|  |
| --- |
| **FREE AND OPEN SOURCE SOFTWARE** |
| **UNIT-1** |
| **BY : RIYA KHANNA AND GEETA** |

**INTRODUCTION**

**# The philosophy of OSS**

There has always been a debate between open and closed source software for a variety of different reasons. One of the most interesting part of this conversation is security. Open source is completely transparent so you don’t need to take a company’s word on what a piece of software does, because you can verify it for yourself by analysing the code.

On the other hand closed source software’s code isn’t available publicly to view, so a company could do something malicious and give a program hidden features to collect information about the users, and there may be no way for the users to be aware of what’s happening. A lot of people would claim that open source software is more exposed as the code is available publicly and there could be a malicious person that finds a bug or vulnerability and uses it to steal sensitive data.

Some would claim that closed source is more secure as the code isn’t publicly available. In this case no bugs would be found by the public, and therefore could not be exploited by a malicious user. Most of the exploits are publicly shared once a patch has been created to fix the problem, however this comes at the cost of transparency.

Releasing patches for proprietary software takes longer because authorized engineers would have to create a patch, test it and then deploy it. Open source works differently, anyone can submit a patch for a program, and because of this bugs usually get fixed much quicker. As the code is publicly available this means it can be looked at and analysed by more people. So the scenario of only one person finding a bug and exploiting it is unlikely, however it is still a possibility.

The coding communities usually get really involved, and this is the real beauty of open source. They could create patches to fix bugs in a particular program. Even some hardware that wasn’t designed to work on Linux and doesn’t have support for it can be used with a custom driver created by other within the community that are facing the same issue.

Linux is a great example when it comes to variety, you have a lot of distributions for a variety of people with different skill levels, and if there’s something you don’t like you have the flexibility to change it. For example a new user to Linux may prefer to use something like Mint, Zorin or Ubuntu, whereas a more advanced user may be more comfortable with Debian and the most advanced users may prefer Arch

**# Advantages of Open Sources**

* Lesser hardware costs
* High-quality software
* No vendor lock-in
* Integrated management
* Simple license management
* Lower software costs
* Abundant support.
* Scaling and consolidating

**# Applications of OSS**

The main application of free and open source software is due to the fact that it is available at very low cost, other applications are from research and training where young minds are taught about the working of software. So that they can build new products.

**A.Business Firms**

FOSS has found a wide application in many of the business firms. As these softwares are available free of cost so companies can gain huge profits by using them. Whereas the proprietary softwares require huge capital investments. One of the essential features of the Free and Open source software is that it can be changed as required. Using this feature they can customise the software as required .So now they need not to be dependent on the vendors for any update or change in the software .Below given are some of the FOSS that can be used in business houses.

* 1. **Linux** –It is a type operating system which is derived from the UNIX family. Linux can be used on wide variety of computer hardware ranging from mobile phones, tablet computers and video game consoles to mainframes and supercomputers. Linux is predominantly known for its usage in server. In a 2009 survey it was found that about 40% of the servers use Linux. The popularity of Linux on standard desktops (and laptops) has been increasing over the years due to the graphical user environments. The two popular environments are GNOME and KDE both of which support a wide variety of languages.
  2. **My SQL**- It is the world's most popular open source database software, with over 100 million copies of its software downloaded or distributed till date. With its superior speed, reliability, and ease of use, My SQL has become the preferred choice for Web, Telecom companies and forward-thinking corporate IT Managers because it eliminates the major problems associated with downtime, maintenance and administration for modern, online applications. Many of the world's largest and fastest-growing organizations use MySQL to save time and money powering their high-volume Web sites, critical business systems, and packaged software — including industry leaders such as Yahoo!, AlcatelLucent, Google, Nokia, YouTube, Wikipedia, and Booking.com. MySQL is a key part of LAMP (Linux, Apache, MySQL, PHP / Perl / Python), the fast-growing open source enterprise software stack. More and more companies are using LAMP as an alternative to expensive proprietary software stacks because of its lower cost and freedom from vendor lock-in.
  3. **Apache**-The Apache HTTP Server, commonly referred to as Apache is web server software notable for playing a key role in the initial growth of the World Wide Web. In 2009 it became the first web server software to surpass the 100 million web site milestone. Apache is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation. The application is available for a wide variety of Operating Systems like GNU, Linux, Solaris, MAC OS X, and Microsoft Windows. Since April 1996 Apache has been the most popular HTTP server software in use. As of February 2010 Apache served over 54.46% of all websites and over 66% of the million busiest sites. Apache is primarily used to serve both static content and dynamic web pages on the World Wide Web.
  4. **BIND** is by far the most widely used DNS software on the Internet. It provides a robust and stable platform on top of which organizations can build distributed computing systems with the knowledge that those systems are fully compliant with published DNS standards. BIND is open source software that implements the Domain Name System (DNS) protocols for the Internet. It is a reference implementation of those protocols, but it is also production-grade software, suitable for use in high volume and high-reliabilityapplications. BIND is available for free download under the terms of the ISC License, a BSD style license.
  5. **Sendmail** is a general purpose network mailing routing facility that supports many kinds of mail-transfer and - delivery methods including the Simple Mail Transfer Protocol used for email transport over the internet. Sendmail is a well-known project of free and open source software that has developed both as proprietary and free software. In 2001, approximately 42% of the publiclyreachable mail-servers on the Internet ran Sendmail. More recent surveys have suggested a decline, with 29.4% of mail servers in August 2007 detected as running Sendmail in a study performed by E-Soft, Inc. Sendmail is trailed by Microsoft Exchange Server, Exim, Postfix; these four being the only mail servers with more than 10% of the total.

**B. Educational Institutes**

FOSS has found a large application in the field of research and education. Educational institutes can use the open source software to teach the future coders about how a software works .These softwares come with open code that can be used, modified many a times to learn different things. In contrast the proprietary software comes with only training of how to use the software i.e. the working manual. In other words it shows with it what one can do. It never discloses the codes to the users. For example while downloading the Linux kernel a open source software one can find there are two options one for new users another for experienced users .If clicked on the other option one can change the settings as per requirement. These features are only available in FOSS.

**# Free And Open Source Software**

Free software is a term coined by Richard Stallman, programmer in MIT Artificial Intelligence Lab. According to him free in free software means the freedom one can get from using these softwares. These softwares could be used, modified, redistributed without any permission required.

Open source software is basically the same in which the source codes of the software are available at free or nominal cost so one can install it and use it.

Free softwares should satisfy the following criteria in order to be called as free software. These features are laid down by Richard Stallman creator of GPL (GNU General public License)

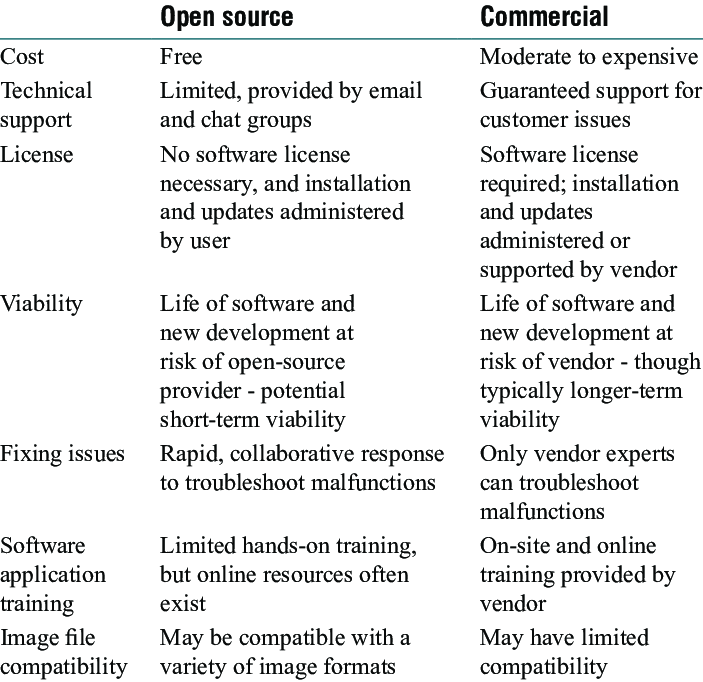
A. The freedom to run the program for any purpose.

B. The freedom to study how the program works and modify it.

C. The freedom to redistribute copies.

D. The freedom to distribute the modified copies so as to help the entire community.

**# Commercial Software vs OSS**



**# Free Software vs Freeware**

|  |  |  |
| --- | --- | --- |
|  | **Free Software** | **Freeware** |
| **About** | Free software is [software](https://www.diffen.com/difference/Hardware_vs_Software) that can be used, studied, and modified without restriction, and which can be copied and redistributed in modified or unmodified form either without restriction. | Freeware refers to software that anyone can download from the Internet and use for free. |
| **Inception** | 1983 by Richard Stallmanto satisfy the need for and to give the benefit of "software freedom" to computer users | The term freeware was first used by Andrew Fluegelman in 1982, when he wanted to sell a communications program named PC-Talk. |
| **License and Copyright** | [GNU General Public License](https://data.diffen.com/GPL) or sometime similar. A copyright is usually put just on the name of the software. | User license or EULA (End User License Agreement) is an important part of freeware. Each license is specific to the freeware. [Copyright](https://www.diffen.com/difference/Copyright_vs_Trademark) laws are also applicable to Freeware. |
| **Features** | All the features are free. | All the features are free. |
| **Distribution** | Programs can be distributed free of cost. | Freeware programs can be distributed free of cost |
| **Example** | [Mozilla Firefox](https://data.diffen.com/Firefox), gedit, vim, pidgin,GNU Coreutils, [Linux](https://www.diffen.com/difference/Linux_vs_Unix) kernel | Adobe PDF, Google Talk, yahoo messenger, MSN messenger |

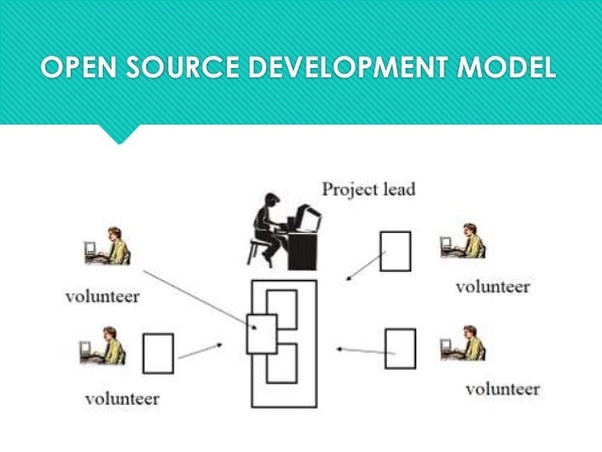
**# Open source development models**

**Open source software** is becoming increasingly popular. Since its inception, the open source movement has provided software users with more choices, lower software acquisition cost, flexible software customization, and some highly popular software products. The success of open source can be partly attributed to the unique development process.

**Some of the unique characteristics of the open source development model include**

* Bottom up development
* Small, Incremental changes
* Release early, release often
* Peer Review
* Continuous quality improvement

**How Open source development model work?**



**Benefits of Adopting Open Source Working Methods**

* Fast development cycle with small incremental changes
* Pay special attention to quality and security
* Encourage reuse
* Build reusable software components
* Respect and follow community coding style
* Flag problems early and review with team
* Foster innovation

The [**Open source development**](https://goo.gl/kfqoND) model has proved to be a successful model that allows faster development, faster testing, higher innovation, peer review, openness and transparency.

Compared to the other techniques, open source web development is pretty easy to learn, understand and master as well. You can check the details of the same from various online sources. If you want to attract many big corporate houses means Open source is an option for your web design.

**HISTORY**

**# History of free and open-source software**

In the 1950s and 1960s, computer operating software and compilers were delivered as a part of hardware purchases without separate fees. At the time, source code, the human-readable form of software, was generally distributed with the software providing the ability to fix bugs or add new functions.[1] Universities were early adopters of computing technology. Many of the modifications developed by universities were openly shared, in keeping with the academic principles of sharing knowledge, and organizations sprung up to facilitate sharing. As large-scale operating systems matured, fewer organizations allowed modifications to the operating software, and eventually such operating systems were closed to modification. However, utilities and other added-function applications are still shared and new organizations have been formed to promote the sharing of software.

**#What is BSD?**

BSD is a free and open-source Unix-like operating system descended from the Berkeley Software Distribution (BSD), which was based on Research Unix.

The first version of FreeBSD was released in 1993.

In 2005, FreeBSD was the most popular open-source BSD operating system, accounting for more than three-quarters of all installed BSD systems

FreeBSD has similarities with Linux, with two major differences in scope and licensing: FreeBSD maintains a complete system, i.e. the project delivers a kernel, device drivers, userland utilities, and documentation, as opposed to Linux only delivering a kernel and drivers, and relying on third-parties for system software

and FreeBSD source code is generally released under a permissive BSD license, as opposed to the copyleft GPL used by Linux.

**#Features of BSD:**

**Uses:**

As a general purpose operating system, FreeBSD is used in various scenarios

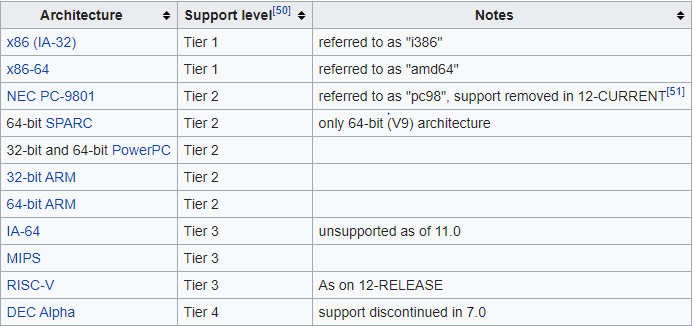
Server:FreeBSD contains a significant collection of server-related software in the base system and the ports collection, allowing FreeBSD to be configured and used as a mail server, web server, firewall, FTP server, DNS server and a router, among other applications.

**Security:**

FreeBSD provides several security-related features including access control lists (ACLs),[37] security event auditing, extended file system attributes, mandatory access controls (MAC)[38] and fine-grained capabilities.[39] These security enhancements were developed by the TrustedBSD project

**Portability**

FreeBSD has been ported to a variety of instruction set architectures. The FreeBSD project organizes architectures into tiers that characterize the level of support provided. Tier 1 architectures are mature and fully supported. Tier 2 architectures are undergoing major development. Tier 3 architectures are experimental or are no longer under active development and Tier 4 architectures have no support at all.



**Kernel :**

FreeBSD's kernel provides support for some essential tasks such as managing processes, communication, booting and filesystems.

FreeBSD has a monolithic kernel,[72] with a modular design. Different parts of the kernel, such as drivers, are designed as modules.

The user can load and unload these modules at any time.

ULE is the default scheduler in FreeBSD since version 7.1, it supports SMP and SMT.

**Documentation and support :**

FreeBSD's documentation consists of its handbooks, manual pages, mailing list archives, FAQs and a variety of articles, mainly maintained by The FreeBSD Documentation Project. FreeBSD's documentation is translated into several languages

**#What is GNU?**

GNU stands for General Public License.

GNU is a Unix-compatible operating system developed by the GNU project, which was started in 1983 by Richard Stallman with the goal of producing nonproprietary software.

users may download, modify and redistribute GNU software.

**#What is GNU project ?**

The GNU project was launched in September 1983 by Richard M. Stallman to create a complete operating system which is Free Software.

Software development work started the following January.

Today we have several Free Software operating systems which respect the users' freedom by giving everybody the right to use, study, share and improve the software for any purpose.

The GNU project consists of numerous smaller sub-projects maintained by volunteers or businesses or combinations of the two.

These sub-projects themselves are also called "GNU projects" or "GNU packages."

**#The free software foundation and the GNU project**

Stallman established the Free Software Foundation in October 1985 to assist administrative, legal, and organisational aspects of the GNU project and also to spread the use and knowledge of Free Software. The main licences of the GNU project are the GNU General Public License (GPL) and the GNU Lesser General Public License (LGPL, originally called GNU Library General Public License). Over the years they have become established as the most widely used licences for Free Software.

**#Open Source Software movement**

The open-source-software movement is a movement that supports the use of open-source licenses for some or all software, a part of the broader notion of open collaboration

The open-source movement was started to spread the concept/idea of open-source software.

Programmers who support the open-source-movement philosophy contribute to the open-source community by voluntarily writing and exchanging programming code for software development.

The term "open source" requires that no one can discriminate against a group in not sharing the edited code or hinder others from editing their already-edited work. This approach to software development allows anyone to obtain and modify open-source code.

These modifications are distributed back to the developers within the open-source community of people who are working with the software. In this way, the identities of all individuals participating in code modification are disclosed and the transformation of the code is documented over time.

This method makes it difficult to establish ownership of a particular bit of code but is in keeping with the open-source-movement philosophy. These goals promote the production of high-quality programs as well as working cooperatively with other similarly-minded people to improve open-source technology.

This led to software such as MediaWiki, the software with which the Wikipedia website is built.

**#Commercial aspects of OSS**

commercial OSS refers to kind of software which is due licensing in contrast to community in other words a software which is more complete and which is made available in exchange foe money is called Commercial OSS.

**#Certification courses issues - Global and Indian**

A certification standard is issued by the open source initiated that the source code of computer program is model available free of charge to general public this project is teh one of largest deployment of free open source.

**FREE AND OPEN SOURCE SOFTWARE**

**UNIT -2**

**BY :**

**PRIYA AND**

**KIRANJEET KAUR**

**COMMUNITY BUILDING**

**Importance of communities in open source movement**

All participant in open source ecosystem have the opportunity to shape and improve the software. User can identify feature they need and contribute code upstream.

Open-source projects involve the hard work of dozens, hundreds, thousands, sometimes millions of developers building software for one another, not selfishly building something for one team. Many businesses who build community do so to “own” relationships, data, conversations, or intellectual property. Not open-source communities. Open-source communities are radically evolved, open communities armed with the tools to build technology’s future

No matter what kind of community you build, these principles for community success hold true. In a more general sense, they can be summed up as:

1. A vision that starts at the core of the company.
2. A value proposition for the members of the community to participate.
3. Sharing goals publically and asking for help.
4. Focusing on organization and building tools, not creating rules.
5. An open platform or safe space on which to offer different levels of responsibility.

You don’t need to be a software developer to contribute to an open source project. The code, documentation and artwork that make up an open source project have all been created, tested, used, discussed and refined by members of the project community. These processes can be broken down into a myriad tasks, requiring different skills, levels of involvement and degrees of technical expertise. So, if you want to get involved in an open source project, there is a range of roles to choose from. These include:

* providing feedback
* helping new users
* recommending the project to others
* testing and reporting or fixing bugs
* requesting new features
* writing and updating software
* creating artwork
* writing or updating documentation
* translating

All of these contributions help to keep a project active and strengthen the community. The project team and the broader community will therefore welcome and encourage participation, and attempt to make it as easy as possible for people to get involved. The exact roles and mechanisms for participating will be dictated by the project’s governance model and vary from one project to another. The tools for communicating within the community and making contributions will also vary

**JBOSS COMMUNITY**

1.Jboss community is a community of open source project. The community hosts a large number of projects that are written in various programming languages. The primary language is java but there are also projects that are written in ruby, php, node and other languages.

2.Project categories range from better testing support over IDES, application servers, application and performance monitoring to micro-services.

3.Apache licence=2.0

**DEVELOPING BLOG**

**# What is a blog anyway?**

In short, a blog is a type of website that focuses mainly on written content, also known as blog posts. In popular culture we most often hear about news blogs or celebrity blog sites, but as you’ll see in this guide, you can start a successful blog on just about any topic imaginable.

Bloggers often write from a personal perspective that allows them to connect directly with their readers. In addition, most blogs also have a “comments” section where readers can correspond with the blogger. Interacting with your readers in the comments section helps to further the connection between the blogger and the reader.

This direct connection to the reader is one of the main benefits of starting a blog. This connection allows you to interact and share ideas with other like-minded people. It also allows you to build trust with your readers. Having the trust and loyalty of your readers also opens up the door to making money from your blog, which is something i discuss later in this guide.

**#How to create successful blog?**

1.Choose a blog topic.

2.Pick a host.

3.Register a domain, subdomain, and subdirectory.

4.Decide who will write and manage your blog.

5.Write compelling and valuable content.

6.Design your blog.

7.Include CTAs(calls-to-action).

8.Determine the frequency in which you’ll share blog posts.

9.Launch your blog.

10.Track and analyze your blog’s success.

**FORUM**

A forum is a discussion platform where related ideas and views on a particular issue can be exchanged. You can setup a forum for your site or blog, where your team, customers, fans, patrons, audience, users, advocates, supporters, or friends can hold public or private discussions, as a whole or in smaller groups.

If you are planning to launch a forum, and you can’t build your own software from scratch, you can opt for any of the existing forum applications out there. Some forum applications allow you to setup only a single discussion site on a single installation, while others support multiple-forums for a single installation instance.

**# 10 best open source forum software for linux systems**

1.Discourse – discussion platform

2.phpBB- bulletin board software

3.vanilla- modern community forum

4.simple machine forum(SMF)

5.bbpress- forum software

6.MyBB- powerful forum software

7.MiniBB- community discussion forum

8.Phorum- forum software

9.FluxBB- forum software

10.PunBB- bulletin board software

# **SOCIAL NETWORK FOR SOCIAL PURPOSE**

Social networking is the use of internet-based social media sites to stay connected with friends, family, colleagues, customers, or client. Social networking can have a social purpose, a business purpose, or both, through sites such as Facebook, Twitter, Linkedln, and Instagram, among others. Social networking has become a significant base for marketers seeking to engage customers.

Despite some stiff competition, Facebook remains the most popular social network, with the reach 90% of U.S mobile user, as of October 2018, the most recent data available, as of early 2019. It was followed, in order of popularity, by Instagram, facebook messenger, twitter, and pinterest.

**# How Social Networking Works**

Marketers use social networking for increasing brand recognition and encouraging brand loyalty. Since it makes a company more accessible to new customers and more recognizable for existing customers, social networking helps promote a brand’s voice and content.

Marketers use social networking for improving [conversion rates](https://www.investopedia.com/terms/c/conversion-rate.asp). Building a following provides access to and interaction with new, recent and old customers. Sharing blog posts, images, videos or comments on social media allows followers to react, visit the company’s website and become customers.

## # Advantages and Disadvantages of Social Networking in Marketing

Customers may complement the company’s offerings and encourage others to buy the products or services. The more customers are talking about a company on social networking, the more valuable the brand authority becomes. As a brand grows stronger, more sales result. Increased company posts rank the company higher in search engines. Social networking can help establish a brand as legitimate, credible, and trustworthy.

A company may use social networking to demonstrate its customer service level and enrich its relationships with consumers. For example, if a customer complains about a product or service on Twitter, the company may address the issue immediately, apologize, and take action to make it right. However, criticism of a brand can spread very quickly on social media. This can create a virtual headache for a company's public relations department.

Although social networking itself is free, building and maintaining a company profile takes hours each week. Costs for those hours add up quickly. In addition, businesses need many followers before a social media marketing campaign starts generating a positive [return on investment (ROI)](https://www.investopedia.com/terms/r/returnoninvestment.asp). For example, submitting a post to 15 followers does not have the same effect as submitting the post to 15,000 followers.

**Free and open source software**

**Unit-3**

**BY:-**

**AASTHA,**

**JAYA**

**AND**

**DHWANI**

# What is Linux?

Just like Windows, iOS, and Mac OS, Linux is an operating system. In fact, one of the most popular platforms on the planet, Android, is powered by the Linux operating system. An operating system is software that manages all of the hardware resources associated with your desktop or laptop. To put it simply, the operating system manages the communication between your software and your hardware. Without the operating system (OS), the software wouldn’t function.

The Linux operating system comprises several different pieces:

1. **Bootloader –** The software that manages the boot process of your computer. For most users, this will simply be a splash screen that pops up and eventually goes away to boot into the operating system.
2. **Kernel –** This is the one piece of the whole that is actually called ‘Linux’. The kernel is the core of the system and manages the CPU, memory, and peripheral devices. The kernel is the lowest level of the OS.
3. **Init system –** This is a sub-system that bootstraps the user space and is charged with controlling daemons. One of the most widely used init systems is systemd? which also happens to be one of the most controversial. It is the init system that manages the boot process, once the initial booting is handed over from the bootloader (i.e., GRUB or GRand Unified Bootloader).
4. **Daemons –** These are background services (printing, sound, scheduling, etc.) that either start up during boot or after you log into the desktop.
5. **Graphical server –** This is the sub-system that displays the graphics on your monitor. It is commonly referred to as the X server or just X.
6. **Desktop environment –** This is the piece that the users actually interact with. There are many desktop environments to choose from (GNOME, Cinnamon, Mate, Pantheon, Enlightenment, KDE, Xfce, etc.). Each desktop environment includes built- in applications (such as file managers, configuration tools, web browsers, and games).
7. **Applications –** Desktop environments do not offer the full array of apps. Just like Windows and macOS, Linux offers thousands upon thousands of high-quality software titles that can be easily found and installed. Most modern Linux distributions (more on this below) include App Store-like tools that centralize and simplify application installation. For example, Ubuntu Linux has the Ubuntu Software Center (a rebrand of GNOME Software? Figure 1) which allows you to quickly search among the thousands of apps and install them from one centralized location.

# How does Linux differ from other operating systems



In many ways, Linux is similar to other operating systems you may have used before, such as Windows, OS X, or iOS. Like other operating systems, Linux has a graphical interface, and types of software you are accustomed to using on other operating systems, such as word processing applications, have Linux equivalents. In many cases, the software’s creator may have made a Linux version of the same program you use on other systems. If you can use a computer or other electronic device, you can use Linux.

But Linux also is different from other operating systems in many important ways. First, and perhaps most importantly, Linux is open source software. The code used to create Linux is free and available to the public to view, edit, and—for users with the appropriate skills—to contribute to.

Linux is also different in that, although the core pieces of the Linux operating system are generally common, there are many distributions of Linux, which include different software options. This means that Linux is incredibly customizable, because not just applications, such as word processors and web browsers, can be swapped out. Linux users also can choose core components, such as which system displays graphics, and other user-interface components.

# What is the difference between Unix and Linux

You may have heard of Unix, which is an operating system developed in the 1970s at Bell Labs by Ken Thompson, Dennis Ritchie, and others. Unix and Linux are similar in many ways, and in fact, Linux was originally created to be similar to Unix. Both have similar tools for interfacing with the systems, programming tools, filesystem layouts, and other key components. However, Unix is not free. Over the years, a number of different operating systems have been created that attempted to be “unix-like” or “unix-compatible,” but Linux has been the most successful, far surpassing its predecessors in popularity.

# Who uses Linux

You’re probably already using Linux, whether you know it or not. Depending on which user survey you look at, between one- and two-thirds of the webpages on the Internet are generated by servers running Linux.

Companies and individuals choose Linux for their servers because it is secure, and you can receive excellent support from a large community of users, in addition to companies like Canonical, SUSE, and Red Hat, which offer commercial support.

Many of the devices you own probably, such as Android phones, digital storage devices, personal video recorders, cameras, wearables, and more, also run Linux. Even your car has Linux running under the hood.



# Who “oWns” Linux

By virtue of its open source licensing, Linux is freely available to anyone. However, the trademark on the name “Linux” rests with its creator, Linus Torvalds. The source code for Linux is under copyright by its many individual authors, and licensed under the GPLv2 license. Because Linux has such a large number of contributors from across multiple decades of development, contacting each individual author and getting them to agree to a new license is virtually impossible, so that Linux remaining licensed under the GPLv2 in perpetuity is all but assured.

# How was Linux created

Linux was created in 1991 by Linus Torvalds, a then-student at the University of Helsinki. Torvalds built Linux as a free and open source alternative to Minix, another Unix clone that was predominantly used in academic settings. He originally intended to name it “Freax,” but the administrator of the server Torvalds used to distribute the original code named his directory “Linux” after a combination of Torvalds’ first name and the word Unix, and the name stuck.

Linux cheat sheets

# How can I contribute to Linux

Most of the Linux kernel is written in the C programming language, with a little bit of assembly and other languages sprinkled in. If you’re interested in writing code for the Linux kernel itself, a good place to get started is in the [Kernel Newbies FAQ,](http://kernelnewbies.org/FAQ/WhereDoIBegin) which will explain some of the concepts and processes you’ll want to be familiar with.

But the Linux community is much more than the kernel, and needs contributions from lots of other people besides programmers. Every distribution contains hundreds or thousands of programs that can be distributed along with it, and each of these programs, as well as the distribution itself, need a variety of people and skill sets to make them successful, including:

* Testers to make sure everything works on different configurations of hardware and software, and to report the bugs when it does not.
* Designers to create user interfaces and graphics distributed with various programs.
* Writers who can create documentation, how-tos, and other important text distributed with software.
* Translators to take programs and documentation from their native languages and make them accessible to people around the world.
* Packagers to take software programs and put all the parts together to make sure they run flawlessly in different distributions.
* Evangelists to spread the word about Linux and open source in general.
* And of course developers to write the software itself.



**How can I get started using Linux**

There’s some chance you’re using Linux already and don’t know it, but if you’d like to install Linux on your home computer to try it out, the easiest way is to pick a popular distribution that is designed for your platform (for example, laptop or tablet device) and give it a shot. Although there are numerous distributions available, most of the older, well-known distributions are good choices for beginners because they have large user communities that can help answer questions if you get stuck or can’t figure things out. Popular distributions include [Debian,](http://www.debian.org/) [Fedora](https://getfedora.org/), [Mint](http://www.linuxmint.com/), and [Ubuntu](http://www.ubuntu.com/), but there are many others.

**What Is Scheduling**

The **Linux scheduler** is a priority based **scheduler** that **schedules** tasks based upon their static and dynamic priorities. When these priorities are combined they form a task's goodness. Each time the **Linux scheduler** runs, every task on the run queue is examined and its goodness value is computed.

**Goals**

* fast process response time
* good throughput for background jobs
* avoidance of process starvation
* reconciliation of needs of low- and high-priority processes
* Scheduling policy is based on a combination of
* Multi-level queues

Different queues for real-time and conventional processes

* Priority scheduling

Low numbers ⇒ high priority

Priorities are dynamic (change with time)

# Round robin scheduling

* process pre-empted on expiry of quantum
* but duration of quantum typically varies from process to process

# FCFS: only for breaking ties

Scheduling policy is pre-emptive

* When a process enters the TASK RUNNING state, kernel checks priority
* If priority of new task is greater than priority of current process, scheduler is invoked

# Scheduler

* Static priority: inherited from parent
* Dynamic priority: function of
  + static priority
  + average sleep time Nature of process: interactive or batch Priorities
* Static priority (static priority)
  + low value ⇒ high priority
  + 0 – 99: real-time processes 100 – 139: conventional process Default value is 120
  + may be changed via nice()
  + new process inherits static priority of its parent
* Base time quantum
  + time (ms) allocated to a process when it has exhausted its previous time quantum

if (static\_prio < 120) base = (140-static\_prio) \* 20; else if (static\_prio >= 120) base = (140- static\_prio)\*5;

* “Average” sleep time: depends on
* Whether process is sleeping in TASK .INTERRUPTIBLE state.
* Whether process is sleeping in TASK .UNINTERRUPTIBLE state.
* decreases while a process is running
* maximum value = 1 second
* Dynamic priority (prio)

Used by scheduler when selecting new process to run

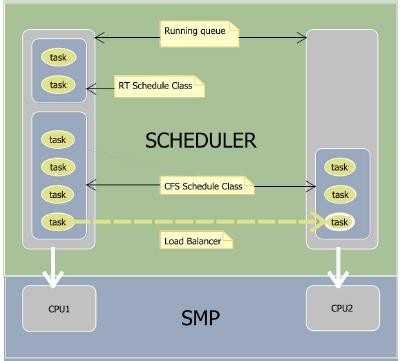
prio = MAX(100, MIN(static prio - bonus + 5, 139)) where bonus = MIN(sleep avg. / 100,

1. . interactive tasks receive a prio bonus CPU bound tasks receive a prio penalty Scheduling in Linux 14

Active vs expired process

* + Active processes: runnable processes that have not yet exhausted their time quantum
  + Expired processes: runnable processes that have exhausted their time quantum
  + Time quantum is recalculated on expiry (cf. base time quantum)
  + Active batch processes that finish time quantum → expire
  + Active interactive processes that finish time quantum:
    - if the eldest expired process has already waited for a long time, or if an expired process has higher static priority than the interactive process → expire
    - otherwise, time quantum is refilled and process remains in the set of active processes
  + Process is interactive if
    - bonus - 5 >= static prio / 4 – 28

## Data structure



Bitmap keeps track of which process lists are non-empty

Invoked once every tick

Steps 1. Decrease the ticks left in the allocated time of the process. (p->counter (2.4), or p-

>time\_slice (2.6))

1. Update dynamic priority using sleep avg.
2. If necessary, refill the time allocation for the process with the base quantum.
3. Insert process in expired queue / active queue based on (a) whether the task is interactive, (b) whether the expired tasks are starving, (c) relative priority of the process w.r.t. expired processes.

## Linux Signals

In Linux, every signal has a name that begins with characters SIG. For example:

* + A SIGINT signal that is generated when a user presses ctrl+c. This is the way to terminate programs from terminal.
  + A SIGALRM is generated when the timer set by alarm function goes off.
  + A SIGABRT signal is generated when a process calls the abort function.
  + Etc.

#### When the signal occurs, the process has to tell the kernel what to do with it. There can be three options through which a signal can be disposed :

1. The signal can be ignored. By ignoring we mean that nothing will be done when signal occurs. Most of the signals can be ignored but signals generated by hardware exceptions like divide by zero, if ignored can have weird consequences. Also, a couple of signals like SIGKILL and SIGSTOP cannot be ignored.
2. The signal can be caught. When this option is chosen, then the process registers a function with kernel. This function is called by kernel when that signal occurs. If the signal is non fatal for the process then in that function the process can handle the signal properly or otherwise it can chose to terminate gracefully.
3. Let the default action apply. Every signal has a default action. This could be process terminate, ignore etc.

#### As we already stated that two signals SIGKILL and SIGSTOP cannot be ignored. This is because these two signals provide a way for root user or the kernel to kill or stop any process in any situation .The default action of these signals is to terminate the process. Neither these signals can be caught nor can be ignored.

What Happens at Program Start-up

It all depends on the process that calls exec. When the process is started the status of all the signals is either ignore or default. Its the later option that is more likely to happen unless the process that calls exec is ignoring the signals.

It is the property of exec functions to change the action on any signal to be the default action. In simpler terms, if parent has a signal catching function that gets called on signal occurrence then if that parent execs a new child process, then this function has no meaning in the new process and hence the disposition of the same signal is set to the default in the new process.

Also, Since we usually have processes running in background so the shell just sets the quit signal disposition as ignored since we do not want the background processes to get terminated by a user pressing a ctrl+c key because that defeats the purpose of making a process run in background.

## Why Signal Catching Functions should be Re- entrant

As we have already discussed that one of the option for signal disposition is to catch the signal. In the process code this is done by registering a function to kernel which the kernel calls when the signal occurs. One thing to be kept in mind is that the function that the process registers should be re-entrant.

#### Before explaining the reason, lets first understand what are reentrant functions?

A reentrant function is a function whose execution can be stopped in between due to any reason (like due to interrupt or signal) and then can be reentered again safely before its previous invocations complete the execution.

Now coming back to the issue, Suppose a function func() is registered for a call back on a signal occurrence. Now assume that this func() was already in execution when the signal occurred. Since this function is call back for this signal so the current execution on this signal will be stopped by the scheduler and this function will be called again (due to signal).

The problem can be if func() works on some global values or data structures that are left in inconsistent state when the execution of this function was stopped in middle then the second call to same function(due to signal) may cause some undesired results.

So we say that signal catching functions should be made reentrant.

## Threads and Signals

Every thread has its own private signal mask (a mask that defines which signals are deliverable) but the way signal disposition is done is shared by all the threads in the application. This means that a disposition for a particular signal set by a thread can easily be overruled by some other thread. In this case the disposition mechanism changes for all the threads.

For example, a thread A can choose to ignore a particular signal but a thread B in the same process can choose to catch the same signal by registering a callback function to the kernel. In this case the request made by thread A gets overruled by thread B’s request.

Signals are delivered only to a single thread in any process. Apart from the the hardware exceptions or the timer expiry (which are delivered to thread which caused the event) all the signals are passed to the process arbitrarily.

## Catching a Signal

#### If a process wishes to handle certain signals then in the code, the process has to register a signal handling function to the kernel.

The following is the prototype of a signal handling function:

void <signal handler func name> (int sig)

The signal handler function has void return type and accepts a signal number corresponding to the signal that needs to be handled.

To get the signal handler function registered to the kernel, the signal handler function pointer is passed as second argument to the ‘signal’ function. The prototype of the signal function is :

void (\*signal(int signo, void (\*func )(int)))(int);

This might seems a complicated declaration. If we try to decode it :

* The function requires two arguments.
* The first argument is an integer (signo) depicting the signal number or signal value.
* The second argument is a pointer to the signal handler function that accepts an integer as argument and returns nothing (void).
* While the ‘signal’ function itself returns function pointer whose return type is void. Well, to make things easier, lets use typedef :

typedef void sigfunc(int)

So, here we have made a new type ‘sigfunc’. Now using this typedef, if we redesign the prototype of the signal handler :

sigfunc \*signal(int, sigfunc\*);

Now we see that its easier to comprehend that the signal handler function accepts an integer and a sigfunc type function pointer while it returns a sigfunc type function pointer.

## Example C Program to Catch a Signal

Most of the Linux users use the key combination Ctr+C to terminate processes in Linux.

Have you ever thought of what goes behind this. Well, whenever ctrl+c is pressed, a signal SIGINT is sent to the process. The default action of this signal is to terminate the process. But this signal can also be handled. The following code demonstrates this :

#include<stdio.h> #include<signal.h> #include<unistd.h>

void sig\_handler(int signo)

{

if (signo == SIGINT) printf("received SIGINT\n");

}

int main(void)

{

if (signal(SIGINT, sig\_handler) == SIG\_ERR) printf("\ncan't catch SIGINT\n");

// A long long wait so that we can easily issue a signal to this process while(1)

sleep(1); return 0;

}

In the code above, we have simulated a long running process using an infinite while loop.

A function sig\_handler is used a s a signal handler. This function is registered to the kernel by passing it as the second argument of the system call ‘signal’ in the main() function. The first argument to the function ‘signal’ is the signal we intend the signal handler to handle which is SIGINT in this case.

## SIGKILL, SIGSTOP and User Defined Signals

Apart from handling the standard signals(like INT, TERM etc) that are available. We can also have user defined signals that can be sent and handled. Following is the code handling a user defined signal USR1 :

#include<stdio.h>

#include<signal.h>

#include<unistd.h>

void sig\_handler(int signo)

{

if (signo == SIGUSR1)

printf("received SIGUSR1\n");

else if (signo == SIGKILL)

printf("received SIGKILL\n");

else if (signo == SIGSTOP)

printf("received SIGSTOP\n");

}

int main(void)

{

if (signal(SIGUSR1, sig\_handler) == SIG\_ERR)

printf("\ncan't catch SIGUSR1\n");

if (signal(SIGKILL, sig\_handler) == SIG\_ERR)

printf("\ncan't catch SIGKILL\n");

if (signal(SIGSTOP, sig\_handler) == SIG\_ERR) printf("\ncan't catch SIGSTOP\n");

// A long long wait so that we can easily issue a signal to this process while(1)

sleep(1); return 0;

}

We see that in the above code, we have tried to handle a user defined signal USR1. Also, as we know that two signals KILL and STOP cannot be handled. So we have also tried to handle these two signals so as to see how the ‘signal’ system call responds in this case.

When we run the above code :

$ ./sigfunc

can't catch SIGKILL

can't catch SIGSTOP

So the above output makes clear that as soon as the system call ‘signal’ tries to register handler for KILL and STOP signals, the signal function fails indicating that these two signals cannot be caught.

Now we try to pass the signal USR1 to this process using the [kill command](https://www.thegeekstuff.com/2009/12/4-ways-to-kill-a-process-kill-killall-pkill-xkill/):

$ kill -USR1 2678

and on the terminal where the above program is running we see :

$ ./sigfunc

can't catch SIGKILL

can't catch SIGSTOP received SIGUSR1

Linux Network Troubleshooting Tools

Computers are connected in a network to exchange information or resources each other. Two or more computer connected through network media called **computer network**. There are number of network devices or media are involved to form computer network. Computer loaded with **Linux Operating System** can also be a part of network whether it is small or large network by its **multitasking and multiuser** natures. Maintaining of system and network up and running is a task of **System / Network Administrator’s** job. In this article we are going to review frequently used network configuration and troubleshoot commands in Linux.

## Linux Network Configuration and Troubleshooting Commands

### ifconfig

**ifconfig** (**interface configurator**) command is use to initialize an interface, assign **IP Address** to interface and **enable** or **disable** interface on demand. With this command you can view **IP Address** and **Hardware** / **MAC address** assign to interface and also **MTU** (**Maximum transmission unit**) size.

#### # ifconfig

eth0 Link encap:Ethernet **HWaddr 00:0C:29:28:FD:4C**

**inet addr:192.168.50.2** Bcast:192.168.50.255 Mask:255.255.255.0

inet6 addr: fe80::20c:29ff:fe28:fd4c/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:6093 errors:0 dropped:0 overruns:0 frame:0 TX packets:4824 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000

RX bytes:6125302 (5.8 MiB) TX bytes:536966 (524.3 KiB)

Interrupt:18 Base address:0x2000

lo Link encap:Local Loopback

inet addr:127.0.0.1 Mask:255.0.0.0 inet6 addr: ::1/128 Scope:Host

UP LOOPBACK RUNNING MTU:16436 Metric:1

RX packets:8 errors:0 dropped:0 overruns:0 frame:0 TX packets:8 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:0

RX bytes:480 (480.0 b) TX bytes:480 (480.0 b)

**ifconfig** with interface (**eth0**) command only shows specific interface details like **IP Address**, **MAC Address** etc. with **-a** options will display all available interface details if it is disable also.

#### # ifconfig eth0

eth0 Link encap:Ethernet HWaddr 00:0C:29:28:FD:4C

inet addr:192.168.50.2 Bcast:192.168.50.255 Mask:255.255.255.0

inet6 addr: fe80::20c:29ff:fe28:fd4c/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:6119 errors:0 dropped:0 overruns:0 frame:0 TX packets:4841 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000

RX bytes:6127464 (5.8 MiB) TX bytes:539648 (527.0 KiB)

Interrupt:18 Base address:0x2000

*Assigning IP Address and Gateway*

Assigning an **IP Address** and **Gateway** to interface on the fly. The setting will be removed in case of system reboot.

#### # ifconfig eth0 192.168.50.5 netmask 255.255.255.0

*Enable or Disable Specific Interface*

To **enable** or **disable** specific Interface, we use example command as follows.

Enable eth0 **# ifup eth0**Disable eth0

#### # ifdown eth0

*Setting MTU Size*

By default **MTU** size is **1500**. We can set required **MTU** size with below command. Replace

**XXXX** with size.

#### # ifconfig eth0 mtu XXXX

*Set Interface in Promiscuous mode*

**Network interface** only received packets belongs to that particular **NIC**. If you put interface in **promiscuous** mode it will received all the packets. This is very useful to capture packets and analyze later. For this you may require superuser access.

#### # ifconfig eth0 - promisc

1. PING Command

**PING** (**Packet INternet Groper**) command is the best way to test connectivity between **two nodes**. Whether it is **Local Area Network** (**LAN**) or **Wide Area Network** (**WAN**). Ping use **ICMP** (**Internet Control Message Protocol**) to communicate to other devices. You can ping host name of **ip address** using below command.

#### # ping 4.2.2.2

PING 4.2.2.2 (4.2.2.2) 56(84) bytes of data.

64 bytes from 4.2.2.2: icmp\_seq=1 ttl=44 time=203 ms 64 bytes from 4.2.2.2: icmp\_seq=2 ttl=44 time=201 ms 64 bytes from 4.2.2.2: icmp\_seq=3 ttl=44 time=201 ms

OR

#### # ping [www.tecmint.com](http://www.tecmint.com/)

PING tecmint.com (50.116.66.136) 56(84) bytes of data.

64 bytes from 50.116.66.136: icmp\_seq=1 ttl=47 time=284 ms 64 bytes from 50.116.66.136: icmp\_seq=2 ttl=47 time=287 ms 64 bytes from 50.116.66.136: icmp\_seq=3 ttl=47 time=285 ms

In **Linux** ping command keep executing until you interrupt. Ping with **-c** option exit after **N**

number of request (success or error respond).

#### # ping -c 5 [www.tecmint.com](http://www.tecmint.com/)

PING tecmint.com (50.116.66.136) 56(84) bytes of data.

64 bytes from 50.116.66.136: icmp\_seq=1 ttl=47 time=285 ms 64 bytes from 50.116.66.136: icmp\_seq=2 ttl=47 time=285 ms 64 bytes from 50.116.66.136: icmp\_seq=3 ttl=47 time=285 ms 64 bytes from 50.116.66.136: icmp\_seq=4 ttl=47 time=285 ms 64 bytes from 50.116.66.136: icmp\_seq=5 ttl=47 time=285 ms

--- tecmint.com ping statistics ---

5 packets transmitted, 5 received, 0% packet loss, time 4295ms rtt min/avg/max/mdev = 285.062/285.324/285.406/0.599 ms

### TRACEROUTE Command

**traceroute** is a network troubleshooting utility which shows number of hops taken to reach destination also determine packets traveling path. Below we are tracing route to global **DNS server IP Address** and able to reach destination also shows path of that packet is traveling.

#### # traceroute 4.2.2.2

traceroute to 4.2.2.2 (4.2.2.2), 30 hops max, 60 byte packets

1 192.168.50.1 (192.168.50.1) 0.217 ms 0.624 ms 0.133 ms

2 227.18.106.27.mysipl.com (27.106.18.227) 2.343 ms 1.910 ms 1.799 ms

3 221-231-119-111.mysipl.com (111.119.231.221) 4.334 ms 4.001 ms 5.619 ms

4 10.0.0.5 (10.0.0.5) 5.386 ms 6.490 ms 6.224 ms

5 gi0-0-0.dgw1.bom2.pacific.net.in (203.123.129.25) 7.798 ms 7.614 ms 7.378 ms

6 115.113.165.49.static-mumbai.vsnl.net.in (115.113.165.49) 10.852 ms 5.389 ms 4.322 ms

7 ix-0-100.tcore1.MLV-Mumbai.as6453.net (180.87.38.5) 5.836 ms 5.590 ms 5.503 ms

8 if-9-5.tcore1.WYN-Marseille.as6453.net (80.231.217.17) 216.909 ms 198.864 ms 201.737 ms

9 if-2-2.tcore2.WYN-Marseille.as6453.net (80.231.217.2) 203.305 ms 203.141 ms 202.888 ms

10 if-5-2.tcore1.WV6-Madrid.as6453.net (80.231.200.6) 200.552 ms 202.463 ms 202.222 ms

11 if-8-2.tcore2.SV8-Highbridge.as6453.net (80.231.91.26) 205.446 ms 215.885 ms 202.867 ms

12 if-2-2.tcore1.SV8-Highbridge.as6453.net (80.231.139.2) 202.675 ms 201.540 ms 203.972 ms

13 if-6-2.tcore1.NJY-Newark.as6453.net (80.231.138.18) 203.732 ms 203.496 ms 202.951 ms

14 if-2-2.tcore2.NJY-Newark.as6453.net (66.198.70.2) 203.858 ms 203.373 ms 203.208 ms

15 66.198.111.26 (66.198.111.26) 201.093 ms 63.243.128.25 (63.243.128.25) 206.597 ms

66.198.111.26 (66.198.111.26) 204.178 ms

16 ae9.edge1.NewYork.Level3.net (4.68.62.185) 205.960 ms 205.740 ms 205.487 ms

17 vlan51.ebr1.NewYork2.Level3.net (4.69.138.222) 203.867 ms

vlan52.ebr2.NewYork2.Level3.net (4.69.138.254) 202.850 ms

vlan51.ebr1.NewYork2.Level3.net (4.69.138.222) 202.351 ms

18 ae-6-6.ebr2.NewYork1.Level3.net (4.69.141.21) 201.771 ms 201.185 ms 201.120 ms

19 ae-81-81.csw3.NewYork1.Level3.net (4.69.134.74) 202.407 ms 201.479 ms ae-92-

92.csw4.NewYork1.Level3.net (4.69.148.46) 208.145 ms

20 ae-2-70.edge2.NewYork1.Level3.net (4.69.155.80) 200.572 ms ae-4-

90.edge2.NewYork1.Level3.net (4.69.155.208) 200.402 ms ae-1-

60.edge2.NewYork1.Level3.net (4.69.155.16) 203.573 ms

21 b.resolvers.Level3.net (4.2.2.2) 199.725 ms 199.190 ms 202.488 ms

### NETSTAT Command

**Netstat** (**Network Statistic**) command display connection info, routing table information etc. To displays routing table information use option as **-r**.

#### # netstat -r

Kernel IP routing table

Destination Gateway Genmask Flags MSS Window irtt Iface 192.168.50.0 \* 255.255.255.0 U 0 0 0 eth0

link-local \* 255.255.0.0 U 0 0 0 eth0

default 192.168.50.1 0.0.0.0 UG 0 0 0 eth0

For more examples of **Netstat Command**, please read our earlier article on 20 Netstat Command Examples in Linux.

### DIG Command

**Dig** (**domain information groper**) query **DNS** related information like **A Record**, **CNAME**, **MX Record** etc. This command mainly use to troubleshoot **DNS** related query.

**# dig www.tecmint.com**; <<>> DiG 9.8.2rc1-RedHat-9.8.2-0.10.rc1.el6 <<>> [www.tecmint.com](http://www.tecmint.com/)

;; global options: +cmd

;; Got answer:

;; ->>HEADER<

For more examples of **Dig Command**, please read the article on 10 Linux Dig Commands to Query DNS.

### NSLOOKUP Command

**nslookup** command also use to find out **DNS** related query. The following examples shows **A Record** (**IP Address**) of **tecmint.com**.

#### # nslookup [www.tecmint.com](http://www.tecmint.com/)

Server: 4.2.2.2

Address: 4.2. 2.2#53

Non-authoritative answer:

[www.tecmint.com](http://www.tecmint.com/) canonical name = tecmint.com. Name: tecmint.com

Address: 50.116.66.136

For more **NSLOOKUP Command**, read the article on 8 Linux Nslookup Command Examples.

### ROUTE Command

**route** command also shows and manipulate **ip** routing table. To see default routing table in

**Linux**, type the following command.

#### # route

Kernel IP routing table

Destination Gateway Genmask Flags Metric Ref Use Iface 192.168.50.0 \* 255.255.255.0 U 0 0 0 eth0

link-local \* 255.255.0.0 U 1002 0 0 eth0

default 192.168.50.1 0.0.0.0 UG 0 0 0 eth0

Adding, deleting routes and default Gateway with following commands.

Route Adding

#### # route add -net 10.10.10.0/24 gw 192.168.0.1

Route Deleting

#### # route del -net 10.10.10.0/24 gw 192.168.0.1

Adding default Gateway

#### # route add default gw 192.168.0.1

1. HOST Command

**host** command to find name to **IP** or **IP** to name in **IPv4** or **IPv6** and also query **DNS** records.

#### # host [www.google.com](http://www.google.com/)

[www.google.com](http://www.google.com/) has address 173.194.38.180 [www.google.com](http://www.google.com/) has address 173.194.38.176 [www.google.com](http://www.google.com/) has address 173.194.38.177 [www.google.com](http://www.google.com/) has address 173.194.38.178 [www.google.com](http://www.google.com/) has address 173.194.38.179 [www.google.com](http://www.google.com/) has IPv6 address 2404:6800:4003:802::1014

Using **-t** option we can find out DNS Resource Records like **CNAME**, **NS**, **MX**, **SOA** etc.

#### # host -t CNAME [www.redhat.com](http://www.redhat.com/)

[www.redhat.com](http://www.redhat.com/) is an alias for wildcard.redhat.com.edgekey.net.

### ARP Command

**ARP** (Address Resolution Protocol) is useful to **view** / **add** the contents of the kernel’s **ARP tables**. To see default table use the command as.

#### # arp -e

Address HWtype HWaddress Flags Mask Iface 192.168.50.1 ether 00:50:56:c0:00:08 C eth0

### ETHTOOL Command

**ethtool** is a replacement of **mii-tool**. It is to view, setting speed and duplex of your **Network Interface Card** (**NIC**). You can set duplex permanently in **/etc/sysconfig/network-scripts/ifcfg- eth0** with **ETHTOOL\_OPTS** variable.

#### # ethtool eth0

Settings for eth0:

Current message level: 0x00000007 (7) Link detected: yes

### IWCONFIG Command

**iwconfig** command in **Linux** is use to configure a **wireless network interface**. You can see and set the basic **Wi-Fi** details like **SSID** channel and encryption. You can refer man page of **iwconfig** to know more.

#### # iwconfig [interface]

1. HOSTNAME Command

**hostname** is to identify in a network. Execute **hostname** command to see the hostname of your box. You can set hostname permanently in **/etc/sysconfig/network**. Need to reboot box once set a proper hostname.

#### # hostname

tecmint.com

### GUI tool system-config-network

Type **system-config-network** in command prompt to configure network setting and you will get nice **Graphical User Interface** (**GUI**) which may also use to configure **IP Address**, **Gateway**, **DNS** etc. as shown below image.

#### # system-config-network

*UNIT-4*

***Submitted by- Anupama, Deepika and Shefali***

*Why to Learn Python?*

Python is a interpreted, interactive and object-oriented language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages**.**

Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning Python:

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

### Python Features

As mentioned before, Python is one of the most widely used language over the web.

* Easy-to-learn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
* Easy-to-read − Python code is more clearly defined and visible to the eyes.
* Easy-to-maintain − Python's source code is fairly easy-to-maintain.
* A broad standard library − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
* Interactive Mode − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
* Portable- Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

### **First Program of Python**

* //Program to print hello world!
* >>>print(“Hello World!”)
* Output
* Hello World!

### **Python Identifiers**

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language.

For Example-

Manpower and manpower are two different identifiers in Python.

Here are naming conventions for Python identifiers −

* Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
* Starting an identifier with a single leading underscore indicates that the identifier is private.
* Starting an identifier with two leading underscores indicates a strongly private identifier.

#### Reserved Words

These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

The following list shows the Python keywords.

|  |  |  |
| --- | --- | --- |
| And | Exec | Not |
| Assert | Finally | Or |
| Break | For | Pass |
| Class | From | Print |
| Continue | Global | Raise |
| Def | If | return |
| Del | Import | try |
| Elif | In | while |
| Else | Is | with |
| Except | Lambda | yield |

##### Lines and Indentation

* Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.
* The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.

For example − if True:

print “True”

else:

print “False”

***However, the following block generates an error−***

If True:

print ”Answer” print “True” else:

print ”Answer” print “False”

Thus, in Python all the continuous lines indented with same number of spaces would form a block.

##### **Quotation in Python**

* Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.
* The triple quotes are used to span the string across multiple lines. For example, all the following are legal − word=‘word’

sentence=“This is a sentence.”

paragraph=“””This is a paragraph. It is made up of multiple lines and sentences.”””

###### **Comments in Python**

* A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.
* Code-
* #First comment
* Print “Hello, Python!” #Second comment
* This produce the following result
* ******Hello, Python!

**Variable**

* Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
* Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

****Assigning Values to Variables-**

* Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.
* The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

For example −

* Counter=100 #An integer assignment
* Miles=1000.0 #A Floating point
* Name=‘John’ #A string
* Print counter
* Print miles
* Print name

Here, 100, 1000.0 and "John" are the values assigned to counter, miles, and name variables, respectively. This produces the following result −

* 100
* 100.0
* John

**Multiple Assignment**

* Python allows you to assign a single value to several variables simultaneously. For example −
* A=b=c=1
* Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example −
* a,b,c =1,2,”john”
* Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them. For example −

* Var1=1
* Var2=10

Python supports four different numerical types –

******int (signed integers)

* long (long integers, they can also be represented in octal and hexadecimal)
* float (floating point real values)
* complex (complex numbers)
* The plus (+) sign is the string concatenation operator and the asterisk (\*) is the repetition operator. For example −
* str=‘Hello World!’
* print str #prints complete string
* print str[0] #Print first character of the string
* print str[2:5] #Print character starting from3rd to 5th
* print str[2:] #Print string starting from 3rd
* print str\*2 #Print string two times ******print str+ “’TEST” #Print concatenated string
* This will produce the following result-
* hello World!
* H
* llo
* llo World!
* Hello World! Hello World!
* Hello World! TEST

Operators

Operators are the constructs which can manipulate the value of operands.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

**Types of Operator**

*Python language supports the following types of operators.*

* Arithmetic Operators
* Comparison (Relational) Operators
* Assignment Operators
* Logical Operators
* Bitwise Operators

Python Arithmetic Operators

Assume variable whose name is a holds 10 and another variable whose name is b holds 20, then

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| *+*  *Addition* | Adds values on either side of the operator. | a + b = 30 |
| *-*  *Subtraction* | Subtracts right hand operand from left hand operand. | a – b = -10 |
| *\**  *Multiplication* | Multiplies values on either side of the operator | a \* b = 200 |
| */*  *Division* | Divides left hand operand by right hand operand | b / a = 2 |
| *%*  *Modulus* | Divides left hand operand by right hand operand and returns remainder | b % a = 0 |
| *\*\**  *Exponent* | Performs exponential (power) calculation on operators | a\*\*b =10 to the power 20 |
| *//* | Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) − | 9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -  4.0 |

##### *Python Comparison Operators*

* These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.
* Assume variable a holds 10 and variable b holds 20, then −

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| == | If the values of two operands are equal, then the condition becomes true. | (a == b) is not true. |
| != | If values of two operands are not equal, then condition becomes true. | (a != b) is true. |
| <> | If values of two operands are not equal, then condition becomes true. | (a <> b) is true. This is similar to != operator. |
| > | If the value of left operand is greater than the value of right operand, then condition becomes true. | (a > b) is not true. |
| < | If the value of left operand is less than the value of right operand, then condition becomes true. | (a < b) is true. |
| >= | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. | (a >= b) is not true. |
| <= | If the value of left operand is less than or equal to the value of right operand, then condition becomes true. | (a <= b) is true. |

### *Python Assignment Operators*

* Assume variable a holds 10 and variable b holds 20, then −

|  |  |  |
| --- | --- | --- |
| ***Operator*** | ***Description*** | ***Example*** |
| = | Assigns values from right side operands to left side operand | c = a + b assigns value of a + b into c |
| +=  Add AND | It adds right operand to the left operand and assign the result to left operand | c += a is equivalent to c = c + a |
| -= Subtract AND | It subtracts right operand from the left  operand and assign the result to left operand | c -= a is equivalent to c = c - a |
| \*=  Multiply AND | It multiplies right operand with the left  operand and assign the result to left operand | c \*= a is equivalent to c = c \* a |
| /= Divide AND | It divides left operand with the right operand and assign the result to left operand | c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a |
| %=  Modulus AND | It takes modulus using two operands and assign the result to left operand | c %= a is equivalent to c = c % a |
| \*\*= Exponent  AND | Performs exponential (power) calculation on operators and assign value to the left operand | c \*\*= a is equivalent to c = c \*\* a |
| //= Floor Division | It performs floor division on operators and assign value to the left operand | c //= a is equivalent to c = c // a |

**Python Bitwise Operators**

* Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13; Now in binary format they will be as follows −
* a = 0011 1100
* b = 0000 1101
* -----------------
* a&b = 0000 1100
* a|b = 0011 1101
* a^b = 0011 0001
* ~a = 1100 0011
* There are following Bitwise operators supported by Python language

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| & Binary AND | Operator copies a bit to the result if it exists in both operands | (a & b) (means 0000 1100) |
| |  Binary OR | It copies a bit if it exists in either operand. | (a | b) = 61 (means 0011 1101) |
| ^  Binary XOR | It copies the bit if it is set in one operand but not both. | (a ^ b) = 49 (means 0011 0001) |
| ~  Binary Ones  Complement | It is unary and has the effect of 'flipping' bits. | (~a ) = -61 (means 1100 0011 in 2's complement form due to a signed binary number. |
| <<  Binary Left Shift | The left operands value is moved left by the number of bits specified by the right operand. | a << 2 = 240 (means 1111 0000) |
| >>  Binary Right Shift | The left operands value is moved right by the number of bits specified by the right operand. | a >> 2 = 15 (means 0000 1111) |

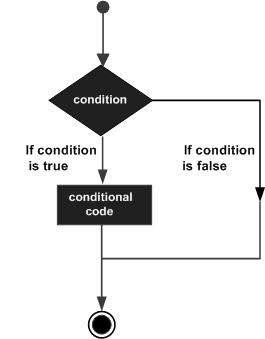
*Python Logical Operators*

* There are following logical operators supported by Python language. Assume variable a holds 10 and variable b holds 20 then

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| and  Logical AND | If both the operands are true then condition becomes true. | (a and b) is true. |
| or  Logical OR | If any of the two operands are nonzero then condition becomes true. | (a or b) is true. |
| not  Logical NOT | Used to reverse the logical state of its operand. | Not(a and b) is false. |

#### Decision making

* Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.
* Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.
* Following is the general form of a typical decision making structure found in most of the programming languages –



Python programming language assumes any non-zero and nonnull values as TRUE, and if it is either zero or null, then it is assumed as FALSE value.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

# Python If ... Else

## 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**

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**

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:

a = 200

b = 33

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 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)

## 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)

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 Sequence**

A sequence is a succession of values bound together by a container that reflects their type. Almost every stream that you put in Python is a sequence.

**Types of Sequences**

* Lists
* Tuples
* Xrange
* String

**List**

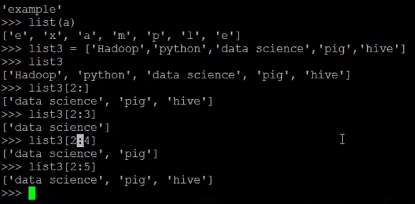
* A list is a sort of container that holds a number of other objects, in a given order.
* The list type implements the sequence protocol, and it also allows you to add and remove objects from the sequence.
* It is an ordered set of elements enclosed in square brackets.

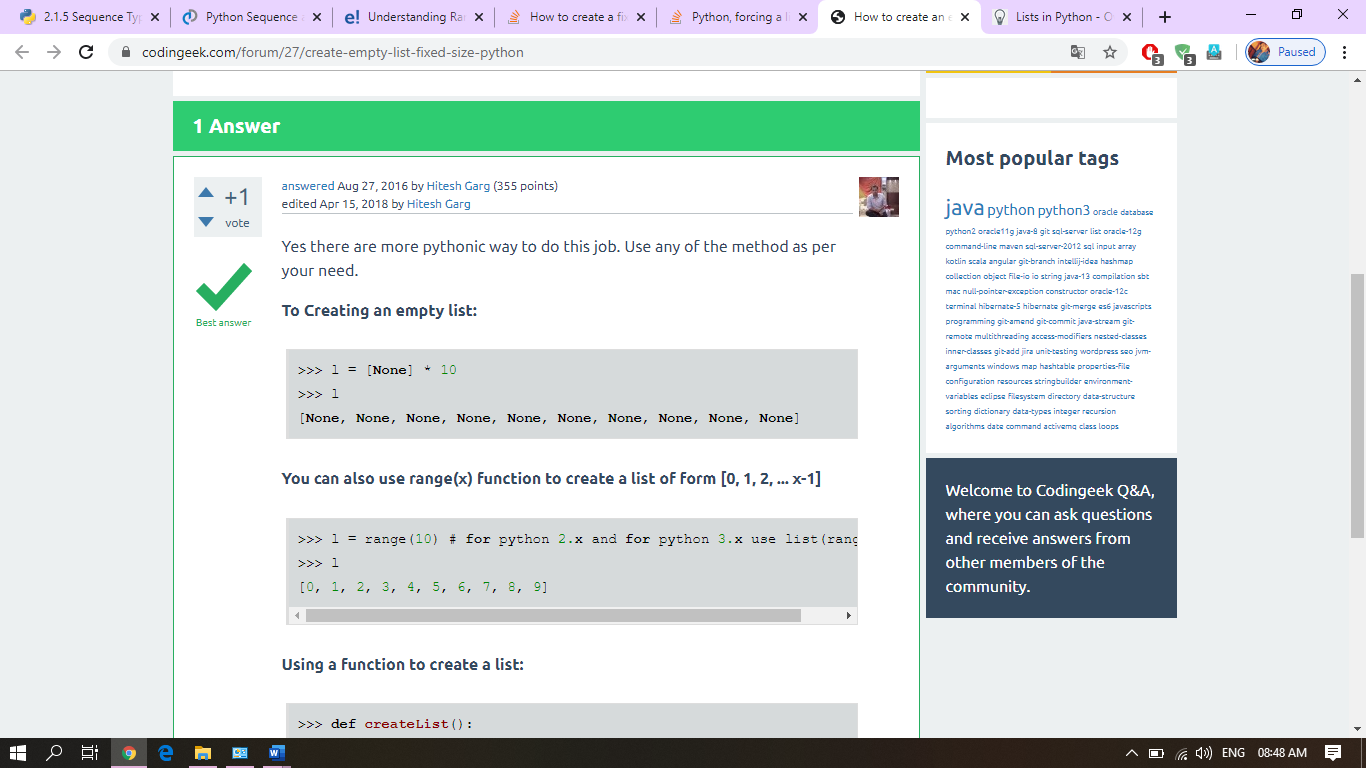
Simple definition of list – li = []

li = list() # empty list

li = list(sequence)

li = list(expression for variable in sequence)





## Iterating through elements in a List

To iterate through a list we can use for loop as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | >>>  >>> marks = [122, 45, 23, 78, 65, 12]  >>> for m in marks:  ...     print(m)  ...  122  45  23  78  65  12  >>> |

In each iteration the variable m is assigned a value from the list. Changing the value of variable m in the loop body doesn’t update the elements in the list. So, this method is commonly used to iterate over list when we don’t need to modify the elements in a list.

To modify elements, we can use a for loop in conjunction with the range() function, as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | >>>  >>> marks = [122, 45, 23, 78, 65, 12]  >>>  >>> import random  >>>  >>> for i in range(len(marks)):  ...     marks[i] = random.randint(1, 100) # assign some random value between 1 to 100 to all elements  ...  >>>  >>> marks  [59, 9, 59, 21, 75, 61]  >>>  >>> |

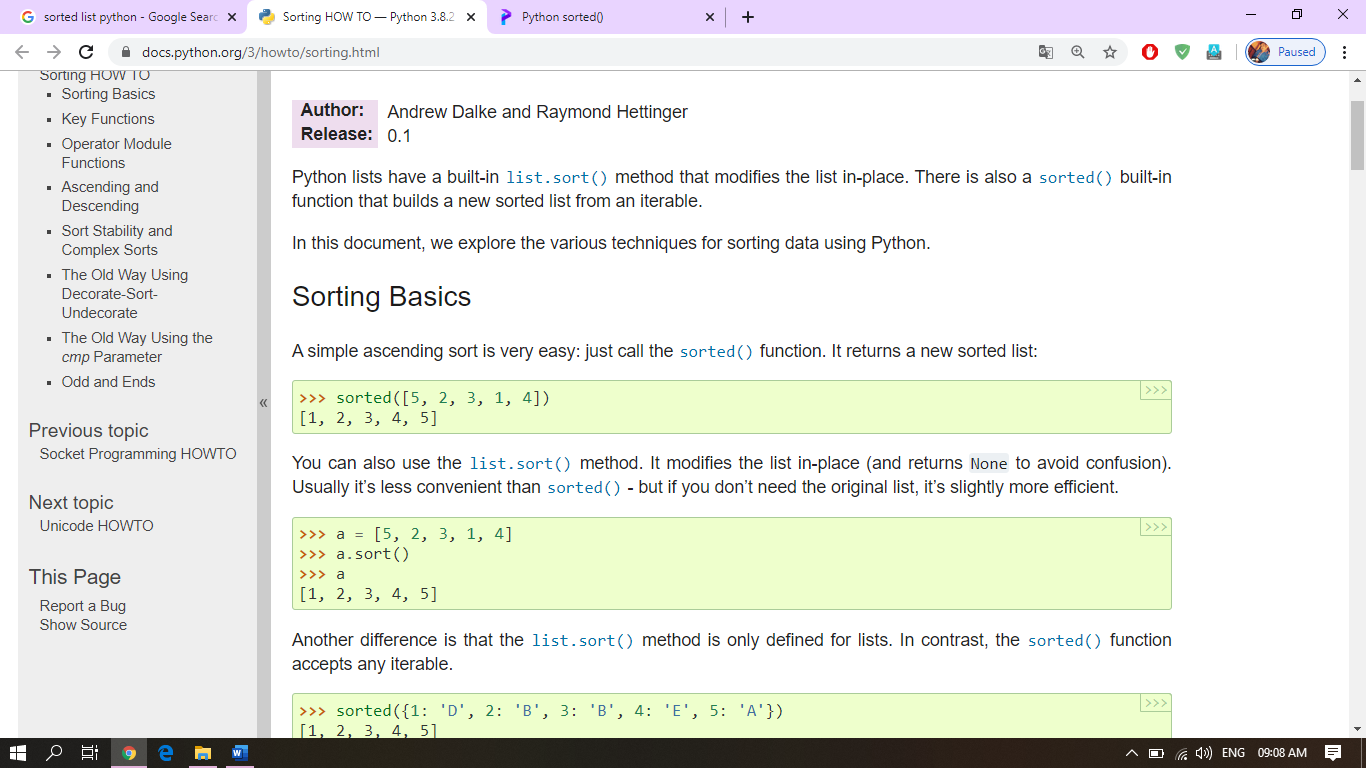
Although the for loop is a preferred way to iterate over a list, if we want, we can use while loop too. For example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | >>>  >>> i = 0  >>>  >>> while i < len(marks):  ...     print(marks[i])  ...     i += 1  ...  59  9  59  21  75  61  >>>  >>> |

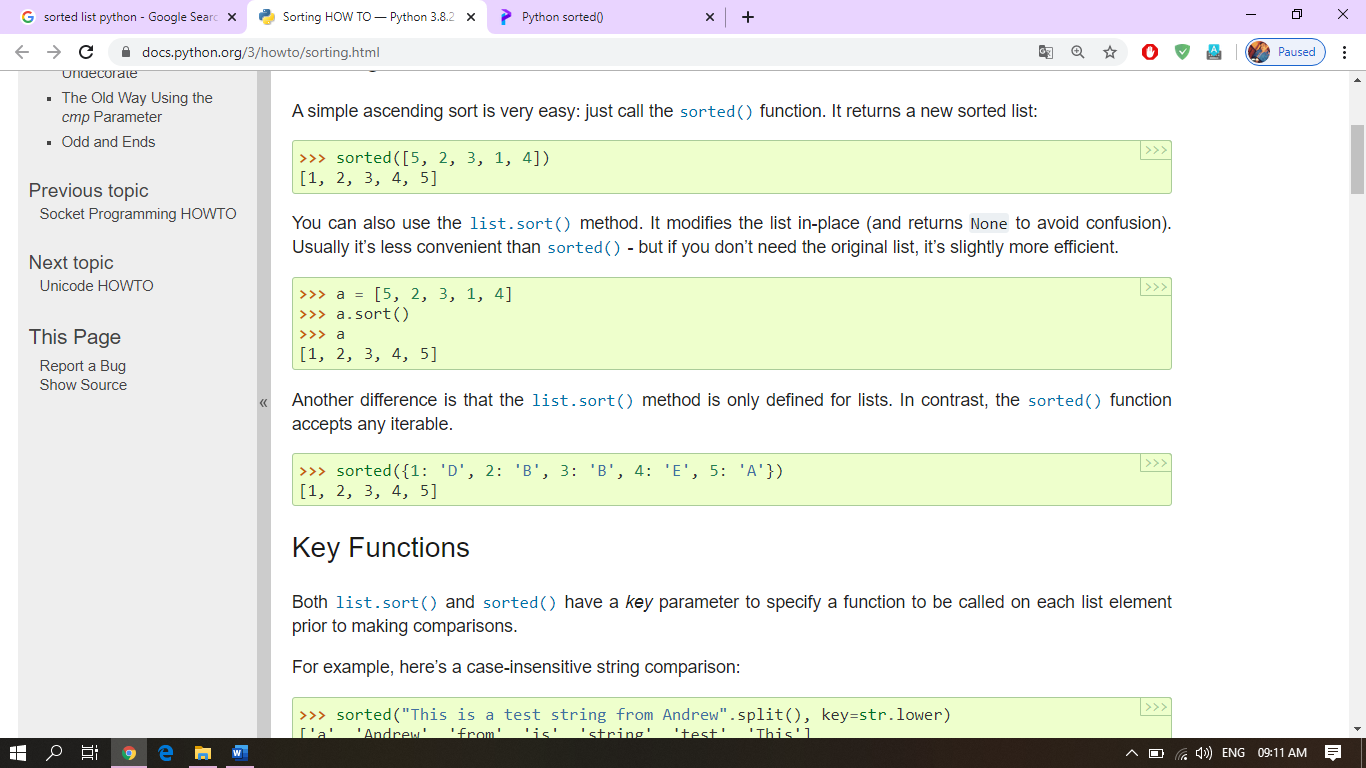
**Sorted list**

Python lists have a built-in [list.sort()](https://docs.python.org/3/library/stdtypes.html" \l "list.sort" \o "list.sort) method that modifies the list in-place. There is also a [sorted()](https://docs.python.org/3/library/functions.html#sorted) built-in function that builds a new sorted list from an iterable.

A simple ascending sort is very easy: just call the [sorted()](https://docs.python.org/3/library/functions.html#sorted) function. It returns a new sorted list:



you can also use the [list.sort()](https://docs.python.org/3/library/stdtypes.html" \l "list.sort" \o "list.sort) method. It modifies the list in-place (and returns None to avoid confusion). Usually it’s less convenient than [sorted()](https://docs.python.org/3/library/functions.html#sorted) - but if you don’t need the original list, it’s slightly more efficient.



# Python Tuple

The Python Tuple is almost similar to a List except that the Tuples are immutable, and Lists are mutable. It means Once we declare the Python Tuple, we cannot change the values or items inside the Tuple, something like Constant keyword in other programming languages.

A Python Tuple is a sequence of multiple values in an ordered sequence. Tuples are declared using Open and Closed Parenthesis ***( )***. Unlike [Python](https://www.tutorialgateway.org/python-tutorial/) Strings, Tuple allows us to store different types of data such as integer, float, string, etc.

## How to Declare Python Tuple?

The following are the list of possible ways to declare a Tuple in Python.

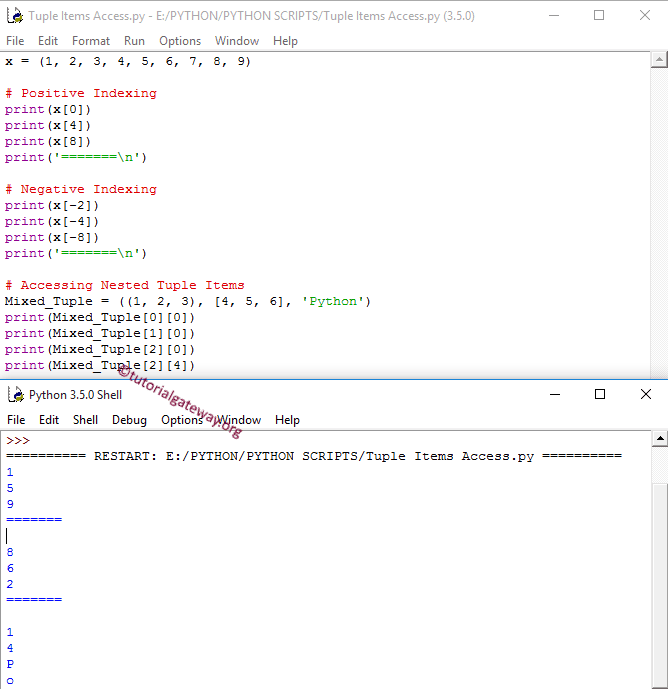
Tuple\_Name = () is an empty Tuple that contains no values.

String\_Tuple = (‘apple’, ‘Orange’, ‘Grape’, ‘Mango’) is a string tuple that contains four string values.

Mixed\_Tuple = (‘apple’, 2, 3.50, ‘Mango’) is a mixed tuple that contains one integer, one float, and two integer values.

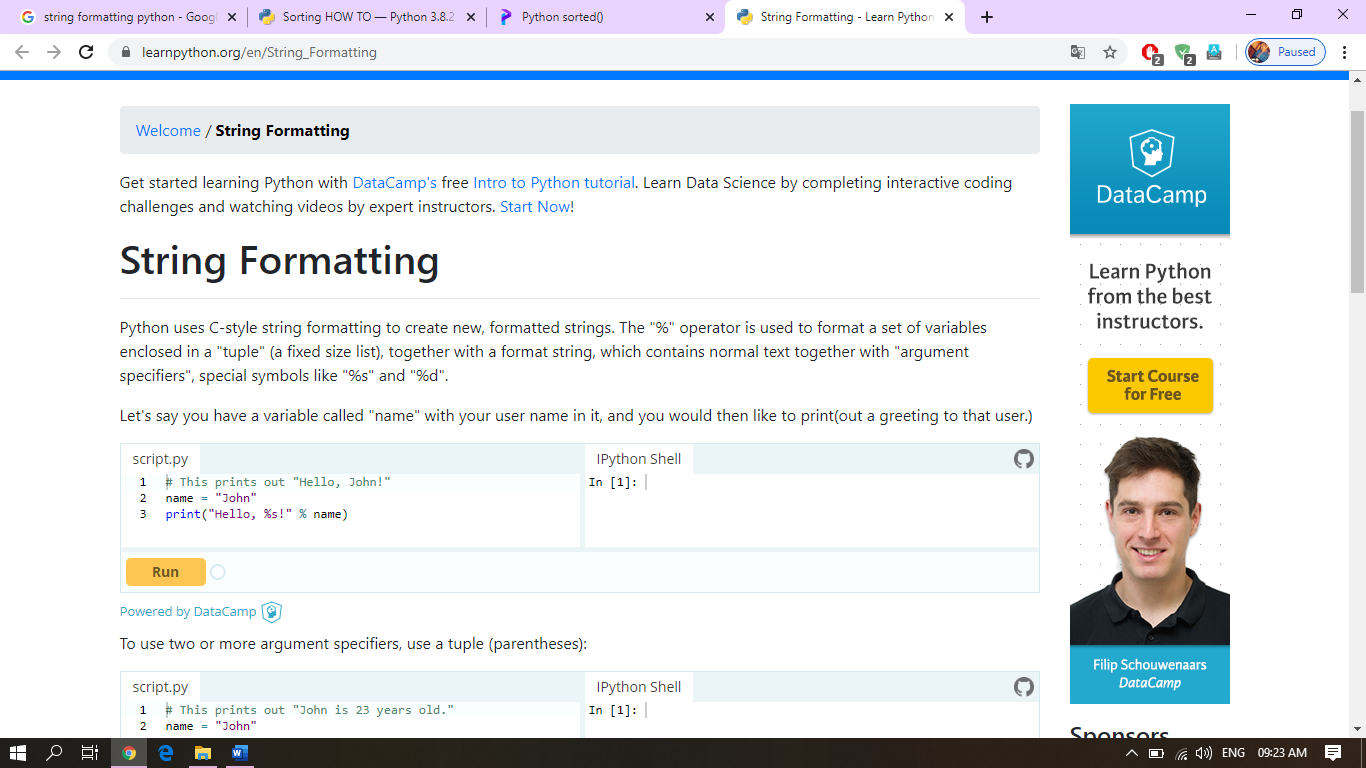
Nested\_Tuple = (‘Python’, ‘Tutorial’, (1, 2, 3) ) is an example of tuple inside another tuple (Nested Tuple)

List\_Tuple = (‘Python’, ‘Tutorial’, [1, 2, 3] ) is an example of List inside a tuple



# String Formatting

Python uses C-style string formatting to create new, formatted strings. The "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list), together with a format string, which contains normal text together with "argument specifiers", special symbols like "%s" and "%d".



# [sets](https://docs.python.org/2/library/sets.html#module-sets) — Unordered collections of unique elements

The [**sets**](https://docs.python.org/2/library/sets.html#module-sets) module provides classes for constructing and manipulating unordered collections of unique elements. Common uses include membership testing, removing duplicates from a sequence, and computing standard math operations on sets such as intersection, union, difference, and symmetric difference.

Like other collections, sets support x in set, len(set), and for x in set. Being an unordered collection, sets do not record element position or order of insertion. Accordingly, sets do not support indexing, slicing, or other sequence-like behavior.

Most set applications use the [**Set**](https://docs.python.org/2/library/sets.html#sets.Set) class which provides every set method except for [**\_\_hash\_\_()**](https://docs.python.org/2/reference/datamodel.html#object.__hash__). For advanced applications requiring a hash method, the **[ImmutableSet](https://docs.python.org/2/library/sets.html" \l "sets.ImmutableSet" \o "sets.ImmutableSet)** class adds a [**\_\_hash\_\_()**](https://docs.python.org/2/reference/datamodel.html#object.__hash__) method but omits methods which alter the contents of the set. Both [**Set**](https://docs.python.org/2/library/sets.html#sets.Set) and **[ImmutableSet](https://docs.python.org/2/library/sets.html" \l "sets.ImmutableSet" \o "sets.ImmutableSet)** derive from **BaseSet**, an abstract class useful for determining whether something is a set: isinstance(obj, BaseSet).

The set classes are implemented using dictionaries. Accordingly, the requirements for set elements are the same as those for dictionary keys; namely, that the element defines both [**\_\_eq\_\_()**](https://docs.python.org/2/reference/datamodel.html#object.__eq__) and [**\_\_hash\_\_()**](https://docs.python.org/2/reference/datamodel.html#object.__hash__). As a result, sets cannot contain mutable elements such as lists or dictionaries. However, they can contain immutable collections such as tuples or instances of **[ImmutableSet](https://docs.python.org/2/library/sets.html" \l "sets.ImmutableSet" \o "sets.ImmutableSet)**. For convenience in implementing sets of sets, inner sets are automatically converted to immutable form, for example, Set([Set(['dog'])]) is transformed to Set([ImmutableSet(['dog'])]).

*class*sets.**Set**([*iterable*])

Constructs a new empty [**Set**](https://docs.python.org/2/library/sets.html#sets.Set) object. If the optional *iterable* parameter is supplied, updates the set with elements obtained from iteration. All of the elements in *iterable* should be immutable or be transformable to an immutable using the protocol described in section [Protocol for automatic conversion to immutable](https://docs.python.org/2/library/sets.html#immutable-transforms).

*class*sets.**ImmutableSet**([*iterable*])

Constructs a new empty **[ImmutableSet](https://docs.python.org/2/library/sets.html" \l "sets.ImmutableSet" \o "sets.ImmutableSet)** object. If the optional *iterable* parameter is supplied, updates the set with elements obtained from iteration. All of the elements in *iterable* should be immutable or be transformable to an immutable using the protocol described in section [Protocol for automatic conversion to immutable](https://docs.python.org/2/library/sets.html#immutable-transforms).

Because **[ImmutableSet](https://docs.python.org/2/library/sets.html" \l "sets.ImmutableSet" \o "sets.ImmutableSet)** objects provide a [**\_\_hash\_\_()**](https://docs.python.org/2/reference/datamodel.html#object.__hash__) method, they can be used as set elements or as dictionary keys. **[ImmutableSet](https://docs.python.org/2/library/sets.html" \l "sets.ImmutableSet" \o "sets.ImmutableSet)** objects do not have methods for adding or removing elements, so all of the elements must be known when the constructor is called.

## 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)

## What are exceptions in Python?

Python has many [built-in exceptions](https://www.programiz.com/python-programming/exceptions) which forces your program to output an error when something in it goes wrong.

When these exceptions occur, it causes the current process to stop and passes it to the calling process until it is handled. If not handled, our program will crash.

For example, if [function](https://www.programiz.com/python-programming/function) A calls function B which in turn calls function C and an exception occurs in function C. If it is not handled in C, the exception passes to B and then to A.

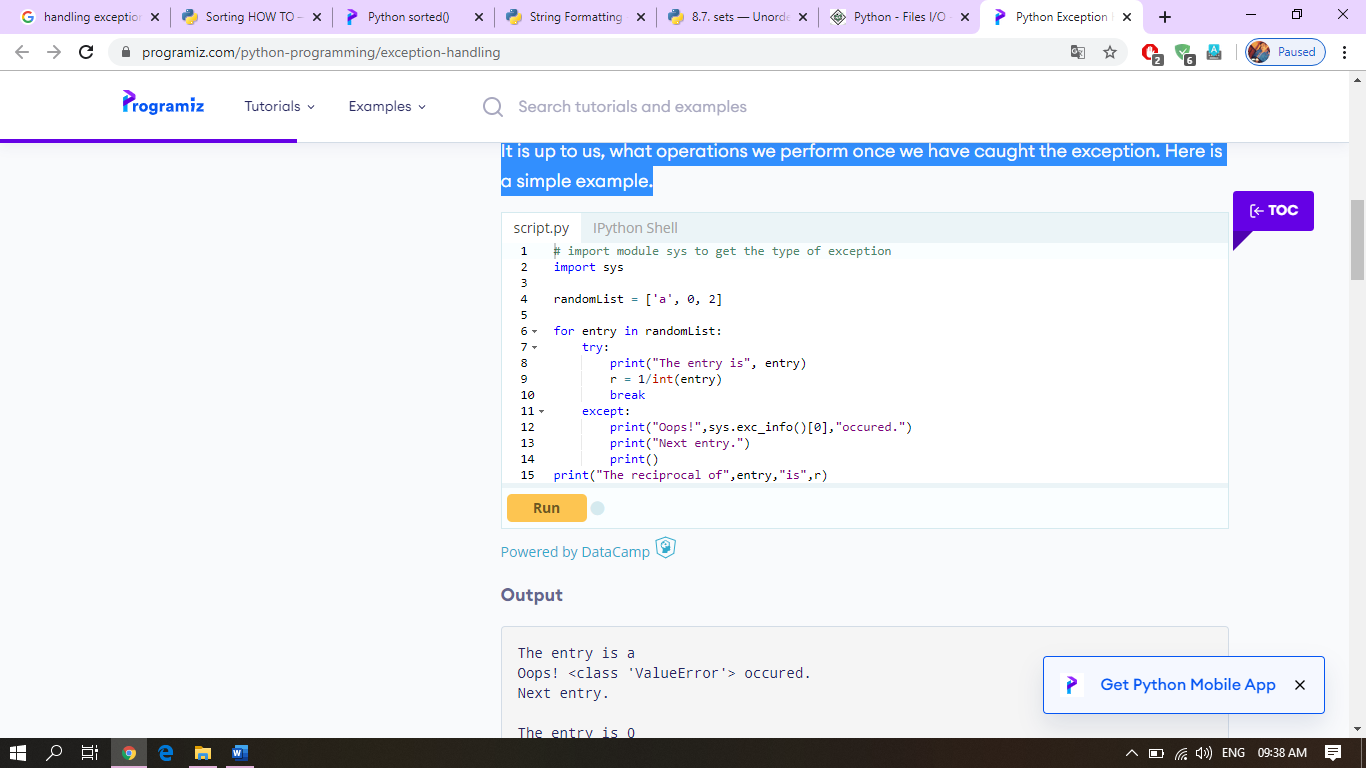
If never handled, an error message is spit out and our program come to a sudden, unexpected halt.

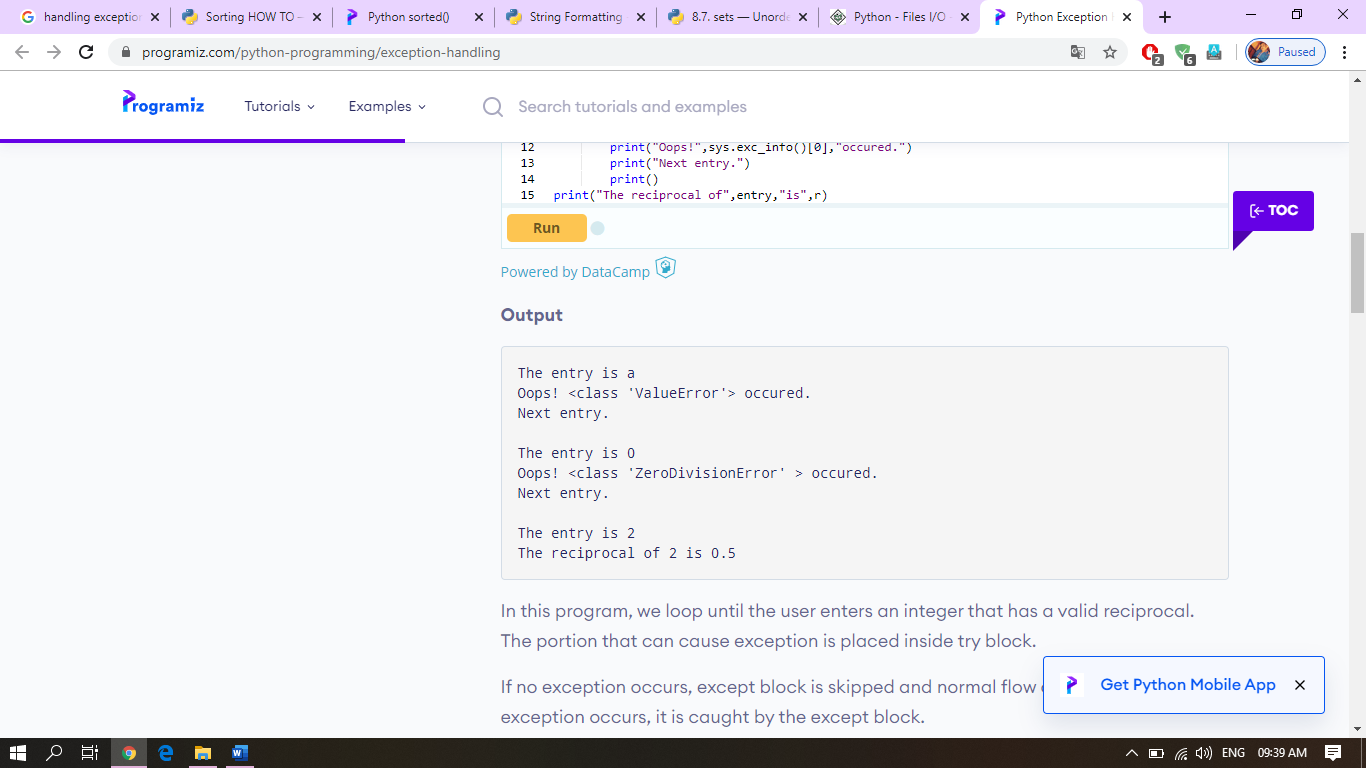
## Catching Exceptions in Python

In Python, exceptions can be handled using a try statement.

A critical operation which can raise exception is placed inside the try clause and the code that handles exception is written in except clause.

It is up to us, what operations we perform once we have caught the exception. Here is a simple example.





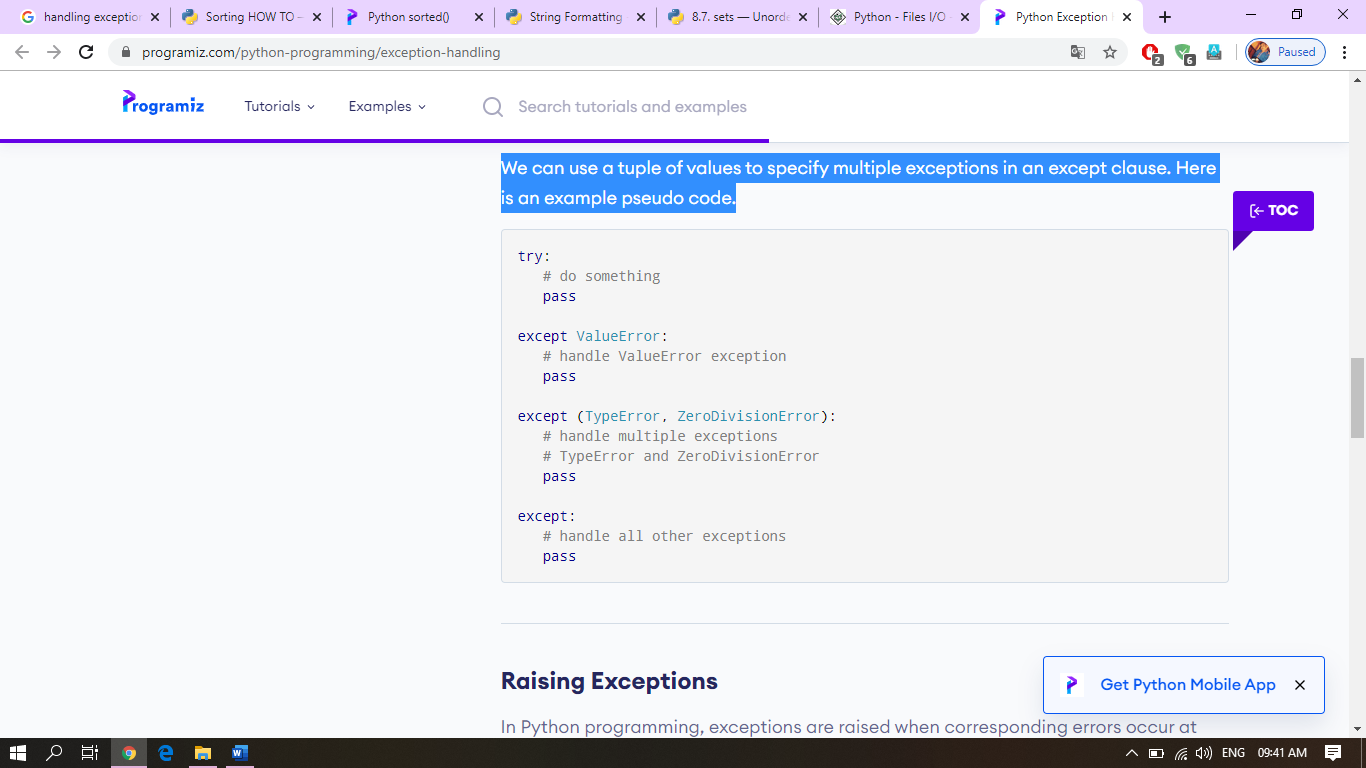
## Catching Specific Exceptions in Python

In the above example, we did not mention any exception in the except clause.

This is not a good programming practice as it will catch all exceptions and handle every case in the same way. We can specify which exceptions an except clause will catch.

A try clause can have any number of except clause to handle them differently but only one will be executed in case an exception occurs.

We can use a tuple of values to specify multiple exceptions in an except clause. Here is an example pseudo code.



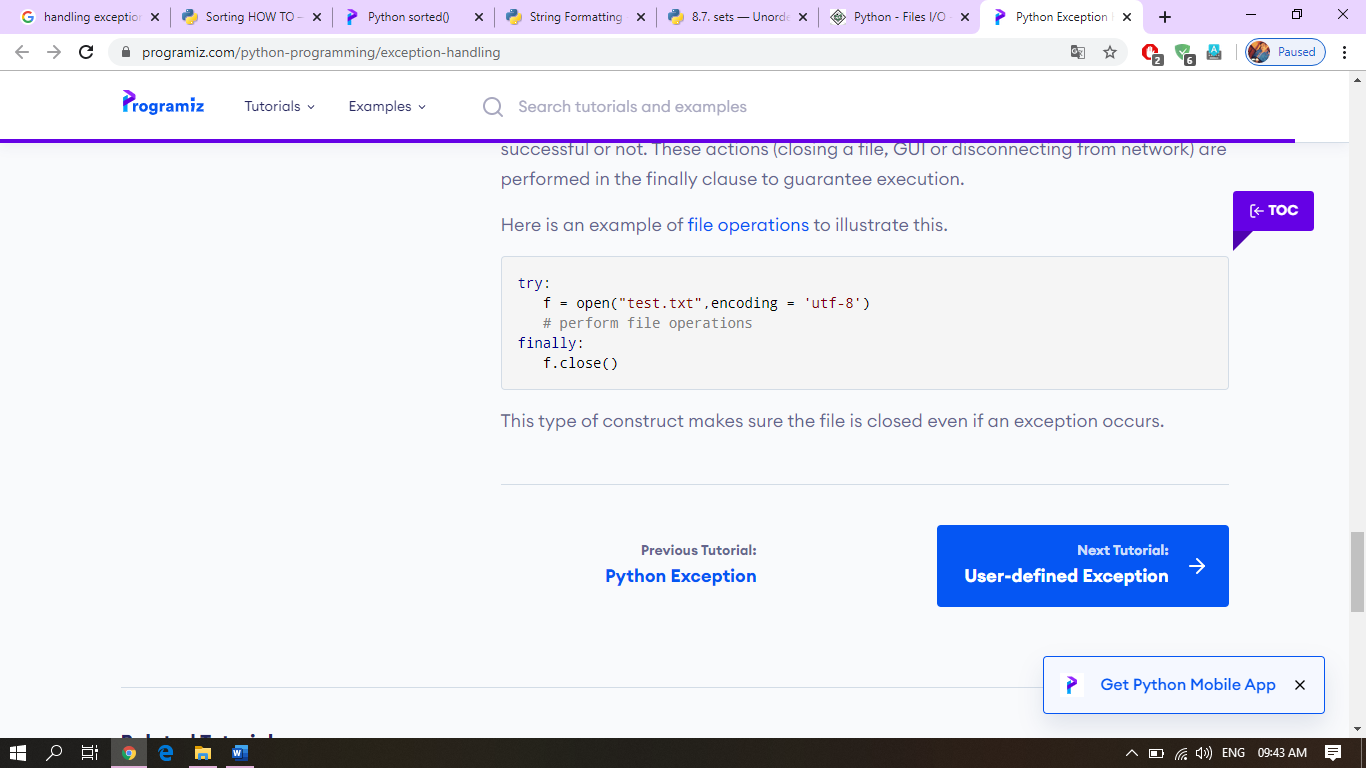
## try...finally

The try statement in Python can have an optional finally clause. This clause is executed no matter what, and is generally used to release external resources.

For example, we may be connected to a remote data center through the network or working with a file or working with a Graphical User Interface (GUI).

In all these circumstances, we must clean up the resource once used, whether it was successful or not. These actions (closing a file, GUI or disconnecting from network) are performed in the finally clause to guarantee execution.

Here is an example of [file operations](https://www.programiz.com/python-programming/file-operation) to illustrate this.



# *Python Dictionary*

**Dictionary**in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds **key:value** pair. Key value is provided in the dictionary to make it more optimized.

**Note –**Keys in a dictionary doesn’t allows Polymorphism.

## Creating a Dictionary

In Python, a Dictionary can be created by placing sequence of elements within curly **{}** braces, separated by ‘comma’. Dictionary holds a pair of values, one being the Key and the other corresponding pair element being its **Key:value**. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can’t be repeated and must be immutable.

**Note –**Dictionary keys are case sensitive, same name but different cases of Key will be treated distinctly.

|  |
| --- |
| # Creating a Dictionary  # with Integer Keys  Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}  print("\nDictionary with the use of Integer Keys: ")  print(Dict)    # Creating a Dictionary  # with Mixed keys  Dict = {'Name': 'Geeks', 1: [1, 2, 3, 4]}  print("\nDictionary with the use of Mixed Keys: ")  print(Dict) |

**Output:**

Dictionary with the use of Integer Keys:

{1: 'Geeks', 2: 'For', 3: 'Geeks'}

Dictionary with the use of Mixed Keys:

{1: [1, 2, 3, 4], 'Name': 'Geeks'}

Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing to curly braces{}.

|  |
| --- |
| # Creating an empty Dictionary  Dict = {}  print("Empty Dictionary: ")  print(Dict)    # Creating a Dictionary  # with dict() method  Dict = dict({1: 'Geeks', 2: 'For', 3:'Geeks'})  print("\nDictionary with the use of dict(): ")  print(Dict)    # Creating a Dictionary  # with each item as a Pair  Dict = dict([(1, 'Geeks'), (2, 'For')])  print("\nDictionary with each item as a pair: ")  print(Dict) |

**Output:**

Empty Dictionary:

{}

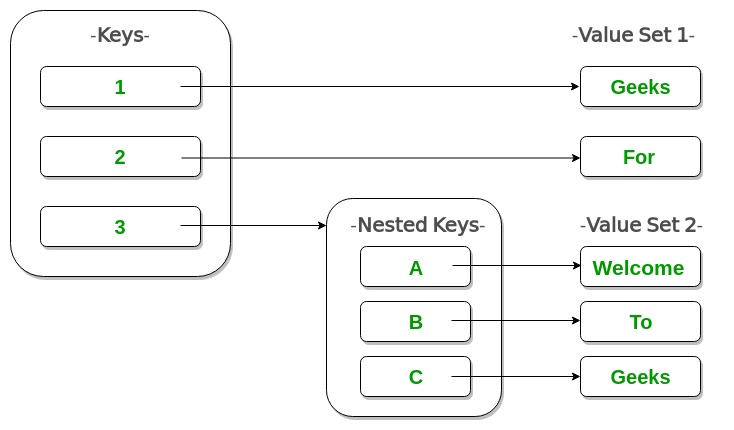
Dictionary with the use of dict():

{1: 'Geeks', 2: 'For', 3: 'Geeks'}

Dictionary with each item as a pair:

{1: 'Geeks', 2: 'For'}

## Nested Dictionary:



|  |
| --- |
| # Creating a Nested Dictionary  # as shown in the below image  Dict = {1: 'Geeks', 2: 'For',          3:{'A' : 'Welcome', 'B' : 'To', 'C' : 'Geeks'}}    print(Dict) |

**Output:**

{1: 'Geeks', 2: 'For', 3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}

## Adding elements to a Dictionary

In Python Dictionary, Addition of elements can be done in multiple ways. One value at a time can be added to a Dictionary by defining value along with the key e.g. Dict[Key] = ‘Value’. Updating an existing value in a Dictionary can be done by using the built-in **update()** method. Nested key values can also be added to an existing Dictionary.  
**Note-** While adding a value, if the key value already exists, the value gets updated otherwise a new Key with the value is added to the Dictionary.

|  |
| --- |
| # Creating an empty Dictionary  Dict = {}  print("Empty Dictionary: ")  print(Dict)    # Adding elements one at a time  Dict[0] = 'Geeks'  Dict[2] = 'For'  Dict[3] = 1  print("\nDictionary after adding 3 elements: ")  print(Dict)    # Adding set of values  # to a single Key  Dict['Value\_set'] = 2, 3, 4  print("\nDictionary after adding 3 elements: ")  print(Dict)    # Updating existing Key's Value  Dict[2] = 'Welcome'  print("\nUpdated key value: ")  print(Dict)    # Adding Nested Key value to Dictionary  Dict[5] = {'Nested' :{'1' : 'Life', '2' : 'Geeks'}}  print("\nAdding a Nested Key: ")  print(Dict) |

**Output:**

Empty Dictionary:

{}

Dictionary after adding 3 elements:

{0: 'Geeks', 2: 'For', 3: 1}

Dictionary after adding 3 elements:

{0: 'Geeks', 2: 'For', 3: 1, 'Value\_set': (2, 3, 4)}

Updated key value:

{0: 'Geeks', 2: 'Welcome', 3: 1, 'Value\_set': (2, 3, 4)}

Adding a Nested Key:

{0: 'Geeks', 2: 'Welcome', 3: 1, 5: {'Nested': {'1': 'Life', '2': 'Geeks'}}, 'Value\_set': (2, 3, 4)}

## Accessing elements from a Dictionary

In order to access the items of a dictionary refer to its key name.Key can be used inside square brackets.

|  |
| --- |
| # Python program to demonstrate  # accessing a element from a Dictionary    # Creating a Dictionary  Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}    # accessing a element using key  print("Accessing a element using key:")  print(Dict['name'])    # accessing a element using key  print("Accessing a element using key:")  print(Dict[1]) |

**Output:**

Accessing a element using key:

For

Accessing a element using key:

Geeks

There is also a method called [**get()**](https://www.geeksforgeeks.org/get-method-dictionaries-python/) that will also help in acessing the element from a dictionary.

|  |
| --- |
| # Creating a Dictionary  Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}    # accessing a element using get()  # method  print("Accessing a element using get:")  print(Dict.get(3)) |

**Output:**

Accessing a element using get:

Geeks

#### Accessing element of a nested dictionary

In order to access the value of any key in nested dictionary, use indexing [] syntax.

|  |
| --- |
| # Creating a Dictionary  Dict = {'Dict1': {1: 'Geeks'},          'Dict2': {'Name': 'For'}}    # Accessing element using key  print(Dict['Dict1'])  print(Dict['Dict1'][1])  print(Dict['Dict2']['Name']) |

**Output:**

{1: 'Geeks'}

Geeks

For

## Removing Elements from Dictionary

#### Using del keyword

In Python Dictionary, deletion of keys can be done by using the **del**keyword. Using del keyword, specific values from a dictionary as well as whole dictionary can be deleted. Items in a Nested dictionary can also be deleted by using del keyword and providing specific nested key and particular key to be deleted from that nested Dictionary.

**Note- del Dict** will delete the entire dictionary and hence printing it after deletion will raise an Error.

|  |
| --- |
| # Initial Dictionary  Dict = { 5 : 'Welcome', 6 : 'To', 7 : 'Geeks',          'A' : {1 : 'Geeks', 2 : 'For', 3 : 'Geeks'},          'B' : {1 : 'Geeks', 2 : 'Life'}}  print("Initial Dictionary: ")  print(Dict)    # Deleting a Key value  del Dict[6]  print("\nDeleting a specific key: ")  print(Dict)    # Deleting a Key from  # Nested Dictionary  del Dict['A'][2]  print("\nDeleting a key from Nested Dictionary: ")  print(Dict) |

**Output:**

Initial Dictionary:

{'A': {1: 'Geeks', 2: 'For', 3: 'Geeks'}, 'B': {1: 'Geeks', 2: 'Life'}, 5: 'Welcome', 6: 'To', 7: 'Geeks'}

Deleting a specific key:

{'A': {1: 'Geeks', 2: 'For', 3: 'Geeks'}, 'B': {1: 'Geeks', 2: 'Life'}, 5: 'Welcome', 7: 'Geeks'}

Deleting a key from Nested Dictionary:

{'A': {1: 'Geeks', 3: 'Geeks'}, 'B': {1: 'Geeks', 2: 'Life'}, 5: 'Welcome', 7: 'Geeks'}

#### Using pop() method

[Pop(](https://www.geeksforgeeks.org/python-dictionary-pop-method/)) method is used to return and delete the value of the key specified.

|  |
| --- |
| # Creating a Dictionary  Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}    # Deleting a key  # using pop() method  pop\_ele = Dict.pop(1)  print('\nDictionary after deletion: ' + str(Dict))  print('Value associated to poped key is: ' + str(pop\_ele)) |

**Output:**

Dictionary after deletion: {3: 'Geeks', 'name': 'For'}

Value associated to poped key is: Geeks

#### Using popitem() method

The popitem() returns and removes an arbitrary element (key, value) pair from the dictionary.

|  |
| --- |
| # Creating Dictionary  Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}    # Deleting an arbitrary key  # using popitem() function  pop\_ele = Dict.popitem()  print("\nDictionary after deletion: " + str(Dict))  print("The arbitrary pair returned is: " + str(pop\_ele)) |

**Output:**

Dictionary after deletion: {3: 'Geeks', 'name': 'For'}

The arbitrary pair returned is: (1, 'Geeks')

#### Using clear() method

All the items from a dictionary can be deleted at once by using **clear()** method.

|  |
| --- |
| # Creating a Dictionary  Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}      # Deleting entire Dictionary  Dict.clear()  print("\nDeleting Entire Dictionary: ")  print(Dict) |

**Output:**

Deleting Entire Dictionary:

{}

## Dictionary Methods

|  |  |
| --- | --- |
| **METHODS** | **DESCRIPTION** |
| [copy()](https://www.geeksforgeeks.org/python-dictionary-copy/) | They copy() method returns a shallow copy of the dictionary. |
| [clear()](https://www.geeksforgeeks.org/python-dictionary-clear/) | The clear() method removes all items from the dictionary. |
| [pop()](https://www.geeksforgeeks.org/python-dictionary-pop-method/) | Removes and returns an element from a dictionary having the given key. |
| [popitem()](https://www.geeksforgeeks.org/python-dictionary-popitem-method/) | Removes the arbitrary key-value pair from the dictionary and returns it as tuple. |
| [get()](https://www.geeksforgeeks.org/get-method-dictionaries-python/) | It is a conventional method to access a value for a key. |
| [dictionary\_name.values()](https://www.geeksforgeeks.org/python-dictionary-values/) | returns a list of all the values available in a given dictionary. |
| str() | Produces a printable string representation of a dictionary. |
| [update()](https://www.geeksforgeeks.org/python-dictionary-update-method/) | Adds dictionary dict2’s key-values pairs to dict |
| [setdefault()](https://www.geeksforgeeks.org/python-dictionary-setdefault-method/) | Set dict[key]=default if key is not already in dict |
| [keys()](https://www.geeksforgeeks.org/python-dictionary-keys-method/) | Returns list of dictionary dict’s keys |
| [items()](https://www.geeksforgeeks.org/python-dictionary-items-method/) | Returns a list of dict’s (key, value) tuple pairs |
| [has\_key()](https://www.geeksforgeeks.org/python-dictionary-has_key/) | Returns true if key in dictionary dict, false otherwise |
| [fromkeys()](https://www.geeksforgeeks.org/python-dictionary-fromkeys-method/) | Create a new dictionary with keys from seq and values set to value. |
| [type()](https://www.geeksforgeeks.org/python-type-function/) | Returns the type of the passed variable. |
| [cmp()](https://www.geeksforgeeks.org/dictionary-methods-in-python-set-1-cmp-len-items/) | Compares elements of both dict. |

Python | Merging two Dictionaries

There are various ways in which Dictionaries can be merged by the use of various functions and constructors in Python. In this article, we will discuss few ways of merging dictionaries.

1. **Using the method update()**  
   By using the method update() in Python, one list can be merged into another. But in this, the second list is merged into the first list and no new list is created. It returns *None*.  
   Example:

|  |
| --- |
| # Python code to merge dict using update() method  def Merge(dict1, dict2):      return(dict2.update(dict1))    # Driver code  dict1 = {'a': 10, 'b': 8}  dict2 = {'d': 6, 'c': 4}    # This return None  print(Merge(dict1, dict2))    # changes made in dict2  print(dict2) |

Output:

None

{'c': 4, 'a': 10, 'b': 8, 'd': 6}

1. **Using**[**\*\* in Python**](https://www.geeksforgeeks.org/args-kwargs-python/)  
   This is generally considered a trick in Python where a single expression is used to merge two dictionaries and stored in a third dictionary. The single expression is \*\*. This does not affect the other two dictionaries. \*\* implies that the argument is a dictionary. Using \*\* [double star] is a shortcut that allows you to pass multiple arguments to a function directly using a dictionary. Using this we first pass all the elements of the first dictionary into the third one and then pass the second dictionary into the third. This will replace the duplicate keys of the first dictionary.  
   Example:

|  |
| --- |
| # Python code to merge dict using a single  # expression  def Merge(dict1, dict2):      res = {\*\*dict1, \*\*dict2}      return res    # Driver code  dict1 = {'a': 10, 'b': 8}  dict2 = {'d': 6, 'c': 4}  dict3 = Merge(dict1, dict2)  print(dict3) |

Output:

{'b': 8, 'a': 10, 'c': 4, 'd': 6}

**File Handling in Python**

Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but alike other concepts of Python, this concept here is also easy and short. Python treats file differently as text or binary and this is important. Each line of code includes a sequence of characters and they form text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun. Let’s start with Reading and Writing files.

**Working of open() function**

We use **open ()** function in Python to open a file in read or write mode. As explained above, open ( ) will return a file object. To return a file object we use **open()** function along with two arguments, that accepts file name and the mode, whether to read or write. So, the syntax being: **open(filename, mode)**. There are three kinds of mode, that Python provides and how files can be opened:

* “**r**“, for reading.
* “**w**“, for writing.
* “**a**“, for appending.
* “**r+**“, for both reading and writing

One must keep in mind that the mode argument is not mandatory. If not passed, then Python will assume it to be “**r**” by default. Let’s look at this program and try to analyze how the read mode works:

|  |
| --- |
| # a file named "geek", will be opened with the reading mode.  file = open('geek.txt', 'r')  # This will print every line one by one in the file  for each in file:      print (each) |

The open command will open the file in the read mode and the for loop will print each line present in the file.

**Working of read() mode**

There is more than one way to read a file in Python. If you need to extract a string that contains all characters in the file then we can use **file.read()**. The full code would work like this:

|  |
| --- |
| # Python code to illustrate read() mode  file = open("file.text", "r")  print file.read() |

Another way to read a file is to call a certain number of characters like in the following code the interpreter will read the first five characters of stored data and return it as a string:

|  |
| --- |
| # Python code to illustrate read() mode character wise  file = open("file.txt", "r")  print file.read(5) |

**Creating a file using write() mode**

Let’s see how to create a file and how write mode works:  
To manipulate the file, write the following in your Python environment:

|  |
| --- |
| # Python code to create a file  file = open('geek.txt','w')  file.write("This is the write command")  file.write("It allows us to write in a particular file")  file.close() |

The close() command terminates all the resources in use and frees the system of this particular program.

**Working of append() mode**

Let’s see how the append mode works:

|  |
| --- |
| # Python code to illustrate append() mode  file = open('geek.txt','a')  file.write("This will add this line")  file.close() |

There are also various other commands in file handling that is used to handle various tasks like:

rstrip(): This function strips each line of a file off spaces from the right-hand side.

lstrip(): This function strips each line of a file off spaces from the left-hand side.

It is designed to provide much cleaner syntax and exceptions handling when you are working with code. That explains why it’s good practice to use them with a statement where applicable. This is helpful because using this method any files opened will be closed automatically after one is done, so auto-cleanup.  
Example:

|  |
| --- |
| # Python code to illustrate with()  with open("file.txt") as file:      data = file.read()  # do something with data |

**Using write along with with() function**

We can also use write function along with with() function:

|  |
| --- |
| # Python code to illustrate with() alongwith write()  with open("file.txt", "w") as f:      f.write("Hello World!!!") |

**split() using file handling**

We can also split lines using file handling in Python. This splits the variable when space is encountered. You can also split using any characters as we wish. Here is the code:

|  |
| --- |
| # Python code to illustrate split() function  with open("file.text", "r") as file:      data = file.readlines()      for line in data:          word = line.split()          print word |
|  |
|  |
|  |
|  |
|  |
|  |
| **Exceptions in Python**  Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them −   * **Exception Handling** − This would be covered in this tutorial. Here is a list standard Exceptions available in Python: [Standard Exceptions](https://www.tutorialspoint.com/python/standard_exceptions.htm). * **Assertions** − This would be covered in [Assertions in Python](https://www.tutorialspoint.com/python/assertions_in_python.htm) tutorial.   List of Standard Exceptions −   |  |  | | --- | --- | | **Sr.No.** | **Exception Name & Description** | | 1 | **Exception**  Base class for all exceptions | | 2 | **StopIteration**  Raised when the next() method of an iterator does not point to any object. | | 3 | **SystemExit**  Raised by the sys.exit() function. | | 4 | **StandardError**  Base class for all built-in exceptions except StopIteration and SystemExit. | | 5 | **ArithmeticError**  Base class for all errors that occur for numeric calculation. | | 6 | **OverflowError**  Raised when a calculation exceeds maximum limit for a numeric type. | | 7 | **FloatingPointError**  Raised when a floating point calculation fails. | | 8 | **ZeroDivisionError**  Raised when division or modulo by zero takes place for all numeric types. | | 9 | **AssertionError**  Raised in case of failure of the Assert statement. | | 10 | **AttributeError**  Raised in case of failure of attribute reference or assignment. | | 11 | **EOFError**  Raised when there is no input from either the raw\_input() or input() function and the end of file is reached. | | 12 | **ImportError**  Raised when an import statement fails. | | 13 | **KeyboardInterrupt**  Raised when the user interrupts program execution, usually by pressing Ctrl+c. | | 14 | **LookupError**  Base class for all lookup errors. | | 15 | **IndexError**  Raised when an index is not found in a sequence. | | 16 | **KeyError**  Raised when the specified key is not found in the dictionary. | | 17 | **NameError**  Raised when an identifier is not found in the local or global namespace. | | 18 | **UnboundLocalError**  Raised when trying to access a local variable in a function or method but no value has been assigned to it. | | 19 | **EnvironmentError**  Base class for all exceptions that occur outside the Python environment. | | 20 | **IOError**  Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist. | | 21 | **IOError**  Raised for operating system-related errors. | | 22 | **SyntaxError**  Raised when there is an error in Python syntax. | | 23 | **IndentationError**  Raised when indentation is not specified properly. | | 24 | **SystemError**  Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit. | | 25 | **SystemExit**  Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit. | | 26 | **TypeError**  Raised when an operation or function is attempted that is invalid for the specified data type. | | 27 | **ValueError**  Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified. | | 28 | **RuntimeError**  Raised when a generated error does not fall into any category. | | 29 | **NotImplementedError**  Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented. |  **Assertions in Python** An assertion is a sanity-check that you can turn on or turn off when you are done with your testing of the program.  The easiest way to think of an assertion is to liken it to a **raise-if** statement (or to be more accurate, a raise-if-not statement). An expression is tested, and if the result comes up false, an exception is raised.  Assertions are carried out by the assert statement, the newest keyword to Python, introduced in version 1.5.  Programmers often place assertions at the start of a function to check for valid input, and after a function call to check for valid output. **The *assert* Statement** When it encounters an assert statement, Python evaluates the accompanying expression, which is hopefully true. If the expression is false, Python raises an *AssertionError* exception.  The **syntax** for assert is −  assert Expression[, Arguments]  If the assertion fails, Python uses ArgumentExpression as the argument for the AssertionError. AssertionError exceptions can be caught and handled like any other exception using the try-except statement, but if not handled, they will terminate the program and produce a traceback. **Example** Here is a function that converts a temperature from degrees Kelvin to degrees Fahrenheit. Since zero degrees Kelvin is as cold as it gets, the function bails out if it sees a negative temperature −  #!/usr/bin/python  def KelvinToFahrenheit(Temperature):  assert (Temperature >= 0),"Colder than absolute zero!"  return ((Temperature-273)\*1.8)+32  print KelvinToFahrenheit(273)  print int(KelvinToFahrenheit(505.78))  print KelvinToFahrenheit(-5)  When the above code is executed, it produces the following result −  32.0  451  Traceback (most recent call last):  File "test.py", line 9, in <module>  print KelvinToFahrenheit(-5)  File "test.py", line 4, in KelvinToFahrenheit  assert (Temperature >= 0),"Colder than absolute zero!"  AssertionError: Colder than absolute zero! What is Exception? An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.  When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits. Handling an exception If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible. **Syntax** Here is simple syntax of *try....except...else* blocks −  try:  You do your operations here;  ......................  except *ExceptionI*:  If there is ExceptionI, then execute this block.  except *ExceptionII*:  If there is ExceptionII, then execute this block.  ......................  else:  If there is no exception then execute this block.  Here are few important points about the above-mentioned syntax −   * A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions. * You can also provide a generic except clause, which handles any exception. * After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception. * The else-block is a good place for code that does not need the try: block's protection.  **Example** This example opens a file, writes content in the, file and comes out gracefully because there is no problem at all −  #!/usr/bin/python  try:  fh = open("testfile", "w")  fh.write("This is my test file for exception handling!!")  except IOError:  print "Error: can\'t find file or read data"  else:  print "Written content in the file successfully"  fh.close()  This produces the following result −  Written content in the file successfully **Example** This example tries to open a file where you do not have write permission, so it raises an exception −  #!/usr/bin/python  try:  fh = open("testfile", "r")  fh.write("This is my test file for exception handling!!")  except IOError:  print "Error: can\'t find file or read data"  else:  print "Written content in the file successfully"  This produces the following result −  Error: can't find file or read data The *except* Clause with No Exceptions You can also use the except statement with no exceptions defined as follows −  try:  You do your operations here;  ......................  except:  If there is any exception, then execute this block.  ......................  else:  If there is no exception then execute this block.  This kind of a **try-except** statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur. The *except* Clause with Multiple Exceptions You can also use the same *except* statement to handle multiple exceptions as follows −  try:  You do your operations here;  ......................  except(Exception1[, Exception2[,...ExceptionN]]]):  If there is any exception from the given exception list,  then execute this block.  ......................  else:  If there is no exception then execute this block. The try-finally Clause You can use a **finally:** block along with a **try:** block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not. The syntax of the try-finally statement is this −  try:  You do your operations here;  ......................  Due to any exception, this may be skipped.  finally:  This would always be executed.  ......................  You cannot use *else* clause as well along with a finally clause. **Example** #!/usr/bin/python  try:  fh = open("testfile", "w")  fh.write("This is my test file for exception handling!!")  finally:  print "Error: can\'t find file or read data"  If you do not have permission to open the file in writing mode, then this will produce the following result −  Error: can't find file or read data  Same example can be written more cleanly as follows −  #!/usr/bin/python  try:  fh = open("testfile", "w")  try:  fh.write("This is my test file for exception handling!!")  finally:  print "Going to close the file"  fh.close()  except IOError:  print "Error: can\'t find file or read data"  When an exception is thrown in the *try* block, the execution immediately passes to the *finally* block. After all the statements in the *finally* block are executed, the exception is raised again and is handled in the *except* statements if present in the next higher layer of the *try-except* statement. Argument of an Exception An exception can have an *argument*, which is a value that gives additional information about the problem. The contents of the argument vary by exception. You capture an exception's argument by supplying a variable in the except clause as follows −  try:  You do your operations here;  ......................  except *ExceptionType, Argument*:  You can print value of Argument here...  If you write the code to handle a single exception, you can have a variable follow the name of the exception in the except statement. If you are trapping multiple exceptions, you can have a variable follow the tuple of the exception.  This variable receives the value of the exception mostly containing the cause of the exception. The variable can receive a single value or multiple values in the form of a tuple. This tuple usually contains the error string, the error number, and an error location. **Example** Following is an example for a single exception −  #!/usr/bin/python  # Define a function here.  def temp\_convert(var):  try:  return int(var)  except ValueError, Argument:  print "The argument does not contain numbers\n", Argument  # Call above function here.  temp\_convert("xyz");  This produces the following result −  The argument does not contain numbers  invalid literal for int() with base 10: 'xyz' Raising an Exceptions You can raise exceptions in several ways by using the raise statement. The general syntax for the **raise** statement is as follows. **Syntax** raise [Exception [, args [, traceback]]]  Here, *Exception* is the type of exception (for example, NameError) and *argument* is a value for the exception argument. The argument is optional; if not supplied, the exception argument is None.  The final argument, traceback, is also optional (and rarely used in practice), and if present, is the traceback object used for the exception. **Example** An exception can be a string, a class or an object. Most of the exceptions that the Python core raises are classes, with an argument that is an instance of the class. Defining new exceptions is quite easy and can be done as follows −  def functionName( level ):  if level < 1:  raise "Invalid level!", level  # The code below to this would not be executed  # if we raise the exception  **Note:** In order to catch an exception, an "except" clause must refer to the same exception thrown either class object or simple string. For example, to capture above exception, we must write the except clause as follows −  try:  Business Logic here...  except "Invalid level!":  Exception handling here...  else:  Rest of the code here... User-Defined Exceptions Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.  Here is an example related to *RuntimeError*. Here, a class is created that is subclassed from *RuntimeError*. This is useful when you need to display more specific information when an exception is caught.  In the try block, the user-defined exception is raised and caught in the except block. The variable e is used to create an instance of the class *Networkerror*.  class Networkerror(RuntimeError):  def \_\_init\_\_(self, arg):  self.args = arg  So once you defined above class, you can raise the exception as follows −  try:  raise Networkerror("Bad hostname")  except Networkerror,e:  print e.args |