# OPEN SOURCE SOFTWARE
# (PROGRAM ELECTIVE-VI)

| | | | |
|---|---|---|---|
| Subject Code | 18ITITP803G | IA Marks | 30 |
| Number of Lecture Hours/Week | 3 | Exam Marks | 70 |
| Total Number of Lecture Hours | 50 | Exam Hours | 03 |

**Credits – 03**

| **Unit -1** | **Hours** |
|---|---|
| **Introduction to Open source:** Need of Open sources, Advantages and Applications of open sources, Open Source Operating Systems: Linux Introduction, General overview - Kernel and user Mode, Linux: Process, Advanced Concepts, scheduling, Personalities, Cloning, Signals, Development with Linux. | **10** |
| **Unit -2** | |
| **Open Source Database:** Introduction to MYSQL, Setting up account, starting, writing own sql Programs, Record selection, working with strings, date and Time, Sorting query Results, generating summary, Working with metadata, Using Sequences, Mysql and web. | **08** |
| **Unit – 3** | |
| **Introduction to PHP:** Programming In web environment, Variable, constants, data types, Operators, statements, Functions and Arrays, OOP, string manipulation, Regular expression, File handling & Data Storage, PHP and SQL database, PHP and LDAP, PHP Connectivity - sending and receiving mails, Debugging and Error handling, Security and Templates. | **10** |
| **Unit – 4** | |
| **PYTHON:** Syntax and Style, Python objects, Numbers, Sequences, Strings, Lists, Tuples, Dictionaries, Conditionals, Loops, Files –Input and Output, Errors and Exceptions, Functions, Modules, Classes and OOP, Execution Environment. | **10** |
| **Unit – 5** | |
| **PERL:** Overview, Variables - scalars, arrays and hashes, Operators, Control Structures - Conditional and looping statements, Subroutines, Packages and Modules, Working with files, Working with Database, Data manipulation. **RUBY:** Overview, Variables - arrays and hashes, Control Structures - Conditional and looping statements, Methods, Blocks, Modules, Iterators, Working with files, Working with Database. | **12** |

# UNIT I

# INTRODUCTION

Introduction to Open sources – Need of Open Sources – Advantages of Open Sources–Application of Open Sources. Open source operating systems:

**LINUX:**

Introduction – GeneralOverview – Kernel Mode and user mode.

## 1.Introduction to Open Sources:

Most software that you buy or download only comes in the compiled ready-to-run version. Compiled means that the actual program code that the developer created, known as the source code, has run through a special program called a compiler that translates the source code into a form that the computer can understand.

Open Source is a certification mark owned by the Open Source Initiative (OSI). It refers to any program whose source code is made available for use or modification as users or other developers. Open source software (OSS) refers to software that is developed, tested, or improved through public collaboration and distributed with the idea that the must be shared with others, ensuring an open future collaboration. **(OSS** is computer software with its source code made available and licensed with a license in which the copyright holder provides the rights to study, change and distribute the software to anyone and for any purpose. Open-source software is very often developed in a public, collaborative manner.**)**

## Definition of Open Source:

Open source refers to any program whose source code is made available for use or modification as users or other developers see fit. Open source software is usually developed as a public collaboration and made freely available.

Open source software refers to applications developed in which the user can access and alter the "source" code itself.

Open-source software is computer software whose source code is available under a license (or arrangement such as the public domain)It is defined set of requirements for open source software from the Open Source Initiative (OSI). The Open Source Definition (OSD) specifies (that permits users, to study, change, and improve the software) not only access to the source code, but also integrity of the code, its free redistribution, a technology-neutral provision, as well as specific anti-discrimination rules.

and to redistribute it in modified or modified form. Based on two principles we can call particular softwareas open source software.

## Principle 1:

The software source code should be available with license and that license contain permissions they are

1) The user is able to study the code
2) The user able to change the code
3) The may able to improve the code

## Principle 2:

The license should not have certain restrictions in terms of

4) Technology
5) Field  3) Hardware

**Technology:** Here Technology means operating system, in the computer science there are many different operating systems available, here the software must support all kinds of operating sytems such as windows, UNIX, Linux and Mac os.

**Field:** Now a days computer enter into many fields such as agriculture, medical and Biotechnology fields. here the software must supports or works in all fields

**Hardware:** In this context hardware means devices such as Nokia, Samsung and celkon. Here the software should work on or supports all kinds of devices.

## Need of Open Sources:

Few reasons why you need an Open Source Strategy are:

**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.

**Source Code**

The program must include source code, and must allow distribution in source code as well as compiledform. 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.

**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.

**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 mayrequire derived works to carry a different name or version number from the original software.

**No Discrimination against Persons or Groups**

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

**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.

**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.

**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.

**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.

**License Must Be Technology-Neutral**

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

## Advantages of Open Sources:

Open source software can have a major impact on your entire organization. There are several advantages of using open source software. The following are a list of the advantages of opting for open source software.

**Lesser hardware costs**

Since Linux and open source solutions are easily portable and compressed, it takes lesser hardware power to carry out the same tasks when compared to the hardware power it takes on servers, such as, Solaris, Windows or workstations. With this less hardware power advantage, you can even use cheaper or older hardware and still get the desired results.

**High-quality software**

Open source software is mostly high-quality software. When you use the open source software, the source code is available. Most open source software are well-designed. Open source software can also be efficiently used in coding. These reasons make open source software an ideal choice for organizations.

**No vendor lock-in**

IT managers in organizations face constant frustration when dealing with vendor lock-ins'. Lack of portability, expensive license fees and inability to customize software are some of the other disadvantages. Using open source software gives you more freedom and you can effectively address all these disadvantages.

**Integrated management**

By using open source software, you can benefit from integrated management. Open source software uses technologies, such as, common information model (CIM) and web based enterprise management(WBEM). These high-end technologies enable you to integrate and combine server, application, service and workstation management. This integration would result in efficient administration.

**Simple license management**

When you use open source software, you would no longer need to worry about licenses. Open source software enables you to install it several times and also use it from any location. You will be free from monitoring, tracking or counting license compliance.

**Lower software costs**

Using open source software can help you minimize your expenses. You can save on licensing fees and maintenance fees. The only expenses that you would encounter would be expenditure for documentation, media and support.

**Abundant support**

You will get ample support when you use open source software. Open source support is mostly freely available and can be easily accessed through online communities. There are also many software companies that provide free online help and also varied levels of paid support. Most organization who create open source software solutions also provide maintenance and support.

**Scaling and consolidating**

Linux and open source software can be easily scaled. With varied options for clustering, load balancing and open source applications, such as email and database, you can enable your organization to either scale up and achieve higher growth or consolidate and achieve more with less.

**Evolving software**

As mentioned, some Open Source software projects can have huge communities of programmers involved, allowing for the rapid implementation of new features and security fixes. The communities of users and programmers are also invaluable resources for asking questions relating to troubleshooting andsuggesting enhancements.

**Rapid debugging, rapid further development**

Because the source code is open, the developer/producer does not just receive feedback on any errorsor problems, or proposals for new functions, but feedback reports that can specify down to the code level what should be done – it is therefore far simpler for the producer to implement changes on the basis of feedback reports since these often say precisely what program changes must be made and also any errors in the original source code may be corrected by the person who detects the error without having to wait for the original programmer.

**Avoiding lock-in to one supplier**

It is obviously great to have one software supplier to turn to – perhaps to provide services connected with the software, such as installation assistance, courses, operation, support and more, and you have someone to ring if you need help or information concerning the software.

**Easy integration and interaction**

Open Source code means that it is relatively simple to adapt programs so that they can work with each other because you can see from the source codes how a program ”thinks” and how you should approach it to share or exchange data.

- **Disadvantages of using Open Source**

There's a flip side to everything, and in the case of Open Source software it all boils down to the old saying of "there's no such thing as a free lunch". Most of the disadvantages only apply if you're not somewhat code-savvy and willing to get your hands dirty:

1. Mostly used commercial applications.
2. Projects can die
3. Support issues

## <u>Application of Open Sources:</u>

Open source technology makes a real business sense. It is free and of very high quality, also it is often moreeffective than most of the products available commercially and non-commercial.

Some applications listed below:

1. Accounting
2. Content Management Systems
3. CRM (Customer Relationship Management)
4. Desktop Environments/ Shell replacements
5. Email Clients
6. Encoding, Conversion & Ripping Tools
7. ERP
8. Filesharing & FTP
9. Graphics-Design & Modeling Tools
10. Messengers & Communication Clients
11. Project Management
12. Reporting Tools
13. RSS
14. Web Browser

| S.No | Application | Open Source Tools |
|---|---|---|
| 1 | Cloud management | Abiquo |
| 2 | Ecommerce | Avactis |
| 3 | Reporting Tools | Actuate |
| 4 | Enterprise Content Management, Web Content Management | Alfresco |
| 5 | Data Backup / Recovery | Bacula |
| 6 | ERP and CRM | Compiere |
| 7 | Office Productivity | Lotus Symphony |
| 8 | RDBMS | Ingres Database |
| 9 | Software Development Tools for C, C++ | Sun Studio |
| 10 | Server and client Linux distribution | Ubuntu |

**Table - List of Commercial Open Source Applications with tools**

## Open source operating systems

## LINUX:

**Introduction: L**inux is an Operating System that was first created at the University of Helsinki in Finland by a young student named Linus Torvalds. At this time the student was working on a UNIX system that was running on an expensive platform. Because of his low budget, and his need to work at home, he decided to create a copy of the UNIX system in order to run it on a less expensive platform, such as an IBM PC.

He began his work in 1991 when he released version 0.02 and worked steadily until 1994 when version 1.0 of the Linux Kernel was released. The current full-featured version at this time is 2.2.X; released January 25, 1999, and development continues.

**Reasons to use Linux**

- **Configurability**

  Linux distributions give the user full access to configure just about any aspect of their system.

  Options range from the simple and straightforward.

- **Convenience**

  While Linux takes some effort to get set up, once it is set up, it is surprisingly low-maintenance.

- **Stability**

  Linux is based on the UNIX kernel. It providespreemptive multitasking and protected memory.

- **Community**

  Linux is part of the greater open-source community. This consists of thousands of developers and many more users world-wide whosupport open software.This user and developer base is also a support base.

- **Freedom**

  Linux is free. This means more than just costingnothing. This means that you are allowed to do whatever you want to with the software. This is why Redhat, Mandrake, and Suse are all allowed to sell their own distributions of Linux

## General Overview:

### ➢ 1.5.2.1 The Linux Kernel

The Linux Kernel is an operating system, which runs on a wide variety of hardware and for a variety of purposes. Linux  is capable of running on devices as simple as a wrist watch, or a cell phone, but it can also run on a home computer using, for example Intel, or AMD processors, and its even capable of running on high end servers using Sun Sparc CPU's or IBM power PC processors. Some Linux distro's can only run one processor, while others can run many at once.

The Linux kernel is an operating system kernel used by the Linux family of Unix-likeoperating systems. It is one of the most prominent examples of free and open source software.

The Linux kernel was initially conceived and created by Finnishcomputer science student Linus Torvalds in 1991. Linux rapidly accumulated developers and users who adopted code from other free software projects for use with the new operating system. The Linux kernel has received contributions from thousands of programmers. Many Linux distributions have been released based upon the Linux kernel.

**Properties of Linux:**

- **Linux Pros**

A lot of the advantages of Linux are a consequence of Linux' origins, deeply rooted in UNIX, except for thefirst advantage, of course:

- Linux is free:
- Linux is portable to any hardware platform:
- Linux was made to keep on running:
- Linux is secure and versatile:
- Linux is scalable:
- The Linux OS and most Linux applications have very short debug-times:

**Linux Cons**

- There are far too many different distributions:
- Linux is not very user friendly and confusing for beginners:
- Is an Open Source product trustworthy?

**Linux vs. Windows**

| Topic | Linux | Windows |
|---|---|---|
| **Price** | The majority of Linux variants are available for free or at a much lower price than Windows. | Microsoft Windows can run between $20.00 - $150.00 US dollars per each license copy. |
| **Reliability** | The majority of Linux variants and versions are notoriously reliable and can often run for months and years without needing to be rebooted. | Although Microsoft Windows has made great improvements in reliability over the last few versions of Windows, it still cannot match the reliability of Linux. |
| **Software** | Linux has a large variety of available software programs, utilities, and games. However, Windows has a much larger selection of available software. | Because of the large amount of Microsoft Windows users, there is a much larger selection of available software programs, utilities, and games for Windows. |
| **Software Cost** | Many of the available software programs, utilities, and games available on Linux are freeware and/or open source. Even such complex programs such as Gimp, OpenOffice, StarOffice, and wine are available for free or at a low cost. | Although Windows does have software programs, utilities, and games for free, the majority of the programs will cost anywhere between $20.00 - $200.00+ US dollars per copy. |
| **Hardware** | Linux companies and hardware manufacturers have made great advancements in hardware support for Linux and today Linux will support most hardware devices. However, many companies still do not offer drivers or support for their hardware in Linux. | Because of the amount of Microsoft Windows users and the broader driver support, Windows has a much larger support for hardware devices and a good majority of hardware manufacturers will support their products in Microsoft Windows. |

|  |  |  |
|---|---|---|
| **Security** | Linux is and has always been a very secure operating system. Although it still can be attacked when compared to Windows, it muchmore secure. | Although Microsoft has made great improvements over the years with security on their operating system, their operating system continues to be the most vulnerable to viruses and other attacks. |
| **Open Source** | Many of the Linux variants and many Linux programs are open source and enable users to customize or modify the code however they wish to. | Microsoft Windows is not open source and the majority of Windows programs are not open source. |
| **Support** | Although it may be more difficult to find users familiar with all Linux variants, there are vast amounts of available online documentation and help available for Linux. | Microsoft Windows includes its own help section, has vast amount of available online documentation and help, as well as books on each of the versions of Windows. |

## Kernel Mode and user:

A kernel is the main program or component of an operating system, it's the thing that does all the work, and without it you have no operating system. Linux is actually nothing more than a kernel; the programs that make up the rest of the operating system are generally GNU software, so the entire operating system is usually referred to as GNU/Linux. User mode is the normal mode of operating for programs. Web browsers, calculators, etc. will all be in user mode. They don't interact directly with the kernel, instead, they just give instructions on what needs to be done, and the kernel takes care of the rest. Kernel mode, on the other hand, is where programs communicate directly with the kernel. A good example of this would be device drivers. A device driver must tell the kernel exactly how to interact with a piece of hardware, so it must be run in kernel mode. Because of this close interaction with the kernel, the kernel is also a lot more vulnerable to programs running in this mode,so it becomes highly crucial that drivers are properly debugged before being released to the public.

A process can run in two modes:1.User Mode. 2.Kernel Mode.

**User Mode**:

A mode of the CPU when running a program.In this mode,the user process has no access to thememory locations used by the kernel.When a program isrunning in User Mode, it cannot directly access the kerneldata structures or the kernel programs.

The kernel-mode programs run in the background, making sureeverything runs smoothly - things like printer drivers,display drivers, drivers that interface with the monitor,keyboard, mouse, etc. These programs all run in such a waythat you don't notice them.

When the computer boots up, Windows calls the KERNEL, themain kernel-mode program that allows all the other programsto run, even the user-mode programs.

User mode is the normal mode of operating for programs, webbrowsers etc. They don't interact directly with the kernel,instead, they just give instructions on what needs to bedone, and the kernel takes care of the rest. Kernel mode, onthe other hand, is where programs communicate directly withthe kernel.

A good example of this would be device drivers.A device driver must tell the kernel exactly how tointeractwith a piece of hardware, so it must be run in kernel mode.Because of this close interaction with the kernel, thekernel is also a lot more vulnerable to programs running inthis mode, so it becomes highly crucial that drivers areproperly debugged before being released to the public.

These are the programs that you run when you want specificprograms - e.g., MS Paint, MS Word, and Calculator. Theseare heavily restricted, as to not crash the system. Windows uses memory-protection services offered by the processor toprevent malicious programs from interferingwith the rest ofthe system and corrupting it.

In User mode, the executing code has no ability to directlyaccess hardware or reference memory.

Code running in user mode must delegate to system APIs to access hardware or memory.

Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode.

**Kernel Mode:**

A mode of the CPU when running a program. In this mode, it is the kernel that is running on behalf of the user process and directly access the kernel data structures or the kernel programs.Once the system callreturns,the CPU switches back to user mode.

The kernel-mode programs run in the background, making sure every thing runs smoothly - things like printer drivers,display drivers, drivers that interface with the monitor, keyboard, mouse, etc. These programs all run in such a waythat you don't notice them.

When the computer boots up, Windows calls the KERNEL, themain kernel-mode program that allows all the other programsto run, even the user-mode programs.Kernel mode, also referred to as system mode, is one of thetwo distinct modes of operation of the CPU in Linux. Whenthe CPU is in kernel mode, it is assumed to be executingtrusted software, and thus it can execute any instructionsand reference any memory addresses (i.e., locations inmemory). The kernel (which is the core of the operatingsystem and has complete control over everything that occursin the system) is trusted software, but all other programsare considered untrusted software.

In this mode executing code has complete and unrestricted access to the underlying hardware. Itcan execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire PC.

When the CPU is in kernel mode, it is assumed to be executing trusted software, and thus it can execute any instructions and reference any memory addresses (i.e., locations in memory).

The kernel (which is the core of the operating system and has complete control over everything that occurs in the system) is trusted software, but all other programs are considered untrusted software. Thus, all user mode software must request use of the kernel by means of a system call inorder to perform privileged instructions, such as process creation or input/output operations.

**Distinguish Between User Mode and Kernel Mode**

- Kernal mode has higher priority while user mode has lowerpriority.
- Kernal mode has ability to read user mode but user mode hasnot priority to interface in kernal mode.
- When a process is in user space, its system calls are being intercepted by the tracing thread. When it's in the kernel, it's not under system call tracing. This is the distinction between user mode and kernel mode.
- The transition from user mode to kernel mode is done by the tracing thread. When a process executes a system call or receives a signal, the tracing thread forces the process to run in the kernel if necessary and continues it without system call tracing.

# UNIT II

**LINUX:** Process – Advanced Concepts – Scheduling – Personalities – Cloning – Signals – Developmentwith Linux. .

## LINUX:

## Process:

**Definition:** A process is an executing (i.e., running) instance of a program. Process are frequently referred toas tasks.

Process (process ID, PID): All software runs within an operating system concept known as a process. Each program running on a system is assigned its own process ID (PID). Users can easily obtain a process list (using Task Manager on Windows or ps on UNIX) in order to see what is running.

**Processes inside out**

**Multi-user and multi-tasking:**

Multiuser refers to having more than 1 person able to log into the computer and each person have their own settings (bookmark, desktop, themes, etc) Multitasking is the ability of the computer to do more than 1 thing (program) at a time. There was a time when you could not surf, type in word and listen to musicon your computer all at one time.

## Process types:

**Interactive processes**

Interactive processes are initialized and controlled through a terminal session. In other words, there has to be someone connected to the system to start these processes; they are not started automatically as partof the system functions

Running a process in the background is only useful for programs that don't need user input (via theshell).Putting a job in the background is typically done when execution of a job is expected to take a long time. In the example, using graphical mode, we open an extra terminal window from the existing one:

```
aravind:~> xterm &
[1] 26558
aravind:~> jobs
[1]+ Running xterm &
```

**Controlling processes**

| part of) command | Meaning |
|---|---|
| regular_command | Runs this command in the foreground. |
| command & | Run this command in the background (release the terminal) |
| Jobs | Show commands running in the background. |
| Ctrl+Z | Suspend (stop, but not quit) a process running in the foreground(suspend). |
| Ctrl+C | Interrupt (terminate and quit) a process running in the foreground. |
| Bg | Reactivate a suspended program in the background. |
| Fg | Puts the job back in the foreground. |
| Kill | End a process |

**Automatic processes**

Automatic or batch processes are not connected to a terminal. Rather, these are tasks that can be queued into a spooler area, where they wait to be executed on a FIFO (first-in, first-out) basis. Such taskscan be executed using one of two criteria:

- At a certain date and time: done using the **at** command.
- At times when the total system load is low enough to accept extra jobs: done using the **batch** command.

**Daemons**

A daemon process has no controlling terminal. It cannot open /dev/tty. Most daemons tend to last a long time, be owned by root, and do something useful. A daemon is just a process that runs in the background, usually waiting for something to happen that it is capable of working with, like  a printer daemon is waiting for print commands.

Daemons are server processes that run continuously. Most of the time, they are initialized at systemstartup and then wait in the background until their service is required.

**Process attributes**

A process has a series of characteristics, which can be viewed with the **ps** command:

- The process ID or PID: a unique identification number used to refer to the process.
- The parent process ID or PPID: the number of the process (PID) that started this process.
- Nice number: the degree of friendliness of this process toward other processes (not

to be confusedwith process priority, which is calculated based on this nice number and recent CPU usage of the process).

- Terminal or TTY: terminal to which the process is connected.
- User name of the real and effective user (RUID and EUID): the owner of the process.
- Real and effective group owner (RGID and EGID): The real group owner of a process is the primarygroup of the user who started the process.

**Displaying process information:**

The **ps** command is one of the tools for visualizing processes. This command has several optionswhich can be combined to display different process attributes.With no options specified, **ps** only gives information about the current shell and eventual processes:

```
aravind:~> ps

PID TTY TIME CMD

4245 pts/7 00:00:00 bash

5314/7 00:00:00 ps
```

**Life and death of a process:**

**Process creation**

A new process is created because an existing process makes an exact copy of itself. This child process has the same environment as its parent, only the process ID number is different. This procedure is called *forking*. After the forking process, the address space of the child process is overwritten with the newprocess data. This is done through an *exec* call to the system.

**Ending processes**

When a process ends normally (it is not killed or otherwise unexpectedly interrupted), the program returns its *exit status* to the parent. This exit status is a number returned by the program providing the resultsof the program's execution.

**Signals:**

Processes end because they receive a signal. There are multiple signals that you can send to a process. Use the **kill** command to send a signal to a process. The command **kill -l** shows a list of signals. Most signals are for internal use by the system, or for programmers when they write code. As a user, youwill need the following signals:

**Common signals**

| Signal name | Signal number | Meaning |
|---|---|---|
| SIGTERM | **15** | Terminate the process in an orderly way. |
| SIGINT | **2** | Interrupt the process. A process can ignore this signal. |
| SIGKILL | **9** | Interrupt the process. A process can not ignore this signal. |
| SIGHUP | **1** | For daemons: reread the configuration file. |

**Boot process, Init and shutdown:**

**PC Boot and Linux Init Process:**

- BIOS: The Basic Input/Output System is the lowest level interface between the computer and peripherals. The BIOS performs integrity checks on memory and seeks instructions on the Master Boor Record (MBR) on the floppy drive or hard drive.
- The MBR points to the boot loader (GRUB or LILO: Linux boot loader).
- Boot loader (GRUB or LILO) will then ask for the OS label which will identify which kernel to runand where it is located (hard drive and partition specified). The installation process requires to creation/identification of partitions and where to install the OS. GRUB/LILO are also configured during this process. The boot loader then loads the Linux operating system.
- The first thing the kernel does is to execute init program. Init is the root/parent of all processesexecuting on Linux.
- The first processes that init starts is a script /etc/rc.d/rc.sysinit
- Based on the appropriate run-level, scripts are executed to start various processes to run the systemand make it functional.

One of the most powerful aspects of Linux concerns its open method of starting and stopping the operating system, where it loads specified programs using their particular configurations, permits you to change those configurations to control the boot process, and shuts down in a graceful and organized way.

Beyond the question of controlling the boot or shutdown process, the open nature of Linux makes itmuch easier to determine the exact source of most problems associated with starting up or shutting down your system. A basic understanding of this process is quite beneficial to everybody who uses a Linux system.A lot of Linux systems use **lilo**, the LInux LOader for booting operating systems. We will only discuss GRUB, however, which is easier to use and more flexible.

**Init:**

The kernel, once it is loaded, finds **init** in `sbin` and executes it.When **init** starts, it becomes the parent or grandparent of all of the processes that start up automatically on your Linux system. The first thing**init** does, is reading its initialization file, `/etc/inittab`. This instructs **init** to read an initial configuration script for the environment, which sets the path, starts swapping, checks the file systems, and so on. Basically, this step takes care of everything that your system needs to have done at system initialization: setting the clock, initializing serial ports and so forth.

Then **init** continues to read the `/etc/inittab` file, which describes how the system should be set upin each run level and sets the default *run level*. A run level is a configuration of processes. All UNIX-like systems can be run in different process configurations, such as the single user mode, which is referred to as run level 1 or run level S (or s). In this mode, only the system administrator can connect to the system. It is used to perform maintenance tasks without risks of damaging the system or user data. Naturally, in this configuration we don't need to offer user services, so they will all be disabled. Another run level is the reboot run level, or run level 6, which shuts down all running services according to the appropriateprocedures and then restarts the system.

Use the **who** to check what your current run level is:

```
aravind@home:~> who -r
run-level 2 2010-10-17 23:22 last =S
```

**Init run levels**

Available run levels are generally described in /etc/inittab, which is partially shown below:
#

# inittab This file describes how the INIT process should set up# the system in a certain run-level.

# Default run level. The run levels are:

# 0 - halt (Do NOT set initdefault to this)# 1 - Single user mode

# 2 - Multiuser, without NFS# (The same as 3, if you do not have networking)# 3 - Full multiuser mode

# 4 - unused# 5 - X11

# 6 - reboot (Do NOT set initdefault to this)#

id:5:initdefault:

<--cut-->

# Advanced Concepts

Scheduling – Personalities – Cloning – Signals – Development with Linux

## 2.2 Scheduling:

A Linux system can have a lot to suffer from, but it usually suffers only during office hours. Whether in an office environment, a server room or at home, most Linux systems are just idling away during the morning, the evening, the nights and weekends. Using this idle time can be a lot cheaper than buying those machines you'd absolutely need if you want everything done at the same time.

There are three types of delayed execution:

- Waiting a little while and then resuming job execution, using the **sleep** command. Execution time depends onthe system time at the moment of submission.
- Running a command at a specified time, using the **at** command. Execution of the job(s) depends on systemtime, not the time of submission.
- Regularly running a command on a monthly, weekly, daily or hourly basis, using the **cron** facilities.

### The at command:

The **at** command executes commands at a given time, using your default shell unless you tell thecommand.The options to **at** are rather user-friendly, which is demonstrated in the examples below:

```
aravind@home:~> at tomorrow + 2 days
warning: commands will be executed using (in order) a) $SHELL
b) login shell c) /bin/sh
at> cat reports | mail myboss@mycompany
at> <EOT>
job 1 at 2010-10-16 12:36
```

Typing **Ctrl**+**D** quits the **at** utility and generates the "EOT" message.User *aravind* does a strange thing here combining two commands

```
aravind@home:~> at 0237
warning: commands will be executed using (in order) a) $SHELL
b) login shell c) /bin/sh at> cd new-programs at> ./configure; make at>
<EOT>
job 2 at 2010-10-14 02:00
```

The -m option sends mail to the user when the job is done, or explains when a job can't be done. Thecommand **atq** lists jobs; perform this command before submitting jobs in order prevent them from starting at the same time as others. With the **atrm** command you can remove scheduled jobs if you change your mind.

The **at** command is used to schedule one or more programs for a single execution at some later time.

There are actually four client commands:

1. at: Runs commands at specified time
2. atq: Lists pending commands
3. atrm: Cancels pending jobs
4. batch: Runs commands when system load permits

The Linux **at** command accepts a number of time specifications, considerably extending the POSIX.2standard.

**Cron and crontab:**

The cron system is managed by the **cron** daemon. It gets information about which programs and when they should run from the system's and users' crontab entries. Only the root user has access to the system crontabs, while each user should only have access to his own crontabs. On some systems (some) users may not have access to the cron facility.

At system startup the cron daemon searches /var/spool/cron/ for crontab entries which are named after accounts in /etc/passwd, it searches /etc/cron.d/ and it searches /etc/crontab, then uses this information every minute to check if there is something to be done. It executes commands as the user who owns the crontab file and mails any output of commands to the owner.

**Syntax**

crontab [-e] [-l] [-r] [filename]

| | |
|---|---|
| -e | edit a copy of the current user's crontab file, or creates an empty file to edit if crontabdoes not exist. |
| -l | list the crontab file for the invoking user. |
| -r | remove a user's crontab from the crontab |
| Filename | The filename that contains the commands to run. |

**Lines that can be in the crontab file.**

minute (0-59),

hour (0-23),

day of the month (1-31), month of the year (1-12),

day of the week (0-6 with 0=Sunday).

**Example: crontab –e fields,**

| min | hour | dayofmonth | monthofyear | dayofweek | Command |
|---|---|---|---|---|---|
| 0 | 12 | 14 | 2 | * | Echo “hai” |

| Options | Explanation |
|---|---|
| * | Is treated as a wild card. Meaning any possible value. |
| */5 | Is treated as ever 5 minutes, hours, days, or months. Replacing the 5 with another numerical valuewill change this option. |
| 2,4,6 | Treated as an OR, so if placed in the hours, this could mean at 2, 4, or 6 o-clock. |
| 9-17 | Treats for any value between 9 and 17. So if placed in day of month this would be days 9 through 17. Or if put in hours it would be between 9 and 5. |

## 3 System Calls Related to Scheduling:

We change the scheduling parameters by means of the system calls illustrated in the Table

**Table 10-1: System Calls Related to Scheduling**

| System Call | Description |
|---|---|
| nice( ) | Change the priority of a conventional process. |
| getpriority( ) | Get the maximum priority of a group of conventional processes. |
| setpriority( ) | Set the priority of a group of conventional processes. |
| sched_getscheduler( ) | Get the scheduling policy of a process. |
| sched_setscheduler( ) | Set the scheduling policy and priority of a process. |
| sched_getparam( ) | Get the scheduling priority of a process. |
| sched_setparam( ) | Set the priority of a process. |
| sched_yield( ) | Relinquish the processor voluntarily without blocking. |
| sched_get_ priority_min( ) | Get the minimum priority value for a policy. |
| sched_get_ priority_max( ) | Get the maximum priority value for a policy. |
| sched_rr_get_interval( ) | Get the time quantum value for the Round Robin policy. |

Most system calls shown in the table apply to real-time processes, thus allowing users to develop real-time applications.

## Personalities:

Linux supports different execution domains, or personalities, for each process. Among other things, execution domains tell Linux how to map signal numbers into signal actions. The execution domain system allows Linux to provide limited support for binaries compiled under other Unix-like operating systems.

This function will return the current personality() when persona equals 0xffffffff. Otherwise, it will make the execution domain referenced by persona the new execution domain of the current process.

**Name**

personality - set the process execution domain

**Synopsis**

#include <sys/personality.h>

int personality(unsigned long *persona*);

**Return Value**

On success, the previous persona is returned. On error, -1 is returned, and errno is set appropriately.

## Cloning:

Making an image copy of your system disk is a great way to create a backup. With the cost of portable USB drives at all time lows, you could keep a couple around for rotation purposes. If your main drive does crash, you could be back up and running in a matter of minutes. Now all that's left is making the actual image copy. There are lots of ways to accomplish this task, and we'll spell a few of them out to help you along.

The biggest issue with making an image copy of your system disk is that you must boot from another device to get a true copy. Options include a "live" CD or bootable USB. You probably have a copy of your favorite distribution's installation disk lying around somewhere, so that would be an obvious choice. For our purposes we'll use the Ubuntu 10.4 distro on a USB disk. The second option would be to use an actual disk duplication distro like Clonezilla. This turns out to be one of the easier ways to get the job done, especially if you're not too comfortable with the command line.

**Option One: Bootable Ubuntu USB Disk:**

Creating a bootable Ubuntu USB disk is a simple task if you have a running system. It's really not that hard if you don't either. The one thing you do need is the distribution ISO file.

You should be able to boot straight from the disk once you have it created. It's possible you might have to change your BIOS setting to allow the system to boot from USB. Many newer systems (like Dell machines) have an option to bring up a boot menu when the machine first powers up by pressing the F12 key. Once you get the system booted you're ready to make your backup image copy. You might want to run the Disk Utility found under the System / Administration list. This will give you the opportunity to look at the disks attached to your system and their organization.

The Disk Utility provides a number of utilities including mount / dismount and format volume. It's a good idea to go ahead and format the drive if you're reusing an old Windows disk. GParted 0.5.1 comes standard with the basic Ubuntu 10.04 boot disk. It includes an option to copy a partition. Instructions for doing the work can be found on either the GParted site or on the Ubuntu forums. There's also a GParted LiveCD if you want to go that route.

Be prepared to wait a while if you choose to backup your system to an external USB drive. The total estimated time in our case was almost four hours. One really good alternative is to use a disk caddy adapter like the Thermaltake BlackX ST0005U. It has an eSATA connector that speeds up the data transfer process tremendously. This is a must have if you're the type that frequently tears into systems or builds new ones.

**Option Two: Clonezilla:**

Clonezilla is a Linux distribution specifically created for the purpose of cloning disk drives. Itworks for virtually any file system you can think of. Clonezilla comes in two basic flavors, Live and SE. The live version works in much the same way as the Ubuntu Live USB disk. You boot your computer from the Live USB and perform the disk copy operations on any drives connected to the computer. Clonezilla uses a number of tools along with a simple menu system to help guide you through the process. The default partition copy tool is Partclone. Clonezilla SE (Server Edition) is meant to be used to clone disks over a network.

The latest release of Clonezilla is 1.2.5.17 and comes in either a Debian or Ubuntu version. You can now download an AMD64 version which includes support for 64-bit versions of all the applications and theimaging of large partitions. All the applications have been updated to the latest versions along with the 2.6.32-12 version of the Linux kernel. The hardest part of using Clonezilla to image your hard drives is making sure you know which drive is the master and which drive will be your copy. Clonezilla also takes care of copying the Master Boot Record (MBR) while accomplishing the same task with the Ubuntu LiveCD method requires some command line magic.

**Option Three: dd:**

If you're a command line wizard, you could always use the dd command. The command to image adrive with dd would be something like the following:

*# dd if=/dev/sda of=/dev/sdb*

This assumes that /dev/sda is the drive you wish to copy and /dev/sdb is the target drive. You'll find this method to be about the same speed as the GParted method mentioned in Option One above. It really doesn'tmatter which method you choose. The important thing is that you do some kind of system backup.

Computers do fail from time to time, and it seems to be just at the time when you can least afford it. Saveyourself some grief down the road and go backup your system now. Go ahead, we'll wait.

## Signals:

A signal is an event generated by Linux in response to some condition, which may
cause a process to take some action when it receives the signal. "Raise" is the term that indicates the generation of a signal. "Catch" is the term to indicate the receipt of a signal. Introduced in UNIX systems to simplify IPC. Used by the kernel to notify processes of system Events. A signal is a short message sent to a process, or group of processes, containing the number identifying the signal.POSIX.4 defines i/f for queuing& ordering RT signals w/ arguments. Linux supports 31 non-real-time signals.

Processes end because they receive a signal. There are multiple signals that you can send to aprocess. Use the **kill** command to send a signal to a process. The command **kill -l** shows a list of signals. Most signals are for internal use by the system or for programmers when they write code.

**Sending signals:**

- A program can signal a different program using the `kill()` system call with prototypeint kill(pid_t pid, int sig);

This will send the signal with number `sig` to the process with process ID `pid`. Signal numbers are small positive integers. (For the definitions on your machine, try `/usr/include/bits/signum.h`. Note that these definitions depend on OS and architecture.)

A user can send a signal from the command line using the `kill` command. Common uses are `kill -9` N to kill the process with pid N, or `kill -1` N to force process N (maybe `init` or `inetd`) to reread its configuration file.

Certain user actions will make the kernel send a signal to a process or group of processes: typing the interrupt character (probably Ctrl-C) causes SIGINT to be sent, typing the quit character (probably Ctrl-\) sends SIGQUIT, hanging up the phone (modem) sends SIGHUP, typing the stop character (probably Ctrl-Z)sends SIGSTOP.

Certain program actions will make the kernel send a signal to that process: for an illegal instruction one gets SIGILL, for accessing non existing memory one gets SIGSEGV, for writing to a pipe while nobody is listening anymore on the other side one gets SIGPIPE, for reading from the terminal while in the background one gets SIGTTIN, etc.

**Receiving signals:**

When a process receives a signal, a default action happens, unless the process has arranged to handle the signal. For the list of signals and the corresponding default actions, see signal(7). For example, by default SIGHUP, SIGINT, SIGKILL will kill the process; SIGQUIT will kill the process and force a core dump; SIGSTOP, SIGTTIN will stop the process; SIGCONT will continue a stopped process; SIGCHLD will be ignored.

Traditionally, one sets up a handler for the signal using the `signal` system call with prototypetypedef void (*sighandler_t)(int);
sighandler_t signal(int sig, sighandler_t handler);

This sets up the routine handler() as handler for signals with number sig. The return value is (the address of) the old handler. The special values SIG_DFL and SIG_IGN denote the default action and ignoring, respectively.

When a signal arrives, the process is interrupted, the current registers are saved, and the signal handler is invoked. When the signal handler returns, the interrupted activity is continued.

It is difficult to do interesting things in a signal handler, because the process can be interrupted in anarbitrary place, data structures can be in arbitrary state, etc. The three most common things to do in a signal handler are (i) set a flag variable and return immediately, and (ii) (messy) throw away all the program was doing, and restart at some convenient point, perhaps the main command loop or so, and (iii) clean up and exit.

**Blocking signals:**

Each process has a list (bitmask) of currently blocked signals. When a signal is blocked, it is not delivered (that is, no signal handling routine is called), but remains pending.

- The `sigprocmask ()` system call serves to change the list of blocked signals.
- The `sigpending ()` system call reveals what signals are (blocked and) pending.
- The `sigsuspend ()` system call suspends the calling process until a specified signal is received.Linux Common Signals are:

| Signal name | Signal number | Meaning |
|---|---|---|
| SIGHUP | 1 | Hangup (POSIX) |
| SIGINT | 2 | Terminal interrupt (ANSI) |
| SIGQUIT | 3 | Terminal quit (POSIX) |
| SIGILL | 4 | Illegal instruction (ANSI) |
| SIGTRAP | 5 | Trace trap (POSIX) |
| SIGIOT | 6 | IOT Trap (4.2 BSD) |
| SIGBUS | 7 | BUS error (4.2 BSD) |
| SIGFPE | 8 | Floating point exception (ANSI) |
| SIGKILL | 9 | Kill(can't be caught or ignored) (POSIX) |
| SIGUSR1 | 10 | User defined signal 1 (POSIX) |

## Development with Linux:

The following areas are Linux development

1. Compilers
2. Development Tools
3. IDE: Integrated Development Environment
4. Memory Debugging Tools
5. Software Design

### Compilers

A compiler that allows a computer program written on one type of computer to be used on another type.

| Language | Linux Compiler |
|---|---|
| C/C++/FORTRAN | GNU C compiler man page (gcc)<br>GNU C++ compiler man page (g++) |
| Java | gcj - GNU JAVA compiler. |
| C# | Ximian: MONO |
| ADA | GNAT |
| LISP | CLISP interpreter, compiler and debugger<br>CMU Common Lisp |
| Cobol | TinyCobol Home Page |
| Pascal | Free Pascal Home page, documentation and download |

**Development Tools**

**C++:**

| Tool | Description |
|---|---|
| c++filt | Demangle C++ symbols |
| SWIG.org | Interface compiler for wrapper routines which run from test scripts for testing "C" and "C++". Works with Perl, Python, Ruby, and Tcl andallows scripts to invoke "C" and "C++" functions, libraries, etc. |
| ParaSoft C++ Test | Static source code tester and automated unit test cases. |

C:

| Tool | Description |
|---|---|
| bison | GNU Project parser generator (yacc replacement) |
| pccts | Purdue Compiler-Compiler (yacc/lex combo) |
| flex | fast lexical analyzer generator |

**Java:**

| Tool | Description |
|---|---|
| kaffe | Java virtual machine |

**DataBase:**

| Tool | Description |
|---|---|
| gdbm | The GNU database manager. |
| Berkeley DB | Open Source embedded database system/toolkit |
| SQLite.org | Open Source embedded database system/toolkit |

### 2.6.3. IDE: Integrated Development Environment

| Language | Linux Compiler |
|---|---|
| Eclipse.org | IBM open source JAVA and C/C++ (with CDT plug-in) IDE. Included with FC4. Extensible IDE consortium - Borland, IBM, Red Hat, Rational.Lots of industry backing. (Open Source) Also see EclipsePluginCentral.com Plugins available for Subversion |

|  |  |
|---|---|
|  | SCM, XML documents, HEX, ... Also see: YoLinux C++ Development |
| Anjuta | C, C++. Written for GTK/Gnome. Solid, simple, intuitive, bug free IDE for C/C++ development on Linux. Search/Indexing, edit, compile and debug. Almost no learning curve for those familiar with IDE's. |
| C++ KDE IDE | C++ KDE IDE |
| Sun Studio | C/C++, FORTRAN IDE for Linux. (Free and supported x86 versions) |
| Source Navigator | C/C++, FORTRAN, COBOL, Tcl, JAVA, asm editor, cross reference tool, class browser and IDE. |
| Cobol | TinyCobol Home Page |
| Pascal | Free Pascal Home page, documentation and download |

**Memory Debugging`**

| Language | Linux Compiler |
|---|---|
| GNU Checker | Warns of using uninitialized variable or memory or unallocated memory. |
| dbmalloc | C malloc, memory, string, and bstring |
| fda | FDA provides a toolkit for validating pointers, checking for leaks, gathering memory statistics, bounds checking... |
| MCheck | C/C++ memory usage and malloc checker for x86/linux. Detects accesses to uninitialized variables, bad pointers, double frees and memory leaks. |
| YAMD | Yet Another Malloc Debugger: C/C++ |

**Software Design**

Argo - GUI UML tool.

GraphViz - ATT Graph Visualization for graph layout. Used by dOxygen for class diagram annotation.

Umbrello UML Modeller IDeogramic - UML modeling tool. (Commercial product)

DIA: DIA links/toolsPoseidon - Written in JAVA to support all platforms including Linux. Eclipse pluginsupport.

Medoosa - cpp2dia - C++ to UML

# UNIT III

# OPEN SOURCE DATABASE

MySQL: Introduction – Setting up account – Starting, terminating and writing your Own SQL programs – Record selection Technology – Working with strings – Date andTime– Sorting Query Results – Generating Summary – Working with metadata – Using sequences – MySQL and Web.

## MySQL:

## Introduction:

MySQL is the most popular open source SQL database management system (DBMS), is developed, distributed, and supported by MySQL AB. MySQL AB is a commercial company, founded by the MySQL developers.

A database is a collection of data that is organized so that its contents can be easily accessed, managed and updated. MySQL is a data storage area. In this storage area, there are small sections called Tables. A Relational Database Management System (RDBMS) may be a DBMS in which data is stored in the form of tables and the relationship among the data is also stored in the form of tables.

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

**Advantages:**

- *MySQL is Cross-Platform. It runs on more than 20 platforms including Linux, Windows, Mac OS, Solaris, HP-UX, IBM AIX, giving you the kind of flexibility that puts you in control.*
- *MySQL is fast. It is used by a variety of corporations that demand performance and stability.*
- *MySQL is free. It is Open Source software. As such you are free to examine the source code andmake any changes you wish. As per its GPL license, you are free to redistribute those changes as long as your software is also Open Source.*
- 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.

## Setting up Account:

In order to provide access the MySQL database you need to create an account to use for connectingto 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 Unix MySQL user names can be up to 16 characters long.

MySQL passwords have nothing to do with passwords for logging in to your operating system. Connecting to a MySQL server requires a username and password. You can also specify the name of the host where the server is running. If you don't specify connection parameters explicitly, *mysql* assumes default values. For example, if you specify no hostname, *mysql* typically assumes the server is running onthe local host.

**Adding User Accounts:**

You can create MySQL accounts in two ways:

- By using statements intended for creating accounts, such as CREATE USER or GRANT. Thesestatements cause the server to make appropriate modifications to the grant tables.
- By manipulating the MySQL grant tables directly with statements such as INSERT, UPDATE, orDELETE.
- Account Management Statements:

### *CREATE USER:*

The CREATE USER statement creates new MySQL accounts

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

Syntax: DROP USER user [, user] ...

Example:
GRANT ALL PRIVILEGES ON *.* TO 'monty'@'localhost' WITH GRANT OPTION;

**RENAME USER :**

The `RENAME USER` statement renames existing MySQL accounts.

Syntax

RENAME USER old_user TO new_user

[, old_user TO new_user] ...

Example

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

### *REVOKE:*

The `REVOKE` statement enables system administrators to revoke privileges from MySQL accounts
Syntax:

REVOKE

priv_type [(column_list)]

[, priv_type [(column_list)]] ...ON [object_type] priv_level FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION

FROM user [, user] ...
Example:

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

### *3.2.5 SET PASSWORD :*

The `SET PASSWORD` statement assigns a password to an existing MySQL user.
Syntax

SET PASSWORD [FOR user] =

{

Example:
PASSWORD('some password')
| OLD_PASSWORD('some password')

| 'encrypted password'

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

## **Starting, terminating and writing your Own SQL programs:**

**Starting MySQL:**

- The MySQL server can be started manually from the command line.
- To start the **mysqld** server from the command line, you should start a console window (or “DOSwindow”) and enter this command:

shell> **"C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqld"**

- The path to **mysqld** may vary depending on the install location of MySQL on your system.

**Login:**

- To start the *mysql* program, type *myql* at your command-line prompt.
- If *mysql*starts up correctly, you'll see a short message, followed by a mysql> prompt that indicatesthe 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 anyfurther examples):
  - % **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 specifyconnection parameters.
- The most commonly needed parameters are the host to connect to (the host where the MySQL serverruns), your MySQL username, and a password.
- For example:
  - % **mysql -h localhost -p -u cbuser**
  - Enter password: **cbpass**

**Stopping MySQL:**

You can stop the MySQL server by executing this command:

shell> **`"C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqladmin" –u root shutdown`**

**Logout:**

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 typingCtrl-D.

## <u>Writing your Own SQL programs:</u>

When existing software doesn't do what you want, you can write your own programs. We are goingto create a simple login system using PHP code on our pages, and a MySQL database to store our user’s information.

Sample Coding

```
                <?php
$host="localhost"; // Host name
$username=""; // Mysql username
$password=""; // Mysql password
$db_name="test"; // Database name
$tbl_name="members"; // Table name

// Connect to server and select databse.
mysql_connect("$host", "$username", "$password")or die("cannot
connect");mysql_select_db("$db_name")or die("cannot select
DB");

// username and password sent from form
$myusername=$_POST['myusername'];
$mypassword=$_POST['mypassword'];
$sql="SELECT * FROM $tbl_name WHERE username='$myusername' and
password='$mypassword'";
$result=mysql_query($sql);

// Mysql_num_row is counting table row
$count=mysql_num_rows($result);
// If result matched $myusername and $mypassword, table row must be 1 row
if($count==1)
{

}
else
{

}
?>
// Register $myusername, $mypassword and redirect to file "login_success.php"
session_register("myusername");
session_register("mypassword"); header("location:login_success.php");
echo "Wrong Username or Password";
```

## Record selection Technology:

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

- SELECT
- FROM
- WHERE

The SELECT clause specifies the table columns that are retrieved. The FROM clause specifies the tables accessed. The WHERE clause specifies which table rows are used. The WHERE clause is optional; if missing, all table rows are used.

For example,

**SELECT name FROM emp WHERE city='Rome'**

This query accesses rows from the table emp. It then filters those rows where the *city* column contains Rome. Finally, the query retrieves the *name* column from each filtered row. Using the example *s* table, this query produces:

```
          +----------------------+
           |        name        |
          +----------------------+
    |     aravindhan      |
    |     anandkumar    |
   +-----------------------+
```

SELECT gives you control over several aspects of record retrieval:

- Which table to use
- Which columns to display from the table
- What names to give the columns
- Which rows to retrieve from the table
- How to sort the rows

Checking what version of the server you're running or the name of the current database:mysql> **SELECT VERSION( ), DATABASE( );**

```
--------------------+----------  -------------------+--------          +
          | VERSION( ) | DATABASE( )|
--------------------+----------  -------------------+--------          +
          | 3.23.51-log | cookbook |
--------------------+----------  -------------------+--------          +
```

**Specifying Which Columns to Display**

To display some or all of the columns from a table. Use * as a shortcut that selects all columns. Orname the columns you want to see explicitly.

For example,

mysql> **SELECT * FROM emp;**

```
--------------------+-----------------------------  -------------------+  -------------------+          +
          |          name          |    age    |     exp  |
--------------------+-----------------------------  -------------------+  -------------------+          +
          |      AnandKumar      |    45     |     12    |
--------------------+-----------------------------  -------------------+  -------------------+          +
```

mysql> **SELECT name,age FROM emp;**

```
--------------------+-----------------------------  -------------------+          +
          |          name          |    age    |
--------------------+-----------------------------  -------------------+          +
          |      AnandKumar      |    45     |
--------------------+-----------------------------  -------------------+          +
```

**Giving Names to Output Columns:**

Whenever you retrieve a result set, MySQL gives every output column a name. (That's how the *mysql* program gets the names that you see displayed as the initial row of column headers in result set output.) MySQL assigns default names to output columns, but if the defaults are not suitable, you can use column aliases to specify your own names.

mysql> **SELECT name,age FROM emp;**

```
-----------------+-------------------------  -----------------+        +
                 |          name           |    age    |
-----------------+-------------------------  -----------------+        +

                 |       AnandKumar        |    45     |
-----------------+-------------------------  -----------------+        +
```

mysql> **SELECT Emp.Name as name, Emp.Age as age FROM emp;**

```
-----------------+-------------------------  -----------------+        +
                 |       Emp.Name          | Emp.Age |
-----------------+-------------------------  -----------------+        +
                 |       AnandKumar        |    45     |
-----------------+-------------------------  -----------------+        +
```

**Specifying Which Rows to Select:**

Add a WHERE clause to the query that indicates to the server which rows to return. Unless you qualify or restrict a SELECT query in some way, it retrieves every row in your table, which may be a lot more information than you really want to see.

mysql> **SELECT name,age FROM emp WHERE name='aravind';**

```
-----------------+-------------------------  -----------------+        +
                 |          name           |    age    |
-----------------+-------------------------  -----------------+        +
                 |         aravind         |    45     |
-----------------+-------------------------  -----------------+        +
```

**Removing Duplicate Rows:**

Use DISTINCT a query eliminate duplicate values. Some queries produce results containing
duplicate records.For example,

mysql> **SELECT name FROM emp;**

```
-----------------+          +
                 |   name   |
-----------------+          +
                 | aravind |
                 | anand    |
                 | Naveen   |
                 | siva     |
                 | araivnd |
| anand    |
+----------+
```

But that result is heavily redundant. Adding DISTINCT to the query removes the duplicaterecords, producing a set of unique values:

mysql>
**SELECT name FROM emp;**

```
+----------+
|   name   |
+----------+
| aravind  |
| anand    |
| Naveen   |
| siva     |
+----------+
```

**Working with NULL Values:**

To summarizing a set of values that may include NULL values and you need to know how tointerpret the results. Most aggregate functions ignore NULL values.

mysql> **SELECT subject, test, score FROM expt ORDER BY subject, test;**

```
+---------+------+-------+
| subject | test | score |
+---------+------+-------+
| Arun    |    A |    47 |
| Chitra  |    B |    50 |
| David   |    C |  NULL |
| Ram     |    D |  NULL |
+---------+------+-------+
```

mysql> **SELECT subject name FROM expt WHERE score IS NULL GROUP BY subject;**

```
+----------+
| Name     |
+----------+
| David     |
| Ram       |
+----------+
```

**Sorting a Result Set:**

Add an ORDER BY clause to tell it exactly how you want things sorted. 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. There are lots of ways to use sorting techniques.

mysql> **SELECT name,age FROM emp WHERE age > 24 ORDER BY name;**

```
+--------+-----+
|  name  | age |
+--------+-----+
| Arun   |  35 |
| Ram     |  45  |
            | David    |  34  |
            | Chitra   |  56  |
------------+ ---------  +      +
```

mysql> **SELECT name,age FROM emp WHERE age > 24 ORDER BY name ASC;**

```
------------+ ---------  +      +
            |  name    | age  |
------------+ ---------  +      +
            | Arun     |  35  |
            | Chitra   |  45  |
            | David    |  34  |
            | Ram      |  56  |
------------+ ---------  +      +
```

## Working with Strings:

A string can be binary or nonbinary. Binary strings are used for raw data such as images, music files,or encrypted values. Nonbinary strings are used for character data such as text and are associated with a character set and collation (sorting order). A character set determines which characters are legal in a string. Collations can be chosen according to whether you need comparisons to be case-sensitive or case- insensitive, or to use the rules of a particular language.

Data types for binary strings are `BINARY`, `VARBINARY`, and `BLOB`. Data types for nonbinary strings are `CHAR`, `VARCHAR`, and `TEXT`, each of which allows `CHARACTER SET` and `COLLATE` attributes.See Recipe 5.2 for information about choosing data types for string columns.

You can convert a binary string to a nonbinary string and vice versa, or convert a nonbinary string from one character set or collation to another.

You can use a string in its entirety or extract substrings from it. Strings can be combined with other strings.

You can apply pattern-matching operations to strings. `FULLTEXT` searching is available for efficientqueries on large collections of text.

**Strings properties:**

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

**Types of Strings**

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

A binary column type is one that contains binary strings. Some of MySQL's column types are binary (case sensitive) and others are not, as illustrated here:

| Column type | Binary/case sensitive |
|---|---|
| CHAR, VARCHAR | No |
| CHAR BINARY, VARCHAR BINARY | Yes |
| TEXT | No |
| BLOB | Yes |
| ENUM,SET | No |

A characteristic of nonbinary strings is that they have a character set. To see which character sets areavailable, use this statement:

mysql> **SHOW CHARACTER SET;**

| Charset | Description | Default collation | Maxlen |
|---|---|---|---|
| big5 | Big5 Traditional Chinese | big5_chinese_ci | 2 |
| dec8 | DEC West European | dec8_swedish_ci | 1 |
| cp850 | DOS West European | cp850_general_ci | 1 |
| hp8 | HP West European | hp8_english_ci | 1 |
| koi8r | KOI8-R Relcom Russian | koi8r_general_ci | 1 |
| latin1 | cp1252 West European | latin1_swedish_ci | 1 |
| latin2 | ISO 8859-2 Central Europea n | latin2_general_ci | 1 |
| ... | | | |
| utf8 | UTF-8 Unicode | utf8_general_ci | 3 |
| ucs2 | UCS-2 Unicode | ucs2_general_ci | 2 |
| ... | | | |

The default character set in MySQL is **latin1**.

The following example illustrates how collation affects sort order. Suppose that a table contains alatin1 string column and has the following rows:

mysql> **CREATE TABLE t (c CHAR(3) CHARACTER SET latin1);**

mysql> **INSERT INTO t (c) VALUES('AAA'),('bbb'),('aaa'),('BBB');**

mysql> **SELECT c FROM t;**

```
----------+         +
          |   C     |
----------+         +
          | AAA     |
          | bbb     |
          | aaa     |
          | BBB     |
----------+         +
```

**String Data Types:**

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

**CHAR() :**

It is a fixed length string and is mainly used when the data is not going to vary much in it's length. It ranges from 0 to 255 characters long. While storing CHAR values they are right padded with spaces to the specified length. When retrieving the CHAR values, trailing spaces are removed.

**VARCHAR() :**

It is a variable length string and is mainly used when the data may vary in length. It ranges from 0 to255 characters long. VARCHAR values are not padded when they are stored.

**Note:**

If CHAR and VARCHAR options are used in the same table, then MySQL will automatically change the CHAR into VARCHAR for compatability reasons. The ( ) bracket allows to enter a maximum number of characters that will be used in the column.

**TINYTEXT, TINYBLOB :**

A string with a maximum length of 255 characters.

**TEXT :**

TEXT columns are treated as character strings(non-binary strings). It contains a maximum length of 65535 characters.

**BLOB :**

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

**MEDIUMTEXT, MEDIUMBLOB :**

It has a maximum length of 16777215 characters.

**LONGTEXT, LONGBLOB :**

It has a maximum length of 4294967295 characters.

**BINARY :**

The BINARY is similar to the CHAR type. It stores the value as binary byte strings instead of non-binary character strings.

**VARBINARY :**

The VARBINARY is similar to the VARCHAR type. It stores the value as binary byte stringsinstead of non-binary character strings.

**ENUM() :**

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

**SET() :**

A set. Each column may have more than one of the specified possible values. It contains up to 64 list items and can store more than one choice. SET values are represented internally as integers.

**Choosing a String Data Type**

Choose the data type according to the characteristics of the information to be stored and how you need to use it. Consider questions such as these:

- Are the strings binary or nonbinary?
- Does case sensitivity matter?
- What is the maximum string length?
- Do you want to store fixed- or variable-length values?
- Do you need to retain trailing spaces?
- Is there a fixed set of allowable values?

MySQL provides several binary and nonbinary string data types. These types come in pairs as shown in thefollowing table.

| **Binary data type** | **Nonbinary datatype** | **Maximum length** |
|---|---|---|
| BINARY | CHAR | 255 |
| VARBINARY | VARCHAR | 65,535 |
| TINYBLOB | TINYTEXT | 255 |
| BLOB | TEXT | 65,535 |
| MEDIUMBLOB | MEDIUMTEXT | 16,777,215 |
| LONGBLOB | LONGTEXT | 4,294,967,295 |

## Date and Time Data Types:

MySQL supports a number of date and time column formats. For all the date and time columns, we canalso assign the values using either string or numbers.

- DATE
- TIME
- DATETIME
- TIMESTAMP
- YEAR

**DATE:**

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

**TIME:**

A Time. The range is -838:59:59 to 838:59:59. The time values are displayed in HH:MM:SS format.

**DATETIME:**

A Date and Time combination. The range is 1000-01-01 00:00:00 to 9999-12-31 23:59:59. The date timevalues are displayed in YYYY-MM-DD HH:MM:SS format.

**TIMESTAMP:**

A Timestamp. The range is 1970-01-01 00:00:01 UTC to partway through the year 2037. A TIMESTAMP column is useful for recording the date and time of an INSERT or UPDATE operation.

**YEAR:**

A Year. The year values are displayed either in two-digit or four-digit format. The range of values for a four-digit is 1901 to 2155. For two-digit, the range is 70 to 69, representing years from 1970 to 2069.

For Example

mysql> **SELECT t1, t2 FROM time_val;**

```
+----------+----------+
|    t1    |    t2    |
+----------+----------+
| 15:00:00 | 15:00:00 |
| 05:01:30 | 02:30:20 |
| 12:30:20 | 17:30:45 |
+----------+----------+
```

mysql> **SELECT d FROM date_val;**

```
+------------+
|     d      |
+------------+
| 1864-02-28 |
| 1900-01-15 |
| 1987-03-05 |
| 1999-12-31 |
| 2000-06-04 |
+------------+
```

**Date and Time Formats:**

Whilst it is most common to store dates using a dash (-) as the delimiter and a colon (:) as the time delimiter it is in fact possible to use any character, or no character between the date and time segments. For example, the following formats all achieve the same result:

2008-10-23 10:37:22

20081023103722

2008/10/23 10.37.22

2008*10*23*10*37*22

**Date and Time Functions:**

In addition to providing mechanisms for storing dates and times, MySQL also provides a wide range of functions that can be used to manipulate dates and times. The following table provides a list of the more common functions available for working with times and dates in MySQL:

| Function | Description |
|---|---|
| ADDDATE() | Add dates |
| ADDTIME() | Add time |
| CONVERT_TZ() | Convert from one timezone to another |
| CURDATE() | Returns the current date |
| CURTIME() | Returns the current system time |
| DATE_ADD() | Add two dates |
| DATE_FORMAT() | Format date as specified |
| DATE_SUB() | Subtract two dates |
| DATE() | Extract the date part of a date or datetime expression |
| DATEDIFF() | Subtract two dates |
| DAYNAME() | Returns the name of the weekday |
| DAYOFMONTH() | Returns the day of the month (1-31) |
| DAYOFWEEK() | Returns the weekday index of the argument |
| DAYOFYEAR() | Returns the day of the year (1-366) |
| EXTRACT | Extract part of a date |
| FROM_DAYS() | Convert a day number to a date |
| FROM_UNIXTIME() | Format date as a UNIX timestamp |
| GET_FORMAT() | Returns a date format string |
| HOUR() | Extract the hour |
| LAST_DAY | Returns the last day of the month for the argument |
| MAKEDATE() | Create a date from the year and day of year |
| MAKETIME | MAKETIME() |
| MICROSECOND() | Returns the microseconds from argument |
| MINUTE() | Returns the minute from the argument |
| MONTH() | Returns the month from the date passed |
| MONTHNAME() | Returns the name of the month |
| NOW() | Returns the current date and time |
| PERIOD_ADD() | Add a period to a year-month |
| PERIOD_DIFF() | Returns the number of months between two periods |
| QUARTER() | Returns the quarter from a date passed as an argument |
| SEC_TO_TIME() | Converts seconds to 'HH:MM:SS' format |
| SECOND() | Returns the second (0-59) |

|  |  |
|---|---|
| STR_TO_DATE() | Convert a string to a date |
| SUBTIME() | Subtract times |
| SYSDATE() | Returns the time at which the function executes |
| TIME_FORMAT() | Format as time |
| TIME_TO_SEC() | Returns the argument converted to seconds |
| TIME() | Extract the time portion of the expression passed as an argument |
| TIMEDIFF() | Subtract time |
| TIMESTAMP() | With a single argument, this function returns the Date or the Datetime expression.With two arguments, the sum of the arguments is returned |
| TIMESTAMPADD() | Add an interval to a datetime expression |
| TIMESTAMPDIFF() | Subtract an interval from a datetime expression |
| TO_DAYS() | Returns the date argument converted to days |
| UNIX_TIMESTAMP() | Returns a UNIX timestamp to a format acceptable to MySQL |
| UTC_DATE() | Returns the current Universal Time (UTC) date |
| UTC_TIME() | Returns the current Universal Time (UTC time |
| UTC_TIMESTAMP() | Returns the current Universal Time (UTC) date and time |
| WEEK() | Returns the week number |
| WEEKDAY() | Returns the weekday index |
| WEEKOFYEAR() | Returns the calendar week of the date (1-53) |
| YEAR() | Returns the year |
| YEARWEEK() | Returns the year and week |

**Display Dates or Times:**

Use the DATE_FORMAT( ) or TIME_FORMAT( ) functions to rewrite the DATE_FORMAT( ) function, which takes two arguments: a DATE, DATETIME, or TIMESTAMP value, and a string describinghow to display the value.

Within the formatting string, you indicate what to display using special sequences of the form %*c*, where *c* specifies which part of the date to display.

For example, %Y, %M, and %d signify the four-digit year, the month name, and the two-digit day of the month.

The following query shows the values in the date_val table, both as MySQL displays them by defaultand as reformatted with DATE_FORMAT( ):

```
mysql> SELECT d, DATE_FORMAT(d,'%M %d, %Y') FROM date_val;
```

```
+------------+-----------------------+
|     d      | DATE_FORMAT(d,'%M %d, %Y') |
+------------+-----------------------+
| 1864-02-28 |     February 28, 1864     |
| 1900-01-15 |     January 15, 1900      |
| 1987-03-05 |      March 05, 1987       |
| 1999-12-31 |     December 31, 1999     |
| 2000-06-04 |     June 04, 2000         |
+------------+-----------------------+
```

## 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.

**Using ORDER BY to Sort Query Results:**

Add an ORDER BY clause. ORDER BY will tell the MySQL server to sort the rows by a column.

Define in which direction to sort, as the order of the returned rows may not yet be meaningful. Rows can be returned in ascending or descending order.

Use ASC or DESC. Using ASC will sort the data so that you see the smallest number first. Using DESC will sort the data so that you see the largest number first. In this query, you are looking for customers with the largest negative balance first. ORDER BY will return the arrays with the greatest negative number (the smallest number) at the top.

**Syntax:**

Here is generic SQL syntax of SELECT command along with ORDER BY clause to sort data fromMySQL table:

SELECT field1, field2,...fieldN table_name1, table_name2...ORDER BY field1, [field2...] [ASC [DESC]]

- You can sort returned result on any field provided that filed is being listed out.
- You can sort result on more than one field.
- You can use keyword ASC or DESC to get result in ascending or descending order. By default itsascending order.
- You can use WHERE...LIKE clause in usual way to put condition.

**Example:**

```
mysql> SELECT * from tutorials_tbl ORDER BY tutorial_author ASC
+-------------+----------------+-----------------+-----------------+
| tutorial_id | tutorial_title | tutorial_author | submission_date |
+-------------+----------------+-----------------+-----------------+
|           2 | Learn MySQL    | Abdul S         | 2007-05-24      |
|           1 | Learn PHP      | John Poul       | 2007-05-24      |
|           3 | JAVA Tutorial  | Sanjay          | 2007-05-06      |
+-------------+----------------+-----------------+-----------------+
```

**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.

ORDER BY doesn't care how many rows there are; it sorts whatever rows the query returns. If you don't want to sort an entire table, add a WHERE clause to indicate which rows to select. For example, to sortthe records for just one of the drivers, do something like this:

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

```
------------------+-------- ---------------- +        +
                  | trav_date | miles |
------------------+-------- ---------------- +        +
                  | 2001-11-26 | 115 |
                  | 2001-11-27 | 96 |
                  | 2001-11-29 | 300 |
                  | 2001-11-30 | 203 |
                  | 2001-12-01 | 197 |
------------------+-------- ---------------- +        +
```

**Displaying One Set of Values While Sorting by Another**

To sort a result set using values that you're not selecting. You can use columns in the ORDER BY clause that doesn’t appear in the column output list. ORDER BY is not limited to sorting only those columnsnamed in the column output list. It can sort using values that are "hidden".

**Sorting and NULL Values**

To sort columns that may contain NULL values. If NULL values don't come out in the desired position within the sort order, False them into appearing where you want. When a sorted column contains NULL values, MySQL puts them all together in the sort order.

mysql> **SELECT NULL = NULL;**

```
------------------+----------            +
                  | NULL = NULL |
------------------+----------            +
                  | NULL |
------------------+----------            +
```

Normally, sorting puts the NULL values at the beginning:

mysql> **SELECT val FROM t ORDER BY val;**

```
-------------     +       +
                  | val |
-------------     +       +
                  | NULL |
                  | NULL |
                  | 3 |
                  | 9 |
                  | 100 |
-------------     +       +
```

**Date-Based Sorting**

To sort in temporal order. Sort using a date or time column type, ignoring parts of the values that are irrelevant if necessary. Many types of information include date or time information and it's very often necessary to sort results in temporal order. MySQL knows how to sort temporal column types, so there's no special trick to ordering values in DATE, DATETIME, TIME, or TIMESTAMP columns. Begin with atable that contains values for each of those types:

mysql> **SELECT * FROM temporal_val;**

```
+------------+---------------------+----------+----------------+
| d          |       dt            |     t    |        ts      |
+------------+---------------------+----------+----------------+
| 1970-01-01 | 1884-01-01 12:00:00 | 13:00:00 | 19800101020000 |
| 1999-01-01 | 1860-01-01 12:00:00 | 19:00:00 | 20210101030000 |
| 1981-01-01 | 1871-01-01 12:00:00 | 03:00:00 | 19750101040000 |
| 1964-01-01 | 1899-01-01 12:00:00 | 01:00:00 | 19850101050000 |
+------------+---------------------+----------+----------------+
```

**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. You ignore the year part of the dates and sort using only the month and day to order records in terms of where they fall during thecalendar year. Suppose you have an event table that looks like this when values are ordered by actual date of occurrence:

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

```
+------------+-------------------------------------+
|    date    |             description             |
+------------+-------------------------------------+
| 1215-06-15 | Signing of the Magna Carta          |
| 1732-02-22 | George Washington's birthday        |
| 1776-07-14 | Bastille Day                        |
| 1789-07-04 | US Independence Day                 |
| 1989-11-09 | Opening of the Berlin Wall          |
+------------+-------------------------------------+
```

## <u>Generating Summary:</u>

Summaries are useful when you want the overall picture rather than the details. Once the bulk (unpacked) data transfer is complete, a summary report of the migration will be displayed. Summary techniques allow you to answer questions such as "How many?" or "What is the total?" or "What is the range of values?" Summary operations in MySQL involve the following SQL constructs:

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 aggregatevalue 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;**

```
--------------------+----          +
                    | COUNT(*)  |
--------------------+----          +
                    | 30         |
--------------------+----          +
```

**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;**

```
--------------------+-------------          +
                    | Low    | High |
--------------------+-------------          +
                    |  100   | 500  |
--------------------+-------------          +
```

**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 emp_tab;**

```
--------------------+-------------  ---------------   _        +
                    | No-Of Emp | Avg Sal |
--------------------+----------------  ------------   _        +
                    | 23456      | 8500     |
--------------------+---------------------------------          +
```

**Using DISTINCT to Eliminate Duplicates:**

A summary operation that doesn't use aggregate functions is to determine which values or rows are contained in a dataset by eliminating duplicates. Do this with DISTINCT. DISTINCT is useful for boiling down a query result, and often is combined with ORDER BY to place the values in more meaningfulorder. For example, if you want to know the names of the employees listed in the emp_tab table, use the following query:

mysql> **SELECT DISTINCT emp_name FROM emp_tab ORDER BY emp_name;**

```
+--------+
| name   |
+--------+
| Arun   |
| John   |
| Ram    |
+--------+
```

A query without DISTINCT produces the same names, but is not nearly as easy to understand:

mysql> **SELECT emp_name FROM emp_tab;**

```
+--------+
| name   |
+--------+
| Arun   |
| Jhon   |
| Ram    |
| Ram    |
| Arun   |
| Arun   |
+--------+
```

If you want to know how many different drivers there are, use COUNT(DISTINCT):

mysql> **SELECT COUNT(DISTINCT emp_name) FROM emp_tab;**

```
+--------------------------+
| COUNT(DISTINCT emp_name) |
+--------------------------+
| 3                        |
+--------------------------+
```

**Dividing a Summary into Subgroups:**

To calculate a summary for each subgroup of a set of rows, not an overall summaryValue use a GROUP BY clause to arrange rows into groups.

Without using GROUP BY

mysql> **SELECT COUNT(*) FROM driver_log;**

```
+----------+
| COUNT(*) |
+----------+
| 10       |
+----------+
```

Using GROUP BY

mysql> **SELECT name, COUNT(name) FROM driver_log GROUP BY name;**

```
+-------+-------------+
| name  | COUNT(name) |
+-------+-------------+
| Ben   |      3      |
| Henry |      5      |
| Suzi  |      2      |
+-------+-------------+
```

**Summaries and NULL Values:**

To summarizing a set of values that may include NULL values and you need to know howto interpret the results. Most aggregate functions ignore NULL values.

mysql> **SELECT subject, test, score FROM expt ORDER BY subject, test;**

```
----------------+ ----------- ---+---------- +        +
                | subject | test | score |
----------------+ ----------- ---+---------- +        +
                | Arun    |    A |    47 |
                | Chitra  |    B |    50 |
                | David   |    C |  NULL |
                | Ram     |    D |  NULL |
----------------+ ----------- ---+---------- +        +
```

mysql> **SELECT subject name FROM expt WHERE score IS NULL GROUP BY subject;**

```
----------------+          +
                | Name     |
----------------+          +
                | David     |
                | Ram       |
----------------+          +
```

**Determining Whether Values are Unique:**

To know whether table values are unique use HAVING in conjunction with COUNT( ).

mysql> **SELECT trav_date, COUNT(trav_date) FROM driver_log GROUP BY trav_dateHAVING COUNT(trav_date) = 1;**

```
----------------+------- ----------------+-----------------          +
                | trav_date  | COUNT(trav_date) |
----------------+------- ----------------+-----------------          +
                | 2001-11-26 |        1         |
                | 2001-11-27 |        1         |
                | 2001-12-01 |        1         |
----------------+------- ----------------+-----------------          +
```

## <u>Working with Metadata:</u>

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.

**Representation of metadata must satisfy these requirements:**

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

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

**Types of Metadata:**

Information about the result of queries

When you delete or update a set of rows, you can determine the number of rows that were changed.

Information about tables and databases.

Information affecting to the structure of tables and databases is useful for applications that need to count a list of tables in a database or databases hosted on a server.

Information about the MySQL server.

Some APIs provide information about the database server or about the status of your current connection with the server.

**Obtaining Result Set Metadata:**

For queries that generate a result set, you can get a number of kinds of metadata.

**PHP:**

In PHP, metadata information is available after a successful call to mysql_query ( ) and remainsaccessible up to the point at which you call mysql_free_result( ). To access the metadata, pass the result set identifier returned by mysql_query( ) to the function thatreturns the information you want.

To get a row or column count for a result set, invoke mysql_num_rows( ) or mysql_num_fields( ).

Metadata information for a given column in a result set is packaged up in a single object.

## Using Sequences:

A sequence is a database object that generates numbers in sequential order. Applications most often use these numbers when they require a unique value in a table such as primary key values. The following list describes the characteristics of sequences.

- Sequences are available to all users of the database
- Sequences are created using SQL statements (see below)
- Sequences have a minimum and maximum value (the defaults are minimum=0 and maximum=$2^{63}$-1); they can be dropped, but not reset
- Once a sequence returns a value, the sequence can never return that same value

- While sequence values are not tied to any particular table, a sequence is usually used to generatevalues for only one table
- Sequences increment by an amount specified when created (the default is 1)

**Creating a Sequence:**

To create sequences, execute a CREATE SEQUENCE statement in the same way as an UPDATE orINSERT statement. The sequence information is stored in a data dictionary file.

The format for a CREATE SEQUENCE statement is as follows:

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

| **Variable** | **Description** |
|---|---|
| INCREMENT BY | The increment value. This can be a positive or negative number. |
| START WITH | The start value for the sequence. |
| MAXVALUE | The maximum value that the sequence can generate. If specifying NOMAXVALUE, the maximum value is $2^{63}$-1. |
| MINVALUE | The minimum value that the sequence can generate. If specifying NOMINVALUE, the minimum value is -$2^{63}$. |
| CYCLE | Specify CYCLE to indicate that when the maximum value is reached the sequence starts over again at the start value. Specify NOCYCLE to generate an error upon reaching the maximum value. |

**Dropping a Sequence:**

To drop a sequence, execute a DROP SEQUENCE statement. Use this function when a sequence is no longer useful, or to reset a sequence to an older number. To reset a sequence, first drop the sequence and then recreate it.Drop a sequence following this format:

**DROP SEQUENCE my_sequence**

**Using a Sequence:**

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

**NEXTVAL:** Returns the next value from the sequence.

**CURVAL:** Returns the value from the last call to NEXTVAL by the current user during the currentconnection.

For example, if User A calls NEXTVAL and it returns 124, and User B immediately calls NEXTVAL getting 125, User A will get 124 when calling CURVAL, while User B will get 125 while calling CURVAL. It is important to understand the connection between the sequence value and a particular connection to the database. The user cannot call CURVAL until making a call to NEXTVAL at least once on the connection. CURVAL returns the current value returned from the sequence on the current connection, not the current value of the sequence.

**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) VALUEScustomer_seq.NEXTVAL, 'Kalam', '123 Gandhi Nagar.')

To find the value just entered into the database:

SELECT customer_seq.CURVAL AS LAST_CUST_NU

## MySQL and Web:

- MySQL makes it easier to provide dynamic rather than static content. Static content exists as pagesin 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 youadd, modify, or delete those documents.
- By contrast, dynamic content is created on demand.

**Basic Web Page Generation:**

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

- HTML stands for **H**yper **T**ext **M**arkup **L**anguage
- HTML is not a programming language, it is a **markup language**
- A markup language is a set of **markup tags**
- HTML uses **markup tags** to describe web pages
- HTML documents **describe web pages**
- HTML documents **contain HTML tags** and plain text
- HTML documents are also **called web pages.**

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

```
<Html>
<Body>
<H1>My Heading</H1>
Welcome To My Web Page
<P>My Paragraph</P>
<body bgcolor="green">
</Body>
</Html>
```

**Static Web Page**: A static web page shows the required information to the viewer, but do not acceptany 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** 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**, (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 SupercomputingApplications).
- 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.

There are typically several directories under the Apache root directory, which I'll assume hereto be

/usr/local/apache. These directories include:

bin : Contains httpd—that is, Apache itself—and other Apache-related executable programs.

conf : For configuration files, notably httpd.conf, the primary file used by Apache.

Htdocs : The root of the document trees

# UNIT IV

# OPEN SOURCE PROGRAMMING LANGUAGES

PHP: Introduction – Programming in web environment – variables – constants – Data types – operators – Statements – Functions – Arrays – OOP – String Manipulation and regular expression – File handling and data storage.

## PHP: Introduction, Programming in web environment:

PHP is a server-side scripting language. PHP stands for Hypertext Preprocessor. This means that the script is run on your web server, not on the user's browser, so you do not need to worry about compatibility issues. PHP is relatively new (compared to languages such as Perl (CGI) and Java) but is quickly becoming one of the most popular scripting languages on the internet. PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.) PHP is an open source software PHP is free to download and use.

PHP files can contain text, HTML tags and scripts.PHP files are returned to the browser as plain HTML.PHP files have a file extension of ".php", ".php3", or ".phtml".

PHP enables you to build large, complex, and dynamic websites. PHP can also increase yourproductivity enormously, both in development time and maintenance time.

Using PHP, you can build websites that do things such as:

- Query a database
- Allow users to upload files
- Create/read files on the server (for example, the files that your users upload)
- Have a "member's area" (i.e. via a login page)
- Have a shopping cart
- Present a customized experience (for example, based on users' browsing history)
- Much, much more

### Creating a PHP File:

To create a PHP file, simply do the following:

- Create a new file in your favorite editor
- Type some PHP code
- Save the file with a .php extension

The .php extension tells the web server that it needs to process this as a PHP file. If you accidentally save it with a .html extension the server won't process your PHP code and the browser will just output it all to the screen.

**Simple Example:**

Every block of PHP code must start with <?php and end with ?>. The following exampleoutputs the text "PHP is easy!" to the screen:

```
<html>

<head>

<title>PHP Syntax Example</title>

</head>

<body>

<?php

echo "PHP is easy!";

?>

</body>

</html>
```

**Comments**

While there is only one type of comment in HTML, PHP has two types. The first type we will discuss is the single line comment. The single line comment tells the interpreter to ignore everything that occurs on that line to the right of the comment. To do a single line comment type "//" or "#" and all text tothe right will be ignored by PHP interpreter.

Example:

```
<html>

<body>
<?php
//This is a comment

/*This isa commentblock
```

```
        */
        ?>
        </body>
        </html>
```

## <u>Variables:</u> Introduction

Variables are named "containers" that allow you to store a value or Variables are nothing but identifiers to the memory location to store data. This value can then be used by other parts of the application, simply by referring to the variable's name. For example, the application could display the contents of the variable to the user.

In PHP, variable names must start with a dollar sign ($). For example:

```
<?php
$myVariable = "PHP is easy!";echo $myVariable;
?>
```

- **Variable Names:**

When creating names for your variables, you need to ensure they comply with the following naming rules:

- All PHP variable names must start with a letter or underscore ( _ )"
- Variable names can only contain alpha-numeric characters and underscores ( i.e. a-z, A-Z, 0-9, or _ )
- Variable names must not contain spaces. For multi-word variable names, either separate thewords with an underscore ( _ ) or use capitalization.

- **PHP is a Loosely Typed Language:**

PHP is a loosely type language, which means that you do not need to declare a variable beforeassigning a value to it .In PHP, the variable is declared automatically when you use it.

For Example,

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

- **String Variables in PHP:**

A string variable is used to store and manipulate text. String variables are used for values that containscharacters. For Example,

```
<?php

$txt="Hello World";echo $txt;

?>
```

- **The Concatenation Operator:**

There is only one string operator in PHP.The concatenation operator (.) is used to put two string values together.To concatenate two string variables together, use the concatenation operator:

```
<?php
$txt1="Hello World!";

$txt2="What a nice day!";echo $txt1 . " " . $txt2;

?>
Output: Hello World! What a nice day!
```

- **The strlen() function:**

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

```
<?php

echo strlen("Hello world!");
?>
Output: 12
```

- **The strpos() function:**

The strpos() function is used to search for character within a string.

```
<?php

echo strpos("Hello world!","world");

?>
```

Output: 6

## Constants: Introduction:

A PHP constant is a variable that cannot be changed after the initial value is assigned during the execution of the script, except for magic constants, which are not really constants.

You create a constant variable using the define () function. The function accepts 2 arguments.The first must be a string and represents the name which will be used to access the constant. A constant name is case-sensitive by default and by convention, are always uppercase. A valid constant name starts with a letter or underscore, followed by any combination of letters, numbers or underscores, however, by convention only capital letters and underscore symbols are used. Constant names must be in quotes when they are defined

The second argument is the value of the constant and can be a string or numbers and can be setexplicitly or as as the result of a function or equation.

The scope of a constant is global, which means that you can access constants anywhere in your scriptwithout regard to scope.

PHP has many pre-defined constants that can be used in your scripts.

```
<?php

    define ("SECE", "Sri Eshwar College Of Engineering");echo SECE;

?>
```

- **Magic Constants:**

Magic constants, as they are called, are not acutally constants but effectively behave like constants and are pre-defined. Magic constants use the same naming convention as PHP constants. The full list of magic constants can be found here.

There are 7 magic constants that change depending on where they are used. They are listed below.

| name | description |
|---|---|
| __LINE___ | The current line number of the file. |
| __ 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__ | he class name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the class T name as it was declared (case-sensitive). In PHP 4 its value is always lowercased. |
| __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 with symlinks resolved whereas in older versions it contained relative path under some circumstances. |
| __DIR__ | The directory of the file. If used inside an include, the directory of the included file is returned. This is equivalent to dirname(__FILE__). This directory name does not have a trailing slash unless it is the root directory. (Added in PHP 5.3.0.) |
| __METHOD__ | The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive). |
| | The name of the current namespace (case-sensitive). This constant is definedincompile-time (Added in PHP 5.3.0 |

## Data Types:

PHP scripts deal with data in one form or another, usually stored in variables. PHP can work with different types of data. For example, a whole number is said to have an integer data type, while a string of text has a string data type.

- **PHP's scalar data types:**

Scalar data is data that only contains a single value. As of version 6, PHP features 6 scalar data types:

| Type | Description | Example value |
|---|---|---|
| Integer | A whole number | 7, -23 |
| float | A floating point number | 7.68, -23.45 |
| string | A sequence of characters | "Hello", "abc123@#$" |
| unicode | A sequence of Unicode characters | "Hello", "abc123@#$" |
| binary | A sequence of binary (non-Unicode) characters | "Hello", "abc123@#$" |
| boolean | Either true or false | true, false |

- **PHP's compound data types:**

Compound data can contain multiple values. PHP has 2 compound data types:

- Array - Can hold multiple values indexed by numbers or strings.
- Object - Can hold multiple values (properties), and can also contain methods (functions) forworking on properties.

An array or object can contain multiple values, all accessed via one variable name. What's more, thosevalues can themselves be other arrays or objects. This allows you to create quite complex collections of data.

## Operators:

Operators are used to operate on values.

**Arithmetic Operators:**

| Operator | Description | Example | Result |
|---|---|---|---|
| + | Addition | x=2<br>x+2 | 4 |
| - | Subtraction | x=2<br>5-x | 3 |
| * | Multiplication | x=4<br>x*5 | 20 |
| / | Division | 15/5<br>5/2 | 3<br>2.5 |
| % | Modulus | 5%2<br>10%8<br>10%2 | 1<br>2<br>0 |
| ++ | Increment | x=5<br>x++ | x=6 |
| -- | Decrement | x=5<br>x-- | x=4 |

**Assignment Operators:**

| Operator | Example | Is The Same As |
|---|---|---|
| = | x=y | x=y |
| += | x+=y | x=x+y |
| -= | x-=y | x=x-y |
| *= | x*=y | x=x*y |
| /= | x/=y | x=x/y |
| .= | x.=y | x=x.y |
| %= | x%=y | x=x%y |

**Comparison Operators:**

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

| Operator | Description | Example |
|---|---|---|
| && | and | x=6y=3<br>(x < 10 && y > 1) returnstrue |
| \|\| | or | x=6y=3<br>(x==5 \|\| y==5) returns false |
| ! | not | x=6y=3<br>!(x==y) returns true |

**Operator Precedence:**

| |
|---|
| * / %<br><br>Highest Precedence |
| + - . |
| < <= > >= |
| = = = != = |
| && |

| |
|---|
| \| \| |
| And |
| XOR |
| OR<br><br>Lowest Precedence |

## Statements:

**1. PHP Control Statements: if, elseif, else**

```
if ( )

{}
```

```
elseif ( )

{}

else

{}
```

**2. Switch:**

```
switch ( )

{

     case condition1

                    break;case condition2
        break;

     }
     while:
```

Syntax:

```
while (condition)

{

}
```

**3. do:**

```
do

{

} while (condition)
```

**4. for:**

Example use of the for statement:

```
<?php
```

```
for ( $n = 1; $n < 10; $n++)

{

echo "$n<BR>";

}          ?>
```

**foreach:**

Versions of PHP prior to version 4 do not support the foreach statement. The following code should list thecontents of the array.

```
<?php

$tree = array("trunk",

"branches",

"leaves");foreach

($tree as $part)

{

echo "Tree part: $part ";

}

?>
```

**break:**

Is used to end the execution of a for, switch, or while statement

**continue:**

This statement is used to skip the rest of the current loop.

```
<?php

for ( $n = 1; $n < 10; $n++)
```

```
{
echo "$n<BR>";
if ($n == 5) continue;
echo "This statement is skipped when $n = 5.<BR>";
}
?>
```

The continue statement can be given an optional parameter such as "continue 2" telling it how many levelsof loops to skip.

## Functions: Definition

A function is a block of code that performs a specific task. It has a name and it is reusable. To keepthe script from being executed when the page loads, you can put it into a function. A function will be executed by a call to the function. You may call a function from anywhere within a page.

**User-defined functions:**

A function may be defined using syntax such as the following:PHP function guidelines:

- Give the function a name that reflects what the function does
- The function name can start with a letter or underscore (not a number)

**Example**

A simple function that writes my name when it is called:

```
<html>
<body>
<?php
function writeName()
{
echo "Kai Jim Refsnes";
}
echo "My name is ";writeName();
```

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

**Output:**

```
My name is Kai Jim Refsnes
```

**PHP Functions - Adding parameters**

To add more functionality to a function, we can add parameters. A parameter is just like avariable.Parameters are specified after the function name, inside the parentheses.

**Example 1**

The following example will write different first names, but equal last name:

```
<html>
<body>
<?php
function writeName($fname)
{
echo $fname . " Refsnes.<br />";

}
echo "My name is "; writeName("Kai Jim"); echo "My sister's name is ";writeName("Hege");
echo "My brother's name is ";writeName("Stale");
?>
</body>
</html>Output:
```

```
My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes. My brother's name is Stale Refsnes.
```

**PHP Functions - Return values:**

To let a function return a value, use the return statement.

**Example**

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

```
<?php
function add($x,$y)
{
$total=$x+$y;return $total;
}
echo "1 + 16 = " . add(1,16);
?>
</body>
</html>Output:
```

```
1 + 16 = 17
```

**Functions within functions:**

```
<?php function foo()
{
function bar()
{
echo "I don't exist until foo() is called.\n";
}
}
/* We can't call bar() yetsince it doesn't exist. */
foo();

/* Now we can call bar(),foo()'s processesing hasmade it accessible. */
bar();
?>
```

All functions and classes in PHP have the global scope - they can be called outside a function even if theywere defined inside and vice versa.

PHP supports passing arguments by value (the default), passing by reference, and default argument values. Variable-length argument lists are supported only in PHP 4 and later; see Variable-length argument lists and the function references for **func_num_args()**, **func_get_arg()**, and **func_get_args()** for more information.

**func_num_args:**

func_num_args -- Returns the number of arguments passed to the function

**Description**
int **func_num_args** ( void )

Gets the number of arguments passed to the function.

This function may be used in conjunction with **func_get_arg()** and **func_get_args()** to allow user-definedfunctions to accept variable-length argument lists.

**Return Values:**
Returns the number of arguments passed into the current user-defined function.

**Errors/Exceptions:**
Generates a warning if called from outside of a user-defined function.

**Examples Example**

```
<?php function foo()
{
$numargs = func_num_args();
echo "Number of arguments: $numargs\n";
}


foo(1, 2, 3);      // Prints 'Number of arguments: 3'
?>
```

**func_get_arg:**

func_get_arg -- Return an item from the argument list

**Description**

mixed **func_get_arg** ( int arg_num )

Gets the specified argument from a user-defined function's argument list.

This function may be used in conjunction with **func_get_args()** and **func_num_args()** to allow user-definedfunctions to accept variable-length argument lists.

**Parameters:**

arg_num

The argument offset. Function arguments are counted starting from zero.

**Return Values**

Returns the specified argument, or **`FALSE`** on error.

**Errors/Exceptions**

Generates a warning if called from outside of a user-defined function, or if `arg_num` is greater than the numberof arguments actually passed.

**Examples**

**Example 1. func_get_arg() example**

```
<?php
function foo()
{
$numargs = func_num_args();
echo "Number of arguments: $numargs<br />\n";if ($numargs >= 2) {
echo "Second argument is: " . func_get_arg(1) . "<br />\n";
}
}
foo (1, 2, 3);
?>
```

**func_num_args:**

func_num_args -- Returns the number of arguments passed to the function

**Description**

int **func_num_args** ( void )

Gets the number of arguments passed to the function.

This function may be used in conjunction with **func_get_arg()** and **func_get_args()** to allow user-definedfunctions to accept variable-length argument lists.

**Return Values**

Returns the number of arguments passed into the current user-defined function.

3.**7.8 Errors/Exceptions**

Generates a warning if called from outside of a user-defined function.

**Examples**

**Example 1. func_num_args() example**

```
<?php
function foo()
{
$numargs = func_num_args();
echo "Number of arguments: $numargs\n";
}

foo(1, 2, 3);           // Prints 'Number of arguments: 3'
?>
```

**Variable functions**

PHP supports the concept of variable functions. This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variableevaluates to, and will attempt to execute it.

**Example for Variable function example**

```
<?php
function foo() {
echo "In foo()<br />\n";
}
function bar($arg = '')
{
echo "In bar(); argument was '$arg'.<br />\n";
}
// This is a wrapper function around echofunction echoit($string)
{
echo $string;
}
$func = 'foo';
$func();                    // This calls foo()
$func = 'bar';
$func('test');  // This calls bar()
$func = 'echoit';
$func('test');  // This calls echoit()?>
```

## Arrays:

### What is an Array?

A variable is a storage area holding a number or text. The problem is, a variable will hold only one value.

PHP array are useful to store multiple data using a single variable. Using index of the PHP array we can accessthe stored variables. Indexes to an array element can either a number or a string. An array is a list variables.

Each item in an array is commonly known as element of the array and can be accessed directly via its index.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1="Saab";
$cars2="Volvo";
$cars3="BMW";
```

An array can hold all your variable values under a single name. And you can access the values by referring tothe array name.

Each element in the array has its own index so that it can be easily accessed.

**Types :**
In PHP, there are three kind of arrays:

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

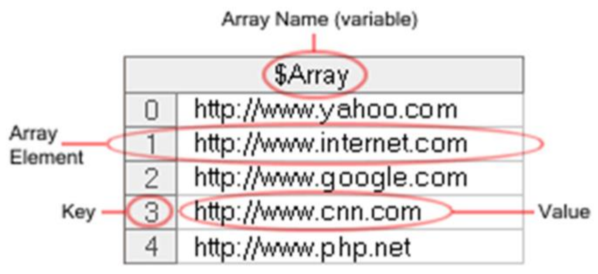


Figure 1.

**Numeric Arrays**

A numeric array stores each array element with a numeric index.There are two methods to create a numeric array.

1. In the following example the index are automatically assigned (the index starts at 0):

```
$cars=array("Saab","Volvo","BMW","Toyota");
```

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

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";

$cars[3]="Toyota";
```

**Example**

In the following example you access the variable values by referring to the array name and index:

```
<?php
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
?>
```

The code above will output:

```
Saab and Volvo are Swedish cars.
```

**Associative Arrays:**

An associative array, each ID key is associated with a value.

When storing data about specific named values, a numerical array is not

always the best way to do it.With associative arrays we can use the values as

keys and assign values to them.

**Example 1**

In this example we use an array to assign ages to the different persons:

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

**Example 2**

This example is the same as example 1, but shows a different way of creating the array:

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
```

```
$ages['Joe'] = "34";
```

The ID keys can be used in a script:

```
<?php
$ages['Peter'] = "32";

$ages['Quagmire'] = "30";
$ages['Joe'] = "34";

echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

The code above will output:

```
Peter is 32 years old.
```

**Multidimensional Arrays**

In a multidimensional 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.

**Example**

In this example we create a multidimensional array, with automatically assigned ID keys:

```
$families = array(
 "Griffin"=>array(
 "Peter",
 "Lois", "Megan"
 ),
 "Quagmire"=>array(
 "Glenn"
 ),
 "Brown"=>array(
 "Cleveland","Loretta", "Junior"
 )
 );
```

The array above would look like this if written to the output:

```
 Array(

[Griffin] => Array(
  [0] => Peter
  [1] => Lois
  [2] => Megan
  )
[Quagmire] => Array(
  [0] => Glenn
  )
[Brown] => Array(
  [0] => Cleveland
  [1] => Loretta
  [2] => Junior
  )
  )
```

## OOP:

Object oriented programming is a concept that was created because of the need to overcome the problemsthat were found with using structured programming techniques. While structured programming uses an approach which is top down, OOP uses an approach which is bottom up.

**Main Principal of OOPS are**

1. Encapsulation : binding the data and function in a unit called class is called a data encapsulation. thisallow the user to hide the information for outside world and doesn't allow the other user to change or modify the internal values of class.
2. Polymorphism: polymorphism is another strong feature of OOPS. it means one term in many forms.
3. Inheritance: another strong property of OOPS. this feature offers to derive a new class from an existingone and acquire all the feature of the existing class the new class which get the feature from existing class called derived class and other class is called base class.

**Creating Classes:**

Basic class definitions begin with the keyword *class*, followed by a class name, followed by a pair of curlybraces which enclose the definitions of the properties and methods belonging to the class.

The class name can be any valid label which is a not a PHP reserved word. A valid class name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, itwould be expressed thus: *[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]**.
A class may contain its own constants, variables (called "properties"), and functions (called "methods").

**Example for Simple Class Definition:**

```
<?php
class SimpleClass
{
// property declaration
public $var = 'a default value';
// method declaration
public function displayVar() {echo $this->var;
}
}
?>
```

**Creating Objects:**

It's much easier to create objects and use them than it is to define object classes, so

before we discuss how to define classes. To create an object of a given class, use the new keyword:

$*object* = new *Class*;

Assuming that a Person class has been defined, here's how to create a Person object:

```
$rasmus = new Person;
```

Do not quote the class name, or you'll get a compilation error:

```
$rasmus = new 'Person'; // does not work
```

Some classes permit you to pass arguments to the new call. The class's documentationshould

say whether it accepts arguments. If it does, you'll create objects like this:

```
$object = new Person('Fred', 35);
```

The class name does not have to be hardcoded into your program. You can supply the class name through a variable:

```
$class = 'Person';

$object = new $class;

// is equivalent to

$object = new Person;
```

Specifying a class that doesn't exist causes a runtime error.

Variables containing object references are just normal variables—theycan be used inthe same ways as other variables. Of particular note is that variable variables work with objects, as shown here:

```
$account = new Account;
$object = 'account'
${$object}->init(50000, 1.10); // same as $account->init
```

## String Manipulation and regular expression:

In PHP, and every other flavor of Web programming, a string is a variable contained between quotes with aliteral value. For example, $number = “2” is a string variable, whereas $number = 2 is not. The first value isseen as text only, the latter as a numeric value. Simply put, a string is a text variable.

1. **ucwords** - this is used to convert a phrase or sentence into a title format.

   ```
   <?php
   $string =”string manipulation in php”;echo ucwords($string);
   ?>
   ```

2. **strtoupper** – this is used to capitalize every letter in a string.
   ```
   <?php
   $string =”string manipulation in php”;echo strtoupper($string);
   ?>
   ```
3. **strtolower** – this is used to bring all letters in a string to lowercase.

   ```
   <?php
   $string =”String Manipulation in PHP”;echo strtolower($string);
   ?>
   ```
4. 
   **str_replace** – this function is used to replace a certain substring in a string

   ```
   <?php
   $string = “This tutorial is very useful for newbies”;
   $newstring = str_replace(”useful”, “difficult”, $string );echo $newstring;
   ?>
   ```

   The str_replace() function takes 3 parameters. The first parameter is the piece of text you want to replace. Thesecond is the text that will replace the original. The third parameter tells PHP what value to do the find and replace on.

5. **strpos** – find position of first occurrence of a string

   Syntax: int **strpos** ( string $haystack , mixed $needle [, int $offset = 0 ] )

   ```
   <?php
   $string = “Hello World”;
   ```

```
echo “The word World is at position number: “.strpos($string,”World”);
?>
```

6. **substr** – Return part of a string

   Syntax: string **substr** ( string $string , int $start [, int $length ] )

   ```
   <?php
   $string = “Hello World”;
   $substring = substr($string,6);echo $substring;
   ?>
   ```

   The example above will display the word “World”.

7. **strlwr:**

   **Prototype: char *strlwr(char *)**

   This function converts the given string to lowercase and returns the same.

8. **strupr:**

   **Prototype: char *strupr(char *)**

   This function converts the given string to UPPERCASE and returns it.

9. **strncat:**

   **Prototype: strncat(char *str1, const char *str2, int n)**

   It appends first ‘n’ characters of str2 to the end of str1.

10. **strncpy:**

    **Prototype: int strncmp(char *str1, const char *str2, int n)**

    This function compares first ‘n’ characters of str1 with str2, it returns 0 if compare was successful.

11. **stricmp:**

    **Prototype: int stricmp(const char *str1, const char *str2)**

    It compares two strings str1 and str2 without regard to case and returns 0 on being successful.

12. **strnicmp:**

    **Prototype: int strnicmp(const char *str1, const char *str2, int n)**
    This function compares first ‘n’ characters of str1 with str2 without regard to case.

13. **strrev:**

**Prototype: char *strrev(char *)**

This function reverses the given string and returns it.

14. **stcstr:**

**Prototype: char *strstr(const char *str1, const char *str2)**

This function returns the pointer to where the first occurrence of string str2 is found inside str1.

## File Handling and Data Storage:

PHP file handling functions are used in different conditions in any script development. You can usethem to create a simple file upload or image upload or gallery script for storing images and many more. These built in functions gives flexibility in developing complex scripts.

**File Read:**

We can open a file or a URL to read by using fopen() function of PHP. While opening we can givemode of file open ( read, write.. etc ). By using fopen we can read any external url also.

We will be using fread() function to read the content by using a file pointer. **fread()** reads up to lengthbytes from the file pointer referenced by fd. Reading stops when length bytes have been read or EOF isreached, whichever comes first.

Example Program:

```
<?
$filename = "delete.htm"; // This is at root of the file using this script.
$fd = fopen ($filename, "r"); // opening the file in read mode
$contents = fread ($fd, filesize($filename)); // reading the content of the filefclose ($fd);   //
Closing the file pointer echo $contents// printing the file content of the file
?>
```

**File Write**

We can write to a file by using fwrite() function PHP. We have to open the file in write mode and ifwrite permission is there then only we can open it in write mode. If the file does not exist then one new file will be created.

Exmple:

```
<?
$body_content="This is my content"; // Store some text to enter inside the file
$file_name="test_file.txt";                // file name
$fp = fopen ($file_name, "w");
// Open the file in write mode, if file does not exist then it will be created.fwrite
($fp,$body_content);                    // entering data to the file
fclose ($fp);                           // closing the file pointer chmod($file_name,0777);
                                        // changing the file permission.
?>
```

**PHP File  delete using unlink function**

We can delete files by giving its URL or path in PHP by using unlink command. This command will work only if write permission is given to the folder or file. Without this the delete command will fail. Here is thecommand to delete the file.

**unlink($path);**

**Here $path is the relative path of the file calculated from the script execution. Here is an example ofdeleting file by using relative path**

**$path="images/all11.css"; if(unlink($path)) echo "Deleted file ";**

We have used if condition to check whether the file delete command is successful or not. But the commandbelow will not work.

**$path="http://domainname/file/red.jpg";if(unlink($path)) echo "Deleted file ";**

The warning message will say unlink() [function.unlink]: HTTP does not allow unlinking. We can suppress the warning message by adding a @ symbol before the unlink command .

Based on the success of the file delete command we can further execute our code by using if else code block.

**$path="test.html";**
**if(@unlink($path)) {echo "Deleted file "; }else{echo "File can't be deleted";}**

**file_exists function**

In our script before executing a program we need to check if file exists or not. This file check method by usingfile_exists function check the presence of file from relative path we specify and then returns TRUE of False based on the presence of the file.

We can best use file_exists function along with an if condition.Here is a sample code.

**$add="images/119.jpg";if(file_exists($add)){ echo "Yes file is there ";**
**}else{echo "Sorry no file at $add ";}**

**Deleting all files of a directory**

Now let us try to develop a function and to this function we will post directory name as parameter and thefunction will use unlink command to remove files by looping through all the files of the directory.

Here is the code to this.

```
function EmptyDir($dir) {
$handle=opendir($dir);

while (($file = readdir($handle))!==false) {echo "$file <br>";
@unlink($dir.'/'.$file);
}
closedir($handle);
}
EmptyDir('images');
```

**Example:**

**Create an Upload-File Form:**

To allow users to upload files from a form can be very useful.Look at the following

HTML form for uploading files:

```
<html>
<body>
<form action="upload_file.php" method="post"enctype="multipart/form-data">
<label for="file">Filename:</label>
<input type="file" name="file" id="file" />
<br />
<input type="submit" name="submit" value="Submit" />
</form>
</body>
</html>
```

Notice the following about the HTML form above:

- The enctype attribute of the <form> tag specifies which content-type to use when submitting the form."multipart/form-data" is used when a form requires binary data, like the contents of a file, to be uploaded
- The type="file" attribute of the <input> tag specifies that the input should be processed as a file. Forexample, when viewed in a browser, there will be a browse-button next to the input field

**Note:** Allowing users to upload files is a big security risk. Only permit trusted users to perform file uploads.

**PHP File Upload:**

**Create The Upload Script**

The "upload_file.php" file contains the code for uploading a file:

```
<?php
if ($_FILES["file"]["error"] > 0)
{
echo "Error: " . $_FILES["file"]["error"] . "<br />";
}
else
{
echo "Upload: " . $_FILES["file"]["name"] . "<br />";
echo "Type: " . $_FILES["file"]["type"] . "<br />";
echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";echo "Stored in: " .
$_FILES["file"]["tmp_name"];
}
?>
```

By using the global PHP $_FILES array you can upload files from a client computer to the remote server.The first parameter is the form's input name and the second index can be either "name", "type", "size",

"tmp_name" or "error". Like this:

- $_FILES["file"]["name"] - the name of the uploaded file
- $_FILES["file"]["type"] - the type of the uploaded file
- $_FILES["file"]["size"] - the size in bytes of the uploaded file
- $_FILES["file"]["tmp_name"] - the name of the temporary copy of the file stored on the server
- $_FILES["file"]["error"] - the error code resulting from the file upload

This is a very simple way of uploading files. For security reasons, you should add restrictions on what the useris allowed to upload.

**Saving the Uploaded File:**

The examples above create a temporary copy of the uploaded files in the PHP temp folder on the server.

The temporary copied files disappears when the script ends. To store the uploaded file we need to copy it to adifferent location:

```
<?php
if ((($_FILES["file"]["type"] == "image/gif")
|| ($_FILES["file"]["type"] == "image/jpeg")
|| ($_FILES["file"]["type"] == "image/pjpeg"))
&& ($_FILES["file"]["size"] < 20000))
{
if ($_FILES["file"]["error"] > 0)
{
echo "Return Code: " . $_FILES["file"]["error"] . "<br />";
}
else
{
echo "Upload: " . $_FILES["file"]["name"] . "<br />";
echo "Type: " . $_FILES["file"]["type"] . "<br />";
echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
echo "Temp file: " . $_FILES["file"]["tmp_name"] . "<br />";

if (file_exists("upload/" . $_FILES["file"]["name"]))
{
echo $_FILES["file"]["name"] . " already exists. ";
}
else
{
move_uploaded_file($_FILES["file"]["tmp_name"],"upload/" . $_FILES["file"]["name"]);
echo "Stored in: " . "upload/" . $_FILES["file"]["name"];
}
     }

   }
  else
   {
   echo "Invalid file";
   }
  ?>
```

The script above checks if the file already exists, if it does not, it copies the file to the specified folder.

**PHP directory listing:**

Many times we have to display the list of files in a directory. We can keep the script in any location and byusing the file path we can display the files inside the directory.

```
// open the current directory by opendir
$handle=opendir(".");

while (($file =
readdir($handle))
!==false) {echo
"$file <br>";
}

closedir($handle);
```

# UNIT V

# PHP and SQL database

PHP and SQL database – PHP and LDAP – PHP Connectivity – Sending and receiving E-mails – Debugging and errorhandling – Security – Templates.

## PHP and LDAP:

## PHP:

### Create a Connection to a MySQL Database

Before you can access data in a database, you must create a connection to the database.In PHP, this is done with the mysql_connect() function.

**Syntax**

mysql_connect(servername,username,password);

| **Parameter** | **Description** |
|---|---|
| Servername | Optional. Specifies the server to connect to. Default value is "localhost:3306" |
| Username | Optional. Specifies the username to log in with. Default value is the name of the userthat owns the server process |
| Password | Optional. Specifies the password to log in with. Default is "" |

**Note:** There are more available parameters, but the ones listed above are the most important.

**Example**

In the following example we store the connection in a variable ($con) for later use in the script. The "die" partwill be executed if the connection fails:

```
<?php
$con = mysql_connect("localhost","peter","abc123");if (!$con)
{
die('Could not connect: ' . mysql_error());
}
// some code
```

```
?>
```

**Closing a Connection:**

The connection will be closed automatically when the script ends. To close the connection before, use themysql_close() function:

```
<?php
$con = mysql_connect("localhost","peter","abc123");if (!$con)
{
die('Could not connect: ' . mysql_error());
}

// some code mysql_close($con);
?>
```

**Create a Database:**

The CREATE DATABASE statement is used to create a database in MySQL.

**Syntax**

```
CREATE DATABASE database_name
```

To get PHP to execute the statement above we must use the mysql_query() function. This function is used tosend a query or command to a MySQL connection.

**Example**

The following example creates a database called "my_db":

```
<?php
$con = mysql_connect("localhost","peter","abc123");if (!$con)
{
die('Could not connect: ' . mysql_error());
}

if (mysql_query("CREATE DATABASE my_db",$con))
{
echo "Database created";
}
else
{
echo "Error creating database: " . mysql_error();
}
mysql_close($con);
```

```
?>
```

**Create a Table:**

The CREATE TABLE statement is used to create a table in MySQL.

**Syntax**

```
CREATE TABLE table_name(
column_name1 data_type,column_name2 data_type,column_name3 data_type,
....
)
```

We must add the CREATE TABLE statement to the mysql_query() function to execute the command.

**Example**

The following example creates a table named "Persons", with three columns. The column names will be"FirstName", "LastName" and "Age":

```
<?php
$con = mysql_connect("localhost","peter","abc123");if (!$con)
{
die('Could not connect: ' . mysql_error());
}

// Create database
if (mysql_query("CREATE DATABASE my_db",$con))
{
echo "Database created";
}
else

{
echo "Error creating database: " . mysql_error();
}

// Create table mysql_select_db("my_db", $con);
$sql = "CREATE TABLE Persons(
FirstName varchar(15),LastName varchar(15),Age int
)";

// Execute query mysql_query($sql,$con);

mysql_close($con);
?>
```

**Insert Data into a Database Table:**

The INSERT INTO statement is used to add new records to a database table.

**Syntax**

It is possible to write the INSERT INTO statement in two forms.

The first form doesn't specify the column names where the data will be inserted, only

their values:INSERT INTO table_name
VALUES (value1, value2, value3,...)

The second form specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

To get PHP to execute the statements above we must use the mysql_query() function. This function is used tosend a query or command to a MySQL connection.

**Example**

In the previous chapter we created a table named "Persons", with three columns; "Firstname", "Lastname" and "Age". We will use the same table in this example. The following example adds two new records to the "Persons" table:

```
<?php
$con = mysql_connect("localhost","peter","abc123");if (!$con)
{
die('Could not connect: ' . mysql_error());
}
mysql_select_db("my_db", $con);

mysql_query("INSERT INTO Persons (FirstName, LastName, Age)VALUES ('Peter',
'Griffin', '35')");

mysql_query("INSERT INTO Persons (FirstName, LastName, Age)VALUES ('Glenn',
'Quagmire', '33')");

mysql_close($con);
?>
```

**Select Data From a Database Table**

The SELECT statement is used to select data from a database.

**Syntax**

SELECT column_name(s)FROM table_name

To get PHP to execute the statement above we must use the mysql_query() function. This function is used tosend a query or command to a MySQL connection.

**Example**

The following example selects all the data stored in the "Persons" table (The * character selects all the data inthe table):

```
<?php
$con = mysql_connect("localhost","peter","abc123");if (!$con)
{
die('Could not connect: ' . mysql_error());
}

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM Persons");

while($row = mysql_fetch_array($result))
{
echo $row['FirstName'] . " " . $row['LastName'];echo "<br />";
}

mysql_close($con);
?>
```

## <u>LDAP:</u>

**Introduction**

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.

In case you're not familiar with LDAP, it is a protocol designed to allow quick, efficient searches of directory services. Built around Internet technologies, LDAP makes it

possible to easily update and query directory services over standard TCP/IP connections, and includes a host of powerful features, including security, access control, data replication and support for Unicode.
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 = AccountscommonName = John Smith
LDAP support in PHP is not enabled by default. You will need to use the `--with-ldap[=DIR]`
configuration option when compiling PHP to enable LDAP support. DIR is the LDAP base install directory.To enable SASL support, be sure `--with-ldap-sasl[=DIR]` is used, and that `sasl.h` exists on the system.

**Predefined Constants:**

The constants below are defined by this extension, and will only be available when the extension has eitherbeen compiled into PHP or dynamically loaded at runtime.

**LDAP_DEREF_NEVER (integer)**
**LDAP_DEREF_SEARCHING (integer)**
**LDAP_DEREF_FINDING (integer)**
**LDAP_DEREF_ALWAYS (integer)**
**LDAP_OPT_DEREF (integer)**

**Using the PHP LDAP calls:**

Before you can use the LDAP calls you will need to know ..

- The name or address of the directory server you will use
- The "base dn" of the server (the part of the world directory that is held on this server, which could be"o=My Company,c=US")
- Whether you need a password to access the server (many servers will provide read access for an"anonymous bind" but require a password for anything else)

The typical sequence of LDAP calls you will make in an application will follow this pattern:

ldap_connect()    // establish connection to server

|

ldap_bind()    // anonymous or authenticated "login"

|

do something like search or update the directoryand display the results

|

ldap_close()    // "logout"

**LDAP Functions:**

- ldap_8859_to_t61 — Translate 8859 characters to t61 characters
- ldap_add — Add entries to LDAP directory
- ldap_bind — Bind to LDAP directory
- ldap_close — Alias of ldap_unbind
- ldap_compare — Compare value of attribute found in entry specified with DN
- ldap_connect — Connect to an LDAP server

## PHP Connectivity:

**Example Program**

```
<?php
// basic sequence with LDAP is connect, bind, search, interpret
search
// result, close connection

echo "<h3>LDAP query test</h3>";echo "Connecting ...";
$ds=ldap_connect("localhost"); // must be a valid LDAP server!echo
"connect result is " . $ds . "<br />";

if ($ds) {
echo "Binding ...";
$r=ldap_bind($ds);           // this is an "anonymous" bind,
typically
                       // read-only accessecho "Bind result is " .
$r . "<br />";

echo "Searching for (sn=S*) ...";
// Search surname entry
$sr=ldap_search($ds, "o=My Company, c=US", "sn=S*");echo "Search
result is " . $sr . "<br />";
echo "Number of entires returned is " . ldap_count_entries($ds, $sr)
. "<br
/>";

echo "Getting entries ...<p>";
$info = ldap_get_entries($ds, $sr);
echo "Data for " . $info["count"] . " items returned:<p>";

for ($i=0; $i<$info["count"]; $i++) {
echo "dn is: " . $info[$i]["dn"] . "<br />";
echo "first cn entry is: " . $info[$i]["cn"][0] . "<br />";
echo "first email entry is: " . $info[$i]["mail"][0] . "<br /><hr
/>";
}

echo "Closing connection";ldap_close($ds);

} else {
echo "<h4>Unable to connect to LDAP server</h4>";

}
?>
```

## Sending and Receiving E-mails:

**Introduction:**
Email is the most popular Internet service today. A plenty of emails are sent and delivered each day.

### Description

bool **mail** ( string to, string subject, string message [, string additional_headers [, string additional_parameters]] )

Sends an email.

### Parameters

to

Receiver, or receivers of the mail.

The formatting of this string must comply with RFC 2822. Some examples are:

user@example.com

user@example.com, anotheruser@example.comUser <user@example.com>

User <user@example.com>, Another User <anotheruser@example.com>

subject

Subject of the email to be sent.

**Return Values**

Returns **TRUE** if the mail was successfully accepted for delivery, **FALSE** otherwise.

It is important to note that just because the mail was accepted for delivery, it does NOT mean the mail willactually reach the intended destination.

**PHP Code(Server side): Example for Sending mail.**

Using **mail()** to send a simple email:

```
<?php
// The message

$message = "Line 1\nLine 2\nLine 3";
// In case any of our lines are larger than 70 characters, we should use
wordwrap()
$message = wordwrap($message, 70);
// Send
mail('caffinated@example.com', 'My Subject', $message);
?>
```

**Example for Sending mail with extra headers.**

The addition of basic headers, telling the MUA the From and Reply-To addresses:

```
<?php
$to         = 'nobody@example.com';
$subject = 'the subject';
$message = 'hello';
$headers = 'From: webmaster@example.com' . "\r\n" .'Reply-To:
webmaster@example.com' . "\r\n" .
    'X-Mailer: PHP/' . phpversion(); mail($to, $subject, $message,
$headers);
?>
```

**HTML code(Client Side):**

**Example for Sending HTML email**

It is also possible to send HTML email with **mail()**.

```
<?php
// multiple recipients
$to  = 'aidan@example.com' . ', '; // note the comma
$to .= 'wez@example.com';
```

```
// subject
$subject = 'Birthday Reminders for August';
// message
$message = '
<html>
<head>
<title>Birthday Reminders for August</title>
</head>
<body>
<p>Here are the birthdays upcoming in August!</p>
<table>
<tr>
<th>Person</th><th>Day</th><th>Month</th><th>Year</th>
</tr>
<tr>
<td>Joe</td><td>3rd</td><td>August</td><td>1970</td>
</tr>
<tr>
<td>Sally</td><td>17th</td><td>August</td><td>1973</td>
</tr>
</table>
</body>
</html>
// To send HTML mail, the Content-type header must be set
$headers  = 'MIME-Version: 1.0' . "\r\n";
$headers .= 'Content-type: text/html; charset=iso-8859-1' . "\r\n";
// Additional headers
$headers .= 'To: Mary <mary@example.com>, Kelly <kelly@example.com>' . "\r\n";
$headers .= 'From: Birthday Reminder <birthday@example.com>' . "\r\n";
$headers .= 'Cc: birthdayarchive@example.com' . "\r\n";
$headers .= 'Bcc: birthdaycheck@example.com' . "\r\n";
// Mail it
mail($to, $subject, $message, $headers);
?>
```

## Debugging and error handling:

### Debugging:

PHP does not have an internal debugging facility. You can use one of the external debuggers though.

The ZEND IDE includes a debugger.

### Using the Debugger

The internal debugger in PHP 3 is useful for tracking down evasive bugs. The debugger works by connecting to a TCP port for every time PHP 3 starts up. All error messages from that request will be sent tothis TCP connection. This information is intended for "debugging server" that can run inside an IDE or programmable editor (such as Emacs).

How to set up the debugger:

1. Set up a TCP port for the debugger in the configuration file (debugger.port) and enable it (debugger.enabled).
2. Set up a TCP listener on that port somewhere (for example **socket -l -s 1400** on Unix systems).
3. In your code, run "debugger_on(host)", where host is the IP number or name of the host running theTCP listener.

Now, all warnings, notices etc. will show up on that listener socket, even if you turned them off with

**error_reporting()**.

### Debugger Protocol

The PHP 3 debugger protocol is line-based. Each line has a type, and several lines compose a message. Each message starts with a line of the type start and terminates with a line of the type end. PHP 3may send lines for different messages simultaneously.

A line has this format:

date time host(pid) type: message-dat

date

Date in ISO 8601 format (yyyy-mm-dd)

time

Time including microseconds: hh:mm:uuuuuuhost

DNS name or IP address of the host where the script error was generated.

pid

PID (process id) on host of the process with the PHP 3 script that generated this error.

type

Type of line. Tells the receiving program about what it should treat the following data as:

### 3.16.1.2 Debugger Error Types

| **Debugger** | **PHP 3 Internal** |
|---|---|
| warning | E_WARNING |
| error | E_ERROR |
| parse | E_PARSE |
| notice | E_NOTICE |
| core-error | E_CORE_ERROR |
| core-warning | E_CORE_WARNING |
| unknown | (any other) |

### 3.16.4 Configure options

## List of core configure optionsMisc options

--enable-debug

Compile with debugging symbols.

--with-layout=TYPE

Sets how installed files will be laid out. Type is one of PHP (default) or GNU.

--with-pear=DIR

Install PEAR in DIR (default PREFIX/lib/php).

--without-pear

Do not install PEAR.

--enable-sigchild

Enable PHP's own SIGCHLD handler.

**PHP options**

--enable-maintainer-mode

Enable make rules and dependencies not useful (and sometimes confusing) to the casual installer.

--with-config-file-path=PATH

Sets the path in which to look for `php.ini`, defaults to PREFIX/lib.

--enable-safe-mode

Enable safe mode by default.

--with-exec-dir[=DIR]

Only allow executables in DIR when in safe mode defaults to /usr/local/php/bin.

**SAPI options**

The following list contains the available SAPI&s (`Server Application Programming Interface`) forPHP.

--with-aolserver=DIR

Specify path to the installed AOLserver.

--with-apxs[=FILE]

Build shared Apache module. FILE is the optional pathname to the Apache apxs tool; defaults to apxs. Make sure you specify the version of apxs that is actually installed on your system and NOTthe one that is in the apache source tarball.

--with-apache[=DIR]

Build a static Apache module. DIR is the top-level Apache build directory, defaults to `/usr/local/apache`.

--with-mod_charset

Enable transfer tables for mod_charset (Russian Apache).
--with-apxs2[=FILE]

Build shared Apache 2.0 module. FILE is the optional pathname to the Apache apxs tool; defaults toapxs.

## Error Handling:

The default error handling in PHP is very simple. An error message with filename, line number and amessage describing the error is sent to the browser.

### PHP Error Handling:

When creating scripts and web applications, error handling is an important part. If your code lackserror checking code, your program may look very unprofessional and you may be open to security risks.

This tutorial contains some of the most common error checking methods in PHP.We will

show different error handling methods:

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

**Basic Error Handling: Using the die() function:**

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

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

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

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

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

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

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

```
File not found
```

The code above is more efficient than the earlier code, because it uses a simple error handling mechanism tostop the script after the error.

However, simply stopping the script is not always the right way to go. Let's take a look at alternative PHPfunctions for handling errors.

**Creating a Custom Error Handler:**

Creating a custom error handler is quite simple. We simply create a special function that can be called whenan error occurs in PHP.

This function must be able to handle a minimum of two parameters (error level and error message) but canaccept 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)
```

| **Parameter** | **Description** |
|---|---|
| error_level | Required. Specifies the error report level for the user-defined error. Must be a value number.See table below for possible error report levels |
| error_message | Required. Specifies the error message for the user-defined error |

| | |
|---|---|
| error_file | Optional. Specifies the filename in which the error occurred |
| error_line | Optional. Specifies the line number in which the error occurred |
| error_context | Optional. Specifies an array containing every variable, and their values, in use when the erroroccurred |

## Error Report levels:

These error report levels are the different types of error the user-defined error handler can be used for:

| Value | Constant | Description |
|---|---|---|
| 2 | E_WARNING | Non-fatal run-time errors. Execution of the script is not halted |
| 8 | E_NOTICE | Run-time notices. The script found something that might be an error, butcould also happen when running a script normally |
| 256 | E_USER_ERROR | Fatal user-generated error. This is like an E_ERROR set by the programmerusing the PHP function trigger_error() |
| 512 | E_USER_WARNING | Non-fatal user-generated warning. This is like an E_WARNING set by theprogrammer using the PHP function trigger_error() |
| 1024 | E_USER_NOTICE | User-generated notice. This is like an E_NOTICE set by the programmerusing the PHP function trigger_error() |
| 4096 | E_RECOVERABLE_ERROR | Catchable fatal error. This is like an E_ERROR but can be caught by a userdefined handle (see also set_error_handler()) |
| 8191 | E_ALL | All errors and warnings, except level E_STRICT (E_STRICT will be part ofE_ALL as of PHP 6.0) |

Now lets create a function to handle errors:function customError($errno, $errstr)

{

```
echo "<b>Error:</b> [$errno] $errstr<br />";echo "Ending Script";
die();
}
```

The code above is a simple error handling function. When it is triggered, it gets the error level and an errormessage. It then outputs the error level and message and terminates the script.

Now that we have created an error handling function we need to decide when it should be triggered.

**Set Error Handler:**

The default error handler for PHP is the built in error handler. We are going to make the function above thedefault error handler for the duration of the script.

It is possible to change the error handler to apply for only some errors, that way the script can handle different errors in different ways. However, in this example we are going to use our custom error handler forall errors:

```
set_error_handler("customError");
```

Since we want our custom function to handle all errors, the set_error_handler() only needed one parameter, asecond parameter could be added to specify an error level.

**Example**

Testing the error handler by trying to output variable that does not exist:

```
<?php
//error handler function
function customError($errno, $errstr)
{
echo "<b>Error:</b> [$errno] $errstr";
}
//set error handler set_error_handler("customError");
//trigger errorecho($test);
?>
```

The output of the code above should be something like this:

**Error:** [8] Undefined variable: test

**Trigger an Error:**

In a script where users can input data it is useful to trigger errors when an illegal input occurs. In PHP, this isdone by the trigger_error() function.

**Example**

In this example an error occurs if the "test" variable is bigger than "1":

```
<?php
$test=2;
if ($test>1)
{
trigger_error("Value must be 1 or below");
}
?>
```

The output of the code above should be something like this:

**Notice**: Value must be 1 or below in **C:\webfolder\test.php** on line **6**

An error can be triggered anywhere you wish in a script, and by adding a second parameter, you can specifywhat error level is triggered.

Possible error types:

- E_USER_ERROR - Fatal user-generated run-time error. Errors that can not be recovered from.Execution of the script is halted
- E_USER_WARNING - Non-fatal user-generated run-time warning. Execution of the script is nothalted
- E_USER_NOTICE - Default. User-generated run-time notice. The script found something that mightbe an error, but could also happen when running a script normally

**Example**

In this example an E_USER_WARNING occurs if the "test" variable is bigger than "1". If an E_USER_WARNING occurswe will use our custom error handler and end the script:

```
<?php
```

```
//error handler function
function customError($errno, $errstr)
{
echo "<b>Error:</b> [$errno] $errstr<br />";echo "Ending Script";
die();
}

//set error handler set_error_handler("customError",E_USER_WARNING);

//trigger error
$test=2;
if ($test>1)
{
trigger_error("Value must be 1 or below",E_USER_WARNING);
}
?>
```

The output of the code above should be something like this:

**Error:** [512] Value must be 1 or belowEnding Script

## Security:

PHP is a powerful language and the interpreter, whether included in a web server as a module or executed as a separate CGI binary, is able to access files, execute commands and open network connectionson the server. These properties make anything run on a web server insecure by default.

PHP is designed specifically to be a more secure language for writing CGI programs than Perl or C, and with correct selection of compile-time and runtime configuration options, and proper coding practices, it cangive you exactly the combination of freedom and security.

### Installed as CGI binary:

#### Possible attacks:

Using PHP as a CGI binary is an option for setups that for some reason do not wish to integrate PHP as a module into server software (like Apache), or will use PHP with different kinds of CGI wrappers to create safe chroot and setuid environments for scripts. Even if the PHP binary can be used as a standalone interpreter, PHP is designed to prevent the attacks this setup makes possible:

- Accessing system files: http://my.host/cgi-bin/php?/etc/passwd
- Accessing any web document on server: http://my.host/cgi-bin/php/secret/doc.html

**Case 1: only public files served**

If your server does not have any content that is not restricted by password or ip based access control, there is no need for these configuration options. If your web server does not allow you to do redirects, or the server does not have a way to communicate to the PHP binary that the request is a safely redirected request, you can specify the option **--enable-force-cgi-redirect** to the configure script.

**Case 2: using cgi.force_redirect**

This compile-time option prevents anyone from calling PHP directly with a URL like http://my.host/cgi-bin/php/secretdir/script.php. Instead, PHP will only parse in this mode if it has gonethrough a web server redirect rule.

**Case 3: setting doc_root or user_dir**

To include active content, like scripts and executables, in the web server document directories is sometimes considered an insecure practice. If, because of some configuration mistake, the scripts are not executed but displayed as regular HTML documents, this may result in leakage of intellectual property or security information like passwords. Therefore many sysadmins will prefer setting up another directory structure for scripts that are accessible only through the PHP CGI, and therefore always interpreted and not displayed as such.

**Case 4: PHP parser outside of web tree**

A very secure option is to put the PHP parser binary somewhere outside of the web tree of files. In
/usr/local/bin, for example.

```
#!/usr/local/bin/php
```

as the first line of any file containing PHP tags.

**Installed as an Apache module**

When PHP is used as an Apache module it inherits Apache's user permissions (typically those of the "nobody" user). This has several impacts on security and authorization. For example, if you are using PHP to access a database, unless that database has built-in access control, you will have to make the database accessible to the "nobody" user.

**File system Security**

PHP is subject to the security built into most server systems with respect to permissions on a file and directory basis. This allows you to control which files in the filesystem may be read.

Consider the following script, where a user indicates that they'd like to delete a file in their home directory. This assumes a situation where a PHP web interface is regularly used for file management, so theApache user is allowed to delete files in the user home directories.

**Example: A filesystem attack**

```
<?php
// removes a file from anywhere on the hard drive that
// the PHP user has access to. If PHP has root access:
$username = $_POST['user_submitted_name']; // "../etc"
$userfile = $_POST['user_submitted_filename']; // "passwd"
$homedir = "/home/$username"; // "/home/../etc" unlink("$homedir/$userfile"); //

"/home/../etc/passwd"


echo "The file has been deleted!";


?>
```

There are two important measures you should take to prevent these issues.

- Only allow limited permissions to the PHP web user binary.
- Check all variables which are submitted.

**Null bytes related issues:**

As PHP uses the underlying C functions for filesystem related operations, it may handle null bytes ina quite unexpected way. As null bytes denote the end of a string in C, strings containing them won't be considered entirely but rather only until a null byte occurs. The following example shows a vulnerable code that demonstrates this problem:

**Script vulnerable to null bytes**

```
<?php
$file = $_GET['file']; // "../../etc/passwd\0"
if (file_exists('/home/wwwrun/'.$file.'.php')) {
// file_exists will return true as the file /home/wwwrun/../../etc/passwd existsinclude
'/home/wwwrun/'.$file.'.php';
// the file /etc/passwd will be included
}
?>
```

**Database Security:**

Nowadays, databases are cardinal components of any web based application by enabling websites to provide varying dynamic content. Since very sensitive or secret information can be stored in a database, you should strongly consider protecting your databases.

**Designing Databases:**

The first step is always to create the database, unless you want to use one from a third party. When a database is created, it is assigned to an owner, who executed the creation statement. Usually, only the owner(or a superuser) can do anything with the objects in that database, and in order to allow other users to use it,privileges must be granted.

Applications should never connect to the database as its owner or a superuser, because these users can execute any query at will, for example, modifying the schema (e.g. dropping tables) or deleting its entirecontent

**Connecting to Database:**

You may want to establish the connections over SSL to encrypt client/server communications for increased security, or you can use ssh to encrypt the network connection between clients and the databaseserver.

**Encrypted Storage Model:**

SSL/SSH protects data travelling from the client to the server, SSL/SSH does not protect thepersistent data stored in a database. SSL is an on-the-wire protocol.

Once an attacker gains access to your database directly (bypassing the webserver), the stored sensitive data may be exposed or misused, unless the information is protected by the database itself. Encrypting the data is a good way to mitigate this threat, but very few databases offer this type of dataencryption.

**SQL Injection:**

SQL injection is a code injection technique that exploits a security vulnerability occurring in the database layer of an application. The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed.

**Preventing SQL injection**

To protect against SQL injection, user input must not directly be embedded in SQL statements. Instead, parameterized statements must be used (preferred), or user input must be carefully escaped orfiltered.

**Parameterized statements**

With most development platforms, parameterized statements can be used that work with parameters(sometimes called placeholders or bind variables) instead of embedding user input in the statement.

***Enforcement at the database level***

Currently only the H2 Database Engine supports the ability to enforce query parameterization.

***Enforcement at the coding level***

Using object-relational mapping libraries avoids the need to write SQL code.

**Escaping**

A straight-forward, though error-prone, way to prevent injections is to escape characters that have aspecial meaning in SQL.

**Error Reporting:**

With PHP security, there are two sides to error reporting. One is beneficial to increasing security, theother is detrimental.

A standard attack tactic involves profiling a system by feeding it improper data, and checking for thekinds, and contexts, of the errors which are returned.

**Attacking Variables with a custom HTML page**

```
<form method="post" action="attacktarget?username=badfoo&amp;password=badfoo">

<input type="hidden" name="username" value="badfoo" />

<input type="hidden" name="password" value="badfoo" />

</form>
```

**Exploiting common debugging variables**

```
<form method="post" action="attacktarget?errors=Y&amp;showerrors=1&amp;debug=1">

<input type="hidden" name="errors" value="Y" />

<input type="hidden" name="showerrors" value="1" />

<input type="hidden" name="debug" value="1" />

</form>
```

**Using Register Globals:**

The register_globals setting controls how you access form, server, and environment variables. By default this variable is set to Off, requiring you to use special arrays to access these variables.

When on, register_globals will inject your scripts with all sorts of variables, like request variables from HTML forms. This coupled with the fact that PHP doesn't require variable initialization means writing insecure code is that much easier. It was a difficult decision, but the PHP community decided to disable this directive by default. When on, people use variables yet really don't know for sure where they come from and can only assume. Internal variables that are defined in the script itself get mixed up with request data sent byusers and disabling register_globals changes this.

**User Submitted Data:**

The greatest weakness in many PHP programs is not inherent in the language itself, but merely an issue of code not being written with security in mind. For this reason, you should always take the time to consider the implications of a given piece of code, to ascertain the possible damage if an unexpected variable is submitted to it.

**Magic Quotes:**

Magic Quotes is a process that automagically escapes incoming data to the PHP script. It's preferred to code with magic quotes off and to instead escape the data at runtime, as needed.

**What are Magic Quotes**

When on, all ' (single-quote), " (double quote), \ (backslash) and NULL characters are escaped with abackslash automatically. This is identical to what **addslashes()** does.

There are three magic quote directives:

- magic_quotes_gpc

  Affects HTTP Request data (GET, POST, and COOKIE). Cannot be set at runtime, and defaults toon in PHP.

  See also **get_magic_quotes_gpc()**.
- magic_quotes_runtime

  If enabled, most functions that return data from an external source, including databases and text files,will have quotes escaped with a backslash. Can be set at runtime, and defaults to off in PHP.

  See also **set_magic_quotes_runtime()** and **get_magic_quotes_runtime()**.

- magic_quotes_sybase

  If enabled, a single-quote is escaped with a single-quote instead of a backslash. If on, it completely overrides magic_quotes_gpc. Having both directives enabled means only single quotes are escaped as ''. Double quotes, backslashes and NULL's will remain untouched and unescaped.

See also **ini_get()** for retrieving its value.

**Why did we use Magic Quotes**

- Useful for beginners
  - Magic quotes are implemented in PHP to help code written by beginners from being dangerous. Although SQL Injection is still possible with magic quotes on, the risk is reduced.
- Convenience
  - For inserting data into a database, magic quotes essentially runs **addslashes()** on all Get, Post, and Cookie data, and does so automagically.

**Why not to use Magic Quotes**

- Portability
  - Assuming it to be on, or off, affects portability. Use **get_magic_quotes_gpc()** to check forthis, and code accordingly.
- Performance
  - Because not every piece of escaped data is inserted into a database, there is a performance loss for escaping all this data. Simply calling on the escaping functions (like **addslashes()**) atruntime is more efficient.
  - Although php.ini-dist enables these directives by default, php.ini-recommended disables it.This recommendation is mainly due to performance reasons.
- Inconvenience
  - Because not all data needs escaping, it's often annoying to see escaped data where it shouldn'tbe. For example, emailing from a form, and seeing a bunch of \' within the email. To fix, this may require excessive use of **stripslashes()**.

**Disabling Magic Quotes**

The magic_quotes_gpc directive may only be disabled at the system level, and not at runtime. Inotherwords, use of **ini_set()** is not an option.

**Hiding PHP**

A few simple techniques can help to hide PHP, possibly slowing down an attacker who is attemptingto discover weaknesses in your system. By setting expose_php = off in your php.ini file, you reduce the amount of information available to them.

Another tactic is to configure web servers such as apache to parse different filetypes through PHP,either with an .htaccess directive, or in the apache configuration file itself.

**Keeping Current**

PHP, like any other large system, is under constant scrutiny and improvement. Each new version willoften include both major and minor changes to enhance security and repair any flaws, configuration mishaps, and other issues that will affect the overall security and stability of your system.

## Templates:

Templating is an easy way to create large sites without much trouble in maintaining a consistent look and feel across them.In its simplest form, a template is an outline for a website. It contains all the layout andperhaps some CSS for that purpose. For the purpose of this tutorial, we will be using a simple html layout with a sidebar menu and a header and a main content section.

This is quite a common layout and we will be making the main content dynamic, so that by clickinga link in the menu, the main content will be changed by including a seperate php file.

**Simple PHP Template Class:**

The used shows how to implement a simple class to handle page templates.

**The Class (template.class.php)<?**

```
class Template { public $template;
   function load($filepath) {
   $this->template = file_get_contents($filepath);
   }
   function replace($var, $content) {
   $this->template = str_replace("#$var#", $content, $this->template);
   }

   function publish() {
   eval("?>".$this->template."<?");
   }
   }
   ?>
```

**The Template File (design.html):**

This file will contain the design of your web site and the blank fields that will be merged with content data.

```
<html>
<head>
<title>#title#</title>
</head>
<body>
    <h3>Hello #name#!</h3>
    <p>The time is: #datetime#</p>
    <? echo "<p>Embedded PHP works too!</p>"; ?>
    </body>
    </html>
```

**Usage (index.php)**

Now we will create a script that load the template and use the class to merge the data.

```
<?
include "template.class.php";
$template = new Template;
$template->load("design.html");
$template->replace("title", "My Template Class");
$template->replace("name", "William");
$template->replace("datetime", date("m/d/y"));
$template->publish();
?>
```

When you run the above script, index.php, it will output the following:

```
 <html>
<head>
<title>My Template Class</title>
</head>
<body>
<h3>Hello William!</h3>
<p>The time is: 03/10/04</p>
<p>Embedded PHP works too!</p>
</body>    </html>
```

**UNIT VI**

# PYTHON

**Syntax and Style – Python Objects – Numbers – Sequences – Strings – Lists andTuples – Dictionaries – Conditionals and Loops.**

## Introduction:

Python is a general purpose interpreted, interactive, object-oriented and high-level programming language. Python was created by Guido van Rossum in the late eighties and early nineties. Like Perl, Pythonsource code is now available under the GNU General Public License (GPL).Python was designed to be highly readable which uses English keywords frequently where as other languages use punctuation and it has fewer syntactical constructions than other languages.

- **Python is Interpreted:** This means that it is processed at runtime by the interpreter and you do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive:** This means that you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented:** This means that Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is Beginner's Language:** Python is a great language for the beginner programmers and supports the development of a wide range of applications, from simple text processing to WWW browsers to games.

## Syntax and Style:

### Statements and Syntax

Some rules and certain symbols are used with regard to statements in Python:

| **Symbol** | **Description** |
|---|---|
| Hash mark ( # ) | Indicates Python comments |
| NEWLINE ( \n ) | The standard line separator (one statement per line)Backslash ( \ ) Continues a line |
| Semicolon ( ; ) | Joins two statements on a line Colon ( : )Separates a header line |

from its suite

**Comments:**

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and upto the physical line end are part of the comment, and the Python interpreter ignores them.

#!/usr/bin/python# First comment

print "Hello, Python!"; # second commentThis will produce following result:

Hello, Python!

A comment may be on the same line after a statement or expression:name = "Madisetti" #

This is again comment

You can comment multiple lines as follows:

# This is a comment.

# This is a comment, too.# This is a comment, too.# I said that already.

**Continuation ( \ ):**

In Python you normally have one instruction per line. Long instructions can span several lines using the line-continuation character "\". Some instructions, as triple quoted strings, list, tuple and dictionary constructors or statements grouped by parentheses do not need a line-continuation character. It is possible to write several statements on the same line, provided they are separated by semi-colons.

# check conditions

**if** (weather_is_hot == 1) **and** \(shark_warnings == 0) :

send_goto_beach_mesg_to_pager()

**Multiple Statement Groups as Suites ( : )**

Groups of individual statements making up a single code block are called "suites" in Python.Compound or complex statements, such as if, while, def, and class, are those which require a header line and a suite. Header lines begin the statement (with the keyword) and terminate with a colon ( : ) and arefollowed by one or more lines which make up the suite.

**Multiple Statements on a Single Line ( ; )**

The semicolon ( ; ) allows multiple statements on the single line given that neither statement starts anew code block. Here is a sample snip using the semicolon:

Example:

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

**Module:**

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use.

A module is a Python object with arbitrarily named attributes that you can bind and reference.Simply, a module is a file consisting of Python code. A module can define functions, classes, and variables. A module can also include runnable code.

**Variable Assignment**

Python 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.

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:

```
#!/usr/bin/python
counter = 100                    # An integer assignmentmiles   = 1000.0              # A
floating point
name              = "John"      # A stringprint counter
print milesprint name
```

Here 100, 1000.0 and "John" are the values assigned to *counter*, *miles* and *name* variables,respectively. While running this program, this will produce following result:

===========

**Frame**

These are objects representing execution stack frames in Python. Frame objects contain all the information the Python interpreter needs to know during a runtime execution environment. Some of its attributes include a link to the previous stack frame, the code object (see above) that is being executed, dictionaries for the local and global namespaces, and the current instruction. Each function call results in a new frame object, and for each frame object, a C stack frame is created as well.

**Traceback**

When you make an error in Python, an exception is raised. If exceptions are not caught or"handled," the interpreter exits with some diagnostic information similar to the output

=========

shown below:

Traceback (innermost last):

File "<stdin>", line N?, in ???ErrorName: error reason

The traceback object is just a data item that holds the stack trace information for an exception and is created when an exception occurs. If a handler is provided for an exception, this handler is given access to the traceback object.

**Slice Objects**

Slice objects are used to represent slices when extended slice syntax is used. This is a slice using twocolons, or multiple slices or ellipses separated by commas, e.g., `a[i:j:step]`, `a[i:j, k:l]`, or `a[..., i:j]`. They are also created by the built-in slice function.

Special read-only attributes:

- start is the lower bound
- stop is the upper bound
- step is the step value

Each is of the above is None if omitted.These attributes can have any type.

Slice objects support one method:

**indices**(self, length)

This method takes a single integer argument length and computes information about the extended slice that the slice object would describe if applied to a sequence of length items. It returns a tuple of three integers; respectively these are the start and stop indices and the step or stride length of the slice. Missing or out-of-bounds indices are handled in a manner consistent with regular slices.

**Ellipsis**

Ellipsis is an object that can appear in slice notation. For example:myList[1:2, ..., 0]

Its interpretation is purely up to whatever implements the_`getitem`_function and sees `Ellipsis` objects there, but its main (and intended) use in in the numeric python extension, which adds a multidementional array type. Since there are more than one dimensions, slicing becomes more complex than just a start and stop index; it is useful to be able to slice in multiple dimentions as well. eg, given a 4x4 array, the top left area would be defined by the slice "[:2,:2]"

```
>>> a
array([ [ 1,  2,  3,  4  ], [ 5, 6,  7,  8  ],
```

[ 9, 10, 11, 12], [13, 14, 15, 16]  ])

>>> a[:2,:2] # top left array([[1, 2], [5, 6]])

Extending this further, Ellipsis is used here to indicate a placeholder for the rest of the array dimensions not specified. Think of it as indicating the full slice [:] for all the dimentions in the gap it is placed, so for a 3d array, `a[...,0]` is the same as `a[:,:,0]` and for 4d, `a[:,:,:,0]`. Similarly `a[0,...,0]` is `a[0,:,:,0]` (with however many colons in the middle make up the full number of dimensions in the array).

**Xrange**

XRange objects are created by the built-in function xrange(), a sibling of the range() built-in function and used when memory is limited and for when range() generates an unusually large data set.

**Standard Type OperatorsComparison Operators:**

Comparison operations may be applied to any object. Comparison operations receive higher priority than Boolean operations. Comparisons can be chained in series: "x < y <= z" is equivalent to "x < y" and "y
<= z" except that "y" is evaluated only once. If "x < y" proves false, "z" is not evaluated at all. The comparison operations may be summarised as follows: Assume variable a holds 10 and variable b holds 20 then:

| 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 two operands are equal or not, if values are not equal then condition becomes true. | (a <> b) is true. This is similar to != operator. |
| > | Checks if the value of left operand is greater than 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 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. |

Comparisons between objects of different types, except different numeric types and different string types, never return "equal". Some types (e.g., file objects) support only a degenerate notion of comparison where any two objects of that type are non-equal -- equality can only be established by comparing the contents. The <, <=, > and >= operators raise a TypeError when an operand is a complex number.

Instances of a class usually compare as non-equal unless the class defines the **__cmp ()** method. Implementation note: Objects of different types (except numbers) are ordered by their type names. Objects of the same type that don't support proper comparison are ordered by their address in memory (RAM).

Two more comparison operations, "in" and "not in", are granted the same syntactic priority but aresupported only by sequence types.

**Standard Type Built-in Functions**

**cmp()**

This function compares elements of two tuples.

**Syntax:**

cmp(tuple1,tuple2)

Or cmp(obj1,obj2)

**Parameters:**

Here is the detail of parameters:

- **tuple1**: first tuple to be compared
- **tuple2**: second tuple to be compared

**Return Value:**

If elements are of the same type, perform the compare and return the result. If elements are different types,check to see if they are numbers.

- If numbers, perform numeric coercion if necessary and compare.
- If either element is a number, then the other element is "larger" (numbers are "smallest").
- Otherwise, types are sorted alphabetically by name.

If we reached the end of one of the tuples, the longer tuple is "larger." If we exhaust both tuples and sharethe same data, the result is a tie, meaning that 0 is returned.

**Example:**

```
#!/usr/bin/python
tuple1, tuple2 = (123, 'xyz'), (456, 'abc')print cmp(tuple1, tuple2);

print cmp(tuple2, tuple1);tuple3 = tuple2 + (786,); print cmp(tuple2, tuple3)
```

This will produce following result:

```
-1

1

-1
```

**str() and repr()**

Return a string containing a nicely printable representation of an object. For strings, this returns the string itself. The difference with `repr(object)` is that `str(object)` does not always attempt to return a string that is acceptable to `eval()`; its goal is to return a printable string. If no argument is given, returns the empty string, `''`.

The str function coerces data into a string. Every datatype can be coerced into a string.str(1)

horsemen = ['war', 'pestilence', 'famine']print horsemen horsemen.append('Powerbuilder')

print str(horsemen)

**Standard Type Operators and Built-In Functions**

Built-In Functions

| Function | Description | Result |
|---|---|---|
| Cmp(pbj1,obj2) | compares two objects | integer |
| repr(*obj*) | string representation | string |
| str(*obj*) | string representation | String |
| type(*obj*) | determines object type | type object |

Value Comparisons

| Operator | Description | Result |
|---|---|---|
| < | less than | Boolean |
| > | greater than | Boolean |
| <= | less than or equal to | Boolean |
| >= | greater than or equal to | Boolean |
| == | equal to | Boolean |
| != | not equal to | Boolean |
| <> | not equal to | Boolean |

Object Comparisons

| Function | Description | Result |
|---|---|---|
| is | the same as | Boolean |
| is not | not the same as | Boolean |

Boolean operators

| Function | Description | Result |
|---|---|---|
| not | logical negation | Boolean |
| And | logical conjuction | Boolean |
| or | logical disjunction | Boolean |

**Unsupported Types**

A list of types that are not supported by Python.

**Boolean**

Unlike Pascal or Java, Python does not feature the Boolean type. Use integers instead.

**char or byte**

Python does not have a char or byte type to hold either single character or 8-bit integers. Use strings of length one for characters and integers for 8-bit numbers.

**pointer**

Since Python manages memory for you, there is no need to access pointer addresses. The closest to an address that you can get in Python is by looking at an object's identity using the id() built-infunction. Since you have no control over this value, it's a moot point.

**int vs. short vs. long**

Python's plain integers are the universal "standard" integer type, obviating the need for three different integer types, i.e., C's int, short, and long. For the record, Python's integers are implemented as C longs. For values larger in magnitude than regular integers (usually your system architecture size, i.e., 32-bit), use Python's long integer.

**float vs. double**

C has both a single precision float type and double-precision double type. Python's float type is actually a C double. Python does not support a single-precision floating point type because its benefits are outweighed by the overhead required to support twotypes of floating point types.

## Numbers:

Number data types store numeric values. They are immutable data types, which means that changing the value of a number data type results in a newly allocated object.Number objects are created when you assign a value to them. For example:

```
var1 = 1
var2 = 10
```

You can also delete the reference to a number object by using the **del** statement. The syntax of the delstatement is:

```
del var1[,var2[,var3[ .................. ,varN]]]]
```

You can delete a single object or multiple objects by using the del statement. For

example:del var

del var_a, var_b

**How to Create and Assign Numbers (Number Objects):**

Creating numbers is as simple as assigning a value to a variable:anInt = 1

1aLong = -9999999999999999L
aFloat = 3.14159265358979323846264338327 95

aComplex = 1.23 + 4.56J

**How to Update Numbers:**

You can "update" an existing number by (re)assigning a variable to another number. Thenew value can be related to its previous value or to a completely different number altogether.

anInt = anInt + 1 aFloat = 2.718281828

**How to Remove Numbers:**

Under normal circumstances, you do not really "remove" a number; you just stop using it! If youreally want to delete a reference to a number object, just use the **del** statement You can no longer use the variable name, once removed, unless you assign it to a new object; otherwise, youwill cause a NameError exception to occur.

```
del anInt
del aLong, aFloat, aComplex
```

**Types:**

Python supports four different numerical types:

- **int (signed integers)**: often called just integers or ints, are positive or negative whole numbers withno decimal point.
- **long (long integers )**: or longs, are integers of unlimited size, written like integers and followed byan uppercase or lowercase L.
- **float (floating point real values)** : or floats, represent real numbers and are written with a decimal point dividing the integer and fractional parts. Floats may also be in scientific notation, with E or e indicating the power of 10 (2.5e2 = 2.5 x $10^2$ = 250).
- **complex (complex numbers)** : are of the form a + bJ, where a and b are floats and J (or j) representsthe square root of -1 (which is an imaginary number). a is the real part of the number, and b is the imaginary part. Complex numbers are not used much in Python programming.

**Examples:**

Here are some examples of numbers:

| **Int** | **long** | **float** | **complex** |
|---|---|---|---|
| 10 | 51924361L | 0.0 | 3.14j |
| 100 | -0x19323L | 15.20 | 45.j |
| -786 | 0122L | -21.9 | 9.322e-36j |
| 080 | 0xDEFABCECBDAECBFBAEl | 32.3+e18 | .876j |
| -0490 | 535633629843L | -90. | -.6545+0J |
| -0x260 | -052318172735L | -32.54e100 | 3e+26J |
| 0x69 | -4721885298529L | 70.2-E12 | 4.53e-7j |

- Python allows you to use a lowercase L with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.
- A complex number consists of an ordered pair of real floating point numbers denoted by a + bj,where a is the real part and b is the imaginary part of the complex number.

**Number Type Conversion:**

Python converts numbers internally in an expression containing mixed types to a common type for evaluation. But sometimes, you'll need to coerce a number explicitly from one type to another to satisfy therequirements of an operator or function parameter.

- Type **int(x)**to convert x to a plain integer.
- Type **long(x)** to convert x to a long integer.
- Type **float(x)** to convert x to a floating-point number.
- Type **complex(x)** to convert x to a complex number with real part x and imaginary part zero.
- Type **complex(x, y)** to convert x and y to a complex number with real part x and imaginary part y. xand y are numeric expressions

**Built-in Number Functions:**

**Mathematical Functions:**

Python includes following functions that perform mathematical calculations.

| **Function** | **Returns ( description )** |
|---|---|
| abs(x) | The absolute value of x: the (positive) distance between x and zero. |
| ceil(x) | The ceiling of x: the smallest integer not less than x |
| cmp(x, y) | -1 if x < y, 0 if x == y, or 1 if x > y |
| exp(x) | The exponential of x: $e^x$ |
| fabs(x) | The absolute value of x. |
| floor(x) | The floor of x: the largest integer not greater than x |
| log(x) | The natural logarithm of x, for x> 0 |
| log10(x) | The base-10 logarithm of x for x> 0 . |
| max(x1, x2,...) | The largest of its arguments: the value closest to positive infinity |

| | |
|---|---|
| min(x1, x2,...) | The smallest of its arguments: the value closest to negative infinity |
| modf(x) | The fractional and integer parts of x in a two-item tuple. Both parts have thesame sign as x. The integer part is returned as a float. |
| pow(x, y) | The value of x**y. |
| round(x [,n]) | **x** rounded to n digits from the decimal point. Python rounds away from zero as atie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0. |
| sqrt(x) | The s |

**Random Number Functions:**

Random numbers are used for games, simulations, testing, security, and privacy applications. Pythonincludes following functions that are commonly used.

| Function | Returns ( description ) |
|---|---|
| choice(seq) | A random item from a list, tuple, or string. |
| randrange ([start,] stop[,step]) | A randomly selected element from range(start, stop, step) |
| random() | A random float r, such that 0 is less than or equal to r and r is less than 1 |
| seed([x]) | Sets the integer starting value used in generating random numbers. Call thisfunction before calling any other random module function. Returns None. |
| shuffle(lst) | Randomizes the items of a list in place. Returns None. |
| uniform(x, y) | A random float r, such that x is less than or equal to r and r is less than y |

**Trigonometric Functions:**

Python includes following functions that perform trigonometric calculations.

| Function | Description |
|---|---|
| acos(x) | Return the arc cosine of x, in radians. |
| asin(x) | Return the arc sine of x, in radians. |

| | |
|---|---|
| atan(x) | Return the arc tangent of x, in radians. |
| atan2(y, x) | Return atan(y / x), in radians. |
| cos(x) | Return the cosine of x radians. |
| hypot(x, y) | Return the Euclidean norm, sqrt(x*x + y*y). |
| sin(x) | Return the sine of x radians. |
| tan(x) | Return the tangent of x radians. |
| degrees(x) | Converts angle x from radians to degrees. |
| radians(x) | Converts angle x from degrees to radians. |

**Mathematical Constants:**

The module also defines two mathematical constants:

| Constant | Description |
|---|---|
| Pi | The mathematical constant pi. |
| E | The mathematical constant e. |

## Python Sequences:

The most basic data structure in Python is the sequence. Each element of a sequence is assigned a number - its position, or index. The first index is zero, the second index is one, and so forth.

Python has six built-in types of sequences. There are certain things you can do with all sequence types. These operations include indexing, slicing, adding, multiplying, and checking for membership. In addition, Python has built- in functions for finding the length of a sequence, and for finding its largest and smallest elements.

## Strings:

Strings are amongst the most popular types in Python. We can create them simply by enclosingcharacters in quotes. Python treats single quotes the same as double quotes.

Creating strings is as simple as assigning a value to a variable. For example:

var1 = 'Hello World!'

var2 = "Python Programming"

**Accessing Values in Strings:**

Python does not support a character type; these are treated as strings of length one, thus alsoconsidered a substring.

To access substrings, use the square brackets for slicing along with the index or indices to obtain yoursubstring:

**Example:**

#!/usr/bin/python var1 = 'Hello World!'

var2 = "Python Programming"print "var1[0]: ", var1[0]

print "var2[1:5]: ", var2[1:5]

This will produce following result:var1[0]: H

var2[1:5]: ytho

**Updating Strings**

You can "update" an existing string by (re)assigning a variable to another string. The new value canbe related to its previous value or to a completely different string altogether.

**Example:**

#!/usr/bin/python var1 = 'Hello World!'

print "Updated String :- ", var1[:6] + 'Python'

This will produce following result: Updated String :-  Hello Python

**Escape Characters**

Following table is a list of escape or non-printable characters that can be represented with backslashnotation.

**NOTE:** In a doublequoted string, an escape character is interpreted; in a singlequoted string, an escapecharacter is preserved.

| **Backslashnotation** | **Hexadecimalcharacter** | **Description** |
|---|---|---|
| \a | 0x07 | Bell or alert |
| \b | 0x08 | Backspace |
| \cx | - | Control-x |
| \C-x | - | Control-x |
| \e | 0x1b | Escape |

| | | |
|---|---|---|
| \f | 0x0c | Formfeed |
| \M-\C-x | - | Meta-Control-x |
| \n | | 0x0a |
| \nnn | | - |
| \r | | 0x0d |
| \s | | 0x20 |
| \t | | 0x09 |
| \v | | 0x0b |
| \x | | - |
| \xnn | | - |

| |
|---|
| Newline |
| Octal notation, where n is in the range 0.7 |
| Carriage return |
| Space |
| Tab |
| Vertical tab |
| Character x |
| Hexadecimal notation, where n is in the range 0.9, a.f, or A.F |

**String Special Operators:**

Assume string variable a holds 'Hello' and variable b holds 'Python' then:

| Operator | Description | Example |
|---|---|---|
| + | Concatenation - Adds values on either side ofthe operator | a + b will give HelloPython |
| * | Repetition - Creates new strings, concatenatingmultiple copies of the same string | a*2 will give - HelloHello |
| [] | Slice - Gives the character from the givenindex | a[1] will give **e** |
| [ : ] | Range Slice - Gives the characters from thegiven range | a[1:4] will give **ell** |
| In | Membership - Returns true if a character existsin the given string | **H in a** will give 1 |
| not in | Membership - Returns true if a character doesnot exist in the given string | **M not in a** will give 1 |
| r/R | Raw String - Suppress actual meaning of Escape characters. The syntax for raw stringsis exactly the same as for normal strings with the exception of the raw string operator, the letter "r," which precedes the quotation marks. The "r" can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark. | **print r'\n'** prints \n and **print R'\n'** prints \n |
| % | Format - Performs String formatting | See at next section |

**String Formatting Operator:**

One of Python's coolest features is the string format operator **%**. This operator is unique to strings andmakes up for the pack of having functions from C's printf() family.

**Example:**

```
#!/usr/bin/python
print "My name is %s and weight is %d kg!" % ('Zara',21)
```

This will produce following result:

```
My name is Zara and weight is 21 kg!
```

Here is the list of complete set of symbols which can be used along with %:

| **Format Symbol** | **Conversion** |
|---|---|
| %c | character |
| %s | string conversion via str() prior to formatting |
| %i | signed decimal integer |
| %d | signed decimal integer |
| %u | unsigned decimal integer |
| %o | octal integer |
| %x | hexadecimal integer (lowercase letters) |
| %X | hexadecimal integer (UPPERcase letters) |
| %e | exponential notation (with lowercase 'e') |
| %E | exponential notation (with UPPERcase 'E') |
| %f | floating point real number |
| %g | the shorter of %f and %e |
| %G | the shorter of %f and %E |

Other supported symbols and functionality are listed in the following table:

| **Symbol** | **Functionality** |
|---|---|
| * | argument specifies width or precision |
| - | left justification |
| + | display the sign |

|  |  |
|---|---|
| <sp> | leave a blank space before a positive number |
| # | add the octal leading zero ( '0' ) or hexadecimal leading '0x' or '0X',depending on whether 'x' or 'X' were used. |
| 0 | pad from left with zeros (instead of spaces) |
| % | '%%' leaves you with a single literal '%' |
| (var) | mapping variable (dictionary arguments) |
| m.n. | m is the minimum total width and n is the number of digits to display afterthe decimal point (if appl.) |

**Triple Quotes:**

Python's triple quotes comes to the rescue by allowing strings to span multiple lines, including verbatimNEWLINEs, TABs, and any other special characters.

The syntax for triple quotes consists of three consecutive **single or double** quotes.

```
#!/usr/bin/python
para_str = """this is a long string that is made up ofseveral lines and non-printable characters such as
TAB ( \t ) and they will show up that way when displayed. NEWLINEs within the string,
whether explicitly given likethis within the brackets [ \n ], or just a NEWLINE within the
variable assignment will also show up.
"""                        print para_str;
```

Note how every single special character has been converted to its printed form, right down to the last NEWLINE at the end of the string between the "up." and closing triple quotes. Also note that NEWLINEs occur either with an explicit carriage return at the end of a line or its escape code (\n): this is a long string that is made up of several lines and non-printable characters such as TAB (     ) and they will show up that way when displayed. NEWLINEs within the string, whether explicitly given likethis within the brackets [ ], or just a NEWLINE within the variable assignment will also show up.

**Raw String:**

Raw strings don't treat the backslash as a special character at all. Every character you put into a raw stringstays the way you wrote it:

```
#!/usr/bin/python

print 'C:\\nowhere'
```

This would print following result:C:\nowhere

Now let's make use of raw string. We would put expression in **r'expression'** as follows:

```
#!/usr/bin/python

print r'C:\\nowhere'
```

This would print following result:

```
C:\\nowhere
```

**Unicode String**

Normal strings in Python are stored internally as 8-bit ASCII, while Unicode strings are stored as 16- bit Unicode. This allows for a more varied set of characters, including special characters from most languages in the world. I'll restrict my treatment of Unicode strings to the following:

```
#!/usr/bin/python print u'Hello, world!'
```

This would print following result:Hello, world!

As you can see, Unicode strings use the prefix u, just as raw strings use the prefix r.

**Built-in String Methods**

Python includes following string method:

| SN | **Methods with Description** | |
|---|---|---|
| 1 | capitalize() | Capitalizes first letter of string |
| 2 | center(width, fillchar) centered to a total of width columns | Returns a space-padded string with the original string |
| 3 | count(str, beg= 0,end=len(string)) Counts how many times str occurs in string, or in a substringof string if starting index beg and ending index end are given | |
| 3 | decode(encoding='UTF-8',errors='strict') Decodes the string using the codec registered for encoding. encoding defaults to the default stringencoding. | |

| | |
|---|---|
| 4 | encode(encoding='UTF-8',errors='strict') Returns encoded string version of string; on error,default is to raise a ValueError unless errors is given with 'ignore' or 'replace'. |
| 5 | endswith(suffix, beg=0, end=len(string)) Determines if string or a substring of string (if startingindex beg and ending index end are given) ends with suffix; Returns true if so, and false otherwise |
| 6 | expandtabs(tabsize=8) spaces per tab if tabsize not provided Expands tabs in string to multiple spaces; defaults to 8 |
| 7 | find(str, beg=0 end=len(string)) Determine if str occurs in string, or in a substring ofstring if starting index beg and ending index end are given; returns index if found and -1 otherwise |
| 8 | index(str, beg=0, end=len(string)):Same as find(), but raises an exception if str not found |
| 9 | isa1num():Returns true if string has at least 1 character and all characters arealphanumeric and false otherwise |
| 10 | isalpha():Returns true if string has at least 1 character and all characters are alphabetic and false otherwise |
| 11 | isdigit():Returns true if string contains only digits and false otherwise |
| 12 | islower():Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise |
| 13 | isnumeric():Returns true if a unicode string contains only numeric characters and falseotherwise |
| 14 | isspace() :Returns true if string contains only whitespace characters and false otherwise |
| 15 | istitle():Returns true if string is properly "titlecased" and false otherwise |
| 16 | isupper() :Returns true if string has at least one cased character and all cased characters are inuppercase and false otherwise |
| 17 | join(seq) :Merges (concatenates) the string representations of elements in sequence seq into a string,with separator string |

| | |
|---|---|
| 18 | len(string):Returns the length of the string |
| 19 | ljust(width[, fillchar]) Returns a space-padded string with the original string left-justified to a total ofwidth columns |
| 20 | lower()Converts all uppercase letters in string to lowercase |
| 21 | lstrip()Removes all leading whitespace in string |
| 22 | maketrans() Returns a translation table to be used in translate function. |
| 23 | max(str) Returns the max alphabetical character from the string str |
| 24 | min(str) Returns the min alphabetical character from the string str |
| 25 | replace(old, new [, max]) Replaces all occurrences of old in string with new, or at most maxoccurrences if max given |
| 26 | rfind(str, beg=0,end=len(string)) Same as find(), but search backwards in string |
| 27 | rindex( str, beg=0, end=len(string)) Same as index(), but search backwards in string |
| 28 | rjust(width,[, fillchar]) Returns a space-padded string with the original string right-justified to a totalof width columns. |
| 29 | rstrip() Removes all trailing whitespace of string |
| 30 | split(str="", num=string.count(str)) Splits string according to delimiter str (space if not provided)and returns list of substrings; split into at most num substrings if given |
| 31 | splitlines( num=string.count('\n')) Splits string at all (or num) NEWLINEs and returns a list of eachline with NEWLINEs removed |
| 32 | startswith(str, beg=0,end=len(string)) Determines if string or a substring of string (if startingindex beg and ending index end are given) starts with substring str; Returns true if so, and false otherwise |
| 33 | strip([chars]) Performs both lstrip() and rstrip() on string |
| 34 | swapcase() Inverts case for all letters in string |

|  |  |
|---|---|
| 35 | title() Returns "titlecased" version of string, that is, all words begin with uppercase, and the rest are lowercase |
| 36 | translate(table, deletechars="") Translates string according to translation table str(256 chars),removing those in the del string |
| 37 | upper() Converts lowercase letters in string to uppercase |
| 38 | zfill (width) Returns original string leftpadded with zeros to a total of width characters;intended for numbers, zfill() retains any sign given (less one zero) |
| 39 | isdecimal()Returns true if a unicode string contains only decimal characters and false otherwise |

## Python Lists:

The list is a most versatile data type available in Python, which can be written as a list of comma- separated values (items) between square brackets. Good thing about a list that items in a list need not all have the same type:

Creating a list is as simple as putting different comma-separated values between squere brackets. Forexample:

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

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

**Accessing Values in Lists:**

To access values in lists, use the square brackets for slicing along with the index or indices to obtain valueavailable at that index:

**Example:**

```
#!/usr/bin/python
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7 ];
print "list1[0]: ", list1[0]
print "list2[1:5]: ", list2[1:5]
```

This will produce following result:

```
list1[0]: physics list2[1:5]:  [2, 3, 4, 5]
```

**Updating Lists:**

You can update single or multiple elements of lists by giving the slice on the left-hand side of theassignment operator, and you can add to elements in a list with the append() method:

**Example:**

```
#!/usr/bin/python
list1 = ['physics', 'chemistry', 1997, 2000];print "Value
available at index 2 : "
print list1[2]; list1[2] = 2001;
print "New value available at index 2 : "print list1[2];
```

**Note:** append() method is discussed in subsequent section.This will produce following result:

```
Value available at index 2 : 1997
New value available at index 2 : 2001
```

**Delete List Elements:**

To remove a list element, you can use either the del statement if you know exactly which element(s) you aredeleting or the remove() method if you do not know.

**Example:**

```
#!/usr/bin/python
list1 = ['physics', 'chemistry', 1997, 2000];print list1;
del list1[2];
print "After deleting value at index 2 : "print list1;
```

This will produce following result: ['physics', 'chemistry', 1997, 2000]After deleting value at index 2 :

['physics', 'chemistry', 2000]

**Note:** remove() method is discussed in subsequent section.

**Basic List Operations:**

Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too,except that the result is a new list, not a string.

| Python Expression | Results | Description |
|---|---|---|
| len([1, 2, 3]) | 3 | Length |
| [1, 2, 3] + [4, 5, 6] | [1, 2, 3, 4, 5, 6] | Concatenation |

| | | |
|---|---|---|
| ['Hi!'] * 4 | ['Hi!', 'Hi!', 'Hi!', 'Hi!'] | Repetition |
| 3 in [1, 2, 3] | True | Membership |
| for x in [1, 2, 3]: print x, | 1 2 3 | Iteration |

**Indexing, Slicing, and Matrixes:**

Because lists are sequences, indexing and slicing work the same way for lists as they do for strings.Assuming following input:

L = ['spam', 'Spam', 'SPAM!']

| **Python Expression** | **Results** | **Description** |
|---|---|---|
| L[2] | 'SPAM!' | Offsets start at zero |
| L[-2] | 'Spam' | Negative: count from the right |
| L[1:] | ['Spam', 'SPAM!'] | Slicing fetches sections |

**Built-in List Functions & Methods**

Python includes following list functions

| **SN** | **Function with Description** | |
|---|---|---|
| 1 | cmp(list1,list2) | Compares elements of both lists. |
| 2 | len(list) | Gives the total length of the list. |
| 3 | max(list) | Returns item from the list with max value. |
| 4 | min(list) | Returns item from the list with min value. |
| 5 | list(seq) | Converts a tuple into list. |

Python includes following list methods

| **SN** | **Methods with Description** | |
|---|---|---|
| 1 | list.append(obj) | Appends object obj to list |
| 2 | list.count(obj) | Returns count of how many times obj occurs in list |
| 3 | list.extend(seq) | Appends the contents of seq to list |
| 4 | list.index(obj) | Returns the lowest index in list that obj |

| | | |
|---|---|---|
| | | appears |
| 5 | list.insert(index,obj) | Inserts object obj into list at offset index |
| 6 | list.pop(obj=list[-1]) | Removes and returns last object or obj from list |
| 7 | list.remove(obj) | Removes object obj from list |
| 8 | list.reverse() | Reverses objects of list in place |
| 9 | list.sort([func]) | Sorts objects of list, use compare func if given |

## Tuples:

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The only difference is that tuples can't be changed ie. tuples are immutable and tuples use parentheses and lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values and optionally you can put these comma-separated values between parentheses also. For example:

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing:tup1 = ();

To write a tuple containing a single value you have to include a comma, even though there is only one value:tup1 = (50,);

Like string indices, tuple indices start at 0, and tuples can be sliced, concatenated and so on.

**Accessing Values in Tuples**

To access values in tuple, use the square brackets for slicing along with the index or indices to obtainvalue available at that index:

**Example:**

```
#!/usr/bin/python
tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5, 6, 7 );
print "tup1[0]: ", tup1[0]
print "tup2[1:5]: ", tup2[1:5]
```

This will produce following result:tup1[0]:  physics

```
tup2[1:5]:  [2, 3, 4, 5]
```

**Updating Tuples:**

Tuples are immutable which means you cannot update them or change values of tuple elements. But we able able to take portions of an existing tuples to create a new tuples as follows:

**Example:**

```
#!/usr/bin/python tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');
# Following action is not valid for tuples# tup1 += tup2;
# So let's create a new tuple as followstup3 = tup1 + tup2;
print tup3;
```

This will produce following result:

```
(12, 34.56, 'abc', 'xyz')
```

**Delete Tuple Elements**

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded. To explicitly remove an entire tuple, just use the **del** statement:

**Example:**

```
#!/usr/bin/python
tup = ('physics', 'chemistry', 1997, 2000);print tup;
del tup;
print "After deleting tup : "print tup;
```

This will produce following result. Note an exception raised, this is because after **del tup** tuple does notexist any more:

```
('physics', 'chemistry', 1997, 2000)After deleting tup :
Traceback (most recent call last):
File "test.py", line 9, in <module>print tup;
NameError: name 'tup' is not defined
```

**Basic Tuples Operations:**

Tuples respond to the + and * operators much like strings; they mean concatenation and repetitionhere too, except that the result is a new tuple, not a string.

| Python Expression | Results | Description |
|---|---|---|
| len((1, 2, 3)) | 3 | Length |
| (1, 2, 3) + (4, 5, 6) | (1, 2, 3, 4, 5, 6) | Concatenation |
| ['Hi!'] * 4 | ('Hi!', 'Hi!', 'Hi!', 'Hi!') | Repetition |
| 3 in (1, 2, 3) | True | Membership |
| for x in (1, 2, 3): print x, | 1 2 3 | Iteration |

**Indexing, Slicing, and Matrixes**

Because tuples are sequences, indexing and slicing work the same way for tuples as they do for strings.

Assuming following input:

L = ('spam', 'Spam', 'SPAM!')

| Python Expression | Resu lts | Description |
|---|---|---|
| L[2] | 'SPAM!' | Offsets start at zero |
| L[-2] | 'Spam' | Negative: count from the right |
| L[1:] | ['Spam', 'SPAM!'] | Slicing fetches sections |

**No Enclosing Delimiters**

Any set of multiple objects, comma-separated, written without identifying symbols, i.e., brackets for lists, parentheses for tuples, etc., default to tuples, as indicated in these short examples:

```
#!/usr/bin/python
print 'abc', -4.24e93, 18+6.6j, 'xyz';x, y = 1, 2;
print "Value of x , y : ", x,y;print var;
```

This will reduce following result:

```
abc -4.24e+93 (18+6.6j) xyzValue of x , y : 1 2
```

**Built-in Tuple Functions**

Python includes following tuple functions

| SN | Function with Description | |
|---|---|---|
| 1 | cmp(tuple1, tuple2) | Compares elements of both tuples. |
| 2 | len(tuple) | Gives the total length of the tuple. |
| 3 | max(tuple) | Returns item from the tuple with max value. |
| 4 | min(tuple) | Returns item from the tuple with min value. |
| 5 | tuple(seq) | Converts a list into tuple. |

## Dictionaries:

A dictionary is mutable and is another container type that can store any number of Python objects, including other container types. Dictionaries consist of pairs (called items) of keys and their corresponding values. Python dictionaries are also known as associative arrays or hash tables. The general syntax of a dictionary is as follows:

dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}You can create dictionary in the following way as well:

```
dict1 = { 'abc': 456 };
dict2 = { 'abc': 123, 98.6: 37 };
```

Each key is separated from its value by a colon (:), the items are separated by commas, and thewhole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curlybraces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be ofany type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

**Accessing Values in Dictionary**

To access dictionary elements, you use the familiar square brackets along with the key to obtain its

value:

**Example:**

```
#!/usr/bin/python
```

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};

print "dict['Name']: ", dict['Name'];

print "dict['Age']: ", dict['Age'];This will produce following result:

dict['Name']:  Zaradict['Age']:  7

If we attempt to access a data item with a key which is not part of the dictionary, we get an error as follows:

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};

print "dict['Alice']: ", dict['Alice'];This will produce following result:

dict['Zara']:

Traceback (most recent call last): File "test.py", line 4, in <module> print "dict['Alice']: ", dict['Alice']; KeyError: 'Alice'

**Updating Dictionary**

You can update a dictionary by adding a new entry or item (i.e., a key-value pair), modifying anexisting entry, or deleting an existing entry as shown below:

**Example:**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}; dict['Age'] = 8; # update existing entry

dict['School'] = "DPS School"; # Add new entrySprint "dict['Age']: ", dict['Age'];

print "dict['School']: ", dict['School'];This will produce following result:

dict['Age']:  8 dict['School']:  DPS School

**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:

**Example:**

#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}; del dict['Name']; # remove entry with key

'Name'dict.clear();             # remove all entries in dict

del dict ;                          # delete entire dictionaryprint "dict['Age']: ", dict['Age'];

```
print "dict['School']: ", dict['School'];
```

This will produce following result. Note an exception raised, this is because after **del dict** dictionary doesnot exist any more:

```
dict['Age']:
Traceback (most recent call last): File "test.py", line 8, in <module>print "dict['Age']: ",
dict['Age'];

TypeError: 'type' object is unsubscriptable
```

**Note:** del() method is discussed in subsequent section.

**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 duplicatekeys encountered during assignment, the last assignment wins.

**Example:**

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'};
        print "dict['Name']: ", dict['Name'];This will produce following result:
```

```
dict['Name']:  Manni
```

**(b)** Keys must be immutable. Which means you can use strings, numbers, or tuples as dictionary keys butsomething like ['key'] is not allowed.

***Example:***

```
#!/usr/bin/python
dict = {['Name']: 'Zara', 'Age': 7};
print "dict['Name']: ", dict['Name'];
```

This will produce following result. Note an exception raised:Traceback (most recent call

```
last):
  File "test.py", line 3, in <module> dict = {['Name']: 'Zara', 'Age': 7}; TypeError: list objects
are unhashable
```

**Built-in Dictionary Functions & Methods**

Python includes following dictionary functions

| SN | Function with Description |
|---|---|
| 1 | cmp(dict1, dict2) Compares elements of both dict. |
| 2 | len(dict) Gives the total length of the dictionary. This would be equal to thenumber of items in the dictionary. |
| 3 | str(dict) Produces a printable string representation of a dictionary |
| 4 | type(variable) Returns the type of the passed variable. If passed variable is dictionarythen it would return a dictionary type. |

Python includes following dictionary methods

| SN | Methods with Description |
|---|---|
| 1 | dict.clear() Removes all elements of dictionary *dict* |
| 2 | dict.copy() Returns a shallow copy of dictionary *dict* |
| 2 | dict.fromkeys() Create a new dictionary with keys from seq and values *set* to *value*. |
| 3 | dict.get(key, default=None) For *key* key, returns value or default if key not in dictionary |
| 4 | dict.has_key(key) Returns *true* if key in dictionary *dict*, *false* otherwise |
| 5 | dict.items() Returns a list of *dict*'s (key, value) tuple pairs |
| 6 | dict.keys() Returns list of dictionary dict's keys |
| 7 | dict.setdefault(key, default=None) Similar to get(), but will set dict[key]=default if *key* is not alreadyin dict |
| 8 | dict.update(dict2) Adds dictionary *dict2*'s key-values pairs to *dict* |
| 9 | dict.values() Returns list of dictionary *dict2*'s values |

## Conditionals and Loops

Conditional constructs are used to incorporate decision making into programs. The result of this decision making determines the sequence in which a program will execute instructions. You can control the flow of a program by using conditional constructs.

**The if statement:**

The **if** statement of Python is similar to that of other languages. The **if** statement contains a logicalexpression using which data is compared, and a decision is made based on the result of the comparison.

**The syntax of the if statement is:**

if expression: statement(s)

**Example:**

```
#!/usr/bin/pythonvar1 = 100
if var1:
print "1 - Got a true expression value"print var1
var2 = 0if var2:
print "2 - Got a true expression value"print var2
print "Good bye!"
```

This will produce following result:

```
1 - Got a true expression value100
Good bye!
```

**The else Statement:**

An **else** statement can be combined with an **if** statement. An **else** statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a false value. The else statement is an optional statement and there could be at most only one **else** statement following **if** .

The syntax of the if...else statement is:

```
if expression:statement(s)
  else:
  statement(s)
```

**Example:**

```
#!/usr/bin/pythonvar1 = 100
if var1:

print "1 - Got a true expression value"print var1
else:
print "1 - Got a false expression value"print var1
```

```
var2 = 0if var2:
print "2 - Got a true expression value"print var2
else:
print "2 - Got a false expression value"print var2
print "Good bye!"
```

This will produce following result:

```
1       - Got a true expression value100
2       - Got a false expression value0
Good bye!
```

**The elif Statement**

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

The syntax of the if...elif statement is:

```
if
  expression1:statement(s)
elif expression2:statement(s)
elif exp
  ression3:statement(s)
  else:


  statement(s)
```

**Note:** Python does not currently support switch or case statements as in other languages.

**Example:**

```
#!/usr/bin/pythonvar = 100
if var == 200:
print "1 - Got a true expression value"print var
elif var == 150:
print "2 - Got a true expression value"print var2
elif var == 100:
print "3 - Got a true expression value"print var
```

```
else:
print "4 - Got a false expression value"print var


print "Good bye!"
```

This will produce following result:

```
3             - Got a true expression value100
Good bye!
```

**The Nested if...elif...else Construct:**

There may be a situation when you want to check for another condition after a condition resolves to true. In such a situation, you can use the nested **if** construct. In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

The syntax of the nested if...elif...else construct may be:

```
if expression1:statement(s)
  if expression2:
  statement(s) elif expression3:
     statement(s)else
     statement(s) elif expression4:
     statement(s)else:
     statement(s)
```

**Example:**

```
#!/usr/bin/python


var = 100
if var < 200:
    print "Expression value is less than 200"if var == 150:
    print "Which is 150"elif var == 100:
    print "Which is 100"elif var == 50:
    print "Which is 50"elif var < 50:
    print "Expression value is less than 50"else:
    print "Could not find true expression"


print "Good bye!"
```

This will produce following result:

```
Expression value is less than 200Which is 100
Good bye!
```

**Single Statement Suites:**

If the suite of an **if** clause consists only of a single line, it may go on the same line as the header statement:Here is an example of a one-line if clause:

```
if ( expression == 1 ) : print "Value of expression is 1"
```

**Python - while Loop Statements**

A loop is a construct that causes a section of a program to be repeated a certain number of times. The repetition continues while the condition set for the loop remains true. When the condition becomes false, the loop ends and the program control is passed to the statement following the loop.

**The while Loop:**

The **while** loop is one of the looping constructs available in Python. The **while** loop continues until the expression becomes false. The expression has to be a logical expression and must return either a true or afalse value.

The syntax of the while loop is:

```
while expression:
statement(s)
```

Here **expression** statement is evaluated first. If expression is true that is, then the statement(s) block is executed repeatedly until expression becomes false. Otherwise, the next statement following the statement(s) block is executed.

**Note:** In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

**Example:**

```
#!/usr/bin/pythoncount = 0
while (count < 9):
print 'The count is:', countcount = count + 1
print "Good bye!"
```

This will produce following result:The count is: 0

```
The count is: 1The count is: 2The count is: 3The count is: 4The count is: 5The count is: 6The count is: 7The count is: 8Good bye!
```

The block here, consisting of the print and increment statements, is executed repeatedly until count isno longer less than 9. With each iteration, the current value of the index count is displayed and thenincreased by 1.

**The Infinite Loops:**

You must use caution when using while loops because of the possibility that this condition never resolves to a false value. This results in a loop that never ends. Such a loop is called an infinite loop.An infinite loop might be useful in client/server programming where the server needs to run continuously so thatclient programs can communicate with it as and when required.

**Example:**

Following loop will continue till you enter CTRL+C :

```
#!/usr/bin/pythonvar = 1
while var == 1 : # This constructs an infinite loopnum = raw_input("Enter a number  :")
   print "You entered: ", numprint "Good bye!"
```

This will produce following result:

```
Enter a number  :20You entered: 20 Enter a number  :29You entered: 29 Enter a number :3
You entered:  3


Enter a number between :Traceback (most recent call last):
File "test.py", line 5, in <module> num = raw_input("Enter a number :")
KeyboardInterrupt
```

Above example will go in an infite loop and you would need to use CTRL+C to come out of the program.

**Single Statement Suites:**

Similar to the **if** statement syntax, if your **while** clause consists only of a single statement, it may beplaced on the same line as the while header.

Here is the syntax of a one-line while clause:

```
while expression : statement
```

**Python - for Loop Statements**

A loop is a construct that causes a section of a program to be repeated a certain number of times. The repetition continues while the condition set for the loop remains true. When the condition becomes false, the loop ends and the program control is passed to the statement following the loop.

**The for Loop:**

The **for** loop in Python has the ability to iterate over the items of any sequence, such as a list or a

string.

The syntax of the loop look is:

```
for iterating_var in sequence:statements(s)
```

**Example:**

```
#!/usr/bin/python
for letter in 'Python':                    # First Exampleprint 'Current Letter :', letter
fruits = ['banana', 'apple',  'mango']
for fruit in fruits:                       # Second Exampleprint 'Current fruit :', fruit
print "Good bye!"
```

This will produce following result:Current Letter : P

Current Letter : y

Current Letter : t Current Letter : h Current Letter : o Current Letter : n Current fruit : banana Current fruit : apple Current fruit : mangoGood bye!

**Iterating by Sequence Index:**

An alternative way of iterating through each item is by index offset into the sequence itself:

**Example:**

```
#!/usr/bin/python

fruits = ['banana', 'apple',  'mango']for index in range(len(fruits)):
   print 'Current fruit :', fruits[index]print "Good bye!"
```

This will produce following result:

Current fruit : bananaCurrent fruit : apple Current fruit : mangoGood bye!

Here we took the assistance of the len() built-in function, which provides the total number of elements in thetuple as well as the range() built-in function to give us the actual sequence to iterate over.

**UNIT VII**

# Files and Input/Output

**Files – Input and Output – Errors and Exceptions – Functions – Modules – Classes and OOP – Execution Environment.**

## Printing to the Screen

The simplest way to produce output is using the *print* statement where you can pass zero or more expressions, separated by commas. This function converts the expressions you pass it to a string and writes the result to standard output as follows:

```
#!/usr/bin/python
    print "Python is really a great language,", "isn't it?";
```

This would produce following result on your standard screen:

```
Python is really a great language, isn't it?
```

## Reading Keyboard Input

Python provides two built-in functions to read a line of text from standard input, which by default comesfrom the keyboard. These functions are:

- raw_input
- input

### *The* raw_input *Function:*

The *raw_input([prompt])* function reads one line from standard input and returns it as a string (removing thetrailing newline):

```
#!/usr/bin/python
str = raw_input("Enter your input: ");print "Received input is : ", str
```

This would prompt you to enter any string and it would display same string on the screen. When I typed"Hello Python!", it output is like this:

```
Enter your input: Hello Python Received input is :  Hello Python
```

### *The* input *Function:*

The *input([prompt])* function is equivalent to raw_input, except that it assumes the input is a valid Pythonexpression and returns the evaluated result to you:

```
#!/usr/bin/python
str = i
n
put("Enter your input: ");print "Received input is : ", str
```

This would produce following result against the entered input:

Enter your input: [x*5 for x in range(2,10,2)]Recieved input is :  [10, 20, 30, 40]

**Opening and Closing Files**

Python provides basic functions and methods necessary to manipulate files by default. You can doyour most of the file manipulation using a **file** object.

**The open Function:**

Before you can read or write a file, you have to open it using Python's built-in open() function. This functioncreates 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 is paramters detail:

- **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 ie. read, write append etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r)
- **buffering:** If the buffering value is set to 0, no buffering will take place. If the buffering value is 1, line buffering will be performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action will be performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

Here is a list of the different modes of opening a file:

| **Modes** | **Description** |
|---|---|
| R | Opens a file for reading only. The file pointer is placed at the beginning of the file. This is thedefault mode. |
| Rb | Opens a file for reading only in binary format. The file pointer is placed at the beginning of thefile. This is the default mode. |
| r+ | Opens a file for both reading and writing. The file pointer will be at the beginning of the file. |
| rb+ | Opens a file for both reading and writing in binary format. The file pointer will be at the beginningof the file. |

| | |
|---|---|
| W | Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, createsa new file for writing. |
| Wb | Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file doesnot exist, creates a new file for writing. |
| w+ | Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the filedoes not exist, creates a new file for reading and writing. |
| wb+ | Opens a file for both writing and reading in binary format. Overwrites the existing file if the fileexists. If the file does not exist, creates a new file for reading and writing. |
| A | Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the fileis in the append mode. If the file does not exist, it creates a new file for writing. |
| 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 forwriting. |
| 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 readingand writing. |
| 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 newfile for reading and writing. |

**The file object attributes**

Once a file is opened and you have one file object, you can get various information related to that file. Hereis a list of all attributes related to file object:

| **Attribute** | **Description** |
|---|---|
| file.closed | Returns true if file is closed, false otherwise. |
| file.mode | Returns access mode with which file was opened. |
| file.name | Returns name of the file. |
| file.softspace | Returns false if space explicitly required with print, true otherwise. |

**Example:**

```
#!/usr/bin/python# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name print "Closed or not : ", fo.closed print "Opening mode : ",
fo.mode print "Softspace flag : ", fo.softspace
```

This would produce following result:

```
Name of the file: foo.txtClosed or not : False Opening mode : wb Softspace flag :  0
```

**The close()** ***Method:***

The close() method of a *file* object flushes any unwritten information and closes the file object, after which no more writing can be done.Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

**Syntax:**

```
fileObject.close();
```

**Example:**

```
#!/usr/bin/python# Open a file
fo = open("foo.txt", "wb")

print "Name of the file: ", fo.name
# Close opend filefo.close()
```

This would produce following result:Name of the file:  foo.txt

**Reading and Writing Files**

The file object provides a set of access methods to make our lives easier. We would see how to use read() and write() methods to read and write files.

**The write() Method:**

The write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text. The write() method does not add a newline character ('\n') to the end of the string:

**Syntax:**

```
fileObject.write(string);
```

Here passed parameter is the content to be written into the opend file.

**Example:**

```
#!/usr/bin/python# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great language.\nYeah its great!!\n");# Close opend file
fo.close()
```

The above method would create foo.txt file and would write given content in that file and finally it wouldclose that file. If you would open this file, it would have following content

```
Python is a great language.Yeah its great!!
```

**The read() Method:**

The read() method read a string from an open file. It is important to note that Python strings can havebinary data and not just text.

**Syntax:**

```
fileObject.read([count]);
```

Here passed parameter is the number of bytes to be read from the opend file. This method starts reading from the beginning of the file and if count is missing then it tries to read as much as possible, maybe until the end of file.

**Example:**

Let's take a file foo.txt which we have created above.

```
#!/usr/bin/python# Open a file
fo = open("foo.txt", "r+")str = fo.read(10);
print "Read String is : ", str# Close opend file fo.close()
```

This would produce following result:Read String is :  Python is

**File Positions**

The tell() method tells you the current position within the file in other words, the next read or write will occur at that many bytes from the beginning of the file:

The seek(offset[, from]) method changes the current file position. The offset argument indicates the number of bytes to be moved. The from argument specifies the reference position from where the bytes areto be moved.

If from is set to 0, it means use the beginning of the file as the reference position and 1 means use thecurrent position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

**Example:**

Let's take a file foo.txt which we have created above.

```
#!/usr/bin/python# Open a file
fo = open("foo.txt", "r+")str = fo.read(10);
print "Read String is : ", str# Check current position position = fo.tell();
print "Current file position : ", position
# Reposition pointer at the beginning once againposition = fo.seek(0, 0);
str = fo.read(10);
print "Again read String is : ", str# Close opend file
fo.close()
```

This would produce following result:

```
Read String is : Python is Current file position :  10 Again read String is :  Python is
```

**Renaming and Deleting Files**

Python **os** module provides methods that help you perform file-processing operations, such asrenaming and deleting files. To use this module you need to import it first and then you can all any related functions.

**The rename() Method:**

The rename() method takes two arguments, the current filename and the new filename.

**Syntax:**

```
os.rename(current_file_name, new_file_name)
```

**Example:**

Following is the example to rename an existing file test1.txt:

```
#!/usr/bin/pythonimport os
# Rename a file from test1.txt to test2.txtos.rename( "test1.txt", "test2.txt" )
```

***The* delete() *Method:***

You can use the *delete()* method to delete files by supplying the name of the file to be deleted as theargument.

***Syntax:***

os.delete(file_name)

***Example:***

Following is the example to delete an existing file *test2.txt*:

```
#!/usr/bin/pythonimport os
# Delete file test2.txtos.delete("text2.txt")
```

**Directories in Python**

All files are contained within various directories, and Python has no problem handling these too. The

**os** module has several methods that help you create, remove, and change directories.

***The* mkdir() *Method:***

You can use the *mkdir()* method of the os module to create directories in the current directory. Youneed to supply an argument to this method, which contains the name of the directory to be created.

**Syntax:**

os.mkdir("newdir")

**Example:**

Following is the example to create a directory *test* in the current directory:

```
#!/usr/bin/pythonimport os
# Create a directory "test"os.mkdir("test")
```

***The* chdir() *Method:***

You can use the *chdir()* method to change the current directory. The chdir() method takes an argument,which is the name of the directory that you want to make the current directory.

***Syntax:***

os.chdir("newdir")

**Example:**

Following is the example to go into "/home/newdir" directory:

```
#!/usr/bin/pythonimport os
# Changing a directory to "/home/newdir"os.chdir("/home/newdir")
```

***The* getcwd() *Method:***

The *getcwd()* method displays the current working directory.

***Syntax:***

```
os.getcwd()
```

***Example:***

Following is the example to give current directory:

```
#!/usr/bin/pythonimport os
# This would give location of the current directoryos.getcwd()
```

***The* rmdir() *Method:***

The *rmdir()* method deletes the directory, which is passed as an argument in the method. Before removing a directory, all the contents in it should be removed.

**Syntax:**

```
os.rmdir('dirname')
```

**Example:**

Following is the example to remove "/tmp/test" directory. It is required to give fully qualified name of thedirectory otherwise it would search for that directory in the current directory.

```
#!/usr/bin/pythonimport os
# This would  remove "/tmp/test"  directory.os.rmdir( "/tmp/test"  )
```

**File & Directory Related Methods**

There are three important sources which provide a wide range of utility methods to handle andmanipulate files & directories on Windows and Unix operating systems. They are as follows:

- File Object Methods: The *file* object provides functions to manipulate files.
- OS Object Methods.: This provides methods to process files as well as directories.

## Errors And Exceptions:

Python provides two very important features to handle any unexpected error in your

Python programsand to add debugging capabilities in them:

- **Exception Handling:** This would be covered in this tutorial.

**What is Exception?**

An exception is an event, which occurs during the execution of a program that disrupts the normal flow ofthe program's instructions.

In general, when a Python script encounters a situation that it can't cope with, it raises an exception. Anexception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise itwould terminate and come out.

**Handling an exception**

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 as elegantly as possible.

***Syntax:***

Here is simple syntax of *try....except...else* blocks:

```
try:
You do your operations here;
......................
except ExceptionI:
  If there is ExceptionI, then execute this block.except ExceptionII:
If there is ExceptionII, then execute this block.
......................
else:
If there is no exception then execute this block.
```

Here are few important points above the above mentioned syntax:

- A single try statement can have multiple except statements. This is useful when the try blockcontains 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 thecode in the try: block does not raise an exception.
- The else-block is a good place for code that does not need the try: block's protection.

**Example:**

Here is simple example which opens a file and writes the content in the file and comes out  gracefullybecause there is no problem at all:

```
#!/usr/bin/pythontry:
fh = open("testfile", "w")
   fh.write("This is my test file for exception handling!!")except IOError:
   print "Error: can\'t find file or read data"else:
print "Written content in the file successfully"fh.close()
```

This will produce following result:

```
Written content in the file successfully
```

**The *except* clause with no exceptions**

You can also use the except statement with no exceptions defined as follows:

```
try:
You do your operations here;
......................
except:
If there is any exception, then execute this block.
......................
else:
If there is no exception then execute this block.
```

This kind of a **try-except** statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice, though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.

**The *except* clause with multiple exceptions**

You can also use the same *except* statement to handle multiple exceptions as follows:

```
try:
You do your operations here;
......................
except(Exception1[, Exception2[,...ExceptionN]]]):
If there is any exception from the given exception list,then execute this block.
......................
else:
If there is no exception then execute this block.
```

**The try-finally clause**

You can use a **finally:** block along with a **try:** block. The finally block is a place to put any code that mustexecute, whether the try-block raised an exception or not. The syntax of the try-finally statement is this:

```
try:
You do your operations here;
......................
   Due to any exception, this may be skipped.finally:
This would always be executed.
......................
```

Note that you can provide except clause(s), or a finally clause, but not both. You can not use *else* clause aswell along with a finally clause.

**Example:**

```
#!/usr/bin/pythontry:
fh = open("testfile", "w")
   fh.write("This is my test file for exception handling!!")finally:
print "Error: can\'t find file or read data"
```

If you do not have permission to open the file in writing mode then this will produce

```
following result:Error: can't find file or read data
```

Same example can be written more cleanly as follows:

```
#!/usr/bin/pythontry:
fh = open("testfile", "w")try:
   fh.write("This is my test file for exception handling!!")finally:
      fh.close() except IOError:
print "Error: can\'t find file or read data"
```

When an exception is thrown in the *try* block, the execution immediately passes to the *finally* block. After allthe statements in the finally block are executed, the exception is raised again and is handled in the *except* statements if present in the next higher layer of the *try-except* statement.

**Argument of an Exception:**

An exception can have an *argument*, which is a value that gives additional information about the problem. The contents of the argument vary by exception. You capture an exception's argument by supplying a variable in the except clause as follows:

```
try:
You do your operations here;
......................
except ExceptionType, Argument:
You can print value of Argument here...
```

If you are writing the code to handle a single exception, you can have a variable follow the name of the exception in the except statement. If you are trapping multiple exceptions, you can have a variable followthe tuple of the exception.

This variable will receive the value of the exception mostly containing the cause of the exception. The variable can receive a single value or multiple values in the form of a tuple. This tuple usually contains the error string, the error number, and an error location.

**Example:**

Following is an example for a single exception:#!/usr/bin/python

```
# Define a function here.
def temp_convert(var):
try:
return int(var)
except ValueError, Argument:
print "The argument does not contain numbers\n", Argument

# Call above function here.temp_convert("xyz");
```

This would produce following result:

```
The argument does not contain numbers invalid literal for int() with base 10: 'xyz'
```

**Raising an exceptions**

You can raise exceptions in several ways by using the raise statement. The general syntax for the **raise**

statement.

**Syntax:**

```
raise [Exception [, args [, traceback]]]
```

Here *Exception* is the type of exception (for example, NameError) and *argument* is a value for the exceptionargument. 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 tracebackobject used for the exception

**Example:**

An exception can be a string, a class, or an object. Most of the exceptions that the Python core raises are classes, with an argument that is an instance of the class. Defining new exceptions is quite easy and can be done as follows:

```
def functionName( level ):
if level < 1:
raise "Invalid level!", level
# The code below to this would not be executed# if we raise the exception
```

**User-Defined Exceptions:**

Python also allows you to create your own exceptions by deriving classes from the standard built-inexceptions.

Here is an example related to *RuntimeError*. Here a class is created that is subclassed from *RuntimeError*.This is useful when you need to display more specific information when an exception is caught.

In the try block, the user-defined exception is raised and caught in the except block. The variable e is used tocreate an instance of the class Networkerror.

```
class Networkerror(RuntimeError):
def__init_(self, arg):
self.args = arg
```

So once you defined above class, you can raise your exception as follows:try:

```
  raise Networkerror("Bad hostname")except Networkerror,e:
print e.args
```

## Functions:

A function is a block of organized, reusable code that is used to perform a single, related action.

Functions provides better modularity for your application and a high degree of code reusing.

**Defining a Function:**

You can define functions to provide the required functionality. Here are simple rules to define a functionin Python:

- Function blocks begin with the keyword **def** followed by the function name and parentheses ( ( ) ).
- Any input parameters or arguments should be placed within these parentheses. You can also defineparameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of thefunction or docstring.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to thecaller. A return statement with no arguments is the same as return None.

**Syntax:**

```
def functionname( parameters ):"function_docstring" function_suite return [expression]
```

By default, parameters have a positional behavior, and you need to inform them in the same order that theywere defined.

**Example:**

Here is the simplest form of a Python function. This function takes a string as input parameter and prints iton standard screen.

```
def printme( str ):
"This prints a passed string into this function"print str
return
```

**Calling a Function:**

Defining a function only gives it a name, specifies the parameters that are to be included in the function, and structures the blocks of code.Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

Following is the example to call printme() function:

```
#!/usr/bin/python
# Function definition is heredef printme( str ):
"This prints a passed string into this function"print str;
return;
```

# Now you can call printme function printme("I'm first call to user defined function!");

printme("Again second call to the same function");This would produce following

result:

I'm first call to user defined function! Again second call to the same function

**Pass by reference vs value:**

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example:

```
#!/usr/bin/python


# Function definition is heredef changeme( mylist ):
"This changes a passed list into this function"mylist.append([1,2,3,4]);
print "Values inside the function: ", mylistreturn
# Now you can call changeme functionmylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

Here we are maintaining reference of the passed object and appending values in the same object. So thiswould produce following result:

```
Values inside the function:  [10, 20, 30, [1, 2, 3, 4]]
Values outside the function:  [10, 20, 30, [1, 2, 3, 4]]
```

There is one more example where argument is being passed by reference but inside the function, but thereference is being over-written.

```
#!/usr/bin/python
# Function definition is heredef changeme( mylist ):
"This changes a passed list into this function"
mylist = [1,2,3,4]; # This would assig new reference in mylistprint "Values inside the
function: ", mylist
return
# Now you can call changeme functionmylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

The parameter mylist is local to the function changeme. Changing mylist within the function does not affectmylist. The function accomplishes nothing and finally this would produce following result:

```
Values inside the function:  [1, 2, 3, 4]
Values outside the function:  [10, 20, 30]
```

**Function Arguments**

You can call a function by using the following types of formal arguments::

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

**Required arguments:**

Required arguments are the arguments passed to a function in correct positional order. Here the number ofarguments in the function call should match exactly with the function definition.

To call the function *printme()* you definitely need to pass one argument otherwise it would give a syntaxerror as follows:

```
#!/usr/bin/python
# Function definition is heredef printme( str ):
"This prints a passed string into this function"print str;
return;
# Now you can call printme functionprintme();
```

This would produce following result:

```
Traceback (most recent call last):
File "test.py", line 11, in <module>printme();
TypeError: printme() takes exactly 1 argument (0 given)
```

**Keyword arguments:**

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name. This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters. You can also make keyword calls to the *printme()* function in the following ways:

```
#!/usr/bin/python
```

Function definition is heredef printme( str ):

"This prints a passed string into this function"print str;

return;

# Now you can call printme functionprintme( str = "My string");

This would produce following result:My string

Following example gives more clear picture. Note, here order of the parameter does not matter:

#!/usr/bin/python

# Function definition is heredef printinfo( name, age ):

"This prints a passed info into this function"print "Name: ", name;

print "Age ", age;return;

# Now you can call printinfo functionprintinfo( age=50, name="miki" );

This would produce following result:

Name: mikiAge  50

**Default arguments:**

A default argument is an argument that assumes a default value if a value is not provided in thefunction call for that argument.

Following example gives idea on default arguments, it would print default age if it is not passed:

#!/usr/bin/python

# Function definition is here def printinfo( name, age = 35 ):

"This prints a passed info into this function"print "Name: ", name;

print "Age ", age;return;

# Now you can call printinfo functionprintinfo( age=50, name="miki" ); printinfo( name="miki" );

This would produce following result:

Name: mikiAge 50 Name: mikiAge  35

**Variable-length arguments:**

You may need to process a function for more arguments than you specified while defining the function. These arguments are called *variable-length* arguments and are not

named in the  function definition, unlike required and default arguments.
The general syntax for a function with non-keyword variable arguments is this:

```
def functionname([formal_args,] *var_args_tuple ):
"function_docstring"function_suite
return [expression]
```

An asterisk (*) is placed before the variable name that will hold the values of all nonkeyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call. For example:

```
#!/usr/bin/python
# Function definition is here def printinfo( arg1, *vartuple ):
"This prints a variable passed arguments"print "Output is: "
print arg1
for var in vartuple:print var
return;
# Now you can call printinfo functionprintinfo( 10 );
printinfo( 70, 60, 50 );
```

This would produce following result:Output is:

```
10
Output is:
70
60
50
```

**The Anonymous Functions:**

You can use the *lambda* keyword to create small anonymous functions. These functions are calledanonymous because they are not declared in the standard manner by using the *def* keyword.

- Lambda forms can take any number of arguments but return just one value in the form of anexpression. They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because lambda requires an expression.
- Lambda functions have their own local namespace and cannot access variables other than those intheir parameter list and those in the global namespace.
- Although it appears that lambda's are a one-line version of a function, they are not equivalent to*inline* statements in C or C++, whose purpose is by passing function

stack allocation during invocation for performance reasons.

**Syntax:**

The syntax of *lambda* functions contains only a single statement, which is as follows:lambda

[arg1 [,arg2,     argn]]:expression

**Example:**

Following is the example to show how *lembda* form of function works:

#!/usr/bin/python
# Function definition is here
sum = lambda arg1, arg2: arg1 + arg2;# Now you can call sum as a function print "Value of total : ", sum( 10, 20 )

print "Value of total : ", sum( 20, 20 )This would produce following result:

Value of total :  30Value of total :  40

**The return Statement**

The statement return [expression] exits a function, optionally passing back an expression to the caller. Areturn statement with no arguments is the same as return None
All the above examples are not returning any value, but if you like you can return a value from a function asfollows:

#!/usr/bin/python
# Function definition is heredef sum( arg1, arg2 ):
# Add both the parameters and return them."total = arg1 + arg2
print "Inside the function : ", totalreturn total;
# Now you can call sum functiontotal = sum( 10, 20 );

print "Outside the function : ", totalThis would produce following result:

Inside the function : 30 Outside the function :  30

**Scope of Variables**

All variables in a program may not be accessible at all locations in that program. This depends on where youhave declared a variable.
The scope of a variable determines the portion of the program where you can access a particular identifier.There are two basic scopes of variables in Python:

- Global variables
- Local variables

**Global vs. Local variables:**

Variables that are defined inside a function body have a local scope, and those defined outside have a globalscope.

This means that local variables can be accessed only inside the function in which they are declared whereasglobal variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope.

**Example:**

```
#!/usr/bin/python
total = 0; # This is global variable.# Function definition is here
def sum( arg1, arg2 ):
# Add both the parameters and return them." total = arg1 + arg2; # Here total is local variable.
print "Inside the function local total : ", total return total;
# Now you can call sum functionsum( 10, 20 );
        print "Outside the function global total : ", totalThis would produce following result:
```

```
Inside the function local total : 30 Outside the function global total :  0
```

## Modules:

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference. Simply, a module is a file consisting of Python code. A module can define functions, classes, and variables. A module can also include runnable code.

**Example:**

The Python code for a module named *aname* normally resides in a file named *aname.py*. Here's anexample of a simple module, hello.py

```
def print_func( par ):
print "Hello : ", parreturn
```

**The import Statement**

You can use any Python source file as a module by executing an import statement in some otherPython source file. *import* has the following syntax:

```
import module1[, module2[,... moduleN]
```

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. Asearch path is a list of directories that the interpreter searches before importing a module.

**Example:**

To import the module hello.py, you need to put the following command at the top of the script:

```
#!/usr/bin/python
# Import module helloimport hello
# Now you can call defined function that module as followshello.print_func("Zara")
```

This would produce following result:Hello : Zara

A module is loaded only once, regardless of the number of times it is imported. This prevents the moduleexecution from happening over and over again if multiple imports occur.

**The from...import Statement**

Python's *from* statement lets you import specific attributes from a module into the currentnamespace:

**Syntax:**

```
from modname import name1[, name2[, ... nameN]]
```

**Example:**

For example, to import the function fibonacci from the module fib, use the following statement:from fib import fibonacci

This statement does not import the entire module fib into the current namespace; it just introduces the itemfibonacci from the module fib into the global symbol table of the importing module.

**The from...import * Statement**

It is also possible to import all names from a module into the current namespace by using thefollowing import statement:

```
from modname import *
```

This provides an easy way to import all the items from a module into the current namespace; however, thisstatement should be used sparingly.

**Locating Modules**

When you import a module, the Python interpreter searches for the module in the following sequences:

- The current directory.
- If the module isn't found, Python then searches each directory in the shell variable PYTHONPATH.
- If all else fails, Python checks the default path. On UNIX, this default path is normally

/usr/local/lib/python/.

The module search path is stored in the system module sys as the **sys.path** variable. The sys.path variablecontains the current directory, PYTHONPATH, and the installation-dependent default.

### 4.10.7 The dir( ) Function

The dir() built-in function returns a sorted list of strings containing the names defined by a module.The list contains the names of all the modules, variables, and functions that are defined in a module.**Example:**

```
#!/usr/bin/python
# Import built-in module mathimport math
content = dir(math)print content;
```

This would produce following result:

```
['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp',
'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',
'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh','sqrt', 'tan', 'tanh']
```

Here the special string variable__*name*_____is the module's name, and __*file*__ is the filename from which themodule was loaded.

**The globals() and locals() Functions:**

The *globals()* and *locals()* functions can be used to return the names in the global and local namespaces depending on the location from where they are called.If locals() is called from within a function, it will return all the names that can be accessed locally from that function.

If globals() is called from within a function, it will return all the names that can be accessed globally from that function.The return type of both these functions is dictionary. Therefore, names can be extracted using the keys() function.

**The *reload()* Function:**

When the module is imported into a script, the code in the top-level portion of a module is executed only once.

Therefore, if you want to reexecute the top-level code in a module, you can use the *reload()* function. Thereload() function imports a previously imported module again.

**Syntax:**

The syntax of the reload() function is this:reload(module_name)

**Packages in Python**

A package is a hierarchical file directory structure that defines a single Python applicationenvironment that consists of modules and subpackages and sub-subpackages, and so on.

**Example:**

Consider a file *Pots.py* available in *Phone* directory. This file has following line of source code:

```
#!/usr/bin/pythondef Pots():
print "I'm Pots Phone"
```

Similar way we have another two files having different functions with the same name as above:

- *Phone/Isdn.py* file having function Isdn()
- *Phone/G3.py* file having function G3()
- Now create one more file __init__.py in *Phone* directory :
- Phone/__init__.py

To make all of your functions available when you've imported Phone, you need to put explicit importstatements in __init_.py as follows:

```
from Pots import Potsfrom Isdn import Isdnfrom G3 import G3
```

After you've added these lines to ___init__.py, you have all of these classes available when you've imported the Phone package:

```
#!/usr/bin/python
# Now import your Phone Package.import Phone

Phone.Pots()Phone.Isdn()Phone.G3()
```

This would produce following result:

```
I'm Pots Phone I'm 3G Phone I'm ISDN Phone
```

**Note:** In the above example, we have taken example of a single functions in each file, but you can keep multiple functions in your files. You can also define different Python classes in those files and then you can create your packages out of those classes.

## Classes and OOP:

Python has been an object-oriented language from day one. Because of this, creating and using classes and objects are downright easy. This chapter helps you become an expert in using Python's object- oriented programming support.

### Overview of OOP Terminology

**Class:** A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

**Class variable:** A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables aren't used as frequently as instance variables are.

**Data member:** A class variable or instance variable that holds data associated with a class and its objects.

**Function overloading:** The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects (arguments) involved.

**Instance variable:** A variable that is defined inside a method and belongs only to the current instance of a class.

**Inheritance :** The transfer of the characteristics of a class to other classes that are derived from it.

**Instance:** An individual object of a certain class. An object obj that belongs to a class Circle, forexample, is an instance of the class Circle.

**Instantiation :** The creation of an instance of a class.

**Method :** A special kind of function that is defined in a class definition.

**Object :** A unique instance of a data structure that's defined by its class. An object comprises bothdata members (class variables and instance variables) and methods.

**Operator overloading:** The assignment of more than one function to a particular operator.

## Creating Classes:

The *class* statement creates a new class definition. The name of the class immediately follows thekeyword *class* followed by a colon as follows:

```
class ClassName:
'Optional class documentation string'class_suite
```

- The class has a documentation string which can be access via *ClassName.__doc__*.
- The *class_suite* consists of all the component statements, defining class members, data attributes, andfunctions.

**Example:**

Following is the example of a simple Python class:

```
class Employee:
'Common base class for all employees'empCount = 0

def___init__(self, name, salary):
self.name = name self.salary = salary Employee.empCount += 1

def displayCount(self):
print "Total Employee %d" % Employee.empCount

def displayEmployee(self):
print "Name : ", self.name,  ", Salary: ", self.salary
```

- The variable *empCount* is a class variable whose value would be shared among all instances of a thisclass. This can be accessed as *Employee.empCount* from inside the class or outside the class.
- The first method *__init_()* is a special method which is called class constructor or initializationmethod that Python calls when you create a new instance of this class.

- You declare other class methods like normal functions with the exception that the first argument toeach method is *self*. Python adds the *self* argument to the list for you; you don't need to include it when you call the methods.

**Creating instance objects**

To create instances of a class, you call the class using class name and pass in whatever arguments its

__*init*_____method accepts.

"This would create first object of Employee class"emp1 = Employee("Zara", 2000)

"This would create second object of Employee class"emp2 = Employee("Manni", 5000)

**Accessing attributes:**

You access the object's attributes using the dot operator with object. Class variable would beaccessed using class name as follows:

emp1.displayEmployee()emp2.displayEmployee()

print "Total Employee %d" % Employee.empCountNow putting it all together:

```
#!/usr/bin/pythonclass Employee:
'Common base class for all employees'empCount = 0

def__init_(self, name, salary):self.name = name
self.salary = salary Employee.empCount += 1

def displayCount(self):
print "Total Employee %d" % Employee.empCount
def displayEmployee(self):
print "Name : ", self.name,  ", Salary: ", self.salary

"This would create first object of Employee class"emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
```

print "Total Employee %d" % Employee.empCountThis would produce following result:

Name : Zara ,Salary: 2000 Name : Manni ,Salary: 5000Total Employee 2

You can add, remove, or modify attributes of classes and objects at any time:

emp1.age = 7 # Add an 'age' attribute.emp1.age = 8  # Modify 'age' attribute.del emp1.age  # Delete 'age' attribute.

Instead of using the normal statements to access attributes, you can use following functions:

- The **getattr(obj, name[, default])** : to access the attribute of object.
- The **hasattr(obj,name)** : to check if an attribute exists or not.
- The **setattr(obj,name,value)** : to set an attribute. If attribute does not exist then it would be created.
- The **delattr(obj, name)** : to delete an attribute.

hasattr(emp1, 'age')                # Returns true if 'age' attribute existsgetattr(emp1, 'age')
                                    # Returns value of 'age' attribute setattr(emp1, 'age', 8) # Set
attribute 'age' at 8 delattr(empl, 'age') # Delete attribute 'age'

**Built-In Class Attributes**

Every Python class keeps following built-in attributes and they can be accessed using dot operator likeany other attribute:

- __**dict_:** Dictionary containing the class's namespace.
- __**doc_:** Class documentation string, or None if undefined.
- __**name__:** Class name.
- __**module_:** Module name in which the class is defined. This attribute is "_main_" in interactivemode.
- __**bases__:** A possibly empty tuple containing the base classes, in the order of their occurrence inthe base class list.

For the above class let's try to access all these attributes:

print "Employee._doc__:", Employee._doc___________ print "Employee.__name__:", Employee.__name__
print "Employee._module_:", Employee._module______ print "Employee.__bases__:", Employee.__bases__ print "Employee.__dict__:", Employee.__dict______

This would produce following result:

Employee._doc_: Common base class for all employeesEmployee._name_: Employee
Employee.__module__: __main_________ Employee._bases_: ()

```
Employee._dict__: {'__module__': '__main__', 'displayCount':
<function displayCount at 0xb7c84994>, 'empCount': 2, 'displayEmployee': <function
displayEmployee at 0xb7c8441c>,'_doc_': 'Common base class for all employees',
'__init__': <function __init__________at 0xb7c846bc>}
```

**Destroying Objects (Garbage Collection)**

Python deletes unneeded objects (built-in types or class instances) automatically to free memory space. The process by which Python periodically reclaims blocks of memory that no longer are in use istermed garbage collection.

Python's garbage collector runs during program execution and is triggered when an object's reference count reaches zero. An object's reference count changes as the number of aliases that point to it changes:

An object's reference count increases when it's assigned a new name or placed in a container (list, tuple, or dictionary). The object's reference count decreases when it's deleted with *del*, its reference is reassigned, or its reference goes out of scope. When an object's reference count reaches zero, Python collectsit automatically.

```
a = 40                    # Create object <40>
b = a                    # Increase ref. count of <40>c = [b]                # Increase ref. count  of
<40>

del a                    # Decrease ref. count  of <40> b = 100            # Decrease ref. count of
<40>c[0] = -1  # Decrease ref. count  of <40>
```

You normally won't notice when the garbage collector destroys an orphaned instance and reclaims its space. But a class can implement the special method*_del_()*, called a destructor, that is invoked when the instance is about to be destroyed. This method might be used to clean up any nonmemory resources used by an instance.

**Example:**

This___del_() destructor prints the class name of an instance that is about to be destroyed:

```
#!/usr/bin/python

class Point:
def___init( self, x=0, y=0):
```

```
self.x = xself.y = y
def___del__(self):
class_name = self._class_._name_________________ print class_name, "destroyed"

pt1 = Point()pt2 =  pt1 pt3 = pt1
print id(pt1), id(pt2), id(pt3) # prints the ids of the obejctsdel pt1
del pt2del pt3
```

This would produce following result: 3083401324 3083401324 3083401324

```
Point destroyed
```

**Class Inheritance**

Instead of starting from scratch, you can create a class by deriving it from a preexisting class bylisting the parent class in parentheses after the new class name.

The child class inherits the attributes of its parent class, and you can use those attributes as if theywere defined in the child class. A child class can also override data members and methods from the parent.

**Syntax:**

Derived classes are declared much like their parent class; however, a list of base classes to inherit from aregiven after the class name:

```
class SubClassName (ParentClass1[, ParentClass2, ...]):'Optional class documentation string'
  class_suite
```

**Example:**

```
  #!/usr/bin/python
class Parent:                  # define parent classparentAttr = 100
  def___init__(self):
  print "Calling parent constructor"

  def parentMethod(self):
      print 'Calling parent method'def setAttr(self, attr):
    Parent.parentAttr = attrdef getAttr(self):
      print "Parent attribute :", Parent.parentAttrclass Child(Parent): # define child class
```

```
def__init__(self):
   print "Calling child constructor"def childMethod(self):
print 'Calling child method'
c = Child()                    # instance of child c.childMethod()                  # child calls its
method
c.parentMethod()               # calls parent's method c.setAttr(200)              # again call
parent's methodc.getAttr()  # again call parent's method
```

This would produce following result:

```
Calling child constructorCalling child method Calling parent method Parent attribute : 200
```

Similar way you can drive a class from multiple parent classes as follows:

```
class A:                    # define your class A
.....

class B:                    # define your calss B
.....

class C(A, B):  # subclass of A and B
.....
```

You can use issubclass() or isinstance() functions to check a relationships of two classes and instances:

- The **issubclass(sub, sup)** boolean function returns true if the given subclass **sub** is indeed a subclassof the superclass **sup**.
- The **isinstance(obj, Class)** boolean function returns true if *obj* is an instance of class *Class* or is aninstance of a subclass of Class

**Overriding Methods**

You can always override your parent class methods. One reason for overriding parent's methods is becauseyou may want special or different functionality in your subclass.

**Example:**

```
#!/usr/bin/python

class Parent:                    # define parent classdef myMethod(self):
print 'Calling parent method'
```

```
class Child(Parent): # define child classdef myMethod(self):
  print 'Calling child method'
  c = Child()                    # instance of child
         c.myMethod()            # child calls overridden methodThis would produce following
```

result:

Calling child method

**Base Overloading Methods**

Following table lists some generic functionality that you can override in your own classes:

| SN | Method, Description & Sample Call | |
|---|---|---|
| 1 | **__init___( self [,args...] )** Constructor (with any optional arguments)Sample Call : *obj = className(args)* | |
| 2 | **__del__( self )** Destructor, deletes an object | Sample Call : *dell obj* |
| 3 | **__repr__( self )** Evaluatable string representation | Sample Call : *repr(obj)* |
| 4 | **__str__( self )** Printable string representation | Sample Call : *str(obj)* |
| 5 | **__cmp___( self, x )** Object comparison | Sample Call : *cmp(obj, x)* |

**Overloading Operators**

Suppose you've created a Vector class to represent two-dimensional vectors. What happens when you usethe plus operator to add them? Most likely Python will yell at you.

You could, however, define the __*add*_____method in your class to perform vector addition, and then the plusoperator would behave as per expectation:

**Example:**

```
#!/usr/bin/pythonclass Vector:
def__init_(self, a, b):
self.a = aself.b = b
def__str_(self):
return 'Vector (%d, %d)' % (self.a, self.b)

def__add_(self,other):
    return Vector(self.a + other.a, self.b + other.b)v1 = Vector(2,10)
v2 = Vector(5,-2)print v1 + v2
```

This would produce following result:Vector(7,8)

**Data Hiding**

An object's attributes may or may not be visible outside the class definition. For these cases, you canname attributes with a double underscore prefix, and those attributes will not be directly visible to outsiders:***Example:***

```
#!/usr/bin/python class JustCounter:
__secretCount = 0def count(self):
self._secretCount += 1print self._secretCount
counter = JustCounter()counter.count() counter.count()
    print counter.__secretCount This would produce following result:
```

```
1
2
Traceback (most recent call last):
File "test.py", line 12, in <module>print counter._secretCount
AttributeError: JustCounter instance has no attribute '_secretCount'
```

Python protects those members by internally changing the name to include the class name.
You can accesssuch attributes as *object._className_attrName*.
If you would replace your last line as following, then it would work for you:

```
.........................
    print counter._JustCounter_secretCountThis would produce following result:
```

```
1
2
2
```

## Execution Environment:

### Callable Objects:

A number of Python objects are what we describe as "callable," meaning any object which can be invoked with the function operator "()". The function operator is placed immediately following the name of the callable to invoke it.

### Functions

There are three types of different function objects, the first being the Python built-in functions.

### Built-in Functions (BIFs)

BIFs are generally written as extensions in C or C++, compiled into the Python interpreter, and loaded into the system as part of the first (built-in) namespace.

| BIF | Attribute Description |
|---|---|
| bif.__doc___ | documentation string |
| bif.__name___ | function name as a string |
| bif.__self___ | set to None (reserved for built-in methods) |

**User-defined Functions (UDFs)**

These are generally defined at the top-level part of a module and hence are loaded as part of theglobal namespace (once the built-in namespace has been established).

| UIF | Attribute Description |
|---|---|
| udf.__doc___ | documentation string (also udf.func_doc) |
| udf.__name___ | function name as a string (also udf.func_name) |
| udf.func_code | byte-compiled code object |
| udf.func_defaults | default argument tuple |
| udf.func_globals | global namespace dictionary; same as calling globals(x) from within function |

**lambda Expressions (Functions named "<lambda>")**

Lambda expressions are the same as user-defined functions with some minor differences. Although they yield function objects, lambda expressions are not created with the **def** statement and instead are created using the **lambda** keyword.

**Methods**

Many Python data types such as lists and dictionaries also have methods, known as built-in methods.

**Built-in Methods (BIMs)**

| BIM Attribute | Description |
|---|---|
| bim.__doc___ | documentation string |
| bim.__name___ | function name as a string |
| bim.__self___ | object the method is bound to |
| udf.func_defaults | default argument tuple |
| udf.func_globals | global namespace dictionary; same as calling globals(x) from within function |

**User-defined Methods (UDMs)**

User-defined methods are contained in class definitions and are merely "wrappers" around standard functions, applicable only to the class they are defined for.

| UDM Attribute | Description |
|---|---|
| udm.__doc___ | documentation string |
| udm.__name___ | method name as a string |
| udm.im_class | class which method is associated with |
| udm.im_func | function object for method (see UDFs) |
| udm.im_self | associated instance if bound, None if unbound |

**Classes**

The callable property of classes allows instances to be created. "Invoking" a class has theeffect of creating an instance, better known as instantiation.

Class instances

Python provides the___call_() special method for classes which allows a programmer to create objects (instances) which are callable. By default, the __call_() method is not implemented, meaning that most instances are not callable.

**Code Objects**

Code objects represent *byte-compiled* executable Python code, or *bytecode*. The difference between acode object and a function object is that the function object contains an explicit reference to the function's globals (the module in which it was defined), while a code object contains no context; also the default argument values are stored in the function object, not in the code object (because they represent values calculated at run-time). Unlike function objects, code objects are immutable and contain no references (directly or indirectly) to mutable objects.

Code objects are used by the implementation to represent ``pseudo-compiled" executable Python code such as a function body. They differ from function objects because they don't contain a reference to their global execution environment. Code objects are returned by the built-in compile() function and can be extracted from function objects through their func_code attribute.

A code object can be executed or evaluated by passing it (instead of a source string) to the exec statement or the built-in eval() function.

**Executable Object Statements and Built-in Functions**

Python provides a number of built-in functions supporting callables and executable objects, including the **exec** statement.

| **Built-in Function or Statement** | **Description** |
|---|---|
| callable(**obj**) | determines if **obj** is callable; returns 1 if so, 0 otherwise |
| compile(**string, file,type**) | creates a code object from **string** of type **type;file** is where the code originates from (usually set to ?) |
| eval(**obj, globals**=globals(),**locals**=locals()) | evaluates **obj,** which is either a expression compiled into a code object or a string expression; global and/or local namespace dictionaries may also be provided, otherwise, |
|  | the defaults for the current environment will be used |
| **exec obj** | execute **obj,** a single Python statement or set ofstatements, either in code object or string format; **obj** may also be a file object (opened to a valid Python script) |
| input(**prompt**='') | equivalent to eval(raw_input(**prompt**='')) |
| intern(**string**) | request intern of **string** |

**callable()**

callable() is a Boolean function which determines if an object type can be invoked via the function operator ( ( ) ). It returns 1 if the object is callable and 0 otherwise. Here are some sample objects and what callable returns for each type:

>>> callable(dir) # built-in function1

>>> callable(1) # integer0

>>> **def** foo(): **pass**

```
    …
>>> callable(foo) # user-defined function1
>>> callable('bar') # string0
>>> class C: pass
…
>>> callable(C) # class1
```

**Compiler()**

Compile the source into a code or AST object. Code objects can be executed by an exec statement or evaluated by a call to eval(). source can either be a string or an AST object. Refer to the ast module documentation for information on how to work with AST objects.

The filename argument should give the file from which the code was read; pass some recognizable value if it wasn't read from a file ('<string>' is commonly used).

The mode argument specifies what kind of code must be compiled; it can be 'exec' if source consists of a sequence of statements, 'eval' if it consists of a single expression, or 'single' if it consists of a single interactive statement (in the latter case, expression statements that evaluate to something other than None will be printed).

The optional arguments flags and dont_inherit control which future affect the compilation of source. If neither is present (or both are zero) the code is compiled with those future statements that are in effect in the code that is calling compile. If the flags argument is given and dont_inherit is not (or is zero) then the future statements specified by the flags argument are used in addition to those that would be used anyway. If dont_inherit is a non-zero integer then the flags argument is it – the future statements in effect around the call to compile are ignored.

Future statements are specified by bits which can be bitwise ORed together to specify multiple statements. The bit field required to specify a given feature can be found as the compiler_flag attribute on the _Feature instance in the_future_module.

This function raises SyntaxError if the compiled source is invalid, and TypeError if the source contains nullbytes.

**eval()**

The arguments are a string and optional globals and locals. If provided, globals must be a dictionary.

If provided, locals can be any mapping object.

The expression argument is parsed and evaluated as a Python expression (technically speaking, a condition list) using the globals and locals dictionaries as global and local namespace. If the globals dictionary is present and lacks '_builtins_', the current globals are copied into globals before expression is parsed. This means that expression normally has full access to the standard_builtin module and restricted environments are propagated. If the locals dictionary is omitted it defaults to the globals dictionary. If both dictionaries are omitted, the expression is executed in the environment where eval() is called. The return value is the result of the evaluated expression. Syntax errors are reported as exceptions. Example:

```
>>> x = 1
>>> print eval('x+1')2
```

This function can also be used to execute arbitrary code objects (such as those created by compile()). In this case pass a code object instead of a string. If the code object has been compiled with 'exec' as the mode argument, eval()'s return value will be None.

**UNIT VIII**

# PERL

Perl backgrounder – Perl overview – Perl parsing rules – Variables and Data – Statements and Control structures – Subroutines, Packages, and Modules- Working with Files –Data Manipulation.

## Introduction to Perl:-

### *What is Perl?*

Perl is a programming language. Perl stands for Practical Report and Extraction Language. You'll notice people refer to 'perl' and "Perl". "Perl" is the programming language as a whole whereas 'perl' is the name of the core executable. There is no language called "Perl5" -- that just means "Perl version 5". Versions of Perl prior to 5 are very old and very unsupported.

Some of Perl's many strengths are:

- **Speed of development.** You edit a text file, and just run it. You can develop programs very quickly like this. No separate compiler needed. I find Perl runs a program quicker than Java, let alone compare the complete modify-compile-run-oh-no-forgot-that-semicolon sequence.
- **Power.** Perl's regular expressions are some of the best available. You can work with objects, sockets...everything a systems administrator could want. And that's just the standard distribution.Add the wealth of modules available on CPAN and you have it all. Don't equate scripting languages with toy languages.
- **Usuability.** All that power and capability can be learnt in easy stages. If you can write a batch file you can program Perl. You don't have to learn object oriented programming, but you can write OO programs in Perl. If autoincrementing non-existent variables scares you, make perl refuse to let you. There is always more than one way to do it in Perl. You decide your style of programming, and Perl will accommodate you.
- **Portability.** On the Superhighway to the Portability Panacea, Perl's Porsche powers past Java's jaded jalopy. Many people develop Perl scripts on NT, or Win95, then just FTP them to a Unix server where they run. No modification necessary.
- **Editing tools** You don't need the latest Integrated Development Environment for Perl. You can develop Perl scripts with any text editor. Notepad, vi, MS Word 97, or even

direct off the console. Of course, you can make things easy and use one of the many freeware or shareware programmer's file editors.

- **Price.** Yes, 0 guilders, pounds, dmarks, dollars or whatever. And the peer to peer support is also free, and often far better than you'd ever get by paying some company to answer the phone and tell you to do what you just tried several times already, then look up the same reference books you already own.

***What can I do with Perl ?***

Just two popular examples :

**The Internet**

Go surf. Notice how many websites have dynamic pages with `.pl` or similar as the filename extension? That's Perl. It is the most popular language for CGI programming for many reasons, most of which are mentioned above. In fact, there are a great many more dynamic pages written with perl that may not have a `.pl` extension. If you code in Active Server Pages, then you should try using ActiveState's PerlScript. Quite frankly, coding in PerlScript rather than VBScript or JScript is like driving a car as opposed to riding a bicycle. Perl powers a good deal of the Internet.

**Systems Administration**

If you are a Unix sysadmin you'll know about sed, awk and shell scripts. Perl can do everything they can do and far more besides. Furthermore, Perl does it much more efficiently and portably. Don't take my word for it, ask around.

If you are an NT sysadmin, chances are you aren't used to programming. In which case, theadvantages of Perl may not be clear. Do you need it? Is it worth it?

A few examples of how I use Perl to ease NT sysadmin life:

- **User account creation**. If you have a text file with the user's names in it, that is all you need. Create usernames automatically, generate a unique password for each one and create the account, plus create and share the home directory, and set the permissions.
- **Event log munging.** NT has great Event Logging. Not so great Event Reading. You can use Perl to create reports on the event logs from multiple NT servers.
- **Anything else** that you would have used a batch file for, or wished that you could automatesomehow. Now you *can*.

## Perl Backgrounder :

### Versions and Naming Conventions

#### Variables

- single word: atom, chain
- multi word: atomName, centralAtomName
- constants: CALPHA_ATOM_NAME or ATOM_NAME_CALPHA, ATOM_NAME_CBETA

  Perl is not type-safe and this can cause confusion and errors. Use a limited prefix notation for such commonbasic types as array, hash, FileHandle.
- array refs ('a' prefix): aAtoms, aChains
- hash refs  ('h' prefix): hNames2Places, hChains
- FileHandle objects ('fh' prefix): fhIn, fhOut, fhPdb
- or ("ist"=input stream, "ost"=output stream): ostPdb, istMsa

#### Functions

- single word: Trim()
- multi word: OpenFilesForReading()

#### Modules (packages that are not classes)

- single word: Assert
- multi word: FileIoHelper

#### Classes

As for modules but with 'C' prefix: CStopwatch, CWindowPanel, Pdb::CResidue

#### Instance methods

- public method: plot(), getColour(), classifyHetGroups()
- private method: _plot(), _getColour(), _classifyHetGroups()
- accessor methods same as JavaBeans: getProperty(), setProperty(), isProperty()

### Perl, perl or PeRl?

There is also a certain amount of confusion regarding the capitalization of Perl. Should it be written Perl or perl? Larry Wall now uses “Perl” to signify the languageproper and “perl” to signify the implementation of the language.

### Perl History

| Version | Date | Version Details |
|---|---|---|
| Perl 0 | | Introduced Perl to Larry Wall’s office associates |
| Perl 1 | Jan 1988 | Introduced Perl to the world |
| Perl 2 | Jun 1988 | Introduced Harry Spencer’s regular expression package |

|  |  |  |  |
|---|---|---|---|
| Perl 3 | Oct 1989 | Introduced the ability to handle binary data |  |
| Perl 4 | Mar 1991 | Introduced the first “Camel” book (Programming |  |
|  |  | Perl, by Larry Wall, Tom Christiansen, and Randal L | Schwartz; |

O’Reilly & Associates). The book drove the name change, just so itcould refer to Perl 4, instead of Perl 3.

|  |  |  |
|---|---|---|
| Perl 4.036 | Feb 1993 | The last stable release of Perl 4 |
| Perl 5 | Oct 1994 | The first stable release of Perl 5, which introduced a |
|  |  | number of new features and a complete rewrite. |
| Perl 5.005_02 | Aug 1998 | The next major stable release |
| Perl 5.005_03 | Mar 1999 | The last stable release before 5.6 |
| Perl 5.6 | Mar 2000 | Introduced unified **fork** support, better threading, |
|  |  | an updated Perl compiler, and the **our** keyword |

**Main Perl Features:**

**a) Perl Is Free**

Perl’s source code is open and free anybody can download the C source that constitutes a Perl interpreter. Furthermore, you can easily extend the core functionality of Perl both within the realms of the interpreted language and by modifying the Perl source code.

**b) Perl Is Simple to Learn, Concise, and Easy to Read:**

It has a syntax similar to C and shell script, among others, but with a less restrictive format. Most programs are quicker to write in Perl because of its use of built-in functions and a huge standard and contributed library. Most programs are also quicker to execute than other languages because of Perl’sinternal architecture.

**c) Perl Is Fast :**

Compared to most scripting languages, this makes execution almost as fast as compiled C code. But, because the code is still interpreted, there is no compilation process, and applications can be written and edited much faster than with other languages, without any of the performance problems normally associated with an interpreted language.

**d) Perl Is Extensible:**

You can write Perl-based packages and modules that extend the functionality of the language. You can also call external C code directly from Perl to extend the functionality.

**e) Perl Has Flexible Data Types:**

You can create simple variables that contain text or numbers, and Perl will treat the variable data accordingly at the time it is used.

f) **Perl Is Object Oriented:**

Perl supports all of the object-oriented features—inheritance, polymorphism, and encapsulation.

g) **Perl Is Collaborative:**

There is a huge network of Perl programmers worldwide. Most programmers supply,and use, the modules and scripts available via CPAN, the Comprehensive Perl Archive Network

**<u>Compiler or Interpreter:</u>**

a) **Compiler**

A program that decodes instructions written in a higher order language and produces an assemblylanguage program.

A compiler that generates machine language for a different type of computer than the one thecompiler is running in.

b) **Interpreter**

In computing, an interpreter is a computer program that reads the source code of another computerprogram and executes that program.

A program that translates and executes source language statements one line at a time.

c) **Difference between Compiler and Interpreter**

A compiler first takes in the entire program, checks for errors, compiles it and then executes it.

Whereas, an interpreter does this line by line, so it takes one line, checks it for errors and then executes it.

Example of Compiler - Java Example of Interpreter – PHP

d) **Perl is interpreter or compiler?**

Neither, and both. Perl is a scripting language. There is a tool, called perl, intended to run programswritten in the perl language.

"Compiled" languages are ones like C and C++, where you have to take the source code, compile itinto an executable file, and THEN run it.

"Interpreted" languages, like Perl, PHP, and Ruby, are ones which do NOT require pre-compiling. They are generally compiled on-the-fly (which is what the perl command-line tool does) into opcodes, and then run. So, Perl is an interpreted language because a tool reads the source code and immediately runs it. Perl is a compiler because it has to compile that source code before it can be run while it's being interpreted.

**<u>Popular “Myth conceptions”</u>**

i) **It’s only for the Web**

Probably the most famous of the myths is that Perl is a language used, designed, and created exclusively for developing web-based applications.

ii) **It’s Not Maintenance Friendly**

Any good (or bad) programmer will tell you that anybody can write unmaintainable code in anylanguage.
Many companies and individuals write maintainable programs using Perl.

iii) **It’s Only for Hackers**

Perl is used by a variety of companies, organizations, and individuals. Everybody fromprogramming beginners through “hackers” up to multinational corporations use Perl to solve their problems.

iv) **It’s a Scripting Language**

In Perl, there is no difference between a script and program. Many large programs and projects have been written entirely in Perl.

v) **There’s No Support**

The Perl community is one of the largest on the Internet, and you should be able to findsomeone, somewhere, who can answer your questions or help you with your problems.

vi) **All Perl Programs Are Free**

Although you generally write and use Perl programs in their native source form, this does not mean that everything you write is free. Perl programs are your own intellectual property and can be bought, sold, and licensed just like any other program.

vii) **There’s No Development Environment**

Perl programs are text based, you can use any source-code revision-control system. The most popularsolution is CVS, or Concurrent Versioning System, which is now supported under Unix, MacOS and Windows.

viii) **Perl Is a GNU Project**

While the GNU project includes Perl in its distributions, there is no such thing as “GNU Perl.” Perl isnot produced or maintained by GNU and the Free Software Foundation. Perl is also made available on a much more open license than the GNU Public License.

ix) **Perl Is Difficult to Learn**

Because Perl is similar to a number of different languages, it is not only easy to learn but also easy tocontinue learning. Its structure and format is very similar to C, **awk**, shell script, and, to a greater or lesser extent, even BASIC.

Perl Overview:

**Installing and using Perl**

Perl was developed by Larry Wall. It started out as a scripting language to supplement rn, the USENET reader. It available on virtually every computer platform.

Perl is an interpreted language that is optimized for string manipulation, I/O, and system tasks. It has built in for most of the functions in section 2 of the UNIX manuals -- very popular with sys administrators. it incorporates syntax elements from the Bourne shell, csh, awk, sed, grep, and C. It provides a quick and effective way to write interactive web applications

**Writing a Perl Script**

Perl scripts are just text files, so in order to actually "write" the script, all you need to do is create a text file using your favorite text editor. Once you've written the script, you tell Perl to execute the text fileyou created.

Under Unix, you would use

```
$ perl myscript.pl
```

and the same works under Windows:C:\> perl myscript.pl

Under Mac OS, you need to drag and drop the file onto the MacPerl application. Perl scripts have a .pl extension, even under Mac OS and Unix.

**Perl Under Unix**

The easiest way to install Perl modules on Unix is to use the CPAN module. For example:

```
shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

The DBD::mysql installation runs a number of tests. These tests attempt to connect to the local MySQL server using the default user name and password. (The default user name is your login name on Unix, and ODBC on Windows. The default password is "no password.") If you cannot connect to the server with those values (for example, if your account has a password), the tests fail. You can use force install DBD::mysql to ignore the failed tests.

DBI requires the Data::Dumper module. It may be installed; if not, you should install it before installing DBI.

**Perl Under Windows**

1. Log on to the Web server computer as an administrator.
2. Download the ActivePerl installer from the following ActiveState Web site: http://www.activestate.com/ (http://www.activestate.com/)
3. Double-click the **ActivePerl** installer.
4. After the installer confirms the version of ActivePerl that it is going to be installed, click **Next**.
5. If you agree with the terms of the license agreement, click **I accept the terms in the license agreement**,and then click **Next**. Click **Cancel** if you do not accept the license agreement. If you do so, you cannot continue the installation.
6. To install the whole ActivePerl distribution package (this step is recommended), click **Next** to continue the installation.The software is installed in the default location (typically C:\Perl).
7. To customize the individual components or to change the installation folder, follow the instructions that appear on the screen.
8. When you are prompted to confirm the addition features that you want to configure during the installation, click any of the following settings, and then click **Next**:

- **Add Perl to the PATH environment variable**: Click this setting if you want to use Perl in a command prompt without requiring the full path to the Perl interpreter.
- **Create Perl file extension association**: Click this setting if you want to allow Perl scripts to be automatically run when you use a file that has the Perl file name extension (.pl) as a command name.
- **Create IIS script mapping for Perl**: Click this setting to configure IIS to identify Perl scripts as executable CGI programs according to their file name extension.
- **Create IIS script mapping for Perl ISAPI**: Click this setting to use Perl scripts as an ISAPI filter.

9. Click **Install** to start the installation process.
10. After the installation has completed, click **Finish**.

**Perl Components :**

**Variables**

Perl Variables with the techniques of handling them are an important part of the Perl language. As a language-type script, Perl was designed to handle huge amounts of data text. Working with variables isfairly straightforward given that it is not necessary to define and allocate them, so no sophisticated techniques for the release of memory occupied by them.

As general information, to note that the names of Perl variables contain alphabetic characters, numbers and the underscore (_) character and are case sensitive.

A specific language feature is that variables have a non-alphabetical prefix that fashion somewhatcryptic the language.
scalar variables – starting with $array variables – starting with @
hashes or associative arrays indicated by %
The $, @ and % characters actually predefine the variable type in Perl. Perl language also offers some built-in predefined variables that facilitate and shorten the programming code.

**Operators**

The operators work with numbers and strings and manipulate data objects called operands. We foundthe operators in expressions which we need to evaluate.

**Statements**

The statements are one of the most important topics in the Perl language, actually for any programming language. We use statements in order to process or evaluate the expressions. Perl uses the values returned by statements to evaluate or process other statements.

A Perl statement ends with the semicolon character (;) which is used to tell interpreter that the statement was complete.

**Subroutines (Functions)**

**Definition: Subroutine (Function) is** a block of source code which does one or some tasks with specifiedpurpose.

**Advantages:**

i) It reduces the Complexity in a program by reducing the code.
ii) It also reduces the Time to run a program.In other way,It’s directly proportional to Complexity.
iii) It’s easy to find-out the errors due to the blocks made as function definition outside the mainfunction.

**Modules:**

A **Perl module** is a discrete component of software for the Perl programming language. Technically, it is a particular set of conventions for using Perl's package mechanismthat has become universally adopted.

A module defines its source code to be in a *package* (much like a Java package), the Perl mechanism for defining namespaces, e.g. *CGI* or *Net::FTP* or *XML::Parser*; the file structure mirrors the namespace structure (e.g. the source code for *Net::FTP* is in *Net/FTP.pm*). A collection of modules, with accompanying documentation, build scripts, and usually a test suite, compose a **distribution**.

## Perl Parsing Rules

**The Execution Process:**

The execution process of perl contains the following steps

- It takes raw input,
- Parses each statement and converts it into a series of opcodes,
- Builds a suitable opcode tree,
    - Executes the opcodes within a Perl "virtual machine."It classifies only two stages
- The parsing stage and the
- Execution or run-time stage

**Syntax and Parsing Rules:**

The Perl parser thinks about all of the following when it looks at a source line:

- **Basic syntax** The core layout, line termination, and so on
- **Comments** If a comment is included, ignore it
- **Component identity** Individual terms (variables, functions and numericaland textual constants) are identified
- **Bare words** Character strings that are not identified as valid terms
- **Precedence** Once the individual items are identified, the parser processes the statements according to the precedence rules, which apply to all operatorsand terms
- **Context** What is the context of the statement, are we expecting a list or scalar,a number or a string, and so on. This actually happens during the evaluation of individual elements of a line, which is why we can nest functions such as sort, reverse, and keys into a single statement line
- **Logic Syntax** For logic operations, the parser must treat different values,whether constant- or variable-based, as true or false values

All of these present some fairly basic and fundamental rules about how Perl looks atan entire script.

**Basic Syntax**

The basic rules govern such things as line termination and the treatment of whitespace. These basic rules are

- Lines must start with a token that does not expect a left operand
- Lines must be terminated with a semicolon, except when it's the last line of a

  block, where the semicolon can be omitted.
- White space is only required between tokens that would otherwise be confusing,so spaces, tabs, newlines, and comments (which Perl treats as white space) are ignored. The line
  sub menu{print"menu"}
  works as it would if it were more neatly spaced.
- Lines may be split at any point, providing the split is logically between twotokens.

  **Component Identity**

  When Perl fails to identify an item as one of the predefined operators, it treats the character sequence as a "term." Terms are core parts of the Perl language and include variables, functions, and quotes. The term-recognition system uses these rules:

- Variables can start with a letter, number, or underscore, providing they followa suitable variable character, such as $, @, or %.
- Variables that start with a letter or underscore can contain any furthercombination of letters, numbers, and underscore characters.
- Variables that start with a number can only consist of further numbers—bewary of using variable names starting with digits. The variables such as $0 through to $9 are used for group matches in regular expressions.
- Subroutines can only start with an underscore or letter, but can then contain any combination of letters, numbers, and underscore characters.
- Case is significant—$VAR, $Var, and $var are all different variables.
- Each of the three main variable types have their own name space—$var, @var,and %var are all separate variables.
- Filehandles should use all uppercase characters—this is only a convention, nota rule, but it is useful for identification purposes.

## <u>**Operators and Precedence8.3.4.1 Operators:**</u>

### <u>**a) Arithmetic Operators:**</u>

The following are the arithmetic operators in Perl.

| Operator | Description |
|---|---|
| + | Addition operator |
| - | Subtraction operator |
| * | Multiplication operator |
| / | Division operator |
| % | Modulus operator |
| ** | Exponentiation operator |

The operators +, -, *, / take two operands and return the sum, difference, product and quotient respectively. Perl does an floating point division not an integral division. To get the integral quotient one has to use int()function. Say if you divide "int(5/2)" the result will be 2, to get the exact result use the code below. **b)Assignment Operator**

| Operator | Function |
|---|---|
| = | Normal Assignment |
| += | Add and Assign |
| -= | Subtract and Assign |
| *= | Multiply and Assign |
| /= | Divide and Assign |
| %= | Modulus and Assign |
| **= | Exponent and Assign |

Everyone knows how to use the assignment operator (=). There are other operators, when used with "="gives a different result.

**c) Increment/Decrement Operators**

The following are the auto increment, decrement operators in Perl.

| Operator | Description |
|---|---|
| ++ | Autoincrement operator |
| -- | Autodecrement operator |

The usage of auto increment operators are same as in C Language. In prefix decrement/increment first thevalue is increased or decreased then the new value is returned eg: "++$a", "--$a".

The vice versa of the above, is post decrement/increment operators. First the old value is returned thenincremented or decremented to give the result. eg: "$a++", "$a--"

**d) Comparison Operator**

| Operator | | Function |
|---|---|---|
| == | eq | Equal to Operator |
| != | ne | Not Equal to Operator |
| < | lt | Less than Operator |

| | | |
|---|---|---|
| > | gt | Greater than Operator |
| <= | le | Lessthan or Equal to Operator |
| >= | ge | Greater than or Equal to Operator |

**e) Logical Operators**

The following are the Logical Operators used in Perl.

| Operator | | Function |
|---|---|---|
| && | and | logical conjunction of the two surrounding expressions |
| \|\| | or | logical disjunction of the two surrounding expressions |
| ! | not | returns the logical negation of the expression to its right |

**f) Bitwise Operators**

| Operator | Function |
|---|---|
| << | Binary shift left |
| >> | Binary shift right |
| & | Bitwise AND |
| \| | Bitwise OR |
| ^ | Bitwise XOR |
| ~ | Bitwise NOT |

The basic criterias for the bitwise shift operators is that the operands should be numerals, but these arerepresented as binary internally.

**g)Bitwise Shift Right & Left:**

The bitwise shift right operator shifts the specified bits to the right or left. First we convert both operands tobinary then shift bits to right or left.The binary digits equivalent decimal is the result.

## <u>8.3.4.2 Precedence</u>

**a) Operator Precedence**

The following table displays the Operator Precedence in Perl.

| **Associativity** | **Operators** |
|---|---|
| left | Terms and list operators (leftward) |
| left | -> |
| nonassoc | ++ -- |
| right | ** |
| right | ! ~ n + - (unary) |
| left | =~ !~ |

| | |
|---|---|
| left | * / % x |
| left | + - . |
| left | << >> |
| nonassoc | named unary operators |
| nonassoc | < > <= >= lt gt le ge |
| nonassoc | == != <=> eq ne cmp |
| left | & |
| left | \|^ |
| left | && |
| left | \|\| |
| nonassoc | .. ... |
| right | ?: |
| right | = += -= *= etc. (assignment operators) |
| left | , => |
| nonassoc | List operators (rightward) |
| right | not |
| left | and |
| left | or xor |

**b) Contexts**

Perl supports a number of different contexts,

**c) Scalar and List Context**

There are two basic contexts scalar and list. These two contexts affect the operation of the function oroperator concerned by implying the accepted value, or value returned. For example:

$size = @list;sort @list;

**d) Numerical and String Context**

In a numerical context, this variable returns the numerical error number of the last error that occurred, and in a string context, the associated message. The interpreter uses this context as the basis for the conversion of values into the internal integer, floating point, or string values that the scalar value is divided into.

**e) Boolean Context**

**Boolean context is where an expression is used solely to identify a true or false value. See the "Logical Values" section, later in the chapter, to see how Perl treats individual values and constants in a logical context.**

1) Void Context

2) Interpolative Context

3) Logical Values

| **Value** | **Logical Value** |
|---|---|
| Negative | number True |
| 0 | False |
| Positive number | True |
| Empty string | False |
| Non-empty string | True |
| Undefined value | False |
| Empty list | False |
| List with at least one element | True |

## Variables and Data:

Perl has three built in variable types:Scalar

ArrayHash

**Basic Naming Rules**

The basic rules that apply to the naming of variables within Perl:

- Variable names can start with a letter, a number, or an underscore, although they normally begin with a letter and can then be composed of any combination of letters, numbers, and the underscore character.
- Variables can start with a number, but they must be entirely composed of that number; for example,

  $123 is valid, but $1var is not.

- Variable names that start with anything other than a letter, digit, or underscore are generally reservedfor special use by Perl (see “Special Variables” later in this chapter).
- Variable names are case sensitive; $foo, $FOO, and $fOo are all separate variables as far as Perl isconcerned.
- As an unwritten (and therefore unenforced) rule, names all in uppercase are constants.
- All scalar values start with $, including those accessed from an array of hash, for example $array[0]or $hash{key}.
- All array values start with @, including arrays or hashes accessed in slices, for example@array[3..5,7,9] or @hash{‘bob’, ‘alice’}.
- All hashes start with %.
- Namespaces are separate for each variable type—the variables $var, @var, and %var are all differentvariables in their own right.

- In situations where a variable name might get confused with other data (such as when embeddedwithin a string), you can use braces to quote the name. For example, ${name}, or %{hash}.

**Scalar Variables:**

A scalar represents a single value as follows:

```
my $animal = "camel"; my $answer = 42;
```

A scalar values can be strings, integers or floating point numbers, and Perl will automatically convert between them as required. There is no need to pre-declare your variable types. Scalar values can be used in various ways:

```
print $animal;
print "The animal is $animal\n";
print "The square of $answer is ", $answer * $answer, "\n";
```

There are a number of "magic" scalars with names that look like punctuation or line noise. These special variables are used for all kinds of purposes and they wioll be discussed in Special Variables sections. The only one you need to know about for now is $_ which is the "default variable".

```
print;      # prints contents of $_ by default
```

**Literals :**

Literal is a value that is represented "as is" or hard-coded in your source code. When you see the fourcharacters 45.5 in programs it really refers to a value of forty-five and a half. Perl uses four types of literals. Here is a quick glimpse at them:

Numbers - This is the most basic data type.

Strings - A string is a series of characters that are handled as one unit.

Arrays - An array is a series of numbers and strings handled as a unit. You can also think of an arrayas a list.

Associative Arrays - This is the most complicated data type. Think of it as a list in which every valuehas an associated lookup item

**Numeric Literals:**

Numeric literals are simply constant numbers. Numeric literals are much easier to comprehend and use than string literals. There are only a few basic ways to express numeric literals.

**a) String Literals i)Single-Quoted-Strings:**

Single quoted are a sequence of characters that begin and end with a single quote. These quotes are not a part of the string they just mark the beginning and end for the Perl interpreter. If you want a ' inside of your string you need to preclude it with a \ like this \' as you'll see below.

| | |
|---|---|
| 'four' | #has four letters in the string |
| 'can\'t' | #has five characters and represents "can't" |
| 'hi\there' | #has eight characters and represents"hi\\there" (one \ in the string) |
| 'blah\\blah' | #has nine characters and represents "blah\\blah" (one\ in the string) |

**ii)Double-Quoted-Strings:**

Double quoted strings act more like strings in C or C++ the backslash allows you to represent control characters. Another nice feature Double-Quoted strings offers is variable interpolation this substitutes the value of a variable into the string.

**Some of the things you can put in a Double-Quoted String**

| Representation | What it Means |
|---|---|
| \a | Bell |
| \b | Backspace |
| \e | Escape |
| \f | Formfeed |
| \n | Newline |
| \r | Return |
| \t | Tab |
| \\ | Backslash |
| \" | Double quote |
| \007 | octal ascii value this time 007 or the bell |
| \x07 | hex ascii value this time 007 or the bell |
| \cD | any control character.. here it is control-D |
| \l | lowercase next letter |
| \u | uppercase next letter |
| \L | lowercase all letters until \E |
| \U | uppercase all letters until \E |
| \Q | Backslash quote all nonletters and nonnumbers until \E |
| \E | Stop \U \L or \Q |

**b) Quotes:**

It is important to note that DOUBLE quotes " and SINGLE quotes ' can have a very different effectin PERL. Single quotes are literal. They will pass on exactly what is inside of them. Conents are not analyzed for variables.

**c) Interpolation of Array Values :**

When you embed an array into a string, the elements of the array are included in order separated by the contents of the $” special variable, which by default is a space:

```
@list = ('hello', 'world');
print "@list\n"; # Outputs 'hello world'
```

Perl will determine whether the name you have supplied is correct, and it’ll raise an error if you’ve tried to interpolate an array that doesn’t exist. This can lead to problems:

```
print "mc@mcslp.com";
```

In this instance, Perl is expecting to find an array called @mcslp, and it will obviouslyfail because we haven’t defined such an array. Generally, Perl will warn you of this error during compilation and tell you to escape the @ sign:

```
print "mc\@mcslp.com@;
```

**i) Arrays:**

An array is just a set of scalars. It's made up of a list of individual scalars that are stored within a singlevariable. You can refer to each scalar within that list using a numerical index.

**ii) Array Creation:**

Array variables are prefixed with the @ sign and are populated using either parentheses or the qwoperator. For example:

```
@array = (1, 2, 'Hello'); @array = qw/This is an array/;
```

The second line uses the qw// operator, which returns a list of strings, separating the delimited string bywhite space.

```
@days = qw/MondayTuesday
...
Sunday/;
```

We can also populate an array by assigning each value individually:

```
$array[0] = 'Monday';
...
$array[6] = 'Sunday';
```

**iii) Extracting Individual Indices:**

When extracting individual elements from an array, you must prefix the variable with a dollar sign and thenappend the element index within square brackets after the name. For example:

```
#!/usr/bin/perl

@shortdays = qw/Mon Tue Wed Thu Fri Sat Sun/;print $shortdays[1];
```

This will printTue

Array indices start at zero, so in the preceding example we.ve actually printed "Tue". You can also give anegative index.in which case you select the element from the end, rather than the beginning, of the array.
This means that

```
print $shortdays[0]; # Outputs Monprint $shortdays[6]; # Outputs Sun
print $shortdays[-1]; # Also outputs Sunprint $shortdays[-7]; # Outputs Mon
```

**iv) Sequential Number Arrays:**

PERL offers a shortcut for sequential numbers and letters. Rather than typing out each element whencounting to 100 for example, we can do something like this:

```
#!/usr/bin/perl @10 = (1 .. 10);
@100 = (1 .. 100;


@1000 = (100 .. 1000);
@abc = (a .. z);
print "@10";  # Prints number starting from 1 to 10 print "@100"; # Prints number starting from 1 to 100 print "@1000"; # Prints number starting from 1 to 1000print "@abc";  # Prints number starting from a to z
```

**v) Array Size:**

The size of an array can be determined using scalar context on the array - the returned value will be thenumber of elements in the array:

```
@array = (1,2,3);
print "Size: ",scalar @array,"\n";
```

The value returned will always be the physical size of the array, not the number of valid elements. You candemonstrate this, and the difference between scalar @array and $#array, using this fragment:

```
#!/uer/bin/perl @array = (1,2,3);
$array[50] = 4;
print "Size: ",scalar @array,"\n"; print "Max Index: ", $#array,"\n";This will return
Size: 51
Max Index: 50
```

There are only four elements in the array that contain information, but the array is 51 elements long, with ahighest index of 50.

**vi) Adding and Removing Elements in Array:**

Use the following functions to add/remove and elements:

- **push():** adds an element to the end of an array.
- **unshift():** adds an element to the beginning of an array.
- **pop():** removes the last element of an array.
- **shift() :** removes the first element of an array.

When adding elements using push() or shift() you must specify two arguments, first the array name and second the name of the element to add. Removing an element with pop() or shift() only requires that you send the array as an argument.

```
#!/usr/bin/perl

# Define an array
@coins = ("Quarter","Dime","Nickel");print "First Statement : @coins";
print "\n";
# Add one element at the end of the array
push(@coins, "Penny");
print "Second Statement : @coins";print "\n";
# Add one element at the beginning of the array
unshift(@coins, "Dollar");
print "Third Statement : @coins";print "\n";

# Remove one element from the last of the array.
pop(@coins);
```

print "Fourth Statement : @coins";print "\n";

# Remove one element from the beginning of the array.

**shift**(@coins);

print "Fifth Statement : @coins";print "@coins";

Now this will produce following resultFirst Statement : Quarter Dime Nickel

Second Statement : Quarter Dime Nickel Penny Third Statement : Dollar Quarter Dime Nickel PennyFourth Statement : Dollar Quarter Dime Nickel

Fifth Statement : Quarter Dime Nickel

**vii) Slicing Array Elements:**

You can also extract a "slice" from an array - that is, you can select more than one item from an array inorder to produce another array.

@weekdays = @shortdays[0,1,2,3,4];

The specification for a slice must a list of valid indices, either positive or negative, each separated by acomma. For speed, you can also use the **..** range operator:

@weekdays = @shortdays[0..4];Ranges also work in lists:

@weekdays = @shortdays[0..2,6,7];

**viii) Replacing Array Elements:**

Replacing elements is possible with the splice() function. Splice() requires a handful of arguments and theformula reads:

*splice(@array,first-element,sequential_length, new elements)*

Essentially, you send PERL an array to splice, then direct it to the starting element, count through how manyelements to replace, and then fill in the missing elements with new information.

#!/usr/bin/perl @nums = (1..20);

splice(@nums, 5,5,21..25);print "@nums";

Here actual replacement begins after the 5th element, starting with the number 6. Five elements are thenreplaced from 6-10 with the numbers 21-25

**ix) Transform Strings to Arrays:**

With the split function, it is possible to transform a string into an array. To do this simply define an array and set it equal to a split function. The split function requires two arguments, first the character of which tosplit and also the string variable.

```
#!/usr/bin/perl
# Define Strings
$astring = "Rain-Drops-On-Roses-And-Whiskers-On-Kittens";
$namelist = "Larry,David,Roger,Ken,Michael,Tom";
# Strings are now arrays. Here '-' and ',' works as delimeter@array = split('-',$astring);
@names = split(',',$namelist);
print $array[3]; # This will print Rosesprint "\n";      # This is a new line
print $names[4];  # This will print Michael
```

Sorting Arrays:

The sort() function sorts each element of an array according to ASCII Numeric standards.

```
#!/usr/bin/perl

# Define an array
@foods = qw(pizza steak chicken burgers);print "Before sorting: @foods\n";
# Sort this  array @foods = sort(@foods);
print "After sorting: @foods\n"; This will produce following result
Before sorting: pizza steak chicken burgersAfter sorting: burgers chicken pizza steak
```

**c) The $[ Special Variable:-**

**$[** is a special variable. This particular variable is a scalar containing the first index of all arrays.  because Perl arrays have zero-based indexing, $[ will almost always be 0. But if you set $[ to 1 then all your arrays will use on-based indexing. It is recommended not to use any other indexing

**d) Hashes:-**

Hashes are an advanced form of array. One of the limitations of an array is that the information contained within it can be difficult to get to. For example, imagine that you have a list of people and their ages.

The hash solves this problem very neatly by allowing us to access that @ages array not by an index, but by a scalar key. For example to use age of different people we can use thier names as key to define a hash.

```
%ages = ('Martin' => 28,'Sharon' => 35,
'Rikke' => 29,);

print "Rikke is $ages{Rikke} years old\n";This will produce following result
Rikke is 29 years old
```

## e) Creation of Hash:-

Hashes are created in one of two ways. In the first, you assign a value to a named key on a one-by-one basis:

```
$ages{Martin} = 28;
```

In the second case, you use a list, which is converted by taking individual pairs from the list: the firstelement of the pair is used as the key, and the second, as the value. For example,

```
%hash = ('Fred' , 'Flintstone', 'Barney', 'Rubble');
```

For clarity, you can use => as an alias for , to indicate the key/value pairs:

```
%hash = ('Fred' => 'Flintstone','Barney' => 'Rubble');
```

### i) Sorting/Ordering Hashes:

There is no way to simply guarantee that the order in which a list of keys, values, or key/value pairs willalways be the same. In fact, it's best not even to rely on the order between two sequential evaluations:

```
#!/usr/bin/perl
print(join(', ',keys %hash),"\n");
print(join(', ',keys %hash),"\n");
```

If you want to guarantee the order, use **sort**, as, for example:

```
print(join(', ',sort keys %hash),"\n");
```

If you are accessing a hash a number of times and want to use the same order, consider creating a single array to hold the sorted sequence, and then use the array (which will remain in sorted order) to iterate overthe hash. For example:

```
my @sortorder = sort keys %hash;foreach my $key (@sortorder)
```

### ii) Hash Size

You get the size - that is, the number of elements - from a hash by using scalar context on either keys orvalues:

```
#!/usr/bin/perl

%ages = ('Martin' => 28, 'Sharon' => 35, 'Rikke' => 29);print "Hash size: ",scalar keys
%ages,"\n";
```

This will produce following resultHash size: 3

**iii) Add & Remove Elements in Hashes:**

Adding a new key/value pair can be done with one line of code using simple assignment operator. But toremove an element from the hash you need to use **delete** function.

#!/usr/bin/perl
%ages = ('Martin' => 28, 'Sharon' => 35, 'Rikke' => 29);# Add one more element in the hash

$age{'John'} = 40;

# Remove one element from the hashdelete( $age{'Sharon'} );

**f) Lists :**

other than zero.

**i) The Lists:**

Lists are really a special type of array - .essentially, a list is a temporary construct that holds a series ofvalues. The list can be "hand" generated using parentheses and the comma operator,

@array = (1,2,3);

or it can be the value returned by a function or variable when evaluated in list context:print

join(',' @array);

Here, the @array is being evaluated in list context because the join function is expecting a list.

**ii) Merging Lists (or Arrays)**

Because a list is just a comma-separated sequence of values, you can combine lists together:

@numbers = (1,3,(4,5,6));

The embedded list just becomes part of the main list.this also means that we can combine

arrays together:@numbers = (@odd,@even);

Functions that return lists can also be embedded to produce a single, final list:@numbers =

(primes(),squares());

**iii) Selecting Elements from Lists:**

The list notation is identical to that for arrays - .you can extract an element from an array by appendingsquare brackets to the list and giving one or more indices:

#!/usr/bin/perl

$one = (5,4,3,2,1)[4];

print "Value of \$one is $one\n"

This will produce follwoing resultValue of $one is 1

Similarly, we can extract slices, although without the requirement for a leading @ character:

#!/usr/bin/perl
@newlist = (5,4,3,2,1)[1..3];
print "value of new list is @newlist\n";This will produce follwoing result value of new list is 4 3 2

**iv) Arrays in List Context:**

When accessing an entire array or slice, arrays work as lists—that is@array = (1,2);
($a, $b) = @array;is equivalent to ($a, $b) = (1, 2);

**g) Hashes in List Context:-**

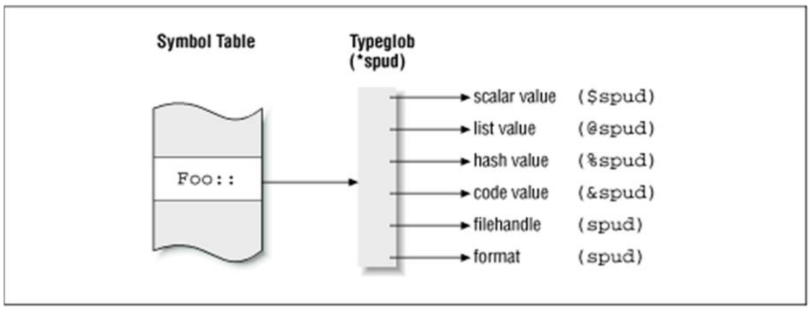
In the same way that hashes are essentially populated using a list, if you evaluate ahash in list context, then what you get is a list of key/value pairs. For example,
my %hash = (Fred => 'Flintstone', Barney => 'Rubble');@list = %hash;
print join(', ',@list);
produces
Barney, Rubble, Fred, Flintstone

**Typeglobs:**

Typeglobs are immensely useful. They allow us to efficiently create aliases of symbols, which isthe basis for a very important module called Exporter that is used in a large number of freely available modules

Typeglobs can also be aliased to ordinary references in such a way that you don't have to use thedereferencing syntax; this is not only easier on the eye, it is faster too.

Closely related to typeglobs and symbol tables is the subject of dynamic versus lexical scoping(using local versus my ). There are a couple of useful idioms that arise from these differences.



## Symbol table and typeglobs

### I) The defined Function and the Undefined Value

**undef:**

Undefines the value of EXPR. Use on a scalar, list, hash, function, or typeglob. Use on a hash with astatement such as undef $hash{$key}; actually sets the value of the specified key to an undefined value. If you want to delete the element from the hash, use the delete function.

***Syntax***

undef EXPRundef

***Example***

```
#!/usr/bin/perl -w

$scalar = 10;
@arrary = (1,2);
print "1 - Value of Scalar is $scalar\n";print "1 - Value of Array is @array\n";undef( $scalar );
undef( @array );
print "2 - Value of Scalar is $scalar\n";print "2 - Value of Array is @array\n";
```

It will produce following results:

```
1       - Value of Scalar is 101 - Value of Array is
Use of uninitialized value in concatenation (.) or
string at test.pl line 13.
2       - Value of Scalar is2 - Value of Array is
```

**defined:**

Returns true if **EXPR** has a value other than the undef value, or checks the value of $_ if EXPR is not specified. This can be used with many functions to detect a failure in operation, since they return undef if there was a problem. A simple Boolean test does not differentiate between false, zero, an empty string, orthe string .0., which are all equally false.

If **EXPR** is a function or function reference, then it returns true if the function has been defined. When used with entire arrays and hashes, it will not always produce intuitive results. If a hash element isspecified, it returns true if the corresponding value has been defined, but does not determine whether thespecified key exists in the hash.

**Return Value**

- 0 if EXPR contains undef
- 1 if EXPR contains a valid value or reference

***Syntax***

defined EXPRdefined

**Example**

```
#!/usr/bin/perl

$var1 = "This is defined";if( defined($var1) ){
print "$var1\n";
}
if( defined($var2) ){
print "var2 is also defined\n";
}else{

print "var2 is not defined\n";
}
```

This will produce following result

```
This is defined var2 is not defined
```

**Default Values:**

It's not necessary within Perl to initialize variables with some default values. Perl automatically creates all scalars as empty (with the **undefined** value). Lists and hashes are automatically created empty

**Other Tokens**

| Token | Value |
|---|---|
| **_ _LINE_ _** | The current line number within the current file. |
| **_ _FILE_ _** | The name of the current file. |

**__PACKAGE_____**

**__END_____**

**__DATA_____**

**The name of the current package. If there is no currentpackage, then it returns the undefined value.**

Indicates the end of the script (or interpretable Perl) within afile before the physical end of file.

As for **__END_**, except that it also indicates the start of the **DATA** filehandle that can be opened with the **open**, thereforeallowing you to embed script and data into the same script.

## Statements and Control Structures:

### Code Blocks:

A sequence of statements is called a code block, or simply just a block. A block of code could be anyof the code examples that we have seen thus far. The only difference is, to make them a block, we would surround them with {}.

```
use strict;
{
my $var; Statement;Statement;Statement;
}
```

Anything that looks like that is a block. Blocks are very simple, and are much like code blocks in languages like C, C++, and Java. However, in Perl, code blocks are decoupled from any particular control structure. The above code example is a valid piece of Perl code that can appear just about anywhere in a Perlprogram. Of course, it is only particularly useful for those functions and structures that use blocks.

### Conditional Statements:

The conditional statements are **if** and **unless**, and they allow you to control the execution of your script.There are five different formats for the **if** statement:

```
if (EXPR)
if (EXPR) BLOCK
if (EXPR) BLOCK else BLOCK
if (EXPR) BLOCK elsif (EXPR) BLOCK ...
if (EXPR) BLOCK elsif (EXPR) BLOCK ... else BLOCKSTATEMENT if (EXPR)
```

The first format is classed as a simple statement, since it can be used at the end of another statement withoutrequiring a block, as in:

```
print "Happy Birthday!\n" if ($date == $today);
```

In this instance, the message will only be printed if the expression evaluates to a true value. The second format is the more familiar conditional statement that you may have come across in otherlanguages:

```
if ($date == $today)
{
print "Happy Birthday!\n";
}
```

This produces the same result as the previous example.

The third format allows for exceptions. If the expression evaluates to true, then the first block is executed;otherwise (else), the second block is executed:

```
if ($date == $today)
{
print "Happy Birthday!\n";
}
else
{
print "Happy Unbirthday!\n";
}
```

The fourth form allows for additional tests if the first expression does not return true. The elsif can berepeated an infinite number of times to test as many different alternatives as are required:

```
if ($date == $today)
{
print "Happy Birthday!\n";
}
elsif ($date == $christmas)
{
print "Happy Christmas!\n";
}
```

The fifth form allows for both additional tests and a final exception if all the other tests fail:

```
if ($date == $today)
{
print "Happy Birthday!\n";
}
elsif ($date == $christmas)
{
print "Happy Christmas!\n";
}else
```

```
{
print "Happy Unbirthday!\n";
}
```

The unless statement automatically implies the logical opposite of **if**, so unless the **EXPR** is true, executethe block. This means that the statement

```
print "Happy Unbirthday!\n" unless ($date == $today);is equivalent to
print "Happy Unbirthday!\n" if ($date != $today);
```

For example, the following is a less elegant solution to the preceding if...else. Although it achieves the sameresult, example:

```
unless ($date != $today)

{
print "Happy Unbirthday!\n";
}
else
{
print "Happy Birthday!\n";
}
```

The final conditional statement is actually an operator.the conditional operator. It is synonymous with theif...else conditional statement but is shorter and more compact. The format for the operator is:

(expression) ? (statement if true) : (statement if false) For example, we can emulate the previous example as follows:

```
($date == $today) ? print "Happy B.Day!\n" : print "Happy Day!\n";
```

**Loops:**

Perl supports four main loop types:

1. while
2. for
3. until
4. foreach

In each case, the execution of the loop continues until the evaluation of the supplied expression changes.

- In the case of a **while** loop execution continues while the expression evaluates to true.

- The until loop executes while the loop expression is false and only stops when the expressionevaluates to a true value.
- The list forms of the **for** and **foreach** loop are special cases. They continue until the end of thesupplied list is reached.

- **while Loops**

The while loop has three forms:while EXPRLABEL

while (EXPR) BLOCKLABEL

while (EXPR) BLOCK continue BLOCK

In first form, the expression is evaluated first, and then the statement to which it applies is evaluated. For example, the following line increases the value of $linecount as long as we continue to read lines from a given file:

```
$linecount++ while ();
```

To create a loop that executes statements first, and then tests an expression, you need to combine while witha preceding **do {}** statement. For example:

```
Do
{
$calc += ($fact*$ivalue);
} while $calc <100;
```

In this case, the code block is executed first, and the conditional expression is only evaluated at the end ofeach loop iteration.

The second two forms of the while loop repeatedly execute the code block as long as the result from theconditional expression is true. For example, you could rewrite the preceding example as:

```
while($calc < 100)
{
$calc += ($fact*$ivalue);
}
```

- **until Loops**

The inverse of the while loop is the until loop, which evaluates the conditional expression and reiterates overthe loop only when the expression returns false. Once the expression returns true, the loop ends.

In the case of a do.until loop, the conditional expression is only evaluated at the end of the code block. In anuntil (EXPR) BLOCK loop, the expression is evaluated before the block executes. Using an until loop, you could rewrite the previous example as:

```
Do
{
$calc += ($fact*$ivalue);
} until $calc >= 100;This is equivalent to
```

```
do
{
$calc += ($fact*$ivalue);
} while $calc <100;
```

- **for Loops**

A for loop is basically a while loop with an additional expression used to reevaluate the original conditionalexpression. The basic format is:

LABEL for (EXPR; EXPR; EXPR) BLOCK

The first EXPR is the initialization - the value of the variables before the loop starts iterating. The second isthe expression to be executed for each iteration of the loop as a test. The third expression is executed for each iteration and should be a modifier for the loop variables. Thus, you can write a loop to iterate 100 times like this:

```
for ($i=0;$i<100;$i++)
{
...
}
```

You can place multiple variables into the expressions using the standard list operator (the comma):for ($i=0, $j=0;$i<100;$i++,$j++)

You can create an infinite loop like this:

```
for(;;)
{
...
}
```

- **foreach Loops**

he last loop type is the foreach loop, which has a format like this:LABEL foreach VAR (LIST) BLOCK

LABEL foreach VAR (LIST) BLOCK continue BLOCKUsing a for loop, you can iterate through the list using:

```
for ($index=0;$index<=@months;$index++)
{
print "$months[$index]\n";
}
```

This is messy, because you.re manually selecting the individual elements from the array and using an additional variable, $index, to extract the information. Using a foreach loop, you can simplify the process:

```
foreach (@months)
{
print "$_\n";
}
```

The foreach loop can even be used to iterate through a hash, providing you return the list of values or keysfrom the hash as the list:

```
foreach $key (keys %monthstonum)
{
print "Month $monthstonum{$key} is $key\n";
}
```

**The continue Block:**

The **continue** block is executed immediately after the main block and is primarily used as a methodfor executing a given statement (or statements) for each iteration, irrespective of how the current iteration terminated. It is somehow equivalent to for loop

```
{
my $i = 0; while ($i <100)
{ ... }
continue
{
```

```
$i++;
}
}
```

This is equivalent to

```
for (my $i = 0; $i < 100; $i++)
{ ... }
```

**Labels:**

Labels can be applied to any block, but they make the most sense on loops. By giving your loop a name, you allow the loop control keywords to specify which loop their operation should be applied to. The format for alabeled loop is:

```
LABEL: loop (EXPR) BLOCK ...
```

For example, to label a for loop:

```
ITERATE: for (my $i=1; $i<100; $i++)
{
print "Count: $i\n";
}
```

**Loop Control:**

There are three loop control keywords: next, last, and redo.

The **next** keyword skips the remainder of the code block, forcing the loop to proceed to the next value in theloop. For example:

```
while (<DATA>)
{
next if /^#/;
}
```

Above code would skip lines from the file if they started with a hash symbol. If there is a **continue** block, itis executed before execution proceeds to the next iteration of the loop.

The **last** keyword ends the loop entirely, skipping the remaining statements in the code block, as well as dropping out of the loop. The last keyword is therefore identical to the break keyword in C and Shellscript.For example:

```
while ()
{
```

```
last if ($found);
}
```

Would exit the loop if the value of $found was true, whether the end of the file had actually been reached ornot. The **continue** block is not executed.

The **redo** keyword reexecutes the code block without reevaluating the conditional statement for the loop.This skips the remainder of the code block and also the **continue** block before the main code block is reexecuted. For example, the following code would read the next line from a file if the current line terminates with a backslash:

```
while(<DATA>)
{

if (s#\\$#)
{
$_ .= <DATA>;
redo;
}
}
```

Here is an example showing how labels are used in inner and outer loops

```
OUTER:
while(<DATA>)
{
chomp; @linearray=split;
foreach $word (@linearray)
{
next OUTER if ($word =~ /next/i)
}
}
```

- **goto**

There are three basic forms: **goto LABEL, goto EXPR,** and **goto &NAME**. In each case, execution ismoved from the current location to the destination.

In the case of goto **LABEL**, execution stops at the current point and resumes at the point of the labelspecified.

The **goto &NAME** statement is more complex. It allows you to replace the currently executing subroutinewith a call to the specified subroutine instead.

## Subroutines, Packages, and Modules

### Subroutines:

### a) Functions:

**Definition:**

Function is a block of source code which does one or some tasks with specified purpose.

**Advantages:**

It reduces the Complexity in a program by reducing the code.

It also reduces the Time to run a program.In other way,It’s directly proportional to Complexity.

It’s easy to find-out the errors due to the blocks made as function definition outside the main function.Subroutines, like variables, can be declared and defined.

Various forms:

```
sub NAME
sub NAME PROTOsub NAME ATTRS
sub NAME PROTO ATTRS
```

where as

PROTO – Prototype, ATTRS – AttributesExamples:

```
sub message
{
print "Hello!\n";
}
```

To call the subroutine and get a result,add(1,2);

### b) Arguments:

In perl we do not declare arguments a function takes while we defining a function, we still may passarguments to a Perl function. All arguments are passed into a function through the special array @_. Thus, we can send as many arguments as we want and also function may receive different number of arguments. Example 1:

simple function adds two numbers and prints the result:

```
sub add
{
$result = $_[0] + $_[1];
print "The result was: $result\n";
}
```

To call the subroutine and get a result,add(1,2);

Example 2:submy_args
{
my $i;
print "My arguments:\n";for($i=0;$i<=$#_;$i++){

print "Argument $i: $_[$i]\n";
}
print "\n";
}
my $name = "arg one";
my @names = ('This', 'is', 'an', 'array');

my_args($name); my_args($name, 123);
my_args(123, 345, @names, $name);
Counting Arguments:

If you want to count the number of arguments that you have received, just access the@_ in a scalar context:

Example:

my $count = @_;

**c)Return Values:**

A subroutine is always part of some expression. The value of the subroutine invocation is called thereturn value . The return value of a subroutine is the value of the return statement or of the last expression evaluated in the subroutine.

For example, let's define this subroutine:subsum_of_a_and_b

{

return $a + $b;

}

The last expression evaluated in the body of this is the sum of $a and $b , so the sum of $a and $b will be thereturn value. Here's that in action:

$a = 3;

$b = 4;

$c = sum_of_a_and_b();# $c gets 7

$d = 3 * sum_of_a_and_b();# $d gets 21

A subroutine can also return a list of values when evaluated in a list context. Consider this subroutine andinvocation:

sublist_of_a_and_b

```
{
return($a,$b);
}
$a = 5;
$b = 6;
@c = list_of_a_and_b();# @c gets (5,6)
```

The last expression evaluated really means the last expression evaluated, rather than the last expressiondefined in the body of the subroutine. For example, this subroutine returns $a if $a >0 ; otherwise it returns $b :

**d) Error Notification:**

The easiest way to report an error within a subroutine is to return the undefined valueundef function.

**i) undef:**

Undefines the value of EXPR. Use on a scalar, list, hash, function, or typeglob. Use on a hash with a statement such as undef $hash{$key}; actually sets the value of the specified key to an undefined value. Ifyou want to delete the element from the hash, use the delete function.

**Syntax**

```
undef EXPRundef
```

**Example**

```
#!/usr/bin/perl -w

$scalar = 10;
@arrary = (1,2);

print "1 - Value of Scalar is $scalar\n";print "1 - Value of Array is @array\n";

undef( $scalar );undef( @array );

print "2 - Value of Scalar is $scalar\n";print "2 - Value of Array is @array\n";
```

It will produce following results:

```
- Value of Scalar is 101 - Value of Array is
Use of uninitialized value in concatenation (.) orstring at test.pl line 13.
1            - Value of Scalar is2 - Value of Array is
```

**e) Context :**

If you write a Perl program, you refer to lists with an "@" symbol in front of the list variable name. Butdepending on the context of how you write it, the program may interpret it as

- The list contents (if the operation is normal for a list).
- The length of the list (i.e. the element count) if that's the only thing that makes sense.
- A space separated string with all the items in the list joined together (if it's in double quotes).

**For example:**

```
@salad = ("apple","banana","cherry");
$salad[3] = "tomato";
$salad[8] = "fig";
print @salad."\n"; # scalar context
print "@salad.\n"; # double quote context
```

**f) will display:**

```
applebananacherrytomatofig9
apple banana cherry tomato          fig.
```

**g) Attributes:**

Attributes allow you to add extra semantics to any Perl subroutine or variable.Currently Perl supports onlythree attributes: locked, method, and lvalue

**i) The locked Attribute:**

# Only one thread is allowed into this function.subafunc : locked { ... }

# Only one thread is allowed into this function on a given object.subafunc : locked method { ... }

Setting the locked attribute is meaningful only when the subroutine or method is intended to be called by multiple threads simultaneously. When set on a nonmethod subroutine, Perl ensures that a lock is acquired on the subroutine itself before that subroutine is entered. When set on a method subroutine (that is,one also marked with the method attribute), Perl ensures that any invocation of it implicitly locks its firstargument (the object) before execution.

Semantics of this lock are the same as using the lock operator on the subroutine as the first statementin that routine.

**ii) The Method attribute:**

subafunc : method { ... }

Currently this has only the effect of marking the subroutine so as not to trigger the "Ambiguous callresolved as CORE::%s" warning. (We may make it mean more someday.)

The attribute system is user-extensible, letting you create your own attribute names. These new attributes must be valid as simple identifier names (without any punctuation other than the "_" character). They may have a parameter list appended, which is currently only checked for whether its parentheses nestproperly.

Here are examples of valid syntax (even though the attributes are unknown):subfnord (&\%) : switch(10,foo(7,3)) : expensive;

subplugh () : Ugly('\(') :Bad;subxyzzy : _5x5 { ... }

Here are examples of invalid syntax:

subfnord : switch(10,foo(); # ()-string not balanced subsnoid : Ugly('('); # ()-string not balanced subxyzzy : 5x5; # "5x5" not a valid identifier

subplugh : Y2::north; # "Y2::north" not a simple identifiersubsnurt : foo + bar; # "+" not a colon or space

The attribute list is passed as a list of constant strings to the code that associates them with the subroutine. Exactly how this works (or doesn't) is highly experimental. Check attributes(3) for currentdetails on attribute lists and their manipulation.

**iii) The lvalue Attribute**

Using lvalue, a subroutine can be used as a modifiablescalar value. For example, you can do this:mysub() = 5;

This is particularly useful in situations where you want to use a method on an object toaccept a setting,instead of setting the value on the object directly.

For example:

submysub : lvalue

{

$val;

}

**h) Prototypes:**

Perl's prototypes are not the prototypes in other languages. Other languages use prototypes to define,name and type the arguments to a subroutine. Perl prototypes are about altering the context in which the subroutine's arguments are evaluated, instead of the normal list context. Perl prototypes should **not** be used merely as a check that the right arguments have been passed in. For that use a module such as **I)Params::Validate.**

The prototype of a function can be queried by calling prototype(). Prefix built-in functions with"CORE::". prototype("CORE::abs").

The prototype function Returns a string containing the prototype of the function or referencespecified by EXPR, or undef if the function has no prototype.

A subroutine's prototype is defined in its declaration, and enforces constraints on the interpretation ofarguments passed to it. For example:

```
$func_prototype = prototype ( "myprint" ); print "myprint prototype is $func_prototype\n";
submyprint($$){
print "This is test\n";
}
```

It will produce following results:

```
Myprint prototype is $$
```

**j) Traps:**

- Prototypes do not work on methods.
- Prototypes do not work on function references.
- Calling a function with a leading & (&foo) disables prototypes. But you shouldn't be callingfunctions like that.

**<u>Packages:</u>**

A package is a collection of code which lives in its own namespace. A namespace is a named collectionof 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 module’s own namespace.

**a) The Package Statement:**

package statement switches the current naming context to a specified namespace (symboltable). If the named package does not exists, a new namespace is first created.

```
$i = 1; print "$i\n"; # Prints "1"package foo;
$i = 2; print "$i\n"; # Prints "2"package main;
print "$i\n"; # Prints "1"
```

The package stays in effect until either another package statement is invoked, or until the endof the end of the current block or file.

You can explicitly refer to variables within a package using the :: package qualifier

**b) Package Symbol Tables :**

The symbol table is the list of active symbols (functions variables objects) within a package. Eachpackage has its own symbol table and with some exceptions all the identifiers starting with letters or underscores are stored within the corresponding symbol table for each package. This means that all otheridentifiers including all of the special punctuation-only variables such as **$_** are stored within the **main** package

The symbol table for a package can be accessed as a hash. For example, the main package's symbol table can be accessed as %main:: or, more simply, as %::. Likewise, symbol tables for other packages are %MyMathLib::. The format is hierarchical, so that symbol tables can be traversed using standard Perl code. For example

```
foreach $symname (sort keys %main::)
{
local *symbol = $main::{$symname};
print "\$$symname is defined\n" if defined $symbol; print "\@$symname is defined\n" if
defined @symbol;print "\%$symname is defined\n" if defined %symbol;
}
```

You can also use the symbol table to define static scalars by assigning a value toatype glob:
*C = 299792458;

**c) Special Blocks BEGIN and END Blocks**

You may define any number of code blocks named BEGIN and END which act as constructors anddestructors respectively.

```
BEGIN { ... }
END { ... }
BEGIN { ... }
END { ... }
```

Every BEGIN block is executed after the perl script is loaded and compiled but before any otherstatement 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.

**<u>Modules:</u>**

Modules are the loadable libraries of the Perl world. 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 on the end). A Perl module is aself-contained piece of [Perl] code that can be used by a

Perl program.

A Perl module file called "Foo.pm" might contain statements like this.

```
#!/usr/bin/perlpackage Foo; sub bar
{
print "Hello $_[0]\n"
}
sub blat {
print "World $_[0]\n"
}
1;
```

**a) <u>Few noteable points about modules</u>**

- The functions require and use will load a module.
- Both use the list of search paths in @INC to find the module (you may modify it!)
- Both call the eval function to process the code
- The 1; at the bottom causes eval to evaluate to TRUE (and thus not fail)

**b) <u>Creating Modules:</u>**

There are two ways for a module to make its interface available to your program: by exporting symbols or by allowing method calls. We'll show you an example of the first style here; the second style isfor object-oriented modules and is described in the next chapter. (Object-oriented modules should export nothing, since the whole idea of methods is that Perl finds them for you automatically, based on the type ofthe object.)

To construct a module called Bestiary, create a file called *Bestiary.pm* that looks like this:

```
package                    Bestiary;require      Exporter;


our @ISA                        = qw(Exporter);
our @EXPORT                       = qw(camel);    # Symbols to be exported by default
our @EXPORT_OK = qw($weight);  # Symbols to be exported on requestour $VERSION   =
1.00;                                         # Version number


### Include your variables and functions here
```

sub camel { print "One-hump dromedary" }
$weight = 1024;1;

A program can now say use Bestiary to be able to access the camel function (but not the $weightvariable), and use Bestiary qw(camel $weight) to access both the function and the variable.

**c) <u>The Exporter Module</u>**

The **Exporter** module supplies the **import** function required by the **use** statement to import functions. Export allows to export the functions and variables of modules to user’s namespace using thestandard import method. This way, we don’t need to create the objects for the modules to access it’s members.
@EXPORT and @EXPORT_OK are the two main variables used during export operation.

@EXPORT contains list of symbols (subroutines and variables) of the module to be exported into the callernamespace.
@EXPORT_OK does export of symbols on demand basis.

**Example:**

Let us use the following sample program to understand Perl exporter.package Arithmetic;
use Exporter;
# base class of this(Arithmetic) module@ISA = qw(Exporter);
# Exporting the add and subtract routine@EXPORT = qw(add subtract);
# Exporting the multiply and divide routine on demand basis.@EXPORT_OK = qw(multiply divide);
sub add
{
my ($no1,$no2) = @_;my $result;
$result = $no1+$no2;return $result;
}

**d) <u>Comparing use and require:</u>**

When you import a module, you can use one of two keywords: **use** or **require.Use :**

1. The method is used only for the modules(only to include .pm type file)
2. The included objects are varified at the time of compilation.
3. No Need to give file extension.

**Syntax:**

use Module;and

use Module LIST;

**Example:**

use MyMathLib qw/add square/;

**Require:**

1. The method is used for both libraries and modules.
2. The included objects are varified at the run time.
3. Need to give file Extension.

**Syntax:**

require Module;

**Example:**

require 'Fcntl.pl';require 5.003;

- **no:**

If MODULE supports it, then **no** calls the **unimport** function defined in MODULE to unimport allsymbols from the current package, or only the symbols referred to by LIST.

It can be said that **no** is opposite of **import.**

***Return Value***

Nothing

***Syntax***

no Module VERSION LISTno Module VERSION

no MODULE LISTno MODULE

- **do:**

- When supplied a block, do executes as if BLOCK were a function, returning the value of the laststatement evaluated in the block.
- When supplied with EXPR, do executes the file specified by EXPR as if it were another Perl script.
- If supplied a subroutine, SUB, do executes the subroutine using LIST as the arguments, raising anexception if SUB hasn.t been defined

***Syntax***

do BLOCK

do EXPR

do SUB(LIST)

**Example**

Following are the usage...eval `cat stat.pl`;
is equivalent to
do 'stat.pl';

**e) Scope:**

Scope refers to the visibility of variables. In other words, which parts of your program can see or useit. Normally, every variable has a global scope. Once defined, every part of your program can access a variable.
It is very useful to be able to limit a variable's scope to a single function. In other words, the variable wilhave a limited scope. This way, changes inside the function can't affect the main program in unexpectedways, func7.pl

```
firstSub("AAAAA", "BBBBB");

sub firstSub{

local ($firstVar) = $_[0]; my($secondVar)    = $_[1]; print("firstSub: firstVar  = $firstVar\n");

print("firstSub: secondVar = $secondVar\n\n");secondSub();

print("firstSub: firstVar = $firstVar\n"); print("firstSub: secondVar = $secondVar\n\n");

}

sub secondSub{

print("secondSub: firstVar = $firstVar\n"); print("secondSub: secondVar = $secondVar\n\n");

$firstVar  = "ccccC";

$secondVar = "DDDDD"; print("secondSub: firstVar  = $firstVar\n");

print("secondSub: secondVar = $secondVar\n\n");

}
```

This program prints:

```
firstSub: firstVar = AAAAA firstSub: secondVar = BBBBB secondSub: firstVar  = AAAAA

Use of uninitialized value at test.pl line 19.secondSub: secondVar =

secondSub: firstVar = ccccC secondSub: secondVar = DDDDDfirstSub: firstVar  = ccccC

firstSub: secondVar = BBBBB
```

The output from this example shows that secondSub() could not access the

$secondVar variable thatwas created with my() inside firstSub(). Perl even prints out an error message that warns about the uninitialized value. The $firstVar variable, however, can be accessed and valued by secondSub().

**Effects of my:**

Declares the variables in LIST to be lexically scoped within the enclosing block. If more than one variable isspecified, all variables must be enclosed in parentheses

**Return Value**

Nothing

***Syntax***

```
my LIST
```

**Example**

```
#!/usr/bin/perl -w
my $string = "We are the world";print "$string\n";
myfunction(); print "$string\n";
sub myfunction
{
my $string = "We are the function";print "$string\n";
mysub();
}
sub mysub
{
print "$string\n";
}
```

**It will produce following results:**

```
We are the world We are the functionWe are the world We are the world
```

**Effects of local:**

Sets the variables in LIST to be local to the current execution block. If more than one value is specified, you must use parentheses to define the list. Note that local creates a local copy of a variable,which then goes out of scope when the enclosing block terminates. The localized value is then used whenever it is accessed, including any subroutines and formats used during that block.

***Return Value***

Nothing

***Syntax***

local LIST

***Example***

```
#!/usr/bin/perl -w

local $foo;                    # make $foo dynamically local
local (@wid, %get);            # make list of variables local
local $foo = "flurp";          # make $foo dynamic, and init it
local @oof = @bar;             # make @oof dynamic, and init it
```

**Effects of our:**

Defines the variables specified in LIST as being global within the enclosing block, file, or eval statement. It is effectively the opposite of my.it declares a variable to be global within the entire scope, rather thancreating a new private variable of the same name. All other options are identical to my;

An our declaration declares a global variable that will be visible across its entire lexical scope, even acrosspackage boundaries. The package in which the variable is entered is determined at the point of the declaration, not at the point of use.

If more than one value is listed, the list must be placed in parentheses.

**Return Value**

nothing

***Syntax***

our EXPR

our EXPR TYPE our EXPR : ATTRS

our TYPE EXPR : ATTRS

**Example**

Try out following example:#!/usr/bin/perl -w

```
our $string = "We are the world";print "$string\n";
myfunction(); print "$string\n";
sub myfunction
{
our $string = "We are the function";print "$string\n";
}
```

It will produce following results:We are the world

We are the functionWe are the function

**f) Scope Within Loops:**

There is a slightly special case when declaring a variable in a loop statement. In the fragment,

```
foreach my $key (sort keys %hash)
{
...
}
```

the $key variable is lexically defined for the duration of the entire statement, which means that it's accessible within the block that makes up the loop (including any continue block in a while or other statement), but it immediately disappears when the loop terminates.

However, be careful where you define the variable. In the fragment,do {

```
my $var = 1;
} while ($var);
```

the $var used in the test has no value—only the $var in the block has a value.

**g) Autoloading:**

Autoloading is a way to intercept calls to undefined methods. An autoload routine may choose tocreate a new function on the fly, either loaded from disk or perhaps just eval()ed right there.The AUTOLOAD subroutine is called with all the same arguments as the unknown routine, and the fully qualified subroutine name is placed into the $AUTOLOAD variable.

Normally, you can't call a subroutine that isn't defined. However, if there is a subroutine namedAUTOLOAD in the undefined subroutine's package.

The AUTOLOAD module can also be used directly within a Perl script to add blanket functionality to ascript without requiring you to create many subroutines.

**Example:**

```
BEGIN
{
$constants{"PI"} = 3.141592654;
}
use subs keys %constants; print "The value of PI is ",PI;sub AUTOLOAD
{
```

```
my $constant = $AUTOLOAD;
$constant =~ s/.*:://;
return $constants{"$constant"};
    }
```

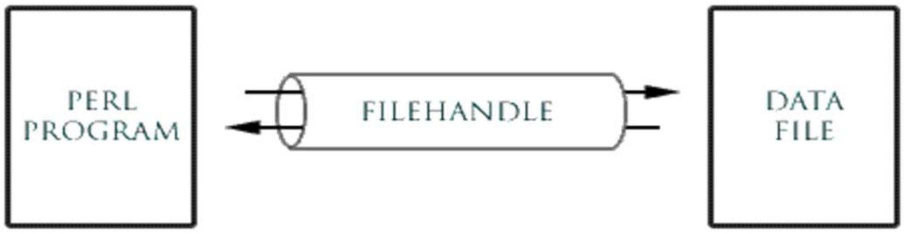
## Working with Files:

**Filehandles:**

The basics of handling files are simple: you associate a filehandle with an external entity (usually afile) and then use a variety of operators and functions within Perl to read and update the data stored withinthe data stream associated with the filehandle.

A filehandle is a named internal Perl structure that associates a physical file with a name. All filehandles are capable of read/write access, so you can read from and update any file or device associatedwith a filehandle. However, when you associate a filehandle, you can specify the mode in which the filehandle is opened.

Three basic file handles are - **STDIN**, **STDOUT**, and **STDERR**.The general scheme looks like this:



**Opening and Closing Files:**

There are following two functions with multiple forms which can be used to open any new orexisting file in Perl.

open FILEHANDLE, EXPRopen FILEHANDLE
sysopen FILEHANDLE, FILENAME, MODE, PERMSsysopen FILEHANDLE, FILENAME, MODE

Here FILEHANDLE is the file handle returned by open function and EXPR is the expression having filename and mode of opening the file.

Following is the syntax to open file.txt in read-only mode. Here less than <signe indicates that filehas 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 fileand will print its content over the screen.

```
#!/usr/bin/perl open(DATA, "<file.txt");while(<DATA>)
{
print "$_";
}
```

**Open Function**

Following is the syntax to open file.txt in writing mode. Here less than >signe indicates that file hasto be opend in writing mode

open(DATA, “>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”);

To truncate the file first:

open DATA, "+>file.txt" or die “Couldn't open file file.txt, $!”;

You can open a file in append mode. In this mode writing point will be set to the end of the

fileopen(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 possible values of different modes

| **Entities** | **Definition** |
|---|---|
| < 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 andO_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 ittakes 0x666.
Following is the table which gives possible values of MODE

| **Value** | **Definition** |
|---|---|
| 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 |

**Close Function**

To close a filehandle, and therefore disassociate the filehandle from the corresponding file, you use the closefunction. This flushes the filehandle's buffers and closes the system's file descriptor.

```
close FILEHANDLEclose
```

If no FILEHANDLE is specified, then it closes the currently selected filehandle. It returns true only if itcould successfully flush the buffers and close the file.

```
close(DATA) || die "Couldn'tclose file properly";
```

**Reading and Writing File handles :**

Once you have an open filehandle, you need to be able to read and write information. There are a number ofdifferent 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 ascalar 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 specifiedfilehandle. 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 isspecified:

```
getc FILEHANDLE
getc
```

If there was an error, or the filehandle is at end of file, then undef is returned instead.

**read Function**

The read function reads a block of information from the buffered filehandle: This function is used to readbinary data from the file.

```
read FILEHANDLE, SCALAR, LENGTH, OFFSETread 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 thenumber of bytes read on success, zero at end of file, or undef if there was an error.

**print Function**

For all the different methods used for reading information from filehandles, the main function for writinginformation back is the print function.

```
print FILEHANDLE LIST
print LISTprint
```

The print function prints the evaluated value of LIST to FILEHANDLE, or to the current output filehandle(STDOUT by default). For example:

```
print "Hello World!\n";
```

**Copying Files**

Here is the example which opens an existing file file1.txt and read it line by line and generate another copyfile2.txt

```
#!/usr/bin/perl
# Open file to read open(DATA1, "<file1.txt");# Open new file to write open(DATA2,
">file2.txt");
# Copy data from one file to another.while(<DATA1>)
{
print DATA2 $_;
}
close( DATA1 );close( DATA2 );
```

**Renaming a file**

Here is an example which shows how we can rename a file file1.txt to file2.txt. Assuming file is available in

/usr/test directory.

```
#!/usr/bin/perl
rename ("/usr/test/file1.txt", "/usr/test/file2.txt" );
```

This function rename takes two arguments and it just rename existing file

**Deleting an exiting file**

Here is an example which shows how to delete a file file1.txt using unlink function.

```
#!/usr/bin/perl
unlink ("/usr/test/file1.txt");
```

**Locating Your Position Within a File:**

You can use to tell function to know the current position of a file and seek function to point a particularposition inside the file.

**tell Function**

The first requirement is to find your position within a file, which you do using the tell function:tell FILEHANDLE

tell

This returns the position of the file pointer, in bytes, within FILEHANDLE if specified, or the currentdefault selected filehandle if none is specified.

**seek Function**

The seek function positions the file pointer to the specified number of bytes within a file:seek

FILEHANDLE, POSITION, WHENCE

The function uses the fseek system function, and you have the same ability to position relative to three different points: the start, the end, and the current position. You do this by specifying a value for WHENCE.Zero sets the positioning relative to the start of the file. For example, the line sets the file pointer to the 256th byte in the file.

seek DATA, 256, 0;

**Miscellaneous Control Functions:**

**binmode:**

Sets the format for FILEHANDLE to be read from and written to as binary on the operating systems that differentiate between the two. Files that are not in binary have CR LF sequences converted to LF on input,and LF to CR LF on output. This is vital for operating systems that use two characters to separate lines within text files (MS-DOS), but has no effect on operating systems that use single characters (Unix, Mac OS, QNX).

**Return Value**

undef on failure or invalid FILEHANDLE.1 on success.

**Syntax:** binmode FILEHANDLE

**eof:**

Returns 1 if the next read on FILEHANDLE will return end of file, or if FILEHANDLE is not open. An eof without an argument uses the last file read. Using eof() with empty parentheses is very different. It refers to the pseudo file formed from the files listed on the command line and accessed via the <> operator.

**Return Value**

undef if FILEHANDLE is not at end of file

1 if FILEHANDLE will report end of file on next read

**Syntax**

eof FILEHANDLE

eof()eof

**Example**

```
# insert dashes just before last line of last filewhile (<>) {
if (eof()) {                    # check for end of last fileprint "..................\n";
}
print;
last if eof();  # needed if we're reading from a terminal
```

}

**fileno:**

Returns the file descriptor number (as used by C and POSIX functions) of the specified FILEHANDLE.This is generally useful only for using the select function and any low-level tty functions.

**Return Value**

File descriptor (numeric) of FILEHANDLEundef on failure

**Syntax**

fileno FILEHANDLE

**Example**

You can use this to find out whether two handles refer to the same underlying descriptor:if

(fileno(THIS) == fileno(THAT)) {

print "THIS and THAT are dups\n";  }

**selece:**

Sets the default filehandle for output to FILEHANDLE, setting the filehandle used by functions such as print and write if no filehandle is specified. If FILEHANDLE is not specified, then it returns the name of thecurrent default filehandle.

select (RBITS, WBITS, EBITS, TIMEOUT )Calls the system function select( ) using the bits specified. Theselect function sets the controls for handling non-blocking I/O requests. Returns the number of filehandles awaiting I/O in scalar context, or the number of waiting filehandles and the time remaining in a list context.**Return Value**

Previous default filehandle if FILEHANDLE specified Current default filehandle if FILEHANDLE is not specified

**Syntax**

select FILEHANDLEselect

select RBITS, WBITS, EBITS, TIMEOUT

**Example**

```
#!/usr/bin/perl -w open(FILE,">/tmp/t.out");
$oldHandle = select(FILE); print("This is sent to /tmp/t.out.\n");select($oldHandle);
print("This is sent to STDOUT.\n");
```

**truncate:**

Truncates (reduces) the size of the file specified by FILEHANDLE to the specified LENGTH (in bytes).Produces a fatal error if the function is not implemented on your system.

**Return Value**

undef if the operation failed1 on success

**Syntax**
truncate FILEHANDLE, LENGTH
**Example**

Following example will truncate file "test.txt" to zero length.

```
#!/usr/bin/perl -w

open( FILE, "</tmp/test.txt" ) || die "Enable to open test file";truncate( FILE, 0 );

close(FILE);
```

**a)File Management**File Information

You can test certain features very quickly within Perl using a series of test operators known collectively as -X tests.

For example, to perform a quick test of the various permissions on a file, you might use a script like this:#/usr/bin/perl

```
my (@description,$size);if (-e $file)
{
push @description, 'binary' if (-B _); push @description, 'a socket' if (-S _); push
@description, 'a text file' if (-T _);
push @description, 'a block special file' if (-b _); push @description, 'a character special file'
if (-c _);push @description, 'a directory' if (-d _);
push @description, 'executable' if (-x _);
push @description, (($size = -s _)) ? "$size bytes" : 'empty';print "$file is ", join(',
',@description),"\n";
}
```

Here is the list of features which you can check for a file

| Operator | Description |
|---|---|
| -A | Age of file (at script startup) in days since modification. |
| -B | Is it a binary file? |
| -C | Age of file (at script startup) in days since modification. |
| -M | Age of file (at script startup) in days since modification. |
| -O | Is the file owned by the real user ID? |
| -R | Is the file readable by the real user ID or real group? |
| -S | Is the file a socket? |
| -T | Is it a text file? |
| -W | Is the file writable by the real user ID or real group? |
| -X | Is the file executable by the real user ID or real group? |
| -b | Is it a block special file? |
| -c | Is it a character special file? |
| -d | Is the file a directory? |
| -e | Does the file exist? |
| -f | Is it a plain file? |

| | |
|---|---|
| -g | Does the file have the setgid bit set? |
| -k | Does the file have the sticky bit set? |
| -l | Is the file a symbolic link? |
| -o | Is the file owned by the effective user ID? |
| -p | Is the file a named pipe? |
| -r | Is the file readable by the effective user or group ID? |
| -s | Returns the size of the file, zero size = empty file. |
| -t | Is the filehandle opened by a TTY (terminal)? |
| -u | Does the file have the setuid bit set? |
| -w | Is the file writable by the effective user or group ID? |
| -x | Is the file executable by the effective user or group ID? |
| -z | Is the file size zero? |

**Basic File Management:**

In unix OS, files are created on a file system by creating a link to an inode, which creates the necessarydirectory entry that links to the file data.

**rename:**

Renames the file with OLDNAME to NEWNAME. Uses the system function rename( ), and so it will notrename files across file systems or volumes. If you want to copy or move a file, use the copy or move command supplied in the File Copy module.

**Return Value**

0 on failure

1 on success

**Syntax**

rename OLDNAME, NEWNAME

**Example**

First create test file in /tmp directory and then use following code to change file name.

```
#!/usr/bin/perl -w
rename("/tmp/test", "/tmp/test2") || die ( "Error in renaming" );
```

**link:**

Creates a new file name, NEWFILE, linked to the file OLDFILE. The function creates a hard link; if youwant a symbolic link, use the symlink function.

**Return Value**

0 on failure and 1 on success

**Syntax**

link OLDFILE,NEWFILE

**Example**

This will create new file using existing file.

```
#!/usr/bin/perl
$existing_file = "/uer/home/test1";
$new_file = "/usr/home/test2";
$retval = link $existing_file, $new_file ;if( $retval == 1 ){
print"Link created successfully\n";
}else{
print"Error in creating link $!\n";
}
```

**symlink:**

Creates a symbolic link between OLDFILE and NEWFILE. On systems that don't support symbolic links,causes a fatal error.

**Return Value**

0 on failure

1 on success

**Syntax**

symlink ( OLDFILE, NEWFILE )

**Example**

First create one file test.txt in /tmp directory and then try out following example it will create a symboliclink in the same directory:

```
#!/usr/bin/perl -w
symlink("/tmp/text.txt", "/tmp/symlink_to_text.txt");
```

**unlink:**

Deletes the files specified by LIST, or the file specified by $_ otherwise. Be careful while using this functionbecause there is no recovering once a file gets deleted.

**Return Value**

Number of files deleted.

**Syntax:** unlink LIST

unlink

**Example**

Create two files t1.txt and t2.txt in /tmp directory and then use the following program to delete these twofiles.

```
#!/usr/bin/perl -w
unlink( "/tmp/t1.txt", "/tmp/t2.txt" );
```

It will produce following results:

Both the files t1.txt and t2.txt will be deleted from /tmp.

**readlink:**

Returns the pathname of the file pointed to by the link EXPR, or $_ if EXPR is not specified.

**Return Value**

undef on error

otherwise pathname of the file
Syntax
readlink EXPRreadlink

Example

#!/usr/bin/perl -w

# assume /tmp/test is a symbolic link on /tmp/usr/test.plreadlink "/tmp/test";

It will produce following results:

/tmp/usr/test.pl

**chmod:**

Changes the mode of the files specified in LIST to the MODE specified. The value of MODE should be in octal. You must check the return value against the number of files that you attempted to change to determinewhether the operation failed. This funcation call is equivalent to Unix Command chmod MODE FILELIST. **Return Value**
Integer, number of files successfully changed

**Syntax**
chmod MODE, LIST
**Example**

$cnt = chmod 0755, 'foo', 'bar';

**chown:**

chmod 0755, @executables;

$mode = '0644'; chmod $mode, 'foo';          # !!! sets mode to

# --w-------------------------------------r-T

$mode = '0644'; chmodoct($mode), 'foo'; # this is better

$mode = 0644;   chmod $mode, 'foo';          # this is best

Changes the owner (and group) of a list of files. The first two elements of the list must be the numeric uidand gid, in that order. This funcation call works in similar way as unix command chown. Thus you shouldhave sufficient privilege to change the permission of the file.

**Return Value**

Returns the number of files successfully changed.

**Syntax**

chown USERID, GROUPID, LIST
**Example**

Here is an example which will change ownership for the given number of files.

```
#!/usr/bin/perl
$cnt = chown $uid, $gid, 'foo', 'bar';chown $uid, $gid, @filenames;
```

**utime:**

Sets the access and modification times specified by ATIME and MTIME for the list of files in LIST. Thevalues of ATIME and MTIME must be numerical. The inode modification time is set to the current time.The time must be in the numeric format (for example, seconds since January 1, 1970).

**Return Value**

Number of files updated

**Syntax:** utime ATIME, MTIME, LIST

**Example**

Create a file t.txt in /tmp directory and use the following program to set its modification and access time tothe current time.

```
#!/usr/bin/perl –w
utime(time(), time(), "/tmp/t.txt");
```

**umask:**

Sets the umask (default mask applied when creating files and directories) for the current process. Value ofEXPR must be an octal number. If EXPR is omitted, simply returns the previous value.

**Return Value**

Previous umask value

**Syntax**

umask EXPRumask

**Example**

```
#!/usr/bin/perl -w
print("The current umask is: ", umask(), "\n");
```

It will produce following results, you can get different result on your computer based on your setting.The current umask is: 18

**Accessing Directory Entries:**

Following are the standard functions used to play with directories.

opendir DIRHANDLE, EXPR # To open a directoryreaddir DIRHANDLE# To read a directory
rewinddir DIRHANDLE          # Positioning pointer to the begining telldir DIRHANDLE
                                              # Returns current position of the dir seekdir DIRHANDLE, POS
# Pointing pointer to POS inside dirclosedir DIRHANDLE            # Closing a directory.

Here is an example which opens a directory and list out all the files available inside this

directory.#!/usr/bin/perl

```
opendir (DIR, '.') or die "Couldn't open directory, $!";while ($file = readdir DIR)
{
print "$file\n";
}
closedir DIR;
```

Another example to print the list of C source code files, you might use#!/usr/bin/perl

```
opendir(DIR, '.') or die "Couldn't open directory, $!";foreach (sort
grep(/^.*\.c$/,readdir(DIR)))
{
print "$_\n";
}
closedir DIR;
```

You can make a new directory using the mkdir function:

To remove a directory, use the rmdir function:

To change the directory you can use chdir function.

**Managing Directories:**

The system chdir() function is supported within Perl in order to change the current directory for the currentprocess.

**chdir:**

Changes the current working directory to EXPR, or to the user's home directory if none is specified. Thisfunction call is equivalent to Unix command cd EXPR.

**Return Value**

0 on failure

1 on success

**Syntax**

**chdir EXPRchdir**

**Example**

Assuming you are working in /user/home/tutorialspoint directory. Execute following program:

```
#!/usr/bin/perl chdir "/usr/home";
# Now you are in /usr/home dir.chdir;
# Now you are in home directory
/user/home/tutorialspoint
```

**mkdir:**

You can make a new directory using the mkdirfunction.Makes a directory with the name and path EXPRusing the mode specified by MODE, which should be supplied as an octal value for clarity.

**Return Value**

**Syntax**

on failure

0 on success

mkdir EXPR,MODE

**Example**

```
#!/usr/bin/perl -w

$dirname ="/tmp/testdir";mkdir $dirname, 0755;
```

It will produce following results in /tmp directory:

```
drwxr-xr-x  2 root   root                    4096 Mar 19 11:55 testdir
```

**rmdir:**

Deletes the directory specified by EXPR, or $_ if omitted. Only deletes the directory if the directory

is empty.

**Return Value**

0 on failure

**1** on success

Syntax

rmdir EXPRrmdir

**Example**

Create one directory testdir inside /tmp and Try out following example:

```
#!/usr/bin/perl -w
```

rmdir ("/tmp/testdir") || die ("error in deleting directory: $?");It will produce following results:

If directory is empty then it will be deletedotherwise error message will be generated.

**File Control with fcntl:**

Performs the function specified by FUNCTION, using SCALAR on FILEHANDLE. SCALAReither contains a value to be used by the function or is the location of any returned information.

**Return Value**

0 but true if the return value from the fcntl()is 0 Value returned by system.

undef on failure

**Syntax:**

fcntl FILEHANDLE, FUNCTION, SCALAR

**Example**

use Fcntl;

fcntl($filehandle, F_GETFL, $packed_return_buffer)or die "can't fcntl F_GETFL: $!";

**I/O Control with ioctl:**

The ioctl function is similar in principle to the fcntl function. It too is a Perl version of the operating system equivalent ioctl() function.

The ioctl function is typically used to set options on devices and data streams,usually relating directly to the operation of the terminal.

**Return Value**

undef on failure otherwise 0 but true if the return value from the ioctl( )is 0

**Syntax**

ioctl FILEHANDLE, FUNCTION, SCALAR

**File Locking:**

File locking is a way of ensuring the integrity of files. It allows many people (actually, processes) toshare a file in a safe way, without stepping on each other's toes.

Supports file locking on the specified FILEHANDLE using the system flock( ), fcntl( ) locking, or lockf( ). The exact implementation used is dependent on what your system supports. OPERATION is one ofthe static values defined here...

| Operation | Result | | |
|---|---|---|---|
| LOCK_SH | Set shared lock. LOCK_EX | | Set exclusive lock. |

LOCK_UNUnlock specified file. LONG_NB Set lock without blocking.

**Return Value**

0 on failure to set/unset lock 1 on success to set/unset lock

**Syntax**

flock FILEHANDLE, OPERATION

**Example**

Here's a mailbox appender for BSD systems: use Fcntl ':flock'; # import LOCK_* constants

```
sub lock {
flock(MBOX,LOCK_EX);
# and, in case someone appended# while we were waiting... seek(MBOX, 0, 2);
}
sub unlock {
flock(MBOX,LOCK_UN);
}
open(MBOX, ">>/usr/spool/mail/$ENV{'USER'}")or die "Can't open mailbox: $!";
lock();
print MBOX $msg,"\n\n";unlock();
```

## Data Manipulation:

Perl was originally designed as a system for processing logs and reporting on the information.The data-manipulation features built into Perl, from the basics of numerical calculations through to basic string handling.

a) **Working with Numbers**

Numbers are scalar data. They exist in PERL as real numbers, float, integers, exponents,

b) **abs-the Absolute Value**

Returns the absolute value of its argument. If pure interger value is passed then it will return it as it isbut if a string is passed then it will return zero. If VALUE is omitted then it uses $_.

**Return Value:**

Returns the absolute value of its argument.

**Syntax:**

abs VALUEabs

**Example:**

print abs(-1.295476);

This should print a value of 1.295476. Supplying a positive value to abs will return thesame positive value or, more correctly, it will return the nondesignated value: all positive values imply a + sign in front of them.

**c) int-Converting Floating Points to Integers**

Returns the integer element of EXPR, or $_ if omitted. The int function does not do rounding. If you need toround a value up to an integer, you should use sprintf.

**Return Value:**

Integer part of EXPR.

**Example:**

printint abs(-1.295476);

This should print a value of 1. The only problem with the int function is that it strictly removes the fractional component of a number; no rounding of any sort is done. If youwant to return a number that has been rounded to a number of decimal places, use the printf or sprintf function:

printf("%.2f",abs(-1.295476));

This will round the number to two decimal places-a value of 1.30 in this example.Note that the 0 is appended in the output to show the two decimal places.

**d) exp-Raising e to the Power**

To perform a normal exponentiation operation on a number, you use the ** operator:

$square - 4**2;

This returns 16, or 4 raised to the power of 2. If you want to raise the natural basenumber e to the power, you need to use the exp function:

exp EXPRexp

If you do not supply an EXPR argument, exp uses the value of the $_variable as theexponent. For example, to find the square of e:

$square -exp(2);

**e) sqrt-the Square Root**

To get the square root of a number, use the built-in sqrt function:

$var-sqrt(16384);

To calculate the n$^{th}$ root of a number, use the ** operator with a fractional number.For example, the following line

    $var- 16384**(1/2);is identical to

$var-sqrt(16384);

To find the cube root of 16,777,216, you might use

    $var- 16777216**(1/3); which should return a value of 256.

**5log-the Logarithm**

To find the logarithm (base e) of a number, you need to use the log function:

```
$log - log 1.43;
```

**Trigonometric Functions**

There are three built-in trigonometric functions for calculating the arctangent squared(atan2), cosine (cos), and sine (sin) of a value:

```
atan2 X,Ycos EXPRsin EXPR
```

If you need access to the arcsine, arccosine, and tangent, then use the POSIXmodule, which supplies the corresponding acos, asin, and tan functions.

Unless you are doing trigonometric calculations, there is little use for thesefunctions in everyday life. However, you can use the sin function to calculate yourbiorhythms using the simple script shown next, assuming you know the number of days you have been alive:

```
my ($phys_step, $emot_step, $inte_step) - (23, 28, 33);use Math::Complex;
print "Enter the number of days you been alive:\n";my $alive -<STDIN>;
$phys-int(sin(((pi*($alive%$phys_step))/($phys_step/2)))*100);
$emot-int(sin(((pi*($alive%$emot_step))/($emot_step/2)))*100);
$inte-int(sin(((pi*($alive%$inte_step))/($inte_step/2)))*100); print "Your Physical is
$phys%, Emotional $emot%, Intellectual
$inte%\n";
```

**Conversion between Bases**

Perl provides automatic conversion to decimal for numerical literals specified in binary, octal, and hexadecimal. However, the translation is not automatic on values contained within strings, either those defined using string literals or from strings imported from the outside world (files, user input, etc.).

To convert a string-based literal, use the oct or hex functions. The hex function converts only hexadecimal numbers supplied with or without the 0x prefix. For example, the decimal value of the hexadecimal string “ff47ace3” (42,828,873,954) canbe displayed with either of the following statements:

```
print hex("ff47ace3");
print hex("0xff47ace3");
```

The hex function doesn’t work with other number formats, so for strings that startwith 0, 0b, or 0x, you are better off using the oct function. By default, the oct function interprets a string without a prefix as an octal string and raises an error if it doesn’t see
it. So this

printoct("755");is valid, but this

printoct("aef");will fail.

If you supply a string using one of the literal formats that provides the necessaryprefix, oct will convert it, so all of the following are valid:

printoct("0755"); printoct("0x7f"); printoct("0b00100001");

Both oct and hex default to using the $_ variable if you fail to supply an argument.To print out a decimal value in hexadecimal, binary, or octal, use printf, or use sprintf to print a formatted base number to a string:

printf ("%lb %lo %lx", oct("0b00010001"), oct("0755"), oct("0x7f"));See printf in Chapter 7 for more information.

- **Conversion Between Characters and Numbers:**

If you want to insert a specific character into a string by its numerical value, you can use the \0 or \x character escapes:

print "\007";print "\x07";

These examples print the octal and hexadecimal values; in this case the “bell” character. Often, though, it is useful to be able to specify a character by its decimal number and to convert the character back to its decimal equivalent in the ASCII table.

The chr function returns the character matching the value of EXPR, or $_if EXPR is not specified. The value is matched against the current ASCII table for the operating system, so it could reveal different values on different platforms for characters with an ASCII value of 128 or higher. This may or may not be useful.

The ord function returns the numeric value of the first character of EXPR, or $_ if EXPR is not specified. The value is returned according to the ASCII table and is always unsigned.

Thus, using the two functions together,printchr(ord('b'));

we should get the character “b”.

- **Random Numbers:**

Perl provides a built-in random number generator. All random numbers need a “seed” value, which is used in an algorithm, usually based on the precision, or lack thereof, for a specific calculation. The format for the rand function isrand EXPR

rand

The function returns a floating-point random number between 0 and EXPR or between 0 and 1 (including 0, but not including 1) if EXPR is not specified. If you wantan integer random number, just use the int function to return a reasonable value, as in this example:

```
printint(rand(16)),"\n";
```

You can use the srand function to seed the random number generator with aspecific value:

```
srand EXPR
```

The rand function automatically calls the srand function the first time rand iscalled, if you don't specifically seed the random number generator.

The following program calculates the number of random numbers generated beforea duplicate value is returned:

```
my %randres; my $counter - 1;
srand((time() ^ (time() % $])) ^ exp(length($0))**$$);while (my $val- rand())
{
last if (defined($randres{$val}));
print "Current count is $counter\n" if (($counter %10000) -- 0);
$randres{$val} - 1;
$counter++;
}
print "Out of $counter tries I encountered a duplicate random number\n";
```

- **Working with StringsString Concatenation**

**Method 1 - using Perl's dot operator**

Concatenating strings in Perl is very easy. There are at least two ways to do it easily. One way to doit - the way I normally do it - is to use the "dot" operator (i.e., the decimal character).

The simple example shown below demonstrates how you can use the dot operator to merge, or concatenate, strings. For simplicity in this example, assume that the variable $name is assigned the value"checkbook". Next, you want to pre-pend the directory "/tmp" to the beginning of the filename, and the filename extension ".tmp" to the end of $name. Here's the code that creates the variable $filename:

```
$name = "checkbook";
$filename = "/tmp/" . $name . ".tmp";

# $filename now contains "/tmp/checkbook.tmp"
```

**Method 2 - using Perl's join function**

A second way to create the desired variable $filename is to use the join function. With the join function, you can merge as many strings together as you like, and you can specify what token you want touse to separate each string.

The code shown in Listing 2 below uses the Perl join function to achieve the same result as the codeshown in Listing 1:

$name = "checkbook";

$filename = join "", "/tmp/", $name, ".tmp";

# $filename now contains "/tmp/checkbook.tmp

➢ **String Length**

To get the length of a string in Perl, use the Perl length function on your string, like this:

$size = length $last_name;

The variable $size will now contain the string length, i.e., the number of characters in the variable named

$last_name.

**Example**

Here's a more complete script to demonstrate this Perl string length technique:

$a = "foo, bar, and baz";print length $a;

When that little script is run through the Perl interpreter it will print the number 17, which is the length ofthat string (i.e., the number of characters in the Perl string).

➢ **Case Modifications**

The four basic functions are lc, uc, lcfirst, and ucfirst. They convert a stringto all lowercase, alluppercase, or only the first character of the string to lowercase oruppercase, respectively.

For Example:

$string = "The Cat Sat on the Mat"; printlc($string) # Outputs 'the cat sat on the mat' printlcfirst($string) # Outputs 'the Cat Sat on the Mat' printuc($string) # Outputs 'THE CAT SAT ON THE MAT'printucfirst($string) # Outputs 'The Cat Sat on the Mat'

These functions can be useful for “normalizing” a string into an all uppercase or lowercase format—useful when combining and de-duping lists when using hashes.

➢ **End-of-Line Character Removal**

The chop function can be used to strip the last character off any expressionwhile(<FH>)

{

chop;

...

}

- **String Location**

The index() function is used to determine the position of a letter or a substring in a string. For example, in the word "frog" the letter "f" is in position 0, the "r" in position 1, the "o" in 2 and the "g" in 3. The substring "ro" is in position 1. Depending on what you are trying to achieve, the index() function may be faster or more easy to understand than using a regular expression or the split() function.

Example:

```
#!/usr/bin/perluse strict;
use warnings;
my $string = 'perlmeme.org';my $char = 'l';
my $result = index($string, $char);print "Result: $result\n";
```

Instead of looping through every occurrence of a letter in a string to find the last one, you can use therindex() function. This works exactly the same as index() but it starts at the end of the string. The index value returned is string from the start of the string though.

Example:

```
#!/usr/bin/perluse strict;
use warnings;
my $string = 'perlmeme.org';my $char = 'e';
my $result = rindex($string, $char);print "Result: $result\n"
```

- **Extracting Substrings**

The substr() function is used to return a substring from the expression supplied as it's first argument.The function is a good illustration of some of the ways in which Perl is different from other languages you may have used. substr() has a variable number of arguments, it can be told to start at an offset from either end of the expression, you can supply a replacement string so that it replaces part of the expression as well as returning it, and it can be assigned to!.

Example:

```
#!/usr/bin/perluse strict;
use warnings;

my $string = 'Now is the time for all good people to come to the aid of their party';my
$fragment =  substr $string, 4;
```

```
print "  string: <$string>\n";
print "fragment: <$fragment>\n";
```

- **Stacks**

One of the most basic uses for an array is as a stack. If you consider that an array is a list of individual scalars, it should be possible to treat it as if it were a stack of papers.Index 0 of the array isthe bottom of the stack, and the last element is the top. You canput new pieces of paper on the top of the stack (push), or put them at the bottom(unshift). You can also take papers off the top (pop) or bottom (shift)of the stack.

**Push and Pop**

In perl stacks are easy to be implemented. Stacks which follows LIFO (Last In First Out) datastructure can be handled in perl with 'POP' and 'PUSH' keywords.

**POP Example:**

```
my $strTestPOP(@arrTest);print "$strTest";
```

In this example the Last element present in @arrTest will be poped and will be assigned to variable $strTest(LIFO). Suppose the array contains value "STAR" at it's last position thus the next line of the code will populate STAR.

**PUSH Example:**

```
PUSH(@arrTest $strTest) if ($strTest);
```

In this example we are pushing $strTest variable to the array @arrTest. After the PUSH statement the arraysize will be increased by one and the last element of the array will be the value contained in variable
$strTest.

**Shift And Unshift:**

The shift function returns the first value in an array, deleting it and shifting theelements of the arraylist to the left by one.

- **Splicing Arrays**

Perl's splice() function is used to cut out and return a chunk or portion of an array. The portion that iscut out starts at the OFFSET element of the array and continues for LENGTH elements. If the LENGTH is not specified, it will cut to the end of the array.

**Example:**

```
@myNames = ('Jacob', 'Michael', 'Joshua', 'Matthew', 'Ethan', 'Andrew');@someNames =
splice(@myNames, 1, 3);
```

**Join:**

Perl's join() function is used to connect all the elements of a specific list or array into a single string
with the JOINER expression. The list is concatenated into one string with the element contained in theJOINER variable in between each item.

Syntax:

$STRING = join(JOINER, @LIST);

**Example:**

@myNames = ('Jacob', 'Michael', 'Joshua', 'Matthew');

$someNames = join(', ', @myNames);

Think of the @myNames array as a row of numbered boxes, each with a name in it. The join() function would take all of the content in those boxes in the @myNames array and connect them together with a comma and a space. The value of @someNames then becomes "Jacob, Michael, Joshua, Matthew".The list of names is concatenated, or joined, by commas.

**split:**

Perl's split() function is used to break a string into an array on a specific pattern. The PATTERN is aregular expression that can be as simple as a single character. By default the STRING is split on every instance of the PATTERN, but you can LIMIT it to a specific number of instances.

**Syntax:**

@LIST = split(/PATTERN/, STRING, LIMIT);

**Example:**

$myNames = "Jacob,Michael,Joshua,Matthew,Ethan,Andrew";@nameList = split(/,/, $myNames);

In the above example, the split function takes a plain string of names, separated by commas. It thenbreaks the string apart on each instance of the comma, putting the results into the array @nameList. The value of @nameList would then be:

@myNames = ('Jacob', 'Michael', 'Joshua', 'Matthew', 'Ethan', 'Andrew');

By using the LIMIT option, you can split the array into a specific size. For example:

$myNames = "Jacob,Michael,Joshua,Matthew,Ethan,Andrew";@nameList = split(/,/, $myNames, 3);

This would split the array into 3 chunks, so the value of @nameList would then be:

@myNames = ('Jacob', 'Michael', 'Joshua,Matthew,Ethan,Andrew');

**grep:**

Perl's grep() function runs a regular expression on each element of an array, and returns only the

elements that evaluate to true. Using regular expressions can be extremely powerful and complex. You canfind out more about them in the regular expressions topic.

**Syntax:**

@LIST = grep(EXPRESSION, @ARRAY);

**Example:**

@myNames = ('Jacob', 'Michael', 'Joshua', 'Matthew', 'Alexander', 'Andrew');@grepNames = grep(/^A/, @myNames);

Think of the @myNames array as a row of numbered boxes, going from left to right, numbered starting with a zero. The grep() function goes through each of the elements (boxes) in the array, and compares their contents to the regular expression. If the result is true, the contents are then added to the new@grepNames array.

In the above example, the regular expression is looking for any value that starts with a capital A. After sifting through the contents of the @myNames array, the value of @grepNames becomes ('Alexander','Andrew').

One quick way to make this particular function more powerful is to reverse the regular expressionwith the NOT operator. The regular expression then looks for elements that evaluate to false and moves them into the new array.

@myNames = ('Jacob', 'Michael', 'Joshua', 'Matthew', 'Alexander', 'Andrew');@grepNames = grep(!/^A/, @myNames);

In the above example, the regular expression is looking for any value that does not start with a capital A. After sifting through the contents of the @myNames array, the value of @grepNames becomes('Jacob', 'Michael', 'Joshua', 'Matthew').

**map:**

Perl's map() function runs an expression on each element of an array, and returns a new array withthe results.

**Syntax:**

@LIST = map(EXPRESSION, @ARRAY);

**Example**

@myNames = ('jacob', 'alexander', 'ethan', 'andrew');@ucNames = map(ucfirst, @myNames);

Think of the @myNames array as a row of numbered boxes, going from left to right, numbered starting with a zero. The map() function goes through each of the elements (boxes) in the array, and runs theEXPRESSION on their contents. The altered contents are then added to the new @ucNames array.

In the above example, EXPRESSION is the built-in function ucfirst that makes the first letter of theword upper case. After running the expression on each element of the @myNames array, the value of @ucNames becomes ('Jacob', 'Alexander', 'Ethan', 'Andrew').

**sort:**

With any list, it can be useful to sort the contents. Doing this manually is a complex process, so Perlprovides a built-in function that takes a list and returns a lexically sorted version.

**Syntax:**

sort @array;

**Example:**

Or to specify an explicit lexical subroutine:sort { $a cmp $b } @array;

**reverse:**

Perl's reverse() function is used to reverse the order of an array. It should be noted that the functionreturns a reversed array but does not actually reverse the array passed into the function.

**Syntax:**

@REVERSED_LIST = reverse(@ARRAY);

**Example:**

@myNames = ('Jacob', 'Michael', 'Ethan', 'Andrew');@reversedNames = reverse(@myNames);

Think of the @myNames array as a row of numbered boxes, going from left to right, numbered starting with a zero. The reverse() function would copy the contents of the boxes from the @myNames arrayinto the @reversedNames array, while reversing their order. The value of @reversedNames then becomes ('Andrew', 'Ethan', 'Michael', 'Jacob'), and @myNames remains the same.

➢ **Regular Expressions**

Regular expressions are very powerful tools for matching, searching, and replacing text. Unfortunately, they are also very obtuse. It does not help that most explanations of regular expressions startfrom the specification.

A regular expression is a string of characters that define the pattern or patterns you are viewing. Thesyntax of regular expressions in Perl is very similar to what you will find within other regular expression.

- **Pattern Modifiers**

Pattern Modifiers change the behavior of the individual operator.

| **Modifier** | **Meaning** |
|---|---|
| `/i` | Ignore alphabetic case distinctions (case insensitive). |
| `/s` | Let `.` match newline and ignore deprecated `$*` variable. |
| `/m` | Let `^` and `$` match next to embedded `\n`. |
| `/x` | Ignore (most) whitespace and permit comments in pattern. |
| `/o` | Compile pattern once only. |

The /i modifier says to match both upper- and lowercase (and title case, under Unicode).
The /s and /m modifiers don't involve anything kinky. Rather, they affect how Perl treats matches against astring that contains newlines. But they aren't about whether your string actually contains newlines.
The /m modifier changes the interpretation of the ^ and $ metacharacters by letting them match next tonewlines within the string instead of considering only the ends of the string.
The /o modifier controls pattern recompilation.
The /x is the *ex*pressive modifier: it allows you to *ex*ploit whitespace and *ex*planatory comments in order to
*ex*pand your pattern's legibility, even *ex*tending the pattern across newline boundaries.

- **The Match Operator**

The match operator, m//, is used to match a string or statement to a regular expression. For example,to match the character sequence "foo" against the scalar $bar, you might use a statement like this:

if ($bar =~ /foo/)

The m// actually works in the same fashion as the q// operator series.you can use any combination of naturally matching characters to act as delimiters for the expression. For example, m{}, m(), and m>< are allvalid.

You can omit the m from m// if the delimiters are forward slashes, but for all other delimiters youmust use the m prefix.

Note that the entire match expression.that is the expression on the left of =~ or !~ and the matchoperator, returns true (in a scalar context) if the expression matches. Therefore the statement:

$true = ($foo =~ m/foo/);

Will set $true to 1 if $foo matches the regex, or 0 if the match fails.

In a list context, the match returns the contents of any grouped expressions. For example, whenextracting the hours, minutes, and seconds from a time string, we can use:

```
my ($hours, $minutes, $seconds) = ($time =~ m/(\d+):(\d+):(\d+)/);
```

**Match Operator Modifiers**

The match operator supports its own set of modifiers. The /g modifier allows for global matching.

The /i modifier will make the match case insensitive. Here is the complete list of modifiers

| **Modifier** | **Descripti** |
|---|---|
| i | Makes the match case insensitive |
| m | Specifies that if the string has newline or carriage return characters, the ^ and $ operators will nowmatch against a newline instead of astring |
| o | Evaluates the expression only once |
| s | Allows use of . to match a newline character |
| x | Allows you to use white space in the expression |
| g | for clarityGlobally finds all matches |
| c | Allows the search to continue even after a global |

**Matching Only Once**

There is also a simpler version of the match operator - the ?PATTERN? operator. This is basicallyidentical to the m// operator except that it only matches once within the string you are searching between each call to reset.

For example, you can use this to get the first and last elements within a list:

```
#!/usr/bin/perl
@list = qw/food foosball subeo footnote terfootcanicfootbrdige/;
foreach (@list)
{
$first = $1 if ?(foo.*)?;
$last = $1 if /(foo.*)/;
}
print "First: $first, Last: $last\n";
```

This will produce following resultFirst: food, Last: footbrdige

**The Substitution Operator**

The substitution operator, s///, is really just an extension of the match operator that allows you toreplace the text matched with some new text. The basic form of the operator is:

```
s/PATTERN/REPLACEMENT/;
```

The PATTERN is the regular expression for the text that we are looking for. The REPLACEMENTis a specification for the text or regular expression that we want to use to replace the found text with.

For example, we can replace all occurrences of .dog. with .cat. using

```
$string =~ s/dog/cat/;Example:
```

```
#/user/bin/perl

$string = 'The cat sat on the mat';
$string =~ s/cat/dog/;
print "Final Result is $string\n"; This will produce following resultThe dog sat on the mat
```

➢ **The Substitution Operator**

Here is the list of all modifiers used with substitution operator

| **Modifier** | **Description** |
|---|---|
| i | Makes the match case insensitive |
| m | Specifies that if the string has newline or carriage return characters, the ^ and $ operators will now match against a newline boundary, instead of a string boundary |
| o | Evaluates the expression only once |
| s | Allows use of . to match a newline character |
| x | Allows you to use white space in the expression for clarity |
| g | Replaces all occurrences of the found expression with the replacement text |
| e | Evaluates the replacement as if it were a Perl statement, and uses its return value as the replacement text |

➢ **Translation**

Translation is similar, but not identical, to the principles of substitution, but unlike substitution, translation (or transliteration) does not use regular expressions for its search on replacement values. Thetranslation operators are:

```
tr/SEARCHLIST/REPLACEMENTLIST/cdsy/SEARCHLIST/REPLACEMENTLIST/cds
```

The translation replaces all occurrences of the characters in SEARCHLIST with the corresponding characters in REPLACEMENTLIST. For example, using the "The cat sat on the mat." string we have beenusing in this chapter:

```
#/user/bin/perl

$string = 'The cat sat on the mat';
$string =~ tr/a/o/;print "$string\n";
```

This will produce following result

The cot sot on the mot.

Standard Perl ranges can also be used, allowing you to specify ranges of characters either by letter ornumerical value. To change the case of the string, you might use following syntax in place of the uc function.

```
$string =~ tr/a-z/A-Z/;
```

**Translation Operator Modifiers**

Following is the list of operators related to translation

| Modifier | Description |
|---|---|
| c | Complement SEARCHLIST. |
| d | Delete found but unreplaced characters. |
| s | Squash duplicate replaced characters. |

The /d modifier deletes the characters matching SEARCHLIST that do not have a corresponding entry inREPLACEMENTLIST. For example:

```
#!/usr/bin/perl

$string = 'the cat sat on the mat.';
$string =~ tr/a-z/b/d;print "$string\n";
This will produce following resultb bb.
```

The last modifier, /s, removes the duplicate sequences of characters that were replaced, so:

```
#!/usr/bin/perl

$string = 'food';
$string = 'food';
$string =~ tr/a-z/a-z/s;print $string;
This will produce following resultFod
```

- **Regular Expression Elements**

Regular expressions are their own little language nestled inside of Perl.For all their power and expressivity, patterns in Perl recognize the same 12 traditional metacharacters (the Dirty Dozen, as it were) found in manyother regular expression packages:

\ | ( ) [ { ^ $ * + ? .

Some simple metacharacters stand by themselves, like .and ^ and $. They don't directly affect anything around them. Some metacharacters work like prefix operators, governing what follows them, like \.Others work like postfix operators, governing what immediately precedes them, like *, +, and ?. One metacharacter, |, acts like an infix operator, standing between the operands it governs. There are even bracketing metacharacters that work like circumfix operators, governing something contained inside them, like (...) and [...]. Parentheses are particularly important, because they specify the bounds of | on the inside, and of *, +, and ?on the outside.

**Metasymbol Tables**

In the following tables, the Atomic column says "yes" if the given metasymbol is quantifiable (if it can match something with width, more or less). Also, we've used "..." to represent "something else". Pleasesee the later discussion to find out what "..." means, if it is not clear from the one-line gloss in the table.)

In this table shows the basic traditional metasymbols. The first four of these are the structural metasymbols we mentioned earlier, while the last three are simple metacharacters. The .metacharacter is an example of an atom because it matches something with width (the width of a character, in this case); ^ and $ are examples of assertions, because they match something of zero width, and because they are onlyevaluated to see if they're true or not.

**General Regex Metacharacters**

| **Symbol** | **Atomic** | **Meaning** |
|---|---|---|
| \... | Varies | De-meta next nonalphanumeric character, meta next alphanumeric character (maybe). |
| ...\|... | No | Alternation (match one or the other). |
| (...) | Yes | Grouping (treat as a unit). |
| [...] | Yes | Character class (match one character from a set). |
| ^ | No | True at beginning of string (or after any newline, maybe). |
| . | Yes | Match one character (except newline, normally). |
| $ | No | True at end of string (or before any newline, maybe). |

- **Regular Expression Variables**

Regular expression variables include $, which contains whatever the last grouping match matched;

$&, which contains the entire matched string; $`, which contains everything before the matched string; and

$', which contains everything after the matched string.The following code demonstrates the result:

```
#!/usr/bin/perl

$string = "The food is in the salad bar";
$string =~ m/foo/; print "Before: $`\n"; print "Matched: $&\n";print "After: $'\n";
```

This code prints the following when executed:

```
Before: The Matched: foo
After: d is in the salad bar
```

- **Regular Expression Extensions/Assertions**

The regular expression engine also allows you to specify a number of additionalextensions, called assertions, within the main expression. These extensions enablemore specific matches to take place withoutthe match affecting the variables and/orgroupings that are in place.

For example, here’s a regular expression match using the (?{code}) assertion:

```
use re 'eval';
$_ = '<A href="/index.shtml">';m<
(?{ $cnt = 0 })
\<A.*"
(.(?{ local $cnt = $cnt + 1;}))*"\>
(?{ $res = $cnt })
>x;
print $res," words\n";
```

It counts the number of letters between the double quotes in the HTML referencespecified in $_.

| Assertion | Meaning |
|---|---|
| (?#text) | Comment text within the brackets isignored. |
| (?:pattern) | Identical to grouping, but does not populate$1, $2, and so on, on a match. |
| (?imsx:pattern) | Identical to grouping, but does not populate $1, $2, and so on, on a match;embeds pattern-match modifiers for theduration of the specified pattern. |
| (?=pattern) | Matches if the regular expression engine would match pattern next, without affecting the result of thematch. |
| (?!pattern) | Matches if the regular expressionengine would not match pattern next. For |

Regular Expression Assertions

- **Precompiling Expressions**

One of the pitfalls of the regular expression mechanism is that when interpolating variables intoexpressions, Perl must recompile the regular expression each time.Forexample, the code:

```
while(<FILE>)
{
foreach $regex (@expressions)
{
print if /$regex/;
}
}
```

would be incredibly time consuming, because for each line in FILE, we have to recompile each of the regular expressions in @expressions, even though the contentsof @expressions don’t change between each line.

The qr// operator takes a regular expression and compiles it as normal, returning a regular expression object such that

```
$regex = qr/[a-z]+/is;s/$regex/nothing/;
```

is the same as

```
s/[a-z]+/nothing/is;
```

Because the returned object is a compiled regular expression, we can solve the earlier problem by precompiling all the expressions before we enter the loop.

That means we could change the preceding example to@regexes = map { qr/$_/ } @expressions; while(<FILE>)

```
{
foreach $regex (@regexes)
{
print if /$regex/;
}
}
```

Now, because the patterns are precompiled, the regular expressions are executedimmediately within the main loop without requiring a new compilation.

The return value of the qr// operator can also be embedded into other expressions:print if /start${regex}end/;

➢ **Regular Expression Support Functions**

There are three functions that support the regular expression engine.

**Pos:**

Used to find the offset or position of the last matched substring. If SCALAR is specified, it will return the offset of the last match on that scalar variable. You can also assign a value to this function (for example, pos($foo) = 20;) in order to change the starting point of the next match operation. Offset is counterstarting from zeroth position.

**Syntax**

pos EXPRpos

***Return Value***

- In scalar context, Interger
- In list context, then positions of all the matches within the regular expression

***Example***

```
#!/usr/bin/perl -w

$name = "This is alpha beta gamma";

$name =~ m/alpha/g; print("pos() ", pos($name), "\n");
```

It will produce following results:

```
pos() 13
```

**quotemeta:**

Escapes all meta-characters in EXPR. For example, quotemeta("AB*..C") returns "'AB*\.\.C".

**Syntax**

quotemeta EXPR

**Return Value**

- A string with all meta-characters escaped.

**Example**

```
#!/usr/bin/perl -w printquotemeta("AB*\n[.]*");
```

It will produce following results:

```
AB\*\
\[\.\]\*
```

**study:**

Takes extra time to study EXPR in order to improve the performance on regular expressions conducted on EXPR. If EXPR is omitted, uses $_. The actual speed gains may be very small, depending onthe number of times you expect to search the string. You can only study one expression or scalar at any onetime.

**Syntax:**

study EXPRstudy