Linux Commands, C, C++, Java and Python Exercises For Beginners

(Practical Exercises For Learners)

"The only true wisdom is in knowing you know nothing."

− **Socrates**

Manjunath.R

#16/1, 8th Main Road, Shivanagar, Rajajinagar, Bangalore560010, Karnataka, India

*Email: manjunath5496@gmail.com

Disclaimer

Dedication

I **dedicate this book** to every individual, programmer, teacher, educational institutions and enterprise corporations in every country of the world for their immense contributions towards the process of creating, designing, deploying and supporting software…

Acknowledgements

Without the amazing work of some renowned programmers, their creativity, and their inventiveness in the field of software programming, this book would not have been accomplished. I would like to use this opportunity to thank my dearest friend and well-wisher "**Lawrence**" for his unwavering support during the **COVID crisis** and for giving me access to all the resources I needed to finish this book. I want to express my gratitude to my family for their support and encouragement as I wrote this book, especially to my **mother**, who has been a tremendous source of inspiration in my life. I owe a lot of gratitude to my mother for teaching me how to be perseverant and strong in life. Finally, I want to emphasize how crucial patience is when writing a book or taking on any other project in life.

Foreword

I'm neither the proprietor of a well-known publishing house or a top IT firm with hundreds of in-house programmers who could easily produce anything I needed. I am a self-employed software engineer who is passionate about what I do, and believe me when I say that a lot of work and effort went into compiling this **comprehensive edition**. I'll be overjoyed if it helps even a few others reach their ideal positions in their professions.

Thank You

− Manjunath.R

An Enjoyable Introduction to Coding
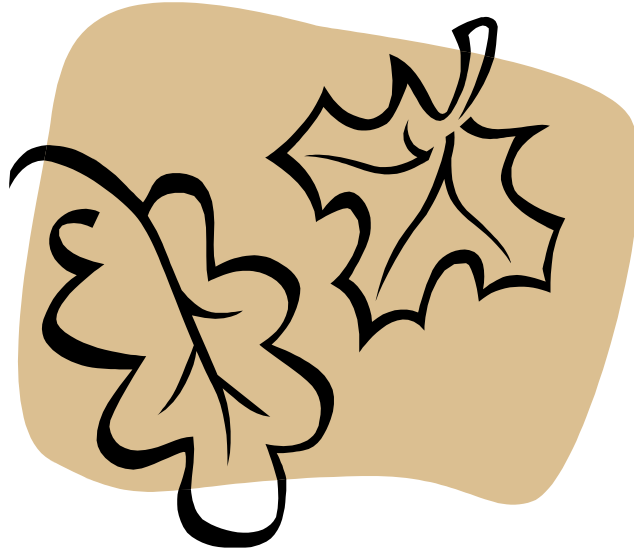
Introduction

Today's devices are mostly powered by software: almost everyone uses Facebook, WhatsApp and Twitter to communicate, many phones have internet-connected desktops, and the majority of office work requires using a computer to do tasks. As a response, there is a huge increase in demand for programmers. **Numerous books**, interactive websites, and programmer training courses make the bold claim that they can turn ambitious novices into software engineers earning six figures. This book is for all programmers, whether you are a novice or an experienced pro. Its numerous examples and **well paced discussions** will be especially beneficial for beginners. Those who are already experienced with programming will probably gain more from this book, of course. You will be at a modest level of programming proficiency when you have finished this book, from where you can take yourself to next levels so that you can automate simple tasks such as:

- Making a file backup
- Get rid of the irritating emails
- Completing online forms

This book will make an amazing complement to any tutorial and serve as a source of information to your specific inquiries if you are just learning what kind of animals C, C+ +, Java, PHP, Python, and JavaScript are. Even if your career has nothing to do with computers, the skills you learn from programming can be valuable at school and at work. **Programming** is a pleasant, occasionally difficult and perhaps frustrating activity. Creativity, logic, and problem-solving are all enhanced through programming.

- Educational institutions are teaching it
- Corporate societies are employing it
- Pupils need it
- (Pedagogues desire it... ;)
- (Coders perceive it... :)

Have Fun!

As you progress through this book, keep in mind that programming can be enjoyable. Do not consider of this as work. Consider programming as a means to develop entertaining games or software applications that you can show off to others or your friends. Programming is a tremendous brain workout and is essential today because so much of our everyday world is automated. But above all, you have access to the quick-paced, creative world that depends on machine connections.

"The only way to learn a new programming language is by writing programs in it."

– Dennis Ritchie
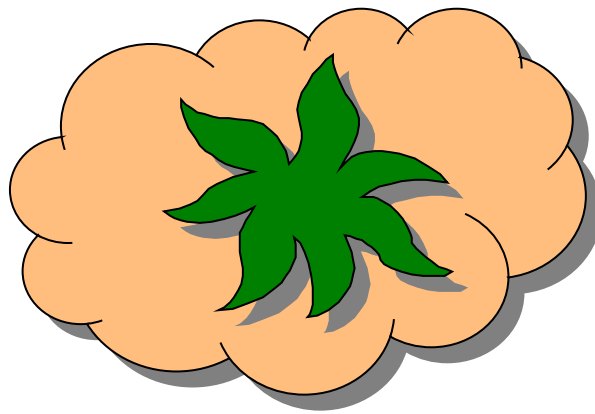
Note:

- **Linux version used:** CentOS Linux release 7.3.1611 (Core)
- **Python version used:** 3.7.3

The Basic Programming Principles That Every Programmer Should Know:

1. Always be aware of the purpose of your software program before beginning to write it.
2. Programming is not the solution; it is merely a means to achieve a solution.
3. Consider the problem rather than just the solution.
4. Always try to make things simpler; anyone can come up with a complicated answer to a problem. To make a solution simple while remaining consistent, it requires extra work and consideration.
5. Reduce Deeply Nested Ifs or Loops: When your software program is deeply nested, your program becomes complicated and disorganized.
6. Delete Unnecessary Code. Make sure your software program is safe, secure, reliable, testable and clear to read.
7. Give code reviews some attention so you can spot bugs early, before they cause serious problems in your software application.
8. Reduce complexity. Software programs must have clear explanations.
9. Generalize your software program. Make sure your software program is documented. Understanding the function of a certain component of the software application is greatly aided by the documentation and comments.

10. Fancy algorithms and data structures are more difficult to implement. Use simple, efficient, appropriate algorithms and data structures.

11. Refractor your software program frequently to improve its internal software attributes in terms of upkeep, testing and comprehension.

12. Each time you make a change to your software program: check it, build it and test it.

13. Before being released, all software codes must pass each and every unit test.

14. Always use caution when using someone else's code. Maintain a standardized, orderly and generally consistent coding style.

15. Avoid implementing a code style that is too hard to understand.

16. Because it makes the code more difficult to maintain, duplication is seen adversely in software programming.

17. Look for bugs and flaws and fix them. Divide your software program into Brief, Concise Units.

18. Avoid overdesigning. Focus your software design on the requirements of the clients.

19. Program defensively. Functions should be simple and do a distinct, defined task.

20. Create reusable functions and Keep the functions as simple, immutable and manageable as possible.

21. When naming your variables and functions, choose names that are meaningful and descriptive.

22. Put your software program's structure on view by using indentation.

23. Delete any unused variables and functions; do not comment them.

24. If you feel that a part of the software program is excessively unorganized, regroup and modify it, or even split it up into different portions.

25. Avoid using GOTO statements because they cause the software program to be unstructured, which makes it harder to understand and makes debugging more complex.

26. Avoid using the same identifier more than once.

27. The length of functions shouldn't be excessively long.

28. Think Twice, Code Once: Encourage yourself to consider the problem more before coming up with a solution.

29. The very first step in making a software program readable by humans is to add comments. Comments should be detailed explanations of a software program.

30. White space should be utilized regularly to increase code readability even though it has little significance to compilers.

31. Coding standards must be followed while formatting code.

32. Avoid security pitfalls and Keep your software code portable.

33. All software design is redesign. Take advice from others' experience.

34. The writing of software program should make it simple for a future software developer to correct errors or modify its functionality.

35. Never compromise clarity for a false sense of efficiency.

"More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason − including blind stupidity."

− W.A. Wulf

36. Enhance the appearance of software program by avoiding excessively long names or ambiguous acronyms

37. Look for a method that employs a loop rather than duplicating lines. Compared to 100 individual blocks of code, one loop that can handle 100 repetitions is simpler to debug.

You're not coding to amaze strangers. You're in this profession to find ways to resolve problems.

Important Programming Concepts Every Programmers and Developer Should Be Familiar With:


1. **Data Types:** Understanding the various data types such as integers, floating-point numbers, characters, and Booleans is crucial for any programmer.
2. **Variables:** Variables allow programmers to store and manipulate data in their programs.
3. **Control Structures:** Control structures such as if statements, loops, and switch statements allow programmers to control the flow of their programs.
4. **Functions:** Functions allow programmers to break their programs down into smaller, more manageable pieces.
5. **Arrays:** Arrays are a way to store multiple values in a single variable.
6. **Pointers:** Pointers are variables that hold memory addresses, and are used to manipulate memory directly.
7. **Object-Oriented Programming (OOP):** OOP is a programming paradigm that focuses on creating objects that encapsulate data and behavior.
8. **Inheritance:** Inheritance allows programmers to create new classes that inherit properties and behaviors from existing classes.
9. **Polymorphism:** Polymorphism allows programmers to use the same method or operator to work with different types of data.
10. **Algorithms:** Algorithms are step-by-step procedures for solving problems, and are a crucial part of programming.
11. **Debugging:** Debugging is the process of finding and fixing errors in code, and is an essential skill for any programmer.
12. **Software Development Life Cycle (SDLC):** SDLC is the process of developing software from initial planning to final deployment.

Understanding these concepts and how to apply them will help programmers write better code and create more robust software.

"The computer programmer is a creator of universes for which he alone is the lawgiver. No playwright, no stage director, no emperor, however powerful, has ever exercised such absolute authority to arrange a stage or field of battle and to command such unswervingly dutiful actors or troops."

− Joseph Weizenbaum
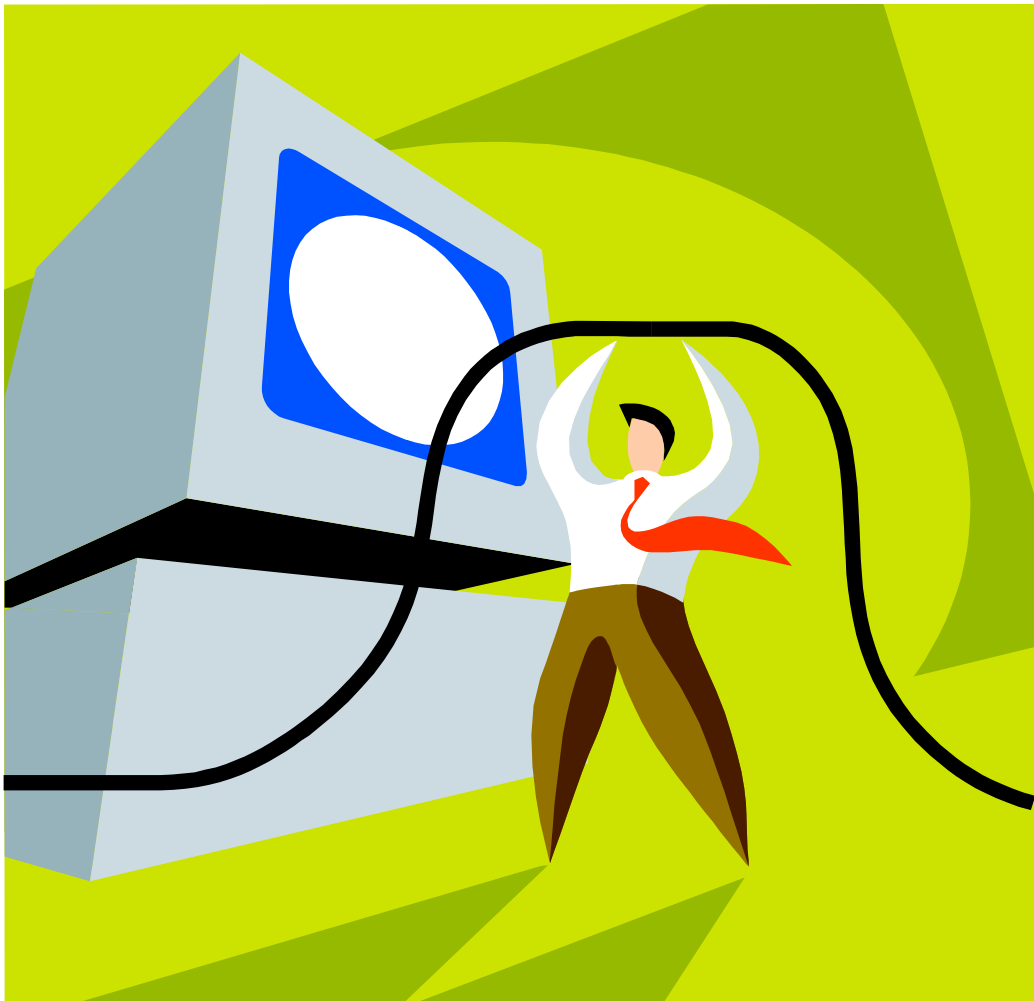
Top 10 Programming Languages and Their Applications

Python	Artificial Intelligence, Deep learning and Machine Learning
JavaScript	Rich Interactive Web Development
Java	Enterprise Application Development
R	Data Analysis
C/C++	Operating Systems and System Tools
Golang	Server-Side Programming
C#	Application and Web Development Using .NET
PHP	Web Development
SQL	Database Management
Swift	For Mobile Application Development on iOS

How much time does it take to become a good programmer?

A skilled coder can identify the best solution to any problem and solve even the most challenging issues. Being a good programmer requires constant knowledge upkeep and the acquisition of new skills. A PhD isn't always necessary to become a skilled programmer, but discipline and determination are. Being a successful programmer demands you to be one step ahead, while becoming a respectable coder takes years of hard effort.
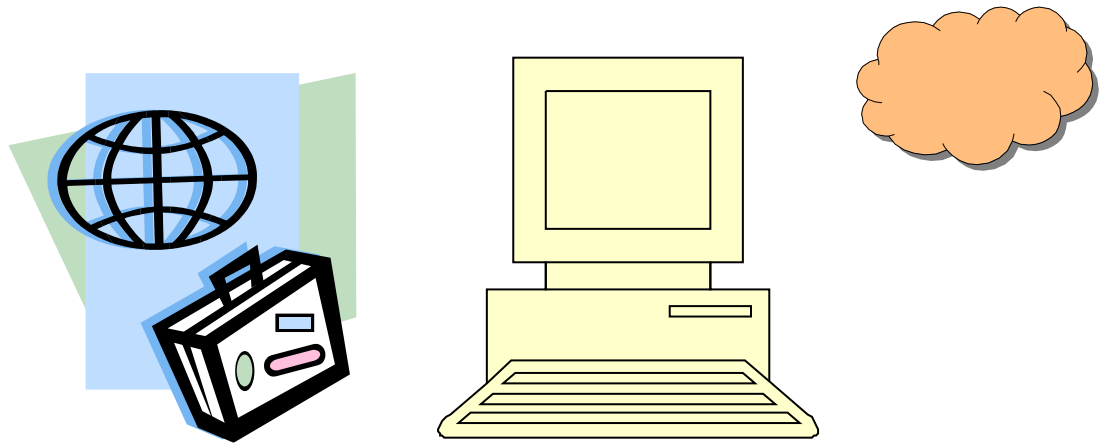
Contents

"A language that doesn't affect the way you think about programming is not worth knowing."

— Alan J. Perlis

LINUX COMMANDS

Linux is an open-source Unix-like operating system built on the Linux kernel, which Linus Torvalds initially made available on September 17, 1991. People switching from Windows or macOS may find Linux difficult to use and comprehend, and many people give up using it because they are not aware of the commands and shortcuts that Linux offers. When using Linux, you can accomplish tasks much more quickly than when using other operating systems by becoming familiar with the useful commands and how to use them correctly. We'll examine a few widely used Linux commands in this chapter.

When working with unstructured files in Linux, whether you are a sys admin or a database administrator, there are a number of commands that will be very helpful to you in your everyday job. Working as developers requires us to use the Linux command line occasionally. Linux is typically considered to be dominant for public Internet servers, powering well over twice as many servers as Windows Server. Linux has a monopoly on the supercomputer market, powering all 500 of the TOP computers. It's quite easy to customize Linux. Your OS can be modified. In this chapter, we'll look at some fundamental Linux commands that every programmer should be familiar with.

17 Principles of the philosophy of UNIX
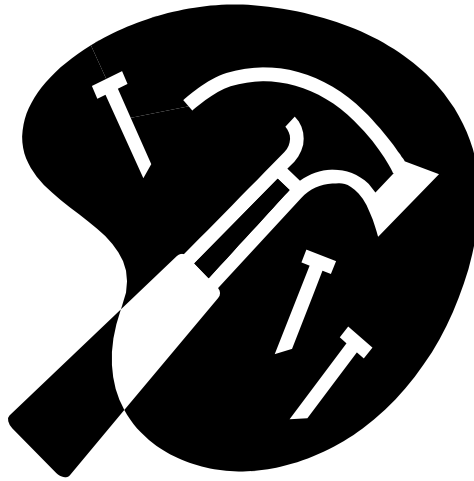
- Principle of **Modularity**: A system should be composed of several components that are joined, collaborate well, and have clearly defined functions
- Principle of **Clarity**: Clearness is better than smartness
- Principle of **Composition**: Create software that can communicate with other software
- Principle of **Separation**: Programming mechanisms and rules should be kept distinct. Keep front-end interfaces and back-end engines separate
- Principle of **Simplicity**: Build for simplicity and only add complexity where necessary
- Principle of **Parsimony**: Only write a large program when it is clear by demonstration that nothing else will work
- Principle of **Transparency**: Design with visibility in view to simplify analysis and troubleshooting
- Principle of **Robustness**: Transparency and ease of use produce robustness
- Principle of **Representation**: Create programs easier to understand for any programmer involved in the project so that it can be maintained

```
When offered the choice, programmers should choose to
complicate the data rather than the procedural logic of
the software because complex data is simpler for us to
understand than complex logic
```

- Principle of **Least Surprise**: Developers should be encouraged to create user-friendly, intuitive products
- Principle of **Silence**: Allow programmers and other programs to get the data they require from a program's output without having to interpret unnecessary extensive and detailed
- Principle of **Repair**: Programmers should create software that fails in a way that is simple to identify and diagnose
- Principle of **Economy**: Project development costs should be minimized
- Principle of **Generation**: Programmers should develop abstract, high-level programs that produce code rather than writing code by hand to decrease human error and save time
- Principle of **Optimization**: Before you can optimize it, get it working. Software should be developed and tested before being masterfully crafted by developers
- Principle of **Diversity**: Make programs flexible, enabling their use in ways other than those that their creators intended
- Principle of **Extensibility**: Increase the usefulness and lifespan of the developer's written code

Better skills come with increased learning.

Your main focus as a novice should be on becoming familiar with the ins and outs of operating system architecture as well as discovering shortcuts and time-saving techniques.
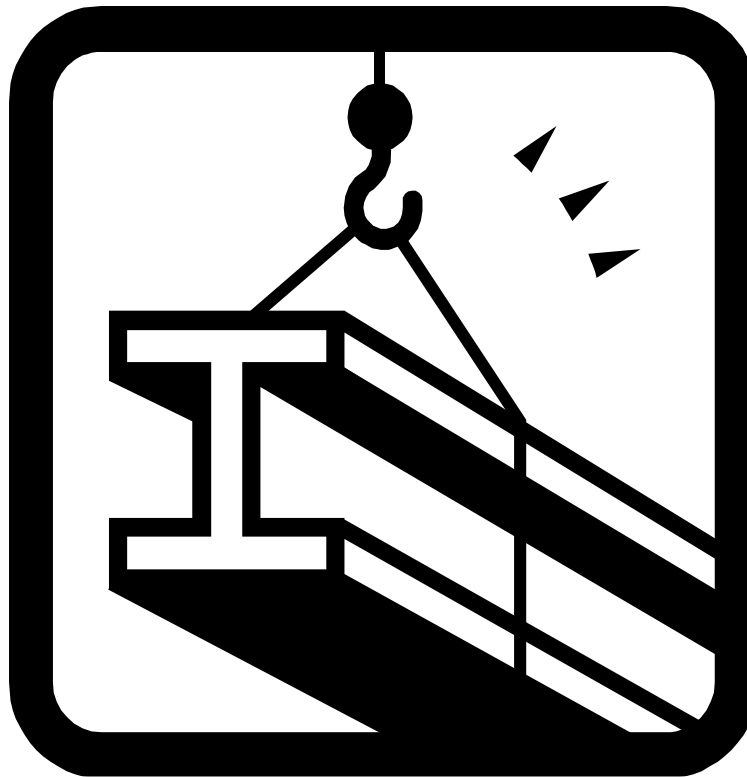
CentOs is a wonderful option if you use Windows and want to learn Linux because it is one of the best Linux distributions for beginners. Your first few days using CentOs won't be that odd. But you must learn how to utilize Linux's command line interface if you want to experience its full capabilities. You will initially experience some difficulty learning several instructions. Although employing the instructions won't make you a genius, it will assist you in carrying out certain fundamental tasks. Here are the most basic CentOs commands for new users to ensure a smooth start. Let's get right into it!
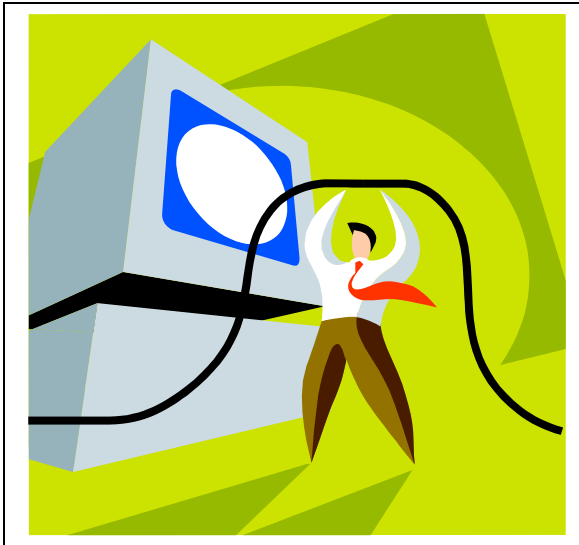
Linux Commands

Description:

Display system date and time.

_____

Command:

```
date
```

_____

Description:

Display calendar.

_____

Command:

```
cal
```

_____

Description:

Display date, time and calendar.

Command:

```
date & cal
```

Description:

Display August month 2016 year calendar.

Command:

```
cal 8 2016
```

Description:

Used to clear the terminal window.

Command:

```
clear
```

Description:

Exit from the terminal window.

Command:

```
exit
```

Description:

Display free and used system memory.

Command:

```
free
```

Description:

Display free and used system memory in bytes.

Command:

```
free  -b
```

Description:

Display free and used system memory in kilobytes.

Command:

```
free  -k
```

Description:

Display free and used system memory in megabytes.

Command:

```
free  -m
```

Description:

Change user password.

Command:

```
passwd
```

Description:

Power-off the machine.

Command:

```
shutdown
```

Description:

Power-off the machine immediately.

Command:

```
shutdown  -h now
```

Description:

Power-off the machine after 10 minutes.

Command:

```
shutdown  -h +10
```

Description:

Print current working directory.

Command:

```
echo $PWD
```

Description:

Print previous working directory.

Command:

```
echo $OLDPWD
```

Description:

Executes the 11th command in command history.

Command:

```
!11
```

Description:

Reveals your command history.

Command:

```
history
```

Description:

Power off or reboot the Operating system.

Command:

```
sudo reboot
```

Description:

Display the IP address of the host.

Command:

```
ip address
```

Description:

List the size of files and directories.

Command:

```
ls  -s
```

Description:

View mounted file systems.

Command:

```
mount
```

Description:

Display the information of disk usage of files and directories.

Command:

```
du
```

Description:

Tells you how long the system has been running.

Command:

```
uptime
```

Description:

Set current date as 02 Nov 1988.

Command:

```
date --set 1998-11-02
```

Description:

Set current time as 12:11:02 IST.

Command:

```
date --set 12:11:02
```

Description:

View Specific Disk Partition in Linux.

Command:

```
fdisk -l /dev/sda
```

Description:

Lists all files and directories in the present working directory.

Command:

```
ls
```

Description:

Report the process information.

Command:

```
ps
```

Description:

Display disk usage.

Command:

```
df
```

Description:

Display disk usage in gigabytes, megabytes, or kilobytes.

Command:

```
df  -H
```

Description:

Delete every file and every directory.

Command:

```
rm  -r *
```

Description:

Provides a quick overview of the currently running processes.

———————————————

Command:

```
top
```

———————————————

Description:

The system performs an immediate reboot.

———————————————

Command:

```
reboot
```

———————————————

Description:

Terminate processes without having to log out or reboot.

———————————————

Command:

```
kill
```

———————————————

Description:

Change the current working directory.

———————————————

Command:

```
cd
```

———————————————

Description:

Create a new session on the system.

Command:

```
login
```

Description:

List open files.

Command:

```
lsof
```

Description:

List USB devices.

Command:

```
lsusb
```

Description:

Check the status of the network services.

Command:

```
service network status
```

Description:

Start the network service.

Command:

```
service network start
```

Description:

Stop the network service.

Command:

```
service network stop
```

Description:

Restart the network service.

Command:

```
service network restart
```

Description:

Report information about the users currently on the machine and their processes.

Command:

```
w
```

Description:

Display the current directory.

Command:

```
pwd
```

Description:

Displays CPU architecture information (such as number of CPUs, threads, cores, sockets, and more).

Command:

```
lscpu
```

Description:

Displays the number of processing units available to the current process.

Command:

```
nproc
```

Description:

The system performs an immediate reboot.

_____

Command:

```
init 6
```

_____

Description:

Power-off the machine.

_____

Command:

```
init 0
```

_____

Description:

List files by date.

_____

Command:

```
ls -lrt
```

_____

Description:

Report information about storage devices such as hard disks, flash drives etc.

_____

Command:

```
lsblk
```

_____

Description:

Show exit status of previous command.

Command:

```
echo $?
```

Description:

Lists a few useful info commands.

Command:

```
info
```

Description:

Prints current year's calendar.

Command:

```
cal -y
```

Description:

Check the status of all the services.

Command:

```
service --status-all
```

Description:

Display time in hh:mm:ss.

Command:

```
date +%T
```

Description:

Tells when the user last logged on and off and from where.

Command:

```
last  -1 username
```

Description:

Sort files and directories by extension name.

Command:

```
ls  -X
```

Description:

Display the manual for the pwd command.

Command:

```
man pwd
```

Description:

Displays information about running processes in the form of a tree.

Command:

```
pstree
```

Description:

Resets your terminal.

Command:

```
reset
```

Description:

Displays What date is it this Friday.

Command:

```
date -d fri
```

Description:

Displays the size of each individual file.

———————————————

Command:

```
du  -a
```

———————————————

Description:

Display information about the Advanced configuration and power Interface.

———————————————

Command:

```
acpi
```

———————————————

Description:

Takes you two folders back.

———————————————

Command:

```
cd ../..
```

———————————————

Description:

Takes you to the previous directory.

———————————————

Command:

```
cd -
```

———————————————

Description:

Displays a list of shell built-in commands.

Command:

```
help
```

Description:

Lists your last logins.

Command:

```
last yourusername
```

Description:

Create a new directory called myfiles.

Command:

```
mkdir myfiles
```

Description:

Remove the directory myfiles.

Command:

```
rmdir myfiles                                    25
```

Description:

Disable password for a specific user "root1".

Command:

```
passwd -d root1
```

Description:

Switch to user "root1".

Command:

```
sudo su root1
```

Description:

Exit from the terminal window.

Command:

```
logout
```

Description:

Creates a user "root1".

Command:

```
useradd "root1"
```

Description:

Assign password to user "root1".

Command:

```
passwd "root1"
```

Description:

Repeats the last command.

Command:

```
!!
```

Description:

Display Who you are logged in as.

Command:

```
whoami
```

Description:

Display the login name of the current user.

Command:

```
logname
```

Description:

Report the name of the kernel.

Command:

```
uname
```

Description:

Print the kernel version.

Command:

```
uname  -v
```

Description:

Print the operating system.

Command:

```
uname  -o
```

Description:

Report the machine hardware name.

Command:

```
uname  -m
```

Description:

Print version information and exit.

Command:

```
uname  --version
```

Description:

Print the kernel release.

Command:

```
uname  -r
```

Description:

Report the network node hostname.

Command:

```
uname  -n                                                          29
```

Description:

Display all port connections (both TCP and UDP).

Command:

```
netstat  -a
```

Description:

Display only TCP (Transmission Control Protocol) port connections.

Command:

```
netstat  -at
```

Description:

Display only UDP (User Datagram Protocol) port connections.

Command:

```
netstat  -au
```

Description:

Display all active listening ports.

Command:

```
netstat  -l
```

Description:

Display all active listening TCP ports.

Command:

```
netstat  -lt
```

Description:

Display all active listening UDP ports.

Command:

```
netstat  -lu
```

Description:

Reveal all the information about the current user (user id, username, group id, group name etc.).

Command:

```
id
```

Description:

Reveal all the information about the user "root1" (user id, username, group id, group name etc.).

Command:

```
id  root1
```

Description:

Print the machine's architecture.

Command:

```
arch
```

Description:

Display the list of available fonts.

Command:

```
fc-list
```

Description:

Create two directories (myfiles, files).

Command:

```
mkdir myfiles files
```

Description:

install apache (CentOS).

Command:

```
yum install httpd
```

Description:

install apache (Ubuntu).

Command:

```
apt install httpd
```

Description:

upgrade apache (CentOS).

Command:

```
yum update httpd
```

Description:

upgrade apache (Ubuntu).

Command:

```
apt update httpd                                          33
```

Description:

uninstall apache (CentOS).

Command:

```
yum remove httpd
```

Description:

uninstall apache (Ubuntu).

Command:

```
apt remove httpd
```

Description:

Display usage summary for the command (date).

Command:

```
date --help
```

Description:

List active connections to/from system.

Command:

```
ss -tup
```

Description:

List internet services on a system.

Command:

```
ss -tupl
```

Description:

Display all active UNIX listening ports.

Command:

```
netstat  -lx
```

Description:

Display all the active interfaces details.

Command:

```
ifconfig
```

Description:

Display information of all network interfaces.

Command:

```
ifconfig  -a
```

Description:

Compare the contents of two files (1.txt, 2.txt).

Command:

```
diff 1.txt 2.txt
```

Description:

Tells you how many lines, words, and characters there are in a file (1.txt).

Command:

```
wc 1.txt
```

Description:

Compresses file (1.txt), so that it take up much less space.

Command:

```
gzip 1.txt
```

Description:

Uncompresses file (1.txt) compressed by gzip.

Command:

```
gunzip 1.txt
```

Description:

Examine the contents of the file (1.txt).

Command:

```
cat 1.txt
```

Description:

Display calendar.

Command:

```
ncal
```

Description:

Removes the file (1.txt).

Command:

```
rm 1.txt
```

Description:

Rename a file named 1.txt to 0.txt.

Command:

```
mv 1.txt 0.txt
```

Description:

Replace the contents of 0.txt with that of 1.txt.

Command:

```
cp 1.txt 0.txt
```

Description:

Create a empty file (test.txt).

Command:

```
touch test.txt
```

Description:

Print the last 10 lines of a file (1.txt).

Command:

```
tail 1.txt
```

Description:

Print N number of lines from the file (1.txt).

Command:

```
tail  -n N 1.txt
```

Description:

Prints the number of words in a file (1.txt).

Command:

```
wc -w 1.txt
```

Description:

Prints the number of characters from a file (1.txt).

Command:

```
wc -m 1.txt
```

Description:

Prints the length of the longest line in a file (1.txt).

Command:

```
wc -L 1.txt
```

Description:

Print information about usb ports, graphics cards, network adapters etc.

Command:

```
lspci
```

Description:

View contents of a file (1.txt).

Command:

```
less 1.txt
```

Description:

Display calendar (last month, current month, and next month).

Command:

```
cal -3
```

Description:

Compare the contents of three files (1.txt, 2.txt, 3.txt) line by line.

Command:

```
diff3 1.txt 2.txt 3.txt
```

Description:

Compare two files (1.txt, 2.txt) line-by-line.

Command:

```
comm 1.txt 2.txt
```

Description:

Perform byte-by-byte comparison of two files (1.txt, 2.txt).

Command:

```
cmp 1.txt 2.txt
```

Description:

Prints the CRC checksum and byte count for the file "myfiles.txt".

Command:

```
cksum myfiles.txt
```

Description:

Append contents of files (1.txt, 2.txt) into one file (0.txt).

Command:

```
cat 1.txt 2.txt > 0.txt
```

Description:

Append contents of files (1.txt, 2.txt, 3.txt) into one file (0.txt).

Command:

```
sed  r 1.txt 2.txt 3.txt > 0.txt
```

Description:

Append contents of files (1.txt, 2.txt, 3.txt) into one file (0.txt).

Command:

```
sed  h 1.txt 2.txt 3.txt > 0.txt
```

Description:

Append contents of files (1.txt, 2.txt, 3.txt) into one file (0.txt).

Command:

```
sed  -n  p 1.txt 2.txt 3.txt > 0.txt
```

Shortcuts:

```
| ctrl+c            | Halts the current command          |
| ctrl+z            | Stops the current command          |
|                   |                                    |
| ctrl+d            | Logout the current session         |
| ctrl+w            | Erases one word in the current line |
|                   |                                    |
| ctrl+u            | Erases the whole line              |
| ctrl+r            | Type to bring up a recent command  |
```

Description:

> Writes contents of a file (0.txt) to output, and prepends each line with line number.

Command:

```
nl 0.txt
```

Description:

> Create a empty file (test1.txt) inside a directory (test).

Command:

```
mkdir test
cd test
pwd
touch test1.txt
```

Description:

Gather information about hardware components such as CPU, disks, memory, USB controllers etc.

Command:

```
sudo lshw
```

Description:

Gather information about file system partitions.

Command:

```
sudo fdisk -l
```

Description:

Displays the line (good morning) in which the string (good) is found in the file (1.txt).

Command:

```
grep good 1.txt
```

Description:

Append contents of files (1.txt, 2.txt, 3.txt) into one file (0.txt) using for loop.

Command:

```
for i in {1..3}; do cat "$i.txt" >> 0.txt; done
```

Description:

Search for files (test.txt, test1.txt, test2.txt, test.php, test.html) in a directory as well as its sub-directories.

Command:

```
find test*
```

Description:

Displays status related to a file (1.txt).

Command:

```
stat 1.txt
```

```
###
```

Command	Description
vi	Open vi editor
i	Go to Insert mode
a =20; b =64;	
print (a + b);	
Hit Escape to return to Normal mode.	
:w hello.py	Save text
:q	Quit
python hello.py	Print the output:84

Description:

Download the file (file.txt) from url "http: //website.com/files/file.txt".

Command:

```
wget http://website.com/files/file.txt
```

Description:

Display host's numeric ID in hexadecimal format.

Command:

```
hostid
```

Description:

Display file type of the file (myfiles.txt).

Command:

```
file myfiles.txt
```

Description:

Create a file (myfiles.txt) containing a text (Hello World).

Command:

```
echo 'Hello World' > myfiles.txt
```

Description:

Create a file (myfiles.txt) containing a text (Hello World).

Command:

```
printf 'Hello World' > myfiles.txt
```

Description:

Display IP address of the hostname.

Command:

```
hostname  -i
```

Description:

Add a new line of text to an existing file (1.txt).

Command:

```
echo "Hello world!" >> 1.txt
echo "this is 2nd line text" >> 1.txt
echo "last line!" >> 1.txt
```

Description:

Displays a single line description about a command (cal).

Command:

```
whatis cal
```

Command	Description
vi	Open vi editor
i	Go to Insert mode
Type some text.	
Hit Escape to return to Normal mode.
:w test.txt	Save text
:q	Quit
:q!	Quit without saving

Command	Description
vi	Open vi editor
i	Go to Insert mode
$name = "Paul";	
print "$name";	
Hit Escape to return to Normal mode.
:w hello.pl	Save text
:q	Quit
perl hello.pl	Print the output: Paul

Command	Description
vi	Open vi editor
i	Go to Insert mode
echo "What is your name?"	
read PERSON	
echo "Hello, $PERSON"	
Hit Escape to return to Normal mode.
:w hello.sh	Save text
:q	Quit
sh hello.sh	Output:
	What is your name?
	If you enter: Zara Ali
	Hello, Zara Ali

Description:

Check the network connectivity between host (your connection) and server (Google server).

Command:

```
ping google.com
```

Description:

Find the location of source/binary file of a command (cal).

Command:

```
whereis cal
```

There are 2 ways to use the command:

- Numeric mode
- Symbolic mode

```
# Overwrite existing file
$ echo "Albert Einstein" > 1.txt
# Append a second line
$ echo "Alan Turing" >> 1.txt
```

Numeric mode	Permission Type	Symbolic mode
0	No Permission	---
1	Execute	--x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r--
5	Read + Execute	r-x
6	Read + Write	rw-
7	Read + Write + Execute	rwx

```
[manju@localhost ~]$ ps -ef | grep sshd
root        988      1  0 06:14 ?        00:00:00 /usr/sbin/sshd
manju      3501   3461  0 06:24 pts/0    00:00:00 grep --color=auto sshd

            # Check if the SSH server (sshd) is running
```

```
cd /etc && ls

# Execute ls after cd /etc


rm myfiles.txt && echo success || echo failed
```

Print 'success' if myfiles.txt is removed and print 'failed' if it is not removed

```
[manju@localhost ~]$ echo This is the $SHELL shell

This is the /bin/bash shell

[manju@localhost ~]$ echo This is $SHELL on computer $HOSTNAME

This is /bin/bash on computer localhost.localdomain

[manju@localhost ~]$ echo The user ID of $USER is $UID

The user ID of manju is 1000

[manju@localhost ~]$ echo My home directory is $HOME

My home directory is /home/manju
```

```
[manju@localhost ~]$ bash -c 'echo $SHELL $HOME $USER'

/bin/bash /home/manju manju

[manju@localhost ~]$ env -i bash -c 'echo $SHELL $HOME $USER'

/bin/bash
```

```
env LANG=C bash -c 'ls test[a-z].txt'

testa.txt   testb.txt   testc.txt


env LANG=en_US.UTF-8 bash -c 'ls test[a-z].txt'

testa.txt   testA.txt   testb.txt   testc.txt   testC.txt
```

```
[manju@localhost ~]$ prefix=John

[manju@localhost ~]$ echo Hello ${prefix}Dalton and ${prefix}Humphrys

Hello JohnDalton and JohnHumphrys
```

```
echo $(a=5;echo $a)
```

```
5
```

```
echo 'a=5;echo $a'
```

```
a=5;echo $a
```

```
[manju@localhost ~]$ touch myfiles.txt

[manju@localhost ~]$ cat myfiles.txt

Hello World

[manju@localhost ~]$ !to

touch myfiles.txt
```

```
[manju@localhost ~]$ echo $HISTSIZE

1000
```

The number of commands that are stored in memory in a history list while your bash session is ongoing

```
[manju@localhost ~]$ echo $HISTFILE

/home/manju/.bash_history
```

Holds the name and location of your **Bash history file**

```
echo $HISTFILESIZE

1000
```

How many commands can be stored in the `.bash_history` file

```
[manju@localhost ~]$ ls *ile1.txt

file1.txt


[manju@localhost ~]$ ls f*ile1.txt

file1.txt


[manju@localhost ~]$ ls f*1.txt

file1.txt


[manju@localhost ~]$ ls file?.txt

file1.txt   file2.txt   file3.txt


[manju@localhost ~]$ ls fil?1.txt

file1.txt


[manju@localhost ~]$ ls fil??.txt

file1.txt   file2.txt   file3.txt


[manju@localhost ~]$ ls file??.txt

file23.txt   file34.txt
```

```
[manju@localhost ~]$ ls test[5A].txt

testA.txt


[manju@localhost ~]$ ls test[A5].txt

testA.txt


[manju@localhost ~]$ ls file[!5]*.txt

file123.txt   file1.txt   file23.txt   file2.txt   file34.txt   file3.txt


[manju@localhost ~]$ ls file[!5]?.txt

file23.txt   file34.txt
```

```
[manju@localhost ~]$ ls [a-z]ile?.txt

file1.txt   file2.txt   file3.txt


[manju@localhost ~]$ ls [A-Z]ile?.txt

file1.txt   file2.txt   file3.txt


[manju@localhost ~]$ echo \*

*

[manju@localhost ~]$ echo '*'

*

[manju@localhost ~]$ echo "*"

*

[manju@localhost ~]$ ls [a-z]*[0-9].txt

file123.txt   file1.txt   file23.txt   file2.txt   file34.txt   file3.txt
```

> List all .txt files starting with a
> letter and ending in a number

```
ls ?????                              53

# List all files that have exactly five characters


ls [fF]*[3A].txt

# List all .txt files that start with f or F and end with 3 or A


ls f[iR]*[0-9].txt

#List all .txt files that start with f have i or R as second character and end in a number


ls [!f]*.txt

# List all .txt files that do not start with the letter "f"
```

```
[manju@localhost ~]$ echo Einstein2 | sed 's/2/36/'

Einstein36

[manju@localhost ~]$ echo Einstein36 | sed 's/Einstein/Hilbert/'

Hilbert36
```

```
[manju@localhost ~]$ echo Hawking6 Lucy8 | sed 's/Hawking/Lucy/'

Lucy6 Lucy8




[manju@localhost ~]$ echo Lucy3 Lucy6 | sed 's/Lucy/Hawking/g'

Hawking3 Hawking6
```

```
[manju@localhost ~]$ who | cut -d' ' -f1 | sort

manju

manju
```

> Display a sorted list
> of logged on users

```
[manju@localhost ~]$ who | cut -d' ' -f1 | sort | uniq

manju
```

> Display a sorted
> list of logged on
> users – but every
> user only once

```
[manju@localhost ~]$ grep bash /etc/passwd

root:x:0:0:root:/root:/bin/bash

manju:x:1000:1000:su,root,yopp,hhhhh:/home/manju:/bin/bash
```

Display a list of all bash user accounts on this computer

```
grep bash /etc/passwd | cut -d: -f1 | sort > bu.txt

# Place a sorted list of all bash users in bu.txt


who | cut -d' ' -f1 | sort > users.txt

# Place a sorted list of all logged on users in users.txt


ls /etc | grep conf

# List of all filenames in /etc that contain the string "conf" in their filename


ls /etc | grep -i conf | sort
```

**Display a sorted list of all files in /etc
that contain the case insensitive string
"conf" in their filename**

90% of the public cloud workload is run on Linux distros.

The first ever Linux kernel just occupied only 65 KB.

import subprocess	import os
subprocess.call ('**linux command**')	os.system('**linux command**')

All of the 500 fastest supercomputers run Linux.

import os os.system('**ls**')	**List all the files and directories in the current directory**
import subprocess subprocess.call ('**ls**')	

Execution of the linux command "ls" using the python program

Command:

```
last reboot
```

Description:

Show system reboot history

Command:

```
dmesg
```

Description:

Displays the messages from the kernel ring buffer (a data structure that records messages related to the operation of the kernel**)**

Command:

```
cat /proc/cpuinfo
```

Description:

Display CPU information

Command:

```
cat /proc/meminfo
```

Description:

Display memory information

Command:

```
lspci -tv
```

Description:

Display PCI (Peripheral Component Interconnect**) devices**

Command:

```
lsusb -tv
```

Description:

Display USB devices

Command:

```
free -h
```

Description:

Display free and used memory (-h for human readable, -m for MB, -g for GB**)**

Command:

```
mpstat 1
```

Description:

Display processor related statistics

Command:

```
vmstat 1
```

Description:

Display virtual memory statistics

Command:

```
iostat 1
```

Description:

Display Input / Output statistics

Command:

```
watch df -h
```

Description:

Execute "df -h" command, showing periodic updates

Command:

```
ps -ef
```

Description:

Display all the currently running processes on the system

Command:

```
ip a
```

Description:

Display all network interfaces and IP address

Command:

```
dig wikipedia.org
```

Description:

Display DNS information for domain (wikipedia.org**)**

Command:

```
host wikipedia.org
```

Description:

Display the IP address details of the specified domain (wikipedia.org**)**

Command:

```
netstat -nutlp
```

Description:

Display listening Transmission Control Protocol (TCP**) and the User Datagram Protocol (**UDP**) ports and corresponding programs**

Command:

```
rpm -qa
```

Description:

List all installed packages

Command:

```
yum list installed
```

Description:

List all installed packages (CentOS)

Command:

```
yum info httpd
```

Description:

Display description and summary information about package "httpd" **(CentOS)**

Command:

```
du -ah
```

Description:

Display disk usage for all files and directories in human readable format

Command:

```
du -sh
```

Description:

Display total disk usage off the current directory

Command:

```
cd /etc
```

Description:

Change to the /etc directory

Command:

```
ps -A
```

Description:

List the status of all the processes along with process id and PID

Command:

```
#include <stdio.h>
int main()
{
  printf("Hello world\n");          Hello.c
  return 0;
}



gcc Hello.c
```

Description:

Compile the C program saved in Hello.c file

Command:

```
#include <iostream>
int main()
{
std::cout << "Hello world!";      Hello.cpp
   return 0;
}



g++ Hello.cpp
```

Description:

Compile the C++ program saved in Hello.cpp file

Command:

```
tty
```

Description:

Displays the file name of the terminal connected to standard input

Command:

```
public class MyClass {
public static void main(String [] args) {
System.out.println("Hello, World!");          MyClass.java
}
}


javac MyClass.java
```

Description:

Compile the Java program saved in MyClass.java file using javac compiler

Command:

```
od -b myfiles.txt
```

Description:

Displays the contents of myfiles.txt **file in octal format**

Command:

```
od -c myfiles.txt
```

Description:

Displays the contents of myfiles.txt **file in character format**

Command:

```
od -An -c myfiles.txt
```

Description:

Displays the contents of myfiles.txt **file in character format but with no offset information**

Command:

```
csplit myfiles.txt 13 62 101
```

Description:

If the file **myfiles.txt** has 123 lines, the **csplit command** would create four files: the xx00 file would contain lines 1−12, the xx01 file would contain lines 13−61, the xx02 file would contain lines 62−100, the xx03 file would contain lines 101−123

Command:

```
md5sum myfiles.txt
```

Description:

Prints a 32-character (128-bit) checksum of myfiles.txt **file using the MD5 algorithm**

Command:

```
more myfiles.txt
```

Description:

Displays the content of myfiles.txt **file**

Command:

```
sha1sum myfiles.txt
```

Description:

Prints SHA1 (160-bit) checksum of myfiles.txt **file**

SHA 1 → Secure Hash Algorithm 1

Command:

```
shred myfiles.txt
```

Description:

Overwrites the myfiles.txt **file repeatedly − in order to make it harder for even very expensive hardware probing to recover the data**

Command:

```
cat myfile.txt

01. Einstein
02. Newton
03. Maxwell
04. Tesla
05. Edison

tac myfile.txt

05. Edison
04. Tesla
03. Maxwell
02. Newton
01. Einstein
```

Description:

Print the lines of myfile.txt **in reverse (from last line to first)**

Command:

```
uniq myfiles.txt
```

Description:

Delete repeated lines in the file (myfiles.txt).

Command:

```
chkconfig --list
```

Description:

Displays a list of system services and whether they are started (on) or stopped (off) in run levels 0−6

Command:

```
halt -p
```

Description:

Power-off the system

Command:

```
xdg-open myfiles.txt
```

Description:

Open a file (myfiles.txt).

Command:

```
lastlog
```

Description:

Prints the details of the last login (login-name, port and last login time**)**

Command:

```
lastlog -t 1
```

Description:

Displays the login information (1 day ago**)**

Command:

```
lastlog -u manju
```

Description:

Display lastlog information for a particular user (manju**)**

Command:

```
cat /etc/passwd
```

```
more /etc/passwd
```

```
less /etc/passwd
```

```
getent passwd
```

Description:

List all users on Linux

Command:

```
tail -5 /etc/passwd

head -5 /etc/passwd
```

Description:

List last 5 users on Linux

List first 5 users on Linux

Command:

```
wall "The system will be shutdown in 10 minutes."
```

Description:

The message (The system will be shutdown in 10 minutes.**) will be broadcasted to**

all users that are currently logged in

Command:

```
chage -l manju
```

Description:

List the password and its related details for a user (manju**)**

Command:

```
chage -M 10 manju
```

Description:

Set Password Expiry Date for an user (manju**)**

Command:

```
chage -E "2020-07-30" manju
```

Description:

Set the Account Expiry Date for an User (manju**)**

Command:

```
chage -I 10 manju
```

Description:

Force the user (manju**) account to be locked after 10 inactivity days**

```
cat /etc/hostname

→ localhost.localdomain
```

Display the hostname of the system

```
hostname

→ localhost.localdomain
```

```
nmtui

# Configure a network interface IPv4 address


yum check-update

# Check whether any updates are available for your installed packages


yum search httpd

#Find any packages containing the specified keyword "httpd"


ls /etc

# List the contents of /etc


ls /bin /sbin

# List the contents of /bin and /sbin


ls -al ~

# List all the files (including hidden files) in the home directory


ls -lh /boot

# List the files in /boot in a human readable format
```

74

```
mkdir ~/mydir
```

Create a directory "mydir" under home directory

```
cd /etc ; mkdir ~/mydir
```

Change to the /etc directory and create a directory "mydir" under home directory.

```
rm -i file.txt
```

```
rm: remove regular empty file 'file.txt'?

If we type "yes"

file.txt is removed


If we type "no"

file.txt is not removed
```

```
rename .txt .backup *.txt
```

Renames all .txt files replacing.txt with .backup

```
ls

file.txt   cod.txt    conf.txt



rename file FILE *              file /bin/cat /etc/passwd /usr/bin/passwd

                               Display the type of file of /bin/cat, /etc/passwd and /usr/bin/passwd


ls

FILE.txt   cod.txt    conf.txt
```

Command:

```
ftp 192.168.42.77
```

Description:

Connect to an FTP server at remote server IP address "192.168.42.77"

Command:

```
arp -a
```

Description:

Lists all the peers connected at various interfaces along with their MAC Addresses **and IP addresses**

Command:

```
dnsdomainname
```

Description:

Display the system's DNS domain name

Command:

```
domainname
```

Description:

Display the name of the domain your machine belongs to

Command:

```
echo 'Hello World!' | base64

Output: SGVsbG8gV29ybGQhCg==
```

Description:

Encode text (Hello World!**) to base64**

Command:

```
echo 'SGVsbG8gV29ybGQhCg==' | base64 -d

Output: Hello World!
```

Description:

Decode (SGVsbG8gV29ybGQhCg==**) to text (**Hello World!**)**

Command:

```
fc-cache -f -v
```

Description:

Build font information cache files

Command:

```
cat 1.txt

Einstein
Newton
Albert

fmt 1.txt

Einstein Newton Albert
```

Description:

Formats text in a single line

```
df -h | sort -rnk 5 | head -3 | \

awk '{ print "Partition " $6 "\t: " $5 " full!" }'
```

```
    Partition /boot : 51% full!

    Partition /     : 29% full!

    Partition /run  : 2% full!
```

```
awk 'BEGIN { FS=":" } { print $1 "\t" $5 }' /etc/passwd

# Display all the users on your system
```

```
ls *.xml

1.xml   2.xml
```

```
ls *.xml > list.txt

[manju@localhost ~]$ cat list.txt

1.xml

2.xml
```

```
for i in `cat list.txt`; do cp "$i" "$i".md ; done
```

```
[manju@localhost ~]$ ls

12.txt    2.xml.md     Documents    file34.txt   Music          Pictures   tree.cpio

13.txt    3.txt        Downloads    file3.txt    mydi           Public     users.txt

145.txt   all          echo         FILE.backup  mydir          SHOW       Videos

1.txt     allfiles.txt file         file.md      mydir1         Templates

1.xml     bu.txt       file123.txt  first.bash   myfiles.txt    test

1.xml.md  Desktop      file1.txt    first.txt    myFILEs.txt.xz testA.txt

2.txt     DICT         file23.txt   foo1.txt     newdir         testB.txt

2.xml     dir          file2.txt    list.txt     nohup.out      text
```

```
[manju@localhost ~]$ df -h /

Filesystem          Size  Used Avail Use% Mounted on

/dev/sda3           18G   5.2G  13G   29%   /
```

> Check the actual used space on the **current root device**

```
less /proc/modules

# Display information about what kernel-modules are loaded on your system


[manju@localhost ~]$ free -tm
              total        used        free      shared  buff/cache   available

Mem:         999936      511156       73480        8572      415300      284236

Swap:       2097148           0     2097148
```

> **Display the memory usage including totals in megabytes**

```
[manju@localhost ~]$ date --date="3 months 1 day ago"

Mon Jul 18 23:17:47 PDT 2022
```

> **Print the date 3 months and 1 day ago from the current date**

```
[manju@localhost ~]$ date -d "3 days"

Fri Apr 22 23:20:01 PDT 2022
```

> **Print the date 3 days in the future from now**

```
[manju@localhost ~]$ cat myfiles.txt

Hello World

[manju@localhost ~]$ cat myfiles.txt | tr 'H' 'A' > myfilesB.txt

[manju@localhost ~]$ cat myfilesB.txt

Aello World
```

```
[manju@localhost ~]$ fgrep 'He' myfiles.txt

Hello World
```

Look for the string **"He"** in the file **"myfiles.txt"**

Command:

```
lsattr
```

Description:

List the files in the current directory

Command:

```
cp {*.txt,*.md} ~
```

Description:

Copy the files ending with .txt or .md to the user's home directory

```
[manju@localhost ~]$ cat myfiles.txt

Hello World
```

[manju@localhost ~]$ grep --color -i Hello myfiles.txt

Hello World

```
ls file*

# List all files in the current directory starting with "file"

ls *file

# List all files in the current directory ending with "file"
```

```
cat phy.txt

Albert Einstein was a German-born theoretical physicist, widely acknowledged to
be one of the greatest physicists of all time. Einstein is known for developing
the theory of relativity, but he also made important contributions to the
development of the theory of quantum mechanics.


fmt -w 1  phy.txt

 Albert
 Einstein
 was
 a
 German-born
 theoretical
 physicist,
 widely
 acknowledged
 to
 be
 one
 of
 the
 greatest
 physicists
 of
 all
 time.
 Einstein
 is
 known
 for
 developing
```

```
the
theory
of
relativity,
but
he
also
made
important
contributions
to
the
development
of
the
theory
of
quantum
mechanics.
```

```
cat phy.txt

Albert Einstein was a German-born theoretical physicist, widely acknowledged to
be one of the greatest physicists of all time. Einstein is known for developing
the theory of relativity, but he also made important contributions to the
development of the theory of quantum mechanics.


fold -w 20  phy.txt

Albert Einstein was
```

```
 a German-born theor
etical physicist, wi
dely acknowledged to
 be one of the great
est physicists of al
l time. Einstein is
known for developing
 the theory of relat
ivity, but he also m
ade important contri
butions to the devel
opment of the theory
 of quantum mechanic
s.
```

Command:

```
traceroute google.com
```

Description:

Prints the route that a packet takes to reach the Google (172.217.26.206**) host from the local machine**

Command:

```
cat 1.txt
```

```
Einstein
Newton
Albert


gzip 1.txt


zcat 1.txt.gz


Einstein
Newton
Albert
```

Description:

View the contents of zipped file

Command:

```
zdiff 1.txt.gz 2.txt.gz
```

Description:

Compare the contents of two zipped files (1.txt.gz, 2.txt.gz**)**

Command:

```
ss | less
```

Description:

List all connections

Command:

```
ss -aA tcp
```

Description:

Filter out TCP (Transmission Control Protocol**) connections**

Command:

```
ss -aA udp
```

Description:

Filter out UDP (User Datagram Protocol**) connections**

Command:

```
ss -lnt
```

Description:

Display only listening sockets

Command:

```
ss -ltp
```

Description:

Print process name and PID

Command:

```
ss -s
```

Description:

Print summary statistics

Command:

```
ss -tl6
```

Description:

Display only IPv6 connections

Command:

```
ss -tl -f inet
```

Description:

Display only IPv4 socket connections

Command:

```
ss -t4 state established
```

Description:

Display all IPv4 TCP sockets that are in connected state

Command:

```
pmap 3244
```

Description:

View the memory map of a process with Process ID (3244**)**

Command:

```
apropos -r 'remove file'
```

Description:

Find command that removes file

Command:

```
apropos editor
```

Description:

Display information about the editing programs that are available on a system

Command:

```
apropos pstree
```

Description:

Provide information about the pstree command (which displays the names of the processes currently on the system in the form of a tree diagram**)**

The **apropos command** is useful when you know what you want to do, but you have no idea what command you should be using to do it. If you were wondering how to locate files, for example, the commands

apropos find

and

apropos locate

would have a lot of suggestions to offer.

```
basename /etc/passwd

Output: passwd

basename /usr/local/apache2/conf/httpd.conf
```

```
Output: httpd.conf
```

```
echo a b c d e f| xargs

Output: a b c d e f


echo a b c d e f| xargs -n 3

Output:                     display only 3 items per line

a b c
d e f
```

Command:

```
env
```

Description:

> **Print out a list of all environment variables**

Command:

```
printenv HOME
```

Description:

Print HOME variable value

```
cat score.txt

Albert-30
John-50
William-80
Stephen-20
Justin-40

cut -d- -f2 score.txt

30
50
80
20
40

cut -d- -f1 score.txt

Albert
John
William
Stephen
Justin
```

```
cat 1.txt

Hello World


cat 2.txt

Computer Program


paste 1.txt  2.txt

Hello World    Computer Program
```

```
cat 1.txt

Hello World


cat 2.txt

Computer Program


join 1.txt  2.txt

Hello World    Computer Program
```

Command:

```
rev 1.txt
```

Description:

Reverse lines of a file (1.txt**)**

```
cat 3.txt

22
33
11
77
55

sort 3.txt

11
22          sorts numeric values in 3.txt file and displays sorted output
33
55
77
```

```
cat 1.txt

Hello World

cat 1.txt | tr "[a-z]" "[A-Z]"

HELLO WORLD
```
convert from lower case to upper case

```
cat 5.txt

zz
zz
yy
yy
yy
xx

uniq 5.txt

zz
yy
xx
```
removes duplicate lines and displays unique lines

```
cat 6.txt

Einstein
Newton
Tesla


nl 6.txt

1 Einstein
2 Newton                        numbers the lines in a file (6.txt)
3 Tesla
```

Command:

```
ls -l *.txt
```

Description:

Lists the files with .txt **extension**

> The thing with Linux is that the developers themselves are actually customers too: that has always been an important part of Linux.
>
> **Linus Torvalds**

```
ls /proc/bus/
```

List the contents of the /proc/bus/ directory

```
[manju@localhost ~]$ dmesg | grep "irq 1[45]"
```

Find **irq's** allocated at boot time

```
[ 2.269581] ata1: PATA max UDMA/33 cmd 0x1f0 ctl 0x3f6 bmdma 0x1060 irq 14

[ 2.269585] ata2: PATA max UDMA/33 cmd 0x170 ctl 0x376 bmdma 0x1068 irq 15
```

cat /proc/ioports

List system's IO ports

```
echo Albert > 1.txt ; echo Einstein > 2.txt

cat 1.txt

Albert

cat 2.txt

Einstein
```

[manju@localhost home]$ echo $-

himBH

```
[manju@localhost ~]$ s=01234567890abcdefgh; echo ${s:7}

7890abcdefgh


[manju@localhost ~]$ cd /home/manju; echo $PWD

/home/manju


[manju@localhost ~]$ cd ..; pwd

/home
```

```
[manju@localhost home]$ w | cut -d " " -f 1 - | grep -v USER | sort -u
manju
```

> Users currently connected

```
[manju@localhost ~]$ echo "\\"
\


[manju@localhost ~]$ echo Al{ber,an,er}t
Albert Alant Alert


[manju@localhost ~]$ echo ${Albert:=Einstein}
Einstein


[manju@localhost ~]$ echo $[5*5]
25
```
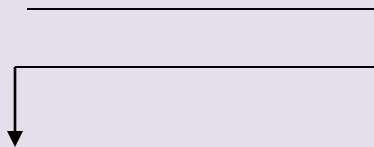
```
[manju@localhost ~]$ ls
```

12.txt	allfiles.txt	echo	file3.txt	mydi	Pictures	text
13.txt	bu.txt	file	FILE.backup	mydir	Public	tree.cpio
145.txt	Desktop	file123.txt	file.md	mydir1	SHOW	users.txt
1.txt	DICT	file1.txt	first.bash	myfiles.txt	Templates	Videos
2.txt	dir	file23.txt	first.txt	myFILEs.txt.xz	test	
3.txt	Documents	file2.txt	foo1.txt	newdir	testA.txt	
all	Downloads	file34.txt	Music	nohup.out	testB.txt	

```
ls -ldh * | grep -v total | \
awk '{ print "Size is " $5 " bytes for " $9 }'
```

Size is 135K bytes for 12.txt

Size is 13M bytes for 13.txt

Size is 0 bytes for 145.txt

Size is 7 bytes for 1.txt

Size is 9 bytes for 2.txt

Size is 8 bytes for 3.txt

Size is 20 bytes for all

Size is 13M bytes for allfiles.txt

Size is 11 bytes for bu.txt

Size is 6 bytes for Desktop

Size is 0 bytes for DICT

Size is 6 bytes for dir

Size is 6 bytes for Documents

Size is 6 bytes for Downloads

Size is 0 bytes for echo

Size is 0 bytes for file

Size is 0 bytes for file123.txt

Size is 0 bytes for file1.txt

Size is 0 bytes for file23.txt

Size is 0 bytes for file2.txt

Size is 45 bytes for mydir

Size is 47 bytes for mydir1

Size is 12 bytes for myfiles.txt

Size is 68 bytes for myFILEs.txt.xz

Size is 0 bytes for file34.txt

Size is 0 bytes for file3.txt

Size is 0 bytes for FILE.backup

Size is 3 bytes for file.md

Size is 13 bytes for first.bash

Size is 13 bytes for first.txt

Size is 66 bytes for foo1.txt

Size is 6 bytes for Music

Size is 31 bytes for mydi

Size is 6 bytes for newdir

Size is 148 bytes for nohup.out

Size is 6 bytes for Pictures

Size is 6 bytes for Public

Size is 0 bytes for SHOW

Size is 6 bytes for Templates

Size is 6 bytes for test

Size is 0 bytes for testA.txt

Size is 0 bytes for testB.txt

Size is 25 bytes for text

Size is 512 bytes for tree.cpio

Size is 12 bytes for users.txt

Size is 6 bytes for Videos

Linux	Unix
Free to use (**open source**)	Licensed Operating System (**closed source**)
Linux is just the kernel	Unix is a complete package of Operating System
Bash (**Bourne Again SHell**) is default shell for Linux	Bourne Shell is default shell for Unix
Portable and is booted from a USB Stick	Unportable
Source code is accessible to the general public	Source code is not accessible to anyone
Uses Graphical User Interface with an optional Command Line Interface	Uses Command Line Interface

Command:

```
echo $SHELL
```

Description:

Print the Default shell of user

Command:

```
echo $0
```

Description:

Display the name of the currently running process (**$0 is the name of the running process**). If you use it inside of a shell then it will return the name of the shell. If you use it inside of a script, it will return the name of the script

Command:

```
echo *
```

Description:

Print all files and folders − similar to ls command

Command:

```
ps -p $$

Output:


  PID  TTY        TIME   CMD
 3352 pts/0    00:00:00 bash
```

Description:

Print the process ID of the current shell ($$ is the process ID of the current shell)

```
sudo du -a Documents/ | sort -n -r | head -n 5

# List 5 biggest files from directory "Documents"
```

Command:

```
cat /etc/shells
```

Description:

List shells

Command:

```
echo m*
```

Description:

Display the files in the current folder that start with the letter "m".

Command:

```
last
```

Description:

List last logins of users and what happened such as "shutdown" **or** "crash" **etc.**

Command:

Command:
```
echo ~
```
Description:

Print your home folder path

```
bzip2 -k phy.txt
```

Description:

Compresses but does not deletes the original file

phy.txt → phy.txt.bz2

Command:

```
bzip2 -d phy.txt.bz2
```

Description:

Decompresses the compressed file (phy.txt.bz2**)**

phy.txt.bz2 → phy.txt

Command:

```
bzcat phy.txt.bz2
```

Description:

Display the contents of compressed file (phy.txt.bz2**)**

Command:

```
bunzip2 phy.txt.bz2
```

Description:

Decompresses the compressed file (phy.txt.bz2**)**

Command:

```
crontab -l
```

Description:

Display current logged-in user's crontab entries

```
cat /dev/null > phy.txt
```

```
cp /dev/null phy.txt
```

```
echo "" > phy.txt
```

```
echo > phy.txt
```

Description:

Empty the content of a file (phy.txt**)**

Command:

```
nohup ping google.com &
```

Description:

Ping google.com **and send the process to the background**

Command:

```
nohup ping google.com > log.txt &
```

Description:

Save the ping logs to log.txt

```
pgrep -a ping

Output:

3858 ping google.com
4200 ping google.com
4236 ping google.com

kill 3858

pgrep -a ping

Output:

4200 ping google.com
4236 ping google.com
```

Command:

```
ls -la /home
```

Description:

Display the contents of /home

Command:

```
sudo shutdown 2
```

Description:

Power-off the machine after 2 minutes

Command:

```
shutdown -c
```

Description:

Cancel the shutdown process

Command:

```
pr 36.txt
```

Description:

Display the contents of the file (36.txt**) one page after the other**

Command:

```
stty -a
```

Description:

Display all current terminal settings

Command:

```
ls -1
```

Description:

List files one per line

Command:

```
yes John
```

Description:

Outputs a string (John) repeatedly until killed

Command:

```
vdir
```

Description:

List files and directories in the current directory (one per line) with details

Command:

```
who -b
```

Description:

Print when the system was booted

```
# Open phy.txt with nano
```

```
    nano phy.txt


    # Open phy.txt with vim

    vim phy.txt
```

Command:

uname -i

Description:

Display the platform of hardware

Command:

uname -p

Description:

Display the type of processor

Command:

cat /proc/interrupts

Description:

Display the interrupts

```
w --ip-addr
```

```
# Displays information regarding the users currently on the machine, login time, IDLE time,
TTY and CPU time


Output:


11:12:10 up  1:29,  2 users,  load average: 0.02, 0.04, 0.10
USER     TTY      FROM  LOGIN@   IDLE    JCPU   PCPU WHAT
manju    :0       :0    02:43    ?xdm?   3:30   0.65s gdm-session-worker [pa
manju    pts/0    :0    11:01    2.00s   0.10s  0.01s w --ip-addr


w –short
# Omits CPU time and login information
Output:


11:11:46 up  1:28,  2 users,  load average: 0.02, 0.04, 0.11
USER     TTY      FROM             IDLE WHAT
manju    :0       :0               ?xdm?  gdm-session-worker [pam/gdm-password]
manju    pts/0    :0                2.00s w --short
```

Command:

```
findmnt
```

Description:

Display a list of currently mounted file systems

Command:

```
ip addr show
```

Description:

List IP addresses and network interfaces

Command:

```
netstat -pnltu
```

Description:

List active (listening) ports

Command:

```
journalctl
```

Description:

Display systemd, kernel and journal logs

Command:

```
sudo systemctl status network
```

Description:

Display the status of network service

Command:

```
sudo systemctl start network
```

Description:

Start the network service

Command:

```
sudo systemctl stop network
```

Description:

Stop the network service

Command:

```
sestatus -b
```

Description:

Display the current state of Booleans

Command:

```
getenforce
```

Description:

Reports whether SELinux **is enforcing, permissive or disabled**

> **Security-Enhanced Linux** (SELinux) is a security architecture for Linux systems that allows administrators to have more control over who can access the system

```
setenforce 0
```

```
getenforce
```

```
Output:

Permissive
```

```
setenforce 1
```

```
getenforce
```

```
Output:

Enforcing
```

- **Enforcing** - SELinux security policy is enforced.
- **Permissive** - SELinux prints warnings instead of enforcing.
- **Disabled** - No SELinux policy is loaded.

```
[manju@localhost ~]$ let a="36 + 5" ; echo $a

41



[manju@localhost ~]$ let a="20 + 50/10" ; echo $a

25



[manju@localhost ~]$ let a="20 - 50/10" ; echo $a

15



[manju@localhost ~]$ let a="20 * 50/10" ; echo $a

100
```

```
[manju@localhost ~]$ grep ^PASS /etc/login.defs

PASS_MAX_DAYS     99999

PASS_MIN_DAYS     0

PASS_MIN_LEN      5

PASS_WARN_AGE     7
```

```
[manju@localhost ~]$ grep PASS /etc/login.defs

#      PASS_MAX_DAYS     Maximum number of days a password may be used.

#      PASS_MIN_DAYS     Minimum number of days allowed between password changes.

#      PASS_MIN_LEN      Minimum acceptable password length.

#      PASS_WARN_AGE     Number of days warning given before a password expires.

PASS_MAX_DAYS     99999

PASS_MIN_DAYS     0

PASS_MIN_LEN      5

PASS_WARN_AGE     7
```

Command:

```
cut -d: -f1 /etc/passwd | column
```

Description:

List all local user accounts in column

Command:

```
mkdir ~/mydir1 ; touch ~/mydir1/myfiles1.txt
```

Description:

Create a directory "mydir1" and create a file "myfiles1.txt" in it

Command:

```
echo hi > file.md ; chmod 744 file.md
```

Description:

Create a file "file.md" and give only read access to others

```
[manju@localhost ~]$ ls -l $(which sudo)

---s--x--x. 1 root root 130776 Nov  5  2016 /bin/sudo
```

Command:

```
sestatus
```

Description:

Display the current status of the SELinux that is running on your system

Command:

```
ps -aef
```

Description:

Display full listing of processes on your system

Command:

```
sar
```

Description:

Command:

```
ulimit
```

Description:

Report the resource limit of the current user

```
Output:

Unlimited          The current user can consume all the resources the current system supports
```

2 types of resource limitation:

- **Hard resource limit:** The physical limit that the user can reach.
- **Soft resource limit:** The limit that is manageable by the user (**its value can go up to the hard limit**)

Command:

```
ulimit -a
```

Description:

Report all the resource limits for the current user

Command:

```
ulimit -s
```

Description:

Check the maximum stack size of the current user

Command:

```
ulimit -e
```

Description:

Check out the max scheduling priority of the current user

Command:

```
ulimit -u
```

Description:

Display the maximum number of user processes

Command:

```
ulimit -v
```

Description:

Check out the size of virtual memory

Command:

```
ulimit -n
```

Description:

Check out how many file descriptors a process can have

Command:

```
man limits.conf
```

Description:

Display the in-depth information on the limits.conf **configuration file**

Command:

```
sar -V
```

Description:

Display the sar version

Command:

```
sar -u 2 5
```

Description:

Report CPU details total 5 times with the interval of 2 seconds

Command:

```
sar -n DEV 1 3 | egrep -v lo
```

Description:

Report about network interface, network speed, IPV4, TCPV4, ICMPV4 network traffic and errors

Command:

```
sar -v 1 3
```

Description:

Report details about the process, kernel thread, i-node, and the file tables

Command:

```
sar -S 1 3
```

Description:

Report statistics about swapping

Command:

```
sar -b 1 3
```

Description:

Report details about I/O operations like transaction per second, read per second, write per second

Command:

```
sudo systemctl status firewalld
```

Description:

Display the status of the firewalld

Command:

```
sudo systemctl start firewalld
```

Description:

Start the firewalld **service**

> **firewalld** is a firewall management tool for Linux operating systems

Command:

```
firewall-config
```

Description:

Start the graphical firewall configuration tool

firewall-cmd

Command:

```
firewall-cmd --list-all-zones
```

Description:

List all zones

Command:

```
firewall-cmd --get-default-zone
```

Description:

Check the currently set default zone

Command:

```
firewall-cmd --list-services
```

Description:

Display currently allowed service on your system

Command:

```
firewall-cmd --list-ports
```

Description:

List the ports that are open on your system

Command:

```
firewall-cmd --zone=work --list-services
```

Description:

List services that are allowed for the public zone

Command:

```
mtr --report google.com
```

Description:

Provides information about the route that Internet traffic takes between the local system and a remote host (google.com**)**

Command:

```
sudo yum install samba
```

Description:

install Samba (CentOS**)**

> **Samba** is client/server technology that implements network resource sharing across operating systems. With Samba, files and printers can be shared across Windows, Mac and Linux/UNIX clients.

Command:

```
sudo firewall-cmd --add-service samba –permanent
```

Description:

Add Samba service to firewalld

Command:

```
zip q.zip q.txt
```

Description:

Create a zip file (q.zip**)**

Command:

```
unzip q.zip
```

Description:

Unzip a zip file (q.zip**)**

```
zipcloak q.zip

--------------------------------------------------------------------------------

# zipcloak prompts you for a password, and then ask you to confirm it:

   Enter password:
   Verify password:

...if the passwords match, it encrypts q.zip file

--------------------------------------------------------------------------------

unzip q.zip

#  When you try to unzip the q.zip file, it prompts you for the password before
allowing you to extract the file (q.txt) it contains
```

Command:

```
zgrep -l "Einstein" *
```

Description:

Display the names of the files with the word (Einstein**) present in it**

Command:

```
zipsplit -n 1048576 q.zip
```

Description:

Split q.zip **file to create a sequence of zipfiles** (q1.zip, q2.zip…..) – each no larger than **1048576 bytes** (one megabyte)

```
You could concatenate (q1.zip, q2.zip…..) into a new file, w.zip, with the
command:

                            cat q*.zip > w.zip
```

Command:

```
mtr google.com
```

Description:

Test the route and connection quality of traffic to the destination host **google.com**

Command:

```
route
```

Description:

Display IP routing table of a Linux system

Command:

```
nmcli dev status
```

Description:

View all your network devices

Command:

```
nmcli con show
```

Description:

Check network connections on your system

Command:

```
ss -ta
```

Description:

List all TCP ports (**sockets**) that are open on a server

Command:

```
ss -to
```

Description:

Display all active TCP connections together with their timers

Command:

```
type -a alias
```

Description:

Check Bash Aliases in Linux

```
# Difference between %Y and %y is %Y will print 4 digits while %y will print the last 2 digits of the year.

echo "We are in the year = $(date +%Y)"
echo "We are in the year = $(date +%y)"
```

```
# Difference between %B and %b is, %B will print full month name while %b will print abbreviated month
name.

echo "We are in the month = $(date +%b)"
```

```
echo "We are in the month = $(date +%B)"


# Difference between %A and %a is, %A will print full Weekday name while %a will print abbreviated weekday
name.


echo "Current Day of the week = $(date +%A)"
echo "Current Day of the week = $(date +%a)"
```

echo "Date using %D = $(date +%D)"

echo "Date using %F = $(date +%F)"

Date using %D = 08/15/21

Date using %F = 2021-08-15

echo "current time in 24 hour format = $(date +%T)"

current time in 24 hour format = 01:27:46

echo "current time in 12 hour format = $(date +%r)"

current time in 12 hour format = 01:27:47 AM

```
# Print yesterday's date and time.
echo "Yesterday = $(date -d "Yesterday")"



# Print Tomorrow date and time.
```

```
echo "tomorrow = $(date -d "tomorrow")"


# Find what is the date and time before 10 days from now.
echo "Before 10 days = $(date -d "tomorrow -10 days")"


# Find last month and next month
echo "Last month = $(date -d "last month" "%B")"
echo "Next month = $(date -d "next month" "%B")"


# Find last year and next year
echo "Last Year = $(date -d "last year" "+%Y")"
echo "Next Year = $(date -d "next year" "+%Y")"
```


Command:


```
ls -lai /
```


Description:

Get the number of inodes of files in a directory (**root directory**)

Command:


```
sudo du --inode /
```


Description:

Get the total number of inodes in the root directory

Command:

```
ss -o state established '( sport = :http or sport = :https )'
```

Description:

Get the list of all clients connected to HTTP (Port 80) or HTTPS (Port 443)

Command:

```
ss -tn src :80 or src :443
```

Description:

List the numerical port numbers

Command:

```
sudo yum install putty
```

Description:

Install **PuTTy** on CentOS

Command:

```
sudo watch netstat -tulpn
```

Description:

Watch TCP and UDP Open Ports in Real-Time

Command:

```
sudo watch ss —tulpn
```

Description:

Watch TCP and UDP Open Ports in Real-Time

Command:

```
timeout 5s ping google.com
```

Description:

Timeout a ping command after 5 seconds

Command:

```
yum install curl
```

Description:

Install curl on CentOS

Command:

```
ss -ua
```

Description:

List all UDP Connections

Command:

```
ss -lu
```

Description:

List all Listening UDP Connections

Command:

```
ss -p
```

Description:

Display the Process IDs related to socket connections

Command:

```
ss -4
```

Description:

Display IPv4 and IPv6 Socket Connections

Command:

```
ss -6
```

Description:

Display IPv6 connections

Command:

```
ss -at '( dport = :22 or sport = :22 )'
```

Description:

Filter Connections by Port Number

"The only way to learn a new programming language is by writing programs in it."

–Dennis Ritchie

```
[manju@localhost ~]$ echo {a..z}

a b c d e f g h i j k l m n o p q r s t u v w x y z



[manju@localhost ~]$ echo {z..a}

z y x w v u t s r q p o n m l k j i h g f e d c b a
```

```
[manju@localhost ~]$ echo {005..10}

005 006 007 008 009 010



mkdir 20{09..11}-{01..12}

# Create directories to group files by month and year
```

```
[manju@localhost ~]$ echo {05..12}

05 06 07 08 09 10 11 12



[manju@localhost ~]$ echo {12..5}

12 11 10 9 8 7 6 5
```

```
[manju@localhost ~]$ echo {12..05}

12 11 10 09 08 07 06 05



[manju@localhost ~]$ echo {x..z}{1..3}

x1 x2 x3 y1 y2 y3 z1 z2 z3



[manju@localhost ~]$ echo {0..10..2}

0 2 4 6 8 10



[manju@localhost ~]$ for i in {a..z..5}; do echo -n $i; done

afkpuz
```

```
[manju@localhost ~]$ ls *.txt; echo $_

12.txt  1.txt   2.txt  abc.txt    my.txt     phy.txt

13.txt  24.txt  3.txt  marks.txt  names.txt  mphy.txt
```

```
[manju@localhost ~]$ cut -d, -f2,1 <<<'Albert,Bob,John'

Albert,Bob

[manju@localhost ~]$ cut -d, -f2,2 <<<'Albert,Bob,John'

Bob

[manju@localhost ~]$ cut -d, -f2,3 <<<'Albert,Bob,John'

Bob,John
```
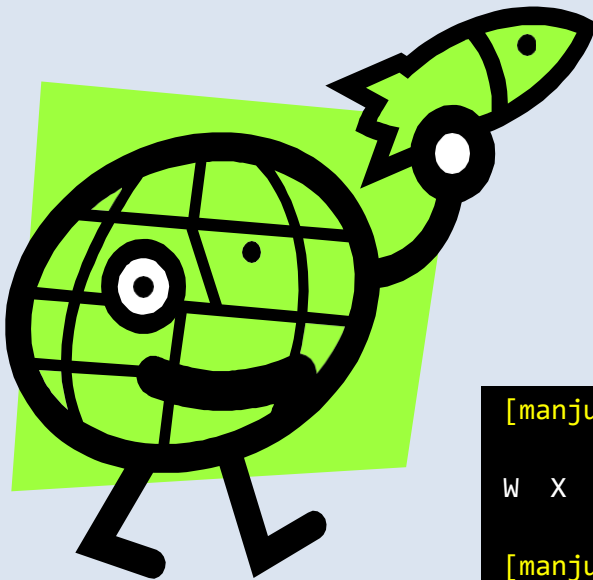
Coding theory is a branch of mathematics and computer science that studies the design, analysis, and application of codes for the purpose of error detection, correction, and confidentiality in communication, storage, and other information processing systems. It involves the study of methods for encoding data into codes that are resilient to errors and noise that may occur during transmission or storage, as well as methods for decoding received codes to recover the original data. Coding theory has applications in various fields, including telecommunications, computer networks, data storage, and cryptography.

```
[manju@localhost ~]$ x="W  X  Y   Z"; echo "$x"

W  X  Y   Z

[manju@localhost ~]$ x="W  X  Y   Z"; echo $x

W X Y Z
```

echo $x and echo "$x" yield different results

Quoting a variable preserves whitespace

```
[manju@localhost ~]$ let x=20+7; echo "The value of \"x\" is $x."

The value of "x" is 27.

[manju@localhost ~]$ x=100; let "x += 1"; echo "x = $x"

x = 101
```

```
[manju@localhost ~]$ x="a+b+c"; IFS=+; echo $x

a b c        The "+" sign will be interpreted as a separator




[manju@localhost ~]$ x="a-b-c"; IFS=-; echo $x

a b c        The "- " sign will be interpreted as a separator




[manju@localhost ~]$ x="a,b,c"; IFS=,; echo $x

a b c        The "comma" will be interpreted as a separator
```

```
free | grep Mem | awk '{ print $4 }'

# Display the unused RAM memory


                                    [manju@localhost ~]$ expr 5 \* 2 + 3

du -ach                             13 # 10 + 3

# Display (disk) file usage         [manju@localhost ~]$ expr 5 \* \( 2 + 3 \)

                                    25 # 5 * 5


readelf -h /bin/bash

# Display information and statistics about a designated elf binary
```

143

```
[manju@localhost ~]$ echo -e "\033[4mAlbert Einstein.\033[0m"

Albert Einstein.



[manju@localhost ~]$ echo -e "\033[1mAlbert Einstein.\033[0m"

Albert Einstein.
```

```
[manju@localhost ~]$ echo -e '\E[34;47mAlbert Einstein'; tput sgr0

Albert Einstein



[manju@localhost ~]$ echo -e '\E[33;44m'"Albert Einstein"; tput sgr0

Albert Einstein



[manju@localhost ~]$ echo -e '\E[1;33;44m'"Albert Einstein"; tput sgr0

Albert Einstein
```

```
[manju@localhost ~]$ x=2; y=3; echo $((2*$x + 3*$y))

13

[manju@localhost ~]$ x=2; y=3; echo $((2*x + 3*y))

13

[manju@localhost ~]$ let x=2+3 y=3+2; echo $x $y

5 5
```

Command:

```
sdiff phy.txt score.txt
```

Description:

Show Difference between Two Files (**phy.txt and score.txt**)

Command:

```
history -c
```

Description:

Delete or clear all the entries from bash history

Command:

```
ping -c 5 www.google.com
```

Description:

The ping test will stop after sending 5 packets

```
# count number of lines in each .txt file

ls *.txt | xargs wc -l

# count number of words in each .txt file

ls *.txt | xargs wc -w

# count number of characters in each .txt file

ls *.txt | xargs wc -c

# count lines, words and characters in each .txt file

ls *.txt | xargs wc
```

Command:

```
lslogins -u
```

Description:

Displays user accounts

Command:

```
systemctl list-units --type=service
```

Description:

List all loaded services on your system (whether active; running, exited or failed)

Command:

```
systemctl --type=service
```

Description:

List all loaded services on your system (whether active; running, exited or failed)

Command:

```
systemctl list-units --type=service --state=active
```

Description:

List all loaded but active services

Command:

```
systemctl --type=service --state=active
```

Description:

List all loaded but active services

Command:

```
systemctl list-units --type=service --state=running
```

Description:

List all running services (i.e., all loaded and actively running services)

Command:

```
systemctl --type=service --state=running
```

Description:

List all running services (i.e., all loaded and actively running services)

```
#scan a single port

nc -v -w 2 z 192.168.56.1 22

# scan multiple ports

nc -v -w 2 z 192.168.56.1 22 80

# scan range of ports

nc -v -w 2 z 192.168.56.1 20-25
```

Command:

```
cat /etc/resolv.conf
```

Description:

Find out your DNS Server IP address

Command:

```
less /etc/resolv.conf
```

Description:

Find out your DNS Server IP address

Command:

```
findmnt --poll --mountpoint /mnt/test
```

Description:

Monitor mount, unmount, remount and move actions on a directory (**i.e., on /mnt/test**)

Command:

```
uptime -p
```

Description:

Check Linux Server Uptime

Command:

```
uptime -s
```

Description:

Check Linux Server Starting Time

Command:

```
uptime -h
```

Description:

Display uptime's version information

Command:

```
grep -o -i Justin score.txt | wc -l
```

Description:

Count the number of times "**Justin**" appears in the file (**score.txt**)

Command:

```
crontab -r
```

Description:

Delete all crontab jobs

```
ADD=$(( 1 + 2 ))
echo $ADD
                                    3


MUL=$(( $ADD * 5 ))
echo $MUL
                                   15


SUB=$(( $MUL - 5 ))
echo $SUB
                                   10


DIV=$(( $SUB / 2 ))
echo $DIV
                                    5
MOD=$(( $DIV % 2 ))
echo $MOD


                                    1
```

Command:

```
expr length "This is myw3schools.com"
```

Description:

Find the length of a string (**This is myw3schools.com**)

```
echo '3+5' | bc
                                8

awk 'BEGIN { a = 6; b = 2; print "(a + b) = ", (a + b) }'

                        (a + b) =  8
```

Command:

```
factor 10
```

Description:

Decompose an integer (**10**) into prime factors

Command:

```
ps -e
```

Description:

Display every active process on a Linux system

Command:

```
ps -x
```

Description:

Display User Running Processes

Command:

```
ps -fU manju
```

Description:

Display a user's processes by user name (**manju**)

Command:

```
ps -fu 1000
```

Description:

Display a user's processes by real user ID (**RUID**)

Command:

```
ps -U root -u root
```

Description:

Display every process running with root user privileges (real and effective ID)

```
    echo -e "The following users are logged on the system:\n\n $(who)"


        manju     :0          Aug 15 03:31 (:0)

        manju     pts/1       Aug 15 03:32 (:0)

```

```
date +%d\-%m\-%Y

# Display the current date in DD-MM-YY format



date +%m\/%d\/%Y

# Display the current date in MM/DD/YYYY format
```

```
date -d "-7 days"

# Display the date 7 days before current date



date -d "7 days"

# Display the date 7 days after current date
```

```
ls -a ~/

# Display all files – including the hidden files


[manju@localhost ~]$ gnome-calculator -s 23500*10%

2350
```

$$23500 \times \frac{10}{100} = 2350$$

```
shuf -i 1-5 -n 5
```

This command produces numbers, in this case will generate 5 numbers between 1 and 5.

```
[manju@localhost ~]$ shuf -i 1-5 -n 5 | awk '{ sum+=$1;print $1} END {print "Sum";print sum}'
4
5
2
3
1
Sum
15
```

```
[manju@localhost ~]$ find ./ -name "*.zip" -type f
./1.zip
./--encrypt.zip        Find File with .zip extension
./my.zip
```

```
find -type f -mtime 0
# Find files updated in the last 24 hours


find -type f -newermt `date +%F`
# Find files updated today


find -type f -newermt `date +%F -d yesterday`
# Find files modified by yesterday


find -type f -mtime -3
# Find files updated in the last 72 hours
```

```
find -type f -newermt 2021-03-01

# Find files modified after March 1, 2021


find -type f -newermt `date +%F -d -3hours`

# Find files updated in the last 3 hours
```

```
find -type f -mmin -180
```

```
if [ 15 -gt 25 ]

then

 echo "True"

else

 echo "False"

fi
```

```
# Output:

                    False
```

```
for i in 1 2 3 4 5

do

echo "Albert"

done
```

```
# Output:

                Albert

                Albert

                Albert

                Albert
```

```
if test 5 -eq 6

then

echo "True"

else

echo "False"

fi
```

```
# Output:
                        False
```

```
yum deplist httpd

# List dependencies of a package "apache"
```

```
yum reinstall httpd

# Reinstall (corrupted) package "apache"
```

```
[manju@localhost ~]$ echo "i=3;++i" | bc

4

[manju@localhost ~]$ echo "i=3;--i" | bc

2

[manju@localhost ~]$ echo "i=4;i" | bc

4
```

```
help cd

# Display the complete information about the cd command


help -d cd
```
```
                Display the short description about the cd command
```

```
help -s cd

# Display the syntax of the cd command
```

```
mkdir -m a=rwx myfiles

  Create the myfiles directory, and set its file mode (-m) so that
      all users (a) may read (r), write (w), and execute (x) it.
```

```
whatis -w make*

# Display the use of all commands which start with make



who -d -H

# Print out the list of all dead processes
```

```
finger -s manju

# Display login details and IDLE status about an user "manju"



last -F

# Display the login and logout time including the dates



locate --regex -i "(\.mp3|\.txt)"

# Search for all .mp3 and .txt files on your system and ignore case
```

```
systemctl -l -t service | less

# List all Systemd services


locate "*.txt" -n 20

# Display 20 results for the searching of file ending with .txt
```

```
locate -c [.txt]*

# Count files ending with .txt and display the result


ls -l . | egrep -c '^-'

# Count the number of files within a current directory
```

```
du -kx | egrep -v "\./.+/" | sort -n

# Search for the largest directories


ls -al --time-style=+%D | grep `date +%D`

# List today's files only
```

```
find . -name '*.gz'

# Find all the gzip archives in the present working directory


du -hsx * | sort -rh | head -10

# Check the top 10 files that are eating out your space
```

```
mpstat

# Display processor and CPU statistics


mpstat -P ALL

# Display processor number of all CPUs
```

```
mpstat 1 5
```

This command will print 5 reports with 1 second time interval

```
[manju@localhost ~]$ printf "%s\n" "Albert Einstein"

Albert Einstein
```

Extract the substring

```
x=HelloWorld;
echo `expr substr $x 6 10`


# Output: World
```

```
find . -size 0k

# Find all empty files in current directory



ls -F | grep / | wc -l

# Count the number of directories in a directory
```

```
fdisk -s /dev/sda

# Display the size of the disk partition in Linux
```

Command:

```
sh <(curl https://nixos.org/nix/install) --daemon
```

Description:

Install Nix Package Manager in Linux

Command:

```
locale
```

Description:

View System Locale in Linux

Command:

```
locale -a
```

Description:

```
cat score.txt


                            Justin-40



cat score.txt | tr [:lower:] [:upper:]


                            JUSTIN-40



cat score.txt | tr [a-z] [A-Z] >output.txt
cat output.txt


                            JUSTIN-40
```

```
cat domainnames.txt


                    www. google. com
                     www. fb. com
                   www. mactech. com


cat domainnames.txt | tr -d ' '
                      www.google.com
                       www.fb.com
                     www.mactech.com
```

Remove the spaces in the domain names

```
cat domainnames.txt

                              www.google.....com
                                 www.fb.com
                              www.mactech.Com


cat domainnames.txt | tr -s ''

                              www.google.com
                                 www.fb.com
                              www.mactech.Com


echo "My UID is $UID"


                              My UID is 0



echo "My UID is $UID" | tr " "  "\n"


                                  My
                                  UID
                                  is
                                  0
```

A space into a **" : "** character

```
echo "myw3schools.com  =>Linux-Books,Src,Tutorials" | tr " " ":"


            myw3schools.com:=>Linux-Books,Src,Tutorials
```

Command:

```
!sud
```

Description:

Re-execute previously used command

Command:

```
!sudo
```

Description:

Re-execute previously used command

Command:

```
cut -d: -f1 < /etc/passwd | sort | xargs
```

Description:

Generate a compact list of all Linux user accounts on the system

Command:

```
zcat phy.txt.gz myfiles.txt.gz
```

Description:

View multiple compressed files (**phy.txt.gz and myfiles.txt.gz**)

Command:

```
find . -type f -name "*.php"
```

Description:

Find all php files in a directory

```
mkdir /tmp/DOCUMENTS

# Create a directory 'DOCUMENTS' under "/tmp" directory
```

Command:

```
find . -type f -perm 0777 -print
```

Description:

Find all the files whose permissions are 777

Command:

```
find / -type f ! -perm 777
```

Description:

Find all the files without permission 777

Command:

```
find / -perm /g=s
```

Description:

Find all SGID set files

Command:

```
find / -perm /u=r
```

Description:

Find all Read-Only files

Command:

```
find / -perm /a=x
```

Description:

Find all Executable files

```
[manju@localhost ~]$ echo "ALBERT" | awk '{print tolower($0)}'
albert
                    Convert text from upper case to lower case
```

Command:

```
find . -type f -name "phy.txt" -exec rm -f {} \;
```

Description:

Find and remove **phy.txt** File

Command:

```
[manju@localhost ~]$ echo "Phone number: 55602369" | tr -cd [:digit:]
55602369
```

Get the digits from string

```
find . -type f -name "*.txt" -exec rm -f {} \;
```

Description:

To find and remove multiple **.txt** files

Command:

```
find . -type f -name "*.mp3" -exec rm -f {} \;
```

Description:

To find and remove multiple **.mp3** files

Command:

```
find /tmp -type d -empty
```

Description:

Find all Empty Directories

Command:

```
find /tmp -type f -name ".*"
```

Description:

File all Hidden Files

```
[manju@localhost ~]$ echo "Phone number: 55602369" | tr -d [:digit:]
Phone number:
                        Remove all digits from string
```

Command:

```
find / -mtime 50
```

Description:

Find Last 50 Days Modified Files

Command:

```
find / -atime 50
```

Description:

Find Last 50 Days Accessed Files

Command:

```
find / -cmin -60
```

Description:

Find Changed Files in Last 1 Hour

Command:

```
find / -mmin -60
```

Description:

Find Modified Files in Last 1 Hour

Command:

```
find / -amin -60
```

Description:

Find Accessed Files in Last 1 Hour

Command:

Command:

```
type cat
```

Description:

Identifies whether the "cat" command is a shell built-in command, subroutine, alias, or keyword.

```
find / -size 50M
```

Description:

Find all 50MB files

Command:

```
find / -type f -size +100M -exec rm -f {} \;
```

Description:

Find and Delete 100MB Files

Command:

```
find / -type f -name *.mp3 -size +10M -exec rm {} \;
```

Description:

Find all .mp3 files with more than 10MB and delete them

```
ls -l --color

# List the files in current directory (with colorized output)


info df

# Loads the "df" info page


ls /usr/include

# List the Header files for compiling C programs


ls /usr/local

# List the Locally installed files


ls /usr/bin/d*

# List all files whose names begin with the letter "d" in the /usr/bin directory
```

```
[manju@localhost ~]$ ls .b*

.bash_history   .bash_logout   .bash_profile   .bashrc
```

```
[manju@localhost ~]$ ls [a-h]*

all           DICT  file1         file2         file34.txt   file.md       foo1.txt

allfiles.txt  echo  file123.txt   file23.txt    file3.txt    first.bash

bu.txt        file  file1.txt     file2.txt     FILE.backup  first.txt
```

```
[manju@localhost ~]$ touch hello.cpp; touch hello.f99

[manju@localhost ~]$ ls *.?[9p]?

hello.cpp  hello.f99
```

```
ls /usr

# List the /usr directory



ls ~ /usr

# List the user's home directory and the /usr directory



[manju@localhost ~]$ echo f*    Display any file beginning with "f"

file file1 file123.txt file1.txt file2 file23.txt file2.txt file34.txt file3.txt
file.md first.bash first.txt foo1.txt


                                  Display any file beginning with "f" followed by
[manju@localhost ~]$ echo f*.txt  any characters and ending with ".txt"

file123.txt file1.txt file23.txt file2.txt file34.txt file3.txt first.txt foo1.txt



sudo vim myfiles.txt

# Open a file "myfiles.txt" using Vim editor



[manju@localhost ~]$ for ((i=0;i<8;i++)); do echo $((i)); done

0

1

2

3

4

5

6

7
```

176

Command:

```
cat /proc/sys/fs/file-max
```

Description:

Find Linux Open File Limit

Command:

```
ulimit -Hn
```

Description:

Check Hard Limit in Linux

Command:

```
ulimit -Sn
```

Description:

Check Soft Limits in Linux

Command:

```
timedatectl  status
```

Description:

Display the current time and date on your system

Command:

```
timedatectl list-timezones
```

Description:

View all available timezones

Command:

```
timedatectl list-timezones |  egrep  -o "Asia/B.*"
timedatectl list-timezones |  egrep  -o "Europe/L.*"
timedatectl list-timezones |  egrep  -o "America/N.*"
```

Description:

Find the local timezone according to your location

Command:

```
timedatectl set-timezone "Asia/Kolkata"
```

Description:

Set your local timezone in Linux

Command:

```
swapon --summary
```

Description:

View a summary of swap space usage by device

Command:

```
cat /proc/swaps
```

Description:

Check swap usage information

```
# start recording of Linux terminal

script history_log.txt

                    Script started, file is history_log.txt

exit

                    Script done, file is history_log.txt
```

Command:

```
dir -shl
```

Description:

List files and their allocated sizes in blocks

Command:

```
less /proc/sys/dev/cdrom/info
```

Description:

Display information about CD-ROM

```
while true; do date >> date.txt ; sleep 5 ; done &

cat date.txt
                        Mon Aug 16 03:05:36 PDT 2021
                        Mon Aug 16 03:05:41 PDT 2021
                        Mon Aug 16 03:05:46 PDT 2021
                        Mon Aug 16 03:05:51 PDT 2021
```

"Don't write better error messages, write code that doesn't need them."

– Jason C. McDonald

```
[manju@localhost ~]$ echo hello > 1.txt

[manju@localhost ~]$ echo world > 2.txt

[manju@localhost ~]$ echo program > 3.txt

[manju@localhost ~]$ cat 1.txt

hello

[manju@localhost ~]$ cat 2.txt

world

[manju@localhost ~]$ cat 3.txt

program

[manju@localhost ~]$ cat 1.txt 2.txt 3.txt

hello

world

program

[manju@localhost ~]$ cat 1.txt 2.txt 3.txt >all

[manju@localhost ~]$ cat all

hello

world

program


 strings /usr/bin/passwd

 # Display the readable character strings from the /usr/bin/passwd


ls -lrS /etc

# List the biggest file in /etc
```

```
cat /etc/passwd >> myfiles.txt

# Create a file named myfiles.txt that contains the contents of myfiles.txt followed by the contents of /etc/passwd


[manju@localhost ~]$ ls /etc/*.conf

/etc/asound.conf                    /etc/kdump.conf         /etc/radvd.conf

/etc/autofs.conf                    /etc/krb5.conf          /etc/request-key.conf

/etc/autofs_ldap_auth.conf          /etc/ksmtuned.conf      /etc/resolv.conf

/etc/brltty.conf                    /etc/ld.so.conf         /etc/rsyncd.conf

/etc/cgconfig.conf                  /etc/libaudit.conf      /etc/rsyslog.conf

/etc/cgrules.conf                   /etc/libuser.conf       /etc/sestatus.conf

/etc/cgsnapshot_blacklist.conf      /etc/locale.conf        /etc/sos.conf

/etc/chrony.conf                    /etc/logrotate.conf     /etc/sudo.conf

/etc/dleyna-server-service.conf     /etc/man_db.conf        /etc/sudo-ldap.conf

/etc/dnsmasq.conf                   /etc/mke2fs.conf        /etc/sysctl.conf

/etc/dracut.conf                    /etc/mtools.conf        /etc/tcsd.conf

/etc/e2fsck.conf                    /etc/nfsmount.conf      /etc/updatedb.conf

/etc/fprintd.conf                   /etc/nsswitch.conf      /etc/usb_modeswitch.conf

/etc/fuse.conf                      /etc/ntp.conf           /etc/vconsole.conf

/etc/GeoIP.conf                     /etc/numad.conf         /etc/wvdial.conf

/etc/host.conf                      /etc/oddjobd.conf       /etc/yum.conf

/etc/idmapd.conf                    /etc/pbm2ppa.conf

/etc/ipsec.conf                     /etc/pnm2ppa.conf
```

Display configuration files located in /etc

```
ls /dev/sd*

/dev/sda  /dev/sda1  /dev/sda2  /dev/sda3
```

Display SATA device files

```
echo \$USER

# $USER


echo -e "2+2\t=4" ; echo -e "12+12\t=24"
```

```
2+2   =4

12+12 =24
```

```
echo Hello && echo World

Hello

World
```

```
echo Hello ; echo World
```

```
Hello

World
```

```
echo Hello || echo Hi ; echo World
```

```
Hello

World
```

```
rm myfiles.txt && echo It worked! || echo It failed!

It worked!

rm files.txt && echo It worked! || echo It failed!

rm: cannot remove 'files.txt': No such file or directory

It failed!
```

```
pwd ; pwd

/home/manju

/home/manju
```

Execute the pwd command twice

```
a=$(pwd)
echo "Current working directory is : $a"


                         /home/manju
```

Command:

```
echo *.jpeg
```

Description:

Print all **.jpeg** files

Command:

```
echo 'linux' | fold -w1
```

Description:

Break down a word (**linux**) into individual

```
l
i
n
u
x
```

Command:

```
find . -user root
```

Description:

Output the files with respect of the user **(root)** owned files in the current directory

Command:

```
strace pwd
```

Description:

Trace a command **(pwd)** execution

Command:

```
top -u manju
```

Description:

Display specific User (**manju**) process details

3 characteristics of big data:

- **Volume** — How much data is there?
- **Variety** — How diverse is different types of data?
- **Velocity** — At what speed is new data generated?

```
[manju@localhost ~]$ netstat -plunt

# print all listening ports


[manju@localhost ~]$ netstat -plunt | grep 8080

# check if server is listening on port 8080 or not


[manju@localhost ~]$ netstat -s

# list statistics of all ports
```

```
[manju@localhost ~]$ cat myfiles.txt

Hello World

[manju@localhost ~]$ cat myfiles.txt | tr ' ' '\n'

Hello

World
```

```
find . -name "*.txt"

# Find files that end in .txt in the current directory and all subdirectories
```

```
find /etc > 12.txt

# Find all files in /etc and place the list in 12.txt


find . -newer file1.txt

# Find files that is newer than file1.txt


[manju@localhost ~]$ date +'%A %d-%m-%Y'

Tuesday 19-04-2022


[manju@localhost ~]$ date -d '2022-04-01 + 2000000000 seconds'

Thu Aug 16 03:33:20 PDT 2085
```

```
find /etc -type f -name '*.txt' | wc -l

# Print the number of .txt files in /etc and all its subdirectories
```

```
[manju@localhost ~]$ cat myfiles.txt

Hello World

[manju@localhost ~]$ grep -E 'o*' myfiles.txt

Hello World

[manju@localhost ~]$ grep -E 'o+' myfiles.txt

Hello World
```

```
[manju@localhost ~]$ echo Albert Einstein | sed 's/Albert/&&/'

AlbertAlbert Einstein


[manju@localhost ~]$ echo Albert Einstein | sed 's/Einstein/&&/'

Albert EinsteinEinstein
```

```
[manju@localhost ~]$ echo Albert | sed 's_\(Alb\)_\1ert_'

Albertert

[manju@localhost ~]$ echo Albert | sed 's_\(Alb\)_\1ert \1_'

Albert Albert
```

```
[manju@localhost ~]$ echo -e 'Albert\tis\tScientist'

Albert   is       Scientist


[manju@localhost ~]$ echo -e 'Albert\tis\tScientist' | sed 's_\s_ _g'

Albert is Scientist
```

```
[manju@localhost ~]$ cat myfiles.txt

Hello World



[manju@localhost ~]$ cat myfiles.txt | sed 's/ll\?/A/'

HeAo World
```

```
[manju@localhost ~]$ cat myfiles.txt

Hello World


[manju@localhost ~]$ cat myfiles.txt | sed 's/l\{2\}/A/'

HeAo World
```

echo Albert `echo -n Einstein`

Albert Einstein

```
[manju@localhost ~]$ test 50 -gt 15 ; echo $?

0        True: 50 is greater than 15


[manju@localhost ~]$ test 5 -gt 15 ; echo $?

1        False: 5 is not greater than 15


[manju@localhost ~]$ test 5 -lt 15 ; echo $?

0        True: 5 is lesser than 15



[manju@localhost ~]$ test 50 -gt 15 && echo true || echo false

true

[manju@localhost ~]$ test 5 -gt 15 && echo true || echo false

false
```

```
[manju@localhost ~]$ [ 50 -gt 15 ] && echo true || echo false

true


[manju@localhost ~]$ [ 5 -gt 15 ] && echo true || echo false

false
```

```
[manju@localhost ~]$ [ 100 -gt 10 -a 100 -lt 150 ] && echo true || echo false

true


[manju@localhost ~]$ [ 100 -gt 10 -a 100 -lt 15 ] && echo true || echo false

false
```

```
[manju@localhost ~]$ a=2; b=a; eval c=\$$b; echo $c

2
```

```
[manju@localhost ~]$ date

Tue Apr 19 02:55:39 PDT 2022


[manju@localhost ~]$ date --date="1 week ago"

Tue Apr 12 02:55:05 PDT 2022
```

Command:

```
uname -or
```

Description:

Find Out Linux Kernel Version

Command:

```
uname -a
```

Description:

Print linux system information

Command:

```
cat /proc/version
```

Description:

Display some of your system information including the Linux kernel version

Command:

```
cat /etc/centos-release
```

Description:

Find Out Linux Distribution Name and Release Version

Command:

```
fuser .
```

Description:

Displays the PIDs of processes currently accessing your current working directory

Command:

```
fuser -v -m .bashrc
```

Description:

Determine which processes are accessing your **~.bashrc file**

Command:

```
sudo fuser --list-signals
```

Description:

Displays all the possible signals that can be used with the fuser tool

Command:

```
sudo fuser -k -HUP /boot
```

Description:

Sends the HUP signal to all processes that have your **/boot directory** open

Command:

```
ls -al
```

Description:

List all the files with the file permissions, the number of links to that file, the owner of the file, the group of the file, the file size in bytes, the file's last modified datetime and the file name

Command:

```
echo "shutdown -h now" | at -m 23:55
```

Everyone can now read the file

chmod a+r myfiles.txt

Everyone can now read and write the file

chmod a+rw myfiles.txt

Others (not the owner, not in the same group of the file) cannot read, write or execute the file

chmod o-rwx myfiles.txt

Description:

Shutdown the system at 23:55 today

Command:

```
echo "updatedb" | at -m 23.55
```

Description:

Creates and updates the database of file names used by locate

Run **updatedb** at 23:55 today

Command:

```
echo $(ls -al)
```

Description:

Execute command "ls -al" and print the result to the standard output

Command:

```
top -b -o +%MEM | head -n 22
```

Description:

Display the top 15 processes sorted by memory use in descending order

Command:

```
top -b -o +%MEM | head -n 22 > report.txt
```

Description:

Redirect the output to a file (**report.txt**) for later inspection

Command:

```
ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head
```

Description:

Check Top Processes sorted by RAM or CPU Usage in Linux

Command:

```
find . -type f \( -name "*.sh" -o -name "*.txt" \)
```

Description:

Find all files in the current directory with **.sh** and **.txt file** extensions

Command:

```
find . -type f \( -name "*.sh" -o -name "*.txt" -o -name "*.c" \)
```

Description:

Find all files in the current directory with **.sh, .c** and **.txt file** extensions

Description:

Find files edited more than 3 days ago.

Command:

```
find . -type f -mtime +3
```

Description:

Find files edited in the last 24 hours.

Command:

```
find . -type f -mtime -1
```

Description:

Find files that have more than 100 characters (bytes) in them.

Command:

```
find . -type f -size +100c
```

Description:

Find files bigger than 100 KB but smaller than 1 MB.

Command:

```
find . -type f -size +100k -size -1M
```

Description:

Deletes all the files edited in the last 24 hours.

Command:

```
find . -type f -mtime -1 -delete
```

Description:

List all files including hidden files.

Command:

```
ls -a
```

Description:

List Files and Directories with "/" Character at the End.

Command:

```
ls -F
```

Description:

List Files in Reverse Order.

Command:

```
ls -r
```

Description:

Sort Files by File Size.

Command:

```
ls -lS
```

Description:

List Files with an inode number.

Command:

```
ls -i
```

Description:

Check the version of the ls command.

Command:

```
ls --version
```

Description:

List files under directory /tmp.

Command:

```
ls -l /tmp
```

Description:

Display UID and GID of files and directories.

Command:

```
ls -n
```

Description:

Find all 30 MB files.

Command:

```
find / -size 30M
```

Description:

Find files with sizes between 100 - 200MB.

Command:

```
find / -size +100M -size -200M
```

Description:

List directories larger than 20 KB.

Command:

```
find / -type d -size +20k
```

Description:

Find empty files and directories.

Command:

```
find ./ -type f -size 0
```

Description:

List files modified within the last 17 hours.

Command:

```
find . -mtime -17 -type f
```

Description:

*** List directories modified within the last 10 days.***

Command:

```
find . -mtime -10 -type d
```

Description:

List all files modified between 6 and 15 days ago in the home directory.

Command:

```
find /home -type f -mtime +6 -mtime -15
```

Description:

Display files with permission 777.

Command:

```
find -perm 777
```

Description:

List files owned by a user (manju).

Command:

```
find /home -user manju
```

Description:

Find all text files owned by user "manju".

Command:

```
find /home -user manju -iname "*.txt"
```

Description:

Find and list files and directories together with their permissions.

Command:

```
find -name "*.conf" | ls -l
```

Description:

List directories only.

Command:

```
ls -d */
```

Description:

List multiple files on a single line.

Command:

```
ls --format=comma
```

Description:

View the process of a specific user "manju".

Command:

```
ps -u manju
```

Description:

Execute a previous command starting with a specific letter "c".

Command:

```
!c
```

Description:

Display BIOS information (You need elevated permissions to run this).

Command:

```
dmidecode -t 0
```

Description:

Display CPU information (You need elevated permissions to run this).

Command:

```
dmidecode -t 4
```

Description:

View all the system logs.

Command:

```
gnome-system-log
```

Description:

Identify SSH Client Version.

Command:

```
ssh -V
```

Description:

Display total connect time of users.

Command:

```
ac -d
```

Description:

Display connect time for all the users.

Command:

```
ac -p
```

Description:

Diaplay connect time report for a specific user "manju".

Command:

```
ac -d manju
```

Description:

Display the modules compiled inside Apache.

Command:

```
httpd -l
```

Description:

*** View Processes Owned by Current User.***

Command:

```
ps U $USER
```

Description:

Display the information about the filesystem Type.

Command:

```
df -Tha
```

Description:

Display Active Connections with Process ID and Program Name.

Command:

```
netstat -tap
```

Description:

Display RAW network statistics.

Command:

```
netstat --statistics --raw
```

Linux User Mode

Command Line Interface

Graphical user interface

LILO (**Linux Loader**)

Load Linux into memory and start the OS

The maximum length for a filename under Linux is 255 bytes.

Logical Volume Manager (LVM)

Manage hard drives and other storage devices on linux

```
[manju@localhost ~]$ PS1="Please enter a command: "

Please enter a command: date

Thu Apr 21 20:51:19 PDT 2022


Please enter a command: cal

      April 2022

Su Mo Tu We Th Fr Sa

                1  2

 3  4  5  6  7  8  9

10 11 12 13 14 15 16

17 18 19 20 21 22 23

24 25 26 27 28 29 30



Please enter a command:
```

```
ps -aux | grep 'httpd'

# Check for the httpd process
```

```
[manju@localhost ~]$ ls /var/spool
```

/var/spool holds spooled files such as those generated for printing jobs and network transfers

```
abrt   abrt-upload   anacron   at   cron   cups   lpd   mail   plymouth   postfix
```

```
[manju@localhost ~]$ ls /usr/share/man
```

/usr/share/man holds the online Man files

```
ca  en  hu  ko     man1x  man3p  man5   man7   man9     pl      ro  tr   zh_TW

cs  es  id  man0p  man2   man3x  man5x  man7x  man9x    pt      ru  uk

da  fr  it  man1   man2x  man4   man6   man8   mann     pt_BR   sk  zh

de  hr  ja  man1p  man3   man4x  man6x  man8x  overrides pt_PT  sv  zh_CN
```

```
[manju@localhost ~]$ ls /etc/gdm
```
List the contents of GDM configuration directory

```
custom.conf   Init   PostLogin   PostSession   PreSession   Xsession
```

```
ls /etc/gconf
```

```
# List the GConf configuration files
```

```
ls /usr/share/gnome
```

```
# List the files used by GNOME applications
```

```
[manju@localhost ~]$ ls /etc/sysconfig
```
List the **system configuration** files

atd	firewalld	libvirt-guests	qemu-ga	samba
authconfig	grub	man-db	radvd	saslauthd
autofs	init	modules	raid-check	selinux
cbq	ip6tables-config	netconsole	rdisc	smartmontools
cgred	iptables-config	network	readonly-root	sshd
console	irqbalance	network-scripts	rpcbind	sysstat
cpupower	kdump	nfs	rpc-rquotad	sysstat.ioconf
crond	kernel	ntpd	rsyncd	virtlockd
ebtables-config	ksm	ntpdate	rsyslog	virtlogd
fcoe	libvirtd	pluto	run-parts	wpa_supplicant

```
ls /etc/rc.d
```

```
# List the system startup and shutdown files
```

```
[manju@localhost ~]$ ls /etc/init.d
```
/etc/init.d holds network scripts to start up network connections

```
functions   netconsole   network   README
```

Important features of Linux Operating System

- Free and Open Source
- Portable and More secure
- Robust and Adaptable

Linux Kernel

The core interface between a computer's hardware and its processes

Manage RAM memory.

+

Manage the processor time.

+

Manage access and use of the various peripherals that are connected to the computer.

Swap space

A space on a hard disk that is used when the amount of **physical memory** (RAM) is full

```
[manju@localhost ~]$ cd /etc

[manju@localhost etc]$ pwd

/etc


[manju@localhost etc]$ cat /etc/hosts    /etc/hosts contains hostnames with their ip address

127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4

::1          localhost localhost.localdomain localhost6 localhost6.localdomain6


chmod u+w myfiles.txt

# Add user write privileges


chmod u-w myfiles.txt

# Remove user write privileges

                                          chmod go-r myfiles.txt

                                          # Remove group and others read privileges
chmod g+w myfiles.txt

# Add group write privileges


chmod g=r myfiles.txt

# Allow only the group read privileges


chmod o+x myfiles.txt

# Add execute privileges for others


chmod a+x myfiles.txt

# Add execute privileges for everyone


chmod a=xr myfiles.txt

# Allow read and execute only to everyone
```

```
ps -L 3315

# List all threads for a particular process (with process ID 3315)


ps aux --sort pmem

# Check the memory status


awk '/Hello/' myfiles.txt

# Find "Hello" in myfiles.txt


awk -F: '{ print $1 }' /etc/passwd | sort

# Display a sorted list of the login names of all users


awk 'END { print NR }' myfiles.txt

# Counts lines in myfiles.txt


[manju@localhost ~]$ awk 'BEGIN { for (i = 1; i <= 7; i++) print int(101 * rand()) }'

24

29

85

15

59

19

81
```

Prints seven random numbers from zero to 100

```
ls -lg *.txt | awk '{ x += $5 } ; END {print "total bytes:" x }'

# Prints the total number of bytes used by all .txt files
```

Random-access memory	Virtual memory
The internal memory of the CPU for storing data, program and program result.	A storage area that holds the files on your hard drive for retrieval when a computer runs out of RAM

Process States in Linux:

- **Ready:** a new process is created and is ready to run.

- **Running:** The process is being executed.

- **Wait:** The process is waiting for input from the user.

- **Completed:** The process has completed the execution.

- **Zombie:** The process is terminated but information regarding the process still exists and is available in the process table.

Cron	Anacron
A service that enables us to run scheduled jobs in Linux/Unix systems every minute.	A service that only enables us to run scheduled jobs in Linux/Unix systems on daily basis.

Command:

```
cat /etc/crontab
```

Description:

View system defined cron jobs

Command:

```
netstat --listen
```

Description:

Check which ports are in listening in Linux Server

Network Interface Card teaming is the process of combining multiple network cards together for performance, load balancing and to increase uptime.

Service	Default Port
DNS	53
SMTP	25
FTP	20 (**Data transfer**), 21 (**Connection established**)
SSH	22
DHCP	67/UDP (**dhcp server**), 68/UDP (**dhcp client**)
squid	3128

Hard Link	Soft Link
A mirror copy of the original file	A symbolic link to the original file
Contains the original content of the file	Does not contain the actual content but contains the **location of** the original file

```
ls /bin

# List the binaries and other executable programs


ls /boot

# List the files needed to boot the operating system


ls /dev

# List the device files - typically controlled by the operating system and the system administrators


ls /etc

# List the System configuration files


ls /lib

# List the System Libraries


ls /lib64

# List the System Libraries(64 bit)


ls /proc

# List the information about running processes


ls /sbin

# List the System administration binaries


ls /var/log

# List the Log files
```

mkdir mydir{1,2,3,4,5}

Create 5 new directories:

- **mydir1**
- **mydir2**
- **mydir3**
- **mydir4**
- **mydir5**

```
[manju@localhost ~]$ ls -l myfiles.txt

-rw-r--r--. 1 manju nath 12 Apr 19 20:22 myfiles.txt
```

Display the permissions for the file "myfiles.txt"

```
find . -mtime +1 -mtime -3

# Display files that are more than 1 day old - but less than 3 days old in the current directory


find . -name "s*" -ls

# Find files that start with the letter "s" and perform the command "ls" on them
```

```
find . -size +3M

# Find files that is larger than 3 megabytes



[manju@localhost ~]$ cat myfile.txt

ffff

b

eee

cc

[manju@localhost ~]$ cat myfile.txt | sort

b

cc

eee

ffff
```

```
[manju@localhost ~]$ touch file1; touch file2

[manju@localhost ~]$ ls file{1,2}

file1  file2
```

```
[manju@localhost ~]$ NUMLOGINS=$(who | grep $USER | wc -l)

[manju@localhost ~]$ echo You have $NUMLOGINS login sessions

You have 2 login sessions
```

Command:

```
chmod go-rwx myfiles.txt
```

Description:

Remove read write and execute permissions on the file **"myfiles.txt"** for the group and others

Command:

```
chmod a+rw myfiles.txt
```

Description:

Give read and write permissions on the file "myfiles.txt" **to all**

Command:

```
!-3
```

Description:

Repeats the third most recent command

```
[manju@localhost ~]$ echo $OSTYPE

linux-gnu          The current operating system you are using
```

Command:

```
df -i /dev/sda1
```

Description:

Check Inodes on File system

Command:

```
ls -il  myfiles.txt
```

Description:

Find Inode number of File (myfiles.txt)

Command:

```
getfacl myfiles.txt
```

Description:

Check ACL (Access control list) configured on a file (myfiles.txt)

SSH (**Secure Shell or Secure Socket Shell**) is a network protocol that gives users and system administrators a secure way to access a computer over an unsecured network.

3 standard streams in Linux:

Command:

```
du -sh /var/log/*
```

Description:

Check information of disk usage of files and directories on a machine.

Command:

```
ldd /bin/cp
```

Description:

Display dependencies of the "cp" command.

Command:

```
ldd -v /bin/cp
```

Description:

Display dependencies of the "cp" command with details.

Command:

```
ldd -u /bin/cp
```

Description:

Display unused direct dependencies of the "cp" command.

```
[manju@localhost ~]$ date; cal

Thu Apr 21 19:44:12 PDT 2022

      April 2022

Su Mo Tu We Th Fr Sa

                1  2

 3  4  5  6  7  8  9

10 11 12 13 14 15 16

17 18 19 20 21 22 23

24 25 26 27 28 29 30
```

> **date command** is executed
> followed by a **cal command**

-gt	Greater than
-lt	Lesser than
-ge	Greater than or equal to
-le	Lesser than or equal to
-eq	Equal to
-ne	Not equal to

```
[manju@localhost ~]$ date && cal

Thu Apr 21 19:44:21 PDT 2022

      April 2022

Su Mo Tu We Th Fr Sa

                1  2

 3  4  5  6  7  8  9

10 11 12 13 14 15 16

17 18 19 20 21 22 23

24 25 26 27 28 29 30
```

> **cal command** is executed
> only if the **date command** is
> successfully executed

```
[manju@localhost ~]$ ls *.c

hello.c  vim.c
```

```
[manju@localhost ~]$ ls *.[co]

hello.c  hello.o  vim.c
```

```
[manju@localhost ~]$ a=`ls *.c`; echo $a

hello.c main.c vim.c


[manju@localhost ~]$ test 50 -ge 15 && echo true || echo false

true


[manju@localhost ~]$ test 50 -ge 50 && echo true || echo false

true


[manju@localhost ~]$ test 20 -le 50 && echo true || echo false

true


[manju@localhost ~]$ test 20 -le 20 && echo true || echo false

true


[manju@localhost ~]$ test 30 -eq 30 && echo true || echo false

true


[manju@localhost ~]$ test 320 -eq 30 && echo true || echo false

false


[manju@localhost ~]$ test 30 -ne 30 && echo true || echo false

false


[manju@localhost ~]$ test 320 -ne 30 && echo true || echo false

true
```

Command:

```
cat /proc/net/dev
```

Description:

Display network adapters and statistics

Command:

```
cat /proc/mounts
```

Description:

Display the mounted file system

Command:

```
telinit 0
```

Description:

Shutdown the system

```
cd /home

# Takes you to the home directory


cd ..

# Takes you one folder back
```

```
ls * [0-9] *
```
Display the files and folders that contain numbers 0 to 9

```
iconv -l

# Display the lists of known ciphers


ls -lSr | more

# Display the size of the files and directories ordered by size


du -sk * | sort -rn

# Display the size of the files and directories ordered by size
```

Command:

ls -lh

Description:

Display permissions

Command:

yum list

Description:

List all packages installed on the system

Command:

yum clean packages

Description:

Clean all the saved packages

Command:

yum clean headers

Description:

Clean package headers

Command:

yum clean all

Description:

Clean all cached information

Command:

yum clean metadata

Description:

Clean Metadata

```
ip link show

# Display the link status of all interfaces


ps -eafw

# Display Linux tasks


lsof -p $$

# Display a list of files opened by processes
```

Command:

```
find /var -atime -90
```

Description:

Find files in the /var directory that have not been accessed in the last 90 days

Command:

```
find / -name core -exec rm {} \;
```

Description:

Search for core files in the entire directory tree and delete them as found without prompting for confirmation

Command:

```
who -r
```

Description:

Check current run level of a Linux server

Bash script:

```
for i in *linux*; do rm $i; done
```

Description:

Delete all the files in the current directory that contains the word "linux"

Command:

```
awk '{print}' myfiles.txt
```

Description:

Display the content of file (myfiles.txt)

```
# Wait for 5 seconds

sleep 5s

# Wait for 5 minutes

sleep 5m                    Sleep commands used to introduce
                              wait time in scripts
# Wait for 5 hours

sleep 5h

# Wait for 5 days

sleep 5d
```

Command:

```
ln myfiles.txt hardF1
```

Description:

Create hard-link to myfiles.txt

Command:

```
cat hardF1
```

Description:

Check content of the hard link - hardF1

Command:

```
ln myfiles.txt softF1
```

Description:

Create Soft-link to myfiles.txt

Command:

```
cat softF1
```

Description:

Check content of the soft link - softF1

Foreground processes	Background processes
Require a user to start them or to interact with them.	Run independently of a user.

Command:

```
ps -p 13
```

Description:

Display information about the process with process ID – 13

Command:

```
ulimit -f 100
```

Description:

Set the file size limit to 51,200 bytes

Command:

```
lsmod
```

Description:

Find out what kernel modules are currently loaded

Absolute path	Relative path
The path of a file or directory from the root directory.	The path of a file or directory from the present working directory.

Command:

```
sudo yum install php
```

Description:

Install php version 7.2

Command:

```
php -r 'echo "Hello World\r\n";'
```

Description:

Run a PHP statement from the command line without creating a file

Command:

```
php -a
```

Description:

Start a PHP interactive shell

Command:

du -h -d 1 /

Description:

Display disk usage of all top-level directories

Command:

```
yum install man
```

Description:

Install man package in Centos

Command:

```
man -f ls
```

Description:

Display man Pages and Print Short Description of the ls command

```
man -a ls

# Display all man Pages of the ls command


man -k ls

# Allows users to search the short command descriptions and manual page names for ls command


man -w ls

# Displays the location of the manual page of the ls command
```

```
cat /etc/redhat-release

# Display Linux distribution name and version


ls ~

# Display the contents of the home directory


ls ../

# Display the contents of the parent directory
```

Command:

ps -U root -u root

Description:

Display all process running under the root user account

Command:

cal -1

Description:

Display current month calendar

Command:

cal -j

Description:

Print the calendar in day numbers

Command:

su

Description:

Used to switch from one account to another

Command:

```
nmcli connection show
```

Description:

Display what are the network connection connected in our system

Command:

```
ps aux | grep 'telnet'
```

Description:

Searches for the id of the process 'telnet'

```
ps r

# List only running processes on Linux


ps T

# List all processes on this current terminal


ps -f

# List processes along with the parent process ID associated with the current Terminal
```

```
ps -x

# View all processes owned by you


ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem

# Display the processes using highest memory


sudo yum list --installed | more

# Lists installed packages on CentOS
```

Command:

```
sudo rpm -qa
```

```
sudo rpm -qa | more
```

Description:

Get a list of all installed packages with rpm command

Command:

```
sudo rpm -q nginx
```

Description:

Check whether nginx package installed or not

Command:

```
sudo rpm -q bash
```

Description:

Check whether bash package installed or not

Command:

```
sudo yum history
```

Description:

List all installed packages with yum on CentOS history command

Command:

```
sudo yum history info 2
```

Description:

Examine history entries in detail using transaction ID [2]

Command:

```
file /etc/passwd
```

Description:

Displays the file type of a given file

```
[root@localhost manju]# file /etc/passwd

/etc/passwd: ASCII text
```

Command:

wc /etc/passwd

Output:

46   91 2373 /etc/passwd

The /etc/passwd file has 46 lines, 91 words and 2373 letters present in it

---

Command:

grep root: /etc/passwd

Description:

Display all lines from /etc/passwd containing the string "**root**"

Command:

grep -n root /etc/passwd

Description:

Display all lines from /etc/passwd containing the string "**root**" with line numbers

---

Command:

grep -c false /etc/passwd

Description:

Display the number of accounts that have /bin/false as their shell

Command:

grep ^root: /etc/passwd

Description:

Display all lines from /etc/passwd starting with the string "**root**" followed by colon

Command:

last | head

Description:

Displays information about the users who logged in and out of the system

(Display the top 10 lines only)

```
lastb

# Display the last unsuccessful login attempts


du /etc/passwd

# Display the disk usage of a /etc/passwd file


killall proc

# Kill all the process named proc


wget https://repo.mysql.com/mysql80-community-release-el8-1.noarch.rpm

# Download the RPM file to install

sudo yum localinstall mysql80-community-release-el8-1.noarch.rpm

# Install the RPM file
```

```
sudo yum localinstall https://repo.mysql.com/mysql80-community-release-el7-1.noarch.rpm

# Install the RPM package via URL


curl --version

# Display curl Version


curl -O http://website.com/myfiles.tar.gz

# Download the file (myfiles.tar.gz) from url "http://website.com/myfiles.tar.gz"

# Saved as myfiles.tar.gz


curl -o files.tar.gz http://website.com/myfiles.tar.gz

# Download the file (myfiles.tar.gz) from url "http://website.com/myfiles.tar.gz"

# Saved as files.tar.gz
```

```
echo 'https://repo.mysql.com/mysql80-community-release-el8-1.noarch.rpm' > urls.txt


xargs -n 1 curl -O < urls.txt

# Download files from a list of URLs in "urls.txt" file


exit 110

# Exit from the terminal window


sudo -l

# know which commands are permitted and not permitted on the current host
```

243

echo -e "\thello\nworld"

```
          hello
world
```

```
history | grep cd | head -12
```

Searches history of first 12 commands which have cd word match

Disadvantages of Open Source Operating System:

- Difficulty to use
- Compatibility Issues

Command:

```
rpm -qa | grep ftp
```

Description:

Check all installed packages of ftp

Command:

```
find /home -mtime +120
```

Description:

Find files in the /home directory which were modified more than 120 days ago

> **Samba enables Linux / UNIX machines to communicate with Windows machines in a network.**

- The **/etc directory** contains configuration files in Linux.
- **The Network File System (NFS)** is a mechanism for storing files on a network.
- **"init"** is the first process in linux which is started by the kernel and its process id is 1.

```
egrep "Hello|Einstein" file.txt
```

Returns line with Hello or Einstein in the file.txt

```
date "+%s"
```

Prints the date in seconds

```
cat file.txt | uniq
```

Display duplicate record only once

Command:

cd ../../..

Takes you three folders back

Command:

```
ps -ef | grep xlogo
```

Description:

List all the processes on the system containing the string **'xlogo'**

```
echo -n "abc";echo "def"
```

abcdef

```
echo "abc";echo "def"
```

abc

def

```
ls -ltr /etc
```

List the files in /etc in order of last modification

246

Command:

```
ls -Rlh /var | grep [0-9]M
```

Description:

List the files in /var larger than 1 megabyte but less than 1 gigabyte

Command:

```
ls -lhS
```

Description:

List files by size

```
cat /etc/passwd /etc/group

# Display the contents of multiple files (/etc/passwd and /etc/group)


find /tmp -name *.txt -exec rm -f {} \;

# Searches for all files in the /tmp directory named *.txt and deletes them
```

```
echo "use" "of" "Linux"
```

use of Linux

```
watch -n 5 tail -n 3 /etc/passwd

# Display the end of the /etc/passwd file every 5 seconds


watch -n 1 'ls -l | wc -l'

# Monitor the number of files in a folder


watch -t -n 1 date

# Display the clock


find / -name "*.txt"

# Search all files with .txt extension


find . -name "*file*"

# Search all files containing "file" in the name


find /home -name "*file*"

# Search all files in /home containing "file" in the name


grep -nre "hello computer" ./*

# Search for files containing the string "hello computer" in the current directory


(echo In Linux; exit 0) && echo OK || echo exit
```

> In Linux
> OK

```
(echo In Linux; exit 4) && echo OK || echo exit
```

> In Linux
> exit

Command:

free -t -m

Display free memory size in MB

Command:

gnome-system-monitor

Description:

Displays what programs are running and how much processor time, memory and disk space are being used

lsblk -m

Display device permissions and ownership

lsblk -S

Display **SCSI** (Small Computer Systems Interface) devices

lsblk -n → List devices without the header

Command:

```
ls -l ~
```

Description:

Check the file and folder permissions

Command:

```
ls ./Documents
```

Description:

Display the list of files that reside in the Documents folder

```
ls -R

# List out all the contents of subdirectories


compgen -c

# Displays the list of all commands which we can use in the command-line interface
```

- `pwd -L` → Prints a symbolic path

- `pwd -P` → Prints the actual full path

```
hostnamectl

# Display system information including operating system, kernel and release version
```

Command:

find . -type f

Find files

Command:

find . -type d

Description:

Find directories

find . -iname "*.jpg"

Find files by case-insensitive extension (ex: .jpg, .JPG, .jpG)

find . -type f -perm 777

Find files by octal permission

cd logs; ls -lt | head; du -sh ; df -h

Concatenating all of the above tasks in a single line using the "; " operator

251

```
{ echo "Albert Einstein"; pwd; uptime; date; }
```

```
Albert Einstein

/home/manju

 00:26:53 up 28 min,  2 users,  load average: 0.00, 0.01, 0.05

Tue Mar 29 00:26:53 PDT 2022
```

```
cal; { date; uptime; }; pwd


   March 2022

   Su Mo Tu We Th Fr Sa

          1  2  3  4  5

    6  7  8  9 10 11 12

   13 14 15 16 17 18 19

   20 21 22 23 24 25 26

   27 28 29 30 31

   Tue Mar 29 00:52:38 PDT 2022

    00:52:38 up 54 min, 2 users, load average: 0.00, 0.01, 0.05

   /home/manju
```

```
shutdown -r

# Kicks off a reboot

shutdown +0

# Shuts down the system immediately

shutdown -r +5

# Begins a reboot of the system in five minutes
```

Command:

kill 12838

Terminate the process with process ID 12838

Command:

ss -t -r state established

Description:

List all the established ports

shutdown -Fr now

Force the file system check during reboot

ss -t -r state listening

List all sockets in listening state

mtr google.com

Diagnose Network Issues

sudo tcpdump --list-interfaces

List all network interfaces

```
ls -al --time-style=+%D | grep `date +%D`
```
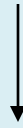
List today's files only

```
mpstat -P 0

# Print processor statistics and helps to monitor CPU utilization on the system


chmod 777 myfiles.txt

# Assign (read, write and execute) permission to everyone


chmod 766 myfiles.txt

# Assign full permission to the owner and read and write permission to group and others


chmod -x myfiles.txt

# Remove the execution permission of myfiles.txt file


history 30

# List the last 30 commands we have entered on the system


find ~ -empty

# Find all empty files in home directory


gzip -l *.gz

# Display compression ratio of the compressed file
```

Command:

ps -efH | more

View current running processes in a tree structure

Command:

df -T

Description:

Display what type of file system

mkdir ~/temp

Creates a directory called temp under home directory

```
ls *py

# List all Python files


chsh -l

# Display the list of all shells


ipcs -a

# Display details about message queue, semaphore and shared memory
```

```
ipcs -q
```
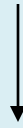
Lists only message queues for which the current process has read access

```
ipcs -s

# List the accessible semaphores


ipcs -m

# List all the Shared Memory


quotastats

# Display the report of quota system statistics gathered from the kernel


rpcinfo

# Display all of the RPC (Remote Procedure Call) services of the local host


slabtop

# Display kernel slab cache information in real-time


tload

# Display a graph of the current system load average to the specified tty
```

```
cat /proc/devices

# Display the device drivers configured for the currently running kernel


cat /proc/dma

# Display the DMA channels currently used


cat /proc/filesystems

# Display the file systems configured into the kernel


cat /proc/kmsg

# Display the messages generated by the kernel


cat /proc/loadavg

# Display the system load average


ls /proc/net

# List the network protocols
```

```
ls /etc/udev

# List the contents of udev configuration directory
```

```
cat /proc/stat

# Display the system operating statistics


cat /proc/uptime

# Display the time the system has been up
```

Command:

```
poweroff -i -f
```

Shutdown the system

```
[2 = 2 ] ; echo $?
# 0 (logically TRUE)


[ 2 = 6 ] ; echo $?
# 1 (logically FALSE)


type echo
# echo is a shell builtin
```

```
find /usr -print
# Find and print all files under "/usr"


systemctl list-units --type=target
# List all target unit configuration


systemctl list-units --type=service
# List all service unit configuration


systemctl list-sockets
# List all socket units in memory
```

```
systemctl list-timers
# List all timer units in memory


systemctl list-dependencies --all
# List dependency of all unit services


systemctl poweroff
# Shut down the system
```

```
systemctl reboot

# Shut down and reboot the system


systemctl suspend
```

```
netstat -ln --tcp

# Find listening TCP ports (numeric)
```

```
# Suspend the system


systemctl hibernate

# Hibernate the system


loginctl user-status

# Display terse runtime status information of the user of the caller's session


loginctl session-status

# Display terse runtime status information of the caller's session


ip route show

# Display all the routing table in numerical addresses


ip neigh

# Display the current content of the ARP (Address Resolution Protocol) cache tables


netstat -l --inet

# Find listening ports
```

Command:

atq

Lists the user's pending jobs

lsof | grep deleted

Print all deleted files which are claiming disk space

echo $$

Display the Process ID of the current process

echo $!

Display the Process ID of most recently started background job

```
date --date="yesterday"

# Display yesterday 's date


date --date="10 days ago"

# Display date 10 days ago


ls / | wc -w

# List the number of directories in the root directory


sudo sfdisk -l -uM

# Display the size of each partition in MB
```

```
sudo parted -l

# Lists out the partition details


df -h | grep ^/dev

# Filter out real hard disk partitions/file systems
```

```
sudo blkid

# Displays information about available block devices


ls / > info.txt

cat info.txt

        bin
        boot
        dev
        etc
        home
        lib
        lib64
        media
        mnt
        opt
        proc
        root
        run
        sbin
        srv
        sys
        tmp
        usr
        var
```

```
export NAME="Albert Einstein"

echo $NAME
```

```
Albert Einstein
```

```
TZ=US/Pacific date

# Display the current date/time in US/Pacific time zone
```

```
ls -l /etc/shadow

# Display the user password stored in an encrypted form and the password expiry data
```

```
sudo journalctl --since yesterday

# Display all the logs since yesterday
```

```
sudo journalctl --since "2019-12-10 13:00:00"

# Display all the logs since 2019-12-10 13:00:00
```

```
journalctl -disk-usage

# Display the total size of the journal logs
```

Command:

```
ls -m
```

Prints out directories and files separated by a comma

ls -Q

Add quotation marks to all directories and files

ss -f unix

List Unix Sockets

```
echo *.desktop

# Lists all of the .desktop files in the current directory
```

ss --raw

List Raw Sockets

```
tracepath www.google.com

# Traces a path to a network host (www.google.com) discovering MTU along the path


echo -e "123\b4"        124          3 is over-written by 4



echo -e "123\r456"     456          123 is overwritten by 456



echo D*

# Lists all of the files and directories in the current directory whose name starts with letter D
```
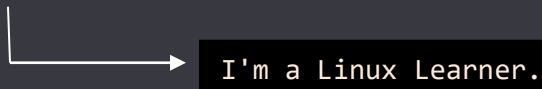
```
echo $'I\'m a Linux Learner.'
```

```
I'm a Linux Learner.
```

```
echo $USER
```

```
# Print the name of the currently logged in user
```

```
echo -e "\033[0;32mGREEN"
```

GREEN

```
echo -e "\033[0;31mRED"
```

RED

```
echo -e 'Hello, \vWorld!'
```

Hello,

World!

```
echo "This is the list of directories and files on this system: $(ls)"
```

This is the list of directories and files on this system: Desktop

Documents

Downloads

Music

Pictures

Public

Templates

Videos

```
echo *s
# Print all files and folders that end by letter "s"


echo [[:upper:]]*
# Print all files and folders that start by upper case character


echo $((2 + 3))
→ 5


echo $(($((2**2)) * 3))
→ 12


echo Four divided by two equals $((4/2))
→ Four divided by two equals 2


echo Capital-{A,B,C}-Letter
→ Capital-A-Letter Capital-B-Letter Capital-C-Letter


echo {1..5}
→ 1 2 3 4 5


echo {A..Z}
→ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z


echo x{P{1,2},Q{3,4}}y
→ xP1y xP2y xQ3y xQ4y
```

```
echo The total price is $500.00
```

→ The total price is 00.00

```
echo "$USER $((3*2)) $(cal)"
```

manju 6   March 2022

Su Mo Tu We Th Fr Sa

       1 2 3 4 5

 6 7 8 9 10 11 12

13 14 15 16 17 18 19

20 21 22 23 24 25 26

27 28 29 30 31

```
echo -e "\aMy Laptop shut \"down\"."
```

→My Laptop shut "down".

```
echo -e "C:\\WIK2N\\LINUX_OS.EXE"
```

→ C:\WIK2N\LINUX_OS.EXE

```
echo $(cal)
```

March 2022 Su Mo Tu We Th Fr Sa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

```
echo "$(cal)"
```

March 2022

Su Mo Tu We Th Fr Sa

       1 2 3 4 5

 6 7 8 9 10 11 12

13 14 15 16 17 18 19

20 21 22 23 24 25 26

27 28 29 30 31

```
echo The total price is \$500.00
```

→The total price is $500.00

```
sudo lsof -i -P -n | grep LISTEN ────────

sudo netstat -tulpn | grep LISTEN ──────


sudo ss -tulw

# Check what ports are open


netstat -ap | grep ssh

# Find out on which port a program is running


[root@localhost manju]# ipcs -m -l

------ Shared Memory Limits --------

max number of segments = 4096

max seg size (kbytes) = 18014398509465599

max total shared memory (kbytes) = 18014398442373116

min seg size (bytes) = 1


[root@localhost manju]# ipcs -m -p

------ Shared Memory Creator/Last-op PIDs --------

shmid       owner       cpid        lpid

131072      manju       2998        3135

163841      manju       2998        3135

327682      manju       3277        6920

360451      manju       2827        1406
```

Check ports in use

Lists the Limits for Inter-process
Communication facility

Display the process ids that accessed
Inter-process Communication facility
recently

```
[root@localhost manju]# ipcs -u

------ Messages Status --------

allocated queues = 0

used headers = 0

used space = 0 bytes

------ Shared Memory Status --------

segments allocated 4

pages allocated 2432

pages resident  319

pages swapped   0

Swap performance: 0 attempts     0 successes

------ Semaphore Status --------

used arrays = 0

allocated semaphores = 0
```

Display the status of current usage of **Inter-process Communication** facility

```
dmidecode -t 16

# Display the maximum RAM supported by the system
```

```
dmidecode -t baseboard

# Display all the system baseboard related information


dmidecode -t bios

# Display the BIOS information
```

Command:

```
dmidecode -t system
```

Description:

Display the information about the manufacturer, model and serial number of the system
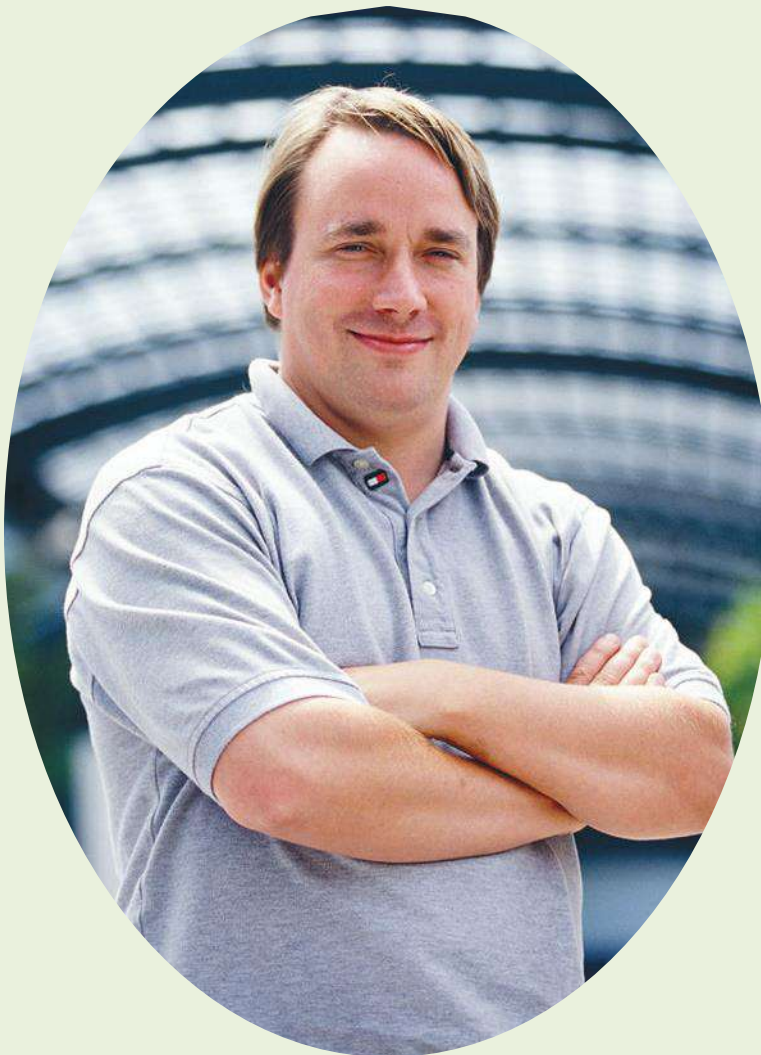
The Linux philosophy is **'Laugh in the face of danger'**. Oops. Wrong One. 'Do it yourself'. Yes, that's it.

Linus Torvalds

```
nmcli con show -a
# Display the active network connections


netstat -r
# Display the kernel routing table


yum install nmap
# Install nmap on CentOS


nmap google.com
# Scan a hostname


nmap 193.169.1.1
# Scan a ip address


nmap --iflist
# Display host interfaces and routes


echo [![:digit:]]*
# Print all files and folders that are not beginning with a numeral


echo *[[:lower:]123]
# Print all files and folders ending with a lowercase letter or the numeral


echo g*
# Print all files and folders beginning with "g"
```

```
echo b*.txt
```

Print all files and folders beginning with "b" followed by any characters and ending with ".txt"

```
echo [abc]*
```

Print all files and folders beginning with either "a", "b" or "c"

```
netstat -t
```

Display the download status of active connections

```
netstat -x
```

Display Information about all connections, listeners and shared endpoints for Network Direct

```
netstat -n
```

Numerical display of addresses and port numbers

```
echo $LANG
```

Display the language of a Linux system

```
echo "AAA" | grep AAA
```

→ AAA

```
echo "BBB" | grep -E 'AAA|BBB'
```

→ BBB

```
echo "AAA" | grep BBB
```

→

```
echo "AAA" | grep -E 'AAA|BBB'
```

→ AAA
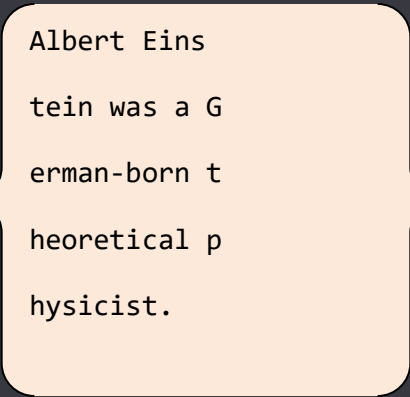
```
echo "albert einstein" | tr a-z A-Z
```

→ ALBERT EINSTEIN

```
echo "albert einstein" | tr [:lower:] E
```
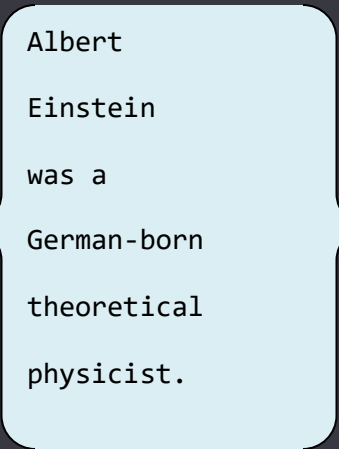
→ EEEEEE EEEEEEEE

```
echo "Albert Einstein was a German-born theoretical physicist." | fold -w 12
```

```
Albert Eins

tein was a G

erman-born t

heoretical p

hysicist.
```

```
echo " Albert Einstein was a German-born theoretical physicist." | fold -w 12 -s
```

```
Albert

Einstein

was a

German-born

theoretical

physicist.
```
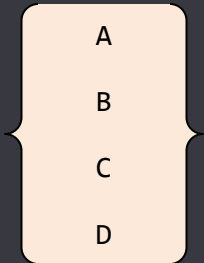
```
printf "English theoretical physicist: %s\n" Hawking
```

→ English theoretical physicist: Hawking

```
ls /usr/bin | pr -3 -w 65 | head
```
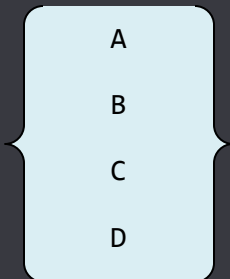
Display a directory listing of /usr/bin in a paginated, three-column output format

```
for i in A B C D; do echo $i; done
```

```
A

B

C

D
```

```
Scientists=("Einstein" "Hawking" "Darwin"); for i
    in ${Scientists[*]}; do echo $i; done
```

```
Einstein

Hawking

Darwin
```
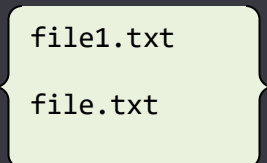
```
for i in {A..D}; do echo $i; done
```

```
A

B

C

D
```

```
Scientists=("Einstein" "Hawking" "Darwin"); for i
    in "${Scientists[*]}"; do echo $i; done
```

```
Einstein Hawking Darwin
```

```
for i in file*.txt; do echo $i; done
```
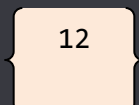
```
file1.txt

file.txt
```

```
echo ${!BASH*}
```
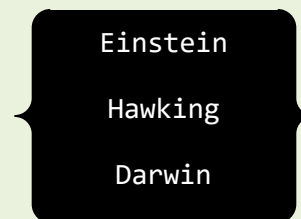
List all the variables in the environment with names that begin with BASH

```
bc <<< "6+6"
```
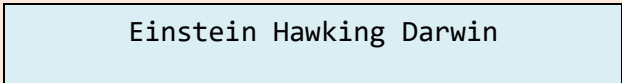
```
12
```

```
df -k

# Check the file system space


ls -alh

# List all folders in directory with details


find /home -name file.txt

# Check all files in /home directory with the name file.txt


find /home -iname File.txt

# Search all files in /home directory irrespective to case sensitive


 find / -ctime +90

# Search for the files which were modified more than 90 days back


find / -size 0c

# Search all empty files


find / -size +1G

# Search all files and folders which are more than 1GB


df -a

# Display the file system's complete disk usage


df -i

# Display used and free inodes
```

df -h

Display disk space in human-readable format

```
du -ch *.png
```

Display the size of each png file in the current directory

```
du -a /etc/ | sort -n -r | head -n 10
```

List top 10 directories consuming disk space in /etc/

```
ac
```

Display the total amount of time users are connected to the system

```
ac --individual-totals
```

Display a report on login times for individual users

```
cancel
```

Cancels print jobs

```
yum install finger
```

Install finger tool (CentOS)

```
finger manju
```

Display the details of a user "manju"

```
chfn
```

Allows you to modify user's information

```
finger -s manju
```

Display idle status and login details of a user "manju"

```
groups
# List all Groups the Current User is a Member of


id -nG
# List all Groups the Current User is a Member of


groupadd mygroup
# Create a new group named "mygroup"


groupdel mygroup
# Delete a group named "mygroup"


less /etc/group
# List all Groups


getent group
# List all Groups


usermod -a -G mygroup manju
# Add an existing user "manju" to a group "mygroup"


userdel manju
# Delete a user "manju"


chgrp mygroup test.txt
# Change the owning group of the file test.txt to the group named "mygroup"
```

```
sudo depmod -a

# Generates a list of all kernel module dependences and associated map files


dirname /usr/bin

→ /usr


dirname /Desktop/root

→ /Desktop


dmesg > kernel_messages.txt

# Read all messages from kernel ring buffer to a file "kernel_messages.txt"


dmesg | grep -i memory

# Display the kernel messages which relate to memory usage


egrep -c '^Hello|World$' myfiles.txt

# Count the number of lines in myfiles.txt which begin with the word 'Hello' or end with the word 'World'


ex myfiles.txt

# Edits the file myfiles.txt


expand myfiles.txt

# Expand the file myfiles.txt – changing tabs to spaces – and display on standard output


expand --tabs=10 myfiles.txt > myfiles0.txt

# Convert the tabs in the file myfiles.txt to 10 spaces each, and write the output to myfiles0.txt
```
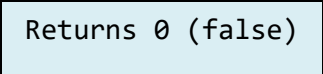
```
expr 2 = 5
# 0
```

Returns 0 (false)

```
fc -l
# Lists the history of commands


!l
# Executes the most recently executed command that begins with the letter "l"


fc -e - l
# Executes the most recently executed command that begins with the letter "l"


fmt myfiles.txt
# Display a reformatted version of the file "myfiles.txt "


fmt < myfiles.txt > myfiles0.txt
# Reformat "myfiles.txt" and write the output to the file "myfiles0.txt"


finger -p manju
# Display information about the user "manju"


fold -w5 myfiles.txt > myfiles0.txt
# Wrap the lines of myfiles.txt to a width of 5 characters and writes the output to myfiles0.txt


for file in *.txt ; do wc -l $file ; done
# Performs a word count of all files in the current directory with the .txt extension
```

```
grep manju /etc/passwd

# Search /etc/passwd for user "manju"


groupmod -n group mygroup

# Change the group "mygroup" to "group"


head myfiles.txt

# Display the first 10 lines of "myfiles.txt"


head -15 myfiles.txt

# Display the first 15 lines of "myfiles.txt"


head myfiles.txt myfiles0.txt

# Display the first 10 lines of both myfiles.txt and myfiles0.txt - with a header before each that indicates the file name


head -n 5K myfiles.txt

# Display the first 5,000 lines of "myfiles.txt"


head -n 4 *.txt

# Display the first 4 lines of every file in the working directory whose file name ends with the .txt extension


iostat

# Display operating system storage input and output statistics


last reboot | less

# Display listing of last logged in users and system last reboot time and date
```

```
last -x | less
```

Display last shutdown date and time


```
last shutdown
```

Display last shutdown date and time


```
ldd /bin/bash
```

Display the shared library dependencies of the program /bin/bash


```
less -N myfiles.txt
```

View the file myfiles.txt – displaying a line number at the beginning of each line


```
ls *.{html,php,txt}
```

List all files with .html, .php and .txt file extension

```
lsof -i 6

List only open IPv6
network files
```

```
ls /
```

List the contents of root directory


```
ls [aeiou]*
```

List only files that begin with a vowel (a, e, i, o and u)


```
lsof -i -U
```

List all open Internet, x.25 (HP-UX) and UNIX domain files


```
lsof -i 4 -a -p 555
```

List all open IPv4 network files in use by the process whose Process ID is 555

279

```
xz myfiles.txt

# Compress the file "myfiles.txt" into "myfiles.txt.xz"


xz -dk myfiles.txt.xz

# Decompress "myfiles.txt.xz" into "myfiles.txt"


mkdir -m a=rwx dir

# Create the directory "dir" and set its file mode so that all users may read, write and execute it


modinfo snd

# Display all available information about the "snd" Linux kernel module


more +3 myfiles.txt

# Display the contents of file "myfiles.txt" beginning at line 3


more +/"Hello" myfiles.txt

# Display the contents of file "myfiles.txt" beginning at the first line containing the string "Hello"


netstat -g

# Display multicast group membership information for both IPv4 and IPv6


netstat -c

# Print netstat information every few second


netstat -natp

# Display statistics about active Internet connections
```

```
netstat -rn

# Display the routing table for all IP addresses bound to the server


netstat -an

# Display information about all active connections to the server


od -b myfiles.txt

# Display the contents of "myfiles.txt" in octal format


od -Ax -c myfiles.txt

# Display the contents of "myfiles.txt" in ASCII character format – with byte offsets displayed as hexadecimal


trap -l

# Display a list of signal names and their corresponding numbers


trap

# Display a list of the currently-set signal traps


yum list openssh

# Search for a package with a name "OpenSSH"


yum grouplist

# List all available Group Packages


yum repolist

# List all enabled Yum repositories
```

```
yum repolist all

# List all Enabled and Disabled Yum Repositories


paste 1.txt 2.txt

# Display the contents of 1.txt and 2.txt side-by-side


ls -a | pr -n -h "Files in $(pwd)" > dc.txt

cat dc.txt
```

2022-04-02 01:10	Files in /home/manju	Page 1
1	.	
2	..	
3	1.txt	
4	2.txt	
5	.bash_history	
6	.bash_logout	
7	.bash_profile	
8	.bashrc	
9	bio.txt	
10	.cache	
11	.config	
12	Data.txt	
13	Desktop	
14	dir	
15	Documents	

```
printf "Hi, I'm %s.\n" $LOGNAME

→ Hi, I'm manju.


printf "%.*s" 6 "abcdefg"

→ abcdef


ps -eLf

# Get information about threads


ps axms

# Get information about threads
```

- ps -eo euser,ruser,suser,fuser,f,comm,label
- ps axZ
- ps -eM

Get security information

Command:

```
                            pstree -h
```

Description:

```
    Display all processes as a tree, with the current process and its ancestors highlighted
```

```
rm -- 1.txt

# Delete "1.txt" file in the current directory


rm /home/manju/2.txt

# Delete "2.txt" file in the directory "/home/manju"
```

rm ./1.txt

```
ip route list

# List current routing table


route -n

# Display routing table for all IPs bound to the server


script -c 'echo "Hello, World!"' hello.txt
```

> Script started, file is hello.txt
>
> Hello, World!
>
> Script done, file is hello.txt

```
  cat hello.txt
```

> Script started on Sat 02 Apr 2022 03:24:52 AM PDT
>
> Hello, World!
>
> Script done on Sat 02 Apr 2022 03:24:52 AM PDT

```
sfdisk -s

# List the sizes of all disks


ls -d ~/.ssh

# Check if the .ssh directory exists or not


sha224sum myfiles.txt

# Display the SHA224 checksum of the "myfiles.txt" file in the current directory
```

```
sha256sum myfiles.txt

# Display the SHA256 checksum of the "myfiles.txt" file in the current directory


sha384sum myfiles.txt

# Display the SHA384 checksum of the "myfiles.txt" file in the current directory


sha512sum myfiles.txt

# Display the SHA512 checksum of the "myfiles.txt" file in the current directory


shutdown 8:00

# Schedule the system to shut down at 8 A.M


shutdown 20:00

# Schedule the system to shut down at 8 P.M


shutdown +15 "The system will be shutdown in 15 minutes."

# Schedule the system to shut down in 15 minutes with the normal message alerting users that the system is shutting down


shutdown -P now

# Power off the system immediately

sleep 10

# Delay for 10 seconds

startx -- -depth 16

# Start an X session at 16 bits color depth
```

```
time cal

# Reports how long it took for the "cal" command to complete


tr "[:lower:]" "[:upper:]" < myfiles.txt

# Translate the contents of "myfiles.txt" to uppercase


tr -cd "[:print:]" < myfiles.txt

# Remove all non-printable characters from "myfiles.txt"


cat myfiles.txt
```

```
    Hello World
```

```
xlsfonts

# Lists all fonts available to the default X server and display
```

```
tr -cs "[:alpha:]" "\n" < myfiles.txt
```

```
Hello

World
```

xset q

Display the values of all current X Window

System preferences

```
uncompress myfiles.txt.xz

# Uncompress the file "myfiles.txt.xz"
```

```
yes | rm -i *.txt
```

Remove all files with the extension .txt

from the current directory

```
w manju

# Display information for the user named "manju"
```

```
write albert

# Write a message to the user "albert"
```

What is Linux and why is it so popular?

Whether you know it or not you are already using Linux (the best-known and most-used open source operating system) every day. From supercomputers to smartphones, the Linux operating system is everywhere. As an operating system, Linux is a family of open source Unix-like software based on the Linux kernel - that sits underneath all of the other software on a computer, receiving requests from those programs and relaying these requests to the computer's hardware. With regard to careers, it is becoming increasingly valuable to have Linux skills rather than just knowing how to use Windows. In general, Linux is harder to manage than Windows, but offers more flexibility and configuration options.

Every desktop computer uses an operating system. The most popular operating systems in use today are: Windows, Mac OS, and LINUX. Linux is the best-known notoriously reliable and highly secure open source portable operating system -- very much like UNIX -- that has become very popular over the last several years -- created as a task done for pleasure by Linus Torvalds -- computer science student at the University of Helsinki in Finland -- in the early 1990s and later developed by more than a thousand people around the world.

Linux is fast, free and easy to use, that sits underneath all the other software on a computer − runs your computer -- handling all interactions between you and the hardware i.e., whether you're typing a letter, calculating a money budget, or managing your food recipes on your computer, the Linux operating system (similar to other Operating Systems, such as Windows XP, Windows 7, Windows 8, and Mac OS X) provides the essential air that your computer breathes.

Linux is the most important technology advancement of the twenty-first century and Licensed under the General Public License (GPL) that Linux uses ensures that the software will always be open to anyone and whose source code is open and available for any user to check, which makes it easier to find and repair vulnerabilities and it power the laptops, development machines and

servers at Google, Facebook, Twitter, NASA, and New York Stock Exchange, just to name a few. Linux has many more features to amaze its users such as: Live CD/USB, Graphical user interface (X Window System) etc.

Why LINUX?

Although Microsoft Windows (which is the most likely the victim of viruses and malware) has made great improvements in reliability in recent years, it considered less reliable than Linux. Linux is notoriously reliable and secure and it is free from constant battling viruses and malware (which may affect your desktops, laptops, and servers by corrupting files, causing slow downs, crashes, costly repairs and taking over basic functions of your operating system) – and it keep yourself free from licensing fees i.e., zero cost of entry ... as in free. You can install Linux on as many reliable computer ecosystems on the planet as you like without paying a cent for software or server licensing. While Microsoft Windows usually costs between $99.00 and $199.00 USD for each licensed copy and fear of losing data.

Below are some examples of where Linux is being used today:

- Android phones and tablets
- Servers
- TV, Cameras, DVD players, etc.
- Amazon
- Google
- U.S. Postal service
- New York Stock Exchange

Linux Operating System has primarily three components:

- **Kernel**

Kernel is the core part of Linux Operating System and interacts directly with hardware. It is responsible for all major activities of the Linux operating system.

- **System Library**

System libraries are special programs using which application programs accesses Kernel's features.

- **System Utility**

 System Utility programs are responsible to do specialized tasks.

Important features of Linux Operating System:

- Portable
- Open Source
- Multi-User
- Multiprogramming
- Hierarchical File System
- Security
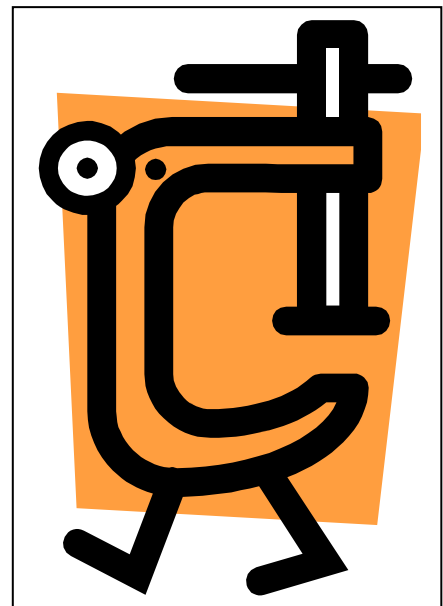
Now Linux (successfully being used by several millions of users worldwide) has grown passed the stage where it was almost exclusively an academic system, useful only to a handful of people with a technical background. It provides more than the operating system: there is an entire infrastructure supporting the chain of effort of creating an operating system, of making and testing programs for it, of bringing everything to the users, of supplying maintenance, updates and support and customizations, runs on different platforms including the Intel and Alpha platform. Today, Linux is ready to accept the challenge of a fast-changing world to do various

types of operations, call application programs etc. Since the hiring focus is shifting more and more toward DevOps type skills, a Linux skill set will be the types of things that will make you very deployable.

```
[manju@localhost ~]$ echo al{an,bert,exander}

alan albert alexander

[manju@localhost ~]$ mkdir {txt,doc,pdf}files

[manju@localhost ~]$ ls

txtfiles docfiles pdffiles
```

```
[manju@localhost ~]$ x=Albert; y="$x won \$100.00"; echo $y

Albert won $100.00

[manju@localhost ~]$ x=5; test $x -eq 10; echo $?

1       exit status of the test command is 1

[manju@localhost ~]$ x=5; test $x -eq 5; echo $?

0       exit status of the test command is 0
```

```
cat  /etc/profile

# Display System login initialization file

cat  /etc/bashrc

# Display System BASH shell configuration file

cat  .bash_profile

# Display Login initialization file
```

```
[manju@localhost ~]$ date

Wed Sep 28 08:14:17 PDT 2022

[manju@localhost ~]$ alias x=date

[manju@localhost ~]$ x

Wed Sep 28 08:14:27 PDT 2022
```

```
[manju@localhost ~]$ echo $BASH_VERSION

4.2.46(1)-release          Display the current BASH version number

[manju@localhost ~]$ echo $HISTCMD

290              Display the number of the current command in the history list

[manju@localhost ~]$ echo $HOSTTYPE

x86_64                 Display the type of machine the host runs on

[manju@localhost ~]$ echo $OSTYPE

linux-gnu            Display the operating system in use

[manju@localhost ~]$ echo $PPID

3563            Display the process ID for shell's parent shell

[manju@localhost ~]$ echo $SHLVL

2                Display the current shell level

[manju@localhost ~]$ echo $TERM

xterm-256color      Display the terminal type

[manju@localhost ~]$ echo $EUID

1000             Display the Effective user ID
```

```
[manju@localhost ~]$ PS1="\d"  ◄──────────────────────┐
                                                        │
Sun Oct 02 # Display the Current date  ─────────────────┘

[manju@localhost ~]$ PS1="\h"◄─────────────────────┐
                                                    │
localhost # Display the Hostname  ──────────────────┘

[manju@localhost ~]$ PS1="\s"  ◄────────────────────────┐
                                                         │
bash  # Display the Shell type currently active  ────────┘

[manju@localhost ~]$ PS1="\t"  ◄───────────────────────────────┐
                                                                │
18:42:10  # Display the Time of day in hours, minutes, and seconds ─┘

[manju@localhost ~]$ PS1="\u"◄────────────┐
                                           │
manju # Display the Username  ─────────────┘

[manju@localhost ~]$ PS1="\v"  ◄──────────┐
                                           │
4.2 # Display the Shell version  ──────────┘
```

PS1="\w"	Display the full pathname of the current working directory
PS1="\W"	Display the name of the current working directory

```
[manju@localhost ~]$ PS1="Hello\n World"

Hello

 World

[manju@localhost ~]$ PS1="Hello \\ World"

Hello \ World
```

Directory	Function
/	The top-level directory of a Linux system that holds all files, device information, and system information organized into directories
/home	Holds users' home directories
/bin	Contains every essential command and utility program
/usr	Contains the commands and files that the system uses
/usr/bin	Include utility programs and user-friendly commands
/usr/sbin	Holds commands for system administration
/usr/lib	Contains programming language libraries
/usr/share/doc	Contains documentation for Linux
/usr/share/man	Contains the online **"man"** files
/var/spool	Contains spooled files, such as those produced for network transfers and printing operations
/sbin	Contains commands for system administration used to boot the system
/var	Holds a variety of files, including mailbox files
/dev	Holds file interfaces for devices like printers and terminals
/etc	Holds all system files, including configuration files

```
find myfiles -name '*.c' -ls
```
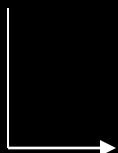
Using the **-ls** command, all files in the **"myfiles"** directory with the **.c extension** are searched and displayed

```
find / -user manju -print
```
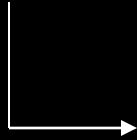
Finds every file in a user's home directory and every
file that user owns in other user directories

```
ls /usr/share/X11

# List the system X11 configuration and support files


ls /etc/X11

# List the configuration files


ls /etc/gdm

# Display the contents of GDM configuration directory


ls /usr/share/gdm

# Display the contents of GDM configuration directory for default settings and themes
```

```
ls /etc/gconf

# List the GConf configuration files


ls /usr/share/gnome

# List the Files used by GNOME applications


ls /usr/share/doc/gnome*

# Display the contents of Documentation for various GNOME packages, including libraries
```

```
ls /usr/share/icons

# List the Icons used in KDE desktop and applications


rpm -qa | more

# Displays a list of all installed packages
```

```
ls /etc/cron.d

# List the directory with numerous crontab files that is only accessible to the root user


ls /etc/cron.hourly

# List the directory for tasks performed hourly


ls /etc/cron.daily

# List the directory for tasks performed daily


ls /etc/cron.weekly

# List the directory for tasks performed weekly


ls /etc/cron.monthly

# List the directory for tasks performed monthly


ls /etc/mtab

# List the currently mounted file systems


ls /etc/services

# List the services run on the system and the ports they use
```

```
ls /etc/cups

# List the CUPS printer configuration files


ls /proc/net

# List the Directory for network devices
```

```
free -s 3

# Display the current usage status of Memory continuously after regular interval


ls -lhR  /var | grep \- | grep [1-9]*M

# List "/var" files larger than 1 MB but less than 1 GB
```

```
whereis -b ls

# Search only the binary file related to a command "ls"


whereis -m ls

# Searches only for man pages related to a command "ls"


whereis -s ls

# Searches only for source files related to a command "ls"


[manju@localhost ~]$ echo "Alan" "Mathison" "Turing"

Alan Mathison Turing
```

```
watch -t -n 1 date

# Display the date



[manju@localhost ~]$ echo "Albert" > 1.txt && cat 1.txt

Albert
```

```
du -sh * --time

# Check each file's size and the date and time it was last edited


dmidecode -s system-serial-number

# Display the serial number of Linux server
```

```
ls -aril

# Display all the files with sequence number


yum search mod_

# Display all the modules
```

```
du -sch *

# Display the sum of size of all files and folders in present directory


dmidecode | grep -A3 '^System Information'

# Display the server hardware name and model


dmesg | grep -i firmware

# Display all firmware error
```

```
cat /proc/cpuinfo | grep processor | wc -l

# Display the number of cores


netstat -ap | grep 80

# Display the process id which is using port number 80
```

```
dmidecode --type memory

# Display the physical memory attached to the Server



dirs

# Display the list of currently remembered directories
```

```
blkid -i /dev/sda
# Display information about available block devices


crontab -e
```

This command runs crontab

```
[manju@localhost ~]$ df -h /home

Filesystem        Size   Used Avail Use% Mounted on

/dev/sda3         18G   5.2G   13G   29%  /
```

Look for free disk space using the **df** command within the **/home** directory

```
[manju@localhost ~]$ hostname -I   ←─────────────────────┐
192.168.6.131 192.168.122.1                              │
                                                         │
    Display all local IP addresses of the host ──────────┘


badblocks -s /dev/sda
# Check for unreadable blocks on disk sda


tail -10 /var/log/messages
# Display the last 10 syslog messages


lsof -u manju
# List files opened by the user "manju"


sudo shutdown -r 2
# Shuts down and reboots the machine in 2 minutes


[manju@localhost ~]$ cat 1.txt
albert
[manju@localhost ~]$ cat 1.txt | tr a-z A-Z > 2.txt
[manju@localhost ~]$ cat 2.txt
ALBERT


cat /etc/passwd | column -t -s :
# Display the contents of "/etc/passwd" in column
```

```
nmcli d

# Display the status of all network interfaces


grep "^[[:alnum:]]" myfiles.txt

# Search for a line which will start with alphanumeric characters in "myfiles.txt"


grep "^[[:alpha:]]" myfiles.txt

# Search for a line which will start with alpha characters in "myfiles.txt"


grep "^[[:blank:]]" myfiles.txt

# Search for a line which will start with blank characters in "myfiles.txt"


grep "^[[:digit:]]" myfiles.txt

# Search for a line which will start with digit characters in "myfiles.txt"
```

```
grep "^[[:lower:]]" myfiles.txt

# Search for a line which will start with lowercase letters in "myfiles.txt"


grep "^[[:punct:]]" myfiles.txt

# Search for a line which will start with punctuation characters in "myfiles.txt"


grep "^[[:graph:]]" myfiles.txt

# Search for a line which will start with graphical characters in "myfiles.txt"


grep "^[[:print:]]" myfiles.txt

# Search for a line which will start with printable characters in "myfiles.txt"
```

```
grep "^[[:space:]]" myfiles.txt

# Search for a line which will start with space characters in "myfiles.txt"


grep "^[[:upper:]]" myfiles.txt

# Search for a line which will start with uppercase letters in "myfiles.txt"


grep "^[[:xdigit:]]" myfiles.txt

# Search for a line which will start with hexadecimal digits in "myfiles.txt"
```

```
vmstat -a

# Display active and inactive system memory


vmstat -s

# Display memory and scheduling statistics
```

```
vmstat -f

# Display number of forks created since system boot


vmstat -D

# Display a quick summary statistic of all disk activity


vmstat -d

# Display a detailed statistic on each disk usage


vmstat 5 -S M  ⟶
```
> This command is used to update the statistics every five seconds and change the display units to megabytes

```
[manju@localhost ~]$ free -h --total
            total        used        free      shared  buff/cache   available
Mem:         976M        566M         75M        8.7M        334M        209M
Swap:        2.0G         84K        2.0G
Total:       3.0G        566M        2.1G
```

- hostname -s
- hostname --short

Display the short version of the hostname

```
hostname --all-ip-addresses

# Display All Network Addresses


date -r /etc/hosts

# Display Last Modified Timestamp of a Date File
```

```
[manju@localhost ~]$ cat 1.txt

Albert Einstein

[manju@localhost ~]$ cat 2.txt

Elsa Einstein

[manju@localhost ~]$ cat 1.txt > 2.txt

[manju@localhost ~]$ cat 2.txt

Albert Einstein
```

```
[manju@localhost ~]$ cat 12.txt

Albert Einstein

Elsa Einstein

[manju@localhost ~]$ cat -n 12.txt

    1      Albert Einstein
    2      Elsa Einstein
```

```
[manju@localhost ~]$ cat 1.txt

Albert Einstein

[manju@localhost ~]$ cat -e 1.txt

Albert Einstein$
```

```
sudo shutdown 08:00
# Shutdown the system at 8 AM in the morning


grep 'but\|is' phy.txt
# Search for the words "but" and "is" in the phy.txt file
```

```
grep 'is\|but\|of' phy.txt
# Search for the words "but", "is" and "of" in the phy.txt file



   grep -e but -e is -e of phy.txt
```

```
echo "The system will be shutdown in 10 minutes." | wall
```

The message (The system will be shutdown in 10 minutes.) will
be broadcasted to all users that are currently logged in

```
[manju@localhost ~]$ echo -e 'Albert    Einstein'

Albert    Einstein

[manju@localhost ~]$ echo -e 'Albert \c Einstein'

Albert [manju@localhost ~]$
```

```
ss --all

# List all listening and non-listening connections


ss --listen

# List only listening sockets
```

```
ss -t state listening

# Find all listening TCP connections


[manju@localhost ~]$ hostname -I | awk '{print $1}'

192.168.6.131
```

System's IP address

```
yum erase httpd

# Uninstall apache
```

- read has the value of 4
- write has the value of 2
- execute has the value of 1
- no permission has the value of 0

```
chmod 644 1.txt
```

- User: 6 = 4 + 2 (read and write)
- Group: 4 = 4 + 0 + 0 (read)
- Others: 4 = 4 + 0 + 0 (read)

- **7 = 4 + 2 + 1** (read, write and execute)
- **6 = 4 + 2 + 0** (read and write)
- **5 = 4 + 0 + 1 (read and execute)**
- **4 = 4 + 0 + 0** (read)

```
rpm -qi httpd

# Display information about a particular package (apache)


sudo rpm -qa | wc -l

# Display the total number of packages installed


sudo repoquery -a --installed

# List all installed packages with the repoquery command
```

```
cat /var/log/boot.log

# Display all information related to booting operations



cat /var/log/maillog

# Display all information related to mail servers and archiving emails



cat /var/log/yum.log

# Display Yum command logs
```

```
mkdir -m777 myfiles

# Create a directory "myfiles" with read, write and execute permissions



rpm -qa centos-release

# Display CentOS version



ps -AlFH

# Get information about threads (LWP and NLWP)


    • ps -eM          Get Security Information of Linux Process

    • ps axZ



ps -auxf | sort -nr -k 4 | head -10

# Display the top 10 memory consuming process
```

```
ps -auxf | sort -nr -k 3 | head -10

# Display the top 10 CPU consuming process


sar -n DEV | more

# Monitor, collect and report Linux system activity
```

```
# create or overwrite "1.txt" file
echo "Albert Einstein" > 1.txt


# create or append to "1.txt" file
echo "Albert Einstein" >> 1.txt
```

```
grep -i "is" phy.txt

# Search for a given string in a file "phy.txt"



grep -A 3 -i "is" phy.txt

# Print the matched line and the following three lines



grep -r "is" *

# Recursively look for a given string in all files



export | grep ORACLE

# Display oracle related environment variables
```

```
chkconfig --list | grep network

# View the startup configuration of Linux network service



shutdown -r 18:30

# Shutdown the system immediately and reboot at 18:30



find /home -size +1024 -print

# Find files above 1MB in home directory



find /home -size +1024 -size -4096 -print

# Find files above 1Mb and below 4MB in home directory
```

```
netstat -ain

# Display the Kernel Interface table



sar -n SOCK | more

# Display networking Statistics



find /home -size +10000k

# Find files greater than 10000k in the home directory



ls -ld /home

# List information about the home directory instead of its contents
```

```
chmod go=+r 1.txt

# Add read permission for the owner and the group


chown manju 1.txt

# Change ownership of a file "1.txt" to user "manju"
```

```
du -sh *

# Display the disk usages of the files in the current directory


du -sh .[!.]* *

# Display the disk usages of the files (including hidden files) in the current directory
```

```
du -sch .[!.]* *
```

Display the total disk usage of the files
(including hidden files) in the current directory

```
du --threshold=1G -sh .[!.]* *
```

Display only files with more than 1GB in size which
located under current directory

```
iostat -kx

# Display general information about the disk operations in real time


netstat -ntlp

# Display open TCP sockets
```

Parted is a well-known command line tool that allows us to easily manage hard disk partitions

```
netstat -nulp

# Display open UDP sockets


netstat -nxlp

# Display open Unix sockets
```

```
sudo yum install parted

# Install parted


parted -v

# Check Parted version


parted -l

# Lists partition layout on all block devices


parted -m

# Displays machine parseable output
```

```
dmidecode -q | less

# Display BIOS information


systemctl --failed

# List failed services


losetup

# Display information about all loop devices
```

```
quit

# Exit the parted shell
```

- getfacl --access 1.txt
- getfacl -a 1.txt

Display the file access control list of a file "1.txt"

- getfacl -n 1.txt
- getfacl --numeric 1.txt

List the numeric user and group IDs w.r.t file "1.txt"

```
sudo tcpdump -D

# List of all available network interfaces in the system
```

```
[manju@localhost ~]$ xz myfiles.txt

[manju@localhost ~]$ ls | grep myfiles

myfiles

myfiles.txt.xz
```

Compress a file **"myfiles.txt"** using xz command

```
[manju@localhost ~]$ free -t | awk 'NR == 2 {print $3/$2*100}'

61.852

[manju@localhost ~]$ free -t | awk 'FNR == 2 {print $3/$2*100}'

61.852
```

Display Memory Utilization

```
[manju@localhost ~]$ free -t | awk 'NR == 3 {print $3/$2*100}'

2.54155

[manju@localhost ~]$ free -t | awk 'FNR == 3 {print $3/$2*100}'

2.54155
```

Display Swap Utilization

```
[manju@localhost ~]$ free -t | awk 'FNR == 2 {printf("%.2f% \n"), $3/$2*100}'

61.86%

[manju@localhost ~]$ free -t | awk 'NR == 2 {printf("%.2f% \n"), $3/$2*100}'

61.86%
```

Display Memory Utilization with Percent Symbol and two decimal places

```
[manju@localhost ~]$ free -t | awk 'FNR == 3 {printf("%.2f% \n"), $3/$2*100}'

2.65%

[manju@localhost ~]$ free -t | awk 'NR == 3 {printf("%.2f% \n"), $3/$2*100}'

2.65%
```

Display Swap Utilization with Percent Symbol and two decimal places

```
[manju@localhost ~]$ top -b -n1 | grep ^%Cpu | awk '{cpu+=$9}END{print 100-cpu/NR}'

100
```

Display CPU Utilization

```
[manju@localhost ~]$ top -b -n1 | grep ^%Cpu | awk '{cpu+=$9}END{printf("%.2f% \n"), 100-cpu/NR}'

100.00%
```

Display CPU Utilization with Percent Symbol and two decimal places

```
swapon -s

# Print swap usage summaries




swapon -a

# Activate all of swap space
```

```
alias -p

# List all Aliases



lsof -i :8080

# Check which process is running on port 8080
```

```
swapoff -a

# Deactivate all of swap space



[manju@localhost ~]$ cat /etc/system-release

CentOS Linux release 7.3.1611 (Core)
```

Display the version of CentOS

```
sudo netstat -anp | grep tcp | grep LISTEN

# Display the various in-use ports and the process using it



sudo netstat -anp | grep 8080

# Display the process listening on port 8080
```

```
printf "%s\n" *

# Prints the files and directories that are in the current directory



printf "%s\n" */

# Prints only the directories in the current directory



printf "%s\n" *.{gif,jpg,png}

# Lists only some image files
```

```
[manju@localhost ~]$ alias x='date'   # create an alias
[manju@localhost ~]$ x   # preview the alias
Fri Oct  7 03:51:39 PDT 2022
[manju@localhost ~]$ unalias x   # remove the alias
[manju@localhost ~]$ x
bash: x: command not found...
```

```
[manju@localhost ~]$ x="alan"; printf '%s\n' "${x^}"

Alan

[manju@localhost ~]$ x="alan"; printf '%s\n' "${x^^}"

ALAN

[manju@localhost ~]$ x="alan"; declare -u name="$x"; echo "$name"

ALAN
```

```
find . -name "xyz[a-z][0-9]"

        Find directories and files with names starting with "xyz" and
                ending with an alpha character after a one-digit


find . -mmin -120

# Search for files changed during the previous two hours


find . -mmin +120

# Search for files that haven't been updated in the past two hours


    find . -mtime -3

    # Find files that have been modified within the last 3 days



    find . -mtime +3

    # Find files that have not been modified within the last 3 days
```

```
[manju@localhost ~]$ names="Albert Alan John Mary"; x=(${names// / }); echo ${x[0]}

Albert

[manju@localhost ~]$ names="Albert Alan John Mary"; x=(${names// / }); echo ${x[3]}

Mary
```

```
names="Albert+Alan+John+Mary";

x=(${names//+/ });

echo ${x[0]}

# Output: Albert


names="Albert+Alan+John+Mary";

x=(${names//+/ });

echo ${x[3]}

# Output: Mary
```

```
x=(hello world); echo "${x[@]/#/A}"

# Output: Ahello Aworld
```

```
[manju@localhost ~]$ awk '{print $2}' <<< "Alan Mathison Turing"

Mathison

[manju@localhost ~]$ awk '{print $1}' <<< "Alan Mathison Turing"

Alan
```

```
x='4 * 2'; echo "$x"

# prints 4 * 2

x='4 * 2'; echo $x

# prints 4, the list of files in the current directory, and 2
```

316

```
x='4 * 2'; echo "$(($x))"

# prints 8
```

```
[manju@localhost ~]$ x="ALAN"; printf '%s\n' "${x,}"

aLAN


[manju@localhost ~]$ x="ALAN"; printf '%s\n' "${x,,}"

alan
```

```
[manju@localhost ~]$ x="Alan"; echo "${x~~}"

aLAN

[manju@localhost ~]$ x="Alan"; echo "${x~}"

alan
```

```
[manju@localhost ~]$ x='You are a genius'; echo "${x/a/A}"

You Are a genius

[manju@localhost ~]$ x='You are a genius'; echo "${x//a/A}"

You Are A genius
```

```
[manju@localhost ~]$ x='You are a genius'; echo "${x/%s/N}"

You are a geniuN


[manju@localhost ~]$ x='You are a genius'; echo "${x/s/}"

You are a geniu
```

```
[manju@localhost ~]$ x='You are a genius'; echo "${x#*a}"

re a genius

[manju@localhost ~]$ x='You are a genius'; echo "${x#*g}"

enius
```

```
[manju@localhost ~]$ foo=25; i=foo; echo ${i}

foo

[manju@localhost ~]$ foo=25; i=foo; echo ${!i}

25
```

```
[manju@localhost ~]$ x='You are a genius'; echo "${x%a*}"

You are

[manju@localhost ~]$ x='You are a genius'; echo "${x%%a*}"

You
```

```
[manju@localhost ~]$ x=Bob-Dev-Fox; echo ${x%%-*}

Bob

[manju@localhost ~]$ x=Bob-Dev-Fox; echo ${x%-*}

Bob-Dev

[manju@localhost ~]$ x=Bob-Dev-Fox; echo ${x##*-}

Fox

[manju@localhost ~]$ x=Bob-Dev-Fox; echo ${x#*-}

Dev-Fox
```

```
find . -type f -path '*/Documents/*'

# Find only files within a folder called Documents



find . -type f -path '*/Documents/*' -o -path '*/ Downloads/*'

# Find only files within a folder called Documents or Downloads
```

```
find . -type f -not -path '*/Documents/*'

# Find all files except the ones contained in a folder called Documents



find . -type f -not -path '*log' -not -path '*/Documents/*'

# Find all files except the ones contained in a folder called Documents or log files
```

```
[manju@localhost ~]$ find /dev -type b
/dev/sr0
/dev/sda3
/dev/sda2        Block devices
/dev/sda1
/dev/sda

[manju@localhost ~]$ echo '16 / 5' | bc

3

[manju@localhost ~]$ echo '16 / 5' | bc -l

3.20000000000000000000


find . -maxdepth 1 -type f -name "*.txt"

# Find every.txt file from the current directory alone



[manju@localhost ~]$ echo "$(printf "%04d" "${x}")"

0000
```

```
[manju@localhost ~]$ echo "$(printf "%05d" "${x}")"

00000


[manju@localhost ~]$ echo "\"'\""

"'"
```

```
[manju@localhost ~]$ echo '3 5 + p' | dc

8

[manju@localhost ~]$ dc <<< '3 5 + p'

8

[manju@localhost ~]$ echo '3 5 * p' | dc

15

[manju@localhost ~]$ dc <<< '3 5 * p'

15
```

```
[manju@localhost ~]$ expr 'Alan Turing' : 'Ala\(.*\)ring'

n Tu

[manju@localhost ~]$ echo '12 == 12 && 18 > 12' | bc

1 (True)

[manju@localhost ~]$ echo '12 == 13 && 18 > 12' | bc

0 (False)
```

```
[manju@localhost ~]$ expr PQRSTUVWXYZ : PQRS

4     Display the number of matching characters



ls -ral

# Listing of all files in reverse alphabetical order
```

- ls -tl
- ls -trl

List the files such that the one that was most recently edited is at the top of the list

```
find . -regex ".*\(\.sh\|\.txt\)$"

# Find .sh or .txt files


[manju@localhost ~]$ find . -iregex ".*\(\.sh\|\.pdf\)$"

./bc.pdf

./1.PDF

./data.sh

./1.sh

./2.SH

./1.pdf

./2.sh
```

```
find . -type f -print

# List only regular files



[manju@localhost ~]$ echo "alan+alan+alan+alan" | xargs -d +

alan alan alan alan

[manju@localhost ~]$ echo "alan+alan+alan+alan" | xargs -d + -n 2

alan alan

alan alan
```

```
[manju@localhost ~]$ echo -e "2\nalbert\n" > 1.txt

[manju@localhost ~]$ cat 1.txt

2

albert
```

- ps -eLf --sort -nlwp | head
- ps -eLf

Display information about process threads

```
systemctl -l -t service | less

# List all Systemd services


[manju@localhost ~]$ echo -e "Albert\nTesla\nJohn"

Albert

Tesla

John


[manju@localhost ~]$ echo -e "Albert\nTesla\nJohn" | nl

     1    Albert

     2    Tesla

     3    John
```

```
[manju@localhost ~]$ echo -e "Albert\nTesla\nJohn" | nl -s ": " -w 1

1: Albert

2: Tesla

3: John
```

```
whiptail --yesno "Do you wish to proceed?" 10 40

# Display a simple yes or no input box on the command-line with whiptail
```

```
du -h -d1

# Check only the current directory's file space usage.



sudo nmcli networking off

sudo nmcli networking on
```

Restart network service
using nmcli tools

```
find / - manju

# Find files and directories owned by user "manju"



locate "*.png"

# Find all files containing '.png' in the name



find . -name '*.txt' -type f -delete
```

Find all files with '.txt' extension in the current
directory, including subdirectories and delete them

```
find . -type f -printf "\"%p\" " | xargs chmod 664

# Find all files in the current directory, including subfolders and assign rights 664


find . -type d -printf "\"%p\" " | xargs chmod 775

# Find all files in the current directory, including subfolders and assign rights 775
```

```
ll

# List the files in current directory                    ls -l
```

```
sudo netstat -nltp

# Display all open ports by process


date +%j

# Convert current date into Julian format


date -d "2022/03/13" +%j

# Convert a specific date into Julian format
```

```
date +%Y%m%d

# Display current date in YYYYMMDD format


date +%d\/%m\/%Y

# Display current date in DD/MM/YYYY format
```

C Exercises

Dennis Ritchie, known as the "Father of C Low-Level Programming Language," created the general-purpose, procedural, imperative computer programming language **"C"** in 1972 at the Bell Telephone Laboratories to be used with the UNIX operating system. It now ranks among the most popular programming languages after spreading to numerous different operating systems. Many other well-known languages, including C++, which was initially created as an improvement to C, have also been strongly inspired by C. Though it is also widely used for creating applications, it is the most frequently used programming language for creating system software. It is one of the programming languages that is most frequently used today. Since 1989, C has been standardized by both the International Organization for Standardization and the American National Standards Institute. Don't worry if you are a beginner; we have exercises for you. We'll concentrate on beginner-level programming problems in this chapter to help you learn C and develop your programming abilities.

Question 1

Question:

Write a program to print Hello, World!.

Solution:

```
#include<stdio.h>
int main() {
printf("Hello, World!");
return 0;
}
```

```
#include<stdio.h>
int main() {
int a = 6;
{
int a = 2;
printf("%d\n", a);
}
printf("%d\n", a);
}
```

Output:

2

6

Question 2

Question:

Write a program to compute the perimeter and area of a rectangle.

Solution:

```
#include<stdio.h>
int main() {
int height = 8;
int width = 5;
```

```c
int perimeter = 2*(height + width);
printf("Perimeter of the rectangle is: %d cm\n", perimeter);
int area = height * width;
printf("Area of the rectangle is: %d square cm\n", area);
return 0;
}
```

Question 3

Question:

Write a program to compute the perimeter and area of a circle.

Solution:

```c
#include<stdio.h>
int main() {
int radius = 4;
float perimeter = 2*3.14*radius;
printf("Perimeter of the circle is: %f cm\n", perimeter);
float area = 3.14*radius*radius;
printf("Area of the circle is: %f square cm\n", area);
return 0;
}
```

Numerous additional programming languages, including C++, Java, JavaScript, Go, C#, PHP, Python, Perl, C-shell, and many others, are based on C.

Question 4

Question:

Write a program that accepts two numbers from the user and calculate the sum of the two numbers.

Solution:

```c
#include<stdio.h>
int main() {
int a, b, sum;
printf("\nEnter the first number: ");
scanf("%d", &a);
printf("\nEnter the second number: ");
scanf("%d", &b);
sum = a + b;
printf("\nSum of the above two numbers is: %d", sum);
return 0;
}
```

```c
#include<stdio.h>

int main() {

int a;

printf("%d", a);

return 0;

}
```

In C Language: if the variable is not assigned a value, it takes a garbage value.

Question 5

Question:

Write a program that accepts two numbers from the user and calculate the product of the two numbers.

Solution:

```c
#include<stdio.h>
int main() {
int a, b, mult;
printf("\nEnter the first number: ");
scanf("%d", &a);
printf("\nEnter the second number: ");
scanf("%d", &b);
mult = a * b;
printf("\nProduct of the above two numbers is: %d", mult);
return 0;
}
```

```c
#include<stdio.h>
int Message() {
printf("Hello, World!");
return 0;
}
int main() {
Message();
}
```

Output:

    Hello, World!

Question 6

Question:

Write a program that accepts three numbers and find the largest of three.

Solution:

```c
#include<stdio.h>
int main() {
int x, y, z;
printf("\nEnter the first number: ");
scanf("%d", &x);
```

```c
printf("\nEnter the second number: ");
scanf("%d", &y);
printf("\nEnter the third number: ");
scanf("%d", &z);

// if x is greater than both y and z, x is the largest
if (x >= y && x >= z)
printf("\n%d is the largest number.", x);

// if y is greater than both x and z, y is the largest
if (y >= x && y >= z)
printf("\n%d is the largest number.", y);

// if z is greater than both x and y, z is the largest
if (z >= x && z >= y)
printf("\n%d is the largest number.", z);

return 0;
}
```

Question 7

Question:

Write a program that reads three floating values and check if it is possible to make a triangle with them. Also calculate the perimeter of the triangle if the entered values are valid.

Solution:

```c
#include<stdio.h>
int main() {
float  x, y, z;
printf("\nEnter the first number: ");
scanf("%f", &x);
printf("\nEnter the second number: ");
scanf("%f", &y);
printf("\nEnter the third number: ");
scanf("%f", &z);

if(x < (y+z) && y < (x+z) && z < (y+x)) {
printf("\nPerimeter of the triangle is: %f\n", x+y+z);
}
else {
printf("\nIt is impossible to form a triangle.");
}
return 0;
}
```

```c
/* Hello World /* Program in C*/ */

#include<stdio.h>

int main() {

printf("Hello World");

return 0;

}
```

Comments cannot be nested.

Error

Question 8

Question:

Write a program that reads an integer between 1 and 7 and print the day of the week in English.

Solution:

```
#include<stdio.h>
int main() {
int day;
printf("\nEnter a number between 1 to 7 to get the day name: ");
scanf("%d", &day);
switch(day) {
case 1 : printf("Monday\n"); break;
case 2 : printf("Tuesday\n"); break;
case 3 : printf("Wednesday\n"); break;
case 4 : printf("Thursday\n"); break;
case 5 : printf("Friday\n"); break;
case 6 : printf("Saturday\n"); break;
case 7 : printf("Sunday\n"); break;
default : printf("Enter a number between 1 to 7.");
}
return 0;
}
```

Question 9

Question:

Write a program to find the sum of two numbers.

Solution:

> As it only supports scalar operations, C is now often regarded as a low level language among programmers, contrary to how it was once thought to be a high level language.

```
#include<stdio.h>
int main() {
int a, b, sum;
```

```c
a=1;
b=2;
sum = a + b;
printf("The sum of a and b = %d", sum);
return 0;
}
```

Question 10

Question:

Write a program to find the square of a number.

Solution:

```c
#include<stdio.h>
#include<math.h>
int main() {
int a, b;
a=2;
b = pow((a), 2);
printf("The square of a = %d", b);
return 0;
}
```

```c
#include<stdio.h>
extern int a;
int main()
{
printf("a = %d", a);
}
int a = 1;
```

Output:

a = 1

Question 11

Question:

Write a program to find the greatest of two numbers.

Solution:

```c
#include<stdio.h>
int main() {
int a, b;
a = 2;
b = 3;
if(a>b) {
printf("a is greater than b");
}
else {
printf("b is greater than a");
}
return 0;
}
```

```c
#include<stdio.h>

#define merge(a, b) a##b

int main()
{
 printf("%d ", merge(12, 09));
 return 0;
}
```

Output:

1209

Question 12

Question:

Write a program to print the average of the elements in the array.

Solution:

```c
#include<stdio.h>
int main() {
int i, avg, sum = 0;
int num [5] = {16, 18, 20, 25, 36};
for(i=0; i<5; i++) {
sum = sum + num [i];
avg = sum/5;
}
printf("Sum of the Elements in the array is: %d\n", sum);
printf("Average of the elements in the array is: %d\n", avg);
return 0;
}
```

Game theory is a mathematical framework used to study decision-making in situations where the outcomes of one person's choices depend on the choices made by others. It is the study of mathematical representations of rational decision-makers interacting strategically. The goal of game theory is to identify the optimal strategies that players can use to achieve their objectives in situations where the choices of one player affect the outcomes for all players involved. Game theory is used in various fields such as economics, political science, psychology, and biology.

Question 13

Question:

Write a program that prints all even numbers between 1 and 25.

Solution:

```c
#include<stdio.h>
int main() {
printf("Even numbers between 1 to 25:\n");
for(int i = 1; i <= 25; i++) {
if(i%2 == 0) {
printf("%d ", i);
```

```
}
}
return 0;
}
```

The most widely used operating system, Linux, has a C-based kernel.

Question:

Write a program that prints all odd numbers between 1 and 50.

Solution:

```c
#include<stdio.h>
int main() {
printf("Odd numbers between 1 to 50:\n");
for(int i = 1; i <= 50; i++) {
if(i%2 != 0) {
printf("%d ", i);
}
}
return 0;
}
```

```c
#include<stdio.h>
int main() {
char c = 'a';
putchar(c);
return 0;
}
```

Output:

a

Question 15

Question:

Write a program to print the first 10 numbers starting from one together with their squares and cubes.

Solution:

```c
#include<stdio.h>
int main() {
for(int i=1; i<=10; i++) {
printf("Number = %d its square = %d its cube = %d\n", i , i*i, i*i*i);
}
return 0;
}
```

Question 16

Question:

Write a program:

If you enter a character M

Output must be: ch = M.

Solution:

```c
#include<stdio.h>

int main() {

int a, b;

for(a=1; a<=5; a++) {

for(b=1; b<=a; b++)

printf("%d", b);

printf("\n");

}

return 0;

}
```

Output:

1

12

123

1234

12345

```c
#include<stdio.h>
int main() {
char M;
printf("Enter any character: ");
scanf("%c", &M);
printf("ch = %c", M);
return 0;
}
```

```c
#include<stdio.h>
int main() {
int x = 50, y, z;
if(x >= 50) {
y = 15;
z = 28;
printf("\n%d %d", y, z);
}
return 0;
}
```

Output:

15 28

Question 17

Question:

Write a program to print the multiplication table of a number entered by the user.

Solution:

```c
#include<stdio.h>
int main() {
int n, i;
printf("Enter any number: ");
scanf("%d", &n);
for(i=1; i<=5; i++) {
printf("%d * %d = %d\n", n, i, n*i);
}
return 0;
}
```

```c
#include<stdio.h>
int main () {
float a;
a = (float) 51/4;
printf("%f", a);
return 0;
}
```

Output:

12.750000

Question 18

Question:

Write a program to print the product of the first 10 digits.

Solution:

> The only ternary operator in the C language is **"?:"**

```c
#include<stdio.h>
int main() {
int i, product = 1;
for(i=1; i<=10; i++) {
product = product * i;
}
printf("The product of the first 10 digits is: %d", product);
return 0;
}
```

Question 19

Question:

Write a program to print whether the given number is positive or negative.

Solution:

```c
#include<stdio.h>
```

```c
int main() {
int a;
a = -35;
if(a>0) {
printf("Number is positive");
}
else {
printf("Number is negative");
}
return 0;
}
```

```c
#include<stdio.h>
int main() {
char name[] = "Einstein";
printf("%c", name[0]);
return 0;
}
```

Output:

E

Question 20

Question:

Write a program to check the equivalence of two numbers entered by the user.

Solution:

```c
#include<stdio.h>
int main() {
int x, y;
printf("\nEnter the first number: ");
scanf ("%d", &x);
printf("\nEnter the second number: ");
scanf ("%d", &y);
if(x-y==0) {
printf("\nThe two numbers are equivalent");
```

```
}
else {
printf("\nThe two numbers are not equivalent");
}
return 0;
}
```

Question 21

Question:

Write a program to print the remainder of two numbers entered by the user.

Solution:

> **"sizeof" is the only operator which is also a keyword.**

```
#include<stdio.h>
int main() {
int a, b, c;
printf("\nEnter the first number: ");
scanf ("%d", &a);
printf("\nEnter the second number: ");
scanf ("%d", &b);
c = a%b;
printf("\nThe remainder of %d and %d is: %d", a, b, c);
return 0;
}
```

Question 22

Question:

Write a program to print the characters from A to Z.

Solution:

```
#include<stdio.h>
int main() {
char i;
for(i='A'; i<='Z'; i++) {
printf("%c\n", i);
}
return 0;
}
```

```
#include<stdio.h>
int main() {
char name[] = "Einstein";
name[0] = 'H';
printf("%s", name);
return 0;
}
```

Output:

Hinstein

Question 23

Question:

Write a program to print the length of the entered string.

Solution:

```
#include<stdio.h>
#include<string.h>
```

```c
int main() {
char str[1000];
printf("Enter a string to calculate its length: ");
scanf("%s", str);
printf("The length of the entered string is: %ld", strlen(str));
return 0;
}
```

Question 24

Question:

Write a program to check whether the given character is a lower case letter or not.

Solution:

In printf() and scanf(), f stands for formatted.

```c
#include<stdio.h>
#include <ctype.h>
int main() {
char ch = 'a';
if(islower(ch)) {
printf("The given character is a lower case letter");
}
else {
printf("The given character is a upper case letter");
}
return 0;
}
```

Question 25

Question:

Write a program to check whether the given character is a upper case letter or not.

Solution:

> There must be a function named main() in every C program.

```c
#include<stdio.h>
#include <ctype.h>
int main() {
char ch = 'A';
if(isupper(ch)) {
printf("The given character is a upper case letter");
}
else {
printf("The given character is a lower case letter");
}
return 0;
}
```

Question 26

Question:

Write a program to convert the lower case letter to upper case letter.

Solution:

```c
#include<stdio.h>
#include <ctype.h>
int main() {
char ch = 'a';
char b = toupper(ch);
printf("Lower case letter '%c' is converted to Upper case letter '%c'", ch, b);
return 0;
}
```

> In a C program, any number of functions can be written. In C, there are two different sorts of functions: user-defined functions and library functions.

Question 27

Question:

Write a program that takes a distance in centimeters and outputs the corresponding value in inches.

Solution:

```c
#include<stdio.h>
#define x 2.54
int main() {
double inch, cm;
printf("Enter the distance in cm: ");
scanf("%lf", &cm);
```

```c
#include<stdio.h>
int main() {
int i = 6;
while(i == 3) {
i = i - 3;
printf ("%d\n", i);
--i;
}
return 0;
}
```

Output:

?

```
inch = cm / x;
printf("\nDistance of %0.2lf cms is equal to %0.2lf inches", cm, inch);
return 0;
}
```

Question 28

Question:

Write a program to print the output:

Einstein [0] = E

Einstein [1] = I

Einstein [2] = N

Einstein [3] = S

Einstein [4] = T

Einstein [5] = E

Einstein [6] = I

Einstein [7] = N

```
#include<stdio.h>
int main() {
const int i = 54;
printf("%d", i);
return 0;
}
```

Output:

54

Solution:

```
#include<stdio.h>
int main() {
char name [8] = {'E' , 'I', 'N', 'S', 'T', 'E', 'I', 'N'};
for(int i=0; i<8; i++) {
printf("\nEinstein [%d] = %c", i, name[i]);
}
return 0;
```

```
}
```

Question 29

Question:

Write a program to print "Hello World" 10 times.

Solution:

```c
#include<stdio.h>
int main() {
for(int i=1; i<=10; i++) {
printf("Hello World \n");
}
return 0;
}
```

```c
#include<stdio.h>
int main() {
int num[] = {5, 7, 9, 42};
printf("%d", num[0]);
return 0;
}
```

Output:

5

Question 30

Question:

Write a program to print first 5 numbers using do while loop statement.

Solution:

```c
#include<stdio.h>

int main() {

int i =1;

do {

printf("%d\n", i++);

} while(i<=5);

return 0;

}
```

```c
#include<stdio.h>

int main () {

char name[9] = {'C', 'P', 'r', 'o', 'g', 'r', 'a', 'm', '\0'};

printf("%s\n", name);

return 0;

}
```

Output:

   CProgram

Question 31

Question:

Write a program to check whether a character is an alphabet or not.

Solution:

```c
#include<stdio.h>
#include<ctype.h>
int main() {
int a =2;
if(isalpha(a)) {
printf("The character a is an alphabet");
}
else {
printf("The character a is not an alphabet");
}
return 0;
}
```

```c
#include<stdio.h>

#define SIZE 3

int main() {

char names[SIZE][8] = {

"Mary",

"Albert",

"John"

};

int i;

for(i=0; i<SIZE; i++)

puts(names[i]);

return 0;

}
```

Output:

   Mary

   Albert

   John

Question 32

Question:

Write a program to check whether a entered number is even or odd.

Solution:

```c
#include<stdio.h>
int main() {
int a;
printf("Enter any number: ");
scanf ("%d", &a);
if(a%2 == 0) {
printf("The entered number is even");
}
else {
printf("The entered number is odd");
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

int num[] = {5, 7, 9, 42};

num[0] = 3;

printf("%d", num[0]);

return 0;

}
```

Output:

3

Question 33

Question:

Write a program to print the ASCII value of the given character.

Solution:

```c
#include<stdio.h>
int main() {
char ch ='A';
printf("The ASCII value of %c is: %d", ch, ch);
return 0;
}
```

Question 34

Question:

Write a program that will print all numbers between 1 to 50 which divided by a specified number and the remainder will be 2.

Solution:

```c
#include<stdio.h>
int main() {
int x, i;
printf("Enter a number: ");
scanf("%d", &x);
for(i=1; i<=50; i++) {
   if((i%x)==2) {
    printf("%d\n", i);
     }
```

```c
#include<stdio.h>
int main() {
int i = 25;
printf("%p", &i);
return 0;
}
```

Program to get the memory address of a variable "i"

```
}
return 0;
}
```

Question 35

Question:

Write a program to determine whether two numbers in a pair are in ascending or descending order.

Solution:

```c
#include<stdio.h>
int main() {
int a, b;
printf("\nEnter a pair of numbers (for example 22,12 | 12,22): ");
printf("\nEnter the first number: ");
scanf("%d", &a);
printf("\nEnter the second number: ");
scanf("%d", &b);
if (a>b) {
printf("\nThe two numbers in a pair are in descending order.");
}
else {
printf("\nThe two numbers in a pair are in ascending order.");
}
return 0;
}
```

Question 36

Question:

Write a program that reads two numbers and divides one by the other. Specify "Division not possible" if that is not possible.

Solution:

```c
#include<stdio.h>
int main() {
int a, b;
float c;
printf("\nEnter the first number: ");
scanf("%d", &a);
printf("\nEnter the second number: ");
scanf("%d", &b);
if(b != 0) {
    c = (float)a/(float)b;
    printf("\n%d/%d = %.1f", a, b, c);
}
else {
    printf("\nDivision not possible.\n");
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

int a = 6;

float b = 6.0;

if(a == b) {

printf("\na and b are equal");

}

else {

printf("\na and b are not equal");

}

return 0;

}
```

Output:

a and b are equal

Question 37

Question:

Write a program that will print all numbers between 1 to 50 which divided by a specified number and the remainder is equal to 2 or 3.

Solution:

```
#include<stdio.h>
int main() {
int x, i;
printf("Enter a number: ");
scanf("%d", &x);
for(i=1; i<=50; i++) {
   if((i%x)==2 || (i%x) == 3) {
    printf("%d\n", i);
     }
}
return 0;
}
```

```
#include<stdio.h>
int main() {
int a = 15, b, c;
b = a = 25;
c = a < 25;
printf ("\na = %d b = %d c = %d", a, b, c);
return 0;
}
```

Output:

a = 25 b = 25 c = 0

Question 38

Question:

Write a program that adds up all numbers between 1 and 100 that are not divisible by 12.

Solution:

```c
#include<stdio.h>
int main() {
int x =12, i, sum = 0;
for(i=1; i<=100; i++) {
   if((i%x)!= 0) {
    sum += i;
     }
}
printf("\nSum: %d\n", sum);
return 0;
}
```

```c
#include<stdio.h>
int main() {
int x = 67;
char y = 'C';
if(x == y) {
printf("Albert Einstein");
}
else {
printf("Elsa Einstein");
}
return 0;
}
```

Output:

Albert Einstein

Question 39

Question:

Write a program to calculate the value of x where x = 1 + 1/2 + 1/3 + ... + 1/50.

Solution:

```c
#include<stdio.h>
int main() {
float x = 0;
for(int i=1; i<=50; i++) {
   x += (float)1/i;
}
printf("Value of x: %.2f\n", x);
```

```c
return 0;
}
```

Question 40

Question:

Write a program that reads a number and find all its divisor.

Solution:

```c
#include<stdio.h>
int main() {
int x, i;
printf("\nEnter a number: ");
scanf("%d", &x);
printf("All the divisor of %d are: ", x);
for(i = 1; i <= x; i++) {
    if((x%i) == 0) {
    printf("\n%d", i);
    }
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

int a = 20, b = 25;

if(a % 2 == b % 5) {

printf("\nPeru");

}

return 0;

}
```

Output:

Peru

Question 41

Question:

Write a program to find the incremented and decremented values of two numbers.

Solution:

```c
#include<stdio.h>
int main() {
int a, b, c, d, e, f;
a = 10;
b=12;
c=a+1;
d=b+1;
e=a-1;
f=b-1;
printf("\nThe incremented value of a =%d", c);
printf("\nThe incremented value of b =%d", d);
printf("\nThe decremented value of a =%d", e);
printf("\nThe decremented value of b =%d", f);
return 0;
}
```

Graph theory is an essential field of study in computer science that deals with the mathematical modeling and analysis of networks or graphs. In computer science, graphs are used to represent and analyze various real-world systems such as communication networks, social networks, transportation networks, and many others. Graph theory provides a set of tools and techniques for solving complex problems that arise in computer science, including algorithm design, network analysis, data structures, and many others.
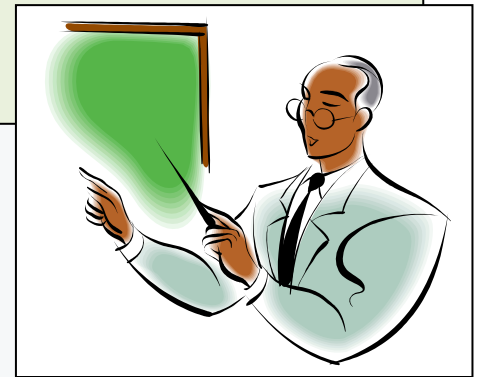
Question 42

Question:

Write a program to find square of a entered number using functions.

Solution:

```c
#include<stdio.h>
int square();
int main() {
int answer;
answer = square();
printf("The square of the entered number is: %d", answer);
return(0);
}
int square() {
int x;
printf("Enter any number: ");
scanf("%d", &x);
return x*x;
}
```

```c
#include<stdio.h>

int main() {

int a = 6, b, c;

b = ++a;

c = a++;

printf ("%d %d %d\n", a, b, c);

return 0;

}
```

Output:

8 7 7

Question 43

Question:

Write a program that accepts principal amount, rate of interest, time and compute the simple interest.
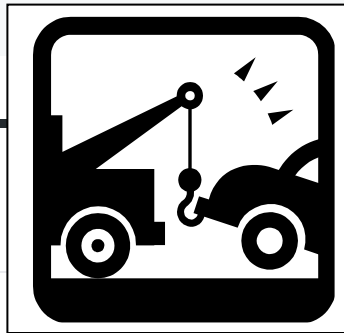
Solution:

```c
#include<stdio.h>
```

```c
int main() {
int p,r,t,SI;
printf("\nEnter the principal amount: ");
scanf("%d",&p);
printf("\nEnter the rate of interest: ");
scanf("%d",&r);
printf("\nEnter the time: ");
scanf("%d",&t);
SI=(p*r*t)/100;
printf("\nSimple interest is: %d", SI);
return 0;
}
```

Mathematical logic and **Boolean logic** are two different branches of logic, although they share some similarities.

The formal study of mathematical reasoning and proof is what **mathematical logic** is all about. It includes propositional logic, predicate logic, set theory, and other related areas. Mathematical logic is used to develop rigorous mathematical proofs and to study the foundations of mathematics.

Boolean logic, on the other hand, is a type of algebraic logic that deals with the manipulation and evaluation of logical statements. It is based on the binary values of true and false, or 1 and 0, and uses logical operators such as AND, OR, and NOT to combine and manipulate logical statements. Boolean logic is widely used in computer science, digital electronics, and other areas where logical reasoning is important.

To sum up, **mathematical logic** is concerned with the study of mathematical reasoning and proof, while **Boolean logic** is concerned with the manipulation and evaluation of logical statements. Both branches of logic have important applications in various fields, including computer science and mathematics.

Question 44

Question:

Write a program that swaps two numbers without using third variable.

Solution:

```c
#include<stdio.h>
int main() {
int a, b;
printf("\nEnter the value for a: ");
scanf("%d",&a);
printf("\nEnter the value for b: ");
scanf("%d",&b);
printf("\nBefore swapping: %d %d",a,b);
```

```c
a=a+b;
b=a-b;
a=a-b;
printf("\nAfter swapping: %d %d",a,b);
return 0;
}
```

Question 45

Question:

Write a program to find the greatest of two entered numbers using pointers.

Solution:

```c
#include<stdio.h>
int main() {
int x, y, *p, *q;
printf("Enter the value for x: ");
scanf("%d", &x);
printf("Enter the value for y: ");
scanf("%d", &y);
p = &x;
q = &y;
if(*p>*q) {
printf("x is greater than y");
}
if(*q>*p) {
printf("y is greater than x");
```

```c
#include<stdio.h>
int main() {
int a = 15, b = 30;
if(a == b) {
printf("a = b");
}
else if(a > b) {
printf("a > b");
}
else if(a < b) {
printf("a < b");
}
return 0;
}
```

Output:

a < b

```
}
return 0;
}
```

Question 46

Question:

Write a program to print the output:

body [b] = b

body [o] = o

body [d] = d

body [y] = y

```
#include<stdio.h>
int main() {
int i = 60;
if(i > 70 && i < 100) {
printf("i is greater than 70 and less than 100");
}
else {
printf("%d", i);
}
return 0;
}
```

Output:

60

Solution:

```
#include <stdio.h>
int main() {
char i;
char body [4] = {'b', 'o', 'd', 'y'};
for(i=0; i<4; i++)
printf("\n body[%c] = %c", body[i] , body[i]);
return 0;
}
```

Question 47

Question:

Write a program to calculate the discounted price and the total price after discount

Given:

If purchase value is greater than 1000, 10% discount

If purchase value is greater than 5000, 20% discount

If purchase value is greater than 10000, 30% discount.

Solution:

```c
#include<stdio.h>
int main() {
double PV;
printf("Enter purchased value: ");
scanf("%lf", &PV);
if(PV>1000) {
printf("\n Discount = %lf", PV* 0.1);
printf("\n Total = %lf", PV - PV* 0.1);
}
else if(PV>5000) {
printf("\n Discount = %lf", PV* 0.2);
printf("\n Total = %lf", PV - PV* 0.2);
}
else {
printf("\n Discount = %lf", PV* 0.3);
printf("\n Total = %lf", PV - PV* 0.3);
}
return 0;
```

```c
#include<stdio.h>

int main() {

printf("%%15s = %15s\n", "albert");

printf("%%14s = %14s\n", "albert");

printf("%%13s = %13s\n", "albert");

printf("%%12s = %12s\n", "albert");

printf("%%11s = %11s\n", "albert");

printf("%%10s = %10s\n", "albert");

printf(" %%9s = %9s\n", "albert");

printf(" %%8s = %8s\n", "albert");

printf(" %%7s = %7s\n", "albert");

printf(" %%6s = %6s\n", "albert");

printf(" %%5s = %5s\n", "albert");

printf(" %%4s = %4s\n", "albert");

return(0);

}
```

?

```
}
```

Question 48

Question:

Write a program to print the first ten natural numbers using while loop statement.

Solution:

```c
#include<stdio.h>
int main() {
int i = 1;
while (i<=10) {
printf("%d\n", i++);
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

int i = 12;

if(i == 12 && i != 0) {

printf("\nHi");

printf("\nEinstein");

}

else {

printf( "Bye Elsa" );

}

return 0;

}
```

Output:

Hi

Einstein

Question 49

Question:

Write a program to shift inputted data by two bits to the left.

Solution:

```c
#include<stdio.h>
int main() {
int x;
printf("Enter the integer from keyboard: ");
scanf("%d",&x);
printf("\nEntered value: %d ",x);
printf("\nThe left shifted data is: %d ", x<<=2);
return 0;
}
```
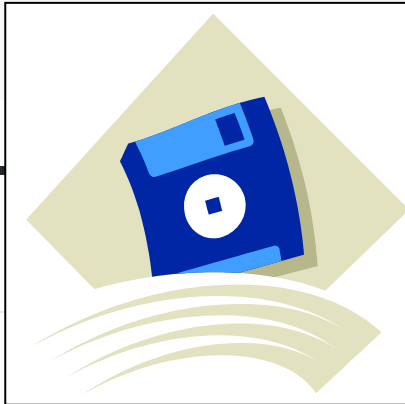
Question 50

Question:

Write a program to shift inputted data by two bits to the Right.

Solution:

```c
#include<stdio.h>
int main() {
int x;
printf("Enter the integer from keyboard: ");
scanf("%d",&x);
printf("\nEntered value: %d ",x);
printf("\nThe right shifted data is: %d ", x>>=2);
return 0;
}
```

An **algorithm** is a set of precise, step-by-step instructions or procedures for solving a problem or achieving a specific goal. Algorithms are used in many areas of science, engineering, and computing, where they help to automate processes and enable efficient computation.

An **algorithm** takes some input and produces some output, often involving some form of data manipulation or transformation. Algorithms are designed to be efficient and accurate, and they can be expressed in various forms, such as Pseudocode, flowcharts, or programming languages. **Algorithms** can range in complexity from simple calculations to complex machine learning models. Some examples of algorithms include sorting algorithms, search algorithms, encryption algorithms, and optimization algorithms. The development and analysis of algorithms are central to the fields of computer science and mathematics.

Question 51

Question:

Write a program to calculate the exact difference between x and 21. Return three times the absolute difference if x is greater than 21.

Solution:

> Because C is a structured (modular) programming language, programmers can divide their code into smaller chunks to make it easier to comprehend and, as a result, make their programs simpler and less redundant.

```c
#include<stdlib.h>
#include<stdio.h>
int main() {
int x;
printf("Enter the value for x: ");
scanf("%d",&x);
if(x<=21){
    printf("%d", abs(x-21));
 }
else if(x>=21) {
    printf("%d", abs(x-21)*3);
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

int x = 25, y;

x >= 16 ? (y = 25) : (y = 30);

printf ("\n%d %d", x, y);

return 0;

}
```

Output:

25 25

Question 52

Question:

Write a program that reads in two numbers and determine whether the first number is a multiple of the second number.

Solution:

```c
#include<stdio.h>
int main() {
int x, y;
printf("\nEnter the first number: ");
scanf("%d", &x);
printf("\nEnter the second number: ");
scanf("%d", &y);
if(x % y == 0) {
printf("\n%d is a multiple of %d.\n", x, y);
}
else {
printf("\n%d is not a multiple of %d.\n", x, y);
}
return 0;
}
```

```c
#include<stdio.h>
int main() {
int x = 10;
(x == 10 ? printf( "True" )  : printf( "False" ));
return 0;
}
```

Output:

True

Question 53

Question:

Write a program to print the output:

Name of the book = B

Price of the book = 135.00

Number of pages = 300

Edition of the book = 8

using structures.

Solution:

```c
#include<stdio.h>
int main() {
struct book {
char name;
float price;
int pages;
int edition;
};
struct book b1;
b1.name = 'B';
b1.price = 135.00;
b1.pages = 300;
b1.edition = 8;
printf("\n Name of the book = %c", b1.name);
printf("\n Price of the book = %f", b1.price);
printf("\n Number of pages = %d", b1.pages);
printf("\n Edition of the book = %d", b1.edition);
```

```c
#include<stdio.h>
int main() {
int num, x, y, z;
printf("Enter a three digit number: ");
scanf("%d", &num);
x=num%10;
y=(num/10)%10;
z=(num/100)%10;
printf("%d is the sum of the digits of the number %d.", x+y+z, num);
return 0;
}
```

Output:

?

```
return 0;
}
```

Question 54

Question:

Write a program to convert Celsius into Fahrenheit.

Solution:

```c
#include<stdio.h>
int main() {
float fahrenheit, celsius;
celsius = 36;
fahrenheit = ((celsius*9)/5)+32;
printf("\nTemperature in fahrenheit is:  %f", fahrenheit);
return 0;
}
```

Question 55

Question:

Write a program that will examine two inputted integers and return true if either of them is 50 or if their sum is 50.

Solution:

```c
#include<stdio.h>
int main() {
int x, y;
printf("\nEnter the value for x: ");
scanf("%d", &x);
printf("\nEnter the value for y: ");
scanf("%d", &y);
if(x == 50 || y == 50 || (x + y == 50)) {
    printf("\nTrue");
}
else {
    printf("\nFalse");
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

while(!printf("Albert Einstein")){}

return 0;

}
```

Output:

Albert Einstein

Question 56

Question:

Write a program that counts the even, odd, positive, and negative values among eighteen integer inputs.

Solution:

```c
#include<stdio.h>
int main () {
int x, even = 0, odd = 0, positive = 0, negative = 0;
printf("\nPlease enter 18 numbers:\n");
for(int i = 0; i < 18; i++) {
scanf("%d", &x);
if (x > 0) {
    positive++;
}
if(x < 0) {
    negative++;
}
if(x % 2 == 0) {
    even++;
}
if(x % 2 != 0) {
    odd++;
}
}
printf("\nNumber of even values: %d", even);
printf("\nNumber of odd values: %d", odd);
printf("\nNumber of positive values: %d", positive);
printf("\nNumber of negative values: %d", negative);
return 0;
}
```

```c
#include<stdio.h>
int main() {
int x = 0, y = 1 ;
if(x == 0) {
(y > 1 ? printf("\nHi") : printf ("\nAlbert"));
}
else {
printf("\nHi Albert!");
}
return 0;
}
```

Output:

Albert

```c
#include<stdio.h>
int main() {
switch(printf("Albert Einstein")){}
return 0;
}
```

Output:

Albert Einstein

Question 57

Question:

Write a program to check whether the person is a senior citizen or not.

Solution:

```c
#include<stdio.h>
int main() {
int age;
printf("Enter age: ");
scanf("%d", &age);
if(age>=60) {
printf("Senior citizen");
}
else {
printf("Not a senior citizen");
}
return 0;
}
```

```c
#include<stdio.h>
int main() {
int x;
printf("Enter any number: ");
scanf ("%d", &x);
if(x > 100) {
        printf ("\nAlbert");
}
else {
    if(x < 15)
        printf ("\nElsa");
    else
        printf ("\nDavid");
}
return 0;
}
```

Output:

David

Question 58

Question:

Write a program that reads a student's three subject scores (0-100) and computes the average of those scores.

Solution:

```c
#include<stdio.h>
int main() {
float score, total_score = 0;
int subject = 0;
printf("Enter three subject scores (0-100):\n");
while (subject != 3) {
scanf("%f", &score);
if(score < 0 || score > 100) {
printf("Please enter a valid score.\n");
}
else {
total_score += score;
subject++;
  }
}
printf("Average score = %.2f\n", total_score/3);
return 0;
}
```

Question 59

Question:

What results would the following programs produce?

```c
#include<stdio.h>
int main() {
for(int i=1; i<=5; i++) {
if(i==3) {
break;
}
printf("%d\n", i);
}
return 0;
}
```

```c
#include<stdio.h>
int main() {
int a = 15, b = 4;
float c = (float)a/(float)b;
printf("%d/%d = %.2f\n", a, b, c);
return 0;
}
```

Output:

15/4 = 3.75

Solution:

```
1
2
```

```c
#include<stdio.h>
int main() {
for(int i=1;i<=5;i++) {
if(i==3) {
goto HAI;
}
printf("\n %d ",i);
}
HAI : printf("\n Linux");
}
```

```c
#include<stdio.h>
int main() {
int a = 6, b = 4, c;
c = a++ +b;
printf ("\n%d %d %d", a, b, c);
return 0;
}
```

Output:

7 4 10

Solution:

```
1
2
Linux
```

```c
#include<stdio.h>
int main() {
for( ; ; ) {
printf("This loop will run forever.\n");
}
return 0;
}
```

Solution:

```
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever. .........
```

```c
#include<stdio.h>
int main() {
float i = 5.5;
while(i == 5.5) {
i = i - 0.8;
printf("\n%f", i);
}
return 0;
}
```

Output:

4.700000

```c
#include<stdio.h>
int main() {
printf("Hello,world!");
return 0;
```

```c
printf("Hello,world!");
}
```

Solution:

```
Hello,world!
```

```c
#include<stdio.h>
#include<stdlib.h>
int main () {
printf("linux\n");
exit (0);
printf("php\n");
return 0;
}
```

```c
#include<stdio.h>
int main() {
float i = 5.5;
while(i == 5.5) {
printf("\n%f", i);
i = i - 0.8;
}
return 0;
}
```

Output:

5.500000

Solution:

```
linux
```

```c
#include<stdio.h>
int main() {
for(int i=1; i<=5; i++) {
if(i==3) {
continue;
```

```
    }
    printf("%d\n ", i);
    }
    return 0;
}
```

Solution:

```
1
2
4
5
```

```
#include<stdio.h>
int main() {
int a = 10, b = 20, c;
c = (a < b) ? a : b;
printf("%d", c);
return 0;
}
```

Solution:

10

```
#include<stdio.h>

int main() {

int a = 6, b = 2;

while(a >= 0) {

a--;

b++;

if(a == b) {

continue;

}

else {

printf("\n%d %d", a, b);

}

}

return 0;

}
```
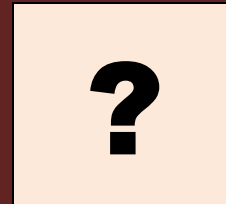
Output:

3 5

2 6

1 7

0 8

-1 9

```c
#include<stdio.h>
#define A 15
int main() {
int x;
x=A;
printf("%d", x);
return 0;
}
```

```c
#include<stdio.h>
int main() {
int x = 0;
for ( ; x ; )
printf ("\nAlbert");
return 0;
}
```

?

Solution:

15

```c
#include<stdio.h>
#include<stdlib.h>
int main() {
int i;
for(i=1; i <= 3; i++) {
printf((i&1) ? "odd\n" : "even\n");
}
exit(EXIT_SUCCESS);
}
```

Solution:

```
odd
even
odd
```

```c
#include<stdio.h>
#include<math.h>
int main() {
double a, b;
a = -2.5;
b = fabs(a);
printf("|%.2lf| = %.2lf\n", a, b);
return 0;
}
```

Solution:

```
|-2.50| = 2.50
```

```c
#include<stdio.h>
int main() {
int x;
float y = 2.0;
switch (x = y + 3) {
    case 5:
            printf("\nAlbert");
            break;
    default:
            printf("\nElsa");
}
return 0;

}
```

Output:

Albert

```c
#include<stdio.h>
#include<stdlib.h>
int main() {
int x=12, y =3;
```

```c
printf("%d\n", abs(-x-y));
return 0;
}
```

Solution:

15

```c
#include<stdio.h>
#include<stdlib.h>
int main() {
int x=12, y =3;
printf("%d\n", -(-x-y));
return 0;
}
```

Solution:

15

```c
#include<stdio.h>
#include<stdlib.h>
```

```c
#include<stdio.h>

int main() {

int x = 5;

switch (x - 6) {

    case -1 :

        printf("\nAlbert");

    case 0 :

        printf("\nJohn");

    case 1 :

        printf("\nMary");

    default :

        printf("\nJames");

    }

    return 0;

}
```

Output:

Albert

John

Mary

James

```
int main() {
int x=12, y =3;
printf("%d\n", x-(-y));
return 0;
}
```

Solution:

15

Question 60

Question:

Write a program to find the size of an array.

```
#include<stdio.h>

int main() {

int y[] = {20, 40, 60, 80, 100};

for(int x = 0; x <= 4; x++) {

printf("\n%d", *(y + x));

}

return 0;

}
```

Output:

20

40

60

80

100

Solution:

```
#include<stdio.h>
int main() {
int num [] = {11, 22, 33, 44, 55, 66};
int n = sizeof(num) / sizeof(num [0]);
printf("Size of the array is: %d\n", n);
return 0;
}
```

Question 61

Question:

Write a program that prints a sequence from 1 to a given integer, inserts a plus sign between these numbers, and then removes the plus sign at the end of the sequence.

Solution:

```c
#include<stdio.h>
int main () {
int x, i;
printf("\nEnter a integer: \n");
scanf("%d", &x);
if(x>0) {
printf("Sequence from 1 to %d:\n", x);
for(i=1; i<x; i++)  {
printf("%d+", i);
}
printf("%d\n", i);
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

char i[2] = "B";

printf("\n%c", i[0]);

printf("\n%s", i);

return 0;

}
```

Output:

B

B

Question 62

Question:

Write a program to verify whether a triangle's three sides form a right angled triangle or not.

Solution:

```c
#include<stdio.h>
int main() {
int a,b,c;
printf("Enter the three sides of a triangle: \n");
scanf("%d %d %d",&a,&b,&c);
if((a*a)+(b*b)==(c*c) || (a*a)+(c*c)==(b*b) || (b*b)+(c*c)==(a*a)) {
printf("Triangle's three sides form a right angled triangle.\n");
}
else {
printf("Triangle's three sides does not form a right angled triangle.\n");
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

printf("%c", "einstein"[4]);

return 0;

}
```

Output:

t

Question 63

Question:

Write a program that will find the second-largest number among the user's input of three numbers.

Solution:

```c
#include<stdio.h>
int main() {
int a, b, c;
printf("\nEnter the first number: ");
scanf("%d", &a);
printf("\nEnter the second number: ");
scanf("%d", &b);
printf("\nEnter the third number: ");
scanf("%d", &c);
if(a>b && a>c) {
    if(b>c)
            printf("\n%d is second largest number among three numbers", b);
        else
            printf("\n%d is second largest number among three numbers", c);
}
else if(b>c && b>a) {
    if(c>a)
            printf("\n%d is second largest number among three numbers", c);
        else
            printf("\n%d is second largest number among three numbers", a);
}
else if(a>b)
```

```
            printf("\n%d is second largest number among three numbers", a);
        else
            printf("\n%d is second largest number among three numbers", b);
    return 0;
}
```

Question 64

Question:

Write a program to calculate the sum of the two given integer values. Return three times the sum of the two values if they are equal.

Solution:

```c
#include<stdio.h>
int myfunc();
int main() {
printf("%d", myfunc(3, 5));
printf("\n%d", myfunc(6, 6));
return 0;
}
int myfunc(int a, int b) {
return a == b ? (a + b)*3 : a + b;
}
```

```c
#include<stdio.h>
int main()  {
int x = 53286, y=0, i;
while(x!=0) {
i=x%10;
y=y*10+i;
x/=10;
}
printf("%d", y);
return 0;
}
```

Output:

68235

Question 65

Question:

Write a program that accepts minutes as input, and display the total number of hours and minutes.

Solution:

```c
#include<stdio.h>
int main() {
int mins, hrs;
printf("Input minutes: ");
scanf("%d",&mins);
hrs=mins/60;
mins=mins%60;
printf("\n%d Hours, %d Minutes.\n", hrs, mins);
return 0;
}
```

```c
#include<stdio.h>
int main() {
float num[5] = { 11.5, 12.5, 13.5, 14.5, 15.5 };
printf("%.1f\n", *(num+1));
printf("%.1f\n", *(num+4));
return 0;
}
```

Output:

12.5

15.5

Question 66

Question:

Write a program to determine whether a positive number entered by the user is a multiple of three or five.

Solution:

```c
#include<stdio.h>
int main() {
int x;
printf("\nEnter a number: ");
scanf("%d", &x);
if(x % 3 == 0 || x % 5 == 0) {
printf("True");
}
else {
printf("False");
}
return 0;
}
```

```c
#include<stdio.h>

#define MULT(i) (i*i)

int main() {

int x = 6;

int a = MULT(x++);

int b = MULT(++x);

printf("\n%d %d", a, b);

return 0;

}
```

Output:

    42 100

Question 67

Question:

Write a program to verify whether one of the two entered integers falls within the range of 100 to 200 included.

Solution:

```c
#include<stdio.h>
int main() {
int x, y;
```

```c
printf("\nEnter the value for x: ");
scanf("%d", &x);
printf("\nEnter the value for y: ");
scanf("%d", &y);
if((x >= 100 && x <= 200) || (y >= 100 && y <= 200)) {
printf("True");
}
else {
printf("False");
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

char x[] = "Albert";

int i = 0;

while(x[i]) {

printf("%c at %p\n", x[i], &x[i]);

i++;

}

return 0;

}
```
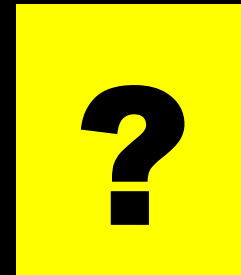
Question 68

Question:

Write a program to determine which of the two given integers is closest to the value 100.

If the two numbers are equal, return 0.

Solution:

```c
#include<stdio.h>
#include<stdlib.h>
int myfunc();
int main() {
printf("%d", myfunc(86, 99));
printf("\n%d", myfunc(55, 55));
```

```c
printf("\n%d", myfunc(65, 80));
return 0;
}
int myfunc(int a, int b) {
int x = abs(a - 100);
int y = abs(b - 100);
return x == y ? 0 : (x < y ? a : b);
}
```

Question 69

Question:

Write a program to determine whether a positive number entered by the user is a multiple of three or five, but not both.

Solution:

```c
#include<stdio.h>
int main() {
int x;
printf("\nEnter a number: ");
scanf("%d", &x);
if(x % 3 == 0 ^ x % 5 == 0) {
printf("True");
}
else {
printf("False");
}
```

```c
#include<stdio.h>
int main() {
printf("%-9s al\n", "alan");
printf("%-8s al\n", "alan");
printf("%-7s al\n", "alan");
printf("%-6s al\n", "alan");
printf("%-5s al\n", "alan");
printf("%-4s al\n", "alan");
return 0;
}
```

```
return 0;
}
```

Question 70

Question:

Write a program to determine whether two entered non-negative numbers have the same last digit.

Solution:

```c
#include<stdio.h>
#include<stdlib.h>
int main() {
int x, y;
printf("\nEnter the value for x: ");
scanf("%d", &x);
printf("\nEnter the value for y: ");
scanf("%d", &y);
if(abs(x % 10) == abs(y % 10)) {
printf("True");
}
else {
printf("False");
}
return 0;
}
```

```c
#include<stdio.h>
#include<ctype.h>
int main() {
char x;
x = getchar();
if(islower(x)) {
putchar(toupper(x));
}
else {
putchar(tolower(x));
}
return 0;
}
```

?

Question 71

Question:

Write a program to determine whether a given non-negative number is a multiple of 12 or it is one more than a multiple of 12.

Solution:

```c
#include<stdio.h>
#include<stdlib.h>
int main() {
int x = 43;
if(x % 12 == 0 || x % 12 == 1) {
printf("True");
}
else {
printf("False");
}
return 0;
}
```

```c
#include<stdio.h>
int main() {
printf(6 + "Albert Einstein");
return 0;
}
```

Output:

Einstein

Question 72

Question:

Write a program that accepts two integers and returns true when one of them equals 6, or when their sum or difference equals 6.

Solution:

```c
#include<stdio.h>
#include<stdlib.h>
int main() {
int x, y;
printf("\nEnter the value for x: ");
scanf("%d", &x);
printf("\nEnter the value for y: ");
scanf("%d", &y);
if(x == 6 || y == 6 || x + y == 6 || abs(x - y) == 6) {
printf("True");
}
else {
printf("False");
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

printf("%f\n", (float)(int)10.5 / 4);

return 0;

}
```

Output:

2.500000

Question 73

Question:

Write a program to check whether it is possible to add two integers to get the third integer from three entered integers.

Solution:

```c
#include<stdio.h>
int main() {
int x, y, z;
printf("\nEnter the value for x: ");
scanf("%d", &x);
printf("\nEnter the value for y: ");
scanf("%d", &y);
printf("\nEnter the value for z: ");
scanf("%d", &z);
if(x == y + z || y == x + z || z == x + y) {
printf("True");
}
else {
printf("False");
}
return 0;
}
```

```c
#include<stdio.h>

int myfunc();

int main() {

printf("\nAlbert Einstein");

myfunc();

return 0;

}

int myfunc() {

printf("\nElsa Einstein");

main();

}
```

?

Question 74

Question:

Write a program that converts kilometers per hour to miles per hour.

Solution:

```c
#include<stdio.h>
int main() {
float kmph;
printf("Enter kilometers per hour: ");
scanf("%f", &kmph);
printf("\n%f miles per hour", (kmph * 0.6213712));
return 0;
}
```

Question 75

Question:

Write a program to calculate area of an ellipse.

Solution:

```c
#include<stdio.h>
#define PI 3.141592
```

```c
#include<stdio.h>

int main() {

printf("Albert Einstein\n");

main();

return 0;

}
```

?

```c
int main() {
float major, minor;
printf("\nEnter length of major axis: ");
scanf("%f", &major);
printf("\nEnter length of minor axis: ");
scanf("%f", &minor);
printf("\nArea of an ellipse = %0.4f", (PI * major * minor));
return 0;
}
```

Question 76

Question:

Write a program to calculate the sum of three given integers. Return the third value if the first two values are equal.

Solution:

```c
#include<stdio.h>
int  myfunc();
int main() {
printf("\n%d", myfunc(11, 11, 11));
printf("\n%d", myfunc(11, 11, 16));
printf("\n%d", myfunc(18, 15, 10));
return 0;
}
int myfunc(int a, int b, int c) {
if (a == b && b == c) return 0;
```

Data structures are ways of organizing and storing data in a computer program or system, so that it can be accessed and used efficiently. They provide a way to manage and manipulate data in a more organized and efficient manner, which is important for many computer applications that deal with large amounts of data.

There are many different types of **data structures**, each with their own strengths and weaknesses. Some common examples of data structures include arrays, linked lists, stacks, queues, trees, graphs, and hash tables. Each of these data structures has different properties, such as the way they store and access data, the complexity of their operations, and their suitability for different types of applications.

Choosing the right **data structure** for a given problem is an important part of designing efficient algorithms and software systems. By selecting the appropriate data structure, developers can improve the performance, scalability, and maintainability of their applications, and provide better user experiences for their users

```c
if (a == b) return c;
if (a == c) return b;
if (b == c) return a;
else return a + b + c;
}
```

Question 77

Question:

Write a program to convert bytes to kilobytes.

Solution:

```c
#include<stdio.h>
int main() {
double bytes;
printf("\nEnter number of bytes: ");
scanf("%lf",&bytes);
printf("\nKilobytes: %.2lf", (bytes/1024));
return 0;
}
```

```c
#include<stdio.h>
#include<stdlib.h>
int main() {
int x = 1;
x++;
if(x <= 6) {
printf("\nC language");
exit(0);
main();
}
return 0;
}
```

Output:

C language

Question 78

Question:

Write a program to convert megabytes to kilobytes.

Solution:

```c
#include<stdio.h>
int main() {
double megabytes, kilobytes;
printf("\nInput the amount of megabytes to convert: ");
scanf("%lf",&megabytes);
kilobytes = megabytes * 1024;
printf("\nThere are %lf kilobytes in %lf megabytes.", kilobytes, megabytes);
return 0;
}
```

Question 79

Question:

Write a program to count the number of even elements in an integer array.

Solution:

```c
#include<stdio.h>
```

```c
int main() {
int array[1000], i, arr_size, even=0;
printf("Input the size of the array: ");
scanf("%d", &arr_size);
printf("Enter the elements in array: \n");
for(i=0; i<arr_size; i++) {
scanf("%d",&array[i]);
}

for(i=0; i<arr_size; i++) {
if(array[i]%2==0) {
    even++;
}
}
printf("Number of even elements: %d", even);
return 0;
}
```

Question 80

Question:

Write a program to count the number of odd elements in an integer array.

Solution:

```c
#include<stdio.h>
int main() {
```

```c
int array[1000], i, arr_size, odd=0;
printf("Input the size of the array: ");
scanf("%d", &arr_size);
printf("Enter the elements in array: \n");
for(i=0; i<arr_size; i++) {
scanf("%d",&array[i]);
}

for(i=0; i<arr_size; i++) {
if(array[i]%2!=0) {
    odd++;
}
}
printf("Number of odd elements: %d", odd);
return 0;
}
```

The ability of machines to carry out operations that ordinarily require human intelligence, such as speech recognition, decision-making, and language translation, is known as **artificial intelligence** (AI). **AI** involves the development of computer programs and algorithms that can learn from data and experience to make predictions, identify patterns, and solve problems. This includes techniques such as machine learning, deep learning, natural language processing, and robotics. **AI** is a rapidly evolving field that has the potential to transform many industries and aspects of everyday life.

Question 81

Question:

Write a program that will accept two integers and determine whether or not they are equal.

Solution:

```c
#include<stdio.h>
int main() {
int x, y;
```

```c
printf("Input the values for x and y: \n");

scanf("%d %d", &x, &y);

if(x == y) {

    printf("x and y are equal\n");

}

else {

    printf("x and y are not equal\n");

}

return 0;

}
```

Automated reasoning refers to the use of computer programs and algorithms to automatically derive logical conclusions from given premises. The goal of automated reasoning is to develop techniques that can automate the process of reasoning and problem-solving, enabling computers to perform tasks that would otherwise require human intelligence.

Automated reasoning systems typically use symbolic logic and formal reasoning methods to analyze the logical structure of a problem, identify potential solutions, and generate a proof or a refutation. These systems can be used in a variety of applications, such as software verification, theorem proving, expert systems, and artificial intelligence.

Automated reasoning is a critical component of many AI systems, particularly in domains that require advanced logical reasoning and decision-making, such as medicine, law, and finance. It is also an active area of research in computer science, mathematics, and philosophy.

Question 82

Question:

Write a program to find the third angle of a triangle if two angles are given.

Solution:

```c
#include<stdio.h>
int main() {
int angle1, angle2;
printf("\nEnter the first angle of the triangle: ");
scanf("%d", &angle1);
printf("\nEnter the second angle of the triangle: ");
scanf("%d", &angle2);
printf("\nThird angle of the triangle is: %d", (180 - (angle1 + angle2)));
return 0;

}
```

Question 83

Question:

Write a program to determine whether a particular year is a leap year or not.

Solution:

```c
#include<stdio.h>
int main() {
int year;
printf("Enter the year: ");
scanf("%d", &year);
if((year % 400) == 0) {
printf("\n%d is a leap year.", year);
}
else if((year % 100) == 0) {
printf("\n%d is a not leap year.", year);
}
else if((year % 4) == 0) {
printf("\n%d is a leap year.", year);
}
else {
printf("\n%d is not a leap year.", year);
}
return 0;
}
```

Computer vision is a field of artificial intelligence and computer science that focuses on enabling machines to interpret and understand visual information from the world around them. It involves the development of algorithms and techniques that allow computers to analyze, process, and interpret digital images and videos.

Computer vision encompasses a range of tasks, including object detection and recognition, facial recognition, image segmentation, scene reconstruction, and motion analysis. It is used in a variety of applications, such as autonomous vehicles, surveillance systems, medical imaging, and augmented reality.

The primary goal of **computer vision** is to enable machines to replicate and improve upon the abilities of human vision, allowing them to perceive, interpret, and understand the visual world in ways that were previously impossible. It is a rapidly evolving field that combines expertise from computer science, mathematics, physics, and neuroscience.
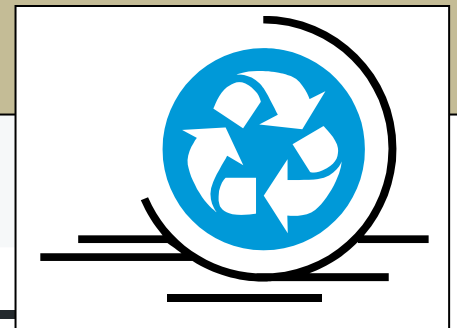
Question 84

Question:

Write a program that reads the candidate's age and determine a candidate's eligibility to cast his own vote.

Solution:

```c
#include<stdio.h>
int main() {
int age;
printf("\nEnter the age of the candidate: ");
scanf("%d",&age);
if(age<18) {
printf("\nWe apologize, but the candidate is not able to cast his vote.");
printf("\nAfter %d year, the candidate would be able to cast his vote.", (18-age));
}
else {
printf("Congratulation! the candidate is qualified to cast his vote.\n");
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

char *x = "Albert Einstein\n";

while(putchar(*x++));

return 0;

}
```

Output:

Albert Einstein

Question 85

Question:

Write a program to Convert Yard to Foot.

Solution:

```c
#include<stdio.h>
int main() {
float yard;
printf("\nEnter the Length in Yard : ");
scanf("%f", &yard);
printf("\n%f Yard in Foot is: %f", yard, (3*yard));
return 0;
}
```

Soft computing is a subfield of artificial intelligence that aims to mimic the human brain's ability to learn and make decisions in uncertain or incomplete environments. It is characterized by its ability to handle imprecise or uncertain information, such as fuzzy logic, neural networks, genetic algorithms, and other probabilistic and statistical techniques. The goal of soft computing is to develop computational techniques that can efficiently solve complex problems in various domains, including engineering, finance, healthcare, and others. The approach is particularly useful in situations where traditional rule-based methods are not effective or where there is a lack of complete information.

Question 86

Question:

Write a program to convert gigabytes to megabytes.

Solution:

```c
#include<stdio.h>
int main() {
```

```c
double gigabytes, megabytes;
printf("\nInput the amount of gigabytes to convert: ");
scanf("%lf", &gigabytes);
megabytes = gigabytes*1024;
printf("\nThere are %lf megabytes in %lf gigabytes.", megabytes, gigabytes);
return 0;
}
```

Question 87

Question:

Write a program to Convert Kilogram to Pounds.

Solution:

```c
#include<stdio.h>
int main() {
float kg, lbs;
printf("\nEnter Weight in Kilogram: ");
scanf("%f", &kg);
lbs = kg*2.20462;
printf("\n%f Kg = %f Pounds", kg, lbs);
return 0;
}
```

Question 88

Question:

Write a program to Convert Kilogram to Ounce.

Solution:

```c
#include<stdio.h>
int main() {
float kg, ounce;
printf("\nEnter Weight in Kilogram: ");
scanf("%f", &kg);
ounce = kg*35.274;
printf("\n%f Kg = %f Ounce", kg, ounce);
return 0;
}
```

Question 89

Question:

Write a program to Convert Pounds to Grams.

Solution:

Machine learning is a subfield of artificial intelligence that involves the development of algorithms and statistical models that enable computer systems to automatically improve their performance on a particular task or problem. Machine learning algorithms use training data to learn patterns and relationships in the data, which are then used to make predictions or decisions on new, unseen data.

Algorithms for **machine learning** can be classified into three categories: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the algorithm is trained with a labelled dataset in which the accurate output is given for each input. In unsupervised learning, the algorithm is trained on an unlabeled dataset, and it must discover patterns and structure in the data without any guidance. Reinforcement learning involves training an agent to make decisions based on feedback received from its environment.

Machine learning has applications in a wide range of fields, including computer vision, natural language processing, robotics, and finance, among others. Anomaly detection, regression, grouping, and classification are just a few of the tasks it is frequently used for.

```c
#include<stdio.h>
int main() {
float pound, gram;
printf("\nEnter Weight in Pounds: ");
scanf("%f", &pound);
gram = pound*453.592;
printf("\n%f Pound = %f Grams", pound, gram);
return 0;
}
```

Question 90

Question:

Write a program to verify whether a triangle is valid or not using angles.

Solution:

```c
#include<stdio.h>
int main() {
int angle1, angle2, angle3, sum;
printf("\nEnter the first angle of the triangle: ");
scanf("%d", &angle1);
printf("\nEnter the second angle of the triangle: ");
scanf("%d", &angle2);
printf("\nEnter the third angle of the triangle: ");
scanf("%d", &angle3);
sum = angle1 + angle2 + angle3;
if(sum == 180) {
```

```c
printf("\nThe triangle is valid.");
}
else {
printf("\nThe triangle is not valid.");
}
return 0;
}
```

Question 91

Question:

Write a program to add the digits of a two-digit number that is entered by the user.

Solution:

```c
#include<stdio.h>
int main() {
int x, y, sum = 0;
printf("\nEnter a two-digit number: ");
scanf("%d", &x);
y = x;
while(y != 0) {
sum = sum + y % 10;
y = y / 10;
}
printf("\nSum of digits of %d is: %d", x, sum);
return 0;
}
```

Question 92

Question:

Write a program to verify if a character you entered is a vowel or a consonant.

Solution:

```c
#include<stdio.h>
int main() {
char ch;
printf("\nEnter a character: ");
scanf("%c", &ch);
if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ||
 ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U' ) {

printf("\n%c is a vowel", ch);
}
else {
printf("\n%c is a consonant", ch);
}
return 0;
}
```

Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) and computer science that deals with the interaction between computers and human language. The goal of **NLP** is to enable computers to understand, interpret, and generate human language, both written and spoken.

NLP involves a range of techniques, including statistical and machine learning algorithms, pattern recognition, semantic analysis, and computational linguistics. It is used to build applications such as language translation, sentiment analysis, speech recognition, chatbots, and question-answering systems, among others.

NLP is a rapidly evolving field, with new techniques and applications being developed all the time. Its ultimate goal is to enable seamless communication between humans and computers, allowing us to interact with technology in more natural and intuitive ways.

Question 93

Question:

Write a program to find factorial of a number.

Solution:

```c
#include<stdio.h>
int main() {
int i, fact=1, num;
printf("\nEnter a number: ");
scanf("%d",&num);
for(i=1; i<=num; i++) {
    fact=fact*i;
}
printf("\nFactorial of %d is: %d", num, fact);
return 0;
}
```

```c
#include<stdio.h>
#define x 2
int main() {
int i;
for(i=24; i<28; i++) {
printf("%d %% %d = %d\n", i, x, i%x);
}
return 0;
}
```

Output:

24 % 2 = 0

25 % 2 = 1

26 % 2 = 0

27 % 2 = 1

Question 94

Question:

Write a program to print number of days in a month.

Solution:

```c
#include<stdio.h>
int main() {
int x[12]={31,28,31,30,31,30,31,31,30,31,30,31}, m;
printf("\nEnter the month number: ");
scanf("%d",&m);
if(m>12 || m<1) {
printf("Invalid input");
}
else if(m==2) {
printf("\nNumber of days in month 2 is either 29 or 28");
}
else {
printf("\nNumber of days in month %d is %d", m, x[m-1]);
}
return 0;
}
```

Question 95

Question:

Write a program to concatenate two strings.

Solution:

```c
#include<stdio.h>
#include<string.h>
int main() {
char a[1000], b[1000];
```

```c
#include<stdio.h>
int main() {
char *names[] = {
"Albert",
"Alan",
"John",
"James",
"Mary"
};
int i;
for(i=0;i<5;i++)
puts(*(names+i));
return(0);
}
```

Output:

Albert

Alan

John

James

Mary

```c
printf("\nEnter the first string: ");
scanf("%s", a);
printf("\nEnter the second string: ");
scanf("%s", b);
strcat(a, b);
printf("\nString produced by concatenation is: %s", a);
return 0;
}
```

Question 96

Question:

Write a program to find maximum between two numbers.

Solution:

```c
#include<stdio.h>
int main() {
int a,b;
printf("Enter two numbers: \n");
scanf("%d%d", &a, &b);
if(a>b) {
printf("\n%d is a maximum number", a);
}
else {
printf("\n%d is a maximum number", b);
}
```

```c
#include<stdio.h>
int main() {
for(int x=2; x<=25; x=x+2) {
printf("%d\t", x);
}
putchar('\n');
return 0;
}
```

?

```
return 0;
}
```

Question 97

Question:

Write a program to compare two strings.

Solution:

```c
#include<stdio.h>
#include<string.h>
int main() {
char a[100], b[100];
printf("Enter the first string: \n");
scanf("%s", a);
printf("Enter the second string: \n");
scanf("%s", b);
if (strcmp(a,b) == 0) {
printf("The 2 strings are equal.\n");
}
else {
printf("The 2 strings are not equal.\n");
}
return 0;
}
```

```c
#include<stdio.h>
int main() {
for(int x=3; x>=1; x--) {
for(int y=1; y<=x; y++) {
printf("%d ", y);
}
printf("*");
}
return 0;
}
```

Output:

1 2 3 *1 2 *1 *

```c
#include<stdio.h>
int main() {
puts("Albert Einstein");
return 0;
}
```

Output:

Albert Einstein

Question 98

Question:

Write a program to convert the upper case letter to lower case letter.

***

Solution:

```c
#include<ctype.h>
#include<stdio.h>
int main() {
char ch;
ch = 'G';
printf("%c in lowercase is represented as %c", ch, tolower(ch));
return 0;
}
```

Question 99

Question:

Write a program to find the quotient and remainder of a entered dividend and divisor.

***

Solution:

```c
#include<stdio.h>
```

```c
int main() {
int dividend, divisor;
printf("\nEnter dividend: ");
scanf("%d", &dividend);
printf("\nEnter divisor: ");
scanf("%d", &divisor);
printf("\nQuotient = %d\n", (dividend / divisor));
printf("\nRemainder = %d", (dividend % divisor));
return 0;
}
```

Question 100

Question:

Write a program to determine the Size of int, float, double and char.

Solution:

```c
#include<stdio.h>
int main() {
printf("Size of char is: %ld byte\n",sizeof(char));
printf("Size of int is: %ld bytes\n",sizeof(int));
printf("Size of float is: %ld bytes\n",sizeof(float));
printf("Size of double is: %ld bytes", sizeof(double));
return 0;
}
```

Question 101

Question:

Write a program to verify the password until it is correct.

Solution:

```c
#include<stdio.h>
int main() {
int pwd, i;
while (i!=0) {
printf("\nEnter the password: ");
scanf("%d",&pwd);
if(pwd==1988) {
printf("The password you entered is correct");
i=0;
}
else {
printf("Incorrect password, try again");
}
printf("\n");
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

int i;

for(i=-3; i<3; i++)

printf("%d ", i);

for(; i>=-3; i--)

printf("%d ", i);

putchar('\n');

return 0;

}
```

Output:

-3 -2 -1 0 1 2 3 2 1 0 -1 -2 -3

```c
#include<stdio.h>

int main() {

int x, y;

printf("\nEnter the value for x: ");

scanf("%d", &x);

printf("\nEnter the value for y: ");

scanf("%d", &y);

(x>y)? printf("\nx is greater"): printf("\ny is greater");

return 0;

}
```

Output: **?**

413

Question 102

Question:

Write a program to find absolute value of a number.

Solution:

```c
#include<stdio.h>
#include<stdlib.h>
int main() {
int num;
printf("Input a positive or negative number: \n");
scanf("%d", &num);
printf("\nAbsolute value of |%d| is %d\n", num, abs(num));
return 0;
}
```

Question 103

Question:

Write a program that will accept a person's height in cm and classify the person based on it.

Solution:

```c
#include<stdio.h>
int main() {
float ht;
printf("\nEnter the height (in cm): ");
scanf("%f", &ht);
if(ht < 150.0) {
printf("Dwarf.\n");
}
else if((ht >= 150.0) && (ht < 165.0)) {
printf("Average Height.\n");
}
else if((ht >= 165.0) && (ht <= 195.0)) {
printf("Taller.\n");
}
else {
printf("Abnormal height.\n");
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

int i = 6;

while(i==1)

i = i-3;

printf("%d\n", i);

}
```

Because the condition is false, the value of "i" will be printed instead of the while loop being executed.

Output:

6

Question 104

Question:

Write a program to calculate the area of different geometric shapes using switch statements.

Solution:

```c
#include<stdio.h>
int main() {
int choice;
float r, l, w, b, h;
printf("\nEnter 1 for area of circle: ");
printf("\nEnter 2 for area of rectangle: ");
printf("\nEnter 3 for area of triangle: ");
printf("\nEnter your choice : ");
scanf("%d", &choice);

switch(choice) {
case 1:
printf("Enter the radius of the circle: ");
scanf("%f", &r);
printf("\nArea of a circle is: %f", (3.14*r*r));
break;
case 2:
printf("Enter the length and width of the rectangle: \n");
scanf("%f%f", &l, &w);
printf("\nArea of a rectangle is: %f", (l*w));
break;
case 3:
printf("Enter the base and height of the triangle: \n");
scanf("%f%f", &b, &h);
printf("\nArea of a triangle is: %f", (0.5*b*h));
break;
default:
printf("\nPlease enter a number from 1 to 3.");
break;
}
return 0;
}
```

Robotics is a branch of engineering and computer science that deals with the design, construction, operation, and use of robots. Robots are programmable machines that can carry out tasks autonomously, or with minimal human intervention.

Robotics involves the integration of various engineering disciplines, including mechanical engineering, electrical engineering, and computer science. Robotics applications can be found in a wide range of industries, from manufacturing and agriculture to healthcare and space exploration.

Robots can be classified based on their functionality, such as industrial robots used for manufacturing, service robots used in healthcare and hospitality, and military robots used in defense and surveillance. They can also be classified based on their mobility, such as wheeled, legged, aerial, or underwater robots.

The **field of robotics** is constantly evolving, with advancements in artificial intelligence, machine learning, and materials science enabling the development of more capable and sophisticated robots. Robotics has the potential to revolutionize various industries and improve the quality of life for people by automating tedious or dangerous tasks and enabling new forms of human-robot collaboration.

Question 105

Question:

Write a program to accept a character from the keyboard and print "Yes" if it is equal to y. Otherwise print "No".

Solution:

```c
#include<stdio.h>
int main() {
char ch;
printf ("Enter a character: ");
ch = getchar ();
if(ch == 'y' || ch == 'Y') {
printf ("Yes\n");
}
else {
printf ("No\n");
}
return(0);
}
```

```c
#include<stdio.h>

int main() {

int num[6] = {21, 22, 23, 24, 25, 26};

for(int i = 0; i <= 7; i++) {

printf("\n%d", num[i]);

}

return 0;

}
```

Output:

21

22

23

24

25

26

-759135232

-1723617269

The garbage values will be printed after i = 5

Question 106

Question:

Write a program that uses bitwise operators to multiply an entered value by four.

Solution:

```
#include<stdio.h>
int main() {
long x, y;
printf("Enter a integer: ");
scanf("%ld", &x);
y = x;
x = x << 2;
printf("%ld x 4 = %ld\n", y, x);
return 0;
}
```

```
#include<stdio.h>
int main() {
for(int x=5; x>=1; x--) {
for(int y=1; y<=x; y++) {
printf("* ");
}
printf("\n");
}
return 0;
}
```

Output:

```
* * * * *
 * * * *
  * * *
   * *
    *
```

Question 107

Question:

Write a program to check whether a number entered by the user is power of 2 or not.

Solution:

```c
#include<stdio.h>
int main() {
int x;
printf("Enter a number: ");
scanf("%d", &x);
if((x != 0) && ((x &(x - 1)) == 0)) {
printf("\n%d is a power of 2", x);
}
else {
printf("\n%d is not a power of 2", x);
}
return 0;
}
```

```c
#include<stdio.h>
#define A 16
#define B 4
int main() {
printf("A+B: %d\n", A+B);
printf("A-B: %d\n", A-B);
printf("A×B: %d\n", A*B);
printf("A/B: %d\n", A/B);
return 0;
}
```

Output:

A+B: 20

A-B: 12

A×B: 64

A/B: 4

Question 108

Question:

Write a program to determine whether a triangle is scalene, isosceles, or equilateral.

Solution:

```c
#include<stdio.h>
int main() {
int side1, side2, side3;
printf("\nEnter the first side of the triangle: ");
scanf("%d",&side1);
printf("\nEnter the second side of the triangle: ");
```

```c
scanf("%d",&side2);
printf("\nEnter the third side of the triangle: ");
scanf("%d",&side3);
if(side1 == side2 && side2 == side3) {
printf("\nThe given Triangle is equilateral.");
}
else if(side1 == side2 || side2 == side3 || side3 == side1) {
printf("\nThe given Triangle is isosceles.");
}
else {
printf("\nThe given Triangle is scalene.");
}
return 0;
}
```

Question 109

Question:

Write a program to print ASCII values of all the letters of the English alphabet from A to Z.

Solution:

```c
#include<stdio.h>
int main() {
int i;
for(i='A'; i<='Z'; i++) {
```

```
printf("ASCII value of %c = %d\n", i, i);

}

return 0;

}
```

Question 110

Question:

Write a program to find sum of even numbers between 1 to n.

Solution:

```c
#include<stdio.h>
int main() {
int i, num, sum=0;
printf("Enter a number: ");
scanf("%d", &num);
for(i=2; i<=num; i=i+2) {
sum = sum + i;
}
printf("\nSum of all even number between 1 to %d is: %d", num, sum);
return 0;
}
```

Question 111

Question:

Write a program to find sum of odd numbers between 1 to n.

Solution:

```c
#include<stdio.h>
int main() {
int i, num, sum=0;
printf("Enter a number: ");
scanf("%d",  &num);
for(i=1; i<=num; i=i+2) {
sum = sum + i;
}
printf("\nSum of all odd number between 1 to %d is: %d", num, sum);
return 0;
}
```

Question 112

Question:

Write a program to find maximum number using switch case.

Solution:

```c
#include<stdio.h>
int main() {
int x, y;
printf("Enter any two numbers: \n");
scanf("%d%d", &x, &y);
switch(x > y) {
case 0: printf("%d is Maximum number", y);
break;
case 1: printf("%d is Maximum number", x);
break;
}
return 0;
}
```

```c
#include<stdio.h>
int main() {
for(int i=5; i>=0; i=i-1) {
printf("%d\n", i);
}
return 0;
}
```

Output:

5

4

3

2

1

0

Question 113

Question:

Write a program that allows you to enter the cost price and the selling price of a product and calculate profit or loss.

Solution:

```c
#include<stdio.h>
int main() {
int cp, sp;
printf("\nInput Cost Price: ");
scanf("%d", &cp);
```

```c
printf("\nInput Selling Price: ");
scanf("%d", &sp);
if(sp > cp) {
printf("Profit = %d", (sp - cp));
}
else if(cp > sp) {
printf("Loss = %d", (cp - sp));
}
else {
printf("No Profit No Loss.");
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

int x[25], a;

for(a = 0; a <= 24; a++);

{

x[a] = a;

printf("\n%d", x[a]);

}

return 0;

}
```

Output:

25

Question 114

Question:

Write a program that display the pattern like a right angle triangle using an asterisk.

Solution:

```c
#include<stdio.h>
int main() {
int rows;
printf("Input the number of rows: ");
scanf("%d", &rows);
for(int x=1; x<=rows; x++) {
```

```
for(int y=1; y<=x; y++)
printf("*");
printf("\n");
}
return 0;
}
```

Question 115

Question:

Write a program that display the pattern like a right angle triangle using a number.

Solution:

```
#include<stdio.h>
int main() {
int rows;
printf("Input the number of rows: ");
scanf("%d",&rows);
for(int x=1; x<=rows; x++) {
for(int y=1; y<=x; y++)
printf("%d", y);
printf("\n");
}
return 0;
}
```

```
#include<stdio.h>
static int b;
int main() {
static int c;
printf("%d %d", b, c);
return 0;
}
```

Output:

0 0

Question 116

Question:

Write a program to determine the number and sum of all integers between 50 and 100 which are divisible by 2.

Solution:

```c
#include<stdio.h>
int main() {
int x, sum=0;
printf("Numbers between 50 and 100, divisible by 2: \n");
for(x=51; x<100; x++) {
if(x%2==0) {
printf("%5d", x);
sum+=x;
}
}
printf("\nThe sum: %d", sum);
return 0;
}
```

```c
#include<stdio.h>

int main() {

int z[] = { 1, 40, 1, 80, 6 };

int y, *x;

x = z;

for(y = 0; y <= 4; y++) {

printf("\n%d", *x);

x++;

}

return 0;

}
```

Output:

1

40

1

80

6

Question 117

Question:

Write a program that uses the function to determine whether a entered number is even or odd.

Solution:

```c
#include<stdio.h>
int myfunc(int x) {
return (x & 1);
}
int main() {
int x;
printf("Enter any number: ");
scanf("%d", &x);
if(myfunc(x)) {
printf("\nThe number you entered is odd.");
}
else {
printf("\nThe number you entered is even.");
}
return 0;
}
```

```c
#include<stdio.h>
int main() {
printf("Enter a character: ");
int c = getchar();
printf("You have entered the character '%c'.\n", c);
return 0;
}
```

?

```c
#include<stdio.h>
#include<math.h>
int main() {
printf("%f\n", log(50.0));
return 0;
}
```

Output:

3.912023

Question 118

Question:

Write a program to find square root of an entered number.

Solution:

```c
#include<stdio.h>
#include<math.h>
int main() {
int x;
printf("Enter any number: ");
scanf("%d",&x);
printf("Square root of %d is %.2lf", x, sqrt(x));
return 0;
}
```

Question 119

Question:

Write a program to find power of a entered number using library function.

Solution:

```c
#include<stdio.h>
```

```c
#include<math.h>

int main() {
int x, y;
printf("\nEnter the value for x: ");
scanf("%d", &x);
printf("\nEnter the value for y: ");
scanf("%d", &y);
printf("\n%d^%d = %ld", x, y, (long)pow(x,y));
return 0;
}
```

Evolutionary computing is a subfield of artificial intelligence and computational intelligence that uses evolutionary algorithms to solve complex optimization problems. It is inspired by the process of natural selection and evolution in biological organisms, and it aims to mimic these processes in order to find optimal solutions to a wide range of problems.

Evolutionary computing typically involves the use of genetic algorithms, which are a type of optimization algorithm that uses principles of natural selection and genetics to evolve a population of potential solutions to a problem. The algorithm generates a population of candidate solutions, which are then evaluated and selected for reproduction based on their fitness or quality. Through a process of mutation, crossover, and selection, the algorithm evolves the population towards better solutions over successive generations.

Evolutionary computing has applications in various domains, including engineering, finance, biology, and artificial life. It can be used to solve problems such as optimization, scheduling, design, and control, among others. The approach is particularly useful in situations where traditional optimization techniques are not effective, or when there are complex, multi-objective optimization problems to be solved.

Question 120

Question:

Write a program to determine if the character entered is an alphabetic or numeric character.

Solution:

```c
#include<stdio.h>
#include<ctype.h>
int main() {
char ch;
printf("Enter a character: ");
scanf("%c", &ch);
if(isdigit(ch)) {
printf("\n%c is a Digit", ch);
}
```

```c
#include<stdio.h>

int main() {

printf("\"Hi,\" Albert, \"Elsa!\"\n");

return 0;

}
```

Output:

"Hi," Albert, "Elsa!"

```c
else if(isalpha(ch)) {
printf("\n%c is an Alphabet", ch);
}
else {
printf("\n%c is not an Alphabet, or a Digit", ch);
}
return 0;
}
```

Question 121

Question:

Write a program to determine whether the character entered is an alphanumeric character or not.

Solution:

```c
#include<stdio.h>
#include<ctype.h>
int main() {
char a;
printf("Enter a character: ");
scanf("%c", &a);
if(isalnum(a)) {
printf("\n%c is an alphanumeric character.", a);
}
else {
```

```c
#include<stdio.h>
int main() {
printf("The Sum is: %f\n",16.0+17);
return 0;
}
```

Output:

The Sum is: 33.000000

```
printf("\n%c is NOT an alphanumeric character.", a);
}
return 0;
}
```

Question 122

Question:

Write a program to determine whether the character entered is an punctuation character or not.

Solution:

```
#include<stdio.h>
#include<ctype.h>
int main() {
char a;
printf("Enter a character: ");
scanf("%c", &a);
if(ispunct(a)) {
printf("\n%c is an punctuation character.", a);
}
else {
printf("\n%c is NOT an punctuation character.", a);
}
return 0;
}
```

```
#include<stdio.h>
#define SIZE 6
int main() {
char name[SIZE];
printf("Enter your name: ");
fgets(name, SIZE, stdin);
printf("Pleased to meet you, %s.", name);
return 0;
}
```

431

Question 123

Question:

Write a program to check whether the entered character is a graphic character or not.

Solution:

```c
#include<stdio.h>
#include<ctype.h>
int main() {
char a;
printf("Enter a character: ");
scanf("%c", &a);
if(isgraph(a)) {
printf("\n%c is a graphic character.", a);
}
else {
printf("\n%c is NOT a graphic character.", a);
}
return 0;
}
```

```c
#include<stdio.h>

int main() {

static int x[8];

for(int i = 0; i <= 7; i++)

printf("\n%d", x[i]);

return 0;

}
```

Output:

0

0

0

0

0

0

0

0

0

Question 124

Question:

Write a program to list all printable characters using isprint() function.

Solution:

```
#include<stdio.h>
#include<ctype.h>
int main() {
int i;
for(i = 1; i <= 127; i++)
if(isprint(i)!= 0)
printf("%c ", i);
return 0;
}
```

```
#include<stdio.h>

int main() {

float i = 6.1 ;

switch (i) {

    case 0.1 :

        printf("\nAlan"); break;

    case 2.1 :

        printf("\nJohn"); break;

    case 4.1 :

        printf("\nMary"); break;

    case 6.1 :

        printf("\nAlbert");

    }

    return 0;

}
```

In the switch condition, "i" is not an integer. We cannot test floats in switch statements.

Error

Question 125

Question:

Write a program to check whether the entered character is a hexadecimal digit character or not.

Solution:

```c
#include<stdio.h>
#include<ctype.h>
int main() {
char a;
printf("Enter a character: ");
scanf("%c", &a);
if(isxdigit(a)) {
printf("\n%c is a hexadecimal digit character.", a);
}
else {
printf("\n%c is NOT a hexadecimal digit character.", a);
}
return 0;
}
```

Question 126

Question:

Write a program to print ASCII value of all control characters.

Solution:

```c
#include<stdio.h>
#include<ctype.h>
int main() {
int i;
printf("The ASCII value of all control characters are: \n");
```

```
for(i=0; i<=127; i++) {
if(iscntrl(i)!=0)
printf("\n %d ", i);
}
return 0;
}
```

```
#include<stdio.h>

int main()  {

if(printf("Albert Einstein")){}

return 0;

}
```

Output:

Albert Einstein

Question 127

Question:

Write a program to check whether the entered character is a white-space character or not.

Solution:

```
#include<stdio.h>
#include<ctype.h>
int main() {
char c;
printf("Enter a character: ");
scanf("%c", &c);
if(isspace(c) == 0) {
printf("Not a white-space character.");
}
else {
printf("White-space character.");
}
return 0;
```

```
#include<stdio.h>

typedef int var;

var main() {

var i = 6;

printf("%d + %d = %d\n", i, i, i+i);

return 0;

}
```

Output:

6 + 6 = 12

```
}
```

Question 128

Question:

Write a program to illustrate isprint() and iscntrl() functions.

Solution:

```c
#include<stdio.h>
#include<ctype.h>
int main() {
char ch = 'a';
if(isprint(ch)) {
printf("\n%c is printable character.", ch);
}
else {
printf("\n%c is not printable character.", ch);
}

if(iscntrl(ch)) {
printf("\n%c is control character.", ch);
}
else {
printf("\n%c is not control character.", ch);
}
return (0);
```

```
}
```

Question 129

Question:

Write a program to calculate surface area of cube.

```c
#include<stdio.h>

int main() {
char name[10];
printf("Enter your name: ");
fgets(name,10,stdin);
printf("Pleased to meet you, %s.\n",name);
return 0;
}
```

Solution:

```c
#include<stdio.h>
int main() {
int side;
long area;
printf("\nEnter the side of cube: ");
scanf("%d", &side);
area = 6*side*side;
printf("\nThe surface area of cube is: %ld", area);
return 0;
}
```

Question 130

Question:

Write a program to subtract 2 numbers without using subtraction operator.

Solution:

```c
#include<stdio.h>
#include<stdlib.h>
int main() {
int x =6, y=3;
printf("%d", x+(~y)+1);
return 0;
}
```

```c
#include<stdio.h>
int main() {
for(int i=12; i<=15; i=i+1) {
printf("%d\t", i);
}
putchar('\n');
return 0;
}
```

Output:

```
12      13      14      15
```

Question 131

Question:

Write a program to add 2 numbers without using addition operator.

Solution:

```c
#include<stdio.h>
#include<stdlib.h>
int main() {
int x =6, y=3;
printf("%d", x-(~y)-1);
return 0;
}
```

```c
#include<stdio.h>
int main() {
int i;
for(i=0; i<5; i=i+1, printf("%d\n", i));
return(0);
}
```

Output:
```
1

2

3

4

5
```

Question 132

Question:

Write a program to multiply a number by 2 without using multiplication operator.

Solution:

```c
#include<stdio.h>
int main() {
int x=2;
printf("%d", x<<1);
return 0;
}
```

```c
#include<stdio.h>
int main() {
printf("%d\n", 49);
printf("%1.2f\n",3.15698222);
printf("%d\n", 496596);
printf("%1.1f\n",0.00056);
return 0;
}
```

Output:

49

3.16

496596

0.0

Question 134

Question:

Write a program to divide a number by 2 without using division operator.

Solution:

```c
#include<stdio.h>
int main() {
int x=12;
printf("%d", x>>1);
```

```
return 0;
}
```

Question 135

Question:

Write a program to calculate volume of sphere.

Solution:

```c
#include<stdio.h>
int main() {
int radius;
float PI = 3.141592;
printf("\nEnter the radius of sphere: ");
scanf("%d", &radius);
float volume = (4/3)*(PI*radius*radius*radius);
printf("\nThe volume of sphere is: %f", volume);
return 0;
}
```

```c
#include<stdio.h>
int main() {
printf("Enter a character: ");
int c = getc(stdin);
printf("You have entered the character '%c'.\n",c);
return 0;
}
```

Question 136

Question:

Write a program to calculate volume of ellipsoid.

Solution:

```c
#include<stdio.h>
int main() {
int r1, r2, r3;
float PI = 3.141592;
printf("\nEnter the radius of the ellipsoid of axis 1: ");
scanf("%d", &r1);
printf("\nEnter the radius of the ellipsoid of axis 2: ");
scanf("%d", &r2);
printf("\nEnter the radius of the ellipsoid of axis 3: ");
scanf("%d", &r3);
float volume = (4/3)*(PI*r1*r2*r3);
printf("\nThe volume of ellipsoid is: %f", volume);
return 0;
}
```

Question 137

Question:

Write a program that uses a for loop to determine power of a number entered by the user.

Solution:

```c
#include<stdio.h>
```

```c
int main() {
int x, y;
long power = 1;
printf("\nEnter the value for x: ");
scanf("%d", &x);
printf("\nEnter the value for y: ");
scanf("%d", &y);
for(int i=1; i<=y; i++) {
power = power * x;
}
printf("%d ^ %d = %ld", x, y, power);
return 0;
}
```

```c
#include<stdio.h>
int main() {
int x, y;
x = 14;
y = x + 3;
if( x < y) {
printf("%d is greater than %d\n", x, y);
}
return(0);
}
```

Output:

14 is greater than 17

Question 138

Question:

Write a program to read three numbers and find average of numbers.

Solution:

```c
#include<stdio.h>
int main() {
int a,b,c;
float avg;
printf("\nEnter the first number: ");
scanf("%d", &a);
printf("\nEnter the second number: ");
```

```c
scanf("%d",&b);

printf("\nEnter the third number: ");

scanf("%d",&c);

avg=(a+b+c)/3.0;

printf("\nAverage of three numbers is: %f", avg);

return 0;

}
```

Question 139

Question:

Write a program to read integer "n" and print first three powers (n^1, n^2, n^3).

Solution:

```c
#include<stdio.h>

#include<math.h>

int main() {

int n;

printf("\nEnter a number: ");

scanf("%d",&n);

printf("%f, %f, %f", pow(n, 1), pow(n, 2), pow(n, 3));

return 0;

}
```

Cryptography is the practice of securing communication from third-party intrusion or eavesdropping by converting plain text into an encoded form that is unintelligible without a secret key. **Cryptography** involves using mathematical algorithms and protocols to transform messages into a secret code that can only be read by someone who has the key to unlock it. The purpose of cryptography is to ensure the confidentiality, integrity, and authenticity of information and to prevent unauthorized access, modification, or interception of data. **Cryptography** is used extensively in computer security to protect sensitive information such as passwords, credit card numbers, and other personal or financial data. It is also used in various communication protocols such as email, instant messaging, and virtual private networks to ensure secure transmission of information over public networks.

static memory allocation	dynamic memory allocation
allocation of memory done at compilation time and it stays the same throughout the entire run of the program	allocation of memory done at the time of running the program and it increases or decreases throughout the entire run of the program and it is released or freed when not required or used

C Program:

```
#include<stdio.h>

int main()
{
    FILE *fp;
    fp = fopen("myfiles.txt","w");
    return 0;
}
```

create a file named **myfiles.txt**

The **w** means that the file is being opened for writing – and if the file does not exist then the new file will be created.

C Program:

```
#include<stdio.h>
int main()
{
    FILE *fp;
    fp = fopen("myfiles.txt","w");
    fprintf(fp, "%s", "C Programming");
    return 0;
}
```

The **fprintf** function writes the text **C Programming** to the file **myfiles.txt**.

```cpp
#include<bits/stdc++.h>

using namespace std;

int main() {

int a = 15, b = 25;

printf("Value of a: %d", a);

printf("\n Value of b: %d", b);

swap(a, b);

printf("\n After swapping, the values are: a = %d, b = %d", a, b);

return 0;

}
```

Output:

Value of a: 15

Value of b: 25

After swapping, the values are: a = 25, b = 15

Computer architecture refers to the design of computer systems, including their organization, components, and how they operate together to perform tasks. It encompasses the hardware and software components of a computer system and their interactions. **Computer architecture** defines the structure and behavior of a computer system, including the processing units, memory, input or output devices, and communication protocols between the components.

The **architecture of a computer system** is critical in determining its performance, power consumption, and overall efficiency. It is also important for determining the compatibility and interoperability of different hardware and software components. Computer architecture plays a crucial role in the design and development of computer systems, including desktops, laptops, servers, and mobile devices. Understanding computer architecture is essential for computer engineers, software developers, and system administrators who are involved in designing, developing, and maintaining computer systems.

C++ Exercises
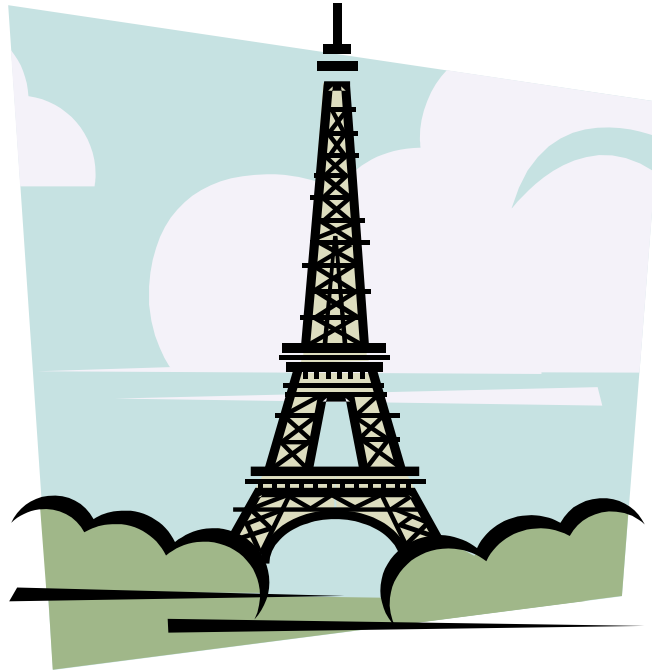
In the 1970s, **Bjarne Stroustrup**, a Danish computer scientist, created the C++ programming language. The initial name of C++ was "C with classes." By practicing C++ programs, you can learn the C++ programming language most effectively. An all-purpose programming language is C++. It offers facilities for low-level memory manipulation together with imperative, object-oriented, and generic programming features. It is used to create machine learning tools, web browsers, video games, and operating systems. Examples on fundamental C++ ideas can be found on this chapter. It is encouraged that you use the programs as references and test the concepts on your own. Exercises in C++ are a great way to practise programming, develop your abilities, and learn more about the language.

Question 1

Question:

Write a program to print Hello, World!.

Solution:

```cpp
#include<iostream>
int main() {
std::cout<<"Hello, World!";
return 0;
}
```
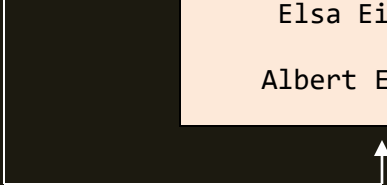
```cpp
#include<iostream>
using namespace std;
int main() {
cout << "Elsa Einstein \n";
cout << "Albert Einstein" << endl;
return 0;
}
```

Output:

```
    Elsa Einstein
    Albert Einstein
```

Question 2

Question:

Write a program to compute the perimeter and area of a rectangle.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
```

```cpp
int height = 8;
int width = 5;
int perimeter = 2*(height + width);
cout<<"Perimeter of the rectangle is: " << perimeter << " cm\n";
int area = height * width;
cout<<"Area of the rectangle is: "<< area << " square cm\n";
return 0;
}
```

Question 3

Question:

Write a program to compute the perimeter and area of a circle.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int radius = 4;
float perimeter = 2*3.14*radius;
cout<<"Perimeter of the circle is: " << perimeter << " cm\n";
float area = 3.14*radius*radius;
cout<<"Area of the circle is: "<< area << " square cm\n";
return 0;
}
```

Question 4

Question:

Write a program that accepts two numbers from the user and calculate the sum of the two numbers.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
float a, b, sum;
cout<<"\nEnter the first number: ";
cin>>a;
cout<<"\nEnter the second number: ";
cin>>b;
sum = a+ b;
cout<<"\nSum of the above two numbers is: "<< sum;
return 0;
}
```

```cpp
#include<iostream>
using namespace std;
int main() {
int x= 26; // Now x is 15
x = 56;    // Now x is 10
cout << x;
// Output: 56
return 0;
}
```

Question

Question:

Write a program that accepts two numbers from the user and calculate the product of the two numbers.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int a, b, mult;
cout<<"\nEnter the first number: ";
cin>>a;
cout<<"\nEnter the second number: ";
cin>>b;
mult = a * b;
cout<<"\nProduct of the above two numbers is: " << mult;
return 0;
}
```

A **relational database** is a type of database management system **(DBMS)** that stores and organizes data in tables with rows and columns. In a relational database, data is stored in multiple related tables, and the relationships between these tables are defined by their shared columns or keys. The most common type of relational database is a Structured Query Language **(SQL)** database, which allows users to manipulate and query the data stored within it using SQL commands.

Relational databases are widely used in business and other applications to manage large amounts of structured data, such as customer information, sales records, and inventory data. They offer several advantages over other types of databases, including their ability to support complex relationships between data, their scalability, and their ability to ensure data consistency and accuracy through the use of constraints and transaction processing.

Question 6

Question:

Write a program that accepts three numbers and find the largest of three.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x, y, z;
cout<<"\nEnter the first number: ";
cin>>x;
cout<<"\nEnter the second number: ";
cin>>y;
cout<<"\nEnter the third number: ";
cin>>z;

// if x is greater than both y and z, x is the largest
if (x >= y && x >= z)
cout<<x<<" is the largest number.";

// if y is greater than both x and z, y is the largest
if (y >= x && y >= z)
cout<<y<<" is the largest number.";

// if z is greater than both x and y, z is the largest
if (z >= x && z >= y)
cout<<z<<" is the largest number.";

return 0;
}
```

Structured storage refers to a type of file system that is designed to store and manage structured data in a hierarchical and organized manner. It is commonly used for storing and managing complex data types such as multimedia files, large text documents, and database files.

Structured storage systems typically use a hierarchical directory structure to organize data into folders and subfolders, with each folder containing files that are related to a particular aspect of the data being stored. These files can be of different sizes, types, and formats, and are typically stored as binary data. One of the key features of structured storage is that it allows applications to access data within files without having to read or write the entire file. This is accomplished by using a mechanism called "streaming", which allows applications to access specific sections of a file without loading the entire file into memory.

Structured storage is commonly used in enterprise applications, such as document management systems, content management systems, and multimedia applications. It is also used in relational database systems, where large amounts of structured data are stored in a hierarchical manner for efficient retrieval and manipulation.

Question

Question:

Write a program that reads three floating values and check if it is possible to make a triangle with them. Also calculate the perimeter of the triangle if the entered values are valid.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
float x, y, z;
cout<<"\nEnter the first number: ";
cin>>x;
cout<<"\nEnter the second number: ";
cin>>y;
cout<<"\nEnter the third number: ";
cin>>z;

if(x < (y+z) && y < (x+z) && z < (y+x)) {
cout<<"\nPerimeter of the triangle is: " << x+y+z;
}
else {
cout<<"\nIt is impossible to form a triangle.";
}

return 0;
}
```

Data mining is the process of discovering hidden patterns, correlations, and trends within large sets of data using statistical and computational methods. It involves extracting meaningful insights from complex data sets by using automated or semi-automated techniques to identify patterns and relationships that may not be immediately apparent.

Data mining typically involves several steps, including data cleaning, data integration, data selection, data transformation, data mining, pattern evaluation, and knowledge representation. The process is often iterative, with the results of each iteration informing subsequent iterations and refining the final output.

Data mining is used in a wide range of applications, including marketing and advertising, healthcare, financial analysis, fraud detection, and scientific research. It is particularly useful in situations where there is a large amount of data available, and where the relationships between the data are complex and difficult to discern using traditional analytical techniques.

The insights gained from data mining can be used to make better business decisions, improve customer service, reduce costs, and identify new opportunities for growth and innovation. However, it is important to use data mining ethically, and to ensure that the privacy and security of individuals' data are protected.

Question 8

Question:

Write a program that reads an integer between 1 and 7 and print the day of the week in English.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int day;
cout<<"\nEnter a number between 1 to 7 to get the day name: ";
cin>>day;
switch(day) {
case 1 : cout<<"Monday\n"; break;
case 2 : cout<<"Tuesday\n"; break;
case 3 : cout<<"Wednesday\n"; break;
case 4 : cout<<"Thursday\n"; break;
case 5 : cout<<"Friday\n"; break;
case 6 : cout<<"Saturday\n"; break;
case 7 : cout<<"Sunday\n"; break;
default : cout<<"Enter a number between 1 to 7.";
}
return 0;
}
```

Compiler theory refers to the study of how programming languages are translated into machine-readable code by a compiler. It encompasses a range of topics, including syntax analysis, semantic analysis, code generation, optimization, and debugging. There are various steps involved in the compilation of a program. First, the source code of the program is analyzed to check for syntax errors and to create a syntax tree that represents the structure of the program. Next, semantic analysis is performed to check for semantic errors and to determine the meaning of the program's constructs. After this, the code is generated, which involves translating the high-level code into low-level code that can be executed by the computer. Finally, the generated code is optimized to improve its performance.

Compiler theory is an important area of study for computer science and programming, as it is essential for developing efficient and effective software. It also plays a critical role in the development of new programming languages, as it provides insights into the design and implementation of language constructs and the trade-offs between expressiveness, ease of use, and performance. Some of the key topics within compiler theory include parsing algorithms, language design and syntax, type systems, memory management, optimization techniques, and debugging and error handling. Advances in compiler theory have led to significant improvements in the performance and reliability of software, and have helped to make programming more accessible and easier to learn for beginners.

Question 9

Question:

Write a program to find the sum of two numbers.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int a, b, sum;
a=1;
b=2;
sum = a + b;
cout<<"The sum of a and b is: " << sum;
return 0;
}
```

```cpp
#include<iostream>
using namespace std;
int main() {
bool x = true;
bool y = false;
cout << x <<endl;
// Output: 1
cout << y<<endl;
// Output: 0
return 0;
}
```

Question 10

Question:

Write a program to find the square of a number.

Solution:

```cpp
#include<iostream>
#include<cmath>
using namespace std;
int main() {
int a, b;
a=2;
b = pow((a), 2);
cout<<"The square of a is: "<< b;
return 0;
}
```

```cpp
#include<iostream>
#include<cstring>
using namespace std;
int main() {
string x = "Albert";
cout << x[0];
// Output: A
return 0;
}
```

Question 11

Question:

Write a program to find the greatest of two numbers.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int a, b;
a = 2;
b = 3;
if(a>b) {
cout<<"a is greater than b";
}
else {
```

```cpp
cout<<"b is greater than a";
}
return 0;
}
```

Question 12

Question:

Write a program to print the average of the elements in the array.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int i, avg, sum = 0;
int num [5] = {16, 18, 20, 25, 36};
for(i=0; i<5; i++) {
sum = sum + num [i];
avg = sum/5;
}
cout<<"\nSum of the Elements in the array is: "<< sum;
cout<<"\nAverage of the elements in the array is: " << avg;
return 0;
}
```

Question 13

Question:

Write a program that prints all even numbers between 1 and 25.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
cout<<"Even numbers between 1 to 25:\n";
for(int i = 1; i <= 25; i++) {
if(i%2 == 0) {
cout<< i << endl;
}
}
return 0;
}
```

```cpp
#include<iostream>
#include<cstring>
using namespace std;
int main() {
string x = "Albert";
x[0] = 'E';
cout << x;
// Output: Elbert
return 0;
}
```

Question 14

Question:

Write a program that prints all odd numbers between 1 and 50.

Solution:

```cpp
#include <iostream>

using namespace std;

int main() {

cout<<"Odd numbers between 1 to 50:\n";

for(int i = 1; i <= 50; i++) {

if(i%2 != 0) {

cout<<i<<endl;

}

}

return 0;

}
```

Question 15

Question:

Write a program to print the first 10 numbers starting from one together with their squares and cubes.

Solution:

```cpp
#include<iostream>

using namespace std;

int main() {

for(int i=1; i<=10; i++) {

cout<<"Number = " << i << " its square = " << i*i << " its cube = " << i*i*i

<<endl;

}

return 0;
```

```
}
```

Question 16

Question:

Write a program:

If you enter a character M

Output must be: ch = M.

```cpp
#include<iostream>
using namespace std;
int main() {
cout << min(15, 60)<<endl;
// Output: 15
cout << max(15, 60)<<endl;
// Output: 60
return 0;
}
```

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
char M;
cout<<"Enter any character: ";
cin>>M;
cout<<"ch = "<< M;
return 0;
}
```

Question 17

Question:

Write a program to print the multiplication table of a number entered by the user.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int n, i;
cout<<"Enter any number: ";
cin>>n;
for( i=1; i<=5; i++)
cout<< n <<" * "<< i <<" = "<< n*i <<endl;
return 0;
}
```

Computability theory is a branch of computer science and mathematics that deals with the study of what can be computed and how effectively it can be computed. It investigates the limitations of computing machines, the scope of what can be computed, and the extent to which problems can be solved algorithmically. One of the main concepts in **computability theory** is the notion of computable functions. A function is said to be computable if there exists an algorithm that can compute its values for any input. The study of computable functions leads to the concept of **Turing machines**, which are abstract models of computation that can perform any computation that can be done by any algorithm. Another important concept in computability theory is the halting problem. The **halting problem** asks whether it is possible to determine, for a given input and program, whether the program will halt or run indefinitely. The answer to this problem is no, and it has important implications for the limits of computability.

Computability theory also investigates the notion of complexity, which is concerned with the resources required to solve a given problem. This includes the time required to compute a solution, as well as the amount of memory or other resources required. **Complexity theory** is closely related to computability theory, as it explores the limits of what can be efficiently computed. Overall, computability theory is a fundamental area of computer science and mathematics, with important applications in areas such as cryptography, artificial intelligence, and the analysis of algorithms.

Question 18

Question:

Write a program to print the product of the first 10 digits.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int i, product = 1;
for(i=1; i<=10; i++) {
product = product * i;
```

```
}
cout<<"The product of the first 10 digits is: " << product;
return 0;
}
```

Question 19

Question:

Write a program to print whether the given number is positive or negative.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int a;
a = -35;
if(a>0) {
cout<<"Number is positive";
}
else {
cout<<"Number is negative";
}
return 0;
}
```

```cpp
#include<iostream>
#include<cmath>
using namespace std;
int main() {
cout << sqrt(36) << endl;
// Output: 6
cout << round(6.68) << endl;
// Output: 7
cout << log(4) << endl;
// Output: 1.38629
return 0;
}
```

461

Question 20

Question:

Write a program to check the equivalence of two numbers entered by the user.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x, y;
cout<<"Enter the first number: ";
cin>>x;
cout<<"Enter the second number: ";
cin>>y;
if(x-y==0) {
cout<<"The two numbers are equivalent";
}
else {
cout<<"The two numbers are not equivalent";
}
return 0;
}
```

```cpp
#include<iostream>
using namespace std;
int main() {
cout << (20 > 19);
// Output: 1
return 0;
}
```

Question 21

Question:

Write a program to print the remainder of two numbers entered by the user.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int a, b, c;
cout<<"Enter the first number: ";
cin>>a;
cout<<"Enter the second number: ";
cin>>b;
c = a % b;
cout<<"The remainder of " << a << " and " << b << " = " << c;
return 0;
}
```

Question 22

Question:

Write a program to print the characters from A to Z.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
char i;
for(i='A'; i<='Z'; i++) {
cout << i << endl;
}
return 0;
}
```

```cpp
#include<iostream>
#include<cstring>
using namespace std;


void myfunc(string name, int age) {
cout << name << " John. " << age << " years old. \n";
}


int main() {
myfunc("Albert", 73);
myfunc("Elsa", 14);
myfunc("David", 30);
return 0;

}
```

Question 23

Question:

Write a program to print the length of the entered string.

Solution:

```cpp
#include<iostream>
#include<string.h>
using namespace std;
int main() {
char str[1000];
cout<<"Enter a string to calculate its length: ";
cin>>str;
cout<<"The length of the entered string is: "<< strlen(str);
return 0;
```

```
}
```

Question 24

Question:

Write a program to check whether the given character is a lower case letter or not.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
char ch = 'a';
if(islower(ch))
cout<<"The given character is a lower case letter";
else
cout<<"The given character is a upper case letter";
return 0;
}
```

Question 25

Question:

Write a program to check whether the given character is a upper case letter or not.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
char ch = 'A';
if(isupper(ch))
cout<<"The given character is a upper case letter";
else
cout<<"The given character is a lower case letter";
return 0;
}
```

Question 26

Question:

Write a program to convert the lower case letter to upper case letter.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
char ch = 'a';
char b = toupper(ch);
```

```
cout<<"Lower case letter "<<ch<<" is converted to Upper case letter "<<b;
return 0;
}
```

Question 27

Question:

Write a program that takes a distance in centimeters and outputs the corresponding value in inches.

Solution:

```cpp
#include<iostream>
using namespace std;
#define x 2.54
int main() {
double inch, cm;
cout<<"Enter the distance in cm: ";
cin>>cm;
inch = cm / x;
cout<<"\nDistance of "<< cm << " cms is equal to " << inch << " inches";
return 0;
}
```

Question 28

Question:

Write a program to print the output:

Einstein [0] = E

Einstein [1] = I

Einstein [2] = N

Einstein [3] = S

Einstein [4] = T

Einstein [5] = E

Einstein [6] = I

Einstein [7] = N

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int i;
char name [8] = {'E' , 'I', 'N', 'S', 'T', 'E', 'I', 'N'};
for(i=0; i<8; i++)
cout<<"Einstein ["<< i <<" ] = "<< name[i] << endl;
return 0;
}
```

Procedural programming is a programming paradigm that is based on the concept of procedures, also known as subroutines or functions. A procedure is a group of instructions that performs a specific task, and can be called or executed from different parts of a program. In procedural programming, a program is composed of one or more procedures, which can be organized into modules or libraries. The procedures are typically organized in a step-by-step manner, with each procedure calling other procedures as needed to perform its task.

The primary focus of procedural programming is on the procedures themselves, and how they interact with each other and with the data in a program. Data in a procedural program is typically organized into data structures, such as arrays, records, or linked lists. One of the main advantages of procedural programming is its simplicity and ease of use. It is often used for small to medium-sized programs, and is particularly well-suited for applications that involve simple data processing, such as mathematical calculations or file manipulation.

However, procedural programming also has some limitations. As programs become larger and more complex, it can become difficult to manage the interactions between procedures and the data in a program. Additionally, procedural programming is not well-suited for applications that require complex data structures or that involve user interfaces. Overall, procedural programming remains an important programming paradigm, and is still widely used today in a variety of applications, particularly in the development of system software and scientific computing.

Question 29

Question:

Write a program to print "Hello World" 10 times.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
for(int i=1; i<=10; i++) {
cout<< "Hello World"<< endl;
}
return 0;
}
```

Question 30

Question:

Write a program to print first 5 numbers using do while loop statement.

Solution:

```cpp
#include<iostream>
using namespace std;
```

```cpp
int main() {
int i  =1;
do {
cout<<" \ni = "<< i++;
} while(i<=5);
return 0;
}
```

Question 31

Question:

Write a program to check whether a character is an alphabet or not.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int a = 2;
if(isalpha(a)) {
cout<<"The character a is an alphabet";
}
else {
cout<<"The character a is not an alphabet";
}
return 0;
}
```

Question 32

Question:

Write a program to check whether a entered number is even or odd.

Solution:

```cpp
#include<iostream>

using namespace std;

int main() {

int a;

cout<<"Enter any number: ";

cin>>a;

if(a%2 == 0) {

cout<<"The entered number is even";

}

else {

cout<<"The entered number is odd";

}

return 0;

}
```

Programming language pragmatics refers to the study of how programming languages are used in practice, including their implementation, performance, and design. It focuses on the practical aspects of programming languages, such as how they are used to solve real-world problems, and how they interact with the hardware and operating system on which they are run. **Programming language pragmatics** is concerned with a range of issues related to the use of programming languages, including:

- **Language syntax and semantics:** How programming languages are structured and how they express different kinds of computations.
- **Language implementation:** How programming languages are implemented, including issues related to compilers, interpreters, and runtime environments.
- **Language performance:** How programming languages perform in terms of efficiency, memory usage, and other metrics.
- **Language design:** How programming languages are designed to meet the needs of different kinds of applications and users.
- **Language evolution:** How programming languages evolve over time, including the introduction of new features and the retirement of old ones.
- **Language usability:** How programming languages are designed to be easy to use and understand, and how they can be made more accessible to novice programmers.

Programming language pragmatics is an important area of study for software developers, as it provides insights into how different programming languages can be used to solve different kinds of problems, and how they can be optimized for performance and ease of use. It is also important for **researchers and educators**, as it helps to inform the design and development of new programming languages and programming curricula.

Question 33

Question:

Write a program to print the ASCII value of the entered character.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
char c;
cout << "Enter a character: ";
cin >> c;
cout << "The ASCII Value of " << c << " is " << int(c);
return 0;
}
```

Question 34

Question:

Write a program that will print all numbers between 1 to 50 which divided by a specified number and the remainder will be 2.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x, i;
cout<<"Enter a number: ";
cin>>x;
for(i=1; i<=50; i++) {
    if((i%x)==2) {
    cout<<i<<endl;
      }
}
return 0;
}
```

Programming language theory is the study of the fundamental concepts and principles that underlie the design and implementation of programming languages. It is concerned with understanding how programming languages work, how they are structured, and how they can be used to express different kinds of computations. **Programming language theory** encompasses a wide range of topics, including:

- **Syntax and semantics:** How programming languages are structured and how they express different kinds of computations.
- **Types and type systems:** How programming languages deal with data types, and how type systems can be used to ensure program correctness and safety.
- **Formal methods:** How mathematical techniques can be used to formally reason about the behavior of programs written in different programming languages.
- **Concurrency and parallelism:** How programming languages can be designed to handle concurrent and parallel execution, including issues related to synchronization, communication, and resource management.
- **Logic programming:** How programming languages based on mathematical logic can be used to express and reason about complex systems.
- **Functional programming:** How programming languages based on the principles of functional programming can be used to express complex computations in a concise and elegant way.

Programming language theory is an important area of study for computer science researchers, as it provides a theoretical foundation for the design and implementation of programming languages. It also plays an important role in the development of new programming languages and the advancement of programming practices, including the development of new programming paradigms and techniques.

Question 35

Question:

Write a program to determine whether two numbers in a pair are in ascending or descending order.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int a, b;
cout<<"\nEnter a pair of numbers (for example 22,12 | 12,22): ";
cout<<"\nEnter the first number: ";
```

```
cin>>a;
cout<<"\nEnter the second number: ";
cin>>b;
if (a>b) {
cout<<"\nThe two numbers in a pair are in descending order.";
}
else {
cout<<"\nThe two numbers in a pair are in ascending order.";
}
return 0;
}
```

Question 36

Question:

Write a program that reads two numbers and divides one by the other. Specify "Division not possible" if that is not possible.

Solution:

```
#include<iostream>
using namespace std;
int main() {
int a, b;
float c;
cout<<"\nEnter the first number: ";
cin>>a;
cout<<"\nEnter the second number: ";
```

```
cin>>b;
if(b != 0) {
    c = (float)a/(float)b;
    cout<<a<<"/"<<b<<" = "<< c;
}
else {
    cout<<"\nDivision not possible.\n";
}
return 0;
}
```

Question 37

Question:

Write a program that will print all numbers between 1 to 50 which divided by a specified number and the remainder is equal to 2 or 3.

Solution:

```
#include<iostream>
using namespace std;
int main() {
int x, i;
cout<<"Enter a number: ";
cin>>x;
for(i=1; i<=50; i++) {
   if((i%x)==2 || (i%x) == 3) {
    cout<<i<<endl;
```

```
    }
}
return 0;
}
```

Question 38

Question:

Write a program that adds up all numbers between 1 and 100 that are not divisible by 12.

Solution:

```
#include<iostream>
using namespace std;
int main() {
int x =12, i, sum = 0;
for(i=1; i<=100; i++) {
   if((i%x)!= 0) {
    sum += i;
    }
}
cout<<"\nSum: "<<sum;
return 0;
}
```

Software engineering is the systematic process of designing, developing, testing, and maintaining software. It is a discipline that applies engineering principles to the creation of software products and systems, with the goal of developing high-quality, reliable, and efficient software that meets the needs of users and stakeholders.

Software engineering involves a range of activities, including requirements analysis, software design, coding, testing, and maintenance. It also includes project management and quality assurance activities, such as software configuration management, software metrics, and software process improvement.

The **software engineering process** typically involves several phases, such as planning, requirements gathering, design, implementation, testing, and deployment. Each of these phases is essential to ensuring that the software is developed to meet the needs of users and stakeholders, is of high quality, and can be maintained over time.

Software engineering is a complex and interdisciplinary field, with many sub-disciplines, such as software architecture, software testing, software maintenance, and software project management. It requires expertise in areas such as computer science, mathematics, and engineering, as well as an understanding of business and user needs.

Question 39

Question:

Write a program to calculate the value of x where x = 1 + 1/2 + 1/3 + ... + 1/50.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
float x = 0;
for(int i=1; i<=50; i++) {
    x += (float)1/i;
}
cout<<"Value of x: "<< x;
return 0;
}
```

Algorithm design is the process of creating a set of instructions, or steps, to solve a particular problem or accomplish a specific task. **Algorithms** are essential in computer science and programming because they provide a way to automate processes and make them more efficient. The following are some general steps for designing an algorithm:

- **Understand the problem:** Before designing an algorithm, it is essential to have a clear understanding of the problem you are trying to solve. Analyze the problem, break it down into smaller sub-problems, and identify the input and output requirements.
- **Plan the algorithm:** Once you have a clear understanding of the problem, plan the algorithm by determining the steps required to solve the problem. Identify the data structures and programming constructs that will be required, such as loops, conditionals, and functions.
- **Code the algorithm:** Write the code for the algorithm using the programming language of your choice. Be sure to use clear and concise code that is easy to understand and debug.
- **Test the algorithm:** Test the algorithm with different inputs to ensure that it works as expected. Debug any errors that are identified during testing.
- **Optimize the algorithm:** Once the algorithm is working correctly, optimize it to improve its efficiency. This may involve reducing the number of operations required, using more efficient data structures, or optimizing the code for the specific hardware and software environment.
- **Document the algorithm:** Finally, document the algorithm, including its purpose, inputs, outputs, and any limitations or assumptions made during the design process. This documentation will help others understand and use the algorithm in the future.

Designing an **algorithm** can be a complex process, and it often requires a deep understanding of the problem domain and the programming language being used. However, with practice and experience, anyone can become proficient in algorithm design and develop efficient and effective solutions to complex problems.

Question 40

Question:

Write a program that reads a number and find all its divisor.

Solution:

```
#include<iostream>

using namespace std;

int main() {

int x, i;

cout<<"\nEnter a number: ";

cin>>x;

cout<<"All the divisor of "<<x<<" are: \n";

for(i = 1; i <= x; i++) {

    if((x%i) == 0) {

    cout<<i<<endl;

    }

}

return 0;

}
```

Automata theory is a branch of computer science that deals with the study of abstract machines or computational models that can perform various operations on strings or symbols. These machines are used to model and describe the behavior of systems that exhibit complex behavior, such as natural language processing, computer languages, and artificial intelligence.

Automata theory includes the study of various types of machines or models, such as finite automata, pushdown automata, Turing machines, and more. Each of these models has different computational power and can perform different types of operations on strings or symbols.

The theory behind automata is to design computational models that can process and recognize patterns in strings or symbols, and make decisions based on the input. The different models of automata are used to study the different levels of complexity in computation, and to design algorithms that can efficiently solve complex problems.

Automata theory has many practical applications, including in compiler design, natural language processing, artificial intelligence, and database systems. By understanding the computational power and limitations of different automata models, computer scientists can design efficient and effective algorithms for solving complex problems in various fields.

Question 41

Question:

Write a program to find the incremented and decremented values of two numbers.

Solution:

```
#include<iostream>

using namespace std;

int main() {

int a, b, c, d, e, f;

a = 10;

b=12;
```

```
c=a+1;
d=b+1;
e=a-1;
f=b-1;
cout<<"The incremented value of a = "<< c << endl;
cout<<"The incremented value of b = "<< d << endl;
cout<<"The decremented value of a = "<< e << endl;
cout<<"The decremented value of b = "<< f << endl;
return 0;
}
```

Question 42

Question:

Write a program to find square of a entered number using functions.

Solution:

```cpp
#include<iostream>
using namespace std;
int square();
int main() {
int answer;
answer = square();
cout<<"The square of the entered number is: "<< answer;
return 0;
}
int square() {
```

```cpp
int x;
cout<<"Enter any number: ";
cin>>x;
return x*x;
}
```

Question 43

Question:

Write a program that accepts principal amount, rate of interest, time and compute the simple interest.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int P,T, R, SI;
cout<<"Enter the principal amount: ";
cin>>P;
cout<<"Enter the time: ";
cin>>T;
cout<<"Enter the rate of interest: ";
cin>>R;
SI = P*T*R/100;
cout<<"The simple interest is: "<<SI;
return 0;
}
```

Question 44

Question:

Write a program that swaps two numbers without using third variable.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int a, b;
cout<<"\nEnter the value for a: ";
cin>>a;
cout<<"\nEnter the value for b: ";
cin>>b;
cout<<"\nBefore swapping: " <<a <<" "<<b;
a=a+b;
b=a-b;
a=a-b;
cout<<"\nAfter swapping: " <<a<<" "<<b;
return 0;
}
```

Git is a distributed version control system that is open-source, free, and made to manage projects of all sizes quickly and effectively. It was created by **Linus Torvalds** in 2005 for development of the **Linux kernel**, and since then has become one of the most widely used version control systems in the software development industry. Git allows multiple users to work on the same project simultaneously, while keeping track of changes made to the codebase and providing features for managing and merging these changes. **Git** is used not only for software development but also for managing changes to any type of file, such as documentation, configuration files, and even images.

Question 45

Question:

Write a program to find the greatest of two entered numbers using pointers.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x, y, *p, *q;
cout<<"Enter the value for x: ";
cin>> x;
cout<<"Enter the value for y: ";
cin>> y;
p = &x;
q = &y;
if(*p>*q) {
cout<<"x is greater than y";
}
else {
cout<<"y is greater than x";
}
return 0;
}
```

Docker is a popular open-source platform for building, deploying, and managing applications in containers. A container is a lightweight and standalone executable package that includes everything needed to run an application, such as code, libraries, and dependencies. **Docker** allows developers to package their applications and dependencies into a container image, which can be easily shared and run across different environments, such as development, testing, and production. **Docker containers** provide a consistent environment for applications, reducing compatibility issues and making it easier to deploy and scale applications. **Docker** is widely used in software development and deployment, particularly in modern cloud-based and microservices architectures. With **Docker, developers** can build, test, and deploy applications quickly and efficiently, while operations teams can manage and scale these applications with ease.

Question 46

Question:

***Write a program to print the output:

body [b] = b

body [o] = o

body [d] = d

body [y] = y

***

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
char i;
char body [4] = {'b', 'o', 'd', 'y'};
for(i=0; i<4; i++)
cout<<"\n body ["<<body[i] <<" ] = "<< body[i] << endl;
return 0;
}
```

Question 47

Question:

Write a program to calculate the discounted price and the total price after discount

Given:

If purchase value is greater than 1000, 10% discount

If purchase value is greater than 5000, 20% discount

If purchase value is greater than 10000, 30% discount.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
double PV;
cout<<"Enter purchased value: ";
cin>>PV;
if(PV>1000) {
cout<<"Discount = "<< PV * 0.1 << endl;
cout<<"Total= "<< PV - PV * 0.1 << endl;
}
else if(PV>5000) {
cout<<"Discount = "<< PV * 0.2 << endl;
cout<<"Total= "<< PV - PV * 0.2 << endl;
}
else {
cout<<"Discount = "<< PV * 0.3 << endl;
cout<<"Total= "<< PV - PV * 0.3 << endl;
}
```

Bootstrap is a free and open-source front-end web development framework that is widely used to create responsive and mobile-first websites and web applications. It was developed by **Twitter** and is now maintained by the **Bootstrap Core Team**.

Bootstrap provides a set of CSS and JavaScript components that can be easily incorporated into web projects. These components include navigation bars, forms, buttons, icons, and more. The framework also offers a grid system that allows developers to create flexible and responsive layouts for their web pages.

One of the key benefits of **Bootstrap** is its focus on mobile-first design. The framework includes CSS classes that are specifically designed for smaller screens, making it easy to create websites that work well on both desktop and mobile devices.

Bootstrap is highly customizable and can be easily modified to match the look and feel of a specific website or brand. It also offers a range of third-party themes and plugins that can be used to extend the functionality of the framework.

Overall, **Bootstrap** is a popular and powerful front-end web development framework that provides a range of CSS and JavaScript components for creating responsive and mobile-first websites and web applications. Its focus on mobile-first design, flexible grid system, and customization options make it a popular choice among web developers.

```
return 0;
}
```

Question 48

Question:

Write a program to print the first ten natural numbers using while loop statement.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int i = 1;
while(i<=10) {
cout<<"\n" << i++;
}
return 0;
}
```

```cpp
#include<iostream>
#include<cstring>
using namespace std;
void myfunc(string x) {
cout << x << " Einstein\n";
}

int main() {
myfunc("David");
myfunc("Elsa");
myfunc("John");
return 0;
}
```
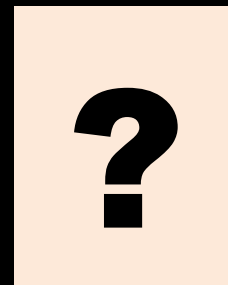
?

Question 49

Question:

Write a program to shift inputted data by two bits to the left.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x;
cout<<"Enter the integer from keyboard: ";
cin>>x;
cout<<"\nEntered value: "<< x;
cout<<"\nThe left shifted data is: " << (x<<=2);
return 0;
}
```

Question 50

Question:

Write a program to shift inputted data by two bits to the Right.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x;
cout<<"Enter the integer from keyboard: ";
cin>>x;
```

```
cout<<"\nEntered value: "<< x;
cout<<"\nThe right shifted data is: " << (x>>=2);
return 0;
}
```

Question 51

Question:

Write a program to calculate the exact difference between x and 21. Return three times the absolute difference if x is greater than 21.

Solution:

```
#include<iostream>
using namespace std;
int main() {
int x;
cout<<"Enter the value for x: ";
cin>>x;
if(x<=21){
    cout<<abs(x-21);
 }
else if(x>=21) {
    cout<<abs(x-21)*3;
}
return 0;
}
```

```
#include<iostream>
using namespace std;


void myfunc() {
cout << "Einstein"<<endl;
}


int main() {
myfunc();
myfunc();
myfunc();
return 0;

}
```

```
# Output:
    Einstein
    Einstein
    Einstein
```

Question 52

Question:

Write a program that reads in two numbers and determine whether the first number is a multiple of the second number.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x, y;
cout<<"\nEnter the first number: ";
cin>>x;
cout<<"\nEnter the second number: ";
cin>>y;
if(x % y == 0) {
cout<<x<<" is a multiple of " <<y;
}
else {
cout<<x<<" is not a multiple of " <<y;
}
return 0;
}
```

Computational complexity theory is a branch of computer science and mathematics that focuses on classifying computational problems according to their level of difficulty, and studying the resources required to solve them. It aims to understand the limitations and capabilities of algorithms and computing systems in solving problems efficiently.

One of the main concepts in **computational complexity theory** is the notion of a computational problem. A computational problem is a task that can be performed by a computer, such as sorting a list of numbers or finding the shortest path between two points. Problems are typically defined by a set of inputs and a desired output.

The difficulty of a **computational problem** is measured in terms of its computational complexity. This includes the amount of time required to solve the problem, the amount of memory needed, and the number of computational steps required. Computational complexity is often divided into different classes, such as polynomial time, exponential time, or non-deterministic polynomial time.

One of the most famous problems in computational complexity theory is the **P vs. NP problem**. This problem asks whether every problem that can be verified by a polynomial-time algorithm can also be solved by a polynomial-time algorithm. The answer to this problem is not yet known, and it is considered one of the most important open problems in computer science.

Another important concept in **computational complexity theory** is the notion of hardness. A problem is said to be hard if it is at least as difficult as the hardest problems in a particular complexity class. This includes problems that are difficult to approximate or require non-deterministic algorithms to solve.

Overall, **computational complexity theory** is a vital area of computer science that seeks to understand the limits and possibilities of computation. It has important applications in areas such as cryptography, optimization, and algorithm design.

Question 53

Question:

Write a program to print the output:

Name of the book = B

Price of the book = 135.00

Number of pages = 300

Edition of the book = 8

using structures.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
struct book {
char name;
float price;
int pages;
int edition;
};
struct book b1= {'B', 135.00, 300, 8};
cout<<"Name of the book = "<< b1.name<< endl;
cout<<"Price of the book = "<< b1.price<<endl;
cout<<"Number of pages = "<< b1.pages<<endl;
cout<<"Edition of the book = "<< b1.edition<< endl;
return 0;
}
```

```cpp
#include<iostream>

using namespace std;


int myfunc(int x) {

   return 15 + x;

}


int main() {

cout << myfunc(13);

return 0;

}
```

489

Question 54

Question:

Write a program to convert Celsius into Fahrenheit.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
float fahrenheit, celsius;
celsius = 36;
fahrenheit = ((celsius*9)/5)+32;
cout<<"\nTemperature in fahrenheit is: "<<fahrenheit;
return 0;
}
```

Question 55

Question:

Write a program that will examine two inputted integers and return true if either of them is 50 or if their sum is 50.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x, y;
cout<<"\nEnter the value for x: ";
cin>>x;
cout<<"\nEnter the value for y: ";
cin>>y;
if(x == 50 || y == 50 || (x + y == 50)) {
    cout<<"\nTrue";
}
else {
    cout<<"\nFalse";
}
return 0;
}
```

Question 56

Question:

Write a program that counts the even, odd, positive, and negative values among eighteen integer inputs.

Solution:

```cpp
#include<iostream>
```

```cpp
using namespace std;
int main () {
int x, even = 0, odd = 0, positive = 0, negative = 0;
cout<<"\nPlease enter 18 numbers: \n";
for(int i = 0; i < 18; i++) {
cin>>x;
if (x > 0) {
    positive++;
}
if(x < 0) {
    negative++;
}
if(x % 2 == 0) {
    even++;
}
if(x % 2 != 0) {
    odd++;
}
}
cout<<"\nNumber of even values: "<<even;
cout<<"\nNumber of odd values: "<<odd;
cout<<"\nNumber of positive values: "<<positive;
cout<<"\nNumber of negative values: "<<negative;
return 0;
}
```


SQL (Structured Query Language) is a programming language designed to manage and manipulate data stored in relational databases. It provides a standardized way to interact with databases, allowing users to create, modify, and retrieve data stored in them. SQL is used to create and manage tables, define relationships between them, insert, update and delete data, and perform queries to retrieve specific information from one or more tables. SQL is widely used in various industries, including finance, healthcare, e-commerce, and government. It is a powerful and flexible language that is relatively easy to learn and can be used with various database management systems (DBMS) such as MySQL, Oracle, and Microsoft SQL Server.

Question 57

Question:

Write a program to check whether the person is a senior citizen or not.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int age;
cout<<"Enter age: ";
cin>>age;
if(age>=60) {
cout<<"Senior citizen";
}
else {
cout<<"Not a senior citizen";
}
return 0;
}
```

Visual Studio Code, commonly known as VS Code, is a free source-code editor developed by Microsoft. It is compatible with Linux, macOS, and Windows. **VS Code** is designed to provide developers with a lightweight and customizable tool for coding, debugging, and version control. It supports a wide range of programming languages and has a vast collection of extensions that can be installed to enhance its functionality. **VS Code** includes features such as syntax highlighting, code completion, debugging, source control integration, and task automation. It also offers an integrated terminal, allowing developers to run commands and scripts directly within the editor. **VS Code** has gained popularity among developers due to its versatility, speed, and ease of use, making it a popular choice for coding projects of all sizes and types.

Question 58

Question:

Write a program that reads a student's three subject scores (0-100) and computes the average of those scores.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
float score, total_score = 0;
int subject = 0;
cout<<"Enter three subject scores (0-100):\n";
while (subject != 3) {
cin>>score;
if(score < 0 || score > 100) {
cout<<"Please enter a valid score.\n";
}
else {
total_score += score;
subject++;
  }
}
cout<<"Average score = "<< (total_score/3);
return 0;
}
```

Google Cloud Platform, also known as **GCP**, is a cloud computing platform and service provided by Google. It provides a wide range of services and solutions, including infrastructure as a service (**IaaS**), platform as a service (**PaaS**), and software as a service (**SaaS**). GCP enables developers to build, deploy, and manage applications and services on Google's global network of data centers.

GCP provides a wide range of services and solutions, including virtual machines, container services, serverless computing, data storage, database services, analytics, and artificial intelligence. It also offers a variety of developer tools and services, including Google Cloud SDK, Google Cloud Build, and Google Cloud Functions.

GCP is designed to be flexible and scalable, making it suitable for businesses of all sizes. It provides various pricing options, including pay-as-you-go, subscription-based, and free plans, allowing businesses to choose the best pricing plan that meets their needs and budget.

GCP also offers a wide range of security features and compliance certifications, ensuring that businesses can securely and compliantly run their applications and services on the platform.

Overall, **GCP** is a powerful and flexible cloud computing platform that provides a wide range of services and solutions for building, deploying, and managing applications and services. It is widely used by businesses of all sizes and industries, from startups to large enterprises, and is considered one of the leading cloud computing platforms in the market.

Question 59

Question:

What results would the following programs produce?

```cpp
#include<iostream>
using namespace std;
int main() {
for(int i=1; i<=5; i++) {
if(i==3) {
break;
}
cout<<"\n"<< i;
}
return 0;
}
```

Solution:

```
1
2
```

```cpp
#include<iostream>
using namespace std;
int main() {
for(int i=1;i<=5;i++) {
if(i==3) {
goto HAI;
}
cout<<"\n "<<i;
}
HAI : cout<<"\n Linux";
}
```

JIRA is a software development tool developed by Atlassian. It is a project management tool that is designed to help teams plan, track, and manage their software development projects. **JIRA** is primarily used for agile software development, including Scrum, Kanban, and other agile methodologies. It provides teams with a centralized platform to track issues, bugs, and project progress, enabling better collaboration and communication between team members.

JIRA offers various features such as customizable workflows, agile boards, customizable dashboards, and reporting. It also integrates with other Atlassian tools, such as Confluence, Bitbucket, and Bamboo, to provide end-to-end software development solutions. **JIRA** can be used by software development teams of all sizes and is popular among both small and large organizations. It is highly configurable and can be customized to meet the specific needs of a development team. Overall, **JIRA** is a powerful and flexible tool that can help software development teams manage their projects efficiently and effectively.

Solution:

```
1
2
Linux
```

```cpp
#include<iostream>
using namespace std;
int main() {
for( ; ; ) {
cout<<"This loop will run forever.\n";
}
return 0;
}
```

Solution:

```
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever. .........
```

Azure, also known as **Microsoft Azure**, is a cloud computing platform and service provided by Microsoft. It provides a wide range of services and solutions, including **infrastructure as a service** (IaaS), **platform as a service (PaaS)**, and **software as a service** (SaaS). **Azure** enables developers to build, deploy, and manage applications and services on Microsoft's global network of data centers.

Azure provides a wide range of services and solutions, including virtual machines, container services, serverless computing, data storage, database services, analytics, and artificial intelligence. It also offers a variety of developer tools and services, including Azure DevOps, Visual Studio, and GitHub integration.

Azure is designed to be flexible and scalable, making it suitable for businesses of all sizes. It provides various pricing options, including pay-as-you-go, subscription-based, and free plans, allowing businesses to choose the best pricing plan that meets their needs and budget.

Overall, **Azure** is a powerful and flexible cloud computing platform that provides a wide range of services and solutions for building, deploying, and managing applications and services. It is widely used by businesses of all sizes and industries, from startups to large enterprises, and is considered one of the leading cloud computing platforms in the market.

```cpp
#include<iostream>
```

```
using namespace std;
int main() {
cout<<"Hello,world!";
return 0;
cout<<"Hello,world!";
}
```

Solution:

```
Hello,world!
```

```
#include<iostream>
using namespace std;
int main () {
cout<<"linux\n";
exit (0);
cout<<"php\n";
return 0;
}
```

Solution:

```
linux
```

```
#include<iostream>
```

Apache Maven is a build automation tool and a powerful project management tool developed by the Apache Software Foundation. It is widely used for Java projects, but can also be used for other programming languages like C#, Ruby, and Scala. **Maven** simplifies the process of building, testing, and deploying Java-based applications by providing a standardized way to manage project dependencies, versioning, and packaging.

Maven uses an XML-based project object model (POM) to define a project, its dependencies, and its build configuration. This makes it easier to manage complex projects and ensures that all developers are using the same build processes and configurations. **Maven** manages dependencies by automatically downloading and integrating them into the project build, reducing the need for manual management.

Maven can be used to automate the building, testing, and deployment of Java applications, as well as generating documentation, creating reports, and generating code from templates. It can also be used to manage multiple projects, making it a popular choice for larger development teams working on multiple projects simultaneously. Overall, **Maven** is a versatile and powerful tool that can help developers manage the complexities of Java projects and streamline the software development process.

```cpp
using namespace std;
int main() {
for(int i=1; i<=5; i++) {
if(i==3) {
continue;
}
cout<<"\n "<<i;
}
return 0;
}
```

Solution:

```
1
2
4
5
```

```cpp
#include<iostream>
using namespace std;
int main() {
int a = 10, b = 20, c;
c = (a < b) ? a : b;
cout<<c;
return 0;
}
```

Jenkins is an open-source automation server that is used to automate the building, testing, and deployment of software. It is a popular tool used for continuous integration and continuous delivery (CI/CD) pipelines. **Jenkins** is written in Java and can be installed on various operating systems, including Windows, macOS, and Linux.

Jenkins provides a web-based interface that enables developers to create and manage workflows, including automated builds, tests, and deployments. It integrates with various development tools and services, including source code management systems like Git and Subversion, build tools like Maven and Gradle, and testing frameworks like JUnit and Selenium. Jenkins can also be used to orchestrate complex workflows, such as multi-stage deployment pipelines that include testing and approvals.

One of the key benefits of **Jenkins** is its flexibility and extensibility. It offers a large collection of plugins that can be used to extend its functionality, and developers can also create their own plugins to meet their specific needs. **Jenkins** is widely used in software development teams of all sizes, from small startups to large enterprises, and is considered a critical tool for implementing agile and DevOps practices. Overall, **Jenkins** is a powerful and flexible automation server that helps teams automate their software development processes and deliver high-quality software faster.

Solution:

```
10
```

```cpp
#include<iostream>
using namespace std;
#define A 15
int main() {
int x;
x=A;
cout<<x;
return 0;
}
```

Solution:

```
15
```

Kubernetes, commonly referred to as **K8s,** is an open-source container orchestration platform developed by Google. It automates the deployment, scaling, and management of containerized applications. **Kubernetes** provides a platform-agnostic way to manage containerized applications and services, enabling developers to deploy and manage their applications across multiple cloud providers and on-premise data centers.

Kubernetes enables developers to deploy containerized applications and services by defining a set of desired states in a **Kubernetes** manifest file. **Kubernetes** then automatically schedules and manages the containers, ensuring that the desired state is maintained. It provides features such as automatic scaling, rolling updates, self-healing, and load balancing, making it easier to manage and scale containerized applications.

Kubernetes is designed to be highly available, scalable, and fault-tolerant. It can run on a cluster of machines, making it possible to manage large-scale applications with ease. **Kubernetes** also provides an API that enables developers to automate the management of their containerized applications, making it a popular choice for DevOps teams.

Overall, **Kubernetes** is a powerful and flexible platform for managing containerized applications and services. It simplifies the management of containerized applications, making it easier for developers to focus on developing their applications rather than managing infrastructure.

```cpp
#include<iostream>
#include<cmath>
using namespace std;
int main() {
int x = 20;
cout<<"Inverse of tan x = "<< atan(x);
```

```
return 0;
}
```

Solution:

```
Inverse of tan x = 1.52084
```

```cpp
#include<iostream>
#include<cmath>
using namespace std;
int main() {
double a, b;
a = -2.5;
b = fabs(a);
cout<<"|"<<a<<"|" << " = "<<b;
return 0;
}
```

Functional programming is a programming paradigm that is focused on writing programs using pure functions, which are functions that do not modify any data outside of their scope, and have no side effects. Functional programming is based on the principles of mathematical functions, and emphasizes the use of expressions and immutability.

In **functional programming**, the emphasis is on writing code that is declarative rather than imperative. This means that instead of telling the computer how to perform a task, you specify what you want to accomplish, and the language takes care of how to achieve it. This approach helps to reduce complexity and increase readability, as the code is more concise and easier to understand.

Functional programming languages are often used in applications that require high levels of concurrency or parallelism, as they are well-suited to working with immutable data and avoiding mutable state. Some **popular functional programming languages** include Haskell, Lisp, Erlang, and Clojure. Many modern programming languages, such as JavaScript, Python, and Ruby, also support functional programming paradigms to some extent.

Solution:

```
|-2.5| = 2.5
```

```cpp
#include<iostream>
using namespace std;
int main() {
int x=12, y =3;
cout<<abs(-x-y);
return 0;
}
```

Solution:

15

```cpp
#include<iostream>
using namespace std;
int main() {
int x=12, y =3;
cout<<-(-x-y);
return 0;
}
```

Solution:

15

Logic programming is a programming paradigm that is based on the use of mathematical logic for problem-solving. In logic programming, programs are written in terms of a set of rules and facts, and the execution of the program involves deriving logical conclusions from these rules and facts.

The most well-known **logic programming language** is Prolog, which stands for "Programming in Logic". In Prolog, the program consists of a set of rules and facts that define relationships between objects. The program is executed by making queries to the database of rules and facts, and the system uses logical inference to derive answers to the queries.

One of the main advantages of **logic programming** is its ability to handle complex symbolic data structures and relationships. Logic programming languages are often used in areas such as natural language processing, expert systems, and artificial intelligence.

However, **logic programming** can be challenging for programmers who are used to imperative or object-oriented programming paradigms, as it requires a different way of thinking about problem-solving. Additionally, the efficiency of logic programming can be a concern in some cases, as the inference process can be computationally expensive.

```
#include <iostream>

using namespace std;

int main() {

int x=12, y =3;

cout<< x-(-y);

return 0;

}
```

A programming paradigm called "object-oriented programming" (**OOP**) is founded on the idea that "objects" can hold both data and the code needed to manipulate that data. In **OOP**, programs are designed by creating classes, which are templates or blueprints for creating objects. Each object created from a class has its own unique data and behavior, but shares the same structure and behavior defined in the class.

The key concepts of **OOP** are inheritance, encapsulation, and polymorphism. **Inheritance** allows a class to inherit properties and behaviors from a parent class, which can be useful for reusing code and creating hierarchies of related objects. **Encapsulation** involves hiding the details of an object's implementation from the outside world, which can make it easier to modify and maintain code. **Polymorphism** allows different objects to respond to the same message or method in different ways, which can make code more flexible and extensible.

OOP languages include Java, C++, Python, Ruby, and many others. **OOP** is widely used in many areas of software development, including desktop applications, web development, game development, and mobile app development. **OOP** has many advantages, including improved code organization, modularity, and code reusability. However, it can also be more complex than other programming paradigms, and can be challenging for beginners to learn.

Solution:

15

Question 60

Question:

Write a program to find the size of an array.

Solution:

```
#include<iostream>

using namespace std;
```

```cpp
int main() {
int num [] = {11, 22, 33, 44, 55, 66};
int n = sizeof(num) / sizeof(num [0]);
cout<<"Size of the array is: " << n;
return 0;
}
```

Question 61

Question:

Write a program that prints a sequence from 1 to a given integer, inserts a plus sign between these numbers, and then removes the plus sign at the end of the sequence.

Solution:

```cpp
#include<iostream>
using namespace std;
int main () {
int x, i;
cout<<"\nEnter a integer: \n";
cin>>x;
if(x>0) {
cout<<"Sequence from 1 to "<< x << ":\n";
for(i=1; i<x; i++)  {
cout<<i<<"+";
}
cout<<i<<"\n";
}
```

```
return 0;
}
```

Question 62

Question:

Write a program to verify whether a triangle's three sides form a right angled triangle or not.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int a,b,c;
cout<<"Enter the three sides of a triangle: \n";
cin>>a;
cin>>b;
cin>>c;
if((a*a)+(b*b)==(c*c) || (a*a)+(c*c)==(b*b) || (b*b)+(c*c)==(a*a)) {
cout<<"Triangle's three sides form a right angled triangle.\n";
}
else {
cout<<"Triangle's three sides does not form a right angled triangle.\n";
}
return 0;
}
```

Question 63

Question:

Write a program that will find the second-largest number among the user's input of three numbers.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int a, b, c;
cout<<"\nEnter the first number: ";
cin>>a;
cout<<"\nEnter the second number: ";
cin>>b;
cout<<"\nEnter the third number: ";
cin>>c;
if(a>b && a>c) {
    if(b>c)
            cout<<b<<" is second largest number among three numbers";
        else
            cout<<c<<" is second largest number among three numbers";
}
else if(b>c && b>a) {
    if(c>a)
            cout<<c<<" is second largest number among three numbers";
        else
```

```
            cout<<a<<" is second largest number among three numbers";
}
else if(a>b)
            cout<<a<<" is second largest number among three numbers";
        else
            cout<<b<<" is second largest number among three numbers";
    return 0;
}
```

Question 64

Question:

Write a program to calculate the sum of the two given integer values. Return three times the sum of the two values if they are equal.

Solution:

```cpp
#include<iostream>
using namespace std;
int myfunc();
int myfunc(int a, int b) {
return a == b ? (a + b)*3 : a + b;
}
int main() {
cout<<""<<myfunc(3, 5);
cout<<"\n"<<myfunc(6, 6);
return 0;
}
```

Question 65

Question:

Write a program that accepts minutes as input, and display the total number of hours and minutes.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int mins, hrs;
cout<<"Input minutes: ";
cin>>mins;
hrs=mins/60;
mins=mins%60;
cout<<hrs<<" Hours,"<<mins<< " Minutes.\n";
return 0;
}
```

AWS stands for Amazon Web Services, which is a cloud computing platform offered by Amazon. AWS provides a wide range of cloud-based services, including computing power, storage, and databases, as well as machine learning, security, and analytics tools. These services are available on-demand, allowing businesses and individuals to quickly and easily access the resources they need without having to invest in and maintain their own physical infrastructure. AWS is one of the leading cloud computing platforms in the world, used by millions of customers in over 190 countries.

Question

Question:

Write a program to determine whether a positive number entered by the user is a multiple of three or five.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x;
cout<<"\nEnter a number: ";
cin>>x;
if(x % 3 == 0 || x % 5 == 0) {
cout<<"True";
}
else {
cout<<"False";
}
return 0;
}
```

Selenium is an open-source testing platform for web browser automation. It provides a set of tools and libraries for automating web browsers, enabling developers and testers to automate web application testing. **Selenium** can be used with various programming languages such as Java, Python, Ruby, and C#.

Selenium can automate various web application testing tasks, including functional testing, regression testing, and user acceptance testing. It can interact with web elements, simulate user actions, and capture results, making it easier to automate web application testing.

Selenium also provides a set of tools for testing web applications across multiple browsers and platforms. It supports various web browsers, including Google Chrome, Mozilla Firefox, and Microsoft Edge, and can be used to test web applications on different operating systems, including Windows, macOS, and Linux.

Selenium is widely used in software development teams of all sizes and is considered a critical tool for implementing agile and DevOps practices. It enables developers to automate web application testing, making it easier to ensure that web applications are tested thoroughly and released with high quality. Overall, **Selenium** is a powerful and flexible testing framework that helps developers and testers automate web application testing and ensure the quality of their web applications.

Question

Question:

Write a program to verify whether one of the two entered integers falls within the range of 100 to 200 included.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x, y;
cout<<"\nEnter the value for x: ";
cin>>x;
cout<<"\nEnter the value for y: ";
cin>>y;
if((x >= 100 && x <= 200) || (y >= 100 && y <= 200)) {
cout<<"True";
}
else {
cout<<"False";
}
return 0;
}
```

Ansible is an open-source IT automation tool that simplifies the management and orchestration of complex IT infrastructure. It allows users to automate repetitive tasks, manage configuration files, and deploy applications across multiple servers and environments. **Ansible** is designed to be simple and easy to use, with a **YAML-based scripting language** that is human-readable and easily understood by both developers and non-developers. It uses a push-based model, meaning that configuration changes are pushed out to the target hosts in a controlled and predictable manner.

Ansible uses **SSH** (Secure Shell) protocol to connect to remote hosts, making it easy to manage and automate servers across a network. It also supports a wide range of modules, which are pre-built pieces of code that perform specific tasks such as installing software packages, configuring services, and creating users. **Ansible** can be used for a wide range of IT automation tasks, including configuration management, application deployment, and orchestration of infrastructure components such as cloud services, storage systems, and networking devices. Overall, **Ansible** is a powerful and flexible IT automation tool that simplifies the management and orchestration of complex IT infrastructure, allowing organizations to streamline their operations and reduce the time and effort required to manage their IT resources.

Question

Question:

Write a program to determine which of the two given integers is closest to the value 100.

If the two numbers are equal, return 0.

Solution:

```cpp
#include<iostream>
using namespace std;
int myfunc();
int myfunc(int a, int b) {
int x = abs(a - 100);
int y = abs(b - 100);
return x == y ? 0 : (x < y ? a : b);
}
int main() {
cout<<" "<< myfunc(86, 99);
cout<<"\n "<<myfunc(55, 55);
cout<<"\n "<<myfunc(65, 80);
return 0;
}
```

Terraform is an open-source infrastructure as code (IaC) tool that is used for building, changing, and versioning infrastructure in a safe, efficient, and repeatable way. It allows developers and infrastructure teams to define infrastructure as code, which can be versioned and managed just like any other software code. **Terraform** works by defining the desired state of the infrastructure in a declarative language called **HashiCorp Configuration Language (HCL)**. The desired state is then applied to the infrastructure by running **Terraform**, which compares the desired state to the current state and makes any necessary changes to bring the infrastructure into compliance with the desired state.

Terraform supports a wide range of cloud providers and services, including Amazon Web Services, Microsoft Azure, Google Cloud Platform, and many others. It also supports on-premises infrastructure, allowing teams to manage infrastructure across multiple cloud and on-premises environments. **Terraform** provides a range of benefits for infrastructure teams, including increased automation, improved collaboration, and reduced risk of human error. It also provides greater transparency and auditability, allowing teams to easily track changes to infrastructure and roll back changes if necessary. Overall, **Terraform** is a powerful infrastructure as code tool that allows teams to define and manage infrastructure in a safe, efficient, and repeatable way. It is widely used in software development teams of all sizes and is considered a critical tool for implementing modern infrastructure practices such as DevOps and infrastructure automation.

Question

Question:

Write a program to determine whether a positive number entered by the user is a multiple of three or five, but not both.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x;
cout<<"\nEnter a number: ";
cin>>x;
if(x % 3 == 0 ^ x % 5 == 0) {
cout<<"True";
}
else {
cout<<"False";
}
return 0;
}
```

Nagios is an open-source monitoring system that is used for monitoring the health and performance of IT infrastructure. It provides a range of tools and features for monitoring servers, network devices, applications, and services, and can be used to detect and diagnose problems before they affect end-users.

Nagios works by monitoring various metrics and alerts, such as CPU usage, memory usage, disk space, network traffic, and application availability. It uses a web interface to display the status of monitored objects and provides alerts and notifications when there are issues or failures.

Nagios is highly customizable and extensible, allowing users to create custom plugins and add-ons to monitor a wide range of systems and services. It also supports various third-party plugins and integrations, making it a versatile and flexible monitoring system.

Nagios is widely used in IT operations teams of all sizes and is considered a critical tool for monitoring and maintaining IT infrastructure. It provides a range of benefits, including increased visibility and control, improved uptime and reliability, and reduced mean time to repair (MTTR) in the event of failures or incidents.

Overall, **Nagios** is a powerful and flexible monitoring system that provides IT teams with the tools and features they need to monitor and maintain IT infrastructure effectively. It is a critical tool for ensuring the performance, availability, and reliability of IT infrastructure and services.

Question

Question:

Write a program to determine whether two entered non-negative numbers have the same last digit.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x, y;
cout<<"\nEnter the value for x: ";
cin>>x;
cout<<"\nEnter the value for y: ";
cin>>y;
if(abs(x % 10) == abs(y % 10)) {
cout<<"True";
}
else {
cout<<"False";
}
return 0;
}
```

Gradle is an open-source build automation tool that is used for building, testing, and deploying software projects. It provides a flexible and powerful build system that can be used to automate various tasks in the software development process.

Gradle uses a build script that is written in a **Groovy or Kotlin-based domain-specific language** (DSL). The build script describes the dependencies and tasks required to build the software project, and **Gradle** uses this information to generate the build output.

Gradle supports a wide range of programming languages and platforms, including Java, Kotlin, Groovy, Android, and C++. It provides a rich set of features for building and testing software projects, including incremental builds, parallel builds, and dependency management.

Gradle also supports various third-party plugins and integrations, making it a versatile and flexible build tool. It can be used with various IDEs, such as Eclipse and IntelliJ IDEA, and can be integrated with various continuous integration and deployment (CI/CD) tools, such as Jenkins and Travis CI.

Gradle is widely used in software development teams of all sizes and is considered a critical tool for implementing modern software development practices such as DevOps and continuous delivery. It provides a range of benefits, including improved automation, increased efficiency, and reduced build times.

Overall, **Gradle** is a powerful and flexible build automation tool that provides developers and development teams with the tools and features they need to build, test, and deploy software projects effectively.

Question 71

Question:

Write a program to determine whether a given non-negative number is a multiple of 12 or it is one more than a multiple of 12.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x = 43;
if(x % 12 == 0 || x % 12 == 1) {
cout<<"True";
}
else {
cout<<"False";
}
return 0;
}
```

API (Application Programming Interface) **testing** is a type of software testing that focuses on verifying the functionality, reliability, security, and performance of an **API**. An **API** is a set of protocols and standards used to build and integrate software applications. It provides a way for two different software systems to communicate with each other. **API testing** involves sending requests to an **API** and receiving responses, then validating that the responses are correct and meet the expected requirements. Some common types of **API tests** include:

- **Functional testing:** verifying that the API performs the intended functions and behaves as expected.
- **Performance testing:** evaluating the response time, throughput, and resource utilization of the API.
- **Security testing:** identifying vulnerabilities and weaknesses in the API that could lead to unauthorized access or data breaches.
- **Compatibility testing:** ensuring that the API works with various operating systems, browsers, and devices.

API testing can be done manually or using automated testing tools. Automated testing tools can save time and increase efficiency by automating repetitive tasks and providing quick feedback on the performance and functionality of the **API**. To sum up, **API testing** is a crucial step in the software development lifecycle to ensure that the **API** is working as intended and meets the desired quality standards.

Question 72

Question:

Write a program that accepts two integers and returns true when one of them equals 6, or when their sum or difference equals 6.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x, y;
cout<<"\nEnter the value for x: ";
cin>>x;
cout<<"\nEnter the value for y: ";
cin>>y;
if(x == 6 || y == 6 || x + y == 6 || abs(x - y) == 6) {
cout<<"True";
}
else {
cout<<"False";
}
return 0;
}
```

Question 73

Question:

Write a program to check whether it is possible to add two integers to get the third integer from three entered integers.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x, y, z;
cout<<"\nEnter the value for x: ";
cin>>x;
cout<<"\nEnter the value for y: ";
cin>>y;
cout<<"\nEnter the value for z: ";
cin>>z;
if(x == y + z || y == x + z || z == x + y) {
cout<<"True";
}
else {
cout<<"False";
}
return 0;
}
```

Question 74

Question:

Write a program that converts kilometers per hour to miles per hour.

Solution:

```cpp
#include<iostream>
```

```cpp
using namespace std;
int main() {
float kmph;
cout<<"Enter kilometers per hour: ";
cin>>kmph;
cout<<(kmph * 0.6213712)<<" miles per hour";
return 0;
}
```

Question 75

Question:

Write a program to calculate area of an ellipse.

Solution:

```cpp
#include<iostream>
using namespace std;
#define PI 3.141592
int main() {
float major, minor;
cout<<"\nEnter length of major axis: ";
cin>>major;
cout<<"\nEnter length of minor axis: ";
cin>>minor;
cout<<"\nArea of an ellipse = "<< (PI * major * minor);
return 0;
}
```

Question 76

Question:

Write a program to calculate the sum of three given integers. Return the third value if the first two values are equal.

Solution:

```cpp
#include<iostream>
using namespace std;
int myfunc();
int myfunc(int a, int b, int c) {
if (a == b && b == c) return 0;
if (a == b) return c;
if (a == c) return b;
if (b == c) return a;
else return a + b + c;
}
int main() {
cout<<"\n"<<myfunc(11, 11, 11);
cout<<"\n"<<myfunc(11, 11, 16);
cout<<"\n"<<myfunc(18, 15, 10);
return 0;
}
```

```cpp
#include<iostream>
#include<cstring>
using namespace std;

void myfunc(string x= "John") {
  cout << x;
}

int main() {
myfunc();
return 0;
}
```

?

Question 77

Question:

Write a program to convert bytes to kilobytes.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int bytes;
cout<<"\nEnter number of bytes: ";
cin>>bytes;
cout<<"\nKilobytes: "<<(bytes/1024);
return 0;
}
```

Question 78

Question:

Write a program to convert megabytes to kilobytes.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
double megabytes, kilobytes;
cout<<"\nInput the amount of megabytes to convert: ";
cin>>megabytes;
kilobytes = megabytes * 1024;
cout<<"\nThere are "<<kilobytes<< " kilobytes in " <<megabytes<< "
megabytes.";
return 0;
}
```

Question 79

Question:

Write a program to count the number of even elements in an integer array.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int array[1000], i, arr_size, even=0;
cout<<"Input the size of the array: ";
cin>>arr_size;
cout<<"Enter the elements in array: \n";
for(i=0; i<arr_size; i++) {
cin>>array[i];
```

```
}

for(i=0; i<arr_size; i++) {
if(array[i]%2==0) {
    even++;
}
}
cout<<"Number of even elements: "<< even;
return 0;
}
```

Question 80

Question:

Write a program to count the number of odd elements in an integer array.

Solution:

```
#include<iostream>
using namespace std;
int main() {
int array[1000], i, arr_size, odd=0;
cout<<"Input the size of the array: ";
cin>>arr_size;
cout<<"Enter the elements in array: \n";
for(i=0; i<arr_size; i++) {
cin>>array[i];
```

```
}

for(i=0; i<arr_size; i++) {
if(array[i]%2!=0) {
    odd++;
}
}
cout<<"Number of odd elements: "<< odd;
return 0;
}
```

Question 81

Question:

Write a program that will accept two integers and determine whether or not they are equal.

Solution:

```
#include<iostream>
using namespace std;
int main() {
int x, y;
cout<<"Input the values for x and y: \n";
cin>>x;
cin>>y;
if(x == y) {
    cout<<"x and y are equal\n";
```

```
}
else {
    cout<<"x and y are not equal\n";
}
return 0;
}
```

Question 82

Question:

Write a program to find the third angle of a triangle if two angles are given.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int angle1, angle2;
cout<<"\nEnter the first angle of the triangle: ";
cin>>angle1;
cout<<"\nEnter the second angle of the triangle: ";
cin>>angle2;
cout<<"\nThird angle of the triangle is: "<< (180 - (angle1 + angle2));
return 0;
}
```

Question 83

Question:

Write a program to determine whether a particular year is a leap year or not.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int year;
cout<<"Enter the year: ";
cin>>year;
if((year % 400) == 0) {
cout<<year<<" is a leap year.";
}
else if((year % 100) == 0) {
cout<<year<<" is a not leap year.";
}
else if((year % 4) == 0) {
cout<<year<<" is a leap year.";
}
else {
cout<<year<<" is not a leap year.";
}
return 0;
}
```

Question 84

Question:

Write a program that reads the candidate's age and determine a candidate's eligibility to cast his own vote.

Solution:

```cpp
#include <iostream>
using namespace std;
int main() {
int age;
cout<<"\nEnter the age of the candidate: ";
cin>>age;
if(age<18) {
cout<<"\nWe apologize, but the candidate is not able to cast his vote.";
cout<<"\nAfter "<< (18-age) <<" year, the candidate would be able to cast his vote.";
}
else {
cout<<"Congratulation! the candidate is qualified to cast his vote.\n";
}
return 0;
}
```

Question 85

Question:

Write a program to Convert Yard to Foot.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
float yard;
cout<<"\nEnter the Length in Yard: ";
cin>>yard;
cout<<yard<<" Yard in Foot is: "<<(3*yard);
return 0;
}
```

Question 86

Question:

Write a program to convert gigabytes to megabytes.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
double gigabytes, megabytes;
cout<<"\nInput the amount of gigabytes to convert: ";
cin>>gigabytes;
megabytes = gigabytes*1024;
cout<<"\nThere are "<<megabytes<<" megabytes in "<<gigabytes<<" gigabytes.";
return 0;
}
```

Question 87

Question:

Write a program to Convert Kilogram to Pounds.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
float kg, lbs;
cout<<"\nEnter Weight in Kilogram: ";
cin>>kg;
lbs = kg*2.20462;
cout<<kg<<" Kg = "<<lbs<<" Pounds";
return 0;
```

```
}                                               527
```

Question 88

Question:

Write a program to Convert Kilogram to Ounce.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
float kg, ounce;
cout<<"\nEnter Weight in Kilogram: ";
cin>>kg;
ounce = kg*35.274;
cout<<kg<<" Kg = "<<ounce<< " Ounce";
return 0;
}
```

Question 89

Question:

Write a program to Convert Pounds to Grams.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
float pound, gram;
cout<<"\nEnter Weight in Pounds: ";
cin>>pound;
gram = pound*453.592;
cout<<pound<<" Pound = "<<gram<<" Grams";
return 0;
}
```

Question 90

Question:

Write a program to verify whether a triangle is valid or not using angles.

Solution:

```cpp
#include <iostream>
using namespace std;
int main() {
int angle1, angle2, angle3, sum;
cout<<"\nEnter the first angle of the triangle: ";
cin>>angle1;
```

```cpp
cout<<"\nEnter the second angle of the triangle: ";
cin>>angle2;
cout<<"\nEnter the third angle of the triangle: ";
cin>>angle3;
sum = angle1 + angle2 + angle3;
if(sum == 180) {
cout<<"\nThe triangle is valid.";
}
else {
cout<<"\nThe triangle is not valid.";
}
return 0;
}
```

Question 91

Question:

Write a program to add the digits of a two-digit number that is entered by the user.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x, y, sum = 0;
cout<<"\nEnter a two-digit number: ";
cin>>x;
y = x;
```

```
while(y != 0) {
sum = sum + y % 10;
y = y / 10;
}
cout<<"\nSum of digits of "<<x<<" is: "<<sum;
return 0;
}
```

Question 92

Question:

Write a program to verify if a character you entered is a vowel or a consonant.

Solution:

```
#include<iostream>
using namespace std;
int main() {
char ch;
cout<<"\nEnter a character: ";
cin>>ch;
if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ||
 ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U' ) {

cout<<ch<<" is a vowel";
}
else {
```

```cpp
cout<<ch<<" is a consonant";
}
return 0;
}
```

Question 93

Question:

Write a program to find factorial of a number.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int i, fact=1, num;
cout<<"\nEnter a number: ";
cin>>num;
for(i=1; i<=num; i++) {
    fact=fact*i;
}
cout<<"\nFactorial of "<<num<<" is: "<<fact;
return 0;
}
```

Question 94

Question:

Write a program to print number of days in a month.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x[12]={31,28,31,30,31,30,31,31,30,31,30,31}, m;
cout<<"\nEnter the month number: ";
cin>>m;
if(m>12 || m<1) {
cout<<"Invalid input";
}
else if(m==2) {
cout<<"\nNumber of days in month 2 is either 29 or 28";
}
else {
cout<<"\nNumber of days in month "<<m<< " is: "<<x[m-1];
}
return 0;
}
```

Question 95

Question:

Write a program to concatenate two strings.

Solution:

```cpp
#include<iostream>
#include<cstring>
using namespace std;
int main() {
char a[1000], b[1000];
cout<<"\nEnter the first string: ";
cin>>a;
cout<<"\nEnter the second string: ";
cin>>b;
strcat(a, b);
cout<<"\nString produced by concatenation is: "<< a;
return 0;
}
```

Question 96

Question:

Write a program to find maximum between two numbers.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int a,b;
cout<<"Enter two numbers: \n";
cin>>a;
cin>>b;
if(a>b) {
cout<<a<<" is a maximum number";
}
else {
cout<<b<<" is a maximum number";
}
return 0;
}
```

Type theory is a branch of mathematical logic and computer science that studies the behavior of types, which are categories of data that share certain properties or operations. The goal of type theory is to formalize the structure and behavior of types, and to use this formalism to reason about programs, systems, and mathematical objects.

In **type theory**, types are used to classify data and to ensure that operations are applied to the right kind of data. For example, in a programming language that supports type checking, a function that is defined to accept integers will not accept strings, because they are of a different type. This type checking ensures that the function is applied only to the appropriate types of data, which can help prevent errors and improve program correctness.

Type theory has many applications in computer science, including programming language design, software verification, and formal methods. It is also used in the study of mathematics, where it provides a way to reason about the behavior of mathematical objects and to prove theorems using formal logic.

Question 97

Question:

Write a program to compare two strings.

Solution:

```cpp
#include<iostream>
```

```cpp
#include<cstring>
using namespace std;
int main() {
char a[100], b[100];
cout<<"Enter the first string: \n";
cin>>a;
cout<<"Enter the second string: \n";
cin>>b;
if (strcmp(a,b) == 0) {
cout<<"The 2 strings are equal.\n";
}
else {
cout<<"The 2 strings are not equal.\n";
}
return 0;
}
```

Numerical analysis is a branch of mathematics that deals with the development and implementation of algorithms for solving mathematical problems using numerical methods. It involves the study of numerical algorithms, their implementation, and their analysis in terms of accuracy, stability, and efficiency.

Numerical analysis is used in many fields, including engineering, science, finance, and computer science. It is used to solve problems that cannot be solved analytically or with closed-form solutions, such as differential equations, optimization problems, and systems of linear equations. Numerical analysis also plays an important role in scientific computing, where it is used to simulate and model complex systems.

Some of the most common techniques used in numerical analysis include numerical integration, numerical differentiation, root-finding algorithms, linear and nonlinear systems of equations, and optimization algorithms. These techniques rely on a combination of mathematical theory, computational algorithms, and numerical experimentation to develop and refine numerical methods for solving complex problems.

To sum up, numerical analysis provides a set of powerful tools for solving mathematical problems that cannot be solved analytically. It has applications in a wide range of fields and is essential for many scientific and engineering applications.

Question 98

Question:

Write a program to convert the upper case letter to lower case letter.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
char ch = 'G';
```

```cpp
char b = tolower(ch);
cout<<ch<<" in lowercase is represented as "<< b;
return 0;
}
```

Question 99

Question:

Write a program to find the quotient and remainder of an entered dividend and divisor.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int dividend, divisor;
cout<<"\nEnter dividend: ";
cin>>dividend;
cout<<"\nEnter divisor: ";
cin>>divisor;
cout<<"\nQuotient = "<< (dividend / divisor);
cout<<"\nRemainder = "<< (dividend % divisor);
return 0;
}
```

Symbolic computation refers to the manipulation of mathematical expressions using symbols and rules of algebra, without numerical computation. In other words, it involves performing algebraic operations on variables and mathematical functions, rather than calculating numerical values.

Symbolic computation can be performed manually, but it is often done using computer software such as computer algebra systems (CAS) or programming languages like Python, MATLAB, or Mathematica. These tools can perform complex symbolic calculations that would be difficult, if not impossible, to do by hand.

Symbolic computation is used in a wide range of applications, including theoretical physics, engineering, computer science, cryptography, and more. It enables researchers and practitioners to explore and understand mathematical concepts and models, without relying on numerical approximations.

Question 100

Question:

Write a program to determine the Size of int, float, double and char.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
cout<<"Size of char is: "<<sizeof(char)<<" byte\n";
cout<<"Size of int is: "<<sizeof(int)<<" bytes\n";
cout<<"Size of float is: "<<sizeof(float)<<" bytes\n";
cout<<"Size of double is: "<<sizeof(double)<<" bytes\n";
return 0;
}
```

Question 101

Question:

Write a program to verify the password until it is correct.

Solution:

```cpp
#include<iostream>
```

```cpp
using namespace std;
int main() {
int pwd, i;
while (i!=0) {
cout<<"\nEnter the password: ";
cin>>pwd;
if(pwd==1988) {
cout<<"The password you entered is correct";
i=0;
}
else {
cout<<"Incorrect password, try again";
}
cout<<"\n";
}
return 0;
}
```

Question 102

Question:

Write a program to find absolute value of a number.

Solution:

```cpp
#include<iostream>
using namespace std;
```

```cpp
int main() {
int num;
cout<<"Input a positive or negative number: \n";
cin>>num;
cout<<"\nAbsolute value of "<<"|"<<num<<"|"<<" is: "<<abs(num);
return 0;
}
```

Question 103

Question:

Write a program that will accept a person's height in cm and classify the person based on it.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
float ht;
cout<<"\nEnter the height (in cm): ";
cin>>ht;
if(ht < 150.0) {
cout<<"Dwarf.\n";
}
else if((ht >= 150.0) && (ht < 165.0)) {
cout<<"Average Height.\n";
```

```
}
else if((ht >= 165.0) && (ht <= 195.0)) {
cout<<"Taller.\n";
}
else {
cout<<"Abnormal height.\n";
}
return 0;
}
```

Question 104

Question:

Write a program to calculate the area of different geometric shapes using switch statements.

Solution:

```
#include<iostream>
using namespace std;
int main() {
int choice;
float r, l, w, b, h;
cout<<"\nEnter 1 for area of circle: ";
cout<<"\nEnter 2 for area of rectangle: ";
cout<<"\nEnter 3 for area of triangle: ";
cout<<"\nEnter your choice : ";
cin>>choice;
```

```cpp
switch(choice) {
case 1:
cout<<"Enter the radius of the circle: ";
cin>>r;
cout<<"\nArea of a circle is: " << (3.14*r*r);
break;
case 2:
cout<<"Enter the length and width of the rectangle: \n";
cin>>l;
cin>>w;
cout<<"\nArea of a rectangle is: "<<(l*w);
break;
case 3:
cout<<"Enter the base and height of the triangle: \n";
cin>>b;
cin>>h;
cout<<"\nArea of a triangle is: "<<(0.5*b*h);
break;
default:
cout<<"\nPlease enter a number from 1 to 3.";
break;
}
return 0;
}
```

Bitbucket is a web-based version control repository hosting service that is used to store and manage code repositories. It is developed by Atlassian and provides Git and Mercurial-based distributed version control systems (DVCS) for source code management. With **Bitbucket**, teams can collaborate on software projects, manage code repositories, track changes, and perform code reviews. It offers a range of features, such as pull requests, code branching, and commit history tracking, which allow developers to work together efficiently and effectively.

Bitbucket is integrated with other Atlassian products, such as Jira and Confluence, making it easy to manage issues and track progress across the entire software development process. It also provides tools for continuous integration and deployment (CI/CD), allowing teams to automate their build and deployment pipelines. **Bitbucket** is available in both cloud-based and self-hosted versions. The cloud-based version offers a range of pricing plans, depending on the number of users and repositories required. The self-hosted version can be installed on-premise or on a cloud-based infrastructure, giving organizations greater control over their source code management. Overall, **Bitbucket** is a powerful and flexible source code management tool that allows teams to collaborate on software development projects, manage code repositories, and automate their build and deployment pipelines. It is a popular choice for teams of all sizes and offers a range of features to support the entire software development process.

Question 105

Question:

Write a program to accept a character from the keyboard and print "Yes" if it is equal to y. Otherwise print "No".

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
char ch;
cout<<"Enter a character: ";
ch = getchar ();
if(ch == 'y' || ch == 'Y') {
cout<<"Yes\n";
}
else {
cout<<"No\n";
}
return(0);
}
```

IntelliJ IDEA is an integrated development environment (IDE) used primarily for developing software applications in the Java programming language. It is developed by **JetBrains** and is available in both community and commercial editions. **IntelliJ IDEA** offers a range of features that make it a popular choice among developers, including intelligent code completion, code analysis, and refactoring tools. It also provides built-in support for a range of programming languages, including Java, Kotlin, Groovy, Scala, and JavaScript.

One of the key strengths of **IntelliJ IDEA** is its support for a wide range of frameworks and technologies, such as Spring, Hibernate, and Maven. It also provides integrations with other tools, such as Git, Subversion, and JUnit, allowing developers to streamline their development workflows. **IntelliJ IDEA** offers a highly customizable user interface and supports a wide range of plugins and extensions, making it possible to tailor the **IDE** to suit the specific needs of the developer. Overall, **IntelliJ IDEA** is a powerful and feature-rich IDE that offers a range of tools and technologies to support software development in the Java programming language. It is highly customizable and provides integrations with a range of tools and technologies, making it a popular choice among developers.

Question 106

Question:

Write a program that uses bitwise operators to multiply an entered value by four.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
long x, y;
cout<<"Enter a integer: ";
cin>>x;
y = x;
x = x << 2;
cout<< y<<" x 4 = "<< x;
return 0;
}
```

Question 107

Puppet is a configuration management tool that is widely used in the DevOps culture to automate the management of IT infrastructure. **DevOps** is a software development approach that emphasizes collaboration and communication between development and operations teams to streamline the software delivery process. In the context of **DevOps**, **Puppet** allows teams to automate the management of infrastructure and applications, ensuring that all systems are configured and deployed consistently and reliably. It provides a centralized platform for managing configurations, deployments, and updates across multiple environments, reducing the time and effort required to manage complex IT infrastructure.

Puppet works by defining infrastructure as code, which means that configurations and deployments are defined using code that can be version-controlled, tested, and audited. This approach helps to eliminate configuration drift and reduces the risk of errors or inconsistencies in the infrastructure. Using **Puppet** as part of a **DevOps** approach helps to promote collaboration and communication between development and operations teams, enabling faster and more frequent software releases. It also allows organizations to scale their infrastructure more easily and respond quickly to changing business needs. To sum up, **Puppet** is a powerful tool that plays a key role in the **DevOps** culture, providing a centralized platform for managing infrastructure and applications, and helping to streamline the software delivery process. It promotes collaboration and communication between development and operations teams, enabling organizations to deliver software more quickly and efficiently.

Question:

Write a program to check whether a number entered by the user is power of 2 or not.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x;
cout<<"Enter a number: ";
```

```cpp
cin>>x;
if((x != 0) && ((x &(x - 1)) == 0)) {
cout<<x<<" is a power of 2";
}
else {
cout<<x<<" is not a power of 2";
}
return 0;
}
```

Chef is a configuration management tool that is widely used in the DevOps culture to automate the management of IT infrastructure. DevOps is a software development approach that emphasizes collaboration and communication between development and operations teams to streamline the software delivery process. In the context of **DevOps**, **Chef** allows teams to automate the management of infrastructure and applications, ensuring that all systems are configured and deployed consistently and reliably. It provides a centralized platform for managing configurations, deployments, and updates across multiple environments, reducing the time and effort required to manage complex IT infrastructure. **Chef** works by defining infrastructure as code, which means that configurations and deployments are defined using code that can be version-controlled, tested, and audited. This approach helps to eliminate configuration drift and reduces the risk of errors or inconsistencies in the infrastructure. Using **Chef** as part of a **DevOps** approach helps to promote collaboration and communication between development and operations teams, enabling faster and more frequent software releases. It also allows organizations to scale their infrastructure more easily and respond quickly to changing business needs. **Chef** is highly extensible and can be used to manage a wide range of systems, including servers, cloud infrastructure, and containers. It also integrates with a range of other tools and technologies, such as Jenkins, Git, and Docker, allowing teams to build end-to-end software delivery pipelines.

Question 108

Question:

Write a program to determine whether a triangle is scalene, isosceles, or equilateral.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int side1, side2, side3;
cout<<"\nEnter the first side of the triangle: ";
cin>>side1;
cout<<"\nEnter the second side of the triangle: ";
cin>>side2;
cout<<"\nEnter the third side of the triangle: ";
```

```
cin>>side3;
if(side1 == side2 && side2 == side3) {
cout<<"\nThe given Triangle is equilateral.";
}
else if(side1 == side2 || side2 == side3 || side3 == side1) {
cout<<"\nThe given Triangle is isosceles.";
}
else {
cout<<"\nThe given Triangle is scalene.";
}
return 0;
}
```

Question 109

Question:

Write a program to print ASCII values of all the letters of the English alphabet from A to Z.

Solution:

```
#include<iostream>
using namespace std;
int main() {
int i;
for(i='A'; i<='Z'; i++) {
cout<<"ASCII value of "<<char(i)<<"="<<int(i)<<endl;
```

```
}
return 0;
}
```

Question 110

Question:

Write a program to find sum of even numbers between 1 to n.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int i, num, sum=0;
cout<<"Enter a number: ";
cin>>num;
for(i=2; i<=num; i=i+2) {
sum = sum + i;
}
cout<<"\nSum of all even number between 1 to " <<num<< " is: "<< sum;
return 0;
}
```

Question 111

Question:

Write a program to find sum of odd numbers between 1 to n.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int i, num, sum=0;
cout<<"Enter a number: ";
cin>>num;
for(i=1; i<=num; i=i+2) {
sum = sum + i;
}
cout<<"\nSum of all odd number between 1 to " <<num<< " is: "<< sum;
return 0;
}
```

Question 112

Question:

Write a program to find maximum number using switch case.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x, y;
cout<<"Enter any two numbers: \n";
cin>>x;
cin>>y;
switch(x > y) {
case 0: cout<<y<<" is Maximum number";
break;
case 1: cout<<x<<" is Maximum number";
break;
}
return 0;
}
```

Question 113

Question:

Write a program that allows you to enter the cost price and the selling price of a product and calculate profit or loss.

Solution:

```cpp
#include<iostream>
```

```cpp
using namespace std;
int main() {
int cp, sp;
cout<<"\nInput Cost Price: ";
cin>>cp;
cout<<"\nInput Selling Price: ";
cin>>sp;
if(sp > cp) {
cout<<"Profit = "<< (sp - cp);
}
else if(cp > sp) {
cout<<"Loss = "<< (cp - sp);
}
else {
cout<<"No Profit No Loss.";
}
return 0;
}
```

Apache NetBeans is a free and open-source integrated development environment (IDE) used primarily for developing software applications in the **Java programming language**. It is developed by the **Apache Software Foundation** and is available for Windows, Linux, and macOS operating systems. **NetBeans** offers a range of features that make it a popular choice among developers, including intelligent code completion, debugging tools, and refactoring tools. It also provides support for a range of programming languages, including Java, PHP, C++, and HTML/CSS/JavaScript.

One of the key strengths of **NetBeans** is its support for a wide range of frameworks and technologies, such as Spring, Hibernate, and Maven. It also provides integrations with other tools, such as Git, Subversion, and JUnit, allowing developers to streamline their development workflows. **NetBeans** offers a highly customizable user interface and supports a range of plugins and extensions, making it possible to tailor the IDE to suit the specific needs of the developer. To sum up, **Apache NetBeans** is a powerful and feature-rich IDE that offers a range of tools and technologies to support software development in the Java programming language and other languages. It is highly customizable and provides integrations with a range of tools and technologies, making it a popular choice among developers.

Question 114

Question:

Write a program that display the pattern like a right angle triangle using an asterisk.

Solution:

```cpp
#include<iostream>
```

```
using namespace std;
int main() {
int rows;
cout<<"Input the number of rows: ";
cin>>rows;
for(int x=1; x<=rows; x++) {
for(int y=1; y<=x; y++)
cout<<"*";
cout<<"\n";
}
return 0;
}
```

Question 115

Question:

Write a program that display the pattern like a right angle triangle using a number.

Solution:

```
#include<iostream>
using namespace std;
int main() {
int rows;
cout<<"Input the number of rows: ";
cin>>rows;
for(int x=1; x<=rows; x++) {
for(int y=1; y<=x; y++)
```

```
cout<<""<<y;
cout<<"\n";
}
return 0;
}
```

Question 116

Question:

Write a program to determine the number and sum of all integers between 50 and 100 which are divisible by 2.

Solution:

```
#include<iostream>
using namespace std;
int main() {
int x, sum=0;
cout<<"Numbers between 50 and 100, divisible by 2: \n";
for(x=51; x<100; x++) {
if(x%2==0) {
cout<<" "<<x;
sum+=x;
}
}
cout<<"\nThe sum: "<< sum;
return 0;
```

```
}
```

552

Question 117

Question:

Write a program that uses the function to determine whether a entered number is even or odd.

Solution:

```cpp
#include<iostream>
using namespace std;
int myfunc(int x) {
return (x & 1);
}
int main() {
int x;
cout<<"Enter any number: ";
cin>>x;
if(myfunc(x)) {
cout<<"\nThe number you entered is odd.";
}
else {
cout<<"\nThe number you entered is even.";
}
return 0;
}
```

Question 118

Question:

Write a program to find square root of a entered number.

Solution:

```cpp
#include<iostream>
#include<cmath>
using namespace std;
int main() {
int x;
cout<<"Enter any number: ";
cin>>x;
cout<<"Square root of "<<x<< " is: "<<(double)sqrt(x);
return 0;
}
```

Question 119

Question:

Write a program to find power of a entered number using library function.

Solution:

```cpp
#include<iostream>
#include<cmath>
using namespace std;
int main() {
int x, y;
cout<<"\nEnter the value for x: ";
cin>>x;
cout<<"\nEnter the value for y: ";
cin>>y;
cout<<x<<"^"<<y<<" = " << (long)pow(x,y);
return 0;
}
```

Question 120

Question:

Write a program to determine if the character entered is an alphabetic or numeric character.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
char ch;
cout<<"Enter a character: ";
```

```cpp
cin>>ch;
if(isdigit(ch)) {
cout<<ch<<" is a Digit";
}
else if(isalpha(ch)) {
cout<<ch<<" is an Alphabet";
}
else {
cout<<ch<<" is not an Alphabet, or a Digit";
}
return 0;
}
```

Question 121

Question:

Write a program to determine whether the character entered is an alphanumeric character or not.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
char a;
cout<<"Enter a character: ";
cin>>a;
```

```
if(isalnum(a)) {
cout<<a<<" is an alphanumeric character.";
}
else {
cout<<a<<" is NOT an alphanumeric character.";
}
return 0;
}
```

Question 122

Question:

Write a program to determine whether the character entered is an punctuation character or not.

Solution:

```
#include<iostream>
using namespace std;
int main() {
char a;
cout<<"Enter a character: ";
cin>>a;
if(ispunct(a)) {
cout<<a<<" is an punctuation character.";
}
else {
```

```
cout<<a<<" is NOT an punctuation character.";
}
return 0;
}
```

Question 123

Question:

Write a program to check whether the entered character is a graphic character or not.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
char a;
cout<<"Enter a character: ";
cin>>a;
if(isgraph(a)) {
cout<<a<<" is a graphic character.";
}
else {
cout<<a<<" is NOT a graphic character.";
}
return 0;
}
```

Question 124

Question:

Write a program to list all printable characters using isprint() function.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int i;
for(i = 1; i <= 127; i++)
if(isprint(i)!= 0)
cout<<" "<<char(i);
return 0;
}
```

Question 125

Question:

Write a program to check whether the entered character is a hexadecimal digit character or not.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
char a;
cout<<"Enter a character: ";
cin>>a;
if(isxdigit(a)) {
cout<<a<<" is a hexadecimal digit character.";
}
else {
cout<<a<<" is NOT a hexadecimal digit character.";
}
return 0;
}
```

Question 126

Question:

Write a program to print ASCII value of all control characters.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int i;
cout<<"The ASCII value of all control characters are: \n";
```

```cpp
for(i=0; i<=127; i++) {
if(iscntrl(i)!=0)
cout<<"\n  "<< i;
}
return 0;
}
```

```cpp
#include<iostream>
#include<cstring>
using namespace std;
int main() {
string x = "Joe";
string* ptr = &x;
cout << ptr << endl;
cout << *ptr << endl;
return 0;
}
```
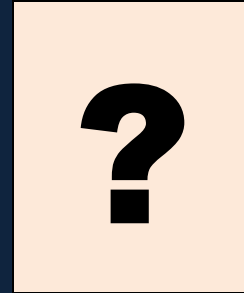
Question 127

Question:

Write a program to check whether the given character is a white-space character or not.

Solution:

```cpp
#include <iostream>
using namespace std;
int main() {
char c;
char ch = ' ';
if(isspace(ch)) {
    cout << "\nNot a white-space character.";
}
else {
    cout << "\nWhite-space character.";
}
return 0;
```

```
}
```

Question 128

Question:

Write a program to illustrate isprint() and iscntrl() functions.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
char ch = 'a';
if(isprint(ch)) {
cout<<ch<<" is printable character."<<endl;
}
else {
cout<<ch<<" is not printable character."<<endl;
}

if(iscntrl(ch)) {
cout<<ch<<" is control character."<<endl;
}
else {
cout<<ch<<" is not control character."<<endl;
}
return (0);
```

```
}                                                    562
```

Question 129

Question:

Write a program to calculate surface area of cube.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int side;
long area;
cout<<"\nEnter the side of cube: ";
cin>>side;
area = 6*side*side;
cout<<"\nThe surface area of cube is: "<< area;
return 0;
}
```

Question 130

Question:

Write a program to subtract 2 numbers without using subtraction operator.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x =6, y=3;
cout<<x+(~y)+1;
return 0;
}
```

Question 131

Question:

Write a program to add 2 numbers without using addition operator.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x =6, y=3;
cout<<x-(~y)-1;
return 0;
}
```

Question 132

Question:

Write a program to multiply a number by 2 without using multiplication operator.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x=2;
cout<< (x<<1);
return 0;
}
```

```cpp
#include<iostream>
#include<cstring>
using namespace std;
int main() {
string x[5] = {"Albert", "John", "Mary", "James"};
x[0] = "Joe";
cout << x[0]; // Output: Joe
return 0;
}
```

Question 134

Question:

Write a program to divide a number by 2 without using division operator.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
```

```cpp
int x=12;
cout<< (x>>1);
return 0;
}
```

Question 135

Question:

Write a program to calculate volume of sphere.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int radius;
float PI = 3.141592;
cout<<"\nEnter the radius of sphere: ";
cin>>radius;
float volume = (4/3)*(PI*radius*radius*radius);
cout<<"\nThe volume of sphere is: "<< volume;
return 0;
}
```

Question 136

Question:

Write a program to calculate volume of ellipsoid.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int r1, r2, r3;
float PI = 3.141592;
cout<<"\nEnter the radius of the ellipsoid of axis 1: ";
cin>>r1;
cout<<"\nEnter the radius of the ellipsoid of axis 2: ";
cin>>r2;
cout<<"\nEnter the radius of the ellipsoid of axis 3: ";
cin>>r3;
float volume = (4/3)*(PI*r1*r2*r3);
cout<<"\nThe volume of ellipsoid is: "<< volume;
return 0;
}
```

```cpp
#include<iostream>
#include<cstring>
using namespace std;
int main() {
string x[5] = {"Albert", "John", "Mary", "James", "Bob"};
for(int i = 0; i < 5; i++) { cout << x[i] << endl; }
return 0;
}
```

Question 137

Question:

Write a program that uses a for loop to determine power of a number entered by the user.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int x, y;
long power = 1;
cout<<"\nEnter the value for x: ";
cin>>x;
cout<<"\nEnter the value for y: ";
cin>>y;
for(int i=1; i<=y; i++) {
power = power * x;
}
cout<<x<<"^"<<y<<" = "<<power;
return 0;
}
```

```cpp
#include<iostream>
#include<cstring>
using namespace std;

int main() {
string x = "Albert";
string &y = x;
cout << x << endl;
// Output: Albert
cout << y << endl;
// Output: Albert
return 0;
}
```

Question 138

Question:

Write a program to read three numbers and find average of numbers.

Solution:

```cpp
#include<iostream>
using namespace std;
int main() {
int a,b,c;
float avg;
cout<<"\nEnter the first number: ";
cin>>a;
cout<<"\nEnter the second number: ";
cin>>b;
cout<<"\nEnter the third number: ";
cin>>c;
avg=(a+b+c)/3.0;
cout<<"\nAverage of three numbers is: "<< avg;
return 0;
}
```

Question 139

Question:

Write a program to read integer "n" and print first three powers (n^1, n^2, n^3).

Solution:

```cpp
#include<iostream>
#include<cmath>
using namespace std;
int main() {
int n;
cout<<"\nEnter a number: ";
cin>>n;
cout<<pow(n, 1)<<" "<< pow(n, 2)<<" "<< pow(n, 3);
return 0;
}
```

```cpp
#include<iostream>

using namespace std;

int main() {

string i[2][4] = {

    { "A", "L", "B", "E" },

    { "R", "T", "J", "H" }

  };

cout << i[0][2];

// Output: B

return 0;

}
```

```cpp
#include<iostream>

#include<cstring>

using namespace std;


int main() {

string x = "Albert";

cout << &x;

return 0;

}
```

?

```cpp
#include<iostream>

using namespace std;

int main() {

string x="C++ ";

string y="programming";

x.append(y);

cout<<" \n "<<x<<'\n';

return 0;

// Output: C++ programming

}
```

```cpp
#include<iostream>

using namespace std;

int main() {

string x="Albert";

cout<<*x.begin();

return 0;

// Output: A

}
```

```cpp
#include<iostream>

using namespace std;

int main() {

string x ="C language";

*x.begin()='J';

cout<<x;

return 0;

// Output: J language

}
```

```cpp
#include<iostream>

using namespace std;

int main() {

string x="This is a C++ Program.";

x.erase(8,1);

cout<<x;

// Output: This is  C++ Program.

return 0;

}
```

```cpp
#include<iostream>

#include<cstring>

using namespace std;

int main() {

string txt = "C++ Programming.";

cout << "The length of the text string is: " << txt.size();

// Output: The length of the text string is: 16

return 0;

}
```

```cpp
#include<iostream>

#include<cstring>

using namespace std;

int main() {

string txt = "C++ Programming.";

cout << "The length of the text string is: " << txt.length();

// Output: The length of the text string is: 16

return 0;

}
```

```cpp
#include<iostream>

using namespace std;

int main() {

string x="c++ programming";

x.erase(x.begin()+0);

cout<<x;

// Output: ++ programming

return 0;

}
```

Information visualization refers to the visual representation of data or information to facilitate the understanding of complex concepts, patterns, and relationships. It involves using visual elements such as charts, graphs, maps, and other visual representations to convey information and insights.

The main goal of **information visualization** is to make data and information more accessible, understandable, and usable for a wide range of users. By presenting information in a visually compelling and interactive way, information visualization can help users to identify trends, patterns, and insights that may not be apparent in traditional data formats.

Information visualization is used in a wide range of applications, including business intelligence, scientific research, healthcare, education, and more. It plays an important role in helping individuals and organizations make informed decisions based on data-driven insights. With the increasing availability of big data and advancements in data visualization tools and techniques, information visualization is becoming an increasingly important field for data scientists, designers, and other professionals who work with data.

C++ is an extension of C programming and the programs written in C language can run in C++ compilers.

C++ Program:

```cpp
#include<bits/stdc++.h>
using namespace std;

int main()
{
    float a =2.33333;
    cout << floor(a) << endl;
    cout << ceil(a) << endl;
    cout << trunc(a) << endl;
    cout << round(a) << endl;
    cout << setprecision(2) << a;
    return 0;
}
```

Output:

2

3

2

2

2.3

C++ Program:

```cpp
#include <cstdlib>
#include <iostream>
using namespace std;

int main()
{
    float a = -43;
    cout << abs(a) << endl;
    cout << labs(a) << endl;
    cout << llabs(a) << endl;
    return 0;
}
```

Output:

43

43

43

JDK	JRE	JVM
Java Development Kit	Java Runtime Environment	Java Virtual Machine
It is the tool necessary to compile, document and package Java programs.	It provides the class libraries and other resources that a specific Java program needs to run.	A virtual machine that enables a computer to run Java programs

Inheritance

↓

A mechanism in which one class acquires the properties of another

Abstraction

↓
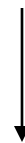
The methodology of hiding the implementation details from the user and only providing the functionality to the users.

Encapsulation

↓

A process of wrapping code and data together into a single unit

Polymorphism

↓
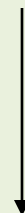
The ability of any data to be processed in more than one form

Collection is a framework that is designed to store the objects and manipulate the design to store the objects.

Java Exercises

Java is the mainstream technology for creating and delivering embedded and mobile software, games, Internet entertainment, and business software. It serves as the foundation for almost any sort of networked application. With over 9 million developers across the globe, Java makes it simple to quickly create, distribute, and use new apps and services. Around 1992, **James Gosling** was employed by Sun Labs. A set-top box was being built by Gosling and his team, who began by "cleaning up" C++ and ultimately came up with a new language and runtime. As a result, Java or Oak was created. C continues to be the top option among developers when it comes to programming languages. However, Java is more popular among developers than C. The second most popular programming language is Java. The common reason for this is because it facilitates the creation of sophisticated applications that operate well and satisfactorily. In addition, Java may be installed and operated on any platform. Approximately 3 billion mobile phones, 125 million TV sets, and every Blu-Ray player currently use Java. You can learn Java programming and advance your skills by practicing and working through problems. If you learn best "by example," this is the chapter for you.

Question 1

Question:

Write a program to print Hello, World!.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
System.out.print("Hello, World!");
}
}
```

Question 2

Question:

Write a program to compute the perimeter and area of a rectangle.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int height = 8;
int width = 5;
```

```
int perimeter = 2 * (height + width);
System.out.println("Perimeter of the rectangle is: " + perimeter + " cm");
int area = height * width;
System.out.println("Area of the rectangle is: " + area + " square cm");
}
}
```

Question 3

Question:

Write a program to compute the perimeter and area of a circle.

Solution:

```
public class MyClass {
public static void main(String[] args) {
int radius = 4;
float perimeter = (float)(2 * 3.14 * radius);
System.out.printf("Perimeter of the circle is: %f cm\n", perimeter);
float area = (float)(3.14 * radius * radius);
System.out.printf("Area of the circle is: %f square cm\n", area);
}
}
```

```
public class MyClass {
public static void main(String[] args) {
int x = 65;
x = 80;
System.out.println(x);          // Output: 80
}
}
```

Question 4

Question:

Write a program that accepts two numbers from the user and calculate the sum of the two numbers.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int a, b, sum;
System.out.print("\nEnter the first number: ");
a = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the second number: ");
b = STDIN_SCANNER.nextInt();
sum = a + b;
System.out.print("\nSum of the above two numbers is: " + sum);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

```java
public class MyClass {
public static void main(String[] args) {
String x = "Einstein";
System.out.println("Albert " + x);
}
                          // Output: Albert Einstein
}
```

Question 5

Question:

Write a program that accepts two numbers from the user and calculate the product of the two numbers.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int a, b, mult;
System.out.print("\nEnter the first number: ");
a = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the second number: ");
b = STDIN_SCANNER.nextInt();
mult = a * b;
System.out.print("\nProduct of the above two numbers is: " + mult);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 6

Question:

Write a program that accepts three numbers and find the largest of three.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x, y, z;
System.out.print("\nEnter the first number: ");
x = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the second number: ");
y = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the third number: ");
z = STDIN_SCANNER.nextInt();

// if x is greater than both y and z, x is the largest
if(x >= y && x >= z) {
System.out.print("\n" + x + " is the largest number.");
}

// if y is greater than both x and z, y is the largest
if(y >= x && y >= z) {
System.out.print("\n" + y + " is the largest number.");
}

// if z is greater than both x and y, z is the largest
if(z >= x && z >= y) {
System.out.print("\n" + z + " is the largest number.");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question:

Write a program that reads three floating values and check if it is possible to make a triangle with them. Also calculate the perimeter of the triangle if the entered values are valid.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
float x, y, z;
System.out.print("\nEnter the first number: ");
x = STDIN_SCANNER.nextFloat();
System.out.print("\nEnter the second number: ");
y = STDIN_SCANNER.nextFloat();
System.out.print("\nEnter the third number: ");
z = STDIN_SCANNER.nextFloat();

if(x < y + z && y < x + z && z < y + x) {
System.out.printf("\nPerimeter of the triangle is: %f\n", x + y + z);
} else {
System.out.print("\nIt is impossible to form a triangle.");
```

```
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 8

Question:

Write a program that reads an integer between 1 and 7 and print the day of the week in English.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int day;
System.out.print("\nEnter a number between 1 to 7 to get the day name: ");
day = STDIN_SCANNER.nextInt();
switch(day) {
case 1:
    System.out.println("Monday");
    break;
case 2:
    System.out.println("Tuesday");
```

582

```java
        break;
case 3:
    System.out.println("Wednesday");
    break;
case 4:
    System.out.println("Thursday");
    break;
case 5:
    System.out.println("Friday");
    break;
case 6:
    System.out.println("Saturday");
    break;
case 7:
    System.out.println("Sunday");
    break;
default:
    System.out.print("Enter a number between 1 to 7.");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 9

Question:

Write a program to find the sum of two numbers.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int a, b, sum;
a = 1;
b = 2;
sum = a + b;
System.out.print("The sum of a and b = " + sum);
}
}
```

Question 10

Question:

Write a program to find the square of a number.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int a, b;
a = 2;
b = (int)Math.pow(a, 2);
System.out.print("The square of a = " + b);
}
}
```

Question 11

Question:

Write a program to find the greatest of two numbers.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int a, b;
a = 2;
b = 3;
if(a > b) {
System.out.print("a is greater than b");
} else {
System.out.print("b is greater than a");
}
}
}
```

Question 12

Question:

Write a program to print the average of the elements in the array.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int avg = 0, sum = 0;
int[] num = {16, 18, 20, 25, 36};
for(int i = 0; i < 5; i++) {
sum = sum + num[i];
avg = sum / 5;
}
System.out.println("Sum of the Elements in the array is: " + sum);
System.out.println("Average of the elements in the array is: " + avg);
}
}
```

Question 13

Question:

Write a program that prints all even numbers between 1 and 25.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
System.out.println("Even numbers between 1 to 25:");
for(int i = 1; i <= 25; i++) {
```

```
if(i % 2 == 0) {
System.out.print(i + " ");
}
}
}
}
```

Question 14

Question:

Write a program that prints all odd numbers between 1 and 50.

Solution:

```
public class MyClass {
public static void main(String[] args) {
System.out.println("Odd numbers between 1 to 50:");
for(int i = 1; i <= 50; i++) {
if(i % 2 != 0) {
System.out.print(i + " ");
}
}
}
}
```

```
public class MyClass {
public static void main(String[] args) {
byte x = 100;
System.out.println(x);
}                          // Output:  100
}
```

Question 15

Question:

Write a program to print the first 10 numbers starting from one together with their squares and cubes.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
for(int i = 1; i <= 10; i++) {
System.out.println("Number = " + i + " its square = " + (i * i) + " its cube
= " + (i * i * i));
}
}
}
```

Question 16

Question:

Write a program:

If you enter a character M

Output must be: ch = M.

Solution:

```java
public class MyClass {
public static void main(String[] args) throws Exception {
char c;
System.out.print("Enter a character: ");
c = (char)System.in.read();
System.out.println("ch = " + c);
}
}
```

Question 17

Question:

Write a program to print the multiplication table of a number entered by the user.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int n;
System.out.print("Enter any number: ");
n = STDIN_SCANNER.nextInt();
for(int i = 1; i <= 5; i++) {
System.out.println(n + " * " + i + " = " + (n * i));
}
```

```
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 18

Question:

Write a program to print the product of the first 10 digits.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int product = 1;
for(int i = 1; i <= 10; i++) {
product = product * i;
}
System.out.print("The product of the first 10 digits is: " + product);
}
}
```

Question 19

Question:

Write a program to print whether the given number is positive or negative.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int a;
a = -35;
if(a > 0) {
System.out.print("Number is positive");
} else {
System.out.print("Number is negative");
}
}
}
```

Question 20

Question:

Write a program to check the equivalence of two numbers entered by the user.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x, y;
```

```java
System.out.print("\nEnter the first number: ");
x = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the second number: ");
y = STDIN_SCANNER.nextInt();
if(x - y == 0) {
System.out.print("\nThe two numbers are equivalent");
} else {
System.out.print("\nThe two numbers are not equivalent");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 21

Question:

Write a program to print the remainder of two numbers entered by the user.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int a, b, c;
System.out.print("\nEnter the first number: ");
a = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the second number: ");
```

```
b = STDIN_SCANNER.nextInt();
c = a % b;
System.out.print("\n The remainder of " + a + " and " + b + " is: " + c);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 22

Question:

Write a program to print the characters from A to Z.

Solution:

```
public class MyClass {
public static void main(String[] args) {
for(byte i = 'A'; i <= 'Z'; i++) {
System.out.println((char)Byte.toUnsignedInt(i));
}
}
}
```

```
public class MyClass {
public static void main(String[] args) {
boolean x = true;
boolean y = false;
System.out.println(x); // Output: true
System.out.println(y); // Output: false
}
}
```

Question 23

Question:

Write a program to print the length of the entered string.

___

Solution:

```java
import java.util.Scanner;
public class MyClass {
public static void main(String[] args) {
String a;
Scanner scan = new Scanner(System.in);
System.out.print("Enter Your Name : ");
a = scan.nextLine();
System.out.println("The length of the String is: " + a.length());
}
}
```

___

Question 24

Question:

Write a program to check whether the given character is a lower case letter or not.

___

Solution:

```java
public class MyClass {
public static void main(String[] args) {
char ch = 'a';
if(Character.isLowerCase(ch)) {
System.out.println("The given character is a lower case letter");
}
else {
System.out.println("The given character is a upper case letter");
}
}
}
```

Question 25

Question:

Write a program to check whether the given character is a upper case letter or not.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
char ch = 'A';
if(Character.isUpperCase(ch)) {
System.out.println("The given character is a upper case letter");
}
else {
System.out.println("The given character is a lower case letter");
}
```

```
}
}
```

Question 26

Question:

Write a program to convert the lower case string to upper case string.

Solution:

```
public class MyClass {
public static void main(String[] args) {
String a = "albert einstein";
System.out.println(a.toUpperCase());
}
}
```

Question 27

Question:

Write a program that takes a distance in centimeters and outputs the corresponding value in inches.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public final static double X = 2.54;
public static void main(String[] args) {
double inch, cm;
System.out.print("Enter the distance in cm: ");
cm = STDIN_SCANNER.nextDouble();
inch = cm / X;
System.out.printf("\nDistance of %.2f cms is equal to %.2f inches", cm,
inch);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 28

Question:

Write a program to print the output:

Einstein [0] = E

Einstein [1] = I

Einstein [2] = N

Einstein [3] = S

Einstein [4] = T

Einstein [5] = E

Einstein [6] = I

Einstein [7] = N

Solution:

```java
public class MyClass {
public static void main(String[] args) throws Exception{
int i;
char [] num = {'E' , 'I', 'N', 'S', 'T', 'E', 'I', 'N'};
for(i=0; i<8; i++)
System.out.println("Einstein [" + i + " ] = " + num[i]);
}
}
```

Question 29

Question:

Write a program to print "Hello World" 10 times.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
for(int i = 1; i <= 10; i++) {
System.out.println("Hello World ");
}
}
}
```

```java
public class MyClass {

static void myMethod(String x) {

System.out.println(x + " Einstein");

}


public static void main(String[] args) {

myMethod("David");

myMethod("Albert");

myMethod("Elsa");

}

}
```

?

Question 30

Question:

Write a program to print first 5 numbers using do while loop statement.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int i = 1;
do {
System.out.println(i++);
} while(i <= 5);
}
}
```

Question 31

Question:

Write a program to check whether a character is an alphabet or not.

Solution:

```java
import java.util.Scanner;

public class Main {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.println("Enter any caracter: ");
char c = scanner.next().charAt(0);
if((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {
System.out.println(c + " is a Alphabet.");
} else {
System.out.println(c + " is not a Alphabet.");
}
}
}
```

Question 32

Question:

Write a program to check whether a entered number is even or odd.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int a;
System.out.print("Enter any number: ");
```

```
a = STDIN_SCANNER.nextInt();
if(a % 2 == 0) {
System.out.print("The entered number is even");
} else {
System.out.print("The entered number is odd");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 33

Question:

Write a program to print the ASCII value of the given character.

Solution:

```
public class MyClass {
public static void main(String[] args) {
byte ch = 'A';
System.out.print("The ASCII value of " + ((char)Byte.toUnsignedInt(ch)) + "
is: " + ch);
}
}
```

Question 34

Question:

Write a program that will print all numbers between 1 to 50 which divided by a specified number and the remainder will be 2.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x;
System.out.print("Enter a number: ");
x = STDIN_SCANNER.nextInt();
for(int i = 1; i <= 50; i++) {
if(i % x == 2) {
System.out.println(i);
}
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Pointers are not used in Java because doing so would weaken the language's security and robustness and make it more complicated.

Question 35

Question:

Write a program to determine whether two numbers in a pair are in ascending or descending order.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int a, b;
System.out.print("\nEnter a pair of numbers (for example 22,12 | 12,22): ");
System.out.print("\nEnter the first number: ");
a = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the second number: ");
b = STDIN_SCANNER.nextInt();
if(a > b) {
System.out.print("\nThe two numbers in a pair are in descending order.");
} else {
System.out.print("\nThe two numbers in a pair are in ascending order.");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 36

Question:

Write a program that reads two numbers and divides one by the other. Specify "Division not possible" if that is not possible.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int a, b;
float c;
System.out.print("\nEnter the first number: ");
a = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the second number: ");
b = STDIN_SCANNER.nextInt();
if(b != 0) {
c = (float)a / (float)b;
System.out.printf("\n%d/%d = %.1f", a, b, c);
} else {
System.out.println("\nDivision not possible.");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 37

Question:

Write a program that will print all numbers between 1 to 50 which divided by a specified number and the remainder is equal to 2 or 3.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x;
System.out.print("Enter a number: ");
x = STDIN_SCANNER.nextInt();
for(int i = 1; i <= 50; i++) {
if(i % x == 2 || i % x == 3) {
System.out.println(i);
}
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

```java
public class MyClass {
public static void main(String[] args) {
double y = 9.78d;
int x = (int) y;
System.out.println(x); // Output: 9
}
}
```

Question 38

Question:

Write a program that adds up all numbers between 1 and 100 that are not divisible by 12.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int x = 12, sum = 0;
for(int i = 1; i <= 100; i++) {
if(i % x != 0) {
sum += i;
}
}
System.out.println("\nSum: " + sum);
}
}
```

Question 39

Question:

Write a program to calculate the value of x where x = 1 + 1/2 + 1/3 + ... + 1/50.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
float x = 0;
for(int i = 1; i <= 50; i++) {
x += (float)1 / i;
}
System.out.printf("Value of x: %.2f\n", x);
}
}
```

Question 40

Question:

Write a program that reads a number and find all its divisor.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x;
System.out.print("\nEnter a number: ");
x = STDIN_SCANNER.nextInt();
System.out.print("All the divisor of " + x + " are: ");
for(int i = 1; i <= x; i++) {
if(x % i == 0) {
System.out.print("\n" + i);
```

```
}
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 41

Question:

Write a program to find the incremented and decremented values of two numbers.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int a, b, c, d, e, f;
a = 10;
b = 12;
c = a + 1;
d = b + 1;
e = a - 1;
f = b - 1;
System.out.print("\nThe incremented value of a =" + c);
System.out.print("\nThe incremented value of b =" + d);
System.out.print("\nThe decremented value of a =" + e);
System.out.print("\nThe decremented value of b =" + f);
}
}
```

Question 42

Question:

Write a program to find square of a entered number using functions.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int answer;
answer = square();
System.out.print("The square of the entered number is: " + answer);
}

public static int square() {
int x;
System.out.print("Enter any number: ");
x = STDIN_SCANNER.nextInt();
return x * x;
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 43

Question:

Write a program that accepts principal amount, rate of interest, time and compute the simple interest.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int p, r, t, SI;
System.out.print("\nEnter the principal amount: ");
p = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the rate of interest: ");
r = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the time: ");
t = STDIN_SCANNER.nextInt();
SI = (p * r * t) / 100;
System.out.print("\nSimple interest is: " + SI);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 44

Question:

Write a program that swaps two numbers without using third variable.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int a, b;
System.out.print("\nEnter the value for a: ");
a = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the value for b: ");
b = STDIN_SCANNER.nextInt();
System.out.print("\nBefore swapping: " + a + " " + b);
a = a + b;
b = a - b;
a = a - b;
System.out.print("\nAfter swapping: " + a + " " + b);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 45

Question:

Write a program to compute the area of a hexagon.

```java
import java.util.Scanner;
public class MyClass {
public static void main(String[] args) {
Scanner input = new Scanner(System.in);
System.out.print("Enter the length of a side of the hexagon: ");
double s = input.nextDouble();
double area = (6*(s*s))/(4*Math.tan(Math.PI/6));
System.out.print("The area of the hexagon is: " + area);
}
}
```

Question 46

Question:

Write a program to print the output:

body [b] = b

body [o] = o

body [d] = d

body [y] = y

Solution:

```
public class MyClass {
public static void main(String[] args) throws Exception{
int i;
char [] body = {'b', 'o', 'd', 'y'};
for(i=0; i<4; i++) {
System.out.println("body [" + body [i] + " ] = " + body [i]);
}
}
}
```

Question 47

Question:

Write a program to calculate the discounted price and the total price after discount

Given:

If purchase value is greater than 1000, 10% discount

If purchase value is greater than 5000, 20% discount

If purchase value is greater than 10000, 30% discount.

Solution:

```java
import java.util.Scanner;
public class MyClass {
public static void main(String[] args) {
double pv;
System.out.print("Enter purchased value: ");
pv = STDIN_SCANNER.nextDouble();
if(pv > 1000) {
System.out.printf("\n Discount = %f", pv * 0.1);
System.out.printf("\n Total = %f", pv - pv * 0.1);
} else if(pv > 5000) {
System.out.printf("\n Discount = %f", pv * 0.2);
System.out.printf("\n Total = %f", pv - pv * 0.2);
} else {
System.out.printf("\n Discount = %f", pv * 0.3);
System.out.printf("\n Total = %f", pv - pv * 0.3);
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 48

Question:

Write a program to print the first ten natural numbers using while loop statement.

Solution:

```java
public class MyClass {
```

```
public static void main(String[] args) {

int i = 1;

while(i <= 10) {

System.out.println(i++);

}

}

}
```

```
import java.util.ArrayList;

public class MyClass {

public static void main(String[] args) {

ArrayList<String> x = new ArrayList<String>();

x.add("Albert");

x.add("Joe");

x.add("Alan");

x.add("Mary");

System.out.println(x);

}

}
```

Question 49

Question:

Write a program to shift inputted data by two bits to the left.

Solution:

```
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x;
System.out.print("Enter the integer from keyboard: ");
x = STDIN_SCANNER.nextInt();
System.out.print("\nEntered value: " + x + " ");
System.out.print("\nThe left shifted data is: " + (x <<= 2) + " ");
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 50

Question:

Write a program to shift inputted data by two bits to the Right.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x;
System.out.print("Enter the integer from keyboard: ");
x = STDIN_SCANNER.nextInt();
System.out.print("\nEntered value: " + x + " ");
System.out.print("\nThe right shifted data is: " + (x >>= 2) + " ");
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 51

Question:

Write a program to calculate the exact difference between x and 21. Return three times the absolute difference if x is greater than 21.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x;
System.out.print("Enter the value for x: ");
x = STDIN_SCANNER.nextInt();
if(x <= 21) {
System.out.print(Math.abs(x - 21));
} else if(x >= 21) {
System.out.print(Math.abs(x - 21) * 3);
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 52

Question:

Write a program that reads in two numbers and determine whether the first number is a multiple of the second number.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x, y;
System.out.print("\nEnter the first number: ");
x = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the second number: ");
y = STDIN_SCANNER.nextInt();
if(x % y == 0) {
System.out.println("\n" + x + " is a multiple of " + y + ".");
} else {
System.out.println("\n" + x + " is not a multiple of " + y + ".");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 53

Question:

Write a program to display the system time.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
```

```
System.out.format("\nCurrent Date time: %tc%n\n",
System.currentTimeMillis());
}
}
```

Question 54

Question:

Write a program to convert Celsius into Fahrenheit.

Solution:

```
public class MyClass {
public static void main(String[] args) {
float fahrenheit, celsius;
celsius = 36;
fahrenheit = (celsius * 9) / 5 + 32;
System.out.printf("\nTemperature in fahrenheit is:  %f", fahrenheit);
}
}
```

```
public class MyClass {
public static void main(String[] args) {
int y = 9;
double x = y;
System.out.println(x); // Output: 9.0
}
}
```

Question 55

Question:

Write a program that will examine two inputted integers and return true if either of them is 50 or if their sum is 50.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x, y;
System.out.print("\nEnter the value for x: ");
x = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the value for y: ");
y = STDIN_SCANNER.nextInt();
if(x == 50 || y == 50 || x + y == 50) {
System.out.print("\nTrue");
} else {
System.out.print("\nFalse");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 56

Question:

Write a program that counts the even, odd, positive, and negative values among eighteen integer inputs.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x, even = 0, odd = 0, positive = 0, negative = 0;
System.out.println("\nPlease enter 18 numbers:");
for(int i = 0; i < 18; i++) {
x = STDIN_SCANNER.nextInt();
if(x > 0) {
    positive++;
}
if(x < 0) {
    negative++;
}
if(x % 2 == 0) {
    even++;
}
if(x % 2 != 0) {
    odd++;
}
}
System.out.print("\nNumber of even values: " + even);
```

```
System.out.print("\nNumber of odd values: " + odd);

System.out.print("\nNumber of positive values: " + positive);

System.out.print("\nNumber of negative values: " + negative);

}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);

}
```

Question 57

Question:

Write a program to check whether the person is a senior citizen or not.

Solution:

```
import java.util.Scanner;


public class MyClass {
public static void main(String[] args) {
int age;
System.out.print("Enter age: ");
age = STDIN_SCANNER.nextInt();
if(age >= 60) {
System.out.print("Senior citizen");
} else {
System.out.print("Not a senior citizen");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
```

```
import java.util.ArrayList;

public class MyClass {

public static void main(String[] args) {

ArrayList<String> x = new ArrayList<String>();

x.add("Albert");

x.add("Joe");

x.add("Alan");

x.add("Mary");

System.out.println(x.get(0));

// Output: Albert

}

}
```

622

```
}
```

Question 58

Question:

Write a program that reads a student's three subject scores (0-100) and computes the average of those scores.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
float score, totalScore = 0;
int subject = 0;
System.out.println("Enter three subject scores (0-100):");
while(subject != 3) {
score = STDIN_SCANNER.nextFloat();
if(score < 0 || score > 100) {
System.out.println("Please enter a valid score.");
} else {
totalScore += score;
subject++;
}
}
System.out.printf("Average score = %.2f\n", totalScore / 3);
}
```

```
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 59

Question:

What results would the following programs produce?

```java
public class MyClass {
public static void main(String[] args) {
for(int i = 1; i <= 5; i++) {
    if(i == 3) {
        break;
    }
System.out.println(i);
}
}
}
```

```java
import java.util.ArrayList;

public class MyClass {

public static void main(String[] args) {

ArrayList<String> x = new ArrayList<String>();

x.add("Albert");

x.add("Joe");

x.add("Alan");

x.add("Mary");

x.clear();

System.out.println(x);

// Output: []

}

}
```

Solution:

1
2

```
public class MyClass {
public static void main(String[] args) {
System.out.println(-7 + 9 * 5);
System.out.println((68+7) % 8);
System.out.println(50 + -6*5 / 5);
System.out.println(6 + 25 / 3 * 6 - 8 % 2);
}
}
```

Solution:

```
38
3
44
54
```

```
public class MyClass {
public static void main(String[] args) {
for(;;) {
System.out.println("This loop will run forever.");
}
}
}
```

Solution:

```
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever.
This loop will run forever. .........
```

```
public class MyClass {
public static void main(String[] args) {
System.out.println((35.5 * 3.7 - 3.6 * 7.5) / (60.8 - 8.9));
}
}
```

Solution:

```
2.010597302504817
```

```
public class MyClass {
public static void main(String[] args) {
System.out.println("linux");
System.exit(0);
System.out.println("php");
}
}
```

Solution:

```
linux
```

```java
public class MyClass {
public static void main(String[] args) {
for(int i = 1; i <= 5; i++) {
    if(i == 3) {
        continue;
    }
System.out.print(i + "\n ");
}
}
}
```

```java
public class MyClass {
// Create a myfunc() method with an integer parameter called x
static void myfunc(int x) {
// If x is less than 18, print "Access denied"
if(x < 18) {
System.out.println("Access denied");
// If x is greater than, or equal to, 18, print "Access granted"
} else {
System.out.println("Access granted");
}
}
public static void main(String[] args) {
myfunc(25); // Call the myfunc method and pass along an age of 25
}
}
```

Solution:

```
1
2
4
5
```

?

```java
public class MyClass {
public static void main(String[] args) {
int a = 10, b = 20, c;
c = a < b ? a : b;
System.out.print(c);
```

```
}
}
```

Solution:

```
10
```

```
public class MyClass {
public final static int A = 15;
public static void main(String[] args) {
int x;
x = A;
System.out.print(x);
}
}
```

Solution:

```
15
```

```
public class MyClass {
```

```
import java.util.ArrayList;
public class MyClass {
public static void main(String[] args) {
ArrayList<String> x = new ArrayList<String>();
x.add("Albert");
x.add("Joe");
x.add("Alan");
x.add("Mary");
for(int i = 0; i < x.size(); i++) {
    System.out.println(x.get(i));
}
}
}
```

?

```java
public static void main(String[] args) {
for(int i = 1; i <= 3; i++) {
System.out.print((i & 1) != 0 ? "odd\n" : "even\n");
}
System.exit(0);
}
}
```

Solution:

```
odd
even
odd
```

```java
public class MyClass {
public static void main(String[] args) {
double a, b;
a = -2.5;
b = Math.abs(a);
System.out.printf("|%.2f| = %.2f\n", a, b);
}
}
```

Solution:

```
|-2.50| = 2.50
```

```java
public class MyClass {
public static void main(String[] args) {
int x = 12, y = 3;
System.out.println(Math.abs(-x - y));
}
}
```

Solution:

```
15
```

```java
public class MyClass {
public static void main(String[] args) {
int x = 12, y = 3;
System.out.println(-(-x - y));
}
}
```

```java
import java.util.ArrayList;

import java.util.Iterator;


public class MyClass {

public static void main(String[] args) {

ArrayList<String> x = new ArrayList<String>();

x.add("Albert");

x.add("John");

x.add("James");

x.add("Mary");


Iterator<String> it = x.iterator();

System.out.println(it.next());

// Output: Albert

}

}
```

Solution:

15

```java
public class MyClass {
public static void main(String[] args) {
int x = 12, y = 3;
System.out.println(x - -y);
}
}
```

Solution:

15

```java
public class MyClass {
  static void myMethod() {
    System.out.println("Anyone who has never made a mistake has never tried anything new.");
  }

public static void main(String[] args) {
    myMethod();
    myMethod();
    myMethod();
}
}
```

Solution:

```
Anyone who has never made a mistake has never tried anything new.
Anyone who has never made a mistake has never tried anything new.
Anyone who has never made a mistake has never tried anything new.
```

Question 60

Question:

Write a program to find the size of an array.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int[] num = {11, 22, 33, 44, 55, 66};
int n = (int)num.length;
System.out.println("Size of the array is: " + n);
}
}
```

Question 61

Question:

Write a program that prints a sequence from 1 to a given integer, inserts a plus sign between these numbers, and then removes the plus sign at the end of the sequence.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x, i;
System.out.println("\nEnter a integer: ");
x = STDIN_SCANNER.nextInt();
if(x > 0) {
System.out.println("Sequence from 1 to " + x + ":");
    for(i = 1; i < x; i++) {
        System.out.print(i + "+");
        }
System.out.println(i);
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 62

Question:

Write a program to verify whether a triangle's three sides form a right angled triangle or not.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int a, b, c;
System.out.println("Enter the three sides of a triangle: ");
a = STDIN_SCANNER.nextInt();
b = STDIN_SCANNER.nextInt();
c = STDIN_SCANNER.nextInt();
if(a * a + b * b == c * c || a * a + c * c == b * b || b * b + c * c == a *
a) {
    System.out.println("Triangle's three sides form a right angled
triangle.");
} else {
    System.out.println("Triangle's three sides does not form a right angled
triangle.");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 63

Question:

Write a program that will find the second-largest number among the user's input of three numbers.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int a, b, c;
System.out.print("\nEnter the first number: ");
a = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the second number: ");
b = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the third number: ");
c = STDIN_SCANNER.nextInt();
if(a > b && a > c) {
    if(b > c) {
        System.out.print("\n" + b + " is second largest number among three
numbers");
    } else {
        System.out.print("\n" + c + " is second largest number among three
numbers");
    }
} else if(b > c && b > a) {
    if(c > a) {
```

```
        System.out.print("\n" + c + " is second largest number among three
numbers");
    } else {
                System.out.print("\n" + a + " is second largest number among
three numbers");
}
} else if(a > b) {
        System.out.print("\n" + a + " is second largest number among three
numbers");
    } else {
        System.out.print("\n" + b + " is second largest number among three
numbers");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 64

Question:

Write a program to calculate the sum of the two given integer values. Return three times
the sum of the two values if they are equal.

Solution:

```
public class MyClass {
public static void main(String[] args) {
System.out.print(myfunc(3, 5));
```

```java
System.out.print("\n" + myfunc(6, 6));
}
public static int myfunc(int a, int b) {
return a == b ? (a + b) * 3 : a + b;
}
}
```

Question 65

Question:

Write a program that accepts minutes as input, and display the total number of hours and minutes.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int mins, hrs;
System.out.print("Input minutes: ");
mins = STDIN_SCANNER.nextInt();
hrs = mins / 60;
mins = mins % 60;
System.out.println("\n" + hrs + " Hours, " + mins + " Minutes.");
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 66

Question:

Write a program to determine whether a positive number entered by the user is a multiple of three or five.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x;
System.out.print("\nEnter a number: ");
x = STDIN_SCANNER.nextInt();
if(x % 3 == 0 || x % 5 == 0) {
    System.out.print("True");
} else {
    System.out.print("False");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 67

Question:

Write a program to verify whether one of the two entered integers falls within the range of 100 to 200 included.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x, y;
System.out.print("\nEnter the value for x: ");
x = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the value for y: ");
y = STDIN_SCANNER.nextInt();
if(x >= 100 && x <= 200 || y >= 100 && y <= 200) {
    System.out.print("True");
} else {
    System.out.print("False");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 68

Question:

Write a program to determine which of the two given integers is closest to the value 100.

If the two numbers are equal, return 0.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
System.out.print(myfunc(86, 99));
System.out.print("\n" + myfunc(55, 55));
System.out.print("\n" + myfunc(65, 80));
}

public static int myfunc(int a, int b) {
int x = Math.abs(a - 100);
int y = Math.abs(b - 100);
return x == y ? 0 : (x < y ? a : b);
}
}
```

```java
public class MyClass {
public static void main(String[] args) {
int a = 15;
int b = 13;
// returns false because 15 is not equal to 13
System.out.println(a == b);
}
}
```

640

Question 69

Question:

Write a program to determine whether a positive number entered by the user is a multiple of three or five, but not both.

Solution:

```java
import java.util.Scanner;
public class MyClass {
public static void main(String[] args) {
int x;
System.out.print("\nEnter a number: ");
x = STDIN_SCANNER.nextInt();
if(x % 3 == 0 ^ x % 5 == 0) {
    System.out.print("True");
} else {
    System.out.print("False");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

```java
public class MyClass {
public static void main(String[] args) {
int x = 15;
// returns true because 15 is greater than 13 AND 15 is less than 30
System.out.println(x > 13 && x < 30);
}
}
```

Question 70

Question:

Write a program to determine whether two entered non-negative numbers have the same last digit.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x, y;
System.out.print("\nEnter the value for x: ");
x = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the value for y: ");
y = STDIN_SCANNER.nextInt();
if(Math.abs(x % 10) == Math.abs(y % 10)) {
    System.out.print("True");
} else {
    System.out.print("False");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 71

Question:

Write a program to determine whether a given non-negative number is a multiple of 12 or it is one more than a multiple of 12.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int x = 43;
if(x % 12 == 0 || x % 12 == 1) {
    System.out.print("True");
} else {
    System.out.print("False");
}
}
}
```

```java
import java.util.ArrayList;

public class MyClass {

public static void main(String[] args) {

ArrayList<Integer> x = new ArrayList<Integer>();

x.add(30);

x.add(45);

x.add(50);

x.add(65);

for(int i : x) {

   System.out.println(i);

}

}

}
```
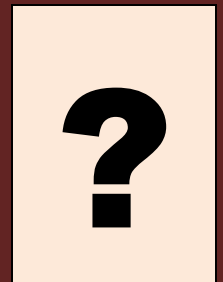
?

Question 72

Question:

Write a program that accepts two integers and returns true when one of them equals 6, or when their sum or difference equals 6.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x, y;
System.out.print("\nEnter the value for x: ");
x = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the value for y: ");
y = STDIN_SCANNER.nextInt();
if(x == 6 || y == 6 || x + y == 6 || Math.abs(x - y) == 6) {
    System.out.print("True");
} else {
    System.out.print("False");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 73

Question:

Write a program to check whether it is possible to add two integers to get the third integer from three entered integers.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x, y, z;
System.out.print("\nEnter the value for x: ");
x = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the value for y: ");
y = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the value for z: ");
z = STDIN_SCANNER.nextInt();
if(x == y + z || y == x + z || z == x + y) {
    System.out.print("True");
} else {
    System.out.print("False");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 74

Question:

Write a program that converts kilometers per hour to miles per hour.

Solution:

```
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
float kmph;
System.out.print("Enter kilometers per hour: ");
kmph = STDIN_SCANNER.nextFloat();
System.out.printf("\n%f miles per hour", kmph * 0.6213712);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 75

Question:

Write a program to calculate area of an ellipse.

Solution:

```
import java.util.Scanner;

public class MyClass {
public final static double PI = 3.141592;
public static void main(String[] args) {
float major, minor;
System.out.print("\nEnter length of major axis: ");
major = STDIN_SCANNER.nextFloat();
System.out.print("\nEnter length of minor axis: ");
```

```
minor = STDIN_SCANNER.nextFloat();

System.out.printf("\nArea of an ellipse = %.4f", PI * major * minor);

}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);

}
```

Question 76

Question:

Write a program to calculate the sum of three given integers. Return the third value if the first two values are equal.

Solution:

```
public class MyClass {
public static void main(String[] args) {
System.out.print("\n" + myfunc(11, 11, 11));

System.out.print("\n" + myfunc(11, 11, 16));

System.out.print("\n" + myfunc(18, 15, 10));

}

public static int myfunc(int a, int b, int c) {
if(a == b && b == c) {
    return 0;
}
if(a == b) {
    return c;
}
```

```java
if(a == c) {

    return b;

}

if(b == c) {

    return a;

} else {

    return a + b + c;

}

}

}
```

```java
import java.util.ArrayList;

public class MyClass {

public static void main(String[] args) {

ArrayList<String> x = new ArrayList<String>();

x.add("Apple");

x.add("Lemon");

x.add("Kiwi");

x.add("Orange");

for(String i : x) {

    System.out.println(i);

}

}

}
```

?

Question 77

Question:

Write a program to convert bytes to kilobytes.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
double bytes;
System.out.print("\nEnter number of bytes: ");
bytes = STDIN_SCANNER.nextDouble();
System.out.printf("\nKilobytes: %.2f", bytes / 1024);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
```

}

Question 78

Question:

Write a program to convert megabytes to kilobytes.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
double megabytes, kilobytes;
System.out.print("\nInput the amount of megabytes to convert: ");
megabytes = STDIN_SCANNER.nextDouble();
kilobytes = megabytes * 1_024;
System.out.printf("\nThere are %f kilobytes in %f megabytes.", kilobytes,
megabytes);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 79

Question:

Write a program to count the number of even elements in an integer array.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int[] array = new int[1000];
int arrSize, even = 0;
System.out.print("Input the size of the array: ");
arrSize = STDIN_SCANNER.nextInt();
System.out.println("Enter the elements in array: ");
for(int i = 0; i < arrSize; i++) {
    array[i] = STDIN_SCANNER.nextInt();
}

for(int i = 0; i < arrSize; i++) {
    if(array[i] % 2 == 0) {
        even++;
    }
}
System.out.print("Number of even elements: " + even);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 80

Question:

Write a program to count the number of odd elements in an integer array.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int[] array = new int[1000];
int arrSize, odd = 0;
System.out.print("Input the size of the array: ");
arrSize = STDIN_SCANNER.nextInt();
System.out.println("Enter the elements in array: ");
for(int i = 0; i < arrSize; i++) {
    array[i] = STDIN_SCANNER.nextInt();
}

for(int i = 0; i < arrSize; i++) {
    if(array[i] % 2 != 0) {
        odd++;
    }
}
System.out.print("Number of odd elements: " + odd);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
```

```
}
```

Question 81

Question:

Write a program that will accept two integers and determine whether or not they are equal.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x, y;
System.out.println("Input the values for x and y: ");
x = STDIN_SCANNER.nextInt();
y = STDIN_SCANNER.nextInt();
if(x == y) {
    System.out.println("x and y are equal");
} else {
    System.out.println("x and y are not equal");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 82

Question:

Write a program to find the third angle of a triangle if two angles are given.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int angle1, angle2;
System.out.print("\nEnter the first angle of the triangle: ");
angle1 = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the second angle of the triangle: ");
angle2 = STDIN_SCANNER.nextInt();
System.out.print("\nThird angle of the triangle is:  " + (180 - (angle1 +
angle2)));
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 83

Question:

Write a program to determine whether a particular year is a leap year or not.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int year;
System.out.print("Enter the year: ");
year = STDIN_SCANNER.nextInt();
if(year % 400 == 0) {
    System.out.print("\n" + year + " is a leap year.");
} else if(year % 100 == 0) {
    System.out.print("\n" + year + " is a not leap year.");
} else if(year % 4 == 0) {
    System.out.print("\n" + year + " is a leap year.");
} else {
    System.out.print("\n" + year + " is not a leap year.");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 84

Question:

Write a program that reads the candidate's age and determine a candidate's eligibility to cast his own vote.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int age;
System.out.print("\nEnter the age of the candidate: ");
age = STDIN_SCANNER.nextInt();
if(age < 18) {
    System.out.print("\nWe apologize, but the candidate is not able to cast
his vote.");
    System.out.print("\nAfter " + (18 - age) + " year, the candidate would be
able to cast his vote.");
} else {
    System.out.println("Congratulation! the candidate is qualified to cast
his vote.");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 85

Question:

Write a program to Convert Yard to Foot.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
float yard;
System.out.print("\nEnter the Length in Yard : ");
yard = STDIN_SCANNER.nextFloat();
System.out.printf("\n%f Yard in Foot is: %f", yard, 3 * yard);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 86

Question:

Write a program to convert gigabytes to megabytes.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
double gigabytes, megabytes;
System.out.print("\nInput the amount of gigabytes to convert: ");
gigabytes = STDIN_SCANNER.nextDouble();
```

```java
megabytes = gigabytes * 1_024;
System.out.printf("\nThere are %f megabytes in %f gigabytes.", megabytes,
gigabytes);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 87

Question:

Write a program to Convert Kilogram to Pounds.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
float kg, lbs;
System.out.print("\nEnter Weight in Kilogram: ");
kg = STDIN_SCANNER.nextFloat();
lbs = (float)(kg * 2.20462);
System.out.printf("\n%f Kg = %f Pounds", kg, lbs);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 88

Question:

Write a program to Convert Kilogram to Ounce.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
float kg, ounce;
System.out.print("\nEnter Weight in Kilogram: ");
kg = STDIN_SCANNER.nextFloat();
ounce = (float)(kg * 35.274);
System.out.printf("\n%f Kg = %f Ounce", kg, ounce);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 89

Question:

Write a program to Convert Pounds to Grams.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
float pound, gram;
System.out.print("\nEnter Weight in Pounds: ");
pound = STDIN_SCANNER.nextFloat();
gram = (float)(pound * 453.592);
System.out.printf("\n%f Pound = %f Grams", pound, gram);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 90

Question:

Write a program to verify whether a triangle is valid or not using angles.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
```

```java
int angle1, angle2, angle3, sum;
System.out.print("\nEnter the first angle of the triangle: ");
angle1 = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the second angle of the triangle: ");
angle2 = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the third angle of the triangle: ");
angle3 = STDIN_SCANNER.nextInt();
sum = angle1 + angle2 + angle3;
if(sum == 180) {
    System.out.print("\nThe triangle is valid.");
} else {
    System.out.print("\nThe triangle is not valid.");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 91

Question:

Write a program to add the digits of a two-digit number that is entered by the user.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
```

```
int x, y, sum = 0;
System.out.print("\nEnter a two-digit number: ");
x = STDIN_SCANNER.nextInt();
y = x;
while(y != 0) {
    sum = sum + y % 10;
    y = y / 10;
}
System.out.print("\nSum of digits of " + x + " is: " + sum);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 92

Question:

Write a program to verify if a character you entered is a vowel or a consonant.

Solution:

```
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.println("Enter a alphabet: ");
char ch = scanner.next().charAt(0);
```

```java
if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ||
ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U' ) {
System.out.println(ch + " is vowel");
}
else {
System.out.println(ch + " is consonant");
}
}
}
```

```java
import java.util.ArrayList;

public class MyClass  {

public static void main(String[] args) {

ArrayList<String> x = new ArrayList<String>();

x.add("Apple");

x.add("Lemon");

x.add("Kiwi");

x.add("Mango");

System.out.println(x.size());

// Output: 4

}

}
```

Question 93

Question:

Write a program to find factorial of a number.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int fact = 1, num;
System.out.print("\nEnter a number: ");
num = STDIN_SCANNER.nextInt();
for(int i = 1; i <= num; i++) {
    fact = fact * i;
```

```
}
System.out.print("\nFactorial of " + num + " is: " + fact);

}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);

}
```

Question 94

Question:

Write a program to print number of days in a month.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int[] x = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int m;
System.out.print("\nEnter the month number: ");
m = STDIN_SCANNER.nextInt();
if(m > 12 || m < 1) {
    System.out.print("Invalid input");
} else if(m == 2) {
    System.out.print("\nNumber of days in month 2 is either 29 or 28");
} else {
    System.out.print("\nNumber of days in month " + m + " is " + x[m - 1]);
}
```

```
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 95

Question:

Write a program to concatenate multiple strings.

Solution:

```
public class MyClass {
public static void main(String[] args) {
String x = "Stephen";
String y = "-William";
String z = "-Hawking";
String c = x.concat(y).concat(z);
System.out.println(c);
}
}
```

```
public class MyClass {

public static void main(String[] args) {

Integer x = 10065;

String y = x.toString();

System.out.println(y.length());

// Output: 5

}

}
```

Question 96

Question:

Write a program to find maximum between two numbers.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int a, b;
System.out.println("Enter two numbers: ");
a = STDIN_SCANNER.nextInt();
b = STDIN_SCANNER.nextInt();
if(a > b) {
    System.out.print("\n" + a + " is a maximum number");
} else {
    System.out.print("\n" + b + " is a maximum number");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
public class MyClass {
public static void main(String args[]) {
double a = 15.143;
double b = 15.656;
System.out.println(Math.max(a, b));
}
}
```

Question 97

Question:

Write a program to compare two strings.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
String x = "Albert";
String y = "Albert";
if(x == y) {
    System.out.println("The 2 strings are equal.");
}
else {
    System.out.println("The 2 strings are not equal.");
}
}
}
```

```java
public class MyClass {
public static void main(String[] args) {
String x = "Albert";
String y = "Albert";
if(x.equals(y)) {
    System.out.println("The 2 strings are equal.");
```

```
    }
else {
    System.out.println("The 2 strings are not equal.");
}
}
}
```

Question 98

Question:

Write a program to convert the upper case string to lower case string.

Solution:

```java
public class MyClass {
public static void main(String args[]) {
String x = new String("ALBERT EINSTEIN");
System.out.println(x.toLowerCase());
}
}
```

```java
public class MyClass {

public static void main(String[] args) {

String a = "20";

int b = 30;

String c = a + b;

System.out.println(c); // Output: 2030

}

}
```

Question 99

Question:

Write a program to find the quotient and remainder of a entered dividend and divisor.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int dividend, divisor;
System.out.print("\nEnter dividend: ");
dividend = STDIN_SCANNER.nextInt();
System.out.print("\nEnter divisor: ");
divisor = STDIN_SCANNER.nextInt();
System.out.println("\nQuotient = " + (dividend / divisor));
System.out.print("\nRemainder = " + (dividend % divisor));
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 100

Question:

Write a program to determine the Size of int, float, double and char.

Solution:

```java
public class MyClass {
public static void main (String[] args) {
System.out.println("Size of int is: " + (Integer.SIZE/8) + " bytes.");
System.out.println("Size of char is: " + (Character.SIZE/8) + " bytes.");
System.out.println("Size of float is: " + (Float.SIZE/8) + " bytes.");
System.out.println("Size of double is: " + (Double.SIZE/8) + " bytes.");
}
}
```

Question 101

Question:

Write a program to promt user for 4 times password check.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
String password = "123";
String inputPass;
Scanner input = new Scanner(System.in);
System.out.println("Enter Your Password: ");
```

```java
inputPass = input.nextLine();
if (inputPass.equals(password)) {
System.out.println("Welcome User!");
}
else {
for(int i = 0; i < 3; i++) {
System.out.println("Enter Your Password:");
inputPass = input.nextLine();
}
System.out.println("Access Denied! Try again");
}
}
}
```

Question 102

Question:

Write a program to find absolute value of a number.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int num;
System.out.println("Input a positive or negative number: ");
```

```java
num = STDIN_SCANNER.nextInt();
System.out.println("\nAbsolute value of |" + num + "| is " + Math.abs(num));
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
public class MyClass {
public static void main(String args[]) {
int x = 820;
int y = -985;
float z = -8.1f;
System.out.printf( "Absolute Value of x: %d \n", Math.abs(x) );
System.out.printf( "Absolute Value of y: %d \n", Math.abs(y) );
System.out.printf( "Absolute Value of z: %f \n", Math.abs(z) );
}
}
```

Question 103

Question:

Write a program that will accept a person's height in cm and classify the person based on it.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
```

```java
float ht;
System.out.print("\nEnter the height (in cm): ");
ht = STDIN_SCANNER.nextFloat();
if(ht < 150.0) {
    System.out.println("Dwarf.");
} else if(ht >= 150.0 && ht < 165.0) {
    System.out.println("Average Height.");
} else if(ht >= 165.0 && ht <= 195.0) {
    System.out.println("Taller.");
} else {
    System.out.println("Abnormal height.");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 104

Question:

Write a program to calculate the area of different geometric shapes using switch statements.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
```

```java
int choice;
float r, l, w, b, h;
System.out.print("\nEnter 1 for area of circle: ");
System.out.print("\nEnter 2 for area of rectangle: ");
System.out.print("\nEnter 3 for area of triangle: ");
System.out.print("\nEnter your choice : ");
choice = STDIN_SCANNER.nextInt();

switch(choice) {
case 1:
System.out.print("Enter the radius of the circle: ");
r = STDIN_SCANNER.nextFloat();
System.out.printf("\nArea of a circle is: %f", 3.14 * r * r);
break;
case 2:
System.out.println("Enter the length and width of the rectangle: ");
l = STDIN_SCANNER.nextFloat();
w = STDIN_SCANNER.nextFloat();
System.out.printf("\nArea of a rectangle is: %f", l * w);
break;
case 3:
System.out.println("Enter the base and height of the triangle: ");
b = STDIN_SCANNER.nextFloat();
h = STDIN_SCANNER.nextFloat();
System.out.printf("\nArea of a triangle is: %f", 0.5 * b * h);
break;
default:
System.out.print("\nPlease enter a number from 1 to 3.");
break;
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
```

```
}
```

Question 105

Question:

Write a program to accept a character from the keyboard and print "Yes" if it is equal to y.
Otherwise print "No".

Solution:

```java
public class MyClass {
public static void main(String[] args) throws Exception {
char ch;
System.out.println("Enter a character: ");
ch = (char)System.in.read();
if(ch == 'y' || ch == 'Y') {
System.out.println("Yes\n");
}
else {
System.out.println("No\n");
}
}
}
```

```java
public class MyClass {
public static void main(String[] args) {
// return a random number between 0 and 1
System.out.println(Math.random());
}
}
```

Question 106

Question:

Write a program that uses bitwise operators to multiply an entered value by four.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
long x, y;
System.out.print("Enter a integer: ");
x = STDIN_SCANNER.nextLong();
y = x;
x = x << 2;
System.out.println(y + " x 4 = " + x);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 107

Question:

Write a program to check whether a number entered by the user is power of 2 or not.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x;
System.out.print("Enter a number: ");
x = STDIN_SCANNER.nextInt();
if(x != 0 && (x & x - 1) == 0) {
    System.out.print("\n" + x + " is a power of 2");
} else {
    System.out.print("\n" + x + " is not a power of 2");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 108

Question:

Write a program to determine whether a triangle is scalene, isosceles, or equilateral.

Solution:

```java
import java.util.Scanner;
```

```java
public class MyClass {
public static void main(String[] args) {
int side1, side2, side3;
System.out.print("\nEnter the first side of the triangle: ");
side1 = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the second side of the triangle: ");
side2 = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the third side of the triangle: ");
side3 = STDIN_SCANNER.nextInt();
if(side1 == side2 && side2 == side3) {
    System.out.print("\nThe given Triangle is equilateral.");
} else if(side1 == side2 || side2 == side3 || side3 == side1) {
    System.out.print("\nThe given Triangle is isosceles.");
} else {
    System.out.print("\nThe given Triangle is scalene.");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 109

Question:

Write a program to print ASCII values of all the letters of the English alphabet from A to Z.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
for(int i = 'A'; i <= 'Z'; i++) {
System.out.println("ASCII value of " + ((char)Byte.toUnsignedInt((byte)i)) +
" = " + i);
}
}
}
```

Question 110

Question:

Write a program to find sum of even numbers between 1 to n.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int num, sum = 0;
System.out.print("Enter a number: ");
num = STDIN_SCANNER.nextInt();
for(int i = 2; i <= num; i = i + 2) {
sum = sum + i;
}
```

```
System.out.print("\nSum of all even number between 1 to " + num + " is: " +
sum);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 111

Question:

Write a program to find sum of odd numbers between 1 to n.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int num, sum = 0;
System.out.print("Enter a number: ");
num = STDIN_SCANNER.nextInt();
for(int i = 1; i <= num; i = i + 2) {
    sum = sum + i;
}
System.out.print("\nSum of all odd number between 1 to " + num + " is: " +
sum);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 112

Question:

Write a program that accepts an integer (x) and computes the value of x+xx+xxx.

Solution:

```java
import java.util.Scanner;
public class MyClass {
public static void main(String[] args) {
int x;
Scanner in = new Scanner(System.in);
System.out.print("Enter a number: ");
x = in .nextInt();
System.out.printf("%d + %d%d  + %d%d%d\n", x, x, x, x, x, x);
}
}
```

Question 113

Question:

Write a program that allows you to enter the cost price and the selling price of a product and calculate profit or loss.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int cp, sp;
System.out.print("\nInput Cost Price: ");
cp = STDIN_SCANNER.nextInt();
System.out.print("\nInput Selling Price: ");
sp = STDIN_SCANNER.nextInt();
if(sp > cp) {
System.out.print("Profit = " + (sp - cp));
} else if(cp > sp) {
System.out.print("Loss = " + (cp - sp));
} else {
System.out.print("No Profit No Loss.");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 114

Question:

Write a program that display the pattern like a right angle triangle using an asterisk.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int rows;
System.out.print("Input the number of rows: ");
rows = STDIN_SCANNER.nextInt();
for(int x = 1; x <= rows; x++) {
for(int y = 1; y <= x; y++) {
System.out.print("*");
}
System.out.println();
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 115

Question:

Write a program that display the pattern like a right angle triangle using a number.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int rows;
System.out.print("Input the number of rows: ");
rows = STDIN_SCANNER.nextInt();
for(int x = 1; x <= rows; x++) {
for(int y = 1; y <= x; y++) {
System.out.print(y);
}
System.out.println();
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 116

Question:

Write a program to determine the number and sum of all integers between 50 and 100 which are divisible by 2.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
```

```java
int sum = 0;
System.out.println("Numbers between 50 and 100, divisible by 2: ");
for(int x = 51; x < 100; x++) {
if(x % 2 == 0) {
    System.out.printf("%5d", x);
        sum += x;
}
}
System.out.print("\nThe sum: " + sum);
}
}
```

Question 117

Question:

Write a program that uses the function to determine whether an entered number is even or odd.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static int myfunc(int x) {
    return x & 1;
}

public static void main(String[] args) {
```

```java
int x;
System.out.print("Enter any number: ");
x = STDIN_SCANNER.nextInt();
if(myfunc(x) != 0) {
    System.out.print("\nThe number you entered is odd.");
} else {
    System.out.print("\nThe number you entered is even.");
}
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 118

Question:

Write a program to find square root of a entered number.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x;
System.out.print("Enter any number: ");
x = STDIN_SCANNER.nextInt();
System.out.printf("Square root of %d is %.2f", x, Math.sqrt(x));
```

```
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 119

Question:

Write a program to find power of a entered number using library function.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x, y;
System.out.print("\nEnter the value for x: ");
x = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the value for y: ");
y = STDIN_SCANNER.nextInt();
System.out.print("\n" + x + "^" + y + " = " + ((long)Math.pow(x, y)));
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 120

Question:

Write a program to read 10 numbers from the keyboard and find their sum and average.

Solution:

```java
import java.util.Scanner;
public class MyClass {
public static void main(String [] args) {
int N1, N2, N3, N4, N5, N6, N7, N8, N9, N10, sum;
float X;
Scanner scan = new Scanner(System.in);
System.out.println("Enter any ten Numbers: ");
N1 = scan.nextInt();
N2 = scan.nextInt();
N3 = scan.nextInt();
N4 = scan.nextInt();
N5 = scan.nextInt();
N6 = scan.nextInt();
N7 = scan.nextInt();
N8 = scan.nextInt();
N9 = scan.nextInt();
N10 = scan.nextInt();
sum = N1 + N2 + N3 + N4 + N5 + N6 + N7 + N8 + N9 + N10;
X = sum /10;
System.out.println("The sum of 10 numbers = " + sum);
System.out.println("The average of 10 numbers = " + X);
}
}
```

Question 121

Question:

Write a program to determine whether the given character is an alphanumeric character or not.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
String x="abc123", y="abc.com";
System.out.println(x.matches("[a-zA-Z0-9]+"));
System.out.println(y.matches("[a-zA-Z0-9]+"));
}
}
```

Question 122

Question:

Write a program to illustrate try-catch statement.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
try {
    int[] num = {1, 2, 3};
    System.out.println(num[3]);
  } catch (Exception e) {
    System.out.println("Something went wrong.");
}
}
}
```

Question 123

Question:

Write a program to remove all whitespaces from a given string.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
String x = "T    his is b  ett     er.";
x = x.replaceAll("\\s", "");
System.out.println(x);
}
```

```
}
```

Question 124

Question:

Write a program to get current working directory.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
String path = System.getProperty("user.dir");
System.out.println("Current Working Directory: " + path);
}
}
```

Question 125

Question:

Write a program to split a sentence into words.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
String x = "Hai this is Alan";
String [] y = x. split(" ", 3);
for(String i : y)
System. out. println(i);
}
}
```

Question 126

Question:

Write a program to replace all occurrences of 'a' to 'e' in a string.

Solution:

```java
public class MyClass {
public static void main(String args[]){
String x="Java is a powerful general-purpose programming language.";
String replaceString=x.replace('a','e');
System.out.println(replaceString);
}
}
```

Question 127

Question:

Write a program to check if the given string is empty or not.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
String a="";
String b="Java";
System.out.println(a.isEmpty());
System.out.println(b.isEmpty());
}
}
```

Question 128

Question:

Write a program to illustrate .join() method.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
```

```java
String a=String.join("-","Java","Programming");
System.out.println(a);
}
}
```

Question 129

Question:

Write a program to calculate surface area of cube.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int side;
long area;
System.out.print("\nEnter the side of cube: ");
side = STDIN_SCANNER.nextInt();
area = 6 * side * side;
System.out.print("\nThe surface area of cube is: " + area);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 130

Question:

Write a program to subtract 2 numbers without using subtraction operator.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int x = 6, y = 3;
System.out.print(x + ~y + 1);
}
}
```

Question 131

Question:

Write a program to add 2 numbers without using addition operator.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int x = 6, y = 3;
System.out.print(x - ~y - 1);
```

```
}

}
```

Question 132

Question:

Write a program to multiply a number by 2 without using multiplication operator.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int x = 2;
System.out.print(x << 1);
}
}
```

Question 134

Question:

Write a program to divide a number by 2 without using division operator.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
int x = 12;
System.out.print(x >> 1);
}
}
```

```java
public class MyClass {

public static void main(String[] args) {

String x = "Albert";

String y = "ALBERT";

System.out.println(x.equalsIgnoreCase(y))
;

// Output: true

}

}
```

Question 135

Question:

Write a program to calculate volume of sphere.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int radius;
float PI = 3.141592f;
System.out.print("\nEnter the radius of sphere: ");
radius = STDIN_SCANNER.nextInt();
float volume = (4 / 3) * (PI * radius * radius * radius);
System.out.printf("\nThe volume of sphere is: %f", volume);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 136

Question:

Write a program to calculate volume of ellipsoid.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int r1, r2, r3;
float PI = 3.141592f;
System.out.print("\nEnter the radius of the ellipsoid of axis 1: ");
r1 = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the radius of the ellipsoid of axis 2: ");
r2 = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the radius of the ellipsoid of axis 3: ");
r3 = STDIN_SCANNER.nextInt();
float volume = (4 / 3) * (PI * r1 * r2 * r3);
System.out.printf("\nThe volume of ellipsoid is: %f", volume);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 137

Question:

Write a program that uses a for loop to determine power of a number entered by the user.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int x, y;
long power = 1;
System.out.print("\nEnter the value for x: ");
x = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the value for y: ");
y = STDIN_SCANNER.nextInt();
for(int i = 1; i <= y; i++) {
    power = power * x;
}
System.out.print(x + " ^ " + y + " = " + power);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 138

Question:

Write a program to read three numbers and find average of numbers.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int a, b, c;
float avg;
System.out.print("\nEnter the first number: ");
a = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the second number: ");
b = STDIN_SCANNER.nextInt();
System.out.print("\nEnter the third number: ");
c = STDIN_SCANNER.nextInt();
avg = (float)((a + b + c) / 3.0);
System.out.printf("\nAverage of three numbers is: %f", avg);
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 139

Question:

Write a program to read integer "n" and print first three powers (n^1, n^2, n^3).

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {
int n;
System.out.print("\nEnter a number: ");
n = STDIN_SCANNER.nextInt();
System.out.printf("%f, %f, %f", Math.pow(n, 1), Math.pow(n, 2), Math.pow(n,
3));
}
public final static Scanner STDIN_SCANNER = new Scanner(System.in);
}
```

Question 140

Question:

Write a program to search the substring in a given string.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
String name="Java is a powerful general-purpose programming language";
System.out.println(name.contains("Java"));
System.out.println(name.contains("programming"));
System.out.println(name.contains("language"));
}
}
```

Question 141

Question:

Write a program to check if the string ends with a given suffix.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
String a="Java Programming";
System.out.println(a.endsWith("g"));
}
}
```

Question 142

Question:

Write a program to check if the string starts with the given prefix.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
String a="Java Programming";
System.out.println(a.startsWith("j"));
System.out.println(a.startsWith("J"));
}
}
```

Question 143

Question:

Write a program to check whether a character is alphabet, digit or special character.

Solution:

```java
import java.util.Scanner;
public class MyClass {
public static void main(String[] args) {
```

```java
char ch;
Scanner x=new Scanner(System.in);
System.out.print("Enter a character: ");
ch=x.next().charAt(0);
if((ch>='a'&&ch<='z')||(ch>='A'&&ch<='Z')) {
System.out.println(ch+" is Alphabet.");
}
else if(ch>='0'&&ch<='9') {
System.out.println(ch+" is Digit.");
}
else {
System.out.println(ch+" is Special Character.");
}
}
}
```

Question 144

Question:

Write a program to Check whether Java is installed on your computer.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
System.out.println("\nJava Version: "+System.getProperty("java.version"));
System.out.println("Java Runtime Version:
"+System.getProperty("java.runtime.version"));
```

```java
System.out.println("Java Home: "+System.getProperty("java.home"));
System.out.println("Java Vendor: "+System.getProperty("java.vendor"));
System.out.println("Java Vendor URL:
"+System.getProperty("java.vendor.url"));
System.out.println("Java Class Path:
"+System.getProperty("java.class.path")+"\n");
}
}
```

Question 145

Question:

Write a program to Check whether Java is installed on your computer.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
System.out.println("\nJava Version: "+System.getProperty("java.version"));
System.out.println("Java Runtime Version:
"+System.getProperty("java.runtime.version"));
System.out.println("Java Home: "+System.getProperty("java.home"));
System.out.println("Java Vendor: "+System.getProperty("java.vendor"));
System.out.println("Java Vendor URL:
"+System.getProperty("java.vendor.url"));
System.out.println("Java Class Path:
"+System.getProperty("java.class.path")+"\n");
}
```

```
}
```

Question 146

Question:

Write a program to get the current system environment and system properties.

Solution:

```
import java.lang.*;
public class Main {
public static void main(String[] args) {
System.out.println("\nCurrent system environment:");
System.out.println(System.getenv());
System.out.println("\n\nCurrent system properties:");
System.out.println(System.getProperties());
}
}
```

Question 147

Question:

Write a program to measure how long code takes to execute in nanoseconds.

Solution:

```java
import java.lang.*;
public class Main {
public static void main(String[] args) {
long startTime = System.nanoTime();
int i;
System.out.println ("The first 5 natural numbers are:\n");
for (i=1;i<=5;i++) {
    System.out.println(i);
}
long estimatedTime = System.nanoTime() - startTime;
System.out.println("Estimated time (in nanoseconds) to get the first 5
natural numbers: "+estimatedTime);
}
}
```

Question 148

Question:

Write a program to replace the spaces of a string with a specific character.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
String a = "Java Programming";
char ch = '-';
```

```
a = a.replace(' ', ch);
System.out.println("String after replacing spaces with the character '-': ");
System.out.println(a);
}
}
```

Question 149

Question:

Write a program to count the total number of punctuations in a given string.

Solution:

```java
public class Main {
public static void main (String args[]) {
int count = 0;
String str = "Logic will get you from A to Z; imagination will get you
everywhere.";
for(int i = 0; i < str.length(); i++) {
if(str.charAt(i) == '!' || str.charAt(i) == ',' || str.charAt(i) == ';'
|| str.charAt(i) == '.' ||  str.charAt(i) == '?' || str.charAt(i) == '-'
|| str.charAt(i) == '\'' || str.charAt(i) == '\"' || str.charAt(i) == ':') {
count++;
}
}
System.out.println("The total number of punctuations in a given string is: "
+count);
}
```

```
}
```

Question 150

Question:

Write a program to convert Decimal to Hexadecimal.

Solution:

```java
public class MyClass {
public static void main(String args[]){
System.out.println(Integer.toHexString(10));
System.out.println(Integer.toHexString(15));
System.out.println(Integer.toHexString(289));
}
}
```

Question 151

Question:

Write a program to convert Decimal to Octal.

Solution:

```java
public class MyClass {
public static void main(String args[]){


System.out.println(Integer.toOctalString(8));
System.out.println(Integer.toOctalString(19));
System.out.println(Integer.toOctalString(81));


}


}
```

Question 152

Question:

Write a program to convert Decimal to Binary.

Solution:

```java
public class MyClass {
public static void main(String args[]){
System.out.println(Integer.toBinaryString(10));
System.out.println(Integer.toBinaryString(21));
System.out.println(Integer.toBinaryString(31));
}


}
```

Question 153

Question:

Write a program to convert Binary to Decimal.

Solution:

```java
public class MyClass {
public static void main(String args[]){
String a="1010";
int decimal=Integer.parseInt(a,2);
System.out.println(decimal);
}
}
```

```java
public class MyClass {

int x = 15;

public static void main(String[] args) {

Main myfunc = new Main();

System.out.println(myfunc.x);

// Output: 15

}

}
```

Question 154

Question:

Write a program to convert Hexadecimal to Decimal.

Solution:

```java
public class MyClass {
public static void main(String args[]){
String hex="a";
int decimal=Integer.parseInt(hex,16);
System.out.println(decimal);
}


}
```

Question 155

Question:

Write a program to determine whether one string is a rotation of another.

Solution:

```java
public class MyClass {
public static void main(String[] args) {
String x = "abcde", y = "deabc";
if(x.length() != y.length()){
   System.out.println("Second string is not a rotation of first string");
}
else {
    x = x.concat(x);

        if(x.indexOf(y) != -1)
```

```
            System.out.println("Second string is a rotation of first
string");
        else
            System.out.println("Second string is not a rotation of first
string");
    }
  }
}
```

Question 156

Question:

Write a program to illustrate the isNaN method.

Solution:

```java
public class MyClass {
public static void main(String args[]) {

/* The isNaN method returns true if the value is NaN. */

Float a = Float.NaN;
Float b = 6.0f;
System.out.println(a +" - " + a.isNaN());
System.out.println(a +" - " + Float.isNaN(a));
System.out.println(b +" - " + Float.isNaN(b));
}
```

}

Question 157

Question:

Write a program to illustrate the isNaN method.

Solution:

```java
public class MyClass {
public static void main(String args[]) {

/* The isNaN method returns true if the value is NaN. */

Float a = Float.NaN;
Float b = 6.0f;
System.out.println(a +" - " + a.isNaN());
System.out.println(a +" - " + Float.isNaN(a));
System.out.println(b +" - " + Float.isNaN(b));
}
}
```

```java
public class MyClass {
public static void main(String[] args) {
String[] x = {"Apple", "Orange", "Kiwi", "Lemon"};
for (String i : x) {
System.out.println(i);
}
}
}
```

Question 158

Question:

Write a program to Design Simple Calculator.

Solution:

```java
import java.util.Scanner;

public class MyClass {
public static void main(String[] args) {

char operator;
Double number1, number2, result;
Scanner input = new Scanner(System.in);
System.out.println("Choose an operator: +, -, *, or /");
operator = input.next().charAt(0);

System.out.println("Enter first number:");
number1 = input.nextDouble();

System.out.println("Enter second number:");
number2 = input.nextDouble();

switch (operator) {

    case '+':
      result = number1 + number2;
      System.out.println(number1 + " + " + number2 + " = " + result);
```

```java
            break;

        case '-':
            result = number1 - number2;
            System.out.println(number1 + " - " + number2 + " = " + result);
            break;

        case '*':
            result = number1 * number2;
            System.out.println(number1 + " * " + number2 + " = " + result);
            break;

        case '/':
            result = number1 / number2;
            System.out.println(number1 + " / " + number2 + " = " + result);
            break;

        default:
            System.out.println("Invalid operator!");
            break;
    }

    input.close();
  }
}
```

```java
public class MyClass  {

public static void main(String[] args) {

int[][] x = { {11, 12, 13, 14}, {15, 16, 17}};

System.out.println(x[1][2]);  // Output: 17

}

}
```

Question 159

Question:

Write a program to print Invert Triangle.

Solution:

```java
public class MyClass {
public static void main(String args[]) {
int x = 9;
while(x > 0) {
    for(int i=1; i<=x; i++) {
        System.out.print(" "+x+" ");
    }
System.out.print("\n");
x--;
}
}
}
```

```java
public class MyClass {

public static void main(String[] args) {

String[] x = {"Albert", "John", "James", "Mary"};

x[0] = "Elsa";

System.out.println(x[0]); // Output: Elsa

}

}
```

```java
public class MyClass {

public static void main (String [] args) {

String x = "Einstein";

System.out.println(x.charAt(1));

// Output: i

}

}
```

```java
public class MyClass {

public static void main (String [] args){

String x = "   Albert ";

System.out.println(x);

System.out.println(x.trim());

}

}
```

```
# Output:

   Albert

Albert
```

```java
public class MyClass {

public static void main (String [] args){

double x =56.698;

double y =-56.898;

double z =56.45;

System.out.println(Math.round(x)); // Output: 57

System.out.println(Math.round(y)); // Output: -57

System.out.println(Math.round(z)); // Output: 56

}

}
```

```java
public class MyClass {

public static void main (String [] args){

String x = "Most Important Programming Concepts";

System.out.println(x.substring(15));

// Output: Programming Concepts

System.out.println(x.substring(26));

// Output: Concepts

System.out.println(x.substring(15, 26));

// Output: Programming

System.out.println(x.substring(0, 5));

// Output: Most

}

}
```

Parallel computing refers to the use of multiple processors or computing systems to solve a single problem or perform a task. In **parallel computing**, a single task is divided into multiple smaller tasks, each of which is executed simultaneously on multiple processors or computing systems.

The main goal of **parallel computing** is to improve the speed and efficiency of computing systems by dividing a large problem into smaller subproblems and processing them simultaneously. **Parallel computing** is used in a wide range of applications, including scientific computing, data analysis, machine learning, image and signal processing, and more.

Parallel computing can be classified into two categories: shared memory and distributed memory. In shared memory systems, multiple processors share a single memory space and can access the same data directly. In distributed memory systems, each processor has its own local memory, and communication between processors is done via message passing.

Parallel computing is becoming increasingly important as the demand for faster and more efficient computing systems continues to grow. It requires specialized hardware and software, as well as expertise in programming and algorithm design. Parallel computing is an important area of study for computer scientists, engineers, and researchers who are involved in designing and developing high-performance computing systems.

```
public class MyClass {

public static void main(String args[]) {

Integer i = new Integer(10);

int a = i;

System.out.println(a);

}

}
```

Output:

10

```
public class MyClass {

public static void main(String args[]) {

StringBuffer buffer=new StringBuffer("Java");

buffer.append(" Programming");

System.out.println(buffer);

}

}
```

Output:

Java Programming

Overloading provides

- code clarity

- reduce complexity

- increases runtime presentation of a code

719

```
public class MyClass {

public static void main(String args[]) {

StringBuilder builder=new StringBuilder("Java");

builder.append(" Programming");

System.out.println(builder);

}

}
```

Output:

Java Programming

Function overloading	Operator overloading
using a single name and giving more functionality to it	adding extra functionality for a certain operator

Python Exercises

Python is a well-known general-purpose, interactive, object-oriented, and high-level programming language. Python is a dynamically typed, garbage-collected programming language. Between 1985 and 1990, **Guido van Rossum** developed it. Python is a wonderful introductory language since its code is clear and simple to read. Python is capable of doing everything you ask of it. Python is the language for you if you're into data research, machine learning, or web development. This chapter includes Python programming language learning problems and solutions. Exercises are an excellent approach to learn the Python programming language since you learn programming best by doing. Python is a very well-liked programming language that enables you to create anything from robotics to web applications.

Question 1

Question:

Write a program to add two numbers.

Solution:

```
a = 1
b = 2
c= a+b
print(c)
```

```
a = int(input("Enter a number: "))
b = int(input("Enter a number: "))
c= a+b
print(c)
```
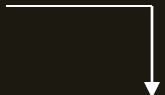
```
import numpy as np

x= np.ones([2,4])

print(x)

print(x.dtype)

print(x.shape)
```

```
# Output:
   [[1. 1. 1. 1.]
   [1. 1. 1. 1.]]
      float64
      (2, 4)
```

Question 2

Question:

Write a program to find whether a given number (accept from the user) is even or odd, print out an appropriate message to the user.

Solution:

```python
a = int(input("Enter a number: "))
if a % 2 == 0:
    print("This is an even number.")
else:
    print("This is an odd number.")
```

```python
import sys

print("Albert (Einstein)")

# Output: Albert (Einstein)

print("Elsa (Einstein)", file=sys.stderr)

# Output: Elsa (Einstein)

sys.stderr.write("David Einstein")

# Output: David Einstein
```

Question 3

Question:

Write a program to check whether a number entered by the user is positive, negative or zero.

Solution:

```python
a = int(input("Enter a number: "))
if a > 0:
    print("Positive number")
elif a == 0:
    print("Zero")
else:
    print("Negative number")
```

```python
for i in "albert":

    print(i)

# Output:

    a

    l

    b

    e

    r

    t
```

Question 4

Question:

Write a program to display the calendar of a given date.

Solution:

```
import calendar
yy = int(input("Enter year: "))
mm = int(input("Enter month: "))
print(calendar.month(yy, mm))
```

```
names = ["Albert", "Paul", "John"]

names[0] = "David"

print(names)

# Output: ['David', 'Paul', 'John']
```

Question 5

Question:

Write a program to ask the user to enter the string and print that string as output of the program.

Solution:

```
x= input("Enter string: ")
print("You entered:", x)
```

```
x = ("ball", "bag", "bat")
y = iter(x)


print(next(y))

print(next(y))

print(next(y))
```

```
# Output:

   ball

   bag

   bat
```

Question 6

Question:

Write a program to concatenate two strings.

Solution:

```python
x = input("Enter first string to concatenate: ")
y = input("Enter second string to concatenate: ")
z = x + y
print("String after concatenation = ", z)
```

Question 7

Question:

Write a program to check if an item exists in the list.

Solution:

```python
x = ["ball", "book", "pencil"]
i = input("Type item to check: ")
if i in x:
 print("Item exists in the list.")
else:
  print("Item does not exist in the list.")
```

```python
x = abs(-9.78)

print(x)

# Output: 9.78
```

Question 8

Question:

Write a program to join two or more lists.

Solution:

```
x = ["This" , "is", "a", "blood", "sample"]
y = [20, 6, 55, 3, 9, 7, 18, 20]
z = x + y
print(z)
```

```
x = min(15, 100, 215)

y = max(15, 100, 215)


print(x)

# Output: 15

print(y)

# Output: 215
```

Question 9

Question:

Write a program to calculate cube of a number.

Solution:

```
import math
x = int(input("Enter a number: "))
y=math.pow(x,3)
print(y)
```

Question 10

Question:

Write a program to calculate square root of a number.

```
import math

x = math.ceil(6.8)

y = math.floor(6.8)

print(x) # Output: 7

print(y) # Output: 6
```

Solution:

```
import math
x = int(input("Enter a number: "))
y=math.sqrt(x)
print(y)
```

Question 11

Question:

Write a program that takes a list of numbers (for example, i = [6, 10, 75, 60, 55]) and makes a new list of only the first and last elements of the given list.

Solution:

```
i = [6, 10, 75, 60, 55]
print([i[0], i[4]])
```

Question 12

Question:

Take a list, say for example this one: x = [1, 1, 2, 3, 2, 8, 18, 31, 14, 25, 78] and write a program that prints out all the elements of the list that are less than 4.

Solution:

```python
x = [1, 1, 2, 3, 2, 8, 18, 31, 14, 25, 78]
for i in x:
    if i < 4:
        print(i)
```

```python
x = "alan"

y = iter(x)


print(next(y))

print(next(y))

print(next(y))

print(next(y))
```

```
# Output:

    a

    l

    a

    n
```

Question 13

Question:

Let's say I give you a list saved in a variable: x = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]. Write one line of Python that takes this list 'x' and makes a new list that has only the even elements of this list in it.

Solution:

```python
x = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
y = [i for i in a if i % 2 == 0]
print(y)
```

Question 14

Question:

Ask the user for a string and print out whether this string is a palindrome or not (A palindrome is a string that reads the same forwards and backwards).

Solution:

```
x=input("Please enter a word: ")
z = x.casefold()
y = reversed(z)
if list(z) == list(y):
    print("It is palindrome")
else:
    print("It is not palindrome")
```

```
import math

x = math.pi

print(x)

# Output: 3.141592653589793
```

Question 15

Question:

Take two lists, say for example these two: x = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89] y = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] and write a program that returns a list that contains only

the elements that are common between the lists (without duplicates). Make sure your program works on two lists of different sizes.

Solution:

```
x = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
y = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
print([i for i in set(x) if i in y])
```

Question 16

Question:

Write a program to add a string to text file.

Solution:

```
file = open("testfile.txt","w")
file.write("Albert Einstein")
file.write("Elsa Einstein")
file.write("David Einstein.")
file.write("Why E=mc squared?.")
file.close()
```

```
x = ["albert", "john", "david"]
for i in x:
  print(i)
  if i == "john":
    break

# Output:
       albert
       john
```

Question 17

Question:

Write a program to read a file and display its contents on console.

Solution:

```python
with open('testfile.txt') as f:
    i = f.readline()
    while i:
        print(i)
        line = f.readline()
```

> A program to split the string
> at every white-space character
>
> ```python
> import re
>
> str = "Stephen William Hawking"
>
> x = re.split("\s", str)
>
> print(x)
>
> # Output: ['Stephen', 'William', 'Hawking']
> ```

Question 18

Question:

Take two sets, say for example these two: x = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89} y = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13} and write a program that returns a set that contains only the elements that are common between the sets.

Solution:

```python
x = {1, 1, 2, 2, 3, 5, 8, 13, 21, 34, 55, 89}
y = {1, 2, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}
print(set(x) & set(y))
```

Question 19

Question:

Write a program to split the characters of the given string into a list.

Solution:

```
x= "albert"
y = list(x)
print(y)
```

A program to replace all white-space characters with the symbol "+"

```
import re

str = "Stephen William Hawking"

x = re.sub("\s", "+", str)

print(x)

# Output: Stephen+William+Hawking
```

Question 20

Question:

Create a program that asks the user for a number and then prints out a list of all the divisors of that number.

Solution:

```
x=int(input("Enter an integer: "))
print("The divisors of the number are: ")
for i in range(1,x+1):
    if(x%i==0):
```

```
        print(i)
```

Question 21

Question:

Write a program to Find the largest of three numbers.

Solution:

```
x = int(input("Enter first number: "))
y = int(input("Enter second number: "))
z = int(input("Enter third number: "))
if (x > y) and (x > z):
    largest = x
elif (y > x) and (y > z):
    largest = y
else:
    largest = z
print("The largest number is", largest)
```

A program to return a list containing
every occurrence of "in"

```
import re

str = "Albert Einstein"

x = re.findall("in", str)

print(x)

# Output: ['in', 'in']
```

Question 22

Question:

Write a program to find absolute value of a number.

Solution:

```python
x = int(input("Enter a number: "))
if x >= 0:
        print(x)
else:
        print(-x)
```

Question 23

Question:

Write a program to find the length of a string.

A program to check if the string starts with "The" and ends with "secretary"

```python
import re
str = "The boy is the sports secretary"
x = re.search("^The.*secretary$", str)

if x:
print("YES! We've got a match!")
else:
print("No match")

# Output: YES! We've got a match!
```

Solution:

```python
print("Enter 'y' for exit.")
i = input("Enter a string: ")
if i == 'y':
    exit()
else:
    print("Length of the string is: ", len(i))
```

Question 24

Question:

Write a program to print natural numbers from 1 to x.

Solution:

```python
x = int(input("Please Enter any Number: "))
for i in range(1, x+1):
    print(i)
```

A program to capitalize the first letter of each word in the string

```python
import camelcase
c = camelcase.CamelCase()
str = "albert einstein"
print(c.hump(str))

# Output: Albert Einstein
```

Question 25

Question:

Write a program to calculate the sum and average of natural numbers from 1 to x.

Solution:

```python
x = int(input("Please Enter any Number: "))
sum = 0
for i in range(1,x+1):
    sum = sum + i
print(sum)
average = sum / x
print(average)
```

```python
x = [(2,6),(4,7),(5,9),(8,4),(2,1)]

x.sort()

print(x)

# Output: [(2, 1), (2, 6), (4, 7), (5, 9), (8, 4)]
```

Question 26

Question:

Write a program to print a statement any number of times.

Solution:

```python
x = int(input("Please Enter any Number: "))
for i in range(x):
    print("Albert Einstein")
```

```python
print(type(True))

# Output: <class 'bool'>

print(type(False))

# Output: <class 'bool'>

print(type([1,2]))

# Output: <class 'list'>

print(type({1,2}))

# Output: <class 'set'>
```

Question 27

Question:

Write a program to multiply two numbers using Function.

Solution:

```python
def myfunc():
    x = int(input("Enter a number: "))
    y=int(input("Enter a number: "))
    z= x*y
    return z
```

```
i = myfunc()
print(i)
```

Question 28

Question:

Write a program to add an item to the end of the list.

Solution:

```
x = ["pen", "book", "ball"]
x.append("bat")
print(x)
```

```
x = 6

y = 2

print(x+y)

# Output: 8

print(x-y)

# Output: 4

print(x*y)

# Output: 12

print(x/y)

# Output: 3.0
```

Question 29

Question:

Write a program to remove an item from the list.

```
x = "23"
y = "54"
z = x + y
print(z)
# Output: 2354
```

Solution:

```
x = ["pen", "book", "ball"]
x.remove("ball")
```

```
print(x)
```

Question 30

Question:

Write a program to print the number of elements in an array.

Solution:

```
x = ["pen", "book", "ball"]
y = len(x)
print(y)
```

```
import sys

sys.stdout.write("Albert ")

sys.stdout.write("Einstein")

# Output: Albert Einstein
```

Question 31

Question:

Write a program to calculate the variance and standard deviation of the elements of the list.

Solution:

```
import numpy as np
x= [2,6,8,12,18,24,28,32]
```

```
variance= np.var(x)
std = np.std(x)
print(variance)
print(std)
```

```
def main():

    print("Albert Einstein")


if___name___== "__main__":

    main()


# Output: Albert Einstein
```

Question 32

Question:

Write a program to get the difference between the two lists.

Solution:

```
x = [4, 5, 6, 7]
y = [4, 5]
print(list(set(x) - set(y)))
```

Concurrency in computer science refers to the ability of a computer system to execute multiple tasks or processes simultaneously, without affecting the overall performance of the system. In concurrent computing, tasks are executed independently, and their execution may overlap in time. This enables the computer system to make more efficient use of its resources and improve overall performance.

Concurrency can be implemented using threads, which are lightweight processes that share the same memory space and execute independently. Multiple threads can be executed simultaneously on multiple processors or cores, enabling the system to perform multiple tasks in parallel.

Concurrency is used in a wide range of applications, including web servers, database systems, operating systems, and more. In web servers, concurrency enables the server to handle multiple requests simultaneously, improving the responsiveness and throughput of the system. In database systems, concurrency enables multiple users to access and modify the same database simultaneously, without conflicts or data corruption.

Question 33

Question:

Write a program to select an item randomly from a list.

Solution:

Concurrency can also introduce new challenges, such as race conditions and deadlocks, which can affect the correctness and reliability of the system. Ensuring correct and safe concurrent execution requires specialized techniques, such as synchronization, locking, and atomic operations, which must be carefully designed and implemented. Concurrency is an important area of study in computer science, and it is essential for the development of high-performance and scalable software systems.

```python
import random
x = ['Paper', 'Pencil', 'Book', 'Bag', 'Pen']
print(random.choice(x))
```

Question 34

Question:

Write a program that prints all the numbers from 0 to 6 except 2 and 6.

Solution:

```python
for x in range(6):
    if (x == 2 or x==6):
        continue
    print(x)
```

```python
def main():

    print("William")


print("Stephen")

main()

print("Hawking")
```

Output:

    Stephen

    William

    Hawking

Question 35

Question:

Write a program that takes input from the user and displays that input back in upper and lower cases.

Solution:

```
x = input("What is your name? ")
print(x.upper())
print(x.lower())
```

```
print("Albert ", sep="")

print("Einstein")
```

```
# Output:

        Albert

        Einstein
```

Question 36

Question:

Write a program to check whether a string starts with specified characters.

Solution:

```
x = "science.com"
print(x.startswith("phy"))
```

```
print("Albert ", end="")

print("Einstein")

# Output: Albert Einstein
```

Question 37

Question:

Write a program to create the multiplication table (from 1 to 10) of a number.

Solution:

```
x = int(input("Enter a number: "))
for i in range(1,11):
```

```
   print(x,'x',i,'=',x*i)
```

Question 38

Question:

Write a program to check a triangle is equilateral, isosceles or scalene.

Solution:

```
print("Enter lengths of the triangle sides: ")
x = int(input("x: "))
y = int(input("y: "))
z = int(input("z: "))
if x == y == z:
    print("Equilateral triangle")
elif x==y or y==z or z==x:
    print("isosceles triangle")
else:
    print("Scalene triangle")
```

```
def main():

  x = input('First number: ')

  y = input('Second number: ')

  print(x + y)


main()                    ?
```
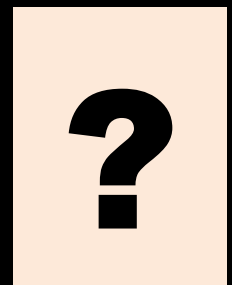
Question 39

Question:

Write a program to sum of two given integers. However, if the sum is between 15 to 20 it will return 20.

Solution:

```python
x = int(input("Enter a number: "))
y = int(input("Enter a number: "))
z= x+y
if z in range(15, 20):
        print (20)
else:
        print(z)
```

```python
def main():

  a = 2.5

  b = 2.9

  print(int(a) + int(b))


main()

# Output: 4
```

Question 40

Question:

Write a program to convert degree to radian.

Solution:

```python
pi=22/7
degree = int(input("Input degrees: "))
radian = degree*(pi/180)
print(radian)
```

```python
x = ["albert", "john", "david"]
for i in x:
  if i == "john":
    continue
  print(i)
```

```
# Output:

        albert

        david
```

Question 41

Question:

Write a program to generate a random number.

Solution:

```python
import random
print(random.randint(0,9))
```

```python
def main():
    a = 2.5
    b = 2.9
    print(float(a) + float(b))


main()
# Output: 5.4
```

Question 42

Question:

Write a program to find the semi-perimeter of triangle.

Solution:

```python
x = int(input('Enter first side: '))
y = int(input('Enter second side: '))
z = int(input('Enter third side: '))
s = (x + y + z) / 2
print(s)
```

Question 43

Question:

Given a list of numbers, iterate it and print only those numbers which are divisible of 2.

Solution:

```
x = [10, 20, 33, 46, 55]
for i in x:
    if (i % 2 == 0):
        print(i)
```

```
x = '2'
print(x)
# Output: 2
print(x.isdecimal())
# Output: True
print(x.isnumeric())
# Output: True

if x.isdecimal():
    y = int(x)
    print(y)
    # Output: 2
```

Question 44

Question:

Write a program to multiply all numbers in the list.

Solution:

```
import numpy
x = [1, 2, 3]
y = numpy.prod(x)
print(y)
```

Question 45

Question:

Write a program to print ASCII Value of a character.

Solution:

```python
x = 'j'
print("The ASCII value of '" + x + "' is", ord(x))
```

```python
x = "56"

print(type(x))

# Output: <class 'str'>

y = int(x)

print(type(y))

# Output: <class 'int'>
```

A program to convert
'string' to 'int'

Question 46

Question:

Write a program to print "#" without a newline or space.

Solution:

```python
for x in range(0, 5):
    print('#', end="")
print("\n")
```

```python
x = 56.39

print(type(x))

# Output: <class 'float'>

y = int(x)

print(type(y))

# Output: <class 'int'>
```

A program to convert
'float' to 'int'

Question 47

Question:

Write a program that will convert a string to a float or an integer.

Solution:

```
x = "546.11235"
print(float(x))
print(int(float(x)))
```

```
print(int(float(5.6)))

# Output: 5

print(int(float("5")))

# Output: 5

print(int(float(5)))

# Output: 5
```

Question 48

Question:

Write a program to add and search data in the dictionary.

Solution:

```
# Define a dictionary
customers = {'1':'Mehzabin Afroze','2':'Md. Ali',
'3':'Mosarof Ahmed','4':'Mila Hasan', '5':'Yaqub Ali'}

# Append a new data
customers['6'] = 'Mehboba Ferdous'
```

```
print("The customer names are:")
# Print the values of the dictionary
for customer in customers:
    print(customers[customer])


# Take customer ID as input to search
name = input("Enter customer ID:")

# Search the ID in the dictionary
for customer in customers:
    if customer == name:
        print(customers[customer])
        break
```

```
def main():

    x = input('First number: ')

    y = input('Second number: ')

    if int(y) == 0:

        print("Cannot divide by 0")

    else:

        print("Dividing", x, "by", y)

        print(int(x) / int(y))
main()
```

Question 49

Question:

Write a program to obtain the details of the math module.

Solution:

```
import math
print(dir(math))
```

Question 50

Question:

Write a program to demonstrate throw and catch exception.

Solution:

```python
# Try block
try:

    # Take a number

    x = int(input("Enter a number: "))

    if x % 2 == 0:

        print("Number is even")

    else:

        print("Number is odd")


# Exception block
except (ValueError):

  # Print error message

  print("Enter a numeric value")
```

```python
import random

names = ["Albert", "John", "Mary", "Alan"]

# pick and print one of the names

print(random.choice(names))

# Output: Albert
```

```python
import array

x = array.array('i', [11,13,15,17,19])

for i in x:

    print(i)
```

Output:

11

13

15

17

19

Question 51

Question:

Write a program to illustrate password authentication.

Solution:

```python
# import getpass module
import getpass

# Take password from the user
passwd = getpass.getpass('Password:')

# Check the password
if passwd == "albert":
    print("You are authenticated")
else:
    print("You are not authenticated")
```

```python
a = 4

b = 2

print(a ** b) # is the same as a ^ b

# Output: 16

print(a % b)

# a is divided by b that returns 0 as the remainder

# Output: 0

a += 2 # is the same as a = a + 2

print(a)

# Output: 6

b -= 1 # is the same as b = b - 1

print(b)

# Output: 1
```

Question 52

Question:

Write a program to calculate the average of numbers in a given list.

Solution:

```python
x=int(input("Enter the number of elements to be inserted: "))
y=[]
for i in range(0,x):
    n=int(input("Enter element: "))
    y.append(n)
avg=sum(y)/x
print("Average of elements in the list: ",round(avg,2))
```

Question 53

Question:

Write a program that sorts three integers without the need of loops or conditional statements.

```python
import random

x = "123456789"

# pick and print one of the numbers
print(random.choice(x))

# Output: 3
```

Solution:

```python
a = int(input("Enter the first number: "))
b = int(input("Enter the second number: "))
c = int(input("Enter the third number: "))

x = min(a, b, c)
z = max(a, b, c)
y = (a + b + c) - x - z
print("Numbers in sorted order: ", x, y, z)
```

```
import random

print(1 + int(3 * random.random()))

print(random.randrange(1, 3))
```

752

Question 54

Question:

Write a program to determine the sum of digits of a number.

Solution:

```
num = int(input("Enter a four digit number: "))
x   = num //1000
y = (num - x*1000)//100
z = (num - x*1000 - y*100)//10
c = num - x*1000 - y*100 - z*10
print("The number's digits add up to: ", x+y+z+c)
```

Question 55

Question:

Write a program to take in the marks of 5 subjects and display the grade.

Solution:

```python
sub1=int(input("Enter marks of the first subject: "))
sub2=int(input("Enter marks of the second subject: "))
sub3=int(input("Enter marks of the third subject: "))
sub4=int(input("Enter marks of the fourth subject: "))
sub5=int(input("Enter marks of the fifth subject: "))
avg=(sub1+sub2+sub3+sub4+sub4)/5
if(avg>=90):
    print("Grade: A")
elif(avg>=80 and avg<90):
    print("Grade: B")
elif(avg>=70 and avg<80):
    print("Grade: C")
elif(avg>=60 and avg<70):
    print("Grade: D")
else:
    print("Grade: F")
```

```python
import random

names = ["Albert", "Alan", "John", "James", "Mary"]

print(random.sample(names, 2))

# Output:  ['Mary', 'James']
```

Question 56

Question:

Write a program to print all numbers in a range divisible by a given number.

Solution:

753

```
x=int(input("Enter lower range limit: "))
y=int(input("Enter upper range limit: "))
n=int(input("Enter the number to be divided by: "))
for i in range(x,y+1):
    if(i%n==0):
        print(i)
```

Question 57

Question:

Write a program to read two numbers and print their quotient and remainder.

Solution:

```
a=int(input("Enter the first number: "))
b=int(input("Enter the second number: "))
quotient=a//b
remainder=a%b
print("Quotient is:", quotient)
print("Remainder is:", remainder)
```

```
for x in range(7):
    print(x)
```

Output

0
1
2
3
4
5
6

Question 58

Question:

Write a program to determine whether a given value is present in a collection of values.

Solution:

```python
def myfunc(x, i):
    for value in x:
        if i == value:
            return True
    return False
print(myfunc([19, 15, 18, 13], 13))
print(myfunc([15, 18, 13], -11))
```

```python
x = 0

if x:

    print("Albert Einstein")

else:

    print("Elsa Einstein")


# Output: Elsa Einstein
```

Question 59

Question:

Write a program to print odd numbers within a given range.

Solution:

```python
x = 1

if x:

    print("Albert Einstein")

else:

    print("Elsa Einstein")


# Output: Albert Einstein
```

```
x=int(input("Enter the lower limit for the range: "))
y=int(input("Enter the upper limit for the range: "))
for i in range(x,y+1):
    if(i%2!=0):
        print(i)
```

Question 60

Question:

Write a program to find the smallest divisor of an integer.

Solution:

```
n=int(input("Enter an integer: "))
a=[]
for i in range(2,n+1):
    if(n%i==0):
        a.append(i)
a.sort()
print("Smallest divisor is:",a[0])
```

```
for i in range(5):
  print(i)
else:
  print("Albert!")
```

```
# Output:
    0
    1
    2
    3
    4
  Albert!
```

Question 61

Question:

Write a program to count the number of digits in a number.

Solution:

```
n=int(input("Enter a number:"))
i=0
while(n>0):
    i=i+1
    n=n//10
print("The number of digits in the number are:", i)
```

```
x = "56"

y = 56

print(x == y)

# Output: False

print(x != y)

# Output: True

print(y == 56.0)

# Output: True

print(None == None)

# Output: True

print(None == False)

# Output: False
```

Question 62

Question:

Write a program to read a number n and print and compute the series "1+2+...+n=".

Solution:

```
n=int(input("Enter a number: "))
```

```
x=[]
for i in range(1, n+1):
    print(i, sep=" ", end=" ")
    if(i<n):
        print("+", sep=" ", end=" ")
    x.append(i)
print("=",sum(x))


print()
```

```
a = 16

b = 4

print(a > b)

# Output: True
```

Question 63

Question:

Write a program to read a number n and print the natural numbers summation pattern.

Solution:

```
n=int(input("Enter a number: "))
for j in range(1, n+1):
    x=[]
    for i in range(1, j+1):
        print(i, sep=" ", end=" ")
        if(i<j):
            print("+",sep=" ",end=" ")
        x.append(i)
    print("=", sum(x))
```

```
a = "16"

b = "4"

print(a > b)

# Output: False
```

```
print()
```

Question 64

Question:

Write a program to read a number n and print an identity matrix of the desired size.

Solution:

```
n=int(input("Enter a number: "))
for i in range(0, n):
    for j in range(0, n):
        if(i==j):
            print("1", sep=" ", end=" ")
        else:
            print("0", sep=" ", end=" ")
    print()
```

```
import array

x = array.array('i', [11,13,15,17,19])

x.append(115)

for i in x:

    print(i)
```

Output:

11

13

15

17

19

115

Question 65

Question:

Write a program to read a number n and print an inverted star pattern of the desired size.

Solution:

```python
n=int(input("Enter number of rows: "))
for i in range (n,0,-1):
    print((n-i) * ' ' + i * '*')
```

```python
x = "Stephen " \
 "William " \
 "Hawking"

print(x)

# Output: Stephen William Hawking
```

Question 66

Question:

Write a program to determine the hypotenuse of a right-angled triangle.

Solution:

```python
from math import sqrt
print("Enter the lengths of shorter triangle sides: ")
x = float(input("x: "))
y = float(input("y: "))
```

```
z = sqrt(x**2 + y**2)
print("The length of the hypotenuse is: ", z)
```

Question 67

Question:

Write a program to find the largest number in a list.

```
x = """Stephen

William

Hawking"""

print(x)
```

Solution:

```
x=[]
n=int(input("Enter number of elements: "))
for i in range(1, n+1):
    y=int(input("Enter element: "))
    x.append(y)
x.sort()
print("Largest element is: ",x[n-1])
```

```
# Output:

    Stephen

    William

    Hawking
```

Question 68

Question:

Write a program to find the second largest number in a list.

Solution:

```
x=[]
n=int(input("Enter number of elements: "))
for i in range(1,n+1):
    y=int(input("Enter element: "))
    x.append(y)
x.sort()
print("Second largest element is: ",x[n-2])
```

```
x = 3 * 'Alan '
print(x)
# Output: Alan Alan Alan
```

Question 69

Question:

Write a program to put the even and odd elements in a list into two different lists.

Solution:

```
a=[]
n=int(input("Enter number of elements:"))
for i in range(1,n+1):
    b=int(input("Enter element:"))
    a.append(b)
even=[]
odd=[]
for j in a:
```

```
x = "Einstein"
a = x[0]
print(a)
# Output: E
b = x[3]
print(b)
# Output: s
```

```
    if(j%2==0):
        even.append(j)
    else:
        odd.append(j)
print("The even list", even)
print("The odd list", odd)
```

```
str = "wxyz"

print(str)

# Output: wxyz

str = str[:2] + 'Q' + str[3:]

print(str)

# Output: wxQz
```

Question 70

Question:

Write a program to concatenate all elements in a list into a string and return it.

Solution:

```
def myfunc(list):
    result= ''
    for i in list:
        result += str(i)
    return result

print(myfunc([2, 4, 13, 4]))
```

```
x = ["Alan", "Albert"]

y = ["Turing", "Einstein"]


for a in x:

  for b in y:

    print(a, b)
```

```
# Output:

  Alan Turing

  Alan Einstein

  Albert Turing

  Albert Einstein
```

763

Question 71

Question:

Write a program to add the three integers given. However, the sum will be zero if two values are equal.

Solution:

```python
def myfunc(a, b, c):
    if a == b or b == c or a==c:
        sum = 0
    else:
        sum = a + b + c
    return sum
print(myfunc(12, 11, 12))
print(myfunc(13, 22, 22))
print(myfunc(22, 22, 22))
print(myfunc(12, 22, 13))
```

```python
x = "albert"

y = x.upper()

print(y)

# Output: ALBERT

print(y.lower())

# Output: albert
```

Question 72

Question:

Write a program to sort a list according to the length of the elements.

Solution:

```
x=[]
n=int(input("Enter number of elements: "))
for i in range(1,n+1):
    y=input("Enter element: ")
    x.append(y)
x.sort(key=len)
print(x)
```

```
x = "Einstein"
if "ein" in x:
  print('Found ein')
else:
  print("NOT found ein")

# Output: Found ein
```

Question 73

Question:

Write a program to create a list of tuples with the first element as the number and the second element as the square of the number.

Solution:

```
x=int(input("Enter the lower range:"))
y=int(input("Enter the upper range:"))
a=[(i,i**2) for i in range(x,y+1)]
print(a)
```

Question 74

Question:

Write a program to create a list of all numbers in a range which are perfect squares and the sum of the digits of the number is less than 10.

Solution:

```
l=int(input("Enter lower range: "))
u=int(input("Enter upper range: "))
a=[]
a=[x for x in range(l,u+1) if (int(x**0.5))**2==x and
sum(list(map(int,str(x))))<10]
print(a)
```

```
for x in range(32, 126):

    print(x, chr(x))
```

Question 75

Question:

Write a program to convert a distance (in feet) to inches, yards, and miles.

Solution:

```
n = int(input("Enter the distance in feet: "))
x = n * 12
y = n / 3.0
z = n / 5280.0

print("The distance in inches is: %i inches." % x)
print("The distance in yards is: %.2f yards." % y)
print("The distance in miles is: %.2f miles." % z)
```

```
x = "Albert Einstein"

print(x[1:4])

# Output: lbe

print(x[2:])

# Output: bert Einstein

print(x[:2])

# Output: Al
```

Question 76

Question:

Write a program to generate random numbers from 1 to 20 and append them to the list.

Solution:

```
import random
a=[]
n=int(input("Enter number of elements:"))
for j in range(n):
    a.append(random.randint(1,20))
print('Randomised list is: ',a)
```

Question 77

Question:

Write a program to sort a list of tuples in increasing order by the last element in each tuple.

Solution:

```python
def last(n):
    return n[-1]

def sort(tuples):
    return sorted(tuples, key=last)

a=input("Enter a list of tuples:")
print("Sorted:")
print(sort(a))
```

```python
import numpy as np

x = np.array([2, 4, 8])

y = np.array([3, 6, 12])

print(x)

# Output: [2 4 8]

print(y)

# Output: [3  6 12]

print(np.multiply(x, y))

# Output: [6 24 96]

print(np.dot(x, y)) # dot product

# Output: 126

print(np.matmul(x, y)) # matrix multiplication

# Output: 126
```

```python
import array

x = array.array('i', [11,13,15,17,19])

x.reverse()

for i in x:

    print(i)
```

Output:

19

17

15

13

11

Question 78

Question:

Write a program to determine whether the system is a big-endian or little-endian platform.

Solution:

```python
import sys
print()
if sys.byteorder == "little":
    print("Little-endian platform.")
else:
    print("Big-endian platform.")
print()
```

```python
x = 'Albert Einstein'

for i in x:

  if i == ' ':

    break

  print(i)
```

```
# Output:
    A
    l
    b
    e
    r
    t
```

Question 79

Question:

Write a program to examine the available built-in modules.

Solution:

```
import sys
x = ', '.join(sorted(sys.builtin_module_names))
print("The available built-in modules are: ")
print()
print(x)
```

```
x = 2

i = x == 2

print(i)

# Output: True

if i:

  print("Albert Einstein")

else:

 print("Elsa Einstein")

# Output: Albert Einstein
```

Question 80

Question:

Write a program to determine an object's size in bytes.

Solution:

```
import sys
x = "three"
y = 154
z = [11, 12, 13, 'Ball', 'Bat']
print("Size of ",x,"=",str(sys.getsizeof(x))+ " bytes")
print("Size of ",y,"=",str(sys.getsizeof(y))+ " bytes")
print("Size of ",z,"=",str(sys.getsizeof(z))+ " bytes")
```

Question 81

Question:

Write a program to concatenate 'n' strings.

Solution:

```
x = ['Stephen', 'William', 'Hawking']
i = '-'.join(x)
print()
print(i)
print()
```

Question 82

Question:

Write a program to display the current date and time.

Solution:

```
import datetime
```

```
x = 'Albert Einstein'

for i in x:

  if i == ' ':

    continue

  print(i)
```

```
# Output:

    A

    l

    b

    e

    r

    t

    E

    i

    n

    s

    t

    e

    i

    n
```

```
print(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
```

```
x = 1915

y = 'Albert'

print("%s  Einstein's  %s papers." % (y, x))

# Output: Albert Einstein's 1915 papers.
```

Question 83

Question:

Write a program to calculate the volume of a sphere with a radius of 8.

Solution:

```
pi=22/7
r = 8.0
V = 4.0/3.0*pi*r**3
print('The volume of the sphere is: ',V)
```

Question 84

Question:

Write a program to check whether every number in a list exceeds a specific number.

Solution:

```
x = [12, 33, 44, 55]
print()
print(all(i > 11 for i in x))
print(all(i > 100 for i in x))
print()
```

```
x = 1915

y = 'Albert'

print(y + " Einstein's " + str(x) + " papers.")

# Output: Albert Einstein's 1915 papers.
```

Question 85

Question:

Write a program that count the occurrences of a particular character within a given string.

Solution:

```
x = "Albert Einstein."
print("Number of occurrence of 'e' in the given string: ")
print(x.count("e"))
```

Question 86

Question:

Write a program to compute simple interest given all the required values.

Solution:

```python
x=float(input("Enter the principal amount: "))
y=float(input("Enter the rate: "))
z=int(input("Enter the time (years): "))
simple_interest=(x*y*z)/100
print("The simple interest is: ", simple_interest)
```

```python
x = 1915

y = 'Albert'

print("{}  Einstein's  {} papers.".format(y, x))

# Output: Albert Einstein's 1915 papers.
```

Question 87

Question:

Write a program to check whether a given year is a leap year or not.

Solution:

```python
year=int(input("Enter the year to be checked: "))
if(year%4==0 and year%100!=0 or year%400==0):
    print("The", year, "is a leap year!")
else:
    print("The", year, "isn't a leap year!")
```

Question 88

Question:

Write a program that determines if a file path points to a file or a directory.

Solution:

```python
import os
path="1.txt"
if os.path.isdir(path):
    print("It is a directory")
elif os.path.isfile(path):
    print("It is a regular file")
else:
    print("It is a unique file (socket, FIFO, device file)" )
print()
```

```python
x = 1915

y = 'Albert'

print("{0}  Einstein's  {1} papers.".format(y, x))

# Output: Albert Einstein's 1915 papers.
```

Question 89

Question:

Write a program to generate all the divisors of an integer.

Solution:

```
x=int(input("Enter an integer: "))
print("The divisors of", x, "are: ")
for i in range(1, x+1):
    if(x%i==0):
        print(i)
```

```
x = 1915

y = 'Albert'

print("{1}  Einstein's  {0} papers.".format(y, x))

# Output: 1915  Einstein's  Albert papers.
```

Question 90

Question:

Write a program to print the table of a given number.

Solution:

```
n=int(input("Enter the number to print the tables for: "))
for i in range(1,11):
    print(n,"x",i,"=",n*i)
```

Question 91

Question:

Write a program to check if a number is an Armstrong number.

Solution:

```python
n=int(input("Enter any number: "))
a=list(map(int,str(n)))
b=list(map(lambda x:x**3,a))
if(sum(b)==n):
    print("The number", n, "is an armstrong number. ")
else:
    print("The number", n, "isn't an arsmtrong number. ")
```

```python
x = 1915

y = 'Albert'

print(f"{y}  Einstein's  {x} papers.")

# Output: Albert  Einstein's  1915 papers.
```

Question 92

Question:

Write a program to find Python site-packages.

Solution:

```python
import site
print(site.getsitepackages())
```

Question 93

Question:

Write a program to check if a number is a perfect number.

Solution:

```python
x = int(input("Enter any number: "))
sum = 0
for i in range(1, x):
    if(x % i == 0):
        sum = sum + i
if (sum == x):
    print("The number", x, "is a perfect number!")
else:
    print("The number", x, "is not a perfect number!")
```

```python
x = 1915

y = 'Albert'

print("{name}  Einstein's  {year} papers.".format(name = y, year = x))

# Output: Albert  Einstein's  1915 papers.
```

Question 94

Question:

Write a program to find the LCM of two numbers.

Solution:

```python
x=int(input("Enter the first number: "))
y=int(input("Enter the second number: "))
if(x>y):
    min=x
else:
    min=y
while(1):
    if(min%x==0 and min%y==0):
        print("LCM is:",min)
        break
    min=min+1
```

Question 95

Question:

Write a program to find the GCD of two numbers.

Solution:

```python
import math
x=int(input("Enter the first number: "))
y=int(input("Enter the second number: "))
print("The GCD of the two numbers is", math.gcd(x,y))
```

```python
x = 'Albert Einstein'

for i in x:

    if i == ' ':

        continue

    if i == 'n':

        break

    print(i)

print('NSTEIN')
```

```
# Output:

    A

    l

    b

    e

    r

    t

    E

    i

    NSTEIN
```

Question 96

Question:

Write a program to determine a file's size.

Solution:

```python
import os
x = os.path.getsize("1.csv")
print("The size of 1.csv is:", x, "Bytes")
print()
```

Question 97

Question:

Write a program to check if two numbers are amicable numbers.

Solution:

```python
x=int(input('Enter number 1: '))
```

```python
x = [
["John", 56],
["Albert", 77],
["Alan", 17],
["Mary", 39],
["James", 44],
]

print(type(x))
# Output: <class 'list'>

for i in x:
  print("{} {}".format(i[0], i[1]))
```

```
# Output:
        John 56
        Albert 77
        Alan 17
        Mary 39
        James 44
```

```
y=int(input('Enter number 2: '))

sum1=0

sum2=0

for i in range(1,x):

    if x%i==0:

        sum1+=i

for j in range(1,y):

    if y%j==0:

        sum2+=j

if(sum1==y and sum2==x):

    print('Amicable!')

else:

    print('Not Amicable!')
```

```
x = "Albert"

print("'{}'".format(x))

print("'{:12}'".format(x))

print("'{:<12}'".format(x))

print("'{:>12}'".format(x))

print("'{:^12}'".format(x))
```

```
# Output:

'Albert'

'Albert      '

'Albert      '

'      Albert'

'   Albert   '
```

Question 98

Question:

Write a program to find the area of a triangle given all three sides.

Solution:

```
import math

a=int(input("Enter first side: "))

b=int(input("Enter second side: "))

c=int(input("Enter third side: "))

s=(a+b+c)/2
```

```
area=math.sqrt(s*(s-a)*(s-b)*(s-c))
print("Area of the triangle is: ",round(area,2))
```

Question 99

Question:

```
x = "Albert"

print("{:s}".format(x))

# Output: Albert
```

Write a program to find the gravitational force acting between two objects.

Solution:

```
m1=float(input("Enter the first mass: "))
m2=float(input("Enter the second mass: "))
r=float(input("Enter the distance between the centers of the masses: "))
G=6.673*(10**-11)
f=(G*m1*m2)/(r**2)
print("Hence, the gravitational force is: ",round(f,2),"N")
```

Question 100

Question:

Write a program to determine if a string is numeric.

Solution:

```python
x = 'x549'
try:
    i = float(x)
    print('Numeric')
except (ValueError, TypeError):
    print('Not numeric')
print()
```

```python
a = 549.9678962314589

print("{:e}".format(a))

# Output: 5.499679e+02

print("{:E}".format(a))

# Output: 5.499679E+02

print("{:f}".format(a))

# Output: 549.967896

print("{:.2f}".format(a))

# Output: 549.97

print("{:F}".format(a))

# Output: 549.967896

print("{:g}".format(a))

# Output: 549.968

print("{:G}".format(a))

# Output: 549.968

print("{:n}".format(a))

# Output: 549.968
```

Question 101

Question:

Write a program to find out which host the routine is running on.

Solution:

```python
import socket
x = socket.gethostname()
print("Host name: ", x)
```

Question 102

Question:

Write a program to find the sum of first n positive integers.

Solution:

```python
n=int(input("Enter a number: "))
sum = 0
while(n > 0):
    sum=sum+n
    n=n-1
print("The sum of first n positive integers is: ", sum)
```

```python
n = int(input("Enter a number: "))
sum = (n * (n + 1)) / 2
print("The sum of first", n ,"positive integers is:", sum)
```

```python
def myfunc(x = "Paul"):
    print("Albert " + x)


myfunc("Einstein")
# Output: Albert Einstein
```

Question 103

Question:

Write a program to find the sum of series: 1 + 1/2 + 1/3 + ... + 1/n.

Solution:

```python
n=int(input("Enter the number of terms: "))
sum=0
for i in range(1,n+1):
    sum=sum+(1/i)
print("The sum of series is: ", round(sum,2))
```

```python
num = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']

print(num[::])

# Output: ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
```

Question 104

Question:

Write a program to get numbers divisible by 12 from a list using an anonymous function.

Solution:

```python
x = [55, 144, 72, 155, 120, 135, 540]
result = list(filter(lambda i: (i % 15 == 0), x))
print("Numbers divisible by 12 are: ", result)
```

Question 105

Question:

Write a program to format a given string with a string length limitation.

Solution:

```python
x = "987653421066"
print('%.5s' % x)
print('%.7s' % x)
print('%.9s' % x)
```

```python
pi = 3.141592653589793

r = 6

print(f"The value of PI is: '{pi:.4}'.")

# Output: The value of PI is: '3.142'.

print(f"The value of PI is: '{pi:.4f}'.")

# Output: The value of PI is: '3.1416'.

print(f"Area of a circle is: {pi * r ** 2}")

# Output: Area of a circle is: 113.09733552923255

print(f"Area of a circle is: {pi * r ** 2:.4f}")

# Output: Area of a circle is: 113.0973
```

Question 106

Question:

Write a program that accepts a number as input and returns an error if it is not a number.

Solution:

```
while True:
    try:
        a = int(input("Enter a number: "))
        print("This is a number")
        break
    except ValueError:
        print("This isn't a number")
        print()
```

```
x = 68

print("<%s>" % x)

# Output: <68>

print("<%10s>" % x)

# Output: <          68>

print("<%-10s>" % x)

# Output: <68          >

print("<%c>" % x)

# Output: D

print("<%d>" % x)

# Output: <68>

print("<%0.5d>" % x)

# Output: <00068>
```

Question 107

Question:

Write a program to filter the negative numbers from a list.

Solution:

```
x = [22, -10, 11, -28, 52, -75]
y = list(filter(lambda i: i <0, x))
print("Negative numbers in the above list: ", y)
```

```
num = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']

print(num[::1])

# Output: ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
```

Question 108

Question:

Write a program to find whether a number is a power of two.

Solution:

```python
def myfunc(n):
    """Return True if n is a power of two."""
    if n <= 0:
        return False
    else:
        return n & (n - 1) == 0


n = int(input('Enter a number: '))

if myfunc(n):
    print('{} is a power of two.'.format(n))
else:
    print('{} is not a power of two.'.format(n))
```

```python
x="Albert"

print(r"b\nc {x}")

# Output: b\nc {x}

print(rf"b\nc {x}")

# Output: b\nc Albert

print(fr"b\nc {x}")

# Output: b\nc Albert
```

```python
import array

x = array.array('i', [11,13,15,17,19,13])

print("Number of times the number 13 appears in the above array is: ", x.count(13))

    # Output: Number of times the number 13 appears in the above array is: 2
```

788

Question 109

Question:

Write a program to solve (a - b) * (a - b).

Solution:

```
a, b = 2, 4
result = a * a - 2 * a * b + b * b
print("({} - {}) ^ 2 = {}".format(a, b, result))
```

```
num = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']

print(num[::2])

# Output: ['1', '3', '5', '7', '9']
```

Question 110

Question:

Write a program to generate a new string with the prefix "Al" from a given string. Return the given text in its original form if it already contains the "Al" prefix.

Solution:

```
def myfunc(x):
  if len(x) >= 2 and x[:2] == "Al":
    return x
```

```
  return "Al" + x
print(myfunc("Albert"))
print(myfunc("bert"))
```

Question 111

```
num = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']

print(num[1::2])

# Output: ['2', '4', '6', '8', '10']
```

Question:

Write a program that count all occurrences of the number 5 in a list.

Solution:

```
def myfunc(y):
  i = 0
  for x in y:
    if x == 5:
      i = i + 1

  return i

print(myfunc([11, 5, 16, 18, 15]))
print(myfunc([17, 14, 5, 12, 9, 5]))
```

```
def myfunc(x):

    return 6 * x

print(myfunc(6))

# Output: 36
```

790

Question 112

Question:

Write a program to determine if a file is present.

Solution:

```
num = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']

print(num[2:8:2])

# Output: ['3', '5', '7']
```

```python
import os.path
print(os.path.isfile('1.txt'))
print(os.path.isfile('1.pdf'))
```

```
x = ['11', '12', '13', '14']

x[0] = '10'

print(x)

# Output: ['10', '12', '13', '14']
```

Question 113

Question:

Write a program to replace all occurrences of 'a' with '$' in a string.

Solution:

```python
x=input("Enter string: ")
x=x.replace('a','$')
x=x.replace('A','$')
print("Modified string: ")
```

```
print(x)
```

```
x = ['11', '12', '13', '14']

x[1:3] = ['alan', 'john']

print(x)

# Output: ['11', 'alan', 'john', '14']
```

Question 114

Question:

Write a program to calculate the product of a list of numbers (without using for loop).

Solution:

```
from functools import reduce
num = [2, 4, 10,11]
result = reduce( (lambda x, y: x * y), num)
print("Product of the above numbers is: ", result)
```

Question 115

```
num = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']

print(num[1:20:3])

# Output: ['2', '5', '8']
```

Question:

Write a program to detect if two strings are anagrams.

Solution:

```
x=input("Enter first string: ")
y=input("Enter second string: ")
if(sorted(x)==sorted(y)):
      print("The 2 strings are anagrams.")
else:
      print("The 2 strings aren't anagrams.")
```

```
x = ['11', '12', '13', '14']

x[1:3] = ['john']

print(x)

# Output: ['11', 'john', '14']
```

Question 116

Question:

Write a program to form a string where the first character and the last character have been exchanged.

Solution:

```
def change(x):
      return x[-1:] + x[1:-1] + x[:1]
x=input("Enter a string: ")
print("Modified string: ")
print(change(x))
```

```
x = ['100', '120', '130', '140']

x[1:2] = ['bat', 'ball']

print(x)

# Output: ['100', 'bat', 'ball', '130', '140']
```

Question 117

Question:

Write a program to count the number of vowels in a string.

Solution:

```
string=input("Enter string:")
vowels=0
for i in string:
    if(i=='a' or i=='e' or i=='i' or i=='o' or i=='u' or i=='A' or i=='E'
or i=='I' or i=='O' or i=='U'):
        vowels=vowels+1
print("Number of vowels are:")
print(vowels)
```

```
x = [11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]
print(x)
# Output: [11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]
x[1::2] = [10, 10, 10, 10, 10, 10]
print(x)
# Output: [11, 10, 13, 10, 15, 10, 17, 10, 19, 10, 21, 10]
```

Question 118

Question:

Write a program to take a string and replace every blank space with a hyphen.

Solution:

```
string=input("Enter a string: ")
string=string.replace(' ','-')
print("Modified string:")
print(string)
```

```
a = ['alan', 'john', 'mary', 'david']

b = a

a[0] = 'computer'

print(a)

# Output: ['computer', 'john', 'mary', 'david']

print(b)

# Output: ['computer', 'john', 'mary', 'david']
```

Question 119

Question:

Write a program to calculate the length of a string without using library functions.

Solution:

```
s=input("Enter string: ")
x=0
for i in s:
     x=x+1
print("Length of the string is: ")
print(x)
```

```
import array

x = array.array('i', [11,13,15,17,19,22])

print("Length of the array is: ", len(x))

                                              # Output: Length of the array is:  6
```

Question 120

Question:

Write a program to determine whether lowercase letters are present in a string.

Solution:

```
x = 'Albert Einstein'
print(any(i.islower() for i in x))
```

```
from copy import deepcopy

a = ['green', 'red', 'blue', 'orange']

b = deepcopy(a)

a[0] = 'albert'

print(a)

# Output: ['albert', 'red', 'blue', 'orange']

print(b)

# Output: ['green', 'red', 'blue', 'orange']
```

Question 121

Question:

Write a program to calculate the number of words and characters present in a string.

Solution:

```
x=input("Enter a string: ")
char=0
word=1
for i in x:
    char=char+1
```

```
x = ["albert", "1915", "john"]

y = ["albert", 1915, "john"]

print(":".join(x))

# Output: albert:1915:john
```

```
        if(i==' '):
                word=word+1
print("Number of words in the string: ")
print(word)
print("Number of characters in the string: ")
print(char)
```

```
x = ["albert", "1915", "john"]

y = ["albert", 1915, "john"]

print(":".join( map(str, y)))

# Output: albert:1915:john
```

Question 122

Question:

Write a program that rounds a floating-point integer to a specified number of decimal places.

Solution:

```
x = 549.968
print('%f' % x)
print('%.2f' % x)
print()
```

```
x = ["albert", "1915", "john"]

y = ["albert", 1915, "john"]

print(":".join(str(i) for i in y))

# Output: albert:1915:john
```

```
def myfunc(x,y):

    print(x+y)

myfunc("5","8")
```

```
# Output:  58
```

Question 123

Question:

Write a program to count number of lowercase characters in a string.

Solution:

```
x=input("Enter string: ")
count=0
for i in x:
        if(i.islower()):
                count=count+1
print("The number of lowercase characters is: ")
print(count)
```

```
print("xy:wz:pq".split(':'))

# Output: ['xy', 'wz', 'pq']
```

Question 124

Question:

Write a program to count the number of lowercase letters and uppercase letters in a string.

Solution:

```
x=input("Enter a string: ")
count1=0
count2=0
for i in x:
        if(i.islower()):
                count1=count1+1
        elif(i.isupper()):
                count2=count2+1
print("The number of lowercase characters is: ")
print(count1)
print("The number of uppercase characters is: ")
print(count2)
```

```
x = "bat ball bag".split()
print(x)
# Output: ['bat', 'ball', 'bag']
```

Question 125

Question:

Write a program to calculate the number of digits and letters in a string.

Solution:

```
x = ['apple', 'orange', 'mango', 'kiwi']
print(x.index('orange'))
# Output: 1
```

```
x=input("Enter a string: ")
count1=0
count2=0
for i in x:
        if(i.isdigit()):
```

```
            count1=count1+1
        count2=count2+1
print("The number of digits is: ")
print(count1)
print("The number of characters is: ")
print(count2)
```

```
x = ['apple', 'orange', 'mango', 'kiwi']

x.insert(2, 'fig')

print(x)

# Output: ['apple', 'orange', 'fig', 'mango', 'kiwi']
```

Question 126

Question:

Write a program to form a new string made of the first 2 characters and last 2 characters from a given string.

Solution:

```
x=input("Enter a string: ")
count=0
for i in x:
        count=count+1
new=x[0:2]+x[count-2:count]
print("Newly formed string is: ")
print(new)
```

```
x = ['apple', 'orange', 'mango', 'kiwi']

x.insert(len(x), 'papaya')

print(x)

# Output: ['apple', 'orange', 'mango', 'kiwi', 'papaya']
```

Question 127

Question:

Write a program to create a bytearray from a list.

Solution:

```
x = [10, 20, 56, 35, 17, 99]
# Create bytearray from list of integers.
y = bytearray(x)
for i in y: print(i)
print()
```

```
x = ['apple', 'orange', 'mango', 'kiwi', 'papaya']

x[2:4] = []

print(x)

# Output: ['apple', 'orange', 'papaya']
```

Question 128

Question:

Write a program to determine whether an integer fits in 64 bits.

Solution:

```
x = 60
if x.bit_length() <= 63:
```

```
    print((-2 ** 63).bit_length())
    print((2 ** 63).bit_length())
```

```
x = [17, 22, -44, 38, 6]

print(x)

# Output: [17, 22, -44, 38, 6]

x.sort(reverse=True)

print(x)

# Output: [38, 22, 17, 6, -44]
```

Question 129

Question:

Write a program that returns true if the two given integer values are equal, or if their sum or difference is 10.

Solution:

```python
def myfunc(a, b):
    if a == b or abs(a-b) == 10 or (a+b) == 10:
        return True
    else:
        return False
print(myfunc(17, 2))
print(myfunc(30, 20))
print(myfunc(5, 5))
print(myfunc(17, 13))
print(myfunc(53, 73))
```

```
x = [17, 22, -44, 38, 6]

print(x)

# Output: [17, 22, -44, 38, 6]

x.sort(key=abs, reverse=True)

print(x)

# Output: [-44, 38, 22, 17, 6]
```

Question 130

Question:

Write a program to add two objects if they are both of the integer type.

Solution:

```python
def myfunc(x, y):
    if not (isinstance(x, int) and isinstance(y, int)):
        return "Inputs have to be integers only!"
    return x + y
print(myfunc(50, 70))
print(myfunc(20, 50.74))
print(myfunc('6', 8))
print(myfunc('8', '8'))
```

```
A program to sort the list
according to length

x = ['alan', 'albert', 'james', 'bob']

x.sort(key=len)

print(x)

# Output: ['bob', 'alan', 'james', 'albert']
```

Question 131

Question:

Write a program to add leading zeros to a string.

Solution:

```
x='122.22'
x = x.ljust(8, '0')
print(x)
x = x.ljust(10, '0')
print(x)
```

```
x = ['alan', 'albert', 'james', 'bob']

x.sort(key=len, reverse=True)

print(x)

# Output: ['albert', 'james', 'alan', 'bob']
```

Question 132

Question:

Write a program that displays strings with double quotes.

Solution:

```
import json
print(json.dumps({'Albert': 1, 'Alan': 2, 'Alex': 3}))
```

Question 133

Question:

Write a program to check if a given key exists in a dictionary or not.

Solution:

```python
d={'A':1,'B':2,'C':3}
key=input("Enter key to check:")
if key in d.keys():
    print("Key is present and value of the key is:")
    print(d[key])
else:
    print("Key isn't present!")
```

```
for x in range(10, 19, 5):

    print(x)
```

```
# Output:

    10

    15
```

Question 134

Question:

Write a program to find the sum all the items in a dictionary.

Solution:

```python
d={'A':100,'B':540,'C':239}
print("Total sum of values in the dictionary is: ")
print(sum(d.values()))
```

Question 135

Question:

Write a program to multiply all the items in a dictionary.

Solution:

```
d={'A':10,'B':10,'C':239}
x=1
for i in d:
    x=x*d[i]
print(x)
```

```
x = ['albert', 'elsa']

y = ['david']

print(x)

# Output: ['albert', 'elsa']

print(x * 2)

# Output: ['albert', 'elsa', 'albert', 'elsa']

print(x + y)

# Output: ['albert', 'elsa', 'david']
```

Question 136

Question:

Write a program to remove the given key from a dictionary.

Solution:

```
d = {'a':1,'b':2,'c':3,'d':4}
print("Initial dictionary")
```

```
print(d)
key=input("Enter the key to delete(a-d):")
if key in d:
    del d[key]
else:
    print("Key not found!")
    exit(0)
print("Updated dictionary")
print(d)
```

```
a, b = 11, 12

print(a)

# Output: 11

print(b)

# Output: 12

a, b = b, a

print(a)

# Output: 12

print(b)

# Output: 11
```

Question 137

Question:

Write a program to list the home directory without using an absolute path.

Solution:

```
import os.path
print(os.path.expanduser('~'))
```

```
x = [5,8,57,35,44,14,28]

print([i for i in x if i%2!=0])
```

```
# Output:
        [5, 57, 35]
```

Question 138

Question:

Write a program to input two integers in a single line.

Solution:

```python
print("Enter the value of a and b: ")
a, b = map(int, input().split())
print("The value of a and b are: ", a, b)
```

```python
x = [0] * 5

for i in range(0, 5):

  print("{} {}".format(i, x[i]))
```

```
# Output:

    0 0

    1 0

    2 0

    3 0

    4 0
```

Question 139

Question:

Write a program to convert true to 1 and false to 0.

```python
print([2] * 5)

# Output: [2, 2, 2, 2, 2]
```

Solution:

```python
a = 'true'
a = int(a == 'true')
print(a)
a = 'xyz'
```

```
a = int(a == 'true')
print(a)
```

```
y = 'AB+CD+EF+GH+IJ+KL+MN'

x = y.split('+')

x.sort(key=len, reverse=True)

print(x)

# Output: ['AB', 'CD', 'EF', 'GH', 'IJ', 'KL', 'MN']
```

Question 140

Question:

Write a program to determine whether a variable is an integer or a string.

Solution:

```
print(isinstance(16,int) or isinstance(16,str))
print(isinstance("16",int) or isinstance("16",str))
```

Question 141

Question:

Write a program to count the number of vowels present in a string entered by the user using sets.

Solution:

```python
s=input("Enter a string: ")
count = 0
vowels = set("aeiou")
for letter in s:
    if letter in vowels:
        count += 1
print("The number of vowels present in a string is: ")
print(count)
```

```python
x= ['Alan Turing', 'Mary John']

y = ['David Hilbert', 'Joseph Sam']

x.extend(y)

print(x)

# Output: ['Alan Turing', 'Mary John', 'David Hilbert', 'Joseph Sam']
```

Question 142

Question:

Write a program to check common letters in the two input strings.

Solution:

```python
x=input("Enter the first string: ")
y=input("Enter the second string: ")
z=list(set(x)&set(y))
print("The common letters are: ")
for i in z:
    print(i)
```

Question 143

Question:

Write a program to display which letters are in the first string but not in the second string.

Solution:

```
x=input("Enter the first string: ")
y=input("Enter the second string: ")
z=list(set(x)-set(y))
print("The letters in the first string but not in the second string are: ")
for i in z:
    print(i)
```

Distributed computing refers to the use of multiple interconnected computers to solve a single problem or perform a single task. The goal of distributed computing is to leverage the processing power and storage capacity of multiple computers to achieve a common goal. In a distributed computing system, each computer works independently and cooperatively with the other computers in the network to share data, resources, and processing tasks.

Distributed computing systems can vary in size and complexity, ranging from small peer-to-peer networks to large-scale grid computing systems that involve thousands of computers. Examples of distributed computing systems include cloud computing platforms, content delivery networks, and distributed databases.

Distributed computing can offer a number of advantages over traditional centralized computing systems, such as increased processing power, scalability, fault tolerance, and reduced latency. However, it can also introduce challenges related to security, data consistency, and communication between the different nodes in the network.

Question 144

Question:

Write a program to determine whether a variable is a list, tuple, or set.

Solution:

```python
i = ['list', True, 8.9, 6]
if type(i) is list:
    print('i is a list')
elif type(i) is set:
    print('i is a set')
elif type(i) is tuple:
    print('i is a tuple')
else:
    print('Neither a set, list, or tuple.')
```

```python
import numpy as np

x = np.random.randint(10, size=(2, 3))

print(x)
```

?

Question 145

Question:

Write a program to determine whether a given number is even or odd recursively.

Solution:

```python
def myfunc(n):
    if (n < 2):
        return (n % 2 == 0)
    return (myfunc(n - 2))
n=int(input("Enter a number: "))
if(myfunc(n)==True):
    print("Number is even!")
else:
```

```python
x = {
"albert" : "46",
"bob" : "18",
"john" : "68",
}


for name, age in x.items():
  print("{} => {}".format(name, age))
```

```
# Output:

    albert => 46

    bob => 18

    john => 68
```

```
    print("Number is odd!")
```

Question 146

Question:

Write a program that examines a list of numbers to see if they are all distinct from one another.

```
x = {}

x['name'] = 'albert'

print(x)

# Output: {'name': 'albert'}

x['email'] = 'albert_john@gmail.com'

print(x)

# Output: {'name': 'albert', 'email': 'albert_john@gmail.com'}
```
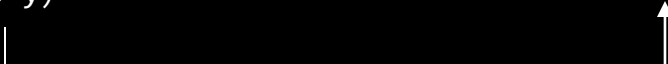
Solution:

```
def myfunc(x):
  if len(x) == len(set(x)):
    return True
  else:
    return False;
print(myfunc([11,15,17,19]))
print(myfunc([12,14,15,15,17,19]))
```

```
x = "Albert "

y = "Einstein"

print(x + y)
```

Output: Albert Einstein

Question 147

Question:

Write a program to add two positive numbers without using the '+' operator.

Solution:

```
def myfunc(x, y):
    while y != 0:
        z = x & y
        x = x ^ y
        y = z << 1
    return x
print(myfunc(12, 50))
```

```
x = lambda a, b: a * b

print(x(2, 2))

# Output: 4
```

```
x = {

'name': 'Albert',

'age': 26,

'email': None

}

print(x.get('name'))

# Output: Albert

print(x.get('age'))

# Output: 26

print(x.get('email'))

# Output: None

print(x.get('address'))

# Output: None
```

Question 148

Question:

Write a program to find the factorial of a number using recursion.

Solution:

```python
def myfunc(n):
    if(n <= 1):
        return 1
    else:
        return(n*myfunc(n-1))
n = int(input("Enter a number: "))
print("Factorial of", n, "is:", myfunc(n))
```

```python
x = {

'name': 'Albert',

'age': 26,

'email': None

}


print('name' in x.values())

# Output: False

print('Albert' in x.values())

# Output: True
```

Question 149

Question:

Write a program to determine whether a right triangle is formed by three given side lengths. If the specified sides make a right triangle, print "Yes," otherwise print "No."

Solution:

```python
print("Enter three side lengths of a triangle: ")
a,b,c = sorted(list(map(int,input().split())))
if a**2+b**2==c**2:
    print('Yes')
else:
    print('No')
```

Question 150

Question:

Write a program to find the number of equal numbers among three given integers.

Solution:

```python
def myfunc(a, b, c):
  result= set([a, b, c])
  if len(result)==3:
    return 0
  else:
    return (4 - len(result))

print(myfunc(11, 11, 11))
print(myfunc(11, 12, 12))
```

```python
x = {
'name': 'Albert',
'age': 26,
'email': None
}


print(x)


# Output: {'name': 'Albert', 'age': 26, 'email': None}
```

Question 151

Question:

Write a program to extract numbers from a given string.

Solution:

```
def myfunc(x):
    result = [int(x) for x in x.split() if x.isdigit()]
    return result
x = "5 bags, 10 pencils and 55 books"
print(myfunc(x))
```

Question 152

Question:

Write a program to get the smallest number from a list.

Solution:

```
def myfunc(list):
    x = list[0]
    for i in list:
        if i < x:
            x = i
    return x
print(myfunc([11, 22, -28, 3]))
```

```
x = {}

x['A'] = 11

x['B'] = 12

x['C'] = 13

x['D'] = 14

print(x)


# Output: {'A': 11, 'B': 12, 'C': 13, 'D': 14}
```

Question 153

Question:

Write a program to determine whether every string in a list is equal to a given string..

Solution:

```python
x = ["ball", "bat", "bag", "book"]
y = ["book", "book", "book", "book"]

print(all(i == 'bag' for i in x))
print(all(i == 'book' for i in y))
```

Question 154

Question:

Write a program to count the number of words in a text file.

```python
x = {

'name': 'Albert',

'email': 'albert_john@gmail.com'

}


for i in x:

  print("{} -> {}".format(i, x[i]))
```

```
# Output:

      name -> Albert

      email -> albert_john@gmail.com
```

Solution:

```python
x = input("Enter the file name: ")
num_words = 0
```

```
with open(x, 'r') as f:
    for line in f:
        num_words += len(line.split())
print("Number of words: ", num_words)
```

```
import numpy as np

x = np.random.random((3, 6))

print(x)
```

Question 155

Question:

Write a program to count the number of lines in a text file.

Solution:

```
x = input("Enter the file name: ")
num_lines = 0
with open(x, 'r') as f:
    for line in f:
        num_lines += 1
print("Number of lines: ", num_lines)
```

```
x = 11

y = 12.5

z = "Albert"

print(type(x)) # Output: <class 'int'>

print(type(y)) # Output: <class 'float'>

print(type(z)) # Output: <class 'str'>
```

Question 156

Question:

Write a program to create a list of empty dictionaries.

Solution:

```
print([{} for _ in range(10)])
```

Question 157

Question:

Write a program that computes the average of two lists.

```
x = set()

print(x)

# Output: set()

x.add('Mary')

print(x)

# Output: {'Mary'}

x.add('Mary')

print(x)

# Output: {'Mary'}

x.add('John')

print(x)

# Output: {'John', 'Mary'}
```

Solution:

```
def myfunc(x, y):
    result = sum(x + y) / len(x + y)
    return result

x = [11, 11, 13, 14, 14, 15, 16, 17]
y = [0, 11, 12, 13, 14, 14, 15, 17, 18]
```

```
print("The Average of two lists: ", myfunc(x, y))
```

Question 158

Question:

Write a program to determine the maximum and minimum value in the three given lists.

Solution:

```
x = [12,13,15,28,27,32,53]
y = [54,32,91,0,41,13,19]
z = [12,11,51,16,15,58,49]
print("Maximum value in the three given lists is: ")
print(max(x+y+z))
print("Minimum value in the three given lists is: ")
print(min(x+y+z))
```

```
x = int(5)
y = int(5.2)
w = float(5.6)
z = int("6")
print(x) # Output: 5
print(y) # Output: 5
print(w) # Output: 5.6
print(z) # Output: 6
```

Question 159

Question:

Write a program to delete empty lists from a given list of lists.

Solution:

```python
x = [[], [], [], ['Ball', 'Bag'], [11,12], ['Bat','Book'], []]
print([i for i in x if i])
```

Question 160

Question:

Write a program to determine whether or not every dictionary in a list is empty.

Solution:

```python
x = [{},{},{}]
y = [{2:6},{},{}]
print(all(not i for i in x))
print(all(not i for i in y))
```

```python
import numpy as np

x = np.eye(2)

print(x)

print()

y = np.eye(2, 4)

print(y)
```

```
# Output:

[[1. 0.]
 [0. 1.]]


[[1. 0. 0. 0.]
 [0. 1. 0. 0.]]
```

Question 161

Question:

Write a program that takes two lists and returns True when at least one of the elements in the lists is shared by both.

Solution:

```python
def myfunc(A, B):
    result = False
    for x in A:
        for y in B:
            if x == y:
                result = True
                return result
print(myfunc([21,22,23,24,25], [25,26,27,28,29]))
print(myfunc([31,32,33,34,35], [36,37,38,39]))
```

```python
x = 5

def myfunc(i):
    i = i+1
    return i


print(x)
# Output: 5
print(myfunc(x))
# Output: 6
print(x)
# Output: 5
```

```python
x = "Albert Einstein"

print(x[2:5]) # Output: ber
```

Question 162

Question:

Write a program to determine whether a list is empty or not.

Solution:

```
x = []

if x:
    print("List is not empty")
else:
    print("List is empty")
```

```
x = lambda a, b, c: a + b + c

print(x(2, 2, 2))

# Output: 6
```

Question 163

Question:

Write a program to read a file and capitalize the first letter of every word in the file.

Solution:

```
x = input("Enter the file name: ")
```

```
with open(x, 'r') as f:
    for line in f:
        print(line.title())
```

```
set(['John', 'Mary'])

x = set(['Mary', 'James', 'Alan'])

y = set(['Bob', 'Joe', 'Albert', 'Mary'])

x.update(y)

print(x)

# Output: {'Joe', 'Alan', 'Mary', 'James', 'Bob', 'Albert'}

print(y)

# Output: {'Joe', 'Bob', 'Mary', 'Albert'}
```

Question 164

Question:

Write a program to read the contents of a file in reverse order.

Solution:

```
x=input("Enter the file name: ")
for line in reversed(list(open(x))):
    print(line.rstrip())
```

```
def myfunc(a, b):
    c = a + b
    return c

print(myfunc(12, 33))
# Output: 45


print(myfunc(28, 93))
# Output: 121
```

Question 165

Question:

Write a program to decapitalize the first letter of a given string.

Solution:

```python
def myfunc(i, x = False):
    return ''.join([i[:1].lower(), (i[1:].upper() if x else i[1:])])
print(myfunc('Albert'))
print(myfunc('John'))
```

```
x :int = 6

print(x)

# Output: 6
```

Question 166

Question:

Write a program to remove spaces from a given string.

Solution:

```python
def myfunc(x):
    x = x.replace(' ','')
    return x

print(myfunc("a lbe rt ein stein"))
print(myfunc("a l a n"))
```

```
x :str = "Albert"

print(x)

# Output: Albert
```

Question 167

Question:

Write a program that calculate the difference between a given number and 10, returning double the absolute difference if the value is higher than 10.

Solution:

```
def myfunc(x):
    if x <= 10:
        return 10 - x
    else:
        return (x - 10) * 2

print(myfunc(8))
print(myfunc(16))
```

```
x :float = 26.69

print(x)

# Output: 26.69
```

Question 168

Question:

Write a program that adds three given numbers, returning three times their sum if the values are equivalent.

Solution:

```python
def myfunc(a, b, c):

    sum = a + b + c

    if a == b == c:
     sum = sum * 3
    return sum


print(myfunc(11, 12, 13))
print(myfunc(13, 13, 13))
```

```python
def myfunc(x :int, y :int) -> int:

  return x+y


print(myfunc(12, 13))

# Output: 25
```

Question 169

Question:

Write a program to check whether an OS is running a Python shell in 32 or 64 bit mode.

Solution:

```python
import platform
print(platform.architecture()[0])
```

Question 170

Question:

Write a program to implement birthday dictionary.

Solution:

```
if __name__ == '__main__':

    birthdays = {
        'Albert Einstein': '03/14/1879',
        'Benjamin Franklin': '01/17/1706',
        'Ada Lovelace': '12/10/1815',
        'Donald Trump': '06/14/1946',
        'Rowan Atkinson': '01/6/1955'}

    print('Welcome to the birthday dictionary. We know the birthdays of:')
    for name in birthdays:
        print(name)

    print('Who\'s birthday do you want to look up?')
    name = input()
    if name in birthdays:
        print('{}\'s birthday is {}.'.format(name, birthdays[name]))
    else:
        print('Sadly, we don\'t have {}\'s birthday.'.format(name))
```

```
import os

x = os.path.join('home', 'etc', 'files')

print(x)

# Output: home\etc\files
```

Question 171

Question:

Write a program to find the name and location of the file that is currently running.

Solution:

```python
import os
print("Current File Name : ", os.path.realpath(__file__))
```

```python
import platform

print(platform.system())

# Output: Windows

print(platform.release())

# Output: 10
```

Question 172

Question:

Write a program to implement password generator.

Solution:

```python
import random
x =
"abcdefghijklmnopqrstuvwxyz01234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$%^&*()?"
```

```
passlen = 8
print("".join(random.sample(x, passlen)))
```

Question 173

```
x = ["apple", "orange", "kiwi", "ball", "bat", "blackboard"]

print(list(filter(lambda i: len(i) == 4, x)))

# Output: ['kiwi', 'ball']
```

Question:

Write a program to display calendar of the given month and year.

Solution:

```
from functools import reduce

x = [11, 12, 13, 14]

print(reduce(lambda a, b: a+b, x))

# Output: 50

print(reduce(lambda a, b: a*b, x))

# Output: 24024
```

```
# importing calendar module
import calendar

yy = 2014   # year
mm = 11      # month

# To take month and year input from the user
# yy = int(input("Enter year: "))
# mm = int(input("Enter month: "))

# display the calendar
print(calendar.month(yy, mm))
```

Question 174

Question:

Write a program to eliminate a newline.

Solution:

```python
x='Albert Einstein\n'
print(x.rstrip())
```

```python
from functools import reduce

print(reduce(lambda a, b: a+b, [], 1))

# Output: 1

print(reduce(lambda a, b: a+b, [5, 5], 0))

# Output: 10

print(reduce(lambda a, b: a*b, [5, 6], 0))

# Output: 0

print(reduce(lambda a, b: a*b, [6, 5], 1))

# Output: 30

print(reduce(lambda a, b: a*b, [], 0))

# Output: 0
```

Question 175

Question:

Write a program to remove existing indentation from all of the lines in a given text.

Solution:

```python
import textwrap
x = '''
    Albert Einstein was a German-born theoretical physicist,
    widely acknowledged to be one of the greatest and most
    influential physicists of all time. Einstein is best known
    for developing the theory of relativity, but he also made
```

```
    important contributions to the development of the theory of

    quantum mechanics.
    '''
print(x)
print(textwrap.dedent(x))
```

Question 176

Question:

Write a program to count the number of CPUs being used.

Solution:

```
import multiprocessing
print(multiprocessing.cpu_count())
```

```
x = ['Mon', 'Tue', 'Wed', 'Thru', 'Fri']

y = [1, 2, 3, 4, 5]

print(zip(x, y))

?
```

Question 177

Question:

Write a program to reverse a string if its length is a multiple of 6.

Solution:

```python
def myfunc(x):
    if len(x) % 6 == 0:
        return ''.join(reversed(x))
    return x


print(myfunc('alan'))
print(myfunc('albert'))
```

```python
x = ['Mon', 'Tue', 'Wed', 'Thru', 'Fri']

y = [1, 2, 3, 4, 5]

print(list(zip(x, y)))

# Output: [('Mon', 1), ('Tue', 2), ('Wed', 3), ('Thru', 4), ('Fri', 5)]
```

Question 178

Question:

Put "xyz" at the end of the specified string (length should be at least 3). If the provided string already ends with "xyz," add "123" to it. If the specified string's length is less than three, leave it unaltered.

Solution:

```python
def myfunc(x):
  i = len(x)

  if i > 2:
    if x[-3:] == 'xyz':
      x += '123'
```

```python
import numpy as np

x= np.zeros(5, dtype='float32')

print(x)

# Output: [0. 0. 0. 0. 0.]

print(x.dtype)

# Output: float32
```

```python
    else:
        x += 'xyz'

    return x

print(myfunc('xy'))
print(myfunc('xyz'))
print(myfunc('morning'))
```

```python
x = ['Mon', 'Tue', 'Wed', 'Thru', 'Fri']

y = [1, 2, 3, 4, 5]

print(dict(zip(x, y)))

# Output: {'Mon': 1, 'Tue': 2, 'Wed': 3, 'Thru': 4, 'Fri': 5}
```

Question 179

Question:

Write a program to shuffle the elements of a given list.

Solution:

```python
import random
x = [11, 12, 13, 14, 15]
random.shuffle(x)
print(x)
```

```python
import numpy as np

x = np.array([6, 8, 10], dtype='int8')

print(x /2)

# Output: [3. 4. 5.]

print(x.dtype)

# Output: int8
```

Question 180

Question:

Three positive numbers are present in a list. Write a program to determine whether the sum of the digits in each number is equal or not. If true, return true; otherwise, return false.

Solution:

```python
def myfunc(x):
    return x[0] % 9 == x[1] % 9 == x[2] % 9
x = [14, 5, 23]
print(myfunc(x))
```

```python
x = [True, True]
y = [True, False]
z = [False, False]
print(all(x))
# Output: True
print(all(y))
# Output: False
print(all(z))
# Output: False
print()
print(any(x))
# Output: True
print(any(y))
# Output: True
print(any(z))
# Output: False
```

Question 181

Question:

Write a program to get IP address of your computer.

Solution:

```python
import socket
x = socket.gethostname()
```

```
y = socket.gethostbyname(x)
print("Your Computer Name is: " + x)
print("Your Computer IP Address is: " + y)
```

Question 182

Question:

Write a program to determine whether a series of integers has an increasing trend or not.

Solution:

```
def myfunc(x):
    if (sorted(x) == x):
        return True
    else:
        return False

print(myfunc([11,12,13,14]))
print(myfunc([11,12,15,13,14]))
```

```
print(12 > 10)

# Output: True

print(10 > 15)

# Output: False
```

```
x = ["alan", "john", "albert"]

x.clear()

print(x) # Output: []
```

Question 183

Question:

Write a program to illustrate Dice Roll Simulator.

Solution:

```python
import random
min = 1
max = 6

roll_again = "yes"

while roll_again == "yes" or roll_again == "y":
    print ("Rolling the dices...")
    print ("The values are....")
    print (random.randint(min, max))
    print (random.randint(min, max))

    roll_again = input("Roll the dices again?")
```

```python
x = [12, 14]

print(all(map(lambda i: i > 5, x)))

# Output: True

print(all(map(lambda i: i > 13, x)))

# Output: False
```

Question 184

Question:

Write a program to convert the temperature in degree centigrade to Fahrenheit.

Solution:

```python
c = input(" Enter temperature in Centigrade: ")
f = (9*(int(c))/5)+32
print(" Temperature in Fahrenheit is: ", f)
```

Question 185

Question:

Write a program to check whether the given integer is a multiple of 5.

```python
import numpy as np

x = np.array([13, 14, 17])

print(x)

# Output: [13 14 17]

print(x * 3)

# Output: [39 42 51]

print(x + 4)

# Output: [17 18 21]

print(x.dtype)

# Output: int32
```

Solution:

```python
x = int(input("Enter an integer: "))
if(x%5==0):
    print(x, "is a multiple of 5")
else:
    print(x, "is not a multiple of 5")
```

Question 186

Question:

Write a program to check whether the given integer is a multiple of both 3 and 5.

Solution:

```python
x = int(input("Enter an integer: "))
if((x%3==0)and(x%5==0)):
    print(x, "is a multiple of both 3 and 5")
else:
    print(x, "is not a multiple of both 3 and 5")
```

Question 187

Question:

Write a program to display all the multiples of 5 within the range 10 to 70.

Solution:

```python
for i in range(10,70):
    if (i%5==0):
        print(i)
```

```python
x = [11, 12, 13, 450]

for i in x:

    print(i)

print(x)
```

?

Question 188

Question:

Write a program to display all integers within the range 50-100 whose sum of digits is an even number.

Solution:

```
for i in range(50,100):
    num = i
    sum = 0
    while(num!=0):
        digit = num%10
        sum = sum + digit
        num = num//10
    if(sum%2==0):
        print(i)
```

```
x = lambda a: a + 5

print(x(2))

# Output: 7
```

Question 189

Question:

Write a program to print the numbers from a given number n till 0 using recursion.

Solution:

```
def myfunc(n):
    if (n==0):
        return
    print(n)
    n=n-1
    myfunc(n)
```

```
from functools import reduce

x = [12, 11, 14, 31]

print(reduce(lambda a,b: a if a < b else b, x))

# Output: 11 (minimum)

print(reduce(lambda a,b: a if a > b else b, x))

# Output: 31 (maximum)
```

```
myfunc(9)
```

```
from functools import reduce

x = 5

print(reduce(lambda a,b: a*b, range(1, x+1), 1))

# Output: 120 (factorial of 5)
```

Question 190

Question:

Write a program to find the odd numbers in an array.

Solution:

```
num = [8,3,1,6,2,4,5,9]
count = 0
for i in range(len(num)):
    if(num[i]%2!=0):
        count = count+1
print("The number of odd numbers in the array are: ", count)
```

Question 191

Question:

Write a program to reverse a given upper case string in lower case.

Solution:

```python
def myfunc(x):
    return x[::-1].lower()
x = "JAVASCRIPT"
print(myfunc(x))
```

```python
import numpy as np

x = np.array([

[ 11, 12, 13, 14, 15],

[ 12, 13, 14, 15, 16]

])

print(x)

print(x * 6)

print(x + 5)
```

?
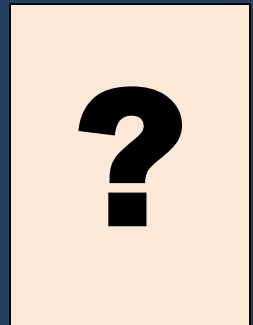
Question 192

Question:

Write a program to find the maximum of two numbers.

Solution:

```python
def maximum(a, b):

    if a >= b:
        return a
    else:
        return b

a = 3
b = 5
print(maximum(a, b))
```

```python
import itertools

for i in itertools.count(start=10, step=1):

    print(i)

    if i > 15:

        break
```

```
# Output:
    10

    11

    12

    13

    14

    15

    16
```

Solution:

843
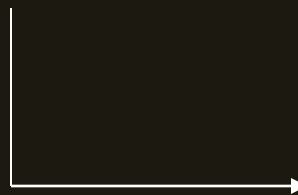
```python
a = 3
b = 5

print(max(a, b))
```

Solution:

```python
a = 3
b = 5

print(a if a >= b else b)
```

```python
from collections.abc import Iterator, Iterable

from types import GeneratorType


print(issubclass(GeneratorType, Iterator))

# Output: True

print(issubclass(Iterator, Iterable))

# Output: True
```

- A generator is an iterator
- An iterator is an iterable

Question 193

Question:

Write a program to find the minimum of two numbers.

Solution:

```python
def minimum(a, b):

    if a <= b:
        return a
    else:
        return b
```

```python
x = [11, 11, 12, 13, 15, 18, 19, 35, 64]

y = x[2:5]

print(y)

# Output: [12, 13, 15]

x[2] = 99

print(x)

# Output: [11, 11, 99, 13, 15, 18, 19, 35, 64]

print(y)

# Output: [12, 13, 15]
```

```
a = 3
b = 9
print(minimum(a, b))
```

Question 194

Question:

Write a program to calculate Profit or Loss.

```
import itertools

i = 0

for x in itertools.cycle(['A', 'B', 'C']):

    print(x)

    i = i+1

    if i >= 4:

        break

print('')
```

```
# Output:

    A

    B

    C

    A
```

Solution:

```python
cp=float(input("Enter the Cost Price : "));
sp=float(input("Enter the Selling Price : "));
if cp==sp:
    print("No Profit or No Loss")
else:
    if sp>cp:
        print("Profit of ",sp-cp)
    else:
        print("Loss of ",cp-sp)
```

```
x = [11, 12, 13, 14]

for i in x:

    print(i)

    if i == 12:

        x.remove(12)

print(x)
```

```
# Output:

    11

    12

    14

    [11, 13, 14]
```

Question 195

Question:

Write a program to find Student Grade.

Solution:

```python
physics = float(input(" Please enter Physics Marks: "))
math = float(input(" Please enter Math score: "))
chemistry = float(input(" Please enter Chemistry Marks: "))


total = physics + math + chemistry
percentage = (total / 300) * 100


print("Total Marks = %.2f"  %total)
print("Percentage = %.2f"  %percentage)


if(percentage >= 90):
    print("A Grade")
elif(percentage >= 80):
    print("B Grade")
elif(percentage >= 70):
    print("C Grade")
elif(percentage >= 60):
    print("D Grade")
elif(percentage >= 40):
    print("E Grade")
else:
    print("Fail")
```

```python
def myfunc():
    yield 24
    yield 48
    yield 64


x = myfunc()
print(type(x))
# Output: <class 'generator'>
print(next(x))
# Output: 24
print(next(x))
# Output: 48
print(next(x))
# Output: 64
```

```python
x = 5

y = 100

print("X") if x > y else print("Y")

# Output: Y
```

Question 196

Question:

Write a program to print Even numbers from 1 to N.

Solution:

```
x = int(input(" Enter the Value of N : "))

for num in range(1, x+1):
    if(num % 2 == 0):
        print("{0}".format(num))
```

Question 197

Question:

Write a program to print Odd numbers from 1 to N.

Solution:

```
x = int(input(" Enter the Value of N : "))

for num in range(1, x+1):
    if(num % 2 != 0):
        print("{0}".format(num))
```

```
def myfunc():

    yield 24

    yield 48

    yield 64


x = myfunc()

print(type(x))

for i in x:

    print(i)
```

```
# Output:
<class 'generator'>
24
48
64
```

Question 198

Question:

Write a program to compute sum of Even numbers from 1 to N.

Solution:

```python
x = int(input(" Enter the Value of N : "))
total = 0

for num in range(1, x+1):
    if(num % 2 == 0):
        print("{0}".format(num))
        total = total + num

print("The Sum of Even Numbers from 1 to {0} is: {1}".format(num, total))
```

```python
def myfunc(name):
    def x():
        print(f"Albert {name}")
    return x

y = myfunc("Einstein")
y()
# Output: Albert Einstein
```

Question 199

Question:

Write a program to compute sum of Odd numbers from 1 to N.

Solution:

```
x = int(input(" Enter the Value of N : "))
total = 0

for num in range(1, x+1):
    if(num % 2 != 0):
        print("{0}".format(num))
        total = total + num

print("The Sum of Odd Numbers from 1 to {0} is: {1}".format(num, total))
```

Question 200

Question:

Write a program to check whether a character is Alphabet or not.

Solution:

```
ch = input("Enter a Character : ")

if((ch >= 'a' and ch <= 'z') or (ch >= 'A' and ch <= 'Z')):
    print(ch, "is an Alphabet.")
else:
    print(ch, "is Not an Alphabet.")
```

```
x = 21

y = 5

z = 35

if x > y and z > x:

   print("Both conditions are satisfied")

# Output: Both conditions are satisfied
```

Question 201

Question:

Write a program to check whether a character is Alphabet or Digit or Special Character.

Solution:

```python
ch = input("Enter a Character : ")

if((ch >= 'a' and ch <= 'z') or (ch >= 'A' and ch <= 'Z')):
    print(ch, "is an Alphabet.")
elif(ch >= '0' and ch <= '9'):
    print(ch, "is a Digit.")
else:
    print(ch, "is a Special Character.")
```

```python
say = print

say("Albert Einstein")

# Output: Albert Einstein
```

Question 202

Question:

Write a program to check whether a character is Lowercase or not.

Solution:

```python
ch = input("Enter a Character : ")
```

```
if(ch.islower()):
    print(ch, "is a Lowercase character")
else:
    print(ch, "is Not a Lowercase character")
```

```
def myfunc():

  x = 2

  yield x


  x += 2

  yield x


  x += 2

  yield x


for i in myfunc():

  print(i)
```

Question 203

Question:

Write a program to check whether a character is Uppercase or not.

Solution:

```
ch = input("Enter a Character : ")

if(ch.isupper()):
    print(ch, "is a Uppercase character")
else:
    print(ch, "is Not a Uppercase character")
```

```
# Output:
    2
    4
    6
```

Question 204

Question:

Write a program to check whether a character is Vowel or Consonant.

Solution:

```python
ch = input("Enter a Character : ")

if(ch == 'a' or ch == 'e' or ch == 'i' or ch == 'o' or ch == 'u' or ch == 'A'
        or ch == 'E' or ch == 'I' or ch == 'O' or ch == 'U'):
    print(ch, "is a Vowel")
else:
    print(ch, "is a Consonant")
```

Question 205

Question:

Write a program to convert string to Uppercase.

Solution:

```python
str = input("Enter a String : ")

string = str.upper()

print("String in Lowercase  =  ", str)
print("String in Uppercase =  ", string)
```

```python
def myfunc():
    x = 1
    while True:
        yield x
        x += 1


for i in myfunc():
    print(i)
    if i >= 5:
        break
```

```
# Output:
    1
    2
    3
    4
    5
```

Question 206

Question:

Write a program to convert string to Lowercase.

Solution:

```python
str = input("Enter a String : ")

string = str.lower()

print("String in Uppercase  =  ", str)
print("String in Lowercase =  ", string)
```

Question 207

Question:

Write a program to convert Decimal to Binary, octal, and Hexadecimal.

Solution:

```python
decimal = int(input("Enter a Decimal Number: "))

binary = bin(decimal)
```

```python
def myfunc():

  i = 0

  i += 2

  return i


print(myfunc())

print(myfunc())

print(myfunc())

# Output:
        2

        2

        2
```
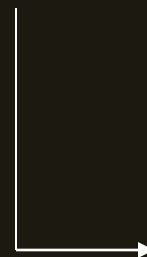
```
octal = oct(decimal)
hexadecimal = hex(decimal)

print(decimal, " Decimal Value = ", binary, "Binary Value")
print(decimal, " Decimal Value = ", octal, "Octal Value")
print(decimal, " Decimal Value = ", hexadecimal, "Hexadecimal Value")
```

Question 208

Question:

Write a program to check a Triangle is Valid or Not.
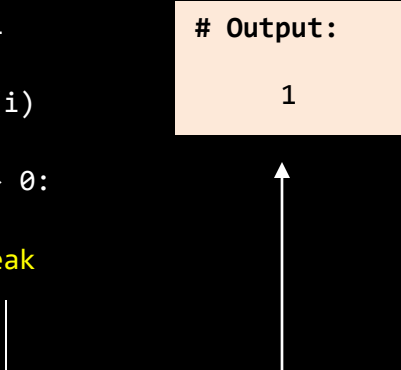
Solution:

```
a = int(input('Please Enter the First Angle of a Triangle: '))
b = int(input('Please Enter the Second Angle of a Triangle: '))
c = int(input('Please Enter the Third Angle of a Triangle: '))


total = a + b + c


if total == 180:
    print("\nThis is a Valid Triangle")
else:
    print("\nThis is an Invalid Triangle")
```

```
i = 0

while True:

    i += 1                    # Output:

    print(i)                       1

    if i > 0:

        break
```

Question 209

Question:

Write a program that inputs an age and print age after 20 years .

Solution:

```
age = int(input("What is your age? "))
print("In twenty years, you will be", age + 20, "years old!")
```

Question 210

Question:

Write a program to print the number of seconds in year.

Solution:

```
days=365
hours=24
minutes=60
seconds=60
print("Number of seconds in a year : ",days*hours*minutes*seconds)
```

```
x = [11, 12, 13, 14]
for i in x[:]:
    print(i)
    if i == 12:
        x.remove(12)
print(x)
```

```
# Output:
11
12
13
14
[11, 13, 14]
```

Question 211

Question:

Write a program that inputs a string and then prints it equal to number of times its length.

Solution:

```
str = input("Enter string: ")
b = len(str)
a = str * b
print(a)
```

```
i = 0

def myfunc():
  global i
  i += 1
  return i

print(myfunc())

print(myfunc())

print(myfunc())
```

```
# Output:
1
2
3
```

Question 212

Question:

Write a program to convert a given list of strings and characters to a single list of characters.

Solution:

```
def myfunc(x):
    result = [i for y in x for i in y]
    return result
```

```
x = ["alan", "john", "w", "p", "james", "q"]
print(myfunc(x))
```

```
x = 22

y = 5

z = 73

if x > y or x > z:

   print("At least one of the conditions is satisfied")



# Output: At least one of the conditions is satisfied
```

```
x = 1
while x < 8:

   print(x)

   if (x == 3):

     break

   x = x+1
```

```
# Output:
    0
    1
    2
    3
```

```
x = 0

while x < 8:

  x = x+1

  if (x == 3):

    continue

  print(x)
```

Output

```
1
2
4
5
6
7
8
```

```
say = print

def mult(x, y):

    return x * y


product = mult

say(product(6, 6))

# Output: 36
```

```
import numpy as np

x = np.array([11, 11, 12, 13, 15, 18, 23, 31, 65])

print(x)

# Output: [11 11 12 13 15 18 23 31 65]

print(x[4])

# Output: 15

print(x[2:5])

# Output: [12 13 15]
```

```
import numpy as np

x = np.array([[2, 4, 6], [8, 10, 12]])

y = np.array([[3, 6, 12], [15, 18, 21]])

print(x)
```

```
# Output:
[[ 2  4  6]
 [ 8 10 12]]
```

```
print(y)
```

```
# Output:
[[ 3  6 12]
 [15 18 21]]
```

```
print(x*y)
```

```
# Output:
[[  6  24  72]
 [120 180 252]]
```

```
print(np.multiply(x, y))
```

```
# Output:

[[  6  24  72]

 [120 180 252]]
```

```
import numpy as np

x = np.array([True, True, False])

print(x.dtype)

# Output: bool

print(x)

# Output: [ True  True False]
```

```
import numpy as np

x = np.array(['Albert', 'James', 'Mary', 'Einstein', 'Bob'])

print(x)

# Output: ['Albert' 'James' 'Mary' 'Einstein' 'Bob']

print(np.vectorize(len)(x))

# Output: [6 5 4 8 3]
```

```
def myfunc():

    return 26

print(myfunc())

print(myfunc())

print(myfunc())
```

```
# Output:

    26

    26

    26
```

```python
a = ['pqrs', 'wxyz']

print(a)

# Output: ['pqrs', 'wxyz']

print(a[0:1])

# Output: ['pqrs']

print(a[0])

# Output: pqrs

print(a[0][0])

# Output: p

print(a[0][1])

# Output: q

print(a[0][0:2])

# Output: pq
```

```python
def myfunc():

    return 26

    return 27

    return 28


print(myfunc())

print(myfunc())

print(myfunc())
```

```
# Output:
    26
    26
    26
```

```python
import numpy as np

x = np.array(['Albert', 'James', 'Mary', 'Einstein', 'Bob'])

print( [(len(i), i) for i in x])

# Output: [(6, 'Albert'), (5, 'James'), (4, 'Mary'), (8, 'Einstein'), (3, 'Bob')]
```

```python
import multiprocessing as mp

print(mp.cpu_count()) # Multiprocess CPU count

# Output: 4
```

860

```python
x = complex(5, 6)

print(x)

# Output: (5+6j)

print(x.real)

# Output: 5.0

print(x.imag)

# Output: 6.0

print((-1) ** 0.5)

# Output: (6.123233995736766e-17+1j)

print(complex(0, 1))

# Output: 1j

print(complex(0, 1) ** 2)

# Output: (-1+0j)
```

```python
def f(x, y = []):

    y.append(x)

    return y


print(f(11))

print(f(12))

print(f(13))
```

```
# Output:

[11]

[11, 12]

[11, 12, 13]
```

```python
def f(x, y = None):

    if y == None:

        y = []

    y.append(x)

    return y


print(f(11))

print(f(12))

print(f(13))
```

```
# Output:

          [11]

          [12]

          [13]
```
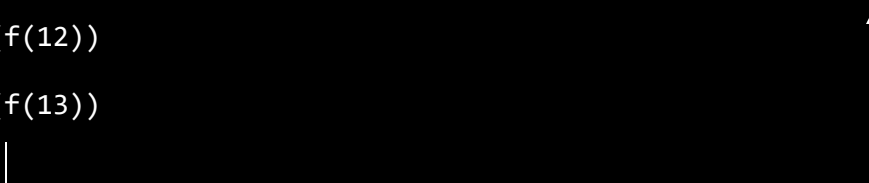
```
x = 59

def f():

    print(x)


f()

# Output: 59
```

```
import pandas

x = pandas.Series([11, 11, 12, 13, 15, 18])

print(x)

print(x.values)

# Output: [11 11 12 13 15 18]

print(x.sum())

# Output: 80

print(x.count())

# Output: 6

print(x.mean())

# Output: 13.333333333333334

print(x.median())

# Output: 12.5

print(x.std())

# Output: 2.7325202042558927
```

```
# Output:

0     11

1     11

2     12

3     13

4     15

5     18

dtype: int64
```

```
import numpy as np

x = np.array([112], 'uint8')

print(x.dtype)

# Output: uint8

print(x)

# Output: [112]

x[0] += 1

print(x)

# Output: [113]

x[0]-=1

print(x)

# Output: [112]

x[0]= 126

print(x)

# Output: [126]

x[0]+=1

print(x)

# Output: [127]
```

```
for _ in range(10):
    print('Albert')
```

?
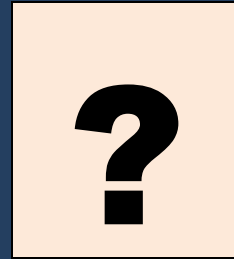
```
import numpy as np

x = np.array([11, 12, 13])

y = np.array([14, 15, 16])

z = np.array([17, 18, 19])

print(x)

# Output: [11 12 13]

print(y)

# Output: [14 15 16]

print(z)

# Output: [17 18 19]

w = np.hstack([x, y])

print(w)

# Output: [11 12 13 14 15 16]

q = np.hstack([w, z])

print(q)

# Output: [11 12 13 14 15 16 17 18 19]
```

```
for i in [1, 2, 3]:

  print(i)

  print('Albert' * i)
```

```
# Output:
```

```
1

Albert

2

AlbertAlbert

3

AlbertAlbertAlbert
```

```
print(6, type(6))

# Output: 6 <class 'int'>

print(6.8, type(6.8))

# Output: 6.8 <class 'float'>

print(-6.8, type(-6.8))

# Output: -6.8 <class 'float'>

print([], type([]))

# Output: [] <class 'list'>

print(False, type(False))

# Output: False <class 'bool'>

print(None, type(None))

# Output: None <class 'NoneType'>
```

```
a = 0

b = 1

for i in [2, 4, 8]:

  a = a + b*i

  b = b + 1

print(a)

# Output: 34
```

```
def x(i):

    return 5*i

b = 3

print(x(b) + x(3*b-1))

# Output: 55
```

864

```
print(format(.1, '.20f'))

# Output: 0.10000000000000000555

print(format(.2, '.20f'))

# Output: 0.20000000000000001110

print(format(.1 + .2, '.20f'))

# Output: 0.30000000000000004441

print(format(.3, '.20f'))

# Output: 0.29999999999999998890
```

```
print('str: {}, int: {}, formatted float: {:.3f}.'.format('Bob', 53, 6.1687))
```

str: Bob, int: 53, formatted float: 6.169.

Image processing refers to the use of algorithms and techniques to analyze, transform, enhance, and manipulate digital images. It involves the application of various mathematical and computational methods to extract useful information from an image, correct image defects, and create new images. **Image processing techniques** can be used for a wide range of applications, including medical imaging, remote sensing, surveillance, computer vision, and digital photography. Some common tasks performed by image processing algorithms include:

- **Image filtering:** This involves the removal of noise, enhancement of image features, and smoothing of edges.
- **Image segmentation:** This involves dividing an image into meaningful regions, based on properties such as color, texture, and intensity.
- **Image compression:** This involves reducing the size of an image while preserving its quality, to save storage space and reduce transmission time.
- **Image recognition:** This involves the automatic identification of objects, patterns, and features in an image, using machine learning and computer vision techniques.
- **Image restoration:** This involves the recovery of lost or damaged image information, such as removing scratches, stains, and other defects.

Overall, **image processing** plays an important role in many fields, enabling researchers and professionals to extract valuable insights and information from digital images.

LINUX − OVERVIEW

Linux is an operating system that is free and open-source and is based on the **Unix system**. It was first released in 1991 by **Linus Torvalds** and has since become one of the most widely used operating systems in the world, powering everything from mobile devices to supercomputers. **Linux** is renowned for its dependability, flexibility, and safety. It is highly customizable and can be used for a wide range of applications, from servers and supercomputers to desktops and laptops. **Linux distributions**, or "**distros"**, are created by different organizations and individuals to cater to different needs and use cases. One of the key features of **Linux** is its command-line interface, which allows users to interact with the system using text commands. This can be daunting for new users, but it offers a high degree of control and customization that is not available in other operating systems. Many **Linux distros** also come with **graphical user interfaces** (GUIs) that make it easier for beginners to use. Another key feature of **Linux** is its package management system, which allows users to easily install and manage software packages from online repositories. This makes it easy to keep the system up-to-date and install new software. **Linux** is also known for its security features, including built-in firewalls, encryption, and permission-based access control. This makes it a popular choice for servers and other applications where security is a top priority. Overall, **Linux** is a powerful and flexible operating system that offers many advantages over other operating systems. While it can be challenging to learn at first, it offers a high degree of control and customization that is not available in other operating systems, which makes it an appealing option for programmers, system administrators, and other power users.

Linux is a free and open-source operating system that has many advantages over proprietary operating systems like **Windows** and **macOS**. Here are some of the main advantages of **Linux**:

- **Cost:** Linux is free to use and distribute, which makes it a cost-effective option for both personal and business use. You don't need to purchase a license or pay for updates, which can save a significant amount of money over time.

- **Flexibility:** Linux is highly customizable and can be tailored to meet the specific needs of users. This allows users to create a personalized computing environment that suits their workflow and preferences.

- **Stability:** Linux is renowned for its dependability and durability. It rarely crashes or freezes, and is not vulnerable to viruses or malware as Windows is. This makes it an ideal choice for mission-critical applications and servers.

- **Security:** Linux is more secure than other operating systems because of its built-in security features, such as secure boot, firewall, and file permissions. Additionally, since Linux is open-source, security experts around the world can contribute to identifying and fixing vulnerabilities, making it more secure than proprietary operating systems.

- **Performance:** Linux is renowned for its quick responsiveness and little resource needs. It can run on older hardware and doesn't require high-end hardware to run smoothly.

- **Open-source community:** Linux has a large and active community of developers and users who contribute to its development and support. This means that there is a wealth of online resources, documentation, and forums available to help users solve problems and get the most out of Linux.

- **Compatibility:** Linux is compatible with a wide range of hardware, making it a versatile choice for users who have older or specialized hardware.

- **Free software:** Linux comes with a wide range of free and open-source software, including productivity tools, media players, and development tools. This means that users can save money on software licenses and have access to high-quality software for free.

Overall, **Linux** is a powerful, customizable, and cost-effective operating system that offers many advantages over proprietary operating systems. It's an ideal choice for users who value performance, stability, security, and flexibility in their computing environment. While **Linux** has many advantages, there are also some disadvantages to using it. Here are some of the main disadvantages of Linux:

- **Lack of user-friendly interface:** While the command-line interface of Linux provides greater control and flexibility, it can be difficult for beginners to use. The graphical user interface of some Linux distributions may also be less polished and intuitive than those of Windows or macOS.

- **Limited software availability:** While there is a wide range of software available for Linux, it can still be more limited than what is available for Windows or macOS. Some specialized software, such as Adobe Photoshop or Microsoft Office, may not be available for Linux.

- **Compatibility issues:** Linux may not be compatible with all hardware or software. This can be especially problematic for users who need to use specific hardware or software for their work or hobbies.

- **Fragmentation:** The open-source nature of Linux means that there are many different distributions and versions available. This can lead to fragmentation in the community and make it more difficult for users to find support or find the right distribution for their needs.

- **Learning curve:** Because Linux is different from **Windows and macOS**, there can be a steep learning curve for new users. Users may need to learn new commands and methods for accomplishing tasks that are familiar on other operating systems.

- **Lack of professional support:** While there are many online resources and forums available to help Linux users, there may be a lack of professional support options for users who need more advanced help or support.

- **Gaming support:** While gaming on Linux has improved significantly in recent years, it still lags behind Windows in terms of support and compatibility for many games.

- **Security:** While Linux is generally considered to be more secure than **Windows**, it is not immune to security vulnerabilities or malware. Linux users must still take appropriate security precautions to protect their systems.

Overall, **Linux** can be a great operating system for many users, but it may not be the best fit for everyone. It's important to carefully consider the advantages and disadvantages of **Linux** before deciding whether to use it.

Here are some funny facts about Linux:
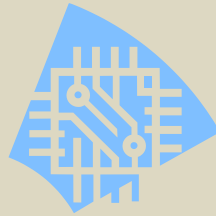
- The original name for Linux was **"Freax"**, which was a combination of "free", "freak", and "Unix". The name was changed to Linux by the creator of the kernel, Linus Torvalds.

- The Linux mascot is a penguin named **Tux**. The original concept art for Tux was created by **Larry Ewing** in 1996.

- The first ever recorded message sent from space to Earth was sent using Linux. The message was sent by astronaut David A. Wolf, who used a laptop running Linux to send the message to Mission Control.

- There is a Linux distribution called **"Gentoo"** that is designed to be compiled from source code by the user. This can take hours or even days, but it allows for a highly customized and optimized system.

- The creator of Linux, Linus Torvalds, once said that he named his kernel **"Linux"** because it sounded better than **"Freax"** and because he wanted to "annoy" the creator of the Unix operating system, which he was inspired by.

- In 2012, a group of programmers from the Netherlands created a fully functioning car that was powered by Linux. The car was named "Edison", and it was able to drive up to 60 miles per hour.

- There is a Linux distribution called **"Damn Small Linux"** that can fit on a business card-sized CD-ROM.

- In 2016, the Linux kernel reached over 20 million lines of code, making it one of the largest software projects in the world.

- The operating system on the International Space Station (ISS) is based on Linux.

- The Linux operating system has been used to power everything from smartphones and tablets to supercomputers and servers. It's even used to power the majority of the world's top 500 supercomputers.

C − OVERVIEW

C programming is a popular programming language that was first developed by **Dennis Ritchie at Bell Labs in 1972**. C is a general-purpose programming language that is widely used for developing operating systems, system software, and embedded systems, as well as applications in various fields such as finance, engineering, and gaming. Here are some key features of the C programming language:

- **Simple and easy to learn:** C has a simple and concise syntax, making it easy for beginners to learn and use.

- **Portable:** C code can be compiled and run on different platforms, including Windows, macOS, Linux, and embedded systems.

- **Low-level programming:** C allows programmers to write code that interacts directly with hardware and memory, making it ideal for developing system software and drivers.

- **Fast and efficient:** C is a fast and efficient programming language that can be used to develop high-performance applications.

- **Modular programming:** C supports modular programming, allowing programmers to divide code into smaller, more manageable functions and modules.

- **Standard library:** C comes with a standard library of functions that provide commonly used functionality, such as input/output, string handling, and mathematical operations.

- **Pointers:** C allows programmers to work with pointers, which are memory addresses that point to specific data in memory. Pointers are a powerful feature of C that allow for more efficient memory management and manipulation.

- **Preprocessor directives:** C includes preprocessor directives, which are special statements that are processed by the compiler before the code is compiled. Preprocessor directives are used to define constants, include header files, and perform other tasks.

Overall, C is a **powerful and versatile programming language** that is widely used for developing a variety of applications, including system software, embedded systems, and high-performance applications. It is a well-liked choice among developers worldwide because of its portability, effectiveness, and simplicity.

Here's an example of a **"Hello, World!"** program in C:

```c
#include<stdio.h>

int main() {

printf("Hello, World!\n");

return 0;

}
```

Explanation:

- **#include<stdio.h>** − This line includes the standard input and output library, which contains the **printf() function** used to display output.

- **int main()** − This line defines the **main() function**, which is the entry point of the program.

- **{}** − The curly braces contain the code that will be executed when the program runs.

- **printf("Hello, World!\n");** − This line uses the **printf() function** to display the text **"Hello, World!"** to the console. The **\n** is a special character that represents a newline, so the text will be displayed on a new line.

- **return 0;** − This line returns a value of **0** from the **main() function**, indicating that the program executed successfully.

When you run this program, it will display the text **"Hello, World!"** on the console. This program is often used as a simple test to make sure that a new programming environment is set up correctly.

In **C programming language**, comments are used to explain the code and make it more readable for other developers. Comments are ignored by the compiler and do not affect the functionality of the code. There are two types of comments in C: **single-line comments** and **multi-line comments**.

- **Single-line comments** start with two forward slashes (**//**) and continue until the end of the line. For example:

```c
// This is a single-line comment
int x = 5; // This line initializes the variable x to 5
```

- **Multi-line comments** start with **/*** and end with ***/**. Everything between these two symbols is considered a comment. For example:

```c
/*
This is a multi-line comment
It can span multiple lines
*/
int y = 10; /* This line initializes the variable y to 10 */
```

Comments are important for making the code more understandable and easier to maintain. They can also be used to temporarily remove a section of code during testing or debugging. For example:

```c
// int z = someFunction(); // This line is temporarily commented out for testing purposes
int a = 10; // This line initializes the variable a to 10
```

In the above example, the line that calls the function **someFunction()** has been commented out for testing purposes. This allows the developer to test the rest of the code without the potentially

problematic function call. Once the testing is complete, the comment can be removed and the code will work as intended.

In **C programming language**, a variable is a named memory location that stores a value of a specific data type. The value of a variable can be changed during program execution. Variables are declared with a data type, a name, and an optional initial value. Here are some examples of variable declarations in C:

```c
int x; // declares an integer variable named x
float y = 3.14; // declares a floating-point variable named y and initializes it to 3.14
char z = 'A'; // declares a character variable named z and initializes it to 'A'
```

In the above examples, **int**, **float**, and **char** are data types in C. The variable **x** is declared without an initial value, while **y** and **z** are declared with initial values. Variables can be assigned new values using the assignment operator (**=**). For example:

```c
x = 5; // assigns the value 5 to the variable x
y = 2.5; // assigns the value 2.5 to the variable y
z = 'B'; // assigns the value 'B' to the variable z
```

Variables can also be used in expressions. For example:

```c
int a = 10;
int b = 5;
int c = a + b; // assigns the value 15 to the variable c
```

In the above example, the variables **a** and **b** are used in an expression to calculate the value of the variable **c**. In C, variables have a scope, which defines the parts of the program where the

variable can be accessed. Variables declared inside a function have local scope, which means they can only be accessed within that function. Variables declared outside of any function have global scope, which means they can be accessed from any part of the program. To sum up, variables in C are named memory locations that store values of specific data types. They can be declared with or without initial values, assigned new values, and used in expressions. Understanding variables is essential for writing C programs.

In **C programming language**, data types specify the type of data that can be stored in a variable. The format specifiers are used to indicate the type of data being printed or scanned in input/output operations. Here are the commonly used data types and format specifiers in C:

Data Types:

- **int:** used to store integer values
- **float:** used to store floating-point values
- **double:** used to store double-precision floating-point values
- **char:** used to store a single character
- **bool:** used to store boolean values (true or false)
- **short:** used to store small integer values
- **long:** used to store large integer values
- **long long:** used to store very large integer values

Format Specifiers:

- **%d:** used to print or scan integer values
- **%f:** used to print or scan floating-point values
- **%lf:** used to print or scan double-precision floating-point values
- **%c:** used to print or scan a single character
- **%s:** used to print or scan a string of characters
- **%u:** used to print or scan unsigned integer values
- **%ld:** used to print or scan long integer values
- **%lld:** used to print or scan very long integer values

- **%x**: used to print or scan hexadecimal values

Here are some examples of using data types and format specifiers in C:

```c
#include<stdio.h>

int main() {
int x = 10;
float y = 3.14;
char z = 'A';

printf("The value of x is %d\n", x);
printf("The value of y is %f\n", y);
printf("The value of z is %c\n", z);

scanf("%d", &x);
scanf("%f", &y);
scanf("%c", &z);

printf("You entered %d\n", x);
printf("You entered %f\n", y);
printf("You entered %c\n", z);

return 0;
}
```

In the above example, the **int**, **float**, and **char** data types are used to declare variables **x**, **y**, and **z**. The **%d**, **%f**, and **%c** format specifiers are used to print or scan these variables. The **scanf** function is used to scan input from the user, and the **printf** function is used to print output to the screen. Understanding data types and format specifiers is essential for working with variables and input or output operations in C.

In **C programming language**, a constant is a value that cannot be changed during program execution. Constants are typically used to represent fixed values that are used in a program. Constants can be of different data types, such as integer, floating-point, character, and boolean. Here are some examples of defining constants in C:

```c
#define PI 3.14159 // defines a constant PI with the value 3.14159
const int MAX = 100; // defines a constant MAX with the value 100
enum { FALSE, TRUE } BOOL; // defines boolean constants FALSE and TRUE
```

In the above examples, **#define**, **const**, and **enum** are used to define constants in C. **#define** is a preprocessor directive that defines a constant value that can be used throughout the program. **const** is a keyword that defines a constant variable that cannot be modified. **enum** is a keyword that defines a set of named integer constants. Here are some examples of using constants in C:

```c
#include<stdio.h>

#define PI 3.14159
const int MAX = 100;

int main() {
float radius = 5.0;
float circumference = 2 * PI * radius;
int count = MAX;

printf("The circumference of the circle is %f\n", circumference);
printf("The maximum count is %d\n", count);

return 0;
}
```

In the above example, the constant **PI** is used to calculate the circumference of a circle, and the constant **MAX** is used to set the maximum count. By using constants instead of hard-coded values, the program becomes more readable and easier to maintain. Constants in C are typically defined at the beginning of a program or in a separate header file. By convention, constant names are written in uppercase letters to distinguish them from variables. Overall, constants in C are values that cannot be changed during program execution. They can be defined using **#define**, **const**, or **enum**, and are typically used to represent fixed values in a program. Understanding constants is essential for writing C programs that are readable and maintainable.

In **C programming language**, a string is a sequence of characters stored in a contiguous memory location. Strings in C are represented as arrays of characters, terminated by a null character **'\0'**. Here is an example of a string in C:

```c
char str[] = "Hello, World!";
```

In this example, the string "Hello, World!" is stored in the array **str**. The null character '\0' is automatically added to the end of the string to indicate its termination. Strings in C can be manipulated using various string functions, such as **strlen()**, **strcpy()**, and **strcat()**. Here are some examples of using string functions in C:

```c
#include<stdio.h>
#include<string.h>

int main() {
char str1[] = "Hello";
char str2[] = "World";
char buffer[50];

// find the length of a string
printf("The length of %s is %d\n", str1, strlen(str1));
```

```c
// concatenate two strings
strcpy(buffer, str1);
strcat(buffer, " ");
strcat(buffer, str2);
printf("The concatenated string is %s\n", buffer);

// compare two strings
if (strcmp(str1, str2) == 0) {
    printf("%s and %s are equal\n", str1, str2);
} else {
    printf("%s and %s are not equal\n", str1, str2);
}
return 0;
}
```

In this example, the **strlen()** function is used to find the length of a string, the **strcpy()** and **strcat()** functions are used to concatenate two strings, and the **strcmp()** function is used to compare two strings. C also provides various ways to input and output strings. The **scanf()** function is used to input strings from the standard input, while the **printf()** function is used to output strings to the standard output. Here is an example of using **scanf()** and **printf()** to input and output strings:

```c
#include<stdio.h>

int main() {
char name[50];
printf("Enter your name: ");
scanf("%s", name);
printf("Hello, %s!\n", name);
return 0;
}
```

In this example, the **scanf()** function is used to input a string from the standard input and store it in the array **name**, while the **printf()** function is used to output a greeting message that includes the input string. To sum up, strings in C are represented as arrays of characters terminated by a null character '\0'. String functions can be used to manipulate strings, and various ways to input and output strings are available in C. Understanding strings is essential for writing C programs that deal with text data.

In **C programming**, operators are symbols that represent actions or computations. They allow you to perform mathematical or logical operations on one or more operands. Here are the most commonly used operators in C:

1. **Arithmetic Operators**

   Arithmetic operators are used to perform mathematical operations on numeric operands. These include:

   - **Addition (+):** Adds two operands
   - **Subtraction (-):** Subtracts the second operand from the first operand
   - **Multiplication (*):** Multiplies two operands
   - **Division (/):** Divides the first operand by the second operand
   - **Modulus (%):** Returns the remainder of a division operation

   Example:

```c
int a = 5, b = 2, c;
c = a + b;  // c is now 7
c = a - b;  // c is now 3
c = a * b;  // c is now 10
c = a / b;  // c is now 2
c = a % b;  // c is now 1
```

2. **Assignment Operators**

Assignment operators are used to assign values to variables. These include:

- **Assignment (=):** Assigns the value of the right operand to the left operand
- **Addition assignment (+=):** Adds the value of the right operand to the left operand and assigns the result to the left operand
- **Subtraction assignment (-=):** Subtracts the value of the right operand from the left operand and assigns the result to the left operand
- **Multiplication assignment (*=):** Multiplies the left operand by the right operand and assigns the result to the left operand
- **Division assignment (/=):** Divides the left operand by the right operand and assigns the result to the left operand
- **Modulus assignment (%=):** Calculates the remainder of the left operand divided by the right operand and assigns the result to the left operand

Example:

```
int a = 5, b = 2;
a += b;  // a is now 7
a -= b;  // a is now 5 again
a *= b;  // a is now 10
a /= b;  // a is now 5 again
a %= b;  // a is now 1
```

3. **Comparison Operators**

Comparison operators are used to compare two values. They return a Boolean value (true or false). These include:

- **Equal to (==):** Returns true if the operands are equal
- **Not equal to (!=):** Returns true if the operands are not equal
- **Greater than (>):** Returns true if the first operand is greater than the second operand
- **Less than (<):** Returns true if the first operand is less than the second operand

- **Greater than or equal to (>=):** Returns true if the first operand is greater than or equal to the second operand
- **Less than or equal to (<=):** Returns true if the first operand is less than or equal to the second operand

Example:

```c
int a = 5, b = 2;
if (a == b) {
  printf("a is equal to b\n");
} else {
  printf("a is not equal to b\n");
}
if (a > b) {
  printf("a is greater than b\n");
}
if (a < b) {
  printf("a is less than b\n");
}
```

4. **Logical Operators**

Logical operators are used to combine multiple conditions. They return a Boolean value (true or false). These include:

- **AND (&&):** Returns true if both operands are true

Example:

```c
int a = 5, b = 2;
if (a > 0 && b > 0) {
  printf("Both a and b are positive\n");
}
```

In this example, the condition inside the **if** statement will be true only if both **a** and **b** are greater than 0.

- **OR (||):** Returns true if either operand is true

Example:

```c
int a = 5, b = 2;
if (a > 0 || b > 0) {
  printf("At least one of a and b is positive\n");
}
```

In this example, the condition inside the **if** statement will be true if either **a** or **b** is greater than 0.

- **NOT (!):** Returns the opposite of the operand's value

Example:

```c
int a = 5, b = 2;
if (!(a > b)) {
  printf("a is not greater than b\n");
}
```

In this example, the condition inside the **if** statement will be true only if **a** is not greater than **b**. The **NOT** operator reverses the result of the comparison operator (**>**), so if **a** is not greater than **b**, the **NOT** operator will return true.

We can also combine multiple logical operators to form complex conditions. For example:

```c
int a = 5, b = 2, c = 7;
if (a > 0 && b > 0 || c > 0) {
  printf("At least one of a and b is positive or c is positive\n");
}
```

In this example, the condition inside the **if** statement will be true if either **a** or **b** is greater than 0, or if **c** is greater than 0. The **AND** operator has higher precedence than the **OR** operator, so the condition inside the parentheses is evaluated first.

C language does not have a built-in Boolean data type. However, **C99** introduced a new data type called **_Bool** (or **bool**) which can be used to represent Boolean values. Additionally, the header file **stdbool.h** provides a macro **bool** which can be used instead of **_Bool**. Here is an example of using **bool** in C:

```c
#include<stdbool.h>
#include<stdio.h>

int main() {
bool a = true;
bool b = false;

if (a && !b) {
    printf("a is true and b is false\n");
  }

return 0;
}
```

In this example, we have defined two Boolean variables **a** and **b**. The values of these variables are assigned using the keywords **true** and **false**, which are defined in the **stdbool.h** header file. We then use these variables in an **if** statement, where we use the "**&&**" (AND) operator and the "**!**"(NOT) operator to create a Boolean expression.

The output of this program will be:

```
a is true and b is false
```

Note that any nonzero value is considered to be true in C, while the value 0 is considered to be false. Therefore, you can also use integers to represent Boolean values in C:

```c
#include <stdio.h>

int main() {
int a = 1;
int b = 0;

if (a && !b) {
    printf("a is true and b is false\n");
  }

  return 0;
}
```

In this example, we have defined two integer variables **a** and **b**. We then use these variables in an **if** statement to create a Boolean expression. Since **a** is nonzero and **b** is 0, the expression evaluates to true and the message is printed. However, this approach can be less clear and more error-prone than using **bool** variables, especially in more complex programs.

> "Any fool can write code that a computer can understand. Good programmers write code that humans can understand."
>
> — Martin Fowler

C program to add two numbers:

```c
#include<stdio.h>

int main() {

int num1, num2, sum;

printf("Enter two numbers to add:\n");

scanf("%d%d", &num1, &num2);

sum = num1 + num2;

printf("The sum of %d and %d is %d\n", num1, num2, sum);

return 0;

}
```

Here's how the program works:

- First, we include the standard input and output library header file **stdio.h.**

- We declare three integer variables **num1**, **num2**, and **sum**.

- We prompt the user to enter two numbers to add using the **printf** function.

- We use the **scanf** function to read in the two numbers entered by the user and store them in the variables **num1** and **num2**.

- We add the two numbers together and store the result in the variable **sum**.

- Finally, we use the **printf** function again to print out the sum of the two numbers.

When the program runs, it will prompt the user to enter two numbers to add. After the user enters the numbers, the program will add them together and display the result.

C program to check whether a number is even or odd:

```c
#include<stdio.h>
int main() {
int num;
printf("Enter an integer:\n");
scanf("%d", &num);
if (num % 2 == 0) {
printf("%d is even\n", num);
} else {
printf("%d is odd\n", num);
}
return 0;
}
```

Here's how the program works:

- First, we include the standard input and output library header file **stdio.h**.

- We declare an integer variable **num**.

- We prompt the user to enter an integer using the **printf** function.

- We use the **scanf** function to read in the integer entered by the user and store it in the variable **num**.

- We use an if-else statement to check if **num** is even or odd. The **%** operator is used to find the remainder of **num** divided by 2. If the remainder is 0, then **num** is even, and the program will print out a message saying so. If the remainder is not 0, then **num** is odd, and the program will print out a message saying so.

- Finally, we use the **return** statement to exit the program.

When the program runs, it will prompt the user to enter an integer. After the user enters the integer, the program will check if it's even or odd and display a message accordingly.

C program to print "Hello World" 10 times on the console screen:

```c
#include<stdio.h>
int main() {
int i;
for (i = 0; i < 10; i++) {
printf("Hello World\n");
}
return 0;
}
```

Here's how the program works:

- First, we include the standard input and output library header file **stdio.h**.

- We declare an integer variable **i**.

- We use a **for** loop to print "Hello World" ten times. The loop runs from **i = 0** to **i < 10**, incrementing **i** by 1 each time the loop runs. Inside the loop, we use the **printf** function to print "Hello World" to the screen.

- Finally, we use the **return** statement to exit the program.

When the program runs, it will print "Hello World" ten times to the screen. The **for** loop allows us to repeat the printing of "Hello World" multiple times without having to write out the same **printf** statement multiple times.

```c
#include<stdio.h>
int main() {
int i = 0;
while (i < 10) {
printf("Hello World\n");
i++;
}
return 0;
}
```

Here's how the program works:

- First, we include the standard input and output library header file **stdio.h**.

- We declare an integer variable **i** and initialize it to 0.

- We use a **while** loop to print "Hello World" ten times. The loop runs as long as **i** is less than 10. Inside the loop, we use the **printf** function to print "Hello World" to the screen and then increment **i** by 1.

- Finally, we use the **return** statement to exit the program.

When the program runs, it will print "Hello World" ten times to the screen using the **while** loop. The **while loop** is another way to repeat a set of statements multiple times, as long as the condition specified in the loop header is true. In this case, the loop runs as long as **i** is less than 10, allowing us to print "Hello World" ten times without having to write out the same **printf** statement multiple times.

```c
#include<stdio.h>
int main() {
int i = 0;
do {
printf("Hello World\n");
i++;
} while (i < 10);
return 0;
}
```

Here's how the program works:

- First, we include the standard input and output library header file **stdio.h**.

- We declare an integer variable **i** and initialize it to 0.

- We use a **do-while** loop to print "Hello World" ten times. The loop runs at least once, even if the condition in the loop header is false. Inside the loop, we use the **printf** function to print "Hello World" to the screen and then increment **i** by 1.

- The loop condition is checked at the end of each iteration of the loop. If the condition is true, the loop continues to run, and if the condition is false, the loop exits.

- Finally, we use the **return** statement to exit the program.

When the program runs, it will print "Hello World" ten times to the screen using the **do-while loop**. The **do-while** loop is another way to repeat a set of statements multiple times, but it guarantees that the loop body will be executed at least once, even if the loop condition is false. In this case, the loop runs as long as **i** is less than 10, allowing us to print "Hello World" ten times without having to write out the same **printf** statement multiple times.

In **C programming language**, `switch` statement is used to execute a block of code depending on the value of an expression. The `switch` statement works by evaluating the expression and then executing the code associated with the matching `case` label. If no `case` label matches the value of the expression, then the `default` code block is executed.

Here's an example of how to use switch statement in C:

```c
#include<stdio.h>
int main() {
char operator;
int operand1, operand2, result;
printf("Enter an operator (+, -, *, /): ");
scanf("%c", &operator);
printf("Enter two operands: ");
scanf("%d %d", &operand1, &operand2);

switch(operator) {
    case '+':
        result = operand1 + operand2;
        printf("%d + %d = %d\n", operand1, operand2, result);
        break;
    case '-':
        result = operand1 - operand2;
        printf("%d - %d = %d\n", operand1, operand2, result);
        break;
    case '*':
        result = operand1 * operand2;
        printf("%d * %d = %d\n", operand1, operand2, result);
        break;
    case '/':
        if(operand2 == 0) {
            printf("Error: Cannot divide by zero\n");
```

```
    } else {
            result = operand1 / operand2;
            printf("%d / %d = %d\n", operand1, operand2, result);
        }
        break;
    default:
        printf("Error: Invalid operator\n");
        break;
    }


    return 0;
}
```

In this example, the program asks the user to enter an operator **(+, -, *, /)** and two operands. The program then uses a **switch** statement to perform the appropriate arithmetic operation based on the operator entered by the user. If an invalid operator is entered, the program displays an error message. The **switch** statement begins with the expression **operator** which is evaluated to determine which **case** label matches. The **case** labels **+, -, *,** and **/** are associated with the appropriate arithmetic operations. The **default** label is used to handle any case where the expression does not match any of the other **case** labels. Each **case** label contains a block of code that is executed when the expression matches that label. The **break** statement is used to exit the **switch** statement after the matching **case** label is executed. In this example, if the user enters the operator **+**, the block of code associated with the case '+' label is executed. This block of code adds the two operands and displays the result. Similarly, the other arithmetic operations are handled by their respective **case** labels. If an invalid operator is entered, the **default** label is executed, displaying an error message. Overall, the **switch** statement provides a convenient way to perform different actions based on the value of a single expression, making it a useful construct in many C programs.

In **C programming language**, **break** and **continue** are two control flow statements that are used to modify the execution of loops (such as **for**, **while**, and **do-while** loops) and switch statements. **break** statement is used to terminate the execution of the loop or switch statement immediately. When a **break** statement is encountered, the control is transferred to the next statement after the loop or switch statement. Here is an example:

```c
#include<stdio.h>
int main() {
int i;
for(i = 1; i <= 10; i++) {
    if(i == 5) {
        break;
        }
printf("%d\n", i);
}
return 0;
}
```

In this example, the program uses a **for** loop to print the values of **i** from 1 to 10. However, when **i** equals 5, the **break** statement is executed, and the loop is terminated immediately. Therefore, the program only prints the values of **i** from 1 to 4, and the loop does not complete. On the other hand, the **continue** statement is used to skip the current iteration of the loop and move to the next iteration. When a **continue** statement is encountered, the control is transferred to the next iteration of the loop. Here is an example:

```c
#include <stdio.h>
int main() {
int i;

for(i = 1; i <= 10; i++) {
    if(i == 5) {
        continue;
```

```
    }
        printf("%d\n", i);
    }


    return 0;
}
```

In this example, the program uses a **for** loop to print the values of **i** from 1 to 10. However, when **i** equals 5, the **continue** statement is executed, and the current iteration of the loop is skipped. Therefore, the program does not print the value 5, but continues with the next iterations and prints the other values. Both **break** and **continue** statements are useful for controlling the flow of execution in loops and switch statements, allowing programs to skip certain iterations or terminate the loop altogether based on certain conditions.

In **C programming language**, an array is a collection of elements of the same data type, stored in contiguous memory locations. Arrays are a powerful and efficient way to store and manipulate large amounts of data in C programs. Here is an example of how to use arrays in C:

```c
#include<stdio.h>

int main() {
int numbers[5]; // declare an array of 5 integers
int i;

    // initialize the array elements
    numbers[0] = 1;
    numbers[1] = 3;
    numbers[2] = 5;
    numbers[3] = 7;
    numbers[4] = 9;
```

```
    // print the array elements
    for(i = 0; i < 5; i++) {
        printf("%d ", numbers[i]);
    }
    return 0;
}
```

In this example, an array named **numbers** is declared, which can store up to 5 integers. The elements of the array are accessed using an index, starting from 0. In this case, the elements of the array are initialized with the values 1, 3, 5, 7, and 9. To access an element of the array, we can use its index as shown in the **for** loop. In this loop, we print the array elements using the **printf** statement, with the **%d** format specifier indicating that we are printing an integer. Another way to initialize an array is to use an initializer list, which allows you to specify the values of the array elements at the time of declaration. Here is an example:

```
#include<stdio.h>
int main() {
int numbers[] = {1, 3, 5, 7, 9}; // declare and initialize an array
int i;

// print the array elements
for(i = 0; i < 5; i++) {
    printf("%d ", numbers[i]);
}
return 0;
}
```

In this example, the array **numbers** is declared and initialized using an initializer list. The size of the array is automatically determined by the number of elements in the list. The output of this program is the same as the previous example. Arrays are a fundamental data structure in C, and

their simplicity and efficiency make them a popular choice for many programming tasks. They are widely used in applications such as sorting, searching, and data processing, among others.

In **C programming language**, a structure is a collection of variables of different data types, grouped together under a single name. Structures are used to represent complex data types and can be used to organize and manipulate large amounts of data in C programs. Here is an example of how to use structures in C:

```c
#include<stdio.h>

// define a structure named "student"
struct student {
    char name[50];
    int roll_no;
    float marks;
};

int main() {
    // declare a structure variable of type "student"
    struct student s1;
    // assign values to the structure members
    strcpy(s1.name, "John");
    s1.roll_no = 1;
    s1.marks = 85.5;
    // print the structure members
    printf("Name: %s\n", s1.name);
    printf("Roll No.: %d\n", s1.roll_no);
    printf("Marks: %f\n", s1.marks);
    return 0;
}
```

In this example, a structure named "student" is defined using the **struct** keyword. The structure has three members: a character array **name**, an integer **roll_no**, and a float **marks**. The **main** function declares a structure variable **s1** of type **student** and assigns values to its members using the "**.**" operator. The **printf** statements then print the values of the structure members. Another way to declare and initialize a structure variable is to use an initializer list, which allows you to specify the values of the structure members at the time of declaration. Here is an example:

```c
#include<stdio.h>

// define a structure named "book"

struct book {
    char title[50];
    char author[50];
    float price;
};

int main() {

    // declare and initialize a structure variable of type "book"

    struct book b1 = {"The C Programming Language", "Brian Kernighan and Dennis Ritchie", 29.99};

    // print the structure members
    printf("Title: %s\n", b1.title);
    printf("Author: %s\n", b1.author);
    printf("Price: %f\n", b1.price);

    return 0;
}
```

In this example, a structure named "book" is defined with three members: a character array **title**, a character array **author**, and a float **price**. The **main** function declares and initializes a structure variable **b1** of type **book** using an initializer list. The **printf** statements then print the values of the structure members. Structures are a powerful feature of C programming language that allows programmers to create custom data types that can be used to represent complex data structures. They are widely used in applications such as database management, file handling, and graphics programming, among others.

In **C programming language**, a pointer is a variable that stores the memory address of another variable. Pointers allow you to manipulate data directly in memory, which can be useful for tasks such as dynamically allocating memory or working with complex data structures. Here's an example of how pointers work:

```c
#include<stdio.h>
int main() {
int var = 10;  /* actual variable declaration */
int *p;  /* pointer variable declaration */

p = &var; /* store address of var in pointer variable */

printf("Address of var variable: %p\n", &var);

/* address stored in pointer variable */
printf("Address stored in p pointer variable: %p\n", p);

/* access the value using the pointer */
printf("Value of var: %d\n", *p);

return 0;
}
```

In this example, we declare an integer variable **var** and a pointer variable **p**. We then store the address of **var** in **p** using the **&** operator. We can then access the value of **var** using the ***** operator on the pointer variable **p**. When we run this program, it will output the following:

```
Address of var variable: 0x7ffeb2e3b9ac
Address stored in p pointer variable: 0x7ffeb2e3b9ac
Value of var: 10
```

Here, the **%p** format specifier is used to print the memory addresses. The addresses of **var** and **p** are the same, as expected, since **p** points to **var**. Pointer arithmetic is another important aspect of pointers in C. You can perform arithmetic operations on pointers such as addition, subtraction, and comparison. Here's an example:

```c
#include<stdio.h>
int main() {
int arr[] = {10, 20, 30, 40, 50};
int *p = arr;
for(int i=0; i<5; i++) {
    printf("Value of arr[%d]: %d\n", i, *(p+i));
}
return 0;
}
```

In this example, we declare an integer array **arr** and a pointer variable **p** initialized to the first element of the array. We then use pointer arithmetic to iterate through the array and print out the values. When we run this program, it will output the following:

```
Value of arr[0]: 10
Value of arr[1]: 20
Value of arr[2]: 30
Value of arr[3]: 40
Value of arr[4]: 50
```

Here, the ***(p+i)** expression is used to access the **i**-th element of the array through pointer arithmetic. The ***(p+i)** is equivalent to **p[i]** and both are valid expressions to access the elements of the array.

In **C programming language**, a function is a group of statements that perform a specific task. Functions provide modularity and reusability to programs by allowing code to be organized into independent units that can be called from other parts of the program. Here's an example of how functions work:

```c
#include<stdio.h>

/* function declaration */
int max(int num1, int num2);

int main() {
int a = 100;
int b = 200;
int ret;

/* calling a function to get max value */
ret = max(a, b);

printf("Max value is : %d\n", ret );

return 0;
}

/* function returning the max between two numbers */
int max(int num1, int num2) {
int result;

if (num1 > num2) {
    result = num1;
} else {
    result = num2;
}
return result;
}
```

In this example, we declare a function named **max** that takes two integer arguments and returns the maximum value. The function is declared before the **main** function using a function prototype or declaration, which specifies the function's name, return type, and parameter types. This allows the compiler to check for errors and ensure that the function is used correctly. In the **main** function, we declare two integer variables **a** and **b** and assign them the values 100 and 200, respectively. We then call the **max** function and pass it **a** and **b** as arguments, storing the result in the **ret** variable. Finally, we print out the maximum value using **printf**. When we run this program, it will output the following:

```
Max value is : 200
```

Functions can also have multiple return statements, which allows for early termination of the function based on certain conditions. Here's an example:

```c
#include<stdio.h>

/* function declaration */
int find_factorial(int num);

int main() {
int num, factorial;

printf("Enter a positive integer: ");
scanf("%d", &num);


factorial = find_factorial(num);

if (factorial != -1) {

    printf("Factorial of %d is %d\n", num, factorial);
```

```c
} else {
    printf("Error: Factorial of negative numbers doesn't exist\n");
}
return 0;
}


/* function to calculate factorial */
int find_factorial(int num) {
int factorial = 1;

if (num < 0) {
    return -1;
    }

for (int i=1; i<=num; i++) {
    factorial *= i;
}
return factorial;
}
```

In this example, we declare a function named **find_factorial** that takes a single integer argument and returns the factorial of the number. The function checks for negative input and returns **-1** if the input is negative. Otherwise, it calculates the factorial using a **for** loop and returns the result. In the **main** function, we prompt the user to enter a positive integer and store it in the **num** variable. We then call the **find_factorial** function and store the result in the **factorial** variable. If the result is not **-1**, we print out the factorial. Otherwise, we print an error message. When we run this program, it will output the following:

```
Enter a positive integer: 5
Factorial of 5 is 120
```

Functions can also have default arguments and variable-length arguments, but these are more advanced features beyond the scope of this explanation.

In **C programming language**, an enum (short for enumeration) is a user-defined data type that consists of a set of named integer constants. Enums provide a way to define a set of related constants with meaningful names, which makes code more readable and easier to maintain. Here's an example of how enums work:

```c
#include<stdio.h>

/* define an enum for days of the week */
enum week { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };

int main() {

/* declare variables of type enum week */
enum week today, tomorrow;

/* assign values to the variables */
today = Wednesday;
tomorrow = Thursday;

/* print out the values */
printf("Today is %d\n", today);
printf("Tomorrow is %d\n", tomorrow);

return 0;
}
```

In this example, we define an enum named **week** that consists of seven named constants representing the days of the week. By default, the constants are assigned integer values starting from 0 for the first constant, 1 for the second constant, and so on. We then declare two variables of type **enum week** named **today** and **tomorrow**, and assign them the values **Wednesday** and **Thursday**, respectively. Finally, we print out the values using **printf**. When we run this program, it will output the following:

```
Today is 3
Tomorrow is 4
```

Enums can also have explicit values assigned to their constants, which allows for more control over the values. Here's an example:

```c
#include<stdio.h>

/* define an enum for colors */
enum color { Red = 1, Green = 2, Blue = 4 };

int main() {

/* declare variables of type enum color */
enum color c1, c2, c3;

/* assign values to the variables */
c1 = Red;
c2 = Green;
c3 = Blue;

/* print out the values */
printf("c1 = %d\n", c1);
```

```
printf("c2 = %d\n", c2);
printf("c3 = %d\n", c3);
return 0;
}
```

In this example, we define an enum named **color** that consists of three named constants representing colors. We assign explicit values to the constants using the assignment operator, so that **Red** is assigned the value 1, **Green** is assigned the value 2, and **Blue** is assigned the value 4. We then declare three variables of type **enum color** named **c1**, **c2**, and **c3**, and assign them the values **Red**, **Green**, and **Blue**, respectively. Finally, we print out the values using **printf**. When we run this program, it will output the following:

```
c1 = 1
c2 = 2
c3 = 4
```

Enums can also be used as switch case statements, allowing for easy handling of related constants. Here's an example:

```
#include<stdio.h>

/* define an enum for days of the week */
enum week { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };

int main() {
/* declare a variable of type enum week */
enum week day;

/* get input from user */
printf("Enter a day of the week (0-6): ");
scanf("%d", &day);
```

```c
/* use a switch statement to handle the input */
    switch (day) {
        case Sunday:
            printf("Sunday\n");
            break;
        case Monday:
            printf("Monday\n");
            break;
        case Tuesday:
            printf("Tuesday\n");
            break;
        case Wednesday:
            printf("Wednesday\n");
            break;
        case Thursday:
            printf("Thursday\n");
            break;
        case Friday:
            printf("Friday\n");
            break;
        case Saturday:
            printf("Saturday\n");
            break;
}
return 0;
}
```

In **C programming language**, a file is a collection of related data that is stored on a secondary storage device like a hard disk or a flash drive. Files can be used to store and retrieve data in a more permanent way than variables, which are lost when a program exits. Here's an example of how files work in C:

To use files in C, you need to include the **stdio.h** header file. This file contains functions for input and output, including functions for reading and writing files. Here's an example of how to open a file for writing:

```c
#include<stdio.h>

int main() {

/* declare a file pointer */
FILE *fptr;

/* open the file for writing */
fptr = fopen("example.txt", "w");

/* write some text to the file */
fprintf(fptr, "This is some example text.\n");

/* close the file */
fclose(fptr);

return 0;
}
```

In this example, we declare a file pointer named **fptr**, which is a variable that holds a reference to a file. We then open a file named **example.txt** for writing using the **fopen** function, which takes two arguments: the name of the file, and the mode in which to open the file. In this case, we're using the mode **"w"**, which means "write mode". If the file doesn't exist, it will be created. Once the file is open, we use the **fprintf** function to write some text to the file. This function works like **printf**, but instead of printing to the console, it writes to the file. Finally, we close the file using the **fclose** function. Here's an example of how to read from a file:

```c
#include<stdio.h>

int main() {

/* declare a file pointer */
FILE *fptr;

/* open the file for reading */
fptr = fopen("example.txt", "r");

/* read some text from the file */
char buffer[100];
fgets(buffer, 100, fptr);
printf("The text in the file is: %s", buffer);

/* close the file */
fclose(fptr);

return 0;
}
```

In this example, we open the same file we created earlier, but this time in read mode **("r")**. We declare a character array named **buffer** to hold the text we read from the file. We then use the **fgets** function to read up to 100 characters from the file into the buffer. Finally, we print out the text using **printf**.

It's important to note that when you open a file in write mode, any existing data in the file will be overwritten. If you want to append data to the end of the file instead, you can use the mode **"a"** (append mode) instead of **"w"**. Here's an example:

```c
#include <stdio.h>

int main() {
    /* declare a file pointer */
    FILE *fptr;

    /* open the file for appending */
    fptr = fopen("example.txt", "a");

    /* write some more text to the file */
    fprintf(fptr, "This is some additional text.\n");

    /* close the file */
    fclose(fptr);

    return 0;
}
```

In this example, we open the same file we created earlier, but this time in append mode **("a")**. This means that any data we write to the file will be added to the end of the file, rather than overwriting existing data. We use the **fprintf** function again to write some additional text to the file, and then close the file using the **fclose** function.

C programming is a popular and widely used programming language with many advantages. Here are some of the key advantages of C programming:

- **Speed and efficiency:** C is a high-performance language that is known for its speed and efficiency. It is commonly used to develop software that requires fast execution and low-level memory manipulation, such as operating systems, device drivers, and embedded systems.

- **Portability:** C programs can be compiled and run on different platforms, including Windows, macOS, Linux, and embedded systems, making it a portable language. This means that code written in C can be easily transferred to other systems without significant modifications.

- **Low-level programming:** C allows programmers to write code that interacts directly with hardware and memory, making it ideal for developing system software and drivers.

- **Memory management:** C provides a level of control over memory management that is not available in many other programming languages. This allows programmers to allocate and deallocate memory manually, resulting in more efficient and optimized code.

- **Extensive standard library:** C comes with an extensive standard library that provides many useful functions, such as input/output, string handling, and mathematical operations.

- **Flexibility:** C is a flexible language that can be used to develop a wide range of applications, from low-level system software to high-level applications.

- **Reusability:** C supports modular programming, which allows code to be divided into smaller, more manageable functions and modules. As a result, code is easier to maintain and more reusable.

- **Widely used:** C is a popular language that is widely used in industry, making it a valuable skill for programmers to have. Many operating systems, embedded systems, and applications are written in C.

Overall, **C programming** is a powerful and versatile language that is well-suited for developing a wide range of applications. Its speed, efficiency, portability, and low-level programming capabilities make it a popular choice for developers around the world. Using C has certain

drawbacks in addition to its many benefits. The following are some of the primary drawbacks of C:

- **Low-level language:** C is a low-level language, which means that it requires a lot of code to perform simple tasks. For example, in C, you need to write a lot of code to read and write files, whereas in higher-level languages like Python, it can be done in just a few lines.
- **No automatic garbage collection:** C does not have automatic garbage collection, which means that you have to manage memory allocation and deallocation yourself. This can be very tedious and error-prone, especially when dealing with complex data structures.
- **No built-in support for object-oriented programming:** Unlike some other programming languages, C does not have built-in support for object-oriented programming. This means that if you want to use OOP concepts in your C code, you have to implement them yourself, which can be a time-consuming process.
- **Vulnerability to buffer overflow attacks:** C is vulnerable to buffer overflow attacks, which occur when a program tries to store too much data in a buffer, causing the excess data to overwrite adjacent memory locations. This can lead to the **program crashing** or, in some cases, being exploited by attackers.
- **Lack of dynamic memory management:** C does not have dynamic memory management built into the language, which means that you have to manually allocate and deallocate memory. This can lead to memory leaks if you forget to **deallocate memory**, which can cause your program to consume more and more memory until it crashes.

  Example:

  Here's an example of a program written in C that illustrates some of these disadvantages:

```c
#include<stdio.h>
int main() {
char buffer[100];
printf("Enter a string: ");
scanf("%s", buffer);
printf("You entered: %s\n", buffer);
return 0;
}
```

In this program, the user is prompted to enter a string, which is then stored in a buffer. However, the program does not perform any input validation, which means that if the user enters a string that is longer than **100 characters**, it will overflow the buffer and potentially overwrite adjacent memory locations, leading to undefined behavior or a crash. Additionally, the program does not perform any memory management, which means that the buffer will remain in memory even after it is no longer needed, potentially causing a memory leak. Finally, the program does not use any object-oriented programming concepts, making it harder to maintain and extend as it grows in complexity.

- **No built-in support for exception handling:** C does not have built-in support for exception handling, which means that error handling must be done manually using conditional statements and error codes. This could make code more challenging to read and maintain.
- **Lack of standardization:** The C language has no official standard library, which means that different implementations of C may have different libraries and functions. This can lead to portability issues when trying to run code on different platforms or compilers.
- **Lack of type safety:** C is a weakly-typed language, which means that it does not enforce strict type checking. This can lead to errors and bugs when working with complex data structures or when trying to perform operations on incompatible data types.
- **Difficulty in debugging:** Debugging C code can be difficult due to its low-level nature and lack of built-in debugging tools. This can make it hard to find and fix bugs in complex code.
- **Potential for security vulnerabilities:** Because of its low-level nature and lack of built-in security features, C code can be vulnerable to security attacks such as buffer overflows, which can lead to serious security issues.

Example:

Here's an example of a program written in **C** that illustrates some of these disadvantages:

```c
#include<stdio.h>

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

if (num % 2 == 0) {

printf("%d is even\n", num);

}

else {

printf("%d is odd\n", num);

}

return 0;

}
```

In this program, the user is prompted to enter a number, which is then checked to see if it's even or odd. However, the program does not perform any input validation, which means that if the user enters a non-numeric value, it will cause an error. Additionally, the program does not handle any exceptions, which means that if an error occurs, it will simply crash. Finally, the program uses a conditional statement to check if the number is even or odd, which can be harder to read and maintain as the code grows in complexity.

C language is a widely used programming language that has **numerous practical applications**. Here are some examples of its practical applications:

- **Operating Systems:** C is used to write low-level system software such as operating systems, device drivers, and firmware. Some of the well-known operating systems written in C include UNIX, Linux, and Windows.

- **Embedded Systems:** C is popularly used to write code for embedded systems, such as microcontrollers and other electronic devices, where the code must be fast, efficient, and lightweight.

- **Database Systems:** C is often used in the development of database management systems (DBMS) like MySQL and PostgreSQL, which are widely used in web and mobile applications.

- **Compilers and Interpreters:** C is used to create compilers and interpreters for other programming languages, which are essential tools for software development.

- **Graphics Applications:** C is often used in the development of graphics applications such as video games, 3D modeling, and animation software.

- **Web Applications:** C is often used in the backend of web applications, where it's used to handle the heavy processing of data and requests, such as parsing and manipulating XML and JSON data.

- **Scientific and Engineering Applications:** C is popularly used in scientific and engineering applications, where high-performance computing is required to handle large amounts of data.

- **Business Applications:** C is also used in business applications, such as accounting software, inventory management software, and customer relationship management (CRM) systems.

- **Artificial Intelligence and Machine Learning:** C is used in the development of artificial intelligence and machine learning algorithms, where its speed and low-level control make it ideal for high-performance computing tasks.

Sure, here are some funny facts about C programming:

- C is often referred to as the **"Dennis Ritchie Language"** after its creator. In the early days of computing, programmers had to write in machine language, which was incredibly difficult and time-consuming. C was created to make programming easier, but ironically, it ended up becoming one of the most complex programming languages of all time!

- C programming can be quite cryptic and hard to read, especially for beginners. Some programmers have joked that C stands for **"Confusing"** or **"Cryptic"!**

- One of the most famous quotes about C programming is from Brian Kernighan, one of the co-authors of **"The C Programming Language": "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."**

- C programming can be a bit of a headache because of its use of pointers. Pointers are variables that store the memory address of another variable, and they can be quite tricky to use correctly.

- Despite its reputation for being difficult, C programming can also be quite fun and rewarding. There's a certain satisfaction that comes with writing code that works perfectly and efficiently, and C programming can provide that feeling in spades.

Overall, **C programming** can be both frustrating and entertaining at the same time. Despite its challenges, it remains one of the most widely used programming languages in the world, and mastering it can be a rewarding experience for anyone who is willing to put in the time and effort.

C++ – OVERVIEW

C++ is a general-purpose, high-level programming language developed by **Bjarne Stroustrup** in 1985 as an **extension of the C programming language**. It is an object-oriented language that supports various programming paradigms, such as procedural, functional, and generic programming. C++ is widely used for developing applications, system software, drivers, embedded systems, games, and other software. Here are some key features and concepts of C++ in detail:

- **Object-oriented programming:** C++ is an object-oriented language, which means it provides features like classes, objects, inheritance, and polymorphism that allow developers to write code in an organized and modular way.
- **Data types:** C++ supports various data types such as integers, floating-point numbers, characters, and Boolean values. It also provides user-defined data types like structures, unions, and enums.
- **Control structures:** C++ provides control structures like if-else statements, switch-case statements, while loops, do-while loops, and for loops to control the flow of execution in a program.
- **Functions:** C++ functions are self-contained units of code that can be reused multiple times in a program. They can be used to perform a specific task, accept parameters, and return values.
- **Pointers:** C++ allows the use of pointers, which are variables that store the memory address of another variable. They are used to manipulate memory, create dynamic data structures, and optimize performance.
- **Memory management:** C++ provides memory management features like dynamic memory allocation and deallocation using operators like new and delete.
- **Exception handling:** C++ supports exception handling to handle runtime errors and ensure graceful termination of a program.
- **Standard Template Library (STL):** C++ provides a rich library of reusable code called the Standard Template Library (STL). It includes containers like vectors, lists, maps, and algorithms like sorting and searching.

Here's an example of a "Hello, World!" program in C++:

```cpp
#include<iostream>
int main() {
std::cout << "Hello, World!" << std::endl;
return 0;
}
```

This program simply prints the string "Hello, World!" to the console.

Let's break down the program line by line:

- **#include<iostream>:** This line includes the **iostream** header file, which provides input and output streams like **cin** and **cout**.
- **int main() {**: This line declares the **main** function, which is the entry point of the program. **int** is the return type of the function, which indicates that it will return an integer value. The empty parentheses indicate that the function takes no arguments.

- **std::cout << "Hello, World!" << std::endl;:** This line prints the string "Hello, World!" to the console. **std::cout** is a standard output stream object, and the **<<** operator is used to insert the string into the stream. **std::endl** is a special character that adds a new line to the output.

- **return 0;:** This line indicates that the program has finished executing and returns the value **0** to the operating system. A return value of **0** typically indicates success.

The **std::** prefix before **cout** and **endl** is used to specify that these names are part of the **std** namespace, which is the standard namespace for C++ library functions and objects. Overall, this program demonstrates the basic syntax and structure of a C++ program. By using the **iostream** library, it shows how to perform basic input or output operations, and by defining the **main** function, it specifies the entry point of the program.

In **C++**, comments are used to add explanatory text to a program that is ignored by the compiler. There are two types of comments in C++: **single-line comments** and **multi-line comments**.

- **Single-line comments** start with **//** and continue until the end of the line. Here's an example:

```
// This is a single-line comment

int main() {
// This is also a single-line comment
return 0;
}
```

In this example, the text after **//** is ignored by the compiler. Single-line comments are often used to explain individual lines of code or to temporarily disable code during development.

- **Multi-line comments** start with **/*** and end with ***/**. Here's an example:

917

```cpp
/*
This is a multi-line comment
that can span multiple lines
*/

int main() {
    /*
    This is also a multi-line comment
    that can span multiple lines
    */
    return 0;
}
```

In this example, the text between **/*** and ***/** is ignored by the compiler. Multi-line comments are often used to explain larger blocks of code or to temporarily disable large sections of code. It's important to note that comments should be used sparingly and only when necessary. Overuse of comments can make code harder to read and maintain, and can also become outdated or misleading if the code changes. However, when used appropriately, comments can be a useful tool for improving the clarity and understandability of a program.

Data that can be used and modified across a program is stored in variables in C++. A **variable** is defined by stating its data type, name, and, if applicable, initial value. **The following are some C++ examples of variable declarations:**

```cpp
int age = 30; // integer variable named "age" with initial value of 30

float pi = 3.14159; // floating-point variable named "pi" with initial value of 3.14159

double salary = 50000.0; // double-precision floating-point variable named "salary" with initial value of 50000.0

char grade = 'A'; // character variable named "grade" with initial value of 'A'

bool isStudent = true; // boolean variable named "isStudent" with initial value of true
```

918

In these examples, we declare variables with different data types including **int**, **float**, **double**, **char**, and **bool**. We also assign an initial value to each variable using the assignment operator **=**. Variables can be modified by assigning a new value to them, as shown in this example:

```cpp
int count = 0; // declare an integer variable named "count" with initial value of 0
count = count + 1; // increment the value of "count" by 1
```

In this example, we increment the value of the **count** variable by 1 using the **+** operator and assignment operator **=**. It's important to note that variables have a scope, which determines where they can be accessed in a program. A variable declared inside a function, for example, can only be accessed within that function. A variable declared outside of any function, on the other hand, can be accessed throughout the entire program.

```cpp
#include<iostream>

int global_var = 10; // Global variable

int main() {
int local_var = 20; // Local variable

std::cout << "Global variable value: " << global_var << std::endl;
std::cout << "Local variable value: " << local_var << std::endl;

return 0;
}
```

In this example, we declare a global variable named **global_var** outside of any function, and a local variable named **local_var** inside the **main** function. We can access both variables within the **main** function, but only the global variable can be accessed outside of the function. Overall,

variables are an essential part of C++ programming and are used to store and manipulate data throughout a program.

In **C++**, data types specify the type of data that can be stored in a variable. C++ supports a wide range of data types, including:

1. **Integers:** used to store whole numbers. There are several integer data types in C++, including **int**, **short**, **long**, and **long long**. Example:

```cpp
int age = 30;
short weight = 150;
long distance = 1000000;
long long total = 9000000000;
```

2. **Floating-point numbers:** used to store numbers with decimal points. There are two floating-point data types in C++, **float** and **double**. Example:

```cpp
float pi = 3.14159;
double salary = 50000.0;
```

3. **Characters:** used to store single characters. Character data types are denoted by the **char** keyword. Example:

```cpp
char grade = 'A';
```

4. **Boolean:** used to store **true** or **false** values. Boolean data types are denoted by the **bool** keyword. Example:

```cpp
bool isStudent = true;
```

5. **Arrays:** used to store a collection of data of the same data type. Example:

```cpp
int numbers[5] = {1, 2, 3, 4, 5};
```

6. **Pointers:** used to store the memory address of another variable. Example:

```cpp
int age = 30;
int* agePtr = &age; // agePtr stores the memory address of age
```

It's important to note that data types have different ranges and precision, and choosing the appropriate data type for a variable is important for performance and accuracy. For example, using a **short** data type instead of an **int** data type for large numbers can cause overflow or truncation of the value. C++ also provides type modifiers, such as **signed** and **unsigned**, to further specify the range of a data type. For example, **unsigned int** can only store non-negative integers. Overall, understanding data types in C++ is important for writing efficient and accurate code.

C++ operators are special symbols and keywords used to perform various operations on variables and values. Here are some common C++ operators with examples:

1. **Arithmetic Operators:**
    - **Addition (+):** `int sum = 5 + 3;` (sum equals 8)
    - **Subtraction (-):** `int difference = 5 - 3;` (difference equals 2)
    - **Multiplication (*):** `int product = 5 * 3;` (product equals 15)
    - **Division (/):** `float quotient = 5.0 / 3.0;` (quotient equals 1.66667)
    - **Modulo (%):** `int remainder = 5 % 3;` (remainder equals 2)
2. **Assignment Operators:**
    - **Simple Assignment (=):** `int a = 5;` (a equals 5)
    - **Compound Assignment (+=, -=, *=, /=, %=):** `a += 3;` (a equals 8)
3. **Comparison Operators:**
    - **Equal to (==):** `bool isEqual = 5 == 3;` (isEqual equals false)

- **Not equal to (!=):** `bool isNotEqual = 5 != 3;` (isNotEqual equals true)
- **Greater than (>):** `bool isGreater = 5 > 3;` (isGreater equals true)
- **Less than (<):** `bool isLess = 5 < 3;` (isLess equals false)
- **Greater than or equal to (>=):** `bool isGreaterOrEqual = 5 >= 3;` (isGreaterOrEqual equals true)
- **Less than or equal to (<=):** `bool isLessOrEqual = 5 <= 3;` (isLessOrEqual equals false)

4. **Logical Operators:**
   - **Logical AND (&&):** `bool isTrue = (5 > 3) && (4 < 6);` (isTrue equals true)
   - **Logical OR (||):** `bool isFalse = (5 < 3) || (4 > 6);` (isFalse equals false)
   - **Logical NOT (!):** `bool isNotTrue = !(5 > 3);` (isNotTrue equals false)

5. **Bitwise Operators:**
   - **Bitwise AND (&):** `int result = 5 & 3;` (result equals 1)
   - **Bitwise OR (|):** `int result = 5 | 3;` (result equals 7)
   - **Bitwise XOR (^):** `int result = 5 ^ 3;` (result equals 6)
   - **Bitwise NOT (~):** `int result = ~5;` (result equals -6)
   - **Left shift (<<):** `int result = 5 << 1;` (result equals 10)
   - **Right shift (>>):** `int result = 5 >> 1;` (result equals 2)

These are just some of the many operators available in C++.

In **C++**, a **string** is a sequence of characters stored in a contiguous block of memory. The string data type is defined in the **<string>** header file. Here are some examples of working with C++ strings:

1. **Creating a string:**

```cpp
#include<string>
using namespace std;

string str1 = "Hello, world!"; // initializing string with a literal
string str2("I am a string."); // initializing string with constructor
string str3; // declaring an empty string
```

2. **Accessing and modifying characters in a string:**

```cpp
char first_char = str1[0]; // accessing the first character of the string
str1[7] = 'W'; // changing the 8th character to 'W'
str1.at(4) = 'o'; // changing the 5th character to 'o' using the at() method
```

3. **Concatenating strings:**

```cpp
string greeting = "Hello";
string name = "Alice";
string message = greeting + ", " + name + "!"; // concatenating strings with the + operator
```

4. **Finding substrings:**

```cpp
string sentence = "The quick brown fox jumps over the lazy dog.";
int position = sentence.find("fox"); // finding the first occurrence of "fox"
string word = sentence.substr(16, 5); // extracting the word "brown" starting from position 16
```

5. **Converting strings to numeric values:**

```cpp
string num_string = "12345";
int num_int = stoi(num_string); // converting string to integer
double num_double = stod("3.14"); // converting string to double
```

6. **Getting the length of a string:**

```cpp
int length = str1.length(); // getting the length of the string
```

7. **Comparing strings:**

```cpp
string s1 = "hello";
string s2 = "world";
if (s1 == s2) {
    // strings are equal
} else {
    // strings are not equal
}
```

In **C++**, the **math library** provides a set of functions for performing mathematical operations. Here are some examples of working with C++ math functions:

1. **Absolute value:**

```cpp
#include<cmath>
using namespace std;


double num = -3.14;
double abs_num = abs(num); // taking the absolute value of a number
```

2. **Square root:**

```cpp
double x = 16.0;
double sqrt_x = sqrt(x); // taking the square root of a number
```

3. **Trigonometric functions:**

```cpp
double angle = 45.0;
double radians = angle * M_PI / 180.0; // converting angle to radians
double sin_val = sin(radians); // taking the sine of an angle in radians
double cos_val = cos(radians); // taking the cosine of an angle in radians
double tan_val = tan(radians); // taking the tangent of an angle in radians
```

4. **Exponential function:**

```cpp
double x = 2.0;
double exp_x = exp(x); // taking the exponential function of a number
```

5. **Logarithmic functions:**

```cpp
double x = 10.0;
double log_x = log(x); // taking the natural logarithm of a number
double log10_x = log10(x); // taking the base 10 logarithm of a number
```

6. **Power function:**

```cpp
double base = 2.0;
double exponent = 3.0;
double result = pow(base, exponent); // taking the power of a base to an exponent
```

7. **Rounding functions:**

```cpp
double x = 3.7;
double rounded_down = floor(x); // rounding down to the nearest integer
double rounded_up = ceil(x); // rounding up to the nearest integer
double rounded_nearest = round(x); // rounding to the nearest integer
```

Note that the **math library** also provides many more functions than what is covered here, such as hyperbolic functions, inverse trigonometric functions, and more. You can include the `<cmath>` header file to use these functions in your C++ programs.

In **C++,** a **boolean** is a data type that represents either true or false. Here are some examples of working with C++ booleans:

1. **Creating a boolean:**

```cpp
#include<iostream>
using namespace std;

bool is_sunny = true; // initializing a boolean with a literal
bool is_raining(false); // initializing a boolean with a constructor
bool is_cloudy; // declaring an uninitialized boolean
```

2. **Comparing values:**

```cpp
bool result = (3 == 3); // comparing two values for equality
bool is_greater = (5 > 3); // comparing two values for greater than
bool is_less_equal = (10 <= 15); // comparing two values for less than or equal to
```

3. **Logical operators:**

```cpp
bool a = true;
bool b = false;
bool and_result = a && b; // logical AND
bool or_result = a || b; // logical OR
bool not_result = !a; // logical NOT
```

4. **Conditional statements:**

```cpp
bool is_sunny = true;
if (is_sunny) {
    cout << "It's a sunny day!" << endl;
} else {
    cout << "It's not a sunny day." << endl;
}
```

5. Boolean expressions in loops:

```cpp
bool is_running = true;
int count = 0;
while (is_running) {
    count++;
    if (count == 10) {
        is_running = false;
    }
}
```

Note that **boolean** values are commonly used in conditional statements, loops, and other control structures in C++. In addition, C++ provides the keywords true and false to represent **boolean** values, and the `<stdbool.h>` header file can be included for compatibility with C.

In **C++**, the `if...else` statement allows you to execute different code based on whether a condition is true or false. Here are some examples of working with `if...else` statements in C++:

1. Basic if statement:

```cpp
int x = 10;
if (x > 5) {
    cout << "x is greater than 5" << endl;
}
```

2. if...else statement:

```cpp
int x = 10;
if (x > 5) {
    cout << "x is greater than 5" << endl;
} else {
    cout << "x is less than or equal to 5" << endl;
}
```

3. **Nested if...else statements:**

```cpp
int x = 10;
if (x > 5) {
    if (x < 15) {
        cout << "x is between 5 and 15" << endl;
    } else {
        cout << "x is greater than or equal to 15" << endl;
    }
} else {
    cout << "x is less than or equal to 5" << endl;
}
```

4. **if...else if...else statement:**

```cpp
int x = 10;
if (x < 5) {
    cout << "x is less than 5" << endl;
} else if (x < 10) {
    cout << "x is between 5 and 10" << endl;
} else {
    cout << "x is greater than or equal to 10" << endl;
}
```

5. **Ternary operator:**

```cpp
int x = 10;
int result = (x > 5) ? 1 : 0; // if x > 5, set result to 1, otherwise set it to 0
```

Note that the **if...else** statement is a fundamental control structure in C++, and can be used to control the flow of your program based on specific conditions. By combining **if...else**

statements with logical operators and comparison operators, you can create complex control structures that can handle a wide variety of conditions.

In **C++**, the `switch` statement allows you to execute different code based on the value of a variable. Here is the basic syntax of a `switch` statement:

```cpp
switch(variable){
  case value1:
    // code to execute when variable == value1
    break;
  case value2:
    // code to execute when variable == value2
    break;
  // additional cases can be added
  default:
    // code to execute when variable does not match any of the cases
    break;
}
```

Here are some examples of using the switch statement in C++:

Example 1: Printing the name of a day based on its number

```cpp
#include<iostream>
using namespace std;

int main() {
  int day = 3;
  switch(day) {
    case 1:
      cout << "Monday";
      break;
```

```cpp
      case 2:
        cout << "Tuesday";
        break;
      case 3:
        cout << "Wednesday";
        break;
      case 4:
        cout << "Thursday";
        break;
      case 5:
        cout << "Friday";
        break;
      case 6:
        cout << "Saturday";
        break;
      case 7:
        cout << "Sunday";
        break;
      default:
        cout << "Invalid day";
        break;
  }
  return 0;
}
```

Output:

```
Wednesday
```

Example 2: Calculating the result of an arithmetic operation

```cpp
#include<iostream>
using namespace std;

int main() {
  int a = 10, b = 5;
  char op = '-';
  int result;
  switch(op) {
    case '+':
      result = a + b;
      break;
    case '-':
      result = a - b;
      break;
    case '*':
      result = a * b;
      break;
    case '/':
      result = a / b;
      break;
    default:
      cout << "Invalid operator";
      return 1;
  }
  cout << "Result: " << result;
  return 0;
}
```

Output:

```
Result: 5
```

Example 3: Checking the type of a variable

```cpp
#include<iostream>
#include<typeinfo>
using namespace std;

int main() {
  auto x = 1.5;
  switch(typeid(x).hash_code()) {
    case typeid(int).hash_code():
      cout << "x is an integer";
      break;
    case typeid(float).hash_code():
      cout << "x is a float";
      break;
    case typeid(double).hash_code():
      cout << "x is a double";
      break;
    default:
      cout << "x has an unknown type";
      break;
  }
  return 0;
}
```

Output:

```
x is a double
```

Loops in C++ allow you to execute a block of code repeatedly based on a certain condition. There are three types of loops in C++:

1. The for loop
2. The while loop
3. The do-while loop

Here is an example and explanation of each loop:

1. **The for loop:**

   The **for loop** is used when you know exactly how many times you want to execute the loop. It has three parts: initialization, condition, and increment or decrement.

   Syntax:

   ```
   for (initialization; condition; increment/decrement) {
     // code to be executed repeatedly
   }
   ```

   Example:

   ```cpp
   #include<iostream>
   using namespace std;

   int main() {
     for (int i = 0; i < 5; i++) {
       cout << i << endl;
     }
     return 0;
   }
   ```

   Output:

```
0

1

2

3

4
```

Explanation:

In this example, the loop will execute five times because the condition is set to run while **i** is less than 5. The **i** variable is initialized to 0 and incremented by 1 at the end of each loop iteration.

2. **The while loop:**

The while loop is used when you don't know exactly how many times you want to execute the loop. It will continue to run until the condition becomes false.

Syntax:

```
while (condition) {
   // code to be executed repeatedly
}
```

Example:

```cpp
#include<iostream>
using namespace std;

int main() {
   int i = 0;
   while (i < 5) {
      cout << i << endl;
      i++;
   }
   return 0;
}
```

Output:

```
0
1
2
3
4
```

Explanation:

This example achieves the same result as the for loop example above. The loop will continue to execute as long as the **i** variable is less than 5.

3. **The do-while loop:**

   The do-while loop is similar to the while loop, but the condition is checked at the end of the loop iteration instead of the beginning. This means that the loop will always execute at least once.

   Syntax:

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

   Example:

   ```cpp
   #include<iostream>
   using namespace std;

   int main() {
     int i = 0;
     do {
       cout << i << endl;
       i++;
     } while (i < 5);
     return 0;
   }
   ```

Output:

```
0

1

2

3

4
```

Explanation:

This example also achieves the same result as the previous two examples. The loop will execute at least once because the condition is checked at the end of the first iteration.

In **C++**, an **array** is a collection of elements of the same data type that are stored in contiguous memory locations. Each element in the array is accessed using an index, which represents the position of the element in the array. **Arrays** can be useful for storing and manipulating large amounts of data.

Syntax to declare an array:

```
dataType arrayName[arraySize];
```

Here, **dataType** is the data type of the elements you want to store, **arrayName** is the name of the array, and **arraySize** is the number of elements in the array.

Example:

```
int arr[5];
```

This creates an array of integers with 5 elements.

Initialization of an array can be done using braces **{}**. The values are separated by commas and the number of values must match the number of elements in the array.

Example:

```cpp
int arr[5] = {1, 2, 3, 4, 5};
```

This initializes an array of integers with 5 elements, where the first element is 1, the second element is 2, and so on.

Accessing elements of an array can be done using indices, starting from 0 for the first element.

Example:

```cpp
cout << arr[0]; // Accesses the first element of the array
cout << arr[3]; // Accesses the fourth element of the array
```

In **C++**, arrays are fixed in size, which means that the number of elements in the array cannot be changed after the array is declared. To change the value of an element in the array, simply assign a new value to the element using the index.

Example:

```cpp
arr[2] = 10; // Changes the third element of the array to 10
```

In addition to one-dimensional arrays, **C++** also supports multidimensional arrays, such as two-dimensional arrays, three-dimensional arrays, and so on. A two-dimensional array can be thought of as a table, with rows and columns of elements.

Example of a two-dimensional array:

```cpp
int arr[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
```

This initializes a two-dimensional array with 3 rows and 4 columns.

To sum up, **arrays** are a fundamental data structure in **C++**, and can be useful for storing and manipulating large amounts of data in a structured manner.

In **C++**, a reference is an alias for a variable, which allows you to refer to the original variable using a different name. **References** can be useful for passing arguments to functions, as they avoid the overhead of making a copy of the argument.

Syntax to declare a reference:

```cpp
dataType& refName = variableName;
```

Here, **dataType** is the data type of the variable being referred to, **refName** is the name of the reference, and **variableName** is the name of the variable being referred to.

Example:

```cpp
int num = 10;
int& numRef = num;
```

This creates a reference to the integer variable **num**, named **numRef**.

Assigning a value to a reference changes the value of the original variable being referred to.

Example:

```
numRef = 20;
cout << num; // Outputs 20
```

This assigns the value 20 to the reference **numRef**, which changes the value of the original variable **num** to 20.

References can be useful for passing arguments to functions by reference, which allows the function to modify the original value of the argument.

Example:

```
void doubleValue(int& num) {
    num *= 2;
}

int main() {
    int num = 10;
    doubleValue(num);
    cout << num; // Outputs 20
    return 0;
}
```

This declares a function named **doubleValue**, which takes an integer reference as an argument and doubles the value of the argument. The function is then called in the **main** function with the integer variable **num** as the argument.

Overall, **references in C++** are aliases for variables, which can be useful for passing arguments to functions and modifying the original value of the argument.

In C++, a **pointer** is a variable that stores the memory address of another variable. **Pointers** can be useful for manipulating memory directly and for implementing complex data structures.

Syntax to declare a pointer:

```
dataType* pointerName;
```

Here, **dataType** is the data type of the variable being pointed to, and **pointerName** is the name of the pointer.

Example:

```
int* numPtr;
```

This declares a pointer to an integer variable.

To assign a value to a pointer, you must use the address-of operator **&** to obtain the memory address of the variable being pointed to.

Example:

```
int num = 10;
int* numPtr = &num;
```

This assigns the memory address of the integer variable **num** to the pointer **numPtr**.

To access the value of the variable being pointed to, you must use the dereference operator *****.

Example:

```
cout << *numPtr;
```

This outputs the value of the integer variable being pointed to by **numPtr**.

Pointers can be useful for passing arguments to functions, as they allow the function to access and modify the original value of the argument.

Example:

```cpp
void doubleValue(int* numPtr) {
    *numPtr *= 2;
}

int main() {
    int num = 10;
    int* numPtr = &num;
    doubleValue(numPtr);
    cout << num; // Outputs 20
    return 0;
}
```

This declares a function named **doubleValue**, which takes a pointer to an integer variable as an argument and doubles the value of the variable using the dereference operator. The function is then called in the **main** function with the pointer to the integer variable **num** as the argument.

In addition to pointers to variables, **C++** also supports pointers to arrays and pointers to functions.

Example of a pointer to an array:

```cpp
int arr[5] = {1, 2, 3, 4, 5};
int* arrPtr = arr;
```

This declares an array of integers with 5 elements and a pointer to the first element of the array.

Example of a pointer to a function:

```cpp
int add(int x, int y) {
    return x + y;
}


int (*addPtr)(int, int) = add;
```

This declares a function named **add**, which takes two integer arguments and returns their sum, and a pointer to the **add** function.

Overall, **pointers in C++** are variables that store the memory address of another variable, which can be useful for manipulating memory directly and for implementing complex data structures.

In C++, files are used for input and output operations. Input operations involve reading data from a file into a program, while output operations involve writing data from a program to a file. C++ provides several file stream classes for handling file input and output operations: **ifstream** for input from a file, **ofstream** for output to a file, and **fstream** for both input and output.

Syntax to open a file:

```cpp
fstream fileStream;
fileStream.open("filename", mode);
```

Here, **filename** is the name of the file to be opened, and **mode** is the mode in which to open the file, which can be either **ios::in** for input, **ios::out** for output, or **ios::in | ios::out** for both input and output.

Example of opening a file for output:

```
ofstream outputFile;
outputFile.open("output.txt", ios::out);
```

This opens a file named **output.txt** in output mode.

To write data to a file, you can use the insertion operator **<<**.

Example of writing data to a file:

```
outputFile << "Hello, world!";
```

This writes the string **"Hello, world!"** to the file **output.txt**.

To read data from a file, you can use the extraction operator **>>**.

Example of reading data from a file:

```
ifstream inputFile;
inputFile.open("input.txt", ios::in);
string data;
inputFile >> data;
```

This opens a file named **input.txt** in input mode and reads a string of data from the file into the variable **data**.

After finishing file operations, it is important to close the file using the **close** function.

Example of closing a file:

```
outputFile.close();
```

This closes the output file **output.txt**.

In addition to the basic file operations, **C++** also provides functions for checking the status of a file, moving the file pointer, and working with binary files. Overall, **files in C++** are used for input and output operations, and **C++** provides several file stream classes for handling file input and output. To perform file operations, a file must be opened, data must be read from or written to the file, and the file must be closed after finishing the operations.

In C++, functions are blocks of code that perform specific tasks. They are used to break down a program into smaller, more manageable pieces of code, and to avoid writing the same code repeatedly. **Here are some examples of C++ functions:**

1. **Function with no parameters and no return value**

```cpp
#include<iostream>

void printHello() {
    std::cout << "Hello, world!" << std::endl;
}

int main() {
    printHello(); // prints "Hello, world!"
    return 0;
}
```

This function **printHello** takes no parameters and returns no value. It simply prints "Hello, world!" to the console.

2. **Function with parameters and no return value**

```cpp
#include <iostream>

void printName(std::string name) {
    std::cout << "Hello, " << name << "!" << std::endl;
}

int main() {
    printName("Alice"); // prints "Hello, Alice!"
    printName("Bob"); // prints "Hello, Bob!"
    return 0;
}
```

This function **printName** takes a **std::string** parameter **name** and returns no value. It prints "Hello, " followed by the value of **name** and an exclamation mark.

3. **Function with parameters and return value**

```cpp
#include<iostream>

int square(int num) {
    return num * num;
}

int main() {
    std::cout << square(5) << std::endl; // prints 25
    std::cout << square(-3) << std::endl; // prints 9
    return 0;
}
```

This function **square** takes an **int** parameter **num** and returns the square of **num**. It is used in **main** to print the square of **5** and **-3**.

4. **Recursive function**

```cpp
#include<iostream>

int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}

int main() {
    std::cout << factorial(5) << std::endl; // prints 120
    std::cout << factorial(0) << std::endl; // prints 1
    return 0;
}
```

This function **factorial** is a recursive function that calculates the factorial of a number **n**. It uses the formula **n! = n * (n-1)!**. The base case is when **n** is **0**, in which case the function returns **1**.

5. **Function with default arguments**

```cpp
#include<iostream>

void printAge(std::string name, int age = 18) {
    std::cout << name << " is " << age << " years old." << std::endl;
}
```

```cpp
int main() {
    printAge("Alice"); // prints "Alice is 18 years old."
    printAge("Bob", 25); // prints "Bob is 25 years old."
    return 0;
}
```

This function **printAge** takes a **std::string** parameter **name** and an optional **int** parameter **age** with a default value of **18.** It prints **name** and **age** to the console.

6. Function overloading

Function overloading in C++ allows multiple functions to have the same name but different parameter lists. This makes it possible to use the same function name for operations that are conceptually similar but differ in the types or number of arguments used**. Here's an example of function overloading:**

```cpp
#include<iostream>

int add(int a, int b) {
    std::cout << "Calling integer version of add()" << std::endl;
    return a + b;
}

double add(double a, double b) {
    std::cout << "Calling double version of add()" << std::endl;
    return a + b;
}

int main() {
    std::cout << add(1, 2) << std::endl;     // calls integer version of add()
    std::cout << add(3.14, 2.71) << std::endl;  // calls double version of add()
    return 0;
}
```

In this example, we have two functions with the same name **add**, but one takes two **int** parameters and the other takes two **double** parameters. When the function is called with **int** arguments, the integer version of the function is called, and when the function is called with **double** arguments, the double version of the function is called. This is possible because the function signature (name and parameter list) determines which function is called. **Function overloading** can be useful in many situations, such as when you need to perform a similar operation on different types of data, or when you want to provide multiple ways to call a function with different numbers or types of arguments. However, care should be taken to ensure that the overloaded functions have different parameter lists to avoid ambiguity.

7. Lambda function

```cpp
#include<iostream>

int main() {
    int x = 5;
    int y = 7;
    auto sum = [](int a, int b) -> int {
        return a + b;
    };
    std::cout << sum(x, y) << std::endl; // prints 12
    return 0;
}
```

This is an example of a **lambda function**. It is an anonymous function that can be assigned to a variable, like **sum**. In this example, it takes two **int** parameters and returns their sum. The **->** **int** syntax is used to specify the return type. **Lambda functions** can be useful for writing concise, functional-style code that is easy to read and understand. They are commonly used in C++ for algorithms such as **std::sort**, **std::transform**, and **std::for_each**, as well as in asynchronous programming with **std::async** and **std::future**.

In C++, a **class** is a user-defined data type that encapsulates data and functions into a single entity. It provides a way to organize and structure code, and to create objects that can be used to represent real-world entities or concepts. In this section, we'll look at how to define and use a class in C++. **Here's an example of a simple class:**

```cpp
#include<iostream>

class Rectangle {
public:
    int width, height;

    int area() {
        return width * height;
    }

    void print() {
        std::cout << "Width: " << width << ", Height: " << height << std::endl;
    }
};

int main() {
    Rectangle r;
    r.width = 10;
    r.height = 20;
    std::cout << "Area: " << r.area() << std::endl;
    r.print();
    return 0;
}
```

In this example, we define a class **Rectangle** that has two data members **width** and **height**, and two member functions **area()** and **print()**. The **area()** function calculates the area of

the rectangle by multiplying the width and height, and the **print()** function prints the width and height to the console. In the **main()** function, we create an object **r** of the **Rectangle** class and set its width and height to 10 and 20, respectively. We then call the **area()** function to calculate the area of the rectangle and print it to the console, followed by a call to the **print()** function to print the width and height of the rectangle.

Classes can also have constructors and destructors, which are special member functions that are called when an object is created and destroyed, respectively. Here's an example that shows how to define a constructor and destructor for the **Rectangle** class:

```cpp
#include<iostream>

class Rectangle {

public:
    int width, height;

    Rectangle() {
        width = 0;
        height = 0;
        std::cout << "Rectangle constructor called" << std::endl;
    }

    Rectangle(int w, int h) {
        width = w;
        height = h;
        std::cout << "Rectangle constructor called" << std::endl;
    }

    ~Rectangle() {
        std::cout << "Rectangle destructor called" << std::endl;
    }
```

```cpp
    int area() {
        return width * height;
    }

    void print() {
        std::cout << "Width: " << width << ", Height: " << height << std::endl;
    }
};

int main() {
    Rectangle r1;
    r1.print();

    Rectangle r2(10, 20);
    r2.print();

    return 0;
}
```

In this example, we define two constructors for the **Rectangle** class: a default constructor that sets the width and height to 0, and a parameterized constructor that takes two integer arguments and initializes the width and height with those values. We also define a destructor that prints a message to the console when the object is destroyed. In the **main()** function, we create two objects of the **Rectangle** class: **r1** using the default constructor, and **r2** using the parameterized constructor with arguments 10 and 20. We then call the **print()** function on both objects to print their width and height to the console. **Classes** can be used to encapsulate data and behavior, and to create objects that represent real-world entities or concepts. They can also be used to define more complex data structures and algorithms, such as linked lists, trees, and graphs. Understanding how to define and use classes is an important part of C++ programming.

In C++, an object is an instance of a class. It is a concrete representation of the data and behavior defined by the class, and can be used to perform operations and interact with other objects and the environment. When you create an object of a **class**, you are essentially creating a new variable of that type. The object has its own set of data members, which are initialized according to the constructor of the class, and its own set of member functions, which can be used to manipulate the data members and perform operations. **Here's an example that demonstrates how to create and use objects in C++:**

```cpp
#include<iostream>

class Rectangle {
public:
    int width, height;

    Rectangle(int w, int h) {
        width = w;
        height = h;
    }

    int area() {
        return width * height;
    }

    void print() {
        std::cout << "Width: " << width << ", Height: " << height << std::endl;
    }
};

int main() {
    Rectangle r1(10, 20);
    r1.print();
    std::cout << "Area: " << r1.area() << std::endl;

    Rectangle r2(5, 15);
    r2.print();
    std::cout << "Area: " << r2.area() << std::endl;

    return 0;
}
```

In this example, we define a class **Rectangle** that has two data members **width** and **height**, a constructor that initializes the data members with the provided values, and two member functions **area()** and **print()** that calculate the area of the rectangle and print its dimensions to the console, respectively. In the **main()** function, we create two objects of the **Rectangle** class, **r1** and **r2**, with different values for the **width** and **height** data members. We then call the **print()** function and the **area()** function on both objects to print their dimensions and calculate their areas. **Objects in C++** are an important tool for representing real-world entities and for creating data structures and algorithms. They provide a way to encapsulate data and behavior into a single entity, and to create multiple instances of that entity with different data values. Understanding how to create and use objects is an essential part of C++ programming.

C++ exceptions are a way to handle runtime errors in a program. Exceptions allow the program to detect and handle errors gracefully, rather than crashing or producing undefined behavior. When an error occurs, an exception object is created and thrown to the nearest enclosing try-catch block that can handle it. If no such block is found, the program terminates. **Here's an example that demonstrates how to use exceptions in C++:**

```cpp
#include<iostream>

int main() {
    int x, y;

    std::cout << "Enter two integers: ";
    std::cin >> x >> y;

    try {
        if (y == 0) {

            throw std::runtime_error("Division by zero");
        }

        std::cout << "Result: " << x / y << std::endl;

    }
```

```
    catch (std::exception& e) {
        std::cout << "Error: " << e.what() << std::endl;
    }


    return 0;
}
```

In this example, we ask the user to enter two integers **x** and **y**, and then attempt to divide **x** by **y**. If **y** is zero, we throw an exception with a message "Division by zero". We catch the exception with a **try-catch** block that handles **std::exception** objects. If an exception is thrown, we print an error message to the console. Here's another example that demonstrates how to define and use custom exception classes:

```
#include<iostream>
#include<string>

class MyException : public std::exception {
private:
    std::string message;
public:
    MyException(std::string msg) {
        message = msg;
    }

    virtual const char* what() const throw() {
        return message.c_str();
    }
};

int main() {

    try {
```

```
        throw MyException("Custom exception");
    }
    catch (std::exception& e) {
        std::cout << "Error: " << e.what() << std::endl;
    }


    return 0;
}
```

In this example, we define a custom exception class **MyException** that inherits from **std::exception**. The **MyException** class has a constructor that takes a message string as input, and a **what()** function that returns the message string when called. In the **main()** function, we throw a **MyException** object with the message **"Custom exception"**, and catch it with a **try-catch** block that handles **std::exception** objects. **Exceptions** are a powerful tool for handling errors and preventing crashes in C++ programs. By throwing and catching exceptions, you can detect and recover from errors gracefully, and provide useful error messages to the user. When designing C++ programs, it's important to consider the potential errors that can occur and to use exceptions to handle them appropriately.

Encapsulation is the concept of hiding the implementation details of a class from the outside world and providing a public interface to access its functionality. **In C++,** encapsulation is achieved through the use of access modifiers like public, private, and protected in the class definition.

Here is an example of encapsulation in C++:

```cpp
#include<iostream>

class BankAccount {
private:
    std::string accountNumber;
    double balance;
public:
    BankAccount(std::string num, double bal) {
        accountNumber = num;
        balance = bal;
    }

    void deposit(double amount) {
        balance += amount;
    }

    void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        }
        else {
            std::cout << "Insufficient balance" << std::endl;
        }
    }

    double getBalance() {
        return balance;
    }
};


int main() {
    BankAccount acc("12345", 1000.0);

    acc.deposit(500.0);
    std::cout << "Balance after deposit: " << acc.getBalance() << std::endl;

    acc.withdraw(2000.0);
    std::cout << "Balance after withdrawal: " << acc.getBalance() << std::endl;

    return 0;
}
```

In this example, we define a class **BankAccount** that has two private data members, **accountNumber** and **balance**, and three public member functions, **deposit()**, **withdraw()**, and **getBalance()**. The **deposit()** and **withdraw()** functions modify the **balance** data member, while the **getBalance()** function returns the **balance** to the calling code. The private data members **accountNumber** and **balance** are not directly accessible from the outside world. The calling code can only interact with the public member functions, which provide a safe and controlled way of accessing and modifying the object's data. This way of encapsulating the data members of a class ensures that they cannot be accidentally modified from outside the class, which makes the code more robust and maintainable.

Inheritance is a key concept in object-oriented programming that allows a new class to be based on an existing class, inheriting its attributes and behavior. In C++, inheritance is implemented using the **class** keyword, followed by the name of the new class, a colon, and the name of the base class. **Here is an example of inheritance in C++:**

```cpp
#include<iostream>

class Shape {
protected:
    int width;
    int height;
public:
    Shape(int w, int h) {
        width = w;
        height = h;
    }

    virtual int area() {
        std::cout << "Parent class area:" << std::endl;

        return 0;

    }
```

```cpp
};
class Rectangle : public Shape {
public:
    Rectangle(int w, int h) : Shape(w, h) {}

    int area() override {
        std::cout << "Rectangle class area:" << std::endl;
        return (width * height);
    }
};
class Triangle : public Shape {
public:
    Triangle(int w, int h) : Shape(w, h) {}

    int area() override {
        std::cout << "Triangle class area:" << std::endl;
        return (width * height / 2);
    }
};

int main() {
    Shape* shape;
    Rectangle rect(10, 5);
    Triangle tri(10, 5);

    shape = &rect;
    std::cout << "Rectangle area: " << shape->area() << std::endl;

    shape = &tri;
    std::cout << "Triangle area: " << shape->area() << std::endl;

    return 0;
}
```

In this example, we define a base class `Shape` that has two protected data members `width` and `height`, and a public member function `area()`. We also define two derived classes `Rectangle` and `Triangle` that inherit from `Shape`. The derived classes `Rectangle` and `Triangle` add their own implementation of the `area()` function, which overrides the implementation in the base class. This is achieved by using the `override` keyword, which tells the compiler to check that the function is actually overriding a base class function. In the `main()` function, we create objects of type `Rectangle` and `Triangle`, and use a pointer of type `Shape*` to point to them. We then call the `area()` function on these objects through the `Shape*` pointer, which invokes the overridden functions in the derived classes. This way of using inheritance allows us to reuse code from the base class and add new functionality in the derived classes, while still maintaining a common interface through the public member functions of the base class.

Polymorphism is a feature of object-oriented programming that allows objects to take on multiple forms or behaviors depending on the context in which they are used. In C++, polymorphism is achieved through the use of virtual functions, which are functions that can be overridden by derived classes. **Polymorphism** can be categorized into two types:

- **Compile-time polymorphism (also known as static polymorphism):** This type of polymorphism is achieved through function overloading and operator overloading.
- **Run-time polymorphism (also known as dynamic polymorphism):** This type of polymorphism is achieved through inheritance and virtual functions.

Run-time polymorphism is the most commonly used type of polymorphism in C++. It allows you to write code that works with objects of different types, as long as they all implement the same interface (i.e. have the same set of virtual functions). When you call a virtual function on a base class pointer or reference, the actual function that gets called depends on the type of the object that the pointer or reference points to. This allows you to write code that works with objects of different types, but still calls the correct function for each type. **For example**, consider a base class `Animal` and two derived classes `Dog` and `Cat`. The `Animal` class has a virtual

function **makeSound()** that is overridden in the **Dog** and **Cat** classes. We can create a pointer of type **Animal*** that can point to objects of type **Dog** and **Cat**. When we call the **makeSound()** function on the **Animal*** pointer, the actual function that gets called depends on the type of the object that the pointer points to. This allows us to write code that works with both **Dog** and **Cat** objects, but still calls the correct **makeSound()** function for each type.

```cpp
#include<iostream>

class Animal {
public:
    virtual void makeSound() {
        std::cout << "Animal sound" << std::endl;
    }
};

class Dog : public Animal {
public:
    void makeSound() override {
        std::cout << "Woof!" << std::endl;
    }
};

class Cat : public Animal {
public:
    void makeSound() override {
        std::cout << "Meow!" << std::endl;
    }
};

int main() {
    Animal* animal;
    Dog dog;
    Cat cat;

    animal = &dog;
    animal->makeSound(); // Output: Woof!

    animal = &cat;
    animal->makeSound(); // Output: Meow!

    return 0;
}
```

In this example, we use the **Animal*** pointer to point to objects of type **Dog** and **Cat**. When we call the **makeSound()** function on the **Animal*** pointer, the actual function that gets called depends on the type of the object that the pointer points to. This allows us to write code that works with both **Dog** and **Cat** objects, but still calls the correct **makeSound()** function for each type.

In C++, a **constructor** is a special member function of a class that is called when an object of that class is created. The purpose of the constructor is to initialize the data members of the object to some initial values. **The syntax for a constructor is as follows:**

```cpp
class ClassName {
public:
    ClassName();    // Constructor declaration
};
```

The constructor has the same name as the class and does not have a return type. It can have parameters, just like a regular function. There are two types of constructors in C++: **default constructor** and **parameterized constructor**. The **default constructor** is a constructor that takes no parameters, and its purpose is to initialize the data members of the object to some default values. If a class does not have any constructors defined, the compiler automatically generates a default constructor for the class. **For example:**

```cpp
#include<iostream>

class Point {
public:
    int x, y;

    Point() {
        x = 0;
        y = 0;
```

```
    }
};

int main() {
    Point p;
    std::cout << p.x << ", " << p.y << std::endl;   // Output: 0, 0

    return 0;
}
```

In this example, we have defined a class **Point** that represents a point in 2D space. We have defined a default constructor that initializes the **x** and **y** data members to 0. When we create an object of the **Point** class using the default constructor, the **x** and **y** data members are initialized to 0. A **parameterized constructor** is a constructor that takes one or more parameters, and its purpose is to initialize the data members of the object to some specific values based on the arguments passed to the constructor. **For example:**

```
#include <iostream>
class Point {
public:
    int x, y;

    Point(int x, int y) {
        this->x = x;
        this->y = y;
    }
};

int main() {
    Point p(1, 2);
    std::cout << p.x << ", " << p.y << std::endl;   // Output: 1, 2

    return 0;
}
```

In this example, we have defined a **parameterized constructor** that takes two integers **x** and **y** as arguments. The constructor initializes the **x** and **y** data members of the object to the values passed as arguments. When we create an object of the **Point** class using the parameterized constructor with arguments 1 and 2, the **x** and **y** data members are initialized to 1 and 2, respectively.

There are several advantages of using C++, some of which are:

- **High performance:** C++ is a compiled language, which means that code written in C++ is compiled into machine code that can be executed directly by the computer's processor. This makes C++ programs very fast and efficient.

- **Object-oriented programming:** C++ supports object-oriented programming (OOP), which allows developers to write code that is organized around objects and classes. OOP makes it easier to write and maintain large, complex programs.

- **Portability:** C++ code can be compiled and run on a wide variety of platforms, including Windows, macOS, Linux, and many others.

- **Standard libraries:** C++ comes with a large number of standard libraries that provide a wide range of functionality, including input/output, strings, math functions, and more. These libraries can save developers a lot of time and effort by providing pre-written code that can be used in their programs.

- **Compatibility with C:** C++ is largely backwards-compatible with C, which means that many C programs can be easily converted to C++.

Here are some examples of applications that use C++:

- **Operating systems:** Many operating systems, including Windows and macOS, are written in C++.

- **Game development:** C++ is a popular language for game development due to its high performance and support for OOP.

- **Web browsers:** Both Google Chrome and Mozilla Firefox are built using C++.

- **Financial software:** C++ is often used in financial software due to its speed and ability to handle large amounts of data.

- **Industrial automation:** Many industrial automation systems use C++ for their control software due to its real-time performance and ability to interface with hardware.

Here are some specific examples of disadvantages of using C++:

- **Memory management:** As mentioned before, C++ requires manual memory management. This can lead to memory leaks, which occur when memory is not properly deallocated, leading to wasted memory and potential crashes. **For example, consider the following code:**

```cpp
int* ptr = new int;
ptr = new int;
```

In this code, memory is allocated twice for **ptr** without deallocating the first allocation. This creates a memory leak.

- **Complexity:** C++ can be complex to learn and use, especially for beginners. **For example, consider the following code:**

```cpp
class Rectangle {
  int width, height;
public:
  Rectangle(int w, int h) {
    width = w;
    height = h;
  }
  int area() {
    return width * height;
  }
};

int main() {
  Rectangle rect(5, 10);
  std::cout << "Area: " << rect.area() << std::endl;
  return 0;
}
```

This code defines a class **Rectangle** with a constructor and a method **area()**. While this is a simple example, it illustrates the complexity of the C++ syntax and the need to understand concepts such as classes and methods.

- **Security vulnerabilities:** C++ is susceptible to security vulnerabilities such as buffer overflows. **For example, consider the following code:**

```cpp
int main() {
  char buffer[8];
  std::strcpy(buffer, "Hello world!");
  return 0;
}
```

This code copies the string "Hello world!" into a buffer that is only 8 bytes long. This can lead to a buffer overflow, where the extra characters overwrite adjacent memory, potentially leading to a security vulnerability.

- **Compiler compatibility issues:** Different compilers can produce different results when compiling C++ code. **For example, consider the following code:**

```cpp
int main() {
  int x = 0;
  std::cout << x++ << ++x << std::endl;
  return 0;
}
```

The output of this code is undefined because the order in which the expressions **x++** and **++x** are evaluated is not specified by the C++ standard. This can lead to compatibility issues between different compilers and environments.

- **Lack of garbage collection:** C++ does not have automatic garbage collection, which means that the programmer is responsible for managing memory manually. For example, consider the following code:

```
int* ptr = new int;
// ...
delete ptr;
```

This code allocates memory for an integer using **new**, and then deallocates the memory using **delete**. If the programmer forgets to deallocate the memory, a memory leak can occur. If the programmer deallocates the memory twice, the program can crash.

- **Slow development time:** C++ can be slower to develop in than other languages such as Python or Ruby, due to its complex syntax and manual memory management.

It's worth noting that many of these disadvantages can be mitigated with careful coding practices and the use of libraries and frameworks that provide higher-level abstractions for memory management and other tasks.

Here are some amazing facts about C++:

- C++ is often described as a "high-level" language, but it was actually created as a successor to the "low-level" language C. The **"++"** in **C++** refers to the fact that it is an "incremented" version of C.
- The **C++ standard** has over 1,400 pages, making it one of the longest programming language standards in existence.
- The name "C++" was chosen by its creator, **Bjarne Stroustrup**, because it is a play on the increment operator "++". He originally wanted to name it **"C with Classes"**, but the name was already taken.
- C++ is known for its ability to perform low-level memory manipulation, but it also includes support for object-oriented programming and other high-level concepts.
- C++ is often used for **performance-critical applications** such as video games and operating systems. This has led to the creation of a community of programmers who are passionate about optimizing their code for maximum speed and efficiency.

- C++ has been used to create some of the **most popular software in the world**, including Microsoft Windows, Adobe Photoshop, and Google Chrome.
- C++ has a reputation for being a difficult language to learn and use, but many **programmers** find it rewarding and even fun to work with.
- There are many puns and jokes related to C++ and programming in general. For example, **"Why did the C++ developer break up with the Java developer? Because she didn't like his class."**
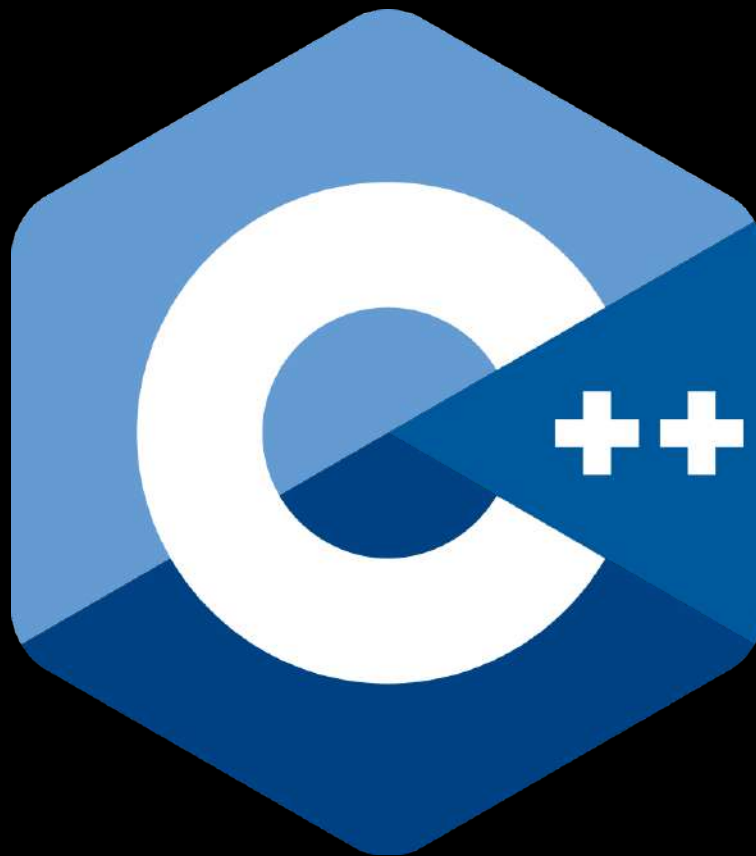
"There are only two kinds of languages: the ones people complain about and the ones nobody uses."

— Bjarne Stroustrup, The C++ Programming Language

JAVA – OVERVIEW

Java is a popular programming language that was created in the mid-1990s by **James Gosling** and his team at **Sun Microsystems**. It is a general-purpose, high-level, object-oriented programming language that is designed to be platform-independent, meaning that it can be run on any operating system or platform that has a **Java Virtual Machine** (JVM). Java is widely used for developing a variety of applications, including desktop applications, web applications, mobile applications, and enterprise applications. It is also used for developing software for **embedded systems** and **Internet of Things** (IoT) devices. Some key features of Java include:

- **Object-Oriented Programming (OOP):** Java is a fully object-oriented programming language, meaning that it uses objects to represent data and behavior.
- **Platform Independence:** Java code can be compiled into bytecode, which can be run on any platform that has a JVM installed.
- **Garbage Collection:** Java has an automatic memory management system that frees up memory automatically when it is no longer needed.
- **Exception Handling:** Java has a built-in mechanism for handling errors and exceptions in a program.
- **Multithreading:** Java supports multithreading, which allows multiple threads to run simultaneously within a program.
- **Standard Library:** Java comes with a vast standard library that provides a wide range of useful functions and classes.

Java is widely used in industries such as finance, healthcare, and technology, and it is also used extensively in education for teaching programming concepts. Many popular software applications, such as **Adobe Creative Suite** and **Minecraft**, are built on Java. Additionally, **Java** is used for developing Android apps, making it a popular choice for mobile development.

Here's an example of how to write "Hello, world!" in Java:

```java
public class MyClass {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Now, let's break down what this code is doing:

- The **public** keyword means that this class is accessible to code outside of its own package.
- The **class** keyword is used to declare a new class.
- **MyClass** is the name of the class. The class name must match the file name.
- The opening and closing curly braces **{ }** define the boundaries of the class definition.
- **public static void main(String[] args)** is the declaration of the main method, which is the entry point of the program. It is required for every Java program.
- The **System.out.println()** statement prints the string "Hello, world!" to the console.

When you run this program, the output will be:

```
Hello, world!
```

Java comments are used to provide information or explanation about the code to other developers who may read the code. **Comments** are not executed by the computer and do not affect the code's functionality. There are three types of comments in Java:

- **Single-line comments** − These comments are created using **//** and are used to explain a single line of code. Everything after the **//** is ignored by the compiler.

    Example:

```
// This is a single-line comment
int x = 5; // this assigns the value 5 to x
```

- **Multi-line comments** − These comments are created using **/*** to start and ***/** to end the comment block. They can span across multiple lines of code.

  Example:
```
/*
This is a multi-line comment
It can span across multiple lines
*/
int y = 10;
```

- **Javadoc comments** − These comments are created using **/**** to start and ***/** to end the comment block. They are used to provide documentation about classes, methods, and fields, and are used by **JavaDoc** tools to create API documentation.

  Example:
```
/**
 * This is a Javadoc comment. It is used to provide documentation for a class,
 * method or field.
 * In this example, we are defining a class called ExampleClass.
 */
public class ExampleClass {
    /**
     * This is a Javadoc comment for the ExampleMethod method. It explains what the
     * method does and what it returns.
     * @param x an integer parameter
     * @return the sum of x and y
     */
    public int ExampleMethod(int x) {
        int y = 5;
        return x + y;
    }
}
```

Overall, **comments** are a useful tool for developers to explain their code and improve its readability.

In Java, a **variable** is a named memory location used to store data. A **variable** can hold different types of data, such as integers, floating-point numbers, characters, and strings. Here are some examples of how to use variables in Java:

1. **Declaring and initializing an integer variable:**

```java
int x;  // declaring an integer variable
x = 10; // initializing the variable with the value 10
```

2. **Declaring and initializing a floating-point variable:**

```java
float y = 3.14f;  // declaring and initializing a float variable with the value 3.14
```

3. **Declaring and initializing a character variable:**

```java
char c = 'a';  // declaring and initializing a character variable with the value 'a'
```

4. **Declaring and initializing a string variable:**

```java
String name = "John";  // declaring and initializing a string variable with the value "John"
```

5. **Declaring and initializing a boolean variable:**

```java
boolean flag = true;  // declaring and initializing a boolean variable with the value true
```

6. **Declaring multiple variables at once:**

```java
int a, b, c;  // declaring three integer variables at once
a = 1;
b = 2;
c = 3;
```

7. Using a variable in an expression:

```
int x = 10;
int y = 20;
int sum = x + y;  // using the variables x and y to calculate the sum
```

Overall, **variables** are an essential part of `Java programming`, and they allow developers to store and manipulate data in their programs.

In Java, data types are used to classify the type of data that a variable can hold. **Java** has two categories of data types: **primitive data types** and **reference data types**.

Primitive Data Types: Primitive data types are the most basic data types in Java, and they are used to store simple values such as integers, floating-point numbers, characters, and boolean values. **There are eight primitive data types in Java:**

- **byte:** Used to store integer values from -128 to 127.
- **short:** Used to store integer values from -32,768 to 32,767.
- **int:** Used to store integer values from -2,147,483,648 to 2,147,483,647.
- **long:** Used to store integer values from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
- **float:** Used to store floating-point numbers with a precision of 6 to 7 decimal digits.
- **double:** Used to store floating-point numbers with a precision of 15 decimal digits.
- **char:** Used to store single characters such as 'a' or 'b'.
- **boolean:** Used to store either true or false values.

Reference Data Types: Reference data types are used to store complex objects and are created using classes or interfaces. These data types refer to a memory location where the object is stored, rather than storing the object directly. **Examples of reference data types include String, Arrays, and Class objects.**

Here are some examples of how to declare and use data types in Java:

1. **Declaring and initializing a variable of type int:**

```
int x = 10; // initializing an int variable with the value 10
```

2. **Declaring and initializing a variable of type double:**

```
double pi = 3.14159; // initializing a double variable with the value 3.14159
```

3. **Declaring and initializing a variable of type char:**

```
char c = 'a'; // initializing a char variable with the value 'a'
```

4. **Declaring and initializing a variable of type String:**

```
String name = "John"; // initializing a String variable with the value "John"
```

5. **Declaring and initializing an array of integers:**

```
int[] arr = {1, 2, 3, 4}; // initializing an integer array with the values 1, 2, 3, and 4
```

Overall, **data types** are used to define the type of data that a variable can hold in Java. The primitive data types are used to store simple values, while reference data types are used to store complex objects. Understanding data types is important for writing **effective Java programs**.

Java Type Casting is the process of converting the value of one data type to another data type. This is necessary when you want to use a value of one data type in an expression that expects another data type. In Java, there are two types of type casting: **explicit and implicit.**

- **Explicit Type Casting:** This is when you convert a data type to another data type explicitly by specifying the target type in parentheses before the value being cast. **For example:**

```
double d = 3.14;
int i = (int) d;
```

In this example, we convert a double value to an int value using explicit type casting. The value of the double variable **d** is cast to an int value and stored in the int variable **i**. Note that this may result in data loss as the decimal part of the double value is truncated.

- **Implicit Type Casting:** This is when you convert a data type to another data type implicitly by the compiler. Implicit type casting occurs when the data type of the target variable is larger than the data type of the source variable. **For example:**

```
int i = 10;
float f = i;
```

In this example, we convert an **int** value to a **float** value using implicit type casting. The value of the int variable **i** is automatically cast to a float value and stored in the float variable **f**. Note that this does not result in data loss as the **float** data type is larger than the **int** data type. It is important to note that type casting can result in data loss or overflow if the target data type is not large enough to hold the value of the source data type. In such cases, you may need to use additional logic to handle such scenarios.

Java operators are symbols or keywords that perform certain operations on one or more operands. They are classified into different categories based on their functionality. **Here are some of the commonly used operators in Java:**

- **Arithmetic Operators:** These operators are used to perform mathematical operations such as addition, subtraction, multiplication, division, and modulus. **The following table shows the arithmetic operators in Java:**

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (Remainder)

- **Assignment Operators:** These operators are used to assign a value to a variable. **The following table shows the assignment operators in Java:**

Operator	Example	Same As
=	a = 10;	a = 10;
+=	a += 10;	a = a + 10;
-=	a -= 10;	a = a - 10;
*=	a *= 10;	a = a * 10;
/=	a /= 10;	a = a / 10;
%=	a %= 10;	a = a % 10;

- **Comparison Operators:** These operators are used to compare two values and return a boolean value (true or false). **The following table shows the comparison operators in Java:**

Operator	Description
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

- **Logical Operators:** These operators are used to combine two or more boolean expressions and return a boolean value. **The following table shows the logical operators in Java:**

Operator	Description

	978
&&	Logical AND
!	Logical NOT

- **Increment and Decrement Operators:** These operators are used to increase or decrease the value of a variable by 1. **The following table shows the increment and decrement operators in Java:**

Operator	Description
++	Increment by 1
--	Decrement by 1

- **Bitwise Operators:** These operators are used to perform bitwise operations on binary numbers. **The following table shows the bitwise operators in Java:**

Operator	Description
&	Bitwise AND

|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift

These are the main categories of Java operators. **Understanding and using these operators correctly is important for writing efficient and effective Java code.**

In Java, a **String** is an object that represents a sequence of characters. **Strings** are widely used in Java programming, and here are some examples of how to work with strings in Java:

1. **Creating a String:**

```java
String greeting = "Hello, world!";
```

In this example, we create a String variable called **greeting** and initialize it with the value
"Hello, world!".

2. **Concatenating Strings:**

```
String firstName = "John";
String lastName = "Doe";
String fullName = firstName + " " + lastName;
```

In this example, we concatenate two String variables, **firstName** and **lastName**, with a space
in between, and store the result in a new String variable called **fullName**.

3. **Getting the Length of a String:**

```
String text = "This is a text string.";
int length = text.length();
```

In this example, we get the length of a String variable called **text** using the **length()** method
and store the result in an integer variable called **length**.

4. **Comparing Strings:**

```
String str1 = "Hello";
String str2 = "World";
boolean result = str1.equals(str2);
```

In this example, we compare two String variables, **str1** and **str2**, using the **equals()** method
and store the result in a boolean variable called **result**.

5. **Converting Strings to Numbers:**

```
String str = "123";
int num = Integer.parseInt(str);
```

In this example, we convert a String variable called **str** to an integer variable using the **parseInt()** method of the **Integer** class.

6. Substring:

```
String text = "This is a text string.";
String subString = text.substring(8, 12);
```

In this example, we get a substring of a String variable called **text** starting from index 8 and ending at **index 12**, and store the result in a new String variable called **subString**. These are just a few examples of how to work with Strings in Java. **Strings** are very versatile and there are many other methods and operations that can be performed on them.

In Java, the **Math class** provides a set of static methods for performing various mathematical operations. Here are some commonly used methods from the **Math class** in Java:

- **Math.abs():** This method returns the absolute value of a number. For example, Math.abs(-5) would return 5.
- **Math.ceil():** This method returns the smallest integer greater than or equal to the specified number. For example, Math.ceil(3.2) would return 4.0.
- **Math.floor():** This method returns the largest integer less than or equal to the specified number. For example, Math.floor(3.9) would return 3.0.
- **Math.round():** This method returns the closest integer to the specified number. For example, Math.round(3.2) would return 3.
- **Math.max():** This method returns the greater of two numbers. For example, Math.max(3, 5) would return 5.
- **Math.min():** This method returns the smaller of two numbers. For example, Math.min(3, 5) would return 3.

- **Math.pow():** This method returns the value of the first argument raised to the power of the second argument. For example, Math.pow(2, 3) would return 8.0.
- **Math.sqrt():** This method returns the square root of a number. For example, Math.sqrt(25) would return 5.0.
- **Math.random():** This method returns a random number between 0.0 and 1.0. For example, Math.random() would return a value between 0.0 and 1.0 (exclusive).
- **Math.PI:** This is a constant in the Math class that represents the value of pi (approximately 3.141592653589793). It can be used in mathematical calculations that involve circles, such as calculating the circumference or area of a circle.

These are just a few examples of the methods available in the **Math** class. There are many other methods available for performing more complex mathematical operations. The **Math** class is an essential part of **Java programming** and is used extensively in many applications.

In Java, **boolean** is a primitive data type that can only have two values: **true** or **false**. Boolean values are used to represent logical values in Java programming. Here are some examples of how to work with **booleans** in Java:

1. **Declaring a Boolean Variable:**

```java
boolean isRaining = true;
```

In this example, we declare a boolean variable called **isRaining** and initialize it with the value **true**.

2. **Comparing Boolean Values:**

```java
boolean a = true;
boolean b = false;
boolean result = a && b;
```

In this example, we compare two boolean variables, **a** and **b**, using the logical AND operator (**&&**) and store the result in a boolean variable called **result**.

3. **Using Boolean Values in Control Statements:**

```java
boolean isRaining = true;
if (isRaining) {
    System.out.println("Bring an umbrella.");
} else {
    System.out.println("Leave the umbrella at home.");
}
```

In this example, we use a boolean variable called **isRaining** in an if-else statement to determine whether to bring an umbrella or leave it at home.

4. **Returning Boolean Values from Methods:**

```java
public static boolean isEven(int num) {
    return (num % 2 == 0);
}
```

In this example, we define a method called **isEven** that takes an integer argument and returns a boolean value indicating whether the number is even or not.

5. **Using Boolean Operators:**

```java
boolean a = true;
boolean b = false;
boolean result1 = a && b;  // logical AND
boolean result2 = a || b;  // logical OR
boolean result3 = !a;      // logical NOT
```

In this example, we use three different **boolean** operators **(&&, ||, and !)** to combine and negate **boolean v**alues. These are just a few examples of how to work with **booleans** in Java. Boolean values are an important part of programming in Java and are used extensively in many applications.

In Java, the **if...else** statement is used to execute different blocks of code depending on whether a particular condition is true or false. The basic syntax for **if...else** is as follows:

```
if (condition) {
  // code to execute if the condition is true
} else {
  // code to execute if the condition is false
}
```

Here, **condition** is a boolean expression that evaluates to either true or false. If the condition is true, the code inside the first set of curly braces is executed. If the condition is false, the code inside the second set of curly braces is executed. For example, let's say we want to write a Java program that checks whether a given number is positive or negative. We can use **if...else** to accomplish this as follows:

```
public class MyClass {
  public static void main(String[] args) {
    int number = -7;

    if (number > 0) {
      System.out.println(number + " is positive");
    } else {
      System.out.println(number + " is negative");
    }
  }
}
```

In this example, we declare an integer variable called **number** and initialize it to −7. We then use **if...else** to check whether **number** is greater than 0. Since −7 is less than 0, the condition is false, so the code inside the else block is executed. This code simply prints a message to the console indicating that **number** is negative. If **number** had been positive, the code inside the **if block** would have been executed instead, printing a message indicating that **number** is positive.

In Java, the **switch** statement provides a way to execute different blocks of code based on the value of a single variable or expression. The basic syntax for **switch** is as follows:

```java
switch (expression) {
  case value1:
    // code to execute if expression == value1
    break;
  case value2:
    // code to execute if expression == value2
    break;
  ...
  case valueN:
    // code to execute if expression == valueN
    break;
  default:
    // code to execute if none of the cases match
}
```

Here, **expression** is the variable or expression that we want to compare against the different cases. Each **case** statement specifies a value that **expression** might take, and the code inside the corresponding block is executed if **expression** has that value. If none of the cases match, the code inside the **default** block is executed. For example, let's say we want to write a Java program that prints the name of a month based on its number (1 for January, 2 for February, etc.). We can use **switch** to accomplish this as follows:

```java
public class MyClass {
  public static void main(String[] args) {
    int month = 5;

    switch (month) {
      case 1:
        System.out.println("January");
        break;
      case 2:
        System.out.println("February");
        break;
      case 3:
        System.out.println("March");
        break;
      case 4:
        System.out.println("April");
        break;
      case 5:
        System.out.println("May");
        break;
      case 6:
        System.out.println("June");
        break;
      case 7:
        System.out.println("July");
        break;
      case 8:
        System.out.println("August");
        break;
      case 9:
        System.out.println("September");
        break;
      case 10:
        System.out.println("October");
        break;
      case 11:
        System.out.println("November");
        break;
      case 12:
        System.out.println("December");
        break;
      default:
        System.out.println("Invalid month number");
    }
  }
}
```

In this example, we declare an integer variable called **month** and initialize it to 5, which corresponds to May. We then use **switch** to check the value of **month**. Since **month** is equal to 5, the case **5** is matched and the code inside the corresponding block is executed, which simply prints the name of the month "May". If **month** had been a different value, the corresponding case block would have been executed instead, or if none of the cases matched, the code inside the **default** block would have been executed.

In Java, loops are used to execute a block of code repeatedly while a certain condition is true. There are three types of loops in Java: **for**, **while**, and **do...while**.

1. **for loop**

The **for** loop is used to execute a block of code for a fixed number of times. The basic syntax for **"for"** is as follows:

```
for (initialization; condition; update) {
  // code to execute repeatedly
}
```

Here, **initialization** is an expression that initializes a loop control variable, **condition** is a boolean expression that is evaluated before each iteration of the loop, and **update** is an expression that is evaluated after each iteration of the loop. The code inside the curly braces is executed repeatedly as long as **condition** is true. **For example**, let's say we want to write a Java program that prints the numbers from 1 to 10. We can use **"for"** to accomplish this as follows:

```java
public class MyClass {
  public static void main(String[] args) {
    for (int i = 1; i <= 10; i++) {
      System.out.println(i);
    }
  }
}
```

In this example, we use **"for"** to initialize a loop control variable **i** to 1, execute the loop as long as **i** is less than or equal to 10, and increment **i** by 1 after each iteration. The code inside the curly braces simply prints the value of **i** to the console.

2. **while loop**

The **while** loop is used to execute a block of code as long as a certain condition is true. The basic syntax for **while** is as follows:

```
while (condition) {
  // code to execute repeatedly
}
```

Here, **condition** is a boolean expression that is evaluated before each iteration of the loop. The code inside the curly braces is executed repeatedly as long as **condition** is true. For example, let's say we want to write a Java program that prints the numbers from 1 to 10 using a **while** loop. We can accomplish this as follows:

```java
public class MyClass {
  public static void main(String[] args) {
    int i = 1;
    while (i <= 10) {
      System.out.println(i);
      i++;
    }
  }
}
```

In this example, we initialize a loop control variable **i** to 1 before entering the loop. We then use **while** to execute the loop as long as **i** is less than or equal to 10. The code inside the curly braces simply prints the value of **i** to the console and increments **i** by 1 after each iteration.

3. do...while loop

The **do...while** loop is similar to the **while** loop, but the condition is evaluated after the code inside the loop is executed, so the code is guaranteed to execute at least once. The basic syntax for **do...while** is as follows:

```
do {
  // code to execute repeatedly
} while (condition);
```

Here, **condition** is a boolean expression that is evaluated after each iteration of the loop. The code inside the curly braces is executed repeatedly until **condition** is false. For example, let's say we want to write a Java program that prints the numbers from 1 to 10 using a **do...while** loop. We can accomplish this as follows:

```java
public class MyClass {
  public static void main(String[] args) {
    int i = 1;
    do {
      System.out.println(i);
      i++;
    } while (i <= 10);
  }
}
```

In this example, we initialize a loop control variable **i** to 1 before entering the loop. We then use **do...while** to execute the loop at least once, printing the value of **i** to the console and incrementing **i** by 1 after each iteration. The loop continues to execute as long as **i** is less than or equal to 10. One important thing to note about **do...while** loops is that the code block inside the loop is guaranteed to execute at least once, regardless of whether the condition is true

or false. This makes **do...while** loops particularly useful when you need to execute a block of code at least once before checking a condition.

In Java, an **array** is a data structure that allows you to store a collection of elements of the same type in a contiguous memory location. **Arrays** are one of the fundamental data structures in programming, and they are used extensively in Java and many other programming languages. To declare an array in Java, you need to specify the type of the elements and the size of the array. **The syntax for declaring an array is as follows:**

```
type[] arrayName = new type[size];
```

Here, **type** is the type of the elements in the array, **arrayName** is the name of the array variable, and **size** is the number of elements in the array. For example, to declare an array of 5 integers, you would use the following code:

```
int[] numbers = new int[5];
```

Once you have declared an array, you can access its elements using an index. The index of the first element in the array is **0**, and the index of the last element is **(size - 1)**. You can access an element in the array using the following syntax:

```
arrayName[index]
```

Here, **arrayName** is the name of the array variable, and **index** is the index of the element you want to access. For example, to assign a value of 10 to the first element of the **numbers** array, you would use the following code:

```
numbers[0] = 10;
```

Arrays can also be initialized with values at the time of declaration using an array initializer. The syntax for initializing an array is as follows:

```
type[] arrayName = {value1, value2, ..., valueN};
```

Here, **type** is the type of the elements in the array, **arrayName** is the name of the array variable, and **value1**, **value2**, ..., **valueN** are the values to be stored in the array. For example, to declare and initialize an array of 3 integers with values of 1, 2, and 3, you would use the following code:

```
int[] numbers = {1, 2, 3};
```

Arrays can also be used with loops to iterate over their elements. For example, to print all the elements in the **numbers** array, you could use a **"for"** loop as follows:

```
for (int i = 0; i < numbers.length; i++) {
    System.out.println(numbers[i]);
}
```

Here, **numbers.length** gives the size of the **numbers** array, and the loop iterates over all the elements in the array, printing each element to the console. **Arrays** are an important data structure in Java, and they are used extensively in many applications for the following reasons:

- **Grouping Data:** Arrays allow you to group related data of the same type into a single data structure. This makes it easier to manage and manipulate the data as a whole.
- **Efficient Access:** Arrays provide efficient access to their elements through their index, which allows you to quickly retrieve or modify a specific element in the array.

- **Iteration:** Arrays can be easily iterated over using loops, which allows you to perform operations on all the elements in the array.

- **Memory Efficiency:** Arrays store their elements in contiguous memory locations, which makes them more memory-efficient than other data structures that require more complex memory allocation.

- **Passing Arrays to Methods:** Arrays can be passed as arguments to methods, which allow you to reuse code and perform operations on arrays in a modular and reusable way.

- **Sorting and Searching:** Arrays provide built-in methods for sorting and searching their elements, which makes it easier to perform these operations without having to write your own algorithms.

In addition to these benefits, `arrays` are also widely used in many algorithms and data structures, such as sorting algorithms, binary search trees, and hash tables. Overall, arrays are an important data structure in Java that provide a powerful and efficient way to group and manage related data.

In Java, a `method` is a block of code that performs a specific task or set of tasks. Methods provide a way to modularize code and reuse it in different parts of a program. Methods can also accept parameters and return values, which make them more flexible and powerful. To define a method in Java, you need to specify its name, return type (if any), and parameter list (if any). **The syntax for defining a method is as follows:**

```
accessModifier returnType methodName(parameterList) {
  // method body
}
```

Here, `accessModifier` specifies the visibility of the method (e.g. `public`, `private`, `protected`, or no modifier), `returnType` specifies the type of value that the method returns (or `void` if it doesn't return anything), `methodName` is the name of the method, and

parameterList is a comma-separated list of parameters that the method accepts (if any). For example, the following method **"add"** takes two integers as parameters and returns their sum:

```java
public int add(int x, int y) {
   int sum = x + y;
   return sum;
}
```

Once a method is defined, it can be called from other parts of the program using its name and passing in the necessary arguments. For example, to call the **"add"** method defined above, you would use the following code:

```java
int result = add(3, 5);
System.out.println(result); // Output: 8
```

Methods can also be overloaded, which means you can define multiple methods with the same name but different parameter lists. Java determines which method to call based on the number and types of arguments passed to the method. **For example**, the following method **"add"** is an overloaded version of the method defined above that takes three integers as parameters and returns their sum:

```java
public int add(int x, int y, int z) {
   int sum = x + y + z;
   return sum;
}
```

Overall, **methods** are an important concept in Java that provides a powerful way to organize code and make it more reusable and flexible.

Here's an example of a simple method in Java:

```java
public class Example {

    public static void main(String[] args) {

        // Call the hello method

        hello();

    }


    // Define the hello method

    public static void hello() {

        System.out.println("Hello, world!");

    }

}
```

In this example, we define a class called **Example** that contains a single method called **hello**. The **"hello"** method simply prints the string "Hello, world!" to the console using the **System.out.println** statement. In the **main** method, we call the **hello** method using the method name and the parentheses operator. When the program runs, it will print the message "Hello, world!" to the console. Here's another example of a method that accepts parameters and returns a value:

```java
public class Example {
    public static void main(String[] args) {
        // Call the add method with two integers
        int result = add(2, 3);
        System.out.println(result); // Output: 5
    }


    // Define the add method that accepts two integers and returns their sum
    public static int add(int x, int y) {
        int sum = x + y;
        return sum;
    }
}
```

In this example, we define a method called **add** that accepts two integer parameters and returns their sum. In the **main** method, we call the **add** method with the integers 2 and 3, and assign the result to the **"result"** variable. We then print the value of **"result"** to the console, which should be 5.

Java is an object-oriented programming language, which means that it revolves around the concept of classes and objects. **Classes** are blueprints for creating objects, and objects are instances of a class. In simpler terms, a class defines the properties and behaviors of an object, while an **object** is a specific instance of that class with its own unique values. Let's take an example to understand this concept better. Suppose we want to create a class named **"Person"** that has the properties of **"name"**, **"age"**, and **"gender"**. We can define the class as follows:

```java
public class Person {
    String name;
    int age;
    char gender;
}
```

In this class, we have defined three properties: **"name"** (a string), **"age"** (an integer), and **"gender"** (a character). Now that we have defined our class, we can create an object of this class using the **"new"** keyword as follows:

```java
Person person1 = new Person();
```

Here, we have created an object named **"person1"** of the **"Person"** class. Now, we can set the values for the properties of this object as follows:

```java
person1.name = "John";
person1.age = 25;
person1.gender = 'M';
```

We can also create multiple objects of the same class, each with its own set of values, as follows:

```java
Person person2 = new Person();
person2.name = "Jane";
person2.age = 30;
person2.gender = 'F';
```

Now we have two objects of the **"Person"** class: **"person1"** and **"person2"**, each with their own set of values for the properties. In addition to properties, classes can also have methods, which are functions that can be called on an **object** of the class. For example, we can add a method to the **"Person"** class that prints out the name and age of the person as follows:

```java
public class Person {
    String name;
    int age;
    char gender;

    public void printDetails() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

Now, we can call the **"printDetails"** method on any object of the **"Person"** class as follows:

```java
person1.printDetails(); // Output: Name: John, Age: 25
person2.printDetails(); // Output: Name: Jane, Age: 30
```

Overall, **classes** are the blueprints for creating objects, and objects are specific instances of a class with their own set of values for the properties. **Classes** can have properties and methods, which define the characteristics and behavior of the objects.

In Java, `modifiers` are keywords that can be added to classes, methods, and variables to modify their behavior or characteristics. There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

Access modifiers control the accessibility of classes, methods, and variables from other parts of the program. There are four access modifiers in Java: public, private, protected, and default.

- **Public:** A public class, method, or variable can be accessed from any other part of the program.
- **Private:** A private class, method, or variable can only be accessed within the same class.
- **Protected:** A protected class, method, or variable can be accessed from within the same package or from a subclass in a different package.
- **Default:** A default (package-private) class, method, or variable can only be accessed within the same package.

Non-access modifiers modify the behavior or characteristics of classes, methods, and variables. There are several non-access modifiers in Java:

- **Final:** A final variable cannot be changed once it is initialized. A final method cannot be overridden in a subclass, and a final class cannot be subclassed.
- **Static:** A static variable or method belongs to the class rather than an instance of the class.
- **Abstract:** An abstract class or method is declared but not defined. An abstract class cannot be instantiated, and an abstract method must be implemented by a subclass.
- **Synchronized:** A synchronized method can only be accessed by one thread at a time, ensuring that the method is thread-safe.
- **Volatile:** A volatile variable is not cached, and its value is always read from the main memory. This is useful for variables that are shared between threads.

By using `modifiers` effectively, you can control the behavior and accessibility of your classes, methods, and variables, making your code more robust and efficient.

In Java, `encapsulation` is a mechanism that allows you to control access to the internal state of an object. It is one of the fundamental principles of object-oriented programming and is achieved through the use of access modifiers and getter or setter methods. **Encapsulation** ensures that the internal state of an object is only modified through well-defined methods, preventing other parts of the program from directly accessing or modifying the object's data. This provides several benefits, including:

- **Improved security:** Encapsulation ensures that the internal state of an object cannot be accessed or modified by unauthorized parts of the program.
- **Improved maintainability:** Encapsulation makes it easier to change the internal implementation of an object without affecting other parts of the program.
- **Improved flexibility:** Encapsulation allows you to define a well-defined interface for an object, making it easier to use in different parts of the program.

Here is an example of encapsulation in Java:

```java
public class BankAccount {
    private double balance;

    public BankAccount(double initialBalance) {
        balance = initialBalance;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
        }
    }

    public double getBalance() {
        return balance;
    }
}
```

In this example, the **balance** variable is declared as **private**, which means it cannot be accessed or modified directly by other parts of the program. Instead, the **deposit** and **withdraw** methods are used to modify the balance. The **getBalance** method is defined as **public**, which allows other parts of the program to retrieve the current balance. This method provides a well-defined interface for accessing the internal state of the **BankAccount** object, ensuring that it is only modified through the **deposit** and **withdraw** methods. By using encapsulation in your **Java programs**, you can ensure that your objects are well-defined and secure, making them easier to maintain and more flexible to use in different parts of your program.

In Java, a **package** is a way of organizing related classes and interfaces into a single unit of code. Packages help to avoid naming conflicts and provide a mechanism for access control. A package can contain sub-packages, classes, interfaces, and other resources like images, audio files, etc. **Java packages** are identified by their package name, which is a unique identifier that is used to distinguish them from other packages. The package name is typically written in reverse domain name notation, such as **com.example.myapp**. Here is an example of how to declare a package in Java:

```java
package com.example.myapp;

public class MyClass {
    // class definition
}
```

In this example, the **MyClass** class is declared in the **com.example.myapp** package. To use a class from another package, you must either import the class or use its fully qualified name. For example:

```java
import com.example.myapp.MyClass;

public class AnotherClass {
    MyClass myObject = new MyClass();
}
```

In this example, the **MyClass** class is imported using the **import** statement, and an instance of **MyClass** is created using the **new** keyword. If you don't want to use an **import** statement, you can use the fully qualified name of the class instead:

```java
public class AnotherClass {
    com.example.myapp.MyClass myObject = new com.example.myapp.MyClass();
}
```

In this example, the fully qualified name of the **MyClass** class is used to create an instance of the class. By using packages in your Java programs, you can organize your code into logical units, avoid naming conflicts, and provide a mechanism for access control.

In Java, **inheritance** is a mechanism that allows you to create a new class based on an existing class. The new class, called the **subclass**, inherits the properties and behavior of the existing class, called the **superclass**, and can also add its own properties and behavior. **Inheritance** is one of the fundamental principles of object-oriented programming and is used to create a hierarchy of related classes. The **superclass** is at the top of the hierarchy, and the subclasses are below it. **Here is an example of inheritance in Java:**

```java
public class Animal {
    private String name;
    private int age;

    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public void eat() {
        System.out.println(name + " is eating.");
    }
}
```

```java
    public void sleep() {
        System.out.println(name + " is sleeping.");
    }
}


public class Cat extends Animal {
    public Cat(String name, int age) {
        super(name, age);
    }


    public void meow() {
        System.out.println("Meow!");
    }
}
```

In this example, the **Animal** class is the superclass, and the **Cat** class is the subclass. The **Cat** class inherits the **name** and **age** properties from the **Animal** class and adds its own method, **meow**. To create an instance of the **Cat** class, you can use the following code:

```java
Cat myCat = new Cat("Fluffy", 2);
myCat.eat();   // outputs "Fluffy is eating."
myCat.sleep(); // outputs "Fluffy is sleeping."
myCat.meow();  // outputs "Meow!"
```

In this example, the **myCat** object is an instance of the **Cat** class and can access the **eat**, **sleep**, and **meow** methods. By using inheritance in your Java programs, you can create a hierarchy of related classes that share common properties and behavior, making your code more modular and easier to maintain.

In Java, an inner class is a class that is defined inside another class. Inner classes are a powerful feature of Java that allows you to logically group classes and interfaces in one place, and to encapsulate them inside another class.

There are four types of inner classes in Java:

- **Member Inner Class:** A member inner class is defined inside a class and can access the members of the enclosing class, including private members. To create an instance of a member inner class, you must first create an instance of the enclosing class.

- **Local Inner Class:** A local inner class is defined inside a method or a block, and can only be accessed within that method or block. Local inner classes are typically used to define an implementation of an interface or an abstract class.

- **Anonymous Inner Class:** An anonymous inner class is a local inner class that does not have a name. Anonymous inner classes are typically used to define an implementation of an interface or an abstract class on the fly.

- **Static Nested Class:** A static nested class is a class that is defined inside another class, but is not an inner class. Static nested classes can only access static members of the enclosing class.

Here is an example of a member inner class:

```java
public class OuterClass {
    private int x = 10;

    class InnerClass {
        public void printX() {
            System.out.println(x);
        }
    }
}
```

In this example, the **InnerClass** is a member inner class of the **OuterClass**. The **printX** method of the **InnerClass** can access the private **x** variable of the **OuterClass**. To create an instance of the **InnerClass,** you must first create an instance of the **OuterClass**:

```
OuterClass outerObject = new OuterClass();
OuterClass.InnerClass innerObject = outerObject.new InnerClass();
innerObject.printX(); // outputs "10"
```

In this example, the **innerObject** is an instance of the **InnerClass** and can access the private **x** variable of the **OuterClass**. By using inner classes in your **Java programs**, you can create more modular and organized code, and encapsulate related classes and interfaces inside another class. Inner classes can also access the members of the enclosing class, which can be useful in certain situations.

Java abstraction and interfaces are two important concepts in object-oriented programming that help in achieving modular, maintainable, and reusable code. **Abstraction** is the process of hiding the implementation details of a class or an object and exposing only the necessary information to the user. It is one of the key principles of object-oriented programming and is used to simplify complex systems by breaking them down into smaller, more manageable pieces. **In Java**, **abstraction** is achieved through abstract classes and interfaces. An **abstract class** is a class that cannot be instantiated, and is used as a base class for other classes. It contains one or more abstract methods, which are declared but not implemented in the abstract class. The subclasses of an abstract class must implement all the abstract methods declared in the abstract class. **Here is an example of an abstract class in Java:**

```java
public abstract class Shape {
    protected String color;

    public Shape(String color) {
        this.color = color;
    }

public abstract double getArea();

public abstract double getPerimeter();
```

```
public String getColor() {
        return color;
    }
}
```

In this example, the **Shape** class is an abstract class that contains two abstract methods **getArea** and **getPerimeter**. The subclasses of the **Shape** class, such as **Circle** and **Rectangle**, must implement these methods. An **interface** in Java is similar to an abstract class, but it only contains abstract methods and constant fields. An **interface** is a contract between the interface and the implementing class, specifying the methods that must be implemented by the implementing class. **Here is an example of an interface in Java:**

```
public interface Drawable {
    public void draw();
}
```

In this example, the **Drawable** interface contains only one method **draw**. Any class that implements the **Drawable** interface must implement the **draw** method. Interfaces can also extend other interfaces, allowing for **multiple inheritance**. **Here is an example:**

```
public interface Shape extends Drawable {
    public double getArea();
    public double getPerimeter();
}
```

In this example, the **Shape** interface extends the **Drawable** interface and contains two abstract methods **getArea** and **getPerimeter**. By using **abstraction and interfaces** in your Java

programs, you can create modular, maintainable, and reusable code. **Abstraction** allows you to hide the implementation details of a class and expose only the necessary information to the user, while interfaces allow you to define a contract between the interface and the implementing class, specifying the methods that must be implemented by the implementing class.

In Java, an **enum** is a special type of class that represents a fixed set of constants. Enums are useful when you have a predefined set of values that a variable can take, and you want to restrict the values that the variable can have. An **enum** is defined using the enum keyword, and the constants are listed inside the curly braces. **Here is an example of an enum in Java:**

```java
public enum Day {
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
}
```

In this example, the **Day** enum represents the days of the week, and the constants are the days themselves. Each constant is implicitly declared as a public static final field of the enum type. You can use an **enum** in your Java program by declaring a variable of the enum type. Here is an example:

```java
Day today = Day.MONDAY;
```

In this example, the variable **today** is declared as an instance of the **Day** enum, and is initialized to the **MONDAY** constant. You can also use enums in switch statements, which can make your code more readable and maintainable. **Here is an example:**

```java
switch (today) {
    case MONDAY:
        System.out.println("It's Monday!");
        break;
    case TUESDAY:
        System.out.println("It's Tuesday!");
        break;
    // ... other cases ...
    default:
        System.out.println("It's not a weekday!");
        break;
}
```

In this example, the **switch** statement is used to print a message depending on the value of the **today** variable. Enums can also have fields, constructors, and methods, just like regular classes.
Here is an example:

```java
public enum Day {
    MONDAY("Monday"),
    TUESDAY("Tuesday"),
    WEDNESDAY("Wednesday"),
    THURSDAY("Thursday"),
    FRIDAY("Friday"),
    SATURDAY("Saturday"),
    SUNDAY("Sunday");

    private final String name;

    private Day(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

In this example, the **Day** enum has a **name** field, a constructor that takes a **name** parameter, and a **getName** method that returns the **name** field. **Enums** are a powerful feature of Java that can make your code more readable and maintainable by restricting the values that a variable can have, and by providing a type-safe way to represent a fixed set of constants.

Here's an example of how to take user input in Java using the Scanner class:

```java
import java.util.Scanner;

public class MyClass {

    public static void main(String[] args) {

        // create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);

        // prompt the user to enter their name
        System.out.print("Please enter your name: ");

        // read the user's input as a String
        String name = scanner.nextLine();

        // prompt the user to enter their age
        System.out.print("Please enter your age: ");

        // read the user's input as an integer
        int age = scanner.nextInt();

        // print out the user's name and age
        System.out.println("Hello, " + name + ". You are " + age + " years old.");

        // remember to close the Scanner object
        scanner.close();
    }
}
```

In this example, we use the **Scanner class** to read user input from the console. We first create a **Scanner** object and pass in **System.in** as the argument, which represents the standard input stream **(i.e., the console)**. We then prompt the user to enter their name using the **System.out.print()** method, and read their input as a String using the **nextLine()** method of the **Scanner** object. We do the same for the user's age, but read the input as an integer using the **nextInt()** method. Finally, we print out the user's name and age using the **System.out.println()** method, and remember to close the **Scanner** object using the **close()** method.

In Java, **ArrayList** is a class that provides a resizable array implementation. Unlike regular arrays, **ArrayList** can dynamically grow or shrink its size as needed, making it more flexible and convenient to use in certain scenarios. Here's an example of how to create and use an **ArrayList** in Java:

```java
import java.util.ArrayList;

public class MyClass {
    public static void main(String[] args) {
        // create an empty ArrayList of integers
        ArrayList<Integer> numbers = new ArrayList<>();

        // add some numbers to the list
        numbers.add(5);
        numbers.add(3);
        numbers.add(8);
        numbers.add(1);

    // print out the entire list

    System.out.println(numbers); // output: [5, 3, 8, 1]
```

```java
        // get the size of the list
        int size = numbers.size();
        System.out.println("The size of the list is " + size);
        // output: The size of the list is 4


        // access an element by index
        int first = numbers.get(0);
        System.out.println("The first element is " + first);
     // output: The first element is 5


        // modify an element by index
        numbers.set(1, 7);
        System.out.println(numbers); // output: [5, 7, 8, 1]


        // remove an element by index
        numbers.remove(2);
        System.out.println(numbers); // output: [5, 7, 1]


        // check if an element is in the list
        boolean containsSeven = numbers.contains(7);
        System.out.println("Does the list contain 7? " + containsSeven);
        // output: Does the list contain 7? true


        // iterate through the list
        for (int number : numbers) {
            System.out.println(number);
        }
    }
}
```

In this example, we first create an empty **ArrayList** of integers by using the **ArrayList<Integer>** syntax. We then add some numbers to the list using the **add()** method, and print out the entire list using the **System.out.println()** method. We then demonstrate some common methods of **ArrayList**, including getting the size of the list using the **size()** method, accessing an element by index using the **get()** method, modifying an element by index using the **set()** method, removing an element by index using the **remove()** method, checking if an element is in the list using the **contains()** method, and iterating through the list using a for-each loop. Note that **ArrayList** can also store objects of other types besides integers, such as strings or custom objects, by simply changing the type parameter in the **ArrayList<>** syntax.

In Java, **HashMap** is a class that provides a hash table implementation of the **Map** interface. A **HashMap** stores key-value pairs, where each key is unique and maps to a corresponding value. **HashMap** provides fast lookup and insertion operations, making it useful in many applications. **Here's an example of how to create and use a HashMap in Java:**

```java
import java.util.HashMap;

public class MyClass {
public static void main(String[] args) {

// create an empty HashMap with keys of type String and values of type Integer
        HashMap<String, Integer> scores = new HashMap<>();

// add some scores to the map
        scores.put("Alice", 80);
        scores.put("Bob", 90);
        scores.put("Charlie", 85);

// get the score for a specific key
        int aliceScore = scores.get("Alice");
```

```java
System.out.println("Alice's score is " + aliceScore); // output: Alice's score is 80


        // check if a key is in the map
        boolean hasBob = scores.containsKey("Bob");
        System.out.println("Does the map have Bob? " + hasBob);
    // output: Does the map have Bob? true


        // update the score for a key
        scores.put("Charlie", 90);
        System.out.println("Charlie's new score is " + scores.get("Charlie"));
    // output: Charlie's new score is 90


        // remove a key-value pair from the map
        scores.remove("Alice");
        System.out.println("The map now has " + scores.size() + " entries");
    // output: The map now has 2 entries


        // iterate through the keys in the map
        for (String key : scores.keySet()) {
            int value = scores.get(key);
            System.out.println(key + ": " + value);
        }
    }
}
```

In this example, we first create an empty **HashMap** with keys of type **String** and values of type **Integer** by using the **HashMap<String, Integer>** syntax. We then add some key-value pairs to the map using the **put()** method, and retrieve a value for a specific key using the **get()** method. We then demonstrate some common methods of **HashMap**, including checking if a key is in the map using the **containsKey()** method, updating the value for a key using the **put()** method, removing a key-value pair from the map using the **remove()** method, and iterating through the keys in the map using the **keySet()** method and a for-each loop. Note that

HashMap can also store objects of other types besides strings and integers, such as custom objects, by simply changing the type parameters in the **HashMap<>** syntax.

In Java, **HashSet** is a class that provides a hash table implementation of the **Set** interface. A **HashSet** stores a collection of unique elements, where the order of the elements is not guaranteed. **HashSet** provides fast insertion and lookup operations, making it useful in many applications where uniqueness of elements is a requirement. Here's an example of how to create and use a **HashSet** in Java:

```java
import java.util.HashSet;

public class MyClass {

public static void main(String[] args) {

    // create an empty HashSet of strings
    HashSet<String> names = new HashSet<>();

    // add some names to the set
    names.add("Alice");
    names.add("Bob");
    names.add("Charlie");

    // add a duplicate name
    names.add("Alice");

// print out the entire set
System.out.println(names); // output: [Charlie, Alice, Bob]

    // get the size of the set
    int size = names.size();
    System.out.println("The size of the set is " + size);
    // output: The size of the set is 3
```

```java
        // check if an element is in the set
         boolean containsBob = names.contains("Bob");
         System.out.println("Does the set contain Bob? " + containsBob);
        // output: Does the set contain Bob? true

         // remove an element from the set
         names.remove("Charlie");
         System.out.println(names); // output: [Alice, Bob]

         // iterate through the set
         for (String name : names) {
             System.out.println(name);
         }
     }
 }
```

In this example, we first create an empty **HashSet** of strings by using the **HashSet<String>** syntax. We then add some names to the set using the **add()** method, including a duplicate name to demonstrate that **HashSet** only stores unique elements. We print out the entire set using the **System.out.println()** method, which shows that the order of the elements is not guaranteed. We then demonstrate some common methods of **HashSet**, including getting the size of the set using the **size()** method, checking if an element is in the set using the **contains()** method, removing an element from the set using the **remove()** method, and iterating through the set using a **for-each** loop. Note that **HashSet** can also store objects of other types besides strings, such as integers or custom objects, by simply changing the type parameter in the **HashSet<>** syntax.

In Java, **Iterator** is an interface that provides a way to traverse a collection of elements, such as an **ArrayList** or a **HashSet**. An **Iterator** allows you to sequentially access the elements

in a collection, one at a time, without exposing the underlying implementation of the collection. Here's an example of how to use an **Iterator** in Java:

```java
import java.util.ArrayList;
import java.util.Iterator;

public class MyClass {
    public static void main(String[] args) {
        // create an ArrayList of integers
        ArrayList<Integer> numbers = new ArrayList<>();
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);

        // create an iterator for the ArrayList
        Iterator<Integer> iterator = numbers.iterator();

        // iterate through the ArrayList using the iterator
        while (iterator.hasNext()) {
            int number = iterator.next();
            System.out.println(number);
        }
    }
}
```

In this example, we first create an **ArrayList** of integers and add some elements to it. We then create an **Iterator** for the **ArrayList** using the **iterator()** method. We iterate through the **ArrayList** using the **hasNext()** and **next()** methods of the **Iterator** interface, which allow us to check if there are more elements in the collection and retrieve the next element, respectively. **The output of this program would be:**

```
1


2


3
```

Note that the **Iterator** interface also provides a **remove()** method, which allows you to remove the current element from the collection while iterating through it. This method should only be used when iterating through a collection using an **Iterator**, as it may cause unexpected behavior when used with other types of loops.

In Java, **wrapper classes** are classes that provide a way to represent primitive data types (such as **int**, **double**, **boolean**, etc.) as objects. This is useful when you need to treat primitive types as objects, for example when you need to pass them to methods that require objects as arguments. **Here is a list of the eight wrapper classes in Java:**

- **Boolean:** wraps a boolean value
- **Byte:** wraps a byte value
- **Short:** wraps a short value
- **Integer:** wraps an int value
- **Long:** wraps a long value
- **Float:** wraps a float value
- **Double:** wraps a double value
- **Character:** wraps a char value

Here's an example of how to use a wrapper class in Java:

```java
Integer myInt = new Integer(42);
System.out.println("The value of myInt is " + myInt.toString());
```

In this example, we create an **Integer** object **myInt** that wraps the **int** value 42. We then use the **toString()** method of the **Integer** class to convert the value to a string and print it out. **Wrapper classes** also provide useful methods for converting between primitive types and objects, and for performing arithmetic and comparison operations on objects. **For example:**

```java
Integer x = 5;
int y = x.intValue(); // convert Integer to int
Integer z = Integer.valueOf(y); // convert int to Integer
System.out.println(x.compareTo(z)); // compare two Integer objects
```

In this example, we create an **Integer** object **x** with the value 5, and then use the **intValue()** method to convert it to an **int** value **y**. We then use the **valueOf()** method of the **Integer** class to convert **y** back to an **Integer** object **z**. Finally, we use the **compareTo()** method of the **Integer** class to compare **x** and **z** and print out the result.

In Java, **exceptions** are a way to handle errors or unexpected situations that occur during the execution of a program. The **try-catch** block is a mechanism in Java for handling exceptions. It consists of two parts:

- The try block, where the code that might throw an exception is placed.
- The catch block(s), where the exception is caught and handled.

The syntax of a try-catch block is as follows:

```java
try {
    // code that might throw an exception
} catch (ExceptionType1 e1) {
    // code to handle exception of type ExceptionType1
} catch (ExceptionType2 e2) {
    // code to handle exception of type ExceptionType2
} finally {
    // optional code that always executes, whether or not an exception is thrown
}
```

When an **exception** is thrown within the try block, the catch block is executed. If the exception matches the type of the first catch block, that block is executed. If not, the exception is passed to the next catch block, and so on. If no catch block matches the exception type, the exception is not caught and the program terminates. The finally block is optional and always executes, whether or not an exception is thrown. This block is typically used to perform cleanup tasks, such as closing files or releasing resources, which need to be done regardless of whether an exception occurs. **Here's an example of using a try-catch block in Java:**

```java
try {
    int result = 10 / 0; // this will throw an ArithmeticException
} catch (ArithmeticException e) {
    System.out.println("Error: " + e.getMessage());
} finally {
    System.out.println("This code always executes");
}
```

In this example, the **try block** performs a division by zero, which will throw an ArithmeticException. The catch block catches the exception, prints an error message, and the program continues to execute. The finally block prints a message indicating that it always executes.

In Java, **regular expressions** are a powerful tool for searching, replacing, and manipulating text. A regular expression, also known as **regex**, is a pattern that describes a set of strings. Java provides the **java.util.regex** package for working with regular expressions. The two main classes in this package are Pattern and Matcher.

Here's an example of using **regular expressions** in Java to match a string:

```java
import java.util.regex.*;

public class MyClass {
    public static void main(String[] args) {
        String input = "The quick brown fox jumps over the lazy dog.";
        Pattern pattern = Pattern.compile("fox");
        Matcher matcher = pattern.matcher(input);
        if (matcher.find()) {
            System.out.println("Match found!");
        } else {
            System.out.println("Match not found.");
        }
    }
}
```

In this example, we create a Pattern object using the **compile()** method of the Pattern class. The argument to **compile()** is the regular expression we want to match, which in this case is the string "fox". We then create a Matcher object using the **matcher()** method of the **Pattern** class, passing in the input string we want to search. We call the **find()** method of the **Matcher** object to search for the pattern within the input string. If the pattern is found, the **find()** method returns true, and we print a message indicating that a match was found.

Some common regular expression syntax in Java includes:

- `. : matches any single character`
- `* : matches zero or more occurrences of the preceding character or group`
- `+ : matches one or more occurrences of the preceding character or group`
- `? : matches zero or one occurrence of the preceding character or group`
- `[] : matches any one of the characters within the brackets`
- `() : groups a series of characters together as a subexpression`

There are many more syntax elements available for **regular expressions** in Java, including character classes, quantifiers, and special characters. **Regular expressions** can be complex, but they are a powerful tool for manipulating text in Java.

In Java, **threads** are a mechanism for running multiple tasks concurrently within a single program. A **thread** is a lightweight sub-process that can run in parallel with other threads within the same program. In Java, **threads** are implemented using the **java.lang.Thread** class. To create a new thread, you can either extend the **Thread** class and override its **run()** method, or implement the **Runnable interface** and pass an instance of that class to a **Thread** object. Here's an example of creating a new thread by extending the **Thread** class:

```java
public class MyThread extends Thread {
    public void run() {
        System.out.println("Thread running!");
    }
}

public class Main {
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.start();
    }
}
```

In this example, we create a new thread by extending the **Thread** class and overriding its **run()** method. We then create an instance of this class and call its **start()** method. This starts a new thread, and the code within the **run()** method is executed concurrently with the main thread. Java also provides the **java.util.concurrent** package for working with threads and managing concurrent access to shared resources. This package includes classes such as

Executor, **ThreadPoolExecutor**, and **FutureTask**, which can be used to create and manage threads more easily and efficiently. When working with threads, it's important to be aware of thread synchronization issues, such as race conditions and deadlocks. Java provides synchronization mechanisms, such as synchronized blocks and methods, to help prevent these issues. Here's an example of using a **synchronized block** in Java to prevent multiple threads from accessing a shared resource at the same time:

```java
public class MyRunnable implements Runnable {
    private int counter = 0;

    public void run() {
        synchronized(this) {
            for (int i = 0; i < 10; i++) {
                counter++;
            }
        }
    }
}


public class Main {
    public static void main(String[] args) {
        MyRunnable runnable = new MyRunnable();
        Thread thread1 = new Thread(runnable);
        Thread thread2 = new Thread(runnable);
        thread1.start();
        thread2.start();
        // wait for threads to finish
        try {
            thread1.join();
            thread2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Counter: " + runnable.counter);
    }
}
```

In this example, we create a new **Runnable** object that contains a counter variable. We then create two Thread objects that share the same Runnable object, and start both threads. The code within the **run()** method increments the counter variable using a synchronized block, which ensures that only one thread can access the block at a time. Finally, we wait for both threads to finish using the **join()** method, and print the final value of the counter variable.

In Java, the **java.io** package provides classes for working with files and directories. The File class represents a file or directory on the file system, and provides methods for creating, deleting, reading, and writing files. **Here's an example of creating a new file and writing to it in Java:**

```java
import java.io.*;

public class FileExample {
    public static void main(String[] args) {
        try {
            File file = new File("myfile.txt");
            FileWriter writer = new FileWriter(file);
            writer.write("Hello, world!");
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

In this example, we create a new File object called **"myfile.txt"** using the File constructor. We then create a **FileWriter** object and pass in the File object to write to the file. We write the string "Hello, world!" to the file using the **write()** method of the **FileWriter** object, and then

close the writer using the **close()** method. Java also provides classes for reading from files, such as **FileReader** and **BufferedReader**. Here's an example of reading from a file using **BufferedReader**:

```java
import java.io.*;

public class FileExample {
    public static void main(String[] args) {
        try {
            File file = new File("myfile.txt");
            BufferedReader reader = new BufferedReader(new FileReader(file));
            String line = null;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
            reader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

In this example, we create a new **BufferedReader** object that wraps a **FileReader** object. We read each line from the file using the **readLine()** method of the **BufferedReader** object, and print it to the console using **System.out.println()**. We continue reading lines until we reach the end of the file, indicated by a null return value from **readLine()**. Finally, we close the reader using the **close()** method. Other useful methods provided by the File class include:

- **exists()** : checks whether a file or directory exists
- **isFile()** : checks whether a file exists and is a regular file
- **isDirectory()** : checks whether a file exists and is a directory

- **delete()** : deletes a file or directory
- **mkdir()** : creates a new directory
- **list()** : returns an array of the names of files and directories in a directory

These methods can be used to perform a wide range of file and directory operations in Java.

Java has several advantages that make it a popular programming language for developing a wide range of applications. Some of these advantages include:

- **Platform independence:** Java is platform-independent, meaning that a Java program can run on any platform without requiring any modification. This is because Java programs are compiled into **bytecode** that can be run on any platform that has a **Java Virtual Machine** (JVM) installed. **For example**, a Java program written on Windows can be run on a Linux or Mac OS X system without any modifications.

- **Object-oriented programming:** Java is an object-oriented programming **(OOP)** language, which means that it follows a programming paradigm based on the concept of objects. This makes it easier to write reusable and modular code, which can help reduce development time and increase code maintainability. **For example**, a Java program can define classes that represent real-world entities, such as customers or products, and use them to create instances of those entities.

- **Large standard library:** Java has a large standard library that provides a wide range of useful classes and methods for performing common programming tasks, such as reading and writing files, networking, and database access. This can help reduce the amount of custom code that developers need to write, and make it easier to develop complex applications. **For example**, the java.net package provides classes for performing network operations, such as opening and closing sockets, while the java.util package provides classes for working with collections, such as arrays and lists.

- **Memory management:** Java provides automatic memory management, meaning that the Java Virtual Machine automatically handles memory allocation and deallocation for Java programs. This helps reduce the likelihood of memory leaks and other memory-related errors that can cause programs to crash or behave unpredictably.

- **Multithreading:** Java provides built-in support for multithreading, which allows multiple threads to run concurrently within the same program. This can help improve program performance by allowing tasks to be executed in parallel, and can also make it easier to write concurrent programs that handle multiple users or tasks. **For example**, a Java web application can use multithreading to handle multiple requests from different users simultaneously.

Overall, these advantages make **Java** a versatile and popular programming language that can be used for a wide range of applications, from desktop and mobile applications to web and enterprise applications. Using **Java** has certain drawbacks in addition to its many benefits. **The following are some of the primary drawbacks of Java:**

- **Performance:** Java is an interpreted language which means it can be slower than compiled languages like C++ or Fortran. **For example**, if you are running a CPU-intensive application like a video game, Java may not be the best choice because it could impact the performance.

- **Memory Management:** Java uses automatic garbage collection which can be a disadvantage in situations where fine-grained control over memory is required. **For example**, in embedded systems or high-performance computing where memory is a limited resource, Java may not be the best choice because it can lead to memory leaks.

- **Security:** Although Java provides a secure sandbox environment, it is still vulnerable to security issues. **For example**, the Java security model can be circumvented if a user runs a malicious applet or application that exploits vulnerabilities in the Java Virtual Machine **(JVM).**

- **Complexity:** Java has a steep learning curve, especially for beginners. **For example**, the syntax of Java can be confusing, and it requires a lot of boilerplate code to accomplish even simple tasks.

- **Compatibility:** Java is not always compatible with older versions of itself. **For example**, if you write code using the latest version of Java, it may not run on older versions of Java without modifications.

- **Overhead:** Java requires a significant amount of memory to run, and it can be resource-intensive. **For example**, if you are running multiple Java applications on a single machine, it can consume a significant amount of resources and slow down the system.

It's important to note that many of these disadvantages can be mitigated through careful design and optimization of your `Java applications`. Additionally, `Java` has many advantages, such as its portability, that may make it a better choice for some projects than other programming languages.

Here are some funny facts about Java:

- Java is known for its slogan **"Write Once, Run Anywhere,"** but some developers joke that it really means "Write Once, Debug Everywhere."
- Java was originally called "Oak" because the developers liked the oak tree outside their office. However, when they found out that name was already trademarked, they had to change it to "Java."
- The first version of Java was released in 1996, and it included a lot of bugs. One of the bugs was so bad that it caused the program to crash if you typed in the word "true" as a variable name.
- Some developers refer to Java as **"Just Another Vague Acronym"** because of the many different interpretations of what the name means.
- Java has been the subject of many jokes and memes in the **programming community**, including a popular meme that shows a picture of a coffee cup with the caption **"I drink Java for breakfast."**
- One of the features of Java is its ability to run code in a sandbox environment, which led to the joke that **"Java is to JavaScript as ham is to hamster."**

While these may be funny, it's important to remember that **Java** is a powerful and widely-used programming language that has many practical applications.

Java is a versatile programming language with many practical applications in a wide range of industries. **Here are some examples of applications of Java:**

- **Web Applications:** Java is used to develop server-side applications and dynamic websites using technologies like `JSP` (JavaServer Pages), Servlets, Spring, and Struts.

- **Mobile Applications:** Java is used to develop Android applications, which are used by billions of users around the world.

- **Desktop Applications:** Java is used to develop desktop applications like Eclipse, NetBeans, and `OpenOffice`.

- **Enterprise Applications:** Java is used to develop large-scale enterprise applications such as banking systems, financial management systems, and inventory management systems.

- **Scientific Applications:** Java is used to develop scientific and mathematical applications, such as those used in data analysis, simulations, and modeling.

- **Gaming:** Java is used to develop games, such as Minecraft, which is one of the most popular video games of all time.

- **Internet of Things (IoT):** Java is used to develop IoT applications and devices, such as smart homes and wearable technology.

- **Big Data:** Java is used to develop big data applications, such as Hadoop and Spark, which are used to process and analyze large amounts of data.

These are just a few examples of the many applications of `Java`. Its flexibility and versatility make it a popular choice for developers across many industries.

> "One of the things that Java is good at is giving you this homogeneous view of a reality that's usually very heterogeneous."
>
> **– James Gosling**

PYTHON – OVERVIEW

Introduced in 1991, **Python** is an interpreted, high-level, all-purpose programming language. It has become one of the most popular programming languages in the world due to its simplicity, versatility, and ease of use. **Here are some details about Python:**

- **Interpreted Language:** Python runs without needing to be compiled because it is an interpreted language. The Python interpreter reads the code and executes it line by line, making it easier to test and debug code.

- **High-Level Language:** Python is a high-level language, which means that it abstracts away much of the underlying machine-level operations, making it easier to read and write code. This also makes Python easier to learn than low-level languages like assembly language.

- **General-Purpose Language:** Python is a general-purpose language, which means that it can be used for a wide range of applications, from web development and data analysis to scientific computing and artificial intelligence.

- **Object-Oriented Language:** Python is an object-oriented language, which means that it uses objects to represent data and behavior. This makes it easier to organize and manage code, and also allows for code reuse.

- **Dynamic Typing:** Python is dynamically typed, which means that the data type of a variable is determined at runtime, rather than being explicitly declared in the code. This allows for more flexible and expressive code, but can also lead to errors if types are not checked carefully.

- **Garbage Collection:** Python uses automatic garbage collection to manage memory, which means that the interpreter automatically frees up memory that is no longer being used by the program.

- **Standard Library:** Python comes with a large standard library that includes modules for tasks such as file I/O, network programming, regular expressions, and more. This makes it easy to perform many common programming tasks without needing to write additional code.

- **Third-Party Libraries:** Python also has a vast ecosystem of third-party libraries, including popular ones like NumPy, Pandas, TensorFlow, and Django, which extend the functionality of Python and allow it to be used for even more applications.

- **Syntax:** Python has a simple and easy-to-read syntax that emphasizes readability and reduces the cost of program maintenance. Indentation is used to indicate block structure, and there are no semicolons or curly braces to clutter the code.
- **Portability:** Python code can be run on many different platforms, including Windows, Mac, Linux, and Unix, making it a highly portable language.

Overall, **Python's simplicity**, **versatility**, and **ease of use** make it a great choice for a wide range of programming tasks.

Here is the **"Hello World"** program in Python:

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

When you run this program, it will output the message "Hello, World!" to the console. This is a simple example of how to use the **print** function in Python to output text.

In Python, you can add comments to your code to make it more readable and to explain what the code is doing. **Comments** are ignored by the Python interpreter, so they do not affect the behavior of your program. **Here are some examples of Python comments:**

Single-line comments:

```python
# This is a single-line comment in Python
```

In this example, the **#** character indicates that the rest of the line is a comment. Anything following the **#** character on the same line will be ignored by the Python interpreter.

Multi-line comments:

```python
"""
This is a multi-line comment in Python.
It can span multiple lines and is enclosed in triple quotes.
"""
```

In this example, the **comment** is enclosed in triple quotes (**"""**). This allows the comment to span multiple lines. However, multi-line comments are not commonly used in Python because single-line comments are usually sufficient. **Comments** can also be used to temporarily disable code, for testing or debugging purposes. **Here's an example:**

```
# This line of code is currently not needed
# print("Hello, World!")
```

In this example, the **print("Hello, World!")** statement is commented out. This means that it will not be executed when the program is run. However, if you later need to re-enable the code, you can simply remove the **#** characters. **Comments** are an important tool for making your code more readable and understandable, both for yourself and for others who may need to read or modify your code. By adding **comments**, you can explain what your code is doing, why you made certain design decisions, and any other important information that will help others understand your code.

In **Python**, variables are used to store data values. They are like containers that hold values that can be accessed and manipulated throughout the program. **Here are some examples and explanations of variables in Python:**

- **Numeric Variables:** Numeric variables are used to store numerical values such as integers or floating-point numbers.

```
# Example of numeric variables

x = 10
y = 3.14
```

In this example, **x** is an integer variable that stores the value 10 and **y** is a floating-point variable that stores the value 3.14.

- **String Variables:** String variables are used to store text values. They are enclosed in either single quotes (`' '`) or double quotes (`""`).

```python
# Example of string variables

name = "John Doe"
message = 'Hello World!'
```

In this example, **name** is a string variable that stores the value "John Doe" and **message** is a string variable that stores the value "Hello World!".

- **Boolean Variables:** Boolean variables are used to store boolean values, which are either True or False.

```python
# Example of boolean variables

is_python_fun = True
is_coding_hard = False
```

In this example, **is_python_fun** is a boolean variable that stores the value True and **is_coding_hard** is a boolean variable that stores the value False.

- **List Variables:** List variables are used to store a collection of values, which can be of different data types. Lists are created by enclosing the values in square brackets `[]` and separating them with commas.

```python
# Example of list variables

fruits = ['apple', 'banana', 'orange']
numbers = [1, 2, 3, 4, 5]
```

In this example, **fruits** is a list variable that stores the values 'apple', 'banana', and 'orange' and **numbers** is a list variable that stores the values 1, 2, 3, 4, and 5.

In Python, a **dictionary** is a collection of key-value pairs. It is a built-in data type that is commonly used to store and manipulate data in a flexible and efficient way. **Here are some examples of dictionary variables in Python:**

Example 1: Creating a Dictionary

```python
person = {"name": "John Doe", "age": 30, "city": "New York"}
print(person)
```

Output:

```
{'name': 'John Doe', 'age': 30, 'city': 'New York'}
```

In this example, we're creating a **dictionary variable** called **person** that contains three key-value pairs. The keys are "name", "age", and "city", and the corresponding values are "John Doe", 30, and "New York".

Example 2: Accessing Values in a Dictionary

```python
person = {"name": "John Doe", "age": 30, "city": "New York"}
print(person["name"])
print(person["age"])
```

Output:

```
John Doe
30
```

In this example, we're accessing the values of the "name" and "age" keys in the person dictionary using square bracket notation. The values are then printed to the console.

Example 3: Updating a Dictionary

```python
person = {"name": "John Doe", "age": 30, "city": "New York"}
person["age"] = 31
print(person)
```

Output:

```
{'name': 'John Doe', 'age': 31, 'city': 'New York'}
```

In this example, we're updating the value of the "age" key in the person dictionary from 30 to 31 using square bracket notation. The updated dictionary is then printed to the console.

Example 4: Adding a Key-Value Pair to a Dictionary

```python
person = {"name": "John Doe", "age": 30, "city": "New York"}
person["state"] = "New York"
print(person)
```

Output:

```
{'name': 'John Doe', 'age': 30, 'city': 'New York', 'state': 'New York'}
```

In this example, we're adding a new key-value pair to the person dictionary using square bracket notation. The new key is "state" and the value is "New York". The updated dictionary is then printed to the console.

Example 5: Deleting a Key-Value Pair from a Dictionary

```python
person = {"name": "John Doe", "age": 30, "city": "New York"}
del person["city"]
print(person)
```

Output:

```
{'name': 'John Doe', 'age': 30}
```

In this example, we're deleting the key-value pair for the "city" key in the person dictionary using the **del** statement. The updated dictionary without the "city" key is then printed to the console. These are just a few examples of variables in Python. **Variables** can hold many different types of values and can be manipulated in various ways throughout a program.

In Python, there are three main types of numerical data: integers, floating-point numbers, and complex numbers.

1. **Integers:** Integers are whole numbers, which means they do not have decimal points. They can be positive, negative, or zero. Integers in Python are of type **'int'**. For example:

```python
x = 5
y = -10
z = 0
```

2. **Floating-point numbers:** Floating-point numbers are decimal numbers, which means they have decimal points. They can be positive, negative, or zero. Floating-point numbers in Python are of type **'float'**. For example:

```
a = 3.14
b = -2.5
c = 0.0
```

Complex numbers: Complex numbers are numbers with both real and imaginary parts. They are expressed in the form a + bj, where a and b are real numbers and j is the imaginary unit **(square root of -1)**. **Complex numbers** in Python are of type 'complex'. For example:

```
d = 2 + 3j
e = -4j
f = 1.5 - 2j
```

Here are some examples of using these types of numerical data in Python:

```python
# Integer arithmetic
x = 5
y = 3
print(x + y)  # Output: 8
print(x - y)  # Output: 2
print(x * y)  # Output: 15
print(x / y)  # Output: 1.6666666666666667
print(x // y)  # Output: 1 (integer division)
print(x % y)  # Output: 2 (modulus or remainder)


# Floating-point arithmetic
a = 3.14
b = 2.0
print(a + b)  # Output: 5.14
print(a - b)  # Output: 1.14
```

```
print(a * b)  # Output: 6.28
print(a / b)  # Output: 1.57


# Complex arithmetic
d = 2 + 3j
e = -4j
print(d + e)  # Output: (2-1j)
print(d - e)  # Output: (2+5j)
print(d * e)  # Output: (-12+8j)
print(d / e)  # Output: (0.75-0.5j)
```

In addition to the **basic arithmetic operations** shown above, **Python** provides many built-in functions for working with numerical data. Here are some commonly used functions:

- **abs():** Returns the absolute value of a number.

```
print(abs(-5))  # Output: 5
print(abs(5))   # Output: 5
```

- **round():** Rounds a number to the nearest integer or to a specified number of decimal places.

```
print(round(3.14159))  # Output: 3
print(round(3.14159, 2))  # Output: 3.14
```

- **pow():** Raises a number to a specified power.

```
print(pow(2, 3))  # Output: 8
print(pow(2, -3))  # Output: 0.125
```

- **min():** Returns the minimum value of a sequence of numbers.

```
print(min(1, 2, 3))  # Output: 1
print(min(-1, -2, -3))  # Output: -3
```

- **max():** Returns the maximum value of a sequence of numbers.

```
print(max(1, 2, 3))  # Output: 3
print(max(-1, -2, -3))  # Output: -1
```

- **sum():** Returns the sum of a sequence of numbers.

```
print(sum([1, 2, 3]))  # Output: 6
```

- **sqrt():** Returns the square root of a number.

```
import math
print(math.sqrt(16))  # Output: 4.0
```

- **floor():** Returns the largest integer less than or equal to a number.

```
import math
print(math.floor(3.7))  # Output: 3
```

- **ceil():** Returns the smallest integer greater than or equal to a number.

```
import math
print(math.ceil(3.7))  # Output: 4
```

- **random():** Generates a random float between 0 and 1.

```
import random
print(random.random())  # Output: a random float between 0 and 1
```

These are just a few examples of the many **built-in functions** available for working with numerical data in **Python**.

In Python, **casting** refers to the process of converting one data type to another. **Here are some examples of casting in Python:**

- **int():** Converts a value to an integer.

```python
x = 5.7
print(int(x))  # Output: 5
```

- **float():** Converts a value to a floating-point number.

```python
x = 5
print(float(x))  # Output: 5.0
```

- **str():** Converts a value to a string.

```python
x = 5
print(str(x))  # Output: "5"
```

- **bool():** Converts a value to a Boolean value.

```python
x = 0
print(bool(x))  # Output: False
```

- **list():** Converts a sequence to a list.

```python
x = (1, 2, 3)
print(list(x))  # Output: [1, 2, 3]
```

- **tuple():** Converts a sequence to a tuple.

```
x = [1, 2, 3]
print(tuple(x))  # Output: (1, 2, 3)
```

- **set():** Converts a sequence to a set.

```
x = [1, 2, 3, 2]

print(set(x))  # Output: {1, 2, 3}
```

- **dict():** Converts a sequence of key-value pairs to a dictionary.

```
x = [("a", 1), ("b", 2), ("c", 3)]
print(dict(x))  # Output: {'a': 1, 'b': 2, 'c': 3}
```

Casting is useful when you need to convert data from one type to another. **For example**, you may need to convert a string input to a numeric type in order to perform mathematical operations on it. **Casting** can also be used to convert data between **Python's built-in data structures**, such as lists and tuples, or to convert data between different types of collections, such as converting a sequence to a set.

In Python, a **string** is a sequence of characters enclosed in either single quotes (' ') or double quotes (" "). Strings are a fundamental data type in **Python** and are widely used in programming. **Here are some examples of strings:**

```
string1 = 'Hello, world!'
string2 = "This is a string."
string3 = "1234"
```

In the examples above, **string1** and **string2** are string literals that contain alphanumeric characters and punctuation. **string3** is a string that contains only digits. **Python strings** support several operations such as indexing, slicing, concatenation, repetition, and more. **Let's discuss some of these operations in more detail below:**

- **Indexing**

Indexing allows you to access individual characters in a string. The index of the first character is 0, and the index of the last character is -1. You can use square brackets to specify the index of the character you want to access.

```python
string = 'Hello, world!'
print(string[0])    # Output: 'H'
print(string[4])    # Output: 'o'
print(string[-1])   # Output: '!'
```

- **Slicing**

Slicing allows you to extract a portion of a string. You can specify a range of indices to extract using the **start:end** syntax. The **start** index is inclusive, and the **end** index is exclusive.

```python
string = 'Hello, world!'
print(string[0:5])    # Output: 'Hello'
print(string[7:])     # Output: 'world!'
```

- **Concatenation**

Concatenation allows you to join two or more strings together using the **+** operator.

```python
string1 = 'Hello, '
string2 = 'world!'
string3 = string1 + string2
print(string3)    # Output: 'Hello, world!'
```

- **Repetition**

Repetition allows you to repeat a string multiple times using the ***** operator.

```python
string = 'Hello, '
print(string * 3)    # Output: 'Hello, Hello, Hello, '
```

- **Length**

You can find the length of a string using the **len()** function.

```python
string = 'Hello, world!'
print(len(string))    # Output: 13
```

- **Formatting**

String formatting allows you to insert values into a string using placeholders. There are several ways to do string formatting in **Python**, but the most common method is to use the **.format()** method.

```python
name = 'Alice'
age = 25
print('My name is {} and I am {} years old.'.format(name, age))
# Output: 'My name is Alice and I am 25 years old.'
```

Overall, **strings** are a powerful and versatile data type in Python that can be used to manipulate and store text data.

Python operators are special symbols that carry out operations on variables and values. **Python** provides various types of operators, including arithmetic operators, comparison operators, logical operators, and assignment operators. **Here's a brief overview of Python operators:**

- **Arithmetic Operators:** These are used to perform mathematical operations, such as addition, subtraction, multiplication, division, modulus, exponentiation, and floor division. **Examples include +, -, *, /, %, **, and //.**
- **Comparison Operators:** These are used to compare two values and return a boolean value (True or False). **Examples include ==, !=, >, <, >=, and <=.**
- **Logical Operators:** These are used to combine or invert boolean expressions. **Examples include and, or, and not.**
- **Assignment Operators:** These are used to assign a value to a variable. **Examples include =, +=, -=, *=, /=, %=, and //=.**
- **Bitwise Operators:** These are used to perform bitwise operations on integers. **Examples include &, |, ^, ~, <<, and >>.**
- **Membership Operators:** These are used to test if a value is a member of a sequence. **Examples include in and not in.**
- **Identity Operators:** These are used to test if two variables point to the same object. **Examples include is and is not.**

You can use these operators in your **Python code** to perform various operations on variables and values. **For example:**

```python
a = 5
b = 2

print(a + b)  # Output: 7
print(a - b)  # Output: 3
print(a * b)  # Output: 10
print(a / b)  # Output: 2.5
print(a % b)  # Output: 1
print(a ** b) # Output: 25
print(a // b) # Output: 2
```

```
a = 5
b = 2

print(a == b)   # Output: False
print(a != b)   # Output: True
print(a > b)    # Output: True
print(a < b)    # Output: False
print(a >= b)   # Output: True
print(a <= b)   # Output: False
```

```
a = 5
b = 2
c = 0

print(a > b and b > c)   # Output: True
print(a < b or b > c)    # Output: True
print(not a == b)        # Output: True
```

```
# Using the `in` operator
fruits = ['apple', 'banana', 'cherry', 'orange']
if 'banana' in fruits:
    print("Yes, banana is in the fruits list.")

# Output: Yes, banana is in the fruits list.

if 'pear' in fruits:
    print("Yes, pear is in the fruits list.")
else:
    print("No, pear is not in the fruits list.")

# Output: No, pear is not in the fruits list.
```

```
# Using the `not in` operator
if 'pineapple' not in fruits:
    print("Yes, pineapple is not in the fruits list.")
else:
    print("No, pineapple is in the fruits list.")


# Output: Yes, pineapple is not in the fruits list.


if 'orange' not in fruits:
    print("Yes, orange is not in the fruits list.")
else:
    print("No, orange is in the fruits list.")


# Output: No, orange is in the fruits list.
```

In the first example, we check if 'banana' is in the **fruits** list using the **in** operator, and it returns True. In the second example, we check if 'pear' is in the **fruits** list using the **in** operator, and it returns False, so we print "No, pear is not in the fruits list." In the third example, we check if 'pineapple' is not in the **fruits** list using the **not in** operator, and it returns True. In the fourth example, we check if 'orange' is not in the **fruits** list using the **not in** operator, and it returns False, so we print "No, orange is in the fruits list."

In Python, a **list** is an ordered collection of items or elements that can be of any data type. Lists are defined using square brackets **[ ]** and each item in the list is separated by a comma. **For example:**

```
my_list = [1, 2, 3, 4, 5]
```

Lists are mutable, which means that their values can be changed after they are created. You can add, remove, or modify items in a list. **Here are some examples of common operations you can perform on lists:**

- **Accessing Elements:** You can access individual elements of a list using their index, which starts at 0. **For example:**

```
print(my_list[0]) # Output: 1
```

- **Slicing:** You can also access a subset of elements in a list using slicing. Slicing uses a colon (:) to separate the start and end indices. **For example:**

```
print(my_list[1:3]) # Output: [2, 3]
```

- **Appending Elements:** You can add new elements to the end of a list using the **append()** method. **For example:**

```
my_list.append(6)

print(my_list) # Output: [1, 2, 3, 4, 5, 6]
```

- **Removing Elements:** You can remove elements from a list using the **remove()** method. **For example:**

```
my_list.remove(3)
print(my_list) # Output: [1, 2, 4, 5, 6]
```

- **Checking if an Element is in a List:** You can check if an element is in a list using the in keyword. For example:

```
print(3 in my_list) # Output: False
```

Lists are a very powerful data structure in **Python** and are used extensively in many different applications.

In Python, a **tuple** is an ordered, immutable collection of elements that can contain any type of data. Tuples are created using parentheses **()** and each element is separated by a comma. **Here is an example of creating a tuple:**

```
my_tuple = (1, 2, 3, 'hello', True)
```

Tuples can also be created without parentheses, by simply separating the elements with commas:

```
my_tuple = 1, 2, 3, 'hello', True
```

Tuples can contain elements of different types, including other tuples:

```
nested_tuple = ((1, 2), ('a', 'b', 'c'), True)
```

Tuples are immutable, which means that once a tuple is created, its elements cannot be changed. However, if a tuple contains mutable objects like lists, the objects inside the tuple can be modified. **Tuples** can be accessed using indexing, which starts from **0**. **Here's an example of accessing the first element of a tuple:**

```
first_element = my_tuple[0]
```

Tuples also support slicing, which allows you to extract a portion of the tuple. **Here's an example of getting the first two elements of a tuple:**

```
first_two_elements = my_tuple[:2]
```

Tuples support several built-in methods like **count()** and **index()**. The **count()** method returns the number of times a specified element appears in the tuple, while the **index()** method returns the index of the first occurrence of a specified element in the tuple.

```python
my_tuple = (1, 2, 3, 'hello', True, 1, 2, 3)
count_of_1 = my_tuple.count(1)  # returns 2
index_of_hello = my_tuple.index('hello')  # returns 3
```

Tuples are often used to group related data together. **For example**, a tuple can be used to represent a point in a 2D coordinate system:

```python
point = (3, 4)
```

Tuples can also be used to return multiple values from a function:

```python
def get_name_and_age():
    name = 'John'
    age = 30
    return name, age


name, age = get_name_and_age()
```

In the above example, the function **get_name_and_age()** returns a tuple containing the name and age. These values are then assigned to the variables name and age using tuple unpacking.

Python sets are a built-in data structure that allows you to store a collection of unique elements. The elements of a set can be of any immutable data type, such as numbers, strings, or **tuples**. The syntax for creating a set is to enclose a comma-separated list of elements in curly braces **({})** or by using the **set()** constructor function. **For example:**

```
# create a set using curly braces
my_set = {1, 2, 3, 4, 5}


# create a set using the set() constructor
my_other_set = set([5, 6, 7, 8, 9])
```

Note that when creating an empty set, you must use the **set()** constructor, as **{}** creates an empty dictionary in Python. **Some of the key characteristics of sets are:**

1. **Sets only store unique elements:** If you try to add an element to a set that already exists in the set, it will be ignored.
2. **Sets are unordered:** Unlike lists and tuples, sets do not maintain any particular order for their elements.
3. **Sets are mutable:** You can add and remove elements from a set after it has been created.

Here are some common operations you can perform with sets in Python:

- **Adding elements to a set:** You can add elements to a set using the **add()** method.

```
my_set = {1, 2, 3}
my_set.add(4)
print(my_set) # outputs {1, 2, 3, 4}
```

- **Removing elements from a set:** You can remove elements from a set using the **remove()** method.

```
my_set = {1, 2, 3, 4}
my_set.remove(4)
print(my_set) # outputs {1, 2, 3}
```

- **Checking if an element is in a set:** You can use the **in** keyword to check if an element is in a set.

```python
my_set = {1, 2, 3}
print(2 in my_set) # outputs True
print(4 in my_set) # outputs False
```

- **Combining sets:** You can combine two sets using the **union()** method or the **|** operator.

```python
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union_set = set1.union(set2)
print(union_set) # outputs {1, 2, 3, 4, 5}

# using the | operator
union_set2 = set1 | set2
print(union_set2) # outputs {1, 2, 3, 4, 5}
```

- **Finding the intersection of sets:** You can find the common elements between two sets using the **intersection()** method or the **&** operator.

```python
set1 = {1, 2, 3}
set2 = {3, 4, 5}
intersection_set = set1.intersection(set2)
print(intersection_set) # outputs {3}

# using the & operator
intersection_set2 = set1 & set2
print(intersection_set2) # outputs {3}
```

- **Finding the difference between sets:** You can find the elements that are in one set but not in the other using the **difference()** method.

```python
set1 = {1, 2, 3}
set2 = {3, 4, 5}

difference_set = set1.difference(set2)
print(difference_set) # outputs {1, 2}
```

In this example, **set1** contains the elements **{1, 2, 3}** and **set2** contains the elements **{3, 4, 5}**. The **difference()** method is called on **set1** with **set2** passed as an argument. This returns a new set containing the elements that are in **set1** but not in **set2**, which in this case is **{1, 2}**. Alternatively, you can use the **-** operator to find the difference between sets:

```python
set1 = {1, 2, 3}
set2 = {3, 4, 5}

difference_set = set1 - set2

print(difference_set) # outputs {1, 2}
```

In this example, the **"-"** operator is used to subtract **set2** from **set1**, which produces the same result as using the **difference()** method.

A **dictionary** in Python is a collection of key-value pairs, where each key is unique and associated with a value. It is also known as an associative array, hash map or a hash table in other programming languages. In Python, **dictionaries** are created using curly braces **{}** or the **dict()** constructor function. The keys and values in a dictionary can be of any data type, including strings, integers, floats, lists, or even other dictionaries. However, keys must be

immutable types such as strings, numbers or tuples. **Here's an example of how to create a dictionary in Python:**

```python
# create a dictionary using {}
my_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}

# create a dictionary using dict()
my_dict = dict(key1='value1', key2='value2', key3='value3')
```

To access a value in a **dictionary**, you can use the key inside square brackets:

```python
# access a value using a key
my_dict['key1']
```

You can also add, remove or modify items in a **dictionary**:

```python
# add a new key-value pair
my_dict['key4'] = 'value4'

# remove a key-value pair
del my_dict['key3']

# modify a value
my_dict['key2'] = 'new_value2'
```

Dictionaries also have several built-in methods to perform common operations, such as getting a list of keys or values:

```
# get a list of keys
my_dict.keys()


# get a list of values
my_dict.values()
```

Overall, **dictionaries** are a powerful data structure in Python that allow you to store and manipulate data in a flexible and efficient way.

In Python, **if...else** is a conditional statement that allows you to execute certain blocks of code based on whether a condition is true or false. The basic syntax of an **if...else** statement in Python is:

```
if condition:
    # code to execute if condition is True
else:
    # code to execute if condition is False
```

The **condition** is a Boolean expression that evaluates to either **True** or **False**. If the **condition** is **True**, the code block inside the **if** statement is executed, and if the **condition** is **False**, the code block inside the **else** statement is executed. **Here's an example:**

```
age = 18


if age >= 18:
    print("You are an adult.")
else:
    print("You are not an adult yet.")
```

In this example, if the value of **age** is greater than or equal to **18**, the message "You are an adult." will be printed to the console. Otherwise, the message "You are not an adult yet." will be printed. You can also add more conditions to your **if...else** statement using **elif** (short for "else if"):

```python
age = 18

if age < 18:
    print("You are not an adult yet.")
elif age < 30:
    print("You are a young adult.")
else:
    print("You are an adult.")
```

In this example, if the value of **age** is less than **18**, the message "You are not an adult yet." will be printed. If **age** is greater than or equal to **18** but less than **30**, the message "You are a young adult." will be printed. And if **age** is greater than or equal to **30**, the message "You are an adult." will be printed. **if...else** statements can also be nested inside other **if...else** statements to create more complex conditions. However, it's important to keep the code readable and not to create excessively nested statements. Overall, **if...else** statements are a fundamental control structure in **Python** that allow you to make decisions based on conditions and execute code accordingly.

In **Python**, **loops** are used to execute a block of code repeatedly, based on a specific condition or range of values. There are two main types of loops in Python: for loops and while loops.

- **For Loops**

    A **for** loop is used to iterate over a sequence of values (such as a list, tuple or string) or a range of numbers. The basic syntax of a **for** loop in Python is:

```
for variable in sequence:
    # code to execute
```

Here's an example of a **for** loop that iterates over a list of names and prints each name to the console:

```python
names = ['Alice', 'Bob', 'Charlie']

for name in names:
    print(name)
```

This loop will print:

```
Alice
Bob
Charlie
```

You can also use the **range()** function to generate a sequence of numbers to iterate over:

```python
for i in range(5):
    print(i)
```

This loop will print:

```
0
1
2
3
4
```

- **While Loops**

  A **while** loop is used to execute a block of code repeatedly as long as a condition is true. The basic syntax of a **while** loop in Python is:

  ```python
  while condition:
      # code to execute
  ```

Here's an example of a **while** loop that counts from 0 to 4 and prints each number to the console:

```python
i = 0

while i < 5:
    print(i)
    i += 1
```

This loop will print:

```
0
1
2
3
4
```

You need to be careful with **while** loops, because if the condition is never met, the loop will continue to run indefinitely, causing what's known as an **infinite loop**. You can use **break** and **continue** statements to control the flow of the loop and exit early if necessary.

Loop Control Statements

In addition to break and continue, there are other loop control statements you can use in Python:

- **break**: exits the loop immediately, skipping any remaining iterations.
- **continue**: skips the current iteration and goes to the next one.
- **pass**: does nothing, used as a placeholder when a statement is required but you don't want to execute any code.

Here's an example of a **for** loop that uses **break** and **continue**:

```python
for i in range(10):
    if i == 5:
        break
    elif i % 2 == 0:
        continue
    else:
        print(i)
```

This loop will print:

```
1
3
7
9
```

The **loop** starts at **0** and goes up to **9**. When **i** is equal to **5**, the **break** statement is executed, and the loop exits early. When **i** is even, the **continue** statement is executed, and the loop skips the rest of the code and goes to the next iteration. Otherwise, the loop prints the value of **i**. Overall, **loops** are a fundamental concept in programming that allows you to execute code repeatedly and perform complex operations on large datasets.

A **function** in Python is a self-contained block of code that performs a specific task. It takes input(s) (if any), processes the input(s), and produces output(s) (if any). In **Python**, a function is defined using the **def** keyword followed by the function name, input parameters (if any), and a colon (:). The body of the function is indented and contains the code to be executed when the function is called. **Here is a simple example of a function that takes two input parameters, adds them together, and returns the result:**

```python
def add_numbers(x, y):
    result = x + y
    return result
```

In this example, **add_numbers** is the function name, **x** and **y** are the input parameters, **result** is a variable that holds the sum of **x** and **y**, and **return result** specifies the output of the function. To call this function, you simply provide values for **x** and **y** in the parentheses:

```python
sum = add_numbers(5, 3)
print(sum)  # Output: 8
```

This will call the **add_numbers** function with **x = 5** and **y = 3**, and assign the result **(8)** to the variable **sum**. The **print** statement will then output the value of **sum**. **Functions** can also have optional parameters, default values, and can return **multiple values** using **tuples**. They can also be used to encapsulate complex operations and simplify the code by breaking it down into smaller, reusable components.

A **lambda** function in Python is a small anonymous function that can have any number of parameters, but can only have one expression. The syntax for creating a lambda function is to use the lambda keyword followed by the parameters (if any) separated by commas, followed by a

colon, and then the expression. **Here is a simple example of a lambda function that takes two parameters and returns their sum:**

```
sum = lambda x, y: x + y
```

In this example, **lambda** is the keyword used to define the function, **x** and **y** are the parameters, and **x + y** is the expression that is evaluated when the function is called. **Lambda functions** are often used as a way to define small, one-off functions that can be passed as arguments to other functions. **For example**, you can use a lambda function to define the key parameter for sorting a list:

```
numbers = [5, 1, 3, 6, 2, 8, 4]
sorted_numbers = sorted(numbers, key=lambda x: x)
print(sorted_numbers)  # Output: [1, 2, 3, 4, 5, 6, 8]
```

In this example, the **sorted** function is called with **numbers** as the first argument and a lambda function as the second argument. The lambda function takes a single parameter **x** and returns **x**, which is used as the sorting key. **Lambda functions** are also used in functional programming paradigms, where they can be used to define higher-order functions that take other functions as parameters or return functions as values.

In Python, a **class** is a blueprint for creating objects. An **object** is an instance of a class that contains data (attributes) and functions (methods). **Here is an example of a class in Python:**

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

```python
    def greet(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")
```

In this example, **Person** is the name of the class. The **__init__** method is a special method called the constructor, which is called when an object is created. The constructor takes two parameters, **name** and **age**, which are used to initialize the **name** and **age** attributes of the object. The **greet** method is a function that takes no parameters and prints a message that includes the **name** and **age** attributes of the object. To create an object of the **Person** class, you would use the following syntax:

```python
person = Person("Alice", 25)
```

This creates a new object of the **Person** class with the **name** attribute set to "Alice" and the **age** attribute set to 25. You can access the attributes and methods of an object using the **dot (.)** operator. **For example:**

```python
print(person.name)    # Output: "Alice"
person.greet()        # Output: "Hello, my name is Alice and I am 25 years old."
```

In this example, the **name** attribute of the **person** object is accessed using the dot operator, and the **greet** method is called on the **person** object. **Classes** and objects are a fundamental concept in **object-oriented programming** (OOP), which is a programming paradigm that emphasizes the use of classes and objects to model real-world entities and their relationships. **OOP** provides a way to encapsulate data and behavior into reusable and modular components, which makes it easier to write and maintain complex software systems.

In Python, an **iterator** is an object that can be iterated (looped) upon. An iterator is an implementation of the iterator protocol, which requires the iterator to have two methods: **__iter__()** and **__next__()**. The **__iter__()** method returns the iterator object itself, and the **__next__()** method returns the next value in the iteration sequence. When there are no more items to return, the **__next__()** method should raise the **StopIteration** exception. **Here is an example of an iterator in Python:**

```python
class MyIterator:
    def __init__(self):
        self.counter = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.counter < 5:
            self.counter += 1
            return self.counter
        else:
            raise StopIteration
```

In this example, **MyIterator** is a class that implements an iterator. The **__init__()** method initializes the **counter** attribute of the iterator to 0. The **__iter__()** method returns the iterator object itself, and the **__next__()** method returns the next value in the iteration sequence. In this example, the iteration sequence consists of the numbers from 1 to 5. The **__next__()** method checks if the **counter** attribute is less than 5. If it is, it increments the **counter** attribute and returns its value. If the counter attribute is equal to or greater than 5, it raises the **StopIteration** exception to indicate that there are no more items in the iteration sequence. To use an iterator, you can use a **for** loop or the built-in **next()** function. **For example:**

```
my_iterator = MyIterator()

# Using a for loop
for num in my_iterator:
    print(num)

# Using the next() function
print(next(my_iterator))  # Output: 1
print(next(my_iterator))  # Output: 2
print(next(my_iterator))  # Output: 3
print(next(my_iterator))  # Output: 4
print(next(my_iterator))  # Output: 5
print(next(my_iterator))  # Raises StopIteration exception
```

In this example, the **MyIterator** class is instantiated as **my_iterator**. The **for** loop and **next()** function are used to iterate over the values returned by the **MyIterator** object. Iterators are used extensively in Python, especially with built-in functions such as **range()**, **map()**, and **filter()**.

In Python, **JSON** (JavaScript Object Notation) is a popular data interchange format that is used to represent data structures as text. **JSON** is a lightweight format that is easy for humans to read and write, and easy for machines to parse and generate. The **Python standard library** provides two modules for working with **JSON**:

- **json:** This module provides methods for encoding Python objects as **JSON** strings and decoding **JSON** strings into Python objects.
- **simplejson:** This is a third-party module that provides a more feature-rich **JSON** implementation than the built-in **json** module.

Here is an example of how to use the **json** module to encode and decode **JSON** data in Python:

```python
import json

# Encoding Python data as a JSON string
data = {'name': 'Alice', 'age': 25}
json_data = json.dumps(data)
print(json_data)  # Output: {"name": "Alice", "age": 25}

# Decoding JSON data into a Python object
json_data = '{"name": "Alice", "age": 25}'
data = json.loads(json_data)
print(data)  # Output: {'name': 'Alice', 'age': 25}
```

In this example, the **json.dumps()** method is used to encode a Python dictionary as a **JSON** string, and the **json.loads()** method is used to decode a **JSON** string into a Python dictionary. **JSON** supports a limited set of data types, including strings, numbers, boolean values, null, arrays, and objects. When encoding Python objects as **JSON** data, the **json** module automatically converts compatible data types to their **JSON** equivalents. Here is an example of encoding a Python object with the **json** module:

```python
import json

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

person = Person("Alice", 25)
json_data = json.dumps(person.__dict__)
print(json_data)  # Output: {"name": "Alice", "age": 25}
```

In this example, a custom Python object of the **Person** class is created and encoded as a JSON string using the **json.dumps()** method. The **__dict__** attribute is used to convert the object's attributes to a dictionary that can be encoded as JSON. **JSON** is commonly used in web development for exchanging data between client and server, as well as in data storage and transfer applications.

In Python, **try** and **except** are used for error handling. The basic idea is to try a block of code that may raise an exception (i.e., an error), and then handle that exception if it occurs. **Here's the basic syntax:**

```
try:
    # code that may raise an exception
except ExceptionType:
    # code to handle the exception
```

The **try** block contains the code that you want to execute, and the **except** block contains the code that should be executed if an exception of type **ExceptionType** occurs. If an exception occurs in the **try** block, Python will jump to the **except** block and execute its code. You can use multiple **except** blocks to handle different types of exceptions. **For example:**

```
try:
    # code that may raise an exception
except TypeError:
    # code to handle a TypeError exception
except ValueError:
    # code to handle a ValueError exception
```

If the code in the **try** block raises a **TypeError** exception, the first **except** block will be executed. If it raises a **ValueError** exception, the second **except** block will be executed. You can also include a **finally** block, which will be executed regardless of whether an exception occurred or not. **For example:**

```python
try:
    # code that may raise an exception
except ExceptionType:
    # code to handle the exception
finally:
    # code that should be executed regardless of whether an exception occurred or not
```

The **finally** block will be executed after the **try** and **except** blocks, regardless of whether an exception occurred or not. This is useful for cleaning up resources or closing files, **for example**. Overall, **try** and **except** are an important part of Python error handling, allowing you to gracefully handle errors and prevent your code from crashing.

In Python, **pip** is a package manager used to install and manage software packages written in Python. It makes it easy to install and uninstall **Python packages**, as well as manage dependencies between packages. **pip** is included with most Python installations by default, and can be accessed from the command line by typing **pip** followed by a command. Here are some common **pip** commands:

- **pip install package_name:** installs a package from the Python Package Index (PyPI) or another package repository.
- **pip uninstall package_name:** uninstalls a package.
- **pip list:** lists all installed packages.
- **pip freeze:** lists all installed packages in a format that can be used to create a requirements.txt file, which is commonly used to specify the dependencies of a Python project.

- **pip search package_name:** searches for a package on PyPI or another package repository.

You can also use **pip** to install packages from a requirements.txt file using the following command:

```
pip install -r requirements.txt
```

This will install all the packages listed in the requirements.txt file. In addition to the standard **pip** command, there are several third-party tools and libraries that extend or enhance its functionality, such as **virtualenv** and **conda**. These tools allow you to create isolated Python environments with their own package installations, making it easy to manage dependencies between projects and avoid conflicts between different packages.

In Python, **file handling** refers to the process of working with files on the file system, such as reading from or writing to a file. Python provides a built-in **open()** function to work with files. **Here's how to open a file for reading:**

```
f = open("filename.txt", "r")
```

This opens the file **filename.txt** in read mode. The **open()** function returns a file object, which you can use to read data from the file. To read the entire contents of the file, you can use the **read()** method:

```
data = f.read()
```

After you're done reading from the file, you should close it using the **close()** method:

```
f.close()
```

Here's how to open a file for writing:

```python
f = open("filename.txt", "w")
```

This opens the file **filename.txt** in write mode. To write data to the file, you can use the **write()** method:

```python
f.write("Hello, world!")
```

After you're done writing to the file, you should close it using the **close()** method:

```python
f.close()
```

It's a good practice to always close files after you're done working with them. Alternatively, you can use a **"with"** statement to automatically close the file when you're done with it:

```python
with open("filename.txt", "r") as f:
    data = f.read()
```

This automatically closes the file after the **"with"** block is finished, even if an error occurs. In addition to reading and writing files, you can also append to an existing file using the **"a"** mode, or read and write to a file simultaneously using the **"r+"** or **"w+"** modes. For more information, check out the **Python documentation** on file handling:

```
https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files
```

Python is a popular programming language for data analysis and manipulation tasks. It provides a variety of libraries and frameworks that can be used to work with different databases, including **MySQL**. **MySQL** is an open-source relational database management system that is widely used in web applications. It uses **SQL (Structured Query Language)** to interact with the database

and perform various operations like insert, update, delete, and retrieve data. To work with **MySQL** in Python, you can use a Python MySQL connector library like **mysql-connector-python**, **PyMySQL**, or **MySQLdb**. These libraries provide a set of methods and classes to connect to a **MySQL** database, execute **SQL** queries, and fetch data from the database. Here is a brief overview of how to work with MySQL in Python:

- **Install the MySQL connector library of your choice using pip:**

```
pip install mysql-connector-python
```

- **Connect to the MySQL database using the** connect() **method:**

```python
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="username",
    password="password",
    database="database_name"
)
```

- **Create a cursor object to execute SQL queries:**

```python
mycursor = mydb.cursor()
```

- **Execute a SQL query using the** execute() **method:**

```python
mycursor.execute("SELECT * FROM customers")
```

- **Fetch the data using the** fetchall() **or** fetchone() **method:**

```
result = mycursor.fetchall()
```

- **Iterate through the result set and print the data:**

```
for row in result:
    print(row)
```

You can also perform other operations like **insert, update, and delete data** using `SQL` queries in Python. Overall, **Python** provides an easy and flexible way to work with `MySQL` databases, making it a popular choice for data analysis and web development tasks.

Python has become the go-to language for many data scientists due to its simplicity, versatility, and powerful libraries and frameworks. **Python** provides a wide range of libraries and frameworks for data analysis and manipulation, including NumPy, Pandas, Matplotlib, Scikit-learn, TensorFlow, Keras, and many more. These libraries and frameworks provide various functionalities such as data manipulation, visualization, statistical analysis, machine learning, and deep learning.

- `NumPy` is a library for numerical computing in Python. It provides a multidimensional array object, which allows efficient computation on large datasets. `NumPy` also provides various mathematical functions and operations for array manipulation.
- `Pandas` is a library for data manipulation and analysis in Python. It provides data structures for **handling tabular data**, including `DataFrame` and `Series objects`. `Pandas` allow easy manipulation of data, including data cleaning, merging, grouping, and filtering.
- **Matplotlib** is a library for data visualization in Python. It provides various plotting functions to create different types of plots, including line plots, scatter plots, bar plots, histograms, and more.
- **Scikit-learn** is a library for machine learning in Python. It provides various algorithms for classification, regression, clustering, and dimensionality reduction. Scikit-learn also provide tools for model selection, evaluation, and preprocessing.
- **TensorFlow** is an open-source library for machine learning developed by `Google`. It provides a platform for building and training machine learning models, including deep learning models.

TensorFlow supports various neural network architectures and allows efficient computation on GPUs.

- **Keras** is a high-level neural networks API built on top of `TensorFlow`. It provides a simplified interface for building and training deep learning models, making it easy for beginners to get started with deep learning.

Overall, `Python` provides a comprehensive set of libraries and tools for data science and analytics, making it a popular choice for data scientists and analysts. With its simplicity, versatility, and powerful libraries and frameworks, Python has become the go-to language for data science and analytics tasks.

Advantages of Python:

1. **Easy to Learn and Use:** Python has a simple and easy-to-read syntax, making it easy for beginners to learn and write code. It has a large standard library and many third-party libraries that can be easily installed and used in Python programs. **Example:** Here's a simple Python program that prints **"Hello, World!"** to the console:

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

2. **Versatility:** Python is a versatile language that can be used for a wide range of tasks, including web development, data analysis, machine learning, and more. **Example:** Here's a simple Python program that calculates the sum of two numbers:

```
a = 2
b = 3
c = a + b
print(c)
```

3. **Large Community:** Python has a large and active community of developers, which means there are plenty of resources, tutorials, and documentation available online. **Example:** The Python community has developed many useful libraries and frameworks

for various tasks, such as **NumPy** for numerical computing, **Pandas** for data analysis, and **Flask** for web development.

4. **Cross-platform:** Python is a cross-platform language, which means it can run on different operating systems like Windows, Mac, and Linux. **Example:** Here's a simple Python program that prints the current time on different platforms:

```python
import datetime

current_time = datetime.datetime.now()
print(current_time)
```

Disadvantages of Python:

1. **Speed:** Python is an interpreted language, which means it can be slower than compiled languages like **C++** or **Java**. **Example:** If you have a Python program that performs complex mathematical operations on large datasets, it may take longer to execute than the same program written in a compiled language.

2. **Weak in Mobile Computing:** Python is not well-suited for mobile computing, as it requires a large runtime environment and may not be optimized for mobile devices. **Example:** If you want to develop a mobile app that requires high performance and low memory usage, you may want to consider using a different language like **Swift** or **Kotlin**.

3. **Not Ideal for Memory-intensive Tasks:** Python is not ideal for memory-intensive tasks, as it has limitations in managing memory efficiently. **Example:** If you want to develop a program that requires large amounts of memory, like a video game or a scientific simulation, you may want to consider using a different language like **C++** or **Fortran**.

4. **Not Ideal for Real-time Applications:** Python is not ideal for real-time applications, as it has limitations in handling time-critical tasks. **Example:** If you want to develop a program that requires real-time performance, like a trading system or a control system, you may want to consider using a different language like **C** or **Rust**.

Here is a brief comparison of C, C++, Java, and Python:

C:

- A low-level programming language used for system programming, embedded systems, and operating systems.
- Provides direct memory access and low-level control over hardware resources.
- Does not support object-oriented programming (OOP) and automatic memory management.

C++:

- An extension of C that adds OOP features.
- Used for developing applications and software that require high performance and low-level hardware access.
- Provides access to memory management and supports encapsulation, polymorphism, and inheritance.

Java:

- A high-level programming language designed for cross-platform applications.
- Runs on the Java Virtual Machine (JVM) and provides automatic memory management through garbage collection.
- Supports OOP features like encapsulation, inheritance, and polymorphism.
- Popular for developing enterprise applications, web applications, and Android mobile applications.

Python:

- A high-level programming language known for its simplicity and readability.
- Provides dynamic typing, automatic memory management, and a large library of modules.
- Supports OOP features and is commonly used for scripting, data analysis, web development, and machine learning.

In terms of performance, **C and C++** are generally faster than **Java and Python**, but they require more manual memory management and may have longer development times. **Java and Python** are generally easier to learn and use, and are popular for their cross-platform capabilities and large communities of developers. Ultimately, the **choice of programming language** depends on the specific project requirements, development resources, and personal preferences of the developers involved.

Best Linux Books that Every Superuser Should Read:

- **How Linux Works: What Every Superuser Should Know**

  Book by Brian Ward

- **The Linux Programming Interface**

  Book by Michael Kerrisk

- **Linux pocket guide**

  Book by Daniel J. Barrett

- **Linux for Beginners**

  Book by Jason Cannon

- **How Linux Works: What Every Superuser Should Know**

  Book by Brian Ward

- **Linux Kernel Development**

  Book by Robert Love

- **Linux: The Complete Reference**

  Book by Richard Petersen

- **Linux in a Nutshell**

  Book by Ellen Siever and Robert Love

- **Linux Basics for Hackers: Getting Started with Networking, Scripting, and Security in Kali**

  Book by OccupyTheWeb

- **Linux Command Line and Shell Scripting Bible**

  Book by Christine Bresnahan and Richard BLUM

- **Linux Administration: The Linux Operating System and Command Line Guide for Linux Administrators**

  Book by Jason Cannon

- **The Art of Unix Programming**

  Book by Eric S. Raymond

- **The Linux Command Line, 2nd Edition: A Complete Introduction**

  Book by William Shotts

- **Linux Bible**

  Book by Christopher Negus

- **Linux System Programming: Talking Directly to the Kernel and C Library**

  Book by Robert Love

- **A Practical Guide to Linux Commands, Editors, and Shell Programming**

  Book by Mark G. Sobell

- **Linux for Beginners and Command Line Kung Fu**

  Book by Jason Cannon

- **Linux Device Drivers**

  Book by Alessandro Rubini, Greg Kroah-Hartman, and Jonathan Corbet

- **Advanced Linux programming**

  Book by Alex Samuel, Jeffrey Oldham, and Mark Mitchell

- **Understanding the Linux Kernel**

Book by Daniel Pierre Bovet and Marco Cesati

- **Learn Linux Quickly: A Beginner-friendly Guide to Getting Up and Running with the World's Most Powerful Operating System**

  Book by Ahmed Alkabary

- **Linux administration**

  Book by Wale Soyinka

- **Linux For Dummies**

  Book by Richard Blum

- **Linux Essentials**

  Book by Christine Bresnahan and Richard BLUM

- **The Linux Command Line Beginner's Guide**

  Book by Jonathan Moeller

- **Linux All-in-One for Dummies**

  Book by Emmett Dulaney

- **Learning the bash Shell**

  Book by Cameron Newham

- **Linux for Developers: Jumpstart Your Linux Programming Skills**

  Book by William "Bo" Rothwell

- **Lfm: Linux Field Manual**

  Book by Tim Bryant

- **CompTIA Linux+ Study Guide: Exam XK0-005**

  Book by Christine Bresnahan and Richard BLUM

- **sed & awk**

  Book by Arnold Robbins and Dale Dougherty

- **Linux From Scratch**

  Book by Gerard Beekmans

Linux is a complex example of the wisdom of crowds. It's a good example in the sense that it shows you can set people to work in a decentralized way - that is, without anyone really directing their efforts in a **particular direction** - and still trust that they're going to come up with good answers.

– James Surowiecki

Best Programming Books that Every Programmer Should Read:

C:

- **The C Programming Language**

  Book by Brian Kernighan and Dennis Ritchie

- **C Programming Absolute Beginner's Guide**

  Book by Dean Miller and Greg Perry

- **Head First C**

  Book by David Griffiths and Dawn Griffiths

- **Expert C Programming**

  Book by Peter van der Linden

- **C Programming: A Modern Approach**

  Book by K. N King

- **C: The complete reference**

  Book by Herbert Schildt

- **Learn C the Hard Way: Practical Exercises on the Computational Subjects You Keep Avoiding (Like C)**

  Book by Zed Shaw

- **C in a Nutshell: The Definitive Reference**

  Book by Peter Prinz and Tony Crawford

- **C Programming In Easy Steps**

  Book by Mike McGrath

- **Computer Fundamentals and Programming in C**

  Book by Reema Thareja

- **Hands-On Network Programming with C: Learn Socket Programming in C and Write Secure and Optimized Network Code**

  Book by Lewis Van Winkle

- **Let Us C**

  Book by Yashavant Kanetkar

- **Low-Level Programming: C, Assembly, and Program Execution on Intel® 64 Architecture**

  Book by Igor Zhirkov

- **Effective C: An Introduction to Professional C Programming**

  Book by Robert C. Seacord

- **Data Structures Using C**

  Book by Reema Thareja

- **A Book on C: Programming in C**

  Book by Al Kelley and Ira Pohl

- **C Traps and Pitfalls**

  Book by Andrew Koenig

- **C Interfaces and Implementations: Techniques for Creating Reusable Software**

  Book by David Hanson and David R. Hanson

- **Mastering algorithms with C**

  Book by Kyle Loudon

- **Extreme C: Taking You to the Limit in Concurrency, OOP, and the Most Advanced Capabilities of C**

Book by Kamran Amini

- **Let Us C Solutions**

  Book by Yashavant Kanetkar

- **Bare Metal C: Embedded Programming for the Real World**

  Book by Stephen Oualline

- **Introduction to C Programming**

  Book by Reema Thareja

- **C - In Depth**

  Book by Deepali Srivastava

- **C How to Program**

  Book by Harvey Deitel and Paul Deitel

- **The C answer book**

  Book by Clovis L. Tondo

- **C Programming For Dummies**

  Book by Dan Gookin

- **Understanding Pointers In C & C++: Fully Working Examples and Applications of Pointers**

  Book by Yashavant Kanetkar

C++:

- **The C++ Programming Language**

  Book by Bjarne Stroustrup

- **Effective Modern C++**

Book by Scott Meyers

- **C++ Primer**

  Book by Barbara E. Moo, Josée Lajoie, and Stanley B. Lippman

- **A Tour of C++**

  Book by Bjarne Stroustrup

- **Programming: Principles and Practice Using C++**

  Book by Bjarne Stroustrup

- **C++ Concurrency in Action**

  Book by Anthony Williams

- **Modern C++ Design**

  Book by Andrei Alexandrescu

- **More Effective C++: 35 New Ways To Improve Your Programs And Designs**

  Book by Scott Meyers

- **Accelerated C++: Practical Programming by Example**

  Book by Andrew Koenig and Barbara E. Moo

- **C++ Templates: The Complete Guide**

  Book by David Vandevoorde, Douglas Gregor, and Nicolai M. Josuttis

- **More Exceptional C++: 40 New Engineering Puzzles, Programming Problems, and Solutions**

  Book by Herb Sutter

- **C++ coding standards**

  Book by Andrei Alexandrescu and Herb Sutter

- **C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond**

  Book by Aleksey Gurtovoy and David Abrahams

- **Beginning C++ Game Programming**

  Book by Michael Dawson

- **Beginning C++ Programming**

  Book by Richard Grimes

- **C++ Programming**

  Book by D. S. Malik

- **Sams Teach Yourself C++ in One Hour a Day**

  Book by Rao Siddhartha

- **Professional C++**

  Book by Nicholas A. Solter and Scott J. Kleper

- **Starting Out with C++**

  Book by Tony Gaddis

- **C++: The Complete Reference**

  Book by Herbert Schildt

- **Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions**

  Book by Herb Sutter

- **C++ High Performance: Master the Art of Optimizing the Functioning of Your C++ Code**

  Book by Björn Andrist and Viktor Sehr

- **The C++ Standard Library: A Tutorial and Reference**

  Book by Nicolai M. Josuttis

- **Effective C++ Digital Collection: 140 Ways to Improve Your Programming**

  Book by Scott Meyers

- **C++17 - The Complete Guide**

  Book by Nicolai M. Josuttis

- **C++ Primer Plus**

  Book by Stephen Prata

- **Data Parallel C++: Mastering DPC++ for Programming of Heterogeneous Systems using C++ and SYCL**

  Book by James Brodman, Michael Kinsner, Xinmin Tian, Ben Ashbaugh, John Pennycook, James Reinders

- **Data Structures Using C++**

  Book by Varsha H. Patil

- **The Design and Evolution of C++**

  Book by Bjarne Stroustrup

- **Modern CMake for C++: Discover a Better Approach to Building, Testing, and Packaging Your Software**

  Book by Rafał Świdziński

- **Expert C++: Become a Proficient Programmer by Learning Coding Best Practices with C++17 and C++20's Latest Features**

  Book by Shunguang Wu and Vardan Grigoryan

- **Learn C++ Quickly: A Complete Beginner's Guide to Learning C++, Even If You're New to Programming**

  Book by Code Quickly

- **Memory Management Algorithms and Implementation in C/C++**

  Book by Bill Blunden

- **Thinking in C++**

  Book by Bruce Eckel

- **Modern C++ Programming Cookbook**

  Book by Marius Bancila

- **Hands-On Design Patterns with C++: Solve Common C++ Problems with Modern Design Patterns and Build Robust Applications**

  Book by Fedor G. Pikus

- **A complete guide to programming in C++**

  Book by Ulla Kirch-Prinz

- **C++ Crash Course: A Fast-Paced Introduction**

  Book by Josh Lospinoso

- **C++ For Dummies**

  Book by Stephen Randy Davis

JAVA:

- **Head First Java**

  Book by Bert Bates and Kathy Sierra

- **Effective Java**

  Book by Joshua Bloch

- **Thinking in Java**

Book by Bruce Eckel

- **Java Concurrency in Practice**

  Book by Brian Goetz

- **Core Java**

  Book by Cay S. Horstmann and Gary Cornell

- **Java: A Beginner's Guide**

  Book by Herbert Schildt

- **Java: The Complete Reference**

  Book by Herbert Schildt

- **Java 8 in Action: Lambdas, streams, and functional-style programming**

  Book by Alan Mycroft and Mario Fusco

- **Beginning Programming With Java for Dummies**

  Book by Barry A. Burd

- **Learn Java in One Day and Learn It Well**

  Book by Jamie Chan

- **Modern Java in Action: Lambdas, Streams, Functional and Reactive Programming**

  Book by Alan Mycroft, Mario Fusco, and Raoul-Gabriel Urma

- **Test Driven: Practical TDD and Acceptance TDD for Java Developers**

  Book by Lasse Koskela

- **Java Puzzlers: Traps, Pitfalls, and Corner Cases**

  Book by Joshua Bloch and Neal Gafter

- **Spring in Action**

  Book by Craig Walls and Ryan Breidenbach

- **Java generics and collections**

  Book by Maurice Naftalin

- **Core Java Volume I−Fundamentals**

  Book by Cay S. Horstmann

- **Think Java: How to think like a computer scientist**

  Book by Allen B. Downey

- **Java SE 8 for the Really Impatient**

  Book by Cay S. Horstmann

- **Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D**

  Book by Brett McLaughlin

- **Learning Java: An Introduction to Real-World Programming with Java**

  Book by Daniel Leuck, Marc Loy, and Patrick Niemeyer

- **Core Java for the Impatient**

  Book by Cay S. Horstmann

- **Java By Comparison: Become a Java Craftsman in 70 Examples**

  Book by Jorg Lenhard, Linus Dietz, and Simon Harrer

- **High-Performance Java Persistence**

  Book by Vlad Mihalcea

- **Java in a Nutshell: A Desktop Quick Reference**

  Book by Benjamin Evans and David Flanagan

- **Spring Microservices in Action**

  Book by Edward John Carnell

- **Optimizing Java: Practical Techniques for Improving JVM Application Performance**

  Book by Benjamin Evans, Chris Newland, and James Gough

- **Java Performance: The Definitive Guide: Getting the Most Out of Your Code**

  Book by Scott Oaks

- **Microservices Patterns: With Examples in Java**

  Book by Chris Richardson

- **Functional Programming in Java: Harnessing the Power of Java 8 Lambda Expressions**

  Book by Venkat Subramaniam

- **Mastering Java Machine Learning**

  Book by Krishna Choppella and Uday Kamath

- **OCA: Oracle Certified Associate Java SE 8 Programmer I Study Guide: Exam 1Z0-808**

  Book by Jeanne Boyarsky and Scott Selikoff

- **Let Us Java: Strong Foundation For Java Programming**

  Book by Yashavant Kanetkar

- **Learn Java: A Crash Course Guide to Learn Java in 1 Week**

  Book by Timothy C. Needham

- **Java Performance**

  Book by Binu John and Charlie Hunt

- **The Java Language Specification**

  Book by Oracle Corporation

- **Elements of Programming Interviews in Java: The Insider's Guide**

  Book by Adnan Aziz, Amit Prakash, and Tsung-Hsien Lee

PYTHON:

- **Fluent Python**

  Book by Luciano Ramalho

- **Learn Python the Hard Way: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code**

  Book by Zed Shaw

- **Python Cookbook: Recipes for Mastering Python 3**

  Book by Brian K. Jones and David M. Beazley

- **Automate the Boring Stuff with Python: Practical Programming for Total Beginners**

  Book by Al Sweigart

- **Python Crash Course: A Hands-On, Project-Based Introduction to Programming**

  Book by Eric Matthes

- **Head First Python**

  Book by Paul Barry

- **Programming Python**

  Book by Mark Lutz

- **Think Python: An Introduction to Software Design**

  Book by Allen B. Downey

- **Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython**

  Book by Wes McKinney

- **Python Tricks: A Buffet of Awesome Python Features**

  Book by Dan Bader

- **Learning Python**

  Book by Mark Lutz

- **Introduction to Machine Learning with Python: A Guide for Data Scientists**

  Book by Andreas C. Müller and Sarah Guido

- **Learning Python: Powerful Object-Oriented Programming**

  Book by Mark Lutz

- **Python for Everybody: Exploring Data Using Python 3**

  Book by Charles Severance

- **Python for Kids: A Playful Introduction To Programming**

  Book by Jason R. Briggs

- **Python Programming: An Introduction to Computer Science**

  Book by John M. Zelle

- **Learn Python in One Day and Learn It Well: Python for Beginners with Hands-on Project**

  Book by Jamie Chan

- **Invent Your Own Computer Games with Python**

  Book by Al Sweigart

- **Python Data Science Handbook: Essential Tools for Working with Data**

  Book by Jake VanderPlas

- **Effective Python: 90 Specific Ways to Write Better Python**

  Book by Brett Slatkin

- **Python Pocket Reference: Python In Your Pocket**

  Book by Mark Lutz

- **Python 3 Object Oriented Programming**

  Book by Dusty Phillips

- **How to think like a computer scientist: Learning with Python**

  Book by Allen B. Downey

- **Python in a nutshell**

  Book by Alex Martelli

- **Natural Language Processing with Python**

  Book by Edward Loper, Ewan Klein, and Steven Bird

- **The Big Book of Small Python Projects: 81 Easy Practice Programs**

  Book by Al Sweigart

- **Python Programming for the Absolute Beginner**

  Book by Michael Dawson

- **Python Essential Reference**

  Book by David M. Beazley

- **Deep Learning with Python**

  Book by François Chollet

- **Dive into Python**

  Book by Mark Pilgrim

- **Impractical Python: Projects Playful Programming Activities to Make You Smarter**

  Book by Lee Vaughan

- **Object-Oriented Python: Master OOP by Building Games and GUIs**

  Book by Irv Kalb

- **The Python Bible 7 in 1: Volumes One To Seven (Beginner, Intermediate, Data Science, Machine Learning, Finance, Neural Networks, Computer Vision)**

  Book by Florian Dedov

- **Coding for Kids: Python: Learn to Code with 50 Awesome Games and Activities**

  Book by Adrienne Tacke

- **Learn More Python 3 the Hard Way: The Next Step for New Python Programmers**

  Book by Zed Shaw

- **Artificial Intelligence with Python**

  Book by Prateek Joshi

- **Python Programming: Using Problem Solving Approach**

  Book by Reema Thareja

- **Django for Beginners: Build websites with Python and Django**

  Book by William S. Vincent

- **Python for Dummies**

  Book by Aahz Maruch and Stef Maruch

- **Beginning Programming with Python For Dummies**

  Book by John Paul Mueller

HTML:

- **Learning Web Design**

  Book by Jennifer Niederst Robbins

- **Head First HTML with CSS and XHTML**

  Book by Elisabeth Robson and Eric Freeman

- **HTML5: The Missing Manual**

  Book by Matthew MacDonald

- **Learn HTML for Beginners: The Illustrated Guide to Coding**

  Book by Jo Foster

- **HTML5 for Web Designers**

  Book by Jeremy Keith

- **HTML 4 for the World Wide Web**

  Book by Elizabeth Castro

- **Introducing HTML5**

  Book by Bruce Lawson and Remy Sharp

- **Core HTML5 Canvas: Graphics, Animation, and Game Development**

  Book by David M. Geary

- **The Definitive Guide to HTML5**

  Book by Adam Freeman

- **Html: Basic Fundamental Guide for Beginners**

  Textbook by M. G. Martin

- **HTML5 in Easy Steps**

  Book by Mike McGrath

CSS:

- **CSS Pocket Reference: Visual Presentation for the Web**

  Book by Eric A. Meyer

- **CSS Secrets: Better Solutions to Everyday Web Design Problems**

  Book by Lea Verou

- **CSS: The Missing Manual**

  Book by David McFarland

- **CSS in Depth**

  Book by Keith Grant

- **CSS mastery**

  Book by Andy Budd

- **CSS: The Definitive Guide: Visual Presentation for the Web**

  Book by Eric A. Meyer and Estelle Weyl

- **CSS Visual Dictionary**

Book by Greg Sidelnikov

- **Cascading Style Sheets: The Definitive Guide**

  Book by Eric A. Meyer

- **The Book of CSS3: A Developer's Guide to the Future of Web Design**

  Book by Peter Gasston

- **CSS3: The Missing Manual**

  Book by David Sawyer McFarland

- **Learn CSS in One Day and Learn It Well: CSS for Beginners With Hands-On Project**

  Book by Jamie Chan

- **CSS Master**

  Book by Tiffany A. Brown

- **Basics of Web Design: HTML5 and CSS**

  Textbook by Terry A. Felke-Morris

- **CSS for Babies**

  Book by John Vanden-Heuvel

- **CSS in Easy Steps**

  Book by Mike McGrath

JAVASCRIPT:

- **JavaScript: The Good Parts**

  Book by Douglas Crockford

- **Eloquent JavaScript: A Modern Introduction to Programming**

  Book by Marijn Haverbeke

- **JavaScript and JQuery: Interactive Front-End Web Development**

  Book by Jon Duckett

- **You Don't Know JS: Scope and Closures**

  Book by Kyle Simpson

- **A Smarter Way to Learn JavaScript**

  Book by Mark Myers

- **Effective JavaScript : 68 specific ways to harness the power of JavaScript**

  Book by David Herman

- **Head First JavaScript Programming: A Brain-Friendly Guide**

  Book by Elisabeth Robson and Eric Freeman

- **JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language**

  Book by David Flanagan

- **Learn JavaScript Visually: With Interactive Exercises**

  Book by Ivelin Demirov

- **The Principles of Object-Oriented JavaScript**

  Book by Nicholas C. Zakas

- **Professional JavaScript for Web Developers**

  Book by Nicholas C. Zakas

- **Speaking JavaScript: An In-Depth Guide for Programmers**

  Book by Axel Rauschmayer

- **Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries**

  Book by Eric Elliott

- **JavaScript Enlightenment**

  Book by Cody Lindley

- **Learning JavaScript Design Patterns**

  Book by Addy Osmani

- **Secrets of the JavaScript Ninja**

  Book by Bear Bibeault, John Resig and Josip Maras

- **Beginning JavaScript**

  Book by Jeremy McPeak and Paul Wilton

- **JavaScript Patterns**

  Book by Stoyan Stefanov

- **Understanding ECMAScript 6: The Definitive Guide for JavaScript Developers**

  Book by Nicholas C. Zakas

- **JavaScript for Kids: A Playful Introduction to Programming**

  Book by Nick Morgan

- **High Performance JavaScript**

  Book by Nicholas C. Zakas

- **Object-Oriented JavaScript**

  Book by Kumar Chetan Sharma and Stoyan Stefanov

- **Maintainable JavaScript**

Book by Nicholas C. Zakas

- **Secrets of the JavaScript Ninja**

  Book by Bear Bibeault and John Resig

- **Coding with JavaScript For Dummies**

  Book by Chris Minnick and Eva Holland

- **DOM Enlightenment**

  Book by Cody Lindley

- **JavaScript from Beginner to Professional: Learn JavaScript Quickly by Building Fun, Interactive, and Dynamic Web Apps, Games, and Pages**

  Book by Laurence Lars Svekis and Rob Percival

- **Testable JavaScript**

  Book by Mark Ethan Trostler

- **You Don't Know JS: Up and Going**

  Book by Kyle Simpson

- **JavaScript Allongé: A strong cup of functions, objects, combinators, and decorators**

  Book by Reginald Braithwaite

- **You Don't Know JS Yet: Get Started**

  Book by Kyle Simpson

- **JavaScript and JQuery: The Missing Manual**

  Book by David Sawyer McFarland

- **The Recursive Book of Recursion: Ace the Coding Interview with Python and JavaScript**

  Book by Al Sweigart

- **The Road to Learn React: Your Journey to Master Plain Yet Pragmatic React. Js**

  Book by Robin Wieruch

- **Javascript In Easy Steps**

  Book by Mike McGrath

- **Building JavaScript Games: For Phones, Tablets, and Desktop**

  Book by Arjan Egges

- **High Performance Browser Networking**

  Book by Ilya Grigorik

- **Learning TypeScript**

  Book by Josh Goldberg

- **Pro JavaScript Techniques**

  Book by John Resig

- **The Little Book on CoffeeScript**

  Book by Alex MacCaw

- **Learning React: Functional Web Development with React and Redux**

  Book by Alex Banks and Eve Porcello

- **Learn JavaScript Quickly: A Complete Beginner's Guide to Learning JavaScript, Even If You're New to Programming**

Book by Code Quickly

- **Test-Driven JavaScript Development**

  Book by Christian Johansen

- **Structure and Interpretation of Computer Programs, JavaScript Edition**

  Textbook by Gerald Jay Sussman and Hal Abelson

- **Functional Programming in JavaScript: How to Improve Your JavaScript Programs Using**

  Functional Techniques

  Book by Luis Atencio

- **Pro JavaScript Design Patterns**

  Book by Dustin Diaz and Ross Harmes

- **Java Script: The Complete Reference 2/E**

  Book by Thomas Powell

PHP:

- **The Joy of PHP: A Beginner's Guide to Programming Interactive Web Applications with**

  PHP and MySQL

  Book by Alan Forbes

- **Learning PHP, MySQL, JavaScript, CSS and HTML5: A Step-by-Step Guide to Creating**

  Dynamic Websites

  Book by Robin Nixon

- **Programming PHP**

Book by Rasmus Lerdorf

- **PHP and MySQL Web Development**

  Book by Laura Thomson and Luke Welling

- **Murach's PHP and MySQL**

  Book by Joel Murach and Ray Harris

- **Head First PHP and MySQL**

  Book by Lynn Beighley and Michael Morrison

- **PHP and MySQL: Novice to Ninja: Get Up to Speed With PHP the Easy Way**

  Book by Tom Butler and Kevin Yank

- **PHP and MySQL: Server-side Web Development**

  Book by Jon Duckett

- **Learning PHP, MySQL & JavaScript: With JQuery, CSS & HTML5**

  Book by Robin Nixon

- **PHP and MySQL in easy steps, 2nd Edition: Updated to cover MySQL 8.0**

  Book by Mike McGrath

- **PHP: A Beginner's Guide**

  Book by Vikram Vaswani

- **Modern PHP: New Features and Good Practices**

  Book by Josh Lockhart

- **Laravel: Up & Running: A Framework for Building Modern PHP Apps**

Book by Matt Stauffer

- **Mastering PHP 7**

  Book by Branko Ajzele

- **PHP and MySQL**

  Book by Jon Duckett

- **PHP and MySQL: The Missing Manual**

  Book by Brett McLaughlin

- **PHP 8 Programming Tips, Tricks and Best Practices: A Practical Guide to PHP 8 Features,**

  Usage Changes, and Advanced Programming Techniques

  Book by Cal Evans and Doug Bierer

- **PHP for the Web: Visual QuickStart Guide**

  Book by Larry Ullman

- **PHP and MySQL for Dynamic Web Sites**

  Book by Larry Ullman

- **PHP: Learn PHP in One Day and Learn It Well. PHP for Beginners with Hands-on Project**

  Book by Jamie Chan

- **PHP 7: Real World Application Development**

  Book by Altaf Hussain, Branko Ajzele and Doug Bierer

- **PHP Cookbook**

  Book by Adam Trachtenberg and David Sklar

- **Full Stack Web Development For Beginners: Learn Ecommerce Web Development Using HTML5, CSS3, Bootstrap, JavaScript, MySQL, and PHP**

  Book by Riaz Ahmed

- **Drupal 9 Module Development: Get Up and Running with Building Powerful Drupal Modules and Applications**

  Book by Daniel Sipos

- **Building Web Apps with WordPress: WordPress as an Application Framework**

  Book by Brian Messenlehner and Jason Coleman

- **PHP and MySQL in easy steps**

  Book by Mike McGrath

- **WordPress Complete**

  Book by Karol Król

- **PHP, MySQL, and JavaScript All-in-One For Dummies**

  Book by Richard BLUM

- **PHP in Action: Objects, Design, Agility**

  Book by Chris Shiflett, Dagfinn Reiersol, and Marcus Baker

- **PHP Advanced and Object-Oriented Programming: Visual QuickPro Guide**

  Book by Larry Ullman

- **Learning PHP 7**

  Book by Antonio Lopez

- **PHP 7 Programming Cookbook**

  Book by Doug Bierer

- **PHP: The Complete Reference**

  Book by Steven Holzner

- **PHP and MySQL for Dummies**

  Book by Janet Valade

- **PHP Beginner's Practical Guide**

  Book by Pratiyush Guleria

- **Building RESTful Web Services with PHP 7**

  Book by Haafiz Waheed-ud-din Ahmad

- **Mastering PHP Design Patterns**

  Book by Junade Ali

- **PHP and MongoDB Web Development Beginner's Guide**

  Book by Rubayeet Islam

- **PHP 7 Data Structures and Algorithms**

  Book by Mizanur Rahman

- **Mastering The Faster Web with PHP, MySQL, and JavaScript: Develop State-of-the-art Web Applications Using the Latest Web Technologies**

  Textbook by Andrew Caya

- **Learning PHP: A Gentle Introduction to the Web's Most Popular Language**

Book by David Sklar

- **Learn PHP 8: Using MySQL, JavaScript, CSS3, and HTML5**

  Book by Steve Prettyman

- **PHP Microservices**

  Book by Carlos Perez Sanchez and Pablo Solar Vilarino

- **Professional CodeIgniter**

  Book by Thomas Myer

- **Symfony 5: The Fast Track**

  Book by Fabien Potencier

- **Practical PHP and MySQL Website Databases: A Simplified Approach**

  Book by Adrian W. West

- **PHP Programming for Beginners: Key Programming Concepts. How to use PHP with MySQL and Oracle databases (MySqli, PDO)**

  Book by Sergey Skudaev

ALGORITHMS:

- **Introduction to Algorithms**
  Book by T Cormen, C Leiserson, R Rivest, C Stein

- **The Algorithm Design Manual**
  Book by Steven Skiena

- **Grokking Algorithms: An Illustrated Guide for Programmers and Other Curious People**

Book by Aditya Bhargava

- **Algorithms**

  Book by Robert Sedgewick

- **Algorithm Design**

  Book by Jon Kleinberg and Éva Tardos

- **Algorithms Illuminated: Algorithms for NP-hard problems**

  Book by Tim Roughgarden

- **Algorithms in a nutshell**

  Book by George T. Heineman

- **Data structures and algorithms made easy in Java: data structure and algorithmic puzzles**

  Book by Narasimha karumanchi

- **A Common-Sense Guide to Data Structures and Algorithms: Level Up Your Core Programming Skills**

  Book by Jay Wengrow

- **Cracking the Coding Interview**

  Book by Gayle Laakmann McDowell

- **Guide to Competitive Programming: Learning and Improving Algorithms Through Contests**

  Book by Antti Laaksonen

- **Algorithms Unlocked**

  Book by Thomas H. Cormen

- **Problem Solving with Algorithms and Data Structures using Python**

  Book by Bradley N Miller and David L. Ranum

- **Computer Science Distilled: Learn the Art of Solving Computational Problems**

  Book by Wladston Ferreira Filho

- **Python Algorithms: Mastering Basic Algorithms in the Python Language**

  Book by Magnus Lie Hetland

- **Advanced Algorithms and Data Structures**

  Book by Marcello La Rocca

- **The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World**

  Book by Pedro Domingos

- **Dive Into Algorithms: A Pythonic Adventure for the Intrepid Beginner**

  Book by Bradford Tuckfield

- **Advanced Data Structures**

  Book by Peter Brass

- **Algorithmic Thinking: A Problem-Based Introduction**

  Book by Daniel Zingaro

- **Algorithms + Data Structures = Programs**

  Book by Niklaus Wirth

- **Algorithms Illuminated (Part 2): Graph Algorithms and Data Structures**

  Book by Tim Roughgarden

- **Introduction to the Design and Analysis of Algorithms**

  Book by Anany Levitin

- **Automate This: How Algorithms Came to Rule Our World**

  Book by Christopher Steiner

- **Think Like a Programmer: An Introduction to Creative Problem Solving**

  Book by V. Anton Spraul

- **Spies, Lies, and Algorithms: The History and Future of American Intelligence**

  Book by Amy Zegart

- **Algorithms Illuminated (Part 3): Greedy Algorithms and Dynamic Programming**

  Book by Tim Roughgarden

- **Algorithms for Decision Making**

  Book by Kyle H. Wray, Mykel J. Kochenderfer and Tim A. Wheeler

- **Fundamentals of Computer Algorithms**

  Book by Ellis Horowitz and Sartaj Sahni

- **Information Theory, Inference and Learning Algorithms**

  Book by David J. C. MacKay

- **9 Algorithms That Changed the Future**

  Book by John MacCormick

- **The Art of Computer Programming**

  Book by Donald Knuth

- **Data Structure and Algorithmic Thinking with Python**

  Book by Narasimha Karumanchi

- **A Common-Sense Guide to Data Structures and Algorithms**

  Book by Jay Wengrow

- **Hello World: How to be Human in the Age of the Machine**

  Book by Hannah Fry

- **The Design and Analysis of Computer Algorithms**

  Book by Alfred Aho, Jeffrey Ullman, and John Hopcroft

- **Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services**

  Book by Brendan Burns

- **A Programmer's Guide to Computer Science: A virtual degree for the self-taught developer**

  Book by William M Springer

- **Concrete Mathematics**

Textbook by Donald Knuth, Oren Patashnik and Ronald Graham

- **Understanding Cryptography: A Textbook for Students and Practitioners**

  Textbook by Christof Paar and Jan Pelzl

- **Algorithms For Dummies**

  Book by John Mueller and Luca Massaron

DATA STRUCTURES:

- **Data Structures and Algorithm Analysis in C++**

  Book by Clifford A Shaffer

- **Purely functional data structures**

  Book by Chris Okasaki

- **Open Data Structures: An Introduction**

  Book by Pat Morin

- **Think Data Structures: Algorithms and Information Retrieval in Java**

  Book by Allen B. Downey

- **Data Structures Using C**

  Book by Reema Thareja

- **Data Structures Through C in Depth**

  Book by Deepali Srivastava and Suresh Kumar Srivastava

- **C++ Plus Data Structures**

  Book by Nell B. Dale

- **JavaScript Data Structures and Algorithms: An Introduction to Understanding and Implementing Core Data Structure and Algorithm Fundamentals**

Book by Sammie Bae

- **Data structures using C and C++**

  Book by Yedidyah Langsam

- **Algorithms and Data Structures: The Basic Toolbox**

  Book by Kurt Mehlhorn and Peter Sanders

- **Data Structures and Algorithm Analysis in Java**

  Book by Clifford A Shaffer

- **Handbook of Data Structures and Applications**

  Book by Dinesh P. Mehta and Sartaj Sahni

- **Data structures with Java**

  Book by J. R Hubbard

- **Java Structures**

  Book by Duane A. Bailey

LINUX:

- **The Linux Command Line: A Complete Introduction**

  Book by William E. Shotts Jr. and William E. Shotts, Jr.

- **The Linux Programming Interface**

  Book by Michael Kerrisk

- **Linux bible**

  Book by Christopher Negus

- **Linux pocket guide**

Book by Daniel J. Barrett

- **Linux for Beginners**

  Book by Jason Cannon

- **How Linux Works: What Every Superuser Should Know**

  Book by Brian Ward

- **Linux Basics for Hackers: Getting Started with Networking, Scripting, and Security in Kali**

  Book by OccupyTheWeb

- **Linux Kernel Development**

  Book by Robert Love

- **Linux Administration: The Linux Operating System and Command Line Guide for Linux Administrators**

  Book by Jason Cannon

- **Linux: The Complete Reference**

  Book by Richard Petersen

- **Linux Command Line and Shell Scripting Bible**

  Book by Christine Bresnahan and Richard BLUM

- **Linux in a Nutshell**

  Book by Ellen Siever and Robert Love

- **The Art of Unix Programming**

  Book by Eric S. Raymond

- **Linux System Programming: Talking Directly to the Kernel and C Library**

  Book by Robert Love

- **A Practical Guide to Linux Commands, Editors, and Shell Programming**

  Book by Mark G. Sobell

- **Learn Linux Quickly: A Beginner-friendly Guide to Getting Up and Running with the World's Most Powerful Operating System**

  Book by Ahmed Alkabary

- **Linux for Beginners and Command Line Kung Fu**

  Book by Jason Cannon

- **The Ultimate Kali Linux Book: Perform Advanced Penetration Testing Using Nmap, Metasploit, Aircrack-Ng, and Empire**

  Book by Glen D. Singh

- **Understanding the Linux Kernel**

  Book by Daniel Pierre Bovet and Marco Cesati

- **Linux for Developers: Jumpstart Your Linux Programming Skills**

  Book by William "Bo" Rothwell

- **Learning the bash Shell**

  Book by Cameron Newham

- **Linux Device Drivers**

  Book by Alessandro Rubini, Greg Kroah-Hartman, and Jonathan Corbet

- **Mastering Linux Shell Scripting: A Practical Guide to Linux Command-line, Bash Scripting, and Shell Programming**

  Book by Andrew Mallett and Mokhtar Ebrahim

- **Efficient Linux at the Command Line**

  Book by Daniel J. Barrett

- **Shell Scripting: How to Automate Command Line Tasks Using Bash Scripting and Shell Programming**

  Book by Jaosn Cannon

- **Advanced Programming in the Unix Environment**

  Book by W. Richard Stevens

- **Lfm: Linux Field Manual**

  Book by Tim Bryant

- **CompTIA Linux+ Certification All-in-One Exam Guide**

  Book by Ted Jordan and Sandor Strohmayer

- **Advanced Linux programming**

  Book by Alex Samuel, Jeffrey Oldham and Mark Mitchell

- **Linux Kernel in a Nutshell**

  Book by Greg Kroah-Hartman

- **Linux Kernel Debugging: Leverage Proven Tools and Advanced Techniques to Effectively Debug Linux Kernels and Kernel Modules**

  Book by Kaiwan N Billimoria

- **Bash cookbook**

  Book by Carl Albing

- **Linux for Beginners: A Practical and Comprehensive Guide to Learn Linux Operating System and Master Linux Command Line.**

  Book by Ethem Mining

- **Beginning Linux Programming**

  Book by Mathew Neil and Richard Stones

- **The Linux Command Line Beginner's Guide**

  Book by Jonathan Moeller

- **Linux System Programming**

  Book by Robert Love

- **A Practical Guide to Fedora and Red Hat Enterprise Linux**

  Book by Mark G. Sobell

- **Kali Linux Hacking: A Complete Step by Step Guide to Learn the Fundamentals of Cyber Security, Hacking, and Penetration Testing**

  Book by Ethem Mining

- **Kali Linux Revealed: Mastering the Penetration Testing Distribution**

  Book by Jim O'Gorman, Mati Aharoni and Raphael Hertzog

- **Linux in Easy Steps**

  Book by Mike McGrath

- **Bash Pocket Reference: Help for Power Users and Sys Admins**

  Book by Arnold Robbins

- **Learn PowerShell in a Month of Lunches: Covers Windows, Linux, and macOS**

  Book by Travis Plunk, James Petty and Leon Leonhardt

- **Linux From Scratch**

  Book by Gerard Beekmans

- **Linux For Dummies**

  Book by Richard Blum

DATABASE:

- **Database Systems: The Complete Book**

  Book by Héctor García-Molina, Jeffrey Ullman and Jennifer Widom

- **Fundamentals of Database Systems**

Book by Ramez Elmasri

- **Database Design for Mere Mortals**

  Book by Michael J. Hernandez

- **An Introduction to Database Systems**

  Book by Christopher J. Date

- **SQL Antipatterns: Avoiding the Pitfalls of Database Programming**

  Book by Bill Karwin

- **Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems**

  Book by Martin Kleppmann

- **Database System Concepts**

  Book by Avi Silberschatz, Henry F. Korth and S. Sudarshan

- **Database Internals: A Deep Dive Into How Distributed Data Systems Work**

  Book by Alex Petrov

- **Beginning Database Design Solutions**

  Book by Rod Stephens

- **SQL Performance Explained**

  Book by Markus Winand

- **Database Management Systems**

  Book by Johannes Gehrke and Raghu Ramakrishnan

- **Oracle PL/SQL programming**

  Book by Scott Urman

- **Database in Depth: Relational Theory for Practitioners**

  Book by Christopher J. Date

- **Database Design and Relational Theory: Normal Forms and All That Jazz**

Book by Christopher J. Date

- **Database Systems: A Practical Approach to Design, Implementation, and Management**

  Book by Carolyn Begg and Thomas M. Connolly

- **Introductory Relational Database Design for Business, with Microsoft Access**

  Book by Bonnie R. Schultz and Jonathan Eckstein

- **Head First SQL: Your Brain on SQL − A Learner's Guide**

  Book by Lynn Beighley

- **Readings in Database Systems**

  Book by Joseph M Hellerstein

- **The art of SQL**

  Book by Stéphane Faroult

- **The theory of relational databases**

  Book by David Maier

- **The Data Warehouse Toolkit**

  Book by Ralph Kimball

- **MongoDB: The Definitive Guide**

  Book by Kristina Chodorow and Michael Dirolf

- **Learning SQL**

  Book by Alan Beaulieu

- **Joe Celko's SQL puzzles and answers**

  Book by Joe Celko

- **Foundations of databases**

  Book by Serge Abiteboul

- **Pro SQL Server Relational Database Design and Implementation: Best Practices for Scalability and Performance**

Book by Louis Davidson

- **Six-Step Relational Database Design: A Step by Step Approach to Relational Database Design and Development**

  Book by Fidel A. Captain

- **Databases, Types and the Relational Model: The Third Manifesto**

  Book by Christopher J. Date

- **Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement**

  Book by Eric Redmond and Jim R. Wilson

- **Practical SQL: A Beginner's Guide to Storytelling with Data**

  Book by Anthony DeBarros

- **Refactoring databases**

  Book by Scott Ambler

- **SQL QuickStart Guide: The Simplified Beginner's Guide to Managing, Analyzing, and Manipulating Data With SQL**

  Book by Walter Shields

- **PHP and MySQL: Server-side Web Development**

  Book by Jon Duckett

- **SQL Cookbook**

  Book by Anthony Molinaro

- **Learn MongoDB 4.x: A Guide to Understanding MongoDB Development and Administration for NoSQL Developers**

  Book by Doug Bierer

- **Professional Azure SQL Managed Database Administration: Efficiently Manage and Modernize Data in the Cloud Using Azure SQL**

  Book by Ahmad Osama and Shashikant Shakya

- **PostgreSQL: Up and Running**

  Book by Leo S. Hsu and Regina O. Obe

- **SQL All-in-One for Dummies**

  Book by Allen G. Taylor

- **An Introduction to Relational Database Theory**

  Book by Hugh Darwen

- **Transaction Processing: Concepts and Techniques**

  Book by Jim Gray

- **Making Sense of NoSQL: A Guide for Managers and the Rest of Us**

  Book by Ann Kelly, Ann Marie Kelly and Dan McCreary

- **A Practical Guide to Database Design**

  Book by Rex Hogan

- **Database Development for Dummies**

  Book by Allen G. Taylor

- **SQL Queries for Mere Mortals: A Hands-on Guide to Data Manipulation in SQL**

  Book by John Viescas and Michael J. Hernandez

- **Pro SQL Server 2012 Relational Database Design and Implementation**

  Book by Jessica Moss and Louis Davidson

- **SQL Performance Explained: Everything Developers Need to Know about SQL Performance**

  Book by Markus Winand

- **Refactoring Databases: Evolutionary Database Design**

  Book by Scott W Ambler and Pramod J Sadalage

- **Designing Data-Intensive Applications**

  Book by Martin Kleppmann

Programming today is a race between software engineers striving to

build bigger and better **idiot-proof programs**, and the Universe trying to

produce bigger and better idiots. So far, the Universe is winning.

— Rick Cook, The Wizardry Compiled

"In some ways, programming is like painting. You start with a blank canvas and certain basic raw materials. You use a combination of science, art, and craft to determine what to do with them."

– Andrew Hunt

One final thought:

If you feel that this information has been useful to you, please take a moment to share it with your friends on LinkedIn, Facebook and Twitter. Consider writing a brief review on **Google Play Books** if you feel that this book has helped you in your programming career and you have learned something worthwhile.

Coding is both a science and creative art to me. It is both incredibly fun and fascinating. I want to spread my passion to as many individuals as I can. I also hope that this is not the end of your learning.

Thank you!

AN APPROACHABLE MANUAL FOR NEW AND EXPERIENCED PROGRAMMERS THAT INTRODUCES THE PROGRAMMING LANGUAGES C, C++, JAVA, AND PYTHON. THIS BOOK IS FOR ALL PROGRAMMERS, WHETHER YOU ARE A NOVICE OR AN EXPERIENCED PRO. IT IS DESIGNED FOR AN INTRODUCTORY COURSE THAT PROVIDES BEGINNING ENGINEERING AND COMPUTER SCIENCE STUDENTS WITH A SOLID FOUNDATION IN THE FUNDAMENTAL CONCEPTS OF COMPUTER PROGRAMMING. IT ALSO OFFERS VALUABLE PERSPECTIVES ON IMPORTANT COMPUTING CONCEPTS THROUGH THE DEVELOPMENT OF PROGRAMMING AND PROBLEM-SOLVING SKILLS USING THE LANGUAGES C, C++, JAVA, AND PYTHON. THE BEGINNER WILL FIND ITS CAREFULLY PACED EXERCISES ESPECIALLY HELPFUL. OF COURSE, THOSE WHO ARE ALREADY FAMILIAR WITH PROGRAMMING ARE LIKELY TO DERIVE MORE BENEFITS FROM THIS BOOK. AFTER READING THIS BOOK YOU WILL FIND YOURSELF AT A MODERATE LEVEL OF EXPERTISE IN C, C++, JAVA AND PYTHON, FROM WHICH YOU CAN TAKE YOURSELF TO THE NEXT LEVELS. THE COMMAND-LINE INTERFACE IS ONE OF THE NEARLY ALL WELL BUILT TRADEMARKS OF LINUX. THERE EXISTS AN OCEAN OF LINUX COMMANDS, PERMITTING YOU TO DO NEARLY EVERYTHING YOU CAN BE UNDER THE IMPRESSION OF DOING ON YOUR LINUX OPERATING SYSTEM. HOWEVER, THIS, AT THE END OF TIME, CREATES A PROBLEM: BECAUSE OF ALL OF SO COPIOUS COMMANDS ACCESSIBLE TO MANAGE, YOU DON'T COMPREHEND WHERE AND AT WHICH POINT TO FLY AND LEARN THEM, ESPECIALLY WHEN YOU ARE A LEARNER. IF YOU ARE FACING THIS PROBLEM, AND ARE PEERING FOR A PAINLESS METHOD TO BEGIN YOUR COMMAND LINE JOURNEY IN LINUX, YOU'VE COME TO THE RIGHT PLACE-AS IN THIS BOOK, WE WILL LAUNCH YOU TO A HOLD OF WELL LIKED AND HELPFUL LINUX COMMANDS. THIS BOOK GIVES A THOROUGH INTRODUCTION TO THE C, C++, JAVA, AND PYTHON PROGRAMMING LANGUAGES, COVERING EVERYTHING FROM FUNDAMENTALS TO ADVANCED CONCEPTS. IT ALSO INCLUDES VARIOUS EXERCISES THAT LET YOU PUT WHAT YOU LEARN TO USE IN THE REAL WORLD.

LINUX COMMANDS, C, C++, JAVA AND PYTHON EXERCISES FOR BEGINNERS

MANJUNATH.R