Definition of IoT(Internet of Things):

The IoT can be defined in two ways based on

- existing Technology
- Infrastructure

Definition of IoT based on existing technology:

IoT is a new revolution to the internet due to the advancement in sensor networks, mobile devices, wireless communication, networking and cloud technologies.

Definition of IoT based on infrastructure:

IoT is a dynamic global network infrastructure of physical and virtual objects having unique identities, which are embedded with software, sensors, actuators, electronic and network connectivity to facilitate intelligent applications by collecting and exchanging data.

Goal of IoT:

The main goal of IoT is to configure, control and network the devices or things, to internet, which are traditionally not associated with the internet i.e thermostats, utility meters, a Bluetooth connected headset, irrigation pumps and sensors or control circuits for an electric car’s engine that make energy, logistics, industrial control, retail, agriculture and many other domain smarter.

Characteristics of IoT:

Various characteristics of IoT are:

- Dynamic and self-adapting
- Self-configuring
- Interoperable Communication protocols
- Unique identity
- Integrated into information network

Dynamic and self-adapting:

The IoT devices can dynamically adapt with sensed environment, their operating conditions, and user’s context and take actions accordingly. For ex: Surveillance System.

Self-configuring:

I. IoT devices can be able to upgrade the software with minimal intervention of user, whenever they are connected to the internet.
II. They can also setup the network i.e a new device can be easily added to the existing network. For ex: Whenever there will be free wifi access one device can be connected easily.

Interoperable Communication:

IoT allows different devices (different in architecture) to communicate with each other as well as with different network. For ex: MI Phone is able to control the smart AC and smart TV of different manufacturer.

Unique identities:

I. The devices which are connected to the internet have unique identities i.e IP address through which they can be identified throughout the network.
II. The IoT devices have intelligent interfaces which allow communicating with users. It adapts to the environmental contexts.
III. It also allows the user to query the devices, monitor their status, and control them remotely, in association with the control, configuration and management infrastructure.

Integrated into information network:

I. The IoT devices are connected to the network to share some information with other connected devices. The devices can be discovered dynamically in the network by other devices. For ex. If a device has wifi connectivity then that will be shown to other nearby devices having wifi connectivity.
II. The devices ssid will be visible though out the network. Due to these things the network is also called as information network.
III. The IoT devices become smarter due to the collective intelligence of the individual devices in collaboration with the information network. For Ex: weather monitoring system. Here the information collected from different monitoring nodes (sensors, arduino devices) can be aggregated and analysed to predict the weather.

Physical Design of IoT:

Things in IoT:

I. IoT i.e Internet of things, where things refer to the IoT devices which have unique identities and can perform remote sensing, actuating and monitoring capabilities (ex: combination of sensors, actuators, Arduino, relay, non IoT devices).
II. The IoT devices can share information with as well as collect information from other connected devices and applications (directly and indirectly).
III. They can process the data locally or in the cloud to find greater insights and put them into action based on temporal and space constraints (i.e space memory, processing capabilities, communication latencies and speeds and deadlines).
IV. IoT devices can be of varied types. For ex: wearable sensors, smart watches, LED lights, automobiles and industrial machines.

Logical design of IoT:

Logical design of IoT refers to an abstract representation of entities and the processes without going into the details of the implementations. The logical design includes functional block of IoT and the communication APIs.

IoT functional Block: (Refer book for the diagram):

The functional block of the system provides the capabilities for identification, sensing, actuation, communication and management. Various components of IoT functional block are as follows.

- Device
- Communications
- Services
- Management
- Security
- Application

Device:

I. IoT i.e Internet of things, where things refer to the IoT devices which have unique identities and can perform remote sensing, actuating and monitoring capabilities (ex: combination of sensors, actuators, Arduino, relay, non IoT devices).
II. The IoT devices can share information with as well as collect information from other connected devices and applications (directly and indirectly).
III. They can process the data locally or in the cloud to find greater insights and put them into action based on temporal and space constraints (i.e space memory, processing capabilities, communication latencies and speeds and deadlines).
IV. IoT devices can be of varied types. For ex: wearable sensors, smart watches, LED lights, automobiles and industrial machines.

Communications:

It refers to various communication protocols which allows different devices to communicate with each other by sharing some information. It also allows interoperability among different devices.

Services:

IoT system provides various services such as device monitoring, device control services, data publishing services, device discovery services.

Management:

Various management functions to govern the IoT system.

Security:

It secures the IoT system by providing authentication, authorization, message and content integrity and data security.

Application:

I. IoT applications provide an interface that the users can use to control and monitor various aspects of the IoT system.
II. It also allows viewing the system status and view or analysing the processed data.

IoT communication model:

To provide communication to various IoT devices, there are various communication models. Such as

- Request-Response communication model
- Publisher-subscriber communication model
- Push-pull communication model
- Exclusive-pair communication model

Request-response communication model: (Refer book for the diagram)

I. In this communication model client will send request to server. Server will receive the request then decides how to respond, fetches the data according to client's request, retrieves resource representations, prepares the response and then sends the response to client.
II. Request-response model is a stateless communication model i.e each request-response pair is independent of others. Ex: HTTPs requests to log in some secure sites like IRCTC.
III. Here only client can request to the server. i.e request is unidirectional but data transfer is bi-directional.
IV. It follows client server architecture. IoT devices act as client. Cloud act as server or local server serves as server.

Publish-Subscribe communication model: (Refer book for the diagram)

I. This communication model comprises of publisher, broker, and consumer.
II. Publishers are the source of data i.e the devices or applications which generate data. Publisher publishes the data to the requested topic by the client or consumer into the server.
III. Broker manages the topics in server. Broker receives the data from the client or publisher and forwards the message to topics on the topics subscribed by the clients.
IV. Consumer or clients (applications that process the data) subscribe to various topics present inside server. When the publisher publishes the data to the requested topics they will receive it from the broker.
V. The publisher doesn't know about the client, but knows about the broker. Likewise client does not know about the publisher but knows about the broker.
VI. It also follows client-server architecture.

VII. Ex1-when one user subscribe to any channel in you-tube. Here let I am a client and I subscribe to Remo's dance channel. Remo may directly create his own channel in You-tube and sends the recorded video to the channel. Here Remo act as publisher, you tube as server and broker. Channel as topic and me as subscriber.
Ex2- sometimes what happened the publisher doesn't create its own channel rather they send the data to an intermediate then they send the data to the server and from that to the subscribed user. That intermediate can be called as broker. There you -tube is called as server. Ex. Cheers is a broker which manages some of the web series. Here the web series are topic. You-tube is the server.

Push-Pull communication model: (Refer book for the diagram)

I. Push-pull is a communication model in which the data producers push the data to the queues and the consumers pull the data from the queues.
II. Producers do not need to be aware of the consumers.
III. Queues act as buffer which helps in situations when there is a mismatch between the rate at which the producers push the data and the rate at which the consumers pull the data.

Exclusive pair communication model: (refer book for the diagram)

I. Exclusive pair is a bi-directional, full duplex communication model.
II. It uses state-full protocols i.e we don't need to log in again and again.
III. It uses a single http connection i.e persistent connection between the client and the server i.e once the connection is set up it remains until the client sends the request to close the connection.
IV. It also supports client-server architecture.
V. It is a stateful communication model and the server is aware about all the open connection.

IoT communication APIs:

What is an API: Application programming interface is a set of requirements that govern how one application can talk to another. API's do all these things by exposing some of program's internal functions to the outside world in a limited fashion.

Mainly two types of communication APIs are used in IoT. Those are as follows

- REST based communication API
- WebSocket based communication API

REST based communication API:

I. REST: Representational State Transfer.
II. It helps to design web services and web APIs that focus on a system's resource and how resource states are addressed and transferred.

III. It follows Request-response communication model and unidirectional communication for request. The clients send request to URIs using methods defined by the HTTP protocols (GET, PUT, POST, DELETE).

IV. RESTful web service is a "web API" implemented using HTTP and REST principle. RESTful web service is a collection of resources which are represented by URIs.

V. RESTful Web services can support various internet media types (JSON, XML).

JSON: Java script object notation (most popular web service).

XML: Extensible mark up language

VI. This communication API follows various constraints such as

- ✓ Client-server
- ✓ Stateless
- ✓ Cache-able
- ✓ Layered System
- ✓ Uniform interface
- ✓ Code on demand
- ✓ Scalability

Client-Server: The principle behind the client-server constraint is the separation of concern. Ex: The server is concerned about the storage part i.e storage of data and the client will not bother about it. The client should concern about the user interface and the server will not bother about it. Due to this type of separation client and server can be independently developed and updated.

Stateless: Each time the request from client to server must contain all the information necessary for understanding the request.

Cache-able: It requires that data within a response to request be implicitly or explicitly labelled as cach-able or non-cache-able. The data can be cached in client side so that it can be reused when requested for the next time in order to minimise the time. It will increase the efficiency and scalability.

Layered System: This constraint limits the behaviour of components i.e each component cannot see beyond the immediate layer with which they are interacting. Ex: client cannot say whether it is connected directly to the end server or to an intermediary. It improves scalability by allowing intermediaries to respond to requests instead of the end server without the client having to do anything different.

Uniform interface: The method of communication between a client and a server must be uniform

Code on demand: Servers can provide executable codes or scripts for clients to execute in their context.

Scalability: it supports both horizontal and vertical scalability. As it is stateless so scalability is easier to implement.

Refer book for the diagram of communication with REST APIs. Also for the methods (GET, PUT, POST, DELETE).

Web Socket-based Communication APIs:

1. Websocket API helps to design web services and web APIs.
2. It allows bi-directional, full-duplex communication between clients and servers.
3. It follows the exclusive pair communication model.
4. It supports stateful protocols. It does not require a new connection to be set up for each message to be sent. There is no overload for connection set up and termination request for each message. So Web socket API reduces the network traffic and latency.
5. It is suitable for IoT applications that have low latency or high throughput requirements.

Refer Book for Exclusive pair model used by web socket API.

Description of the Diagram:

In web socket communication first the client sets up connection with the server. This request is sent over the HTTP and the server interprets as an upgrade request (called Websocket handshake). If the server supports websocket protocol then only it will respond to this handshake. If the server supports then client and server can send message to each other in full-duplex mode.

Difference between REST and Websocket:

REST	Websocket
1. It supports Request-response communication model. 2. It supports stateless protocol. 3. It supports unidirectional communication between client and server as only client can send request to server and server only respond to the request. 4. It is half duplex. 5. It uses multiple TCP connection for each search over HTTP. 6. Since it does not store the request information so each time it needs to provide all the information while creating communication with server. For this reason header overhead increases. 7. It supports both horizontal and vertical scalability.	1. It supports Exclusive-pair communication model. 2. It supports stateful protocol. 3. It supports bidirectional communication between client and server i.e client and server both can request to each other. 4. It is full duplex. 5. It uses single TCP connection for search over HTTP. 6. Header overhead is less. 7. Vertical scaling is easier than horizontal scaling.

IoT enabling technologies:

The technologies which are cooperative with IoT those are as follows.

- Wireless sensor networks
- Cloud computing
- Big Data analytics
- Embedded systems
- Communication protocols

Wireless Sensor networks:

1. Wireless sensor network comprises of distributed devices, wireless sensors. These devices with sensors are used to monitor the environment and physical conditions. Since all the nodes are wireless so they communicate with each other through wifi or Bluetooth.
2. A WSN consists of several end nodes and routers as well as coordinator.
3. Sensors are attached with end nodes. Each router can also be called as end node.
4. Routers are responsible for routing the data packets from end nodes to the coordinator nodes. Coordinator node connects the WSN to the internet. The Coordinator node can be another arduino, raspberry pi or any other IoT DIY device.
5. It collects the data from all the nodes.
6. WSNs are enabled by wireless communication protocols such as IEEE802.15.4.
7. It can also be enabled by ESP 8266 and ZigBee.
8. ZigBEE Bluetooth module is based on IEEE802.15.4. It operates at 2.4 GHz frequency. It offers data rate up to 250 KB/s and ranges from 10 to 100 meters depending upon power output and environmental conditions. In WSN the devices can reconfigure themselves i.e new nodes can be added to the networks and software can be updated automatically whenever they will be connected to the internet.
9. Ex. of Wireless sensor network: Weather monitoring system, Indoor air quality monitoring, soil moisture monitoring, surveillance system, smart grids, machine prognosis and diagnosis.

Cloud Computing:

1. It is an emerging technology which enables on-demand network access to computing resources like network servers, storage, applications and services that can be rapidly provisioned and released.
2. On demand: we invoke cloud services only when we need them, they are not permanent part of IT infrastructure.
3. Pay as you go model: You pay for the cloud services when you use them, either for the short period of time or longer duration (for cloud based storage).
4. Cloud provides various services such as

i. IAAS: Infrastructure as a service
ii. PAAS: Platform as a service
iii. SAAS: Software as a service

IAAS:

Instead of creating a server room we will hire it from a cloud service provider. Here user will not use its local computer, storage and processing resources rather it will use virtual machine and virtual storage, servers, networking of third party. Here the client can deploy the OS (operating system), application of his own choice. User can start, stop, configure and manage the virtual machine instances and virtual storage.

PAAS:

User can develop and deploy applications. For ex. We are using various online editors to write codes like online arduino IDE, C IDE, APIs, software libraries. Here we don't need to install anything. The cloud service provider will manage servers, network, OS and storage. The users will develop, deploy, configure and manage applications on the cloud infrastructure.

SAAS:

It provides complete software application or the user interface to the application itself. The user is not concerned about the underlying architecture of cloud only service provider is responsible for this. It is platform independent and can be accessed from various client devices such as workstation, laptop, tablet and smart phone, running different OS. Ex: The online software we use like online image converter, doc converter etc.

Big data analytics:

Big data refers to large amount of data which cannot be stored, processed and analysed using traditional database like (oracle, mysql) and traditional processing tools. In big data analytics BIG refers to 5 Vs.

- Volume
- Velocity
- Variety
- Veracity
- Value

Volume: volume refers to the massive amount of data generated from the IoT systems. There is no threshold value for generated data. It is difficult to store, process and analyse using traditional database and processing tools. Ex: The volume of data generated by modern IT, industrial and healthcare system.

Velocity: The rate at which the data is generated from the IoT system. This is the primary reason for the exponential growth of data. Velocity refers to how fast the data is generated and how frequently it varies. Ex: Modern IT, industrial and other systems like social networking sites are generating data at increasingly higher speed.

Variety: Variety refers to different forms of data. Since there are various domain of IoT so various type of data are generated from different IoT domain. Those data is called as sparse data. Those data include text, audio, video etc.. The variety of data is mainly divided into 3 types i.e.

- ✓ structured
- ✓ semi structured
- ✓ unstructured

Structured data: The data which has a fixed format to be stored is known as structured data. The data stored in database like oracle, mysql is an example of structured data. With a simple query data can be retrieved from the database.

Semi-structured data: The data which has not a fixed format to be stored but uses some elements and components through which they can be analysed easily is known as semi-structured data. Ex: HTML, XML, JSON data

Unstructured data: The data which has not any fixed format. It is difficult to store and analyse. It can be analysed after converting into structured data. Ex: Audio, video (gif, audio with lyrics), Text (containing special symbols).

Veracity: The data in doubt is known as veracity. Sometimes what happen it is very difficult accept the data stored in database. This happens due to typical error, corrupted storage or data.

Value: It is efficient to access big data if we can turn it into values i.e we can find greater insights from it so that we can perform some action to get the desired output. This will be beneficial for the organisation. Otherwise it has no use.

Embedded Systems:

1. An embedded system is a computer system that has hardware and software embedded to perform specific task.
2. The key components of an embedded system include microprocessor or micro controller, memory (RAM, ROM, Cache), networking units (Ethernet, Wi-Fi adapter), input/output units (display, keyboard, etc) and storage (flash memory). They use some special types of processor such as digital signal processor, graphics processor and application specific processor). Embedded system uses embedded OS like RTOS.
3. Ex. Of embedded systems: digital watch, digital camera, vending machines.

Communication protocols:

1. Protocol is nothing but rules and regulations. Communication protocol is the backbone of the IoT system.

2. It allows interoperability among various devices. It enables network connectivity and coupling to applications.
3. It allows devices to exchange data over the network. These protocols define data exchange format, data encoding, addressing schemes for devices and routing of packets from source to destination. It also includes sequence control, flow control and retransmission of lost packets.

DIK principle (Data Information and Knowledge):

IoT is based on DIK principle. DIK refers to

- Data
- Information
- Knowledge

Data: It refers to raw and unprocessed values that are generated by the IoT devices. It does not have meaning until it is contextualised and processed into useful information.

Information: The raw data is processed, contextualised, filtered, contextualised, categorised and condensed in order to refer information. For this different algorithms and applications on IoT networks are used to extract and create information from lower level data. The raw data will be given as input to this program then they will be processed in order to get some kind of information.

Knowledge: Knowledge can be inferred from information by organizing and structuring information and that can be put into action to achieve specific objectives.

Ex: Consider a series of raw sensor measurement ((72, 45); (84, 56);) generated by a weather monitoring station. This does not have some meaning. To give meaning to the data, a context is added ex: in this example we can add that the data represents the temperature and humidity measured every minute. After adding this we get the information about the data tuple. Further information is obtained by categorising, condensing or processing the data. For ex: the average temperature and humidity reading for last five minutes is obtained by averaging the last five data tuples. The next step is to organise the information and understand the relationships between pieces of information to infer knowledge which can be put into actions. For ex: an alert is raised if the average temperature in last five minutes exceeds 120F.

Generic Block Diagram of IoT devices:

For diagram refer to Book.

Connectivity (RJ 45, USB): This component is used to connect the IoT device to the internet.

Processing unit (CPU): This is used to control all the other components as well as it processes the instruction present in the algorithm.

Audio/ video unit: This is used to connect the monitor or speaker.

HDMI: High definition multimedia interface. It supports uncompressed all digital audio video interfaces which provides all digital audio and video via a single cable. HDMI provides an interface between any audio/video source such as set-top box, DVD player or audio/ video receiver and audio and or video monitor such as a digital television over a single cable. HDMI supports high definition video, plus multi-channel digital audio on a single cable.

2.5/3.5 mm audio: This is used for audio.

RIC: Radio Corporation of America. It is sometimes called as phono connector. It is a type of electrical connector commonly used to carry audio and video signals.

I/O interfaces: This is used to connect sensors, actuators, relay or any external devices.

UART: universal Asynchronous Receiver Transmitter. It is used to provide serial communication.

SPI: Serial peripheral interface. It is used to provide serial communication between different devices or you can say between monitoring node and controlling node or coordinator node. In SPI there will be a single master and multiple slaves. There are 4signals. CHIPENABLE with a bar (active low) , MISO, MOSI, clk/Slk/Sck.

Since multiple slaves are there so the slave will choose them using CHIPENABLE signal. If low voltage will be given to the slave then that slave will be able to communicate with the master.

MISO: Master in Slave out. i.e Master will not send anything it will only receive and slave out i.e slave will only send.

MOSI: Master out slave in i.e master will send the data and receiver will receive.

clk/slk/sck: It will synchronise the master and slave.

Refer the note book for the diagram

I2C: Inter integrated circuit. Here multiple masters can communicate with multiple slaves.

Refer the note book for the diagram

CAN: Controller area network. It is designed to allow micro controller and devices to communicate with each other in applications without a host computer.

Storage: This is used to store or record the sensed data.

SD: Secure digital. This is used to hold the SD card which is a non-volatile memory card format. SD card is a popular storage media for digital cameras and other mobile devices.

SDIO: It is used to connect external HDD.

MMC: Multi Media card. It is a flash memory based memory card standard. It is a popular storage media for digital cameras and other mobile devices.

GPU: Graphics processing unit. It is useful for the resolution of the screen and used for high end games.

Memory: There are used for temporary storage and for execution of instruction.

IoT levels:

Based upon the number of monitoring nodes used, type of data base used, complexity/ simplicity of analysis, computation there are 6 levels of IoT. Different applications are implemented based on this level. The IoT systems consist of these following components.

- Device
- Resources
- Controller Service
- Database
- Web Service
- Analysis Component
- Application

Device: The IoT device allows identification, remote sensing, actuating, and remote monitoring capabilities.

Resource: Resources are the software components on the IoT device for accessing, processing and storing sensor information, or controlling actuators connected to the device Resources include the software components that enable network access for the device. For ex: The programs that we have written for object detection using IR sensor, to find out the distance using ultra sonic sensor etc.

Controller Service: Controller service is a native service that runs on the device and interacts with the web services. Controller service sends data from the device to the web service and receives commands from the application for controlling the web services. For ex: The ESP 8266 programming, setting of API keys, SSID etc. .

Database: Database can be either local or in the cloud and stores the data generated by the IoT device.

Web Service: This act as an interface between IoT device, application, database and analysis components. Web services can be implemented using HTTP and REST principle or using WebSocket protocol.

Analysis Component: The analysis component is responsible for analysing the IoT data and generates results inform which are easy for the user to understand. Analysis can be performed either locally or in the clouds.

Application: IoT applications provide an interface that the user can use to control and monitor various aspects of the IoT system.

IoT level 1:

Refer diagram from book

I. It has single node/device for sensing, monitoring, actuating, storing data, performing analysis and hosting application.
II. Data involved is not big. So data is stored in local database.
III. It is suitable for modelling design low cost and low complexity solution.
IV. Primary analysis requirement is not computationally intensive. So can be analysed locally.
V. EX: Home automation

IoT level 2:

Refer diagram from book

I. It has single node/device for sensing, monitoring, actuating, performing analysis and hosting application.
II. Data involved is big. So data is stored in cloud.
III. It uses cloud based application to visualise data.
IV. Primary analysis requirement is not computationally intensive. So can be analysed locally.
V. EX: smart irrigation

IoT level 3:

Refer diagram from book

I. It has single node/device for sensing, monitoring, actuating and hosting application.
II. Data involved is big. So data is stored in cloud.
III. It uses cloud based application to visualise data.
IV. Primary analysis requirement is computationally intensive. So can be aggregated and analysed in cloud.

EX: tracking package handling.

IoT level 4:

Refer diagram from book

I. It has multiple nodes/devices for sensing, monitoring, actuating, performing analysis and hosting application.
II. Data involved is big. So data is stored in cloud.
III. It uses cloud based application to visualise data.
IV. Primary analysis requirement is computationally intensive. So can be aggregated and analysed in cloud.
V. It has two observer nodes i.e local and cloud based. They can subscribe to and receive information collected in cloud from IoT device. They can process and use those information for various applications
VI. Observer node does not perform any control function.

EX: Noise monitoring

IoT level 5:

Refer diagram from book

I. It has multiple nodes/devices for sensing, monitoring, actuating. One coordinator node for collecting and sending the data to cloud by controller service.
II. Data involved is big. So data is stored in cloud.
III. It uses cloud based application to visualise data. Suitable for wireless sensor network.
IV. Primary analysis requirement is computationally intensive. Analytic component analyse the data and stores result in cloud and make prediction.
V. Ex: Forest Fire Detection

IoT level 6:

Refer diagram from book

I. It has multiple independent nodes/devices for sensing, monitoring, actuating and sending the data to cloud by controller service.
II. Data involved is big. So data is stored in cloud.
III. It uses cloud based application to visualise data.
IV. Primary analysis requirement is computationally intensive. Analytic component analyse the data and stores result in cloud.
V. It has centralized controller which is aware of the status of all the end nodes and sends control command to the nodes.
VI. Weather monitoring and structural health monitoring.

Networking Model:

It explains how to facilitate communication between two different systems without requiring changes to the logic of underlying hardware and software i.e to allow interoperability.

Based on ISO standard there are basically two networking model.

- OSI model: Open system Interconnection
- TCP/IP model: Transmission control protocol/ Internet protocol

ISO: International standard organisation/ International organisation for standardisation. It is dedicated to worldwide agreement of international standards. The ISO standard that covers all aspect of network communication is the OSI or TCP/IP model.

OSI model: It is a model for understanding and designing a network architecture which is flexible, robust and interoperable. It is a layered framework. The layers of OSI model define a part of the process of moving information. Each layer defines a family of functions distinct from those of the other layer.

Open System: Set of protocols that allow any two different systems to communicate regardless of their underlying architecture.

OSI and TCP/IP is mainly divided into 3 units i.e

- ✓ Network support layer or lower layer
- ✓ User support Layer or upper layer
- ✓ Transport layer or Intermediate Layer

Network support layer: It comprises of Physical link layer, data link layer, network layer. It deals with physical aspects of moving data from one device to other (such as electrical specification, physical connection, physical addressing, and transfer timing and reliability). It is implemented using hardware and software.

User support layer: It comprises of session layer, presentation layer, and application layer. It allows interoperability among unrelated software system. It is implemented using software.

Transport layer: It links the two subgroups and ensures that what the lower layer have transmitted is in a form that the upper layer can use.

OSI model: It consists of 7 layers. It is a logical concept. It not practically implemented.

- ✓ Physical link layer
- ✓ Data link layer
- ✓ Network layer
- ✓ Transport layer
- ✓ Session layer
- ✓ Presentation layer
- ✓ Application layer

Physical link layer functionalities:

I. The protocol data unit (PDU) is bits/ signal. It coordinates the function which will help carrying bit streams over a physical media.
II. It deals with electrical and mechanical specification of transmission medium and interface.
III. It describes the physical characteristics of interface and transmission medium, representation of bits, data rate, transmission rate, synchronisation of bits, line configuration, and transmission mode.

Data link layer functionalities:

I. The PDU is frame.
II. It provides hop to hop connection in same network.
III. It deals with physical addressing.
IV. It supports flow control, error control, access control.

Network layer functionalities:

I. The PDU is packet.
II. It provides source to destination delivery of individual packet in different network.
III. It deals with logical addressing (IP addressing)
IV. It supports fragmenting, routing, traffic shaping, load shading

Transport layer functionalities:

I. The PDU is segment/ datagram
II. It provides process to process delivery of entire message i.e end to end communication.
III. It deals with port addressing.
IV. It supports segmentation and reassembling.
V. It provides flow control, error control and congestion control

Session Layer functionalities:

I. PDU is message.
II. It provides dialog control.
III. It allows a process to add a check point or synchronisation points to a stream of data (to provide synchronisation).

Presentation layer:

I. PDU is message.
II. It is concerned with syntax and semantics of the information exchanged between two systems.
III. It is responsible for translation, encryption, and compression.

Application Layer:

I. PDU is message.
II. It defines how the application interface with the lower layer protocols to send the data over the network.

TCP/IP:

This networking model consists of 4 layers. It is practically implemented. Now-a-days this networking architecture or model is used. The layers are as follows:

- ✓ Link layer
- ✓ Network layer/ Internet layer
- ✓ Transport layer
- ✓ Application layer

The protocol suite of TCP/ IP model:

Refer book for the diagram:

Application layer protocol:

- HTTP (Hyper Text Transfer Protocol)
- CoAP (Constrained application protocol)
- Websocket
- MQTT (Message queue telemetry transport)
- XMPP (Extensible messaging and presence protocol)
- DDS (Data distribution service)
- AMQP (Advanced message queuing protocol)

HTTP:

I. It is a web transfer protocol.
II. It suitable for web browsing.
III. It is a stateless protocol.
IV. It uses ort number TCP: 80 in transport layer.
V. It uses URI to identify HTTP Resources.
VI. It uses Request- response model.
VII. It uses client server architecture.
VIII. IT uses PUT, GET, POST, DELETE HEAD, TRACE, OPTIONS method to communicate with the resources.

CoAP:

I. It is a web transfer protocol.
II. It is suitable for machine to machine communication.
III. It is applicable for constrained environment with constrained device and network. It is a stateful protocol.

IV. It uses UDP 5683 port number in transport layer.
V. It uses request- response model.
VI. It uses client server architecture.
VII. It uses PUT, GET, POST, DELETE, HEAD, TRACE, OPTIONS.
VIII. It is designed to work with HTTP.

Websocket:

I. It allows full duplex communication over a single socket connection to provide data transfer between client and server keeping TCP connection open.
II. It is a stateful protocol.
III. It uses TCP as transport layer protocol.
IV. It uses request-response communication model.
V. It uses client server architecture.

MQTT:

I. It is a low weight messaging protocol (small message size).
II. Suitable for constrained environment where devices has low processing, capabilities, less memory and less band width.
III. It uses TCP as transport layer protocol and uses port number TCP 1883.
IV. It uses Publish-subscribe model.
V. It uses client- server architecture.

XMPP:

I. It is a decentralised protocol.
II. It is suitable for real time communication and streaming small chunk of XML data between network entities in real time.
III. It uses TCP as transport layer protocol.
IV. Client server architecture allows client to server and server to server communication.
V. It provides range of application such as messaging, multiparty chat, voice/ video call, gaming.

DDS:

I. It is a data centric middleware standard.
II. It provides device to device or machine to machine communication.
III. Uses publish subscribe model.
IV. It provides QoS (Quality of Service) control and configurable reliability.

AMQP:

I. It is an open application layer protocol for business messaging.
II. Supports point to point and publish/ subscribe model, routing, queuing.
III. It uses 5672 TCP.

Transport Layer:

Various transport layer protocols are as follows:

- TCP (Transmission control protocol)
- UDP (User data gram protocol)

Services supported by TCP/UDP	TCP	UDP
1. Abbreviation	Transmission Control Protocol	User Datagram protocol
2. Protocol data unit	segment	datagram
3. Connection	Connection oriented	Connection less
4. Reliability	Reliable and in order as connection oriented	Unreliable and out of order as connection less.
5. Phases	3 phase i.e connection establishment, data transfer, connection release.	One phase i.e data transfer
6. Delay between data unit	Uniform delay as throughout the travel they follow same path.	Non uniform delay as throughout the travel they follow different path.
7. Congestion	Occurs during connection establishment	Occurs during data transfer.
8. Resources	Dedicated	Shared
9. Utilisation of resources	Less utilisation due to connected path and dedicated resources.	Proper utilisation due to shared resources
10. Fault tolerant technique	No fault tolerant technique	Uses fault tolerant technique
11. Message size	Prefer long message	Prefer short message
12. Flow control	Provides flow control with help of ACK (acknowledgement) field and sequence number field.	Does not provide flow control
13. Error control	It provides error control	It does not provide error control
14. Dependency on ICMP (Internet control Message Protocol)	Does not depend upon ICMP	depends upon ICMP at network layer.
15. Multi casting	Does not support	Supports multicasting
16. Broad casting	Does not support	Supports broadcasting
17. Examples of applications using TCP/ UDP	HTTP: 80 HTTPS: 443 SMTP: 25 Telnet: 23	TFTP: 69 NTP: 123 DHCP: 67,68 SNMP: 161

Network layer:

The network layer protocols are as follows:

- IPv4: ex: 192.168.32.65
- IPv6: ex: ffef: 1234: 123f: 56de: 12d6:ffff:34ab:dcab
- 6LOWPAN

IPv4	IPv6
Internet protocol version 4	Internet protocol version 6
It is 32 bit dotted decimal	It is 128 bit colon hexa decimal
It supports 2^{32} number of addresses	It supports 2^{128} number of addresses
It supports point to multipoint connection.	It supports point to point connection
All the applications have same priority.	Different applications have different priority.
It is connection less.	It is connection oriented.
Here at each router fragmentation and defragmentation takes place.	Here at each router fragmentation and defragmentation takes place.

6LoWPAN:

IPv6 over low power personal area network. It brings IP protocol to the low power devices which have limited processing capability. It operates in 2.4 GHz frequency. It supports data transfer rate up to 250kb/s. 6LoWPAN works with the 102.15.4 link layer protocol.

Link layer:

Link layer protocols are as follows.

- IEEE 802.3- Ethernet
- IEEE 802.11- WiFi
- IEEE802.16- WiMax
- IEEE 802.15.4-LR-WPAN
- 2G/3G/4G mobile communications.

IEEE802.3:

I. It is a standard for coaxial cable, twisted pair cable and fiber optic cable as a shared medium.
II. 802.3 is the standard for 10Base5 Ethernet over coaxial cable as shared medium. 10 means10Mbps and 5 means 500 meter.
III. 802.3 i is the standard for 10Base-T Ethernet over copper twisted-pair connection.
IV. 802.3 j is the standard for 10Base-F Ethernet over copper twisted-pair connection. 802.3 ae is the standard for 10 Gbits /Ethernet over fiber.
V. It provides data transfer rate from 10 Mb/s to 40 Gb/s and higher.

Services	WiFi	WiMax
Abbreviation	Wireless Fidelity	Worldwide interoperability

		for microwave access.
Official release	1997	2004
IEEE standard	Defined under IEEE802.11 x standard where x is various wifi versions.	Is standardised under 802.16 y family of wireless networking where y refers to various wimax versions
Versions of the standards	802.11a, 802.11b, 802.11g, 802.11n, 802.11ac, 802.11ad, 802.11i This is wireless LAN standard. Data transfer rate from 1Mbps to 6.75 Gbps	802.16a, 802.16d, 802.16e, 802.16 m. This is wireless broadband standard.
Frequency band	Defines under ISM band where user has to pay no extra charging for utilising those bands.	No bar on frequency usage in the WiMAX i.e protocols might wirk in the ISM bands or they might use a licensed frequency version for which the user probably would be charged.
Range	An idle wifi based network reaches around 100 m, as it maximum range.	An idle wi-max network can reach about 80-90 kms
Data transfer rate	Can transfer data at speeds up to 54mbps.	Can exchange data at speed upto 40 mbps.
Channel bandwidth	20 MHz	Ranges from 1.25 MHz to 20 MHz

802.15.4- LR WPAN:

It is for low rate wireless personal network. Zigbee supports this protocol. It provides data transfer rate from 40 kbps to 250 kbps. It is applicable for power constrained device providing low cost and low speed communication.

2G/3G/ 4G mobile communications:

There are different generation of mobile communication standards including second generation (GSM,CDMA), third generation (WCDMA, UMTS, CDMA2000) and fourth generation (LTE). The IoT devices based on these standards can communicate over cellular networks. It supports data transfer rate from 9.2 kb/s up to 100 mb/s.

Sensor:

- Sensor is a device/ module/ subsystem which purpose is to detect the event or change in its environment and send the information to other electronics frequently a computer processor.
- We can say it as a device which detects and measures a physical property and records, indicates or otherwise responds to it.

- It responds to some type of input from the physical environment. The specific input could be light, heat, motion, moisture, pressure or any one of a great number of other environmental phenomena. The output is generally a signal i.e converted to human readable display at the sensor location or transmitted electronically over a network for reading or further processing. It responds to electrical or optical signal.

Actuator:

It is a device which converts energy into motion.

M2M	IoT
1. M2M is subset of IoT	1. IoT is super set of M2M. It is a vast network of connected devices that may or may not include human interaction.
2. Point to point communication usually embedded within hardware at the customer site. Ex (Tap to pay, NFC). The service in Samsung taps to pay without using card. You just need to bring your mobile near the swipe machine. Using your ID card to open or close the door.	2. Multipoint Communication. Ex: By showing the ID card then your attendance will be registered. Monitoring the health parameters through the help of android app or web app.
3. It uses cellular or wired network.	3. It uses wireless network.
4. Do not necessarily require internet connection.	4. Majority of cases require Internet connection.
5. It is more hardware based.	5. It is more software based.
6. On premises applications. (i.e software and technology i.e located within the physical confines of an enterprise or company's data centre.) Ex. Diagnosis applications, Service management applications and on-premises enterprises applications.	6. It uses cloud based application such as analytic applications, enterprise applications, and remote diagnosis and management applications.
7. Data collection and analysis is done in on premises storage infrastructure.	7. Data collection and analysis is done in cloud mostly.
8. Non IP based communication.	8. IP based communication.
9. M2M protocols include ZigBee, Bluetooth, ModBus, M-Bus, Wireless M-Bus, Power Line communication (PLC), 6LoWPAN, IEEE 802.15.4, Z-wave. These protocols work below the network layer.	9. IoT protocols include HTTP, CoAP, Web Socket, MQTT, XMPP, DDS, AMQP etc.. The protocols work above the network layer.
10. The devices used are homogeneous type within an M2M area network.	10. The devices used are heterogeneous type such as fire alarms, door alarms, lighting control devices, etc..

The protocols of M2M:

i. ZigBee
ii. Bluetooth
iii. 6LoWPAN
iv. IEEE 802.15.4
v. ModBus
vi. M-Bus
vii. Wireless M-Bus
viii. PLC

ZigBee:

1. It is a new wireless technology based on IEEE802.15.4 standards.
2. It is created for remote control and sensor networks.
3. It is created by ZigBee Alliance. The members of these alliances are Philips, Motorola, intel, HP.
4. It is implemented with low cost. It provides reliable data transfer, short range operations, very low power consumptions, adequate security features.
5. Useful for home automation.

ModBus:

1. It is a serial communication protocol.
2. It is developed by Modicon published by Modicon in 1979.
3. It is used for PLC (programmable logic controllers).
4. It is a method for transmitting information over serial lines between electronic devices.
5. It is an open protocol i.e it's free for manufacturers to build into their equipment without having to pay royalties.
6. It is typically used to transmit signals from instrumentation and control devices back to main controller or data gathering system. Ex: A system measures temperature and humidity and communicate the result to computer. ModBus is generally used to connect a supervisory computer with remote terminal unit in supervisory control and data acquisition (SCADA).

M-Bus (Meter bus):

1. It is an Europian Standard for the remote reading of gas or electricity meters.
2. M-Bus is also usable for other type of consumption meters (ex-water, gas).
3. The M-Bus interface is made for communication on two-wires, making it cost effective.

PLC (Power Line Communication):

1. It is a communication technology that enables sending data over existing power cables.
2. The power cable can both power up the device and at the same time control/ retrieve data from it in a half duplex manner.

SDN (Software defined network):

SDN is defined as a network architecture that separates the control plane from the data plane and centralises the network controller.

1. So the main components of SDN are Control Plane, Data Plane and centralised SDN controller, programmable open APIs, standard communication interface (open flow).
2. **Control Plane (CP)**: It is the pat of the network that carries the signalling and routing message traffic. It takes the decision about how the packets should flow through the network. It decides the path as well as the metrics to be followed in order to decrease

the traffic. The CP allows dynamic access and administration. A network administrator can shape the traffic from a centralised controller console without touching individual switches to change the metrics in the preloaded program in switches and routers (i.e the administrator can change any network switches rules when necessary). Prioritising, de-prioritising or even blocking specific type of packets i.e called dynamic routing. This is especially helpful in cloud computing.

3. **Data plane:** It is the part of the network that carries the payload data traffic from one place to other.
4. **SDN controller:** SDN controller manages the data traffic. The SDN controller is a software installer in server at data centre. It is based on protocols. It acts in between network devices at one end and applications at other end. Any communication between applications and devices has to go through controller.
5. **Programmable open APIs:** The SDN applications can be deployed through programmable open APIs. It acts as an interface between SDN application and control layers (north bound interface). This helps to implement various network services like routing, access control and quality of service (QoS).
6. **Standard communication interface (Openflow):** It is the interface between the control and infrastructure layers (south bound interface). OpenFlow is defined by Open networking Foundation (ONF). With the OpenFlow the forwarding plane of the network devices can be directly accessed and manipulated. It uses concept of flows to identify network traffic based on predefined match rules. Flows can be programmed statically or dynamically by the SDN control software.

6.1. <u>Working of OpenFlow Switch:</u>

i. The components of OpenFlow switch are one or more flow tables and group table, which perform packet lookups and forwarding and openflow channel to an external controller.

ii. The OpenFlow protocol is implemented on both the sides of the interface between the controller and the network devices.

iii. The controller manages the switch via OpenFlow switch protocol. The controller can add, update, and delete flow entries.

6.2. OpenFlow flow table:

1. Each flow table contains set of flow entries.
2. Each flow entry consists of match fields. Counters and set of instructions to apply to matching packets.
3. Matching starts at the first flow table and may continue to additional flow table of the pipeline.

Benefits of SDN over conventional network architecture:

1. It makes networks flexible with the help of software by removing the demerits of traditional or conventional network architecture.
2. It reduces the complexity of increasing number of distributed protocols and the use of proprietary hardware and interfaces which are used to be implemented in conventional network. It uses simple packet forwarding technique as opposing to conventional network.

3. It separates control plane from the data plane and centralises the network controller. But in conventional network architecture the control plane and data plane are coupled.
4. The other benefit of SDN is network management and end to end visibility. The network admin only deal with one centralised controller to distribute policies to the connected switches instead of configuring multiple individual devices.
5. SDN applications can be deployed through programmable open APIs so this speeds up the innovation as the network administrators no longer need to wait for the device vendors to embed new features in their proprietary hardware.

SDN architecture:

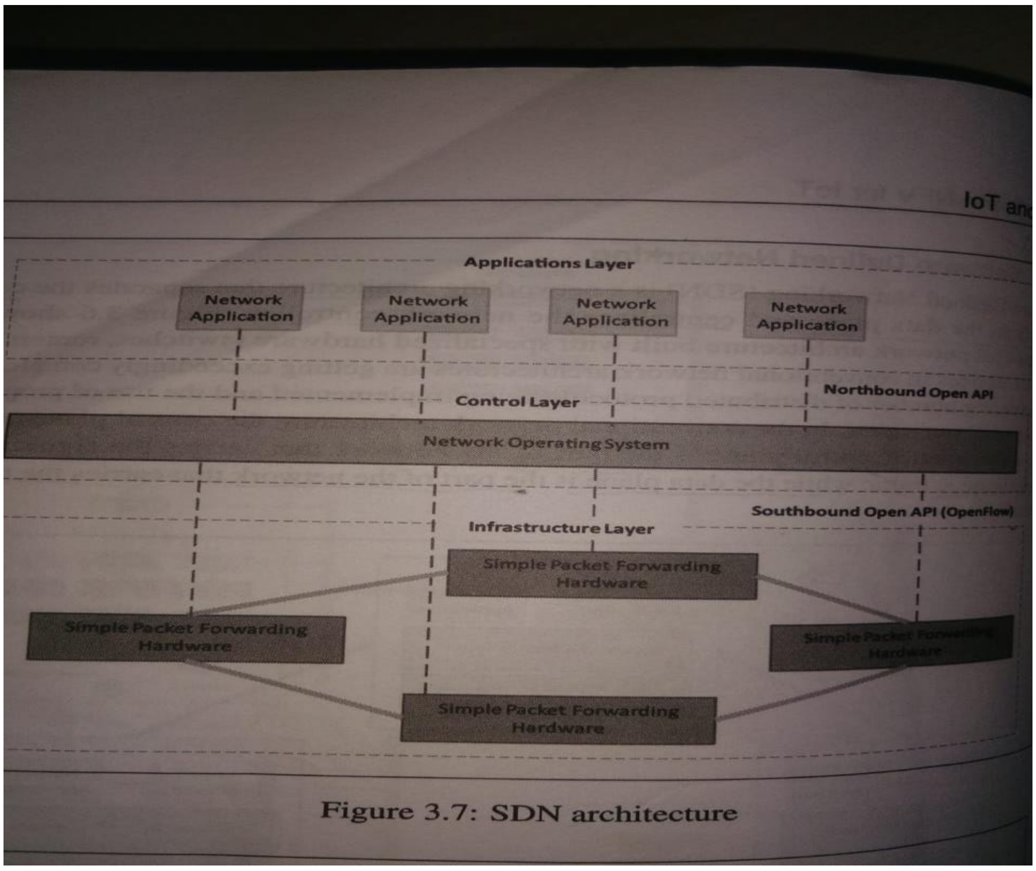


Figure 3.7: SDN architecture

Challenges of SDN:

If the controller will be hacked or corrupted then whole system will not work.

Note: ONF is the broadly accepted SDN protocol for the southbound interface.

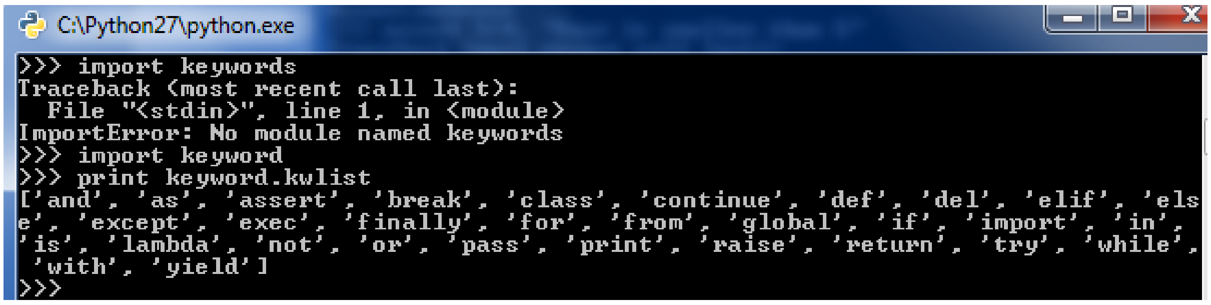
Python Keywords:

Key words are also known as reserved words whose meaning is already defined to the compiler. We cannot use keywords as variable name. In Python 2.7 there are 31 keywords and in python 3.3 there are 33 key words. Depending upon the versions the keywords may vary. But we will discuss some of the common keywords.

Instructions to find the keywords in any version of Python are

>>>import keyword

>>>print (keyword.kwlist)



C:\Python27\python.exe

```
>>> import keywords
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named keywords
>>> import keyword
>>> print keyword.kwlist
['and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'els
e', 'except', 'exec', 'finally', 'for', 'from', 'global', 'if', 'import', 'in',
'is', 'lambda', 'not', 'or', 'pass', 'print', 'raise', 'return', 'try', 'while',
 'with', 'yield']
>>>
```

Various keywords:

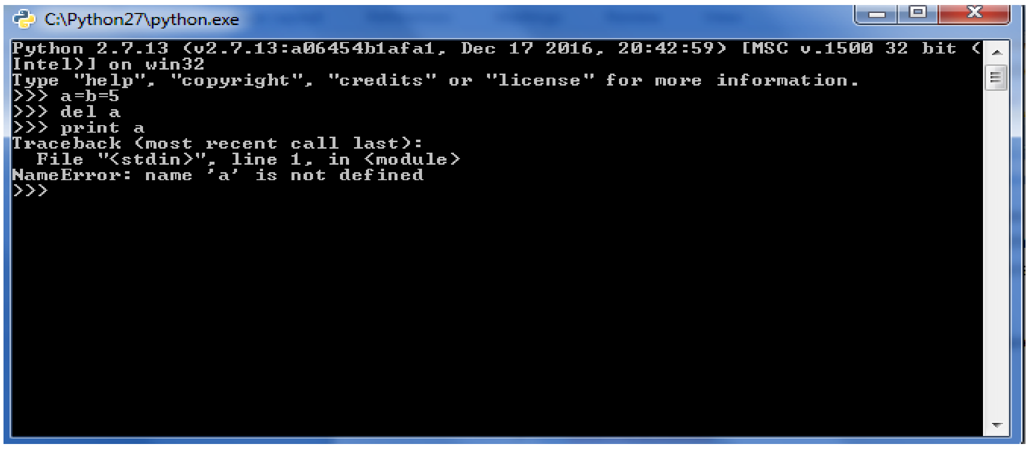
del	assert	while	return	except	is
True(3.3)	import	for	None(3.3)	raise	exec
False(3.3)	from	break	if	finally	pass
and	as	continue	else	global	yield
or	in	def	elif	Nonlocal(3.3)	range
not	print	lambda	try	Class	with

Del: This key word is used to delete the reference of a variable or an object from the memory. Ex:

>>> a=b=5

>>>del a

>>>print a



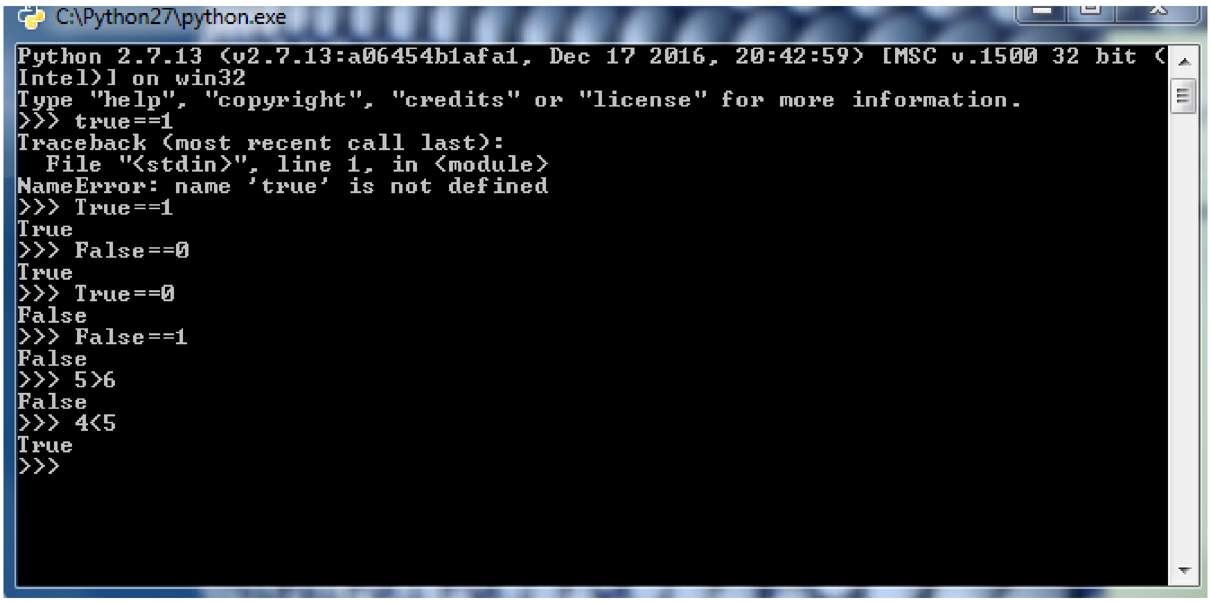
C:\Python27\python.exe

```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a=b=5
>>> del a
>>> print a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
>>>
```

Since we have deleted the reference of the variable ‘a’ so while printing ‘a’ it is showing error.

‘del’ is also used to delete an item from list or dictionary.

True False: These are truth values in python. These are result of comparison operation and logical operation. Because if we are comparing anything then that might be true or false but not the both.0 is also considered as false and 1 is also considered as true.



```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> true==1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
>>> True==1
True
>>> False==0
True
>>> True==0
False
>>> False==1
False
>>> 5>6
False
>>> 4<5
True
>>>
```

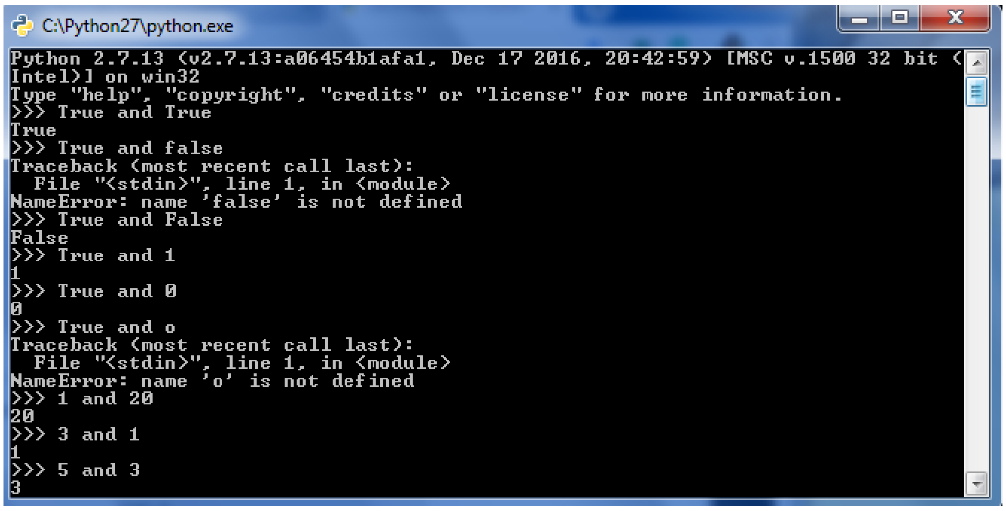
True+True=2

and , or, not: These are the logical operators in python. These are used to combine two or more statements or instructions or invert one instruction. Those statements are called operands.

and: if truth value of both the operands are true then only the result is true otherwise false.

Operand A	operand B	A and B
False	False	False
False	True	False
True	False	False
True	True	True

Truth table of AND



```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> True and True
True
>>> True and false
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'false' is not defined
>>> True and False
False
>>> True and 1
1
>>> True and 0
0
>>> True and o
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'o' is not defined
>>> 1 and 20
20
>>> 3 and 1
1
>>> 5 and 3
3
```

or: if the truth value of one of the operands is true then the result is true otherwise false.

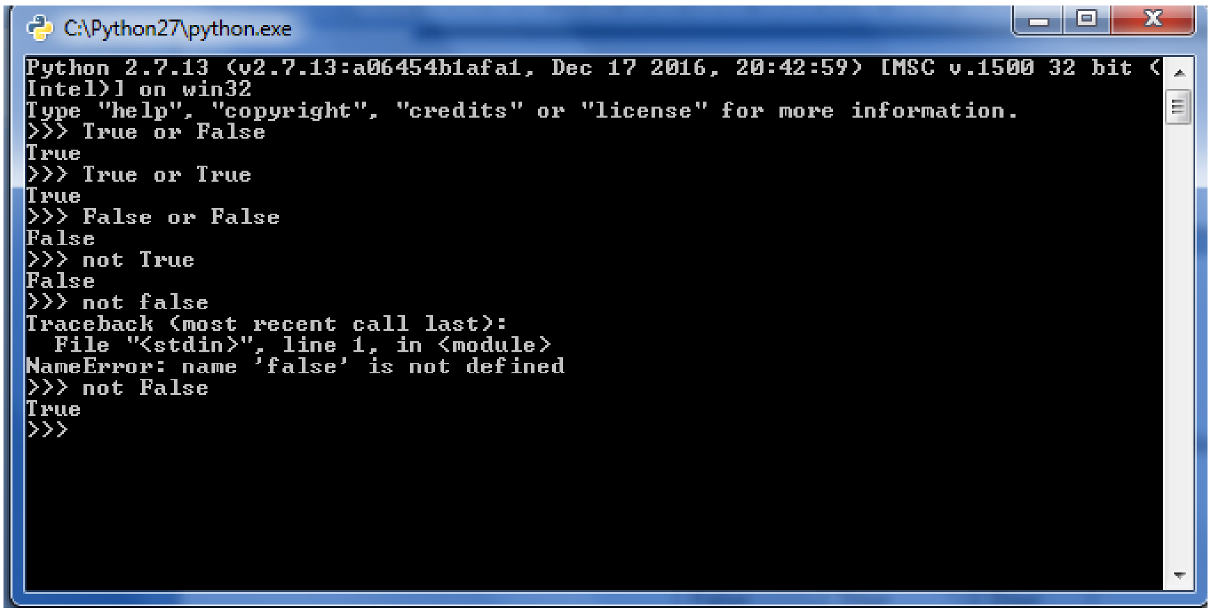
Operand A	operand B	A and B
False	False	False
False	True	True
True	False	True
True	True	True

Truth table of OR

not: this is used to invert the statement.

Operand A	Not A
True	False
False	True

Truth table of Not



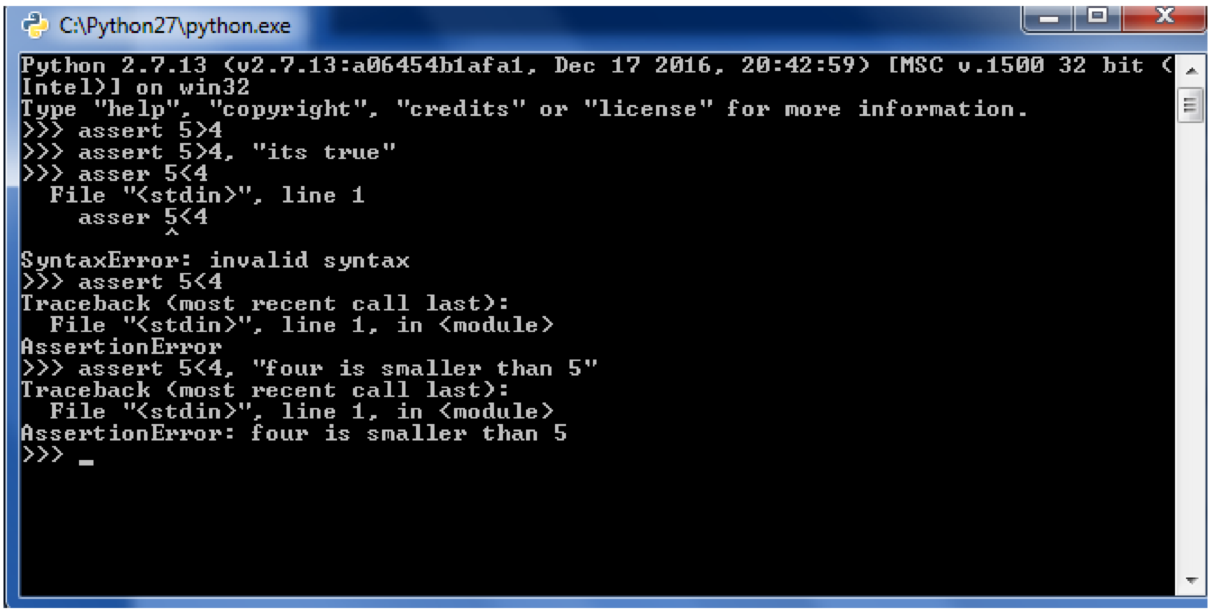
```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> True or False
True
>>> True or True
True
>>> False or False
False
>>> not True
False
>>> not false
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'false' is not defined
>>> not False
True
>>>
```

assert: assert is used for debugging purpose. It helps to find out the bugs conveniently. It is used to check internal state or whether any assumption is true or false. It is associated with a condition. If the condition is true then it will not return anything but if the condition is false then it will return an AssertionError. If we want to print some error message then also we can do. i.e

>>> assert condition, “message you want to print”

The above instruction is equivalent to >>> if not condition:

raise AssertionError (message)



```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> assert 5>4
>>> assert 5>4, "its true"
>>> asser 5<4
  File "<stdin>", line 1
    asser 5<4
          ^
SyntaxError: invalid syntax
>>> assert 5<4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
>>> assert 5<4, "four is smaller than 5"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError: four is smaller than 5
>>> _
```

Import: Import key word is used to import a module to the current program or namespace.

>>> import module name

Ex: let we want to import “math module” which contains all mathematical as well as trigonometric attributes and functions. Math module is a standard module in python.

>>> import math

>>>print math.cos()

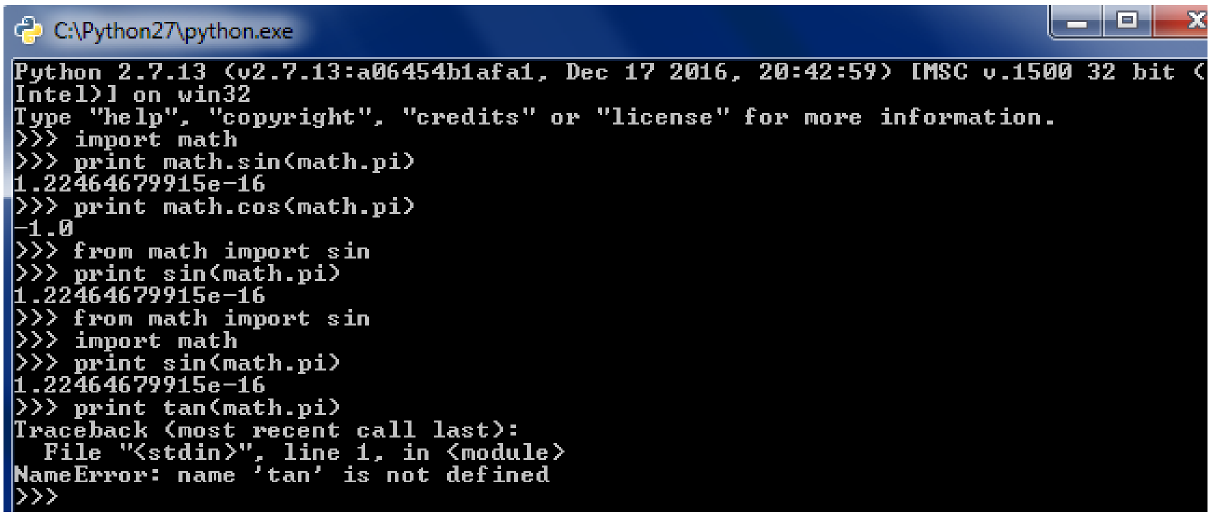
>>>print math.sin()

From: if we want to import a specific attributes or functions of the module then we use from---import.

>>> from module name import attribute name.

Ex: let we want to use only sin () from the math module then

>>>from math import sin



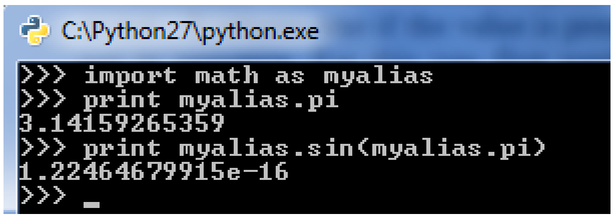
```
C:\Python27\python.exe
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> print math.sin(math.pi)
1.22464679915e-16
>>> print math.cos(math.pi)
-1.0
>>> from math import sin
>>> print sin(math.pi)
1.22464679915e-16
>>> from math import sin
>>> import math
>>> print sin(math.pi)
1.22464679915e-16
>>> print tan(math.pi)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'tan' is not defined
>>>
```

As: This is used to create an alias while importing a module. It gives a user defined name to the module while importing it.

Ex: we want to find the result of sin pi

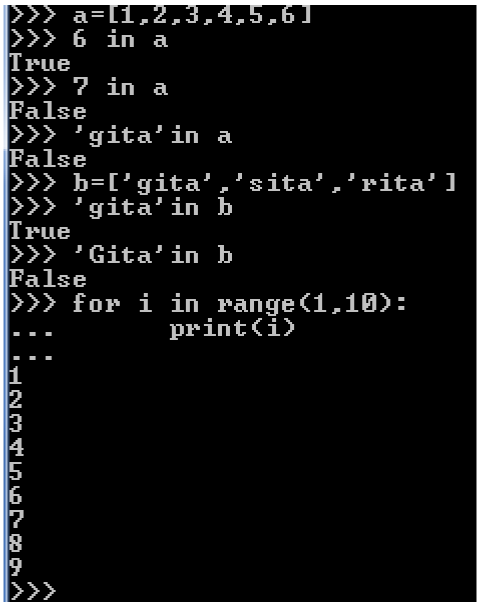
>>> import math as myalias

>>> print myallias.sin(myalias.pi)



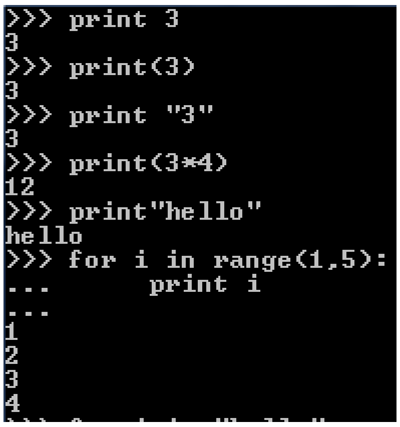
```
C:\Python27\python.exe
>>> import math as myalias
>>> print myalias.pi
3.14159265359
>>> print myalias.sin(myalias.pi)
1.22464679915e-16
>>> _
```

In: it is used to check whether a sequence (list, string, tuple, dictionary, set) contains a value or not. It returns true if the value is present in the sequence and returns false if the value is not in sequence. For this you first need to define a sequence. It is also used to traverse through a sequence in for loop.



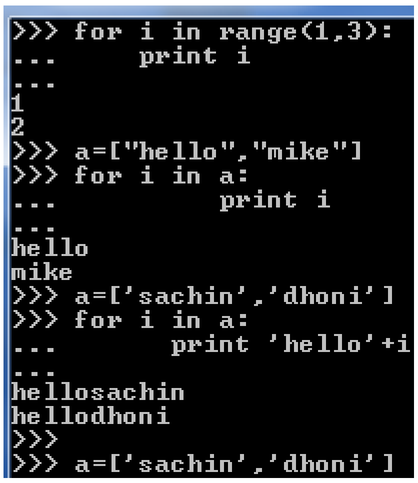
```
>>> a=[1,2,3,4,5,6]
>>> 6 in a
True
>>> 7 in a
False
>>> 'gita'in a
False
>>> b=['gita','sita','rita']
>>> 'gita'in b
True
>>> 'Gita'in b
False
>>> for i in range(1,10):
...         print(i)
...
1
2
3
4
5
6
7
8
9
>>>
```

Print: It is a standard library function in python. It is used to print a string or value of an expression or value of a variable etc.

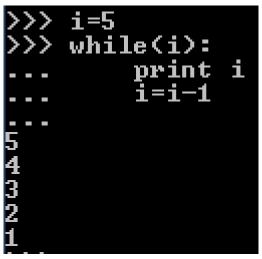


```
>>> print 3
3
>>> print(3)
3
>>> print "3"
3
>>> print(3*4)
12
>>> print"hello"
hello
>>> for i in range(1,5):
...       print i
...
1
2
3
4
```

While, for: these are loop control instructions. If want to execute two or more statements repeatedly until and unless the condition is false or a break statement incidents. Generally for loop is used when we know the total number of iterations. We can use for with any type of sequence also.

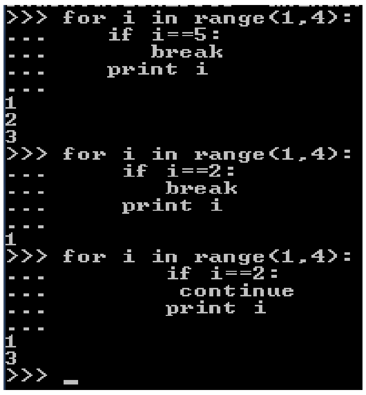


```
>>> for i in range(1,3):
...      print i
...
1
2
>>> a=["hello","mike"]
>>> for i in a:
...           print i
...
hello
mike
>>> a=['sachin','dhoni']
>>> for i in a:
...         print 'hello'+i
...
hellosachin
hellodhoni
>>>
>>> a=['sachin','dhoni']
```



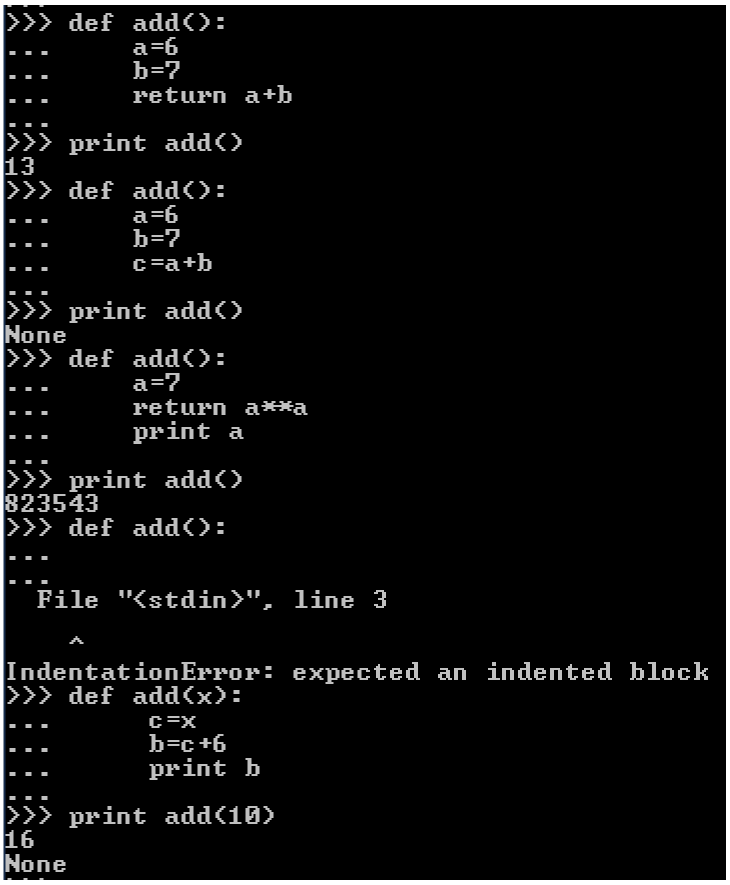
```
>>> i=5
>>> while(i):
...      print i
...      i=i-1
...
5
4
3
2
1
```

Break, continue: These are used with for and while to alter their normal behaviour. When Break statement will be encountered it will terminate the smallest loop it is in and the control will directly move to the statement just below the loop. Continue terminates the current iteration but not the whole loop.

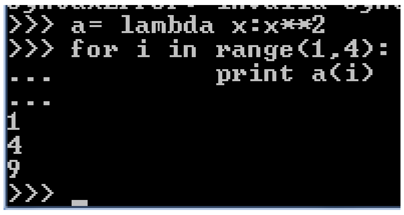


```
>>> for i in range(1,4):
...     if i==5:
...         break
...     print i
...
1
2
3
>>> for i in range(1,4):
...      if i==2:
...          break
...      print i
...
1
>>> for i in range(1,4):
...         if i==2:
...          continue
...         print i
...
1
3
>>> _
```

Def, lambda: def is used to define a user defined function. Lambda is used to create an anonymous function i.e the name of the function is not known. Function is a block of related statements to perform a specific task. It organises the code and do some repetitive task. It is defined once and can be called many times inside the program. It may or may not contain return statement. If it does not contain return statement explicitly then it implicitly returns none. Lambda is associated with an expression. That expression is evaluated and returned. It is an inline function. It does not contain any return statement explicitly. The important parts of function are function definition, function call. Function definition is used to mention the instructions that will be executed repeatedly. Those statements will be executed only when the function will be called.

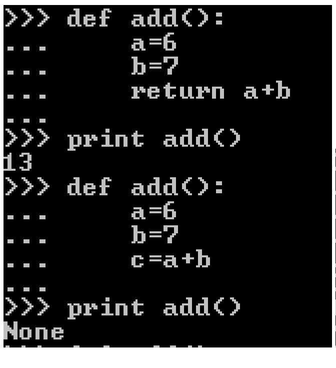


```
>>> def add():
...     a=6
...     b=7
...     return a+b
...
>>> print add()
13
>>> def add():
...     a=6
...     b=7
...     c=a+b
...
>>> print add()
None
>>> def add():
...     a=7
...     return a**a
...     print a
...
>>> print add()
823543
>>> def add():
...
...
  File "<stdin>", line 3

    ^
IndentationError: expected an indented block
>>> def add(x):
...      c=x
...      b=c+6
...      print b
...
>>> print add(10)
16
None
```

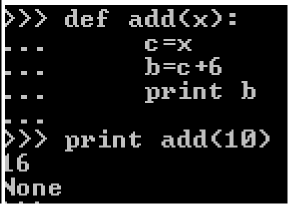


```
>>> a= lambda x:x**2
>>> for i in range(1,4):
...         print a(i)
...
1
4
9
>>> _
```

Return: return statement is used inside a function to exit and return a value. If the return statement is not written explicitly then it will return the' none' implicitly.

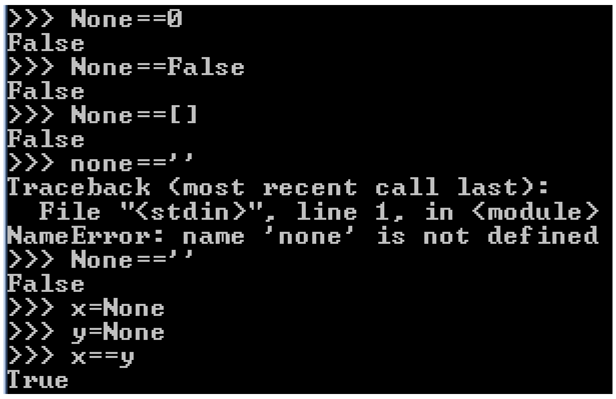


```
>>> def add():
...     a=6
...     b=7
...     return a+b
...
>>> print add()
13
>>> def add():
...     a=6
...     b=7
...     c=a+b
...
>>> print add()
None
```



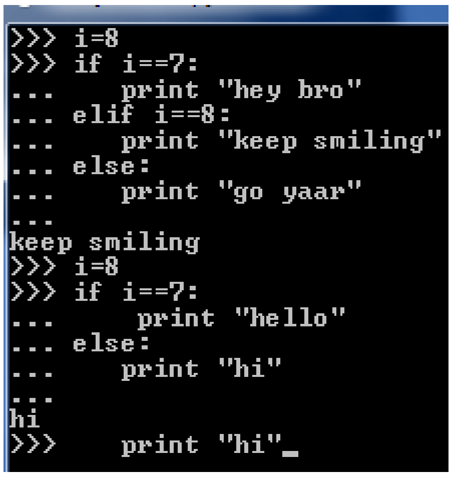
```
>>> def add(x):
...      c=x
...      b=c+6
...      print b
...
>>> print add(10)
16
None
```

None: it is a special constant in python. It represents the absence of a value i.e null in python. It is an object of its own data type i.e NoneType. We cannot create multiple 'none' but we can assign it to variables. And those variables will be equal to each other. If a function does not return any value explicitly then ‘none’ will be returned implicitly. ‘None’ should not imply false, zero, empty list, dictionary or string.

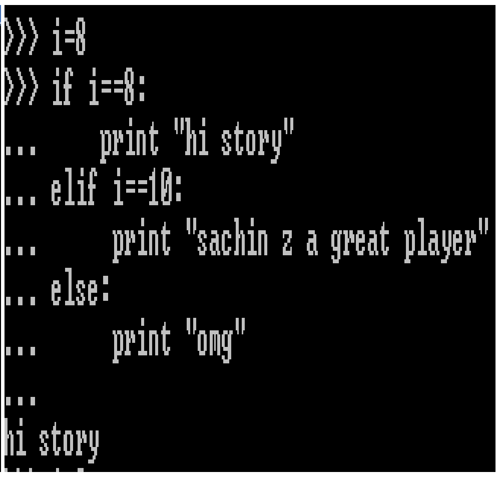


```
>>> None==0
False
>>> None==False
False
>>> None==[]
False
>>> none==''
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'none' is not defined
>>> None==''
False
>>> x=None
>>> y=None
>>> x==y
True
```

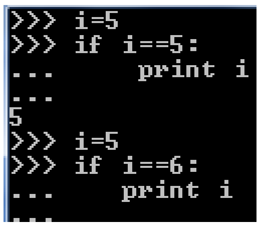
If, else, elif: these are used as selection statement. If we have many choices then for decision making or condition branching these are used. When we want to execute a block of statements if the condition is true then we will use if, elif. If the condition is false then the else statement will be executed. ‘elif’ is used for else if.



```
>>> i=8
>>> if i==7:
...     print "hey bro"
... elif i==8:
...     print "keep smiling"
... else:
...     print "go yaar"
...
keep smiling
>>> i=8
>>> if i==7:
...       print "hello"
... else:
...     print "hi"
...
hi
>>>     print "hi"_
```

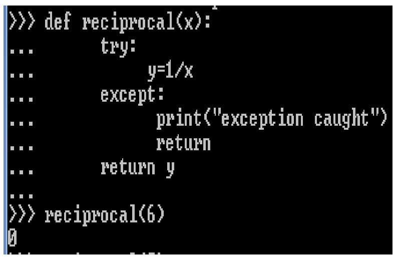


```
>>> i=8
>>> if i==8:
...     print "hi story"
... elif i==10:
...      print "sachin z a great player"
... else:
...      print "omg"
...
hi story
```

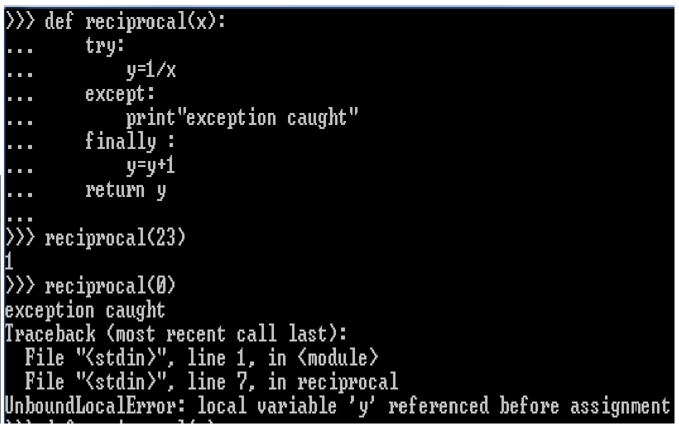


```
>>> i=5
>>> if i==5:
...       print i
...
5
>>> i=5
>>> if i==6:
...     print i
```

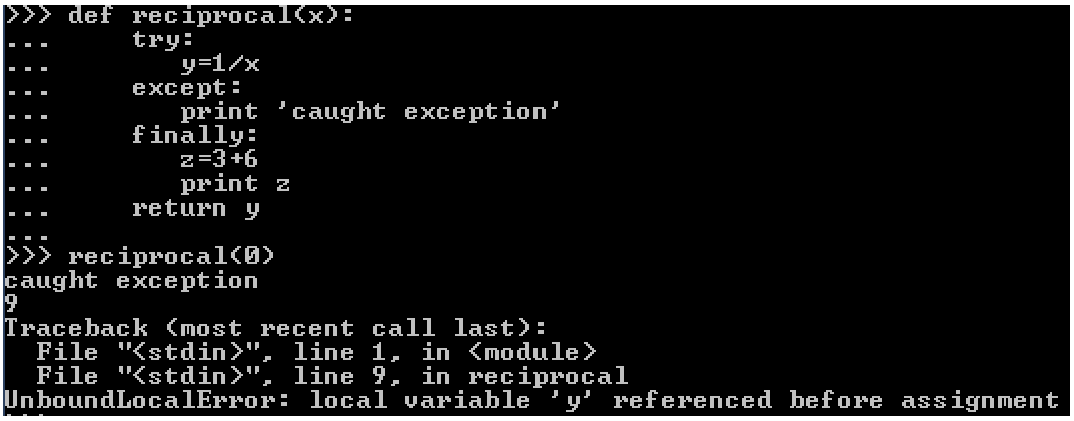
Try, except, raise: these are used with exceptions in python. Exceptions are errors which occur in our program if something goes wrong during execution. Ex of some exceptions: ZeroDivisionError, IOError, ValueError, ImportError, NameError, TypeError. Try....except blocks are used to handle the exceptions in the program. If an exception can be caught then try...except block will return ‘none’. We can also return an exception message with the help of ‘raise’ keyword.



```
>>> def reciprocal(x):
...     try:
...         y=1/x
...     except:
...         print("exception caught")
...         return
...     return y
...
>>> reciprocal(6)
0
```



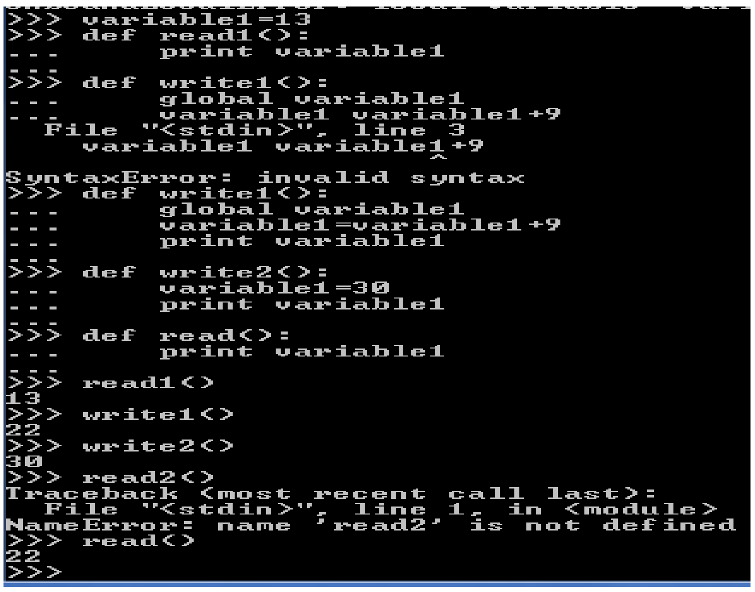
```
>>> def reciprocal(x):
...     try:
...         y=1/x
...     except:
...         print"exception caught"
...     finally :
...         y=y+1
...     return y
...
>>> reciprocal(23)
1
>>> reciprocal(0)
exception caught
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 7, in reciprocal
UnboundLocalError: local variable 'y' referenced before assignment
```



```
>>> def reciprocal(x):
...     try:
...         y=1/x
...     except:
...         print 'caught exception'
...     finally:
...         z=3+6
...         print z
...     return y
...
>>> reciprocal(0)
caught exception
9
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 9, in reciprocal
UnboundLocalError: local variable 'y' referenced before assignment
```

Finally: Finally is used with try....except block to close up resources or file streams. The block of codes inside finally is executed even if there exists some unhandled error or exception.

Global : it is used to define a variable as global inside a function. Because generally the variables inside the functions are local so those cannot be accessed outside. If we want to read the value of a global variable then it is not required to declare it inside the function but if we want to modify the value of a global variable inside a function then we must declare it as global inside the corresponding function. If we will not mention global and create a variable then that will be a local variable.



```
>>> variable1=13
>>> def read1():
...     print variable1
...
>>> def write1():
...     global variable1
...     variable1 variable1+9
  File "<stdin>", line 3
    variable1 variable1+9
                      ^
SyntaxError: invalid syntax
>>> def write1():
...     global variable1
...     variable1=variable1+9
...     print variable1
...
>>> def write2():
...     variable1=30
...     print variable1
...
>>> def read():
...     print variable1
...
>>> read1()
13
>>> write1()
22
>>> write2()
30
>>> read2()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'read2' is not defined
>>> read()
22
>>>
```

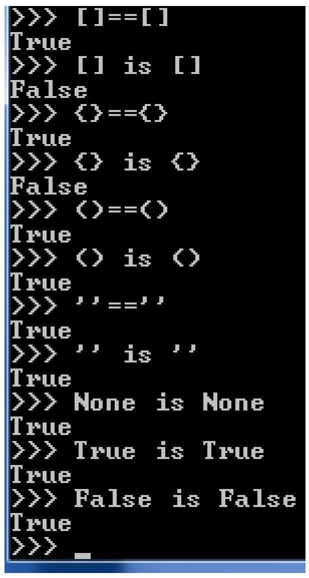
Nonlocal: it is used to define that a variable inside a nested function (function inside function) is not local to that function. If we want to change the value of the non-local variable inside the nested function then we need to declare it as nonlocal. Otherwise a local variable with a same name is created inside the method.

Class: class is used to create new user defined class. It is a collection of related attributes and functions that try to represent a real time situation. It is a concept of object oriented programming.

Ex: class ExampleClass:

```
        def fun1(parameters):

            .............

        Def fun2 (parameters):

            .............
```

Is: It is used to check the identity of an object while the= = operator is used to test whether two variables are equal or not. 'Is' is used to check if two variables refer to same object or not. It returns true if the variables are identical else returns false. None, True, False refer to only one object so they will always give true. List, and Dictionary are mutable. There value can be changed. If value will be changed they can refer to two objects in memory. So they will give False while compared through 'is' operator. But string and tuple are immutable. We cannot change their value once defined. So they will give True while compared through 'is' operator.

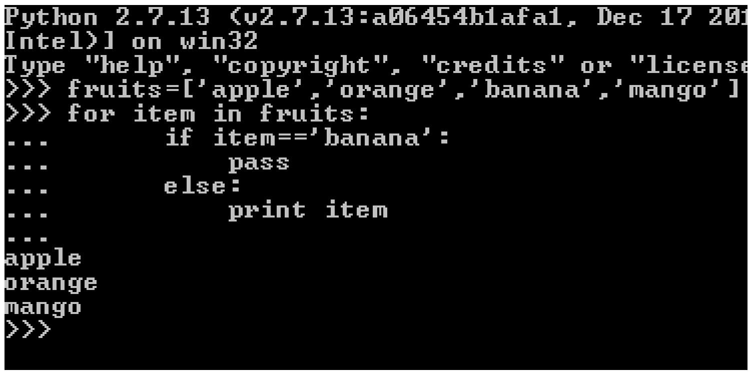


```
>>> []==[]
True
>>> [] is []
False
>>> {}=={}
True
>>> {} is {}
False
>>> ()==()
True
>>> () is ()
True
>>> ''==''
True
>>> '' is ''
True
>>> None is None
True
>>> True is True
True
>>> False is False
True
>>> _
```

Pass: pass statement is a null statement in python. It works like a place holder. Nothing happens when it is executed. If we want to implement a function in near future and it is not implemented yet. If we will write like

>>> Def future (parameters): Then it will give indentation error in the middle of the program. So we will write as:

```
>>>def future(parameters):
            Pass
```

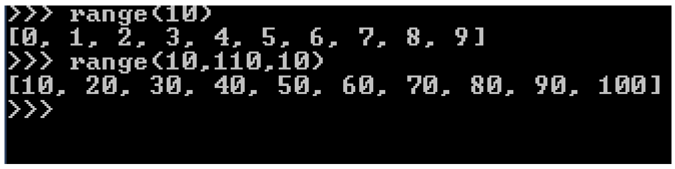


```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 201
Intel)] on win32
Type "help", "copyright", "credits" or "license
>>> fruits=['apple','orange','banana','mango']
>>> for item in fruits:
...         if item=='banana':
...             pass
...         else:
...             print item
...
apple
orange
mango
>>>
```

Note: We can use this while finding whether the given number is prime or not.

Same thing can be done with a class.

Range: This statement in python a list of numbers in arithmetic progression.



```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(10,110,10)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>>
```

Limitations and disadvantages of conventional network architecture:

1. Complex Network Devices
2. Management Overhead
3. Limited Scalability

Complex Network Devices:

1. The protocol implementation overhead to improve link speed and reliability and most of the protocols are designed for specific applications.
2. It supports limited interoperability due to lack of standard and open interfaces i.e hardware and software of network are proprietary.
3. It has very slow product life cycles which limit the opportunity of innovations.
4. It is well suited for static traffic patterns but difficult to implement the protocols for dynamic traffic patterns as in IoT and Cloud.

Management Overhead:

1. As it does not support interoperability so it is difficult to manage multiple network devices and interfaces from multiple vendors.
2. For upgrading the networks it needs configuration changes in multiple devices like switches, routers, firewalls etc.

Limited scalability:

It is difficult to implement virtualisation, implementation of distributed algorithm and data analytics for distributed applications and big data with minimal manual configurations.

NFV:(Network Function Virtualisation):

- It is a technology which gets maximum benefits by providing virtualisation to the industry standard high volume servers, switches and storages. So it reduces the hardware cost and power consumption. It comprises of the software implemented functions that can run in the cloud. So it is able to differentiate among hardware and software.
- NFV provides infrastructure on which SDN can run. SDN and NFV are complementary to each other. But they can work independently.
- Hardware remains same so network functions can be easily tested and upgraded.
- NFV is applicable to fixed and mobile network control plane and data plane.

The key components of NFV architecture:

- Virtualized Network Function(VNF)
- NFV infrastructure(NFVI)
- NFV management and orchestration

VNF:

- It is the software implementation of the network functionality.
- This can run over NFV in cloud so can be shared for multiple network services.

NFVI:

- It includes the resources to be virtualised such as computing, network and storage resources.

NFV Management and orchestration:

- It manages the task related to virtualisation.
- It deals with the lifecycle management of physical and software resources, VNFs.

EX: Use of NFV for virtualisation of home networks:

The home network includes:

- Home Gateway
- IP enabled devices

1. Home gateway provides wide area network connectivity to the IP enabled devices such as IPTV, VoIP.
2. Various functionalities of home gateway are: DHCP (Dynamic Host Configuration Protocol), NAT (Network Address Translation), application specific gateway and Firewall.

3. Through the help of the DHCP the IP enabled devices will be able to get IP addresses whenever they will switch on and connected to the home gateway. DHCP is configured in the Home Gateway.
4. Whatever the devices will be connected to the home network they will get the private address. So to communicate with the internet i.e outer world they need a public IP. That public IP is configured at the external interface of the home gateway. So whenever the devices will try to connect to the internet the private address will be translated to one public address through the help of the NAT.
5. The gateway provides application specific routing for applications such as VOIP and IPTV.

Difference between SDN and NFV:

Services	**SDN**	**NFV**
Abbreviation	1. Software Defined Network	1.Network Function Virtualisation
Basic Idea	**2.** SDN separates control plane from data plane and centralizes the control and programmability of the network	2. NFV transfers dedicated application to generic servers.
Areas of Operations	3. SDN operates in data centre, campus and/or cloud.	3. It operates in service provider network or operators.
Initial application Target	4. SDN targets cloud or orchestrations and networking	4. NFV targets router, firewalls, gateways, WAN, CDN, accelerators and SLA assurance.
Protocols	5. Open Flow	5. None
Supporting Organisation	6. Open Networking Foundation.	6. ETSI NFV working group

What is IoT orchestration?

1. It is the process of integrating IoT applications with enterprise IT systems, cloud services, mobile applications within and across the boundaries of the company.
2. It is typically used to automate or improve process, and potentially synchronise data all in real-time.

Characteristics of M2M:

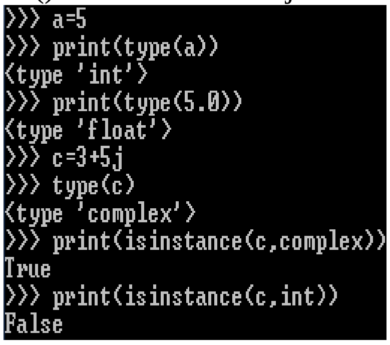
1. Point to point communication usually embedded within hardware at the customer site. Ex (Tap to pay, NFC). The service in Samsung taps to pay without using card. You just need to bring your mobile near the swipe machine. Using your ID card to open or close the door.
2. It uses cellular or wired network.
3. Do not necessarily require internet connection.
4. It is more hardware based.
5. On premises applications. (i.e software and technology i.e located within the physical confines of an enterprise or company's data centre.) Ex. Diagnosis applications, Service management applications and on-premises enterprises applications.
6. Data collection and analysis is done in on premises storage infrastructure.
7. Non IP based communication.
8. M2M protocols include ZigBee, Bluetooth, ModBus, M-Bus, Wireless M-Bus, Power Line communication (PLC), 6LoWPAN, IEEE 802.15.4, Z-wave. These protocols work below the network layer.

Data Type:

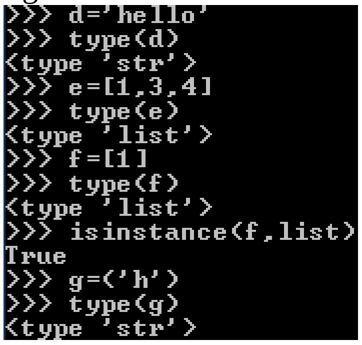
Every value in python has a data type. Generally data types are classes and the variables are object in python.

type() :is used to know which class a variable or value belongs to.

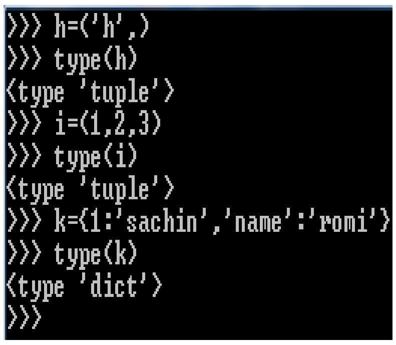
isinstance(): to check if an object belongs to a class.



```
>>> a=5
>>> print(type(a))
<type 'int'>
>>> print(type(5.0))
<type 'float'>
>>> c=3+5j
>>> type(c)
<type 'complex'>
>>> print(isinstance(c,complex))
True
>>> print(isinstance(c,int))
False
```



```
>>> d='hello'
>>> type(d)
<type 'str'>
>>> e=[1,3,4]
>>> type(e)
<type 'list'>
>>> f=[1]
>>> type(f)
<type 'list'>
>>> isinstance(f,list)
True
>>> g=('h')
>>> type(g)
<type 'str'>
```



```
>>> h=('h',)
>>> type(h)
<type 'tuple'>
>>> i=(1,2,3)
>>> type(i)
<type 'tuple'>
>>> k={1:'sachin','name':'romi'}
>>> type(k)
<type 'dict'>
>>>
```

Some of the basic data types are as follows.

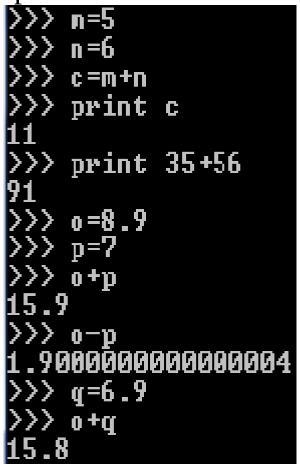
1. Number Data Type: Int, float, complex.
2. Sequence Data types: String, list, tuple, dictionary, set

<u>Number Data Type: it is an immutable data type.</u>

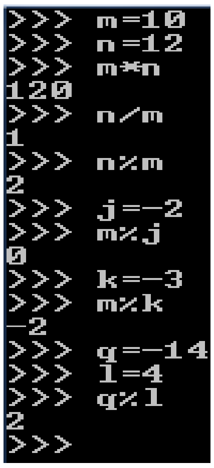
Int: integers can be of any length and it is only limited by the memory available.

Float: a decimal point number is accurate up to 15 decimal places i.e 16th place is inaccurate. It can be rounded one.

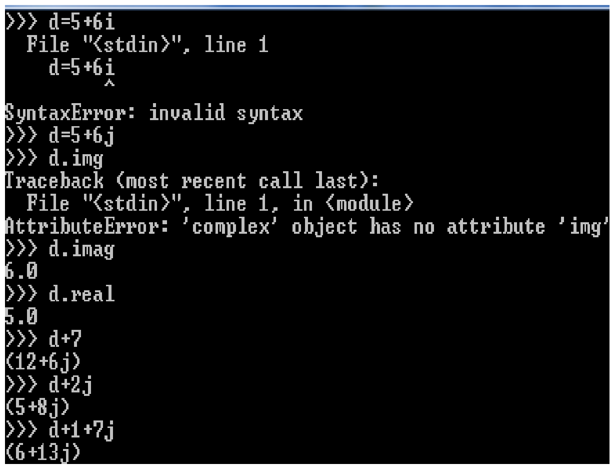
Complex: complex numbers are written in the form of X+Yj, where X is the real part and y is the imaginary part.



```
>>> m=5
>>> n=6
>>> c=m+n
>>> print c
11
>>> print 35+56
91
>>> o=8.9
>>> p=7
>>> o+p
15.9
>>> o-p
1.9000000000000004
>>> q=6.9
>>> o+q
15.8
```



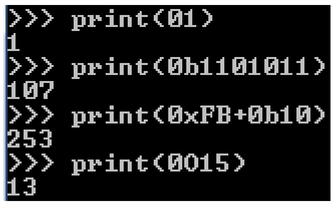
```
>>> m=10
>>> n=12
>>> m*n
120
>>> n/m
1
>>> n%m
2
>>> j=-2
>>> m%j
0
>>> k=-3
>>> m%k
-2
>>> q=-14
>>> l=4
>>> q%l
2
>>>
```



```
>>> d=5+6i
  File "<stdin>", line 1
    d=5+6i
         ^
SyntaxError: invalid syntax
>>> d=5+6j
>>> d.img
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'complex' object has no attribute 'img'
>>> d.imag
6.0
>>> d.real
5.0
>>> d+7
(12+6j)
>>> d+2j
(5+8j)
>>> d+1+7j
(6+13j)
```

Note: Numbers we deal with everyday are decimal (base 10) number system. But computer programmers (generally embedded programmer) need to work with binary (base 2), hexadecimal (base 16) and octal (base 8) number systems. In Python, we can represent these numbers by appropriately placing a prefix before that number. Following table lists these prefix.

Number system	Prefix
Binary	0b or 0B
octal	0o or 0O
Hexadecimal	0x or 0X

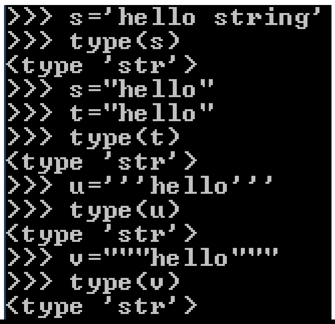


```
>>> print(01)
1
>>> print(0b1101011)
107
>>> print(0xFB+0b10)
253
>>> print(0015)
13
```

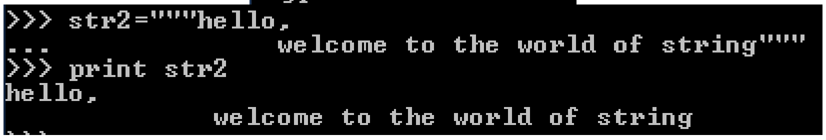
<u>Sequence Data Type:</u>

String:

1. string is a sequence of Unicode characters. There are no limits to the number of characters in a string. A string with zero characters is known as an empty string.
2. We can use single quotes, ‘ ’ or double quotes “ ” to represent characters. Multi line string can be denoted using triple quotes, ’ ’ ’ or “ ” ”.

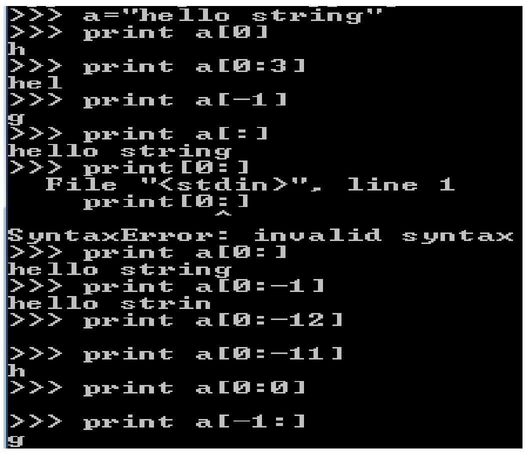


```
>>> s='hello string'
>>> type(s)
<type 'str'>
>>> s="hello"
>>> t="hello"
>>> type(t)
<type 'str'>
>>> u='''hello'''
>>> type(u)
<type 'str'>
>>> v="""hello"""
>>> type(v)
<type 'str'>
```



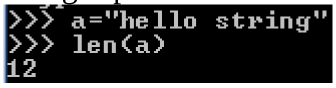
```
>>> str2="""hello,
...               welcome to the world of string"""
>>> print str2
hello,
               welcome to the world of string
```

3. Slicing operator i.e colon ‘:’ and index operator ‘[]’ can be used in string to retrieve range of character from the existing string. String is immutable i.e once it is defined, memory size will be fixed. You cannot append anything to the existing string and you cannot change/update any of the character of the string.
4. Index operator i.e pair of square braces’ []’ can be used in string to retrieve a character from the existing string. Indexing can be +ve indexing and –ve indexing. +ve indexing starts from 0 i.e from left to right and –ve indexing starts from -1 from the right to left. Index value must be integer type.



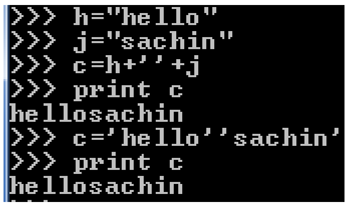
```
>>> a="hello string"
>>> print a[0]
h
>>> print a[0:3]
hel
>>> print a[-1]
g
>>> print a[:]
hello string
>>> print[0:]
  File "<stdin>", line 1
    print[0:]
           ^
SyntaxError: invalid syntax
>>> print a[0:]
hello string
>>> print a[0:-1]
hello strin
>>> print a[0:-12]

>>> print a[0:-11]
h
>>> print a[0:0]

>>> print a[-1:]
g
```

5. ‘ len()‘is used to find the length of the string. Space is also counted.



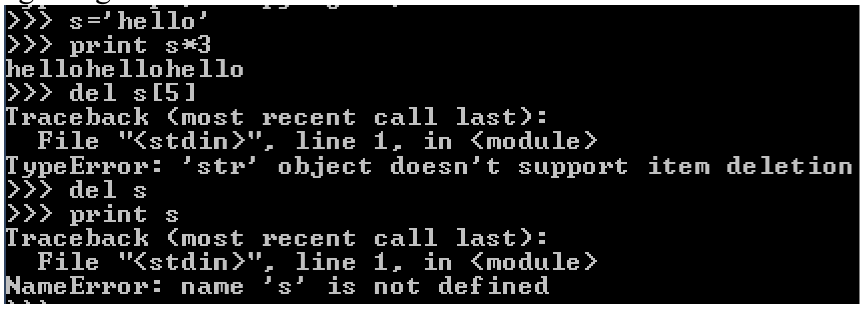
```
>>> a="hello string"
>>> len(a)
12
```

6. ‘+ ‘operator is used to concatenate two strings.



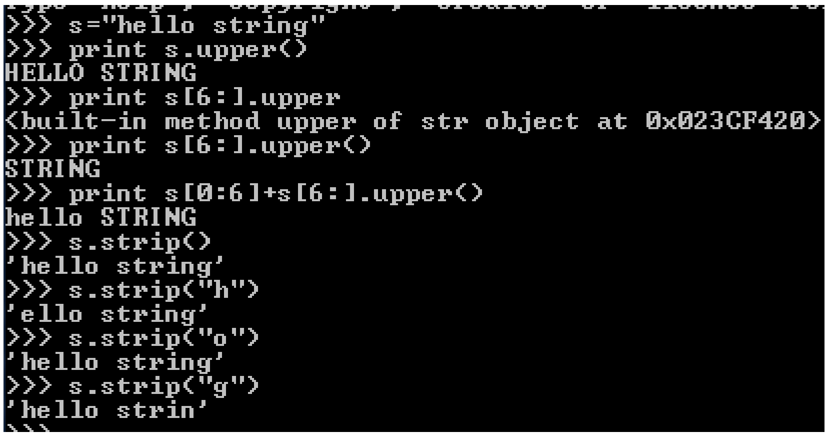
```
>>> h="hello"
>>> j="sachin"
>>> c=h+''+j
>>> print c
hellosachin
>>> c='hello''sachin'
>>> print c
hellosachin
```

7. The elements of list can repeated for the mentioned number of times using '*' operator. We can delete the entire string using 'del'.



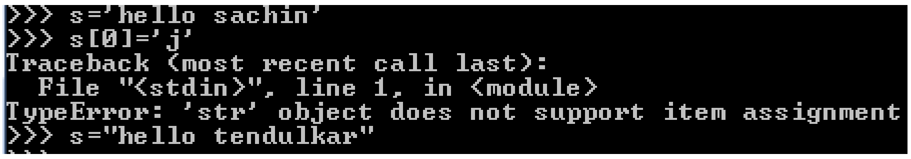
```
>>> s='hello'
>>> print s*3
hellohellohello
>>> del s[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object doesn't support item deletion
>>> del s
>>> print s
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 's' is not defined
```

8. Convert to uppercase or lower case we can use upper() or lower().
9. We can use' strip()' to return a copy of the string with the trailing and leading characters removed.



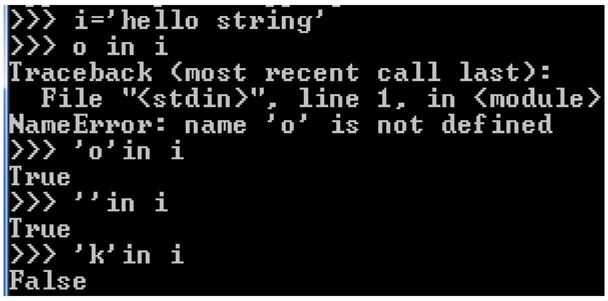
```
>>> s="hello string"
>>> print s.upper()
HELLO STRING
>>> print s[6:].upper
<built-in method upper of str object at 0x023CF420>
>>> print s[6:].upper()
STRING
>>> print s[0:6]+s[6:].upper()
hello STRING
>>> s.strip()
'hello string'
>>> s.strip("h")
'ello string'
>>> s.strip("o")
'hello string'
>>> s.strip("g")
'hello strin'
```

10. Strings are immutable i.e once defined memory is fixed and value is fixed. So changing the value of a string, appending and removing the characters to/from the existing string is not possible. But we can assign a new string to the existing with the help of the assignment operator.

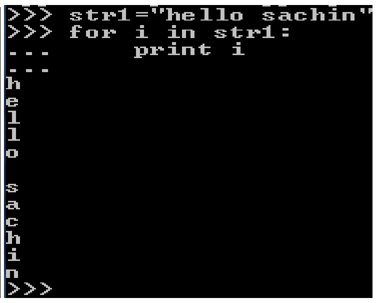


```
>>> s='hello sachin'
>>> s[0]='j'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> s="hello tendulkar"
```

11. 'In' is used to check the members of the list. It is also used to traverse the entire list.



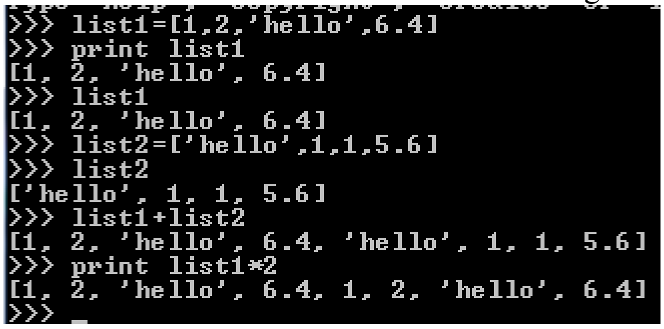
```
>>> i='hello string'
>>> o in i
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'o' is not defined
>>> 'o'in i
True
>>> ''in i
True
>>> 'k'in i
False
```



```
>>> str1="hello sachin"
>>> for i in str1:
...     print i
...
h
e
l
l
o

s
a
c
h
i
n
>>>
```

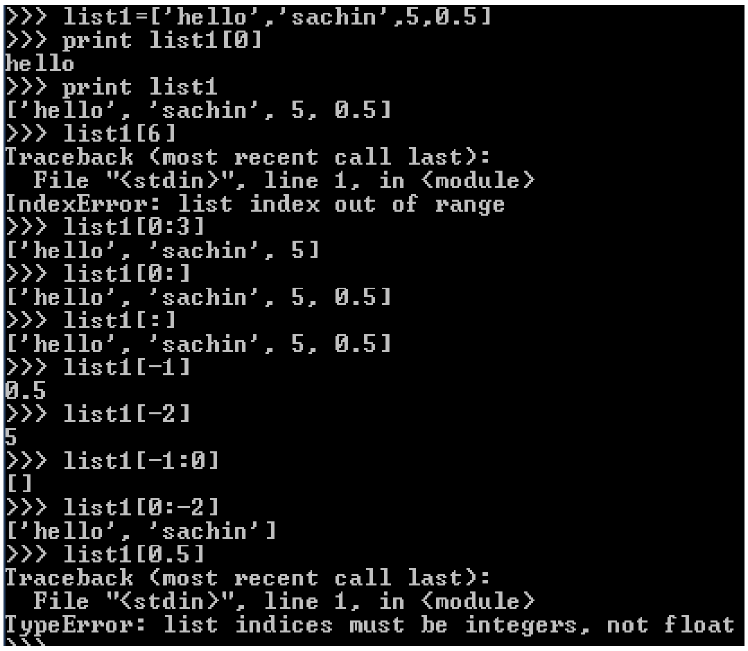
List:
1. List is a compound data type used to group together other values. List items can be of any data type. It contains items separated by commas and enclosed within square brackets.

2. List allows duplication of items.
3. Two lists can be combined or merged using ' +' operator.
4. The elements of list can repeated for the mentioned number of times using '*' operator.



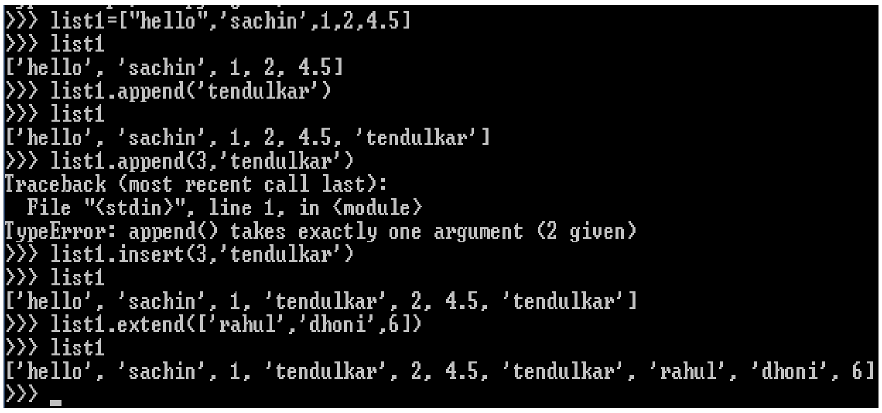
```
>>> list1=[1,2,'hello',6.4]
>>> print list1
[1, 2, 'hello', 6.4]
>>> list1
[1, 2, 'hello', 6.4]
>>> list2=['hello',1,1,5.6]
>>> list2
['hello', 1, 1, 5.6]
>>> list1+list2
[1, 2, 'hello', 6.4, 'hello', 1, 1, 5.6]
>>> print list1*2
[1, 2, 'hello', 6.4, 1, 2, 'hello', 6.4]
>>> _
```

5. Slicing operator ':' and index operator '[]' can be used to extract an item or range of items from the list. In slicing operator we use the index value to retrieve any element. Index value refers to the position of the element in a list. The index can only be an integer. It can never be floating. If the index will be other than integer then it will give TypeError. Type Indexing can be done in two ways i.e +ve indexing and –ve indexing. +ve indexing starts from zero i.e from left to right in python. –ve index starts from -1 i.e from right to left. If you will try to retrieve the elements out of the index range then it will show IndexError.



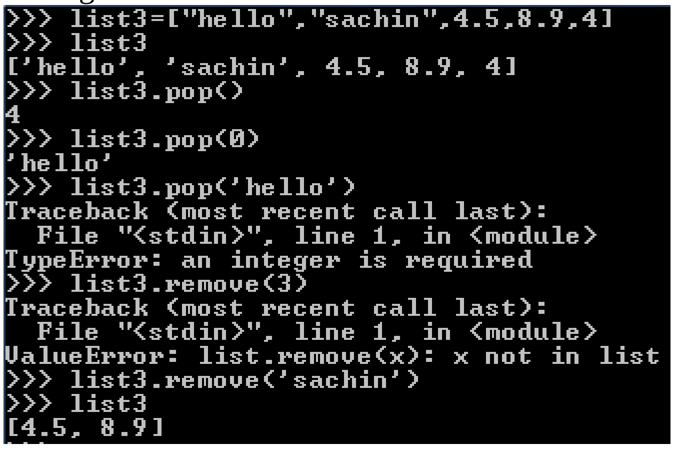
```
>>> list1=['hello','sachin',5,0.5]
>>> print list1[0]
hello
>>> print list1
['hello', 'sachin', 5, 0.5]
>>> list1[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> list1[0:3]
['hello', 'sachin', 5]
>>> list1[0:]
['hello', 'sachin', 5, 0.5]
>>> list1[:]
['hello', 'sachin', 5, 0.5]
>>> list1[-1]
0.5
>>> list1[-2]
5
>>> list1[-1:0]
[]
>>> list1[0:-2]
['hello', 'sachin']
>>> list1[0.5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers, not float
```

6. Len(): is used to find the number of items present in the existing list.
7. It is mutable i.e the memory size is not fixed. You can append , remove any item to/from the existing list. We can also change the individual elements of the existing list. We will use assignment operator to change an item or range of item.
8. append()is used to add an item at the last position or the index of the existing list.
9. Etend(): is used to add several items at a time at the last position or the index of the existing list.
10. Insert() is used to add an item at the desired position of the existing list.



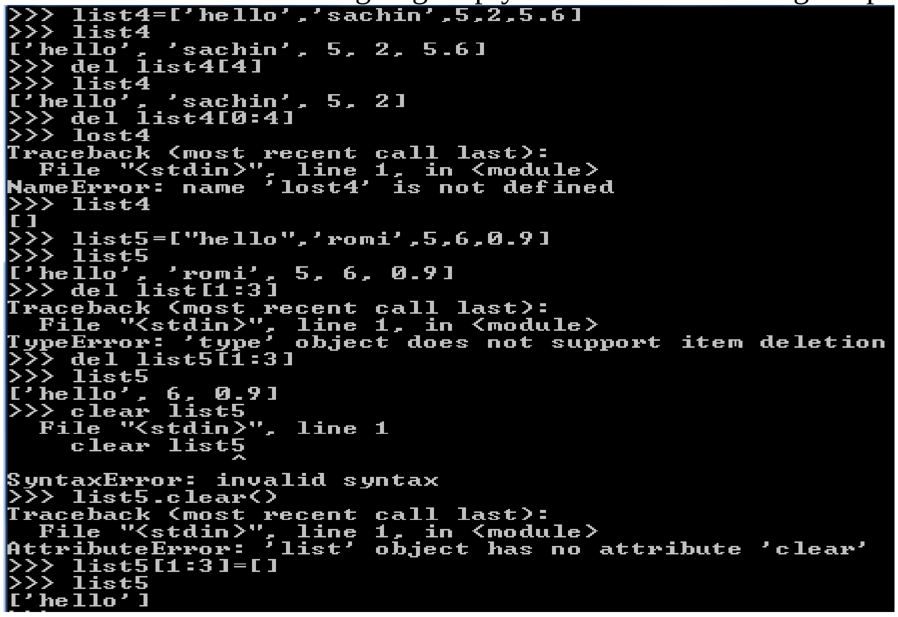
```
>>> list1=["hello",'sachin',1,2,4.5]
>>> list1
['hello', 'sachin', 1, 2, 4.5]
>>> list1.append('tendulkar')
>>> list1
['hello', 'sachin', 1, 2, 4.5, 'tendulkar']
>>> list1.append(3,'tendulkar')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: append() takes exactly one argument (2 given)
>>> list1.insert(3,'tendulkar')
>>> list1
['hello', 'sachin', 1, 'tendulkar', 2, 4.5, 'tendulkar']
>>> list1.extend(['rahul','dhoni',6])
>>> list1
['hello', 'sachin', 1, 'tendulkar', 2, 4.5, 'tendulkar', 'rahul', 'dhoni', 6]
>>> _
```

11. Remove()is used to remove an item from the existing list. It can be from any position or index and assign the remaining items into the existing list.
12. Pop() is used to remove an item from a given location of the existing list and assign the remaining items into the existing list.



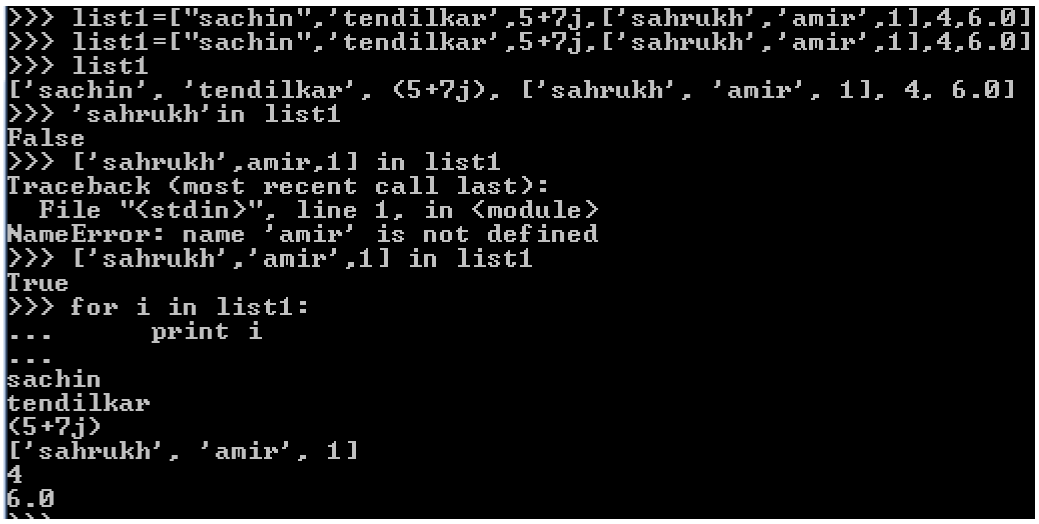
```
>>> list3=["hello","sachin",4.5,8.9,4]
>>> list3
['hello', 'sachin', 4.5, 8.9, 4]
>>> list3.pop()
4
>>> list3.pop(0)
'hello'
>>> list3.pop('hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: an integer is required
>>> list3.remove(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
>>> list3.remove('sachin')
>>> list3
[4.5, 8.9]
```

13. Del(): is used to delete an element from the existing list or to delete an entire list.
14. Clear(): is used to clear all the elements of the existing list.(it is not supported in 2.7 but in higher versions).
15. We can also delete the elements of a list assigning empty list to the desired range of position.



```
>>> list4=['hello','sachin',5,2,5.6]
>>> list4
['hello', 'sachin', 5, 2, 5.6]
>>> del list4[4]
>>> list4
['hello', 'sachin', 5, 2]
>>> del list4[0:4]
>>> lost4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'lost4' is not defined
>>> list4
[]
>>> list5=["hello",'romi',5,6,0.9]
>>> list5
['hello', 'romi', 5, 6, 0.9]
>>> del list[1:3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'type' object does not support item deletion
>>> del list5[1:3]
>>> list5
['hello', 6, 0.9]
>>> clear list5
  File "<stdin>", line 1
    clear list5
              ^
SyntaxError: invalid syntax
>>> list5.clear()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'list' object has no attribute 'clear'
>>> list5[1:3]=[]
>>> list5
['hello']
```

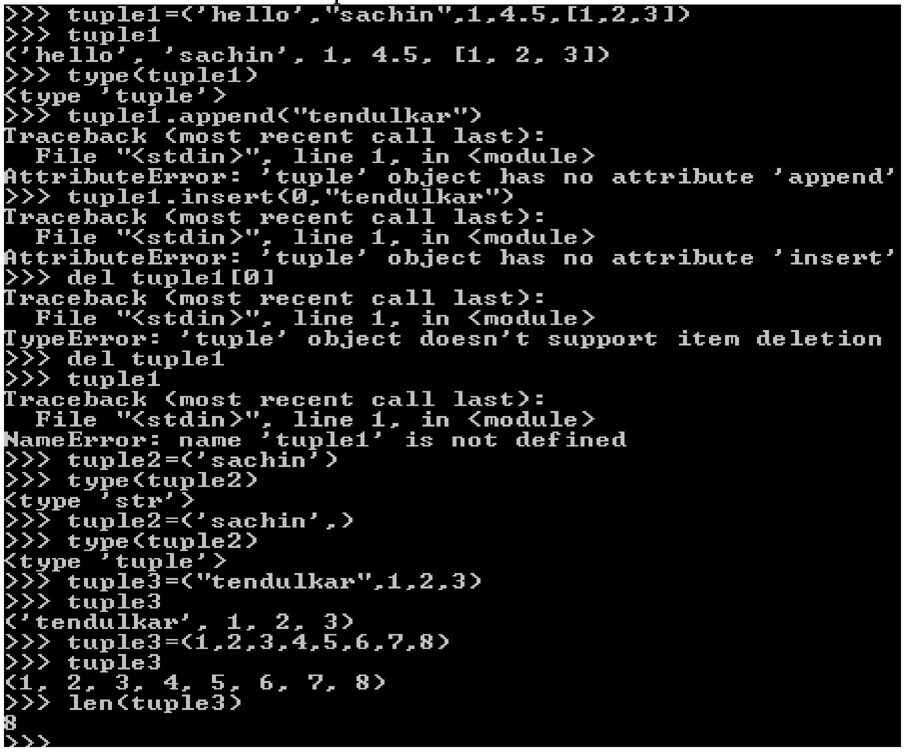
16. 'In' is used to check the members of the list. It is also used to traverse the entire list.
17. Lists can be nested i.e another list can be an item in the existing list.



```
>>> list1=["sachin",'tendilkar',5+7j,['sahrukh','amir',1],4,6.0]
>>> list1=["sachin",'tendilkar',5+7j,['sahrukh','amir',1],4,6.0]
>>> list1
['sachin', 'tendilkar', (5+7j), ['sahrukh', 'amir', 1], 4, 6.0]
>>> 'sahrukh'in list1
False
>>> ['sahrukh',amir,1] in list1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'amir' is not defined
>>> ['sahrukh','amir',1] in list1
True
>>> for i in list1:
...       print i
...
sachin
tendilkar
(5+7j)
['sahrukh', 'amir', 1]
4
6.0
```

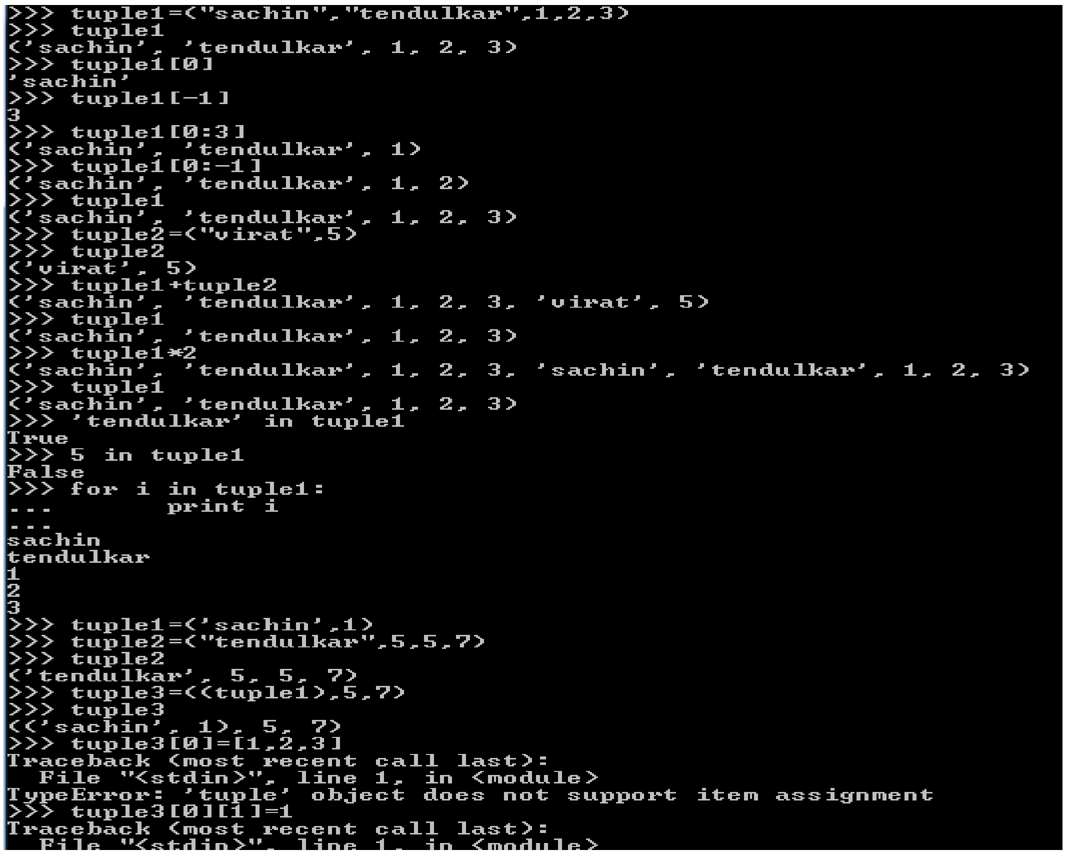
Tuples:

1. A tuple consists of a number of values separated by commas and enclosed within parentheses.
2. To create a tuple with single element we need to give a trailing comma.
3. A tuple is a sequence data type that is similar to the list. But it is immutable i.e if once defined then memory will be fixed. You cannot append, remove items to /from the existing list you cannot alter the value of the existing tuple . But we can delete the entire tuple using 'del' key word.
4. You cannot alter the items in a tuple once it is defined i.e can be thought as read-only list, but you can assign another tuple to the previous.
5. Len(): is used to find the number of elements present inside list.



```
>>> tuple1=('hello',"sachin",1,4.5,[1,2,3])
>>> tuple1
('hello', 'sachin', 1, 4.5, [1, 2, 3])
>>> type(tuple1)
<type 'tuple'>
>>> tuple1.append("tendulkar")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>> tuple1.insert(0,"tendulkar")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'insert'
>>> del tuple1[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
>>> del tuple1
>>> tuple1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'tuple1' is not defined
>>> tuple2=('sachin')
>>> type(tuple2)
<type 'str'>
>>> tuple2=('sachin',)
>>> type(tuple2)
<type 'tuple'>
>>> tuple3=("tendulkar",1,2,3)
>>> tuple3
('tendulkar', 1, 2, 3)
>>> tuple3=(1,2,3,4,5,6,7,8)
>>> tuple3
(1, 2, 3, 4, 5, 6, 7, 8)
>>> len(tuple3)
8
>>>
```

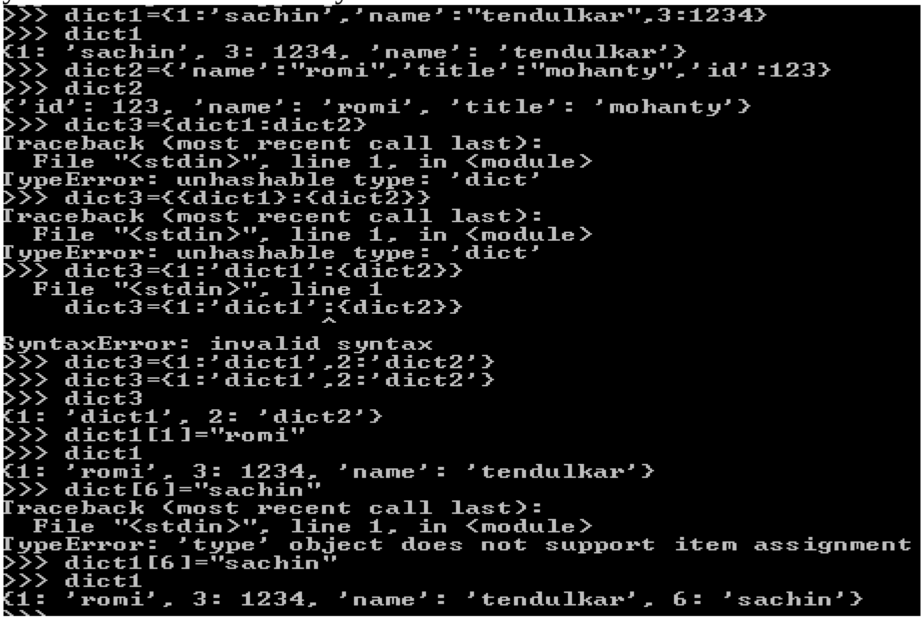
6. Slicing operator':' and indexing operator' []' is used to retrieve an element or group of elements from the existing tuple.
7. Two tuples can be merged or combined or concatenate using '+' operator and new memory location will be allocated to the resultant tuple.
8. The elements of list can be repeated for the mentioned number of times using '*' operator.
9. 'In' is used to check the members of the list. It is also used to traverse the entire tuple.
10. Tuple can be nested i.e another tuple is an element of the existing tuple.
11. It allows duplication of elements.



```
>>> tuple1=("sachin","tendulkar",1,2,3)
>>> tuple1
('sachin', 'tendulkar', 1, 2, 3)
>>> tuple1[0]
'sachin'
>>> tuple1[-1]
3
>>> tuple1[0:3]
('sachin', 'tendulkar', 1)
>>> tuple1[0:-1]
('sachin', 'tendulkar', 1, 2)
>>> tuple1
('sachin', 'tendulkar', 1, 2, 3)
>>> tuple2=("virat",5)
>>> tuple2
('virat', 5)
>>> tuple1+tuple2
('sachin', 'tendulkar', 1, 2, 3, 'virat', 5)
>>> tuple1
('sachin', 'tendulkar', 1, 2, 3)
>>> tuple1*2
('sachin', 'tendulkar', 1, 2, 3, 'sachin', 'tendulkar', 1, 2, 3)
>>> tuple1
('sachin', 'tendulkar', 1, 2, 3)
>>> 'tendulkar' in tuple1
True
>>> 5 in tuple1
False
>>> for i in tuple1:
...         print i
...
sachin
tendulkar
1
2
3
>>> tuple1=('sachin',1)
>>> tuple2=("tendulkar",5,5,7)
>>> tuple2
('tendulkar', 5, 5, 7)
>>> tuple3=((tuple1),5,7)
>>> tuple3
(('sachin', 1), 5, 7)
>>> tuple3[0]=[1,2,3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> tuple3[0][1]=1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

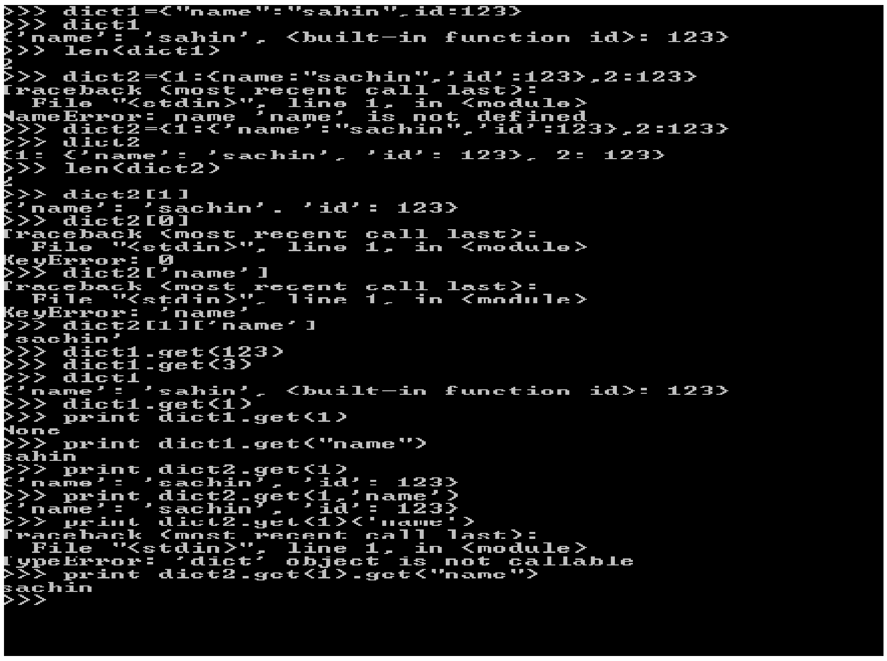
Dictionaries:

1. It is a mapping data type or a kind of hash table that maps keys to values and enclosed with curly braces. It is an unordered collection of values.
2. Keys in a dictionary can be of immutable data type like number, string and tuple and must be unique.
3. Value in a dictionary can be of any data type or object.
4. Dictionary is mutable so its values can be changed, with the help of the assignment operator. If the value is not present then a new key: value pair will be created.
5. A value in a dictionary can be another dictionary.



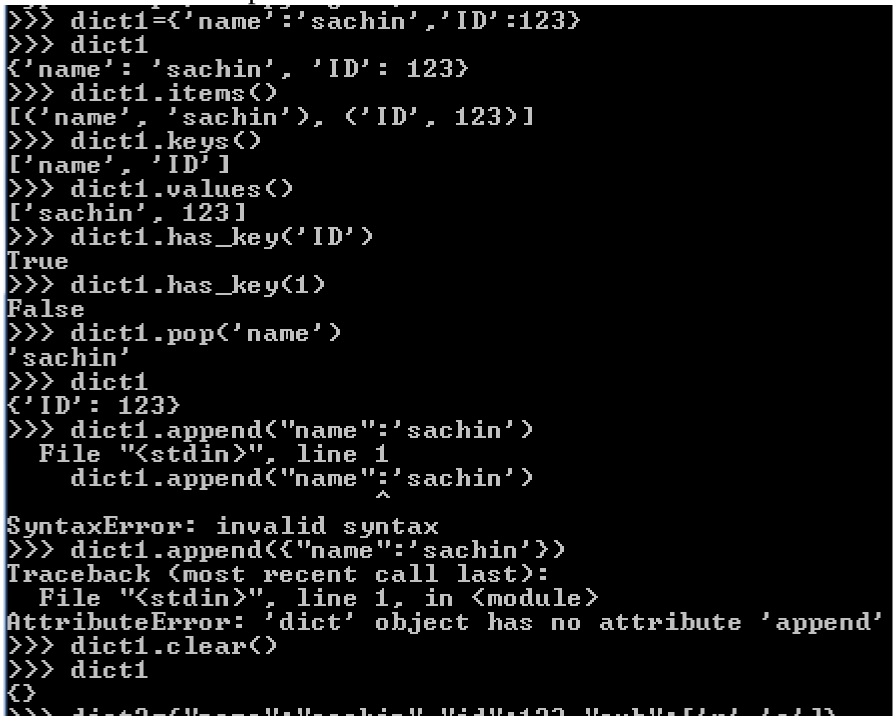
```
>>> dict1={1:'sachin','name':"tendulkar",3:1234}
>>> dict1
{1: 'sachin', 3: 1234, 'name': 'tendulkar'}
>>> dict2={'name':"romi",'title':"mohanty",'id':123}
>>> dict2
{'id': 123, 'name': 'romi', 'title': 'mohanty'}
>>> dict3={dict1:dict2}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'dict'
>>> dict3={{dict1}:{dict2}}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'dict'
>>> dict3={1:'dict1':{dict2}}
  File "<stdin>", line 1
    dict3={1:'dict1':{dict2}}
                    ^
SyntaxError: invalid syntax
>>> dict3={1:'dict1',2:'dict2'}
>>> dict3={1:'dict1',2:'dict2'}
>>> dict3
{1: 'dict1', 2: 'dict2'}
>>> dict1[1]="romi"
>>> dict1
{1: 'romi', 3: 1234, 'name': 'tendulkar'}
>>> dict[6]="sachin"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'type' object does not support item assignment
>>> dict1[6]="sachin"
>>> dict1
{1: 'romi', 3: 1234, 'name': 'tendulkar', 6: 'sachin'}
>>>
```

6. len(): is used to find the number of elements in the existing dictionary.
7. Indexing operator [] is used to retrieve an element from the existing dictionary with the help of key.
8. Get(): is used to access the values from the dictionary with the help of the key. If key is not present then it will give None instead of key-error.



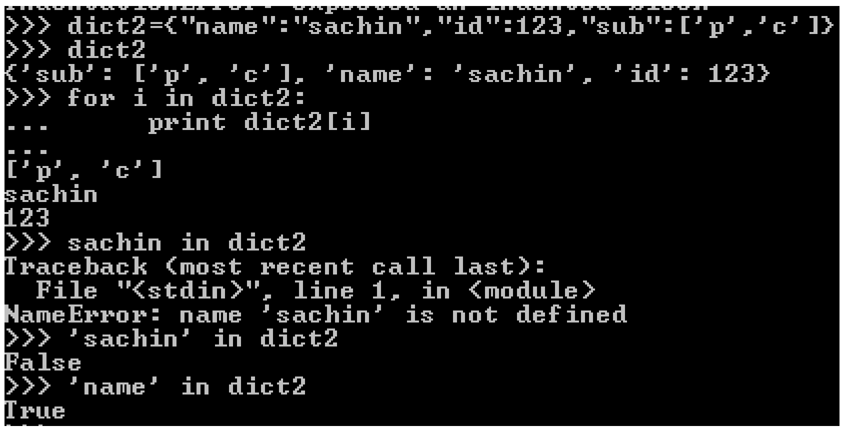
```
>>> dict1={"name":"sahin",id:123}
>>> dict1
{'name': 'sahin', <built-in function id>: 123}
>>> len(dict1)
2
>>> dict2={1:{name:"sachin",'id':123},2:123}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'name' is not defined
>>> dict2={1:{'name':"sachin",'id':123},2:123}
>>> dict2
{1: {'name': 'sachin', 'id': 123}, 2: 123}
>>> len(dict2)
2
>>> dict2[1]
{'name': 'sachin', 'id': 123}
>>> dict2[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
>>> dict2['name']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'name'
>>> dict2[1]['name']
'sachin'
>>> dict1.get(123)
>>> dict1.get(3)
>>> dict1
{'name': 'sahin', <built-in function id>: 123}
>>> dict1.get(1)
>>> print dict1.get(1)
None
>>> print dict1.get("name")
sahin
>>> print dict2.get(1)
{'name': 'sachin', 'id': 123}
>>> print dict2.get(1,'name')
{'name': 'sachin', 'id': 123}
>>> print dict2.get(1)('name')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'dict' object is not callable
>>> print dict2.get(1).get("name")
sachin
>>>
```

9. Items (): is used to get all the items in a dictionary.
10. Keys(): is used to get all the keys in a dictionary.
11. Values(): is used to get all the values in a dictionary.
12. Has_key (): is used to check if dictionary has a key. If present then it returns true otherwise returns false.
13. Pop(): is used to delete a value, corresponding to the given key inside the pop(), from the existing dictionary and assign the remaining values to the existing dictionary.
14. Whenever you will print the dictionary elements it will be printed in dictionary order i.e alphabetical order. The numbers will be printed last.



```
>>> dict1={'name':'sachin','ID':123}
>>> dict1
{'name': 'sachin', 'ID': 123}
>>> dict1.items()
[('name', 'sachin'), ('ID', 123)]
>>> dict1.keys()
['name', 'ID']
>>> dict1.values()
['sachin', 123]
>>> dict1.has_key('ID')
True
>>> dict1.has_key(1)
False
>>> dict1.pop('name')
'sachin'
>>> dict1
{'ID': 123}
>>> dict1.append("name":'sachin')
  File "<stdin>", line 1
    dict1.append("name":'sachin')
                       ^
SyntaxError: invalid syntax
>>> dict1.append({"name":'sachin'})
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'dict' object has no attribute 'append'
>>> dict1.clear()
>>> dict1
{}
```

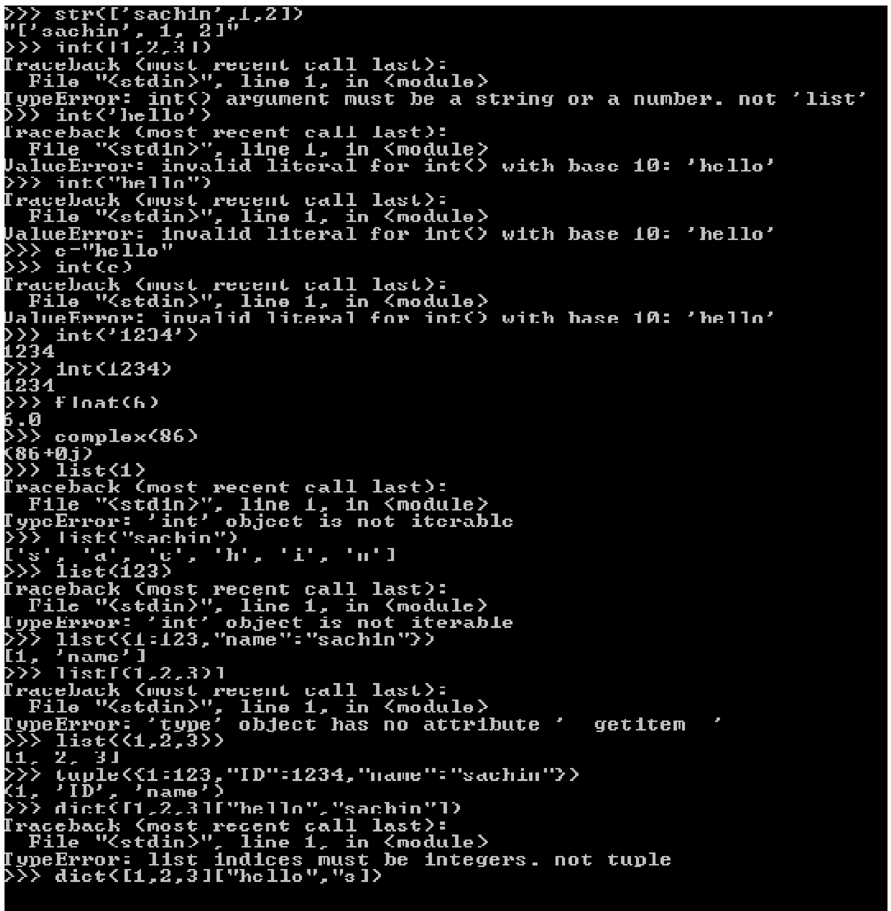
15. ‘In’ is used to check the members of the list. It is also used to traverse the entire dictionary.



```
>>> dict2={"name":"sachin","id":123,"sub":['p','c']}
>>> dict2
{'sub': ['p', 'c'], 'name': 'sachin', 'id': 123}
>>> for i in dict2:
...     print dict2[i]
...
['p', 'c']
sachin
123
>>> sachin in dict2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sachin' is not defined
>>> 'sachin' in dict2
False
>>> 'name' in dict2
True
```

<u>Type conversions:</u>
One type of data type converted to another data type is also known as coercion. Operation like addition, subtraction, multiplication, division coerce the integer data type implicitly to float if one of the operand is float. To explicitly convert we should use some of the built in methods or functions as follows.

1. Str(): is used to convert other data type to string. It must contain compatible value.
2. Int(): is used to convert other data type to int
3. Float(): is used to convert other data type to float
4. Long(): is used to convert other data type to long.
5. List(): is used to convert other data type to list.
6. Set():is used to convert other data type to set.
7. tuple():is used to convert other data type to tuple.



```
>>> str(['sachin',1,2])
"['sachin', 1, 2]"
>>> int([1,2,3])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: int() argument must be a string or a number, not 'list'
>>> int('hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'hello'
>>> int("hello")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'hello'
>>> c="hello"
>>> int(c)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'hello'
>>> int('1234')
1234
>>> int(1234)
1234
>>> float(6)
6.0
>>> complex(86)
(86+0j)
>>> list(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
>>> list("sachin")
['s', 'a', 'c', 'h', 'i', 'n']
>>> list(123)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
>>> list({1:123,"name":"sachin"})
[1, 'name']
>>> list[(1,2,3)]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'type' object has no attribute '__getitem__'
>>> list((1,2,3))
[1, 2, 3]
>>> tuple({1:123,"ID":1234,"name":"sachin"})
(1, 'ID', 'name')
>>> dict([1,2,3]["hello","sachin"])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers, not tuple
>>> dict([1,2,3]["hello","s])
```