ALAGAPPA UNIVERSITY

(Accredited with ‘A+’ Grade by NAAC (with CGPA: 3.64) in the Third Cycle and Graded as category - I University by MHRD-UGC)
(A State University Established by the Government of Tamilnadu)

KARAIKUDI – 630 003

DIRECTORATE OF DISTANCE EDUCATION

B.Sc. (INFORMATION TECHNOLOGY)

Second Year – Fourth Semester

12943–Open Source Software

ALAGAPPA UNIVERSITY

(Accredited with ‘A+’ Grade by NAAC (with CGPA: 3.64) in the Third Cycle and Graded
as category - I University by MHRD-UGC)
(A State University Established by the Government of Tamilnadu)

KARAIKUDI – 630 003

DIRECTORATE OF DISTANCE EDUCATION

B.Sc. (INFORMATION TECHNOLOGY)

Second Year – Fourth Semester

12943–Open Source Software

Author:

Dr. K. SHANKAR
Assistant Professor
Department of Computer Science and Information Technology
Kalasalingam Academy of Research and Education
Anand Nagar, Krishnankoil. 626126.

SYLLABI-BOOK MAPPING TABLE

Open Source Software

Syllabi	Mapping in Book
BLOCK1: INTRODUCTION **Unit-1:Introduction to Open Sources:** Need of Open Sources-Advantages of Open Sources-Applications of Open sources **Unit-2:Open Source Operating System:** LINUX: Introduction-General Overview-Kernel Mode and user mode-Process **Unit-3:Advanced Concepts:** Scheduling-Personalities-Cloning-Signals-Development with Linux	**Unit-1:** Introduction **(Pages 1-7)** **Unit-2:** Open Source Operating System **(Pages 8-15)** **Unit-3:** Advanced Concepts **(Pages 16-26)**
BLOCK 2: OPEN SOURCE DATABASE **Unit-4:MySQL:** Introduction -Setting up Account Starting, terminating and writing your Own SQL programs **Unit-5:Record selection Technology**-Working with strings-Date and Time **Unit-6:Sorting query Results**-Generating Summary-Working with Metadata- Using Sequences- My SQL and Web	**Unit-4:** MySQL **(Pages 27-35)** **Unit-5:** Record selection Technology **(Pages 36-46)** **Unit-6:** Sorting query Results **(Pages 47-55)**
BLOCK 3: OPEN SOURCE PROGRAMMING LANGUAGE-PHP **Unit-7:PHP:** Introduction- Programming in Web Environment-Variables-Constants-Data types-Operators- Statements **Unit-8:Functions:** Arrays-OOP- String Manipulation and Regular Expression-File Handling and Data Storage **Unit-9: PHP and MySQL database**: PHP and LDAP- PHP Connectivity- Sending and Receiving E mails- Debugging and Error Handling-Security-Templates	**Unit-7:** PHP **(Pages 56-71)** **Unit-8:** Functions **(Pages 72-87)** **Unit-9:** PHP and MySQL database **(Pages 88-105)**
BLOCK 4: OPEN SOURCE PROGRAMMING LANGUAGE PYTHON **Unit-10:** Syntax and Style-Python Objects-Numbers-Sequences-Strings-Lists and Tuples-Dictionaries-Conditionals and Loops **Unit-11:** Files-Input and Output-Errors and Exceptions-Functions-Modules-Classes and OOP-Execution Environment	**Unit-10:** Syntax and Style **(Pages 106-129)** **Unit-11:** Files **(Pages 130-164)**
BLOCK 5: OPEN SOURCE PROGRAMMING LANGUAGE-PERL **Unit-12: Perl Backgrounder**- Perl Overview-Perl Parsing rules-Variables and Data-Statements **Unit-13: Control Structures-**Subroutines **Unit-14: Packages and Modules**- Working with Files- Data Manipulation	**Unit-12:** Perl Backgrounder **(Pages 165-176)** **Unit-13:** Control Structures **(Pages 177-186)** **Unit-14:** Packages and Modules **(Pages 187-199)**

CONTENTS

BLOCK I
INTRODUCTION

NOTES

UNIT 1
INTRODUCTION

Structure

1.0 INTRODUCTION

This unit explains the open source software concepts and basics of the system with which we work .The applications and need of this software and how it is freely available is discussed.

1.1 OBJECTIVE

This unit briefs the open source software needs and makes the user to understand and learn the following concepts

- Need of open sources
- Applications
- Advantages

1.2 SOFTWARE TERMINOLOGIES

Some of the common terms used in the field of software.

1.2.1 Public Domain Software

Public domain software refers to any program that is not copy righted. This software is free and can be used without restrictions, that is, the user can copy, distribute, and even modify the software without obtaining permission from the software developer.

1.2.2 Freeware

The term freeware is commonly used for copyrighted software given away free by its author. It is available for free but the author retains the copyright, which means that a user does not have the right to modify anything in the software that is not explicitly allowed by the developer. Thus, freeware software permits re-distribution but not modification.

1.2.3 Shareware

Shareware is the software which comes with permission for people to redistribute copies for a limited period. Anyone who continues to use a copy is required to pay a license fee. Therefore, a free use of the software is usually limited to a period. It is distributed without payment ahead of time.

1.2.4 Firmware

Firmware is a combination of software, permanently stored in the memory. As the name suggests, firmware is a program or data that has been written onto read only memory (ROM).

1.2.5 Proprietary Software

Proprietary describes a technology or product that is owned exclusively by a single company that carefully guards knowledge about the technology or the product’s internal working. Proprietary software is also called as Closed Source Software. Its use, redistribution or modification is prohibited or is restricted so much that the user effectively cannot use it freely.

1.2.6 Open Source Software

It is created by generous programmers and released into the public domain for public use. The underlying programming code is available to the users so that they may read it, make changes to it, and build new versions of the software incorporating their changes for software. Usually this software is distributed under an open source license-GPL.

Open source doesn't just mean access to the source code. The distribution terms of open-source software must obey with the following criteria:

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale

2. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

3. Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of the Author's Source Code

The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination against Persons or Groups

The license must not discriminate against any person or group of persons.

6. No Discrimination against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

10. License Must Be Technology-Neutral

No provision of the license may be predicated on any individual technology or style of interface.

1.3 NEED OF OPEN SOURCES

The name “FOSS” is a recursive acronym for “Free Open Source Software”. Free software is a matter of the users’ freedom to run, copy, distribute, study, change and improve the software. More precisely, it refers to four kinds of freedom, for the users of the software:

1. The freedom to run the program, for any purpose

2. The freedom to study how the program works, and change it to make it do what you wish.

3. The freedom to redistribute copies so you can help your neighbour.

4. The freedom to improve the program, and release your improvements (and modified versions in general) to the public, so that the whole community benefits.

Free software is software that gives you the user the freedom to share study and modify it. We call this free software because the user is free. To use free software is to make a political and ethical choice asserting the right to learn, and share what we learn with others. Free software has become the foundation of a learning society where we share our knowledge in a way that others can build upon and enjoy.

Currently, many people use proprietary software that denies users these freedoms and benefits. If we make a copy and give it to a friend, if we try to figure out how the program works, if we put a copy on more than one of our own computers in our own home, we could be caught and fined or put in jail. That’s what’s in the fine print of the license agreement you accept when using proprietary software.

The corporations behind proprietary software will often spy on your activities and restrict you from sharing with others. And because our computers control much of our personal information and daily activities, proprietary software represents an unacceptable danger to a free society.

1.4 ADVANTAGES OF OPEN SOURCES

· Reliability

· Security

· Combats Piracy

· Total Cost of Ownership

· Non Quantitative Issues

· Freedom from control by another

· Protection from licensing litigation

· Flexibility

· Social/Moral/Ethical Issues

· Innovation

1.5 APPLICATIONS OF OPEN SOURCES

- Active user groups of GNU/Linux in various Indian cities
- Goa Schools Computers Project – GSCP – A collaborative effort
- Localisation of GNU/Linux in Tamil, Hindi, Gujrathi, Bengali and Punjabi
- Contributions to KDE, GNOME
- Anjuta -IDE for C & C++ on GNU/Linux by Naba Kumar
- Mayavi – Scientific Data Visualizer
- KGDB – Kernel patch to debug the Linux Kernel
- National Resource Center for Free Open Source Software-(NRCFOSS) jointly implemented by AU-KBC Centre, MIT Campus.

NOTES

Check your Progress

1. What is Public Domain Software?
2. What is Freeware?
3. What is Firmware?
4. What is FOSS?
5. What arean Advantages of Open Sources?

1.6. ANSWERS TO CHECK YOUR PROGRESS

1. Public domain software refers to any program that is not copy righted. This software is free and can be used without restrictions, that is, the user can copy, distribute, and even modify the software without obtaining permission from the software developer.
2. The term freeware is commonly used for copyrighted software given away free by its author. It is available for free but the author retains the copyright, which means that a user does not have the right to modify anything in the software that is not explicitly allowed by the developer.
3. Firmware is a combination of software, permanently stored in the memory. As the name suggests, firmware is a program or data that has been written onto read only memory (ROM).
4. The name “FOSS” is a recursive acronym for “Free Open Source Software”. Free software is a matter of the users’ freedom to run, copy, distribute, study, change and improve the software.
5. Some of advantages are:
   - Reliability
   - Security
   - Combats Piracy
   - Total Cost of Ownership
   - Non-Quantitative Issues
   - Freedom from control by another

- Protection from licensing litigation
- Flexibility
- Social/Moral/Ethical Issues
- Innovation

NOTES

1.7. SUMMARY

- Firmware is a combination of software, permanently stored in the memory. As the name suggests, firmware is a program or data that has been written onto read only memory (ROM).
- Proprietary describes a technology or product that is owned exclusively by a single company that carefully guards knowledge about the technology or the product's internal working.
- The program must include source code, and must allow distribution in source code as well as compiled form.
- The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

1.8. KEYWORDS

FOSS: The name "FOSS" is a recursive acronym for "Free Open Source Software". Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software.

Free software: Free software is software that gives you the user the freedom to share study and modify it.

Free Redistribution: The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources.

1.9. SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer questions:

1. What is Distribution of License?
2. What is Shareware?
3. What is Proprietary Software?
4. What are the needs of open source?
5. What are applications of open source?

Long Answer questions:

1. Explain briefly about open sources and its advantages?
2. Explain about Need of open sources?
3. What are Software terminologies?

1.10. FURTHER READINGS

Rémy Card, Eric Dumas, and Franck Mével. The Linux kernel book. John Wiley & Sons, Inc., 2003.

Steve Suchring. MySQL BBible. John Wiley, 2002.

Rasmus Lerdorf and Levin Tatroe. Programming PHP. " O'Reilly Media, Inc., 2002.

Wesley J. Chun. Core Python Programming. Prentice Hall, 2001.

Martin C. Brown. Perl: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Steven Holzner. PHP: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

NOTES

UNIT 2
OPEN SOURCE OPERATING SYSTEMS

Structure

2.0 INTRODUCTION

This unit briefs the open source operating system Linux with its general overview of how its available and its usage and working mode are explained,

2.1OBJECTIVE

This unit helps the users to understand linx by

- Learning the overview of Linux
- User and Kernel Mode
- Process in Linux

2.2 OPEN SOURCE OPERATING SYSTEM

Linux

Linus Torvalds [Matt Welsh], a student at the University of Helsinki, created the first version of “Linux” in August 1991. Released as an open-source software under the Free Software Foundation's GNU General Public License (GPL), Linux quickly grew into a complete operating-system package, with contributions from hundreds of programmers. Since the release of version 1.0 in 1994, organizations have been able to download free copies of Linux. One could also purchase commercial distributions of Linux from companies such as Slackware, Red 3 Hat etc who also provide consultancy, services, and maintenance [Carla Schroder]. Many people raise a question about Linux “if it's released under the Free Software

Foundation's GPL, shouldn't it be free?”. The answer is no. A company can charge money for products that include Linux, as long as the source code is made available. The GPL allows people to distribute (and charge for) their own versions of free software [Carla Schroder]. According to the Free Software Foundation, the "free" in free software refers to freedom or liberty, not price. In the foundation's definition, organizations have the freedom to run software for any purpose, study how it works, modify, improve and re-release it.

Another common misconception about Linux is that it's a complete operating system. In reality Linux refers to the “ kernel or core ” of the operating system. Combining Linux with a set of open-source GNU programs from the Free Software Foundation turns it into what most people know as Linux ”forming both the full operating system and the core of most Linux distributions”. Distributions are the versions of Linux, GNU programs, and other tools that are offered by different companies, organizations, or individuals. Popular distributions include Red Hat, Debian, SuSE, Caldera, and others. Each distribution might be based on a different version of the Linux kernel, but all migrate forward over time, picking up core changes that are made to the kernel and keeping everything in somewhat loose synchronization.

Eric S. Raymond's famous essay, "The Cathedral and the Bazaar," argues that most commercial software is built like cathedrals by small groups of artisans working in isolation. Open-source software, like Linux, is developed collectively over the Internet, which serves as an electronic bazaar for innovative ideas. The first of the two programming styles is closed source – the traditional factory-production model of proprietary software, in which customers get a sealed block of computer binary that they cannot examine, modify, or evolve. The other style is opensource, the Internet engineering tradition in which software source code is generally available for inspection, independent peer review, and rapid evolution. Linux operating environment is the standard-bearer of the open source approach.

With Open Source products like Linux, new changes come through an open development model, meaning that all new versions are available to the public, regardless of their quality. "Linux' s versioning scheme is designed to let users understand whether they're using a stable version or adevelopment version," says Jim Enright, director of Oracle's Linux program office. "Even decimal-numbered releases [such as 2.0, 2.2, and 2.4] are considered stable versions, while odd numbered releases [such as 2.3 and 2.5] are beta-quality releases intended for developers only." For much of the 1990s, Linux was primarily an experiment: something that developers fiddled with and used on local servers to see how well it worked and how secure it was. Then, with the internet boom of the late 1990s, many companies started using Linux for their Web servers, fueling the first wave of corporate Linux adoption leading to over 30 percent penetration of web server market by 2002.

2.3 GENERAL OVERVIEW

Here are some of the benefits and features that Linux provides over single-user operating systems and other versions of UNIX for the PC.

• **Full multitasking and 32-bit support** : Linux, like all other versions of UNIX, is a real multitasking system, allowing multiple users to run many programs on the same system at once. The performance of a 50 MHz 486 system running Linux is comparable to many low- to medium-end workstations, such as those from Sun Microsystems and DEC, running proprietary versions of UNIX. Linux is also a full 32-bit operating system, utilizing the special protected-mode features of the Intel 80386 and 80486 processors.

• **TCP/IP networking support** : TCP/IP (“Transmission Control Protocol/Internet Protocol”) is the set of protocols which links millions of computers into a worldwide network known as the Internet. With an Ethernet connection, one can have access to the Internet or to a local area network from your Linux system. Or, using SLIP (“Serial Line Internet Protocol”), you can access the Internet over the phone lines with a modem.

• **Virtual memory and shared libraries**: Linux can use a portion of the hard drive as virtual memory, expanding the total amount of available RAM. Linux also implements shared libraries, allowing programs which use standard subroutines to find the code for these subroutines in the libraries at runtime. This saves a large amount of space, as each application doesn't store its own copy of these common routines.

Linux Distributions

Here are some of the more popular distributions of Linux.

* Mandrake

* Red Hat

* SuSE

* Caldera

* Corel

* Debian

* Slackware

* TurboLinux

* Ubuntu

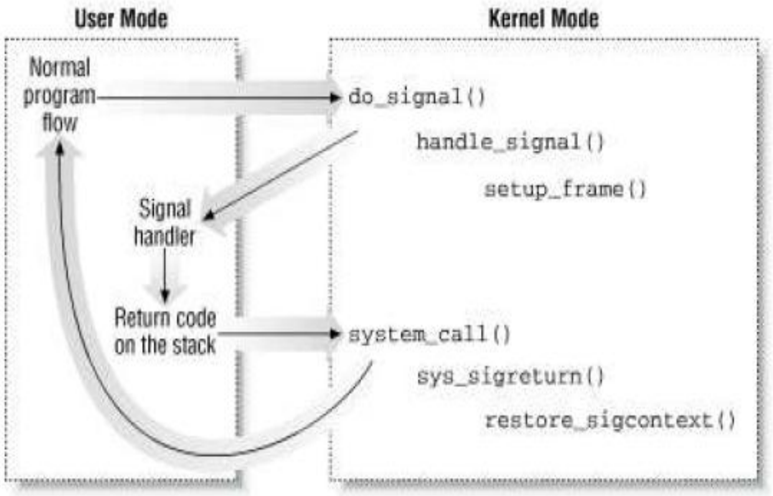
2.4 KERNEL MODE

The system starts in kernel mode when it boots and after the operating system is loaded, it executes applications in user mode. There are some privileged instructions that can only be executed in kernel mode. These are interrupt instructions, input output management etc. If the privileged instructions are executed in user mode, it is illegal and a trap is generated.

The mode bit is set to 0 in the kernel mode. It is changed from 0 to 1 when switching from kernel mode to user mode

2.5 USER MODE

The system is in user mode when the operating system is running a user application such as handling a text editor. The transition from user mode to kernel mode occurs when the application requests the help of operating system or an interrupt or a system call occurs. The mode bit is set to 1 in the user mode. It is changed from 1 to 0 when switching from user mode to kernel mode.



Necessity of Dual Mode (User Mode and Kernel Mode) in Operating System

The lack of a dual mode i.e user mode and kernel mode in an operating system can cause serious problems. Some of these are: A running user program can accidentally wipe out the operating system by overwriting it with user data. Multiple processes can write in the same system at the same time, with disastrous results. These problems could have occurred in the MS-DOS operating system which had no mode bit and so no dual mod

2.6 PROCESS

A process is usually defined as an instance of a program in execution; thus, if 16 users are running vi at once, there are 16 separate processes..

Process Descriptor

To manage processes, the kernel must know the process's priority, whether it is running on the CPU or blocked on some event, what address space has been assigned to it and so on. This is the role of the process descriptor, that is, of a task_struct type

Process State

The following are the possible process states:

TASK_RUNNING The process is either executing on the CPU or waiting to be executed.

TASK_INTERRUPTIBLE The process is suspended (sleeping) until some condition becomes true.

TASK_UNINTERRUPTIBLE like previous state, except that delivering a signal to the sleeping process leaves its state unchanged.

TASK_STOPPED Process execution has been stopped:

The process list To allow an efficient search through processes of a given type the kernel creates several lists of processes. Each list consists of pointers to process descriptors

2.6.1 Creating Processes

The clone( ), fork( ), and vfork( ) System Calls

Lightweight processes are created in Linux by using a function named __clone( ), which makes use of four parameters:

fn: Specifies a function to be executed by the new process; when the function returns, the child terminates.

Arg: Pointer to data passed to the fn( ) function.

Flags: Miscellaneous information.

child_stack Specifies the User Mode stack pointer to be assigned to the esp register of the child

2.6.2 Destroying Processes

Process Termination

All process terminations are handled by the do_exit( ) function, which removes most references to the terminating process from kernel data structures. The do_exit( ) function executes the following actions:

1. Sets the PF_EXITING flag in the flag field of the process descriptor to denote that the process is being eliminated.

2. Examines the process's data structures related to paging, filesystem, open file descriptors, and signal handling, respectively, with the __exit_mm( ), __exit_files( ), __exit_fs( ), and _ _exit_sighand( ) functions.

3. Sets the state field of the process descriptor to TASK_ZOMBIE.

4. Sets the exit_code field of the process descriptor to the process termination code.

5. Invokes the exit_notify( ) function to update the parenthood relationships of both the parent process and the children processes.

6. Invokes the schedule( ) function to select a new process to run. Since a process in a TASK_ZOMBIE state is ignored by the scheduler, the process will stop executing right after the switch_to macro in schedule( ) is invoked.

2.6.3 Process Removal

The release( ) function releases the process descriptor of a zombie process by executing the following steps:

1. Invokes the free_uid( ) function to decrement by 1 the number of processes created up to now by 3 the user owner of the terminated process.

2. Invokes add_free_taskslot( ) to free the entry in task that points to the process descriptor to be released.

3. Decrements the value of the nr_tasks variable.

4. Invokes unhash_pid( ) to remove the process descriptor from the pidhash hash table.

5. Uses the REMOVE_LINKS macro to unlink the process descriptor from the process list.

6. Invokes the free_task_struct( ) function to release the 8 KB memory area used to contain the process descriptor and the Kernel Mode stack.

Check your Progress

1. What are the distributions of Linux?
2. What is Kernel Mode?
3. What is User Mode?
4. What are the two different Styles of Programming?
5. State the Necessity of Dual Mode in Operating

2.7 ANSWERS TO CHECK YOUR PROGRESS

1. Some of the more popular distributions of Linux are:
    - Mandrake
    - Red Hat
    - SuSE
    - Caldera
    - Corel
    - Debian
    - Slackware
    - TurboLinux
    - Ubuntu
2. The system starts in kernel mode when it boots and after the operating system is loaded, it executes applications in user mode. There are some

privileged instructions that can only be executed in kernel mode. These are interrupt instructions, input output management etc. If the privileged instructions are executed in user mode, it is illegal and a trap is generated. The mode bit is set to 0 in the kernel mode. It is changed from 0 to 1 when switching from kernel mode to user mode.

3. The system is in user mode when the operating system is running a user application such as handling a text editor. The transition from user mode to kernel mode occurs when the application requests the help of operating system or an interrupt or a system call occurs. The mode bit is set to 1 in the user mode. It is changed from 1 to 0 when switching from user mode to kernel mode.

4. The first of the two programming styles is closed source – the traditional factory-production model of proprietary software, in which customers get a sealed block of computer binary that they cannot examine, modify, or evolve. The other style is opensource, the Internet engineering tradition in which software source code is generally available for inspection, independent peer review, and rapid evolution.

5. The lack of a dual mode i.e. user mode and kernel mode in an operating system can cause serious problems. Some of these are: A running user program can accidentally wipe out the operating system by overwriting it with user data. Multiple processes can write in the same system at the same time, with disastrous results. E.g., These problems could have occurred in the MS-DOS operating system which had no mode bit and so no dual mode.

2.8 SUMMARY

- Open-source software, like Linux, is developed collectively over the Internet, which serves as an electronic bazaar for innovative ideas.
- Linux, like all other versions of UNIX, is a real multitasking system, allowing multiple users to run many programs on the same system at once.
- The system starts in kernel mode when it boots and after the operating system is loaded, it executes applications in user mode.
- The system is in user mode when the operating system is running a user application such as handling a text editor.

2.9 KEYWORDS

GNU software support : Linux supports a wide range of free software written by the GNU Project, including utilities such as the GNU C and C++ compiler, gawk, groff, and so on.

X Window System : The X Window System is the de facto industry standard graphics system for UNIX machines. A free version of The X Window System (known as “Xfree86”) is available for Linux.

Virtual memory and shared libraries: Linux also implements shared libraries, allowing programs which use standard subroutines to find the code for these subroutines in the libraries at runtime.

2.10 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer questions:

1. What is Virtual Memory and Shared Libraries? Mention the uses.
2. Write a short note on Process State
3. What is the Role of ProcessDescriptor?
4. What is Creating and Destroying in Process?
5. Write a short note on The X Window System?

Long Answer questions:

1. Explain about Process Creating and steps involved in it.
2. Explain briefly about Process Destroying and Process Removal.
3. Explain and illustrate User Mode and Kernel Mode in Operating system and its necessity.

2.11 FURTHER READINGS

Rémy Card, Eric Dumas, and Franck Mével. The Linux kernel book. John Wiley & Sons, Inc., 2003.

Steve Suchring. MySQL BBible. John Wiley, 2002.

Rasmus Lerdorf and Levin Tatroe. Programming PHP. " O'Reilly Media, Inc., 2002.

Wesley J. Chun. Core Python Programming. Prentice Hall, 2001.

Martin C. Brown. Perl: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Steven Holzner. PHP: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Vikram Vaswani. MySQL: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

UNIT 3
ADVANCED CONCEPTS

Structure

3.0 INTRODUCTION

This unit explains the advanced concepts of open source software with linx where the scheduling and signals are discussed briefly. The development of the open source software Linux is also discussed

3.1 OBJECTIVE

This unit helps the users to understand the advance concepts such as

- Scheduling
- Cloning
- Personalities
- Signals

3.2 SCHEDULING CONCEPTS

3.2.1 Scheduling Policy

The scheduling algorithm of traditional UNIX operating systems must fulfil several conflicting objectives: fast process response time, good throughput for background jobs, avoidance of process starvation, reconciliation of the needs of low- and high-priority processes, and so on. The set of rules used to determine when and how selecting a new process to run is called scheduling policy. Linux scheduling is based on the time-sharing technique several processes are allowed to run "concurrently,"

3.2.2 Process Pre-emption

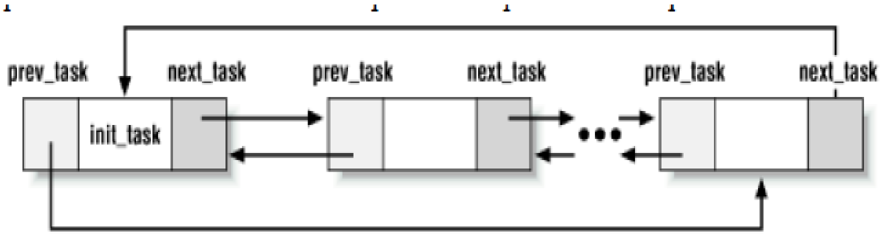
Linux processes are pre-emptive. If a process enters the TASK_RUNNING state, the kernel checks whether its dynamic priority is greater than the priority of the currently running the execution of current is interrupted and the scheduler is invoked to select another process to run. Be aware that a pre-empted process is not suspended, since it remains in the TASK_RUNNING state; it simply no longer uses the CPU. The Linux kernel is not pre-emptive, which means that a process can be pre-empted only while running in User Mode

3.2.3 The Scheduling Algorithm

The Linux scheduling algorithm works by dividing the CPU time into epochs. In a single epoch, every process has a specified time quantum whose duration is computed when the epoch begins. The time quantum value is the maximum CPU time portion assigned to the process in that epoch. When a process has exhausted its time quantum, it is preempted and replaced by another

runnable process. The epoch ends when all runnable processes have exhausted their quantum.

The schedule( ) Function- schedule( ) implements the scheduler. Its objective is to find a process in the run queue list and then assign the CPU to it.



Direct invocation

The scheduler is invoked directly when the current process must be blocked right away because the resource it needs is not available. In this case, the kernel routine that wants to block it proceeds as follows:

1. Inserts current in the proper wait queue

2. Changes the state of current either to TASK_INTERRUPTIBLE or to PROCESS TASK_UNINTERRUPTIBLE 4

3. Invokes schedule( )

4. Checks if the resource is available; if not, goes to step 2

5. Once the resource is available, removes current from the wait queue

Lazy invocation

The scheduler can also be invoked in a lazy way by setting the need_resched field of current to 1. Actions performed by schedule( )

1. The schedule( ) function starts by running the functions left by other kernel control paths in various queues. The function invokes run_task_queue(&tq_scheduler); The function then executes all active unmasked bottom halves.

2. Now comes the actual scheduling, and therefore a potential process switch. The value of current is saved in the prev local variable. First, a check is made to determine whether prev is a Round Robin real-time process that has exhausted its quantum. If so, schedule( ) assigns a new quantum to prev and puts it at the bottom of the runqueue list:

3. schedule( ) examines the state of prev. If it has nonblocked pending signals and its state is TASK_INTERRUPTIBLE, the function wakes up the process as Task_running.

4. If prev is not in the TASK_RUNNING state, prev must be removed from the runqueue list:

5. Next, schedule( ) must select the process to be executed in the next time quantum. To that the function scans the runqueue list. The objective is to store in next the process descriptor pointer of the highest priority process. schedule( ) repeatedly invokes the goodness( ) function on the runnable processes to determine the best candidate:

6. A further check must be made at the exit of the loop to determine whether c is 0. This occurs only when all the processes in the runqueue list have exhausted their quantum

7. If a process other than prev has been selected, a process switch must take place. Before performing it, however, the context_swtch field of kstat is increased by 1 to update the statistics maintained by the kernel.

3.3 SIGNALS

A signal is a very short message that may be sent to a process or to a group of processes. The only information given to the process is usually the number identifying the signal. A set of macros whose names start with the prefix SIG is used to identify signals; For example, the SIGCHLD macro yields the identifier of the signal that is sent to a parent process when some child stops or terminates.

Signals serve two main purposes:

- To make a process aware that a specific event has occurred
- To force a process to execute a signal handler function included in its code

The kernel distinguishes two different phases related to signal transmission:

Signal sending The kernel updates the descriptor of the destination process to represent that a new signal has been sent. Signal receiving The kernel forces the destination process to react to the signal by changing its execution state or by starting the execution of a specified signal handler or both. The following factors must be taken into consideration:

- Signals are usually received only by the currently running process

- Signals of a given type may be selectively blocked by a process; in this case, the process will not receive the signal until it removes the block.

3.3.1 Process

Actions Performed upon Receiving a Signal 5 There are three ways in which a process can respond to a signal:

- Explicitly ignore the signal.

- Execute the default action associated with the signal

Abort

The process is destroyed. Dump The process is destroyed and a core file containing its execution context is created.

Ignore

The signal is ignored. Stop- The process is stopped, that is, put in a TASK_STOPPED state

Continue If the process is stopped (TASK_STOPPED), it is put into the TASK_RUNNING

state. Catch the signal by invoking a corresponding signal-handler function.

3.3.2 Sending a Signal

When a signal is sent to a process, the kernel delivers it by invoking the send_sig_info( ), send_sig( ),

force_sig( ), or force_sig_info( ) functions.

The send_sig_info( ) and send_sig( ) Functions

The send_sig_info( ) function acts on three parameters:

Sig The signal number. Info Either the address of a siginfo_t table associated with real-time signals or one of two special values. A pointer to the descriptor of the destination process.

1. The send_sig_info( ) function starts by checking whether the parameters are correct:

if (sig < 0 || sig > 64) return -EINVAL;

2. The function checks then if the signal is being sent by a User Mode process.

3. If the signal is sent by a User Mode process, the function determines whether the operation is allowed. If the sig parameter has the value 0, the function returns immediately without sending any signal: since is not a valid signal number. Some types of signals might nullify other pending signals for the destination process.

4. Next, send_sig_info( ) checks whether the new signal can be handled immediately.

3.3.3 Receiving Signals

The kernel checks whether there are nonblocked pending signals before allowing a process to resume its execution in User Mode. This check is performed in ret_from_intr( ) every time an interrupt or an exception has been handled by the kernel routines. In order to handle the nonblocked pending signals, the kernel invokes the do_signal( ) function, which receives two parameters: Regs The address of the stack area

3.4 PROCESS

Oldest The address of a variable where the function is supposed to save the bit mask array of 6 blocked signals. The heart of the do_signal( ) function consists of a loop that repeatedly invokes dequeue_signal( ) until no more nonblocked pending signals are left. The return code of dequeue_signal( ) is stored in the signr local variable: if its value is 0, it means that all pending signals have been handled and do_signal( ) can finish

Ignoring the Signal

When a received signal is explicitly ignored, the do_signal( ) function normally just continues with a new execution of the loop and therefore considers another pending signal. If the signal received is SIGCHLD, the sys_wait4( ) service routine of the wait4( ) system call is invoked to force the process to read information about its children.

Executing the Default Action for the Signal

If ka->sa.sa_handler is equal to SIG_DFL, do_signal( ) must perform the default action of the signal. The only exception comes when the receiving process is init, in which case the signal is discarded. The signals whose default action is "stop" may stop the current process. In order to do this, do_signal( ) sets the state of current to TASK_STOPPED and then invokes the schedule( ) Function.

3.5 CLONING

A clone process is created, using primitive type clone, by duplicating its parent process. But, unlike classical processes, it may share its context with its parent.

The standard form of the clone function is as follows:

```
int clone(int (*fn)(), void *child stack, int flag, int nargs,…);
```

- The parameter fn is the pointer from the child process to the function to be executed.

- The parameter child stack is the pointer to the zone of memory allocated for the stack of child process. The parameter Flags defines method for cloning.

- The parameter nargs defines the number of arguments to be passed to the function pointed by fn and is followed by these arguments

However, it is not necessary to have the source code in order to make a clone of a program. In fact, having access to the source code can be undesirable because it could subconsciously influence a programmer and result in imitation of part of the code rather than creating entirely original code. A variety of techniques have been developed for cloning software, including reimplementation from official documentation accompanying the software and from unofficial documentation. Reverse engineering becomes particularly important when the documentation is incomplete, which is often the case. Among the ways in which reverse engineering can be accomplished are through observation of information exchange, disassembly using a disassembler and decompilation using a decompiler. A compiler converts source code into machine language, which can be read directly by a computer's CPU (central processing unit); a decompiler attempts to convert machine language back into source code. An assembler is a program that translates an assembly language (which is a very low level language close to machine language) into machine language.

Clones are generally cheaper than the original software, and in many cases they are free software (i.e., software that is free both in a monetary sense and with regard to use). However, they are not necessarily inferior. In fact, in some cases they are as good as or superior to the originals, as is clearly illustrated by Linux. They also frequently have the benefit of providing competition for the original software, thereby stimulating its developers to improve its performance and features and to lower its price. In contrast to proprietary software, there is no reason to make clones of free software. This is because, by definition, the source code for free software is freely available to anyone to use for any purpose, including modification and redistribution of such modified versions. Occasionally, there is sufficient dissatisfaction with the way in which a free software program is being developed to result in a project fork, which is the starting of a new development branch independent of the the existing project but based on the same source code.

A second meaning of the word clone in a software context is to make an exact copy of a file, directory or disk inclusive of any files and subdirectories within that directory or disk.The term is also used in a hardware context. Most notably, when IBM introduced its revolutionary IBM PC in 1981, other companies decided to develop clones as a legal reimplementation from the PC's documentation and through reverse engineering. Because most of the components were publicly available with the exception of the BIOS (basic input output system), the only major task was reverse engineering the BIOS.

3.6 PERSONALITIES

In order to allow programs coming from other operating system to be run in Linux, Linux supports the idea of personalities. Each process is assigned to an execution domain. This domain specifies the way in which system calls are carried out, and the way in which messages are processed.

System calls: Linux uses software interrupts to change into kernel mode, whilst other UNIX system use an inter- segment jump.

Message number specified by processes: When a process specifies a message number, for example in calling the primitive sanction or kill, the message number is converted by means of look-up-table

Number of messages sent to processes: when a message is to be sent to a process, the message number is converted by means of look-up-table

The system call personality allows a process to modify its execution domain in order that Linux can emulate the behaviour of another operating system. int personality(int pers);

3.7 DEVELOPMENT WITH LINUX

It's Free Linux is free. Really and truly free. One can browse to any of the distributors of Linux, find the "download" link and download a complete copy of the entire operating system plus extra software without paying any thing. One can also buy a boxed version of course. For a nominal price, the CDs are available, manuals get door-delivered, plus there is telephone or online support. By comparison, "home" versions of popular commercial OS would cost thousands of rupees.

With Linux you also don't have to worry about paying again every time you upgrade the operating system - the upgrades are obviously free too. With commercial OS, upgrades also have to be paid for every time one is announced.

It's Open Source

This means two things: First, that the CDs (or the download site) contain an entire copy of the source code for Linux. Secondly, the user can legally make modifications to improve it. While this might not mean much to non-programmers, there are thousands of people with programming capability who could improve the code or fix problems quickly. When a problem is found, it is sent off to the coordinating team in charge of the module in question, who will update the software and issue a patch. What all this boils down to is that bugs in Linux get fixed much faster than any other operating system.

It's Modular

Commercial Operating Systems normally get installed as a complete unit. One cannot, for 7 example, install them without their Graphical User Interface, or without its printing support - install everything or nothing. Linux, on the other hand, is a very modular operating system. One could install or run exactly the bits and pieces of Linux that are needed. In most cases, the choice is on one of the predefined setups from the installation menu, but is not compulsory. In some cases this makes a lot of sense. For example, while setting up a server, one might want to disable the graphical user interface once it is set up correctly, thus freeing up memory and the processor for the more important task at hand.

It also allows the users to upgrade parts of the operating system without affecting the rest. For example, one could get the latest version of Gnome or KDE without changing the kernel.

It's got More Choices

Also due to its modularity, there is more choice of components to use. One example is the user interface. Many users choose KDE, which is very easy to learn for users with Windows experience. Others choose Gnome, which is more powerful but less similar to Windows. There are also several simple alternatives for less-powerful computers, which make less demand on the hardware available.

It's Portable

Linux runs on practically every piece of equipment which qualifies as a computer. It can be run on huge multiprocessor servers or a PDA. Apart from Pentiums of various flavors, there are versions of Linux (called "ports") on Atari, Amiga, Macintosh, PowerMac, PowerPC, NeXT, Alpha, Motorola, MIPS, HP, PowerPC, Sun Sparc, Silicon Graphics, VAX/MicroVax, VME, Psion 5, Sun UltraSparc, etc.

It's got lots of Extras

Along with the Linux CD, normally quite a lot of software gets thrown in, which is not usually included with operating systems. Using only the applications that come with Linux, one could setnup a full web, ftp, database and email server for example. There is a firewall built into the kernel of the operating system, one or more office suites, graphics programs, music players, and lots more. Different distributions of Linux offer different "extra programs". Slackware, for example, is quite simple (though it still provides all the commonly needed programs), while SuSE Linux comes with seven CDs and a DVD-ROM!

It is Stable

All applications can crash, but in many systems, the only recourse is to switch off and reboot (and with some new "soft-switch" PCs, even that doesn't work - you have to pull out the power cable). In comparison, Linux is rock-solid. Every application runs independently of all others - if one crashes, it crashes alone. Most Linux servers run for months on end, never shutting down or rebooting. Even the GUI is independent of the kernel of the operating system.

Linux Configuration Tool

LinuxConf is a popular utility which allows the configuration of most parts of Linux and its applications from one place

Check your Progress

1. What is called scheduling Policy?
2. State the Process of Pre-emption.
3. How the Scheduling Algorithm works?
4. What is Signal? Mention the Properties?
5. Write Short note on Personalities? And where it is used in Linux?

3.8 ANSWERS TO CHECK YOUR PROGRESS

1. The scheduling algorithm of traditional UNIX operating systems must fulfil several conflicting objectives: fast process response time, good throughput for background jobs, avoidance of process starvation, reconciliation of the needs of low- and high-priority processes, and so on. The set of rules used to determine when and how selecting a new process to run is called scheduling policy.
2. Linux processes are pre-emptive. If a process enters the TASK_RUNNING state, the kernel checks whether its dynamic priority is greater than the priority of the currently running the execution of current is interrupted and the scheduler is invoked to select another process to run. Be aware that a pre-empted process is not suspended, since it remains in the TASK_RUNNING state; it simply no longer uses the CPU. The Linux kernel is not pre-emptive, which means that a process can be pre-empted only while running in User Mode
3. The Linux scheduling algorithm works by dividing the CPU time into epochs. In a single epoch, every process has a specified time quantum whose duration is computed when the epoch begins. The time quantum value is the maximum CPU time portion assigned to the process in that epoch. When a process has exhausted its time quantum, it is pre-empted and replaced by another
4. A signal is a very short message that may be sent to a process or to a group of processes.
    - To make a process aware that a specific event has occurred
    - To force a process to execute a signal handler function included in its code

5. In order to allow programs coming from other operating system to be run in Linux, Linux supports the idea of personalities. Each process is assigned to an execution domain. This domain specifies the way in which system calls are carried out, and the way in which messages are processed.
    - System calls

- Message number specified by processes
- Number of messages sent to processes

3.9 SUMMARY

- The Linux scheduling algorithm works by dividing the CPU time into epochs. In a single epoch, every process has a specified time quantum whose duration is computed when the epoch begins.
- The kernel checks whether there are nonblocked pending signals before allowing a process to resume its execution in User Mode.
- A clone process is created, using primitive type clone, by duplicating its parent process. But, unlike classical processes, it may share its context with its parent.
- Copyright is the power granted by a government to the creator of a creative work (e.g., the source code for a program, a magazine article, a poem, a painting or a musical composition) that gives that creator the exclusive, although transferable, right to copy or perform that work for a defined period of time.

3.10 KEYWORDS

Process Pre-emption: Linux processes are pre-emptive. If a process enters the TASK_RUNNING state, the kernel checks whether its dynamic priority is greater than the priority of the currently running the execution of current is interrupted and the scheduler is invoked to select another process to run.

runnable process. The epoch ends when all runnable processes have exhausted their quantum.

Signal: A signal is a very short message that may be sent to a process or to a group of processes.

System calls: Linux uses software interrupts to change into kernel mode, whilst other UNIX system use an inter- segment jump.

Message number specified by processes: When a process specifies a message number, for example in calling the primitive sanction or kill, the message number is converted by means of look-up-table.

3.11 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer questions:

1. What is Virtual Memory and Shared Libraries? Mention the uses.
2. What is wine? And why we using it?
3. What are the three ways a process responds to a signal?
4. What is the Reason for the existence of Cloning software?

Long Answer questions:

1. Explain about Sending and Receiving Signal.
2. Explain briefly about Scheduling Algorithm
3. Explain about Cloning.

3.12. FURTHER READINGS

Rémy Card, Eric Dumas, and Franck Mével. The Linux kernel book. John Wiley & Sons, Inc., 2003.

Steve Suchring. MySQL BBible. John Wiley, 2002.

Rasmus Lerdorf and Levin Tatroe. Programming PHP. " O'Reilly Media, Inc., 2002.

Wesley J. Chun. Core Python Programming. Prentice Hall, 2001.

Martin C. Brown. Perl: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

BLOCK – II
OPEN SOURCE DATABASE

UNIT 4
MySQL

NOTES

Structure

4.0 INTRODUCTION

This unit explains the basic programming of MySQL by establishing and connecting with the server with the root password and how to write own sql programs.

4.1 OBJECTIVE

This unit helps the users to

- Undertand and install mysql
- Write own programs

4.2 MySQL

MySQL is the most popular open source SQL database management system (DBMS). A fast, reliable, easy-to-use, multi-user multi-threaded relational database system. It is freely available and released under GPL (GNU General Public License). MySQL is a data storage area. In this storage area, there are small sections called Tables.

Advantages

- MySQL is Cross-Platform.
- MySQL is fast.
- MySQL is free.
- Reliable and easy to use.
- Multi-Threaded multi-user and robust SQL Database server.

Disadvantages

- Missing Sub-selects.
- MySQL doesn't yet support the Oracle SQL extension.
- Does not support Stored Procedures and Triggers.
- MySQL doesn't support views, but this is on the TODO.

NOTES

Following are the tools to manage MySQL server:

- mysqld - MySQL server daemon. It is used to start the mysql server.
- mysqladmin – Used to perform administrative tasks.
- mysql - A command-line interface for end users to manage user data objects.
- mysqlcheck - A command-line interface for administrators to check and repair tables.
- mysqlshow - A command-line interface for end users to see information on tables and columns.
- mysqldump - A command-line interface for administrators or end users to export data from the server to files.
- mysqlimport - A command-line interface for administrators or end users to load data files into tables program tool to load data into tables.

4.3 SETTING UP ACCOUNT

In order to provide access the MySQL database you need to create an account to use for connecting to the MySQL server running on a given host. Use the GRANT statement to set up the MySQL user account. Then use that account's 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 UnixMySQL user names can be up to 16 characters long.

MySQL passwords have nothing to do with passwords for logging in to your operating system.

Account Management Statements

- CREATE USER
- DROUP USER
- RENAME USER
- REVOKE
- SET PASSWORD

CREATE USER

Syntax:

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

Example:

CREATE USER 'monty'@'localhost' IDENTIFIED BY 'some_pass';

NOTES

DROP USER

Syntax:

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

Example:

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

RENAME USER

Syntax

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

Example

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

SET PASSWORD

Syntax

```
SET PASSWORD [FOR user] =
    {
        PASSWORD('some password')
| OLD_PASSWORD('some password') | 'encrypted password'
    }
```

Example:

```
    SET PASSWORD FOR 'bob'@'%.loc.gov' =
PASSWORD('newpass');
```

4.4 STARTING, TERMINATING AND WRITING YOUR OWN SQL PROGRAMS

Invoking MySQL Programs

To invoke a MySQL program from the command line (that is, from your shell or command prompt), enter the program name followed by any options or other arguments needed to instruct the program what you want it to do. The following commands show some sample program invocations. “shell>” represents the prompt for your command interpreter; it is not part of what you type. The particular prompt you see depends on your command interpreter. Typical prompts are $ for sh or bash, % for csh or tcsh, and C:\> for the Windows command.com or cmd.exe command interpreters.

```
shell> mysql --user=root test

shell> mysqladmin extended-status variables

shell> mysqlshow --help

shell> mysqldump -u root personnel
```

Arguments that begin with a single or double dash (“-”, “--”) specify program options. Options typically indicate the type of connection a program should make to the server or affect its operational mode Non-option arguments (arguments with no leading dash) provide additional information to the program. For example, the mysql program interprets the first non-option argument as a database name, so the command mysql --user=root test indicates that you want to use the test database.

4.4.1 Connecting to the MySQL Server

For a client program to be able to connect to the MySQL server, it must use the proper connection parameters, such as the name of the host where the server is running and the username and password of your MySQL account. Each connection parameter has a default value, but you can override them as necessary using program options specified either on the command line or in an option file.The examples here use the mysql client program, but the principles apply to other clients such as mysqldump, mysqladmin, or mysqlshow.

This command invokes mysql without specifying any connection parameters explicitly:

```
shell> mysql
```

Because there are no parameter options, the default values apply:The default hostname is localhost. On Unix, this has a special meaning, as described later.The default username is ODBC on Windows or your Unix login name on Unix.No password is sent if neither -p nor --password is given.To specify the hostname and username explicitly, as well as a password, supply appropriate options on the command line:

```
shell> mysql --host=localhost --user=myname --password=mypass
shell> mysql -h localhost -u myname -pmypass
```

For password options, the password value is optional:If you use a -p or --password option but do not specify the password value, the client program prompts you to enter the password. The password is not displayed as you enter it. This is more secure than giving the password on the command line. Any user on your system may be able to see a password specified on the command line by executing a command such as ps auxw. See Section 5.5.6, “Keeping Your Password Secure”.If you use a -p or --password option and do specify the password value, there must be no space between -p or --password= and the password following it.

On Unix, MySQL programs treat the hostname localhost specially, in a way that is likely different from what you expect compared to other network-based programs. For connections to localhost, MySQL programs attempt to connect to the local server by using a Unix socket file. This occurs even if a --port or -P option is given to specify a port number. To ensure that the client makes a TCP/IP connection to the local server, use --host or -h to specify a hostname value of 127.0.0.1, or the IP address or name of the local server. You can also specify the connection protocol explicitly, even for localhost, by using the --protocol=TCP option. For example:

```
shell> mysql --host=127.0.0.1

shell> mysql --protocol=TCP
```

The --protocol option enables you to establish a particular type of connection even when the other options would normally default to some other protocol.On Windows, you can force a MySQL client to use a named-pipe connection by specifying the --pipe or --protocol=PIPE option, or by specifying . (period) as the host name. If named-pipe connections are not enabled, an error occurs. Use the --socket option to specify the name of the pipe if you do not want to use the default pipe name.

Connections to remote servers always use TCP/IP. This command connects to the server running on remote.example.com using the default port number (3306):

```
shell> mysql --host=remote.example.com
```

To specify a port number explicitly, use the --port or -P option:

```
shell> mysql --host=remote.example.com --port=13306
```

You can specify a port number for connections to a local server, too. However, as indicated previously, connections to localhost on Unix will use a socket file by default. You will need to force a TCP/IP connection as already described or any option that specifies a port number will be ignored. For this command, the program uses a socket file on Unix and the --port option is ignored:

```
shell> mysql --port=13306 --host=localhost
```

To cause the port number to be used, invoke the program in either of these ways:

```
shell> mysql --port=13306 --host=127.0.0.1
```

```
shell> mysql --port=13306 --protocol=TCP
```

The following options may be used to control how client programs connect to the server:

--host=host_name, -h host_name

The host where the server is running. The default value is localhost.

--password[=pass_val], -p[pass_val]

The password of the MySQL account. As described earlier, the password value is optional, but if given, there must be no space between -p or --password= and the password following it. The default is to send no password.

--pipe, -W

On Windows, connect to the server via a named pipe. This option applies for connections to a local server only. The server must have been started with the --enable-named-pipe option to enable named-pipe connections.

--port=port_num, -P port_num

The port number to use for the connection, for connections made via TCP/IP. The default port number is 3306.

--protocol={TCP|SOCKET|PIPE|MEMORY}

This option explicitly specifies a protocol to use for connecting to the server. It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want. For example, connections on Unix to localhost are made via a Unix socket file by default:

```
shell> mysql --host=localhost
```

To force a TCP/IP connection to be used instead, specify a --protocol option:

```
shell> mysql --host=localhost --protocol=TCP
```

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. To illustrate, here’s what the welcome message looks like (to save space, I won’t show it in any further examples):

Starting and Terminating

```
% 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
```

In general, I'll show mysql commands in examples with no connection parameter options. I assume that you'll supply any parameters that you need, either on the command line, or in an option file (Recipe 1.5) so that you don't have to type them each time you invoke mysql. If you don't have a MySQL username and password, The syntax and default values for the connection parameter options are shown in the following table. These options have both a single-dash short form and a double-dash long form.

```
% mysql -pEnter password: ← enter your password here
```

If you like, you can specify the password directly on the command line by using either - ppassword (note that there is no space after the -p) or --password=password. I don't recommend doing this on a multiple-user machine, because the password may be visible momentarily to other users who are running tools such as ps that report process information. If you get an error message that mysql cannot be found or is an invalid command when you try to invoke it,.To terminate a mysql session, issue a QUIT statement:

```
mysql> QUIT
```

You can also terminate the session by issuing an EXIT statement or (under Unix) by typing Ctrl-D.The way you specify connection parameters for mysql also applies to other MySQL programs such as mysqldump and mysqladmin.

Check your Progress

1. Write short notes on MySQL.
2. Name the tools used to manage MySQL server.
3. Mention the Account Management Statements.
4. How to invoke a MySQL from command line?
5. What are the Advantages of MySQL?

NOTES

4.5 ANSWERS TO CHECK YOUR PROGRESS

1. MySQL is the most popular open source SQL database management system (DBMS). A fast, reliable, easy-to-use, multi-user multi-threaded relational database system.
2. Some of the tools used are
   - Mysqld
   - Mysqladmin
   - Mysql
   - mysqlcheck
   - mysqlshow
   - mysqldump
   - mysqlimport
3. Some of the account management statements are

   - CREATE USER
   - DROUP USER
   - RENAME USER
   - REVOKE
   - SET PASSWORD
4. To invoke a MySQL program from the command line (that is, from your shell or command prompt), enter the program name followed by any options or other arguments needed to instruct the program what you want it to do. The following commands show some sample program invocations. “shell>” represents the prompt for your command interpreter
5. The advantages of MySQL are
   - MySQL is Cross-Platform.
   - MySQL is fast.
   - MySQL is free.
   - Reliable and easy to use.
   - Multi-Threaded multi-user and robust SQL Database server.

4.6 SUMMARY

- In order to provide access the MySQL database you need to create an account to use for connecting to the MySQL server running on a given host.
- To invoke a MySQL program from the command line (that is, from your shell or command prompt), enter the program name followed by any options or other arguments needed to instruct the program what you want it to do.
- You can also terminate the session by issuing an EXIT statement or (under Unix) by typing Ctrl-D.
- 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.

4.7 KEYWORDS

MySQL: It is the most popular open source SQL database management system (DBMS).

User names: It is used by MySQL for authentication purposes, have nothing to do with user names (login names) as used by Windows or UnixMySQL user names can be up to 16 characters long.

4.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer questions:

1. What are the disadvantages of MySQL?
2. Write a syntax with example to Create User and Drop User.
3. Write a syntax with example to Rename User and Revoke.
4. Write a syntax with example to Set a new Password and change the Old Password.

Long Answer questions:

1. Explain the Process involved in Connecting to the MySQL Server.

4.9 FURTHER READINGS

Rémy Card, Eric Dumas, and Franck Mével. The Linux kernel book. John Wiley & Sons, Inc., 2003.

Steve Suchring. MySQL BBible. John Wiley, 2002.

Rasmus Lerdorf and Levin Tatroe. Programming PHP. " O'Reilly Media, Inc., 2002.

Wesley J. Chun. Core Python Programming. Prentice Hall, 2001.

Martin C. Brown. Perl: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Steven Holzner. PHP: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Vikram Vaswani. MySQL: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

UNIT 5
RECORD SELECTION TECHNOLOGY

Structure

5.0 INTRODUCTION

This unit covers the Mysql commands for selection of records from the tables and how strings are manipulated in Mysql and the date and time functions used in Mysql are decribed with their syntax and examples

5.1 Objective

This unit helps the users to learn and understand the following

- Record selection
- Working with strings
- Date and time functions

5.2 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

Select command is used to collect date from the table. We will discuss more on this by adding more commands to this select command. If you have not created the tables then please go to sql introduction page to use the sql commands to create the tables and fill the table with some records (data). Now your table with some data is ready. We will apply select command to our table (name student) and fetch all the records.

```
SELECT * FROM `student`
```

id	name	mark	class
1	John Deo	Four	75
2	Max Ruin	Three	85
3	Arnold	Three	55
4	Krish Star	Four	60
5	John Mike	Four	60
6	Alex John	Four	55
7	My John Rob	Fifth	78
8	Asruid	Five	5
9	Tes Qry	Six	78
10	Big John	Four	55

That's all to get all the records from the table student. We have not placed any restriction here and asked for all the fields with all the records. Now if we want to restrict our results and get only name field from the table.

```
SELECT name FROM `student`
```

This will return only name field from the table. We can ask for more fields and all field names we have to separate by comma. We can include as many field names we want from a table.SELECT name, class FROM `student`

name	class
John Deo	Four
Max Ruin	Three
Arnold	Three
Krish Star	Four
John Mike	Four
Alex John	Four
My John Rob	Fifth
Asruid	Five
Tes Qry	Six
Big John	Four

Now we will go for bit more and restrict the number of records we are getting. We are interested say in only 3 records. Our command should return 3 records only. We will use SQL limit command. This will take two parameters. One is the starting point and other is number of records required. Say we are interested in 3 records starting from beginning. Our command will be

```
SELECT * FROM `student` LIMIT 0,3
```

Related Tutorial

Copy data to new table

SQL Left Join

SQL UNION

This command with LIMIT command will return us 3 records starting from 0 (or first) record. This is very important when we use this command with ORDER BY command. Now let us try to list all the students who have come in first 3 ranks. We are required to list here 3 records who have mark more than the others. The top 3 students we want to display in order of first, second and third. The order we can display are in by default in ascending order but we require the listing should return in descending order so we will get the highest ranked student at the top. Before that let us start with a simple example of ORDER BY command.

```
SELECT * FROM `student` ORDER BY mark
```

This will display the records of students in the order of lowest mark to highest mark.

id	name	class	mark
19	Tinny	Nine	18
17	Tumyu	Six	54
10	Big John	Four	55
22	Reggid	Seven	55
29	Tess Played	Seven	55
6	Alex John	Four	55
3	Arnold	Three	55
5	John Mike	Four	60
4	Krish Star	Four	60
20	Jackly	Nine	65

We will change it to display in reverse order so it will display highest mark at the top and lowest mark at the bottom.

```
SELECT  * FROM `student` ORDER BY `mark` DESC
```

With the addition of command DESC we can change the order of display to keep the highest mark at the top of the list and lowest mark at the bottom of the list. Now let us add the LIMIT command to display only the top 3 records. We already have the list in the order of highest mark to lowest mark so by just limiting the number of records to 3 will give our required top three student records.

SELECT * FROM `student` ORDER BY `mark` DESC LIMIT 0,3

id	name	class	mark
33	Kenn Rein	Six	96
12	Recky	Six	94
32	Binn Rott	Seven	90

This is the SQL query which will display top three students based on the mark they scored. In the next section we will use sql WHERE clause to restrict or filter the records.

5.3 WORKING WITH STRINGS

String functions are used to perform an operation on input string and return an output string. Following are the string functions defined in SQL:

1. ASCII(): This function is used to find the ASCII value of a character.

Syntax: SELECT ascii('t');

Output: 116

2. CHAR_LENGTH(): This function is used to find the length of a word.

Syntax: SELECT char_length('Hello!');

Output: 6

3. CHARACTER_LENGTH(): This function is used to find the length of a line.

Syntax: SELECT CHARACTER_LENGTH('geeks for geeks');

Output: 15

4. CONCAT(): This function is used to add two words or strings.

Syntax: SELECT 'Geeks' || ' ' || 'forGeeks' FROM dual;

Output: ‘GeeksforGeeks’

5. CONCAT_WS(): This function is used to add two words or strings with a symbol as concatenating symbol.

Syntax: SELECT CONCAT_WS('_', 'geeks', 'for', 'geeks');

Output: geeks_for_geeks

6. FIND_IN_SET(): This function is used to find a symbol from a set of symbols.

Syntax: SELECT FIND_IN_SET('b', 'a, b, c, d, e, f');

Output: 2

7. FORMAT(): This function is used to display a number in the given format.

Syntax: Format("0.981", "Percent");

Output: ‘98.10%’

8. INSERT(): This function is used to insert the data into a database.

Syntax: INSERT INTO database (geek_id, geek_name) VALUES (5000, 'abc');

Output: successfully updated

9. INSTR(): This function is used to find the occurrence of an alphabet.

Syntax: INSTR('geeks for geeks', 'e');

Output: 2 (the first occurrence of ‘e’)

Syntax: INSTR('geeks for geeks', 'e', 1, 2);

Output: 3 (the second occurrence of ‘e’)

10. LCASE(): This function is used to convert the given string into lower case.

Syntax: LCASE ("GeeksFor Geeks To Learn");

Output: geeksforgeeks to learn

11. LEFT(): This function is used to SELECT a sub string from the left of given size or characters.

Syntax: SELECT LEFT('geeksforgeeks.org', 5);

Output: geeks

12. LENGTH(): This function is used to find the length of a word.

Syntax: LENGTH('GeeksForGeeks');

Output: 13

13. LOCATE(): This function is used to find the nth position of the given word in a string.

Syntax: SELECT LOCATE('for', 'geeksforgeeks', 1);

Output: 6

14.LOWER(): This function is used to convert the upper case string into lower case.

Syntax: SELECT LOWER('GEEKSFORGEEKS.ORG');

Output: geeksforgeeks.org

15.LPAD(): This function is used to make the given string of the given size by adding the given symbol.

Syntax: LPAD('geeks', 8, '0');

Output: 000geeks

16.LTRIM(): This function is used to cut the given sub string from the original string.

Syntax: LTRIM('123123geeks', '123');

Output: geeks

17. MID(): This function is to find a word from the given position and of the given size.

Syntax: Mid ("geeksforgeeks", 6, 2);

Output: for

18. POSITION(): This function is used to find position of the first occurrence of the given alphabet.

Syntax: SELECT POSITION('e' IN 'geeksforgeeks');

Output: 2

19.REPEAT(): This function is used to write the given string again and again till the number of times mentioned.

Syntax: SELECT REPEAT('geeks', 2);

Output: geeksgeeks

20. REPLACE(): This function is used to cut the given string by removing the given sub string.

Syntax: REPLACE('123geeks123', '123');

Output: geeks

21.REVERSE(): This function is used to reverse a string.

Syntax: SELECT REVERSE('geeksforgeeks.org');

Output: ‘gro.skeegrofskeeg’

22.RIGHT(): This function is used to SELECT a sub string from the right end of the given size.

Syntax: SELECT RIGHT('geeksforgeeks.org', 4);

Output: ‘.org’

23.RPAD(): This function is used to make the given string as long as the given size by adding the given symbol on the right.

Syntax: RPAD('geeks', 8, '0');

Output: ‘geeks000’

24.RTRIM(): This function is used to cut the given sub string from the original string.

Syntax: RTRIM('geeksxyxzyyy', 'xyz');

Output: ‘geeks’

25.SPACE(): This function is used to write the given number of spaces.

Syntax: SELECT SPACE(7);

Output: ‘   ‘

26.STRCMP(): This function is used to compare 2 strings.

If string1 and string2 are the same, the STRCMP function will return 0.

If string1 is smaller than string2, the STRCMP function will return -1.

If string1 is larger than string2, the STRCMP function will return 1.

Syntax: SELECT STRCMP('google.com', 'geeksforgeeks.com');

Output: -1

27. SUBSTR(): This function is used to find a sub string from the a string from the given position.

Syntax:SUBSTR('geeksforgeeks', 1, 5);

Output: ‘geeks’

28. SUBSTRING(): This function is used to find an alphabet from the mentioned size and the given string.

Syntax: SELECT SUBSTRING('GeeksForGeeks.org', 9, 1);

Output: ‘G’

29. SUBSTRING_INDEX(): This function is used to find a sub string before the given symbol.

Syntax: SELECT SUBSTRING_INDEX('www.geeksforgeeks.org', '.', 1);

Output: ‘www’

30.TRIM(): This function is used to cut the given symbol from the string.

Syntax: TRIM(LEADING '0' FROM '000123');

Output: 123

31.UCASE(): This function is used to make the string in upper case.

Syntax: UCASE ("GeeksForGeeks");

Output:GEEKSFORGEEKS

5.4 DATE AND TIME FUNCTIONS

In SQL, dates are complicated for newbies, since while working with database, the format of the date in table must be matched with the input date in order to insert. In various scenarios instead of date, datetime (time is also involved with date) is used.

In MySql the default date functions are:

NOW(): Returns the current date and time. Example:

SELECT NOW();

Output:

2017-01-13 08:03:52

CURDATE(): Returns the current date. Example:

SELECT CURDATE();

Output:

2017-01-13

CURTIME(): Returns the current time. Example:

SELECT CURTIME();

Output:

08:05:15

DATE(): Extracts the date part of a date or date/time expression. Example:For the below table named 'Test'

Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

SELECT Name, DATE(BirthTime) AS BirthDate FROM Test;

Output:

```
Name   BirthDate
Pratik 1996-09-26
```

EXTRACT(): Returns a single part of a date/time. Syntax:

EXTRACT(unit FORM date);

Check your Progress

1. Mention the use of SELECT statement and also Mention it clauses.
2. Name any 6 String Operations in SQL.
3. Explain the syntax with example of MID() and POSITION()
4. Write short notes on Date and Time Functions.
5. What is the difference between SUBSTR() and SUBSTRING()

5.5 ANSWERS TO CHECK YOUR PROGRESS

1. MySQL is the most popular open source SQL database management system (DBMS). A fast, reliable, easy-to-use, multi-user multi-threaded relational database system.
2. The string operations in SQL are
   i. ASCII()
   ii. CHAR_LENGTH()
   iii. CHARACTER_LENGTH()
   iv. CONCAT()
   v. FIND_IN_SET()
   vi. REPLACE()
3. Different syntaxes are
   i. MID(): This function is to find a word from the given position and of the given size.
      - Syntax: Mid ("geeksforgeeks", 6, 2);
      - Output: for
   ii. POSITION(): This function is used to find position of the first occurrence of the given alphabet.
      - Syntax: SELECT POSITION('e' IN 'geeksforgeeks');
      - Output: 2
4. The Date and Time Functions are given below.
   i. NOW(): Returns the current date and time. Example: SELECT NOW();

      Output:

      2017-01-13 08:03:52

   ii. CURDATE(): Returns the current date. Example: SELECT CURDATE();

      Output:

      2017-01-13

   iii. CURTIME(): Returns the current time. Example: SELECT CURTIME();

      Output:

      08:05:15

5. Two of them are defined below.

i. SUBSTR(): This function is used to find a sub string from the a string from the given position.
Syntax:SUBSTR('geeksforgeeks', 1, 5);
Output: ‘geeks’

ii. SUBSTRING(): This function is used to find an alphabet from the mentioned size and the given string.

NOTES

5.6 SUMMARY

- The SELECT statement is used to select data from a database. The statement begins with the SELECT keyword.
- String functions are used to perform an operation on input string and return an output string.
- In SQL, dates are complicated for newbies, since while working with database, the format of the date in table must be matched with the input date in order to insert.
- With the addition of command DESC we can change the order of display to keep the highest mark at the top of the list and lowest mark at the bottom of the list.

5.7 KEYWORDS

SUBSTR(): This function is used to find a sub string from the a string from the given position.

RTRIM(): This function is used to cut the given sub string from the original string.

POSITION(): This function is used to find position of the first occurrence of the given alphabet

5.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer questions:

1. Write Appropriate syntax to select first 3 record from table.
2. Write Appropriate syntax to sort the table (student mark) with Rank
3. What is difference between TRIM() and RTRIM()
4. What are the units Considered in Date and Time?

Long Answer questions:

1. Briefly Explain Strings and its Functions with example.
2. Explain About Record Selection Technology.

5.9 FURTHER READINGS

Rémy Card, Eric Dumas, and Franck Mével. The Linux kernel book. John Wiley & Sons, Inc., 2003.

Steve Suchring. MySQL BBible. John Wiley, 2002.

Rasmus Lerdorf and Levin Tatroe. Programming PHP. " O'Reilly Media, Inc., 2002.

Wesley J. Chun. Core Python Programming. Prentice Hall, 2001.

Martin C. Brown. Perl: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Steven Holzner. PHP: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Vikram Vaswani. MySQL: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

UNIT 6
WORKING WITH SQL

NOTES

Structure

6.0 INTRODUCTION

This unit helps the user to use and query with MySQL by sorting the queries using the sorting commands and the commands used with the web environment for development. It also explains the metadata and how to generate the summary from the tables.

6.1 OBJECTIVE

This unit explains to users the following concepts

- Learn the sorting queries
- To understand metadata
- Work with sequences
- Working with web environment

6.2 SORTING QUERY RESULTS

This MySQL tutorial explains how to use the MySQL ORDER BY clause with syntax and examples.

Description

The MySQL ORDER BY clause is used to sort the records in your result set.

Syntax

The syntax for the ORDER BY clause in MySQL is:

```
SELECT expressions

FROM tables

[WHERE conditions]

ORDER BY expression [ ASC | DESC ];
```

Parameters or Arguments

Expressions

The columns or calculations that you wish to retrieve.

Tables

The tables that you wish to retrieve records from. There must be at least one table listed in the FROM clause.

WHERE conditions

Optional. The conditions that must be met for the records to be selected.

ASC

Optional. It sorts the result set in ascending order by expression (default, if no modifier is provider).

DESC

Optional. It sorts the result set in descending order by expression.

If the ASC or DESC modifier is not provided in the ORDER BY clause, the results will be sorted by expression in ascending order. This is equivalent to ORDER BY expression ASC. The ORDER BY clause can be used in a SELECT statement, SELECT LIMIT statement, and DELETE LIMIT statement in MySQL.

Example - Sorting without using ASC/DESC attribute

The MySQL ORDER BY clause can be used without specifying the ASC or DESC modifier. When this attribute is omitted from the ORDER BY clause, the sort order is defaulted to ASC or ascending order.

For example:

```
SELECT city

FROM customers

WHERE customer_name = 'Apple'

ORDER BY city;
```

This MySQL ORDER BY example would return all records sorted by the city field in ascending order and would be equivalent to the following ORDER BY clause:

```
SELECT city

FROM customers

WHERE customer_name = 'Apple'

ORDER BY city ASC;
```

Sorting Subsets of a Table

You don't want to sort an entire table, just part of it. Add a WHERE clause that selects only the records you want to see.

```
mysql> SELECT trav_date, miles FROM driver_log WHERE name =
'Henry'                                         ORDER BY trav_date
```

Sorting and NULL Values
To sort columns that may contain NULL values

```
mysql> SELECT NULL = NULL;
```

Sorting by Calendar Day

To sort by day of the calendar year. Sort using the month and day of a date, ignoring the year. Sorting in calendar order differs from sorting by date.

```
mysql> SELECT date, description FROM event ORDER BY date;
```

6.3 GENERATING SUMMARY

Database systems are useful for data storage and retrieval, but can also summarize your data in more concise forms. Summaries are useful when you want the overall picture, not the details. They're more readily understood than a long list of records. They enable you to answer questions such as "How many?" or "What is the total?" or "What is the range of values?" If you run a business, you may want to know how many customers you have in each state, or how much sales volume you generate each month.

The preceding examples include two common summary types: counting summaries and content summaries. The first (the number of customer records per state) is a counting summary. The content of each record is important only for purposes of placing it into the proper group or category for counting. Such summaries are essentially histograms, where you sort items into a set of bins and count the number of items in each bin. The second example (sales volume per month) is a content summary, in which sales totals are based on sales values in order records.

Another summary type produces neither counts nor sums, but simply a list of unique values. This is useful if you care which values are present rather than how many of each there are. To determine the states in which you have customers, you need a list of the distinct state names contained in the records, not a list consisting of the state value from every record.

Using aggregate function we can achieve summary of values. Aggregate functions are COUNT (), MIN (), MAX (), SUM. (), AVG () and GROUP BY clause to group the rows into subsets and obtain an aggregate value for each one.To getting a list of unique values, use SELECT DISTINCT rather than SELECT.Using COUNT (DISTINCT) - To count how many distinct values there are

Summarizing with COUNT( )

To count the number of rows in an entire table or that match particular conditions, use the

COUNT( ) function.

For example,

```
mysql> SELECT COUNT(*) FROM emp_tab;
```

Summarizing with MIN( ) and MAX( )

Finding smallest or largest values in an entire table, use the MIN () and MAX () function. For example,

```
mysql> SELECT MIN(Sal) AS low, MAX(sal) AS

high FROM emp_tab;
```

Summarizing with SUM( ) and AVG( )

SUM( ) and AVG( ) produce the total and average (mean) of a set of values:

For Example,

```
mysql> SELECT SUM(rno) AS 'No-Of Emp', AVG(sal) AS 'Avg Sal' FROM
```

Sorting Query Results

SQL SELECT command to fetch 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 a query doesn't come out in the order you want

6.4 WORKING WITH METADATA

Metadata is data about data.

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

6.4.1 Types of Metadata

- Information about the result of queries-This includes the number of records affected by any SELECT, UPDATE or DELETE statement.

- Information about the tables and databases − This includes information pertaining to the structure of the tables and the databases.

- Information about the MySQL server − This includes the status of the database server, version number, etc.

6.4.2 Obtaining Metadata with SHOW

MySQL provides a SHOW statement that displays many types of database metadata. SHOW is helpful for keeping track of the contents of your databases and reminding yourself about the structure of your tables. The following examples demonstrate a few uses for SHOW statements.

List the databases you can access:

SHOW DATABASES;

Display the CREATE DATABASE statement for a database:

SHOW CREATE DATABASE db_name;

List the tables in the default database or a given database:

SHOW TABLES;

SHOW TABLES FROM db_name;

6.4.3 Obtaining Metadata with INFORMATION_SCHEMA

Another way to obtain information about databases is to access the INFORMATION_SCHEMA database. INFORMATION_SCHEMA is based on the SQL standard. That is, the access mechanism is standard, even though some of the content is MySQL-specific. This makes INFORMATION_SCHEMA more portable than the various SHOW statements, which are entirely MySQL-specific.

You can think of INFORMATION_SCHEMA as a virtual database in which the tables are views for different kinds of database metadata. To see what tables INFORMATION_SCHEMA contains, use SHOW TABLES.

6.5 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.

6.5.1Creating a Sequence

CREATE SEQUENCE sequence_name

```
[INCREMENT BY #]

[START WITH #]

[MAXVALUE # | NOMAXVALUE]

[MINVALUE # | NOMINVALUE]

[CYCLE | NOCYCLE]
```

6.5.2 Dropping a Sequence

DROP SEQUENCE my_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 sequence

NEXTVAL: Returns the next value from the sequence.

CURVAL: 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.')

6.6 MY SQL 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

6.6.1 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

- HTML documents describe web pages
- HTML documents contain HTML tags and plain text
- HTML documents are also called web pages

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), is executed by the server (Web Server), and the page that is sent to the browser is produced by the serve-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.

Check your Progress

1. Which clause is used to Sort the Records? What is the Use of where Condition?
2. Write short note on Client-Side scripting.
3. Where SUMMARY Generating is used?
4. What are the types of Metadata?
5. What is the use of SHOW in SQL?

6.7 ANSWERS TO CHECK YOUR PROGRESS

1. The MySQL ORDER BY clause is used to sort the records in your result set.
   The conditions that must be met for the records to be selected.
2. 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
3. Summaries are useful when you want the overall picture, not the details. They’re more readily understood than a long list of records. They enable you to answer questions such as “How many?” or “What is the total?” or “What is the range of values?”

4. The types of metadata are
    - Information about the result of queries
    - Information about the tables and databases
    - Information about the MySQL server
5. SHOW statement that displays many types of database metadata. SHOW is helpful for keeping track of the contents of your databases and reminding yourself about the structure of your tables.
6.

NOTES

6.8 SUMMARY

- Database systems are useful for data storage and retrieval, but can also summarize your data in more concise forms.
- Information about the result of queries-This includes the number of records affected by any SELECT, UPDATE or DELETE statement.
- 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.
- MySQL makes it easier to provide dynamic rather than static content.

6.9 KEYWORDS

Sorting Query Results: SQL SELECT command to fetch 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.

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.

6.10 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer questions:

1. How to create a sequence?
2. What is Dropping a Sequence and what are their Functions.
3. Write short notes on HTML.
4. What is the Difference Between Static and Dynamic web Pages?
5. Write short nodes on Server-side Scripting.

Long Answer questions:

1. Explain the Process of Basic Web Page Generation.
2. Explain briefly about Generating Summary with Example.

6.11 FURTHER READINGS

Rémy Card, Eric Dumas, and Franck Mével. The Linux kernel book. John Wiley & Sons, Inc., 2003.

Steve Suchring. MySQL BBible. John Wiley, 2002.

Rasmus Lerdorf and Levin Tatroe. Programming PHP. " O'Reilly Media, Inc., 2002.

Wesley J. Chun. Core Python Programming. Prentice Hall, 2001.

Martin C. Brown. Perl: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Steven Holzner. PHP: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Vikram Vaswani. MySQL: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

BLOCK – III
OPEN SOURCE PROGRAMMING LANGUAGE PHP

UNIT 7
PHP

NOTES

Structure

7.0 INTRODUCTION

Open source programming languages plays an pivotal role in creation of real time applications as the software is freely available and it can be used with good packages. The worked modules can be shared for later use of developers. In this unit, the open source programming language PHP is explained with the basics of programming

7.1 OBJECTIVE

After reading this unit you will be able to

- Understand PHP
- Learn the web environment
- Basics of PHP

7.2 PHP

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994. PHP is a recursive acronym for "PHP: Hypertext Pre-processor". PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites. It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.

PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the UNIX side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time. PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.

7.2.1 Common uses of PHP

PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them. PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user. You add, delete, modify elements within your database through PHP. Using PHP, you can restrict users to access some pages of your website. It can encrypt data.

7.2.2 Characteristics of PHP

Five important characteristics make PHP's practical nature possible

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

7.3 PROGRAMMING IN WEB ENVIRONMENT

PHP programs are written using a text editor, such as Notepad, Simple Text, or vi, 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.

PHP not only allows HTML pages to be created on the fly, but it is invisible to your web site visitors. The only thing they see when they view the source of your code is the resulting HTML output. In this respect, PHP gives you a bit more security by hiding your programming logic. HTML can also be written inside the PHP code of your page, which allows you to format text while keeping blocks of code together. This will also help you write organized, efficient code, and the browser (and, more importantly, the person viewing the site) won’t know the difference.

PHP can also be written as a standalone program with no HTML at all. This is helpful for storing your connection variables, redirecting your visitors to another page of your site, or performing other functions

7.4 PHP VARIABLES

NOTES

The main way to store information in the middle of a PHP program is by using a variable. Here are the most important things to know about variables in PHP.All variables in PHP are denoted with a leading dollar sign ($).The value of a variable is the value of its most recent assignment. Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right. Variables can, but do not need, to be declared before assignment. Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.

Variables used before they are assigned have default values. PHP does a good job of automatically converting types from one to another when necessary. PHP variables are Perl-like.

7.4.1 Variable Scope

Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types −

- Local variables
- Function parameters
- Global variables
- Static variables
- Variable Naming

Rules for naming a variable is:

- Variable names must begin with a letter or underscore character.
- A variable name can consist of numbers, letters, underscores but you cannot use characters like + , - , % , (,) . & , etc
- There is no size limit for variables.

7.5 CONSTANTS

A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default, a constant is case-sensitive. By convention, constant identifiers are always uppercase. A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. If you have defined a constant, it can never be changed or undefined.

To define a constant you have to use define() function and to retrieve the value of a constant, you have to simply specifying its name. Unlike with variables, you do not need to have a constant with a $. You can also use the function constant() to read a constant's value if you wish to obtain the constant's name dynamically.

constant() function

As indicated by the name, this function will return the value of the constant. This is useful when you want to retrieve value of a constant, but you do not know its name, i.e. It is stored in a variable or returned by a function.

constant() example

```
<?php

  define("MINSIZE", 50);

     echo MINSIZE;

  echo constant("MINSIZE"); // same thing as the previous line

?>
```

Only scalar data (boolean, integer, float and string) can be contained in constants.

Differences between constants and variables are 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. Valid and invalid constant names

```
// Valid constant names

define("ONE",     "first thing");

define("TWO2",    "second thing");

define("THREE_3", "third thing");

// Invalid constant names

define("2TWO",    "second thing");

define("__THREE__", "third value");
```

7.5.1 PHP Magic constants

PHP provides a large number of predefined constants to any script which it runs. There are five magical constants that change depending on where they are used. For example, the value of LINE depends on the line that it's used on in your script. These special constants are case-insensitive and are as follows.

A few "magical" PHP constants are given below

LINE- The current line number of the file.

FILE- The full path and filename of the file. If used inside an include, the name of the included file is returned. Since PHP 4.0.2, __FILE__ always contains an absolute path whereas in older versions it contained relative path under some circumstances.

FUNCTION- The function name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the function name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.

CLASS-The class name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the class name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.

METHOD- The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).

7.6 DATA TYPES

PHP has a total of eight data types which we use to construct our variables

Integers − are whole numbers, without a decimal point, like 4195.

Doubles − are floating-point numbers, like 3.14159 or 49.1.

Booleans − have only two possible values either true or false.

NULL − is a special type that only has one value: NULL.

Strings − are sequences of characters, like 'PHP supports string operations.'

Arrays − are named and indexed collections of other values.

Objects − are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.

Resources − are special variables that hold references to resources external to PHP (such as database connections).

The first five are simple types, and the next two (arrays and objects) are compound - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot. We will explain only simple data type in this chapters. Array and Objects will be explained separately.

7.6.1 Integers

They are whole numbers, without a decimal point, like 4195. They are the simplest type .they correspond to simple whole numbers, both positive and negative. Integers can be assigned to variables, or they can be used in expressions, like so −

```
$int_var = 12345;
```

```
$another_int = -12345 + 12345;
```

Integer can be in decimal (base 10), octal (base 8), and hexadecimal (base 16) format. Decimal format is the default, octal integers are specified with a leading 0, and hexadecimals have a leading 0x.For most common platforms, the largest integer is (2**31 . 1) (or 2,147,483,647), and the smallest (most negative) integer is . (2**31 . 1) (or .2,147,483,647).

NOTES

7.6.2 Doubles

They like 3.14159 or 49.1. By default, doubles print with the minimum number of decimal places needed. For example, the code −

Live Demo

```
<?php
  $many = 2.2888800;
  $many_2 = 2.2111200;
  $few = $many + $many_2;
    print("$many + $many_2 = $few <br>");
?>
```

It produces the following browser output −

```
2.28888 + 2.21112 = 4.5
```

7.6.3 Boolean

They have only two possible values either true or false. PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE, which can be used like so −

```
if (TRUE)
  print("This will always print<br>");
else
  print("This will never print<br>");
```

Interpreting other types as Booleans. Here are the rules for determine the "truth" of any value not already of the Boolean type −If the value is a number, it is false if exactly equal to zero and true otherwise. If the value is a string, it is false if the string is empty (has zero characters) or is the string "0", and is true otherwise. Values of type NULL are always false.

If the value is an array, it is false if it contains no other values, and it is true otherwise. For an object, containing a value means having a member variable that has been assigned a value. Valid resources are true (although some functions that return resources when they are successful will return FALSE when unsuccessful).Don't use double as Booleans.

Each of the following variables has the truth value embedded in its name when it is used in a Boolean context.

```
$true_num = 3 + 0.14159;

$true_str = "Tried and true"

$true_array[49] = "An array element";

$false_array = array();

$false_null = NULL;

$false_num = 999 - 999;

$false_str = "";
```

7.6.4 NULL

NULL is a special type that only has one value: NULL. To give a variable the NULL value, simply assign it like this −

```
$my_var = NULL;
```

The special constant NULL is capitalized by convention, but actually it is case insensitive; you could just as well have typed −

```
$my_var = null;
```

A variable that has been assigned NULL has the following properties −It evaluates to FALSE in a Boolean context. It returns FALSE when tested with IsSet() function.

7.6.5 Strings

They are sequences of characters, like "PHP supports string operations". Following are valid examples of string

```
$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
```

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

Live Demo

```
<?php

   $variable = "name";

   $literally = 'My $variable will not print!';
```

```
    print($literally);
  print "<br>";
    $literally = "My $variable will print!";
  print($literally);
?>
```

This will produce following result −

```
My $variable will not print!
My name will print
```

7.6.6 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 its easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID c which is called array index.

- **Numeric array** − An array with a numeric index. Values are stored and accessed in linear fashion.
- **Associative array** − An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
- **Multidimensional array** − An array containing one or more arrays and values are accessed using multiple indices

7.7 OPERATORS

PHP language supports following type of operators.

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

7.7.1Arithmetic Operators

There are following arithmetic operators supported by PHP language .Assume variable A holds 10 and variable B holds 20 then

Show Examples

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

7.7.2 Comparison Operators

There are following comparison operators supported by PHP languageAssume variable A holds 10 and variable B holds 20 then

Show Examples

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then	(A > B) is not true.

	condition becomes true.	
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

7.7.3 Logical Operators

There are following logical operators supported by PHP language Assume variable A holds 10 and variable B holds 20 then −Show Examples

NOTES

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then condition becomes true.	(A and B) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then condition becomes true.	(A or B) is true.
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.

7.7.4 Assignment Operators

There are following assignment operators supported by PHP language −Show Examples

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the	C += A is equivalent to C = C + A

	result to left operand	
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

NOTES

7.7.5 Conditional Operator

There is one more operator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation. The conditional operator has this syntax −

Show Examples

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

7.7.6 Operators Categories

All the operators we have discussed above can be categorised into following categories

- Unary prefix operators, which precede a single operand.
- Binary operators, which take two operands and perform a variety of arithmetic and logical operations.
- The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.
- Assignment operators, which assign a value to a variable.

7.7.7 Precedence of PHP Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator

For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedence than + so it first get multiplied with 3*2 and then adds into 7. Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Unary	! ++ --	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=	Right to left

7.8 STATEMENT

A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.Prepared statements basically work like this:

Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it

Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values.Compared to executing SQL statements directly, prepared statements have three main advantages:

NOTES

- Prepared statements reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times).
- Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query.Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped.
- If the original statement template is not derived from external input, SQL injection cannot occur.

Check your Progress

1. Write short notes on PHP?
2. What are the Characteristics of PHP?
3. What is Variable Scope? Mention in types.
4. What are Constants?

7.9. ANSWERS TO CHECK YOUR PROGRESS

1. PHP started out as a small open source project that evolved as more and more people found out how useful it was. PHP is a server-side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.

2. The characteristics of PHP are
   - Simplicity
   - Efficiency
   - Security
   - Flexibility
   - Familiarity

3. Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types:

- Local variables
- Function parameters
- Global variables
- Static variables
- Variable Naming

4. A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default, a constant is case-sensitive.

7.10 SUMMARY

- 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 its easy to define an array of 100 length.
- Resources are special variables that hold references to resources external to PHP (such as database connections).
- To define a constant you have to use define() function and to retrieve the value of a constant, you have to simply specifying its name.

7.11 KEYWORDS

Numeric array − An array with a numeric index. Values are stored and accessed in linear fashion

Boolean: They have only two possible values either true or false. PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE,

Resources: They are special variables that hold references to resources external to PHP (such as database connections).

Constants: A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script.

7.12 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer questions:

1. List some of the PHP variables?
2. Write short notes on constants.
3. What are the different data types?
4. Write short notes on PHP Magical constants.

Long Answer questions:

1. Explain the type of operators supported in PHP.
2. Explain briefly about the eight data types.

7.13. FURTHER READINGS

Rémy Card, Eric Dumas, and Franck Mével. The Linux kernel book. John Wiley & Sons, Inc., 2003.

Steve Suchring. MySQL BBible. John Wiley, 2002.

Rasmus Lerdorf and Levin Tatroe. Programming PHP. " O'Reilly Media, Inc., 2002.

Wesley J. Chun. Core Python Programming. Prentice Hall, 2001.

Martin C. Brown. Perl: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Steven Holzner. PHP: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Vikram Vaswani. MySQL: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

UNIT 8
FUNCTIONS

Structure

8.0 INTRODUCTION

The programming of PHP with functions helps for the manipulation of data in a much easier manner. The file handling functions and the storage of data is helpful for the usage of large databases. IN this unit, the functions of PHP and its usage with files and strings are explained.

8.1 OBJECTIVE

This unit helps the user to understand the following concepts

- Function
- Strings
- Arrays
- OOP
- Regular Expression
- File Handling

8.2 FUNCTIONS

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value. You already have seen many functions like fopen() and fread() etc. They are built-in functions but PHP gives you option to create your own functions as well. There are two parts which should be clear to you

- Creating a PHP Function

- Calling a PHP Function

In fact you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement. Please refer to PHP Function Reference for a complete set of useful functions.

8.2.1 Creating PHP Function

It's very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called writeMessage() and then calls it just after creating it. Note that while creating a function its name should start with keyword function and all the PHP code should be put inside { and } braces as shown in the following example below −

Live Demo

```
<html>

    <head>

    <title>Writing PHP Function</title>

  </head>

    <body>

        <?php

      /* Defining a PHP Function */

      function writeMessage() {

        echo "You are really a nice person, Have a nice time!";

      }

      /* Calling a PHP Function */

      writeMessage();

    ?>

      </body>

</html>
```

This will display following result −

```
You are really a nice person, Have a nice time!
```

8.2.2 PHP Functions with Parameters

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters your like. These parameters work like variables inside your function. Following example takes two integer parameters and add them together and then print them.

Live Demo

```
<html>
    <head>
    <title>Writing PHP Function with Parameters</title>
  </head>
    <body>
      <?php
      function addFunction($num1, $num2) {
        $sum = $num1 + $num2;
        echo "Sum of the two numbers is : $sum";
      }
       addFunction(10, 20);
    ?>
      </body>
</html>
```

This will display following result −

```
Sum of the two numbers is : 30
```

8.2.3 Passing Arguments 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.

Following example depicts both the cases.

Live Demo

```
<html>
    <head>
    <title>Passing Argument by Reference</title>
  </head>
    <body>
        <?php
```

```
      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 />";
    ?>
     </body>
</html>
```

This will display following result −

```
Original Value is 10
Original Value is 16
```

8.2.4 PHP Functions Returning Value

A function can return a value using the return statement in conjunction with a value or object. return stops the execution of the function and sends the value back to the calling code.You can return more than one value from a function using return array(1,2,3,4).Following example takes two integer parameters and add them together and then returns their sum to the calling program. Note that return keyword is used to return a value from a function.

Live Demo

```
<html>
    <head>
    <title>Writing PHP Function which returns value</title>
  </head>
    <body>
      <?php
```

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

This will display following result –Ret

urned value from the function : 30

8.2.5 Setting Default Values for Function Parameters

You can set a parameter to have a default value if the function's caller doesn't pass it. Following function prints NULL in case use does not pass any value to this function.

Live Demo

```
<html>
      <head>
      <title>Writing PHP Function which returns value</title>
   </head>
      <body>
            <?php
         function printMe($param = NULL) {
            print $param;
         }
          printMe("This is test");
         printMe();
      ?>
      </body>
</html>
```

This will produce following result

This is test

8.2.6 Dynamic Function Calls

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself. Following example depicts this behaviour.

Live Demo

```
<html>
    <head>
    <title>Dynamic Function Calls</title>
  </head>
    <body>
     <?php
      function sayHello() {
        echo "Hello<br />";
      }
      $function_holder = "sayHello";
      $function_holder();
    ?>
    </body>
</html>
```

This will display following result

```
Hello
```

8.3 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 its easy to define an array of 100 length. is called array index.

- Numeric array − An array with a numeric index. Values are stored and accessed in linear fashion.
- Associative array − An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
- Multidimensional array − An array containing one or more arrays and values are accessed using multiple indices

8.3.1 Numeric Array

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default array index starts from zero.

Example

Following is the example showing how to create and access numeric arrays.Here we have used array() function to create array. This function is explained in function reference.

Live Demo

```
<html>
  <body>
      <?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 />";
      }
    ?>
    </body>
</html>
```

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
```

8.3.2 Associative Arrays

The associative arrays are very similar to numeric arrays in term 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. To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

Example

Live Demo

```
<html>
  <body>
    <?php
      /* First method to associate create array. */
      $salaries = array("mohammad"=>2000,"qadir"=>1000,"zara"=>500);
      echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
      echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
      echo "Salary of zara is ".  $salaries['zara']. "<br />";
       /* Second method to create array. */
      $salaries['mohammad'] = "high";
      $salaries['qadir'] = "medium";
      $salaries['zara'] = "low";
      echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
      echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
      echo "Salary of zara is ".  $salaries['zara']. "<br />";
    ?>
    </body>
</html>
```

This will produce the following result −

```
Salary of mohammad is 2000
Salary of qadir is 1000
Salary of zara is 500
Salary of mohammad is high
Salary of qadir is medium
```

Salary of zara is low

8.4 OBJECT ORIENTED CONCEPTS

Before we go in detail, let’s define important terms related to Object Oriented Programming.

Class − this is a programmer-defined data type, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.

Object − an individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.

Member Variable − these are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions. These variables are called attribute of the object once an object is created.

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 function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.

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 subclass or derived class.

Polymorphism − this is an object oriented concept where same function can be used for different purposes. For example function name will remain same but it takes different number of arguments and can do different task.

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 (abstracted).

Encapsulation − refers to a concept where we encapsulate all the data and member functions together to form an object.

Constructor − refers to a special type of function which will be called automatically whenever there is an object formation from a class.

Destructor − refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

8.5 STRING MANIPULATION AND REGULAR EXPRESSION

Strings that are delimited by double quotes (as in "this") are pre-processed in both the following two ways by PHP. Certain character sequences beginning with backslash (\) are replaced with special characters. Variable names (starting with $) are replaced with string representations of their values.The escape-sequence replacements are

\n is replaced by the newline character

\r is replaced by the carriage-return character

\t is replaced by the tab character

\$ is replaced by the dollar sign itself ($)

\" is replaced by a single double-quote (")

\\ is replaced by a single backslash (\)

8.5.1 String Concatenation Operator

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

Live Demo

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

This will produce the following result −

```
Hello World 1234
```

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string.Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

Using the strlen() function

The strlen() function is used to find the length of a string.Let's find the length of our string "Hello world!"

Live Demo

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

```
?>
```

This will produce the following result −

```
12
```

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string)

Using the 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.Let's see if we can find the string "world" in our string

Live Demo

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

This will produce the following result −

```
6
```

As you see the position of the string "world" in our string is position 6. The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1

Regular Expressions

Regular expressions are nothing more than a sequence or pattern of characters it. They provide the foundation for pattern-matching functionality. Using regular expression you can search a particular string inside a another string, you can replace one string by another string and you can split a string into many chunks.PHP offers functions specific to two sets of regular expression functions, each corresponding to a certain type of regular expression. You can use any of them based on your comfort.

- POSIX Regular Expressions
- PERL Style Regular Expressions
- POSIX Regular Expressions

The structure of a POSIX regular expression is not dissimilar to that of a typical arithmetic expression: various elements (operators) are combined to form more complex expressions.

The simplest regular expression is one that matches a single character, such as g, inside strings such as g, haggle, or bag. Let’s give explanation for few concepts being used in POSIX regular expression. After that we will introduce you with regular expression related functions.

Brackets

Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

[0-9]

It matches any decimal digit from 0 through 9.

Quantifiers

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character having a specific connotation. The +, *, ?, {int. range}, and $ flags all follow a character sequence.

Sr.No Expression & Description

1 p+ It matches any string containing at least one p

.

a p followed by zero or more instances of the sequence php.

Predefined Character Ranges

For your programming convenience several predefined character ranges, also known as character classes, are available. Character classes specify an entire range of characters, for example, the alphabet or an integer set

Sr.No Expression & Description

1 [[:alpha:]] It matches any string containing alphabetic characters aA through zZ.

PHP's Regexp POSIX Functions

PHP currently offers seven functions for searching strings using POSIX-style regular expressions

Sr.No Function & Description

1 ereg() The ereg() function searches a string specified by string for a string specified by pattern, returning true if the pattern is found, and false otherwise.

8.6 FILE HANDLING AND DATA STORAGE

It 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

This is a strong point of PHP which helps in creating functions, headers, footers, or elements that can be reused on multiple pages. This will help developers to make it easy to change the layout of complete website with minimal effort. If there is any change required then instead of changing thousands of files just change included file.

8.6.1 The include() Function

The include() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file then, the include() function generates a warning but the script will continue execution. Assume you want to create a common menu for your website. Then create a file menu.php with the following content.

```
<a href="http://www.tutorialspoint.com/index.htm">Home</a> -

<a href="http://www.tutorialspoint.com/ebxml">ebXML</a> -

<a href="http://www.tutorialspoint.com/ajax">AJAX</a> -

<a href="http://www.tutorialspoint.com/perl">PERL</a> <br />
```

Now create as many pages as you like and include this file to create header. For example now your test.php file can have following content.

```
<html>

  <body>

      <?php include("menu.php"); ?>

    <p>This is an example to show how to include PHP file!</p>

      </body>

</html>
```

It will produce the following result

Include

8.6.2 The require() Function

The require() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file then the require() function generates a fatal error and halt the execution of the script. So there is no difference in require() and include() except they handle error conditions. It is recommended to use the require() function instead of include(), because scripts should not continue executing if files are missing or misnamed.

You can try using above example with require() function and it will generate same result. But if you will try following two examples where file does not exist then you will get different results.

```
<html>
  <body>
       <?php include("xxmenu.php"); ?>
     <p>This is an example to show how to include wrong PHP file!</p>
     </body>
</html>
```

This will produce the following result −

```
This is an example to show how to include wrong PHP file!
```

Now lets try same example with require() function.

```
<html>
  <body>
      <?php require("xxmenu.php"); ?>
      <p>This is an example to show how to include wrong PHP file!</p>
     </body>
</html>
```

This time file execution halts and nothing is displayed.

Check your Progress

1. What arethe Concepts of PHP?
2. What are Arrays?
3. What is Multidimensional Array**?**
4. What areConstructor Functions?
5. What are Function Overriding?

8.7 ANSWERS TO CHECK YOUR PROGRESS

1. The Concepts of PHP are:
    - Function
    - Strings
    - Arrays
    - OOP
    - Regular Expression
    - File Handling
2. 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’s easy to define an array of 100 length. is called array index.
3. 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.

4. Constructor Functions are special type of functions which are called automatically whenever an object is created. So, we take full advantage of this behaviour, by initializing many things through constructor functions. PHP provides a special function called construct () to define a constructor.
5. Function definitions in child classes override definitions with the same name in parent classes. In a child class, we can modify the definition of a function inherited from parent class.

8.8 SUMMARY

- A multi-dimensional array each element in the main array can also be an array.
- Object is an individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.
- Polymorphism is an object oriented concept where same function can be used for different purposes. For example function name will remain same but it takes different number of arguments and can do different task.
- Overloading is 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.

8.9 KEYWORDS

Function: A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

Associative array: An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.

Multidimensional array: An array containing one or more arrays and values are accessed using multiple indices

Inheritance: When a class is defined by inheriting existing function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.

Parent class: A class that is inherited from by another class. This is also called a base class or super class.

8.10 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer questions:

1. What are Interfaces?
2. What is String Manipulation?
3. Explain about Regular Expression?
4. What is the include () function?
5. What is the require () function?

Long Answer questions:

1. Explain briefly about Object Oriented Concepts?
2. Explain about Arrays and its Types?
3. Explain about Functions?

8.11. FURTHER READINGS

Rémy Card, Eric Dumas, and Franck Mével. The Linux kernel book. John Wiley & Sons, Inc., 2003.

Steve Suchring. MySQL BBible. John Wiley, 2002.

Rasmus Lerdorf and Levin Tatroe. Programming PHP. " O'Reilly Media, Inc., 2002.

Wesley J. Chun. Core Python Programming. Prentice Hall, 2001.

Martin C. Brown. Perl: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Steven Holzner. PHP: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Vikram Vaswani. MySQL: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

UNIT 9
PHP AND MySQL DATABASE

Structure

9.0 INTRODUCTION

The open source software PHP programming is combined with other backend open source available software's for creation of real time applications. This unit covers the advance concepts of PHP for better programming skills in connection of PHP with other languages.

9.1 OBJECTIVE

This unit helps you to

- Understand PHP and SQl
- Combine PHP with PDAP
- PHP connectivity and error handling with debugging

9.2 PHP AND MYSQL

MySQL easily fits into server-side programming languages, through a domain. Before your MySQL functions will be recognizable, make sure to enable MySQL in your php.ini file. You can use MySQL commands within PHP code almost as seamlessly as you do with HTML. Numerous PHP functions work specifically with MySQL to make your life easier.

Some of the more commonly used functions are:

- mysql_connect([$host[, $username[, $password]]]): Connects to the MySQL server and returns a resource which is used to reference the connection.

- mysql_select_db($database[, $resource]): Equivalent to the MySQL command USE and sets the active database.

- mysql_query($query[, $resource]): Used to send any MySQL command to the database server. In the case of SELECT queries, a reference to the result set will be returned.

- mysql_fetch_array($result): Return a row of data from the query's result set as an associative array, numeric array or both.

  - mysql_fetch_assoc($result): Return a row of data from the query's result set as an associative array.

- mysql_error([$resource]): Shows the error message generated by the previous query.

9.2.1 Connecting to the MySQL Server

Before you can do anything with MySQL, you must first connect to the MySQL server using your specific connection values. Connection variables consist of the following parameters: Hostname: In our case, this is localhost because everything has been installed locally. You will need to change this to whatever host is acting as your MySQL server, if MySQL is not on the same server.

Username and password: This is to authenticate securely over server end. You issue this connection command with the PHP function called mysql_connect(). As with all of your PHP/MySQL statements, you can either put the information into variables or leave it as text in your MySQL query.

Here's how you would do it with variables:

```
$host = ‘localhost’;

$user = ‘user_name’;

$pass = ‘password’;

$db = mysql_connect($host, $user, $pass);
```

The following statement has the same effect:

```
$db = mysql_connect(‘localhost’, ‘user_name’, ‘password’);
```

For the most part, your specific needs and the way you are designing your table will dictate what piece of code you use. Most people use the first method for security ‘sake and put them variables in a different file. Then they include them wherever they need to make a connection to the database.

9.2.2 Web Interface with Apache, MySQL, and PHP

phpMyAdmin is a tool written in PHP intended to handle the administration of MySQL over the Web. Currently it can create and drop databases, create/drop/alter tables, delete/edit/add fields, execute any SQL statement, manage keys on fields. Features provided by the program include:

1. Web interface

2. MySQL database management

3. Import data from CSV and SQL

4. Export data to various formats: CSV, SQL, XML, PDF (via the TCPDF library), ISO/IEC 26300 - OpenDocument Text and Spreadsheet, Word, Excel, LaTeX and others PHP in the Web Environment

5. Administering multiple servers

6. Creating PDF graphics of the database layout

7. Creating complex queries using Query-by-example (QBE)

8. Searching globally in a database or a subset of it

9. Transforming stored data into any format using a set of predefined functions, like displaying BLOB-data as image or download-link

10. Active query monitor (Processes)

AMP Bundles

There are many AMP bundles such as LAMP, XAMP, PHPTriad, WAMP, MAMP, FoxServ,Etc. consists of AMP build in itself. So, there is no need to configure AMPs manually. Someof the AMPs which are available for various operating systems are LAMP (for Linux);WAMP (for Windows); MAMP (for Macintosh); SAMP (for Solaris); and FAMP (forFreeBSD)

9.3 PHP AND LDAP

LDAP is the Lightweight Directory Access Protocol, and is a protocol 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, organisations 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 organise 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.

9.4 PHP CONNECTIVITY

In order to develop and run PHP Web pages three vital components need to be installed on our computer system.Web Server − PHP will work with virtually all Web Server software, including Microsoft's Internet Information Server (IIS) but then most often used is freely available Apache Server. Download Apache for free here https://httpd.apache.org/download.cgi

Database − PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database. Download MySQL for free here − https://www.mysql.com/downloads/.

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. This tutorial will guide you how to install PHP parser on your computer.

PHP Parser Installation-Before you precede it is important to make sure that you have proper environment setup on your machine to develop your web programs using PHP.

Type the following address into your browser's address box.

http://127.0.0.1/info.php

If this displays a page showing your PHP installation related information then it means you have PHP and Webserver installed properly. Otherwise you have to follow given procedure to install PHP on your computer.This section will guide you to install and configure PHP over the following four platforms

Apache Configuration

If you are using Apache as a Web Server then this section will guide you to edit Apache Configuration Files.Just Check it here − PHP Configuration in Apache Server

PHP.INI File Configuration

The PHP configuration file, php.ini, is the final and most immediate way to affect PHP's functionality.

9.5 SENDING AND RECEIVING E-MAILS

Windows users should ensure that two directives are supplied. The first is called SMTP that defines your email server address. The second is called send mail from which defines your own email address. The configuration for Windows should look something like this −

```
[mail function]

; For Win32 only.

SMTP = smtp.secureserver.net

; For win32 only

sendmail_from = webmaster@tutorialspoint.com
```

Linux users simply need to let PHP know the location of their send mail application. The path and any desired switches should be specified to the send mail_path directive. The configuration for Linux should look something like this −

```
[mail function]

; For Win32 only.

SMTP =

; For win32 only

sendmail_from =

; For Unix only

sendmail_path = /usr/sbin/sendmail -t -i
```

Now you are ready to go −

Sending plain text email

PHP makes use of mail() function to send an email. This function requires three mandatory arguments that specify the recipient's email address, the subject of the message and the actual message additionally there are other two optional parameters.

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

Here is the description for each parameters.

Sr.No	Parameter & Description
1	to Required. Specifies the receiver / receivers of the email
2	subject Required. Specifies the subject of the email. This parameter cannot contain any newline characters
3	message Required. Defines the message to be sent. Each line should be separated with a LF (\n). Lines should not exceed 70 characters
4	headers Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n)
5	parameters Optional. Specifies an additional parameter to the send mail program

As soon as the mail function is called PHP will attempt to send the email then it will return true if successful or false if it is failed. Multiple recipients can be specified as the first argument to the mail() function in a comma separated list.

9.5.1 Sending HTML email

When you send a text message using PHP then all the content will be treated as simple text. Even if you will include HTML tags in a text message, it will be displayed as simple text and HTML tags will not be formatted according to HTML syntax. But PHP provides option to send an HTML message as actual HTML message.While sending an email message you can specify a Mime version, content type and character set to send an HTML email.

Example

Following example will send an HTML email message to xyz@somedomain.com copying it to afgh@somedomain.com. You can code this program in such a way that it should receive all content from the user and then it should send an email.

```
<html>
    <head>
    <title>Sending HTML email using PHP</title>
  </head>
    <body>
        <?php
      $to = "xyz@somedomain.com";
      $subject = "This is subject";

      $message = "<b>This is HTML message.</b>";
```

```
        $message .= "<h1>This is headline.</h1>";
        $header = "From:abc@somedomain.com \r\n";
        $header .= "Cc:afgh@somedomain.com \r\n";
        $header .= "MIME-Version: 1.0\r\n";
        $header .= "Content-type: text/html\r\n";
        $retval = mail ($to,$subject,$message,$header);
         if( $retval == true ) {
           echo "Message sent successfully...";
        }else {
           echo "Message could not be sent...";
        }
     ?>
     /body>
</html>
```

9.5.2 Sending Attachments with Email

To send an email with mixed content requires to set Content-type header to multipart/mixed. Then text and attachment sections can be specified within boundaries. A boundary is started with two hyphens followed by a unique number which can not appear in the message part of the email. A PHP function md5() is used to create a 32 digit hexadecimal number to create unique number. A final boundary denoting the email's final section must also end with two hyphens.

```
<?php
   // request variables // important
   $from = $_REQUEST["from"];
   $emaila = $_REQUEST["emaila"];
   $filea = $_REQUEST["filea"];
   if ($filea) {
      function mail_attachment ($from , $to, $subject, $message,
$attachment){
         $fileatt = $attachment; // Path to the file
         $fileatt_type = "application/octet-stream"; // File Type
         $start = strrpos($attachment, '/') == -1 ?
           strrpos($attachment, '//') : strrpos($attachment, '/')+1;
           $fileatt_name = substr($attachment, $start,
           strlen($attachment)); // Filename that will be used for the
   file as the attachment
  $email_from = $from; // Who the email is from
  $subject = "New Attachment Message";
  $email_subject =  $subject; // The Subject of the email
  $email_txt = $message; // Message that the email has in it
  $email_to = $to; // Who the email is to
  $headers = "From: ".$email_from;
  $file = fopen($fileatt,'rb');
  $data = fread($file,filesize($fileatt));
  fclose($file);
   $msg_txt="\n\n You have recieved a new attachment message from
 $from";
  $semi_rand = md5(time());
  $mime_boundary = "==Multipart_Boundary_x{$semi_rand}x";
   $headers .= "\nMIME-Version: 1.0\n" . "Content-Type:
 multipart/mixed;\n" . "
   boundary=\"{$mime_boundary}\"";
   $email_txt .= $msg_txt;
   $email_message .= "This is a multi-part message in MIME
   format.\n\n" .
    "--{$mime_boundary}\n" . "Content-Type:text/html;
    charset = \"iso-8859-1\"\n" . "Content-Transfer-Encoding: 7bit\n\n"
.
    $email_txt . "\n\n";


  $data = chunk_split(base64_encode($data));
  $email_message .= "--{$mime_boundary}\n" . "Content-Type:
{$fileatt_type};\n" .
    " name = \"{$fileatt_name}\"\n" . //"Content-Disposition:
attachment;\n" .
    //" filename = \"{$fileatt_name}\"\n" . "Content-Transfer-
Encoding:
    base64\n\n" . $data . "\n\n" . "--{$mime_boundary}--\n";
     $ok = mail($email_to, $email_subject, $email_message,
$headers);
     if($ok) {
    echo "File Sent Successfully.";
    unlink($attachment); // delete a file after attachment sent.
```

```
        }else {
          die("Sorry but the email could not be sent. Please go back and try
again!");
        }
      }
      move_uploaded_file($_FILES["filea"]["tmp_name"],
        'temp/'.basename($_FILES['filea']['name']));

      mail_attachment("$from", "youremailaddress@gmail.com",
        "subject", "message", ("temp/".$_FILES["filea"]["name"]));
   }
?>
<html>
   <head>
      <script language = "javascript" type = "text/javascript">
        function CheckData45() {
          with(document.filepost) {
            if(filea.value ! = "") {
              document.getElementById('one').innerText =
                "Attaching File ... Please Wait";
            }
          }
        }
      </script>
      </head>
   <body>
      <table width = "100%" height = "100%" border = "0"
        cellpadding = "0" cellspacing = "0">
        <tr>
          <td align = "center">
            <form name = "filepost" method = "post"
              action = "file.php" enctype = "multipart/form-data" id = "file">
               <table width = "300" border = "0" cellspacing = "0"
                cellpadding = "0">
                  <tr valign = "bottom">
                  <td height = "20">Your Name:</td>
                </tr>
                 <tr>
                   <td><input name = "from" type = "text"
                      id = "from" size = "30"></td>
                </tr>
                <tr valign = "bottom">
                   <td height = "20">Your Email Address:</td>
                </tr>
                <tr>
                   <td class = "frmtxt2"><input name = "emaila"
                      type = "text" id = "emaila" size = "30"></td>
                </tr>
                 <tr>
                   <td height = "20" valign = "bottom">Attach File:</td>
                </tr>
                <tr valign = "bottom">
                   <td valign = "bottom"><input name = "filea"
                      type = "file" id = "filea" size = "16"></td>
                </tr>
                 <tr>
                   <td height = "40" valign = "middle"><input
                      name="Reset2" type="reset" id ="Reset2" value="Reset">
                   <input name = "Submit2" type = "submit"
                      value = "Submit" onClick = "return
CheckData45()"></td>
                </tr>
              </table>
               </form>
              <center>
              <table width = "400">
                 <tr>
                   <td id = "one">
                   </td>
                </tr>
                </table>
            </center>
            </td>
        </tr>
      </table>
      </body>
</html>
```

9.6 DEBUGGING AND ERROR HANDLING

Programs rarely work correctly the first time. Many things can go wrong in your programs that cause the PHP interpreter to generate an error message. You have a choice about where those error messages go. The messages can be sent along with other program output to the web browser. They can also be included in the web server error log. To make error messages display in the browser, set the display errors configuration directive to on. To send errors to the web server error log, set log errors to on. You can set them both to on if you want error messages in both places.

PHP defines some constants you can use to set the value of error_reporting such that only errors of certain types get reported: E_ALL (for all errors except strict notices), E_PARSE (parse errors), E_ERROR (fatal errors), E_WARNING (warnings), E_NOTICE (notices), and E_STRICT (strict notices). While writing your PHP program, it is a good idea to use PHP-aware editors like BBEdit or Emacs. One of the special special features of these editors is syntax highlighting. It changes the color of different parts of your program based on what those parts are. For example, strings are pink, keywords such as if and while are blue, comments are grey, and variables are black.

Another feature is quote and bracket matching, which helps to make sure that your quotes and brackets are balanced. When you type a closing delimiter such as }, the editor highlights the opening { that it matches. There are following points which need to be verified while debugging your program.

Missing Semicolons − Every PHP statement ends with a semicolon (;). PHP doesn't stop reading a statement until it reaches a semicolon. If you leave out the semicolon at the end of a line, PHP continues reading the statement on the following line.

Not Enough Equal Signs − When you ask whether two values are equal in a comparison statement, you need two equal signs (==). Using one equal sign is a common mistake.

Misspelled Variable Names − If you misspelled a variable then PHP understands it as a new variable. Remember: To PHP, $test is not the same variable as $Test.

Missing Dollar Signs − A missing dollar sign in a variable name is really hard to see, but at least it usually results in an error message so that you know where to look for the problem.

Troubling Quotes − You can have too many, too few, or the wrong kind of quotes. So check for a balanced number of quotes.

Missing Parentheses and curly brackets − They should always be in pairs.

Array Index − All the arrays should start from zero instead of 1.

Moreover, handle all the errors properly and direct all trace messages into system log file so that if any problem happens then it will be logged into system log file and you will be able to debug that problem.

Error handling is the process of catching errors raised by your program and then taking appropriate action. If you would handle errors properly then it may lead to many unforeseen consequences. It’s very simple in PHP to handle an errors.

Using die() function

While writing your PHP program you should check all possible error condition before going ahead and take appropriate action when required.Try following example without having /tmp/test.xt file and with this file.

```
<?php

  if(!file_exists("/tmp/test.txt")) {

    die("File not found");

  }else {

    $file = fopen("/tmp/test.txt","r");

    print "Opend file sucessfully";

  }

  // Test of the code here.

?>
```

This way you can write an efficient code. Using above technique you can stop your program whenever it errors out and display more meaningful and user friendly message.

9.6.1 Defining Custom Error Handling Function

PHP and MySQL Database

You can write your own function to handling any error. PHP provides you a framework to define error handling function. This function must be able to handle a minimum of two parameters (error level and error message) but can accept up to five parameters (optionally: file, line-number, and the error context)

Syntax

error_function(error_level,error_message,
error_file,error_line,error_context);

Sr.No **Parameter & Description**

1 error_level Required - Specifies the error report level for the user defined error. Must be a value number

2 error_message Required - Specifies the error message for the user-defined error

3 error_file Optional - Specifies the file name in which the error occurred

4 error_line Optional - Specifies the line number in which the error occurred

5 error_context Optional - Specifies an array containing every variable and their values in use when the error occurred

9.6.2 Possible Error levels

These error report levels are the different types of error the user-defined error handler can be used for. These values cab used in combination using | operator

Sr.No **Constant & Description**

1 E_ERROR Fatal run-time errors. Execution of the script is halted

2 E_WARNING Non-fatal run-time errors. Execution of the script is not halted

3 E_PARSE Compile-time parse errors. Parse errors should only be generated by the parser

4 E_NOTICE Run-time notices. The script found something that might be an error, but could also happen when running a script normally

5 E_CORE_ERROR Fatal errors that occur during PHP's initial start-up.

```
int error_reporting ( [int $level] )
```

Following is the way you can create one error handling function:

```
<?php
  function handleError($errno, $errstr,$error_file,$error_line) {
    echo "<b>Error:</b> [$errno] $errstr - $error_file:$error_line";
    echo "<br />";

    echo "Terminating PHP Script";


    die();

  }

?>
```

9.6.3 Exceptions Handling

PHP 5 has an exception model similar to that of other programming languages. Exceptions are important and provide a better control over error handling. Let’s explain their new keyword related to exceptions.

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. Exception can be thrown, and caught ("catched") 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

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php

  try {

    $error = 'Always throw this error';

    throw new Exception($error);

    // Code following an exception is not executed.

    echo 'Never executed';
```

PHP and MySQL Database

```
  }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. There are following functions which can be used from Exception class.

getMessage() − message of exception

getCode() − code of exception

getFile() − souce filename

getLine() − source line

getTrace() − n array of the backtrace()

getTraceAsString() − formated string of trace

9.7 TEMPLATES

Many programming instructors emphasize separating content, layout, and data. However, the way PHP is often written, it combines all three elements. As a response, web developers often use some form of templating system to try to separate the content from the view. The simplest form of a template is something like the following code:

```
<?php include_once("vars.php") ?>

<!doctype html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title><?=$title?></title>

</head>

<body>

  <h1><?=$heading?></h1>

  <div id = "main">

    <?=$content?>
  </div>
  <div id = "footer">
    <?=$footer?>
  </div>
</body>
</html>
```

The page holds the structure, but none of the actual contents. The contents are stored in PHP variables, which are stored in another file, called (in this example) vars.php. Here's what vars.php might look like:

```
<?php
$title = "template demo";
$heading = "Template Demo";
$content = <<<HERE
This is a very simple template example, which allows the user to load content as variables from an external file. Of course, template systems get much more complex than this.
HERE;
$footer = "from HTML5 / CSS All in One for Dummies";
?>
```

In this extremely simple example, the second PHP file simply defines variables containing the various values, achieving separation of data from view. Often, the secondary PHP file is more sophisticated, grabbing contents from a database or other storage mediu

Check your Progress

1. What is PHP?
2. What is LDAP?
3. What are Missing Semicolons?
4. What are Missing Dollar Signs?
5. What are templates?

9.8 ANSWERS TO CHECK YOUR PROGRESS

1. Python is the open source programming language which is very easy and effective among programmers. This unit helps the user to start from the basic of python in defining their syntax, control and

loops which makes the beginners to learn and understand more easily.

2. There are three numeric types in Python:
   - int
   - float
   - complex

3. Immutable Objects are of in-built types like int, float, bool, string, unicode, tuple. In simple words, an immutable object can't be changed after it is created.
4. A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.
5. Python conditional statements are also known as selection statements.Among multiple options only one statement is selected and executed is selection statements.Conditional statements in python are if statement, if-else statements and if elif in python.

9.9 SUMMARY

- The open source software PHP programming is combined with other backend open source available software's for creation of real time applications.
- phpMyAdmin is a tool written in PHP intended to handle the administration of MySQL over the Web.
- LDAP is the Lightweight Directory Access Protocol, and is a protocol used to access "Directory Servers".
- In order to develop and run PHP Web pages three vital components need to be installed on our computer system.
- Windows users should ensure that two directives are supplied. The first is called SMTP that defines your email server address. The second is called send mail from which defines your own email address.

9.10 KEYWORDS

Error handling: It is the process of catching errors raised by your program and then taking appropriate action.

Missing Semicolons − Every PHP statement ends with a semicolon (;). PHP doesn't stop reading a statement until it reaches a semicolon.
Catch: A "catch" block retrieves an exception and creates an object containing the exception information.

PHP and MySQL Database

9.11 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer questions:

1. What arePossible Error levels?

2. Define Custom Error Handling Function?
3. Define Debugging and Error Handling?
4. What areAMP Bundles?
5. Explain about combine PHP with PDAP?

Long Answer questions:

1. Explain briefly PHP and LDAP?
2. Explain about Sending and Receiving E-Mails?
3. Explain about Debugging and Error Handling?
4. Explain about Exceptions Handling?

9.12 FURTHER READINGS

Rémy Card, Eric Dumas, and Franck Mével. The Linux kernel book. John Wiley & Sons, Inc., 2003.

Steve Suchring. MySQL BBible. John Wiley, 2002.

Rasmus Lerdorf and Levin Tatroe. Programming PHP. " O'Reilly Media, Inc., 2002.

Wesley J. Chun. Core Python Programming. Prentice Hall, 2001.

Martin C. Brown. Perl: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Steven Holzner. PHP: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Vikram Vaswani. MySQL: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

BLOCK – IV
OPEN SOURCE PROGRAMMING LANGUAGE PYTHON

UNIT 10
SYNTAX AND STYLE

Structure

10.0 INTRODUCTION

Python is the open source programming language which is very easy and effective among programmers. This unit helps the user to start from the basic of python in defining their syntax, control and loops which makes the beginners to learn and understand more easily

10.1 OBJECTIVE

The users after going through this unit can start programming in python by learning the following concepts

- Phython objects
- Numbers
- Sequences
- Strings
- Loops

10.2 SYNTAX AND STYLE

We learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

```
>>> print("Hello, World!")

Hello, World!
```

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

```
C:\Users\Your Name>python myfile.py
```

10.3 CREATING AN OBJECT IN PYTHON

We saw that the class object could be used to access different attributes. It can also be used to create new object instances (instantiation) of that class. The procedure to create an object is similar to a function call.

```
>>> ob = MyClass()
```

This will create a new instance object named ob. We can access attributes of objects using the object name prefix. Attributes may be data or method. Method of an object are corresponding functions of that class. Any function object that is a class attribute defines a method for objects of that class. This means to say, since MyClass.func is a function object (attribute of class), ob.func will be a method object.

10.4 PYTHON NUMBERS

There are three numeric types in Python:

- int
- float

Int

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

Example

Integers:

```
x = 1

y = 35656222554887711

z = -3255522

print(type(x))
print(type(y))

print(type(z))
```

Float

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

Example

Floats:

```
x = 1.10

y = 1.0

z = -35.59

print(type(x))

print(type(y))

print(type(z))
```

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

Complex

Complex numbers are written with a "j" as the imaginary part:

Example

Complex:

```
x = 3+5j

y = 5j

z = -5j

print(type(x) )

print(type(y))

print(type(z))
```

10.5 SEQUENCE

In Python, sequence is the generic term for an ordered set. There are several types of sequences in Python, the following three are the most important. Lists are the most versatile sequence type. The elements of a list can be any object, and lists are mutable - they can be changed. Elements can be reassigned or removed, and new elements can be inserted.

Tuples are like lists, but they are immutable - they can't be changed. Strings are a special type of sequence that can only store characters, and they have a special notation. However, all of the sequence operations described below can also be used on strings.

10.6 STRINGS

A string is a sequence of characters.

Strings are the data types in Python.

Python treats single quotes the same as double quotes.

```
var1 = 'Hello World!'
var2 = "Python Programming"
```

Simple program

```
my_string = "Hello“
print(my_string)
```

Output:

```
Hello
```

String Slicing method

Given a string s, the syntax for a slice is:

```
s[startIndex:pastIndex]
```

The startIndex is the start index of the string. pastIndex is one past the end of the slice.

If you omit the first index, the slice will start from the beginning. If you omit the last index, the slice will go to the end of the string.

Program:

```
var1 = 'Hello World!'
var2 = "Python Programming"
print ("var1[0]: ", var1[0] )
print( "var2[1:5]: ", var2[1:5])
```

Output:

```
var1[0]: H
var2[1:5]: ytho
```

Program:

```
str = 'programiz‘
print('str = ', str)
print('str[0] = ', str[0])
print('str[-1] = ', str[-1])
print('str[1:5] = ', str[1:5])
print('str[5:-2] = ', str[5:-2])
```

Output:

```
str =  programiz
str[0] =  p
str[-1] =  z
str[1:5] =  rogr
str[5:-2] =  am
```

String Built in Function

1-lower()

Converts all uppercase letters in string to lowercase.

Syntax:

s.lower()

Program:

```
string = "Hello World"
print (string.lower() )
```

Output:

```
Hello world
```

2-upper()

returns uppercase version of the string

Syntax:

s.upper()

Program:

```
string = "Hello World"
print (string.upper() )
```

Output:

```
HELLO WORLD
```

3-strip()

returns a string with whitespace removed from the start andend

Syntax:

s.strip()

Program:

```
string = "Hello World"
print (string.strip() )
```

Output:

```
HelloWorld
```

4-replace('old', 'new')

returns a string where all occurrences of 'old' have been replaced by 'new‘

Syntax:

s.replace()

Program:

```
string = "Hello World"
print (string.replace(“Hello”,”Welcome”) )
```

Output:

```
Welcome World
```

5-split('delim')

returns a list of substrings separated by the given delimiter.

Syntax:

s.split()

Program:

```
string = "Hello World"
print (string.split() )
```

Output:

```
['Hello', 'World']
```

6-join(list)

opposite of split(), joins the elements in the given list together using the string as the delimiter.

Syntax:

```
s.join()
```

Program:

```
string = ['Hello', 'World']
print (string.join() )
```

Output:

```
"Hello World"
```

7-capitalize()

Capitalizes first letter of string

Syntax:

```
s.capitalize()
```

Program:

```
string = "hello world"
print (string.capitalize() )
```

Output:

```
Hello world
```

8-len(string)

Returns the length of the string

Syntax:

```
len(string)
```

Program:

```
string = "Hello World"
print (len(string) )
```

Output:

```
10
```

9-isalnum()

Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.

Syntax:

```
s.isalnum()
```

Program:

```
string = "Hello World"
print (string.isalnum() )
```

Output:

```
False
```

10-isalpha() -

Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.

Syntax:

```
s.alpha()
```

Program:

```
string = "Hello World"
print (string.isalpha() )
```

Output:

```
True
```

11-isdigit()

Returns true if string contains only digits and false otherwise.

Syntax:

```
s.isdigit()
```

Program:

```
string = "Hello World"
print (string.isdigit() )
```

Output:

```
False
```

12-islower()

Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.

Syntax:

```
s.islower()
```

Program:

```
string = "Hello World"
print (string.islower() )
```

Output:

```
False
```

13-isnumeric()

Returns true if a unicode string contains only numeric characters and false otherwise.

Syntax:

```
s.isnumeric()
```

Program:

```
string = "Hello World"
print (string.isnumeric() )
```

Output:

```
False
```

14-isspace()

Returns true if string contains only whitespace characters and false otherwise.

Syntax:

```
s.isspace()
```

Program:

```
string = "Hello World"
print (string.isspace() )
```

Output:

```
--------------True
```

15-istitle()

Returns true if string is properly "titlecased" and false otherwise.

Syntax:

```
s.istitle()
```

Program:

```
string = "Hello World"
print (string.istitle() )
```

Output:

```
------------True
```

16-isupper()

Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.

Syntax:

```
s.isupper()
```

Program:

```
string = "Hello World"
print (string.isupper() )
```

Output:

```
False
```

String Module

String module is a python script file, which contains several number of related functions to strings that script is used as module without its extension(.py) in other python program. This is called string module

Program:

```
import string   #importing srting module
text = "Monty Python's Flying Circus"
print ("upper", "=>", string.upper(text))
print ("lower", "=>", string.lower(text))
print ("split", "=>", string.split(text))
print ("join", "=>", string.join(string.split(text), "+"))
print ("replace", "=>", string.replace(text, "Python", "Java"))
print ("find", "=>", string.find(text, "Python"), string.find(text,
"Java"))
Print("count", "=>", string.count(text, "n"))
```

Output:

```
upper => MONTY PYTHON'S FLYING CIRCUS
lower => monty python's flying circus
split => ['Monty', "Python's", 'Flying', 'Circus']
join => Monty+Python's+Flying+Circus
replace => Monty Java's Flying Circus
find => 6 -1
count => 3
```

Immutability in python:

Mutable vs Immutable Objects in Python

- Every variable in python holds an instance of an object. There are two types of objects in python i.e. Mutable and Immutable objects.
- Whenever an object is instantiated, it is assigned a unique object id. The type of the object is defined at the runtime and it can’t be changed afterwards. However, it’s state can be changed if it is a mutable object.
- mutable objects can change their state or contents and immutable objects can’t change their state or content.

Immutable Objects : These are of in-built types like int, float, bool, string, unicode, tuple. In simple words, an immutable object can’t be changed after it is created.

Example:

```
# Python code to test that
# tuples are immutable

tuple1 = (0, 1, 2, 3)
tuple1[0] = 4
print(tuple1)
Error :


Traceback (most recent call last):
  File "e0eaddff843a8695575daec34506f126.py", line 3, in
tuple1[0]=4
TypeError: 'tuple' object does not support item assignment.
```

10.7 LISTS

Lists are the one of data types in python.

A list contains items or elments separated by commas and enclosed within square brackets ([]).

Lists are similar to arrays in python.

One of the differences between them is that all the items belonging to a list can be of different data type.

Example:

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
```

10.7.1 Creating List

Creating a list is as simple as putting different comma-separated values between square brackets. For example −

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"]
```

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

10.7.2 List Slices

Accessing List values or (List Slicing)

The values stored in a list can be accessed using the slice operator [:] with indexes starting at 0 in the beginning of the list and working their way to end -1.

Example:

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
print (list)            # Prints complete list
print (list[0])         # Prints first element of the list abcd
print (list[1:3])       # Prints elements starting from 2nd till 3rd
print (list[2:])        # Prints elements starting from 3rd element
```

Output:

```
['abcd', 786, 2.23, 'john', 70.2]
```

```
['abcd‘]
[786, 2.23]
[2.23, 'john', 70.2]
```

NOTES

10.7.3 List Methods

i) insert() method:

The method **insert()** inserts object *obj* into list at offset *index.*

Syntax

list.insert(index, obj) Parameters

index − This is the Index where the object obj need to be inserted.

obj − This is the Object to be inserted into the given list.

Example

```
aList = [123, 'xyz', 'zara', 'abc']
aList.insert( 3, 2009)
print ("Final List : ", aList)
```

Output:

```
Final List : [123, 'xyz', 'zara', 2009, 'abc']
```

ii) remove() method:

Removes object obj from list.

Syntax

list.remove(obj)

obj − This is the Object to be removed from the given list.

Example

```
aList = [123, 'xyz', 'zara', 'abc', ‘kyz'];
aList.remove('xyz');
Print("List : ", aList)
aList.remove('abc');
Print("List : ", aList)
```

Output:

```
List :  [123, 'zara', 'abc', ‘kyz']
List :  [123, 'zara', ‘kyz']
```

iii) pop() method:

The method pop() removes and returns last object or obj from the list.

Syntax

list.pop(obj = list[-1])

obj − This is an optional parameter, index of the object to be removed from the list.

Example

```
aList = [123, 'xyz', 'zara', 'abc'];
print ("A List : ", aList.pop())
print ("B List : ", aList.pop(2))
```

Output:

```
A List :  abc
```

```
B List :  zara
```

iv) reverse() method:

The method **reverse()** reverses objects of list in place.

Syntax

```
list.reverse()
```

NOTES

Example

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz'];
aList.reverse();
Print("List : ", aList)
```

Output:

```
List :  ['xyz', 'abc', 'zara', 'xyz', 123]
```

v) append() method:

The method append() appends a passed obj into the existing list.

Syntax

```
list.append(obj)
```

obj − This is the object to be appended in the list.

Example

```
List = [123, 'xyz', 'zara', 'abc'];
aList.append( 2009 );
Print("Updated List : ", aList)
```

Output:

```
Updated List :  [123, 'xyz', 'zara', 'abc', 2009]
```

vi) extend() method:

The method extend() appends the contents of seq to list.

Syntax

```
list.extend(seq)
```

seq − This is the list of elements

Example

```
aList = [123, 'xyz', 'zara', 'abc', 123];
bList = [2009, 'manni'];
aList.extend(bList)
print("Extended List : ", aList)
```

Output:

```
Extended List :  [123, 'xyz', 'zara', 'abc', 123, 2009, 'manni']
```

vii) sort() method:

The method sort() sorts the contents of seq to list.

Syntax

```
list.sort()
```

Example

```
aList = [50, 30, 40, 10, 20];
aList.sort();
print ("List : ", aList)
```

Output:

```
List :  [10, 20, 30, 40, 50]
```

viii) index() method:

The method index() returns the lowest index in list that obj appears.

Syntax

```
list.index(obj)
    obj − This is the object to be find out.
```

Example

```
aList = [123, 'xyz', 'zara', 'abc'];
print ("Index for xyz : ", aList.index( 'xyz' ) )
print ("Index for zara : ", aList.index( 'zara' ) )
```

Output:

```
Index for xyz :  1
Index for zara :  2
```

ix) count() method:

The method count() returns count of how many times obj occurs in list.

Syntax

```
list.count(obj)
    obj − This is the object to be counted in the list.
```

Example

```
aList = [123, 'xyz', 'zara', 'abc', 123];
print ("Count for 123 : ", aList.count(123))
print ("Count for zara : ", aList.count('zara'))
```

Output:

```
Count for 123 :  2
Count for zara :  1
```

10.8 TUPLES: TUPLE ASSIGNMENT, TUPLE AS RETURN VALUE

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

10.8.1 Tuple Creation

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also.

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 they can be sliced, concatenated, and so on.

10.8.2 Tuple Slicing or 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. For example −

```
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]
```

10.8.3 Tuple Mutability or Updating Tuples

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates −

```
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')
```

10.8.4 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. For example −

```
tup = ('physics', 'chemistry', 1997, 2000);
print tup;
del tup;
print "After deleting tup : ";
print tup;
```

This produces the 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
```

10.9 DICTIONARY

Python dictionary is an unordered collection of items.
While other compound data types have only value as an element, a dictionary has a key: value pair.
Dictionaries are optimized to retrieve values when the key is known.

- Creating a dictionary is as simple as placing items inside curly braces {} separated by comma.
- An item has a key and the corresponding value expressed as a pair, key: value.
- 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.

10.9.1 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 −

Example:

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

Output

```
dict['Name']: Zara
dict['Age']: 7
```

10.9.2 Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example −

Program

```
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']
```

Output

```
dict['Age']: 8
dict['School']: DPS School
```

10.9.3 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 −

Program

```
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']
```

Output

```
Error undefined
```

10.9.4 Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys

(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins.

For example

```
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}
print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result −

```
dict['Name']:  Manni
```

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

Following is a simple example

```
dict = {['Name']: 'Zara', 'Age': 7}
print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result −

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    dict = {['Name']: 'Zara', 'Age': 7};
TypeError: unhashable type: 'list'
```

10.10 CONTROL FLOW

The statements inside your source files are generally executed from top to bottom, in the order that they appear. Control flow statements, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code.

In python, flow control is divided into three types which are as follows

- Conditional statements
- Iterative statements
- Transfer statements

Python Conditional statements

- Python conditional statements are also known as selection statements.
- Among multiple options only one statement is selected and executed is selection statements.
- Conditional statements in python are if statement, if-else statements and if elif in python.

Python Iterative statements

- Python iterative statements are used to execute the same Statement repeatedly.
- Sometimes multiple statements want to execute repeatedly is iterative statements.
- Iterative statements in python: for loop and while loop.

Python Transfer statements

- Transfer statements in python are the statements which are used to transfer execution to other statements.
- Python transfer statements are break statement, continue statement and pass statement.

Decision making statements (Conditional)

Decision making constructs with Boolean expression, an expression returns either TRUE or FALSE
(i.e., 0-false and 1-true). Decision making structure is to perform an action or a calculation only when a certain condition is met. There are four types of decision making structure. They are,

1. **if statement** (Conditional statement)
2. **if … else statement** (Alternative statement)
3. **elif statement** (Chained condition)
4. **nested if statement**

10.10.1 if statement

The program evaluates the condition and will execute statement(s) or process only if the test expression is True.

Syntax:

```
if (test expression/condition):
        True statement
```

Program:

```
num = 3
if num > 0:
        print(num, "is a positive number.")
print("This is always printed.")
```

Output:

```
3 is a positive number
This is always printed
```

10.10.2 if…else statement

The if else statement evaluates condition and will execute body of if only when test condition is True. If the condition is False, body of else is executed.

Syntax:

```
if (test expression/condition):
        True statement
else:
        False statement
```

Program:

```
num=3
if (num >= 0):
    print("Positive or Zero")
else:
    print("Negative number")
```

Output:

```
Positive or Zero
```

10.10.3 if…elif…else statement

The elif is short for else if. It allows us to check for multiple expressions. If the condition for if is False, it checks the condition of the next elif block and so on. If all the conditions are False, body of else is executed.

Syntax:

```
if (test expression/condition):
        True statement 1
elif (test expression/condition):
        True statement 2
else:
        False statement
```

Program:

```
num=3
if num > 0:
        print("Positive number")
elif num = = 0:
        print("Zero")
```

```
else:
        print("Negative number")
```

Output:

```
Positive number
```

10.10.4 Nested if statement

Nested conditional statements are used whenever there is a need to check for another condition after the first condition has been evaluated as True. if...else statement inside another if...else statement.

Syntax:

```
if (test expression/condition 1):
        if (test expression/condition 2):
                True statement 1
        else:
                False statement 1
else:
        False statement 2
```

Program:

```
num = int(input("Enter a number: "))
if num >= 0:
        if num = = 0:
                print("Zero")
        else:
                print("Positive number")
else:
        print("Negative number")
```

Output:

```
Enter a number: 5
Positive number
```

10.11 ITERATION (LOOPING STATEMENTS):

(Looping/ Repetition statement)

Loop statement is to execute a specific block of code in multiple numbers of times. A loop is a programming control structure that facilitates the repetition execution of a statement or group of statement. There are two types of loop statement. They are,

1. **while loop**
2. **for loop**

10.11.1 while statement

A while loop statement in Python programming language repeatedly executes a block of statement until the condition is True. It tests the condition before executing the loop body.

Syntax:

```
while (test expression/ condition):
        body of loop
```

Program:

```
count = 0
while (count < 5):
        print ('The count is:', count )
        count = count + 1
    print ("Good bye!”)
```

Output:

```
    The count is:0
    The count is:1
    The count is:2
    The count is:3
    The count is:4
    Good bye!
```

10.11.2 Nested while loop

Nesting defined as the placing of one while loop inside the body of another while loop.

Syntax:

```
    while (test expression/ condition):
        while (test expression/ condition):
            body of loop
```

Program:

```
    count = 1
    while (count!=0):
        while(count<5):
            print ('The count is:', count )
            count = count + 1
    print ("Good bye!”)
```

Output:

```
    The count is:1
    The count is:2
    The count is:3
    The count is:4
    Good bye!
```

10.11.3 Using else statement with while loops

If the else statement is used with a while loop, the else statement is executed when the condition become false.

Program:

```
    i=0
    while i < 5:
        print(i,”is less than 5”)
        i=i+1
    else:
        print(i,”is not less than 5)
```

Output

```
    0 is less than 5
    1 is less than 5
    2 is less than 5
    3 is less than 5
    4 is less than 5
    5 is not less than 5
```

10.11.4. The Infinite while Loop

NOTES

While the loop runs continuously without termination

Program:

```
while (1):
        num = input("Enter a number :")
        print ("You entered: ", num )
print "Good bye!“
```

Output

```
Enter a number: 20
You entered:  20
Enter a number: 25
You entered:  25
```

10.11.5 for loop

for loop executes a sequence of statements that allows a code to be repeated a certain number of times using “range” function.

Syntax:

```
for  variableName  in  <Group of numbers>:
        body of loop
```

Program:

```
num = [6, 5, 3, 8, 4, 2, 5, 4, 11]
sum = 0
for v in num:
        sum = sum+v
print("The sum is", sum)
```

Output:

```
The sum is 48
```

10.11.6 Nested for Loop

Nesting defined as the placing of one for loop inside the body of another for loop.

Syntax:

```
for  variableName  in  <Group of numbers>:
        for  variableName  in  <Group of numbers>:
                body of loop
```

Program:

```
S=0
for i in range(3):
        for j in range(3):
                S=i+j
                Print(s)
print(“end”)
```

Output:

```
0
2
4
end
```

Self-Instructional Material

10.11.7 Using else Statement with for Loops

If the else statement is used with a for loop, the else statement is executed when the loop has exhausted iterating the list.

Program:

```
for i in range(0,5):
        print(i,”is less than 5”)
else:
        print(i,”is not less than 5)
```

Output:

```
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5
```

10.11.8. for Loop using Range

The range function specifies a range of integers:

range (*start, stop*)

The integers between *start* (inclusive) and *stop* (exclusive)

Syntax:

for *variableName* in range (*start, stop*):

statements

Example:

```
for x in range(1, 6):
        print (x, "squared is", x * x)
```

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

How to use range:

```
range(10)       #produces the list: [0,1,2,3,4,5,6,7,8,9]
range(1, 7)     #produces the list: [1,2,3,4,5,6]
range(0, 30, 5)         #produces the list: [0,5,10,15,20,25]
range(5, -1, -1)        #produces the list: [5,4,3,2,1,0]
```

10.11.9 for Loop using Variable Name

Syntax:

for *variableName* **in** *listname***:**

statements

Program:

```
list=[“apple”,”orange”,”banana”]
for fruits in list:
        print(“current fruit:”,fruits)
print(“End”)
```

Output:

```
Current fruit : apple
Current fruit : orange
Current fruit : banana
End
```

for loop using Length of the list:

Program:

```
lst=[“sam”,”abc”,”zara”]
for i in range(len(lst)):
        print(lst[i])
print(“End”)
```

Output:

```
Sam
abc
zara
End
```

Check your Progress

1. What is Python?
2. What are numeric types in python?
3. What are Immutable Objects?
4. What isTuple?
5. What is Python Conditional statements?

10.12 ANSWERS TO CHECK YOUR PROGRESS

1. Python is the open source programming language which is very easy and effective among programmers. This unit helps the user to start from the basic of python in defining their syntax, control and loops which makes the beginners to learn and understand more easily.

2. There are three numeric types in Python:
   - int
   - float
   - complex

3. Immutable Objects are of in-built types like int, float, bool, string, unicode, tuple. In simple words, an immutable object can’t be changed after it is created.
4. A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.
5. Python conditional statements are also known as selection statements.Among multiple options only one statement is selected and executed is selection statements.Conditional statements in python are if statement, if-else statements and if elif in python.

10.13 SUMMARY

- In Python, sequence is the generic term for an ordered set. There are several types of sequences in Python, the following three are the most important.
- Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.
- In python, flow control is divided into three types which are as follows: Conditional statements, Iterative statements and Transfer statements.
- In python, flow control is divided into three types which are as follows: Conditional statements, Iterative statements and Transfer statements.
- Every variable in python holds an instance of an object. There are two types of objects in python i.e. Mutable and Immutable objects.

10.14 KEYWORDS

Deleting List values: To remove a list element, the del statement can be used if the element(s) that we want to delete is known exactly or the remove() method

Tuples: They are like lists, but they are immutable - they can't be changed. Strings are a special type of sequence that can only store characters, and they have a special notation.

String module: String module is a python script file, which contains several number of related functions to strings that script is used as module without its extension(.py) in other python program.

10.15 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer questions:

6. What is Python Transfer statements?
7. What is Python Iterative statements?
8. What is Iteration?
9. What are built-in Dictionary Functions & Methods?
10. What is Dictionary?

Long Answer questions:

1. Explain about Tuple and its built-in functions?
2. Explain about String Built in Function?
3. Explain about Python and its concepts?

10.16 FURTHER READINGS

Rémy Card, Eric Dumas, and Franck Mével. The Linux kernel book. John Wiley & Sons, Inc., 2003.

Steve Suchring. MySQL BBible. John Wiley, 2002.

Rasmus Lerdorf and Levin Tatroe. Programming PHP. " O'Reilly Media, Inc., 2002.

Wesley J. Chun. Core Python Programming. Prentice Hall, 2001.

Martin C. Brown. Perl: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Steven Holzner. PHP: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Vikram Vaswani. MySQL: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

UNIT 11
FILES

Structure

11.0 INTRODUCTION

The exception handling and File handling in Python helps the users to understand and work with programming language more efficiently. This unit explains the concepts of Exception handling and file handling with example programs of how it can be implemented and used in real time.

11.1 OBJECTIVE

This unit helps the user to learn and understand

- Functions in Python
- Files
- Exception Handling
- Classes and OOp
- Exceution Environment

11.2 FILES

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

- Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.
- In Python, there is no need for importing external library to read and write files. Python provides an inbuilt function for creating, writing and reading files.
- Hence, in Python, a file operation takes place in the following order.
    - Open a file
    - Read or write (perform operation)
    - Close the file

11.2.1 Types of File

In Python, a file is categorized as

- Text file
- Binary file

11.2.1.1 Text File

- Text files are structured as a **sequence of lines**, where each line includes a sequence of characters.
- It is a sequence of characters stored on a permanent medium like a hard drive, flash memory, or CD-ROM
- A text file is a file that contains printable characters and whitespace, organized in to lines separated by newline characters
- Each line is terminated with a special character, called the EOL or **End of Line character**.
- There are several types, but the most common is the **comma {,} or newline character**. It ends the current line and tells the interpreter a new one has begun.

11.2.1.2 Binary File

- It is any type of file that is not a text file
- It can only be **processed by an application** that know or understand the file's structure.

11.3 FILE MODES

S.No	Mode & Description
1.	**r** Opens a file for reading only in text format. The file pointer is placed at the beginning of the file. This is the default mode.
2.	**rb** Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
3.	**r+** Opens a file for both reading and writing. The file pointer placed at the beginning of the file.

4.	**rb+** Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
5.	**w** Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
6.	**wb** Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
7.	**w+** Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
8.	**wb+** Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
9.	**a** Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
10.	**ab** Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
11.	**a+** Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
12.	**ab+** Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

11.3.1 File Object Attributes

S.No	File object attributes& Description
1	**file.closed** Returns true if file is closed, false otherwise.
2	**file.mode** Returns access mode with which file was opened.
3	**file.name** Returns name of the file.

NOTES

11.3.2 Opening and closing a file

The open ()

- To open a file 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.
- Syntax:
    - file object = open(file_name [, access_mode][, buffering])
- Here are parameter details −
- file_name – The file_name argument is a string value that contains the name of the file that you want to access.
- access_mode – The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. Default file access mode is read (r).
- buffering − If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file.

ExamplePrgram:

```
# Open a file
fo = open(“days.txt", “r")
print ("Name of the file: ", fo.name)
print ("Closed or not : ", fo.closed)
print ("Opening mode : ", fo.mode)
fo.close()
```

Output:

```
Name of the file: file1.txt
Closed or not : False
Opening mode : r
```

The close ()

The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Syntax: fileObject.close();

Program:

```
# Open a file
fo = open("file1.txt", “r")
print ("Name of the file: ", fo.name)
# Close opened file
fo.close()
print ("Closed or not : ", fo.closed)
```

Output:

```
Name of the file: file1.txt
Closed or not : True
days.txt
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

11.3.3 Reading and writing a file

The read() Method

To read a file in Python, we must open the file in reading mode.

The code for opening myfile.txt for input:

```
>>>ƒfƒ=ƒopen(“myfile.txt”,ƒ'r')
```

Syntax:fileObject.read(size);

There are various methods available for this purpose. We can use the read(size) method to read in size number of data. If size parameter is not specified, it reads and returns up to the end of the file.

Program:

```
# Open a file
fo = open(“days.txt", "r")
str1= fo.read(7)
str2 = fo.read()
print ("String is : ", str1)
print ("String is : ", str2)
# Close opened file
fo.close()
Output:
String is : Monday
String is : ‘Tuesday\nWednesday\nThursday\nFriday\nSaturday\nSunday\n’
```

The readline() Method

It reads a single line from the file; a newline character (\n) is left at the end of the string, and is only omitted on the last line of the file if the file doesn’t end in a newline.

- **Syntax:**fileObject.readline(line number);

Program:

```
# Open a file
fo = open(“days.txt", "r")
print(fo.readline())
print(fo.readline(3))
Output:
Monday
Tuesday
```

The readlines() Method

It return every line in the file, properly separated

Syntax:fileObject.readlines();

Program:

```
# Open a file
fo = open(“days.txt", "r")
fl =fo.readlines()
for x in fl:
print x
```

Output:

```
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

The write() Method

The write() method writes any string to an open file.

Data can be output to a text file using a file object. Python’s open function, which expects a file pathname and a mode string as arguments, opens a connection to the file on disk and returns a file object.

The mode string is 'r' forinput files and 'w' for output files. Thus, the following code opens a file objecton a file named myfile.txt for output:

```
    >>>ƒfƒ=ƒopen(“myfile.txt”,ƒ'w')
```

- The write() method does not add a newline character ('\n') to the end of the string. String data are written (or output) to a file using the method write with the file object. The write method expects a single string argument. If you want the output text to end with a newline, you must include the escape character \n in the string. The next statement writes two lines of text to the file:

```
    >>>ƒf.write(“Firstƒline.\nSecondƒline.\n”)
```

- When all of the outputs are finished, the file should be closed using the method close, as follows:

```
    >>>ƒf.close()
```

- Failure to close an output file can result in data being lost.

Program 1:

```
# Open a file
fo = open("file1.txt", "w")
fo.write( "Python is a great language.\nYeah its great!!\n")
# Closeopened file
fo.close()
```

Output: file1.txt contains

```
Python is a great language.
Yeah its great!!
```

Program 2:

Fivehundred random integers between 1 and 500 are generated and written to a text file named integers.txt. The newline character is the separator.

```
importƒrandom
fƒ=ƒopen("integers.txt",ƒ'w')
forƒcountƒinƒrange(500):
ƒƒƒƒnumberƒ=ƒrandom.randint(1,ƒ500)
ƒƒƒƒf.write(str(number)ƒ+ƒ"\n")
f.close()
```

The writelines() Method

The method **writelines()** writes a sequence of strings to the file. The sequence can be any iterable object producing strings, typically a list of strings. There is no return value.

Syntax

Following is the syntax for **writelines()** method −

```
fileObject.writelines( sequence )
```

Parameters

- **sequence** − This is the Sequence of the strings.

Return Value

This method does not return any value.

Example

The following example shows the usage of writelines() method.

```
This is 1st line
This is 2nd line
This is 3rd line
This is 4th line
This is 5th line
# Open a file in witre mode
fo = open("foo.txt", "rw+")
print "Name of the file: ", fo.name
# Assuming file has following 5 lines
# This is 1st line
# This is 2nd line
# This is 3rd line
# This is 4th line
# This is 5th line
```

Program :

```
seq = ["This is 6th line\n", "This is 7th line"]
```

```
# Write sequence of lines at the end of the file.
fo.seek(0, 2)
line = fo.writelines( seq )
# Now read complete file from beginning.
fo.seek(0,0)
for index in range(7):
        line = fo.next()
        print "Line No %d - %s" % (index, line)
# Close opend file
fo.close()
```

Output:

```
Name of the file:  foo.txt
Line No 0 - This is 1st line
Line No 1 - This is 2nd line
Line No 2 - This is 3rd line
Line No 3 - This is 4th line
Line No 4 - This is 5th line
Line No 5 - This is 6th line
Line No 6 - This is 7th line
```

NOTES

11.3.4 The append()

The mode “a” is used to append a new text to the already existing file or the new file

Program 1:

```
# Open a file
fo = open("file1.txt", “a")
fo.write(“Python is interpreted language")
# Close opend file
fo.close()
```

Output: file1.txt contains

```
Python is a great language.
Yeah its great!!
Python is interpreted language
```

Program 2:

```
fo = open("test10.txt", "w+")
fo.write( "Python is a great language.\nYeah its great!!\n")
fo.close()
fo = open("test10.txt", "r")
print(fo.readline())
fo.close()
```

Output

```
Python is a great language
```

Program 3:

```
import sys
program_name = sys.argv[0]
```

```
arguments = sys.argv[1:]
count = len(arguments)
print(program_name)
print(sys.argv[2]
print(count)
```

Output: python filename.py 5 8

```
filename.py
8
2
```

11.3.5 File Positions

Tell() Method

The method **tell()** returns the current position of the file read/write pointer within the file.

Syntax

Following is the syntax for **tell()** method

```
fileObject.tell()
```

Return Value

This method returns the current position of the file read/write pointer within the file.

Example

The following example shows the usage of tell() method.

```
This is 1st line
This is 2nd line
This is 3rd line
This is 4th line
This is 5th line
```

Program:

```
# Open a file
fo = open("foo.txt", "rw+")
print "Name of the file: ", fo.name
# Assuming file has following 5 lines
# This is 1st line
# This is 2nd line
# This is 3rd line
# This is 4th line
# This is 5th line

line = fo.readline()
print "Read Line: %s" % (line)
# Get the current position of the file.
pos = fo.tell()
print "Current Position: %d" % (pos)
# Close opend file
fo.close()
```

Output:

```
Name of the file:  foo.txt
Read Line: Python is a great language.
Current Position: 28
```

Seek() Method

The method **seek()** sets the file's current position at the offset. The whence argument is optional and defaults to 0, which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

There is no return value. Note that if the file is opened for appending using either 'a' or 'a+', any seek() operations will be undone at the next write.

If the file is only opened for writing in append mode using 'a', this method is essentially a no-op, but it remains useful for files opened in append mode with reading enabled (mode 'a+').

If the file is opened in text mode using 't', only offsets returned by tell() are legal. Use of other offsets causes undefined behavior.

Note that not all file objects are seekable.

Syntax

Following is the syntax for **seek()** method −

```
fileObject.seek(offset[, whence])
```

Parameters

- **offset** − This is the position of the read/write pointer within the file.
- **whence** − This is optional and defaults to 0 which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

Return Value

This method does not return any value.

Example

The following example shows the usage of seek() method.

```
# Open a file
fo = open("foo.txt", "rw+")
print "Name of the file: ", fo.name
```

Program:

```
line = fo.readline()
print "Read Line: %s" % (line)

# Again set the pointer to the beginning
fo.seek(0, 0)
line = fo.readline()
print "Read Line: %s" % (line)

# Close opend file
fo.close()
```

Output:

Name of the file: foo.txt
Read Line: Python is a great language.

Read Line: Python is a great language.

11.4 ERRORS AND EXCEPTIONS

11.4.1 Errors

Errors or mistakes in a program are often referred to as bugs. They are almost always the fault of the programmer. The process of finding and eliminating errors is called debugging. Errors can be classified into three major groups:

1. Syntax errors
2. Runtime errors
3. Logical errors

1. Syntax errors

Syntax errors are mistakes in the use of the Python language, and are analogous to spelling or grammar mistakes in a language like English: for example, the sentence *Would you some tea?* does not make sense – it is missing a verb.

Common Python syntax errors include:

- leaving out a keyword
- putting a keyword in the wrong place
- leaving out a symbol, such as a colon, comma or brackets
- misspelling a keyword
- incorrect indentation
- empty block

Some examples of syntax errors in Python:

- myfunction(x, y):
  return x + y
- else:
  print("Hello!")
- if mark >= 50
  print("You passed!")
- if arriving:
  print("Hi!")
- esle:
  print("Bye!")
- if flag:

print("Flag is set!")

2. Runtime errors

If a program is syntactically correct – that is, free of syntax errors – it will be run by the Python interpreter. However, the program may exit unexpectedly during execution if it encounters a *runtime error* – a problem which was not detected when the program was parsed, but is only revealed when a particular line is executed.

Examples of Python runtime errors:

- division by zero
- performing an operation on incompatible types

- using an identifier which has not been defined
- accessing a list element, dictionary value or object attribute which doesn’t exist
- trying to access a file which doesn’t exist

3. Logical errors

Logical errors are the most difficult to fix. They occur when the program runs without crashing, but produces an incorrect result. The error is caused by a mistake in the program’s logic.

Some examples of mistakes which lead to logical errors:

- using the wrong variable name
- indenting a block to the wrong level
- using integer division instead of floating-point division
- getting operator precedence wrong
- making a mistake in a boolean expression
- off-by-one, and other numerical errors

11.4.2 Parts in an Error Message

The error message has two parts:

a. the type of error before the colon, and
b. speci_cs about the error after the colon

11.4.3 Exceptions

- An exception is an error that happens during execution of a program. When that error occurs, Python generate an exception that can be handled, which avoids the program to crash.
- 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.
- Whenever a runtime error occurs, it creates an exception. The program stops execution and prints an error message. For example, dividing by zero creates an exception:

  For example

  ```
  print 55/0
  ZeroDivisionError: integer division or modulo
  ```
- When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.
- Program without handling an exception:

  ```
  a = 10
  b = 0
  result = a/b
  print (result)
  ```
- The above code cause zero division error that disrupts the normal flow of the program's instructions.

11.4.4 Built-In Exceptions

- Prohibited operations can elevate exceptions.
- All the built-in exceptions can be analyzed by means of local( ) built in function as follows:
- Syntax:

  >>>locals( ) [‘__builtins__’]

Sr.No.	Exception Name & Description
1	**Exception** Base class for all exceptions
2	**StopIteration** Raised when the next() method of an iterator does not point to any object.
3	**SystemExit** Raised by the sys.exit() function.
4	**StandardError** Base class for all built-in exceptions except StopIteration and SystemExit.
5	**ArithmeticError** Base class for all errors that occur for numeric calculation.
6	**OverflowError** Raised when a calculation exceeds maximum limit for a numeric type.
7	**FloatingPointError** Raised when a floating point calculation fails.
8	**ZeroDivisionError** Raised when division or modulo by zero takes place for all numeric types.
9	**AssertionError** Raised in case of failure of the Assert statement.
10	**AttributeError** Raised in case of failure of attribute reference or assignment.
11	**EOFError** Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
12	**ImportError** Raised when an import statement fails.
13	**KeyboardInterrupt** Raised when the user interrupts program execution, usually by pressing Ctrl+c.
14	**LookupError** Base class for all lookup errors.
15	**IndexError** Raised when an index is not found in a sequence.
16	**KeyError** Raised when the specified key is not found in the dictionary.
17	**NameError** Raised when an identifier is not found in the local or global namespace.
18	**UnboundLocalError**

	Raised when trying to access a local variable in a function or method but no value has been assigned to it.
19	**EnvironmentError** Base class for all exceptions that occur outside the Python environment.
20	**IOError** Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
21	**IOError** Raised for operating system-related errors.
22	**SyntaxError** Raised when there is an error in Python syntax.
23	**IndentationError** Raised when indentation is not specified properly.
24	**SystemError** Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
25	**SystemExit** Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
26	**TypeError** Raised when an operation or function is attempted that is invalid for the specified data type.
27	**ValueError** Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
28	**RuntimeError** Raised when a generated error does not fall into any category.
29	**NotImplementedError** Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

NOTES

- It is achievable to define the own exception in Python. We can handle these built-in and user defined exceptions in Python using try, except and finally statements.

11.4.5 Error Messages that are Displayed for the Following Exceptions

a. Accessing a non-existent list item
IndexError: list index out of range

b. Accessing a key that isn’t in the dictionary
KeyError: what

c. Trying to open a non-existent file
IOError: [Errno 2] No such file or directory: 'filename'

11.5 HANDLING AN EXCEPTION

Python using a key word try to organize a block of code that is expected to produce an error and throw an exception.

- The *defined* except block is used to catch the exception thrown by the try block and handle it.
- The *try* block can contain one or more statements that could produce an exception. If anyone of the statement produces an exception, subsequently the remaining statements in the block are omitted and execution hops to except block which is located next to the try block.
- The *except* block can include more than one statement and if the except argument matches with the type of the Exception object, then the exception is fixed and statements in the except block will be executed.
- Note: Each try block must be tracked by atleast one except statement.

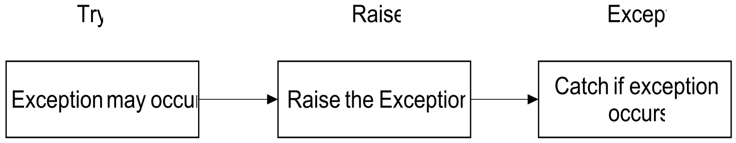


Fig 1:Exception Handling

- While these exceptions happen, it causes the present process to stop and passes it to the calling process until it is handled. If it is not handled properly then the program will stop working.
- For example, if function student( ) calls the function branch( ) which in turn calls the function curricular( ). Consider that the function curricular( ) is met with an exception. While the exception is not handled in curricular( ), then it passes to branch( ) and next to student( ). If not at all handled, an error message emits and the program will come to an abrupt, unpredicted halt.

If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem efficiently as possible.

try: The suspicious code that may raise an exception can be placed in try block.

except: The block of code which handles the problem as elegantly as possible

Syntax

Here is simple syntax of *try....except...else* blocks −

```
try
  You do your operations here;
  ......................
exceptExceptionI:
```

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.

Program with handling an exception:

```
a = 10
b = int(input(“Please enter the integer greater than:”))
try:
        result = a/b
        print (result)
except ZeroDivisionError:
        print (“Error, you have entered zero”)
```

Output 1:

```
Please enter the integer greater than 0: 5
        2.0
```

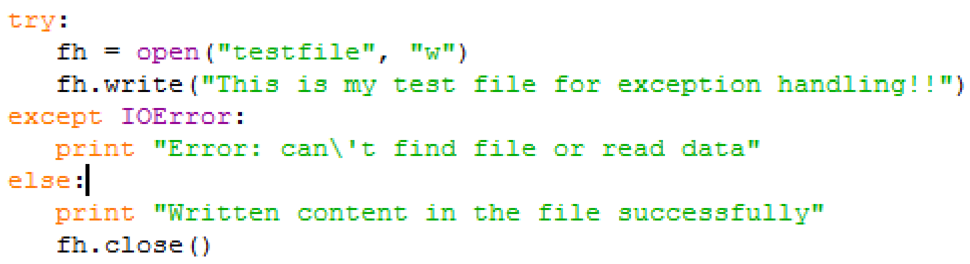
Output 1:

```
Please enter the integer greater than 0: 0
        Error, you have entered zero
```

In the above program, the string value is given as inputs then it produce a ValueError exception.

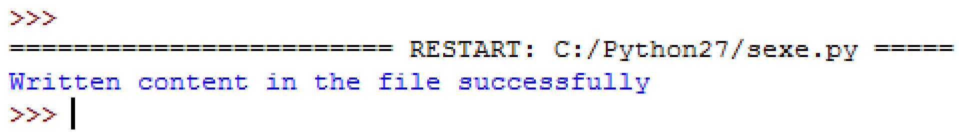
Program 1:

This example opens a file, writes content in the, file and comes out because there is no problem at all:



```
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()
```

Output:

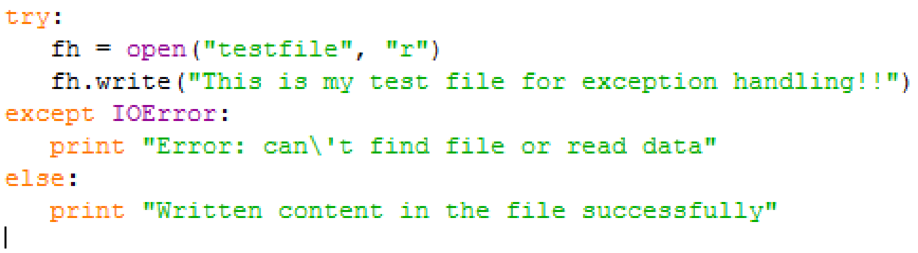


```
>>>
======================== RESTART: C:/Python27/sexe.py =====
Written content in the file successfully
>>>
```

Program 2:

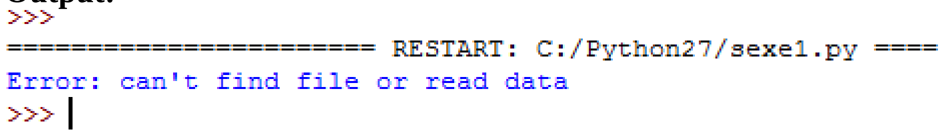
NOTES

This example tries to open a file where you do not have write permission, so it raises an exception –



```
try:
   fh = open("testfile", "r")
   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"
```

Output:



```
>>>
======================= RESTART: C:/Python27/sexe1.py ====
Error: can't find file or read data
>>>
```

11.5.1 except Clause with No Exceptions

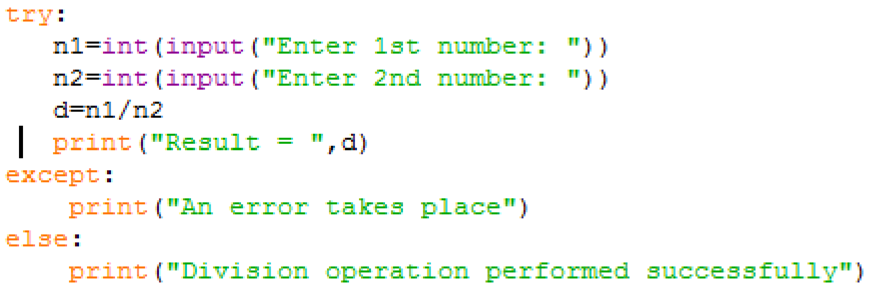
In Python it is possible to use an except statement with no exceptions defined as follow:

Syntax:

```
        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.
```

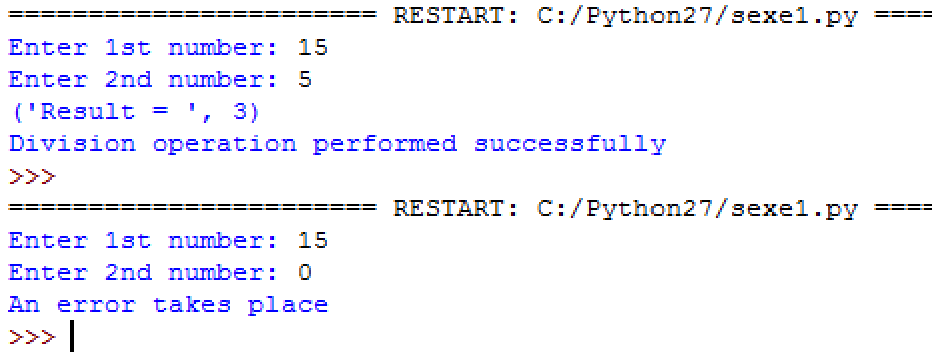
This kind of a **try-except** statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.

Program 3:



```
try:
   n1=int(input("Enter 1st number: "))
   n2=int(input("Enter 2nd number: "))
   d=n1/n2
   print("Result = ",d)
except:
    print("An error takes place")
else:
    print("Division operation performed successfully")
```

Output:



```
====================== RESTART: C:/Python27/sexe1.py ====
Enter 1st number: 15
Enter 2nd number: 5
('Result = ', 3)
Division operation performed successfully
>>>
====================== RESTART: C:/Python27/sexe1.py ====
Enter 1st number: 15
Enter 2nd number: 0
An error takes place
>>> |
```

11.5.2 Handling Multiple Exceptions

To avoid the ValueError exception, multiple except clause can be used.
Program with multiple except clause:

```
a = 10
b = int(input("Please enter the integer greater than:"))
try:
        result = a/b
        print (result)
exceptZeroDivisionError:
        print ("Error, you have entered zero")
exceptValueError:
        print ("Error, Inappropriate value")
```

Output:

```
        Please enter the integer greater than 0: a
         Error, Inappropriate value
```

11.5.3 The except clause with Multiple Exceptions

To handle the multiple exception with the help of tuple as follows.

Program with except clause have a tuple

```
a = 10
b = int(input("Please enter the integer greater than:"))
try:
        result = a/b
        print (result)
except (ZeroDivisionError,ValueError ):
        print ("Error, you have entered zero or inappropriate value")
except:
        print ("Unknown Error")
```

Output:

```
        Please enter the integer greater than 0: a
         Error, you have entered zero or inappropriate value
```

11.5.4 Handling an Exception by try/except/else clause

else clause is used to verify if no exception occurred in try.

Program with else clause have a tuple

```
a = 10
b = int(input("Please enter the integer greater than:"))
try:
        result = a/b
except (ZeroDivisionError,ValueError ):
        print ("Error, you have entered zero or inappropriate value")
except:
        print ("Unknown Error")
else:
        print (result)
```

Output:

```
        Please enter the integer greater than 0: a
         Error, you have entered zero or inappropriate value
```

11.5.5 Handling an Exception by try/except/finally clause

finallyclause is mainly used to close the resources. It must be executed even though exception has been raised.

Program with finally clause have a tuple

```
a = 10
b = int(input("Please enter the integer greater than:"))
result = None
try:
        result = a/b
        print (result)
except (ZeroDivisionError,ValueError ):
        print ("Error, you have entered zero or inappropriate value")
except:
        print ("Unknown Error")
finally:
        print (result)
```

Output:

```
Please enter the integer greater than 0: 0
Error, you have entered zero or inappropriate value
None
```

11.5.6 Raising an Exceptions

The raise statement allows the programmer to force a specified exception to occur.It is used to raise an exception when the program detects an error. It takes two arguments: the exception type and specific information about the error.

The general syntax for the **raise** statement is as follows.

Syntax

```
        raise [Exception [, args [, traceback]]]
```

Here,

- *Exception* is the type of exception (for example, NameError) and
- *args* 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:

```
>>> raise MemoryError
Traceback (most recent call last):
...
MemoryError
>>> raise MemoryError("This is an argument")
Traceback (most recent call last):
...
MemoryError: This is an argument
>>> try:
    a = int(input("Enter a positive integer value: "))
if a <= 0:
raiseValueError("This is not a positive number!!")
exceptValueError as ve:
print(ve)
```

Output:

```
If we enter a negative number:
Enter a positive integer: –5
This is not a positive number!!
```

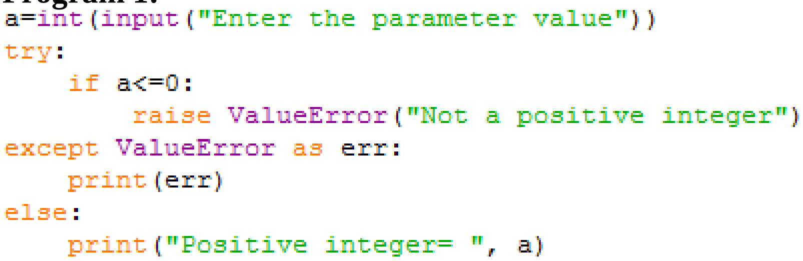
- The sole argument to **raise** indicates the exception to be raised. This must be either an exception instance or an exception class (a class that derives from Exception).
- If an exception class is passed, it will be implicitly instantiated by calling its constructor with no arguments:

raiseValueError

It is the shorthand for *'raise ValueError()'*
A simpler form of the raise statement allows you to re-raise the exception is given below:

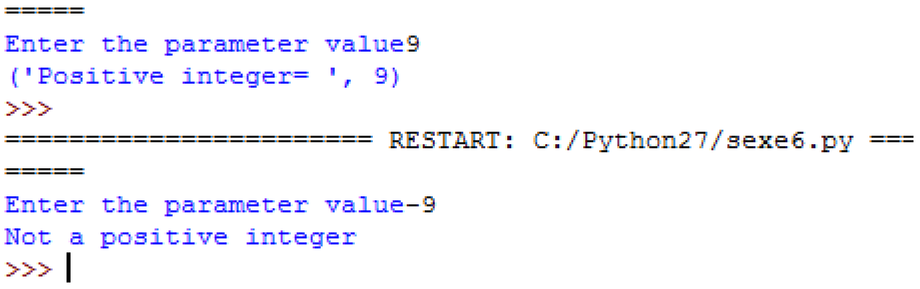
```
>>>
>>> try:
... raiseNameError('HiThere')
... exceptNameError:
... print('An exception flew by!')
... raise
...
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
NameError: HiThere
```

Program 1:



```
a=int(input("Enter the parameter value"))
try:
    if a<=0:
        raise ValueError("Not a positive integer")
except ValueError as err:
    print(err)
else:
    print("Positive integer= ", a)
```

Output:



```
=====
Enter the parameter value9
('Positive integer= ', 9)
>>>
====================== RESTART: C:/Python27/sexe6.py ===
=====
Enter the parameter value-9
Not a positive integer
>>> |
```

11.5.7 User-Defined Exceptions

- Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.
- We can add user-defined exceptions by creating a new class in Python. The trick here is to derive the custom exception class from the base exception class. Most of the built-in exceptions use the same technique to enforce their exceptions.

Program 1:

Creating a user defined exception class which throws an exception when age is less than 18.

```
classMyErrorException(Exception):
def  _str_(self):
                    returnrepr(str(self.errCode) + " " + self.errMessage)
age = int(input("Enter the age:"))
try:
if(age < 18):
raiseMyErrorException(age, " is less than 18")
exceptMyErrorException as exp:
print(exp)
print("Unauthorized for voting")
```

Ouput:

```
Enter the age: 7
7 is less than 18
Unauthorized for voting
```

Explanation:

- **MyErrorException** class is a **user defined exception** which **inherits** the properties from **Exception** class.
- It overrite the **str() function** to display the user defined error message.

- If the **age** value is given **less than 18**, then it **raise**an user defined exception called**MyErrorException**.
- Once MyErrorException is raised, it needs to be **handled in except clause** to display the related error message.

11.6 FUNCTIONS

NOTES

A function is a block of organized and reusable program code that performs a single, specific and well-defined task. It is a group of statements that perform a specific task.

Needs for function:

- If a program is large, it is difficult to understand the steps involved in it. Hence, it is subdivided into a number of small program called subprogram or modules.
- Each subprogram specifies one or more actions to be performed for the larger program. Such subprograms are called as functions.

Advantage of function:

- Reduce the program development time and code.
- It ensures code readability.
- The function can be reused with or without modification when it is needed.

Why use function?

- Maximizing code reuse and minimizing redundancy.
- Procedural decomposition.

Types of function:

There are two types of function. They are,

1. **Built-in/ pre-defined function** – function that are built into python.
2. **User defined function** – functions defined by user.

Built-in function:

The python interpreter has a number of functions that are always available for use. These functions are called Built-in functions.

Built-in function	**Description**
input()	Read a line from input, convert it to a string.
print()	Print objects to the stream.
len()	Return the length of an object.
abs(n)	Return absolute value of a number.
pow(n,d)	Return n raised to the power d.
sqrt(n)	Returns the square root of number.
int()	Convert string, floating point into integer data type.
float()	Convert string, integer into floating point data type.
str()	Convert floating point, integer, list, tuple, and dictionary into string data type.
list()	Convert string, tuple, and dictionary into list data type.
tuples()	Convert string and list into tuple data type.

Table 1: Python built-in function

Program:

```
n=int(input(“enter the number”))
print(“the square root of a number: ”,sqrt(n))
```

Output:

```
Enter the number: 4
The square root of a number: 16
```

User defined function:

The functions defined by the users according to their requirements are called user defined function. The user can modify the function according to their requirements.

Function definition:

The functions are created by user in their programs using the ***def*** keyword. The important points for creating a function as follows as,

- Functions blocks starts with the keyword def.
- The def keyword followed by function name and parentheses “( ) “.
- After the parentheses a colon (:) is placed.
- Parameter or arguments that the function accepts are placed within parentheses.
- Return statement is optional

The function definition includes two parts:

1. **Function header –** it begins with keyword def and ends with colon (:)
2. **Function body –** it consisting of one or more python statements and it is indented.

Syntax:

```
def  functionName(variable1,variable2,….) :
        “””docstring”””
        block of statements
        return [expression]
```

Function call (function invocation):

A function is used within a program through function call. It invokes the function. When function is invoked, then program control jumps to the called function to execute the statements that are part of that function. Once the function is executed, the program control is return back to calling function.

Syntax:

functionName(variable1,variable2,….)   or
functionName( )

Example:	harmonic(n)

Return statements:

A function may or may not return a value. The return statement is used for two things:

1. Return a value to the called function.

2. To end and exit a function and go back to the called function.

Syntax: **return [expression]**

The expression is optional. If the expression is presented, it is evaluated and resultant value is return to the calling function. If no expression is specified then the function will return none.

Program:

```
def add(a,b):
        sum=a+b
        return sum
a=5
b=4
c=add(a,b)
print(“sum:”,c)
```

Output:

```
Sum: 9
```

NOTES

Function with multiple return value:

The function returns more than one value at a time. These multiple return value is considered as tuple.

Syntax: **return value1, value2, value3, …value n**

Program:

```
def arithmetic(a,b):
        sum=a+b
        sub=a-b
        mul=a*b
        return sum,sub,mul
a=5
b=4
c=arithmetic(a,b)
print(“tuple :”,c)
```

Output:

```
tuple: (9, 1, 20)
```

11.7 PARAMETERS

The variables that are passed to the function in function definition are called **parameters**.

Pass by value: (call by value)

Pass by reference: (call by reference)

Pass by Value:

The most common strategy is the call-by-value evaluation, sometimes also called pass-by-value. In call-by-value, the argument expression is evaluated, and the result of this evaluation is bound to the corresponding variable in the function. So, if the expression is a variable, a local copy of its value will be used, i.e. the variable in the caller's scope will be unchanged when the function returns.

```
def change(x):
   x=x+1
   print('Inside change function x = ', x)
x=10
```

```
change(x)
print('Outside change function x = ', x)
```

Output:

```
Inside change function x =  11
Outside change function x =  10
```

Pass by Reference:

In call-by-reference evaluation, which is also known as pass-by-reference, a function gets an implicit reference to the argument, rather than a copy of its value. As a consequence, the function can modify the argument, i.e. the value of the variable in the caller's scope can be changed.

```
def changeme( mylist ):
print ("Values inside the function before change: ", mylist)
mylist[2]=50
print ("Values inside the function after change: ", mylist)
return
mylist = [10,20,30]
changeme( mylist )
print ("Values outside the function: ", mylist)
```

Output:

```
Values inside the function before change: [10, 20, 30]
Values inside the function after change: [10, 20, 50]
Values outside the function: [10, 20, 50]
```

11.8 ARGUMENTS

When the function with parameters is called, the values that are passed to the calling function are called **arguments**.

Function with arguments

Function without arguments

Function arguments:

There are four types of formal arguments. They are:

1. Required arguments
2. Keyword arguments
3. Default arguments
4. Variable-length arguments

1. Required arguments:

The arguments are passed to a function in correct positional order. The number of arguments in the function call should exactly match with the number of arguments specified in the function definition.

Syntax:

```
        def  functionName(variable1,variable2,….) :
                block of statements
                return [expression]
```

Program 1:

```
        def display( name, age, salary):
                print(“name = ”, name)
```

```
          print (“age = ", age)
          print(“salary = ”,salary)
display(“abc”, 20, 3000)
```

Output:

```
Name = abc
Age = 20
Salary = 3000
```

Program 2:

```
def display( name, age, salary):
          print(“name = ”, name)
          print (“age = ", age)
          print(“salary = ”,salary)
display(“abc”, 20)
```

Output:

```
Error: required argument value
          display(“abc”, 20)
                              ^
```

2. Keyword arguments:

The values are assigned to the argument based on their position. By using keyword argument, the order (position) of the argument can be changed. The values are not assigned to arguments according to their position but based on their name (keyword).

Syntax:

functionName(variable1 = value1, variable2 = value2)

Program:

```
def display( name, age, salary):
          print(“name = ”, name)
          print (“age = ", age)
          print(“salary = ”,salary)
display(age = 20, salary = 3000, name =  “abc”)
```

Output:

```
Name = abc
Age = 20
Salary = 3000
```

3. Default arguments:

The default value is assigned to the function argument. The default value to an argument is provided by using the assignment operator (=).

Syntax:

```
def  functionName(variable1,variable2 = value) :
          block of statements
          return [expression]
```

Program:

```
def display( name, age, salary = 4000):
          print(“name = ”, name)
```

```
                print (“age = ", age)
                print(“salary = ”,salary)
        display(“abc”, 20)
        display(“abc”, 20, 5000)
```

Output:

```
        Name = abc
        Age = 20
        Salary = 4000
        Name = abc
        Age = 20
        Salary = 5000
```

4. Variable-length arguments: (arbitrary arguments)

In some situation, it is not known in advanced how many arguments will be passed to a function. In such case, python allows programmers to make function calls with arbitrary (any) number of arguments by using asterisk (*) before the parameter name.

Syntax:

```
        def  functionName( * variable) :
                block of statements
                return [expression]
```

Program:

```
        def display( * var):
                print(“list = ”, var)
        display(“abc”, 20, 3000)
```

Output:

```
        list = [abc,  20, 4000]
```

Function composition:

Function composition is a way of combining function such that the result of each function is passed as the argument of the next function. It is ability to call function from within another function.

Example:

```
        def  f():
                return g()
        def  g( ):
                print(“inner function”)
        f()
```

11.9 FRUITFUL FUNCTION: (FUNCTION RETURNING VALUES)

The function which returns values is called the fruitful function. The statement return [expression] exits in a function pass back an expression to the called function. A return statement with no arguments is return None value.

Program:

```
def sum( a, b):
        total = a + b
        print (“Inside the function : ", total )
        return total
c = sum( 10, 20 )
print (“outside the function : ", c )
```

Output:

```
Inside the function: 30
Outside the function: 30
```

Void function:

The function without return statement is called the void function.

Program:

```
def sum( a, b):
        total = a+b
        print (“Inside the function : ", total )
sum( 10, 20 )
```

Output:

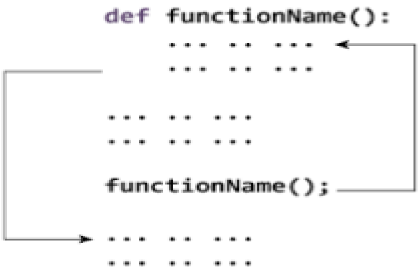
```
Inside the function: 30
```

Flow of execution:

Before using the function, it is important to define it. To define a function, we must know the sequence in which the statement of function should run. This order of statement execution is called as flow of execution.

How function works:

- Once a function is called, it takes some data from the ***calling function*** and returns back some value to the ***called function.***
- Whenever function is called, control passes to the called function and working of the calling function is temporarily stopped, when the execution of the called function is completed then control returns back to the calling function and execute the next statements.
- The function operates on formal and actual arguments and sends back the result to the calling function using return statement.



```
def functionName():
    ... .. ...
    ... .. ...

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

functionName();

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

Fig: calling a function

11.10 VARIABLE SCOPE AND LIFETIME:

- **Scope of the variable:** part of the program in which a variable is accessible is called its scope.

- **Lifetime of the variable:** duration for which the variable exists is called its lifetime.

Local and global variable:

- **Global variable:** variables are defined in main body of the program file. They are access throughout the program file. It is accessible to all function.
- **Local variable:** variable is defined within a function is local variable to that function. A local variable can be accessed only within (inside) the function. It is not accessible to other function.

Program:

```
num=10                          #num = global variable
print("the global variable num=", num)
def func(num2):                        #num2 = local variable
        print("the local variable num2=", num2))
        num3=30                 #num3 = local variable
        print("the local variable num3=", num3))
func(20)
print("Accessing global variable num=", num)
```

Output:

```
the global variable num=10
the local variable num2=20
the local variable num3=30
Accessing global variable num=10
```

Using global variable:

The variable is defined inside a function as global by using keyword global.

Program:

```
num=10                          #num = global variable
print("the global variable num=", num)
def func(num2):                        #num2 = local variable
        print("the local variable num2=", num2))
        global num3=30                 #num3 = global variable
        print("the global variable num3=", num3))
func(20)
print("Accessing global variable num=", num)
print("Accessing global variable num3=", num3)
```

Output:

```
the global variable num=10
the local variable num2=20
the global variable num3=30
Accessing global variable num=10
Accessing global variable num3=10
```

11.11 FUNCTION RECURSION:

When a function call itself is knows as recursion. Recursion works like loop but sometimes it makes more sense to use recursion than loop. You can convert any loop to recursion.This condition is known as base condition. A base condition is must in every recursive programs otherwise it will continue to execute forever like an infinite loop.

Overview of how recursive function works

1. Recursive function is called by some external code.
2. If the base condition is met then the program do something meaningful and exits.
3. Otherwise, function does some required processing and then call itself to continue recursion.

Here is an example of recursive function used to calculate factorial.

```
def fact(n):
   if n == 0:
      return 1
   else:
      return n * fact(n-1)
print(fact(0))
print(fact(5))
```

Expected Output:

```
1
120
```

Lambda function: (anonymous function)

In lambda function, the function is not declared by using the keyword ***def.*** Rather than the function created by using the ***lambda*** keyword.

Syntax:

Lambda arguments : expression

Program:

```
a=10
sum = lambda a : a + 5
print("sum = ",sum)
```

Output:

```
Sum = 15
```

11.12 MODULES AND PACKAGES

11.12.1 Modules

- Modules are files containing Python definitions and statements (ex. *name*.py).A module's definitions can be imported into other modules by using "import *name*".
- The module's name is available as a global variable value. To access a module's functions, type *"name. function()"*
- Modules can contain executable statements along with function definitions. Each module has its own private symbol table used as the global symbol table by all functions in the module. Modules can import other modules.

- Each module is imported once per interpreter session
  - *reload(name)*
- Can import names from a module into the importing module’s symbol table
  - *from mod import m1, m2 (*or **)*
  - *m1()*
- *python name.py <arguments>*
  - Runs code as if it was imported
  - Setting *_name_ == “_main_” the file can be used as a script and an importable module*

11.12.1.1 The import Statement

- The import Statement can use any Python source file as a module by executing an import statement in some other Python source file.
- When the interpreter encounters an import statement, it imports the module if the module is present in the search path.
- A search path is a list of directories that the interpreter searches before importing a module. A module is loaded only once, regardless of the number of times it is imported.
- This prevents the module execution from happening repeatedly, if multiple imports occur.
- The import has the following syntax-
  import module1[, module2[,... moduleN]
- The from...import Statement imports specific attributes from a module into the current namespace. The from...import has the following syntax-
  frommodname import name1[, name2[, ... nameN]]

For example, to import the function fibonacci from the module fib, use the following statement-

```
#!/usr/bin/python3
        # Fibonacci numbers module
        def fib(n):                # return Fibonacci series up to n
        result = [ ]
        a, b = 0, 1
        while b < n:
        result.append(b)
        a, b = b, a+b
        return result
        >>>from fib import fib
        >>>fib(100)
        [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

This statement does not import the entire module fib into the current namespace; it just introduces the item fibonacci from the module fib into the global symbol table of the importing module.

- The from...import * Statement provides an easy way to import all the items from a module into the current namespace; however, this statement should be used sparingly.
- It is also possible to import all names from a module into the current namespace by using the following import statement.
- The from...import * Statement has the following syntax- from modname import *

```
>>>from fib import *
>>>fib1(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>>fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>>
```

Here the statement from fib import * imports all functions defined inside the module fib.py

11.12.1.2 Standard Modules

Python comes with a library of standard modules described in the Python Library Reference. Some are built into interpreter. For example

```
>>> import sys
>>> sys.s1
'>>> '
>>> sys.s1 = 'c> '
c> print 'Hello'
Hello
c>
```

sys.path determines the interpreters's search path for modules, with the default path taken from PYTHONPATH.It can be modified with append() (ex. *Sys.path.append('SOMEPATH')*

dir() **Function**

- Used to find the names a module defines and returns a sorted list of strings
  - *>>> import mod*
    >>>dir(mod)
    ['_name_', 'm1', 'm2']
- Without arguments, it lists the names currently defined (variables, modules, functions, etc)
- Does not list names of built-in functions and variables

Use *_bulltin_*to view all built-in functions and varia

11.13 CLASSES AND OOP

Python is an object oriented programming language. Unlike procedure oriented programming, where the main emphasis is on functions, object oriented programming stress on objects.Object is simply a collection of data (variables) and methods (functions) that act on those data. And, class is a blueprint for the object.We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object.

As, many houses can be made from a description, we can create many objects from a class. An object is also called an instance of a class and the process of creating this object is called instantiation.

11.13.1 Defining a Class in Python

Like function definitions begin with the keyword def, in Python, we define a class using the keyword class. The first string is called docstring and has a brief description about the class. Although not mandatory, this is recommended. Here is a simple class definition.

```
class MyNewClass:

    '''This is a docstring. I have created a new class'''

    pass
```

A class creates a new local namespace where all its attributes are defined. Attributes may be data or functions. There are also special attributes in it that begins with double underscores For example, doc gives us the do string of that class. As soon as we define a class, a new class object is created with the same name. This class object allows us to access the different attributes as well as to instantiate new objects of that class.

11.14 EXECUTION ENVIRONMENT

The interpreter has a number of options that control its runtime behavior

and environment. Options are given to the interpre ter on the command line as follows:

```
python [options] [-c cmd | filename | - ] [args]
```

Here's a list of the most common command-line options:

S.No.	Option & Description
1	**-d** It provides debug output.
2	**-O** It generates optimized bytecode (resulting in .pyo files).
3	**-S** Do not run import site to look for Python paths on startup.
4	**-v** verbose output (detailed trace on import statements).
5	**-X** disable class-based built-in exceptions (just use strings); obsolete starting with version 1.6.
6	**-c cmd** run Python script sent in as cmd string
7	**file** run Python script from given file

Check your Progress

1. What are Files?
2. What are the types of file?
3. What are Errors?
4. What is User-Defined Exceptions?
5. What is Function Invocation?

NOTES

11.15 ANSWERS TO CHECK YOUR PROGRESS

1. File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory

2. In Python, a file is categorized as:
   - Text file
   - Binary file
3. Errors or mistakes in a program are often referred to as bugs. They are almost always the fault of the programmer. The process of finding and eliminating errors is called debugging. Errors can be classified into three major groups:
   - Syntax errors.
   - Runtime errors.
   - Logical errors.
4. Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions. We can add user-defined exceptions by creating a new class in Python. The trick here is to derive the custom exception class from the base exception class. Most of the built-in exceptions use the same technique to enforce their exceptions
5. A function is used within a program through function call. It invokes the function. When function is invoked, then program control jumps to the called function to execute the statements that are part of that function. Once the function is executed, the program control is return back to calling function.

11.16 SUMMARY

- Text files are structured as a **sequence of lines**, where each line includes a sequence of characters.
- File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).
- Errors can be classified into three major groups: Syntax errors, Runtime errors and Logical errors
- The error message has two parts: type of error before the colon, and speci_cs about the error after the colon
- Python using a key word try to organize a block of code that is expected to produce an error and throw an exception.

11.17 KEYWORDS

Seek (): It sets the file's current position at the offset. The whence argument is optional and defaults to 0, which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

NOTES

Errors: Errors or mistakes in a program are often referred to as bugs. They are almost always the fault of the programmer.

ValueError: It is raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.

Exception: An exception is an error that happens during execution of a program. When that error occurs, Python generate an exception that can be handled, which avoids the program to crash.

11.18 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer questions:

1. What are parameters?
2. What are arguments?
3. What are fruitful functions?
4. Explain about Local and Global variable?
5. Explain about Function Recursion?

Long Answer questions:

1. Explain about Modules and Packages?
2. Explain about Arguments?

Explain about Functions?

11.19 FURTHER READINGS

Rémy Card, Eric Dumas, and Franck Mével. The Linux kernel book. John Wiley & Sons, Inc., 2003.

Steve Suchring. MySQL BBible. John Wiley, 2002.

Rasmus Lerdorf and Levin Tatroe. Programming PHP. " O'Reilly Media, Inc., 2002.

Wesley J. Chun. Core Python Programming. Prentice Hall, 2001.

Martin C. Brown. Perl: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Steven Holzner. PHP: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Vikram Vaswani. MySQL: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

BLOCK – V
OPEN SOURCE PROGRAMMING LANGUAGE-PERL

UNIT 12
PERL

Structure

12.0 INTRODUCTION

Open source programming languages are freely available to the user one such software is Perl Programming language which consists of many packages for the web development creating a platform for all users to work with it.

12.1 OBJECTIVE

This unit helps the user to learn and understand the Perl

- Parsing Rules
- Variables and Datatypes

12.2 PERL BACKGROUNDER

Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development, and more. Perl is a stable, cross platform programming language. Though Perl is not officially an acronym but few people used it as Practical Extraction and Report Language. It is used for mission critical projects in the public and private sectors. Perl is Open Source software, licensed under its Artistic License, or the GNU General Public License (GPL).Perl was created by Larry Wall. Perl 1.0 was released to usenet's alt.comp.s ources in 1987.At the time of writing this tutorial, the latest version of Perl was 5.16.2.Perl is listed in the Oxford English Dictionary. PC Magazine announced Perl as the finalist for its 1998 Technical Excellence Award in the Development Tool category.

12.3 PERL OVERVIEW

- Perl takes the best features from other languages, such as C, awk, sed, sh, and BASIC, among others.
- Perls database integration interface DBI supports third-party databases including Oracle, Sybase, Postgres, MySQL and others.
- Perl works with HTML, XML, and other mark-up languages.
- Perl supports Unicode.
- Perl is Y2K compliant.
- Perl supports both procedural and object-oriented programming.
- Perl interfaces with external C/C++ libraries through XS or SWIG.
- Perl is extensible. There are over 20,000 third party modules available from the Comprehensive Perl Archive Network (CPAN).
- The Perl interpreter can be embedded into other system
- Perl used to be the most popular web programming language due to its text manipulation capabilities and rapid development cycle.
- Perl is widely known as "the duct-tape of the Internet".
- Perl can handle encrypted Web data, including e-commerce transactions.
- Perl can be embedded into web servers to speed up processing by as much as 2000%.
- Perl's mod_perl allows the Apache web server to embed a Perl interpreter.
- Perl's DBI package makes web-database integration easy.

Perl is an interpreted language, which means that your code can be run as is, without a compilation stage that creates a non-portable executable program. Traditional compilers convert programs into machine language. When you run a Perl program, it's first compiled into a byte code, which is then converted (as the program runs) into machine instructions. So it is not quite the same as shells, or Tcl, which are strictly interpreted without an intermediate representation. It is also not like most versions of C or C++, which are compiled directly into a machine dependent format. It is somewhere in between, along with Python and awk and Emacs .elc files.

12.4 PARSING RULES

The Swedish Chef lex grammar, by John Hagerman, is a great example of a simple text filter. It's also a lot of fun, having amused many computer science and engineering students on the eve of their finals. I will show an example of porting the chef's grammar to Perl using the Parse::RecDescent module (not the ideal choice for this task -- the Parse::Lex module would be a better one). This section is intended only as an introduction to the rules of building a Parse::RecDescent syntax and will include actions, remembering the state, rejecting productions, and lexing text. Remember to try the chef.pl script yourself -- you may just find yourself addicted.

The chef.pl script is an almost exact copy of the chef.l lex grammar. The $niw variable is set to 0 at startup, because many rules test it to see if they should be accepted or rejected. $niw stands for "not in word", and it is set to 1 when the parser is inside a word. The directive to Parse::RecDescent will reject a rule if the variable named in the directive is non-zero. So keep in mind that $niw = 0 means that you are not inside a word.

The skip variable was set to '' (empty string), so all input, including spaces, goes to the token directive. Also, the chef rule ends on \z, which is the end of the string. Usually, \Z is used, but that can also match a newline in Perl, and those may also be in the input.

The chef rule: The grammar begins with the chef rule. The chef rule matches a number of tokens, up to the \z end of string. Those two elements of the chef rule are called "productions." Any rule must be made up of productions. An action can be part of a production; it is marked by braces {} and contains Perl code. It doesn't match anything -- it is simply executed.

The token rule: The token rule can match any of a number or sequences, somewhat arbitrarily, which I named to match the chef.l grammar. I'll explain a few examples, so that the grammar correspondence is clear.

The basic grammar definitions of a word/non-word character:

```
chef.pl: WC: /[A-Za-z']/

chef.pl: NW: /[^A-Za-z']/

chef.l:  WC  [A-Za-z']

chef.l:  NW  [^A-Za-z']
```

The an rule: The simplest rules do not depend on anything. The an rule is a good example: Whenever it sees 'an', it prints 'un'. Also, it sets $niw to 1 (remember, that means you are inside a word).

```
chef.pl: an: /an/ { $niw = 1; print 'un' }

chef.l:  "an"       { BEGIN INW; printf("un"); }
```

The ax rule: The next more complex rule is the ax rule. It says, if an 'a' shows up, and is followed by a word character WC, print 'e'. The ...WC production syntax means that a word character must follow the a, but will not be consumed in the match. Thus, 'aan' produces 'eun' with an and ax rules. The rule sets $niw to 1 (inside a word).

```
chef.pl: ax: /a/ ...WC { $niw = 1; print "e" }

chef.l:  "a"/{WC}   { BEGIN INW; printf("e"); }
```

The en rule: The en rule works exactly like the ax rule, but with a NW (non-word) production anticipated to follow. This means that the 'en' must be at the end of a word.

```
chef.pl: en: /en/ ...NW { $niw = 1; print "ee" }

chef.l:  "en"/{NW}  { BEGIN INW; printf("ee"); }
```

The ew rule: The ew rule succeeds only if you are inside a word. That's why you reject it if $niw is 0.

```
chef.pl: ew:  /ew/ { $niw = 1; print "oo" }

chef.l:  "ew"   { BEGIN INW; printf("oo"); }
```

The i rule: The i rule will succeeds only if you are inside a word, and have not seen another i yet. It augments $i_seen to 1, and $i_seen is only set back to 0 if a non-word character or a newline are seen.

```
chef.pl: i:   /i/ { $niw=1;$i_seen=1; print "ee" }

chef.l:  "i"    { BEGIN INW; printf(i_seen++ ? "i" : "ee"); }
```

The end of sentence rule: The end of sentence markers [.!?] in any quantity, followed by space, will be printed, followed by the famous (or infamous, take your pick) "Bork Bork Bork!" message. The actual behavior deviates slightly from the original chef filter, only because I like it better that way (one can never have enough Bork messages). The $item[1] syntax means that the spaces won't be matched, since they are known as $item[2] to Parse::RecDescent.

```
chef.pl: end_of_sentence: /[.?!]+/ /\s+/
        { $niw = 0; $i_seen = 0;
          print $item[1] . "\nBork Bork Bork!\n" }
chef.l:  [.!?]$
        { BEGIN NIW;
          i_seen = 0;
          printf("%c\nBork Bork Bork!",
          yytext[0]);
        }
```

The extend-grammar process rule: The process rule can consist of a query or a definition. Unless it finds one or the other, it triggers a message in the main loop. The extend-grammar query rules: The query rule consists of a do you know production, followed by a name or a proper name. For a name, the action is to print a message that it is unknown. For a proper name (as defined by the proper name rule), the action is to print out a message that it is known. Two rules of the same name are equivalent to one rule with two alternative productions, the extend-grammar definition rule: The heart of this extensible grammar is the definition rule. If a name is followed by 'exists', then the action will extend the parser with a new rule for proper name. You can modify the very grammar you are executing, while it's running.

12.5 VARIABLES

Variables are the reserved memory locations to store values. This means that when you create a variable you reserve some space in memory. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or strings in these variables. We have learnt that Perl has the following three basic data types −

- Scalars
- Arrays
- Hashes

Accordingly, we are going to use three types of variables in Perl. A scalar variable will precede by a dollar sign ($) and it can store either a number, a string, or a reference. An array variable will precede by sign @ and it will store ordered lists of scalars. Finaly, the Hash variable will precede by sign % and will be used to store sets of key/value pairs.Perl maintains every variable type in a separate namespace. So you can, without fear of conflict, use the same name for a scalar variable, an array, or a hash. This means that $foo and @foo are two different variables.

12.5.1 Creating Variables

Perl variables do not have to be explicitly declared to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables. Keep a note that this is mandatory to declare a variable before we use it if we use strict statement in our program. The operand to the left of the = operator is the name of the variable, and the operand to the right of the = operator is the value stored in the variable. For example

```
$age = 25;             # An integer assignment
$name = "John Paul";   # A string
$salary = 1445.50;     # A floating point
```

Here 25, "John Paul" and 1445.50 are the values assigned to $age, $name and $salary variables, respectively. Shortly we will see how we can assign values to arrays and hashes.

12.5.2 Scalar Variables

A scalar is a single unit of data. That data might be an integer number, floating point, a character, a string, a paragraph, or an entire web page. Simply saying it could be anything, but only a single thing. Here is a simple example of using scalar variables

Live Demo

```
#!/usr/bin/perl
```

```
$age = 25;             # An integer assignment
$name = "John Paul";   # A string
$salary = 1445.50;     # A floating point
print "Age = $age\n";
print "Name = $name\n";
print "Salary = $salary\n";
```

This will produce the following result −

```
Age = 25
Name = John Paul
Salary = 1445.5
```

12.5.3 Array Variables

An array is a variable that stores an ordered list of scalar values. Array variables are preceded by an "at" (@) sign. To refer to a single element of an array, you will use the dollar sign ($) with the variable name followed by the index of the element in square brackets. Here is a simple example of using array variables

Live Demo

```
#!/usr/bin/perl
@ages = (25, 30, 40);
@names = ("John Paul", "Lisa", "Kumar");
print "\$ages[0] = $ages[0]\n";
print "\$ages[1] = $ages[1]\n";
print "\$ages[2] = $ages[2]\n";
print "\$names[0] = $names[0]\n";
print "\$names[1] = $names[1]\n";
print "\$names[2] = $names[2]\n";
```

Here we used escape sign (\) before the $ sign just to print it. Other Perl will understand it as a variable and will print its value. When executed, this will produce the following result −

```
$ages[0] = 25
$ages[1] = 30
$ages[2] = 40
$names[0] = John Paul
$names[1] = Lisa
$names[2] = Kumar
```

12.5.4 Hash Variables

A hash is a set of key/value pairs. Hash variables are preceded by a percent (%) sign. To refer to a single element of a hash, you will use the hash variable name followed by the "key" associated with the value in curly brackets. Here is a simple example of using hash variables

Live Demo

```
#!/usr/bin/perl
%data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);
print "\$data{'John Paul'} = $data{'John Paul'}\n";
print "\$data{'Lisa'} = $data{'Lisa'}\n";
print "\$data{'Kumar'} = $data{'Kumar'}\n";
```

This will produce the following result −

```
$data{'John Paul'} = 45
$data{'Lisa'} = 30
$data{'Kumar'} = 40
```

12.5.5 Variable Context

Perl treats same variable differently based on Context, i.e., situation where a variable is being used. Let's check the following example

Live Demo

```
#!/usr/bin/perl
@names = ('John Paul', 'Lisa', 'Kumar');
@copy = @names;
$size = @names;
print "Given names are : @copy\n";
print "Number of names are : $size\n";
```

This will produce the following result −

```
Given names are : John Paul Lisa Kumar
Number of names are : 3
```

Here @names is an array, which has been used in two different contexts. First we copied it into any other array, i.e., list, so it returned all the elements assuming that context is list context.

12.6 DATATYPES

Perl is a loosely typed language and there is no need to specify a type for your data while using in your program. The Perl interpreter will choose the type based on the context of the data itself.

Perl has three basic data types: scalars, arrays of scalars, and hashes of scalars, also known as associative arrays. Here is a little detail about these data types.

Scalars are simple variables. They are preceded by a dollar sign ($). A scalar is either a number, a string, or a reference. A reference is actually an address of a variable, which we will see in the upcoming chapters.

Arrays are ordered lists of scalars that you access with a numeric index, which starts with 0. They are preceded by an "at" sign (@).

Hashes-Hashes are unordered sets of key/value pairs that you access using the keys as subscripts. They are preceded by a percent sign (%).

Numeric Literals-Perl stores all the numbers internally as either signed integers or double-precision floating-point values. Numeric literals are specified in any of the following floating-point or integer formats

String Literals-Strings are sequences of characters. They are usually alphanumeric values delimited by either single (') or double (") quotes. They work much like UNIX shell quotes where you can use single quoted strings and double quoted strings.

Double-quoted string literals allow variable interpolation, and single-quoted strings are not. There are certain characters when they are proceeded by a back slash, have special meaning and they are used to represent like newline (\n) or tab (\t).You can embed newlines or any of the following Escape sequences directly in your double quoted strings

Escape sequence	**Meaning**
\\	Backslash
\'	Single quote
\"	Double quote
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\0nn	Creates Octal formatted numbers
\xnn	Creates Hexideciamal formatted numbers
\cX	Controls characters, x may be any character
\u	Forces next character to uppercase
\l	Forces next character to lowercase
\U	Forces all following characters to uppercase
\L	Forces all following characters to lowercase

\Q	Backslash all following non-alphanumeric characters
\E	End \U, \L, or \Q

Example

Let's see again how strings behave with single quotation and double quotation. Here we will use string escapes mentioned in the above table and will make use of the scalar variable to assign string values.

Live Demo

```
#!/usr/bin/perl
# This is case of interpolation.
$str = "Welcome to \ntutorialspoint.com!";
print "$str\n";
# This is case of non-interpolation.
$str = 'Welcome to \ntutorialspoint.com!';
print "$str\n";

# Only W will become upper case.
$str = "\uwelcome to tutorialspoint.com!";
print "$str\n";
# Whole line will become capital.
$str = "\UWelcome to tutorialspoint.com!";
print "$str\n";
# A portion of line will become capital.
$str = "Welcome to \Ututorialspoint\E.com!";
print "$str\n";
# Backsalash non alpha-numeric including spaces.
$str = "\QWelcome to tutorialspoint's family";
print "$str\n";
```

This will produce the following result −

```
Welcome to
```

```
tutorialspoint.com!

Welcome to \ntutorialspoint.com!

Welcome to tutorialspoint.com!

WELCOME TO TUTORIALSPOINT.COM!

Welcome to TUTORIALSPOINT.com!

Welcome\ to\ tutorialspoint\'s\ famil
```

Check your Progress

1. What is Perl?
2. What are Productions in Perl?
3. What is the Token rule?
4. What are three basic data types in Perl?
5. What are Scalar Variables?

12.7 ANSWERS TO CHECK YOUR PROGRESS

1. Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development, and more.
2. The grammar begins with the chef rule. The chef rule matches a number of tokens, up to the \z end of string. Those two elements of the chef rule are called "productions." Any rule must be made up of productions. An action can be part of a production; it is marked by braces {} and contains Perl code. It doesn't match anything -- it is simply executed.
3. The token rule can match any of a number or sequences, somewhat arbitrarily, which I named to match the chef.l grammar.
4. Perl has the following three basic data types:
    - Scalars
    - Arrays
    - Hashes
5. A scalar is a single unit of data. That data might be an integer number, floating point, a character, a string, a paragraph, or an entire web page. Simply saying it could be anything, but only a single thing.

12.8 SUMMARY

- Perl has three basic data types: scalars, arrays of scalars, and hashes of scalars, also known as associative arrays.
- An array is a variable that stores an ordered list of scalar values.

- A scalar is a single unit of data. That data might be an integer number, floating point, a character, a string, a paragraph, or an entire web page.
- Perl variables do not have to be explicitly declared to reserve memory space.

NOTES

12.9 KEYWORDS

The extend-grammar process rule: The process rule can consist of a query or a definition.

Hashes-Hashes are unordered sets of key/value pairs that you access using the keys as subscripts. They are preceded by a percent sign (%).

Numeric Literals-Perl stores all the numbers internally as either signed integers or double-precision floating-point values. Numeric literals are specified in any of the following floating-point or integer formats

Until loop: Repeats a statement or group of statements until a given condition becomes true. It tests the condition before executing the loop body.

12.10 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer questions:

1. What are Array Variables?
2. What are Hash Variables?
3. What is Variable context?
4. What are Datatypes?
5. What are String Literals?

Long Answer questions:

1. Explain about datatypes and its basic data types?
2. Explain about Variables and its basic datatypes?
3. Explain about Parsing Parsing Rules?

12.11. FURTHER READINGS

Rémy Card, Eric Dumas, and Franck Mével. The Linux kernel book. John Wiley & Sons, Inc., 2003.

Steve Suchring. MySQL BBible. John Wiley, 2002.

Rasmus Lerdorf and Levin Tatroe. Programming PHP. " O'Reilly Media, Inc., 2002.

Wesley J. Chun. Core Python Programming. Prentice Hall, 2001.

Martin C. Brown. Perl: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Steven Holzner. PHP: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Vikram Vaswani. MySQL: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

UNIT 13
CONTROL STRUCTURES

Structure

13.0 INTRODUCTION

The control structures are important statements in programming languages. This unit explains the basic control structures and subroutines of Perl Programming Languages. The different control statements and their syntax are discussed

13.1 OBJECTIVE

This unit helps the user to start their Perl programming by understanding

- Control Structures
- Subroutines

13.2 CONTROL STRUCTURES

Perl conditional statements helps in the decision making, which require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false. Following is the general from of a typical decision making structure found in most of the programming languages

ures

13.2.1 Decision making statements in Perl

The number 0, the strings '0' and "" , the empty list () , and undef are all false in a boolean context and all other values are true. Negation of a true value by! Or not returns a special false value. Perl programming language provides the following types of conditional statements.

1**. if statement**-An if statement consists of a boolean expression followed by one or more statements.

2.**if...else statement**- An if statement can be followed by an optional else statement.

3.**if...elsif...else statement** -An if statement can be followed by an optional elsif statement and then by an optional else statement.

4.**unless statement**-An unless statement consists of a boolean expression followed by one or more statements.

5. **unless...else statement**-An unless statement can be followed by an optional else statement.

6. **unless...elsif..else statement**-An unless statement can be followed by an optional elsif statement and then by an optional else statement.

7. **switch statement**-With the latest versions of Perl, you can make use of the switch statement. which allows a simple way of comparing a variable value against various conditions.

8. The ? : Operator-Let's check the conditional operator ? :which can be used to replace if...else statements. It has the following general form

Exp1 ? Exp2 : Exp3;

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

The value of a ? Expression is determined like this: Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression. If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression. Below is a simple example making use of this operator?

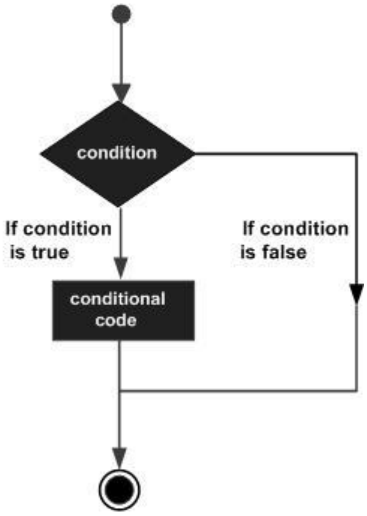


Figure 13.1 Control structure

Live Demo

```
#!/usr/local/bin/perl
 $name = "Ali";
$age = 10;
$status = ($age > 60 )? "A senior citizen" : "Not a senior citizen";
print "$name is  - $status\n";
```

This will produce the following result −

```
Ali is - Not a senior citizen
```

Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages programming language provides the following types of loop to handle the looping requirements.

While loop-Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

Until loop-Repeats a statement or group of statements until a given condition becomes 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.

foreach loop-The foreach loop iterates over a normal list value and sets the variable VAR to be each element of the list in turn.

do...while loop-Like a while statement, except that it tests the condition at the end of the loop body.

nested loops-You can use one or more loop inside any another while, for or do..while loop.

Loop Control Statements-Loop control statements change the execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

next statement-Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

last statement-Terminates the loop statement and transfers execution to the statement immediately following the loop.

continue statement-A continue BLOCK, it is always executed just before the conditional is about to be evaluated again.

redo statement-The redo command restarts the loop block without evaluating the conditional again. The continue block, if any, is not executed.

goto statement-Perl supports a goto command with three forms: goto label, goto expr, and goto &name.

The Infinite Loop-A loop becomes infinite loop if a condition never becomes false. The for loop is traditionally used for this purpose. Since none of the three expressions that form the for loop are required, you can make an endless loop by leaving the conditional expression empty.

```
#!/usr/local/bin/perl

 for( ; ; ) {

   printf "This loop will run forever.\n";

}
```

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but as a programmer more commonly use the for (;;) construct to signify an infinite loop.

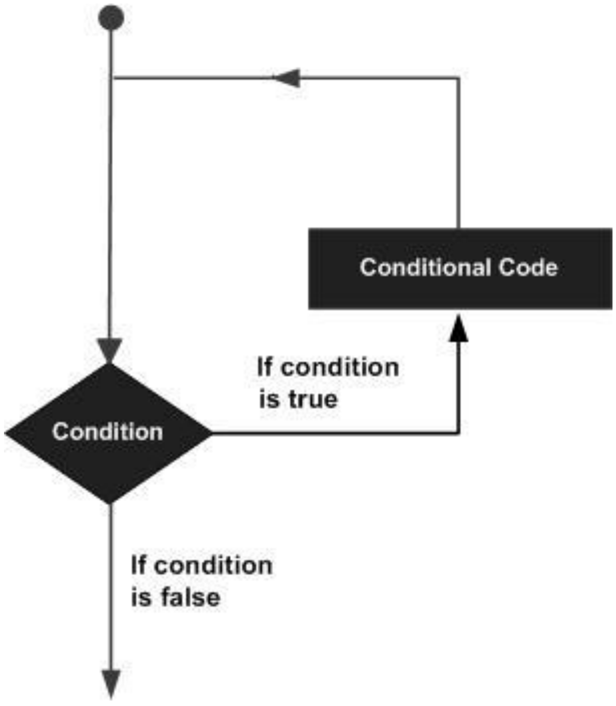


Figure 12.2 Condition block

13.3 SUBROUTINES

A Perl subroutine or function is a group of statements that together performs a task. You can divide up your code into separate subroutines. How you divide up your code among different subroutines is up to you, but logically the division usually is so each function performs a specific task.

Perl uses the terms subroutine, method and function interchangeably.

13.3.1 Define and Call a Subroutine

The general form of a subroutine definition in Perl programming language is as follows

```
sub subroutine_name {

 body of the subroutine

}
```

The typical way of calling that Perl subroutine is as follows

```
subroutine_name( list of arguments );
```

In versions of Perl before 5.0, the syntax for calling subroutines was slightly different as shown below. This still works in the newest versions of Perl, but it is not recommended since it bypasses the subroutine prototypes.

```
&subroutine_name( list of arguments );
```

Let's have a look into the following example, which defines a simple function and then call it. Because Perl compiles your program before executing it, it doesn't matter where you declare your subroutine.

Live Demo

```
#!/usr/bin/perl

# Function definition

sub Hello {

   print "Hello, World!\n";

}

# Function call

Hello();
```

When above program is executed, it produces the following result −

```
Hello, World!
```

13.3.2 Passing Arguments to a Subroutine

You can pass various arguments to a subroutine like you do in any other programming language and they can be acessed inside the function using the special array @_. Thus the first argument to the function is in $_[0], the second is in $_[1], and so on.

You can pass arrays and hashes as arguments like any scalar but passing more than one array or hash normally causes them to lose their separate identities. So we will use references (explained in the next chapter) to pass any array or hash. Let's try the following example, which takes a list of numbers and then prints their average.

Live Demo

```
#!/usr/bin/perl
# Function definition
sub Average {
   # get total number of arguments passed.
   $n = scalar(@_);
   $sum = 0;
   foreach $item (@_) {
      $sum += $item;
   }
   $average = $sum / $n;
   print "Average for the given numbers : $average\n";
}
# Function call
Average(10, 20, 30);
```

When above program is executed, it produces the following result

```
Average for the given numbers : 20
```

13.3.3 Passing Lists to Subroutines

Because the @_ variable is an array, it can be used to supply lists to a subroutine. However, because of the way in which Perl accepts and parses lists and arrays, it can be difficult to extract the individual elements from @_. If you have to pass a list along with other scalar arguments, then make list as the last argument as shown below

Live Demo

```
#!/usr/bin/perl
# Function definition
sub PrintList {
   my @list = @_;
   print "Given list is @list\n";
}
$a = 10;
@b = (1, 2, 3, 4);
# Function call with list parameter
PrintList($a, @b);
```

When above program is executed, it produces the following result

```
Given list is 10 1 2 3 4
```

13.3.4 Passing Hashes to Subroutines

When you supply a hash to a subroutine or operator that accepts a list, then hash is automatically translated into a list of key/value pairs. For example

Live Demo

```
#!/usr/bin/perl
# Function definition
sub PrintHash {
   my (%hash) = @_;
   foreach my $key ( keys %hash ) {
      my $value = $hash{$key};
      print "$key : $value\n";
   }
}
%hash = ('name' => 'Tom', 'age' => 19);
# Function call with hash parameter
PrintHash(%hash);
```

When above program is executed, it produces the following result −

```
name : Tom
age : 19
```

13.3.5 Returning Value from a Subroutine

You can return a value from subroutine like you do in any other programming language. If you are not returning a value from a subroutine then whatever calculation is last performed in a subroutine is automatically also the return value. You can return arrays and hashes from the subroutine like any scalar but returning more than one array or hash normally causes them to lose their separate identities. So we will use references (explained in the next chapter) to return any array or hash from a function. Let's try the following example, which takes a list of numbers and then returns their average

Live Demo

```
#!/usr/bin/perl
# Function definition
sub Average {
   # get total number of arguments passed.
   $n = scalar(@_);
```

```
  $sum = 0;
  foreach $item (@_) {
    $sum += $item;
  }
  $average = $sum / $n;
  return $average;
}
```

Check your Progress

1. What are Subroutines?
2. Define Passing Arguments to a Subroutine?
3. Define Passing Hashes to Subroutines?
4. Define Returning Value from a Subroutine?
5. What is Control Structures?

13.4 ANSWERS TO CHECK YOUR PROGRESS

1. A Perl subroutine or function is a group of statements that together performs a task. You can divide up your code into separate subroutines. How you divide up your code among different subroutines is up to you, but logically the division usually is so each function performs a specific task.
2. You can pass various arguments to a subroutine like you do in any other programming language and they can be accessed inside the function using the special array @_. Thus the first argument to the function is in $_[0], the second is in $_[1].
3. When you supply a hash to a subroutine or operator that accepts a list, then hash is automatically translated into a list of key/value pairs.
4. You can return a value from subroutine like you do in any other programming language. If you are not returning a value from a subroutine then whatever calculation is last performed in a subroutine is automatically also the return value. You can return arrays and hashes from the subroutine like any scalar but returning more than one array or hash normally causes them to lose their separate identities.

5. Perl conditional statements helps in the decision making, which require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

13.5 SUMMARY

- Perl conditional statements helps in the decision making, which require that the programmer specifies one or more conditions to be evaluated
- A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages programming language provides the following types of loop to handle the looping requirements.
- Loop becomes infinite loop if a condition never becomes false. The for loop is traditionally used for this purpose.

13.6 KEYWORDS

if statement: An if statement consists of a boolean expression followed by one or more statements.

While loop: Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

Until loop: Repeats a statement or group of statements until a given condition becomes true. It tests the condition before executing the loop body.

13.7 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer questions:

1. Explain about The ? : Operator?
2. What is Loop Control Statements?
3. What is The Infinite Loop?
4. Explain about Passing Lists to Subroutines?
5. Explain about Define and Call a subroutine?

Long Answer questions:

1. Explain briefly about subroutines?
2. Explain about Decision making statements in Perl?

13.8 FURTHER READINGS

Rémy Card, Eric Dumas, and Franck Mével. The Linux kernel book. John Wiley & Sons, Inc., 2003.

Steve Suchring. MySQL BBible. John Wiley, 2002.

Rasmus Lerdorf and Levin Tatroe. Programming PHP. " O'Reilly Media, Inc., 2002.

Wesley J. Chun. Core Python Programming. Prentice Hall, 2001.

Martin C. Brown. Perl: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Steven Holzner. PHP: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Vikram Vaswani. MySQL: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

UNIT 14
PACKAGES AND MODULES

Structure

14.0. INTRODUCTION

This unit describes the Perl programming language packages and modules regarding the working of files and data manipulation techniques. The working with files and how to establish the modules with data manipulation in Perl is explained.

14.1 OBJECTIVE

The users understands and learns the Perl Programming Concepts such as

- Working with Files
- Data Manipulation

14.2 PACKAGES

The package statement switches the current naming context to a specified namespace (symbol table). Thus A package is a collection of code which lives in its own namespace.A namespace is a named collection of unique variable names (also called a symbol table).Namespaces prevent variable name collisions between packages.

Packages enable the construction of modules which, when used, won't clobber variables and functions outside of the modules's own namespace. The package stays in effect until either another package statement is invoked, or until the end of the current block or file. You can explicitly refer to variables within a package using the :: package qualifier. Following is an example having main and Foo packages in a file. Here special variable PACKAGE has been used to print the package name.

Live Demo

```
#!/usr/bin/perl
# This is main package
$i = 1;
print "Package name : " , __PACKAGE__ , " $i\n";
package Foo;
# This is Foo package
$i = 10;
print "Package name : " , __PACKAGE__ , " $i\n";
package main;
# This is again main package
$i = 100;
print "Package name : " , __PACKAGE__ , " $i\n";
print "Package name : " , __PACKAGE__ ,  " $Foo::i\n";
```

When above code is executed, it produces the following result −

```
Package name : main 1
Package name : Foo 10
Package name : main 100
Package name : main 10
```

BEGIN and END Blocks

You may define any number of code blocks named BEGIN and END, which act as constructors and destructors respectively.

```
BEGIN { ... }
END { ... }
BEGIN { ... }
END { ... }
```

Every BEGIN block is executed after the perl script is loaded and compiled but before any other statement is executed.Every END block is executed just before the perl interpreter exits.The BEGIN and END blocks are particularly useful when creating Perl modules.

Following example shows its usage

Live Demo

```
#!/usr/bin/perl
package Foo;
print "Begin and Block Demo\n";
BEGIN {
  print "This is BEGIN Block\n"
}
END {
  print "This is END Block\n"
}
1;
```

When above code is executed, it produces the following result

```
This is BEGIN Block
Begin and Block Demo
This is END Block
```

14.3 MODULES

A Perl module is a reusable package defined in a library file whose name is the same as the name of the package with a .pm as extension.A Perl module file called Foo.pm might contain statements like this.

```
#!/usr/bin/perl
package Foo;
sub bar {
  print "Hello $_[0]\n"
}
sub blat {
  print "World $_[0]\n"
}
1;
```

The Require Function

A module can be loaded by calling the require function as follows

```
#!/usr/bin/perl
require Foo;
Foo::bar( "a" );
Foo::blat( "b" );
```

You must have noticed that the subroutine names must be fully qualified to call them. It would be nice to enable the subroutine bar and blat to be

NOTES

imported into our own namespace so we wouldn't have to use the Foo:: qualifier.

The Use Function

A module can be loaded by calling the use function.

```
#!/usr/bin/perl

use Foo;

bar( "a" );

blat( "b" );
```

Notice that we didn't have to fully qualify the package's function names. The use function will export a list of symbols from a module given a few added statements inside a module.require Exporter;

```
@ISA = qw(Exporter);
```

Then, provide a list of symbols (scalars, lists, hashes, subroutines, etc) by filling the list variable named @EXPORT: For Example −

```
package Module;

require Exporter;

@ISA = qw(Exporter);

@EXPORT = qw(bar blat);

sub bar { print "Hello $_[0]\n" }

sub blat { print "World $_[0]\n" }

sub splat { print "Not $_[0]\n" }  # Not exported!

1;
```

14.3.1 Create the Perl Module Tree

When you are ready to ship your Perl module, then there is standard way of creating a Perl Module Tree. This is done using h2xs utility. This utility comes along with Perl. Here is the syntax to use h2xs $h2xs -AX -n ModuleName. For example, if your module is available in Person.pm file, then simply issue the following command

```
$h2xs -AX -n Person
```

This will produce the following result

```
Writing Person/lib/Person.pm

Writing Person/Makefile.PL
```

Writing Person/README

Writing Person/t/Person.t

Writing Person/Changes

Writing Person/MANIFEST

Here is the descritpion of these options −

-A omits the Autoloader code (best used by modules that define a large number of infrequently used subroutines).

-X omits XS elements (eXternal Subroutine, where eXternal means external to Perl, i.e., C).

-n specifies the name of the module.

So above command creates the following structure inside Person directory. Actual result is shown above.

14.3.2 Installing Perl Module

Download a Perl module in the form tar.gz file. Use the following sequence to install any Perl Module Person.pm which has been downloaded in as Person.tar.gz file.

```
tar xvfz Person.tar.gz

cd Person

perl Makefile.PL

make

make install
```

The Perl interpreter has a list of directories in which it searches for modules (global array @INC)

14.4 WORKING WITH FILES

The basics of handling files are simple: you associate a file handle with an external entity (usually a file) and then use a variety of operators and functions within Perl to read and update the data stored within the data stream associated with the file handle. A file handle is a named internal Perl structure that associates a physical file with a name. All file handles are capable of read/write access, so you can read from and update any file or device associated with a file handle. However, when you associate a file handle, you can specify the mode in which the file handle is opened.

Three basic file handles are - STDIN, STDOUT, and STDERR, which represent standard input, standard output and standard error devices respectively.

14.4.1Opening and Closing Files

There are following two functions with multiple forms, which can be used to open any new or existing file in Perl.

```
open FILEHANDLE, EXPR

open FILEHANDLE

sysopen FILEHANDLE, FILENAME, MODE, PERMS

sysopen FILEHANDLE, FILENAME, MODE
```

Here FILEHANDLE is the file handle returned by the open function and EXPR is the expression having file name and mode of opening the file.

Open Function

Following is the syntax to open file.txt in read-only mode. Here less than < sign indicates that file has to be opend in read-only mode.

```
open(DATA, "<file.txt");
```

Here DATA is the file handle, which will be used to read the file. Here is the example, which will open a file and will print its content over the screen.

```
#!/usr/bin/perl

open(DATA, "<file.txt") or die "Couldn't open file file.txt, $!";

while(<DATA>) {

   print "$_";

}
```

Following is the syntax to open file.txt in writing mode. Here less than > sign indicates that file has to be opend in the writing mode.

```
open(DATA, ">file.txt") or die "Couldn't open file file.txt, $!";
```

This example actually truncates (empties) the file before opening it for writing, which may not be the desired effect. If you want to open a file for reading and writing, you can put a plus sign before the > or < characters.

For example, to open a file for updating without truncating it −

```
open(DATA, "+<file.txt"); or die "Couldn't open file file.txt, $!";

open(DATA,">>file.txt") || die "Couldn't open file file.txt, $!";
```

A double >> opens the file for appending, placing the file pointer at the end, so that you can immediately start appending information. However, you can't read from it unless you also place a plus sign in front of it −

```
open(DATA,"+>>file.txt") || die "Couldn't open file file.txt, $!";
```

Following is the table, which gives the possible values of different modes

< or r-Read Only Access

> or w-Creates, Writes, and Truncates

>> or a-Writes, Appends, and Creates

+< or r+-Reads and Writes

+> or w+-Reads, Writes, Creates, and Truncates

+>> or a+-Reads, Writes, Appends, and Creates

Sysopen Function

The sysopen function is similar to the main open function, except that it uses the system open() function, using the parameters supplied to it as the parameters for the system function For example, to open a file for updating, emulating the +<filename format from open

sysopen(DATA, "file.txt", O_RDWR);

Or to truncate the file before updating

sysopen(DATA, "file.txt", O_RDWR|O_TRUNC);

You can use O_CREAT to create a new file and O_WRONLY- to open file in write only mode and O_RDONLY - to open file in read only mode.The PERMS argument specifies the file permissions for the file specified, if it has to be created. By default it takes 0x666.

Following is the table, which gives the possible values of MODE.

O_RDWR-Read and Write

O_RDONLY-Read Only

O_WRONLY-Write Only

O_CREAT-Create the file

O_APPEND-Append the file

O_TRUNC-Truncate the file

O_EXCL-Stops if file already exists

O_NONBLOCK-Non-Blocking usability

14.4.2 Close Function

To close a filehandle, and therefore disassociate the filehandle from the corresponding file, you use the close function. This flushes the filehandle's buffers and closes the system's file descriptor.

close FILEHANDLE

close

If no FILEHANDLE is specified, then it closes the currently selected filehandle. It returns true only if it could successfully flush the buffers and close the file.

close(DATA) || die "Couldn't close file properly";

Reading and Writing Files

Once you have an open filehandle, you need to be able to read and write information. There are a number of different ways of reading and writing data into the file.

The <FILEHANDL> Operator

The main method of reading the information from an open filehandle is the <FILEHANDLE> operator. In a scalar context, it returns a single line from the filehandle. For example −

```
#!/usr/bin/perl

print "What is your name?\n";

$name = <STDIN>;

print "Hello $name\n";
```

When you use the <FILEHANDLE> operator in a list context, it returns a list of lines from the specified filehandle. For example, to import all the lines from a file into an array −

```
#!/usr/bin/perl

open(DATA,"<import.txt") or die "Can't open data";

@lines = <DATA>;

close(DATA);
```

getc Function

The getc function returns a single character from the specified FILEHANDLE, or STDIN if none is specified

getc FILEHANDLE

getc

Read Function

The read function reads a block of information from the buffered filehandle: This function is used to read binary data from the file.

read FILEHANDLE, SCALAR, LENGTH, OFFSET

read FILEHANDLE, SCALAR, LENGTH

The length of the data read is defined by LENGTH, and the data is placed at the start of SCALAR if no OFFSET is specified. Otherwise data is placed after OFFSET bytes in SCALAR. The function returns the number of bytes read on success, zero at end of file, or undef if there was an error.

14.5 DATA MANIPULATION

NOTES

e Command

The most useful way to use the command-line options is by writing Perl one-liners right in the shell. The -e option is the basis for most command-line programs. It accepts the value of the parameter as the source text for a program:

```
$ perl -e'print "Hello, World!\n"'
```

```
Hello, World!
```

Since this is a single statement in a block, you can omit the semicolon. Also, when the -e option is used, Perl no longer looks for a program name on the command line. This means you can't mix code with -e and a program file.

The -e option is repeatable, which lets you create entire scripts on the command line:

```
$ perl -e'print "Hello, ";' -e'print "World!\n"'
```

```
Hello, World!
```

When chaining together multiple -e options, make sure you keep your semicolons in the right place. I reflexively put semicolons in my -e lines just for safety's sake, even if it's not strictly necessary because there's only one -e option.With the -e option, any shell window becomes a Perl IDE. Use it as your calculator to figure out how many 80-line records are in a megabyte:

```
$ perl -e'print 1024*1024/80, "\n"'
```

```
13107.2
```

Escaping Shell Characters

When you're creating command-line programs, it's important to pay attention to quoting issues. In all my examples, I've quoted with single quotes—not double quotes—for two reasons. First, I want to be able to use double quotes inside my programs for literals, and double quotes don't nest in the shell. Second, I have to prevent shell interpolation, and single quotes make it easy. For example, if I use double quotes, then

```
$ perl -MCGI -e"print $CGI::VERSION"
```

gets the $CGI interpolated as a shell variable. Consequently, unless you have a shell variable called $CGI, Perl sees

```
print ::VERSION
```

You can escape the shell variables with a backslash:

```
$ perl -MCGI -e"print \$CGI::VERSION"
```

but that gets to be tough to maintain. That's why I stick with single quotes:

```
$ perl -MCGI -e'print $CGI::VERSION'
```

Windows has slightly different quoting issues. Windows doesn't have shell variable interpolation, so there's no need for escaping variables with dollar signs in them. On the other hand, you can use only double quotes under Windows, which can be a challenge if you want to use double quotes in your program. Under Windows, your "Hello, World" would look like this:

```
C:\> perl -e"print \"Hello, World!\n\""
```

The inner double quotes are escaped with backslashes.

The Diamond Operator

Perl's diamond operator, <>, has a great deal of magic built into it, making operations on multiple files easy.Have you ever written something like this:

```
for my $file ( @ARGV ) {
    open( my $fh, $file ) or die "Can't open $file: $!\n";
    while ( my $line = <$fh> ) {
       # do something with $line
    }
    close $fh;
 }


 $ perl myprog.pl file1.txt file2.txt file3.txt
```

So that your program can operate on three files at once? Use the diamond operator instead. Perl keeps track of which file you're on, and opens and closes the filehandle as appropriate. With the diamond operator, it's as simple as:

```
while ( my $line = <> ) {
    # do something
}
```

Perl keeps the name of the currently open file in $ARGV. The $. line counter does not reset at the beginning of each file.The diamond operator figures prominently in much Perl command-line magic, so it behooves you to get comfortable with it.

-n and -p: Automatic Looping Powerhouses

The -n and -p options are the real workhorse options. They derive from the Awk metaphor of "Do something to every line in the file," and work closely with the diamond operator.The following program prepends each line with its line number:

```
while (<>) {

   $_ = sprintf( "%05d: %s", $., $_ );

   print;  # implicitly print $_

}
```

The construct of "Walk through a file, and print $_ after you do some magic to it" is so common that Perl gives us the -p option to implement it for us. The previous example can be written as:

```
#!/usr/bin/perl -p

$_ = sprintf( "%05d: %s", $., $_ );
```

or even shorter as:

```
$ perl -p -e'$_ = sprintf( "%05d: %s", $. $_ )'
```

The -n option is just like -p, except that there's no print at the bottom of the implicit loop. This is useful for grep-like programs when you're only interested in selected information. You might use it to print only commented-out lines from your input, defined as beginning with optional whitespace and a pound sign:

```
$ perl -n -e'print if /^\s*#/'
```

The next program prints every numeric value that looks like it's part of a dollar value, as in "$43.50."

```
#!/usr/bin/perl -n

while ( /\$(\d+\.\d\d)/g ) {

   print $1, "\n";

}
```

Check your Progress

1. What is symbol table?
2. What is the Require function in module?
3. What is the Use function in module?
4. What is Perl Module Tree?
5. What is Read Function?

14.6 ANSWERS TO CHECK YOUR PROGRESS

1. The package statement switches the current naming context to a specified namespace (symbol table). Thus, A package is a collection of code which lives in its own namespace. A namespace is a named collection of unique variable names (also called a symbol table).
2. A module can be loaded by calling the require function as follows.
    i. #!/usr/bin/perl

    ii. require Foo;

    iii. Foo::bar("a");

    iv. Foo::blat("b");

3. A module can be loaded by calling the use function.
    i. #!/usr/bin/perl

    ii. use Foo;

    iii. bar("a");

    iv. blat("b");

4. When you are ready to ship your Perl module, then there is standard way of creating a Perl Module Tree. This is done using h2xs utility. This utility comes along with Perl. Here is the syntax to use h2xs $h2xs -AX -n ModuleName.

5. The read function reads a block of information from the buffered filehandle: This function is used to read binary data from the file. read FILEHANDLE, SCALAR, LENGTH, OFFSET

    read FILEHANDLE, SCALAR, LENGTH

14.7 SUMMARY

- Packages enable the construction of modules which, when used, won't clobber variables and functions outside of the modules's own namespace.
- A Perl module is a reusable package defined in a library file whose name is the same as the name of the package with a .pm as extension.
- A file handle is a named internal Perl structure that associates a physical file with a name.
- Perl's diamond operator, <>, has a great deal of magic built into it, making operations on multiple files easy.
- Three basic file handles are - STDIN, STDOUT, and STDERR, which represent standard input, standard output and standard error devices respectively.

14.8 KEYWORDS

Sysopen Function: The sysopen function is similar to the main open function, except that it uses the system open() function, using the parameters supplied to it as the parameters for the system function

Close Function: To close a filehandle, and therefore disassociate the filehandle from the corresponding file, you use the close function.

e Command: The most useful way to use the command-line options is by writing Perl one-liners right in the shell.

14.9 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short Answer questions:

6. What is Perl Diamond Operator?
7. What is Escaping Shell Characters?
8. Explain about working with files?
9. Explain about Modules?
10. What are Packages?

Long Answer questions:

4. Explain briefly about the Perl Module Tree?
5. Explain briefly about Working with files?
6. Explain about Packages?

14.10. FURTHER READINGS

Rémy Card, Eric Dumas, and Franck Mével. The Linux kernel book. John Wiley & Sons, Inc., 2003.

Steve Suchring. MySQL BBible. John Wiley, 2002.

Rasmus Lerdorf and Levin Tatroe. Programming PHP. " O'Reilly Media, Inc., 2002.

Wesley J. Chun. Core Python Programming. Prentice Hall, 2001.

Martin C. Brown. Perl: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Steven Holzner. PHP: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

Vikram Vaswani. MySQL: The complete reference. 2nd Edition, Tata McGraw-Hill Publishing Company Limited, Indian Reprint, 2009.

MODEL QUESTION PAPER

Time: Three hours **Maximum: 75 marks**

SECTION-A **(10X2=20 marks)**

Answer ALL questions.

All questions carry equal marks.

1. Differentiate the terms “free open-source software” and “commercial opensource software”.
2. Specify in what ways open source software is reliable and stable.
3. List out the data types of PHP.
4. Distinguish between implementing a web server in kernel mode and user mode.
5. Write a short note on squirrel mail web mail services.
6. List three levels of network firewall security.
7. What is the use of GNU linker?
8. What is SSI?
9. List some X- Clients and its functionalities.
10. What is localization?

SECTION-B **(5X5=25 marks)**

Answer ALL questions.

All questions carry equal marks.

11. a) Explain the overview of Free/Open Source Software.

(OR)

b) Explain the advantages of Free Software and GNU/Linux, FOSS usage.

12. a) What are shells? Describe the different types of shells available in Linux.

(OR)

b) Explain: Setting up email servers. (i) Using postfix (SMTP services) (ii) Courire (IMAP& POP3).

13. a) What are the main objectives of file sharing? Explain the methods used in Linux for performing file sharing services.

(OR)

b) Explain the usage of source code versioning and management tools using CVS to manage source code revisions, patch and duff.

14. a) Explain the Model Driven Architecture.

(OR)

b) Explain the Features of Meta Object facility.

15. a) Discuss the basics of the X Windows server architecture.

(OR)

b) What is GTK+ programming? Explain the applications of GTK+.

SECTION-C (3X10=30 marks)

Answer any THREE questions.

All questions carry equal marks.

16. Explain the compiler tools in GNU.

17. Discuss the GNU Libc libraries and linker.

18. Explain the basics of X Windows server architecture.

19. Discuss about python programming.

20. How to setting up the firewall using netfilter?