OPEN SOURCE PROGRAMMING

SUBJECT CODE: U5CA22MT

Introduction : Linux Essential Commands – File system Concept – Standard Files – The Linux Security Model – Vi Editor – Partitions Creation – Shell Introduction – String Processing– Investigation and Managing Processes – Network Clients – Installing Application.

Difference between OSS and Proprietary S/W

OSS

- Permits Users to study, change and Improve
- Redistribute the s/w without restriction
- Access the source code
- Modify the source code
- Distribute the modified version of the s/w.

Proprietary s/w

- Non free software
- Copy Right (License)
- Usually not allow their source code to modify.

Advantages of Open Sources

Lower Costs

- Usually not require licensing fee
- Small Businesses easily choose to adopt

Flexibility

- A programmer modify to better suit its needs
- Add new function or remove function

High-Quality Software

- Source code available and well designed

Reduces “Vendor Lock-in”

- If you are using proprietary s/w they may restricted
- Independent of vendors in OSS.

Simple License Management

- No worry about licenses
- You can install several times from any location

Abundant Support

- Most organization gives support and maintenance

Disadvantages of open source

Support

- One disadvantage to open source involves frequently poor support

Documentation

- Many open source products were poorly documented - or not documented at all.

Complexity

- Difficult to learn and administer when problems occurs

Advertising

- Annoying advertising components may be another factor which takes the

Advantages of proprietary software

- Single vendor
- Professional interface
- Routine updates
- Integration across products

Disadvantages of proprietary software

- Products can be bulky
- Cost surprises

Applications of open source softwares

There are over 480 OSS application to use or build upon.

Accounting

Adaptive Planning Express- An enabling medium sized companies to automate budgeting and forecasting

Buddi - A simple budgeting program for users with no financial background

CheckInOut – Railway Applications

To manage money accounts

CRM(Customer Relationship Management)

OpenCRM :

Tracking clients and project management

OPenCRX:

Bug tracking and activity management features

Graphics –Design & Modeling Tools

Flowchart Studio – User to draw a flowchart through graphical editor

Media Player

Mplayer – Written in Python

Need of Open sources

- Free Redistribution
- Source Code
- Derived Works
- No Discrimination against Persons or Groups
- License must not be specific to a product
- License must not restrict other software
- License must be technology - Neutral

Examples Of Open Source Software

- Apache HTTP Server [http://httpd.apache.org/] (web server)
- GNOME [http://www.gnome.org/] (Linux desktop environment)
- GNU Compiler Collection [http://www.gnu.org/software/gcc/gcc.html] (GCC, a suite of compilation tools for C, C++, etc)
- KDE [http://www.kde.org/] (Linux desktop environment)
- Firefox [http://www.mozilla.com/en-US/firefox/] (web browser based on Mozilla)
- MySQL [http://www.mysql.com/] (database)
- OpenOffice.org [http://www.openoffice.org/] (office suite, including word processor, spreadsheet, and presentation software)
- PHP [http://www.php.net/] (web development)
- Perl [http://www.perl.org/] (programming/scripting language)
- Python [http://www.python.org/] (programming/scripting language)
- Samba [http://www.samba.org/] (file and print server)

Open Source Operating System(Linux)

Introduction to Linux

- Unix-like computer operating system
- Assembled under open source development and distribution
- The defining component of Linux is Linux Kernel
- An Operating system kernel first released on 5th october 1991 by **Linux Torvalds.**

What is Linux?
Linux + GNU Utilities = Free Unix

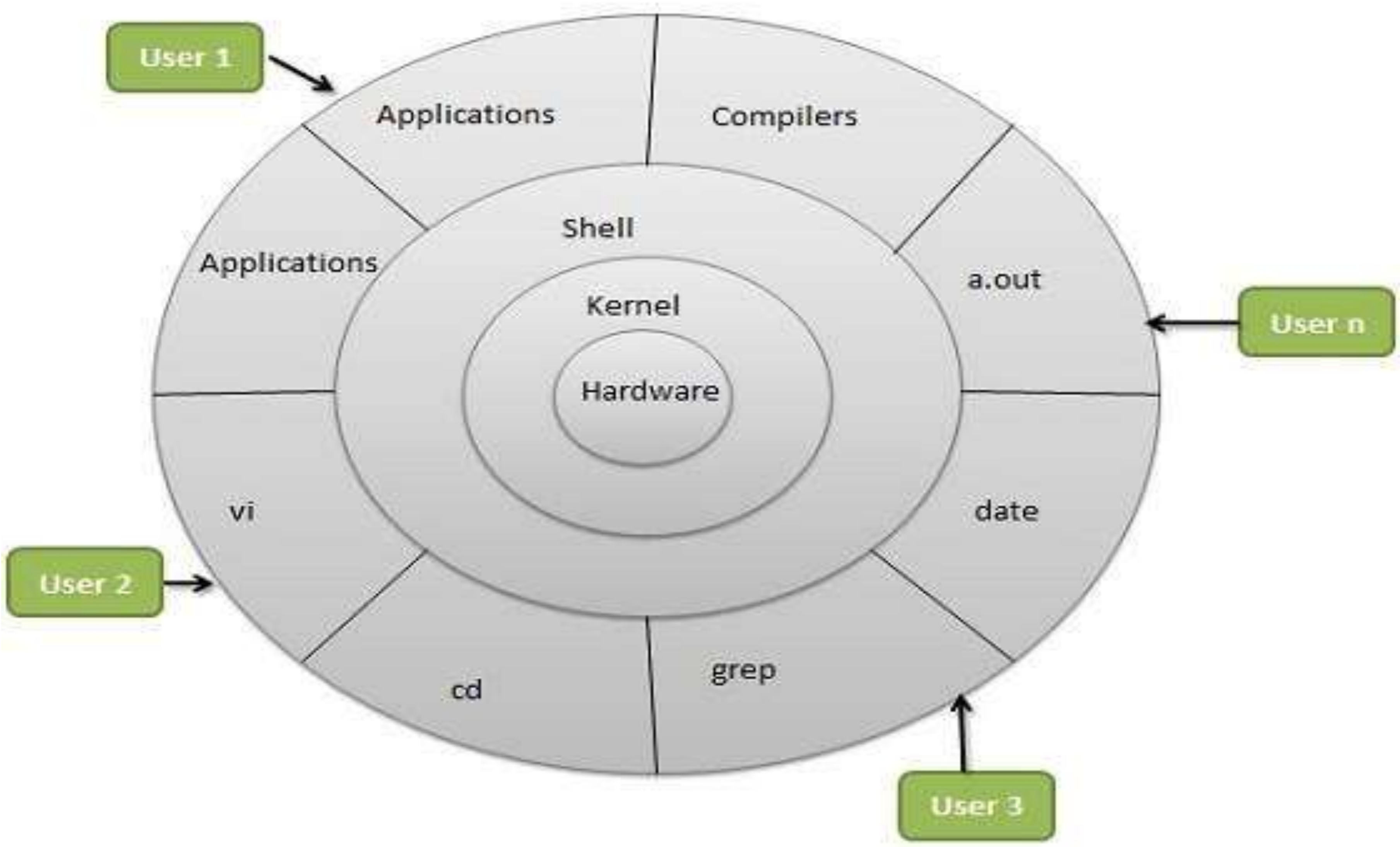
- Linux is an O/S core written by Linus Torvalds and others AND

- a set of small programs written by Richard Stallman and others. They are the GNU utilities.

http://www.gnu.org/

- Linux was orginally developed as free operating system for Intel x86 based personal computers.
- It has since been ported to more computer hardware platforms than any other operating system.
- It is a leading operating system on servers ,mainframe computers and supercomputers
- Linux also runs on embedded systems
- such as mobile phones, tablet, network routers, televisions and video games.

- The architecture of a Linux System consists of the following layers



Linux Advantages

- Low Cost
- Stability
- Performance
- Network friendliness
- Flexibility
- Compatibility
- Choice
- Fast and easy installation
- Full use of hard disk
- Multitasking
- Security

Linux OS Distributions

- Red Hat
- Ubuntu
- Debian
- SuSE
- Mandrake
- Stackware

Linux Has Many Distributions

Linux Essential Commands

pwd command

- pwd’ command prints the absolute path to current working directory.
- $ pwd

```
/home/raghu
```

date command

- Displays current time and date.
- $ date
- Fri Jul 6 01:07:09 IST 2012
- If you are interested only in time, you can use 'date +%T' (in hh:mm:ss):
- $ date +%T
- 01:13:14

Displays the calendar of the current month.

```
$ cal
July 2012
Su Mo Tu We Th Fr Sa
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

'cal ' will display calendar for the specified month and year.

```
$ cal 08 1991
August 1991
Su Mo Tu We Th Fr Sa
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

clear command

```
$ clear
```

This command clears the screen.

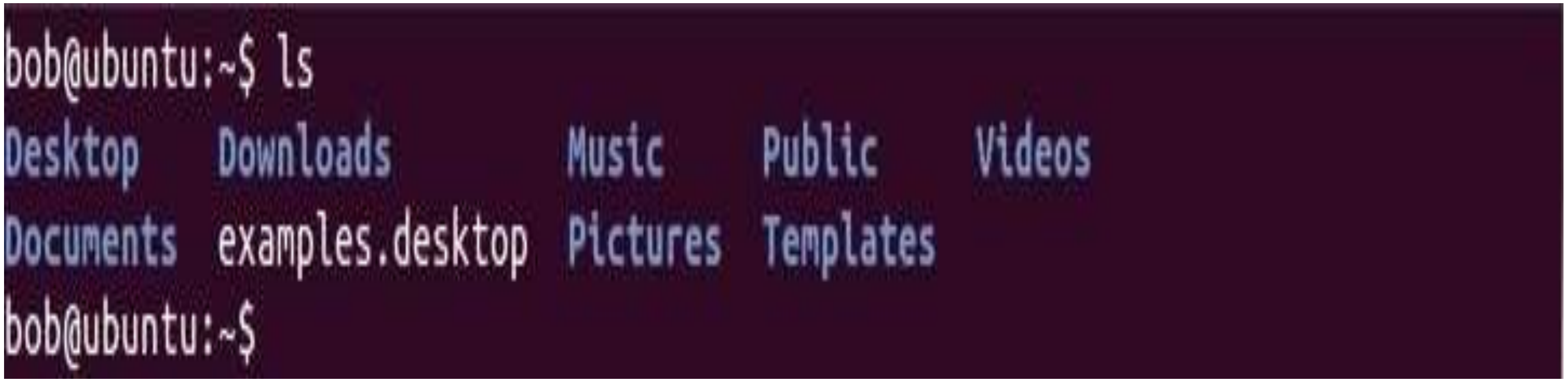
wc command(Word count)

This command counts lines, words and letters of the input given to it.

```
$ wc /etc/passwd
```

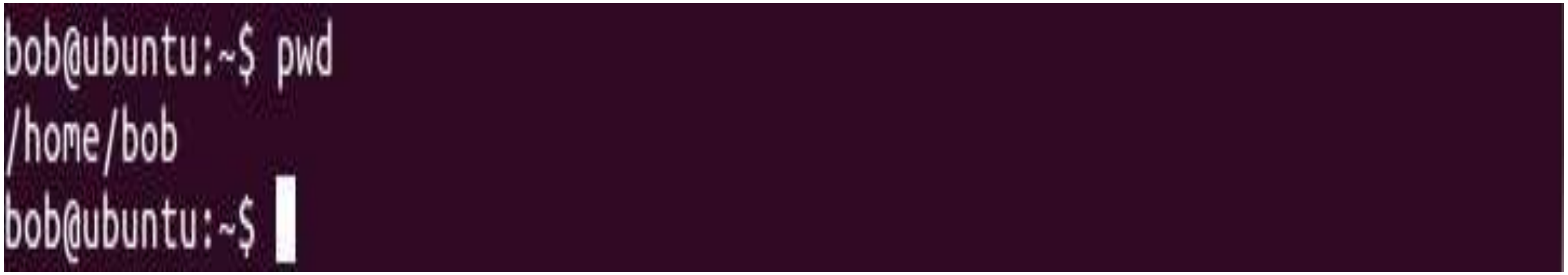
The /etc/passwd file has 35 lines, 57 words, and 1698 letters present in it.

- **ls command**
  The *ls* command lists the directory content. If no directory is specified, the command will display the content of the working directory.



```
bob@ubuntu:~$ ls
Desktop    Downloads         Music     Public     Videos
Documents  examples.desktop  Pictures  Templates
bob@ubuntu:~$
```

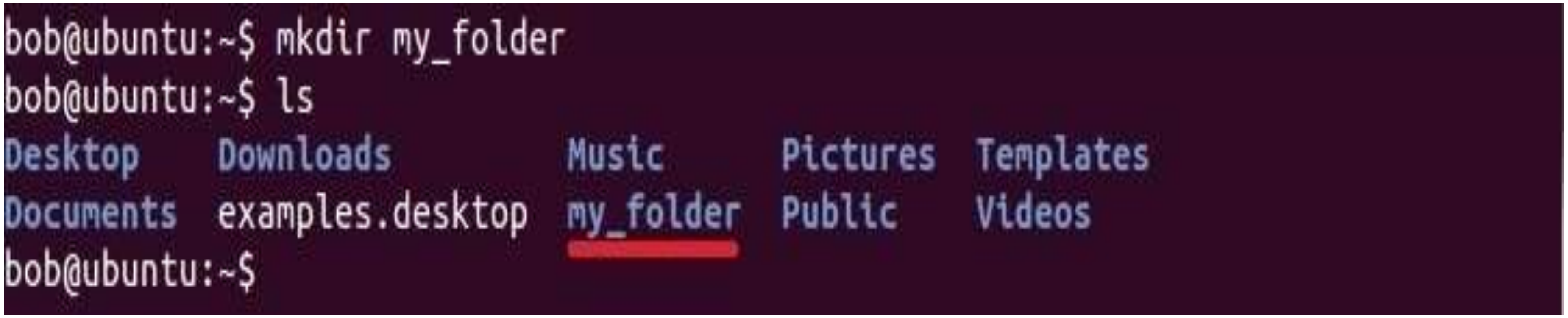
- **pwd command**
  The *pwd* command is used to print the path of the current directory.



```
bob@ubuntu:~$ pwd
/home/bob
bob@ubuntu:~$ 
```

mkdir command

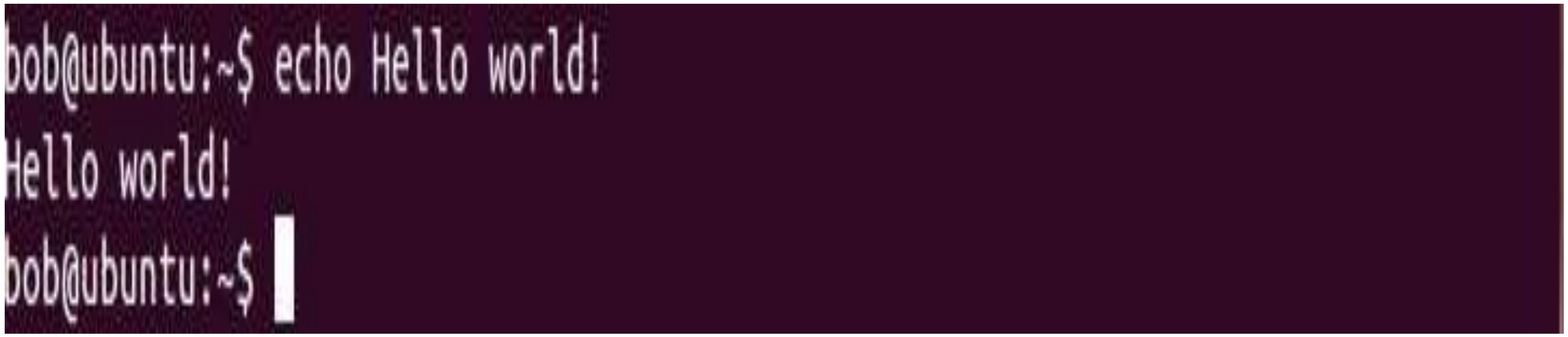
To create a new directoy, the *mkdir* command is used. You must specify the name of the directory. If no path is specified, the directory is created inside the working directory.



```
bob@ubuntu:~$ mkdir my_folder
bob@ubuntu:~$ ls
Desktop    Downloads         Music      Pictures  Templates
Documents  examples.desktop  my_folder  Public    Videos
bob@ubuntu:~$
```

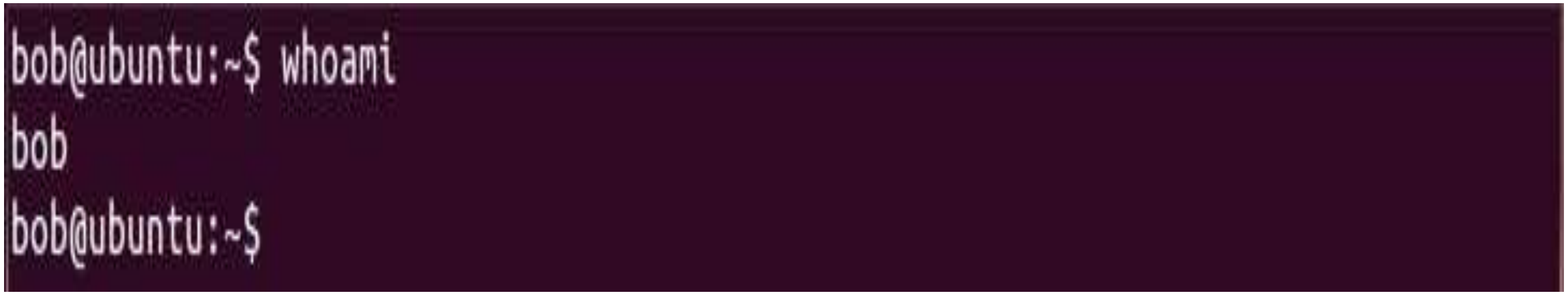
In the picture above you can see that we've created a directory called **my_folder** using the *mkdir* command. We have then displayed the content of the directory with the *ls* command.

- **echo command**
  The *echo* command is used to output text to the screen. You simply type *echo* and then the text you would like to display.



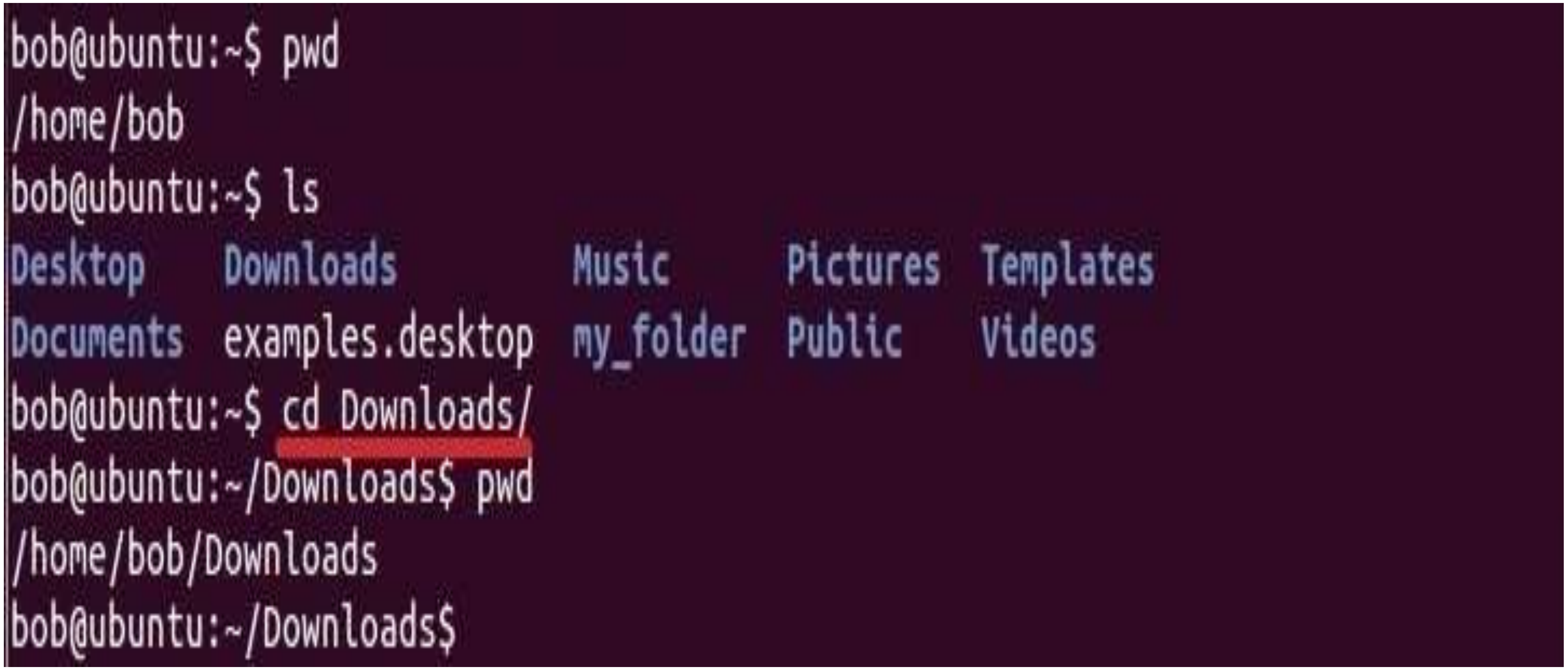
```
bob@ubuntu:~$ echo Hello world!
Hello world!
bob@ubuntu:~$
```

- **whoami command**
  The *whoami* command displays the username of the current user.



```
bob@ubuntu:~$ whoami
bob
bob@ubuntu:~$
```

Cd command
To change the current working directory we use the *cd* command. You must specify the path of the directory you would like to access.



```
bob@ubuntu:~$ pwd
/home/bob
bob@ubuntu:~$ ls
Desktop    Downloads        Music      Pictures  Templates
Documents  examples.desktop my_folder  Public    Videos
bob@ubuntu:~$ cd Downloads/
bob@ubuntu:~/Downloads$ pwd
/home/bob/Downloads
bob@ubuntu:~/Downloads$
```

cd command

- Let’s say you’re in **/home/username/Documents** and you want to go to **Photos**, a subdirectory of **Documents**.
- To do so, simply type the following command: **cd Photos**.
- Another scenario is if you want to switch to a completely new directory, for example,**/home/username/Movies**. In this case, you have to type **cd** followed by the directory’s absolute path:

  cd /home/username/Movies.
- **cd ..** (with two dots) to move one directory up
- **cd** to go straight to the home folder

ls command

- The **ls** command will display the contents of your current working directory.
- If you want to see the content of other directories, type **ls** and then the directory's path. For example, enter **ls /home/username/Documents** to view the content of **Documents**.
- There are variations you can use with the **ls** command:
- **ls -R** will list all the files in the sub-directories as well
- **ls -a** will show the hidden files
- **ls -al** will list the files and directories with detailed information like the permissions, size, owner, etc.

cat command (concatenate)

i) Cat file.txt It is used to list the contents of a file on the standard output (sdout).

Here are other ways to use the **cat** command:

ii) cat > filename creates a new file

iii) cat filename1 filename2>filename3

joins two files (1 and 2) and stores the output of them in a new file (3)

iv) to convert a file to upper or lower case use,

cat filename | tr a-z A-Z >output.txt

cp command (copy command)

- Use the **cp** command to copy files from the current directory to a different directory.
- **cp scenery.jpg /home/username/Pictures** would create a copy of **scenery.jpg** (from your current directory) into the **Pictures** directory.

mv command

- The arguments in mv are similar to the cp command. You need to type **mv**, the file’s name, and the destination’s directory.
- For example: **mv file.txt /home/username/Documents**.

mkdir command

- Use **mkdir** command to make a new directory — if you type **mkdir Music** it will create a directory called **Music**.
- There are extra **mkdir** commands as well:
- To generate a new directory inside another directory, use this Linux basic command **mkdir Music/Newfile**
- use the **p** (parents) option to create a directory in between two existing directories. For example, **mkdir -p Music/2020/Newfile** will create the new “2020” file.

rmdir command

- If you need to delete a directory, use the **rmdir** command. However, rmdir only allows you to delete empty directories.

rm command

- The **rm** command is used to delete directories and the contents within them. If you only want to delete the directory — as an alternative to rmdir — use **rm -r**.
- **Note**: Be very careful with this command and double-check which directory you are in. This will delete everything and there is no undo.

touch command

- The **touch** command allows you to create a blank new file through the Linux command line. As an example, enter touch **/home/username/Documents/Web.html** to create an HTML file entitled **Web** under the **Documents** directory.

locate command

- You can use this command to **locate** a file, just like the search command in Windows. What's more, using the **-i** argument along with this command will make it case-insensitive, so you can search for a file even if you don't remember its exact name.
- To search for a file that contains two or more words, use an asterisk **(*)**. For example, **locate -i school*note** command will search for any file that contains the word "school" and "note", whether it is uppercase or lowercase.

find command

- Similar to the **locate** command, using **find** also searches for files and directories. The difference is, you use the **find** command to locate files within a given directory.
- As an example, find **/home/ -name notes.txt** command will search for a file called **notes.txt** within the home directory and its subdirectories.
- Other variations when using the **find** are:
- To find files in the current directory use, **find . -name notes.txt**
- To look for directories use, **/ -type d -name notes. txt**

grep command

- Another basic Linux command that is undoubtedly helpful for everyday use is **grep**. It lets you search through all the text in a given file.
- To illustrate, **grep blue notepad.txt** will search for the word blue in the notepad file. Lines that contain the searched word will be

File System Concept

A file system is a logical collection of files on a partition or disk.

- Types of File Systems

The most important types of files system are

- Disk File System
- Network File System

Disk File System

- A disk file system takes advantages of the ability of disk storage media to randomly address data in a short amount of time.
- The Linux kernel supports the most popular of which are ext2, ext3 and XFS.

Network File System

- That acts on client and server providing access
- Ex. NFS, AFS, SMB Protocols.

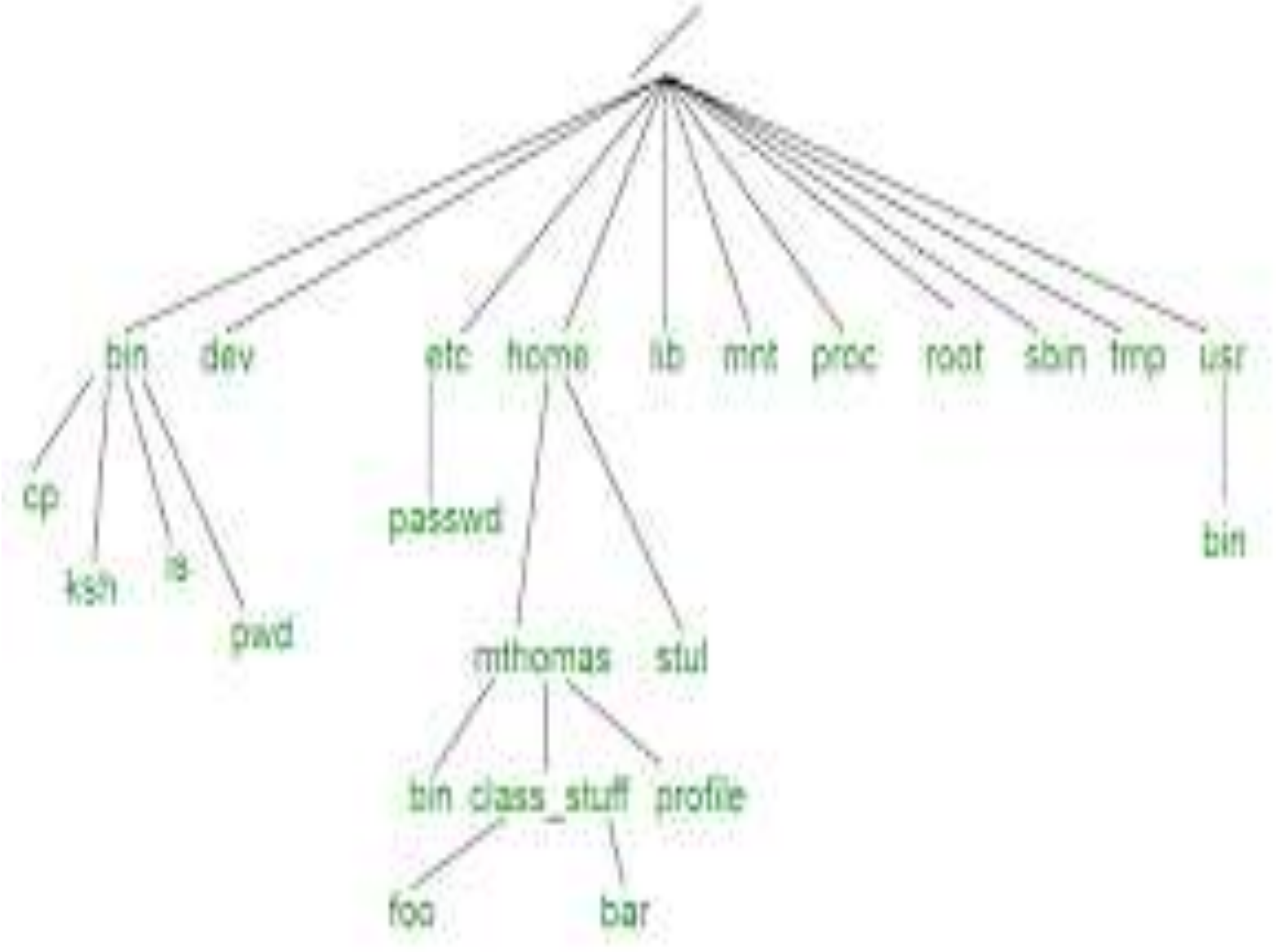
Standard Files

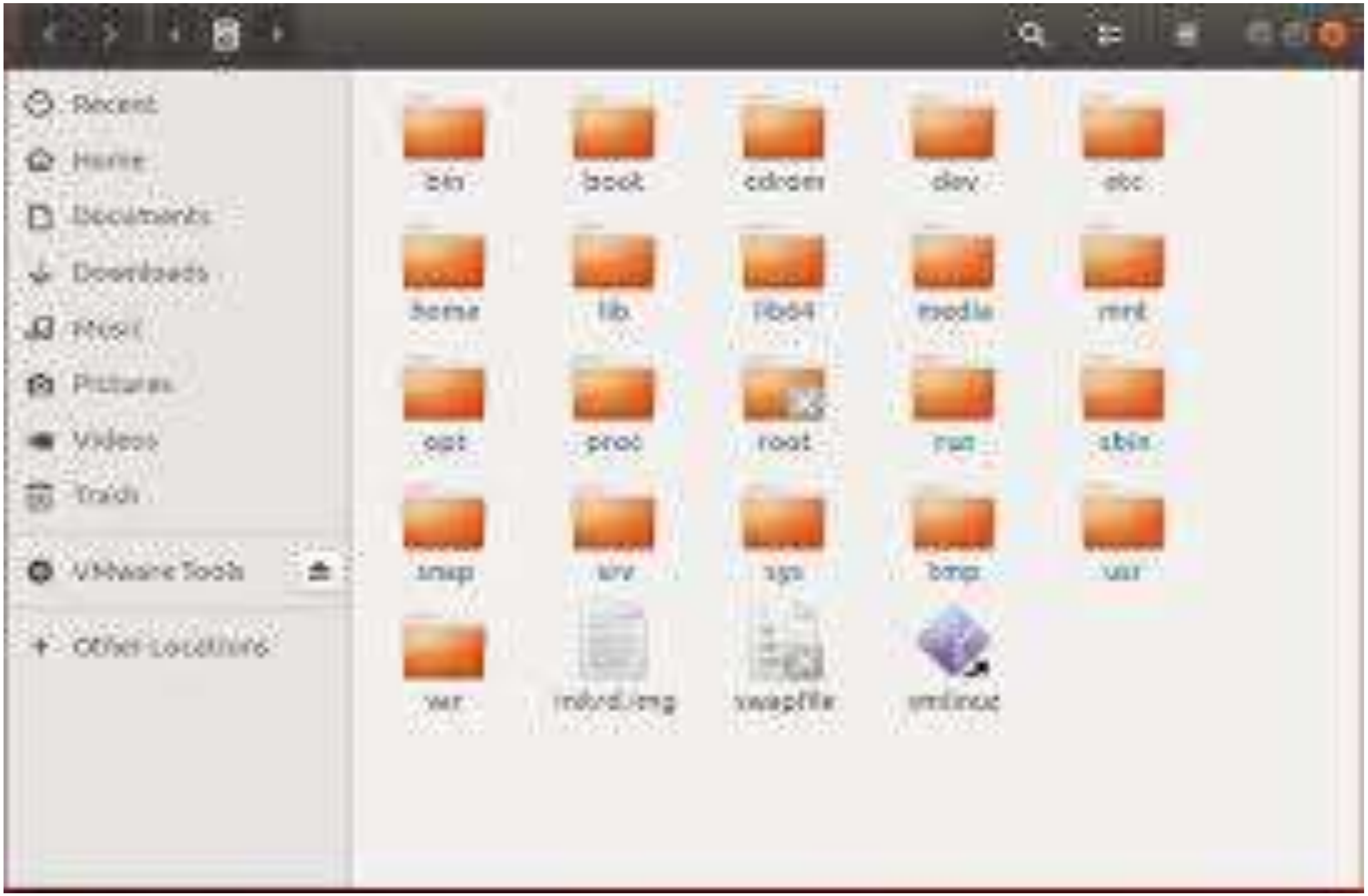
- **Ext, Ext2, Ext3, Ext4, JFS, XFS, btrfs and swap**
- **Ext**: an old one and no longer used due to limitations.
- **Ext2**: first Linux file system that allows two terabytes of data allowed.
- **Ext3**: came from Ext2, but with upgrades and backward compatibility.
- **Ext4**: faster and allow large files with significant speed.

- **JFS**: old file system made by IBM. It works very well with small and big files, but it failed and files corrupted after long time use, reports say.
- **XFS**: old file system and works slowly with small files.
- **Btrfs:** made by Oracle. It is not stable as Ext in some distros, but you can say that it is a replacement for it if you have to. It has excellent performance.
- You may notice From the comparison above that **Ext4** is the **best Linux File System.**

File System Structure (Directory Structure)

- Unix uses a hierarchical file system structure, much like an upside-down tree, with root (/) at the base of the file system and all other directories spreading from there.
- A Unix filesystem is a collection of files and directories that has the following structure.


/
bin
dev
etc
home
lib
mnt
proc
root
sbin
tmp
usr
cp
ksh
ls
pwd
passwd
bin
mthomas
stul
bin
class_stuff
profile
foo
bar


Recent
Home
Documents
Downloads
Music
Pictures
Videos
Trash
VMware Tools
Other Locations
bin
boot
cdrom
dev
etc
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
snap
srv
sys
tmp
usr
var
initrd.img
swapfile
vmlinuz

- **/bin**: Where Linux core commands reside like

  ls, mv, cp, mv, rm, cat
- **/boot**: Where boot loader and boot files are located.

  This directory contains everything required for the boot process and installer.

  Thus, the /boot directory stores data that is used before the kernel begins executing user-mode programs.

- **/dev**: Where all physical drives are mounted like USBs DVDs.
- **/etc**: Contains configurations for the installed packages
- **/home**: Where every user will have a personal folder to put his folders with his name like /home/likegeeks.
- **/lib**: Where the libraries(functions/routines) of the installed packages located since libraries shared among all packages,
- unlike Windows, you may find duplicates in different folders.
- **/media**: Here are the external devices like DVDs and USB sticks that are mounted, and you can access their files from here.

- **/mnt**: Where you mount other things Network locations and some , you may find your mounted USB or DVD.

    Temporary mount point useful for when you insert your USB stick and it gets mounted under /mnt.

- **/opt**: Some optional packages are located here and managed by the package manager.
- **/proc**: Because everything on Linux is a file, this folder for processes running on the system,

   you can access them and see much info about the current processes.

- **/root**: The home folder for the root user.
- **/sbin**: Like /bin, but binaries here are for root user only.
- **/tmp**: Contains the temporary files.
- **/usr**: Where the utilities and files shared between users on Linux.
- **/var:** Contains system logs and other variable data.

- **/tmp**

  Holds temporary files used between system boots
- **/usr**

  Used for miscellaneous purposes, and can be used by many users. Includes administrative commands, shared files, library files, and others
- **/var**

  Typically contains variable-length files such as log and print files and any other type of file that may contain a variable amount of data
- **/sbin**

  Contains binary (executable) files, usually for system administration. For example, ***fdisk*** and ***ifconfig*** utlities

- Congiguration files(config)

are files used to configure the parameters and initial settings for some computer programs. They are used for user applications, server processes and operating system settings.

Navigating the File System

Navigating to the files system

- 1 **cat filename**
- Displays a filename
- 2 **cd dirname**
- Moves you to the identified directory
- 3 **cp file1 file2**
- Copies one file/directory to the specified location
- 4 **file filename**
- Identifies the file type (binary, text, etc)
- 5 **find filename dir**
- Finds a file/directory

- **head filename**
- Shows the beginning of a file
- **less filename**
- Browses through a file from the end or the beginning
- **ls dirname**
- Shows the contents of the directory specified
- **mkdir dirname**
- Creates the specified directory
- **more filename**
- Browses through a file from the beginning to the end
- **mv file1 file2**
- Moves the location of, or renames a file/directory
- **pwd**
- Shows the current directory the user is in

- **pwd**
- Shows the current directory the user is in
- **rm filename**
- Removes a file
- **rmdir dirname**
- Removes a directory
- **tail filename**
- Shows the end of a file
- **touch filename**
- Creates a blank file or modifies an existing file or its attributes
- **whereis filename**
- Shows the location of a file
- **which filename**
- Shows the location of a file if it is in your PATH

• Linux supports multi processor computers.

4.4 Difference between MS-DOS and Linux

MS-DOS	Linux
Low cost	Free
Cannot provide complete UNIX interface	Provides complete UNIX interface
Does utilize the functional features of 80386/486 processor	Utilizes all the functional features of 80386/486 processor
Does not multitasking facility	Provides multitasking facility
Single user	Multi user

4.6 Difference between Linux and Windows

Attributes	Linux	Windows
What is it?	Linux is an example of Open Source software development and Free Operating System (OS).	Windows is the family of operating system (OS) from Microsoft, which is the most famous OS in the world.
Cost	Linux can be freely distributed.	For desktop or home use, Windows can be expensive.
Processors	Dozens of different kinds.	Intel and AMD, but WinCE runs on some additional processors. (see: WinCE)
GUI	Linux typically provides two GUIs, KDE and Gnome. But Linux GUI is optional.	The Windows GUI is an integral component of the OS and it is mainly influenced by Apple Macintosh OS and Xerox.
File system support	Ext2, Ext3, Ext4, Jfs, ReiserFS, Xfs, Btrfs, FAT, FAT32, NTFS	FAT, FAT32, NTFS,
Text mode interface	BASH (Bourne Again SHell) is the Linux default shell. It can support multiple command interpreters.	Windows uses a command shell and each version of Windows has a single command interpreter with dos-like commands, recently there is the addition of the optional PowerShell that uses more Unix-like commands.
Security	Linux has had about 60-100 viruses listed till date. None of them actively spreading nowadays.	According to Dr. Nic Peeling and Dr Julian Satchell's "Analysis of the Impact of Open Source Software" there have been more than 60,000 viruses in Windows
Examples	Ubuntu, Fedora, Red Hat, Debian, Archlinux, Android etc.	Win-98, Win-2000, Win 2007, Win-8, etc.

4.5 Difference between Linux and Unix

Attributes	Linux	Unix
What is it?	Linux is an example of Open Source software development and Free Operating System (OS).	Unix is an operating system is very popular in univer companies, big enterprises etc
Manufacturer	Linux kernel is developed by the community. Linus Torvalds oversees things.	Three biggest distributions Solaris (Oracle), AIX (IBM) HP-UX Hewlett Packard. An Apple Makes OSX, an Un based OS.
Price	Free but support is available for a price.	Three biggest distributions Solaris (Oracle), AIX (IBM) & HP-UX Hewlett Packard. And Apple Makes OSX, an Unix based OS..
Processors	Dozens of different kinds.	x86/x64, Sparc, Power, Itanium PA-RISC, PowerPC and many others.
GUI	Linux typically provides two GUIs, KDE and Gnome. But Linux GUI is optional.	Initially Unix was a command based OS, but later a GUI was created called Common Desktop Environment. Most distributions now ship with Gnome
File system support	Ext2, Ext3, Ext4, Jfs, ReiserFS, Xfs, Btrfs, FAT, FAT32, NTFS	jfs, gpfs, hfs, hfs+, ufs, xfs, zfs format
Text mode interface	BASH (Bourne Again SHell) is the Linux default shell. It can support multiple command interpreters.	Originally the Bourne Shell. Now it's compatible with many others including BASH, Korn & C.
Security	Linux has had about 60-100 viruses listed till date. None of them actively spreading nowadays.	A rough estimate of UNIX viruses is between 85 -120 viruses reported till date.
Examples	Ubuntu,,Red Hat,etc.	OS X, Solaris, All Linux

The Linux Security Model

In the UNIX System model,

- o People or processes with “root” privileges can do anything; others accounts can do much less.
- o From attackers perspective, the challenge is
- o Cracking a Linux system and to get gaining the root privileges.
- o Once that happens, attackers can erase or edit the logs files.

- Hide their processes
- Basically redefine the reality of the system like Files a and directories.
- How can such a powerful operating system with such a limited security model?

So to secure Linux systems by making careful use of native

- Linux security controls,
- plus add-on tools such as sudo or Tripwire.

However, the Linux Security model relies on **Discretionary Access Controls(DAC)**

- o In the linux DAC system,

  there are users,

  each of which belongs to one or more groups; and there are also objects: files and directories.

- o Users read, write, and execute these objects, based on the objects permissions, of which each object has three sets

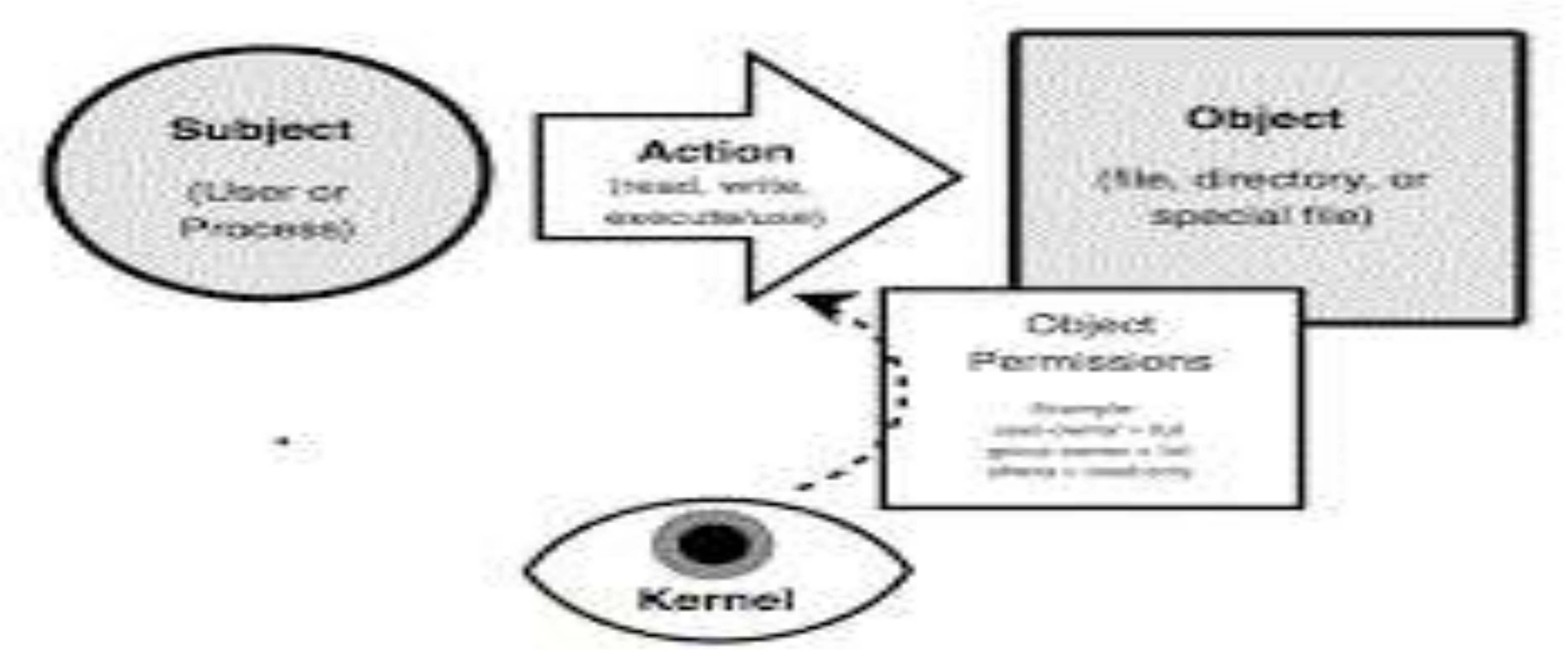
  i)permissions for the objects user-owner

  Ii)Group-owner iii) other

- These permissions are enforced by the Linux kernel, the “brain” of the operating system.
- Since process/program is actually just a file that gets copied into executable memory when run.
- When running processes attempts to

  Read, write or execute some other objects,
- The kernel will first evaluate the objects permissions against the process’s user and group identity.

- This basic transaction, a process attempts to some action read, write and execute against some object file, directory, special file is illustrated in



- Whoever owns an object can set or change its permissions.
- Herein lies the Linux DAC models weakness.
- The system super user account called “root” has the ability to both take ownership and change the permissions of all the objects in the system.

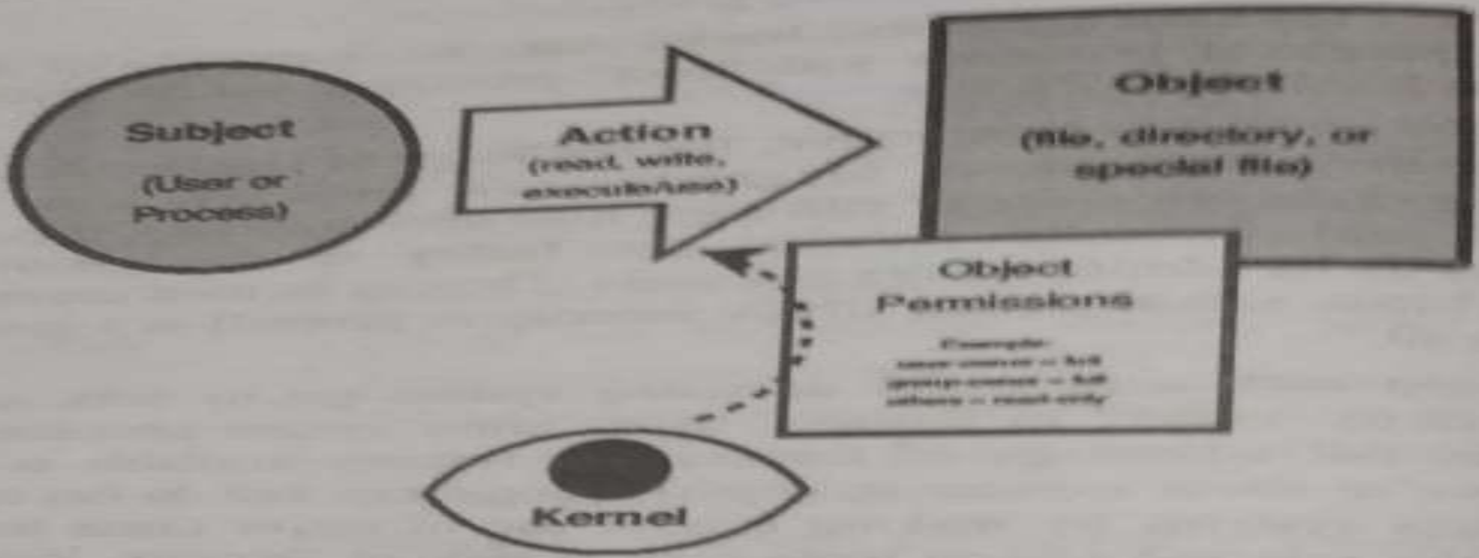


Figure — Linux Security Transactions

Whoever owns an object can set or change its permissions. Herein lies the Linux DAC model's real weakness: The system **superuser** account, called "root," has the ability to both take ownership and change the permissions of all objects in the system. And as it happens, it's not uncommon for both processes and administrator-users to routinely run with root privileges, in ways that provide attackers with opportunities to hijack those privileges. Those are the basic concepts behind the Linux DAC model. The same concepts in a different arrangement will come into play later when we examine Mandatory Access Controls such as SELinux. Now let's take a closer look at how the Linux DAC implementation actually works.

4.13Vi Editor

vi (pronounced vee-EYE, short for "visual") provides basic text editing capabilities. The vi editor is the most popular editor in linux. vi is a very powerful text editor. The editor that is installed with every linux system in existence. It runs in a standard terminal and uses the standard keyboard (it is not reliant on the arrow keys, or the insert, home, pgup keys). This makes it equally as useful on a graphics workstation as on a minimal text based terminal or ind

- all files
- directories
- running processes
- and system resources are associated with a user and group.

The security can be set

- independently for the user
- Owner
- or a group.

A third set of permissions is maintained

- for everyone on the system
- who is neither the owner nor in the group associated with a resource.
- This is commonly referred to as the permissions for 'other' users.

These security levels, user, group and other, each have a set of permissions associated with them.

- Typical permissions are **read, write and execute and depending on the type of resource,**
- these will determine what a given user is allowed to do with the resource.

- In this example, the ls -l command is used to show the ownership and permissions of the file status.sh.

- The permissions portion of the output breaks down further to indicate the level of access for the owner, group and other users.

- Each user on the system is associated with a primary group but can also belong to additional groups, adding to the flexibility of Linux security.
- A user has access rights granted to his or her primary group as well as any additional groups to which the user belongs.
- Though we use user and group names are used to interact with the system, these identifiers are tracked within the system by ID numbers.
- The user ID (UID)and group ID (GID) numbers are associated with user and group names through the /etc/passwd and /etc/group files, respectively.

- With this understanding of why users and groups are so important, now take a look at how to create, modify an remove them.

Vi Editor?

- vi is generally considered the de facto standard in Unix editors because –
- It's usually available on all the flavors of Unix system.
- Its implementations are very similar across the board.
- It requires very few resources.
- It is more user-friendly than other editors such as the **ed** or the **ex**.
- You can use the **vi** editor to edit an existing file or to create a new file from scratch. You can also use this editor to just read a text file.

vi editor

- It’s easy to invoke vi. At the command line, you type vi <filename> to either create a new file, or to edit an existing one.
- **$ vi filename.txt**
- The vi editor has two modes:

  Command and Insert.
- When you first open a file with vi, you are in Command mode.

- Command mode means that you can **use keyboard keys to navigate, delete, copy, paste, and do a number of other tasks—except entering text.**
- To enter Insert mode, press i.
- In Insert mode, you can enter text, use the **Enter** key to go to a new line,
- use the arrow keys to navigate text
- To return to Command mode, press the **Esc** key once.

Starting the vi Editor

Sr.No.	Command & Description
1	**vi filename** Creates a new file if it already does not exist, otherwise opens an existing file.
2	**vi -R filename** Opens an existing file in the read-only mode.
3	**view filename** Opens an existing file in the read-only mode.

Following is an example to create a new file **testfile** if it already does not exist in the current working directory −

- **$vi testfile**
- **The above command will generate the following output −**
- **~**
- **~**
- **~**
- **~**
- **~**
- **"testfile" [New File]**
- You will notice a **tilde** (~) on each line following the cursor.
- A tilde represents an unused line.
- You now have one open file to start working on.

Operation Modes

vi editor, we usually come across two modes

- **Command mode** – This mode enables you to perform administrative tasks such as **saving the files, executing the commands, moving the cursor, cutting and pasting the lines or words, as well as finding and replacing.** In this mode, whatever you type is interpreted as a command.

- **Insert mode** – This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and placed in the file..

- vi always starts in the **command mode**. To enter text, you must be in the insert mode for which simply type **i**. To come out of the insert mode, press the **Esc** key, which will take you back to the command mode.
- **Hint** – If you are not sure which mode you are in, press the Esc key twice; this will take you to the command mode. You open a file using the vi editor. Start by typing some characters and then come to the command mode to understand the difference.

Getting Out of vi

- The command to quit out of vi is **:q**.
- Once in the command mode, type colon, and 'q', followed by return.
- If your file has been modified in any way, the editor will warn you of this, and not let you quit. To ignore this message, the command to quit out of vi without saving is **:q!**. This lets you exit vi without saving any of the changes.
- The command to save the contents of the editor is **:w**. You can combine the above command with the quit command, or use **:wq** and return.

- The easiest way to **save your changes and exit vi** is with the ZZ command. When you are in the command mode, type **ZZ**. The **ZZ** command works the same way as the **:wq** command.

- If you want to specify/state any particular name for the file, you can do so by specifying it after the **:w**. For example, if you wanted to save the file you were working on as another filename called **filename2**, you would type **:w filename2** and return.

Moving within a File

- To move around within a file without affecting your text, you must be in the command mode (press Esc twice).

- The following table lists out a few commands you can use to move around one character at a time –

1 K

Moves the cursor up one line

2 J

Moves the cursor down one line

3 H

Moves the cursor to the left one character position

4 l

Moves the cursor to the right one character position

Editing files

- 1 **i**
- Inserts text before the current cursor location
- 2 **I**
- Inserts text at the beginning of the current line
- 3 **a**
- Inserts text after the current cursor location
- 4 **A**
- Inserts text at the end of the current line

- 5 **o**
- Creates a new line for text entry below the cursor location
- 6 **O**
- Creates a new line for text entry above the cursor location

Deleting Characters

Here is a list of important commands, which can be used to delete characters and lines in an open file −

Sr.No. Command & Description

1 **X**

Deletes the character under the cursor location

2 **X**

Deletes the character before the cursor location

3 **dw**

Deletes from the current cursor location to the next word

4 **d^**

Deletes from the current cursor position to the beginning of the line

5 **d$**

Deletes from the current cursor position to the end of the line

6 **D**

Deletes from the cursor position to the end of the current line

7 **dd**

Deletes the line the cursor is on

- As mentioned above, most commands in vi can be prefaced by the number of times you want the action to occur. For example, **2x** deletes two characters under the cursor location and **2dd** deletes two lines the cursor is on.
- It is recommended that the commands are practiced before we proceed further.

Copy and paste command

1 **yy**

Copies the current line.

2 **yw**

Copies the current word from the character the lowercase w cursor is on, until the end of the word.

3 **p**

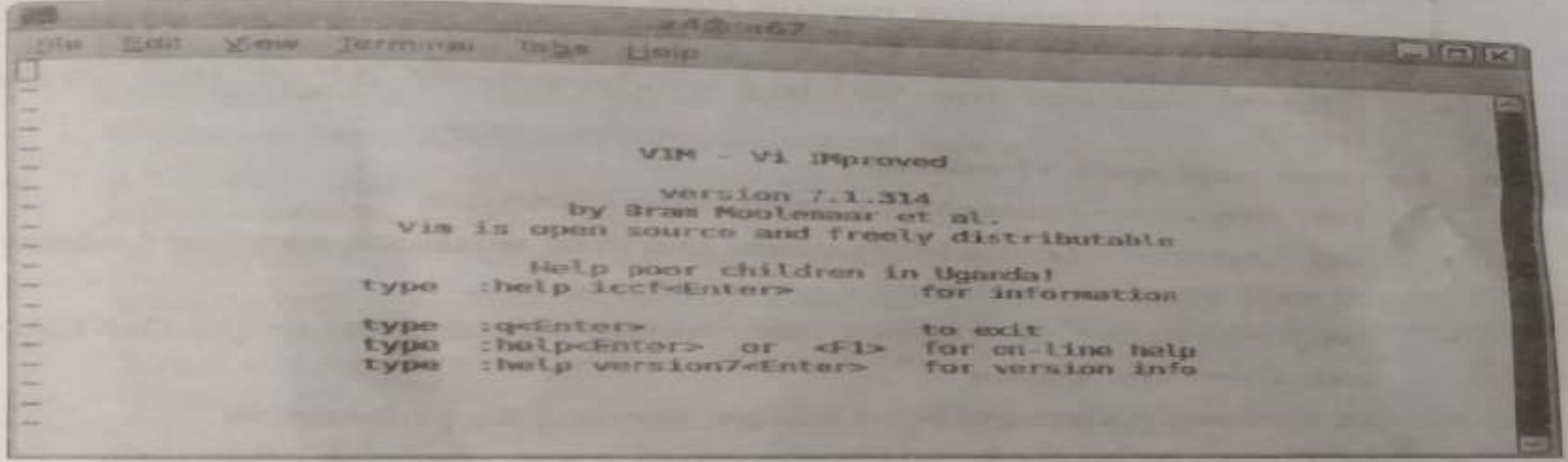
Puts the copied text after the cursor.

4 **P**

Puts the text before the cursor.

4.13.1 Getting Started

To start Vi, open a terminal or console and simply type "vi" (without the quotation marks) followed by the name of any existing file or a new file you want to create. when you open vi, but this will show you the opening screen that you otherwise wouldn't see.



```
                    VIM - Vi IMproved

                     version 7.1.314
                by Bram Moolenaar et al.
       Vim is open source and freely distributable

              Help poor children in Uganda!
type  :help iccf<Enter>           for information

type  :q<Enter>                   to exit
type  :help<Enter>  or  <F1>      for on-line help
type  :help version7<Enter>       for version info
```

4.13.2 Modes of vi Editor

Command Mode and Input Mode

Vi works in two main modes, *command mode* and *input mode*. In command mode, characters you type perform actions like moving the cursor, cutting or copying text, or searching for some particular text. In input mode, you type to insert or overwrite text. When you start vi, it is in command mode.

To switch from command to input mode, press the "i" key (you do not need to press RETURN). vi lets you insert text beginning at the current cursor location. To switch back to command mode, press ESC.

You can also use ESC to cancel an unfinished command in command mode.

Unfortunately, vi does not normally indicate which mode you are in. The next exercise turns on a mode indicator. If you are uncertain about the current mode, you can press ESC a few times. When vi beeps, you have returned to command mode.

4.13.2 Modes of vi Editor

Command Mode and Input Mode

Vi works in two main modes, *command mode* and *input mode*. In command mode, characters you type perform actions like moving the cursor, cutting or copying text, or searching for some particular text. In input mode, you type to insert or overwrite text. When you start vi, it is in command mode.

To switch from command to input mode, press the "i" key (you do not need to press RETURN). vi lets you insert text beginning at the current cursor location. To switch back to command mode, press ESC.

You can also use ESC to cancel an unfinished command in command mode.

Unfortunately, vi does not normally indicate which mode you are in. The next exercise turns on a mode indicator. If you are uncertain about the current mode, you can press ESC a few times. When vi beeps, you have returned to command mode.

How to start vi editor

- **vi** opens vi editor environment.
- **vi myfile** creates or opens the file named myfile starting at line 1.
- **vi -r myfile** recovers myfile that was being edited when system chrashed.
- **<Esc>** changes the writing environment of vi into command mode.

How to save and quit vi editor

- **:w!** <return> (enterkey) saves the changes .
- **:q!** <return > (enterkey) quits the file without saving changes to exit from the editor.
- **:wq!** <return> _rst saves the changes and then quits the file to exit from the editor.
- **:x** <return> also quits vi editor saving the changes.
- **:ZZ** also exits from vi environment saving the changes.
- **:w!** myfile <return> If the vi editor is opened but it has been given no any file name then one use this command to save the file as myfile.

Moving the Cursor in vi editor

- **w** moves the cursor at the beginning of the next word.
- **b** moves the cursor back at the beginning of the previous word
- **h** or -> moves the cursor forward .
- **l** or <- moves the cursor backward .
- **j** or <return (enter) > or # moves the cursor downward.
- **k** or " moves the cursor downward.
- **0** moves the cursor at the starting of the current line
- **$** moves the cursor at the end of the current line.
- **:0**<return> or 1G moves the cursor at the start of the first line

- :n<return> or nG moves the cursor at the start of the nth line.
- :$<return> or G moves the cursor at the start of the last line.
- :f<return> or <CTRL>G displays the current file and the number of current line at the end of the screen

Inserting text in vi editor

- i means insert before cursor. After typing i, you can insert text before cursor, until <Esc> hit.
- I means insert at the beginning of line.
- a means append after cursor. After typing a, you can insert text after cursor, until <Esc> hit.
- A means append at the end of the line.

  means open a line below. This command opens and put text in a new line below current line, until <Esc> hit. means line above. This command opens and put text in a new line below current line, until <Esc> hit.
- s means substitute a character.
- S means substitute entire line

Changing text in vi editor

- r: After typing r, you can replace a single character under the cursor.

  No need to hit <Esc>.
- R: After typing R, you can replace characters starting with current with the letter under the cursor until you hit <Esc>.
- cc: After typing cc, change (replace) the current entire line, stops when you hit <Esc>.
- C: After typing C, change (replace) the characters in the current line, until you hit <Esc>.
- u: u is used to undo your last action on the vi editor.
- <CTRL>r: <CTRL>r is used to redo the changes which were undone.

- **x** deletes a single character under the cursor. No need to hit <Esc>.
- **dw** deletes a word, no need to hit <Esc>.
- **dd** deletes entire line, no need to hit <Esc>.
- **ndd** deletes n lines, no need to hit <Esc>.
- **D** deletes the remainder of the line starting with the cursor.

Copying and pasting the text

- **yy** copies the current line.
- **nyy** copies n lines including the correct line.
- **p** puts and paste the copied line after the line where the cursor is.

Moving and manipulating the screen of vi editor

- **<CTRL>f** scrolls the screen on times forward(downward).
- **<CTRL>b** scrolls the screen on times backward(upward).
- **<CTRL>d** scrolls the screen on times downward(forward).
- **<CTRL>u** scrolls the screen on times upward(backward).

4.14 Partitions Creation

In most Linux systems, you can use the fdisk utility to create a new partition and to do other disk management operations.

Note: To be able to execute the commands necessary to create a new partition on Linux, you must have the root privileges.

As a tool with a text interface, fdisk requires typing the commands on the fdisk command line. The following fdisk commands may be helpful:

Options	*Description*
M	Displays the available commands.
P	Displays the list of existing partitions on your hda drive. Unpartitioned space is not listed.
N	Creates a new partition.
Q	Exits fdisk without saving your changes.
L	Lists partition types.
W	Writes changes to the partition table.

4.14 Partitions Creation

In most Linux systems, you can use the fdisk utility to create a new partition and to do other disk management operations.

Note: To be able to execute the commands necessary to create a new partition on Linux, you must have the root privileges.

As a tool with a text interface, fdisk requires typing the commands on the fdisk command line. The following fdisk commands may be helpful:

Options	*Description*
M	Displays the available commands.
P	Displays the list of existing partitions on your hda drive. Unpartitioned space is not listed.
N	Creates a new partition.
Q	Exits fdisk without saving your changes.
L	Lists partition types.
W	Writes changes to the partition table.

To create a new partition on Linux

1. Start a terminal.
2. Start fdisk using the following command:

   /sbin/fdisk /dev/hda

   where /dev/hda stands for the hard drive that you want to partition.

. In fdisk, to create a new partition, type the following command:

   n

   - When prompted to specify the **Partition type**, type p to create a primary partition or e to create an extended one. There may be up to four primary partitions. If you want to create more than four partitions, make the last partition extended, and it will be a container for other logical partitions.
   - When prompted for the **Number**, in most cases, type 3 because a *typical* Linux virtual machine has two partitions by default.
   - When prompted for the **Start cylinder**, type a starting cylinder number or press **Return** to use the first cylinder available.
   - When prompted for the **Last cylinder**, press **Return** to allocate all the available space or specify the size of a new partition in cylinders if you do not want to use all the available space.

ystem ID of 83. If you're unsure of the partition's **System ID**, use the command to check it.

4. Use the command to write the changes to the partition table.

   w

5. Restart the virtual machine by entering the command.

   reboot

6. When restarted, create a file system on the new partition. We recommend that you use the same file system as on the other partitions. In most cases it will be either the Ext3 or ReiserFS file system. For example, to create the Ext3 file system, enter the following command:

   /sbin/mkfs -t ext3 /dev/hda3

7. Create a directory that will be a mount point for the new partition. For example, to name it data, enter:

   mkdir /data

8. Mount the new partition to the directory you have just created by using the following command:

   ```
   mount /dev/hda3 /data
   ```

9. Make changes in your static file system information by editing the /etc/fstab file in any of the available text editors. For example, add the following string to this file:

   ```
   /dev/hda3 /data ext3 defaults 0 0
   ```

   In this string /dev/hda3 is the partition you have just created, /data is a mount point for the new partition, Ext3 is the file type of the new partition. For the exact meaning of other items in this string, consult the Linux documentation for the mount and fstab commands.

10. Save the /etc/fstab file.

4.15 Introduction to Shells

Shell acts as an interface between the user and the kernel. It is the command interpreter of the OS. The shell

Partitioning with fdisk

- This section shows you how to actually partition your hard drive with the **fdisk** utility.
- Linux allows only 4 primary partitions. You can have a much larger number of logical partitions by sub-dividing one of the primary partitions. Only one of the primary partitions can be sub-divided.

fdisk usage

- **fdisk** is started by typing (as root)
- **fdisk *device*** at the command prompt.
- *device* might be something like /dev/hda or /dev/sda .

The basic **fdisk** commands you need are:

- **p** print the partition table
- **n** create a new partition
- **d** delete a partition
- **q** quit without saving changes
- **w** write the new partition table and exit

- **Four primary partitions**
- Divide up the remaining space for the three other partitions.
- Example:
- I start fdisk from the shell prompt:
- **fdisk /dev/hdb**

Shell Scripting

A shell script is a computer program designed to be run by the Unix/Linux shell which could be one of the following:

- **The Bourne Shell**
- **The C Shell**
- **The Korn Shell**

A shell is a command-line interpreter and typical operations performed by shell scripts include file manipulation, program execution, and printing text.

4.15 Introduction to Shells

Shell acts as an interface between the user and the kernel. It is the command interpreter of the OS. The shell is an integral part of Linux, and is part of the design of Linux and UNIX. The commands given from the user are moved to the shell. The shell analyses and interprets these commands into the machine understandable form. The commands can be either typed in through the command line or contained in a file called shell script. Hence,

(i) **Bourne Shell:** It is developed by Steve Bourne at AT&T Bell Labs for Unix V7 (1979). Small, simple, and (originally) very few internal commands, so it called external programs for even the simplest of tasks. It is always available on everything that looks vaguely like Unix.

(ii) **Korn shell:** It is developedby David Korn of AT&T Bell Labs. Written as a major upgrade to "sh" and backwards compatible with it, but has many internal commands for the most frequently used functions. It also incorporates many of the features from tcsh, which enhance interactive use (command line history recall etc.).

(iii) **C Shell:** It is developed by Bill Joy at Berkeley (who went on to found Sun Microsystems). Many things in common with the Bourne shell, but many enhancements to improve interactive use. The internal commands used only in scripts are very different from "sh", and similar (by design) to the C" language syntax

4.15.1 Features of Shell

- All communications between user and Kernel takes place through the shell.
- It allows the tasks to run on background.
- It also enables us to construct scripts like a programming language.
- A group of files can be executed using a single command.

4.15.2 Scripting Basics

Shell scripts are text files that contain a series of commands or statements to be executed. Shell scripts are useful for:

- Automating commonly used commands.
- Performing system administration and troubleshooting.
- Creating simple applications.
- Manipulation of text or files.

Why use Shell Scripts

- Combine lengthy and repetitive sequences of commands into a single, simple command.
- Generalize a sequence of operations on one set of data, into a procedure that can be applied to any similar set of data. Create new commands using combinations of utilities in ways the original authors never thought of.
- Simple shell scripts might be written as shell aliases, but the script can be made available to all users and all processes. Shell aliases apply only to the current shell.
- Wrap programs over which you have no control inside an environment that you can control.
- Create customized datasets on the fly, and call applications (e.g. matlab, sas, idl,gnuplot) to work on them, or create customized application commands/procedures.
- Rapid prototyping (but avoid letting prototypes become

How to use Shell Scripting?

- Create a file using a vi editor(or any other editor).
- Name **script** file with extension .**sh**.
- Start the **script** with #! /bin/**sh**.
- Write some code.
- Save the **script** file as filename.**sh**.
- For executing
  the **script** type **bash** filename.**sh**.

- **Ex.**The following script uses the **read** command which takes the input from the keyboard and assigns it as the value of the variable PERSON and finally prints it on STDOUT.

```
#!/bin/sh
echo "What is your name?"
read PERSON
echo "Hello, $PERSON“
```

Here is a sample run of the script –

```
$./test.sh
What is your name?
 Zara Ali
Hello, Zara Ali $
```

What are Shell Variables?

Variables store data in the form of characters and numbers. Similarly, Shell variables are used to store information and they can by the shell only.

For example, the following creates a shell variable and then prints it:

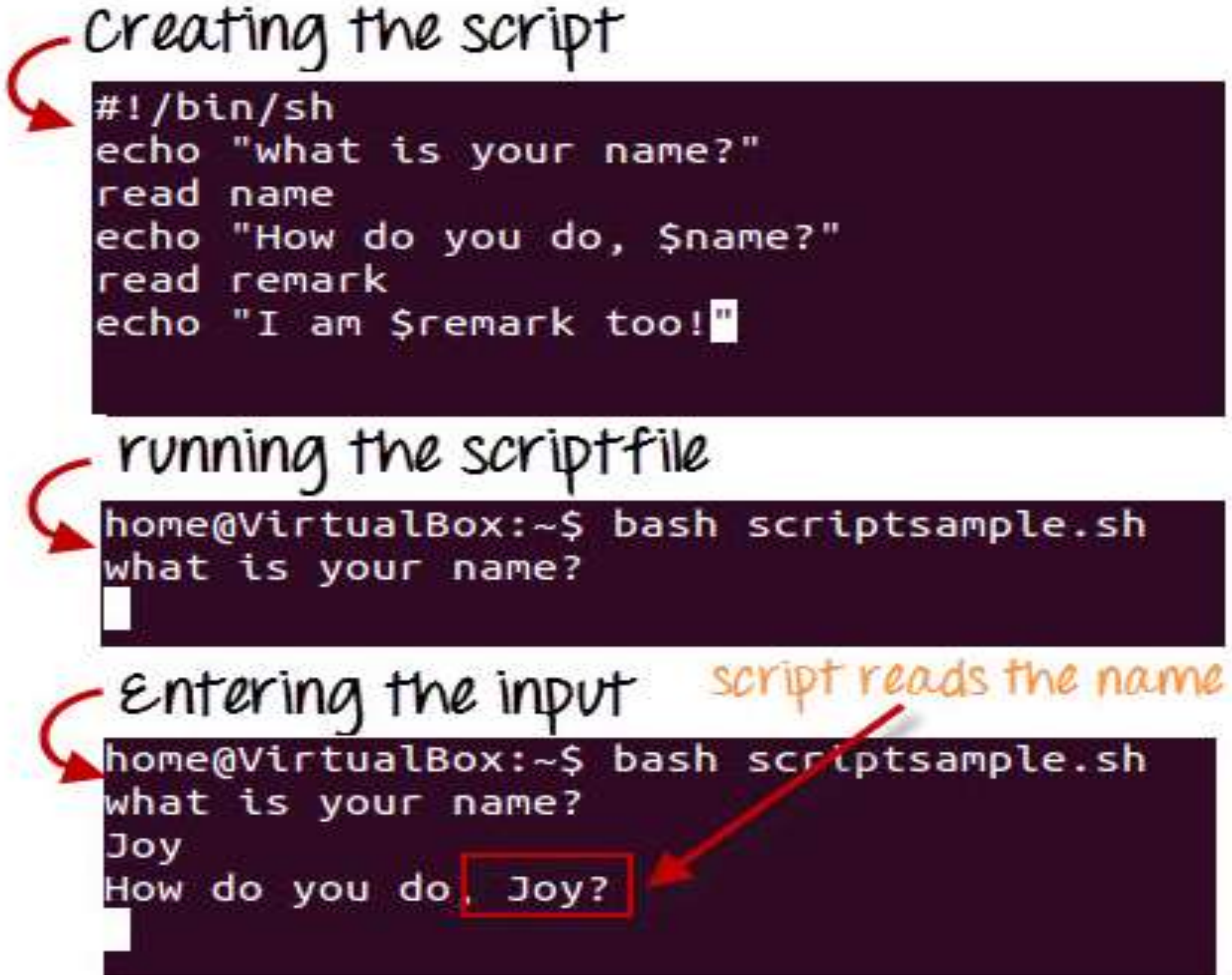
```
variable ="Hello"
echo $variable
```

Below is a small script which will use a variable.

```
#!/bin/sh
echo "what is your name?"
read name
echo "How do you do, $name?"
read remark
echo "I am $remark too!"
```

```
read remark
echo "I am $remark too!"
```

Let's understand, the steps to create and execute the script



```
#!/bin/sh
echo "what is your name?"
read name
echo "How do you do, $name?"
read remark
echo "I am $remark too!"
```

```
home@VirtualBox:~$ bash scriptsample.sh
what is your name?
```

```
home@VirtualBox:~$ bash scriptsample.sh
what is your name?
Joy
How do you do, Joy?
```

Shell Operators

There are many operators in Shell Script some of them are discussed based on string.

Equal operator (=): This operator is used to check whether two strings are equal.

Syntax:

```
Operands1 = Operand2
#!/bin/sh
str1="GeeksforGeeks";
str2="geeks";
if [ $str1 = $str2 ]
  then
  echo "Both string are same";
  else
  echo "Both string are not same";
fi
```

Output: Both string are not same

.

Not Equal operator (!=): This operator is used when both operands are not equal.

Syntax:

```
Operands1 != Operands2
```

Example:

```
#!/bin/sh
str1="GeeksforGeeks";
str2="geeks";
if [ $str1 != $str2 ]
then
   echo "Both string are not same";
else
   echo "Both string are same";
fi
```

Output:

```
Both string are not same
```

Less then (\<): It is a conditional operator and used to check operand1 is less then operand2.

Syntax: Operand1 \< Operand2

Example:

```
 #!/bin/sh
str1="GeeksforGeeks";
str2="Geeks";
if [ $str1 \< $str2 ]
then
   echo "$str1 is less then $str2";
else
   echo "$str1 is not less then $str2";
fi
```

Output: GeeksforGeeks is not less then Geeks

Greater then (\>): This operator is used to check the operand1 is greater then operand2.

Syntax:
Operand1 \> Operand2

Example:

```
#!/bin/sh
str1="GeeksforGeeks";
str2="Geeks";
if [ $str1 \> $str2 ]
then
   echo "$str1 is greater then $str2";
else
   echo "$str1 is less then $str2";
fi
```

Output:

```
GeeksforGeeks is greater then Geeks
```

Check string length greater then 0: This operator is used to check the string is not empty.

Syntax:

```
[ -n Operand ]
```

Example:

```
#!/bin/sh
str="GeeksforGeeks";
if [ -n $str ]
then
   echo "String is not empty";
else
   echo "String is empty";
fi
```

Output:

```
String is not empty
```

Check string length equal to 0: This operator is used to check the string is empty.

Syntax:

```
[ -z Operand ]
```

Example:

```
#!/bin/sh
 str="";
if [ -z $str ]
then
   echo "String is empty";
else
   echo "String is not empty";
fi
```

Output:

```
String is empty
```

Conditional Statements | Shell Script

Conditional Statements: There are total 5 conditional statements which can be used in bash programming

if statement

if-else statement

if..elif..else..fi statement (Else If ladder)

if..then..else..if..then..fi..fi..(Nested if)

switch statement

Their description with syntax is as follows:

if statement

This block will process if specified condition is true.

Syntax:

```
if [ expression ]then   statement   fi
```

if-else statement

If specified condition is not true in if part then else part will be execute.

Syntax

```
if [ expression ]then
Statement1
else   statement2
fi
```

if..elif..else..fi statement (Else If ladder)
To use multiple conditions in one if-else block, then elif keyword is used in shell. If expression1 is true then it executes statement 1 and 2, and this process continues. If none of the condition is true then it processes else part.
Syntax

```
 if [ expression1 ]then   statement1   statement2   .
 elif [ expression2 ]
then
statement3
 statement4
  .   .
else
Statement5
fi
```

switch statement

case statement works as a switch statement if specified value match with the pattern then it will execute a block of that particular pattern.
When a match is found all of the associated statements until the double semicolon (;;) is executed.
A case will be terminated when the last command is executed.
If there is no match, the exit status of the case is zero.

Syntax:

```
case  in
Pattern  1)
        Statement 1;;
Pattern  n)
         Statement n;;
esac
```

Implementing switch statement

```
CARS="bmw“
#Pass the variable in string
case "$CARS" in
   #case 1
   "mercedes") echo "Headquarters - Affalterbach,
Germany" ;;
   #case 2
   "audi") echo "Headquarters - Ingolstadt, Germany" ;;
   #case 3
   "bmw") echo "Headquarters - Chennai, Tamil Nadu,
India" ;;
esac
```

Output

```
$bash -f main.sh
Headquarters - Chennai, Tamil Nadu, India.
```

```
#Initializing two variables
a=10
b=20
#Check whether they are equal
if [ $a == $b ]
then
   echo "a is equal to b"
fi

#Check whether they are not equal
if [ $a != $b ]
then
   echo "a is not equal to b"
fi
```

Output

```
$bash -f main.sh a is not equal to b
```

```
#Initializing two variables
   a=20
   b=20
   if [ $a == $b ]
   then
  #If they are equal then print this
        echo "a is equal to b"
        else
   #else print this
         echo "a is not equal to b"
        fi
```

Output

```
$bash -f main.sh a is equal to b
```

Basic String Operations

String Length

```
STRING="this is a string"
echo ${#STRING}          # 16
```

Substring Extraction

Extract substring of length $LEN from $STRING starting after position $POS. Note that first position is 0.

```
STRING="this is a string"
POS=1
LEN=3
echo ${STRING:$POS:$LEN}   # his
```

- **wc command**
- wc stands for **word count**.
- It is used to find out **number of lines**, **word count**, **byte and characters count** in the files specified in the file arguments.
- First column shows number of lines present in a file specified, second column shows number of words present in the file, third column shows number of characters present in file and fourth column itself is the file name which are given as argument.
- **Syntax:**
- **wc [OPTION]... [FILE]...**

- Let us consider two files having name **state.txt** and **capital.txt** containing 5 names of the Indian states and capitals respectively.
- **$ cat state.txt**
- Andhra Pradesh
- Arunachal Pradesh
- Assam
- Bihar
- Chhattisgarh
- **$ cat capital.txt**
- Hyderabad
- Itanagar
- Dispur
- Patna
- Raipur

- **Passing only one file name in the argument.**
- **$ wc state.txt**
- 5 7 63 state.txt
- OR
- **$ wc capital.txt**
- 5 5 45 capital.txt
- **Passing more than one file name in the argument.**
- $ wc state.txt capital.txt
- 5 7 63 state.txt
- 5 5 45 capital.txt
- 10 12 108 total

4.15.10.2 Repetition or iteration structure

Loops are powerful programming tool that enables you to execute a set of commands repeatedly. There are three types of repetition or iteration or looping structures for carrying out commands more than once:

- while
- for
- until

while

The while command allows you to define a command to test, then loop through a set of commands for as long as the defined test command returns a zero exit status. It tests the test command at the start of each iteration. When the test command returns a non-zero exit status, the while commands stops executing the set of commands. The body of the loop is put between do and done. The general format is

```
while [ condition ]
do
        command1
        ....
done
```

Example

```
    ....
done
```

Example

Here is a simple example that uses the while loop to display the numbers zero to nine:

```
#!/bin/bash
a=0
while [ $a -lt 10 ]
do
  echo $a
  a=`expr $a + 1`
done
```

This will produce following result:

```
0 1 2 3 4 5 6 7 8 9
```

for

The for command to allow you to create a loop that iterates through a series of values. Each iteration performs a defined set of commands using one of the values in the series. The basic format of the bash shell for command is:

```
for variable in list
do
        command(s)
done
```

Example

Here is a simple example that uses for loop to span through the give list of numbers:

```
#!/bin/bash
for var in 0 1 2 3 4 5 6 7 8 9
do
  echo $var
done
```

This will produce following result:

```
0 1 2 3 4 5 6 7 8 9
```

until

1 2 3 4 5 6 7 8 9

until

The until command works exactly the opposite way from the while command. The until command requires that you to specify a test command that normally produces a non-zero exit status. As long as the exit status of the test command is non-zero, the bash shell executes the commands listed in the loop. Once the test command returns a zero exit status, the loop stops.

The format of the until loop is:

```
until condition
do
        command(s)
done
```

Example

Here is a simple example that uses the until loop to display the numbers zero to nine

```
#!/bin/bash
a=0
until [ ! $a -lt 10 ]
do
  echo $a
  a=`expr $a + 1`
done
```

4.16 String processing

A *string* is any finite sequence of *characters* (i.e., letters, numerals, symbols and punctuation marks). In Linux and other Unix-like operating systems, there are numerous utilities for processing strings. These are the same utilities that are used for manipulating *text files*(i.e., files that contain only text and no binary data), because in such operating systems text files and strings are considered to be essentially the same thing. The most common basic string processing utilities used in linux are

- wc
- sort
- uniq
- tr
- cut
- paste

4.16.1 String processing commands

4.16.1 String processing commands

wc

Short for word count, wc displays a count of lines, words, and characters in a file

Syntax

```
wc [-c | -m | -C ] [-l] [-w] [ file ... ]
```

-c	Count bytes.
-m	Count characters.
-C	Same as -m.
-l	Count lines.
-w	Count words delimited by white space characters or new line characters. Delimiting characters are Extended Unix Code (EUC) characters from any code set defined by iswspace()
File	Name of file to word count.

Examples

wc myfile.txt - Displays information about the file myfile.txt. Below is an example of the output.

```
5 13 57 myfile.txt
```

5 = Lines 13 = Words 57 = Characters

sort

sort command is used to sort the lines in a text file

Syntax

```
sort [options] filename
```

Options

-r	Sorts in reverse order.
-u	If line is duplicated display only once.
-o filename	Sends sorted output to a file.

Example

1. sort test.txt

Sorts the 'test.txt' file and prints result in the screen.

2. sort -r test.txt

Sorts the 'test.txt' file in reverse order and prints result in the screen.

uniq

Examples

uniq myfile1.txt > myfile2.txt - Removes duplicate lines in the first file1.txt and outputs the results to the second file.

tr

tr command is used to translate character.

Syntax

```
tr [-c] [-d] [-s] [string1] [string2]
```

-c	Complement the set of characters specified by string1.
-d	Delete all occurrences of input characters that are specified by string1.
-s	Replace instances of repeated characters with a single character.
string1	First string or character to be changed.
string2	Second string or character to change the string1.

Example

```
echo "12345678 9247" | tr 123456789 computerh
```

This example takes an echo response of '12345678 9247' and pipes it through the tr replacing the appropriate numbers with the letters. In this example it would return *computer hope*.

cut

cut command is used to cut out selected fields of each line of a file. The cut command uses delimiters to determine where to split fields.

Syntax

```
cut [options]
```

Options

-c	Specifies character positions.
-b	Specifies byte positions.
-d flags	Specifies the delimiters and fields

Example

1. cut -c1-3 text.txt

Output:

```
Thi
```

Cut the first three letters from the above line.

paste

paste

paste command is used to paste the content from one file to another

It is also used to set column format for each line.

Syntax

```
paste [options]
```

Options

-s	Paste one file at a time instead of in parallel.
-d	Reuse characters from LIST instead of TABs

Example

1. paste test.txt>test1.txt

   Paste the content from 'test.txt' file to 'test1.txt' file.

4.17 Investigation and Managing Processes

wc command in Linux with examples

Last Updated: 27-05-2019

wc stands for **word count**. As the name implies, it is mainly used for counting purpose.

- It is used to find out **number of lines**, **word count**, **byte and characters count** in the files specified in the file arguments.
- By default it displays **four-columnar output.**
- First column shows number of lines present in a file specified, second column shows number of words present in the file, third column shows number of characters present in file and fourth column itself is the file name which are given as argument.

Syntax:

```
wc [OPTION]... [FILE]...
```

Let us consider two files having name **state.txt** and **capital.txt** containing 5 names of the Indian states and capitals respectively.

```
$ cat state.txt
Andhra Pradesh
Arunachal Pradesh
Assam
Bihar
Chhattisgarh

$ cat capital.txt
Hyderabad
Itanagar
Dispur
Patna
Raipur
```

Passing only one file name in the argument.

```
$ wc state.txt
 5  7 63 state.txt
       OR
$ wc capital.txt
 5  5 45 capital.txt
```

Passing more than one file name in the argument.

```
$ wc state.txt capital.txt
  5   7  63 state.txt
  5   5  45 capital.txt
 10  12 108 total
```

INVESTIGATION AND MANAGING PROCESS

What is process?

A process is an executing with several components and properties including a memory context, priority, and environment. The Linux kernel tracks every aspect of a process by its PID under /proc/PID.

Viewing All Processes

ps command is used to report the process status. ps is the short name for Process Status.

Syntax

ps [options]

Options

-a	List information about all processes most frequently requested: all those except process group leaders and processes not associated with a terminal..
-A or e	List information for all processes.
-d	List information about all processes except session leaders.
-e	List information about every process now running.
-f	Generates a full listing.
-j	Print session ID and process group ID.
-l	Generate a long listing.

Example

1. ps

Output:

```
PID TTY TIME CMD
2540 pts/1 00:00:00 bash
2621 pts/1 00:00:00 ps
```

In the above example, typing ps alone would list the current running processes.

2. ps -f

Output:

```
UID PID PPID C STIME TTY TIME CMD
rizwan 2540 2536 0 15:31 pts/1 00:00:00 bash
rizwan 2639 2540 0 15:51 pts/1 00:00:00 ps -f
```

Displays full information about currently running processes.

Process States

Every process has a *state* property, which describes whether the process is actively using the cpu (*Running*), in memory but not doing anything (*Sleeping*), waiting for a resource to become available (*Uninterruptable Sleep*) or terminated, but not flushed from the process list (*Zombie*). There are other process states listed in man ps, but the four described above are the most common. Of these, Running and Sleeping are normal, but the presence of Uninterruptable Sleep or Zombie processes may indicate problems lurking on your system.

Uninterruptable sleep: Process is sleeping and cannot be woken up until an event occurs. It cannot be woken up by a signal. Typically, the result of an I/O operation, such as a failed network connection (for NFS hard mounts).

Zombie: Just before a process dies, it sends a signal to its parent and waits for an acknowledgment before terminating. Even if the parent process does not immediately acknowledge this signal, all resources except for the process identity number (PID) are released. Zombie processes are cleared from the system during the next system reboot and do not adversely affect system performance.

Signals

Sending Signals

kill can send many signals, but processes only respond to the signals they have been programmed to recognize.

If a process will not respond to a TERM signal, the KILL signal can be used. However, the process may not be ended cleanly. The KILL signal should be used only if a process will not respond to a *Ctrl-C* or a TERM signal. Using KILL signals on a routine basis may cause zombie processes and lost data.

The following are all identical and will send the default TERM signal to the process with PID number 3428:

```
[student@stationX ~]$ kill 3428
[student@stationX ~]$ kill -15 3428
[student@stationX ~]$ kill -SIGTERM 3428
[student@stationX ~]$ kill -TERM 3428
```

Schedule priorities

Every running process has a scheduling priority: a ranking among running processes determining which should get the attention of the processor. The formula for calculating this priority is complex, but users can affect the priority by setting the "niceness" value, one element of this complex formula. The niceness value defaults to zero but can be set from -20 (least nice, highest priority) to 19 (most nice, lowest priority).

Adjusting priorities for programs

To set the niceness value to a specific value when starting a process, use nice -n:

```
[student@stationX ~]$ nice -n 15 myprog
```

Altering priorities of running programs

All users may raise the niceness (reduce the priority) of their own jobs using the renice command:

```
[student@stationX ~]$ renice 15 -p PID
```

Interactive Process Management Tools

top command

list of the processes running on your system, those processes.

4.18 Network Clients

4.18.1 Web Clients

Firefox

Firefox is a free, open-source web browser for Windows, Linux and Mac OS X (and many other platforms) based on the Mozilla code base it is small, fast and easy to use. Firefox offers many advantages over Internet Explorer, such as tabbed windows, quick links, security and the ability to block ads.

Firefox supports one of the newer things in web browsing: Tabbed browsing. Tabbed browsing allows a user to quickly navigate and access multiple web pages in a single browser window. When you create a bookmark by selecting Bookmarks->Bookmark This Page... you will notice a checkbox called Bookmark All Tabs in a Folder. If this is selected then all open pages will be bookmarked as a group. Selecting the group from the Bookmarks menu will allow you to re-open the tabs at a later date.

Browsing the web without a GUI: links

A number of graphical web clients are provided, discussed in earlier slides, but Red Hat also provides a text-based web client called **links**. This tool can be used on the command line to test network availability as well as provide a fast and stable utility for browsing web pages without graphics. It is capable of retrieving files via HTTP and FTP sites, and can be used to download a web page non-interactively. It is frames-capable and has built in support for SSL.

Useful Arguments to links

-dump	argument causes links to print the text of a page to the standard output and then exit. This provides a fast and convenient way to test web connectivity or retrieve text-based files from websites.
-source	which does the same thing as -dump, but with the web page's html source rather than the rendered content.

Automated http and ftp retrieval: wget

wget can be used to retrieve a single file via HTTP or FTP. wget a command-line utility which downloads files over a network.

```
[student@stationX ~]$ wget
http://www.redhat.com/training/index.html
--17:17:59-- http://www.redhat.com/training/index.html
=> `index.html'
Resolving www.redhat.com... done.
Connecting to www.redhat.com[66.187.232.56]:80... connected.
HTTP request sent, awaiting response... 200 OK
...output truncated...
17:18:00 (295.44 KB/s) - `index.html' saved [28438]
```

Email Protocols

Mail clients require mail servers. Typically different server types will be used to send mail from those used to deliver mail. Mail pickup is typically done through a member of either the IMAP, POP, or Microsoft Exchange family of protocols. The most popular variants of IMAP/POP are IMAPS and POP3S, forms of the protocol that encrypt data over the wire. Mail delivery is most typically done through one of the SMTP family of protocols, either SMTP or ESMTP. Microsoft Exchange also supports its own variant of these. Email clients must be configured to use the proper protocols for pickup and delivery.

Security and Anti-Spam Features

Evolution also supports encrypted email using the Gnu Privacy Guard (GnuPG) and has powerful filters for managing spam. Bayesian spam filters help cut down on junk mail by allowing the user to make unwanted email as junk. This allows the filters to become increasingly personalized and accurate over time.

Non-GUI Mail Clients

Mutt

Mutt is a command line based Email client. It's a very useful and powerful tool to send and read mails from command line in linux based systems. Mutt also supports POP and IMAP protocols for receiving mails. It opens with a colored interface to send Email which makes it user friendly to send emails from command line.

Mailboxes in mutt

You read one "mailbox" (a local mail spool, imap account or pop account) at a time when using **mutt**. You can specify the mailbox you wish to start in by running mutt with the **-f** argument.

For example:

```
[student@stationX ~]$ mutt -f imaps://user@server
```

If no mailbox is specified then your local mail spool will be viewed While in mutt you can switch mailboxes by pressing the **c** key and typing the url of the mailbox you would like to read. You can change which mailbox mutt views by default by altering its configuration file ~/.muttrc.

Mutt Features

Some other important features of **Mutt** is as follows:

- Its very Easy to install and configure.
- Allows us to send emails with **attachments** from the comman line.
- It also has the features to add **BCC (Blind carbon** copy and **CC (Carbon copy)** while sending mails.
- It allows message **threading**.
- It provides us the facility of mailing **lists**.
- It also support so many mailbox form like **maildir, mbox, MH** and **MMDF**.
- Supports at least **20** languages.
- It also support **DSN (Delivery Status Notification)**

Instant messaging and more: Gaim

```
jmh@hork's password:
Filesystem Size Used Avail Use% Mounted on
/dev/hda9 252M 137M 102M 57% /
/dev/hda1 19M 5.5M 12M 31% /boot
/dev/hda6 508M 29M 453M 6% /tmp
/dev/hda5 2.0G 1.7G 221M 88% /usr
/dev/hda7 252M 119M 120M 50% /var
/dev/hdc1 2.3G 1.9G 298M 87% /home
```

Secure File Transfer: scp

scp works like **cp**, except that it copies from one host to another in a secure encrypted channel.

```
student@stationX ~]$ scp hork:proclog .
jmh@hork's password:
proclog 100% |*********************************|
28117 00:00 ETA
```

Also available is **sftp**, an interactive file-transfer program similar to a simple ftp client. The remote host's sshd needs to have support for **sftp** in order for the this to work.

order for the

scp requires that the destination be a directory if the source is a directory or consists of more than one file.

Efficient Network Copies: rsync

rsync is a program that works in much the same way that the older rcp does, but has many more options and uses the **rsync** remote-update protocol to greatly increase the speed of file transfers when the destination file already exists.

The **rsync** remote-update protocol allows **rsync** to transfer just the differences between two sets of files using an efficient checksum-search algorithm. This utility is useful for tasks like updating web content, because it will only transfer the changed files.

Useful options to **rsync**

-e command	specifies an external, rsh-compatible program to connect with (usually **ssh**)

--partial	continues partially downloaded files
--progress	prints a progress bar while transferring
-P	is the same as **--partial –progress**

FTP Clients

Command-line ftp: lftp

Several FTP clients are available, including the traditional, somewhat fundamental, **ftp**. More featureful clients include **lftp**, the no interactive **lftpget**, and the graphical **gftp**.

The **lftp** client includes such useful features as bookmarks and t completion. Below is an example of using lftpget non-interactively:

```
[student@stationX ~]$ lftpget ftp://ftp.example.com/pub/file.txt
```

Graphical FTP: gFTP

gFTP is a graphical FTP client that can be used to upload a download files from remote servers. It can handle authentication us anonymous or real user access. If the SSH2 protocol is chosen, authentication process as well as all transmitted data is encrypted. If have public key authentication configured, gFTP can use it rather th password when authenticating.

authentication process as well as all transmitted data is encrypted. I
have public key authentication configured, gFTP can use it rather
password when authenticating.

smbclient

7 **smbclient** is a command-line tool that provides access to SMB
(most commonly implemented as Microsoft Windows Ne
Neighborhood) shares.

Useful options include:

-W	workgroup or domain
-U	username
-N	suppress password prompt (otherwise you will be asked for password)

Once connected, **smbclient** behaves very much like an ftp clie
You can navigate using **cd** and **ls**, upload and download using put a
get, etc.

Basic Network Diagnostic Tools

A number of network diagnostic tools are available:

Ping	detects if it is possible to communicate with another system. Many systems no longer respond to pings.
Traceroute	displays the computers through which a packet must pass to reach another system. The mtr command is a repetitive version of traceroute, giving continually updated connection time statistics.
Host	performs hostname to IP address translations, as well as the reverse.
Dig	performs a service similar to host in greater detail.
Netstat	provides a number of network statistics.
gnome-nettool	a graphical frontend to the tools listed above (as well as some others) in a single, simple interface. gnome-nettool can be run from the command line or by selecting its icon from the Internet section of the Applications Menu. Note that this tool may not be installed by default.

4.19 Installing Applications

The RPM Package Manager

All software on a Red Hat Enterprise Linux system is divided into RPM packages which can be installed, upgraded, or removed using the RPM package manager.

Package installation is never interactive. In contrast to package management on some other platforms, RPM's design does not provide interactive configuration of software as part of the package load process. RPM can perform configuration actions as part of the installation, but these are scripted not interactive. It is common for packages to install with reasonable default configurations applying. On the other hand some software installs in an un-configured state.

Applies to all software. On some other common platforms, the package management system applies only to part of the installed software. The scope of RPM include core operating systems as well as services and applications. It is common and desirable to run Red Hat Enterprise Linux systems such that all software installed falls under the management of RPM. This is a great aid for management and configuration control, since one framework applies for the management of every installed file.

No such thing as a patch. It is common on other platforms to have operating system updates released as software objects (eg. "service packs") which represent incremental changes to a large number of installed component packages. RPM never does this. If part of any given software package is changed as part of an errata or bug fix, then that entire package will be re-released in its entirety at a new version. The implications are that the installed state of an RPM-managed system can be described as the version number of all the installed components.

Software to be installed using **rpm** is distributed through rpm package files, which are essentially compressed archives of files and associated dependency information. Package files are named using the following format:

name-version-release.architecture.rpm

The version refers to the open source version of the project, while the release refers to Red Hat internal patches to the open source code.

The Yum Package Management Tool

The command-line utility **yum** gives you an easy way to manage the packages on your system:

```
[root@stationX ~]# yum install firefox
```

The above command will

on your system:

```
[root@stationX ~]# yum install firefox
```

The above command will search the configured repositories for package named firefox, and if found will install the latest version, pulli in dependencies if needed.

```
[root@stationX ~]# yum remove mypackage
```

The above command will try to remove the package nam mypackage from your system. If any other package depends mypackage **yum** will prompt you about this, giving you the optio remove those packages as well.

```
[root@stationX ~]# yum update [mypackage...]
```

If any packages are specified on the command-line **yum** will sea the configured repositories for updated versions of those packages install them. When no packages are specified **yum** will search updates to all of your currently installed packages.

```
[root@stationX ~]# yum list available
```

The above command will list all packages in the yum reposit available to be installed.

Graphical Package Management

The Graphical Updater pup is a graphical tool based on yum that provides the functionality to check all available (enabled> repositories for updates. Access to pup is obtained through the Applications menu or by typing pup on the command-line which will launch an easy-to-use graphical interface.

pup

- Applications->System Tools->Software Updater
- List and install software updates

The companion tool pirut, provides an equally intuitive graphical interface to view, install, an remove packages. Access to pirut can be obtained through the Applications menu or by typing pirut on the command-line.

pirut

- Applications->Add/Remove Software
- View, install and un-install other packages

4.20 Sample Programs

Exercise

1. **Write a shell program to find the factorial of a given number**

```
#!bin/bash
echo "Enter the number"
read n
i=1
fact=1
while [ $i -le $n ]
do
((fact=$fact*$i))
((i=$i + 1))
done
echo "Factorial of $n is: $fact"
```

Output:

```
[root@localhost root]# bash factor
Enter the number
6
Factorial of 6 is :720
```

8. **Write a shell program for finding the sum and average of 4 integers**

```
#!bin/bash
echo "Enter four integers"
read a
read b
read c
read d
sum=`expr $a + $b + $c + $d`
avg=`expr$sum / 4`
dec=`expr \( $dec \* 1000 \) / 4`
echo "Sum=$sum"
echo "Average=$avg.$dec"
```

Output:

```
Enter four integers
10
20
30
40
Sum=100
Average=25
```

9. **Write a shell program to compute simple interest and compound**

Write a shell program to find the sum of odd/even numbers from M to N.

```
#!bin/bash
echo "Enter the value of m"
read m
echo "Enter the value of n"
read n
k=0
p=0
while [ $m -le $n ]
do
((t=m%2))
if [ $t -eq 0 ]
then ((p=p+m))
else
((k=k+m))
fi
((m=m+1))
done
echo "The sum of odd numbers are $k"
echo "The sum of even numbers are $p"
```

Output:

```
[root@localhost root]# bash pg2
Enter the value of m
1
Enter the value of n
10
The sums of odd numbers are 25
The sums of even numbers are 30
```

Write a program to find greatest of given three numbers.

Stewaugh
8

7. **Write a shell program to develop a calculator application.**

```
#!bin/bash
j=1
while [ $j -eq 1 ]
do
echo "Enter the First Operand;"
read f1
echo "Enter the second operand:"
read f2
echo "1-> Addition"
echo "2-> Subtraction"
echo "3-> Multiplication"
echo "4-> Division"
echo "Enter your choice"
read n
case "$n" in
1)
echo "Addition"
f3=$((f1+f2))
echo "The result is:$f3"
;;
2)
echo "Subtraction"
let "f4=$f1 - $f2"
echo "The result is:$f4"
;;3)
```

```
echo "Multiplication"
let "f5=$f1 * $f2"
echo "The result is:$f5"
;;
4)
echo "Division"
let "f6=$f1 / $f2"
echo "The result is:$f6"
;;
esac
echo "Do you want to continue(press:1 otherwise press any key
to
quit)"
read j
done
```

Output:

```
[su@localhost su]$ bash u
Enter the First Operand;
23
Enter the second operand:
23
1-> Addition
2-> Subtraction
3-> Multiplication
4-> Division
Enter your choice
1
Addition
The result is:46
Do you want to continue(press:1 otherwise press any key to quit)
1
Enter the First Operand;
20
```

5. **Write a shell program to check the given number is prime or not.**

```
#!bin/bash
echo "enter a number"
read n
i=2
while [ $i -lt $n ]
do
a=`expr $n % 0`
if [ $a -eq 0 ]
then
echo "$n is not a prime number "
echo "since it is divisible by $i"
exit
fi
i=`expr $i + 1`
done
echo "$n is a prime number"
```

Output:

```
Enter a number:
33
33 is not a prime number
Since it is divisible by 3
```

5. **Write a shell program to check whether the given string is palindrome or not.**

```
#!bin/bash
echo "Enter the string:"
read str
len=${#str}
k=$len-1
i=0
flag=1
while [ $i -le $((len / 2))
do
s1=${str:$i:1}
s2=${str:$k-i:1}
if [ $s1 != $s2 ]
then
flag=0
fi
let i++
done
if [ $flag -eq 1 ]
then
echo "Palindrome"
else
echo "Not Palindrome"
fi
```

Output:

```
Enter the string:
LIRIL
Palindrome
Enter the string:
Lecturer
Not Palindrome
```

UNIT- II

Introduction to MY SQL – The show Databases and Table – The USE command – Create Database and Tables – Describe Table – Select, Insert, Update, and Delete statement – Some Administrative detail – Table Joins – Loading and Dumping a Database.

Introduction to MY SQL

What is MySQL?

- MySQL is a database system used for developing web-based software applications.
- MySQL used for both small and large applications.
- MySQL is a relational database management system *(RDBMS).*
- MySQL is fast, reliable, and flexible and easy to use.
- MySQL supports standard SQL *(Structured Query Language).*
- MySQL is free to download and use.
- MySQL was developed by Michael Widenius and David Axmark in 1994.
- MySQL is presently developed, distributed, and supported by Oracle Corporation.
- MySQL Written in C, C++.

Main Features of MySQL

- MySQL server design is multi-layered with independent modules.
- MySQL is fully multithreaded by using kernel threads. It can handle multiple CPUs if they are available.
- MySQL provides transactional and non-transactional storage engines.
- MySQL has a high-speed thread-based memory allocation system.
- MySQL supports in-memory heap table.
- MySQL Handles large databases.
- MySQL Server works in client/server or embedded systems.
- MySQL Works on many different platforms.

Who Uses MySQL

- Some of the most famous websites like Facebook, Wikipedia, Google (not for search), YouTube, Flickr.

- Content Management Systems (CMS) like WordPress, Drupal, Joomla, phpBB etc.
- A large number of web developers worldwide are using MySQL to develop web applications.

The show Databases and Table

SHOW DATABASES lists the databases on the MySQL server host.
SHOW SCHEMAS is a synonym for SHOW DATABASES.

To show the tables in a MySQL database:

1. Log into your database using the mysql command line client
2. Issue the use command to connect to your desired database (such as, use mydatabase)
3. Use the MySQL *show tables* command, like this:

```
show tables;
```

A complete explanation follows.

MySQL 'show tables': A complete example

First, connect to your MySQL database using your MySQL client from your operating system command line:

```
$ mysql -u root -p
```

Next, after you're logged into your MySQL database, tell MySQL which database you want to use:

```
mysql> use pizza_store;
```

Now issue the MySQL show tables command to list the tables in the current database:

```
mysql> show tables;
```

For instance, if I issue this MySQL show tables command in one of my example MySQL databases, I'll see this output:

```
mysql> show tables;

+-----------------------+

| Tables_in_pizza_store |

+-----------------------+

| crust_sizes           |

| crust_types           |

| customers             |

| orders                |

| pizza_toppings        |

| pizzas                |

| toppings              |

+-----------------------+

7 rows in set (0.00 sec)
```

USE Statement

```
USE db_name
```

The USE statement tells MySQL to use the named database as the default (current) database for subsequent statements. This statement requires some privilege for the database or some object within it.

The named database remains the default until the end of the session or another USE statement is issued:

```
USE db1;

SELECT COUNT(*) FROM mytable;   # selects from db1.mytable

USE db2;

SELECT COUNT(*) FROM mytable;   # selects from db2.mytable
```

The database name must be specified on a single line. Newlines in database names are not supported.

Making a particular database the default by means of the USE statement does not preclude accessing tables in other databases. The following example accesses the author table from the db1 database and the editor table from the db2 database:

```
USE db1;

SELECT author_name,editor_name FROM author,db2.editor

  WHERE author.editor_id = db2.editor.editor_id;
```

Create MySQL Tables

To begin with, the table creation command requires the following details −

- Name of the table
- Name of the fields
- Definitions for each field

Syntax

Here is a generic SQL syntax to create a MySQL table −

CREATE TABLE table_name (column_name column_type);

Now, we will create the following table in the **TUTORIALS** database.

```
create table tutorials_tbl(
  tutorial_id INT NOT NULL AUTO_INCREMENT,
  tutorial_title VARCHAR(100) NOT NULL,
  tutorial_author VARCHAR(40) NOT NULL,
  submission_date DATE,
  PRIMARY KEY ( tutorial_id )
);
```

Here, a few items need explanation −

- Field Attribute **NOT NULL** is being used because we do not want this field to be NULL. So, if a user will try to create a record with a NULL value, then MySQL will raise an error.
- Field Attribute **AUTO_INCREMENT** tells MySQL to go ahead and add the next available number to the id field.
- Keyword **PRIMARY KEY** is used to define a column as a primary key. You can use multiple columns separated by a comma to define a primary key.

Creating Tables from Command Prompt

It is easy to create a MySQL table from the mysql> prompt. You will use the SQL command **CREATE TABLE** to create a table.

Example

Here is an example, which will create **tutorials_tbl** −

```
root@host# mysql -u root -p
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> CREATE TABLE tutorials_tbl(
  -> tutorial_id INT NOT NULL AUTO_INCREMENT,
  -> tutorial_title VARCHAR(100) NOT NULL,
  -> tutorial_author VARCHAR(40) NOT NULL,
  -> submission_date DATE,
  -> PRIMARY KEY ( tutorial_id )
  -> );
Query OK, 0 rows affected (0.16 sec)
mysql>
```

NOTE − MySQL does not terminate a command until you give a semicolon (;) at the end of SQL command.

DESCRIBE and EXPLAIN in MySQL?

November 19, 2018

In MySQL, the `DESCRIBE` and `EXPLAIN` statements are synonyms, used either to obtain information about table structure or query execution plans.

1. To describe a table:

Even though `DESCRIBE` and `EXPLAIN` statements are synonyms, the `DESCRIBE` statement is used more to obtain information about a table structure while `EXPLAIN` statement is used to obtain a query execution plan.

The `DESCRIBE` statement is a shortcut for `SHOW COLUMN` statement:

```
DESCRIBE table_name;
```

is equivalent to this `SHOW COLUMN` statement:

```
SHOW COLUMNS FROM table_name;
```

Or you can also use the short form of describe:

```
DESC table_name;
```

Those describe statements above show the columns in the table and all their attributes such as name, data type, collation, Nullability, Primary key, default, comment, etc.

Instead of DESCRIBE or DESC, you can use EXPLAIN statement which works the same:

```
EXPLAIN table_name;
```

2. To describe a query execution plan

We often use `EXPLAIN`. It provides information about how your SQL database executes a query.

`EXPLAIN` works with `SELECT`, `DELETE`, `INSERT`, `REPLACE`,
and `UPDATE` statements. It also requires the SELECT privilege for any tables or views accessed, including any underlying tables of views. For views, EXPLAIN also requires the SHOW VIEW privilege.

```
EXPLAIN SELECT * FROM table_name;
```

Inserting data in Table

The INSERT statement is used to insert data into tables.

We will create a new table, where we will do our examples.

```
mysql> CREATE TABLE Books(Id INTEGER PRIMARY KEY, Title VARCHAR(100),
    -> Author VARCHAR(60));
```

We create a new table Books, with Id, Title and Author columns.

```
mysql> INSERT INTO Books(Id, Title, Author) VALUES(1, 'War and Peace',
    -> 'Leo Tolstoy');
```

This is the classic INSERT SQL statement. We have specified all column names after the table name and all values after the VALUES keyword. We add our first row into the table.

```
mysql> SELECT * FROM Books;
+----+---------------+-------------+
| Id | Title         | Author      |
+----+---------------+-------------+
|  1 | War and Peace | Leo Tolstoy |
+----+---------------+-------------+
```

We have inserted our first row into the Books table.

```
mysql> INSERT INTO Books(Title, Author) VALUES ('The Brothers Karamazov',
    -> 'Fyodor Dostoyevsky');
```

We add a new title into the Books table. We have omitted the Id column. The Id column has AUTO_INCREMENT attribute. This means that MySQL will increase the Id column automatically. The value by which the AUTO_INCREMENT column is increased is controlled by auto_increment_increment system variable. By default it is 1.

```
mysql> SELECT * FROM Books;
+----+------------------------+--------------------+
| Id | Title                  | Author             |
+----+------------------------+--------------------+
|  1 | War and Peace          | Leo Tolstoy        |
|  2 | The Brothers Karamazov | Fyodor Dostoyevsky |
+----+------------------------+--------------------+
```

Here is what we have in the Books table.

```
mysql> INSERT INTO Books VALUES(3, 'Crime and Punishment',
    -> 'Fyodor Dostoyevsky');
```

In this SQL statement, we did not specify any column names after the table name. In such a case, we have to supply all values.

```
mysql> REPLACE INTO Books VALUES(3, 'Paradise Lost', 'John Milton');
Query OK, 2 rows affected (0.00 sec)
```

The REPLACE statement is a MySQL extension to the SQL standard. It inserts a new row or replaces the old row if it collides with an existing row. In our table, there is a row with Id=3. So our previous statement replaces it with a new row. There is a message that two rows were affected. One row was deleted and one was inserted.

```
mysql> SELECT * FROM Books WHERE Id=3;
+----+---------------+-------------+
| Id | Title         | Author      |
+----+---------------+-------------+
|  3 | Paradise Lost | John Milton |
+----+---------------+-------------+
```

This is what we have now in the third column.

We can use the INSERT and SELECT statements together in one statement.

```
mysql> CREATE TABLE Books2(Id INTEGER PRIMARY KEY AUTO_INCREMENT,
    -> Title VARCHAR(100), Author VARCHAR(60)) type=MEMORY;
```

First, we create a temporary table called Books2 in memory.

```
mysql> INSERT INTO Books2 SELECT * FROM Books;
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

Here we insert all data into the Books2 that we select from the Books table.

```
mysql> SELECT * FROM Books2;
+----+-----------------------+--------------------+
| Id | Title                 | Author             |
+----+-----------------------+--------------------+
|  1 | War and Peace         | Leo Tolstoy        |
|  2 | The Brothers Karamazov | Fyodor Dostoyevsky |
|  3 | Paradise Lost         | John Milton        |
+----+-----------------------+--------------------+
```

We verify it. All OK.

```
mysql> INSERT INTO Books(Title, Author) VALUES ('The Insulted and Humiliated',
    -> 'Fyodor Dostoyevsky'), ('Cousin Bette', 'Honore de Balzac');
```

```
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

We can insert more than one row into the table with the INSERT statement. Here we show how.

We can insert data from a file on the filesystem. First, we dump data from the Books table in a books.csv file.

```
mysql> SELECT * INTO OUTFILE '/tmp/books.csv'
    -> FIELDS TERMINATED BY ','
    -> LINES TERMINATED BY '\n'
    -> FROM Books;
mysql> TRUNCATE Books;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM Books;
Empty set (0.00 sec)
```

We delete all data from the table.

Deleting data in Table

In MySQL, we can delete data using the DELETE and TRUNCATE statements.
The TRUNCATE statement is a MySQL extension to the SQL specification. First, we are going to delete one row from a table. We will use the Books2 table that we have created previously.

```
mysql> DELETE FROM Books2 WHERE Id=1;
```

We delete a row with Id=1.

```
mysql> SELECT * FROM Books2;
+----+------------------------+--------------------+
| Id | Title                  | Author             |
+----+------------------------+--------------------+
|  2 | The Brothers Karamazov | Fyodor Dostoyevsky |
|  3 | Paradise Lost          | John Milton        |
+----+------------------------+--------------------+
```

We verify the data.

```
mysql> DELETE FROM Books2;
mysql> TRUNCATE Books2;
```

These two SQL statements delete all data in the table.

Updating data

The UPDATE statement is used to change the value of columns in selected rows of a table.

```
mysql> SELECT * FROM Books;
+----+-----------------------------+---------------------+
| Id | Title                       | Author              |
+----+-----------------------------+---------------------+
|  1 | War and Peace               | Leo Tolstoy         |
|  2 | The Brothers Karamazov      | Fyodor Dostoyevsky  |
|  3 | Paradise Lost               | John Milton         |
|  4 | The Insulted and Humiliated | Fyodor Dostoyevsky  |
|  5 | Cousin Bette                | Honore de Balzac    |
+----+-----------------------------+---------------------+
```

We recreate the table Books. These are the rows.

Say we wanted to change 'Leo Tolstoy' to 'Lev Nikolayevich Tolstoy' table. The following statement shows, how to accomplish this.

```
mysql> UPDATE Books SET Author='Lev Nikolayevich Tolstoy'
    -> WHERE Id=1;
```

The SQL statement sets the author column to 'Lev Nikolayevich Tolstoy' for the column with Id=1.

```
mysql> SELECT * FROM Books WHERE Id=1;
+----+---------------+--------------------------+
| Id | Title         | Author                   |
+----+---------------+--------------------------+
|  1 | War and Peace | Lev Nikolayevich Tolstoy |
+----+---------------+--------------------------+
```

The row is correctly updated.

Some administrative commands

•

MySQL provides various administration features/tools to configure the MySQL server. These includes server maintenance, database backup, MySQL security, user management, start up/shutdown, replication management, Configuration of parameters (resource management), check performance/ status etc.

Creating and grant permissions to a User

Users information of MySQL exists in a database named mysql and the tables named user.Privileges of user exists in different tables according to privilege level on different database although the main privileges are in user table itself.

There are two ways for creating a basic user for accessing mysql server:

1. **Create user command:**

To create a new userthat connects from with the password user_password , you can use the statement as follows:

```
CREATE USER username@localhostIDENTIFIED BY ‘user_password’ ;
```

To allow user to connect from any host you use the% wildcard, which means any host.

```
CREATE USER username@%IDENTIFIED BY ‘user_password’ ;
```

2. **Insert command:**

To create a new userthat connects from with the password user_password , you can use the statement as follows:

```
INSERT INTO user
(host,user,password)VALUES('localhost',’username’,PASSWORD(‘user_password'));
```

To allow user to connect from every host you can use the % wildcard, which means all hosts.

```
INSERT INTO user
(host,user,password)VALUES('localhost',’username’,PASSWORD(‘user_password'));
```

The above two methods create the basic user for connecting to MySQL server. Although MySQL provides the user management according to the privileges given to the specified user for the security measures.

For example: If your sales teamneeds to analyze the data then you should create a read user for that team and not giving all other privileges to them. For read only you have to give only select permissions to them with the following command:

?

```
CREATE USER user_sales@%IDENTIFIED BY ‘sales password’ ;
Grant select on *.* to user_sales@% ;
Flush privileges;
```

This grant statement gives only select permission on all the database for analyzing purpose to user_sales user.

MySQL Administration provides various kinds of privileges to separate out the user and hence improve the security through proper user management. Some important privileges are as follows:

Select_priv: For only reading/selecting the data.

Insert_priv: For only insertion of the data on the databases.

Delete_priv: Allow only deleting the data.

Create_priv: For creating a database or a table on MySQL server.

Drop_priv: For dropping any specified table/database.

Grant_priv: Allow to give permission to any other user or able to create a new user.

File_priv: Allow to create a file from the output command and write the data out from the MySQL sever.

Alter_priv: Allow to change the schema of the existing tables.

Process_priv: To check the running process on the server, basic command is show processlist.

Shutdown_priv: Allow user to stop/start the MySQL Process.

Index_priv: Allow user to create indexes on the tables.

Reload_priv: To apply the changes in user management at that time.

All the above privileges can have a value of yes or no by the flags= 'Y' or 'N'. You can give the privileges using the grant statement.

For Example if we need to create a user having select, insert, update privileges on the test database only for localhost.

```
Grant select, insert, update on test.* to user123@% ;
```

If a user with name user123 is not exists in the database then a new user with blank password is created automatically. So the grant command also works to directly create the user as:

```
Grant select, insert, update on test.* to user123@% identified by password('password123');
```

Handling Server Status

MySQL Administration allows to start, stop, restart and checkthe status of the MySQL server. It provides a tool called mysqladmin to do so with the following commands:

?

```
To shut down the running MySQL server:
root@localhost#./mysqladmin -u root -p shutdown
Enter password:********

To start the MySQL server:
root@localhost#./safe_mysqld&
```

And if you already add the mysql.server(mysqld daemon) file to default location /etc/init.d as specified in installation steps then you can start, stop, restart and check the status as:

For start

```
root@localhost# /etc/init.d/mysqld  start
```

For stop

```
root@localhost# /etc/init.d/mysqld  stop
```

For restart

```
root@localhost# /etc/init.d/mysqld  restart
```

For checking the status

```
root@localhost# /etc/init.d/mysqld  status
```

NOTE: Use the above methods to stop the MySQL server and not kill the process ID directly as it causes the MySQL tables to crash which is very difficult to repair.

Administrative Commands

There are certain administrative commands run by database administrator on routine basis or on requirement basis of the business or application. For example some business/Application requires there tables to be checked at every night for data inconsistency, so we need to run check table and optimize them for eliminating any holes at the physical level for best performance. But we can't run all commands in every scenario because in some cases the database is of 24*7 hour nature and there is no downtime so we can't run optimize and check at the runtime to avoid transaction locking.

There are some commands related to security like updating passwords, updating bind-address, changing permission/grants and there are some commands related to monitoring the database system performance like show variables, show processlist, show full processlist, bechmarking etc.

Update user password: You can use the update command to change the password of the user in mysql as:

```
Update mysql.user set password = 'NEW_PASSWORD' where user = 'username' and
host='hostname'
```

NOTE: Always give username and hostname both to avoid updation of more than one row.Because the username and hostname define uniqueness on combination.

Set variables: SET is a admin command to set variables in the MySQL server. There are two types of variables session and global variables that can be set with this simple set command. For example you can skip slave error in mysql as:

```
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = 1 ;
```

Analyze table: This command is used for statistics of the table. It checks for the key distribution of the table and then stores the relevant values into the information_schema (contains meta information of the tables). This command is generally used after alter queries to change engine or some other column types. Command:

```
Analyze table table_name;
```

Repair table: This command is use to repair the table as the name indicates. Repair in MySQL means to correct any faulty key/index in the table or any row corruption due to which whole table act abnormally. NOTE: Repair table may changes your row count and deleted some corrupted data. The repair command:

    Repair table table_name;

Optimize table: This command is used to optimize table for better performance, it eliminates any present holes in the data or index so that provide fast access to the data, rebuilt indexes in the database. Optimize can be run as:

    Optimize table table_name;

Kill query: You can kill a query during its execution. For this you can pass the kill query with query id as the parameter in the command. The query id is shown in the processlist output. For example:

?

```
mysql> show processlist;
+----+------+-----------------+-----------+---------+------+-------------+----------------------------------+
| Id | User | Host            | db        | Command | Time | State       | Info                             |
+----+------+-----------------+-----------+---------+------+-------------+----------------------------------+
| 19 | root | localhost:55107 | town_list | Query   |    0 | System lock | alter table toen engine='innodb' |
+----+------+-----------------+-----------+---------+------+-------------+----------------------------------+
2 rows in set (0.00 sec)
Here the query id is 19 for the command " alter table toen engine='innodb' ".
So you can kill the query in middle as:
mysql> kill 19 ;
```

Show grants: Grants means the privileges to a mysql user, you can see the grant of any user by the help of show grants command, for this you need to pass the user and its host as it uniquely defines the user. The show grants command can be used as:

?

```
mysql> show grants for greg@xxx.xxx.xxx.xxx ;
GRANT SELECT, INSERT, UPDATE, DELETE, DROP on `test`.* TO ‘greg'@'xxx.xxx.xxx.xxx'
```

```
Here greg is the username and xxx.xxx.xxx.xxx is the host name for greg.
```

Show variables: This command is used to check the system variables at the current time. These are default variables or we can set them in the configuration file of MySQL server. The command to show the session or global variables are:

?

```
<span style="font-size: 13px; line-height: 20.0063037872314px;">For global:
</span>mysql> show global variables like 'join_buffer_size' ;

Variable_name     Value
join_buffer_size   536870912
1 row in set (0.00 sec)

For session:
mysql> show session variables like 'join_buffer_size' ;
Variable_name     Value
join_buffer_size   12480970
1 row in set (0.00 sec)
```

Table Joins

SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Let's look at a selection from the "Orders" table:

OrderID	CustomerID	OrderDate

10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Then, look at a selection from the "Customers" table:

CustomerID	**CustomerName**	**ContactName**	**Country**
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

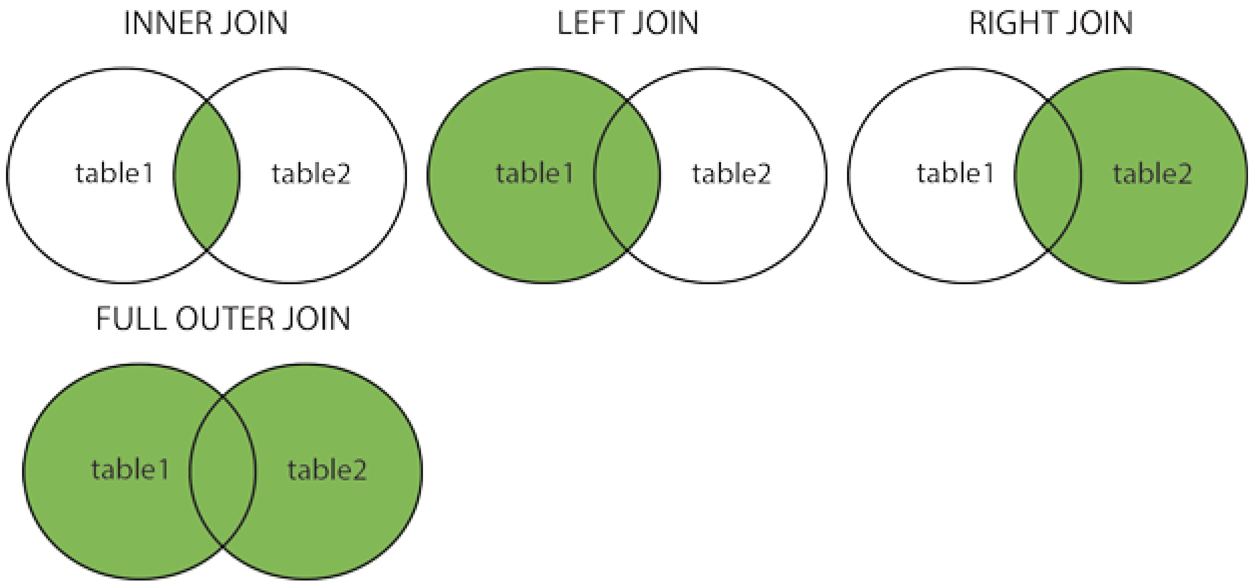
and it will produce something like this:

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table



Exercise:

Insert the missing parts in the JOIN clause to join the two tables Orders and Customers, using the CustomerID field in both tables as the relationship between the two tables.

```
SELECT *
FROM Orders
LEFT JOIN Customers
[        ] = [        ];
```

Loading and Dumping a Database.

We can load a database or otherwise execute SQL commands from a file. We simply put the commands or database into a file—let's call it mystuff.sql—and load it in with this command:

```
$ mysql people < mystuff.sql
```

We can also dump out a database into a file with this command:

```
$ mysqldump people > entiredb.sql
```

For fun, try the mysqldump command with the people database (a gentle reminder: the password is LampIsCool):

```
$ mysqldump -uapache -p people

Enter password:
```

Notice that this outputs all the SQL needed to create the table and insert all the current records. For more information, see man mysqldump.

< Back **Page 5** of 7 Next >

Introduction to mysqldump tool

The mysqldump tool allows you to make a backup of one or more databases by generating a text file that contains SQL statements which can re-create the databases from scratch.
The mysqldump tool is located in the root/bin directory of the MySQL installation directory.
To access the mysqldump tool, you navigate to the root/bin folder and use the mysqldump command with the following options.
Here are the common mysqldump options:

add-drop-table
Includes a DROP TABLE statement for each table in the database.

add-locks
Includes LOCK TABLES and UNLOCK TABLES statements before and after each INSERT statement. It improves the data restoration speed from the dump file.

all-databases
Creates a dump of all databases on the MySQL server.

create-options
Includes ENGINE and CHARSET options in the CREATE TABLE statement for each table.

databases
Creates a dump of one or more databases.

UNIT-III

PHPIntroduction–GeneralSyntacticCharacteristics–PHPScripting–Commentingyour code – Primitives,OperationsandExpressions–PHPVariables–Operationsand Expressions-Controlstatement–Array–Functions-BasicFormProcessing–Fileand FolderAccess – Cookies– Sessions– DatabaseAccess with PHP(MY SQL).

PHPIntroduction

PHP is one of the most widely used server side scripting language for web development. Popular websites like Facebook, Yahoo, Wikipedia etc, are developed using PHP.

PHP is so popular because it's very simple to learn, code and deploy on server, hence it has been the first choice for beginners since decades.

What is PHP?

PHP stands for **Hypertext Pre-Processor**. PHP is a scripting language used to develop static and dynamic webpages and web applications.

Here are a few important things you must know about PHP:

1. PHP is an Interpreted language, hence it doesn't need a compiler.
2. To run and execute PHP code, we need a Web server on which PHP must be installed.
3. PHP is a server side scripting language, which means that PHP is executed on the server and the result is sent to the browser in plain HTML.
4. PHP is open source and free.

Is PHP the right language to learn?

If you are still confused about whether you should learn PHP or is PHP the right language for your web project, then here we have listed down some of the features and usecases of PHP language, which will help you understand how simple yet powerful PHP scripting language is and why you should learn it.

1. PHP is **open source** and **free**, hence you can freely download, install and start developing using it.
2. PHP has a very **simple and easy to understand syntax**, hence the learning curve is smaller as compared to other scripting languages like JSP, ASP etc.
3. PHP is **cross platform**, hence you can easily develop and move/deploy your PHP code/project to almost all the major operating systems like Windows, Linux, Mac OSX etc.
4. All the popular **web hosting services support PHP**. Also the web hosting plans for PHP are generally the amongst the cheapest plans becasue of its popularity.
5. With PHP, you can create static and dynamic webpages, perform file handling operations, send emails, access and modify browser cookies, and almost everything else that you might want to implement in your web project.
6. PHP is **fast** as compared to other scripting languages like JSP and ASP.

7. PHP has in-built support for **MySQL**, which is one of the most widely used Database management system.

These are some of the main features of PHP, while as you will learn the language you will realise that apart from these features,

Uses of PHP

To further fortify your trust in PHP, here are a few applications of this amazing scripting language:

1. It can be used to **create Web applications** like Social Networks(Facebook, Digg), Blogs(Wordpress, Joomla), eCommerce websites(OpenCart, Magento etc.) etc.
2. **Comman Line Scripting**. You can write PHP scripts to perform different operations on any machine, all you need is a PHP parser for this.
3. **Create Facebook applications** and easily integrate Facebook plugins in your

website, using Facebook's PHP SDK. Check this link for more information.

4. **Sending Emails** or building email applications because PHP provides with a robust email sending function.

Characteristics of PHP

Five important characteristics make PHP's practical nature possible −

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity
- **Simple, Familiar and ease of use:** Its popularly known for its simplicity, familiarity and easy to learn the language as the syntax is similar to that of ‘C’ or Pascal language.
  So the language is very logical and well

organizedgeneral-purpose programming language.

- Even people with a normal programming background can easily understand and capture the use of language.
- PHP is very advantageous for new users as its a very reliable, fluent, organized, clean, demandable and efficient.

The main strength of PHP is the availability of rich pre-defined functions. The core distribution helps the developers implement dynamic websites very easily with secured data. PHP applications are very easy to optimize.

- **Loosely typed language:** PHP encourages the use of variables without declaring its data types. So this is taken care at the execution time depending on the value assigned to the variable. Even the variable name can be changed dynamically.
- **Flexibility:** PHP is known for its flexibility and embedded nature as it can be well integrated with **HTML**, **XML**, **Javascript** and

many more. PHP can run on multiple operatingsystems like **Windows**, **Unix**, **Mac OS**, **Linux**, **etc**. The PHP scripts can easily run on any device like laptops, mobiles, tablets, and computer. It is very comfortably integrated with various Databases. Desktop applications are created using advanced PHP features. The executable PHP can also be run on command-line as well as directly on the machine. Heavyweight applications can be created without a server or browser. It also acts as an excellent interface with relational databases.

- **Open Source:** All PHP frameworks are open sources, No payment is required for the users and its completely free. User can just download PHP and start using for their applications or projects. Even in companies, the total cost is reduced for software development providing more reliability and flexibility.

It supports a popular range of databases like MySQL, SQLite, Oracle, Sybase, Informix, andPostgreSQL.

PHP provides libraries to access these databases to interact with web servers.

- **Cross-platform compatibility:** PHP is multi-platform and known for its portability as it can run on any operating System and windows environments. The most common are XAMPP (**Windows**, **Apache Server**, **MySQL**, **Perl**, and **PHP**) and LAMP (**Linux**, **Apache**, **MySQL**, **PHP**).
- As PHP is platform-independent, it’s very easy to integrate with various databases and other technologies without re-implementation. It effectively saves a lot of energy, time and money.
- **Error reporting and exceptions:**

PHP supports much errors reporting constants to generate errors and relevant warnings at run time.

-
- **Fast and efficient performance:** Users generally prefer fast loading websites. For any web development, speed becomes an important aspect which is taken care of by PHP.

PHP scripts are faster than other scripting languages like **ASP.NET**, **PERL**, and **JSP**.

Maintenance: When dealing with big projects, maintenance of code is also an important aspect of the web development process. There are many PHP frameworks for example MVC (Model View Controller) which makes development and maintenance of code easier. Files belonging to the different module are maintained separately.

- **Third-party application support and security:** Many PHP's predefined functions support data encryption options keeping it more secure. Even the users can use third-party applications to secure data.

- **Real time access monitoring:** PHP also provides a summary of user's recent logging accesses.
- **Memory and CPU usage information:** PHP can provide memory usage information from functions like **memory_get_usage()** or **memory_get _peak_usage()**, which can help the developers optimize their code. In the similar way, the CPU power consumed by any script can be retrieved for further optimization.
- **Object oriented features:** PHP supports object-oriented programming features, resulting in increased speed and introducing added features like data encapsulation and inheritance at many levels.
- **Magic Constants:** PHP provides many built-in magic methods starting with __**(double underscore)** which are called during specific events.

PHP Script

PHP was established as the leading website programming language several years ago, even though it is much younger than other languages. The cause of the extensive popularity the PHP distribution enjoys is the easy-to-grasp syntax, allowing even people with no coding experience whatsoever to quickly enter into the PHP realm. And it is exactly the easily created scripts that make PHP so popular among the Internet community.

PHP Scripts
PHP scripts and database interaction
PHP scripts and email

A sample PHP script:

```
<?php
echo '<p>This is a PHP script</p>';
?>
```

PHP scripts can be created using any basic text editor or HTML editing software tool. Each

PHP file must be saved with a .php file extension in order to be recognized as a functioning PHP script.

When the Apache server has the appropriate settings, PHP code can be recognized also in .html files.

PHP in an HTML file

```
<html>
<head></head>
<body class="page_bg">
Hello, today is <?php echo date('l, F jS, Y'); ?>.
</body>
</html>
```

PHP scripts and database interaction
The PHP language has been especially designed to enable the development of dynamic and interactive web pages.
PHP offers excellent connectivity with many databases,among them MySQL, PostgreSQL and Generic ODBC. The common combination between the PHP v.4 or the PHP v.5 script and the MySQL

database is supported on almost every Linux-based web server.

Connecting to a MySQL database with a PHP script:

```
<?php
$dbhandle = mysql_connect('localhost',
'phptest', '3579php');
if ($dbhandle)
{
echo "Connected to MySQL Database<br>";
mysql_close($dbhandle);
} else {
echo "Unable to connect to MySQL
Database<br>";
}
?>
```

PHP scripts and email

As a web-oriented scripting language, PHP contains all the necessary functionalities to perform different Internet operations. It is exceptionally easy indeed to create a PHP script for connecting to remote servers, for checking email via POP3 or IMAP, for URL encoding, setting cookies, redirecting, etc. PHP is extensively used for the creation and operation of online forms and automated email services.

Here you can see the combination of a pure HTML input form and a corresponding PHP script, which will handle the form’s data.

A sample HTML form:

```
<html>
<head><title>Example Form</title></head>
<body>
<form action="mail.php" method="POST">
<b>Email address</b><br>
```

```
<input type="text" name="email" size=40>
<p><b>Subject</b><br>
<input type="text" name="subject" size=40>
<p><b>Message</b><br>
<textarea cols=40 rows=10
name="message"></textarea>
<p><input type="submit" value=" Send your
email ">
</form>
</body>
</html>
```

And here is the PHP code, which will handle and process the information:

How to write comments in PHP

Single Line Comments

```
<?php
echo"Hello World!"; // Output "Hello World!"
?>
```

Multi-line Comments

```
<?php
/* The following line of code
will output the "Hello World!" message */
echo"Hello World!";
?>
```

Primitives,OperationsandExpressions

Variables

Variables in PHP are represented by a dollar sign followed by the name of the variable. The variable name is case-sensitive.

PHP supports eight primitive types:

- Four scalar types:
  - boolean
  - integer
  - float (floating-point number, aka double)
  - string
- Two compound types:
  - array

    - object
- And finally three special types:
    - resource
    - NULL
    - callable

```
// Integers: decimal, octal or hexadecimal
$Var = 123;
// Floating point
$Var = 1.3e4;
// Arrays or vectors
$Var[2] = 123;
// Text Strings
$Var = "A Text String\n";
// Objects
$Var = new oMyClass();
```

Constants

A constant is an identifier (name) for a simple value. That value cannot change during the execution of the script. A constant is case-sensitive by default. By convention, constant identifiers are always uppercase

There are two built-in constants, TRUE and FALSE (case-insensitive), which representthe two possible boolean values.

Expressions

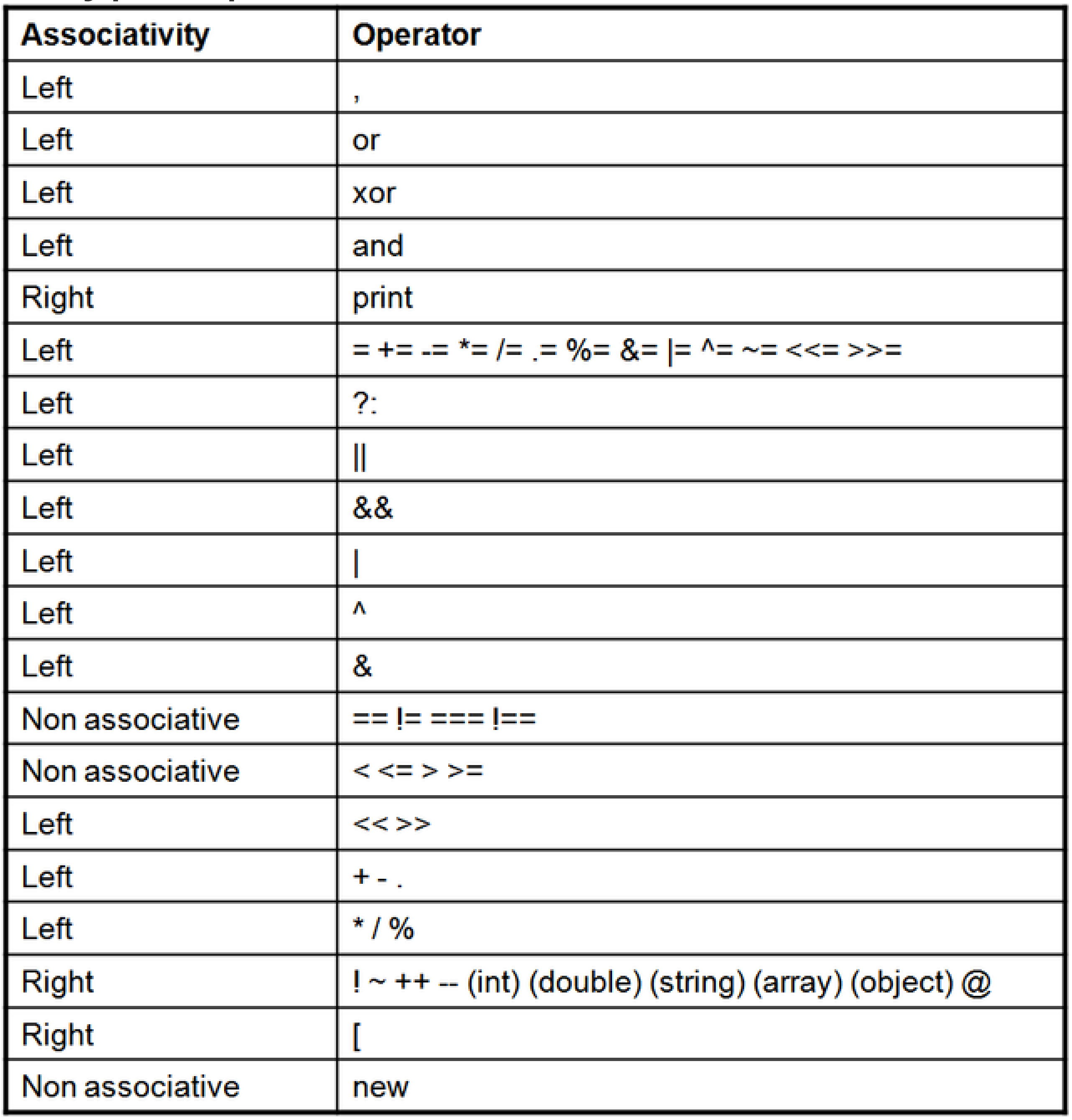
Expressions are the most important building stones of PHP. The most basic forms of expressions are constants and variables.

A very common type of expressions are comparison expressions. These expressions evaluate to either FALSE or TRUE. These expressions are most commonly used inside conditional execution, such as if statements.

Operators

- Operator Precedence
- Arithmetic Operators
- Assignment Operators
- Bitwise Operators
- Comparison Operators
- Error Control Operators
- Execution Operators
- Incrementing/Decrementing Operators
- Logical Operators
- String Operators

- Array Operators
- Type Operators



Associativity	Operator
Left	,
Left	or
Left	xor
Left	and
Right	print
Left	= += -= *= /= .= %= &= |= ^= ~= <<= >>=
Left	?:
Left	||
Left	&&
Left	|
Left	^
Left	&
Non associative	== != === !==
Non associative	< <= > >=
Left	<< >>
Left	+ - .
Left	* / %
Right	! ~ ++ -- (int) (double) (string) (array) (object) @
Right	[
Non associative	new

PHP | Operators

Operators are used to perform operations on some values.

- Arithmetic Operators
- Logical or Relational Operators
- Comparison Operators
- Conditional or Ternary Operators
- Assignment Operators
- Spaceship Operators (Introduced in PHP 7)
- Array Operators
- Increment/Decrement Operators
- String Operators

Let us now learn about each of these operators in details:

Arithmetic Operators

The arithmetic operators are use to perform simple mathematical operations like addition, subtraction, multiplication etc. Below is the list of arithmetic operators along with there syntax and operations, that PHP provides us:

OPERATOR	NAME	SYNTAX	OPERATION
+	Addition	$x + $y	Sum the operands
–	Subtraction	$x – $y	Differences the operands
*	Multiplication	$x * $y	Product of the operands
/	Division	$x / $y	Quotient of the operands
**	Exponentiation	$x ** $y	$x raised to the power $y
%	Modulus	$x % $y	Remainder of the operands

```
<?php
$x= 29;
// variable 2
$y= 4;
// some arithmetic operations on
echo($x+ $y), "\n";
echo($x- $y), "\n";
echo($x* $y), "\n";
echo($x/ $y), "\n";
echo($x% $y), "\n";
?>
```

Output:

```
33
25
116
7.25
1
```

Logical or Relational Operators

These are basically used to operate with conditional statements and expressions.

Here are the logical operators along with there syntax and operations, that PHP provides us:

OPERATOR	NAME	SYNTAX	OPERATION
And	Logical AND	$x and $y	True if both the operands are true else false
Or	Logical OR	$x or $y	True if either of the operand is true else false
Xor	Logical XOR	$x xor $y	True if either of the operand is true and false if both are true

&&	Logical AND	$x &&$y	True if both the operands are true else false
||	Logical OR	$x || $y	True if either of the operand is true else false
!	Logical NOT	!$x	True if $x is false

Example:

```
<?php
```

```
 $x= 50;
$y= 30;
 if($x== 50 and$y== 30)
    echo"and Success \n";
 if($x== 50 or$y== 20)
    echo"or Success \n";

if($x== 50 xor$y== 20)
    echo"xor Success \n";

if($x== 50 && $y== 30)
    echo"&& Success \n";

if($x== 50 || $y== 20)
    echo"|| Success \n";
```

```
if(!$z)

    echo"! Success \n";

 ?>
```

Output:

```
and Success
or Success
xor Success
&& Success
|| Success
! Success
```

Comparison Operators

These operators are used to compare two elements and outputs the result in boolean form. Here are the comparison operators along with there syntax and operations, that PHP provides us:

OPERATOR	NAME	SYNTAX	OPERATION
==	Equal To	$x == $y	Returns True if both the operands are equal
!=	Not Equal To	$x != $y	Returns True if both the operands are not equal
<>	Not Equal To	$x <> $y	Returns True if both the operands are unequal
===	Identical	$x === $y	Returns True if both the operands

			are equal and are of the same type
!==	Not Identical	$x == $y	Returns True if both the operands are unequal and are of different types
<	Less Than	$x < $y	Returns True if $x is less than $y
>	Greater Than	$x > $y	Returns True if $x is greater than $y
<=	Less Than or Equal To	$x <= $y	Returns True if $x is less than or equal to $y

>=	Greater Than or Equal To	$x >= $y	Returns True if $x is greater than or equal to $y

Example:

```
<?php
$a= 80;
$b= 50;
$c= "80";
// Here var_dump function has been used to
// display structured information. We will learn
// about this function in complete details in further
// articles.
```

```
var_dump($a== $c) + "\n";
var_dump($a!= $b) + "\n";
var_dump($a<> $b) + "\n";
var_dump($a=== $c) + "\n";
var_dump($a!== $c) + "\n";
var_dump($a< $b) + "\n";
var_dump($a> $b) + "\n";
var_dump($a<= $b) + "\n";
var_dump($a>= $b);

?>
```

Output:

```
bool(true)
bool(true)
bool(true)
bool(false)
bool(true)
bool(false)
bool(true)
bool(false)
bool(true)
```

Conditional or Ternary Operators

These operators are used to compare two values and take either of the result simultaneously, depending on whether the outcome is TRUE or FALSE. These are also used as shorthand notation for if…else statement that we will read in the article on decision making.

Syntax:

```
$var = (condition)? value1 : value2;
```

Here, condition will either evaluate to true or false. If the condition evaluates to True, then value1 will be assigned to the variable $var otherwise value2 will be assigned to it.

OPERATOR	NAME	OPERATION
?:	Ternary	If condition is true ? then $x : or else $y. This means that if

condition is true then left result of the colon is accepted otherwise the result on right.

Example:

```
<?php
 $x= -12;
 echo($x> 0) ? 'The number is positive':
'The number is negative';
 ?>
```

Output:

```
The number is negative
```

Assignment Operators

These operators are used to assign values to different variable, with or without mid-operations. Here are the assignment operators along with there syntax and operations, that PHP provides us:

Example:

```
<?php

// simple assign operator
$y= 75;
echo$y, "\n";

// add then assign operator
$y= 100;
$y+= 200;
echo$y, "\n";

// subtract then assign operator
```

```
$y= 70;
$y-= 10;
echo$y, "\n";

// multiply then assign operator
$y= 30;
$y*= 20;
echo$y, "\n";

// Divide then assign(quotient) operator
$y= 100;
$y/= 5;
echo$y, "\n";

// Divide then assign(remainder) operator
$y= 50;
```

```
$y%= 5;
echo$y;


?>
```

Output:

```
75
300
60
600
20
0
```

Array Operators

These operators are used in case of arrays. Here are the array operators along with there syntax and operations, that PHP provides us:

Example:

```
<?php


$x= array("k"=> "Car", "l"=> "Bike");

$y= array("a"=> "Train", "b"=> "Plane");


var_dump($x+ $y);

var_dump($x== $y) + "\n";

var_dump($x!= $y) + "\n";

var_dump($x<> $y) + "\n";

var_dump($x=== $y) + "\n";

var_dump($x!== $y) + "\n";


?>
```

Output:

```
array(4) {
  ["k"]=>
  string(3) "Car"
  ["l"]=>
  string(4) "Bike"
  ["a"]=>
  string(5) "Train"
  ["b"]=>
  string(5) "Plane"
}
bool(false)
bool(true)
bool(true)
bool(false)
bool(true)
```

Increment/Decrement Operators

These are called the unary operators as it work on single operands. These are used to increment or decrement values.

Example:

```
<?php
x= 2;
echo++$x, " First increments then prints \n";
echo$x, "\n";

$x= 2;
echo$x++, " First prints then increments \n";
echo$x, "\n";

$x= 2;
echo--$x, " First decrements then prints \n";
echo$x, "\n";
```

```
$x= 2;

echo$x--, " First prints then decrements \n";

echo$x;

?>
```

Output:

```
3 First increments then prints
3
2 First prints then increments
3
1 First decrements then prints
1
2 First prints then decrements
1
```

String Operators

Example:

```
<?php

$x= "Geeks";
```

```
$y= "for";

$z= "Geeks!!!";

echo$x. $y. $z, "\n";

$x.= $y. $z;

echo$x;

?>
```

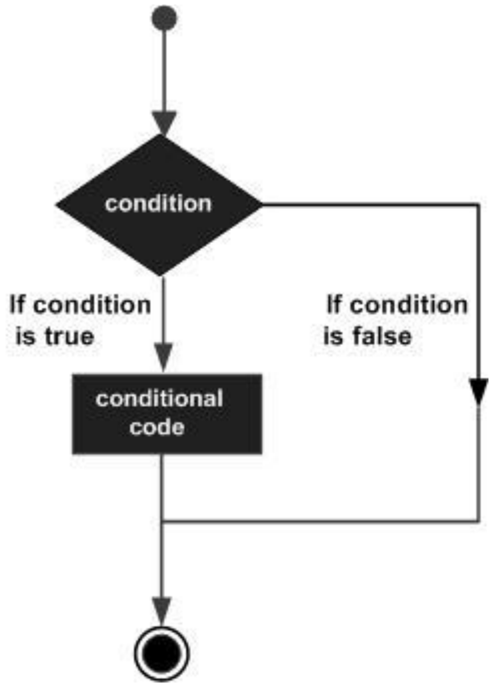
Output:

```
GeeksforGeeks!!!
GeeksforGeeks!!!
```

PHP - Decision Making

The if, elseif ...else and switch statements are used to take decision based on the different condition.

You can use conditional statements in your code to make your decisions. PHP supports following three decision making statements −



- **if...else statement** − use this statement if you want to execute a set of code when a condition is true and another if the condition is not true
- **elseif statement** − is used with the if...else statement to execute a set of code if **one** of the several condition is true

- **switch statement** − is used if you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.

The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.

Syntax

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

Example

The following example will output "Have a nice weekend!" if the current day is Friday, Otherwise, it will output "Have a nice day!":

```
<html>
<body>

<?php
```

```
        $d = date("D");

if($d =="Fri")
          echo "Have a nice weekend!";

else
          echo "Have a nice day!";
?>

</body>
</html>
```

```
<html>
<body>

<?php
```

```
        $d = date("D");

if($d =="Fri")
           echo "Have a nice weekend!";

        elseif ($d =="Sun")
           echo "Have a nice Sunday!";

else
           echo "Have a nice day!";
?>

</body>
</html>
```

It will produce the following result –

Have a nice Weekend!

The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement.

The switch statement is used to avoid long blocks of if..elseif..else code.

Syntax

switch (*expression*){
  case *label1:*
code to be executed if expression = label1;
    break;

  case *label2:*
code to be executed if expression = label2;
    break;
    default:

code to be executed
  if expression is different
  from both label1 and label2;
}

Example

```
<html>
<body>

<?php
        $d = date("D");
```

```
switch($d){
case"Mon":
		echo "Today is Monday";
break;

case"Tue":
		echo "Today is Tuesday";
break;

case"Wed":
		echo "Today is Wednesday";
break;

case"Thu":
		echo "Today is Thursday";
break;

case"Fri":
		echo "Today is Friday";
break;

case"Sat":
		echo "Today is Saturday";
break;
case"Sun":
              echo "Today is Sunday";
break;

default:
              echo "Wonder which day is this ?";
}
?>

</body>
</html>
```

It will produce the following result –
Today is Monday

PHP - Loop Types

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

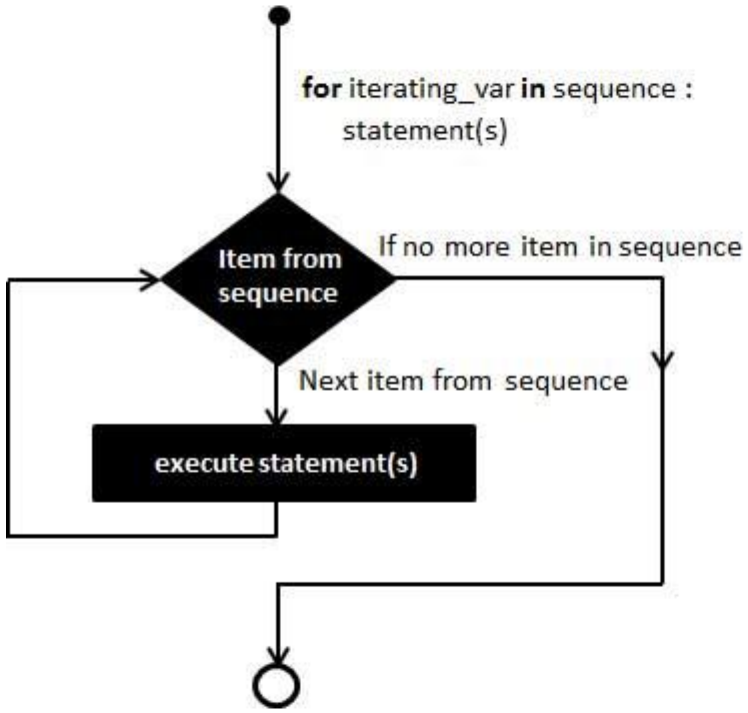
- **for** − loops through a block of code a specified number of times.
- **while** − loops through a block of code if and as long as a specified condition is true.

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

We will discuss about **continue** and **break** keywords used to control the loops execution.

The for loop statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.



Syntax

for (*initialization*; *condition*; *increment*){
code to be executed;
}

The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is traditional to name it $i.

Example

The following example makes five iterations and changes the assigned value of two variables on each pass of the loop −

```
<html>
<body>

<?php
        $a =0;
        $b =0;

for( $i =0; $i<5; $i++){
           $a +=10;
           $b +=5;
}
```

```
         echo ("At the end of the loop a = $a and b
= $b");
?>

</body>
</html>
```

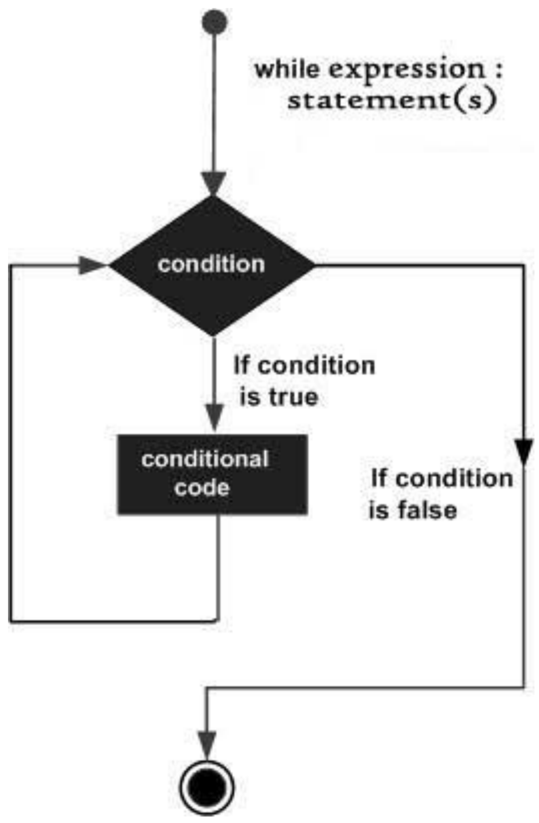
This will produce the following result −

```
At the end of the loop a = 50 and b = 25
```

The while loop statement

The while statement will execute a block of code if and as long as a test expression is true.

If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.



Syntax

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

Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation is false and the loop ends

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

```
<?php
      $i =0;
      $num =50;

while( $i <10){
         $num--;
         $i++;
}

      echo ("Loop stopped at i = $i and num =
$num");
?>

</body>
</html>
```

This will produce the following result −

```
Loop stopped at i = 10 and num = 40
```

The do...while loop statement

The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

Syntax

```
do {
```

code to be executed;
}
while (*condition*);

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 10 –

```
<html>
<body>

<?php
        $i =0;
        $num =0;

do{
           $i++;
}
```

```
while( $i <10);
        echo ("Loop stopped at i = $i");
?>

</body>
</html>
```

This will produce the following result –

```
Loop stopped at i = 10
```

The foreach loop statement

The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax

Foreach (*array* as *value*) {
code to be executed;
}

Example

```
<html>
<body>

```

```
<?php
        $array = array(1,2,3,4,5);

foreach( $array as $value ){
            echo "Value is $value <br />";
}
?>

</body>
</html>
```

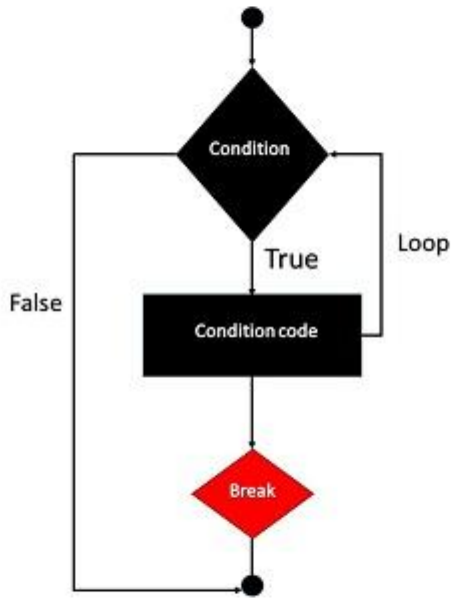
This will produce the following result −

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
```

The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely.

The **break** statement is situated inside the statement block. It gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.



Example

In the following example condition test becomes true when the counter value reaches 3 and loop terminates.

```
<html>
<body>

<?php
        $i =0;

while( $i <10){
           $i++;
if( $i ==3)break;
}
        echo ("Loop stopped at i = $i");
?>

</body>
</html>
```

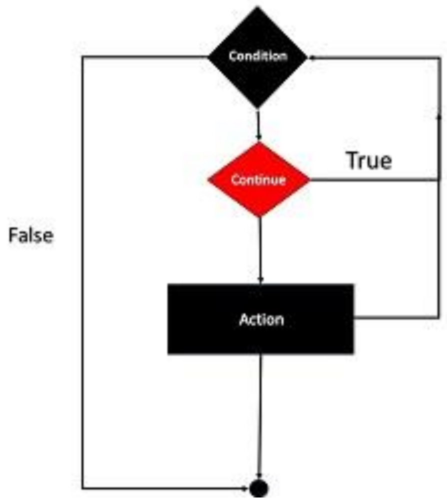
This will produce the following result −

```
Loop stopped at i = 3
```

The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop.

Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.



Example

In the following example loop prints the value of array but for which condition becomes true it just skip the code and next value is printed.

```
<html>
<body>

<?php
       $array = array(1,2,3,4,5);

foreach( $array as $value ){
if( $value ==3)continue;
         echo "Value is $value <br />";
}
?>
```

```

</body>
</html>
```

This will produce the following result −

```
Value is 1
Value is 2
Value is 4
Value is 5
```

PHP - Arrays

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

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

- **Numeric array** − An array with a numeric index. Values are stored and accessed in linear fashion.
- **Associative array** − An array with strings as index. This stores element values in

association with key values rather than in a strict linear index order.

- **Multidimensional array** − An array containing one or more arrays and values are accessed using multiple indices

NOTE − Built-in array functions is given in function reference PHP Array Functions

Numeric Array

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

Example

Following is the example showing how to create and access numeric arrays.

Here we have used **array()** function to create array. This function is explained in function reference.

```
<html>
```

```
<body>

<?php
/* First method to create array. */
        $numbers = array(1,2,3,4,5);

foreach( $numbers as $value ){
           echo "Value is $value <br />";
}

/* Second method to create array. */
        $numbers[0]="one";
        $numbers[1]="two";
        $numbers[2]="three";
        $numbers[3]="four";
        $numbers[4]="five";

foreach( $numbers as $value ){
           echo "Value is $value <br />";
}
?>

</body>
</html>
```

This will produce the following result −

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

Associative Arrays

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

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

NOTE − Don't keep associative array inside double quote while printing otherwise it would not return any value.

Example

```
<html>
<body>

<?php
/* First method to associate create array. */
        $salaries                                    =
array("mohammad"=>2000,"qadir"=>1000,"zara
"=>500);

        echo   "Salary   of   mohammad   is   ".
$salaries['mohammad']."<br />";
        echo    "Salary    of    qadir    is    ".
$salaries['qadir']."<br />";
        echo    "Salary    of    zara    is    ".
$salaries['zara']."<br />";
```

```
/* Second method to create array. */
         $salaries['mohammad']="high";
         $salaries['qadir']="medium";
         $salaries['zara']="low";

         echo "Salary of mohammad is ". $salaries['mohammad']."<br />";
         echo "Salary of qadir is ". $salaries['qadir']."<br />";
         echo "Salary of zara is ". $salaries['zara']."<br />";
?>

</body>
</html>
```

This will produce the following result −

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

Multidimensional Arrays

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

Example

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

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

```
<html>
<body>

<?php
        $marks = array(
"mohammad"=> array (
"physics"=>35,
"maths"=>30,
"chemistry"=>39
),

"qadir"=> array (
"physics"=>30,
"maths"=>32,
"chemistry"=>29
),

"zara"=> array (
"physics"=>31,
"maths"=>22,
"chemistry"=>39
)
);

/* Accessing multi-dimensional array values */
        echo "Marks for mohammad in physics : ";
        echo  $marks['mohammad']['physics']."<br
/>";

        echo "Marks for qadir in maths : ";
        echo $marks['qadir']['maths']."<br />";

        echo "Marks for zara in chemistry : ";
        echo $marks['zara']['chemistry']."<br />";
?>

</body>
```

```
</html>
```

This will produce the following result −

```
Marks for mohammad in physics : 35
Marks for qadir in maths : 32
Marks for zara in chemistry : 39
```

PHP - Functions

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

You already have seen many functions like **fopen()** and **fread()** etc. They are built-in functions but PHP gives you option to create your own functions as well.

There are two parts which should be clear to you −

- Creating a PHP Function
- Calling a PHP Function

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

Please refer to PHP Function Reference for a complete set of useful functions.

Creating PHP Function

Its very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called writeMessage() and then calls it just after creating it.

Note that while creating a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below −

```
<html>

<head>
<title>Writing PHP Function</title>
</head>

<body>
```

```
<?php
/* Defining a PHP Function */
function writeMessage(){
        echo "You are really a nice person,
Have a nice time!";
}

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

</body>
</html>
```

This will display following result −

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

PHP Functions with Parameters

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

Live Demo

```
<html>

<head>
<title>Writing PHP Function with Parameters</title>
</head>

<body>

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

       addFunction(10,20);
?>

</body>
</html>
```

This will display following result −

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

Passing Arguments by Reference

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

Following example depicts both the cases.

Live Demo

```
<html>

<head>
<title>Passing Argument by Reference</title>
</head>

<body>
```

```
<?php
function addFive($num){
          $num +=5;
}

function addSix(&$num){
          $num +=6;
}


        $orignum =10;
        addFive( $orignum );

        echo "Original Value is $orignum<br />";

        addSix( $orignum );
        echo "Original Value is $orignum<br />";
?>


</body>
</html>
```

This will display following result −

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

PHP Functions returning value

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

You can return more than one value from a function using **return array(1,2,3,4)**.

Following example takes two integer parameters and add them together and then returns their sum to the calling program. Note that **return** keyword is used to return a value from a function.

```
<html>

<head>
<title>Writing PHP Function which returns value</title>
</head>

<body>

<?php
function addFunction($num1, $num2){
```

```
            $sum = $num1 + $num2;
return $sum;
}
         $return_value = addFunction(10,20);

         echo "Returned value from the function : $return_value";
?>

</body>
</html>
```

This will display following result −

```
Returned value from the function : 30
```

Setting Default Values for Function Parameters

You can set a parameter to have a default value if the function's caller doesn't pass it.

Following function prints NULL in case use does not pass any value to this function.

Live Demo

```
<html>

<head>
```

```
<title>Writing PHP Function which returns
value</title>
</head>

<body>

<?php
function printMe($param = NULL){
print $param;
}

        printMe("This is test");
        printMe();
?>

</body>
</html>
```

This will produce following result −

```
This is test
```

Dynamic Function Calls

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

Live Demo

```
<html>

<head>
<title>Dynamic Function Calls</title>
</head>

<body>

<?php
function sayHello(){
          echo "Hello<br />";
}

        $function_holder ="sayHello";
        $function_holder();
?>

</body>
</html>
```

This will display following result −

Hello

FormProcessing

PHP Form Processing

In this article, we will discuss how to process form in PHP. HTML forms are used to send the user information to the server and returns the result back to the browser. For example, if you want to get the details of visitors to your website, and send them good thoughts, you can collect the user information by means of form processing. Then, the information can be validated either at the client-side or on the server-side. The final result is sent to the client through the respective web browser. To create a HTML form, **form** tag should be used.

Attributes of Form Tag:

ATTRIBUTE	DESCRIPTION
name or id	**It specifies the name of the form and is used to identify individual forms.**

action	**It specifies the location to which the form data has to be sent when the form is submitted.**
method	**It specifies the HTTP method that is to be used when the form is submitted. The possible values are** get **and** post**. If** get **method is used, the form data are visible to the users in the url. Default HTTP method is** get**.**
encType	**It specifies the encryption type for the form data when the form is submitted.**
novalidate	**It implies the server not to verify the form data when the form is submitted.**

Controls used in forms: Form processing contains a set of controls through which the client and server can communicate and share information. The controls used in forms are:

- **Textbox:** Textbox allows the user to provide single-line input, which can be used for getting values such as names, search menu and etc.
- **Textarea:** Textarea allows the user to provide multi-line input, which can be used for getting values such as an address, message etc.
- **DropDown:** Dropdown or combobox allows the user to provide select a value from a list of values.
- **Radio Buttons:** Radio buttons allow the user to select only one option from the given set of options.
- **CheckBox:** Checkbox allows the user to select multiple options from the set of given options.
- **Buttons:** Buttons are the clickable controls that can be used to submit the form.

Creating a simple HTML Form: All the form controls given above is designed by using the **input** tag based on the **type** attribute of the tag. In the below script, when the form is submitted, no event handling mechanism is done. Event handling refers to the process done while the form is submitted. These event handling mechanisms can be done by using javaScript or PHP. However, JavaScript provides only client-side validation. Hence, we can use PHP for form processing.

HTML Code:

```
<!DOCTYPE html>

<html>

 <head>

   <title>Simple Form Processing</title>

</head>

 <body>

   <formid="form1"method="post">

       FirstName:

       <inputtype="text"name="firstname"require
d/>

       <br>

       <br>
```

```
        LastName
        <inputtype="text"name="lastname"require
d/>
        <br>
        <br>
        Address
        <inputtype="text"name="address"required/
>
        <br>
        <br>
        Email Address:
        <inputtype="email"name="emailaddress"re
quired/>
        <br>
        <br>
        Password:
        <inputtype="password"name="password"r
equired/>
        <br>
        <br>
        <inputtype="submit"value="Submit”/>
    </form>
</body>
</html>
```

Form Validation: Form validation is done to ensure that the user has provided the relevant information. Basic validation can be done using HTML elements. For example, in the above script, the email address text box is having a type value as “email”, which prevents the user from entering the incorrect value for an email. Every form field in the above script is followed by a required attribute, which will intimate the user not to leave any field empty before submitting the form. PHP methods and arrays used in form processing are:

- **isset():** This function is used to determine whether the variable or a form control is having a value or not.
- **$_GET[]:** It is used the retrieve the information from the form control through the

parameters sent in the URL. It takes the attribute given in the url as the parameter.

- **$_POST[]:** It is used the retrieve the information from the form control through the HTTP POST method. IT takes name attribute of corresponding form control as the parameter.
- **$_REQUEST[]:** It is used to retrieve an information while using a database.

Form Processing using PHP: Above HTML script is rewritten using the above mentioned functions and array. The rewritten script validates all the form fields and if there are no errors, it displays the received information in a tabular form.

```
<?php
if(isset($_POST['submit']))
{
    if((!isset($_POST['firstname']))                ||
(!isset($_POST['lastname'])) ||
        (!isset($_POST['address']))                  ||
    (!isset($_POST['emailaddress'])) ||

        (!isset($_POST['password']))                 ||
    (!isset($_POST['gender'])))

      {

        $error= "*". "Please fill all the required
    fields";

      }

      else

      {

        $firstname= $_POST['firstname'];

        $lastname= $_POST['lastname'];

        $address= $_POST['address'];

        $emailaddress=
    $_POST['emailaddress'];

        $password= $_POST['password'];

        $gender= $_POST['gender'];
    }
}
?>
<html>

<head>
    <title>Simple Form Processing</title>
</head>

<body>
    <h1>Form Processing using PHP</h1>
    <fieldset>
        <form
id="form1"method="post"action="form.php">
            <?php
                if(isset($_POST['submit']))
```

```
{
    if(isset($error))
    {
        echo"<p style='color:red;'>"
        . $error. "</p>";
    }
}
?>

FirstName:
<input
type="text"name="firstname"/>
 <span style="color:red;">*</span>
<br>
<br>
Last Name:
<input
```

```
type="text"name="lastname"/>
                  <span style="color:red;">*</span>
                <br>
                <br>
                Address:
                <input type="text"name="address"/>
                  <span style="color:red;">*</span>
                <br>
                <br>
                Email:
                <input
type="email"name="emailaddress"/>
                  <span style="color:red;">*</span>
                <br>
                <br>
                Password:
                <input
```

```
type="password"name="password"/>
                    <span style="color:red;">*</span>
                <br>
                <br>
                Gender:
                <input type="radio"
                       value="Male"
                       name="gender"> Male
                <input type="radio"
                       value="Female"
                       name="gender">Female
                <br>
                <br>
                <input
type="submit"value="Submit"name="submit"/>
          </form>
```

```
    </fieldset>
    <?php
      if(isset($_POST['submit']))
      {
        if(!isset($error))
         {
                echo"<h1>INPUT
RECEIVED</h1><br>";
                echo"<table border='1'>";
                echo"<thead>";
                echo"<th>Parameter</th>";
                echo"<th>Value</th>";
                echo"</thead>";
                echo"<tr>";
                echo"<td>First Name</td>";
                echo"<td>".$firstname."</td>";
```

```
echo"</tr>";
echo"<tr>";
echo"<td>Last Name</td>";
echo"<td>".$lastname."</td>";
echo"</tr>";
echo"<tr>";
echo"<td>Address</td>";
echo"<td>".$address."</td>";
echo"</tr>";
echo"<tr>";
echo"<td>Email Address</td>";
echo"<td>".$emailaddress."</td>";
echo"</tr>";
echo"<tr>";
echo"<td>Password</td>";
echo"<td>".$password."</td>";
```

```
                    echo"</tr>";

                    echo"<tr>";

                    echo"<td>Gender</td>";

                    echo"<td>".$gender."</td>";

                    echo"</tr>";

                    echo"</table>";

            }

          }

        ?>

    </body>


    </html>
```

- **Output:**

Note: When the PHP and HTML are coded in a single file, the file should be saved as PHP. In the form, the value for the action parameter should be a file name.

PHP Cookies

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the **server embeds on the user's computer**. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the `setcookie()` function.

Syntax

setcookie(*name, value, expire, path, domain, secure, httponly*);

Only the *name* parameter is required. All other parameters are optional.

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time()
+ (86400 * 30), "/"); // 86400 = 1 day
?>
```

```
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
  echo "Cookie named '" . $cookie_name . "' is
not set!";
} else {
  echo "Cookie '" . $cookie_name . "' is
set!<br>";
  echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Run example »

Note: The setcookie() function must appear BEFORE the <html> tag.

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time()
+ (86400 * 30), "/");
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
  echo "Cookie named '" . $cookie_name . "' is
not set!";
} else {
  echo "Cookie '" . $cookie_name . "' is
set!<br>";
  echo "Value is: " . $_COOKIE[$cookie_name];
}
```

```
?>

</body>
</html>
```

Delete a Cookie

To delete a cookie, use the `setcookie()` function with an expiration date in the past:

Example

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>

<?php
echo "Cookie 'user' is deleted.";
?>
```

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

Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the `setcookie()` function, then count the $_COOKIE array variable:

Example

```
<?php
setcookie("test_cookie", "test", time()
+ 3600, '/');
?>
<html>
<body>

<?php
if(count($_COOKIE) > 0) {
  echo "Cookies are enabled.";
} else {
  echo "Cookies are disabled.";
}
?>

</body>
</html>
```

PHP Sessions

A session is a way to store information

(in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.

What is a PHP Session?

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

Start a PHP Session

A session is started with the session_start() function.

Session variables are set with the PHP global variable: $_SESSION.

Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

Example

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
```

```
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

Note: The session_start() function must be the very first thing in your document. Before any HTML tags.

Get PHP Session Variable Values

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).

Also notice that all session variable values are stored in the global $_SESSION variable:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Echo session variables that were set on
```

```
previous page
echo "Favorite color is
" . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is
" . $_SESSION["favanimal"] . ".";
?>

</body>
</html>
```

Another way to show all the session variable values for a user session is to run the following code:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
```

```
<?php
print_r($_SESSION);
?>

</body>
</html>
```

How does it work? How does it know it's me?

Most sessions set a user-key on the user's computer that looks something like this: 765487cf34ert8dede5a562e4f3a7e12. Then, when a session is opened on another page, it scans the computer for a user-key. If there is a match, it accesses that session, if not, it starts a new session.

Modify a PHP Session Variable

To change a session variable, just overwrite it:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>

</body>
</html>
```

Run example »

Destroy a PHP Session

To remove all global session variables and destroy the session, use session_unset() and session_destroy():

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

</body>
</html>
```

Run example »

PHP MySQL Database

With PHP, you can connect to and manipulate databases.

MySQL is the most popular database system used with PHP.

What is MySQL?

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

- MySQL is developed, distributed, and supported by Oracle Corporation
- MySQL is named after co-founder Monty Widenius's daughter: My

The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

Databases are useful for storing information categorically. A company may have a database with the following tables:

- Employees
- Products
- Customers
- Orders

PHP + MySQL Database System

- PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform)

Database Queries

A query is a question or a request.

We can query a database for specific information and have a recordset returned.

Look at the following query (using standard SQL):

```
SELECT LastName FROM Employees
```

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

To learn more about SQL, please visit our SQL tutorial.

Download MySQL Database

If you don't have a PHP server with a MySQL Database, you can download it for free here: http://www.mysql.com

---

Facts About MySQL Database

MySQL is the de-facto standard database system for web sites with HUGE volumes of both data and end-users (like Facebook, Twitter, and Wikipedia).

Another great thing about MySQL is that it can be scaled down to support embedded database applications.

Look at http://www.mysql.com/customers/ for an overview of companies using MySQL.

PreviousNext

Mysql connect

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

// Create connection
```

```
$conn = new mysqli($servername, $username,
$password);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn-
>connect_error);
}
echo "Connected successfully";
?>
```

MYSQL CREATE TABLE

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

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

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT
PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT
CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
  echo "Table MyGuests created successfully";
} else {
  echo "Error creating table: " . $conn->error;
}

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

FILE HANDLING CONCEPTS

File handling is an important part of any web application. You often need to open and process a file for different tasks.

PHP Manipulating Files

PHP has several functions for creating, reading, uploading, and editing files.

PHP Open File - fopen()

A better method to open files is with the `fopen()` function. This function gives you more options than the `readfile()` function.

We will use the text file, "webdictionary.txt", during the lessons:

The first parameter of `fopen()` contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

Example

```
<?php
$myfile =
fopen("webdictionary.txt", "r") or die("U
nable to open file!");
echo fread($myfile,filesize("webdictionar
y.txt"));
fclose($myfile);
?>
```

Run example »

Tip: The `fread()` and the `fclose()` functions will be explained below.

The file may be opened in one of the following modes:

Modes	Description
R	**Open a file for read only**. File pointer starts at the beginning of the file

W	**Open a file for write only**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
A	**Open a file for write only**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
X	**Creates a new file for write only**. Returns FALSE and an error if file already exists
r+	**Open a file for read/write**. File pointer starts at the beginning of the file
w+	**Open a file for read/write**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	**Open a file for read/write**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file

	doesn't exist
x+	**Creates a new file for read/write**. Returns FALSE and an error if file already exists

PHP Read File - fread()

The `fread()` function reads from an open file.

The first parameter of `fread()` contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

```
fread($myfile,filesize("webdictionary.txt")
);
```

PHP Close File - fclose()

The `fclose()` function is used to close an open file.

It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

The `fclose()` requires the name of the file (or a variable that holds the filename) we want to close:

```
<?php
$myfile = fopen("webdictionary.txt", "r");
// some code to be executed....
fclose($myfile);
?>
```

PHP Read Single Line - fgets()

The `fgets()` function is used to read a single line from a file.

The example below outputs the first line of the "webdictionary.txt" file:

Example

```
<?php
$myfile =
fopen("webdictionary.txt", "r") or die("Una
ble to open file!");
echo fgets($myfile);
fclose($myfile);
?>
```

Run example »

Note: After a call to the `fgets()` function, the file pointer has moved to the next line.

PHP Check End-Of-File - feof()

The `feof()` function checks if the "end-of-file" (EOF) has been reached.

The `feof()` function is useful for looping through data of unknown length.

The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

Example

```
<?php
$myfile =
fopen("webdictionary.txt", "r") or die("Una
ble to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

Run example »

PHP Read Single Character - fgetc()

The `fgetc()` function is used to read a single character from a file.

The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

Example

```
<?php
$myfile =
fopen("webdictionary.txt", "r") or die("Una
ble to open file!");
// Output one character until end-of-file
while(!feof($myfile)) {
  echo fgetc($myfile);
}
fclose($myfile);
?>
```

Run example »

Note: After a call to the `fgetc()` function, the file pointer moves to the next character.

Exercise:

Open a file, and write the correct syntax to output one character at the time, until end-of-file.

```
$myfile = fopen("webdict.txt", "r");
while(!☐($myfile)) {
  echo ☐($myfile);
}
```

Unit – 4

Perl back grounder-perl overview-perl parsing rules -variables &data-statements &control structures-subroutines-packages &modules - working with files - data manipulation

PERL INTRODUCTION

Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development, and more.

What is Perl?

- Perl is a stable, cross platform programming language.
- Though Perl is not officially an acronym but few people used it as Practical Extraction and Report Language.
- It is used for mission critical projects in the public and private sectors.
- Perl is an *Open Source* software, licensed under its *Artistic License*, or the *GNU General Public License (GPL)*.
- Perl was created by Larry Wall.
- Perl 1.0 was released to usenet's alt.comp.sources in 1987.
- At the time of writing this tutorial, the latest version of perl was 5.16.2.

Perl features:

1. Mission critical

   Used for mission critical projects in the public and private sectors.

2. Object-oriented, procedural and functional

Supports object-oriented, procedural and functional programming.

3. Easily extendible

   There are over 25,000 open source modules available from the Comprehensive Perl Archive Network (CPAN).

4. Text manipulation

   Perl includes powerful tools for processing text that make it ideal for working with HTML, XML, and all other mark-up and natural languages.

5. Unicode support

   Supports Unicode version 6 (from Perl 5.14).

6. Database integration

   Perl's database integration interface (DBI) supports third-party databases including Oracle,
   Sybase, Postgres, MySQL and many others.

7. C/C++ library interface

   Perl interfaces with external C/C++ libraries through XS or SWIG.

8. Embeddable

   The Perl interpreter can be embedded into other systems such as web servers and database servers.

9. Open Source

   Perl is Open Source software, licensed under its Artistic License, or the GNU General Public License (GPL).

Perl and the Web:

- Ideal web programming language
- Web Frameworks
- Database integration
- Web modules
- Duct-tape of the internet, and more...
- Encryption capable
- Embed into Apache

Advantages of Perl:

- Perl Provides supports for cross platform and it is compatible with mark-up languages like HTML, XML etc.
- It is very efficient in text-manipulation i.e. Regular Expression. It also provides the socket capability.
- It is free and a Open Source software which is licensed under *Artistic* and *GNU General Public License (GPL)*.
- It is an embeddable language that's why it can embed in web servers and database servers.
- It supports more than 25, 000 open source modules on CPAN(Comprehensive Perl Archive Network) which provide many powerful extensions to the standard library. For example, XML processing, GUI(Graphical User Interface) and DI(Database Integration) etc.

Disadvantages of Perl:

- Perl doesn't supports portability due to CPAN modules.
- Programs runs slowly and program needs to be interpreted each time when any changes are made.
- In Perl, the same result can be achieved in several different ways which make the code untidy as well as unreadable.
- Usability factor is lower when compared to other languages.

Applications:

- One of the major application of Perl language is to processing of text files and analysis of the strings.
- Perl also used for *CGI(Common Gateway Interface)* scripts.
- Used in web development, GUI(Graphical User Interface) development.
- Perl’s text-handling capabilities is also used for generating SQL queries.

Programming in Perl:

- Since the Perl is a lot similar to other widely used languages syntactically, it is easier to code and learn in Perl.
- Programs can be written in Perl in any of the widely used text editors like Notepad++, gedit etc.
- After writing the program save the file with the extension *.pl* or *.PL*
- To run the program use perl file_name.pl on the command line.

Example:

A simple program to print *Welcome to GFG!*

```
# Perl program to print Welcome to GFG!

#!/usr/bin/perl

# Below line will print "Welcome to GFG!"

print "Welcome to GFG!\n";
```

Output:

```
Welcome to GFG!
```

Comments:

Comments are used for enhancing the readability of the code. The interpreter will ignore the comment entries and does not execute them. There are two types of comment in Perl:

Single line comments:

- Perl single line comment starts with hashtag symbol with no white spaces (#) and lasts till the end of the line.
- If the comment exceeds one line then put a hashtag on the next line and continue the comment.
- Perl's single line comments are proved useful for supplying short explanations for variables, function declarations, and expressions.

Syntax:

```
# Single line comment
```

Example:

```
#!/usr/bin/perl

$b = 10;    # Assigning value to $b

$c = 30;    # Assigning value to $c

$a = $b + $c;   # Performing the operation

print "$a";     # Printing the result
```

Multi-line string as a comment:

- Perl multi-line comment is a piece of text enclosed within "=" and "=cut".
- They are useful when the comment text does not fit into one line; therefore needs to span across lines.
- Multi-line comments or paragraphs serve as documentation for others reading your code.
- Perl considers anything written after the '=' sign as a comment until it is accompanied by a '=cut' at the end.
- Please note that there should be no whitespace after the '=' sign.

Syntax:

```
= Multi line comments
Line start from  = is interpreted as the
starting of multiline comment and =cut is
consider as the end of multiline comment
=cut
```

Example:

```
#!/usr/bin/perl
  =Assigning values to
variable $b and $c
=cut
$b = 10;
$c = 30;
  =Performing the operation
and printing the result
=cut
$a = $b + $c;
print "$a";
```

PERL OVERVIEW

- According to Wall, Perl has two slogans.
- The first is "There's more than one way to do it," commonly known as TMTOWTD.
- The second slogan is "Easy things should be easy and hard things should be possible".

OVERVIEW

- Features
- Design
- Appilication
- Implementation
- Availability
- Database interfaces
- Comparative performance
- Optimizing

FEATURES

- The overall structure of Perl derives broadly from C.
- Perl is procedural in nature, with variables ,expressions,assignment statements, brace-delimited blocks, control structures, subroutines.
- Perl also takes features from shell programming.
- All variables are marked with leading sigils, which allow variables

  to be interpolated directly into strings.
- Perls database integration interface DBI supports third-party

  databases including Oracle, Sybase, Postgres, MySQL and others.
- Perl works with HTML, XML, and other mark-up languages.
- Perl supports Unicode.
- Perl supports both procedural and object-oriented programming.
- Perl interfaces with external C/C++ libraries through XS or SWIG.
- Perl is extensible. There are over 20,000 third party modules available from the Comprehensive Perl Archive Network (CPAN).
- The Perl interpreter can be embedded into other systems..

Design

- The design of Perl can be understood as a response to

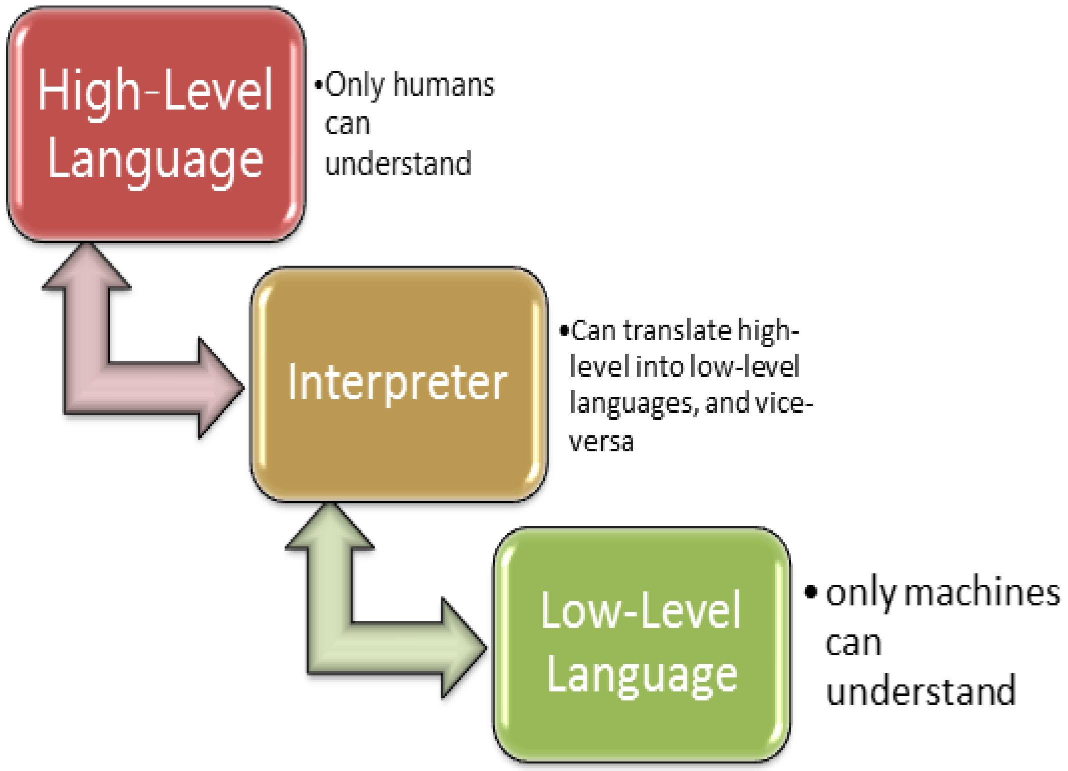
three broad trends in the

computer industry: falling hardware costs, rising labor

costs, and improvements in compiler technology.

- Many earlier computer languages, such as Fortran and C,aimed to make efficient use of expensive computer hardware.
- In contrast, Perl was designed so that computer programmers could write programs more quickly and easily.

- Perl does not enforce any particular programming paradigm (procedural, object-oriented, functional, or others) or even require the programmer to choose among them.
- No written specification or standard for the Perl language exists for Perl versions through Perl 5, and there are no plans to create one for the current version of Perl.
- There has been only one implementation of the interpreter, and the language has evolved along with it.


High-Level Language
•Only humans can understand
Interpreter
•Can translate high-level into low-level languages, and vice-versa
Low-Level Language
• only machines can understand

- Perl has many features that ease the task of the programmer at the expense of greater CPU and memory requirements.
- These include automatic memory management; dynamic typing; strings, lists, and hashes; regular expressions; introspection; and an `eval()` function.

Application

- Perl has many and variety applications, compounded by the availability of many standard and third-party modules.
- Perl has chiefly been used to write CGI scripts: large projects written in Perl include cPanel, Slash, Bugzilla, RT, TWiki,
- It is also an optional component of the popular LAMP technology stack for Web development, in lieu of PHP or Python.
- Perl is used extensively as a system programming language in the Debian GNU/Linux distribution.[81]

- The combination makes Perl a popular all-purpose language for system administrators, particularly because short programs, often called "one-liner programs," can be entered and run on a single command line.
- Perl code can be made portable across Windows and Unix; such code is often used by suppliers of software to simplify packaging and maintenance of software build- and deployment-scripts

Perl is available on a wide variety of platforms −.

- Unix (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX etc
- Win 9x/NT/2000/
- WinCE
- Macintosh (PPC, 68K)
- Solaris (x86, SPARC)
- OpenVMS
- Alpha (7.2 and later)

- Graphical user interfaces (GUIs) may be developed using Perl.
- For example, Perl/Tk and wxPerl are commonly used to enable user interaction with Perl scripts. Such interaction may be synchronous or asynchronous, using callbacks to update the GUI.

Implementation

- Perl is implemented as a core interpreter, written in C,together with a large collection of modules, written in Perl and C.
- The interpreter has an object-oriented architecture. All of the elements of the Perl language scalars, arrays, hashes, coderefs, file handlesare representent in

  the interpreter by C structs.

- The life of a Perl interpreter divides broadly into a compile phase and a run phase.[83]
- In Perl, the **phases** are the major stages in the interpreter's life-cycle. Each interpreter goes through each phase only once, and the phases follow in a fixed sequence.
- Perl is distributed with over 250,000 functional tests for core Perl language.

Availability

- Perl is dual licensed under both the Artistic License 1.0 and the GNU General Public License.[6] Distributions are available for most operating systems. It is particularly prevalent on Unix and Unix-like systems, but it has been ported to most modern (and many obsolete) platforms.

- Because of unusual changes required for the classic Mac OS environment, a special port called MacPerl was shipped independently.[89
- The Comprehensive Perl Archive Network carries a complete list of supported platforms with links to the distributions available on each.

Optimizing

- Because Perl is an interpreted language, it can give problems when efficiency is critical; in such situations, the most critical routines can be written in other languages (such as C), which can be connected to Perl via simple In line modules or the more complex but flexible XS mechanism.

Database interfaces

- Perl's text-handling capabilities can be used for generating SQL queries; arrays, hashes, and automatic memory management make it easy to collect and process the returned data In Perl 5, database interfaces are implemented by Perl DBI modules.
- The DBI (Database Interface) module presents a single, database independent interface to Perl applications, while the DBD (Database Driver) modules handle the details of accessing some 50 different databases; there are DBD drivers for most ANSI SQL databases.

Comparative performance

- The Computer Language Benchmarks compares the performance of implementations of typical programming problems in several programming languages.
- Large Perl programs start more slowly than similar programs in compiled languages because perl has to compile the source every time it runs.
- In a talk at the YAPC::Europe 2005 conference and subsequent article "A Timely Start," Jean-Louis Leroy found

that his Perl programs took much longer to run than expected because the perl interpreter spent significant time finding modules within his over-large include path.

- ❖ Once Perl code is compiled, there is additional overhead during the execution phase that typically is not present for programs written in compiled languages **such as C or C++.**[1

PERL DATATYPES AND VARIABLES

Perl Data Types:

- Data types specify the type of data that a valid Perl variable can hold. Perl is a **loosely typed language**.
- There is no need to specify a type for the data while using in the Perl program.
- The Perl interpreter will choose the type based on the context of the data itself.

There are 3 data types in Perl as follows:

1. **Scalars**
2. **Arrays**
3. **Hashes(Associative Arrays)**

1. Scalars:

- It is a single unit of data which can be an integer number, floating point, a character, a string, a paragraph, or an entire web page.
- They are preceded by a dollar sign ($).

Example:

```
# Perl Program to demonstrate the
# Scalars data types
# An integer assignment
$age = 1;
# A string
$name = "ABC";
# A floating point
$salary = 21.5;
# displaying result
print "Age = $age\n";
print "Name = $name\n";
print "Salary = $salary\n";
```

Output:

```
Age = 1

Name = ABC

Salary = 21.5
```

- **Scalar Operations:**
  - There are many operations which can be performed on the scalar data types like addition, subtraction, multiplication etc.

Example:

```
# Perl Program to demonstrate
# the Scalars operations
#!/usr/bin/perl
# Concatenates strings
$str = "GFG" . " is the best";
# adds two numbers
$num = 1 + 0;
# multiplies two numbers
$mul = 4 * 9;
# concatenates string and number
$mix = $str . $num;
# displaying result
print "str = $str\n";
print "num = $num\n";
print "mul = $mul\n";
print "mix = $mix\n";
```

Output:

```
str = GFG is the best

num = 1

mul = 36

mix = GFG is the best1
```

2. Arrays:

- An array is a variable that stores the value of the same data type in the form of a list.
- To declare an array in Perl, we use '@' sign in front of the variable name.

Example:

```
@age=(10, 20, 30)
```

It will create an array of integers which contains the value 10, 20 and 30. To access a single element of an array, we use the '$' sign.

```
$age[0]
```

It will produce the output as 10.

Example:

```
# Perl Program to demonstrate
# the Arrays data type
#!/usr/bin/perl
# creation of arrays
@ages = (33, 31, 27);
@names = ("Geeks", "for", "Geeks");
# displaying result
print "\$ages[0] = $ages[0]\n";
print "\$ages[1] = $ages[1]\n";
print "\$ages[2] = $ages[2]\n";
print "\$names[0] = $names[0]\n";
print "\$names[1] = $names[1]\n";
print "\$names[2] = $names[2]\n";
```

Output:

```
$ages[0] = 33

$ages[1] = 31

$ages[2] = 27

$names[0] = Geeks

$names[1] = for

$names[2] = Geeks
```

3. Hashes(Associative Arrays):

- It is a set of key-value pair.
- It is also termed as the Associative Arrays.
- To declare a hash in Perl, we use the '%' sign. To access the particular value, we use the '$' symbol which is followed by the key in braces.

Example:

```
#!/usr/bin/perl
%data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);
```

```
print "\$data{'John Paul'} = $data{'John Paul'}\n";
print "\$data{'Lisa'} = $data{'Lisa'}\n";
print "\$data{'Kumar'} = $data{'Kumar'}\n";
```

This will produce the following result –

```
$data{'John Paul'} = 45
$data{'Lisa'} = 30
$data{'Kumar'} = 40
```

Perl Variables:

- Variables in Perl are used to store and manipulate data throughout the program.
- When a variable is created it occupies memory space.
- The data type of a variable helps the interpreter to allocate memory and decide what to be stored in the reserved memory.
- Therefore, variables can store integers, decimals, or strings with the assignment of different data types to the variables.

Naming of a Variable:

A variable in Perl can be named anything with the use of a specific datatype. There are some rules to follow while naming a variable:

- Variables in Perl are case sensitive.
  Example:
  ```
  $John and $john are two different variables
  ```
- It starts with $, @ or % as per the datatype required, followed by zero or more letters, underscores and digits
- Variables in Perl cannot contain white spaces or any other special character except underscore.
  Example:
  ```
  $my-name = "John"; // Invalid

  $my name = "John"; // Invalid

  $my_name = "John"; // Valid
  ```

Declaration of a Variable

Variable Declaration is done on the basis of the datatype used to define the variable. These variables can be of three different datatypes:

- **Scalar Variables:** It contains a single string or numeric value. It starts with $ symbol.
  Syntax: *$var_name = value;*
  Example:
  ```
  $item = "Hello"

  $item_one = 2
  ```
- **Array Variables:** It contains a randomly ordered set of values. It starts with @ symbol.
  Syntax : *@var_name = (val1, val2, val3, ...);*
  Example:
  ```
  @price_list = (70, 30, 40);

  @name_list = ("Apple", "Banana", "Guava");
  ```
- **Hash Variables:** It contains (key, value) pair efficiently accessed per key. It starts with % symbol.
  Syntax : *@var_name = (val1 = key1, val2 = key2, val3 = key3, ...);*
  Example:
  ```
  %item_pairs = ("Apple" =>2, "Banana'=>3);

  %pair_random = ("Hi" =>8, "Bye"=>9);
  ```

<u>Modification of a Variable</u>

Perl allows to modify its variable values anytime after the variable declaration is done. There are various ways for the modification of a variable:

- A scalar variable can be modified simply by redefining its value.
  Example:
  ```
  $name = "John";

  # This can be modified by simply

  # redeclaring the variable $name.

  $name = "Rahul";
  ```
- An element of an array can be modified by passing the index of that element to the array and defining a new value to it.
  Example:
  ```
  @array = ("A", "B", "C", "D", "E");
  ```

```
# If value of second variable is to
# be modified then it can be done by
@array[2] = "4";
# This will change the array to,
# @array = ("A", "B", "4", "D", "E");
```

- A value in a hash can be modified by using its Key.
  Example:

```
%Hash = ("A", 10, "B", 20, "C", 30)
# This will modify the value
# assigned to Key 'B'
$Hash{"B"} = 46;
%Hash = (“A”, 10, ”B”, 46, ”C”, 30)
```

<u>Variable Interpolation:</u>

- Perl provides various methods to define a String to a variable.
- This can be done with the use of single quotes, double quotes, using q-operator and double-q operator, etc.
- Using single quotes and double quotes for writing strings is same but there exists a slight difference between how they work.
- Strings which are written with the use of single quotes display the content written within it exactly as it is.
  Example:

```
$name = "John"
print 'Hi $name\nHow are you?'
```

The above code will print:

```
Hi $name\nHow are you?
```

Whereas strings written within double quotes replace the variables with their value and then displays the string. It even replaces the escape sequences by their real use.

Example:

```
$name = "John"
print "Hi $name\nHow are you?"
```

The above code will print:

```
Hi John

How are you?
```

Example Code:

```
#!/usr/bin/perl
use Data::Dumper;
# Scalar Variable
$name = "GeeksForGeeks";
# Array Variable
@array = ("G", "E", "E", "K", "S");
# Hash Variable
%Hash = ('Welcome', 10, 'to', 20, 'Geeks', 40);

# Variable Modification
@array[2] = "F";
print "Modified Array is @array\n";
# Interpolation of a Variable
# Using Single Quote
print 'Name is $name\n';
# Using Double Quotes
print "\nName is $name";
# Printing hash contents
print Dumper(\%Hash);
```

Output:

```
Modified Array is G E F K S

Name is $name\n

Name is GeeksForGeeks$VAR1 = {

          'to' => 20,

          'Welcome' => 10,

          'Geeks' => 40

        }
```

STATEMENT AND CONTROL STRUCTURES:-

PERL DEFINTION:

- Perl is a programming language.

- It's interpreted.
- It's designed for text processing.
- It's general-purpose, high-level-language.

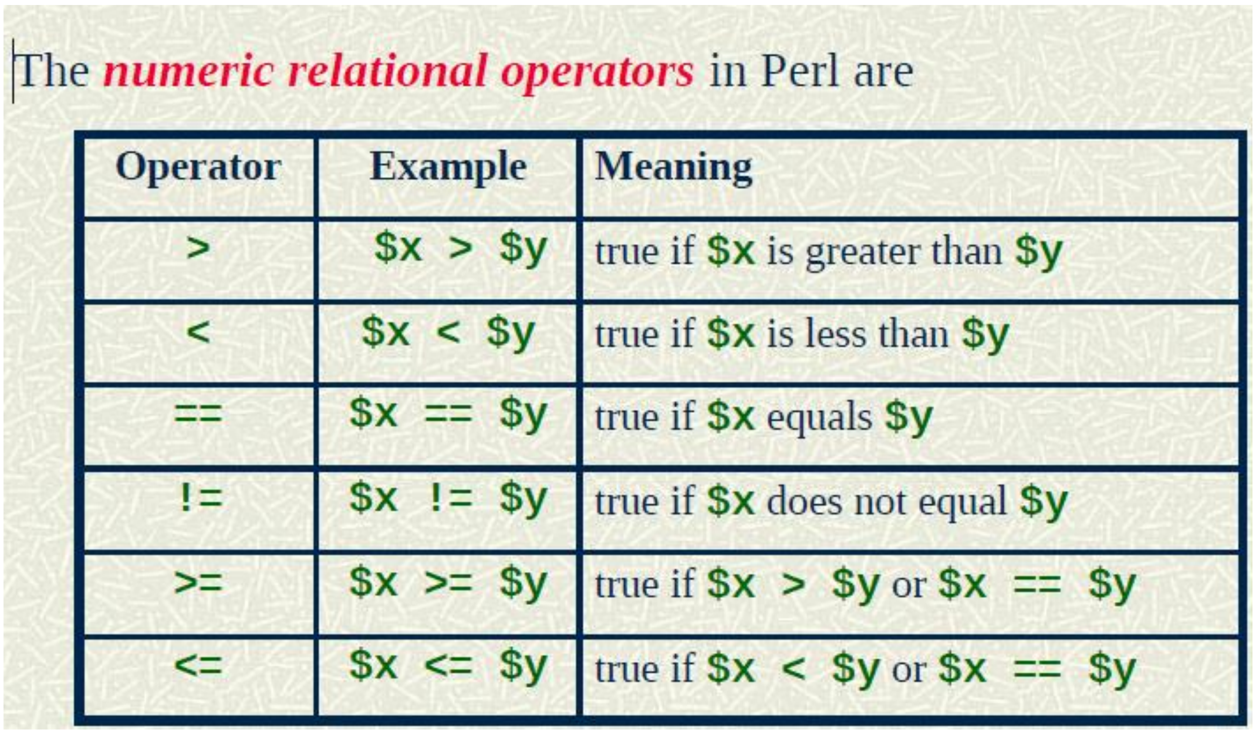
A program is a collection of statements. After the program executes one statement, it "moves" to the next statement and executes that one. If you imagine that a statement is a stepping stone, then you can also think of the *execution flow* of the program as a sequence of "stones".

Operators:

➢ Although any expression can be used as a condition, is usually one that is built from *relational operators* and/or *logical operators.*

Relational operators:

- Relational operators are operators that compare two expressions.
- In math we use operators like >, <, and ≠ to compare numeric expressions.
- There are six numeric relational operators in Perl, which are listed in the next slide.


The *numeric relational operators* in Perl are

Operator	Example	Meaning
>	$x > $y	true if $x is greater than $y
<	$x < $y	true if $x is less than $y
==	$x == $y	true if $x equals $y
!=	$x != $y	true if $x does not equal $y
>=	$x >= $y	true if $x > $y or $x == $y
<=	$x <= $y	true if $x < $y or $x == $y

Logical operators:

The following are logical operators in Perl:

- $a && $b performs logical AND of two variables or expressions. The logical && operator checks if both variables or expressions are true.
- $a || $b performs logical OR of two variables or expressions. The logical || operator checks either a variable or expression is true.
- !$a performs logical NOT of the variable or expression. The logical ! operator inverts the value of the followed variable or expression. In the other words, it converts true to false or false to true.

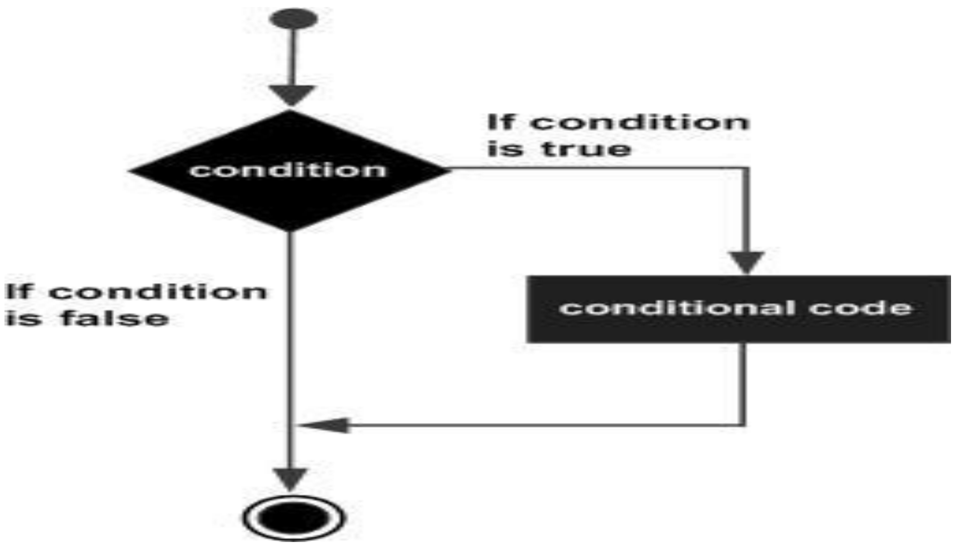
Types of conditional statements:

Sr.No.	Statement & Description
1	if statement An if statement consists of a boolean expression followed by one or more statements.
2	if...else statement An if statement can be followed by an optional else statement.
3	if...elsif...else statement An if statement can be followed by an optional elsif statement and then by an optional else statement.
4	unless statement An unless statement consists of a boolean expression followed by one or more statements.
5	unless...else statement An unless statement can be followed by an optional else statement.
6	unless...elsif..else statement An unless statement can be followed by an optional elsif statement and then by an optional else statement.
7	switch statement With the latest versions of Perl, you can make use of the switch statement. which allows a simple way of comparing a variable value against various conditions.

1.The if statement:-

- If the boolean expression evaluates to true then the block of code inside the if statement will be executed.
- If boolean expression evaluates to false then the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Flow Diagram:



Syntax:

```
If(expression)
{ statement1;
Statement2;...
}
```

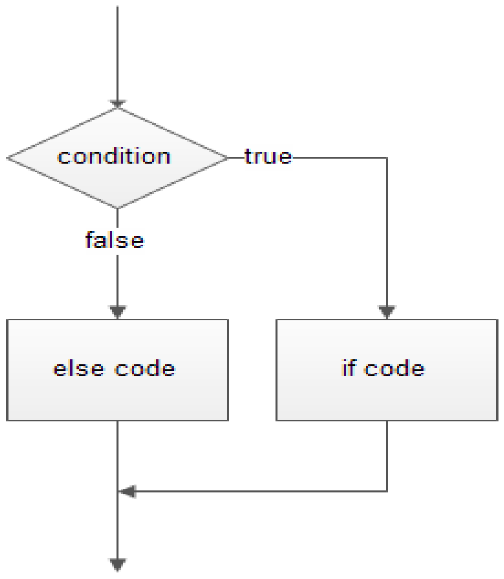
Example:

```
$a = 1;

if($a == 1)

{

  print("Welcome to the Perl if tutorial!\n");}
```

2.If-else statement:

- Perl provides the if else statement that allows you to execute a code block if the expression evaluates to true , otherwise, the code block inside the else branch will execute.

Flow chart:



Syntax:

```
if(expression){

   //if code block;

}else{

   //else code block;

}
```

Example:

```
$a = 1;

$b = 2;

if($a == $b){

 print("a and b are equal\n");

}else{

 print("a and b are not equal\n");

}
```

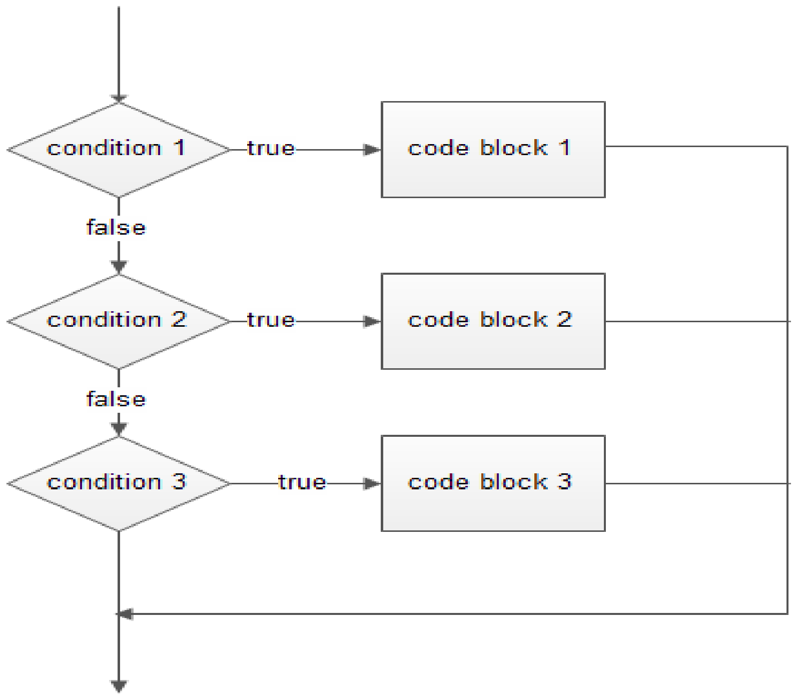
3.if...elsif...else statement:

An if statement can be followed by an optional elsif...else statement, which is very useful to test the various conditions using single if...elsif statement.

When using if , elsif , else statements there are few points to keep in mind.

- An if can have zero or one else's and it must come after any elsif's.
- An if can have zero to many elsif's and they must come before the else.
- Once an elsif succeeds, none of the remaining elsif's or else's will be tested.

<u>Flow chart:</u>



Syntax

```
if(expression){
 ...
}elsif(expression2){
 ...
}elsif(expression3{
  ...
}else{
   ...
}
```

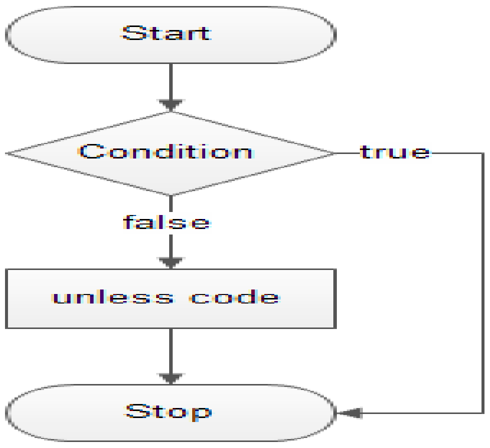
<u>Example</u>:

```
$a = 1;
$b = 2;
if($a == $b){
  print("a and b are equal\n");
}elsif($a > $b){
  print("a is greater than b\n");
}else{
  print("a is less than b\n");
}
```

4.unless statement:

- Perl executes the statement from right to left, if the condition is false , Perl executes the statement that precedes the unless .
- If the condition is true , Perl skips the statement.
- If the condition evaluates to false , Perl executes the code block, otherwise, it skips the code block.

Flow Chart:



Syntax:

```
unless(condition){
  // code block
}
```

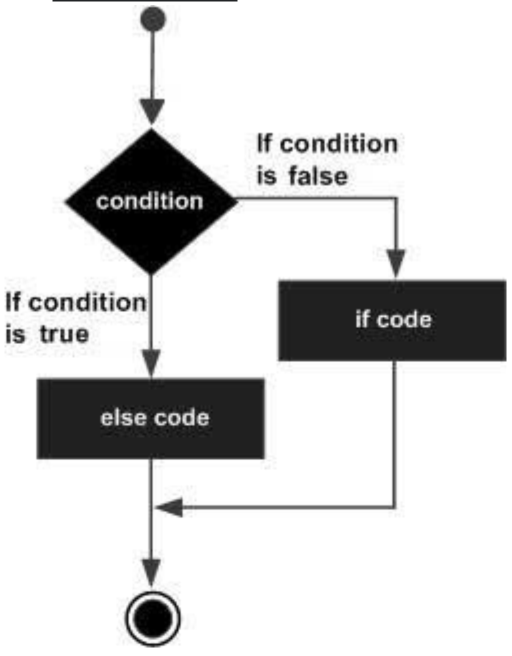
Example:

```
$a = 10;

unless($a <= 0){

  print("a is greater than 0\n")

}
```

5.unless...else statement:

- Similar to unless statement, the unless-else statement in Perl behaves opposite to the if-else statement.
- In unless-else, the statements inside unless gets executed if the condition is false and statements inside else gets executed if the condition is true.

Flow chart:



Syntax:

```
unless(condition){

  // unless code block

}else{

  // else code block

}
```

Example:

```
$a = 10;

unless($a >= 0){

  print("a is less than 0\n");

}else{

  print("a is greater than or equal 0\n");

}
```

6.Unless elseif else statement:

- If you have more than one condition to check with the unless statement, you can use the unless elsif else statement as follows:

Syntax:

```
unless(condition_1){

  // unless code block

}elsif(condition_2){

  // elsif code block

}else{

  // else code block

}
```

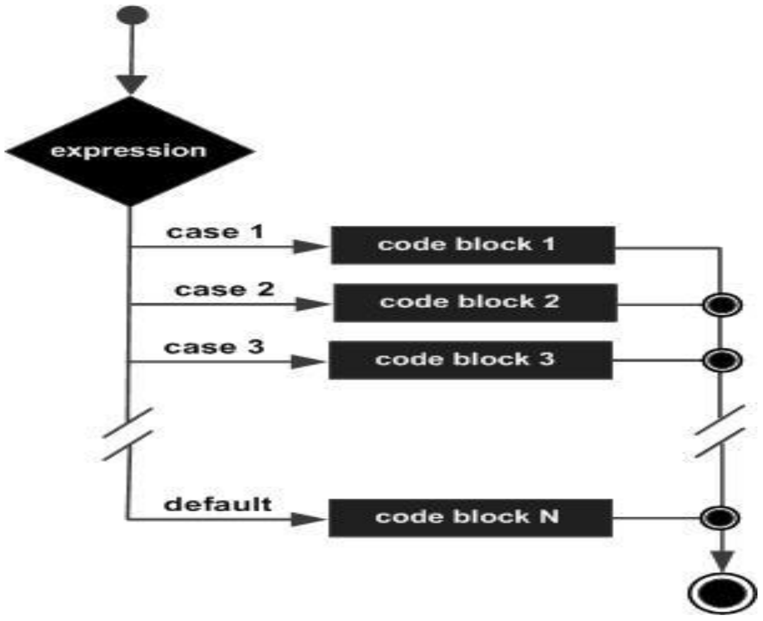
Example:

```
#!/usr/local/bin/perl

$a = 20;

# check the boolean condition using if statement

unless( $a  ==  30 ) {

  # if condition is false then print the following

  printf "a has a value which is not 20\n";

} elsif( $a ==  30 ) {
  # if condition is true then print the following
  printf "a has a value which is 30\n";
} else {
  # if none of the above conditions is met
  printf "a has a value which is $a\n";
}
```

7.Switch statement:

- A switch statement allows a variable to be tested for equality against a list of values.
- Each value is called a case, and the variable being switched on is checked for each switch case.

Flow Diagram:



Syntax:

```
given(expression)
{
when ( first value)
{
statement to be executed;
}
when (second value)
{
statement to be executed;
}
....
...
when (nth value)
{
statement to be executed;
}
default
{
statement to be executed if all the cases are not matched.;
}
}
```

Example:

```
use feature qw(switch say);
print "Enter the Number for the Week \n";
chomp( my $week = <> );
given ($week)
{
when('1')
{
say "Monday";
}
when('2')
{
say "Tuesday";
}
when('3')
{
say "Wednesday";
}
when('4')
{
say "Thursday";
}
when('5')
{
say "Friday";
}
when('6')
{
say "Saturday";
}
when('7')
{
say "Sunday";
}
default
{
say "Please Enter valid Week Number";
}
}
```

Loops

Looping in programming languages is a feature which facilitates the execution of a set of instructions or functions repeatedly while some condition evaluates to true. Loops make the programmers task simpler. Perl provides the different types of loop to handle the condition based situation in the program. The loops in Perl are :

- **for loop**
- **foreach loop**
- **while loop**
- **do…. while loop**
- **until loop**
- **Nested loops**

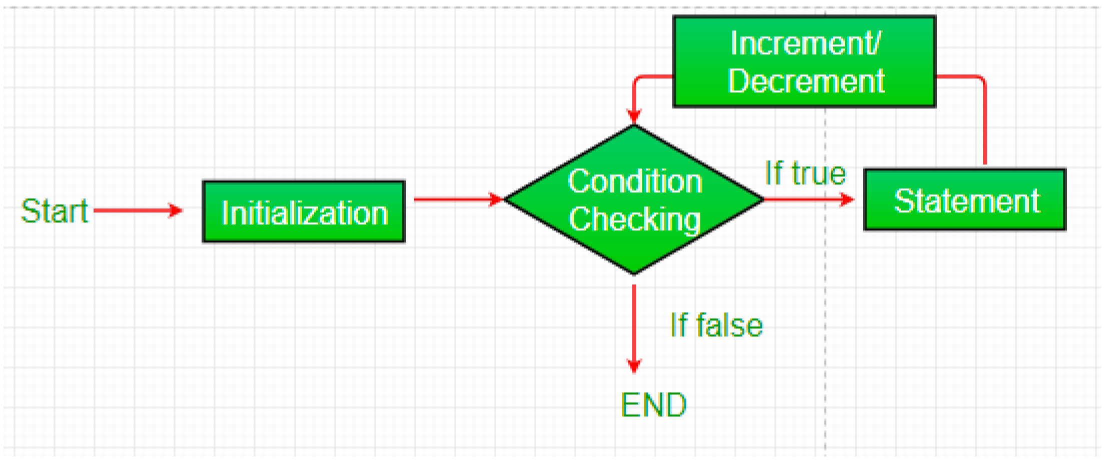
for Loop

"for" loop provides a concise way of writing the loop structure. A for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

Syntax:

```
for (init statement; condition; increment/decrement )

{
    # Code to be Executed
}
```

Flow Chart:



Example :

```
# Perl program to illustrate
# the for loop

# for loop
for ($x =10; $x<=15 ; $x++)
{
    print "$x\n"
}
```

Output:
```
10
11
12
13
14
15
```

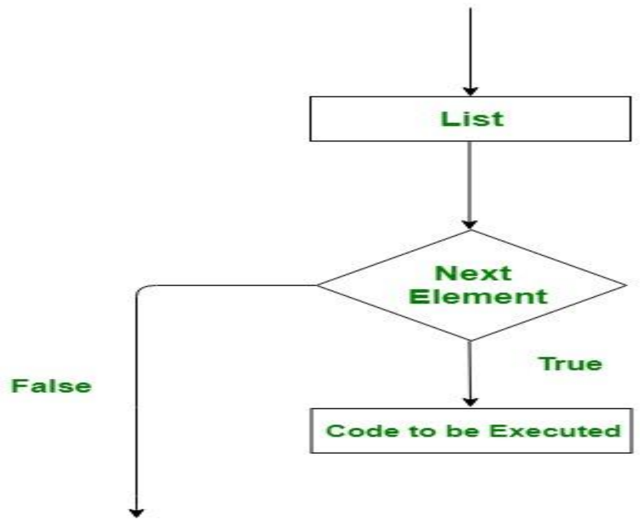
foreach Loop

A foreach loop is used to iterate over a list and the variable holds the value of the elements of the list one at a time. It is majorly used when we have a set of data in a list and we want to iterate over the elements of the list instead of iterating over its range. The process of iteration of each element is done automatically by the loop.

Syntax:

```
foreach variable

{

    # Code to be Executed

}
```

Flow Chart:



Example:

```
# Perl program to illustrate
# the foreach loop

# Array
@data = ('GEEKS', 'FOR', 'GEEKS');
```

```
# foreach loop
foreach $word (@data)
{
    print $word
}
```

Output:

```
GEEKSFORGEEKS
```

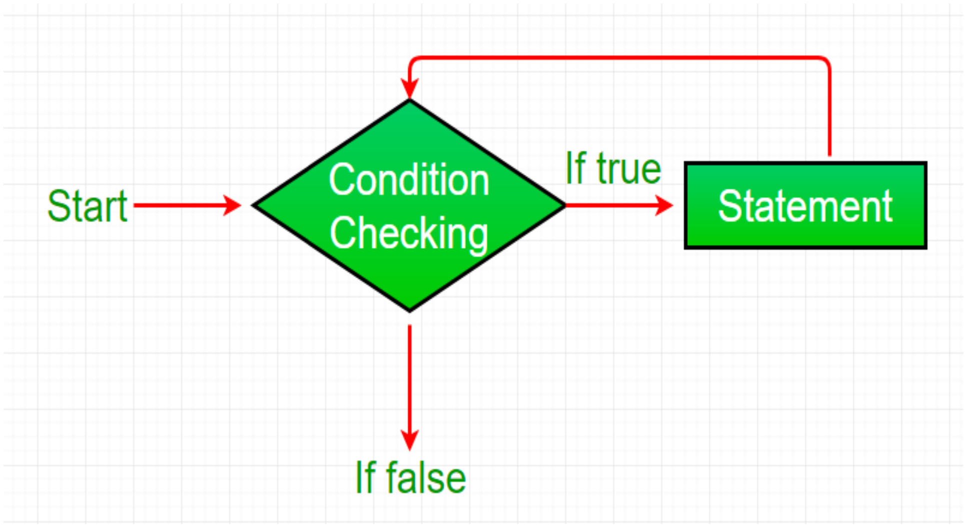
while Loop

- A while loop generally takes an expression in parenthesis.
- If the expression is True then the code within the body of while loop is executed.
- It is also known as a **entry controlled loop** as the condition is checked before executing the loop.

Syntax :

```
while (condition)

{

    # Code to be executed

}
```

Flow Chart:



Example :

```
# Perl program to illustrate
# the while loop

# while loop
$count = 3;
while ($count >= 0)
{
    $count = $count - 1;
    print "GeeksForGeeks\n";
}
```
Output:
```
GeeksForGeeks

GeeksForGeeks

GeeksForGeeks

GeeksForGeeks
```
Infinite While Loop: While loop can execute infinite times which means there is no terminating condition for this loop. In other words, we can say there are some conditions which always remain true, which causes while loop to execute infinite times or we can say it never terminates.

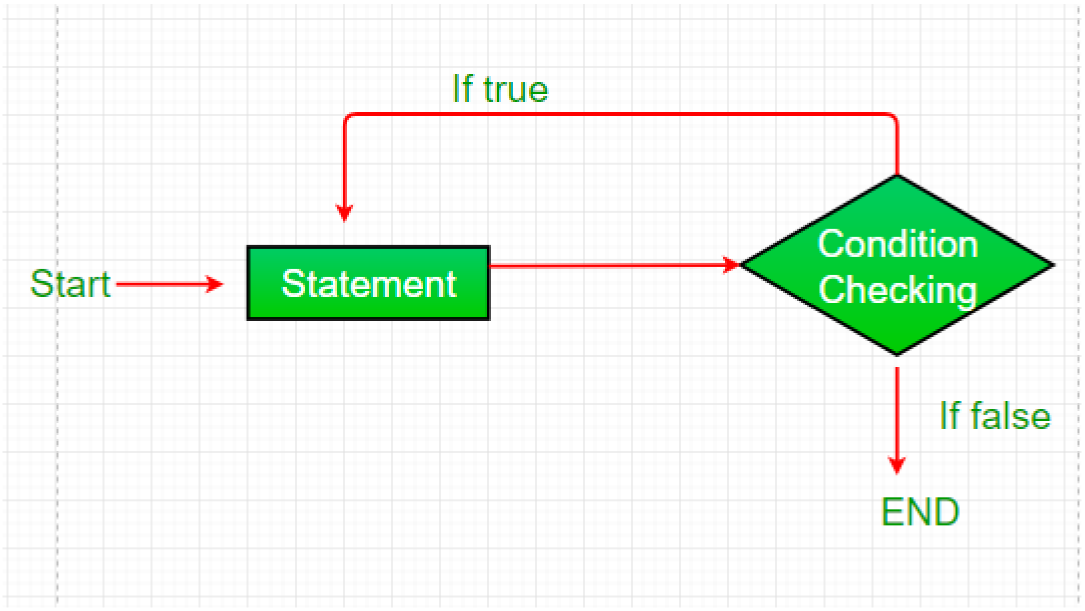
```
# Perl program to illustrate
# the infinite while loop

# infinite while loop
# containing condition 1
# which is always true
while(1)
{
    print "Infinite While Loop\n";
}
```
Output:
```
Infinite While Loop

Infinite While Loop

Infinite While Loop

Infinite While Loop

.

.

.

.
```
do…. while loop

- A do..while loop is almost same as a while loop.
- The condition is checked after the first execution.
- It is also known as **exit controlled loop** as the condition is checked after executing the loop.

Syntax:

```
do {

    # statments to be Executed

} while(condition);
```

Flow Chart:



Example :

```
# Perl program to illustrate
# do..while Loop

$a = 10;

# do..While loop
do {

    print "$a ";
    $a = $a - 1;
} while ($a > 0);
```

Output:

```
10 9 8 7 6 5 4 3 2 1
```

until loop

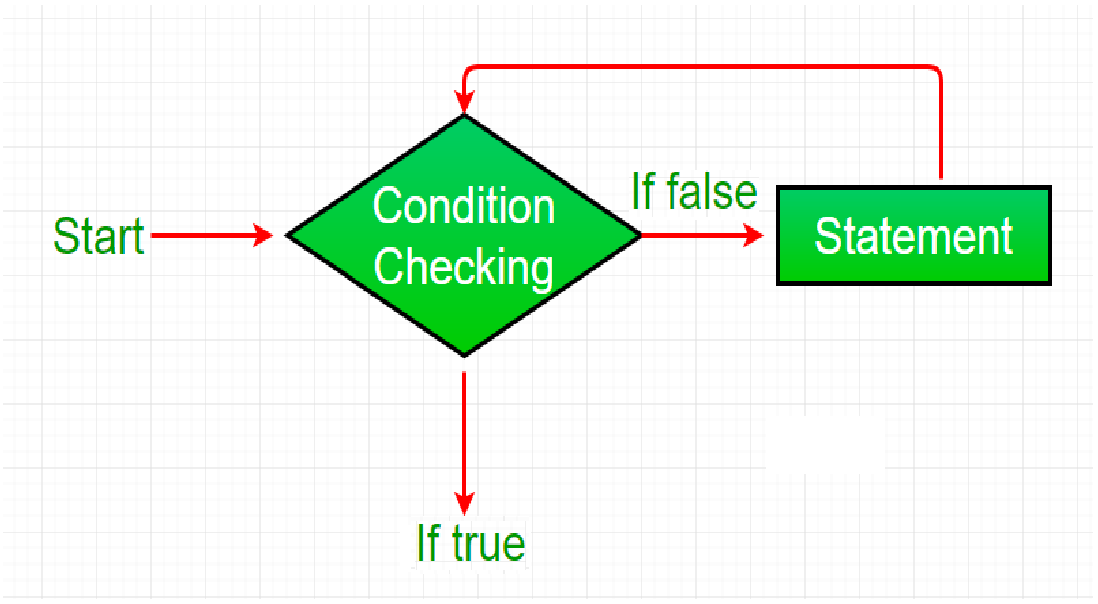
- **until loop** is the opposite of while loop. It takes a condition in the parenthesis and it only runs until the condition is false.
- Basically, it repeats an instruction or set of instruction until the condition is FALSE.

- It is also entry controller loop i.e. first the condition is checked then set of instructions inside a block is executed.

Syntax:

```
until (condition)
{
   # Statements to be executed
}
```

Flow Chart:



Example :

```
# Perl program to illustrate until Loop

$a = 10;

# until loop
until ($a < 1)
{
    print "$a ";
    $a = $a - 1;
}
```

Output:

```
10 9 8 7 6 5 4 3 2 1
```

Nested Loops

A nested loop is a loop inside a loop. Nested loops are also supported by Perl Programming.

Syntax for while nested loop in Perl:

- **Nested while loop**

```
while (condition)
{
   while (condition)
    {
        # Code to be Executed
    }
}
```

Example :

```
# Perl program to illustrate
# nested while Loop

$a=5;
#outer while loop
while($a>0)
{
    $b=1;
    #inner while loop
    while($b<=$a)
    {
        print "*";
        $b=$b+1;
    }
    $a=$a-1;
    print "\n”;
}
```

Output:

```
*****
****
***
**
*
```

PERL SUBROUTINES

Definition

- A Perl function or subroutine is a group of statements that together perform a specific task.
- In every programming language user want to reuse the code.
- So the user puts the section of code in function or subroutine so that there will be no need to write code again and again.
- In Perl, the terms *function, subroutine, and method are the same* but in some programming languages, these are considered different.
- The word subroutines is used most in Perl programming because it is created using keyword ***sub***.
- Whenever there is a call to the function, Perl stop executing all its program and jumps to the function to execute it and then returns back to the section of code that it was running earlier.

Define and Call a Subroutine

The general form of a subroutine definition in Perl programming language is as follows −

```
sub subroutine_name {
   body of the subroutine
}
```

The typical way of calling that Perl subroutine is as follows −

```
subroutine_name( list of arguments );
```

In versions of Perl before 5.0, the syntax for calling subroutines was slightly different as shown below. This still works in the newest versions of Perl.

```
&subroutine_name( list of arguments );
```

Let's have a look into the following example, which defines a simple function and then call it. Because Perl compiles your program before executing it, it doesn't matter where you declare your subroutine.

```
#!/usr/bin/perl

# Function definition
sub Hello {
   print "Hello, World!\n";
}

# Function call
Hello();
```

When above program is executed, it produces the following result −

```
Hello, World!
```

Passing Arguments to a Subroutine

You can pass various arguments to a subroutine like you do in any other programming language and they can be acessed inside the function using the special array @_. Thus the first argument to the function is in $_[0], the second is in $_[1], and so on.

You can pass arrays and hashes as arguments like any scalar but passing more than one array or hash normally causes them to lose their separate identities. So we will use references to pass any array or hash.

Let's try the following example, which takes a list of numbers and then prints their average −

```
#!/usr/bin/perl

# Function definition
sub rectangle
{
     $area=@_[0]*@_[1];
     $perimeter=2*(@_[0]+@_[1]);
     print "Area of the rectangle is: $area
sq.units\n";
     print "Perimeter of the rectangle is: $perimeter
units\n";
}
# Function call
```

```
rectangle(10, 20);
```

Output

Area of the rectangle is: 200 sq.units

Perimeter of the rectangle is: 60 units

Passing Lists to Subroutines

Because the @_ variable is an array, it can be used to supply lists to a subroutine. However, because of the way in which Perl accepts and parses lists and arrays, it can be difficult to extract the individual elements from @_. If you have to pass a list along with other scalar arguments, then make list as the last argument as shown below −

```
#!/usr/bin/perl

# Function definition
sub random
{
     @hobbies=@_;
     print "Hobbies: @hobbies\n";
}
@favs=("Playing Guitar","Playing Chess","Reading
Books");
# Function call with list parameter
random(@favs);
```

When above program is executed, it produces the following result –

```
Hobbies: Playing Guitar Playing Chess Reading Books
```

Passing Hashes to Subroutines

When you supply a hash to a subroutine or operator that accepts a list, then hash is automatically translated into a list of key/value pairs. For example −

```
#!/usr/bin/perl
```

```
# Function definition
sub PrintHash {
   my (%hash) = @_;

   foreach my $key ( keys %hash ) {
      my $value = $hash{$key};
      print "$key : $value\n";
   }
}
%hash = ('name' => 'Tom', 'age' => 19);

# Function call with hash parameter
PrintHash(%hash);
```

When above program is executed, it produces the following result −

```
name : Tom
age : 19
```

Returning Value from a Subroutine

You can return a value from subroutine like you do in any other programming language.

You can return arrays and hashes from the subroutine like any scalar but returning more than one array or hash normally causes them to lose their separate identities. So we will use references to return any array or hash from a function.

Let's try the following example, which takes a list of numbers and then returns their average −

```
#!/usr/bin/perl

# Function definition
sub circle
{
     $radius=@_[0];
     return 3.14*$radius*$radius;
}
# Function call
$area=circle(10);
print "The area of the circle is $area sq. units\n";
```

When above program is executed, it produces the following result −

```
The area of the circle is 314 sq.units
```

Private Variables in a Subroutine

By default, all variables in Perl are global variables, which means they can be accessed from anywhere in the program. But you can create **private** variables called **lexical variables** at any time with the **my** operator.

The **my** operator confines a variable to a particular region of code in which it can be used and accessed. Outside that region, this variable cannot be used or accessed. This region is called its scope..

Following is an example showing you how to define a single or multiple private variables using **my** operator −

```
#!/usr/bin/perl

# Global variable
$var=500;

# Function definition
sub test {
   # Private variable
   my $var=50;
   print "Value of variable inside the subroutine
$var\n";
}
# Function call
test();
print "Value of variable outside the subroutine
$var\n";
```

When above program is executed, it produces the following result −

```
Value of variable inside the subroutine: 50
Value of variable outside the subroutine: 500
```

Packages and modules

What are Packages?

The package statement switches the current naming context to a specified namespace (symbol table). Thus −

- A package is a collection of code which lives in its own namespace.
- A namespace is a named collection of unique variable names (also called a symbol table).
- Namespaces prevent variable name collisions between packages.
- You can explicitly refer to variables within a package using the :: package qualifier.

Following is an example having main and Foo packages in a file. Here special variable __PACKAGE__ has been used to print the package name.

```
#!/usr/bin/perl

# This is main package
$i = 1;
print "Package name : " , __PACKAGE__ , " $i\n";

package Foo;
# This is Foo package
$i = 10;
print "Package name : " , __PACKAGE__ , " $i\n";

package main;
# This is again main package
$i = 100;
print "Package name : " , __PACKAGE__ , " $i\n";
print "Package name : " , __PACKAGE__ ,  " $Foo::i\n";

1;
```

OUTPUT:

```
Package name : main 1
Package name : Foo 10
Package name : main 100
Package name : main 10
```

BEGIN and END Blocks

You may define any number of code blocks named BEGIN and END, which act as constructors and destructors respectively.

```
BEGIN { ... }
END { ... }
BEGIN { ... }
END { ... }
```

- Every **BEGIN** block is executed after the perl script is loaded and compiled but before any other statement is executed.
- Every END block is executed just before the perl interpreter exits.
- The BEGIN and END blocks are particularly useful when creating Perl modules.

Following example shows its usage −

```
#!/usr/bin/perl

package Foo;
print "Begin and Block Demo\n";

BEGIN {
   print "This is BEGIN Block\n"
}

END {
   print "This is END Block\n"
}

1;
```

OUTPUT:

```
This is BEGIN Block
Begin and Block Demo
This is END Block
```

What are Perl Modules?

- A Perl module is a reusable collection of related variables and subroutines that perform a set of programming tasks.
- There are a lot of Perl modules available on the Comprehensive Perl Archive Network (CPAN).

Perl module example

Suppose you are working on a project that requires logging functionality. You have done a research on CPAN but didn't find any module that meets your requirements. You decided to create your own Perl module.

The first thing comes to your mind is the module name e.g., *FileLogger*. The main functionality of the *FileLogger* module is to:

- Open the log file.
- Write log messages to the log file based on log level.
- Close the log file.

To create *FileLogger* module, you need to do the following steps:

1. First, create your own module name, in this case, you call it *FileLogger.*
2. Second, create a file named *modulename.pm*. In this case, you need to create a new file named *FileLogger.pm*. pm stands for Perl module.
3. Third, make the *FileLogger* module a package by using the

   syntax: *package FileLogger;* at the top of the *FileLogger.pm* file.

4. Fourth, write the code for subroutines and variables, and put the code into the *FileLogger.pm* file.
5. Fifth, put the last statement in the *FileLogger.pm* file: 1; to make the file returns *true*.

Let's get started to develop the FileLogger module.

1. First, create a new file named *FileLogger.pm*
2. Second, put the package name at the top of the *FileLogger.pm*

```
package FileLogger;
```

3. Third, put the global variable $LEVEL so that any subroutine can access it.

```
my $LEVEL = 1;
```

4. Fourth, develop subroutines to handle logging functionality. We need a subroutine to open the log file for writing log messages.

```
sub open{
  my $logfile = shift;
  # open log file for appending
  open(LFH, '>>', $logfile) or die "cannot open the log file $logfile: $!";
  # write time:
```

```
  print LFH "Time: ", scalar(localtime), "\n";
}
```

We need another subroutine to append log messages to the log file. We only log messages if the input log level is lower than the current module's log level. We use print() function to write log messages into the log file.

```
sub log{
  my($log_level,$log_msg) = @_;

  if($log_level <= $LEVEL){
    print LFH "$log_msg\n";
  }
}
```

We need a subroutine to close the log filehandle:

```
sub close{
  close LFH;
}
```

We could allow other programs to change the log level $LEVEL from the outside of the module. We can do this by creating a new subroutine set_level(). Inside the subroutine, we'll check if the passed log level is a number using a regular expression before setting the module log level.

```
sub set_level{
  my $log_level = shift;

  if($log_level =~ /^\d+$/){
    $LEVEL = $log_level;
  }
}
```

Fifth, at the end of the FileLogger.pm file, we put the statement: 1;

Example:

- A Perl module file called **Foo.pm** might contain statements like this.

```
#!/usr/bin/perl

package Foo;
sub bar {
   print "Hello $_[0]\n"
}

sub blat {
   print "World $_[0]\n"
}
1;
```

Few important points about Perl modules:

- The functions require and use will load a module.
- Both use the list of search paths in @INC to find the module.
- Both functions require and use call the eval function to process the code.
- The 1; at the bottom causes eval to evaluate to TRUE (and thus not fail).

The Require Function:

A module can be loaded by calling the require function as follows −

```
#!/usr/bin/perl

require Foo;

Foo::bar( "a" );
Foo::blat( "b" );
```

You must have noticed that the subroutine names must be fully qualified to call them. It would be nice to enable the subroutine bar and blat to be imported into our own namespace so we wouldn't have to use the Foo:: qualifier.

The Use Function:

A module can be loaded by calling the use function.

```
#!/usr/bin/perl

use Foo;

bar( "a" );
blat( "b" );
```

Notice that we didn't have to fully qualify the package's function names. The use function will export a list of symbols from a module given a few added statements inside a module.

```
require Exporter;
@ISA = qw(Exporter);
```

Then, provide a list of symbols (scalars, lists, hashes, subroutines, etc) by filling the list variable named @EXPORT: For Example −

```
package Module;
```

```
require Exporter;
@ISA = qw(Exporter);
@EXPORT = qw(bar blat);

sub bar { print "Hello $_[0]\n" }
sub blat { print "World $_[0]\n" }
sub splat { print "Not $_[0]\n" }  # Not exported!
1;
```

Modules in different directory

If a module is present in some sub-directory, then also we use :: to tell the path of the module. For example, if a module 'b' is present in a sub-directory 'a', then we use *use* a::b; to load the module b. Let's see an example to understand this:

```
#p.pm is defined in directory ab which

#is in parent directory

use strict;

use warnings;

#using package p

use ab::p;

#Function Hello of p

p::Hello();
```

Working with files:

To read or write files in perl you need to open a file handle. files handles in perl are yet another kind of variable. they are act as convenient reference handle, if you will between your program and the operating system about a particular file.

Perl is an outstanding language for reading from and writing to files on disk or else where. Begin to incorporate files into your perl programs by learning how to open, read, write and test files.

To learn:-

- To open and close files
- To write data to files
- To read data from files
- To write perl defensively

Opening files:-

To read or write files in perl you need to open a file handle. File handles in perl are yet another kind of variable. They act as conversion reference(handles, if you will) between your program and the operating system about a particular file. They contain information about how the files was opened and how far along you are in reading (or writing) the file they also contain user-definable attributes about how the file is to be read or write

Any time you need to access a file on your disk. You need to create a new file handled and prepare it by operating the file handled. You open file handled. Not surprisingly, with the open function. They syntax of open function is as follows:

```
open (file handled, path name)
```

The open function takes a file handles as its first argument and a path name as the second argument. The path name indicates which file you want to open.so if you don't specify a path name such as c:\windows\system\.open will try to open the file in the current directory.

Closing Files:-

To close a file handled, and therefore disassociate the file handled from the corresponding file you use the close function. This flushes the file handle's buffers and closes the system's file descriptor.

```
CLOSE FILE HANDLE

CLOSE
```

If no FILEHANDLED is specified, then it closes the currently selected file handled.it returns true only if it could success fully flush the buffers and close the file.

Path:-

Until now, you've opened only files with simple name like novel.txt that did not include a path. When you try to open a file name that doesn't specify a directory name. perl assumes the file is in the current directory. To open a file that's in another directory. You must use a path name. the path name describes the path that perl must take to find the file on your system.

Reading:-

You can read from perl's file handling in a couple of different ways. The most common method is to use the file input operator also called the angle operator(<>). To read a file handle, simply put the file handle name inside the angle operator and assign the value to a variable.

```
Open(my file ,"my file")||die "can't open my file:
$line=<MYFILE>;
#Reading the file handle
```

The angle operator in a scalar context reads one line of input from the file. When called after the entire file has been read, the angle operator returns the value undef.

Writing:-

To write data to a file, you must first have a file-handle open for writing. Up till now, all open statements you have seen have opened the file handle for reading only. The syntax for opening a file for writing is almost identical to that for reading.

```
Open(file handle,">path name")
Open(file handle,">>path name")
```

The signifies to perl that the file specified at the path name should be over written with new data, that any existing data should be discarded and the file handle is open for writing.

Perl Denfensive programming:-

- Use strict
- #!/usr/bin/perl-w
- Check all syscall returns values, printing $:
- Watch for external program failures in$?
- Check$ @ after eval" "or sillee
- Parameter asserts
- #!/usr/bin/perl-T
- Always have an else after a chain of elseif
- Put commands at the end of lists to so your program won't break if some one insterts another item at the end of the list.

Data manipulation in perl:

- The simplest uses of perl involve reading or more text files a lines at a time,changeing The line in fashion, and sending the result to an output file.
- Usually the perl program is stored in a file but simple, "one liner" applications can be written the commands line after a-e flag as in sed and awk.
- By andy lester, andy manages programmers for follett library resources

  - -e-command
  - the diamond operator
  - n and p ; automatic looping power houses
  - l: line -ending handling
  - -i: edit in place
  - -o[octal]:specify input-record separator

-e command:

- The most useful way to use the command-line options is writing perl one-liners right in the shell.
- e e-option is the basis for most command-line programs.
- It accepts the value of the parameter as the source text for a program.
- Since this is a single statement in a block, you can omit the semicolon.
- When e-option is uses, perl no longer looks for a program name on the command line.

```
$perl-e print “hello,world !  \n “
```

Escaping shell characters:

- When you re creating command-line programs it’s important to pay attention.
- L’ve quote with single quotes not double quotes for two reasons.
- First, I want to be able to use double quotes doen’t nest in the shell.
- Second, I have to prevent shell interpolation,and single quote make it easy.
- You can escape the shell variables with a backslash.

The diamond operator:

- Perl’s diamond operator, <>,has a great deal of magic built into it making operations on multiple files easy.so that your program can operator on three files at once?use the diamond operator instead.
- Perl keeps track on which file your on and opens and clases file handle as appropriate with the diamond operator.
- Perl keeps the name of currently open file in $ARGV. The $line counter does not reset at the beginning of each file.The diamond operator figures programming in much perl command-line magic so , it you to get comfortable with it.

-n and -p: Automatic looping power houses

- The -n and -p options are the real work horse options. They derive from the AWK metaphor of "Do something to every line in the file" and work closely with the diamond operator.

-l: Line Ending Handling

- when you're working with lines in a file.   You will find you are doing lots chomping and print in the simplest sense adding -l when you are using -n or -p automatically does a champ on the input record and adds "\n" after everything you print. It makes command-line one-line much easier.
- This example only shows the first 40 characters of each line in the input of whether or not the line is longer than 40 characters not counting the line ending.
- For command-line programmers -l is a good send because it means you can use print $800 instead of print $800 "\n" to get the result what you want.

-i: Edit in Place

- All the options you've learned so far are great for writing filters where a number of files or standard input get used out to standard output unfortunately.
- perl comes to the resource again with the -i option. Adding -i tells perl to edit your files in place.

-o[octal]: specify input record seperate

- When working on the command line you want to specify your input record seperator.
- This is possible with e BEGIN {$ / : ...}, its easier with the -o option.
- Two special values for the -o option and are -oo for paragraph mode equivalent to $/="" and -o>>> to entire files equivalent to $/=undef.

UNIT -V

Python Syntax and Style – Python Objects – Numbers – **Sequences** : Strings –Lists and Tuples – Dictionaries – Conditionals and Loops – Files – Input and Output – Errors and Exceptions – Functions – Modules – Classes and OOP – Execution Environment.

Python Introduction

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).

Python has a simple syntax similar to the English language.

Python has syntax that allows developers to write programs with fewer lines than some other programming languages.

Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

Python can be treated in a procedural way, an object-orientated way or a functional way.

Python Syntax compared to other programming languages

Python was designed for readability, and has some similarities to the English language with influence from mathematics.

Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.

Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Python Syntax and Style

Execute Python Syntax

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

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

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

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

Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

Example

```
if 5 > 2:
  print("Five is greater than two!")
```

Python will give you an error if you skip the indentation:

Example

Syntax Error:

```
if 5 > 2:
print("Five is greater than two!")
```

The number of spaces is up to you as a programmer, but it has to be at least one.

Example

```
if 5 > 2:
 print("Five is greater than two!")
if 5 > 2:
        print("Five is greater than two!")
```

You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

Example

Syntax Error:

```
if 5 > 2:
 print("Five is greater than two!")
        print("Five is greater than two!")
```

Python Variables

In Python, variables are created when you assign a value to it:

Example

Variables in Python:

```python
x = 5
y = "Hello, World!"
```

Python has no command for declaring a variable.

Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

Example

Comments in Python:

```python
#This is a comment.
print("Hello, World!")
```

Python Comments

Comments can be used to explain Python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

Creating a Comment

Comments starts with a #, and Python will ignore them:

Example

```
#This is a comment
print("Hello, World!")
```

Comments can be placed at the end of a line, and Python will ignore the rest of the line:

Example

```
print("Hello, World!") #This is a comment
```

Comments does not have to be text to explain the code, it can also be used to prevent Python from executing code:

Example

```
#print("Hello, World!")
print("Cheers, Mate!")
```

Multi Line Comments

Python does not really have a syntax for multi line comments.

To add a multiline comment you could insert a # for each line:

Example

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

Example

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

As long as the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multiline comment.

Python Numbers

There are three numeric types in Python:

- `int`
- `float`
- `complex`

Variables of numeric types are created when you assign a value to them:

Example

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

To verify the type of any object in Python, use the `type()` function:

Example

```
print(type(x))
print(type(y))
print(type(z))
```

Int

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

Example

Integers:

```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

Float

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

Example

Floats:

```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

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

Example

Floats:

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

Complex

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

Example

Complex:

```
x = 3+5j
y = 5j
z = -5j
print(type(x))
print(type(y))
print(type(z))
```

Type Conversion

You can convert from one type to another with the `int()`, `float()`, and `complex()` methods:

Example

Convert from one type to another:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```

Note: You cannot convert complex numbers into another number type.

Random Number

Python does not have a `random()` function to make a random number, but Python has a built-in module called `random` that can be used to make random numbers:

Example

Import the random module, and display a random number between 1 and 9:

```
import random

print(random.randrange(1, 10))
```

Python Strings

String Literals

String literals in python are surrounded by either single quotation marks, or double quotation marks.

`'hello'` is the same as `"hello"`.

You can display a string literal with the `print()` function:

Example

```
print("Hello")
print('Hello')
```

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

```
a = "Hello"
print(a)
```

Multiline Strings

You can assign a multiline string to a variable by using three quotes:

Example

You can use three double quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

Or three single quotes:

Example

```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

Note: in the result, the line breaks are inserted at the same position as in the code.

Strings are Arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

Example

Get the character at position 1 (remember that the first character has the position 0):

```python
a = "Hello, World!"
print(a[1])
```

Slicing

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

Example

Get the characters from position 2 to position 5 (not included):

```python
b = "Hello, World!"
print(b[2:5])
```

Negative Indexing

Use negative indexes to start the slice from the end of the string:

Example

Get the characters from position 5 to position 1 (not included), starting the count from the end of the string:

```python
b = "Hello, World!"
print(b[-5:-2])
```

String Length

To get the length of a string, use the `len()` function.

Example

The `len()` function returns the length of a string:

```python
a = "Hello, World!"
print(len(a))
```

String Methods

Python has a set of built-in methods that you can use on strings.

Example

The `strip()` method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

Example

The `lower()` method returns the string in lower case:

```
a = "HELLO, World!"
print(a.lower())
```

Example

The `upper()` method returns the string in upper case:

```
a = "Hello, World!"
print(a.upper())
```

Example

The `replace()` method replaces a string with another string:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

Example

The `split()` method splits the string into substrings if it finds instances of the separator:

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

Check String

To check if a certain phrase or character is present in a string, we can use the keywords `in` or `not in`.

Example

Check if the phrase "ain" is present in the following text:

```
txt = "The rain in Spain stays mainly in the plain"
x = "ain" in txt
print(x)
$returns true if it is available in the string
```

Example

Check if the phrase "ain" is NOT present in the following text:

```
txt = "The rain in Spain stays mainly in the plain"
x = "ain" not in txt
print(x)
```

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

Example

Merge variable `a` with variable `b` into variable `c`:

```
a = "Hello"
b = "World"
c = a + b
print(c)
```

Example

To add a space between them, add a `" "`:

```
a = "Hello"
b = "World"
c = a + " " + b
print(c)
```

String Format

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

Example

```
age = 36
txt = "My name is John, I am " + age

print(txt)
```

But we can combine strings and numbers by using the `format()` method!

The `format()` method takes the passed arguments, formats them, and places them in the string where the placeholders `{}` are:

Example

Use the `format()` method to insert numbers into strings:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

You can use index numbers `{0}` to be sure the arguments are placed in the correct placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item
{1}."
print(myorder.format(quantity, itemno, price))
```

Escape Character

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash `\` followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

Example

You will get an error if you use double quotes inside a string that is surrounded by double quotes:

```
txt = "We are the so-called "Vikings" from the north."
```

To fix this problem, use the escape character \":

Example

The escape character allows you to use double quotes when you normally would not be allowed:

```
txt = "We are the so-called \"Vikings\" from the north."
```

Other escape characters used in Python:

Code	Result	Try it
\'	Single Quote	Try it »
\\	Backslash	
\n	New Line	Try it »
\r	Carriage Return	
\t	Tab	Try it »
\b	Backspace	

\f	Form Feed	
\ooo	Octal value	
\xhh	Hex value	Try it »

String Methods

Python has a set of built-in methods that you can use on strings.

Note: All string methods returns new values. They do not change the original string.

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string

count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found

isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable

isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations

partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string

split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case

zfill()	Fills the string with a specified number of 0 values at the beginning

Example:

```
txt = "hello, and welcome to my world."

x = txt.capitalize()

print (x)
```

Python Lists

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

List

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

Example

Create a List:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

Access Items

You access the list items by referring to the index number:

Example

Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

Try it Yourself »

Negative Indexing

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

Example

Return the third, fourth, and fifth item:

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon", "man
go"]
print(thislist[2:5])
```

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

Example

This example returns the items from index -4 (included) to index -1 (excluded)

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon", "man
go"]
print(thislist[-4:-1])
```

Change Item Value

To change the value of a specific item, refer to the index number:

Example

Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Loop Through a List

You can loop through the list items by using a `for` loop:

Example

Print all items in the list, one by one:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

You will learn more about `for` loops in our Python For Loops Chapter.

Check if Item Exists

To determine if a specified item is present in a list use the `in` keyword:

Example

Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

List Length

To determine how many items a list has, use the `len()` function:

Example

Print the number of items in the list:

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

Add Items

To add an item to the end of the list, use the `append()` method:

Example

Using the `append()` method to append an item:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

To add an item at the specified index, use the `insert()` method:

Example

Insert an item as the second position:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

Remove Item

There are several methods to remove items from a list:

Example

The `remove()` method removes the specified item:

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

Example

The `pop()` method removes the specified index, (or the last item if index is not specified):

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

Example

The `del` keyword removes the specified index:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

Example

The `del` keyword can also delete the list completely:

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

Example

The `clear()` method empties the list:

```python
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

Copy a List

You cannot copy a list simply by typing `list2 = list1`, because: `list2` will only be a *reference* to `list1`, and changes made in `list1` will automatically also be made in `list2`.

There are ways to make a copy, one way is to use the built-in List method `copy()`.

Example

Make a copy of a list with the `copy()` method:

```python
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

Another way to make a copy is to use the built-in method `list()`.

Example

Make a copy of a list with the `list()` method:

```python
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

Example

Join two list:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

Another way to join two lists are by appending all the items from list2 into list1, one by one:

Example

Append list2 into list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

Or you can use the `extend()` method, which purpose is to add elements from one list to another list:

Example

Use the `extend()` method to add list2 at the end of list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

The list() Constructor

It is also possible to use the `list()` constructor to make a new list.

Example

Using the `list()` constructor to make a List:

```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```

List Methods

Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list

count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Python Tuples

A tuple is a collection which is ordered and **unchangeable**. In Python tuples are written with round brackets.

Example

Create a Tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

Example

Print the second item in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Negative Indexing

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new tuple with the specified items.

Example

Return the third, fourth, and fifth item:

```
thistuple =
("apple", "banana", "cherry", "orange", "kiwi", "melon", "man
go")
print(thistuple[2:5])
```

Try it Yourself »

Note: The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the tuple:

Example

This example returns the items from index -4 (included) to index -1 (excluded)

```
thistuple =
("apple", "banana", "cherry", "orange", "kiwi", "melon", "man
go")
print(thistuple[-4:-1])
```

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

Example

Convert the tuple into a list to be able to change it:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

Loop Through a Tuple

You can loop through the tuple items by using a `for` loop.

Example

Iterate through the items and print the values:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

Check if Item Exists

To determine if a specified item is present in a tuple use the `in` keyword:

Example

Check if "apple" is present in the tuple:

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

Try it Yourself »

Tuple Length

To determine how many items a tuple has, use the `len()` method:

Example

Print the number of items in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

Add Items

Once a tuple is created, you cannot add items to it. Tuples are **unchangeable**.

Example

You cannot add items to a tuple:

```
thistuple = ("apple", "banana", "cherry")
thistuple[3] = "orange" # This will raise an error
print(thistuple)
```

Remove Items

Note: You cannot remove items in a tuple.

Tuples are **unchangeable**, so you cannot remove items from it, but you can delete the tuple completely:

Example

The `del` keyword can delete the tuple completely:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple
no longer exists
```

Join Two Tuples

To join two or more tuples you can use the `+` operator:

Example

Join two tuples:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

The tuple() Constructor

It is also possible to use the `tuple()` constructor to make a tuple.

Example

Using the tuple() method to make a tuple:

```
thistuple = tuple(("apple", "banana", "cherry")) # note the
double round-brackets
print(thistuple)
```

Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	**Description**
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Python Dictionaries

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

Example

Create and print a dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

Example

Get the value of the "model" key:

```
x = thisdict["model"]
```

There is also a method called `get()` that will give you the same result:

Example

Get the value of the "model" key:

```
x = thisdict.get("model")
```

Change Values

You can change the value of a specific item by referring to its key name:

Example

Change the "year" to 2018:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
```

Loop Through a Dictionary

You can loop through a dictionary by using a `for` loop.

When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.

Example

Print all key names in the dictionary, one by one:

```
for x in thisdict:
  print(x)
```

Example

Print all *values* in the dictionary, one by one:

```
for x in thisdict:
  print(thisdict[x])
```

Example

You can also use the `values()` method to return values of a dictionary:

```
for x in thisdict.values():
  print(x)
```

Example

Loop through both *keys* and *values*, by using the `items()` method:

```
for x, y in thisdict.items():
  print(x, y)
```

Check if Key Exists

To determine if a specified key is present in a dictionary use the `in` keyword:

Example

Check if "model" is present in the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict
dictionary")
```

Dictionary Length

To determine how many items (key-value pairs) a dictionary has, use the `len()` function.

Example

Print the number of items in the dictionary:

```
print(len(thisdict))
```

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

Example

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

Removing Items

There are several methods to remove items from a dictionary:

Example

The `pop()` method removes the item with the specified key name:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

Example

The `popitem()` method removes the last inserted item (in versions before 3.7, a random item is removed instead):

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

Example

The del keyword removes the item with the specified key name:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

Example

The del keyword can also delete the dictionary completely:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict
print(thisdict) #this will cause an error because "thisdict"
no longer exists.
```

Example

The clear() method empties the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

Copy a Dictionary

You cannot copy a dictionary simply by typing `dict2 = dict1`, because: `dict2` will only be a *reference* to `dict1`, and changes made in `dict1` will automatically also be made in `dict2`.

There are ways to make a copy, one way is to use the built-in Dictionary method `copy()`.

Example

Make a copy of a dictionary with the `copy()` method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

Another way to make a copy is to use the built-in function `dict()`.

Example

Make a copy of a dictionary with the `dict()` function:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

Nested Dictionaries

A dictionary can also contain many dictionaries, this is called nested dictionaries.

Example

Create a dictionary that contain three dictionaries:

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
```

Or, if you want to nest three dictionaries that already exists as dictionaries:

Example

Create three dictionaries, then create one dictionary that will contain the other three dictionaries:

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

The dict() Constructor

It is also possible to use the `dict()` constructor to make a new dictionary:

Example

```
thisdict = dict(brand="Ford", model="Mustang", year=1964)
# note that keywords are not string literals
# note the use of equals rather than colon for the assignment
print(thisdict)
```

Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

Method	**Description**
clear()	Removes all the elements from the dictionary

copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value

update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

CONDITIONAL AND LOOPS

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the `if` keyword.

Example

If statement:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

In this example we use two variables, `a` and `b`, which are used as part of the if statement to test whether `b` is greater than `a`.
As `a` is `33`, and `b` is `200`, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Example

If statement, without indentation (will raise an error):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

Elif

The `elif` keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

Example

```python
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

In this example `a` is equal to `b`, so the first condition is not true, but the `elif` condition is true, so we print to screen that "a and b are equal".

Else

The `else` keyword catches anything which isn't caught by the preceding conditions.

Example

```python
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

In this example `a` is greater than `b`, so the first condition is not true, also the `elif` condition is not true, so we go to the `else` condition and print to screen that "a is greater than b".

You can also have an `else` without the `elif`:

Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.

Example

One line if statement:

```
if a > b: print("a is greater than b")
```

Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:

Example

One line if else statement:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

This technique is known as **Ternary Operators**, or **Conditional Expressions**.

You can also have multiple else statements on the same line:

Example

One line if else statement, with 3 conditions:

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

And

The and keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, AND if c is greater than a:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

Or

The or keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, OR if a is greater than c:

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

Nested If

You can have `if` statements inside `if` statements, this is called *nested* `if` statements.

Example

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

The pass Statement

`if` statements cannot be empty, but if you for some reason have an `if` statement with no content, put in the `pass` statement to avoid getting an error.

Example

```
a = 33
b = 200

if b > a:
  pass
```

Python While Loops

Python Loops

Python has two primitive loop commands:

- `while` loops
- `for` loops

The while Loop

With the `while` loop we can execute a set of statements as long as a condition is true.

Example

Print i as long as i is less than 6:

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

Note: remember to increment i, or else the loop will continue forever.

The `while` loop requires relevant variables to be ready, in this example we need to define an indexing variable, `i`, which we set to 1.

The break Statement

With the `break` statement we can stop the loop even if the while condition is true:

Example

Exit the loop when i is 3:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

The continue Statement

With the `continue` statement we can stop the current iteration, and continue with the next:

Example

Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

The else Statement

With the else statement we can run a block of code once when the condition no longer is true:

Example

Print a message once the condition is false:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

Python For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Example

Print each fruit in a fruit list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

The for loop does not require an indexing variable to set beforehand.

Looping Through a String

Even strings are iterable objects, they contain a sequence of characters:

Example

Loop through the letters in the word "banana":

```
for x in "banana":
  print(x)
```

The break Statement

With the `break` statement we can stop the loop before it has looped through all the items:

Example

Exit the loop when `x` is "banana":

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

Example

Exit the loop when `x` is "banana", but this time the break comes before the print:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

The continue Statement

With the `continue` statement we can stop the current iteration of the loop, and continue with the next:

Example

Do not print banana:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

Try it Yourself »

The range() Function

To loop through a set of code a specified number of times, we can use the `range()` function,

The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example

Using the range() function:

```
for x in range(6):
  print(x)
```

Try it Yourself »

Note that `range(6)` is not the values of 0 to 6, but the values 0 to 5.

The `range()` function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: `range(2, 6)`, which means values from 2 to 6 (but not including 6):

Example

Using the start parameter:

```
for x in range(2, 6):
  print(x)
```

The `range()` function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: `range(2, 30, 3)`:

Example

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
  print(x)
```

Else in For Loop

The `else` keyword in a `for` loop specifies a block of code to be executed when the loop is finished:

Example

Print all numbers from 0 to 5, and print a message when the loop has ended:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example

Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

The pass Statement

`for` loops cannot be empty, but if you for some reason have a `for` loop with no content, put in the `pass` statement to avoid getting an error.

Example

```
for x in [0, 1, 2]:
  pass
```

Python File Open

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

File Handling

The key function for working with files in Python is the `open()` function.

The `open()` function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

`"r"` - Read - Default value. Opens a file for reading, error if the file does not exist

`"a"` - Append - Opens a file for appending, creates the file if it does not exist

`"w"` - Write - Opens a file for writing, creates the file if it does not exist

`"x"` - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

`"t"` - Text - Default value. Text mode

`"b"` - Binary - Binary mode (e.g. images)

Syntax

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

Because `"r"` for read, and `"t"` for text are the default values, you do not need to specify them.

Python File Open

Open a File on the Server

Assume we have the following file, located in the same folder as Python:

demofile.txt

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

To open the file, use the built-in `open()` function.

The `open()` function returns a file object, which has a `read()` method for reading the content of the file:

Example

```
f = open("demofile.txt", "r")
print(f.read())
```

Run Example »

If the file is located in a different location, you will have to specify the file path, like this:

Example

Open a file on a different location:

```
f = open("D:\\myfiles\welcome.txt", "r")
print(f.read())
```

Run Example »

Read Only Parts of the File

By default the `read()` method returns the whole text, but you can also specify how many characters you want to return:

Example

Return the 5 first characters of the file:

```
f = open("demofile.txt", "r")
print(f.read(5))
```

Python File Write

Write to an Existing File

To write to an existing file, you must add a parameter to the `open()` function:

`"a"` - Append - will append to the end of the file

`"w"` - Write - will overwrite any existing content

Example

Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Example

Open the file "demofile3.txt" and overwrite the content:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the appending:
f = open("demofile3.txt", "r")
print(f.read())
```

Note: the "w" method will overwrite the entire file.

Create a New File

To create a new file in Python, use the `open()` method, with one of the following parameters:

`"x"` - Create - will create a file, returns an error if the file exist

`"a"` - Append - will create a file if the specified file does not exist

`"w"` - Write - will create a file if the specified file does not exist

Example

Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

Result: a new empty file is created!

Example

Create a new file if it does not exist:

```
f = open("myfile.txt", "w")
```

Python Delete File

Delete a File

To delete a file, you must import the OS module, and run its `os.remove()` function:

Example

Remove the file "demofile.txt":

```
import os
os.remove("demofile.txt")
```

Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

Example

Check if file exists, *then* delete it:

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

Delete Folder

To delete an entire folder, use the `os.rmdir()` method:

Example

Remove the folder "myfolder":

```
import os
os.rmdir("myfolder")
```

Errors and Exceptions

Python Try Except

The `try` block lets you test a block of code for errors.

The `except` block lets you handle the error.

The `finally` block lets you execute code, regardless of the result of the try- and except blocks.

Exception Handling

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the `try` statement:

Example

The `try` block will generate an exception, because `x` is not defined:

```
try:
  print(x)
except:
  print("An exception occurred")
```

Since the try block raises an error, the except block will be executed.

Without the try block, the program will crash and raise an error:

Example

This statement will raise an error, because `x` is not defined:

```
print(x)
```

Many Exceptions

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

Example

Print one message if the try block raises a `NameError` and another for other errors:

```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
```

```
except:
  print("Something else went wrong")
```

Else

You can use the else keyword to define a block of code to be executed if no errors were raised:

Example

In this example, the try block does not generate any error:

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

Finally

The finally block, if specified, will be executed regardless if the try block raises an error or not.

Example

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

This can be useful to close objects and clean up resources:

Example

Try to open and write to a file that is not writable:

```
try:
  f = open("demofile.txt")
  f.write("Lorum Ipsum")
except:
  print("Something went wrong when writing to the file")
finally:
  f.close()
```

The program can continue, without leaving the file object open.

Raise an exception

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the `raise` keyword.

Example

Raise an error and stop the program if x is lower than 0:

```
x = -1

if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

The `raise` keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

Example

Raise a TypeError if x is not an integer:

```
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

Python Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

Creating a Function

In Python a function is defined using the `def` keyword:

Example

```
def my_function():
  print("Hello from a function")
```

Calling a Function

To call a function, use the function name followed by parenthesis:

Example

```
def my_function():
  print("Hello from a function")

my_function()
```

Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Arguments are often shortened to *args* in Python documentations.

Parameters or Arguments?

The terms *parameter* and *argument* can be used for the same thing: information that are passed into a function.

From a function's perspective:

A parameter is the variable listed inside the parentheses in the function definition.

An argument is the value that is sent to the function when it is called.

Number of Arguments

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

Example

This function expects 2 arguments, and gets 2 arguments:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

If you try to call the function with 1 or 3 arguments, you will get an error:

Example

This function expects 2 arguments, but gets only 1:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil")
```

Arbitrary Arguments, *args

If you do not know how many arguments that will be passed into your function, add a `*` before the parameter name in the function definition.

This way the function will receive a *tuple* of arguments, and can access the items accordingly:

Example

If the number of arguments is unknown, add a * before the parameter name:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

Try it Yourself »

Arbitrary Arguments are often shortened to **args* in Python documentations.

Keyword Arguments

You can also send arguments with the *key* = *value* syntax.

This way the order of the arguments does not matter.

Example

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3
= "Linus")
```

The phrase *Keyword Arguments* are often shortened to *kwargs* in Python documentations.

Arbitrary Keyword Arguments, **kwargs

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

This way the function will receive a *dictionary* of arguments, and can access the items accordingly:

Example

If the number of keyword arguments is unknown, add a double ** before the parameter name:

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

Arbitrary Kword Arguments are often shortened to ***kwargs* in Python documentations.

Default Parameter Value

The following example shows how to use a default parameter value.

If we call the function without argument, it uses the default value:

Example

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Passing a List as an Argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

E.g. if you send a List as an argument, it will still be a List when it reaches the function:

Example

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

Return Values

To let a function return a value, use the `return` statement:

Example

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Try it Yourself »

The pass Statement

function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

Example

```
def myfunction():
  pass
```

Recursion

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

In this example, tri_recursion() is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

To a new developer it can take some time to work out how exactly this works, best way to find out is by testing and modifying it.

Example

Recursion Example

```
def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
tri_recursion(6)
```

Python Modules

What is a Module?

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

Create a Module

To create a module just save the code you want in a file with the file extension `.py`:

Example

Save this code in a file named `mymodule.py`

```
def greeting(name):
  print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the `import` statement:

Example

Import the module named mymodule, and call the greeting function:

```
import mymodule

mymodule.greeting("Jonathan")
```

Note: When using a function from a module, use the syntax: *module_name.function_name*.

Variables in Module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):

Example

Save this code in the file `mymodule.py`

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example

Import the module named mymodule, and access the person1 dictionary:

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

Naming a Module

You can name the module file whatever you like, but it must have the file extension `.py`

Re-naming a Module

You can create an alias when you import a module, by using the `as` keyword:

Example

Create an alias for `mymodule` called `mx`:

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

Built-in Modules

There are several built-in modules in Python, which you can import whenever you like.

Example

Import and use the `platform` module:

```
import platform

x = platform.system()
print(x)
```

Using the dir() Function

There is a built-in function to list all the function names (or variable names) in a module. The `dir()` function:

Example

List all the defined names belonging to the platform module:

```
import platform

x = dir(platform)
print(x)
```

Note: The dir() function can be used on *all* modules, also the ones you create yourself.

Import From Module

You can choose to import only parts from a module, by using the `from` keyword.

Example

The module named `mymodule` has one function and one dictionary:

```
def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example

Import only the person1 dictionary from the module:

```python
from mymodule import person1

print (person1["age"])
```

Python Classes and Objects

Python Classes/Objects

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword `class`:

Example

Create a class named MyClass, with a property named x:

```python
class MyClass:
  x = 5
```

Create Object

Now we can use the class named MyClass to create objects:

Example

Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)
```

The __init__() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in __init__() function.

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Example

Create a class named Person, use the __init__() function to assign values for name and age:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Note: The `__init__()` function is called automatically every time the class is being used to create a new object.

Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

Let us create a method in the Person class:

Example

Insert a function that prints a greeting, and execute it on the p1 object:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Note: The `self` parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

The self Parameter

The `self` parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named `self` , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Example

Use the words *mysillyobject* and *abc* instead of *self*:

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Modify Object Properties

You can modify properties on objects like this:

Example

Set the age of p1 to 40:

```
p1.age = 40
```

Delete Object Properties

You can delete properties on objects by using the `del` keyword:

Example

Delete the age property from the p1 object:

```python
del p1.age
```

Delete Objects

You can delete objects by using the `del` keyword:

Example

Delete the p1 object:

```python
del p1
```

The pass Statement

`class` definitions cannot be empty, but if you for some reason have a `class` definition with no content, put in the `pass` statement to avoid getting an error.

Example

```python
class Person:
  pass
```