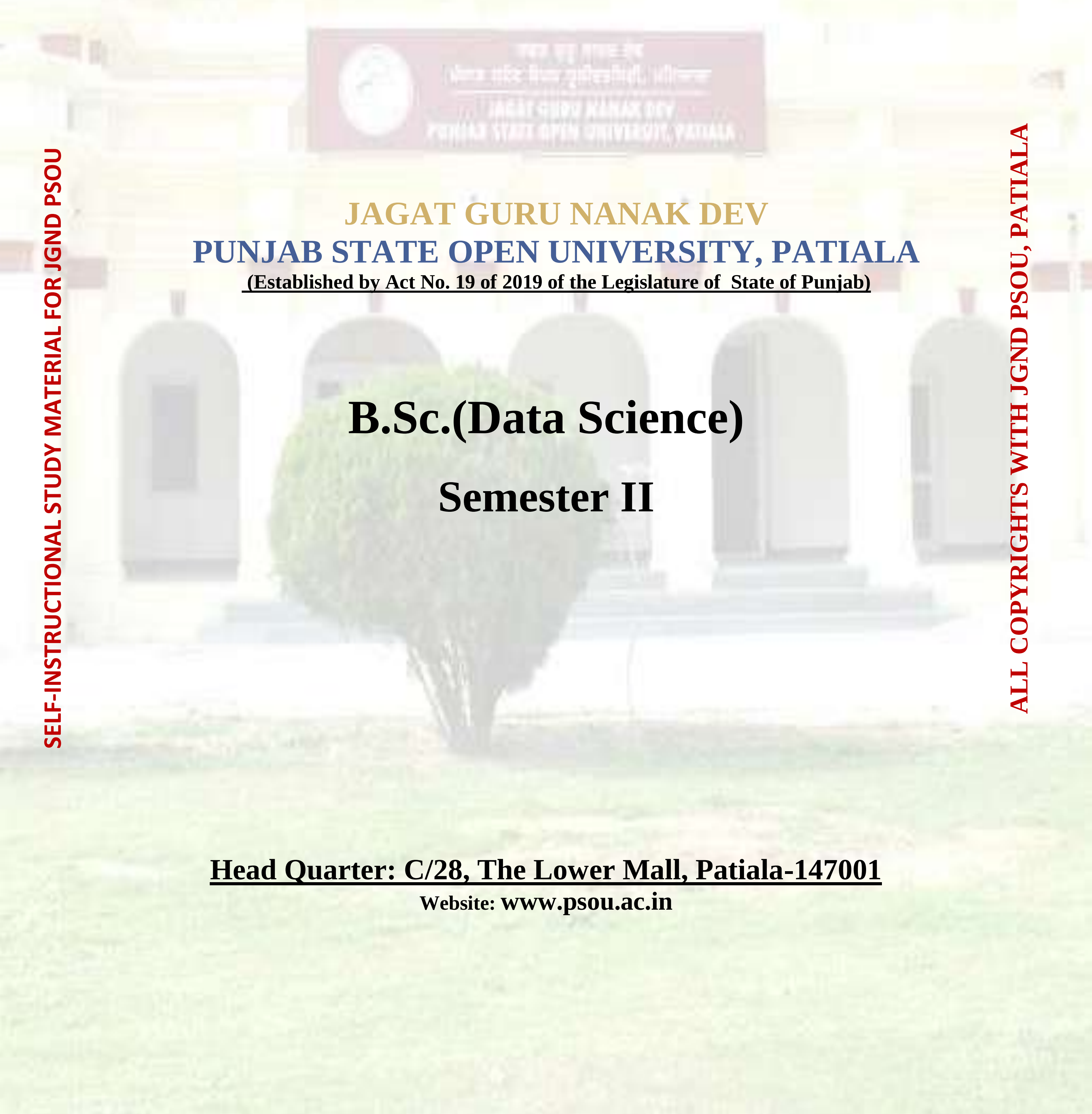
ਜਗਤ ਗੁਰੂ ਨਾਨਕ ਦੇਵ
ਪੰਜਾਬ ਸਟੇਟ ਓਪਨ ਯੂਨੀਵਰਸਿਟੀ
ਪਟਿਆਲਾ

The Motto of Our University
(SEWA)

SKILL ENHANCEMENT
EMPLOYABILITY
WISDOM
ACCESSIBILITY



JAGAT GURU NANAK DEV
PUNJAB STATE OPEN UNIVERSITY, PATIALA

(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

B.Sc.(Data Science)

Semester II

Head Quarter: C/28, The Lower Mall, Patiala-147001
Website: www.psou.ac.in

The Study Material has been prepared exclusively under the guidance of Jagat Guru Nanak Dev Punjab State Open University, Patiala, as per the syllabi prepared by Committee of experts and approved by the Academic Council.

COURSE COORDINATOR AND EDITOR:

Dr. Amitoj Singh
Associate Professor
School of Sciences and Emerging Technologies
Jagat Guru Nanak Dev Punjab State Open University

Code	**Subject**	**Nature of Course**	**Credits**
BSDB31201T	Operating System	CC-3	4
BSDB31202T	Statistical Foundation	DSC-5	4
BSDB31203T	Introduction to Logic	DSC-6	4
BSDB31201P	Operating System Lab	CC-3	2
BSDB31202P	Statistical Foundation Lab	DSC-7	2
BSDB31203P	Introduction to Logic Lab	DSC-8	2
AE2B31204T	Environmental Studies	AECC-2	4

ਜਗਤ ਗੁਰੂ ਨਾਨਕ ਦੇਵ
ਪੰਜਾਬ ਸਟੇਟ ਓਪਨ ਯੂਨੀਵਰਸਿਟੀ
ਪਟਿਆਲਾ

The Motto of Our University
(SEWA)
SKILL ENHANCEMENT
EMPLOYABILITY
WISDOM
ACCESSIBILITY

JAGAT GURU NANAK DEV
PUNJAB STATE OPEN UNIVERSITY, PATIALA
(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

B.Sc.(Data Science)

Semester II

BSDB31201T

Operating Systems

Head Quarter: C/28, The Lower Mall, Patiala-147001
Website: www.psou.ac.in

The Study Material has been prepared exclusively under the guidance of Jagat Guru Nanak Dev Punjab State Open University, Patiala, as per the syllabi prepared by Committee of experts and approved by the Academic Council.

COURSE COORDINATOR AND EDITOR:

Dr. Amitoj Singh
Associate Professor
School of Sciences and Emerging Technologies
Jagat Guru Nanak Dev Punjab State Open University

LIST OF CONSULTANTS/ CONTRIBUTORS

Sr. No.	Name
1	Dr. Rohit Sachdeva
2	Dr. Amitoj Singh
3	Dr. Gurpreet Singh

JAGAT GURU NANAK DEV PUNJAB STATE OPEN UNIVERSITY, PATIALA
(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

PREFACE

Jagat Guru Nanak Dev Punjab State Open University, Patiala was established in December 2019 by Act 19 of the Legislature of State of Punjab. It is the first and only Open University of the State, entrusted with the responsibility of making higher education accessible to all, especially to those sections of society who do not have the means, time or opportunity to pursue regular education.

In keeping with the nature of an Open University, this University provides a flexible education system to suit every need. The time given to complete a programme is double the duration of a regular mode programme. Well-designed study material has been prepared in consultation with experts in their respective fields.

The University offers programmes which have been designed to provide relevant, skill-based and employability-enhancing education. The study material provided in this booklet is self-instructional, with self-assessment exercises, and recommendations for further readings. The syllabus has been divided in sections, and provided as units for simplification.

The University has a network of 10 Learner Support Centres/Study Centres, to enable students to make use of reading facilities, and for curriculum-based counselling and practicals. We, at the University, welcome you to be a part of this instituition of knowledge.

Prof. Anita Gill
Dean Academic Affairs

B.Sc. (Data Science)
Core Course(CC)
Semester II
BSDB31201T: Operating Systems

Total Marks: 100
External Marks: 70
Internal Marks: 30
Credits: 4
Pass Percentage: 35%

Objective

Understanding basics of operating system viz. system programs, system calls, user mode and kernel mode. Working with CPU scheduling algorithms for specific situation, and analyze the environment leading to deadlock and its rectification. Exploring memory management techniques viz. caching, paging, segmentation, virtual memory, and thrashing.

INSTRUCTIONS FOR THE PAPER SETTER/EXAMINER

1. The syllabus prescribed should be strictly adhered to.
2. The question paper will consist of three sections: A, B, and C. Sections A and B will have four questions from the respective sections of the syllabus and will carry 10 marks each. The candidates will attempt two questions from each section.
3. Section C will have fifteen short answer questions covering the entire syllabus. Each question will carry 3 marks. Candidates will attempt any ten questions from this section.
4. The examiner shall give a clear instruction to the candidates to attempt questions only at one place and only once. Second or subsequent attempts, unless the earlier ones have been crossed out, shall not be evaluated.
5. The duration of each paper will be three hours.

INSTRUCTIONS FOR THE CANDIDATES

Candidates are required to attempt any two questions each from the sections A and B of the question paper and any ten short questions from Section C. They have to attempt questions only at one place and only once. Second or subsequent attempts, unless the earlier ones have been crossed out, shall not be evaluated.

SECTION A

Unit- I: Introduction and System Structures: Computer-System Organization, Computer-System Architecture, Operating-System Structure, Operating-System Operations, Process Management, Memory Management, Storage Management, Protection and Security, Computing Environments, Operating-System Services, User and Operating-System Interface, System Calls, Types of System Calls, System Programs.

Unit II: Process Management: Process Concept, Process Scheduling, Operations on Processes, Multi-threaded programming: Multithreading Models, Process Scheduling: Basic Concepts, Scheduling Criteria, and Scheduling Algorithms.

Unit III: Deadlock: System Model, Deadlock Characterization, Methods for Handling Deadlocks, Deadlock Prevention, Deadlock Avoidance, Deadlock Detection, Recovery from Deadlock.

Unit IV: Memory Management: Basic Hardware, Address Binding, Logical and Physical Address, Dynamic linking and loading, Swapping, Contiguous Memory Allocation, Segmentation, Paging, Demand Paging, Page Replacement algorithms

SECTION B

Unit V: File Systems: File Concept, Access Methods, Directory and Disk Structure, File-System Structure, File-System Implementation, Directory Implementation, Allocation Methods, Free-Space Management.

Unit VI: **Introduction to Linux:** Linux's shell, Kernel, Features of Linux, Using file system: Filenames, Introduction to different types of directories: Parent, Subdirectory, Home directory; rules to name a directory, Important directories in Linux File System,

Unit VII: Linux Commands: cal, date, echo, bc, who, cd, mkdir, rmdir, ls, cat cp, rm, mv, more, gzip, tar, File ownership, file permissions, chmod, Directory permission, change file ownership,

Unit VIII: Shell Scripting: Creating and Executing Shell Programs, Using variables: Assigning a value to a variable, Accessing the value of a variable, Positional Parameters and other Built-In Shell Variables; Special Characters, Conditional Statements : if Statement, case Statement; Iteration Statements : for Statement, while Statement, until Statement

Suggested Readings

1. A Silberschatz, P.B. Galvin, G. Gagne, Operating Systems Concepts, 8th Edition, John Wiley Publications, 2009
2. A.S. Tanenbaum, Modern Operating Systems, 3rd Edition, Pearson Education, 2014
3. G. Nutt, Operating Systems: A Modern Perspective, 2nd Edition Pearson Education, 2000
4. S. Das, Unix Concepts and Applications, 4th edition, McGraw Hill Education, 2017

JAGAT GURU NANAK DEV PUNJAB STATE OPEN UNIVERSITY, PATIALA
(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

BSDB31201T: OPERATING SYSTEMS

COURSE COORDINATOR AND EDITOR: DR. AMITOJ SINGH

UNIT NO.	UNIT NAME
UNIT 1	INTRODUCTION AND SYSTEM STRUCTURES
UNIT 2	PROCESS MANAGEMENT
UNIT 3	DEADLOCK
UNIT 4	MEMORY MANAGEMENT
UNIT 5	FILE SYSTEMS
UNIT 6	INTRODUCTION TO LINUX
UNIT 7	LINUX COMMANDS
UNIT 8	SHELL SCRIPTING

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT I: INTRODUCTION AND SYSTEM STRUCTURES

STRUCTURE

1.0 OBJECTIVE

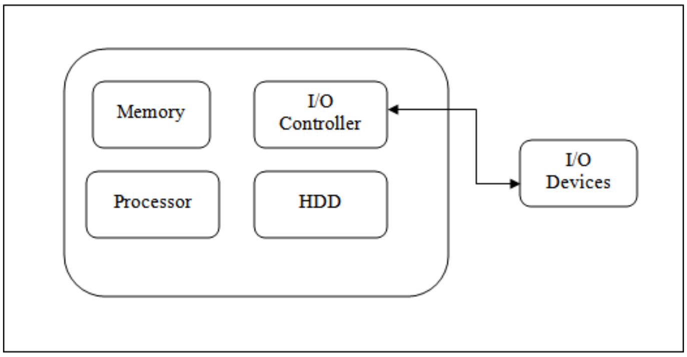
To understand the following

- Computer-System Architecture
- Operating-System Structure, Operating-System Operations
- Operating-System Services, User and Operating-System Interface
- System Calls, Types of System Calls, System Programs.

1.1 Operating System

- An operating system is a program which manages all the computer hardware.
- It provides the base for application program and acts as an intermediary between a user and thecomputer hardware.
- The operating system has two objectives such as:
    - Firstly, an operating system controls the computer's hardware.
    - The second objective is to provide an interactive interface to the user and interpretcommands so that it can communicate with the hardware.
- The operating system is very important part of almost every computer system.

Managing Hardware



- The prime objective of operating system is to manage & control the various hardware resourcesof a computer system.
- These hardware resources include processer, memory, and disk space and so on.
- The output result was display in monitor. In addition to communicating with the hardware the operating system provides on error handling procedure and display an error notification.
- If a device not functioning properly, the operating system cannot be communicate with thedevice.

Providing an Interface

- It provides a stable and consistent way for applications to deal with the hardware without theuser having known details of the hardware.

- If the program is not functioning properly, the operating system again takes control, stops theapplication and displays the appropriate error message.
- Computer system components are divided into 5 parts
  - Computer hardware
  - Operating system
  - Utilities
  - Application programs
  - End user
- The operating system controls and coordinate a user of hardware and various applicationprograms for various users.
- It is a program that directly interacts with the hardware.
- The operating system is the first encoded with the Computer and it remains on the memory alltime thereafter.

System Goals

- The purpose of an operating system is to be provided an environment in which an user canexecute programs.
- Its primary goals are to make the computer system convenience for the user.
- Its secondary goals are to use the computer hardware in efficient manner.

1.2 VIEW OF OPERATING SYSTEM

- **User view:**The user view of the computer varies by the interface being used. The examplesare -windows XP, vista, windows 7 etc. Most computer user sit in the in front of personal computer (pc) in this case the operating system is designed mostly for easy use with some attention paid to resource utilization. Some user sit at a terminal connected to amainframe/minicomputer. In this case other users are accessing the same computer through the other terminals. There user are share resources and may exchange the information. The operating system in this case is designed to maximize resources utilization to assume that all available CPU time, memory and I/O are used efficiently and no individual user takes more than his/her fair and share.The other users sit at workstations connected to network of other workstations and servers. These users have dedicated resources but they share resources such as networking and servers like file, compute and print server. Here the operating system is designed to compromise between individual usability and resource utilization.
- **System view:** From the computer point of view the operating system is the program which is most intermediate with the hardware. An operating system has resources as hardware and software which may be required to solve a problem like CPU time, memory space, file storage space and I/O devices and so on. That’s why the operating

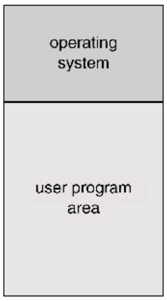
system acts as manager of these resources. Another view of the operating system is it is a control program. A control program manages the execution of user programs to present the errors in proper use of the computer. It is especially concerned of the user the operation and controls the I/O devices.

1.3 TYPES OF OPERATING SYSTEM

1. **Mainframe System:** It is the system where the first computer used to handle many commercialscientific applications. The growth of mainframe systems traced from simple batch system where the computer runs one and only one application to time shared systems which allowed for user interaction with the computer system

   a. **Batch /Early System:** Early computers were physically large machine. The common input devices were card readers, tape drivers. The common output devices were line printers, tape drivers and card punches. In these systems the user did not interact directlywith the computer system. Instead the user preparing a job which consists of programming data and some control information and then submitted it to the computer operator after some time the output is appeared.

Memory Layout for a Simple Batch System



The output in these early computer was fairly simple is main task was to transfer control automatically from one job to next. Theoperating system always resides in the memory. To speed up processing operators batched the jobs with similar needs and ran then together as a group. The disadvantages of batch system are that in this execution environment the CPU is often idle because thespeed up of I/O devices is much slower than the CPU.

   b. Multiprogrammed System: Multiprogramming concept increases CPU utilization by organization jobs so that the CPU always has one job to execute the idea behind multiprogramming concept. This set of job is subset of the jobs kept in the job pool. The operating system picks and beginning to execute one of the jobs in the memory. In this environment the operating system simply switches and executes another job. When a job needs to wait the CPU is simply switched to another job and so on. The multiprogramming operating system is sophisticated because the operating system makes decisions for the user. This is known asscheduling. If several jobs are ready to run at the same time the system choose one among them. This is known as CPU scheduling. The disadvantages of the multiprogrammedsystem are

- It does not provide user interaction with the computer system during the program execution.
- The introduction of disk technology solved these problems rather than reading the cards from card reader into disk. This form of processing is known as spooling.

SPOOL stands for simultaneous peripheral operations online. It uses the disk as a huge buffer for reading from input devices and for storing output data until the output devices accept them. It is also use for processing data at remote sides. The remote processing is done and its own speed with no CPU intervention. Spooling overlaps the input, output one job with computation of other jobs. Spooling has a beneficial effect onthe performance of the systems by keeping both CPU and I/O devices working at much higher time.

c. **Time Sharing System:** The time sharing system is also known as multi user systems. TheCPU executes multiple jobs by switching among them but the switches occurs so frequently that the user can interact with each program while it is running. An interactive computer system provides direct communication between a user and system. The user gives instruction to the operating systems or to a program directly using keyboard or mouse and wait for immediate results. So the response time will be short. The time sharing system allows many users to share the computer simultaneously. Since each action in this system is short, only a little CPU time is needed for each user. The system switchesrapidly from one user to the next so each user feels as if the entire computer system is dedicated to his use, even though it is being shared by many users. The disadvantages oftime sharing system are:

- It is more complex than multiprogrammed operating system
- The system must have memory management & protection, since several jobs are keptin memory at the same time.
- Time sharing system must also provide a file system, so disk management is required.
- It provides mechanism for concurrent execution which requires complex CPU scheduling schemes.

2. **Personal Computer System/Desktop System:** Personal computer appeared in 1970's. They are microcomputers that are smaller & less expensive than mainframe systems. Instead of maximizing CPU & peripheral utilization, the systems opt for maximizing user convenience & responsiveness. At first file protection was not necessary on a personal machine. But when other computers 2nd other users can access the files on a pc file protection becomes necessary.The lack of protection made if easy for malicious programs to destroy data on such systems. These programs may be self replicating& they spread via

worm or virus mechanisms. They can disrupt entire companies or even world wide networks. E.g : windows 98, windows 2000, Linux.

3. **Microprocessor Systems/ Parallel Systems/ Tightly coupled Systems**: These Systems have more than one processor in close communications which share the computer bus, clock, memory & peripheral devices. Ex: UNIX, LINUX. Multiprocessor Systems have 3 main advantages.
   a. **Increased throughput:** No. of processes computed per unit time. By increasing the no. of processors move work can be done in less time. The speed up ratio with N processors is not N, but it is less than N. Because a certain amount of overhead is incurred in keeping all the parts working correctly.
   b. **Increased Reliability:** If functions can be properly distributed among several processors, then the failure of one processor will not halt the system, but slow it down. This ability to continue to operate in spite of failure makes the system fault tolerant.
   c. **Economic scale:** Multiprocessor systems can save money as they can share peripherals,storage & power supplies.

   The various types of multiprocessing systems are:

   - **Symmetric Multiprocessing (SMP):** Each processor runs an identical copy of the operating system & these copies communicate with one another as required. Ex: Encore's version of UNIX for multi max computer. Virtually, all modern operating system including Windows NT, Solaris, Digital UNIX, OS/2 & LINUX now provide support for SMP.
   - **Asymmetric Multiprocessing (Master – Slave Processors):** Each processor is designed for a specific task. A master processor controls the system & schedules & allocates the work to the slave processors. Ex- Sun's Operating system SUNOS version 4provides asymmetric multiprocessing.

4. **Distributed System/Loosely Coupled Systems**: In contrast to tightly coupled systems, the processors do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with each other by various communication lines such as high speed buses or telephone lines. Distributed systems depend on networking for their functionalities. By being able to communicate distributed systems are able to share computational tasks and provide a rich set of features to the users. Networks vary by the protocols used, the distances between the nodes and transport media. TCP/IP is the most common network protocol. The processor is a distributed system varies in size and function. Itmay microprocessors, work stations, minicomputer, and large general purpose computers. Network types are based on the distance between the nodes such as LAN (within a room, flooror building) and WAN (between buildings, cities or countries). The advantages of distributed system are resource sharing, computation speed up, reliability, communication.

5. **Real time Systems:** Real time system is used when there are rigid time requirements on

the operation of a processor or flow of data. Sensors bring data to the computers. The computer analyzes data and adjusts controls to modify the sensors inputs. System that controls scientific experiments, medical imaging systems and some display systems are real time systems. The disadvantages of real time system are:

a. A real time system is considered to function correctly only if it returns the correct result within the time constraints.

b. Secondary storage is limited or missing instead data is usually stored in short term memory or ROM.

c. Advanced OS features are absent.Real time system is of two types such as:

- **Hard real time systems:** It guarantees that the critical task has been completed on time.The sudden task is takes place at a sudden instant of time.
- **Soft real time systems:** It is a less restrictive type of real time system where a critical task gets priority over other tasks and retains that priority until it computes. These have more limited utility than hard real time systems. Missing an occasional deadline is acceptable

  e.g. QNX, VX works. Digital audio or multimedia is included in this category.

It is a special purpose OS in which there are rigid time requirements on the operation of a processor. A real time OS has well defined fixed time constraints. Processing must be done within the time constraint or the system will fail. A real time system is said to function correctly only if it returns the correct result within the time constraint. These systems are characterized by having time as a key parameter.

1.4 Functions of Operation System

The various functions of operating system are as follows:

1. **Process Management**

- A program does nothing unless their instructions are executed by a CPU.A process is a program in execution. A time shared user program such as a complier is a process. A word processing program being run by an individual user on a pc is a process.
- A system task such as sending output to a printer is also a process. A process needs certain resources including CPU time, memory files & I/O devices to accomplish its task.
- These resources are either given to the process when it is created or allocated to it while it is running. The OS is responsible for the following activities of process management.
- Creating & deleting both user & system processes.
- Suspending & resuming processes.
- Providing mechanism for process synchronization.
- Providing mechanism for process communication.
- Providing mechanism for deadlock handling.

2. **Main Memory Management**

The main memory is central to the operation of a modern computer system. Main memory is a large array of words or bytes ranging in size from hundreds of thousand to billions. Main memory stores the quickly accessible data shared by the CPU & I/O device. The central processor reads instruction from main memory during instruction fetch cycle & it both reads & writes data from main memory during the data fetch cycle. The main memory is generally theonly large storage device that the CPU is able to address & access directly. For example, for theCPU to process data from disk. Those data must first be transferred to main memory by CPU generated E/O calls. Instruction must be in memory for the CPU to execute them. The OS is responsible for the following activities in connection with memory management.

- Keeping track of which parts of memory are currently being used & by whom.
- Deciding which processes are to be loaded into memory when memory space becomesavailable.
- Allocating &deallocating memory space as needed.

3. **File Management**

File management is one of the most important components of an OS computer can store information on several different types of physical media magnetic tape, magnetic disk & optical disk are the most common media. Each medium is controlled by a device such as disk drive or tape drive those has unique characteristics. These characteristics include access speed, capacity, data transfer rate & access method (sequential or random).For convenient use of computer system the OS provides a uniform logical view of information storage. The OS abstracts from the physical properties of its storage devices to define a logical storage unit the file. A file is collection of related information defined by its creator. The OS is responsible for the followingactivities of file management.

- Creating & deleting files.
- Creating & deleting directories.
- Supporting primitives for manipulating files & directories.
- Mapping files into secondary storage.
- Backing up files on non-volatile media.

4. **I/O System Management**

One of the purposes of an OS is to hide the peculiarities of specific hardware devices from the user. For example, in UNIX the peculiarities of I/O devices are hidden from the bulk of the OSitself by the I/O subsystem. The I/O subsystem consists of:

- A memory management component that includes buffering, catching & spooling.
- A general device- driver interfaces drivers for specific hardware devices. Only the device driver knows the peculiarities of the specific device to which it is assigned.

5. **Secondary Storage Management**

The main purpose of computer system is to execute programs. These programs with the data they access must be in main memory during execution. As the main memory is too small to accommodate all data & programs & because the data that it holds are lost when power is lost. The computer system must provide secondary storage to back-up main memory. Most modern computer systems are disks as the storage medium to store data & program. The operating system is responsible for the following activities of disk management.

- Free space management.
- Storage allocation.
- Disk scheduling

Because secondary storage is used frequently it must be used efficiently.

1.5 NETWORKING

A distributed system is a collection of processors that don't share memory peripheral devices or a clock. Each processor has its own local memory & clock and the processor communicate with one another through various communication lines such as high speed buses or networks. The processors in the system are connected through communication networks which are configured in a number of different ways. The communication network design must consider message routing & connection strategies are the problems of connection & security.

1.6 PROTECTION OR SECURITY

If a computer system has multi users & allow the concurrent execution of multiple processes then the various processes must be protected from one another's activities. For that purpose, mechanisms ensure that files, memory segments, CPU & other resources can be operated on by onlythose processes that have gained proper authorization from the OS.

1.7 SYSTEM CALLS

System calls provide the interface between a process & the OS. These are usually available in the form of assembly language instruction. Some systems allow system calls to be made directly from a high level language program like C, BCPL and PERL etc. systems calls occur in different ways depending on the computer in use. System calls can be roughly grouped into 5 major categories.

1.7.1 Process Control

- **End, abort:** A running program needs to be able to has its execution either normally (end) orabnormally (abort).
- **Load, execute:** A process or job executing one program may want to load and executes another program.
- **Create Process, terminate process:** There is a system call specifying for the purpose

of creating a new process or job (create process or submit job). We may want to terminate a job or process that we created (terminates process, if we find that it is incorrect or no longer needed).

- **Get process attributes, set process attributes:** If we create a new job or process we should able to control its execution. This control requires the ability to determine & reset the attributes of a job or processes (get process attributes, set process attributes).
- **Wait time:** After creating new jobs or processes, we may need to wait for them to finish theirexecution (wait time).
- **Wait event, signal event:** We may wait for a specific event to occur (wait event). The jobs orprocesses then signal when that event has occurred (signal event).

1.7.2 File Manipulation

- **Create file, delete file:** We first need to be able to create & delete files. Both the system callsrequire the name of the file & some of its attributes.
- **Open file, close file:** Once the file is created, we need to open it & use it. We close the file when we are no longer using it.
- **Read, write, reposition file:** After opening, we may also read, write or reposition the file (rewind or skip to the end of the file).
- **Get file attributes, set file attributes:** For either files or directories, we need to be able to determine the values of various attributes & reset them if necessary. Two system calls get fileattribute & set file attributes are required for their purpose.

1.7.3 Device Management

- **Request device, release device:** If there are multiple users of the system, we first request the device. After we finished with the device, we must release it.
- **Read, write, reposition:** Once the device has been requested & allocated to us, we can read,write & reposition the device.

1.7.4 Information Maintenance

- **Get system data, set system data:** Other system calls may return information about thesystem like number of current users, version number of OS, amount of free memory etc.
- **Get process attributes, set process attributes:** The OS keeps information about all itsprocesses & there are system calls to access this information.

1.7.5 Communication

There are two modes of communication such as:

a. **Message passing model:** Information is exchanged through an inter process communication facility provided by operating system. Each computer in a network has a name by which it is known. Similarly, each process has a process name which is translated toan equivalent identifier by which the OS can refer to it. The get hostid and get processed systems calls to do this translation. These identifiers are then passed to the general purpose open & close calls provided by the file system or to specific open connection system call. Therecipient process must give its permission for communication to take place with an accept connection call. The source of the communication known as client & receiver known as server exchange messages by read message & write message system calls. The closeconnection call terminates the connection.

b. **Shared memory model:** processes use map memory system calls to access regions of memory owned by other processes. They exchange information by reading & writing data inthe shared areas. The processes ensure that they are not writing to the same locationsimultaneously.

1.8 SYSTEM PROGRAMS

System programs provide a convenient environment for program development & execution. Theyare divided into the following categories.

a. File manipulation: These programs create, delete, copy, rename, print & manipulate filesand directories.

b. Status information: Some programs ask the system for date, time & amount of availablememory or disk space, no. of users or similar status information.

i. **File modification:**Several text editors are available to create and modify the contents of filestored on disk.

ii. **Programming language support:** compliers, assemblers & interpreters are provided to the user with the OS.

iii. **Programming loading and execution:** Once a program is assembled or compiled, it mustbe loaded into memory to be executed.

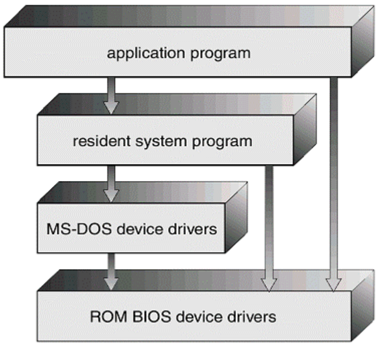
iv. **Communications:** These programs provide the mechanism for creating virtual connectionsamong processes users 2nd different computer systems.

v. **Application programs:** Most OS are supplied with programs that are useful to solve common problems or perform common operations. Ex: web browsers, word processors & text formatters etc.

1.8 SYSTEM STRUCTURE

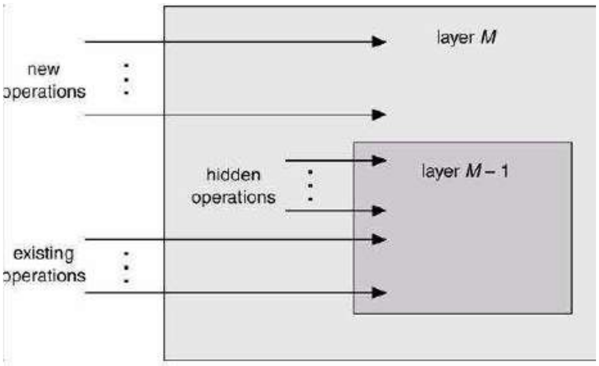
1. **Simple structure:** There are several commercial system that don’t have a well-defined structure such operating systems begins as small, simple & limited systems and then grow beyond their original scope. MS-DOS is an example of such system. It was

not divided intomodules carefully. Another example of limited structuring is the UNIX operating system.



2. **Layered approach:** In the layered approach, the OS is broken into a number of layers (levels) each built on top of lower layers. The bottom layer (layer o) is the hardware & top most layer (layer N) is the user interface. The main advantage of the layered approach is modularity.

- The layers are selected such that each users functions (or operations) & services of only lower layer.



This approach simplifies debugging & system verification, i.e. the first layer can be debugged without concerning the rest of the system. Once the first layer is debugged, its correct functioning is assumed while the 2nd layer is debugged & so on.

If an error is found during the debugging of a particular layer, the error must be on that layer because the layers below it are already debugged. Thus the design & implementation of the system are simplified when the system is broken down into layers.

Layers	Functions
5	User Program
4	I/O Management
3	Operator Process Communication
2	Memory Management
1	CPU Scheduling
0	Hardware

Each layer is implemented using only operations provided by lower layers. A layer doesn't need to know how these operations are implemented; it only needs to know what these operations do.

The layer approach was first used in the operating system. It was defined in six layers.

The main disadvantage of the layered approach is:

- The main difficulty with this approach involves the careful definition of the layers, because a layer can use only those layers below it. For example, the device driver for thedisk space used by virtual memory algorithm must be at a level lower than that of the memory management routines, because memory management requires the ability to usethe disk space.
- It is less efficient than a non layered system (Each layer adds overhead to the system call & the net result is a system call that take longer time than on a non layered system).

1.10 Operating System Services

An operating system provides an environment for the execution of the program. It provides some services to the programs. The various services provided by an operating system are as follows:

- **Program Execution:** The system must be able to load a program into memory and to run that program. The program must be able to terminate this execution either normally or abnormally.
- **I/O Operation:** A running program may require I/O. This I/O may involve a file or a I/O device for specific device. Some special function can be desired. Therefore the operating system must provide a means to do I/O.
- **File System Manipulation:** The programs need to create and delete files by name and read and write files. Therefore the operating system must maintain each and every files correctly.
- **Communication:** The communication is implemented via shared memory or by the technique of message passing in which packets of information are moved between the processes by the operating system.
- **Error detection:** The operating system should take the appropriate actions for the occurrences of any type like arithmetic overflow, access to the illegal memory location andtoo large user CPU time.
- **Research Allocation:** When multiple users are logged on to the system the resources must be allocated to each of them. For current distribution of the resource among the various processes the operating system uses the CPU scheduling run times which determine which process will be allocated with the resource.
- **Accounting:** The operating system keep track of which users use how many and which kindof computer resources.
- **Protection:** The operating system is responsible for both hardware as well as software protection. The operating system protects the information stored in a multiuser computer system.

1.11 PRACTICE EXERCISE

a. Why is the operating system important?

b. What's the main purpose of an OS? What are the different types of OS?

c. What are the benefits of a multiprocessor system?

d. What is a bootstrap program in OS?

e. What is the main purpose of an operating system?

f. What are the different operating systems?

g. What is kernel?

h. What is monolithic kernel?

i. What do you mean by a process?

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT II: PROCESS MANAGEMENT

STRUCTURE

2.0 OBJECTIVES

To understand the following

- Different types of Process Scheduling
- Operations on Processes
- Multi-threaded programming and Model
- CPU Scheduling
- Scheduling Concepts
- Scheduling Criteria
- Scheduling Algorithms

2.1 INTRODUCTION TO PROCESS

2.1.1 Process Basics

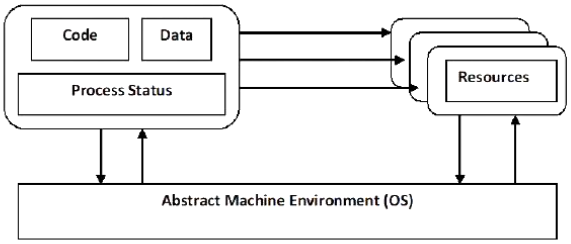
The fundamental activity of an operating system is the creation, management, and termination of processes. Now the question comes to mind, what is a process?

A process is a program under execution or the "animated" existence of a program or an identifiable entity executed on a processor by the operating system.

A process may be defined as an instance of a program in execution. It is also known as a task. An operating system manages each hardware resource attached to the computer by representing it as an abstraction. Abstraction hides unwanted details from the users and programmers allowing them to have a view of the resources in the form, which is convenient to them. A process is an abstract model of a sequential program in execution. The operating system can schedule a process as a unit of work. A process can be identifying in an operating system by its following components:

The object program (or code) to be executed.
The data on which the program will execute.
The status of the process execution.
The stack is associated with a process to store parameters and calling addresses.

A process may be represented schematically as in the figure.



Process Abstraction

2.1.2 Process Description

To control and manage the processes, the Operating system records the information about them in the primary process table. The primary process table is used to keep one entry per process in the operating system. Each entry contains at least one pointer to a process image. The Process Image contains **User Data**: It contains program data that can be modified etc., **Code:** The sequence of instructions (program) to be executed, **Stack:** Each process has one or more stacks associated with it. A stack is used to saved parameters of the process and calling addresses for process, system calls, and **Process Control Block (PCB)** of process in which data needed by the operating system to control the process (attributes and information about the process) is stored.

2.1.3 Process Control Block

The data structure that stores information about a process is called Process Control Block. When a process is initialized, the corresponding process control block of the process is created. Information in a process control block is updated during the transition of process states.

A process control block is a location in the main memory, where various information of a process regarding memory, process, and I/O management is stored. Each process has a single process control block. When a process is completed the process control block is unloaded from the memory. The information stored in a process control block is:

Pointer	**Process State**
Process Number	
Program Counter	
Registers	
Memory Limits **List of open files**	
. . . .	

Figure: Process Control Board (PCB)

Each process has a **priority**, implemented in terms of numbers. The higher priority processes have precedence over lower priority processes. The priority field is used for storing the priority of a process.

A new process can be created from the existing process. The existing process is called the parent of the newly created process. The field, **link** (pointer) to the parent process stores the address of the process control block of the parent process in the main memory. The field, link (pointer) to the child process stores the address of the process control block of the child process in the main memory.

Process State field stores the information about the recent state of the process. The state of the process may be new, waiting, ready, running, and so on.

Process Number filled to store the numeric value which is an identifier of the process.

The PCB stores information regarding the programming environment of a process. The programming environment information includes the value of the registers, stack, and program counter.

A **program counter** is a special register that saves the address of the next instruction to be executed for this process.

A PCB also stores information regarding memory management, such as the number of memory units allocated to the process and the addresses of the memory chunks allocated.

Registers: It includes a general-purpose register, stack pointers, index register, and accumulators, etc. Many register and accumulators etc. Some register and type of register depend upon the computer architecture.

The PCB control block stores file management information. An example of the file management information stored in the process control block is:

- The number of files opens.
- List of the open files.
- The access right of the files opens, such as read-only or read-write.

2.1.4 Role of PCB

The process control block is the most important data structure in an OS. Each Process Control Block contains all of the data about a process that is required by the Operating System. The blocks are read as well as modified by virtually every unit in the Operating System, including those which involved scheduling, interrupt processing, resource allocation, and performance monitoring and analysis of the process.

One can say that the set of process control blocks defines the state of the OS. This brings up an important design issue. Many routines within the OS will need access to information in process control blocks. The provision of direct access to these tables is not difficult. Each process is equipped with a unique ID, and this can be used as an index into a table of pointers to the process control blocks.

The difficulty is not access but rather protection. There are two problems :

1. A bug in a single routine, such as an interrupt handler, could damage process control blocks, which could destroy the system’s ability to manage the affected.
2. A design change in the structure or semantics of the process control block could affect many modules in the OS.

2.2 PROCESS SCHEDULERS

Several types of schedulers can be used in an OS. Schedulers are classified according to the type and duration of processes. Schedulers are classified as:

- Long-term Scheduler/High-Level Scheduler
- Medium-term Scheduler/ Intermediator Scheduler
- Short-term Scheduler/Low-Level Schedular

2.2.1 Long-term Scheduler

Long-term scheduling identifying which programs is to be admitted to the system as new processes. Once a new process is accepted, it may enter the scheduling queues in one of two places:

- If all resources are initially fully available to the new process, they may be admitted to the tail of the ready queue.
- If all resources are not immediately available, the new process may be entered in the blocked-suspended queue until those resources are provided.

Long-term Scheduler plans the CPU scheduling for batch jobs. Processes, which are resource-intensive and have a low priority, are called batch jobs are executed in a group or bunch. An example of a batch job is a user request for printing a bunch of files.

2.2.2 Medium-term Scheduler

Medium-term scheduling is part of the swapping role of an operating system. Medium-term Scheduler plans CPU scheduling for the processes that have been waiting for the completion of another process or an I/O task that requires a long time. A process is suspended or blocked marked as waiting if it is waiting for the completion of a long time I/O task. These processes are removed from the main memory and stored in the swapped-out queue in the secondary memory to create space in the main memory. The swapped-out queue is implemented in the secondary memory for storing the waiting processes that have been swapped out of the main memory. After completion of the I/O operation, the suspended or blocked processes are resumed and placed in the ready queue in the main memory. If the process is waiting for the completion of a short-term I/O task, the process is not swapped out of the main memory and is not handled by the medium-term scheduler. The success of the medium-term schedules is based on the degree of multiprogramming that it can maintain, by keeping as many processes "runnable" as possible.

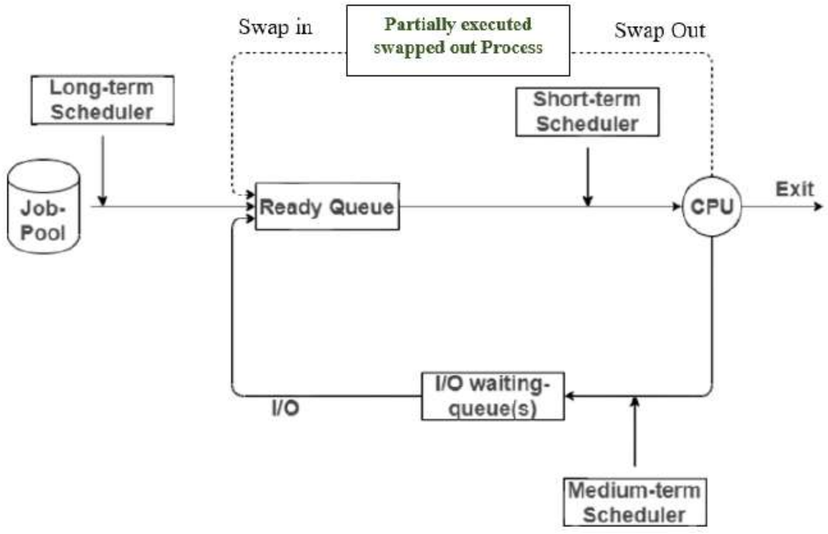
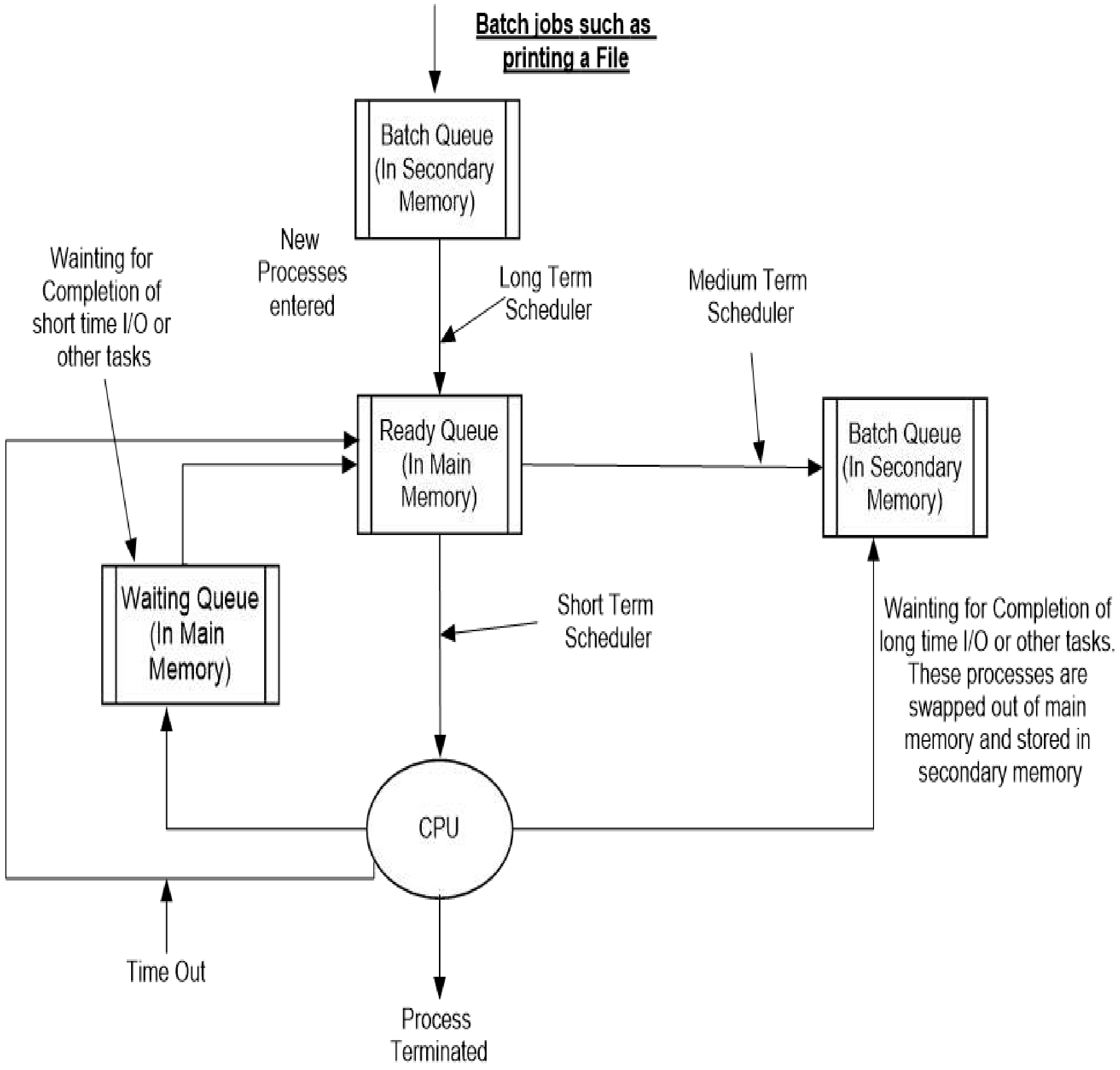


Figure: Working of medium-term scheduler

2.2.3 Short-term Scheduler

Short-term Scheduler plans the scheduling of the processes that are in a ready state. Short-term schedulers retrieve a process from the ready queue and allocate CPU time to it. The process state is changed from ready to run. If an interrupt or time-out occurs the scheduler places the running process back into the ready queue and marks the running process as ready. **The figure below shows how short, medium, long term schedulers work.**



Batch jobs such as printing a File
Batch Queue (In Secondary Memory)
New Processes entered
Long Term Scheduler
Medium Term Scheduler
Wainting for Completion of short time I/O or other tasks
Ready Queue (In Main Memory)
Batch Queue (In Secondary Memory)
Waiting Queue (In Main Memory)
Short Term Scheduler
Wainting for Completion of long time I/O or other tasks. These processes are swapped out of main memory and stored in secondary memory
CPU
Time Out
Process Terminated

Figure: Role of various type schedulers

If a new process is a batch job, it is placed in the batch queue; otherwise, it is ready into the ready queue. The long-term scheduler selects a batch job on a first-come-first-served basis and sends it to the ready queue for execution.

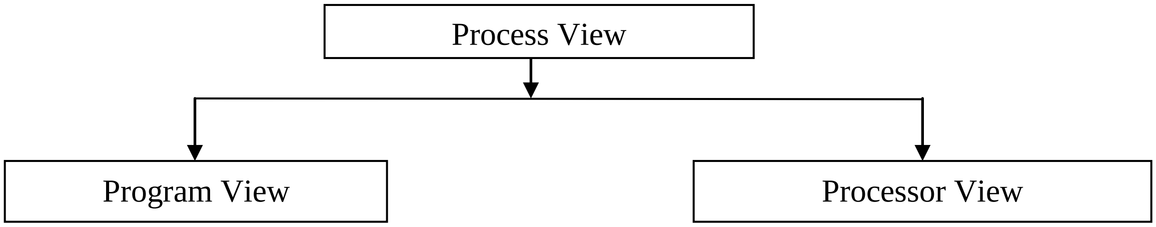
The short-term scheduler selects a process from the ready queue for execution. If a process waits for a short time for completion of an I/O task it is placed in the waiting queue, otherwise, the waiting process is swapped out of the main memory and placed in the swapped-out queue, which is implemented in the secondary memory.

Sr. No.	Long Term	Short Term	Medium Term
1.	It is a job scheduler.	It is a CPU scheduler.	It is swapping.
2.	Speed is less than short term schedular	Speed is very fast.	Speed is in between both.
3.	It controls the degree of Multiprogramming.	Less control over the degree of Multiprogramming.	Reduce the degree of Multiprogramming.
4.	Absent or minimal in time sharing system	Minimal in a time-sharing system	The Time-sharing system uses a medium-term scheduler.
5.	It selects processes from pool and load them into memory for execution.	It selects from among the processes that are ready to execute	The process can be reintroduced into memory and its execution can be continued.
6.	Process state is (new to Ready.)	Process state is (Ready to Running)	Process state is (waiting)
7.	Select a good process, a mix of I/O bound and CPU bound	Select a new process for a CPU quite frequently.	Select suspended Process

2.3 OPERATION ON PROCESSES

Process State

The principal function of a processor is to execute machine instructions residing in the main memory. We can view the process from two points of view: Program View and Processor View.



Different Process View

Program View: Its execution involves a sequence of instructions within that program. The behavior of individual process can be characterized by a list of the sequence of instructions – a ***trace*** of the process

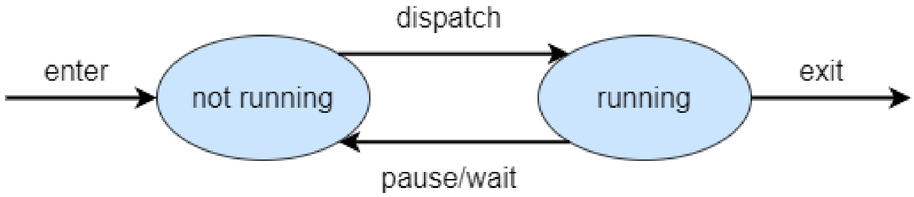
Processor View: It executes instructions from the main memory, as dictated by changing values in the program counter register. The behavior of the processor can be characterized by showing how the traces of various processes are interleaved.

The Simple Two-State Process Model

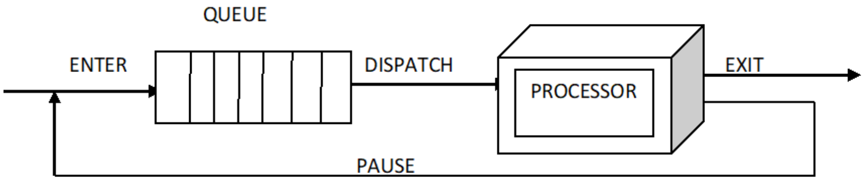
The operating system's principal responsibility is controlling the execution of processes. This includes determining the interleaving pattern for execution and allocating resources to processes. The first step in designing an OS to control processes is to describe the behavior that we would like the processes to exhibit.

We can construct the simplest possible model by observing that, at any time, a process is either being executed by a processor or not. In this model, a process may be in one of two states: **Running or Not Running**, as shown in the figure Two-State Process Model **State transition Diagram**. When the OS creates a new process, it creates a Process Control Block for the process and enters that process into the system in the Not Running state. The process exists, is known to the OS, and is waiting for an opportunity to execute. From time to time, the currently running process will be interrupted and the dispatcher portion of the OS will select some other process to run. The former process moves from the Running state to the Not Running state, and one of the other processes moves to the Running state.

From this simple model, we can already begin to appreciate some of the design elements of the OS. Each process must be represented in some way so that the OS can keep track of it. That is, there must be some information relating to each process, including the current state and location in memory; this is the Process Control Block. Processes that are not running must be kept in some sort of queue, waiting their turn to execute.



Two-State Process Model State transition Diagram



Two-State Process Model Queuing Diagram

Process Creation

When a new process is to be added to those currently being managed, the OS builds the data structures that are used to manage the process and allocates address space in the main memory to the process. These actions constitute the creation of a new process. Four common events leading to the creation of a process. In a batch environment, a process is created in response to the submission of a job. In an interactive environment, a process is created when a new user attempts to log on. In both cases, the OS is responsible for the creation of the new process. An OS may also create a process on behalf of an application. For example, if a user requests that a file be printed, the OS can create a process that will manage the printing. The requesting process can thus proceed independently of the time required to complete the printing task.

Traditionally, the OS created all processes in a way that was transparent to the user or application program, and this is still commonly found with many contemporary operating systems. However, it can be useful to allow one process to cause the creation of another. For example, an application process may generate another process to receive data that the application is generating and to organize those data into a form suitable for later analysis. The new process runs in parallel to the original process and is activated from time to time when new data are available. This arrangement can be very useful in structuring the application. As another example, a server process (which may be a print server or file server) may create a new process for each request that it handles. When the OS creates a new process at the explicit request by another process, this work is known as **process spawning**.

Parent Process

When one process spawns another, the former is referred to as the **parent process**.

Child Process

The spawned process is referred to as the **child process**.

Typically, the “related” processes need to communicate and cooperate. Achieving this cooperation is a difficult task for the programmer.

Process Termination

Any computer system must provide a means for a process to indicate its completion. A batch job should include a Halt instruction or an explicit OS service call for termination. In the former case, the Halt instruction will generate an interrupt to alert the OS that a process has been completed. For an interactive application, the action of the user will indicate when the process is completed. When the parent process terminates, the OS may automatically terminate all its children. For example, in a time-sharing system, the process for a particular user is to be terminated when the user logs off or turns off his or her terminal. On a personal computer or workstation, a user may quit an application (e.g., word processing or spreadsheet). All of these actions ultimately result in a service request to the OS to terminate the requesting process.

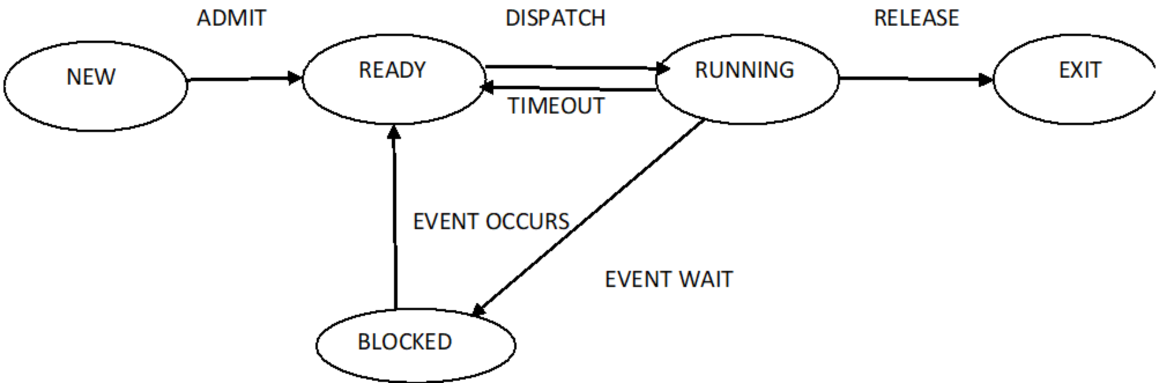
Five State Process Model

A process is a program in execution. In the five-state process model, there are five states of a process. A process may be in any one of the states during its lifetime. First of all, a process arrives into the system for its execution, then it becomes ready for execution and, then it gets the attention of the processor, and at last, the process is terminated after the completion of its execution. When the parent process terminates, the operating system may automatically terminate all its children.

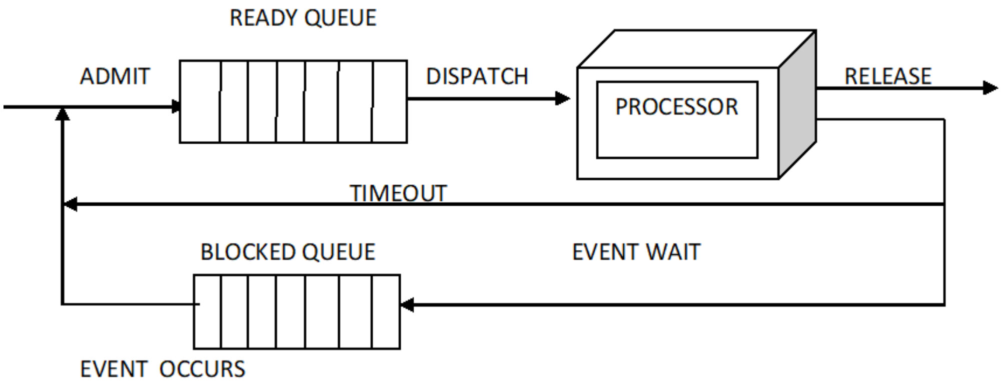
The queue is a first-in-first-out list and the processor operates in a **round-robin** fashion on the available processes (each process in the queue is given a certain amount of time, in turn, to execute and then returned to the queue, unless blocked). Some processes in the Not Running state are ready to execute, while others are blocked, waiting for an I/O operation to complete.

Thus, using a single queue, the dispatcher could not just select the process at the oldest end of the queue. Rather, the dispatcher would have to scan the list looking for the process that is not blocked and that has been in the queue the longest.

A more natural way to handle this situation is to split the Not Running state into two states: Ready and Blocked. We have added two additional states that will prove useful. The five states in this **new** diagram are as follows:



(i) **Five-State Process Model** State Transition Diagram



(ii) **Five-State Process Model** Queuing Diagram

So, the process either be in any of one of the below-mentioned states -:

1. **NEW**: The process is said to be in the NEW state when it arrives into the system for its execution. In other words, the process enters the JOB POOL of the system. A process that has been newly created but has not admitted yet into the pool of executable processes by the OS. Typically, a new process has not yet been loaded into the main memory, although its process control block has been created.

2. **READY**: The process is supposed to be in the READY state when it is ready to get the attention of the CPU In other words, the process enters the READY QUEUE of the system. A READY process is that, which is prepared to execute when given the opportunity.

3. **RUNNING**: A process is supposed to be running, when it gets the attention of the CPU for its execution. In other words, the process is being executed by the CPU in this stage.

4. **WAITING**: A process is said to be waiting when the process is blocked for some time due to some reason. In other words, the process switches (jumps) from the RUNNING stage to the BLOCKED stage for some time. A process that cannot execute until some event happens, the event may be the completion of an Input or Output operation. *Waiting* is a frequently used alternative term for *Blocked* as a process state. There are various reasons for the blocking of a process like I/O Required, Time Slice Elapses, Higher Priority Job Arrives, etc.

5. **TERMINATED**: A process is supposed to be in the TERMINATED state when the process completes its execution successfully. A process which released from the pool of executable processes by the Operating System, either because it paused or because it was abandoned for some reason.

There are two queues now: ready queue and blocked queue

When the process is admitted in the system, it is placed in the ready queue and when a process is removed from the processor, it is either placed in the ready queue or a blocked queue (depending on circumstances). If event time out occurs, then it moves to ready queue, and if event wait occurs then it moves to blocked queue. When an event occurs, all the processes waiting on that event are moved from the blocked queue onto the ready queue.

The New and Exit states are useful constructs for process management. The New state refers to a process when it has been just defined. For example, if a new user starts to log onto a system or a new batch job is submitted to the OS for execution, the OS can define a new process in two stages. First, the OS performs the necessary housekeeping chores. An identifier is associated with the process. Any tables that will be needed to manage the process are allocated and built. At this point, the process is in a New state. This means that the OS has performed the necessary actions to create the process but has not committed itself to the execution of the process. For example, the OS may limit the number of processes that may be in the system for reasons of performance or main memory limitation. While a process is in the new state, information concerning the process that is needed by the OS is maintained in control tables in the main memory. However, the process itself is not in the main memory. That is, the code of the program to be executed is not in the main memory, and no space has been allocated for the data associated with that program.

While the process is in the New state, the program remains in secondary storage, typically disk storage Similarly, a process exits a system in two stages. First, a process is terminated once it reaches its usual completion point, when it aborts due to an unrecoverable error, or when another process with the appropriate authority causes the process to abort. Termination moves the process to the exit state. At this point, the process is no longer eligible for execution. The tables and other information associated with the job are temporarily preserved by the OS, which provides time for auxiliary or support programs to extract any needed information. For example, an accounting program may need to record the processor time and other resources utilized by the process for billing purposes. A utility program may need to extract information about the history of the process for purposes related to performance or utilization analysis. Once these programs have extracted the needed information, the OS no longer needs to maintain any data relating to the process and the process is deleted from the system. Memory committed to existing processes. This limit assures that there are not so many active processes as to degrade performance.

2.4 MULTI-THREADED PROGRAMMING

A thread is a single sequential flow of control within a program. A process is defined sometimes as a ***heavyweight process*** and a thread is defined as a ***lightweight process***. A thread belongs only to one process. It is a unit of computation associated with a particular heavyweight process, using many of the associated process's resources. It has a minimum internal state and a minimum of allocated resources. Threads can share the same resources (files, memory space, etc) which are in that process. Thread can be created faster as compare to the process. Threads are widely used in real-time operating systems and modern operating systems. Each thread has its control block, with a state (Running/Blocked/etc.), saved registers, instruction pointer. Threads improve application performance through parallelism. This concept is useful in a server-client environment. A process can have a single thread or multiple threads. In multiple threads, each thread is associated precisely to one process and it always inside of the process, which means no thread can exist outside a process.

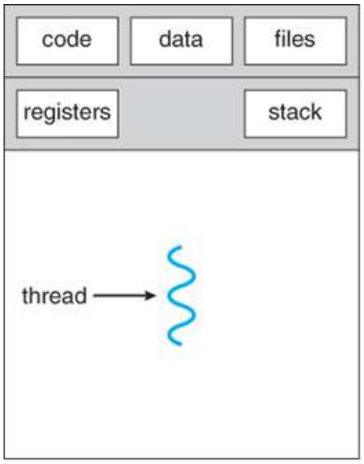


Figure: Representation of Single Thread Process

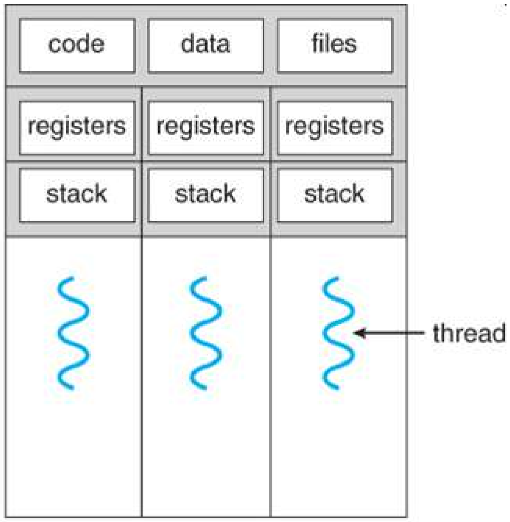


Figure: Representation of Multi-Thread Process

Advantages of Threads

- Thread minimizes context switching time.
- Threads help in the parallel execution of an application on shared-memory multiprocessors.
- The benefits of multi-threading can be greatly increased in a multiprocessor architecture.
- It is more cost-effective to create and context switch threads.
- Threads are dependent on each other so it has efficient communication.
- Threads improve application performance through parallelism.

Threads can be created at the user level and the kernel level so it is implemented by two ways:

(i) User Level (ii) Kernel Level

(i) **User Level**

In the user-level thread, all of the work of thread management is done by the application and the Kernel does not aware of its existence. In user space, the thread library contains code for creating and destroying threads, message passing, data, scheduling of thread execution, etc.

Characteristic of User Level Threads

- It is generally fast to create and easy to manage.
- User-type threads can run on any OS.
- A multithreaded application cannot take advantage of multiprocessing.
- Scheduling can be application-specific.

(ii) **Kernel Level**

In Kernel-level thread, all of the work of thread management is done by the Kernel. Kernel threads are directly supported by the operating system. The kernel maintains context information for the process as a whole and individual threads within the process. In Kernel space, thread library contains code for creating and destroying threads, message passing, data, scheduling of thread execution, etc.

Characteristic of Kernel Level threads

- In the kernel-level thread, scheduling is on a thread basis.
- It is generally slow to create.
- The kernel can schedule multiple threads from the same process on multiple processes.
- Kernel routines themselves can be multithreaded.
- Context switching between threads is time-consuming.

Comparison between process and thread

Process	**Thread**
The process is termed the heavyweight process	Thread is termed lightweight process
Processes are independent of one another	Threads are not independent of one another
In multiple process implementations, each the process executes the same code but has its memory and file resources	All threads can share the same set of open files, child processes.
It takes more time to create a new process.	It takes less time to create a new thread.
It takes more time to terminate a process	It takes less time to terminate a thread
While context switching needs the interface with the operating system.	In the thread, switching does not need to call an OS and cause an interrupt to the kernel.
It takes more time to switch between two processes	It takes less time to switch between two threads

Difference between User-level thread and Kernel level thread

User Level Thread	**Kernel level thread**
It is maintained at the user level.	It is maintained at the Kernel level.
User-level threads are quicker to create	Kernel level threads are not quickly created
It is easy and speedily managed	It is slowly managed.
It runs on any Operating System	These are specific to Operating System.
Implemented by a thread library at the user level	OS support directly to Kernel threads.

Daemon

A Daemon is a system process. It is created at boot time and keeps executing in the background. A daemon is created to perform specific tasks. It gets activated automatically when a request is received for performing a particular task. After completion of the task, it again goes back to the background.

2.5 MULTI-THREADED MODELS

Some operating system allows the facility of both level threads user-level thread and Kernel level thread. An example of this combined approach is Solaris. Multithreading permits the accomplishment of multiple parts of a program run in parallel. These parts of a program are called threads. It was also known as the lightweight process. By using multitasking and multithreading, CPU utilization increased.

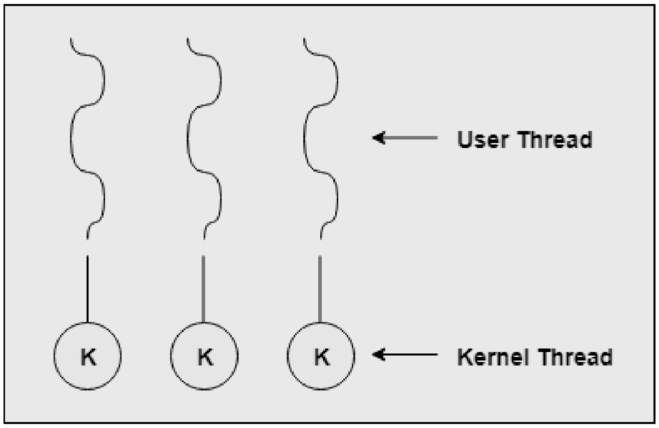
Multithreading models are 3 models namely:

one-one model, many to one model, and many to many models which are described below:

One to One Model

There is 1 to1 association between both the thread's user-level thread and kernel-level thread. This model offers more concurrency than the M:1 model. It also permits another thread to complete its work when a thread makes a blocking system call. A major disadvantage of this is when a user-level thread is created then the requirement of corresponding kernel thread. Therefore a lot of kernel-level threads are required which is a burden on the system, but there is a limit on the number of threads in the system.

A diagram that reveals the one-to-one model is given below –



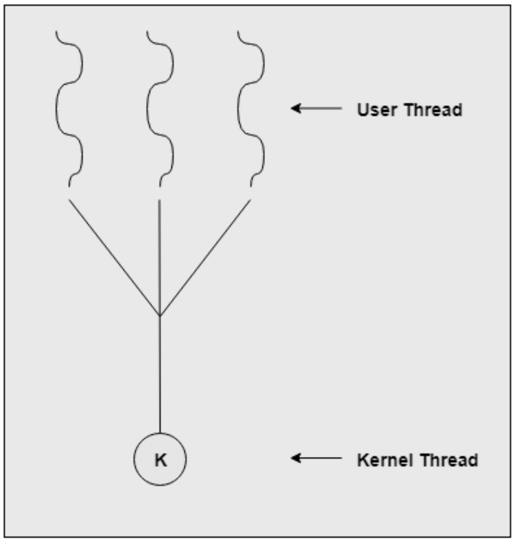
One to One Model

Many to One Model

The many to one model, in this model many of the user threads maps to a single kernel thread. This model is relatively efficient as compare to the 1:1 model as in this the user space was managed by the thread management.

A disadvantage of this model is that when a thread blocking system calls then it blocks the whole process. Another disadvantage is that multiple threads cannot run simultaneously because one thread can access the kernel at a single time.

A given below diagram that shows the many to one model:-



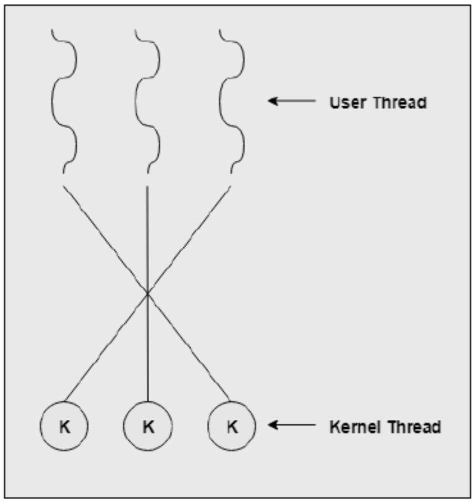
Many to One Model

Many to Many Model

In the many-to-many model, any number of user threads correspond with any number of kernel-level threads. There is no limit. So there is no disadvantage as compared to other models. These threads can execute simultaneously on a multiprocessor.

2.6 CPU SCHEDULING

Scheduling refers to the set of policies and mechanisms that an OS supports for determining the order of execution of pending jobs and processes. A scheduler is an OS Module that determines the next pending job to be admitted into the system for execution or the next ready process to be dispatched to RUN state.



Many to Many Model

Introduction

In the multiprogramming OS, the method or procedure for switching the CPU among multiple processes is called **CPU scheduling**. The CPU scheduler is a part of an OS, which is responsible for CPU scheduling. When a process performs an I/O-related task, it does not use certain resources, such as CPU and these resources remain idle.

CPU scheduling enables processes to utilize idle resources by assigning them to other processes. Whenever CPU becomes idle the CPU scheduler chooses a process among the processes which are in the ready queue and sends the process to the CPU for execution.

For example, process A is running and needs to perform an I/O-related task. Process A does not need the CPU while performing the I/O-related task. Process scheduler changes the state of process A from running to waiting and enables process B to use the CPU.

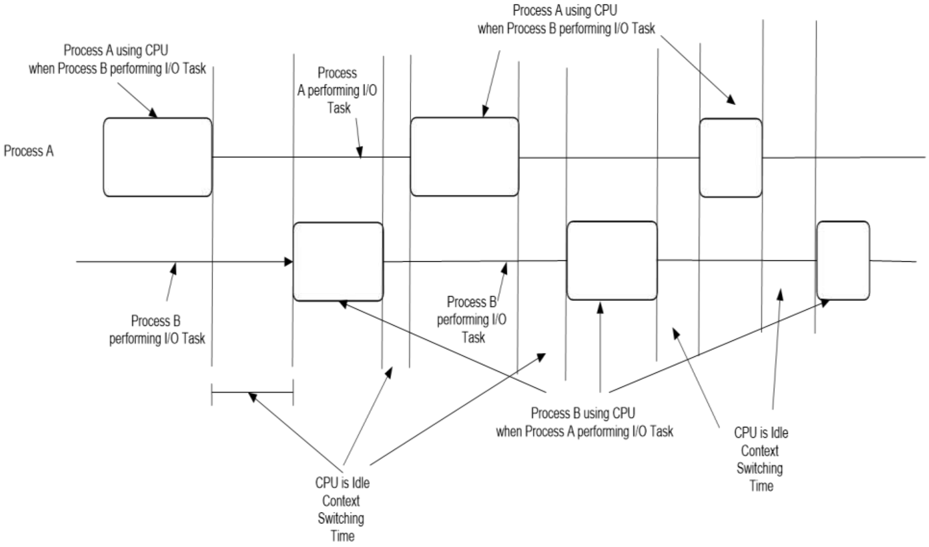

Process A using CPU when Process B performing I/O Task
Process A performing I/O Task
Process A using CPU when Process B performing I/O Task
Process A
Process B performing I/O Task
Process B performing I/O Task
Process B using CPU when Process A performing I/O Task
CPU is Idle Context Switching Time
CPU is Idle Context Switching Time

Figure: CPU Scheduling

For example, in the process of CPU scheduling, two processes, A and B, share the CPU times. When process A uses the CPU, process B performing I/O operations, and when the CPU is accessed by process B, Process A performs I/O operations.

Goals of Scheduling

(i) To optimize the utilization of system resources.
(ii) To ensure that more critical processes get priority over others processes.
(iii) To provide as fair a deal as possible to all jobs and processes which are pending.

When a process, in the ready state, is allocated the CPU in the place of a partially done process, the value of the various field of the partially done process, such as process state and I/O status is updated in the Process Control Block of the partially done process. The OS marks the partially done process as ready and sends it back to the ready queue. The OS reads the data from the process control block of the ready process and allocates CPU to it and marks the process state running. This is called **context switching**.

In other words, “Context switching” refer to the process of transferring control of the CPU from the currently running process to another process from the ready queue.

For example, a time-out has occurred for process A. Process A will be released. Process B is to be allocated the CPU. The steps performed by the **dispatcher** in context switching are:

- It retrieves and analyzes information about process A regarding program counter, memory, and registers.
- It updates the PCB of process A by writing the new values of the various fields of PCB. This includes changing the state of process A from running to ready or waiting.
- It moves process A to the appropriate queue. If the state of process A is changed into a waiting state, it is inserted in the waiting queue. If process A is transformed into the ready state, it is inserted into the ready queue.
- It retrieves and analyzes information about process B from the PCB of process B regarding program counter, memory, and registers.
- It allocates the CPU time to process B
- It restores the environment values, such as program counter, memory, and registers of process B.

The context switching is pure overhead. The extent of this overhead depends on the size of the process content. The larger the process content, the higher will be the context-switching overhead.

2.7 CPU SCHEDULING CRITERIA

Factors for measuring the performance of an operating system

There are certain factors on which the performance of an operating system depends. The efficiency and overall performance of the operating system can be measured in terms of the following factors:

(a) **CPU utilization**: CPU utilization refers to the usage of the processor during the execution of a process.CPU utilization may vary from 40% to 90%.CPU should remain as busy as possible. So, CPU utilization should be the maximum for the better efficiency of a system.

(b) **Throughput**: Throughput may be defined as the total number of processes completed per unit of time. It is the measure of work done by the CPU. It is expressed in terms of the number of jobs done in a given unit of time. It is important to know that the value of throughput does not depend only on the capability of the system but also on the nature of jobs, so, it may vary accordingly. If the jobs are CPU bound then the throughput will be less and if the jobs are I/O bound, the throughput will be high.

If the jobs are long and heavy, the throughput may be one or two processes per hour. On the other hand, if the jobs are short and light, the throughput may be 100 processes per hour. However, it should be as maximum as possible.

(c) **Turnaround time**: Turnaround time is the time which is the difference between the time of submission or entered and the time of ending of the job. It is a metric for batch systems. Turnaround time is the sum of the following components:

□ The time spends in waiting for entry into the system.

□ Total time spent in the ready queue.

□ Total time spent in the device queue.

□ Time spent in the execution of the process.

□ Time spend in doing the Input/Output operation.

However, turnaround time should be as minimum as possible.

(d) **Response time:** Response time is defined as the difference between the time of submission of the job for processing and the time when it gets the first response of the system. It is considered the best metric for interactive systems.

(e) **Waiting time:** Waiting time may be defined as the sum of intervals for which a process has to wait in the ready queue. It should be minimum for the better efficiency of the system. The scheduling strategies try to minimize the waiting time for the processes.

Average Turnaround Time: The turnaround time of a process is the total time elapsed from the time the process is submitted to the time the process is completed. It is calculated as

Turnaround time (TAT) = Process finish time (T_1) – Process Arrwal time (T_0)

Average Turnaround Time =

The lower the average turnaround time, the better it is.

Average Waiting Time: Waiting time of a process is defined as the total time spent by the process while waiting in a ready state or suspended state.

Waiting time (WT) = Turnaround time (TAT) – Actual Execution time (□t)

Average Waiting Time =

The lower the average waiting time, the better it is.

For the better performance and efficiency of a system the CPU utilization should be ***maximum***, Response time should be ***minimum***, Waiting time should be ***minimum***, Throughput should be ***maximum*** and the Turnaround time should be ***minimum***.

DISPATCHER

The dispatcher is a module of an OS that gives control of the CPU to the process selected by the scheduler. The dispatcher is responsible for the context switching. The time taken by the dispatcher for halting the running process and starting the process selected by the scheduler for executions is called **dispatch latency**.

SCHEDULING STRATEGIES

Scheduling strategies refer to the various algorithms used to choose a particular process from among the various processes in the ready queue.

There may be several processes waiting for the attention of the CPU in the ready queue. The operating system chooses a particular process from the ready queue and allocates the processor to it. So, the operating system requires a mechanism to decide, which process should be chosen next for execution.

There are two types of jobs:

- **CPU Bound Jobs:** The jobs, that spend more time performing CPU operations and less I/O operations are called CPU-bound jobs/processes.
- **I/O Bound Jobs:** The jobs that spend more time performing I/O operations and less time doing CPU operations are called I/O bound jobs/processes.

CPU Scheduling can be **preemptive** or **non-preemptive**.

In **preemptive scheduling**, the scheduler removes the running process from the CPU before its completion so that another process can run. In other words, in preemptive scheduling, the running process only gives up the control of the CPU voluntarily.

In **non-preemptive scheduling,** nothing can remove a process from utilizing CPU time until it completes or a time-out occurs.

There are several strategies used for *CPU Scheduling,* called *Scheduling Strategies* like:

1. FIRST COMES FIRST SERVE (FCFS)
2. SHORTEST JOB FIRST (SJF) (Preemptive and Non-preemptive)
3. PRIORITY SCHEDULING (Preemptive and Non-preemptive)
4. ROUND ROBIN SCHEDULING (RR SCHEDULING)

The detailed description of the scheduling strategies is as follows:

First Come First Serve (FCFS)

It is the simplest of all the scheduling algorithms. It is purely non-preemptive scheduling. The key concept of this algorithm is:

"allocate the CPU to the processes in the order in which they arrive".

According to this algorithm, the process which arrives first will get the CPU before any other process. Same way, the process which arrives as a second, will get the CPU after the first process and so on. It assumes the Ready Queue as the FIFO QUEUE. When a process completes its execution, the CPU is allocated to that process which is the first process in queue i.e the FRONT of the queue. If a new process arrived then it enters in REAR of the queue.

When a process starts running then it is removed from the queue. This algorithm is NON-PREEMPTIVE by default. This means once the CPU is assigned to a process, that process keeps the CPU till the end of its execution.

Advantages:

The code for FCFS scheduling is simple to write and understand.

It is suitable for Batch systems.

It is considered to be a fair policy, as the job which arrives first will get the CPU first.

Disadvantages:

If shorter jobs arrive after the longer jobs, then the waiting time will be large.

It is not suitable for Time-sharing systems.

Low CPU utilization, because all the other processes wait for one long job to get off the CPU**.**

This algorithm is never recommended whenever performance is a major issue.

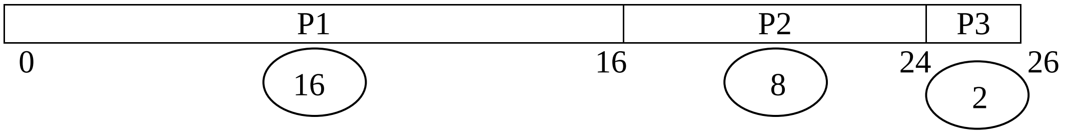
Example:

Consider the following snapshot of processes that arrive at a different time with the CPU-burst in milliseconds.

Note: Ignoring the arrival time in non-preemptive scheduling, the lowest number has the highest priority and time slice 2ms.

Process ID	Arrival Time (T_0) ms	Priority	Next CPU Burst Time (Δt) ms
P1	0	2	16
P2	1	3	8
P3	2	1	2

The Gantt chart of execution of the processes according to FCFS is the following:



Waiting time for P1 = 0 ms

Waiting time for P2 = 16 ms

Waiting time for P3 = 24 ms

Table depicting performance of FCFS

Process ID	**Arrival Time (T_0) ms**	**Next CPU Burst Time (Δt) ms**	**Finish Time (T_1) ms**	**Turnaround Time $TAT=T_1-T_0$ (ms)**	**Waiting Time= $TAT-\Delta t$ (ms)**
P1	0	16	16	16	0
P2	0	8	24	24	16
P3	0	2	26	26	24
				66	40

As mentioned in question ignore the arrival time in non-preemptive scheduling so, here arrival time mentioned 0

Total waiting time	= 0+16+24	= 40 milliseconds
Average waiting time	= 40/3	= 13.33 milliseconds.
Total turnaround time	= 16+24+26	= 66 milliseconds
Average turnaround time	= 66/3	= 22 milliseconds.

However, if the jobs arrive in a different order, then the waiting time and turnaround time can be reduced. So, the waiting time and turnaround time depend upon the order of the jobs in which they arrive.

Shortest Job First (SJF)

The key concept of this algorithm is

“allocate the processor to the job which has the least CPU burst time”.

The ready queue has all the processes which require the processor for their execution. According to this algorithm, the processor is allocated to that job that has the smallest CPU burst time amongst all the processes in the ready queue.

If two processes have the same CPU burst, then the processor is allocated to the process which arrives first. This algorithm can either be PREEMPTIVE or NON-PREEMPTIVE.

Advantages

- This is considered to be an optimal algorithm as it helps to achieve the minimum waiting time.
- The shorter jobs have to wait for less time as compared to the longer jobs.

Disadvantages

- There is a need for the mechanism to know about the CPU burst of all the processes in advance.
- If the shorter jobs arrive again and again, the longer jobs may wait for a long period.

Example

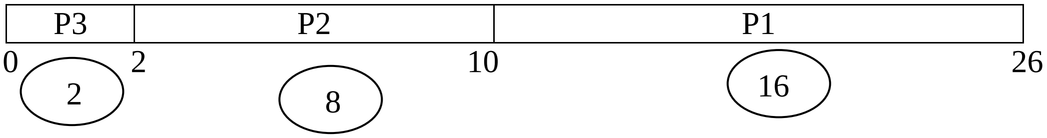
NON-PREEMPTIVE SJF

Consider the following snapshot of processes that arrive at a different time with the CPU-burst in milliseconds.

Note: Ignoring the arrival time in non-preemptive scheduling, the lowest number has the highest priority and time slice 2ms.

Process ID	**Arrival Time (T_0) ms**	**Priority**	**Next CPU Burst Time (Δt) ms**
P1	0	2	16
P2	1	3	8
P3	2	1	2

The Gantt chart of execution of the processes according to NON-PREEMPTIVE SJF is the following:



Waiting time for P1 = 10 ms
Waiting time for P2 = 2 ms
Waiting time for P3 = 0 ms

Table depicting performance of non-preemptive SJF

Process ID	**Arrival Time (T_0) ms**	**Next CPU Burst Time (Δt) ms**	**Finish Time (T_1) ms**	**Turnaround Time** **TAT=T_1-T_0 (ms)**	**Waiting Time= TAT-Δt (ms)**
P1	0	16	26	26	10
P2	0	8	10	10	2
P3	0	2	2	2	0
				38	12

Total waiting time	= 10+2+0	= 12 milliseconds
Average waiting time	= 12/3	=4 milliseconds.
Total turnaround time	= 26+10+2	= 38 milliseconds
Average turnaround time	=38/3	= 12.6 milliseconds.

PREEMPTIVE SJF: It is also termed as Shortest Remaining Time Next (SRTN) and Shortest Remaining Time First(SRTF).

Consider the set of the following processes with the CPU-burst in milliseconds.

Process ID	Arrival Time (T_0) ms	Priority	Next CPU Burst Time (Δt) ms
P1	0	2	16
P2	1	3	8
P3	2	1	2

Here, first of all, P1 arrives and the processor is allocated to it. After one second P2 arrives, since P2 has a smaller CPU burst as compare to P1, therefore the processor is preempted from P1 and is allocated to P2. Same way, after one second, P3 arrives whose CPU burst is less than the P2, so the processor is preempted from P2 and allocated to the P3. After the execution of P3, the processor is again allocated to P2 and so on.

The Gantt chart of execution of the processes according to PRE EMPTIVE SJF is the following:

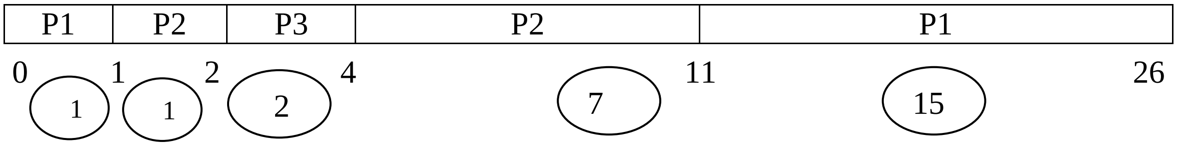


Table depicting performance of preemptive SJF

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Finish Time (T_1) ms	Turnaround Time $TAT=T_1-T_0$ (ms)	Waiting Time= $TAT-\Delta t$ (ms)
P1	0	16	26	26	10
P2	1	8	11	10	2
P3	2	2	4	2	0
				38	12

Total waiting time	**= 10 + 2 + 0**	**= 12 milliseconds**
Average waiting time	=12/3	= 4 milliseconds.
Total turnaround time	= 26 + 10 + 2	= 38 milliseconds
Average turnaround time	=38/3	= 12.6 milliseconds.

In the case where it is not possible to know the CPU time for each process, this is estimated using predictors:

– $P_n = aO_{n-1} + (1-a)P_{n-1}$ where
  - O_{n-1} = previous service time
  - P_{n-1} = previous predictor
  - a is within [0,1] range

– If a = 1 then P_{n-1} is ignored

– P_n is dependent upon the *history* of the process evolution

Priority Scheduling

Priority scheduling is the scheduling mechanism in which each process in the system is assigned a priority. The processor is allocated to the processes according to their priority. The key concept of this algorithm is

“***Allocate the processor to the process which has higher priority”***.

The ready queue is assumed to be a priority queue in which each process is assigned a priority. First of all, the processor is allocated to the process having higher priority and then to the process having lower priority and so on. Priority scheduling can be of two types:

PREEMPTIVE or NON-PREEMPTIVE

Example:

NON-PREEMPTIVE PRIORITY SCHEDULING

Consider the set of the following processes with the CPU-burst in milliseconds.

Note: Ignoring the arrival time in non-preemptive scheduling, the lowest number has the highest priority and time slice 2ms.

Process ID	**Arrival Time (T_0) ms**	**Priority**	**Next CPU Burst Time (Δt) ms**
P1	0	2	16
P2	1	3	8
P3	2	1	2

The Gantt chart for the execution of the programs is as follows:

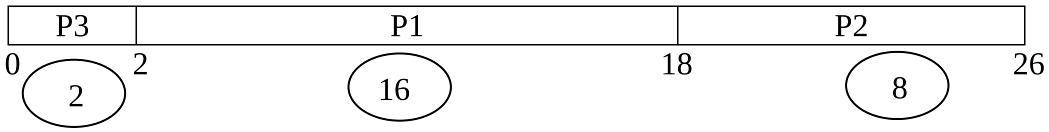


Table depicting performance of non-preemptive priority scheduling

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Finish Time (T_1) ms	Turnaround Time TAT=T_1-T_0 (ms)	Waiting Time= TAT-Δt (ms)
P1	0	16	18	18	2
P2	0	8	26	26	18
P3	0	2	2	2	0
				46	20

Waiting time for P1 = 2 ms
Waiting time for P2 = 18 ms
Waiting time for P3 = 0 ms

Total waiting time	= 2 +18 + 0	= 20 milliseconds
Average waiting time	=20/3	= 6.6 milliseconds.
Total turnaround time	= 18 + 26 + 2	= 46 milliseconds
Average turnaround time	=46/3	= 15.3 milliseconds.

PREEMPTIVE PRIORITY SCHEDULING

Consider the set of the following processes with the CPU-burst in milliseconds.

Process ID	Arrival Time (T_0) ms	Priority	Next CPU Burst Time (Δt) ms
P1	0	2	16
P2	1	3	8
P3	2	1	2

The Gantt chart for the execution of the programs is as follows:

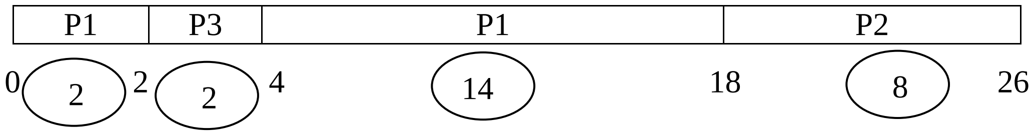


Table depicting performance of preemptive priority scheduling

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Finish Time (T_1) ms	Turnaround Time $TAT=T_1-T_0$ (ms)	Waiting Time= $TAT-\Delta t$ (ms)
P1	0	16	18	18	2
P2	1	8	26	25	17
P3	2	2	4	2	0
				46	19

Total waiting time = 2 + 17 + 0 = 19 milliseconds
Average waiting time =19/3 = 6.3 milliseconds.
Total turnaround time = 18 + 25 + 2 = 45 milliseconds
Average turnaround time =45/3 = 15 milliseconds.

The problem in Preemptive Priority Scheduling (Starvation or Blocking)

The main problem with priority scheduling is that low-priority processes can wait for a long time due to high-priority process arrivals. If an Operating system has not any precautions and just chooses the process with the highest priority, low priority processes wouldn’t get CPU, as long as high priority processes are runnable. This problem is known as **Starvation.**

Solution of Starvation (AGING)

Aging is a method of slowly increasing the priority of processes that are waiting in the system for a long time. The simplest solution is dynamic priorities. On one hand, the operating system can reduce the priority of a running process for each time quantum it used the CPU and on the other hand, it could increase the priority of other processes (which finally leads to the first situation because the boost should only be temporary) which didn’t get the CPU for a certain amount of time. Whether the operating system uses one or another solution, processes have a base priority that remains unchanged, and a real priority that is used for scheduling. Often the real priority is limited to a specific range so that important process still gets the processor when they need it.

Another way to avoid this problem is static priorities. The OS has to keep a record of how long a process has used the CPU. If it reaches a certain limit, the next highest priority process is allowed to run. This is not practically good.

Round Robin Algorithm (RR Algo)

The Round Robin scheduling algorithm is designed especially for the Time Sharing systems. It can be considered as FCFS scheduling along with the preemption. The processor is allocated to a process for a fixed amount of time called, **TIME SLOT** or **TIME QUANTUM**, or, **TIME SLICE**. It is a purely preemptive algorithm. Likewise FCFS, the processor is allocated to the processes in the order in which they arrive, but the processor is preempted from the process after the time slice is over, and the processor is allocated to the next process in the ready queue. A time quantum generally varies from 10 milliseconds to 100 milliseconds. The ready queue is assumed to be a circular queue. The short-term scheduler goes on allocating the processor to the processes in the ready queue for a fixed amount of time.

However, if a process has its CPU burst less than the time quantum, then the process releases the CPU voluntarily (itself). Otherwise, the processor is allocated to the process in the ready queue for a fixed amount of time (time quantum) and after that CONTEXT SWITCHING take place, and the processor is allocated to the next process in the ready queue, keeping the former process at the end of the ready queue. The RR algorithm is PREEMPTIVE by default.

The performance of the Round Robin algorithm depends on the value of time quantum or slice

- If the value of time quantum is large, then this algorithm becomes the same as FCFS.
- If the value of the time quantum is small, then the number of the context switching will be increased considerably, this is not desirable at all. It affects the system throughput adversely.
- Thus, the size of the time quantum should neither be very large nor too small for better efficiency.

Example:

Consider the set of the following processes with the CPU-burst in milliseconds.
Note: Ignoring the arrival time in non-preemptive scheduling, the lowest number has the highest priority and time slice 2ms.

Process ID	**Arrival Time (T_0) ms**	**Priority**	**Next CPU Burst Time (Δt) ms**
P1	0	2	16
P2	1	3	8
P3	2	1	2

The Gantt chart of execution of the processes according to the RR scheduling algorithm is the following. Let the time quantum be 2 milliseconds-:

P1	P2	P3	P1	P2	P1	P2	P1	P2	P1

0 2 4 6 8 10 12 14 16 18
26

2 2 2 2 2 2 2 2 2 8

Table depicting performance of Round Robin scheduling

Process ID	**Arrival Time (T_0) ms**	**Next CPU Burst Time (Δt) ms**	**Finish Time (T_1) ms**	**Turnaround Time $TAT=T_1-T_0$ (ms)**	**Waiting Time= $TAT-\Delta t$ (ms)**
P1	0	16	26	26	10
P2	1	8	18	17	9
P3	2	2	6	4	2
				47	21

Waiting time for P1 = 10 ms
Waiting time for P2 = 9 ms
Waiting time for P3 = 2 ms

Total waiting time = 10 + 9 + 2 = 21 milliseconds
Average waiting time = 21/3 = 7 milliseconds.
Total turnaround time = 26 + 17 + 4 = 47 milliseconds
Average turnaround time = =47/3 = 15.6 milliseconds.

Comparison between FCFS and Round Robin Scheduling Algorithm

FCFS	Round Robin
FCFS is purely non-preemptive Scheduling	It is purely preemptive Scheduling
It has minimum overhead	It has a higher overhead as compare to FCFS as context switching occurs more.
Response time depends upon the size of the process	It offers a better response time
It is not designed for a time-sharing system	It is designed for a time-sharing system
The workload is simply processed in the order process arrived	It is similar to FCFS but it uses a time slice which means one process maximum uses CPU for time slice at once.

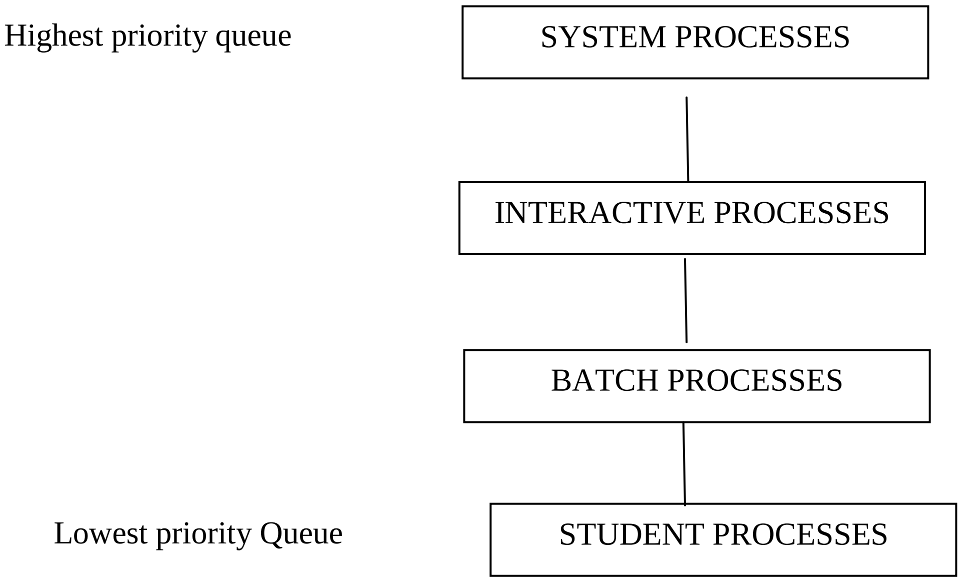
Multilevel Queue Scheduling

A multilevel queue scheduling algorithm is used when the processes are to be divided into different groups. Sometimes such a situation arises where two or more processes have different response time requirements, so there is a need for a different kind of schedule for both processes.

A multilevel queue scheduling division of the ready queue into numerous separate queues. The processes are permanently allocated to a particular queue, based on the properties of the process.

Sometimes there are two or more processes, among which some processes have higher priority and other having low priority. So, there is a need for a separate queue for the processes with higher priority and the processes with lower priority.

Consider an example of multilevel queue scheduling for different kinds of processes in the ready queue. There are four different queues for different processes which are made based on their priority in the system.



Each queue may have its scheduling criteria, based upon the nature and properties of the jobs.

Advantages

- Low scheduling overhead can be achieved by using a multilevel scheduling algorithm.

Disadvantages

- It is not considered as efficient in some of the cases, like if higher priority queues don’t become empty for a long time, then the lower priority jobs may starve.

Multilevel Feedback Queue Scheduling

It is an enhancement of multilevel queue scheduling.

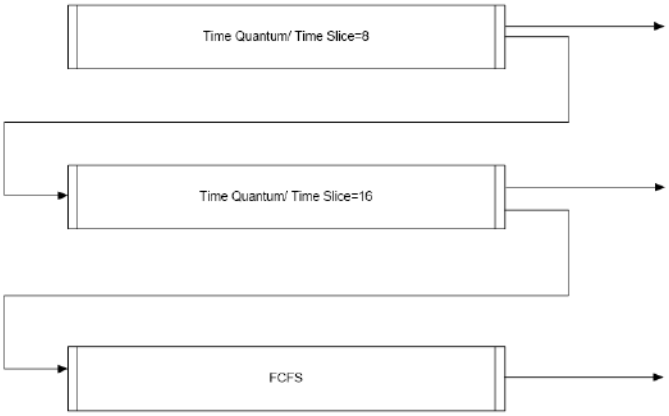
Actually, in multilevel queue scheduling, the processes are assigned a particular queue on entering the system. The processes do not move from a queue to any other queue. So, it is an inflexible approach.

Multilevel feedback queue scheduling permits a process to move between the queues. For example, if a process CPU-bound process which means it uses too much CPU then the process is moved (shifted) to a lower priority queue. Correspondingly, a lower priority job can also be shifted to a higher priority queue if it is waiting for a long time, etc.

Advantages

- It allows a process to move to any other queue.
- It is more flexible as compared to any other scheduling algorithm.

- A lower priority job, which is waiting for a long time, can be shifted to a higher priority queue.
- A higher priority job, affecting negatively the efficiency, can be moved to a lower priority queue.



Multilevel Queue Scheduling

Disadvantages

- However, it is considered to be a complex scheduling algorithm.
- While moving the processes from a queue to any other queue increases the CPU overhead.

Illustration

With reference to the following set of processes

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Priority
P1	0	29	1
P2	6	14	5
P3	8	10	3
P4	10	8	1
P5	13	6	2

Determine Average Waiting Time and Turn Around Time, for the following scheduling.

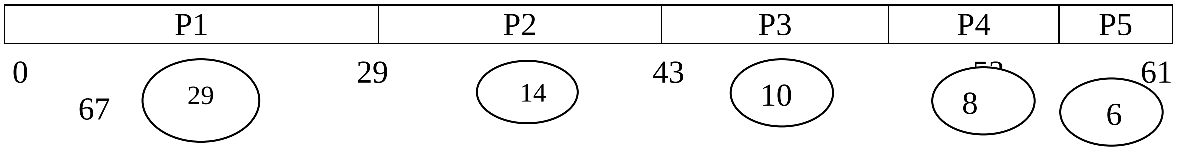
(a) First Come First Serve (FCFS)
(b) Shortest Job First (SJF) (Non-preemptive)
(c) Shortest Job First (Preemptive) or Shortest Remaining Time Next (SRTN)
(d) Priority Scheduling (Preemptive)
(e) Priority Scheduling (Non-preemptive)

(f) Round Robin (RR)

Note: Time Slice – 5 ms, the highest number has the highest priority and ignores the arrival time in non-preemptive scheduling.

(a) **First Come First Serve (FCFS)**

In this scheduling, jobs or processes are scheduled to run in the same order as those that have arrived in the system. It's a pure non-preemptive algorithm. Process P1. will execute first because its ID is 1, then P2, P3, P4, and P5. So its Gantt Chart is given below:



Calculating Waiting Time and Turnaround Time

In this problem, in a non-preemptive algorithm arrival time is ignored so, waiting time for all processes is when they start to run and turnaround time is when they complete the process.

Note Arrival time 0 in this problem because in non-preemptive algorithm arrival time is ignored.

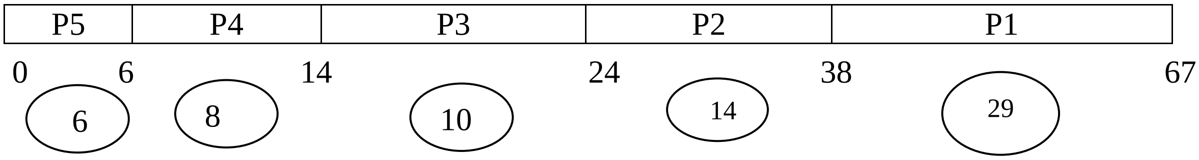
Table depicting performance of FCFS

Process ID	**Arrival Time (T_0) ms**	**Next CPU Burst Time (Δt) ms**	**Finish Time (T_1) ms**	**Turnaround Time TAT=T_1-T_0 (ms)**	**Waiting Time= TAT-Δt (ms)**
P1	0	29	29	29	0
P2	0	14	14	43	29
P3	0	10	10	53	43
P4	0	8	8	61	53
P5	0	6	6	67	61
				253	186

Average Waiting Time = (0+29+43+53+61)/5=37.2 ms
Average Turnaround Time = (29+43+53+61+67)/5=50.6ms

(b) **Shortest Job First (SJF)**

In shortest job first, job to be dispatch will be the one, which happens to be the shortest amongst the pending jobs. This is non-preemptive. So a job, once scheduled, is permitted to complete its next burst. In this problem, Arrival time is ignored for non-preemptive scheduling so, P5 has the shortest next burst time hence it will run first then P4, P3, P2 & P1 will run. So its Gantt chart is



Calculation of Waiting Time & Turnaround Time

In this problem, in a non-preemptive algorithm arrival time is ignored so, waiting time for all processes is when they start to run and turnaround time is when they complete the process.

Table depicting performance of non-preemptive SJF scheduling

Process ID	**Arrival Time (T_0) ms**	**Next CPU Burst Time (Δt) ms**	**Finish Time (T_1) ms**	**Turnaround Time TAT=T_1-T_0 (ms)**	**Waiting Time= TAT-Δt (ms)**
P1	0	29	67	67	38
P2	0	14	38	38	24
P3	0	10	24	24	14
P4	0	8	14	14	6
P5	0	6	6	6	0
				149	82

Note: Arrival time is 0 in this problem because non-preemptive algorithm arrival time is ignored.

Average Waiting Time = (38+24+14+6+0)/5=16.4 ms
Average Turnaround Time = (67+38+24+14+6)/5=29.8 ms

(c) **Shortest Remaining Time Next (SRTN)**

This is a preemptive algorithm, where the next job/process to be dispatched will be one that happens to be shortest amongst the pending jobs, at the time of making the decision. However, if a process/job arrives later, whose next burst time to be less than the remaining burst time of the currently running process, the currently running process will be preempted by the new process. The preempted process will be later re-dispatched when its remaining burst happens to be shortest amongst the pending processes.

In this problem, at 0 arrival time, only one process is there i.e. P1. So, P1 will execute first. After 6ms, a new process P2 arrives which has a 14ms burst time and P1 has 23ms (29-6) remaining burst time. So, P1 will be preempted by the P2 because P2 has the shortest burst time at 6ms. After 2ms i.e. 8ms a new process P3 arrives which has 10ms burst time and P2 has 12ms (14-2) remains burst time P3 has the shortest burst time at 8ms. So, P2 will be preemptive by P3. After 10ms a new process P4 arrives which has an 8ms burst time and P3 has 8ms (10-2) remains burst time. So there is a tie between P3 and P4. To break tie FCFS algorithm is adopted so, P3 will continue to execute. At time 13ms a new process, P5 arrived with 6ms burst time. At that time P3 has (10-(13-8)) i.e. 5ms remaining burst time which shortest than among all-time so P3 will continue. After completion of P3 at 18ms time. There is no new process arrived. Now remains burst time for a process are P1 is 23ms, P2 is 12ms, P4 is 8ms and P5 is 6ms. So amongst P5 is shortest than P4, P2 and P1 will execute. Gantt chart of this is given below:

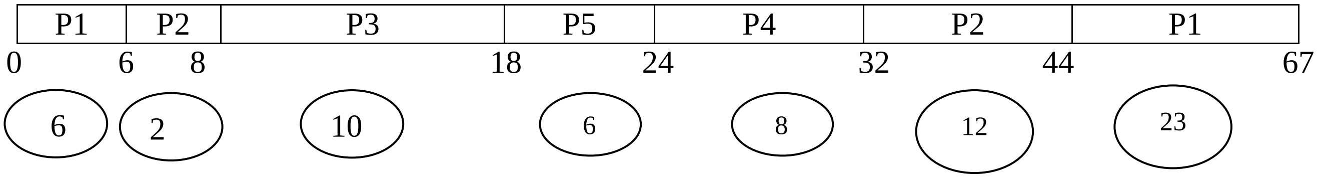


Table Depicting Performance of SRTN Algorithm

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Finish Time (T_1) ms	Turnaround Time TAT=T_1-T_0 (ms)	Waiting Time= TAT-Δt (ms)
P1	0	29	67	67	38
P2	6	14	44	38	24
P3	8	10	18	10	0
P4	10	8	32	22	14
P5	13	6	24	11	5
				148	81

Average Waiting Time =(38+24+0+14+5)/5=16.2 ms
Average Turnaround Time = (67+38+10+22+11)/5= 29.6 ms

(d) **Priority Scheduling (Preemptive)**

At the time of schedules, a dispatcher dispatched that process which has the highest priority amongst the processes which are waiting in the ready queue. When a process Pi is executing, if another process Pj arrived and has higher priority, then Pi will be preempted by Pj.

In this problem, at time 0ms only one process P1 is there so it is executed till a new process has arrived i.e. 6ms. After 6ms a new process P2 comes which has priority 5 which is greater than P1 priority i.e. 2 so, P1 will be preempted by P2. Now P2 will execute at 8ms a new process P3 arrive which have priority 3 but P2 have the highest priority at that time so, P2 will continue, after 10ms P4 arrive which have priority value 1 which lowest so, P2 will continue. After that at 13ms P5 comes whose priority is lower than P2. So P2 will continue till 20ms i.e. (6 + 14).

Now there are 4 processes at 20ms P1, P3, P4, P5 are ready and priorities are 1, 3, 1, 2 respectively. Among them, 3 is the highest priority. So, P3 will execute then P5. After that P1, P4 have the same priority so, there is a tie. To break the tie FCFS algorithm was adopted so P1 executes than P4. Gantt chart of this algorithm is:

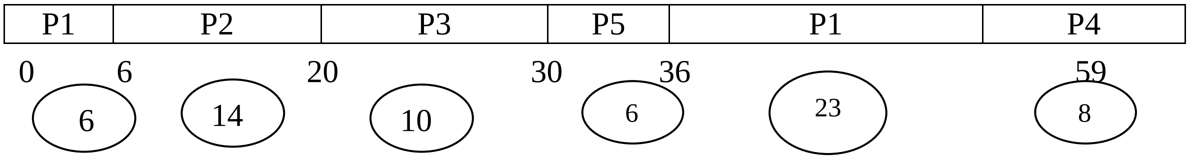


Table depicting performance of a priority-based preemptive algorithm.

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Finish Time (T_1) ms	Turnaround Time TAT=T_1-T_0 (ms)	Waiting Time= TAT-Δt (ms)
P1	0	29	59	59	30
P2	6	14	20	14	0
P3	8	10	30	22	12
P4	10	8	67	57	49
P5	13	6	36	23	17
				175	108

Average Waiting Time = (30+0+12+49+17)/5=21.6 ms
Average Turnaround Time = (59+14+22+57+23)/5=35 ms

(e) **Priority Scheduling (Non-preemptive)**

At the time of scheduling, a process that has the highest priority amongst all the processes which are waiting in the ready queue. Once dispatched, a process Pi is allowed to complete its burst time, even if Pj another process of having higher priority becomes ready while running Pi.

In this scheduling, arrival time is ignored so amongst all processes P2 has the highest priority which will execute first. Priority of above processes is:

P2 > P3 > P5 > P1 > P4

Gantt chart is:

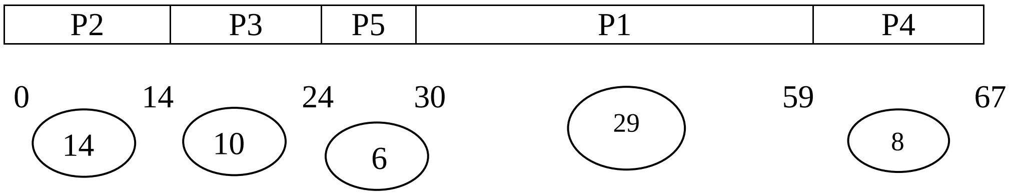


Table depicting performance of a priority-based non-preemptive algorithm.

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Finish Time (T_1) ms	Turnaround Time TAT=T_1-T_0 (ms)	Waiting Time= TAT-Δt (ms)
P1	0	29	59	59	30
P2	0	14	14	14	0
P3	0	10	24	24	14
P4	0	8	67	67	59
P5	0	6	30	30	24
				194	127

Note: Arrival time is 0 in this problem because non-preemptive algorithm arrival time is ignored.

Average Waiting Time = (30+0+14+59+24)/5=25.4 ms

Average Turnaround Time = (59+14+24+67+30)/5=38.8 ms

(f) **Round Robin (RR)**

It is purely a preemptive scheduling algorithm. A small unit of time is called a time slice or time quantum. It is the maximum time for which a process can execute at a time. New processes are added at the end of the ready queue and the head of the ready queue process Pi is dispatched for the time slice. If the process Pi has not finished its execution, then it is linked to the tail of the ready queue, and the next process in the ready queue is dispatched for the next time slice and so on.

In this problem, the time slice is 5ms so P1 will execute for 5ms but there is no more process at 5ms so, P1 again executes for the next 5ms. After 10ms P2 execute for 5ms and P3, P4, P5 so on. This process will continue till the process complete their execution.

Gantt Chart is:

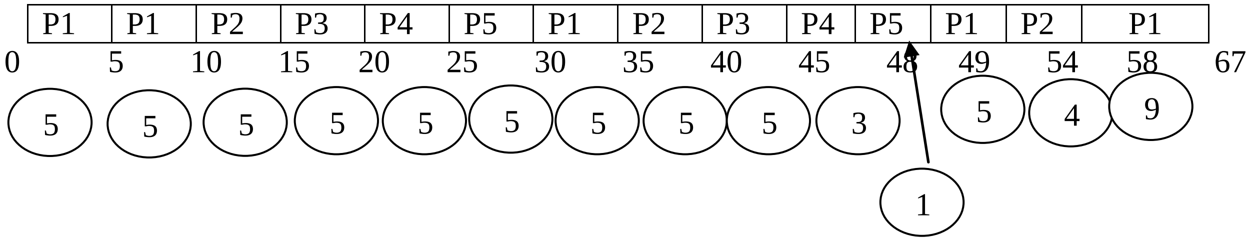


Table depicting performance of round robin algorithm

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Finish Time (T_1) ms	Turnaround Time TAT=T_1-T_0 (ms)	Waiting Time= TAT-Δt (ms)
P1	0	29	67	67	38
P2	6	14	58	52	38
P3	8	10	45	37	27
P4	10	8	48	38	30
P5	13	6	49	36	30
				230	163

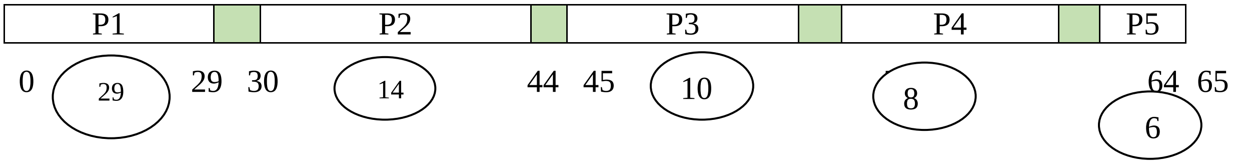
Average Waiting Time = (38+38+27+30+30)/5 =32.6 ms
Average Turnaround Time = (67+52+37+38+36)/5=46 ms

CPU Scheduling Algorithms analysis

Scheduling Algorithm	Turnaround Time	Waiting Time	Average Turnaround Time	Average Waiting Time
FCFS	253	186	50.6	37.2
SJF	149	82	29.8	16.4
SRTN	148	81	29.6	16.2
Priority(Preemptive)	175	108	35	21.6
Priority(non-Preemptive)	194	127	38.8	25.4
Round Robin	230	163	46	32.6

The above table depicts that SRTN scheduling gives better performance among the other scheduling.

Note: Some time context switching time is given in problem then that time is switching time and it is pure overhead. It will be increased the waiting time and turnaround time. If the context switch time is 1ms then in the above example the Gantt chart of FCFS as below:



Performance of all CPU Scheduling Algorithms

Scheduling Algorithm	CPU Utilization	Response Time	Average Waiting Time	Average Turnaround Time	Throughput
FCFS	Low	Low	High	High	Low
SJF	Medium	Medium	Medium	Medium	High
Priority	Medium	High	High	High	Low
Round Robin	High	High	Medium	Medium	Medium
Multi-Level	High	Medium	Medium	Medium	High

Points to Remember

- A program is a static entity.
- A process is an instance of a program under execution.
- The process is a dynamic entity.
- Process state is defined as the current activity of the process.
- In two states process model process has two states: Running or not running.
- In five states process model process has five states: New, Ready, Waiting Running, and Terminating.
- Each process has its Process Control Board (PCB).
- In-Process Control Board (PCB) attributes and information about the process is stored which is needed by OS to control the process.
- Schedulers are of three types: Long term Scheduler, Medium Term Scheduler, Short Term Scheduler.
- Long term scheduler is also known as a job scheduler. It selects the processes and loads them into memory. It changes process states from new to ready state.
- A short-term scheduler is also known as a CPU scheduler or dispatcher. It changes process states from ready to running state.
- The process is called the heavyweight process.
- A process can have a single thread as well as multiple threads.
- Thread is called a lightweight process.
- Threads are not independent of one another.
- Threads can be created at a user level and the kernel level.
- A context switch is switching the CPU to another process which requires saving the state of the old process and loaded the saved state for the new process.
- A Daemon is a system process that is created at boot time and keeps executing in the background.
- CPU scheduling is used to increase CPU utilization.
- In non-preemptive scheduling, once the process has been assigned to the CPU, the CPU cannot be taken away from that process until it terminates or is blocked.
- In preemptive scheduling, the CPU can be taken away from the process during execution.
- Context switching is required in preemptive scheduling.

- Throughput means how many processes the system can execute in a unit of time. Higher the number, the better it is.
- Response Time is the time from the entry of request until the first response is produced.
- Finish time means when the process finishes its execution (T1).
- Arrival time is the time when the process arrived (T0).
- Burst Time is the estimated time the process needed CPU for execution (**Δt**).
- Turnaround time is computed by subtracting the time the process entered the system from the time it terminated (TAT= T1-T0).
- Waiting Time is time spent by the process for CPU (WT=TAT- **Δt**).
- CPU-bound process spends most of its time in CPU.
- I/O bound process spends most of its time in I/O operations.
- First Come First Serve (FCFS) is a purely non-preemptive algorithm.
- In FCFS, the CPU is allocated to the process in order of arrival.
- Shortest Job First (SJF) scheduling algorithm can be either a non-preemptive or preemptive algorithm.
- In SJF, the CPU is allocated to the process which has the smallest burst time.
- Preemptive SJF is known as Shortest Remaining Time Next (SRTN).
- In Shortest Job First (SJF) scheduling (non-preemptive and preemptive) the next CPU burst time must be known in advance.
- A priority scheduling algorithm can be either a non-preemptive or preemptive algorithm.
- In priority scheduling, the CPU is allocated to the process which has the highest priority.
- In preemptive priority scheduling, there is a big problem known as **starvation.**
- In starvation, blocking of low priority processes due to high priority jobs keep arriving one after another.
- **Aging** is a solution to starvation, after some time makes the priority of a process go up the longer it stays run-able but isn't run.
- Round Robin is a purely preemptive algorithm.
- CPU is allocated to all processes in a queue for small-time, which is known as Time Slice, Time Quantum.
- Multilevel queue algorithms allow different algorithms to be used for various classes of processes.
- Multilevel feedback queues allow processes to move from one queue to another.

EXERCISES

1. What is a Process? Explain different states of a process.
2. Explain the Process State diagram in detail.
3. What do you mean by Process Creation and Termination?
4. What is Process Control Block?
5. Explain the following terms:

   (i) Throughput
   (ii) Waiting Time

(iii) Turn Around Time
(iv) Response Time
(v) CPU utilization

6. What are the various factors for measuring the performance of an operating system?
7. Explain the Operations on Processes.
8. What do you mean by Thread and explain different types of threads?
9. Difference between Thread and Process.
10. Difference between User Level Thread and Kernel Level Thread.
11. Explain how threads can improve system performance.
12. Discuss the following terms:

□ Process Spawning
□ Parent Process
□ Child Process
□ Halt
□ Dispatcher
□ Ready Queue
□ Blocked Queue
□ Daemon

13. What do you mean by CPU scheduling and why it is required?
14. What do you mean by I/O bound and CPU bound process?
15. Define the differences between preemptive and non-preemptive scheduling?
16. What are the scheduling criteria? Explain it.
17. Explain the different types of schedulers.
18. Consider the following set of processes

Process	**Arrival Time**	**Priority**	**Burst Time**
P0	0	4	20
P1	1	2	8
P2	3	1	10
P3	5	3	4
P4	8	1	2
P5	9	5	7

Calculate Average Turnaround Time, Average waiting time for the following algorithms. The time slice is 3 ms and the lowest number has the highest priority for the following scheduling.

(a) FCFS (b) SJF (Preemptive and non-preemptive)
(c) Priority (Preemptive and non-preemptive) (d) Round Robin

19. What is Multilevel queue scheduling. Why we use it?
20. Explain between long-term and short-term scheduler.
21. Explain with example the following:

(a) FCFS (b) SJF

22. Explain with example the following:

(a) Priority Scheduling (b) Round Robin Scheduling

23. Difference between FCFS and RR scheduling.

24. Consider the following set of processes

Process	**Arrival Time**	**Priority**	**Burst Time**
P1	0	3	5
P2	1	1	2
P3	1	2	8
P4	3	1	5
P5	7	4	7

Calculate Average Turnaround Time, Average waiting time for the following algorithms. The time slice is 2 ms and the highest number has the highest priority (ignore the arrival time in non-preemptive scheduling) for the following scheduling.

(a) FCFS (b) SJF (Preemptive and non-preemptive)

(c) Priority (Preemptive and non-preemptive) (d) Round Robin

25. Consider the following set of processes

Process	**Arrival Time**	**Priority**	**Burst Time**
P1	0	1	20
P2	0	5	12
P3	3	3	3
P4	4	4	18
P5	8	2	8

Calculate Average Turnaround Time, Average waiting time for the following algorithms. The time slice is 5 ms and the lowest number has the highest priority (1 ms is context switching time and ignore the arrival time in non-preemptive scheduling) for the following scheduling.

(a) FCFS (b) SJF (Preemptive and non-preemptive)

(c) Priority (Preemptive and non-preemptive) (d) Round Robin

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT III:DEADLOCK

STRUCTURE

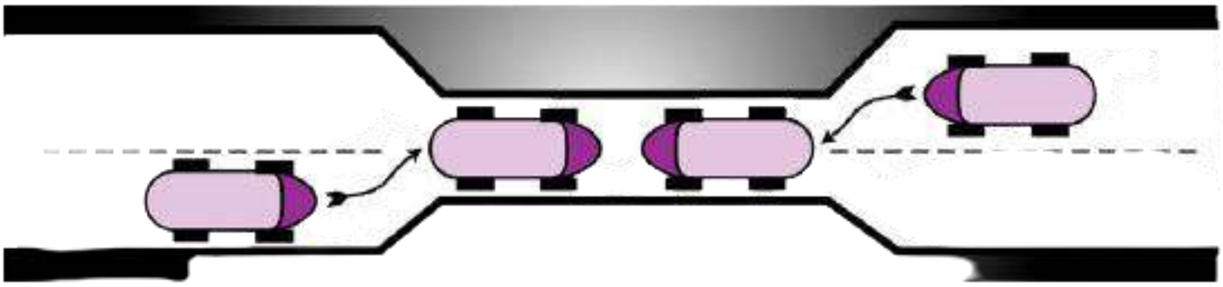
3.0 OBJECTIVES

- To understand the Deadlock Problem
- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

3.1 INTRODUCTION TO DEADLOCK

There is a situation in which two processes wait for each other. The resources of a system may be limited or less as compared to the number of processes. In multiprogramming, numerous processes may require a predetermined number of resources. When a process requests for a resource, the Operating System checks whether the resource is available or not. If the resource is available, then it is allocated to the process. On the other hand, if the resource is not free (available), the process enters the waiting state. Sometimes it happens that the process remains to wait, for a long time because the requested resource is holding by other waiting processes. This situation leads to **DEADLOCK**. It is a condition, wherein a set of processes are waiting forever for the resources, held by each other. None of them can proceed with its execution.

In routine life, a traffic jam on the narrow bridge is an example of Deadlock, when cars came from both side, but only a single car crossed from the bridge at a time and both car drivers refuge to back the cars as shown in the figure below:

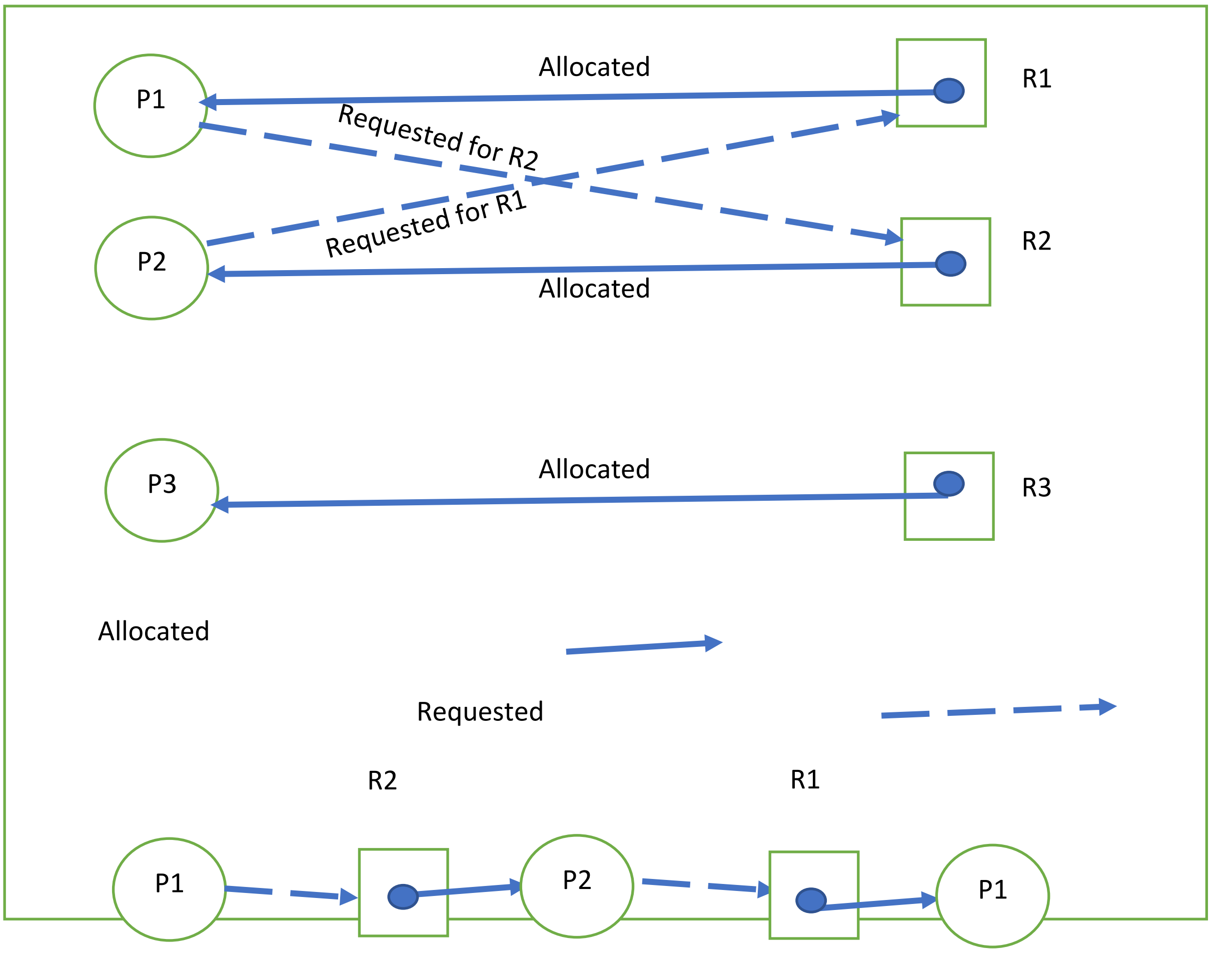


Example: Consider a scenario where there are three processes P1, P2, P3, and three resources R1, R2, R3. P1 requires two resources R1 and R2 for its execution. Also, P2 requires two resources R1 & R2 for its execution .P3 requires only R3. Initially, R1 is allocated to P1, and R2 is allocated to P2.

Here, (i) R1 is allocated to P1 and P1 is requesting R2.

(ii) R2 is allocated to P2 and P2 is requesting for R1.

(iii) R3 is allocated to P3.



3.2 SYSTEM MODEL

In the system, there are n number of processes namely p_1,p_2,p_3, …… $p_{n-1,}$ $p_{n,}$ and m number of resource types namely R_1, R_2,…..R_{m-1}, $R_{m.}$ In the System model, R resources are to be circulated among some processes P. The resources are then divided into many types, each consisting of some definite quantity of identical instances which is represented as w. The main example of resource types is *CPU cycles, memory space, Input-Output devices* such as keyboards, printers, and CD-DVD drives, directories, and files. Each resource type has instances. Instances mean the number of resource types that represent as w. Each resource type R_i has W_i instances When a system has 4 printers, then the resource type printer got four instances. Each process uses a resource as the following sequence:

- **Request:** Process request for the resources. If resources are available then the system allocates to process otherwise process waits for resources.
- **Use**: Then process used the resources when the system allocates them.
- **Release:** Process used the resources and then released them.

3.3 DEADLOCK CHARACTERIZATION

3.3.1 Necessary Conditions For Deadlock

The following four conditions must hold simultaneously, for a deadlock to occur:

1. **Mutual exclusion**

Mutual exclusion implies that a resource can be utilized exclusively by only one process at a time in a non-shareable mode. If any other process wants to use that resource, then it must have to wait, until it is released by the former process. For example, P1 holds R1 and P2 holds R2, both in mutually-exclusive mode.

2. **Hold and wait**

It implies that there exists a process that is holding at least one resource which is waiting for another resource or resources, which is being held by other processes. It implies that there exists a process that must be holding some resources (at least one) in a non-sharable mode and at the same time must be waiting for the other resources, which are held by other processes in a non-sharable mode. Like in the above example: P1 is holding R1 in a non-shareable mode and at the same time trying to acquire R2, which is currently held by P2 in a non-sharable mode. Similarly, P2 is holding R2 in a non-shareable mode and at the same time trying to acquire R1, which is currently held by P1 in a non-sharable mode. This situation is called hold and wait for conditions.

3. **No pre-emption**

It implies that the resources cannot be pre-empted forcefully from a process. After the completion of the process, a resource is released voluntarily by the process.

4. **Circular wait**

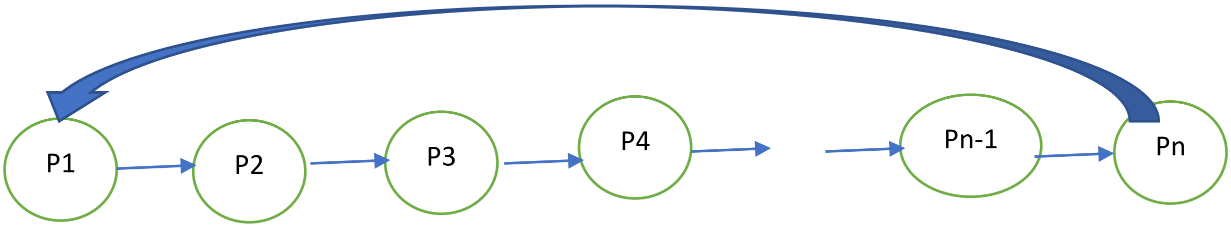
Circular wait implies that the processes are waiting for resources that are circularly held by another process. In other words, Pi is waiting for a resource held by Pi+1 and so on Pn-1 is waiting for a resource which is held by Pn, and at last, Pn is waiting for a resource held by Pi. For example, Let a set of processes, say (P1, P2, P3, ---, Pn). They must wait in a circular way for the resources held by each other i.e.

P1 is waiting for a resource which is held by P2

P2 is waiting for a resource which is held by P3

Pn-1 is waiting for a resource which is held by Pn

Pn is waiting for a resource which is held by P1



3.3.2 Deadlock Detection

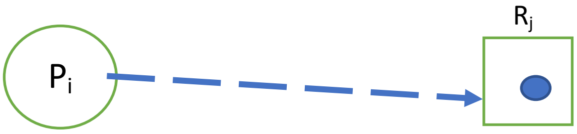
When all the four essential condition holds at the same time then deadlock occurs. Another way to detect deadlock is Resource Allocation Graph (RAG)

Resource Allocation Graph

The Resource Allocation Graph is a directed graph which is a set of vertices which is denoted by V and a set of edges which is denoted by E. Vertices (V) is divided into two types. Processes (P) and Resources (R). P = {P_1, P_2, - - -, $P_{n-1,}$ P_n} the set consisting of all the processes in the system. A process is represented by a circle, with the process name indicated with a label inside the circle. R = {R_1, R_2, - - - -, R_{n-1}, R_n} the set containing of all resources types in the system. A resource is represented by a square, with dots inside the square representing different instances of that resource.

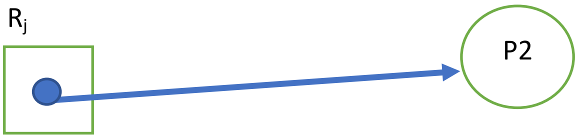
There are two types of edges: Request Edge & Assignment edge.

An edge from process P_1 to Resource R_j represents a request edge.



Request Edge

An edge from resource Rj to process Pi represents an assignment edge.



Assignment Edge

Note: The assignment edges originate from the instance within the resource symbol, which indicates which instance is assigned to the process.

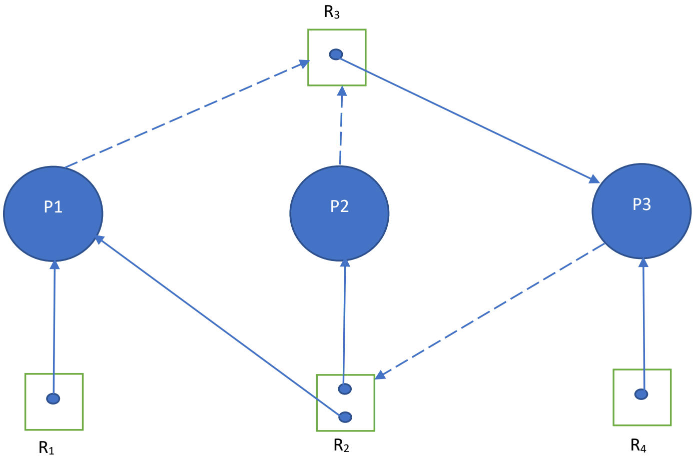
If there is no cycle in RAG, it indicates that there is no Deadlock occurred.

If there is a cycle detected in the graph and the resources involved in the cycle have only one instance per resource, then it indicates that a deadlock exists.

If there is a cycle detected in the graph and some of the resources involved in the cycle have multiple instances per resource, there may be a deadlock exists.

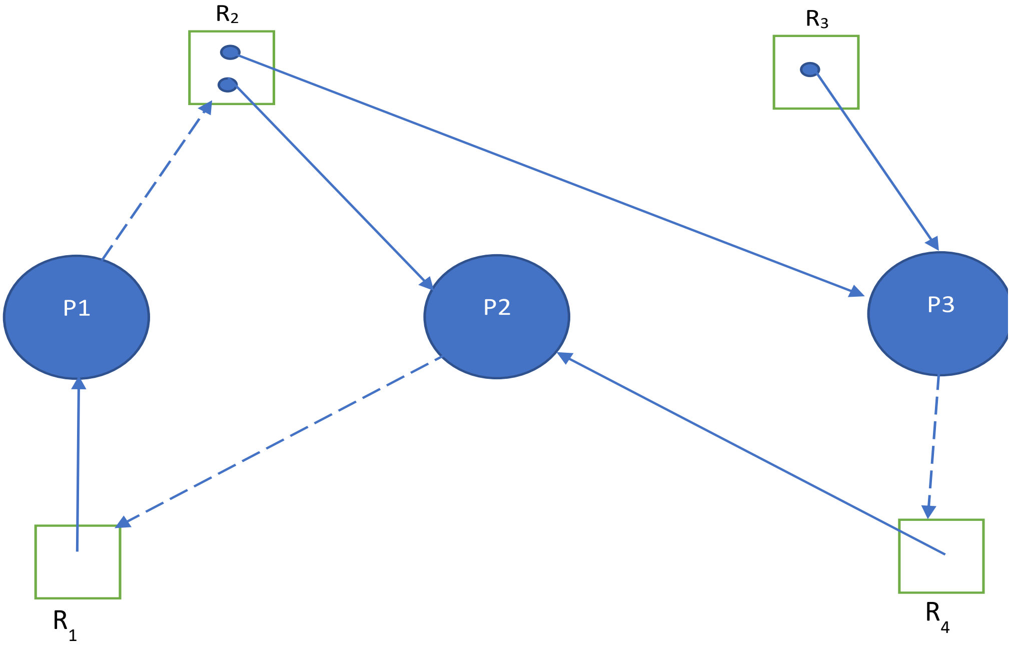
For example

Let there are 3 processes P1, P2, P3, and 4 resources R1, R2, R3, R4. R1 has one instance, R2 has 2 instances and R3 has one instance, R4 has one instance. R1, R2 is allocated to P1 and requested for R3. R2 is allocated to P2 and requested for R3. R3, R4 are allocated to P3 and requested for R2.



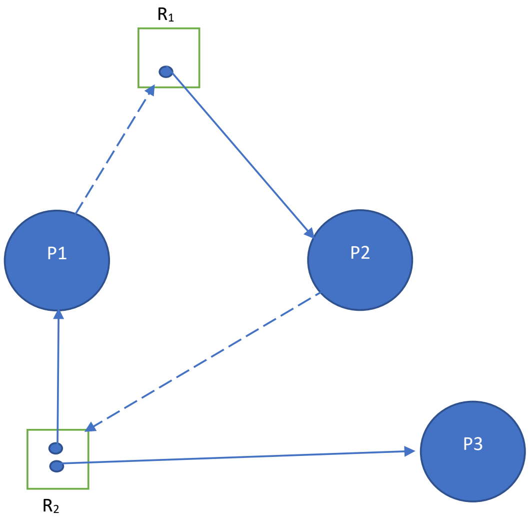
Resource Allocation Graph

Example of RAG with a cycle Deadlock



Resource Allocation Graph with Deadlock

Example of RAG with a cycle but no deadlock

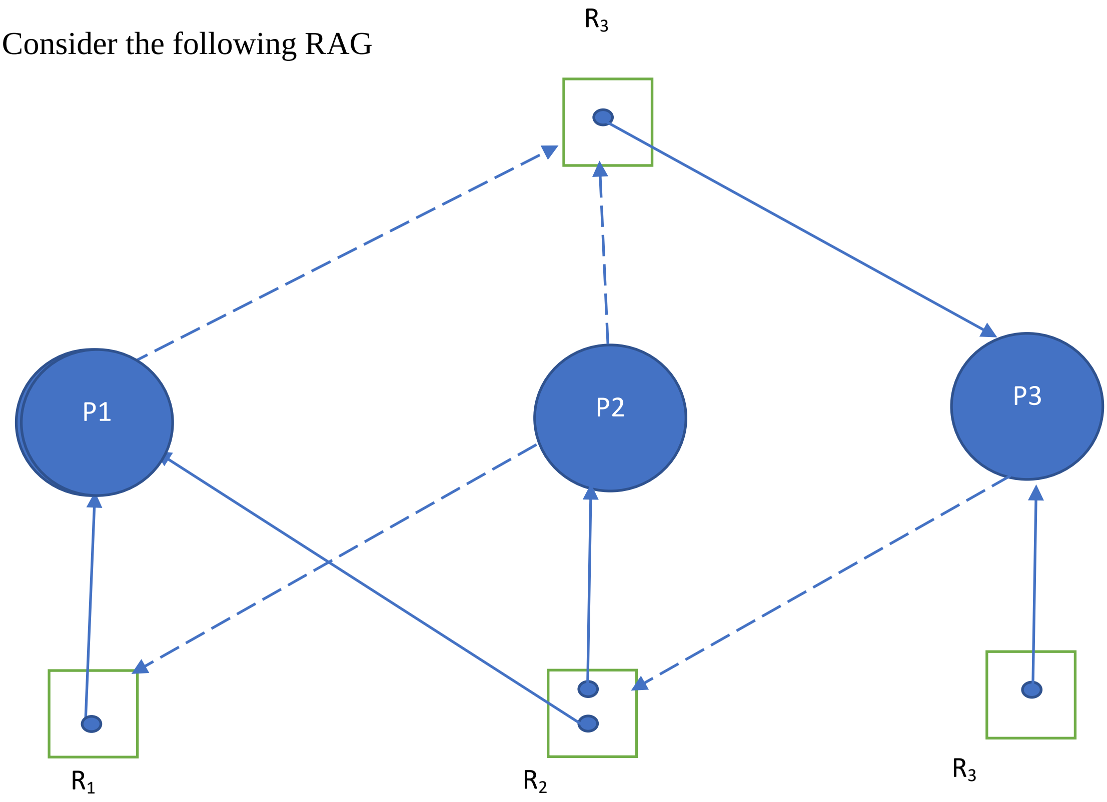


Resource Allocation Graph with Cycle but no Deadlock

Process Wait for Graph (PWFG)

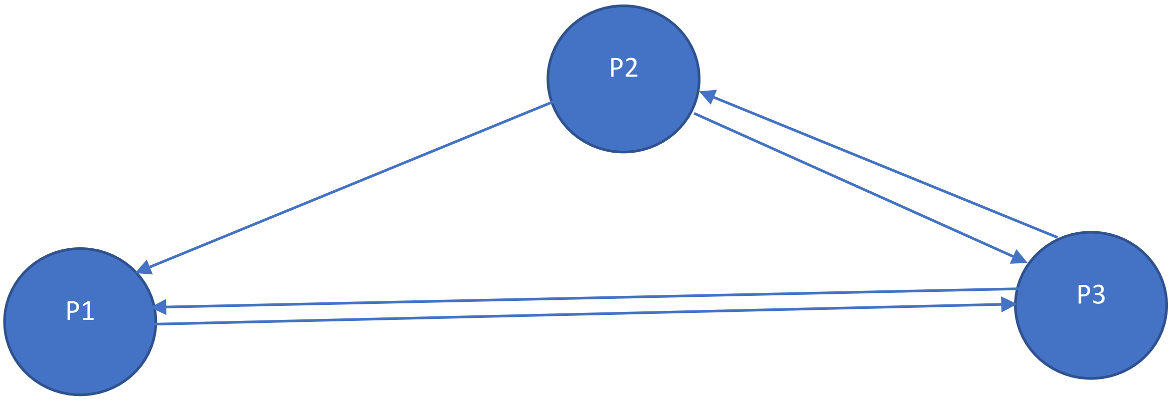
PWFG can be obtained by collapsing the resources symbol in RAG. An edge from process P1 to P2 indicates that P1 waiting for a resource that is currently held by P2.

Consider the following RAG



Example of Resource Allocation Graph

Equivalent PWFG for above RAG



Process Wait for Graph (PWFG)

The PWFG contains a cycle, thus indicated that deadlock exists.

3.3.3 Deadlocks Management

Different OS adopts different strategies to handle deadlocks. The policies to handle deadlocks can be classified into four major groups:

- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection and Recovery
- Deadlock Ignorance

The deadlock prevention techniques prevent the occurrence of at least one of the four conditions that cause deadlocks. By preventing are of the necessary four conditions deadlock can be prevented. The deadlock avoidance techniques acquire information in advance about which resource a process will claim at what stage of execution. The OS allocates resources in such a manner that no deadlock occurs. Deadlocks can be avoided by maintaining the system always in a safe state. The system is said to be in a safe state if all pending processes can be successfully executed in some sequence. That sequence is called a safe sequence and processes are in a safe state. The third category of deadlock management strategies is known as detection and recovery. These types of techniques allow deadlocks to occur detect deadlock when it occurs and then apply certain methods to recover from deadlock.

Finally, an OS can assume that deadlock will never happen or rarely occur and fully ignore it. This approach might be referred to as the no-policy approach. The positive point of this approach is it saves CPU time and memory space for deadlock management, unlike those required for detection, prevention, or avoidance methods. This strategy has been adopted in UNIX.

3.3.4 Deadlock Prevention

This technique allocates resources in such a manner that at least one of the four necessary conditions of deadlock cannot occur. It denies at least one of the four conditions required for the occurrence of a deadlock. These techniques do not utilize a resource properly. For example, an OS adopting a deadlock prevention technique allows recourses to be preempted from a blocked process. No preemption condition is violated and deadlock does not occur.

We have already discussed that there are four conditions to be satisfied for the occurrence of a deadlock. Below is a brief discussion about how the conditions for a deadlock can be prevented.

1. **Mutual exclusion condition**

The mutual exclusion condition indicates the locking of resources in exclusive, also known as the non-shareable mode that leads to blocking of resources. An OS can avoid blocking resources by locking the resources in shareable mode, if feasible. For example, two processes need to read data from a file.

The OS can allow locking of that file in shareable mode by two processes simultaneously. Locking resources in shareable mode prevents waiting for a resource. When a process request to look at a resource in shareable mode, the OS allows the locking instantly and no time is wasted waiting to lock the resource exclusively.

Some resources cannot be shared by multiple processes. For example, two processes cannot simultaneously share a scanner. Again when a process writes data to a file, it needs to lock the file in exclusive mode.

2. **Hold and Wait**

An OS can avoid this by adopting a strategy that a process must request for all resources it requires at the same time. This strategy can be implemented in two ways.

Firstly, a process p requires all the resources it needs when the execution begins. On the other hand, a process can start execution with a minimum set of resources required and request the other resources when required at the time of execution. In the second approach, a process must release all the resources it presently holds before requesting any other required resources.

For example, a process p requires resources R1, R2, R3, and R4 in exclusive mode in the sequence: R3 and R1 simultaneously at the beginning of execution, then R1, R2, and R3 simultaneously, and lastly R1 and R4 simultaneously. When applying the first strategy, process P acquires all four simultaneously before it starts execution.

When applying the second strategy, the process p needs not to acquire all the four resources at the beginning. The process p1 acquires the resources R1 and R3 at the time of starting execution. Soon the process p requires R2. The process p releases all the acquired resources R1 and R3 and sends a request to the OS for acquiring the resources R1, R2, and R3. Later, the resources R2 and R3 are not required and R4 is needed. The process p releases all the acquired resources R1, R2, and R3 and sends a request to acquire R1 and R4.

3. **No Preemption**

An OS can adopt a policy to avoid this condition. If a process requests an unavailable resource, the process must release all the resources it has presently acquired and wait for the required resources. In other words, when a process, P requests a busy resource this policy allows other processes to preempt the resources currently held by P.

Certain resources are good candidates to be preempted. For example, CPU, and memory need a mention. While revoking the control of the resources from a process the OS updates the process control block of the process. Not all the resources can be preempted. For example, a partly updated file cannot be preempted since data will be lost.

4. **Circular Wait**

An OS can impose an ordering of resources for avoiding the circular wait. Resources in a system are grouped into certain categories. For example, all magnetic disks form a group and all scanners from another group. Each category is assigned a number. When a process acquires a resource belonging to a specific category, no other process belonging to a category having a lower number can be claimed.

Formally, if there are n categories of resources in a system, ranging from 0 to n-1, and if a process acquires a resource belonging to category c, then the process can only request a resource belonging to category c+1.

This helps to prevent a formation circle because a process holding the control of a resource of the category n-1 cannot claim a resource of the type 0. The disadvantage of this technique is each process needs to acquire all the required resources in a predetermined and specific order depending upon the arrangement of numbers assigned to various categories of resources

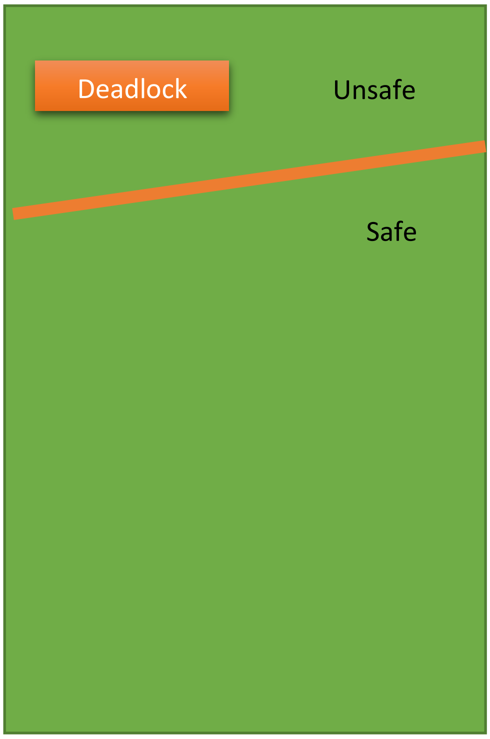
3.3.5 Deadlock Avoidance

In deadlock avoidance, Operating System requires additional information about which process requires which resource in advance to avoid a deadlock. The fundamental concept of deadlock avoidance is that OS only entertains those requests for resources by processes that will not lead to a deadlock. The deadlock-avoidance algorithm regularly inspects the resource-allocation state of the processes to confirm that there can never occur a circular-wait condition. Resource-allocation *state* is explained by the total number of available resources and number of resources allocated to the processes, and the maximum demands of resources by the processes.

Safe State

When processes request resources and it is available in the system and allocated immediately then the system in a **safe state**. The system is in a safe state if there exists an order of processes <P_1, P_2, ..., P_n> which satisfied their resources in that sequence. For each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < i$. That is: If P_i resources wanted are not instantly available, then P_i can wait till all P_j have finished. When P_j is finished, P_i can get needed resources, execute, return allocated resources, and terminate. When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

If a system is in a safe state then there is no Deadlock. If a system is in an unsafe state then there is the possibility of deadlock. To avoid deadlock, it is must confirm that a system never entered an unsafe state and it should be in a safe state.



Relationship between safe, unsafe, and deadlock state

Avoidance algorithms

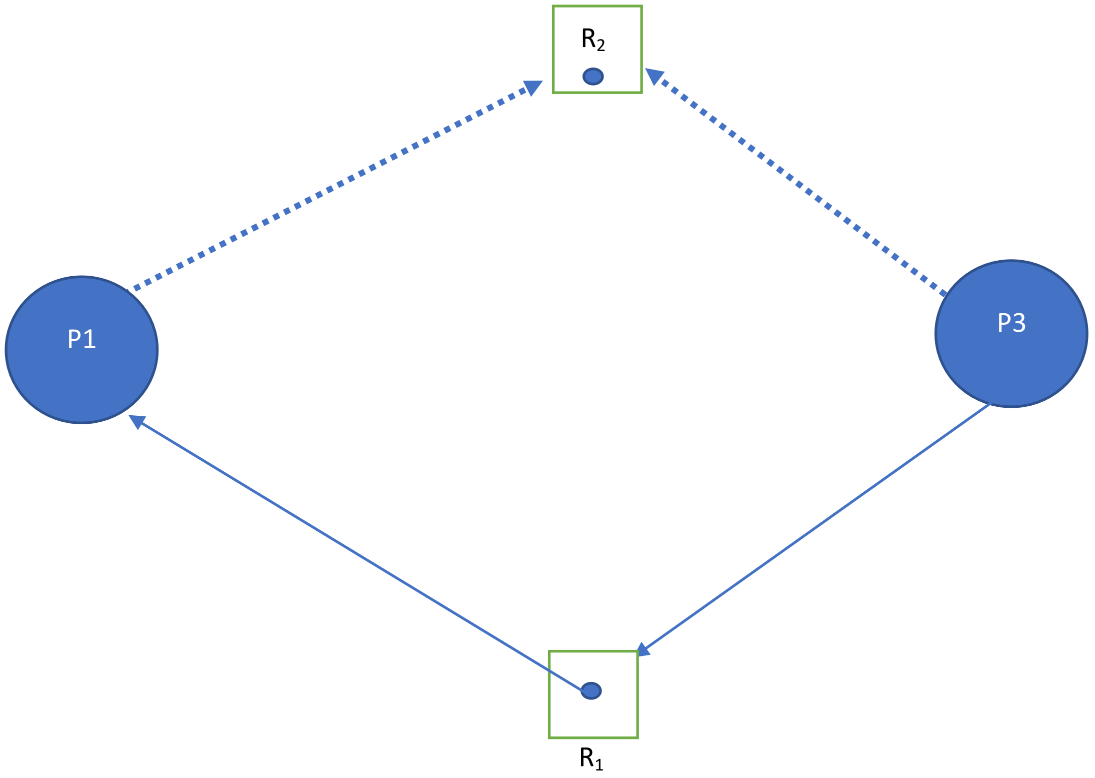
There are two algorithms to avoid deadlock

For the Single instance of a resource type, in this algorithm, we use a Resource-Allocation Graph (RAG) and the second algorithm which is deals with multiple instances of a resource type, then we use the Banker's algorithm.

Resource-Allocation Graph (RAG) Scheme

In Resource allocation Graph Scheme Claim edge $P_i \rightarrow R_j$ showed that process P_i may demand resource R_j; it is denoted by a dashed line. Claim edge changes to request edge when a process needs a resource. Request edge changed to an assignment edge when the resource is assigned to the process. When a resource is free by a process after execution, the assignment edge reconverts into a claim edge. Resources must be demanded in advance in the system.

Suppose that process P_i requests a resource $R_{j.}$ The request can be approved if and only if changing the request edge to an assignment edge does not result in the creation of a cycle in the Resource Allocation Graph(RAG).



(i) Unsafe state in RAG

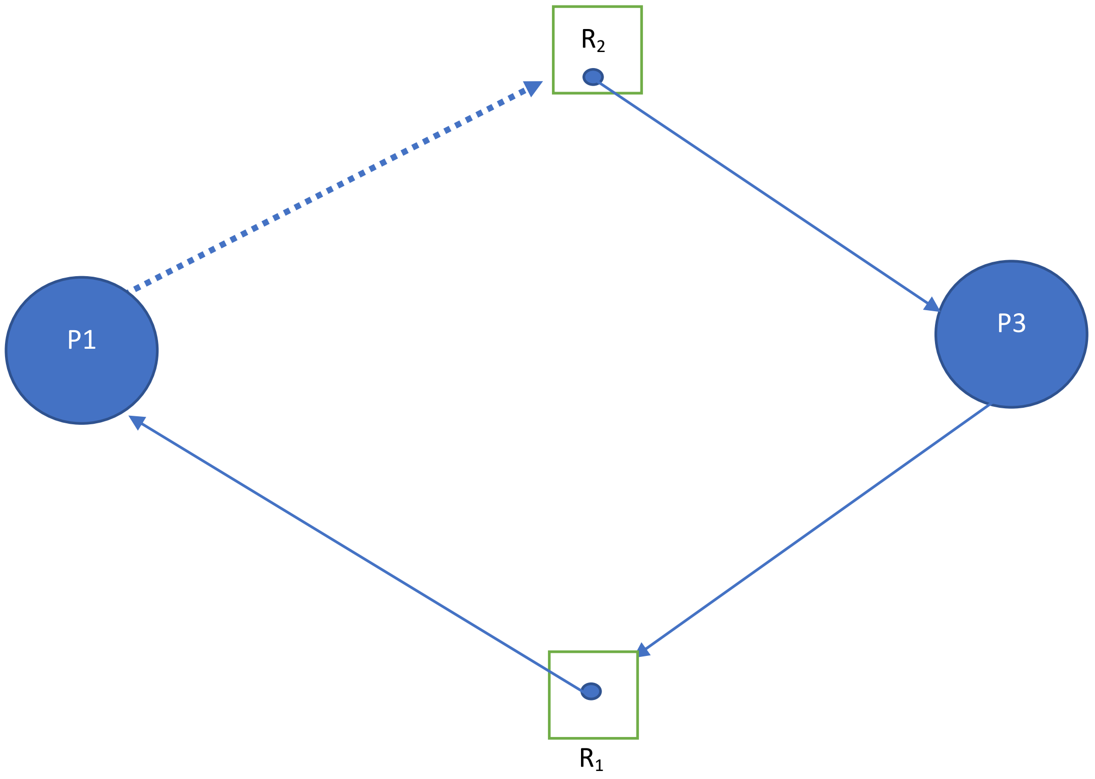


Figure showing (i) safe and (ii) unsafe state in RAG

BANKER’S ALGORITHM

The **banker’s algorithm** adopts the deadlock avoidance strategy. The name, banker’s algorithm, depicts the similarity of the concept in the field of banking. A banker never allots the cash available in such a manner that the banker cannot fulfill the needs of its clients.

A scheduling algorithm that can avoid deadlocks is BANKER’S ALGORITHM. It is modeled on the way that a single banker might deal with a group of customers to whom he has granted lines of credit. The banker’s algorithm can be implemented for a single resource or multiple resources. The banker algorithm keeps track of all the resources currently available to the operating system and all the resources which are allocated to the processes. It maintains the various data structures to store the information about the number (instances) of resources. Whenever a process requests a resource, the banker’s algorithm checks the availability of the resources and acts accordingly. Finally, it helps to check that the system is in a SAFE STATE or not.

Data Structures for the Banker’s Algorithm

Suppose n be the number of processes, and m be the number of resource types.

Available: Array of length m. If available $[j] = k$, it means that k instances are available of resource type R_j.

Max: n x m matrix. where n is the number of rows and m is the number of columns. $Max\ [i,j] = k$, it means that k instances maximum needed by process P_i of the resource type R_j.

Allocation: n x m matrix. where n is the number of rows and m is the number of columns Allocation$[i,j] = k$, it means that k instances are currently allocated to process P_i of the resources R_j.

Need: n x m matrix. where n is the number of rows and m is the number of columns $Need[i,j] = k$, it means that P_i may need k more instances of R_j to complete its task. It can be calculated by matrix subtraction:

$Need\ [i,j] = Max[i,j] – Allocation\ [i,j]$.

Safe State Algorithm

This algorithm is applied by an Operating System to determine whether a system is in a safe state or not. A system is safe if the system can allocate resources to every process in some order and still is in the safe state which means that its avoiding deadlock. In other words, a system is in a safe state only if there occurs, a safe sequence.

Let ‘m’ be the total number of resources and ‘n’ be the total number of processes in the system.

Let ‘Work’ and ‘Finish’ be the two arrays of length ‘m’ and ‘n’.

(1) Set Work = Available and also Set value FALSE for all the members of array Finish

FINISH[i] = FALSE, for i = 0 to n - 1

In other words, it is assumed that all the processes are to be executed and neither of the processes is completed yet.

(2) Find a process "i" such that following both conditions should be satisfied

Finish[i] = FALSE and Need[i] < = Work

If no such "i" exists ,then go to step (4).

In other words, find a process that is not completed yet and whose need is less than available.

(3) Set Work = Work + Allocation$_i$ (Allocation of process i)

Set Finish[i] = TRUE.

Go to step (2).

In other words, after the completion of the execution of a process, the process will return all the resources to the system. So, the available resources of the system will increase.

(4) Check if Finish[i] = TRUE for all "i" at that time

System is in safe state

else

System is not in safe state

Consider an example

Let there are 3 resources and 5 processes such that

Processes	Allocation			Max			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P0	0	1	0	6	5	3	2	3	2
P1	2	0	0	3	3	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	5	3	2			

Find the Need matrix by using the formula:

Need = Max – Allocation

So, the need matrix is:

Processes	Need		
	R1	R2	R3
P0	6	4	3
P1	1	3	2
P2	6	0	0
P3	0	1	1
P4	5	3	0

Hereby considering the need of all the processes, we have to find such a process whose need is less than available. It is visibly that the need of the process P1 is less than Available, so the process P1 will be executed first. (As 1 3 2 < 2 3 2)

After the execution of process P1, the P1will return all the resources to the system. So, the available resources will increase as the following :

Available = Available + Allocation_i

In the same way, after the completion of a process, the Available will be updated, as the process returns the resources to the system after being executed.

So, the processes will be executed in the following sequence

< P1, P3, P4, P0, P2>

Explanation of Safety Algorithm

First, we set all processes in the false state which means that no processes are executed or completed yet, and set work is equal to available. Then we find repeatedly a process that is false state and whose need is less than equal to work. If we find such a process then we change its state to true and its resources are deallocated which is added in work. In this way, the number of available resources is increased. Then we check all the processes are true or not. If all the processes are true then the system is in a safe state. If even one process is in a false state then the system is in an unsafe state.

Resource Request Algorithm

The second part of the banker's algorithm, known as the resource-request algorithm determines whether granting a resource according to any claim by a process leads to an unsafe state. This algorithm is applied when a process sends a request to grant one or a set of resources.

Request_i = request vector for process

P_i = If request_i [j] = K,

it means that P_i requested the K instances of resource type R_j.

(1) If Request[i] <= Need [i] (Check request is genuine or not)

then

Goto step (2)

Else

"Raise an error condition"

(Because the process has exceeded its maximum claim)

(2) If Request[i] <= Available (Check weather system has available resources or not)

Then

Goto step (3)

Else

The process P_i has to wait because the resource is not available yet.

(3) Let the system pretend that the resources have been allocated by changing the following:

Available = Availablei – Requesti ;

Allocationi = Allocationi + Requesti ;

Needi = Needi – Requesti;

Then check with the safety algorithm, if the result of the safety algorithm is safe, then the transaction is completed and the process is allocated the requested resources.

However, if the system is in an unsafe state, then the process must wait and the old allocation state is restored.

Explanation of Resource Request Algorithm

In this algorithm, first, we check whether the request of the process is less than its need or not. If the request is more than need then it is an error because the process claims more resources from its maximum claim. Else we check current request is less than available or not. If it is less than or equal to available then we allocate the request and update data structure such as allocation of the process by adding a request in allocation, the request is subtracted from need as well as available. Then we check the system is in a safe state or not by applying a safe state algorithm. If available is less than need then the request can not be granted and the process will wait because the resource is not available yet.

Advantages

Advantages of banker’s algorithm, it allows mutual exclusion, hold and wait, no preemption conditions. The system assures that a process will be allocated resources without deadlock.

Disadvantages

The disadvantage of a banker’s algorithm is the overhead of simulation and calculation before each time a process requests for a resource. Deadlock avoidance techniques cannot be applied until an OS knows the resource requirement of a process in advance. This approach may lead a process to starvation while for a long time to get the requested resources allocated by the OS.

Illustration

Consider the following table with 5 processes P_0 to P_4 and 3 resource types namely X Y and Z; X has 9 instances, Y has 5 instances, Z has 8 instances.

	ALLOCATION	MAX
	X Y Z	**X Y Z**
P_0	2, 2, 3	6, 2, 3
P_1	2, 0, 1	6, 1, 2
P_2	0, 1, 1	3, 1, 2
P_3	1, 0, 0	1, 0, 1
P_4	1, 1, 2	6, 2, 3

Using Banker's algorithm check whether the system is in a safe state or not. If it is a safe state then if P_1 requests (1, 0, 1) then it is a safe state or not. If it is not in a safe state then, what changes are required in available resources to make the system safe?

Total instance of resources is

X = 9, Y = 5, Z = 8

Total allocated resources to P_0 to P_4 are

X = 6, Y = 4, Z = 7

Now available is

X = 3, Y = 1, Z = 1

Now calculate Need: i.e. $\text{Need}_i = \text{MAX}_i - \text{Allocation}_i$

Process	**Allocation**	**MAX**	**NEED**	**Available**
	X Y Z	**X Y Z**	**X Y Z**	**X Y Z**
P_0	2, 2, 3	6, 2, 3	4,0,0	3,1,1
P_1	2, 0, 1	6, 1, 2	4,1,1	
P_2	0, 1, 1	3, 1, 2	3,0,1	
P_3	1, 0, 0	1, 0, 1	0,0,1	
P_4	1, 1, 2	6, 2, 3	5,1,1	

Now apply safe state algorithm to check whether the system is safe state or not.

First step is set all process equal to false i.e. P_0, P_1, P_2, P_3, P_4 are FALSE.

Now find such process which false and whose $\text{need}_i \leq$ Available i.e. P_2.

Now update data structure and status of processes i.e. $finish_i$

Available = Available + $Allocation_2$

$Finish_2$ = True

Now new Available = 3, 1, 1 + 0,1, 1

= 3, 2, 2

All processes are not true so again go to step - 2 to find a process that is yet false and needs is less than equal to available.

$Need_3$ < = Available i.e. P_3

Now update data structure and status of processes i.e. $finish_i$

Now Available = Available + $Allocation_3$

New Available = 3, 2, 2 + 1, 0, 0

=> 4, 2, 2

$Finish_3$ = True

Now 3 processes are false i.e. P_0, P_1 & P_4 & Available = 4, 2, 2

All processes are not true so again go to step - 2 to find a process that is yet false and needs is less than equal to available.

$Need_0$ < = Available i.e. P_0

Now update data structure and status of processes i.e. $finish_i$

Now Available = Available + $Allocation_0$

=> 4, 2, 2 + 2,2,3

= 6, 4, 5

$Finish_0$ = True

Now 2 process are false i.e. P_1 & P_4 and Available = 6, 4, 5

All processes are not true so again go to step - 2 to find a process that is yet false and needs is less than equal to available.

$Need_1$ < = Available i.e. P_1

Now update data structure and status of processes i.e. $finish_i$

New Available = Available + $Allocation_1$

= 6,4,5 + 2, 0, 1

=> 8, 4, 6

Finish$_1$ = True

Now only one process i.e. P$_4$ is false if its need is greater than Available then system is not safe state but need$_4$ < Available

Update Available & Finish

New Available = Available + Allocation$_4$

= 8, 4, 6 + 1, 1, 2

=> 9, 5, 8

Now finish is true so there is no false in finish and all resources are available now i.e. 7, 3, 6.

□ System is safe state & sequence is

< P$_2$, P$_3$, P$_0$, P$_1$, P$_4$>

The system is in a safe state if the process runs in the above sequence.

Now check if P$_1$ requests (1, 0, 1) then if it is allocated to the process then whether it is in safe-state or not? To check that the resource request algorithm follows.

In this algorithm first request is to check whether it is less than Need$_i$ or not. If it is greater than Need$_i$ its mean P$_i$ request more resources as it claims earlier so it can't be given & request directly terminated. If Request$_i$ is less than Need$_i$ then we check Request$_i$ < = Available if it is true then the request is granted & we check that now system is in a safe state or not by using the safe state algorithm used earlier. If it is false then at that time Request$_i$ can not be fulfilled and the process P$_i$ made to wait.

In this case Request P$_1$ is (1, 0, 1) which less than Need$_1$ and Available so we run resource request algorithm.

Set Available = Available - Request$_i$

= 3, 3, 1 - 1, 0, 1

Now Available = 2, 3, 0

Set Allocation$_1$ = Allocation1 + Request$_1$

= 2, 0, 1 + 1, 0, 1

Now Allocation$_1$ = 3, 0, 2

Set Need$_1$ = Need1 - Request$_1$

= 4, 1, 1 - 1, 0, 1

Now Need$_1$ = 3, 1, 0

Now Request is granted now we check after allocating the request whether the system is in a safe state or not. To check that we used a safe state algorithm. Now the situation of processes is given below:

Process	**Allocation**	**MAX**	**NEED**	**Available**
	X Y Z	**X Y Z**	**X Y Z**	**X Y Z**
P_0	2, 2, 3	6, 2, 3	4,0,0	2,3,0
P_1	3, 0, 2	6, 1, 2	3,1,0	
P_2	0, 1, 1	3, 1, 2	3,0,1	
P_3	1, 0, 0	1, 0, 1	0,0,1	
P_4	1, 1, 2	6, 2, 3	5,1,1	

Now by applying a safety algorithm there is no such process whose need_i < = available. So if the request of process_1 is granted then the system comes in an unsafe state. So, we revoke the request of Process_1 so that system may run safely. To overcome this problem we must need at least one A-type and at least two C-type Resource. If it will available in the available list then the system may come in a safe state.

3.3.6 Deadlock Detection and Recovery

Another approach to deadlock management is detection and recovery. In this approach, the system is not prevented from occurring a deadlock. The operating system periodically searches or analyzes that whether a deadlock has occurred. If the system is in deadlock, OS recovers from deadlock.

This approach of deadlock management has two phases, detection, and recovery of deadlocks. The algorithm to detect deadlocks in an OS can use a data structure similar to that of the data structures used in the context of deadlock avoidance.

A variation of the safety algorithm discussed in deadlock avoidance has been discussed below. In this algorithm, the data structures ALLOCATION, CLAIMS (Request), and AVAILABLE are the same as the data structure used in the context of deadlock avoidance. Another data structure used in this algorithm is FLAG.

The FLAG is a vector of the length equal to the number of processes in a system. FLAG is used to mark and unmark a process. For example, FLAG[X]=TRUE means the process Px is marked, and FLAG[Y]=FALSE means that the process Py is unmarked.

An OS verifies after a certain interval whether a deadlock has occurred. This algorithm finds a sequence of execution of processes that do not lead to an unsafe state. For example, there are 3 processes P1, P2, P3. Executing these processes in the sequence P2□P1□P3 does not lead the system to deadlock. The steps in this algorithm are:

1. Initialize FLAG[i] to FALSE, where 0 <= i<=s-1, and s denotes the number of processes.
2. Find a process Px such that FLAG[X]=FALSE and CLAIMS_X<=AVAILABLE. Go to step 6, if no such process is found.

3. Update the AVAILABLE vector because, after the termination of the process, Px, OS revokes the control of all the resources held by Px. Perform AVAILABLE = AVAILABLE + ALLOCATION$_X$.
4. Mark the process Px as FLAG[x]=TRUE.
5. Go to step 2.
6. If FLAG[i]=TRUE for all i, where 0<=i<=s-1, then all processes can be completed properly and there exists no deadlock, else the system is deadlocked with the processes which are unmarked that is FLAG[i] is set to FALSE.

The algorithm needs an order of $O(m \times n^2)$ processes to detect that the system is in a deadlock state or not.

The frequency of verifyin5r4ag the system whether a deadlock has occurred is an important issue. An OS can adopt any of the two policies. First, an OS can verify the system state whenever a process sends a request for resources. Alternatively, an OS can verify the system state after a certain interval of time.

An OS can adopt two strategies to recover from deadlock. Firstly, all the processes involved in the deadlock can be **terminated forcefully**. Alternatively, the OS can apply a **preemption technique** from a deadlock period.

Aborting deadlocked processes can be done in two ways.

An OS can abort all the deadlocked processes when a deadlock is detected.

On the other hand, an OS can apply the trial and error method. This means, one of the deadlocked processes is aborted and it is verified that whether the deadlock is over. This procedure is continued until the system reaches a safe state.

Which process is to be aborted first depends on the cost of terminating the process. If a process is 95% done, OS tries to avoid aborting that process. The number of processes to be aborted is also important in this context. For example, four processes are deadlocked. The OS can recover itself from deadlock by aborting either all of the processes A, B, and C or a single process D. In this case, an OS will sacrifice a single process to recover from deadlock.

The second approach of recovery from deadlock is preempting a resource from a process. In this case, a victim is select, the process from which a resource is revoked by an OS is rolled back to a safe previous state and continued later on. In this approach, the starvation problem may occur by selecting the same process as a victim again and again.

3.3.7 Deadlock Ignorance

In deadlock ignorance, we assume that our system never enters in the deadlock condition. It is only a theoretical concept. Practically it is not possible because if the system is not in a safe state then the system may be in a deadlock state or we can say that if the system satisfied all the necessary four conditions simultaneously then the system must be deadlock state which can be ignored.

Comparison Between Various Deadlock Management

Deadlock Prevention	Deadlock Avoidance	Deadlock Detection and Recovery
It is easy method	It is complex method	It is also a complex method
It adversely affects Utilization	Utilization improved at a cost	Depends on the frequency of detection
Involves violating at least one of the necessary conditions for deadlock	RAG Algorithm Bankers Algorithm (Multiple instances)	Maintain a wait-for graph and periodically invoke a "cycle-search" algorithm
Used in critical systems	Used in applications that have deterministic needs	Used in Database Applications

Points to Remember

- Deadlock: Process is said to be in a deadlock state if the process is waiting for a specific event that will not occur.
- The reusable resource can be safely used by only one process at a time.
- Four necessary conditions for deadlock are:
- (i) Mutual exclusion (ii) Hold and wait
- (iii) Circular waiting (iv) No preemption
- Resource Allocation Graph is used to detect the deadlock.
- Resource Allocation Graph is also known as RAG.
- If RAG contains a cycle then it may or may not deadlock the situation.
- In RAG, the process is represented by a circle and the resource is represented by a square.
- Process Wait For Graph is obtained from Resource Allocation Graph.
- Process Wait For Graph is known as PWFG.
- The policies to handle deadlock can be classified into four major groups
- Deadlock Prevention
    - Deadlock Avoidance
    - Deadlock Detection and Recovery
    - Deadlock Ignorance
- Deadlock prevention is a set of methods for ensuring that at least one of the necessary conditions cannot hold.
- Deadlock avoidance needs extra information about how resources are to be demanded.
- Safe order is an order of the process in which there exists one order in which all the processes can be executed without resulting in a deadlock.
- The deadlock avoidance algorithm enthusiastically examines the resource-allocation state to confirm that a circular wait condition can never exist.
- A safe state is not a deadlock state.
- Not all unsafe states are deadlocks.

- Deadlocks occur only when some process makes a request that cannot be granted immediately.
- Banker's algorithm is a deadlock avoidance algorithm.
- A safety algorithm is used to find the safe state of the system.

3.4 PRACTICCE EXERCISES

1. What is deadlock?
2. What are the necessary conditions for the occurrence of a deadlock?
3. How deadlock can be avoided.
4. What do you mean by RAG?
5. Explain methods for deadlock detection.
6. Explain safe state.
7. What are the different methods of handling deadlock.
8. Explain the Banker's algorithm with an example.
9. Explain the safe state algorithm with an example.
10. Explain the resource algorithm with an example.
11. Consider the following snapshot of a system.

Process	**Allocation**	**MAX**	**Available**
	M N O P	**M N O P**	**M N O P**
P_0	2, 2, 1, 3	7, 5, 3, 3	1, 1, 2, 1
P_1	2, 2, 1, 1	6, 2, 2, 4	
P_2	1, 2, 1, 0	4, 2, 4, 0	
P_3	1, 1, 0, 1	2, 2, 2, 3	
P_4	2, 1, 2, 0	2, 4, 3, 3	

Whether the system is a safe state or not. If process P3 request 0, 0, 1, 2 can this request granted immediately or not and also check after granting above request system is in sate state or not. If it is not in a safe state then what changes are required in available resources to make the system safe.

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT IV: MEMORY MANAGEMENT

STRUCTURE

4.0 OBJECTIVES

- Understanding Memory, Address Binding, Logical Vs Physical Memory
- Understanding Concept of Dynamic Loading and Linking, Swapping, Contiguous Memory Allocation, Segmentation, Paging, Demand Paging, Page Replacement algorithms

4.1 INTRODUCTION

The important task of allocating memory to processes, and efficiently, ensuring that processes have their instructions and data in main memory when needed, is termed as Memory Management.

Memory is one of the important resources of the computer system. The main memory is usually divided into two partitions, one for the resident operating system and the other for the user processes.

The operating system is responsible for memory management. It keeps track of the status of the memory, makes policy, allocates the memory, and then deallocates the memory from the process.

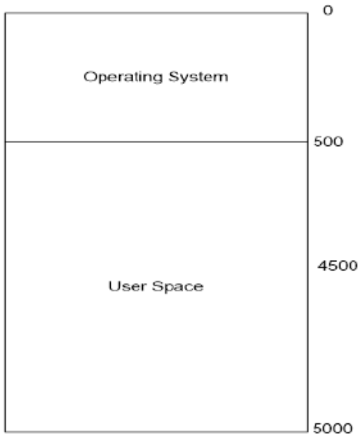


Figure: Main Memory

The functions of Memory Management

An operating system performs various activities for memory management:

1. It keeps track of the status of the memory, that whether it is free, allocated, available, or not available (full).
2. If the memory is free for the execution of the process, the Operating System chooses a policy to allocate the memory to the process.
3. According to the chosen policy (algorithm), it allocates the memory to the process.

4. After the execution of the process, it deallocates the memory.

4.2 ADDRESS BINDING

Address binding means map from one address space to another i.e. the logical addresses to real physical addresses in memory. To execute a program, it must be loaded into the main memory at a particular location. The instructions that use addresses in a program must be bound to proper address space in the main memory. Many instructions use “fixed” addresses these must be *bound* to “fixed” locations in the memory.

This binding of addresses of instruction and data to actual physical addresses can take place at compile-time, load time, and run time during the execution of a process.

To run a program there is a sequence of steps to execute a particular program which is as follow:

- A program is known as a source program and it is loaded into the main memory.
- A compiler or assembler or interpreter is required for compilation or assembly.
- After compilation or assembly, a code is generated by them which, is known as Object code.
- When a program is executed then, the program links with other object modules or system library which are loaded into linkage editors with the help of linker and loader.

At last, the process is stored in memory as a binary image, and system library files are dynamically loaded and dynamically linked.

4.3 DYNAMIC LOADING AND LINKING

In any program execution linking and loading plays a key role. Linking means creating an executable module of a program by linking the object codes which is created by the assembler or compiler. Other hand loaders, load these runnable modules to the main memory for execution.

Linking: It is the **procedure of linking all the modules or the sub-program or all the functions in a program for whole program execution**. It takes more than one object module and combines it into a single object file. The linker is also called a link editor. It takes object modules from the compiler or assembler and forms a runnable file for the loader.

Based on time Linking is classified into two categories – static linking and dynamic linking:

Linking of object modules is done at compile time and at load time. Compile-time linking is completed when the source code is converted into machine code is called Static linking. The load-time linking is done while the program is loaded into memory by the loader.

Loading

Loading is the **procedure of loading the program from auxiliary memory to the main memory** for execution.

Dynamic loading

It involves loading routines into memory only when required. This is done during execution. Dynamic loading reduces the memory requirements of large programs. This is especially the case if there is a large set of infrequently used routines.

Dynamic linking

It is often used for libraries. Only a "stub" of the library is kept in the image of the program. When a program calls one of these routines, the routine is loaded and linked into memory. All programs share one copy of the same library routine.

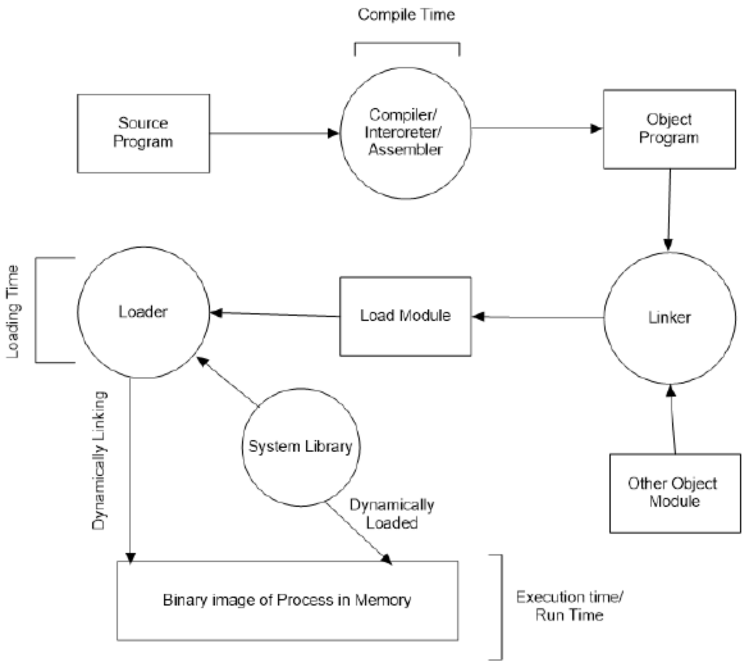


Figure Processing of a User Program

Address binding of instructions and data to memory addresses can happen at three different stages i.e. compile time, load time, execution time (as shown in figure Processing of User Program)

Compile-time: If the memory location is known a priori, **absolute code** can be generated. It must recompile code if starting location changes, after compiling the process.

Load time: It must generate **relocatable code** if memory location is not known at compile time. Final binding is delayed until load time.

Execution time: If the process can be moved during its execution from one memory segment to another, binding may be delayed. Then hardware support is needed for address maps. For example base and limit registers. Most general-purpose operating systems use the execution time-binding method.

4.4 LOGICAL VS. PHYSICAL ADDRESS SPACE

When a process is executing, the CPU would generate addresses, called **Logical Ad dresses**. There are two registers used to define logical address space. These are the Base register and Limit register. Base register stores the starting memory address of the process and the Limit register stores the size of a process. Let a process P whose size is 5000 i.e. value of the Limit register and it has the start address is 1000. i.e. value of Base Register. A pair of **base** and **limit** registers define the logical address space as the shown figure below

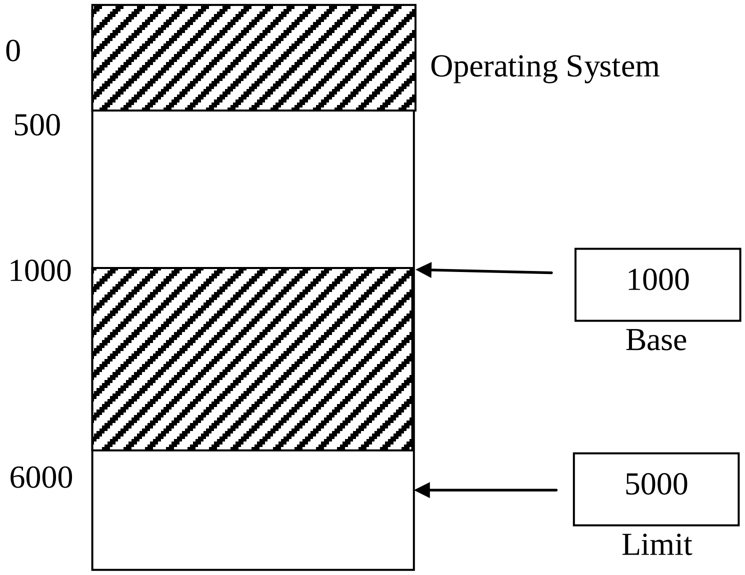


Figure Base and Limit Register

Executing processes that occupy corresponding physical addresses in the physical memory is known as **Physical Addresses.** Logical and physical addresses are the same in compile-time and load-time address-binding schemes but logical (virtual) and physical addresses differ in the execution-time address-binding scheme. A physical address is the effective memory address of instruction or data.

4.5 SWAPPING

Swapping is a mechanism in which a process can be swapped temporarily out of memory to a storage device (backing store) and another process brings in memory for execution. If a swapped-out process requires again then it is brought back into memory for continued execution.

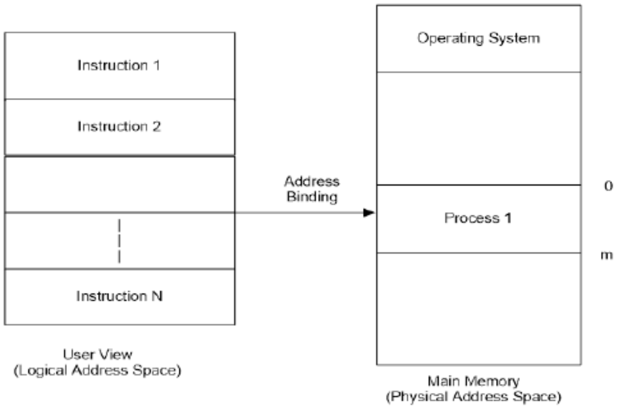


Figure Logical and Physical Address Space

The backing store is a fast disk large enough to accommodate copies of all memory images for all users. It must provide direct access to these memory images. When the process swapped out then it is called **Roll out** and when it brings in then it is called **Roll in.** The swapping variant is used for priority-based scheduling algorithms where the lower-priority process is swapped out due to the arrival of a higher-priority process. It can be loaded and executed. A major part of swap time is transfer time. The system maintains a ready queue of ready-to-run processes which have memory images on disk.

Example: Let there are n processes in the storage device and P4 in main memory which is not required. Now P2 is required so P4 is roll out and P2 is rolled in which is known as swapping as shown in the figure.

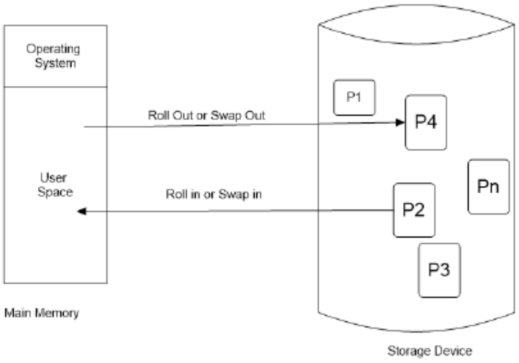


Figure: Swapping

4.6 MEMORY ALLOCATION

Memory management includes the various methods of allocating the memory to the different processes. The memory management techniques are basically of two types:

(i) Contiguous memory allocation.
(ii) Non-contiguous memory allocation.

Contiguous Memory Allocation

In contiguous memory allocation, the data and instructions of a program are sure to reside in a single contiguous memory area. It is a simple technique of allocating memory to the processes. There is no possibility of sharing data and instructions among the processes, which are loaded into entirely different portions of memory. The entire process is allocated a single contiguous memory area for its execution. Addressing is very simple as compare to the non-contiguous allocation methods.

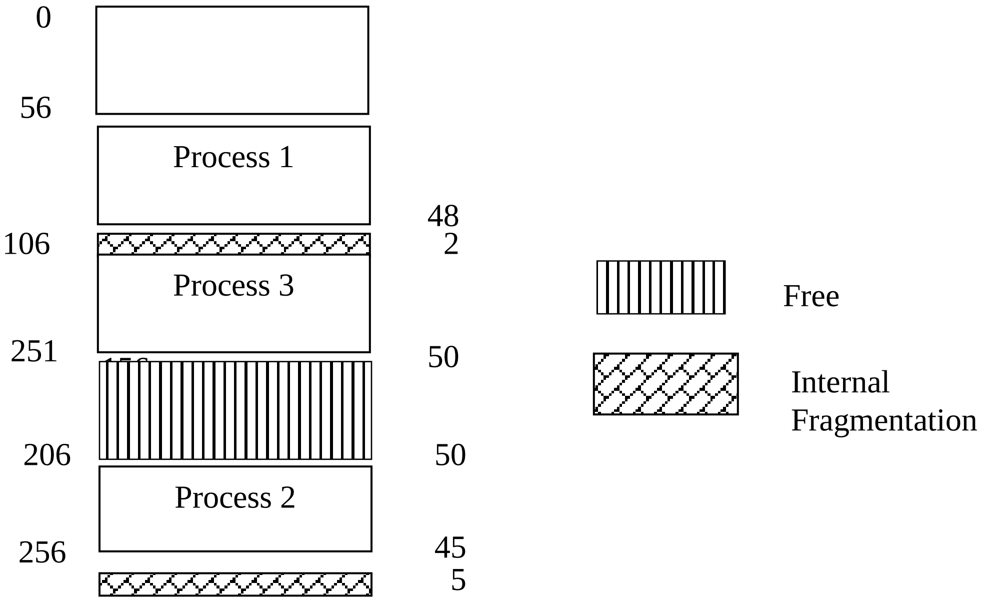
Fragmentation

In contiguous memory allocation, one problem occurs i.e. Fragmentation. It can be classified into two categories:

1. Internal Fragmentation
2. External Fragmentation

1. **Internal Fragmentation:** In fixed-size memory management, allocated memory to a process may be slightly larger than the requested memory by the process. The size difference between occupied memory and partition size, which is not used by another program is known as Internal Fragmentation.

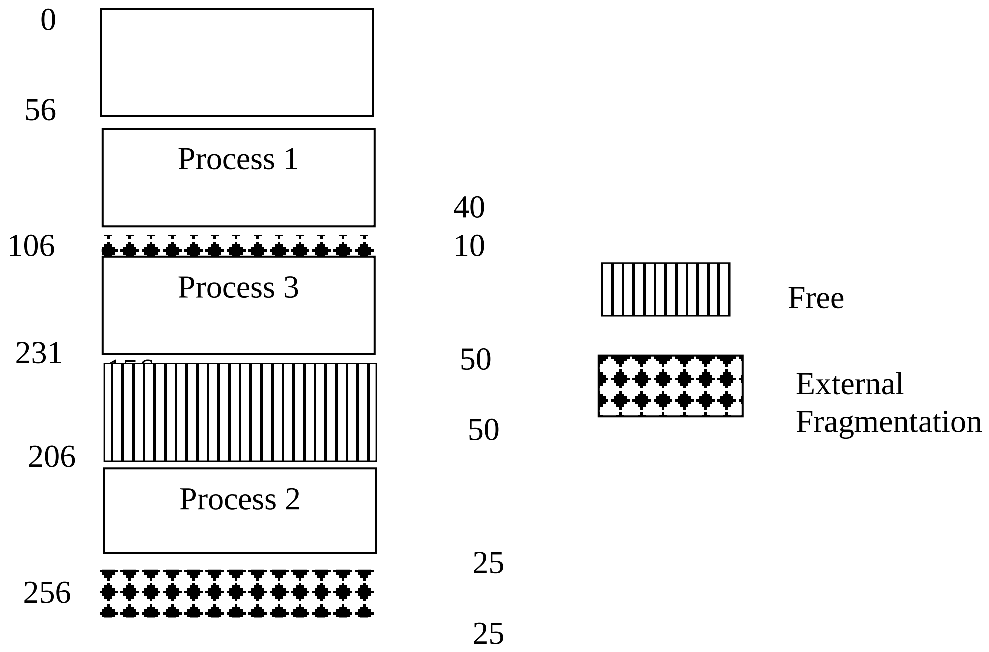
Example of Internal Fragmentation: Let, there are three processes P1, P2, P3 whose size is 48, 45, 50 respectively. The total memory size is 256 and OS occupies 56K and the rest of the memory is divided into fix partition of 50k.



Example of Internal Fragmentation

2. **External Fragmentation:** In variable size memory management, some small chunks are available in memory that cannot be effectively used. Total free memory is larger than the required memory by the requested process but it is not contiguous. It is known as External Fragmentation.

Example of External Fragmentation: Let, there are three processes P1, P2, P3 whose size is 40,25,50 respectively. The total memory size is 256 and OS occupies 56K and the rest of the memory is divided into fix partition of 50k.



Example of External Fragmentation

Non-contiguous Memory Allocation

In non-contiguous memory allocation, the data and instructions of a program may reside in non-contiguous memory locations. It is possible to allocate different memory locations to a process for its execution. In other words, the process can be divided into parts and can be allocated to different memory areas. It offers various advantages over, contiguous memory allocation like it permits sharing of code and data amongst processes. There is no external fragmentation of physical memory. It supports the vertical memory concept.

Contiguous Memory Allocation Methods

The memory is partitioned into different blocks of different sizes for accommodating the programs. The partitioning is of three types:

1. Single partitioning memory management
2. Multiple Fixed Partitioning
3. Multiple Variable Partitioning

Single Partition Memory Management

The main memory is divided into two partitions. One partition is for the resident operating system and, the other for the user processes. In the single partition memory management technique, the user area has a single partition. It is the simplest memory management technique.

Consider an example, where the total main memory is 256 MB. 56 MB is reserved for the operating system and 200 MB is for the user processes.

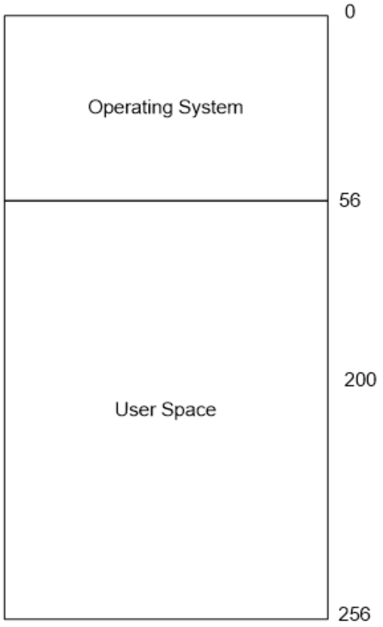


Figure: Single Partitioned Main Memory

Here, the user area has a single partition. At a time, only one program can be executed. The maximum length of the program, which the user can execute, is equal to the size of the partition of the user area. Here, the maximum size of a program can be 200 MB.

Internal fragmentation is much more. Since there is a single partition of memory, we can execute only one program at a time. Consider an example, where the size of a program is 50 MB. In this case, there will be an internal fragmentation of about 150.

Here, external fragmentation is also more. However, if the size of the program increases from the size of the partition, then it is not possible to execute the program.

Advantages

- It is a very simple memory management technique.
- Any program of size less than the size of the user area can be executed easily.

Disadvantages

- Multiprogramming is not possible.

- Internal fragmentation is more.
- It is not possible to execute a program of size greater than the size of the user area.
- The maximum size of a program to be executed is fixed.

Multiple Partitions with Equal Size

In this, whole memory is divided into a fixed number of partitions of different sizes, which may suit the program sizes. Each partition accommodates exactly one process. When a program needs, it is loaded into a partition that one big enough to accommodate the program. Sometimes, some space may be left unoccupied in a partition after loading a program, which is not used by another program that space is wasted and it is known as **Internal Fragmentation**. The degree of multiprogramming is fixed since the number of partitions is fixed is the drawback of this partitioning.

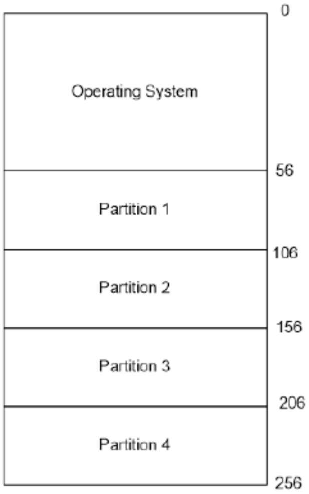


Figure: Multiple Fixed Partitioned Main Memory

In this technique, internal fragmentation is less as compared to the single partition technique. Consider an example, where a program of size 50 MB arrives. In this case, there will be no internal fragmentation. However, if a program of size 40 MB or 30MB arrives, then there will be an internal fragmentation of 10 MB or 20 MB respectively.

In this technique, external fragmentation may be more. Consider an example, where a process of size 55 MB arrives. Even if four of the partitions(200 MB) are free, but it is not possible to execute a program of 55 MB.

Advantages

- It is possible to execute more than one program at a time. In other words, multiprogramming is possible.
- Internal fragmentation is less as compared to a single partition.
- It is simple to maintain.

- The degree of multiprogramming depends upon the number of partitions.

Disadvantages

- It limits the size of the program, the user can execute.
- External fragmentation is high.
- It is not possible to execute a program having its size greater than the size of the partition.

Multiple Partitions with Variable Size

This scheme is free from the limitation encountered in the case of fixed partitioning. In this, the entire available memory is treated as a single partition. All programs, requesting memory are store in a waiting queue and loaded only when a free partition available which is big enough to occupy that program. When a program is allocated a space exactly equal to its size, the balance unoccupied space is treated as another free partition. When a free partition is too small to accommodate any program, it is called **External Fragmentation.**

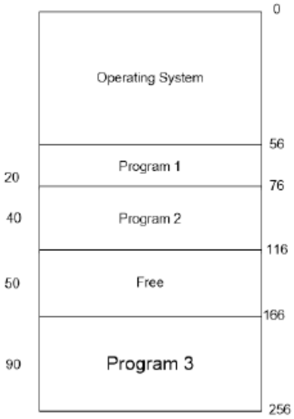


Figure: Multiple Variable Size Partitioned Main Memory

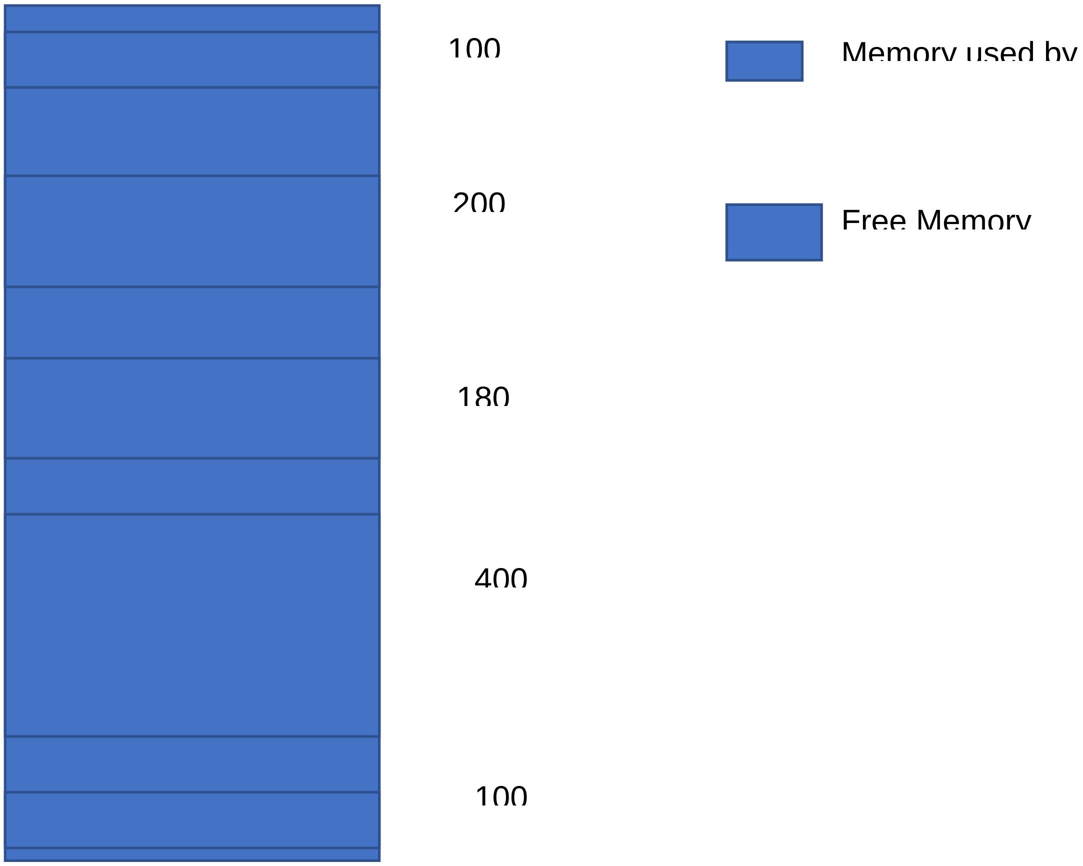
Three strategies can be used to allocate the memory to a process.

1. **First Fit:** According to this strategy, the O/S chooses the very first partition available, whose size is equal to or greater than the size of the process.
2. **Best Fit:** According to this strategy, the O/S chooses a partition that is smallest and whose size is greater than or equal to the size of the process. It may lead to the formation of small-sized unusable holes of memory.
3. **Worst Fit:** According to this strategy, the O/S chooses the largest partition and allocates it to the process. It may result in bigger-sized unusable holes of memory.

From the point of view of the size of the unusable holes, the first fit and best fit are better than the worst fit. From the point of view of time taken for searching the partition, the first fit is better than the other two i.e. Best fit and Worst fit

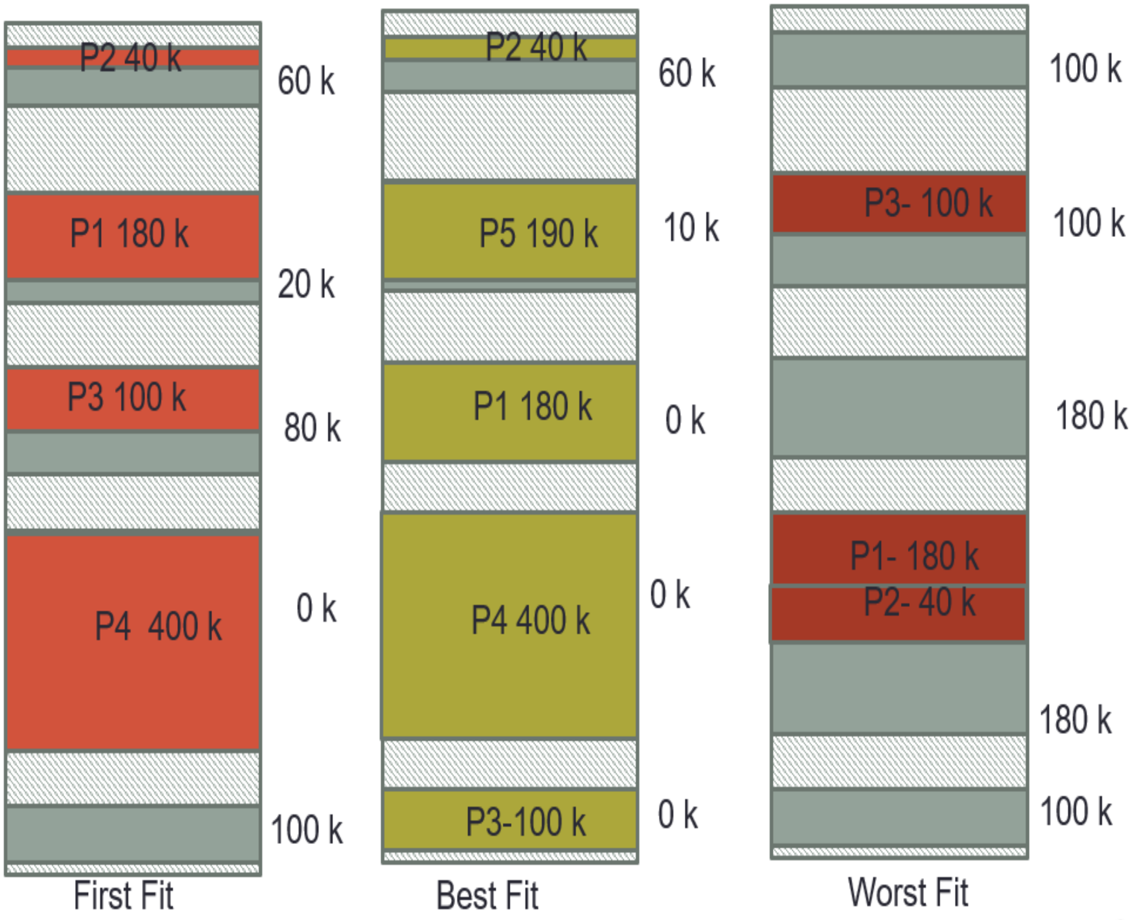
Example of Best Fit

Consider the requests from processes in given order 180K, 40K, 100K,400k, and 190K. Let there be four blocks of memory available of size 100K, 200k, 180K, 400k followed by a block size 100K.



Let there are three process P1, P2, P3, P4, and P5 of sizes 180, 40, 100, 400, and 190 respectively

By using an algorithm of First Fit, Best Fit & Worst Fit is shown by the figure below:



Total free 260 k but process 190k not loaded as memory is not in contiguous

Total free 70 k all process allocated in memory

Total free 660 k but process P4- 400 k and P5- 190k not loaded as memory is not in contiguous

Compaction

Compaction is a technique to deal with the problem of external fragmentation. Compaction means to move the processes in such a way that scattered pieces of unused (free) memory blocks can be placed together so that they can be used by any other process.

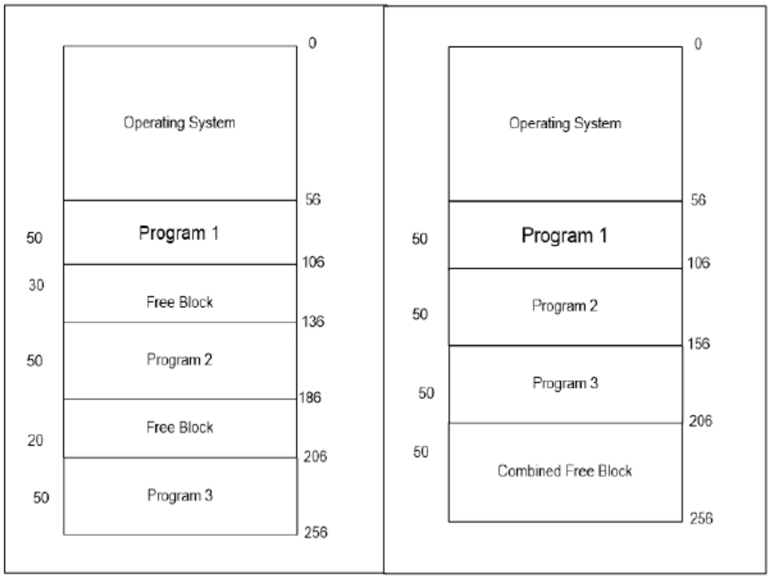


Figure: Before Compaction and After Compaction

Compaction involves the dynamic relocation of a program. This is done by using the relocation register.

However, the compaction algorithm is costly. The problem with the compaction algorithm is that it needs the dynamic relocation of addresses at the execution time of the process. So, the compaction is not possible if the relocation is done statically.

The cost of the compaction increases with the increase in the number of processes to be moved. Moreover, there is a need for a strategy to decide the direction (whether to move upwards or downwards) to move the processes.

Advantages

- Internal fragmentation is very less.
- The degree of multiprogramming can vary.

Disadvantages

- The O/S has to decide about the partition size, almost whenever a program arrives for execution.
- External fragmentation exists.

Memory Protection Contiguous Memory Allocation

When a logical address is generated by the CPU, then it is compared with the content of Limit Register (M). If the logical address is less than equal to M, then it is a valid address else it is an invalid address error, and the process is terminated. If the logical address is valid then the corresponding physical address is computed by adding the content of Relocation Register (B) i.e. base address to the logical address.

Physical Address = Logical Address L + Relocation value R

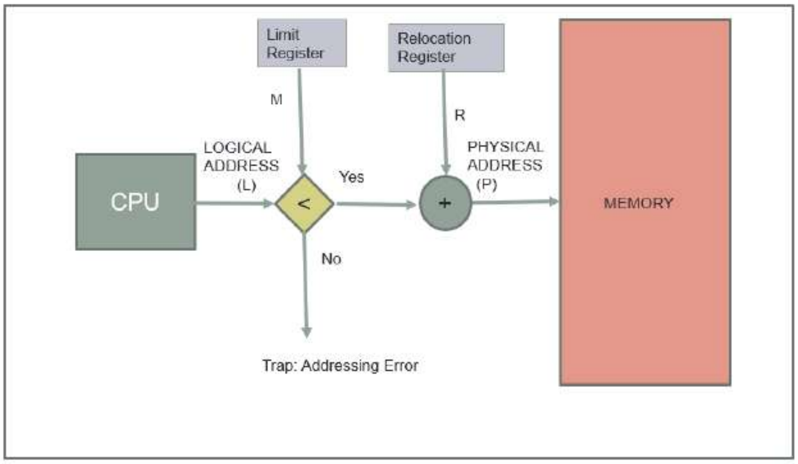


Figure: Memory Protection in Contiguous Memory Allocation

Non-Contiguous Memory Allocation Methods

Non-contiguous memory allocation involves a complex implementation and involves additional costs in terms of memory and processing.

The non-contiguous memory allocation method includes the various strategies like:

1. Paging
2. Segmentation
3. Paging with Segmentation

Paging

Many solutions have been suggested to handle the external fragmentation problem of the memory during the process execution. One scheme permits the logical address space of a process to be noncontiguous, thus allowing a process to be allocating physical memory wherever the latter is available. Memory paging implements this scheme. In the memory paging technique, the logical address space is divided into fixed-sized blocks known as **pages**. The physical address space is also divided into fixed-size blocks known as **frames**. A page is mapped into a frame. Individual pages and frames are recognized by a unique number known as page number and frame number respectively. The size of a page is equal to the size of a frame. A page number forms a part of the logical address and a frame number forms a part of a physical address.

Both page and frame size is usually a power of 2 and depends on the hardware. If a logical address consists of P number of digits, this means there are 2^p addresses in the logical address space. Similarly, if physical addresses consist of q number of digits, this means there are 2^q addresses in the physical address space. **It is always power by 2 because this is to divide logical address into a page number and page offset, easily.**

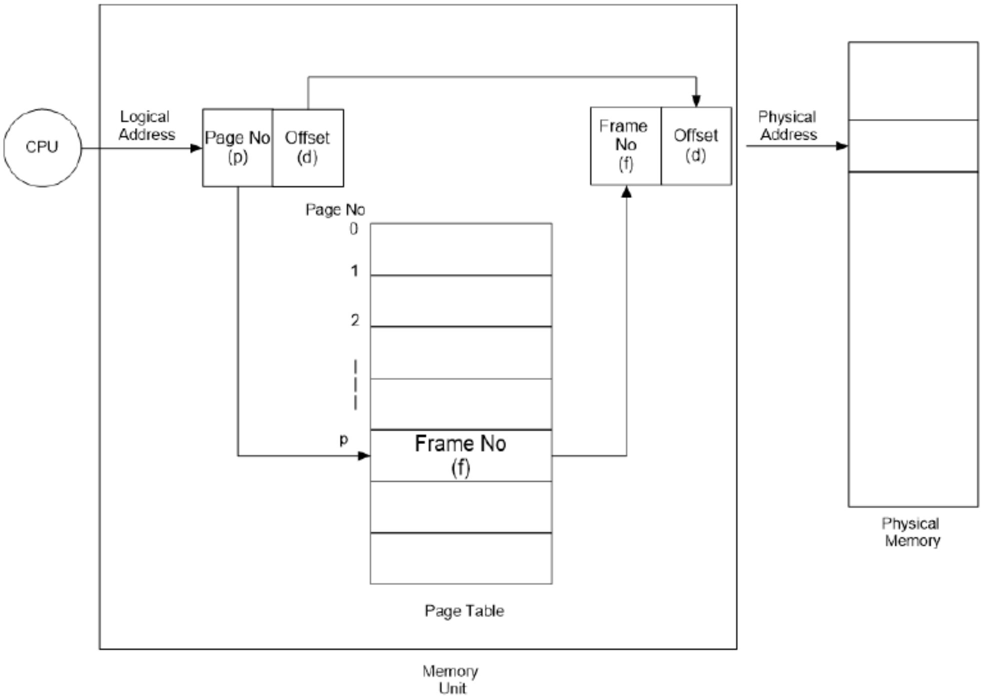


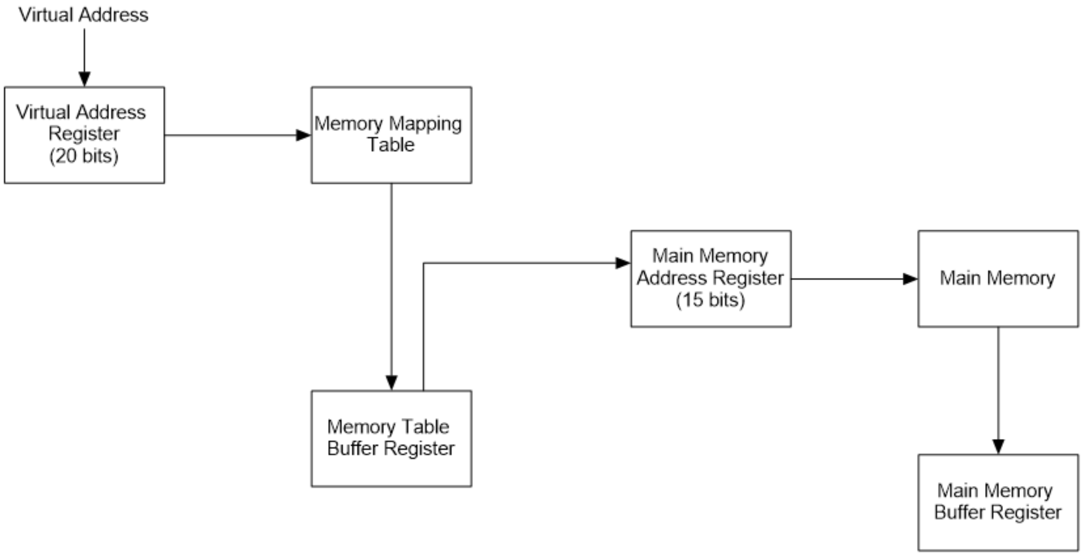
Figure: Paging Hardware

Mapping of Pages to Frame

A page table is maintained for mapping. It divides into two parts: Page Number & Page offset.

p	d
Page No	Offset

A logical address consists of two parts: Page number and Offset. If a logical address is P digits long and page size is 2^{n} digits long, then the higher or leftmost p-q digits in a logical address denote the page number, and the rest of the logical address denotes offset.


Virtual Address
Virtual Address Register (20 bits)
Memory Mapping Table
Main Memory Address Register (15 bits)
Main Memory
Memory Table Buffer Register
Main Memory Buffer Register

Memory Table for Mapping a Logical Address

For example, the logical address space can hold 2^{13} addresses. This means a logical address is 13 bits long. In other words, the logical address space is 8 KB. There are 8 pages in the logical address space. This means the size of each page is 2^{10} bits=1 KB. The size of the physical address space is 2^{12} bits= 4 KB. This means a physical address is 12 bit long. In the logical address of 13 bits the higher 13 bits (13-10) =3 bits denote the page number and the rest 10 bits denote the offsets. The logical address 011 1010101010 denotes page number 011 and the offset 1010101010.

A physical address consists of two parts: frame number and offset.

f	d
Frame No	Offset

The offset is the same as it is in the corresponding logical address. For example, in the previous example, the offset 1010101010 of the logical address 011 1010101010 is 10 bits long. The physical address corresponding to this logical address can be 01 1010101010, where 01 is the frame number and 1010101010 is the offset.

The operating system maintains a table to convert a logical address into its corresponding physical address. This table is called a **memory page table**. The method of converting a logical address into its corresponding physical address is known as **address translation**.

There are two fields in the page table. The first field stores the corresponding frame number of a page. For example, page 3 is stored in the 13th frame. In the first field of the page table value of the 4th cell will be 13. The second field stores a binary value either 0 or 1 and is known as the **presence bit**. If there exists a frame corresponding to a page, the value of the cell is marked as 1 otherwise the value is set to 0.

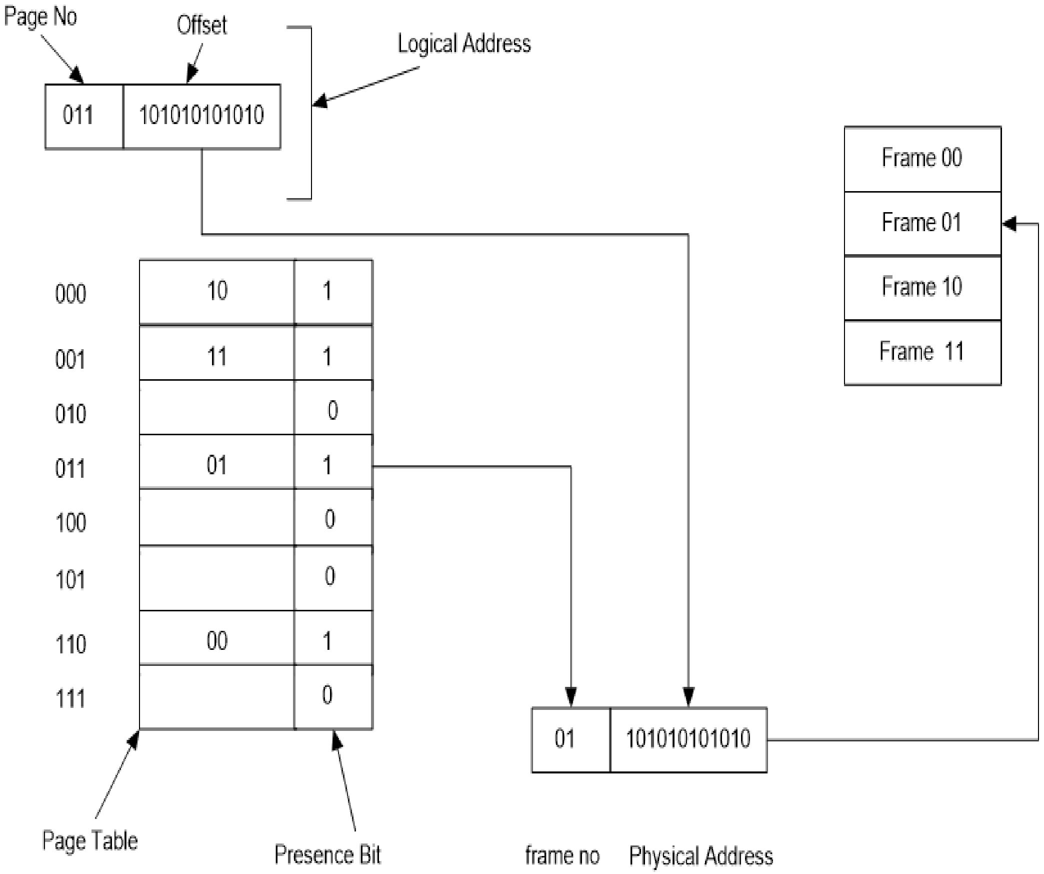
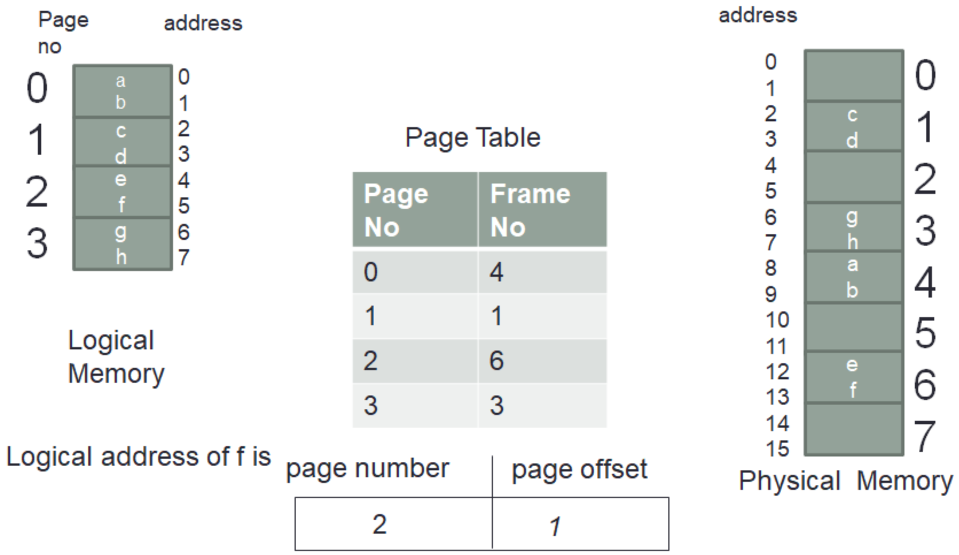


Figure: Mapping

Address Translation in a paging Scheme

The logical address 011 1010101010 is mapped into memory. From the page table, the operating system verifies the presence bit of cell 011, which is 1. This indicates that page 011 has been stored in a frame in the main memory.

Another example Using a page size of 2 bytes and a physical memory of 16 bytes (8 pages), we show how the user’s view of memory can be mapped into physical memory.



Page Table

Page No	Frame No
0	4
1	1
2	6
3	3

page number	page offset
2	1

so physical address calculated (frame no x size)+ offset

The physical address of ‘f’ calculated: ‘f’ stores on page no 2 and offset is 1. page no 2 is loaded in frame 6.

So address is (6x2)+1 i.e. 13.

In a paging scheme, an operating system maintains a list of free frames. This is also known as **free frame pool, or chain or list**. Any frame that is not allocated to any process is known as a free frame. Whenever a page is released from memory after a process is terminated or aborted the page is added to the free list. When an operating system allocated pages to a process for storing instruction or data, it assigns a free page from the **free frames list**.

Structure of Page Table

Hierarchical Page Tables

With the increasing need for larger applications, there is also a need for larger memory space in a computer. As a result, it becomes difficult to work with long memory addresses. In paged memory system the page table itself becomes very lengthy. The solution is hierarchical paging.

In this, the page table itself is paged. There are two-page tables: outer and inner.

The inner page table is similar to that of a single-page table scenario. The outer page table stores the links for the inner page table. The logical address consists of three parts: outer page table entry, inner page table entry, and offset.

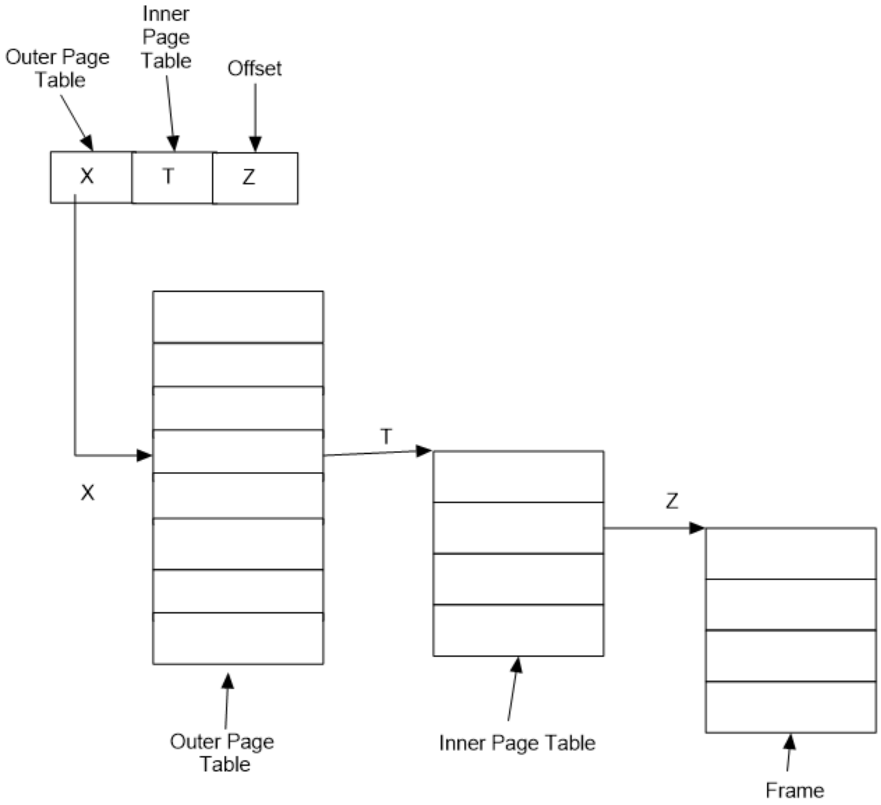


Figure: Hierarchical Page Table

In the process of hierarchical paging, a logical address has been divided into three parts: X, Y, and Z. the symbol X represents the entry in the outer page table, and Y represents the entry in the inner page table, Z represents the offset. The Xth position in the outer page table store the address of the inner page table associated with X. The Y position is searched in that inner page table and Y contains the address of the frame containing the searched page.

Hashed Page Table

This page table is created of length M. Whenever, logical address of generated, a hashing function is applied to the page Number p, to generate an index value i.

i = p % M;

The index value i is used to indexing into the page table. Each entry in the page table is a pointer to a link list. It will provide a mapping between page number p and the corresponding frame number f. It accessed through the index value i will be traversed, till a match forms for page number p and the corresponding frame number f is obtained.

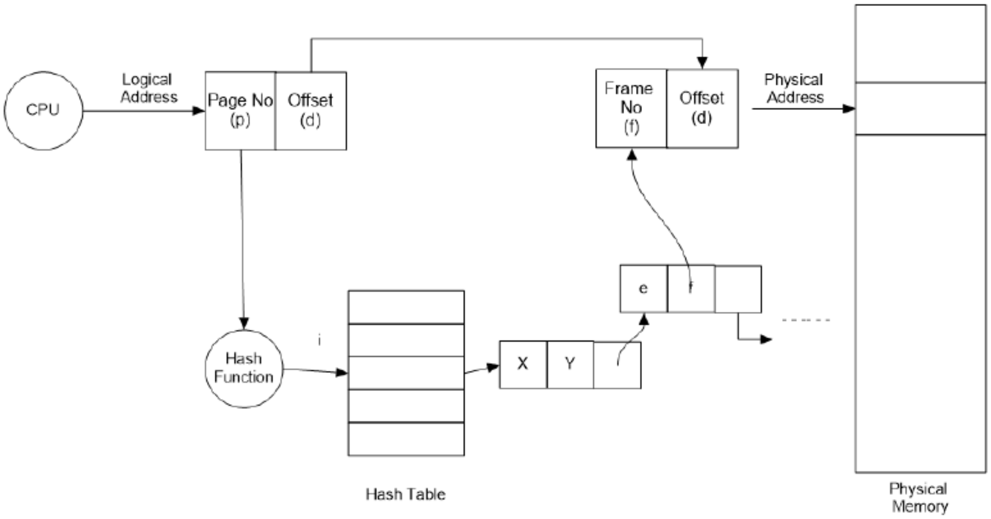


Figure Hashed Page Table

Inverted Page Table (IPT)

An inverted page table is similar to a simple page table along with another entry in the table i.e. process id along with page number. IPT is not process-specific and it does not need switching during content switching. A logical address generated by CPU contains:

Process id (P id),
Page number (p), and
Page offset (d).

Process id & page number is found and corresponding offset (d) which gives the frame number (f), where the desired page is residing. The frame number (f) combined with the offset d, gives the intended physical address.

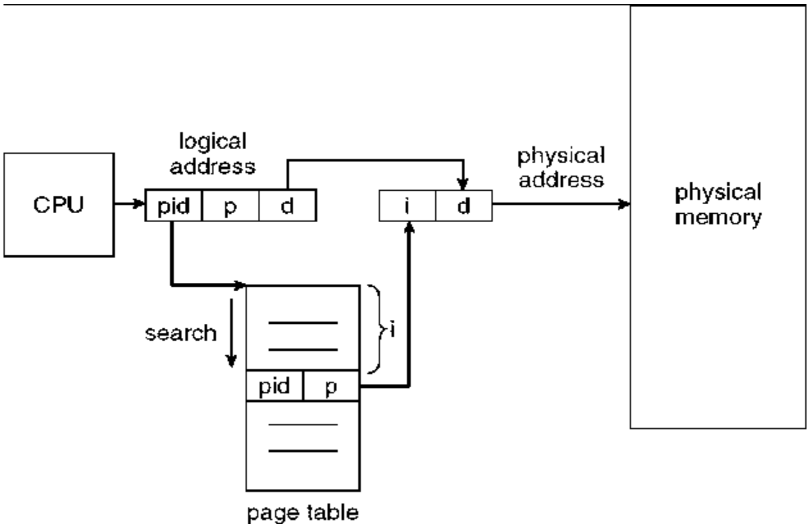


Figure: Inverted Page Table

Translation Look Aside Buffer (TLB)

The page table is implemented in the hardware can enhance the performance substantially. However, it does increase the cost proportionally. Though several schemes of a hardware implementation of the page table exist, in the simplest case, the page table is implemented as a set of dedicated registers. Care must be taken to employ very high-speed logic for these register otherwise the performance will not be up to the mark. The CPU dispatcher reloads these registers, just as it reloads the other registers. Instruction to load is modified the page-table register are, of course, privileged, so that only the operating system can change the memory map. This architecture is used in the DEC PDP-11 computers. It has a 16-bit address while the page size is 8K. As a result, the page consists of 8 entries that are kept in fast registers.

Having a dedicated set of registers for the implementation of the page table has a serious limitation. The register can be used for the purpose only when the page table is reasonably small. Modern computers are capable of having page tables containing as many as 1 million entries. Register implementation in such a system is grossly infeasible. The modern practice is to implement the page into the main memory rather than in the register. The page table is maintained in the main memory and is accessed by a pointer stored in the base register PTBR (Page Table Base Register). One advantage of this practice is that changing the page table requires only changing the register value thus avoiding time-consuming context switching.

This scheme is not without its downside. It suffers from memory access delay. With this approach, a memory location is accessed indirectly. First, the logical memory is indexed into the page table. The page table itself is accessed using the content of the PTBR register which provides the frame number, which is combined with the page offset to produce the actual address.

This approach requires two accesses to memory – one for the page table and one for the index into the page table thereby reducing the CPU time it may become unacceptable. The process can be accelerated using high-speed associative memory or a set of translation look-aside buffer (TLB_S). In associative memory, each register memory each register contains two entries – one for key and another for a value. During a search, the keys are matched with the search value and the corresponding value field is the result if the key was found. The use of associative memory enhances the search speed though at a price for the additional hardware support.

Even associative memory is not unlimited. Therefore, only a few entries of the page table are stored in the associative memory. When the CPU generates a logical address while executing a process the page number of the generated address is searched into the current associative memory for where the frame number is acquired. However, when the page number being searched does not exist in the associative memory the same is searched into the memory and the associative memory is updated with this page number so that it may be found in the next reference top the associative memory.

In this arrangement, a page being searched may not always be found in the associative registers. The extent to which a reference page is found in the associative memory is expressed in terms of hit ratio. Hit ratio is the percentage of times that a page number being searched is found in the memory. For instance, a hit ratio of 60% indicates that out of 100 times a page being searched it is found in the associative memory 60 times. Therefore, one of the performance goals of an operating system is to increase this hit ratio.

The time delay in memory access can be easily estimated in the following manner. Assuming that it takes 10 nanoseconds to search the associative registers, and 50 nanoseconds to access memory, the mapped memory access would take 10 + 50 = 60 ns provided the page number is found in the associative registers.

However, if the search ends in a miss in the associative register after spending 10 ns, then memory access is made for the page table and frame number taking additional 50 ns. Following this, the desired byte is accessed in the memory costing another 50 ns. The total time elapsed comes out to be 50 + 50 + 10=110 ns.

Example: Assuming a hit ratio of 60%, the effective or expected memory-access time can be calculated as shown here under:

Effective access time = (Probability of hit) * Access Time on hit +

(Probability of miss) * Access Time on miss

= 60% of 60 + 40% of 110

= 80 nanosecond

One way of increasing the hit ratio is to use more associative registers. However, as mentioned earlier it can be very costly. Therefore, usually, a trade-off is considered in most cases. One estimate indicates that a hit ratio of 80% to 90% can be achieved with 16 to 12 associative registers.

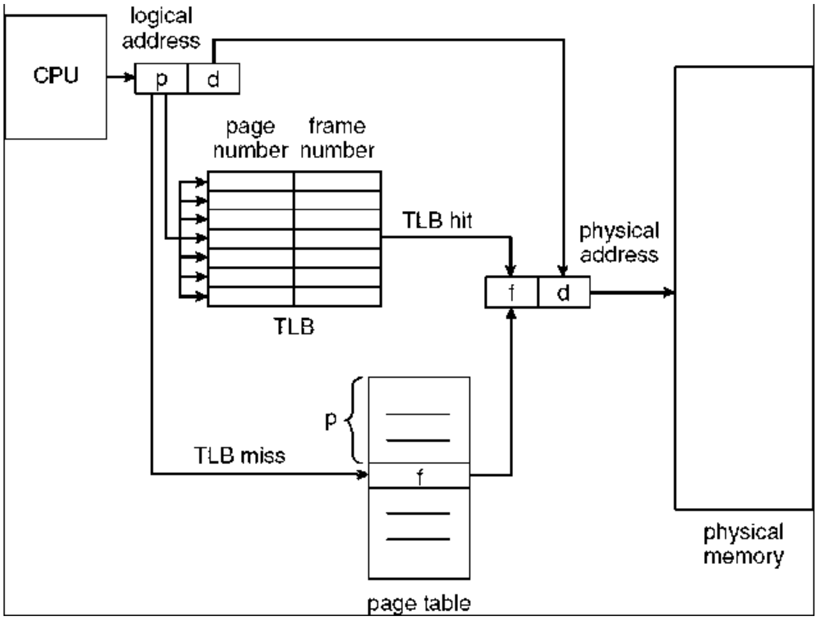


Figure: Paging with TLB

Protection

The issue of memory protection in a paging system is handled by attaching a few bits to each of the frames in the page table. The bit may indicate read-only or read-write or execute-only behaviors of the frame. At the time of accessing the frame number from the page table, the corresponding bits are also examined.

Here if a process attempts to write into a read-only frame a system interrupt is generated by the hardware. This interrupt is caught by the operating system, which in turn takes the memory protection violation action.

Notes

In addition to the protection bits, one more bit attached to each frame. This bit determines whether the frame/page is valid or invalid.

The "valid" status of the bit indicates that the associated page in the process's logical address space, and is thus a legal page. Otherwise, the page is not in the process's logical address space. Illegal addresses are trapped by the operating system using the valid-invalid bit. The operating system controls the accessibility of a frame by setting this bit.

Page A
Page B
Page C
Page D
Page E
Page F

Logical Memory

Page	Page No.	Valid/ Invalid Bit
A		I
B	3	V
C	7	V
D		I
E	1	V
F		I

Page Table

0	
1	E
2	
3	B
4	
5	
6	
7	C

Main Memory

Figure: Paging Protection

Memory Segmentation

Despite all the obvious merits of a memory-paging scheme of memory management, it does not reflect the way a programmer would like to view the memory. In a paging memory system, there is a clear-cut distinction between the users and system views of memory. The user's view of memory is not the same as the actual physical memory. The user's view is translated onto physical memory by the operating system with the use of special hardware. It is due to this translation that the logical view is different from the physical view of the memory.

A programmer would not like to think that the memory is simply a larger array of bytes wherein at someplace data is stored and at another executable code is stored. The most preferred view of the majority of programmers is that a program is divided into different sections or segments. If you have programmed in COBOL you would appreciate that a COBOL program has different divisions each having one or more optional sections.

Similarly, in a C program, the code is divided between many functions. Your program may have any number of functions, subroutines, data structures, and several modules. Each of these modules or data elements is assigned a unique name by the programmer. Where and how these elements are loaded in the memory is not the concern of the programmer. The programmer does care as to whether the segment containing symbols table is stored in the memory before the main function or after it. Each of these segments must be of variable length. The element of the program should be addressable by the respective names assigned to them.

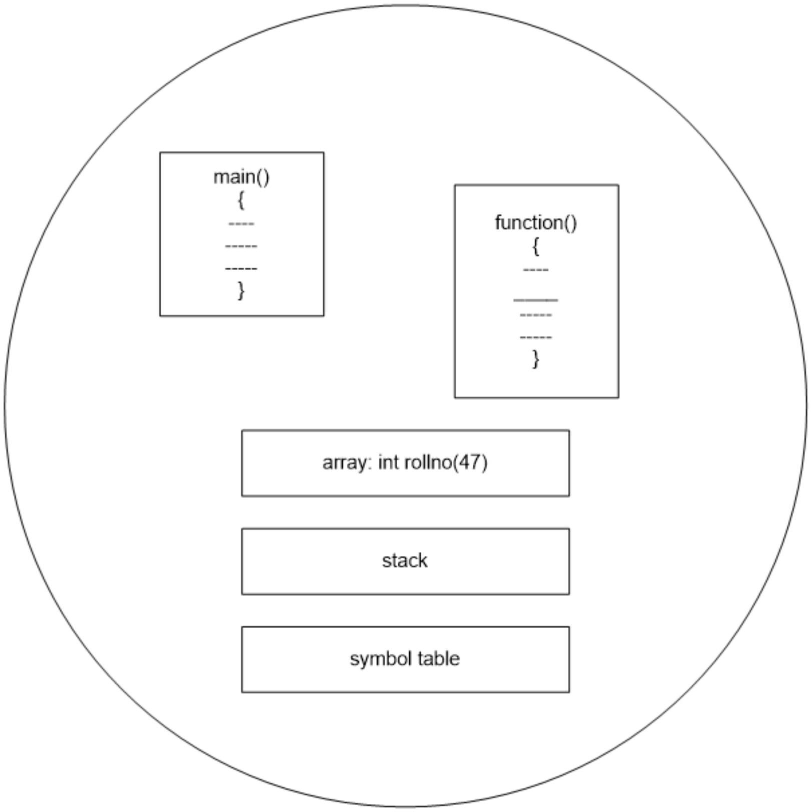


Figure: User's view of a program

An operating system may adopt a memory management scheme to simulate this view of memory.

Segmentation is a technique of non-contiguous memory management technique that employs this scheme. In this scheme, the logical memory space is divided into certain unequal size chunks known as **segments**. According to segmentation, an application is loaded into memory as a collection of modules. In other words, logically related instruction and data items are grouped and loaded into a segment.

For example, an application can consist of five modules: the main program, function A, function B, the stack for storing data when a function is called, and shared data for all modules such as a global variable. Each module is loaded in a distinct segment in the main memory.

Accessing Addresses in Segmentation:

In a segmented memory, a logical address consists of two parts: segment number and offset.

s	d
Segment No	Offset

Figure Address of Segment

An offset is a location within a segment, where any particular data is stored. For example, the logical address 3: 10000 indicates offset 10000 and segment number 3. A segment table is used for the validation of an address. A segment table has two fields: base address of the segment, and size of the segment. The base address of a segment is the location of a particular segment memory and its corresponding segment table:

Segment No	Base Address of the Segment	Size of the Segment
0	80000	6000
1	10000	10000
2	20000	15000
3	40000	35000
4	90000	5000

▨ Free Space

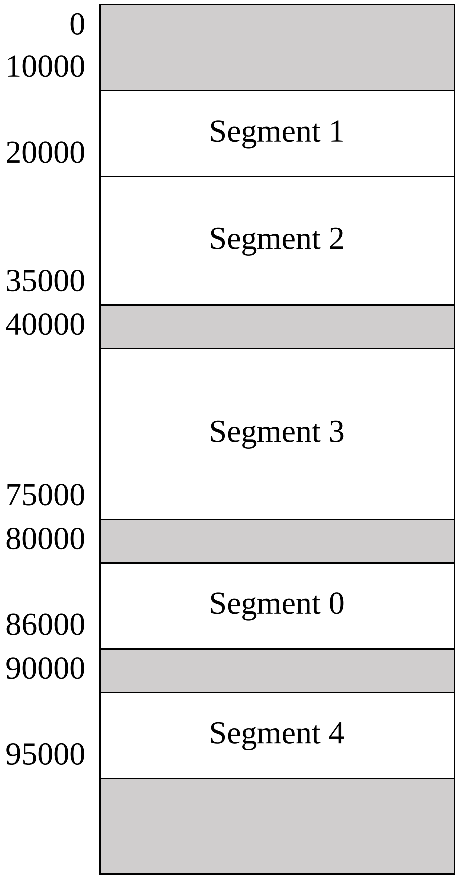


Figure: Segment Table

Segment and Segment Table

The above figure segment table elaborates various memory segments in the main memory and their corresponding entries in the segment table. Segments need not be loaded sequentially or contiguously. For example, segment 2 has been loaded in a lower position in the memory than segment 0.

The starting address and size of a segment are stored in the segment table. For example, the base or starting address of segment 2 is 20000, and is the size of segment 2 is 15000 bytes.

When accessing an address in a segmented memory, the segment table is used for address validation. The logical address 2: 12000 indicates the location 12000 starting from the base of segment 2. Segment 2 starts from location 20000. The size of segment 2 is 15000 bytes. In other words, range of the segment 2 is 20000 to 74FFF. The physical address corresponding to the logical address 2: 12000 is 20000 + 12000=32000. the address 32000 is within the range of segment 2 (i.e. 20000 to 74FFF as 32000<74FFF). Hence, the logical address 2: 12000 is a valid address.

Implementation of Segment Tables

As in the case of the paging system, segmentation can be implemented in a variety of ways. The segment table can reside either in the register or in fast memory. The advantage of register implementation is that both the limits checking and referencing can be done extremely rapidly and simultaneously.

The feasibility of register implementation depends on the number of segments a program has. The larger the number the more register are required making the proposition infeasible. In smaller segmentation register implementation may be recommended. However, modern programs tend to become very large causing a large number of segments. Therefore, in such a system it is more appropriate to maintain the segment table in the memory rather than in the registers. The data structure used for implementation is often a **Segment-Table Base Register or STBR**. This pointer points to the segment table located in the memory. Note that the number of segments used by a program varies widely. Therefore, a Segment-Table Length Register or STLR is used. The address translation for a given logical address (s, d) proceeds as follows:

1. Check that the segment number is legal (i.e. S< STLR)
2. Add the segment number to the STBR(i.e., STBR+S)
3. Read entry from memory at STBR + S
4. Check the offset against the segment length (d< limit)
   (a) if true then compute the physical address of the desired bytes as S+d
   (b) else raise trap to the OS

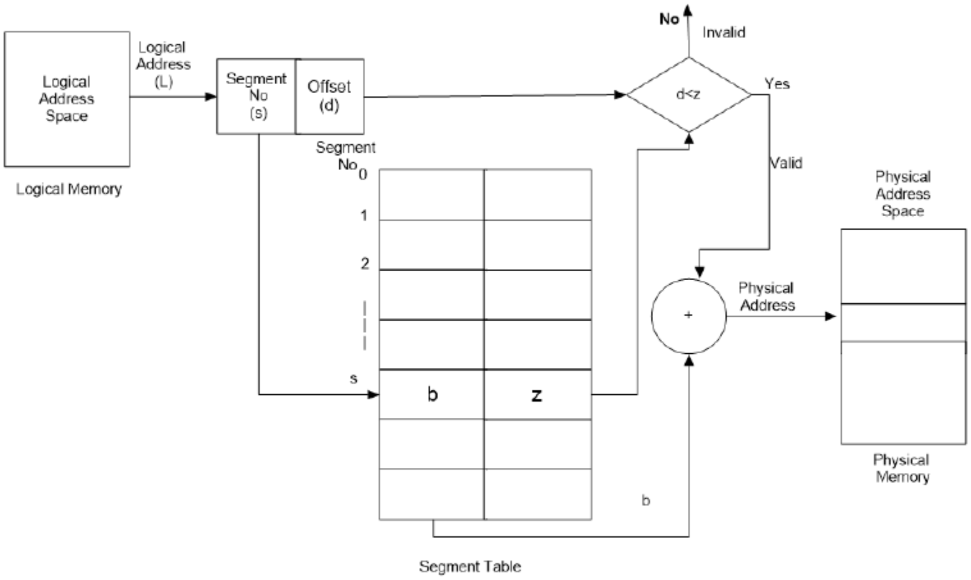


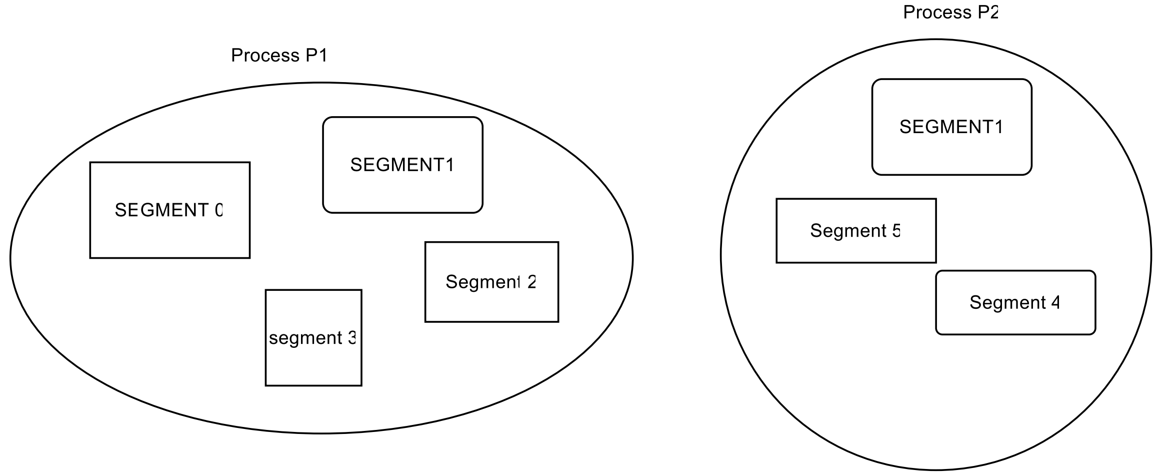
Figure: Segmentation Hardware

Both the schemes- paging and segmentation requires two memory references during address translation. This slows down the OS effectively by a factor of 2. As in the case of paging, a set of an associative registers can be used to speed up the operation. It has been found that a small set of associative registers generally reduces the time required for memory accesses.

Protection and Sharing

Segmentation enforces protection intuitively. Since each segment is a semantically cohesive unit all the corresponding entries are likely used similarly. Therefore, one needs to protect at the segment level using a few protection bits. For instance, since codes do not rewrite themselves the code segment can be assigned a read-only status whereas the data segment can be allowed to perform read and write operations. The memory mapping hardware will check the protection bit associated with each segment table entry to prevent illegal access to memory, such as attempts to write into a read-only segment or to use an execute-only segment as data. For example, by placing an array in its segment, the memory-management hardware will automatically check that array indexes are legal and do not stray outside the array boundaries. This enables the system to take corrective measures early on if a memory protection error does occur.

The main advantage of segmentation is that it can be shared between processes. This is true, particularly with code segments. For each process, a segment table is associated, which the dispatcher uses to define the hardware segment table when this process can share the same code segment as shown in the figure.



Segment Table P1

Segment No	Base Address of the Segment	Size of the Segment
0	6000	500
1	4000	200
2	8000	600
3	5000	400

Segment Table P2

Segment No	Base Address of the Segment	Size of the Segment
1	4000	200
4	8600	200
5	8800	200

▨

Free Space

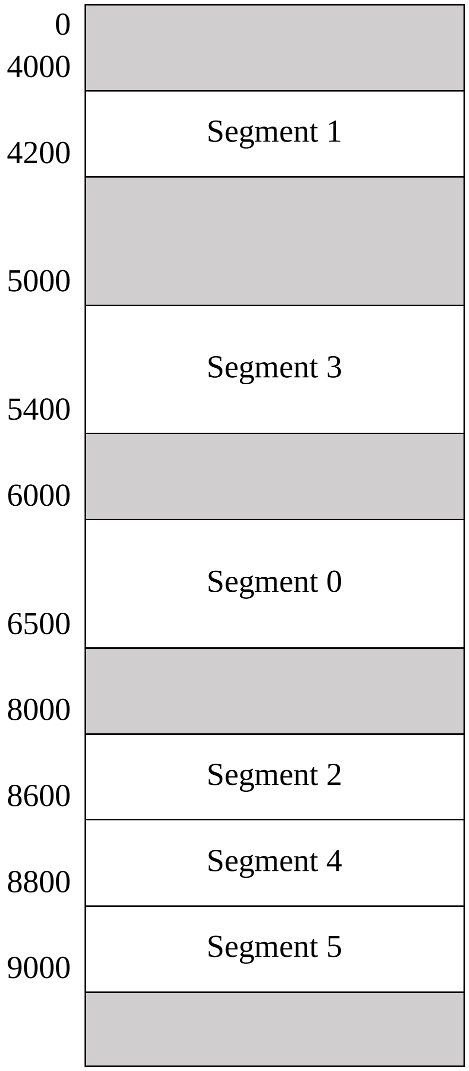


Figure: Segmentation Sharing

Here Segment 1 is shared between P1 & P2

Sharing of the segment in a segment memory system

Here in this example, segment (1) is being shared between the two processes p1 and p2. Simply by managing entries in the segment table of processes a segment can be shared among them. The sharing occurs at the segment level. Thus, any information can be shared if it is defined to be segmented. Several segments can be shared in this way.

Segmentation with paging

Segmentation of memory can be implemented with a paging scheme. Unequal size segments can be represented by equal size pages. For example, in a logical address space of 64 KB, there are 32 pages of 2KB each. A program is divided into 3 segments of the size 15 KB, 18 KB, and 10 KB. These three segments require 8, 9, and 5 pages to be loaded into memory. The first segment occupies the memory space of 8 pages of 2 KB each or 16 KB, whereas its requirement is 15 KB memory space. Thus internal fragmentation of 1 KB occurs.

In segmentation with a paging scheme two tables are used: a segment table and a page table. The segment table has two fields: base address and size. The base address of a segment is the address of the first page in the segment. The size of a segment is the number of pages a segment occupies. The page table stores the addresses of the frame where a page is loaded into the main memory. For example, the value of the 15th cell of the page table is 38000. This means page 15 has been loaded into the frame located at the address 38000 in the main memory.

A logical address has three parts: segment number, page number, and offset. The logical address 4:3: 9E indicates the address 9E on page number 3 of segment number 4.

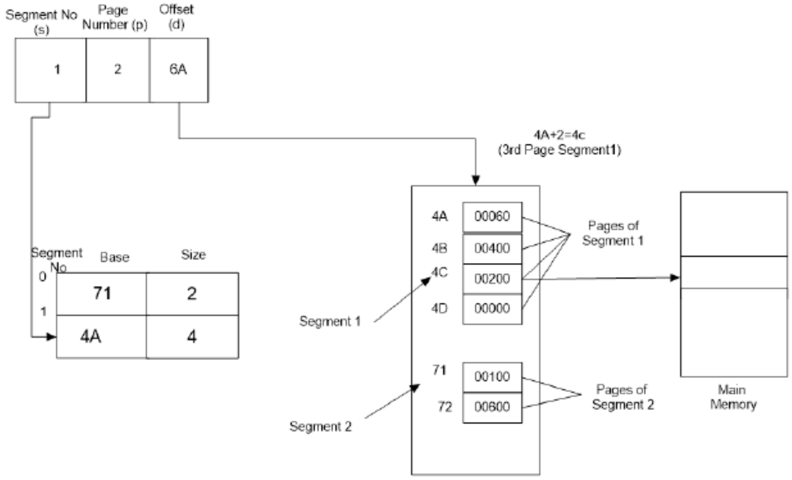


Figure: Segmentation with paging

The main memory is divided into certain equal size frames of 256 bytes (decimal 256 = 100 in hexadecimal) each. The address 6A of page number 2 of segment number 1 is being located. In the segment table, we found that the address of the first page in the page table of segment 1 is 4A. Segment 1 is consists of 4 pages.

From the page table we see that address of the frame corresponding to page 2 has been stored in the location 4C in the page table (because 4a + 2 = 4C). The cell number 4C of the page table is loaded in the frame located at the address 00200 in the main memory.

Shared Pages

In multiprogramming, more than one program can be loaded. It is common for many users to be executing the same program. If individual copies of these programs were given to each user, much of the main memory would be wasted. Its solution is to share those pages that can be shared. One copy of read-only code shared among processes (i.e., text editors, compilers, window systems). Shared code must appear in the same location in the logical address space of all processes. Each process keeps a separate copy of the code and data. The pages for the private code and data can appear anywhere in the logical address space.

Example

Let, there are four processes P1, P2, P3, and P4. P1 has four pages lib1, lib2, lib3, and data1. P2 has four pages lib1, lib4, lib5, and data2. P3 has four pages lib1, lib2, lib5, and data3. P4 has four pages lib1, lib2, lib4, and data4. So, to load all processes 16 frames are required. In all these processes some pages are shared. So, if we share those pages and loaded all shared pages once by giving the same frame number, then it needs only 8 pages as shown in the figure below.

lib1	2	lib1	2
lib 2	6	lib 4	12
lib 3	9	lib 5	0
data 1	4	data 2	7
Process P1	Page Table P1	Process P2	Page Table P2

lib1	2	lib1	2
lib 2	6	lib 2	6
lib 5	0	lib 4	12
data 3	10	data 4	14
Process P3	Page Table P3	Process P4	Page Table P4

0	lib 5
1	
2	lib1
3	
4	data 1
5	
6	lib 2
7	data 2
8	
9	lib 3
10	data 3
11	
12	lib 4
13	
14	data 4
15	

Shared Pages among Processes

Figure: Shared Pages

Demand Paging

The basic concept of virtual memory is storing instructions and data of a program in the secondary memory, and then they are loaded in the main memory. In other words, the part of a program that is required at any instant is loaded in the main memory while the rest of the program is in the secondary memory.

Virtual Memory

Virtual memory is useful especially in a multitasking operating system, where memory available in the main memory for user application is divided among several user programs and every program has to compete for its memory requirement.

Virtual memory is a technique that permits the execution of processes, with their code only partially loaded into physical memory. Virtual memory is used for the separation of logical address space available to the user and the actual physical memory. CPU-generated addresses are known as **logical addresses** or **virtual addresses**.

Programmers use virtual addresses in applications. The MMU converts the virtual addresses into the corresponding physical memory address.

Programmers are notified that they can fully utilize the logical address space. Since the logical address space, programmers have the illusion that they have a larger memory space at their disposal.

Virtual memory is implemented using a non-contiguous memory allocation technique known as demand paging. This technique is similar to paging except that in-demand paging, pages are swapped in and out of main memory.

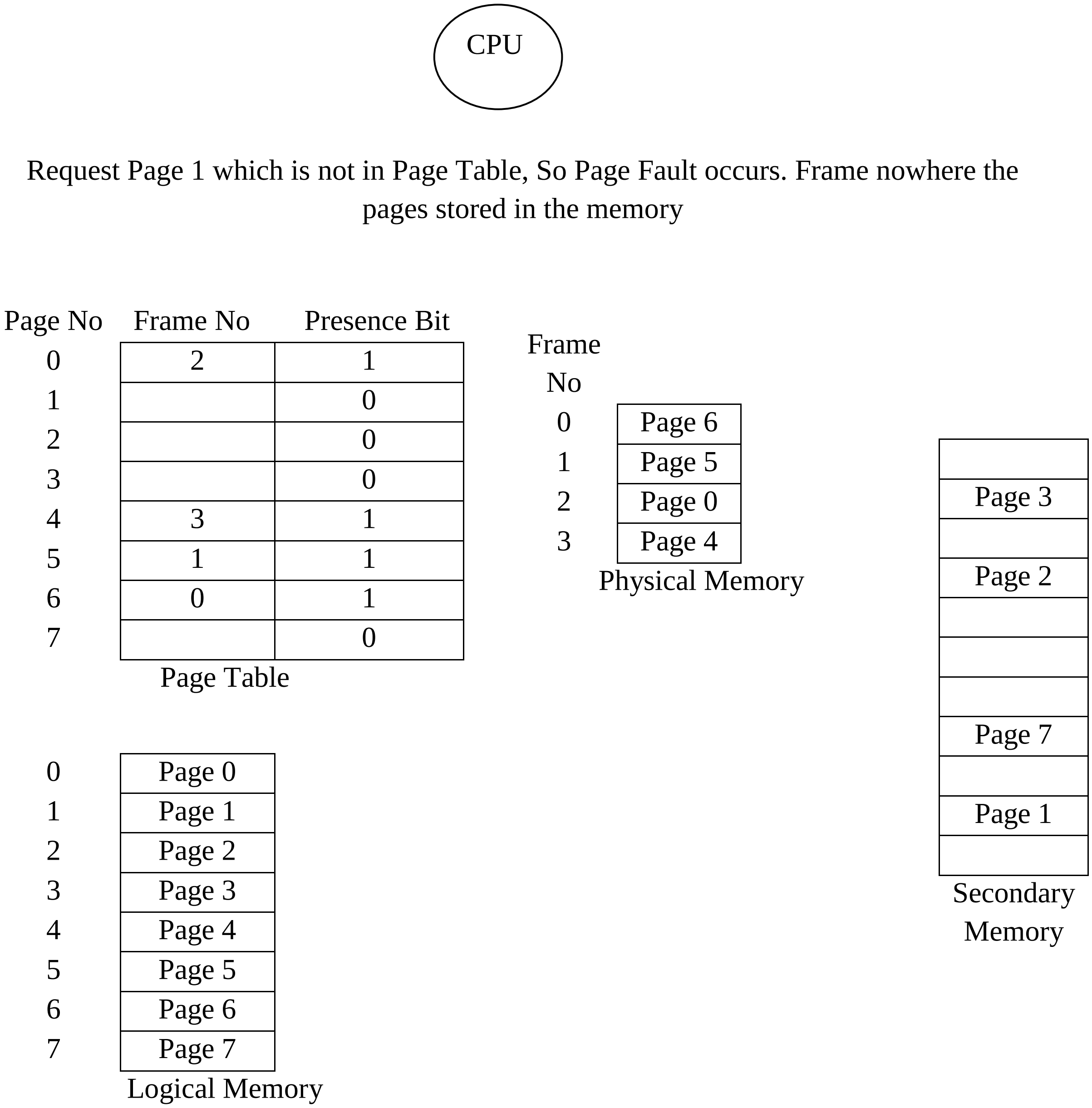
A program is initially stored in the secondary memory. When a page in the program is required, it is swapped into the main memory. This is called demand paging because until a page is not required, it is not loaded.

In demand paging, MMU also maintains an indexed table for address translation, known as a **page table**, similar to that of a simple paging scheme. The page table stores the information about, which page of a program is stored in which frame of the physical memory.

When a page is required, the operating system scans the page table to find out the physical address in the main memory that is in which frame the page is loaded. If the page is found, the operating system continues its processing. In case the page is not present in the main memory, a **page fault** occurs. A page fault is not a fault or an error; rather it indicates a situation that the page is requested by a program that is not currently loaded in the main memory.

When a page fault occurs, MMU swaps in the requested page from the secondary memory into the main memory. In case there is no free space in the main memory, MMU finds free space in the secondary memory, which is also known as the backing store in the context of virtual memory. MMU swaps out an unused page from the main memory and stores the unused page in the free space in the secondary memory for creating space in the main memory. The requested page is loaded into the main memory. Finally, the page table is updated.

Another way is a page table with a valid or invalid bit. The page table includes a valid or invalid bit for each entry. When a page is loaded in memory its frame number is entered and the page validity bit is set to valid. Thus if the bit is set to valid, it indicates that the page is in memory. If the page presence bit is invalid, it indicates that either page does not belong to the logical address space of the process, or it is still not loaded into memory.



Request Page 1 which is not in Page Table, So Page Fault occurs. Frame nowhere the pages stored in the memory

Page No	Frame No	Presence Bit
0	2	1
1		0
2		0
3		0
4	3	1
5	1	1
6	0	1
7		0

Page Table

Frame No	
0	Page 6
1	Page 5
2	Page 0
3	Page 4

Physical Memory

Page 3
Page 2
Page 7
Page 1

Secondary Memory

0	Page 0
1	Page 1
2	Page 2
3	Page 3
4	Page 4
5	Page 5
6	Page 6
7	Page 7

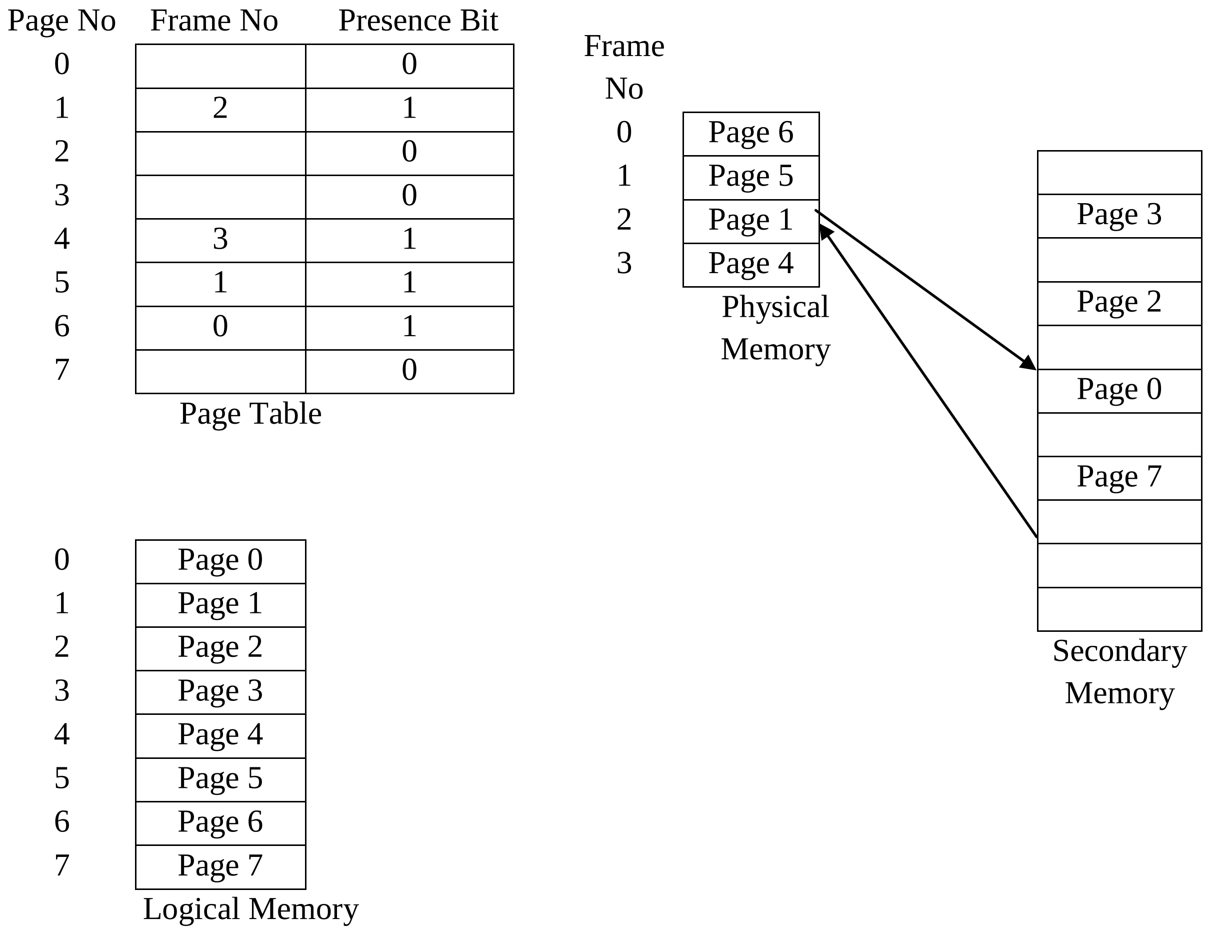
Logical Memory

Figure: Before Page Replacement

Let, CPU request page 1. This page is not present in the page table because the presence bit of page1 in the page table is 0 that indicates the cell is empty. Thus page fault occurs.

When a page fault occurs, the operating system searches and loads that page in the memory replacing an existing page.

Page 1 caused a page fault. Page 0 is swapped out of the main memory and stored in the disk to create room for page 1 in the main memory. Then page 1 is swapped in the main memory.



Page No	Frame No	Presence Bit
0		0
1	2	1
2		0
3		0
4	3	1
5	1	1
6	0	1
7		0

Page Table

Frame No	
0	Page 6
1	Page 5
2	Page 1
3	Page 4

Physical Memory

Page 3
Page 2
Page 0
Page 7

Secondary Memory

0	Page 0
1	Page 1
2	Page 2
3	Page 3
4	Page 4
5	Page 5
6	Page 6
7	Page 7

Logical Memory

Figure: After Page Replacement

In the above figure, an abstract and basic idea of the functionality of virtual memory has been shown. Actual implementation varies in different operating systems.

Advantages

- It uses memory more efficiently.

- There is no limit on the degree of multiprogramming.

Steps in Handling a Page Fault

Initially, memory is empty, if there is a reference of a page, the first reference to that page will trap to an operating system which is page faults. To handle the page fault there are the following steps:

1. Operating system looks at another table to decide whether the demand is valid or not in memory. If it is invalid demand then it is aborted, else it is not in memory.
2. After that it searches the empty frame if it is not available then the victim finds and swaps out.
3. Demanded page is swapped into that frame.
4. Page tables are reset which assign the frame number to the corresponding page number.
5. Set the valid-invalid bit to valid bit.
6. Restart the instruction that caused the page fault.

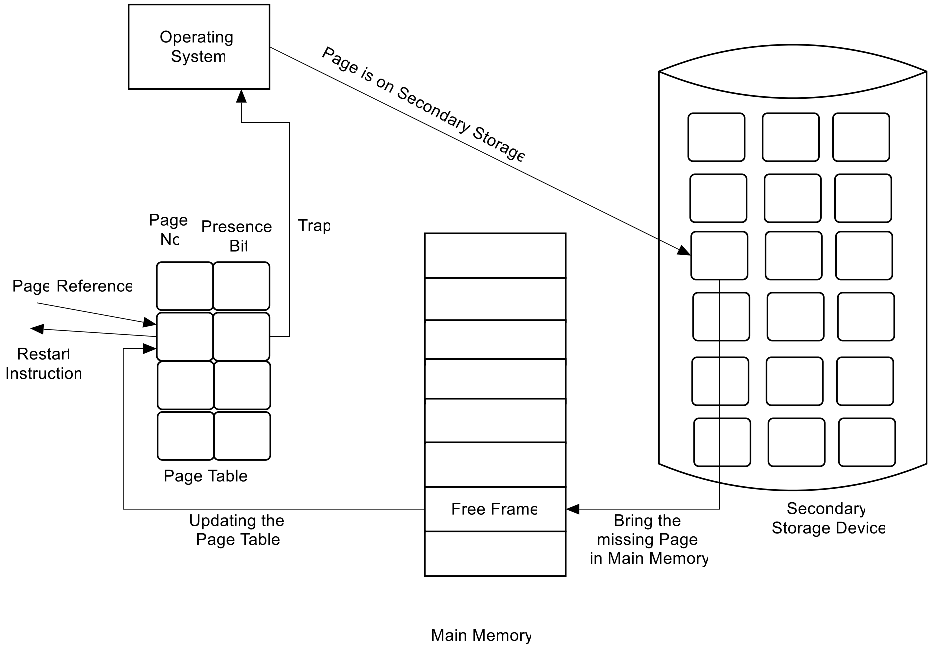


Figure: Steps in Handling Page Fault

Performance of Demand Paging

Demand paging can have a significant effect on the performance of a computer system. Let us calculate effective access time for a demand paged memory. Let P be the probability of a page fault $0 \leq P \leq 1.0$ if $P = 0$ no page faults and if $P = 1$ every reference is a fault.

Effective Access time =

$(1 – p) \times ma + pft$

p = page fault

ma = memory access time

pft = p × (page fault overhead + swap page out + swap page in + restart overhead)

EAT is directly proportional to the page fault rate. If the page fault rate is low then EAT is decreased otherwise.

Example: Let memory access time = 50 nanoseconds

Average page fault service time = 4 ms

Then EAT = $(1 – p) \times ma + pft$

$= 1 – p \times 50 + p \times (4 \text{ ms})$

$= 50 - 50p + 4000000p$

$= 50 + 3999950p$ (nanosecond)

Page Replacement

As the number of processes and the number of pages in the main memory for each process increase, at some point in time, all the page frames become occupied. At this time, if a new page is to be brought in, the OS has to overwrite some existing pages in the memory. The page to be chosen is selected by the page replacement policy.

A **page** is a fixed-length block of memory that is used as a unit of transfer between physical memory and external storage like a disk, and a **page fault** is an interrupt (or exception) to the software raised by the hardware when a program accesses a page that is mapped in address space but not loaded in physical memory. The hardware that detects this situation is the memory management unit in a processor. The exception handling software that handles the page fault is generally part of an operating system. The operating system tries to handle the page fault by making the required page accessible at a location in physical memory or kills the program in case it is illegal to access.

The hardware generates a page fault for page accesses where:

- The page corresponding to the requested address is not loaded in memory.
- The page corresponding to the memory address accessed is loaded, but its present status is not updated in hardware.

Example

Let program A have four pages i.e. P1, P2, P3, P4. Currently, P1, P2, P3 are in memory and it requests page 4 i.e. P4. Program B has four pages i.e. P8, P9, P10, P11.

Currently, P8, P10, P11 are in memory. Only six pages can be stored in memory and currently, P1, P2, P3, P8, P10, P11 are in memory so there is no memory space. To load P4, one page must be swapped out. Let it be P2, P2 store in frame 3 which is swapped out, and P4 is loaded in frame 3 which is currently free. The figure shows memory position before swapping page 4 i.e. P4 and the next figure shows the memory position after swapping page 4.

0	Page 1
1	Page 2
2	Page 3
3	Page 4

Logical Address Space of Program A

Page No	Frame No	Valid/ Invalid Bit
Page 1	4	v
Page 2	3	v
Page 3	6	v
Page 4		i

Page Table of Program A

Operating System	
Page 11	1
Page 8	2
Page 2	3
Page 1	4
Page 10	5
Page 3	6

Secondary Memory

0	Page 8
1	Page 9
2	Page 10
3	Page 11

Logical Address Space of Program B

Page No	Frame No	Valid/ Invalid Bit
Page 8	2	v
Page 9		i
Page 10	5	v
Page 11	1	v

Page Table of Program B

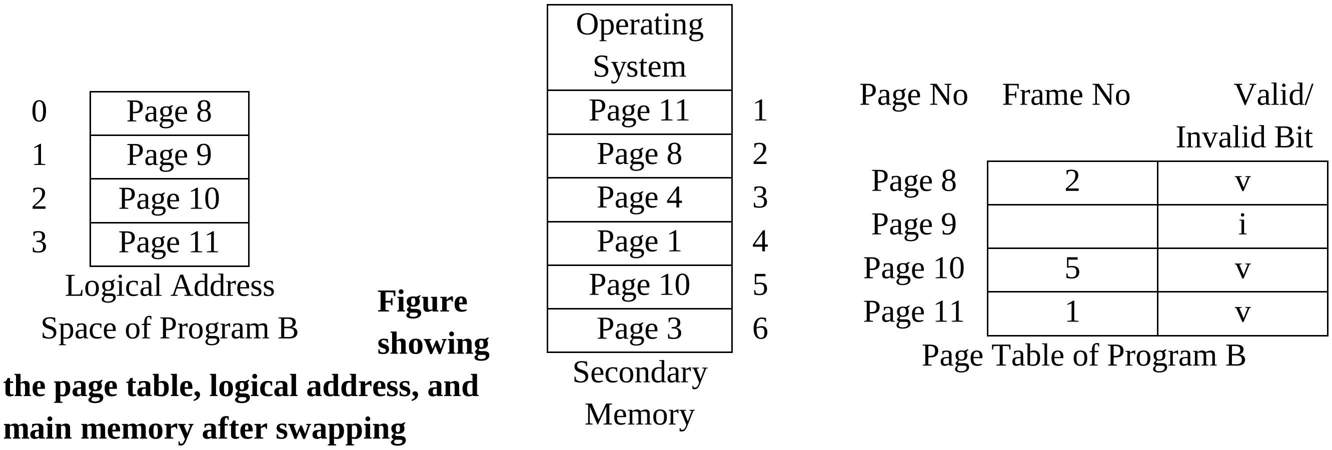
Figure showing the page table, logical address, and main memory before swapping

0	Page 1
1	Page 2
2	Page 3
3	Page 4

Logical Address Space of Program A

Page No	Frame No	Valid/ Invalid Bit
Page 1	4	v
Page 2		i
Page 3	6	v
Page 4	3	v

Page Table of Program A



Page No	Frame No	Valid/ Invalid Bit
Page 8	2	v
Page 9		i
Page 10	5	v
Page 11	1	v

Figure showing the page table, logical address, and main memory after swapping

Page Replacement Algorithms

When a page fault occurs, the operating system has to choose a page to remove from memory to brought a demanded page in memory.

A page replacement algorithm is logic or policy regarding how to select a page to be swapped out from main memory to create space for the page, known as the requested page, which has caused a page fault. These are several page replacement algorithms such as :

- First In First Out (FIFO)
- Optimal Page Replacement Algorithm
- Least Recently Used (LRU)
- Clock Algorithm (Second Chance Algorithm)
- Counting Based Algorithms
- Least Frequently Used (LFU)
- Most Frequently Used (MFU)

First In First Out (FIFO) Algorithm

The FIFO algorithm is the simplest of all the page replacement algorithms. It conveys a basic idea that when a page fault occurs, the oldest page in the main memory is to be swapped out of the main memory to create a room or the required page that needs to be executed. It replaces the page that has been in the memory longest.

> **The oldest page in the main memory is one that should be selected for replacement first. (if the number of frames is 3 then page repeated 3 times will replace)**

For example, let there are four pages i.e. 14, 21, 18, 36. Pages number 14, 21, and 18 are present in the memory and page number 21 was loaded first, followed by 14 and then 18. If page number 36 is required to be accessed by the CPU, it results in a page fault because page number 36 is not present in the main memory. The page loaded first, that is page number 21 is swapped out of the main memory and stored in the secondary memory to create room for page number 36. Then page 36 is loaded in the memory. One possible implementation is a FIFO queue of existing pages in memory. The oldest page will be at the FRONT of the queue. Whenever a page fault occurs, the page at the FRONT of the queue is mode victim and the new page is put at the REAR of the queue.

Advantages

It is very simple and easy to implement.
Programmers can easily code this algorithm.

Disadvantages

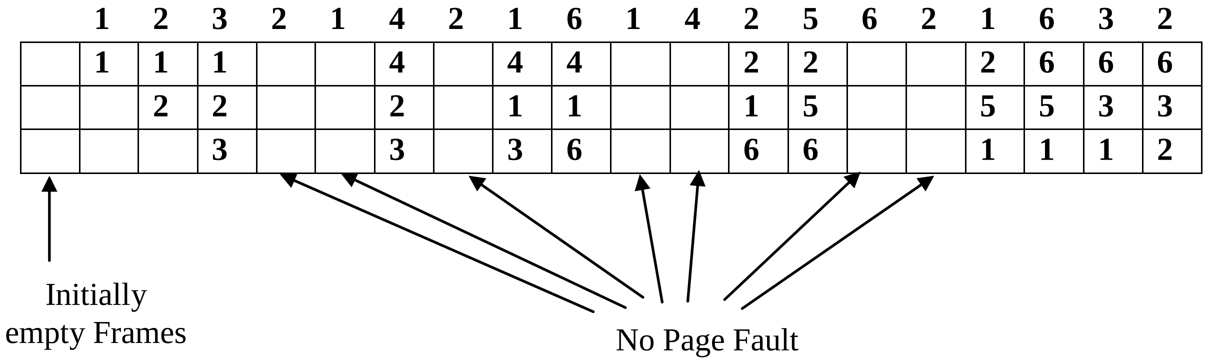
This algorithm has a severe drawback. If the oldest page is accessed frequently, the performance of this algorithm declines, since whenever the page swapped out will be required, it will result in another page fault, which degrades the performance.

Reference String

A reference string refers to the sequence of page numbers referenced by a program during its execution.

Assume a reference string:
1, 2, 3, 2,1,4, 2,1, 6, 1, 4, 2,5, 6, 2,1, 6, 3, 2

Let frames 3 are available for allocation page frame. Find number of page faults.



	1	**2**	**3**	**2**	**1**	**4**	**2**	**1**	**6**	**1**	**4**	**2**	**5**	**6**	**2**	**1**	**6**	**3**	**2**
	1	**1**	**1**			**4**		**4**	**4**			**2**	**2**			**2**	**6**	**6**	**6**
		2	**2**			**2**		**1**	**1**			**1**	**5**			**5**	**5**	**3**	**3**
			3			**3**		**3**	**6**			**6**	**6**			**1**	**1**	**1**	**2**

Initially, all 3 slots are empty, so when 1, 2,3 came they are allocated to the empty slots —> **3 Page Faults.**

Next page 2 refer, it is already in memory so —> **0 Page Faults.**

next page 1 refers, it is already in memory so —> **0 Page Faults.**

next page 4 refer, it is not available in memory and no free frame so it replaces the oldest page slot i.e 1. —>**4th Page Fault**

Next page 2 refer, it is already in memory so —> **0 Page Faults.**

Next page 1 refers, it is not available in memory and no free frame so it replaces the oldest page slot i.e 2. —>**5th Page Fault**

next page 6 refer, it is not available in memory and no free frame so it replaces the oldest page slot i.e 3. —>**6th Page Fault**

Next page 1 refers, it is already in memory so —> **0 Page Faults.**

Next page 4 refer, it is already in memory so —> **0 Page Faults.**

next page 2 refer, it is not available in memory and no free frame so it replaces the oldest page slot i.e. 4 —>**7th Page Fault**

next page 5 refer, it is not available in memory and no free frame so it replaces the oldest page slot i.e. 1 —>**8th Page Fault**

Next page 6 refer, it is already in memory so —> **0 Page Faults.**

Next page 2 refer, it is already in memory so —> **0 Page Faults.**

next page 1 refers, it is not available in memory and no free frame so it replaces the oldest page slot i.e. 6 —>**9th Page Fault**

next page 6 refer, it is not available in memory and no free frame so it replaces the oldest page slot i.e. 2 —>**10th Page Fault**

next page 3 refer, it is not available in memory and no free frame so it replaces the oldest page slot i.e. 5 —>**11th Page Fault**

next page 2 refer, it is not available in memory and no free frame so it replaces the oldest page slot i.e. 1 —>**12th Page Fault**

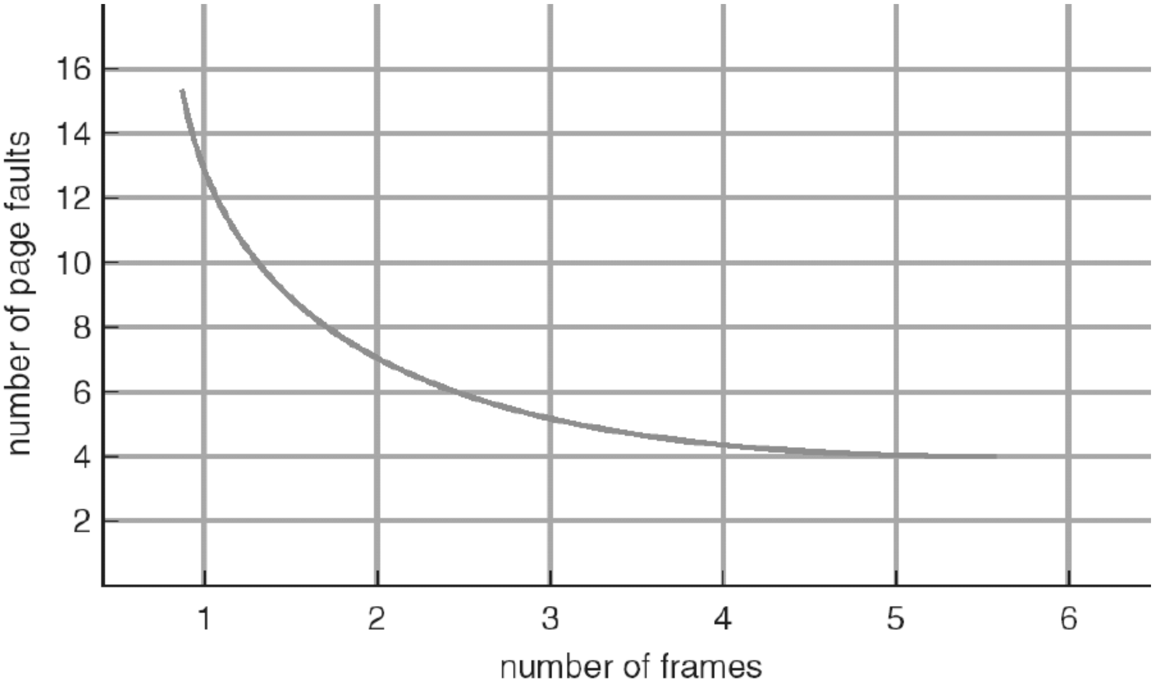
The number of page faults in FIFO is 12

Let frames 4 are available for allocation:

	1	**2**	**3**	**2**	**1**	**4**	**2**	**1**	**6**	**1**	**4**	**2**	**5**	**6**	**2**	**1**	**6**	**3**	**2**
	1	**1**	**1**			**1**			**6**	**6**		**6**	**6**					**3**	
		2	**2**			**2**			**2**	**1**		**1**	**1**					**1**	
			3			**3**			**3**	**3**		**2**	**2**					**2**	
						4			**4**	**4**		**4**	**5**					**5**	

Page fault in FIFO with 4 frames is 9.

By observing these two examples it states that if the number of frames will increase then the number of page fault must be decrease



Graph of Page Faults Versus The Number of Frames

The above graph reveals that if the number of frames increases then the number of page fault decreases but Belady’s anomaly proves that if the number of frames increases then it may be increased in page fault also while using the First in First Out (FIFO) page replacement algorithm.

Let string is 5, 6, 7, 8, 5, 6, 9, 5, 6, 7, 8, 9, 5, and frameset is 3.

	5	**6**	**7**	**8**	**5**	**6**	**9**	**5**	**6**	**7**	**8**	**9**	**5**
	5	**5**	**5**	**8**	**8**	**8**	**9**			**9**	**9**		**5**
		6	**6**	**6**	**5**	**5**	**5**			**7**	**7**		**7**
			7	**7**	**7**	**6**	**6**			**6**	**8**		**8**

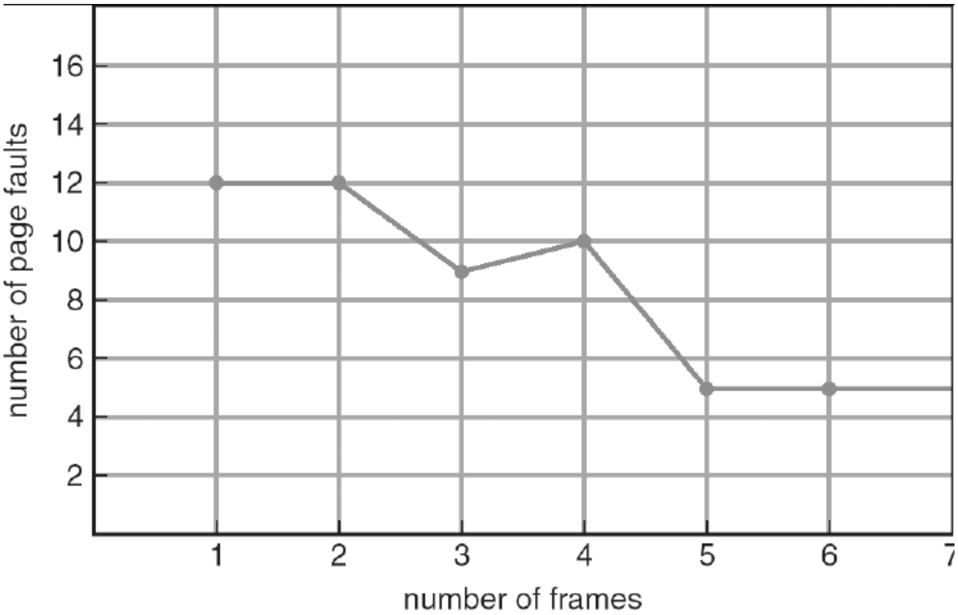
There are 10-page faults

Now with 4 frames set

	5	**6**	**7**	**8**	**5**	**6**	**9**	**5**	**6**	**7**	**8**	**9**	**5**
	5	**5**	**5**	**5**			**9**	**9**	**9**	**9**	**8**	**8**	**8**
		6	**6**	**6**			**6**	**5**	**5**	**5**	**5**	**9**	**9**
			7	**7**			**7**	**7**	**6**	**6**	**6**	**6**	**5**
				8			**8**	**8**	**8**	**7**	**7**	**7**	**7**

there are 11-page faults.

So by the increasing number of frames, it may be the increase of the number of page faults like in the above example where with 3 frames set there are 10-page faults and with 4 framesets there are 11-page faults. This is the Belady's Anomaly.



FIFO Illustrating Belady's Anomaly

The Optimal (OPT) Page Replacement Algorithm

Optimal page replacement algorithm is considered the best possible page replacement policy in a virtual memory theoretically but it is difficult to implement. According to the optimal page replacement policy, the page in the main memory, which will not be referred to for the longest time is swapped out from the main memory to create room for the requested page. A page should be replaced, which is to be referenced in the most distant future. Since, it requires knowledge of the future reference string, which is not practical. In the optional page replacement algorithm, the number of page faults is minimum as compared to another algorithm.

Practically, implementation of the optimal page replacement algorithm is done as follows:

Usually, all pages are labeled with the number of instructions that will be executed before this page will be used again in the future. When a page fault occurs the page with the highest number is replaced with the requested page that has caused a page fault.

For example, let pages 14, 21, and 18 are present in the main memory. Page 82 is required to be loaded resulting in a page fault since page 82 is not there in the memory at present. Page 14 is not required till the next 2000 instruction. Page 21 is not required till the next 1500 instructions. Page 18 is not required till the next 1700 instructions. This means page 14 is the one that will not be accessed by the CPU for the longest time. Page 14 is swapped out of the main memory to create room for page 82.

Advantages

It has the lowest rate of occurrence of page faults.

It improves the system performance by reducing overhead for several page faults and swapping pages in and out when a page fault occurs.

Disadvantages

It is very difficult to implement.

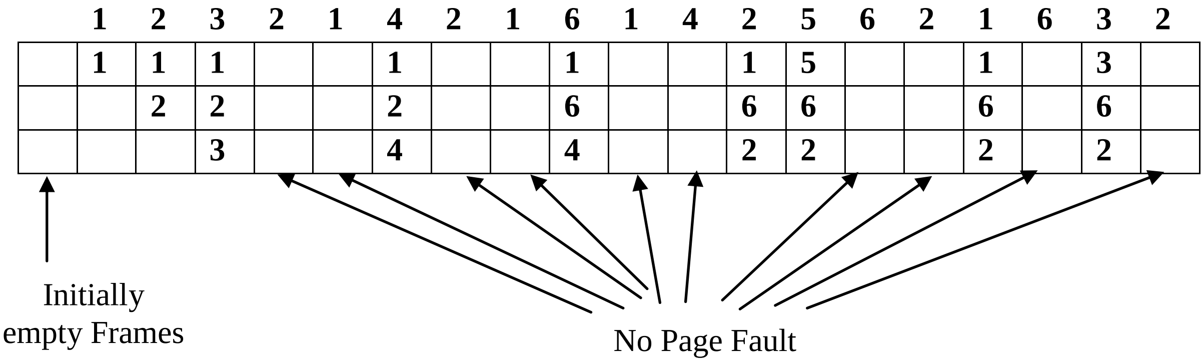
The situation is very similar to that of implementing the SJF algorithm in process management. It becomes very difficult for an operating system to calculate after what interval a page is to be referred to.

Example:

Let a reference string as follow:

1, 2, 3, 2,1,4, 2,1, 6, 1, 4, 2,5, 6, 2,1, 6, 3, 2

Let frames 3 are available for allocation page frame. Find number of page faults.



	1	2	3	2	1	4	2	1	6	1	4	2	5	6	2	1	6	3	2
	1	1	1			1			1			1	5			1		3	
		2	2			2			6			6	6			6		6	
			3			4			4			2	2			2		2	

There is a 9-page fault in the optional page replacement algorithm but in FIFO there are 12-page faults.

Least Recently Used (LRU)

The LRU algorithm uses information about the pages accessed in the recent past to predict the near future. The LRU algorithm is when a page fault occurs, the page that has not been referred to for the longest time is swapped out of the main memory to create space for the requested page that has caused the page fault. It replaces the page which has been used least recently.

Implementation of LRU can be done in various ways. One of the common methods to apply the LRU in a scheme for virtual management is using an array. The array stores the information about the page present in the main memory. The front end of the array stores the page accessed recently. The rear end of the array stores the page that has not been accessed for the longest time.

Whenever a page, that is present in the main memory, is accessed, the information about the page in the array is shifted to the front end of the array. If a page fault occurs the page indicated by the rear end of the array is swapped out of the main memory and the requested page is swapped in the main memory. Information about the page swapped in is stored in the front end of the array.

For example, the array stores four-page numbers {12, 56, 27, 61}. Page 12 is at the front end of the array. This indicates that page 12 has been accessed recently. Page 61 is at the rear end of the array. This indicates that the page that has not been referenced for the longest time is page 61. Page 27 is accessed. No page fault occurs because page 27 is present as the main memory. Information about page 27 is shifted at the front end of the array. The array becomes { 27, 12, 56, 61}. Page 43 is required to be accessed. A page fault occurs since page 43 is not in the main memory. Page 61 is at the rear end of the array, which indicates that it has not been accessed for the longest time. Page 61 is swapped out of the main memory and information about page 61 is removed from the array. Page 43 is swapped in the main memory and information about page 43 is inserted at the front end of the array. The array becomes { 43, 27, 12, 56}.

Advantages

It is very feasible to implement.

This is not as simple as the FIFO algorithm but not as complicated to implement as the optimal page replacement algorithm.

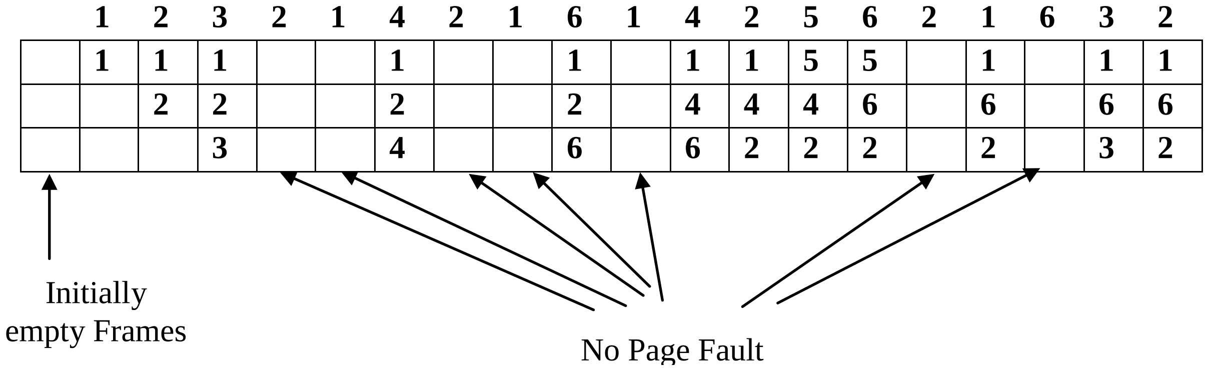
Disadvantage

The LRU algorithm requires additional data structure and hardware support for its implementation.

Let a reference string as follow:

1, 2, 3, 2,1,4, 2,1, 6, 1, 4, 2,5, 6, 2,1, 6, 3, 2

Let frames 3 are available for allocation page frame. Find number of page faults.



	1	**2**	**3**	**2**	**1**	**4**	**2**	**1**	**6**	**1**	**4**	**2**	**5**	**6**	**2**	**1**	**6**	**3**	**2**
	1	**1**	**1**			**1**			**1**		**1**	**1**	**5**	**5**		**1**		**1**	**1**
		2	**2**			**2**			**2**		**4**	**4**	**4**	**6**		**6**		**6**	**6**
			3			**4**			**6**		**6**	**2**	**2**	**2**		**2**		**3**	**2**

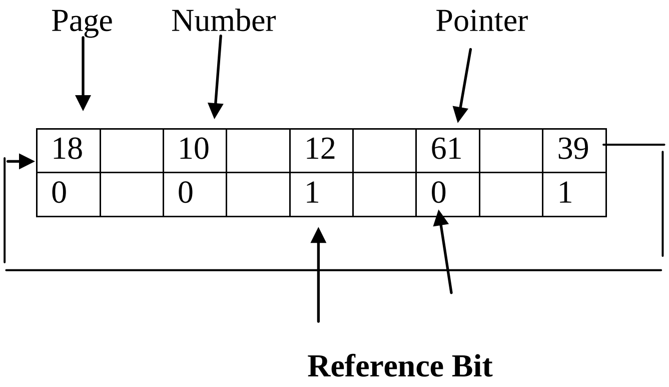
Page fault in LRU is 12. In this case, it is higher than optimal because a page that swap out is needed after one page so, it increases the page fault but in normal case page fault in LRU is less than FIFO but greater than the optional page replacement algorithm.

Clock algorithm

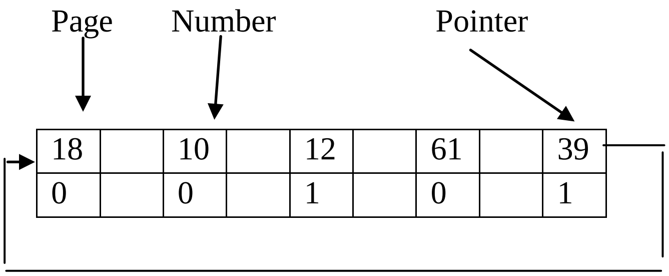
The clock algorithm also known as the second chance algorithm is a variation of the FIFO algorithm. In this, the page present for the longest time in the memory is given a second chance to remain loaded in the main memory. When that page is encountered for the second time, is swapped out to create room for the page that has caused a page fault. It is also referred to as **Not Recently Used (NRU)**. It replaces a resident page, which has not been accessed in the near past.

Working

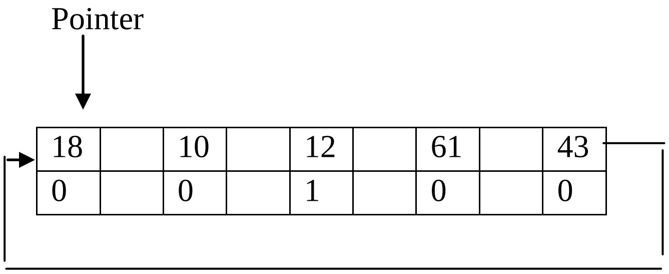
This is implemented using the concept of a circular queue. Each cell in the circular queue contains two values: a page number and its corresponding **reference bit**. The value of the reference bit can be either 0 or 1. If the value of a reference bit of a page is 1 it means that the page was encountered as the oldest page and followed a second chance. If the value of the reference bit is 0 this indicates that this page has not been encountered as the oldest page yet. When a page is found the oldest page present in the memory for the first time, its reference bit is set from 0 to 1. The next time when that page is found, it is swapped out of the main memory for creating free space in the main memory.



18		10		12		61		39
0		0		1		0		1



18		10		12		61		39
0		0		1		0		1



18		10		12		61		43
0		0		1		0		0

Page 43 swapped in from the secondary memory replacing page 39

A circular chain or pull is organized to implement the clock algorithm for selecting a page to be swapped out in a virtual memory scheme. The pointer is directing presently at page number 61. a page fault occurs. Page 61 has not been given a chance to remain loaded in the memory since its references bit is 0. The reference bit of page 61 is made 1 and the pointer is moved to the next page in the pull that is page 39.

The reference bit on page 39 is 1. This indicates that previously page 39 has been given a chance to be remain loaded in the memory. Thus this page is swapped out of the memory to create space for the new to swapped on page 43.

Counting Based Algorithms

Some page replacement algorithms apply the logic based on how many times a page has been accessed. Usually, these types of algorithms are implemented using an array that stores information about a page number and the number of times it has been accessed. The type of counting algorithms is:

Least Frequently Used (LFU) page replacement algorithm

Most Frequently Used (MFU) page replacement algorithm

When a page fault occurs, the operating system verifies how many times it has been accessed and proceeds according to the logic provided by the counting algorithm adopted for page replacement in the virtual memory by the operating system.

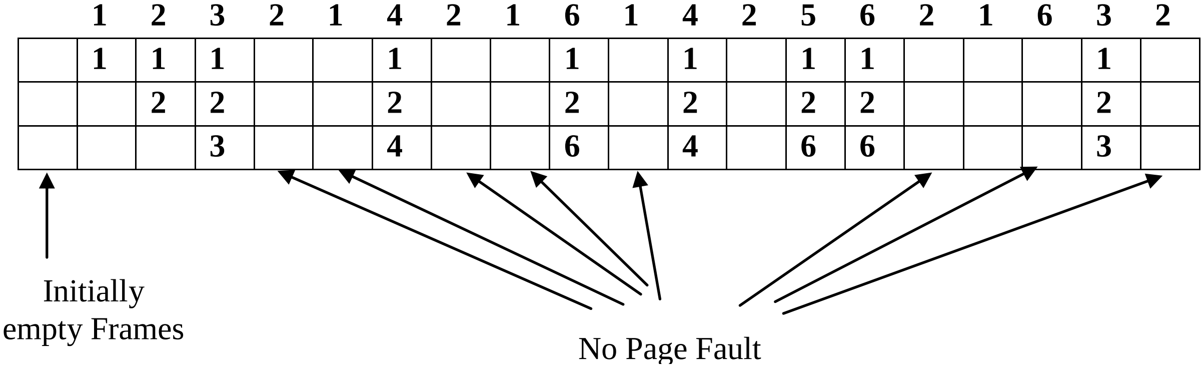
Least Frequently Used (LFU)

In the LFU algorithm, the page that has been accessed for the lowest/fewest number of times from the time when the page is loaded in the memory is replaced when a page fault occurs. The logic behind applying this algorithm is that some pages are accessed more frequently than others. Counting how many times a page has been accessed is used as an estimate of the probability of a page being referenced. Whenever a replacement is necessary, a page with the least count is replaced. The main drawback of this algorithm is that some pages may have a high usage initially and may build a high count but they have low usage subsequently, would remain in memory due to high count.

Let a reference string as follow:

1, 2, 3, 2,1,4, 2,1, 6, 1, 4, 2,5, 6, 2,1, 6, 3, 2

Let frames 3 are available for allocation page frame. Find number of page faults.



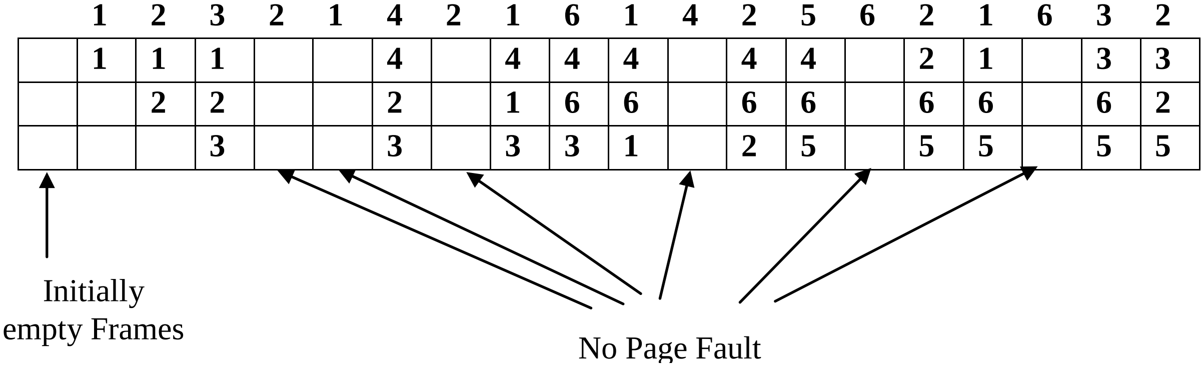
	1	2	3	2	1	4	2	1	6	1	4	2	5	6	2	1	6	3	2
	1	1	1			1			1		1		1	1				1	
		2	2			2			2		2		2	2				2	
			3			4			6		4		6	6				3	

There are 9-page faults is LFU

Note: If the counter of all the pages is the same then we adopt FIFO.

Most Frequently Used (MFU)

This algorithm replaces the page with the largest usage count. It is based on the assumption that the pages, with a smaller count, have been brought in recently and would need to be resident.



	1	2	3	2	1	4	2	1	6	1	4	2	5	6	2	1	6	3	2
	1	1	1			4		4	4	4		4	4		2	1		3	3
		2	2			2		1	6	6		6	6		6	6		6	2
			3			3		3	3	1		2	5		5	5		5	5

There are 13-page fault is MFU

Note: If the counter of all the pages is the same then we adopt FIFO.

Thrashing

In certain situations, a system spends more time processing page fault by swapping in and out pages than executing an instruction of processes. In other words, sometimes handling the pages faults becomes a huge overhead. This situation is called **thrashing**. Thrashing degrades the performance of a system. When multiprogramming increase more which degrades the performance is known as thrashing.

In a situation, where too many pages are active and required frequently, a page is required that is not present in the memory resulting in a page fault. One of the pages loaded in the main memory, which is also accessed frequently, is swapped out to create room for the required page. The required page is swapped in; the page swapped out is required for execution causing another page fault. Thus, a series of pages fault occurs and swapping becomes a large overhead.

In case a large number of processes are running in a multitasking operating system simultaneously, the memory becomes over-committed and thrashing occurs. In another word, the degree of multiprogramming is directly related to thrashing. The term, degree of multiprogramming, indicates many processes that are being simultaneous. This figure shows how system performance declines because of thrashing:

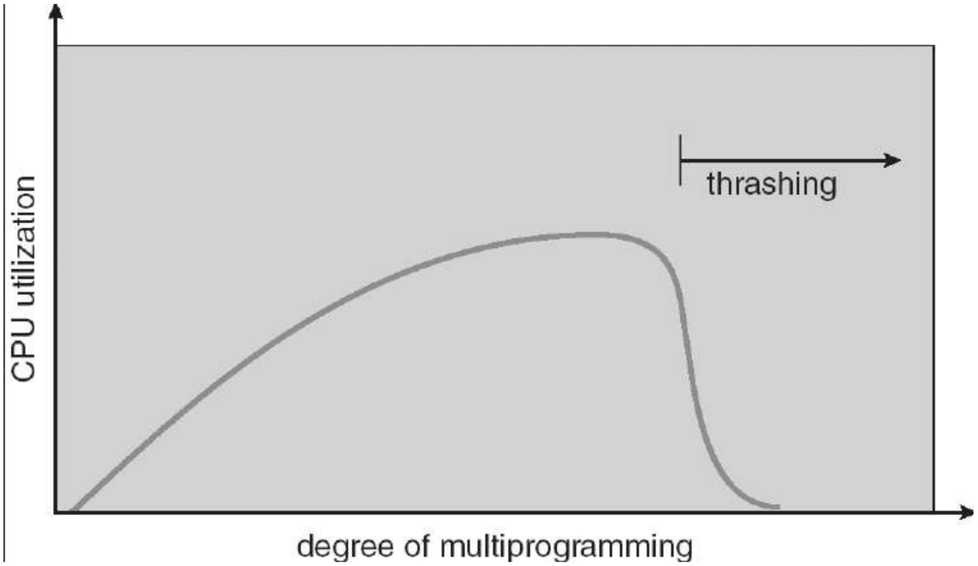


Figure: Effect of Thrashing on System Performance

This figure shows that system performance in terms of throughput reaches an optimal level to an extent if the degree of multiprogramming is increased. If the degree of multiprogramming is further extended thrashing occurs and system performance degrades. The peek of the graph represents the optimal performance of the system.

In certain cases, thrashing can be avoided. In most cases, an operating system attempt to recover from thrashing by suspending the execution of current processes and preventing the execution of new processes to start.

Selecting an appropriate page replacement policy to operate virtual memory plays an important role in the paging of thrashing. A page replacement strategy based on the local mode prevents the occurrence of thrashing to an extent. In local mode, while running a process, When a page fault occurs, swapping out of a page, which belongs to another process, is not allowed. A page belonging to the same running process is selected and swapped out to create room for the required page. In such cases, the thrashing of a particular process does not affect other processes. If other processes have occupied enough frames in the main memory, they can continue execution properly.

A page replacement policy based on the global model can lead the system to thrash. In global mode, when a page fault occurs, any page, regardless of which process it belongs to, is swapped out to create room for the required process that needs immediate execution. In global mode, a process may occupy a huge part of the main memory while other processes are competing for room in the main memory and hence resulting in page fault frequently.

Points to Remember

- The logical address is generated by the CPU.
- The logical address is also known as the virtual address.
- Logical and Physical addresses are different for execution time address binding.
- The base register is also called the relocation register.
- The user program deals with the logical address.
- Swapping is a technique to temporarily removing the inactive program from the memory of a system.
- Memory allocation methods are: first fit, Best fit, Worst fit.
- Paging is a memory management method.
- Logical memory divided it into similar size called pages.
- Physical memory divided it into similar size called frames.
- Paging eliminates fragmentation.
- In segmentation, the program is divided into variable size segments.
- Paging and segmentation can be shared.
- Virtual memory allows the execution of partially loaded processes.
- Virtual memory is the separation of user logical memory from physical memory.
- Demand paging is similar to a paging system with swapping.
- With the use of virtual memory, CPU utilization can be increased.
- In demand paging, pages are swapped in and out of the main memory.
- In demand paging, MMU also maintains an indexed table for address translation, known as a page table.
- This table is similar to that of a simple paging scheme.
- When the page is not present in the main memory a page fault occurs.
- Demand paging uses memory more efficiently.
- In demand paging, there is no limit on the degree of multiprogramming.
- Effective Access time calculated as

  $(1 - p) \times ma + pft$
  where p = page fault

  ma = memory access time

pft = p × (page fault overhead + swap page out + swap page in + restart overhead)

- EAT is directly proportional to the page fault rate.
- Demand paging can have a significant effect on the performance of a computer system.
- Page replacement algorithms are:
  - First In First Out (FIFO)
  - Optimal Page Replacement Algorithm
  - Least Recently Used (LRU)
  - Clock Algorithm (Second Chance Algorithm)
  - Counting Based Algorithms

  Least Frequently Used (LFU)
  Most Frequently Used (MFU)
- FIFO replaces the page at FRONT of Queue.
- Optimal it replaces the page that will not be used for the longest time.
- LRU replaces the page that has not been used for the longest time.
- LFU replaces the page, which has the smallest count.
- MFU replaces the page, which has the highest count.
- Belady anomaly state that page fault rate may increase as the number of allocated frames increases,
- FIFO replacement algorithm suffers from Belady anomaly.
- The degree of multiprogramming increases more which degrade the performance of the system is known as thrashing.
- Thrashing degrades the performance of a system.

4.7 PRACTICE EXERCISES

1. What is memory management?
2. How Operating system helps in memory management?
3. What are the different types of memory management techniques?
4. Explain the following terms:

   (i) Internal fragmentation

   (ii) External fragmentation.

   (iii) Compaction
5. Explain the strategies of the first fit, best fit, and worst fit in the multiple partition memory management techniques?
6. What is the basic difference between Single partition and Multiple partition memory management techniques?
7. What is Paging?
8. Why is page size always power by 2?
9. Explain the technique of Segmentation in memory management?

10. Why paging is used?
11. Describe the following:
    - Swapping
    - Structure of Page Table
    - First Fit
    - Best Fit
    - Worst Fit
    - Address Binding
    - Production and Sharing

12 Write a note on virtual memory.

13. Explain the demand paging.
14. What do you mean by page fault?
15. Write steps handling the page fault.
16. Explain the performance of demand paging.
17. Why we need the page replacement algorithm.
18. Describe the working of various page replacement algorithms.
19. What do you mean by reference string?
20. What is the Belady anomaly?
21. Write a short note on Thrashing.
22. Consider the following page reference string:
    1, 2, 3, 4, 7, 5, 1, 2, 1, 7, 5, 3, 2, 4, 2, 3, 1, 6, 4, 2, 1, 2, 3, 6
    How many page faults will occur for the following page replacement algorithms? Assume a set of four page-frames (initially all empty).
    FIFO
    LRU
    Optimal

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT IV: FILE SYSTEM

STRUCTURE

5.5 File-System Implementation

5.6 Directory Implementation

5.7 Allocation Methods

5.8 Free-Space Management

5.9 Practice Exercises

5.0 OBJECTIVE

- Understanding Concept of File, different methods to access files.
- Using File system structure and their implementation,
- Concept of Directory and its allocation methods

5.1 FILE CONCEPT

A file is a group of similar records which is stored in memory. The file is treated as one unit by users and applications. It may be mentioned by name. The filename should be sole which means in the same location file's name should be unique. It may be created, deleted, appended, truncated. There should be file manager which provides a protection mechanism to allow machine user to administrator how processes executing on behalf of different users can access the information in a file.

5.1.1 File System

The file system in the operating system is assigned with the work of storing, controlling, and managing data that is stored on disks or secondary storage in the form of files. File system management is responsible for maintaining consistency in data when multiple users access files concurrently. It also provides measures for file protection at times when the system crashes.

5.1.2 Introduction

Files are stored permanently on secondary storage devices, such as hard disks. A file system is a part of the OS that is responsible for controlling secondary storage space. It hides device-specific complexities and provides a uniform logical view of users. A function of file system includes:

- It allows users to provide a facility to give the name of the file as user-defined names, to create, append, truncate, and delete files.
- It maps user-defined names with low-level spotters, names in a machine-understandable format so that the machine finds a file uniquely.
- It provides a uniform logical view of data to users rather than a physical view i.e. internal structure of files in which they are stored on the disk, by giving user-friendly interface.
- It controls transferring of data blocks between secondary storage and main memory and also between different files.
- It offers semantics or the file-sharing rules and regulations between numerous processes and users.
- It also allocates and manages space for files on secondary storage devices, such as disks or magnetic tapes. Space management is an important part of the file system.
- It protects the files from system failures and applies measures for recovery and backup.
- It provides security measures for confidential data such as electronic funds or criminal records.

- It also provides encryption and decryption facilities to users. Encryption is a mechanism of converting data into some code form that is unreadable to everyone except the recipient. Decryption is the reverse process of encryption.

A file system is implemented as a layer of OS and is placed between the kernel and the memory manager. It consists of utility programs that run as constrained applications that are used to control access to files. Users may access files with the help of the file system only.

5.1.3 File Attributes and Naming

The file is the named collection i.e. each unit is having a name, that is unique and the end-user identifies a file by its name, usually considered as a linear array i.e. sequentially arranged, of date and records. Almost all input/output operations are performed through files. All inputs are done via files and all outputs are recorded in files. Data or information stored in files is of many types. A file has a certain defined structure according to its types such as executable programs, numeric and textual data, pictures, images, and sound recordings. Each file is saved in a directory which is acts as a container for that file. Access to file is done through these directories. A user assigns a name with each file through which it can refer. Moreover, the file system requires more information about a file that is attached with the file in the form of attributes to maintain consistency and reliability of data and control multiple accesses.

5.1.4 File Attributes

File attributes are required by any file system to manage or maintain a file. It may differ from one operating system to another operating system. File attributes are the information about a file that is associated with every file. Some attributes are unique for each file in disks such as file locations, name and creation, date, and time. Few attributes are accessible for users, such as access privileges, name, or size of a file, whereas some of them are specifically assigned to a file for file system usage.

File attributes vary from one OS to another, but few of them are needed by every OS. The major types of attributes are:-

(1) **User-Defined attributes**

(2) **System-Defined Attributes**

5.1.4.1 User-Defined attributes

- **File name:** An identifier chosen by the user to address the file. Usually a string of alphanumeric characters, some OS allows the use of special characters, such as #, *, or $, in file names, and must be unique in its file directory.
- **File type:** Type of information stored i.e. binary file, text file, picture file, or program file.
- **Owner:** Name of the creator of files that controls and provides access privileges, such as read-only or read-write, to other users.
- **Permitted privileges:** It contains information that determines read privileges, write privileges and execute privileges.

5.1.4.2 System-Defined Attributes

- **Low-level identifier:** Machine understandable names, usually in binary digits that are used by hardware to identify a file by mapping it with a user-defined name. This attribute also consists of numbers.
- **Location:** Address of sector or area where the file is stored on the disk. Usually, this information is used as a pointer; a value used by programs to find the location of a certain sector or file, to the location and consists of numbers.
- **File size:** Current file size in bytes, or words, or in blocks.
- **Creation date:** This contains the date when the file was created.
- **Allocated size:** Total allocated size of a file by the file system.
- **Volume:** Indicates which type of device is used for storing files.
- **Date of last modification:** Contains date and time of last update, insertion, and deletion in a file.

All the attributes of a file are stored in a directory where the file resides. Their storage takes more than 1 Kb space per file. Some attributes are stored in the header record associates with each file.

5.1.5 Naming

The most significant attribute of a file is its name that is given by the user who created it. Users should associate a name with every file to uniquely identify it and access files through these names. The file's name should be unique in its directory. In a shared system, it is recommended that the user must assign a unique name to a file.

Files are accessed by giving a complete path or address i.e., you have to specify the names of directory and subdirectories with the filename, all when combined makes a complete address for a file. It is possible, that a directory is also stored in another directory that is contained in the root directory, a directory at the top level. Therefore, a file in a system can be located by a giving-by-giving complete path from the root directory to the files, specifying all the intermediates directories, to the file system. All the directories, other than the root directory, are the subdirectories of the root directory. The root directory is denoted by '\'.

An example of the pathname for a file:

C: \ Rohit \ rsts \ os.doc

The pathname of a file is also called the complete address for a file. Slash is used to restricting names in a sequence and also indicate subdirectories of a directory. Two or more files are allowed to have similar names provided they have different pathnames i.e. their parent directory should be different.

The process that calls a file is associated with it a current directory also called the working directory. In this case, a complete pathname is not required rather; a user can access files giving pathname starting from this working directory.

5.1.6 File Operations

Basic operations on files are:

Create a File: For creating a file, address space in the file system is required. After creating a file, the entry of that file must be entered in the directory. Then directory entry contains the file name and location of that file which is stored in the file system.

Writing a File: A system call is used for writing into a file. It requires two parameters one file name and the second information which is to be written.

Reading a File: A system call is called to read a file. It requires two parameters: one the file name and its memory address.

Delete a File: System will search the directory with the file to be deleted. It releases all file space that can be reused by another file.

Truncating File: The user may need to remove the contents of the file but want to keep its attributes. It recreates a file.

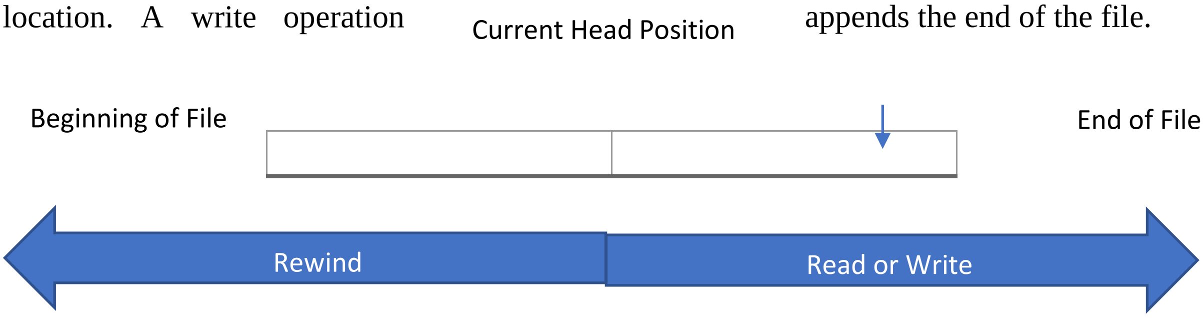
5.2 ACCESS METHODS

In the file, information can be accessed in various ways. Different types of file access methods are:

1. Sequential access
2. Direct access
3. Indexed Access

1. **Sequential Access**

It is a very simple method among the other methods. In the file, information is serially accessed which means one record after other records. For example: in a tape recorder, the tape plays in sequential mode. A read process reads the file's next portion and automatically advances a file pointer, that keeps tracks of Input/Output location. A write operation appends the end of the file.



2. **Direct Access**

It allows random access to any block. This model is based on a disk model of a file. It allows programs to read and write records in any order.

3. **Indexed Access**

The records in a logical sequence according to a key contained in each record. The system maintains an index containing the physical address of certain records. By using the key attributes, the records can be indexed directly.

Sequential and direct access method both are not supported by all operating system. Some operating system uses sequential access method, some operating system uses direct access method and some operating system used both access methods.

5.3 DIRECTORY AND DISK STRUCTURE

Directories are considered as symbolic tables of files that are store all the related information about the file it holds, with the content. This information includes file attributes, location, type, and access privileges. Directories are also known as containers of files.

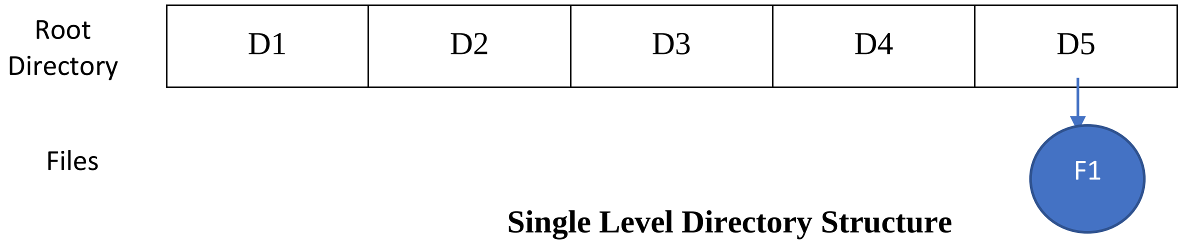
The directory is itself a file that is owned by the OS. Millions of files present in the system need to be managed. Directories provide means to organize files in a structure. Each entry in a directory contains information about a file. Similar to files, operations such as insertion, deletion, and searching can be performed on it. Operations performed on these entries are:

- **Searching a file:** Whenever a file is referenced, the directory must search for the related entry.
- **Create a file:** An entry of every newly created file needs to be added to the directory.
- **Delete a file:** Whenever a file is deleted, the related entry should be removed from the directory.
- **List directory:** A list of files in a directory should be shown whenever a user requests it.
- **Rename a file:** The name should be changeable when the use of the file changed or its location changes.
- **Update directory:** Whenever a file attributes changes, its corresponding entry needs to be updated.

Based on these entries and their operations, the structure for directories can be organized in different ways. The three most common structures for organizing directory are, single-level, two-level, and hierarchical structures.

5.3.1 Single-Level Structure

It is the simplest form of directory structure having only one level of directories. The entire files are contained in the same directory. It appears like the list of files or sequential files having file names serving as the key. The logical structure of a single directory is given in the figure below:



Single Level Directory Structure

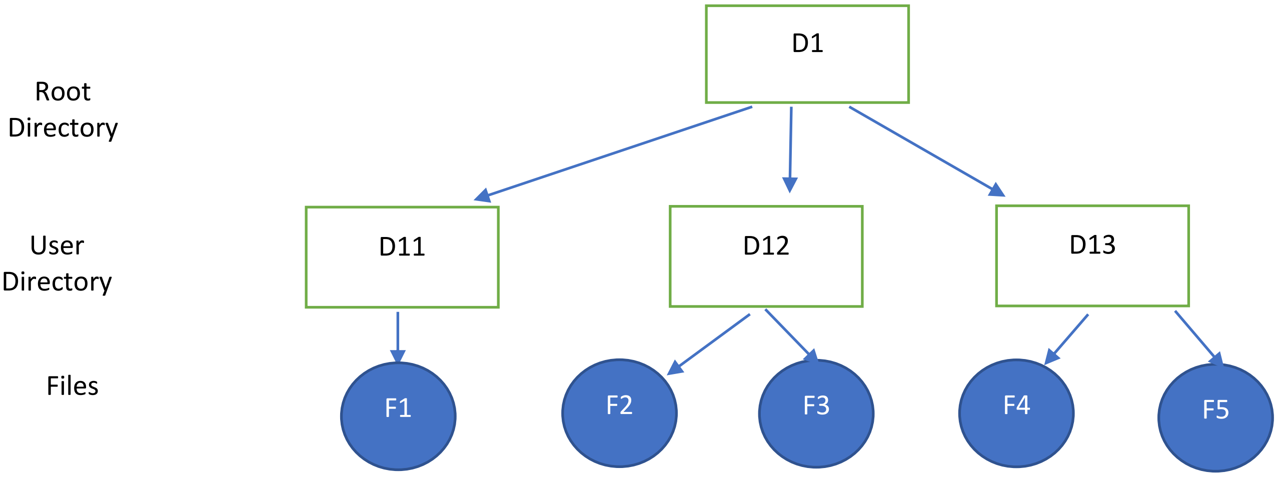
The directory structure is implemented in an earlier single-user system. It becomes outdated and inadequate in the multiple-user system. Even for a single user, it is difficult to keep track of the files if the number of files increases. Moreover, files are of different types, such as graphic files, text files, and executable files, and if the user wants to arrange these files in an organized manner such as group files by type, this structure becomes inconvenient.

Files in the single-level directory should have unique names because they are contained in one single directory. In a shared system, unique naming becomes a serious problem. These drawbacks lead us to design another structure of directories named a two-level structure.

5.3.2 Two-Level Structure

As the name suggests, this structure is divided into two levels of directories i.e. a master directory and user, and all these directories are contained and indexed in the master directory. The user directory represents a simple list of files of that user.

The two-level structure looks like an inverted tree of height 2. The root of this tree is the master directory having user directories as its branches. Files are the leaves of these branches. The logical structure of two-level directories is shown in the figure below:



Two Level Directory Structure

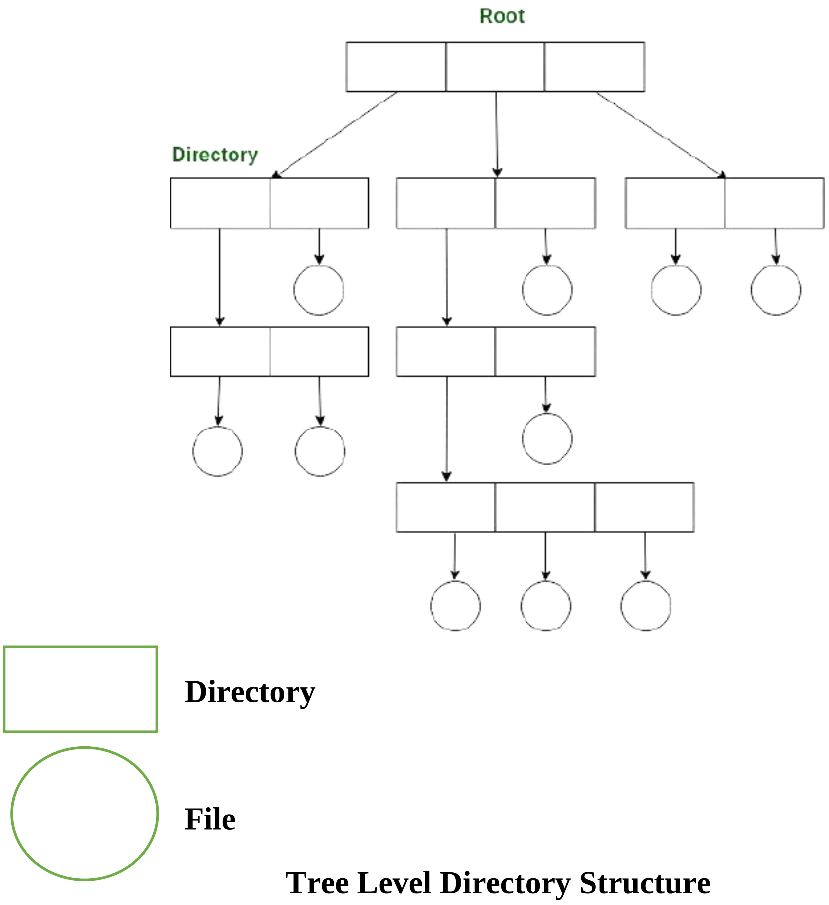
A user name and file name are the pathnames for a file. This structure solves the problem of unique names up to a certain extent i.e. user can assign duplicate names to files provided files are present in different directories. Names need to be unique in the user's directory because two file names cannot be the same in one single directory. User searches a file in his directory only, which allows different users to have files with the same names.

To create or delete a file, OS searches only the user's directory that initiates the command. OS uses a special system program or system calls, to create or delete a user directory. This system program creates or deletes a user directory entry from the master directory.

This structure provides no help in grouping files of different types. In a shared system, one user wants to access another user's file because files are shared in a shared system network, which again creates the problem of uniqueness in file names. The user has to give a complete pathname to name a file in another user's directory.

5.3.3 Hierarchical Structure or Tree-Structured Directories

This is the most powerful and flexible structure and is implemented in almost every OS. The two-level structure is extended into a more advanced hierarchical structure of arbitrary levels. It uses the same concept of a two-level structure of master directory having user-directories as subdirectories. In addition, each user directory, in turn, has subdirectories and files as its branches and leaves. A typical hierarchical structure of directories and files is shown in the figure below.



Tree Level Directory Structure

Users can create their subdirectories to organize files of different types, such as separate subdirectories for graphic files, or separate subdirectories for text files. Special system calls are used to create or delete directories. Internal formats, i.e. the internal structure in which the details of the directory are stored, of each directory ha

s an entry that stores special bits representing a subdirectory or a file i.e. 0 bit represents a file and 1 bit represents a directory.

The user always works on the files in the current directory. The current directory holds all the files, which a user currently requires. OS searches the current directory for reference to a file. In a hierarchical structure, the user can access a file, which is not in the current directory, giving pathname. Users can change the current directory also through system calls.

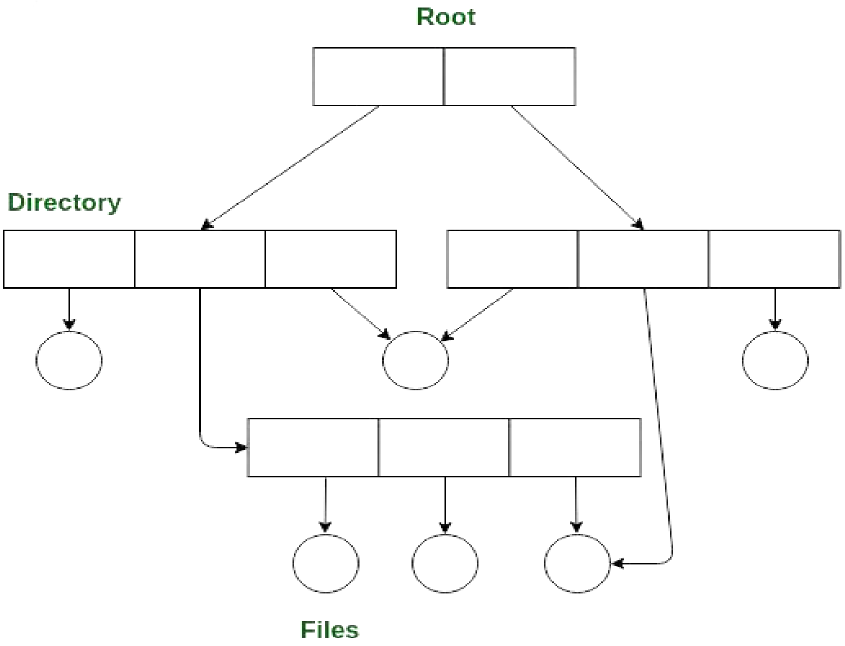
In a hierarchical structure, a file can be referenced in two ways, *absolute pathname,* and *relative pathname.*

Absolute pathname starts from the root and ends at the required file following a path of directories and subdirectories.

Relative pathname starts from the current directory to the file. Users can access another user file by giving its pathname. In a hierarchical structure, the pathname to a file can be longer than a two-level directory. This increases the search time for a file that resides in other user directories.

5.3.4 Acyclic Graph Directories

It always directories to shared subdirectories and files. Some files or directories may be in two different directories. Shared files and directories can be implemented by using links. Link is implemented as an absolute path or relative pathname. It is more flexible than a simple tree structure but sometimes it is more complain.



The acyclic graph directory structure

Characteristics

1. Files and sub-directories can be shared.
2. It is more flexible than three structures.
3. Deletion of a shared file is difficult.
4. Consistency should be taken care of.

5.3.5 General Graph Directories

In this structure, cycles are also permissible within a directory structure are from more than one parent directory, multiple directories can be made. The main disadvantage of the general graph directory structure is to compute the total size or memory required which has been taken by the files and directories.

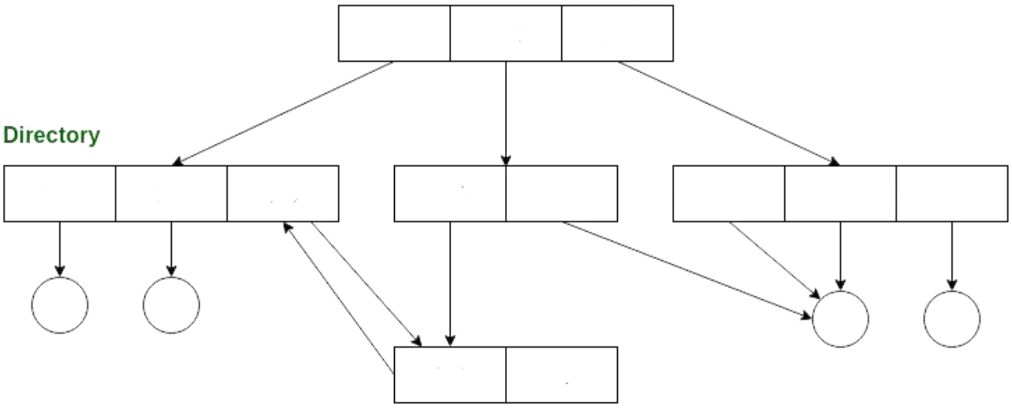
Characteristics

It permits more cycles.

It is more flexible as compared to the other directories structure.

It is more expensive than others directory structures.

It requires garbage collection.



A General Graph Directory structure

5.4 FILE-SYSTEM STRUCTURE

The file system resembles a complete directory structure that includes a root directory following subdirectories and files under it. The file system is required to enhance the function of retrieval and storage of files. We can mount more than one file system that creates a directory structure showing an image of a single file system. Commands used for file system operations are:

Chfs: Change the file system characteristics

Crfs: Add a new file system

Rmfs: Remove a file system

Mount: Make files system available for user access.

In the real world, different Operating systems use different file systems. Each file system has a different directory structure and different ways to represent files.

1. **CD ROM**

CD ROM is an acronym for Compact Disc-Read Only Memory. Data is stored on one side of it that is coated with the highly reflected material. Data is stored in the form of series of microscopic pits, i.e. data is stored in binary digits of 0 and 1 and a pit represents a binary 1, which is read by a low-powered laser in a drive unit. To read data, the laser rotates the disk at various speeds at a Constant Linear Velocity (CLV).

CD ROM is used for storing audio in digital format and is of size 6547.4 MB. CD ROM can store data equivalent to 500 floppy disks. Data on CD ROM can be accessed much faster than any other tape. Accessing speed of CD ROM is 75-150 sectors per second. To read a CD ROM, we need a CD ROM reader and an appropriate software driver that controls the functionality of CD ROM.

Data is stored in the form of sectors with error-correction codes i.e. codes that are used to debug any error if occurred in the stored data. Data on CD ROM can be stored and retrieved, as files for that we need a file system to manage the stored data. The file system allows you to access the contents of CD ROM using the interface of a normal file system i.e. the file system stored on the disk.

2. **FAT16**

To understand the FAT 16 file system, a discussion on File Allocation Table (FAT) is necessary. FAT is a table that has entries in the form of pointers. These pointers point to the files that are present in a partition. Partition is a logical division of the table. Each entry in this table consists of a block number, a 0 valued block number represents as unused block any new entry is stored on it. Partition of the disk depends on the file system used. The two most common file systems are FAT 16 and FAT 32.

FAT 16 is the old MS-DOS file system created by Microsoft in 1977 that used 16 bits table. FAT 16 support filenames of 8.3 standard i.e. 8 characters for the name and 3 characters for file extension.

The value stored in the FAT table is 16 bits. A 16-bit file allocation table means that the size of the address stored in the entry of FAT is limited to 16 bits. This implies that you can’t have storage allocation units i.e. the entries in the FAT 16 table are restricted to 65536 units. This is the main drawback of FAT 16.

Another drawback of FAT 16 is that its partition is of fixed size i.e. of 32k. That wastes a lot of disk space in storing files of small size such as pictures, HTML documents, etc.

3. **FAT32**

It is an advancement of FAT 16 that supports 32 bits addresses. Entries in FAT 32 tables can hold a value up to 32 bits. It divides the hard disk into smaller partitions of a small disk area than FAT 16 so that disk space is used more efficiently with increased storage capacity.

FAT 32 allows you to create a partition up to 2048 GB size. The number of clusters in FAT 32 ranges around 4 billion i.e. 4,294,967,296 inexact. FAT 32 supports long file names of a maximum of up to 255 characters. File names are not case sensitive i.e. no matter whether the names are in capital letters or small letters; it saves the case of filenames once created. Minimum disk space needed for FAT 32 is 8 GB. Os that supports FAT 32 are: OSR2, Windows 98, Windows 2000, and Linux.

4. **Unix**

Unix is a multi-user and multitasking OS that allows multiple users to access multiple files concurrently. It is a data structure that resides on disks. Unix file system has a tree-like structure having a root directory (/) at the top node. This hierarchical structure appears to users as a single file system having all types of files in one tree, the concept of drives such as A, C: etc is not found here. Each node, a vertex of the UNIX file system tree, represents a separate file system.

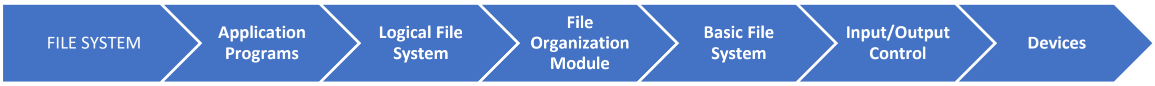
Thus, the separate device is mounted in one single file system. A file in UNIX is a collection of randomly addressable bytes, not sequentially; each byte has a different address, not in the continuity of a previously-stored byte. Files are allocated on the block randomly on the disk. The index method is used to keep track of each file. A filename can extent up to 255 characters excluding forward-slash (/) & Null and are case sensitive. File in Unix may contain holes i.e. no data blocks. The directory is also a file except the user can’t write on it. Unix associates its file with three access permissions: read, write, and executable. Users can access files giving absolute and relative pathname.

UNIX file system is made up of four parts:

- **Boot block:** Contains boot program and load kernel into memory when a user boots the system or switch on the system.
- **Superblock:** Contains all information about the attributes of a file.
- **I-list:** Contain a list of I-nodes of the file system. I-node is a data structure that holds information about an individual file, such as file size, access privileges, or file name. Each file is related to I-node and is identified by I-nodes number.
- **Data block:** Contains actual file contents.

5.5 FILE-SYSTEM IMPLEMENTATION

File system structured in the layers which are: Application Programs, Logical File System, File Organization Module, Basic File System, Input/Output Control, Devices



- **Input/Output Control level –**
  Device drivers perform as an interface between devices and Operating Systems, these drivers help to transfer data from disk to main memory and vice versa. It picks block numbers as input and as output, it returns low-level hardware-specific instruction.

- **Basic file system –**
  It refers to normal commands to the particular device driver to do operations: read and write on particular physical blocks on the disk. A block is part of memory in the buffer that can contain the contents of the disk block and cache stores frequently used file system metadata. It also handles the memory buffers and caches.

- **File organization Module –**
  It contains information about the files such as their location, and their logical blocks, and physical blocks. Physical blocks were entirely different so they do not match with logical numbers of the logical block which are numbered from 0 to N. It contains some free space that is tracked by unallocated blocks.
- **Logical file system –**
  It manages metadata information about a file i.e includes all details about a file except the actual contents of the file. It also keeps with the help of File Control Blocks. File Control Block (FCB) contains all the information about a file such as the owner of the file, size of the file, permissions granted, file location, and its contents.

Advantages :

1. Replication of code is lessened.
2. Each file system can have its independent logical file system.

Disadvantages :

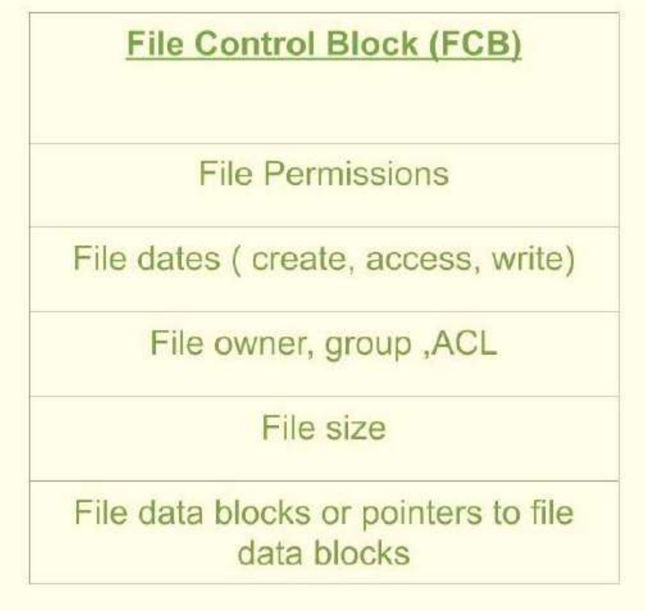
If someone accesses many files simultaneously then its outcomes will reduce the performance of the system.

We can **implement** a file system by using data structures which are:

1. **On-disk Structures –**
   Generally, it contains the following information:
   a. total number of disk blocks available
   b. free disk blocks available
   c. location of them.

Below given are different on-disk structures :

1. **Boot Control Block –**
   It is generally the 1st block of volume and it has information that is needed to boot any operating system. In UNIX, this block is known as a boot block and in NTFS it is known as a partition boot sector.
2. **Volume Control Block –**
   It contains information about a particular partition for example:- count of free blocks, size of blocks, and its pointers also. In UNIX it is known as superblock and in NTFS it is stored in the master file table.
3. **Directory Structure –**
   They kept file names and linked inode numbers. In UNIX, it contains file names and associated file names and in NTFS, it is stored in the master file table.
4. **Per-File FCB –**
   It holds details about files and it has an exclusive identifier number to allow association with the directory entry. In NTFS it is kept in the master file table.



File Control Block (FCB)
File Permissions
File dates (create, access, write)
File owner, group ,ACL
File size
File data blocks or pointers to file data blocks

Structure of File Control Block (FCB)

In-Memory Structure :
They are maintained in the main memory and these are helpful for file system management for caching. Several in-memory structures given below:

1. **Mount Table:** It is a table in which information about every mounted volume is stored.
2. **Directory-Structure cache –** It contains the directory information about which directory is freshly accessed.
3. **System-wide open-file table** –It holds the copy of the File Control Block (FCB) of every open file.

4. **Per-process open-file table** – It holds information on which process opened the file and it also maps with appropriate system-wide open-file.

5.6 DIRECTORY IMPLEMENTATION

A directory can be implemented in two ways:

1. Linear List
2. Hash Table

Linear List

The linear list is the simplest method. In this linear list of filenames used. For searching purposes, a linear search technique is used to find a particular entry. It is very simple but time-consuming to execute. Directory information is used often. Users also notice the slow implementation of access to it.

Hash Table

With the hash table, it decreases the directory search time. Insertion and deletion are very simple. Hash table takes the value which is compiled from the file name and then it returns a pointer to a file name in the form of a linear list. It always uses a fixed size.

5.7 ALLOCATION METHODS

The space allocation strategy is often closely related to the efficiency of file accessing and of logical to physical mapping of disk addresses. Three major methods of allocating disk space are widely used which are:

Contiguous Allocation Method

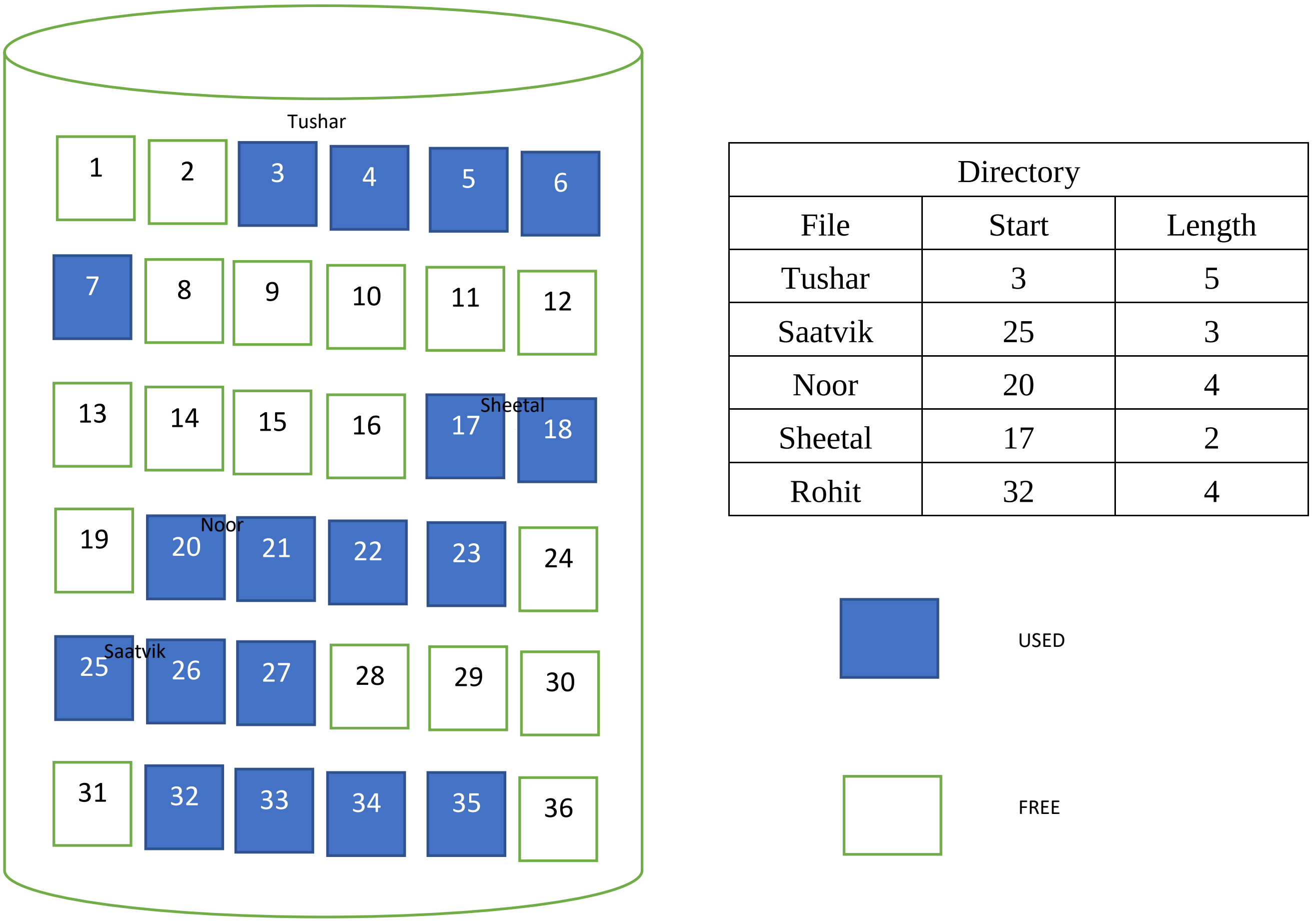
The contiguous allocation method always requires every file to occupy a set of adjoining addresses or blocks on the disk. Disk addresses describe in a linear order on the disk. The important thing to be noted is that, in this ordering, the block of memory represents as B, accessing block where is B+1 after block B normally requires no head movement. When head movement is needed, it is only one track. Thus, the number of disks seeks required for accessing contiguous allocated files is minimal.

Contiguous allocation of a file is defined by the disk address or blocks and the length of the first block. Suppose, the file has a size of N blocks, and its starting location is location B, then it must occupy consecutive blocks namely B, B+1, B+2,….., B+n-1. The directory entry must contain for each file indicates the address of the starting block of the file and the total length of the file in terms of blocks. This is the simplest method of allocation. Performance is good because the entire file can be read from the disk in a single operation.

The difficulty with a contiguous allocation is finding space for a new file. If the file to be created is n blocks long, then the OS must search for n free contiguous blocks. The most common approaches used to select a free hole among the set of available holes are First-fit, best-fit, and worst fit. Both first-it and best-fit are a better option as compare to worst fit in terms of both criteria which is time and storage utilization. Neither first-fit nor best fit is best in terms of storage utilization, but first-fit is generally faster.

These algorithms also have a disadvantage which is external fragmentation. As files are allocated and also deleted, the free disk space is broken into tiny chunks. External fragmentation exists when enough total disk space exists to satisfy a request, but this space not contiguous; storage is fragmented into a larger number of small holes.

Another problem with a contiguous allocation is determining how much disk space is needed for a file. At the file creation time, how much space it requires must be known and allocated. One question arises here, how does the creator know about the size of the file which is to be created? In some cases, this determination may be fairly simple, but generally, the estimation of the size of an output file is a little difficult. A directory containing the file information is maintained. Starting block address and a total number of blocks in a file are read from this directory to read/write a file.



Directory		
File	Start	Length
Tushar	3	5
Saatvik	25	3
Noor	20	4
Sheetal	17	2
Rohit	32	4

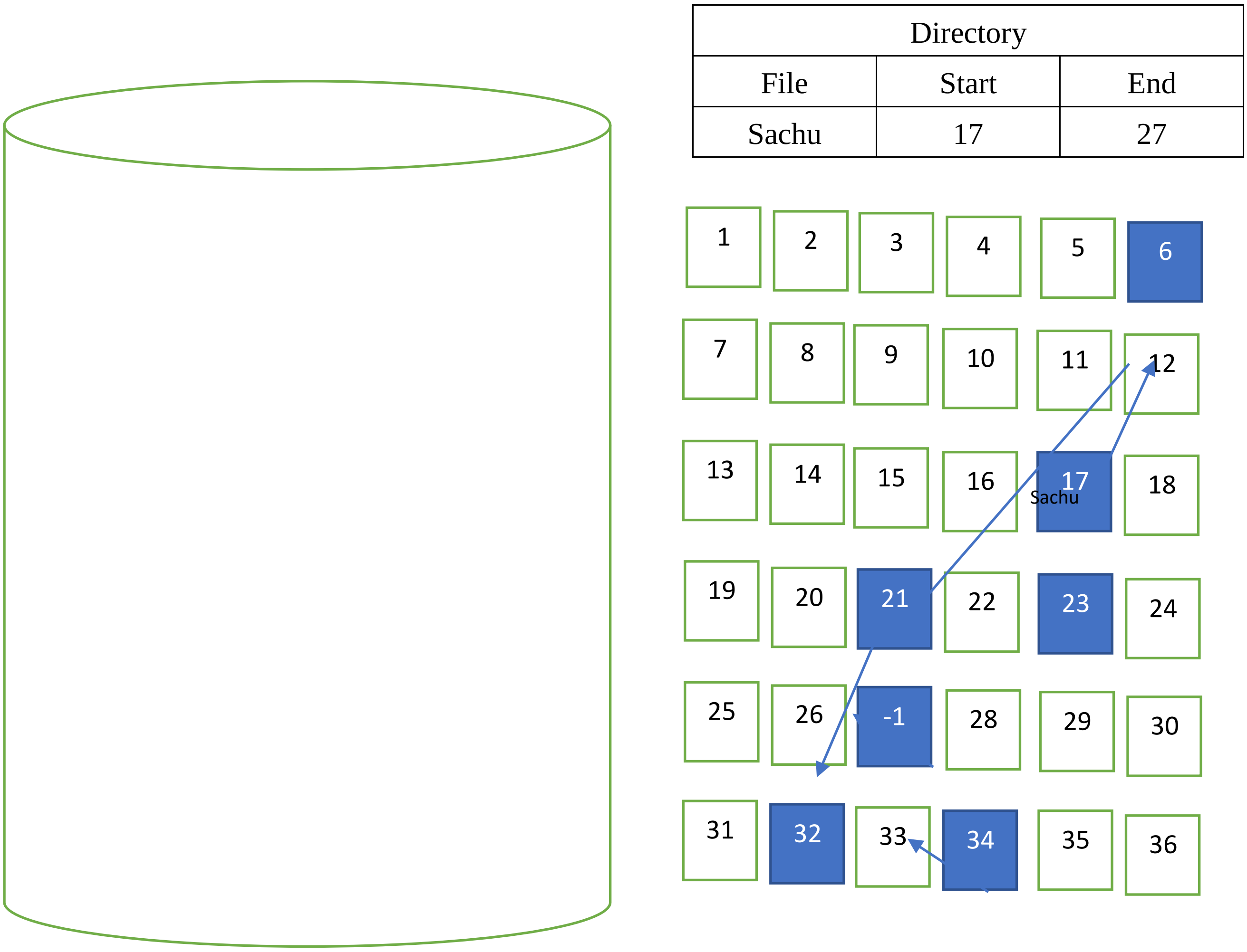
Contiguous File Allocation

Characteristics

1. It supports flexible size portions.
2. It requires Pre-allocation.
3. It needs only a single entry of a file.
4. Allocation frequency is only once.

Linked List Allocation method

The problems in contiguous allocation can be traced directly to the requirement that the spaces be allocated contiguously and that the files that need these spaces are of different sizes. These requirements can be avoided by using linked allocation.



Directory		
File	Start	End
Sachu	17	27

Link List Allocation

In linked allocation, every file is fragmented into Blocks and makes a linked list of these disk blocks. The directory stores a pointer to the first Block of the file. For example, a file of 7 blocks starts at block 17, they might be next block 6, then block 21, block 32, block 23, block 34, and finally last block 27. Each block holds a pointer to the next block and the last block contains a NULL pointer which shows that end of the link list. The value -1 may be used for NULL to differentiate it from block 0.

In the linked allocation method, every directory entry contains a pointer to the file's first disk block. Initially, the pointer is initialized with a null value which shows that it is an empty file. A write to a file removes the first free block from the free block list and writes to that block. Then the address of this new block is associated with the last block of the file and then linked to this block at the end of the file. To read a file, the pointers are used by a followed pointer from block to block. The problem of the **Contiguous File** allocation method i.e., external fragmentation is resolved in the linked allocation. Any free block can be utilized to fulfill a request. Notice also that there is no need to declare the size of a file when that file is created. A file will still grow as possible as there are free blocks.

The linked allocation has some disadvantages. The foremost problem is that it is inefficient to support direct access; it is effective only for sequential-access files. To locate the i^{th} block of a file, it should begin at the start of that file and follow the pointers till the i^{th} block is reached. Note that every access to a pointer needs a disk read.

Another severe problem is reliability. A bug in OS or disk hardware failure might result in the pointer being lost and damaged. The effect of which could be picking up a wrong pointer and linking it to a free block or into another file.

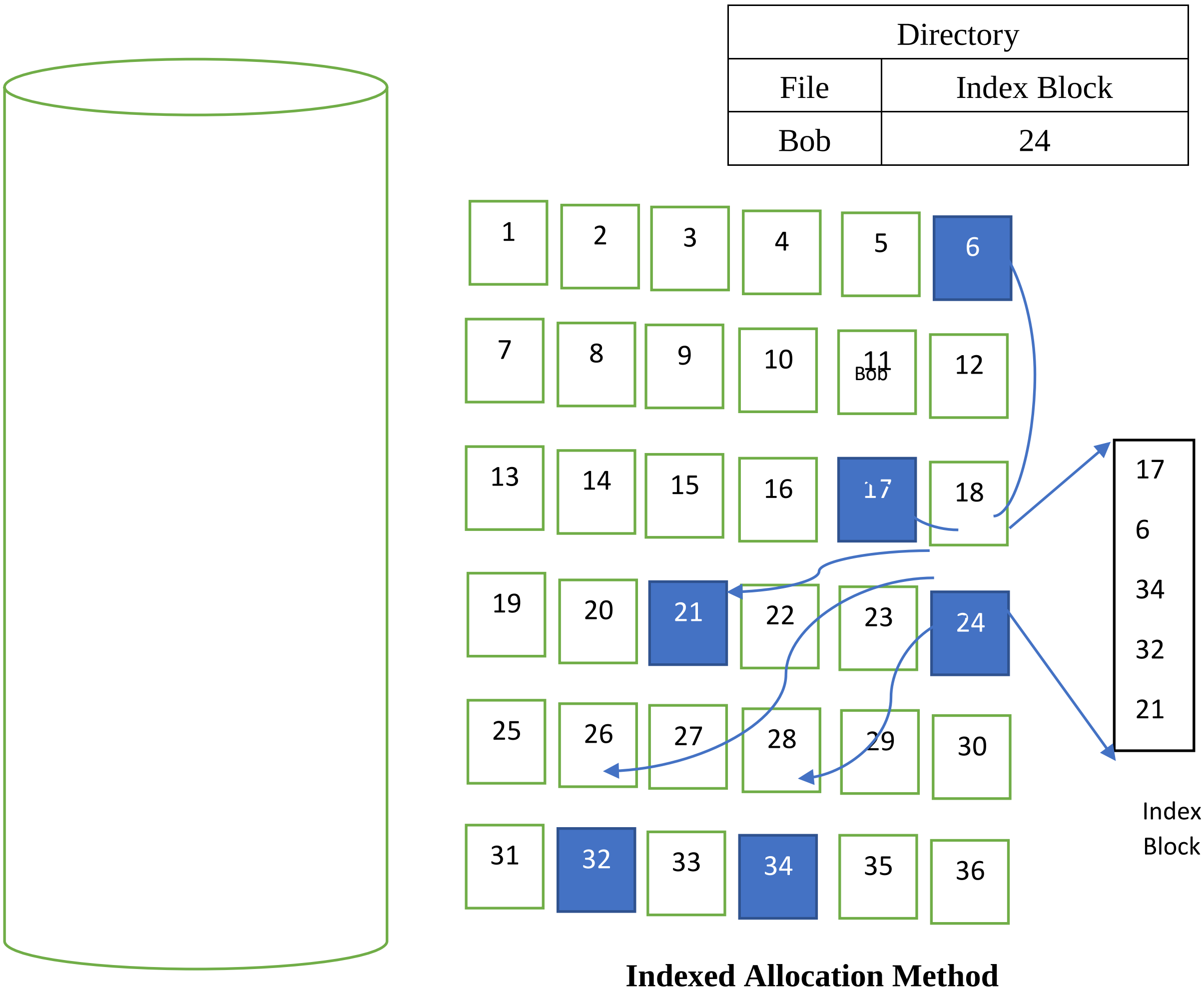
Characteristics

1. For a File there is one entry in the File allocation table size.
2. It's supports fixed size portions.
3. In this allocation Pre-allocation is feasible.
4. Allocation occurrence is Low to High.

Indexed Allocation Method

The problem of both contiguous and linked allocation is resolved in the indexed allocation method. In this allocation, one block is used as an Index block in which all the pointers bring together into one location.

Of course, the index block will occupy some space and thus could be considered as an overhead of the method. In indexed allocation, every file has its induvial index block, which is nothing an array of disk sectors of addresses. The j^{th} sector of the file is pointed by the j^{th} entry in the index block. The directory keeps the address of the index block of a file only. To read the j^{th} sector of the file, from the index block, the pointer in the j^{th} entry is read to locate the desires sector. Indexed allocation supports direct access, it resolves the external fragmentation. Any free block anyplace on the disk might satisfy a request for extra space.



Directory	
File	Index Block
Bob	24

Indexed Allocation Method

Characteristics

1. It supports both sequential and direct access.
2. There is no external fragmentation.
3. It does suffer wasted space.
4. Manage the Pointer is overhead.
5. It is faster than the other two approaches.

5.8 Free-Space Management

Since there is only a restricted amount of disk space, it is necessary to utilize the space from deleted files for new files. To keep track of free disk space, a free-space list is maintained by the system. This list keeps records of all disk blocks which are free. Whenever a file is created, the free space list has to be searched for the required amount of space and allocate space to a newly created file. After that from the free-space list that space is removed. Whenever a file is deleted then the free space list is updated by deleted file space added to the free-space list.

Bitmaps or Bit – Vector

In the bitmap method a table, called bit map is maintained for keeping track of the information about which part of memory is allocated to which process and which portion of memory is free. The bitmap is dynamic. This means the table is updated each time when a new process is allocated memory space, an existing process is swapped out of the main memory, a process is swapped in the main memory, or a process is completed and released from memory.

In the bitmap method, the memory is divided into certain equal size units or blocks. While allocating memory space to a program an OS can only allocate a complete unit of memory space and not a part of the unit. There is a cell in the bitmap for each unit of memory. If a unit is allocated to any process the value of the cell corresponding to the unit is set to 1 otherwise if the memory unit is free, the value is set to 0.

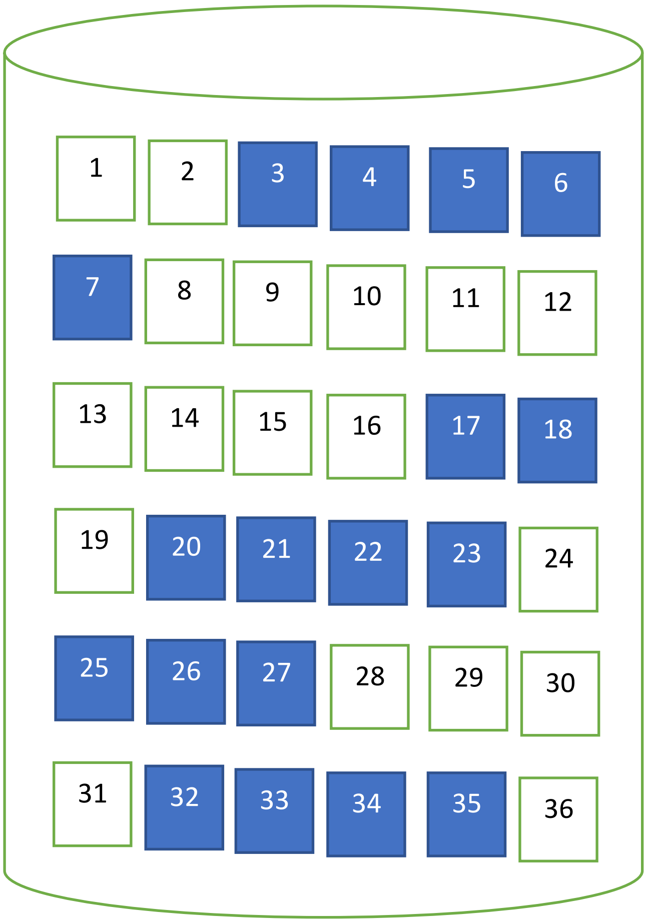


Figure Bit Vector

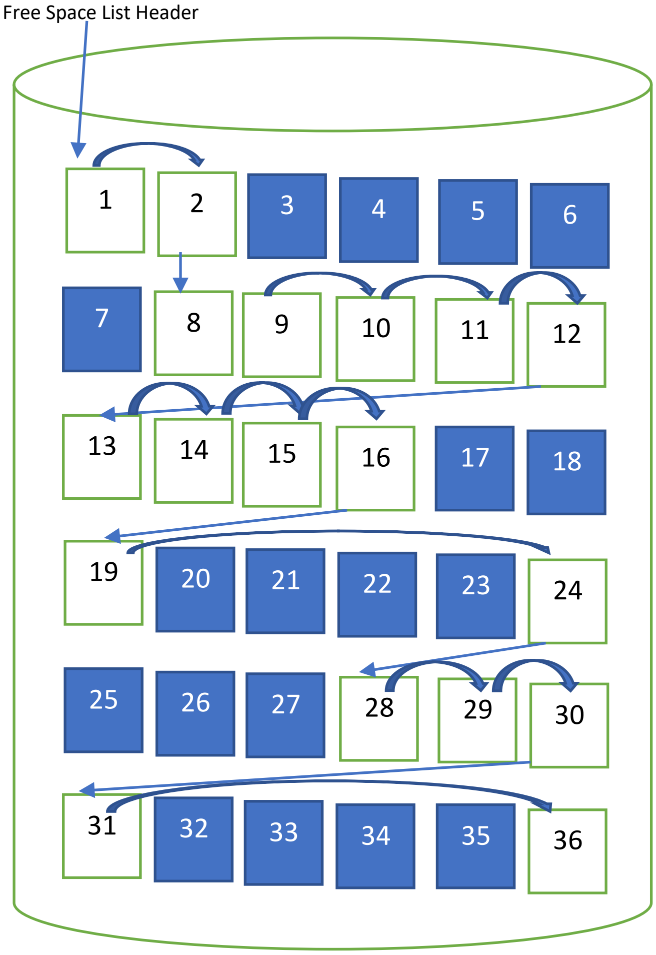
1, 2, 8, 9, 10, 11, 12,13,14,15,16,19,24,28,29,30,31,36 are flocks free so bit map of this example is 0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,1,1,0,1,1,1,1,0,1,1,1,0,0,0,0,1,1,1,1,0

There is a cell in the bit map corresponding to every memory unit. The value of the 3 cell in the bitmap is 1 because its corresponding memory unit is in use by the process. Value of the first and second cell is 0, because it's free.

An **advantage** of the bitmap method is that it is very simple to implement. On the other hand, the **disadvantage of this method is** it is very slow. OS needs much time to search for a hole, when allocating memory space to a new process or an existing process swapped in.

Linked List

In this method, a pool or chain of nodes is created to keep track of memory that is free and maintain a linked list i.e. the first block contains the address of the next free block and so on. The address of the very first free block is stored in a special node called as START node. The last block of the linked list contains NULL, which implies that there is no other free block.



Link Free Space List on Disk

For accessing the information in the linked list, it is loaded into physical memory. Whenever any allocation is to be done, blocks are removed from the head of the list and allocated. It occupies a large space. Traversing is also time-consuming. The following list shows a free block list.

Grouping

A modification of the free-list approach, in the first free block, keeps the addresses of n free blocks. There are only the first n-1 blocks that are free. The last one is the disk address of another block containing addresses of another n free block. The importance of this implementation is that a block contains the addresses of a larger number of free blocks so it can be found quickly if we need the number of free blocks.

Example: Block 1 Contains the address of Block 2. Block 2 contains the address of the first block of the next group block i.e., 8, Block 8 stores the address of 9 to 16 and the last block is 16 so it contains the address of the first block of the next group 19. As 19 is a single free block so, it contains the next group address i.e. 28. Block 28 keeps the address of 29 and 30. Block 30 contains the address of 31 and again Block 31 is the only single block in Grouping so it contains the address of 36. 36 is the last block os it stores nothing

Counting

Another method is to take benefit of the fact that, generally, many contiguous blocks may be allocated or freed contiguously, mainly when we used the contiguous allocation method. Thus, rather than keeping the entire list of free disk addresses, we noted the address of the first free block along with it the number n of free contiguous blocks which follow the first block. Each entry in the free-space list then contains the disk address along with the count. Although each entry needs more space as compared to a simple disk address, the overall list will be shorter, as long as the count is generally greater than 1.

First Free Block	Count
1	2
8	9
19	1
28	3
31	1
36	1

Points to Remember

The file system must identify and locate the selected file before operation on a file.

The file control block contains information about the file.

A disk can be divided into multiple partitions.

The file can be accessed by

— Sequential Access

— Direct Access

— Indexed Access

Three major methods of allocating disk space are widely used which are

Contiguous Allocation Method

Linked Allocation Method

— Indexed Allocation Method

The directory can be implemented in two ways:

— Linear List

— Hash Table

Free space management techniques are:

— Bit Vector

— Linked List

— Grouping

— Counting

5.9 PRACTICE EXERCISES

1. What is a File system?
2. Write a note on file attributes and naming?
3. What is File Organization?
4. Discuss the various methods of file allocation?
5. What are the disadvantages of the contiguous file allocation method?
6. What is the difference between linked allocation and indexed allocation methods?
7. What is a directory?
8. What are the different levels of the directory structure?
9. Discuss the various file systems?
10. Discuss the concept of free space management?
11. What are the various methods of Free Space Management?

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT VI: INTRODUCTION TO LINUX

STRUCTURE

6.0 OBJECTIVES

- To Understand Linux's shell, Kernel, and file system of Linux.
- Introduction to different types of directories: Parent, Subdirectory, Home directory; rules to name a directory, Important directories in Linux File System

6.1 INTRODUCTION TO LINUX

Linux which is pronounced as Lin-nucks is an operating system similar to UNIX. Some people also called it as descended form of UNIX. Linux is first released in 1991 by Linus Torvalds at the University of Helsinki. Since then it has gain huge popularity among the programmers. The Linux is an open-source operating system. It means it comes with the source code, so that one can change and customize the operating system according to the requirements. By the term open source it is meant that

- The independence to run the program in operating system for any function.
- The freedom to study any program that how it works and to alter it to make it work according to your need.
- The liberty to redistribute copies of the program to facilitate your neighbours.
- The free will to share copies of your modified versions to others for use.

Initially, Linux was created for personal computers and gradually it was used in other machines like servers, mainframe computers, supercomputers, etc. Nowadays, Linux is also used in embedded systems like routers, automation controls, televisions, digital video recorders, video game consoles, smart watches, etc. The biggest success of Linux is Android (operating system) it is based on the Linux kernel that is running on smart phones and tablets. Due to android Linux has the largest installed base of all general-purpose operating systems. Linux is generally packaged in a Linux distribution.

6.2 LINUX DISTRIBUTIONS

Linux has a numeral of dissimilar versions to ensemble any kind of user. Roughly all the editions of the Linux can be downloaded free of charge. These editions are labelled allocations (or, in the small form, “distros”). There are additional 600 distributions of Linux**.** Some of the general distributions of the Linux are:

- LINUX MINT
- MY LINUX
- MANJARO
- DEBIAN
- UBUNTU
- DEEPIN

- ANTERGOS
- ARCH LINUX
- GENTOO LINUX
- KALI LINUX
- SOLUS
- FEDORA
- ELEMENTARY OS
- OPENSUSE

Every distribution has a dissimilar take on the desktop. A small amount of the Server distributions of the Linux comprises:

- Red Hat Enterprise Linux
- Ubuntu Server
- Centos
- SUSE Enterprise Linux

A few of the above server distributions are at no cost (such as Ubuntu Server and CentOS) and a little have an associated cost (such as Red Hat Enterprise Linux and SUSE Enterprise Linux). Those with an associated price also comprise support.

6.3 CHARACTERISTICS OF LINUX OPERATING SYSTEM

Following are a few of the significant characteristics of Linux Operating System.

- **Portable** – Portability means software be able to work on dissimilar types of hardware in similar way. Linux kernel and application programs sustain their setting up on any kind of hardware stage.
- **Open Source** – Linux source code is liberally accessible and it is community based growth project. Multiple teams carry out task in collaboration to augment the potential of Linux operating system and it is constantly evolving.
- **Multi-User** – Linux is a multiuser scheme means numerous users can access network resources like memory/ ram/ application programs at similar time.
- **Multiprogramming** – Linux is a multiprogramming arrangement means numerous functions can run at similar time.
- **Hierarchical File System** – Linux offers a standard file arrangement in which organization files/ user files are ordered.

- **Shell** – Linux provides a particular interpreter program that can be employed to carry out commands of the operating system. It can be employed to do a variety of kinds of processes, call application programs etc.

- **Security** – Linux offers user security by means of authentication characteristics like password protection/ controlled admittance to particular files/ encryption of information.

Linux is fast, free and easy to use, power laptops and servers around the world. Linux has numerous more characteristics to amaze its user such as:

- **Live CD/USB:** Approximately all Linux distributions have Live CD/USB characteristic by which user can run/try the OS even with no mounting it on the scheme.

- **Graphical user interface (X Window System):** People think that Linux is a command line OS, wherever its precise also but not essentially, Linux have packages that can be mounted to construct the whole OS graphics based as Windows.

- **Support's** most national **or customized keyboards:** Linux is engaged worldwide and consequently obtainable in many languages, and supports on the entire of their custom national keyboards.

- **Application Support:** Linux has its own software ordnance through which users can download and set up thousands of desires just by concerning a command in Linux Terminal or Shell. Linux can3 besides run Windows demand if essential.

6.4 HISTORY OF LINUX OPERATING SYSTEM

The History of Linux commenced in 1991 with the beginning of a personal project by a Finland student Linus Torvalds to create a new open operating scheme kernel. From then, the resulting Linux kernel has been perceptible by fixed expansion during history.

- In the year 1991, Linux was started by a Finland student Linus Torvalds.
- Hewlett Packard UNIX(HP-UX) 8.0 was enlightened.
- In the year 1992, Hewlett Packard 9.0 was liberated.
- In the year 1993, NetBSD 0.8 and FreeBSD 1.0 liberated.
- In the year 1994, Red Hat Linux was initiated. Caldera was originated by Bryan Sparks and Ransom Love and NetBSD1.0 liberated.
- In the year 1995, FreeBSD 2.0 and HP UX 10.0 were liberated.
- In the year 1996, K Desktop Environment was initiated by Matthias Ettrich.
- In the year 1997, HP-UX 11.0 was liberated.
- In the year 1998, the fifth invention of SGI Unix i.e IRIX 6.5, Sun Solaris 7 operating system, and Free BSD 3.0 was liberated.
- In the year 2000, the conformity of Caldera Systems with the SCO server software division and the professional services division was declared.
- In the year 2001, Linus Torvalds liberated the Linux 2.4 edition source code.

- In the year 2001, Microsoft ordered a trademark suit besides Lindows.com
- In the year 2004, Lindows name was altered to Linspire.
- In the year 2004, the first liberate of Ubuntu was discharged.
- In the year 2005, The project, open SUSE begin a complimentary allocation from Novell's community.
- In the year 2006, Oracle liberated its own allocation of Red Hat.
- Dell initiated allocated laptops with Ubuntu pre-installed in it, in the year 2007.
- The Linux kernel 3.0 version was liberated in the year 2011.
- Google Linux-based Android declared 75% of the smart phone market share, in form of the number of phones dispatched, In the year 2013
- Ubuntu asserted 22,000,000 users in the year 2014.

6.5 ARCHITECTURE OF LINUX

Linux architecture has the subsequent components:

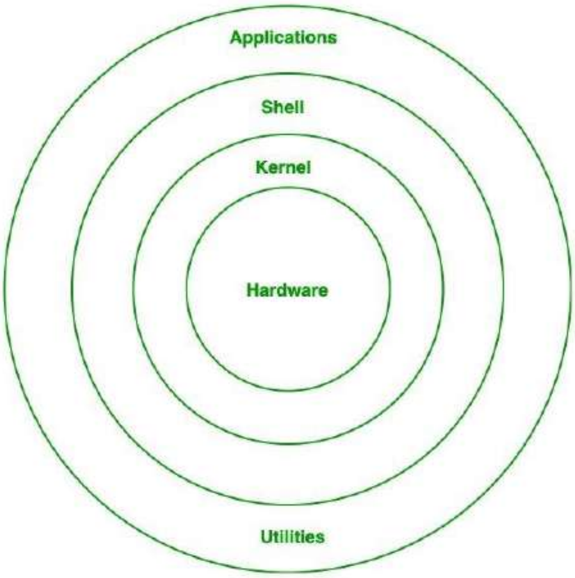


Figure 6.1: Architecture of Linux Operating System

I. **Kernel:** Kernel is the heart of the Linux supported operating system. The kernel is the mainly significant part or mind of every operating system as it handles the conversation among a machine's hardware and its software. In easy words, the kernel acts as a spirit of the network and handles the memory, peripheral apparatus, and CPU. The kernel places at the "lowest" point of the OS. This creates the method seem as if it is the only procedure running on the device. The kernel too well is accountable for avoiding and justifying clashes among diverse processes. Particular kernel types are as

   - Monolithic Kernel,
   - Hybrid Kernels,
   - Micro Kernels and

- Kernels from Exo.

A few of the key design ethics executed by Linus kernel are:

- In Linux, the whole thing is a file philosophy
- Multi-users ability
- Multitasking capability
- Portability among GPU architecture
- Modularity
- Security
- Configurability

II. **Shell:** It is an interface to the kernel which covers the difficulty of the kernel's tasks from the client. It takes instructions from the user and carry out the kernel's functions. Linux include two kinds of command keys: text mode comparable to those originated in mainly UNIX systems (example the bourne shell, the bourne over shell, the c shell, the turbo C shell and the korn shell) and graphical user interface (GUI) such as the KDE (K Desktop Environment) and GNOME (GNU Network Object Model Environment). **Terminal** is the purpose that brings it all collectively, in the sense that it offers a visual illustration of the shell for the client to penetrate commands. In other words, in a GUI (graphical user interface), where requests and other descriptions are visually symbolized by images that the client can manoeuvre by ticking on them with a cursor, a terminal request release a window where the client can type in instructions for the shell to infer into binary communication for the kernel.

III. **System Library:** It is the exceptional kind of tasks that are used to execute the functionality of the operating system. These pieces carry out the background facility of the Linux operating system similar to scheduling, printing, sound, etc that each start-up during boot of the operating scheme, or after you register into your system.

IV. **Hardware Layer:** This layer comprises of all the peripheral apparatus like RAM/ HDD/ CPU etc. Some of the substantial hardware that is incorporated in any tool comprises of mouse, keyboard, the graphics chipset, display, and, but you do have one, your system interface card

V. **System Utility:** It offers the functionalities of an operating system to the client. A Linux-based scheme will typically appear with such a set of established Unix-like utility services; these were all normally basic tasks used in day-to-day operating system procedures, and also precise devices and requests. This is frequently software that was available and liberated under the open-source license by the GNU Project, consequently the software can be explicitly downloaded, restructured, and reallocated to all.

Advantages of Linux

The main benefit of Linux, is it is an open-source operating system. This means the source code is simply obtainable for everybody and you are authorized to give, alter and allocate the code to anybody without any consent.

- In terms of security, Linux is further secure than any other operating scheme. It does not signify that Linux is 100 percent safe it has a little malware for it but is fewer susceptible than any other operating scheme. So, it does not necessitate any anti-virus software.
- The software renewed in Linux are simple and recurrent**.**
- A variety of Linux distributions are accessible so that you be able to utilize them according to your desires or according to your flavour.
- Linux is liberally available to employ on the internet.
- It has huge community support.
- It offers high constancy. It seldom slows down or freezes and there is no necessitating rebooting it after a small time.
- It preserves the privacy of the client.
- The presentation of the Linux scheme is much elevated than other operating scheme. It permits a large quantity of people to employee at the similar time and it grips them resourcefully.
- It is network pleasant.
- The flexibility of Linux is elevated. There is no necessitate to establish a complete Linux suit; you are authorized to establish only required apparatus.
- Linux is companionable with a huge number of file set-ups.
- It is quick and simple to fit from the web. It can also mount on any hardware even on your previous computer structure.
- It carry out all tasks correctly even if it has partial liberty on the hard disk.

Disadvantages of Linux

- It is not extremely user-friendly. So, it might be puzzling for beginners.
- It has tiny marginal hardware drivers as contrast to windows.

6.6 Widows Vs Linux

Few Differences between Linux and Windows

S.NO	Linux	Windows
1.	Linux is a open source operating scheme.	While windows are the not the open source operating scheme.
2.	Linux is at no cost.	While it is expensive.

S.NO	Linux	Windows
3.	It's file name case-sensitive.	While it's file name is case-insensitive.
4.	In linux, monolithic kernel is employed.	While in this, micro kernel is employed.
5.	Linux is more competent in comparison of windows.	While windows are fewer competent.
6.	There is forward slash is employed for separating the directories.	While there is back slash is employed for isolating the directories.
7.	Linux offers more protection than windows.	While it offers less safety than linux.
8.	Linux is extensively employed in hacking point based systems.	While windows does not offers much competence in hacking.

6.7 UNIX VS LINUX

Now, we will observe what is the distinction between linux and unix:

Key Differences	Linux	Unix
Cost	Linux is liberally dispersed, downloaded during magazines, Books, website, etc. There are waged versions also accessible for Linux.	Diverse flavors of Unix have dissimilar pricing depending upon the kind of vendor.
Development	Linux is Open Source, and thousands of programmers work together online and supply to its expansion.	Unix systems have dissimilar versions. These editions are primarily created by AT&T as well as other business vendors.
User	Everybody. From home customer to developers and computer aficionado alike.	The UNIX can be employed in internet servers, workstations, and PCs.
Text made	BASH is the Linux evasion shell. It presents support for numerous command	Initially prepared to job in Bourne Shell. Though, it is now

Key Differences	Linux	Unix
interface	interpreters.	well-suited with numerous others software.
GUI	Linux offers two GUIs,viz., KDE and Gnome. Although there are numerous substitutes such as Mate, LXDE, Xfce, etc.	General Desktop atmosphere and also has Gnome.
Viruses	Linux has had concerning 60-100 virus listed today which are at present not scattering.	There are among 80 to 120 viruses informed till date in Unix.
Threat detection	Threat finding and way out is extremely quick since Linux is chiefly community driven. So, if any Linux clients post any type of threat, a team of competent developers commence working to overcome this risk.	Unix client necessitate longer wait instance, to acquire the appropriate bug fixing patch.
Architectures	Originally urbanized for Intel's x86 hardware processors. It is obtainable for over twenty dissimilar kinds of CPU which also comprises an ARM.	It is obtainable on PA-RISC and Itanium machines.
Usage	Linux OS can be fitted on a variety of kinds of plans like mobile, tablet computers.	The UNIX operating scheme is employed for internet servers, workstations & PCs.
Best feature	Kernel renewed with no reboot	Feta ZFS - next age group file system DTrace - dynamic Kernel Tracing
Versions	Dissimilar editions of Linux are OpenSuse ,Redhat, Ubuntu, etc.	Dissimilar editions of Unix are BSD ,HP-UX, AIS, etc.
Supported file type	The File arrangement maintained by file kind like devpts, xfs, nfs, cramfsm ext 1 to 4, ufs, NTFS.	The File arrangement sustained by file kinds are vxfs, zfs, hfx, GPS, xfs.
Portability	Linux is moveable and is booted as of a USB Stick	Unix is not moveable

Key Differences	Linux	Unix
Source Code	The source is obtainable to the common public	The source code is not accessible to everyone.

6.8 TYPES OF FILES

In Linux and UNIX, the whole thing is a file. Directories are records; files are files, and tools like keyboard, Printer, mouse, etc. are files.

Let's look into the File types in more detail.

General Files

General Files also called as Ordinary files. They can include image, video, program or simply text. They can be in ASCII or a Binary format. These are the most commonly used files by Linux Users.

Directory Files

These files are a warehouse for other file types. You can have a directory file within a directory (sub-directory).You can take them as 'Folders' found in Windows operating system.

Device Files

In MS Windows, devices like Printers, CD-ROM, and hard drives are represented as drive letters like G: or H:. In Linux, these are represented as files. For example, if the first SATA hard drive had three primary partitions, they would be named and numbered as /dev/s*da1*, /dev/sda2 and /dev/sda3.

Note: All device files reside in the directory /dev/

All the above file types (including devices) have permissions, which allow a user to read, edit or execute (run) them. This is a powerful Linux/Unix feature. Access restrictions can be applied for different kinds of users, by changing permissions.

6.8.1 Linux Directory Structure

In **Linux**, everything is a file. A directory is nothing but a special file that contains details about all the files and subdirectories housed within it. The Linux contains following directories for different purpose:

i. **Root Directory (/)**
The **root directory** also represented as “ / “ is the topmost level of the system drive. Every other directory in your Linux system is located under the root directory. You may imagine the / directory similar to the C:\ directory or drive on your Windows system. This is not at all true but you may understand it as imaginary vision because

Linux doesn't have drive letters. It is commonly used to represent a filesystem. A filesystem is the hierarchy of directories that is used to organize directories and files on a computer.

ii. **Essential User binaries (/bin)**

This folder contains the essential user programs which are also called as binaries whichare present when the system is mounted in single-user mode. It also contain the files for common commands likes ls, cat, cd, cp etc. The '/**bin**' **directory** also contains executable files

iii. **Static Boot Files (/boot)**

It contains the files which are required to boot the system such Linux kernel files, GRUB configuration files.

iv. **Historical mount point for CD-ROMs (/cdrom)**

/mnt/cdrom and /cdrom are the mount point directories for the CD-ROM drive. The /cdrom directory isn't part of the Filesystem Hierarchy Standard (FHS), but you'll still find it on Ubuntu and other operating systems.

v. **Device Files (/dev)**

In Linux, the /dev directory contains device and other special files. To see a list of these items, run the following command in a terminal session.

```
ls –l /dev
```

These include terminal devices, usb, or any device attached to the system.

vi. **Configuration Files (/etc)**

The /etc/ directory contains system-wide configuration files — user-specific configuration files are located in each user's home directory.

vii. **Home Folder (/home)**

The home directory for each user takes the form /home/username (where username is the name of the user account). For example, if your user name is john, the system will have a home folder located at /home/john

viii. **Essential Shared Libraries (/lib)**

The lib folder is a library files directory which contains libraries needed by the essential binaries to be used by the system. These are supportive files which are used by an application or a process for their proper execution. The commands in /bin or /sbin dynamic library files are located just in this directory.

ix. **Recovered Files (/lost+found)**

Whenever a there is crashed in the file system, the system boots again and check is performed. Any file which got corrupted by crash will be found in lost+found folder.Every disk partition has a lost+found directory.

x. **Removable Media (/media)**

This Directory is used for mounting files systems on removable media like Zip drives, floppy drives, CD-ROM drives.

xi. **Temporary Mount Points (/mnt)**

This directory is used for temporarily mounted filesystem. For example, if you're mounting a FAT partition for some operations, it might be mounted at /mnt/fat.

xii. **Optional Packages (/opt)**

It contains subdirectories for optional software packages which are written as a result of copy/install operations.

xiii. **Kernel and Process Files (/proc)**
It is a special directory for virtual filesystem. Instead of standard files it contains special files that represent system and process information.

xiv. **Root Home Directory (/root)**
This directory is distinct from / directory which is used for the system root directory. It is the home directory for root user. Instead of using the /home/root, it uses the location at /root.

xv. **Application State Files (/run)**
This folder contains data which describe the system since it has been booted. It gives applications a standard place to store temporary/transient files containing data related to sockets and process Ids.

xvi. **System Administration Binaries (/sbin)**
It contains files related to administrative commands. This directory includes essential binaries that might be needed to run by the root user for system administrator of the system.

xvii. **SELinux Virtual File System (/selinux)**
Ubuntu doesn't use SELinux. The /selinux directory contains special files used by SELinux for security (example Fedora and Red Hat). It is similar to /proc folder.

xviii. **Service Data (/srv)**
This holds the data for system services like Apache HTTP, FTP, etc.

xix. **Temporary Files (/ tmp)**
When the system or user executes the applications, the temporary files are stored in this folder. These temporary files are usually deleted automatically when the system gets started again.

xx. **User Binaries and Read Only Data (/usr)**
This directory is meant to contain the applications and files which belong to users, dissimilar to applications and files used by the system.

xxi. **Variable data Files (/var)**
This folder holds numerous system files which may be related to log, mail directories, print spool, etc. that may change over time in numbers and size.

6.8.2 Parent, Subdirectory, Home Directory

Root Directory (/) holds the top position as the main directory. All the directories mentioned in above section from sr. No. 2 to sr. No. 21 are the subdirectory of the root directory. /Home and /root are the home directories for the normal user and root user respectively.

6.8.3 Naming Rules for Directory and Files

1. The file names in the Linux are case sensitive. For example a filenamesarea.txt, Area.txt and AREA.txt all are considered as different files.

2. The filename can be created either by using uppercase letters, lowercase letters, numeric numbers, "_" (underscore) and "." (dot) symbols.

3. In Linux, other special characters such as blank space can also be used, but the rules for using other symbols are hard and it is heal their not to use them.

4. As mentioned in above point filenames may contain any character but one cannot use slash sign“/” (root directory), because it is reserved character and can only be used as the separator between files and directories while mentioning the pathname. One cannot use the null character.

5. .(dot) sign should not be used in the filename. Some time dot sign increases the readability of filenames but this may add confusion in understanding the extension of files. Dot sign should be used to identify the extension of the file. For example:

   - .tar.gz = zipped or Compressed files

   - .sh = used to identify the shell file

6. Older versions of the Unix operating system uses filenames upto 14 characters long. But in today’s era latest version of Linux and UNIX can use filename length upto 255 characters (255 bytes).

7. A filename should be unique in a particular directory. For example, in directory /home/john, one cannot give two file name as try.txt and try.txt. However, different directory can have same filenames inside them. For example, you can create try.txt in /tmp directory and try.txt inside /home/john directory.

6.9 PRACTICE QUESTIONS

Q1.What are the basic components of Linux?
Q2.Write down the name of some Linux variants.
Q3.Which popular office suite is available free for both Microsoft and Linux?
Q4.Suppose your company is recently switched from Microsoft to Linux and you have some MS Word document to save and work in Linux, what will you do?
Q5.What is the difference between Linux and Unix?
Q6.What is Linux Kernel? Is it legal to edit Linux Kernel?
Q7.What is the difference between BASH and DOS?
Q8.What are the process states in Linux?
Q9.What is the advantage of open source?

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT VII: LINUX COMMANDS

STRUCTURE

7.0 OBJECTIVE

- Understanding Linux commands

7.1 OVERVIEW OF LINUX COMMANDS

The command line is one of the most powerful features of Linux. There are a countless number of commands in Linux. In this chapter, we will introduce few of most commonly and regularly used Linux commands for easy learning.

cal / ncal command

The **cal** command is used to print the calendar for a specified month and/or year on the monitor i.e. standard output device.

$cal: This will print the calendar for the current

$cal 2020: This command will print the twelve month calendar for the year 2020.

$cal 7 2020: This command will print the calendar July 2020.

mkdir command

This command is used to make the subdirectory under the current directory

$mkdir student: This command will create the directory student under the current directory

cd command

The cd command is used to change the current directory to the specified directory

$cd student: This command will change the current directory to student directory.

mv command

This command is used move the files from one folder to another folder.

$mv /home/john/try.txt /home/merry/. This command will move the try.txt file from /home/john directory to home/merry folder

$mv /home/merry/*.* /home/john/. This command will copy all the files in /home/merry folder to /home/john folder.

cp command

This command is used to copy the files from one folder to another folder.

$cp /home/john/try.txt /home/merry/. This command will copy the try.txt file from /home/john directory to home/merry folder

$cp /home/merry/*.cpp /home/john/. This command will copy all the files with extension .cpp from /home/merry folder to /home/john folder.

date command

The date command displays the current day, date, time, and year.

$ date: It will display current system date like “**Mon May 24 17:14:57 IST 2021**”

echo command

The echo command prints whatever you will write except quote. This command is used to insert the text on a file as shown below.

$echo "Welcome to The Linux Class": This command will print the line Welcome to The Linux Class.

rm command:

One can use “rm” command to delete any file or directory. To delete directories recursively we can use rm command along with -r , -R

$rm –r fname: This command will remove the folder with name fname.

$rm rea*.* : This command will remove all files starting with name prefix rea in current folder

rmdir command:

one can use rmdir to delete a directory. This command can only be used to delete an empty directory.

$rmdir dir_name: This command will remove the folder with name dir_name.

cat command:

cat is used to display the contents of the file whose name is mntioned along with the command. Example to see the text inside the file area.txt we may write the command as follows.

$cat area.txt: It will display the contents inside the filename area.txt

It is also used to merge the contents of the two files. The output is printed on the standard output device.

$cat filename1 filename2: It will display the contents of both the files together after merging.

pwd command

pwd command is used to display the name of present working directory. With the help of pwd command we are able to know about the directory in which we are working with. It gives us the absolute path, which means the path that starts from the root.

$pwd : It will the complete path of current directory. Example if you are working under user john under home directory then we will get output as /home/john.

who command

who command used to find out last system boot time, the system's current run level, List of the users who logged in the system and more. The different variations of who command are as follows:

Description	**Example**
The who command displays the following information for each user currently logged in to the system if no option is provided	$who
To display host name and user associated with standard input such as keyboard	$who -m –H
To show all active processes which are spawned by INIT process	$who -p –H
To show status of the users message as +, – or ?	$who -T –H
To show list of users logged in to system	$who –u
To show time of the system when it booted last time	$who -b –H
To show details of all dead processes	$who -d –H
To show system login process details	$who -l -H
To count number of users logged on to system	$who -q –H
To display current run level of the system	$who -r

To display all details of current logged in user	$who –a
To display system’s username	$whoami
To display list of users and their activities	$ w
To display user identification information	$ id

pwd command

pwd command is used to display the name of present working directory. With the help of pwd command we are able to know about the directory in which we are working with. It gives us the absolute path, which means the path that starts from the root.

$pwd : It will the complete path of current directory. Example if you are working under user john under home directory then we will get output as /home/john.

ls command

This is called list command. It is used to display the files stored in a particular directory.

ls: To view the brief, multi-column list of the files in the current directory.

ls –a: To also see files starting with symbol "dot" (configuration files that begin with a period, such as .login).

ls –la: To see the permissions, owners and size of the files along with their names.

ls -la | less: If the listing is enough long that it is not fit into one screen view then reading becomes difficult. This command combines ls with the less utility for the same.

bc Command

This command is used for arbitrary precision CLI calculator which can be used like this: **$ echo 30.07 + 16.00 | bc**

more Command

This command enables us to view the large text data one screen display at a time.

gzip Command

gzip command is used to compress a file, replaces it with a file with .gz extension. The example of usage of gzip command is as shown below:

```
$ gzip passwds.txt

$ cat filename1 filename2 | gzip > try.gz
```

tar Command

This powerful utility command is a used for archiving files in Linux as follows.

```
$ tar -czf  demo.tar.gz
```

7.2 FILE PERMISSIONS IN LINUX

Linux is a **multi-user operating system** which means it can be accessed by many users simultaneously. The multi-user concern of Linux imposes security concerns as an unsolicited or **malign user** can **corrupt, change or remove crucial data**. Linux categorizes authorization into 2 levels.

1. Ownership
2. Permission

Ownership in Linux

Every file and directory in Linux operating system can be assigned 3 different types of owner, given below.

User

The person who creates the file is by default the owner of the file. Such user is the owner of the file. Therefore, a user is also sometimes called an owner.

Group

A group or user- group consists of multiple users. All the users which belong to a same group will have the same access permissions to the file. For example there is a project where the number of people/users needs access to a file. You could add all users to a single group and assign the permissions to the group instead of assigning the permissions to each user individually. No one other than the group member can read or modify the files.

Other

In this category, any general user who can access to a file is included in other category. This person has neither produced the file, nor do he / she belong to a particular usergroup who could own the file. Practically, it means everybody else. Therefore, this category is also called as the set permissions for the world.

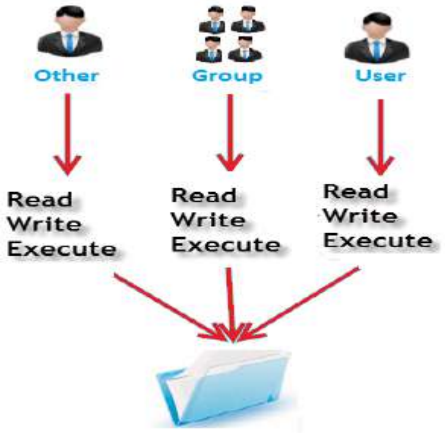
It is matter of the question that how Linux distinguishes between the three category of the users. One user 'A' cannot affect a file that contains the information of some other user 'B'. It means that you do not want another colleague in your organization to access your data. This is where the role of setting permissions comes in, and this also defines user behavior.

Let us understand the role of setting permissions in the Linux OS.

Permissions

Every file and directory in your Linux operating system has three types of permissions defined for all the 3 types of owners.

- **Read:** The read permission gives the authority to read as well as open the file. This permission on a directory gives you the capability to view its contents.
- **Write:** This permission gives a person the authority to modify the contents inside the file. You can rename, remove and add files present in the directory. For example if you don't have write permission on a folder but have the write permission on one of its file then you will be able to change the contents of the file but you will not be able to remove, move or rename the file placed inside the directory.
- **Execute:** Similar to Windows where files with ".exe" extension are executable. But in Linux OS, one cannot execute any file without having permission on it. If the execute permission is not granted on file, then you will not be able to execute the file. However you can see/modify the contents inside the file.



Categorization of Permissions that can be assigned to different types of Users

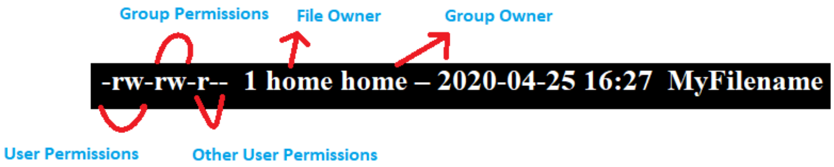
Let us see view this with the help of commands:

Issue command ls - l on terminal gives you the filenames with file types and access permissions.

ls – l

home$ ls –l

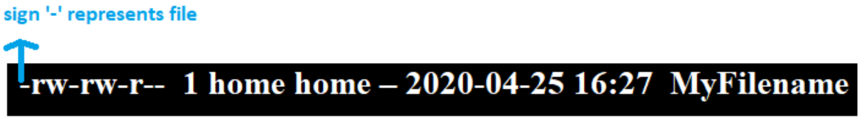
-rw-rw-r-- 1 home home – 2020-04-25 16:27 MyFilename



```
-rw-rw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

The string **'-rw-rw-r--'** in the output describes about the permissions given to the owner, user group and the world.

Here, the first '-' represents that this line indicates information about file

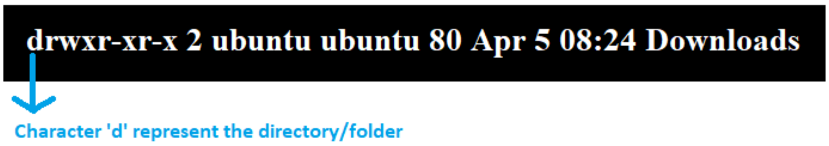


```
-rw-rw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

If it could have been a directory then **d** would have been shown as below:

drwxr-xr-x 2 ubuntu ubuntu 80 Apr 5 08:24 Downloads

In the above output, the first character 'd' represent the directory.



```
drwxr-xr-x 2 ubuntu ubuntu 80 Apr 5 08:24 Downloads
```

The characters shown in the above example are easy to remember.

- = have no permission
x = the permission to execute
w = the permission to write
r = the permission to read
Let us explore the meaning of the output as follows:

The first part in the output line is **'rw-'**. This indicates that the owner 'home' can have two permissions i.e. read and write the file but the owner is not having the permission to execute that is why sign '-' has been used.

However, many distributions of the Linux like Ubuntu, CentOs and Fedora etc. can add users to the same group name which is the name of the user. Therefore a username 'john' may be added to a group having name as 'john'.

The second part of the output is **'rw-'** which indicates that group having named as '**home'** have the permission to read and write the file but don't have the permission to edit the file**.**

Third part is used to represent the permissions for the other person from the outside world. It means there could be any other user. The string **'r--'** indicates that other users have the permission to only read the file but these persons may not be able to write and execute the file.

Using 'chmod' command to change the directory/file permissions

Permissions help you to restrict the others in performing operations on files/directories in form of read, write and execute. This can be accomplished by changing permissions on files/directories.

We can use the '**chmod**' command which stands for 'change mode'. Using the command, we can set permissions (read, write, execute) on a file/directory for the owner, group and the world. The Syntax of command is as follows:

chmod permissions filename

There are 2 types of modes for using chmod command -

1. **Absolute mode**
2. **Symbolic mode**

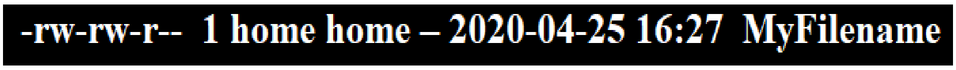
Absolute Mode / Numeric Mode

In this technique/mode, the permissions on files are represented as three-digit octal number instead as characters. The following table represents the different numbers for each type of permissions.

Number	**Permission Type**	**Symbol**
0	No Permission	---
1	Execute	--x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r--
5	Read + Execute	r-x
6	Read +Write	rw-
7	Read + Write +Execute	Rwx

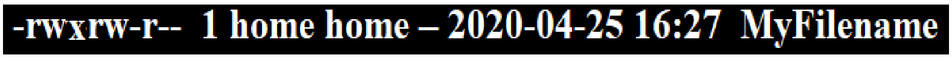
Let us look at the examples of using chmod command.

Command **"ls –l MyFilename"** shows the current permissions on the MyFilename



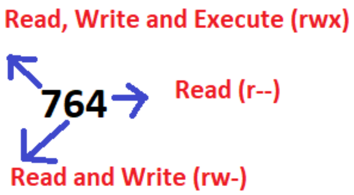
```
-rw-rw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

If we issue the command **“chmod 764 MyFilename”** then we will see the following results



```
-rwxrw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

In the above-given terminal window, we have changed the permissions of the file 'sample to '764'.



'764' absolute coding can be interoperated as follows:

- Owner can read, write and execute
- User group can read and write
- World can only read

This can be seen in the output as '-rwxrw-r-

In this way; you can change the permissions on file by assigning an absolute number.

Symbolic Mode

In the Absolute mode, you change permissions for all 3 owners. In the symbolic mode, you can modify permissions of a specific owner. It makes use of mathematical symbols to modify the file permissions.

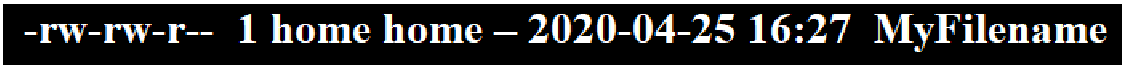
Operator	Action of applying the operator
+	Adds the particular permission to the file / directory
-	Removes the particular permission from file / directory
=	Sets the new permission by overriding the previous permissions.

The various types of owners of file and directories are given as follows:

User Denotations	
U	user/owner
G	Group
O	Other
A	All

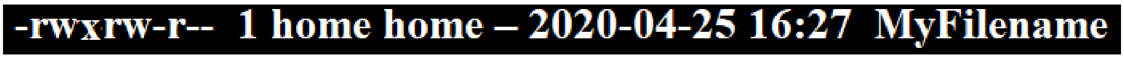
In symbolic mode we shall be setting permissions by making use of characters like rwx unlike numbers in absolute mode eg 764. Let us apply example for symbolic mode using the previous file permissions on MyFilename:

Command **“ls –l MyFilename”** shows the current permissions on the MyFilename



```
-rw-rw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

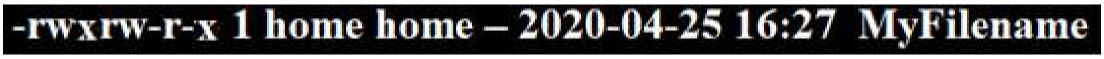
After applying the command **“Chmod u = rwx MyFilename”**, we shall get the following result:



```
-rwxrw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

In above command “u” stands for user. It means new permissions will be set for the user. The command will assign the user permissions of read, write and execute to the MyFilename.

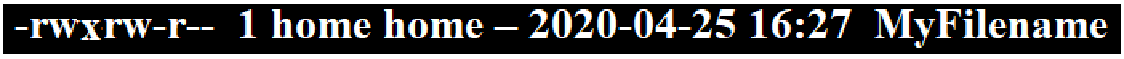
Let us apply another command as **“chmod o+x MyFilename”**. Then the permissions will be changed as follows.



```
-rwxrw-r-x 1 home home – 2020-04-25 16:27  MyFilename
```

String **“o+x”** in above command will add the execute permission for the other user.

Let us again apply command as **“chmod o-x MyFilename”**. Then the execute permission which was given just now will be removed. The output of **“ls –l MyFilename”** command will look as follows after applying the command.



```
-rwxrw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

7.3 CHANGE IN OWNERSHIP AND THE ASSOCIATED GROUP

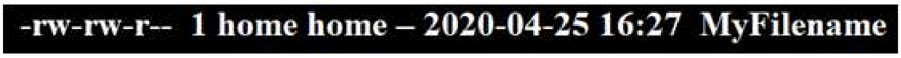
To change the ownership of particular file/directory, we may use the **“chown”** command as follows:

chown new_user_name

If there is need to change the group along with the user for a file or directory, then we may use the following command

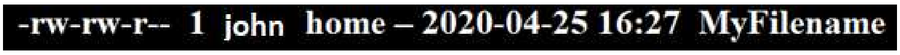
chown new_user_name:new_group_name filename

Let us look at the output of **"ls – l MyFilename"** command



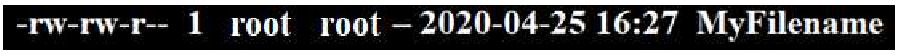
```
-rw-rw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

Now apply the "**chown john MyFilename"** command and then apply **"ls – l MyFilename"** command. It you are not logged in as root user then you may use **"sudo chown john MyFilename"** command to act as super user. The system may ask you the password for the super user in such case.



```
-rw-rw-r--  1  john  home – 2020-04-25 16:27  MyFilename
```

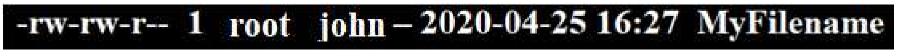
Let us change the user and group both to root by applying **"sudo chown root:root MyFilename"** command. The output of the file may look as follows after applying the command.



```
-rw-rw-r--  1   root   root – 2020-04-25 16:27  MyFilename
```

If you want to change only the group of the user then may also apply **"chgrp"** command. Command "**chgrp"** stands for change group.

Let us apply the command **"sudo chgrp john MyFilename"** to change the group from root to john. Then we will get output as follows:



```
-rw-rw-r--  1   root   john – 2020-04-25 16:27  MyFilename
```

<u>7.4 ADVICE BEFORE USING COMMANDS</u>

- The file stored in the **/etc/group** contains the name of all the groups defined in the system at any moment of time.
- We may use the command **"groups"** on terminal to find the name of all the groups where we are a members.
- We may use the command **"newgrp"** to work as a member a group other than your default group
- No file/directory can own to two groups simultaneously.
- Linux do not have nested groups.
- x- eXecuting a directory means Being allowed to "enter" a dir and gain possible access to sub-directories.

7.5 SUMMARY

- Linux being a multi-user system uses permissions and ownership for security.
- There are three user types on a Linux system viz. User, Group and Other
- Linux divides the file permissions into read, write and execute denoted by r,w, and x
- The permissions on a file can be changed by 'chmod' command which can be further divided into Absolute and Symbolic mode
- The 'chown' command can change the ownership of a file/directory. Use the following commands: chown user file or chown user:group file
- The 'chgrp' command can change the group ownership **chrgrp group filename**
- What does x - executing a directory mean? A: Being allowed to "enter" a dir and gain possible access to sub-dirs.

7.6 PRACTICE QUESTIONS

Q1.What are the basic commands for user management?

Q2.Which command is used to uncompressed grip files?

Q3.What are the modes used in VI editor?

Q4.What are the file permissions in Linux?

Q5.Which are the Linux Directory Commands?

Q6.What are inode and process id?

Q7.Explain Process Management System Calls in Linux

Q8.Explain the redirection operator.

Q9.How to copy a file in Linux?

Q10. How to terminate a running process in Linux?

Q11. How to create a new file or modify an existing file in vi?

Q12. Explain Regular Expressions and Grep

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT VIII: SHELL SCRIPTING

STRUCTURE

8.0 OBJECTIVE

To learn the usefulness of shell scripting

8.1 INTRODUCTION TO SHELL

- The shell is a user program or it is an environment provided for user interaction.
- It is a command language interpreter that executes commands read from the standard input device such as keyboard or from a file.
- The shell gets started when you log in or open a console (terminal).
- Quick and dirty way to execute utilities.
- The shell is not part of system kernel, but uses the system kernel to execute programs, create files etc.

Different shells are available in Linux which are as under:

- **BASH (Bourne-Again SHell)** - Most common shell in Linux. It's Open Source.
- **CSH (C SHell) -** The C shell's syntax and usage are very similar to the C programming language.
- **KSH (Korn SHell) -** Created by David Korn at AT & T Bell Labs. The Korn Shell also was the base for the POSIX Shell standard specifications.
- **TCSH** - It is an enhanced but completely compatible version of the Berkeley UNIX C shell (CSH).

8.2 SHELL SCRIPTING

A shell script is just a normal Linux file which contains Linux and shell commands. The simplest shell scripts simply group together commonly used sequences of commands.

More complex scripts use the shell's programming syntax to perform more advanced tasks. Shell scripts are not compiled but interpreted. This means that each time they are run a shell is executed to read the file and run the commands it contains. Languages which are compiled, such as C, will produce a compiled code which will run faster than an equivalent shell script.

Shell scripts usually begin with a #! and a shell name.
For example: #!/bin/sh
If they do not, the user's current shell will be used

8.3 SHELL PROMPT

There are various ways to get shell access:

- **Terminal** - Linux desktop provide a GUI based login system. Once logged in you can gain access to a shell by running X Terminal (XTerm), Gnome Terminal (GTerm), or KDE Terminal (KTerm) application.
- **Connect via secure shell (SSH)** - You will get a shell prompt as soon as you log in into remote server or workstation.
- **Use the console** - A few Linux system also provides a text-based login system. Generally you get a shell prompt as soon as you log in to the system.

You may type the following into the Terminal to find out the shells available in your system.

cat /etc/shells

Why write shell scripts?

– To avoid repetition: If you do a sequence of steps with standard Linux commands over and over, why not do it all with just one command?
– To automate difficult tasks: Many commands have subtle and difficult options that you don't want to figure out or remember every time.
– Creating your own power tools/utilities.
– Automating command input or entry.
– Customizing administrative tasks.
– Creating simple applications.
– Since scripts are well tested, the chances of errors are reduced while configuring services or system administration tasks such as adding new users.

Advantages

- Easy to use.
- Quick start, and interactive debugging.
- Time Saving.
- Sys Admin task automation.
- Shell scripts can execute without any additional effort on nearly any modern UNIX / Linux / BSD / Mac OS X operating system as they are written an interpreted language.

Disadvantages

- Compatibility problems between different platforms.
- Slow execution speed.
- A new process launched for almost every shell command executed.

8.4 VARIABLE NAMES

Variables names may comprise upper and lower case alphabetic characters, digits and underscores. A user defined variable name cannot start with a digit. As with most things in Unix variable names are case sensitive, e.g. FOO and foo are two distinct variables.

In Linux, there are two types of variable
1) System variables - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.
2) User defined variables (UDV) - Created and maintained by user. This type of variable defined normally by using lower case LETTERS.

Some System variables

You can see system variables by giving command like $ set, Some of the important System variables are

System Variable	**Meaning**
BASH=/bin/bash	Our shell name
BASH_VERSION=1.14.7(1)	Our shell version name
COLUMNS=80	No. of columns for our screen
HOME=/home/vivek	Our home directory
LINES=25	Number of columns for our screen
LOGNAME=students	Our logging name
OSTYPE=Linux	Our o/s type : -)
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Our path settings
PS1=[\u@\h \W]\$	Our prompt settings
PWD=/home/students/Common	Our current working directory
SHELL=/bin/bash	Our shell name
USERNAME=vivek	User name who is currently login to this PC

Special Shell Variables

Some variables have special significance to the shell. For example, the PATH variable is used by the shell to contain the list of directories to search for commands. Other variables have special meaning to other Linux utilities, for example the TERM variable contains the current terminal type.

```
bash$ variable1=23
```

Use echo command to display variable value.

```
bash$ echo variable1
        variable1
bash$ echo $variable1
        23
```

To display the program search path, type:

```
bash$ echo "$PATH"
```

To display your prompt setting, type:

```
bash$ echo "$PS1"
```

All variable names must be prefixed with $ symbol, and the entire construct should be enclosed in quotes. Try the following example to display the value of a variable without using $ prefix:

```
bash$ echo "HOME"
```

To display the value of a variable with echo $HOME:

```
bash$ echo "$HOME"
```

You must use $ followed by variable name to print a variable's contents.

The variable name may also be enclosed in braces:

```
bash$ echo "${HOME}"
```

This is useful when the variable name is followed by a character that could be part of a variable name:

```
bash$ echo "${HOME} work"
```

To define User Defined Variable (UDV) use following syntax

Syntax: *variablename=value*

NOTE: Here 'value' is assigned to given 'variablename' and Value must be on right side = sign For e.g.

```
$ no=10 # this is ok
$ 10=no # Error, NOT Ok, Value must be on right side of = sign.
```

To define variable called 'vech' having value Bus

```
$ vech=Bus
```

To define variable called n having value 10

```
$ n=10
```

Q.1.How to Define variable x with value 10 and print it on screen

```
$ x=10
$ echo $x
```

Linux Shell Script Tutorial

Q.2.How to Define variable xn with value Rani and print it on screen

```
$ xn=Rani
$ echo $xn
```

Q.3.How to print sum of two numbers, let's say 6 and 3

```
$ echo 6 + 3
```

This will print 6 + 3, not the sum 9, To do sum or math operations in shell use expr, syntax is as follows Syntax: *expr op1 operator op2*

Where, op1 and op2 are any Integer Number (Number without decimal point) and operator can be

+ Addition
- Subtraction
/ Division
% Modular, to find remainder For e.g. 20 / 3 = 6 , to find remainder 20 % 3 = 2, (Remember its integer calculation)
* Multiplication

```
$ expr 6 + 3
```

Now It will print sum as 9 , But

```
$ expr 6+3
```

will not work because space is required between number and operator (See Shell Arithmetic)

Q.4.How to define two variable x=20, y=5 and then to print division of x and y (i.e. x/y)

```
$x=20
$ y=5
$ expr x / y
```

Q.5.Modify above and store division of x and y to variable called z

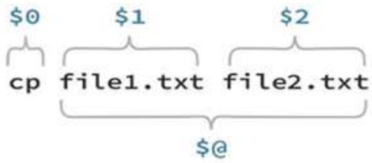
```
$ x=20
$ y=5
$ z=`expr x / y`
$ echo $z
```

Note : For third statement, read Shell Arithmetic.

8.5 POSITIONAL PARAMETER / COMMAND LINE ARGUMENTS

Command line arguments are important part of writing scripts. Command line arguments define the expected input into a shell script. For example, we may want to pass a file name or folder name or some other type of argument to a shell script.

A **positional parameter** is a variable within a **shell** program; its value is set from an **argument** specified on the **command** line that invokes the program. **Positional parameters** are numbered and are referred to with a preceding ``$": $1, $2, $3, and so on. A **shell** program may reference up to nine **positional parameters**.



Consider the following bash command. Let the command name is mycommand. The command line has three parameters: one, two, and three four.

$ mycommand one two “three four”

Variable name	**Value**
$0	Mycommand
$1	One
$2	Two
$3	three four
$#	3
$@	one two three four
$*	one two three four
$!	This parameter represents the process number of the background that was executed last.
$?	This parameter represents exit status of the last command that was executed. Here 0 represents success and 1 represents failure.
$_	This parameter represents the command which is being executed previously.
$-	This parameter will print the current options flags where the set command can be used to modify the options flags.

```
$ cat  program.sh
echo "The File Name is: $0"
echo "The First argument is: $1"
echo "The Second argument is: $2"

$ sh program.sh ab cd
The File Name: program.sh
The First argument is: ab
The Second argument is: cd
```

If any arguments are supplied, they become the positional parameters when filename is executed. Otherwise, the positional parameters remain unchanged.

if-then-else

Like most programming languages, shell script supports the if statement, with or without an else. The general form is below:

```
if (condition)
        simple commands

if (condition)
then
        commands
        …
fi

if (condition)
then
        commands
…
Else
        Commands
…
fi
```

You can have nested if-else-fi

The *condition* used as the predicate can be any program or expression. The results are evaluated with a 0 return being true and a non-0 return being false.

If ever there is the need for an empty if-block, the null command, a :, can be used in place of a command to keep the syntax legal.

The following is a nice, quick example of an if-else:

```
if [ "$LOGNAME"="guna" ]
then
        printf "%s is logged in" $LOGNAME
else
        printf "Intruder! Intruder!"
fi
```

The elif construct

Shell scripting also has another construct that is very helpful in reducing deep nesting. It is unfamilar to those of us who come from languages like C and Perl. It is the elif, the "else if". This probably made its way into shell scripting because it drastically reduces the nesting that would otherwise result from the many special cases that real-world situatins present -- without functions to hide complexity (shell does have functions, but not parameters -- and they are more frequently used by csh shell scripters than traniditonalists).

```
if condition
then
        command
        command
        ...
        command
elif condition
then
        command
        command
        ...
        command
elif condition
then
        command
        command
        ...
        command
fi
```

The switch statement

Much like C, C++, or Java, shell has a case/switch case statement. The form is as follows:

```
case “$variable” in

pattern1)
command
```

```
command
...
command
;;   # Two ;;'s serve as the break
Pattern2)
command
command
...
command
;; # Two ;;'s serve as the break
Pattern3)
command
command
...
command
;;  # Two ;;'s serve as the break
esac
```

Here's a quick example:

```
#!/bin/sh
case "$char"
in
"+")
        ans=`expr $1 + $3`
        printf "%d %s %d = %d\n" $1 $2 $3 $ans
        ;;
"-")
        ans=`expr $1 - $3`
        printf "%d %s %d = %d\n" $1 $2 $3 $ans
        ;;
"\*")
        ans=`expr "$1 * $3"`
        printf "%d %s %d = %d\n" $1 $2 $3 $ans
        ;;
"/")
        ans=`expr $1 / $3`
        printf "%d %s %d = %d\n" $1 $2 $3 $ans
        ;;
# Notice this: the default case is a simple *
*)
        printf "Don't know how to do that.\n"
        ;;
esac
```

The for Loop

The for loop provides a tool for processing a list of input. The input to for loop is a list of values. Every iteration through the loop extracts one value into a variable and then enters the body of the loop. The loop stops when the extract fails because there are no more values in the list.

for loop operates on a list and repeats commands in the block for each element on the list.

```
for x in [ list ]
do
        commands
done
```

Let's consider the following example which prints each of the command line arguments, one at a time. We'll extract them from "$@" into $arg:

```
for var in "$@"
do
        printf "%s\n" $var
done
```

Much like C or Java, shell has a *break* command, also. As you might guess, it can be used to get out of a loop. Consider this example which stops printing command line arguments, when it gets to one whose value is "quit":

```
for var in "$@"
do
        if [ "$var" = "quit" ]
then
        break
fi
        printf "%s\n" $var
done
```

Similarly, shell has a *continue* that works just like it does in C or Java.

```
for var in "$@"
do
        if [ "$var" = "me" ]
then
        continue
elif [ "$var" = "mine" ]
then
        continue
elif [ "$var" = "myself" ]
```

```
then
        continue
fi
if [ "$var" = "quit" ]
then
        break
fi
printf "%s\n" $var
done
```

The while and until Loops

Shell has a while loop similar to that seen in C or Java. It continues until the predicate is false. And, like the other loops within shell, break and continue can be used. Here's an example of a simple while loop:

```
# This lists the files in a directory in alphabetical order

# It continues until the read fails because it has reached the end of input

ls | sort |

        while read file
        do
                echo $file
        done
```

In the above code, | called a "pipe" directs output from one process to another process. For

example, ls | sort takes the output from ls command, and sort the output stream received. Pipe is a form of inter process communication which we will discuss later.

There is a similar loop, the *until* loop that continues *until* the condition is successful -- in other words, while the command failes. This will pound the user for input until it gets it:

```
printf "ANSWER ME! "
until read $answer
do
        printf "ANSWER ME! "
done
```

8.6 PRACTICE QUESTIONS

Program1: Write a shell script program to display "HELLO WORLD".

Program2: Write a shell script to find the factorial of given integer

Program3: Write a shell Script program to check whether the given number is even or odd.

The Motto of Our University
(SEWA)
SKILL ENHANCEMENT
EMPLOYABILITY
WISDOM
ACCESSIBILITY

JAGAT GURU NANAK DEV
PUNJAB STATE OPEN UNIVERSITY, PATIALA

(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

B.Sc.(Data Science)

Semester II

BSDB31202T

Statistical Foundation

SELF-INSTRUCTIONAL STUDY MATERIAL FOR JGND PSOU

Head Quarter: C/28, The Lower Mall, Patiala-147001
Website: www.psou.ac.in

The Study Material has been prepared exclusively under the guidance of Jagat Guru Nanak Dev Punjab State Open University, Patiala, as per the syllabi prepared by Committee of experts and approved by the Academic Council.

COURSE COORDINATOR AND EDITOR:

Dr. Amitoj Singh
Associate Professor
School of Sciences and Emerging Technologies
Jagat Guru Nanak Dev Punjab State Open University

LIST OF CONSULTANTS/ CONTRIBUTORS

Sr. No.	Name
1	Dr. Pardeek Kumar
2	Dr. Chetan Dudhagara
3	Dr. Amit Kamra
4	Dr. Raju D. Chaudhary

JAGAT GURU NANAK DEV PUNJAB STATE OPEN UNIVERSITY, PATIALA
(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

PREFACE

Jagat Guru Nanak Dev Punjab State Open University, Patiala was established in December 2019 by Act 19 of the Legislature of State of Punjab. It is the first and only Open University of the State, entrusted with the responsibility of making higher education accessible to all, especially to those sections of society who do not have the means, time or opportunity to pursue regular education.

In keeping with the nature of an Open University, this University provides a flexible education system to suit every need. The time given to complete a programme is double the duration of a regular mode programme. Well-designed study material has been prepared in consultation with experts in their respective fields.

The University offers programmes which have been designed to provide relevant, skill-based and employability-enhancing education. The study material provided in this booklet is self-instructional, with self-assessment exercises, and recommendations for further readings. The syllabus has been divided in sections, and provided as units for simplification.

The University has a network of 10 Learner Support Centres/Study Centres, to enable students to make use of reading facilities, and for curriculum-based counselling and practicals. We, at the University, welcome you to be a part of this instituition of knowledge.

Prof. Anita Gill
Dean Academic Affairs

B.Sc. (Data Science)
Discipline Specific Elective (DSE)
Semester II
BSDB31202T: Statistical Foundation

Total Marks: 100
External Marks: 70
Internal Marks: 30
Credits: 4
Pass Percentage: 35%

Objective: This course will enable students to understand the fundamentals of statistics to apply descriptive measures and probability for data analysis. Students will able to infer the concept of correlation and regression for relating two or more related variables and probabilities for various events.

INSTRUCTIONS FOR THE PAPER SETTER/EXAMINER

1. The syllabus prescribed should be strictly adhered to.
2. The question paper will consist of three sections: A, B, and C. Sections A and B will have four questions from the respective sections of the syllabus and will carry 10 marks each. The candidates will attempt two questions from each section.
3. Section C will have fifteen short answer questions covering the entire syllabus. Each question will carry 3 marks. Candidates will attempt any ten questions from this section.
4. The examiner shall give a clear instruction to the candidates to attempt questions only at one place and only once. Second or subsequent attempts, unless the earlier ones have been crossed out, shall not be evaluated.
5. The duration of each paper will be three hours.

INSTRUCTIONS FOR THE CANDIDATES

Candidates are required to attempt any two questions each from the sections A and B of the question paper and any ten short questions from Section C. They have to attempt questions only at one place and only once. Second or subsequent attempts, unless the earlier ones have been crossed out, shall not be evaluated.

Section A

Unit I: Origin and Development of Statistics: Scope, limitation and misuse of statistics. Types of data: primary, secondary, quantitative and qualitative data. Types of Measurements: nominal, ordinal, discrete and continuous data.

Unit II: Presentation of Data by Tables: construction of frequency distributions for discrete and continuous data, graphical representation of a frequency distribution by histogram and frequency polygon, cumulative frequency distributions. Classification and

Graphical representation of data (Pie Chart, Bar Diagram, Histogram, Frequency Polygon, Ogive Curve, etc.).

Unit III: Measures of Central Tendency – Arithmetic Mean, Median and Mode and its Graphical representation, Measures of dispersion – range, variance, mean deviation, standard deviation and coeff. of variation, Concepts and Measures of Skewness and Kurto.

Unit IV: Descriptive and Exploratory Analysis: Descriptive Statistics, Exploratory data analysis, Coefficient of variation, Data visualization, Scatter diagram, Grouped data,

Section B

Unit V: Correlation: Scatter plot, Karl Pearson coefficient of correlation, Spearman's rank correlation coefficient, multiple and partial correlations (for 3 variates only). Regression: Introduction to regression analysis: Modelling a response, overview and applications of regression analysis, Simple linear regression (Two variables)

Unit VI: **Mathematical and Statistical probability:** Introduction Elementary events, Sample space, Compound events, Types of events, Random experiment, sample point and sample space, event, algebra of events.

Unit VII: Definition of Probability: classical, empirical and axiomatic approaches to probability, properties of probability. Theorems on probability, conditional probability and independent events

Unit VIII: Statistical Inference: Introduction, Concept of Random Variable, Probability Mass Function & Density Function, Mathematical Expectation (meaning and properties), Moments, Moment Generating Function and Characteristic Function

Suggested Readings

1. José Unpingco Python for Probability, Statistics, and Machine Learning, Springer, 2019
2. Gupta, S.C. and Kapoor, V.K.: Fundamentals of Mathematical Statistics, Sultan & Chand & Sons, New Delhi, 11th Ed, 2014
3. Allen B. Downey, Think Stats: Exploratory Data Analysis, O'Reilly Media, 2014
4. Statistics for Beginners in Data Science: Theory and Applications of Essential Statistics Concepts using Python, Ai Publishing, 2020

JAGAT GURU NANAK DEV PUNJAB STATE OPEN UNIVERSITY, PATIALA

(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

BSDB31202T: STATISTICAL FOUNDATION

COURSE COORDINATOR AND EDITOR: DR. AMITOJ SINGH

UNIT NO.	UNIT NAME
UNIT 1	ORIGIN AND DEVELOPMENT OF STATISTICS
UNIT 2	PRESENTATION OF DATA BY TABLES
UNIT 3	MEASURES OF CENTRAL TENDENCY
UNIT 4	DESCRIPTIVE AND EXPLORATORY ANALYSIS
UNIT 5	CORRELATION
UNIT 6	MATHEMATICAL AND STATISTICAL PROBABILITY
UNIT 7	DEFINITION OF PROBABILITY
UNIT 8	STATISTICAL INFERENCE

B.Sc.(DATA SCIENCE)

SEMESTER-II

STATISTICAL FOUNDATION

UNIT I: ORIGIN AND DEVELOPMENT OF STATISTICS

STRUCTURE

1.0 OBJECTIVES

In this module, we will try to understand about the historical development of statistics. Various definitions given by different scientists will also be explored in this module. Further need of statistics along with its applications will also be investigated. Limitations and misuse of statistics will also be looked upon. In the end, various types of data used in statistics will also be explored.

1.1 INTRODUCTION

This module is designed to know about the development of statistics using historical background. Statistics is not a subject that can be studied alone, rather it proves to be basis for almost all other subjects as data handling is essential in almost all fields of life. Due to this reason, a number of definitions are given to statistics. Some of the major fields where statistics is prominently used are planning, finance, business, agriculture, biology, economics, industry, education, etc. Actually a country's growth is very much dependent on statistics as without statistics it would not be possible to estimate the requirements of the country. However, statistics is based on probabilistic estimations and therefore not actual (in some cases), therefore can't be believed with 100% guarantee. Also some people may misuse statistics for their own benefits. But, still statistics is very essential and very much need of the life. There are various classifications of the data used in statistics viz., continuous, discrete, nominal, ordinal, etc. The data can be used as per requirement for a particular application.

1.2 MAIN CONTENT

1.2.1 Origin and Development of Statistics

The term Statistics, is not a new term rather it is as old as being human society. However, it has come up leaps and bounds with the emergence of data analytics and machine learning and the role of statistics in those areas. Directly and indirectly, statistics has been used right from the ancient days, although as a subject it was introduced much later.

In the earlier days it was regarded more or less as a by-product of the administrative activity for analysing the various activities. The word Statistics is said to be derived from a Latin word 'status' or an Italian word 'statista' or may be the German word 'statistik' or the even from the French word 'statistique', each of which actually means a political state. The term was used to collect information about the population in a country so that various schemes could be implemented depending on the requirement and size of population.

If we talk specifically about India, various efficient mechanisms for collecting various type of official statistics existed even 2000 years ago, during the tenure of Chandragupta Maurya. In history, evidences have been found for the existence of an excellent system of collecting important statistics and registration of births and deaths, even before 300 B.C. are available in Kautilya's 'Arthashastra'. Further, the records of land, agriculture and wealth statistics were maintained by Todermal, a well-known land and revenue minister in the era of Akbar (1556-1605 A.D). A detailed information about the administrative and statistical surveys conducted during Akbar's era is very much available in the book "Ain-e- Akbari" written by Abul Fazl (1596-97), one of the nine gems of Akbar.

The Statistics was applied for collecting the data related to the movements of heavenly bodies in the 16th century, viz., stars and planets so that their position may be known and various Eclipses may be predicted. Further, in the seventeenth century, the vital statistics was originated. Captain John Graunt of London (1620-1674) is known as the Father of vital statistics. He was the person behind a systematic study of the birth and death statistics.

Modern stalwarts involved in the development of the subject of Statistics, are various Englishmen, who did revolutionary work in applying the Statistics to different disciplines. Francis Galton (1822-1921) pioneered the study of 'Regression Analysis' in Biometry; Karl Pearson (1857-1936), founder of the greatest statistical laboratory in England pioneered the study of 'Correlation Analysis'. His Chi-Square test (X^2-test) of Goodness of Fit is one of the most important tests of significance in Statistics; W.S. Gosset introduced t-test, which escorted the era of exact (small) sample tests.

However most of the work in the statistical theory during the past few decades can be attributed to a single person Sir Ronald A. Fisher (1890-1962), who applied statistics to a variety of diversified fields such as genetics, biometry, psychology and education, agriculture, etc., and who is rightly termed as the Father of Statistics. He not only enhanced the existing statistical theory, but also he is the pioneer in Estimation Theory; Exact (small) Sampling Distributions; Analysis of Variance and Design of Experiments. One can easily say that R.A. Fisher is the real giant in the development of the theory of Statistics. It is due to the outstanding contributions of R. A. Fisher that put the subject of Statistics on a very firm footing and earned for it the status of a full-fledged science.

1.2.2 Definitions of Statistics

Statistics has been defined by number of authors in different ways. The main reason for the various definitions are the changes that has taken place in statistics from time to time. Statistics in general is defined in two different ways viz., as 'statistical data', i.e., based on numerical statement of data and facts, and as 'statistical methods', i.e., based on the principles and techniques used in collecting and analysing such data. Some of the important definitions under these two categories are given below.

Statistics as Statistical data

Webster defines Statistics as "classified facts representing the conditions of the people in a State, especially those facts which can be stated in numbers or in any other tabular or classified arrangement." Bowley defines Statistics as "numerical statements of facts in any department of enquiry placed in relation to each other."

A more exhaustive definition is given by Prof. Horace Secrist as follows: "By statistics we mean aggregation of facts affected to a marked extent by multiplicity of causes numerically expressed, enumerated or estimated according to reasonable standards of accuracy, collected in a systematic manner for a predetermined purpose and placed in relation to each other."

Statistics as Statistical Methods

Bowley himself has defined Statistics in a number of ways:

(i) Statistics may be called the science of counting.

(ii) Statistics may rightly be called the science of averages.

(iii) Statistics is the science of the measurement of social organism, regarded as a whole in all its manifestations.

However, these definitions are not complete in any sense as they don't provide the complete view of statistics. According to Boddington, "Statistics is the science of estimates and probabilities." Again this definition is not complete as statistics is not just probabilities and estimates but more than that.

Some other definitions are: "The science of Statistics is the method of judging collective, natural or social phenomenon from the results obtained from the analysis or enumeration or collection of estimates."- as provided by King.

"Statistics is the science which deals with collection, classification and tabulation of numerical facts as the basis for explanation, description and comparison of phenomenon." as given by Lovitt.

But the best definition is the one given by Croxton and Cowden, according to whom Statistics may be defined as "the science which deals with the collection, analysis and interpretation of numerical data."

1.2.3 Importance and Scope of Statistics

Statistics is primarily used either to make predictions based on the data available or to make conclusions about a population of interest when only sample data is available. In both cases statistics tries to make sense of the uncertainty in the available data. When making predictions statisticians determine if the difference in the data points are due to chance or if there is a systematic relationship. The more the systematic relationship that is observed the better the prediction a statistician can make. The more random error that is observed the more uncertain the prediction.

Statisticians can provide a measure of the uncertainty to the prediction. When making inference about a population, the statistician is trying to estimate how good a summary statistic of a sample really is at estimating a population statistic.

For computer students, knowing the basic principles and methods in statistics could help them in doing their research work like comparing the speed of internet connection in different countries and the probability of how many times does each experience the same level of internet connection speed in a week, month or year. It could also be helpful in determining the best operating system to use. Whenever there is the need to compare data and know the best option that we should take statistics can give the answer.

Statistics is having applications in almost all sciences - social as well as physical such as biology, psychology, education, economics, business management, etc. It is hardly possible to think of even a single department of human activity where statistics is not involved. It has rather become indispensable in all phases of human endeavour.

Statistics and Planning

Statistics is mother of planning. In the modern age which is termed as 'the age of planning', almost all over the world, particularly of the upcoming economies, are resorting to planning for the economic development. In order that planning is successful, it must be based soundly on the correct analysis of complex statistical data.

Statistics and Economics

Statistical data and technique of statistical analysis have proved immensely useful in solving various economic problems, such as wages, prices, analysis of time series and demand analysis. A number of applications of statistics in the study of economics have led to the development of new disciplines called Economic Statistics and Econometrics.

Statistics and Business

Statistics is an essential tool for production control. Statistics not only helps the business executives to know the requirements of the consumers, but also for many other purposes. The success of a business actually depends upon the accuracy and precision of his statistical forecasting. Wrong analysis, due to faulty and inaccurate analysis of various causes affecting a particular phenomenon, might prove to be a disaster. Consider an examples of manufacturing readymade garments. Before starting one must have an overall idea as to ‘how many garments are to be manufactured', 'how much raw material and labour is needed for that', and 'what is the quality, shape, color, size, etc., of the garments to be manufactured'. If these questions are not analysed statistically in a proper manner, the business is bound to be failed. Therefore, most of the large industrial and commercial enterprises are employing trained and efficient statisticians.

Statistics and Industry

In industry, statistics is very widely used in 'Quality Control'. In production engineering, to find whether the product is conforming to specifications or not, statistical tools, viz. inspection plans, control charts, etc., are of extreme importance.

Statistics and Mathematics

Statistics and mathematics arc very intimately related. Recent advancements in statistical techniques are the outcome of wide applications of advanced mathematics. Main contributors to statistics, namely, Bernouli, Pascal, Laplace, De-Moirve, Gauss, R. A. Fisher, to mention only a few, were primarily talented and skilled mathematicians. Statistics may be regarded as that branch of mathematics which provided us with systematic methods of analysing a large number of related numerical facts. According to Connor, " Statistics is a branch of Applied Mathematics which specialises in data."

Statistics and Biology, Astronomy and Medical Science

The association between statistical methods and biological theories was first studied by Francis Galton in his work in Regression. According to Prof. Karl Pearson, the whole 'theory of heredity' rests on statistical basis. He said, "The whole problem of evolution is a problem of vital statistics, a problem of longevity, of fertility, of health, of disease and it is impossible to discuss the national mortality without an enumeration of the population, a classification of deaths and knowledge of statistical theory." In astronomy, the theory of Gaussian 'Normal Law of Errors' for the study of the movement of stars and planets is developed by using the 'Principle of Least Squares'. In medical science also, the statistical tools for the collection, presentation and analysis of observed facts relating to the causes and diseases and the results obtained from the use of various drugs and medicines, are of great importance. Moreover, the efficacy of a manufactured drug or injection or medicine is tested by analysing the 'tests of significance'.

Statistics and Psychology and Education

In education and psychology, too, statistics has found wide applications, e.g., to determine the reliability and validity of a test, 'Factor Analysis', etc., so much so that a new subject called 'Psychometry' has come into existence.

Statistics and War

In war, the theory of 'Decision Functions' can be of great assistance to military and technical personnel to plan 'maximum destruction with minimum effort'. Thus, we see that the science of Statistics is associated with almost all the sciences - social as well as physical. Bowley has rightly said, "A knowledge of Statistics is like a knowledge of foreign language or algebra; it may prove of use at any time under any circumstance."

1.2.4 Limitations of Statistics

Statistics, with its wide applications in almost every sphere of human activity; is not without limitations. The following are some of its important limitations:

(i) **Statistics is not suited to the study of qualitative phenomenon.** Statistics, being a science dealing with a set of numerical data, is applicable to the study of only those subjects of enquiry which are capable of quantitative measurement. As such; qualitative phenomena like honesty, poverty, culture, etc., which cannot be expressed numerically, are not capable of direct statistical analysis. However, statistical techniques may be applied indirectly by first reducing the qualitative expressions to precise quantitative terms. For example, the intelligence of a group of candidates can be studied on the basis of their scores in a certain test.

(ii) **Statistics does not study individuals.** Statistics deals with an aggregate of objects and does not give any specific recognition to the individual items of a series. Individual items, taken separately, do not constitute statistical data and are meaningless for any statistical enquiry. For example, the individual figures of agricultural production, industrial output or national income of any country for a particular year are meaningless unless, to facilitate comparison, similar figures of other countries or of the same country for different years are given. Hence, statistical analysis is suited to only those problems where group characteristics are to be studied.

(iii) **Statistical laws are not exact.** Unlike the laws of physical and natural sciences, statistica1laws are only approximations and not exact. On the basis of statistical analysis, we can talk only in terms of probability and chance and not in terms of certainty. Statistical conclusions are not universally true, rather they are true only on an average.

1.2.5 Misuse of Statistics

Statistics is liable to be misused. As they say, "Statistical methods are the most dangerous tools in the hands of the in experts. Statistics is one of those sciences whose adepts must exercise the self-restraint of an artist." The use of statistical tools by inexperienced and untrained persons might lead to very fallacious conclusions. One of the greatest shortcomings of statistics is that by just looking at them one can’t comment about their quality and as such can be represented in any manner to support one's way of argument and reasoning. As King said, "Statistics are like clay of which one can make a god or devil as one place." The requirement of experience and judicious use of statistical methods restricts their use to experts only and limits the chances of the mass popularity of this useful and important science.

It may be pointed out that Statistics neither proves anything nor disproves anything. It is only a tool which if rightly used may prove extremely useful and if misused might be disastrous. According to Bowley, "Statistics only furnishes a tool necessary though imperfect, which is dangerous in the hands of those who do not know its use and its deficiencies." It is not the statistics which can be blamed but those persons who twist the numerical data and misuse them either due to ignorance or deliberately for personal selfish motives. As King pointed out, "Science of Statistics is the most useful servant but only of great value to those who understand its proper use."

A few interesting examples showing the impact of misrepresentation of statistical data are:

(i) A statistical report, "The number of accidents taking place in the middle of the road is much less than the number of accidents taking place on its side. Hence it is safer to walk in the middle of the road." This conclusion is obviously wrong since we are not given the proportion of the number of accidents to the number of persons walking in the two cases.

(ii) Another saying that, "The number of students taking up Computer Science in a University has increased 5 times during the last 3 years. Thus, Computer Science is gaining popularity among the students of the university." Again, the conclusion is faulty since we are not given any such details about the other subjects and hence comparative study is not possible.

(iii) One more interesting examples says that, "99% of the people who drink alcohol die before attaining the age of 100 years. Hence drinking is harmful for longevity of life." This statement, too, is incorrect since nothing is mentioned about the number of persons who do not alcohol and die before attaining the age of 100 years. Thus, statistical arguments based on incomplete data often lead to fallacious conclusions.

1.2.6 Types of Data

In statistics, the data are the individual pieces of factual information recorded, and it is used for the purpose of the analysis process. The two processes of data analysis are interpretation and presentation. Statistics are the result of data analysis. Data classification and data handling are an important process as it involves a multitude of tags and labels to define the data, its integrity and confidentiality. The data can be classified as shown in figure 1.1 and has been described as follows:

Qualitative or Categorical Data

Qualitative data, also known as the categorical data, describes the data that fits into the categories. Qualitative data are not numerical. The categorical information involves categorical variables that describe the features such as a person's gender, home town etc. Categorical measures are defined in terms of natural language specifications, but not in terms of numbers.

Sometimes categorical data can hold numerical values (quantitative value), but those values do not have mathematical sense. Examples of the categorical data are birthdate, favourite sport, school postcode. Here, the birthdate and school postcode hold the quantitative value, but it does not give numerical meaning. It can be further classified as nominal and ordinal data.

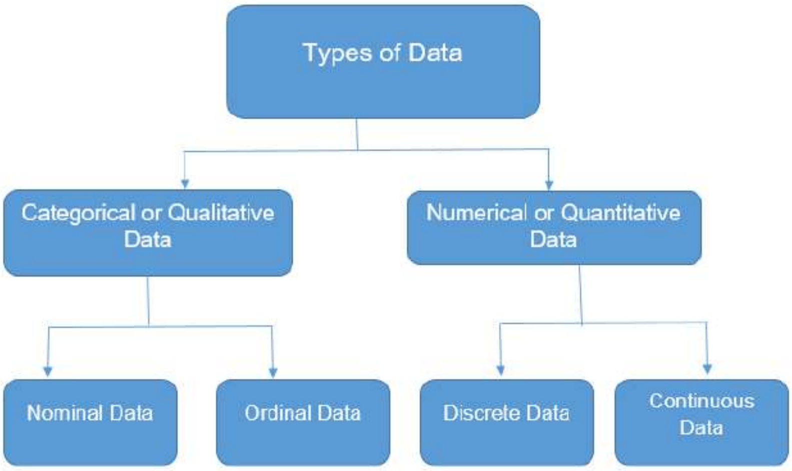


Figure 1.1: Classification of Data used in Statistics

Nominal Data: Nominal data is one of the types of qualitative information which helps to label the variables without providing the numerical value. Nominal data is also called the nominal scale. It cannot be ordered and measured. But sometimes, the data can be qualitative and quantitative. Examples of nominal data are letters, symbols, words, gender etc.

The nominal data are examined using the grouping method. In this method, the data are grouped into categories, and then the frequency or the percentage of the data can be calculated. These data are visually represented using the pie charts.

Ordinal Data: Ordinal data is a type of data which follows a natural order. The significant feature of the nominal data is that the difference between the data values is not determined. This variable is mostly found in surveys, finance, economics, questionnaires, and so on.

The ordinal data is commonly represented using a bar chart. These data are investigated and interpreted through many visualisation tools. The information may be expressed using tables in which each row in the table shows the distinct category.

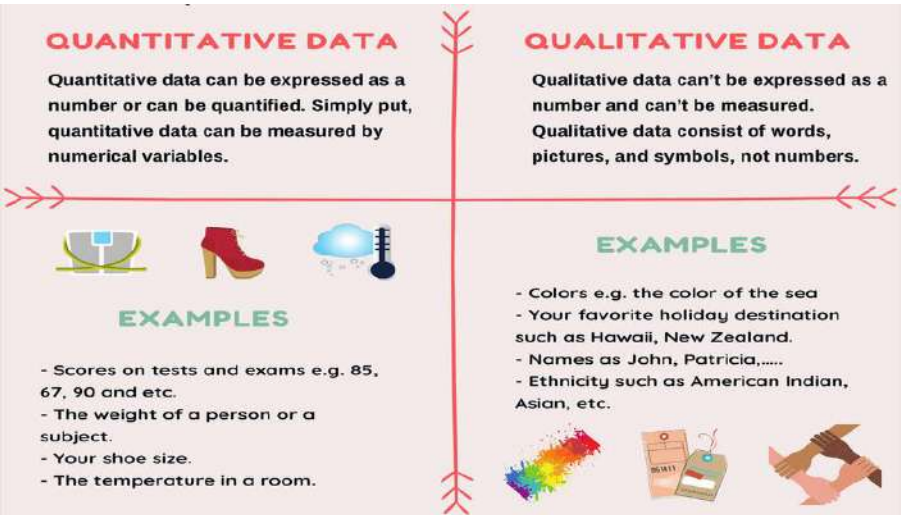
Quantitative or Numerical Data

Quantitative data is also known as numerical data which represents the numerical value (i.e., how much, how often, how many). Numerical data gives information about the quantities of a specific thing. Some examples of numerical data are height, length, size, weight, and so on. The quantitative data can be classified into two different types based on the data sets. The two different classifications of numerical data are discrete data and continuous data.

Discrete Data: Discrete data can take only discrete values. Discrete information contains only a finite number of possible values. Those values cannot be subdivided meaningfully. Here, things can be counted in the whole numbers e.g. Number of students in the class

Continuous Data: Continuous data is data that can be calculated. It has an infinite number of probable values that can be selected within a given specific range e.g. Temperature range.

The quantitative and qualitative data can be represented as in figure 1.2.



QUANTITATIVE DATA

Quantitative data can be expressed as a number or can be quantified. Simply put, quantitative data can be measured by numerical variables.

EXAMPLES

- Scores on tests and exams e.g. 85, 67, 90 and etc.
- The weight of a person or a subject.
- Your shoe size.
- The temperature in a room.

QUALITATIVE DATA

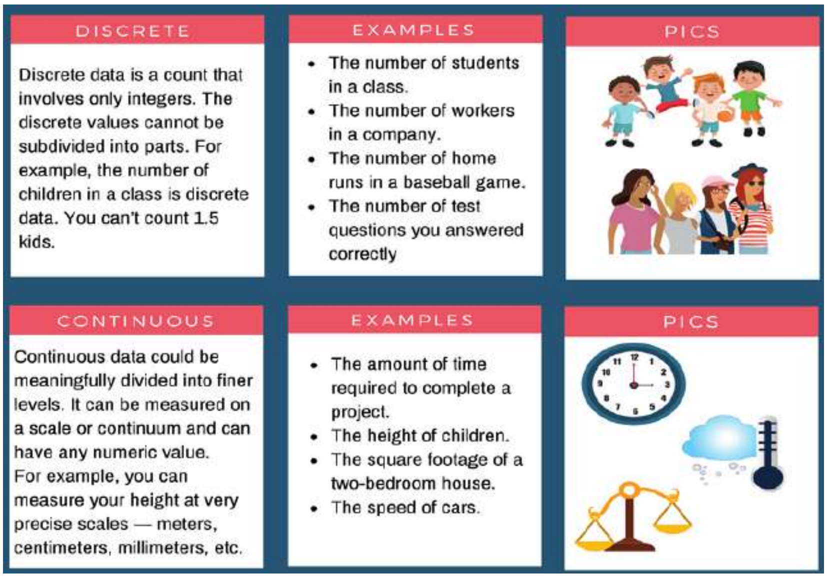
Qualitative data can't be expressed as a number and can't be measured. Qualitative data consist of words, pictures, and symbols, not numbers.

EXAMPLES

- Colors e.g. the color of the sea
- Your favorite holiday destination such as Hawaii, New Zealand.
- Names as John, Patricia,.....
- Ethnicity such as American Indian, Asian, etc.

Figure 1.2: Quantitative and Qualitative Data

Figure 1.3 shows the types of qualitative data i.e. discrete and continuous data.



DISCRETE

Discrete data is a count that involves only integers. The discrete values cannot be subdivided into parts. For example, the number of children in a class is discrete data. You can't count 1.5 kids.

EXAMPLES

- The number of students in a class.
- The number of workers in a company.
- The number of home runs in a baseball game.
- The number of test questions you answered correctly

PICS

CONTINUOUS

Continuous data could be meaningfully divided into finer levels. It can be measured on a scale or continuum and can have any numeric value. For example, you can measure your height at very precise scales — meters, centimeters, millimeters, etc.

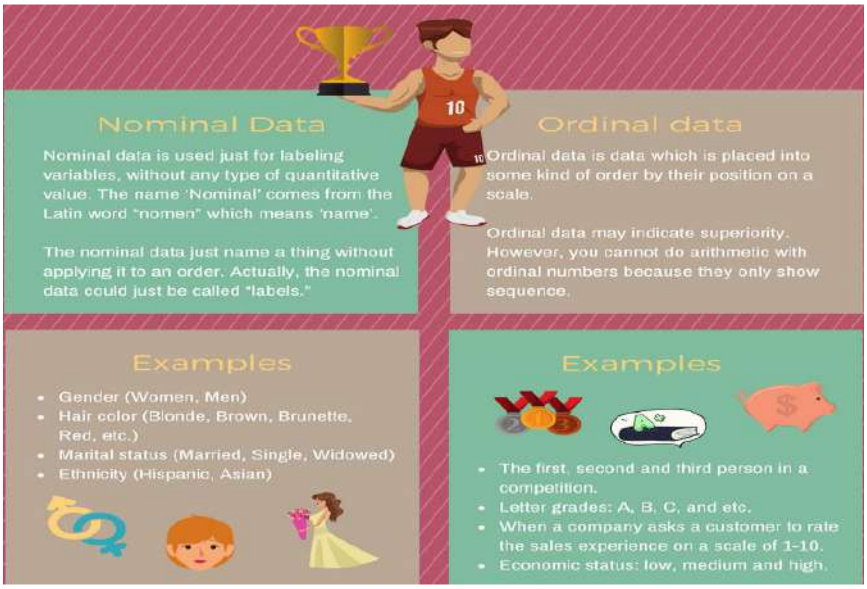
EXAMPLES

- The amount of time required to complete a project.
- The height of children.
- The square footage of a two-bedroom house.
- The speed of cars.

PICS

Figure 1.3: Types of Qualitative Data viz., Discrete and Continuous

Figure 1.4 shows the types of quantitative data i.e. nominal and ordinal data.



Nominal Data

Nominal data is used just for labeling variables, without any type of quantitative value. The name 'Nominal' comes from the Latin word "nomen" which means 'name'.

The nominal data just name a thing without applying it to an order. Actually, the nominal data could just be called "labels."

Ordinal data

Ordinal data is data which is placed into some kind of order by their position on a scale.

Ordinal data may indicate superiority. However, you cannot do arithmetic with ordinal numbers because they only show sequence.

Examples

- Gender (Women, Men)
- Hair color (Blonde, Brown, Brunette, Red, etc.)
- Marital status (Married, Single, Widowed)
- Ethnicity (Hispanic, Asian)

Examples

- The first, second and third person in a competition.
- Letter grades: A, B, C, and etc.
- When a company asks a customer to rate the sales experience on a scale of 1-10.
- Economic status: low, medium and high.

Figure 1.4: Types of Quantitative Data viz., Nominal and Ordinal

1.2.7 Data Collection

Depending on the source, it can be classified as primary data or secondary data. Let us take a look at them both.

Primary Data

These are the data that are collected directly by an investigator for a specific purpose. Primary data are 'pure' in the sense that no statistical operations have been performed on them and they are original. An example of primary data is the Census of India.

Secondary Data

They are the data that are sourced from someplace that has originally collected it. This means that this kind of data has already been collected by some researchers or investigators in the past and is available either in published or unpublished form. This information is impure as statistical operations may have been performed on them already. An example is an information available on the Government of India, the Department of Finance's website or in other repositories, books, journals, etc.

1.3 SUMMARY

In this module, the overall development and history of statistics has been discussed. Various definitions given be various authors have been provided and discussed. Following this, the applications along with advantages and limitations of statistics have also been discussed in detail. One of the very important aspect, misuse of statistics has also been explored along with few examples for misuse of the statistics. Then the various classifications of statistical data have also been discussed in detail. In the end, the types of data collection methods have been discussed in brief. Overall, this module provides an overview of what is statistics along with its applications in depth.

1.4 QUESTIONS FOR PRACTICE

- Give a historical background of statistics.
- Write various definitions of statistics and discuss these definitions in brief.
- State and explain various applications of statistics.
- What are the various limitations of statistics?
- "Statistics don't lie". Comment on this statement.
- Provide a few examples which can lead to incorrect conclusion due to wrong analysis of statistics.
- Give any two examples of collecting data from day-to-day life.
- How can you classify the statistical data?
- Categorize the following data in various types: (i) Speed (ii) Gender (iii) Height (iv) Grades (v) No. of Employees (vi) Time (vii) Colour (viii) Score (ix) Weight.
- The word statistics seems to have been derived from which word?
- From ________ it is known that even before 300 B.C. a very good system of collecting "Vital Statistics" and registration of births and deaths was in vogue__________.
- Who is known as the father of "Vital Statistics"?

REFERENCES

- A. Abebe, J. Daniels, J.W.Mckean, "Statistics and Data Analysis".
- Clarke, G.M. & Cooke, D., "A Basic course in Statistics", Arnold.
- David M. Lane, "Introduction to Statistics".
- S.C.Gupta and V.K.Kapoor, "Fundamentals of Mathematical Statistics", Sultan Chand & Sons, New Delhi.

B.Sc.(DATA SCIENCE)

SEMESTER-II

STATISTICAL FOUNDATION

UNIT II: PRESENTATION OF DATA BY TABLES

STRUCTURE

2.0 OBJECTIVES

In this module, we will try to understand about the representation of data through tables. The various types of distribution of data and tabular presentation will be discussed in this module. Further, graphical representation of data through some popular methods such as histogram and frequency polygons will also be discussed in detail. Thereafter, the classification and graphical representation of statistical data through various methods will be explained. Further, we shall also learn the various types of representations (both tabular and graphical) of data in Python.

2.1 INTRODUCTION

This module is designed to know about the representation of data in tabular and graphical forms. For analysing the statistical data, it must be represented in a tabular form and this module does the same, i.e., describe the techniques to convert the data in tabular forms. For the purpose of planning and interpreting the data, visual effects are very useful and necessary. The visual effects in statistics can be obtained by representing the data through graphs. In this module, various types of graphs, viz., histogram, frequency polygons, bar graphs, pie charts, ogives, etc. have been discussed. Further, an automated tool for representing such type of data must be used for quick analysis and better representation. One such tool Python is used in this chapter to represent the data.

2.2 MAIN CONTENT

2.2.1 Presentation of Data by Tables

Whenever we want to analyse and interpret the data, it can be done in an effective manner only when it is represented in tabular and/or graphical manner. The data in tabular form can be represented by using frequency distributions as explained in the following section.

2.2.2 Construction of Frequency Distribution

Frequency Distributions

When observations, discrete or continuous, are available on a single characteristic of a large number of individuals, often it becomes necessary to condense the data as far as possible without losing any information of interest. For condensing the data, it is represented using either discrete or continuous frequency distribution tables.

Discrete frequency distribution: In discrete frequency distribution, values of the variable is arranged individually. The frequencies of the various values are the number of times each value occurs. For examples, the weekly wages paid to the workers are given below.

300, 240, 240, 150, 120, 240, 120, 120, 150, 150, 150, 240, 150, 150, 120, 300, 120, 150, 240, 150, 150, 120, 240, 150, 240, 150, 120, 120, 240, 150.

There are various ways to form a frequency distribution for this data. In the first case, let us assume that data is represented in terms of tally marks in a tabular manner as shown below in the table 2.1:

Table 2.1: Representation of Data using Tally Marks

Weekly Wages	Tally Marks	No. of Workers
120	~~IIII~~ III	8
150	~~IIII~~ ~~IIII~~ II	12
240	~~IIII~~ III	8
300	II	2

This data can also be represented without using tally marks i.e. using frequency only as shown in table 2.2 and is known as frequency table.

Table 2.2: Frequency Table for the Data in Table 2.1

Weekly Wages (x)	120	150	240	300	Total
No. of Workers (f)	8	12	8	2	30

The frequency table 2.1 is ungrouped frequency table. We can also draw a grouped frequency table depending on the data we are having. For designing a grouped frequency table, let us consider the following example regarding daily maximum temperatures in in a city for 50 days.

28, 28, 31, 29, 35, 33, 28, 31, 34, 29, 25, 27, 29, 33, 30, 31, 32, 26, 26, 21, 21, 20, 22, 24, 28, 30, 34, 33, 35, 29, 23, 21, 20, 19, 19, 18, 19, 17, 20, 19, 18, 18, 19, 27, 17, 18, 20, 21, 18, 19.

Table 2.3: Grouped Frequency Table

Temperature	Frequency
17-21	17
22-26	9
27-31	13
32-36	11
Total	50

The classes of type 17-21 and 22-26 are inclusive in nature i.e. both the lower bound and upper bound are included in the limit.

Although there are no hard and fast rules that have been laid down for it The following points may be kept in mind for classification:

(i) The classes should be clearly defined and should not lead to ambiguity.

(ii) The classes should be exhaustive, i.e., each of the given values should be included in one of the classes.

(iii) The classes should be mutually exclusive and non-overlapping.

(iv) The classes should be of equal width. The principle, however, cannot be rigidly followed.

(v) Indeterminate classes, e.g., the open-end classes such as less than 'a' or greater than 'b' should be avoided as far as possible since they create difficulty in analysis and interpretation.

(vi) The number of classes should neither be too large nor too small. It should preferably lie between 5 and 15. However. the number of classes may be more than 15 depending upon the total frequency and the details required. But it is desirable that it is not less than 5 since in that case the classification may not reveal the essential characteristics of the population.

Terms used in frequency distribution

Class Interval: The whole range of variable values is classified in some groups in the form of intervals. Each interval is called a class interval.

Class Frequency: The number of observations in a class is termed as the frequency of the class or class frequency.

Class limits and Class boundaries: Class limits are the two endpoints of a class interval which are used for the construction of a frequency distribution. The lowest value of the variable that can be included in a class interval is called the lower class limit of that class interval. The highest value of the variable that can be included in a class interval is called the upper-class limit of that class interval. In the table 2.3, the class intervals are 17-21, 22-26, 27-31 and 32-36. Here, say for the class 17-21, the lower-class limit is 17 and the upper-class limit is 21. Both 17 and 21 are part of this class. This is called inclusive class. Another type of class is exclusive class as shown below in table 2.4:

Table 2.4: Exclusive Class Grouped Frequency Table

Temperature	Frequency
17-21	17
21-25	7
25-29	10
29-33	9
33-37	7
Total	50

In table 2.4 upper values are excluded from the class i.e., in the class 17-21 only values from 17 to 20 are taken and the values of 21 in considered in the next class. Such type of distribution is known as exclusive class.

Open-end classes: It may be the case that some values in the data set are extremely small compared to the other values of the data set and similarly some values are extremely large in comparison. Then what we do is we do not use the lower limit of the first class and the upper limit of the last class. Such classes are called open end classes.

Table 2.5: Open-end Class Grouped Frequency Table

Temperature	Frequency
Below 21	17
21-25	7
25-29	10
29-33	9
Above 33	7
Total	50

Size of the Class: The length of the class is called the class width. It is also known as class size.

Class interval or size of the class = Upper Limit – Lower Limit

Mid-point of the Class: The midpoint of a class interval is called Mid-point of the Class. It is the representative value of the entire class.

Mid-point of the class = (Upper Limit + Lower Limit) / 2

Continuous Frequency Distribution: If we deal with a continuous variable, it is not possible to arrange the data in the class intervals of above type. Let us consider the distribution of age in years. If class intervals are 15-19, 20-24 then the persons with ages between 19 and 20 years are not taken into consideration. In such a case we form the class intervals as shown below in table 2.6.

Table 2.6: Continuous Data

Age(in Years)
Below 5
5 or more but less than 10
10 or more but less than 15
15 or more but less than 20
20 or more but less than 25
…

As all cases have been covered in this table. But it is difficult to perform calculations using this table, therefore data is represented as in the table 2.7.

Table 2.7: Continuous Data using Classes

Age(in Years)
0-5
5-10
10-15
15-20
20-25
25-30

This form of frequency distribution is known as continuous frequency distribution. It should be clearly understood that in the above classes, the upper limits of each class are excluded from the respective classes. Such classes in which the upper limits are excluded from the respective classes and are included in the immediate next class are known as 'exclusive classes' and the classification is termed as 'exclusive type classification'.

2.2.3 Graphical Representation of a Frequency Distribution

It is often useful to represent a frequency distribution by means of a diagram which makes the data easily understandable and conveys the general information about the data. Diagrammatic representation also facilitates the comparison of two or more frequency distributions.

Graphs are charts consisting of points, lines and curves. Charts are drawn on graph sheets. Scales are to be chosen suitably in both X and Y axes so that entire data can be presented in the graph sheet. Statistical measures such as quartiles, median and mode can be found from the appropriate graph. Graphs are useful for analysis of time series, regression analysis, business forecasting, interpolation, extrapolation, etc.

Types of graphs

Graphs in statistics are broadly divided into two categories.

i) Graphs of time series or Historigrams
ii) Graphs of frequency distribution

Graphs of time series or Historigrams: A historigram is a graph to show a time series. It shows the fluctuation of a variable over a given period. X axis is used to denote the time and Y axis the value of the variable. Each pair of (time, variable) is denoted by a point on the graph. After plotting all such points, successive points are joined by straight lines. The resulting curve is historigram.

For example, let us draw a historigram (as in figure 2.1) to show the population in various census years with the given data as in table 2.8.

Table 2.8: Population in Various Census Years

Census Year	1951	1961	1972	1981	1998	2017
Population(in Million)	33.44	42.88	65.31	83.78	130.58	200.17

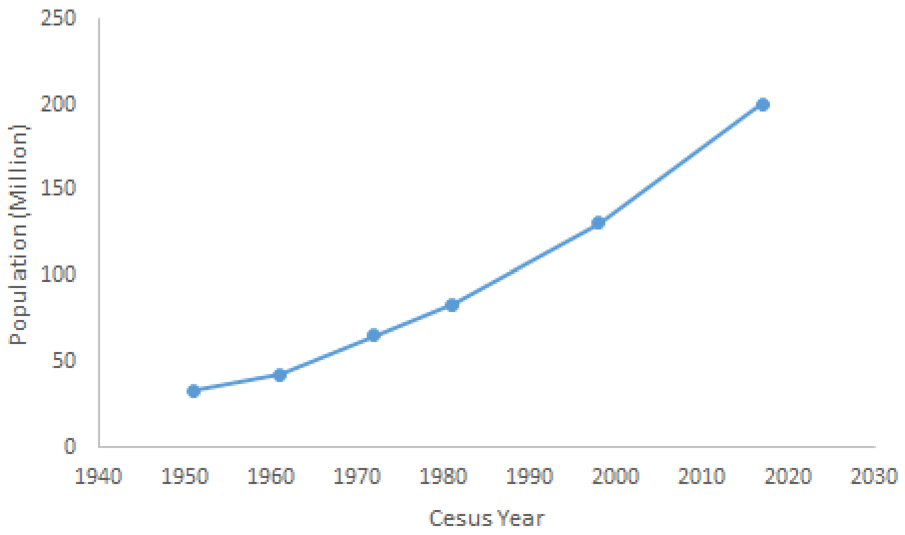


Figure 2.1: Historigram for data in table 2.8

Graphs of frequency distribution: There are various types of graphs of frequency distribution such as:

a) Histogram
b) Frequency polygon used to present continuous frequency distribution
c) Frequency curve

d) Ogive curve used to represent cumulative frequency distribution

e) Pie chart used to represent relative frequency.

f) Bar Diagram used to compare the frequencies.

HISTOGRAM: In drawing the histogram of a given continuous frequency distribution we first mark off along the x-axis all the class intervals on a suitable scale. On each class interval rectangles are drawn with heights proportional to the frequency of the corresponding class interval. The diagram of continuous rectangles so obtained is called histogram.

For examples, the table 2.9 gives the life times of 400 bulbs.

Table 2.9: Lifetime of Bulbs

Lifetime (in hours)	Number of bulbs
300 – 400	14
400 – 500	56
500 – 600	60
600 – 700	86
700 – 800	74
800 – 900	62
900 – 1000	48

The histogram for table 2.9 is:

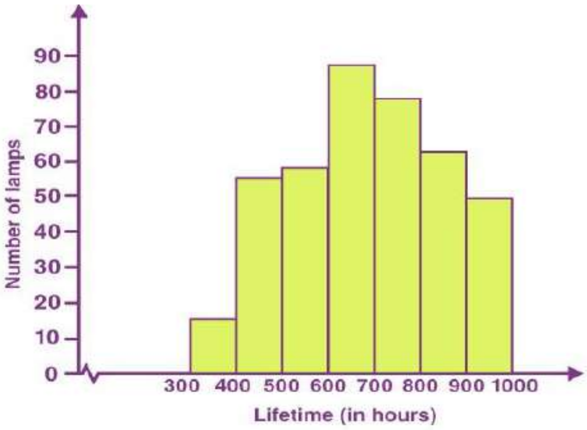


Figure 2.2: Histogram for data in table 2.9

Histogram in Python: It is important to represent the given data using histogram. But it is more important to represent it using a tool. In this course, we will be using Python as a tool for representing any data. From this point onwards, we will learn the various representations using Python too.

Creating Numpy Histogram: Numpy has a built-in numpy.histogram() function which represents the frequency of data distribution in the graphical form. The rectangles having equal horizontal size corresponds to class interval called bin and variable height corresponding to the frequency. It can be created using following statement:

numpy.histogram(data, bins=10, range=None, normed=None, weights=None, density=None).

Where,

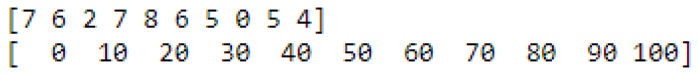
Attribute	*Parameter*
data	*array or sequence of array to be plotted*
bins	*int or sequence of str defines number of equal width bins in a range, default is 10*
range	*optional parameter sets lower and upper range of bins*
normed	*optional parameter same as density attribute, gives incorrect result for unequal bin width*
weights	*optional parameter defines array of weights having same dimensions as data*
density	*optional parameter if False result contain number of sample in each bin, if True result contain probability density function at bin*
data	*array or sequence of array to be plotted*
bins	*int or sequence of str defines number of equal width bins in a range, default is 10*
range	*optional parameter sets lower and upper range of bins*
normed	*optional parameter same as density attribute, gives incorrect result for unequal bin width*
weights	*optional parameter defines array of weights having same dimensions as data*
density	*optional parameter if False result contain number of sample in each bin, if True result contain probability density function at bin*

The creation of Numpy histogram can be better understood by the following programs:

```
# Program 2.1: Histogram Numeric Representation
# Import libraries
import numpy as np
# Creating dataset
a = np.random.randint(100, size =(50))
```

```
# Creating histogram
np.histogram(a, bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
hist, bins = np.histogram(a, bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
# printing histogram
print (hist)
print (bins)
```

Output:



```
[7 6 2 7 8 6 5 0 5 4]
[  0  10  20  30  40  50  60  70  80  90 100]
```

The above numeric representation of histogram can be converted into a graphical form. The plt() function present in pyplot submodule of Matplotlib takes the array of dataset and array of bin as parameter and creates a histogram of the corresponding data values.

```
# Program 2.2: Histogram
# import libraries
from matplotlib import pyplot as plt
import numpy as np
# Creating dataset
a = np.random.randint(100, size =(50))
# Creating plot
fig = plt.figure(figsize =(10, 7))
plt.hist(a, bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
plt.title("Numpy Histogram")
# show plot
plt.show()
```

Output:

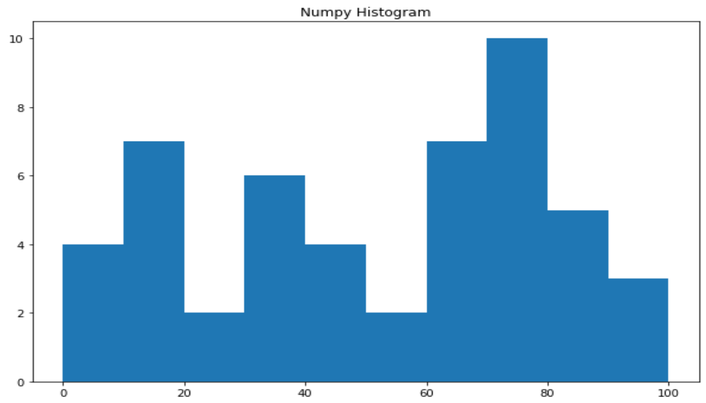


Figure 2.3: Histogram using Program 2.2

Creating Histogram using Matplotlib: To create a histogram the first step is to create bin of the ranges, then distribute the whole range of the values into a series of intervals, and the count the values which fall into each of the intervals. Bins are clearly identified as consecutive, non-overlapping intervals of variables. The matplotlib.pyplot.hist() function is used to compute and create histogram of x.

The following table shows the parameters accepted by matplotlib.pyplot.hist() function :

Attribute	*parameter*
X	*array or sequence of array*
bins	*optional parameter contains integer or sequence or strings*
density	*optional parameter contains boolean values*
range	*optional parameter represents upper and lower range of bins*
histtype	*optional parameter used to creae type of histogram [bar, barstacked, step, stepfilled], default is "bar"*
align	*optional parameter controls the plotting of histogram [left, right, mid]*
weights	*optional parameter contains array of weights having same dimensions as x*
bottom	*location of the baseline of each bin*
rwidth	*optional parameter which is relative width of the bars with respect to bin width*

Attribute	*parameter*
color	*optional parameter used to set color or sequence of color specs*
label	*optional parameter string or sequence of string to match with multiple datasets*
Log	*optional parameter used to set histogram axis on log scale*

Frequency Polygon and Curves: For an ungrouped distribution, the frequency polygon is obtained by plotting points with corresponding frequencies and joining the plotted points by means of straight lines. For a grouped frequency distribution, the points are mid-values of the class intervals. For equal class intervals the frequency polygon can be obtained by joining the middle Points of the upper sides of the adjacent rectangles of the histogram by means of straight lines. If the class intervals are of small width the polygon can be approximated by a smooth curve. The frequency curve can be obtained by drawing a smooth freehand curve through the vertices of the frequency polygon.

For example, let us present the following data given for **a batch of 400 students, the height of students is provided in the table 2.10, using frequency polygon.**

Table 2.10: Heights of Students Data

Height (In cm)	Number of Students
140-150	74
150-160	163
160-170	135
170-180	28
Total	400

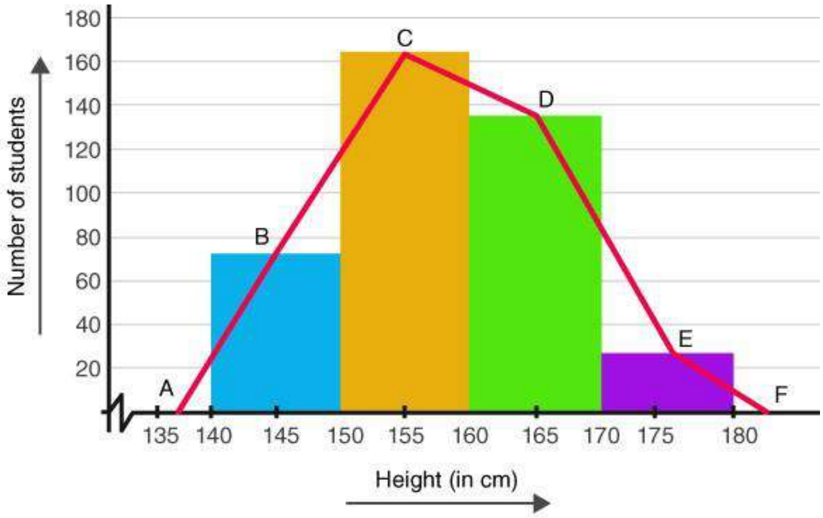


Figure 2.4: Frequency Polygon for the data in table 2.10

ABCDEF represents the given data graphically in form of frequency polygon as shown above in figure 2.4.

Now let us consider an example to draw a frequency curve.

Table 2.11: Data for Frequency Curve

Seed Yield (gms)	No. of Plants
2.5-3.5	4
3.5-4.5	6
4.5-5.5	10
5.5-6.5	26
6.5-7.5	24
7.5-8.5	15
8.5-9.5	10
9.5-10.5	5

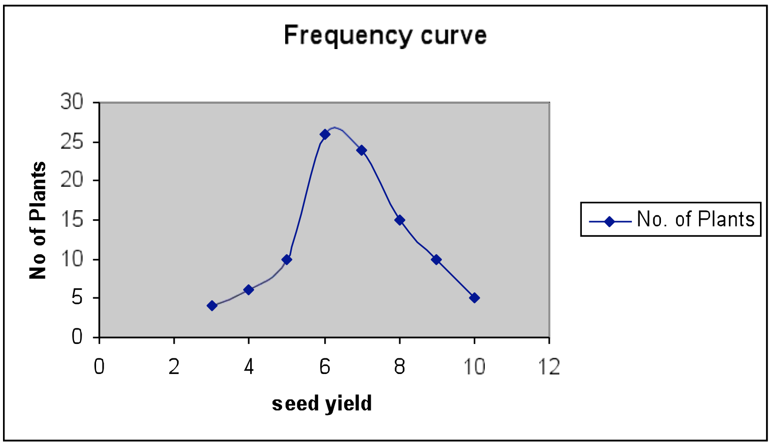


Figure 2.5: Frequency Curve for data in table 2.11

Drawing a frequency polygon in Python may be understood from the following example.

Suppose you have only the angle values for a set of data. Now you need to plot an angle distribution curve i.e., angle on the x axis v/s no. of times/frequency of angle occurring on the y axis. These are the angles sorted out for a set of data: -

[98.1706427, 99.09896751, 99.10879006, 100.47518838, 101.22770381, 101.70374296, 103.15715294, 104.4653976,105.50441485, 106.82885361, 107.4605319, 108.93228646, 111.22463712, 112.23658018, 113.31223886, 113.4000603, 114.14565594, 114.79809084, 115.15788861, 115.42991416, 115.66216071, 115.69821092, 116.56319054, 117.09232139, 119.30835385, 119.31377834, 125.88278338, 127.80937901, 132.16187185, 132.61262906, 136.6751744, 138.34164387,]

The data can easily be represented using Python with the help of following code:

```
# Program 2.3: Frequency Polygon
from matplotlib import pyplot as plt
import numpy as np
angles = [98.1706427, 99.09896751, 99.10879006, 100.47518838, 101.22770381,
101.70374296, 103.15715294, 104.4653976, 105.50441485, 106.82885361, 107.4605319,
108.93228646, 111.22463712, 112.23658018, 113.31223886, 113.4000603, 114.14565594,
114.79809084, 115.15788861, 115.42991416, 115.66216071, 115.69821092, 116.56319054,
117.09232139, 119.30835385, 119.31377834, 125.88278338, 127.80937901, 132.16187185,
132.61262906, 136.6751744, 138.34164387, ]
hist,edges = np.histogram(angles, bins=20)
bin_centers = 0.5*(edges[:-1] + edges[1:])
bin_widths = (edges[1:]-edges[:-1])
plt.bar(bin_centers,hist,width=bin_widths)
plt.plot(bin_centers, hist,'r')
plt.xlabel('angle [$^\circ$]')
plt.ylabel('frequency')
plt.show()
```

Output:

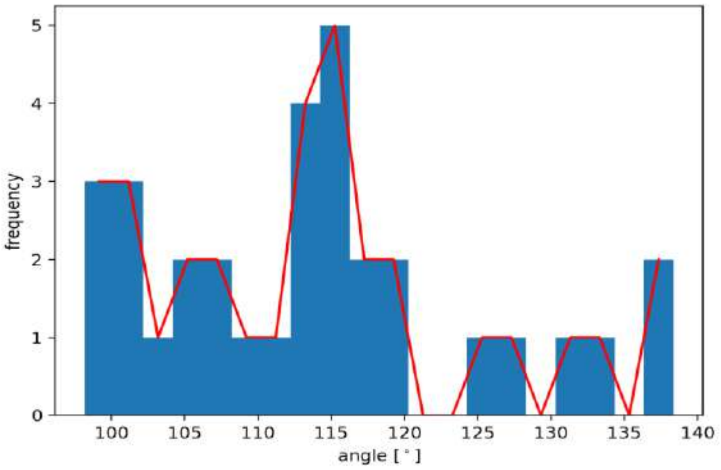


Figure 2.6: Frequency Polygon

2.2.4 Cumulative Frequency Distribution

Cumulative frequency is defined as a running total of frequencies. The frequency of an element in a set refers to how many of that element there are in the set. Cumulative frequency can also be defined as the sum of all previous frequencies up to the current point.

Consider an example which shows the ages of participants in a certain class. We need to draw a cumulative frequency table for the data given in table 2.12.

Table 2.12: Frequency Table

Age	Frequency
10	3
11	18
12	13
13	12
14	7
15	27

The cumulative frequency table for the above data can be drawn as table 2.13. In this frequencies are the sum of the current frequency and previous frequencies. In other words, we can say that cumulative frequency shows the number of participants under or equal to the age of 10, 11, 12, 13, 14 and 15 respectively.

Table 2.13: Cumulative Frequency Table

Age	Frequency	Cumulative Frequency
10	3	3
11	18	18+3=21
12	13	21+13=34
13	12	34+12=46
14	7	46+7=53
15	27	53+27=80

However, there are two kinds of cumulative frequency distribution.

i) Less than cumulative frequency distribution

ii) More than cumulative frequency distribution

Less than cumulative frequency distribution: Frequency distribution both discrete and continuous are to be taken in ascending order. The total of the frequencies from the beginning up to and including each frequency is found. That cumulative frequency shows how many items are less than or equal to the corresponding value of the class interval.

More than cumulative frequency distribution: Frequency distribution both discrete and continuous are to be taken in ascending order. The total of the frequencies from the end up to and including each frequency is found. That cumulative frequency shows how many items are more than or equal to the corresponding value of the class interval.

Consider an example for both these types of cumulative frequencies for ungrouped data using the following table 2.14.

Table 2.14: Less than and More than Cumulative Frequency Curve

Weekly Wages (X)	Number of Workers (F)	Less than Cumulative Frequency	More Than Cumulative Frequency
120	8	8	30
150	12	20	22
240	8	28	10
300	2	30	2

Consider another example for cumulative frequencies using grouped data as shown in the following table 2.15.

Table 2.15: Less than and More than Cumulative Frequency Curve for Grouped Data

Marks (x)	No. of Students (f)	Marks below	No. of students	Marks above	No. of students
		Upper limit	Less than Cumulative Frequency (C.F.)	Lower limit	More than Cumulative Frequency (C.F.)
0-20	2	20	2	0	40
20-40	7	40	9	20	38
40-60	15	60	24	40	31
60-80	9	80	33	60	16
80-100	7	100	40	80	7
Total	40				

Ogive Curve for Cumulative Frequency

Let us now draw Ogive curve for both less than and greater(more) than using an example. Suppose we are given with weekly wages of various workers as shown in the table 2.16:

Table 2.16: Grouped Data

Weekly Wages (x)	No. of Workers (f)
0-20	41
20-40	51
40-60	64
60-80	38
80-100	7

First let us convert this table into less than c.f. and more than c.f.. **Table 2.17: Less than and More than Cumulative Frequency Curve for data in table 2.16**

Weekly Wages (x)	No. of Workers (f)	C.F.(Less than)	C.F.(More than)
0-20	41	41	201
20-40	51	92	160
40-60	64	156	109
60-80	38	194	45
80-100	7	201	7

Less than ogive: Upper limits of class intervals are marked on the x-axis and less than type cumulative frequencies are taken on y-axis. For drawing less than type curve, points (20, 41), (40, 92), (60, 156), (80, 194), (100, 201) are plotted on the graph paper and these are joined by free hand to obtain the less than ogive.

Greater than ogive: Lower limits of class interval are marked on x-axis and greater than type cumulative frequencies are taken on y-axis. For drawing greater than type curve, points (0, 201), (20, 160), (40, 109), (60, 45) and (80, 7) are plotted on the graph paper and these are joined by free hand to obtain the greater than type ogive.

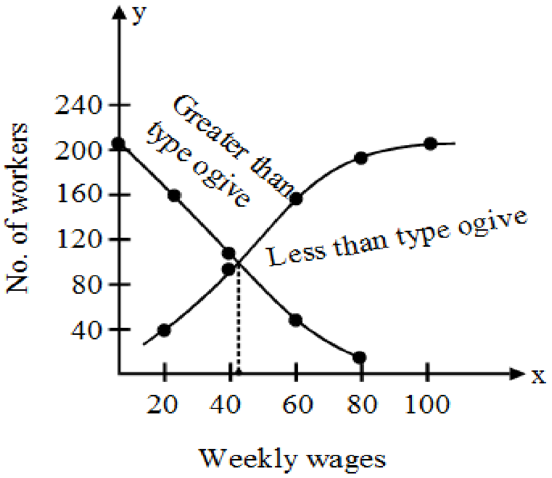


Figure 2.7: Less than and Greater than Ogive

Drawing Ogive in Python

We can draw both types of ogives in python. Let us understand by taking suitable examples. First we consider more than ogive. The more than ogive graph shows the number of values greater than the class intervals. The resultant graph shows the number of values in between the class interval, e.g., 0-10,10-20 and so on. Let us take a dataset, and we will now plot it's more than ogive graph- [22,87,5,43,56,73, 55,54,11,20,51,5,79,31,27]. For this data the table 2.18 can be created as follows:

Table 2.18: Data for drawing more than Ogive

Class-Interval (x)	Frequency (f)	Cumulative Frequency (Less than)
0-10	2	2
10-20	1	3
20-30	3	6
30-40	1	7
40-50	1	8
50-60	4	12
60-70	0	12
70-80	2	14
80-90	1	15

Approach for drawing the ogive follows three steps:

(i) Import the modules (matplotlib and numpy)
(ii) Calculate the frequency and cumulative frequency of the data.
(iii) Plot it using the plot() function.

```
# Program 2.4: More than Ogive
# importing modules
import numpy as np
import matplotlib.pyplot as plt
# creating dataset
data = [22, 87, 5, 43, 56, 73, 55, 54, 11, 20, 51, 5, 79, 31, 27]
# creating class interval
classInterval = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
# calculating frequency and class interval
values, base = np.histogram(data, bins=classInterval)
# calculating cumulative sum
cumsum = np.cumsum(values)
# plotting the ogive graph
plt.plot(base[1:], cumsum, color='red', marker='o', linestyle='-')
plt.title('Ogive Graph')
plt.xlabel('Marks in End-Term')
plt.ylabel('Cumulative Frequency')
```

Output:

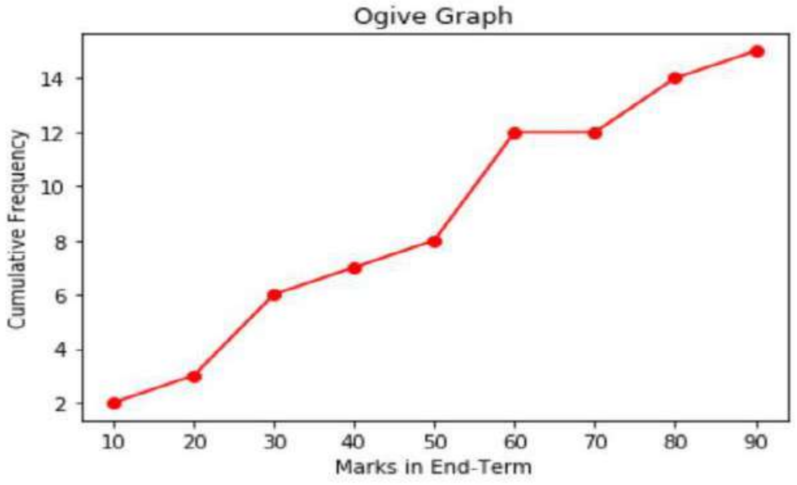


Figure 2.8: More than Ogive

Similarly, we can draw the less than ogive using following example.

In this example, we will plot less than Ogive graph which will show the less than values of class intervals. Dataset: [44,27,5,2,43,56,77,53,89,54,11,23, 51,5,79,25,39]. For this data the table can be created as follows:

Table 2.19: Data for drawing less than Ogive

Class-Interval (x)	Frequency (f)	Cumulative Frequency (More than)
0-10	3	17
10-20	1	14
20-30	3	13
30-40	1	10
40-50	2	9
50-60	4	7
60-70	0	3
70-80	2	3
80-90	1	1

Approach is same as above only the cumulative sum that we will calculate will be reversed using **flipud()** function present in the numpy library.

```
# Program 2.5: Less than Ogive
# importing modules
import numpy as np
import matplotlib.pyplot as plt
# creating dataset
data = [44, 27, 5, 2, 43, 56, 77, 53, 89, 54, 11, 23, 51, 5, 79, 25, 39]
# creating class interval
classInterval = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
# calculating frequency and intervals
values, base = np.histogram(data, bins=classInterval)
# calculating cumulative frequency
cumsum = np.cumsum(values)
# reversing cumulative frequency
res = np.flipud(cumsum)
# plotting ogive
plt.plot(base[1:], res, color='brown', marker='o', linestyle='-')
plt.title('Ogive Graph')
plt.xlabel('Marks in End-Term')
plt.ylabel('Cumulative Frequency')
```

Output:

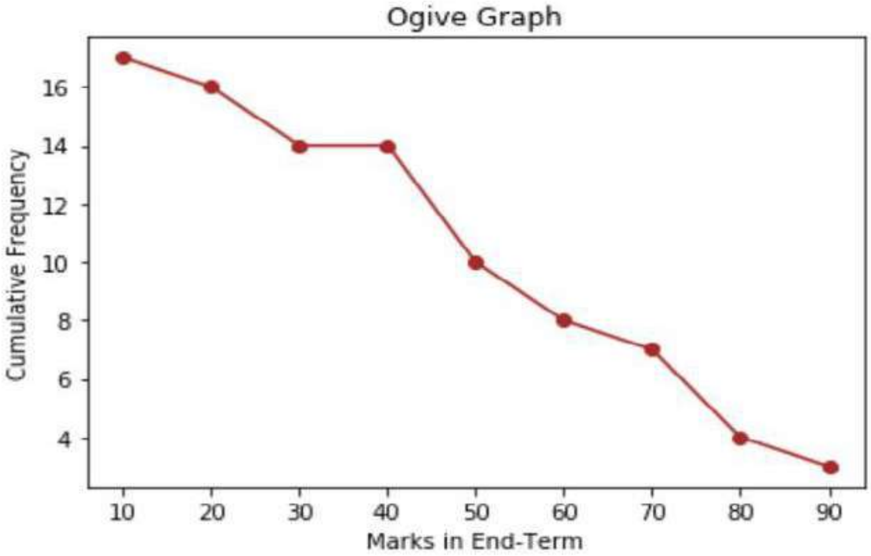


Figure 2.9: Less than Ogive

2.2.5 Pie Charts for Data Representation

A pie chart is a type of graph that represents the data in the circular graph. The slices of pie show the relative size of the data. It is a type of pictorial representation of data. A pie chart requires a list of categorical variables and the numerical variables. Here, the term "pie" represents the whole, and the "slices" represent the parts of the whole. Each slice denotes a proportionate part of the whole. The pie chart is an important type of data representation. It contains different segments and sectors in which each segment and sectors of a pie chart forms a certain portion of the total(percentage). The total of all the data is equal to 360°. **The total value of the pie is always 100%.**

The steps to design a pie chart are:

- Categorize the data
- Calculate the total
- Divide the categories
- Convert into percentages
- Finally, calculate the degrees
- Therefore, the pie chart formula is given as

Formula for pie chart = (Given Data/Total value of Data) × 360°.

Consider an example to draw a pie chart in a step by step manner:

Imagine a teacher surveys her class on the basis of their favourite Sports:

Football	**Hockey**	**Cricket**	**Basketball**	**Badminton**
10	**5**	**5**	**10**	**10**

This can be represented in the following table using above-said steps:

Table 2.20: Data based on favourite sports

Sports	No. of Students	Percentage	Degree(pie)
Football	10	(10/40)*100=25%	(10/40)*360=90^0
Hockey	5	(5/40)*100=12.5%	(5/40)*360=45^0
Cricket	5	(5/40)*100=12.5%	(5/40)*360=45^0
Basketball	10	(10/40)*100=25%	(10/40)*360=90^0
Badminton	10	(10/40)*100=25%	(10/40)*360=90^0
Total	40		

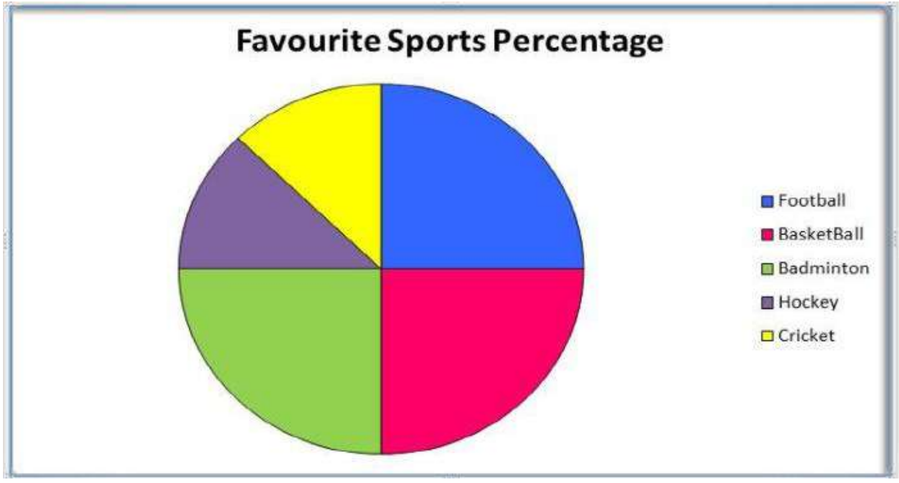


Figure 2.10: Pie Chart for table 2.20

Pie-chart using Python

Matplotlib API has pie() function in its pyplot module which create a pie chart representing the data in an array.

Syntax: matplotlib.pyplot.pie(data, explode=None, labels=None, colors=None, autopct=None, shadow=False)

Where,

data *represents the array of data values to be plotted, the fractional area of each slice is represented by data/sum(data). If sum(data)<1, then the data values returns the fractional area directly, thus resulting pie will have empty wedge of size 1-sum(data).*
labels *is a list of sequence of strings which sets the label of each wedge.*
color *attribute is used to provide color to the wedges.*
autopct *is a string used to label the wedge with their numerical value.*
shadow *is used to create shadow of wedge.*

Let’s create a simple pie chart using the pie() function:

```
# Program 2.6: Pie Chart

# Import libraries

from matplotlib import pyplot as plt
```

```
import numpy as np
# Creating dataset
cars = ['AUDI', 'BMW', 'FORD', 'TESLA', 'JAGUAR', 'MERCEDES']
data = [23, 17, 35, 29, 12, 41]
# Creating plot
fig = plt.figure(figsize =(10, 7))
plt.pie(data, labels = cars)
# show plot
plt.show()
```

Output:

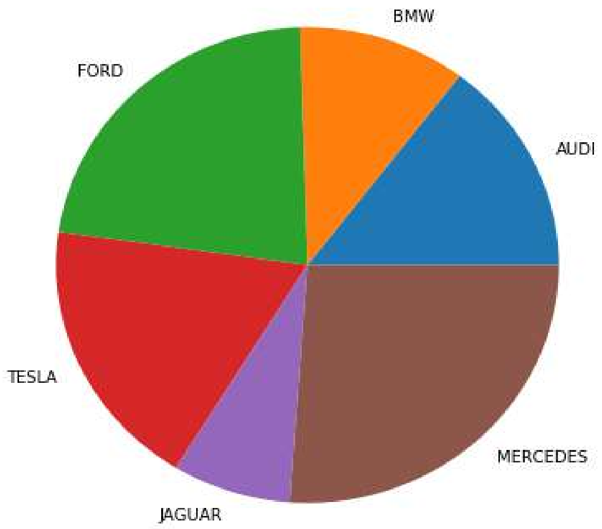


Figure 2.11: Pie Chart

2.2.6 Bar Diagram for Data Representation

Bar graphs are the pictorial representation of data in the form of vertical or horizontal rectangular bars, where the length of bars are proportional to the measure of data. They are also known as bar charts. Bar graphs are one of the means of data handling in statistics.

The bars drawn are of uniform width, and the variable quantity is represented on one of the axes. Also, the measure of the variable is depicted on the other axes. The heights or the lengths of the bars denote the value of the variable, and these graphs are also used to compare certain quantities. The frequency distribution tables can be easily represented using bar charts which simplify the calculations and understanding of data.

The three major attributes of bar graphs are:

- The bar graph helps to compare the different sets of data among different groups easily.
- It shows the relationship using two axes, in which the categories on one axis and the discrete values on the other axis.
- The graph shows the major changes in data over time.

Types of Bar Charts

The bar graphs can be vertical or horizontal. The primary feature of any bar graph is its length or height. If the length of the bar graph is more, then the values are greater than any given data.

Bar graphs normally show categorical and numeric variables arranged in class intervals. They consist of an axis and a series of labelled horizontal or vertical bars. The bars represent frequencies of distinctive values of a variable or commonly the distinct values themselves. The number of values on the x-axis of a bar graph or the y-axis of a column graph is called the scale.

The types of bar charts are as follows:

- Vertical bar chart
- Horizontal bar chart

Even though the graph can be plotted using horizontally or vertically, the most usual type of bar graph used is the vertical bar graph. The orientation of the x-axis and y-axis are changed depending on the type of vertical and horizontal bar chart. Apart from the vertical and horizontal bar graph, the two different types of bar charts are:

- Grouped Bar Graph
- Stacked Bar Graph

Now, let us discuss the four different types of bar graphs.

Vertical Bar Graphs: When the grouped data are represented vertically in a graph or chart with the help of bars, where the bars denote the measure of data, such graphs are called vertical bar graphs. The data is represented along the y-axis of the graph, and the height of the bars shows the values.

Horizontal Bar Graphs: When the grouped data are represented horizontally in a chart with the help of bars, then such graphs are called horizontal bar graphs, where the bars show the measure of data. The data is depicted here along the x-axis of the graph, and the length of the bars denote the values.

Grouped Bar Graph: The grouped bar graph is also called the clustered bar graph, which is used to represent the discrete value for more than one object that shares the same category. In this type of bar chart, the total number of instances are combined into a single bar. In other words, a grouped bar graph is a type of bar graph in which different sets of data items are compared. Here, a single colour is used to represent the specific series across the set. The grouped bar graph can be represented using both vertical and horizontal bar charts.

Stacked Bar Graph: The stacked bar graph is also called the composite bar chart, which divides the aggregate into different parts. In this type of bar graph, each part can be represented using different colours, which helps to easily identify the different categories. The stacked bar chart requires specific labelling to show the different parts of the bar. In a stacked bar graph, each bar represents the whole and each segment represents the different parts of the whole.

Drawing a Bar Graph: In order to visually represent the data using the bar graph, we need to follow the steps given below.

- First, decide the title of the bar graph.
- Draw the horizontal axis and vertical axis.
- Now, label the horizontal axis.
- Write the names on the horizontal axis.
- Now, label the vertical axis.
- Finalise the scale range for the given data.
- Finally, draw the bar graph.

Bar Graph Examples: To understand the above types of bar graphs, consider the following examples:

In a firm of 400 employees, the percentage of monthly salary saved by each employee is given in the following table. Represent it through a bar graph.

Table 2.21: Data of Savings

Savings (in percentage)	Number of Employees (Frequency)
20	105
30	199
40	29
50	73
Total	400

The given data can be represented as a vertical bar graph:

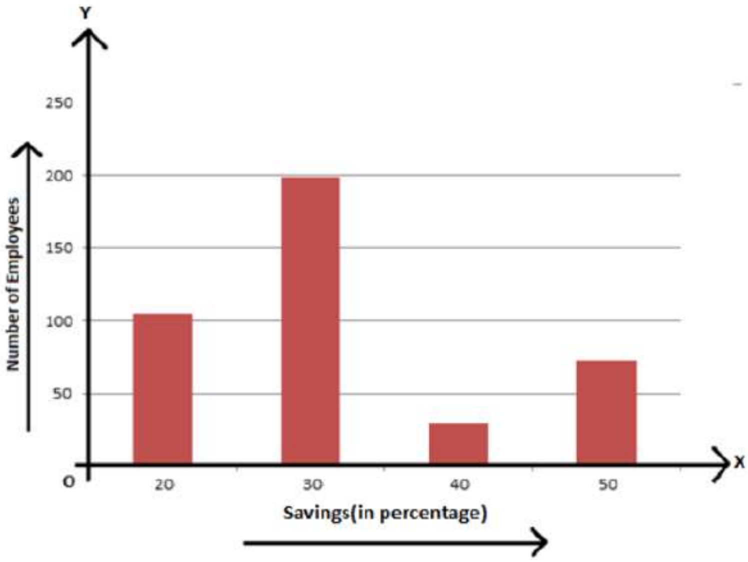


Figure 2.12: Vertical Bar Diagram for table 2.21

This can also be represented using a horizontal bar graph as follows:

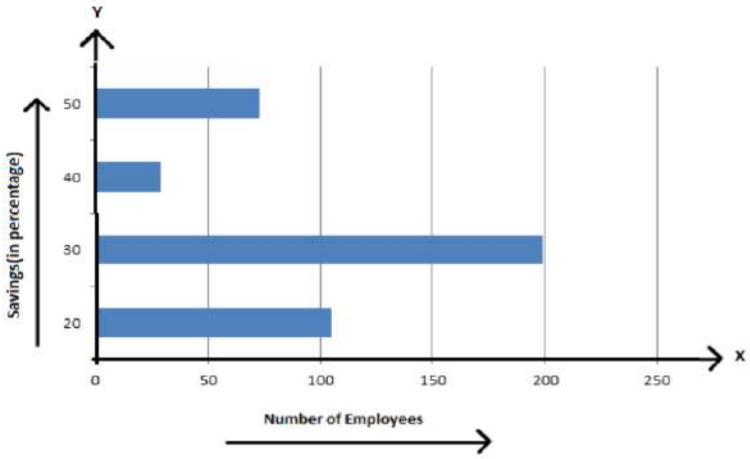


Figure 2.13: Horizontal Bar Diagram for table 2.21

Let as consider another example of grouped bar diagram: **A cosmetic company manufactures 4 different shades of lipstick. The sale for 6 months is shown in the table. Represent it using bar charts.**

Table 2.22: Data of Lipsticks

Month	Sales (in units)			
	Shade 1	Shade 2	Shade 3	Shade 4
January	4500	1600	4400	3245
February	2870	5645	5675	6754
March	3985	8900	9768	7786
April	6855	8976	9008	8965
May	3200	5678	5643	7865
June	3456	4555	2233	6547

The graph given below depicts the following data:

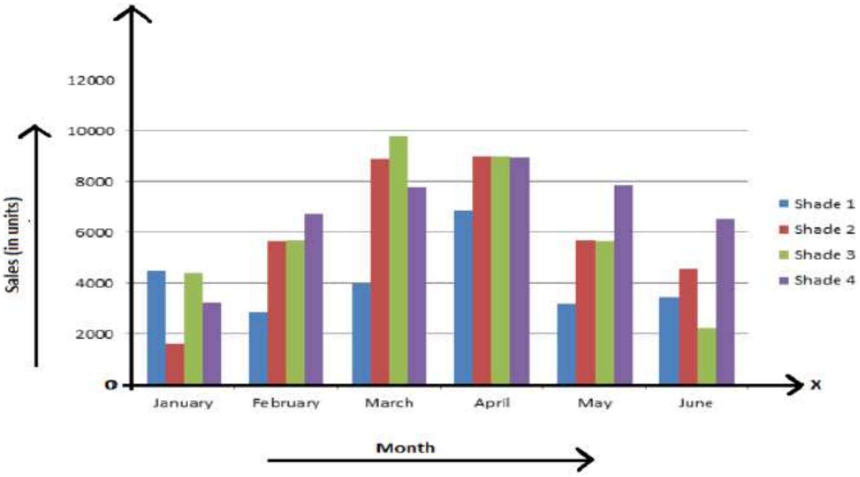


Figure 2.13: Grouped Bar Diagram for table 2.22

Bar diagram in Python

The **matplotlib** API in Python provides the bar() function. The syntax of the bar() function to be used with the axes is as follows:-

plt.bar(x, height, width, bottom, align)

The following program creates a bar plot bounded with a rectangle depending on the given parameters. Following is a simple example of the bar plot, which represents the number of students enrolled in different courses of an institute.

```
# Program 2.7: Bar Diagram
import numpy as np
import matplotlib.pyplot as plt
# creating the dataset
data = {'C':20, 'C++':15, 'Java':30,     'Python':35}
courses = list(data.keys())
values = list(data.values())
fig = plt.figure(figsize = (10, 5))
# creating the bar plot
plt.bar(courses, values, color ='maroon',       width = 0.4)
plt.xlabel("Courses offered")
plt.ylabel("No. of students enrolled")
plt.title("Students enrolled in different courses")
plt.show()
```

Output:

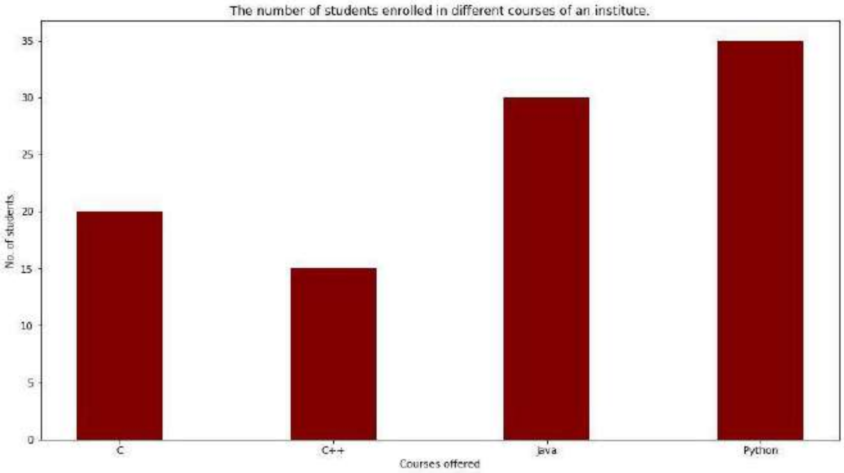


Figure 2.14: Bar Diagram

Horizontal charts can also be designed using Matplotlib. To create a horizontal bar chart:

```
#Program 2.8: Horizontal Bar Diagram

import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt
objects = ('Python', 'C++', 'Java', 'Perl', 'Scala', 'Lisp')
y_pos = np.arange(len(objects))
performance = [10,8,6,4,2,1]
plt.barh(y_pos, performance, align='center', alpha=0.5)
plt.yticks(y_pos, objects)
plt.xlabel('Usage')
plt.title('Programming language usage')
plt.show()
```

Output:

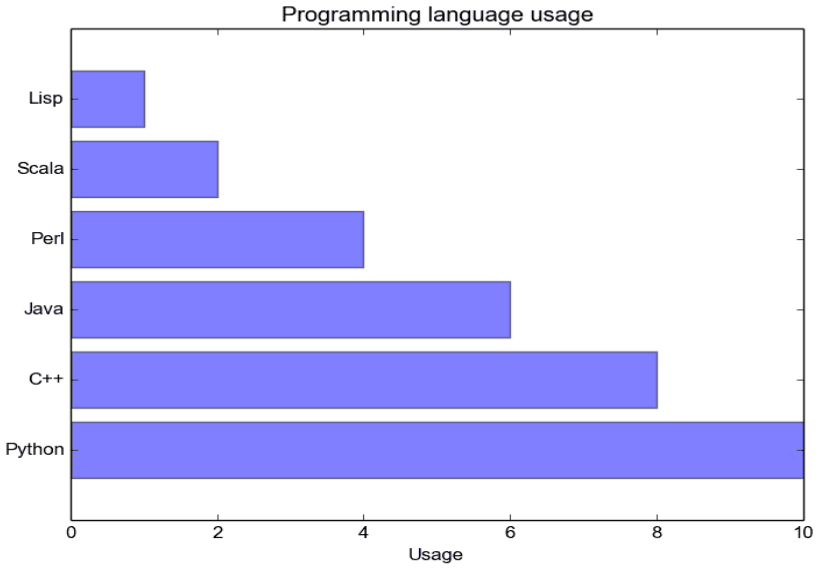


Figure 2.15: Horizontal Bar Diagram

Multiple bar plots: Multiple bar plots are used when comparison among the data set is to be done when one variable is changing. It can be drawn using python as shown in the following program.

```
#Program 2.9: Multiple Bar Plot
import numpy as np
import matplotlib.pyplot as plt
# set width of bar
barWidth = 0.25
fig = plt.subplots(figsize =(12, 8))
# set height of bar
IT = [12, 30, 1, 8, 22]
ECE = [28, 6, 16, 5, 10]
CSE = [29, 3, 24, 25, 17]
# Set position of bar on X axis
br1 = np.arange(len(IT))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]
# Make the plot
plt.bar(br1, IT, color ='r', width = barWidth, edgecolor ='grey', label ='IT')
plt.bar(br2, ECE, color ='g', width = barWidth, edgecolor ='grey', label ='ECE')
plt.bar(br3, CSE, color ='b', width = barWidth, edgecolor ='grey', label ='CSE')

# Adding Xticks

plt.xlabel('Branch', fontweight ='bold', fontsize = 15)

plt.ylabel('Students passed', fontweight ='bold', fontsize = 15)

plt.xticks([r + barWidth for r in range(len(IT))], ['2015', '2016', '2017', '2018', '2019'])

plt.legend()

plt.show()
```

Output:

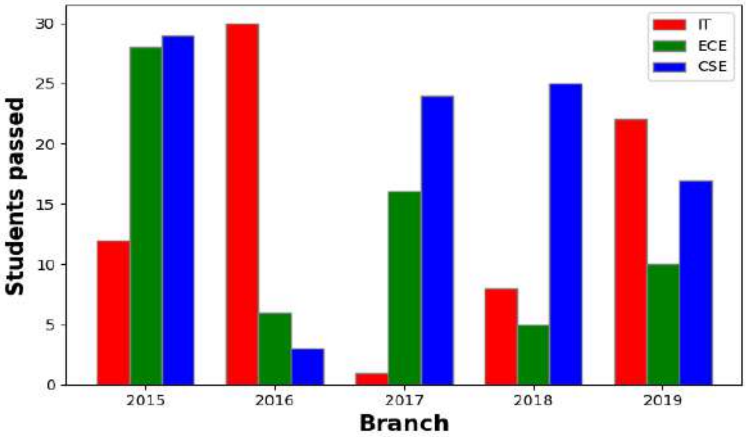


Figure 2.16: Multiple Bar Diagram

Stacked bar plot: Stacked bar plots represent different groups on top of one another. The height of the bar depends on the resulting height of the combination of the results of the groups. It goes from the bottom to the value instead of going from zero to value. The following bar plot represents the contribution of boys and girls in the team.

```
#Program 2.10: Stacked Bar Plot

import numpy as np

import matplotlib.pyplot as plt

N = 5

boys = (20, 35, 30, 35, 27)

girls = (25, 32, 34, 20, 25)

boyStd = (2, 3, 4, 1, 2)

girlStd = (3, 5, 2, 3, 3)

ind = np.arange(N)

width = 0.35
```

```
fig = plt.subplots(figsize =(10, 7))
p1 = plt.bar(ind, boys, width, yerr = boyStd)
p2 = plt.bar(ind, girls, width, bottom = boys, yerr = girlStd)
plt.ylabel('Contribution')
plt.title('Contribution by the teams')
plt.xticks(ind, ('T1', 'T2', 'T3', 'T4', 'T5'))
plt.yticks(np.arange(0, 81, 10))
plt.legend((p1[0], p2[0]), ('boys', 'girls'))
plt.show()
```

Output:

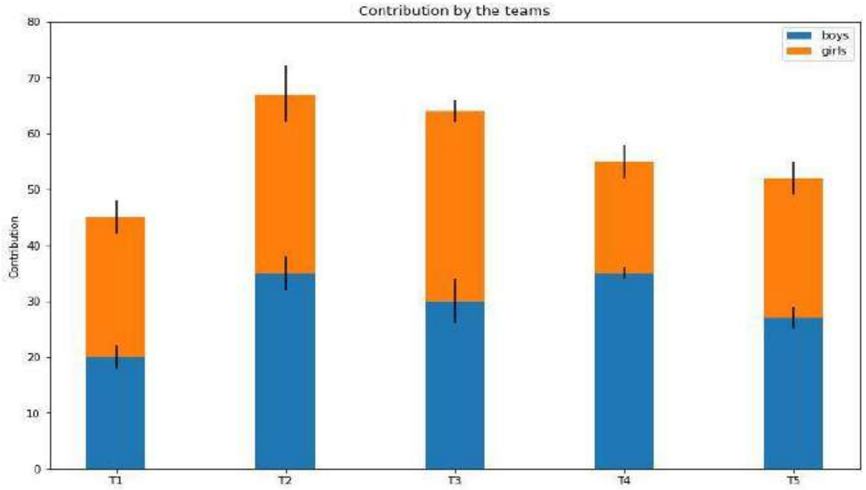


Figure 2.17: Stacked Bar Diagram

2.3 SUMMARY

In this module, representation of statistical data in the tabular and graphical forms has been discussed. The data representation in tabular form has been discussed terms of both grouped and ungrouped data. The presentation of continuous and discrete data has also been explained in this module. Further, the classification of data and its graphical representation has also been discussed. The various types of graphs such as histogram, frequency polygon, bar diagram for presentation of data has been discussed at length. The concept of cumulative frequency distribution along with its tabular and graphical representation has also been explained in detail. The implementation of various types of graphs has been done in python. Overall, this module provides an insight of how to represent the statistical data.

2.4 PRACTICE QUESTION

Q.1 What are grouped and ungrouped frequency distributions? What are their uses? What are the considerations that one has to bear in mind while forming the frequency distribution?

Q.2 Explain the method of constructing Histogram and Frequency Polygon. Which out of these two is better representative of frequencies of (i) a particular group and (ii) whole group.

Q.3 What are the principles governing the choice of (i) Number of class intervals, (ii) The length of the class interval, (iii) The mid-point of the class interval.

Q.4 Write short notes on: (i) Frequency distribution, (ii) Histogram, frequency. polygon and frequency curve, (iii) Ogive.

Q.5 Write a program in python to draw various types of bar diagram considering your own data.

Q.6 Explain various types of bar diagram using suitable examples.

Q.7 What is cumulative frequency distribution? How can you represent c.f. graphically?

Q.8 The following numbers give the weights of 55 students of a class. Prepare a suitable frequency table.

42 74 40 60 82 115 41 61 75 83 63 53 110 76 84 50 67 65 78 77 56 95 68 69 104 80 79 79 54 73 59 81 100 66 49 77 90 84 76 42 64 69 70 80 72 50 79 52 103 96 51 86 78 94 71

(i) Draw the histogram and frequency polygon of the above data. (ii) For the above weights, make a cumulative frequency table and draw the less than ogive.

Q.9 A sample consists of 34 observations recorded correct to the nearest integer, ranging in value from 20l to 331. If it is decided to use seven classes of width 20 integers and to begin the first class at 199·5, find the class limits and class marks of the seven classes.

REFERENCES

- A. Abebe, J. Daniels, J.W.Mckean, “Statistics and Data Analysis”.
- A. Martelli, A. Ravenscroft, S. Holden, “Python in a Nutshell”, OREILLY.
- Clarke, G.M. & Cooke, D., “A Basic course in Statistics”, Arnold.
- David M. Lane, “Introduction to Statistics”.
- Eric Matthes, “Python Crash Course: A Hands-On, Project-Based Introduction to Programming”.
- S.C.Gupta and V.K.Kapoor, “Fundamentals of Mathematical Statistics”, Sultan Chand & Sons, New Delhi.
- Weiss, N.A., “Introductory Statistics”, Addison Wesley

B.Sc.(DATA SCIENCE)

SEMESTER-II

STATISTICAL FOUNDATION

UNIT III: MEASUREMENT OF CENTRAL TENDENCY

STRUCTURE

3.0 OBJECTIVES

In this module, we will try to understand about the various measures of central tendency. The various measures of central tendency i.e., mean, median and mode will be discussed in detail. The various ways to find these measures will be explored in this module. Various measures of dispersion will also be discussed in this module. The various measures of dispersion that would be discussed are range, standard deviation, variance, coefficient of variation and mean deviation. In the end the concept of skewness and kurtosis will also be deliberated.

3.1 INTRODUCTION

This module is designed to know about the different measures of central tendency. In statistics, the measure of central tendency plays very important role as most of the analysis and interpretation surrounds these measures. In most of the cases either mean or median is used for interpreting any data. Mode is used to know about the locality of references in the data. Further, the scatteredness of data also plays an important role in data analysis. The scatteredness can be easily identified by measures of dispersion. There are number of measures of dispersion viz., range, mean deviation, variance, standard deviation, etc. Even though the combination of measures of central tendency and dispersion are good enough for very good interpretation about data, yet there are two more measures named as skewness and kurtosis with the help of which it can be known about the shape of the curve and hence better interpretation of data. This module is a helping hand to the basics of interpreting and analysing the data.

3.2 MAIN CONTENT

3.2.1 Measures of Central Tendency

According to Professor Bowley, averages are "statistical constants which enable us to comprehend in a single effort the significance of the whole." They give us an idea about the concentration of the values in the central part of the distribution. Plainly speaking, an average of a statistical series is the value of the variable which is representative of the entire distribution. The following are the five measures of central tendency that are in common use: (i) Arithmetic Mean or Simply Mean, (ii) Median, (iii) Mode, (iv) Geometric Mean, and (v) Harmonic Mean.

However, in this course, we will be focussing only on first three measures.

Requirements for an Ideal Measure of Central Tendency

According to Professor Yule, the following are the characteristics to be satisfied by-an ideal measure of central tendency;

(i) It should be rigidly defined.
(ii) It should be readily comprehensible and easy to calculate.
(iii) It should be based on all the observations.
(iv) It should be suitable for further mathematical treatment.
(v) It should be affected as little as possible by fluctuations of sampling.
(vi) It should not be affected much by extreme values.

Arithmetic Mean

Arithmetic mean of a set of observations is their sum divided by the number of observations. e.g the arithmetic mean x of n observations $\bar{x}$ of n observations $x_1, x_2, \ldots, x_n$, is given by

$\bar{x} = \frac{1}{n}(x_1 + x_2 + \ldots + x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i$

In case of frequency distribution, $x_i|f_i$, i = 1, 2, ..., n. where f_i is the frequency of the variable x_i;

$\bar{x} = \frac{x_1f_1+x_2f_2+\cdots+x_nf_n}{f_1+f_2+\cdots+f_n} = \frac{\sum_{i=1}^{n} f_ix_i}{\sum_{i=1}^{n} f_i} = \frac{1}{N}\sum_{i=1}^{n} f_ix_i$, where $\sum_{i=1}^{n} f_i = N$.

In case of grouped or continuous frequency distribution. X is taken as the mid. value of the corresponding class.

Let us understand it via some examples.

- First consider the following data: 1600, 1590, 1560, 1610, 1640, 1630. Find the arithmetic mean

$\bar{x} = \frac{1600+1590+1560+1610+1640+1630}{6} = 1605.$

- Now consider the following frequency distribution and find its arithmetic mean.

X	1	2	3	4	5	6	7
F	5	9	12	17	14	10	6

X	F	fx
1	5	5
2	9	18
3	12	36
4	17	68
5	14	70
6	10	60
7	6	42
Total	73	299

Arithmetic Mean = $\bar{x} = \frac{299}{73} = 4.09$

- Let us consider another example:

Marks (x)	0-10	10-20	20-30	30-40	40-50	50-60
No. of students (f)	12	18	27	20	17	6

Marks	No. of students (f)	Mid-Point (x)	fx
0-10	12	5	60
10-20	18	15	270
20-30	27	25	675
30-40	20	35	700
40-50	17	45	765
50-60	6	55	330
Total	100		2800

Arithmetic Mean = $\bar{x} = \frac{2800}{100}$ = 28

It may be noted that if the values of x or/and f are large, the calculation of mean is quite time consuming and tedious. The arithmetic is reduced to a great extent, by taking the deviations of the given values from any arbitrary point 'A', as explained below.

Let $d_i = x_i – A$, then $f_id_i = f_i(x_i – A) = f_ix_i - f_iA$.

Applying summation both sides, we get

$\sum_{i=1}^{n} f_i\, d_i = \sum_{i=1}^{n} f_i\, x_i$ – A $\sum_{i=1}^{n} f_i$ = $\sum_{i=1}^{n} f_i\, x_i$ – AN

$\frac{1}{N}\sum_{i=1}^{n} f_i\, d_i = \frac{1}{N}\sum_{i=1}^{n} f_i\, x_i$ – A = $\bar{x}$ - A

$\bar{x}$ = A + $\frac{1}{N}\sum_{i=1}^{n} f_i\, d_i$

This formula is easy to handle with as compared to the earlier formula.

Any number can serve the purpose of arbitrary point 'A' but. usually, the value of x corresponding to the middle part of the distribution will be much more convenient.

In case of grouped or continuous frequency distribution, the arithmetic is reduced to a still greater extent by taking

$d_i = \frac{x_i - A}{h}$, where A is an arbitrary point and h is the common magnitude of class interval. In this case, we have $hd_i=x_i$-A, and proceeding exactly similarly as above, we get

$\bar{x}$ = A + $\frac{h}{N}\sum_{i=1}^{n} f_i\, d_i$

Let us understand by an example:

- Consider the following distribution

Class Interval	0-8	8-16	16-24	24-32	32-40	40-48
Frequency	8	7	16	24	15	7

Let h = 8, A = 28

Class Interval	Mid-point (x)	Frequency (f)	D = (x – A) / h	fd
0-8	4	8	-3	-24
8-16	12	7	-2	-14
16-24	20	16	-1	-16

24-32	28	24	0	0
32-40	36	15	1	15
40-48	44	7	2	14
Total		77		-25

$\bar{x}$ = A + $\frac{h}{N}\sum_{i=1}^{n} f_i \, d_i$ = 28 + $\frac{8}{28}$ (-25) = 25.404

Properties of Arithmetic Mean:

Property 1. Algebraic sum of the deviations of a set of values from their arithmetic mean is zero. If $x_i|f_i$, i = 1, 2, ... ,n is the frequency distribution, then

$\sum_{i=1}^{n} f_i \, (x_i - \bar{x})$ = 0, $\bar{x}$ being the mean of distribution.

Property 2. The sum of the squares of the deviations of a set of values is minimum when taken about mean.

Property 3. (Mean of the composite series). If $\overline{x_i}$, (i = 1, 2, ... , k) are the means of k-component series of sizes n_i, (i = 1, 2, ... , k) respectively, then the mean $\bar{x}$ of this composite series obtained on combining the component series given by the formula:

$$\bar{x} = \frac{n_1\overline{x_1}+n_2\overline{x_2}+\cdots+n_k\overline{x_k}}{n_1+n_2+\cdots+n_k} = \frac{\sum_{i=1}^{k} n_i\overline{x_i}}{\sum_{i=1}^{k} n_i}$$

Merits and Demerits of Arithmetic Mean

Merits. (i) It is rigidly defined. (ii) It is easy to understand and easy to calculate. (iii) It is based upon all the observations. (iv) Of all the averages, arithmetic mean is affected least by fluctuations of sampling. This property is sometimes described by saying that arithmetic mean is, a stable average.

Demerits. (i) It cannot be determined by inspection nor it can be located graphically. (ii) Arithmetic mean cannot be used if we are dealing with qualitative characteristics which cannot be measured quantitate; such as, intelligence, honesty, beauty, etc. In such cases median is the only average to be used. (iii) Arithmetic mean cannot be obtained if a single observation is missing or lost or is illegible unless we drop it out and compute the arithmetic mean of the remaining values. (iv) Arithmetic mean is affected very much by extreme values. In case of extreme items, arithmetic mean gives a distorted picture of the distribution and no longer remains representative of the distribution. (v) Arithmetic mean may lead to wrong conclusions if the details of the data from which it is computed are not given. (vi) Arithmetic mean cannot be calculated if the extreme class is open. Moreover, even if a single observation is missing mean cannot be calculated. (vii) In extremely asymmetrical (skewed) distribution, usually arithmetic mean is not a suitable measure of location.

Median

Median of a distribution is the value of the variable which divides it into two equal parts. It is the va1ue which exceeds and is exceeded by the same number of observations, i.e., it is the value such that the number of observations above it is equal to the number of observations below it. The median is thus a positional average. In case of ungrouped data, if the number of observations is odd then median is the middle value after the values have been arranged in ascending or descending order of magnitude. In case of even number of observations, there are two middle terms and median is obtained by taking the arithmetic mean of the middle terms. For example, the median of the values 25, 20, 15, 35, 18, i.e., 15, 18, 20, 25, 35 is 20 and the median of 8, 20, 50, 25, 15, 30, i.e., of 8, 15, 20, 25, 30, 50 is (20 + 25)/2 = 22·5.

In case of discrete frequency distribution median is obtained by considering the cumulative frequencies. The steps for calculating median are given below:

(i) Find N/2, where N = $\sum_{i=1}^{n} f_i$.

(ii) See the (less than) cumulative frequency (cf.) just greater than N/2.

(iii) The corresponding value of x is median.

Let us understand it via an example: Obtain the median for the following frequency distribution:

X	1	2	3	4	5	6	7	8	9
F	8	10	11	16	20	25	15	9	6

X	F	c.f.
1	8	8
2	10	18
3	11	29
4	16	45
5	20	65
6	25	90
7	15	105
8	9	114
9	6	120

Here N = 120, Therefore N/2 = 60 and cumulative frequency just greater than 60 is 65. The value corresponding to 65 is 5. Hence median is 5.

In the case of continuous frequency distribution, the class corresponding to the cf. just greater than N/2 is called the median class and the value of median is obtained by the following formula:

Median = l + $\frac{h}{f}(\frac{N}{2} - c)$

where l is the lower limit of the median class,

f is the frequency of the median class,

h is the magnitude of the median class,

'c' is the c.f. of the class preceding the median class,

And N = $\sum_{i=1}^{n} f_i$.

Let us consider another example using above formula.

- Find the median wage of the following distribution:

Wages	20-30	30-40	40-50	50-60	60-70
No. of Labourer	3	5	20	10	5

Wages	No. of Labourer (f)	C.F.
20-30	3	3
30-40	5	8
40-50	20	28
50-60	10	38
60-70	5	43

N/2 = 21.5 and the value just greater than this is 28 in the c.f. column. Therefore, according the above-said formula:

Median = l + $\frac{h}{f}(\frac{N}{2} - c)$ = 40 + $\frac{10}{20}(\frac{43}{2} - 8)$ = 40 + 6.75 = 46.75.

Merits and Demerits of Median

Merits. (i) It is rigidly defined. (ii) It is easily understood and is easy to calculate. In some cases, it can be located merely by inspection. (iii) It is not at all affected by extreme values. (iv) It can be calculated for distributions with open-end classes.

Demerits. (i) In case of even number of observations median cannot be determined exactly. We merely estimate it by taking the mean of two middle terms. (ii) It is not based on all the observations. (iii) It is not amenable to algebraic treatment. (iv) As compared with mean, it is affected much by fluctuations of sampling.

Mode

In statistics, the **mode** is the value which is repeatedly occurring in a given set. We can also say that the value or number in a data set, which has a high frequency or appears more frequently is called mode or modal value.

For example, mode of the set {3, 7, 8, 8, 9}, is 8. Therefore, for a finite number of observations, we can easily find the mode. A set of values may have one mode or more than one mode or no mode at all.

Bimodal, Trimodal & Multimodal (More than one mode)

- When there are two modes in a data set, then the set is called **bimodal**

For example, the mode of Set A = {2,2,2,3,4,4,5,5,5} is 2 and 5, because both 2 and 5 is repeated three times in the given set.

- When there are three modes in a data set, then the set is called **trimodal**

For example, the mode of set A = {2,2,2,3,4,4,5,5,5,7,8,8,8} is 2, 5 and 8

- When there are four or more modes in a data set, then the set is called **multimodal**

In the case of discrete frequency distribution mode is the value of x corresponding to maximum frequency. For example, in the following frequency distribution:

X	1	2	3	4	5	6	7	8
f	4	9	16	25	22	15	7	3

The value of X corresponding to the maximum frequency, viz., 25 is 4. Hence mode is 4.

In the case of grouped frequency distribution, calculation of mode just by looking into the frequency is not possible. To determine the mode of data in such cases we calculate the modal class. Mode lies inside the modal class. The mode of data is given by the formula:

$$Mode = l + \left(\frac{f_1 - f_0}{2f_1 - f_0 - f_2}\right) \times h$$

Where,

l = lower limit of the modal class

h = size of the class interval

f_1 = frequency of the modal class

f_0 = frequency of the class preceding the modal class

f_2 = frequency of the class succeeding the modal class

Let us take an example to understand this clearly.

- Find the mode for the following distribution:

Class Interval	0-10	10-20	20-30	30-40	40-50	50-60	60-70	70-80
Frequency	5	8	7	12	28	20	10	10

Here maximum frequency is 28. Thus the class 40-50 is the modal class. The value of mode is given by

Mode = 40 + $\frac{10(28-12)}{(2*28-12-20)}$ = 40 + 6.66 = 46.66

Sometimes mode is estimated from the mean and the median. For a symmetrical distribution, mean, median and mode coincide. If the distribution is moderately asymmetrical, the mean, median and mode obey the following empirical relationship (due to Karl Pearson):

Mean - Median = $\frac{1}{3}$ (Mean - Mode)

Mode = 3 Median - 2 Mean

Merits and Demerits or Mode

Merits. (i) Mode is readily comprehensible and easy to calculate. Like median, mode can be located in some cases merely by inspection. (ii) Mode is not at all affected by extreme values. (iii) Mode can be conveniently located even if the frequency distribution has class-intervals of unequal magnitude provided the modal class and the classes preceding and succeeding it are of the same magnitude. Open end classes also do not pose any problem in the location of mode.

Demerits. (i) Mode is ill-defined. It is not always possible to find a clearly defined mode. (ii) It is not based upon all the observations. (iii) It is not capable of further mathematical treatment. (iv) As compared with mean, mode is affected to a greater extent by fluctuations of sampling.

Measures of Central Tendencies using Python:

We can find mean of given data using following function:

mean([data-set])

Parameters :[data-set] : List or tuple of a set of numbers.

Returns: Simple arithmetic mean of the provided data-set.

```
# Program 3.1: Python program to demonstrate mean() function from the statistics module

# Importing the statistics module

import statistics

# list of positive integer numbers

data1 = [1, 3, 4, 5, 7, 9, 2]

x = statistics.mean(data1)

# Printing the mean

print("Mean is :", x)


Output:

Mean is : 4.428571428571429
```

We can find median of given data using following function:

median([data-set])

Parameters: [data-set]: List or tuple or an iterable with a set of numeric values

Returns: Return the median (middle value) of the iterable containing the data

```
# Program 3.2: Python code to demonstrate the working of median() function.

# importing statistics module

import statistics

# unsorted list of random integers

data1 = [2, -2, 3, 6, 9, 4, 5, -1]
```

```
# Printing median of the
# random data-set
print("Median of data-set is : % s " % (statistics.median(data1)))


Output:
Median of data-set is : 3.5
```

We can find mode of given data using following function:

mode([data-set])

Parameters : [data-set] *which is a tuple, list or a iterator of real valued numbers as well as Strings.*

Return type : *Returns the most-common data point from discrete or nominal data.*

```
# Program 3.3: Python code to demonstrate the use of mode() function
import statistics


# declaring a simple data-set consisting of real valued positive integers.
set1 =[1, 2, 3, 3, 4, 4, 4, 5, 5, 6]
# In the given data-set we can infer that 4 has the highest population distribution
# So mode of set1 is 4
# Printing out mode of given data-set
print("Mode of given data set is % s" % (statistics.mode(set1)))


Output:
Mode of given data set is 4
```

3.2.2 Measures of Dispersion

Literal meaning of dispersion is scatteredness. We study dispersion to have an idea about the homogeneity or heterogeneity of the distribution. The measures of central tendency serve to locate the center of the distribution, but they do not reveal how the items are spread out on either side of the center. This characteristic of a frequency distribution is commonly referred to as dispersion. In a series all the items are not equal. There is difference or variation among the values. The degree of variation is evaluated by various measures of dispersion. Small dispersion indicates the high uniformity of the items, while large dispersion indicates less uniformity.

Characteristics of a good measure of Dispersion

An ideal measure of dispersion is expected to possess the following properties

- It should be rigidly defined
- It should be based on all the items.
- It should not be unduly affected by extreme items.
- It should lend itself for algebraic manipulation.
- It should be simple to understand and easy to calculate.

The following are the measures of dispersion:

(i) Range, (ii) Mean deviation, (iii) Standard deviation, (iv) Variance, and (v) Coefficient of Variation.

Range

The range is the difference between two extreme observations, or the distribution. If A and B are the greatest and smallest observations respectively in a distribution, then its range is A-B. Range is the simplest but a crude measure of dispersion. Since it is based on two extreme observations which themselves are subject to chance fluctuations, it is not at all a reliable measure of dispersion.

In individual observations and discrete series, A and B are easily identified. In continuous series, the following method is followed. A = Upper boundary of the highest class B = Lower boundary of the lowest class.

Find the value of range for the following data: 8,10, 5,9,12,11

A = 12, B = 5

Range = A – B = 7

Calculate the range from the following distribution:

Class Interval	0-10	10-20	20-30	30-40	40-50	50-60	60-70	70-80
Frequency	5	8	7	12	28	20	10	10

Here the range = 80 – 0 = 80

Merits: 1. It is simple to understand. 2. It is easy to calculate. 3. In certain types of problems like quality control, weather forecasts, share price analysis, etc., range is most widely used.

Demerits: 1. It is very much affected by the extreme items. 2. It is based on only two extreme observations. 3. It cannot be calculated from open-end class intervals. 4. It is not suitable for mathematical treatment. 5. It is a very rarely used measure.

Mean Deviation

If $x_i|f_i$, i = 1, 2, …, n is the frequency distribution, then mean deviation from the average A, (either mean, median or mode), is given by

Mean deviation = $\frac{1}{N}\sum_i f_i|x_i - A|$, $\sum_i f_i = N$

where |Xi – A| represents the modulus or the absolute value of the deviation (Xi - A).

Since mean deviation is based on all the observations. it is a better measure of dispersion than range. But the step of ignoring the signs of the deviations |Xi – A| creates artificiality and, renders it useless for further mathematical treatment. It may be pointed out here that mean deviation is least when taken from median.

Let us consider few examples:

- Calculate Mean Deviation about Mean for the numbers given below: 1,2,3,4,5.

Here, Mean = $\bar{x} = \frac{\sum_i x_i}{n} = \frac{15}{5} = 3$

x	|x - $\bar{x}$|
1	2
2	1
3	0
4	1
5	2
$\sum_i x_i$=15	$\sum_i$ |x $-$ $\bar{x}$| = 6

M.D. about Mean = $\frac{\sum_i |x - \bar{x}|}{N} = \frac{6}{5} = 1.2$

Consider another examples based of discrete frequency distribution.

- Calculate the M.D. from Mean for the following data:

X	F	Fx	|x $-$ $\bar{x}$| = |x $-$ 6|	f|x $-$ $\bar{x}$|
2	1	2	4	4
4	4	16	2	8
6	6	36	0	0
8	4	32	2	8
10	1	10	4	4
Total	16	96	14	24

Mean = $\bar{x} = \frac{\sum_i f_i x_i}{\sum_i f_i} = \frac{96}{16} = 6$

M.D. about Mean = $\frac{\sum_i f_i |x - \bar{x}|}{\sum_i f_i} = \frac{24}{16} = 1.5$

- Consider another examples based of continuous frequency distribution.

Marks	No. of Students(f)	Middle-Point(x)	fx	|x $-$ $\bar{x}$| = |x $-$ 33.4|	f|x $-$ $\bar{x}$|
0-10	6	5	30	28.4	170.4
10-20	5	15	75	18.4	92
20-30	8	25	200	8.4	67.2
30-40	15	35	525	1.6	24
40-50	7	45	315	11.6	81.2
50-60	6	55	330	21.6	129.6
60-70	3	65	195	31.6	94.8
Total	50		1670		659.2

Mean = $\overline{x} = \frac{\sum_i f_i x_i}{\sum_i f_i} = \frac{1670}{50}$ = 33.4

M. D. about Mean = $\frac{\sum_i f_i |\mathrm{x} - \overline{x}\,|}{\sum_i f_i} = \frac{659.2}{50}$ = 13.18

Merits: 1. It is simple to understand and easy to compute. 2. It is rigidly defined. 3. It is based on all items of the series. 4. It is not much affected by the fluctuations of sampling. 5. It is less affected by the extreme items. 6. It is flexible, because it can be calculated from any average. 7. It is a better measure of comparison.

Demerits: 1. It is not a very accurate measure of dispersion. 2. It is not suitable for further mathematical calculation. 3. It is rarely used. It is not as popular as standard deviation. 4. Algebraic positive and negative signs are ignored. It is mathematically unsound and illogical.

Mean Deviation using Python:

Absolute mean deviation can be calculated in python using following code:

Using Numpy

```
# Program 3.4: Mean Deviation using Numpy
# Importing mean, absolute from numpy
data = [75, 69, 56, 46, 47, 79, 92, 97, 89, 88, 36, 96, 105, 32, 116, 101, 79, 93, 91, 112]
# Absolute mean deviation
mean(absolute(data - mean(data)))
Output:
20.055
```

Using Pandas

```
#Program 3.5: Mean Deviation using Pandas
# Import the pandas library as pd
import pandas as pd
data = [75, 69, 56, 46, 47, 79, 92, 97, 89, 88, 36, 96, 105, 32, 116, 101, 79, 93, 91, 112]
# Creating data frame of the given data
df = pd.DataFrame(data)
# Absolute mean deviation
df.mad() # mad() is mean absolute deviation function
Output:
20.055
```

Standard Deviation and Variance

Standard deviation, usually denoted by the sigma (σ), is the positive square root of the arithmetic mean of the squares of the deviations of the given values from their arithmetic mean. For the frequency distribution $x_i|f_i$, i = 1,2, ... , n.

$\sigma = \sqrt{\frac{1}{N}\sum_i f_i(x_i - \bar{x})^2}$, where $\bar{x}$ is the arithmetic mean of the distribution and $\sum_i f_i$ = N.

The step of squaring the deviations $(x_i - \bar{x})$ overcomes the drawback of ignoring the signs in mean deviation. Standard deviation is also suitable for further mathematical treatment. Moreover, of all the measures, standard deviation is affected least by fluctuations of sampling.

The square of standard deviation is called the variance and is given by

$$\sigma^2 = \frac{1}{N}\sum_i f_i(x_i - \bar{x})^2$$

For individual series, standard deviation and variance can be calculated as follows:

$$\sigma = \sqrt{\frac{\sum x^2}{n} - (\frac{\sum x}{n})^2}, \qquad \sigma^2 = \frac{\sum x^2}{n} - (\frac{\sum x}{n})^2$$

Different formulae for calculating variance:

Now we know that variance for any frequency distribution may be written as:

$$\sigma^2 = \frac{1}{N}\sum_i f_i(x_i - \bar{x})^2$$

However, it can better be written as:

$$\sigma_x^2 = \frac{1}{N}\sum_i f_i(x_i - \bar{x})^2$$

If $\bar{x}$ is not a whole number, the calculation of σ_x^2 is very cumbersome and time consuming. Therefore, formula can be changed to

$$\sigma_x^2 = \frac{1}{N}\sum_i f_i x_i^2 - (\frac{1}{N}\sum_i f_i x_i)^2$$

If the values of x and f are large, the calculation of fx, fx^2 is quite tedious. In that case we take the deviations from any arbitrary point 'A '. Generally, the point in the middle of the distribution is much convenient and therefore we have

$\sigma_x^2 = \frac{1}{N}\sum_i f_i d_i^2 - (\frac{1}{N}\sum_i f_i d_i)^2$, where $d_i = x_i - A$

We can make the calculations easier by using

$\sigma_x^2 = h^2[\frac{1}{N}\sum_i f_i d_i^2 - (\frac{1}{N}\sum_i f_i d_i)^2]$, where $d_i = \frac{xi - A}{h}$

Now let us take some examples to understand the standard deviation and variance.

- Calculate S.D. for the data given below.

Sr. No.	Marks (X)	X^2
1	5	25
2	10	100
3	20	400
4	25	625
5	40	1600
6	42	1764
7	45	2025
8	48	2304
9	70	4900
10	80	6400
Total	385	20143

Mean = $\overline{x}$ = $\frac{385}{10}$ = 38.5, $\sigma = \sqrt{\frac{\sum x^2}{n} - (\frac{\sum x}{n})^2} = \sqrt{\frac{20143}{10} - 38.5^2} = 23.07$

- Calculate S.D. for the following data.

X	F	Fx	X^2	Fx^2
6	7	42	36	252
9	12	108	81	972
12	13	156	144	1872
15	10	150	225	2250
18	8	144	324	2592
Total	50	600		7938

$\sigma = \sqrt{\frac{\sum fx^2}{\sum f} - [\frac{\sum fx}{\sum f}]^2} = \sqrt{\frac{7938}{50} - 12^2} = 3.84$

- Calculate the mean and standard deviation for tile following table giving tile age distribution of 542 members.

Age	20-30	30-40	40-50	50-60	60-70	70-80	80-90
No. of Members	3	61	132	153	140	51	2

In this data let us take d = $\frac{x-A}{h} = \frac{x-55}{10}$

Age	Mid-Value(x)	Members(f)	D=$\frac{x-55}{10}$	Fd	Fd^2
20-30	25	3	-3	-9	27
30-40	35	61	-2	-132	244
40-50	45	132	-1	-132	132
50-60	55	153	0	0	0
60-70	65	140	1	140	140
70-80	75	51	2	102	204
80-90	85	2	3	6	18
Total		542		-15	765

$\bar{x} = \text{A} + \text{h}\frac{\sum fd}{N} = 55 + 10(\frac{-15}{542}) = 54.72$

$\sigma_x^2 = \text{h}^2[\frac{1}{N}\sum_i f_i d_i^2 - (\frac{1}{N}\sum_i f_i d_i)^2] = 100[\frac{765}{542} - \left(\frac{-15}{542}\right)^2] = 133.3$

σ = 11.55 years.

Standard Deviation and Variance in Python:

It can be calculated in python using many ways. However, we will be considering only two ways i.e. using Numpy and Statistics packages.

Using Numpy:

One can calculate the standard deviation by using **numpy.std()** function in python.

numpy.std(a, axis=None, dtype=None, out=None, ddof=0, keepdims=<no value>)

Parameters:

a: Array containing data to be averaged

axis: Axis or axes along which to average a

dtype: Type to use in computing the variance.

out: Alternate output array in which to place the result.

ddof: Delta Degrees of Freedom

keepdims: If this is set to True, the axes which are reduced are left in the result as dimensions with size one

```
# Program 3.6: Python program to get standard deviation of a list

# Importing the NumPy module

import numpy as np

# Taking a list of elements

list = [2, 4, 4, 4, 5, 5, 7, 9]

# Calculating standard deviation using std()

print(np.std(list))

Output:

2.0
```

One can calculate the variance by using **numpy.var()** function in python.

numpy.var(a, axis=None, dtype=None, out=None, ddof=0, keepdims=<no value>)

Parameters:

a: Array containing data to be averaged

axis: Axis or axes along which to average a *dtype: Type to use in computing the variance.*

out: Alternate output array in which to place the result.

ddof: Delta Degrees of Freedom

keepdims: If this is set to True, the axes which are reduced are left in the result as dimensions with size one

```
# Program 3.7: Python program to get variance of a list

# Importing the NumPy module

import numpy as np

# Taking a list of elements

list = [2, 4, 4, 4, 5, 5, 7, 9]

# Calculating variance using var()

print(np.var(list))

Output:

4.0
```

Using Statistics:

variance([data], xbar)

Parameters :
[data] : An iterable with real valued numbers.
xbar (Optional) : Takes actual mean of data-set as value.

Returnype : Returns the actual variance of the values passed as parameter.

```
# Program 3.8: Python code to demonstrate the working of variance() function of Statistics

# Importing Statistics module

import statistics

# Creating a sample of data

sample = [2.74, 1.23, 2.63, 2.22, 3, 1.98]

# Prints variance of the sample set

# Function will automatically calculate it's mean and set it as xbar

print("Variance of sample set is % s" %(statistics.variance(sample)))


Output:

Variance of sample set is 0.40924
```

Another example of coding for finding the variance is:

```
# Program 3.9: Python code to demonstrate the use of xbar parameter for variance
# Importing statistics module
import statistics
# creating a sample list
sample = (1, 1.3, 1.2, 1.9, 2.5, 2.2)
# calculating the mean of sample set
m = statistics.mean(sample)
# calculating the variance of sample set
print("Variance of Sample set is % s" %(statistics.variance(sample, xbar = m)))


Output:
Variance of Sample set is 0.3656666666666667
```

Standard Deviation in Python using Statistics package can be coded as:

stdev([data-set], xbar)

Parameters :

[data] : An iterable with real valued numbers.

xbar (Optional): Takes actual mean of data-set as value.

Returnype : Returns the actual standard deviation of the values passed as parameter.

```
# Program 3.10: Python code to demonstrate stdev() function
# importing Statistics module
import statistics
# creating a simple data - set
sample = [1, 2, 3, 4, 5]
# Prints standard deviation
# xbar is set to default value of 1
print("Standard Deviation of sample is % s " % (statistics.stdev(sample)))


Output:
Standard Deviation of the sample is 1.5811388300841898
```

Another example of coding for finding the standard deviation is:

```
# Program 3.11: Python code to demonstrate use of xbar parameter while using stdev() function

# Importing statistics module

import statistics

# creating a sample list

sample = (1, 1.3, 1.2, 1.9, 2.5, 2.2)

# calculating the mean of sample set

m = statistics.mean(sample)

# xbar is nothing but stores the mean of the sample set

# calculating the standard deviation of sample set

print("Standard Deviation of Sample set is % s" %(statistics.stdev(sample, xbar = m)))


Output:

Standard Deviation of Sample set is 0.6047037842337906
```

Coefficient of Variation

Coefficient of variation can be found using the following formula:

Coefficient of Variation (C.V.) = $\frac{Standard\ Deviation}{Arithmetic\ Mean} * 100 = \frac{\sigma}{\bar{x}} * 100.$

Let us consider an example for this:

- The means and standard deviation values for the number of runs of two players A and B are 55; 65 and 4.2; 7.8 respectively. Who is the more consistent player?

Coefficient of variation of Player A = $\frac{\sigma}{\bar{x}} * 100 = \frac{4.2}{55} * 100 = 7.64$

Coefficient of variation of Player B = $\frac{\sigma}{\bar{x}} * 100 = \frac{7.8}{65} * 100 = 12$

Coefficient of variation of player A is less. Therefore, Player A is the more consistent player.

Coefficient of variation can be calculated using python as follows:

scipy.stats.variation(arr, axis = None) *function computes the coefficient of variation. It is defined as the ratio of standard deviation to mean.*

Parameters:

arr: *[array_like] input array.*

axis: *[int or tuples of int] axis along which we want to calculate the coefficient of variation.*

-> *axis = 0 coefficient of variation along the column.*

-> *axis = 1 coefficient of variation working along the row.*

Results: *Coefficient of variation of the array with values along specified axis.*

```
# Program 3.12: Coefficient of Variation
from scipy.stats import variation
import numpy as np
arr = np.random.randn(5, 5)
print ("array : \n", arr)
# rows: axis = 0, cols: axis = 1
print ("\nVariation at axis = 0: \n", variation(arr, axis = 0))
print ("\nVariation at axis = 1: \n", variation(arr, axis = 1))



Output:
array :
 [[-1.16536706 -1.29744691 -0.39964651  2.14909277 -1.00669835]
 [ 0.79979681  0.91566149 -0.823054    0.9189682  -0.01061181]
 [ 0.9532622   0.38630077 -0.79026789 -0.70154086  0.79087801]
 [ 0.53553389  1.46409899  1.89903817 -0.35360202 -0.14597738]
 [-1.53582875 -0.50077039 -0.23073327  0.32457064 -0.43269088]]

Variation at axis = 0:
 [-12.73042404   5.10272979 -14.6476392    2.15882202  -3.64031032]

Variation at axis = 1:
 [-3.73200773  1.90419038  5.77300406  1.29451485 -1.27228112]
```

3.2.3 Skewness

Literally skewness means 'lack of symmetry'. We study skewness to have an idea about the shape of the curve which we can draw with the help of the given data. A distribution is said to be skewed if

(i) Mean, median and mode fall at different points. i.e., Mean ≠ Median ≠ Mode

(ii) The curve drawn with the help of the given data is not symmetrical but stretched more to one side than to the other.

Measures of Skewness: Various measures of skewness are

- $S_k = M - M_d$
- $S_k = M - M_o$,

where M is the mean, M_d the median and M_o is the mode of the distribution.

These are the absolute measures of skewness. As in dispersion, for comparing two series we do not calculate these absolute measures but we calculate the relative measures called the co-efficient of skewness which are pure numbers independent of units of measurement. The following is the coefficients of Skewness.

Prof. Karl Pearson's Coefficient of Skewness (S_k) = $\frac{(M-M_0)}{\sigma}$, where σ is the standard deviation of the distribution.

If mode is ill-defined, then using the relation, $M_o = 3M_d - 2M$, for a moderately asymmetrical distribution, we get

$S_k = \frac{3(M-M_d)}{\sigma}$

The limits for Karl Pearson's coefficient of skewness are ± 3. In practice, these limits are rarely attained. Skewness is positive if $M > M_o$ or $M > M_d$ and negative if $M < M_o$ or $M < M_d$.

Skewness can be represented graphically as shown in figure 3.1:

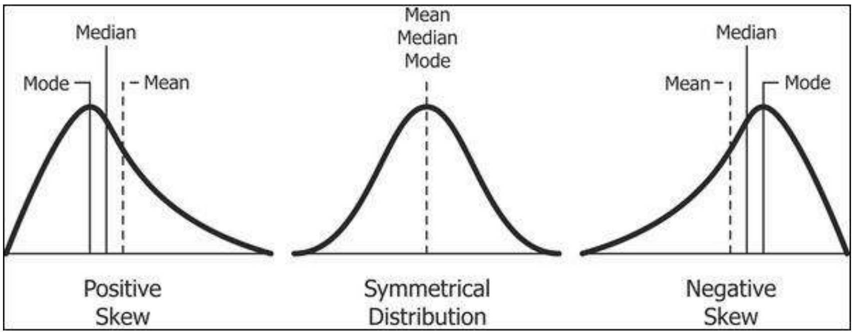


Figure 3.1: Skewness

Skewness can be calculated as per following examples:

- From the marks secured by 120 students in Section A and B of a class, the Following measures are obtained:

Section A: $\bar{X}$ = 46.83; S.D = 14.8; Mode = 51.67

Section B: $\bar{X}$ = 47.83; S.D = 14.8; Mode = 47.07

Determine which distribution of marks is more skewed.

Karl Pearson's Coefficient of Skewness for Section A = $S_k = \frac{(M-M_0)}{\sigma} = \frac{(46.83-51.67)}{14.8}$ = -0.3270

Karl Pearson's Coefficient of Skewness for Section B = $S_k = \frac{(M-M_0)}{\sigma} = \frac{(47.83-47.07)}{14.8} = 0.05135$

Marks of Section A is more Skewed. But marks of Section A are negatively Skewed. Marks of Section B are Positively skewed.

- From a moderately skewed distribution of retail prices for men’s shoes, it is found that the mean price is Rs. 20 and the median price is Rs. 17. If the coefficient of variation is 20%, find the Pearsonian coefficient of skewness of the distribution.

Coefficient of Variation (C.V.) = $\frac{Standard\ Deviation}{Arithmetic\ Mean} * 100 = \frac{\sigma}{\bar{x}} * 100$

$20 = \frac{\sigma}{20} * 100 \implies \sigma = 4$

$S_k = \frac{3(M-M_d)}{\sigma} = \frac{3(20-17)}{4} = 2.25.$

Skewness may be calculated using python as follows:

scipy.stats.skew(array, axis=0, bias=True) *function calculates the skewness of the data set.*

Parameters :
array : *Input array or object having the elements.*
axis : *Axis along which the skewness value is to be measured. By default axis = 0.*
bias : *Bool; calculations are corrected for statistical bias, if set to False.*

Returns: *Skewness value of the data set, along the axis.*

```
#Program 3.13: finding Skewness

from scipy.stats import skew

import numpy as np

# random values based on a normal distribution

x = np.random.normal(0, 2, 10000)

print ("X : \n", x)

print('\nSkewness for data : ', skew(x))


Output:

X :

 [ 0.03255323 -6.18574775 -0.58430139 ...  3.22112446  1.16543279 0.84083317]

Skewness for data:  0.03248837584866293
```

3.2.4 Kurtosis

If we know the measures of central tendency, dispersion and skewness, we still cannot form a complete idea about the distribution as will be clear from the figure 3.2 in which all the three curves A, B and C are symmetrical about the mean and have the same range.

In addition to these measures we should know one more measure which Prof. Karl Pearson calls as the Convexity of curve or Kurtosis. Kurtosis enables us to have an idea about the flatness -or peakedness of the curve, It is measured by the co-efficient β_2 or its derivation γ_2 given by

$\beta_2 = \frac{\mu_4}{\mu_2^2}$, $\gamma_2 = \beta_2 - 3$

Where, $\mu_2, and\ \mu_4$ are moments about mean and can be calculated using following formulae:

The rth moment of a variable about the mean $\bar{X}$, usually denoted by μ_r is given by:

$\mu_r = \frac{1}{N}\sum_i f_i(x_i - \bar{x})^r$, In particular $\mu_0 = \frac{1}{N}\sum_i f_i(x_i - \bar{x})^0 = \frac{\sum_i f_i}{N} = 1$.

$\mu_1 = \frac{1}{N}\sum_i f_i(x_i - \bar{x})^1 = 0$, being the algebraic sum of deviations from the mean.

Also, $\mu_2 = \frac{1}{N}\sum_i f_i(x_i - \bar{x})^2 = \sigma^2$

The rth moment of a variable x about any arbitrary point x = A, usually denoted by μ_r' is given by:

$\mu_r' = \frac{1}{N}\sum_i f_i(x_i - A)^r = \frac{1}{N}\sum_i f_i\, d_i$

Where $d_i = x_i - A$.

The relations between moments about an arbitrary point and about mean is represented as:

$\mu_2 = \mu_2' - \mu_1'^2$,

$\mu_3 = \mu_3' - 3\mu_2'\mu_1' + 2\mu_1'^3$,

$\mu_4 = \mu_4' - 4\mu_3'\mu_1' + 6\mu_2'\mu_1'^2 - 3\mu_1'^4$.

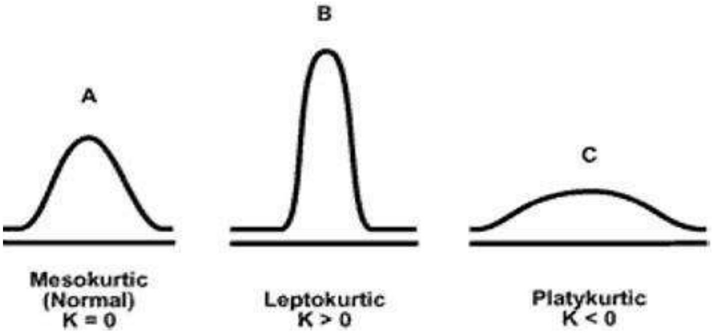


Figure 3.2: Kurtosis

Curve of the type 'A' which is neither flat nor peaked is called the normal curve or mesokurtic curve and for such a curve β_2 = 3, i.e., γ_2 = O. Curve of the type 'C' which is flatter than the normal curve is known as platykurtic curve and for such a curve β_2 < 3, i.e., γ_2 < O. Curve of the type 'B' which is more peaked than the normal curve is called leptokurtic and for such a curve, β_2 > 3 i.e.; γ_2 > 0.

Let us understand the concept by using an example.

- The data on daily wages of 45 workers of a factory are given. Compute skewness and kurtosis using moment about the mean. Comment on the results.

Wages	100-120	120-140	140-160	160-180	180-200	200-220	220-240
No. of Workers	1	2	6	20	11	3	2

Solution:

Wages	No. of Workers(f)	Mid-Point(x)	D=$\frac{x-170}{20}$	Fd	Fd^2	Fd^3	Fd^4
100-120	1	110	-3	-3	9	-27	81
120-140	2	130	-2	-4	8	-16	32
140-160	6	150	-1	-6	6	-6	6
160-180	20	170	0	0	0	0	0
180-200	11	190	1	11	11	11	11
200-220	3	210	2	6	12	24	48
220-240	2	230	3	6	18	54	162
Total	45			10	64	40	330

$\mu_1' = \frac{\sum fd}{N} * h = \frac{10}{45} * 20 = 4.44$

$\mu_2' = \frac{\sum fd^2}{N} * h^2 = \frac{64}{45} * 20^2 = 568.88$

$\mu_3' = \frac{\sum fd^3}{N} * h^3 = \frac{40}{45} * 20^3 = 7111.11$

$\mu_4' = \frac{\sum fd^4}{N} * h^4 = \frac{330}{45} * 20^4 = 1173333.33$

Moments about mean are:

$\mu_2 = \mu_2' - \mu_1'^2 = 549.16$

$\mu_3 = \mu_3' - 3\mu_2'\mu_1' + 2\mu_1'^3 = -291.32$

$\mu_4 = \mu_4' - 4\mu_3'\mu_1' + 6\mu_2'\mu_1'^2 - 3\mu_1'^4 = 1113162.18$

Therefore,

Skewness = $\beta_1 = \mu_3^2 = 0.00051$

Kurtosis = $\beta_2 = \frac{\mu_4}{\mu_2^2} = 3.69$

From the above calculations, it can be concluded that skewness is almost zero, thereby indicating that the distribution is almost symmetrical. Kurtosis, has a value greater than 3, thus implying that the distribution is leptokurtic.

Kurtosis in python can be calculated as follows:

scipy.stats.kurtosis(array, axis=0, fisher=True, bias=True) *function calculates the kurtosis (Fisher or Pearson) of a data set.*

Parameters:

array: *Input array or object having the elements.*

axis: *Axis along which the kurtosis value is to be measured. By default, axis = 0.*

fisher: *Bool; Fisher's definition is used (normal 0.0) if True; else Pearson's definition is used (normal 3.0) if set to False.*

bias: *Bool; calculations are corrected for statistical bias, if set to False.*

Returns: *Kurtosis value of the normal distribution for the data set.*

3.3 SUMMARY

In this module, various measures of central tendency viz., mean, median and mode have been discussed. Various methods for calculating mean, median and mode have been discussed for grouped, ungrouped, discrete and continuous data. The measure of dispersion such as range, mean deviation, standard deviation, variance and coefficient of variation have been discussed in detail. Different techniques for evaluating these measures have also been elaborated. Following this, skewness and kurtosis along with their measures have also been inspected. All these types of measures have been implemented in python using numpy, pandas, matplotlib and statistics packages.

3.4 PRACTICE QUESTION

Q.1 Compare mean, median and mode as measures of location of a distribution.

Q.2 Describe the different measures of central tendency of a frequency distribution, mentioning their merits and demerits.

Q.3 Given below is the distribution of 140 candidates obtaining marks X or higher in it certain examination (all marks are given in whole numbers):

X	10	20	30	40	50	60	70	80	90	100
c.f.	140	133	118	100	75	45	25	9	2	0

Calculate the mean, median and mode of the distribution.

Q.4 The mean of marks obtained in an examination by a group of 100 students was found to be 49·96. The mean of the marks obtained in the same examination by another group of 200 students was 52·32. Find the mean of the marks obtained by both the groups of students taken together.

Q.5 Which measure of location will be suitable to compare: (i) heights of students in two classes; (ii) size of agricultural holdings; (iii) average sales for various years; (iv) intelligence of students; (v) per capita income in several countries.

Q.6 Explain with suitable examples the term dispersion. State the relative and absolute measures of dispersion and describe the merits and demerits of these.

Q.7 Explain the main difference between mean deviation and standard deviation.

Q.8 Age distribution of hundred life insurance policy holders is as follows:

Age	17-19.5	20-25.5	26-35.5	36-40.5	41-50.5	51-55.5	56-60.5	61-70.5

Number	9	16	12	26	14	12	6	5

Calculate mean deviation from median age.

Q.9 What is standard deviation? Explain its superiority over other measures of dispersion.

Q.10 Calculate the mean and standard deviation of the following distribution:

X	2.5-7.5	7.5-12.5	12.5-17.5	17.5-22.5	22.5-27.5	27.5-32.5	32.5-37.5
f	65	121	175	198	176	120	66

Q. 11 The mean of 5 observations is 4·4 and variance is 8·24. It three of the five observation are 1, 2 and 6; find the other two.

Q.12 Lives of two models of refrigerators turned in for new models in a recent survey are:

Life	Model A	Model B
0-2	5	2
2-4	16	7
4-6	13	12
6-8	7	19
8-10	5	9
10-12	4	1

What is the average life of each model of these refrigerators? Which model shows more uniformity?

Q. 13 What do you understand by skewness? How is it measured? Distinguish clearly, by giving figures, between positive and negative skewness.

Q.14 Explain the methods of measuring skewness and kurtosis of a frequency distribution.

Q.15 Obtain Karl Pearson's measure of skewness and kurtosis for the following data:

Values	5-10	10-15	15-20	20-25	25-30	30-35	35-40
Frequency	6	8	17	21	15	11	2

REFERENCES

- A. Abebe, J. Daniels, J.W.Mckean, "Statistics and Data Analysis".
- A. Martelli, A. Ravenscroft, S. Holden, "Python in a Nutshell", OREILLY.
- Clarke, G.M. & Cooke, D., "A Basic course in Statistics", Arnold.
- David M. Lane, "Introduction to Statistics".
- Eric Matthes, "Python Crash Course: A Hands-On, Project-Based Introduction to Programming".
- S.C.Gupta and V.K.Kapoor, "Fundamentals of Mathematical Statistics", Sultan Chand & Sons, New Delhi.

B.Sc.(DATA SCIENCE)

SEMESTER-II

STATISTICAL FOUNDATION

UNIT IV: DESCRIPTIVE STATISTICS

STRUCTURE

4.0 OBJECTIVES

In this module, we will try to understand about the descriptive statistics along with its importance and limitations followed by its implementation on python. Then a detailed discussion about exploratory data analysis will be undertaken. Various tools used in exploratory data analysis will also be explored. The implementation of these tools in python will also be deliberated upon. The concept of data visualization will also be taken up in the last part. The various types of data visualization methods will be considered along with their implementation in python.

4.1 INTRODUCTION

This module is designed to know about the overall presentation of data both in a textual manner as well as graphical manner. Descriptive statistics is used to describe the data as a whole, which mean the various measures of data are evaluated using descriptive statistics. It provides an overall description about the data and that’s why known descriptive statistics. The exploratory data analysis presents the various tools used for interpreting and analysing the data. Various types of graphical methods are used in this type of analysis. Further, data visualization is a concept, which is backbone of data analysis. If we are able to analyse data in a good manner, but not able to present it in an effective manner, it will be no or little use. That’s where comes the need of data visualization. Data visualization provides an effective way of presenting our data.

4.2 MAIN CONTENT

4.2.1 Descriptive Statistics

Descriptive statistics summarize and organize characteristics of a data set. A data set is a collection of responses or observations from a sample or entire population. In quantitative research, after collecting data, the first step of statistical analysis is to describe characteristics of the responses, such as the average of one variable (e.g., age), or the relation between two variables (e.g., age and creativity).

Purpose of Descriptive Statistics

The essential features of the data of a study are defined using descriptive statistics. The sample and measurements are summarized. They form the basis of practically any quantitative analysis of the data in combination with simple graphic analysis. Also, a data set can be summarised and described using descriptive statistics through a variety of tabulated and graphical explanations and discussion of the observed results. Complex quantitative data are summed up in descriptive statistics. Descriptive statistics can be helpful to

- providing essential data on variables in a dataset
- highlighting possible relationships between variables.

Types of descriptive statistics

There are 3 main types of descriptive statistics:

- The distribution concerns the frequency of each value.
- The central tendency concerns the averages of the values.

- The variability or dispersion concerns how spread out the values are.

We have already discussed all three types of descriptive statistics in the previous units.

We can apply these to assess only one variable at a time, in univariate analysis, or to compare two or more, in bivariate and multivariate analysis.

Univariate descriptive statistics: Univariate descriptive statistics focus on only one variable at a time. It's important to examine data from each variable separately using multiple measures of distribution, central tendency and spread.

Bivariate and Multivariate descriptive statistics: If we've collected data on more than one variable, we can use bivariate or multivariate descriptive statistics to explore whether there are relationships between them. In bivariate analysis, you simultaneously study the frequency and variability of two variables to see if they vary together. You can also compare the central tendency of the two variables before performing further statistical tests. Similarly, multivariate analysis can be applied for more than two variables.

Importance of Descriptive Statistics

Descriptive statistics allow for the ease of data visualization. It allows for data to be presented in a meaningful and understandable way, which, in turn, allows for a simplified interpretation of the data set in question. Raw data would be difficult to analyse, and trend and pattern determination may be challenging to perform. In addition, raw data makes it challenging to visualize what the data is showing.

Limitations of Descriptive Statistics

Descriptive statistics are so small that only the individuals or items you have calculated are summed up. The data you have obtained cannot be used to generalize to others or objects. When testing a drug to beat cancer in your patients, for example, you cannot say it would operate only based on descriptive statistics in other cancer patients.

Examples for Descriptive Statistics

- Take a simple number to sum up how well a batter does in baseball, the average batting. This figure is just the number of hits divided by the number of times at bat.
- You want to study the popularity of different leisure activities by gender. You distribute a survey and ask participants how many times they did each of the following in the past year: Go to a library, Watch a movie at a theatre, Visit a national park. Your data set is the collection of responses to the survey. Now you can use descriptive statistics to find out the overall frequency of each activity (distribution), the averages for each activity (central tendency), and the spread of responses for each activity (variability).
- There are 100 students enrolled for a particular module. To find the overall performance of the students taking the respective module and the distribution of the marks, descriptive statistics must be used. Getting the marks as raw data would prove the determination of the overall performance and the distribution of the marks to be challenging.

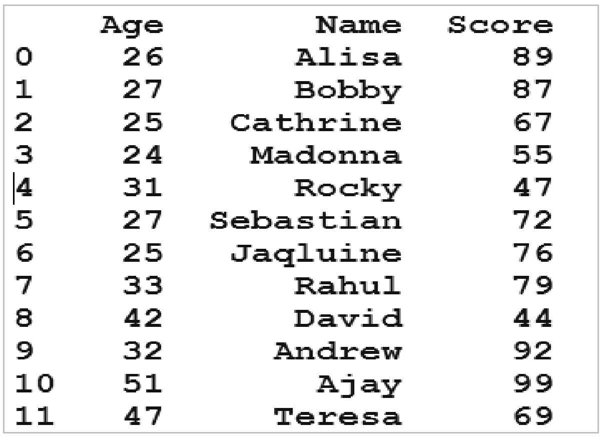
Descriptive Statistics in Python

It can be done in python using Pandas Describe() function. Describe Function gives the mean, std and IQR values. Generally describe() function excludes the character columns and gives summary statistics of numeric columns. We need to add a variable named include='all' to get the summary statistics or descriptive statistics of both numeric and character column.

Let's see with an example

```
# Program 4.1: Creation of Data Frame

Import pandas as pd

Import numpy as np

# Create a Dictionary of Series

d = {'Name':pd.Series(['Alisa', 'Bobby', 'Cathrine', 'Madonna', 'Rocky', 'Sebastian', 'Jaqluine', ‘Rahul', 'David', 'Andrew', 'Ajay', 'Teresa']), 'Age':pd.Series([26, 27, 25, 24, 31, 27, 25, 33, 42, 32, 51, 47]), 'Score':pd.Series([89,87,67,55,47,72,76,79,44,92,99,69])}

# Create a Data Frame

df = pd.DataFrame(d)

print df
```

Output:



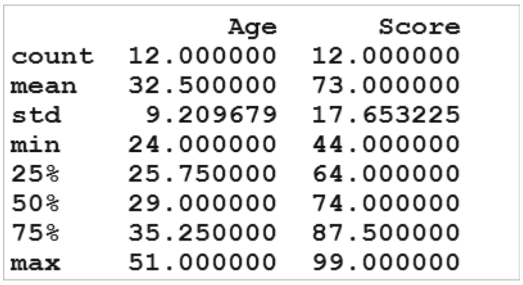
```
    Age       Name  Score
0    26      Alisa     89
1    27      Bobby     87
2    25   Cathrine     67
3    24    Madonna     55
4    31      Rocky     47
5    27  Sebastian     72
6    25   Jaqluine     76
7    33      Rahul     79
8    42      David     44
9    32     Andrew     92
10   51       Ajay     99
11   47     Teresa     69
```

Describe() Function gives the mean, std and IQR values. It excludes character column and calculate summary statistics only for numeric columns.

If we write the following statement:

Print df.describe()

The output will be:



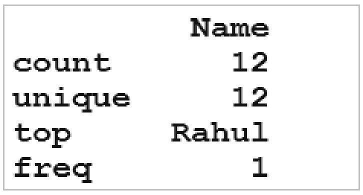
```
             Age      Score
count  12.000000  12.000000
mean   32.500000  73.000000
std     9.209679  17.653225
min    24.000000  44.000000
25%    25.750000  64.000000
50%    29.000000  74.000000
75%    35.250000  87.500000
max    51.000000  99.000000
```

describe() Function with an argument named include along with value object i.e include='object' gives the summary statistics of the character columns.

If we write the following statement:

Print df.decribe(include=['object'])

Then the output will be:



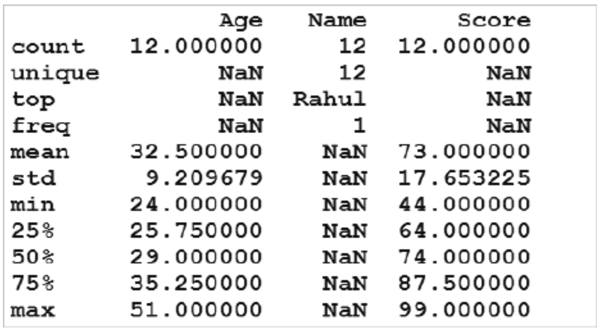
```
         Name
count      12
unique     12
top     Rahul
freq        1
```

describe() Function with include='all' gives the summary statistics of all the columns.

If we write the following statement:

Print df.decribe(include='all')

Then the output will be:



```
              Age   Name      Score
count   12.000000     12  12.000000
unique        NaN     12        NaN
top           NaN  Rahul        NaN
freq          NaN      1        NaN
mean    32.500000    NaN  73.000000
std      9.209679    NaN  17.653225
min     24.000000    NaN  44.000000
25%     25.750000    NaN  64.000000
50%     29.000000    NaN  74.000000
75%     35.250000    NaN  87.500000
max     51.000000    NaN  99.000000
```

However, if we don't require the output for all the functions then we can use a specific function for the output purpose.

Sr.No.	Function	Description
1	count()	Number of non-null observations
2	sum()	Sum of values
3	mean()	Mean of Values
4	median()	Median of Values

5	mode()	Mode of values
6	std()	Standard Deviation of the Values
7	min()	Minimum Value
8	max()	Maximum Value
9	abs()	Absolute Value
10	prod()	Product of Values
11	cumsum()	Cumulative Sum
12	cumprod()	Cumulative Product

For example, if we wish to find standard deviation, then we can write

```
# Program 4.2: To find standard deviation
import pandas as pd
import numpy as np
#Create a Dictionary of series
d = {'Name':pd.Series(['Tom', 'James', 'Ricky', 'Vin', 'Steve', 'Smith', 'Jack', 'Lee', 'David',
'Gasper', 'Betina', 'Andres']), 'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65]) }
#Create a DataFrame
df = pd.DataFrame(d)
print df.std()


Output :
Age      9.232682
Rating   0.661628
dtype: float64
```

4.2.2 Exploratory Data Analysis

Exploratory data analysis (EDA) is used by data scientists to analyse and investigate data sets and summarize their main characteristics, often employing data visualization methods. It helps determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions.

EDA is primarily used to see what data can reveal beyond the formal modelling or hypothesis testing task and provides a provides a better understanding of data set variables and the relationships between them. It can also help determine if the statistical techniques you are considering for data analysis are appropriate. Originally developed by American mathematician John Tukey in the 1970s, EDA techniques continue to be a widely used method in the data discovery process today.

Importance of Exploratory Data Analysis

The main purpose of EDA is to help look at data before making any assumptions. It can help identify obvious errors, as well as better understand patterns within the data, detect outliers or anomalous events, find interesting relations among the variables.

Data scientists can use exploratory analysis to ensure the results they produce are valid and applicable to any desired business outcomes and goals. EDA also helps stakeholders by confirming they are asking the right questions. EDA can help answer questions about standard deviations, categorical variables, and confidence intervals. Once EDA is complete and insights are drawn, its features can then be used for more sophisticated data analysis or modelling, including machine learning.

Exploratory data analysis tools

Specific statistical functions and techniques you can perform with EDA tools include:

- Clustering and dimension reduction techniques, which help create graphical displays of high-dimensional data containing many variables.
- Univariate visualization of each field in the raw dataset, with summary statistics.
- Bivariate visualizations and summary statistics that allow you to assess the relationship between each variable in the dataset and the target variable you're looking at.
- Multivariate visualizations, for mapping and understanding interactions between different fields in the data.
- K-means Clustering is a clustering method in unsupervised learning where data points are assigned into K groups, i.e. the number of clusters, based on the distance from each group's centroid. The data points closest to a particular centroid will be clustered under the same category. K-means Clustering is commonly used in market segmentation, pattern recognition, and image compression.
- Predictive models, such as linear regression, use statistics and data to predict outcomes.

Types of Exploratory Data Analysis

There are four primary types of EDA:

- Univariate non-graphical. This is simplest form of data analysis, where the data being analysed consists of just one variable. Since it's a single variable, it doesn't deal with causes or relationships. The main purpose of univariate analysis is to describe the data and find patterns that exist within it.

- Univariate graphical. Non-graphical methods don’t provide a full picture of the data. Graphical methods are therefore required. Common types of univariate graphics include: (i) Stem-and-leaf plots, which show all data values and the shape of the distribution. (ii) Histograms, a bar plot in which each bar represents the frequency (count) or proportion (count/total count) of cases for a range of values. (iii) Box plots, which graphically depict the five-number summary of minimum, first quartile, median, third quartile, and maximum.
- Multivariate non-graphical: Multivariate data arises from more than one variable. Multivariate non-graphical EDA techniques generally show the relationship between two or more variables of the data through cross-tabulation or statistics.
- Multivariate graphical: Multivariate data uses graphics to display relationships between two or more sets of data. The most used graphic is a grouped bar plot or bar chart with each group representing one level of one of the variables and each bar within a group representing the levels of the other variable. Other common types of multivariate graphics include: (i) Scatter plot, which is used to plot data points on a horizontal and a vertical axis to show how much one variable is affected by another. (ii) Multivariate chart, which is a graphical representation of the relationships between factors and a response. (iii) Run chart, which is a line graph of data plotted over time. (iv) Bubble chart, which is a data visualization that displays multiple circles (bubbles) in a two-dimensional plot. (v) Heat map, which is a graphical representation of data where values are depicted by color.

Exploratory Data Analysis Tools

Some of the most common data science tools used to create an EDA include:

Python: An interpreted, object-oriented programming language with dynamic semantics. Its high-level, built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for rapid application development, as well as for use as a scripting or glue language to connect existing components together. Python and EDA can be used together to identify missing values in a data set, which is important so you can decide how to handle missing values for machine learning.

R: An open-source programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians in data science in developing statistical observations and data analysis.

Steps in Exploratory Data Analysis in Python

There are many steps for conducting Exploratory data analysis.

- Description of data
- Handling missing data
- Handling outliers
- Understanding relationships and new insights through plots

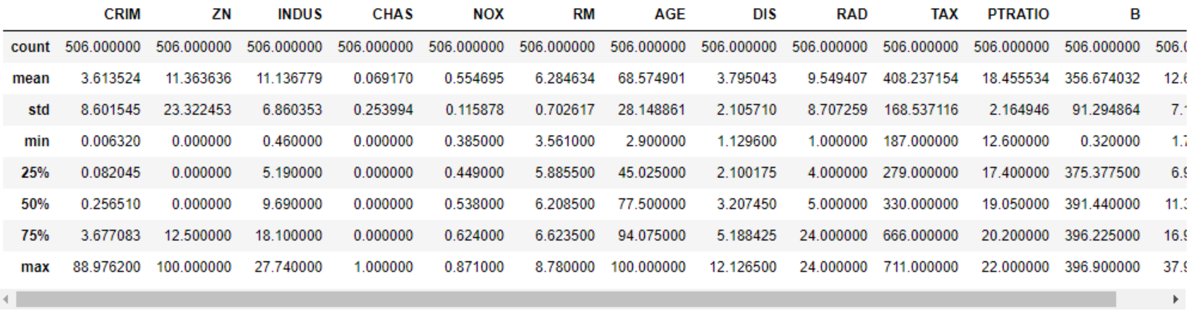
Description of data: We will be using the Boston Data Set for our examples, which can be imported from sklearn.datasets import load_boston.

We need to know the different kinds of data and other statistics of our data before we can move on to the other steps. A good one is to start with the describe() function in python. In Pandas, we can apply describe() on a DataFrame which helps in generating descriptive statistics that summarize the central tendency, dispersion, and shape of a dataset's distribution, excluding NaN values.

The result's index will include count, mean, std, min, max as well as lower, 50 and upper percentiles. By default, the lower percentile is 25 and the upper percentile is 75. The 50 percentile is the same as the median.

```
# Program 4.3: Loading the Dataset
import pandas as pd
from sklearn.datasets import load_boston
boston = load_boston()
x = boston.data
y = boston.target
columns = boston.feature_names
# creating dataframes
boston_df = pd.DataFrame(boston.data)
boston_df.columns = columns
boston_df.describe()
```

Output:



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.7
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.9
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.3
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.9
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.9

Handling missing data: Data in the real-world are rarely clean and homogeneous. Data can either be missing during data extraction or collection due to several reasons. Missing values need to be handled carefully because they reduce the quality of any of our performance matrix. It can also lead to wrong prediction or classification and can also cause a high bias for any given model being used. There are several options for handling missing values. However, the choice of what should be done is largely dependent on the nature of our data and the missing values. Some of the techniques for this are (i) Drop NULL or missing values, (ii) Fill Missing Values, and (iii) Predict Missing values with an ML Algorithm.

The dropping of NULL values is the fastest and easiest step to handle missing values. However, it is not generally advised. This method reduces the quality of our model as it reduces sample size because it works by deleting all other observations where any of the variables is missing. It can be achieved by using dropna() function as shown below:

```
Boston_df.shpae
Output:
(506,13)
```

```
Boston_df = boston_df.dropna()
Boston_df.shape
Output:
(506,13)
```

After implementing the above code, it will indicate that there are no null values in our data set.

Another way of handling missing values is by filling missing values. This is a process whereby missing values are replaced with a test statistic like mean, median or mode of the particular feature the missing value belongs to. Let’s suppose we have a missing value of age in the boston data set. Then the below code will fill the missing value with the 30.

```
Boston_df[‘AGE’] = boston_df[‘AGE’].fillna(30)
Boston_df.shpae
Output:
(506,13)
```

The third way is via predicting missing values with an ML Algorithm. This is by far one of the best and most efficient methods for handling missing data. Depending on the class of data that is missing, one can either use a regression or classification model to predict missing data.

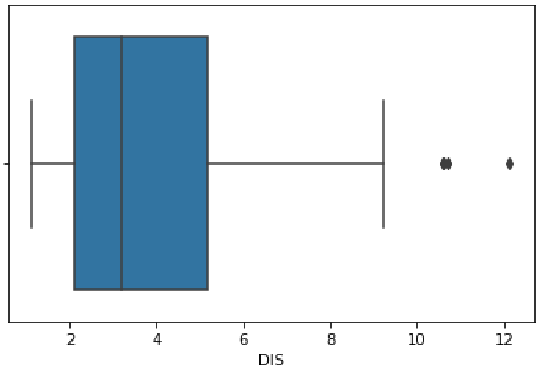
Handling outliers: An outlier is something which is separate or different from the crowd. Outliers can be a result of a mistake during data collection or it can be just an indication of variance in your data. Some of the methods for detecting and handling outliers are (i) BoxPlot (ii) Scatterplot (iii) Z-score and (iv) IQR(Inter-Quartile Range).

BoxPlot: A box plot is a method for graphically depicting groups of numerical data through their quartiles. The box extends from the Q1 to Q3 quartile values of the data, with a line at the median (Q2). The whiskers extend from the edges of the box to show the range of the data. Outlier points are those past the end of the whiskers. Boxplots show robust measures of location and spread as well as providing information about symmetry and outliers.

```
# Program 4.4: BoxPlot

import seaborn as sns

sns.boxplot(x=boston_df['DIS'])
```

Output:

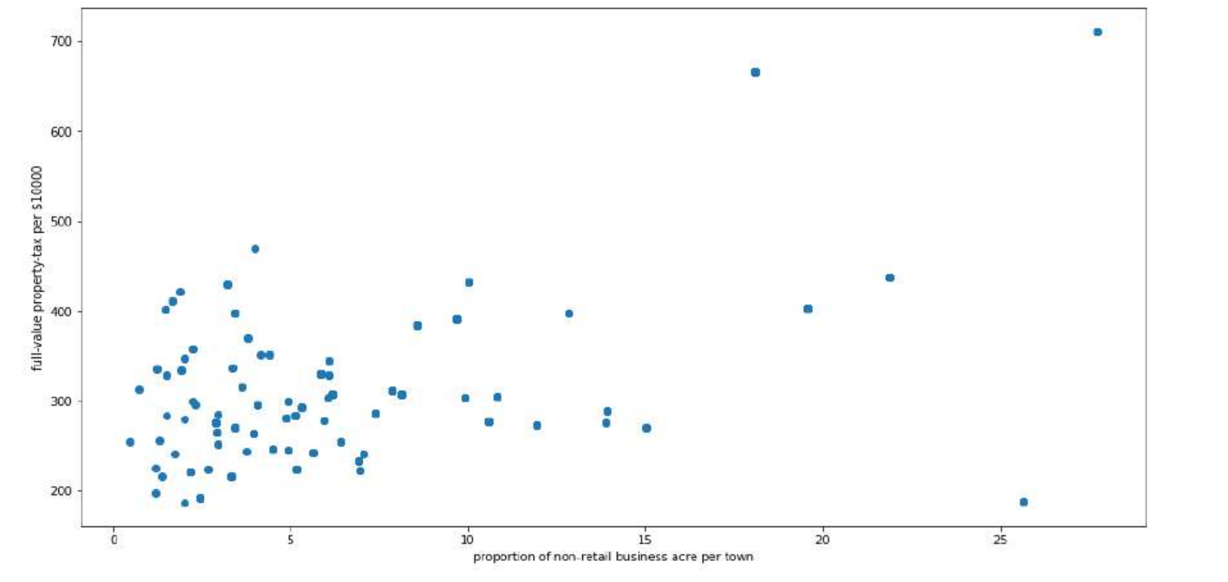
```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1525a92b240>
```



Scatterplot: A scatter plot is a mathematical diagram using Cartesian coordinates to display values for two variables for a set of data. The data are displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable determining the position on the vertical axis. The points that are far from the population can be termed as an outlier.

```
# Program 4.5: Scatter Plot

import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(16,8))

ax.scatter(boston_df['INDUS'] , boston_df['TAX'])

ax.set_xlabel('proportion of non-retail business acre per town')

ax.set_ylabel('full-value property-tax per $10000')

plt.show()
```

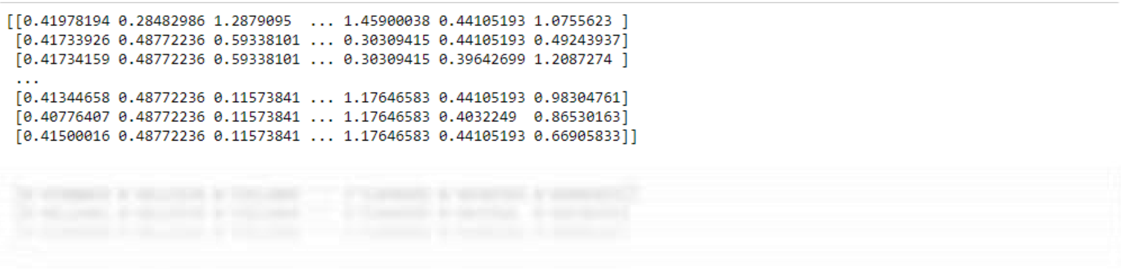
Output:



Z-score: The Z-score is the signed number of standard deviations by which the value of an observation or data point is above the mean value of what is being observed or measured. While calculating the Z-score we re-scale and center the data and look for data points that are too far from zero. These data points which are way too far from zero will be treated as the outliers. In most of the cases a threshold of 3 or -3 is used i.e. if the Z-score value is greater than or less than 3 or -3 respectively, that data point will be identified as outliers.

```
# Program 4.6: Z-Score
From scipy import stats
Import numpy as np
Z = np.abs(stats.zscore(boston_df))
Print(Z)
```

Output:



```
[[0.41978194 0.28482986 1.2879095  ... 1.45900038 0.44105193 1.0755623 ]
 [0.41733926 0.48772236 0.59338101 ... 0.30309415 0.44105193 0.49243937]
 [0.41734159 0.48772236 0.59338101 ... 0.30309415 0.39642699 1.2087274 ]
 ...
 [0.41344658 0.48772236 0.11573841 ... 1.17646583 0.44105193 0.98304761]
 [0.40776407 0.48772236 0.11573841 ... 1.17646583 0.4032249  0.86530163]
 [0.41500016 0.48772236 0.11573841 ... 1.17646583 0.44105193 0.66905833]]
```

For removing outliers, following statements may be used:

```
Boston_df_outlier_Zscore = boston_df[(z<3).all(axis=1)]
Boston_df_outlier_Zscore.shape
```

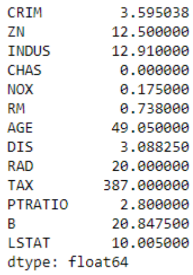
```
Output:

(415,13)
```

We can see from the above code that the shape changes, which indicates that our dataset has some outliers.

IQR: The interquartile range (IQR) is a measure of statistical dispersion, being equal to the difference between 75th and 25th percentiles, or between upper and lower quartiles.

IQR = Q3 − Q1 and can be found using following statements:

```
Q1 = boston_df.quantile(0.25)

Q3 = boston_df_quantile(0.75)

IQR = Q3 – Q1

Print(IQR)


Output:
```



```
CRIM        3.595038
ZN         12.500000
INDUS      12.910000
CHAS        0.000000
NOX         0.175000
RM          0.738000
AGE        49.050000
DIS         3.088250
RAD        20.000000
TAX       387.000000
PTRATIO     2.800000
B          20.847500
LSTAT      10.005000
dtype: float64
```

Once we have IQR scores below code will remove all the outliers in our dataset.

```
Boston_df_outlier-IQR= boston_df[~((boston_df < (Q1 – 1.5 * IQR)) | (boston_df  > (Q3 + 1.5 * IQR))).any(axis=1)]

Boston_df_outlier_IQR.shape

Output:

(274, 13)
```

4.2.3 Data Visualization

Data visualization is a graphical representation of quantitative information and data by using visual elements like graphs, charts, and maps. Data visualization convert large and small data sets into visuals, which is easy to understand and process for humans. Data visualization tools provide accessible ways to understand outliers, patterns, and trends in the data.

In the world of Big Data, the data visualization tools and technologies are required to analyse vast amounts of information. Data visualizations are common in our everyday life, but they always appear in the form of graphs and charts.

Data visualizations are used to discover unknown facts and trends. We can see visualizations in the form of line charts to display change over time. Bar and column charts are useful for observing relationships and making comparisons. A pie chart is a great way to show parts-of-a-whole and maps are the best way to share geographical data visually.

Today's data visualization tools go beyond the charts and graphs used in the Microsoft Excel spreadsheet, which displays the data in more sophisticated ways such as dials and gauges, geographic maps, heat maps, pie chart, and fever chart.

Effective Data Visualization

American Statistician and Yale Professor Edward Tufte saya useful data visualizations consist of complex ideas communicated with clarity, precision, and efficiency.

To craft an effective data visualization, we need to start with clean data that is well-sourced and complete. Once the data is ready to visualize, we need to pick the right chart to visualize. After that, we need to design and customize our visualization according to requirements. Simplicity is essential as we don't want to add any elements that distract from the data.

Importance of Data Visualization

- Data visualization is important because of the processing of information in human brains. Using graphs and charts to visualize a large amount of the complex data sets is more comfortable in comparison to studying the spreadsheet and reports.
- Data visualization is an easy and quick way to convey concepts universally. We can experiment with a different outline by making a slight adjustment.
- Data visualization can identify areas that need improvement or modifications.
- Data visualization can clarify which factor influence customer behavior.
- Data visualization helps you to understand which products to place where.
- Data visualization can predict sales volumes.

Uses of Data Visualization

- To make easier in understand and remember.
- To discover unknown facts, outliers, and trends.
- To visualize relationships and patterns quickly.
- To ask a better question and make better decisions.
- To competitive analyse.
- To improve insights.

Types of Data Visualizations

The earliest form of data visualization can be traced back the Egyptians in the pre-17th century, largely used to assist in navigation. As time progressed, people leveraged data visualizations for broader applications, such as in economic, social, health disciplines. Perhaps most notably, Edward Tufte published "The Visual Display of Quantitative Information" , which illustrated that individuals could utilize data visualization to present data in a more effective manner. His book continues to stand the test of time, especially as companies turn to dashboards to report their performance metrics in real-time. Dashboards are effective data visualization tools for tracking and visualizing data from multiple data sources, providing visibility into the effects of specific behaviors by a team or an adjacent one on performance. Dashboards include common visualization techniques, such as:

Tables: This consists of rows and columns used to compare variables. Tables can show a great deal of information in a structured way, but they can also overwhelm users that are simply looking for high-level trends.

Pie charts and Stacked Bar Charts: These graphs are divided into sections that represent parts of a whole. They provide a simple way to organize data and compare the size of each component to one other.

Line graphs and Area charts: These visuals show change in one or more quantities by plotting a series of data points over time. Line graphs utilize lines to demonstrate these changes while area charts connect data points with line segments, stacking variables on top of one another and using color to distinguish between variables.

Histograms: This graph plots a distribution of numbers using a bar chart (with no spaces between the bars), representing the quantity of data that falls within a particular range. This visual makes it easy for an end user to identify outliers within a given dataset.

Scatter plots: These visuals are beneficial in revealing the relationship between two variables, and they are commonly used within regression data analysis. However, these can sometimes be confused with bubble charts, which are used to visualize three variables via the x-axis, the y-axis, and the size of the bubble.

Heat maps: These graphical displays are helpful in visualizing behavioral data by location. This can be a location on a map, or even a webpage.

Tree maps: These display hierarchical data as a set of nested shapes, typically rectangles. Treemaps are great for comparing the proportions between categories via their area size.

Data Visualization in Python

Data visualization in python is perhaps one of the most utilized features for data science with python in today's day and age. The libraries in python come with lots of different features that enable users to make highly customized, elegant, and interactive plots.

Useful packages for visualizations in python are Matplotlib, Seaborn, Statistics, Pandas and Numpy.

Matplotlib: Matplotlib is a visualization library in Python for 2D plots of arrays. Matplotlib is written in Python and makes use of the NumPy library. It can be used in Python and IPython shells, Jupyter notebook, and web application servers. Matplotlib comes with a wide variety of plots like line, bar, scatter, histogram, etc., which can help us, into understanding trends, patterns, correlations.

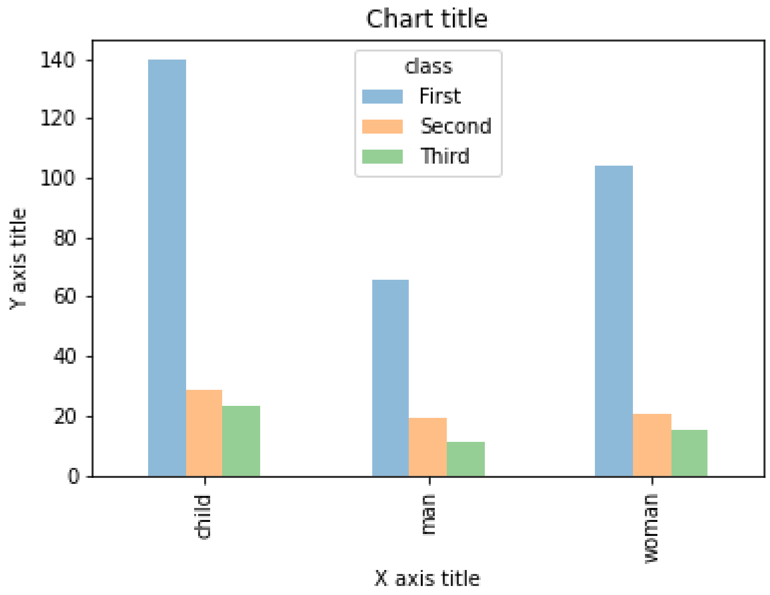
Seaborn: Seaborn is a dataset-oriented library for making statistical representations in Python. It is developed atop matplotlib and to create different visualizations. It is integrated with pandas data structures. The library internally performs the required mapping and aggregation to create informative visuals.

Grouped bar chart: A grouped bar chart is used when we want to compare the values in certain groups and sub-groups

Grouped bar chart using Matplotlib: In the following code a data set named as Titanic has been used which can easily be downloaded from Kaggle.

```
# Program 4.7: Grouped Bar Chart using Matplotlib
#Creating the dataset
import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset('titanic')
df_pivot = pd.pivot_table(df, values="fare", index="who", columns="class",
aggfunc=np.mean)
#Creating a grouped bar chart
ax = df_pivot.plot(kind="bar", alpha=0.5)
#Adding the aesthetics
plt.title('Chart title')
plt.xlabel('X axis title')
plt.ylabel('Y axis title')
# Show the plot
plt.show()
```

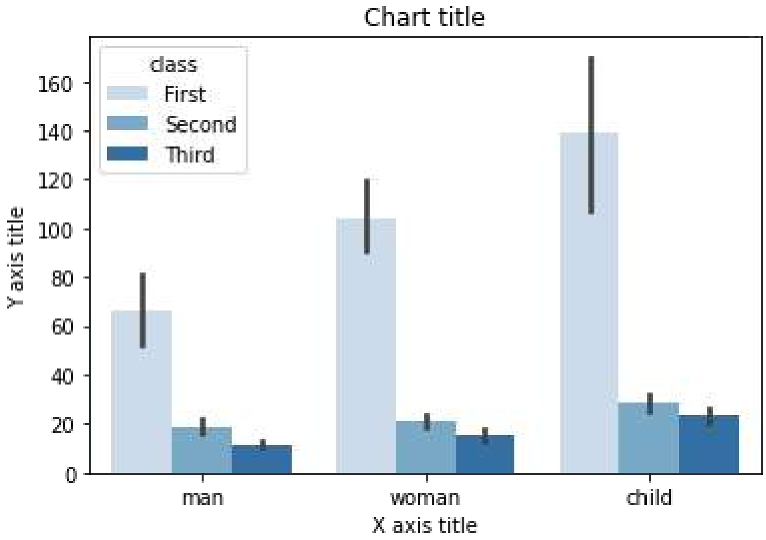
Output:



Grouped bar chart using Seaborn:

```
# Program 4.8: Grouped bar chart using seaborn
#Reading the dataset
import seaborn as sns
import matplotlib.pyplot as plt
titanic_dataset = sns.load_dataset('titanic')
#Creating the bar plot grouped across classes
sns.barplot(x = 'who', y = 'fare', hue = 'class', data = titanic_dataset, palette = "Blues")
#Adding the aesthetics
plt.title('Chart title')
plt.xlabel('X axis title')
plt.ylabel('Y axis title')
# Show the plot
plt.show()
```

Output:



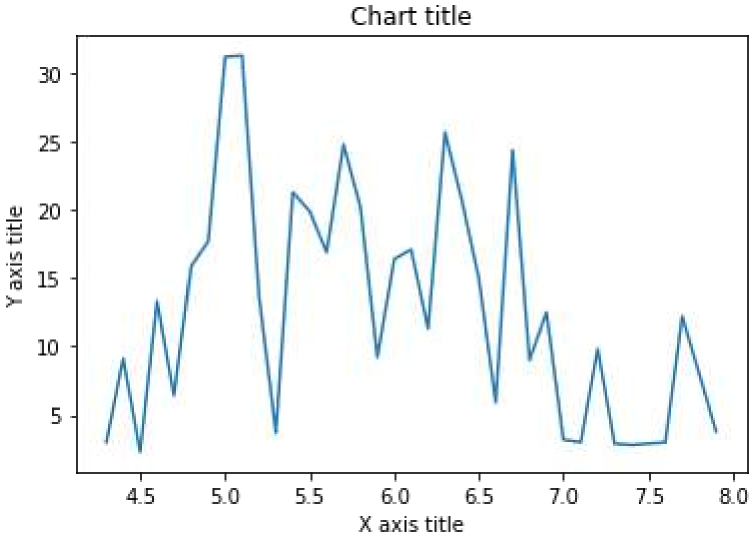
Line chart: A line chart is used for the representation of continuous data points. This visual can be effectively utilized when we want to understand the trend across time.

In the following example "iris" dataset has been used which can be downloaded from Kaggle.

```
#Program 4.9: Line Chart
#Creating the dataset
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df = sns.load_dataset("iris")

df=df.groupby('sepal_length')['sepal_width'].sum().to_frame().reset_index()

#Creating the line chart

plt.plot(df['sepal_length'], df['sepal_width'])

#Adding the aesthetics

plt.title('Chart title')

plt.xlabel('X axis title')

plt.ylabel('Y axis title')

#Show the plot

plt.show()
```

Output:

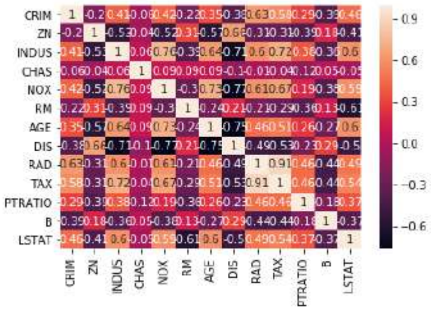


HeatMaps: The Heat Map procedure shows the distribution of a quantitative variable over all combinations of 2 categorical factors. If one of the 2 factors represents time, then the evolution of the variable can be easily viewed using the map. A gradient color scale is used to represent the values of the quantitative variable. The correlation between two random variables is a number that runs from -1 through 0 to +1 and indicates a strong inverse relationship, no relationship, and a strong direct relationship, respectively.

It can be created using following statement using boston data set as used in the earlier section

```
Corr_mat = boston_df.corr().round(2)

Sns.heatmap(data=corr_mat, annot=True)
```

Output:



Other types of graphs have already been discussed in the previous modules.

4.3 SUMMARY

In this module, three important topics have been discussed. The topics discussed in this module are descriptive statistics, exploratory data analysis and data visualization. All the three topic are related to each other. Starting with descriptive statistics involves the steps to analyse the data using the measures discussed in previous two modules i.e. module-II and module-III. The descriptive statistics explain the measures of central tendency, measure of dispersion along with the frequency distribution used for the given data. Following this exploratory data analysis involves the various steps of data analysis such as identifying data sources, cleaning, presenting and interpreting. When we talk about data presenting, data visualization comes into picture. Data visualization involves the presentation of data using suitable means i.e. graphs so that it becomes easy to understand and interpret. The implementation of these topics have been explained using python.

4.4 PRACTICE QUESTIONS

Q.1 What is meant by descriptive statistics? Describe the purpose of descriptive statistics.

Q.2 Explain various types of descriptive statistics in detail.

Q.3 Describe various functions in python for performing descriptive statistics.

Q.4 What do you mean by exploratory data analysis. Explain its significance.

Q.5 Write and explain the various steps for performing exploratory data analysis.

Q.6 What are the various packages in python that can be used for performing exploratory data analysis? Explain in detail.

Q.7 How can you draw (i) histogram (ii) bar chart (iii) heat map (iv) grouped bar chart (v) pie chart, in python? Explain by writing suitable programs.

Q.8 What is meant by data visualization? Explain the various steps in data visualization.

Q.9 Discuss various tools used for data visualization.

REFERENCES

1. A. Abebe, J. Daniels, J.W.Mckean, “Statistics and Data Analysis”.
2. A. Martelli, A. Ravenscroft, S. Holden, “Python in a Nutshell”, OREILLY.

3. Clarke, G.M. & Cooke, D., “A Basic course in Statistics”, Arnold.
4. David M. Lane, “Introduction to Statistics”.
5. Eric Matthes, “Python Crash Course: A Hands-On, Project-Based Introduction to Programming”.\
6. S.C.Gupta and V.K.Kapoor, “Fundamentals of Mathematical Statistics”, Sultan Chand & Sons, New Delhi.
7. Weiss, N.A., “Introductory Statistics”, Addison Wesley

B.Sc.(DATA SCIENCE)

SEMESTER-II

STATISTICAL FOUNDATION

UNIT V: CORRELATION AND REGRESSION

STRUCTURE

5.0 OBJECTIVES

The main goal of this module is to help students learn, understand and practice the basics of statistics which will helpful to do the research in the social sciences. In this module you will learn the basics of statistics which covers two fundamentals concepts correlation and regression. The examples of this module were calculated manual as well as using programming language. Python language is used in this module.

5.1 INTRODUCTION

Data science become a buzzword that everyone talks about the data science. Data science is an interdisciplinary field that combines different domain expertise, computer programming skills, mathematics and statistical knowledge to find or extract the meaningful or unknown patterns from unstructured and structure dataset.

Data science is useful for extraction, preparation, analysis and visualization of data. Various statistical methods can be applied to get insight in the data.

Data science is all about using data to solve problems. Data has become the fuel of industries. It is most demandable field of 21st century. Every industry require data to functioning, searching, marketing, growing, expanding their business.

The application of areas of data science are health care, fraud detection, disease predicting, real time shipping routes, speech recognition, targeting advertising, gaming and many more.

5.2 CORRELATION

Correlation is a statistical measure that expresses the extent to which two or more variables are changes together at a constant rate. It is a relationship between two or more variables. The data can be represented by the pairs (x, y) where x is an independent variable and y is a dependent variable.

Correlations are useful for describing simple relationships among the data. It quantifies the degree and direction to which two variables are related. It is a measure of the extent to which two variables are related.

5.3 TYPES of CORRELATION

The different types of correlations on the basis of: the degree of correlation, numbers of variables used and linearity.

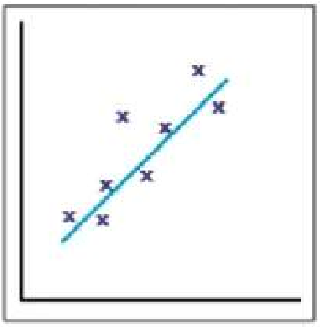
5.3.1 Positive, Negative and Zero Correlation

The positive, negative and zero correlations are basis on the degree of correlation.

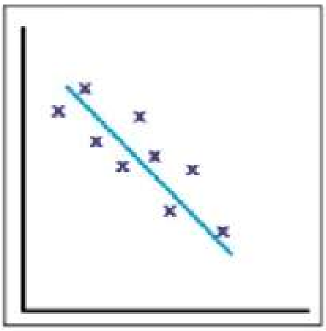
- **Positive Correlation** – A relationship between two variables in which both the variables move in same direction, it means that when one variable is increases as the other variable increases, or one variable is decreases while the other decreases. The examples of positive correlation are: height and weight of person, price and supply of a commodity, etc.
- **Negative Correlation –** A relationship between two variables in which both the variable moves in opposite direction. That means when one variable is increase as the other variable decreases, or one variable decrease while the other increases.

The examples of negative correlation are: no. of absent and grade of student, speed of train and time to reach destination, etc.

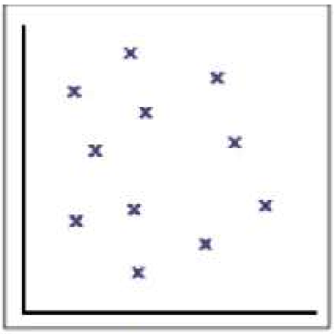
- **No / Zero Correlation** – There is no linear dependence or no relationship between two variables. The example of no / zero correlation is the coffee drunk and level of IQ.



Positive Correlation



Negative Correlation



No Correlation

5.3.2 Simple, Multiple and Partial Correlation

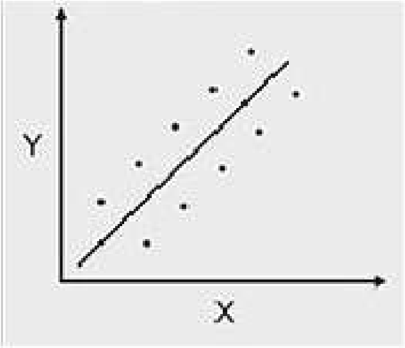
The simple, multiple and partial correlations are depending on the number of variables studied in the analysis.

- **Simple Correlation** – There are only two variables are studied and check the correlation between them is called simple correlation. Examples of simple correlation are age and height of students, price and demand of commodities.
- **Multiple Correlation** – There are three or more variables are studied for correlation simultaneously is called multiple correlation. Example of multiple correlation is to study the relationship between the yield of any crop, amount of fertilizers used and amount of rainfall.
- **Partial Correlation** – There are three or more variables are studied. When one or more variables are kept constant and the relationship is studied between others is called partial correlation. Example of partial correlation is the price of ice-cream, temperature and demand. If we kept the price of ice-cream is constant and studied the correlation between temperature and demand of ice-cream.

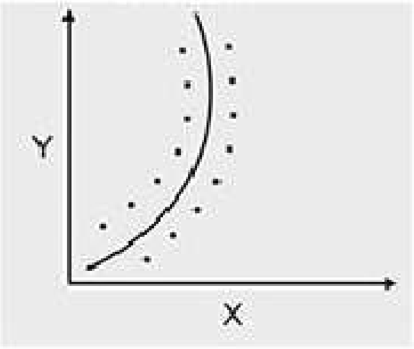
5.3.3 Linear and Non-Linear Correlation

The linear and non-linear correlation are on the basis of linearity of data.

- **Linear Correlation** – The correlation is said to be a linear if the ratio of change of the two variables is constant. If we plot all the points on the scatter diagram tend to lie near a line which like a straight line.
- **Non-Linear Correlation** – The correlation is said to be a non-linear or curvilinear if the ratio of change of the two variables is not constant. If we plot all the points on the scatter diagram tend to lie near a smooth curve which like a curve.



Linear Correlation



Non-Linear Correlation

The **corr()** function is used to find the correlation between two variables in Python. Here we take an example of boys age and weight to calculate the correlation.

Example: Find the correlation of boys age and weight.

Age	10	20	30	40	50	60	70	80
Weight	32	48	59	64	70	67	78	82

The below code will find the correlation of boys age and weight.

```
# Importing library
import pandas as pd

# Data of Age and Weight
Age = pd.Series([10, 20, 30, 40, 50, 60, 70, 80])
Weight = pd.Series([32, 48, 59, 64, 70, 67, 78, 82])

#Calculating Correlation
correlation = Age.corr(Weight)
correlation
```

The above code will calculate the correlation between age and weight of boys as follow:

```
0.9501687384314103
```

Example: Find the correlation of boys age and weight.

Age	10	20	30	40	50	60	70	80
Weight	32	48	59	64	70	67	78	82

The below code will find the correlation of boys age and weight.

```
# Importing library
import pandas as pd

# Data of Age and Weight
data = {
   "Age" : [10, 20, 30, 40, 50, 60, 70, 80],
   "Weight" : [32, 48, 59, 64, 70, 67, 78, 82]
            }

# Creation of Dataframe
df = pd.DataFrame(data)

# Creation of Correlation Matrix
df.corr()
```

The above code will calculate the correlation between age and weight of boys as follow:

	Age	**Weight**
Age	1.000000	0.950169
Weight	0.950169	1.000000

Now we take an example of three subjects mark of science student to calculate the correlation.

Example: Find the correlation between marks of three subjects maths, chemistry and physics of science students.

Maths	100	86	90	80	96	95	92	99
Chemistry	88	90	80	90	90	86	94	88
Physics	92	88	89	94	90	87	93	91

The below code will find the correlation between marks of three subjects such as maths, chemistry and physics of science students.

```
# Importing library
import pandas as pd

data = {
   "Maths" : [100, 86, 90, 80, 96, 95, 92, 99],
   "Chemistry" : [88, 90, 80, 90, 90, 86, 94, 88],
   "Physics" : [92, 88, 89, 94, 90, 87, 93, 91]
             }

df = pd.DataFrame(data)
```

```
df.corr()
```

The above code will calculate the correlation between maths, chemistry and physics subject marks of science students as follow:

	Maths	**Chemistry**	**Physics**
Maths	1.000000	-0.096004	-0.180719
Chemistry	-0.096004	1.000000	0.502519
Physics	-0.180719	0.502519	1.000000

From the above table we can see that, the correlation between maths and chemistry is negative (-0.096004), between maths and physics is also negative (-0.180719) and between chemistry and physics is positive (0.502519).

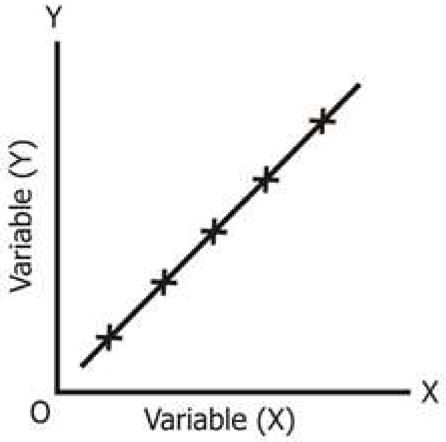
5.4 TECHNIQUES FOR MEASURING CORRELATION

The different techniques for measuring correlation are: graphical method and algebraic method. Scatter diagram is a graphical method. Karl Pearson’s correlation coefficient and Spearman’s rank correlation coefficient are algebraic methods.

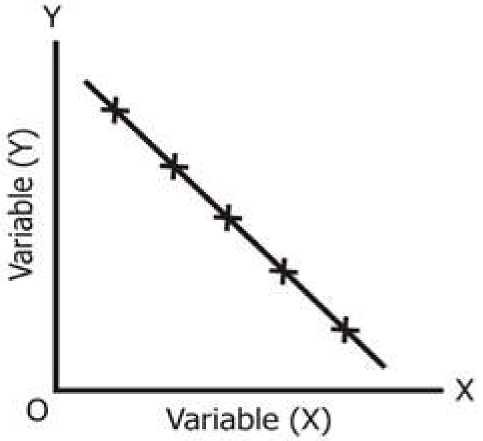
5.4.1 Scatter Plot

Scatter diagram is a graph in which the values of two variables are plotted along with two axes. It is a most basic type of plot that helps you visualize the relationship between two variables. Each value in this plot is represent by a dot. It is a set of dotted points to represent the data on both horizontal and vertical axis.

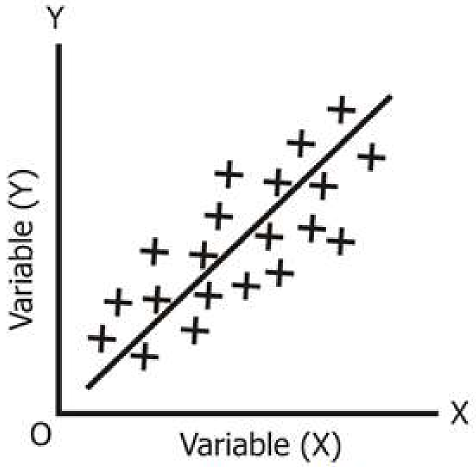
- **Perfect positive correlation :** All the plotted points are on a straight line which is rising from lower left corner to the upper right corner in a scatter diagram.
- **Perfect negative correlation :** All the plotted points are on a straight line which is rising from upper left corner to the lower right corner in a scatter diagram.
- **Strong positive correlation :** All the plotted points are closer to a straight line which is rising from lower left corner to the upper right corner in a scatter diagram.
- **Strong negative correlation :** All the plotted points are closer to a straight line which is rising from upper left corner to the lower right corner in a scatter diagram.
- **Weak positive correlation :** All the plotted points are not closer (lie away) to a straight line which is rising from lower left corner to the upper right corner in a scatter diagram.
- **Weak negative correlation :** All the plotted points are not closer (lie away) to a straight line which is rising from upper left corner to the lower right corner in a scatter diagram.
- **No correlation :** All the plotted points are scattered randomly across the graph.



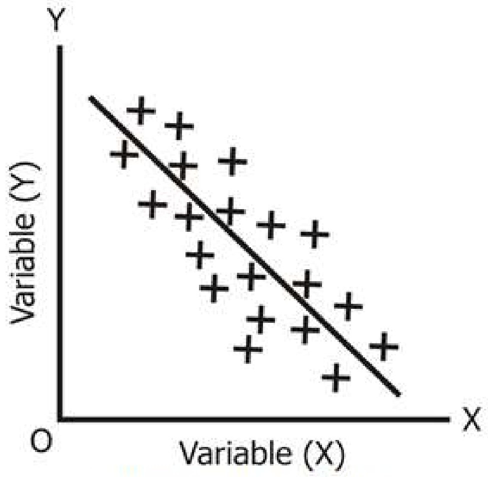
Perfect Positive Correlation



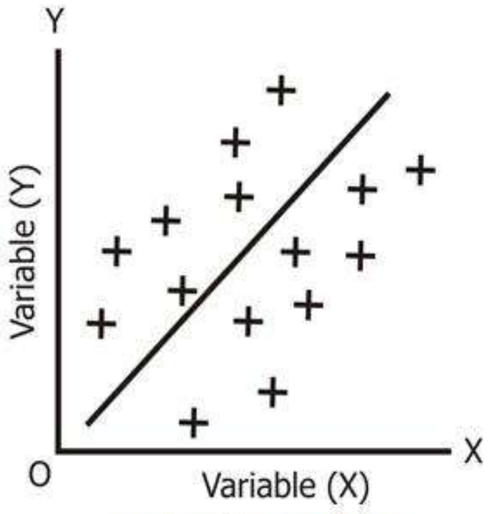
Perfect Negative Correlation



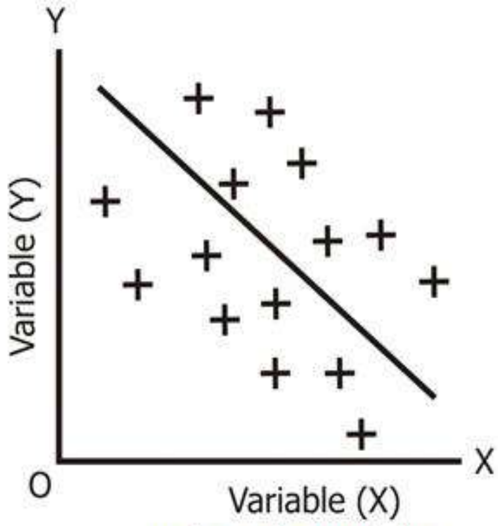
Strong Positive Correlation



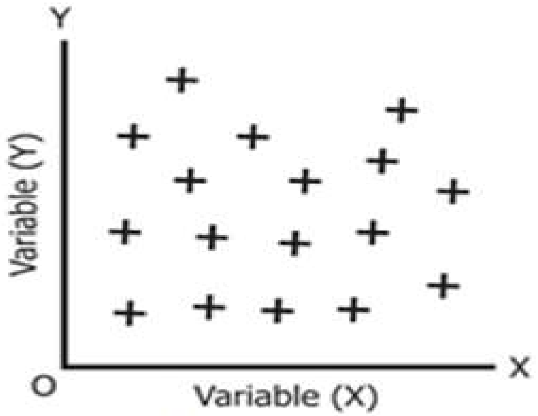
Strong Negative Correlation



Weak Positive Correlation



Weak Negative Correlation



No Correlation

The **scatter()** function is used to draw the scatter plot in Python. It plots one dot for each observation. It required two different set of observation with same length for both the axis.

Here we take an example of boys age and weight. The x-axis represents age and y-axis represents weight of boys.

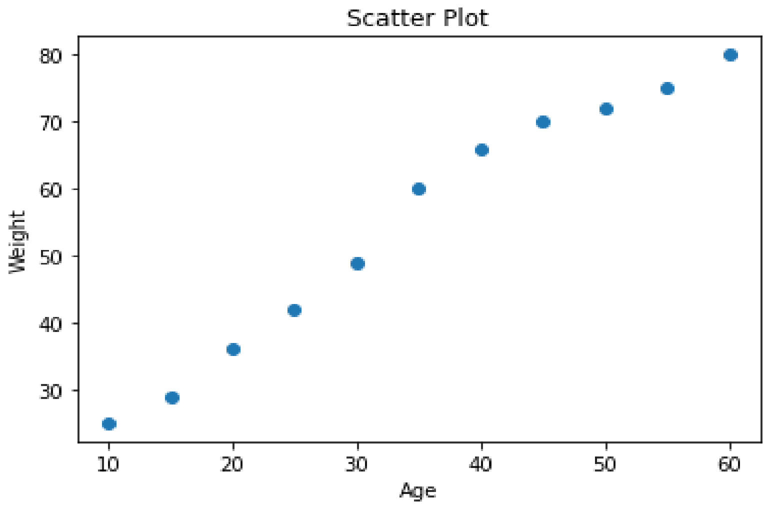
Example: Plot the scatter diagram of age and weight of boys.

Age	10	15	20	25	30	35	40	45	50	55	60
Weight	25	29	36	42	49	60	66	70	72	75	80

The below code will plot the scatter diagram of age and weight of boys.

```
# Importing library
import matplotlib.pyplot as plt

# Data of Age and Weight
Age = [10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60]
Weight = [25, 29, 36, 42, 49, 60, 66, 70, 72, 75, 80]

# Plotting scatter plot with title and label
plt.scatter(Age, Weight)
plt.title("Scatter Plot")
plt.xlabel("Age")
plt.ylabel("Weight")

# Show plot
plt.show()
```

The above code will create scatter plot as follow:



In the above scatter plot we can see the relationship between two variables age and weight. There is a positive correlation between boys age and weight.

Another method is to draw scatter plot using ***"kind"*** parameter. Here we take an example of two subject marks. The x-axis represents chemistry and y-axis represents physics subject marks.

Example: Plot the scatter diagram of chemistry and physics subject marks.

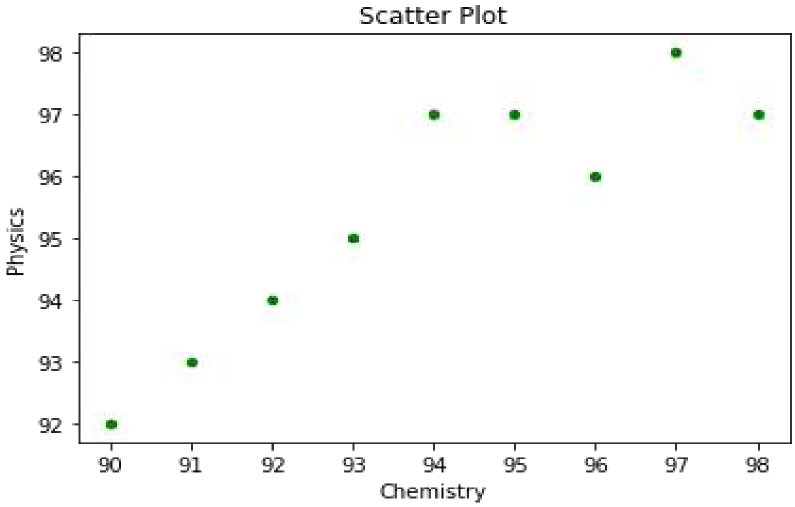
Chemistry	90	92	97	98	96	94	91	93	95
Physics	92	94	98	97	96	97	93	95	97

The below code will plot the scatter diagram of chemistry and physics subject marks.

```
# Importing library
import pandas as pd
import matplotlib.pyplot as plt

# Data of Chemistry and Physics subject marks
data = {
  "Chemistry_Marks" : [90, 92, 97, 98, 96, 94, 91, 93, 95],
  "Physics_Marks" : [92, 94, 98, 97, 96, 97, 93, 95, 97]
             }


# Dataframe Creation
df = pd.DataFrame(data)

# Plotting scatter plot with title and label
df.plot(x='Chemistry_Marks', y='Physics_Marks', kind = 'scatter', color = 'Green')
plt.title("Scatter Plot")
plt.xlabel("Chemistry")
plt.ylabel("Physics")
```

```
# show the plot
plt.show()
```

The above code will create scatter plot as follow:



In the above scatter plot we can see the relationship between chemistry and physics subject marks.

5.4.2 Karl Pearson Coefficient of Correlation

This is the most common measure of correlation. It is also known as Pearson Product Moment Correlation (PPMC). It is used to measure the strength and direction of the linear relationship between two variables in correlation analysis. It is used to measure the degree of association between variables. The correlation coefficient is symbolized by r which lies between -1 and 1.

- The value of r is 1 indicates perfect positive correlation.
- The value of r is -1 indicates perfect negative correlation.
- The value of r is 0 indicates no relationship.

The following formula is used to calculate the Pearson correlation.

$$\boldsymbol{r} = \frac{\boldsymbol{n}\,(\sum \boldsymbol{xy}\,) - (\sum \boldsymbol{x}\,)\,(\sum \boldsymbol{y}\,)}{\sqrt{[\,\boldsymbol{n}\,\sum \boldsymbol{x}^{2} - (\sum \boldsymbol{x}\,)^{2}\,][\,\boldsymbol{n}\,\sum \boldsymbol{y}^{2} - (\sum \boldsymbol{y}\,)^{2}\,]}}$$

Here,

n = No. of values
ΣX = Total of the First Variable Value
ΣY = Total of the Second Variable Value
ΣXY = Sum of the Product of first and Second Value
ΣX^2 = Sum of the Squares of the First Value
ΣY^2 = Sum of the Squares of the Second Value

Example : Calculate the Karl Pearson's Coefficient of correlation from the following data.

X	10	20	30	40	50	60	70	80
Y	32	48	59	64	70	67	78	82

Solution:

X	**Y**	**XY**	**X^2**	**Y^2**
10	32	320	100	1024
20	48	960	400	2304
30	59	1770	900	3481
40	64	2560	1600	4096
50	70	3500	2500	4900
60	67	4020	3600	4489
70	78	5460	4900	6084
80	82	6560	6400	6724
ΣX = 360	**ΣY = 500**	**ΣXY = 25150**	**ΣX^2 = 20400**	**ΣY^2 = 33102**

Now, we calculate

$$r = \frac{n\,(\sum xy\,) - (\sum x\,)\,(\sum y\,)}{\sqrt{[\,n\,\sum x^2 - (\sum x\,)^2\,][\,n\,\sum y^2 - (\sum y\,)^2\,]}}$$

$$r = \frac{8\,(25150) - (360)(500)}{\sqrt{[\,8\,(20400) - (360)(360)][\,8\,(33102) - (500)(500)\,]}}$$

$$r\ = \frac{201200 - 180000}{\sqrt{[\,163200 - 129600\,]\,[\,264816 - 250000\,]}}$$

$$r\ = \frac{21200}{\sqrt{(\,33600\,)\,(14816)}}$$

$$r\ = \frac{21200}{\sqrt{497817600}}$$

$$r\ = \frac{21200}{22311.826}$$

$$\boldsymbol{r\ =\ 0.9501687}$$

Now we will calculate the Karl Pearson's Coefficient of correlation using Python.

The below code will calculate the Karl Pearson's Coefficient.

```
# Importing library
import pandas as pd

# Data of Age and Weight
data = {"Age" : [10, 20, 30, 40, 50, 60, 70, 80],
       "Weight" : [32, 48, 59, 64, 70, 67, 78, 82]}

df = pd.DataFrame(data)

#Calculating Correlation
df.corr(method='pearson')
```

The above code will give the following result :

	Age	**Weight**
Age	1.000000	0.950169
Weight	0.950169	1.000000

Here we can see that the value of Karl Pearson's Coefficient of correlation (r) is 0.950169 which is same as the manual calculation as above.

5.4.3 **Spearman's Rank Correlation Coefficient**

In year 1904, Charles Edward Spearman introduced a new method of measuring the correlation between two variables. It is applicable to individual observation. In this method, the rank (or order) of the observation is taken instead of the value of variable. This correlation coefficient is called rank correlation coefficient. This is useful to measure the qualitative characteristics such as beauty, honesty, height etc.

The following formula is used to calculate the Spearman's Rank Coefficient of correlation.

$$\boldsymbol{r = 1 - \frac{6 \sum D^2}{n\,(n^2 - 1)}}$$

Here,

r = Spearman rank correlation coefficient

n = No. of pairs of observation

D = Differences in ranks between pair observation

Example : Calculate the Spearman's Rank Coefficient of correlation from the following data.

X	1	6	5	10	3	2	4	9	7	8
Y	6	4	9	8	1	2	3	10	5	7

Solution:

Rank (X)	Rank (Y)	D = R (X) – (Y)	D^2
1	6	-5	25
6	4	2	4
5	9	-4	16
10	8	2	4
3	1	2	4
2	2	0	0
4	3	1	1
9	10	-1	1
7	5	2	4
8	7	1	1
N = 10			**$\sum D^2 = 60$**

Now, we calculate

$$r = 1 - \frac{6 \sum D^2}{n\,(n^2 - 1)}$$

$$r = 1 - \frac{6\,(60)}{10\,(100 - 1)}$$

$$r = 1 - \frac{360}{990}$$

$$r = 1 - \ 0.36363$$

$$\boldsymbol{r = 0.63636}$$

Now we will calculate the Spearman’s Rank Coefficient of correlation using Python.

The below code will calculate the Spearman’s Rank Coefficient.

```
# Importing library
import pandas as pd

# Data
data = {"X" : [1,6,5,10,3,2,4,9,7,8],
        "Y" : [6,4,9,8,1,2,3,10,5,7]}

df = pd.DataFrame(data)

#Calculating Correlation
df.corr(method='spearman')
```

The above code will give the following result :

	X	Y
X	1.000000	0.636364
Y	0.636364	1.000000

Here we can see that the value of Spearman’s Rank Coefficient of correlation (r) is 0.636364 which is same as the manual calculation as above.

5.5 REGRESSION ANALYSIS

Regression analysis is a set of various statistical methods for estimating the relationship between a dependent variable (also called outcome variable) and one or more independent variables (also called predictor variable). This is widely used to predict the outputs, forecasting, time series analysis and finding the causal effect dependencies.

There are many different types of regression techniques based on the number of dependent variables, the dimensionality of regression line and the types of dependent variable such as linear regression, polynomial regression, decision tree regression, support vector regression, random forest regression, logistic regression, etc. The most frequently used regression analysis is linear regression.

The following terms are related to regression analysis:

- **Dependent variable or target variable:** Variable to predict.
- **Independent variable or predictor variable:** Variables to estimate the dependent variable.
- **Outlier:** The observation which differs significantly from the other observation.
- **Normality**: The data follows a normal distribution.
- **Multicollinearity:** It is a situation in which two or more independent variables are highly linearly related.
- **Homoscedasticity:** It is a situation in which the error term is the same across all values of the independent variables. It is also called homogeneity of variance.

5.6 APPLICATIONS OF REGRESSION ANALYSIS

Regression analysis refers to a group of techniques for studying the relationships among two or more variables based on a sample. Linear regression is one of the most commonly used techniques in statistics which is used to quantify the relationship between one or more predictor variables and a response variable.

Regression analysis is used for prediction and forecasting. This statistical method is used in various sectors. The most common applications of regression analysis are:

- **Financial Sector** – Regression analysis is used to calculate the Beta (volatility of returns relative to the overall market) which is used as a measure of risk. A company with a higher beta has a greater risk and greater expected returns also. It is also used in stock market to understand the trend of stocks and forecast the prices of different stocks.

- **Marketing Sector** – Regression analysis is used to understand the effectiveness of various marketing campaigns, forecasting the sales and pricing of the products. It is also used to measure the strength of a relationship between different variables such as customer satisfaction with product quality and price of product.

- **Manufacturing Sector** – Regression analysis is used to evaluate the relationship of variables that determine to define a better engine to provide better performance. An important application of regression analysis in manufacturing sector is to estimate the cost of product. In manufacturing industries, an accurate cost prediction during a new product development process is most important factor for manufacturing firms to survive in this competition era.

- **Sales and Promotion Sector** – Regression analysis is used to analyses promotions on sales of product. It is used to find the relationship between the amount spent on advertising on a product and determines the amount of its sales. It finds the return on investment such as a company want to find the amount that have invested in marketing of particular products or brands and sales of that product.

- **Medical Sector** – Regression analysis is an important statistical method for the analysis of medical data. It enables the identification and characterization of relationships among multiple factors in online public health data. It is used for prediction and clarification can both be appropriate for public health data analysis for better understanding of public health outcomes. It is also used to forecast the different combination of medicines to prepare generic medicines for diseases.

5.7 TYPES OF REGRESSION ANALYSIS

There are three different types of regression analysis as follows:

- ➢ Simple Linear Regression
- ➢ Multiple Linear Regression
- ➢ Non-Linear Regression

SIMPLE LINEAR REGRESSION
MULTIPLE LINEAR REGRESSION
NON-LINEAR REGRESSION

5.7.1 Simple Linear Regression

Simple linear regression analysis is a statistical tool for finding the best relationship between one independent (predictor or explanatory) variable and one dependent (response, outcome) variable which is continuous in nature. Linear Regression is a predictive model used for finding the linear relationship between a dependent variable and one or more independent variables. This relationship represents how an input variable is related to the output variable and how it is represented by a straight line.

The simple linear regression model is expressed by using the following equation:

Y = a + bX + □

Here,

Y = Dependent variable
a = Intercept
b = Slop
X = Independent variable
□ = Error (residual)

Example : Obtain the equation of the lines of regression from the following data. Also estimate the value of Y for X = 28.

X	10	20	30	40	50	60	70	80
Y	32	48	59	64	70	67	78	82

Solution:

X	**Y**	**XY**	**X^2**	**Y^2**
10	32	320	100	1024
20	48	960	400	2304
30	59	1770	900	3481
40	64	2560	1600	4096
50	70	3500	2500	4900
60	67	4020	3600	4489
70	78	5460	4900	6084
80	82	6560	6400	6724
ΣX = 360	**ΣY = 500**	**ΣXY = 25150**	**ΣX^2 = 20400**	**ΣY^2 = 33102**

Now, we calculate

$$\overline{\mathrm{X}} = \frac{\sum x}{n} = \frac{360}{8} = 45$$

$$\overline{\mathrm{Y}} = \frac{\sum y}{n} = \frac{500}{8} = 62.5$$

$$SSxy = \sum xy - \frac{(\sum x)(\sum y)}{n} = 25150 - \frac{(360)(500)}{8} = 25150 - 22500 = 2650$$

$$SSxx = \sum x^2 - \frac{(\sum x^2)}{n} = 20400 - \frac{(360)^2}{8} = 20400 - 16200 = 4200$$

$$SSyy = \sum y^2 - \frac{(\sum y^2)}{n} = 33102 - \frac{(500)^2}{8} = 33102 - 31250 = 1852$$

$$b = \frac{SSxy}{SSxx} = \frac{2650}{4200} = 0.63095$$

$$a = \overline{\mathrm{Y}} - b\,\overline{\mathrm{X}} = 62.5 - (0.63095)(45) = 62.5 - 28.39 = 34.11$$

So, we get the following equation for regression line.

Y = 34.22 + 0.63095 X

Now, we can use this equation for prediction. So, we can predict the value of Y, when X = 28.

So, Y = 34.22 + 0.63095 X = 34.22 + 0.63095 (28) = 34.22 + 7.66 = 51.77

From the above example, we get the predicted value of Y is 51.77 when the value of X is 28.

Now we will calculate the intercept and slope using Python.

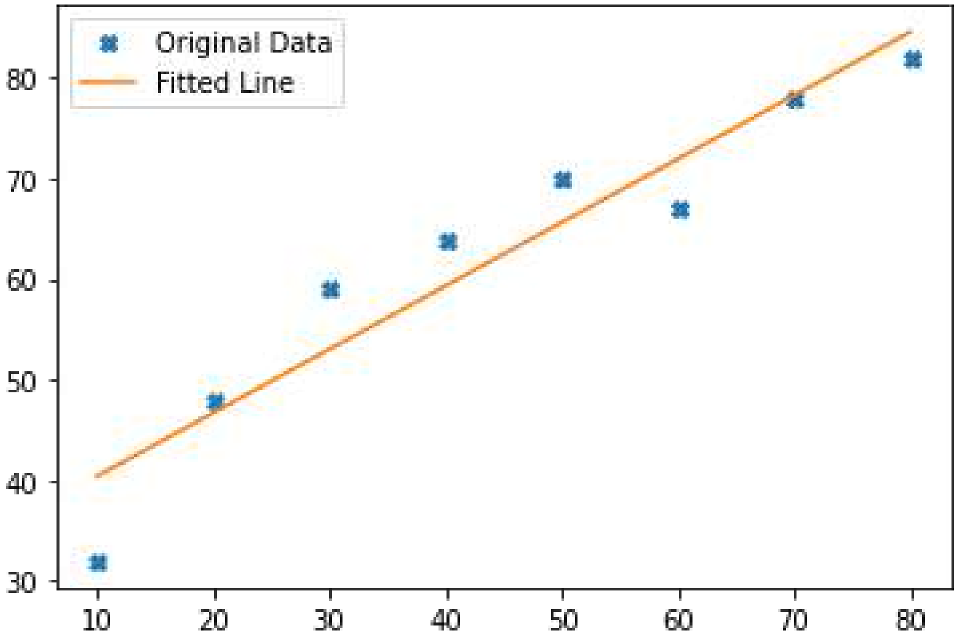
The below code will calculate the intercept and slope and to get the equation of the line of regression. It will also estimate the value of Y for X = 28.

```
# Importing library
import matplotlib.pyplot as plt
from scipy import stats

# Data
x = [10, 20, 30, 40, 50, 60, 70, 80]
y = [32, 48, 59, 64, 70, 67, 78, 82]

# Calculation slop and intercept
slope, intercept, r, p, std_err = stats.linregress(x, y)
print("Slop =", slope)
print("Intercept =", intercept)
print("R value =", r)
print("P value =", p)
print("Standard Error =", std_err)

# Function
def reglinefun(x):
  return intercept + slope * x

# Line
line = list(map(reglinefun, x))

# Plotting
```

```
plt.plot(x, y, 'X', label="Original Data")
plt.plot(x, line, label="Fitted Line")
plt.legend()
plt.show()

# Prediction
predict = reglinefun(28)
print("Predicted value =", predict)
```

The above code will give the following result :

```
Slop = 0.6309523809523809
Intercept = 34.10714285714286
R value = 0.9501687384314103
P value = 0.00029790068633099245
Standard Error = 0.08450983564345486
```



From the above code we can get the estimated value of Y is 51.77 when X = 28.

```
Predicted value = 51.773809523809526
```

5.8 SUMMARY

The students will learn many things related to basic statistics in this module and they will be able to calculate the measures of statistics such as correlation and regression. They will also able to perform the various statistical analysis using Python.

- Ability to understand the correlation in the statistics.

- Ability to understand the various types of correlation such as positive, negative, zero, simple, partial, multiple, linear and non-linear correlation.
- Ability to do understand the methods of studying correlation using scatter diagram.
- Ability to calculate the Karl Pearson correlation coefficient and Spearman rank correlation coefficient.
- Ability to understand the regression analysis and applications of regression analysis.
- Ability to obtain the line of regression and predication using simple linear regression.

5.9 PRACTICE QUESTIONS

Short Answer:

1. What is correlation?
2. Define positive and negative correlation.
3. Define simple correlation and multiple correlation.
4. What is partial correlation?
5. Define linear and non-linear correlation
6. What is regression?
7. List types of regression.

Long Answer:

1. What is correlation? Explain types of correlation.
2. Explain scatter diagram for correlation.
3. Explain Karl Pearson correlation coefficient with example.
4. Explain Spearman Rank correlation coefficient with example.
5. Explain applications of regression analysis.
6. Explain simple linear regression with example.

PRACTICALS

1. Find the correlation between height of father and son (in cm) of the following.

Height of Father	65	66	67	67	68	69	71	73
Height of Son	64	68	65	69	71	70	69	71

2. Find the correlation between marks obtained by eight students in physics, chemistry and biology.

Physics	75	56	70	82	64	78	49	59
Chemistry	80	67	67	75	70	60	55	68
Biology	67	64	68	77	72	73	58	62

3. Draw the scatter plot of age of father and daughter (in year) of the following.

Age of Father	40	45	48	50	51	54	58	60
Age of Daughter	12	13	18	20	22	26	28	32

4. Calculate the Karl Pearson correlation coefficient of the following:

Price	10	20	30	40	50	60	70
Supply	8	7	13	15	11	18	20

5. Calculate the Karl Pearson correlation coefficient of the following:

Age of Husband	25	27	28	30	30	31	34	38
Age of Wife	23	26	27	29	26	28	34	36

6. Calculate the Spearman Rank correlation coefficient of the following:

X	65	66	67	67	68	69	71	73
Y	64	68	65	69	71	70	69	71

7. Calculate the Spearman Rank correlation coefficient of the following:

X	4	3	8	7	1	5	2	6
Y	6	2	7	5	3	4	1	8

8. Obtain the equation of line of regression from the following data.

X	65	66	67	67	68	69	71	73
Y	64	68	65	69	71	70	69	71

9. Obtain the line of regression equation age and blood pressure of the following and predict the value for blood pressure when age is 50 year.

Age	42	55	61	38	68	74	64	46	58	70
Blood Pressure	126	140	148	121	145	160	155	130	142	156

REFERENCES

Books

1. Gupta, S.C. and Kapoor, V.K.: Fundamentals of Mathematical Statistics, Sultan Chand & Sons, New Delhi, 11th Ed
2. Hastie, Trevor, et al. The Elements of Statistical Learning, Springer
3. Ross, S.M.: Introduction to Probability and Statistics for Engineers and Scientists, Academic Press
4. Papoulis, A. and Pillai, S.U.: Probability, Random Variables and Stochastic Processes, McGraw Hill

Web References

1. https://www.geeksforgeeks.org

2. https://www.tutorialspoint.com
3. https://www.w3schools.com
4. https://pandas.pydata.org
5. https://pbpython.com
6. https://www.statisticshowto.com
7. https://realpython.com

B.Sc.(DATA SCIENCE)

SEMESTER-II

STATISTICAL FOUNDATION

UNIT VIII: STATISTICAL INFERENCE

STRUCTURE

8.0 OBJECTIVES

a. To elaborate Statistical Inference
b. To discuss Random variables
c. To discuss and Probability and Density Functions
d. To explore Moments in Python

8.6 INTRODUCTION TO STATISTICAL INTERFERENCE

In python, statistical inference is used to draw and infer conclusions from set of values in given dataset. Initially, some random samples are used and extracted from given population which is used to describe and draw relevant inferences (conclusions) for entire population (Vondrejc, 2019). There are numerous Inferential Statistics available which can be used along with Python and are named as:

a. Z Test and Z Scores
b. t-Test
c. F-Tests
d. Correlation Coefficients
e. Chi square

8.1.1 Need of Preliminary Libraries

Distinct preliminary set of libraries is required to get imported in Python when one wants to work with Arrays, Data frames and different tools for performing statistical analysis. Numpy and Pandas come under this category. These both are highly important and different packages where Numpy can be used to perform distinct operations on Arrays whereas another platform Pandas is used to carry different operations on Data frames. These two libraries can be imported as:

```
import numpy as np

import pandas as pd
```

8.1.2 Z Scores and Z-test

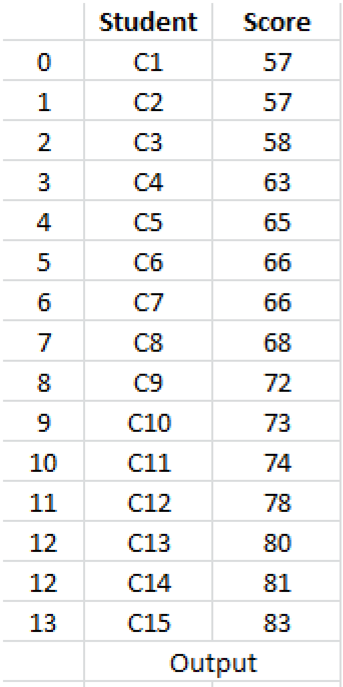
- **Z Scores**

Z Scores computes the probability of score which are calculated from normal distribution. This helps in comparing scores for two or more given normal distributions

- **Z Value**
- **Importing Dataset**

Here, dataset which contains exam scores for some of the students has been imported:

```
Z_scoresdata=pd. Read_excel(C:/Users/user/Desktop/Datasets/marks_Score.xls”)

Z_scoresdata
```



	Student	Score
0	C1	57
1	C2	57
2	C3	58
3	C4	63
4	C5	65
5	C6	66
6	C7	66
7	C8	68
8	C9	72
9	C10	73
10	C11	74
11	C12	78
12	C13	80
12	C14	81
13	C15	83
	Output	

- **Z Score Calculation**

Following is the code by which Z scores can be calculated. Here, Z Scores have been calculated for a column with name ‘Score’ column of Z Score dataset. The function ‘ddof’ can be used to alter the divisor sum of squares of mean sample. Initially, ddof holds 0 but for std we can use ddof=1

```
Z_scoredata[‘scores_zScore’]=(z_scoresdata[‘score’]-z_scoredata[‘score’].mean())/z
_scoredata[‘Score’].std(ddof=1)
```

- **Z Score calculation in Python—**

```
import numpy as np

from scipy import stats

Arayr1= [[19,3,6,2,35],

[49,11,12,35,5]]

Array2 = [[49,11,11,34,5],

[12,10,10,34,22]]

Print (“array1:”, array1)

Print (“\narray2:”, array2)

Print (“z score for array1:, stats,zscore(array1))
```

```
Print("\nz score for array1:", stats.zscore(array1,axis=1))
```

- **Calculating Percentage**

For above considered example, percentage of people scored more than 70 can be calculated. Mean and Standard Deviation is to be taken for calculating area under curve. The code that can be used to find area under curve can be depicted as:

```
Cutoff= 70

Print(1-(scipy.stats.norm(70.5, 7.06).cdf(70)))
```

Where 7.06 is standard deviation and 70.5 is mean for finding percentage.

- **Z Test**

This test determines whether the given two datasets are similar or not.

- **Importing dataset**

By taking initial dataset from population and some random samples of this dataset, importing dataset is initial set which can be done in following way:

```
HghtDataPop=pd.read_csv("C:/User/ABC/Datasets/Heightof100ppl.csv")
```

Now, by taking random sample of above mentioned dataset:

```
HghtdataSample= pd.read_excel("C:/User/ABC/Datasets/HeightdataSample.xls")
```

- **Package to be imported for applying Z Test-**

```
from statesmodel.stats.weightstats import ztest
```

- **How to run Z Test**

```
Ztest(A2,b2=None,value=mean1)
```

- **Code for implementing Z test in Python**

```
    # considering an array of 55 numbers with mean 105 and std dev 16
import math
import numpy as np
from numpy.random import randn
from statsmodels.stats.weightstats import ztest
    mean = 105
    std_dev = 16/math.sqrt(55)
    alpha =0.01
    nul_mean =101
    data = sd_iq*random(55)+mean
    # printing mean and std_dev
    print('mean=%f std_dev=%f' % (np1.mean(data), np1.std(data)))
    # Z test
    ztest_Score, p1_value= ztest(data1,value = null_mean, alternative='larger')
      if(p1_value <  alpha):
    print("Rejecting Null Hypothesis")
    else:
print("Failed to Reject Null Hypothesis")
```

8.1.3 t Test

t-Test is used to evaluate similarity level of groups. This can also be done by using Z test, the difference is Z Test is better to apply for the case sample size greater than 30 whereas t-Test is used for sample size less than 30.

- **Importing package for applying t-Test**

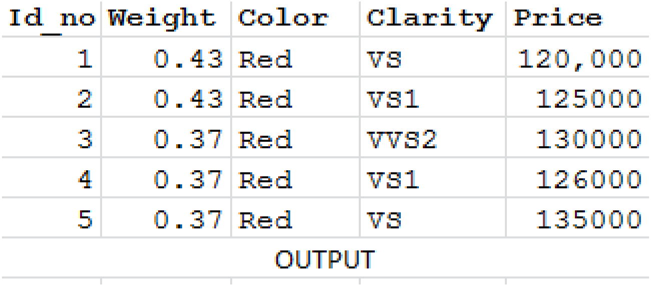
```
Import scipy.stats as stats
```

- **Importing Dataset**

For instance, considering hypothetical *Rubyjewellery* dataset which contains all necessary information of Rubyjewellery, which is sold in a store of jewellery.

```
Rubyjewellery-pd.read_excel(“C:/Users/Login/Rubyjewellery.xls”)
```

```
Rubyjewellery
```



Id_no	Weight	Color	Clarity	Price
1	0.43	Red	VS	120,000
2	0.43	Red	VS1	125000
3	0.37	Red	VVS2	130000
4	0.37	Red	VS1	126000
5	0.37	Red	VS	135000

OUTPUT

- **Code for t-test in Python**

```
from numpy.random import randn
from scipy.stats import ttest_ind
seed(1)
data_1 = 4 * randn(105) + 51
data_2 =4 * randn(105) + 52
# comparing samples
Stat1, p1 = t_test_ind(data_1, data_2)
print('t=%f, p1=%f' % (sta1t, p1))
```

8.1.4 F –test

F test is a statistical test which has F-distribution under null hypothesis. F test uses F statistics which actually is the ratio of two different variances; hence, they use F Distribution. Such test is applicable in comparing two regression models to check for statistical significance. Moreover, unlike Z and t-Test, where comparison is done on two datasets for inferring results, in F Test compares two variances.

8.2 CONCEPT OF RANDOM VARIABLE

A random variable in Python Statistical Inference is a variable whose values are possibly the outcomes of random process (Julier, 2004). There are two main categories of random variables such as discrete and continuous variables.

8.2.1 Discrete Random Variable

The variable which takes only one countable number of different numbers and values and also which can be quantified. For instance, ***A*** to be possible number which comes up when rolling a fair dice. ***A*** can take different values: [1, 2, 3, 4, 5, 6]. Hence, is a discrete random variable.

- **Probability Distribution:**

It is associated with list of probabilities along with each possible value. It is also refereed as *probability function.* Mathematically, it can be interpreted as, suppose there is random variable ***Y*** which may take ***n*** different values with probability that **$Y = y_i$** can be defined as **$P(Y = y_i) = p_i$.** In this case, probability pi need to satisfy following conditions:

1. $0 \leq pi \leq 1$ for every i

2. p1+p2+p3+…..+ pn = 1

Bernoulli distribution, Binomial distribution and Poisson distribution can be considered as best examples of Discrete Probability Distributions.

8.2.2 Continuous Random Variable

The variable which can take infinite number of values is continuous random variable. For example, Y is taken as variable to store the height of players in a group. Such variables are interpreted as continuous random variables over interval of class, such variables are inferred by using area under curve or by integrals. Distribution of these variables is also interpreted as probability distribution functions. It can be demonstrated by using p(x), which must also satisfy following condition:

1. When curve is not having negative values p(x)>0
2. When area under curve =1

A curve that meets such requirements is taken as a density curve. Normal distribution, Exponential distribution and Beta distribution are common examples of Continuous Probability Distributions .

8.2.3 Importing some important distributions in Python

- **Uniform Distribution**

In this distribution all the outcomes of probability distribution are equally like. In Python, this distribution can be visualized by importing *uniform function* from *scipy.stats* module as:

```
      # import uniform distribution
    from scipy.stats import uniform
      # random numbers
      N=1000
      Start=5
      Width=10
Data_uniform= uniform.rvs (size=n, loca=start, scale=width)
```

- **Code for uniform distribution in Python can be written as:**

```
From numpy import random
A=random.uniform(size(3,3))
Print(a)
```

- **Normal Distribution**

It is also considered as bell curve and occurs quite naturally in various situations. It is also called as Gaussian distribution and most commonly applicable in statistical inference. It poses bell shaped density curve and is defined by mean (μ) and standard deviation (σ).Here, the density curve is symmetric and centered about its mean. In Python, it can be imported as:

```
from scipy.stats import norm

Data1_normal = norm.rvs (size=1000, loca=0, scale=1)
```

Where *scipy.stats* is a relative module, *norm.rvs ()* method, *loc* is mean of distribution, *scale* is standard deviation and *size* is total number of random variables. Random normal distribution of size (3*3) can be generated by:

```
From numpy import random

A= random.normal(size=(3,3))

Print(a)
```

- **Gamma Distribution**

It belongs to two-parametric family under continuous probability distribution. It is used rarely. Most of the times it is used in modeling in engineering areas where the variables are always positive with skewed results. This can be imported in Python as:

```
from scipy.stats import gamma

Data1_gamma = gamma.rvs (x = 10, size_s= 1000)
```

- **Exponential Distribution**

This distribution is used to describe the time interval between different events in a process of Poisson point i.e., the process in which any event occurs in a continuous and independent way with constant and average piece of rate. In python it can be imported as:

```
from sipy.stats import expon

Data1_expon = expon.rvs (scale=1, loca=0, size = 1000)
```

- **Poisson Distribution**

This distribution is used to show the event occurrence within specific period of time. For instance, users visited a website in a particular interval is a Poisson Process in itself. An event can occur n number of times and average number of events in an particular interval is depicted as lambda λ. Poisson distribution can be imported in Python as:

```
from scipy.stats import poisson

Data_poisson = poisson.rvs (mu= 2, size, 1000)
```

- **Bernoulli Distribution**

In this distribution, only two outcomes are expected to come such as failure or success and the probability of each success and failure is entirely same for all number of trials and is known as Binomial distribution. In Python, this distribution can be added as:

```
from scipy.stats import binom

Data1_binom = binom.rvs (n = 20, p = 0.5, size = 1000)
```

8.3 PROBABILITY MASS FUNCTION

This function describes the probability which is associated with given random number y (Vondrejc, 2019). The function is depicted as P(y).

Example, when rolling a die infinitely and looking on proportion 1, then 2 and so on. The random variable can correspond to outcome of a dice roll. So, random variable can take following discrete values i.e., 1, 2, 3, 4, 5 or 6. The main aim of the probability mass function is to describe possibility/probability of every possible value. In this example, there is probability to get 1, 2 and so on. For example in rolling of dice, there is equal probability of getting any number, which can be written as:

$$P(y = 1) = P\ (y = 2)$$

$$=P(y=3)$$

$$=P(y=4)$$

$$=P(y=5)$$

$$=P(y=6)$$

So, 1/6.

In this way, distribution demonstrates similar probability for every possible value, and called as uniform distribution. Probability mass function looks like:

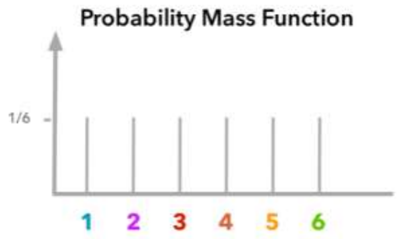


Fig. 1 Probability mass function (for dice)

Here, x-axis depicts outcome and y axis depicts probability. The code that can demonstrate this in more clear way and can be written as:

```
Numb_throws=1000

Outcomes=np.zero(numb_throws)

For I in range(numb_throws);

#when rolling a dice

Outcome=np.random.choiceA(‘1’,’2’,’3’,’4’,’5’,’6’)

Outcome[i]=outcome

val.cnt=np.uniquee(outcomes, return.count=True)

prop=cnt/len(outcomes)

# after rolling dice 1000 times, plotting of results

Plt.bar(val,propi)

Plt.xlabel(“outcome”)

Plt.ylabel(“probability”)

Plt.show()

Plt.close()
```

8.3.1 Properties of Probability Mass Function

Probability Mass function, if and only if,

$$\forall x \in x, 0 \leq P(x) \leq 1$$

Here, symbol ∀ depicts for any or for all. This represents, for every x within range of x (in case of rolling dice, set of possible values are 1, 2,3, 4,5 and 6). The probability of each outcome can vary between 0 and 1. Here, 0 represents event has not occurred and 1 represents event has occurred. In particular case of dice, the probability of each value is 1/6 i.e., between 0 and 1.

8.3.2 Probability Density Function

All the discrete variables cannot take infinite values at a certain period of time. Still, it needs to state probability which is associated with all possible outcomes. The equivalence of probability mass function with continuous variable is known as **Probability Density Function.**

- **Important property of Probability Density Function**

$$\forall x \in x, p(x) \geq 0$$

Here, p(x) needs not to be less than 1, because it is not corresponding to probability.

8.4 MATHEMATICAL EXPECTATION

Probability describes happening of certain events and occurrence of particular event depends upon previous experience. The Mathematical Expectation represents all those events which are almost impossible for any experiment. Probability for that particular event remains 0 and Probability of an event remains 1 where both of the numerator and denominator remain equal.

Mathematical Expectation is also referred as expected value, that uses the notation E[X]. This can be computed as probability weighted sum of all possible values which can be drawn by as follows:

E[X]= sum (x1*p1, x2*p2*………*xnpn)

Following is the code with 6-element vector and calculation of mean:

```
From numpy import array

From numpy import mean

M1 = array_example ([1,2,3,4,5,6])

Print (M1)

Result = mean (M1)

Print (result)
```

```
[1 2 3 4 5 6]

3.5
```

- **Properties of mathematical Expectation-**

1. For two variables X and Y, total sum of these two variables is equals to sum of mathemeatical expectationof X and Y respectively i.e., E(X+Y)= E(x)+E(Y).
2. For two independent variables, mathematical expectation would be the product of those considered variables.
3. Mathematical Expectation of constant sum and considered function of any random variable is equals to sum of that constant of function of given random variable.
4. The mathematical expectation of sum and product of a constant and function of any random variable and another constant is all equals to sum of product of that constant and mathematical expectation of given function and random variable and constant.
5. Linear combination of all random variables and a constant is all equals to sum of product of n constants and mathematical expectation of all n numbers.

8.5 MOMENTS

Moments are used to calculate the n moments about mean for given sample i.e., all the array elements with particular axis of that given array (Fan, 2016). In Python, Moments can be calculated as:

```
From scipy import stats

import numpy as np

arry = np.array ([1,27, 31, 21, 13, 9],[12,8,4,8,7,10])

print (“0th moments is :”, stats.moment(arry,moment=0))
```

8.5.1 Moment generating function and characteristic function

```
  from random import choice
  import matplotlib.pyplot as plt
  import numpy as np
  exp_value = lambda values: sum(values) / len(values)
  std_deviation = lambda values, exp_value: np.sqrt(sum([(v - exp_value)**2 for v in values])  /
  len(values))

muu, sigmaa = 40, 1
  population = np.random.normal(muu, sigmaa, 100000)
  mean = expected_value(population)
  print(
'''population: Expected_value: {0} Standard_deviation: {1}
     '''.format(mean, standard_deviation(population, mean))
  )
  plt.hist(population, 70, density=True)
  plt.show()

  randomly_select_items = [choice(population) for _ in range(63)]
  mean = exp_value(randomly_selected_items)
  s_d = std_deviation(randomly_selected_items, mean)
  print(format(mean, s_d))
plt.hist(randomly_selected_items, 10, density=True)
  plt.show()
  xsss = np.arange(63, 44, 0.001)
  actual_ys = norm.pdf(xs, muu, sigmaa)
  ys = norm.pdf(xs, mean, s_d)
  plt.plot(xs, actual_ys, label='actual_population_distribution')
  plt.plot(xs, ys, label='sample_distribution')
  plt.legend()
  plt.show()
```

8.6 PRACTICE QUESTIONS

Q1. What is Statistical Inference?

Q2. What are different Inferential Statistics available can be used along with Python? Q3. What are random variable?

Q4. Explain difference between F-test and t-test along with suitable example.

Q5. How Z scores and Z test is calculated. Demonstrate with suitable dataset.

Q6. How moments can be imported and calculated in Python.

Q7. Explain different properties of Mathematical Expectation.

Q8. Explain different distributions. Which distribution is frequently used and why?

REFERENCES

[1] Vondrejc, J. & Matthies, H. G. (2019) “Accurate Computation of Conditional Expectation for Highly Nonlinear Problems”, ,SIAM/ASA Journal on Uncertainty Quantification 7(4), pp. 1349-1368.

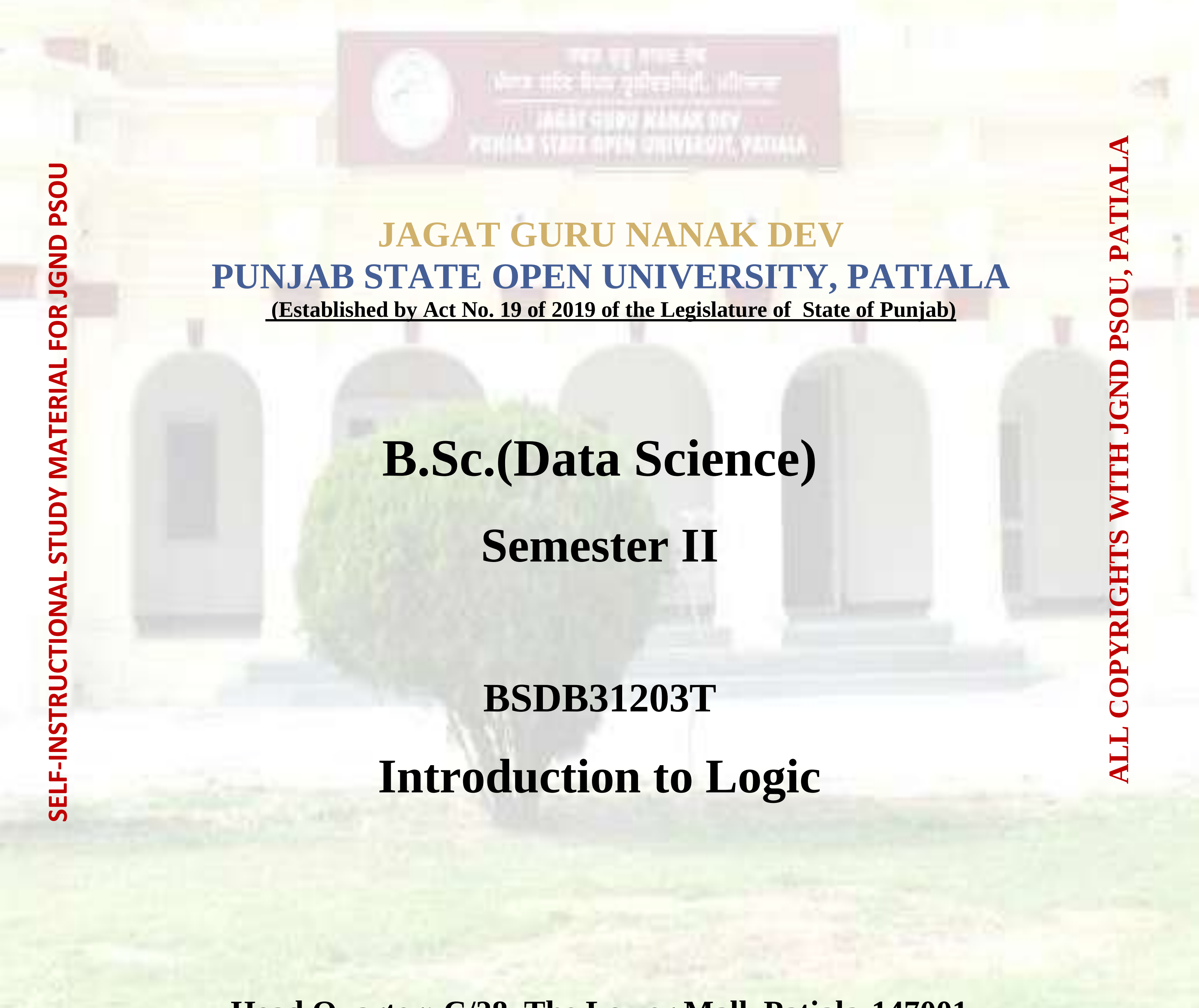
[2] Julier, S. J. & Uhlmann, J. K. (2004) “Unscented Filtering and Non linear Estimation”, Proceedings of IEEE, 92, pp. 401-422.

[3] Ale, A. , Kirk, P. & & Stumpf, M.P. (2013) “A General moment expansion method for stochastic kinetic models”, Journal of Chemistry Physics, 138 (17).

[4] Fan, S. , Geissmann, Q. , Lakatos, E. & Lukauskas, S. (2016) “Means: Python package for Moment Expansion Approximation, Inference and Simulation”, Bioinformatics, 32 (18).

ਜਗਤ ਗੁਰੂ ਨਾਨਕ ਦੇਵ
ਪੰਜਾਬ ਸਟੇਟ ਓਪਨ ਯੂਨੀਵਰਸਿਟੀ
ਪਟਿਆਲਾ

The Motto of Our University
(SEWA)
SKILL ENHANCEMENT
EMPLOYABILITY
WISDOM
ACCESSIBILITY



SELF-INSTRUCTIONAL STUDY MATERIAL FOR JGND PSOU

JAGAT GURU NANAK DEV
PUNJAB STATE OPEN UNIVERSITY, PATIALA
(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

B.Sc.(Data Science)

Semester II

BSDB31203T

Introduction to Logic

Head Quarter: C/28, The Lower Mall, Patiala-147001
Website: www.psou.ac.in

The Study Material has been prepared exclusively under the guidance of Jagat Guru Nanak Dev Punjab State Open University, Patiala, as per the syllabi prepared by Committee of experts and approved by the Academic Council.

COURSE COORDINATOR AND EDITOR:

Dr. Amitoj Singh
Associate Professor
School of Sciences and Emerging Technologies
Jagat Guru Nanak Dev Punjab State Open University

LIST OF CONSULTANTS/ CONTRIBUTORS

Sr. No.	Name
1	Dr. Amanpreet kaur
2	Er. Harpreet Kaur

JAGAT GURU NANAK DEV PUNJAB STATE OPEN UNIVERSITY, PATIALA
(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

PREFACE

Jagat Guru Nanak Dev Punjab State Open University, Patiala was established in December 2019 by Act 19 of the Legislature of State of Punjab. It is the first and only Open University of the State, entrusted with the responsibility of making higher education accessible to all, especially to those sections of society who do not have the means, time or opportunity to pursue regular education.

In keeping with the nature of an Open University, this University provides a flexible education system to suit every need. The time given to complete a programme is double the duration of a regular mode programme. Well-designed study material has been prepared in consultation with experts in their respective fields.

The University offers programmes which have been designed to provide relevant, skill-based and employability-enhancing education. The study material provided in this booklet is self-instructional, with self-assessment exercises, and recommendations for further readings. The syllabus has been divided in sections, and provided as units for simplification.

The University has a network of 10 Learner Support Centres/Study Centres, to enable students to make use of reading facilities, and for curriculum-based counselling and practicals. We, at the University, welcome you to be a part of this instituition of knowledge.

Prof. Anita Gill
Dean Academic Affairs

B.Sc. (Data Science)
Discipline Specific Elective (DSE)
Semester II
BSDB31203T: Introduction to Logic

Total Marks: 100
External Marks: 70
Internal Marks: 30
Credits: 4
Pass Percentage: 35%

Objective: This course will enable students to understand the fundamentals of logic. Students will be able to infer the concept Logic and inference.

INSTRUCTIONS FOR THE PAPER SETTER/EXAMINER

1. The syllabus prescribed should be strictly adhered to.
2. The question paper will consist of three sections: A, B, and C. Sections A and B will have four questions from the respective sections of the syllabus and will carry 10 marks each. The candidates will attempt two questions from each section.
3. Section C will have fifteen short answer questions covering the entire syllabus. Each question will carry 3 marks. Candidates will attempt any ten questions from this section.
4. The examiner shall give a clear instruction to the candidates to attempt questions only at one place and only once. Second or subsequent attempts, unless the earlier ones have been crossed out, shall not be evaluated.
5. The duration of each paper will be three hours.

INSTRUCTIONS FOR THE CANDIDATES

Candidates are required to attempt any two questions each from the sections A and B of the question paper and any ten short questions from Section C. They have to attempt questions only at one place and only once. Second or subsequent attempts, unless the earlier ones have been crossed out, shall not be evaluated.

Section A

Unit- I: Definition, Nature and Significance of Logic; Nature of Implication; Truth and Validity; Laws of Thought, Nature of Proposition; Traditional classification of Propositions; Quality, Quantity and Distribution of Terms.

Unit-II: Immediate Inferences: Conversion, Obversion & Contraposition; Square of Opposition; Mediate Inferences; Categorical Syllogism: Rules, Fallacies & Validity through Venn-Diagrammes.

Unit-III: Truth-functional Logic: Truth-functional compound statements; Negation, Conjunction, Disjunction and Implication. Validity & Invalidity through Truth-table Method; Statement Forms: Tautology, Contradictory and Contingent.

Unit IV: Propositional Logic: Syntax of Propositional Logic, Logical Connectives: Truth Tables, Validity, Consistency, Logical Equivalence. Conjunctive and Disjunctive Normal Forms

Section B

Unit V: **Predicate Logic:** Quantifiers, Translating simple syllogistic sentences to Predicate logic, Semantics of Predicate Logic, Conversion to Clausal form Resolution, Unification, Truth, satisfiability, validity in Predicate Logic.

Unit VI: **Fuzzy Logic**: Basic concepts of fuzzy set theory – operations of fuzzy sets – properties of fuzzy sets – Crisp relations – Fuzzy relational equations – operations on fuzzy relations – fuzzy systems,

Unit VII: **Prolog:** Introduction, Variables and atoms, Facts and predicates, data types, goal finding, Clauses, Central Idea of Prolog, Execution of Prolog Programs, backtracking, simple object, compound objects,

Unit VIII: Prolog Operations: Arithmetic Operators, Program Termination, Use of cut and fail predicates, Satisfiability: Use Unification, recursion, lists, simple input/output, dynamic database.

Suggested Readings

1. Cohen & Nagal : Introduction to Logic and Scientific Method, Macmillan Publishing Company, London, 1934
2. Copi, Cohen, Jetli : Introduction to Logic, Pearson Education, 12th Edition, 2013
3. Timothy J.Ross, Fuzzy logic with Engineering Applications, 3rd Ed. McGraw Hill, 2011
4. Ivan Bratko, PROLOG Programming For Artificial Intelligence, Addison Wesley, 2011

JAGAT GURU NANAK DEV PUNJAB STATE OPEN UNIVERSITY, PATIALA
(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

BSDB31203T: INTRODUCTION TO LOGIC

COURSE COORDINATOR AND EDITOR: DR. AMITOJ SINGH

B.SC.(DATA SCIENCE)

SEMESTER-II

INTRODUCTION TO LOGIC

UNIT 1: DEFINITION, NATURE AND SIGNIFICANCE OF LOGIC

STRUCTURE

1.0 OBJECTIVE

To understand the followings:

- Nature of Implication; Truth and Validity; Laws of Thought,
- Nature of Proposition;
- Traditional classification of Propositions; Quality, Quantity and Distribution of Terms.

1.1 INTRODUCTION

As per the definition of the Oxford dictionary, is **"the reasoning conducted or assessed according to strict principles and validity"**.

Logic can be defined as a scientific study of the process of reasoning and the system of rules and procedures that help in the reasoning process.

Logic is the study of the methods and principles used in distinguishing correct from incorrect reasoning.

The philosophical definition is that logic is a description of how one should think. In the context of AI, logic is "formal," which means it resembles math in its clarity and lack of ambiguity.

One of the prime activities of human intelligence is reasoning involves construction, organisation and manipulation of the statements to arrive at new conclusions. Basically, the logic process takes in some information and produces some outputs. There exist some basic laws that help in theorem proving. There are few laws associated with standard formulas help to extract new information from the existing ones.

Logic known as the validation or proof that works behind any reason that provided. It is simply the ‘dialectics behind reasoning’. It was essential to include logic in Artificial Intelligence because we want our system to think and act humanly that should have calibre to take any decision based on the recent circumstances or situation. If we talk about normal human behaviour, then a decision is made by choosing an option from the various available options. There are various valid reasons for selecting or rejecting a one option. So, our artificial system (agent) should also work in this manner.

1.2 TYPES OF LOGICS IN ARTIFICIAL INTELLIGENCE

There are two types of Logics in Artificial Intelligence (AI):-

a. Deductive logic
b. Inductive logic

a) Deductive Logic

The complete evidence is assuming about the truth or reality of the conclusion that is made in the deductive logic. The system (agent) uses a unique and perfect premises that pursue to a specific result or conclusion. A specific example for this logic can be view in an expert (simulate) system designed to prescribe medicines to the patient or survivor. The system (agent) provides the complete evidence about the medicines suggested by the system, like the particular medicines are prescribed to a patient or to a person because the patient has specific type of symptoms.

2) Inductive Logic

This type of Logic considers as a bottom up and the reasoning is done through this approach. It means is that the system(agent) takes specific or unique information after that generalizes it for the cause of complete understanding. Let's take an example of this, that can be view in the Natural Language Processing (NLP) by a system (agent) in which it sums up the words according to their given categories, i.e., verb, noun article, etc., and then deduce the meaning or result of that sentence or paragraph.

1.3 NATURE AND SIGNIFICANCE OF THE LOGICS

The most essential in logical Artificial Intelligence (AI) is John McCarthy. McCarthy was one of the founders of Artificial Intelligence (AI), as well as consistently supported a Research Methodology (RM) that uses logical or thinking techniques to approbate the reasoning problems that Artificial Intelligence (AI) needs to solve.

The motivation for using logic is that, these implementations do not simply and directly use logical reasoning techniques like theorem. A logical formalization helps us to understand the reasoning problem of any situation by itself. Without an understanding, the claim is that what the reasoning problems are. It would not be feasible to implement their solutions. Credible, as this Platonic argument may seem that it is in fact controversial in the context of Artificial Intelligence (AI). An alternative methodology would seek to learn as well as evolve the desired behaviours. This methodology would produce might be too complex to characterize or to understand at a conceptual level.

Now it has been cleared that McCarthy thought of Artificial Intelligence (AI) and his methodology is related to overlapping to a large extent with traditional philosophy. But in addition, that the need to inform the design of programs capable of manifesting Artificial Intelligence (AI). This idea is not uncongenial to some philosophers. In practice, the actual theories that have emerged from McCarthy's methodology while practicing and are influenced most strongly by work in philosophical logic. The research tradition in logical AI represents a more or less direct development of this work along with some changes in emphasis. This review will concentrate on logical AI in relation to philosophical logic, without further comment on relations to philosophy in general or to the feasibility of developing human-level intelligent systems.

1.4 NATURE OF IMPLICATION

Each coin has two sides than AI having some positive as well as some negative impacts. Let us see the negative impact the AI will have on human society:

1. A huge social change that disrupts the way where the human community will occur. Humankind has to be industrious to make their living, but the service of AI, we can do program so that the machine to do a thing for us without using a tool. Human

closeness will be gradually diminishing as AI will replace the need for people to meet face to face for idea exchange.

2. Unemployment is the next negative impact because lots of works will be replaced by machinery. These days various automobile assembly lines have been packed with many machineries; robots that coursing traditional workers to left their jobs. In supermarket, the store clerks will not be required the digital device anymore to take over human labour.

3. Next is Wealth inequality that would be created as the investors of Artificial Intelligence (AI). It will move up the major share of the earnings. the rich and the poor will be widened and this is the gap in between. This is called as "M" shape wealth distribution will be more obvious.

4. New issues surface not only in a social sense but also in Artificial Intelligence (AI) itself as the Artificial Intelligence (AI) is being trained and learned that how to operate the given task can eventually take off to the stage that human has no control, thus creating un-anticipated problems and consequences. It refers to AI's capacity after being loaded with all needed algorithm may automatically function on its own course ignoring the command given by the human controller

5. The human masters who create AI may invent something that is racial bias or egocentrically oriented to harm certain people or things. For instance, the United Nations has voted to limit the spread of nucleus power in fear of its indiscriminative use to destroying humankind or targeting on certain races or region to achieve the goal of domination. AI is possible to target certain race or some programmed objects to accomplish the command of destruction by the programmers, thus creating world disaster.

   Let us see the positive impact the AI will have in the field of healthcare. Artificial Intelligence (AI) shows computers the capacity and capability to learn, reason, and apply logic. Medical Researchers, clinicians, mathematicians, and engineers and Scientists, when they working together that can design an AI. This AI is lined up at medical diagnosis and treatments, thus offering reliable and safe systems of health-care delivery. As a Medical researchers and health professors endeavour to search emerging and efficient ways of treating diseases. The digital computer can assist in analysing, robotic systems can also be created to perform some delicate medical procedures with precision.

   Here, we see the contribution of AI to health care:

1. **Fast and accurate diagnostics:** -IBM's Watson's computer has been used to diagnose with the interesting result or response. In AI, loading the data to the computer will instantly get AI's diagnosis. AI can also provide various ways of treatment for physicians to consider. The procedure is something like this: To load the digital results of physical examination to the computer that will consider all possibilities and

automatically diagnose. By this researcher can get, the patient suffers from some deficiencies and illness. It will provide the valuable treatment to patient.

2. **Socially therapeutic robots**: **-** In AI, Pets are recommended to senior citizens to ease their tension as well as reduce the blood pressure, anxiety, loneliness, and increase social interaction. Therapeutic robots and the socially assistive robot technology help improve the quality of life for seniors and physically challenged.

3. **Reduce errors related to human fatigue**:- In software Companies, Human error at workforce is inevitable and costly, the greater the level of fatigue, the higher the risk of errors occurring. Al technology, however, does not suffer from fatigue or emotional distraction. It saves errors and can accomplish the duty faster and more accurately.

4. **Artificial intelligence-based surgical contribution**: -AI-based surgical procedures have been available for people to choose. Although this AI still needs to be operated by the health professionals, it can complete the work with less damage to the body. For Example, The da Vinci surgical system, a robotic technology allowing surgeons to perform minimally invasive procedures, is available in most of the hospitals now. These systems enable a degree of precision and accuracy far greater than the procedures done manually.

5. **Improved radiology**:-In 1971,the first computed tomography scanners were implemented. The first Magnetic Resonance Imaging (MRI) scan of the human body took place in 1977. Cardiac MRI, Body MRI, and Fetal imaging, became routine in 2000s. The search continues for long time to search new algorithms for detection of specific diseases and to analyse the results of scans.

6. **Virtual presence: -** In the AI era, **t**he virtual presence technology can enable a distant diagnosis of the diseases. The patient does not have to leave his/her bed but using a remote presence robot, doctors can check the patients without actually being there. Health professionals can move around and interact almost as effectively as if they were present. This allows specialists to assist patients who are unable to travel.

1.5 TRUTH AND VALIDITY

There are three essential concepts in Logics that mentioned below

1. Truth: - a property of the statements that they are the cases. Truth is the complete accuracy of whatever was, is, or will be, along with error-proof, dispute or debate, beyond doubt, a final test of right or wrong of people's ideas and beliefs. Various valid arguments contain only **TRUE**, for example:

All men are mortal,

All students are men,

Therefore, all students are mortal.

2. Validity: -a property of arguments, that they have a good and planned structure. Validity is defined as the internal consistency of an argument. That is the conclusion reached consistent and reasonable with the information used to reach that conclusion? Various valid argument contains entirely **FALSE**, for example:

All ten-legged creatures have wings,

All spiders have ten legs,

Therefore, all spiders have wings.

3. Soundness:- a property of both arguments and the statements in them, the argument is valid and all the statement are true. **Sound Argument**: (1) valid, (2) true premisses (obviously the conclusion is true as well by the definition of validity).

The fact that a deductive argument is not valid. It assure us that any of the statements in the argument are TRUE. This fact only tells us that the conclusion must be true if the premisses are true.

1. The variety of valid arguments that can be given as sorted by the truth of premisses and conclusion:

Cases —>	1	2	3	4
Premiss(es)	T	F	T	F
Conclusion	T	T	~~F~~	F

If it were possible to have true premisses and a false conclusion, logic would be useless to prove anything.

2. The variety of **invalid** arguments that can be given as sorted by the truth of premisses and conclusion:

Cases —>	1	2	3	4
Premiss(es)	T	F	T	F
Conclusion	T	T	F	F

That is, all possibilities can be represented; examples are given in the syllabus.

1.6 LAWS OF THOUGHTS IN ARTIFICIAL INTELLIGENCE

One of the Greek philosophers named Aristotle was the first to attempt to codify the right thinking that is undisputable reasoning processes. His famous **syllogisms** provided patterns for argument structures that always gave correct conclusions given correct premises. For example, “Socrates is a man; all men are mortal; Socrates is mortal." These laws of thought were supposed to govern the operation of the mind, and initiated the field of **logic.** The development of formal logic in the late 19th and early 20th centuries, come up with a precise notation for statements about all kinds of things in the world and the relations between them. (Contrast this with ordinary arithmetic notation, which provides mainly for equality and inequality statements about numbers.) By 1965, programs existed that could, given enough time and memory, take a description of a problem in logical notation and find the solution to the problem, if one exists. (If there is no solution, the program might never stop looking for it.) The so-called **logicist** tradition within artificial intelligence hopes to build on such programs to create intelligent systems.

There are two main obstacles to this approach.

1. It is not easy to take informal knowledge and state it in the formal terms required by logical notation. It is particularly working when the knowledge is less than 100% certain.

2. There is a big difference between being able to solve a problem ``in principle" and doing so in practice. Even problems with just a few dozen facts can exhaust the computational resources of any computer unless it has some guidance as to which reasoning steps to try first.

Although both of these obstacles apply to *any* attempt to build computational reasoning systems, they appeared first in the logicist tradition because the power of the representation and reasoning systems are well-defined and fairly well understood.

1.7 NATURE OF PROPOSITION

Proposition means sentences. Propositional logic applies the Boolean logic to convert actual data into a format that is readable to the computer system. For Example, if we say 'It is hot and humid today', the machine (Computer System) won't understand. But if we can create propositional logic for this sentence, then, it would be easy to understand the machine readable, and interpret our message.

Derived from Boolean logic, the heart of propositional logic is the idea that the final output of all propositions is either TRUE or FALSE or It can't be both.

For example, 'Earth is round', the output for this proposition is TRUE. If we say, 'Earth is square', then the output is FALSE. Propositional logic applies to those sentences where the output can only be either TRUE or FALSE. But if we refer to the sentence like 'Some children are lazy' then here we have two possible outputs. This preposition is TRUE and good example for those children who are lazy, but it is FALSE and good example for those children who are not lazy. So, for such sentences/propositions where two or more outputs are possible, propositional logic doesn't apply [1].

1.8 PROPERTIES OF PROPOSITIONS

- **Satisfiable:** A atomic propositional formula is satisfiable if there is an interpretation for which it is true.
- **Tautology:** A propositional formula is valid or a tautology it is true for all possible interpretations.
- **Contradiction**: A propositional formula is contradictory (unsatisfiable) if there is no interpretation for which it is true.
- **Contingent:** A propositional logic can be contingent which means it can be neither a tautology nor a contradiction.

Following are some basic facts about propositional logic [2]:

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and **logical connectives**.
- These connectives are also called logical operators.
- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.
- A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- A proposition formula which is always false is called **Contradiction**.
- A proposition formula which has both true and false values is called
- Statements which are questions, commands, or opinions are not propositions such as "**Where is Rohini**", "**How are you**", "**What is your name**", are not propositions.

1.9 STANDARD THEOREMS OF PROPOSITIONAL LOGIC

Assuming p, q, and r represent the propositions which can be true or false and they are chosen arbitrarily. The list of standard theorems in propositional logic are as follows

1. $p, q \Rightarrow p \land q$

2. $p, p \rightarrow q \Rightarrow q$ (Modus Ponens)

3. $\neg p, p \lor q \Rightarrow q$ (law of disjunctive inference)

4. $\neg q, p \rightarrow q \Rightarrow \neg p$ (Modus Tollens)

5. $p \lor q, p \rightarrow r, q \rightarrow r \Rightarrow r$

6. $p \rightarrow q, q \rightarrow r \Rightarrow p \rightarrow r$ (Chaining)

7. $p, p \rightarrow q, q \rightarrow r \Rightarrow r$ (Modus Ponens & Chaining)

8. $p \lor (q \land \neg q) \Leftrightarrow p$

9. $p \land (q \lor \neg q) \Leftrightarrow p$

10. $p \rightarrow q \Leftrightarrow \neg p \lor q$

11. $\neg(p \rightarrow q) \Leftrightarrow p \land \neg q$

12. $p \leftrightarrow q \Leftrightarrow (p \rightarrow q) \land (q \rightarrow p)$ (Bidirectional elimination)

13. $p \leftrightarrow q \Leftrightarrow (p \land q) \lor (\neg p \rightarrow \neg q)$

14. $p \rightarrow (q \rightarrow r) \Leftrightarrow (p \land q) \rightarrow r$

15. $p \rightarrow q \Leftrightarrow \neg q \rightarrow \neg p$ (Contraposition theorem)

1.10 PREDICATE LOGIC

Where system finds the result either TRUE or FALSE that propositional logic works. But there are many situations that cannot be handled with two options.

Lets take an example:-

All mammals suckle their young ones. Since elephant is a mammal, it suckles its young ones.

These above statements are correct but propositional logic unable to express them. In order to overcome the deficiency, first order logic or predicate logic will work after using three additional notations.

1. Predicates

2. Terms

3. Quantifiers

Now, Predicate is defined as a relation that binds two or more atoms together. Example:- Umang likes aeroplanes is represent as

LIKES (Umang, aeroplanes)

In this example, LIKEs is a predicates that links two atoms like Umang and aeroplanes. Even, We can generalise the predicates

LIKES (x, y) where x and y are variables meaning x likes y.

It is also to have a function as an argument, eg. “Ravi’s father is Rani’s Father is represented as FATHER (father (Ravi, Rani)

Here FATHER is predicate and father (Ravi) is a function to indicate Ravi’s father.

Term are those arguments in a predicate. Consider the example given above:

FATHER (father (Ravi, Rani)

Here in the example, Ravi’s father is not explicitly stated. But father(Ravi) represents a person. Since father is a function, it is a term in first-order logic.

Term can be defind as follows:

- A constant is a term.
- A variable is a Term
- If f is a function and x1,x2,x3,……xn are terms, the f(x1,x2,x3,……xn) is a term.
- All terms are generated by applying the above-mentioned rules.

Quantifiers: is a symbol that permits one to declare or identify the range or scope of the variables in a logical expression.

Two quantifiers used in Logic:

1. Universal Quantifier (ﻹ)

2. Existence Qualifier (Ʒ)

1.11 TRADITIONAL CLASSIFICATION OF PROPOSITION

In modern logic proposition means a statement which is said to be true or false. For example, 'Asoka was the king of Pataliputra', 3+6=9, 'Socrates was a philosopher' 'No man is immortal'. Truth or falsity can be ascribed to every one of them. Every one of them can be either true or false. But 'May God help us' -is not a proposition; because the property's truth or falsity can not be ascribed to it.

In modern logic the components of a proposition i.e. the subject and the predicate are known as 'constituents' of a proposition. The truth or falsity of a proposition is called its truth value. The truth-value of a proposition is determined by the fact. If a proposition represents a fact as it is, then the proposition is true; otherwise it is false. But a proposition is different from a fact.

1.11.1 Proposition and Sentence

There is a clear difference between a proposition and a sentence. Though a proposition is expressed in the form of a sentence, it is not identical with a sentence. We can point out following points as their differences,

- A proposition may be understood as that to which truth or falsity can be ascribed. But truth or falsity cannot be ascribed to all kinds of sentences.
- All grammatical sentences are not propositioning. Only indicative sentence can be judged to be either true or false. Interrogative sentences, Imperative sentences, Optative sentences cannot be judged to be either true or false. For example, 'Have you ever gone to Calcutta'? 'May God help us!' Truth or falsity cannot be ascribed to it. So they are not propositions.
- The same proposition may have different verbal expressions. Therefore, the same proposition may be expressed by different sentences. But various sentences have a single underlying meaning which is called a proposition.

1.11.2 Simple Proposition

A simple proposition expresses a simple fact and it cannot be analyzed into further propositions. Moreover a simple proposition makes an assertion about an individual, a person, a place, a thing, a country and so on. For example: Seema is doing a course in logic. Simple proposition has four forms:

(1) Subject less proposition

(2) Subject-predicate proposition

(3) Relational proposition

(4) Class-membership proposition.

1. **Subject less proposition:** The simplest kind of proposition is the subjectless proposition e.g. Fire! Thieves! These propositions are also known as the exclamatory propositions. The proposition such as: It rains, it thunders. etc. are known as impersonal propositions. Such propositions have no logical subject. In these propositions the thinker asserts something but the statement is not fully expressed. These propositions give information and therefore they are regarded as propositions.

2. **Subject-predicate proposition**: A proposition which asserts that a quality or an attribute belongs to something is called a subject-predicate proposition. e.g. 1. Shyam is intelligent. 2. This paper is white. Subject of this type proposition is a singular term. The subject-predicate type of proposition is represented simply as 'S-P'. In predicate logic small letters a, b, c, --- are used to symbolize individuals, and capital letters. A, B, C, --- and are used to symbolize attributes. For example: Shyam is intelligent. The symbolic expression of this proposition is -I Here the symbols 'I' stands for intelligent and's' stands for Shyam.

3. **Relational proposition**: Relational proposition asserts a relation between two or more constituents. There are various ways to express relations. We may use the verbs -drink, love, enemy, hurt, or words greater than, smaller than etc. to express various relations. For example: the lecturer teaches a course in English, he owes me fifty rupees for a bag. A relational proposition may contain any number of constituents. According to the number of constituents the relations are called dyadic, triadic, tetradic, pentadic. Relations involving more than five terms are called polyadic relations.

4. **Class-membership proposition**: Class-membership proposition asserts that an individual is a member of a class. For example: Rabindra Nath Tagore is a poet. This class-membership proposition is symbolically expressed as r ? P. Here 'r' stands for Rabindra Nath Tagore and 'P' stands for the class of poet. In symbolic logic capital letters A, B, C---are used to symbolize the class and small letters a, b, c, ---are used to symbolize the individual.

1.12 CATEGORIAL PROPOSITION

A categorial proposition is defined as any proposition that can be interpreted as asserting a relation of inclusion or exclusion, complete or partial, between two classes.

A class is defined as a collection of all objects which have some specified characteristic in common. Understanding this definition is no more complicated than observing that the class of “lightbulbs” all have the common characteristic of “being a lightbulb.”

Thus, we can have four possible class relations in the various kinds of categorical propositions:

Utilizing the classes, “people” and “good beings” we can illustrate these possibilities:

a. complete inclusion ⟶ “All people are good beings.”

b. complete exclusion ⟶ “No people are good beings.”

c. partial inclusion ⟶ “Some people are good beings.”

d. partial exclusion ⟶ “Some people are not good beings.”

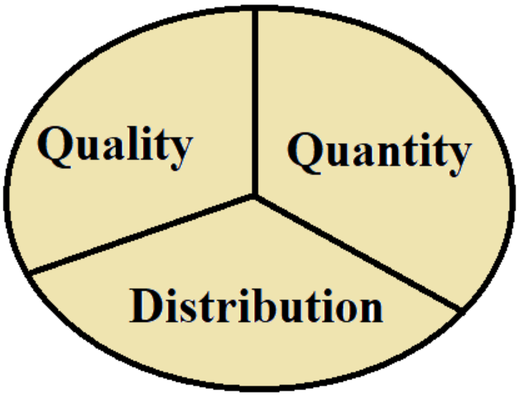


Figure 1: - Quality, Quantity, Distribution

1.12.1 Quality

Every standard-form categorical proposition either affirms, or denies, some class relation, as we have seen. If the proposition affirms some class inclusion, whether complete or partial, its quality is affirmative. So the A proposition, “All S is P,” and the I proposition, “Some S is P,” are both affirmative in quality. Their letter names, A and I, are thought to come from the Latin word, “AffIrmo,” meaning, “I affirm.” If the proposition denies class inclusion, whether complete or partial, its quality is negative. So the E proposition, “No S is P,” and the O proposition, “Some S is not P,” are both negative in quality. Their letter names, E and O, are thought to come from the Latin word, “nEgO,” meaning “I deny.” Every categorical proposition has one quality or the other, affirmative or negative.

1.12.2 Quantity

Every standard-form categorical proposition has some class as its subject. If the proposition refers to all members of the class designated by its subject term, its quantity is universal. So the A proposition, “All S is P,” and the E proposition, “No S is P,” are both universal in quantity. If the proposition refers only to some members of the class designated by its subject term, its quantity is particular. So the I proposition, “Some S is P,” and the O proposition, “Some S is not P,” are both particular in quantity.

The quantity of a standard-form categorical proposition is revealed by the word with which it begins, either “all,” “no,” or “some.” “All” and “no” indicate that the proposition is universal; “some” indicates that the proposition is particular. The word “no” serves also, in the case of the E proposition, to indicate its negative quality, as we have seen.

Because every standard-form categorical proposition must be either affirmative or negative, and must be either universal or particular, the four names uniquely describe each one of the four standard forms by indicating its quantity and its quality: universal affirmative (A), particular affirmative (I), universal negative (E), particular negative (O).

1.12.3 Distribution

Categorical propositions are regarded as being about classes, the classes of objects designated by the subject and predicate terms. We have seen that a proposition may refer to classes in different ways; it may refer to all members of a class or refer to only some members of that class. Thus, the proposition, “All senators are citizens,” refers to, or is about, all senators, but it does not refer to all citizens. That proposition does not affirm that every citizen is a senator, but it does not deny it either. Every A proposition is thus seen to refer to all members of the class designated by its subject term, S, but does not refer to all members of the class designated by its predicate term, P.

To characterize the ways in which terms can occur in categorical propositions, we introduce the technical term distribution. A proposition distributes a term if it refers to all members of the class designated by that term. In A, E, I, and O propositions, the terms that are distributed vary, as follows.

In the A proposition (e.g., “All senators are citizens”): In this proposition, “senators” is distributed, but “citizens” is not. In A proposition (universal affirmatives) the subject term is distributed, but the predicate term is undistributed.

In the E proposition (e.g., “No athletes are vegetarians”): The subject term, “athletes,” is distributed, because the whole class of athletes is said to be excluded from the class of vegetarians. But in asserting that the whole class of athletes is excluded from the class of vegetarians, it is also asserted that the whole class of vegetarians is excluded from the class of athletes. Of each and every vegetarian, the proposition says that he or she is not an athlete. Unlike an A proposition, therefore, an E proposition refers to all members of the class designated by its predicate term, and therefore also distributes its predicate term. E propositions (universal negatives) distribute both their subject and their predicate terms.

In the I proposition (e.g., “Some soldiers are cowards”): No assertion is made about all soldiers in this proposition, and no assertion is made about all cowards either. It says nothing about each and every soldier, and nothing about each and every coward. Neither class is wholly included, or wholly excluded, from the other. In I propositions (particular affirmatives) both subject and predicate terms are not distributed.

In the O proposition (e.g., “Some horses are not thoroughbreds”): Nothing is said about all horses. The proposition refers to some members of the class designated by the subject term; it says that of this part of the class of horses that it is excluded from the class of all thoroughbreds. But they are excluded from the whole of the latter class. Given the particular horses referred to, the proposition says that each and every member of the class of thoroughbreds is not one of those particular horses. When something is said to be excluded from a class, the whole of the class is referred to, just as, when a person is excluded from a country, all parts of that country are forbidden to that person. In O propositions (particular negatives) the subject term is not distributed, but the predicate term is distributed.

We thus see that universal propositions, both affirmative and negative, distribute their subject terms, whereas particular propositions, whether affirmative or negative, do not distribute their subject terms. Thus, the quantity of any standard-form categorical proposition determines whether its subject term is distributed or undistributed. We likewise see that affirmative propositions, whether universal or particular, do not distribute their predicate terms, whereas negative propositions, both universal and particular, do distribute their predicate terms. Thus, the quality of a standard-form categorical proposition determines whether its predicate term is distributed or undistributed.

1.13 ANALYSIS OF THE CATEGORICAL PROPOSITION

A. The quantity of a categorical proposition is determined by whether or not it refers to all members of its subject class (That is, the statement is considered either universal or particular in quantity.) To question "How many members of the subject class are being discussed?" asks for quantity.

B. The quality of a categorical proposition is determined by whether the asserted class relation is one of inclusion or exclusion (That is, the statement or proposition is considered either affirmative or negative in quality.)

C. Indicators of "how many" are called quantity indicators (i.e., quantifiers) and specifically are "All," "No," and "Some."

D. Indicators of affirmative and negative are quality indicators (i.e., qualifiers) and specifically are "are," "are not," "is," "is not," and "No."

E. In sum, thorough knowledge of the following table is absolutely essential to do well in categorical logic:

Name	**Form**	**Quantity**	**Quality**	**Distribution**	
				Subject	**Predicate**
A	All S is P	universal	affirmative	distributed	undistributed
E	No S is P	universal	negative	distributed	distributed
I	Some S is P	particular	affirmative	undistributed	undistributed
O	Some S is not P	particular	negative	undistributed	distributed

1.14 PRACTICE QUESTIONS

Question 1: Explain the term Artificial Intelligence?
Question 2: List down the most popular programming language used in AI?
Question 3: Write down the difference between strong AI and weak AI?
Question 4: What are some examples of Artificial Intelligence in use?

Question 5: Give some real-world applications of AI.

Question 6: What are the types of AI based on capabilities?

Question 7: -What are the various types of Artificial Intelligence based Functionalities?

Question 8: What do you Understand by Logical Connectives in AI?

Question 9: -What are the various functions of Creativeness?

Question 10: - What are the different components of the Expert System?
Question 11: - Name the different domains of Artificial Intelligence.
Question 12: - How you can find out the Artificial Intelligence is different than Machine Learning?

REFERENCE

Links

[1] https://www.upgrad.com/blog/propositional-logic-foundation-of-ai/

[2] https://www.javatpoint.com/propositional-logic-in-artificial-intelligence

[3]https://wps.prenhall.com/wps/media/objects/5909/6050951/MyLogicLab_ebook/MLL_Copi_13e_Ch05/0136141390_Ch05_04.pdf

[4] https://philosophy.lander.edu/logic/prop.html

[5]https://analyticsindiamag.com/why-propositional-logic-is-the-foundation-for-artificial-intelligence/

[6] http://egyankosh.ac.in/bitstream/123456789/37954/1/Unit-4.pdf

Books: -

1. Artificial Intelligence – A Modern Approach (3rd Edition) By – Stuart Russell and Peter Norvig
2. A First Course in Artificial Intelligence By – Deepak Khemani
3. Artificial Intelligence and Expert Systems for Engineers By C.S. Krishnamoorthy, S. Rajeev
4. Artificial Intelligence & Expert Systems Sourcebook by Hunt, V. Daniel

B.SC.(DATA SCIENCE)

SEMESTER-II

INTRODUCTION TO LOGIC

UNIT II: IMMEDIATE INFERENCES

STRUCTURE

2.0 OBJECTIVES

To understand the concept of Conversion, Obversion & Contraposition;

To understand Square of Opposition; Mediate Inferences; Categorical Syllogism and Rules, Fallacies

2.1 CONVERSION

Conversion is the easiest as well as simplest operation used to perform in Artificial Intelligence (AI). In the conversion two terms like subject and predicate term. There are two statements E and I , One conversion is related to this is only to remember that is *only to be used on* E and I statements. It means that whenever a programmer converts an E or an I statement, then new and emerging statements will maintain the same truth value (TRUE or FALSE) as the one programmer just converted (Original Statement). When programmers use conversion on an A or O statement, the resulting statement does not necessarily have the same truth value as the original that making it untrustworthy for drawing inferences.

Example: - No cats are dogs → no dogs are cats.

ORIGINAL	CONVERSE	VALIDITY	LIMITATION
All S are P	All P are S	FALSE	Some P are S
No S are P	No P are S	TRUE	
Some S are P	Some P are S	TRUE	
Some S are not P	Some P are not S	FALSE	

2.2 OBVERSION

Here, Obversion is consider as a bit complicated. But it is consider useful for all statement types (A, E, I and O). It is a two-step process:

- You must change the quality (affirmative/negative) of the statement.
- You must replace the predicate term with its complement.

It is used to change the quality but not the quantity means that A and E statements will switch with one another or that I and O statements will switch with one another. For e.g., "All S are P" becomes "No S are P."

Example: - **No cats are dogs → All cats are non-dogs**.

Obversion		
ORIGINAL	**OBVERSE**	**VALIDITY**
All S are P	No S are non-P	TRUE
No S are P	All S are non-P	TRUE
Some S are P	Some S are not non-P	TRUE
Some S are not P	Some S are non-P	TRUE

2.3 CONTRAPOSITION

Here We observed that contraposition is easier to perform than obversion, although it also involves a two-step process. You can only contrapose A and O statements, if we wants to keep a truth value.

- Like conversion, You switch the subject and predicate terms.
- You replace both the subject and predicate terms with their complements.

Example: -

Some cats are not black animals → Some non-black animals are not non-cats.

Contraposition			
ORIGINAL	**CONTRAPOSITIVE**	**VALIDITY**	**LIMITATION**
All S are P	All non-P are non-S	TRUE	
No S are P	No non-P are non-S	FALSE	Some non-P are not non-S
Some S are P	Some non-P are non-S	FALSE	
Some S are not P	Some non-P are not non-S	TRUE	

2.4 Conversion, Obversion & Contraposition: -

OPERATION		**PROCESS**	**USE FOR**
Conversion	-	Switch subject and predicate	E & I
Obversion	-	Change quality and negate* predicate	All
Contraposition	-	Switch subject and predicate and negate* both terms	A & O

2.5 KINDS OF OPPOSITION

There are four kinds of opposition:

1. Contrary Opposition is opposition between two universals of different quality (A and E).

2. Contradictory Opposition is that between a universal and a particular of different quality (A and O; E and I).

3. Subcontrary Opposition is opposition between two particulars of different quality (I and O).

4. Subaltern opposition is that between a universal and a particular of the same quality (A and I; E and O).

There are four kinds of categorical propositions: A, E, I, and O.

A: All S are P	Universal Affirmative
E: No S are P (All S are not P)	Universal Negative
I: Some S are P	Particular Affirmative
O: Some S are not P	Particular Negative

Each of these propositions can serve as a assertion for immediate inference. Thus, if an A proposition is used as a premise, then one can immediately infer that the corresponding O proposition (having the same subject and predicate terms as the A) is false. Let's see, if the A proposition: "All car salesmen are liars" is given as TRUE, then the O proposition: "Some car salesmen are not liars" must be FALSE.

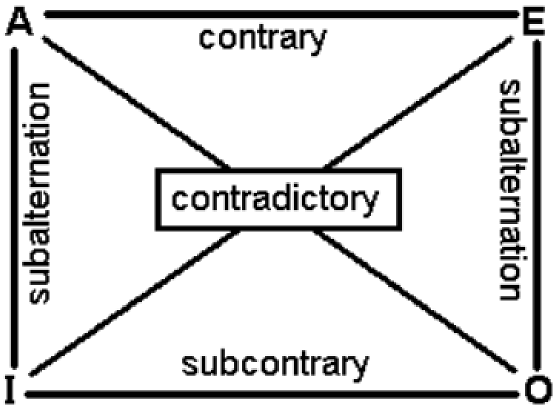


Figure 2.1 Kinds of Opposition

1. A and O propositions are contradictory, where E and I proposition. We have Propositions are contradictory when the truth of one implicit the false of the other one that can be converse. Here is the truth of a proposition of the form that All S are P implicit the false of the corresponding proposition of the form. Some S are not P. Let's take an example, if the proposition "all industrialists are capitalists" (A) is true, then the proposition "some industrialists are not capitalists" (O) must be false. In same directions, if "no mammals are aquatic" (E) is false, then the proposition "some mammals are aquatic" must be true.

2. A and E propositions are contrary. the propositions are contrary when they both cannot be true. An A proposition, for e.g. “all giraffes have long necks” cannot be true at the same time as the corresponding E proposition: “no giraffes have long necks.” However, that corresponding A and E propositions are not contradictory. While they both cannot be true, they can both be false, as with the examples of “all planets are gas giants” and “no planets are gas giants.”

3. I and O propositions are subcontrary. Propositions are subcontrary when it is impossible for both to be false. Because “some lunches are free” is false, “some lunches are not free” must be true. It is possible for corresponding I and O propositions both to be true, as with “some nations are democracies,” and “some nations are not democracies.” Again, I and O propositions are subcontrary, but not contrary or contradictory.

4. Here, Two propositions are said to stand in the relation of subalternation when the truth of the first (“the superaltar”) implies the truth of the second (“the subaltern”), but not conversely. A proposition stands in the subalternation relation with the corresponding I propositions.

The truth of the A proposition “all plastics are synthetic,” implies the truth of the proposition “some plastics are synthetic.” However, the truth of the O proposition “some cars are not American-made products” does not imply the truth of the E proposition “no cars are American-made products.”

2.6 RULES OF OPPOSITION

We derive rules of immediate inference of the four types of opposition mentioned here. These inferences will be from a premise to a conclusion. The premise will be indicated by enclosing its truth-value in a small box within the square as shown in figure. For example: In the diagram, the A proposition is the premise. **We read it as: “Given the A**

as true, therefore, the O is. . .”

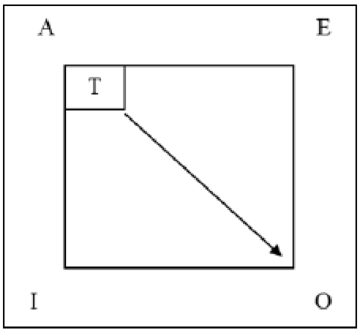


Figure 2.2 Rules of Opposition (a)

Deriving the Rules of Inference for Contradictory Opposition. Contradictory Opposition is that between the A and the O, and the E and the I proposition.

Let us consider the A and the O first. If the statement, “All coins in my pocket are quarters” (A proposition), is true, what can be deduce about the opposed statements, "Some coins in my pocket are not quarters" (O proposition)? True or False? It requires little reflection to see that *if* the A is true, then the O cannot also be true. And it is equally as easy to see that *if* the O is true in the example - Some coins in my pocket are not quarters, then the A must be false for example- All coins in my pocket are quarters.

We can summarize our analysis in a table as follows: Premise Conclusion

Given A as true, therefore O is false.

Given O as true, therefore A is false.

Using the square, we can explain the inference :

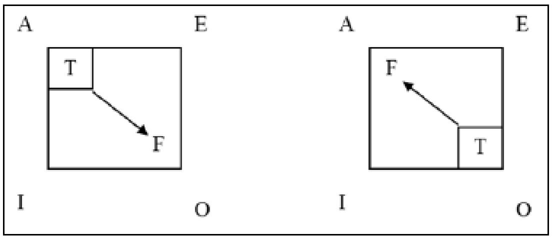


Figure 2.3 Rules of Opposition (b)

- The truth-value in the small box shows the truth-value of the premise
- The truth-value outside the box is the truth-value of the possible conclusions
- The arrow shows the direction of the inference.

2.7 CATEGORICAL PROPOSITIONS

A deductive argument composed of three categorical propositions, one of the statements which shows as the conclusion of the arguments and the other two of which serve as the major and minor premises respectively. For example: -

Some cats are friendly. All cats are mammals. Therefore, some mammals are friendly.

In this categorical syllogism the major term is “friendly,” the minor term is “mammals” and the middle term is “cats.”

- The first proposition here is the major premise.
- The second is the minor premise.
- The third is the conclusion.

Now it so happens, this syllogism is valid. We will get to why it is presently.

The Structure of the Categorical Syllogism: -The standard form of a categorical syllogism has (1) three propositions, and (2) three terms.

The three propositions are:

1. The major premise

2. The minor premise

3. The conclusion

2.7.1 Premises

The **major premise** is the premise that contains the major term. In this case, the major premise is “All animals are organisms.”

The **minor premise** is the premise that contains the minor term. In this case, the minor premise is “All bears are animals.”

In a **standard-form categorical syllogism**, the major premise must come first, next, the minor premise, and lastly, the conclusion. The example above thus has the following structure:

- All animals (M) are organisms (P). (Major premise)
- All bears (S) are animals (M). (Minor premise)
- All bears (S) are organisms (P). (Conclusion)
- In a lateral format, this would be written:
- All M are P. / All S are M. // All S are P.

The three terms are:

1. The major term	P
2. The minor term	S
3. The middle term	M

2.7.2 Terms

The first step in evaluating any argument is identifying its conclusion. From our earlier study of distinguishing premises from conclusions, we know that the word “therefore” is a conclusion indicator. Thus, the last statement in the syllogism is the conclusion, and the **major term** is the predicate term, P, of the conclusion. In the syllogism given, the term “organisms” is the major term of the syllogism.

The **minor term** is the subject term, S, of the conclusion. In this case, it is the term “bear.” The predicate term of the conclusion is called the major term because the predicate of a proposition is usually wider in extension than the subject term. Thus, the predicate term, “organisms,” is wider in scope than the subject term, “bear.”

The **middle term** is the term, M, that appears only in the premises and connects the major and minor terms. In this case, it is the term “animals.”

Observe that each term appears twice in the syllogism. The major term (P) appears once in the conclusion and once in the major premise. The minor term (S) appears once in the conclusion and once in the minor premise. The middle term (M) appears once in the major premise and once in the minor premise, but it never appears in the conclusion.

There are only four standard form categorical figures. They are as follows:

Categorical Figure No. 1 The middle term (M) can occur as the subject term of the major premise and the predicate term of the minor premise. This is called Figure No. 1. For example consider the following syllogistic form in the AAA mood. The form of a standard form categorical syllogism consists of both its mood and its figure. The following argument has the form AAA-1. All M are A All B are M All B are A

Categorical Figure No. 2 The middle term (M) can occur as the predicate terms of both the major and minor premises. This is called Figure No. 2. The following argument has the form AAA-2. All A are M All B are M All B are A

Categorical Figure No. 3 The middle term (M) can occur as the subject terms of both the major premise and the minor premise. This is called Figure No. 3. The following argument has the form AEE-3. All M are A No M are B No B are A

Categorical Figure No. 4 The middle term can occur as the predicate term of the major premise and the subject term of the minor premise. This is called Figure No. 4. The following argument has the form AOO-4. All A are M Some M are not B Some B are not A.

We can now calculate that with the addition of figures to the 64 possible moods, there are 256 standard form categorical syllogistic forms. This might sound like a lot, until we realize that there are thousands upon thousands of existential categorical syllogisms, precisely because there are thousands upon thousands of class terms. But there is even better news. As it turns out there are only 15 of these forms that are valid. This is very helpful to know, for it tells us that any existential categorical syllogism that is a substitution instance of one of these valid forms will also be valid.

2.7.3 Categorical Syllogism Rules

- The middle term must be distributed (universal) in at least one premise. A violation of Rule 1 commits the fallacy of the **undistributed middle term**.
- No term can be distributed in the conclusion which is undistributed in the premise. A violation of Rule 2 commits either the fallacy of the **illicit minor term** or the fallacy of the **illicit major term**.
- No standard form categorical syllogism is valid which has two negative premises. A violation of Rule 3 commits the fallacy of **exclusive premises.**
- A negative premise requires a negative conclusion; a negative conclusion requires a negative premise. A violation of Rule 4 commits either the fallacy of drawing **an affirmative conclusion from negative premises** or the fallacy of drawing a **negative conclusion from affirmative premises**.

- From two universal premises a particular conclusion cannot be drawn. A violation of Rule 5 commits the **existential fallacy** of assuming a universal statement is non-empty.
- A valid syllogism must contain only three terms, each of which is used in the same sense throughout the argument. A violation of Rule 6 commits the **fallacy of four terms**. A specific form of this fallacy is called the fallacy of the ambiguous middle term.

2.8 FALLACIES & VALIDITY THROUGH VENN-DIAGRAMS

The method of Venn diagrams is considered as a effective and best method to test the validity. Since a categorical syllogism has three terms, we require a Venn diagram using three intersecting circles, one circle representing each of the three terms in a categorical syllogism. A three-term diagram has eight regions (the number of regions being 2^n where n is the number of terms).

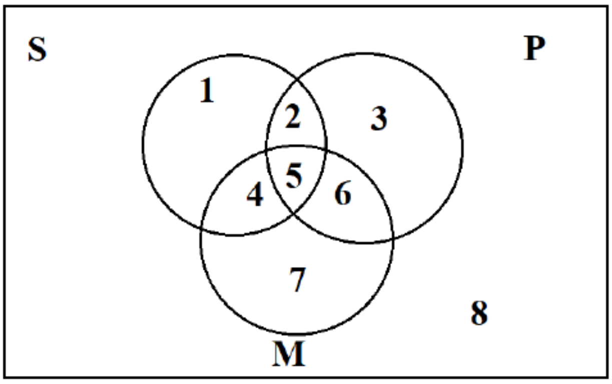


Figure 2.4 Venn-Diagrams (a)

The following table shows the extension of the predicates in the various regions of the diagram.

Region	S (Minor Term)	P (Major Term)	M (Middle Term)
1	TRUE	FALSE	FALSE
2	TRUE	TRUE	FALSE
3	FALSE	TRUE	FALSE
4	TRUE	FALSE	TRUE
5	TRUE	TRUE	TRUE
6	FALSE	TRUE	TRUE
7	FALSE	FALSE	TRUE
8	FALSE	FALSE	FALSE

To use a Venn diagram to test a syllogism, the diagram must be filled in to reflect the contents of the premises. Remember, colouring an area means that that area is empty, the term represented has no extension in that area. What one is looking for in a Venn diagram test for validity is an accurate diagram of the conclusion of the argument that logically follows from a diagram of the premises. Since each of the premises of a categorical syllogism is a categorical proposition, in diagram the premise sentences independently and then see whether the conclusion has already been diagrammed. If so, the argument is valid. If not, then it is invalid.

Keep in mind that one definition of validity is that the propositional or informational content of the conclusion is already expressed in the premises. Venn diagram validity tests provide a graphic tool for using this approach to testing for validity. A categorical syllogism is valid if, but only if, a diagram of its premises produces a diagram that expresses the propositional content of its conclusion. At the starting of the process by preparing a three term Venn diagram. As a convention, organize the diagram as given, 2 circles at the top of the diagram, one centered at the bottom. The upper left-hand circle should represent the minor term that designated S as this term is the subject of the conclusion. The upper right-hand circle should represent the major term and designated P as this term is the predicate of the conclusion. The lower circle should represent the middle term as designated M. Your diagram should look like this:

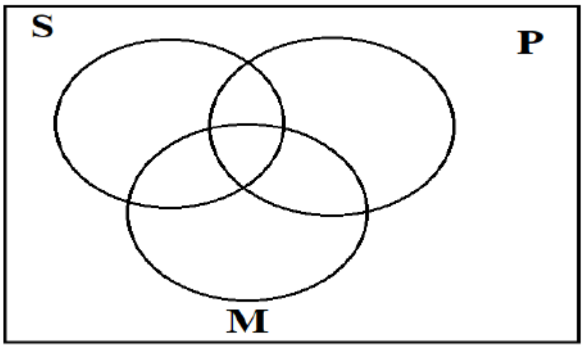


Figure 2.5 Venn-Diagrams (b)

Consider the following argument:

- All Greeks are mortal. (All M are P)
- All Athenians are Greek. (All S are M)
- All Athenians are mortal. (All S are P)

Next, diagram each of the premises. When doing this, act as if there are only 2 relevant circles. Begin with the first premise (frequently the premise involving the major term, sometimes called the major premise). In this example we need to diagram the proposition "All M are P". Ignoring for a moment the circle representing the minor term, your diagram sho8uld look like this:

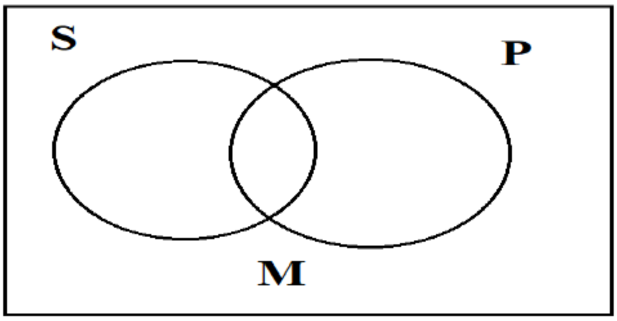


Figure 2.6 Venn-Diagrams (c)

Following the standard conventions, we get:

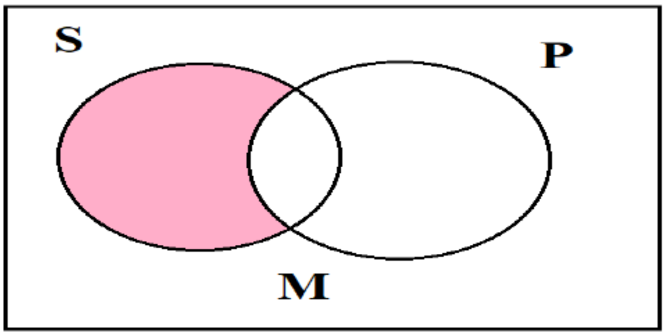


Figure 2.7 Venn-Diagrams (d)

Next, diagram the second premise--"All S are M"- to get:

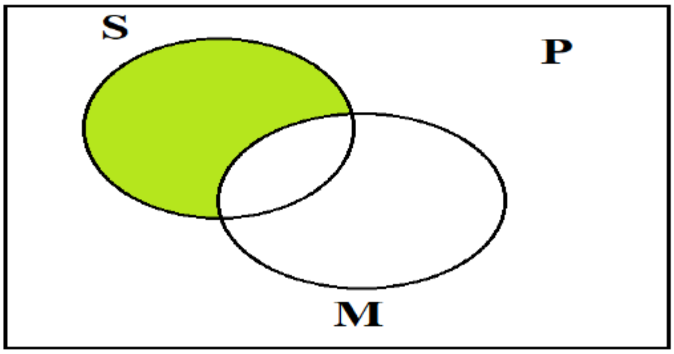


Figure 2.8 Venn-Diagrams (e)

Now, if we overlap the diagrams of the premises we get a diagram of the argument, and we are ready to determine whether the argument is valid or not:

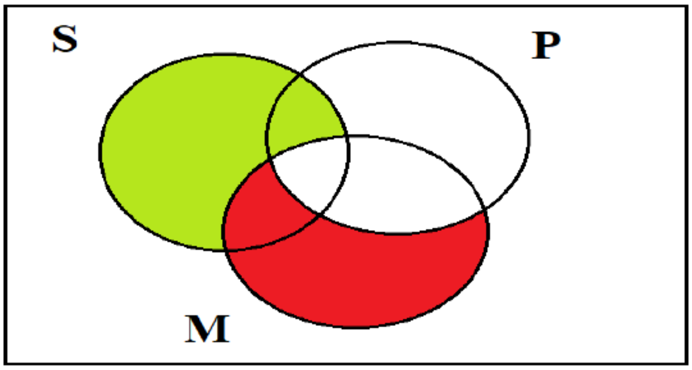


Figure 2.9 Venn-Diagrams (f)

Consider another argument:

- All mathematicians are rational. (All P are M)
- All philosophers are rational. (All S are M)
- All philosophers are mathematicians. (All S are P)

Beginning with the first premise we get:

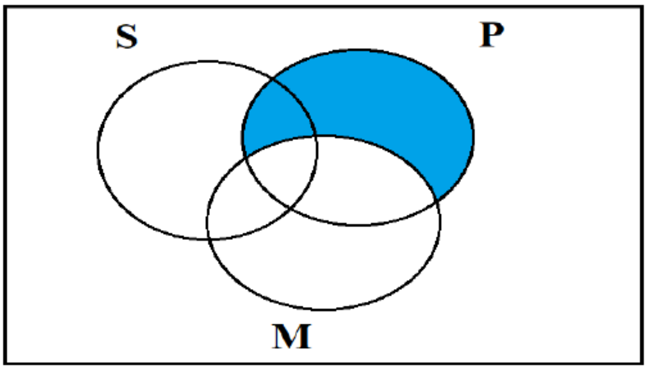


Figure 2.10 Venn-Diagrams (g)

Adding the second premise we get:

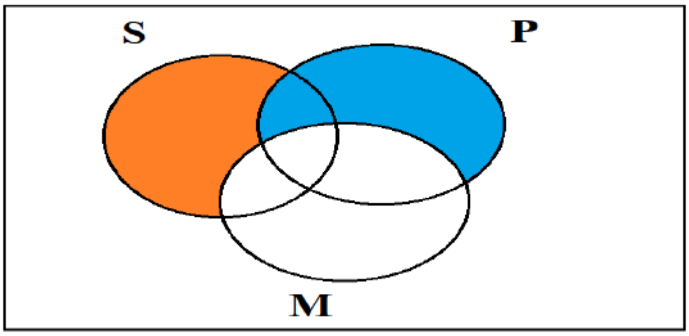


Figure 2.11 Venn-Diagrams (h)

Does this diagram express the informational content of the conclusion "All S are P"? NO. Region 4 of the diagram is not shaded (not empty) so it is possible that there is an S that is not a P. Accordingly, the argument is NOT VALID.

2.9 PRACTICE EXERCISES

Question 1:- What do you mean by Conversions?

Question 2:- How conversion is different than Obversions? Give example?

Question 3:- What are Oppositions? Give an example.

Question 4 :- Let A = {1, 2, 3, 5, 6}, B = {3, 4, 6, 8} be two subsets of the universal set ξ = {1, 2, 3, 4, 5, 6, 7, 8}

Draw Venn diagrams to represent the following sets:

(a) A'

(b) B'

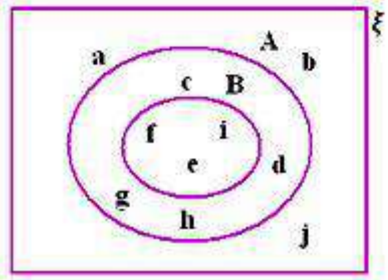
(c) A ∪ B

(d) A ∩ B

(e) (A ∪ B)'

(f) (A ∩ B)'

Question 5:- Use the adjacent Venn diagram to find



(a) A

(b) B

(c) A'

(d) B'

(e) A - B

(f) B - A

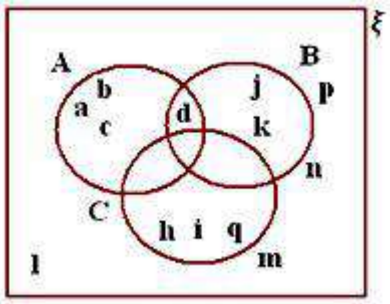
(g) (A - B)'

(h) (B - A)'

Question 6- Use Venn diagram to show (A ∩ B)’ = A’∪B’

- to show A ∩ B when BCA
- to show A ∪ B when BCA

Question 7- Use the adjoining figure to find the following sets:



(a) A ∪ B

(b) B ∩ C

(c) C - A

(d) A - B

(e) (B - C) ∪ A

(f) (C ∩ B) ∪ A

(g) (A ∪ B) ∩ C

(h) (B ∪ C)'

(i) (A ∪ B) - C

(j) (B - A)'

Question 8: - How rules work on opposition. Give an example?

Question 9: -What are the various terms of Propositions?

Question 10: -How middle term is different than Minor term. Give an example?

REFERENCES

[1] https://www.upgrad.com/blog/propositional-logic-foundation-of-ai/

[2] https://www.javatpoint.com/propositional-logic-in-artificial-intelligence

[3]https://wps.prenhall.com/wps/media/objects/5909/6050951/MyLogicLab_ebook/MLL_Copi_13e_Ch05/0136141390_Ch05_04.pdf

[4] https://philosophy.lander.edu/logic/prop.html

[5]https://analyticsindiamag.com/why-propositional-logic-is-the-foundation-for-artificial-intelligence/

[6] http://home.olemiss.edu/~wlawhead/prac2tst.2011.html

Reference Books: -

1. Artificial Intelligence – A Modern Approach (3rd Edition) By – Stuart Russell and Peter Norvig

2. A First Course in Artificial Intelligence By – Deepak Khem ani

3. Artificial Intelligence and Expert Systems for Engineers By C.S. Krishnamoorthy, S. Rajeev

4. Artificial Intelligence & Expert Systems Sourcebook by Hunt, V. Daniel

B.SC.(DATA SCIENCE)

SEMESTER-II

INTRODUCTION TO LOGIC

UNIT III: TRUTH-FUNCTIONAL LOGIC

STRUCTURE

3.0 OBJECTIVES

- To Understand Truth-functional compound statements, Negation, Conjunction, Disjunction and Implication.
- To know about Validity & Invalidity through Truth-table Method; Statement Forms: Tautology, Contradictory and Contingent.

3.1 TRUTH-FUNCTIONAL COMPOUND STATEMENTS

Truth-functional propositional logic is the simplest and expressively weakest member of the class of deductive systems designed to capture the various valid arguments and patterns of reasoning that are specifiable in formal terms. The term ‘formal’ in the context of logic has a number of aspects. (Some of these aspects—those connected with the distinction between form and content—will be discussed a bit later). The aspect concerning us most right now is that connected with the capacity to specify meanings and draw inferences by the more or less automatic application of predetermined rules (rules previously either learned, deduced, or arbitrarily stipulated). This involves far more than the substitution of simple symbols for words. The examples to have in mind are the rules and operations employed in arithmetic and High School algebra. Once we learn how to add, subtract, multiply, and divide the whole numbers {0,1,2,3,...} in elementary school, we can apply these rules, say, to calculate the sum of any two numbers, automatically, without thinking, whether we understand why the rules work or not. The uniformity, simplicity, and regularity of these arithmetical rules, and their applicability with minimal understanding, is shown by the existence of extremely simple artificial devices for effective arithmetical calculation such as the ancient abacus. Before any system can be “automated” in this way (that is, before it can be converted into a system for calculation (i.e., a calculus), its various elements must be represented in an appropriate form.

For example, if each natural number was represented by a symbol possessing no relationship to the symbol representing any other number, no method could exist for adding two numbers by operations employing paper and pencil. It is necessary to represent the system of numbers in a systematic form, to formalize their representation, if such mechanical operations are to be possible.

Truth-functional propositional logic, also known as sentential logic, the sentential calculus, the statement calculus, etc. studies the expressive and deductive relationships among certain combinations of propositions.

The only objects of propositional logic that possess autonomous expressive significance are complete sentences, statements, or propositions that make assertions possessing definite truth values. (There is a subtlety here that we will illuminate shortly). All the deductive logics we are examining in this course are bivalent, meaning that (for the purposes of this course) each sentence is to be considered either true or false. (There are logics in which more than two truth values are admitted, and logics in which there are “truth-value gaps”, but unfortunately, we will not have time to consider them here). To reinforce the convention that we are admitting no alternative to a sentence’s truth or falsity (meaning that the values ‘true’ and ‘false’ are exhaustive), it is best to interpret ‘false’ in this context as equivalent to ‘not true’, meaning that the denial of a sentence’s truth is considered equivalent to the assertion of its falsity.

The symbols A1 , A2 , A3 ,... represent the atomic (elementary) sentence forms from which all others are formed. Although it is probably best to imagine these symbolic complexes as potentially standing for comparatively simple statements such as "It is snowing", "It is sunny", "Albert Einstein published the Special Theory of Relativity in 1905", and "The moon orbits the earth".

3.2 THE PROPOSITIONAL CALCULUS

Consider the following two-premise deductive arguments:

	Argument	Argument Form
(a)	Today is Wednesday or today is Thursday	A or B
	Today is not Thursday not	B
	Therefore, today is Wednesday.	A
(B)	If the gas tank is empty, then the car will not start.	If C then not-D
	The gas tank is empty.	C
	Therefore, the car will not start.	not-D

Table 3.1 Deductive Arguments

Therefore, the car will not start. not-D Now, these two examples embody two common argument forms. It will be the purpose of this chapter to develop a system of logic by means of which we can evaluate such arguments. This system of logic is called Truth-Functional or modern Logic. In truth-functional logic we begin with atomic or simple propositions as our building blocks, from which we construct more complex propositions and arguments. The propositional calculus provides us with techniques for calculating the truth or falsity of a complex proposition based on the truth or falsity of the simple propositions from which it is constructed. These techniques also provide us with a means of calculating whether an argument is valid or invalid.

3.3 TRUTH-FUNCTIONAL OPERATORS

The expressions "not", "and", "or", "if... then . . . ", and "if and only if may be said to be sentential operators just insofar as each may be used in ordinary language and logic alike to 'operate' on a sentence or sentences in such a way as to form compound sentences. The sentences on which such operators operate are called the arguments of those operators. When such an operator operates on a single argument (i.e., when it operates on a single sentence, whether simple or compound), to form a more complex one, we shall say that it is a monadic operator.

Thus the expressions "not" and "it is not the case that" are monadic operators insofar as we may take a simple sentence like

(1.1) "Jack will go up the hill" and form from it the compound sentence

(1.2) "Jack will not go up the hill"

(1.3) "It is not the case that Jack wil l go up the hill." O r we may take a compound sentence like (1.4) "Jack wil l go up the hill and Jil l wil l go up the hill" and form from it a still more complex sentence such as (1.5) "It is not the case that Jack will go up the hill and Jil l will go up the hill." 1

When an expression takes as its arguments two sentences and operates on them to form a more complex sentence we shall say that it is a dyadic operator.

Thus , the expression "and" is a dyadic operator insofar as we may take two simple sentences like (1.1) "Jack wil l go up the hill" and (1.6) "Jil l wil l go up the hill" and form from them a compound sentence such as (1.7) "Jack and Jil l wil l go up the hill" or (more transparently) (1.8) "Jack wil l go up the hill and Jil l will go up the hill."

Or we may take two compound sentences like (1.2) "Jack wil l not go up the hill" and (1.9) "Jil l wil l not go up the hill" and form from them a still more complex sentence such as (1.10) "Jack wil l not go up the hill and Jil l wil l not go up the hill."

T he expressions "or", "if .. . then . .. ", and "if and only i f are also dyadic operators. Dyadi c operators are sometimes called sentential connectives since they connect simpler sentences to form more complex ones.

Note that this sentence is ambiguous between "It is not the case that Jack will go up the hill and it is the case that Jill will go up the hill" and "It is not the case both that Jack will go up the hill and Jill will go up the hill." This ambiguity, along with many others, is easily removed in the conceptual notation of symbolic logic, as we shall shortly see. 2. Some authors like to regard "it is not the case that" as a sort of degenerate or limiting case of a connective — a case where it 'connects' just one sentence. We, however, will reserve the term "connective" for dyadic operators only. § 2 Truth-Functional Operators 249 Now each of the sentential operators cited above is commonly said to be truth-functional in the sense that each generates compound sentences out of simpler ones in such a way that the truth-values of the propositions expressed by the compound sentences are determined by, or are a function of, the truth-values of the propositions expressed by the simpler sentential components. Thus it is commonly said that "it is not the case that" is truth-functional since the compound sentence "It is not the case that Jack will go up the hill" expresses a proposition which is true in just those possible worlds in which the proposition expressed by its simple sentential component "Jack will go up the hill" is false, and expresses a proposition which is false in just those possible worlds in which the proposition expressed by the latter sentence is true; that "and" is truth-functional since the compound sentence "Jack will go up the hill and Jill will go up the hill" expresses a proposition which is true in just those possible worlds in which the propositions expressed by the sentential components "Jack will go up the hill" and "Jill will go up the hill", are both true, and expresses a proposition which is false in all other possible worlds; that "or" is truth-functional since the compound sentence "Jack will go up the hill or Jill will go up the hill" expresses a proposition which is true in all those possible worlds in which at least one of the propositions expressed by the sentential components is true, and expresses a proposition which is false in all other possible worlds; and so on.

The truth-functional properties of the concept of negation can be displayed in the simple sort of chart which logicians call a truth-table. The truth-table for negation may be set out thus:

	P	**~P**
row 1	F	T
row 2	T	F

Table 3.2-Truth table

In effect, a truth-table is an abbreviated worlds-diagram.3 In the (vertical) column, to the left of the double line, under the letter "P", we write a "T" and an "F" to indicate, respectively, all those possible worlds in which the proposition P is true, and all those possible worlds in which the proposition P is false. "T" represents the set of all possible worlds (if any) in which P is true; "F" represents the set of all possible worlds (if any) in which P is false. Together these two subsets of possible worlds exhaust the set of all possible worlds. Each possible world is to be thought of as being included either in the (horizontal) row marked by the "T" in the left-hand column of table or in the (horizontal) row marked by the "F" in that column. In short, the rows of the left-hand column together represent an exhaustive classification of all possible worlds.

3.4 TRUTH-FUNCTIONAL COMPOUND PROPOSITIONS

Truth-functional compound propositions are conventionally classified into five kinds:

1. Negations: The denial of a statement in English is usually made by inserting a “not” into the original statement. For example, the statement “John F. Kennedy was killed by the Mafia” is negated by inserting “not” to produce “John F. Kennedy was not killed by the Mafia.” One can also express a negation in English by such alternative phrases as “it is false that” or “it is not the case that.”

2. Conjunctions: When two statements are joined by the word “and” between them, it is called a conjunction. The component statements are called conjuncts. Thus, the compound statement, 192 “Mr. Obama is the President of the United States and Mr. Biden is his Vice-President” is a conjunction. Conjunctions can be expressed in alternative ways in ordinary English. To illustrate, the standard conjunction “Peter went to school and Paul stayed home” can be expressed in non-standard forms as: Peter went to school, while Paul stayed home. Peter went to school, but Paul stayed home. Peter went to school, even though Paul stayed home. Although Peter went to school, Paul stayed home. Although Paul stayed home, Peter went to school. Peter went to school; however, Paul stayed home. There is no logical limit to the number of components in a conjunction. It could have more than two, as in “Mary baked a cake and Susan went shopping while Tina slept and John drove to work.”

3. Disjunctions: A compound statement in which two or more components are connected by the word “or” is called a disjunction. The component statements so combined are called disjuncts (or alternatives). Example: “We economize on the use of energy or we will have run-away inflation.” (We will later make a distinction between the ‘inclusive’ and ‘exclusive’ senses of the word “or”.) In ordinary English, one and the same disjunction can be expressed in alternative ways. For example, a disjunction that has the same subject term as “Jones is the employer or Jones is the owner” might be equally well expressed as “Jones is the employer or the owner.” The second expression avoids the repetition of the common part of the disjunction which, in this case, is the subject term Jones. The same procedure is followed in a disjunction where the predicate term is the same. Thus, “Brown is innocent or Smith is innocent” can be expressed as “Brown or Smith is innocent.” The negation of a disjunction in English is often expressed by the phrase 193 "neither...nor." Thus, the negation of the disjunction, “Jones or Smith is the best player on the team” is expressed as “Neither Jones nor Smith is the best player on the team.”

4. Conditionals: When two statements are joined by placing the word “if” before the first statement and the word “then” before the second statement, the compound statement resulting is called a conditional (also called an implication or an implicative statement or a hypothetical). Example: “If property taxes continue to rise, then retired people cannot afford to keep their homes.” The statement following “if” is called the antecedent (also called the implicans or protasis) and the statement following “then” is called the consequent (also called the implicate or apodosis). Examples of conditional statements in English that we will consider are:

(1) If Mary gets her car repaired, then she will take a trip. antecedent consequent

(2) If the snow melts in the mountains, then there will be flooding in the valley. antecedent consequent

(3) If Jones is married, then she has a spouse. antecedent consequent

In ordinary English, the standard-form conditional “If Smith is President of the United States, then he is over thirty-five” can be expressed in various ways such as: If Smith is president of the U.S., she is over thirty-five. Smith is over thirty-five if she is president of the U.S. Smith is over thirty-five, provided she is president of the U.S. On condition that Smith is president of the U.S., then she is over thirty-five. Smith is over thirty-five in the case that she is president of the U.S. Smith is president of the U.S. only if she is over thirty-five. 194 Smith is not president of the U. S. unless she is over thirty-five. Unless she is over thirty-five, Smith is not president of the U.S.

5.Bicondionals: A compound statement in which the component statements are joined by the phrase, “if and only if” is called a biconditional. The components of a biconditional are called the right-hand component and the left-hand component.

Examples: D. C. residents can have voting rights if and only if a majority of the states ratify the voting rights amendment. Today is Friday if and only if yesterday was Thursday. John is neither tall nor handsome if and only if John is not tall and John is not handsome.

3.5 STATEMENT CONNECTIVES

We can identify the kinds of compound propositions enumerated above by the kinds of statement connectives used to join the simple propositions into compound ones. Statement connectives combine simple propositions into compound propositions.

The five connectives are:

1. Negation: not, it is false that, it is not the case that.
2. Conjunction: and, but, while, likewise, etc.

3. Disjunction: either...or, or

4. Conditional: if...then, only if, unless, provided, etc.

5. Biconditional: if and only if.

3.6 THE FULL TRUTH-TABLE METHOD

In this tutorial we study how to make use of *full truth-table method* to check the validity of a sequent in SL. Consider this valid sequent:

P, (P→Q) ⊧ Q

To prove that it is valid, we draw a table where the top row contains all the different sentence letters in the argument, followed by the premises, and then the conclusion. Then, using the same method as in drawing complex truth-tables, we list all the possible assignments of truth-values to the sentence letters on the left. In our particular example, since there are only two sentence letters, there should be 4 assignments:

P Q	**P**	**(P→Q)**	**Q**
T T			
T F			
F T			
F F			

Table 3.3: Truth Table1

The next step is to draw the truth-table for all the premises and also the conclusion:

P Q	**P**	**(P→Q)**	**Q**
T T	T	T	T
T F	T	F	F
F T	F	T	T
F F	F	T	F

Table 3.4: Truth Table 2

In the completed truth-table, the first two cells in each row give us the assignment of truth-values, and the next three cells tell us the truth-values of the premises and the conclusion under each of the assignment. If an argument is valid, then every assignment where the premises are all true is also an assignment where the conclusion is true. It so happens that there is only one assignment (the first row) where both premises are true. We can see from the last cell of the row that the conclusion is also true under such an assignment. So, this argument has been shown to be valid.

In general, to determine validity, go through every row of the truth-table to find a row where ALL the premises are true AND the conclusion is false. Can you find such a row? If not, the argument is valid. If there is one or more rows, then the argument is not valid.

Example: -Remember that “(P→Q), ~P, therefore ~Q” is invalid. Look at the truth-table, and determine which line is supposed to show that?

P Q	**(P→Q)**	**~P**	**~Q**
T T	T	F	F
T F	F	F	T
F T	T	T	F
F F	T	T	T

Table 3.5: Truth Table

To show that a sequent is invalid, we find one or more assignment where all the premises are true and the conclusion is false. Such an assignment is known as an *invalidating assignment* (a counterexample) for the sequent.

Let's look at a slightly more complex sequent and draw the truth-table:

(~P∨Q), ~(Q→P) ⊨ (Q↔~P)

Again, we draw a truth-table for the premises and the conclusion:

P Q	**(~P ∨ Q)**	**~ (Q→ P)**	**(Q ↔ ~ P)**
T T	F T T T	F T T T	T F F T
T F	F T F F	F F T T	F T F T
F T	T F T T	T T F F	T T T F
F F	T F T F	F F T F	F F T F

Table 3.6 Truth Table

To help us calculate the truth-values of the WFFs under each assignment, we use the full truth-table method to write down the truth-values of the sentence letters first, and then work out the truth-values of the whole WFFs step by step. The truth-values of the complete WFFs under each assignment is written beneath the main operator of the WFFs. As you can see, the critical one to check is the third assignment. Since there is no assignment where the premises are true and the conclusion is false, the sequent is valid.

3.7 PROPOSITIONS

tautologies, contradictions, and contingencies. Whether a proposition is a tautology, contradiction, or contingency depends on its form—it’s logical structure.

3.7.1 Tautology Proposition

A *tautology*, or *tautologous proposition*, has a logical form that cannot possibly be false (no matter what truth values are assigned to the sentence letters).For Example-

The following propositions are tautologies:

- (A ⊃ A)
- (A ∨ ~A)

- ~ (A • ~A)
- ((A • B) ⊃ (A ∨ B))
- ((A ∨ B) ≡ (B ∨ A))

3.7.2 Contradiction, or Self-Contradictory Proposition

A ***contradiction***, or ***self-contradictory proposition***, has a logical form that cannot possibly be true (no matter what truth values are assigned to the sentence letters).
The following propositions are contradictions:

- (A • ~A)
- ~ (A ∨ ~A)
- ~ (A ⊃ A)
- ((A ∨ ~A) ⊃ (B • ~B))
- ~ ((A ∨ B) ≡ (B ∨ A))

3.7.3 Contingency, or Contingent Proposition

A *contingency*, or ***contingent proposition***, has a logical form that can be either true or false (depending on what truth values are assigned to the sentence letters). Example:-

The following propositions are contingencies:

- A
- ~A
- (A ∨ B)
- ~ (A • B)
- (A ⊃ B)
- ((A ∨ B) ⊃ (C • D))
- ((A • B) ≡ (C ∨ D))

To determine whether a proposition is a tautology, contradiction, or contingency, we can construct a truth table for it. If the proposition is true in every row of the table, it's a tautology. If it is false in every row, it's a contradiction. And if the proposition is neither a tautology nor a contradiction—that is, if there is at least one row where it's true and at least one row where it's false—then the proposition is a contingency.

Consider the following proposition:

If roses are red and violets are blue, then roses aren't red.

This may sound like a contradiction—a proposition that couldn't possibly be true. However, our intuitions about logical properties are often mistaken. To find out which type of proposition it really is, let's symbolize the sentence and construct a truth table for it:

R	**B**	**((R • B) ⊃ ~R)**
0	0	1
0	1	1
1	0	1
1	1	0

As we can see from the truth table above, the proposition is definitely not a contradiction. In fact, there are more ways for it to be true than there are ways for it to be false: it is true in every row except the last row. Since it is true in at least one row and false in at least one row, it is a *contingency*.

Let's look at a few more examples. The following truth table shows the possible truth values for three compound propositions. One of the propositions is a tautology, one is a contradiction, and one is a contingency.

A	**B**	**C**	**(~ (A ∨ B) • B)**	**((A • B) ∨ C)**	**(B ⊃ (B ∨ C))**
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	0	1	1

- The proposition (~ (A ∨ B) • B) is a contradiction, because it is false in every row.
- The proposition ((A • B) ∨ C) is a contingency, because it is true in some rows and false in others.
- The proposition (B ⊃ (B ∨ C)) is a tautology, because it is true in every row.

3.8 TAUTOLOGY EXAMPLES

3.8.1 Tautology in Everyday Language

The highlighted words in these examples are tautological; that is, they have similar meanings. This is no need to use both:

- Remember when 4G cell phones were a ***new innovation****?*
- The ***evening sunset*** was beautiful.
- I need a new ***hot*** water ***heater.***
- Charlie proudly told his mom ***he*** *made* the ***hand-made*** scarf *himself*.
- The careful, there is a lot of ***frozen ice*** on the road!
- I know it's true because I ***heard*** it with my own *ears*.
- She always ***over-exaggerates***.
- In Rome, we saw ***dilapidated ruins.***
- Let's order a ***hoagie sandwich***.
- Alice started her presentation with a ***short summary.***
- He is always making ***predictions*** about the ***future.***
- The school was in ***close proximity*** to the explosion.
- The Gobi is a very ***dry desert.***
- In my *opinion*, *I t****hink*** he is wrong.
- The storm hit at ***2 p.m***. in the *afternoon*.
- The students will ***take turns,*** *one after the other*.
- Having a drug test is a ***necessary requirement*** for the job.
- They hiked to the ***summi****t* at the *top* of the mountain.
- I'm sorry to hear about your ***sad misfortune***.
- She was a ***dark-haired brunette***.
- The hotel room wasn't great, but it was ***adequate enough.***
- I loved reading Sam's ***autobiography*** of his *own life*.

3.8.2 Tautologies from Famous Speakers

Even the best speakers and writers will sometimes let tautology slip into their work. Politicians are particularly prone to this verbal tic as they speak for hours on end and often give slightly different versions of prepared remarks during many campaign stops.

- "It's no exaggeration to say the undecideds could go one way or another." - George H. W. Bush
- "Our nation must come together to unite." - George W. Bush
- "It's deja vu all over again." - Yogi Berra

- "They are simply going to have to score more points than the other team to win the game" - John Madden
- "If we do not succeed, we run the risk of failure." - Dan Quayle
- "Smoking can kill you, and if you've been killed, you've lost a very important part of your life." - Brooke Shields.
- "With malice toward none, with charity for all, with firmness in the right as God gives us to see the right." - Abraham Lincoln

3.9 PRACTICE QUESTIONS

Question 1: - What do you understand by Propositional calculus?

Question 2: - What is the relationship between Relations between propositions: Consistency, Entailment, and Equivalence?

Question 3: - Is the following sentence a tautology?

Either roses are red and violets are blue, or roses are red only if violets aren't blue.

Question 4: - Construct a truth table for the formula $\neg P \wedge (P \rightarrow Q)$.

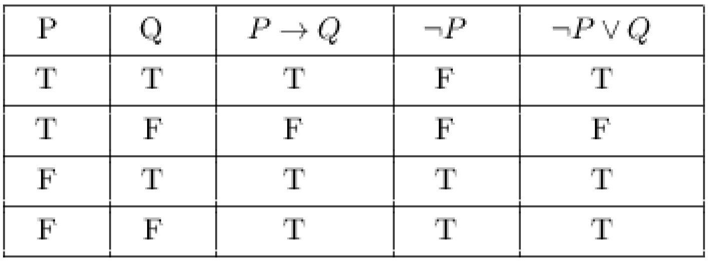
Question 5: - Construct a truth table for $(P \rightarrow Q) \wedge (Q \rightarrow R)$.

Question 6: -

(a) Suppose that P is false and $P \vee \neg Q$ is true. Tell whether Q is true, false, or its truth value can't be determined.

(b) Suppose that $(P \wedge \neg Q) \rightarrow R$ is false. Tell whether Q is true, false, or its truth value can't be determined.

Question 7: - Show that $P \rightarrow Q$ and $\neg P \vee Q$ are logically equivalent.



P	Q	$P \rightarrow Q$	$\neg P$	$\neg P \vee Q$
T	T	T	F	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Since the columns for $P \rightarrow Q$ and $\neg P \vee Q$ are identical, the two statements are logically equivalent. This tautology is called Conditional Disjunction. You can use this equivalence to replace a conditional by a disjunction.

Question 8: - The three propositions (A ⊃ B), (A ∨ B), and ~A are consistent. Draw a truth table.

Question 9: - What do you mean by Logical Equivalence? Give an example?

Question 10: - Difference between Tautologies and Contradiction with example?

REFERENCE

[1] https://www.upgrad.com/blog/propositional-logic-foundation-of-ai/

[2] https://www.javatpoint.com/propositional-logic-in-artificial-intelligence

[3]https://wps.prenhall.com/wps/media/objects/5909/6050951/MyLogicLab_ebook/MLL_Copi_13e_Ch05/0136141390_Ch05_04.pdf

[4] https://philosophy.lander.edu/logic/prop.html

[5]https://analyticsindiamag.com/why-propositional-logic-is-the-foundation-for-artificial-intelligence/

[6] https://literarydevices.net/tautology/

Reference Books: -

1. Artificial Intelligence – A Modern Approach (3rd Edition) By – Stuart Russell and Peter Norvig

2. A First Course in Artificial Intelligence By – Deepak Khemani

3. Artificial Intelligence and Expert Systems for Engineers By C.S. Krishnamoorthy, S. Rajeev

4. Artificial Intelligence & Expert Systems Sourcebook by Hunt, V. Daniel

B.SC.(DATA SCIENCE)

SEMESTER-II

INTRODUCTION TO LOGIC

UNIT IV: PROPOSITIONAL LOGIC

STRUCTURE

4.0 OBJECTIVES

- Understanding Propositional Logic, Logical Connectives: Logical Equivalence.
- To know about Conjunctive and Disjunctive Normal Forms

4.1 FIRST ORDER PREDICATE LOGIC(FOPL)

FOPL is the oldest and most important scheme for logic. FOPL was developed by the logicians as a means for formal reasoning, primarily in the area of mathematics. Few additional representation methods become popular in the coming years and used by the researchers in AI and related fields for use in representing knowledge.

The application of logic as a practical means of representing and manipulating the knowledge in a computer. Now a days, FOPL or predicate calculus as it is sometimes called, has assumed one of the important roles in AI. Reasons to use logics:-

- Logic offers the only formal approach to reasoning that has a theoretical foundation. Specially, used to mechanize or automate the reasoning process.
- Structure of FOPL is flexible enough to permit the accurate representation of Natural Language. It is effective , transformation between Natural Language and representation scheme must be easy.
- FOPL is commonly used in the AI field as one of the popular representation method and used in program designs.
- FOPL statements from a natural language like English are translated into symbolic structures comprised of predicates, functions, variables, constants, quantifiers.
- Propositional Logic is one of the important methods of First Order Predicate Logic.

4.1.1 Syntax of Propositional Logic

The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:

Atomic Propositions- Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

Example-

1. 2+2 is 4, it is an atomic proposition as it is a **true** fact.
2. "The Sun is cold" is also a proposition as it is a **false** fact.

Compound propositions- Compound proposition: Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

1. "It is raining today, and street is wet."
2. "Ankit is a doctor, and his clinic is in Mumbai."

The semantics or meaning of a sentence is just the value true or false; it is an assignment of a truth value to the sentence. the values true and false should not be confused with the symbols

T and F which can appear within a sentence. An interpretation for a sentence or group of sentences is an assignment of a truth value to each propositional symbol.

Formally, the function is defined as follows:

- v¯(⊤)=Tv¯(⊤)=   T.
- v¯(⊥)=Fv¯(⊥)=   F.
- v¯(ℓ)=v(ℓ)v¯(ℓ)= v(ℓ), **where ℓℓ is any propositional variable**.
- v¯(¬A) = T
- v¯(¬A) = T (**if v¯(A)v¯(A) is FF, and vice versa)**
- v¯(A∧B) = T v¯(A∧B)=T (**if v¯(A)v¯(A) and v¯(B)v¯(B) are both TT, and FF otherwise).**
- v¯(A∨B) = T
- v¯(A∨B) = T **(if at least one of v¯(A)v¯(A) and v¯(B)v¯(B) is TT; otherwise FF).**
- v¯(A→B) = T
- v¯(A→B) =T (**if either v¯(B)v¯(B) is TT or v¯(A)v¯(A) is FF)** , and FF otherwise**.(Equivalently, v¯(A→B)=Fv¯(A→B)=F if v¯(A)v¯(A) is TT and v¯(B)v¯(B) is FF, and TT otherwise.)**

The rules for conjunction and disjunction are easy to understand. "AA and BB" is true exactly when AA and BB are both true; "AA or BB" is true when at least one of AA or BB is true. Example, trickier. People are often surprised to hear that any if-then statement with a false hypothesis is supposed to be true. The statement "if I have two heads, then circles are squares" may sound like it ought to be false, but by our reckoning, it comes out true. To make sense of this, think about the difference between the two sentences:

- "If I have two heads, then circles are squares."
- "If I had two heads, then circles would be squares."

4.1.2 Properties of Statement

Satisfiable :- A statement is satisfiable if there is some interpretation means not all or not always. For which it is true.

Contradiction :-A sentence is contradictory means unsatisfactory if there is no interpretation which it is true.

Valid :- A sentence is valid if it is true for zll interpretation. Valid sentences are also called tautologies.

Equivalence:- Two sentences are equivalent if they have the same value under all interpretation.

4.2 LOGICAL CONSEQUENCE

Logical consequences are the natural outcomes that result from a child's actions with others or property. Following through on logical consequences means that the adult guides the child to take responsibility for any harm caused or damage done. The intent is to teach your child that every action has a reaction.

A sentence is a logical consequence of another if it is satisfied by every interpretation which satisfy the first. Actually, it is a logical sequence of other statements if and only if for any interpretation in which the statement is true, the result statement is true. A valid statement is satisfiable and a contradictory statement is invalid, but the converse is not necessarily true.

- P is satisfiable but not valid since an interpretation that assigns false to P assign false to the sentence P.
- P *V ~P* is valid since every interpretation result in a value of true for (P *V ~P)*
- P *& ~P* is a contradiction since every interpretation result in a value of false for
- (P *& ~P)*
- P and *~(~P*) are equivalent since each has the same truth values under every interpretation.
- P is a logical consequence of (*P & Q*) since any interpretation for which (*P & Q*) is true, P is also true.

  The notion of logical consequence provides us with no means to perform valid inferencing in Propositional Logic.

4.3 LOGICAL CONNECTIVES

Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:

1. **Negation:** A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.
2. **Conjunction:** A sentence which has ∧ connective such as, **P ∧ Q** is called a conjunction.

   Example: Rohan is intelligent and hardworking. It can be written as, **P= Rohan is intelligent**, **Q= Rohan is hardworking. → P∧ Q**.
3. **Disjunction:** A sentence which has ∨ connective, such as **P ∨ Q**. is called disjunction, where P and Q are the propositions. **Example: "Ritika is a doctor or Engineer"**, Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as **P ∨ Q**.
4. **Implication:** A sentence such as P → Q, is called an implication. Implications are also known as if-then rules. It can be represented as **If** it is raining, then the street is wet. Let P= It is raining, and Q= Street is wet, so it is represented as P → Q
5. **Biconditional:** A sentence such as **P⇔ Q is a Biconditional sentence, example If I am breathing, then I am alive** P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.

4.3.1 Propositional Logic Connectives

Connective Symbols	Words	Technical Terms	Example
^	AND	Conjunction	**A ^ B**
∨	OR	Disjunction	**A ∨ B**
→	implies	Implication	**A → B**
↔	if and only if	Biconditional	**A ↔ B**
˜	Not	Negation	**˜A OR˜B**

For Negation

P	˜P
TRUE	FALSE
FALSE	TRUE

For Conjunction

A	B	A ^ B
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

For Disjunction

A	B	A ∨ B
TRUE	TRUE	TRUE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	FALSE	FALSE

For Implication

A	B	A → B
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	TRUE
FALSE	FALSE	TRUE

For Biconditional

A	B	A ↔ B
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	TRUE

Truth table with three propositions:

We can build a proposition composing three propositions P, Q, and R. This truth table is made-up of 8n Tuples as we have taken three proposition symbols.

A	**B**	**C**	**˜C**	**A ∨ B**	**A ∨ B →˜C**
TRUE	TRUE	TRUE	FALSE	TRUE	FALSE
TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE	TRUE
FALSE	TRUE	TRUE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE	TRUE
FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
FALSE	FALSE	FALSE	TRUE	FALSE	TRUE

4.4 PRECEDENCE OF CONNECTIVES

Just like arithmetic operators, there is a precedence order for propositional connectors or logical operators. This order should be followed while evaluating a propositional problem. Following is the list of the precedence order for operators:

Precedence	**Operators**
First Precedence	Parenthesis
Second Precedence	Negation
Third Precedence	Conjunction (AND)
Fourth Precedence	Disjunction (OR)
Fifth Precedence	Implication
Six Precedence	Biconditional

Example: How complex propositions can be represented through propositional logic in artificial intelligence so that a machine can understand or interpret the meaning of the propositions:-

- **Ram can play tennis** (let's take it as variable **X**)
- **Ram cannot play tennis** – There is a negation in the sentence, so symbolic representation will be **˜ X**
- **Ram can play tennis and badminton** – Note, there is a new addition 'Badminton', let's take it as variable **Y**. Now, this sentence has a Conjunction, so symbolic representation will be **X** ∧ **Y**
- **Ram can play tennis or badminton** – Here is a Disjunction, so symbolic representation will be **X** ∨ **Y**

- **If Ram can play tennis then he can play badminton** – There is a condition, so symbolic representation will be **X → Y**
- **Ram can play tennis if and only if he can play badminton** – It is a biconditional sentence, so symbolic representation will be **X ↔ Y**

Once the machine reads the massage, it applies the Boolean logic-based formulas to create the TRUE and FALSE chart to interpret the final output of a complex proposition.

4.5 VALIDITY

An argument is known as a set of statements expressing as well as evidence-based conclusion. An argument can be considered as valid if and only if it would be contradictory for the conclusion to be false if all of the premises are true. Validity doesn't require the truth of the premises, instead it merely necessitates that conclusion follows from the formers without violating the correctness of the logical form. If also the premises of a valid argument are proven true, this is said to be sound.

The corresponding conditional of a valid argument is a logical truth and the negation of its corresponding conditional is a contradiction. The conclusion is a logical consequence of its premises.

An argument that is not valid is said to be "invalid". An example :

All men are mortal.
Socrates is a man.
Therefore, Socrates is mortal.

4.6 CONSISTENCY

A sentence φ is consistent with a sentence ψ if and only if there is a truth assignment that satisfies both φ and ψ. A sentence ψ is consistent with a set of sentences Δ if and only if there is a truth assignment that satisfies both Δ and ψ.

For example, the sentence (p ∨ q) is consistent with the sentence (¬p ∨ ¬q). However, it is not consistent with (¬p ∧ ¬q). As with logical equivalence, we can use the truth table method to determine logical consistency. The following truth table shows all truth assignments for the propositional constants. The third column shows the truth values for the first sentence; the fourth column shows the truth values for the second sentence, and the fifth column shows the truth values for the third sentence. The second and third truth assignments here make (p ∨ q) true and also (¬p ∨ ¬q); hence (p ∨ q) and (¬p ∨ ¬q) are consistent. By contrast, none of the truth assignments that makes (p ∨ q) true makes (¬p ∧ ¬q) true; hence, they are not consistent.

p	*Q*	*p* ∨ *q*	¬*p* ∨ ¬*q*	¬*p* ∧ ¬*q*
T	T	T	F	F
T	F	T	T	F
F	T	T	T	F
F	F	F	T	T

4.7 LOGICAL EQUIVALENCE

Two propositions are logically equivalent if they have the same truth values for all combinations of truth values for their atomic parts. For example: -the proposition (A → B) is logically equivalent to (~A ∨ B). the logical equivalence is generally checked using truth tables. For Example: -

IF the machine is defective THEN production is less.

First , if A is TRUE and B is also TURE then the implication is TRUE. IF the machine is defective THEN production is less which is true.

Second, if A is FALSE and B is TRUE, then the implication is TRUE. IF the machine is not defective THEN the production is less.

Third, if A is TRUE and B is FALSE, the implication is FALSE, i.e, IF the machine is defective THEN the production is not less which cannot be admitted. Hence implication is FALSE.

Last, if A and B are FALSE, the implication is TRUE. Example, IF the machine is not defective THEN the production is not less which is TRUE.

A	**B**	**A → B**	**~A**	**~A ∨ B**
T	T	T	F	F
T	F	T	T	F
F	T	T	T	F
F	F	F	T	T

4.8 TAUTOLOGIES

A tautology is a formula which is "always true" that is, it is true for every assignment of truth values to its simple components. You can think of a tautology as a rule of logic.

Propositions that are TRUE for all possible combinations of Truth values of their atomic parts are called tautologies. It implies that tautologies are always TRUE irrespective of the values of its components e.g. The propositions.

((A & (A → B)) → B) is tautology.

Example. Show that (A→B) ∨ (B → A) is a tautology.

I construct the truth table for (A→B) ∨ (B → A) and show that the formula is always true.

A	B	A → B	(B → A)	(A→B) ∨ (B → A)
T	T	T	T	T
T	F	F	T	T
F	T	T	F	T
F	F	T	T	T

The last column contains only T's. Therefore, the formula is a tautology.

Commonly used Logical Equivalence:

1	A	Is logically equivalence to	(~~A)
		Is logically equivalence to	(A & A)
2	(A & B)	Is logically equivalence to	(B & A)
3	(A ∨ B)	Is logically equivalence to	(B ∨ A)
4	(A & (B & C))	Is logically equivalence to	((A & B) & C)
	(A ∨ (B ∨ C)	Is logically equivalence to	(A ∨ B) ∨ C)
5	~ (A & B)	Is logically equivalence to	(~A ∨ ~B)
	~ (A ∨ B)	Is logically equivalence to	(~A & ~B)
6	(A→B)	Is logically equivalence to	(~A∨B)
	(A→B)	Is logically equivalence to	(~A & ~B)
	(A→B)	Is logically equivalence to	(~B → ~A)
7	(A → (B → C))	Is logically equivalence to	((A → B) → C)
8	(A ↔ B)	Is logically equivalence to	(A & B) ∨ (~A & ~B)
	(A ↔ B)	Is logically equivalence to	(A ↔ B) & (B ↔A)
	(A & (B ∨ C))	Is logically equivalence to	((A & B) ∨(A & C))
	(A ∨ (B & C)	Is logically equivalence to	((A ∨ B) & ((A∨ C))

It can be observed that two proposition A and B are said to be logically equivalent if (A ↔ B) is a tautology.

In Truth Table, the Truth values of a sentence is always FALSE for all combinations of its constituents, then the sentence is considered as contradictions. Example: - The statement (A & ~A) is a contradiction because a statement and its negation cannot exist together. If consider the case of tautologies, truth tables are to be constructed for contradictions. The Truth table for the above problem is :-

A	**~A**	**A & ~A**
T	**F**	**F**
F	**T**	**F**

A statement A is a tautology if and only if its negation (~A) is a contradiction. There are some statements that are consider as contradiction.

1	**(A & ~A)**
2	**(A ∨ B) & (~A & ~B)**
3	**(A ∨ B) & (A ∨ ~B) & (~A ∨ B) & (~A ∨ ~B)**
4	**((A & C) ∨ (B & ~C)) ↔((~A & C) ∨ (~B & ~C))**

4.9 CONJUNCTIVE NORMAL FORM (CNF)

CNF is an procedure to Boolean logic that expresses formulas as conjunctions of clauses with an AND or OR. Each clause connected by a conjunction, or AND, must be either a literal or contain a disjunction, or OR operator. CNF is useful for automated theorem proving. Example:

We can resolve two clauses which are given below:

[Snakes (g(a) V Loves (f(a), a)]   and   [¬ Loves(x, y) V ¬ Kill(x, y)]

Where two complimentary literals are: **Loves (f(a), a) and ¬ Loves (x, y)**

These literals can be unified with unifier **θ= [x/f(a), and y/a]** , and it will generate a clause: **[Snakes (g(a) V ¬ Kill(f(a), a)].**

Steps for Resolution: -

- Conversion of facts into first-order logic.
- Convert FOL statements into CNF
- Negate the statement which needs to prove (for contradiction)
- Generate resolution graph.

Example :-

a. Rehana likes all kind of food.
b. Apple and vegetable are food.
c. Anything anyone eats and not killed is food.
d. John eats peanuts and still alive.
e. Harry eats everything that Anil eats.

4.9.1 Prove by Resolution

Step-1: Conversion of Facts into FOL: In the first step we will convert all the given statements into its first order logic.

∀x : food(x) →likes (Rehana, x)

Food (Apple)^food(vegetables)

∀x ∀y: eats (x, y) ^˥killed(x) →food(y)

eats (John, Peanuts) ^ ˥killed(x) →food(y)

∀x : eats(John, x) → eats(Harry, x)

∀x : ˜killed(x) → alive(x)

∀x : ˜alive(x) → killed (x)

Step-2: Conversion of FOL into CNF

In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.

Eliminate all implication (→) and rewrite:-

∀x ¬ food(x) V likes(Rehana, x)

food(Apple) Λ food(vegetables)

∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)

eats (john, Peanuts) Λ alive(john)

∀x ¬ eats(Anil, x) V eats(Harry, x)

∀x¬ [¬ killed(x)] V alive(x)

∀x ¬ alive(x) V ¬ killed(x)

Move negation (¬)inwards and rewrite:-

∀x ¬ food(x) V likes(John, x)

food(Apple) Λ food(vegetables)

∀x ∀y ¬ eats(x, y) V killed(x) V food(y)

eats (Anil, Peanuts) Λ alive(Anil)

∀x ¬ eats(Anil, x) V eats(Harry, x)

∀x ¬killed(x)] V alive(x)

∀x ¬ alive(x) V ¬ killed(x)

Rename variables or standardize variables:-

∀x ¬ food(x) V likes(Rehana, x)

food(Apple) Λ food(vegetables)

∀y ∀z ¬ eats(y, z) V killed(y) V food(z)

eats (John, Peanuts) Λ alive(John)

∀w¬ eats(Anil, w) V eats(Harry, w)

∀g ¬killed(g)] V alive(g)

∀k ¬ alive(k) V ¬ killed(k)

Eliminate existential instantiation quantifier by elimination. We will eliminate existential quantifier ∃, and this process is known as Skolemization. But here is no existential quantifier so all the statements will remain same in this step. Drop Universal quantifiers. In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.

¬ food(x) V likes(John, x)

food(Apple)

food(vegetables)

¬ eats(y, z) V killed(y) V food(z)

eats (Anil, Peanuts)

alive(Anil)

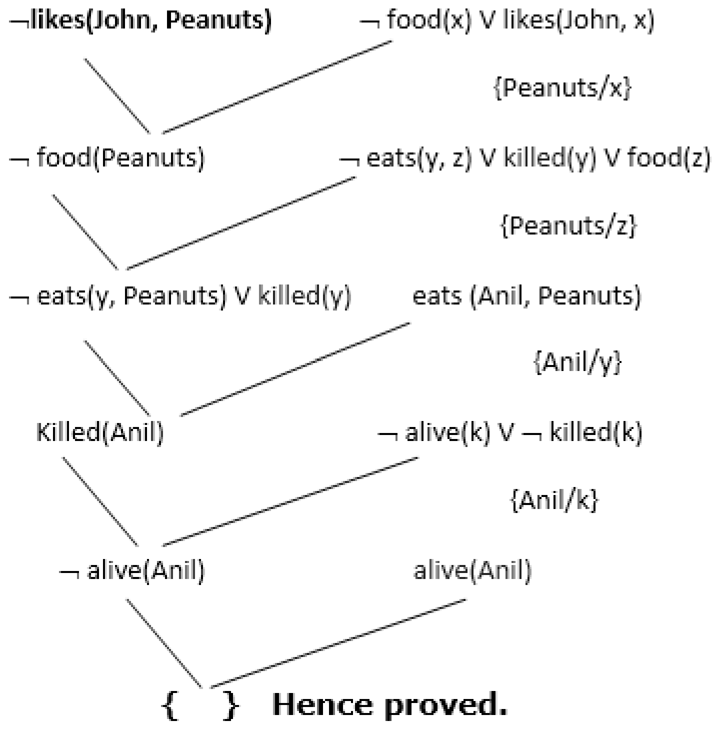
¬ eats(Anil, w) V eats(Harry, w)

killed(g) V alive(g)

¬ alive(k) V ¬ killed(k)

Step-3: Negate the statement to be proved: -In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)

Step-4: Draw Resolution graph: Now in this step, we will solve the problem by resolution tree using substitution. For the above problem, it will be given as follows:



4.10 DISJUNCTIVE NORMAL FORMS

A statement is in disjunctive normal form if it is a disjunction (ORs sequence) consisting of one or more disjuncts, each of which is a conjunction (AND) of one or more literals (i.e., statement letters and negations of statement letters; Mendelson 1997, p. 30). Disjunctive normal form is not unique.

If A, B are two statements, then "A or B" is a compound statement, denoted by A ∨ B and referred as the disjunction of A and B. The disjunction of A and B is true whenever at least one of the two statements is true, and it is false only when both A and B are false.

A	**B**	**A ∨ B**
T	T	T
T	F	T
F	T	T
F	F	F

Example: - if A is "4 is a positive integer" and B is "√5 is a rational number", then A ∨ B is true as statement A is true, although statement B is false.

4.11 PRACTICE EXERCISES

1. Construct a truth table for the expression (A &(A∨B)). What single term is this expression equivalent to?

2. Given the following PL expressions, place parentheses in the appropriate places to form fully abbreviated wffs:

(a) ~P ∨ Q & R →S→U & Q

(b) ~P ∨ Q & P ↔U → ~R

(c) Q ∨ P ∨ ~R & S → ~U & P ↔R

3. Find the meaning of the statements: -

(~P ∨ Q) & R → S ∨ (~R & Q)

For each of the interpretations given below:

(a) *I1*: P is true, Q is true, R is false, S is true

(b) *I2*: P is true, Q is false, R is true, S is true.

4. Transform the following sentences into the disjunctive normal form :-

(a) ~(P ∨ ~Q) & (R → S)

(b) P→((Q & R) ↔ S)

5. Transform the following sentences into the conjunctive normal form :-

(a) P ∨ (~P & Q & R)

(b) (~P & Q) ∨ (P & ~Q) & S

6. What is the difference between Conjunctive normal form and Disjunctive normal forms with example.

7. What is the syntax for propositional Logic?

8. List down the commonly used logical equivalence with example.

9. Draw the Truth table with three propositions.

10. Difference between Logical Consequences and logical Corrective Ness.

Reference Links for practice and understanding: -

[1] https://www.upgrad.com/blog/propositional-logic-foundation-of-ai/

[2] https://www.javatpoint.com/propositional-logic-in-artificial-intelligence

[3]https://wps.prenhall.com/wps/media/objects/5909/6050951/MyLogicLab_ebook/MLL_Copi_13e_Ch05/0136141390_Ch05_04.pdf

[4] https://philosophy.lander.edu/logic/prop.html

[5]https://analyticsindiamag.com/why-propositional-logic-is-the-foundation-for-artificial-intelligence/

Reference Books: -

1**.** Artificial Intelligence – A Modern Approach (3rd Edition) By – Stuart Russell and Peter Norvig
2. A First Course in Artificial Intelligence By – Deepak Khemani
3. Artificial Intelligence and Expert Systems for Engineers By C.S. Krishnamoorthy, S. Rajeev
4. Artificial Intelligence & Expert Systems Sourcebook by Hunt, V. Daniel

B.Sc.(DATA SCIENCE)

SEMESTER-II

INTRODUCTION TO LOGIC

UNIT V: PREDICATE LOGIC

STRUCTURE

5.0 OBJECTIVES

- To Understand Preidcate logic , its syntax and semantics
- Translating simple syllogistic sentences to Predicate logic,
- To perform Conversion to Clausal form Resolution, Unification,

5.1 ABOUT LOGIC

Logic is concerned with reasoning and the validity of arguments. In general, in logic, we are not concerned with the truth of statements, but rather with their validity. That is to say, although the following argument is clearly logical, it is not something that we would considerto be true:

All lemons are blue
Mary is a lemon
Therefore, Mary is blue

This set of statements is considered to be valid because the conclusion (Mary is blue) follows logically from the other two statements, which we often call the premises

5.1.1 Validity

In everyday language a person utters a validity if he or she says something which is true but only because of its form, like every day is rainy or else some days are not rainy.

A formula of predicate logic is valid, or a validity, if it is satisfied by every assignment in every structure.

- $\exists xA \leftrightarrow \neg\forall x\neg A$
- $\forall xA \leftrightarrow \neg\exists x\neg A$
- $\exists x(A\vee B) \leftrightarrow \exists xA\vee\exists xB$
- $\forall x(A\wedge B) \leftrightarrow \forall xA\wedge\forall xB$
- $\forall x\ x=x$ (identity axioms have the special role that they are always assumed when "=" is part of the formula.)

5.1.2 Satisfiable

A formula is satisfiable if it is satisfied by some assignment in some structure.

- $\forall x\exists yR_0(x,y)\wedge\neg\exists y\forall xR_0(x,y)$
- $\forall x(P_0(x)\vee P_1(x))\wedge(\neg\forall xP_0(x)\vee\neg\forall xP_1(x))$
- $\neg(\exists xP_0(x)\rightarrow\forall xP_0(x))$

5.2 PREDICATE CALCULUS

The vocabulary of Predicate Logic

1. Individual constants: {d, n, j, ...}

2. Individual variables: {x, y, z, ...} The individual variables and constants are the terms.

3. Predicate constants: {P, Q, R, ...} Each predicate has a fixed and finite number of arguments called its arity or valence. As we will see, this corresponds closely to argument positions for natural language predicates.

4. Connectives: $\{\neg, \vee, \wedge, \rightarrow, \leftrightarrow\}$ 5. Quantifiers: $\{\forall, \exists\}$ 6. Constituency labels: parentheses, square brackets and commas.

5.3.1 Syntax

- Predicate calculus allows us to reason about properties of objects and relationships between objects.
- In propositional calculus, we could express the English statement “I like cheese” by A. This enables us to create constructs such as $\neg$A, which means “I do not like cheese,” but it does not allow us to extract any information aboutthe cheese, or me, or other things that I like.
- In predicate calculus, we use predicates to express properties of objects. So the sentence “I like cheese” might be expressed as L(me, cheese) where L is a predicate that represents the idea of “liking.” Note that as well as expressing a property of me, this statement also expresses a relationship between me and cheese. This can be useful, as we will see, in describing environments for robots and other agents.\
- For example, a simple agent may be concerned with the location of various blocks, and a statement about the world might be T(A,B), which could mean: Block A is on top of Block B.
- It is also possible to make more general statements using the predicate calculus.
- For example, to express the idea that everyone likes cheese, we might say $(\forall x)(P(x) \rightarrow L(x, C))$
- The symbol $\forall$ is read “for all,” so the statement above could be read as “for every x it is true that if property P holds for x, then the relationship L holds between x and C,” or in plainer English: “every x that is a person likes cheese.” (Here we are interpreting P(x) as meaning “x is a person” or, more precisely, “x has property P.”)
- Note that we have used brackets rather carefully in the statement above.
- This statement can also be written with fewer brackets: : $\forall x\ P(x) \rightarrow L(x, C)$, $\forall$ is called the universal quantifier.
- The quantifier $\exists$ can be used to express the notion that some values do have a certain property, but not necessarily all of them: $(\exists x)(L(x,C))$
- This statement can be read “there exists an x such that x likes cheese.”
- This does not make any claims about the possible values of x, so x could be a person, or a dog, or an item of furniture. When we use the existential quantifier in this way, we are simply saying that there is at least one value of xfor which L(x,C) holds.

5.2.2 Relationships between $\forall$ and $\exists$

- It is also possible to combine the universal and existential quantifiers, such asin the following statement: $(\forall x)\ (\exists y)\ (L(x,y))$.
- This statement can be read “for all x, there exists a y such that L holds for x and y,” which we might interpret as “everyone likes something.”

- A useful relationship exists between $\neg$ and $\forall$. Consider the statement “not everyone likes cheese.” We could write this as

  $\neg(\forall x)(P(x)\rightarrow L(x,C))$--------------(1)

- As we have already seen, $A\rightarrow B$ is equivalent to $\neg A \vee B$. Using DeMorgan’s laws, we can see that this is equivalent to $\neg(A \wedge \neg B)$. Hence, the statement above, can be rewritten:

  $\neg(\forall x)\neg(P(x) \wedge \neg L(x,C))$------------(2)

- This can be read as “It is not true that for all x the following is not true: x is a person and x does not like cheese.” If you examine this rather convoluted sentence carefully, you will see that it is in fact the same as “there exists an x such that x is a person and x does not like cheese.” Hence we can rewrite it as

  $(\exists x)(P(x) \wedge \neg L(x,C))$ ------------------(3)

- In making this transition from statement (2) to statement (3), we have utilized the following equivalence: $\exists x \cong \neg(\forall x)\neg$

- In an expression of the form $(\forall x)(P(x, y))$, the variable x is said to be bound, whereas y is said to be free. This can be understood as meaning that the variable y could be replaced by any other variable because it is free, and the expression would still have the same meaning, whereas if the variable x were to be replaced by some other variable in $P(x,y)$, then the meaning of the expression would be changed $(\forall x)(P(y, z))$ is not equivalent to $(\forall x)(P(x, y))$, whereas $(\forall x)(P(x, z))$ is.

- Note that a variable can occur both bound and free in an expression, as in $(\forall x)(P(x,y,z) \rightarrow (\exists y)(Q(y,z)))$

- In this expression, x is bound throughout, and z is free throughout; y is free in its first occurrence but is bound in $(\exists y)(Q(y,z))$. (Note that both occurrences ofy are bound here.)
- Making this kind of change is known as substitution. Substitution is allowed of any free variable for another free variable.

5.4 <u>FUNCTIONS</u>

- In much the same way that functions can be used in mathematics, we can express an object that relates to another object in a specific way using functions.
- For example, to represent the statement “my mother likes cheese,” we might use L(m(me),cheese)
- Here the function m(x) means the mother of x. Functions can take more than one argument, and in general a function with n arguments is represented as f(x1,

x2, x3, . . . , x_n)

5.4 ELEMENTS OF PREDICATE LOGIC

To strengthen your understanding of Predicate Logic and its basic elements,. Consider the following sentences

- Aristotle is a man
- Socrates is a man
- Bob is a man

These sentences express different propositions but they have the same syntactic form. Each one has a subject (Aristotle, Socrates, and Bob) and a verb phrase (is a man). Proper names like Aristotle, Socrates, and Bob are expressed in Predicate Logic using terms like a, s, b, which are called **individual constants** (or simply **individuals**).

a =Aristotle

s = Socrates

b = Bob

Each individual constant refers to the **entity** in the world which bears that name. For example, the proper name Aristotle is expressed by the individual constant a which denotes the person who bears that name in the world. More succinctly, Aristotle denotes the person who bears that name in the world.

Now consider the sentences in which the subject is the same but the verb phrase changes.

a. Aristotle is a man
b. Aristotle is pompous
c. Aristotle jogs

Verb phrases like is a man, is pompous, and jogs, express **predicate constants** (simply **predicates**), which are written using uppercase letters like M, P, J; alternatively, MAN, POMPOUS, JOG.[1] A predicate constant denotes a property of entities in the world. For instance, the verb phrase is pompous expresses thepredicate constant P (or POMPOUS) and this predicate constant denotes the property of being pompous in the world, whose extension is the set of entities that are pompous.

a. MAN = is a man
b. POMPOUS = is pompous
c. JOG = jogs

Predicate constants are not propositions, much like verbs are not sentences. Like a verb, a predicate constant has to combine with one or more elements to form a proposition. These elements are typically, but not exclusively, individual constants. Traditionally, the elements required by a verb are called its **arguments**. Simple predicate constants, need only combine with one argument to form a proposition. If a predicate constant only needs one argument, then it is called a 1-place predicate; if it requires two, it is called a 2-place predicate, and so on.

a. Aristotle is a man
   MAN(a)
b. Socrates is pompous

POMPOUS(s)

c. Bob jogs
JOG(b)

A sentence like Aristotle is a man is expressed in Predicate Logic by the proposition M(a), which isobtained by combining the 1-place predicate M and the individual a. The proposition M(a) is true if and only if it correctly describes a situation in which the entity bearing the name Aristotle has the property of being aman in the world.

There are of course verb phrases that denote relationships between multiple entities, a Aristotle chased Socrates
Kermit kissed Ms. Piggy
Dorothy met the Wizard of Oz

The sentences contain verbs that require both a **subject** and an **objects**. Each verb corresponds to a relationship between the subject and its object. In this case, the predicate constant expressed by each verb needs two arguments to form a proposition,

a. Aristotle chased SocratesCHASE(a,s)
b. Kermit kissed Ms. PiggyKISS(k,p)
c. Dorothy met the Wizard of Oz
MET(d,z)

This is where Predicate Logic gets complicated. Individual constants, like a, s, and b, denote specific entities in the world. For this reason, they are excellent translations of the proper names Aristotle, Socrates,and Bob.

a. Every man is mortal
b. No man chased Socrates
c. Kermit kissed someone

It is intuitively clear from these examples that noun phrases like every man and no man do not refer to specific entities in the world. Translating them into individual constants would therefore be inappropriate. We need to abstract away from specific entities to deal with these meanings. Predicate Logic contains a set of special elements called **individual variables** (or simply **variables**), written x, y, z,…, that serve this purpose. An individual variable does not have a constant reference to a specific entity. You can think of a variable as a place-holder for the argument of a predicate.

d. MAN(x) x is a man
e. CHASE(x,s) x chased Socrates
f. KISS(k,x) Kermit chased x
g. LOVE(x,y) x loves y

Since the variables in do not refer to specific entities, the expressions have no truth values. I.e., you cannot check to see whether x is actually a man in the world without knowing what x refersto. Consequently, the expressions in are not propositions. But there is a sense in which they are well- formed and almost propositions: if in each case, the variables were replaced by an individual constant, theresulting expression would be a proposition. For example, by replacing the variable x with the individual constant a and the variable y with s, we get the propositions

a. MAN(a) Aristotle is a man

b. CHASE(a,s) Aristotle chased Socrates

c. KISS(k,a) Kermit chased Aristotle

d. LOVE(a,s) Aristotle loves Socrates

How this helps us with phrases like every man. It is crucial that you first understand what a variable is. Unlike individual constants, variables do not correspond to the any thing in the outside world. That makes it quite difficult to describe what variables are. The closest thing to a variable in natural language is a pronoun, like he, which also do not have constant reference to the outside world.

The pronoun he does not refer to a specific entity. What it refers to is dependent on the context in which the pronoun is spoken. The context determines the value of the pronoun.

a. He is a man

b. He chased Socrates

c. Socrates kissed him

The interpretation of the pronoun he can also come from the preceding noun phrase Aristotle. In this instance, the interpretation of the pronoun is bound by the interpretation of the noun phrase Aristotle. We say that they are coreferential. This further suggests that the interpretation of the pronoun is not fixed in the way that the interpretation of the proper name Aristotle is.

a. Aristotle said that he is a man

b. Aristotle said that he chased Socrates

c. Aristotle said that Socrates kissed him

Variables behave much like pronouns. They are meaningful when they are combined with another element in the language. In Predicate Logic, each variable combines with and is bound by a single **quantifier**. Predicate Logic has two such quantifiers: ∀ (the universal quantifier) and ∃ (the existential quantifier). Since a predicate can combine with more than one variable, it is necessary to write the variable immediately after the quantifier to indicate which variable the quantifier interacts with. In other words, we write ∀x to indicate that the universal quantifier ∀ is interacting with the variable x and not, say, with the variable y.

The logical expressions in are wffs but not propositions. When they combine with a quantifier(one for each variable), the result is which are all propositions.

a. ∀x MAN(x) Everything is a man

b. ∀x CHASE(x,s) Everything chased Socrates

c. ∀x KISS(k,x) Kermit chased everything

d. ∀x ∀y LOVE(x,y) Everything loves everything

a. ∃x MAN(x) *Something is a man*

b. ∃x CHASE(x,s) *Something chased Socrates*

c. ∃x KISS(k,x) *Kermit chased something*

d. $\exists x \exists y$ LOVE(x,y) *Something loves something*

When both □ and □ are used in a single expression, their order is relevant. The paraphrases are lengthy to avoid the ambiguity present in the English sentences. The meaning of each arrangement is different. This is also why we need to keep track of the variable that each quantifier interacts with.

a. $\forall x \exists y$ LOVE(x,y) *For every thing (x) there is a thing (y) such that x loves y*

b. $\exists y \forall x$ LOVE(x,y) *There is a thing (y) such that for every thing (x) x loves y*

Predicate Logic is a more detailed logical language than Propositional Logic but we are still interested in propositions and truth-conditions. Our goals have not changed. The difference is that, in Predicate Logic, propositions are built up by combining individuals and predicates, which denote entities and sets of entities, respectively. To translate expressions like every man and some man, Predicate Logic includes variables and quantifiers to bind the variables. What remains is to give a formal definition of Predicate Logic, its syntax and its semantics, and to reformulate what we mean by a model to fit this new logic.

5.5 SYNTAX

Predicate Logic is defined in the same way as Propositional Logic. The major difference you will find is that the lexicon for Predicate Logic is substantially bigger. There are three primitives in this logic, plus quantifiers. The rules are more extensive but their purpose is the same: to determine the well-formed formulas (wffs) in the language. There is one additional difference I want to emphasize. While the syntax below determines the formulas that are well-formed, it does not determine which wffs are propositions. This is a big departure from Propositional Logic, in which every wff was a proposition. In Predicate Logic, not all wffs are propositions. I will elaborate on this shortly. For now focus on the wffs.

The syntax of predicate logic

A. Lexicon

i. Individual constants: j, m, s, …

ii. Individual variables: x, y, z, … (also $x_1, x_2, \ldots, x_n$)

iii. Predicate constants: P, Q, R, …
(each with a fixed finite number of argument places)

iv. A binary identity predicate: =

v. Logical connectives: ¬, ∧, ∨, →, ↔

vi. Quantifiers: ∀, ∃

vii. Brackets: (,), [,]

B. The Syntactic Rules

(Individual terms are just the individual constants and individual variables)

i. If t_1 and t_2 are individual terms, then $t_1 = t_2$ is a wff

ii. If P is an n-place predicate, and $t_1, t_2, \ldots, t_n$ are terms, then $P(t_1, t_2, \ldots, t_n)$ is a wff

iii. If ϕ and ψ are wff, then $\neg\phi$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, $(\phi \leftrightarrow \psi)$ are wffs

iv. If ϕ is a wff and x is an individual variable, then $\forall x\ \phi$ and $\exists x\ \phi$ are wffsNothing else is a wff

1. Rule 1. is straightforward but its use may be unclear. It is used to equate two individuals

   a. Aristotle is not Socrates

   $\neg a=s$

   Aristotle is not equal to Socrates

   b. Some man is Aristotle

   $\exists x\ (\ MAN(x) \wedge x=a)$

   There is an entity x such that x is a man and x is equal to Aristotle

   c. Every wizard who is not Voldemorte is mortal

   $\forall x\ (\ (WIZARD(x) \wedge \neg x=v) \rightarrow MORTAL(x)\ )$

   For every entity x, if x is a wizard and x is not equal to Voldemorte, then x is mortal

2. Rule II. is the main way of forming well-formed formulas. It states that if you combine an n-placepredicate with n terms (individuals and variables) then the result is a wff. This rule is mainly responsible forforming propositions. In fact, if the n terms are all individuals constants then the result is a proposition.

a) CAT(b) The 1-place predicate CAT and the individual b combine to form a wff. *Bob is a cat*

b) KISS(f,s) The 2-place predicate KISS combines with the individuals f and s to form a wff. *Fred kissed Sue*

c) CAT(x) The 1-place predicate CAT and the variable x combine to form a wff. *x is a cat*

d) KISS(x,s) The 2-place predicate KISS combines with the variable x and individual s toform a wff. *x kissed Sue*

e) KISS(x,y) The 2-place predicate KISS combines with the variables x and y to form a wff. *x kissed y.*

3.Rule III takes a wff ϕ and forms a complex wff, called a quantified expression, by combining ϕ with a quantifier (for a given variable x). The wff ϕ is called the scope of the quantifier ∀x or ∃x. The purpose ofthis rule will be discussed below.

a. ∀x CAT(b) The wff CAT(b) is turned into a quantified expression for the variable x. Since CAT(b) has not variable x in it, this application of rule iii. does nothing.

For every entity x, Bob is a cat

b. ∀x KISS(x,s) The wff KISS(x,s) is turned into a quantified expression for the variable x. For every entity y, y kissed Sue; in other words, everything kissed Sue

c. y KISS(x,y) The wff KISS(x,y) is turned into a quantified expression for the variable x. There is an entity y such that x kissed y; in other words, x kissed something

5.5.1 Other Syntax Rules

Syntactic rules of Lx

a. If δ is a one place predicate and α is a name, then $\delta(\alpha)$ is a formula.
b. If γ is a two place predicate and α and β are names then $\gamma(\alpha, \beta)$ is a formula.
c. If ϕ is a formula. then $\neg\phi$ is a formula.
d. If ϕ and ψ are formulas then $[\phi \wedge \psi]$ is a formula.
e. If ϕ and ψ are formulas then $[\phi \vee \psi]$ is a formula.
f. If ϕ and ψ are formulas then $[\phi \rightarrow \psi]$ is a formula.
g. If ϕ and ψ are formulas then $[\phi \leftrightarrow \psi]$ is a formula.

5.6 SEMANTIC RULES

Expressions are interpreted in models. A model M is a pair ⟨D, I⟩, where D is the domain, a set of individuals, and I is an interpretation function: an assignment of semantic values to every basic expression (constant) in the language.

Models are distinguished both by the objects in their domains and by the values assigned to the expressions of the language by I — by the particular way that the words of the language are “linked” to the things in the world.

M1 = ⟨D1 , I1 ⟩, where:

a. D1 = {David, Nancy, John, Mary}

b. I1 determines the following mapping mapping between names and predicate terms in Lx and objects in D1 :

5.6.1 Converting a Sentence into Clauses

Quantifiers and Variables

Predicate logic, in addition to the individual and predicate constants, and connectives, includes individual variables and the existential and universal quantifiers. x, y, z are variables that range over individuals. Variables enable us to refer to individuals without referring to any particular individual.

If ϕ is a formula and x is an individual variable, then $\forall x\phi$ and $\exists x\phi$ are formulas.

Step I: Elimination of if-then operator:

Replace”→” operator by ¬ & ∨operator.

By replacing ‘if-then’ operator by negation and OR operator, we find.

∀X (¬ (Child (X)∧ □Y (Takes (X, Y) ∧ Biscuit (Y))) ∨ Loves (john, X)

Step II: Reduction of the scope of negation:

Replace ¬ sign by choosing any of the following:

(*a*) $\neg(P \vee Q) = \neg P \wedge Q$

(*b*) $\neg(P \wedge Q) = \neg P \vee Q$

(*c*) $\neg(\neg P) = P$

(*d*) $\neg(\exists X\, P) = \forall X \neg P$

(*e*) $\neg(\forall X\, P) = \exists X \neg P$

In the present context, we rewrite the sentences as:

$$\forall X(\neg \text{Child}(X) \vee \neg(\exists Y(\text{Takes}(X, Y) \wedge \text{Biscuit}(Y))) \vee \text{Loves}(\text{john}, X))$$

$$\Rightarrow \forall X(\neg \text{Child}(X) \vee \forall Y(\neg \text{Takes}(X, Y) \vee \neg \text{Biscuit}(Y)) \vee \text{Loves}$$

Step III: Renaming the variable within the scope of quantifiers:

Rename □X by □Y when {□X} is a subset/proper subset of {∀X}. In the present context, since X and Yare distinct, the above operation cannot be carried out.

Step IV: Moving of quantifiers in the front of the expression:

Bring all quantifiers at the front of the expression.

Applying this on the example yields.

$$\forall X \forall Y \neg \text{Child}(X) \vee \neg \text{Takes}(X, Y) \vee \neg \text{Biscuit}(Y) \vee \text{Loves}(\text{john}, X)$$

Step V: Replacing existential quantifier as Skolem function of essential quantifiers:

When an existential quantifier (Y) precedes an essential quantifier (X), replace Y as S (X), where S is the Skolem function. In this example, since Y is not a subset of X, such a situation does not arise. Also the essential quantifier is dropped from the sentence.

Step VI: Putting the resulting expression in conjunctive normal form (CNF):

For example, if the original expression is in the from P ∨ (Q ∧ R), then replace it by (P ∨ Q) ∧ (Q ∧ R).

In the present context, the resulting expression corresponding to expression (3) being in CNF, we need not do any operation at this step.

Step VII: Writing one clause per line:

If the original expression is of the following CNF, then rewrite each clause/line, as illustrated below.

Original Expression:

$$\begin{array}{l}
(\neg P_{11} \vee \neg P_{12} \cdots \vee \neg P_{1n} \vee Q_{11} \vee \neg Q_{12} \cdots \vee Q_{1m}) \wedge \\
(\neg P_{21} \vee \neg P_{22} \cdots \vee \neg P_{2n} \vee Q_{21} \vee \neg Q_{22} \cdots \vee Q_{2m}) \wedge \\
\hdashline
(\neg P_{11} \vee \neg P_{12} \cdots \vee \neg P_{tn} \vee Q_{11} \vee Q_{12} \cdots \vee Q_{tm})
\end{array}$$

After writing one clause per line, the resulting expression become as follow:

$$P_{11}, P_{12}, \cdots P_{1n} \rightarrow Q_{11}, Q_{12} \cdots Q_{1m}$$
$$P_{21}, P_{22}, \cdots P_{2n} \rightarrow Q_{21}, Q_{22} \cdots Q_{2m}$$
$$P_{11}, P_{12}, \cdots P_{2n} \rightarrow Q_{11}, Q_{12} \cdots Q_{1m}$$

5.7 UNIFICATION

Let us understand first substitution-

Consider literal, Man (Gaurav) and its complement ¬ Man (Gaurav)

Consider together, Man (Gaurav) and ¬ Man(Gaurav) produce a contradiction (produces an empty clause). But the clauses Man (Gaurav) and ¬ Man (Saurabh) do not produce contradiction. Can you find the reason? To establish contradiction there should exists a substitution. A simple recursive method of substitution is called unification.

For unification to work the following procedure is adopted:

Procedure:

Check predicate symbols, if same, proceed, otherwise report failure. For example, LOVE (Rama, Ravana) and HATE (Ram, Ravana) cannot be unified because the predicates are different though the arguments are the same.

First, the predicate symbols should match only then we check the arguments, one pair at a time. If first matches, continue with second and so on. To test each argument pair call the unification procedure recursively. Any substitution which makes two or more expressions equal is called a unifier for the expressions.

The following are the rules of unification:

Example 1:

Unify the expressions:

P(x,y)

P(x, z) P(y,z)

compare x and y, if x is replaced by y/x or y is replaced by x/y they become after, first substitution P(y, y), P(y, z). Now compare the second argument y and z. Substituting z/y or y/z in the first and second P literals respectively for the second argument make the two literals similar. So the two literals match when

x is substituted by y/x and

y is substituted by z/y

in the first literal the two literal now become:

P (y, z)

that is the same. P (y, z)

These all equivalents well match despite the lexical variation. But the last two substitutions though do produce the match cannot be considered good substitution because more substitutions are made than those required absolutely. So the most general unifier need be determined.

Conditions of Unification:

(i) Both the predicates to be unified should have an equal number of terms.

(ii) Neither t_i nor s_i can be a negation operator, or predicate or functions of different variables, or if t_i = term belonging to s_i, or if s_i, = term belonging to t_i then unification is not possible..

Unification Algorithm:

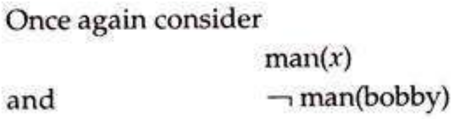
Given, Predicates P (t_1 t_2…, t_n and Q ($S_{1,}$ S_2,.. .,S_m)

To decide whether the predicates P and Q are unifiable and a set S which includes the possible substitution

```
Procedure Unification (P, Q, S, unify)
Begin
S: = Null;
While P and Q contain a Next-symbol do
Begin
Symb1: ≡ Next-symbol (P);
Symb2: ≡ Next-symbol (Q);
If Symb1 ≠ Symb2 Then do
```

5.8 RESOLUTION IN PREDICATE LOGIC

In propositional logic it is easy to examine that the two complementary literals cannot both be true at the same time. Simply look for P and ∼ P. In predicate logic this matching process is more complicated since the arguments of the literals must also be compared.



Once again consider

man(x)

and ¬ man(bobby)

Since man(x) and ¬ man (bobby) can be unified (for x = bobby) the above two expressions are unifiable. This corresponds to the intuition that man(r) cannot be true for all x if there is known to be some x, say bobby, for which man (x) is false. Thus, in order to use resolution for expressions in predicate logic to draw inference (theorem proving) rules we use the unification algorithm of locating pairs to literals which cancel out.

Let us now assume that we want to resolve two clauses:

1. man (Daryodana)

2. ¬ man (x) v mortal (x)

The literal man (Daryodana) can be unified with the literal ¬ man(x) with substitution Daryodana/x, meaning thereby that if x = Daryodana is true and therefore ¬ man (Daryodana) is false. But we cannot simply cancel out the two man literals as we had done in the propositional logic and generate the resolvent mortal (x).

Clause 2, here states that for x, either ¬ man(x) or mortal (x) is true. So, for it to be true, we can now conclude that only that mortal (Daryodana) must be true. It is necessary that mortal (x) be true for all x, since for some values of Man(x) might be true, making mortal (x) irrelevant to the truth of the complete clause.

So the resolvent generated by clauses 1 and 2 must be mortal (Daryodana) which we get by applying the result of the unification process to the resolvent. The resolution process can then proceed from there to discover whether mortal (Daryodana) leads to a contradiction with other available clauses.

With the help of this example, we have illustrated how the variables are standardised in the process of converting the expression in to clause form. This standardisation of variables also makes easy the use of the unifier to perform substitutions in order to create the resolvent. If two instances of the same variable occur they must be given identical substitutions.

Now the resolution Algorithm can be easily stated for theorem proving in FOL assuming a set of given set of statements F and a statement P to be proved.

Algorithm Resolution:

1. Convert all the statements of F to clause form.

2. Negative P and convert the result to clause form. Add it to the set of clauses obtained in 1.

3. Repeat until either a contradiction is found, no progress can be made or a predetermined amount of effort has been spent.

i. Select two clauses. Call these the parent clauses.

ii. Resolve them together. The resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitutions performed and with the following exception. If there is one pair of literals T_1 and T_2 such that one of the parent clauses contains T_1 and the other contains T_2 and if T_1 and T_2 are unifiable, then neither T_1 nor T_2 should appear in the resolvent. T_1 and T_2 are called complimentary literals. Use the substitution produced by the unification to create the resolvent. If there is more than one pair of complimentary literals, only one pair should be omitted from the resolvent.

iii. If its resolvent is empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

Example:

Theorem Proving in FOL, with Resolution Principle,:

Suppose, we have to prove a theorem Th from a set of axioms. We denote it by

$(A_1 A_2, \ldots, A_n) = \text{Th}$

Let

A_1 = Biscuit (coconut-crunchy)

A_2 = Child (mary) ∧ Takes (mary, coconut-crunchy)

A_3 = ∀X (Child (X) ∧ □Y (Takes (X, Y) a Biscuit (Y))) → Loves (john, X)

and A_4 = Th = Loves (john, mary) A_4

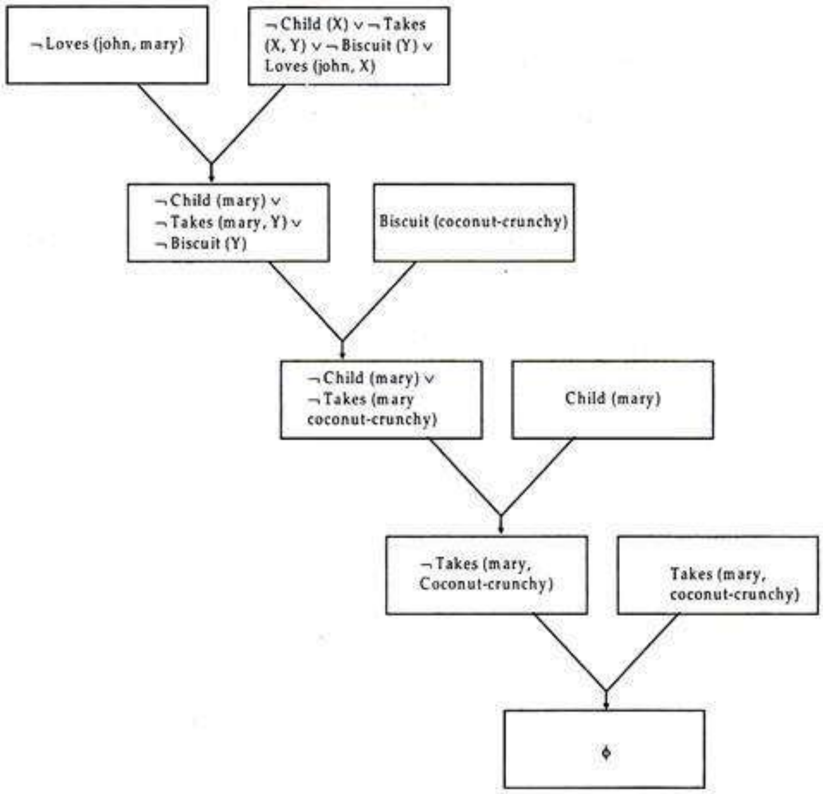
First of all, let us express A_1 through A_4 in CNF. Expressions A_1 and A_4 are already in CNF. Expression A_2 can be converted into CNF by breaking it into two clauses:
Child (mary) and

Takes (mary, coconut-crunchy)

Further, the CNF of expression A_3 is becomes after simplification
¬ Child (X)∨ ¬ Takes (X, Y) ∨ ¬ Biscuit (Y) ∨ Loves (john, X)

Explanation:

In order to prove the theorem we have formed pairs of clauses, one of which contains a positive predicate and the other one of the same predicate in the negative form. By Robinson's rule, to be proved subsequently both the complimentary predicates will drop out. Suitable values of the variables used for unification should be substituted in the resulting expression. The resolution tree is shown in Fig. 6.4 to prove the theorem (Goal).


¬Loves (john, mary)
¬Child (X) v ¬Takes (X, Y) v ¬Biscuit (Y) v Loves (john, X)
¬Child (mary) v ¬Takes (mary, Y) v ¬Biscuit (Y)
Biscuit (coconut-crunchy)
¬Child (mary) v ¬Takes (mary coconut-crunchy)
Child (mary)
¬Takes (mary, Coconut-crunchy)
Takes (mary, coconut-crunchy)
ϕ

John loves Mary: loves (john, mary).

5.9 PRACTICE EXERCISE

Generate Predicates

1. Mary loves everyone.
2. Everyone loves everyone except himself
3. Every student smiles.
4. Every student walks or talks
5. Every boy who loves Mary hates every other boy who Mary loves.

B.Sc.(DATA SCIENCE)

SEMESTER-II

INTRODUCTION TO LOGIC

UNIT VI: FUZZY LOGIC

STRUCTURE

6.0 OBJECTIVES

To Understand Basic concepts of fuzzy set theory, operations and Properties of fuzzy sets To know the working of Crisp relations fuzzy relational equations and operations on fuzzy relations

6.1 FUZZY SET

The word "fuzzy" means "vagueness". Fuzziness occurs when the boundary of a piece of information is not clear-cut. Fuzzy sets have been introduced by Lotfi A. Zadeh (1965) as an extension of the classical notion of set. Classical set theory allows the membership of the elements in the set in binary terms, a bivalent condition - an element either belongs or does not belong to the set. Fuzzy set theory permits the gradual assessment of the membership of elements in a set, described with the aid of a membership function valued in the real unit interval [0, 1].

Example:

- Words like young, tall, good, or high are fuzzy.
- There is no single quantitative value which defines the term young.
- For some people, age 25 is young, and for others, age 35 is young.
- The concept young has no clean boundary.
- Age 1 is definitely young and age 100 is definitely not young;
- Age 35 has some possibility of being young and usually depends on the context in which it is being considered.

In real world, there exists much fuzzy knowledge;

- Knowledge that is vague, imprecise, uncertain, ambiguous, inexact, or probabilistic in nature.

- Human thinking and reasoning frequently involve fuzzy information, originating from inherently inexact human concepts. Humans, can give satisfactory answers, which are probably true.

- However, our systems are unable to answer many questions. The reason is, most systems are designed based upon classical set theory and two-valued logic which is unable to cope with unreliable and incomplete information and give expert opinions.

6.2 CLASSICAL SET THEORY

A Set is any well defined collection of objects. An object in a set iscalled an element or member of that set.

- Sets are defined by a simple statement describing whether a particular element having a certain property belongs to that particular set.

- Classical set theory enumerates all its elements using

  A = { a1 , a2 , a3 , a4 , an }

- If the elements ai (i = 1, 2, 3, . . . n) of a set A are subset of universal set X, then set A can be represented for all elements $x \in X$ by its characteristic function

$$\mu_A(x) = \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{otherwise} \end{cases}$$

- A set A is well described by a function called characteristic function

This function, defined on the universal space X, assumes :

a value of 1 for those elements x that belong to set A, and

a value of 0 for those elements x that do not belong to set A.

The notations used to express these mathematically are

$$\left.\begin{array}{l} A : X \text{ -> } [0, 1] \\ A(x) = 1 \text{ , } x \text{ is a member of } A \\ A(x) = 0 \text{ , } x \text{ is not a member of } A \end{array}\right\} \quad \text{Eq.(1)}$$

Alternatively, the set A can be represented for all elements $x \in X$ by its characteristic function $\mu_A(x)$ defined as

$$\mu_A(x) = \begin{cases} 1 & \text{if } \quad x \in X \\ 0 & \text{otherwise} \end{cases} \qquad \text{Eq.(2)}$$

- Thus in classical set theory $\mu_A(x)$ has only the values 0 ('false') and 1 ('true"). Such sets are called crisp sets.

6.3 FUZZY SET THEORY

Fuzzy set theory is an extension of classical set theory where elements have varying degrees of membership. A logic based on the two truth values, *True* and *False*, is sometimes inadequate when describing human reasoning. Fuzzy logic uses the whole interval between 0 (false) and 1 (true) to describe human reasoning

- A Fuzzy Set is any set that allows its members to have different degree of membership, called membership function, in the interval[0 , 1].

- The degree of membership or truth is not same as probability;
  - fuzzy truth is not likelihood of some event or condition.
  - fuzzy truth represents membership in vaguely defined sets
  - Fuzzy logic is derived from fuzzy set theory dealing with reasoning that is approximate rather than precisely deduced from classical predicate logic.
  - Fuzzy logic is capable of handling inherently imprecise concepts

- Fuzzy logic allows in linguistic form the set membership values to imprecise concepts like "slightly", "quite" and "very".

- Fuzzy set theory defines Fuzzy Operators on Fuzzy Sets

6.4 CRISP AND NON-CRISP SET

- As said before, in classical set theory, the characteristic function $\mu_A(x)$ of Eq.(2) has only values 0 ('false') and 1 ('true''). Such sets are crisp sets.
- For Non-crisp sets the characteristic function $\mu_A(x)$ can be defined
- The characteristic function $\mu_A(x)$ of Eq. (2) for the crisp set is generalized for the Non-crisp sets.
- This generalized characteristic function $\mu_A(x)$ of Eq.(2) is called membership function.

Such Non-crisp sets are called Fuzzy Sets.

- For Non-crisp sets the characteristic function $\mu_A(x)$ can be defined. The characteristic function $\mu_A(x)$ of Eq. (2) for the crisp set is generalized for the Non-crisp sets
- This generalized characteristic function $\mu_A(x)$ of Eq.(2) is called membership function.

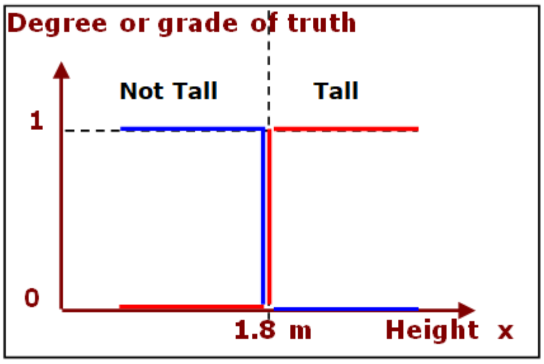
Such Non-crisp sets are called Fuzzy Sets.

- Crisp set theory is not capable of representing descriptions and classifications in many cases; In fact, Crisp set does not provide adequate representation for most cases.

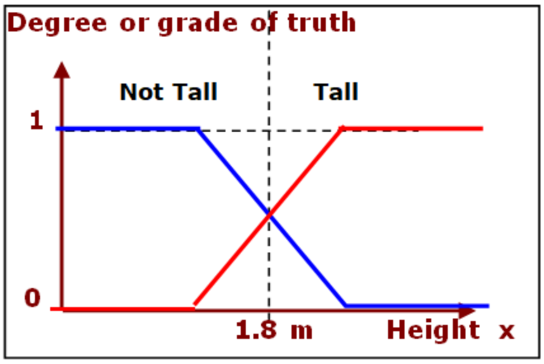
Representation of Crisp and Non-Crisp Set Example: Classify students for a basketball teamThis example explains the grade of truth value.

- tall students qualify and not tall students do not qualify
- if students 1.8 m tall are to be qualified, then should we exclude a student who is $^{1}/_{10}$" less? orshould we exclude a student who is 1" shorter?

Non-Crisp Representation to represent the notion of a tall person



Crisp logic



Non-crisp logic

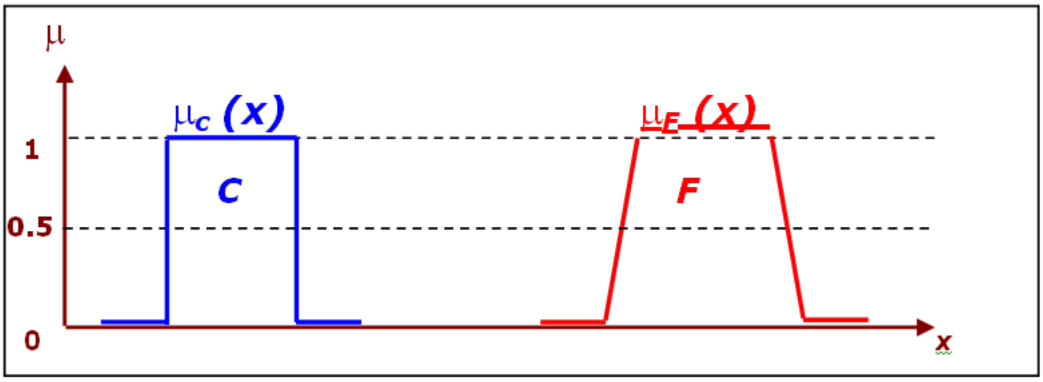
A student of height 1.79m would belong to both tall and not tall sets with a particular degree of membership.

As the height increases the membership grade within the tall set wouldincrease whilst the membership grade within the not-tall set would decrease

6.5 CAPTURING UNCERTAINTY

Instead of avoiding or ignoring uncertainty, Lotfi Zadeh introduced FuzzySet theory that captures uncertainty.

- A fuzzy set is described by a membership function μ_A (x) of A.
- This membership function associates to each element $x \in X$ a number as μ_A (x_σ) in the closed unit interval [0, 1]. The number μ_A (x_σ) represents the degree of membership of x_σ in the notation used for membership function μ_A (x) of a fuzzy set A is $A : X \to [0, 1]$
- Each membership function maps elements of a given universal base set X , which is itself a crisp set, into real numbers in [0, 1] .
- Example



- In the case of Crisp Sets the members of a set are : either out of the set, with membership of degree " 0 ",or in the set, with membership of degree " 1 ",

Therefore, Crisp Sets ⊆ Fuzzy Sets

In other words, Crisp Sets are Special cases of Fuzzy Sets.

6.6 FUZZY SET

A Fuzzy Set is any set that allows its members to have different degree of membership, called membership function, in the interval [0 , 1].

A fuzzy set A, defined in the universal space X, is a function defined in X which assumes values in the range [0, 1].

A fuzzy set A is written as a set of pairs {x, A(x)} as

A = {{x , A(x)}} , x in the set X

Where x is an element of the universal space X, and A(x) is the value of the function A for this element.

The value A(x) is the membership grade of the element x in a fuzzy set A.

6.6.1 Universal Space

Originally the universal space for fuzzy sets in fuzzy logic was defined only on the integers

Example : Set SMALL in set X consisting of natural numbers □ to 12. Assume: SMALL(1) = 1, SMALL(2) = 1, SMALL(3) = 0.9, SMALL(4) = 0.6,

SMALL(5) = 0.4, SMALL(6) = 0.3, SMALL(7) = 0.2, SMALL(8) = 0.1, SMALL(u) = 0 for. Now, the universal space for fuzzy sets and fuzzy relations is defined with three numbers.
The first two numbers specify the start and end of the universal space, and the third argument specifies the increment between elements. This gives the user more flexibility in choosing the universal space.

Example : The fuzzy set of numbers, defined in the universal space
X = { xi } = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12} is presented as
SetOption [FuzzySet, UniversalSpace □ {1, 12, 1}]

6.7 FUZZY MEMBERSHIP

A fuzzy set A defined in the universal space X is a function defined in X which assumes values in the range [0, 1].
A fuzzy set A is written as a set of pairs {x, A(x)}.
A = {{x , A(x)}} , x in the set X
where x is an element of the universal space X, and A(x) is the value of the function A for this element.

The value A(x) is the degree of membership of the element x in a fuzzy set A.
The Graphic Interpretation of fuzzy membership for the fuzzy sets : Small, Prime Numbers, Universal-space, Finite and Infinite UniversalSpace, and Empty are illustrated in the next few slides.

6.7.1 Finite and Infinite Universal Space

Universal sets can be finite or infinite.

Any universal set is finite if it consists of a specific number of different elements, that is, if in counting the different elements of the set, the counting can come to an end, else the set is infinite.

Examples:

1. Let N be the universal space of the days of the week.

N = {Mo, Tu, We, Th, Fr, Sa, Su}. N is finite.

2. Let M = {1, 3, 5, 7, 9, ...}. M is infinite.
3. Let L = {u | u is a lake in a city }. L is finite.
(Although it may be difficult to count the number of lakes in a city,but L is still a finite universal set.)

6.8 FUZZY OPERATIONS

A fuzzy set operations are the operations on fuzzy sets. The fuzzy set operations are generalization of crisp set operations. Zadeh [1965] formulated the fuzzy set theory in the terms of standard operations: Complement, Union, Intersection, and Difference.

In this section, the graphical interpretation of the following standard fuzzy set terms and the Fuzzy Logic operations are illustrated:

Inclusion : FuzzyInclude [VERYSMALL, SMALL]
Equality : FuzzyEQUALITY [SMALL, STILLSMALL]
Complement : FuzzyNOTSMALL = FuzzyCompliment
[Small]Union : FuzzyUNION = [SMALL ∪ MEDIUM]
Intersection : FUZZYINTERSECTON = [SMALL ∪ MEDIUM]

- Inclusion
  - Let A and B be fuzzy sets defined in the same universal space X.
  - The fuzzy set A is included in the fuzzy set B if and only if for every x inthe set X we have A(x) □ B(x)
- Comparability

  Two fuzzy sets A and B are comparableif the condition A ⊂ B or B ⊂ A holds, ie, if one of the fuzzy sets is a subset of the other set, they are comparable.
  Two fuzzy sets A and B are incomparableIf the condition A ⊄ B or B ⊄ A holds.

- Equality

  Let A and B be fuzzy sets defined in the same space X.

  Then A and B are equal, which is denoted X = Y

  if and only if for all x in the set X, A(x) = B(x).

- Complement

  Let A be a fuzzy set defined in the space X.

  Then the fuzzy set B is a complement of the fuzzy set A, if and only if,for all x in the set X, B(x) = 1 - A(x).

  The complement of the fuzzy set A is often denoted by A' or Ac or $\overline{A}$

Fuzzy Complement : $Ac(x) = 1 – A(x)$

- Union

  Let A and B be fuzzy sets defined in the space X.

  The union is defined as the smallest fuzzy set that contains both A and B.The union of A and B is denoted by A ∪ B.
  The following relation must be satisfied for the union operation : for all x in the set X, $(A \cup B)(x) = \text{Max}(A(x), B(x))$.

  Fuzzy Union : (A □ B)(x) = max [A(x), B(x)] for all x □ X

- Intersection

  Let A and B be fuzzy sets defined in the space X. Intersection is defined as the greatest fuzzy set that include both A and B. Intersection of A andB is denoted by A ∩ B. The following relation must be satisfied for the intersection operation : for all x in the set X, $(A \cap B)(x) = \text{Min}(A(x), B(x))$.

Fuzzy Intersection : $(A \cap B)(x) = \min [A(x), B(x)]$ for all $x \cap X$

- Difference

  Let A and B be fuzzy sets defined in the space X. The difference of A and B is denoted by A ∩ B'.
  Fuzzy Difference : $(A - B)(x) = \min [A(x), 1- B(x)]$ for all $x \cap X$.
  Example : Difference of MEDIUM and SMALL

6.9 FUZZY PROPERTIES

Properties related to Union, Intersection, Differences are illustrated below.

- Properties Related to Union
  The properties related to union are :
  Identity, Idempotence, Commutativity and Associativity.

  - Identity:

    $A \cup \Phi = A$
    input = Equality [SMALL ∪ EMPTY , SMALL]
    output = True

    $A \cup X = X$

input = Equality [SMALL □ UnivrsalSpace, UnivrsalSpace]output = True

  - Idempotence :

    $A \cup A = A$

input = Equality [SMALL □ SMALL , SMALL]
output = True

- Commutativity :

  $A \cup B = B \cup A$

  input = Equality [SMALL ∪ MEDIUM, MEDIUM ∪ SMALL]

  output = True

- Associativity:

  $A \cup (B \cup C) = (A \cup B) \cup C$

  input = Equality [Small ∪ (Medium ∪ Big) , (Small ∪ Medium) ∪ Big] output = True

- Properties Related to Intersection

Absorption, Identity, Idempotence, Commutativity, Associativity.

- Absorption by Empty Set :

  $A \cap \Phi = \Phi$

  Input = Equality [Small ∩ Empty , Empty] output = True

- Identity :

  $A \cap X = A$

  input = Equality [Small ∩ UnivrsalSpace , Small] output = True

- Idempotence :

  $A \cap A = A$

  input = Equality [Small ∩ Small , Small] output = True

- Commutativity :

  $A \cap B = B \cap A$

  Input = Equality [Small ∩ Big , Big ∩ Small] output = True

- Associativity :

  $A \cap (B \cap C) = (A \cap B) \cap C$

  input = Equality [Small ∩ (Medium ∩ Big), (Small ∩ Medium) ∩ Big] output = True

6.10 FUZZY SYSTEMS

- Fuzzy Systems include Fuzzy Logic and Fuzzy Set Theory.

- Knowledge exists in two distinct forms :
  - the Objective knowledge that exists in mathematical form is used inengineering problems; and

- the Subjective knowledge that exists in linguistic form, usuallyimpossible to quantify.

Fuzzy Logic can coordinate these two forms of knowledge in a logical way.

- Fuzzy Systems can handle simultaneously the numerical data and linguistic knowledge.

- Fuzzy Systems provide opportunities for modeling of conditions which are inherently imprecisely defined.
- Many real world problems have been modeled, simulated, andreplicated with the help of fuzzy systems.
- The applications of Fuzzy Systems are many like : Information retrievalsystems, Navigation system, and Robot vision.
- Expert Systems design have become easy because their domains are inherently fuzzy and can now be handled better;

examples : Decision-support systems, Financial planners, Diagnostic system, and Meteorological system.

Any system that uses Fuzzy mathematics may be viewed as Fuzzy system.

The Fuzzy Set Theory - membership function, operations, properties and the relations have been described in previous lectures. These are the prerequisites for understanding Fuzzy Systems. The applications of Fuzzy set theory is Fuzzy logic which is covered in this section.

Here the emphasis is on the design of fuzzy system and fuzzy controller in a closed–loop. The specific topics of interest are :

Fuzzification of input information,

Fuzzy Inferencing using Fuzzy sets ,

De-Fuzzification of results from the Reasoning process, and

Fuzzy controller in a closed–loop.

Fuzzy Inferencing, is the core constituent of a fuzzy system. A block schematic of Fuzzy System is shown in the next slide. Fuzzy Inferencing combines the facts obtained from the Fuzzification with the fuzzy rule base and conducts the Fuzzy Reasoning Process.

6.10.1 Fuzzy System Elements

- Input Vector :X = $[x_1 , x_2, \ldots x_n]^T$ are crisp values, which aretransformed into fuzzy sets in the fuzzification block.
- Output Vector : Y = $[y_1 , y_2, \ldots y_m]^T$ comes out from the defuzzification block, which transforms an output fuzzy set back to a crisp value.

- Fuzzification : a process of transforming crisp values into grades of membership for linguistic terms, "far", "near", "small" of fuzzy sets.
- Fuzzy Rule base : a collection of propositions containing linguistic variables; the rules are expressed in the form:

  If (x is A) AND (y is B). . . . . .THEN (z is C)

  where x, y and z represent variables (e.g. distance, size) and

  A, B and Z are linguistic variables (e.g. `far', `near', `small').
- Membership function : provides a measure of the degree of similarity of elements in the universe of discourse U to fuzzy set.
- Fuzzy Inferencing : combines the facts obtained from the Fuzzificationwith the rule base and conducts the Fuzzy reasoning process.
- Defuzzyfication: Translate results back to the real world values.

6.11 FUZZY LOGIC

A simple form of logic, called a two-valued logic is the study of "truth tables" and logic circuits. Here the possible values are true as 1, and false as 0.

This simple two-valued logic is generalized and called fuzzy logic which treats "truth" as a continuous quantity ranging from 0 to 1.

Definition : Fuzzy logic (FL) is derived from fuzzy set theory dealing with reasoning that is approximate rather than precisely deduced from classical two-valued logic.

- FL is the application of Fuzzy set theory.
- FL allows set membership values to range (inclusively) between 0 and 1.
- FL is capable of handling inherently imprecise concepts.
- FL allows in linguistic form, the set membership values to imprecise concepts like "slightly", "quite" and "very".

B.SC.(DATA SCIENCE)

SEMESTER-II

INTRODUCTION TO LOGIC

UNIT VII: PROLOG

STRUCTURE

7.0 OBJECTIVES

The objective of the unit is that the students should have enough understanding with Prolog to be able to practice any undergraduate course that makes use of Prolog. At the end of the course student should be:

- familiar with basic syntax of the language
- able to apply the basic programming techniques
- familiar with idea of program as data
- able to understand the fundamental ideas of predicate calculus

7.1 INTRODUCTION: VARIABLES AND ATOMS

Prolog stands for programming in logic is one of the classical programming languages that emerged in the early1970s to use specifically for applications in AI. In comparison to the imperative languages such as C or Java that happens to be object oriented, Prolog is a declarative programming language. The academic computer science community considers Prolog an important language because it is about executable logic. This means that to solve a problem the user has to specify the situation (rules and facts) and the goal (query) and let the Prolog interpreter get the solution. Prolog programming consists of small set of basic mechanism like pattern matching, tree-base data structuring and automatic back tracking etc. Prolog is based on First Order Predicate Logic (FOPL) that implies the presence of predicate symbols along with set of connectives. Prolog is very useful for problem solving tasks in different domains of AI like search, planning, and knowledge representation.

The Prolog system identifies the type of an object in the program by its syntactic form. This is possible due the syntax of Prolog that specifies different forms for each type of data objects as shown in Fig. 3.1.

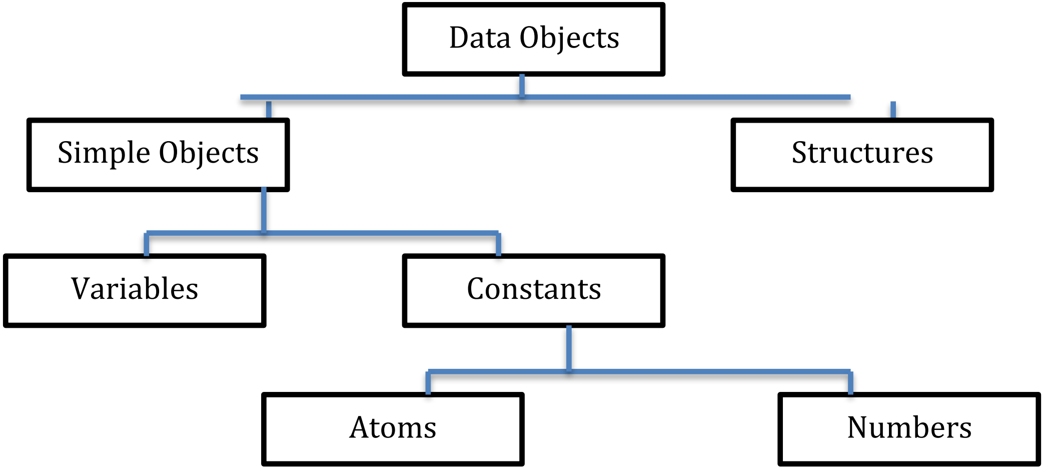


Fig 3.1: Data Objects in Prolog

A variable is written as a sequence of letters and digits, beginning with an upper-case letter or underscore (_). Variables are assumed to be universally quantified and read as ‘for all'. A single underscore (_) denotes an anonymous variable and means "any term". Unlike other variables, the underscore does not represent the same value everywhere it occurs within a predicate definition.

Examples:

X, Sister, _, _thing, _y47, First_name, Z2

The variable _ is used as a "don't-care" variable, when we don't mind what value the variable has. For example:

```
is_a_parent(X) :- father_of(X, _).
is_a_parent(X) :- mother_of(X, _).
```

That is, X is a parent if they are a father or a mother, but we don't need to know who they are the father or mother of, to establish that they are a parent.

Atoms are usually strings consist of lower (a,b, …., z) and uppercase letters (A, B, …, Z), digits (0,1,2, …, 9) and the underscore (_), starting with a lowercase letter. Atoms can be constructed in 3 ways:

1. Strings of letters, digits and the underscore (_) starting with a lower-case letter.
2. Strings of special characters (+, - ,*, /,<, >, =, : ., &, _): When using atoms of this form, some care is necessary because some strings of special characters already have a predefined meaning.
3. Strings of characters enclosed in single quotes. This is useful if we want, for example, to have an atom that starts with a capital letter. By enclosing it in quotes we make it distinguishable from variable: 'India' 'Sohan'.

7.2 FACTS AND PREDICATES

Prolog programs are made up of facts and rules. Facts and rules are also called clauses that define predicates. A fact must start with predicate expression followed by a full stop that makes a declarative statement about the problem domain. Facts consist of either a particular item or a relation between items.

For examples:

```
likes(vijay, rani).                 /* Vijay likes Rani */

likes(X, rani).                     /* Everyone likes Rani */

likes(vijay, Y).                    /* Vijay likes everybody */

likes(vijay, Y), likes(Y, vijay).       /* Vijay likes everybody and everybody likes Vijay */

likes(vijay, rani); likes(vijay,rekha). /* Vijay likes Rani or Vijay likes Rekha */

not(likes(vijay,pizza)).              /* Vijay does not like pizza */

likes(vijay,rani) :- likes(vijay,rekha)./* Vijay likes Rani if Vijay likes Rekha..
```

Predicate may be followed by one or more arguments which are enclosed by parentheses. The arguments can be atoms, numbers, variables or lists. Arguments are separated by commas. The predicate name is a Prolog atom. Each argument is an arbitrary Prolog term. The predicate of the fact describes a property of the objects on considering the arguments in a fact to be objects. The intuitive meaning of a fact defines a certain instance of a relation as being true. In a Prolog program, a presence or absence of a fact indicates a statement that is true or not true respectively

For example:

```
sunny.
father(vijay, prince).
father(vijay, rekha).
mother(sujata, prince).
```

Goals/Queries and their results:

```
?- sunny.    /* The response is yes because the fact "sunny." is present. */
yes
?- rainy.    /* There is an error because there is no predicate "rainy". */
Erroneous result.
?- father(vijay, rekha).
yes
?- mother(sujata, rekha). /* This cannot be deduced. */
no
?- father(vijay, sujata). /* This cannot be deduced. */
no
```

Names of relationship and objects must begin with a lowercase letter. Relationship is written first (typically the predicate of the sentence). Objects are written separated by commas and are enclosed by a pair of round brackets. The full stop character '.' must come at the end of a fact.

7.3 DATA TYPES

The different types of data types in Prolog are listed below:

1. **Numbers:** All the Prolog versions allow the use of integer number and real numbers. They are written as any number sequence from 0 to 9, optionally preceded by a + or - sign, for example, 262, -39, +80, 55. Most Prolog versions also allow the use of decimal point numbers. They are written in an equal way as integers, however, contain a single decimal point, anywhere barring before a non-compulsory + or - sign, e.g. 3.46, -.452, +637. Real numbers are not used very much in Prolog programming because Prolog is primarily a language for symbolic, non-numeric computation
2. **Atoms:** As discussed above atoms are constants without numerical values and can be written in three ways.
3. **Variables:** As mentioned earlier variable is a name for a term to be determined in a query. Any sequence of one or more letters (upper or lower case), numerals and underscores, starting with an upper case letter or underscore denotes the name of the variable.

4. **Compound Terms:** A variable is a name for a term to be determined in a query. Any sequence of one or more letters (upper or lower case), numerals and underscores, starting with an upper case letter or underscore denotes the name of the variable.
5. **Lists:** A list is often considered as a special type of compound term, but it is treated as a separate type of data object. Lists are written as an unlimited number of arguments enclosed in square brackets and separated by commas, e.g. [dog, cat, fish, man]. In contrast to the arity of a compound term, it is not necessary to decide in advance the number of elements a list has when a program is written.
6. **Other types:** Some dialects of Prolog allow other types of the term, e.g. character strings. As atoms, strings are sequences of Unicode code points. They have the same limits as atoms, except for the maximum length. As strings live on the global stack, their length is limited by the available stack space.

7.4 GOAL FINDING

A goal is something that Prolog tries to satisfy by finding values of the variables (in this case Student and Subject) that make the goal succeed. Prolog searches from to bottom in the database. It examines the clauses that have heads with the same functor and arity. These value(s) are bound to the variable(s). If Prolog is unable to do this, the goal fails (and Prolog will print "false" in response to the query). If all the goals in a query succeed, Prolog prints the bindings necessary to make the query succeed. Prolog will backtrack and look for another set of bindings that will satisfy the goals in the query. A query to the Prolog interpreter consists of one or more goals.

For example,

?- lectures (vijay, subject),

studies (student, subject) .

It is not mandatory to bind variables in order to satisfy a goal.

For example, there is no variable to bind, when the goal is likes (rekha, pizza) and the Prolog database already contains likes (rekha, pizza). In this case, Prolog will print "true" in response to the query, rather than printing bindings.

7.5 CLAUSES

Clauses are the structural elements of a program. Writing a collection of clauses in a text file develops a Prolog program. The clauses into the Prolog environment are loaded by specifying the name of the text file in the consult command. In other words a clause in Prolog is a unit of information in a Prolog program ending with a full stop (".").

Clauses are of two types:

1. **Facts:** As discussed above a fact is an atom or structure followed by a full stop.

   For example,

   cold., male(homer).

   father(homer,bart).

2. **Rules:** A rule consists of a head and a body. The head and body are separated by a :- and followed by full stop. If the body of a clause is true then the head of the clause is true.

   For example:

   bigger (X,Y) :- X > Y.

   parents (F,M,C) :- father (F,C), mother (M,C).

7.6 CENTRAL IDEA OF PROLOG

In this section let’s understand how Prolog works. Although in Prolog programming the programmer specify the programs in syntax close to logic but that is not how Prolog works. In fact, just understanding of logic is not useful for understanding Prolog. Prolog can be better understood as a language with two extraordinary features--unification and backtracking. Lot of applications can benefit from automatic pattern matching and backtracking search in AI and more. It makes excess use of recursion, which is more common than unification and backtracking but difficult to grasp if user have not encountered it before. Prolog's interface along with understanding of the Prolog literature, deliberately leads the developer into a conceptual model of logic and Prolog's true inner workings.

In this procedural way Prolog seems less glamorous than it as pure logic programming. However, this view leads to a deeper understanding of the beauty and power of Prolog. For example, consider the problem of building a forward-chaining (data-driven) inference engine. This is useful for building several types of complex expert systems, such as configuration and scheduling systems. The inference engine goes though a cycle of looking for rules whose conditions are met, and then taking the appropriate actions.

Prolog handles these easily with unification, backtracking, and recursion. A simple forward-chaining inference engine is implemented with these few lines of code:

```
forward_chain :-
rule(conditions(X), actions(Y)),
call(X),
call(Y),
forward_chain.
```

The search for rules that match proceeds via unification and backtracking, making the most complex part of the inference engine very simple. Recursion handles the looping. Once a developer grasps a good conceptual model of how Prolog really works, all of the language's possibilities become clear.

7.7 EXECUTION OF PROLOG

In Prolog goals were proved by matching patterns of a query goal with the patterns of the clauses in a Prolog program. This is done by finding a clause whose head matches the query goal and then trying to prove the goals, if any, in the clause's body.

Prolog needs a method for matching the goal it is currently trying to match against heads of clauses. When the head of a clause and the current goal match, the clause is chosen and Prolog goes on to try and prove the goals in its body (if any). The act of matching two Prolog terms is called unification. The rules for the same are written below:

1. Two atoms unify if they are the same.
2. Two numbers unify if they have the same value.
3. Two structures unify if they have the same name and arity, and each pair of respective arguments unify.
4. Two lists unify if their initial elements unify.
5. A variable unifies with a non-variable by being replaced by the non-variable. This is known as binding.
6. Two variables unify by agreeing to "share" bindings. This means that if later on, one or the other unifies with another term, then both unify with the term.
7. Two strings unify if and only if they have precisely the same characters in them.
8. A string and an atom will unify if they have precisely the same characters in them.

When a clause is under consideration to match against a goal, space is reserved to hold variable bindings. If the clause is chosen again later on in the proof, then new space is reserved for the variable bindings caused by the new choice.

7.8 BACKTRACKING

The backtracking term is very common in different programming environments for algorithm designing. In Prolog program the backtracking is a procedure, that searches the true values of different predicates by checking whether they are correct or not. It tries to backtrack until it reaches proper destination and when the destination is found, it stops. The Figure 3.2 shows how backtracking takes place using one tree like structure −

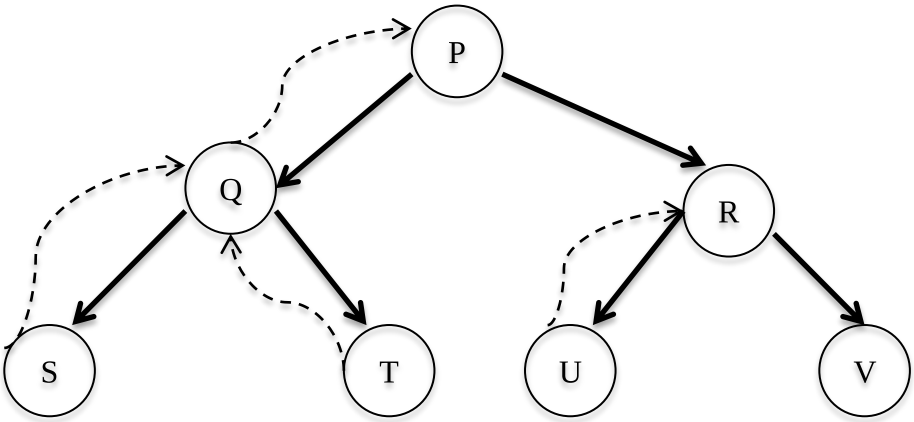


Fig 3.2: Backtracking

Let P to V are some rules and facts. It start from P and want to reach V. The proper path will be P-R-V, but at first, it will go from P to Q, then Q to S. When it finds that S is not the destination, it backtracks to Q, then go to T, and backtracks again to Q, as there is no other child of Q, then it backtracks to P, thus it searches for V, and finally found V in the path P-R-V. Dashed lines are indicating the backtracking. So when it finds V, it stops. It is worthwhile to mention that during backtracking there may be multiple answers, then press semicolon (;)

to get next answers one by one, that helps to backtrack. Otherwise when we get one result, it will stop.

Now after understanding backtracking in Prolog. For example:- Consider a situation, where two people A and B can pay each other, but the condition is that a Man can pay to a Lady, so A will be a Man, and B will be a Lady. So for these lets defined some facts and rules – (knowledge base)

```
man ( sunny ) .

man ( bob ) .

lady ( loyna ) .

lady ( lili ) .

pay ( A, B ) :- man ( A )
```

The illustration of the above scenario is shown in Figure 3.3.

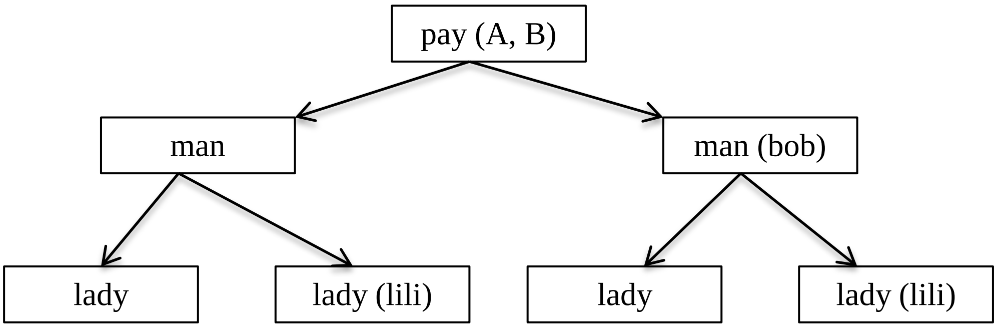


Fig 3.3: Tree representation of example

Output:

```
| ?- [backtrack].

compiling D:/TP Prolog/Sample_Codes/backtrack.pl for byte code...

D:/TP Prolog/Sample_Codes/backtrack.pl compiled, 5 lines read - 703 bytes written, 22 ms

Yes

| ?- pay(A, B).

A = sunny

B = loyna ?

(15 ms) yes

| ?- pay(A,B).

A = sunny

B = loyna ? ;

A = sunny B = lili ? ;

A = bob B = loyna ? ;
```

```
A = bob B = lili
yes
| ?- trace.
The debugger will first creep -- showing everything (trace)
(16 ms) yes
{trace}
| ?- pay(A,B).
1 1 Call: pay(_23,_24) ?
2 2 Call: man(_23) ?
2 2 Exit: man(sunny) ?
3 2 Call: lady(_24) ?
3 2 Exit: lady(loyna) ?
1 1 Exit: pay(sunny,loyna) ?
A = sunny
B = loyna ? ;
1 1 Redo: pay(sunny,loyna) ?
3 2 Redo: lady(loyna) ?
3 2 Exit: lady(lili) ?
1 1 Exit: pay(sunny,lili) ?
A = sunny B = lili ? ;
1 1 Redo: pay(sunny,lili) ?
2 2 Redo: man(sunny) ?
2 2 Exit: man(bob) ?
3 2 Call: lady(_24) ?
3 2 Exit: lady(loyna) ?
1 1 Exit: pay(bob,loyna) ?
A = bob
B = loyna ? ;
1 1 Redo: pay(bob,loyna) ?
3 2 Redo: lady(loyna) ?
3 2 Exit: lady(lili) ?
1 1 Exit: pay(bob,lili) ?
```

```
A = bob

B = lili

yes

{trace}

| ?-
```

7.9 SIMPLE OBJECT

In this section the different types of data structures and data objects that a Visual Prolog program can contain are discussed. A simple data object is either a variable or a constant. Don’t confuse the word “constant” with the symbolic constants, defined in the constant section of a program. Here a constant, is anything identifying an object not subject to variation, such as character (a char), a number (an integral value or a real), or an atom (a symbol or string).

4. **Variables as Data objects:** As discussed earlier that variables must begin with an upper-case letter (A-Z) or an underscore (_). Prolog variables are local, not global. That is, if two clauses each contain a variable called X, these Xs are two distinct variables. They may get bound to each other if they happen to be brought together during unification, but ordinarily they have no effect on each other.
4. **Constants as Data Objects:** Constants refer the characters, numbers, and atoms. A constant's value is its name. That is, the constant 1 can only stand for the number 1, and the constant abracadabra can only stand for the symbol abracadabra. Characters are the digits 0-9, upper-case letters A-Z, lower-case letters a-z, and the punctuation. Characters outside this range may not be portable between different platforms; in particular, characters less than ASCII 32 (space) are control characters, traditionally used by terminals and communication equipment. Numbers and atoms are already discussed above.

7.10 COMPOUND DATA OBJECTS

Compound data objects allow you to treat several pieces of information as a single item in such a way that you can easily pick them apart again. Consider, for instance, the date June 29, 2021. It consists of three pieces of information--the month, day, and year, but it's useful to treat the whole thing as a single object with a treelike structure:

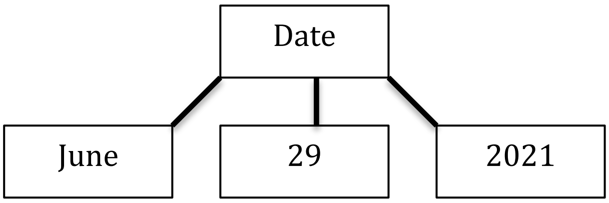


Fig 3.4: Compound data object: date

Now let us declare a domain containing the compound object date:

```
DOMAINS
 date_cmp = date(string,unsigned,unsigned)
```

and then simply writing e.g.

```
..., D = date("June",29,2021), ...
```

This looks like a Prolog fact, but it is not, it's just a data object, which can be handle the same way as a symbol or number. It begins with a name, usually called a functor (here date), followed by three arguments.

Remember a functor in Visual Prolog has nothing to do with a function as in other programming languages. A functor does not stand for some computation to be performed. It is just a name that identifies a kind of compound data object and holds its arguments together.

Reference:

1. Ivan Bratko. Prolog: Programming For Artificial Intelligence, 3/E. India: Pearson Education, 2001.
2. Clocksin, William F., Mellish, Christopher S. Programming in Prolog: Using the ISO Standard. Germany: Springer London, Limited, 2012.
3. Bramer, Max. Logic Programming with Prolog. United Kingdom: Springer London, 2005.

Exercise: Solved programs in Prolog

1. **Write simple fact for the statements using PROLOG**

a. Sham likes banana.

b. Reema is a girl.

c. Marry likes Candy.

d. Sky is blue.

e. Sohan owns diamond.

Program:

```
Clauses
likes(sham ,banana).
girl(Reema).
blue(sky).
likes(Marry ,candy).
owns(sohan , diamond).
```

Output:

```
Goal
queries
?-likes(sham,X).
X= banana
```

```
?-likes(Y,candy).
Y= candy
?-blue(X).
X= sky
?-owns(Y,X).
Y= Sohan
X= diamond.
```

OUTCOME: The student will understand the concept of how to write simple facts using prolog.

2. Write predicates One converts centigrade temperatures to Fahrenheit, the other checks if a temperature is below freezing.

Program:

Production rules:

Arithmetic:

```
c_to_f f is c * 9 / 5 +32
freezing f < = 32
```

Rules:

```
c_to_f(C,F) :-
F is C * 9 / 5 + 32.
freezing(F) :-
F =< 32.
```

Output:

Queries:

```
?- c_to_f(95,X).
X = 203
?- freezing(12)
 true
?- freezing(44).
 false
```

OUTCOME: The student will understand the concept of how to write a program using the rules.

3. Program to read the address of a person using compound variable.

```
domains
    student=address(name,houseno,city,state,pin)
    name,houseno,city,state,pin =String
```

```
predicates
     readaddress(student)
     go
clauses
    go:-

readaddress(Address),nl,write(Address),nl,nl,write("Accept(Y/N)?"),readchar(Reply),Reply='
y',!.
    go:-
    nl,write("please re-enter"),nl,go.
    readaddress(address(name,houseno,city,state,pin)):-
    write("Name:"),readln(name),
    write("House No:"),readln(houseno),
    write("City:"),readln(city),
    write("State:"),readln(state),
    write("PIN:"),readln(pin).
```

Output:

```
Accept(Y/N?) Yes
go
Name: Harry
House No: 12/2
City: Moga
State:Punjab
PIN: 123456
address(Harry, 12/2, Moga, Punjab,123456)
```

**4.	Program to append an integer into the list.**

```
domains
     x=integer
     list=integer*
predicates
     append(x,list,list)
clauses
     append(X,[],[X]).
     append(X,[H|T],[H|T1]):-
     append(X,T,T1).
```

Output:

```
append(1,[2,3,4,5],X).
X=[2,3,4,5,1]
```

5. Program to replace an integer from the list.

```
domains
    list=integer*
predicates
    replace(integer,integer,list,list)
clauses
    replace(X,Y,[X|T],[Y|T]).
    replace(X,Y,[H|T],[H|T1]):-replace(X,Y,T,T1).
```

Output:

```
replace(1,2,[1,4,5,5,6],X).
X = [2, 4, 5, 5, 6]
replace(4,3,[2,3,4,5,6],X).
X = [2, 3, 3, 5, 6]
```

6. Program to delete an integer from the list .

```
domains
    list=integer*
predicates
     del(integer,list,list)
clauses
    del(X,[X|T],T).
    del(X,[H|T],[H|T1]):-
    del(X,T,T1).
```

Output:

```
del(3,[2,3,4],X).
X = [2, 4]
del(1,[1,2,3,4,5],X).
X = [2, 3, 4, 5]
```

7. Program to demonstrate family relationship

```
predicates
   parent(symbol,symbol)
   child(symbol,symbol)
   mother(symbol,symbol)
   brother(symbol,symbol)
```

```
    sister(symbol,symbol)
    grandparent(symbol,symbol)
    male(symbol)
    female(symbol)
clauses
    parent(a,b).
    sister(a,c).
    male(a).
    female(b).
    child(X,Y):-parent(Y,X).
    mother(X,Y):-female(X),parent(X,Y).
    grandparent(X,Y):-parent(X,Z),parent(Z,Y).
    brother(X,Y):-male(X),parent(V,X),parent(V,Y).
child(X,h).
false
female(b).
true
male(a).
true
grandparent(a,X).
false
grandparent(a,d).
false
```

7.11 PRACTICE EXERCISE: QUESTIONS FOR PRACTICE

1. What is prolog programming language? Explain its features.
2. Name the areas in which prolog programming language is used?
3. How variables are used in prolog?
4. How backtracking in prolog is done?
5. Write a Prolog predicate to remove the N’th item from a list.
6. What is the difference between the following two rules?
   blah :- a(X) , b(X).
   blah :- a(_) , b(_).
7. Define a Prolog predicate sort(X, Y) that asserts that given X is a list of integers then Y is the same list but sorted in ascending order. Your algorithm MUST be the following Repeatedly choose the smallest remaining element from X and add it to Y.

B.SC.(DATA SCIENCE)

SEMESTER-II

INTRODUCTION TO LOGIC

UNIT VIII: PROLOG OPERATIONS

STRUCTURE

8.0 OBJECTIVES

The objective of this module is that the students should understand the following concepts at the end of the course: -

- Understanding of Arithmetic operators
- Implementation of predicates and program termination
- Abel to understand the dynamic data base

8.1 ARITHMETIC OPERATORS: AN INTRODUCTION

In this module the different types of arithmetic operators and the techniques of program termination in Prolog are discussed. How these are different from any other high level language operators, how they are syntactically different, and how they are different in their work. Some practical demonstrations to understand the usage of different operators are also mentioned.

Arithmetic operators are used to perform arithmetic operations. There are few different types of arithmetic operators as shown in Table 1.1. The usage of these operators can be seen in a practical program written below:

Table 1.1: Arithmetic operators

Operator	+	-	*	/	**	//	mod
Meaning	Addition	Subtraction	Multiplication	Division	Power	Integer Division	Modulus

```
calc :- P is 75 + 75,write('75 + 75 is '),write(P),nl,

Q is 200 - 100,write('200 - 100 is '),write(Q),nl,

R is 15 * 200,write('12 * 200 is '),write(R),nl,

S is 60 / 20,write('60 / 20 is '),write(S),nl,

T is 100 // 30,write('100 // 30 is '),write(T),nl,

U is 100 ** 2,write('100 ** 2 is '),write(U),nl,

D is 100 mod 30,write('100 mod 30 is '),write(D),nl.
```

Here nl stands for newline.

Output

```
| ?- change_directory('D:/TP Prolog/Sample_Codes').

yes

| ?- [op_arith].

compiling D:/TP Prolog/Sample_Codes/op_arith.pl for byte code... D:/TP
Prolog/Sample_Codes/op_arith.pl compiled, 6 lines read - 2390 bytes written, 11 ms

yes
```

```
| ?- calc.

75 + 75 is 150

200 - 100 is 100

15 * 200 is 30000

60 / 20 is 3

100 // 30 is 3

100 ** 2 is 10000.0

100 mod 30 is 10

yes | ?-
```

8.2 PROGRAM TERMINATION

Prolog is a programming language based on logic programming. However the use of a fixed selection rule combined with depth first search in the resulting search trees makes Prolog and logic programming different. As a consequence various completeness results linking the procedural and declarative interpretation of logic programs cannot be directly applied to Prolog programs. This mismatch makes it difficult to study Prolog programs using only logic programming theory. Clearly the main problem is the issue of termination: a Prolog interpreter will miss a solution if all success nodes lie to the right of an infinite path in the search tree. Termination finiteness of all possible Prolog derivations starting in the initial goal. This stronger notion of termination abstracts from the ordering of the program clauses and seems to follow a good programming practice. When studying Prolog programs from the point of view of termination it is useful to notice that some programs terminate for all ground goals for all selection rules.

However, some Prolog programs satisfy such a strong termination property but fail to terminate for certain desired forms of inputs for some selection rules. An example is the following append3 program in which the append program is used:

```
append3 (Xs, Ys, Zs, Us),

append (Xs, Ys, Vs),

append (Vs, Zs, Us).
```

Then append3 is a terminating program which terminates for the goal~ append3 (xs, ys, zs, Us), where xs, ys, z s are lists and Us a variable, when the Prolog selection rule is used but fails to terminate when the rightmost selection rule is used.

some programs fail to be terminating even though they terminate for the Prolog selection rule for the desired class of inputs. An example is the flatten program which collects all the nodes of a tree in a list:

```
flatten(nil, []),

flatten(t(L, X, R), Xs),

flatten(L, Xls),
```

flatten(R, X2s),

append(Xls, [X I X2s], Xs).

flatten is not a terminating program but it terminates for the goal flatten (x, Xs), where x is a ground term and Xs a variable, when the Prolog selection rule is used.

8.3 USE OF CUT AND FAIL PREDICATES

The concept of backtracking was discussed in Module – III. lets see few drawbacks of backtracking. Sometimes the same predicates were used more than once as per the need of program like to make some decision making systems or to write recursive rules. In such cases uncontrolled backtracking may cause inefficiency in a program. To resolve this, the **Cut** is used in Prolog.

Suppose we have some rules as written below: -

Double step function

- Rule 1 &minnus; if A < 3 then B = 0
- Rule 2 &minnus; if 3 <= A and X < 6 then B = 2
- Rule 3 &minnus; if 6 <= A then B = 4

In Prolog syntax,

- f(A,0) :- A < 3. % Rule 1
- f(A,2) :- 3 =< X, X < 6. % Rule 2
- f(A,4) :- 6 =< X. % Rule 3

Now if we ask for a question as f (1,B), 2 < B.

The first goal f(1,B) instantiated B to 0. The second goal becomes 2 < 0 which fails. Prolog tries through backtracking two unfruitful alternatives (Rule 2 and Rule 3). On watching carefully, the observations are:

- The three rules are mutually exclusive and one of them at most will succeed.
- As soon as one of them succeeds there is no point in trying to use the others as they are bound to fail.

Now use **cut** to resolve this. The **cut** can be expressed using Exclamation symbol. The prolog syntax is as follows −

- f (A,0) :- A < 3, !. % Rule 1
- f (A,2) :- 3 =< X, X < 6, !. % Rule 2
- f (A,4) :- 6 =< X. % Rule 3

Now if we use the same question, ?- f (1,B), 2 < B. Prolog choose rule 1 since 1 < 3 and fails the goal 2 < B fails. Prolog will try to backtrack, but not beyond the point marked ! In the program, rule 2 and rule 3 will not be generated.

Program

```
f(A,0) :- A < 3. % Rule 1
f(A,2) :- 3 =<A, A < 6. % Rule 2
f(A,4) :- 6 =< A. % Rule 3
```

Output

```
| ?- [backtrack].
compiling D:/TP Prolog/Sample_Codes/backtrack.pl for byte code... D:/TP Prolog/Sample_Codes/backtrack.pl compiled, 10 lines read - 1224 bytes written, 17 ms
yes | ?- f(1,B), 2<B.
no | ?- trace .
The debugger will first creep -- showing everything (trace)
yes {trace}
| ?- f(1,B), 2<B.
1 1 Call: f(1,_23) ?
2 2 Call: 1<3 ?
2 2 Exit: 1<3 ?
1 1 Exit: f(1,0) ?
3 1 Call: 2<0 ?
3 1 Fail: 2<0 ?
1 1 Redo: f(1,0) ?
2 2 Call: 3=<1 ?
2 2 Fail: 3=<1 ?
2 2 Call: 6=<1 ?
2 2 Fail: 6=<1 ?
1 1 Fail: f(1,_23) ?
(46 ms) no
{trace}
| ?-
```

Now execute the same program using cut.

Program

```
f(A,0) :- A < 3, ! . % Rule 1
f(A,2) :- 3 =<A, A < 6, ! . % Rule 2
f(A,4) :- 6 =< A. % Rule 3
```

Output

```
| ?- [backtrack].
1 1 Call: [backtrack] ?
compiling D:/TP Prolog/Sample_Codes/backtrack.pl for byte code... D:/TP
Prolog/Sample_Codes/backtrack.pl compiled, 10 lines read - 1373 bytes written, 15 ms
1 1 Exit: [backtrack] ?
(16 ms) yes
{trace}
| ?- f(1,B), 2<B.
1 1 Call: f(1,_23) ?
2 2 Call: 1<3 ?
2 2 Exit: 1<3 ?
1 1 Exit: f(1,0) ?
3 1 Call: 2<0 ?
3 1 Fail: 2<0 ?
no
{trace}
| ?-
```

Negation as failure, when conditions do not satisfy. A statement, “Jyoti likes all fruits but oranges”, will be expressed in Prolog as. It would be very easy and straight forward, if the statement is “Jyoti likes all fruits”. In that case we can write “Jyoti likes A if A is an fruit”. And in prolog we can write this statement as, likes(jyoti, A) := fruit(A).

Our actual statement can be expressed as −

- If A is orange, then “Jyoti likes A” is not true
- Otherwise if A is an fruit, then Jyoti likes A.

In prolog this can be written as −

- likes(jyoti,A) :- orange(A), !, fail.
- likes(jyoti, A) :- fruit(A).

The ‘fail’ statement causes the failure. Now let us see how it works in Prolog.

Program

```
fruit(plum).
fruit(banana).
fruit(mango).
fruit(apple).
fruit(pear).
fruit(pineapple).
orange(pear).
orange(pineapple).
likes(jyoti, A) :- orange(A), !, fail.
likes(jyoti, A) :- fruit(A) .
```

Output

```
| ?- [negate_fail].
compiling D:/TP Prolog/Sample_Codes/negate_fail.pl for byte code... D:/TP Prolog/Sample_Codes/negate_fail.pl compiled, 11 lines read - 1118 bytes written, 17 ms
yes
| ?- likes(jyoti,mango).
yes
| ?- likes(jyoti,apple).
yes
| ?- likes(jyoti,pineapple).
no
| ?- likes(jyoti,pear).
no
| ?- trace .
The debugger will first creep -- showing everything (trace)
yes
{trace}
| ?- likes(jyoti,plum).
1 1 Call: likes(jyoti,plum) ?
2 2 Call: orange(plum) ?
2 2 Fail: orange(plum) ?
2 2 Call: fruit(plum) ?
2 2 Exit: fruit(plum) ?
1 1 Exit: likes(jyoti,plum) ?
yes
```

```
{trace}
| ?- likes(jyoti,pineapple).
1 1 Call: likes(jyoti,pineapple) ?
2 2 Call: orange(pineapple) ?
2 2 Exit: orange(pineapple) ?
3 2 Call: fail ?
3 2 Fail: fail ?
1 1 Fail: likes(jyoti,pineapple) ?
no
{trace}
| ?-
```

8.4 SATISFIABILITY: USE UNIFICATION

Unification is important concepts in Prolog that explains how the language works and operates on both basic and complex levels. Basic idea of what unification is: identical terms. That is, two terms which are exactly the same will unify. Suppose the data is:-

constants atoms (prince, a_cat, 'Mr Darcy') or numbers (42, -9, 3.14)

variables with no fixed value (X, John, _MZ40)

complex terms with the form functor(arg_1, arg_2, ..., arg_n)

For example, the atom prince unifies with prince, the number 11 unifies with 11, and the variable X unifies with X – all of these are pairs of identical terms. This also holds for complex terms: happy(prince) unifies with happy(prince). Two different terms will not unify (e.g. jack does not unify with jill).

The second part to basic definition: because variables have no fixed value, two terms can also unify if they contain variables which may be given values (instantiated) in such a way that the resultant terms would unify by the first part of the definition. Using this knowledge, we can now understand that prince and the variable X will unify, because X can be instantiated with the value prince, meaning the two terms are now the same, and will unify. In just the same way, happy(prince) and happy(P) will unify, because the variable P can be instantiated to have the value 'Prince', meaning they would be the same, and thus unify

For example: -

Prolog has a few useful built-in predicates that can make use of – one of these is used to test whether two terms will unify or not: =/2. Remember that predicates in Prolog by writing the functor, then a slash, then the arity. This built-in predicate works like any other, and so if we are at the Prolog prompt, and ask the query =(prince, prince)., Prolog will tell us true, as these atoms unify. However, asking =(prince, preet). will give us false. Writing this test like this feels a little awkward, so Prolog lets us use infix notation for this predicate, putting the equals symbol between the two terms we are testing: prince = prince. returns true, and so on. What should happen with the following queries?

1. bing = bong.

2. 2 = 2.

3. ‘prince’ = prince.

4. ‘2’ = 2.

5. prince = X.

6. X = Y.

7. J = prince, J = preet.

8. ancestor(french(jean), B) = ancestor(A, irish(prince)).

9. likes(X, X) = likes(prince, pizza).

10. father(X) = X.

8.5 RECURSION

Recursion is an extremely powerful tool widely used in Prolog in which one predicate uses itself (may be with some other predicates) to find the truth value.

Let us understand this definition with the help of an example −

- is_digesting(A,B) :- just_ate(A,B).
- is_digesting(A,B) :-just_ate(A,C),is_digesting(C,B).

So this predicate is recursive in nature. Suppose we say that just_ate(cow, grass), it means is_digesting(cow, grass) is true. Now if we say is_digesting(lion, grass), this will be true if is_digesting(lion, grass) :- just_ate(lion, cow), is_digesting(cow, grass), then the statement is_digesting(lion, grass) is also true.

There may be some other examples also, so let us see one family example. So if we want to express the predecessor logic as shown in figure 4.1.

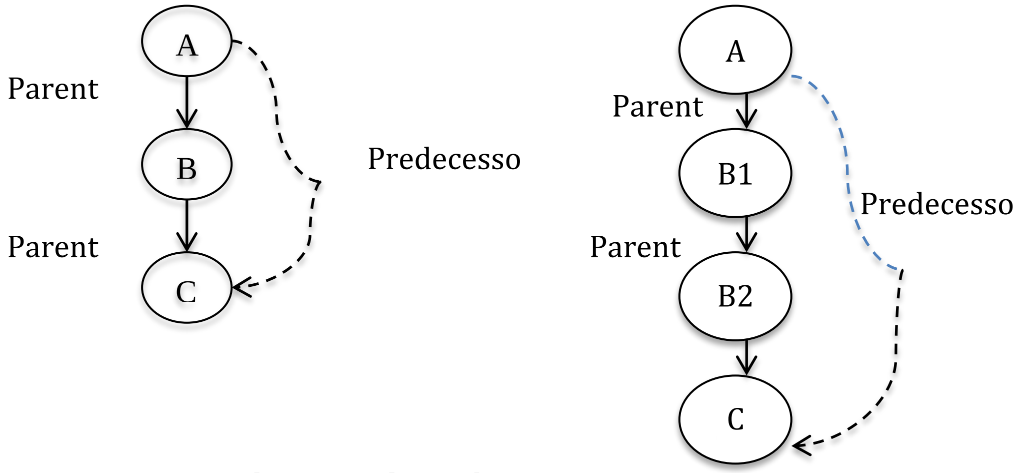


Fig 4.1: Predecessor relationship

Now it is clear that the predecessor relationship is recursive and can be expressed using the following syntax −

```
predecessor(A, C) :- parent(A, C).

predecessor(A, C) :- parent(A, B),predecessor(B, C).
```

For example:

```
on_route(india).

on_route(Place):-

        move(Place,Method,NewPlace),
        on_route(NewPlace).

move(home,car,delhi).
move(delhi,train,heathrow).
move(heathrow,plane,india).
```

on_route is a recursive predicate. This program sees if it is possible to travel to India from a particular place. The first clause sees if we have already reached India, in which case we stop. The second clause sees if there is a move from the current place, to somewhere new, and then recursive sees if the NewPlace is on_route to India. The database of moves that we can make is on the right.

Let's now consider what happens when we pose the query ?- on_route(home). This matches clause two of on_route (it can't match clause one because home and India don't unify). The second on_route clause consists of two subgoals. The first asks whether you can move from home to some new location i.e. move(home,Method,NewPlace). This succeeds with Method = car, NewPlace = delhi. This says that yes we can move from home by taking a car to delhi. Next we recursively see if we can find a route from delhi to India by doing the same thing again. This is done by executing the new subgoal on_route(delhi).

8.6 LISTS

A list in Prolog is a collection of terms, which is useful for grouping items together, or for dealing with large volumes of related data, etc. It can be used in different cases for non-numeric programming.

List consists of any number of items, for example, yellow, black, red, purple, light. It will be represented as, [yellow, black, red, purple, light]. The list of elements will be enclosed with square brackets []. The length of the list is 5.

A list can be either empty or non-empty. In the first case, the list is simply written as a Prolog atom, []. In the second case, the list consists of two things as given below −

- The first item, called the head of the list;
- The remaining part of the list, called the tail.

Suppose we have a list like: [yellow, black, red, purple, light]. Here the head is yellow and tail is [black, red, purple, light]. So the tail is another list. The fact that the Tail of a list is also a list makes it easy to see how you can recurse through all of the elements of a list.

Since the empty list cannot be split into a Head and Tail, it will often serve as a way of stopping the recursion going on forever, like a base case should.

Now consider a list, L = [x, y, z].

If Tail = [y, z] then it can also write the list L as

L = [x | Tail].

Here the vertical bar (|) separates the head and tail parts. So the following list representations are also valid −

- [x, y, z] = [x | [y, z]]
- [x, y, z] = [x, y | [z]]
- [x, y, z] = [a, y, z | [ ]]

For these properties List can be defined as −

A data structure that is either empty or consists of two parts − a head and a tail. The tail itself has to be a list.

Note that a list can be stored in knowledge base by writing a complex term with list as an argument into Prolog file.

For example: peopleList([tom, joti, prince, harry, rocky]).

The Prolog prompt can be used as a variable to extract the list:

?- peopleList(List), enumerate(List, X).

Do not be tempted to write something into knowledge base like List = [x, y, z], as this is an evaluation, and not an assignment. It will not store the list in the variable as you might expect. Use the complex term method to achieve this effect.

Table 8.1 shows the different type of basic operations on list.

Operations	**Definition**
Membership Checking	During this operation, we can verify whether a given element is member of specified list or not?
Length Calculation	With this operation, we can find the length of a list.
Concatenation	Concatenation is an operation which is used to join/add two lists.
Delete Items	This operation removes the specified element from a list.
Append Items	Append operation adds one list into another (as an item).
Insert Items	This operation inserts a given item into a list.

8.7 SIMPLE INPUT/OUTPUT

Prolog facilitates to enable the input and output either of character or of terms. It is simpler to use terms. Initially, it will be assumed that all input is from the keyboard of user and all output is at the screen of the user. But here it input and output using external files like CR-ROM or hard disk is also discussed. Just like many other built-in predicates, the predicates of input and output are unsatisfiable that means while backtracking they always fail.

To provide the input terms, user may use a built-in predicate **read/1**. This predicate takes a single argument and that argument must be a variable. When it evaluate this predicate, due to this the next term will be read from the current input stream. By default, the current input stream is the keyboard of the user.

In the input stream, at least one white space character or new line and a dot('.') follows the term. The white space and dot are not part of the term, but they are read in.

Note that if the user input is required from the keyboard, a prompt character like colon will be displayed. Before Prolog accepts the input from user, user has to press the 'return' key.

When Prolog evaluates a **read** goal, the input term is unified with the variable of the argument. If the variable of the argument is unbound, it is bound to the input value.

For example:

```
?- read(A).

: 15.

A = 15

?- read(A).

: honey.

= honey

?- read(B).

: 'example of input term'.

B = 'example of input term'

?- read(A).

: pred(x,y,z).

A = pred(x,y,z)

?- read(C).

: [x,y,pred(u,v,w),[c,b,a]].

C = [x,y,pred(u,v,w),[c,b,a]]
```

To provide the input terms, use a built-in predicate **write/1**. This predicate takes a single argument. When evaluating the predicate, the term will be written to the current output stream. By default, the current output stream is the screen of user.

Here it has been used many times a built-in predicate **n1/0**. This predicates takes no arguments. When we evaluate nl predicate, this will cause a new line to be output to the current output stream.

For example:

```
?- write('examples of output term'),nl.

examples of output term

yes

?- write(15),nl.

15

yes

?- write(pred1(x, y, z)),nl.

pred1(x, y, z)

yes

?- write([x,y,z[a,b,c,d]]),nl.

[x,y,z[a,b,c,d]]

yes

?- write('example of nl'),nl,nl,write('prolog'),nl,write('example of output')

example of nl

prolog

example of output
```

8.8 DYNAMIC DATABASE

The database is a non-logical extension to Prolog. The basic philosophy of Prolog is that it searches for a set of variable bindings that makes a query true. It uses the depth-first search, binding variables to values. If an inconsistent variable binding is detected, the system backtracks: it finds the last choice it took reverts all actions done since and continues with the next alternative. This is the core power of Prolog that give it its logical basis. Databases can be classified as static and dynamic.

Static database is a part of the program that is complied along with it. It does not change during execution of the program.

Dynamic database can change dynamically at execution time and are of two types.

Type 1: created at each execution. It grows, shrinks and is deleted at the end of program. This type of database is no longer available after program finishes its execution and is called working memory.

Type 2: These are stored in files and called database files and are consulted in any program whenever required. These are not part of any particular program and are available for use in future by different programs using system defined predicates called save and consult. While executing a Prolog program one can load database file(s) using 'consult' predicate. These files can be updated dynamically at run time and saved using 'save' predicate.

The format of predicates 'save' and 'consult' are as follows: −

save(filename) - succeeds after saving the contents of a file named 'filename'.

consult(filename) - succeeds after loading or adding all the clauses from a file stored in 'filename' in the current program being executed.

reconsult(filename) - succeeds after loading all the clauses by superseding all the existing clauses for the same predicate from a file 'filename'.

Grouping of rules and facts into partitions can be easily.

Example: Load a file named 'more' if predicates A and B succeed, C:- A, B, consult('more')

Exercise: Solved programs in Prolog

1. Program to add two numbers

```
predicates
    add
clauses
    add:- write("Input first  number:-"),
    readint(A),
    write("Input second number:-"),
    readint(B),
    C=A+B,write("Output=",C).
```

Output:

```
add
Input first number:-5
Input second number:-4
Output=9
true
```

2. Program to categorize animal characteristics.

```
predicates
    small(X)
    large(X)
   color(X,X)
clauses.
```

```
    small(aunt).
    large(elephant).
    color(buffalo,black).
    color(cow,white).
    color(X,dark):-
        color(X,black);color(X,brown).
```

Output:

```
small(X)
X=aunt
large(elephant)
true
large(Y)
Y=elephant
color(aunt,brown)
false
color(buffalo,black)
true
color(cow,white)
true
color(X,white).
X=white
```

3. Program of fun to show the concept of cut operator

```
Predicates
    fun(integer,integer)
clauses.
    fun(X,1):-X<3,!.
    fun(X,2):-X>3,X<=10,!.
    fun(X,3):-X>10,!.
```

Output:

```
fun(4,1)
false
fun(2,3)
```

```
false
fun(5,2)
false
fun(12,3)
true
```

4. Program to count the number of elements in a list.

```
domains
   x=integer
   list=integer*
predicates
   count(list,x)
clauses
   count([],0).
   count([_|T],N):-count(T,N1),N=N1+1.
```

Output:

```
count([],X).
X=0
count([1,2,3,4,5,6], X).
X=6
```

5. Program to reverse the list.

```
domains
    x=integer
    list=integer*
predicates
    append(x,list,list)
    rev(list,list)
clauses
    append(X,[],[X]).
    append(X,[H|T],[H|T1]):-append(X,T,T1).
    rev([],[]).
    rev([H|T,rev):-rev(T,L),append(H,L,rev).
```

Output:

```
append(2,[3,4,5], X)
X=[3,4,5,2]
rev([1,2,3,4],X)
X=[4,3,2,1]
```

6. Program to show concept of list.

```
domains
    name=symbol*
predicates
    itnames(name)
clauses
    itnames([sham,john,sherry]).
    itnames([sham,sherry,john]).
```

Output:

```
itnames(Y)
Y = [sham, john, sherry]
Y = [sham, sherry, john]
itnames([sham|T])
T = [john, sherry]
T = [sherry, john]
```

8.9 PRACTICE EXERCISE: QUESTIONS FOR PRACTICE

1. Name some data types in prolog programming language.
2. How prolog language can be stated as procedural language?
3. Discuss the different arithmetic operators used in prolog.
4. Elaborate the concept of unification in prolog with suitable example.
5. Define list in prolog and write its various operations.

REFERENCE

1. Ivan Bratko. Prolog: Programming For Artificial Intelligence, 3/E. India: Pearson Education, 2001.
2. Clocksin, William F., Mellish, Christopher S. Programming in Prolog: Using the ISO Standard. Germany: Springer London, Limited, 2012.
3. Bramer, Max. Logic Programming with Prolog. United Kingdom: Springer London, 2005.

ਜਗਤ ਗੁਰੂ ਨਾਨਕ ਦੇ ਨੂਰ ਚ ਰੌਸ਼ਨ ਹੈ ਇਹ ਵਿਸ਼ਵ ਵਿਦਿਆਲਾ
ਜਗਤ ਗੁਰੂ ਨਾਨਕ ਦੇਵ
ਸ਼ਬਦ ਗੁਰੂ ਨਾਨਕ ਦੇਵ
ਕਿਰਤ ਕਰਮ ਦੀ
ਸ਼ਬਦ ਸੁਰਤ ਦੀ

ਸੰਗਤ ਪੰਗਤ
ਵੰਡ ਛਕਣ ਦੀ

ਖੋਜ, ਵਿਵੇਕ ਅਤੇ ਸਿਰਜਣ ਦੀ
ਕਰਤਾ ਪੁਰਖ ਰਹੱਸ ਦਰਸ਼ਨ ਦੀ

ਸਿੱਖਿਆ ਦੇਵਣ ਵਾਲਾ
ਰੌਸ਼ਨ ਹੈ ਇਹ ਵਿਸ਼ਵ ਵਿਦਿਆਲਾ

ਗਗਨ ਮੰਡਲ ਵਿਚ ਜਗਦੇ ਤਾਰੇ
ਦੀਪਕ ਸੋਹਣ ਦੁਆਰੇ ਦੁਆਰੇ

ਕਾਇਆ ਕਾਗਦ ਅੱਖਰ ਜਗਦੇ
ਪੁਸ਼ਪ ਸੁਹਾਵਣ ਧਰਤੀ ਹਿਰਦੇ

ਇਹ ਤੇਰੀ ਲੀਲਾ ਵਿਸਮਾਦੀ
ਨਿਤ ਨਵੇਲੀ ਆਦਿ ਜੁਗਾਦੀ

ਇਸ ਲੀਲਾ ਦੇ ਕਰਮ ਖੰਡ ਵਿਚ
ਤੇਰਾ ਸ਼ਬਦ ਸਵਾਰਨਹਾਰਾ
ਤੇਰਾ ਨਾਦ ਉਜਾਲਾ

ਰੌਸ਼ਨ ਹੈ ਇਹ ਵਿਸ਼ਵ ਵਿਦਿਆਲਾ

ਜਗਤ ਗੁਰੂ ਨਾਨਕ ਦੇਵ
ਸ਼ਬਦ ਗੁਰੂ ਨਾਨਕ ਦੇਵ

ਜਗਤ ਗੁਰੂ ਨਾਨਕ ਦੇਵ
ਪੰਜਾਬ ਸਟੇਟ ਓਪਨ ਯੂਨੀਵਰਸਿਟੀ
ਪਟਿਆਲਾ

JAGAT GURU NANAK DEV
PUNJAB STATE OPEN UNIVERSITY, PATIALA
(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)