Unit-2

Machine-to-Machine Communications	1
Difference between IoT and M2M	4
Interoperability in IoT	8
Types of interoperability	8
HandlingApproaches	
Interoperability as per NPTEL(syllabus)	18
Introduction to Arduino Programming	27
Aurdino Board	29
Built in Functions	33
Program for Blinking LED	45
Traffic system	47
Integration of Sensors and Actuators with Arduino	
Sensors with Aurdino	50
Actuators with Aurdino	53

UNIT-2

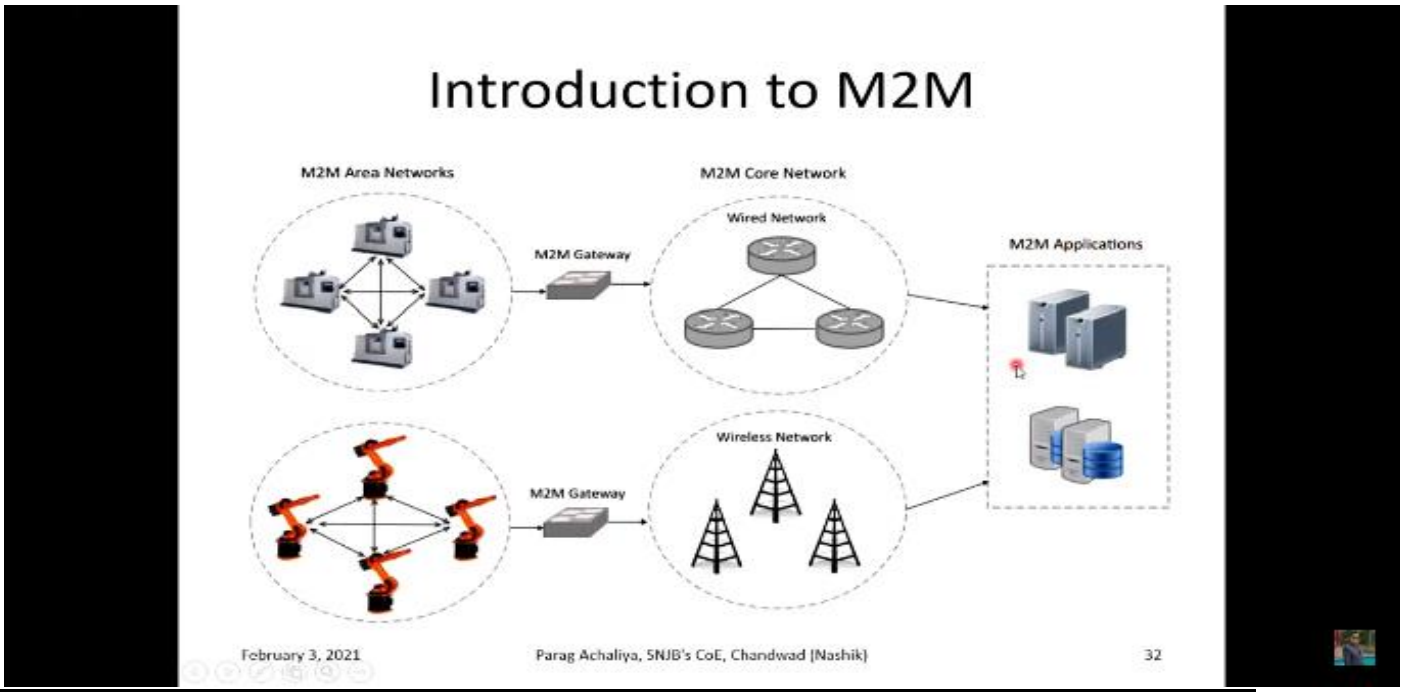
Machine to Machine :
This is commonly known as Machine to machine communication. It is a concept where two or more than two machines communicate with each other without human interaction using a wired or wireless mechanism. M2M is an technology that helps the devices to connect between devices without using internet.

Internet of Things :
IOT is known as the Internet of Things where things are said to be the communicating devices that can interact with each other using a communication media. Usually every day some new devices are being integrated which uses IoT devices for its function. These devices use various sensors and actuators for sending and receiving data over the internet. It is an ecosystem where the devices share data through a communication media known as the internet.

Machine-to-Machine (M2M)

- Machine-to-Machine (M2M) refers to networking of machines (or devices) for the purpose of remote monitoring and control and data exchange.

M2M System architecture

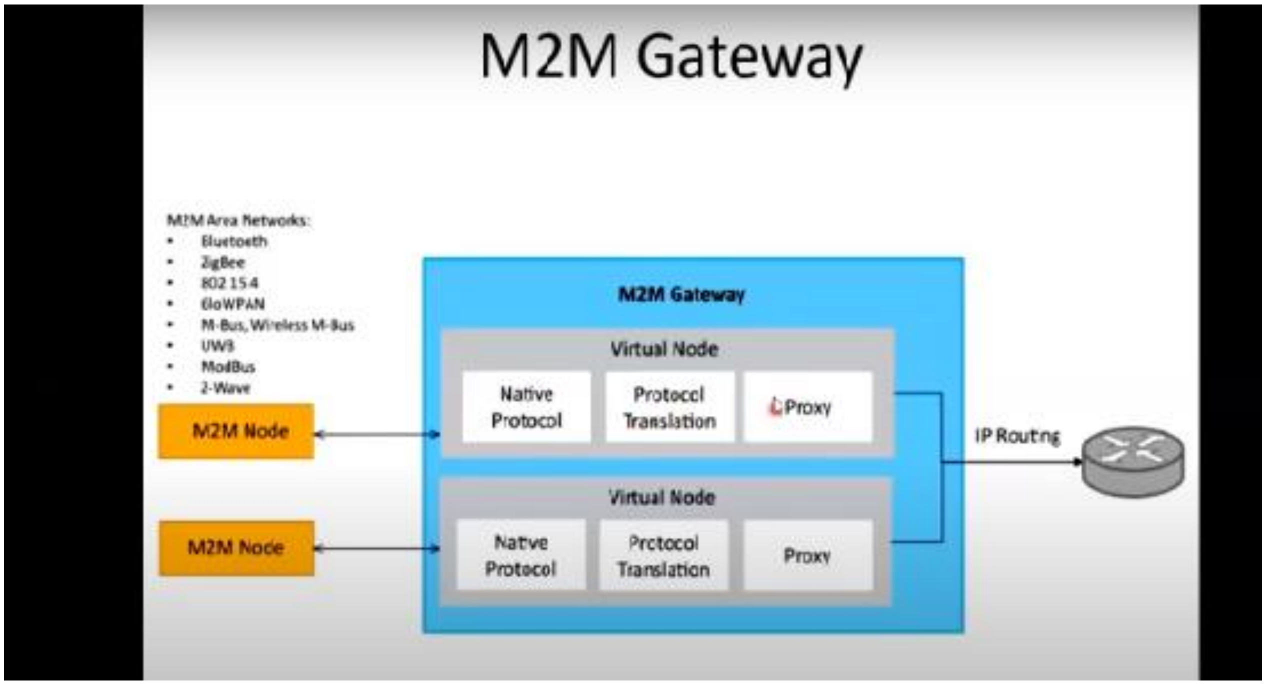


Introduction to M2M

February 3, 2021 Parag Achaliya, SNJB's CoE, Chandwad (Nashik) 32

[redacted]

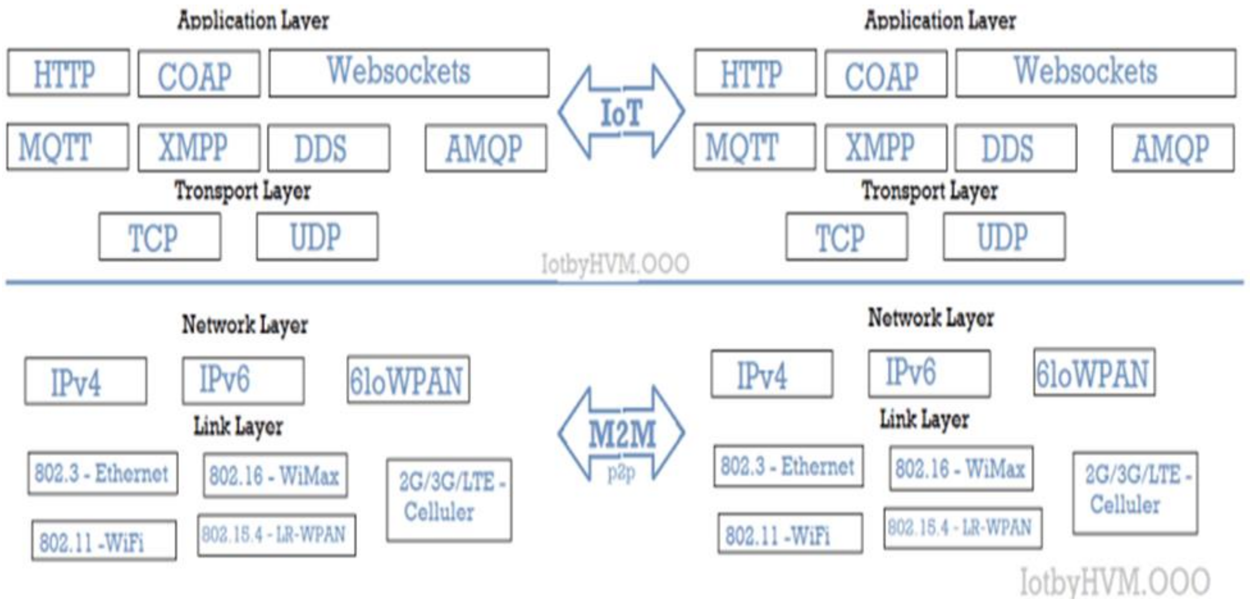
Machine-to-Machine (M2M)
• An M2M area network comprises of machines (or M2M nodes) which have embedded hardware modules for sensing, actuation and communication.
• Various communication protocols can be used for M2M local area networks such as ZigBee, Bluetooh, ModBus, M-Bus, Wirless M-Bus, Power Line Communication (PLC), 6LoWPAN, IEEE 802.15.4, etc.
• The communication network provides connectivity to remote M2M area networks.
• The communication network can use either wired or wireless networks (IPbased).
• M2M area networks use either proprietary or non-IP based communication protocols, the communication network uses IP-based networks M2M gateway
• Since non-IP based protocols are used within M2M area networks, the M2M nodes within one network cannot communicate with nodes in an external network.
• To enable the communication between remote M2M area networks, M2M gateways are used.



DIFFERENCE BETWEEN IOT AND M2M

Communication Protocols

M2M and IoT can differ in how the communication between the machines or devices happens. M2M uses either proprietary or non-IP based communication protocols for communication within the M2M area networks. Commonly uses m2m protocols include **Zigbee, bluetooth, Modbus, M-Bus, wireless M-bus, Power line communication, 6lowpan, IEEE 802.15.4, Z-wave** etc. The focus of communication in M2M is usually on the protocols below the network layer. The focus of communication in IoT is usually on the protocols above above the network layer such as **HTTP, COAP, Websockets, MQTT, XMPP, DDS, AMQP** etc.

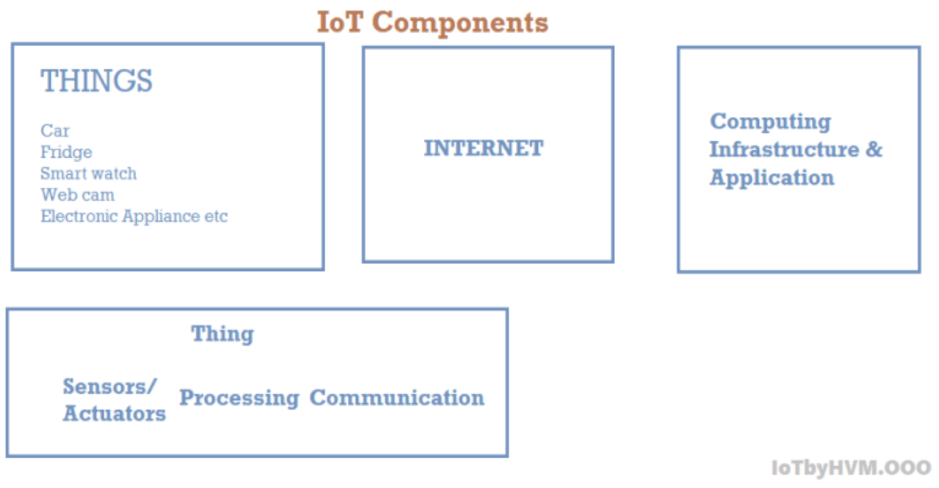


Machines in M2M vs things in IoT

The **things** in IoT refers to physical objects that have unique identifiers and can sense and communicate with their external environment (and user applications) or their internal physical States. The unique identifier for the things in IoT are the IP addresses (or Mac addresses). Things have software components for accessing, processing and storing sensor information, or controlling actuators connected. IoT systems can have heterogeneous thing (e.g., home automation IoT system can include IoT system can include IoT devices of various types such as fire alarms, door alarms, lighting control devices etc) **M2M systems, in contrast to IoT, typically have have homogeneous machine type within a and M2M area network.**

Hardware vs Software Emphasis

While the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software. IoT devices run specialized software for sensor data collection, data analysis and interfacing with the cloud through IP based communication.



.

Data Collection and Analysis

M2M data is collected in point Solutions and often in on-premises storage infrastructure. In context to M2M, the data in IoT is collected in the cloud (can be public, private or hybrid cloud). The various IoT Levels, and IoT components deployed in the cloud. The analytics component analyzes the data and stores the results in the cloud database. The IoT data and analysis results are visualized with the cloud based applications. The centralized controller is aware of the status of all the end nodes and sends control commands to the nodes. Observer nodes can process information and use it for various applications, however, observer nodes do not perform any control functions.

Applications

M2M data is collected in point Solutions and can be accessed by on-premises application such as diagnosis applications, service management applications and on-premises enterprise applications. But We collect IoT Data in the cloud and can be accessed by cloud applications such as analytics applications, enterprise applications, remote diagnosis and management applications etc. Since the scale of data collected in IoT is so massive, cloud based real time and batch data analysis frameworks are used for data analysis. Communication in IoT is IP based networks. Communication within M2M area network is based on protocols below the network layer whereas IoT is based on protocols above the network layer.

Difference between IoT and M2M : (in table format)

Basis of	IoT	M2M
Abbreviation	Internet of Things	Machine to Machine
Intelligence	Devices have objects that are responsible for decision making	Some degree of intelligence is observed in this
Connection type used	The connection is via Network and using various communication types.	The connection is a point to point
Communication protocol used	Internet protocols are used such as HTTP, FTP, and Telnet.	Traditional protocols and communication technology techniques are used
Data Sharing	Data is shared between other applications that are used to improve the end-user experience.	Data is shared with only the communicating parties.
Internet	Internet connection is required for communication	Devices are not dependent on the Internet.
Scope	A large number of devices yet scope is large.	Limited Scope for devices.
Business Type used	Business 2 Business(B2B) and Business 2 Consumer(B2C)	Business 2 Business (B2B)

Basis of	IoT	M2M
Open API support	Supports Open API integrations.	There is no support for Open Api's
Examples	Smart wearables, Big Data and Cloud, etc.	Sensors, Data and Information, etc.

INTEROPERABILITY IN IOT

Interoperability is defined by IEEE as "*the ability of two or more systems or components to exchange information and to use the information that has been exchanged.*

In IoT interoperability can be defined as the ability of two systems to communicate and share services with each other

The interoperability issues in IoT can be seen from different perspectives due to heterogeneity. Heterogeneity is not a new concept nor restricted to a domain. Even in the physical world there are many types of heterogeneities for example, people speak dissimilar languages, but they can still communicate with each other through a translator (human/tools) or by using a common language. similarly the diverse elements comprising IoT (devices, communication, services, applications, etc.) should seamlessly cooperate and communicate with each other to realize the full potential of IoT ecosystem. **As shown in figure, IoT interoperability can be seen from different perspectives such as**

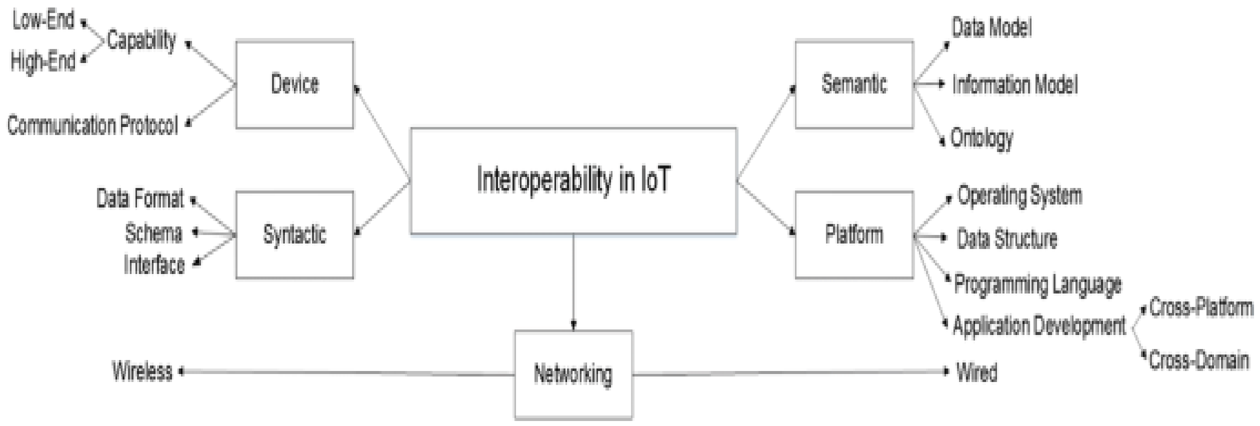
device interoperability

networking interoperability

syntactic interoperability

semantic interoperability and

platform interoperability.



Device interoperability

IoT is composed of a variety of devices which are called “smart objects/things”, which may consist of *high-end devices* or *low-end devices*. The *high-end IoT devices* have enough resources and computational capabilities such as Raspberry Pi and smartphones. On the other hand, the *low-end IoT devices* are resource-constrained in terms of energy, processing power and communication capabilities , tiny and low-cost sensors, and actuators, Arduino.

various communication protocols have emerged due to the different requirements of IoT markets. For example, IoT devices such as Smart TV, printers, air conditioners support traditional Wi-Fi technologies and 3G/4G cellular communications, IoT medical devices are based on ANT+ standard; other wearable devices mostly support Bluetooth SMART and NFC, while the environmental sensors use ZigBee-based on IEEE 802.15.4 standard.

Device interoperability refers to enabling the integration and interoperability of such heterogenous devices with various communication protocols and standards.

Device interoperability is concerned with

- **(i)** **the exchange of information between heterogeneous devices and heterogenous communication protocols and**
- **(ii)** **(ii) the ability to integrate new devices into any IoT platform.**

Network interoperability

Network level interoperability deals with mechanisms to enable seamless message exchange between systems through different networks (networks of networks) for end-to-end communication. To make systems interoperable, i.e each system should be able to exchange messages with other systems through various types of networks. Due to the dynamic and heterogenous network environment in IoT, the network interoperability level should handle issues such as addressing, routing, resource optimization, security, QoS, and mobility support

Syntactical interoperability

Syntactic interoperability refers to interoperation of the format as well as the data structure used in any exchanged information or service between heterogeneous IoT system entities.

The content of the messages need to be serialized to be sent over the channel and the format to do so (such as XML or JSON). The message sender encodes data in a message using syntactic rules, specified in some grammar. The message receiver decodes the received message using syntactic rules defined in the same or some other grammar**.** Syntactic interoperability problems arise when the sender's encoding rules are incompatible with the receiver's decoding rules, which leads to mismatching message parse trees.

semantic interoperability

semantic interoperability is defined as "enabling different agents, services, and applications to exchange information, data and knowledge in a meaningful way, on and off the Web"**.** The WoT addresses the current fragmentation by exposing things and systems data and metadata through API. But, such efforts have been hampered because the corresponding parties need to share knowledge of an API [27] and many devices do not speak the same language and cannot exchange across different gateways and smart hub.

For ex: the data generated by things about the environment may have a defined data format (e.g. JSON, XML or CSV), but the data models and schemas used by different sources are usually dissimilar and not always compatible.

This semantic incompatibility between data models and information models results in IoT systems not being able to dynamically and automatically inter-operate as they have different descriptions or understandings of resources and operational procedures, even if IoT systems expose their data and resources to others.

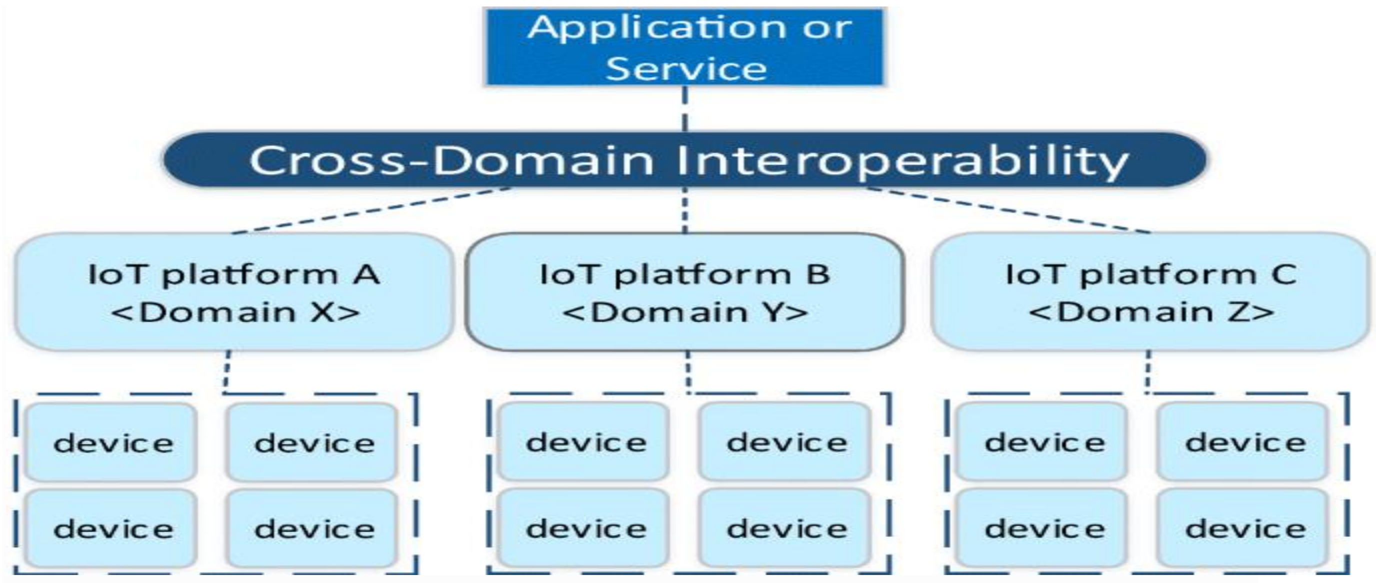
Platform interoperability

Platform interoperability issues in IoT arises due to the availability of diverse operating systems (OSs), programming languages, data structures, architectures and access mechanisms for things and data.

Developers need to obtain extensive knowledge of the platform specific APIs and information models of each different platform to be able to adapt their applications from one platform to another.

A cross-platform IoT application can access different IoT platforms and integrate data from various platforms. **The cross-platform interoperability between things and data in this scenario enables interoperability across separate IoT platforms specific to one vertical domain such as smart home, smart healthcare, smart garden, etc.** After cross-platform interoperability is enabled, cross-domain interoperability can be achieved in which different platforms within heterogenous domains are federated to build horizontal IoT applications**.** Fig.. shows **the concept behind cross-domain interoperability where different IoT platforms from different IoT domains (e.g. health, home, transport, etc.) can be integrated to build new innovative applications. For example, a smart home platform can provide domain-specific enablers such as air temperature and the lighting conditions.**

Fig. 2.


Application or Service
Cross-Domain Interoperability
IoT platform A <Domain X>
IoT platform B <Domain Y>
IoT platform C <Domain Z>
device

Interoperability handling approaches in IoT

To improve the state of IoT interoperability, researchers have leveraged numerous approaches and technologies which we refer to interoperability handling approaches

1.Adapters/gateways

2.Virtual networks/ overlay-based solutions

3.Networking technologies

IP based approaches

Software-defined networking (SDN)

Network function virtualization

Fog computing

4.Service oriented architecture (SOA)

5.Open API

6.Semantic web technologies

7.Open standard

Adapters/gateways

Gateways or adapters are the class of schemes which address interoperability through the development of an intermediate tool sometimes called mediators to improve interoperability between IoT devices.

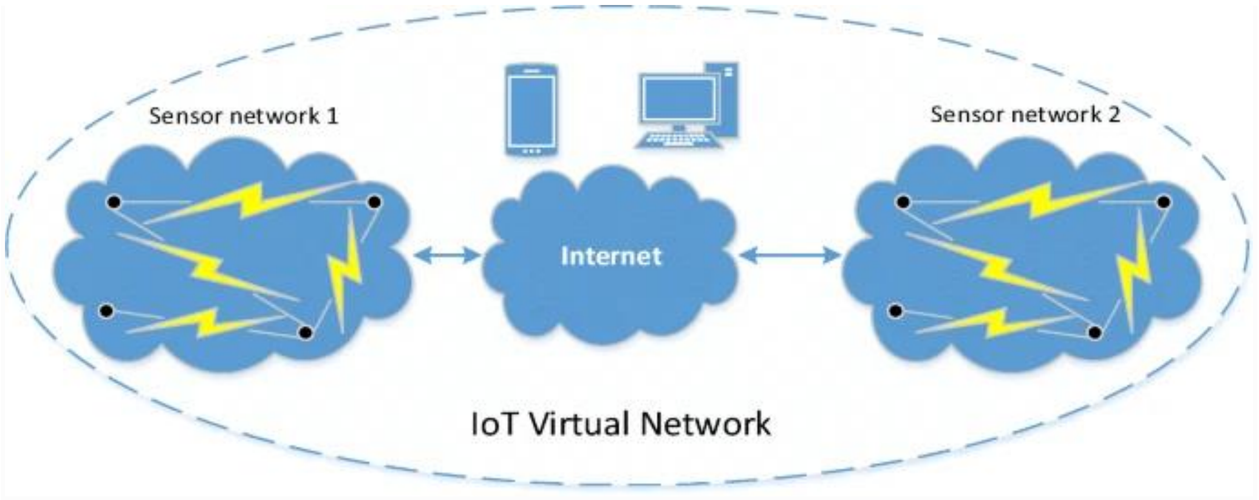
The objective here to bridge between different specifications, data, standards, and middleware's etc. **To perform a conversion between the protocol of the sending device and the protocol of the receiving device, the gateway can be expanded with the use of plug-ins.**

For example, when IoT devices use dissimilar communication technologies (i.e., Bluetooth and ZigBee) or when they use dissimilar application layer protocols (i.e., XMPP and MQTT)

Gateways can be dedicated hardware, or the function can be embedded in the firmware or software of an intelligent device such as a programmable logic controller (PLC), human-human interface (HMI), or computer.

Virtual networks/ overlay-based solutions

Virtual networks or Overlay-based solutions have been proposed in as **"Managed Ecosystems of Networked Objects" (MENO), with the aim to integrate sensor and actuators and other IP-smart objects seamlessly to the Internet for end-to-end** communication. **The main idea behind MENO is to create a virtual network on top of physical networks and thereby allow communication with other types of devices, including sensor nodes. Within each virtual network, end-to-end communication is possible using different protocols**. Once end-to-end communication is enabled, it becomes possible for application developers to write new applications that utilize sensors, actuators, and other devices



The advantage of this approach is enabling end-to-end communication between devices, however the key issues are scalability and binding to specific protocols.

Networking technologies

Different networking protocols and technologies are used to provide networking interoperability .

For ex : The conventional universal plugin (UPnp) and DLNA protocols are used for communication between iot devices and different gateways.

The main technologies / solutions for interoperability at network levels are

IP BASED APPROACHES

SDN

NFV

FOG COMPUTING

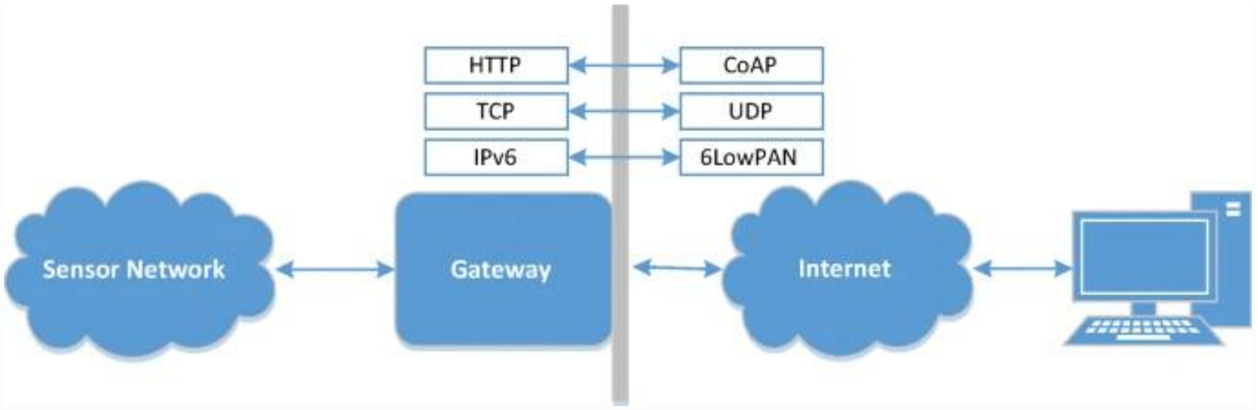
IP-based approaches

The IP-based approaches embed the full TCP/IP stack on smart devices. By embedding the TCP/IP stack as shown in Fig. the sensor and actuators are directly connected to the IP network to allow end-to-end communication between sensor network and IP network.

The key benefit of implementing the TCP/IP stack on sensor nodes is that gateways and protocol translations are not required.

However, an all IP sensor network is not possible on sensor nodes because of their resource-constraint property.

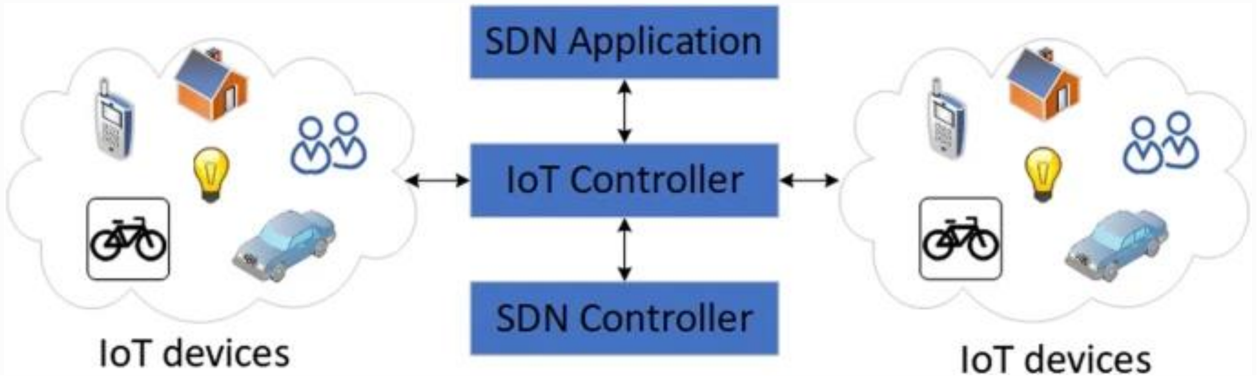
protocols at the network layer such as Routing Over Low Power and Lossy Networks (ROLL) ,IPv6 over Low Power WPAN (6LoWPAN), Constrained Application Protocol (CoAP) which is based on UDP, and Constrained Restful Environment to solve the connectivity problem of resource-constrained devices. This approach, still uses gateways to convert between standard protocols used in the Internet and protocols used in the sensor network, e.g. IPv6 to 6LoWPAN.



Software-defined networking (SDN)

Software defined networking (SDN) is a new networking paradigm to make the current wireless and mobile networks more “intelligent”, efficient, secure, and scalable in order to handle the large amount of data produced in the IoT

One of the main idea of SDN , is to separate the control and data planes in networking devices



SDN is used to allow different objects from different networks to communicate with each other using IPv6 and at the same time simplify the management and control operations of various objects types by adding an additional IoT controller over the SDN controller. Thus, even so the devices have different protocols, the forwarding devices in the router convert it in a form understandable by the receiver. This enables the communication of diverse devices in the network.

Network function virtualization

NFV separates the physical network equipment's (i.e., network address translator, firewall) from the functions that run on them. This way, numerous service providers can create several isolated virtual networks which could then share the physical network equipment's provided by the network infrastructure providers.

NFV has the potential to reduce Operational Expenditure (OPEX) and Capital Expenditure (CAPEX) costs by sharing the network infrastructure, dynamic scaling, on-the-fly, and flexible network function deployment

Fog computing

The cloud has been used as a medium to address interoperability called the Fog of Things where the computing, storage and networking services are placed at the edge of the network rather than centralized cloud servers, i.e., as close as possible to the end user devices.

Service oriented architecture (SOA)

To provide syntactic interoperability between heterogeneous devices and across all systems, researchers have proposed Service Oriented Architecture.

In the SOA of the IoT, the interaction with and operations of different wireless devices are classified into different service components and the application layer software can access resources exposed by devices as services. Exposing each component's functionalities as a standard service can significantly increase the interoperability of both network and device.

Semantic web technologies

.

The Semantic Web of Things (SWoT) paradigm is proposed for the integration of the Semantic Web with the WoT, for realizing a common understanding of the various entities which form the IoT.

Ontologies (or vocabularies) in IoT are a set of objects and relationships used to define and represent an area of concern. They represent an abstraction technology which aims to hide heterogeneity of IoT entities, acting as a mediator between IoT application provider and consumers, and to support their semantic matchmaking

Some popular approaches

Ontology

- Device ontology
- Physical domain ontology
- Estimation ontology

Collaborative conceptualization theory

- Object is defined based on the collaborative concept, which is calledcosign
- The representation of a collaborative sign is defined as follows:
- cosign of a object = (A, B, C, D), where A is a cosign internal identifier, B isa natural language, C is the context of A, and D is a definition of the object
- As an example of CCTV, cosign = (1234, English, CCTV, “Camera Type: Bullet, Communication: Network/IP, Horizontal Resolution: 2048 TVL”)

This solution approach is applicable for different domains/contexts

Many ontologies have been proposed in the context of IoT such as W3C Semantic Sensor Network (SSN) , IoT-Ontology, SAREF and OpenIoT.

Open API

With the massive development of IoT platform providers a vast silo of diverse APIs has been created that increases the difficulty of developing applications as well as interoperability issues.

HyperCat is a specification which provides syntactic interoperability between different APIs and services based on a Catalog that can be tagged with metadata.

Moreover, the symbIoTe and Big-IoT European projects are working on a generic interworking API to provide uniform access to resources of all existing and future IoT platforms to address syntactic and cross-platform interoperability.

Open standards

Open standards are one significant means to provide interoperability between and within different domains

Interoperability in IoT as per NPTEL

Current Challenges in IoT

Large Scale of Co-Operation:

The cooperation and coordination of millions of distributed devices are required onInternet

Global Heterogeneity:

Heterogeneous IoT devices and their subnets

Unknown IoT Device Configuration:

The different configuration modes for IoT devices which come from unknownowners

Semantic Conflicts:

Different processing logics applied to same IoT networked devices or applications.

What is Interoperability?

- ✓ Interoperability is a characteristic of a product or system, whose interfaces are completely understood, to work with other products or systems, present or future, in either implementation or access, without any restrictions.
  - Communicate meaningfully
  - Exchange data or services

Why Interoperability is Important in Context of IoT?

To fulfill the IoT objectives

Physical objects can interact with any other physical objects and can share their information

Any device can communicate with other devices anytime from anywhere

Machine to Machine communication(M2M), Device to Device Communication (D2D), Device to Machine Communication (D2M)

Seamless device integration with IoT network

Why Interoperability is required?

Heterogeneity

1. Different wireless communication protocols such as ZigBee (IEEE 802.15.4), Bluetooth (IEEE 802.15.1), GPRS, 6LowPAN, and Wi-Fi (IEEE802.11)

2. Different wired communication protocols like Ethernet (IEEE 802.3)

andHigher Layer LAN Protocols (IEEE 802.1)

3. Different programming languages used in computing systems and websites such as JavaScript, JAVA, C, C++, Visual Basic, PHP, and Python

4. Different hardware platforms such as Crossbow, NI, etc.

5. Different operating systems

As an example for sensor node: TinyOS, SOS, Mantis OS, RETOS, andmostly vendor specific OS

As an example for personal computer: Windows, Mac, Unix, and Ubuntu

6. Different databases: DB2, MySQL, Oracle, PostgreSQL, SQLite, SQL Server, and Sybase

7. Different data representations

8. Different control models

9. Syntactic or semantic interpretations

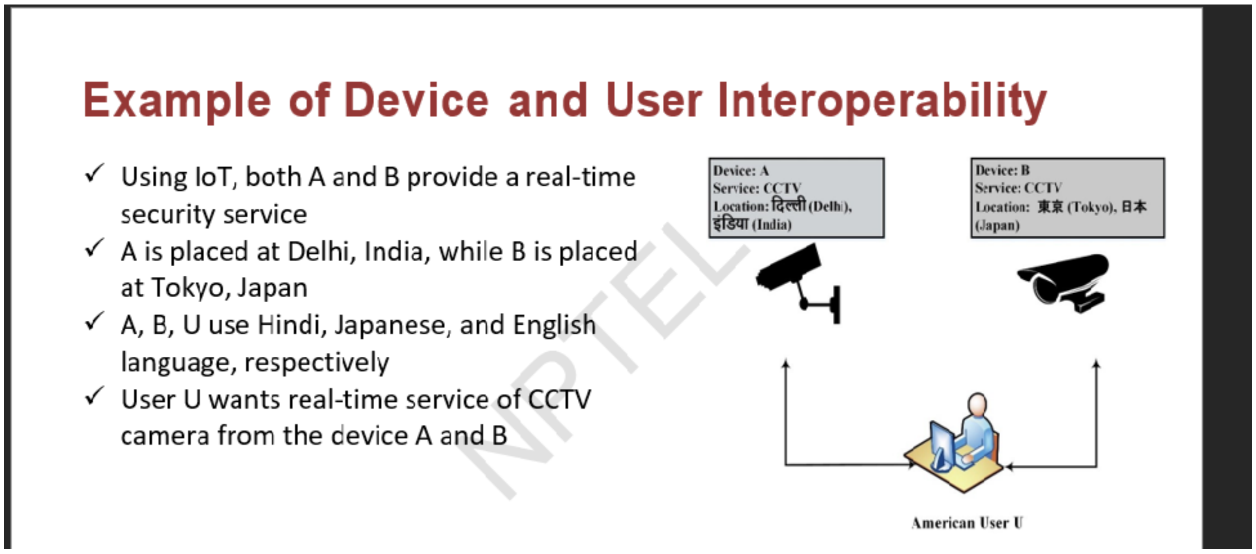
Different Types of Interoperability?

User Interoperability

- Interoperability problem between a user and a device

Device Interoperability

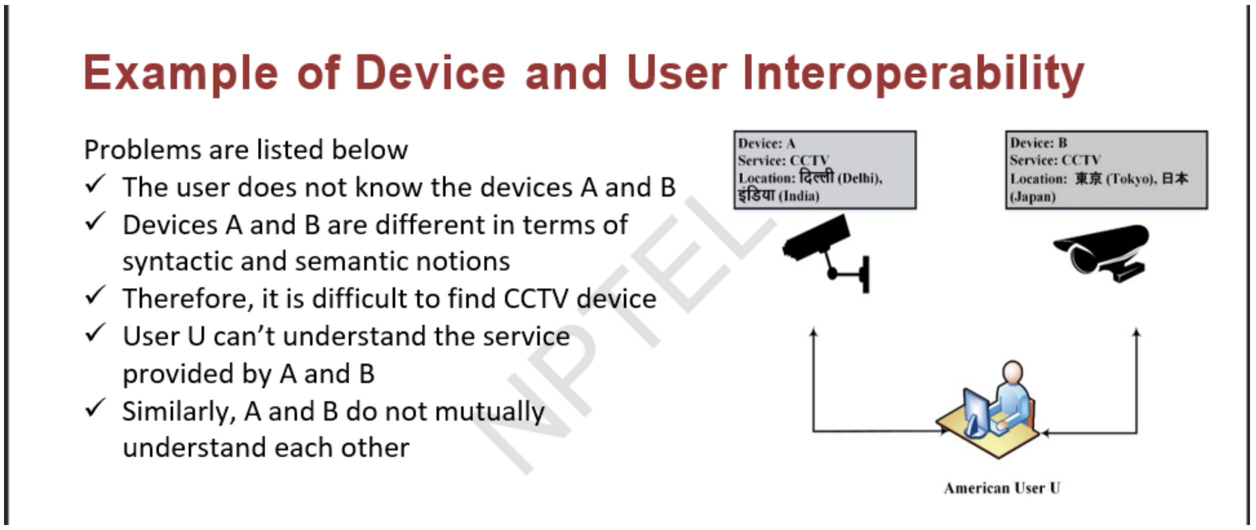
- Interoperability problem between two different devices



Example of Device and User Interoperability

- ✓ Using IoT, both A and B provide a real-time security service
- ✓ A is placed at Delhi, India, while B is placed at Tokyo, Japan
- ✓ A, B, U use Hindi, Japanese, and English language, respectively
- ✓ User U wants real-time service of CCTV camera from the device A and B

Device: A
Service: CCTV
Location: दिल्ली (Delhi), इंडिया (India)
Device: B
Service: CCTV
Location: 東京 (Tokyo), 日本 (Japan)
American User U



Example of Device and User Interoperability

Problems are listed below

- ✓ The user does not know the devices A and B
- ✓ Devices A and B are different in terms of syntactic and semantic notions
- ✓ Therefore, it is difficult to find CCTV device
- ✓ User U can't understand the service provided by A and B
- ✓ Similarly, A and B do not mutually understand each other

Device: A
Service: CCTV
Location: दिल्ली (Delhi), इंडिया (India)
Device: B
Service: CCTV
Location: 東京 (Tokyo), 日本 (Japan)
American User U

User Interoperability

The following problems need to be solved

→Device identification and categorization for discovery.

→Syntactic interoperability for device interaction.

→Semantic interoperability for device interaction.

Device identification and categorization fordiscovery

There are different solutions for generating unique address

- ✓ Electronic Product Codes (EPC)
- ✓ Universal Product Code (UPC)
- ✓ Uniform Resource Identifier (URI)
- ✓ IP Addresses
  - ▪ IPv6

There are different device classification solutions

- ✓ United Nations Standard Products and Services Code (UNSPSC) *
  - ▪ an open, global, multi-sector standard for efficient, accurate, flexibleclassification of products and services.
- ✓ eCl@ss **
  - ▪ The standard is for classification and clear description of cross-industryproducts

Syntactic Interoperability for Device Interaction

The interoperability between devices and device user in termof message formats

The message format from a device to a user is understandablefor the user’s computer

On the other hand, the message format from the user to thedevice is executable by the device

Some popular approaches are

Service-oriented Computing (SOC)-based architecture

Web services

RESTful web services

Open standard protocols such as IEEE 802.15.4, IEEE 802.15.1, and WirelessHART*

Closed protocols such as Z-Wave*

- ✓Middleware technology
  - Software middleware bridge
  - Dynamically map physical devices with different domains
  - Based on the map, the devices can be discovered and controlled,remotely
- ✓Cross-context syntactic interoperability
  - Collaborative concept exchange
  - Using XML syntax
- ✓

The above methodologies can achieve :
1.The interoperability between devices and device user in termof message's meaning

2.The device can understand the meaning of user's instructionthat is sent from the user to the device.

3.Similarly, the user can understand the meaning of device's response sent from the device

Some popular approaches in Ontology are:

Device ontology

Physical domain ontology

Estimation ontology

✓ Collaborative conceptualization theory

- Object is defined based on the collaborative concept, which is calledcosign
- The representation of a collaborative sign is defined as follows:
- cosign of a object = (A, B, C, D), where A is a cosign internal identifier, B isa natural language, C is the context of A, and D is a definition of the object
- As an example of CCTV, cosign = (1234, English, CCTV, “Camera Type: Bullet, Communication: Network/IP, Horizontal Resolution: 2048 TVL”)

✓ This solution approach is applicable for different domains/contexts

Device Interoperability

Solution approach for device interoperability

✓ Universal Middleware Bridge (UMB)

- Solves seamless interoperability problems caused by the heterogeneity of several kinds of home network middleware
- UMB creates virtual maps among the physical devices of all middleware home networks, such as HAVI, Jini, LonWorks, and UPnP
- Creates a compatibility among these middleware home networks

UMB consists

✓ UMB Core (UMB-C)

✓ UMB Adaptor (UMB-A)

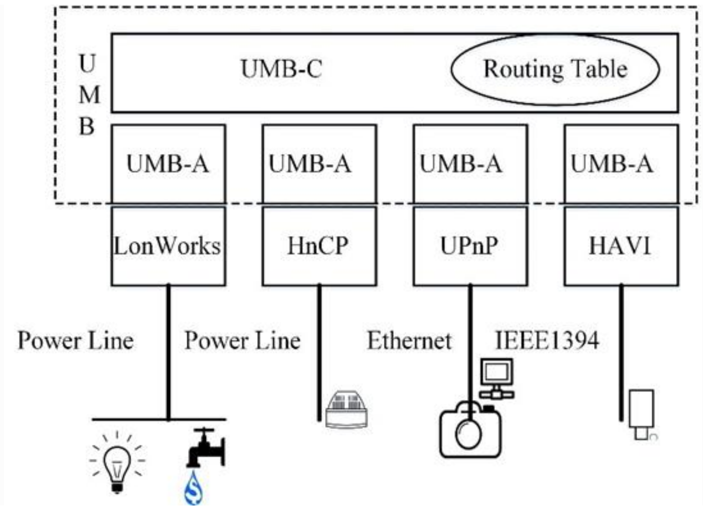
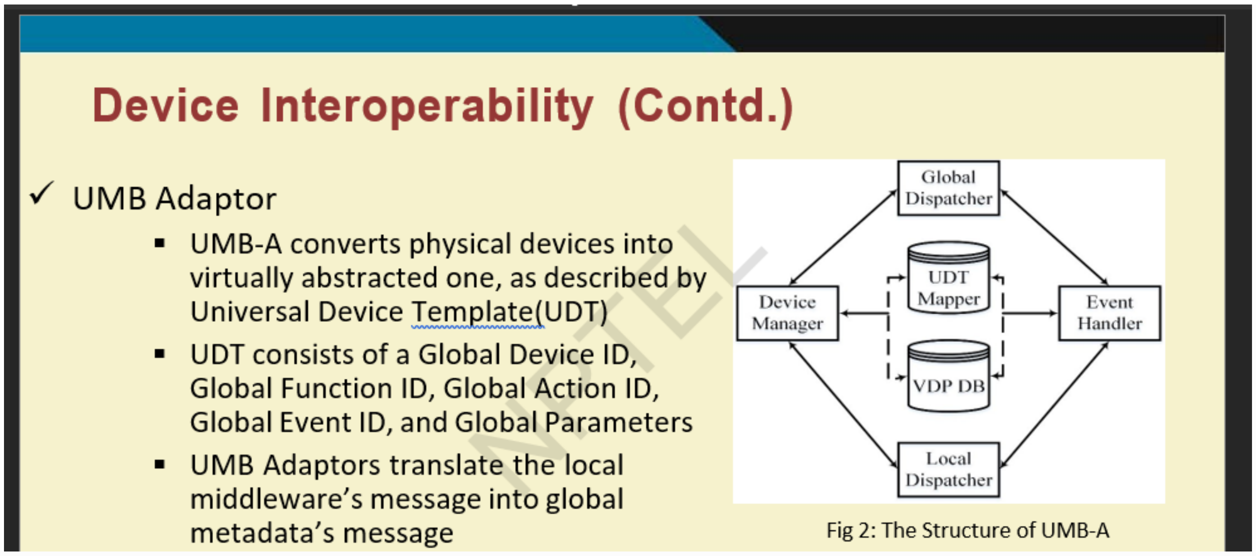


Fig 1: The Architecture of Universal Middleware Bridge

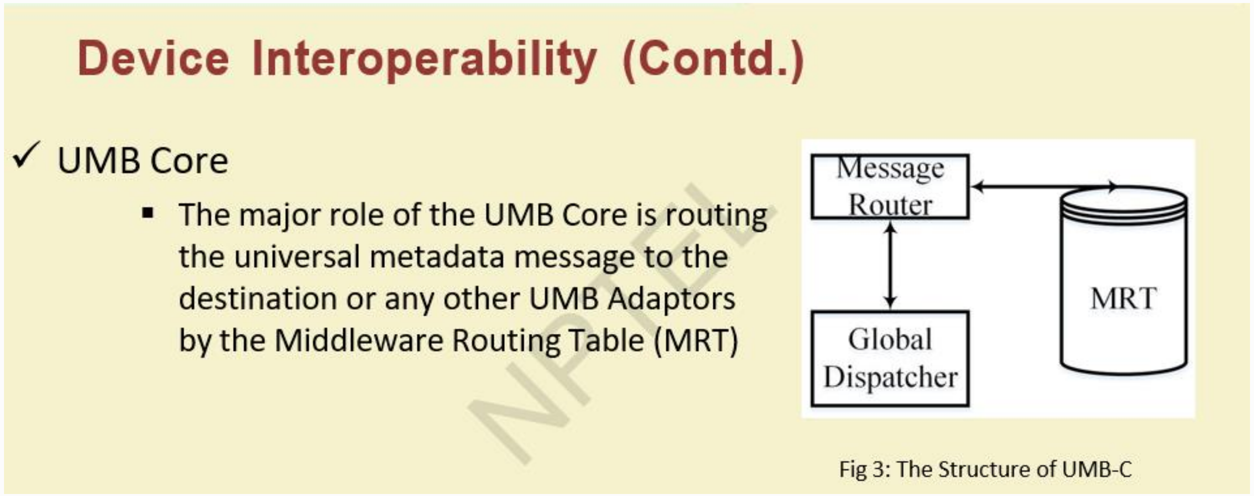


Device Interoperability (Contd.)

- ✓ UMB Adaptor
  - UMB-A converts physical devices into virtually abstracted one, as described by Universal Device Template(UDT)
  - UDT consists of a Global Device ID, Global Function ID, Global Action ID, Global Event ID, and Global Parameters
  - UMB Adaptors translate the local middleware's message into global metadata's message

Global Dispatcher
UDT Mapper
Device Manager
Event Handler
VDP DB
Local Dispatcher

Fig 2: The Structure of UMB-A

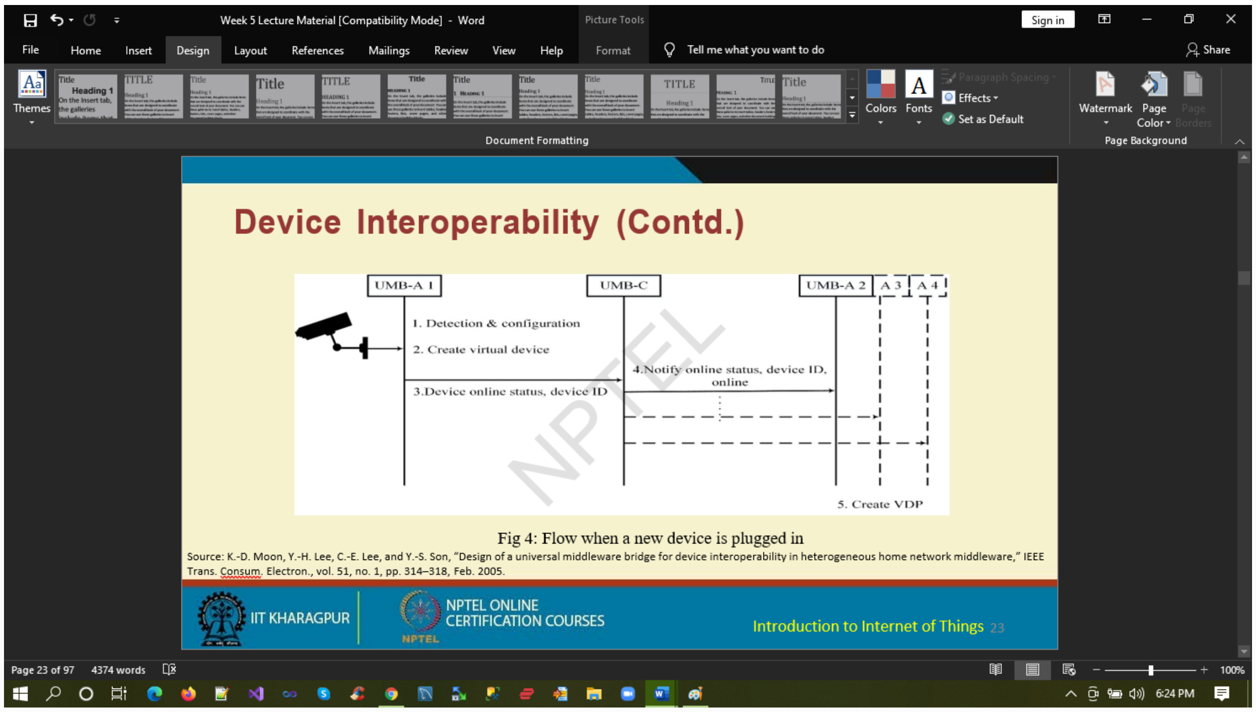


Device Interoperability (Contd.)

- ✓ UMB Core
  - The major role of the UMB Core is routing the universal metadata message to the destination or any other UMB Adaptors by the Middleware Routing Table (MRT)

Message Router
MRT
Global Dispatcher

Fig 3: The Structure of UMB-C

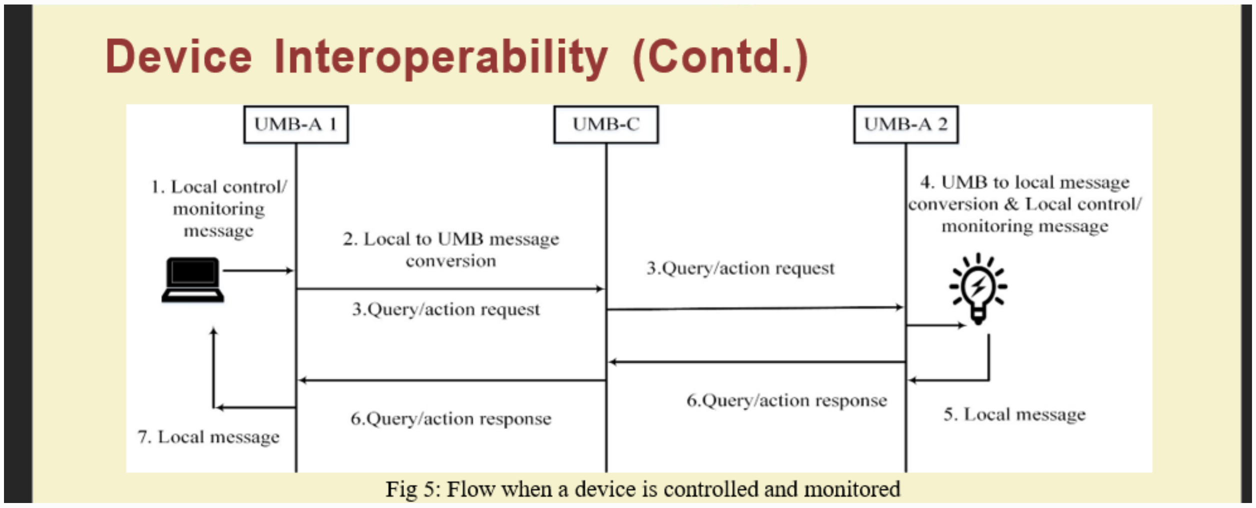


Device Interoperability (Contd.)

Fig 4: Flow when a new device is plugged in

Source: K.-D. Moon, Y.-H. Lee, C.-E. Lee, and Y.-S. Son, "Design of a universal middleware bridge for device interoperability in heterogeneous home network middleware," IEEE Trans. Consum. Electron., vol. 51, no. 1, pp. 314–318, Feb. 2005.

Introduction to Internet of Things 23



Device Interoperability (Contd.)

Fig 5: Flow when a device is controlled and monitored

AURDINO

What is Arduino?

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Arduino has been used in thousands of different projects and applications. The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux.

for example

Designers and architects build interactive prototypes, musicians and artists use it for installations and to experiment with new musical instruments. Makers, of course, use it to build many of the projects exhibited at the Maker Faire,

Arduino simplifies the process of working with microcontrollers, but it offers some advantage

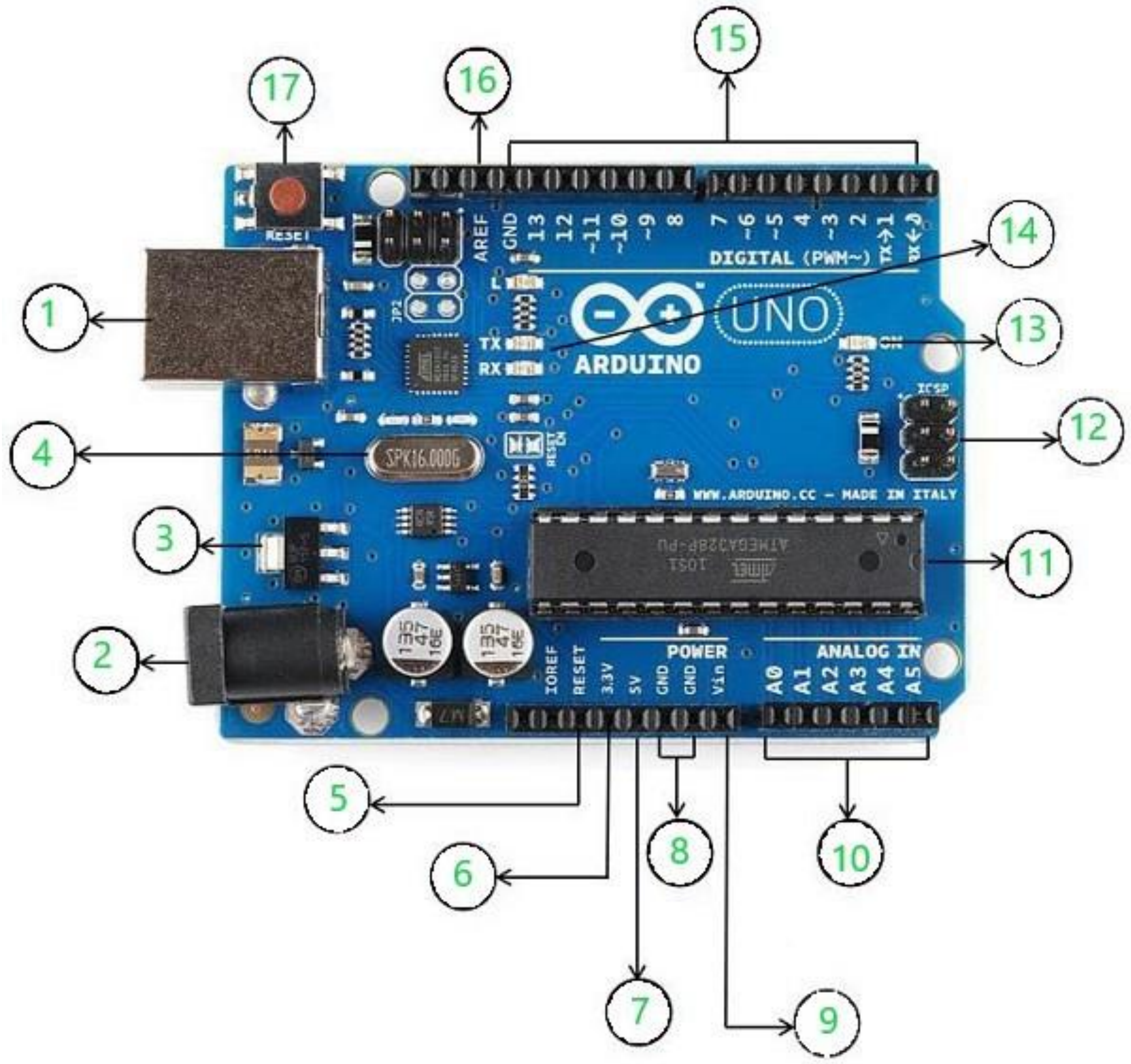
- **Inexpensive** - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50
- **Cross-platform** - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- **Simple, clear programming environment** - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. .
- **Open source and extensible software** - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and

people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.

- **Open source and extensible hardware** - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works .

The Arduino employs an 8-bit ATmega series **microcontroller** whereas the **Raspberry Pi** is based around a 32-bit ARM **processor**,

AURDINO BOARD AND ITS COMPONENTS



Using the above image as a reference, the labeled components of the board respectively are-

	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).

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

	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.
	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.
	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.
	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.
	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.
	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.

Programming with the Arduino IDE

The **Arduino** Integrated Development Environment - or **Arduino** Software (**IDE**) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the **Arduino** and Genuino hardware to upload programs and communicate with them.

Arduino IDE is an open source software that is used to program the Arduino controller board It can be downloaded from Arduino's official website and installed into PC .

Set Up

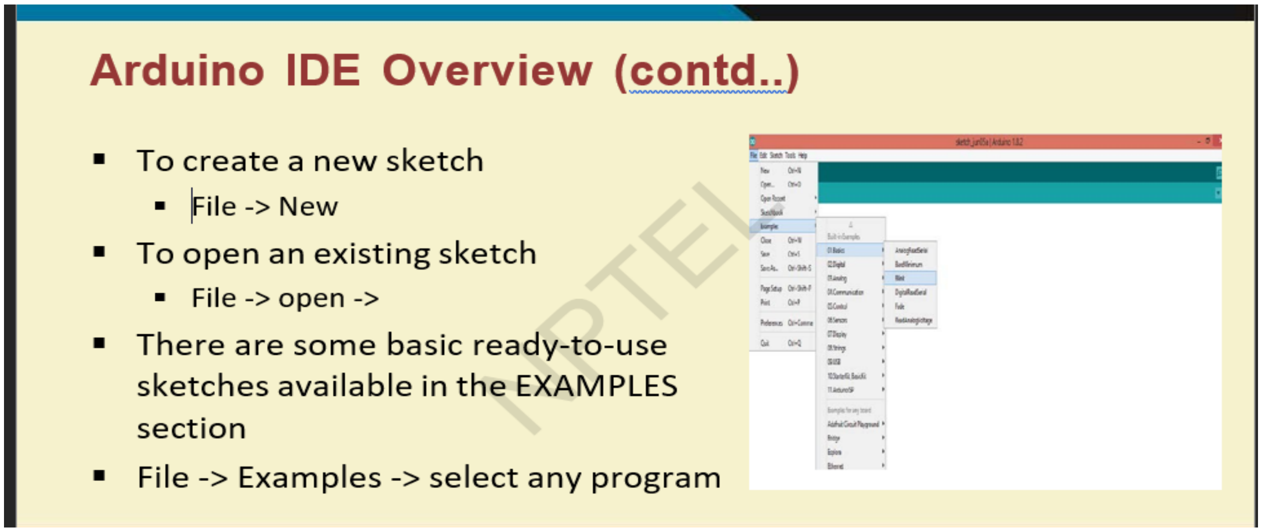
1. Power the board by connecting it to a PC via USB cable
2. Launch the Arduino IDE
3. Set the board type and the port for the board
4. TOOLS -> BOARD -> select your board
5. TOOLS -> PORT -> select your port

Programs written using Arduino Software (IDE) are called **sketches.**

1. These sketches are written in the text editor and are saved with the **file extension. ino**

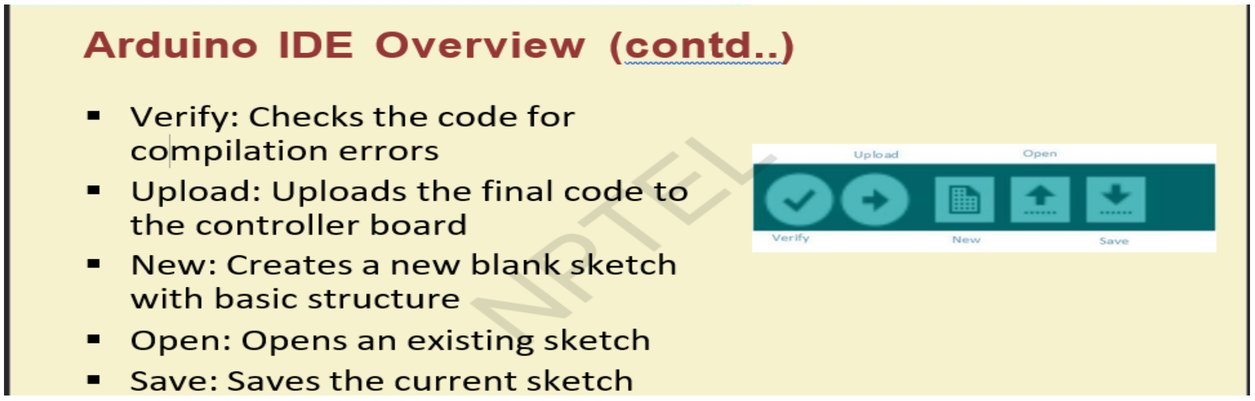
2. The editor has features for cutting/pasting and for searching/replacing text.
3.The message area gives feedback while saving and exporting and also displays errors.

4.The console displays text output by the Arduino Software (IDE), including complete error messages and other information.
5.The bottom righthand corner of the window displays the configured board and serial port.
6.The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.



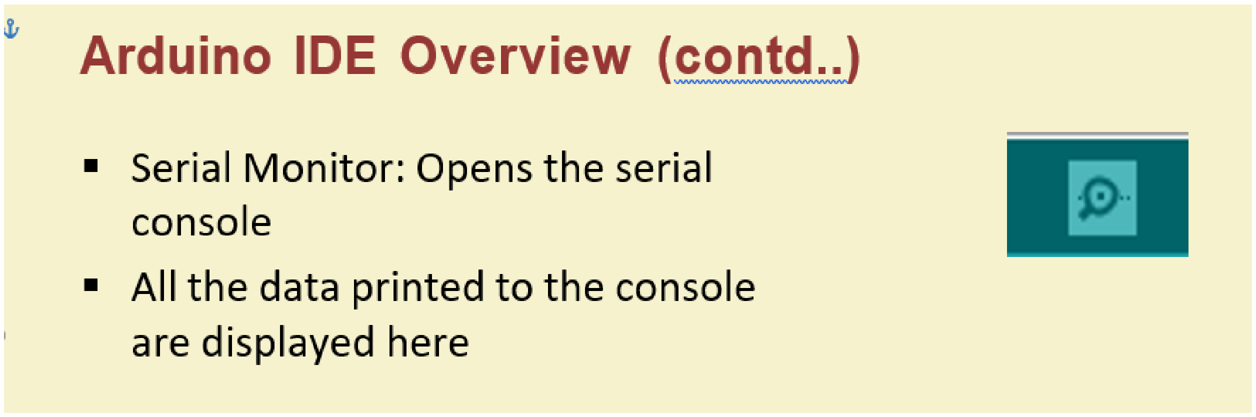
Arduino IDE Overview (contd..)

- To create a new sketch
  - File -> New
- To open an existing sketch
  - File -> open ->
- There are some basic ready-to-use sketches available in the EXAMPLES section
- File -> Examples -> select any program



Arduino IDE Overview (contd..)

- Verify: Checks the code for compilation errors
- Upload: Uploads the final code to the controller board
- New: Creates a new blank sketch with basic structure
- Open: Opens an existing sketch
- Save: Saves the current sketch



Arduino IDE Overview (contd..)

- Serial Monitor: Opens the serial console
- All the data printed to the console are displayed here

The Arduino Programming Language and Built-in Functions

Program lifecycle/structure of aurdino program

Structure

Arduino programs can be divided in three main parts: **Structure, Values** (variables and constants), and **Functions**. In this tutorial, we will learn about the Arduino software program, step by step, and how we can write the program without any syntax or compilation error.

Let us start with the **Structure**. Software structure consist of two main functions −

- Setup( ) function
- Loop( ) function

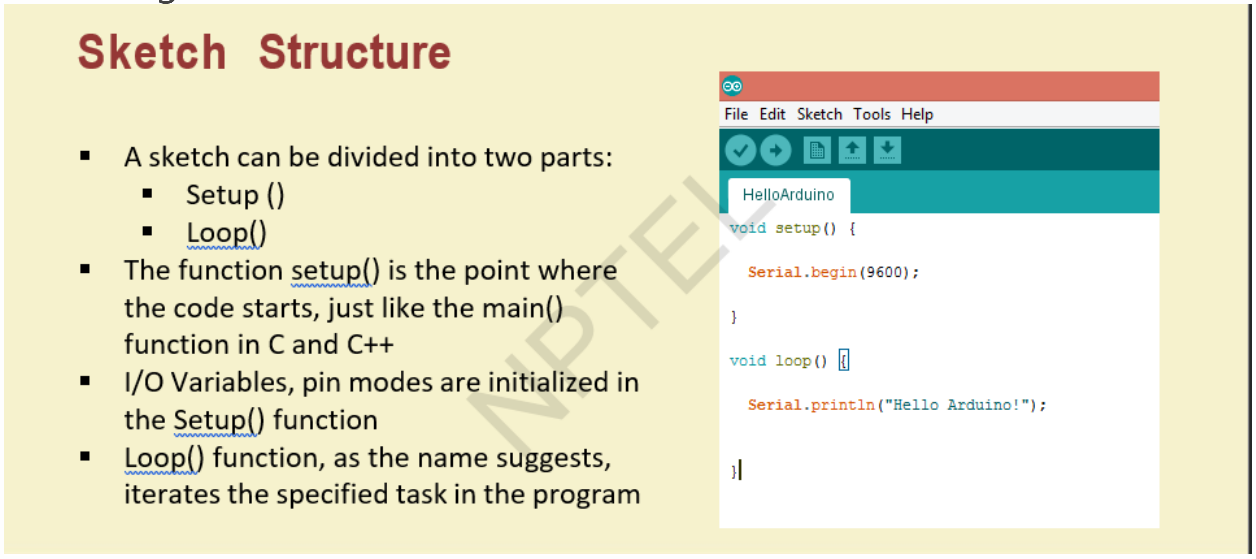
```
Void setup ( ) {

}
```

- **PURPOSE** − The **setup()** function is called when a sketch starts. Use it to initialize the variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board.
- **INPUT** − -
- **OUTPUT** − -
- **RETURN** − -

```
Void Loop ( ) {

}
```

- **PURPOSE −** After creating a **setup()** function, which initializes and sets the initial values, the **loop()** function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.
- **INPUT** − -
- **OUTPUT** − -
- **RETURN** − -

- `setup()` this function is called once, when the program starts, and when the Arduino is shut down and restarted.
- `loop()` this function is repeatedly called while the Arduino program is running.



Sketch Structure

- A sketch can be divided into two parts:
  - Setup ()
  - Loop()
- The function setup() is the point where the code starts, just like the main() function in C and C++
- I/O Variables, pin modes are initialized in the Setup() function
- Loop() function, as the name suggests, iterates the specified task in the program

File Edit Sketch Tools Help
HelloArduino

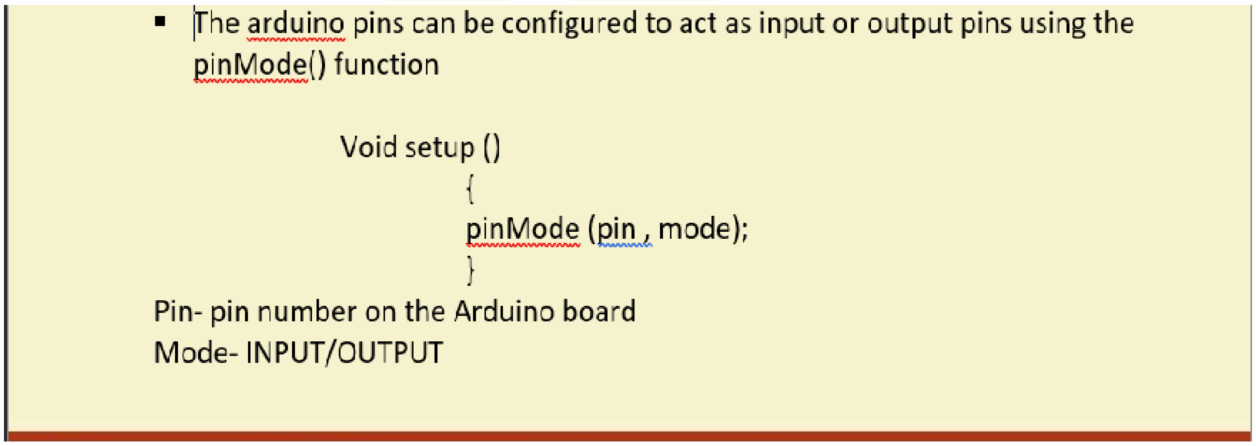
```
void setup() {

  Serial.begin(9600);

}

void loop() {

  Serial.println("Hello Arduino!");


}
```

Handling I/O

The following functions help with handling input and output from your Arduino device.

1.Digital I/O

- **`pinMode()` sets a pin to be an input, or an output. You pass the pin number and the `INPUT` or `OUTPUT` value as parameters.**



- The arduino pins can be configured to act as input or output pins using the pinMode() function

```
Void setup ()
          {
          pinMode (pin , mode);
          }
```

Pin- pin number on the Arduino board
Mode- INPUT/OUTPUT

- `digitalRead()` reads the value from a digital pin. Accepts a pin number as a parameter, and returns the `HIGH` or `LOW` constant.
- `digitalWrite()` writes a `HIGH` or `LOW` value to a digital output pin. You pass the pin number and `HIGH` or `LOW` as parameters.

- `pulseIn()` reads a digital pulse from `LOW` to `HIGH` and then to `LOW` again, or from `HIGH` to `LOW` and to `HIGH` again on a pin. The program will block until the pulse is detected. You specify the pin number and the kind of pulse you want to detect (LHL or HLH). You can specify an optional timeout to stop waiting for that pulse.
- `pulseInLong()` is same as `pulseIn()`, except it is implemented differently and it can't be used if interrupts are turned off. Interrupts are commonly turned off to get a more accurate result.
- `shiftIn()` reads a byte of data one bit at a time from a pin.
- `shiftOut()` writes a byte of data one bit at a time to a pin.
- `tone()` sends a square wave on a pin, used for buzzers/speakers to play tones. You can specify the pin, and the frequency. It works on both digital and analog pins.

- `noTone()` stops the `tone()` generated wave on a pin.

2.Analog I/O

- `analogRead()` reads the value from an analog pin.
- `analogReference()` configures the value used for the top input range in the analog input, by default 5V in 5V boards and 3.3V in 3.3V boards.
- `analogWrite()` writes an analog value to a pin
- `analogReadResolution()` lets you change the default analog bits resolution for `analogRead()`, by default 10 bits. Only works on specific devices (Arduino Due, Zero and MKR)
- `analogWriteResolution()` lets you change the default analog bits resolution for `analogWrite()`, by default 10 bits. Only works on specific devices (Arduino Due, Zero and MKR)

Time functions

- **`delay()`** pauses the program for a number of milliseconds specified as parameter
  - Delay() function is one of the most common time manipulation function usedto provide a delay of specified time. It accepts integer value (time in miliseconds)
-
- `delayMicroseconds()` pauses the program for a number of microseconds specified as parameter
- `micros()` the number of microseconds since the start of the program. Resets after ~70 minutes due to overflow

- `millis()` the number of milliseconds since the start of the program. Resets after ~50 days due to overflow

Math functions

- `abs()` the absolute value of a number
- `constrain()` constrains a number to be within a range, see usage
- `map()` re-maps a number from one range to another, see usage
- `max()` the maximum of two numbers
- `min()` the minimum of two numbers
- `pow()` the value of a number raised to a power
- `sq()` the square of a number
- `sqrt()` the square root of a number
- `cos()` the cosine of an angle
- `sin()` the sine of an angle
- `tan()` the tangent of an angle
  Note: there are more built-in mathematical functions if you need them, documented here.

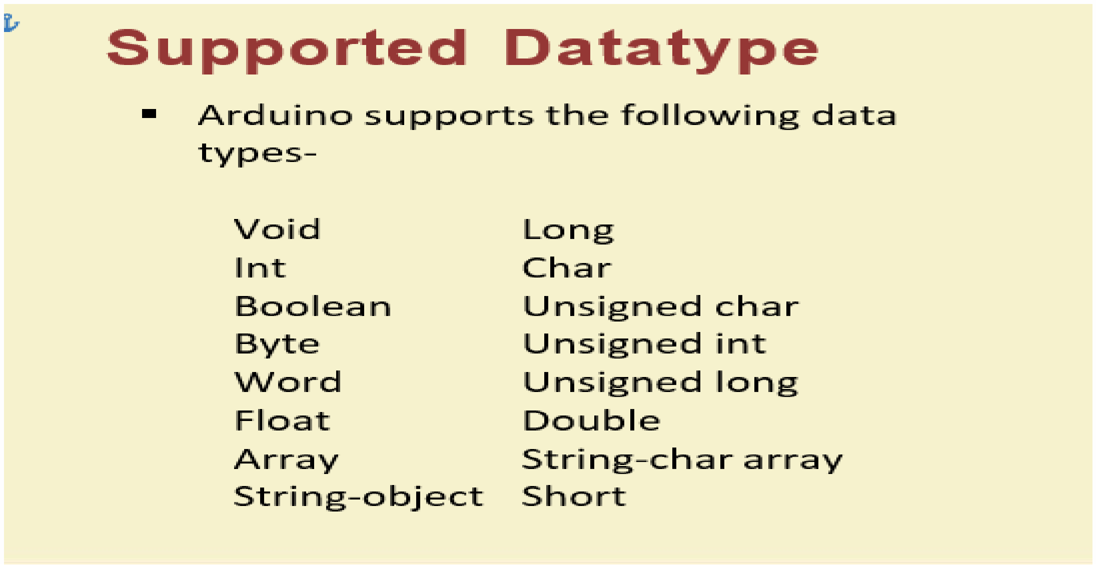
Working with alphanumeric characters

- `isAlpha()` checks if a char is alpha (a letter)
- `isAlphaNumeric()` checks if a char is alphanumeric (a letter or number)
- `isAscii()` checks if a char is an ASCII character
- `isControl()` checks if a char is a control character
- `isDigit()` checks if a char is a number
- `isGraph()` checks if a char is a printable ASCII character, and contains content (it is not a space, for example)

- `isHexadecimalDigit()` checks if a char is an hexadecimal digit (A-F 0-9)
- `isLowerCase()` checks if a char is a letter in lower case
- `isPrintable()` checks if a char is a printable ASCII character
- `isPunct()` checks if a char is a punctuation (a comma, a semicolon, an exclamation mark etc)
- `isSpace()` checks if a char is a space, form feed `\f`, newline `\n`, carriage return `\r`, horizontal tab `\t`, or vertical tab `\v`.
- `isUpperCase()` checks if a char is a letter in upper case
- `isWhitespace()` checks if a char is a space character or an horizontal tab `\t`

Data types :

Data types refers to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in the storage and how the bit pattern stored is interpreted.

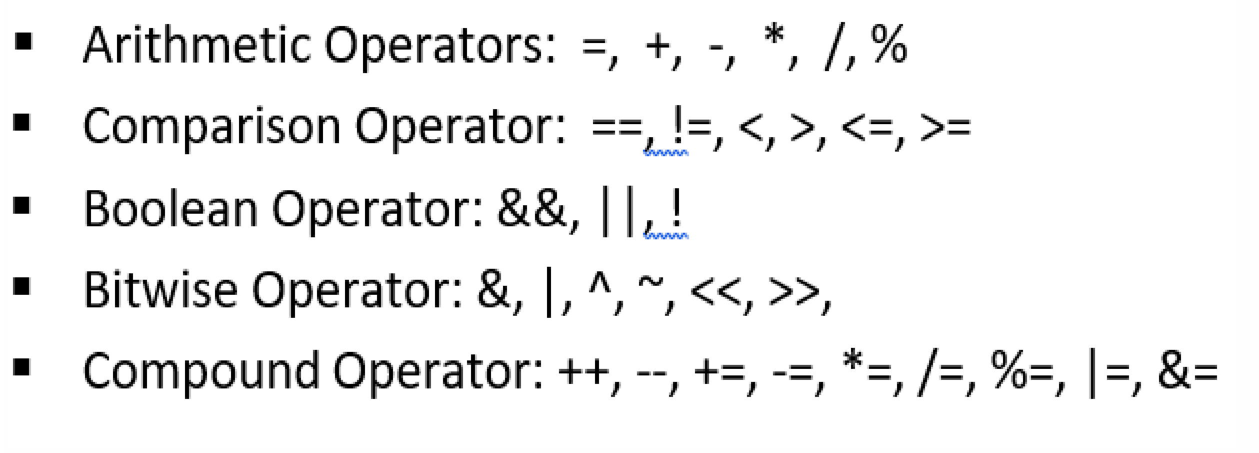


Supported Datatype

- Arduino supports the following data types-

Void	Long
Int	Char
Boolean	Unsigned char
Byte	Unsigned int
Word	Unsigned long
Float	Double
Array	String-char array
String-object	Short

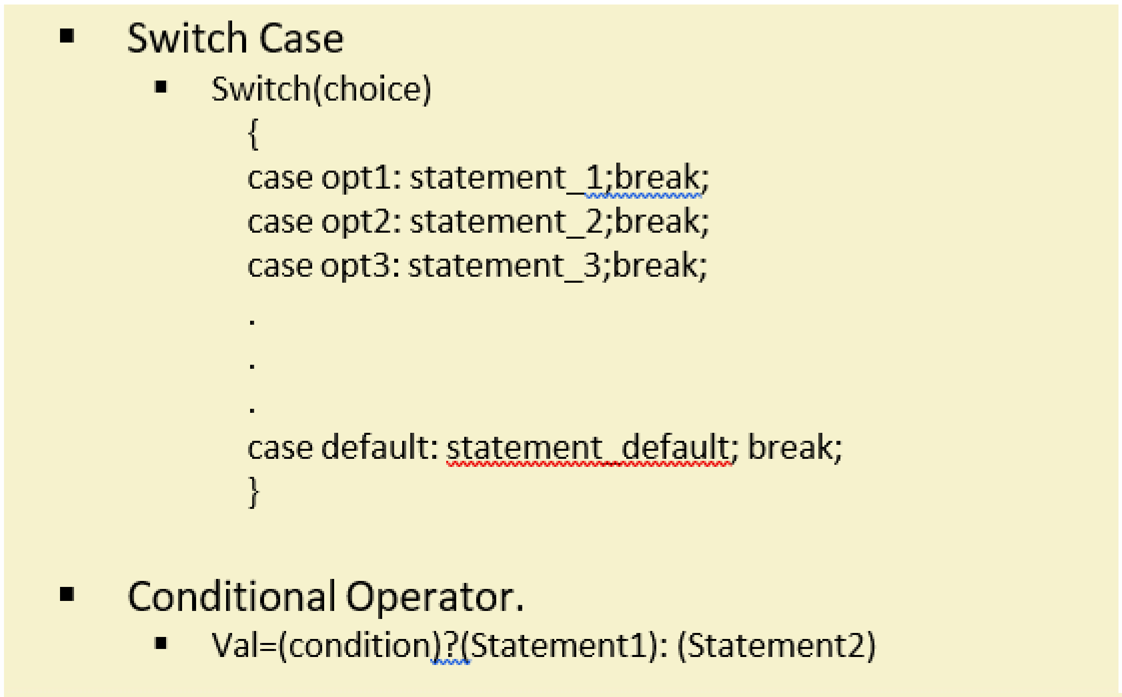
Operators : An operator is a symbol that tells the compiler to perform specific mathematical or logical functions



- Arithmetic Operators: =, +, -, *, /, %
- Comparison Operator: ==, !=, <, >, <=, >=
- Boolean Operator: &&, ||, !
- Bitwise Operator: &, |, ^, ~, <<, >>,
- Compound Operator: ++, --, +=, -=, *=, /=, %=, |=, &=

Control statement :

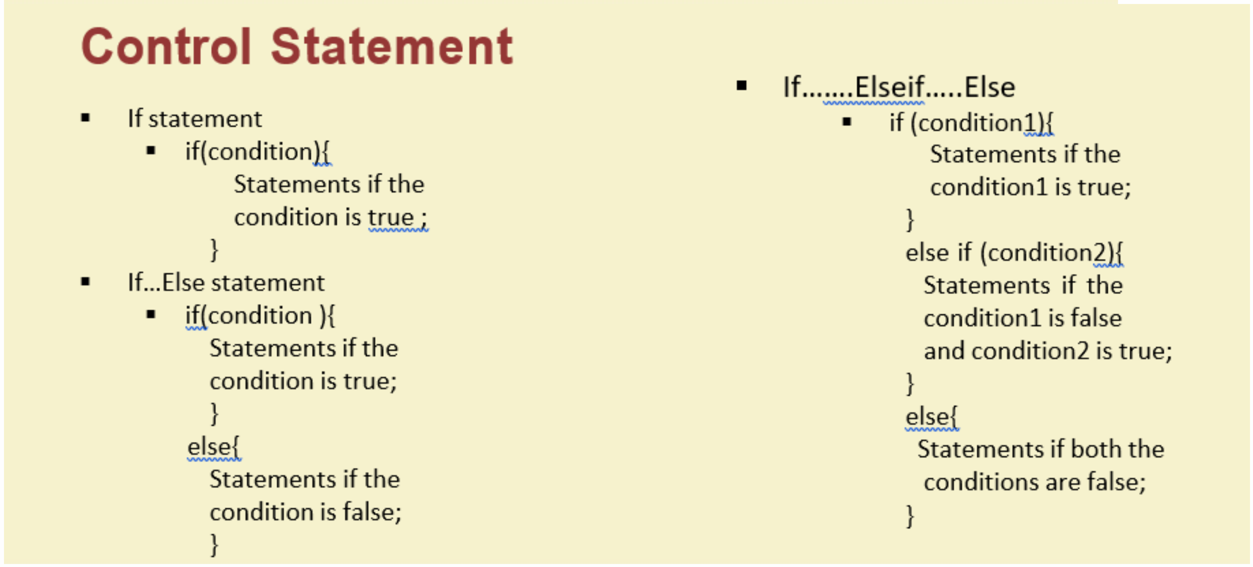
Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program. It should be along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.



- Switch Case
  - Switch(choice)

```
{
case opt1: statement_1;break;
case opt2: statement_2;break;
case opt3: statement_3;break;
.
.
.
case default: statement_default; break;
}
```

- Conditional Operator.
  - Val=(condition)?(Statement1): (Statement2)



Control Statement

- If statement
  - if(condition){

```
    Statements if the
    condition is true ;
}
```

- If...Else statement
  - if(condition){

```
    Statements if the
    condition is true;
}
else{
    Statements if the
    condition is false;
}
```

- If…….Elseif…..Else
  - if (condition1){

```
    Statements if the
    condition1 is true;
}
else if (condition2){
    Statements if the
    condition1 is false
    and condition2 is true;
}
else{
    Statements if both the
    conditions are false;
}
```

LOOP

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages –

- For loop
  - for(initialization; condition; increment){
    Statement till the condition is true;
    }
- While loop
  - while(condition){
    Statement till the condition is true;
    }
- Do... While loop
  - do{
      Statement till the condition is true;
    }while(condition);

- Nested loop: Calling a loop inside another loop

- Infinite loop: Condition of the loop is always true, the loop will never terminate

Arrays

- Collection of elements having homogenous datatype that are stored in adjacent memory location.
- The conventional starting index is 0.
- Declaration of array:

  <Datatype> array_name[size];

  Ex: int arre[5];

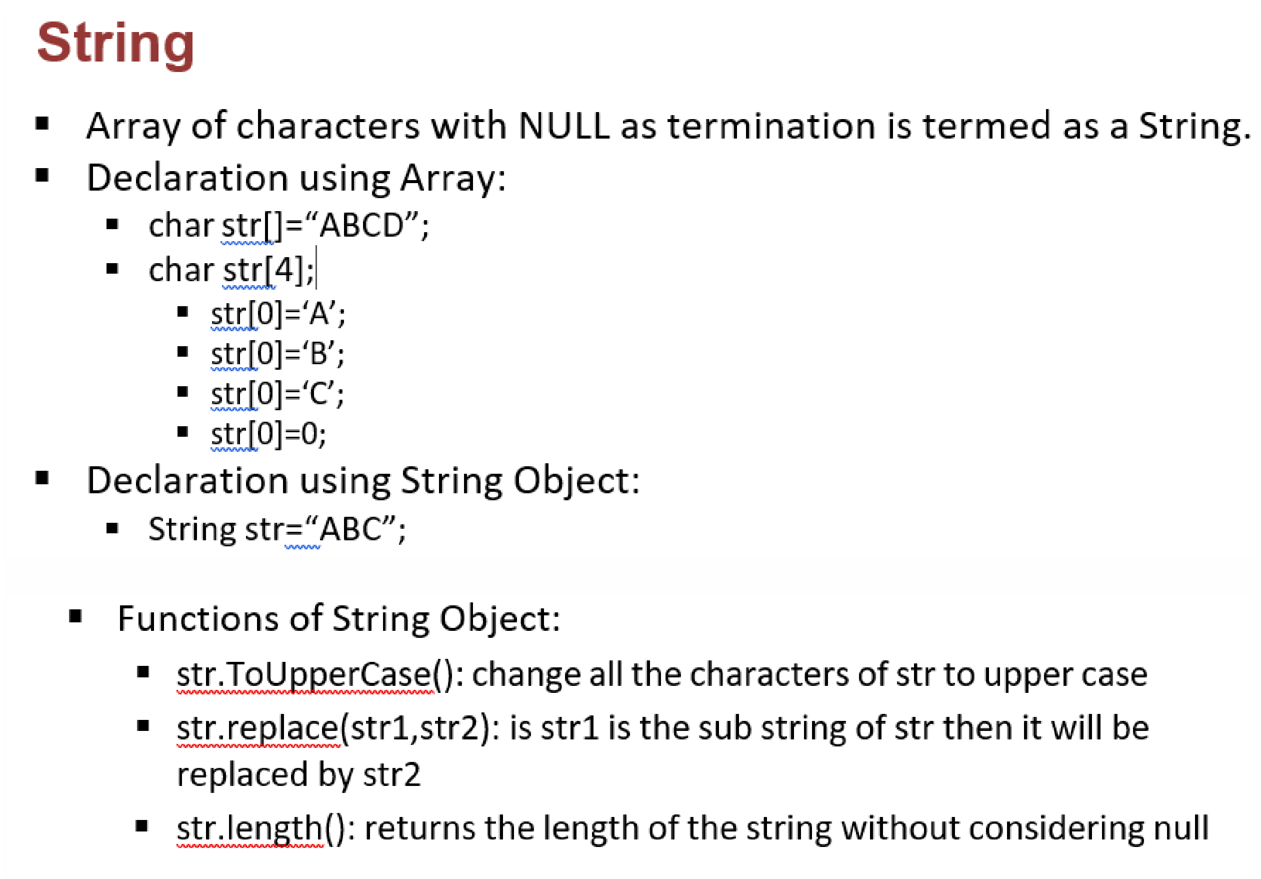
- Alternative Declaration:

  int arre[]={0,1,2,3,4};

  int arre[5]={0,1,2};
- Multi-dimentional array Declaration:

  <Datatype> array_name[n1] [n2][n3]....;

  Ex: int arre[row][col][height];



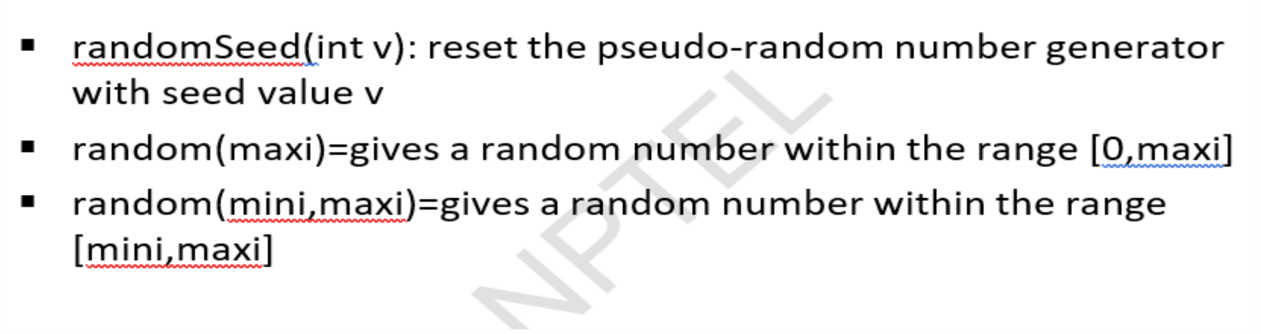
String

- Array of characters with NULL as termination is termed as a String.
- Declaration using Array:
  - char str[]="ABCD";
  - char str[4];
    - str[0]='A';
    - str[0]='B';
    - str[0]='C';
    - str[0]=0;
- Declaration using String Object:
  - String str="ABC";
- Functions of String Object:
  - str.ToUpperCase(): change all the characters of str to upper case
  - str.replace(str1,str2): is str1 is the sub string of str then it will be replaced by str2
  - str.length(): returns the length of the string without considering null

Random numbers generation

To generate random numbers, you can use Arduino random number functions. We have two functions −

- randomSeed(seed)
- random()



- randomSeed(int v): reset the pseudo-random number generator with seed value v
- random(maxi)=gives a random number within the range [0,maxi]
- random(mini,maxi)=gives a random number within the range [mini,maxi]

Working with bits and bytes

- `bit()` computes the value of a bit (0 = 1, 1 = 2, 2 = 4, 3 = 8...)
- `bitClear()` clear (sets to 0) a bit of a numeric variable. Accepts a number, and the number of the bit starting from the right
- `bitRead()` read a bit of a number. Accepts a number, and the number of the bit starting from the right
- `bitSet()` sets to 1 a bit of a number. Accepts a number, and the number of the bit starting from the right
- `bitWrite()` write 1 or 0 to a specific bit of a number Accepts a number, the number of the bit starting from the right, and the value to write (0 or 1)
- `highByte()` get the high-order (leftmost) byte of a word variable (which has 2 bytes)
- `lowByte()` get the low-order (rightmost) byte of a word variable (which has 2 bytes)

Interrupts

Interrupts stop the current work of Arduino such that some other work can be done.

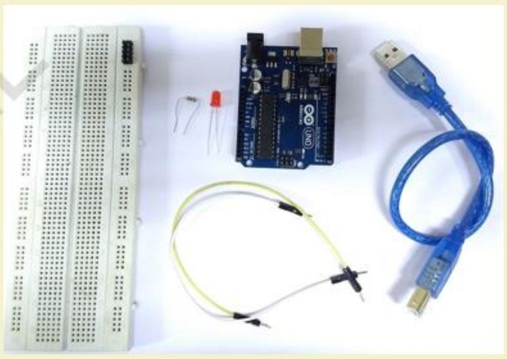
- `noInterrupts()` disables interrupts
- `interrupts()` re-enables interrupts after they've been disabled
- `attachInterrupt()` allow a digital input pin to be an interrupt. Different boards have different allowed pins, check the official docs.
- `detachInterrupt()` disables an interrupt enabled using `attachInterrupt()`

PROGRAMS WITH AURDINO

1.

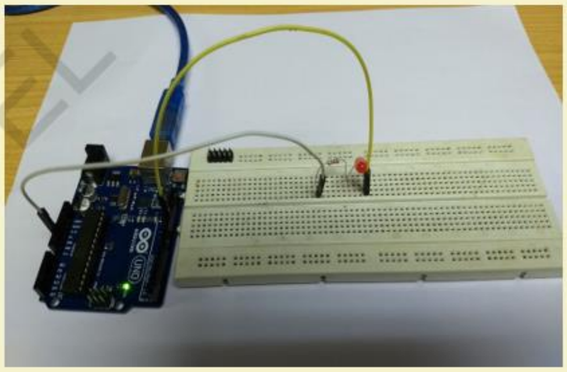
Example- Blinking LED

- Requirement:
  - Arduino controller board, USB connector, Bread board, LED, 1.4Kohm resistor, connecting wires, Arduino IDE
- Connect the LED to the Arduino using the Bread board and the connecting wires
- Connect the Arduino board to the PC using the USB connector
- Select the board type and port
- Write the sketch in the editor, verify and upload.



Example- Blink (contd..)

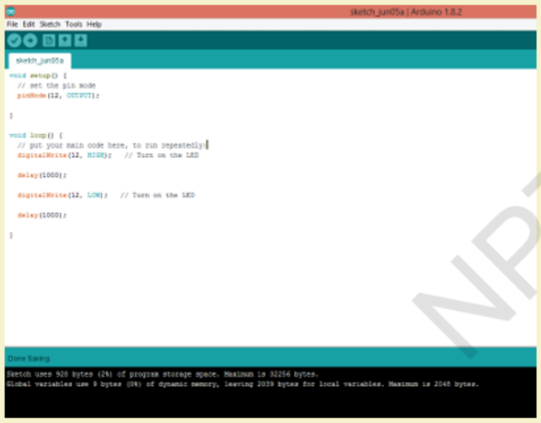
Connect the positive terminal of the LED to digital pin 12 and the negative terminal to the ground pin (GND) of Arduino Board



Example- Blink (contd..) image setup

```
void setup() {
   pinMode(12, OUTPUT); // set the pin mode
}
void loop() {
   digitalWrite(12, HIGH);   // Turn on the LED
   delay(1000);
   digitalWrite(12, LOW);  //Turn of the LED
   delay(1000);
}
```

Example- Blink (contd..)



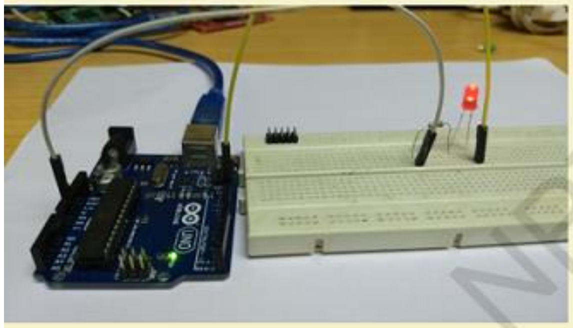
```
void setup() {
  // set the pin mode
  pinMode(12, OUTPUT);

}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(12, HIGH);   // Turn on the LED

  delay(1000);

  digitalWrite(12, LOW);   // Turn on the LED

  delay(1000);

}
```

```
Sketch uses 928 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.
```

Set the pin mode as output which is connected to the led, pin 12 in this case.

Use digitalWrite() function to set the output as HIGH and LOW

Delay() function is used to specify the delay between HIGH-LOW transition of the output



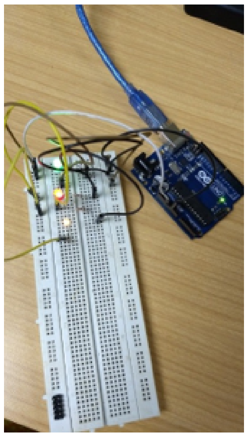
- Connect he board to the PC
- Set the port and board type
- Verify the code and upload, notice the TX – RX led in the board starts flashing as the code is uploaded.

2

Example: Traffic Control System

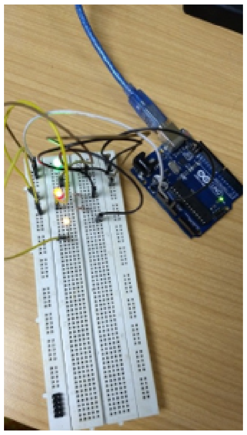
Requirement:

- Arduino Board
- 3 different color LEDs
- 330 Ohm resistors
- Jumper wires



Connection:

- Connect the positive terminals of the LEDs to the respective digital output pins in the board, assigned in the code.
- Connect the negative terminals of the LEDs to the ground



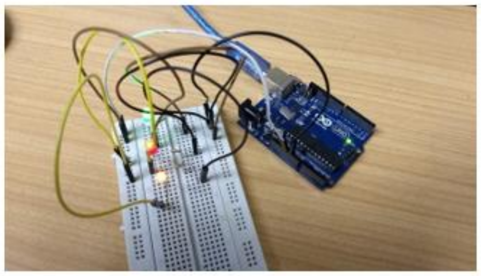
Sketch

```
 //LED pins
  int r =  2;
  int g = 3;
  int y = 4;
  void setup()
  {
        Serial.begin(9600);
        pinMode(r, OUTPUT); digitalWrite(r,LOW);
        pinMode(g, OUTPUT); digitalWrite(g,LOW);
        pinMode(y , OUTPUT); digitalWrite(y, LOW);
  }
```

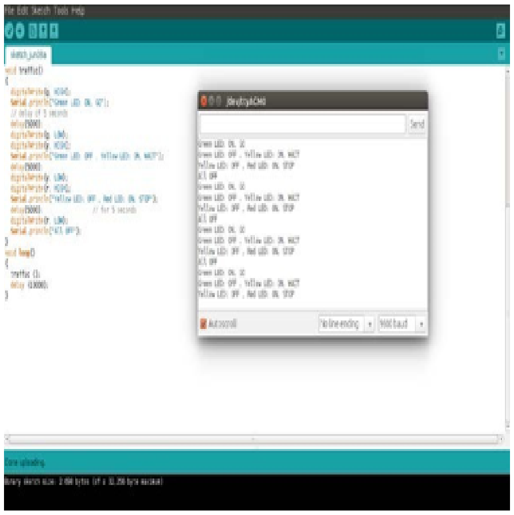
```
void traffic()
{
        digitalWrite(g, HIGH);
        Serial.println("Green LED: ON, GO");
        // delay of 5 seconds
        delay(5000);
        digitalWrite(g, LOW);
        digitalWrite(y, HIGH);
        Serial.println("Green LED: OFF ; Yellow LED: ON, WAIT");
        delay(5000);


                        digitalWrite(y, LOW);
                        digitalWrite(r, HIGH);
                        Serial.println("Yellow LED: OFF ; Red LED: ON, STOP");
                        delay(5000);                // for 5 seconds
                        digitalWrite(r, LOW);
                        Serial.println("All OFF");
                }

           void loop()
           {
                traffic  ();
                delay (10000);
           }
```

Output:

- Initially, all the LEDs are turned off
- The LEDs are turned on one at a time with a delay of 5 seconds
- The message is displayed accordingly
- Figure showing all the LEDs turned on

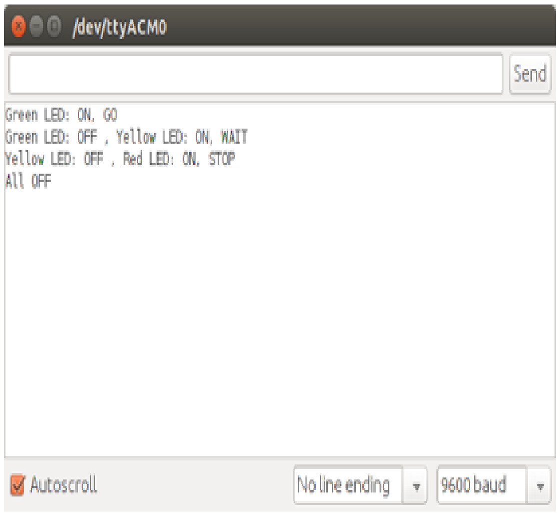


Output



```
void traffic()
{
  digitalWrite(g, HIGH);
  Serial.println("Green LED: ON, GO");
  // delay of 5 seconds
  delay(5000);
  digitalWrite(g, LOW);
  digitalWrite(y, HIGH);
  Serial.println("Green LED: OFF , Yellow LED: ON, WAIT");
  delay(5000);
  digitalWrite(y, LOW);
  digitalWrite(r, HIGH);
  Serial.println("Yellow LED: OFF , Red LED: ON, STOP");
  delay(5000);                // for 5 seconds
  digitalWrite(r, LOW);
  Serial.println("All OFF");
}
void loop()
{
  traffic ();
  delay (10000);
}
```

```
Green LED: ON, GO
Green LED: OFF , Yellow LED: ON, WAIT
Yellow LED: OFF , Red LED: ON, STOP
All OFF
Green LED: ON, GO
Green LED: OFF , Yellow LED: ON, WAIT
Yellow LED: OFF , Red LED: ON, STOP
All OFF
Green LED: ON, GO
Green LED: OFF , Yellow LED: ON, WAIT
Yellow LED: OFF , Red LED: ON, STOP
All OFF
Green LED: ON, GO
Green LED: OFF , Yellow LED: ON, WAIT
Yellow LED: OFF , Red LED: ON, STOP
```



```
Green LED: ON, GO
Green LED: OFF , Yellow LED: ON, WAIT
Yellow LED: OFF , Red LED: ON, STOP
All OFF
```

INTEGRATING SENSORS AND ACTUATORS WITH AURDINO

Sensors

- Basic electronic Device
- Convert a physical quantity/ measurements into electrical signals
- Can be analog or digital

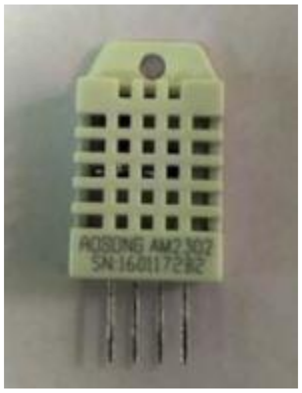
Types of Sensors

Some commonly used sensors :

- Temperature
- Humidity
- Compass
- Light
- Sound
- Accelerometer

Sensor Interface with Arduino

Digital Humidity and Temperature Sensor (DHT)



PIN 1,2,3,4 (from left to right)

- PIN 1-3.3V-5V Power supply
- PIN 2- Data
- PIN 3-Null
- PIN 4- Ground

DHT Sensor Library

- Arduino supports a special library for the DHT11 and DHT22 sensors
- Provides function to read the temperature and humidity values from the data pin

dht.readHumidity()

dht.readTemperature()

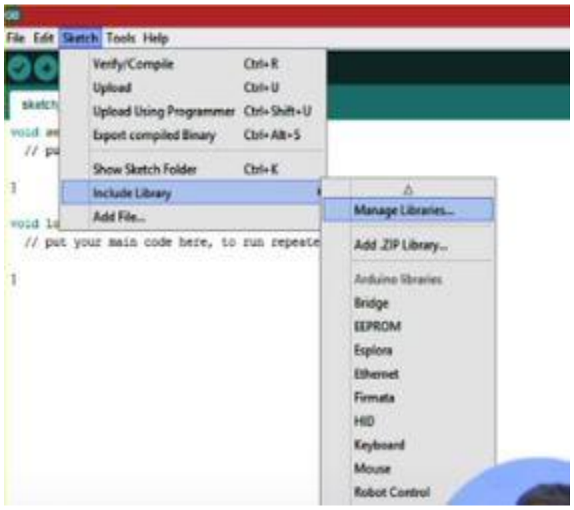
Connection

- Connect pin 1 of the DHT to the 3.3 V supply pin in the board
- Data pin (pin 2) can be connected to any digital pin, here 12
- Connect pin 4 to the ground (GND) pin of the board

Sketch: DHT_Sensor

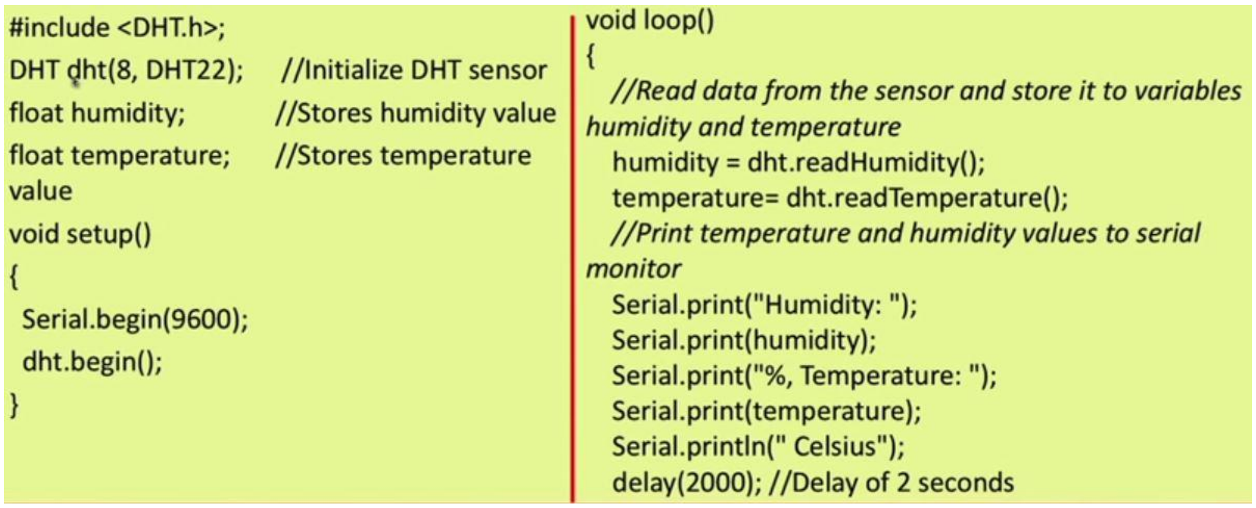
Install the DHT Sensor Library

- Go to sketch -> Include Library -> Manage Library



```
// put your main code here, to run repeate
```

- Search for DHT Sensor
- Select the "DHT sensor library" and install it



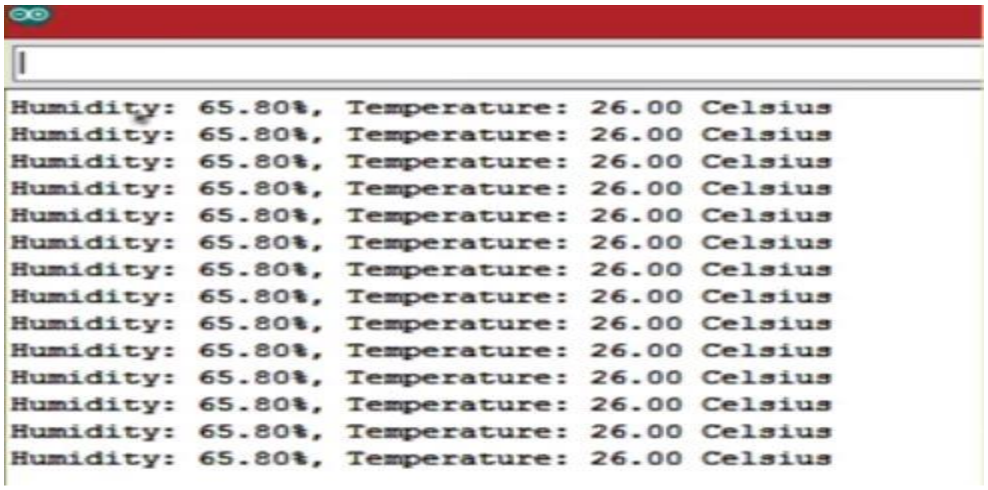
```
#include <DHT.h>;
DHT dht(8, DHT22);     //Initialize DHT sensor
float humidity;        //Stores humidity value
float temperature;     //Stores temperature
value
void setup()
{
 Serial.begin(9600);
 dht.begin();
}
```

```
void loop()
{
  //Read data from the sensor and store it to variables
humidity and temperature
  humidity = dht.readHumidity();
  temperature= dht.readTemperature();
  //Print temperature and humidity values to serial
monitor
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print("%, Temperature: ");
  Serial.print(temperature);
  Serial.println(" Celsius");
  delay(2000); //Delay of 2 seconds
```

- Connect the board to the PC
- Set the port and board type
- Verify and upload the cod

OutPut

The readings are printed at a delay of 2 seconds as specified by the delay() function



```
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
Humidity: 65.80%, Temperature: 26.00 Celsius
```

Acutators

Types of Motor acutators

1.Servo Motor

2.Stepper Motor

3.Hydraulic Motor

4.Ac motor and so on

Servo Motor :

Servo Motor is a high Precision motor which provides rotary motion 0 to 180 degree

The 3 wires in the servo motor are

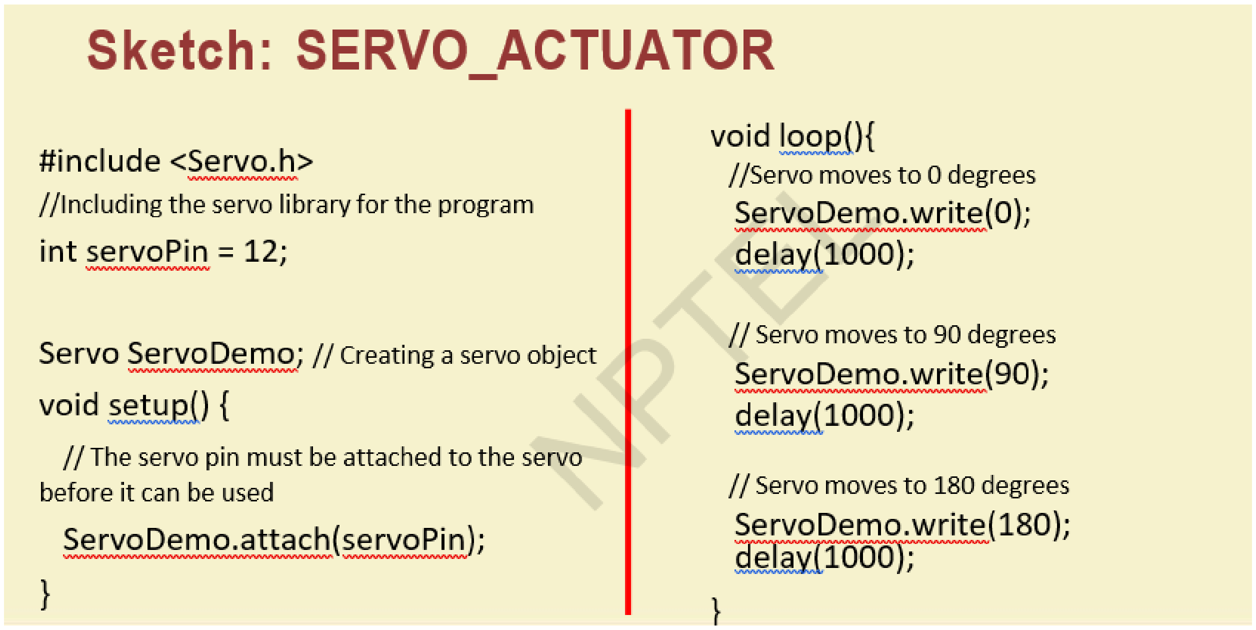
Black or the darkest one is ground Red is power supply and Yellow is for signal pin

Servo Library on Aurdino

Aurdino provides separate library SERVO to operate the servo motor .

Create an instnace of servo to use it in the sketch .

Syntax : Servo myservo



Sketch: SERVO_ACTUATOR

```
#include <Servo.h>
//Including the servo library for the program
int servoPin = 12;

Servo ServoDemo; // Creating a servo object
void setup() {
  // The servo pin must be attached to the servo
before it can be used
  ServoDemo.attach(servoPin);
}
```

```
void loop(){
  //Servo moves to 0 degrees
  ServoDemo.write(0);
  delay(1000);

  // Servo moves to 90 degrees
  ServoDemo.write(90);
  delay(1000);

  // Servo moves to 180 degrees
  ServoDemo.write(180);
  delay(1000);
}
```

Here ServoDemo is the instance of Servo.

Write() takes the degree value and rotates the motor accordingly.

Output :

The motor turns 0, 90 and 180 degrees with a delay of 1 second each.