

**GOVERNMENT OF TAMILNADU
DIRECTORATE OF TECHNICAL EDUCATION
CHENNAI – 600 025**

STATE PROJECT COORDINATION UNIT

Diploma in Computer Engineering

**Course Code: 1052
M – Scheme**

**e-TEXTBOOK
on
Open Source Software
for
VI Semester DCOMP**

Convener for COMPUTER ENGINEERING Discipline:

Mrs.A.Ghousia Jabeen,
Principal,
Thanthai Periyar E.V.Ramasamy Govt. Polytechnic College ,
Vellore – 632 002

Team Members for Open Source Software:

Mrs. K. Valarmathi,
HOD/ Computer Engg.
V.S.V.N. Polytechnic College,
Virudhunagar – 626001

Mrs.V.Bhuvaneswari,
Lecturer / Computer Engg.
Govt Polytechnic College
Dharmapuri – 635 205

Mrs.S.A.Amudha,
Lecturer/ Computer Engg.
V.S.V.N. Polytechnic College,
Virudhunagar – 626001

Validated By

Mr. V.G.Ravindhren
Lecturer(SG)/HOD incharge /Computer Engg.
Seshasayee Institute of Technology,
Trichy- 620 010

STATE BOARD OF TECHNICAL EDUCATION & TRAINING, TAMILNADU.

DIPLOMA IN COMPUTER ENGINEERING

M- SCHEME

(to be implemented to the student Admitted from the Year 2015-2016 on wards)

Course Name : Diploma in Computer Engineering.

Subject Code : 35282

Semester : VI

Subject title : **OPEN SOURCE SOFTWARE**

TEACHING & SCHEME OF EXAMINATION:

No. of weeks per Semester: 15 Weeks

Subject	Instructions		Examination			Duration
	Hours / Week	Hours / Semester	Internal Assessment	Board Examination	Total	
OPEN SOURCE SOFTWARE	5	75	25	75	100	3 Hrs

TOPICS AND ALLOCATION OF HOURS			
UNIT NO.	TOPIC	NO .OF HOURS	MARKS
I	OVERVIEW OF OPEN SOURCE SOFTWARE AND OPERATING SYSTEM	10	20
II	OPEN SOURCE PROGRAMMING LANGUAGE – PHP	13	20
III	OPEN SOURCE DATABASE	12	20
IV	PYTHON	15	20
V	OPEN SOURCE SOFTWARE TOOLS AND TECHNOLOGIES	15	20
TEST & REVISION		10	
TOTAL		75	100

RATIONALE

The main aim of this subject is to enable the students to know the basic concepts of open source software and tools. The students will learn about the principles of open source software, web servers, databases, operating systems, programming languages and application development.

OBJECTIVES

On completion of the following units of syllabus contents, the students must be able to :

- Understand the need, advantages and disadvantages of Open Source software.
- Understand the general concepts and modes of Linux Operating System.
- Understand the advanced concepts like Scheduling, Time Accounting, Personalities and coning.
- Understand Linux Networking.
- Know the basic concepts of Open Source Database.
- Know how to connect MYSQL database and closing connection.
- Write Simple MYSQL Programs.
- Creating database and tables in MYSQL.
- Manipulate database tables in MYSQL.
- Understand the concepts of Record Selection technologies
- Install and Configure of PHP on Windows.
- Understand the basic concepts of PHP.
- Understand the String and Array concepts in PHP.
- List the advanced features of PHP.
- Discuss the Memory Management, Parameter Handling and Variables in PHP.
- Understand how to access a database using PHP
- Discuss about the advanced Database techniques.
- Discuss about the Apache Web Server and Configuring the server.
- Explain the History and Architecture of Eclipse IDE Platform.
- Understand the basics of Python
- Knowing the building blocks of python language
- Knowing the development process of a Python program,
- Understanding file handling using python

DETAILED SYLLABUS

UNIT I	1.1 Introduction : Need of Open Sources – Advantages of Open Sources – Applications – FOSS – FOSS usage – Free Software Movement, Commercial aspects of Open Source movement - Certification courses issues - global and Indian. Application of Open Sources 1.2 Open source software operating systems – LINUX – features of linux – Linux architecture - Linux advanced concepts 1.3 Open SPARC Project – Open source compilers – Model driven architecture – Eclipse IDE Platform
UNIT II	2.1 Introduction: What is PHP? - Basic Syntax of PHP - programming in web environment - Common PHP Script Elements - Using Variables - Constants – Data

	types - Operators ; Statements - Working With Arrays –Using Functions – OOP - String Manipulation and Regular Expression 2.2 File and Directory Handling - Including Files - File Access 2.3Working With Forms -Processing Forms -Form Validation – Introduction to advanced PHP concept Simple programs Using PHP
UNIT III	3.1 MySQL: Introduction - Setting up an account - Starting, Terminating and writing your own MySQL Programs – Record Selection Technology - Working with Strings – Date and Time - Sorting Query Results module – Generating Summary – Working with Metadata - Using Sequences – MySQL and Web 3.2 PHP and SQL database: PHP and LDAP ; PHP Connectivity ; Sending and receiving emails 3.3 PHP Database Connectivity: Retrieving data from MySQL - Manipulating data in MySQL using PHP
UNIT IV	4.1 Basic features of Python: Overview – Installing – Running in windows/Linux 4.2 Variables and Strings: Data types - Operators – Decision Control – Conditional Statements - Loops – Example Programs 4.3 Sequences: Lists: Introduction –Fixed size lists and arrays – Lists and Loops – Assignment and references –Identity and equality – Sorted lists – Tuples: Tuples and string formatting – String functions - Sets: Unordered Collections – Simple programs Dictionaries, Sets)Using modules – File Handling -Exception – Handling exception 4.4 Dictionaries : Introduction – Combining two dictionaries with UPDATE – Making copies – Persistent variables – Internal Dictionaries 4.5 Functions and Files : Functions - File Handling – Exception – Handling Exceptions
UNIT V	5.1 WEB SERVER : Apache Web server – Working with web server – Configuring and using apache web server 5.2. Open Source Software tools and Processors : Introduction – Eclipse IDE Platform – Compilers – Model driven architecture tools 5.3CASE STUDY: Government Policy toward Open Source (E- Governance) – Wikipedia as an open Source Project

REFERENCE BOOKS:

Sl.No.	Book Title	Author	Publisher
1	The Complete Reference Linux	Richard Peterson	TataMcGraw Hill, New Delhi Third Edition
2	Web Programming	Chris Bates	Wiley India, New Delhi Third Edition, Reprint 2011
3	MySQL Bible	Steve Suchring	John Wiley sons 2002
4	Programming PHP	Ramsus Lerdof And Levin Tatroe	O'Reilly Publications2002

WEBSITES

<http://developer.android.com>

Learning to Program - A free Python web-book by Alan Gauld

<http://www.freenetpages.co.uk/hp/alan.gauld/>

http://en.wikibooks.org/wiki/Python_Programming

Contents

Unit No.	Unit Title	Page No.
I	Introduction to Open Source Software	7 to 47
II	PHP	48 to 90
III	MySQL	91 to 115
IV	Python	116 to 169
V	Open Source Tools – Web server and Hardware	170 to 182

UNIT – I

Introduction to Open source software

Objectives

- Understand the need for, advantages and applications of open source
- Learn the Linux operating system features and commands
- Explain the Open SPARC project, MDA tools, Eclipse IDE

1.1.1 Definition - Open Source

Open source usually refers to software that is released with source code under a license that ensures that derivative works will also be available as source code, protects certain rights of the original authors, and prohibits restrictions on how the software can be used or who can use it.

Basic principles of the open source

- Freedom to re-distribute software.
- Availability of source code.
- Freedom to copy and modify.
- License travels with the software.
- No discrimination based on technology, field or hardware and so on.

1.1.2 Need for open sources

- Reduce dependency on closed source vendors.
- Increase in software maintenance costs and increased costs of employee health care.
- More access to development and testing tools, project and portfolio management tools, network monitoring, security, content management, etc.
- Get familiar with the tools.
- Great support and a 24/7 online community that responds quickly.
- Access to source code and the ability to customize if you desire.
- Great negotiating power when dealing with closed source vendors.
- Feature set is not bloated and is driven by collaboration amongst the community.
- More secure than most closed source vendors.
- Bug fixes are implemented faster than closed source vendors.

1.1.3 Advantages of open sources

- **Lower software costs** - Open source solutions generally require no licensing fees. The logical extension is no maintenance fees.
- **Simplified license management** - Obtain the software once and install it as many times and in as many locations as needed.
- **Lower hardware costs** - Linux and open source solutions are elegantly compact and portable, and as a result require less hardware power to accomplish the same tasks as on conventional servers (Windows, Solaris) or workstations.
- **Scaling/consolidationpotential** - Linux and open source applications and services can often scale considerably.
- **Ample support** - Open source support is freely available and accessible through the online community via the Internet
- **Unified management**—Specific open source technologies such as CIM (Common Information Model) and WBEM (Web Based Enterprise Management) provide the capability to integrate or consolidate server, service, application, and workstation management for powerful administration.
- **Quality software**—Evidence and research indicate that open source software is good stuff.

1.1.4 Applications of open sources

- Blogging, Discussions, Forums
WordPress
- CD/DVD Creation
DVD Flick
InfraRecorder
- Desktop
 - GNOME
 - KDE
- Ebook Readers
 - BookReader
 - bookworm
 - Calibre
 - EPUBReader
 - FBReader
 - Plucker

- Ecommerce
 - Open For Business
 - Zen Cart
- Education
 - Moodle
 - Sakai
 - Sugar
- Email & Calendar
 - Evolution
 - Thunderbird-Lightening
- Instant Messaging
 - Pidgin
 - PHIRC (IRC)
 - Spark
- Media player
 - OpenLP
- Video Conferencing
 - Ekiga
- Video Editing
 - Avidemux
 - CamStudio
 - Cinelerra
 - Liconcomp
 - PiTiVi
 - Kdenlive
- Web Browser
 - Conkeror
 - Firefox
 - Google Chrome
- Web Conferencing
 - BigBlueButton
 - DimDim
 - OpenMeetings
 - VMukti
 - WebHuddle
- Web Development
 - Amaya

- Bluefish
- BlueGriffon
- Drupal
- Joomla
- KompoZer
- Quanta
- Wordpress

1.1.5 Free and open-source software (FOSS)

FOSS is computer software that can be classified as both free software and open-source software. That is, anyone is freely licensed to use, copy, study, and change the software in any way, and the source code is openly shared so that people are encouraged to voluntarily improve the design of the software. This is in contrast to proprietary software, where the software is under restrictive copyright and the source code is usually hidden from the users.

1.1.6 Benefits of using FOSS

- Decreased software costs
- Increased security and stability (especially in regard to malware)
- Protected privacy
- Users can have more control over their own hardware.

1.1.7 Free software movement (FSM)

The Free Software Movement (FSM) or Free / Open Source Software Movement (FOSSM) or Free / Libre Open Source Software (FLOSS) is a social movement with the goal of obtaining and guaranteeing certain freedoms for software users, namely the freedom to run the software, to study and change the software, and to redistribute copies with or without changes.

1.1.8 Commercial aspects of open source

Open-source software is widely used for public and non-commercial applications. In addition, many independent software vendors (ISVs), value-added resellers (VARs), and hardware vendors (OEMs or ODMs) use open-source frameworks, modules, and libraries inside their proprietary, for-profit products and services. From the customer's point of view, the ability to use open-source technology under standard commercial terms and support is valuable. Customers are willing to pay for the legal protection and high-touch support/training/consulting that are typical of commercial software built on top of the innovation and independence that comes with open source.

Commercial goals

- A dual-license model, where a code base is published under a traditional open-source license and a commercial license simultaneously. Vendors typically charge a perpetual license fee for additional closed-source features, supplementary documentation, testing, and quality, as well as intellectual property indemnification to protect the purchaser from legal liability.
- Functional encapsulation, where an open-source framework or library is installed on a user's computer separately from the commercial product, and the commercial product uses the open source functionality in an "arm's length" way. Vendors typically charge a perpetual license fee for the functionality that they provide under closed source.
- A software as a service model, under the argument that the vendor is charging for the services, not the software itself. Vendors typically charge a monthly subscription fee for use of their hosted applications.
- Not charging for the software, but only for the support, training, and consulting services that assist users of the open-source software. Vendors typically charge an annual fee.
- Charging for the software as part of an information appliance or other hardware device. In this model, the software (e.g., development libraries, administrative tools, or example applications) is delivered as part of a proprietary chip, subsystem, or hardware solution with the binaries pre-installed (sometimes burned into firmware) while the source tree is posted on Sourceforge or other public open-source repository.
- (Freemium) model, making a basic version of the software available for free and charging for premium features, or applications

1.1.9 Certification courses – issues

Open source software (OSS) systems are being used for increasingly critical functions in modern societies, e.g., in health care, finance, government, defense, and other safety and security sensitive sectors. There is an increasing interest in software certification as a means to assure quality and dependability of such systems. However, the development processes and organizational structures of OSS projects can be substantially different from traditional closed-source projects.

1. **Certifications are Vendor-centric**
Every vendor has its own set of certification criteria; none of them match, and there is no uniformity.
2. **Certification's Life Cycle Is Short!**
A vendor can revise, revamp, or completely redo a certification as often as it wants.
3. **Certifications Are Not Real-World Oriented**
Certifications are vendor-oriented and they do not prepare user for the real world.
4. **Certifications Have Been Devalued**
Testing has been inconsistent and all over the map devaluing certification.
5. **No Oversight Body**
Because certifications are vendor-centric, no one is overseeing the whole process.
6. **Degree vs. Certification vs. Experience**
There is still tension in the market over the value/need for a degree versus the need for a certification versus the need for experience.
7. **HR People Are Not In Touch with the Real World**
HR people, including headhunters/recruiters, give no guidance and do nothing to help the situation.
8. **Budget Cuts**
Cuts have killed training dollars—and consequently the certification market—because it costs money to get certified.
9. **Surplus number of Certified People**
It is a major reason for the declining interest in certification.
10. **No One Knows Which Certificates Matter**
No one really knows which certificates are needed to get a job.

1.2 Linux Operating System (Open Source operating system)

Linux is a free open-source operating system based on Unix. Linux was originally created by Linus Torvalds with the assistance of developers from around the globe. Linux is free to download, edit and distribute. Linux is a very powerful operating system and it is gradually becoming popular throughout the world.

1.2.1 Features of Linux

Low cost: There is no need to spend time and huge amount money to obtain licenses since Linux and much of it's software come with the GNU (Generally not Unix) General Public License.

Stability: Linux has high stability compared with other operating systems.

Performance: Linux provides high performance on various networks. It has the ability to handle large numbers of users simultaneously.

Networking: Linux provides a strong support for network functionality; client and server systems can be easily set up on any computer running Linux. It can perform tasks like network backup more faster than other operating systems.

Flexibility: Linux is very flexible. Linux can be used for high performance server applications, desktop applications, and embedded systems.

Compatibility: It runs all common Unix software packages and can process all common file formats.

Wider Choice: There is a large number of Linux distributions which gives you a wider choice. Example: Fedora, Ubuntu, CentOS, Debian, Arch Linux, Linux Mint , RedHat etc.

Fast and easy installation: Linux distributions come with user-friendly installation.

Better use of hard disk: Linux uses its resources well enough even when the hard disk is almost full.

Multitasking: Linux is a multitasking operating system. It can handle many things at the same time.

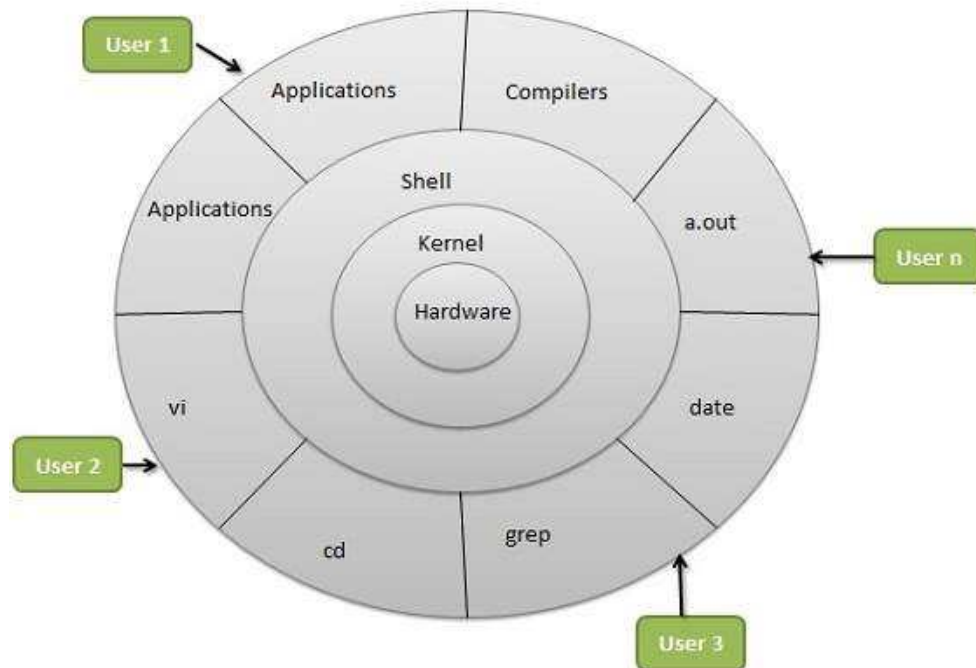
Security: Linux is one of the most secure operating systems. File ownership and permissions make linux more secure.

Open source: Linux is an Open source operating system. The source code for linux can be easily obtained and edited to develop personal operating system.

1.2.2 INTERNAL STRUCTURE OF LINUX

Linux has Two Major components

1. Shell
2. Kernel



1.2.2.1 Kernel

The main component of the Linux O.S is the kernel which makes a direct interface with the hardware components and it performs different functions. It is nothing but the OS itself.

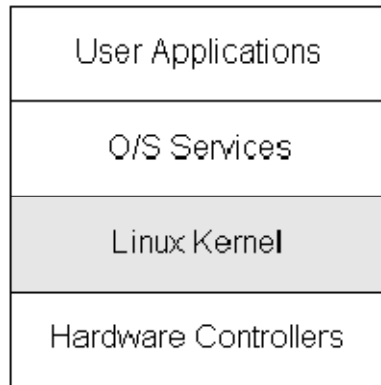
- The kernel make creation and deletion of processes, schedule the memory management and I/O management of the processor.
- It provides a mechanism for synchronization of processes so that processes synchronize their actions.
- It provides mechanism for inter process communication

1.2.2.2 Shell

- Shell is the command interpreter which reads the program we type at terminal, line by line and perform the required operations.
- The shell is the part of Linux O.S that acts as an intermediary between user and the O.S.
- It develops a shell around the system that converts our instructions into commands, which helps the system to understand and act on it.

- Linux system provides every user its own copy of shell program which makes him work freely without any interference of other users.

1.2.3 LINUX SYSTEM ARCHITECTURE



The Linux operating system is composed of four major subsystems

- **User Applications**
Depending on the usage of computer system, the set of applications are different to be used. The examples include a word-processing application, a web browser etc.
- **O.S Services**
O.S services are considered as part of the operating system(a windowing system, command line etc) which also includes the programming interface to the kernel(compiler tool and library)
- **Linux kernel**
The Linux kernel abstracts and mediates access to the hardware resources including the CPU.
- **Hardware Controllers**
The subsystem consists of all the possible physical devices in a Linux installation; for example, the CPU, memory hardware, hard disks, and network hardware.

1.2.3.1 User mode

It is a non-privileged mode in which each process starts out. It is non-privileged in that it is forbidden for processes in this mode to access those portions of memory that have been allocated to the kernel or to other programs. The kernel is a controller of processes, and it alone has access to all resources on the system.

When a user mode process wants to use a service that is provided by the kernel, it must switch temporarily into kernel mode, which has administrative privileges, including root access permissions. When the kernel has satisfied the process's request, it restores the process to user mode.

A user process will enter kernel-mode

- When it decides to execute a system-call
- When it is interrupted (e.g. by the timer)
- When 'exception' occurs (e.g. divide by 0)

1.2.3.2 Kernel mode

System calls act as entry points to the OS kernel. There are certain tasks that can only be done if a process is running in kernel mode. Examples of these tasks can be interacting with hardware etc. So if a process wants to do such kind of task then it would require itself to be running in kernel mode which is made possible by system calls.

In Kernel Mode, user programs can be executed as user processes that have the privilege level of kernel mode. The benefit of executing user programs in kernel mode is that the user programs can access a kernel address space directly. For example, user programs can invoke system calls very fast because it is unnecessary to switch between a kernel mode and a user mode by using costly software interruptions or context switches.

1.2.4 Advanced Concepts

1.2.4.1 Scheduling

Scheduling is a method by which processes are given access to system resources.

1.2.4.1.1 Goals for Scheduling

- **Utilization/Efficiency:** Keep the CPU busy 100% of the time with useful work
- **Throughput:** Maximize the number of jobs processed per hour.
- **Turnaround time:** Time from submission to the time of completion. Should be Minimum
-
-
-
- **Waiting time:** Sum of times spent in ready queue – Should be Minimum
- **Response Time:** Time from submission till the first response is produced. Should be Minimum.
- **Fairness:** Make sure each process gets a fair share of the CPU

1.2.4.1.2 Types of scheduling

- Tasks are usually assigned with priorities. At times it is necessary to run a certain task that has a higher priority before another task although it is running. Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution. This is called **preemptive scheduling**.
Eg: Round robin
- In **non-preemptive scheduling**, a running task is executed till completion. It cannot be interrupted.
Eg First In First Out

1.2.4.1.3 Linux process scheduling

It can be classified into three categories.

1. Real Time FIFO

This contains processes with highest priority. These processes cannot be preempted. The only way to preempt a process in this category is through a new real time FIFO process.

2. Real Time Round Robin

These are similar to real time FIFO processes except that the CPU clock can preempt them.

3. Others (Timesharing)

These are ordinary processes which do not have any urgency and are scheduled by using default time sharing algorithms.

Each process has a scheduling priority, a number between 1 and 40. The OS always executes a process with higher priority. Each process also has a value known as quantum which is equal to the number of clock ticks.

A process stops executing if

- Time slice is over
- The process needs I/O
- Another process with high priority arrives or a previously blocked high priority process becomes ready

The act of switching from one process to another is called a "Context Switch".

1.2.1.4.4 Scheduling classes

There are three classes of processes. (all are defined in include/linux/sched.h)

```
#define SCHED_OTHER 0
#define SCHED_FIFO 1
#define SCHED_RR 2
```

1.2.4.2 Time accounting

Process accounting allows you to view every command executed by a user including CPU and memory time. With process accounting sys admin always find out which command executed at what time

The psacct package contains several utilities for monitoring process activities, including ac, lastcomm, accton and sa.

- 1, The **ac** command displays statistics about how long users have been logged on.
- 2, The **lastcomm** command displays information about previous executed commands.
- 3, The **accton** command turns process accounting on or off.
- 4, The **sa** command summarizes information about previously executed commands.

1.2.4.2.1 ac command

ac command prints out a report of connect time in hours based on the logins/logouts. A total is also printed out.

1. Display total connect time

```
$ ac
```

Output:

```
total    95.08
```

2. Display totals for each day

```
$ ac -d
```

Output:

```
Nov 1 total    8.65
```

```
Nov 2 total    5.70
```

```
...
```

```
Nov 13 total   4.55
```

```
Today total    0.52
```

3. Display time totals for each user

```
$ ac -p
```

Output:

vivek		87.49
root		7.63
total		95.11

1.2.4.2.2lastcomm command

lastcomm command prints out information about previously executed commands. You can search command using usernames, tty names, or by command names itself.

Display command executed by the user vivek.

```
$ lastcomm vivek
```

Output:

```
userhelper      S  X vivekpts/0    0.00 secs Mon Nov 13 23:58
whichvivekpts/0 0.00 secs Mon Nov 13 23:44
bash            F  vivekpts/0    0.00 secs Mon Nov 13 23:44
lsvivekpts/0    0.00 secs Mon Nov 13 23:43
rmvivekpts/0    0.00 secs Mon Nov 13 23:43
vi              vivek pts/0    0.00 secs Mon Nov 13 23:43
ping            S  vivekpts/0    0.00 secs Mon Nov 13 23:42
catvivekpts/0   0.00 secs Mon Nov 13 23:42
```

For each entry the following information is printed. Consider the first output line

```
userhelper S X vivekpts/0 0.00 secs Mon Nov 13 23:58
```

Where, userhelper is command name of the process

S and X are flags, as recorded by the system accounting routines.

S -- command executed by super-user

F -- command executed after a fork but without a following exec

D -- command terminated with the generation of a core file

X -- command was terminated with the signal SIGTERM

vivek the name of the user who ran the process

pts/0 terminal name

0.0 secs - time the process exited

Search the accounting logs by command name

```
$ lastcommrm
```

```
$ lastcommpasswd
```

Search the accounting logs by terminal name pts/1

```
$ lastcommpts/1
```

1.2.4.2.3. sa command

sa command is used to print summarize information about previously executed commands, number of times the command was called and the system resources used. The information can also be summarized on a per-user basis; sa will save this information into a file named usracct.

```
# sa
```

Output:

579	222.81re	0.16cp	7220k	
4	0.36re	0.12cp	31156k	up2date
8	0.02re	0.02cp	16976k	rpmq
2	8.46re	0.00cp	13510k	bash
8	9.52re	0.00cp	1018k	less

Ex:

```
4 0.36re 0.12cp 31156k up2date
```

Where, **0.36re** "real time" in wall clock minutes

0.12cp sum of system and user time in cpu minutes

31156k cpu-time averaged core usage, in 1k units

up2date command name

Display output per-user

```
# sa -u
```

Display the number of processes and number of CPU minutes on a per-user basis

```
# sa -m
```

1.2.4.2 Personalities

Linux has the ability to execute files compiled for other operating systems. Of course, this is possible only if the files include machine code for the same computer architecture on which the kernel is running. Two kinds of support are offered for these "foreign" programs:

- Emulated execution: necessary to execute programs that include system calls that are not POSIX-compliant.
- Native execution: valid for programs whose system calls are totally POSIX-Compliant.

A process can change its personality by issuing a suitable system call named `personality()`; typical values assumed by the system call's parameter are listed in table.

Few Personalities supported by the Linux kernel

Personality	Operating system
PER_LINUX	Standard execution domain
PER_SVR4	System V Release 4
PER_SCOSVR3	SCO Unix Version 3.2
PER_ISCR4	Interactive Unix
PER_BSD	BSD Unix
PER_SUNOS	SunOS
PER_XENIX	Xenix
PER_IRIX32	SGI IRIX -5 32 bit
PER_RISCOS	RISC OS
PER_SOLARIS	Sun's Solaris

1.2.4.2.1 Personality command

The command `setsid` sets the process execution domain

```
int personality(unsigned long persona);
```

Linux supports different execution domains, or personalities, for each process. Execution domains tell Linux how to map signal numbers into signal actions.

On success, *persona* is made the new execution domain and the previous *persona* is returned. On error, -1 is returned, and *errno* is set appropriately.

1.2.4.3 Clone & Backup your Linux system

clonesys is a Shell script that can be used to get an image of running Linux boxes.

The image can then be burned on CD/DVD. This CD/DVD is bootable and can be used to restore the system as well as to install new similar boxes.

clonesys **IS NOT** a backup tool: it should not be used to backup users data.

Clonesys is easy to use and is fast solution.

1.2.4.3.1 Main Features

- Works with Linux kernel 2.4 and 2.6 (v1.2.0)
- Handles the standard (old) /dev structure, devfs and udev (v1.2.0)
- Supports software RAID built with raidtools or mdadm (v1.2.0)
- Supports LVM 1 & 2 and handles the metadata (v1.2.2)
- Supports ext2/3, JFS (v1.3.0), ReiserFS (v1.3.0) and FAT/DOS filesystems
- Supports Boot Loader on both MBR and Boot Partition (v1.2.2)
- Backups and restores Extended Attributes and ACLs (v1.2.0)
- Recreates the filesystems with the same options if they are not the default ones
- Allow images span on multiple CD/DVD (v1.2.0)

1.2.4.3.2 Using clonesys script

1. Check the configuration

The file "config.ini" contains some global variables that impact the tool behavior.

The tool also builds an archive of the system files. The files that must be included in this archive must be listed in the "backup.ini" file.

Directives can include or exclude files and/or directories.

If some specific modules must be loaded at boot time (a non-standard SCSI driver, for example), we can add them to the optional "moremodules.list" file.

At last, if your system contains LVM and/or RAID partitions which are not described in the /etc/fstab file, you must describe them in the "fs.ini" configuration file.

2. Run the cloning script

Launch the "clonesys.sh" script to create the ISO image. According to the parameters in "config.ini", the ISO image can be burned on a CD-RW.

3. Test the CD

The created CD is a bootable CD. It can be used to recreate a new Linux Box that has the same characteristics as the original one.

1.2.4.3.3 Backing up and restoring data

The common types of backup media are

1. Tape drives
2. Removable disk drives
3. CD-R and DVD-R drives

1. Using cp command

Backing up files or entire directory trees to removable disk devices is done using the mount and cp commands.

First, make a mount point where your files will appear

```
#mkdir /mnt/removable
```

Insert and mount the disk so that you can copy files to it. Use vfat if it was formatted for windows

```
#mount -t vfat /dev/hdc /mnt/removable
```

Copy files using the cp command. Use cp command with R option to copy entire directory trees or simply supply a list of files and directory trees you want to copy as arguments. Use -v option if you want to see the files listed as they are copied.

```
#cp -R -V /var/www/mypage.html /home/you /mnt/removable
```

Unmount the removable disk.

```
#unmount /mnt/removable
```

2. Using tar command

Tar(tape archive) command is used to back up large amounts of data to magnetic tape and it can also be used to save the permissions of files backed up on windows formatted media.

```
#tar -c -v -f dest /path1 [/path2...]
```

The -c option tells tar to create a backup

The -v option causes tar to display the name of each file as it is backed up.

Dest stands for the device that should hold back up data.

```
/dev/st0 – First SCSI magnetic tape drive  
/dev/st1 – Second SCSI magnetic tape drive  
/dev/ht0 – First IDE magnetic tape drive  
/dev/ht1 – Second IDE magnetic tape drive
```

Eg:-

1. Back up /home directory tree to the first SCSI tape device

```
tar -c -v -f /dev/st0 /home
```

2. Back up files in /var/www/html and /var/ftp directory trees to first IDE tape drive

```
tar -c -v -f /dev/ht0 /var/www/html /var/www/cgi-bin
```

3. back up the /home /var/www and /var/ftp directory trees to a backup file called backup-oct52002.tar.

```
tar -c -f /mnt/opticaldisk/backup-oct52002.tar /home /var/www /var/ftp
```

Restoring tar backups

To restore files from a backup with tar command.

```
tar -x -v -f source [pattern...]
```

The -x option tells tar you want to extract from a backup.

The -v option displays the name of each file as it is restored.

Pattern contains a list of quote enclosed file names to be restored. If pattern is not specified all files are restored.

Eg:

1. restore all files from first SCSI magnetic tape drive.

```
tar -x -v -f /dev/st0
```

2. restore only the files from /var/www tree stored in second IDE magnetic tape drive.

```
tar -x -v -f /dev/ht1 "var/www/*"
```

Linux backup utilities

1.fwbackups

It is cross platform, has a user-friendly interface, and can do single backups or recurring scheduled backups. The fwbackups tool allows you to do backups either locally or remotely in tar, tar.gz, tar.bZ, or rsync format.

2.Bacula

Bacula is a powerful Linux backup solution, and it's one of the few Linux open source backup solutions that's truly enterprise ready. Bacula contains a number of components:

- Director — This is the application that supervises all of Bacula.

- Console — This is how you communicate with the Bacula Director.
- File — This is the application that's installed on the machine to be backed up.
- Storage — This application performs the reading and writing to your storage space.
- Catalog — This application is responsible for the databases used.
- Monitor — This application allows the administrator to keep track of the status of the various Bacula tools.

3. Rsync

Rsync is one of the most widely used Linux backup solutions. With rsync, you can do flexible incremental backups, either locally or remotely. Rsync can update whole directory trees and file systems; preserve links, ownerships, permissions, and privileges; use rsh, ssh, or direct sockets for connection; and support anonymous connections

4. Amanda

Amanda allows an administrator to set up a single backup server and back up multiple hosts to it. It's robust, reliable, and flexible. Amanda uses native Linux dump and/or tar to facilitate the backup process.

5. Arkeia

Arkeia is best suited for large business to enterprise-level needs.

6. Box Backup

Box Backup is unique in that not only is it fully automated but it can use encryption to secure your backups. Box Backup uses both a client daemon and server daemon, as well as a restore utility. Box Backup uses SSL certificates to authenticate clients, so connections are secure.

7. Kbackup

Kbackup is a simple backup utility that backs up locally to any media (hard drive or mounted device) that can be written to. Kbackup uses the tar format to restore backups, which is as simple as using ARK as a GUI for unpacking the backup files.

1.2.4.4 Linux Signals

Signals are software interrupts.

A robust program is needed to handle signals. This is because signals are a way to deliver asynchronous events to the application.

A user hitting ctrl+c, a process sending a signal to kill another process etc are all such cases where a process needs to do signal handling.

Important Linux Signals are:

Signal Name	Number	Description
SIGHUP	1	Hangup (POSIX)
SIGINT	2	Terminal interrupt (ANSI)
SIGTRAP	5	Trace trap (POSIX)
SIGBUS	7	BUS error (4.2 BSD)
SIGFPE	8	Floating point exception (ANSI)
SIGKILL	9	Kill(can't be caught or ignored) (POSIX)
SIGTERM	15	Termination (ANSI)
SIGSTKFLT	16	Stack fault
SIGCONT	18	Continue executing, if stopped (POSIX)
SIGSTOP	19	Stop executing(can't be caught or ignored) (POSIX)
SIGXCPU	24	CPU limit exceeded (4.2 BSD)
SIGXFSZ	25	File size limit exceeded (4.2 BSD)
SIGIO	29	I/O now possible (4.2 BSD)
SIGPWR	30	Power failure restart (System V)

Processes can ignore, block, or catch all signals except SIGSTOP and SIGKILL. If a process catches a signal, it means that it includes code that will take appropriate action when the signal is received. If the signal is not caught by the process, the kernel will take default action for the signal.

1.2.4.5 Development with Linux

An integrated development environment (IDE) (sometimes known as an integrated design environment or integrated debugging environment) is a software application that provides comprehensive facilities to programmers for software development.

An IDE normally comprises of a

- Source code editor (essentially a text editor with additional features such as code completion, structural navigation, and syntax highlighting)
- Debugger
- Compiler and/or interpreter
- Build automation tools

A good IDE helps developers produce bug-free, maintainable applications quicker than by using an editor and console based tools.

Integrated Development Environments

Anjuta	Versatile development environment for GNOME
Eclipse	Java based environment combining a number of different Eclipse projects
KDevelop	An easy to use IDE for KDE
NetBeans IDE	Java based IDE from Sun Microsystems
Oracle Studio	(formerly Sun Studio) Software development product
CodeLite	Powerful and lightweight C/C++ IDE

1.2.4.6 Linux Networking

Configure network card

First change directory to /etc/sysconfig/network-scripts

```
# cd /etc/sysconfig/network-scripts/
```

Edit / create files as follows:

- **/etc/sysconfig/network-scripts/ifcfg-eth0** : First Ethernet card configuration file
- **/etc/sysconfig/network-scripts/ifcfg-eth1** : Second Ethernet card configuration file

To edit/create first NIC file, type command

```
# vi ifcfg-eth0
```

Append/modify as follows

```
# Intel Corporation 82573E Gigabit Ethernet Controller (Copper)
DEVICE=eth0
BOOTPROTO=static
DHCPCLASS=
HWADDR=00:30:48:56:A6:2E
IPADDR=10.10.29.66
NETMASK=255.255.255.192
ONBOOT=yes
```

Save and close the file. Define default gateway (router IP) and hostname in /etc/sysconfig/network file

```
# vi /etc/sysconfig/network
```

Append/modify configuration as follows:

```
NETWORKING=yes
HOSTNAME=www1.nixcraft.in
GATEWAY=10.10.29.65
```

Save and close the file. Restart networking:

```
# /etc/init.d/network restart
```

Make sure you have correct DNS server defined in /etc/resolv.conf file:

```
# vi /etc/resolv.conf
```

Setup DNS Server as follows:

```
nameserver 10.0.80.11
nameserver 10.0.80.12
nameserver 202.67.222.222
```

Save and close the file. Now you can ping the gateway/other hosts:

```
$ ping 10.0.80.12
```

Output:

```
PING 10.0.80.12 (10.0.80.12) 56(84) bytes of data.
64 bytes from 10.0.80.12: icmp_seq=1 ttl=251 time=0.972 ms
64 bytes from 10.0.80.12: icmp_seq=2 ttl=251 time=1.11 ms
```

You can also check for Internet connectivity with nslookup or host command:

```
$ nslookup cyberciti.biz
```

Output:

```
Server:      10.0.80.11
Address:     10.0.80.11#53
Non-authoritative answer:
Name:   cyberciti.biz
Address: 75.126.43.232
```

You can also use host command:

```
$ host nixcraft.in
```

Output:

```
nixcraft.in has address 75.126.43.232
nixcraft.in mail is handled by 10 mail.nixcraft.in.
```

Setup a default Gateway

Open /etc/sysconfig/network file:

```
# vi /etc/sysconfig/network
```

Setup GATEWAY={Router-IP}, if router IP is 74.86.49.129, type:

```
GATEWAY=74.86.49.129
```

Save and close the file.

You need to restart networking service:

```
# /etc/init.d/networking restart
```

Set the DNS from the command line

Edit the /etc/resolv.conf file and add the line:

```
nameserver {IP-OF-THE-DNS-1}
```

```
nameserver {IP-OF-THEISP-DNS-SERVER-2}
```

Login as the root, enter:

```
# vi /etc/resolv.conf
```

OR

```
$ sudo vi /etc/resolv.conf
```

Modify or enter nameserver as follows:

```
nameserver 208.67.222.222
```

```
nameserver 208.67.220.220
```

Save and close the file.

To test DNS configuration type any one of the following command:

```
$ host google.com
```

```
$ dig google.com
```

```
$ ping google.com
```

```
$ nslookup your-domain.com
```

Output:

```
google.com has address 72.14.207.99
```

```
google.com has address 64.233.187.99
```

```
google.com mail is handled by 10 smtp2.google.com.
```

```
google.com mail is handled by 10 smtp3.google.com.
```

Network Configuration

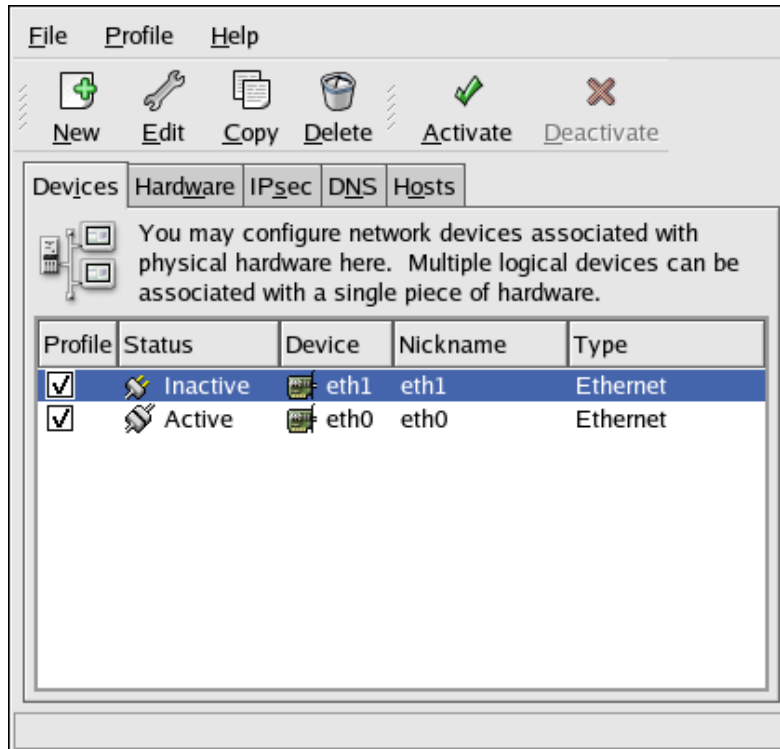
To communicate with each other, computers must have a network connection. This is accomplished by having the operating system recognize an interface card (such as Ethernet, ISDN modem, or token ring) and configuring the interface to connect to the network.

The **Network Administration Tool** can be used to configure the following types of network interfaces:

- Ethernet
- ISDN
- modem
- xDSL
- token ring
- CIPE
- wireless devices

It can also be used to configure IPsec connections, manage DNS settings, and manage the **/etc/hosts** file used to store additional hostnames and IP address combinations.

To use the **Network Administration Tool**, you must have root privileges. To start the application, go to the Applications (the main menu on the panel) > **System Settings** > **Network**, or type the command **system-config-network** at a shell prompt. To use the command line version, execute the command **system-config-network-cmd --help** as root to view all of the options.



1.4.2 Network Administration Tool

Overview

To configure a network connection with the **Network Administration Tool**, perform the following steps:

1. Add a network device associated with the physical hardware device.
2. Add the physical hardware device to the hardware list, if it does not already exist.
3. Configure the hostname and DNS settings.
4. Configure any hosts that cannot be looked up through DNS.

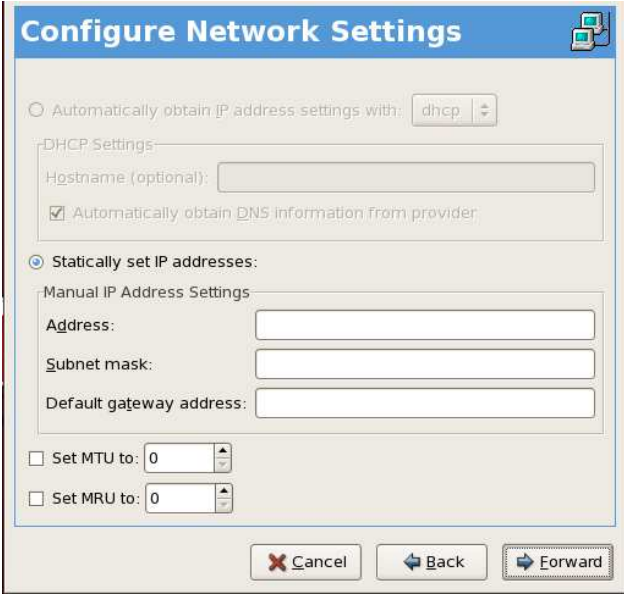
Establishing an Ethernet Connection

To establish an Ethernet connection, you need a network interface card (NIC), a network cable (usually a CAT5 cable), and a network to connect to.

To add an Ethernet connection, follow these steps:

1. Click the **Devices** tab.
2. Click the **New** button on the toolbar.
3. Select **Ethernet connection** from the **Device Type** list, and click **Forward**.
4. If you have already added the network interface card to the hardware list, select it from the **Ethernet card** list. Otherwise, select **Other Ethernet Card** to add the hardware device.

5. If you selected **Other Ethernet Card**, the **Select Ethernet Adapter** window appears. Select the manufacturer and model of the Ethernet card. Select the device name. If this is the system's first Ethernet card, select **eth0** as the device name; if this is the second Ethernet card, select **eth1** (and so on). The **Network Administration Tool** also allows you to configure the resources for the NIC. Click **Forward** to continue.

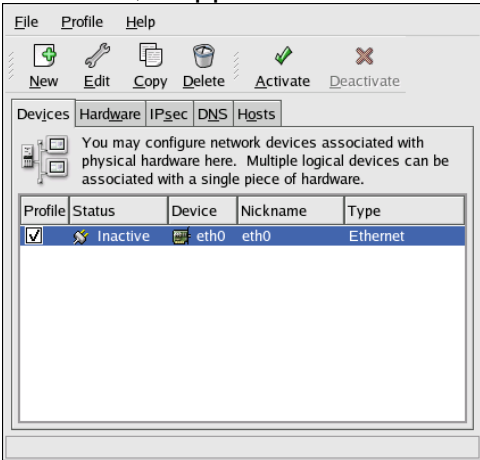


The image shows the 'Configure Network Settings' window. It has a blue title bar with the text 'Configure Network Settings' and a small icon. Below the title bar, there are two radio buttons: 'Automatically obtain IP address settings with:' (selected) and 'dhcp'. Below this is a section for 'DHCP Settings' with a 'Hostname (optional):' text box and a checked checkbox 'Automatically obtain DNS information from provider:'. Below that is a section for 'Statically set IP addresses:' with a 'Manual IP Address Settings' section containing three text boxes: 'Address:', 'Subnet mask:', and 'Default gateway address:'. Below these are two checkboxes: 'Set MTU to:' with a value of 0 and 'Set MRU to:' with a value of 0. At the bottom are three buttons: 'Cancel', 'Back', and 'Forward'.

6. In the **Configure Network Settings** window shown above choose between DHCP and a static IP address. If the device receives a different IP address each time the network is started, do not specify a hostname. Click **Forward** to continue.
7. Click **Apply** on the **Create Ethernet Device** page.

Ethernet Settings

After configuring the Ethernet device, it appears in the device list.



The image shows the 'Network Administration Tool' window. It has a menu bar with 'File', 'Profile', and 'Help'. Below the menu bar is a toolbar with icons for 'New', 'Edit', 'Copy', 'Delete', 'Activate', and 'Deactivate'. Below the toolbar is a tabbed interface with tabs for 'Devices', 'Hardware', 'IPsec', 'DNS', and 'Hosts'. The 'Devices' tab is selected. Below the tabs is a text box with the text: 'You may configure network devices associated with physical hardware here. Multiple logical devices can be associated with a single piece of hardware.' Below the text box is a table with the following columns: 'Profile', 'Status', 'Device', 'Nickname', and 'Type'. The table has one row with the following values: 'eth0', 'Inactive', 'eth0', 'eth0', and 'Ethernet'.

Profile	Status	Device	Nickname	Type
eth0	Inactive	eth0	eth0	Ethernet

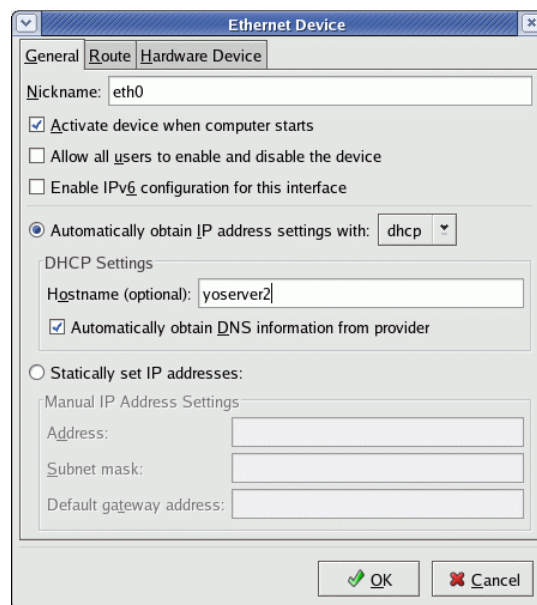
Select **File > Save** to save the changes.

After adding the Ethernet device, you can edit its configuration by selecting the device from the device list and click **Edit**.

When the device is added, it is not activated immediately, as seen by its **Inactive** status. To activate the device, select it from the device list, and click the **Activate** button.

Assigning an IP address

Select your Ethernet card(eth0 or eth1) in network administration window and click on the Edit button. You can now assign IP address, net mask, default gateway and other properties.



Click Ok button to save the changes.

1.4.4 Subnets

Subnet (short for "sub network") is an identifiably separate part of an organization's network. Typically, a subnet may represent all the machines at one geographic location, in one building, or on the same local area network (LAN). Having an organization's network divided into subnets allows it to be connected to the Internet with a single shared network address.

A Subnet mask is a 32-bit number that masks an IP address, and divides the IP address into network address and host address. Subnet Mask is made by setting network bits to all "1"s and setting host bits to all "0"s. Within a given network, two host addresses are reserved for special purpose. The "0" address is assigned a network address and "255" is assigned to a broadcast address, and they cannot be assigned to a host.

For example, consider the IP address 150.215.017.009. Assuming this is part of a Class B network, the first two numbers (150.215) represent the Class B network address, and the second two numbers (017.009) identify a particular host on this network.

IP address in binary format.

The full address is:

10010110.11010111.00010001.00001001

The Class B network part is:

10010110.11010111

and the host address is

00010001.00001001

If this network is divided into 14 subnets, however, then the first 4 bits of the host address (0001) are reserved for identifying the subnet.

The subnet mask is the network address plus the bits reserved for identifying the subnetwork -- by convention, the bits for the network address are all set to 1, though it would also work if the bits were set exactly as in the network address. In this case, therefore, the subnet mask would be 11111111.11111111.11110000.00000000. It's called a mask because it can be used to identify the subnet to which an IP address belongs by performing a **bitwise AND operation** on the mask and the IP address. The result is the sub network address:

Subnet Mask	255.255.240.000	11111111.11111111.11110000.00000000
IP Address	150.215.017.009	10010110.11010111.00010001.00001001
Subnet Address	150.215.016.000	10010110.11010111.00010000.00000000

The subnet address, therefore, is 150.215.016.000.

Routes

1, Static routes

IP (Internet Protocol) uses a routing table to determine where packets should be sent. First the packet is examined to see if its destination is for the local or remote network. If it is to be sent to a remote network, the routing table is consulted to

determine the path. If there is no information in the routing table then the packet is sent to the default gateway. Static routes are set with the route command and with the configuration file

/etc/sysconfig/network-scripts/route-eth0

2, Dynamic routes

RIP (Routing Information Protocol) is used to define dynamic routes. If multiple routes are possible, RIP will choose the shortest route. Routers use RIP to broadcast the routing table over UDP port 520. The routers would then add new or improved routes to their routing tables.

Route command

It is used to show / manipulate the IP routing table (Static route)

Show routes:

Option	Description
-n	display IP addresses. Do not resolve host names for faster results.
-e	Print more extensive information about routes.
-v	Verbose.
--help	Route command information.

Manipulate routes:

Option	Description
add or del or neither	Add or delete route information. If not specified then print route table information.
-host XXX.XXX.XXX.XXX	Add a single computer host identified by the IP address.
-net XXX.XXX.XXX.XXX	Add a network identified by the network address, to the route.
gw XXX.XXX.XXX.XXX	Specify the network gateway.
netmask XXX.XXX.XXX.XXX	Specify the network netmask.
Default	Of all the routes specified, identify one as the default network route. (typically the gateway is specified as the default route)

Examples:

1, To display routing table type the following command at UNIX / Linux shell prompt:

```
$ netstat -r -n
```

or

\$ route -n

Sample output:

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Interface
192.168.2.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
192.168.1.0	0.0.0.0	255.255.255.0	U	0	0	0	ra0
169.254.0.0	0.0.0.0	255.255.0.0	U	1000	0	0	eth0
0.0.0.0	192.168.1.1	0.0.0.0	UG	100	0	0	ra0

The output of the kernel routing table is organized in the following columns:

- Destination : The destination network or destination host.
- Gateway : The gateway address or '*' if none set.
- Genmask : The netmask for the destination net; 255.255.255.255 for a host destination and 0.0.0.0 for the default route.
- Flags : Possible flags include
 - U (route is up)
 - H (target is a host)
 - G (use gateway)
 - R (reinstate route for dynamic routing)
 - D (dynamically installed by daemon or redirect)
 - M (modified from routing daemon or redirect)
 - A (installed by addrconf)
 - C (cache entry)
 - ! (reject route)
- Metric : The distance to the target (usually counted in hops). It is not used by recent kernels, but may be needed by routing daemons.
- Ref : Number of references to this route. (Not used in the Linux kernel.)
- Use : Count of lookups for the route. Depending on the use of -F and -C this will be either route cache misses (-F) or hits (-C).
- Iface : Interface to which packets for this route will be sent.

2, To add static route to a network in the routing table

```
route add -net 192.168.1.0 netmask 255.255.255.0 gw 192.168.1.1 dev eth0
```

In the above command

add – indicates the route is added to the Routing table

-net - indicates the destination is a network

192.168.0.1 – indicates IP address of destination network.

Netmask – indicates the subnet mask of destination network

gw 192.168.1.1 – indicates the gateway of destination network

dev eth0 – indicates the packets are routed via the interface eth0.

3, To delete a route from the routing table

```
route del -net 192.168.1.0 netmask 255.255.255.0 gw 192.168.1.1 dev eth0
```

Tunneling

Tunneling, also known as "port forwarding," is the transmission of data intended for use only within a private, usually corporate network through a public network in such a way that the routing **nodes** in the public network are unaware that the transmission is part of a private network. Tunneling allows the use of **the Internet**, which is a public network, to convey data on behalf of a private network.

Tunnels are managed with ip program, part of Iproute2:

```
$ /sbin/ip tunnel help
```

```
Usage: ip tunnel { add | change | del | show } [ NAME ]  
[ mode { ipip | gre | sit } ] [ remote ADDR ] [ local ADDR ]  
[ [i|o]seq ] [ [i|o]key KEY ] [ [i|o]csum ]  
[ ttl TTL ] [ tos TOS ] [ [no]pmtudisc ] [ dev PHYS_DEV ]
```

```
Where: NAME := STRING  
ADDR := { IP_ADDRESS | any }  
TOS := { NUMBER | inherit }  
TTL := { 1..255 | inherit }  
KEY := { DOTTED_QUAD | NUMBER }
```

Types of Tunnels

1, IPIP tunnels

IPIP kind of tunnels is the simplest one. It has the lowest overhead, but can encapsulate only IPv4 unicast traffic, so you will not be able to setup OSPF, RIP or any other multicast-based protocol. You can setup only one tunnel for unique tunnel endpoints pair. It can work with FreeBSD and cisco IOS. Kernel module is 'ipip'.

2, GRE tunnels

GRE tunnels can encapsulate IPv4/IPv6 unicast/multicast traffic, so it is de-facto tunnel standard for dynamic routed networks. You can setup up to 64K tunnels for a unique tunnel endpoints pair. It can work with FreeBSD and cisco IOS. Kernel module is 'ip_gre'.

3, SIT tunnels

SIT stands for Simple Internet Transition. Its main purpose is to interconnect isolated IPv6 networks, located in global IPv4 Internet. SIT works like IPIP. It can work with FreeBSD and cisco IOS. Kernel module is 'ipv6'.

Linux networking commands

1. netstat - Display connections, routing tables, statistics etc
 - a: Show both listening and non-listening sockets.
 - p: Show PID of process owning socket
 - u: Show UDP
 - t: Show TCP
 - n: Show IP addresses only. Don't resolve host names
 - g: Show multi-cast group membership info
 - c: Continuous mode - update info every second
 - v: Verbose
 - e: Extended information
 - o: show network timer information
2. ping - send ICMP ECHO_REQUEST packets to network hosts. Use Cntl-C to stop ping.
3. traceroute - print the route packets take to network host.
4. mtr - a network diagnostic tool - Like traceroute except it gives more network quality and network diagnostic info. Leave running to get real time statistics. Reports best and worst round trip times in milliseconds.
5. whois - Lookup a domain name in the internal whois database.
6. finger - Display information on a system user.
7. socklist - Display list of open sockets, type, port, process id and the name of the process. Kill with fuser or kill.
8. host - Give a host name and the command will return IP address.
9. nslookup - Give a host name and the command will return IP address.

Enable Forwarding

Forwarding allows the network packets on one network interface (i.e. eth0) to be forwarded to another network interface (i.e. eth1).

A router configuration can support multicast and basic IP routing using the "route" command. Turn on IP forwarding to allow Linux computer to act as a gateway or router.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Default is 0. One can add firewall rules by using ipchains.

Another method is to alter the Linux kernel config file: /etc/sysctl.conf Set the following value:

```
net.ipv4.ip_forward = 1
```

See file /etc/sysconfig/network for storing this configuration.

```
FORWARD_IPV4=true
```

Change the default "false" to "true". All methods will result in a proc file value of "1".

1.3.1 Open SPARC Project

OpenSPARC was an open-source hardware project started in December 2005. The initial contribution to the project was Sun Microsystems' register-transfer level (RTL) Verilog code for a full 64-bit, 32-thread microprocessor, the UltraSPARC T1 processor. On March 21, 2006, Sun released the source code to the T1 IP core under the GNU General Public License. The full OpenSPARC T1 system consists of 8 cores, each one capable to execute 4 threads concurrently, for a total of 32 threads. Each core executes instruction in order and its logic is split among 6 pipeline stages.

On December 11, 2007, Sun also made the UltraSPARC T2 processor's RTL available via the OpenSPARC project. OpenSPARC T2 has 8 cores, 16 pipelines with 64 threads

1.3.2 Compilers

A compiler is a computer program (or set of programs) that transforms source code written in a programming language) into an executable program.

1.3.2.1 GCC

The GNU C compiler by Richard Stallman et al. A very high quality, very portable compiler for C, C++ and Objective C. The compiler is designed to support multiple front-

ends and multiple back-ends by translating first into Register Transfer Language and from there into assembly code for the target architecture.

Features

- GCC is a portable compiler.
- GCC is not only a native compiler--it can also *cross-compile*.
- GCC has multiple language *frontends*.
- GCC has a modular design, allowing support for new languages and architectures to be added.
- GCC is free software, distributed under the GNU General Public License (GNU GPL).

1.3.2.2 Open64

Open64 is an open source, optimizing compiler for the Itanium and x86-64 microprocessor architectures.

The Open64 compiler suite includes optimizing compilers and runtime support for C, C++ and FORTRAN 77/90/95. These compilers produce code that follows the corresponding ABI on IA-32/Linux, x86_64/Linux and IA-64/Linux and be compatible with the latest GCC revision. This means that objects produced by the Open64 compilers can link with objects produced by other ABI compliant compilers.

Open64 is used in a number of research projects, such as the Unified Parallel C (UPC) and speculative multithreading work at various universities.

Features

- C compatible with gcc, with gcc extensions
- C++ compatible g++, with g++ extensions
- FORTRAN 77/90/95 support
- Platform-independent optimizations
 - Code Motion
 - Constant propagation
 - Dead code elimination
 - Expression simplification
 - Common subexpression elimination
 - Strength reduction
 - Partial redundancy elimination
 - Loop optimizations, enabled at '-O3'

- Interprocedural analysis and optimization, enabled with '-ipa'
- Platform-dependent optimizations
 - Software pipelining on IA-64
 - Control and Data speculation on IA-64
 - Integrated instruction scheduling with resource management
- OpenMP support in C/C++/Fortran

1.3.2.3 Free Pascal

Originally known as FPK-Pascal, the Free Pascal compiler is a 32 and 64 bit Turbo Pascal and Delphi compatible Pascal compiler.

This compiler is available for several architectures, x86, Sparc (v8,v9), ARM, x86_64 (AMD64/Opteron) and Powerpc.

It provides a completely portable RunTime Library (RTL) available on many platforms and compatible with Turbo Pascal, but also a platform independent class based Free Component Library (FCL) adding many Delphi extensions and interfacing many popular open source libraries.

Features

- Language syntax has excellent compatibility with TP 7.0 as well as with most versions of Delphi (classes, rtti, exceptions, ansistrings, widestrings, interfaces)
- Function overloading
- Operator overloading
- Global properties
- Rich set of code libraries
- Unlike most programming languages, Pascal does not need Makefiles
- Pascal compilers are Fast with a big F and Free Pascal is no exception
- Each unit has its own identifiers
- Free Pascal comes with an IDE which works on several platforms, in which you can write, compile and debug your programs
- Great integration with assembler
- Object oriented programming
- Free Pascal's smart linker leaves out any variable or code that you do not use. That makes small programs small with a big S, while they are still statically linked

1.3.3 Model driven Architecture tools

Model Driven Architecture (MDA) is a software design approach for the development of software systems. It provides a set of guidelines for the structuring of specifications, which are expressed as models. Model-driven architecture is a kind of domain engineering, and supports model-driven engineering of software systems. It was launched by the Object Management Group (OMG) in 2001.

An MDA tool is a tool used to develop, interpret, compare, align, measure, verify, transform, etc. models or meta models. In any MDA approach we have essentially two kinds of models: *initial models* are created manually by human agents while *derived models* are created automatically by programs. For example an analyst may create a UML initial model from its observation of some loose business situation while a Java model may be automatically derived from this UML model by a Model transformation operation.

Types of MDA tool

- **Creation Tool**
A tool used to elicit initial models and/or edit derived models.
- **Analysis Tool**
A tool used to check models for completeness, inconsistencies, or error and warning conditions. Also used to calculate metrics for the model.
- **Transformation Tool**
A tool used to transform models into other models or into code and documentation.
- **Composition Tool**
A tool used to compose.
- **Test Tool**
A tool used to "test" models as described in Model-based testing.
- **Simulation Tool**
A tool used to simulate the execution of a system represented by a given model.
- **Metadata Management Tool**
A tool intended to handle the general relations between different models, including the metadata on each model and the mutual relations between these models.
- **Reverse Engineering Tool**
A tool intended to transform particular legacy or information artifact portfolios into full-fledged models.

One of the characteristics of MDA tools is that they mainly take models as input and generate models as output.

1.3.4 Eclipse IDE Platform

Eclipse is not just for Java , it is

- A language-neutral, integrated development environment (IDE) for building, deploying, and managing software across the entire software lifecycle
- An IDE framework that is easily extended via the concept of plug-ins
- Supported by an open source community focused on building more functionality for the IDE
- A set of more than 60 open source projects

Workbench

The Eclipse Platform is an IDE for anything, and for nothing in particular.

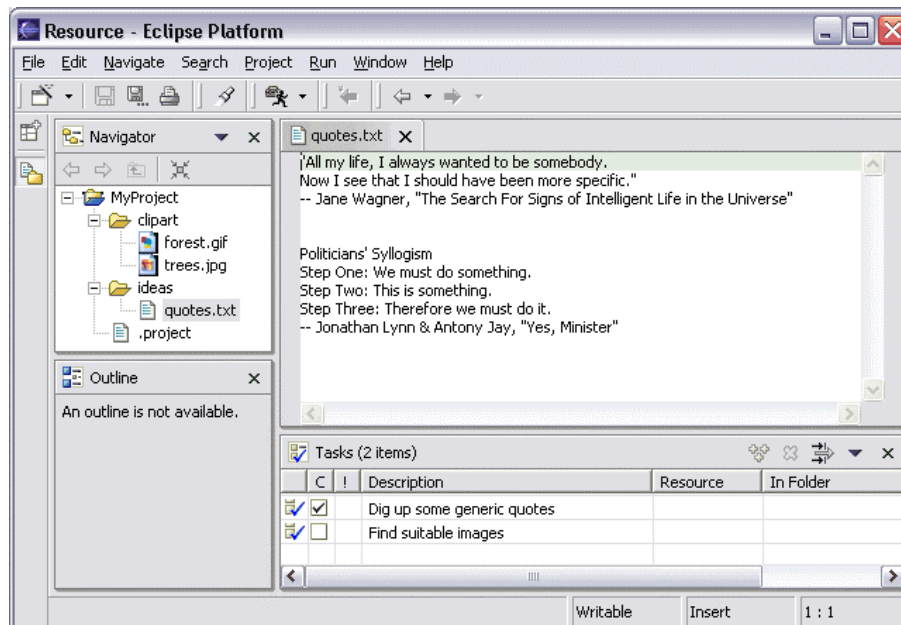
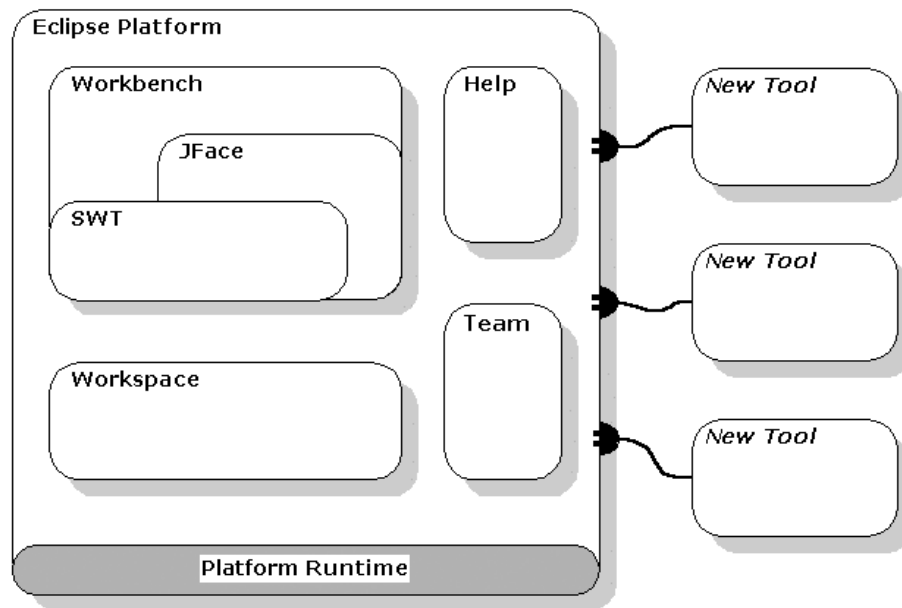


Figure shows a screen capture of the main workbench window of the Eclipse Platform.

- The navigator view (top left) shows the files in the user's workspace
- The text editor (top right) shows the content of a file
- The tasks view (bottom right) shows a list of to-dos
- The outline view (bottom left) shows a content outline of the file being edited (not available for plain text files)

1.3.4.1 Eclipse Platform architecture



Platform Runtime and Plug-in Architecture

A *plug-in* is the smallest unit of Eclipse Platform function that can be developed and delivered separately. Usually a small tool is written as a single plug-in, whereas a complex tool has its functionality split across several plug-ins. Except for a small kernel known as the Platform Runtime, all of the Eclipse Platform's functionality is located in plug-ins, coded in JAVA.

A plug-in is *activated* when its code actually needs to be run. Once activated, a plug-in uses the plug-in registry to discover and access the extensions contributed to its extension points.

Workspaces

The workspace consists of one or more top-level *projects*, where each project maps to a corresponding user-specified directory in the file system.

Workbench and UI Toolkits

The Eclipse Platform UI is built around a workbench that provides the overall structure and presents an extensible UI to the user. The workbench API and implementation are built from two toolkits:

- SWT - a widget set and graphics library integrated with the native window system but with an OS-independent API.

- JFace - a UI toolkit implemented using SWT that simplifies common UI programming tasks.

SWT

The Standard Widget Toolkit (SWT) provides a common OS-independent API for widgets and graphics implemented in a way that allows tight integration with the underlying native window system.

JFace

JFace is a UI toolkit with classes for handling many common UI programming tasks.

JFace is window-system-independent in both its API and implementation, and is designed to work with SWT without hiding it.

JFace includes the usual UI toolkit components of image and font registries, dialog, preference, and wizard frameworks, and progress reporting for long running operations.

Workbench

The *workbench* provides the UI personality of the Eclipse Platform, and supplies the structures in which tools interact with the user. The workbench API is dependent on the SWT API, and to a lesser extent on the JFace API. The workbench implementation is built using both SWT and JFace; Java AWT and Swing are not used.

Team Support

The Eclipse Platform allows a project in the workspace to be placed under version and configuration management with an associated team repository.

Help

The Eclipse Platform Help mechanism allows tools to define and contribute documentation to one or more online books.

1.3.4.2 History of Eclipse

Industry leaders Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft and Webgain formed the initial eclipse.org Board of Stewards in November 2001. By the end of 2003, this initial consortium had grown to over 80 members.

On Feb 2, 2004 the Eclipse Board of Stewards announced Eclipse's reorganization into a not-for-profit corporation. Originally a consortium that formed when IBM released the Eclipse Platform into Open Source, Eclipse became an independent body that will drive the platform's evolution to benefit the providers of software development offerings and end-users. All technology and source code provided to and developed by this fast-growing community is made available royalty-free via the Eclipse Public License.

The founding Strategic Developers and Strategic Consumers were Ericsson, HP, IBM, Intel, MontaVista Software, QNX, SAP and Serena Software.

1.3.4.3 Simultaneous Releases

Each year the Eclipse Foundation and its projects produce a release on a coordinated schedule. This release is often referred to as the simultaneous release, coordinated release, release train, or annual release of Eclipse. Each release typically occurs in June, with follow-up service releases in Sep (SR1) and Feb (SR2).

Release name	Platform version	Main release	SR1	SR2
Callisto	3.2	June 26, 2006	N/A	N/A
Europa	3.3	June 27, 2007	Sep 28, 2007	Feb 29, 2008
Ganymede	3.4	June 25, 2008	Sep 24, 2008	Feb 25, 2009
Galileo	3.5	June 24, 2009	Sep 25, 2009	Feb 26, 2010
Helios	3.6	June 23, 2010	Sep 24, 2010	Feb 25, 2011
Indigo	3.7	June 22, 2011	Sep 23, 2011	Feb 24, 2012
Juno	4.2	June 27, 2012	Sep 28, 2012	Feb 22, 2013
Kepler (planned)	4.3	June 26, 2013	Sep 27, 2013	Feb 28, 2014

Review Questions

Part A

1. Define open source.
2. Define FSM
3. List the advantages of open sources.
4. Give any two features of linux?
5. What is tunneling?
6. What is openSPARC project?

Part B

1. List few applications of open sources.
2. What are the benefits of FOSS?
3. Give the internal structure of Linux.
4. Write short notes on Linux signals.
5. List few open source compilers.
6. How will you backup data?

Part C

1. Explain the architecture of Linux IDE.
2. Write short notes on MDA tools.
3. What are the certification course issues?
4. Explain the architecture of Linux.
5. Write short notes on Linux networking commands.
6. How will you clone and backup linux system?

UNIT – II

PHP (PHP: HYPERTEXT PREPROCESSOR)

LEARNING OBJECTIVES

At the end of this unit, the student will be able to

- Understand the basic concepts of PHP.
- Understand the String and Array concepts in PHP.
- Discuss the variables, constants and operators in PHP.
- Learn about decision making control statements and the way they are used.
- Learn about looping control constructs.
- Understand what an array is.
- Learn about one-dimensional arrays, their declaration, initialization, ways to access individual elements and other possible operations.
- Define two dimensional and multidimensional arrays.
- Learn about string handling functions.
- Write simple programs using decision making statements, arrays and strings.
- Learn about regular expressions and pattern matching.
- Learn about accessing the files and directories.
- Write programs using form and learn about form validation.
- List the advanced features of PHP.
- Understand how to access database using PHP.

2.1 INTRODUCTION

PHP is a server-side, HTML-embedded scripting language that may be used to create dynamic Web pages. It is available for most operating systems and Web servers, and can access most common databases, including MySQL. PHP may be run as a separate program or compiled as a module for use with a Web server.

PHP is a powerful tool for making dynamic and interactive Web pages. PHP is the widely-used, free, and efficient alternative to Microsoft's ASP.

2.1.1 WHAT IS PHP?

- PHP stands for PHP: Hypertext Preprocessor
- PHP is a server-side scripting language, like ASP
- PHP scripts are executed on the server
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- PHP is an open source software
- PHP is free to download and use

PHP files can contain text, HTML, CSS, JavaScript, and PHP code. PHP codes are executed on the server, and the result is returned to the browser as plain HTML. PHP files have extension ".php".

Why PHP?

- PHP runs on different platforms (Windows, Linux, Unix, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP is FREE to download from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

2.1.2 BASIC SYNTAX OF PHP

PHP code is executed on the server, and the plain HTML result is sent to the browser. A PHP scripting block always starts with `<?php` and ends with `?>`. A PHP scripting block can be placed anywhere in the document.

```
<?php  
  
?>
```

A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code. Below example shows a complete PHP page which sends the text "Hello World" to the browser:

```
<html>  
<body>  
<?php  
echo "Hello World";  
?>  
</body>  
</html>
```

Comments in PHP

In PHP, we use `//` to make a single-line comment or `/*` and `*/` to make a large comment block.

```
<html>  
<body>  
<?php  
//This is a comment  
/*  
This is  
a comment  
block  
*/  
?>  
</body>  
</html>
```

2.1.3 PROGRAMMING IN WEB ENVIRONMENT

PHP programs are written using a text editor, such as Notepad, Simple Text, or vi editor, just like HTML pages. However, unlike HTML, PHP files end with a `.php`

extension. This extension signifies to the server that it needs to parse the PHP code before sending the resulting HTML code to the viewer's web browser.

In PHP, on the fly method is adopted to publish the document. Hence, the PHP developer can generate not only web pages, but also other web embedding documents like PDF, PNG, GIF, etc. The PHP web environment is usually set with AMP (Apache, MySQL, and PHP/Perl/Python), which are linked together.

In order to develop and run PHP Web pages three vital components need to be installed on your computer system.

- **Web Server** – PHP will work with virtually all Web Server software, including Microsoft's Internet Information Server (IIS) but most often it is used with freely available Apache Server.
- **Database** – PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database
- **PHP Parser** – In order to process PHP script instructions a parser must be installed to generate HTML output that can be sent to the Web Browser.

2.1.4 COMMON PHP SCRIPT ELEMENTS

2.1.4.1 Using Variables

Variables are "containers" for storing information. Variables are used for storing values, like text strings, numbers or arrays. Variables in PHP are represented by a dollar sign followed by the name of the variable.

Rules for PHP variables:

- A variable starts with the \$ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores
- Variable names are case-sensitive.

Here are some valid variable names:

<pre>\$bill \$head_count \$MaximumForce \$I_HEART_PHP \$_underscore \$_int</pre>
--

Here are some illegal variable names:

\$not valid

\$3wa

Example

```
<?php
$txt="Hello World!";
$x=16;
echo $txt;
?>
```

2.1.4.2 Constants

A constant is an identifier for a simple value; only scalar values—Boolean, integer, double and string—can be constants. Once set, the value of a constant cannot change.

Constants are like variables except that once they are defined they cannot be changed or undefined. A valid constant name starts with a letter or underscore. \$sign should not be given before the constant name.

To create a constant, use the define() function.

Syntax

```
define(name, value, case-insensitive)
```

Parameters:

- *name*: Specifies the name of the constant
- *value*: Specifies the value of the constant
- *case-insensitive*: Specifies whether the constant name should be case-insensitive. Default is false

The example below creates a constant with a **case-sensitive** name:

Example

```
<?php
define("GREETING", "Welcome to learn PHP" );
echo GREETING;
?>
```

Differences between constants and variables

- There is no need to write a dollar sign (\$) before a constant, where as in variable one has to write a dollar sign.
- Constants cannot be defined by simple assignment, they may only be defined using the define() function.

- Constants may be defined and accessed anywhere without regard to variable scoping rules.
- Once the Constants have been set, may not be redefined or undefined.

2.1.4.3 Data Types

- PHP provides eight different data types.
- Four are scalar (single-value) types: integers, floating-point numbers, strings, and Booleans.
- Two are compound (collection) types: arrays and objects.
- The remaining two are special types: resource and NULL.

Integers

Integers are whole numbers, such as 1, 12, and 256. The range of acceptable values varies from -2,147,483,648 to +2,147,483,647. Specifically, the range is equivalent to the range of the long data type of your C compiler. The sequence may begin with a plus (+) or minus (-) sign. If there is no sign, positive is assumed. In the following example \$x is an integer. The PHP var_dump() function returns the data type and value.

Example

```
<?php
$x = 5985;
var_dump($x);
?>
```

Floating-Point Numbers

Floating-point numbers (also known as real numbers) represent numeric values with decimal digit. In the following example \$x is a float. The PHP var_dump() function returns the data type and value:

Example

```
<?php
$x = 10.365;
var_dump($x);
?>
```

Strings

A string is a sequence of characters delimited by either single or double quotes.

```
<?php
$txt="Hello World";
echo $txt;
?>
```

Variables are expanded within double quotes, while within single quotes they are not:

```
<?php
$str= "name";
$var1 = 'My $str will not print!\n';
```

```
print($vae1);  
$var1 = "My $str will print!\n";  
print(var1);  
?>
```

This will produce the following result:

```
My $str will not print!\n  
My name will print
```

Double quotes also support a variety of string escapes

```
\ " Double quotes  
\n Newline  
\r Carriage return  
\t Tab  
\ Backslash  
\$ Dollar sign  
\{ Left brace  
\} Right brace  
\[ Left bracket  
\] Right bracket
```

Booleans

A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;  
$y = false;
```

Booleans are often used in conditional testing.

Arrays

An array is a data structure that stores one or more similar type of values in a single value. For example, if you want to store 100 numbers, then instead of defining 100 variables, it is easy to define an array of 100 lengths.

The array () construct creates an array. Here is the example:

```
$person = array ("Arul", "Balu", "Chitra");
```

Objects

PHP also supports object-oriented programming (OOP). OOP promotes clean modular design, simplifies debugging and maintenance, and assists with code reuse.

Classes are the building blocks of object-oriented design. A class is a definition of a structure that contains properties (variables) and methods (functions). Classes are defined with the class keyword:

```
class Person
{
    public $name = ' ';
    function name ($newname = NULL)
    {
        if (!is_null($newname)) {
            $this->name = $newname;
        }
        return $this->name;
    }
}
```

Once a class is defined, any number of objects can be made from it with the new keyword, and the object's properties and methods can be accessed with the -> construct:

```
$ed = new Person;
$ed->name('Balu');
echo "Hello, {$ed->name}\n";
$tc = new Person;
$tc->name('Charles');
echo "Look out below {$tc->name}\n";
```

The result of the above program is
Hello, Balu
Look out below Charles

2.1.4.4 Operators

PHP language supports following type of operators.

- Arithmetic Operators
- String concatenation Operator
- Comparison Operators
- Assignment Operators
- Logical (or Relational) Operators
- Conditional (or ternary) Operators

Arithmetic operators

The following arithmetic operators are supported by PHP Language

Operator	Description	Example	Result
+	Addition	$x=2$ $x+2$	$x=4$
-	Subtraction	$x=2$ $5-x$	$x=3$
*	Multiplication	$x=4$ $x*5$	$x=20$
/	Division	$15/5$, $5/2$	3, 2.5
%	Modulus (division remainder)	$5\%2$, $10\%8$, $10\%2$	1, 2, 0
++	Increment	$x=5$ $x++$	$x=6$
--	Decrement	$x=5$ $x--$	$x=4$

String Concatenation Operator

The string concatenation operator (.) is used to concatenate two strings. The concatenation operator appends the right-hand operand to the left-hand operand and returns the resulting string. Operands are first converted to strings, if necessary. For example:

```
$n = 5;
$s = 'There were ' . $n . ' Books.';
// $s is 'There were 5 Books'
```

Comparison Operators

Comparison operators compare operands. The result is always either true, if the comparison is truthful, or false otherwise.

Operator	Description	Example
==	is equal to	$5==8$ returns false
!=	is not equal	$5!=8$ returns true
>	is greater than	$5>8$ returns false
<	is less than	$5<8$ returns true
>=	is greater than or equal to	$5>=8$ returns false
<=	is less than or equal to	$5<=8$ returns true

Assignment Operator

The basic assignment operator (=) assigns a value to a variable. The left-hand operand is always a variable. The right-hand operand can be any expression or value.

Operator	Example	Is The Same As
=	$x=y$	$x=y$
+=	$x+=y$	$x=x+y$
-=	$x-=y$	$x=x-y$

<code>*=</code>	<code>x*=y</code>	<code>x=x*y</code>
<code>/=</code>	<code>x/=y</code>	<code>x=x/y</code>
<code>.=</code>	<code>x.=y</code>	<code>x=x.y</code>
<code>%=</code>	<code>x%=y</code>	<code>x=x%y</code>

Logical Operators

Operator	Description	Example
<code>&&</code>	and	<code>x=6 y=3</code> <code>(x < 10 && y > 1)</code> returns true
<code> </code>	or	<code>x=6 y=3</code> <code>(x==5 y==5)</code> returns false
<code>!</code>	not	<code>x=6 y=3</code> <code>!(x==y)</code> returns true

Conditional Operator

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Operator	Description	Example
<code>? :</code> Then	Conditional Expression	If Condition is true ? value X :Otherwise value Y

Example:

```
<html>
<head><title>Arithmetical Operators</title></head>
<body>
<?php
$a = 10;
$b = 20;
/* If condition is true then assign a to result otherwise b */
$result = ($a > $b) ? $a : $b;
echo "TEST1 : Big is $result<br/>";
/* If condition is true then assign a to result otherwise b */
$result = ($a < $b) ? $a : $b;
echo "TEST2 : Small is $result<br/>";
?>
</body>
</html>
```

2.1.5 STATEMENTS

PHP supports a number of traditional programming constructs for controlling the flow of execution of a program.

Conditional statements, such as if-else and switch, allow a program to execute different pieces of code depending on some condition. Loops, such as while and for, support the repeated execution of particular segments of code.

2.1.5.1 Conditional Statements

if Statement

Use the if statement to execute some code only if a specified condition is true

Syntax :

```
if (expression)  
    statement
```

Example:

```
<html>  
<body>  
<?php  
$d=date("D");  
if ($d=="Fri")  
    echo "Have a nice weekend!";  
?>  
</body>  
</html>
```

if...else Statement

if....else statement is used to execute some code if a condition is true and another code if a condition is false.

Syntax

```
if (condition)  
    code to be executed if condition is true;  
else  
    code to be executed if condition is false;
```

Example

```
<?php  
$d=date("D");  
if ($d=="Fri")  
    echo "Have a nice weekend!";  
else  
    echo "Have a nice day!";  
?>
```

Else if ladder Statement

Use the if....elseif...else statement to select one of several blocks of code to be executed.

Syntax;

```
if (condition)
```

```
        code to be executed if condition is true;
elseif (condition)
        code to be executed if condition is true;
else
        code to be executed if condition is false;
```

Exmample

```
<?php
$d=date("D");
if ($d=="Fri")
    echo "Have a nice weekend!";
elseif ($d=="Sun")
    echo "Have a nice Sunday!";
else
    echo "Have a nice day!";
?>
```

Switch Statement

Conditional statements are used to perform different actions based on different conditions

Syntax

```
switch (expression)
{
case label1:
    code to be executed if n=label1;
    break;
case label2:
    code to be executed if n=label2;
    break;
default:
    code to be executed if n is different from both label1 and label2;
}
```

Example

```
<?php
$favcolor = "red";

switch ($favcolor) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is blue!";
        break;
    case "green":
```

```
    echo "Your favorite color is green!";  
    break;  
default:  
    echo "Your favorite color is neither red, blue, nor green!";  
}  
?>
```

2.1.5.2 Looping Statements

In PHP, we have the following looping statements:

- while - loops through a block of code while a specified condition is true
- do...while - loops through a block of code once, and then repeats the loop as long as a specified condition is true
- for - loops through a block of code a specified number of times
- foreach - loops through a block of code for each element in an array

while Loop

The while loop executes a block of code while the condition is true.

Syntax

```
while (condition)  
{  
    code to be executed;  
}
```

Example

The example below defines a loop that starts with i=1. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<?php  
$i=1;  
While($i<=5)  
{  
    echo "The number is " . $i . "  
    $i++;  
}  
?>
```

Output:

```
The number is 1  
The number is 2  
The number is 3  
The number is 4  
The number is 5
```

do...while Statement

The do...while statement will always execute the block of code once, it will then check the condition, and repeat the loop while the condition is true.

Syntax

```
do
{
code to be executed;
} while (condition);
```

Example

The example below defines a loop that starts with $i=1$. It will increment i with 1 and executes the loop as long as i is less than, or equal to 5:

```
<?php
$i=1;
do
{
$i++;
echo "The number is " . $i . "<br/>";
}while($i<=5) ;
?>
```

Output:

```
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
```

for loop

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (initial value; condition; increment)
{
code to be executed;
}
```

Parameters:

1. initial value: Mostly used to set a counter
2. condition: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
3. increment: Mostly used to increment a counter

Example

The example below defines a loop that starts with $i=1$. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<?php
for($i=1;$i<=5; $i++)
{
```

```
echo "The number is " . $i . "<br/>"  
}  
?>
```

Output:

The number is 1
The number is 2
The number is 3
The number is 4
The number is 5

The foreach Loop

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

```
foreach ($array as $value)  
{  
    code to be executed;  
}
```

Example

The following example demonstrates a loop that will print the values of the given array:

```
<?php  
$x=array("one","two","three");  
foreach ($x as $value)  
{  
    echo $value. "<br/>";  
}  
?>
```

Output:

one
two
three

2.1.6 WORKING WITH ARRAYS

An *array* is a collection of data values organized as an ordered collection of key-value pairs. In PHP, there are three kinds of arrays:

- **Numeric array** - An array with a numeric index
- **Associative array** - An array where each ID key is associated with a value
- **Multidimensional array** - An array containing one or more arrays

2.1.6.1 Numeric Arrays

A numeric array stores each array element with a numeric index.

There are two methods to create a numeric array.

1. In the following example the index are automatically assigned (the index starts at 0):
`$cars=array ("Hyundai","Volvo","BMW","Toyota");`

2. In the following example we assign the index manually:

```
$cars [0] ="Hyundai";  
$cars [1] ="Volvo";  
$cars [2] ="BMW";  
$cars [3] ="Toyota";
```

Example

```
<?php  
/* First method to create array. */  
$numbers = array( 1, 2, 3, 4, 5);  
foreach( $numbers as $value )  
{  
    echo "Value is $value <br />";  
}  
/* Second method to create array. */  
$numbers[0] = "one";  
$numbers[1] = "two";  
$numbers[2] = "three";  
$numbers[3] = "four";  
$numbers[4] = "five";  
foreach( $numbers as $value )  
{  
    echo "Value is $value <br />";  
}  
?>
```

This will produce the following result:

```
Value is 1  
Value is 2  
Value is 3  
Value is 4  
Value is 5  
Value is one  
Value is two  
Value is three  
Value is four  
Value is five
```

2.1.6.2 Associative Arrays

The associative arrays are very similar to numeric arrays in terms of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

Example

```
<?php
/* First method to create associate array */
$salaries = array
(
    "Arjun" => 2000,
    "Balu" => 1000,
    "Chitra" => 500
);
echo "Salary of Arjun is ". $salaries['Arjun'] . "<br />";
echo "Salary of Balu is ". $salaries['Balu']. "<br />";
echo "Salary of Chitra is ". $salaries['Chitra']. "<br />";

/* Second method to create array. */
$salaries['Arjun'] = "high";
$salaries['Balu'] = "medium";
$salaries['Chitra'] = "low";
echo "Salary of Arjun is ". $salaries['Arjun'] . "<br />";
echo "Salary of Balu is ". $salaries['Balu']. "<br />";
echo "Salary of Chitra is ". $salaries['Chitra']. "<br />";
?>
```

This will produce the following result:

```
Salary of Arjun is 2000
Salary of Balu is 1000
Salary of Chitra is 500
Salary of Arjun is high
Salary of Balu is medium
Salary of Chitra is low
```

2.1.6.3 Multidimensional Arrays

A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple indexes.

Example

In this example, we create a two dimensional array to store marks of three students in three subjects:

This example is an associative array, you can create numeric array in the same fashion.

```
<?php
$marks = array(
    "Arjun" => array
    (
        "physics" => 35,
        "maths" => 30,
```

```

"chemistry" => 39
),
"Balu" => array
(
"physics" => 30,
"maths" => 32,
"chemistry" => 29
),
"Chitra" => array
(
"physics" => 31,
"maths" => 22,
"chemistry" => 39
)
);
/* Accessing multi-dimensional array values */
echo "Marks for Arjun in physics : " ;
echo $marks['Arjun']['physics'] . "<br />";
echo "Marks for Balu in maths : ";
echo $marks['Balu']['maths'] . "<br />";
echo "Marks for Chitra in chemistry : " ;
echo $marks['Chitra']['chemistry'] . "<br />";
?>

```

This will produce the following result:

Marks for Arjun in physics : 35

Marks for Balu in maths : 32

Marks for Chitra in chemistry : 39

2.1.7 USING FUNCTIONS

A *function* is a named block of code that performs a specific task, possibly acting upon a set of values given to it, or *parameters*, and possibly returning a single value.

2.1.7.1 Defining a PHP Function

To define a function, use the following syntax

```

function functionName()
{
code to be executed;
}

```

Example

```

<?php
/* Defining a PHP Function */
function writeMessage()
{
echo "Good Morning, Have a nice day!";
}

```



```
/* Calling a PHP Function  
writeMessage();  
?>
```

Output:

Good Morning, Have a nice day!

2.1.7.2 PHP Functions with Parameters

To add more usefulness to a function, parameters are included in PHP functions parameters. A parameter is just like a variable. Parameters are specified after the function name, inside the parentheses.

There are two different ways to pass parameters to a function.

1. Pass by Value
2. Pass by Reference

2.1.7.2.1 Passing Parameters by Value

Normally we can pass parameters by value. The argument is any valid expression. The expression is evaluated, and the resulting value is assigned to the appropriate variable in the function.

Example

```
<?php  
function add($num1, $num2)  
{  
    $sum = $num1 + $num2;  
    echo "Sum of the two numbers is : $sum";  
}  
add(10, 20);  
?>
```

2.1.7.2.2 Passing Parameters by Reference

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value. Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

Example

```
<?php  
/* Passing Argument by Reference*/  
function addFive($num)  
{  
    $num += 5;  
}  
function addSix(&$num)
```

```

{
$num += 6;
}
$orignum = 10;
addFive(&$orignum );
echo "Original Value is $orignum<br />";
addSix( $orignum );
echo "Original Value is $orignum<br />";
?>

```

Output:

Original Value is 15

Original Value is 21

2.1.7.3 PHP Functions returning value

A function can return a value using the **return** statement in conjunction with a value or object. Return statement stops the execution of the function and sends the value back to the calling code.

Example

```

<?php
/* Function returning value*/
function addFunction($num1, $num2)
{
    $sum = $num1 + $num2;
    return $sum;
}
$return_value = addFunction(10, 20);
echo "Returned value from the function : $return_value";
?>

```

Output:

Returned value from the function :30

2.1.8 OOP (OBJECT-ORIENTED PROGRAMMING)

Object-oriented programming (OOP) helps to develop cleaner designs, for easy maintenance, and greater code reuse. PHP supports many useful features of OOP. OOP acknowledges the fundamental connection between data and the code that works on that data, and it lets you design and implement programs around that connection.

2.1.8.1 Object Oriented Concepts

Important terms related to Object Oriented Programming

- **Class:** This is a user-defined data type, which includes local functions as well as local data.
- **Object:** An Object is an instance of the class.
- **Member Variable:** These are the variables defined inside a class. This data will be invisible outside of the class and can be accessed via member functions.

- **Member function:** These are the function defined inside a class and are used to access object data.
- **Inheritance:** When a class is defined by inheriting existing class, then it is called inheritance.
- **Parent class:** A class that is inherited from by another class. This is also called a base class or super class.
- **Child Class:** A class that inherits from another class. This is also called a sub class or derived class.
- **Polymorphism:** This is an object oriented concept where the same function can be used for different purposes.
- **Overloading:** A type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly, functions can also be overloaded with different implementation.
- **Data Abstraction:** Any representation of data in which the implementation details are hidden.
- **Encapsulation:** refers to a concept where we data and member functions are packaged together in a single unit. (Ex: class)
- **Constructor:** It is a special type of function which will be called automatically whenever there is an object is instantiated (created) from a class.
- **Destructors:** It is a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

2.1.8.2 Defining PHP Classes

The general form for defining a new class in PHP is as follows:

```
<?php
class ClassName
{
    var $var1;
    var $var2 = "constant string";
    function myfunc ($arg1, $arg2)
    {
        [..]
    }
    [..]
}
?>
```

Example

Here is an example for Books type

```
<?php
class Book
```

```

{
/* Member variables */
var $price;
var $title;

/* Member functions */
function setPrice($par)
{
$this->price = $par;
}
function getPrice()
{
echo $this->price . "<br/>";
}
function setTitle($par)
{
$this->title = $par;
}
function getTitle()
{
echo $this->title . " <br/>";
}
}
?>

```

The variable **\$this** is a special variable and it refers to the current object.

2.1.8.3 Creating Objects in PHP

Once you defined your class, then you can create as many objects as you like of that class type. Following is an example of how to create object using **new** operator.

```

$physics= new Book;
$chemistry = new Book;
$maths = new Book;

```

2.1.8.4 Calling Member Functions

After creating your objects, you will be able to call member functions related to that object. One member function will be able to process member variable of related object only. Following example shows how to set title and prices for the three books by calling member functions.

```

$physics->setTitle( "Physics for High School" );
$chemistry->setTitle( "Advanced Chemistry" );
$maths->setTitle( "Algebra" );
$physics->setPrice( 10 );
$chemistry->setPrice( 15 );
$maths->setPrice( 7 );

```

Now you call another member functions to get the values set by in above example:

```
$physics->getTitle();  
$chemistry->getTitle();  
$maths->getTitle();  
$physics->getPrice();  
$chemistry->getPrice();  
$maths->getPrice();
```

This will produce the following result:

```
Physics for High School  
Advanced Chemistry  
Algebra  
10  
15  
7
```

2.1.8.5 Constructor Functions

Constructor Functions are special type of functions which are called automatically whenever an object is created. PHP provides a special function called **__construct ()** to define a constructor. You can pass any number of arguments into the constructor function.

```
function __construct( $par1, $par2 )  
{  
    $this->price = $par1;  
    $this->title = $par2;  
}
```

Now we don't need to call set function separately to set price and title. We can initialize these two member variables at the time of object creation only. Check the following example

```
$physics = new Book( "Physics for High School", 10 );  
$maths = new Book ( "Advanced Chemistry", 15 );  
$chemistry = new Book ("Algebra", 7 );
```

```
/* Get those set values */  
$physics->getTitle();  
$chemistry->getTitle();  
$maths->getTitle();  
$physics->getPrice();  
$chemistry->getPrice();  
$maths->getPrice();
```

This will produce the following result:

```
Physics for High School  
Advanced Chemistry
```

Algebra

10

15

7

2.1.8.6 Destructor

Like a constructor function you can define a destructor function using function **__destruct()**. You can release all the resources with-in a destructor.

2.1.9 String manipulation

Strings are sequences of characters, like "PHP supports string operations". Some valid examples of strings are as follows:

```
$string_1 = "This is a string in double quotes";  
$string_2 = "This is a somewhat longer, singly quoted string";  
$string_39 = "This string has thirty-nine characters";  
$string_0 = ""; // a string with zero characters
```

String Concatenation Operator

To concatenate two string variables together, use the dot (.) operator:

```
<?php  
$string1="Hello World";  
$string2="1234";  
echo $string1 . " " . $string2;  
?>
```

This will produce the following result:

Hello World 1234

strlen() function

The strlen() function is used to find the length of a string.

```
<?php  
echo strlen("Hello world!");  
?>
```

This will produce the following result:

12

strpos() function

The strpos() function is used to search for a string or character within a string. If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

```
<?php
```

```
echo strpos("Hello world!", "world");  
?>
```

This will produce the following result:

6

2.1.10 REGULAR EXPRESSIONS

A regular expression is a string that represents a *pattern*. The regular expression functions compare the pattern to the string and see if any of the string matches the pattern.

There are three uses for regular expressions

1. Matching
2. Substituting new text for matching text
3. Splitting a string into an array of smaller chunks

Most characters in a regular expression are literal characters, meaning that they match only themselves. For example, if you search for the regular expression `/India/` in the string "I Love India", you get a match because "India" occurs in that string.

Some characters have special meanings in regular expressions. For instance, a caret (^) at the beginning of a regular expression indicates that it must match the beginning of the string.

```
preg_match("/^India/", "I Love India"); // returns false  
preg_match("/^India/", "India is my country"); // returns true
```

A dollar sign (\$) at the end of a regular expression means that it must match the end of the string.

```
preg_match("/India$/", "India is my Country"); // returns false  
preg_match("/cow$/", "I Love India"); // returns true
```

A period (.) in a regular expression matches any single character:

```
preg_match("/c.t/", "cat"); // returns true  
preg_match("/c.t/", "cut"); // returns true  
preg_match("/c.t/", "c t"); // returns true  
preg_match("/c.t/", "bat"); // returns false  
preg_match("/c.t/", "ct"); // returns false
```

Regular expressions are case-sensitive by default, so the regular expression `/India/` doesn't match the string "INDIA".

2.1.10.1 Meta characters

A meta character is simply an alphabetical character preceded by a backslash. Following is the list of meta characters which can be used in PERL Style RegularExpressions.

Character	Description
.	a single character

\s	a whitespace character (space, tab, newline)
\S	non-whitespace character
\d	a digit (0-9)
\D	a non-digit
\w	a word character (a-z, A-Z, 0-9, _)
\W	a non-word character
[aeiou]	matches a single character in the given set
[^aeiou]	matches a single character outside the given set

2.1.10.2 Modifiers

Several modifiers are available that can make your work with regular expressions much easier, like case sensitivity, searching in multiple lines etc.

Modifier	Description
i	Makes the match case insensitive
o	Evaluates the expression only once
x	Allows you to use white space in the expression for clarity
g	Globally finds all matches
cg	Allows a search to continue even after a global match fails

2.1.10.3 PHP's Regular Expression Functions

Function	Description
preg_match()	The preg_match() function searches string for pattern, returning true if pattern exists, and false otherwise
preg_match_all()	The preg_match_all() function matches all occurrences of pattern in string.
preg_replace()	The preg_replace() function searches for string specified by pattern and replaces pattern with replacement if found
preg_split()	The preg_split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.
preg_grep()	The preg_grep() function searches all elements of input array, returning all elements matching the regular expression pattern
preg_quote()	Quote regular expression characters

Function preg_match()

The preg_match() function searches string for pattern, returning true if pattern exists, and false otherwise.

Example

```
<?php
$line = "Vi is the greatest word processor ever created!";
// perform a case-Insensitive search for the word "Vi"
if (preg_match("/\bVi\b/i", $line, $match)) :
print "Match found!";
endif;
?>
```

Output

Match Found

Function preg_match_all()

The preg_match_all() function matches all occurrences of pattern in string

Return Value

Returns the lines that contains the pattern

Example

```
<?php
$userinfo = "Name: <b>John Poul</b><br> Title: <b>PHP Guru</b>";
preg_match_all ("/<b>(.*?)</b>/U", $userinfo, $pat_array);
print $pat_array[0][0]. " <br> ".$pat_array[0][1]. "\n";
?>
```

Output

John Poul
PHP Guru

Function preg_replace()

The preg_replace() function searches for string specified by pattern and replaces pattern with replacement if found.

Return Value

After the replacement has occurred, the modified string will be returned. If no matches are found, the string will remain unchanged

Example

```
<?php
$copy_date = "Copyright 1999";
$copy_date = preg_replace("([0-9]+)", "2000", $copy_date);
```

```
print $copy_date;  
?>
```

Output

Copyright 2000

Function preg_split()

The preg_split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.

Return Value

Returns an array of strings after splitting up a string.

Example

```
<?php  
$ip = "123.456.789.000"; // some IP address  
$iparr = split ("\./", $ip);  
print "$iparr[0] <br />";  
print "$iparr[1] <br />";  
print "$iparr[2] <br />";  
print "$iparr[3] <br />";  
?>
```

Output

123
456
789
000

Function preg_grep()

Returns the array consisting of the elements of the input array that match the given pattern.

Return Value

Returns an array indexed using the keys from the input array.

Example

```
<?php  
$foods = array("pasta", "steak", "fish", "potatoes");  
// find elements beginning with "p", followed by one or more letters.  
$p_foods = preg_grep("/p(\w+)/", $foods);  
print "Found food is " . $p_foods[0];  
print "Found food is " . $p_foods[1];  
?>
```

Output

Found food is pasta
Found food is potatoes

Function preg_quote()

preg_quote() takes str and puts a backslash in front of every character that is part of the regular expression syntax.

Return Value

Returns the quoted string.

Example

```
<?php
$keywords = '$40 for a g3/400';
echo preg_quote($keywords);
?>
```

Output

\\$40 for a g3/400\

2.2 FILE AND DIRECTORY HANDLING

FILE HANDLING

File handling explain the following functions related to files:

- ❖ Opening a file
- ❖ Reading a file
- ❖ Writing a file
- ❖ Closing a file

Opening a file

Files are opened in PHP using the fopen function . The command takes two parameters, the file to be opened, and the mode in which to open the file.

```
$fp = fopen("myfile.txt", "r");
```

If fopen is unable to open the file, it returns 0. This can be used to exit the function with an appropriate message

Example

```
<?php
$handle = fopen("c:\\folder\\file.txt", "r");
?>
```

Closing a file

After making a changes to the opened file it is important to close it with the **fclose()** function. The **fclose()** function requires a file pointer as its argument.

Example

```
<?php
fclose($handle);
?>
```

Reading a file

Once a file is opened using **fopen()** function it can be read with a function called **fread()**. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.

The file's length can be found using the **filesize()** function which takes the file name as its argument and returns the size of the file expressed in bytes. So here are the steps required to read a file with PHP.

- ❖ Open a file using **fopen()** function.
- ❖ Get the file's length using **filesize()** function.
- ❖ Read the file's content using **fread()** function.
- ❖ Close the file with **fclose()** function.

Example

```
<?php
$filename = "c:\\myfile.txt";
$handle = fopen($filename, "r");//open file in read mode
$contents = fread($handle, filesize($filename));//read file
echo $contents;//printing data of file
fclose($handle);//close file
?>
```

Writing a File

A new file can be written or text can be appended to an existing file using the **fwrite()** function. This function requires two arguments specifying a **file pointer** and the string of data that is to be written.

Example

```
<?php
$filename = "/home/user/guest/newfile.txt";
$file = fopen( $filename, "w" );
if( $file == false )
{
    echo ( "Error in opening new file" );
    exit();
}
fwrite( $file, "This is a simple test\n" );
fclose( $file );
?>
```

Directory Handling

PHP includes set of directory functions to deal with the operations, like, listing directory contents, and create/ open/ delete specified directory and etc. These basic functions are listed below.

- `mkdir()`: To make new directory.
- `opendir()`: To open directory.
- `readdir()`: To read from a directory after opening it.
- `closedir()`: To close directory with resource-id returned while opening.
- `rmdir()`: To remove directory.
- `chdir()`: To change the directory

2.2.1 INCLUDING FILES

You can include the content of a PHP file into another PHP file before the server executes it. There are two PHP functions which can be used to included one PHP file into another PHP file.

The `include()` Function

The `require()` Function

include () Function

`Include()` function takes all the text in a specified file and copies it into the file that uses the `include` function.

Assume we have a file called "cars.php":

```
<?php
$color='red';
$car='BMW';
?>
```

Example

```
<html>
<body>
<h1>Welcome to my home page!</h1>
<?php
include 'cars.php';
echo "I have a $color $car.";
?>
</body>
</html>
```

require () Function

The require function is also used to include a file into the PHP code. However, there is one big difference between include and require; when a file is included with the **include** statement and PHP cannot find it, the script will continue to execute.

Example

```
<html>
<body>
<h1>Welcome to my home page!</h1>
<?php
include 'noFileExists.php';
echo "I have a $color $car.";
?>
</body>
</html>
```

If we do the same example using the **require** statement, the echo statement will not be executed because the script execution dies after the require statement returned a fatal error:

Example

```
<html>
<body>
<h1>Welcome to my home page!</h1>
<?php
require 'noFileExists.php';
echo "I have a $color $car.";
?>
</body>
</html>
```

2.2.2 FILE ACCESS

Various file modes are available to give permission to the users for accessing the file

File Modes

The following table shows the different modes the file may be opened in.

.

Mode	Purpose
r	Opens the file for reading only. Places the file pointer at the beginning of the file.
r+	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
w	Opens the file for writing only. Places the file pointer at the beginning of the file. If the file does not exist, then it attempts to create a file.

w+	Opens the file for reading and writing only. Places the file pointer at the beginning of the file. If the file does not exist, then it attempts to create a file.
a	Opens the file for writing only. Places the file pointer at the end of the file. If the file does not exist, then it attempts to create a file
a+	Opens the file for reading and writing only. Places the file pointer at the end of the file. If the file does not exist, then it attempts to create a file.

2.3 WORKING WITH FORMS

A form is a document that containing blank fields, that the user can fill the data or user can select the data.

Example

The example below contains an HTML form with two input fields and a submit button:

```
<html>
<body>
<form action="welcome.php" method="post">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
</body>
</html>
```

When a user fills out the form above and click on the submit button, the form data is sent to a PHP file, called "welcome.php":

"welcome.php" looks like this:

```
<html>
<body> Welcome
<?php echo $_POST["fname"]; ?>!<br />
You are <?php echo $_POST["age"]; ?> years old.
</body>
</html>
```

2.3.1 PROCESSING FORMS

It is easy to process forms with PHP, as the form parameters are available in the \$_GET and \$_POST arrays.

2.3.1.1 The \$_GET Function

The built-in \$_GET function is used to collect values from a form sent with method="get".

Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send (max. 100 characters).

```
<form action="welcome.php" method="get">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
```

When the user clicks the "Submit" button, the URL sent to the server could look something like this:

<http://www.example.com/welcom.php?fname=JFKennedy&age=25>

The "welcome.php" file can now use the \$_GET function to collect form data

```
Welcome <?php echo $_GET["fname"]; ?>.<br />
You are <?php echo $_GET["age"]; ?> years old!
```

Uses

- However, because the variables are displayed in the URL, it is possible to bookmark the page.
- This method should not be used when sending passwords or other sensitive information

2.3.1.2 The \$_POST Function

The built-in \$_POST function is used to collect values from a form sent with method="post". Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

```
<form action="welcome.php" method="post">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
```

When the user clicks the "Submit" button, the URL will look like this:

<http://www.example.com/welcom.php>

The "welcome.php" file can now use the \$_POST function to collect form data

```
Welcome <?php echo $_POST["fname"]; ?>!<br />
You are <?php echo $_POST["age"]; ?> years old.
```

Uses

- Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.
- However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

2.3.2 FORM VALIDATION

Validation means check the input submitted by the user. There are two types of validation are available in PHP. They are as follows –

- **Client-Side Validation** – Validation is performed on the client machine web browsers.
- **Server Side Validation** – after submitted the form, the form has to be sent to a server and perform validation checks in server machine.

Client side validation is faster and reduces the server load. A more secure choice is to use PHP to do the validation. Below example shows a self-processing page with a form. The page allows the user to input the name and age. If the user neglects to give a value, the page displays the warning message.

Example

```
<?php
$Name = $_POST['name'];
$Age = $_POST['age'];

$status = $_POST['status'];
$tried = ($_POST['tried'] == 'yes');
if($tried)
{
    $validated = (!empty($name) && !empty($Age));
    if(!$validated)
    { ?>
    <p>The name and age are required fields. Please fill them out to continue.</p>
    <?php
    }
    }
    if($tried && $validated)
    {
    echo"<p>The item has been created.</p>";
```

```

}
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
Name: <input type="text" name="name" value="<?= $name; ?>" /><br />
Status: <input type="checkbox" name="status" value="active"
<?php
if ($status == "active") { echo "checked"; } ?> /> Active<br />
Age: <input type="text" name="age" value="<?= $age; ?>" /><br />
Status: <input type="checkbox" name="status" value="active"
<?php
if ($status == "active") { echo "checked"; } ?> /> Active<br />
<input type="submit" value="<?php echo $tried ? "Continue" : "Create"; ?>" />
</form>

```

In this case, the validation is simply a check that a value was supplied. We set \$validated to be true only if \$name and \$age are nonempty.

2.3.3 INTRODUCTION TO ADVANCED PHP CONCEPT

2.3.3.1 PHP - Error& Exception Handling

Error handling is an important part when creating scripts and web applications. It is the process of catching errors raised by the program and then taking appropriate action.

Different error handling methods are

- Simple "die()" statements
- Custom errors and error triggers
- Error reporting

Basic Error Handling: Using the die() function

The first example shows a simple script that opens a text file

```

<?php
$file = fopen("welcome.txt","r");
?>

```

If the file does not exist you might get an error like this:

Warning: fopen(welcome.txt) [function.fopen]: failed to open stream:
No such file or directory in C:\webfolder\test.php on line 2

To avoid that the user gets an error message like the one above, we test if the file exist before we try to access it:

```

<?php
if(!file_exists("welcome.txt"))
{
    die("File not found");
}
else
{
    $file = fopen("welcome.txt","r");
}

```

```
    print "File opened successfully";  
}  
?>
```

Now if the file does not exist you get an error like this:
File not found

Exceptions Handling

Exceptions are an abnormal condition. It is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.

Three keywords related to exceptions are

- **try** – A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown".
- **throw** – This is how you trigger an exception. Each "throw" must have at least one "catch".
- **catch** – A "catch" block retrieves an exception and creates an object containing the exception information.

When an exception is thrown, code following the statement will not be executed, and PHP will attempt to find the first matching catch block. If an exception is not caught, a PHP Fatal Error will be issued with an "Uncaught Exception ...

- An exception can be thrown, and caught within PHP. Code may be surrounded in a try block.
- Each try must have at least one corresponding catch block. Multiple catch blocks can be used to catch different classes of exceptions.
- Exceptions can be thrown (or re-thrown) within a catch block.

Example:

```
<?php  
try
```

```

{
    $error = 'Always throw this error';
    throw new Exception($error);
    // Code following an exception is not executed.
    echo 'Never executed';
}
catch (Exception $e)
{
    echo 'Caught exception: ', $e->getMessage(), "\n";
}

```

```

// Continue execution
echo 'Hello World';
?>

```

In the above example `$e->getMessage()` function is used to get error message.

Output:

Caught exception: Always throw this error

Hello World

2.3.3.2 PHP & MySQL

PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database.

MySQL is the most popular database system used with PHP.

What is MySQL?

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation

Database Queries

A query is a question or a request.

We can query a database for specific information and have a record set returned. For example

```
SELECT LastName FROM Employees
```

The query above selects all the data in the "LastName" column from the "Employees" table.

Open a Connection to MySQL and create a database

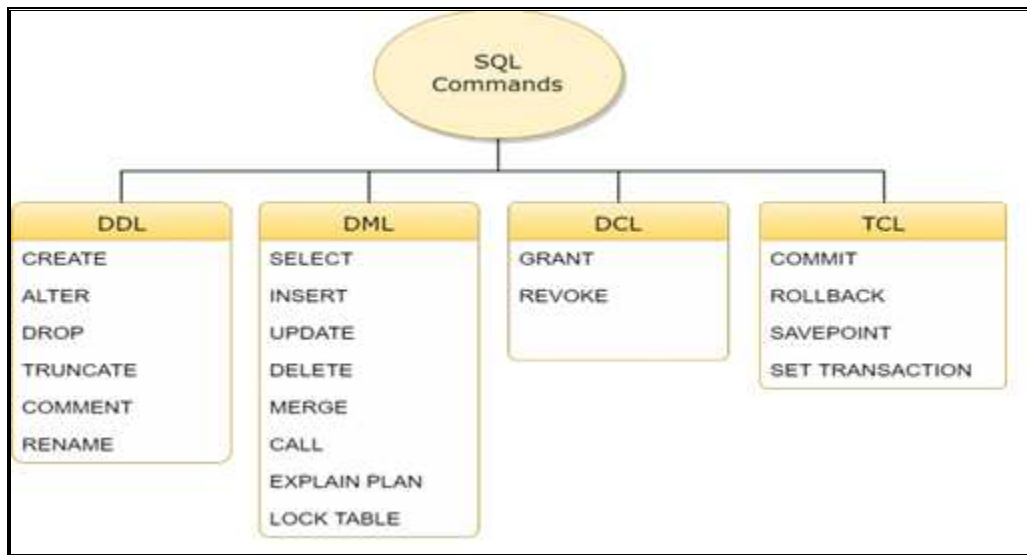
```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($conn->connect_error)
{
    die("Connection failed: " . $conn->connect_error);
}

// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE)
{
    echo "Database created successfully";
}
else
{
    echo "Error creating database: " . $conn->error;
}

$conn->close();
?>
```

Once database is created we can create any numbers of tables as per our requirement. Also we can perform all DDL (**Data Definition Language**), DML (**Data Manipulation Language**) and DCL (**Data Control Language**) Commands to get the desired output.



2.3.3.3 PHP - Date & Time

PHP provides powerful tools for date arithmetic that make manipulating dates easy.

```
<?php
    $date_array = getdate();
    foreach ( $date_array as $key => $val )
    {
        print "$key = $val<br />";
    }
    $formatted_date = "Today's date: ";
    $formatted_date .= $date_array['mday'] . "/";
    $formatted_date .= $date_array['mon'] . "/";
    $formatted_date .= $date_array['year'];
    print $formatted_date;
?>
```

Output:

```
seconds = 20
minutes = 22
hours = 12
mday = 19
wday = 0
mon = 11
year = 2017
yday = 322
weekday = Sunday
```

month = November
0 = 1511074340
Today's date: 19/11/2017

2.3.4 SIMPLE PROGRAMS USING PHP

1. Sum of 10 Numbers

```
<?php
$n=10;
$sum=0;
for($i=1;$i<=$n;$i++)
{
    $sum=$sum+1;
}
echo 'Sum= '.$sum;
?>
```

Output:

Sum=10

2. Largest among three numbers

```
<?php
$a = 10;
$b = 25;
$c = 20;
if($a > $b)
{
    if($a > $c)
    {
        echo "a is biggest number";
    }
}
else
{
    echo "c is biggest number";
}
}
if($b > $c)
{
    echo "b is biggest number";
}
?>
```

Output:

b is biggest number

3. Reverse the string

```
<?php
$my_str = 'welcome';

// Display reversed string
echo strrev($my_str);
?>
```

Output:

emoclew

4. Electricity Bill Calculation

Create a HTML file with name (**Bill.HTML**) create a form like this

```
<form action="data.php" method="post">
Enter current reading: <br/><input type="text" name="cu"/><br/>
Enter previous reading :<br/><input type="text" name="pu"/><br/>
<input type="submit" name="submit" value="submit"/>
</form>
```

Create another file with name (**data.php**) write a code

```
<?php
$cu = $_POST['cu'];
$pu = $_POST['pu'];
$units=$cu-$pu;

if ($units>300)
    $bill=$units*3.5 + $units*0.05/100;

if ($units<=300)
    $bill=$units*3.0;
{
    echo "Electricity Bill: ".$bill;
}
?>
```

5. Date in different format

```
<!DOCTYPE html>
<html>
<body>

<?php
echo "Today is " . date("Y/m/d") . "<br>";
```



```
echo "Today is " . date("Y.m.d") . "<br>";  
echo "Today is " . date("Y-m-d") . "<br>";  
echo "Today is " . date("l");  
?>
```

```
</body>  
</html>
```

Output:

Today is 2017/11/19

Today is 2017.11.19

Today is 2017-11-19

Today is Sunday

REVIEW QUESTIONS

PART - A

1. What is PHP?
2. What are the benefits of PHP?
3. Write the basic syntax of PHP program.
4. List the variable naming conventions in PHP.
5. Is PHP variable and keywords case sensitive.?
6. What are the data types available in PHP?
7. What are the types of operators available in PHP?
8. Define array.
9. How do you create an array in PHP?
10. What are the types of arrays supported by PHP?
11. Define Function
12. Give the general form for defining the function in PHP.
13. What are the methods to pass parameters to the function?
14. Define regular expression
15. What are the two types of form validation?
16. What is the use of die () function?
17. What is MySQL?

PART – B

1. Define constant. What is the main difference between the variable and the constant?
2. What are the three vital components needed in order to develop and run PHP Web pages?
3. Which operator is used as a string concatenation operator? Give an example.
4. Differentiate between numeric array and associative array

5. What are the flow control statements available in PHP? Give the syntax of each statement.
6. What is the difference between require() and include()?
7. Differentiate between Get() and Post() functions
8. List the keywords related to Exception handling.
9. Write a PHP program to print current date in different format.

PART-C

1. Explain any two operators supported by PHP with example.
2. Explain the various conditional statements used in PHP
3. Differentiate while and do while statements with example
4. Explain with example about associative array
5. Explain Passing parameters by values and reference with an example
6. Explain class concept in PHP
7. Explain constructor and destructor with example in PHP
8. Explain string manipulation operation with example in PHP
9. Explain any two PHP's Regular Expression Functions with an example
10. Explain file handling in detail
11. Explain in detail about processing forms in PHP
12. Explain form validation with an example
13. Explain how to Open a Connection to MySQL and create a database from PHP with an example

REFERENCES

1. Rasmus Lerdorf and Levin Tatroe, "Programming PHP", O'Reilly Publications 2002
2. Robin Nixon, "Learning PHP, MySQL & JavaScript With jQuery, CSS & HTML5", O'Reilly Publications 2014
3. <http://dashmeshedu.com/uploads/docNotice134.pdf>
4. <https://www.w3schools.com/php/default>
5. <https://www.tutorialspoint.com/php/>

UNIT – III

MYSQL

LEARNING OBJECTIVES

At the end of this unit, the student will be able to

- Know the basic concepts of Open Source Database.
- Know how to connect MYSQL database and closing connection.
- Write Simple MYSQL Programs.
- Create database and tables in MYSQL.
- Manipulate database tables in MYSQL.
- Understand the concepts of Record Selection technologies
- Know how to work with strings
- Display date and time
- Use order by and group by clauses to sort the query results
- Generate the summary
- Define metadata
- Identify the various types of metadata
- Know how to create and drop sequences
- Understand the basics of MYSQL and WEB

3.1 INTRODUCTION

MySQL is the most popular open source SQL database management system (DBMS). It is a fast, reliable, easy-to-use, multi-user and multi-threaded relational database system. It is freely available and released under GPL (GNU General Public License).

A database is a collection of data that is organized so that its contents can be easily accessed. MySQL is a data storage area. In this storage area, there are small sections called Tables. The data in MySQL is stored in database objects called tables.

A table is a collection of data entries and it consists of columns and rows. The MySQL database has become the world's popular open source database because of its consistent fast performance, high reliability and ease of use.

Advantages

- MySQL is Cross-Platform. It runs on more than 20 platforms including Linux, Windows, Mac OS, Solaris etc.,
- MySQL is fast.
- MySQL is free. It is open source software.
- Reliable and easy to use.
- Multi-Threaded, multi-user and robust SQL Database server.

Disadvantages

- MySQL tends to be somewhat less reliable than its peers.
- MySQL has a troubling tendency to come grinding to a halt if it's forced to deal with too many operations at a given time.
- Its functionality tends to be heavily dependant On Addons
- The database isn't fully SQL-compliant, and tends to be limited in areas including data warehousing, fault tolerance, and performance diagnostics

3.1.1 SETTING UP AN ACCOUNT

In order to provide access to the MySQL database you need to create an account for connecting to the MySQL server running on a given host. Use the create statement to set up the MySQL user account. Then use the user name and password to make connections to the server. User names, as used by MySQL for authentication purposes, have nothing to do with user names (login names) as used by Windows or UNIX.

MySQL user names can be up to 16 characters long. MySQL passwords have nothing to do with passwords for logging in to your operating system.

3.1.1.1 Account Management Statements

The various account management statements used are

- CREATE USER
- DROP USER
- RENAME USER
- REVOKE
- SET PASSWORD

CREATE USER

The CREATE USER statement creates new MySQL account.

Syntax:

```
CREATE USER user [IDENTIFIED BY [PASSWORD] 'password'] [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
```

Example:

```
Mysql>CREATE USER 'monty'@'localhost' IDENTIFIED BY 'some_pass';
```

```
Mysql> GRANT all privileges on *.* to 'monty'@'localhost';
```

The GRANT statement enables the system administrator to grant privileges to user accounts

DROP USER

The DROP USER statement removes one or more MYSQL accounts and their privileges.

Syntax:

```
DROP USER user [, user] ...
```

Example:

```
DROP USER 'jeffrey'@'localhost';
```

RENAME USER

The RENAME USER statement renames the existing MYSQL accounts

Syntax

```
RENAME USER old_user TO new_user [, old_user TO new_user] ...
```

Example

```
RENAME USER 'jeffrey'@'localhost' TO 'jeff'@'127.0.0.1';
```

REVOKE

The REVOKE statement enables the system administrator to revoke privileges from MYSQL accounts.

Syntax

```
REVOKE priv_type [column list] from user[,user...]
```

SET PASSWORD

The SET PASSWORD statement Assigns a password to an existing MYSQL user.

Syntax

```
SET PASSWORD [FOR user] =  
{
```

```
PASSWORD ('some password')| OLD_PASSWORD ('some password')| 'encrypted password'
}
```

Example:

```
SET PASSWORD FOR 'bob'@'localhost' = PASSWORD ('newpass');
```

3.1.2 STARTING, TERMINATING AND WRITING YOUR OWN SQL PROGRAMS

3.1.2.1 Starting MySQL

The MYSQL server can be started manually from the command line. To start the mysqld server from the command line, you should start a console window and enter this command.

```
C:\Program Files\MYSQL\MySQL Server5.0\bin\mysqld
```

The path to mysqld may vary depending on the installed location of MySQL on your System.

Login

To start the MySQL program, try just typing its name at your command-line prompt. If MySQL starts up correctly, you'll see a short message, followed by a mysql> prompt that indicates the program is ready to accept queries.

```
% MySQL
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 18427 to server version: 3.23.51-log
```

```
Type 'help;' or '\h' for help.Type '\c' to clear the buffer.
```

```
mysql>
```

If MySQL tries to start but exits immediately with an “access denied” message, you'll need to specify connection parameters. The most commonly needed parameters are the host to connect to (the host where the MySQL server runs), your MySQL username, and a password.

For example:

```
% Mysql -h localhost -p -u cbuser
```

```
Enter password: cbpass
```

3.1.2.2 Terminating (Stopping) MySQL

To terminate a MySQL session, issue a QUIT statement:

```
mysql>QUIT
```

You can also terminate the session by issuing an EXIT statement

3.1.2.3 Writing your Own SQL programs

We will create a simple login system using PHP code on our pages, and a MySQL database to store our user's information.

```
<?php
$host="localhost";
$username="";
$password="";
$db_name="test";
$tbl_name="members";
//Connect to server and select database
mysql connect (" $host", "username", "$password") or die("cannot connect");
mysql select db("$db_name") or die("cannot select DB");
//username and password sent from form
$myusername = $_POST["myusername"];
$mypassword=$_POST["mypassword"];
$sql="Select * from $tbl_name where username='$myusername'
and password ='$mypassword';
$result = mysqli_query($sql);

// mysqli_num_rows counting table row
$count=mysqli_num_rows($result);
if($count==1)
{
    session_register("myusername");
    session_register("mypassword");
    header("location:login_success.php");
}
else
{
    echo "wrong username and password";
}
?>
```

3.1.3 RECORD SELECTION TECHNOLOGY

The SELECT statement is used to select data from a database. The statement begins with the SELECT keyword. The basic SELECT statement has 3 clauses:

- SELECT
- FROM

- WHERE

The select clause specifies the table columns that are retrieved. The FROM clause specifies the tables accessed. The WHERE clause specifies which table rows are used.

Example

Select name from emp where age = 41;

This query accesses rows from the table emp. It then filters those rows where the age column contains the value 41. Finally the query retrieves the name column from each filtered row.

Output

This query produces

Name
Anand
Balu

3.1.3.1 Specifying which column to display

- To display all columns from a table, use * as shortcut that selects all columns.
- To display some columns from a table, use only the columns that you want to see explicitly.

Example

Mysql> select * from emp;

Name	Age	Exp
Anand	41	10
Aravind	45	12
Balu	41	5
Ramesh	50	20

Mysql> select Name, Age from emp;

Name	Age
Anand	41
Aravind	45
Balu	41
Ramesh	50

3.1.3.2 Giving names to output columns

Whenever you retrieve a result set, MySQL gives every output column a name

Mysql> SELECT Name as EmpName, Age as EmpAge FROM emp;

EmpName	EmpAge
Anand	41
Aravind	45
Balu	41
Ramesh	50

3.1.3.3 Specifying which row to select

Add a WHERE clause to the query that indicates to the server which rows to return.

Mysql> SELECT name,age FROM emp WHERE name="Anand";

name	age
Anand	41

3.1.3.4 Removing Duplicate Rows

Use DISTINCT in a query will eliminate duplicate values.

Example

Mysql>Select name from emp;

Name
Anand
Aravind
Balu
Ramesh
Anand

Mysql>Select Distinct name from emp;

name
Anand
Aravind
Balu
Ramesh

3.1.4 WORKING WITH STRINGS

A string can be binary or non-binary. Binary strings are used for raw data such as images, music files or encrypted values. Non binary strings are used for character data such as text and are associated with character set.

Data types for binary strings are BINARY, VARBINARY and BLOB. Data types for non-binary strings are CHAR, VARCHAR and TEXT.

3.1.4.1 String Properties

- Strings can be case sensitive (or not), which can affect the outcome of string operation.
- You can compare entire string or just parts of them by extracting substrings.
- You can apply pattern-matching operations to look for strings that have certain structure.

3.1.4.2 Types of strings

MySQL can operate on regular strings or binary strings. "Binary" in this context has little to do with the presence of non-ASCII values, so it's useful right at the outset to make a distinction.

- Binary *data* may contain bytes that lie outside the usual range of printable ASCII characters.
- A binary *string* in MySQL is one that MySQL treats as case sensitive in comparisons. For binary strings, the characters **A** and **a** are considered different. For non-binary strings, they're considered the same.

3.1.4.3 String Data types

The various string data types supported in MySQL are

- CHAR
- VARCHAR
- TINYTEXT
- TEXT
- BLOB
- MEDIUMTEXT
- LONGTEXT
- BINARY
- VARBINARY
- ENUM
- SET

CHAR ()

It is a fixed length string and is mainly used when the data is not going to vary much in its length. It ranges from 0 to 255 characters long. While storing values they are right padded with spaces to the specified length.

VARCHAR ()

It is a variable length string and is mainly used when the data may vary in length. It ranges from 0 to 255 characters long. VARCHAR values are not padded when they are stored.

TINYTEXT

String with maximum length of 255 characters.

TEXT

TEXT columns are treated as character string. It contains a maximum of 65535 characters.

BLOB

BLOB stands for Binary Large Objects. It can hold a variable amount of data. BLOB columns are treated as byte strings. It contains a maximum of length of 65535 characters.

MEDIUM TEXT

It has a maximum length of 16777215 characters. (2^{24})

LONG TEXT

It has a maximum length of 4294967295 characters. (2^{32})

BINARY

THE BINARY is similar to the CHAR type. It stores the values as binary byte strings instead of non-binary character string.

VARBINARY

The VARBINARY is similar to VARCHAR type. It stores the values as binary byte string instead of non-binary character strings.

ENUM()

Enumeration data type. Each column may have one of the specified possible values. It can store one of the values that are declared in the specified list contained in the () brackets. The ENUM() ranges up to 65535 values.

SET()

In a set each column may have more than one of the specified possible values. It contain up to 64 list items and can store more than one choice

3.1.5 DATE AND TIME

MySQL supports a number of date and time column formats. For the entire date and time column we can assign the values using either string or number.

- DATE
- TIME
- DATETIME
- TIMESTAMP
- YEAR

DATE

The range is 1000-01-01 to 9999-12-31. The date values are displayed in YYYY-MM-DD format.

TIME

The time values are displayed in HH:MM:SS format.

DATETIME

The date time values are displayed in YYYY-MM-DD HH:MM:SS format.

TIMESTAMP

A **TIMESTAMP** column is useful for recording the date and time of an **INSERT** or **UPDATE** operation.

YEAR

The year values are displayed either in two digit or four digit format.

3.1.5.1 Date and Time Functions

In addition to providing mechanism for storing dates and times, MySQL also provides wide range of functions that can be used to manipulate dates and times. The following table provides some common functions available for working with times and dates in MySQL.

Function	Description
ADDDATE()	Add Dates
ADDTIME()	Add time
CONVERT_TZ()	Convert from one time zone to another
CURDATE()	Returns the current date
CURTIME()	Returns the current time
DATE_ADD()	Add two dates
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract two dates
DATE()	Extract the date part of the date

DATEDIFF()	Subtract two dates
DAYNAME()	Returns the name of the weekday
HOUR()	Extract the hour
MAKEDATE()	Create a date from the year
MONTHNAME()	Returns the name of the month
NOW()	Returns the current date and time
TIME_FORMAT()	Format as time

3.1.5.2 Display date and time

DATE_FORMAT accepts two arguments: a date and a string describing how to display the values.

For example

%Y, %M,%d signifies the four-digit year, the month name and the two-digit day of the month respectively.

Mysql> [SELECT CURRENT_DATE](#), DATE_FORMAT([CURRENT_DATE](#), "%M %d %Y")

[CURRENT_DATE](#) [DATE_FORMAT\(CURRENT_DATE, "%M %d %Y"\)](#)

2017-11-20

November 20 2017

3.1.6 SORTING QUERY RESULTS MODULE

The SQL select command is used to fetch the data from MYSQL table. When you select rows, the MySQL server is free to return them in any order, unless you instruct it otherwise by saying how to sort the result. But, you sort a result set by adding an ORDER BY clause that names the column or columns which you want to sort.

3.1.6.1 USING ORDER BY to sort query results

ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns.

Syntax

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, ..columnN] [ASC | DESC];
```

Consider the CUSTOMERS table having the following records –

ID	Name	Age	Address	Salary
1	Ramesh	32	Ahmedabad	2000
2	Khilan	25	Delhi	1500
3	Kaushik	23	Kota	2000
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bhopal	8500
6	Kamal	22	Chennai	10500
7	Muffy	24	Indore	10000

The following code block has an example, which would sort the result in an ascending order by the NAME and the SALARY

```
SQL> SELECT * FROM CUSTOMERS ORDER BY NAME, SALARY;
```

This would produce the following result

ID	Name	Age	Address	Salary
4	Chaitali	25	Mumbai	6500
5	Hardik	27	Bhopal	8500
6	Kamal	22	Chennai	10500
3	Kaushik	23	Kota	2000
2	Khilan	25	Delhi	1500
7	Muffy	24	Indore	10000
1	Ramesh	32	Ahmedabad	2000

The following code block has an example, which would sort the result in the descending order by NAME.

```
SQL> SELECT * FROM CUSTOMERS ORDER BY NAME DESC;
```

ID	Name	Age	Address	Salary
1	Ramesh	32	Ahmedabad	2000
7	Muffy	24	Indore	10000
2	Khilan	25	Delhi	1500
3	Kaushik	23	Kota	2000
6	Kamal	22	Chennai	10500
5	Hardik	27	Bhopal	8500
4	Chaitali	25	Mumbai	6500

3.1.7 GENERATING SUMMARY

Summaries are useful when you want the overall pictures rather than details. Using aggregate functions we can achieve summary of values. Aggregate functions available are COUNT (), MIN (), MAX (), SUM (), AVG () and GROUP BY clause.

3.1.7.1 Summarizing with COUNT ()

The COUNT() function returns the number of rows that matches a specified condition

Syntax

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

Example

```
Mysql> select count(*) from emp;
```

```
Count (*)
30
```

3.1.7.2 Summarizing with MIN () and MAX ()

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

Example

```
Mysql>select MIN(sal) as Low, MAX(sal) as High from emp;
```

```
Low  High
100  500
```

3.1.7.3 Summarizing with SUM() and AVG()

The SUM() function returns the total sum of a numeric column.

The AVG() function returns the average value of a numeric column.

Syntax

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

Syntax

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

Example

Mysql> select count(ID) as 'NoofCustomers', AVG(salary) as 'AvgSal' from Customer;

'NoofCustomers'	'AvgSal'
7	5857.1429

3.1.7.4 Dividing a summary into subgroups

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Example

Mysql> select Name, Test, Score from exam;

Name	Test	Score
Arun	A	47
Chitra	B	50
David	C	NULL
Ram	D	NULL

Mysql> select Name from exam where score is NULL group by name;

name
David
Ram

3.1.8 WORKING WITH METADATA

Metadata is data about data. Anything that *describes* the database is metadata. Thus column names, database names, user names, version names, and most of the string results from SHOW are metadata.

For example - Consider a file with a picture. The picture or the pixels inside the file are data. A description of the picture, like "JPEG format, 300x400 pixels, 72dpi", is metadata, because it describes the content of the file, although it's not the actual data.

Representation of metadata must satisfy these requirements:

All metadata must be in the same character set. Otherwise, neither the SHOW statements nor SELECT statements for tables in INFORMATION_SCHEMA would work properly because different rows in the same column of the results of these operations would be in different character sets.

Metadata must include all characters in all languages. Otherwise, users would not be able to name columns and tables using their own languages.

3.1.8.1 Types of Metadata

- Information about the result of queries
- Information about tables and databases.
- Information about the MySQL server.

3.19 USING SEQUENCES

A sequence is a database object that generates numbers in sequential order. Applications most often use these numbers when they require a unique value in a table such as primary key values.

The following list describes the characteristics of sequences.

- Sequences are available to all users of the database
- Sequences are created using SQL statements
- Sequences have a minimum and maximum value. They can be dropped, but not reset.
- Once a sequence returns a value, the sequence can never return the same value
- Sequences increment by an amount specified when created

3.1.9.1 Creating a Sequence

The general form to create a sequence is

```
CREATE SEQUENCE sequence_name
[INCREMENT BY #]
[START WITH #]
[MAXVALUE # | NOMAXVALUE]
[MINVALUE # | NOMINVALUE]
[CYCLE | NOCYCLE];
```

Where

CREATE, SEQUENCE Keywords

Sequence name Name of the sequence

INCREMENT BY n Increment value. N is a positive or negative integer. If n is positive it creates a sequence in ascending order. If n is

	negative it creates a sequence in descending order
START WITH n	Gives the start value n of the sequence
MAX VALUE n	Gives the maximum value n of the sequence
MIN VALUE n	Gives the minimum value n of the sequence
CYCLE	To generate the sequence from the beginning after reaching maximum value
NOCYCLE	The default value. It generates an error when maximum value of the sequence is reached
CACHE	To keep a set of n sequence numbers in memory, so that they can be accessed fast. The default value is 20
NOCACHE	This is the default value. To specify that the sequence number is created only when needed

3.1.9.2 Dropping a Sequence

To drop a sequence, execute a Drop Sequence Statement. Use this function when a sequence is no longer useful or to reset a sequence to an older number. To reset a sequence, first drop the sequence and then recreate it.

DROP SEQUENCE command is used to remove the sequence from the database. The general form is

```
DROP SEQUENCE sequence-name;
```

3.1.9.3 Using a Sequence

Use sequences when an application requires a unique identifier. INSERT statements, and occasionally UPDATE statements, are the most common places to use sequences. Two functions are available on sequences:

- NEXTVAL: Returns the next value from the sequence.
- CURRVAL: Returns the value from the last call to NEXTVAL by the current user during the current connection.

Examples

To create the sequence:

```
CREATE SEQUENCE customer_seq INCREMENT BY 1 START WITH 100
```

To use the sequence to enter a record into the database:

```
INSERT INTO customer (cust_num, name, address)
VALUES (customer_seq.NEXTVAL, 'Kalam', '123 Gandhi Nagar.')
```

To drop the sequence:

```
DROP sequence sequence-name;
```

3.1.10 MYSQL AND WEB

MySQL makes it easier to provide dynamic rather than static content. Static content exists as pages in the web server's document that are served exactly as is. Visitors can access only the documents that you place in the tree, and changes occur only when you add, modify, or delete those documents. By contrast, dynamic content is created on demand.

Basic Web Page Generation

Using HTML we can generate your own Web site. HTML is a language for describing web pages.

- HTML stands for Hyper Text Markup Language
- HTML is not a programming language, it is a markup language
- A markup language is a set of markup tags
- HTML uses markup tags to describe web pages
- HTML documents describe web pages
- HTML documents contain HTML tags and plain text
- HTML documents are also called web pages.

Here is a very simple HTML page that specifies a title in the page header, and a body with white background containing a simple paragraph:

```
<Html>
  <Body>
    <H1> My Heading </H1>
    Welcome to my web page
    <P> My paragraph </P>
    <body bgcolor = "green">
  </Body>
</Html>
```

Static Web Page

A static web page shows the required information to the viewer, but do not accept any information from the viewer.

Dynamic Web Page

A dynamic web page displays the information to the viewer and also accepts the information from the user. Railway reservation, online shopping etc are examples of dynamic web page.

Client side scripting

It is a script, (ex. Javascript, VB script), that is executed by the browser (i.e. Firefox, Internet Explorer, Safari, Opera, etc.) that resides at the user computer.

Server side scripting

It is a script (ex. ASP .NET, ASP, JSP, PHP, Ruby, or others), executed by the server (Web Server), and the page that is sent to the browser is produced by the server-side scripting.

Using Apache to Run Web Scripts

- Open-Source Web server originally based on NCSA server (National Center for Supercomputing Applications).
- Apache is the most widely used web server software package in the world.
- Apache is highly configurable and can be setup to support technologies such as, password protection, virtual hosting (name based and IP based), and SSL encryption.

3.2 PHP AND SQL DATABASE: PHP AND LDAP

LDAP is the Lightweight Directory Access Protocol, and used to access "Directory Servers". The Directory is a special kind of database that holds information in a tree structure.

The concept is similar to your hard disk directory structure, except that in this context, the root directory is "The world" and the first level subdirectories are "countries". Lower levels of the directory structure contain entries for companies, organizations or places, while yet lower still we find directory entries for people, and perhaps equipment or documents.

To refer to a file in a subdirectory on your hard disk, you might use something like:

```
/usr/local/myapp/docs
```

The forwards slash marks each division in the reference, and the sequence is read from left to right.

The equivalent to the fully qualified file reference in LDAP is the "distinguished name", referred to simply as "dn". An example dn might be:

```
cn=John Smith,ou=Accounts,o=My Company,c=US
```

The comma marks each division in the reference, and the sequence is read from right to left. You would read this dn as:

```
country = US  
organization = My Company  
organizationalUnit = Accounts  
commonName = John Smith
```

In the same way as there are no hard rules about how you organize the directory structure of a hard disk, a directory server manager can set up any structure that is meaningful for the purpose. However, there are some conventions that are used. The message is that you

cannot write code to access a directory server unless you know something about its structure, any more than you can use a database without some knowledge of what is available.

3.2.1 PHP Connectivity

```
<?php
echo "<h3> LDAP query test </h3>";
echo "Connecting...";
$ds=ldap_connect("localhost");
echo "connect result is". $ds. "<br/>";
if($ds)
{
echo "Binding";
$r=ldap_bind($ds);
echo "Bind result is".$r."<br/>";
echo "searching for (sn=S*)...";
$sr=ldap_search($ds,"o=My company,c=US","sn=S*");
echo "Search result is" . $sr."<br/>";
echo "Number of entries returned is" .ldap_count_entries($ds,$sr). "<br/>";
echo "Getting entries...<p>";
$info=ldap_get_entries($ds,$sr);
echo "Data for" . $info["count"]. "Items returned: <p>";
for($i=0;$i<$info["count"];$i++)
{
echo "dn is:" . $info[$i]["dn"]. "<br/>";
echo "first cn entry is:" . $info[$i]["cn"][0]. "<br/>";
echo "first email entry is:" . $info[$i]["mail"][0]. "<br/><hr/>";
}
echo "Closing connection";
ldap_close($ds);
}
else
{
echo "<h4>unable to connect to LDAP server </h4>";
}
?>
```

3.2.2 SENDING AND RECEIVING EMAILS

Email is the most popular Internet Service today. A plenty of emails are sent and delivered each day.

The PHP mail() Function

The PHP mail() function is used to send emails from inside a script.

Syntax

mail(to,subject,message,headers,parameters)

Parameter	Description
To	Required. Specifies the receiver/receivers of the email
Subject	Required. Specifies the subject of the email. Note: This parameter cannot contain any newline characters
Message	Required. Defines the message to be sent. Each line should be separated with a LF (\n). Lines should not exceed 70 characters
Headers	Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n)
Parameters	Optional. Specifies an additional parameter to the sendmail program

Note: For the mail functions to be available, PHP requires an installed and working email system. The program to be used is defined by the configuration settings in the php.ini file.

PHP Simple E-Mail

The simplest way to send an email with PHP is to send a text email.

In the example below we first declare the variables (\$to, \$subject, \$message, \$from, \$headers), then we use the variables in the mail() function to send an e-mail:

```
<?php
$to = "someone@example.com";
$subject = "Test mail";
$message = "Hello! This is a simple email message.";
$from = "someoneelse@example.com";
$headers = "From:" . $from;
mail($to,$subject,$message,$headers);
echo "Mail Sent.";
?>
```

PHP Mail Form

With PHP, you can create a feedback-form on your website. The example below sends a text message to a specified e-mail address:

```
<html>
<body>
<?php
if (isset($_REQUEST['email']))
//if "email" is filled out, send email
{
//send email
$email = $_REQUEST['email'] ;
$subject = $_REQUEST['subject'] ;
$message = $_REQUEST['message'] ;
mail("someone@example.com", $subject, $message, "From:" . $email);
echo "Thank you for using our mail form";
}
```

```

else
//if "email" is not filled out, display the form
{
echo "<form method='post' action='mailform.php'>
Email: <input name='email' type='text'><br>
Subject: <input name='subject' type='text'><br>
Message:<br>
<textarea name='message' rows='15' cols='40'>
</textarea><br>
<input type='submit'>
</form>";
}
?>
</body>
</html>

```

This is how the example above works:

- First, check if the email input field is filled out
- If it is not set (like when the page is first visited); output the HTML form
- If it is set (after the form is filled out); send the email from the form
- When submit is pressed after the form is filled out, the page reloads, sees that the email input is set, and sends the email

Note: This is the simplest way to send e-mail, but it is not secure.

Reading Emails with PHP

```

<php
Class email_reader
{
public $conn;
private $inbox;
private $msg_cnt;
private $server='yourserver.com';
private $user='email@yourserver.com';
private $pass='yourpassword';
private $port=143;

function __construct()
{
$this->connect();
$this->inbox();
}
function close()
{
$this->inbox=array();
Imap_close($this->conn);
}
}

```

```

}
function connect()
{
$this->conn=imap_open('{'.$this->server.'/notls}', $this->user, $this->pass);
}
function move($msg_index, $folder='INBOX.Processes')
{
Imap_mail_move($this->conn, $msg_index, $folder);
Imap_expunge($this->conn);
$this->inbox();
}
function get($msg_index=NULL)
{
If(count($this->inbox)<=0)
{
return array();
}
elseif(!is_null($msg_index)&&isset($this->inbox[$msg_index]))
{
return $this->inbox[$msg_index];
}
return $this->inbox[0];
}
function inbox()
{
$this->msg_cnt=imao_num_msg($this->conn);
$in=array();
for($i=1; $i<=$this->msg_cnt; $i++)
{
$in[]=array(
'index' => $i,
'header' =>imap_headerinfo($this->conn, $i),
'body' =>imap_body($this->conn, $i),
'structure' =>imap_fetchstructure($this->conn, $i)
);
}
$this->inbox=$in;
}
}
?>

```

The inline comments indicates the function of each segment of the code. The important method is inbox(). The IMAP inbox is much like an array with a numbered key starting at 1. In the inbox() method, I store that index so that the email can be moved, deleted, or read again later.

Next, the header is stored with the function `imap_headerinfo()`. This pulls down an object from the server containing information like the Subject, From: address, To: address, and text encoding type. Using `imap_body()`, the body text of the email is retrieved.

REVIEW QUESTIONS

PART – A (2 marks)

1. How to add new user in MySQL?
2. Which command is used to give the privileges to the users?
3. What is the use of revoke statement?
4. What is the use of grant statement?
5. How to remove duplicate rows from the table?
6. List any two record selection technology
7. What are the three basic clauses of select statements?
8. List any two string properties in MySQL
9. How to display date and time in MySQL?
10. What is the use of order by clause?
11. How to count the number of rows in an entire table?
12. What is meant by metadata?
13. What is meant by sequence?
14. What is HTML?
15. Define static web page.
16. Define dynamic web page

PART – B (3 marks)

1. How to start and terminate MySQL
2. List any two record selection technologies.
3. What is the use of min () & max ()?
4. What is the use of group by statement? Give an example.
5. How to create and drop sequence?
6. Differentiate client side scripting and server side scripting

PART-C (5 marks)

1. Explain about setting up an account in MySQL.
2. Write a PHP program for simple login system in MySQL
3. Discuss record selection technology with example.
4. Discuss data and time data types.
5. Explain sorting query result.
6. Explain how will you generate summary
7. Explain sequence with example
8. Explain MySQL and web.

REFERENCES

1. Steve Suchring, "MySQL Bible" John Wiley sons 2002
2. <http://http://codingcyber.com>
3. <https://www.tutorialspoint.com/mysql>
4. <http://www.mysqltutorial.org/basic-mysql-tutorial.aspx>

UNIT – IV

Python

Objectives:

- Learn the fundamentals of Python
- Understand the data types and control structures.
- Understand and learn to manipulate List, Tuple, Set and Dictionary
- Get familiarize with Exception handling and Files

4.1 Basic features of Python:

What is Python?

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

4.1.1 Basic features of Python:

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming.

Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

Features of Python:

Simple: Python is a simple language. This pseudo-code nature of Python is one of its greatest strengths.

Easy to Learn: Python is very easy to get started with and has an extraordinarily simple syntax.

Free and Open Source : Python is an example of an open source software. In simple terms, you can freely distribute copies of the software, read the source code, make changes to it and use pieces of it in new free programs. Open source is based on the concept of a community which shares knowledge.

High-level Language : When you write programs in Python, you do not need to bother about low-level details such as managing the memory used by your program, etc.

Portable : Due to its open source nature, Python has been ported to (i.e. changed to make it work on) many platforms. All your Python programs can work on any of these platforms without

requiring any changes as long as you are careful to avoid any system-specific features. You can use Python on Linux, Windows, Pocket PC and even Nokia Series 60 mobile phones!

Interpreted: A program written in a compiled language like C or C++ is converted from the source language (i.e. C or C++) into a language that is spoken by your computer (binary code i.e. 0s and 1s) using a compiler with various flags and options. When you run the program, the linker/loader software copies the program from the hard disk to memory and starts running it. Python, on the other hand, does not need the compilation and linking/loading steps. You just *run* the program directly from source code. Internally, Python converts the source program into an intermediate form called byte codes and then translates this into the native language of your specific system and then runs it.

Object Oriented: Python supports both procedure-oriented programming as well as object-oriented programming. In *procedure-oriented* programming, the program is built around procedures or functions which are just reusable pieces of programs to which data is fed. In *object-oriented* programming, the program is built around objects which combine both data and functionality. Python has a very powerful but simplistic way of doing object-oriented programming, especially when compared to big languages like C++ or Java.

Extensible: If you need a critical piece of code in your program to run very fast or want to have a piece of algorithm to be hidden from the outside world, then you can write that part of the program in languages like C or C++ and then use that part from your Python programs.

Embeddable: You can embed Python in your programs written in other languages like C or C++ to give 'scripting' capabilities for your program's users.

Extensive Libraries: The Python Standard Library is huge. It can help you with regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, FTP, email, XML, HTML, WAV files, cryptography, GUI using Tk, and many other system-specific functionality as well. All these modules are part of the standard Python installation.

4.1.2 Installing Python

4.1.2.1 For Windows users

Using Python at the Windows prompt

In order to use Python from the Windows DOS prompt, we need to set the PATH environment variable appropriately.

For Windows 2000, XP and 2003, click on Control Panel -> System -> Advanced -> Environment Variables. In the System Variables section, click on the variable named PATH, then select Edit and add C:\Python24 to the end of the information already present. Make sure you are adding the correct folder path. For older versions of Windows, add the line PATH=%PATH%; C:\Python24 to the file C:\AUTOEXEC.BAT and restart the system. For Windows NT, use the AUTOEXEC.NT file.

4.1.2.2 For Linux/BSD users

If you are using a Linux distribution (such as Fedora, Mandrake, Debian, Gentoo, etc.) or a BSD system (such as FreeBSD, OpenBSD, etc.), then you probably already have Python installed on your system.

To test if you have Python installed, open a shell program (such as konsole, gnome-terminal, etc.) and enter the command **python -V** and press **enter** key (notice the capital 'V').

```
$ python -V
Python 2.4
```

Note

\$ is the prompt of the shell. This will be different for different systems, hence I will indicate the prompt by just the \$ symbol.

If you see such version information, then you have Python installed already.

However, if you get a message like this:

```
$ python -V
bash: python: command not found
```

Then, Python is not installed already. This is highly unlikely but possible.

In this case, you have two ways of installing Python on your system.

- Install Python using the package management software that comes with your system. Examples of such software are yum in Fedora, urpmi in Mandrake, apt-get in Debian, emerge in Gentoo, and pkg_add in FreeBSD. Note that you need an internet connection on your system to use this method of installation.
- You can compile and build Python from the [source code](#) and then install it. The instructions are pretty straightforward and are part of the README document that comes with the source code.

4.2 Basics of Python

4.2.1 Literal Constants

An example of a literal constant is a number like 5, 1.23, 2.95e-3 or a string like 'This is a string' or "It's a string!". It is called a literal because you use its value *literally*, for example, the number 2 always represents itself and nothing else. It is a *constant* because its value cannot be changed. Hence, they are referred to as *literal constants*.

We will encounter two types of literal constants - numbers and strings.

Numbers

Numbers in Python can be of four types:

- Integers such as 2 which are just whole numbers.
- Long integers, which are just bigger whole numbers.
- Floating point numbers (or *floats*) such as 3.23 and 52.3E-4. The E notation indicates power of 10. In this case, 52.3E-4 means $52.3 * 10^{-4}$.
- Complex numbers such as (-5+4j) and (2.3 - 4.6j).

Booleans There are two types of Boolean values - True and False, they are used to check whether a condition is valid or not respectively.

Strings A string is a *sequence of characters*. In other words, a string is basically a bunch of words and/or symbols.

You will use strings in almost every program that you will write, hence it is very important to understand strings properly.

- **Using Single Quotes (')**

You can specify strings using single quotes such as 'Quote me on this'.

- **Using Double Quotes (")**

Strings in double quotes work exactly the same way as strings in single quotes. An example is "What's your name?"

- **Using Triple Quotes (""" or """)**

You can specify multi-line strings using triple quotes. You can use single quotes and double quotes freely within the triple quotes (that is quite a mouthful to say!). An example is shown below.

```
"""This is a multi-line string. This is the first line.  
This is the second line.  
"What's your name?," I asked.  
He said "Bond, James Bond."  
"""
```

- **Escape Sequences**

Suppose you wanted to use a string which contains a single quote ('), how will you specify the string? For example, the string is What's your name? You cannot use 'What's your name?' since Python will be confused as to where the string starts and ends. In this

case, we need to tell Python that the second single quote does not specify the end of the string. We do this by putting a backslash before that single quote. So, we use the string as 'What\'s your name?'. The sequence \' is called an escape sequence because it causes the single quote to *escape* its special meaning. There are many other types of escape sequences as well.

For example:

```
"This is the first sentence. \  
This is the second sentence."
```

is equivalent to "This is the first sentence. This is the second sentence."

- **Raw Strings** *If you need to specify some strings as-is, that is, without any special processing for escape sequences, then you can specify the string as a raw string by prefixing r or R to the string. An example is r "Newlines are indicated by \n".*
- **Unicode Strings** Unicode is a standard way of writing text in multiple human languages. If you want to write text in your native language such as Kannada or Italian, then you need to have a Unicode-enabled text editor. Similarly, Python gives you the ability to handle Unicode text, all you need to do is prefix u or U to the string. An example is u "This is a Unicode string".

Remember to use Unicode strings when you are dealing with text files, especially when you know that the file will contain text written in languages other than English.

- **Strings are immutable**

Immutable means 'cannot be changed'. This means that once you create a string, you cannot change it. However, you can create new strings based on this string.

- **Concatenation of string literals**

If you place two string literals side by side, they are automatically concatenated i.e. a new string equivalent to joining of the two strings is given by Python. For example, 'What\'s 'your name?' is concatenated to get "What's your name?"

4.2.2 Variables

Using just literal constants can soon become boring since your program will give the same results every time. We need some way of storing information and manipulating them so that we can do interesting things. We can achieve this using *variable*. Variables are exactly what they mean, their value can vary - you can store any value using a variable. Variables are just

parts of your computer's memory which you are using to store some information. Unlike literal constants, we need some way to use these variables and hence we give names for variables.

Identifier Naming

Variables are examples of identifiers. *Identifiers* are names given to identify *something*. We can choose any name for identifiers as long as they satisfy these simple rules:

- The name can consist of alphabets (both lower and uppercase), underscore ('_') and digits (0-9).
- Only the first character of the name cannot be a digit.
- Names are case-sensitive. For example, myname and myName are **not** the same names, they are two different variables. Note the lowercase n in the former and the uppercase N in the latter.
- Examples of *valid* identifier names are i, __my_name, name_23 and a1b2_c3.
- Examples of *invalid* identifier names are 2things, this is spaced out, and my-name.

Data Types

Variables can hold values of different types, which are called as **data types**.. The basic types are numbers and strings which we have already discussed.

Objects

Python refers to anything used in a program as an *object*. What it really means is that Python is strongly object-oriented in the sense that everything in Python is an object including numbers, strings and even functions.

We have not yet explored what an object really means; we will explore it in later chapters. However, it is **very** important to keep the idea of an object in mind. You can picture an object like a bag where you can put in stuff or take out stuff.

Using Variables

Example 4.1. Using Variables and Literal Constants

```
#!/usr/bin/env python
# Filename: var.py

i = 5
print i
i = i + 1 # add number 1 to value of i and then store again in i
print i
```

```
s = "This is a multi-line string.  
This is the second line."  
print s
```

Output

```
$ python var.py  
5  
6  
This is a multi-line string.  
This is the second line.
```

4.2.3 Operators and Expression

Introduction

In Python, an expression is something that conveys a message to the interpreter. We can also consider the message as an operation since we are telling the interpreter what to do. In order to specify an operation, we need operators and operands.

Operators are functionality that do something and can be represented by symbols or by special keywords. Operators require some data to operate on and such data are called *operands*.

A simple example is $2 + 3$ where 2 and 3 are the operands and + is the operator.

Operators

We will briefly take a look at the operators and their usage.

Tip

You can evaluate expressions using the interactive prompt. For example, run $2+3$ at the prompt and get instant results.

```
>>> 2 + 3  
5  
>>> 3 * 5  
15
```

4.2.3.1 Operators and their usage

Operator	Name	Explanation	Examples
----------	------	-------------	----------

Operator	Name	Explanation	Examples
+	Plus	Adds the two objects	3 + 5 gives 8. 'a' + 'b' gives the string 'ab'.
-	Minus	'Subtract' one object's value from the other. It is also used for negative numbers.	-5.2 gives a negative number. 50 - 24 gives 26.
*	Multiply	Gives the 'multiplication' of two objects. Strings are repeated.	2 * 3 gives 6. 'la' * 3 gives the string 'lalala'.
**	Power	Returns x to the power of y	3 ** 4 gives 81 (i.e. 3 * 3 * 3 * 3)
/	Divide	Divide x by y	4/3 gives 1. Notice that division of integers also returns an integer. 4.0/3 or 4/3.0 gives 1.3333333333333333. Notice that division involving float numbers return a float number.
//	Floor Division	Returns the floor of the quotient	4 // 3.0 gives 1.0.
%	Modulo	Returns the remainder of the division	8%3 gives 2. -25.5%2.25 gives 1.5.
<<	Left Shift	Shifts the bits of the number to the left by specified amount. Numbers are represented in the computer's memory using binary digits i.e. 0s and 1s.	2 is represented as 0010 in binary. 2 << 1 gives 4 because its binary representation is 0100.
>>	Right Shift	Shifts the bits of the number to the right by specified amount.	8 is represented as 1000 in binary. 8 >> 3 gives 1 because its binary representation is 0001.
&	Bitwise AND	Bitwise AND of the numbers where each corresponding bit of the two numbers are	5 & 3 gives 1 since 0101 & 0011 gives 0001 (binary representations).

Operator	Name	Explanation	Examples
		multiplied to give a new number	
	Bitwise OR	Bitwise OR of the numbers where each corresponding bit of the two numbers are ORed.	5 3 gives 7 since 0101 0011 gives 0111 (binary representations).
^	Bitwise XOR	Bitwise XOR of the numbers where each corresponding bit of the two numbers are XORed.	5 ^ 3 gives 6 since 0101 % 0011 gives 0011 (binary representations).
~	Bitwise invert	Flip each bit of the binary representation of the number	~5 gives -6 since ~0101 gives 1010 (binary representations, note that negative numbers are indicated using 1's complement, explanation of this is beyond the scope of this book)
<	Less than	Returns whether x is less than y. All comparison operators return True or False.	5 < 3 gives False. Note that comparisons can be chained arbitrarily. For example, 3 < 5 < 7 gives True.
>	Greater than	Returns whether x is greater than y	5 > 3 gives True.
<=	Less than or equal to	Returns whether x is less than or equal to y	x = 3; y = 6; x <= y returns True.
>=	Greater than or equal to	Returns whether x is greater than or equal to y	x = 4; y = 3; x >= y returns True.
==	Equal to	Compare if two objects are equal	x = 2; y = 2; x == y returns True. x = 'str'; y = 'stR'; x == y returns False. x = 'str'; y = 'str'; x == y returns True.
!=	Not equal	Compares if the objects are not	x = 2; y = 3; x != y returns True.

Operator	Name	Explanation	Examples
	to	equal	
Not	Boolean NOT	if x is True, return False. if x is False, return True.	x = True; not y returns False.
And	Boolean AND	x and y returns False if x is False, otherwise is returns evaluation of y.	x = False; y = True; x and y returns False since x is False. In this case, Python will not evaluate y since it knows that the value of the expression will have to be False since x is False. This is called short-circuit evaluation.
Or	Boolean OR	If x is True, return True, else return the evaluation of y.	x = True; y = False; x or y returns True. Short-circuit evaluation applies here as well.

Boolean Operations — and, or, not

These are the Boolean operations, ordered by ascending priority:

Operation	Result	Notes
x or y	if x is false, then y, else x	(1)
x and y	if x is false, then x, else y	(2)
not x	if x is false, then True, else False	(3)

Notes:

1. This is a short-circuit operator, so it only evaluates the second argument if the first one is **False**.
2. This is a short-circuit operator, so it only evaluates the second argument if the first one is **True**.
3. Not has a lower priority than non-Boolean operators, so not a == b is interpreted as not (a == b), and a == not b is a syntax error.

Comparisons

Comparison operations are supported by all objects. They all have the same priority (which is higher than that of the Boolean operations). Comparisons can be chained arbitrarily; for example, $x < y \leq z$ is equivalent to $x < y$ and $y \leq z$, except that y is evaluated only once (but in both cases z is not evaluated at all when $x < y$ is found to be false).

This table summarizes the comparison operations:

Operation	Meaning	Notes
<code><</code>	strictly less than	
<code><=</code>	less than or equal	
<code>></code>	strictly greater than	
<code>>=</code>	greater than or equal	
<code>==</code>	Equal	
<code>!=</code>	not equal	(1)
<code>is</code>	object identity	
<code>is not</code>	negated object identity	

Notes:

`!=` can also be written `< >`, but this is an obsolete usage kept for backwards compatibility only.

New code should always use `!=`.

4.2.3.2 Numeric Types — int, float, long, complex

There are four distinct numeric types:

plain integers,
long integers,
floating point numbers,
and complex numbers.

All built-in numeric types support the following operations. See *The power operator* and later sections for the operators' priorities.

Operation	Result
<code>x + y</code>	sum of x and y
<code>x - y</code>	difference of x and y

Operation	Result
<code>x * y</code>	product of <code>x</code> and <code>y</code>
<code>x / y</code>	quotient of <code>x</code> and <code>y</code>
<code>x // y</code>	(floored) quotient of <code>x</code> and <code>y</code>
<code>x % y</code>	remainder of <code>x / y</code>
<code>-x</code>	<code>x</code> negated
<code>+x</code>	<code>x</code> unchanged
<code>abs(x)</code>	absolute value or magnitude of <code>x</code>
<code>int(x)</code>	<code>x</code> converted to integer
<code>long(x)</code>	<code>x</code> converted to long integer
<code>float(x)</code>	<code>x</code> converted to floating point
<code>complex(re,im)</code>	a complex number with real part <i>re</i> , imaginary part <i>im</i> . <i>im</i> defaults to zero.
<code>c.conjugate()</code>	conjugate of the complex number <code>c</code> . (Identity on real numbers)
<code>divmod(x, y)</code>	the pair <code>(x // y, x % y)</code>
<code>pow(x, y)</code>	<code>x</code> to the power <code>y</code>
<code>x ** y</code>	<code>x</code> to the power <code>y</code>

Notes:

1. For (plain or long) integer division, the result is an integer. The result is always rounded towards minus infinity: $1/2$ is 0, $(-1)/2$ is -1, $1/(-2)$ is -1, and $(-1)/(-2)$ is 0. Note that the result is a long integer if either operand is a long integer, regardless of the numeric value.
2. Conversion from floats using **int()** or **long()** truncates toward zero like the related function, **math.trunc()**. Use the function **math.floor()** to round downward and **math.ceil()** to round upward.
3. *Deprecated since version 2.3:* The floor division operator, the modulo operator, and the **divmod()** function are no longer defined for complex numbers. Instead, convert to a floating point number using the **abs()** function if appropriate.

4. Also referred to as integer division. The resultant value is a whole integer, though the result's type is not necessarily int.
5. float also accepts the strings "nan" and "inf" with an optional prefix "+" or "-" for Not a Number (NaN) and positive or negative infinity.
6. Python defines pow(0, 0) and 0 ** 0 to be 1, as is common for programming languages.

All **numbers.Real** types (**int**, **long**, and **float**) also include the following operations:

Operation	Result
<code>math.trunc(x)</code>	x truncated to Integral
<code>round(x[, n])</code>	x rounded to n digits, rounding half to even. If n is omitted, it defaults to 0.
<code>math.floor(x)</code>	the greatest integral float $\leq x$
<code>math.ceil(x)</code>	the least integral float $\geq x$

4.2.3.3 Bitwise Operations on Integer Types

Bitwise operations only make sense for integers. Negative numbers are treated as their 2's complement value.

The priorities of the binary bitwise operations are all lower than the numeric operations and higher than the comparisons; the unary operation `~` has the same priority as the other unary numeric operations.

This table lists the bitwise operations sorted in ascending priority (operations in the same box have the same priority):

Operation	Result
<code>x y</code>	bitwise <i>or</i> of x and y
<code>x ^ y</code>	bitwise <i>exclusive or</i> of x and y
<code>x & y</code>	bitwise <i>and</i> of x and y
<code>x << n</code>	x shifted left by n bits
<code>x >> n</code>	x shifted right by n bits
<code>~x</code>	the bits of x inverted

Notes:

1. Negative shift counts are illegal and cause a **ValueError** to be raised.
2. A left shift by n bits is equivalent to multiplication by $\text{pow}(2, n)$. A long integer is returned if the result exceeds the range of plain integers.
3. A right shift by n bits is equivalent to division by $\text{pow}(2, n)$.

4.2.3.4 Operator Precedence

In the expression $2 + 3 * 4$, is the addition or multiplication carried out first? Our high school maths tells us that multiplication should be done first before the addition. This means that multiplication has higher precedence than addition.

Similarly, Python has precedence for all of its operators. The following table gives the operator precedence table for Python from the lowest precedence to the highest precedence. This means, that in a given expression, Python will first evaluate the operators lower in the table before the operators listed higher in the table.

The following table is the same as the one in the Python reference manual and is provided for the sake of completeness. I recommend that you glance through this table and refer to it only when required. If you want to avoid ambiguity in your own mathematical expressions, you should use parentheses. For example, $2 + (3 * 4)$ is more clearer than $2+3 * 4$

Operator	Description
Lambda	lambda expression
Or	Boolean OR
And	Boolean AND
not x	Boolean NOT
in, not in	membership tests
is, is not	identity tests
<, <=, >, >=, !=, ==	comparisons
	bitwise OR
^	bitwise XOR
&	bitwise AND
<<, >>	shifts
+, -	addition, subtraction

Operator	Description
*, /, %	multiplication, division, remainder
+x, -x	positive, negative
~x	bitwise NOT
**	exponentiation
x.attribute	attribute reference
x[index]	subscription
x[start:end]	slicing
f(arguments...)	function call
(expressions, ...)	binding or tuple display
[expressions, ...]	list display
{key:datum, ...}	dictionary display
`expressions, ...`	string conversion

That seems like an awful lot of operators, but don't worry, they're very easy to use once you understand them. We've already come across some of these operators, we'll learn the others in later chapters. Note that operators with the *same precedence* are listed in the same row in the above table. For example, + and - have the same precedence.

4.2.4 Expressions

Using Expressions

```
#!/usr/bin/env python
# Filename: expression.py
length = 5 # assignment statement
breadth = 2
area = length * breadth # typical assignment using value of an expression
print 'Area is', area
print 'Perimeter is', 2 * (length + breadth) # notice use of expressions directly
```

Output

```
$ python expression.py
Area is 10
Perimeter is 14
```

4.2.4 Control Flow

Introduction

In all the programs we have written till now, we have a set of steps in an order (formally referred to as an *algorithm*) and Python faithfully executes them in the same order. What if you wanted to change this flow of the program? For example, I want to write a program that says Good Morning or Good Evening depending on the time of the day. To achieve this, we need some way to control the flow of our program.

There are three types of control flow statements in Python - if, for and while.

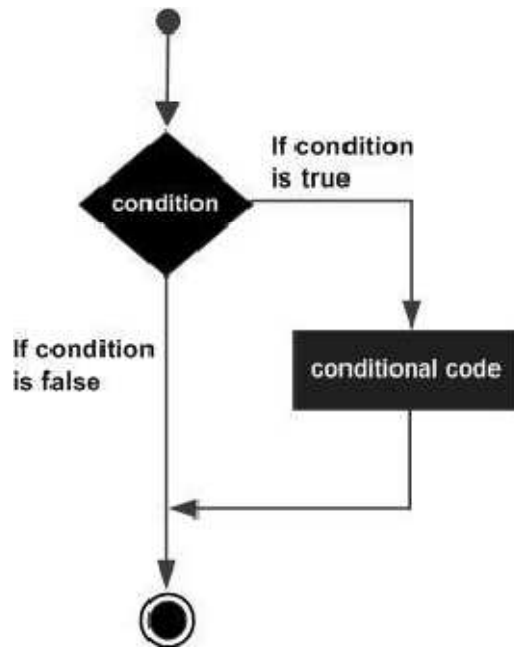
The if statement is used to check if a condition is True and if it is True, we run a corresponding set of steps which are specified in a separate *block*. A block of statements is specified using indentation, that is, using spaces or tabs.

if statement Syntax

if expression:

statement(s)

If the boolean **expression** evaluates to **true**, then the block of statement(s) inside the if statement will be executed. If boolean expression evaluates to **false**, then the first set of code after the end of the if statement(s) will be executed. Python programming language assumes any **non-zero** and **non-null** values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.



```

#!/usr/bin/env python
# Filename: if1.py
age = 22 # mention your age here
if age >= 18: # notice the colon
    print 'You are old enough to vote now!' # use tabs or 4 spaces for indentation
  
```

Output

```

$ python if1.py

You are old enough to vote now!
  
```

if-else statement

An **else** statement can be combined with an **if** statement. An **else** statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a false value.

The *else* statement is an optional statement and there could be at most only one **else** statement following **if** .

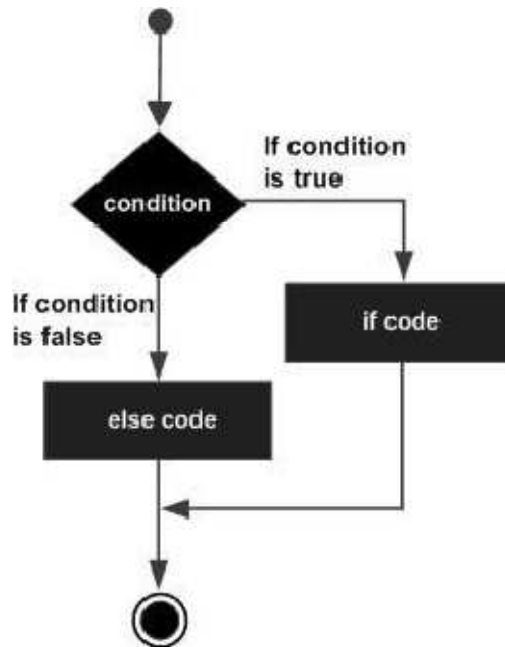
Syntax:

The syntax of the *if...else* statement is:

```

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

Flow Diagram:



```
#!/usr/bin/env python
```

```
# Filename: if2.py
```

```
age = 17 # mention your age here
```

```
if age >= 18: # notice the colon
```

```
    print 'You are old enough to vote now!' # use tabs or 4 spaces for indentation
```

```
else:
```

```
    print 'Sorry, you are not old enough to vote'
```

```
$ python if2.py
```

```
Sorry, you are not old enough to vote
```

The else clause is part of the if statement and it can have a corresponding block of statements that will be executed if the if condition is False.

The else Statement Used with Loops

Python supports to have an **else** statement associated with a loop statements.

- If the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has exhausted iterating the list.

- If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

The following example illustrates the combination of an else statement with a for statement that searches for prime numbers from 10 through 20.

```
#!/usr/bin/python
for num in range(10,20): #to iterate between 10 to 20
    for i in range(2,num): #to iterate on the factors of the number
        if num%i == 0:    #to determine the first factor
            j=num/i        #to calculate the second factor
            print '%d equals %d * %d' % (num,i,j)
            break #to move to the next number, the #first FOR
    else:                # else part of the loop
        print num, 'is a prime number'
```

When the above code is executed, it produces following result:

```
10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
13 is a prime number
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number
```

4.2.5 Loop Statements

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loop inside any another while, for or do..while loop.

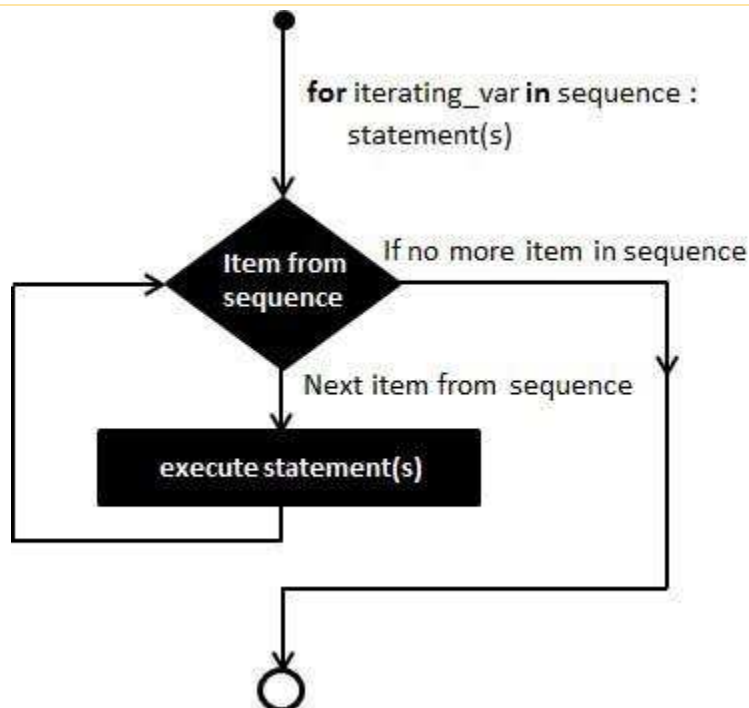
for Loop Statements

The **for** loop in Python has the ability to iterate over the items of any sequence, such as a list or a string.

Syntax:

The syntax of a **for** loop look is as follows:

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



If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable *iterating_var*. Next, the statements block is executed. Each item in the list is assigned to *iterating_var*, and the statements(s) block is executed until the entire sequence is exhausted.

```
>>> # Measure some strings:
```

```
... words = ['cat', 'window', 'defenestrate']
```

```
>>> for w in words:
```

```
...     print w, len(w)
```

```
...
```

```
cat 3
```

```
window 6
```

```
Defenestrate 12
```

If you need to modify the sequence you are iterating over while inside the loop (for example to duplicate selected items), it is recommended that you first make a copy. Iterating over a sequence does not implicitly make a copy. The slice notation makes this especially convenient:

```
>>> for w in words[:]: # Loop over a slice copy of the entire list.
```

```
...     if len(w) > 6:
```

```
...         words.insert(0, w)
```

```
...
```

```
>>> words
```

```
['defenestrate', 'cat', 'window', 'defenestrate']
```

4.2.5.1 nested loops

Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax:

The syntax for a **nested for loop** statement in Python is as follows:

```
for iterating_var in sequence:
```

```
    statements(s)
```

```
statements(s)
```

The syntax for a **nested while loop** statement in Python programming language is as follows:

```
while expression:
```

```
    statement(s)
```

```
statement(s)
```

A final note on loop nesting is that you can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

Example: The following program uses a nested for loop to find the prime numbers from 2 to 100:

```
#!/usr/bin/python
i = 2
while(i < 100):
    j = 2
    while(j <= (i/j)):
        if not(i%j): break
        j = j + 1
    if (j > i/j) : print i, " is prime"
    i = i + 1
print "Good bye!"
```

When the above code is executed, it produces following result:

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
```

```
89 is prime  
97 is prime  
Good bye!
```

4.2.5.2 while loop statement

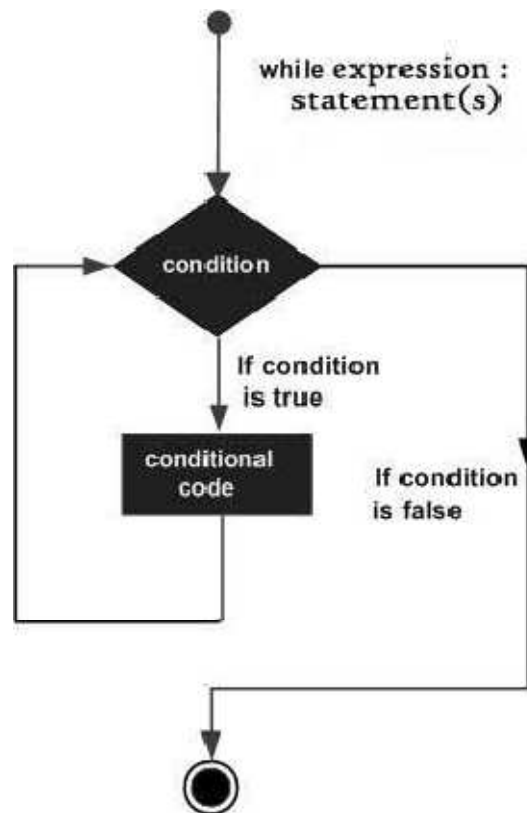
A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax:

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

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

When the condition becomes false, program control passes to the line immediately following the loop.



Example:

```
#!/usr/bin/python
count = 0
while (count < 9):
    print 'The count is:', count
    count = count + 1
print "Good bye!"
```

When the above code is executed, it produces following result:

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

The block here, consisting of the print and increment statements, is executed repeatedly until count is no longer less than 9. With each iteration, the current value of the index count is displayed and then increased by 1.

4.3 The range() Function

If you do need to iterate over a sequence of numbers, the built-in function **range()** comes in handy. It generates lists containing arithmetic progressions:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The given end point is never part of the generated list; `range(10)` generates a list of 10 values, the legal indices for items of a sequence of length 10. It is possible to let the range start at another number, or to specify a different increment (even negative; sometimes this is called the 'step'):

```
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

To iterate over the indices of a sequence, you can combine **range()** and **len()** as follows:

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print i, a[i]
...
0 Mary
1 had
2 a
3 little
4 lamb
```

In most such cases, however, it is convenient to use the **enumerate()** function.

4.3.1 break and continue Statements, and else Clauses on Loops

The **break** statement, like in C, breaks out of the smallest enclosing **for** or **while** loop.

Loop statements may have an **else** clause; it is executed when the loop terminates through exhaustion of the list (with **for**) or when the condition becomes false (with **while**), but not when the loop is terminated by a **break** statement. This is exemplified by the following loop, which searches for prime numbers:

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print n, 'equals', x, '*', n/x
...             break
...     else:
...         # loop fell through without finding a factor
...         print n, 'is a prime number'
```

Output:

2 is a prime number

3 is a prime number

4 equals 2 * 2

5 is a prime number

6 equals 2 * 3

7 is a prime number

8 equals 2 * 4

9 equals 3 * 3

(Yes, this is the correct code. Look closely: the `else` clause belongs to the `for` loop, **not** the `if` statement.)

When used with a loop, the `else` clause has more in common with the `else` clause of a `try` statement than it does that of `if` statements: a `try` statement's `else` clause runs when no exception occurs, and a loop's `else` clause runs when no `break` occurs. For more on the `try` statement and exceptions, see *Handling Exceptions*.

The **`continue`** statement, also borrowed from C, continues with the next iteration of the loop:

```
>>> for num in range(2, 10):
```

```
...     if num % 2 == 0:
```

```
...         print "Found an even number", num
```

```
...         continue
```

```
...     print "Found a number", num
```

Found an even number 2

Found a number 3

Found an even number 4

Found a number 5

Found an even number 6

Found a number 7

Found an even number 8

Found a number 9

4.3.2 Lists

Python knows a number of *compound* data types, used to group together other values. The most versatile is the *list*, which can be written as a list of comma-separated values (items) between square brackets.

Creating List

List can be created using square bracket. List items need not all have the same type.

```
>>> a = ['spam', 'eggs', 100, 1234]
```

```
>>> a
```

```
['spam', 'eggs', 100, 1234]
```

List Functions

<code>list.append(x)</code>	Add an item to the end of the list; equivalent to <code>a[len(a):] = [x]</code> .
<code>list.extend(L)</code>	Extend the list by appending all the items in the given list; equivalent to <code>a[len(a):] = L</code> .
<code>list.insert(i, x)</code>	Insert an item at a given position. The first argument is the index of the element before which to insert, so <code>a.insert(0, x)</code> inserts at the front of the list, and <code>a.insert(len(a), x)</code> is equivalent to <code>a.append(x)</code> .
<code>list.remove(x)</code>	Remove the first item from the list whose value is <code>x</code> . It is an error if there is no such item.
<code>list.pop([i])</code>	Remove the item at the given position in the list, and return it. If no index is specified, <code>a.pop()</code> removes and returns the last item in the list.
<code>list.index(x)</code>	Return the index in the list of the first item whose value is <code>x</code> . It is an error if there is no such item.
<code>list.count(x)</code>	Return the number of times <code>x</code> appears in the list.
<code>list.sort()</code>	Sort the items of the list, in place.
<code>list.reverse()</code>	

Reverse the elements of the list, in place.

An example that uses most of the list methods:

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print a.count(333), a.count(66.25), a.count('x')
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
```

```
[66.25, 333, -1, 333, 1, 1234.5, 333]
```

```
>>> a.index(333)
```

```
1
```

```
>>> a.remove(333)
```

```
>>> a
```

```
[66.25, -1, 333, 1, 1234.5, 333]
```

```
>>> a.reverse()
```

```
>>> a
```

```
[333, 1234.5, 1, 333, -1, 66.25]
```

```
>>> a.sort()
```

```
>>> a
```

```
[-1, 1, 66.25, 333, 333, 1234.5]
```

4.3.3 TUPLE

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The only difference is that tuples can't be changed i.e., tuples are immutable and tuples use parentheses and lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values and optionally you can put these comma-separated values between parentheses also. For example:

```
tup1 = ('physics', 'chemistry', 1997, 2000);
```

```
tup2 = (1, 2, 3, 4, 5 );
```

```
tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing:

```
tup1 = ();
```

To write a tuple containing a single value you

have to include a comma, even though there is only one value:

```
tup1 = (50,);
```

Like string indices, tuple indices start at 0, and tuples can be sliced, concatenated and so on.

Accessing Values in Tuples:

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. Following is a simple example:

```
#!/usr/bin/python
```

```
tup1 = ('physics', 'chemistry', 1997, 2000);
```

```
tup2 = (1, 2, 3, 4, 5, 6, 7 );
```

```
print "tup1[0]: ", tup1[0]
```

```
print "tup2[1:5]: ", tup2[1:5]
```

When the above code is executed, it produces the following result:

```
tup1[0]: physics
tup2[1:5]: [2, 3, 4, 5]
```

Updating Tuples:

Tuples are immutable which means you cannot update them or change values of tuple elements. But we are able to take portions of an existing tuple to create a new tuple as follows. Following is a simple example:

```
#!/usr/bin/python
```

```
tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');
```

```
# Following action is not valid for tuples
# tup1[0] = 100;
```

```
# So let's create a new tuple as follows
```

```
tup3 = tup1 + tup2;
print tup3;
```

When the above code is executed, it produces the following result:

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements:

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement. Following is a simple example:

```
#!/usr/bin/python
```

```
tup = ('physics', 'chemistry', 1997, 2000);
print tup;
del tup;
print "After deleting tup : "
print tup;
```

This will produce following result. Note an exception raised, this is because after **del tup** tuple does not exist any more:

```
('physics', 'chemistry', 1997, 2000)
```

After deleting tup :

Traceback (most recent call last):

File "test.py", line 9, in <module>

print tup;

NameError: name 'tup' is not defined

Basic Tuples Operations:

Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter :

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
['Hi!'] * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Built-in Tuple Functions:

Python includes the following tuple functions:

SN	Function	Description
1	<code>cmp(tuple1, tuple2)</code>	Compares elements of both tuples.
2	<code>len(tuple).</code>	Gives the total length of the tuple
3	<code>max(tuple)</code>	Returns item from the tuple with max value.
4	<code>min(tuple)</code>	Returns item from the tuple with min value.
5	<code>tuple(seq)</code>	Converts a list into tuple.

4.3.4 Sets

Python also includes a data type for *sets*. A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Curly braces or the `set()` function can be used to create sets. Note: to create an empty set you have to use `set()`, not `{}`; the latter creates an empty dictionary, a data structure that we discuss in the next section.

Here is a brief demonstration:

```
>>>
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)           # show that duplicates have been removed
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket      # fast membership testing
True
>>> 'crabgrass' in basket
False
```

```
>>> # Demonstrate set operations on unique letters from two words
```

```
...
```

```
>>> a = set('abracadabra')
```

```
>>> b = set('alacazam')
```

```
>>> a                                # unique letters in a
```

```
{'a', 'r', 'b', 'c', 'd'}
```

```
>>> a - b                            # letters in a but not in b
```

```
{'r', 'd', 'b'}
```

```
>>> a | b                            # letters in either a or b
```

```
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
```

```
>>> a & b                            # letters in both a and b
```

```
{'a', 'c'}
```

```
>>> a ^ b                            # letters in a or b but not both
```

```
{'r', 'd', 'b', 'm', 'z', 'l'}
```

Similarly to list comprehensions, set comprehensions are also supported:

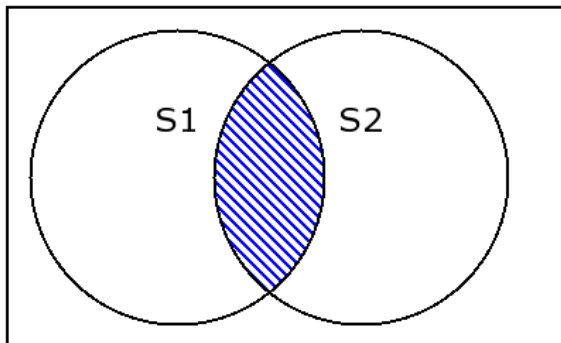
```
>>>
```

```
>>> a = {x for x in 'abracadabra' if x not in 'abc'}
```

```
>>> a
```

```
{'r', 'd'}
```

Figure 4.1 Set Intersection of s1 & s2



Difference, -

The resulting set has elements of the left-hand set with all elements from the right-hand set removed. An element will be in the result if it is in the left-hand set and not in the right-hand set.

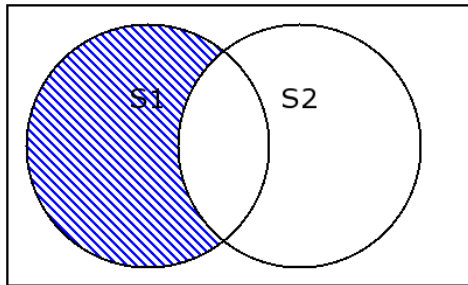
```
>>> fib - prime
```

```
set([8, 1])
```

```
>>> prime - fib
```

```
set([11, 7])
```

Figure 4.2 Set Difference, S1-S2



Symmetric Difference, \wedge

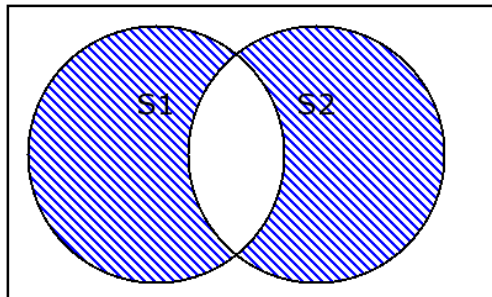
The resulting set has elements which are unique to each set. An element will be in the result set if either it is in the left-hand set and not in the right-hand set or it is in the right-hand set and not in the left-hand set. Whew!

```
>>>
```

```
fib ^ prime
```

```
set([8, 1, 11, 7])
```

Figure 4.3 Set Symmetric Difference, $S2 \wedge S2$



4.3.5 STRINGS

In Python, string is a sequence of Unicode character. Unicode was introduced to include every character in all languages and bring uniformity in encoding.

How to create a string?

Strings can be created by enclosing characters inside a single quote or double quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

all of the following are equivalent

```
my_string = 'Hello'
```

```
print(my_string)
```

```
my_string = "Hello"
```

```
print(my_string)
```

```
my_string = "Hello"
print(my_string)
```

```
# triple quotes string can extend multiple lines
my_string = """Hello, welcome to
    the world of Python"""
print(my_string)
```

Output:

```
Hello
Hello
Hello
Hello, welcome to
    the world of Python
```

4.3.5.1 Built-in functions

Name	Description	Example
capitalize()	Capitalizes first letter of string	<pre>a='computer' b=a.capitalize() print(b)</pre> Output Computer
count(str)	Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given	<pre>a='computer' b=a.count('er') print(b)</pre> Output 1
isalpha()	Returns true if string has at least 1 character and all characters are alphabetic and false otherwise	<pre>a='computer7' b=a.isalpha() print(b)</pre> Output False <pre>a='Computer' b=a.isalpha() print(b)</pre> Output True

isdigit()	Returns true if string contains only digits and false otherwise	a='Computer' b=a.isdigit() print(b) Output False a='125' b=a.isdigit() print(b) Output True
islower()	Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise	a='CoMpuTer' b=a.islower() print(b) Output False a='dept' b=a.islower() print(b) Output True
isspace()	Returns true if string contains only whitespace characters and false otherwise	a=' ' b=a.isspace() print(b) Output True a=' w ' b=a.isspace() print(b) Output False
isupper()	Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise	a='DEPT' b=a.isupper() print(b) Output True a='DePt' b=a.isupper() print(b) Output False
len(str)	Returns the length of the string	a='welcome' b=len(a) print(b)

		Output 7
lower()	Converts all uppercase letters in string to lowercase	a='RoSe' b=a.lower() print(b) Output rose
max(str)	Returns the max alphabetical character from the string str	a='zero' b=max(a) print(b) Output z
min(str)	Returns the min alphabetical character from the string str	a='zero' b=min(a) print(b) Output e
upper()	Converts lowercase letters in string to uppercase	a='OpenSourCe' b=a.upper() print(b) Output OPENSOURCE
join(seq)	Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string	a='zero' c='r,w,q' b=a.join(c) print(b) Output rzero,zerowzero,zeroq

Escape Characters:

Following table is a list of escape or non-printable characters that can be represented with backslash notation.

An escape character gets interpreted; in a singlequoted as well as doublequoted strings.

Backslash notation	Hexadecimal character	Description
\a	0x07	Bell or alert
\b	0x08	Backspace
\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed
\M-\C-x		Meta-Control-x
\n	0x0a	Newline

\nnn		Octal notation, where n is in the range 0-7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Hexadecimal notation, where n is in the range 0-9, a-f, or A-F

4.3.5.2 String Special Operators:

Assume string variable a holds 'Hello' and variable b holds 'Python', then:

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
In	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give 1
r/R	Raw String - Suppresses actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter "r," which precedes the quotation marks. The "r" can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark.	print r'\n' prints \n and print R'\n' prints \n
%	Format - Performs String formatting	See at next section

4.3.5.3 String Formatting Operator:

One of Python's coolest features is the string format operator %. This operator is unique to strings and makes up for the pack of having functions from C's printf() family. Following is a simple example:

```
#!/usr/bin/python
print "My name is %s and weight is %d kg!" % ('Zara', 50)
```

When the above code is executed, it produces the following result:

My name is Zara and weight is 50 kg!

Here is the list of complete set of symbols which can be used along with %:

Format Symbol	Conversion
%c	Character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

Other supported symbols and functionality are listed in the following table:

Symbol	Functionality
*	argument specifies width or precision
-	left justification
+	display the sign
<sp>	leave a blank space before a positive number
#	add the octal leading zero ('0') or hexadecimal leading '0x' or '0X', depending on whether 'x' or 'X' were used.
0	pad from left with zeros (instead of spaces)
%	'%%' leaves you with a single literal '%'
(var)	mapping variable (dictionary arguments)
m.n.	m is the minimum total width and n is the number of digits to display after the decimal point (if appl.)

4.4 Dictionaries

A dictionary in Python is a collection of unordered values which are accessed by key.

A dictionary is mutable and is another container type that can store any number of Python objects, including other container types. Dictionaries consist of pairs (called items) of keys and their corresponding values.

Python dictionaries are also known as associative arrays or hash tables. The general syntax of a dictionary is as follows:


```
dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
```

You can create dictionary in the following way as well:

```
dict1 = { 'abc': 456 };
```

```
dict2 = { 'abc': 123, 98.6: 37 };
```

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

4.4.1 Accessing Values in Dictionary:

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example:

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 10, 'Class': 'First'};
```

```
print "dict['Name']: ", dict['Name'];
```

```
print "dict['Age']: ", dict['Age'];
```

When the above code is executed, it produces the following result:

```
dict['Name']: Zara
```

```
dict['Age']: 10
```

If we attempt to access a data item with a key, which is not part of the dictionary, we get an error as follows:

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
```

```
print "dict['Alice']: ", dict['Alice'];
```

When the above code is executed, it produces the following result:

```
dict['Zara']:
```

```
Traceback (most recent call last):
```

```
File "test.py", line 4, in <module>
```

```
    print "dict['Alice']: ", dict['Alice'];
```

```
KeyError: 'Alice'
```

4.4.2 Updating Dictionary:

You can update a dictionary by adding a new entry or item (i.e., a key-value pair), modifying an existing entry, or deleting an existing entry as shown below in the simple example:

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
```

```
dict['Age'] = 8; # update existing entry
```

```
dict['School'] = "DPS School"; # Add new entry
```

```
print "dict['Age']: ", dict['Age'];
```

```
print "dict['School']: ", dict['School'];
```

When the above code is executed, it produces the following result:

```
dict['Age']: 8
```

```
dict['School']: DPS School
```

4.4.3 Delete Dictionary Elements:

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the **del** statement. Following is a simple example:

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
```

```
del dict['Name']; # remove entry with key 'Name'
```

```
dict.clear();    # remove all entries in dict
```

```
del dict ;      # delete entire dictionary
```

```
print "dict['Age']: ", dict['Age'];
```

```
print "dict['School']: ", dict['School'];
```

This will produce the following result. Note an exception raised, this is because after **del dict** dictionary does not exist any more:

```
dict['Age']:
```

```
Traceback (most recent call last):
```

```
File "test.py", line 8, in <module>
```

```
print "dict['Age']: ", dict['Age'];
```

TypeError: 'type' object is unsubscriptable

Note: del() method is discussed in subsequent section.

4.4.4 Combining two Dictionaries

You can combine two dictionaries by using the update method of the primary dictionary. Note that the update method will merge existing elements if they conflict.

The method **update()** adds dictionary dict2's key-values pairs in to dict. This function does not return anything.

Syntax

Following is the syntax for **update()** method:

```
dict.update(dict2)
```

Parameters

- **dict2** -- This is the dictionary to be added into dict.

Return Value

This method does not return any value.

Example

The following example shows the usage of update() method.

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7}
dict2 = {'Sex': 'female' }
dict.update(dict2)
print "Value : %s" % dict
```

Let us compile and run the above program, this will produce the following result:

```
Value : {'Age': 7, 'Name': 'Zara', 'Sex': 'female'}
```

4.4.5 Built-in Dictionary Functions & Methods:

Python includes the following dictionary functions:

Sl.No.	Function with Description
1	cmp(dict1, dict2)

- Compares elements of both dict.
- len(dict)
2 Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
- str(dict)
3 Produces a printable string representation of a dictionary
- type(variable)
4 Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Python includes following dictionary methods

Sl.No.	Methods with Description
1	dict.clear() Removes all elements of dictionary <i>dict</i>
2	dict.copy() Returns a shallow copy of dictionary <i>dict</i>
3	dict.fromkeys() Create a new dictionary with keys from seq and values set to <i>value</i> .
4	dict.get(key, default=None) For <i>key</i> key, returns value or default if key not in dictionary
5	dict.has_key(key) Returns <i>true</i> if key in dictionary <i>dict</i> , <i>false</i> otherwise
6	dict.items() Returns a list of <i>dict</i> 's (key, value) tuple pairs
7	dict.keys() Returns list of dictionary dict's keys
8	dict.setdefault(key, default=None) Similar to get(), but will set dict[key]=default if <i>key</i> is not already in dict
9	dict.update(dict2) Adds dictionary <i>dict2</i> 's key-values pairs to <i>dict</i>
10	dict.values() Returns list of dictionary <i>dict</i> 's values

4.5 File I/O

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a **file** object.

The *open* Function

Before you can read or write a file, you have to open it using Python's built-in *open()* function. This function creates a **file** object, which would be utilized to call other support methods associated with it.

Opening Files: file **open**(string filename, string mode):

open can be used to open files for reading, writing, and appending. It binds a named file object to a stream that can then be used to read/write data. Possible modes include:

- 'r': Open for reading.
- 'w': Open for writing. Any existing data will be overwritten.
- 'a': Open for writing. New data will be appended to existing data.
- 'b': Use this flag when working with binary files (e.g. 'rb').

Checking Files: Python supports several methods of checking if a file exists and checking its properties:

- bool **os.access**(string path, int mode): returns TRUE if the filename exists and matches the mode query. The mode query can be any of the following constants:
 - **os.F_OK**: test the existence of path
 - **os.R_OK**: tests if path exists and is readable
 - **os.W_OK**: tests if path exists and is writable
 - **os.X_OK**: tests if path exists and is executable

Reading Files: Files can be read by several methods.

- string **read**([int length]): Reads up to a specified number of bytes from the file into a string. It will read until it encounters EOF or the specified length is reached (default is all data).
- string **readline**([int length]): Reads one entire line from a file, or up to length bytes, into a string. Reading stops when length bytes have been read or a newline or EOF is reached. A trailing newline character is kept in the string (but may be absent on the last line of the file).
- list **readlines**([int sizehint]): Reads from a file using *readline()* until EOF and returns a list containing the lines read. If sizehint is present, whole lines totaling approximately sizehint bytes are read.

EOF: end-of-file is reached when *read* or *readline* returns an empty string. while (s != ""):

```
s = f.readline()
do_something
```

Writing to files: Files that have been opened for writing with *open* can be written to by two methods.

- void **write**(string string): Writes the contents of string to the file. Does not append a newline character to the string. Only strings can be written so other datatypes must be converted to strings.
- void **writelines**(list data): Writes a list or array of strings to the file. Newlines will not be added between the elements of the list/array.

Examples:

```
file = open("data/teams.txt", "rb")
```

```

team = "nonempty"
while (team != ""):
    team = file.readline()
    if (team != ""): print team[:-1] #get rid of extra newline character
file.close()

file = open("data/teams.txt", "rb")
team = file.readlines()
file.close()

list = ["Florida", "Clemson", "Duke"]
file = open("data/teams.txt", "wb")
for j in list: file.write(j+"\n")
file.close()

import pickle, fcntl
player = Player("J.J. Redick", "Duke", 4)
file = open("data/players.txt", "a")
fcntl.flock(file.fileno(), fcntl.LOCK_EX)
pickle.dump(player, file)
fcntl.flock(file.fileno(), fcntl.LOCK_UN)
file.close()

```

4.5.1 Functions

Function

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like `print()`, etc. but you can also create your own functions. These functions are called *user-defined functions*.

4.5.1.1 Defining a Function

- Function blocks begin with the keyword **def** followed by the function name and parentheses (()).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

Syntax:

```

def functionname( parameters ):
    "function_docstring"
    function_suite
    return [expression]

```

By default, parameters have a positional behavior and you need to inform them in the same order that they were defined.

Example:

```
def printme( str ):
    "This prints a passed string into this function"
    print str
    return
```

4.5.1.2 Calling a Function

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call `printme()` function:

```
#!/usr/bin/python

# Function definition is here
def printme( str ):
    "This prints a passed string into this function"
    print str;
    return;

# Now you can call printme function
printme("I'm first call to user defined function!");
printme("Again second call to the same function");
```

When the above code is executed, it produces the following result:

```
I'm first call to user defined function!
Again second call to the same function
```

4.5.1.3 Function Arguments:

You can call a function by using the following types of formal arguments:

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Required arguments:

Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

To call the function `printme()`, you definitely need to pass one argument, otherwise it would give a syntax error as follows:

```
#!/usr/bin/python

# Function definition is here
def printme( str ):
    "This prints a passed string into this function"
```

```
print str;  
return;
```

Now you can call printme function

```
printme();
```

When the above code is executed, it produces the following result:

Traceback (most recent call last):

File "test.py", line 11, in <module>

```
printme();
```

TypeError: printme() takes exactly 1 argument (0 given)

Keyword arguments:

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters. You can also make keyword calls to the *printme()* function in the following ways:

```
#!/usr/bin/python
```

Function definition is here

```
def printme( str ):
```

```
    "This prints a passed string into this function"
```

```
    print str;
```

```
    return;
```

Now you can call printme function

```
printme( str = "My string");
```

When the above code is executed, it produces the following result:

```
My string
```

Default arguments:

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument. Following example gives an idea on default arguments, it would print default age if it is not passed:

```
#!/usr/bin/python
```

Function definition is here

```
def printinfo( name, age = 35 ):
```

```
    "This prints a passed info into this function"
```

```
    print "Name: ", name;
```

```
    print "Age ", age;
```

```
    return;
```

Now you can call printinfo function

```
printinfo( age=50, name="miki" );
```



```
printinfo( name="miki" );
```

When the above code is executed, it produces the following result:

```
Name: miki
Age 50
Name: miki
Age 35
```

Variable-length arguments:

You may need to process a function for more arguments than you specified while defining the function. These arguments are called *variable-length* arguments and are not named in the function definition, unlike required and default arguments.

The general syntax for a function with non-keyword variable arguments is this:

```
def functionname([formal_args,] *var_args_tuple ):
    "function_docstring"
    function_suite
    return [expression]
```

An asterisk (*) is placed before the variable name that will hold the values of all nonkeyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call.

```
#!/usr/bin/python

# Function definition is here
def printinfo( arg1, *vartuple ):
    "This prints a variable passed arguments"
    print "Output is: "
    print arg1
    for var in vartuple:
        print var
    return;

# Now you can call printinfo function
printinfo( 10 );
printinfo( 70, 60, 50 );
```

When the above code is executed, it produces the following result:

```
Output is:
10
Output is:
70
60
50
```

4.5.1.4 The *return* Statement:

The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

```
#!/usr/bin/python
```

```

# Function definition is here
def sum( arg1, arg2 ):
    # Add both the parameters and return them."
    total = arg1 + arg2
    print "Inside the function : ", total
    return total;

# Now you can call sum function
total = sum( 10, 20 );
print "Outside the function : ", total

```

When the above code is executed, it produces the following result:

```

Inside the function : 30
Outside the function : 30

```

4.5.2 Exception

What is Exception?

An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions. An exception is a Python object that represents an error.

4.5.2.1 Handling Exceptions

When a Python script raises an exception, it must either handle the exception immediately otherwise it would terminate and come out.

Python uses `try...except` to handle exceptions and `raise` to generate them. Java and C++ use `try...catch` to handle exceptions, and `throw` to generate them.

Syntax:

Here is simple syntax of *try....except...else* blocks:

`try:`

You do your operations here;

.....

`except ExceptionI:`

If there is ExceptionI, then execute this block.

`except ExceptionII:`

If there is ExceptionII, then execute this block.

.....
else:

If there is no exception then execute this block.

Here are few important points about the above-mentioned syntax:

- A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
- You can also provide a generic except clause, which handles any exception.
- After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.
- The else-block is a good place for code that does not need the try: block's protection.

Example:

Here is simple example, which opens a file and writes the content in the file and comes out gracefully because there is no problem at all:

```
#!/usr/bin/python
try:
    fh = open("testfile", "w")
    fh.write("This is my test file for exception handling!!")
except IOError:
    print "Error: can't find file or read data"
else:
    print "Written content in the file successfully"
    fh.close()
```

This will produce the following result:

Written content in the file successfully

If tries to open a file where you do not have permission to write in the file, so it raises an exception:

Error: can't find file or read data

The *except* clause with no exceptions:

You can also use the except statement with no exceptions defined as follows:

try:

You do your operations here;

```
.....
except:
    If there is any exception, then execute this block.
```

```
.....
else:
    If there is no exception then execute this block.
```

This kind of a **try-except** statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.

Eg

```
#!/usr/bin/python
try:
    fh = open("testfile", "w")
    fh.write("This is my test file for exception handling!!")
except :
    print "Error: can't find file or read data"
else:
    print "Written content in the file successfully"
    fh.close()
```

This will produce the following result:

Written content in the file successfully

If any exception:

Error: can't find file or read data

The *except* clause with multiple exceptions:

You can also use the same *except* statement to handle multiple exceptions as follows:

```
try:
    You do your operations here;
    .....
except(Exception1[, Exception2[,...ExceptionN]]):
    If there is any exception from the given exception list,
```

then execute this block.

.....

else:

If there is no exception then execute this block.

```
#!/usr/bin/python
```

try:

```
fh = open("testfile", "w")
```

```
fh.write("This is my test file for exception handling!!")
```

except IOError,Network Error:

```
print "Error: can't find file or read data"
```

else:

```
print "Written content in the file successfully"
```

```
fh.close()
```

This will produce the following result:

Written content in the file successfully

If IOError,Networkerror exception:

Error: can't find file or read data

The try-finally clause:

You can use a **finally:** block along with a **try:** block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not. The syntax of the try-finally statement is this:

try:

You do your operations here;

.....

Due to any exception, this may be skipped.

finally:

This would always be executed.

.....

Note that you can provide except clause(s), or a finally clause, but not both. You can not use *else* clause as well along with a finally clause.

Example:

```
#!/usr/bin/python
```

```
try:
```

```
    fh = open("testfile", "w")
```

```
    fh.write("This is my test file for exception handling!!")
```

```
finally:
```

```
    print "Error: can't find file or read data"
```

If you do not have permission to open the file in writing mode, then this will produce the following result:

```
Error: can't find file or read data
```

Same example can be written more cleanly as follows:

```
#!/usr/bin/python
```

```
try:
```

```
    fh = open("testfile", "w")
```

```
    try:
```

```
        fh.write("This is my test file for exception handling!!")
```

```
    finally:
```

```
        print "Going to close the file"
```

```
        fh.close()
```

```
except IOError:
```

```
    print "Error: can't find file or read data"
```

When an exception is thrown in the *try* block, the execution immediately passes to the *finally* block. After all the statements in the finally block are executed, the exception is raised again and is handled in the *except* statements if present in the next higher layer of the *try-except* statement.

4.5.2.2 BUILT IN EXCEPTION

Exceptions are everywhere in Python. Virtually every module in the standard Python library uses them, and Python itself will raise them in a lot of different circumstances.

- Accessing a non-existent dictionary key will raise a **KeyError exception**.
- Searching a list for a non-existent value will raise a **ValueError exception**.
- Calling a non-existent method will raise an **AttributeError exception**.

- Referencing a non-existent variable will raise a **NameError exception**.
- Mixing data types without coercion will raise a **TypeError exception**.

In each of these cases, you were simply playing around in the Python IDE: an error occurred, the exception was printed (depending on your IDE, perhaps in an intentionally jarring shade of red), and that was that. This is called an *unhandled* exception.

4.5.2.3 User-Defined Exceptions:

Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.

```
class Networkerror(RuntimeError):
```

```
    def __init__(self, arg):
        self.args = arg
```

So once you defined above class, you can raise your exception as follows:

```
try:
```

```
    raise Networkerror("Bad hostname")
```

```
except Networkerror,e:
```

```
    print e.args
```

Raising an exceptions:

You can raise exceptions in several ways by using the raise statement. The general syntax for the **raise** statement.

Syntax:

```
raise [Exception [, args [, traceback]]]
```

Here, *Exception* is the type of exception (for example, NameError) and *argument* is a value for the exception argument. The argument is optional; if not supplied, the exception argument is None.

The final argument, traceback, is also optional (and rarely used in practice), and if present, is the traceback object used for the exception.

Example:

An exception can be a string, a class or an object. Most of the exceptions that the Python core raises are classes, with an argument that is an instance of the class.

```
def functionName( level ):
```

```
    if level < 1:
```

```
raise "Invalid level!", level
```

FileDict – a Persistent Dictionary in Python

The dictionary resides in memory, and so has three main “faults”:

1. It only lasts as long as your program does.
2. It occupies memory that might be useful for other, more commonly accessed, data.
3. It is limited to how much memory your machine has.

The first can be solved by pickling and unpickling the dictionary, but will not survive an unexpected shutdown (even putting the pickling in a try-finally block won't protect it against all errors).

FileDict

```
Variablename=filedict.filedict("filename")
```

```
D=filedict.filedict("example.txt")
```

Here d is a persistent variable.

Uses

FileDict can be used for many purposes, including:

- Saving important data in a convenient manner
- Managing large amounts of data in dictionary form, without the mess of implementing paging or other complex solutions
- Communication between processes (split supports multiple connections and implements ACID)

Limitations

- All data (keys and values) must be pickle-able
- Keys must be hashable (*perhaps this should be removed by hashing the pickled key*)
- Keys and values are stored as a copy, so changing them after assignment will not update the dictionary.

Review Questions

Part-A (2 marks)

1. What is Python?
2. What is an Unicode string?
3. What is an immutable string?
4. What is a Python object?
5. What is a raw string?
6. What is the use of range() function?
7. What is a Tuple?
8. What is a Set?

9. What is a Dictionary?
10. What is meant by default argument?
11. What is Exception?

Part-B (3 marks)

1. How will you use Python in Windows Command prompt?
2. How will you use Python in Linux?
3. List the four types of numbers in Python.
4. List the identifier naming rules in Python.
5. Describe the bitwise operation on integers?
6. Write the syntax of if..else in Python.
7. Write the syntax of for statement in Python.
8. What is List in Python? How will you create a list?
9. List the method used to read from files.
10. Write the syntax of defining function in Python.
11. What is Built-in Exception? Give two examples.
12. What is the significance of FileDict? Explain.

Part-C (5 marks)

1. List and explain the features Python.
2. Explain the different methods of quoting strings in Python with examples.
3. Tabulate the operators and their usage.
4. Explain the use of else in Python loop statements with example.
5. Write a Python program using nested for loop to find the prime numbers from 2 to 100.
6. Enumerate and discuss the functions of List.
7. Discuss the operation on Tuple with examples.
8. Discuss the operation on Set with diagrams and examples.
9. List and explain the String built-in functions.
10. Explain the procedure of creating and updating a dictionary.
11. Discuss the operation on Dictionary with examples.
12. List and discuss different types of function arguments in Python.
13. List and explain the Exception handling statements in Python.

UNIT – V

Open Source Tools and Technologies

Objectives

- 1. Practice how to configure and use apache web server.**
- 2. Learn about open source tools and processors.**
- 3. Understand E-Governance and Government policy towards open source.**

5.1 Web Server

5.1.1 Apache web server

Apache is the world's most popular Web server (HTTP server) designed for Unix environment. The Apache Web server has been ported to Windows and other network operating systems. The name "Apache" derives from the word "patchy" that the Apache developers used to describe earlier versions of their software.

The Apache Web server is widely used because

1. It is free to download and install.
2. It is open source: the source code is visible to anyone and everyone, which basically enables anyone to adjust the code, optimize it, and fix errors and security holes. People can add new features and write new modules.
3. It suits all needs: Apache can be used for small websites of one or two pages, or huge websites of hundreds and thousands of pages. It can serve both static and dynamic content.⁴
4. It provides a full range of Web server features, including CGI, SSL, and virtual domains.
5. It supports plug-in modules for extensibility.

5.1.2 Configuring and using apache web server

5.1.2.2 Install Apache on Linux

1, Download Apache

Download Apache from the Apache HTTP Server download site. Download the source files appropriate to your system.

2, Extract the Apache Files

After downloading the files, uncompress them

```
gunzip -d httpd_2_0_NN.tar.gz
tar xvf httpd_2_0_NN.tar
```

This creates a new directory under the current directory with the source files.

3, Configuring Your Server for Apache

The easiest way is to accept all the defaults and just type

```
./configure
```

The most important option is the `prefix=PREFIX` option. This specifies the directory where the Apache files will be installed. We can also set specific environment variables and modules.

4, Build Apache

Build the installation using

```
make
make install
```

5, Customize Apache

Editing the `httpd.conf` file located in the `PREFIX/conf` directory.

6, Test Apache Server

Open a web browser and type `http://localhost/` in the address box. The message "Seeing this instead of the website you expected?" shows that server is installed correctly.

5.1.3 Install Apache on Windows

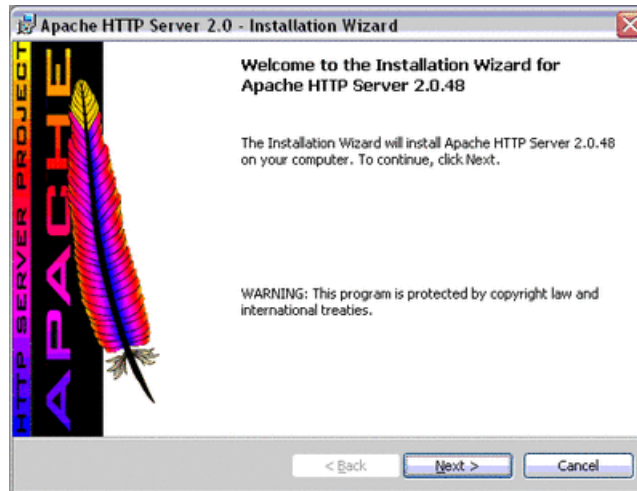
1, Download Apache

Download Apache from the Apache HTTP Server download site. Download the binary file for Win32.

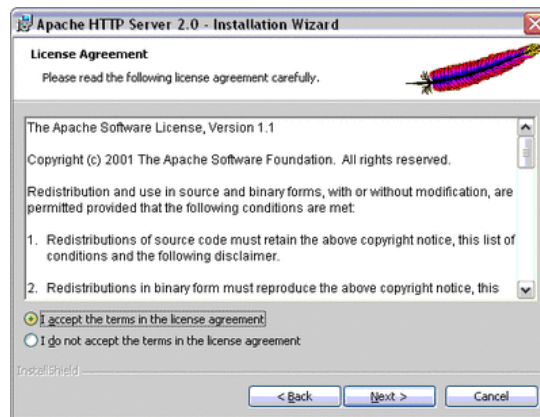
2, Extract the Apache Files

Run the downloaded binary, to start the self-installation.

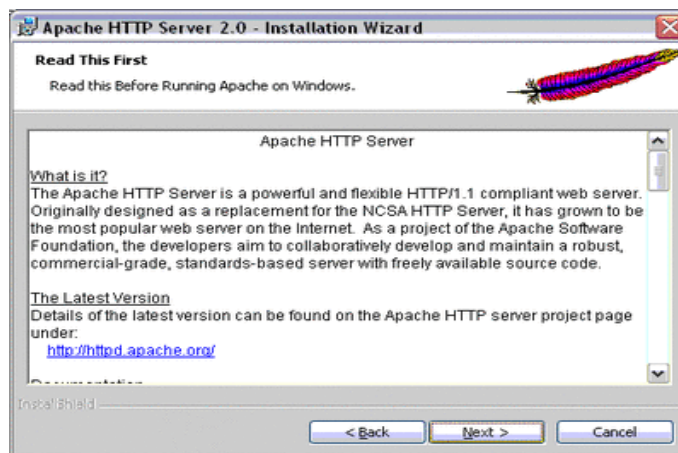
1, Double Click the Apache Executable and Get the Welcome Screen and click NEXT



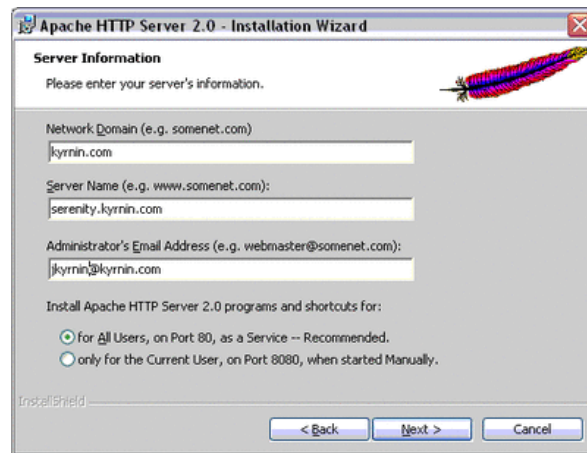
2. Read the License and click NEXT.



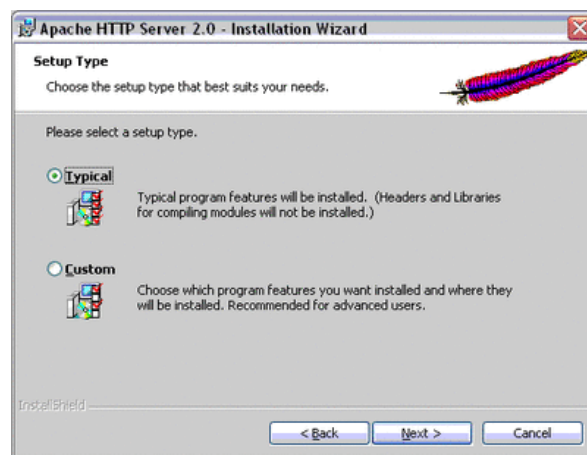
3. Read the README File and click NEXT



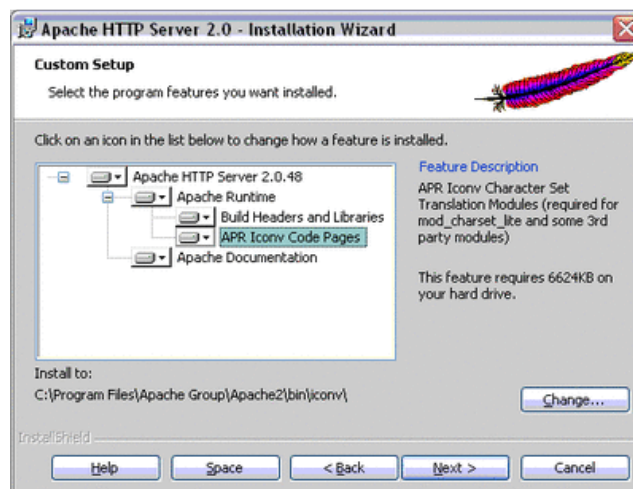
4. Provide Server Information and click next



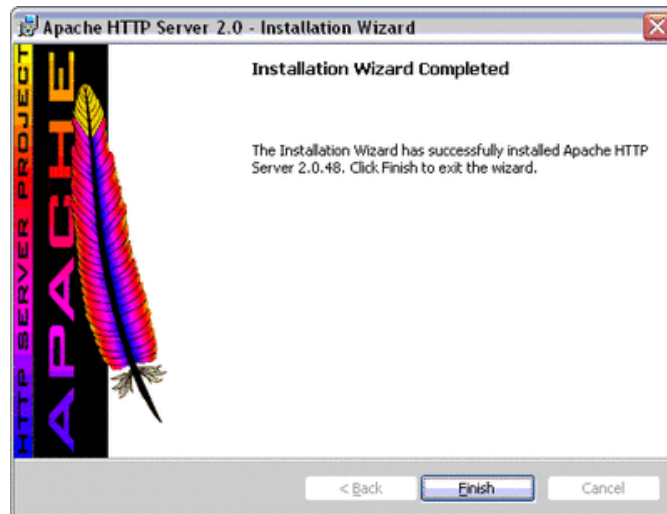
5. Choose Typical Setup and click NEXT



6. Review Custom Options and click Next.



7. Click Finish.



3, Customize Apache

Edit the httpd.conf file. This file is located in the
C:/Program Files/ApacheGroup/Apache2/conf directory.

4, Test Apache Server

Open a web browser and type `http://localhost/` in the address box. The message "Seeing this instead of the website you expected?" shows that server is installed correctly.

5.2 Open Source Software tools

5.2.1 Browsers

A web browser is a software application for retrieving, presenting and traversing information resources on the World Wide Web. An *information resource* is identified by a Uniform Resource Identifier (URI) and may be a web page, image, video or other piece of content. Hyperlinks present in resources enable users easily to navigate their browsers to related resources.

The major open source web browsers are Google Chrome, Mozilla Firefox, Opera, and Safari.

5.2.1.1 Google Chrome

It is a freeware web browser developed by Google and it uses the WebKit layout engine. It was released as a beta version for Microsoft Windows on September 2, 2008, and as a stable public release on December 11, 2008.

Features

- 1, Speed
- 2, Simplicity
- 3, Security
- 4, Auto Updates
- 5, Safe browsing
- 6, Privacy
- 7, Signing in

5.2.1.2 Mozilla firefox

Mozilla Firefox is a free and open source web browser developed for Microsoft Windows, OS X, and Linux, coordinated by Mozilla Corporation and Mozilla Foundation. Firefox uses the Gecko layout engine to render web pages, which implements current and anticipated web standards.

Features

- 1, Extended Tabbed Browsing
- 2, Restoring sessions
- 3, Spell check
- 4, Suggests during search
- 5, Live Bookmark
- 6, Pop-Up Blocker
- 7, Friendly User Interface
- 8, Automatic updates
- 9, Download Manager
- 10, Privacy

5.2.1.3 Opera

Opera is a web browser and Internet suite developed by Opera Software. The browser handles common Internet-related tasks such as displaying web sites, sending and receiving e-mail messages, managing contacts, chatting on IRC, downloading files via Bit Torrent and reading web feeds.

Features

1. **Cached image**
2. **Go to URL**
3. **Create search**
4. **Search in address bar**
5. **Create buttons**
6. **Stop executing script**
7. **Fit to width**
8. **Click to select**
9. **Site preferences**

5.2.1.4 Safari

Safari is a web browser developed by Apple Inc. and included with the Mac OS X and iOS operating systems. First released as a public beta on January 7, 2003 on the company's OS X operating system, it became Apple's default browser beginning with Mac OS X v10.3 "Panther". Safari is also the native browser for iOS. A version of Safari for the Microsoft Windows operating system was first released on June 11, 2007, and supported Windows XP, Windows Vista, and Windows 7.

Features

1. Ability to save webpage clips for viewing on the Apple Dashboard (Mac OS X only)
2. A resizable web-search box in the toolbar
3. Bookmark integration with Address Book
4. Bookmark management
5. Built-in password management via Keychain (Mac OS X only)
6. History and bookmark search
7. Mail integration (Mac OS X only)
8. Pop-up ad blocking
9. Private browsing
10. Spell checking
11. Support for CSS animation
12. Support for HTML5
13. Support for Transport Layer Security protocol (version unknown)
14. Tabbed browsing
15. Text search

5.2.2 Processors

5.2.2.1 Open SPARC

OpenSPARC was an open-source hardware project started in December 2005. The initial contribution to the project was Sun Microsystems' register-transfer level (RTL) Verilog code for a full 64-bit, 32-thread microprocessor, the UltraSPARC T1 processor. On March 21, 2006, Sun released the source code to the T1 IP core under the GNU General Public License.

The full OpenSPARC T1 system consists of 8 cores, each one capable to execute 4 threads concurrently, for a total of 32 threads. Each core executes instruction in order and its logic is split among 6 pipeline stages.

On December 11, 2007, Sun also made the UltraSPARC T2 processor's RTL available via the OpenSPARC project. OpenSPARC T2 is 8 cores, 16 pipelines with 64 threads.

5.2.2.2 OpenRISC

The first architectural description is for the OpenRISC 1000, describing a family of 32 and 64-bit processors with optional floating point and vector processing support.

A team from OpenCores provided the first implementation, the OpenRISC 1200, written in the Verilog hardware description language. The hardware design was released under the GNU Lesser General Public License (LGPL), while the models and firmware were released under the GNU General Public License (GPL). A reference SoC implementation based on the OpenRISC 1200 was developed, known as ORPSoC (the OpenRISC Reference Platform System-on-Chip). A number of groups demonstrated ORPSoC and other OR1200 based designs running on FPGA.

5.2.2.3 Amber

The Amber processor core is an open-source ARM-compatible 32-bit RISC processor. The Amber core is fully compatible with the ARMv2 instruction set and is therefore supported by the GNU tool chain. The Amber project provides a complete embedded FPGA system incorporating the Amber core and a number of peripherals, including UARTs, timers and an Ethernet MAC.

There are two versions of the core provided in the Amber project.

- The Amber 23 has a 3-stage pipeline, a unified instruction and data cache, a Wishbone interface, and is capable of 0.75 DMIPS per MHz. The Amber 23 core is a very small 32-bit core that provides good performance
- The Amber 25 has a 5-stage pipeline, separate data and instruction caches, a Wishbone interface, and is capable of 1.0 DMIPS per MHz. Both cores implement exactly the same ISA and are 100% software compatible. The Amber 25 core provides 30 to 40% better performance.

Both cores have been verified by booting a Linux 2.4 kernel. The cores were developed in Verilog 2001 and are optimized for FPGA synthesis.

5.2.3 Model driven Architecture tools

Model Driven Architecture (MDA) is a software design approach for the development of software systems. It provides a set of guidelines for the structuring of specifications, which are

expressed as models. Model-driven architecture is a kind of domain engineering, and supports model-driven engineering of software systems. It was launched by the Object Management Group (OMG) in 2001.

An MDA tool is a tool used to develop, interpret, compare, align, measure, verify, transform, etc. models or meta models. In any MDA approach we have essentially two kinds of models: *initial models* are created manually by human agents while *derived models* are created automatically by programs. For example an analyst may create a UML initial model from its observation of some loose business situation while a Java model may be automatically derived from this UML model by a Model transformation operation.

Types of MDA tool

- **Creation Tool**
A tool used to elicit initial models and/or edit derived models.
- **Analysis Tool**
A tool used to check models for completeness, inconsistencies, or error and warning conditions. Also used to calculate metrics for the model.
- **Transformation Tool**
A tool used to transform models into other models or into code and documentation.
- **Composition Tool**
A tool used to compose.
- **Test Tool**
A tool used to "test" models as described in Model-based testing.
- **Simulation Tool**
A tool used to simulate the execution of a system represented by a given model.
- **Metadata Management Tool**
A tool intended to handle the general relations between different models, including the metadata on each model and the mutual relations between these models.
- **Reverse Engineering Tool**
A tool intended to transform particular legacy or information artifact portfolios into full-fledged models.

One of the characteristics of MDA tools is that they mainly take models as input and generate models as output.

5.3 Case Study

5.3.1 Government policy towards open source

Prior to 2001, there was almost no activity in policy related to open-source, which could be the result of a lack of maturity in open-source software development up until this point and/or difficulty in finding documentation of older open-source policies online. The first year in which we see a significant increase in open-source policies is 2002, followed by a sharp jump in 2003.

Between 2006 and 2007, we see a second boost in open-source policies.

1. The Government will actively and fairly consider open source solutions alongside proprietary ones in making procurement decisions.
2. Procurement decisions will be made on the basis on the best value for money solution to the business requirement, taking account of total lifetime cost of ownership of the solution, including exit and transition costs, after ensuring that solutions fulfill minimum and essential capability, security, scalability, transferability, support and manageability requirements. Where a 'perpetual license' has previously been purchased from a proprietary vendor (and therefore often giving the appearance of a zero cost to a project), a shadow license cost shall be applied to ensure a fair comparison of total cost of ownership. The shadow license cost will be equivalent to the published list price of the product (no discounts can be factored in), or the price the public sector pays overall on a 'crown' deal.
3. The Government will expect those putting forward IT solutions to develop where necessary a suitable mix of open source and proprietary products to ensure that the best possible overall Solution can be considered. Vendors will be required to provide evidence of this during a procurement exercise. Where no evidence exists in a bid that full consideration has been given to opensourceproducts, the bid will be considered non compliant and is likely to be removed from the vender process.
4. Where there is no significant overall cost difference between open and non-open source products, open source will be selected on the basis of its additional inherent flexibility.

Existing Government support for open source

1, Asia / Singapore

Singapore Government offered Tax reductions and financial grants to fund Linux related projects.

2, Europe / Germany

German Government is offering discount to IBM machines with pre installed Suse Linux distributed by a German company.

3, Europe / France

Government related institutions are strictly instructed to use open source software.

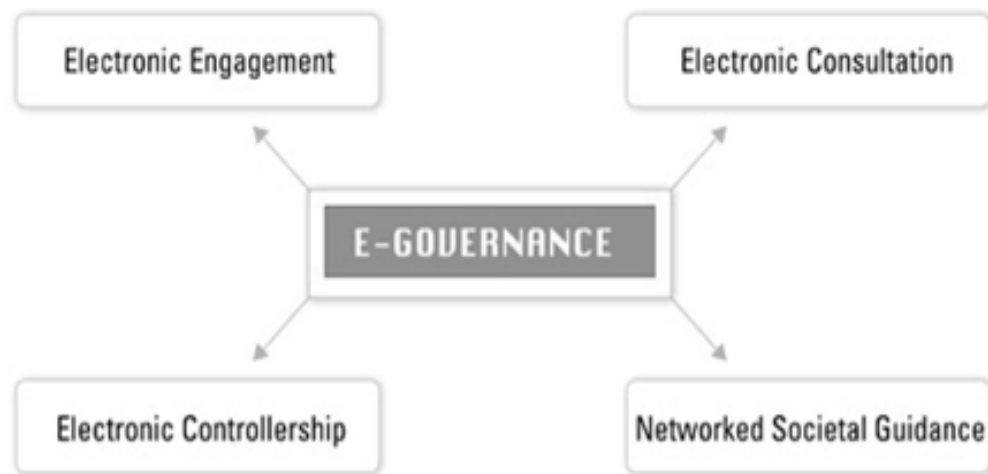
4, Latin America / Argentina

Legislative proposals mandating preference for open source software in all Government policies.

5.3.2 E-Governance

Governance is the societal synthesis of politics, policies, and programs. E-governance is the application of ICT to the system of governance to ensure a wider participation and deeper involvement of citizens, institutions, NGOs as well as private firms in the decision making process. The characteristics of e-governance include

- Electronic Engagement
- Electronic Consultation
- Electronic Controllership
- Networked Societal Guidance



Benefits of e-governance

1. Improved & Enhanced delivery of Government Services
2. Empowerment of citizens through greater access to government information and ability to interact and participate
3. Enhanced Transparency & Increased Accountability of the Government
4. Increasing the internal efficiency and revenue generation by the government
5. Improving the relationship between the government and the citizens

Forms of Interaction

- G2G: Government to Government interaction involves sharing of data and conduct of electronic information exchange amongst various government departments and other entities. This exchange could be both intra and inter agency at the National level as well as exchanges among the national, provincial and local levels.
- G2C: Government to Citizen interaction where electronic dissemination of information and electronic delivery of services takes place, fulfilling the primary objective of e-government. Initiatives in this form of interaction attempt to make transactions such as obtaining certificates, renewing licenses, paying taxes/bills and applying for government

schemes less time consuming and convenient. Also included is the key component of citizen participation in the processes and policy formulation by the government.

- G2B: Government to Business interaction involving improved and efficient procurement of goods and services by the government from the commercial business entities. It also includes sale of government goods to the public and has the potential for reducing costs through improved procurement practices and increased competition. Further, this type of interaction involves the transaction and exchange between the government and the businesses regarding licenses, taxation and policies issued for various sectors.
- G2E : Government to Employee interaction covering employment opportunities, work guidelines, rules & regulations, benefits and pay structures for the government employees, employee welfare schemes, work rules and regulations, government housing etc.

5.2.3 Wikipedia as a open source project

Open-source software (OSS) is computer software with its source code made available with a license in which the copyright holder provides the rights to study, change, and distribute the software to anyone and for any purpose. Open-source software may be developed in a collaborative public manner. The online encyclopedia Wikipedia is often referred to as an "open-source" project because it is written, edited and policed by a global group of volunteers.

Review Questions

Part A

1. What is meant by E-Governance?
2. What is Apache web server?
3. List any three types of MDA tools.
4. List any two open source processors.
5. List any two open source browsers.
6. List any two features of opera.
7. Mention any two features of SPARC.

Part B

1. What are the benefits of E-Governance?
2. List the reasons for the popularity of Apache server.
3. List the features of safari web browser.
4. Write short notes on open source processor amber.
5. Write short notes on open source browser Mozilla firefox.

Part C

1. Explain the Model Driven Architecture in detail.
 2. Explain the different forms of interaction in E-Governance.
 3. Explain the characteristics any two open browsers.
 4. Write short notes on open source processors.
 5. How will you install and configure apache in windows?
 6. How will you install and configure apache in Linux?
-