V: BUILDING IoT WITH ARDUINO &RASPBERRY PI

Building IOT with Arduino- Building IOT with RASPERRY PI- IoT Systems - Logical Design using Python – IoT Physical Devices & Endpoints - IoT Device -Building blocks - Pi - Raspberry Pi Interfaces - **Case study:**Smart Home & Smart Industry.

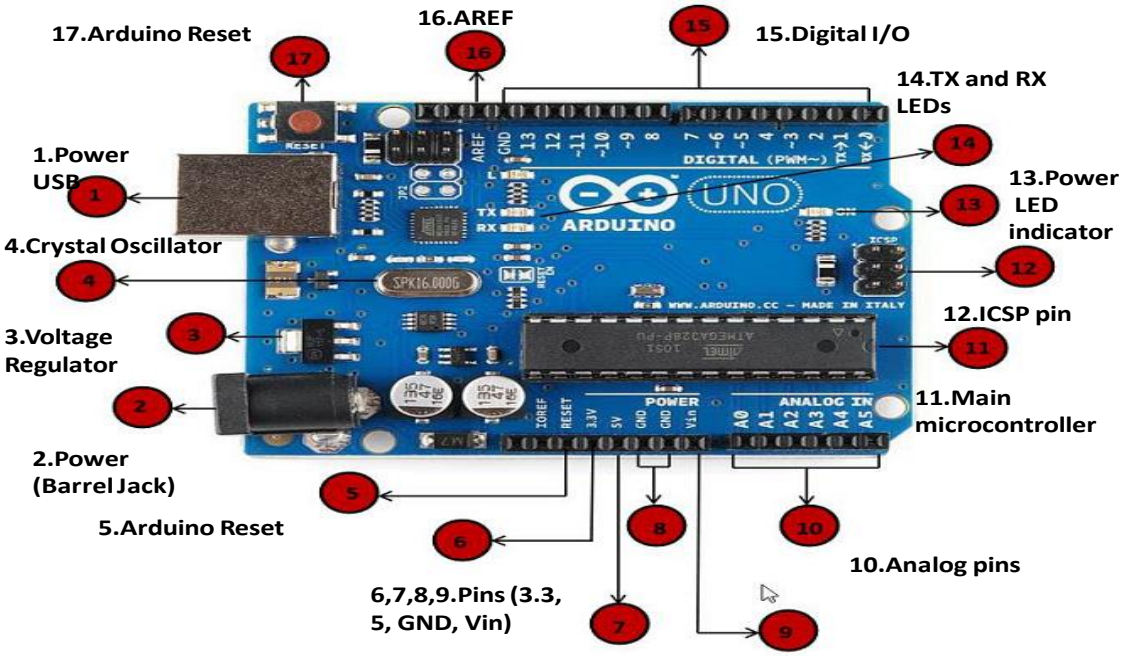
5.1 Building IOT with Arduino

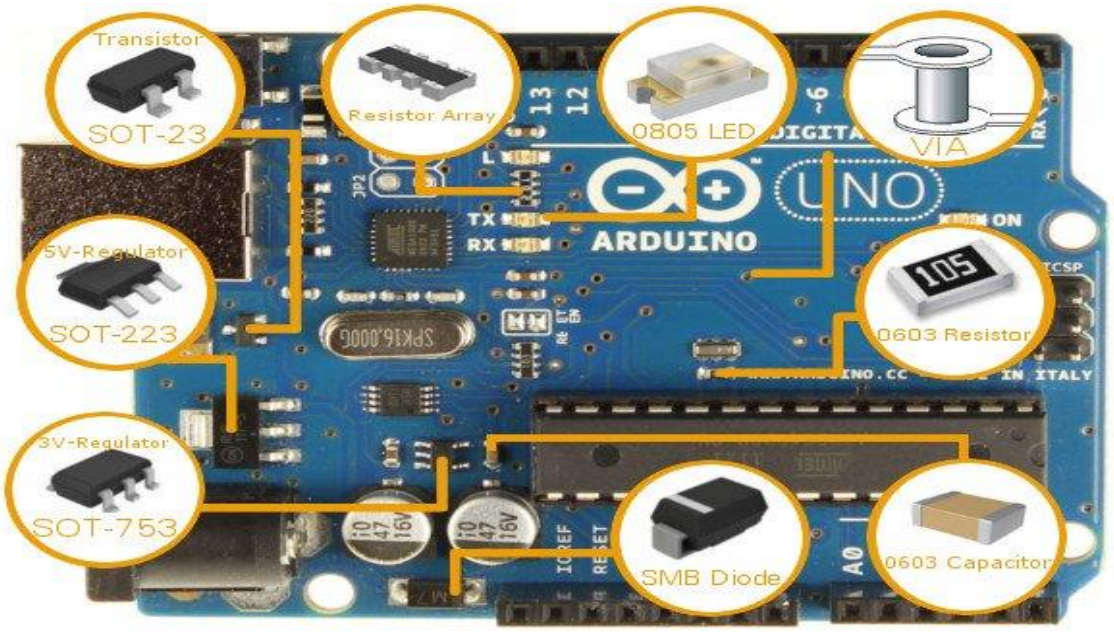
Internet of Things

The Internet of Things (IoT) is a scenario in which objects, animals or people are provided with single identifiers and the capability to automatically transfer and the capability to automatically transfer data more to a network without requiring human-to-human or human-to-computer communication.

Arduino Board:

- An Arduino is actually a microcontroller based kit.
- It is basically used in communications and in controlling or operating many devices.
- Arduino UNO board is the most popular board in the Arduino board family.
- In addition, it is the best board to get started with electronics and coding.
- Some boards look a bit different from the one given below, but most Arduino's have majority of these components in common.
- It consists of two memories- Program memory and the data memory.
- The code is stored in the flash program memory, whereas the data is stored in the data memory.
- Arduino Uno consists of 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button





1.Power USB

Arduino board can be powered by using the USB cable from your computer. All you need to do is connect the USB cable to the USB connection (1).

2.Power (Barrel Jack)

Arduino boards can be powered directly from the AC mains power supply by connecting it to the Barrel Jack (2).

3.Voltage Regulator

The function of the voltage regulator is to control the voltage given to the Arduino board and stabilize the DC voltages used by the processor and other elements.

4.Crystal Oscillator

The crystal oscillator helps Arduino in dealing with time issues. How does Arduino calculate time? The answer is, by using the crystal oscillator. The number printed on top of the Arduino crystal is 16.000H9H. It tells us that the frequency is 16,000,000 Hertz or 16 MHz.

5,17.Arduino Reset

You can reset your Arduino board, i.e., start your program from the beginning. You can reset the UNO board in two ways. First, by using the reset button (17) on the board. Second, you can connect an external reset button to the Arduino pin labelled RESET (5).

6,7,8,9.Pins (3.3, 5, GND, Vin)

- 3.3V (6) − Supply 3.3 output volt
- 5V (7) − Supply 5 output volt
- Most of the components used with Arduino board works fine with 3.3 volt and 5 volt.
- GND (8)(Ground) − There are several GND pins on the Arduino, any of which can be used to ground your circuit.
- Vin (9) − This pin also can be used to power the Arduino board from an external power source, like AC mains power supply.

10.Analog pins

The Arduino UNO board has six analog input pins A0 through A5. These pins can read the signal from an analog sensor like the humidity sensor or temperature sensor and convert it into a digital value that can be read by the microprocessor.

11.Main microcontroller

Each Arduino board has its own microcontroller (11). You can assume it as the brain of your board. The main IC (integrated circuit) on the Arduino is slightly different from board to board. The microcontrollers are usually of the ATMEL Company. You must know what IC your board has before loading up a new program from the Arduino IDE. This information is available on the top of the IC. For more details about the IC construction and functions, you can refer to the data sheet.

12.ICSP pin

Mostly, ICSP (12) is an AVR, a tiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, and GND. It is often referred to as an SPI (Serial Peripheral Interface), which could be considered as an "expansion" of the output. Actually, you are slaving the output device to the master of the SPI bus.

13.Power LED indicator

This LED should light up when you plug your Arduino into a power source to indicate that your board is powered up correctly. If this light does not turn on, then there is something wrong with the connection.

14.TX and RX LEDs

On your board, you will find two labels: TX (transmit) and RX (receive). They appear in two places on the Arduino UNO board. First, at the digital pins 0 and 1, to indicate the pins responsible for serial communication. Second, the TX and RX led (13). The TX led flashes with different speed while sending the serial data. The speed of flashing depends on the baud rate used by the board. RX flashes during the receiving process.

15.Digital I/O

- The Arduino UNO board has 14 digital I/O pins (15) (of which 6 provide PWM (Pulse Width Modulation) output. These pins can be configured to work as input digital pins to read logic values (0 or 1) or as digital output pins to drive different modules like LEDs, relays, etc. The pins labeled “~” can be used to generate PWM.

16.AREF

- AREF stands for Analog Reference. It is sometimes, used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

Program an Arduino

- The most important advantage with Arduino is the programs can be directly loaded to the device without requiring any hardware programmer to burn the program.
- This is done because of the presence of the 0.5KB of Bootloader which allows the program to be burned into the circuit.
- All we have to do is to download the Arduino software and writing the code.
- The Arduino tool window consists of the toolbar with the buttons like verify, upload, new, open, save, serial monitor.
- It also consists of a text editor to write the code, a message area which displays the feedback like showing the errors, the text console which displays the output and a series of menus like the File, Edit, Tools menu.

Steps to program an Arduino

- Programs written in Arduino are known as sketches. A basic sketch consists of 3 parts

  1. Declaration of Variables
  2. Initialization: It is written in the setup () function.
  3. Control code: It is written in the loop () function.

- The sketch is saved with .ino extension. Any operations like verifying, opening a sketch, saving a sketch can be done using the buttons on the toolbar or using the tool menu.
- The sketch should be stored in the sketchbook directory.
- Chose the proper board from the tools menu and the serial port numbers.
- Click on the upload button or chose upload from the tools menu. Thus the code is uploaded by the bootloader onto the microcontroller.

Basic Adruino functions are:

- **digitalRead**(pin): Reads the digital value at the given pin.
- **digitalWrite**(pin, value): Writes the digital value to the given pin.
- **pinMode**(pin, mode): Sets the pin to input or output mode.
- **analogRead**(pin): Reads and returns the value.
- **analogWrite**(pin, value): Writes the value to that pin.

- **serial.begin**(baud rate): Sets the beginning of serial communication by setting the bit rate.

Design your own Arduino

- The following components are needed to design Arduino Board- A breadboard, a led, a power jack, a IC socket, a microcontroller, few resistors, 2 regulators, 2 capacitors.
    - The IC socket and the power jack are mounted on the board.
    - Add the 5v and 3.3v regulator circuits using the combinations of regulators and capacitors.
    - Add proper power connections to the microcontroller pins.
    - Connect the reset pin of the IC socket to a 10K resistor.
    - Connect the crystal oscillators to pins 9 and 10
    - Connect the led to the appropriate pin.
    - Mount the female headers onto the board and connect them to the respective pins on the chip.
    - Mount the row of 6 male headers, which can be used as an alternative to upload programs.
    - Upload the program on the Microcontroller of the readymade Adruino and then pry it off and place back on the user kit.

Advantages of Arduino Board

1. It is inexpensive
2. It comes with an open source hardware feature which enables users to develop their own kit using already available one as a reference source.
3. The Arduino software is compatible with all types of operating systems like Windows, Linux, and Macintosh etc.
4. It also comes with open source software feature which enables experienced software developers to use the Arduino code to merge with the existing programming language libraries and can be extended and modified.
5. It is easy to use for beginners.

6. We can develop an Arduino based project which can be completely stand alone or projects which involve direct communication with the software loaded in the computer.
7. It comes with an easy provision of connecting with the CPU of the computer using serial communication over USB as it contains built in power and reset circuitry.

Interfaces

UART Peripheral:

- A UART (Universal Asynchronous Receiver/Transmitter) is a serial interface.
- It has only one UART module.
- The pins (RX, TX) of the UART are connected to a USB-to-UART converter circuit and also connected to pin0 and pin1 in the digital header.

SPI Peripheral:

- The SPI (Serial Peripheral Interface) is another serial interface. It has only one SPI module.

TWI:

- The I2C or Two Wire Interface is an interface consisting of only two wires, serial data, and a serial clock: SDA, SCL.
- You can reach these pins from the last two pins in the digital header or pin4 and pin5 in the analog header.

5.2 Building IOT with RASPERRY PI

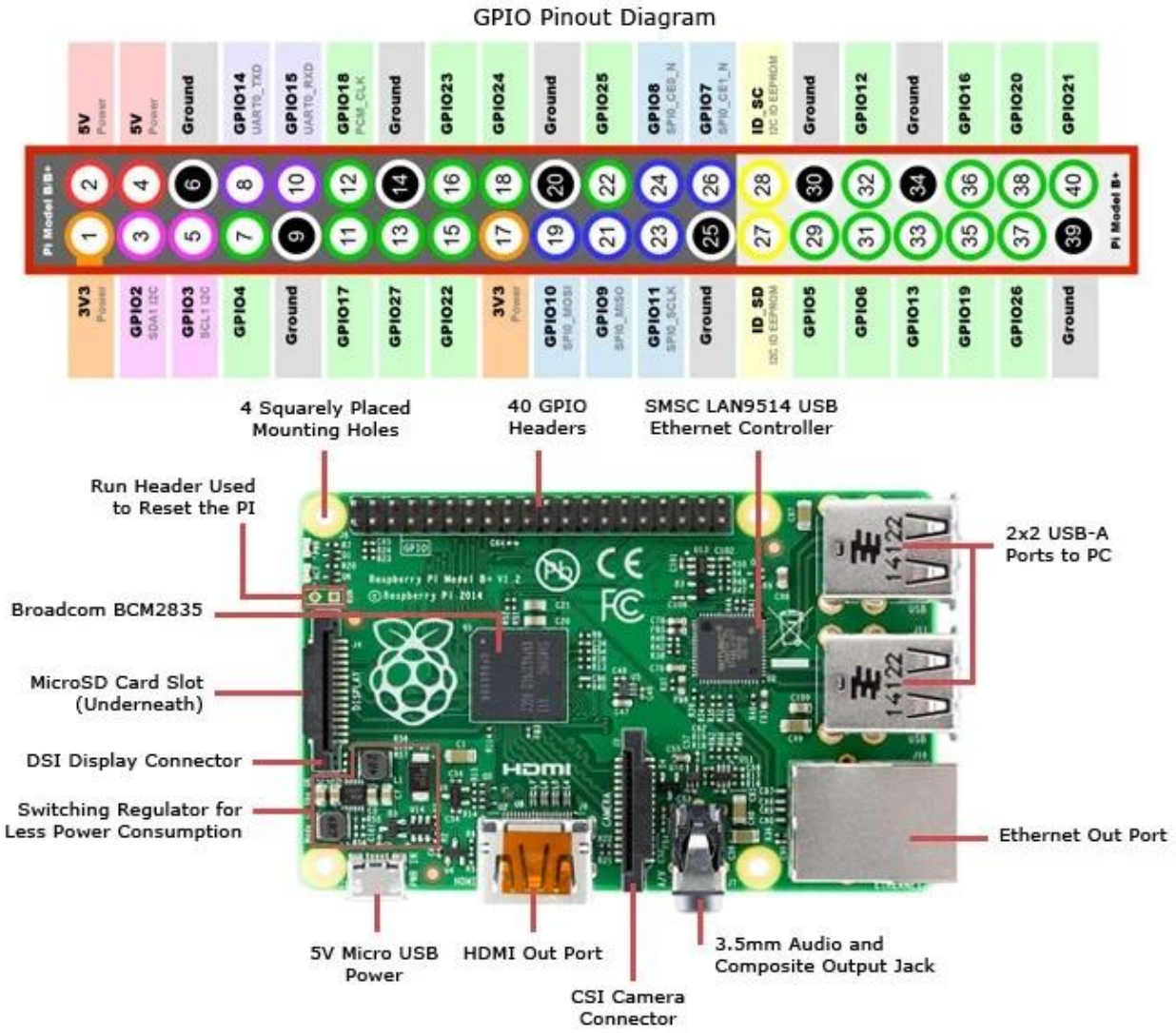
Internet of Things

The Internet of Things (IoT) is a scenario in which objects, animals or people are provided with single identifiers and the capability to automatically transfer and the capability to automatically transfer data more to a network without requiring human-to-human or human-to-computer communication.

Raspberry Pi

- The Raspberry Pi is a very cheap computer that runs Linux, but it also provides a set of GPIO (general purpose input/output) pins that allow you to control electronic components for physical computing and explore the Internet of Things (IoT).
- Raspberry Pi was basically introduced in 2006.
- It is particularly designed for educational use and intended for Python.
- A Raspberry Pi is of small size i.e., of a credit card sized single board computer, which is developed in the United Kingdom(U.K) by a foundation called Raspberry Pi.

- There have been three generations of Raspberry Pis: Pi 1, Pi 2, and Pi 3
- The first generation of Raspberry (Pi 1) was released in the year 2012, that has two types of models namely model A and model B.
- Raspberry Pi can be plugged into a TV, computer monitor, and it uses a standard keyboard and mouse.
- It is user friendly as can be handled by all the age groups.
- It does everything you would expect a desktop computer to do like word-processing, browsing the internet spreadsheets, playing games to playing high definition videos.
- All models feature on a broadcom system on a chip (SOC), which includes chip graphics processing unit GPU(a Video Core IV), an ARM compatible and CPU.
- The CPU speed ranges from 700 MHz to 1.2 GHz for the Pi 3 and on board memory range from 256 MB to 1 GB RAM.
- An operating system is stored in the secured digital SD cards and program memory in either the MicroSDHC or SDHC sizes.
- Most boards have one to four USB slots, composite video output, HDMI and a 3.5 mm phone jack for audio. Some models have WiFi and Bluetooth.
- Several generations of Raspberry Pis have been released.
- All models feature a Broadcom system on a chip (SoC) with an integrated ARM-compatible central processing unit (CPU) and on-chip graphics processing unit (GPU).
- Processor speed ranges from 700 MHz to 1.4 GHz for the Pi 3 Model B+ or 1.5 GHz for the Pi 4; on-board memory ranges from 256 MB to 1 GB with up to 4 GB available on the Pi 4 random-access memory (RAM).
- Secure Digital (SD) cards in MicroSDHC form factor (SDHC on early models) are used to store the operating system and program memory.
- The boards have one to five USB ports. For video output, HDMI and composite video are supported, with a standard 3.5 mm tip-ring-sleeve jack for audio output.
- Lower-level output is provided by a number of GPIO pins, which support common protocols like I²C. The B-models have an 8P8C Ethernet port and the Pi 3 and Pi Zero W have on-board Wi-Fi and Bluetooth.



Components and Peripherals

- **Voltages:** Two 5V pins and two 3V3 pins are present on the board, as well as a number of ground pins (0V). The remaining pins are all general purpose 3V3 pins
- A GPIO pin designated as an output pin can be set to high (3V3) or low (0V). A GPIO pin designated as an input pin can be read as high (3V3) or low (0V).
- **Processor & RAM:** Raspberry based on ARM11 processor. Latest version supports 700MHz processor and 512MB SDRAM. The Central processing unit is the brain of the raspberry pi board and that is responsible for carrying out the instructions of the computer through logical and mathematical operations.
- **Ethernet:** The Ethernet port of the raspberry pi is the main gateway for communicating with additional devices. The raspberry pi Ethernet port is used to plug your home router to access the internet.

- **USB Ports:** It has 2 USB ports. USB port provide current upto 100mA. For connecting devices that draw current more than 100mA, an external USB powered hub is required.
- **Ethernet Port:** It has standard RJ45 Ethernet port. Connect Ethernet cable or USB wifi adapter to provide internet connectivity.
- **HDMI Output:** It supports both audio and video output. Connect raspberry Pi to monitor using HDMI cable.
- **Composite video Output:** Raspberry comes with a composite video output with an RCA jack that supports both PAL and NTSC video output.
- **Audio Output:** It has 3.5mm audio output jack. This audio jack is used for providing audio output to old television along with RCA jack for video.
- **GPIO Pins:** It has a number of general purpose input/output pins. These pins are used to connect other electronic components. For example, you can connect it to the temperature sensor to transmit digital data.
- **Display Serial Interface (DSI):** DSI interface are used to connect an LCD panel to Raspberry PI.
- **Cameral Serial Interface(CSI):** CSI interface are used to connect a camera module to Raspberry PI.
- **SD Card slot:** Raspberry does not have built in OS and storage. Plug in an SD card loaded with Linux to SD card slot.
- **Power Input:** Raspberry has a micro USP connector for power input.
- **Memory:** The raspberry pi model A board is designed with 256MB of SDRAM and model B is designed with 51MB.Raspberry pi is a small size PC compare with other PCs. The normal PCs RAM memory is available in gigabytes. But in raspberry pi board, the RAM memory is available more than 256MB or 512MB
- **Status LEDs:** Raspberry has 5 status LEDs.

Status LED	Function
ACT	SD card Access
PWR	3.3V power is present
FDX	Full duplex LAN Connected
LNK	Link/Network Activity
100	100 Mbit LAN connected

Raspberry PI Interfaces:

- It supports SPI, serial and I2C interfaces for data transfer.
- **Serial :** Serial Interface on Raspberry has receive(Rx) and Transmit(Tx) pins for communication with serial peripherals.
- **SPI:** Serial Peripheral Interface (SPI) is a synchronous serial data protocol used for communicating with one or more peripheral devices. In an SPI connection, there is one master device and one or more peripheral devices. There are 5 pins Raspberry for SPI interface.
    - **MISO(Master In Slave Out):** Master line for sending data to the peripherals.
    - **MOSI(Master Out Slave In):** Slave Line for sending data to the master.
    - **SCK(Serial Clock):** Clock generated by master to synchronize data transmission.
    - **CE0(Chip Enable 0):** To enable or disable devices.

- **CE1(Chip Enable 1):** To enable or disable devices.

- **I2C:** I2C Interface pins are used to connect hardware modules. I2C interface allows synchronous data transfer with two pins: SDA(data line) and SCL (Clock Line)

Features of Raspberry PPI

1. Where the system processing is huge. They can process high end programs for applications like Weather Station, Cloud server, gaming console etc. With **1.2GHz clock speed** and **1 GB RAM RASPBERRY PI** can perform all those advanced functions.

2. RASPBERRY PI 3 has wireless LAN and Bluetooth facility by which you can setup WIFI HOTSPOT for internet connectivity.

3. RASPBERRY PI had dedicated port for connecting touch LCD display which is a feature that completely omits the need of monitor.

4. RASPBERRY PI also has dedicated camera port so one can connect camera without any hassle to the PI board.

5. RASPBERRY PI also has PWM outputs for application use.

6. It supports HD steaming

Applications

- ✓ Hobby projects.
- ✓ Low cost PC/tablet/laptop
- ✓ IoT applications
- ✓ Media center
- ✓ Robotics
- ✓ Industrial/Home automation
- ✓ Server/cloud server
- ✓ Print server
- ✓ Security monitoring
- ✓ Web camera
- ✓ Gaming
- ✓ Wireless access point

5.3 IoT Systems - Logical Design using Python

5.3.1 Introduction

Characteristics of Python:

- **Multi-paradigm programming language**: Python supports more than one programming paradigms including object oriented programming and structured programming.

- **Interpreted Language:** It is an interpreted language and does not require an explicit compilation step. Python interpreter executes the program source code directly and statement by statement interpreted.
- **Interactive Language:** user can submit commands at the python prompt and interact with the interpreter directly.
- **Easy to Learn and Use:** Python is easy to learn and use. It is developer-friendly and high level programming language.
- **Object and procedure oriented:** python supports both procedure and object oriented programming. It allows programs to be written around procedures or functions that allow reuse of code.
- **Extendable:** It is an extendable language and allows integration of low level modules written in languages such as C/C++.
- **Scalable:** It provides a manageable structure for large programs.
- **Portable:** Python programs are executed directly from source code and copy from one machine to other without worry about portability. Python interpreter converts source code to an intermediate form called byte code and then translate this into the native language of your specific system and then runs it.
- **Broad Library support:** It supports and works on different platforms such as windows, Linux, Mac etc. Large number of packages are available for various applications such as machine learning, image processing, network programming, cryptography etc.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Uses / Applications of Python

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

5.3.2 Installing Python

It is highly portable language that works on different platforms such as windows, Linux, Mac etc.

Windows

Python 2.7 can be downloaded from http://www.python.org/ftp/python/2.7.5/python-2.7.5.msi

Linux

Install python in Ubuntu Linux like:

#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
```

5.3.3 Python data types & data structures

5.3.3.1 Numbers

Number data type are used to store numeric values.

Numbers are immutable data types that is changing the value of a number data type

Int

- ✓ Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.
- ✓ Example

```
x = 1
y = 35656222554887711
z = -3255522
print(type(x))
print(type(y))
print(type(z))
```

- ✓ Output

```
<class 'int'>
<class 'int'>
<class 'int'>
```

Float

- ✓ Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

```
x = 1.10
y = 1.0
z = -35.59
print(type(x))
print(type(y))
print(type(z))
```

- ✓ Output

```
<class 'float'>
<class 'float'>
<class 'float'>
```

Complex

- ✓ Complex numbers are written with a "j" as the imaginary part:

```
x                                   = 3+5j
y                           =       5j
z                           =       -5j
print(type(x))
print(type(y))
print(type(z))
print("Real Part is :", x.real)
print("Imaginary Part is :",x.imag)
```

- ✓ Output

```
<class 'complex'>
<class 'complex'>
<class 'complex'>
Real Part is : 3.0
Imaginary Part is : 5.0
```

5.3.3.2 Strings

- ✓ String literals in python are surrounded by either single quotation marks, or double quotation marks.
- ✓ 'hello' is the same as "hello".
- ✓ One or more character is called string
- ✓ A string with zero character is called empty string.
- ✓ Example:

```
a="Hello"
b="World!"
print(type(a))
print(b)
#String concatination
print(a+b)
#String Length
print(len(a))
#Convert to upper and lower
print(a.lower())
print(a.upper())
# Substring 4th character not included
print(a[1])
print(a[1:4])
print(a[1:])
#strip() method removes any whitespace or characters
print(b.strip("Wor"))
print(b.strip("!"))
#The replace() method replaces a string with another string
```

```
print(a.replace("H","J"))
#Format Method
age = 36
txt = "My age is {}"
print(txt.format(age))
```

- ✓ Output

```
<class 'str'>
World!
HelloWorld!
5
hello
HELLO
e
ell
ello
ld!
World
Jello
My age is 36
```

5.3.3.3 List

- It is a Compound data type used to group together other values.
- List items need not have the same data type.
- **List** is a collection which is ordered and changeable. It allows duplicate members.
- To add an item to the end of the list, use the **append()** method
- To add an item at the specified index, use the **insert()** method
- The **remove()** method removes the specified item
- List items are separated by commas and enclosed with in **square brackets**.
- To access the list items by referring to the **index number**.
- To change the value of a specific item, refer to the **index number**
- **Program**

```
li=['apple','orange','mango','banana']
print(type(li))
#print all item in a list
print(li)
#print items in a list based on index start is 0
print(li[0])
print(li[1:])
#append() add an item in end of the list
li.append("Berry")
print(li)
#remove() delete an item
li.remove("apple")
print(li)
#Insert an item based on index
li.insert(2,"apple")
```

```
print(li)
#Crate mixed list
mi=['Hai',2,5.999]
print(mi)
print(li+mi)
```

5.3.3.4 Tuple

- **Tuple** is a collection which is ordered and unchangeable. It allows duplicate members.
- In Python tuples are written with round brackets.
- You can access tuple items by referring to the index number, inside square brackets.
- Once a tuple is created, you cannot change its values. Tuples are **unchangeable**.
- To determine how many items a tuple has, use the len() method
- You cannot add items to a tuple.
- Tuples are **unchangeable**, so you cannot remove items from it, but you can delete the tuple completely
- Example:

```
tu=('apple','orange','mango','banana')
print(type(tu))
#print all item in a list
print(tu)
#print items in a tuple based on index start at 0
print(tu[0])
print(tu[1:3])
print(tu[:2])
#len() length of the tuple
print(len(tu))

#create Mixed tuple
mi=('Hai',2,5.999)
print(tu+mi)
```

- Output

```
<class 'tuple'>
('apple', 'orange', 'mango', 'banana')
apple
('orange', 'mango')
('apple', 'orange')
4
('apple', 'orange', 'mango', 'banana', 'Hai', 2, 5.999)
```

5.3.3.5 Dictionaries

- A dictionary is a collection which is unordered, changeable and indexed.
- In Python dictionaries are written with curly brackets, and they have keys and values.
- You can access the items of a dictionary by referring to its key name,
- To determine how many items (key-value pairs) a dictionary has, use the len() method.
- Example

```
dict = {
  "brand": "Ford",
```

```
  "model": "Mustang",
  "year": 1964
}
print(type(dict))
print(dict)
#len() to determine length of dictionary
print(len(dict))
#print model value
x = dict["model"]
print(x)
# print all items
print(dict.items())
# get keys
print(dict.keys())
#get values
print(dict.values())
#add new key & value
dict["color"] = "red"
print(dict)
# last item deleted
dict.popitem()
print(dict)
del dict["model"]
print(dict)
```

- Output

```
<class 'dict'>
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
3
Mustang
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
dict_keys(['brand', 'model', 'year'])
dict_values(['Ford', 'Mustang', 1964])
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
{'brand': 'Ford', 'year': 1964}
```

5.3.3.6 Type Conversion

- ✓ convert from one type to another with the int(), float(), and complex() methods:
- ✓ Example

```
x = 1 # int
y = 2.8 # float
z = 1j # complex
#convert from int to float:
a = float(x)
#convert from float to int:
b = int(y)
#convert from int to complex:
```

```
c                                                    = complex(x)

print(b)
print(c)
print(type(a))
print(type(b))
print(type(c))
s="aeiou"
print(list(s))
```

- Output

```
1.0
2
(1+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
['a', 'e', 'i', 'o', 'u']
```

5.3.4 Control Flow

5.3.4.1 Control Statements

if

- ✓ An "if statement" is written by using the if keyword.
- ✓ The **if** statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.
- ✓ Syntax

```
if expression:
  statement(s)
```

- ✓
- ✓ Example

```
a=int(input('Enter a Number'))
if a>0:
   print("It is Positive Number")
```

- ✓ Output

```
Enter a Number78
It is Positive Number
```

elif and else

- ✓ The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".
- ✓ An **else** statement can be combined with an **if** statement.
- ✓ An **else** statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

- ✓ Example
- ✓ syntax

```
if expression1:
  statement(s)
elif expression2:
  statement(s)
elif expression3:
  statement(s)
else:
  statement(s)
```

- ✓ Example:

```
a=int(input('Enter a Number'))
if a>0:
   print(a," is Positive Number")
elif a<0:
   print(a," is Negative Number")
else:
   print(a," is Zero")
```

- ✓ Output

```
Enter a Number-67
-67  is Negative Number
```

Nested if

- ✓ In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.
- ✓ Syntax

```
if expression1:
   statement(s)
   if expression2:
      statement(s)
   elif expression3:
      statement(s)
   elif expression4:
      statement(s)
   else:
      statement(s)
 else:
   statement(s)
```

- ✓ Example

```
var = 100
if var < 200:
```

```
  if var == 150:
    print ("Which is 150")
  elif var == 100:
    print ("Which is 100")
  elif var == 50:
    print ("Which is 50")
  elif var < 50:
    print ("Value is less than 50")
else:
  print ("Values more than 200..")
```

For

- ✓ The for **loop in Python** is used to iterate the statements or a part of the program several times.
- ✓ It is frequently used to traverse the data structures like list, tuple, or dictionary.
- ✓ syntax

```
for iterating_var in sequence:
  statement(s)
```

- ✓ Example

```
#print natural numbers
i=1
n=int(input("Enter the number up to print the natural numbers : "))
for i in range(0,n):
   print(i,end=' ')

#print String
s="Hello World"
print("\n")
for c in s:
    print(c,end=' ')

#Multiplication Table
i=1;
print("\n")
num = int(input("Enter a number:"));
for i in range(1,11):
    print("%d X %d = %d"%(num,i,num*i));
```

- ✓ Output

```
Enter the number up to print the natural numbers : 2
0 1
```

```
H e l l o  W o r l d
Enter a number:4
4 X 1 = 4
4 X 2 = 8
4 X 3 = 12
4 X 4 = 16
4 X 5 = 20
4 X 6 = 24
4 X 7 = 28
4 X 8 = 32
4 X 9 = 36
4 X 10 = 40
```

while

- ✓ With the while loop we can execute a set of statements as long as a condition is true.
- ✓ Syntax

```
while expression:
  statement(s)
```

- ✓ **statement(s)** may be a single statement or a block of statements.
- ✓ The **condition** may be any expression, and true is any non-zero value.
- ✓ The loop iterates while the condition is true.
- ✓ Program

```
#print 1 to 5
i = 1
while i < 6:
   print(i)
   i=i+1

#print even number until 10
j=1
print("Even Numbers are")
while j<=10:
   if j%2==0:
      print(j)
   j=j+1
```

- ✓ Output:

```
1
2
3
4
5
```

```
Even Numbers are
2
4
6
8
10
```

range() Function

- ✓ To loop through a set of code a specified number of times, we can use the range() function,
- ✓ The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.
- ✓ The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, **3**):
- ✓ Program

```
for x in range(6):
    print(x)

for x in range(2, 6):
 print(x)

for x in range(2, 30, 3):
 print(x)
```

- ✓ Output

```
0 1 2 3 4 5
2 3 4 5
2 5 8 11 14 17 20 23 26 29
```

Break & Continue

- ✓ With the break statement we can stop the loop before it has looped through all the items.
- ✓ With the continue statement we can stop the current iteration of the loop, and continue with the next.
- ✓ Program

```
a=[1,2,3,4]
for x in a:
   if x==2:
      continue
   print(x)

for x in a:
   if x==3:
      break
```

```
        print(x)
```

- ✓ Output

```
1 3 4 1 2
```

Pass

- ✓ It is null operation
- ✓ It is used when a statement is required syntactically but not execute any code.
- ✓ Program

```
for x in a:
  if x==1:
    pass
  else:
    print(x)
```

- ✓ Output

```
2 3 4
```

5.3.5 Python Functions

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

Creating a Function

- In Python a function is defined using the **def** keyword.
- Function begins with def keyword followed by function name and parenthesis.
- Function parameters are enclosed within the parenthesis.

Calling a Function

- To call a function, use the function name followed by parenthesis:
- Information can be passed to functions as parameter.
- Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.
- Program

```
#function1
def fun1():
  print("Hello from a function")
fun1()

#function2
def fun2(fname):
  print("Hello "+ fname)
fun2("Cathy")
fun2("Raju")

#function3
def fun3(country = "Norway"):
  print("I am from " + country)

fun3("Sweden")
```

```
fun3("India")
fun3()
fun3("Brazil")

#function4
def fun4(food):
  for x in food:
    print(x)
fruits = ["apple", "banana", "cherry"]
fun4(fruits)

#function5
def fun5(x):
  return 5 * x

print(fun5(3))
print(fun5(5))
print(fun5(9))
```

- Output

```
Hello from a function
Hello Cathy
Hello Raju
I am from Sweden
I am from India
I am from Norway
I am from Brazil
apple
banana
cherry
15
25
45
```

5.3.6 Modules

- Consider a module to be the same as a code library.
- A file containing a set of functions you want to include in your application.
- It allows organizing of the program code into different modules which improves the code readability and management.
- A module is a python file that defines some functionality in the form of function or classes.
- Create a Module : To create a module just save the code you want in a file with the file extension .py:
- **filename: module1.py**

```
def f1(name):
```

```
print("Hello, " + name)
```

- Use a Module: Now we can use the module we just created, by using the import statement:

```
import module1
module1.f1("World!")
```

- Output : Hello, World!
- Example :

1. module3.py

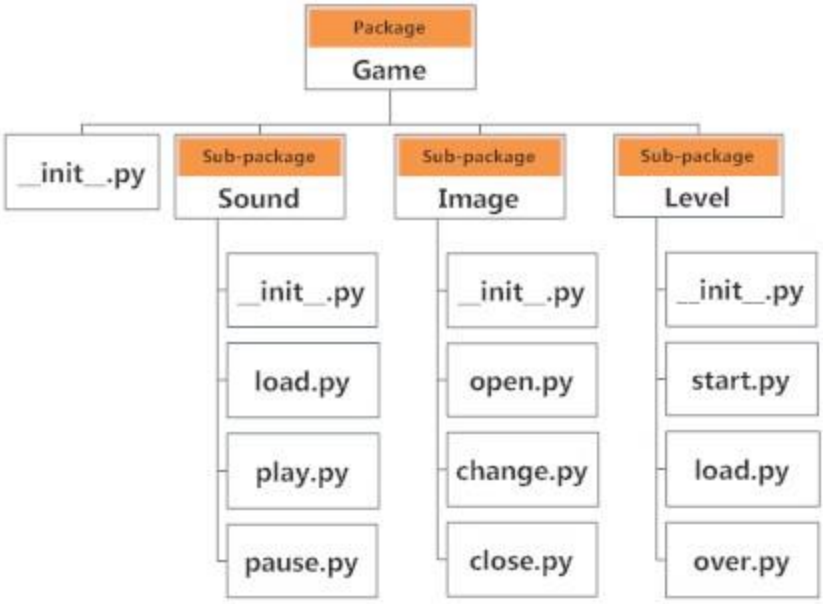
```
def add(x,y):
   print("Addition is :",x+y)
def sub(x,y):
   return(x-y)
```

2. module4.py

```
import module3
module3.add(10,5)
print("Subtraction is :",module3.sub(100,50))
```

5.3.7 Packages

- Python package is hierarchical file structure that consists of modules and sub packages.
- Packages are organized into root directory with sub directories
- Each directory contains a special file named __init__.py which tells python to treat directory as packages.
- Packages are a way of structuring many packages and modules which helps in a well-organized hierarchy of data set, making the directories and modules easy to access.
- To tell Python that a particular directory is a package, we create a file named __init__.py inside it and then it is considered as a package.
- We may create other modules and sub-packages within it.
- This __init__.py file can be left blank or can be coded with the initialization code for the package.
- **To create a package in Python, we need to follow these three simple steps:**
- First, we create a directory and give it a package name, preferably related to its operation.
- Then we put the classes and the required functions in it.
- Finally we create an __init__.py file inside the directory, to let Python know that the directory is a package.



- We can import modules from packages using the dot (.) operator.
- Example: import Game.Level.start

5.3.8 File Handling

- Python using reading and writing files to files using file object.
- open(filename,mode) is used to get a file object.
- Mode can be read(r), write(w),append(a),read and write(r+ or w+), read-binary(rb), writebinary(wb),create(x).
- close() used to close the file.
- read() method for reading the content of the file
- You can return one line by using the readline() method:
- Example1:

```
#read whole file
f = open("ruff.txt", "r")
print(f.read())
f.close()
#read line by line
f = open("ruff.txt", "r")
print(f.readline())
print(f.readline())
f.close()
#use loop to print
f = open("ruff.txt", "r")
for x in f:
  print(x)
```

- Output:

```
Java Programming
C Programming
C++ Programming
Python Programming
PHP Programming
HTML Programming
Java Programming
C Programming
Java Programming
C Programming
C++ Programming
Python Programming
PHP Programming
HTML Programming
```

- Example2:

```
#create new file
f = open("myfile.txt", "x")
#write data into file
f = open("myfile.txt", "a")
f.write("Now the file has more content!")
f.close()
#read data from file
f = open("myfile.txt", "r")
print(f.read())
f.close()
```

- Output:

```
Now the file has more content!
```

5.3.9 Date and Time Operations

- Calendar module in Python has the calendar class that allows the calculations for various task based on date, month, and year.
- The TextCalendar and HTMLCalendar class in Python allows you to edit the calendar and use as per your requirement.
- import a module named datetime to work with dates as date objects.
- Return the year and name of weekday:
- The method is called strftime(), and takes one parameter, format, to specify the format of the returned string.

Directive	Description	Example
%a	Weekday, short version	Wed

%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018

5.3.10 Classes

- Python is an object oriented programming language.
- Almost everything in Python is an object, with its properties and methods.
- A Class is like an object constructor, or a "blueprint" for creating objects.
- To create a class, use the keyword class:
- Python supports classes, class variables, class methods, inheritance, function overloading, operator overloading.
- **Class:** A class is simply a representation of a type of object and user defined prototype for an object that is composed of three things like name, attribute and operations/methods.
- **Object:** Object is an instance of the data structure defined by a class.
- **Inheritance:** It is the process of forming a new class from an existing class or base class.
- **Function overloading:** It is a form of polymorphism that allows a function to have different meanings, depending on its context.
- **Operator overloading:** It is a form of polymorphism that allows assignment of more than one function to a particular operator.
- **Function overriding:** It allows a child class to provide a specific implementation of a function that is already provided by the base class. Child class implementation of the overridden function has the same name, parameters and return type as the function in the base class.
- All classes have a function called __init__(), which is always executed when the class is being initiated.
- Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:
- The __init__() function is called automatically every time the class is being used to create a new object.

- The `self` parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.
- It does not have to be named `self` , you can call it whatever you like, but it has to be the first parameter of any function in the class:
- You can delete properties on objects by using the del keyword.
- Example1:

```
class Person:
  def __init__(self, name, age):
     print("Constructor Called..")
     self.name = name
     self.age = age
  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
print(p1.name)
print(p1.age)
p1.myfunc()
```

- Output1:

```
Constructor Called..
John
36
Hello my name is John
```

- Example2:

```
#Inhertitance Program
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of",
self.graduationyear)

x = Student("Mike", "Olsen", 2019)
x.welcome()
```

- Output2:

```
Welcome Mike Olsen to the class of 2019
```

5.3.11 Python Packages for IoT

5.3.11.1 JSON

- Javascript Object Notation(JSON) is an easy to read and write data-interchange format.
- It is an alternate to XML.
- It is built on 2 structure: collection of name-value pairs(dictionary) and ordered list of values(list)
- JSON format is used for serializing and transmitting structured data over a network connection.
- If you have a JSON string, you can parse it by using the json.loads() method.
- If you have a Python object, you can convert it into a JSON string by using the json.dumps() method
- Example:

```
import json

# some JSON:
x =  '{ "name":"John", "age":30, "city":"New York"}'

# parse x:
y = json.loads(x)

# the result is a Python dictionary:
print(y["age"])

# a Python object (dict)
x = {
  "name": "John",
  "age": 30,
  "city": "New York"
}

# convert into JSON:
y = json.dumps(x)

# the result is a JSON string:
print(y)
```

- Output:

```
30
{"name": "John", "age": 30, "city": "New York"}
```

5.3.11.2 XML

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

- It was designed to store and transport small to medium amounts of data and is widely used for sharing structured information.
- In order to parse an XML document using minidom, we must first import it from the xml.dom module. This module uses the parse function to create a DOM object from our XML file.
- getElementByTagName() to find a specific tag.
- Example:

```
from xml.dom import minidom

# parse an xml file by name
mydoc = minidom.parse('items.xml')

items = mydoc.getElementsByTagName('item')


# all item attributes
print('\nAll attributes:')
for elem in items:
   print(elem.attributes['name'].value)

# one specific item's data
print('\nItem #2 data:')
print(items[1].firstChild.data)


# all items data
print('\nAll item data:')
for elem in items:
   print(elem.firstChild.data)
```

- Output:

```
All attributes:
item1
item2

Item #2 data:
Mango

All item data:
Apple
Mango
```

5.11.3.3 HTTPlib and URLlib

- HTTPLib2 and URLLib2 are python libraries used in network/internet programming.
- HTTPLib2 is an HTTP client library and URLLib2 is a library for fetching URLs.
- import urllib.requests. From there, we assign the opening of the url to a variable, where we can finally use a .read() command to read the data.

- Example1:

```
import http.client
conn = http.client.HTTPSConnection("www.python.org")
conn.request("GET", "/")
r1 = conn.getresponse()
print(r1.status, r1.reason)
data1 = r1.read()  # This will return entire content.
# The following example demonstrates reading data in chunks.
conn.request("GET", "/")
r1 = conn.getresponse()
while not r1.closed:
    print(r1.read(200))
```

- Example2:

```
import urllib.request
x = urllib.request.urlopen('https://www.gmail.com/')
print(x.read())
```

5.11.3.4 SMTPLib

- Simple Mail Transfer Protocol (SMTP) is a protocol, which handles sending an e-mail and routing e-mail between mail servers.
- Python provides **smtplib** module, which defines an SMTP client session object that can be used to send mails to any Internet machine with an SMTP or ESMTP.
- Syntax:

```
import smtplib
smtpObj = smtplib.SMTP( [host [, port [, local_hostname]]] )
```

- Here is the detail of the parameters −
- **host** − This is the host running your SMTP server. You can specifiy IP address of the host or a domain name like tutorialspoint.com. This is an optional argument.
- **port** − If you are providing *host* argument, then you need to specify a port, where SMTP server is listening. Usually this port would be 25.
- **local_hostname** − If your SMTP server is running on your local machine, then you can specify just *localhost* the option.
- An SMTP object has an instance method called **sendmail**, which is typically used to do the work of mailing a message. It takes three parameters −
- The *sender* − A string with the address of the sender.
- The *receivers* − A list of strings, one for each recipient.
- The *message* − A message as a string formatted as specified in the various RFCs.
- Example:

```
import smtplib

sender = 'from@fromdomain.com'
receivers = ['to@todomain.com']

message = """From: From Person <from@fromdomain.com>
```

```
To: To Person <to@todomain.com>
Subject: SMTP e-mail test

This is a test e-mail message.
"""

try:
   smtpObj = smtplib.SMTP('localhost')
   smtpObj.sendmail(sender, receivers, message)
   print "Successfully sent email"
except SMTPException:
   print "Error: unable to send email"
```

5.4 IoT physical Devices & Endpoints

5.4.1 IoT Device

- Thing in Internet of Things(IoT) can be any object that has a unique identifier and which can send/receive data over a network.
- IoT devices are connected to the internet and send information about themselves or about their surroundings over a network or allow actuation upon the physical entities or environment around them remotely.
- Some example of IoT devices are:
  - A Home Automation device that allows remotely control and monitor the appliances.
  - An Industrial Machine which sends information about its operation and health monitoring data to a server.
  - A car sends information about its location to a cloud based service.
  - A wireless enabled wearable device that measures data about a person such as number of steps walked and send the data to a cloud based service.

5.4.2 Basic Building block of an IoT device

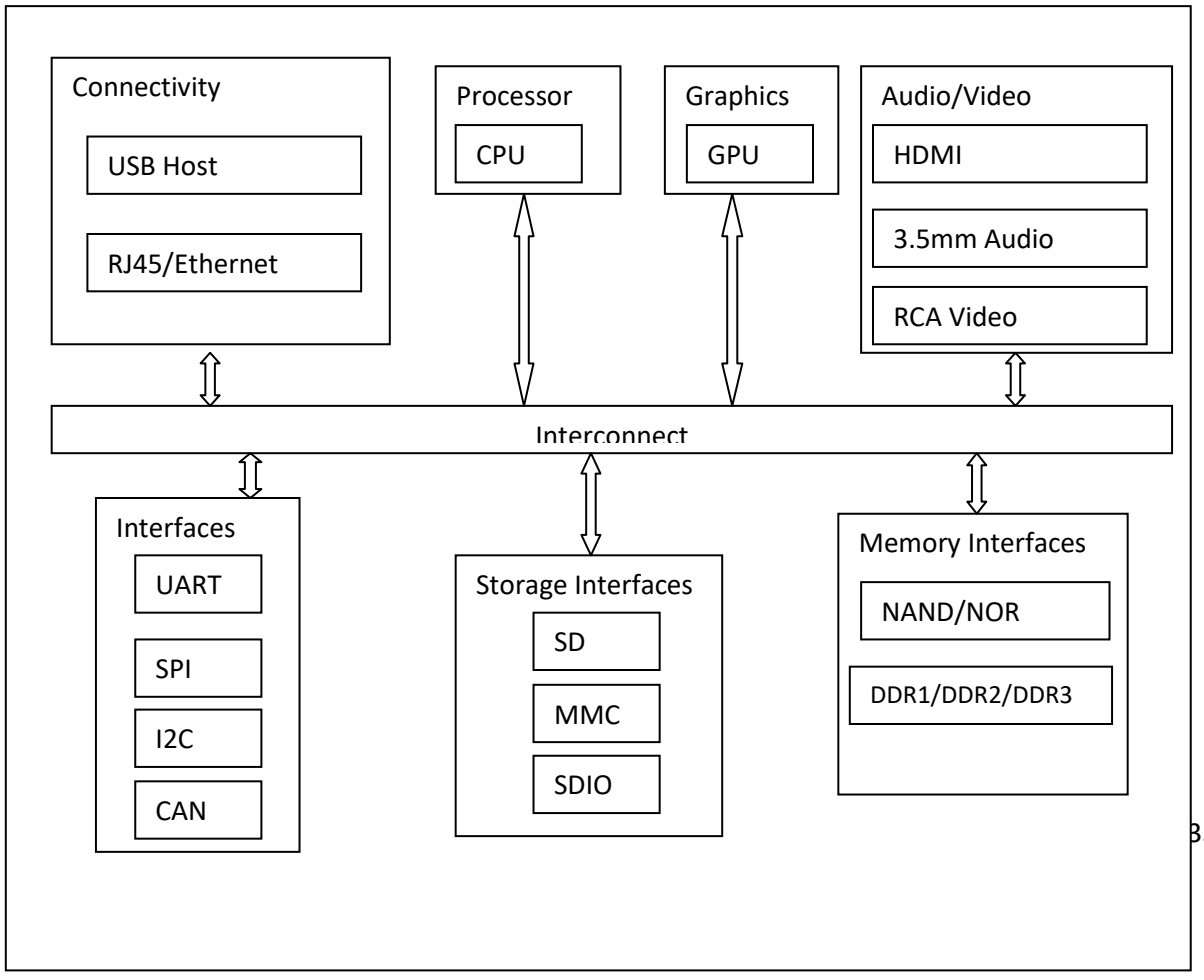
- IoT stands for "Internet of Things," which means using the Internet to connect different things. IoT is an intersection between the physical and virtual worlds. It essentially maps virtual operations onto real interactions.

IoT device consists of a number of modules based on functional attributes:

- **Sensors:** It can be either on board the IoT device or attached to the device. IoT device can collect various types of information from the onboard or attached sensors such as temperature, humidity, light intensity etc. The sensed information can be communicated either to other devices or cloud based server/storage. These form the front end of the IoT devices. These are the so called “Things” of the system. Their main purpose is to collect data from its surrounding (sensors) or give out data to its surrounding (actuators). These have to be uniquely identifiable devices with a unique IP address so that they can be

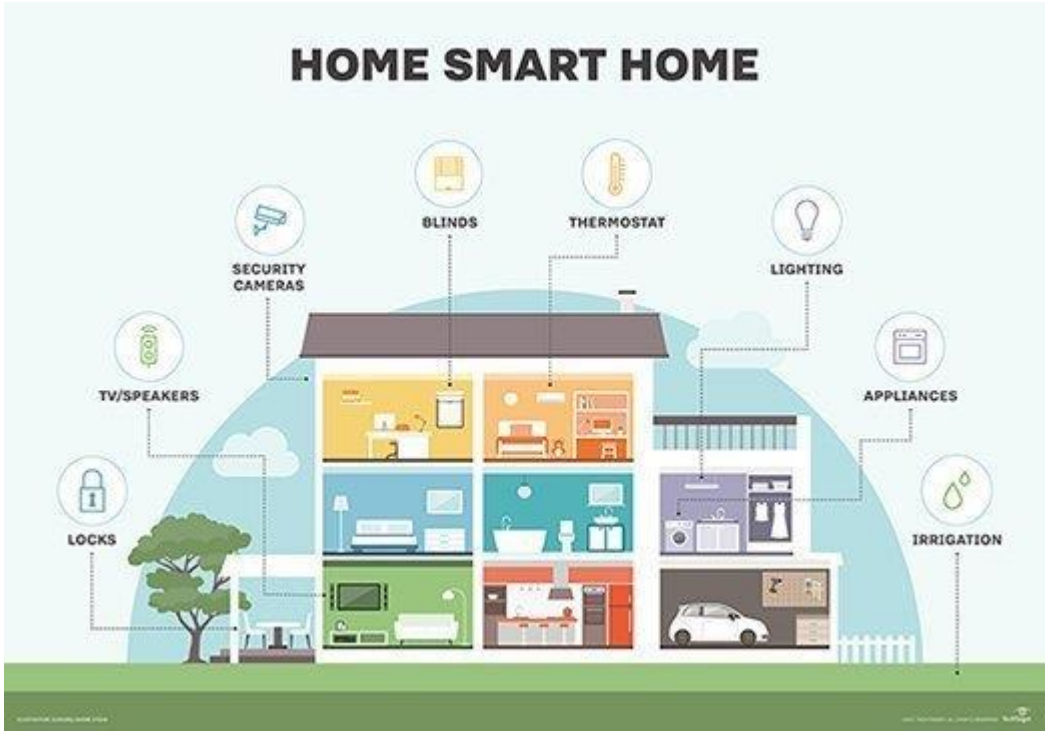
easily identifiable over a large network. It should be able to collect real time data. Examples of sensors are: gas sensor, water quality sensor, moisture sensor etc.

- **Actuation:** IoT devices can have various types of actuators attached that allow taking actions upon the physical entities in the vicinity of the device. Example relay switch connected to an IoT device can turn appliances on/off based on the commands send to the device.
- **Processors:** Processors are the brain of the IoT system. Their main function is to process the data captured by the sensors and process them so as to extract the valuable data from the enormous amount of raw data collected. It gives intelligence to the data. Processors mostly work on real-time basis and can be easily controlled by applications. These are also responsible for securing the data – that is performing encryption and decryption of data.
- **Communication:** It is responsible for sending collected data to other devices or cloud based servers/storage and receiving data from other devices and commands from remote applications. Gateways are responsible for routing the processed data and send it to proper locations for its (data) proper utilization. Gateway helps in to and fro communication of the data. It provides network connectivity to the data. Network connectivity is essential for any IoT system to communicate. LAN, WAN, PAN etc are examples of network gateways.
- **Analysis & Processing:** These are responsible for taking decision based upon the collected data.
- Given diagram shows the Single Board Computer(SBC) based IoT device that includes CPU, GPU,RAM, storage and various types of interfaces and peripherals.



5.5 Case study:Smart Home

- In IoT enabled Smart Home environment various things such as lighting, home appliances, computers, security camera etc. all are connected to the Internet and allowing user to monitor and control things regardless of time and location constraint.
- A smart home is a residence that uses internet-connected devices to enable the remote monitoring and management of appliances and systems, such as lighting and heating.
- Smart home technology, also often referred to as home automation or domotics (from the Latin "domus" meaning home), provides home owners security, comfort, convenience and energy efficiency by allowing them to control smart devices, often by a smart home app on their smartphone or other networked device.
- Most smart home systems are controlled by smartphones and microcontrollers. A smartphone application is used to control and monitor home functions using wireless communication techniques.
- A Smart home, then, may be defined as a residence or a building with equipment which can be remotely controlled and operated from any location in the world by means of Smart Devices or through a smart phone.
- Smart Homes comprise of Devices that provide comfort, security, convenience, energy efficiency and enhance intelligent living.
- For example, a smart home may have controls for lighting, temperature, multi-media, security, window and door operations, as well as many other functions.



Examples of smart home technologies

- Smart TVs connect to the internet to access content through applications, such as on-demand video and music. Some smart TVs also include voice or gesture recognition.
- In addition to being able to be controlled remotely and customized, smart lighting systems, such as Hue from Philips Lighting Holding B.V., can detect when occupants are in the room and adjust lighting as needed. Smart light bulbs can also regulate themselves based on daylight availability.
- Smart thermostats, such as Nest from Nest Labs Inc., come with integrated Wi-Fi, allowing users to schedule, monitor and remotely control home temperatures. These devices also learn homeowners' behaviors and automatically modify settings to provide residents with maximum comfort and efficiency. Smart thermostats can also report energy use and remind users to change filters, among other things.
- Using smart locks and garage-door openers, users can grant or deny access to visitors. Smart locks can also detect when residents are near and unlock the doors for them.
- With smart security cameras, residents can monitor their homes when they are away or on vacation. Smart motion sensors are also able to identify the difference between residents, visitors, pets and burglars, and can notify authorities if suspicious behavior is detected.
- Pet care can be automated with connected feeders. Houseplants and lawns can be watered by way of connected timers.
- Kitchen appliances of all sorts are available, including smart coffee makers that can brew you a fresh cup as soon as your alarm goes off; smart refrigerators that keep track of expiration dates, make shopping lists or even create recipes based on ingredients currently on hand; slower cookers and toasters; and, in the laundry room, washing machines and dryers.
- Household system monitors may, for example, sense an electric surge and turn off appliances or sense water failures or freezing pipes and turn off the water so there isn't a flood in your basement.

Advantages:

1. Managing all of your home devices from one place.
2. Flexibility for new devices and appliances.
3. Maximizing home security.
4. Remote control of home functions.
5. Increased energy efficiency.
6. Improved appliance functionality
7. Home management insights.

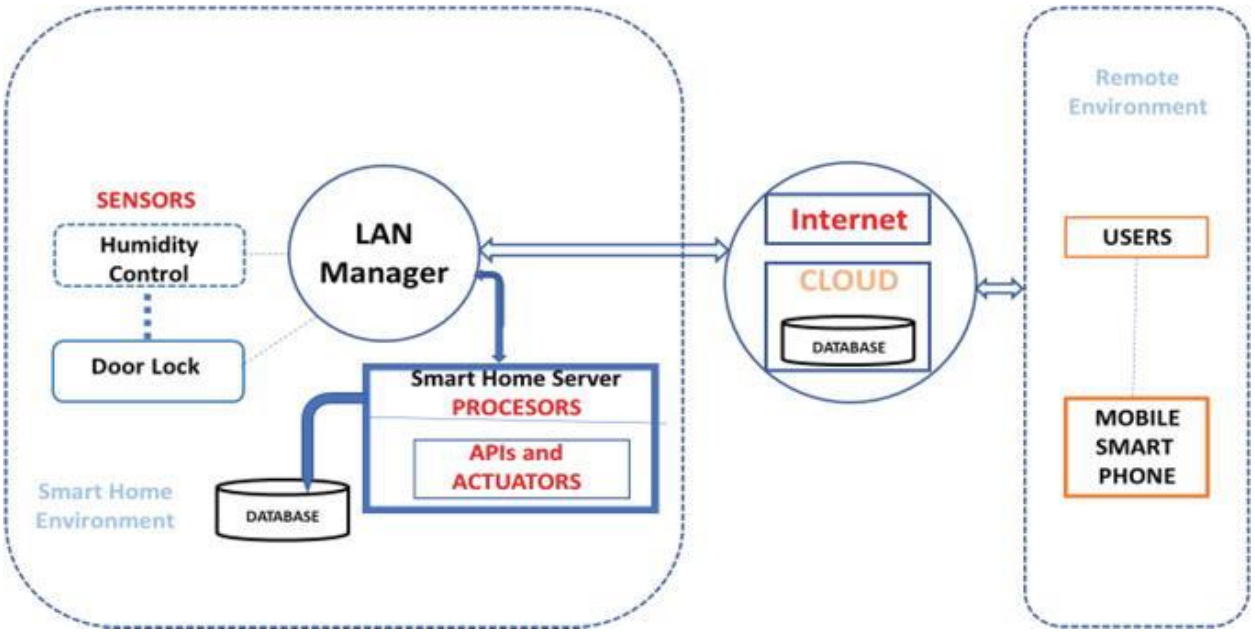
8. Convenience
9. Save time and Money
10. Improving the quality of life

Applications:

- Controlled electrical fixtures such as lights and air conditioners
- Simplified garden or lawn management
- Controlled smart home appliances
- Enhanced safety and security at home
- Water and air quality control and monitoring
- Voice based home assistant supporting natural language
- Smart locks and switches

Main components

To enable all of the above described activities and data management, the system is composed of the following components in Figure 1.



1. **Sensors:** Sensors to collect internal and external home data and measure home conditions. These sensors are connected to the home itself and to the attached-to-home devices. These sensors are attached to home appliances. The sensors' data is collected and continually transferred via the local network, to the smart home server.

2. **Processors:** Processors for performing local and integrated actions. It may also be connected to the cloud for applications requiring extended resources. The sensors' data is then processed by the local server processes.

3. **APIs:** A collection of software components wrapped as APIs, allowing external applications execute it, given it follows the pre-defined parameters format. Such an API can process sensors data or manage necessary actions.

4. **Actuators:** Actuators to provision and execute commands in the server or other control devices. It translates the required activity to the command syntax; the device can execute. During processing the received sensors' data, the task checks if any rule became true. In such case the system may launch a command to the proper device processor.

5. **Database:** Database to store the processed data collected from the sensors [and cloud services]. It will also be used for data analysis, data presentation and visualization. The processed data is saved in the attached database for future use.

Smart Home Services

- **Measuring Home Conditions:** A typical smart home is equipped with a set of sensors for measuring home conditions, such as: temperature, humidity, light and proximity. Each sensor is dedicated to capture one or more measurement. Temperature and humidity may be measured by one sensor, other sensors calculate the light ratio for a given area and the distance from it to each object exposed to it. All sensors allow storing the data and visualizing it so that the user can view it anywhere and anytime.
- **Managing Home Appliances:** The managing service allows the user, controlling the outputs of smart actuators associated with home appliances, such as such as lamps and fans. Smart actuators are devices, such as valves and switches, which perform actions such as turning things on or off or adjusting an operational system. Actuators provides a variety of functionalities, such as on/off valve service, positioning to percentage open, modulating to control changes on flow conditions, emergency shutdown (ESD). To activate an actuator, a digital write command is issued to the actuator.
- **Controlling Home Access:** Home access technologies are commonly used for public access doors. A common system uses a database with the identification attributes of authorized people. When a person is approaching the access control system, the person's identification attributes are collected instantly and compared to the database. If it matches the database data, the access is allowed, otherwise, the access is denied.

Communication Protocols

- Zigbee and Z-Wave are two of the most common home automation communications protocols in use today.

- Both mesh network technologies, they use short-range, low-power radio signals to connect smart home systems.
- Though both target the same smart home applications, Z-Wave has a range of 30 meters to Zigbee's 10 meters, with Zigbee often perceived as the more complex of the two.

Smart Home Hub

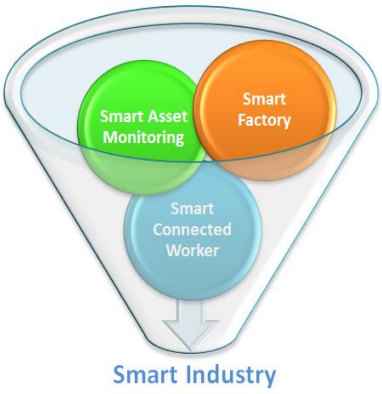
- The smart home hub is a hardware device that acts as the central point of the smart home system and is able to sense, process data and communicate wirelessly.
- It combines all of the disparate apps into a single smart home app that can be controlled remotely by homeowners.
- Examples of smart home hubs include Amazon Echo, Google Home, Insteon Hub Pro, Samsung SmartThings and Wink Hub, among others.

Events

- In simple smart home scenarios, events can be timed or triggered.
- Timed events are based on a clock, for example, lowering the blinds at 6:00 p.m., while triggered events depend on actions in the automated system.
- For example, when the owner's smart phone approaches the door, the smart lock unlocks and the smart lights go on.

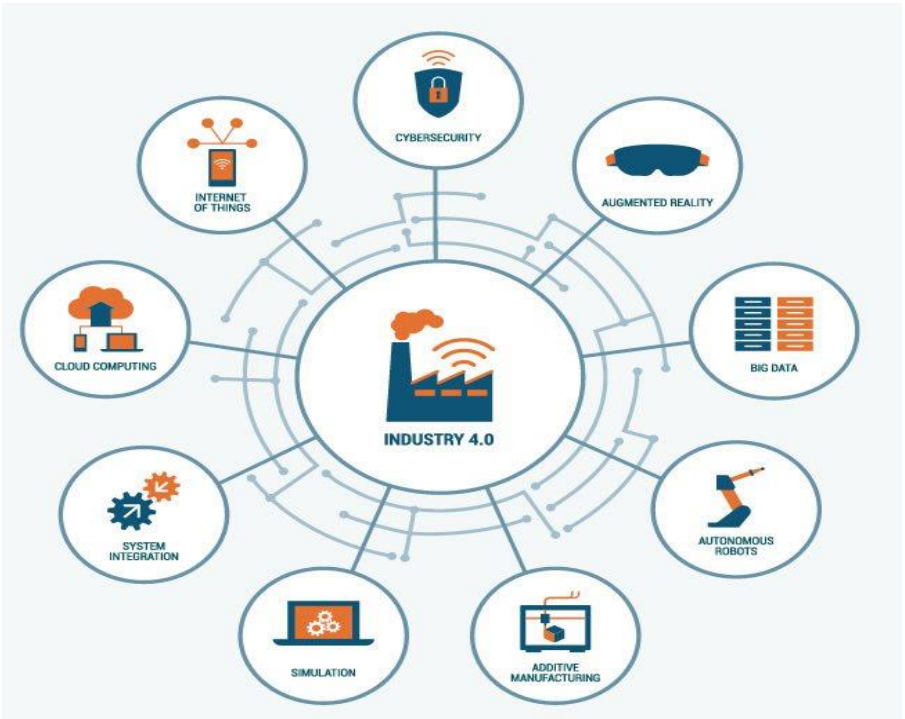
5.6 Case study: Smart Industry

- **The Industrial Internet of Things (IIoT), being the basis for the Industry 4.0 and Smart Factory**, provides connectivity for smart factories, machines, industrial infrastructure, management systems, and more, to **streamline business operations**, creating intelligent, self-optimizing industrial equipment and facilities.
- The Industrial Internet of Things (IIoT) puts a focus on this interconnectivity and data provided by the terminals, sensors, and other systems on the factory floor.
- IoT enabled machines and endpoints are capable of communicating operational information to personnel both inside and peripheral to your organization.
- That includes machine operators, managers, field service personnel and even partners like suppliers, subcontractors, and OEMs.
- This connectivity delivers mission-critical data and information to operation managers and factory leadership on-site and out-of-office.



Applications:

- Vehicle and asset tracking
- connected factory applications
- staff safety applications
- health monitoring (real-time)
- smart ventilation and air quality management
- smart environmental measurement
- access control (security)
- smart measurement of presence/levels of liquids, gases, radiation and dangerous materials
- asset protection
- facility management
- risk measurement



Advantages

- **Improved employee productivity:** Using real-time data from sensors allows employees to monitor and improve processes efficiently without delays, which enhances productivity.
- **Asset optimization:** Sensors track assets (machinery, equipment, tools, trucks, etc.) in real-time, providing visibility of their potential. On the basis of this data, businesses can make quick decisions and optimize their asset usage at maximum capacity.
- **Reduction of operating costs:** Intelligent machines and data analytics lead to reduced consumption of fuel and electricity, cuts inefficiencies and decreases overall expenses.
- **Improved quality of goods:** Automation eliminates human error, and companies are able to produce higher-quality goods.
- **Inventory management:** Together with radio frequency identification (RFID), IoT makes inventory management an efficient and seamless process. Every item in the inventory gets an RFID tag, and each tag has a unique identification number (UID) comprising encoded digital information about the item. It creates a record of the location of inventory items, their statuses and their movements in the supply chain and gives users comparable results.
- **Predictive maintenance:** Manufacturers can avoid such ineffective maintenance routines by leveraging industrial IoT and data science for predictive maintenance.
- **More Safety in operations:** In combination with big data analytics, IoT also optimises the safety of workers, equipment and operations in a manufacturing plant.
- **Smart Metering:** IoT has also introduced the manufacturing sector, utilities and other industries to the world of smart meters that can monitor the consumption of water, electric power and other fuels.
- **Smart packaging:** Smart packaging may manifest itself in the form of recipe videos, beauty tutorials and other demonstrations to explain the product usage.
-

Operations

- **Manufacturing operations:** Manufacturing operations include the several elements which are typical in Manufacturing Operations Management (MOM), such as asset management, intelligent manufacturing, performance optimization and monitoring, planning, human machine interaction, end-to-end operational visibility and these cyber-physical systems.
- **Production asset management and maintenance:** It includes production asset monitoring and tracking, from location to the monitoring of parameters in several areas such as quality, performance, potential damage or breakdowns, bottlenecks, the list goes on.
- **Field Service:** From product-related services to business-related services: the (field) service organizations of manufacturers are important drivers of growth and, obviously, of margin. It's clear that information in the hyper-connected and hyper-aware digitized and IoT-enabled manufacturing ecosystem, along with the tools to plan, schedule and pro-actively service, are important differentiators.