [✓ Enabled] Disabled

How to write C program on Raspbian OS

So, let's write our First C code on Raspbian and execute it.
First Create Empty file and label it with .c extension.
Now write a small program to print "Hello World"

```
#include<stdio.h>
 int main()
{
printf("Hello World");
return 0;
 }
```

After writing the code, open terminal (ctrl+alt+t) to execute it. Then, type following commands for compiling and execution.

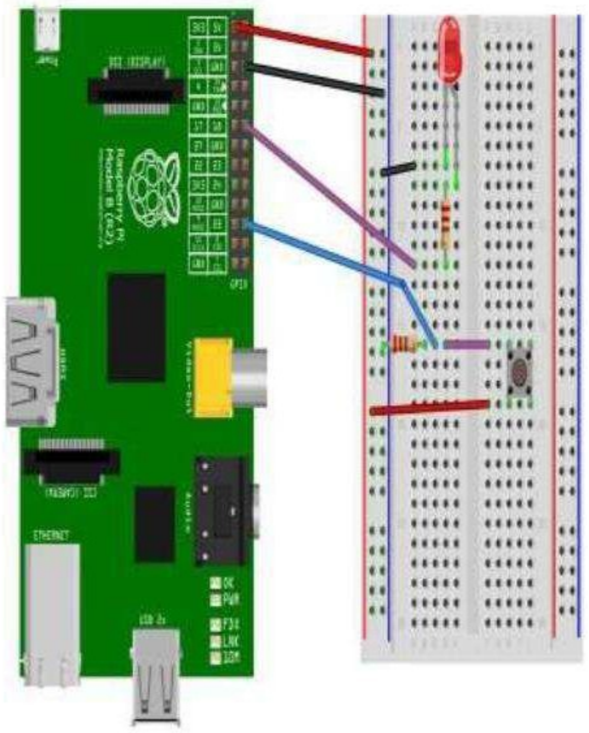
RASPBERRY PI INTERFACES

1) **Serial:** The serial interface on Raspberry Pi has receive (Rx) and transmit (Tx) pins for communication with serial peripherals.
2) **SPI:** Serial Peripheral Interface (SPI) is a synchronous serial data protocol used for communicating with one or more peripheral devices.
3) **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).

Raspberry Pi Example:

Interfacing LED and switch with Raspberry Pi:

```
from time import sleep
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM
)
#Switch Pin
GPIO.setup(25,
GPIO.IN)  #LED Pin
GPIO.setup(18, GPIO.OUT)
state=false
def toggleLED(pin):
        state = not state
        GPIO.output(pin,
        state)
while True:
        try:
                if (GPIO.input(25) ==
                        True):
                        toggleLED(pin)
                sleep(.01)
                except
                        KeyboardInterrupt:
                        exit()
```



How can IoT Applications use Raspberry Pi?

With an in-built quadcore processor, Raspberry Pi can serve as the "Internet Gateway" for IoT devices. Powered by a cloud network, Pi acts as a web server for uploading and transiting sensor data on IoT platforms.

Newer Raspberry Pi models come with a standard 10/100 Mbit/s Ethernet port that you can use to connect the device to the Internet. You simply need to plug an Ethernet cable to the Raspberry Pi and connect it to your Internet router.

What are the disadvantages of Raspberry Pi?

1) Not able to run Windows Operating system.
2) Impractical as a Desktop Computer.
3) Graphics Processor Missing.
4) Missing eMMC Internal Storage. Since the raspberry pi doesn't have any internal storage, it requires a micro-SD card to work as an internal storage.

INTERFACING RASPBERRY PI WITH BASIC PERIPHERALS

- The other two serial interfaces are the Serial Peripheral Interface (SPI) and Inter-Integrated-Circuit bus (I2C).
- SPI on the Pi allows for up to two attached devices, while I2C potentially allows for many devices, as long as their addresses don't conflict.
- Pi Camera module is a camera which can be used to take pictures and high-definition video.
- Raspberry Pi Board has CSI (Camera Serial Interface) interface to which we can attach Pi Camera module directly. This Pi Camera module can attach to the Raspberry Pi's CSI port using 15-pin ribbon cable.

Pi Camera Module Interface with Raspberry Pi using Python:

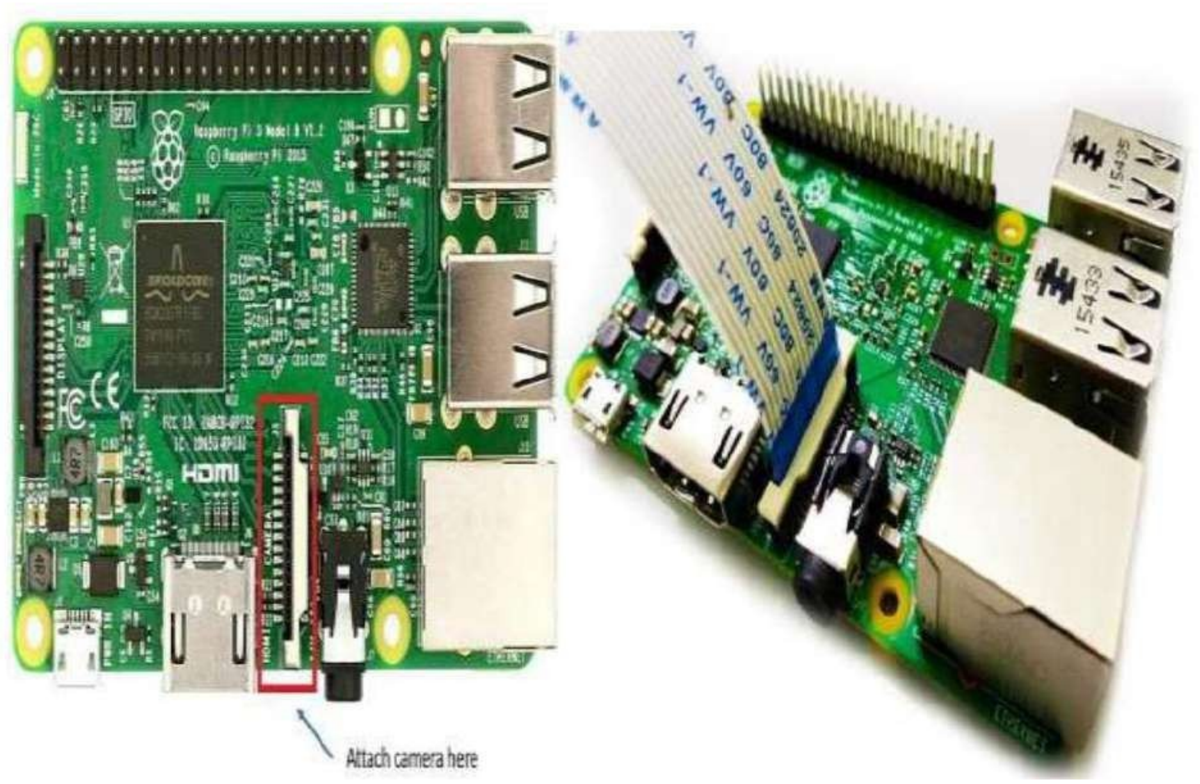
Fig: Pi Camera Module (v1.3)

- Pi Camera module is a camera which can be used to take pictures and high-definition video.
- Raspberry Pi Board has CSI (Camera Serial Interface) interface to which we can attach Pi Camera module directly.
- This Pi Camera module can attach to the Raspberry Pi’s CSI port using 15-pin ribbon cable.

Features of Pi Camera: Here, we have used Pi camera v1.3. Its features are listed below,

- Resolution – 5 MP
- HD Video recording – 1080p @30fps, 720p @60fps, 960p @45fps and so on.
- It Can capture wide, still (motionless) images of resolution 2592x1944 pixels.
- CSI Interface enabled.

How to attach Pi Camera to Raspberry Pi?



Now, we can use Pi Camera for capturing images and videos using Raspberry Pi.

Before using Pi Camera, we need to enable camera for its working.

How to Enable Camera functionality on Raspberry Pi?

For enabling camera in Raspberry Pi, open raspberry pi configuration using following command

“sudo raspi-config”

then select Interfacing options in which select camera option to enable its functionality.

reboot Raspberry Pi.

Now we can access camera on Raspberry Pi.

Now we can capture images and videos using Pi Camera on Raspberry Pi.

Example:

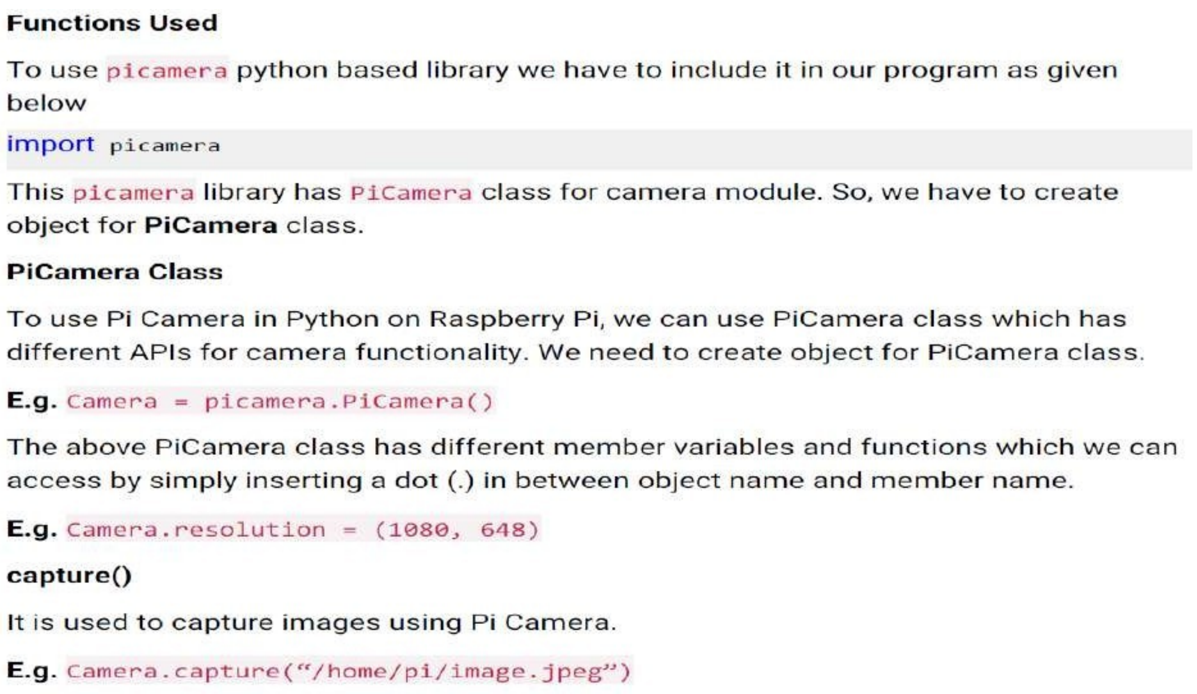
Capture images and save it to the specified directory.

We can capture images using Python. Here, we will write a Python program to capture images using Pi Camera on Raspberry Pi.

Here, we have used Pi Camera package(library) which provides different classes for Raspberry Pi. Out of which we are mainly interested in Pi Camera class which is for camera module.

Python Program for Image Capture:

```
import picamera
from time import sleep
#create object for PiCamera class
camera = picamera.PiCamera()
#set resolution
camera.resolution = (1024, 768)
camera.brightness = 60
camera.start_preview()
#add text on image
camera.annotate_text = 'Hi Pi User'
sleep(5)
#store image
camera.capture('image1.jpeg')
camera.stop_preview()
```



Functions Used

To use `picamera` python based library we have to include it in our program as given below

```
import picamera
```

This `picamera` library has `PiCamera` class for camera module. So, we have to create object for **PiCamera** class.

PiCamera Class

To use Pi Camera in Python on Raspberry Pi, we can use PiCamera class which has different APIs for camera functionality. We need to create object for PiCamera class.

E.g. `Camera = picamera.PiCamera()`

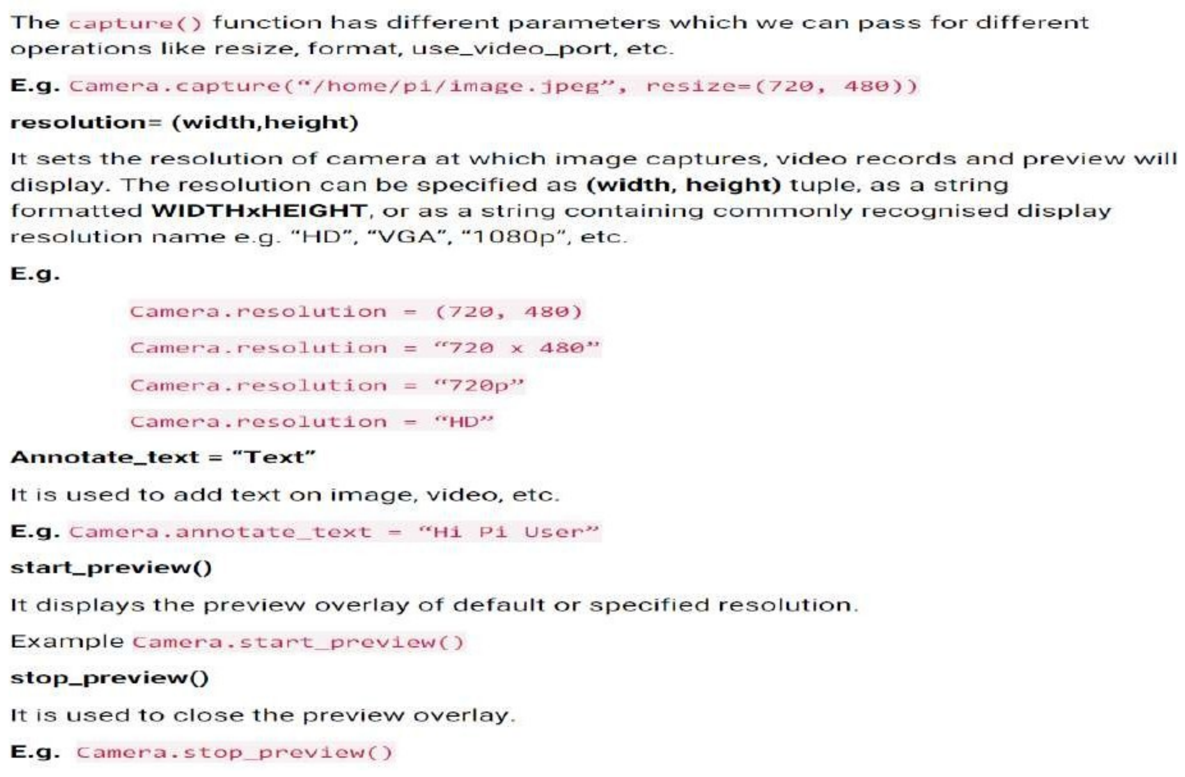
The above PiCamera class has different member variables and functions which we can access by simply inserting a dot (.) in between object name and member name.

E.g. `Camera.resolution = (1080, 648)`

capture()

It is used to capture images using Pi Camera.

E.g. `Camera.capture("/home/pi/image.jpeg")`



The `capture()` function has different parameters which we can pass for different operations like resize, format, use_video_port, etc.

E.g. `Camera.capture(“/home/pi/image.jpeg”, resize=(720, 480))`

resolution= (width,height)

It sets the resolution of camera at which image captures, video records and preview will display. The resolution can be specified as **(width, height)** tuple, as a string formatted **WIDTHxHEIGHT**, or as a string containing commonly recognised display resolution name e.g. “HD”, “VGA”, “1080p”, etc.

E.g.

```
Camera.resolution = (720, 480)
Camera.resolution = “720 x 480”
Camera.resolution = “720p”
Camera.resolution = “HD”
```

Annotate_text = “Text”

It is used to add text on image, video, etc.

E.g. `Camera.annotate_text = “Hi Pi User”`

start_preview()

It displays the preview overlay of default or specified resolution.

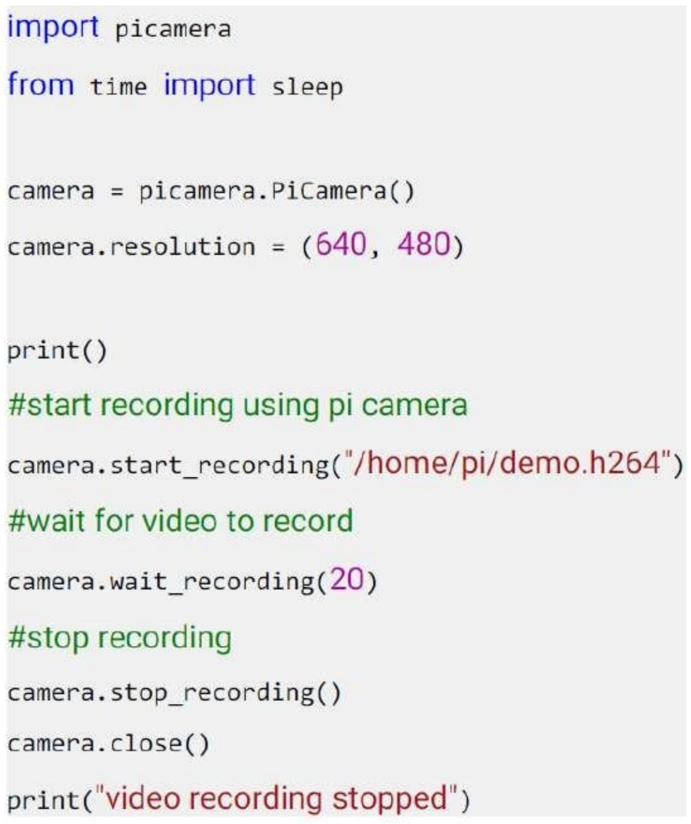
Example `Camera.start_preview()`

stop_preview()

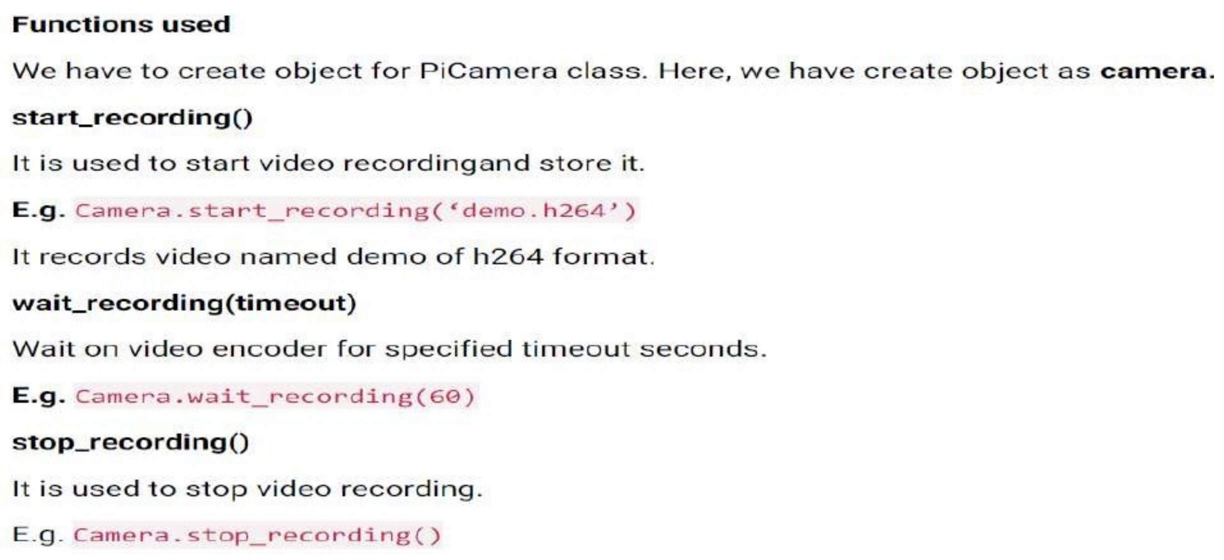
It is used to close the preview overlay.

E.g. `Camera.stop_preview()`

Python Program for Video Recording:



```
import picamera
from time import sleep

camera = picamera.PiCamera()
camera.resolution = (640, 480)

print()
#start recording using pi camera
camera.start_recording("/home/pi/demo.h264")
#wait for video to record
camera.wait_recording(20)
#stop recording
camera.stop_recording()
camera.close()
print("video recording stopped")
```



Functions used

We have to create object for PiCamera class. Here, we have create object as **camera**.

start_recording()

It is used to start video recordingand store it.

E.g. `Camera.start_recording('demo.h264')`

It records video named demo of h264 format.

wait_recording(timeout)

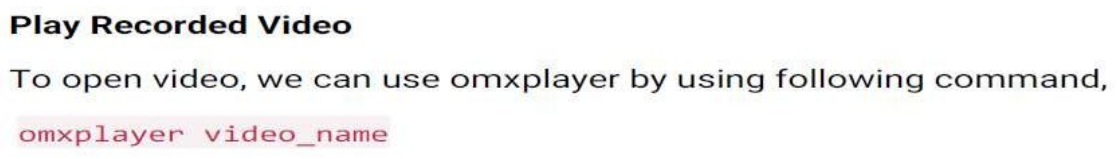
Wait on video encoder for specified timeout seconds.

E.g. `Camera.wait_recording(60)`

stop_recording()

It is used to stop video recording.

E.g. `Camera.stop_recording()`



Play Recorded Video

To open video, we can use omxplayer by using following command,

```
omxplayer video_name
```

IMPLEMENTATION OF IOT WITH RASPBERRY PI

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.

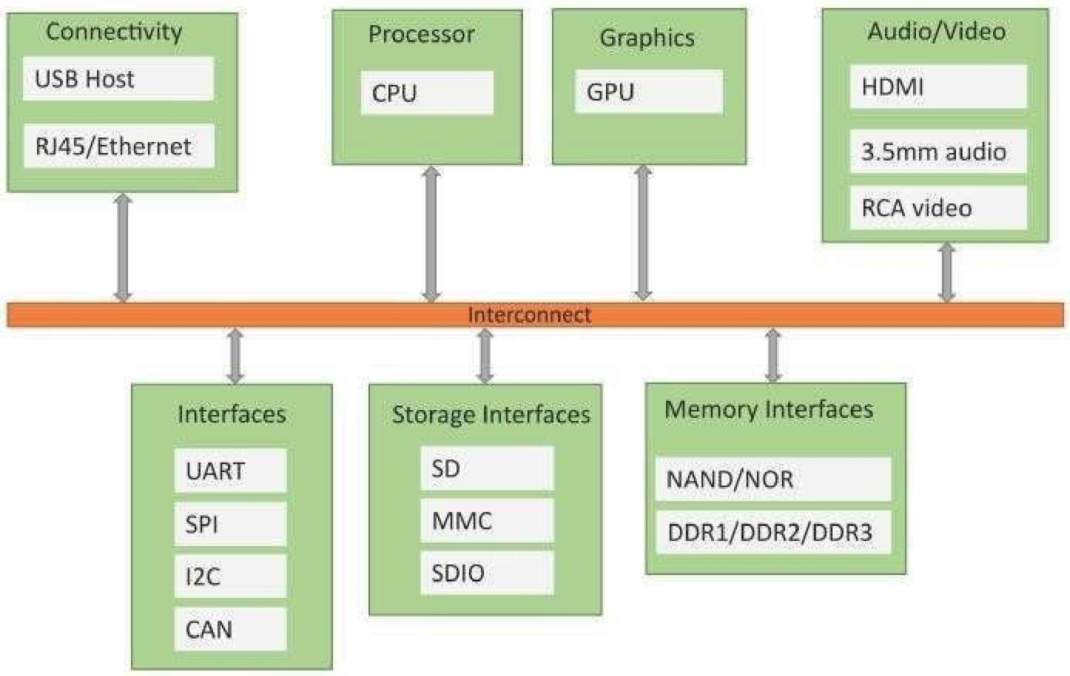
IoT Device Examples:

1) A home automation device that allows remotely monitoring the status of appliances and controlling the appliances.
2) An industrial machine which sends information abouts its operation and health monitoring data to a server.
3) A car which sends information about its location to a cloud-based service.
4) 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:

1) **Sensing:** Sensors can be either on-board the IoT device or attached to the device.
2) **Actuation:** IoT devices can have various types of actuators attached that allow taking actions upon the physical entities in the vicinity of the device.
3) **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.
4) **Analysis & Processing:** Analysis and processing modules are responsible for making sense of the collected data.

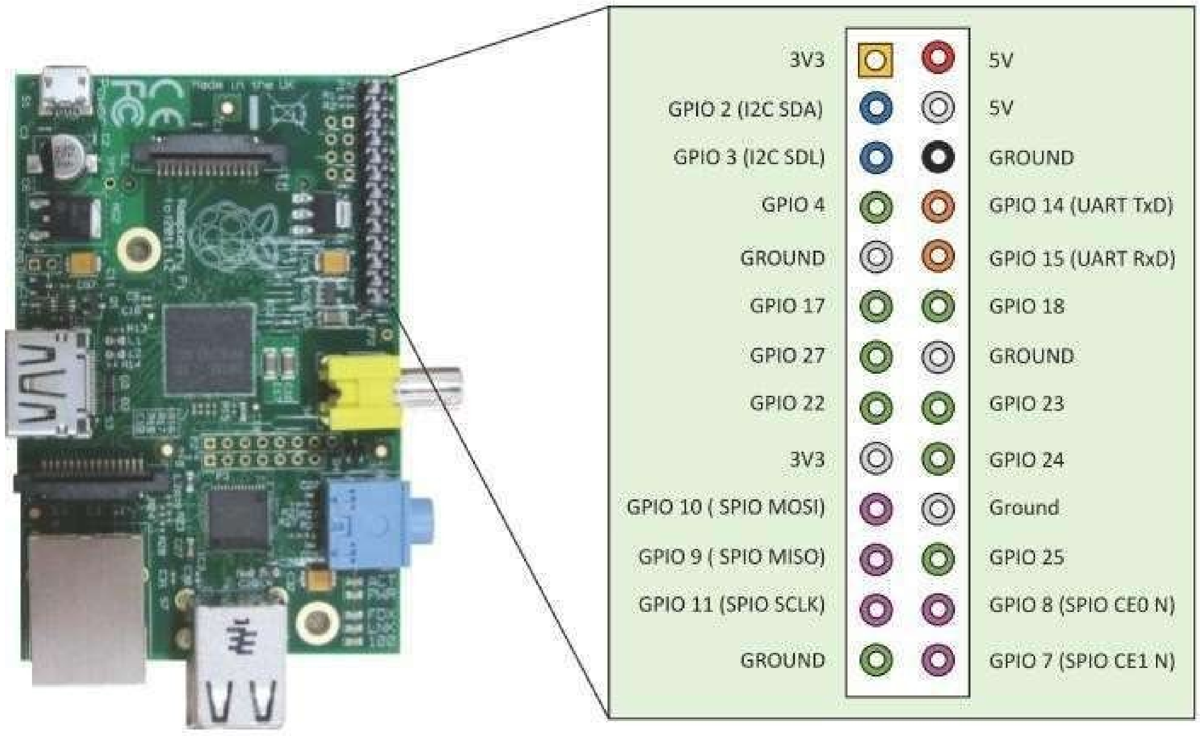
Block diagram of an IoT Device:



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 GPIO:


3V3
5V
GPIO 2 (I2C SDA)
5V
GPIO 3 (I2C SDL)
GROUND
GPIO 4
GPIO 14 (UART TxD)
GROUND
GPIO 15 (UART RxD)
GPIO 17
GPIO 18
GPIO 27
GROUND
GPIO 22
GPIO 23
3V3
GPIO 24
GPIO 10 (SPIO MOSI)
Ground
GPIO 9 (SPIO MISO)
GPIO 25
GPIO 11 (SPIO SCLK)
GPIO 8 (SPIO CE0 N)
GROUND
GPIO 7 (SPIO CE1 N)

*****

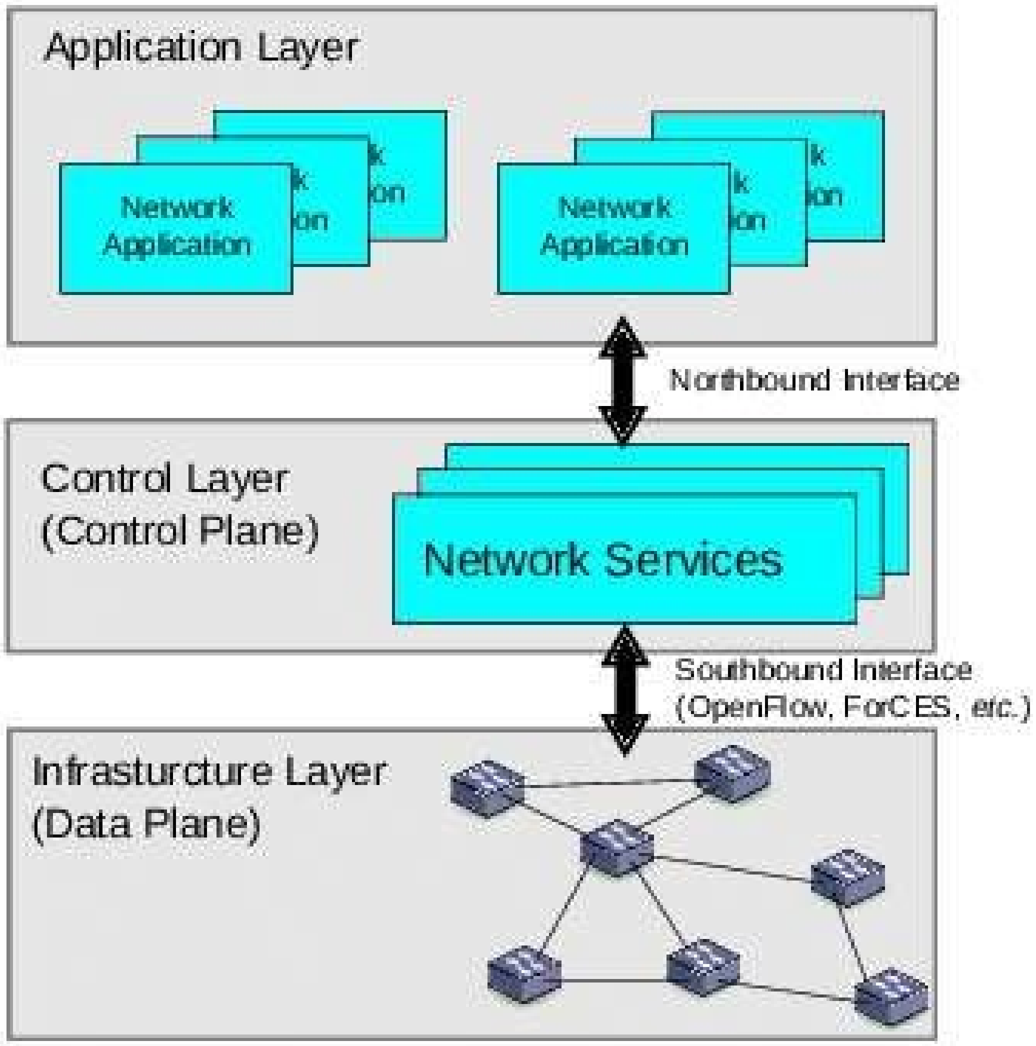
UNIT – IV

INTRODUCTION TO SOFTWARE DEFINED NETWORK (SDN)

- Software-Defined Networking (SDN) is a networking architecture that separates the control plane from the data plane and centralizes the network controller.
- Software-based SDN controllers maintain a unified view of the network and make configuration, management and provisioning simpler.
- The underlying infrastructure in SDN uses simple packet forwarding hardware as opposed to specialized hardware in conventional networks.

Key elements of SDN:

1) **Network Controller:** With decoupled control and data planes and centralized network controller, the network administrators can rapidly configure the network.
2) **Programmable Open APIs:** SDN architecture supports programmable open APIs for interface between the SDN application and control layers (Northbound interface).
3) **Standard Communication Interface (OpenFlow):** SDN architecture uses a standard communication interface between the control and infrastructure layers (Southbound interface). OpenFlow, which is defined by the Open Networking Foundation (ONF) is the broadly accepted SDN protocol for the Southbound interface.


Application Layer
Network Application
Network Application
Northbound Interface
Control Layer (Control Plane)
Network Services
Southbound Interface (OpenFlow, ForCES, etc.)
Infrasturcture Layer (Data Plane)

SDN FOR IOT

- Software-Defined Networking (SDN) is networking architecture that separates the control plane from the data plane and centralizes the network controller.
- Software-based SDN controllers maintain a unified view of the network.
- The underlying infrastructure in SDN uses simple packet forwarding hardware as opposed to specialized hardware in conventional networks.

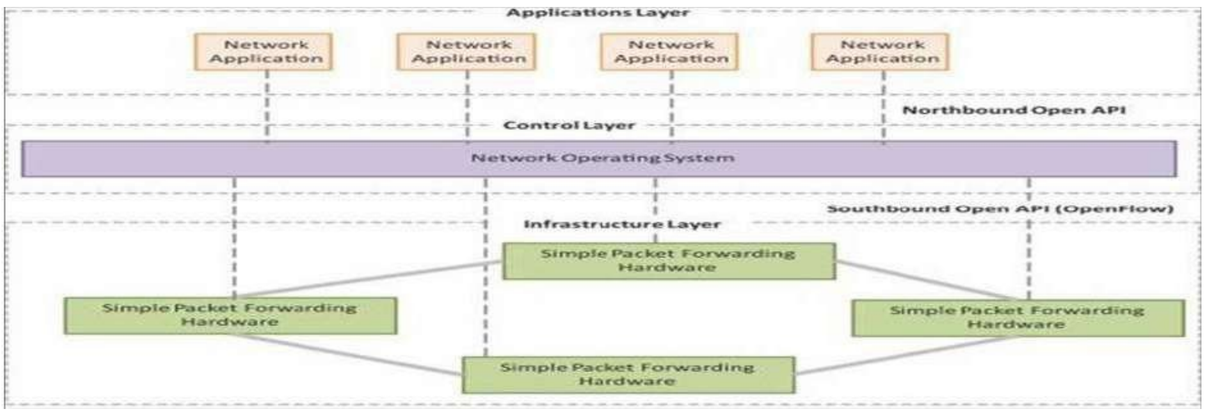


Fig: SDN Architecture

Key elements of SDN:

4) **Centralized Network Controller:** With decoupled control and data planes and centralized network controller, the network administrators can rapidly configure the network.
5) **Programmable Open APIs:** SDN architecture supports programmable open APIs for interface between the SDN application and control layers (Northbound interface).
6) **Standard Communication Interface (OpenFlow):** SDN architecture uses a standard communication interface between the control and infrastructure layers (Southbound interface). OpenFlow, which is defined by the Open Networking Foundation (ONF) is the broadly accepted SDN protocol for the Southbound interface.

*****

UNIT – V

CLOUD COMPUTING

Introduction to Cloud Computing:

The Internet of Things (IoT) involves the internet-connected devices we use to perform the processes and services that support our way of life. Another component set to help IoT succeed is cloud computing, which acts as a sort of front end. Cloud computing is an increasingly popular service that offers several advantages to IOT, and is based on the concept of allowing users to perform normal computing tasks using services delivered entirely over the internet. A worker may need to finish a major project that must be submitted to a manager, but perhaps they encounter problems with memory or space constraints on their computing device. Memory and space constraints can be minimized if an application is instead hosted on the internet. The worker can use a cloud computing service to finish their work because the data is managed remotely by a server.

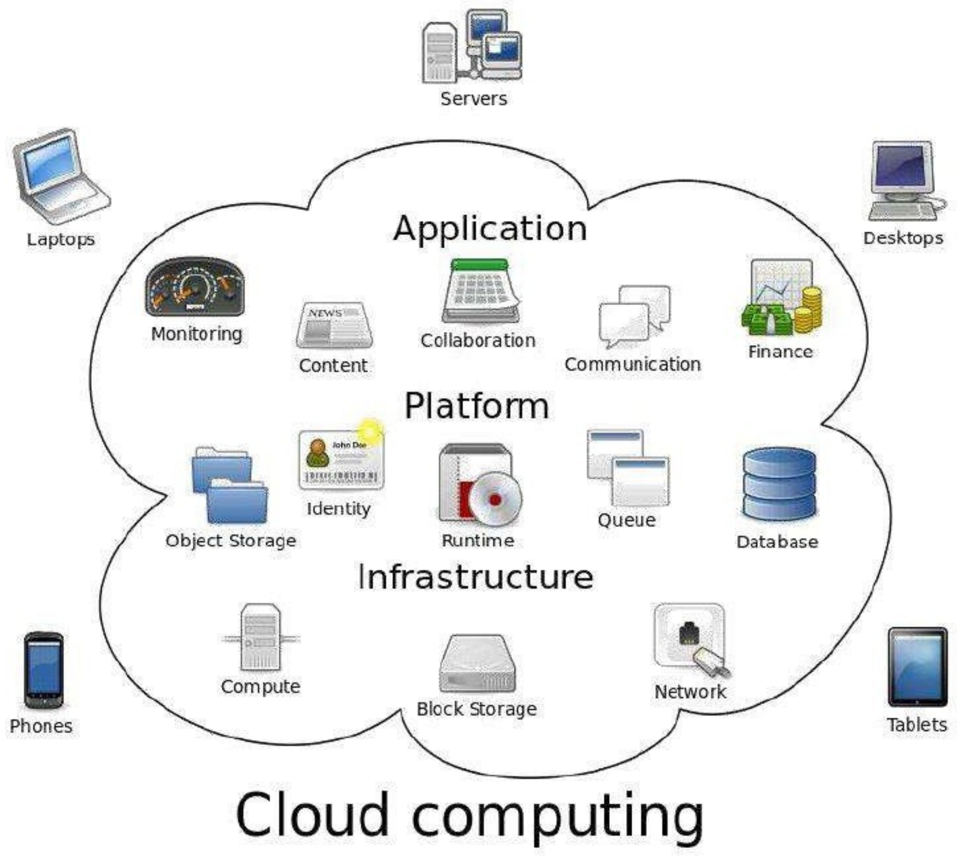
Another example: you have a problem with your mobile device and you need to reformat it or reinstall the operating system. You can use Google Photos to upload your photos to internet based storage. After the reformat or reinstall, you can then either move the photos back to you device or you can view the photos on your device from the internet when you want.

Concept:

In truth, cloud computing and IoT are tightly coupled. The growth of IoT and the rapid development of associated technologies create a widespread connection of things. This has lead to the production of large amounts of data, which needs to be stored, processed and accessed.

Cloud computing as a paradigm for big data storage and analytics. While IoT is exciting on its own, the real innovation will come from combining it with cloud computing. The combination of cloud computing and IoT will enable new monitoring services and powerful processing of sensory data streams. For example, sensory data can be uploaded and stored with cloud computing, later to be used intelligently for smart monitoring and actuation with other smart devices. Ultimately, the goal is to be able to transform data to insight and drive productive, cost-effective action from those insights. The cloud effectively serves as the brain to improved decision-making and optimized internet-based interactions. However, when IoT meets cloud, new challenges arise.

There is an urgent need for novel network architectures that seamlessly integrate them. The critical concerns during integration are quality of service (QoS) and quality of experience (QoE), as well as data security, privacy and reliability. The virtual infrastructure for practical mobile computing and interfacing includes integrating applications, storage devices, monitoring devices, visualization platforms, analytics tools and client delivery. Cloud computing offers a practical utility-based model that will enable businesses and users to access applications on demand anytime and from anywhere.



Characteristics:

First, the cloud computing of IoT is an on-demand self service, meaning its there when you need it. Cloud computing is a web-based service that can be accessed without any special assistance or permission from other people; however, you need at minimum some sort of internet access.

Second, the cloud computing of IoT involves broad network access, meaning it offers several connectivity options. Cloud computing resources can be accessed through a wide variety of internet-connected devices such as tablets, mobile devices and laptops. This level of convenience means users can access those resources in a wide variety of manners, even from older devices. Again, though, this emphasizes the need for network access points.

Third, cloud computing allows for resource pooling, meaning information can be shared with those who know where and how (have permission) to access the resource, anytime and anywhere. This lends to broader collaboration or closer connections with other users. From an IoT perspective, just as we can easily assign an IP address to every "thing" on the planet, we can share the "address" of the cloud-based protected and stored information with others and pool resources.

Fourth, cloud computing features rapid elasticity, meaning users can readily scale the service to their needs. You can easily and quickly edit your software setup, add or remove users, increase storage space, etc. This characteristic will further empower IoT by providing elastic computing power, storage and networking.

Finally, the cloud computing of IoT is a measured service, meaning you get what you pay for. Providers can easily measure usage statistics such as storage, processing, bandwidth and active user accounts inside your cloud instance. This pay per use (PPU) model means your costs scale with your usage. In IoT terms, it's comparable to the ever-growing network of physical objects that feature an IP address for internet connectivity, and the communication that occurs between these objects and other internet-enabled devices and systems; just like your cloud service, the service rates for that IoT infrastructure may also scale with use.

Service and Deployment Service

models:

Service delivery in cloud computing comprises three different service models: software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS).

Software as a service (SaaS) provides applications to the cloud ‘s end user that are mainly accessed via a web portal or service-oriented architecture-based web service technology. These services can be seen as ASP (application service provider) on the application layer. Usually, a specific company that uses the service would run, maintain and give support so that it can be reliably used over a long period of time.

Platform as a service (PaaS) consists of the actual environment for developing and provisioning cloud applications. The main users of this layer are developers that want to develop and run a cloud application for a particular purpose. A proprietary language was supported and provided by the platform (a set of important basic services) to ease communication, monitoring, billing and other aspects such as startup as well as to ensure an application ‘s scalability and flexibility. Limitations regarding the programming languages supported, the programming model, the ability to access resources, and the long-term persistence are possible disadvantages.

Infrastructure as a service (IaaS) provides the necessary hardware and software upon which a customer can build a customized computing environment. Computing resources, data storage resources and the communications channel are linked together with these essential IT resources to ensure the stability of applications being used on the cloud. Those stack models can be referred to as the medium for IoT, being used and conveyed by the users in different methods for the greatest chance of interoperability. This includes connecting cars, wearables, TVs, smartphones, fitness equipment, robots, ATMs, and vending machines as well as the vertical applications, security and professional services, and analytics platforms that come with them.

Deployment models:

Deployment in cloud computing comprises four deployment models: private cloud, public cloud, community cloud and hybrid cloud.

A private cloud has infrastructure that's provisioned for exclusive use by a single organization comprising multiple consumers such as business units. It may be owned, managed and operated by the organization, a third party or some combination of them, and it may exist on or off premises.

A public cloud is created for open use by the general public. Public cloud sells services to anyone on the internet. (Amazon Web Services is an example of a large public cloud provider.) This model is suitable for business requirements that require management of load spikes and the applications used by the business, activities that would otherwise require greater investment in infrastructure for the business. As such, public cloud also helps reduce capital expenditure and bring down operational IT costs.

A community cloud is managed and used by a particular group or organizations that have shared interests, such as specific security requirements or a common mission.

Finally, a hybrid cloud combines two or more distinct private, community or public cloud infrastructures such that they remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability. Normally, information that's not critical is outsourced to the public cloud, while business-critical services and data are kept within the control of the organization.

1) **Infrastructure-as-a-Service (IaaS) IoT/Clouds**: These services provide the means for accessing sensors and actuator in the cloud. The associated business model involves the IoT/Cloud provide to act either as data or sensor provider. IaaS services for IoT provide access control to resources as a prerequisite for the offering of related pay-as-you-go services.

2) **Platform-as-a-Service (PaaS) IoT/Clouds**: This is the most widespread model for IoT/cloud services, given that it is the model provided by all public IoT/cloud infrastructures outlined above. As already illustrate most public IoT clouds come with a range of tools and related environments for applications development and deployment in a cloud environment. A main characteristic of PaaS IoT services is that they provide access to data, not to hardware. This is a clear differentiator comparing to IaaS.
3) **Software-as-a-Service (SaaS) IoT/Clouds**: SaaS IoT services are the ones enabling their uses to access complete IoT-based software applications through the cloud, on-demand and in a pay-as-you-go fashion. As soon as sensors and IoT devices are not visible, SaaS IoT applications resemble very much conventional cloud-based SaaS applications. There are however cases where the IoT dimension is strong and evident, such as applications involving selection of sensors and combination of data from the selected sensors in an integrated application. Several of these applications are commonly called Sensing- as-a-Service, given that they provide on-demand access to the services of multiple sensors. Note that SaaS IoT applications are typically built over a PaaS infrastructure and enable utility based business models involving IoT software and services.

IOT IN SMART HOME AND SMART CITY APPLICATION

Implementing IoT system in home and city leads them to become as smart home and smart city. Smart home or smart city make life quite easier and smarter.

A smart home system can be something that makes our life quite easy. Starting from energy management where the power controls system in the AC appliances where we use the thermostat, all this is managed to cut down the power consumption that's taking place. A door management system, security management system, water management system are the part of this as well. Still, these are vital things that stand out in the smart home system.

The limitation of IoT in smart home application stops where our imagination stops. Anything that we wish to automate or want to make our life easier can be a part of smart home, a smartphone system as well.

Smart Home
Smart Air Conditioner
Lighting Control System
Energy Management
Appliance Control
Smart Thermostat
Security System
Smart Bathroom Appliance
Smart Door Lock
Garage Door Opener

CONNECTED VEHICLES

Connectivity will be at the heart of next generation vehicles. Whether it will be real-time traffic flow information, mapping, infotainment or remote access to emergency services, all these services will require connectivity.

Connected vehicle applications and services have distinctive features; they need to operate globally and usually have a very long 'device' lifetime, however can be integrated with local intelligent transport solutions and need to comply with local security and emergency regulations.

Connected vehicle and smart transport applications have the potential to bring substantial benefits to consumers, including making travel safer, reducing congestion, and providing real time information to passengers.

The GSMA is working with mobile operators and automotive OEMs to align the industry and wider ecosystem around a common approach to security and network connectivity to accelerate the growth of the Connected Vehicle market.

Governments can help encourage the development of the connected vehicle and intelligent transport ecosystems by:

- Introducing incentives for innovation and investment.

- Leading with light-touch regulation that will allow the market to scale while building trust and confidence of consumers.
- Promoting research and development programmers for connected and autonomous vehicles.
- Supporting services, applications and network industry-led standards and interoperability.

SMART GRID

Smart Grid is conceptualized as a combination of electrical network and communication infrastructure. With the implementation of bidirectional communication and power flows, a smart grid is capable of delivering electricity more efficiently and reliably than the traditional power grid. A smart grid consists of a power network with ‘intelligent’ entities that can operate, communicate, and interact autonomously, in order to efficiently deliver electricity to the customers. This heterogeneity in architecture of a smart grid motivates the use of advanced technology for overcoming various technical challenges at different levels. Any smart grid infrastructure should support real-time, two-way communication between utilities and consumers, and should allow software systems at both the producer and consumer ends to control and manage the power usage.

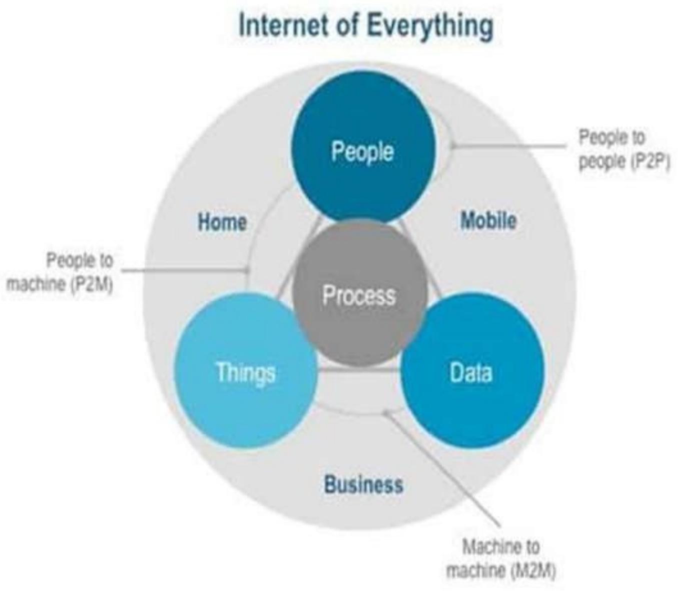
We study the impact on smart grid from different perspectives: energy management and pricing mechanism.

Cloud Architecture and Energy Management in Smart Grid:

Future smart grids are expected to have reliable, efficient, secured, and cost-effective power management with the implementation of distributed architecture. To focus on these requirements, we provide a comprehensive survey on different cloud computing applications for the smart grid architecture, in three different areas — energy management, information management and security.

<u>INDUSTRIAL IOT</u>

The industrial internet of things (IIoT) refers to the extension and use of the internet of things (IoT) in industrial sectors and applications. ... The IIoT encompasses industrial applications, including robotics, medical devices, and software-defined production processes.



*****

Prepared By:

RIYAZ

MOHAMMED