UNIT II

Apache Hadoop, Hadoop Map Reduce Job Execution, Hadoop Schedulers, Hadoop Clusterset up

Introduction to Hadoop Framework

 Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models.

 Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

 Hadoop runs applications using the MapReduce algorithm, where the data is processed in parallel on different CPU nodes.

Hadoop=MapReduce + HDFS

(MapReduce Processing;HDFS Storage

Users of Hadoop:

 Hadoop is running search on some of the Internet's largest sites:

o Amazon Web Services: Elastic MapReduce

o AOL: Variety of uses, e.g., behavioral analysis & targeting

o Ebay: Search optimization (532-node cluster)

o Facebook: Reporting/analytics, machine learning (1100 m.)

o LinkedIn: People You May Know (2x50 machines)

o Twitter: Store + process tweets, log files, other data Yahoo: >36,000 nodes; biggest cluster is 4,000 nodes

Hadoop Architecture

 Hadoop has a Master Slave Architecture for both Storage & Processing

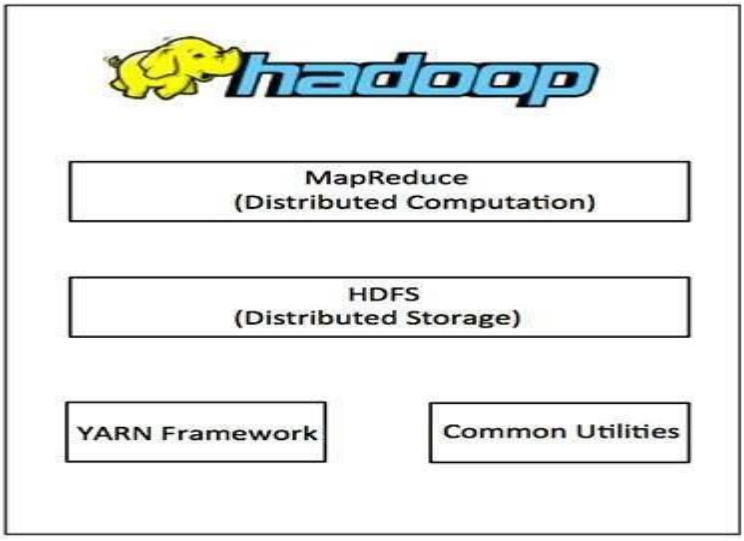
 Hadoop framework includes following four modules:

 Hadoop Common: These are Java libraries and provide file system and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.

Hadoop YARN: This is a framework for job scheduling and cluster resource management.

 Hadoop Distributed File System (HDFS): A distributed file system that provides high- throughput access to application data.

 HadoopMapReduce: This is system for parallel processing of large data sets.



 The Hadoop core is divided into two fundamental layers:

 MapReduce engine

 HDFS

 The MapReduce engine is the computation engine running on top of HDFS as its data storage manager.

 HDFS: HDFS is a distributed file system inspired by GFS that organizes files and stores their data on a distributed computing system.

 HDFS Architecture: HDFS has a master/slave architecture containing a single Name Node as the master and a number of Data Nodes as workers (slaves).

HDFS

 To store a file in this architecture,

 HDFS splits the file into fixed-size blocks (e.g., 64 MB) and stores them on workers (Data Nodes).

 The mapping of blocks to Data Nodes is determined by the Name Node.

 The NameNode (master) also manages the file system's metadata and namespace.

 Namespace is the area maintaining the metadata, and metadata refers to all the information stored by a file system that is needed for overall management of all files.

 NameNode in the metadata stores all information regarding the location of input splits/blocks in all DataNodes.

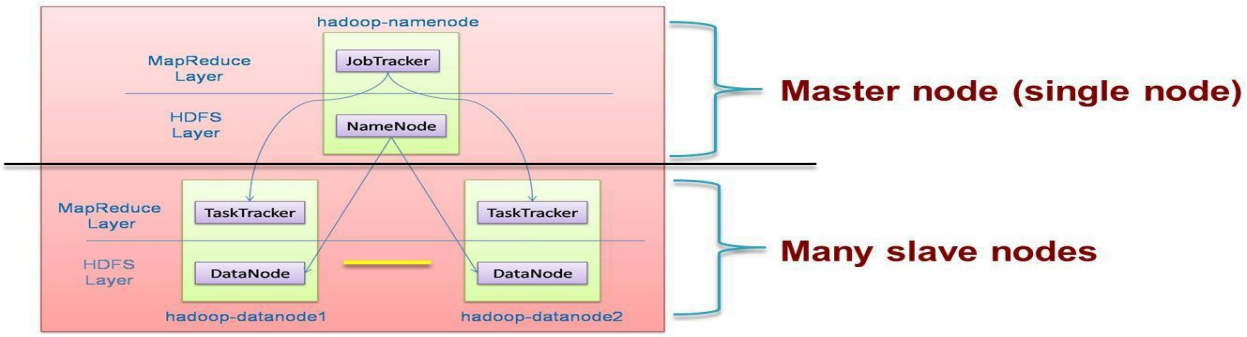
 Each DataNode, usually one per node in a cluster, manages the storage attached to the node.

 Each DataNode is responsible for storing and retrieving its file blocks

Architecture of Mapreduce in Hadoop

• Distributed file system (HDFS)

• Execution engine (MapReduce)



Properties of Hadoop Engine

 HDFS has a master/slave architecture containing

 A single NameNode as the master and

 A number of DataNodes as workers (slaves).

 To store a file in this architecture, HDFS splits the file into fixed-size blocks (e.g., 64 MB) and stores them on workers (DataNodes).

 The NameNode (master) also manages the file system's metadata and namespace.

 Job Tracker is the master node (runs with the namenode)

o Receives the user's job

o Decides on how many tasks will run (number of mappers)

o Decides on where to run each mapper (concept of locality)

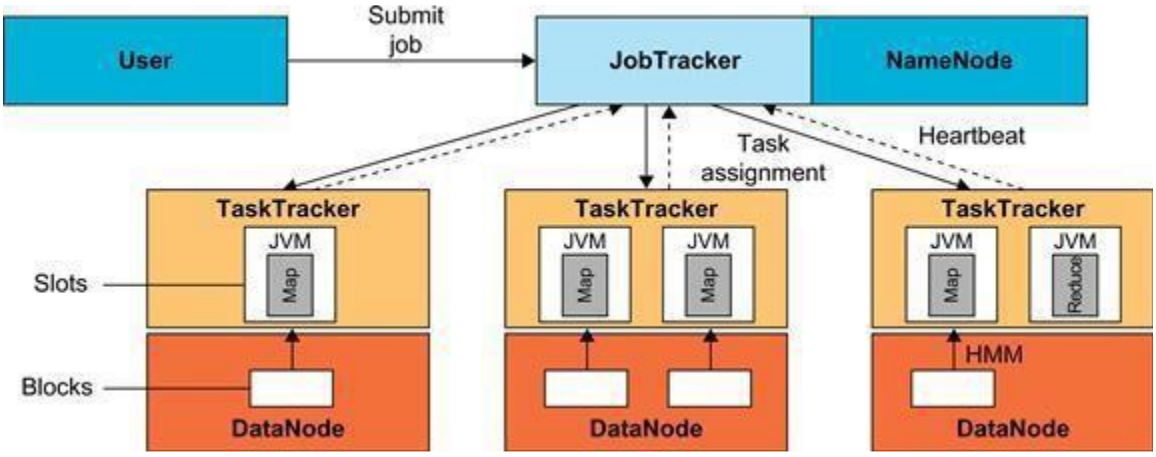
 Task Tracker is the slave node (runs on each datanode)

o Receives the task from Job Tracker

o Runs the task until completion (either map or reduce task)

o Always in communication with the Job Tracker reporting progress (heartbeats)

Running a Job in Hadoop



Three components contribute in running a job in this system:

o a user node,

o a JobTracker, and

o severalTaskTrackers.

 The data flow starts by calling the runJob(conf) function inside a user program running on the user node, in which conf is an object containing some tuning parameters for the MapReduce

 Job Submission: Each job is submitted from a user node to the JobTracker node.

 Task assignment : The JobTracker creates one map task for each computed input split

 Task execution : The control flow to execute a task (either map or reduce) starts inside the TaskTracker by copying the job JAR file to its file system.

 Task running check : A task running check is performed by receiving periodic heartbeat messages to the JobTracker from the TaskTrackers.

 Heartbeat:notifies the JobTracker that the sending TaskTracker is alive, and whether the sending TaskTracker is ready to run a new task.

MAP REDUCE

 MapReduce is a programming model for data processing.

 MapReduce is designed to efficiently process large volumes of data by connecting many commodity computers together to work in parallel

 Hadoop can run MapReduce programs written in various languages like Java, Ruby, and Python

 MapReduce works by breaking the processing into two phases:

o The map phase and

o The reduce phase.

 Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer.

 The programmer also specifies two functions:

o The map function and

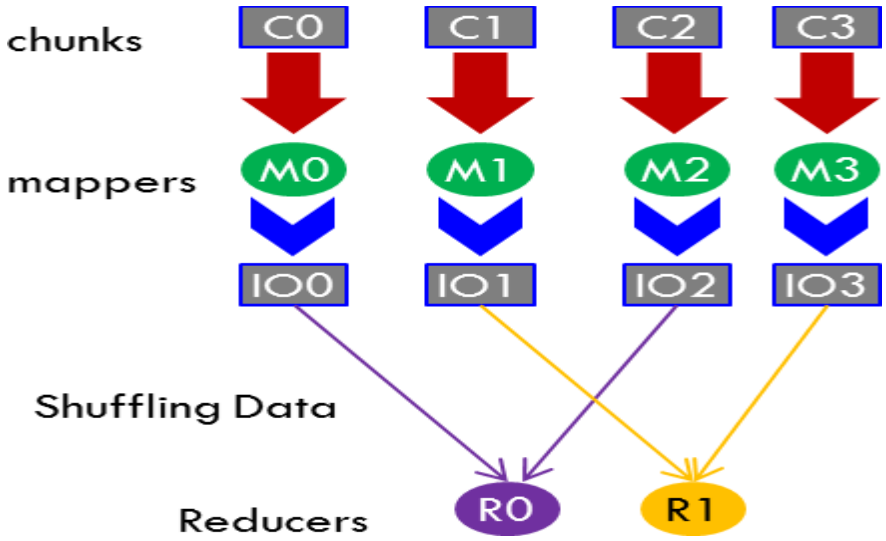
o The reduce function.

 In MapReduce, chunks are processed in isolation by tasks called Mappers

 The outputs from the mappers are denoted as intermediate outputs (IOs) and are brought into a second set of tasks called Reducers

 The process of bringing together IOs into a set of Reducers is known as shuffling process

 The Reducers produce the final outputs (FOs)



 Overall, MapReduce breaks the data flow into two phases, map phase and reduce phase Mapreduce Workflow

Application writer specifies

 A pair of functions called Mapper and Reducer and a set of input files and submits the job

 Input phase generates a number of FileSplits from input files (one per Map task)

 The Map phase executes a user function to transform input key-pairs into a new set of key- pairs

 The framework Sorts & Shuffles the key-pairs to output nodes

 The Reduce phase combines all key-pairs with the same key into new keypairs

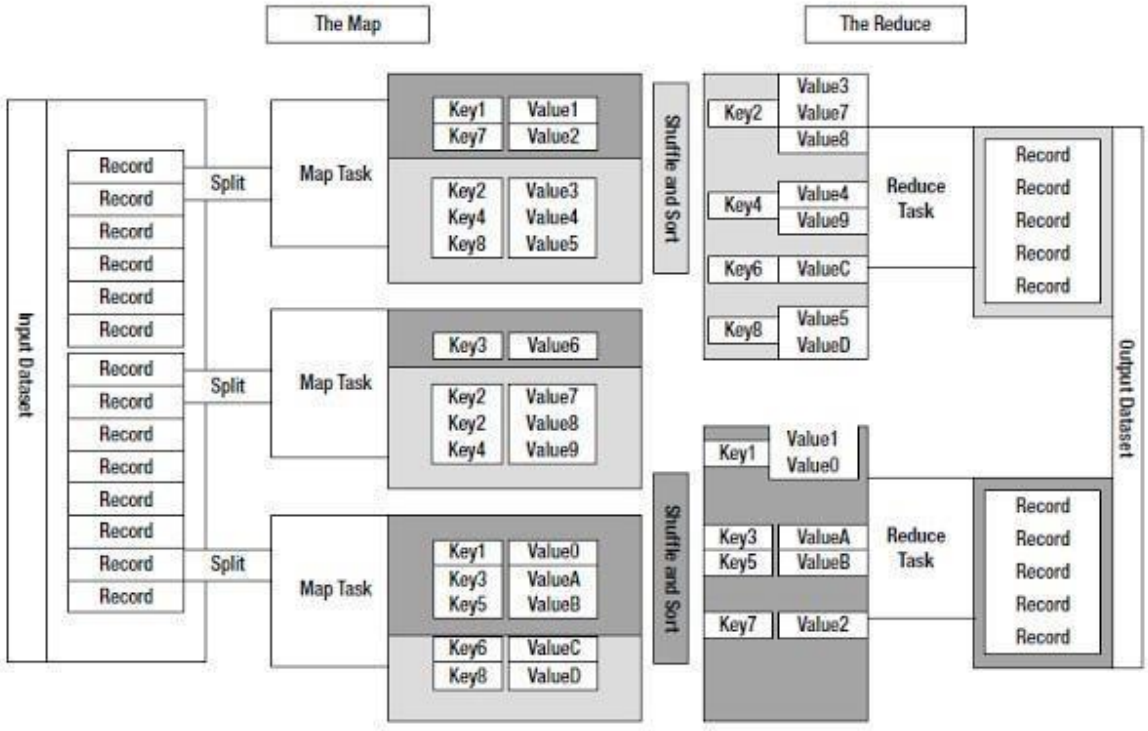
 The output phase writes the resulting pairs to files as “parts”

Characteristics of MapReduce is characterized by:

 Its simplified programming model which allows the user to quickly write and test distributed systems

 Its efficient and automatic distribution of data and workload across machines

 Its flat scalability curve. Specifically, after a Mapreduce program is written and functioning on 10 nodes, very little-if any- work is required for making that same program run on 1000 nodes



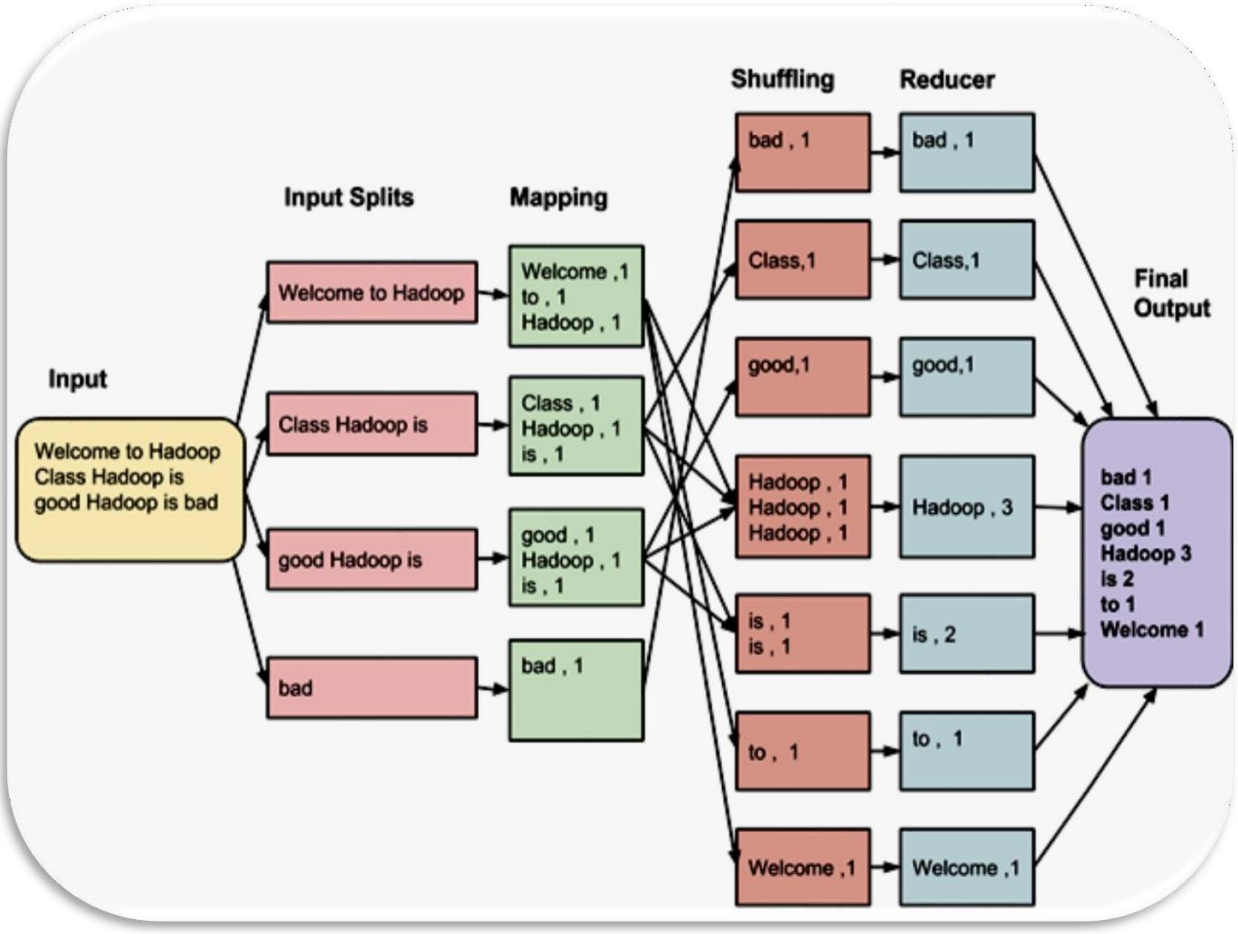
.

Map Reduce Example:

Input File:

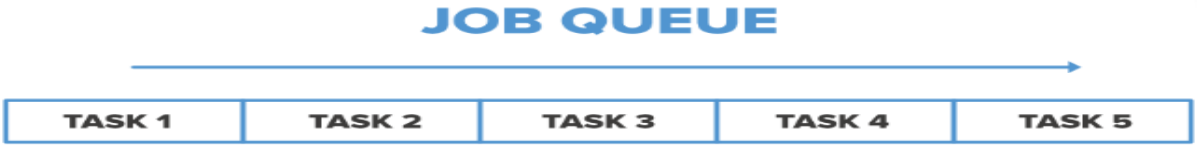
Welcome to Hadoop Class Hadoop is good

Hadoop is bad



Hadoop - Schedulers and Types of Schedulers

- FIFO (First In First Out) Scheduler.
- Capacity Scheduler.
- Fair Scheduler.
- These Schedulers are actually a kind of algorithm that we use to schedule tasks in a Hadoop cluster when we receive requests from different-different clients.



FIFO SCHEDULERS

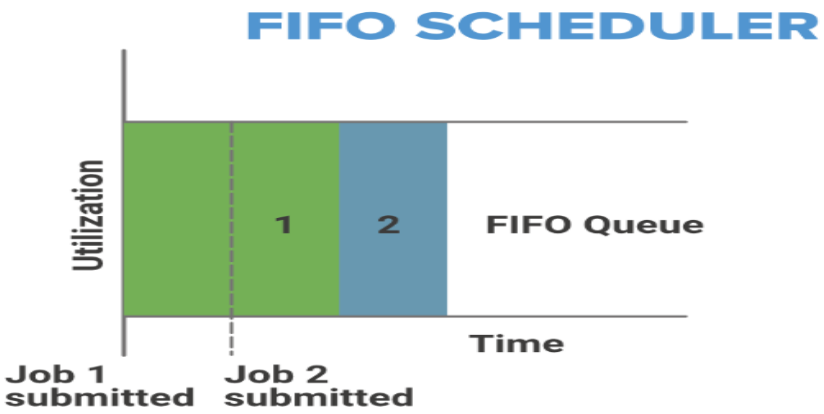
- FIFO i.e. First In First Out, so the tasks or application that comes first will be served first.
- This is the default Scheduler we use in Hadoop.
- The tasks are placed in a queue and the tasks are performed in their submission order.
- In this method, once the job is scheduled, no intervention is allowed. So sometimes the high-priority process has to wait for a long time since the priority of the task does not matter in this method.

Advantage:

- No need for configuration
- First Come First Serve
- simple to execute

Disadvantage:

- Priority of task doesn't matter, so high priority jobs need to wait
- Not suitable for shared cluster



CAPACITY SCHEDULER

- The Capacity Scheduler allows multiple occupants to share a large size Hadoop cluster.

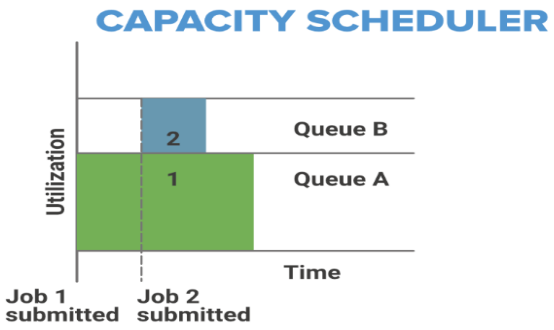
- In Capacity Scheduler corresponding for each job queue, we provide some slots or cluster resources for performing job operation.
- Each job queue has it's own slots to perform its task.

Advantage:

- Best for working with Multiple clients or priority jobs in a Hadoop cluster
- Maximizes throughput in the Hadoop cluster

Disadvantage:

- More complex
- Not easy to configure for everyone



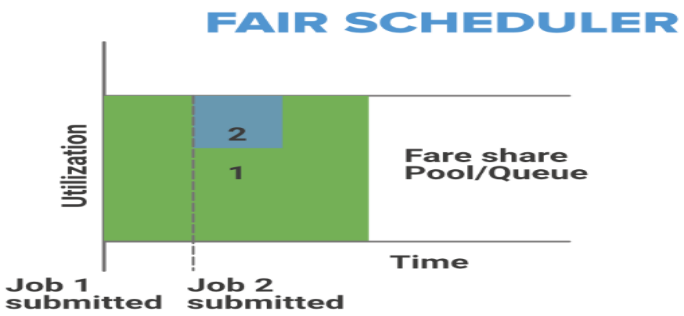
FAIR SCHEDULER

- It allocates resources evenly between multiple jobs and also provides capacity guarantees.
- It assigns resources to jobs such that each job gets an equal share of the available resources.

Advantages:

- Resources assigned to each application depend upon its priority.
- it can limit the concurrent running task in a particular pool or queue.

Disadvantages: The configuration is required.



HADOOP CLUSTER SET UP

- A Hadoop cluster is a type of computer cluster made up of commodity hardware inexpensive and widely available devices. These nodes work together to store and process large volumes of data in a distributed environment.In a Hadoop cluster:
- **Master Nodes:** Include the NameNode (handles file system metadata) and the ResourceManager (manages cluster resources).
- **Slave Nodes:** Include DataNodes (store actual data) and NodeManagers (manage execution of tasks).

TYPES OF DATA PROCESSED BY HADOOP

- Hadoop clusters can handle various types of data:
- **Structured Data:** Organized in a fixed schema, e.g., MySQL databases.
- **Semi-Structured Data:** Partially organized data, e.g., XML, JSON.
- **Unstructured Data:** No fixed format, e.g., images, videos, audio files.

TYPES OF HADOOP CLUSTERS

1. Single Node Hadoop Cluster

In a single node cluster, all Hadoop components (NameNode, DataNode, ResourceManager, NodeManager, etc.) run on the same machine. This setup uses a single JVM (Java Virtual Machine) and is mainly used for learning or testing purposes.

2. Multi Node Hadoop Cluster

A multi node cluster has multiple machines (nodes). Master daemons like NameNode and ResourceManager run on powerful systems, while slave daemons like DataNode and NodeManager run on other, often less powerful, machines. This setup is used in real-world, large-scale data processing.

HADOOP CLUSTERS

- Setting up a Hadoop cluster involves configuring multiple machines to work together as a unified system for processing and storing large datasets. Here's a basic outline of the process:
- **Hardware Setup**:

  Choose suitable hardware for the cluster, including servers with sufficient RAM, CPU, and storage capacity.

  Ensure that all machines have a reliable network connection.
- **Operating System Installation**:

  Install a compatible operating system (e.g., Linux) on each machine in the cluster.
- **Java Installation**:

  Install Java Development Kit (JDK) on all machines, as Hadoop is built using Java.
- **Hadoop Installation**:

  Download the Hadoop distribution and extract it on all machines.

  Configure the Hadoop environment variables, such as HADOOP_HOME and JAVA_HOME.
- **Configuration Files**:

  Modify the core-site.xml, hdfs-site.xml, and yarn-site.xml configuration files to specify the cluster settings, such as the Name node and Data node details.

- **SSH Setup**:

  Set up password less SSH between the master and slave nodes to enable secure communication.

- **Hadoop Daemons**:

  Start the Hadoop daemons, including the Namenode, Datanode, Resource Manager, and Node Manager, on their respective machines.

- **Testing**:

  Verify the cluster setup by running sample MapReduce jobs and checking the Hadoop web interface for cluster status.

- **Maintenance**:

  Regularly monitor the cluster for performance, resource utilization, and any potential issues.

UNIT II

Cloud Application Design

1)Introduction: -

Web applications have evolved significantly in the past decade and are now quite dynamic in nature allowing users to interact and collaborate, include user generated content such as comments and discussions, integrate social networks and multimodal content such as text, images, audio, video, presentations, etc., in various formats.

Due to this dynamic nature of modern web applications, the traffic patterns for such applications are becoming more and more unpredictable.

Some applications experience seasonal variations in their workloads, e.g., e-Commerce websites see elevated user traffic in holiday seasons.

Traditional approaches that were based on the 'one size fits all' paradigm no longer work for modern applications.

Here will learn about the design considerations for cloud applications, the reference architectures for various types of applications and design methodologies such as SOA, CCM and MVC.

2) Design Considerations for Cloud Applications

There are important design considerations for developing applications that can leverage the benefits of cloud computing. Some of them are shown below:

a)Scalability
b)Reliability & Availability
c)Security
d)Maintenance & Upgradation
e)Performance

a)Scalability :-

Scalability is an important factor that drives the application designers to move to cloud computing environments. Building applications that can serve millions of users without taking a hit on their performance has always been challenging.

With the growth of cloud computing application designers can provision adequate resources to meet their workload levels. However, simply provisioning more and more resources may not bring performance gains if the applications are not designed to scale well.

Traditional approaches were based on either **over- provisioning of resources** to handle the peak workload levels expected or **provisioning based on average workload levels**.

Both approaches have their disadvantages.

-While the over- provisioning approach leads to underutilization of resources and increased costs,
-the approach based on average workload levels can lead to traffic overloads, slow response times, low throughputs and hence loss of opportunity to serve the customers.

In order to leverage the benefits of cloud computing such as dynamic scaling, the following design considerations must be kept in mind:

- **Loose coupling of components:** By designing loosely coupled components, it is possible to scale each component independently.
- **Asynchronous communication:** By allowing asynchronous communication between components, it is possible to add capacity by adding additional servers when the application load increases.
- **Stateless design:** Stateless designs that store state outside of the components in a separate database

Availability is the probability that a system will perform a specified function under given conditions at a prescribed time.

The important considerations to be kept in mind while developing highly reliable and available applications are:

- **No single point of failure:** To high achieve reliability and availability, having a redundant resource or an automated fallback resource is important.
- **Trigger automated actions on failures:** By using failures and triggers for automated actions it is possible to improve the application reliability and availability.
  For example, if an application server experiences high CPU usage and is a unable to serve new requests, a new application server is automatically launched.
- **Graceful degradation:** Graceful degradation means that if some component of the application becomes unavailable the application as a whole would still be available and continue to serve the users, though, with limited functionality.
  For example, in an e-Commerce application, if a component that manages a certain category of products becomes unavailable, the users should still be able to view products from other categories.
- **Logging:** Logging all events in all the application components can help in detecting bottlenecks and failures so that necessary design/deployment changes can be made to improve application reliability and availability.
- **Replication:** All application data should be replicated. Replication is used to create and maintain multiple copies of the data in the cloud. In the event of data loss at the primary location, organizations can continue to operate their applications from secondary data sources.

c)Security:-

Security is an important design consideration for cloud applications given the out- sourced nature of cloud computing environments.

In domains such as healthcare there are several government laws that require the applications to ensure security of health information of patients.

Key security considerations for cloud computing environments include:

-Securing data at rest
-Securing data in motion
-Authentication
-Authorization
-Identity and access management
-Key management
-Data integrity
-Auditing

d)Maintenance & Upgradation: -

To achieve a rapid time-to-market, businesses typically launch their applications with a core set of features ready and then incrementally add new features as and when they are complete.

Businesses may need to adapt their applications based on the feedback from the users. In such scenarios, it is important to design applications with low maintenance and upgradation costs.

Design decisions such as loosely coupled components help in reducing the application maintenance and upgradation time.

In applications with loosely coupled components, changes can be made to a component without affecting other components. Moreover, components can be tested individually.

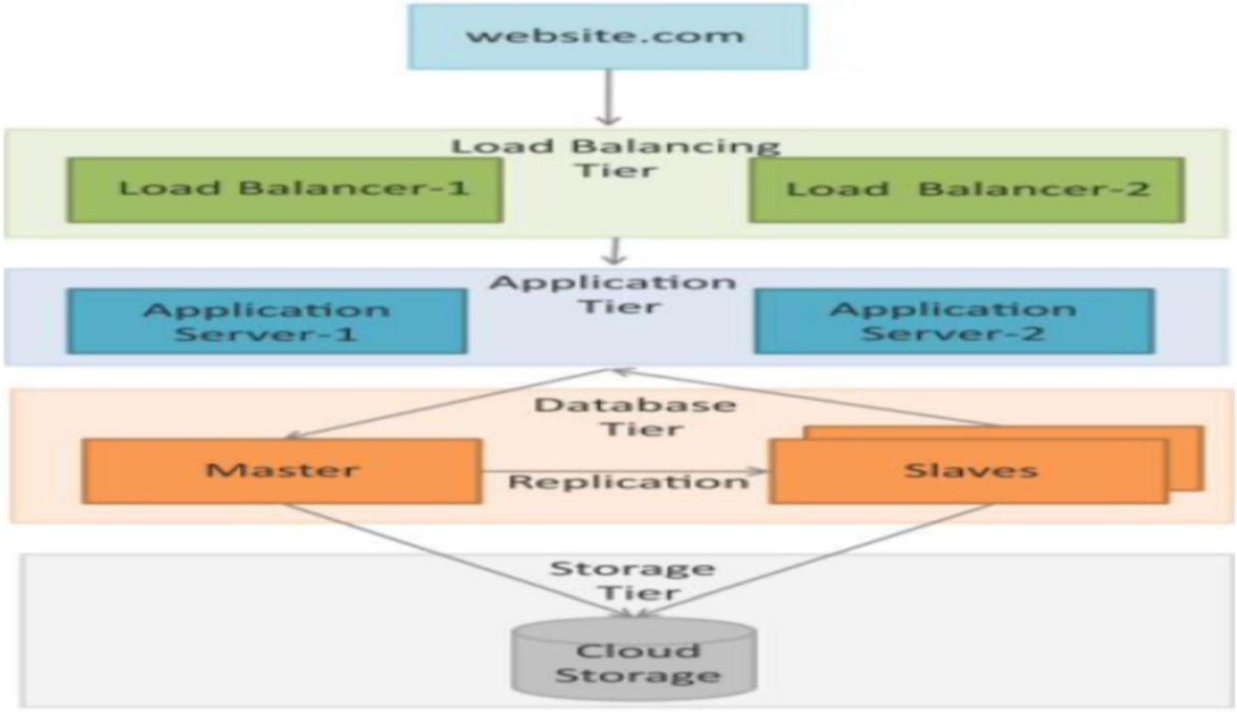
Other decisions such as logging and triggering automated actions also help in lowering the maintenance costs.

Reference Architectures for Cloud Applications

Introduction:-

Multi-tier cloud applications can have various deployment architecture alternatives. Choosing the right deployment architecture is important to ensure that the application meets the specified performance requirements.

Below diagram shows a typical deployment architecture for e-Commerce, Business-to-Business, Banking and Financial applications.



The various tiers in this deployment include:

-Load Balancing Tier, Application Tier and Database Tier

Load Balancing Tier:-

The first tier is the load balancing tier. Load balancing tier consists of one or more load balancers. It is recommended to have at least two load balancer instances to avoid the single point of failure.

Whenever possible, it is also recommended to provision the load balancer instances in separate availability zones of the cloud service provider to improve reliability and availability.

Application Tier:-

It consists of one or more application servers. For this tier, it is recommended to configure auto scaling. Auto scaling can be triggered when the recorded values for any of the specified metrics such as CPU usage, memory usage, etc. goes above defined thresholds.

The minimum and maximum size of the application server auto scaling groups can be configured. It is recommended to have at least two application servers running at all times to avoid a single point of failure.

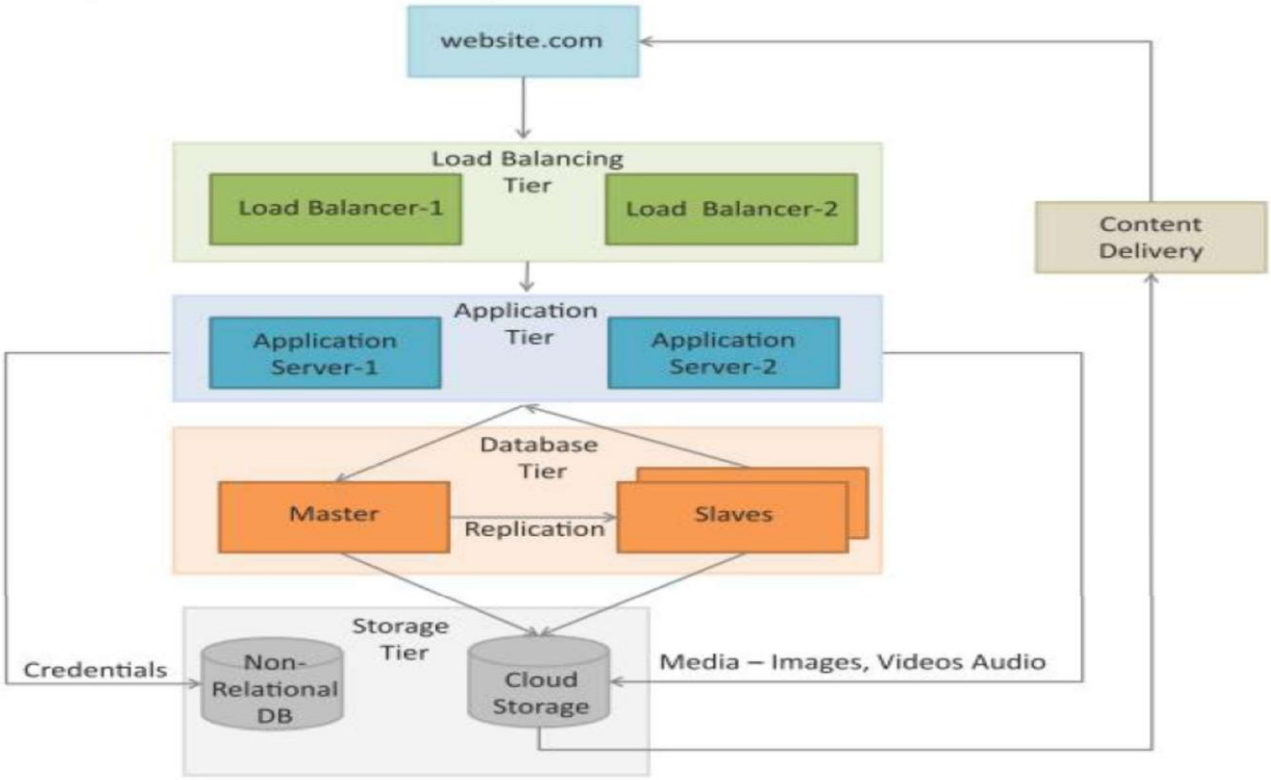
When the workload increases rapidly, the existing application server instances may fail to serve all requests. Therefore, it is recommended to set the threshold values for the auto scaling

Multiple slave nodes also serve as a backup for the master node. In the event of failure of the master node, one of the slave modes can be automatically configured to become the master.

Regular snapshots of the database are recommended. The frequency of snapshots may be configured to be daily or hourly. It is recommended to store snapshots in distributed persistent cloud storage solutions (such as Amazon S3).

Architecture for content delivery applications

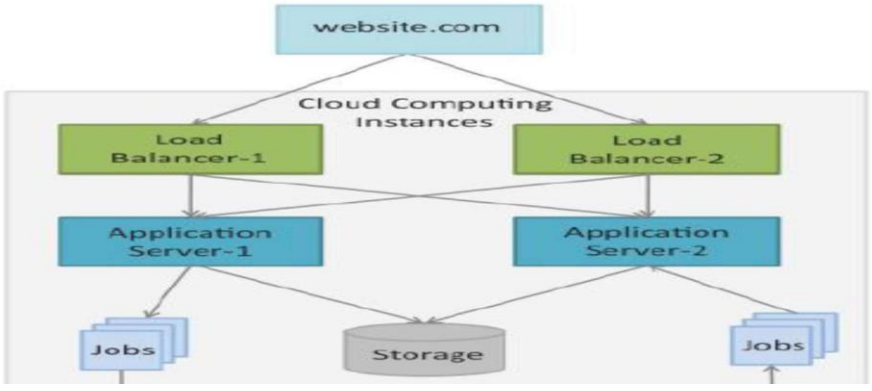
Below diagram shows typical deployment architecture for content delivery applications such as online photo albums, video webcasting, etc.



Both relational and non-relational data stores are shown in this deployment. A content delivery network (CDN) which consists of a global network of edge locations is used for media delivery. CDN is used to speed up the delivery of static content such as images and videos.

Architecture for compute intensive applications:-

Below diagram shows typical deployment architecture for compute intensive applications such as Data Analytics, Media Transcoding, etc.



The above diagram shows web, application, storage, computing/analytics and database tiers. The analytics tier consists of cloud- based distributed batch processing frameworks such as Hadoop which are suitable for analyzing big data.

Data analysis jobs (such as MapReduce) are submitted to the analytics tier from the application servers. The jobs are queued for execution and upon completion the analyzed data is presented from the application servers.

Cloud Application Design Methdologies

Introduction:-

Some of the design methodologies used for cloud applications include the following:

-Software Oriented Architecture
-Cloud Component Model
-IaaS, PaaS and SaaS Services for Cloud Applications
-Model View Controller
-RESTful Web Services

1)Software Oriented Architecture (SOA):-

Service Oriented Architecture is a Solution for making two Software Components communicates each other efficiently.

Suppose we have software called **service consumer** that wants to talk with another software called **service provider**. The Service Consumer going to send a service request to provider, and the Provider going to reply with service response.

The primary characteristic of SOA is that Service Provider Software publishes its service Description, which is placed in certain directory as shown below:

The Consumer software can make a query against the service directory to find what services are available and how to communicate with service provider

WSDL- Service Description is written in special Language called Web Service Description language (WSDL). This is a special Industry accepted language for placing web service description.

SOAP- It is a Protocol to talk to talk to the Directory, the service provider will communicate with Directory using SOAP Protocol to send its service description.

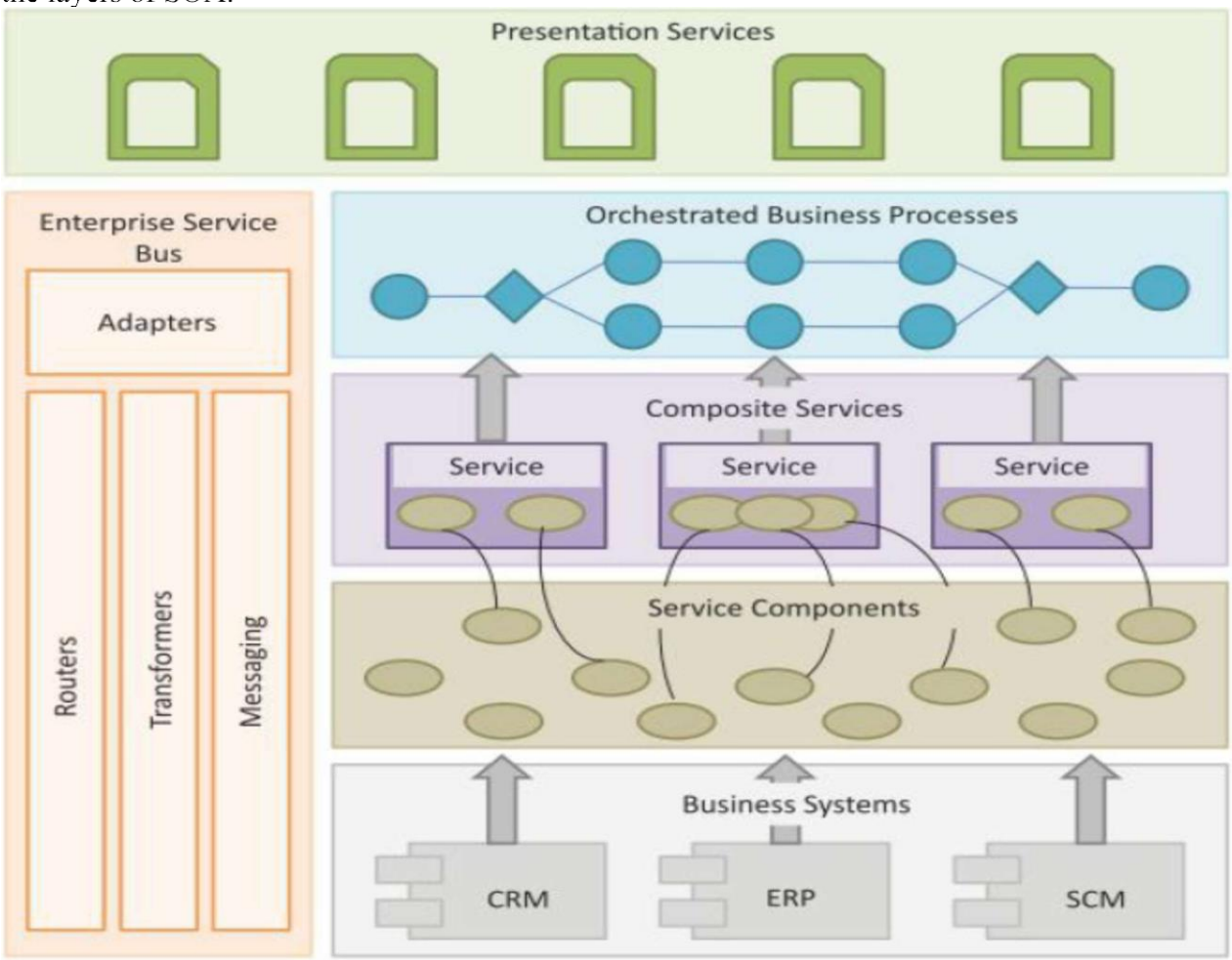
XML- The Service Consumer software formulate its message to be sent to the Provider Software based on XML (Extensible Markup Language). Service Provider will generate its response back to consumer software which is written in XML.

A WSDL 2.0 description contains:

- **Service:** Service describes a discrete system function that is exposed as a web service.
- **Endpoint:** Endpoint is the address of the web service.
- **Binding:** Binding specifies the interface and transport protocol.
- **Interface:** Interface defines a web service and the operations that can be performed by the service and the input and outputs.
- **Operation:** Operation defines how the message is decoded and the actions that can be performed.
- **Types:** Types describe the data

SOA services communicate using the Simple Object-Oriented Protocol (SOAP). SOAP is a **protocol** that allows exchange of structured information between web services. WSDL in combination with SOAP is used to provide web services over the internet.

SOA allows reuse of **services** for multiple applications. Since each service is designed to perform a small function (such as display the balance in a bank account, show a list of recent transactions, etc.) developers can orchestrate existing SOA services in an ad-hoc fashion to create new application without the need for re-implementing the services. The following diagram shows the layers of SOA.



The layers of SOA include:

- **Business Systems:** This layer consists of custom-built applications and legacy systems such as Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), Supply Chain Management (SCM), etc.
- **Service Components:** The service components allow the layers above to interact with the business systems. The service components are responsible for realizing the functionality of the services exposed.
- **Composite Services:** These are coarse-grained services which are composed of two or

IBM SOA Foundation is based on an SOA reference architecture that define the comprehensive IT Services required to support SOA at the model, assemble, deploy, manage and governance stages of the SOA life Cycle.
Microsofts's windows Communication Foundation is a runtime and a set of API's for building connected, service-oriented applications.
Oracle SOA Suite is a comprehensive, pluggable software suite to build, deploy and manage service-oriented architectures. Software- based SOA requires deploying and maintaining software and infrastructure.
Salesforce SOA delivers SOA as a service, run on Salesforce.com's on-demand platform, and allows developers to reuse software components.
Limitation of traditional SOA is that it uses a messaging layer' above HTTP by using SOAP which imposes prohibitive constraints for web application developers and requires greater implementation effort.

2)Cloud Component Model:-

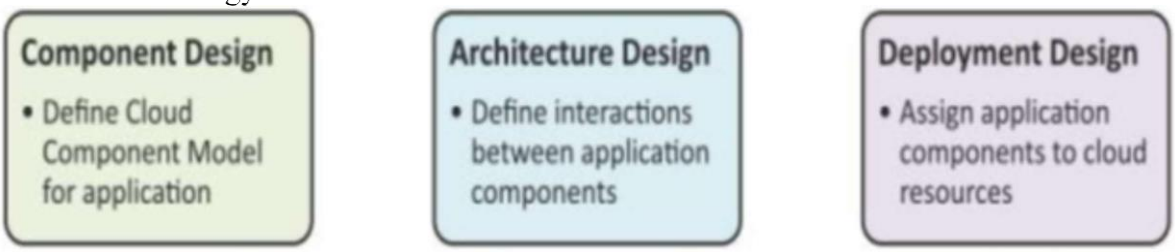
Cloud Component Model is an application design methodology that provides a flexible way of creating cloud applications in a rapid, convenient and platform independent manner.

CCM is an architectural approach for cloud applications that is not tied to any specific programming language or cloud platform.

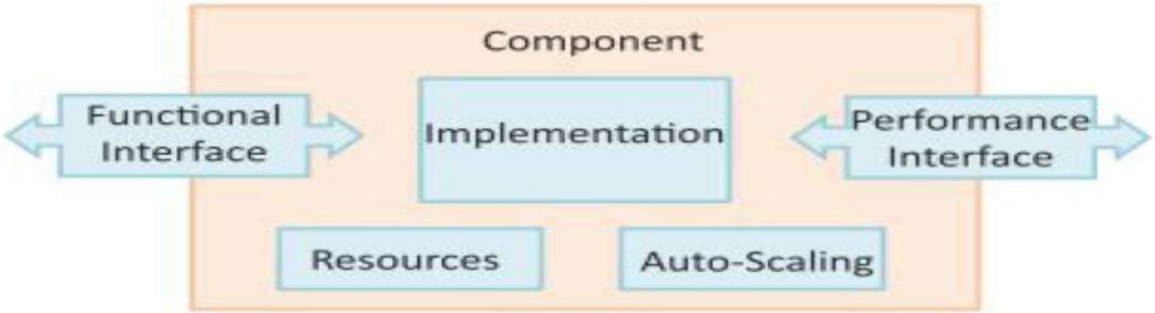
Cloud applications designed with CCM approach can have innovative hybrid deployments in which different components of an application can be deployed on cloud infrastructure and platforms of different cloud vendors.

Applications designed using CCM have better portability and interoperability. CCM based applications have better scalability by decoupling application components and providing asynchronous communication mechanisms.

Below diagram shows the steps involved in application design using Cloud Component Model Methodology:



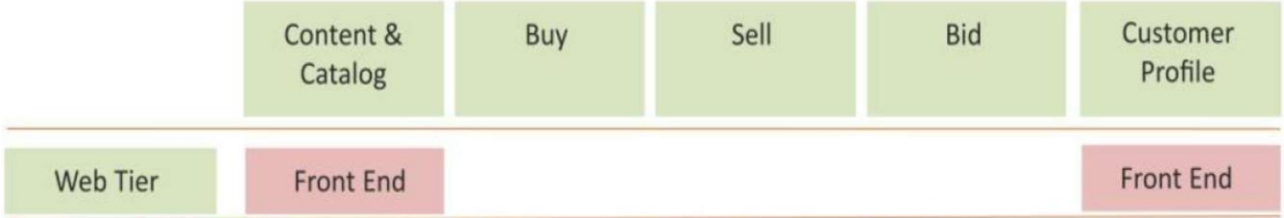
Architecture of CMM Component is shown below:



(b)

Example:

The following diagram shows CMM Map for an E-Commerce Application.



3)IaaS, PaaS and SaaS Services for Cloud Applications:-

Cloud Service Providers such as Amazon, Google, Microsoft, etc. provide PaaS, IaaS, and SaaS services that the developers can use for developing and deploying applications in cloud computing environments.

Cloud service providers make cloud resources available to the users with cloud APIs via a web- based interface. There are numerous cloud service providers with their own cloud APIs.

For example, **Force.com PaaS** allows developers to create applications using Apex (a proprietary programming language) and Visualforce (an XML-like syntax for building user interfaces).

Similarly, **applications designed for Google App Engine** (GAE) platform must implement GAE specific interfaces and have a GAE specific deployment descriptor.

Amazon CloudFormation allows developers to create, provision and manage cloud deployments using Amazon specific Resources such as Amazon EC2, Amazon EBS, etc.

Red Hat's open source PaaS, OpenShift provides built-in support for a variety of programming languages (Ruby, Python, PHP, Java, etc.) and frameworks (Ruby on Rails, Django, etc.).

CloudFoundry is another open PaaS offering that supports a variety of clouds (Amazon Web Services, Rackspace, etc.), frameworks (Spring for Java, Scala, etc.) and application services (MySQL, MongoDB, etc.).

Open PaaS offerings provide greater flexibility than vendor-specific PaaS offerings (such as GAE, Force.con, etc.) as they support more programming languages and frameworks and provide a greater choice of clouds for deployment, thus ending the vendor lock-in.

4)Model View Controller:-

Model View Controller (MVC) is a popular software design pattern tor web applications. The MVC pattern consists of three parts:

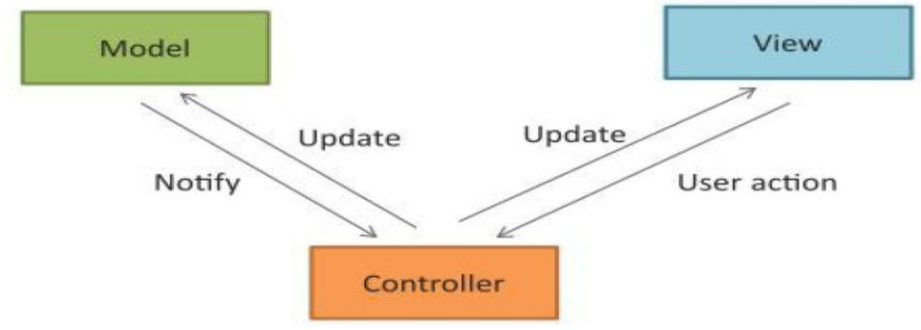
- **Model:** Model manages the data and the behaviour of the applications. Model processes events sent by the controller. Model responds to the requests for information about its state (from the view) and responds to the instructions to change state (from controller).

- **View:** View prepares the interface which is shown to the user. Users interact with the application through views.

- **Controller:** Controller glues the model to the view. Controller processes user requests and updates the model when the user manipulates the view.

MVC separates the application logic, the data, and the user interface. The benefit of using

The Model View Controller Architecture is shown below:



5)RESTful Web Services:-

Representational State Transfer (REST) is a set of Architectural principles by which you can design web services and web APIs that focus on a system's resources and how resource states are addressed and transferred.

The REST architectural constraints apply to the components, connectors, and data elements, within a distributed hypermedia system. The REST architectural constraints are as follows:

- **Client-Server:** The principle behind the client-server constraint is the separation of concerns. For example, clients should not be concerned with the storage of data which is a concern of the server. Similarly, the server should not be concerned about the user interface, which is a concern of the client. Separation allows client and server to be independently developed and updated.
- **Stateless:** Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. The session state is kept entirely on the client.
- **Cacheable:** Cache constraint requires that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. **If a response is cacheable**, then a Client cache is given the right to reuse that response data for later equivalent requests. Caching can partially or completely eliminate some interactions and improve efficiency and scalability.
- **Layered System:** Layered system constrains the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting. For example, a client cannot tell whether it is connected directly to the end server, or to an intermediary along the way. System scalability can be improved by allowing intermediaries to respond to requests instead of the end server, without the client having to do anything different.
- **Uniform Interface:** Uniform Interface constraint requires that the method of communication between a client and a server must be uniform. Resources are identified in the requests (by URLs in web based systems) and are themselves separate from the representations of the resources that are returned to the client. When a client holds a representation of a resource it has all the information required to update or delete the resource (provided the client has required permissions). Each message includes enough information to describe how to process the message.
- **Code on demand:** Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

A RESTful web service is a web API implemented using HTTP and REST principles. RESTful web service is a collection of resources which are represented by URLs.

RESTful web API has a base URL (e.g. http://example.com/api/tasks/). The clients send requests to these URLs using the methods defined by the HTTP protocol (e.g., GET, PUT, POST, or DELETE).

A RESTful web service can support various Internet media types (JSON being the most popular media type for RESTful web services).

Data Storage Approaches

Introduction:-

There are two broad categories of approaches for databases.

-Relational (SQL) Approach

-Non-Relational (No-SQL) Approach

-

1)Relational (SQL) Approach:

A relational database is database that conforms to the relational model that was popularized by Edgar Codd in 1970.

A relational database has a collection of relations (or Tables). A relation is a set of tuples (or rows). Each relation has a fixed schema that defines the set of attributes (or columns in a table) and the constraints on the attributes.

Table 5.3 summarizes the 12 rules that Codd introduced for relational databases.

Rule	Description
Information rule	All information in a relational database is represented explicitly at the logical level and in exactly one way - by values in tables.
Guaranteed access rule	All data must be accessible. Every individual scalar value in a relational database is guaranteed to be logically accessible by specifying the table name, primary key value, and column name.
Systematic treatment of null values	DBMS must allow null values for all fields. Null values are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way.
Dynamic online catalog based on relational model	The system must support an online, relational catalog to authorized users using the same relational language.
Comprehensive sub-language rule	A relational system must support at least one language whose statements are expressible, by a well-defined syntax, as character strings, that is comprehensive in supporting all of the following items: Data Definition, View Definition, Data manipulation, Integrity Constraints, Authorization, Transaction boundaries.
View updating rule	All views that are theoretically updatable are also updatable by the system.
High level insert, update, delete	The system must support set-at-a-time insert, update, and delete operators. This means that the capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update, and deletion of data.
Physical data independence	Any changes made in either storage representations or access methods, should not require the application to be changed.
Logical data independence	Changes to the logical level (tables, columns, rows, and so on) must not require a change to an application based on the structure.
Integrity independence	Integrity constraints specific to a particular relational database must be definable in the relational data sub-language and storable in the catalog, not in the application programs.
Distribution independence	A relational DBMS has distribution dependence. That is, existing applications should continue to operate successfully, when a distributed version of the DBMS is first introduced and when existing distributed data are redistributed around the system.
Non-subversion rule	If a relational system has a low-level (single record at a time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher-level relational language (multiple records at a time).

A relational database has various constraints described as follows:

- **Domain Constraint:** Domain constraints restrict the domain of each attribute or the set of possible values of the attribute. Domain constraints specify that the value of each attribute must be a value from the domain of the attribute.
- **Entity Integrity Constraint:** Entity integrity constraint states that no primary key value can be null. Since primary key is used to uniquely identify each tuple in a relation, having null value for a primary key value will make it impossible to identify triples in the relation.
- **Referential Integrity Constraint:** Referential integrity constraints are required to maintain consistency among the tuples in two relations. Referential integrity requires every value of one attribute of a relation to exist as a value of another attribute in another relation.
- **foreign Key:** For cross-referencing between multiple relations foreign keys are used. Foreign key is a key in a relation that matches the primary key of another relation.
-

Relational databases support at least one comprehensive sub-language, the most popular being the Structured Query Language (SQL). Relational databases provide ACID guarantees that are a set of properties that guarantee that database transactions are processed reliably, described as follows:

- **Atomicity:** Atomicity property ensures that each transaction is either "all or nothing". In other words, an atomic transaction ensures that all parts or the transaction complete or the database state is left unchanged. Partially completed transactions in the event of system outages can lead to an invalid state. Atomicity ensures that the transaction is indivisible and is either committed or aborted.
- **Consistency:** Consistency property ensures that each transaction brings the database from one valid state to another.
- **Durability:** Durability property ensures that once a transaction is committed, the data remains as it is, i.e. it is not affected by system outages such as power loss. Durability guarantees that the database can keep track of changes and can recover from abnormal terminations.
- **Isolation:** Isolation property ensures that the database state obtained after a set of concurrent transactions is the same as would have been if the transactions were executed serially. This provides concurrency control, i.e. The transactions are isolated from each other until they finish.

Table 5.5 lists some popular relational databases.

Relational Database
Oracle database
Microsoft SQL Server
DB2
SAP Sybase Adaptive Server Enterprise
SAP Sybase IQ
MySQL
PostgreSQL
Teradata
Informix
Ingres
Aster Data
Netezza

Pros	Cons
Well defined consistent model. An application that runs on one relational database (such as MySQL) can be easily changed to run on other relational databases (eg. Microsoft SQL server). The underlying model remains unchanged.	Performance is the major constraint for relational databases. The performance depends on the number of relations and the size of the relations. Scaling out relational database deployments is difficult.
Provide ACID guarantees.	Limited support for complex data structures. Eg. If the data is naturally organized in a hierarchical manner and stored as such, the hierarchical approach can allow quick analysis of data.
Relational integrity maintained through entity and referential integrity constraints.	A complete knowledge of the database structure is required to create ad hoc queries.
Well suited for Online Transaction Processing (OLTP) applications.	Most relation database systems are expensive.
Sound theoretical foundation (based on relational model) which has been tried and tested for several years. Stable and standardized databases available.	Some relational databases have limits on the size of the fields.
The database design and normalization steps are well defined and the underlying structure is well understood.	Integrating data from multiple relational database systems can be cumbersome.

2)Non-Relational (No-SQL) Approach:-

Non-relational databases (or popularly called No-SQL databases) are becoming popular with the growth of cloud computing. Non-relational databases have better horizontal scaling capability and improved performance for big data at the cost of less rigorous consistency models. Unlike relational databases, No-relational databases do not provide ACID guarantees.

Most Non-relational databases offer "eventual" consistency, which means that given a sufficiently long period of time over which no updates are made, all updates can be expected to propagate eventually through the system and the replicas will be consistent.

BASE (Basically Available, Soft state, Eventual consistency) guarantees for non-relational datal bases as opposed to AC1D guarantees provided by relational databases.

The driving force behind the non-relational databases is the need for databases that can achieve high scalability, fault tolerance and availability. These databases can he distributed on a large cluster of machines. Fault tolerance is provided by storing multiple replicas of data on different machines.

For, example with a replication factor set equal to N for a non-relational database, each record has N replicas on different machines.

Non-relational databases are popular for Applications in which the scale of data involved is massive, the data way not be structured and real-time performance is important as opposed to consistency. These systems are optimized for fast retrieval and appending operations on records.

Unlike relational databases, the non-relational databases do not have a strict schema. The records can be in the form of key-value pairs or documents. Most non-relational databases are classified in terms of the data storage model or type of records that Can be stored. The commonly

keys or attributes) but there are no strict requirements for a schema. Documents are organized in different ways in different solutions such as collections, buckets, tags, etc.

- **Graph store:** Graph stores are designed for storing data that has graph structure (nodes and edges). These solutions are suitable for applications that involve graph data such as social networks, transportation systems, etc.
- **Object store:** Object store solutions are designed for storing data in the form of objects defined in an object-oriented programming language.

PYTHON BASICS

BASICS OF PYTHON

Python, an exceedingly resourceful language resembling everyday language, is astonishingly hassle-free to learn, even for novice computer programmers. It has witnessed an enormous surge in its acceptance after the announcement and growth of the Raspberry Pi, for which Python is the legitimately documented programming language. Python provides a diversity of convenient built-in data structures, like lists, sets, and dictionaries. We can also use it for various system management, text processing, and internet-related tasks. Unlike many languages, its core language is minor and easy to acquire. An actual object-oriented language symbolizes an intuitive method for representing information and actions in a program.

Why to use Python:

The following are the primary factors to use python in day-to-day life:

1. Python is object-oriented Structure supports such concepts as polymorphism, operation overloading and multiple inheritance.
2. Indentation is one of the greatest feature in python
3. It's free (open source) Downloading python and installing python is free and easy
4. It's Powerful
   - Dynamic typing
   - Built-in types and tools
   - Library utilities
   - Third party utilities (e.g. Numeric, NumPy, sciPy)
   - Automatic memory management
5. It's Portable
   - Python runs virtually every major platform used today
   - As long as you have a compaitable python interpreter installed, python programs will run in

exactly the same manner, irrespective of platform.

6. It’s easy to use and learn
   - No intermediate compile
   - Python Programs are compiled automatically to an intermediate form called byte code, which the interpreter then reads.
   - This gives python the development speed of an interpreter without the performance loss inherent in purely interpreted languages.
   - Structure and syntax are pretty intuitive and easy to grasp.
7. Interpreted Language Python is processed at runtime by python Interpreter
8. Interactive Programming Language Users can interact with the python interpreter directly for writing the programs
9. Straight forward syntax The formation of python syntax is simple and straight forward which also makes it popular.

Installing on Windows

1. **Download:** Go to the official Python website, navigate to the Downloads section, and select the latest version for Windows.
2. **Run Installer:** Open the downloaded .exe file.
3. **Configure: Important:** Check the box that says "Add Python to PATH" or "Add Python 3.x to PATH".
4. **Install:** Click "Install Now".
5. **Verify:** Open Command Prompt, type python --version, and press Enter to confirm.

Data types:

The data stored in memory can be of many types. For example, a student roll number is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Int:

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

```
>>> print(24656354687654+2)
24656354687656
>>> print(20)
20
```

```
>>> print(0b10)
2
>>> print(0B10)
2
>>> print(0X20)
32
>>> 20
20
>>> 0b10
2
>>> a=10
>>> print(a)
10
```

Float:

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

Float can also be scientific numbers with an "e" to indicate the power of 10.

```
>>> y=2.8
>>> y
2.8
>>> y=2.8
>>> print(type(y))
<class 'float'>
>>> type(.4)
<class 'float'>
>>> 2.
2.0
```

Example:

```
x = 35e3
y = 12E4
z = -87.7e100
print(type(x))
print(type(y))
print(type(z))
```

Output:

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

<class 'float'>

<class 'float'>

Boolean:

Objects of Boolean type may have one of two values, True or False:

>>> type(True)

<class 'bool'>

>>> type(False)

<class 'bool'>

String:

1. Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes.

- 'hello' is the same as "hello".
- Strings can be output to screen using the print function. **For example: print("hello").**

>>> print(" college")

college

>>> type(" college")

<class 'str'>

List:

・It is a general purpose most widely used in data structures

・List is a collection which is ordered and changeable and allows duplicate members.

(Grow and shrink as needed, sequence type, sortable).

・To use a list, you must declare it first. Do this using square brackets and separate values with commas.

・We can construct / create list in many ways.

Ex:

>>> list1=[1,2,3,'A','B',7,8,[10,11]]

>>> print(list1)

[1, 2, 3, 'A', 'B', 7, 8, [10, 11]]

>>> x=list()

>>> x

[]

```
>>> tuple1=(1,2,3,4)
>>> x=list(tuple1)
>>> x
[1, 2, 3, 4]
```

Variables:

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Assigning Values to Variables:

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

Expressions:

An expression is a combination of values, variables, and operators. An expression is evaluated using assignment operator.

Examples: Y=x + 17

```
>>> x=10
>>> z=x+20
>>> z
30
>>> x=10
>>> y=20
>>> c=x+y
>>> c
30
```

A value all by itself is a simple expression, and so is a variable.

```
>>> y=20
>>> y
20
```

Python also defines expressions only contain identifiers, literals, and operators. So,

Identifiers: Any name that is used to define a class, function, variable module, or object is an identifier.

Literals: These are language-independent terms in Python and should exist independently in any programming language. In Python, there are the string literals, byte literals, integer literals, floating point literals, and imaginary literals.

Operators: In Python you can implement the following operations using the corresponding token

Statements:

A statement is an instruction that the Python interpreter can execute. We have normally two basic statements, the assignment statement and the print statement. Some other kinds of statements that are if statements, while statements, and for statements generally called as control flows.

Examples:

An assignment statement creates new variables and gives them values:

```
>>> x=10
```

Precedence of Operators:

Operator precedence affects how an expression is evaluated.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first multiplies 3*2 and then adds into 7.

Example 1:

```
>>> 3+4*2
11
```

Multiplication gets evaluated before the addition operation

```
>>> (10+10)*2
40
```

Parentheses () overriding the precedence of the arithmetic operators

Example 2:

```
a = 20
b = 10
c = 15
d = 5
```

```
e = 0
e = (a + b) * c / d #( 30 * 15 ) / 5
print("Value of (a + b) * c / d is ", e)
```

Comments:

Single-line comments begins with a hash(#) symbol and is useful in mentioning that the whole line should be considered as a comment until the end of line.

A Multi line comment is useful when we need to comment on many lines. In python, triple double quote(“ “ “) and single quote(‘ ‘ ‘)are used for multi-line commenting.

Modules: Python module can be defined as a python program file which contains a python code including python functions, class, or variables. In other words, we can say that our python code file saved with the extension (.py) is treated as the module. We may have a runnable code inside the python module. A module in Python provides us the flexibility to organize the code in a logical way. To use the functionality of one module into another, we must have to **import** the specific module.

Syntax:

```
import <module-name>
```

Every module has its own functions, those can be accessed with . (dot)

Note: In python we have help ()

Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To quit this help utility and return to the interpreter, just type "quit".

Some of the modules like os, date, and calendar so on.....

```
>>> import sys
>>> print(sys.version)
3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)]
>>> print(sys.version_info)
sys.version_info(major=3, minor=8, micro=0, releaselevel='final', serial=0)
>>> print(calendar.month(2021,5))
      May 2021
Mo Tu We Th Fr Sa Su
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31

>>> print(calendar.isleap(2020))
True
>>> print(calendar.isleap(2017))
```

Functions and its use: Function is a group of related statements that perform a specific task. Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable. It avoids repetition and makes code reusable.

Basically, we can divide functions into the following two types:

1. **Built-in functions -** Functions that are built into Python.

Ex: abs(),all().ascii(),bool().....so on....

```
integer = -20
print('Absolute value of -20 is:', abs(integer))
```

Output:

Absolute value of -20 is: 20

User-defined functions - Functions defined by the users themselves.

```
def add_numbers(x,y):
sum = x + y
return sum
print("The sum is", add_numbers(5, 20))
```

Output:

The sum is 25

Parameters and arguments:

Parameters are passed during the definition of function while Arguments are passed during the function call**.**

Example:

```
#here a and b are parameters
def add(a,b): #//function definition
return a+b
#12 and 13 are arguments
#function call
result=add(12,13)
print(result)
```

Output:

C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/paraarg.py

25

There are three types of Python function arguments using which we can call a function.

1. Default Arguments
2. Keyword Arguments
3. Variable-length Arguments

Syntax:

def functionname():

statements

.

.

.

functionname()

Function definition consists of following components:

1. Keyword **def** indicates the start of function header.

2. A function name to uniquely identify it. Function naming follows the same rules of writing identifiers in Python.

3. Parameters (arguments) through which we pass values to a function. They are optional.

4. A **colon (:)** to mark the end of function header.

5. Optional documentation string (docstring) to describe what the function does.

6. One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).

7. An optional return statement to return a value from the function.

Example:

```
def hf():
hello world
hf()
```

#calling function in python:

```
def hf():
print("hello world")
hf()
```

Output:

hello world

```
def hf():
print("hw")
print("gh kfjg 66666")
hf()
hf()
hf()
```

Output:

hw

gh kfjg 66666

hw

gh kfjg 66666

hw

gh kfjg 66666

```
def add(x,y):
c=x+y
print(c)
add(5,4)
```

Output:

9

```
def add(x,y):
c=x+y
return c
print(add(5,4))
```

Keyword Arguments

When we call a function with some values, these values get assigned to the arguments according to their position.

Python allows functions to be called using keyword arguments. When we call functions in this way, the order (position) of the arguments can be changed

```
def func(a, b=5, c=10):
print 'a is', a, 'and b is', b, 'and c is', c
func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

Output:

a is 3 and b is 7 and c is 10

a is 25 and b is 5 and c is 24

a is 100 and b is 5 and c is 50

Default Arguments

Function arguments can have default values in Python.

We can provide a default value to an argument by using the assignment operator (=)

```
def hello(wish,name='you'):
return '{},{}'.format(wish,name)
print(hello("good morning"))
```

Output:

good morning,you

```
def hello(wish,name='you'):
return '{},{}'.format(wish,name) //print(wish + ‘ ‘ + name)
print(hello("good morning","nirosha")) // hello("good morning","nirosha")
```

Variable-length arguments

Sometimes you may need more arguments to process function then you mentioned in the definition. If we don’t know in advance about the arguments needed in function, we can use variable-length arguments also called arbitrary arguments.

For this an asterisk (*) is placed before a parameter in function definition which can hold non-keyworded variable-length arguments and a double asterisk (**) is placed before a parameter in function which can hold keyworded variable-length arguments.

If we use one asterisk (*) like *var, then all the positional arguments from that point till the end are collected as a tuple called ‘var’ and if we use two asterisks (**) before a variable like **var, then all the positional arguments from that point till the end are collected as a dictionary called ‘var’.

```
def wish(*names):
"""This function greets all
the person in the names tuple."""
# names is a tuple with arguments
for name in names:
print("Hello",name)
wish("college","CSE","SIR","MADAM")
```

CONTROL FLOW, LOOPS

Conditional (if):

The if statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.

Syntax:

if expression:

statement(s)

If the boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed. If boolean expression evaluates to FALSE, then the first set of code after the end of the if statement(s) is executed.

if Statement Flowchart:

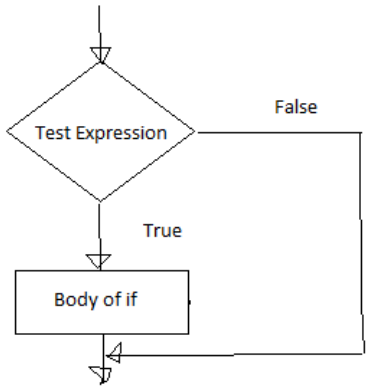


Fig: Operation of if statement

Example: Python if Statement

```
a = 3
if a > 2:
print(a, "is greater")
print("done")
a = -1
if a < 0:
print(a, "a is smaller")
print("Finish")
```

Output:

```
C:/Users/AppData/Local/Programs/Python/Python38-32/pyyy/if1.py
3 is greater
done
-1 a is smaller
Finish
```

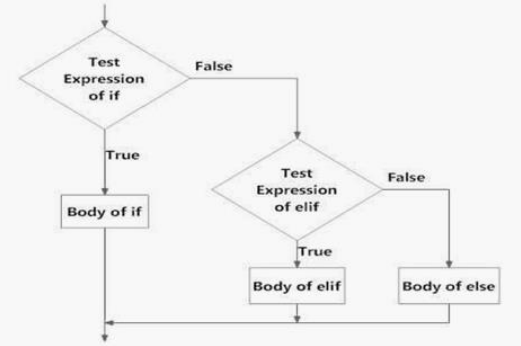
If-elif-else Statement:

The elif statement allows us to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE. Similar to the else, the elif statement is optional. However, unlike else, for which there can be at most one statement, there can be an arbitrary number of elif statements following an if.

Syntax of if – elif - else :

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

Flowchart of if – elif - else:



Example of if - elif – else:

```
a=int(input('enter the number'))
b=int(input('enter the number'))
c=int(input('enter the number'))
if a>b:
print("a is greater")
elif b>c:
print("b is greater")
else:
print("c is greater")
```

Output:

```
C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/ifelse.py
enter the number5
enter the number2
enter the number9
a is greater
```

Iteration:

A loop statement allows us to execute a statement or group of statements multiple times as long as the condition is true. Repeated execution of a set of statements with the help of loops is called iteration.

Loops statements are used when we need to run same code again and again, each time with a different value.

Statements:

In Python Iteration (Loops) statements are of three types:

1. While Loop
2. For Loop
3. Nested For Loops

While loop:

・Loops are either infinite or conditional. Python while loop keeps reiterating a block of code defined inside it until the desired condition is met.

・The while loop contains a boolean expression and the code inside the loop is repeatedly executed as long as the boolean expression is true.

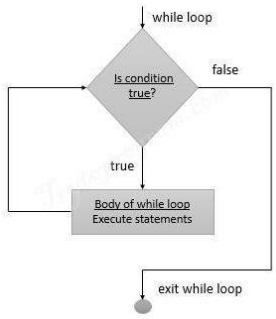
・The statements that are executed inside while can be a single line of code or a block of multiple statements.

Syntax:

while(expression):

Statement(s)

Flowchart:



Example Programs:

1. ----------

```
i=1
while i<=6:
print(" college")
i=i+1
```

output:

```
C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/wh1.py
 college
 college
 college
 college
 college
```

college

For loop:

Python **for loop** is used for repeated execution of a group of statements for the desired number of times. It iterates over the items of lists, tuples, strings, the dictionaries and other iterable objects

Syntax: for var in sequence:

Statement(s) A sequence of values assigned to var in each iteration

Holds the value of item

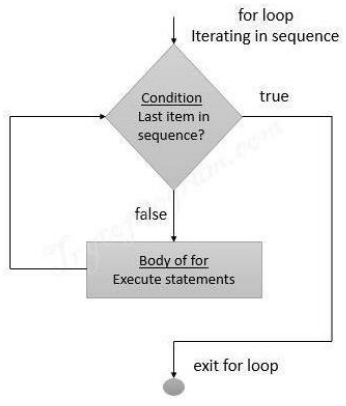
in sequence in each iteration

Sample Program:

```
numbers = [1, 2, 4, 6, 11, 20]
seq=0
for val in numbers:
seq=val*val
print(seq)
```

Output:

```
C:/Users//AppData/Local/Programs/Python/Python38-32/fr.py
1
4
16
```



Iterating over a list:

```
#list of items

list = ['C','O','C','E','T']

i = 1

#Iterating over the list

for item in list:

print ('college ',i,' is ',item)

i = i+1
```

Output:

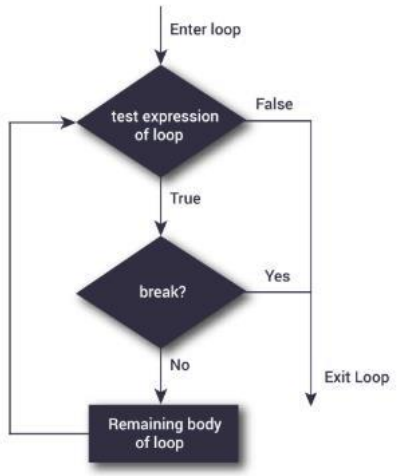
```
C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/lis.py

college 1 is C

college 2 is O

college 3 is C
```

In Python, **break and continue** statements can alter the flow of a normal loop. Sometimes we wish to terminate the current iteration or even the whole loop without checking test expression. The break and continue statements are used in these cases.

Break:

The break statement terminates the loop containing it and control of the program flows to the statement immediately after the body of the loop. If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

Flowchart:



The following shows the working of break statement in for and while loop:

for var in sequence:

code inside for loop

If condition:

break (if break condition satisfies it jumps to outside loop)

    # code inside for loop

    # code outside for loop

while test expression

    # code inside while loop

If condition:

break (if break condition satisfies it jumps to outside loop)

    # code inside while loop

    # code outside while loop

Example:

for val in " COLLEGE":

if val == " ":

break

print(val)

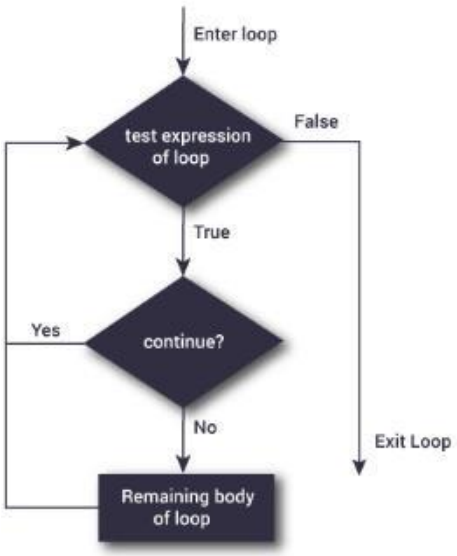
print("The end")

Output:

The end

Continue:

- The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.



Example:

```
# Program to show the use of continue statement inside loops
for val in "string":
if val == "i":
continue
print(val)
print("The end")
```

Output:

```
C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/cont.py
s
t
r
n
g
The end
```

String module:

This module contains a number of functions to process standard Python strings. In recent versions, most functions are available as string methods as well. It's a built-in module and we have to **import** it before using any of its constants and classes

Syntax: import string

Note:

help(string) --- gives the information about all the variables ,functions, attributes and classes to be used in string module.

Example:

```
import string
print(string.ascii_letters)
print(string.ascii_lowercase)
print(string.ascii_uppercase)
print(string.digits)
print(string.hexdigits)
#print(string.whitespace)
print(string.punctuation)
```

Output:

```
C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/strrmodl.py
========================================
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
0123456789abcdefABCDEF
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

Files, Exceptions, Modules, Packages:

Files and exception:

A file is some information or data which stays in the computer storage devices. Python gives you easy ways to manipulate these files. Generally files divide in two categories, text file and binary file. Text files are simple text where as the binary files contain binary data which is only readable by computer.

- **Text files:** In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is the new line character (‘\n’) in python by default.
- **Binary files:** In this type of file, there is no terminator for a line and the data is stored after converting it into machine understandable binary language.

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

Text files:

We can create the text files by using the syntax:

Variable name=open (“file.txt”, file mode)

For ex: f= open ("hello.txt","w+")

・We declared the variable f to open a file named hello.txt. **Open** takes 2 arguments, the file that we want to open and a string that represents the kinds of permission or operation we want to do on the file

・Here we used "w" letter in our argument, which indicates write and the plus sign that means it will create a file if it does not exist in library The available option beside "w" are "r" for read and "a" for append and plus sign means if it is not there then create it

File Modes in Python

Mode	**Description**
'r'	This is the default mode. It Opens file for reading.
'w'	This Mode Opens file for writing. If file does not exist, it creates a new file. If file exists it truncates the file.
'x'	Creates a new file. If file already exists, the operation fails.
'a'	Open file in append mode. If file does not exist, it creates a new file.
't'	This is the default mode. It opens in text mode.
'b'	This opens in binary mode.
'+'	This will open a file for reading and writing (updating)

Reading and Writing files:

 # Write a python program to open and read a file

```
a=open(“one.txt”,”r”)
print(a.read())
a.close()
```

Output:

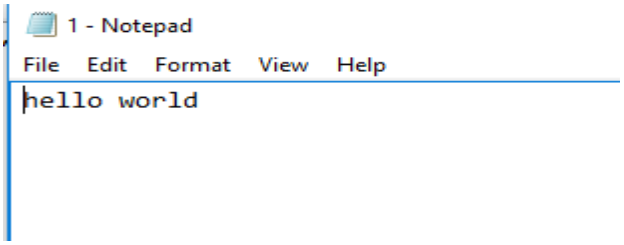
```
C:/Users//AppData/Local/Programs/Python/Python38-32/filess/f1.py
welcome to python programming
```

f.close() ---- This will close the instance of the file somefile.txt stored

 # Write a python program to open and write “hello world” into a file?

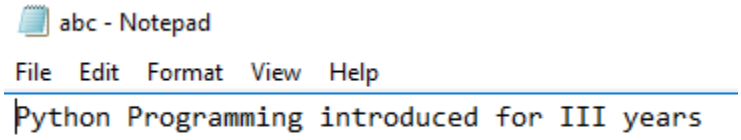
```
f=open("1.txt","a")
f.write("hello world")
f.close()
```

Output:



 # Write a python program to open and write the content to file and read it.

```
fo=open("abc.txt","w+")
fo.write("Python Programming")
print(fo.read())
fo.close()
```

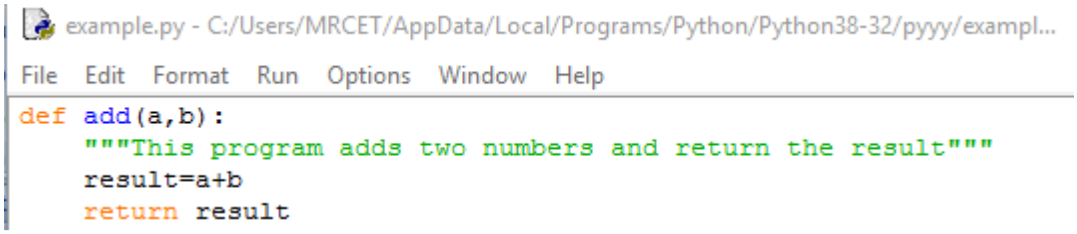


Modules (Date, Time, os, calendar, math):

- Modules refer to a file containing Python statements and definitions We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code.
- We can define our most used functions in a module and import it, instead of copying their definitions into different programs.
- **Modular programming** refers to the process of breaking a large, unwieldy programming task into separate, smaller, more manageable subtasks or **modules**.

A file containing Python code, for e.g.: example.py, is called a module and its module name would be example.

```
>>> def add(a,b):
result=a+b
return result
"""This program adds two numbers and return the result""“
```



Here, we have defined a function add() inside a module named example. The function takes in two numbers and returns their sum.

How to import the module is:

- We can import the definitions inside a module to another module or the Interactive interpreter in Python.
- We use the import keyword to do this. To import our previously defined module example we type the following in the Python prompt.
- Using the module name we can access the function using dot (.) operation. For Eg:

```
>>> import example
>>> example.add(5,5)
10
```

- Python has a ton of standard modules available. Standard modules can be imported the same way as we import our user-defined modules.

Reloading a module:

```
def hi(a,b):
print(a+b)
hi(4,4)
```

Output:

```
C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/add.py
8
>>> import add
8
>>> import add
>>> import add
>>>
```

Python provides a neat way of doing this. We can use the reload() function inside the imp module to reload a module. This is how its done.

- >>> import imp
- >>> import my_module
- This code got executed >>> import my_module >>> imp.reload(my_module) This code got executed <module 'my_module' from '.\\my_module.py'>how its done.

```
>>> import imp
>>> import add
>>> imp.reload(add)
8
<module 'add' from 'C:/Users//AppData/Local/Programs/Python/Python38-
32/pyyy\\add.py'>
```

Datetime module:

 # Write a python program to display date, time

```
>>> import datetime
>>> a=datetime.datetime(2019,5,27,6,35,40)
>>> a
datetime.datetime(2019, 5, 27, 6, 35, 40)
```

 # write a python program to display date

```
import datetime
a=datetime.date(2000,9,18)
print(a)
```

Output:

```
C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/d1.py =
2000-09-18
```

 # write a python program to display time

```
import datetime
a=datetime.time(5,3)
print(a)
```

Output:

```
C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/d1.py =
05:03:00
```

#write a python program to print date, time for today and now.

```
import datetime
a=datetime.datetime.today()
b=datetime.datetime.now()
print(a)
print(b)
```

Output:

```
C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/d1.py =
2019-11-29 12:49:52.235581
2019-11-29 12:49:52.235581
```

#write a python program to add some days to your present date and print the date added.

```
import datetime
a=datetime.date.today()
b=datetime.timedelta(days=7)
```

print(a+b)

Output:

C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/d1.py =

2019-12-06

#write a python program to print the no. of days to write to reach your birthday

import datetime

a=datetime.date.today()

b=datetime.date(2020,5,27)

c=b-a

print(c)

Output:

C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/d1.py =

180 days, 0:00:00

#write an python program to print date, time using date and time functions

import datetime

t=datetime.datetime.today()

print(t.date())

print(t.time())

Output:

C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/d1.py =

2019-11-29

12:53:39.226763

Time module:

#write a python program to display time.

import time

print(time.time())

Output:

C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/t1.py =

1575012547.1584706

#write a python program to get structure of time stamp.

import time

print(time.localtime(time.time()))

Output:

C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/t1.py =

time.struct_time(tm_year=2019, tm_mon=11, tm_mday=29, tm_hour=13, tm_min=1,

tm_sec=15, tm_wday=4, tm_yday=333, tm_isdst=0)

#write a python program to make a time stamp.

```
import time
a=(1999,5,27,7,20,15,1,27,0)
print(time.mktime(a))
```

Output:

C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/t1.py =

927769815.0

#write a python program using sleep().

```
import time
time.sleep(6) #prints after 6 seconds
print("Python Lab")
```

Output:

C:/Users//AppData/Local/Programs/Python/Python38-32/pyyy/t1.py =

Python Lab (#prints after 6 seconds)