Full stack with Python:
==================

full stack:
========

it is a combination of "front-end and back-end"

what is front-end?
==============

front-end means "application user interface" and using this we

can able to make the user can interact with application

to make the application user interface, we will use the following

technologies:
==================================================

html:
====

it is used to create the  Skelton or structure of the

application UI

html stands for "hypertext markup language"

markup language is a language and which is uses "tags" to

describe the data

in general ,we will use the following markup languages:
=============================================

1.html (to design or create structure of the webpage)

2.XML (it is used for data exchange format between the servers)

CSS:
====

css stands for cascading style sheets

css is used in application UI, to give the look and feel of the

webpage or application

JavaScript:
========

we will use javascript, to write code or any coding at front-end level

jQuery:
======

jQuery is a "javascript library" and it provides various functions

for events and effects for application UI

bootstrap:
========

bootstrap is a css framework and it is used for "responsive web

design"

what is back-end?
==============

back-end means "application server and application database"

what is mean by application server?
===========================

application server is a machine or computer,which can able

to process the all users actions/request in the application

to create the any server-side code, we will use the following

languages:
===============================================

1.python

2.java

3.php

4.asp.net

5.ruby,.....................................

what is mean by application database?
==============================

application database is used to store the "application users

data or customers data"

application database means "storage unit of application"

what is mean by data?
=================

data means "what we can able to store in the computer

memory"

example:  image, audio, video, text document,.......

what is mean by information?
=======================

when we keep data in process, we will get information

or

processed data is also known as "information"

example:
========

10+20 ====>30 here 10,20 are data and + is process

30 is information

today gold price

today company stock price

in real time we will store the data in the two types of databases:
=================================================

1.sql-database:
============

in sql-databases data will be stored in "table format" and to

work with this database, we will use "sql" as language

example:
=======

MySQL,oracle,sql server,postgres sql,................

2.no-sql database:
==============

in no-sql database, data will be stored in "document format"

and this we never use any language like "SQL"

example:

mongoDB, cassendra,.............

in this course we are learning the following:
====================================

front-end:
========

html, css, javascript, jquery and bootstrap

back-end:
=======

server-side:  python(core and advance python)

databases-side:

sql with MySQL

mongoDB

framework:
=========

DJango

types of applications:
=================

in real time ,we will have the following types of applications:
==================================================

1.stand-alone application/desktop application:
======================================

if we say any application is "stand-alone application", then

the application does not uses "internet" to access the

application and "server" not run the application

example:
=======

calculator, clock, notepad,..............

the major drawback of the stand-alone application is "data can

not be shared "

2.web application:
==============

web application is a application which requires "internet to access and server to run the application"

example:
=======

gmail, whatsapp, college website, company website,...........

in this application, we can share the data from one device

to other device

3.enterprise application:
===================

enterprise application is a "web application"

in this application also ,we need both internet to access and

server to run the application

example:  amazon, mynthra, flipkart,............

in this application we will have "Sever to server" communication

4.mobile application:
=================

5.distributed application
=================================================

python:
=======

what is python?
============

python is a "high-level, strongly-typed, dynamic typed,

object-oriented, programming language"

why python is "high-level" ?
=====================

in general, in computers we will have three types of languages:
=================================================

1.high-level language:
=================

high-level language means "a language which is used by the

programmers or developers"

example: python, c, c++, java, php, ruby,...........

the high-level language is again converted into two types:
=============================================

1.programming language

2.scripting language

when we write any python program, first we need to convert

the given python program into "low-level/binary code",

in order to convert the python program into binary, in

computers, we will use the following language translators:
==============================================

1.compiler:
==========

compiler is a language translator and using compiler we can

convert the high-language code into binary language code

basically ,"compiler is a software"

in general, all programming language uses "compiler" as a

translator

compiler takes whole program and convert into "binary" at

a time

compiler performance is very high than "interpreter"

in the case of compiler, the de-bugging is very difficult

de-bugging is means "finding and rectifying the errors from

code/program"

the following languages uses compiler as translator:
=========================================

python

java

c

c++

if any language uses "compiler" as translator, then the

language called as "programming language"

compiler will check only "syntax", but not logic of the  program

3.interpreter:
===========

interpreter is a language translator and using interpreter we

can convert the high-language code into binary language code

basically ,"interpreter is a software"

in general, all scripting language uses "interpreter" as a

translator

interpreter takes line by line from  program and convert into

"binary", if any error occurs while translation, it halt the process

interpreter performance is very poor than "compiler"

in the case of interpreter, the de-bugging is very easy

the following languages uses interpreter as translator:
==========================================

python

java

javascript

PHP,....

if any language uses "interpreter" as translator, then the

language called as "scripting language"

in general,  script will take less code the program

3.binary language/machine language/low-level language:
=============================================

a language which is understand by the "machine(computer)",

is known as "binary/machine/low-level"

binary language is "language  and which is understand by

machine and which is composed using only 0’s and 1’s"

why python is "Strongly-typed" language?
================================

in python, when we are performing any operation on

operands(on which we will perform operation),the operands

must be similar type/same type ,other wise python simply

raises an error(TypeError), that is reason python is called as

"strongly typed language"

if any language does not check any type, while performing

the operation of the operands, then the language is called as

"loosely typed language"

example:
=======

java script is a loosely typed language

why python is "dynamic-typed" language?
================================

in python, when we store any data into variable, we no need

to specify the data type of the variable ,the data type of the

variable will decided at runtime based on the value what we

store in the variable, that reason python is called as "dynamic

typed language"

if any language makes the programmer or developer has to

define the "Data type" of the variable, before the use,then

the language is called as "static-typed language"

example:
=======

c, c++, java..........

why python is "object-oriented" language:
================================

objected-oriented means " it is very popular programming

paradigm(model) in computer science"

if any language supports the following features, then the

language is called as "object oriented programming language":

1.classes and objects

2.Data Abstraction

3.Data Encapsulation

4.Polymorphism

5.Inheritance

python supports above all features, that is reason python called

as "object-oriented" language

why we need to learn python?
=======================

1.python is simple and easy to due to "syntax is so simple than

other languages"

2.python is platform independent:
===========================

platform====>operating system + processor

when we write any program in python,

we need to run the python program, while running the python

program, it is a two step process:

let assume the python file name is "sample.py"

step-1:
======

first "Sample.py" will given to "python compiler"

sample.py ====>python compiler ==> Sample.pyc(byte code)

step-2:
======

once we got the byte code, this byte code will run in any

platform(os +processor),

this byte code can understand only by "PVM(python virtual

machine)"

this will convert byte code into executable code, once

executable code given to processor, processor will execute give

the "output"

3.python is a freeware and open source (the complete

implementation/source code of python is visible, any one

access it and change any thing the way we want

because of the open source, we will have different python

flavours:
=================================================

1.cpython(most commonly ,we are using cPython)

2.jython or jpython

3.IronPython

4.anaconda python(for bigdata)

5.stackless(for concurrency)

4.python is used to develop the following:
================================

1.Application Development

2.AI

3.IOT

4.Data Science and Data Analytics

5.Data Engineering

6.Cloud and Devops

7.Gaming development

8.Testing

9.Networking,.......................

5.python is extensible and embedded:
================================

we can write the python code any where in other languages,

because  python is can be "embedded"

we can write any other language code along with python

because python is can be "Extensible"

6.python is having huge libraries then other languages

7.python is dynamic typed and strongly typed

how to work with python:
=====================

to write any python program, we are going to use the following

IDLE’s:
=====================================================

1.python idle(which is comes from python, when we install )

2.spyder

3.jupyter note book

4.vs code editor

5.pycharm

6.google colab

note:
=====

IDLE allows the programmer "can write, run and test  the code"

Core Python:
==========

1.python language fundamentals:
=============================

1.python character set:
====================================

python is also a language, in python we will have the following

characters:
====================================================

1.english alphabets:
==================

uppercase : A,B,C,D,....Z

lowercase : a,b,c,d,e,...z

2.digits:  0,1,2,3,4,5,6,7,8,9

3.special symbols/chracters:
======================
!,@,#,$,%,^(caret),&,*,(,),{,},[,],:,;,<,>,?,/,\,+,=,-,.,,,.........

2.python keywords or reserved words:
===================================================

keywords or reserved words are given by the python language,

using this keywords we can able perform a specified task in

the program

in python, we will have the following keywords:
=======================================

for logical operartors:  or,and and not

for identity operators:  is

for membership operators:  in

for conditional statements:  if, else, elif, match, case

for un-conditional statements: break, continue and return

for empty block: pass

for looping:  while  and for

for functions:  def, lambda

for Boolean values:  True and False

for modules and packages:  from , import , as

for file handling:  with

for exception handling:  try, except , raise, finally, assert for multi-threading: async

for oops: class, self ,

for none data type: None

3.python comments
==================================================

comments " are going to describe the code"

comments are written in natural language like "English"

comments are never going to effect the program output

comments are going to be ignored "while translating the code"

by the compiler

when we write the comments in python program, the program

can gets readability

in python, we will write the comments in two ways:
=========================================

1.single line comments:
===================

writing the comments in single line using a symbol called

"#"

2.multi-line comments:
==================

writing the comments in multiple lines using a symbol called

"triple quotes(''' '''  or """ """)"

4.python identifiers:
==================================================

identifier means "it represent any name in the program"

the name may be "variable name, function name, class name,

list name, tuple name,................."

to create the identifier in python program, we will use

the following rules:
=================================================

1.every identifier(name) must start with letter or underscore(_)

2.every identifier(name) may have the following characters:
==============================================

1.all letters(a-z,A-Z)

2.all digits(0-9)

3.underscore(_)

3.every identifier(name) can be alphanumeric(it can have both

letters and digits)

4.every identifier(name)  can not have "whitespace(space)"

5.every identifier(name) can not be reserved word or keyword

name

6.every identifier length can be anything

example:
=======

abc_123(valid)

123abc(invalid)

abc#(invalid)

_123(valid)

a_123(valid)

#abc(valid(comment))

python input and output statements:
=============================

python input statement:
===================

in python, to give any input ,to the program we will use a

function called "input()"

here "input()" function is a "built-in function" ( built-in function

is a function which is given python)

input ===>input() ===> python program

input() is used as "input" function in python from version 3.0

onwards

before python 3.0 version ,in python to take any input to the

program ,we are using a function called "raw_input()" (it used

in python 2.0 version)

in python ,the input() function will always takes "any input"  as

string only

number ====>input() ===>string

string ====>input() ====>string

any data ====>input() ====>string

when we take any data from the input() function, the data will

always "string", due to reason, in python we are using a process

a called "type conversion or type casting"

type conversion is process of "converting one datatype data

into another desired data type"

in python, we will have two types of type conversion:
=========================================

1.implicit type conversion:
=====================

this conversion is done by the PVM or machine, but not by

the programmer or developer

2.explicit type conversion:
=====================

this type conversion will done by the programmer or developer,

but not by PVM or machine

in python to implement, type conversion we will use the

following functions:

int():
====

int() is a built-in function (which is given by python)

int() function is used to " to convert the given into integer type"

syntax:
======

int(data) ===>integer type

example:
=======
```
"""
author:Ram
program to work with int() function
"""
a=int(10)
print(a)
a=int(1.234)
print(a)
a=int("10")
print(a)
a=int()
print(a)
a=int("+12")
print(a)
a=int("-12")
print(a)
#a=int("12.34")#error
a=int("12_34")
print(a)
#a=int(10+5j)
a=int("0123")
print(a)
a=int(+123)
print(a)
a=int(-123)
print(a)
a=int(True)
print(a)
a=int(False)
print(a)
```

note:
====

while using int() function, if we give any number as "String",

then number must has only following special characters:
==========================================

+/- at begin

_ at between digits only

2.float():
=======
this function is used to convert  given data into "Float" type

float(data) ===> float number

example:
=======
```
 """
author:Ram
program to work with float() function
"""
a=float(10)
print(a)
a=float(+1.234)
print(a)
a=float(0123.456)
print(a)
a=float(12_45)
print(a)
#a=float(10+5j)
a=float("12.34")
print(a)
a=float("12.")
print(a)
a=float("+1_2.")
print(a)
```

3.bool():
=======

this function is used to convert  given data into "Boolean" type

bool(data) ===> boolean number

example:
========
```
"""
author:Ram
program to work with bool() function
"""
a=bool(100)#True
print(a)
a=bool(-100)
print(a)
a=bool(None)
print(a)
a=bool(‘’)
print(a)
a=bool(10-10)
print(a)
a=bool(-45678)
print(a)
a=bool([])
print(a)
```

a=bool(())
print(a)
a=bool({})
print(a)
a=bool(set())
print(a)
a=bool(range(1,5))
print(a)

4.complex():
========

this function is used to convert the given data into

"complex" number

complex(data) ===>complex number

example:
=======
```
"""
author:Ram
program to work with complex() function
"""
a=complex(100)
print(a)
a=complex(1.234)
print(a)
a=complex("-5j")
print(a)
a=complex(5j)
print(a)
a=complex(5J)
print(a)
a=complex(True)
print(a)
a=complex(False)
print(a)
a=complex("+12_34")
print(a)
a=bool(10+5j)
print(a)
a=bool(10-60j)
print(a)
a=bool(-10j)
print(a)
a=bool(0j)
print(a)
```

5.str():
=====

str() function is to convert  any data into string type

str(data) ===>string data

example:
=======

```
"""
author:Ram
program to work with str()
"""
a=str(100)
print(a,type(a))
a=str(1.234)
print(a,type(a))
a=str(True)
print(a,type(a))
a=str(10+5j)
print(a,type(a))
a=str("abc")
print(a,type(a))
a=str(range(1,10))
print(a,type(a))
```

note:
=====

type conversion functions table
============================

| from | to | function |
|---|---|---|
| ==== | === | ======= |
| integer | float | float() |
| float | integer | int() |
| float | string | str() |
| string | integer | int() |
| string | float | float() |
| string | complex | complex() |
| complex | string | str() |
| integer | string | str() |
| float | complex | complex() |
| integer | complex | complex() |
| complex | integer | it is not able to convert |
| complex | float | it is not able to convert |
| integer | Boolean | bool() |
| float | Boolean | bool() |

| | | |
|---|---|---|
| complex | Boolean | bool() |
| Boolean | int | int() |
| Boolean | float | float() |
| Boolean | complex | complex() |

input() vs raw_input()
================

input() function is input function in python 3.0 version onwards

raw_input() is a input function in python 2.0 version onwards

input() function will always take any data in string format

raw_input() will always take any data, the data what format

we given

python output statement:
====================

when we write any program, after writing the program we will

execute the program, once program is executed, the result

of the any python program will be displayed via using

"output statement"

In python , to display any output of the program, we will use

a function called "print()"

print() is a built-in function in python 3.0 version onward

before python 3.0 version ,print act as a "statement" ,not like

a function in python 2.0 version

to use the print() function in python program , we need to use

the following syntax:
==============================================

print(data, sep="",end="")

or

print(data1,data2,...datan,sep=" ",end=" ")

here in this print() function:
=====================

data may be anything(it may be number, character, string,.......)

if the data is string or character, we need given in single or

double or triple quotes

sep in the print() function, is used to give the separator for

multiple outputs which are displaying using print() function

the default value for sep is "space or whitespace"(when we

not given any value for sep, then sep will take space or white

space)

end in the print() function, is used to give the delimeter or

symbol to represent where the output has to display on

same line or different line

the default value for end is "\n" or new line(when we

not given any value for end, then end will take \n)

write a python program to a message called "Hello World!"
============================================

code:
====

```
"""
author: Ram
program to display a message called "Hello World!"
"""

print('Hello World!')
```

write a python to display various outputs using print()

function:
==============================================

code:
====

```
"""
```

author:Ram
python to display various outputs using print()
function:
"""
print(10)
print(1.234)
print("hello")
print(10+5j)

5.python data types
==================================================

data type means "what type of data, the variable has"

in python, we will have the following data types:
======================================

1.numeric type:
=============

in python, we will have three different types of numbers:
=============================================

1.integer number:
===============

a number which is does not have "any decimal/fractional"

part, is called as "integer"

this number may be positive or negative

example: 12500,-578,..........

2.float/real number:
================

a number which is having "any decimal/fractional"

part, is called as "float or real number"

this number may be positive or negative

example: 125.00,-5.78,..........

3.complex number:
===============

a number is which is "combination of both real number and

imaginary number"

in python, we will use to represent the complex number, as

follows:

================================================

 a + bj <=== here we always has to use "j" as character only

note:
====

in python every number will considered as "object"

all integers numbers will considered as "integer" objects

all float/real number will considered as "float" objects

all complex numbers will considered as "complex" objects

all integer objects will have a class called  "int", where

"int" is a built-in class( the class which is given by python)

all float objects will have a class called "float" , where

"float" is a built-in class

all complex objects will have a class called "complex", where

"complex"  is a built-in class

when we want to check the any type  of the data, we will get

it’s class name as "type"

to know the type of the "any data" in python, we will use a

function called "type()" (it is a built-in function)

type() function will always return "class name" as result

  6.python variables:
==================================================

variable is nothing but "named memory location"

variable also called as "containers which may hold value(object)"

when we want to store any value in the memory, we will use

a concept called  "variable"

using variable only, we can store any value in the memory

when we store any value in python we will use the variable, the

value of the variable may changes entire program when we store any value into the variable, the variable will

get the following:
=====================================================

1.name of the vairable

2.value(object in python)

3.memory address of the object

how to create a variable in python:
==============================

variable_name = value(object)

in python, when we want to see the memory address of the

variable , we will use a function called "id()"

id() function will return "object memory address"

id() function is also a "built-in function"

in python, when two objects(values) will have same

memory ,then objects(values) are similar

when we store the two same values(objects) in the memory ,

the two same values will have same memory location, but may

have different names

write a python program to work with variable:
========================================

example-1:
==========

```
"""
author:Ram
python program to work with variable
"""
a=100
b=1.234
print(a,b,sep=",")
print(id(a))#address of the a
print(id(b))#address of the b
c=100
print(id(c))
d=b
print(id(d))
```

exmaple-2:
=========

```
"""
author:Ram
python program to work with variable
"""
#mutiple variables with multiple values
a,b,c=10,20,30
print(a,b,c,sep=",")
#mutiple variables with same value/data
a=b=c=100
print(a,b,c,sep=",")
#we can give muitple values with single variable
a=10,20,30,40,50,60,70,80,90
print(a)
print(type(a))
a=10
print(a,type(a))
b=20,
print(b,type(b))
```

exmaple-3:
=========

```
"""
author:Ram
python program to work with numeric data type
in Python
"""
#stroing a integer object
a=123
print(a,type(a))
#stroing a float object
b=1.234
print(b,type(b))
#stroing a complex number
c=10+5j
print(c,type(c))
```

2.chracter or text type:
==================

in python, any character type data is termed as "string"

in python, we will represent any string in the following formats:
=============================================

1.single quotes(’ ’)

2.double quotes(" ")

3.triple quotes(’’’ ’’’ or ’’""" """)

if any string is represented in "triple" quotes, then the string

is called as "doc string" (this string will be used for

documentation or comments very often python)

example:
=======

x=’h’ ===>String data( string object)

x="hello" ===>string data (String object)

x="'’hello world"'’ ===>string data (string object)

note:
====

in general , the triple quotes will used to represent the

"multi line text/string" in python

in python ,every string data is itself "string object" and it’s

class name is "str" in Python

example:
=======

```
"""
author: Ram
program to work with character type data
in python
"""
x="hello"
print(x,type(x))
x=’h’
print(x,type(x))
x="""this is the string is represented in multi line"""
print(x,type(x))
```

3.boolean type:
============

in python ,we will also  have Boolean data , those are True and

False

True in Numeric is "one"

False in Numeric is "Zero"

the both True and False also can be considered as "Boolean objects, those are represent a class called "bool" class

example:
=======

```
"""
author:Ram
program to work with booelan objects in Python
"""
x=True
print(x,type(x))
y=False
print(y,type(y))
```

4.sequence type:
=============

in python, we will use sequence type "to store the multiple

values under single name"

in python, we will have the following sequence type:
========================================

1.list

2.tuple

3.range()

note:
====

when we say any data is "sequence type", then it follows both

indexing and slicing

in python, list , tuple, range() and string follows "indexing and

slicing"

in python, both list and tuple are also called as "ordered

collection"

ordered collection means "the way we given data, in the same

way it stores the data in the memory"

list is also called as "mutable type" , on list we can able to

perform insert, update and delete operations

in python, we will have the following mutable objects:
==========================================

1.list

2.set

3.dictionary

tuple is also called "immutable type", on tuple we can not able

to perform any operation like insert/update /delete

in python, we will have the following immutable objects:
============================================

1.tuple

2.string

3.range()

4.frozenset

example:
=======

program to work with sequence type data:
=================================

code:
====

```
"""
author:Ram
program to work with sequence type data
"""
#list(list object)
l1=[1,2,3,4,5,6,7,8,9,10]
print(l1)
print(type(l1))#list object class name is "list"
#tuple (tuple object)
t1=(1,2,3,4,5,6,7)
print(t1)
print(type(t1))#tuple object class name is "tuple"
#range()(range object)
r1=range(1,10)
print(r1)
print(type(r1))#range object class name is "range"
```

5. set type :
==========

set type is used to "store a group of unique/distinct elements(values)"

set type will not allow duplicate values

to create the set in python, we will use "{}" (curly braces"

to create the "empty set" in python ,we will use "set()"

constructor or function

in python, empty curly braces {} <== is represented as

"dictionary"

in python, set is termed as "un-ordered collection", it means

the way we give the data, it may store or may not store

set type is "mutable" type(we can do insert/update/delete)

set type is also "immutable", when we convert the set into

frozen set using "frozenset()" function

frozenset will not allow (insert/update and delete)

example:
=======

program to work with set type:
=======================

```
"""
author:Ram
program to work with set type
"""
#creating the set object
s1={1,2,3,4,1,2,3,4}
print(s1)
print(type(s1))#set object class name is "Set"
s2=set()#to create the empty set,we use set() function
print(s2,type(s2))
#create the frozen set
s3=frozenset({1,2,3,4})
print(s3)
print(type(s3))#frozenset objec class name is "frozenset"
```

6. map type:
==========

map type is used to "store the data in the form of key and value

pair"

in python, map type is "dictionary"

dictionary is "ordered collection", the way we give, in the same

way it stores the data

dictionary is "mutable" (it allows insert/update/delete)

in dictionary,

keys are always "unique" (keys never be duplicate)

values are may be" duplicate"

to create the dictionary, we will use "{}"(curly braces)

syntax for dictionary:
=================

{key1:value1,key2:value2,key3:value3,......keyn:valuen}

in dictionary ,both key and value are separated by ":"

before ":" is "key" in dictionary

after ":" is "value" in dictionary

example:
=======

program to work with dictionary:
=========================

```
"""
author:Ram
program to work with dictionary
"""
#to create the dictionary object,we will use {}
d1={}
print(d1,type(d1))#dictionary object class name is "dict"
d2={1:2,3:4,5:6,7:8}#keys are 1,3,5,7,values: 2,4,6,8
print(d2)
print(type(d2))
d3={1:2,1:3,1:4}
print(d3)
print(type(d3))
```

7.binary type:
===========

in python, when we want to work with binary files, network

communication, low level data manipulations, python uses

"binary type"

in this, we will have the following :
============================

1.bytes ==>to create bytes, we will use prefix as "b"

2.bytearray ===>to create bytearray,we will use a function

called bytearray()

3.memoryview()===>to create the memoryview,we will use a

function called "memoryview()"

example:
=======

program to work with bytes, bytearray, memoryview():
=========================================

```
"""
author:Ram
program to work with bytes, bytearray, memoryview()
"""
#create the bytes object
data=b"hello"
print(data)
print(type(data))#byte object class name is "bytes"
#create the bytearray object
data=bytearray([1,2,3,4,5,6])
print(data)
print(type(data))
memory_view=memoryview(data)
print(memory_view)
print(type(memory_view))
```

8.None Type:
============

none type is "to represent where there is not value"

none is also termed as " empty value or null value or not a

value"

to this value, python uses a keyword called "None"

example:
=======

program to work with none type:
==========================

```
"""
author:Ram
program to work with None type
```

"""
a=None
print(a)
print(type(a))#none object class name is NoneType

,.................

python literals:
==============

python literals means "values"

the value what we given to the variable "is known as"  literals

in python, we will have the following literals:
=====================================

1.integer literals ex: 12,1234,-567,..........

2.floating-point or real numbers  ex: 1.234,5.678,.............

3.boolean literals ex: True and False

4. complex literals ex: 10+5j, -11j,.............

5.list literals ex: [1,2,3,4,5,6]

6.tuple literals ex: (1,2,3,4,5,6)

7.set literals ex: {1,2,3,4,5,6,7}

8.dictionary literal ex: {1:2,3:4,5:6,7:8}

9.range literal ex: range(1,10)

10.None literal ex: None

11.special literals(Special values in the python):
=======================================

1.binary literal(it means binary value)

2.octal literal (it means octal value)

3.hexadecimal literal(it means hexadecimal value)

in computers, we have 4 different types of numbers:
===========================================

1.binary number:
==============

radix/base of binary number: 2

no of digits: 2

digits:0,1

minimum digit: 0

maximum digit: 1(radix-1/base-1)

these are dedicated in the computer science, for machine under

standing purpose

to store any binary number in Python, we will use "0b" or "0B"

as prefix to the number

2.octal number :
==============
radix/base of octal number: 8

no of digits: 8

digits:0,1,2,3,4,5,6,7

minimum digit: 0

maximum digit: 7(radix-1/base-1)

these are dedicated in the computer science, for representing

memory addresses in computers

to store any octal number in Python, we will use "0o" or "0O"

as prefix to the number

3.decimal number :
================
radix/base of decimal number: 10

no of digits: 10

digits:0,1,2,3,4,5,6,7,8,9

minimum digit: 0

maximum digit: 9(radix-1/base-1)

these are dedicated in the computer science, for users of the

computers

to store any decimal number in Python, we will use numbers

as it is, by default every integer number in python is decimal

4.hexadecimal number:
==================

radix/base of hexadecimal number: 16

no of digits: 18

digits:0,1,2,3,4,5,6,7,8,9,A(10),B(11),C(12),D(13),E(14),F(15)

minimum digit: 0

maximum digit: F(15)(radix-1/base-1)

these are dedicated in the computer science, for representing

memory addresses in computers

to store any hexadecimal number in Python, we will use "0x" or

"0X" as prefix to the number

in python , to represent binary number we will use "0b or 0B"

or bin() function(it is a built-in function)"

in python , to represent octal number we will use "0o or 0O"

or oct() function(it is a built-in function)"

in python , to represent hexadecimal number we will use "0x or

0X" or hex() function(it is a built-in function)"

program to work with binary,octal and hexadecimal numbers

in python:

```
"""
author:Ram
program to work with binary,octal and hexadecimal numbers
in python
"""
a=0b1010 #here we store the binary value using prefix
print(a)#when we print,it will display the value in decimal
a=bin(0b1010)#here we store the binary data using bin()
print(a)
print(type(a))
a=0o127 #here we store the octal value using prefix
print(a)#when we print,it display the value in decimal
a=oct(0o127)#here we store the octal data using oct()
print(a)
print(type(a))
```

a=0xabc #here we store the hexadecimal  value using prefix
print(a)#when we print,it display the value in decimal
a=hex(0xabc)#here we store the octal data using oct()
print(a)
print(type(a))

7.python operators:
=================

operator means "symbol" and which is used to perform a

specific operation on "operands"

operand means "on which we will perform operation, genereally

operand means data"

example:
=======

a=10,b=20

a+b ===>30 ===>here a,b are operends and + is operator

expression in python:
=================

expression means "combination both operators and operands"

example:
=======
c=a+b ,here + is addition and = is assignment

in python, we will have the following operators:
=====================================

1. Arithmetic operator:
==================

this operator is used in python to perform "all arithmetic

operations"

in python, we will the following arithmetic operators:
=========================================

| operator | meaning | example |
|---|---|---|
| + | addition | a=10,b=20 ===>a+b ===>30 |
| - | subtraction | a=10,b=20 ===>a-b ===>-10 |
| * | multiplication | a=10,b=20 ===>a*b ===>200 |

/ real/True division a=10,b=20 ==>a/b===>0.5

// floor division a=10,b=20 ===>a/b ==>0

% modulo division a=10,b=20 ===> a%b===>10

** power/exponent a=10,b=2 ===>a**b===>100

note:
====

in python,

"/" performs "real/true" division, after division it will give the

result as it is and it always real number(float number)

"//" performs "floor" division, after division it will give the

result as "integer number" as quotient

"%" will always gives remainder

if n1%n2(n1<n2),then the result is "n1"

if n1%n2(n1>n2),then the result is "actual remainder"

example:
=======

5%16 ==>5

16%5 ===>1

write a python program to work with "arithmetic operators"
==============================================

code:
====

```
"""
author:Ram
program to work with arithmetic operators
"""
a=10
b=20
print("addition:",a+b)
print("subtraction:",a-b)
print("multiplication:",a*b)
print("real division:",a/b)
print("real division:",4/2)
print("floor division:",a//b)
print("modulo division:",a%b)
print("exponenet:",a**b)
```

2.write a python program to perform arithmetic operations

on the given two numbers as user input:
==================================================

code:
=====

```
"""
author:Ram
program to work with arithmetic operators
"""
a=int(input("enter the value for a:"))
b=int(input("enter the value for b:"))
print(a,b)
print(type(a),type(b))
print("addition:",a+b)
print("subtraction:",a-b)
print("multiplication:",a*b)
print("real division:",a/b)
print("floor division:",a//b)
print("modulo division:",a%b)
print("exponenet:",a**b)
```

2. Relational Operator:
==================

relational operator "is used to compare any two values"

in python, the relational operator will give the result in

Boolean (True or False)

relational operator is also called as "Boolean operator"

in python, we will have the following relational operator:
==============================================

> => greater than

< => less than

>= => greter than or equal to

<= => less than or equal to

== => equal to

!= => not equal to

program to work with relational operator:
=================================

```
"""
author:Ram
Program to work with relational operator
"""
a=int(input("enter the value for a:"))#10
b=int(input("enter the value for b:"))#12
print("a>b",a>b)#False
print("a<b",a<b)#True
print("a>=b",a>=b)#False
print("a<=b",a<=b)#True
print("a==b",a==b)#False
print("a!=b",a!=b)#True
```

3.Logical Operator:
===============

logical operator s used in python, "to compare any two

conditions"

logical operator will give the result in "Boolean"

logical operator is also called as "Boolean operator"

in python, we will have the following logical operators:
==========================================

1.logical OR:
==========

symbol: or (it is a keyword in python)

in the case of logical or, if any one condition is True, then the

entire result is True

in python ,logical or is short-circuit operator

in python, when we are working with logical or on multiple

conditions, while checking the conditions, if any one condition

is True, then logical or will simply return result as "True", it

never check the remaining conditions after that condition,due

to it is a short-circuit operator

example:
=======

a=10,b=20 ===>(a>b) or (a<b) ===>False or True ===>True a=10,b=20===>(a<b) or (a>b) ===>True

2.logical AND:
===========

symbol: and (it is a keyword in python)

in the case of logical and, if any one condition is False, then the

entire result is False

in python ,logical and is short-circuit operator

in python, when we are working with logical and on multiple

conditions, while checking the conditions, if any one condition

is false, then logical and will simply return result as "False", it

never check the remaining conditions after that condition, due

to it is a short-circuit operator

3.logical NOT:
===========

this will make the result of any condition as follows:
=========================================

if the result of the condition is "True", then logical not will make

"False"

if the result of the condition is "False", then logical not will

make  "True"

program to work with logical operators:
===============================

code:
====

```
"""
author:Ram
Program to work with logical operators
"""
a=int(input("enter the value for a:"))#10
b=int(input("enter the value for b:"))#12
#logical or
print((a>b) or (a<b))#True
#logical and
```

```
print((a>b) and (a<b))#False
#logical not
print(not(a>b))#True
```

example:
=======

```
"""
author:Ram
Program to work with logical operators
"""
x=int(input("a:")) or int(input("b:"))
print(x)
x=int(input("a:")) and int(input("b:"))
print(x)
```

4.Assignment Operator:
==================

this operator is used to "assign a value to variable"

symbol: =

example:
=======

```
a=10 <=== here we assign a value 10 to the a

b=1.234 <=== here we assign a value 1.234 to the b
```

in python, using assignment operator, we can able to perform

"Compound operation" (doing more than one operation at a

time)

example:
=======

let assume a=100

```
a+=10 ====>a=a+10 ===>a=100+10=110

a-=10 ====>a=a-10 ====>a=110-10=100

a*=10 ====>a=a*10 ===>a=a*10=100*10=1000

a%=10 ===>a=a%10 ===>a=1000%10=0
```

program to work with compound operations:
==================================

```
"""
author:Ram
program to work with compound operations
```

(using assignment operator)
"""
a=int(input("enter the value for a:"))
print(a)
a+=10
print(a)
a-=10
print(a)
a*=10
print(a)
a%=10
print(a)

5.Bitwise Operator:
===============

these operators are used to perform operation of binary

data of the given data

when we give the any data to "bitwise operator", first it will

convert the given data into binary, on that it will perform the

operation, when it giving result, again it will convert the

binary value into decimal value

in python, we will have the following bitwise operators:
===========================================

1.bitwise or:
==========

symbol: |(pipe)

rule:
====

in the case of bitwise or, if any one input is "1",then the entire

result is "1"

example:
=======

a=10===>01010

b=20===>10100
==============
a|b=====>11110===>30

a=23

b=43

a|b ===>63

2.bitwise and:
===========

symbol: &(ampracend)

rule:
====

in the case of bitwise and, if any one input is "0",then the entire

result is "zero"

example:
=======

a=10===>01010

b=20===>10100
==============
a&b====>00000===>0

a=23

b=43

a|b ===>3

3.bitwise exclusive or:
=================

symbol: ^(caret)

rule:
====

in the case of bitwise exclusive or, if both inputs are same, the

result is zero, otherwise result is "one"

example:
=======

a=10===>01010

b=20===>10100
==============
a^b====>11110===>30

a=23

b=43

a^b ===>60

4.bitwise left shift:
===============

symbol: <<

formula: n<<s =n* 2 power s

example:
=======

10<<2 ===>10*2power 2==>10*4=40

5.bitwise right shift:
================

symbol: >>

formula: n>>s =n//2 power s

example:
=======

10>>2 ===>10//2power 2==>10//4=2

6.bitwise one’s complement :
======================

symbol: ~(tlide)

formula:  ~n=-(n+1)

example:
=======

~10 ===>  -(10+1) ===>-11

program to work with bitwise operators:
===============================

code:
====

```
"""
author:Ram
program to work with bitwise operators
"""
a=int(input("enter the value for a:"))#12
b=int(input("enter the value for b:"))#16
```

```
#bitwise or
print("a|b:",a|b)#28
#bitwise and
print("a&b:",a&b)#0
#bitwsie exclusive or
print("a^b:",a^b)#28
#bitwise left shift
print("a<<2",a<<2)#48
#bitwise right shift
print("a>>2:",a>>2)#3
```

6.Membership Operator:
===================

this operator is used in " python, to check the presence of the

given value in the given iterable"

this operator will give the result either "True or False"

this operator will be used on the "iterables" in python

in python, iterables are "list, tuple, string, range(), set,

dictionar ..........."

the following membership operators in Python:
=====================================

1. in (it check the value is present or not)

2. not in (it checks the given value is not present or not)

example-1:
=========

```
"""
author:Ram
program to work with membership operators
"""
x="123"
print('5' in x)
print("1" in x)
print("hello" in "hello world")
print(" " in "hello world")
print("Hello" in "hello world")
print(2 in [1,2,3,4,5,6,7,8,9,10,"hello"])
print(4 not in (1,2,3,4,5,6,7,8,9,10))
print(10 in range(1,11))
```

7.Identity Operator:

================

in python,  we will use this operator "two compare given two

values memory locations are identical or not "

this operator will never compare "values", but compare values

memory locations

this operator will return "result" as Boolean

in python, we will have the following identity operators:
==========================================

1.is  (this will compare two values memory location are same

or not/similar or not /equal or not)

2.is not(this will compare two values memory location are not

same or not)

example:
=======

```
"""
author:Ram
program to work with identity operators
"""
a=10#here a value is 10
print(id(a))#adress of a
b=20#here b value is 20
print(id(b))#address of b
c=a #here c value is a(a value is 10),so c is 10
print(id(c))#address of c
d=b#here d value is b(b value is 20), so b value is 20
print(id(d))#address of d
h1="hello"#here h1 value is "hello
print(id(h1))#address of h1
print(a is b)#False
print(a is not b)#True
print(a is not c)#False
print(b is not a)#True
print(d is not c)#True
```

note:
====

a is b ===>       id(a)==id(b)

a is not b ===> id(a)!=id(b)

8.Conditional Operator:
===================

this operator will perform operation based on the condition

syntax:
======

expression1 if condition else expression 2

if the condition is True, then the before "if" what we write is

executed

if the condition is False, then the after "else" what we write is

executed

example:
========

```
"""
author:Ram
program to work with conditional operator
"""
x,y=10,20
print(x) if x>y else print(y)# 20
print(x) if x<y else print(y)# 10
result=x if x>y else y
print(result)#20
result=x if x!=y else y
print(result)#10
```

9.Warlus Operator

8.flow control statements:
=================================================

these statements are used in python to execute any logic,

based on certain condition

in python, we have two types of flow-control statements:
===========================================

flow control statements with "conditions"(conditional):
========================================

simple if statement:
================

when we have "only one condition and based on the condition if we need to execute any logic, then in python we will use

"simple if statement or if statement"

the logic what we write "under the if" will executed only the

condition is "True", otherwise the logic whatever we write under

the if will not executed( when condition is False)

syntax for simple if or if statement:
=============================

```
if condition:

    #logic
```

note:
=====

when we writing any code in the "if", every line starting we need

to take some space ,that space we are called as "indentation"

example:
=======

```
a=int(input("enter the value for a:"))
b=int(input("enter the value for b:"))
if a>b:
    #part of the if-block
    print("the value of a is greter than b")
```

example-2:
=========

```
if True:#always executed
    print("this is first if")
if 10+10:
    print("this is 10+10")
if False:
    print("this is not executed")
if None:
    print("this is not executed")
if '':
    print("this is not executed")
if []:
    print("this is not executed")
if ():
    print("This is not executed")
if -100:
    print("this is executed")
```

how to create the empty if statement in python:
=======================================

empty if statement in python means "if with no code or

no implementation"

in order to create the "empty if statement", in python we will

use a keyword called "pass"

pass is " a placeholder , which represents there is no code"

example:
=======

```
if True:#always executed
  pass
if 10+10:
    pass
```

example:
========

```
if True:print("this is if statment")
```

note:
====

the above format "we will write in python, when we have

only one statement for if"

if-else statement:
=============

when we have two condtions and we need to execute any

based on condition( either True or false), in python we will

use "if-else" statement

syntax for if-else in python:
======================

```
if codition:

    #logic

else:

    #logic
```

when the condition what we take for "if" is True, then the logic what we write under "if" will executed

else will executed only when the condition what we taken for

if has to be False  and else is the counter part of the "If"

in python,

we can write write the "if" without else

we can not write "else" alone

## example:

```
a=int(input("enter the value for a:"))
b=int(input("enter the value for b:"))
if a>b:
    print("a>b")
else:
    print("a<b")
```

## else if ladder:

when we have three or more than three conditions, in python

we will use "else-if ladder"

## syntax for else-if ladder:

```
if condition:

     #logic

elif codition:

    #logic

elif condition:

    #logic

elif condition:

    #logic

.
.
.
else:
 #logic
```

note:
====

else -if ladder always starts with "if"

else-if ladder may ends with "else"

example:
=======

```
a=int(input("enter the value for a:"))
b=int(input("enter the value for b:"))
c=int(input("enter the value for c:"))
if a>b and a>c:
    print("max:a")
elif b>c:
    print("max:b")
else:
    print("max:c")
```

nested conditional statements:
=========================

writing the conditional statements inside "another" conditional

statements

syntax:
======

```
 if condition:

        if condition:

            #logic

       else:
          #logic
else:

       if condition:

           #logic

       else:

          #logic
```

example:
=======

```
age=int(input("enter the age:"))
if age>=21:
    if age>=21 and age<=30:
```

```
        print("enjoy your bachelor life")
    elif age>=30 and age<=40:
        print("right age,check with uncles")
    elif age>=40 and age<=60:
        print("already you are in heaven,do not think too much")
    elif age>=60 and age<=80:
        print("you already ready to hit the bucket")
    else:
        print("RIP!")
else:
    print("hey lilliput,this is not video game")
```

example-2:
=========

```
a,b,c=10,20,30
if a>=10:
    a+=90
    if a>10 and b>=20:
        a+=110
        if a>150 and b>20:
            a+=300
        else:
            b+=300
    else:
         a+=500
print(a,b)
```

match statement :
==============

match statement in python is same as "switch in c/java"

using this statement, we can able to select one option or case

based on the given match value

syntax:

```
match  value:

  case  value1:  code

  case  value2:  code

  case  value3: code

  .
  .
  .
  case _: code
```

note:
====

in match statement, case _ is called as "default" case and this

will be executed only "when if match value is not matches with

any case value"

example:
=======

```
#program to work with match statement
value=int(input("enter the value:"))
match value:
    case 1:print("this is case-1")
    case 2:print("this is case-2")
    case 3:print("this is case-3")
    case _:print("Invalid Option") #<== default case
```

flow control statements without "conditions"(un-conditional)
=================================================

break (we can write only with looping statements):
=========================================

break is a un-conditional statement in python

break never take any condition like if or elif

break can used only in the loop statement (either in while

or for loop)

break used to "terminate the loop execution"

example:
=======

```
a=1
while a<=10:
    if a==5:
        break
    print(a)#1 2 3 4
    a+=1
```

continue  (we can write only with looping statements):
============================================

continue is a un-conditional statement

continue is used  only with loop

continue "skip only the current iteration"

example:
========

```
a=1
while a<=10:
    if a==5:
        a+=1
        continue
    print(a)
    a+=1
```

return (We can write only with functions)

9.looping statements/iterative statements
==================================================

looping statements are also called as "iterative statements"

these statements are used in python "to execute a logic for

"n" number of times until the condition become false"

in python we will have following looping statements:
==========================================

1. while loop(this loop will work only with conditions in python):
==================================================

in Python, when we want to work with any "while" loop, we need

condition or any while is created only with condition in python

in python, while loop is called "conditional loop"

syntax:
======

```
while condtion:

   #logic
```

note:
====

in general, loops are two types:
========================

finite loop:
=========

this loop will executed for a specific number of times only

infinite loop:
==========

this loop will executed for infinite number of times

when the  loop is infinite loop, we will never get any output,

from loop, in the program we need avoid the infinite loops

example:
=======

```
#program on while loop
a=1
while a<=10:
    print(a)
    a+=1
```

example-2:
=========

```
#program on while loop
a=1
while a<=10:
    print(a)#1 6
    a+=5 #a=11
```

example-3:
=========

```
#program on while loop
a=1
while a<=10:
    print(a)
    if i%2==0:
        a+=1
```

ouput:

error (i is not defined)

example-4:
=========

```
#program on while loop
a=1
while a<=10:
    print(a)
    if a%2==0:
        a+=1
```

example-5:
=========

```
#program on while loop
a=1
while a<=10:
    print(a)
    if a%2==0:
```

```
        a+=1
    else:
        a+=4
```

inner loops or nested loops in python using while loop:
=============================================

we can create the a loop inside another loop ,this is known as

"inner loop or nested loop"

syntax:
======

```
while condition:

         while condition:

                  while condition:
```

note:
====

when we create any inner loop or nested loop inside another

loop(main loop), the main loop will wait until inner loop or

nested loop until it stop the execution

example-1:
=========

```
a=1
b=1
times=0
while a<=5:
    while b<=5:
        b+=1
        times+=1
    a+=1
print(times)#
```

example-2:
=========

```
a,b,times=1,1,0
while a<=5:
    while b<=5:
        if b==3:
            break
        b+=1
        times+=1
    a+=1
    b=1
print(times)
```

example-3:
=========

```
a,b,c,times=1,1,1,0
while a<=5:
    while b<=5:
        while c<=5:
            if c==3:
                break
            c+=1
            times+=1
        c=1
        b+=1

    a+=1
    b=1
    c=1
print(times)
```

example-4:
=========

```
a,b,c,d,times=1,1,1,0,1
while a<=5:
    while b<=5:
        while c<=5:
            while d<=5:
                if d==4:
                    break
                times+=1
            d=1
            if c==3:
                break
            c+=1
        b+=1
        if b==3:
            break
    b,c,d=1,1,1
    if a==3:
        break
    a+=1
```

example-5:
=========

```
a,b,times=1,1,0
while a<=10:
    while b<=5:
        if b==5:
            b+=1
            times+=1
            break
        else:
            b+=1
            times+=1
            continue
```

```
    if a%2==0:
        a+=3
        b=1
    else:
        a+=1
        b=1
print(times)
```

example:
=======

```
a,b,c,times=1,1,1,0
while a<=10:
    while b<=10:
        while c<=10:
            if c%2==0:
                c+=1
                times+=1
            else:
                c+=1
                times+=1
        c=1
        b+=1
        if b%5==0:
            break
    if a%2==0:
        a+=2
        b,c=1,1
    else:
        a+=1
        b,c=1
print(times)
```

write a python program , multiply the given number with

2 without using "*,+,-" and any built-in function:
==============================================

input:  20

output: 40

code:
=====

```
number=int(input("Enter the number:"))
print("result",number<<1)
```

3.write a python program find out the remainder of the

given two number without using "%" and any built-in

function:
================================================

input:

=====

5 10 ===>5

10 5 ===>0

24  12 ===>0

code:
=====

```
num1=int(input("Enter the num1:"))
num2=int(input("enter the num2:"))
if num1<num2:
    print(num1)
elif num1>num2:
    print(num1-(num2*(num1//num2)))
else:
    print(0)
```

2. for loop(this loop will work only with iterables in Python):
==============================================

for loop is basically work with iterables in python

in python, iterables are list, tuple, set,  string, dictionary,

range(),...............

for loop is not a "conditional based loop"

when we create any for loop, we will never allowed to use

conditions

syntax for for-loop:
===============

```
for i in iterable_name:

      #logic
```

working with range():
=================

range() is a "built-in function in python"

using range() function, we can able to generate the values

for given start, end values

in general , range() function will give values ranges from

start to end-1

syntax:
======

range(start, end, step)

note:
====

default value for start is "zero"

default value for step is "one"

start can be "+ve or -ve or zero"

end can be "+ve or -ve or zero"

step can be "+ve or -ve or not zero"

in range, we never generally give "start and end both are same"

in range, we never give start more than end for +ve step

in range, to print the values in the descending order, we will

use "start more than end , step should be -ve"

in range() function, every value has to be "integer", it does not

support float values

example:
=======

range(1,10) ====>1,2,3,4,5,6,7,8,9

range(1,11) ===>1,2,3,4,5,6,7,8,9,10

example:
=======

```
for j in range(1,10):
    print(j)
for j in range(10):
    print(j)
for j in range(1,10,4):
    print(j)
for j in range(10,1,-1):
    print(j)
a1=range(1,10)
print(a1)
```

print(type(a1))

example:
========

```
for i in range(1,10):
    print(i)
    i=i+10
for i in range(1,10):
    i+=10#i=1,i=11
    print(i)
for i in range(1,10):
    if i==5:
        break
    print(i)
for i in range(1,10):
    if i==5:
        continue
    print(i)
```

indexing and slicing on range():
==========================

indexing with range():
=================

in python, we will have two types of indexing:
=====================================

1.forward indexing or positive indexing:
=================================

this indexing will start from o to n-1

this indexing will start from "left to right" or "start to end"

where n is "number of the values in the range"

2.backword indexing or negative indexing:
==================================

this indexing will start from -1 to -n

this indexing will start from "right to left" or "end to start"

where n is "number of the values in the range"

note:
====

to find the number of values in the range(), we will use a

function called "len()"

example:
=======

```
a1=range(1,10)#1 to 9
print(len(a1))#9
a1=range(1,20,5)
print(len(a1))
a1=range(20,1,-4)
print(len(a1))
```

to get the any element from the range(), using index,  we will

use the following syntax:
==================================================

range_object_name[index]

example:
=======

```
a1=range(1,11)#1,2,3,4,5,6,7,8,9,10
print(a1[0])#1
print(a1[7])#8
print(a1[9])#10
print(a1[3])#4
print(a1[1])#2
print(a1[-1])#10
print(a1[-5])#6
print(a1[-10])#1
print(a1[-3])#8
```

print the elements in the range() using while loop via indexing:
==================================================

code:
=====

```
a1=range(10,110,10)#10,20,30,40,50,60,70,80,90,100
index=0
while index<len(a1):
    print(a1[index])
    index+=1
```

slicing with range() function:
======================

slicing will give "zero or more elements"

indexing will give "only one element"

slicing result of the range() function  will always in "range()"

form only

syntax for slicing:
==============

range_obejct_name[start:end:step]

note:
=====

slice result always "from start index to end index -1"

example:
=======
```
a1=range(10,110,10)#10,20,30,40,50,60,70,80,90,100
print(a1[1:6])#range(20,70,10)
print(a1[2:9])#range(30,100,10)
print(a1[:9])#range(10,100,10)
print(a1[3:])#range(40,110,10)
```

example-2:
=========
```
a1=range(10,110,10)#10,20,30,40,50,60,70,80,90,100
print(a1[1::2])#range(20,110,20)
print(a1[::4])#range(10,110,40)
print(a1[3::4])#range(40,110,40)
for i in a1[::-1]:
    print(i)
for i in a1[::-4]:
    print(i)
for i in a1[-10::4]:
    print(i)
for i in a1[-1:-1:1]:
    print(i)
for i in a1[-4:-6:2]:
    print(i)
for i in a1[-4:-6:-2]:
    print(i)
```

programming with python:
=====================

1.write a python program compare the given three

  numbers which is maximum number, without using

  > ,<,>= and <= and not using any pre-define function
=============================================

```
#input
a,b,c=int(input("a:")),int(input("b:")),int(input("c:"))
sum=a+b+c
```

```
#logic
while sum!=0:
    if sum==a or sum==b or sum==c:
        print("max:",sum)
        break
    sum-=1
```

2.write a python program find the "highest common

factor/GCD/HCF" of the given two numbers:
=============================================

code:
====

```
#input
num1,num2=int(input("num1:")),int(input("num2:"))
fact=1
max=0
#logic
if num1>num2: num1=num2
while fact<=num1:
    if num1%fact==0 and num2%fact==0:
        if max<fact:
            max=fact

    fact+=1
print(max)
```

3.write a python program print the maximum even

number for the given range, without range() function:
==============================================

code:
====

```
#input
start,end=int(input("start:")),int(input("end:"))
max=0
if start>end:start,end=end,start
while start<=end:
    if start%2==0:
        if max<start:
            max=start
    start+=1
print(max)
```

or

```
#input
start,end=int(input("start:")),int(input("end:"))
if start>end:
    if start%2!=0:
        start=end-1
```

```
        print(start)
    else:
        print(start)
else:
    if end%2!=0:
        end=end-1
        print(end)
    else:
        print(end)
```

4.write a python program print the highest digit in the

given number:
=============================================

code:
====

```
#input
number=int(input("number:"))#1789
max1=0
for i in str(number):
    if max1<int(i):
        max1=int(i)
print(max1)
```

4.write a python  program  print the numbers for given range ,

each number must have 9 as digit at unit place or as starting

digits
=================================================

example:
========

start ===>10

end====>20

ouput:
=====

19

example:
=======

start ====>10

end====>100

output:
======

19,29,39,49,59,69,79,89,90,91,92,93,94,95,96,97,98,99

code:
=====

```
#input
start=int(input("start:"))
end=int(input("end:"))
if start>end: start,end=end,start
#logic
if start!=end and start>0 and end>0:
  for i in range(start,end+1):
    if str(i)[0]==’9’ or str(i)[-1]==’9’:
        print(i)
elif start==end and start>0 and end>0:
    if str(start)[0]==’9’ or str(start)[-1]==’9’:
        print(start)
else:
    print("both start and end are positive")
```

5.write a python program to print the numbers for the given

range, where the number will have all digits are same
==================================================

example:
=======

start: 10

end: 100

output:
======

11,22,33,44,55,66,77,88,99

code:
=====

```
#input
start=int(input("start:"))
end=int(input("end:"))
if start>end: start,end=end,start
#logic
if start!=end and start>0 and end>0:
    if start>=1 and end<=10:
        for i in range(start,end):
            print(i)
    if start>=10 and end<=100:
        for i in range(start,end+1):
            if i%11==0:
                print(i)
    else:
        for i in range(start,end):
```

```
            flag=str(i)[0]
            for j in str(i):
                if flag!=j:
                    break
            else:
                print(i)
```

```
else:
    print("both start and end are positive")
```

example:
=======

```
i=1
while i!=1:
    #print(i)
    i+=1
print("final value of the i:",i)
```

example:
=======

```
i=1
while i<=100:
    if i<50:
        i+=10
    else:
        i+=40
print("the finbal value of the i:",i)
```

example:
=======

```
i=1
j=1
while i<=100:
    while j<=10:
        if j>3:
            break
        i+=10
    i+=30
print("the final i value is ",i)
```

example:
=======

```
i=1
j=1
```

```
while i<=100:
    while j<=10:
        if j>3:
            break
        i+=10
        j+=1

    i+=30
print("the final i value is ",i)
```

problem solving with python:
=======================

problems on digits:
===============

1.print the digits of the given number:
===============================

code:
=====

```
#print the digits of the given number
#take the input from the user
number=int(input("enter the number:"))
if number<0:number=-number
for i in str(number):
    print(i,end=",")
```

2.print the even digits in the given number:
==================================

code:
====

```
#print the even digits of the given number
#take the input from the user
number=int(input("enter the number:"))
if number<0:number=-number
for i in str(number):
    if int(i)%2==0:
       print(i,end=",")
```

3.print the odd digits in the given number:
=================================

code:
====

```
#print the odd digits of the given number
#take the input from the user
number=int(input("enter the number:"))
if number<0:number=-number
for i in str(number):
```

```
    if int(i)%2!=0:
        print(i,end=",")
```

4.print the prime digits in the given number:
====================================

code:
====

```
#print the prime digits of the given number
#take the input from the user
number=int(input("enter the number:"))
if number<0:number=-number
for i in str(number):
    if i in "2357":
        print(i,end=",")
```

5.print the perfect digits in the given number:
=====================================

code:
====

```
#print the prime digits of the given number
#take the input from the user
number=int(input("enter the number:"))
if number<0:number=-number
for i in str(number):
    if i in "16":
        print(i,end=",")
```

6.print the sum of the digits in the given number:
========================================

code:
====

```
#print the sum of digits of the given number
#take the input from the user
number=int(input("enter the number:"))
sum_of_digits=0
if number<0:number=-number
for i in str(number):
   sum_of_digits+=int(i)
print("sum of the digits:",sum_of_digits)
```

7.print the sum of the even digits in the given number:
============================================
code:
====

```
#print the sum of even digits of the given number
#take the input from the user
number=int(input("enter the number:"))
sum_of_digits=0
if number<0:number=-number
```

```
for i in str(number):
    if int(i)%2==0:
      sum_of_digits+=int(i)
print("sum of the digits:",sum_of_digits)
```

8.print the sum of the odd digits in the given number:
==========================================

code:
====

```
#print the sum of odd digits of the given number
#take the input from the user
number=int(input("enter the number:"))
sum_of_digits=0
if number<0:number=-number
for i in str(number):
    if int(i)%2!=0:
      sum_of_digits+=int(i)
print("sum of the digits:",sum_of_digits)
```

9.print the sum of the prime digits in the given number:
============================================

```
#print the sum of prime digits of the given number
#take the input from the user
number=int(input("enter the number:"))
sum_of_digits=0
if number<0:number=-number
for i in str(number):
    if i in "2357":
      sum_of_digits+=int(i)
print("sum of the digits:",sum_of_digits)
```

10.print the sum of the perfect digits in the given number:
==============================================

code:
====

```
#print the sum of perfect digits of the given number
#take the input from the user
number=int(input("enter the number:"))
sum_of_digits=0
if number<0:number=-number
for i in str(number):
    if i in "16":
      sum_of_digits+=int(i)
print("sum of the digits:",sum_of_digits)
```

11.print the sum of the alternate digits in the given number:
================================================

code:
====

```
#print the sum of alternate digits of the given number
#take the input from the user
number=int(input("enter the number:"))
sum_of_digits=0
pos=1
if number<0:number=-number
for i in str(number):
    if pos%2!=0:
      sum_of_digits+=int(i)
    pos+=1
print("sum of the digits:",sum_of_digits)
```

12.print the maximum  digit in the given number:
=========================================

13.print the maximum  even digit in the given number:
============================================

14.print the maximum  odd digit in the given number:
============================================

15.print the maximum  prime digit in the given number:
=============================================

16.print the maximum  perfect digit in the given number:
==============================================

17.print the minimum digit in the given number:
=======================================

code:
=====

```
#print the minimum digit of the given number
#take the input from the user
number=int(input("enter the number:"))
min=9
if number<0:number=-number
for i in str(number):
    if min>int(i):
        min=int(i)
print("minimum digit:",min)
```

18.print the minimum even digit in the given number:
===========================================

code:
=====

```
#print the minimum even digit of the given number
```

```
#take the input from the user
number=int(input("enter the number:"))
min=9
if number<0:number=-number
for i in str(number):
  if i in "02468":
    if min>int(i):
        min=int(i)
print("even minimum digit:",min)
```

19.print the minimum odd digit in the given number:
==========================================

code:
====

```
#print the minimum odd digit of the given number
#take the input from the user
number=int(input("enter the number:"))
min=9
if number<0:number=-number
for i in str(number):
  if i in "13579":
    if min>int(i):
        min=int(i)
print("odd minimum digit:", min)
```

20.print the minimum prime digit in the given number:
============================================

code:
====

```
#print the minimum prime digit of the given number
#take the input from the user
number=int(input("enter the number:"))
min=9
if number<0:number=-number
for i in str(number):
  if i in "2357":
    if min>int(i):
        min=int(i)
print("prime minimum digit:",min)
```

21.print the minimum perfect digit in the given number:
=============================================

code:
====

```
#print the minimum perfect digit of the given number
#take the input from the user
number=int(input("enter the number:"))
min=9
```

```
if number<0:number=-number
for i in str(number):
  if i in "16":
    if min>int(i):
        min=int(i)
print("perfect minimum digit:",min)
```

22. count number of digits in the given number:
=======================================

code:
====

```
#print the count number of digits given number
#take the input from the user
number=int(input("enter the number:"))
count=0
if number<0:number=-number
for i in str(number):
  count+=1
print("number of digits in the given number:",count)
```

or

```
#print the count number of digits given number
#take the input from the user
number=int(input("enter the number:"))
if number<0:number=-number
print("number of digits in the given number:",len(str(number)))
```

23. count number of even digits in the given number:
===========================================

code:
====

```
#print the count number of even digits given number
#take the input from the user
number=int(input("enter the number:"))
count=0
if number<0:number=-number
for i in str(number):
 if int(i)%2==0:
  count+=1
print("number of even digits in the given number:",count)
```

24. count number of odd digits in the given number:
==========================================

code:
====

```
#print the count number of odd digits given number
#take the input from the user
number=int(input("enter the number:"))
count=0
if number<0:number=-number
for i in str(number):
 if int(i)%2!=0:
  count+=1
print("number of odd digits in the given number:",count)
```

25. count number of prime digits in the given number:
==========================================

code:
====

```
#print the count number of prime digits given number
#take the input from the user
number=int(input("enter the number:"))
count=0
if number<0:number=-number
for i in str(number):
 if i in "2357":
  count+=1
print("number of prime digits in the given number:",count)
```

26. count number of perfect digits in the given number:
============================================

code:
====

```
#print the count number of perfect digits given number
#take the input from the user
number=int(input("enter the number:"))
count=0
if number<0:number=-number
for i in str(number):
 if i in "16":
  count+=1
print("number of perfect digits in the given number:", count)
```

27. find average of digits in the given number:
====================================

code:
=====

```
#print the average of the digits of the given number
#take the input from the user
number=int(input("enter the number:"))
sum=0
count=0
```

```
if number<0:number=-number
for i in str(number):
  sum+=int(i)
  count+=1
print("number of even digits in the given number:",sum/count)
```

28. find average of even digits in the given number:
==========================================

code:
====

```
#print the average of the even digits of the given number
#take the input from the user
number=int(input("enter the number:"))
sum=0
count=0
if number<0:number=-number
for i in str(number):
 if int(i)%2==0:
    sum+=int(i)
    count+=1
print(" average of the even digits of the  given   number:",sum/count)
```

29. find average of odd digits in the given number:
=========================================

code:
====

```
#print the average of the odd digits of the given number
#take the input from the user
number=int(input("enter the number:"))
sum=0
count=0
if number<0:number=-number
for i in str(number):
 if int(i)%2!=0:
    sum+=int(i)
    count+=1
print(" average of the odd digits of the  given   number:",sum/count)
```

30. find average of prime digits in the given number:
==========================================

```
#print the average of the prime digits of the given number
#take the input from the user
number=int(input("enter the number:"))
sum=0
count=0
if number<0:number=-number
```

```
for i in str(number):
 if i in "2357":
    sum+=int(i)
    count+=1
print(" average of the prime digits of the  given   number:",sum/count)
```

31. remove the duplicate digits in the given number:
==================================================

code:
====

```
#remove the duplicate digits in the given number
#take the input from the user
number=int(input("enter the number:"))#1123
res=’’
if number<0:number=-number
if number==0:
    print(number)
else:
    for i in str(number):#1123
        if i not in res:
            res+=i #res=123
    print(res)
```

32. find each digit is how many times repeated in the given number(excluding the duplicates):
================================================

code:
====

```
#print the each digit count in the given number
#take the input from the user
number=int(input("enter the number:"))#1123
res=’’
if number<0:number=-number
if number==0:
    print(number)
else:
    for i in str(number):#1123
        if i not in res:
            res+=i #res=123
    count=0
    for i in str(res):
        for j in str(number):
            if i==j:
                count+=1
        print(i," count:",count)
        count=0
```

33. check given digit is present in the given number or not:
==================================================

code:
====

```
#print the each digit count in the given number
#take the input from the user
number=int(input("enter the number:"))#1123
digit=int(input("enter the digit:"))
if number<0:number=-number
elif number==0: number=0
if str(digit) in str(number):
    print("Found")
else:
    print("Not Found")
```

34.print the middle digit of the given number:
=======================================

code:
====

```
#print the each digit count in the given number
#take the input from the user
number=int(input("enter the number:"))#1123
if number<0:number=-number
elif number!=0:
    length=len(str(number))
    if length%2!=0:
        print(str(number)[length//2])
    else:
        print((int(str(number)[(length//2)-1])+int(str(number)[(length//2)]))//2)
```

35. print the digits which are greater than 5 in the given

number:
============================================

36. print the digits which are less than 5 in the given

number:
==================================================

problems on factors:
=================

1.print the factors of the given number:
================================

code:
====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
if number<0: number=-number
while fact<=number:
    if number%fact==0:
        print(fact)
    fact+=1
```

or

```
#take the nummber
number=int(input("enter the number:"))
fact=1
if number<0: number=-number
for fact in range(1,number+1):
    if number%fact==0:
        print(fact)
    fact+=1
```

2.print the all even factors of the given number:
==============================================

code:
====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
if number<0: number=-number
for fact in range(1,number+1):
    if number%fact==0 and fact%2==0:
        print(fact)
    fact+=1
```

3.print the all odd factors of the given number:
=======================================

code:
====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
```

```
if number<0: number=-number
for fact in range(1,number+1):
    if number%fact==0 and fact%2!=0:
        print(fact)
    fact+=1
```

4.count the number of the factors of the given number:
==========================================

code:
=====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
count=0
if number<0: number=-number
for fact in range(1,number+1):
    if number%fact==0:
       count+=1
    fact+=1
print("the number of factors:",count)
```

5.count the number of the even factors of the given number:
===============================================

code:
====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
count=0
if number<0: number=-number
for fact in range(1,number+1):
    if number%fact==0 and fact%2==0:
       count+=1
    fact+=1
print("the number of even factors:",count)
```

6.count the number of the odd factors of the given number :
================================================

code:
=====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
count=0
if number<0: number=-number
for fact in range(1,number+1):
    if number%fact==0 and fact%2!=0:
       count+=1
```

```
    fact+=1
print("the number of odd factors:",count)
```

7.check the given number is prime or not
==================================

code:
====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
count=0
if number<0: number=-number
for fact in range(1,number+1):
    if number%fact==0:
       count+=1
    fact+=1
if count==2:print("prime")
else:print("not a prime")
```

8.check the given number is perfect or not :
====================================

code:
====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
sum=0
if number<0: number=-number
if number>1:
  for fact in range(1,number):
    if number%fact==0:
       sum+=fact
    fact+=1
else: sum=1
print("perfect") if sum==number and number!=0 else print("not perfect")
```

9.print the prime factors of the given number :
======================================

TCM:
====

| input: | exp .o/p | actual. o/p | status |
|---|---|---|---|
| 6 | 2,3 | | |
| 12 | 2,3 | | |
| 24 | 2,3 | | |

35 5,7

-90 2,3,5

1 0

5 5

code:
====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
count=0
if number<0: number=-number
if number==1:print(number)
else:
    for fact in range(1,number+1):
        if number%fact==0:
           #factor has to check prime or not
           for j in range(1,fact+1):
               if fact%j==0:
                   count+=1
           if count==2: print(fact)
        fact+=1
        count=0
```

10.print the maximum factor of the given number(exclude

the given number):
================================================

code:
====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
max=0
if number<0: number=-number
for fact in range(number-1,1,-1):
    if number%fact==0:
       if max<fact:
           max=fact
           break
    fact+=1
print("maximum factor:",max)
```

11.print the maximum even factor of the given number(exclude

the given number):
==================================================

code:

====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
max=0
if number<0: number=-number
for fact in range(number-1,1,-1):
    if number%fact==0:
      if max<fact:
          max=fact
          break
    fact+=1
print("maximum factor:",max)
```

12.print the maximum even factor of the given

number(exclude the given number):
============================================

code:
====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
max=0
if number<0: number=-number
for fact in range(number-1,1,-1):
    if number%fact==0 and fact%2==0:
      if max<fact:
          max=fact
          break
    fact+=1
print("maximum factor:",max)
```

13.print the maximum odd factor of the given

number(exclude the given number):
=================================================

code:
=====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
max=0
if number<0: number=-number
for fact in range(number-1,1,-1):
    if number%fact==0 and fact%2!=0:
      if max<fact:
          max=fact
          break
```

```
    fact+=1
print("maximum factor:",max)
```

14.print the maximum prime factor of the given

number(exclude the given number):
==================================================

TCM:
====

| input: | exp .o/p | actual. o/p | status |
|---|---|---|---|
| 6 | 3 | | |
| 12 | 3 | | |
| 24 | 3 | | |
| 35 | 7 | | |
| -90 | 5 | | |
| 1 | 1 | | |
| 5 | 5 | | |

code:
====

```
#take the nummber
number=int(input("enter the number:"))
fact=number
count=0
max=0
times=0
if number<0: number=-number
if number==1:print(number)
else:
    for fact in range(number,0,-1):
        if number%fact==0:
           #factor has to check prime or not
           for j in range(1,fact+1):
               if fact%j==0:
                   count+=1
           if count==2:
               max=fact
        if max==0:fact+=1
        else:
            print(max)
            break
        count=0
        times+=1
```

print(times)

15.print the minimum factor of the given number(exclude

the given number and 1):
==================================================

code:
====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
min=number
if number<0: number=-number
for fact in range(2,number-1):
    if number%fact==0:
       if min>fact:
           min=fact
           break
    fact+=1
print("minimum factor:",min)
```

16.print the minimum even factor of the given number(exclude

the given number and 1)
===================================================

code:
====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
min=number
if number<0: number=-number
for fact in range(2,number-1):
    if number%fact==0 and fact%2==0:
       if min>fact:
           min=fact
           break
    fact+=1
print("minimum factor:",min)
```

17.print the minimum odd factor of the given number(exclude

the given number and 1):
==================================================

code:

====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
min=number
if number<0: number=-number
for fact in range(2,number-1):
    if number%fact==0 and fact%2!=0:
       if min>fact:
           min=fact
           break
    fact+=1
print("minimum factor:",min)
```

18.print the minimum prime factor of the given

number(exclude the given number and 1):
================================================

```
#take the nummber
number=int(input("enter the number:"))
fact=1
count=0
min=0
if number<0: number=-number
if number==1:print(number)
else:
    for fact in range(1,number+1):
        if number%fact==0:
           #factor has to check prime or not
           for j in range(1,fact+1):
               if fact%j==0:
                   count+=1
           if count==2:
               min=fact
        if min==0:fact+=1
        else:
            print(min)
            break
        count=0
```

19.find the sum of the factors of the given number
(Exclude the given number  as factor):
===========================================

code:
====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
```

```
sum=0
if number<0: number=-number
for fact in range(1,number):
    if number%fact==0:
        sum+=fact
    fact+=1
print("sum of the factors:",sum)
```

20.find the sum of the even factors of the given number
(Exclude the given number  as factor):
============================================

code:
====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
sum=0
if number<0: number=-number
for fact in range(1,number):
    if number%fact==0 and fact%2==0:
        sum+=fact
    fact+=1
print("sum of the factors:",sum)
```

21.find the sum of the odd factors of the given number
(Exclude the given number  as factor):
===============================================

code:
=====

```
#take the nummber
number=int(input("enter the number:"))
fact=1
sum=0
if number<0: number=-number
for fact in range(1,number):
    if number%fact==0 and fact%2!=0:
        sum+=fact
    fact+=1
print("sum of the factors:",sum)
```

22.find the product of the factors of the given number:
==========================================

```
#take the number
```

```
number=int(input("enter the number:"))
fact=1
prod=1
if number<0: number=-number
for fact in range(1,number):
    if number%fact==0:
        prod*=fact
    fact+=1
print("product of the factors:",prod)
```

23.find the product of the even factors of the given number:
==================================================

code:
====

```
#take the number
number=int(input("enter the number:"))
fact=1
prod=1
if number<0: number=-number
for fact in range(1,number):
    if number%fact==0 and fact%2==0:
        prod*=fact
    fact+=1
print("product of the factors:",prod)
```

24.find the product of the odd factors of the given number:
==================================================

code:
====

```
#take the number
number=int(input("enter the number:"))
fact=1
prod=1
if number<0: number=-number
for fact in range(1,number):
    if number%fact==0 and fact%2!=0:
        prod*=fact
    fact+=1
print("product of the factors:",prod)
```

25.find the hcf or gcd of the given two numbers:
========================================

TCM:
====

| input1 | input2 | ex. o/p | actual. o/p | status |
|---|---|---|---|---|
| 10 | 20 | 10 | | |

15 55 5

40 12 4

-12 15 3

1 0 no output

0 1 no output

code:
====

```
#take the number1
num1=int(input("enter the number1:"))
num2=int(input("enter the number2:"))
times=0
if num1!=0 and num2!=0:
    if num1<0: num1=-num1
    if num2<0: num2=-num2
    if num1>num2: num1,num2=num2,num1
    for fact in range(num1,1,-1):
        if num1%fact==0 and num2%fact==0:
            print(fact)
            break
        times+=1
    print(times)
else:
    print("no output")
```

26. find the lcm of the given two numbers:
==================================

TCM:
====

| input1 | input2 | ex. o/p | actual. o/p | status |
|---|---|---|---|---|
| 10 | 20 | 20 | | |
| 15 | 55 | 165 | | |
| 40 | 12 | 120 | | |
| -12 | 15 | 60 | | |
| 1 | 0 | no output | | |
| 0 | 1 | no output | | |

code:
====

```
#take the number1
num1=int(input("enter the number1:"))
num2=int(input("enter the number2:"))
if num1!=0 and num2!=0:
    if num1<0: num1=-num1
    if num2<0: num2=-num2
    if num1>num2: num1,num2=num2,num1
    for fact in range(num1,1,-1):
        if num1%fact==0 and num2%fact==0:
            print((num1*num2)//fact)
            break
else:
    print("no output")
```

27. from the given two numbers, which number is having more

    factors , display the number ( while calculating exclude 1

    and that number as factor)
==================================================

tcm:
====

| input1 | input2 | expected output |
|---|---|---|
| 10 | 20 | 20 |
| 13 | 40 | 40 |

code:
====

for with else in python:
==================

```
for i in range(1,10):
    print(i)
else:
    print("i am executed")
for i in range(1,10):
    if i==5:
        break
    else:
        print(i)
```

```
else:
    print("i am executed")
```

note:
====

when we write "for with else" in python program, else block

is always executed, but else block will not executed when

break statement is executed in the for loop

=====================================================

problems on numbers:
===================

1.Questions on prime number:
=========================

1.check the given number is prime or not

2.print the prime numbers for the given range:
=======================================

TCM
====

| start | end | ex. o/p | ac. o/p | status |
|---|---|---|---|---|
| 2 | 10 | 2,3,5,7 | | |
| 10 | 15 | 11,13 | | |
| 15 | 10 | 11,13 | | |
| 1 | 0 | no output | | |
| 12 | 16 | 13 | | |
| 15 | 16 | no output | | |
| 0 | 0 | no output | | |
| -10 | -13 | 11,13 | | |

code:
=====

```
start=int(input("start:"))
end=int(input("end:"))
if start<=0: start=-start
if end<=0: end=-end
if start>end:start,end=end,start
if start==0 or end==0:
    print("no output")
```

```
elif end==1 or start==1:
    print("no output")
else:
    for i in range(start,end+1):
        for fact in range(2,i):
            if i%fact==0:
                break
        else:
            print(i)
```

3.print the maximum prime number for the given range:
==========================================

TCM
====

| start | end | ex. o/p | ac. o/p | status |
|---|---|---|---|---|
| 2 | 10 | 7 | | |
| 10 | 15 | 13 | | |
| 15 | 10 | 13 | | |
| 1 | 0 | no output | | |
| 12 | 16 | 13 | | |
| 15 | 16 | no output | | |
| 0 | 0 | no output | | |
| -10 | -13 | 13 | | |

code:
====

```
start=int(input("start:"))
end=int(input("end:"))
max=0
if start<=0: start=-start
if end<=0: end=-end
if start>end:start,end=end,start
if start==0 or end==0:
    print("no output")
elif end==1 or start==1:
    print("no output")
else:
    for i in range(end,start+1,-1):
        for fact in range(2,i):
            if i%fact==0:
                break
```

```
        else:
            max=i
        if max!=0:
            break
print(max)
```

4.print the minimum prime number for the given range :
=============================================

TCM
====

| start | end | ex. o/p | ac. o/p | status |
|---|---|---|---|---|
| 2 | 10 | 2 | | |
| 10 | 15 | 11 | | |
| 15 | 10 | 11 | | |
| 1 | 0 | no output | | |
| 12 | 16 | 13 | | |
| 15 | 16 | no output | | |
| 0 | 0 | no output | | |
| -10 | -13 | 11 | | |

code:
====

```
start=int(input("start:"))
end=int(input("end:"))
min=0
if start<=0: start=-start
if end<=0: end=-end
if start>end:start,end=end,start
if start==0 or end==0:
    print("no output")
elif end==1 or start==1:
    print("no output")
else:
    for i in range(start,end+1):
        for fact in range(2,i):
            if i%fact==0:
                break
        else:
            min=i
        if min!=0:
            break
print(min)
```

5.count the how many prime numbers for the given range :
================================================

TCM
====

| start | end | ex. o/p | ac. o/p | status |
| --- | --- | --- | --- | --- |
| 2 | 10 | 4 | | |
| 10 | 15 | 2 | | |
| 15 | 10 | 2 | | |
| 1 | 0 | 0 | | |
| 12 | 16 | 1 | | |
| 15 | 16 | 0 | | |
| 0 | 0 | 0 | | |

code:
====

```
start=int(input("start:"))
end=int(input("end:"))
count=0
if start<=0: start=-start
if end<=0: end=-end
if start>end:start,end=end,start
if start==0 or end==0:
    print(0)
elif end==1 or start==1:
    print(0)
else:
    for i in range(start,end+1):
        for fact in range(2,i):
            if i%fact==0:
                break
        else:
          count+=1

print(count)
```

6. print the prime numbers which are having at least one even

digit for the given range:
==================================================

TCM
====

| start | end | ex. o/p | ac. o/p | status |
|---|---|---|---|---|
| 2 | 10 | 2 | | |
| 20 | 30 | 23,29 | | |
| 15 | 10 | 0 | | |
| 1 | 0 | 0 | | |
| 12 | 16 | 0 | | |
| 15 | 16 | 0 | | |
| 0 | 0 | 0 | | |

code:
====

```
start=int(input("start:"))
end=int(input("end:"))
count=0
if start<=0: start=-start
if end<=0: end=-end
if start>end:start,end=end,start
if start==0 or end==0:
    print(0)
elif end==1 or start==1:
    print(0)
else:
    for i in range(start,end+1):
        for fact in range(2,i):
            if i%fact==0:
                break
        else:
            for k in str(i):
                if k in "2468":
                    count+=1
            if count>=1:
                print(i)
        count=0
```

2.questions on perfect number:
========================

1.check given number is perfect or not

2.print the perfect numbers in between given range:
========================================

```
start=int(input("start:"))
end=int(input("end:"))
count=0
if start<=0: start=-start
if end<=0: end=-end
if start>end:start,end=end,start
data=[1,6,28,496,8128] #list of perfect numbers
result=[]#empty list
for i in data:
    if start<=i<=end:
        result+=[i]
if len(result)==0:
    print("no ouput")
else:
    for i in result:print(i,end=",")
```

3.print the maximum perfect numbers in the given range:
==============================================

TCM
====

| start | end | ex. o/p | ac. o/p | status |
|---|---|---|---|---|
| 2 | 10 | 6 | | |
| 20 | 30 | 28 | | |
| 15 | 10 | 0 | | |
| 1 | 0 | 0 | | |
| 12 | 16 | 0 | | |
| 15 | 16 | 0 | | |
| 0 | 0 | 0 | | |

code:
=====

```
start=int(input("start:"))
end=int(input("end:"))
count=0
if start<=0: start=-start
if end<=0: end=-end
if start>end:start,end=end,start
data=[1,6,28,496,8128] #list of perfect numbers
result=[]#empty list
for i in data:
    if start<=i<=end:
```

```
        result+=[i]
if len(result)==0:
    print(0)
else:
    print(result[-1])#last number in the list
```

4.print the minimum perfect number in the given range :
==============================================

TCM
====

start     end      ex. o/p      ac. o/p     status

2          10        6

20        30       28

15        10       0

1           0         0

12         16       0

15         16      0

0            0       0

code:
====

```
start=int(input("start:"))
end=int(input("end:"))
count=0
if start<=0: start=-start
if end<=0: end=-end
if start>end:start,end=end,start
data=[1,6,28,496,8128] #list of perfect numbers
result=[]#empty list
for i in data:
    if start<=i<=end:
        result+=[i]
if len(result)==0:
    print(0)
else:
    print(result[0])#starting number in the list
```

5.count the perfect numbers in the given range :
========================================

TCM
====

start     end      ex. o/p      ac. o/p     status

| | | |
|---|---|---|
| 2 | 10 | 1 |
| 20 | 30 | 1 |
| 15 | 10 | 0 |
| 1 | 0 | 0 |
| 12 | 16 | 0 |
| 15 | 16 | 0 |
| 0 | 0 | 0 |

code:
=====

```
start=int(input("start:"))
end=int(input("end:"))
if start<=0: start=-start
if end<=0: end=-end
if start>end:start,end=end,start
data=[1,6,28,496,8128] #list of perfect numbers
result=[]#empty list
for i in data:
    if start<=i<=end:
        result+=[i]
if len(result)==0:
    print(0)
else:
    print(len(result))#total numbers in the list
```

6.count the how many 2 digit perfect number for the

given range

7.print the numbers for the given range, where each number

will have at least one perfect digit

3.programs on strong numbers:
=========================

write a python program to find the factorial of the given

number:
============================================

TCM

| number | expected o/p | actual 0/p | status |
|---|---|---|---|

5 120

-5 not defined

-3 not defined

-100 not defined

code:
====

```
number=int(input("enter the number:"))
if number<0 and number!=0:
    print("not defined")
elif number==0 or number==1:
    print(1)
elif number>=2:
    fact=1
    for i in range(1,number+1):
        fact*=i
    print(fact)
```

1.check given number is strong number or not:
=====================================

TCM:

input1 expected.o/p actual. o/p
=============================

145 strong

1 strong

-145 strong

10 not strong

0 not strong

3 not strong

code:
====

```
number=int(input("enter the number:"))#145
if number<0: number=-number
fact=1
sum=0
for i in str(number):#145
    for j in range(1,int(i)+1):
        fact*=j
    sum+=fact
```

```
    fact=1
if sum==number:print("strong")
else:print("not strong")
```

2.print the strong numbers in between the given range:
==============================================

TCM:

| start | end | expected.o/p | actual. o/p | status |
|---|---|---|---|---|
| 1 | 10 | 1,2 | | |
| 1 | 100 | 1,2 | | |
| 1 | 1000 | 1,2,145... | | |

code:
====

```
start=int(input("start:"))
end=int(input("end:"))
if start<0:start=-start
if end<0:end=-end
if start>end:start,end=end,start
times=0
fact=1
sum=0
for number in range(start,end+1):
  for i in str(number):
     for j in range(1,int(i)+1):
        fact*=j
     sum+=fact
     fact=1
     times+=1
  if sum==number:print(number)
  sum=0
  times+=1
print(times)
```

or

```
start=int(input("start:"))
end=int(input("end:"))
if start<0:start=-start
if end<0:end=-end
if start>end:start,end=end,start
data=[1,2,145,40585]
result=[]
for i in data:
    if start<=i<=end:
        result+=[i]
if len(result)==0:
```

```
    print("no ouput")
else:
    for i in result:print(i,end=" ,")
```

4.problems on Armstrong number:
===========================

1.check given number is Armstrong or not:
==================================

code:
====

```
number=int(input("enter the number:"))
if number<=0:number=-number
sum=0
#here we find the length of the number
power=len(str(number))
for i in str(number):#153
    sum+=int(i)**power
if number==sum:print("Armstrong")
else:print("not a armstrong")
```

2.print the Armstrong numbers for the given range:
========================================

code:
====

```
start=int(input("start:"))
end=int(input("end:"))
if start<0:start=-start
if end<0:end=-end
if start>end:start,end=end,start
sum=0
for number in range(start,end+1):
    power=len(str(number))
    for i in str(number):
        sum+=int(i)**power
    if sum==number:print(number)
    sum=0
```

5.program on disarium numbers:
==========================

1.check the given number is disarium number or not:
==========================================

code:
====

```
number=int(input("enter the number:"))
if number<0:number=-number
pow=1
sum=0
```

```
for i in str(number):
    sum+=int(i)**pow
    pow+=1
if sum==number:print("disarium number")
else:print("not a disarium number")
```

2.print the disarium numbers for given range:
=====================================

code:
====

```
start=int(input("start:"))
end=int(input("end:"))
if start<0:start=-start
if end<=0:end=-end
if start>end:start,end=end,start
pow=1
sum=0
for number in range(start,end+1):
    for  i in str(number):
        sum+=int(i)**pow
        pow+=1
    if sum==number:print(number)
    pow=1
    sum=0
```

6.programs on buzz number(number ends with 7 or divisible

by 7)
================================================

1.check given number is buzz number or not:
=====================================

code:
====

```
number=int(input("enter the number:"))
if number<0:number=-number
if str(number)[-1]=='7' or number%7==0:
    print("buzz number")
else:print("not a buzz number")
```

2.print the buzz numbers for given range:
==================================

code:
=====

```
start=int(input("Start:"))
end=int(input("End:"))
if start<0:start=-start
```

```
if end<0:end=-end
if start>end:start,end=end,start
for number in range(start,end+1):
    if str(number)[-1]=='7' or number%7==0:
        print(number)
```

7. harshad number or niven number:
==============================

if a number which is divisible by the "sum of the digits"

of the number

example:
========

36 ===>3+6===>36%9===>0 (it is a niven number or harshad)

18 ====>1+8===>18%9 ===>0  (it is a niven number or

harshad)

25====>2+5==>25%7====>4 (not a niven number or harshad)

1.check the given number is harshad number or not:
==========================================

code:
====

```
number=int(input("enter the number:"))
sum=0
if number<0:number=-number
for i in str(number):
    sum+=int(i)
if number%sum==0:print("harshad/niven number")
else:print("not a harshad or niven number")
```

2.print the hashad numbers for the given range:
=======================================

code:
====

```
start=int(input("Start:"))
end=int(input("End:"))
if start<0:start=-start
if end<0:end=-end
if start>end:start,end=end,start
sum=0
for number in range(start,end+1):
    for i in str(number):
        sum+=int(i)
    if number%sum==0:print(number)
    sum=0
```

8. programs on tech number:
=======================

tech number:
===========

```
   81 ====>
                  8,1

                  8+1=9*9=81

  2025 ====> 20,25===>20+25==>45*45=2025
```

number length should be even

divide the number into two halfs

sum the two halfs

square the sum and check with number

if both are same, then number is "Tech number"

otherwise it is not a "Tech number"

code:
====

```
number=int(input("enter the number:"))
if number<0:number=-number
if len(str(number))%2==0:
    position=len(str(number))//2
    sum=int(str(number)[:position])+int(str(number)[position:])
    if number==sum**2:print("Tech number")
    else:print("not a Tech number")
else:
    print("given number is not a Tech number")
```

2.print the tech numbers for the given range:
===================================

```
start=int(input("Start:"))
end=int(input("End:"))
if start<0:start=-start
if end<0:end=-end
if start>end:start,end=end,start
sum=0
for number in range(start,end+1):
    if len(str(number))%2==0:
        pos=len(str(number))//2
        sum=int(str(number)[:pos])+int(str(number)[pos:])
        if number==sum**2:print(number)
```

9.programs on magic number:
========================

if number a "sum of the digits of the number is 1,we will do

sum of the digits of the given number until to get single digit"

example:
========

101 ====>1+0+0+1 ===>2(it is not a magic number)

100====>1+0+0===>1(it is a magic number)

989===>9+8+9===>26==>2+6==>8(it is not a magic number)

code:
====

```
number=int(input("enter the number:"))
if number<0:number=-number
sum=0
flag=True
while flag:
    sum+=(number%10)
    number=number//10
    if number==0:
        if sum<10:
            flag=False
        else:
            number=sum
            sum=0
print(sum)
```

functions in python:
================

function is a "block of statements and which is going to

preform a specified task in the program"

in python ,functions are used to eliminate "code reduancy"

and gives "code reusability"

with help of functions, we can write a any code once, use as

many times we want in the code

to create the function in python , we will use the following

syntax:
======

```
def function_name(arg1,arg2,arg3,arg4,....argn):

    #logic for function
```

note:
====

def is a keyword in python and it is used to create the function

in python

every function will have a unique name and using this we

can identify every function in python

in python , we can able to create any number of functions

in python, defining the arguments or parameters in the

function are optional

the arguments in the function are called as "formal arguments"

how to execute the function in the python:
==================================

creating the function is not enough to execute a function, it

means when we create a function in the program, it will never

execute , until we will call the function

calling the function means "Executing the function"

to call the function in python, we will use the following syntax:
==================================================

```
name_of_function(arg1,arg2,....argn)
```

note:
====

to call the function, we will use "name of the function"

the arguments in the function calling, is always depend on

function definition(how we define the function)

if the function is having arguments, while calling the function

we need to pass arguments(pass the data to the function) arguments is nothing "data to the function"

the arguments in the function calling are known as "actual

arguments"

if any function has arguments, they can  get the data from

the function calling.

if the function is not having any arguments, while calling the

function, we no need to pass the any arguments

example:
=======

```
#create the function with name "display()"
def display(): #here display is fucntion name (Called fucntion)
    print("this is my first fucntion in python")
#calling the fucntion
display() #caller fucntion
```

how many to create a function in python:
=================================

1.function without arguments and return type:
======================================

in this model, function is not going take any arguments and

after the execution of the function ,function will not return any

result

example:
=======

```
#function without arguments and return type
def display():
    x=int(input("x:"))
    y=int(input("y:"))
    print(x+y)
#calling the fucntion
display()
```

2.function with arguments and  without return type:
==========================================

in this model, function is  going take arguments and

after the execution of the function ,function will not return any result

example:
=======

```
#function without arguments and return type
def display(a,b):#a=x,b=y
    print(a+b)
#calling the fucntion
x=int(input("x:"))
y=int(input("y:"))
display(x,y)
```

3.function with arguments and with return type:
========================================

in this model, function is  going to take arguments and

after the execution of the function ,function will  return any

result

to return any result by the function, function uses a keyword

called "return"

syntax for return in python function:
=============================

return val1,val2,val3,...valn;

in python, return can return any number of values

in python, function will have the return statement only at

end of the function, because after return statement in function

if we write any code, it will not be executed

return often called as "exit" statement in the function

example:
=======

```
#function with arguments and with return type
def display(a,b):#a=x,b=y
   return a+b
#calling the fucntion
print(display(int(input("x:")),int(input("y:"))))
result=display(100,200)
print(result)
```

4.function without arguments and with return type:
==========================================

in this model, function is  not going to take any arguments and

after the execution of the function ,function will  return any

result

example:
=======

```
#function with arguments and with return type
def display():
    x=int(input("x:"))
    y=int(input("y:"))
    return x+y
#calling the fucntion
print(display())
result=display()
print(result)
```

types of arguments in functions in Python:
==================================

python functions will have the following type of arguments:
================================================

1.positional arguments:
===================

these arguments will give the data to "function arguments"

based on the position

when we are working with function with positional arguments,

function will take same number of arguments from the

function calling, if we miss argument, python will simply raises

an error

example:
=======

```
#function with positional arguments
def display(x,y,z):#x=10,y=20,z=30
    print(x,y,z)
#calling the fucntion
display(10,20,30)
display(100,200)
```

2.keyword arguments:
==================

these arguments will give the data to "function arguments"

based on the "name" (here name is act like keyword)

when we are working with function with keyword arguments,

function will take same number of arguments from the

function calling via name, if we miss any argument, python will

simply raises an error

when we are work with keyword arguments, the function will

take the value from  the function calling with help of name,  if

we give any wrong name in the function calling, it simply raises

an error (Wrong name, any name in the function which is not

matching)

example:
=======

```
#function with keyword arguments
def display(x,y,z):
    print(x,y,z)
#calling the fucntion
display(z=10,x=20,y=30)
display(y=100,z=200,x=10)
display(y=10,x=20,z=30)
```

3.default arguments:
=================

in python, when we are creating the function or when we are

defining the function, we will define the value for each

argument in the function, these values are act as default values

when we are not given any value to the argument in the

function  or when we skip the value to argument in the function

example:
=======

```
#function with default arguments
def display(x=10,y=20,z=30):
    print(x,y,z)
#calling the fucntion
display()
display(100)
display(y=200)
display(z=1000,y=200,x=10)
display(10,15,25)
```

4.var-length arguments:

1.arbitary arguments:

==================================================

with help of arbitrary arguments, we can give any number

arguments from the function calling to function.

the arbitrary arguments will taken by the function in python

as "tuple" format

example:

=======

```
#function with arbitary arguments
def display(*x):
    print(x,type(x))
#calling the fucntion
display()# zero arguments
display(1,2,3,4)
display(10,20,30,40,50,60,70)
display(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17)
```

2.keyword arbitrary arguments:

=========================

with help of keyword arbitrary arguments, we can give any

number  arguments from the function calling to function.

the  keyword arbitrary arguments will taken by the function in

python as "dictionary" format

we will give the any number keyword arbitrary arguments to

the function using keyword

to define the keyword arbitrary arguments in the python function, we will use "**"

example:
=======

```
#function with keyword arbitary arguments
def display(**x):
    print(x,type(x))
#calling the fucntion
display()# zero arguments
display(a=10,b=20,c=30,d=40,e=50)
display(a=1,b=2,c=3,d=4,e=5,f=6,g=8,i=9)
```

note:
====

in python, when we are defining the function with

arbitrary arguments, we will use name as "*arg"

in python, when we are defining the function with

keyword arbitrary arguments, we will use name as "*kwarg"

lambda functions in python:
======================

lambda function is a function  and which created using

"lambda" keyword

lambda function is also called as "single line function"

lambda function is also called as "anonymous function",

because lambda function will not have any name like normal

function

lambda function can take any number of arguments ,but it

can have only single line as code

syntax:
======

lambda arg1,arg2,arg3,....argn: expression

example:
=======

lambda:print("this is lambda function")

how to call the lambda function in python:
=================================

generally lambda function is anonymous function, due to the

reason, we will store the lambda function, in some name like

a variable

example:
=======

```
result=lambda:print("this is lambda function")
```

example:
=======

```
result=lambda:print("this is lambda function")
print(type(result))
#call the lambda function
result()
result=lambda a,b:print(a+b)
result(10,20)
result=lambda a,b,c:print((a+b+c)//3)
result(10,20,30)
```

note:
====

in python, lambda function will always return a value, except

when write print() function in lambda

example-2:
=========

```
result=lambda a,b:a+b
value=result(10,20)
print(value)
print(result(5,4))
```

functional cloning in python:
======================

functional cloning means "creating the function as same as

original function using reference(name) in python"

example:
=======

```
def is_even_or_odd(number):
    if number%2==0:
        return "even"
    else:
        return "odd"
print(type(is_even_or_odd))
a=is_even_or_odd
print(a(100))
print(is_even_or_odd(100))
print(id(a))
print(id(is_even_or_odd))
```

monkey patching in python:
======================

using monkey patching , we can able to change the source code

of the any function, without changing the original code of the

function directly

example:
=======

```
def is_even_or_odd(number):
    if number%2==0:
        return "even"
    else:
        return "odd"
def check_number_is_positive_even_or_not(number):
    if number>0 and number%2==0:
        return "postive even"
    elif number<0 and number%2==0:
        return "not a positive even"
print(is_even_or_odd(-10))
#performing monkey patching
is_even_or_odd=check_number_is_positive_even_or_not
print(is_even_or_odd(-10))
```

python closures or python inner functions:
===================================

inner function means " we can create a function inside another

function"

inner functions are also called as "nested functions"

in python, we can create a function inside another function

when we are working with inner functions, we should know

the following:
==================================================

1.inner function can be called, only where we create the inner

function

2.inner function can access the "outer function or main

function data"

3.outer function can not access the "inner function data"

4.in outer function, we can able to create the "any number of

inner functions"

example-1:
=========

```
def outer():
    def inner():
        print("this is my first inner fucntion")
    inner()
outer()
def outer2(x,y):
    def inner():
        print(x)
        print(y)
    inner()
outer2(10,20)
def outer3(x,y):
    def inner(a,b,c):
        print(a,b,c)#10,20,30
    inner(x,y,x+y)
outer3(10,20)
```

example-2:
=========

how to send the inner function as value to outside the outer

function or how to export the inner function from outer

function in python:

to send the inner function as value to outside the outer

function or  to export the inner function from outer

function in python ,we will use "return" keyword

code:
====

```
def outer():
    def inner():
        print("this is my first inner fucntion")
    return inner
result=outer()
print(type(result))
result()
```

example-2:
=========

```
def outer(x,y):
    def inner(z):
        return x*y*z
    return inner
result=outer()
print(type(result))
result()
```

example-3:
=========

```
def outer(y):
    return lambda x:y*x
result=outer(10)#y=10
print(result(10))#100
```

example-4:
=========

```
result=lambda x,y:lambda z:z*x*y
result2=result(10,20)
print(result2(30))#30*10*20
```

recursion in python:
================

recursion "means doing a same job for multiple times"

in python, functions can able to follow the "recursion"

if a function follows "recursion" , then the function is called as

"recursive function"

recursive function "Can able to call by itself"

example:
=======

```
def display(x):
    if x<=5:
        print(x)
```

```
        x+=1
        display(x)
display(1)#call the display function
```

write a python program to find the sum of the numbers for

the given range using recursion:
==================================================

code:
====

```
def recursive_sum(start,end,sum):
    if start<=end:
        sum+=start
        start+=1
        recursive_sum(start, end, sum)
    else:
          print(sum)
recursive_sum(int(input("start:")), int(input("end:")), 0)
```

write a python program to calculate the factorial of the given

number using recursion:
====================================================

code:
====

```
def recursive_factorial(number):
    if number==0 or number==1:
        return 1
    elif number<0:
        return "give valid +ve integer number"
    elif number>0:
        return number*recursive_factorial(number-1)
print(recursive_factorial(int(input("enter the number:"))))
```

write a python program to print the number from 1 to 100,

where all numbers are even numbers using recursion
===================================================

code:
=====

```
def recursive_even(start,end):
    if start<=end:
        if start%2==0:print(start)
        start+=1
        recursive_even(start,end)
recursive_even(1,100)
```

write a python program to print the numbers which are prime

for the given range without any loop statement(Recursion):
==================================================

code:
====

```
def recursive_prime(start,end,times):
    if start<=end:
        def check_prime(fact,start,count,times):
            if fact<=start:
              if start%fact==0:
                count+=1
              fact+=1
              times+=1
              check_prime(fact,start,count,times)
            else:
                if count==2:
                    print(start)
        check_prime(1,start,0)
        start+=1
        times+=1
        recursive_prime(start,end)
recursive_prime(int(input("start:")),int(input("end:")),0)
```

python packing and un-packing:
=========================

python packing:
=============

packing means "storing the multiple values under the single

name"

in python, using packing, we can able to store the multiple

values under the single name

when we are doing the packing in python, the data type of the

packing result is always "tuple"

example:
=======

```
#packing in the python
a=10,20
b=1,2,3,4,5
c=10,20,30,40,50,60
print(c,type(c))
print(b,type(b))
print(a,type(a))
```

python un-packing:
===============

un-packing is a "process storing the values to the multiple

variables"

or

un-packing is a "process of the getting the values from a

iterable or collection in python"

example:
=======

```
#packing in the python
a=10,20 #here we pack 2 values
b=1,2,3,4,5 #here we pack 5 values
c=10,20,30,40,50,60 #here we pack 6 values
#un-packing
x,y=a#(2 values)
a1,b1,*c1=b #(5 values)
print(a1)
print(b1)
print(c1)#[3,4,5]
x1,y1,z1,*a1=c #(6 values)
print(x1)
print(y1)
print(z1)
print(a1)
```

packing and un-packing at function level:
================================

example:
=======

```
def sample_for_packing():
    return 10,20,30,40,50,60,70,80,90,100
#here we are performing the packing
result=sample_for_packing()
#we are performing un-packing
a,b,*c=result
print(a)
print(b)
print(c)
```

python annotations:
================

annotations will be used to give the " data type of the argument in python functions and used to give the data type of the

return type of the function"

to give the annotations for python function arguments , in

python we will use a symbol called ":"

syntax for function with annotations:
=============================

```
def function_name(arg1_name:type,arg2_name:type...)->type:

        #here we will write the logic for function
```

example:
=======

```
def display(a:int,b:int)->int: #here int means integer type
    return a+b
def display2(a:float,b:float)->float: #here float means float type
    return a+b
def display3(a:str,b:str)->str:#here str means string type
    return a+b
"""to know the annotations of the
function in python,we will use __annotations__
syntax:
    function_name.__annotations__
"""
print(display.__annotations__)
print(display2.__annotations__)
print(display3.__annotations__)
```

note:
====

to know the annotations of the any function, we will use

a variable called "__annotations__" and it will all annotations

of the function arguments and return type as a "dictionary"

for format

annotations will alert the programmer or developer, to give the

what arguments to the function in the real time

annotations will never give any error, even the programmer

given the wrong data type arguments from the function calling

example:

=======

```
def display(a:int,b:int)->int: #here int means integer type
    return a+b
def display2(a:float,b:float)->float: #here float means float type
    return a+b
def display3(a:str,b:str)->str:#here str means string type
    return a+b
"""to know the annotations of the
function in python,we will use __annotations__
syntax:
    function_name.__annotations__
"""
print(display.__annotations__)
print(display2.__annotations__)
print(display3.__annotations__)
print(display(10,20))
print(display(10.5,45.6))
print(display("hello","world"))
```

how to comment about arguments and the function in python:
================================================

when we want to write the comments in the python function,

we will use "triple quotes"

in python, any function description or comments, can be get

via "__doc__"

syntax:
======

function.__doc__

example:
=======

```
def display(a:int,b:int)->int: #here int means integer type
   """
    a ===> a is integer type
    b ===> b is integer type
    in this function we are going to perform the given two
    numbers addition and after the addition we will send
    the result
   """
   return a+b
print(display.__doc__)
```

example-2:
=========

```
print(print.__doc__)
```

print(input.__doc__)
print(int.__doc__)

python decorators:
===============

decorator is a "function and which is takes another function as

a argument" and using decorator, we can change the any

functionality or code of the function or enhance the function,

without changing the function directly

decorator is also called as "higher-order" function

in python, if we say any function is a higher-order, then the

function simply takes "a" function as an argument

python decorator simply "takes a function as arguments and

return an enhanced function as a result"

```
              as a argument                returns
 function ===========>decorator======>enhanced-function
```

in python, we will have two types of decorators:
=====================================

1. built-in decorators:
=================

the decorators which are given by python or which are not

created by the programmer or developer, those are called as

"built-in decorators"

in python, we will have the following built-in decorators:
===========================================

@staticmethod

@classmethod

@property ...............................

2. user-defined or custom decorators
=============================

the decorators which are not given by python or which are

created by the programmer or developer, those are called as "user-defined or custom decorators"

steps to work with decorator:
======================

step-1: create a function which is act as "decorator"

step-2: create a function which is argument to the "Decorator"

step-3: call the decorator function with argument as

function

step-4: call the function is returned by the decorator

example:
=======

```
#create a decorator
def mydecorator(func):#decorator
    def wrapper():
        func()
        print("This is the code is added with mydecorator_arguemnt")
    return wrapper
#create a function which is an argument to wrapper(Decorator)
def mydecorator_arguemnt():
    print("this is the wrapper fucntion for decoratror")
#call the decorator
result=mydecorator(mydecorator_arguemnt)
result()
```

note:
====

in python, we can give the decorator function, directly to

the function which we are giving as an argument using a

symbol called "@" and also give the name of the decorator

syntax:
======

@decorator_name

example:
=======

```
#create a decorator
def mydecorator(func):#decorator
    def wrapper():
        func()
        print("This is the code is added with mydecorator_arguemnt")
```

```
    return wrapper
#create a function which is an argument to wrapper(Decorator)
@mydecorator
def mydecorator_arguemnt():
    print("this is the wrapper fucntion for decoratror")
#call the decorator
mydecorator_arguemnt()
```

example-2:
=========

```
#create the decorator
def makeup(func):
   def wrapper():
       print("before i am devil,after makeup now i am looking a some girl")
   return wrapper
def some_girl(func):
   def wrapper():
       print("before i am evil, now i am  some wife")
   return wrapper

#function
@makeup
def devil():
    print("i am a devil")
@some_girl
def evil():
    print("i am a evil")
devil()
evil()
```

example-3:
=========

```
def mydecorator(func):
    def wrapper():
        a,b=func() #a=x,b=y
        return b,a
    return wrapper
@mydecorator
def take_two_numbers():
    x,y=10,20
    return x,y
x,y=take_two_numbers()
print(x,y)
```

example-4:
=========

```
def mydecorator(func):
    def wrapper(a,b):
        a,b=b,a
        return func(a,b)
    return wrapper
```

```
@mydecorator
def swap_two_numbers(x,y):
    return x,y
x,y=swap_two_numbers(10,20)
print(x,y)
```

python exception handling:
=====================

exception:
========

exception is a "runtime-error", but not "Syntax error"

exception is occurred at during the runtime and it will raises by

the processor, when it is not able to process the code

due to exceptions, the complete the code not executed,

because when code caught any exception, program terminated

from there itself

exceptions in the code cause "no-output"

when we have the exceptions in the code, we need to handle

the exceptions, to get output

the process of handling the exceptions in the given code or

program, is called as "Exception Handling"

how we can handle the exceptions in the given code in

python:
==================================================

to handle the exceptions, in python we will use the following

exception handlers:
================================================

1.try:
=====

try is a block and in this "we  will keep the code which will give

the exception"

or

in try block, we will write the "code which will have the exception"

in python, try block will have "risky code"

2.except:
=======

except is a block and in this "We will write the code, which may

give the details about exception or we can write the code which

is related to handle the exception"

3.raise:
======

raise is used to "raise the exception by the programmer

explicitly"

4.finally :
=======

finally is a block and in this "we will write a code and it has

to execute whether there is exception or there is no exception

generally it will have "safety code"

python exception handling template(code+exception handling):
================================================

```
try:

     here write the code which will give the exception

except:

     here we write the code related to exception

         or

    here we will write the code for exception handling

else:

    this code will be executed only, when there is no exception

    in try block

finally:
```

here we write the code, which will be executed when there

is exception or there is no exception

note:
====

we can not write the except  block without "try"

we can write the  try and except without finally , finally block

optional

in exception handling , else block is also optional

in python, we will have the following built-in exceptions:
===========================================

in python ,every exception name "ends" with "Error"

1.TypeError:
==========

it will get in the python, when we are performing the operation

on two different operands

example:
=======

```
a=10
b="hello"
print(a+b) #TypeError
```

2.ValueError:
==========

when we the given any wrong data to process by any function

example:
=======

```
a=int(input("enter the value for a:"))#ValueError(if we wrong data)
print(a)
```

3.IndexError :
===========

when we process "any index , something which is not avliable

on the list or string or range or tuple", we will get this

exception

example:
=======

```
a=range(1,11)
print(a[0])
print(a[20])
```

4.NameError :
============

using a name which is not defined in the program ,then will

raises this exception

example:
=======

```
a=10
b=20
print(x+y)
```

example-1:
=========

```
#swapping of the two numbers
try:
  a=int(input("enter the number for a:"))
  b=int(input("enter the number for b:"))
except ValueError:
    print("please give the valid integer number only")
else:
    a,b=b,a
    print(a,b)
```

example-2:
=========

```
#check given number is Armstrong or not:
try:
  number=int(input("Enter the number:"))
except ValueError:
    print("plese give valid integer number only")
else:
    length=len(str(number))
    sum=0
    for i in str(number):
        sum+=(int(i)**length)
    if sum==number:print("armstrong")
    else:print("not a armstrong")
finally:
    print("i am always executed")
```

example:
=======

any number we divide by the zero

retrieving the data from range() using index, which is not

there in range()

giving the wrong data to any function to process

giving a "character to int() function"

example:
=======

""""
write a python program print the numbers for given range ,

which are having starting digit more than ending digit

and having number of 9’s in the number should be minimum

1
""""
```
try:
  start=int(input("Start:"))
  end=int(input("End:"))
except ValueError:
    print("please give valid integer")
else:
   ninecount=0
   if start<0:start=-start
   if end<0:end=-end
   if start>end:start,end=end,start
   for number in range(start,end+1):
    if str(number)[0]>str(number)[-1]:
        for i in str(number):
            if i==’9’ and ninecount<=1:
              ninecount+=1
            elif ninecount>1:
                break
        if ninecount==1:
          print(number)
        ninecount=0
```

python Scope:
===========

in python, Scope refers "where we can able to access the

data in the python program"

scope defines "Accessibility of the data in the code"

in general, we will have two types of scope:
===================================

1.local scope:
===========

when we define the any data inside the "function" in

python or the data inside the function are having local

scope in python

2.global scope:
============

when we define the data outside the function, then the

data will have global scope in python

local data means "data which is related to function"

global data means "data which is defined outside the

function"

when we say any data is local, the data which we can

access where define the data

when we say any data is global, the data can be accessed

anywhere in the program

example:
=======

```
x,y=100,200 #here x and y are global data
#create the function
def display():
    a,b=10,20 #here a, b are local data
    print(a)
    print(b)
    print(x)
    print(y)
display()
```

note:
====

global data we can not change in the functions

global data can be change only where we create in python code

example-2:
=========

```
x,y=100,200 #here x and y are global data
#create the function
def display():
    a,b=10,20 #here a, b are local data
    print(a)
    print(b)
    print(x)
    print(y)
display()
x+=100
y+=200
print(x,y)
```

when python accessing the data via scope, using a rule called

"LEGB"

L ===>LOCAL

E===>ENCLOSED

G===>GLOBAL

B===>BUILT-IN

example-3:
=========

```
x=100 #global data x=100
def display():
    x=10#local data x=10
    print(x)#10
    def display2():
        x=300#local data x=300
        print(x)#300
        def display4():
            x=400#local data x=400
            print(x)#400
        print(x) #300
        display4()
    display2()
print(x)#100
display()
```

example-4:
=========

```
x=100
def display():
    print(x)#100
    def display2():
        x=300
        print(x)#300
        def display3():
            print(x)#300
            def display4():
                print(x)#300
            display4()
        display3()
    display2()
print(x)#100
display()
```

global keyword in python:
====================

global keyword is used in python, to make any local data as

"Global" and it make any global data can be manipulated

anywhere in the program or code

using global keyword "we are making the data as global"

example:
=======

```
x=100
print(id(x))
def display():
    global x
    print(id(x))
    x+=400
    print(x)#500

print(x)#100
display()
print(x)#500
```

example-2:
=========

```
x=10
def display():
    global x #
    print(x)#10
    x+=50  #60
    def display2():
```

```
        global x
        print(x)#60
        x=400
        x+=500
        print(x)#900
    print(x)#900
    display2()
display()
print(x)#900
```

example-3:
=========

```
x=100
def display():
    x=500
    def display2():
        global x #(100)
        x+=500
        print(x)#600
        def display3():
            global x
            x+=1000
        display3()
    display2()
display()
print(x)#600
```

example-4:
=========

```
x=50
def display():
    global x
    x=1000
    def display2():
        x=500
        x+=1000
        def display3():
            global x
            x=1000
        display3()
        def display4():
            global x
            x=2000

        display4()
    display2()
display()
print(x)
```

nonlocal keyword in python:
======================

nonlocal is keyword and which is used to reprersent

"Enclosed scope" data(generally function local data)

when we want to apply the any changes on the "local data of

the outer function in the it’s inner functions"

example-1:
=========

```
x=50
def display():
    def display2():
        x=500
        x+=500
    display2()
    print(x)#50
display()
print(x)#50
```

example-2:
=========

```
x=50
def display():
    def display2():
        global x
        x=500
        x+=500
    display2()
    def display3():
        global x
        x=500
        def display4():
            global x
            x=500
            print(x)#500
            def display5():
                global x
                x=0
            display5()
        display4()
    display3()
display()
print(x)#0
```

example-3:
=========

```
def display():
    def display2():
```

```
        x=100
        def display3():
            y=1000
            print(y)
            def display4():
                x=100
                x+=1000
            display4()
        display3()
    display2()
    global x
    print(x)#
display()
```

example-4:
=========

```
def display():
  x=100
  def display2():
      x=1000
      x+=1000
      print(x)#2000
  display2()#
  print(x)#
display()
```

example-5:
=========

```
def display():
    x=100
    def display2():
      nonlocal x
      x+=100
      print(x)
    display2()
display()
```

example-6:
=========

```
def display():
   def display2():
       x=1000
       def display3():
           nonlocal x
           x+=1000
       print(x)#1000
       display3()
       print(x)
   display2()
display()
```

example-7:
=========

```
def display():
   x=1000
   def display2():
       nonlocal x
       x+=1000#2000
       def display3():
           nonlocal x
           x+=1000
       print(x)#2000
       display3()
       print(x)#3000
   display2()
   print(x)#3000
display()
```

example-8:
=========

```
def display():
  x=1000
  def display2():
      x=1000
      x+=1000
      print(x)#2000
      def display3():
          nonlocal x
          x+=1000
          print(x)
      x+=1000
      print(x)#3000
  def display3():
     nonlocal x
     x+=3000
  display2()
  print(x)#1000
  display3()
  print(x)#4000
display()
```

example-9:
=========

```
x=1000
def display():
    x=1000
    x+=2000 #3000
    def display2():
        nonlocal x
        x+=3000 #6000
        def display3():
            global x
```

```
            x+=4000
    display2()
    print(x)#6000
display()
print(x)#1000
```

example-10:
==========

```
x=1000
def display():
    x=1000
    x+=2000 #3000
    def display2():
        nonlocal x
        x+=3000 #6000
        def display3():
            nonlocal x
            x+=4000
            print(x)
        display3()
    display2()
    print(x)
display()
print(x)#1000
```

example-11:
==========

```
x=1000
def display():
    x=1000
    x+=2000 #3000
    def display2():
        nonlocal x
        x+=3000 #6000
        def display3():
            nonlocal x
            x+=4000
            print(x)
        display3()
    display2()
    print(x)
display()
print(x)#1000
```

example-12:
==========

```
def display():
    x=100 # x is local variable of display
    def display2():
        y=200# y is local variable of display2
        print(x)#100
```

```
        def display3():
            nonlocal x # x=100
            nonlocal y # y=200
            x+=100 #200
            y+=100 #300
        print(x)#100
        print(y)#200
        display3()
    display2()
    print(x)#200
    print(y)

display()
```

python modules & packages:
=======================

module is a "python file"

using module(python file), we can send the data and functions

of one python file to another python file

using modules, we can able to "re-useability"

in order to work with python modules, we need to follow

the following steps:
================================================

step-1:  create a file with data and functions

step-2: save the file with "somename.py" and run the file

mymodule.py:
===========

```
#data
x=10
y=20
z=30
#functions
def add(x,y):print(x+y)
def sub(x,y):print(x-y)
def mul(x,y):print(x*y)
def div(x,y):print(x/y)
def int_div(x,y):print(x//y)
def remainder(x,y):print(x%y)
```

step-3: create the another file with "Somename.py"

step-4:  when we want to access the any data or functions from

any python module, we will use the following syntax:

import  module_name

example:  import mymodule

note:  module name is same as "file name"

step-5: once we import the any module in python ,we can

access any data or function of the module using

following syntax:

module_name.data _name

or

module_name.function_name

example:
========

test.py
=====

```
import mymodule
"""
here we are accessing the data from
mymodule
"""
print(mymodule.x)
print(mymodule.y)
print(mymodule.z)
"""
here we are accessing the functions from
mymodule
"""
mymodule.add(10,20)
mymodule.sub(10,20)
mymodule.mul(10,20)
mymodule.div(10,20)
mymodule.int_div(10, 20)
```

how to alias the module name in python:
================================

to alias the module name in python, we will use  a keyword

called "as"

syntax:
======

import module_name as alias_name

example:
=======

import mymodule as mm

example:
=======

import mymodule as mm
""""
here we are accessing the data from
mymodule
""""
print(mm.x)
print(mm.y)
print(mm.z)
""""
here we are accessing the functions from
mymodule
""""
mm.add(10,20)
mm.sub(10,20)
mm.mul(10,20)
mm.div(10,20)
mm.int_div(10, 20)

how to access only specific data or functions from the module

in python
==================================================

to access only specific data or functions from the module

in python, we will use a keyword called "from"

syntax:
======

from module_name import data_name1,data_name2,.....

example:
=======

from mymodule import x,y,add,sub
""""
here we are accessing the data from
mymodule
""""
print(x)
print(y)

""""
here we are accessing the functions from
mymodule

"""
add(10,20)
sub(10,20)

note:
=====

in python, we can access the data or functions from one or

more python modules at a time

example:
=======

import mymodule as mm,mymodule2 as mm2
"""
here we are accessing the data from
mymodule and mymodule2
"""
print(mm2.a)
print(mm.x)
print(mm.y)
print(mm.z)
print(mm2.b)
print(mm2.c)

example-2:
=========

to avoid the name while accessing the any data or functions

from module, we will use the following syntax:

from module_name import *

example
=======

code:
====

from mymodule import *
from mymodule2 import *
print(x)
print(y)
print(z)
print(a)
print(b)
print(c)
add(10,20)
sub(10,20)
mul(10,20)
div(10,20)
int_div(10, 20)

packages in Python:
================

package means "folder of python files" or "a group of python

files under a folder or collection of python modules under

a folder"

package ====>folder

module =====>file

when we want to create multiple python files(modules) under a

folder(package), with help of that we can able to reuse the

python file data or functions any where in the project in real-

time

to work with python projects, we will use the following steps:
==============================================

step-1: create a folder in system with some name

step-2: create an empty python file with name "__init__.py"

in the earlier created folder

step-3: create the all python modules(files) here

in order to access the python package module data or methods

we will use the following syntax:
================================================

import package_name.module_name.method_name()

or

import package_name.module_name.data_name

example-1:
=======

```
import mypackage.mymodule as mm
mm.add(10,20)
mm.sub(10,20)
mm.mul(10,20)
```

example-2:
=========

```
from mypackage.mymodule import *
add(10,20)
sub(10,20)
mul(10,20)
div(10,20)
int_div(10, 20)
```

in python, we can also create "sub-packages" in packages

step-1: create the package with name "mypackage2"

step-2: create empty python file with name "__init__.py"

step-3: create another package inside "mypackage2" with name

"mysubpackage"

step-4: once we create the "mysubpackage" in mypackage2,

again we need to create "an empty python file" with

name "__init__.py" inside the mysubpackage

step-5: create the modules inside the package and sub-

packages

example-1:
=========

```
from mypackage2.mymodule import *
from mypackage2.mysubpackage2.mymodule2 import *
add(10,20)
sub(10,20)
mul(10,20)
div(10,20)
int_div(10, 20)
print(a)
print(b)
print(c)
```

python built-in functions:
====================

in python, we will have two types of functions:
=====================================

1.user-defined functions:
====================

if a function which is created by the programmer or developer

then the function is called as "user-defined" function in python, we will create a function using "def" keyword

2.built-in functions:
===============

if a function which is given by the python, then the function is

called as "built-in" function

in python, we will have the following important built-in

functions:
=================================================

1.print() ==> to display any output

in python, using print() function we can able to format the

output:
=================================================

1.print the output using format specifiers:
=================================

in python, like C-language we can able to format the output:
===============================================

in python, we will have the following format specifiers:
==========================================

1. %d  <=== decimal integer

2. %f   <=== real number

3. %s   <==== string

4.%o <=== octal number

5.%x<=== hexadecimal number

example-1:
=========

```
a=10
b=20
c=30
print("the value of a is:%d"%(a))
print("the value of b is:%d"%(b))
print("the value of c is:%d"%(c))
print("the value of a:%d and b:%d is : %d"%(a,b,a+b))
d=1.2345
print("the value of d is:%f"%(d))
s1="hello"
```

print("the value of s1 is:%s"%(s1))

example-2:
========

a=10
b=20
c=30
print("the value of a is in octal :%o"%(a))
print("the value of a is in hexadecimal :%x"%(a))
print("the value of b is in octal :%o"%(b))
print("the value of c is in octal :%o"%(c))
print("the value of b is in hexadecimal :%X"%(b))
print("the value of c is in hexadecimal :%X"%(c))

note:
====

the above formatting output is like "printf style in C-language"

in python

2.using format() method, we are formatting the output in

Python:
==============================================

here format() is function and which used along with string

object only

syntax:
======

"string".format(variable1,variable2,...)

example-1:
=========

a,b,c=10,20,30
print("the value of a is:{}".format(a))
print("the value of b is:{}".format(b))
print("the value of c is:{}".format(c))
print("the value of a is:{} and b is:{}".format(b,a))

example-2:
=========

a,b,c=10,20,30
print("the value of a is:{0} and b is:{2} and c is {1}".format(b,a,c))
print("the value of a is {value1} and b is :{value2}".format(value1=a,value2=b))

3.format the output using f-strings in python:
====================================

f-string can start with "f" or "F"

syntax:
======

f’ write the output string ’

or

F’ write the output string’

example:
=======

```
a,b,c=10,20,30
print(f’the value of a is:{a}’)
print(F’the value of b is:{b}’)
print(f"the value of a+b is: {a+b}")
print(F"the value of a>b is:{a>b}")
print("the value of a is:",a)
```

2.input():<=== using this we can take input from the user

3.int()<=== it used to convert the given data into decimal

integer

example:
=======

```
print(int())
print(int(1.234))
```

4.float()<== it is used to convert the given data into real

number

example:
========

```
print(float())
print(float(1234))
```

5.complex()<== it is used to convert the given data into

complex number

example:
=======

```
print(complex())
print(complex(10+5j))
```

```
result=complex(10,20)
print(result)
print(result.real)
print(result.imag)
print(complex(100))
print(complex(1.234))
```

6.bool():<=== it is used to convert the given data into Boolean

the result of this function is either True or False

example:
=======

```
print(bool())
print(bool(0))
print(bool(''))
print(bool(-5000))
print(bool([]))
```

7.str()<=== it is used to convert the given data into string type

example:
=======

```
print(str())
print(type(str(10+5j)))
print(type(str([1,2,3,4,5,6,7,8])))
```

8.bin():
======

bin() function will convert the given number into binary number

example:
=======

```
print(bin(100))#decimal to binary
print(bin(0o127))#octal to binary
print(bin(0xab))#hexadecimal to binary
```

9.oct():
======

oct() function will convert the given number into octal number

example:
=======
```
print(oct(100))#decimal to octal
```

print(oct(0b1010101))#binary to octal
print(oct(0xab))#hexadecimal to octal

10.hex():
=======

hex() function will convert the given number into hexadecimal

number

example:
=======

```
print(hex(100))#decimal to hexadecimal
print(hex(0b1010101))#binary to hexadecimal
print(hex(0o127))#octal to hexadecimal
```

11.list():
=======

it is used to convert the given iterable into list

iterables in python(list,tuple,set,range,string,dictionary...)

example:
=======

```
print(list((1,2,3,4,5,6)))
print(list({1,2,3,4,5,65}))
print(list(range(1,11)))
print(list("hello"))
print(list({1:2,3:4,5:6}))
```

12.tuple():
========

it is used to convert the given iterable into tuple

iterables in python(list, tuple, set, range, string, dictionary...)

example:
=======

```
print(tuple((1,2,3,4,5,6)))
print(tuple({1,2,3,4,5,65}))
print(tuple(range(1,11)))
print(tuple("hello"))
print(tuple({1:2,3:4,5:6}))
print(tuple((10,)))
```

13.set():
========

it is used to convert the given iterable into set

iterables in python(list, tuple, set, range, string, dictionary...)

examples:
========

```
print(set((1,2,3,4,5,6,10,1,3,4,2,6)))
print(set({1,2,3,4,5,65}))
print(set(range(1,11)))
print(set("hello"))
print(set({1:2,3:4,5:6}))
print(set((10,)))
```

14.dict():
=========
 it is used in python to create the dictionary

example:
=======

```
print(dict(a=10,b=20,c=30))
print(dict(x=2,y=4,z=6))
print(dict(name="raj",salary=1000,location="hyderabad"))
```

15.eval():
=========

this function will be used in python "to evaluate the given

expression as string"

example:
=======

```
print(eval("10+20"))
print(eval("10*20"))
a,b=100,200
print(eval("a/b"))
print(eval("a==10"))
```

16.exec():
=========

this will execute any code "which we will given as a string"

example:

=======

```
exec("a=10")
exec("b=20")
exec("print(10+20)")
exec("print(a+b)")
exec("print(10>20)")
exec("result=(10<=20)")
exec("print(result)")
```

17.sum():
==========

this function is used find the "sum of the values of the given

iterable"

example:
=======

```
print(sum((1,2,3,4,5,6,10,1,3,4,2,6)))
print(sum({1,2,3,4,5,65}))
print(sum(range(1,11)))
print(sum({1:2,3:4,5:6}))
print(sum((10,)))
```

18.min():
==========

this function is used find the "minimum value of the given

given iterable values"

example:
=======

```
print(min((1,2,3,4,5,6,10,1,3,4,2,6)))
print(min({1,2,3,4,5,65}))
print(min(range(1,11)))
print(min({1:2,3:4,5:6}))
print(min((10,)))
```

19.max():
============

this function is used find the "maximum value of the given

given iterable values"

example:
=======

```
print(max((1,2,3,4,5,6,10,1,3,4,2,6)))
print(max({1,2,3,4,5,65}))
print(max(range(1,11)))
print(max({1:2,3:4,5:6}))
print(max((10,)))
```

20.abs():
=======

this function will give the always positive number

example:
=======

```
print(abs(10))
print(abs(-10))
print(abs(-1.234))
```

21.enumerate()

22.zip()

23.map()

24.filter()

25.setattribute()<== oops

26.frozenset()

27.reversed()

28.sorted()

29.isinstance()

30.len()

31.range()

32.all():
=======

all() function will take the data as iterable and it will return

"True", if all values in the given iterable are True, otherwise

"False"

example:
=======

```
"""working with all() function"""
print(all((1,2,3,4,5,6,7)))
print(all((1,2,3,4,5,6,7,-100)))
print(all((10,20,30,40,50,60,70,0)))
print(all([1,2,3,4,5,-1,10]))
```

33.any():
=======

any() function will take the data as iterable and it will return

"False", if all values in the given iterable are False, otherwise

"True" (even if single value in given iterable is True)

example:
========

```
"""working with any() function"""
print(any((1,2,3,4,5,6,7)))
print(any((1,2,3,4,5,6,7,-100)))
print(any((10,20,30,40,50,60,70,0)))
print(any([1,2,3,4,5,-1,10]))
print(any({0,0,0}))
```

34.divmod():
==========

divmod() function is going take "two numbers" as arguemns

and return (quotient, remainder) as result

divmod() performs floor division and modulo division on the

given numbers

syntax:
======

divmod(num1,num2) ===> (num1//num2,num1%num2)

example:
=======

```
"""working with divmod() function"""
print(divmod(10,20))#(0,10)
print(divmod(20,10))#(2,0)
```

35.dir():
======

using dir() function, we can able to know the function of the

given object in python

or

what functions the given object is having, can be known  with

help of "dir()" in python

example:
=======

```
"""working with dir() function"""
a=10 #integer object
b=1.234#float object
s1="hello" #string object
print(dir(a))
print(dir(b))
print(dir(s1))
```

36.callable():
==========

this function will check given "name" is function name or not

it will give the result either "True or False"

example:
========

```
"""working with callable() function"""
def display1():
    pass
def display2():
    pass
def display3():
    pass
a1=100
b1=200
c1=300
print(callable(a1))
print(callable(b1))
print(callable(c1))
print(callable(display1))
print(callable(display2))
print(callable(display3))
```

37.chr():
=======

this function will return the "given number related character"

example:

=======

```
"""working with chr() fucntion"""
print(chr(97))
print(chr(98))
print(chr(65))
print(chr(90))
print(chr(122))
```

38.ord():
=======

this function will return the "given character related Unicode

number"

example:
=======

```
"""working with chr() fucntion"""
print(ord('A'))
print(ord('B'))
print(ord('a'))
print(ord('b'))
print(ord('z'))
```

39.locals():
=========

it will return a "dictionary as a result" and the dictionary shows

all local variables of a function

40.globals():
==========

it will return a "dictionary as a result" and the dictionary shows

all global variables of a program

example:
=======

```
"""working with globals() and locals()"""
a=10
b=20
c=30
def display():
    x=10
    y=20
    print(locals())
display()
print(globals())
```

41.repr():
========

this is function used in python, to represent the given object

in the form of string

example:
=======
```
"""working with repr()"""
print(repr(10))
print(repr([1,2,3,4]))
result=repr(100)
print(result)
print(type(result))
```

42.id():
======

using this function we can know "object address at memory

location"

example:
========
```
"""working with id() """
a,b,c=1,1.234,"hello"
print(id(a))
print(id(b))
print(id(c))
```

43. round():
==========

this function we will use "to represent real number as rounded

number"

or

using round() function, we can round the "given real number"

syntax:
======

round(real_number)

example:
=======

```
"""working with round() function """
print(round(1.234,0))
print(round(1.23456,3))#1.235
print(round(1.234567,4))#1.2346
```

44.hash():
========

using this function we can "convert the given object  into hash

code"

example:
=======

```
"""working with hash() function """
print(hash(10))
print(hash("hello"))
print(hash(1.234))
```

45.vars():
========

this function is also work like "locals()"

this function will simply return the "local variables as a

dictionary"

example:
========

```
"""working with vars() function """
def display():
    a=10
    b=20
    c=30
    """vars():it will return all local variables of the
    display function"""
    print(vars())
display()
```

46.slice():
========

this function is used to create the "Slice" object

the given slice object we can apply any sequence type

(list, tuple, string, range())

example:
=======

```
"""working with slice() fucntion"""
s1=slice(0,4)
s2=slice(0,5)
s3=slice(3,6)
l1=[1,2,3,4,5,6,7,8,9,10]
print(l1[s1])#l1[0:4]
print(l1[s2])#l1[0:5]
```

print(l1[s3])#l1[3:6]

47.property()

48.hasattr()

49.super()

50.type():
========

type() is used to "to know the type of the given object"

example:
========

"""working with type() fucntion"""
print(type(10))
print(type(type(1.234)))
print(type(type(type(1.234))))

python built-in modules:
====================

these modules are given by "python"

1.math:
======

math module is given by python and using this module we can

able to perform "all mathematical operations"

in order to work with math module, we will use the following

synax:
==================================================

import math as mt

in math module we will have the following some important

functions:
==================================================

1.floor():
======================================

this function will give the "lowest integer value for the given real number or floating-point number"

example:
=======

```
import math as mt
print(mt.floor(1.234))
print(mt.floor(4.567))
print(mt.floor(-4.567))
```

2.ceil():
==================================================

this function will give the "height integer value for the given

real number or floating-point number"

example:
=======

```
import math as mt
print(mt.ceil(1.234))#2
print(mt.ceil(4.567))#5
print(mt.ceil(-4.567))#-4
```

3.trunc():
=======

using this function we can "truncate/ remove the fractional

part in the given real or floating-point number"

example:
=======

```
import math as mt
print(mt.trunc(1.234))#2
print(mt.trunc(4.567))#5
print(mt.trunc(-4.567))#-4
```

4.fabs():
==============================================

it will give the "absolute value of the given number and it

always give the result as float value"

example:
=======

```
import math as mt
print(mt.fabs(12))
print(mt.fabs(100.0))
print(mt.fabs(-12))
print(abs(-100))
```

5.perm():
=======

using this we can find the permutation value for given

n and r

example:
=======

```
import math as mt
print(mt.perm(10,3))
print(mt.perm(10,10))
print(mt.perm(10,20))
print(mt.perm(10,-2))
```

6.comb():
=======

using this we can find the combination value for given

n and r

example:
=======

```
import math as mt
print(mt.comb(10,3))
print(mt.comb(10,10))
print(mt.comb(10,20))
```

7.fsum():
=======

it is used to find the "sum of value of the given iterable and

it will give the result always float value"

example:
=======

```
import math as mt
print(mt.fsum([1,2,3,4]))
print(mt.fsum(range(1,10)))
print(mt.fsum((1,2,3,4,5)))
```

8.sqrt():
=======

it will give the square root of the given number

example:
========

```
import math as mt
print(mt.sqrt(2))
print(mt.sqrt(16))
print(mt.sqrt(54))
print((2)**(0.5))
```

9.cbrt():
=======

it will give the cube root of the given number

example:
=======

```
import math as mt
print(mt.cbrt(2))
print(mt.cbrt(16))
print(mt.cbrt(54))
print((2)**(1/3))
```

10.gcd():
=======

using this function we can find the "gcd of the given two

numbers"

example:
=======

```
import math as mt
print(mt.gcd(10,20))
print(mt.gcd(16,4))
print(mt.gcd(100,10))
```

11.pow():
========

using this function can find the "power of the given base and

power"

example:
=======

```
import math as mt
print(mt.pow(10,2))
print(mt.pow(16,4))
print(mt.pow(100,5))
```

12.prod():
========

using this function we can find the "product of the given

two numbers"

example:
=======

```
import math as mt
print(mt.prod([1,2,3,4,5]))
print(mt.prod([10,20,30,40]))
```

13.log10():
=========

this function is used to find the "log value for given value

with base-10"

example:
=======

```
import math as mt
print(mt.log10(1))
#print(mt.log10(0))
print(mt.log10(10))
```

14.log2():
========

this function is used to find the "log value for given value

with base-2"

example:
=======

```
import math as mt
print(mt.log2(1))
#print(mt.log10(0))
print(mt.log2(10))
print(mt.log2(16))
```

15.dist():
========

this function used to find the Euclidian distance between given

two points

example:
=======

```
import math as mt
print(mt.dist((1,2), (4,5)))
```

print(mt.dist([5],[6]))

math module also provides, the functions which are related to

trigonometry:
==================================================

sin(), cos(), tan()

asin(), acos(), atan()

sinh(), cosh(), tanh()

2.random:
========

this module is used to "generate the random numbers"

to import the random module, we will use the following syntax:

import random as rd

in this, module we will have the following functions:
=========================================

1.random():
==========

this function will give the random number in between 0 and 1

this function will give the random number  always "real number

or floating-point number"

example:
=======

```
import random as rd
print(rd.random())
```

example-02:
==========

```
import random as rd
print(int(rd.random()*10))
print(int(rd.random()*100))
print(int(rd.random()*1000))
print(int(rd.random()*10000))
```

2.randint():
=========

this function will give the random number between the given

range

example:
=======

```
import random as rd
print(rd.randint(10,100))
print(rd.randint(1000,2000))
print(rd.randint(10000,20000))
```

3.randrange():
============

this function will give random number and  work like "range()"

function

example:
=======

```
import random as rd
print(rd.randrange(10,100,5))
print(rd.randrange(1000,2000,100))
print(rd.randrange(10000,20000,500))
```

4.shuffle():
=========

this is used on the mutable sequence type data and using this

we can shuffle or re-arrange the data in random order

example:
=======

```
import random as rd
l1=[10,20,30,40,50,60,70,80,90,100]
rd.shuffle(l1)
print(l1)
```

5.sample():
=========

this is used get sample of data from the given mutable

sequence  based on the given sample size

example:
=======

```
import random as rd
l1=[10,20,30,40,50,60,70,80,90,100]
l2=rd.sample(l1,3)
```

```
print(l2)
l1=rd.sample(l1,6)
print(l1)
```

generate a passcode and which will have both "letters" and

"digits" and the length of the passcode is always "7"
==================================================

code:
====

```
import random as rd
length=7
passcode=''
while True:
    if length%2!=0:
        passcode+=chr(rd.randrange(97,122))
        length-=1
    else:
        passcode+=chr(rd.randrange(48,58))
        length-=1
    if length==0:
        break
print(passcode)
```

generate the password and which will meet the following

conditions:

password length must be 15

password atleast have one alphabet

password atleast have 4 digits

password atleast have 3 special chracters (#,$,^,&,_)

password must starts with alphabet and ends with digit

code:
=====

```
import random as rd
length=13
passcode=''
special_string="#$^&_"
times=0
while True:
     passcode+=chr(rd.randrange(97,122))
     if length>0:length-=1
     passcode+=special_string[rd.randrange(0,len(special_string))]
     if length>0:length-=1
     passcode+=chr(rd.randrange(48,58))
     if length>0:length-=1
```

```
    times+=1
    if length==0:
        break
print(passcode)
print(times)
```

A website requires the users to input username and password to register. Write a Python Program to check the validity of password input by users.

Following are the criteria for checking the password:

1. At least 1 letter between [a-z]

2. At least 1 number between [0-9]

3. At least 1 letter between [A-Z]

4. At least 1 character from [$#@]

5. Minimum length of transaction password: 6

6. Maximum length of transaction password: 12

3.time:
================================================

this module, will give the functions related to "time":
=========================================

1.time():
=======

this function will the time in seconds

2. sleep():
========

this function will be used "to make the python interpreter

into idle for the given time slice"

example:
=======

```
import time as t
print(t.time())
for i in range(1,6):
    print(i)
    t.sleep(3)
```

3.localtime():
==========

this function is also gives the time for given  seconds

example:
=======
```
import time as t
time=t.localtime(t.time())
print(time.tm_mday)#month date
print(time.tm_mon)#month name
print(time.tm_year)#year
print(time.tm_hour)#hour
print(time.tm_min)#minutes
print(time.tm_sec)#seconds
print(t.ctime(t.time()))
```

4.ctime() :
========

this function is gives the time as a "string"

example:
=======
```
import time as t
print(t.ctime(t.time()))
```

5.perf_counter():
=============

this is used to calculate the time which is taken processor

to execute the given code

example:
========

```
import time as t
start=t.perf_counter() #start_time
for i in range(1,10):
    t.sleep(0.3)
end=t.perf_counter() #end-time
print(f"total time taken:{end-start}")
```

6.strftime():
=========
using this , we can format the given time as a string

example:
=======
```
import time as t
print(t.strftime("%Y-%m-%d",t.localtime(t.time())))
print(t.strftime("%H:%M:%S",t.localtime(t.time())))
print(t.strftime("%Y-%m-%d %H:%M:%S",t.localtime(t.time())))
```

7.strptime():
=========
using this, we can parse the given date and time

example:
=======

```
import time as t
time1="2024-5-5 1:50:00"
result=t.strptime(time1,"%Y-%m-%d %H:%M:%S")
print(result)
result=t.strptime("2024-5-5","%Y-%m-%d")
print(result)
result=t.strptime("15:45:58","%H:%M:%S")
print(result)
```

4.os module:
==========

using this module, we can able work with directories and

files

this module will provide the various functions related to

directories and files

in this module, we will have the following some important

functions related to directory:
================================================

mkdir():   ====> make directory
======

this function is used to create the "directory or folder"

makedirs()  ====>make directories
======

this function is used to create the" directory and along with

all sub directories"

chdir(): ===> change directory
======

this function is used to change the current working directory

getcwd(): ====>get the current working directory
=======

this function is used to "know the current working directory"

rmdir():====>remove directory
=====

this function is used to remove the directory

rmdirs(): ====>remove directories
======

this function is used to remove the "directories"

rename():
=======

this function is used to "rename the directory"

remove():
=======

this function is used to remove the file in the current working

directory

os.path.join():
===========

this is used to joing the given file or path with another path

os.path.exists():
=============

this is used to check the given path is valid or invalid, it

will return True(if path is valid) or False(if path is invalid)

os.path.isdir():
===========

this is used to check the given directory is in the current path

or not

os.path.isfile():
============

this is used to check the given file is in the current path

or not

example:
=======

```
import os
"""get the current working directory"""
print(os.getcwd())
"""create a directory using mkdir()"""
os.mkdir("sample1")
```

example-2:
=========

```
import os
print(dir(os))
"""get the current working directory"""
print(os.getcwd())
"""create a directoryes using mkdir()"""
os.makedirs("mysample/mysubsample/sample")
```

example-3:
=========

```
import os
"""get the current working directory"""
print(os.getcwd())
"""change the directory path"""
os.chdir("D:/training/360digrii/python_fp6_2024/python_practical/mysample/mysubsample")
print(os.getcwd())
"""create the directory"""
os.mkdir("mysample2")
```

example-4:
=========

```
import os
"""getting the current working path"""
print(os.getcwd())
"""create the directory in the current working path"""
os.mkdir("mysample3")
"""rename the directory in the current working path"""
os.rename("mysample3","my_sample4_new")
```

example-5:
========

```
import os
"""getting the current working path"""
current_path=os.getcwd()
print(current_path)
"""get the all directories in the current path"""
print(os.listdir(current_path))
"""retriving the only directories in the current path"""
result=[mydir for mydir in os.listdir(current_path)
        if os.path.isdir(os.path.join(current_path,mydir))]
print(result)
"""retriving the only files in the current path"""
result=[mydir for mydir in os.listdir(current_path)
        if os.path.isfile(os.path.join(current_path,mydir))]
print(result)
```

exmple-6:
========

```
import os
```

```
"""change the my current path"""
os.chdir("C:/Users/saira/Downloads")
"""getting the current working path"""
current_path=os.getcwd()
print(current_path)
"""list the all files,directories in the given path"""
result=os.listdir(current_path)
print(result)
"""
display only text files count
"""
filecount=0
for i in os.listdir():
    if ".txt" in i:
        print(i)
        filecount+=1
print(f"total text files count:{filecount}")
```

example-7:
=========

```
import os
"""change the my current path"""
os.chdir("D:/training/360digrii/python_fp6_2024/python_practical")
"""getting the current working path"""
current_path=os.getcwd()
print(current_path)
"""remove the empty directory,we will use rmdir()"""
os.rmdir("my_sample4_new")
```

example-8:
========

```
import os
import shutil as sh
"""change the my current path"""
os.chdir("D:/training/360digrii/python_fp6_2024/python_practical")
"""getting the current working path"""
current_path=os.getcwd()
print(current_path)
"""remove the non-empty directory,we will use sh.rmtree()"""
sh.rmtree("mysample")
```

example-9:
=========
write a python program , to create the directory with "20 sub

directories" for particular path:
====================================================

```
import os
"""change the my current path"""
os.chdir("D:/training/360digrii/python_fp6_2024/python_practical")
"""getting the current working path"""
current_path=os.getcwd()
```

```
print(current_path)
"""create the directory"""
os.mkdir("mysample")
"""change the my current path"""
os.chdir("D:/training/360digrii/python_fp6_2024/python_practical/mysample")
total_dir=int(input("enter the number of directories:"))
for i in range(1,total_dir+1):
    os.mkdir("sample"+str(i))
```

example-10:
==========

write a python program to remove only empty directories from

the given directory path:
==================================================

code:
====

```
import os
"""change the my current path"""
os.chdir("D:/training/360digrii/python_fp6_2024/python_practical/mysample")
"""getting the current working path"""
current_path=os.getcwd()
print(current_path)
"""print list of directories"""
for i in os.listdir():
    if len(os.listdir(current_path+"/"+i))==0:
        os.rmdir(current_path+"/"+i)
```

exampel-11:
==========
write a python program to count "the number of directories

are available in the each directory"
==================================================

```
import os
"""change the my current path"""
os.chdir("D:/training/360digrii/python_fp6_2024/python_practical/sample1")
"""getting the current working path"""
current_path=os.getcwd()
print(current_path)
"""print list of directories"""
for i in os.listdir():
    print(i,"total count:",len(os.listdir(current_path+"/"+i)))
```

working with files:
==============

FILE means " auxiliary memory location name"

file is created by the "operating system"

in Python, we will perform the following file operations:
===========================================

1.opening a file:
=============

when we want to perform any file

operation(read/write/append), first we need to open the file

2.read a file:
==========

using read operation on the file, we can view the content of the

file

3.write a file:
==========

using write operation on the file, we can add the content into

the file

in python, file write operation means "overwriting the file"

(it means it will remove the previous content, then the add the

new content)

4.append a file:
============

using append operation on the file, we can add the content into

the file

in Python, when we perform the append operation, the new

content we will added at the end of the file

5.close a file:
===========

after doing all operations, we need to close the file

in python, when we want to work with files, we will use the

following functions:
================================================

1.open() <=== it is used to open the "file"

note:
====

when we are opening the file, we need to open the file  with

some mode , the mode of the file describes the "what kind of

operation we are performing on the file"

in general we will have the following modes:
===================================

r =====> this mode is for "to read the file"

w =====> this mode is for "to write the file"

a =====> this mode is for "to append the file"

r+ ====>this mode is for "to read and  write the file"

w+ ====>this mode is for "to write and read the file"

a+====>this mode is for "to append and read the file"

in order open or work with binary file, we will use the following

modes:
==================================================

rb ====>this is used to "read binary file"

wb====>this is used to "write the binary file"

ab====>this is used to "append the binary file"

rb+ ====>this is used to "read and write the binary file"

wb+ ====>this is used to "write and read the binary file"

ab+====>this is used to "append and write  the binary file"

2.close()<=== it is used to close the "file"

3.read()<=== it used to "read the data from the file in a

character by character"

4.readline()<=== this is used to read the file data line by line

5.readlines()<=== this is used to read the file data line by line

and it will give the result in the form of list

note:
====

when we are performing the read operation, file need to be

there in the system, if the file is not available in system ,while

doing read operation, python will raises an error called

"FileNotFoundError"

6.write()<=== this is used to perform both append and

write operations on the file

when we wan to perform write, open the file with "w" mode

when we want to perform append, open the file with "a" mode

when we want to perform any write or append operation,

file need not to created or need not to be there in the system,

even though if file is not available while doing write or append,

operation, python will create the file automatically, and then

write or append operation

if the file is available while doing the write operation, python

simply performs the "overwrite" operation

if the file is available while doing the append operation, python

simply performs the "append" operation at the end of the file

when the file is empty, both write and append operations are

same

7.seek() <=== this used to set the "file pointer" in file

8.tell()<=== this used to know the file pointer position in the

file

file handling code template:(with out exception handling)
==========================================

```
"""open the file"""
fp=open("file_name.format","mode")
```

```
"""perform the operation(read/write/append)"""
here write the code related for operation on file

"""close the file"""
fp.close()
```

file handling code template:(with  exception handling)
============================================

```
try:
      """open the file"""
      fp=open("file_name.format","mode")

    """perform the operation(read/write/append)"""
     here write the code related for operation on file

except exception_name1:
                              #write the code here

except exception_name2:
                               #write the code here

.
.

except:

                                #write the code here

finally:

      """close the file"""
        fp.close()
```

example-1:
=========

```
"""write operation on the file using
python"""
"""open the file"""
fp=open("sample.txt","w")
"""add the content into file using write() function"""
fp.write("this is the content is added into the file using write")
"""close the file"""
fp.close()
```

example-2:
=========

```
"""write operation on the file using
python"""
"""open the file"""
try:
   fp=open("sample.txt","w")
```

```
   """add the content into file using write() function"""
   fp.write("this is the content is added into the file using write")
except FileNotFoundError:
    print("please check file is there or not")
except:
    print("unable to perform read operation")
finally:
   """close the file"""
   fp.close()
```

example-3:
=========

```
"""working with file(Write operation on file

which is not avalible in the given path)
"""
try:
    """open the file"""
    fp=open("sample2.txt","w")
    fp.write("this is the content is added")
except FileNotFoundError:
    print("the file is not found in the current path")
except FileExistsError:
    print("file is not define with given name")
except:
    print("file is not writeable")
finally:
    fp.close()
```

example-4:
========

```
"""working with file(Write operation on file

which is not avalible in the given path)
"""
try:
    """open the file"""
    fp=open("sample.txt","w")
    fp.write("this is the content is added")
except FileNotFoundError:
    print("the file is not found in the current path")
except FileExistsError:
    print("file is not define with given name")
except:
    print("file is not writeable")
finally:
    fp.close()
```

example-5:
=========

```
"""working with file(append operation on file
which is avalible in the given path)
"""
```

```
try:
    """open the file"""
    fp=open("sample.txt","a")
    fp.write("this is the content is added")
except FileNotFoundError:
    print("the file is not found in the current path")
except FileExistsError:
    print("file is not define with given name")
except:
    print("file is not writeable")
finally:
    fp.close()
```

example-6:
=========

```
"""working with file(read operation on file
which is avalible in the given path)
"""
try:
    """open the file"""
    fp=open("sample.txt","r")
    print(fp.read())
except FileNotFoundError:
    print("the file is not found in the current path")
except FileExistsError:
    print("file is not define with given name")
except:
    print("file is not writeable")
finally:
    fp.close()
```

example-7:
=========

```
"""working with file(read operation on file
which is avalible in the given path)
"""
try:
    """open the file"""
    fp=open("sample.txt","r")
    #read only 10 chracters from the file
    print(fp.read(10))
except FileNotFoundError:
    print("the file is not found in the current path")
except FileExistsError:
    print("file is not define with given name")
except:
    print("file is not writeable")
finally:
    fp.close()
```

exmaple-8:
=========

```
"""working with file(read operation on file
which is avalible in the given path)
"""
try:
    """open the file"""
    fp=open("sample.txt","r")
    """we are finding the file pointer position"""
    print(fp.tell())#0
    print(fp.read(5))#this
    print(fp.tell())#5
    print(fp.read(5))#
except FileNotFoundError:
    print("the file is not found in the current path")
except FileExistsError:
    print("file is not define with given name")
except:
    print("file is not writeable")
finally:
    fp.close()
```

example-9:
=========

```
"""working with file(read operation on file
which is avalible in the given path)
"""
try:
    """open the file"""
    fp=open("sample.txt","r")
    print(fp.readline())
    print(fp.readline())
    print(fp.readline())
except FileNotFoundError:
    print("the file is not found in the current path")
except FileExistsError:
    print("file is not define with given name")
except:
    print("file is not writeable")
finally:
    fp.close()
```

example-10:
==========

```
"""working with file(read operation on file
which is avalible in the given path)
"""
try:
    """open the file"""
    fp=open("sample.txt","r")
    """read only 5 characters using readline()"""
    print(fp.readline(5))
    print(fp.tell())
except FileNotFoundError:
```

```
    print("the file is not found in the current path")
except FileExistsError:
    print("file is not define with given name")
except:
    print("file is not writeable")
finally:
    fp.close()
```

example-11:
==========

```
"""working with file(read operation on file
which is avalible in the given path)
"""
try:
    """open the file"""
    fp=open("sample.txt","r")
    print(fp.readlines())
except FileNotFoundError:
    print("the file is not found in the current path")
except FileExistsError:
    print("file is not define with given name")
except:
    print("file is not writeable")
finally:
    fp.close()
```

exmaple-12:
==========

```
"""working with file(read operation on file
which is avalible in the given path)
"""
try:
    """open the file"""
    fp=open("sample.txt","r")
    print(fp.readlines(40))
except FileNotFoundError:
    print("the file is not found in the current path")
except FileExistsError:
    print("file is not define with given name")
except:
    print("file is not writeable")
finally:
    fp.close()
```

example-13:
==========

```
"""working with file(read operation on file
which is avalible in the given path)
"""
try:
    """open the file"""
    fp=open("sample.txt","r")
```

```
    print(fp.tell())
    """we can set the file pointer position"""
    fp.seek(5)
    print(fp.tell())
    print(fp.read())
except FileNotFoundError:
    print("the file is not found in the current path")
except FileExistsError:
    print("file is not define with given name")
except:
    print("file is not writeable")
finally:
    fp.close()
```

with keyword in Python:
===================

using with keyword in python, we can able to "open and close

the any resource automatically"

while working with files, database connections, network

connections,....................

syntax for with keyword:
===================

```
with open("file_name.format","mode") as filepointer:

        #code
```

example:
=======

```
with open("sample.txt","r") as fp:
    print(fp.read(5))
#fp.read(5)
```

pickling and un-pickling in python:
===========================

pickling means "converting the python object into byte stream"

pickling is also called as "serialization or marshalling"

un-pickling means "converting the byte stream into python

object"

un-pickling is also called as "de-serialization or unmarshalling"

pickling is always safe process and recommended when we

are sending data over a network

un-pickling is always un-safe process due to , if we may got

the data from "un-authorized or un-known" source, un-pickling

is always safe, if we know the source

to work with pickling and un-pickling, we will use a module

called "pickle"

this module will provides, the following functions:
========================================

1.dump()<=== to perform the pickling(Serialization)

2.load()<==== to perform the un-pickling(de-Serialization)

example:
=======

```
import pickle as ple
import os
"""to know the current working path"""
print(os.getcwd())
"""implementing the pickle process (Serializzation)"""
with open("sample.pkl","wb") as fp:
    data={"name":"ram","location":"hyderabad","email":"Ram@gmail.com"}
    """to implement the pickling, pickle module provides
    a function called dump()"""
    ple.dump(data,fp)
```

example-2:
=========

```
import pickle as ple
import os
"""to know the current working path"""
print(os.getcwd())
"""implementing the un-pickle process (de-Serializzation)"""
with open("sample.pkl","rb") as fp:
   """to implement the unpickling, pickle module provides
    a function called load()"""
   data=ple.load(fp)
   print(data)
```

working with json data:
===================

json is used to store the data, in the form  of  key and value

pair and basically it javascript object structure

json is stands for "javascript object notation"

this format data is very popular in between the server to client

or client-server communication, json is way we will use  to send

the data via network

syntax for json data structure:
========================

{key1:value1,key2:value2,key3:value3,.........}

example:
=======

{name:"Ram",location:"hyderabad"};

in order to work with json data, python provides the a module

called json

this module, we will provides the following functions:
=========================================

dumps()===>used to convert python dictionary into json data

loads()====>used to convert the json data into python dict.

dump()===>to write the python dic. data into json file,we will

use dump()

load()====>to read the any data from jsonfile into pyton, we

will use load() function

note:
====

in python json data is "assumed as string format"

exmaple-1:
=========

```
import json
d1={"name":"ram","location":"hyderabad"}
print(type(d1))
"""convert the d1 into json using dumps()"""
d1=json.dumps(d1)
print(d1)
print(type(d1))
"""convert the d1 json data into d1 dictionary using
loads()"""
d1=json.loads(d1)
print(d1)
print(type(d1))
```

example-2:
=========

```
import json
d1={"name":"ram","location":"hyderabad"}
"""write the data into file as data.json using
dump() function"""
with open("data.json","w") as fp:
    json.dump(d1,fp)
```

example-3:
=========

```
import json
"""write the data into file as data.json using
dump() function"""
with open("data.json","r") as fp:
    d1=json.load(fp)
    print(d1)
```

5.datetime:
=========

in python ,datetime module is used to work with date and time

this module, will provide the following classes:
====================================

1. date class:
============

this class will give the information, about the

date(year, month and day number)

example-1:
========

```
"""working with datetime module, in that we are working with
date class"""
from datetime import date
d1=date(2024,8,10)
```

print(d1)
"""to print the todaya data, we will have a function called today() function"""
d1=date.today()
print(d1)
print(d1.day)#day number
print(d1.month)#month number
print(d1.year)#year number

example-2:
=========

"""working with datetime module, in that we are working with date class"""
from datetime import date
d1=date(2024,8,10)
print(d1)
print(d1.day)#day number
print(d1.month)#month number
print(d1.year)#year number
"""to print the todaya data, we will have a function called today() function"""
d1=date.today()
print(d1)
print(d1.day)#day number
print(d1.month)#month number
print(d1.year)#year number

2. time class:
===========

this class will give the information, about the

time(hour, minute, second and microsecond)

example:
=======

"""working with datetime module, in that we are working with time class"""
from datetime import time
t1=time(23,45,45,1200)
print(t1)
print(t1.hour)
print(t1.minute)
print(t1.second)
print(t1.microsecond)

3.datetime class:
=============

this class used to handle the both date and time at a time

this class will have the attributes like year, month, day, Hour, minute, second, microsecond

example:
=======

```
"""working with datetime module, in that we are working with
datetime class"""
from datetime import datetime
dt1=datetime(2024,10,4,23,45,45,1200)
print(dt1.year)
print(dt1.month)
print(dt1.day)
print(dt1.hour)
print(dt1.minute)
print(dt1.second)
print(dt1.microsecond)
"""to get the both current date and time"""
dt1=datetime.now()
print(dt1.year)
print(dt1.month)
print(dt1.day)
print(dt1.hour)
print(dt1.minute)
print(dt1.second)
print(dt1.microsecond)
```

4.timedelta class:
===============

using this class we can add and subtract dates

example-1:
=========

```
"""working with datetime module, in that we are working with
datetime class"""
from datetime import date,timedelta
"""today date using date.today()"""
d1=date.today()
"""add the 10 days to the today date"""
print(d1+timedelta(days=10))
"""add the 10 months to the today date"""
print(d1+timedelta(days=300))
"""add the 100 hours to the today date"""
print(d1+timedelta(hours=100))
"""add the 1000 minutes to the today date"""
print(d1+timedelta(minutes=1000))
"""subtract the 10 days to the today date"""
print(d1-timedelta(days=10))
"""subtract the 10 months to the today date"""
print(d1-timedelta(days=300))
"""subtract the 100 hours to the today date"""
print(d1-timedelta(hours=100))
"""subtract the 1000 minutes to the today date"""
```

print(d1-timedelta(minutes=1000))

example-2:
=========

```
"""working with datetime module, in that we are working with
datetime class"""
from datetime import date,timedelta
"""Calcuate the difference of the two dates"""
d1=date(2024,10,30)
d2=date(2023,11,30)
print(d1-d2)
"""convert the days into years"""
print(round((d1-d2).days/366))
result=d1-d2
print(result.days)
```

write a python program calculate the  age of the person:
============================================

code:
====

```
"""working with datetime module, in that we are working with
datetime class"""
from datetime import date,timedelta
"""Calcuate the age of the person"""
d1=date(2003,4,12)
d2=date.today()
print(round((d2-d1).days/365),"years")
```

write a python program to "find the difference of the given

two dates from user as a input and find the difference in

hours"

=================================================

code:
====

```
"""working with datetime module, in that we are working with
datetime class"""
from datetime import date,timedelta
"""Calcuate the age of the person"""
daynumber=int(input("day number"))
month=int(input("month:"))
year=int(input("year:"))
d1=date(year,month,daynumber)
d2=date.today()
print(round(((d2-d1).days/365)*24),"hours")
```

6.sys:
=====

sys module is used to in python, to work with system related

operations:
================================================

1.sys.version:
===========

it will give the python interpreter version

2.sys.platform:
============

it will give the platform of the system

example:
=======

```
"""working with sys module"""
import sys
"""the belowe will show python interpreter version"""
print(sys.version)
"""the below will show the platform name(os name)"""
print(sys.platform)
```

3.getsizeof():
===========

it is used to find the "object memory size in bytes"

example:
=======

```
"""working with sys module"""
import sys
a=10
print(sys.getsizeof(a))
b=20
print(sys.getsizeof(b))
c=1.234
print(sys.getsizeof(c))
d=-6.7890
print(sys.getsizeof(d))
s1="hello world"
print(sys.getsizeof(s1))
```

4.sys.maxsize:
==============

it will show the maximum integer number allow by the python

interpreter

example:
========
```
"""working with sys module"""
import sys
print(sys.maxsize)
```

5. sys.getrecurssionlimit()
====================

using this we get the "recursion limit of the python interpreter"

it means "how many times a function calls itself" determines

recursion limit

in python, using sys module we can set and get the recursion

limit

6.sys.setrecursionlimit():
===================

using this this" we can set the recursion depth of the

python interpreter"

example:
========

```
"""working with sys module"""
import sys
print(sys.getrecursionlimit())
"""set the recursion limit using
setrecursionlimit() function of sys module"""
"""create a recursion function"""
sys.setrecursionlimit(55)
print(sys.getrecursionlimit())
def display(num,times):
    times+=1
    if num<=60:
        display(num+1,times)
display(1,0)
```

7.sys.getrefcount():
===============

```
"""working with sys module"""
```

```
import sys
a=10
b=20
c=30
f=40
print(sys.getrefcount(a))
print(sys.getrefcount(b))
print(sys.getrefcount(c))
print(sys.getrefcount(f))
result=f+10
d=a
print(sys.getrefcount(a))
e=b
print(sys.getrefcount(b))
c=10
print(sys.getrefcount(c))
print(sys.getrefcount(a))
print(sys.getrefcount(f))
```

8.sys.argv:
========

in python, we can run the python program on command line

when we are running the python program on the command

line, if we pass any arguments, then the arguments are called

as "command-line arguments"

argv is represents in python "command line arguments",

when we are running any program on command line, at that

time if we pass any argument, then the arguments are called as

"command-line arguments"

in python command line arguments can be represent as "list"

format

while running the python program on the command line, even

filename is also considered as "command line argument"

every command line argument type is "string" type

```
"""working with command line arguments"""
import sys
print(sys.argv)
for i in sys.argv:
    print(f"the value of {i} and it’s type is:{type(i)}")
```

example:
=======

```
"""working with sys module"""
import sys
print(sys.argv)
```

example-2:
========

```
"""working with command line arguments"""
import sys
print(sys.argv)
for i in sys.argv:
    print(f"the value of {i} and it’s type is:{type(i)}")
```

working with OOPS in Python:
=======================

python is a "object oriented programming language", if any

programming language is "object-oriented", then the language

must follow the following:
=================================================

1. class and object

2. inheritance

3. Data Encapsulation

4. Data Abstraction

5. Polymorphism

python supports above all 5 features

class and objects in python:
======================

class is a "collection of data and methods"

or

class is a "collection of related objects"

in python, the class can have "both data and method" as

members

data in a class represents ===>variable, list, tuple, string,....

methods in a class represents ===> function

to create the class in python, we will use the following syntax:
================================================

class class_name:

    #data(variable, list, tuple, set, string,.............)

    #methods (Function)

note: where class is a python keyword and using this keyword

        we can create the class in python

when we create the class, all class members(data and methods)

can accessed via "object"

object is nothing "instance of a class"  or "class type variable"

object is a physical entity and which is exists in the real world

when we create the class with properties, the class will never

memory

the class properties will get memory only via "object"

to create the object for a class ,we will use the following syntax

in python:

object_name=class_name()

note:
====

any class member( data / methods) can accessed via "object"

in general

using following syntax, we can access the any member of the

class via object in python

object_name.data_name or method_name()

example:
=======

```
"""working with class in python"""
class Sample:
    #data
    x=10
    y=20
```

```
    z=30
    l1=[1,2,3,4,5]
    #method
    def display(self):
        print("this is my first class in python")
"""create the object to the Sample class"""
s1=Sample()
print(s1.x)
print(s1.y)
print(s1.z)
s1.display()
```

when we create any method in the class , it will have a default

argument called "Self", this is not a pre-defined name, we can

take any name, for the standard we will use "self"

using this "self", we can access the same class data or method

in the  another method of the same class

example:
=======

```
"""working with class in python"""
class Sample:
    #data
    x=10
    y=20
    z=30
    l1=[1,2,3,4,5]
    #method
    def display(self):
       print(self.x)
       print(self.y)
       print(self.z)
    def display2(self):
         self.display()
s1=Sample()
print(s1.x)
print(s1.y)
print(s1.z)
s1.display2()
```

in a Python class, we can have the following types of methods:
==================================================

1.instance method:
===============

instance method is a method and which can access only "object

of the class"

2.class method:
=============

class method is a method and which can access by the class

name or object of the class

to create the class method in python, we will use a built-in

decorator called "@classmethod"

3.static method:
=============

static method is a method and which can access by the class

name but not the object

to create the static method in python, we will use a built-in

decorator called "@staticmethod"

4.abstract method:
===============

abstract method is a method and which is does not any code

and the code of the abstract method can be given by sub

classes  of the class where abstract method is present

if a class is having abstract method, then the class is called as

"abstract class"

to create the abstract method in python class, we will use a

decorator called "@abstractmethod"  and which is present  in

abc module

exmaple-01:
```
=========="""working with various methods of the class
in python"""
class Sample:
    @classmethod
    def display(cls):
        print("this is my class method")
    @staticmethod
    def display2():
        print("this is my static method")
    #instance method
    def display3(self):
        print("this is instance method")
```

```
s1=Sample()
s1.display() #here we called display using object
Sample.display() #here we are called display using class name
s1.display2()
Sample.display2()
s1.display3()
```

exmaple-2:
=========

```
 """working with static, class, instance methods"""
class Sample:
    #class data
    x=10
    y=20
    z=30
    #class method
    @classmethod
    def myclassmethod(cls):
        print(cls.x)
        print(cls.y)
        print(cls.z)
   #instance method
    def myinstancemethod(self):
       print(self.x)
       print(self.y)
       print(self.z)
Sample.myclassmethod()
s1=Sample()
s1.myinstancemethod()
```

note:
====

class method and instance method can access the "Class data"

in python

static mehod can not able to access the "class data"

constructor in python:
=================

constructor is a method in python and which is used in python

to initialize the data of the object

the data which is initialize by the constructor is called as

"instance data" and this data can be accessed only by the

object of the class

constructor can be called automatically , when we create the object to the class

constructor will never be called "Explicitly", by the programmer

in python program

in python, we will have two types of constructors:
=======================================

1. default constructor:
==================

default constructor is a constructor and which is does not have

any arguments or does not take any arguments

2. parameterized constructor:
=======================

parameterized constructor is a constructor and which is going

take arguments or constructor with arguments can be called

as "parameterized constructor"

in python, to create the constructor, we will use a pre-defined

name called "__init__"

example-1:
=========

```
"""working with default constructor"""
class Sample:
    """default constructor"""
    def __init__(self):
        print("this is default constructor")
s1=Sample()
```

example-2:
=========

```
"""working with parameterized constructor"""
class Sample:
    """parameterized constructor"""
    def __init__(self,a,b,c):
        print("this is parameterized constructor")
        print(a)
        print(b)
        print(c)
s1=Sample(10,20,30)
```

example-3:
=========

```
"""working with parameterized constructor"""
class Sample:
    """here a1,b1,c1 are class data"""
    a1=100
    b1=200
    c1=300
    """parameterized constructor"""
    def __init__(self,a,b,c):
        """here a,b c are parameters of the
        parameterized constructor"""
        """here we creating the instance data"""
        self.x=a
        self.y=b
        self.z=c
s1=Sample(10,20,30)
print(s1.x)
print(s1.y)
print(s1.z)
"""in python, to know the instance data and instance
methods of the class, we will use vars()"""
print(vars(s1))
```

note:
====

in a class, we can have only "one constructor" (it can be either

default or parameterized constructor)

in a python, in general, constructor can be called automatically

when we create the object and in python we can also call the

constructor, using object of the class also(but it not

recommended)

in python, when we create the any data using "constructor",

then the data can be called as "instance data"

in python, to know the instance data and instance

methods of the class, we will use vars()

in python to know the any class data and methods or any class

members, we will use a function called "dir()"

using vars(), we can retrieve the data of the class using "class

name or object name"

example:
=======

```
"""working with parameterized constructor"""
class Sample:
    """here a1,b1,c1 are class data"""
    a1=100
    b1=200
    c1=300
    def display(self):
        print("this is instance method")
    @classmethod
    def display2(cls):
        print("this is class method")
    @staticmethod
    def display3():
        print("this is static method")

    """parameterized constructor"""
    def __init__(self,a,b,c):
        """here a,b c are parameters of the
        parameterized constructor"""
        """here we creating the instance data"""
        self.x=a
        self.y=b
        self.z=c
s1=Sample(10,20,30)
"""here vars() will give only instance data
 of the class"""
print(vars(s1))
"""here dir() will give both class data and instance
data"""
print(dir(s1))
```

example:
=======

```
"""working with parameterized constructor"""
class Sample:
    """here a1,b1,c1 are class data"""
    a1=100
    b1=200
    c1=300
    def display(self):
        print("this is instance method")
    @classmethod
    def display2(cls):
        print("this is class method")
    @staticmethod
    def display3():
        print("this is static method")

    """parameterized constructor"""
    def __init__(self,a,b,c):
        """here a,b c are parameters of the
```

```
        parameterized constructor"""
        """here we creating the instance data"""
        self.x=a
        self.y=b
        self.z=c
s1=Sample(10,20,30)
"""here vars() will give only instance data
 of the class"""
print(vars(Sample))
"""here dir() will give both class data and instance
data"""
print(dir(Sample))
```

inheritance in python:
=================

inheritance allows the programmer can give the "properties

or members of the one class to another class"

the class which is taking the properties of another class, is

known as "child class or sub class or derived class"

the class which is giving the properties of another class, is

known as " parent class or super class or Base class"

in general, we will have three types of inheritance:
=======================================

1.single inheritance or mono inheritance:
================================

in this, only one class(child class or sub class)  can take the

properties from the another class(parent class or super class)

2.multiple inheritance:
==================

in this, one class can take  properties "Two or more super

classes"

3.multi-level inheritance:
===================

in this, one class will give the properties to another class, the

same class will give the properties to another class, and so on..

to apply the inheritance to the class in python, we will use

the following syntax:
====================================================

class class_name(super_class_name1,super_class_name2....):

    #data

    #method

note:
====

Python supports the "multiple inheritance"

when we are working with inheritance, we will use child class

object to access the both parent class and child class members

instead of creating the separate object for both classes

in child class constructor, if we want to call the "parent class"

constructor, we will use "super()" function in python

in child class method, if we want to access the parent class

data or methods, we will use "super()" function

super() is a built-in function and using this function we can

access the "parent class data or methods" inside the child class

example-1:
=========

```
"""working with single inheritance"""
class A:
    #data
    a,b,c=10,20,30
    #method
    def display(self):
        print("this method is from class A")
print(dir(A))
class B(A):
    #data
    x,y,z=100,200,300
    #method
    def display2(self):
        print("this method is from class B")
print(dir(B))
```

```
"""creating the object for class B"""
b1=B()
print(b1.a,b1.b,b1.c)
print(b1.x,b1.y,b1.z)
b1.display()
b1.display2()
```

exmaple-2:
=========

```
"""working with multiple inheritance"""
class A:
    #data
    a,b,c=10,20,30
    #method
    def display(self):
        print("this method is from class A")
class B:
    #data
    x,y,z=100,200,300
    #method
    def display2(self):
        print("this method is from class B")
class C(A,B):
    def display3(self):
        print("this method is from class C")
print(dir(C))
"""create the object for the class C"""
c1=C()
c1.display()
c1.display2()
c1.display3()
```

example-3:
=========

```
"""working with multi-level inheritance"""
class A:
    #data
    a,b,c=10,20,30
    #method
    def display(self):
        print("this method is from class A")
class B(A): #B===>A+B
    #data
    x,y,z=100,200,300
    #method
    def display2(self):
        print("this method is from class B")
class C(B):#C===>A+B+C
    def display3(self):
        print("this method is from class C")
print(dir(C))
"""create the object for the class C"""
```

```
c1=C()
c1.display()
c1.display2()
c1.display3()
```

how to call the super class method in the child class method:
=================================================

to call the super class method in the child class method, in

Python we will use "super()" method

example-1:
=========

```
"""working with multi-level inheritance"""
class A:
    #method
    def display(self):
        print("this method is from class A")
class B(A): #B===>A+B
    #method
    def display2(self):
        super().display()
        print("this method is from class B")
class C(B):#C===>A+B+C
    def display3(self):
        super().display2()
        print("this method is from class C")
print(dir(C))
"""create the object for the class C"""
c1=C()
c1.display3()
```

how to call the super class constructor inside the child class

constructor in python:
==================================================

to call the super class constructor inside the child class

constructor in python, we will use "super()" method

example:
=======

```
"""working with multi-level inheritance"""
class A:
    #create the constructor
    def __init__(self):
        print("this constructor is from class A")
class B(A):
    #create the constructor
    def __init__(self):
```

```
        super().__init__()
        print("this is constrcutor is from class B")
"""create the object for B class"""
b1=B()
```

example-2:
=========

```
"""working with multi-level inheritance"""
class A:
    #create the parameterized constructor
    def __init__(self,a,b):
        print("this constructor is from class A")
        print(a,b)
class B(A):
    #create the parameterized constructor
    def __init__(self,x,y,z):
        super().__init__(x,y)
        print("this is constrcutor is from class B")
        print(x,y,z)

"""create the object for B class"""
b1=B(10,20,30)
```

Abstract class:
============

Abstract class is a class and which is having at least one

abstract method

abstract method is a method and which is does not have any

implementation or no code

to create the abstract method in python, we will use a decorator

called "@abstractmethod" and this decorator is from a module

called "abc"

in python, when we say any class is "Abstract class", it must the

following:

(i) at least one abstract method

(ii) the class must take "ABC" as base class or super class

[ where ABC stands for "abstract base class" and it is

super class for all abstract classes in the Python and this is available from abc module ]

when we say any class is "abstract class" , then the abstract

methods of the that class will get implementation by it’s child

classes, otherwise child classes are also become "abstract

classes"

for an abstract class, we can not create "object"

example-1:
=========

```
"""working with abstract class"""
from abc import ABC,abstractmethod
class A(ABC):
    #create the normal method
    def display(self):
        print("this is normal method")
    #create the abstract method
    @abstractmethod
    def display2(self):
        pass
```

exmaple-2:
=========

```
"""working with abstract classes"""
from abc import ABC,abstractmethod
class Sample(ABC):
    def display(self):
        print("this is nomral method")
    def display2(self):
        print("this is nomarl method 2")
    @abstractmethod
    def display3(self):
        pass
class Sample2(Sample):
    def display3(self):
        print("this abstact method got implementation by sample 3 class")

"""create the object for Sample2"""
s2=Sample2()
s2.display3()
```

abstract method:
==============

abstract method is a method and which does not have any

implementation or code and which is defined in python using abstract method decorator

if class is having at least one abstract method, then the class

is called as " abstract class", to work with abstract classes we

will use "abc" module

concreate method:
===============

concreate method is a method which is having

"implementation" or code

if class is having  all are concrete methods, then the class is

called as "concreate class"

Data Encapsulation and Abstraction:
==============================

Data Encapsulation:
================

encapsulation allows programmer or developer

to wrap the "both data and methods as a single unit"

in Python, class is a "best example for encapsulation"

Data abstraction:
==============

it a way of "making  the class  what data to show/ share and

and what data not to show /share"

in python, the data abstraction will be done using access

specifiers:

1.public :
========

when we define the any member of the class(data/ method)

as public ,then the member can access by any class and outside

the class

in python, by default all members of the class are "public"

2.private:
========

when we define the any member of the class(data/ method)

as private ,then the member can not access by any class and

not outside the class

in python, to define any member of the class as private, we

will define with "__" , at beginning of the member name

in general , all private members can access only in the class,

where we define the "private members"

when we define  class with all private members, then the class

is called as "sealed class"

when we define the any method as "private", then the method is

called as "sealed method", sealed method can not be override

if a class is having at least one private, then the class is

called encapsulated class

3.protected:
==========

when we define the any member of the class as "protected",

then the class member accessed by "same class, and it’s sub

classes" only

in python, all protected members can act like "public members"

to define the protected members in python, we will use "_" at

starting of name of the any member in the class

example-1:
=========

```
"""working with access specifiers"""
class Sample:
    #public data
    a=100
```

```
    b=200
    #private data
    __x=100
    __y=200
    #protected data
    _a1=100
    _b1=200
"""create the object for class Sample"""
s1=Sample()
print(s1.a)
print(s1.b)
print(s1._a1)
print(s1._b1)
```

example-2:
=========

```
"""working with access specifiers"""
class Sample:
    #public method
    def display(self):
        print("this is public method")
    #private method
    def __display2(self):
        print("this is private method")
    #procted method
    def _display3(self):
        print("this is protected method")
"""create the object for class Sample"""
s1=Sample()
s1.display()
s1._display3()
```

example-3:
=========

```
"""working with access specifiers"""
class Sample:
    #private class data
    __a=100
    __b=200
    __c=300
    #display the private data using public method
    def display(self):
        print(self.__a)
        print(self.__b)
        print(self.__c)
class Sample2(Sample):
    def display2(self):
        print("this is child class")
print(dir(Sample2))
"""create the object for class Sample"""
s2=Sample2()
s2.display()
```

s2.display2()

polymorphism:
============

poly means "many"

morph means "form"

exhibiting the "many forms based on the scenario"

in python we will have two types of polymorphism:
========================================

1. static or compile time polymorphism:
===============================

this polymorphism will be exhibited at the time of "compile

time"

the best example is "method overloading and operator

overloading"

how to implement the method overloading in python:
===============================================

method overloading:
=================

method overloading means "defining the method with same

name but different in number of arguments or different in

type of arguments"

in python, we can not implement method overloading as same

as  java or C++

in python, to implement method overloading, we will use

"default arguments" with function

example:
=======

```
"""working with method overloading"""
class Sample:
    def sum_of_two_numbers(self,a=10,b=20):
        print(a+b)
s1=Sample()
```

```
s1.sum_of_two_numbers()
s1.sum_of_two_numbers(100)
s1.sum_of_two_numbers(100,200)
s1.sum_of_two_numbers(10,20)
```

2.dynamic or run-time polymorphism:
===================================

this polymorphism will be exhibited at the time of "run

time"

the best example is "method overriding"

in python to implement the method overriding, we need to

use inheritance

method overriding means "defing the method with same name

and same number of arguments" in both parent and child class

the child class method will override the parent class method

example:
=========

```
"""working with method overriding"""
class NormalIndividual():
    def interest_amount(self,fd_amount,rate):
        amount=(fd_amount)*(rate)//100
        print(f"normal individual Amount is:{amount}")
class Minor(NormalIndividual):
    def interest_amount(self,fd_amount,rate):
        amount=(fd_amount)*(rate)//100
        print(f"Minor Amount is:{amount}")
class Senior_citizen(NormalIndividual):
    def interest_amount(self,fd_amount,rate):
        amount=(fd_amount)*(rate)//100
        print(f"Senior Citizen Amount is:{amount}")
class Govt_employee(NormalIndividual):
    def interest_amount(self,fd_amount,rate):
        amount=(fd_amount)*(rate)//100
        print(f"Govt. Employee Amount is:{amount}")
m1=Minor()
m1.interest_amount(1000000,15)
```

note:
====

using method overriding, we can enhance the parent class method in child class

object class in python:
=================

object class is a  super class for all "classes" in python

every class by default will have  "object" class as "super"

class

example-1:
=========

```
class A: #it is having a super class called "object"
    pass
class B(A): #this class having super classes are A and object
    pass
class C(B): #this class having super classes are A,B and object
    pass
print(dir(A))
print(dir(B))
print(dir(C))
print(dir(object))
```

working with isinstance(), getattr(), setattr() functions:
==========================================

isinstance():
=========

using this method/function, we can check the given object is

related to given class or not

this will return the result either True or False

example:
========

```
print(isinstance(10,int))
print(isinstance(1.234,int))
print(isinstance("hello",str))
print(isinstance(10+5j,complex))
print(isinstance([1,2,3,4,5],list))
print(isinstance(1.234,float))
print(isinstance({1,2,3,4},set))
print(isinstance({},dict))
class A:
    pass
class B:
    pass
a1=A()
```

```
b1=B()
print(isinstance(a1,A))
print(isinstance(b1,A))
print(isinstance(b1,B))
```

working with getattr() and setattr():
===========================

getattr() function is used to "retrieve the instance data of the

object"

setattr() function is used to "create the instnace data of the

object"

example:
=======

```
class A:
    #data
    x=100
    y=200
    z=300
    def __init__(self,a,b,c):
        """here we are creating the instance data
         a,b,c using self and constrcutor
        """
        self.a=a
        self.b=b
        self.c=c
"""create the object for the class"""
a1=A(10,20,30)
print(vars(a1))
print(a1.a)
print(a1.b)
print(a1.c)
```

example:
=======

```
class A:
    pass
a1=A()
print(vars(a1))
"""using setattr() we can create the instace data
of object"""
setattr(a1,"a",100)
setattr(a1,"b",200)
setattr(a1,"c",300)
print(vars(a1))
"""get the instance data of the class using
getattr() function"""
print(getattr(a1,"a"))
```

```
print(getattr(a1,"b"))
print(getattr(a1,"c"))
```

example:
=======

```
class A:
    pass
a1=A()
print(vars(a1))
setattr(a1,"a",10)
setattr(a1,"b",20)
setattr(a1,"c",30)
"""get the data using getattr() using a1"""
print(getattr(a1,"a",100))
print(getattr(a1,"b",200))
print(getattr(a1,"c",300))
print(vars(a1))
```

getters and setters in python classes:
==============================

getters are used "to get the instance data of the class"

setters are used "to set the instance data of the class"

when we are working with getters and setters in python, the

data need to be "private"

when we are working  with getters and setters:

 the name of the getter will start with getdataname

 the name of the setter will start with setdataname

example:
=======

```
class A:
   def __init__(self,name,email,mobileno):
       self.__name=name
       self.__email=email
       self.__mobileno=mobileno
   """create the getters for the instance data"""
   def getname(self):
       return self.__name
   def getemail(self):
       return self.__email
   def getmobileno(self):
       return self.__mobileno
   """create the setters for the instance data"""
   def setname(self,name):
       self.__name=name
```

```
   def setemail(self,email):
       self.__email=email
   def setmobileno(self,mobileno):
       self.__mobileno=mobileno

a1=A("Ram","ram@gmail.com",9874561230)
print(a1.getname())
print(a1.getemail())
print(a1.getmobileno())
a1.setname("kumar")
a1.setemail("kumar@gmail.com")
a1.setmobileno(8523697410)
print(a1.getname())
print(a1.getemail())
print(a1.getmobileno())
```

example:
=======

```
class A:
   def __init__(self,name):
       self.__name=name
   """create the getter for name"""
   @property
   def name(self):
       return self.__name
   """create the setter for name"""
   @name.setter
   def name(self,name):
       self.__name=name
   """create the deleter for the name"""
   @name.deleter
   def name(self):
       print("the name is deleting")
       del self.__name

a1=A("Ram")
print(a1.name)
a1.name="kumar"
print(a1.name)
del a1.name
#print(a1.name)
```

note:
====
property decorator is built-in decorator and using we can create

getter, setter and delete methods

using property decorator, we can create the following:
==========================================

getter ====>to get the data or to retrieve he data for this we will use function and name of the function is same

as property name

syntax:
======

```
@property
def dataname(self):

    #logic
```

setter=====> to set the data or to update the data

syntax:
======

```
@dataname.setter
def dataname(self,value):

    #logic
```

deleter====> to delete the data from the class

```
@dataname.deleter
de dataname(self):

    #logic for deletetion
```

Inner classes in Python:
==================

inner class means "creating a class inside another class"

inner class is also called as "nested class"

in python, we can create a class inside another class

example:
=======

```
class Sample:#here Sample is main class
    #data
    x,y,z=10,20,30
    class Sample1:#here Sample1 is inner class 1
        def display(self):
            print(Sample.x)
    class Sample2:#here Sample1 is inner class 2
      def display(self):
          print(Sample.y)
```

```
    class Sample3:#here Sample1 is inner class 3
        def display(self):
            print(Sample.z)
"""create the object main class called Sample"""
s1=Sample()
"""create the object for inner class called Sample1"""
inner_s1=s1.Sample1()
inner_s1.display()
"""create the object for inner class called Sample2"""
inner_s2=s1.Sample2()
inner_s2.display()
"""create the object for inner class called Sample3"""
inner_s3=s1.Sample3()
inner_s3.display()
```

example-2:
=========

```
"""working with inner classes"""
class OuterClass:
    """create the data in outer class"""
    a,b,c=100,200,300
    """creating the inner classes"""
    class Innerclass1:
        x=10
        def display(self):
            print(self.x)
            print(OuterClass.a)

    class Innerclass2:
        y=10
        def display(self):
            print(self.y)
            print(OuterClass.b)
    class Innerclass3:
        z=10
        def display(self):
            print(self.z)
            print(OuterClass.c)
"""create the object for outer class"""
o1=OuterClass()
"""create the obejct for inner class 1"""
i1=o1.Innerclass1()
"""create the obejct for inner class 2"""
i2=o1.Innerclass2()
"""create the obejct for inner class 3"""
i3=o1.Innerclass3()
i1.display()
i2.display()
i3.display()
```

note:
====

in classes, self name used to "Access the current class data or

methods"

in inner classes, to access the outer class data or methods, we

will use outer class name.

to create the object to inner classes, we will use outer class

object name

syntax:
======

inner_class_object_name=outer_class_object_name.inner_class()

magic methods or dunder methods:
============================

magic method is a method and which is called automatically

when we are doing a specific operation in the program

magic method name always starts with "__" and ends with "__" ,

that is reason magic method also called as "dunder methods"

dunder means (double underscore)

magic methods never called explicitly like normal methods in

the classes or functions in the program

when we are doing an operation, it will have a specific magic

method in Python

the following some important magic methods in python:
============================================

1. __add__():
===========

this is a magic method and it is called automatically when

perform addition of two numbers

example:
=======

a=10, b=20 ===>print(a+b) ====>a.__add__(b)

example:
=======

```
"""working with magic methods"""
a=10
b=20
print(a.__add__(b))#a+b
print(a.__sub__(b))#a-b
print(a.__mul__(b))#a*b
```

2.__mul__():
==========

this is a magic method and it is called automatically when

perform multiplication of two numbers

3.__div__() :
==========

this is a magic method and it is called automatically when

perform division of two numbers

4.__init__() :
==========

this is a magic method and it is called automatically when

create the object to the class

5.__new__():
==========

using this we can create the a new object to class

to understand about use of the magic methods in the python,

we will use a concept called "operator overloading" :

=================================================

operator overloading means "making the operator, to perform

more than one operation, based on the object type"

in python,  we will use "+" operator for the following ways:
==============================================

1.addtion of two integer objects

2.merge the two strings

3.merge the two lists

4.merge the two tuples

use the "+" operator add two objects of the classes:
==========================================

code:
====

```
"""working with magic methods"""
class Sample:
    def __init__(self,x,y):
        self.x=x
        self.y=y
    def __add__(self,second):
        return self.x+second.x,self.y+second.y
s1=Sample(10,20)
s2=Sample(20,30)
print(s1+s2)#(30,50)
```

use the ">" operator, to compare two objects of the class:
===============================================

code:
====

```
"""working with magic methods"""
class Sample:
    def __init__(self,x):
        self.x=x

    def __gt__(self,second):
        return self.x>second.x
    def __lt__(self,second):
        return self.x<second.x
s1=Sample(10)
s2=Sample(20)
print(s1>s2)#False
print(s1<s2)#True
```

user defined exceptions / custom exceptions:
======================================

when we want to create the our own exception, in python

we can also create our own exception using "classes"

user defined exception is exception and which is created or

defined by the programmer or developer using classes

when we want to create the user defined exception in python,

we will use the following steps:
==================================================

step-1:
=====

create a class with name and the name of the class refers to

exception name

the exception name or class name must starts with upper

case letter and ends with "Error"

syntax:
=======

class classnameError:

pass

step-2:
======

while creating the class , the class must take "Exception" has

super class

syntax:
======

syntax:
=======

class classnameError(Exception):

pass

step-3:
=====

in the class, we do not write the any code and make class as

empty

syntax:
======

syntax:
=======

```
class classnameError(Exception):

      pass
```

example:
=========

```
"""working with custom exception"""
class NumberLessThanZero(Exception):
    pass
class NumberMoreThan100(Exception):
    pass
class NumberNotDivisibleBy2(Exception):
    pass
try:
    num=int(input("number:"))
    if num<0:
        raise NumberLessThanZero("give value more than 0")
    elif num>100:
        raise NumberMoreThan100("give value less than 100")
    elif num%2!=0:
        raise NumberNotDivisibleBy2("give the even number")
except TypeError:
    print("give the valid error")
except NumberLessThanZero:
    print("give the value more than 0")
except NumberMoreThan100:
    print("give the value less than 100")
else:
    print(num)
```

note:
====

raise is a keyword and it is used to "raise the exception" by

the programmer or developer explicitly

in python, always exception name must end with "Error"

assert in Python:
=============

these are used to check the each line of the python code

is correct or not logiclly

in python, in order to work with assert, we will use a keyword

called "assert"

syntax:
======

assert condition, message

example:
========

```
"""working with asserts"""
x=int(input("Enter the value for x:"))
assert x>5, "x is greter than 5"
y=int(input("Enter the value for y:"))
assert y>5, "y is greter than 5"
sum=x+y
assert sum>10, "sum is greater 10"
print(sum)
```

note:
====

assert will give an error called "assertion error", when we

give  the condition for assert is True

assert will  not give an error called "assertion error", when we

give  the condition for assert is False

Duck Typing in Python:
==================

using this, we can implement the "polymorphism" without

inheritance

using duck typing, we can able to  "call the any class method

using object", by another method, based on the what class

object we given to the method instead of calling directly using

object of the class

duck typing eliminates "method overriding" in python

example-1:
=========

```
class Human():
    def move(self):
        print("it is calling from Human class")
class Bird():
  def move(self):
      print("it is calling from Bird class")
class Snake():
```

```
    def move(self):
        print("it is calling from Snake class")
class Frog():
   def move(self):
       print("it is calling from Frog class")
"""create the object for classes"""
h1=Human()
b1=Bird()
s1=Snake()
f1=Frog()
#duck method
def call_my_method(obj):#object as argument
    obj.move()
call_my_method(h1)
call_my_method(s1)
call_my_method(f1)
call_my_method(b1)
```

monkey patching at class level:
========================

example:
=======

```
"""create the class with a method"""
class Sample:
    def display(self):
        print("this is the method from class called display")
def update_display():
    print("display of sample method is updated")
"""create the object for class called Sample"""
s1=Sample()
s1.display()
"""apply the monkey patching"""
s1.display=update_display
s1.display()
```

Meta class in Python:
=================

object is a "instance of a class"

class defines "behaviour of the object"

meta class is a class and which defined behavior of another

using meta class, we can create another meta class

meta class is about another class

using python, we can able to create the meta classes:
==========================================

class meta_class_name(type):

```
    def __new__():
          pass
```

note:

every meta class will take "type" as super class

when we want to create the any class using meta class, we will

use following syntax in Python:
=================

```
class class_name(meta=meta_class_name):

    #logi c
```

example:
=======

```
"""create the meta class"""
class MetaClass1(type):
    def __new__(cls,name,bases,class_dict):
        print(f"the class name:{name}")
class MetaClass2(type):
    def __new__(cls,name,bases,class_dict):
        print(f"the class name:{name}")
class MetaClass3(type):
    def __new__(cls,name,bases,class_dict):
        print(f"the class name:{name}")

class Sample1(metaclass=MetaClass1):
    pass
class Sample2(metaclass=MetaClass1):
    pass
class Sample3(metaclass=MetaClass1):
    pass
```

example-2:
=========

```
"""create the meta class"""
class MetaClass1(type):
    def __new__(cls,name,bases,class_dict):
        print(f"the class name:{name}")
        print(f"the class dict:{class_dict}")
class MetaClass2(type):
    def __new__(cls,name,bases,class_dict):
        print(f"the class name:{name}")
        print(f"the bases class is:{bases}")
        print(f"the class dict:{class_dict}")
class MetaClass3(type):
```

```
    def __new__(cls,name,bases,class_dict):
        print(f"the class name:{name}")
        print(f"the class dict:{class_dict}")
class Sample1(metaclass=MetaClass1):
    x,y,z=100,200,300
    def display():
        print("this is Sample1 class")
class Sample2(metaclass=MetaClass2):
    a,b,c=10,20,30
    def display():
        print("this is Sample1 class")
class Sample3(metaclass=MetaClass3):
    p,q,r=1,2,3
    def display():
        print("this is Sample1 class")
```

example-3:
=========

```
"""create the meta class"""
class MetaClass1(type):
    def __new__(cls,name,bases,class_dict):
        print(f"the class name:{name}")
        print(f"the bases are:{bases}")
        print(f"the class dict:{class_dict}")
class MySample1:
    pass
class MySample2:
    pass
class Sample1(MySample1,MySample2,metaclass=MetaClass1):
    x,y,z=100,200,300
    def display():
        print("this is Sample1 class")
```

example-4:
=========

```
"""create the meta class"""
class MetaClass1(type):
    def __new__(cls,name,bases,class_dict):
        print(f"the class name:{name}")
        print(f"the bases are:{bases}")
        print(f"the class dict:{class_dict}")
class MetaClass2(type):
    def __new__(cls,name,bases,class_dict):
        print(f"the class name:{name}")
        print(f"the bases are:{bases}")
        print(f"the class dict:{class_dict}")
class MySample1:
    pass
class MySample2:
    pass
class Sample1(MySample1,MySample2,metaclass=MetaClass2):
    x,y,z=100,200,300
    def display():
```

print("this is Sample1 class")

note:
====

a class can take only one Meta Class

a Meta class always takes "type" as super class

when we are creating the Meta class, we will create with a

method called "__new__()"

the method "__new__()" will always takes the following data

or arguments:
================================================

1.cls <==== it represents , the method __new__() is class

method

2.name<=== it represents, the class name of the class which is

created using meta class

3.bases<=== it represents, the class base classes of the class

which is created using meta class

4.class_dict<=== it represent, the class members of the class

which is created using meta class

Multi-Threading In Python:
=====================

when we write a program and program we are given to the

processor, the program is going to be converted as "process"

process<=== "Program under execution"

in computer, in order to convert the program into process ,

and this job will taken care by " OS "

when we open multiple software’s in the system, the operating

system makes every software as "process"

when we opening multiple software’s, the operating system

will always will multiple processes and every will have a unique process id and it is helpful for OS, to identify the

process in computer

every activity of the process can be known by the OS using

"PCB" (process control block) / "process table"

in computer, every process is always independent with each

other when we compare with other process

when we are working with multiple processes at a time in the

computer with help of "Operating system", then we say it is

"multi-programming" . it means the CPU can able to process

the more than one process

multi-processing means " we can use one or more processors

to execute one more process" in computer. due to this we

can dedicate each process to each CPU or processors and

will will execute all process in same time

when multiple processes are executing and taking same

resources, there may be chance of "RACE condition"( due to this

other process will wait , until it get the resource what

process is required, if any process is in the race condition,

then we say the process in "starvation" zone)

when we are working with process in computer, in computer

OS again divide the "process into "n" number of sub-process"

due to dividing the " a process into sub process, it will get

more faster execution than normal way executing the process"

Multi-Threading with Python:
=====================

multi-threading is derives from a word "multi-tasking" multi-tasking is a process of "doing multiple tasks/jobs in a

particular interval of time"

who perform multi-tasking ,is termed as "multi-tasker"

in computers , the operating system can able to manage the

several jobs at a time using a concept called "process(process

management)"

operating system means "system software"  and which is going

to manage computer user and computer hardware

in computer ,process means " program under execution"

in a system/computer, all process are managed by the OS

with process id(it an unique identification number given os

for every new process in the system)

every process is independent with respect to another process ,

but two or more process can able to share same resources in

the machine(i/o devices, memory, .............)

in python, we have concept called "multi threading",

in os- level , we can also performs "multi-threading", it means

os supports "multi-threading"

multi-threading means "executing multiple threads concurrently

or in a parrell mode"

thread is a "flow of execution and it is an independent unit

or program, and it performs a specific job in program"

in python, thread is "object"

in python, we  can divide entire program into multiple threads

run all threads at time along with program"

in python, when we want to work with multi-threading, we will

use a module called "threading"

to create the thread in the python program we will have the

following ways:
=============================================

create the thread using "function":
===========================

step-1: create the thread as a function and in a program

we can create the any number of threads

step-2: create the object for thread using threading module

thread_object_name=threading.Thread(target=thread_name)'

step-3: using thread object name ,we will initiate a method

called "start()", when we call the start() method with

thread object name, thread will automatically will into

"running" state / thread starts it execution, when call

start() method using "thread object", it internally called

run() method and this method available in the Thread

class

example:
=======

```
"""working with multi-threading"""
import threading,time
"""create a thread 1"""
def mythread1():
    for i in range(1,5):
        print(f"my thread-1:{i}")
        time.sleep(0.1)
"""create a thread 1"""
def mythread2():
    for i in range(1,10):
        print(f"my thread-2:{i}")
        time.sleep(0.2)
"""create a thread 1"""
def mythread3():
    for i in range(1,15):
        time.sleep(0.3)
        print(f"my thread-3:{i}")
"""create the object for threads using
threading class"""
t1=threading.Thread(target=mythread1)
t2=threading.Thread(target=mythread2)
```

```
t3=threading.Thread(target=mythread3)
"""execute the threads"""
t1.start()
t2.start()
t3.start()
```

create the thread  using class with extending Thread class
===============================================

step-1: create the class and take "Thread" as super class

where "Thread" is a class which is available in

threading module

step-2:  in the create the thread, using "run()" method

step-3: create the object for class and call the start method

using "thread object"

example:
=======

```
"""working with multi-threading"""
import threading,time
class Mythreadclass(threading.Thread):
    """create a thread 1 without using self"""
    def mythread1():
        for i in range(1,5):
            print(f"my thread-1:{i}")
            time.sleep(0.1)
    """create a thread 2 without using self"""
    def mythread2():
        for i in range(1,10):
            print(f"my thread-2:{i}")
            time.sleep(0.2)
    """create a thread 3 without using self"""
    def mythread3():
        for i in range(1,15):
            time.sleep(0.3)
            print(f"my thread-3:{i}")
"""create the object for threads using
threading class"""
"""execute the threads"""
t1=Mythreadclass(target=Mythreadclass.mythread1)
t2=Mythreadclass(target=Mythreadclass.mythread2)
t3=Mythreadclass(target=Mythreadclass.mythread3)
t1.start()
t2.start()
t3.start()
```

create the thread using "class" without extending Thread class:
==================================================

step-1:  create the class

step-2: in the class created a method as thread with any name

step-3: create the object for the class

step-4: create the thread object using thread class, while

creation of thread object,will give the target as

"object_name.method" of the class what we created

step-5:  start the thread (to run the thread)

example:
=======

```
"""working with multi-threading"""
import threading,time
class Mythreadclass():
    """create a thread 1 without using self"""
    def mythread1(self):
        for i in range(1,5):
            print(f"my thread-1:{i}")
            time.sleep(0.1)
    """create a thread 2 without using self"""
    def mythread2(self):
        for i in range(1,10):
            print(f"my thread-2:{i}")
            time.sleep(0.2)
    """create a thread 3 without using self"""
    def mythread3(self):
        for i in range(1,15):
            time.sleep(0.3)
            print(f"my thread-3:{i}")
"""create the objetc for class called Mythreadclass"""
m1=Mythreadclass()
"""create the object for threads using
threading class"""
t1=threading.Thread(target=m1.mythread1)
t2=threading.Thread(target=m1.mythread2)
t3=threading.Thread(target=m1.mythread3)
"""execute the threads"""
t1.start()
t2.start()
t3.start()
```

how to see the "Threads are executed very faster than normal

way of executing the functions" in Python:

==================================================

code:
====

```
"""working with multi-threading"""
import threading,time
start_time=time.perf_counter()
def mythread_1():
    for i in range(1,5):
        time.sleep(1.4)
def mythread_2():
    for i in range(1,5):
        time.sleep(1.4)
def mythread_3():
    for i in range(1,5):
        time.sleep(1.4)
t1=threading.Thread(target=mythread_1)
t2=threading.Thread(target=mythread_2)
t3=threading.Thread(target=mythread_3)
end_time=time.perf_counter()
print(f"total_time_taken={end_time-start_time}")
```

how to get the thread name in python:
=================================================

to get the thread name in python, we will use the following

syntax:
=================================================

thread_object_name.name

or

threading.current_thread().name

to set the thread name in python, we will use following syntax:
==================================================

Thread(target=" ", name=" ")

or

threading.current_thread().name="thread name"

how to see the number threads, which are active in the given

python code:
==================================================

to see the number threads, which are active in the given

python code, python uses "threading" module and this module

will have a function called "enumerate()"

example:
=======

```
"""working with multi-threading"""
import threading,time
def mythread_1():
   threading.current_thread().name="thread-1"
   for i in range(1,5):
       print(i)
       time.sleep(0.2)
def mythread_2():
   threading.current_thread().name="thread-2"
   for i in range(1,5):
        print(i)
        time.sleep(0.3)

t1=threading.Thread(target=mythread_1)
t2=threading.Thread(target=mythread_2)
t1.start()
t2.start()
for thread_details in threading.enumerate():
    print(f"thread_details:{thread_details.name}")
```

example-2:
=========

```
"""working with multi-threading"""
import threading,time
def mythread_1():
   threading.current_thread().name="thread-1"
   for i in range(1,5):
       pass
def mythread_2():
   threading.current_thread().name="thread-2"
   for i in range(1,5):
       pass
t1=threading.Thread(target=mythread_1)
t2=threading.Thread(target=mythread_2)
t1.start()
t2.start()
count=0
for thread in threading.enumerate():
    if thread.is_alive():
      print(f"thread_details:{thread.name}")
      count+=1
print("all active threads in the program:",count)
```

in python, threading.enumerate() function will return all

active threads, demon threads of the given program in python, to know the active threads in the given program,

we will following syntax:
=====================================================

threading.active_count()

example:
=======

```
"""working with multi-threading"""
import threading,time
def mythread_1():
   threading.current_thread().name="thread-1"
   time.sleep(1)
def mythread_2():
   threading.current_thread().name="thread-2"
   time.sleep(2)
t1=threading.Thread(target=mythread_1)
t2=threading.Thread(target=mythread_2)
t1.start()
t2.start()
print("number of active threads:",threading.active_count())
```

how to get the id of the threads in Python:
====================================

to get the id of the threads in Python, in python we will

two functions:
===========

1.threading.get_ident()<=== this function will give the thread

id which is given and used by the

python

2.threading.get_native_id()<== this function will give the  true

or native id of the thread and

which is used by OS

example:
=======

```
"""working with multi-threading"""
import threading,time
def mythread_1():
   print("mythread_1 id by Python:",threading.get_ident())
   print("mythread_1 id by OS:",threading.get_native_id())
   time.sleep(1)
def mythread_2():
```

```
   print("mythread_2 id by Python:",threading.get_ident())
   print("mythread_2 id by OS:",threading.get_native_id())
   time.sleep(3)
t1=threading.Thread(target=mythread_1)
t2=threading.Thread(target=mythread_2)
t1.start()
t2.start()
```

working with join() method and demon thread in python:
============================================

example-1:
=========

```
"""working with multi-threading"""
import threading,time
"""thread-1"""
def mythread_1():
  for i in range(1,5):
     time.sleep(1)
     print("thread-1",i)
"""thread-2"""
def mythread_2():
   for i in range(1,5):
      time.sleep(2)
      print("thread-2",i)
"""main thread"""
for i in range(1,5):
    time.sleep(0.3)
    print("main thread",i)
t1=threading.Thread(target=mythread_1)
t2=threading.Thread(target=mythread_2)
t1.start()
t2.start()
t1.join()
t2.join()
```

example-2:
=========

```
"""working with join method"""
import threading,time
"""create the thread-1"""
def my_thread1():
    for i in range(1,11,2):
        print("odd:",i)
        time.sleep(0.5)

def my_thread2():
    for i in range(0,11,2):
        print("even:",i)

"""create the thread objects using
```

```
Thread class"""
t1=threading.Thread(target=my_thread1)
t2=threading.Thread(target=my_thread2)
t1.start()
t2.start()
t1.join()
t2.join()
```

daemon thread:
============

daemon thread is a "background thread and which is runs in

background and it execution automatically stop the once the

main thread execution is stops"

daemon thread execution is always depends on the

main thread

to create the any thread as daemon thread, we will use the

following syntax:

```
thread_object_name=threading.Thread(target=thread_name,
                                    daemon=True)
```

example:
=======

```
"""working with join method"""
import threading,time
"""create the thread-1"""
def my_daemon_thread():
    while True:#infinite loop
        print("This is daemon Thread-1")
        time.sleep(1)

"""create the thread objects using
Thread class
"""
t1=threading.Thread(target=my_daemon_thread,daemon=True)
t1.start()
for i in range(1,6):
    print("this is main thread-1")
    time.sleep(0.1)
```

example-2:
=========

```
"""working with join method"""
import threading,time
```

```
"""create the thread-1"""
def my_daemon_thread():
    while True:#infinite loop
        print("This is daemon Thread-1")
        time.sleep(2)

"""create the thread objects using
Thread class
"""
t1=threading.Thread(target=my_daemon_thread,daemon=True)
"""how to check the thread is daemon or not"""
if t1.isDaemon():
    print("the thread is daemon")
else:
    print("the thread is not daemon")
t1.start()
print("this is main thread-1")

example-3:
=========

"""working with join method"""
import threading,time
"""create the thread-1"""
def my_daemon_thread():
    while True:#infinite loop
        print("This is daemon Thread-1")
        time.sleep(2)

"""create the thread objects using
Thread class
"""
t1=threading.Thread(target=my_daemon_thread,daemon=True)
"""how to check the thread is daemon or not"""
if t1.daemon:
    print("the thread is daemon")
else:
    print("the thread is not daemon")
t1.start()
print("this is main thread-1")

example-4:
=========

"""working with join method"""
import threading,time
"""create the thread-1"""
def my_daemon_thread():
    while True:#infinite loop
        print("This is daemon Thread-1")
        time.sleep(2)

"""create the thread objects using
Thread class
"""
```

```
t1=threading.Thread(target=my_daemon_thread)
"""how to check the thread is daemon or not"""
if t1.daemon:
    print("the thread is daemon")
else:
    print("the thread is not daemon")

print("this is main thread-1")
```

example-5:
========

```
"""working with join method"""
import threading,time
"""create the thread-1"""
def my_daemon_thread():
    while True:#infinite loop
        print("This is daemon Thread-1")
        time.sleep(2)

"""create the thread objects using
Thread class
"""
t1=threading.Thread(target=my_daemon_thread)
"""how to check the thread is daemon or not"""
t1.daemon=True
if t1.daemon:
    print("the thread is daemon")
else:
    print("the thread is not daemon")

print("this is main thread-1")
```

example-6:
========

```
"""working with join method"""
import threading,time
"""create the thread-1"""
def my_daemon_thread():
    while True:#infinite loop
        print("This is daemon Thread-1")
        time.sleep(2)

"""create the thread objects using
Thread class
"""
t1=threading.Thread(target=my_daemon_thread)
"""how to check the thread is daemon or not"""
t1.daemon=True
if t1.daemon:
    print("the thread is daemon")
else:
```

```
    print("the thread is not daemon")
t1.start()
t1.join()
print("this is main thread-1")
```

note:
====

when we main thread execution ends or exists, daemon thread

will stops its execution automatically

to make the main thread has to wait until daemon thread

completes its execution using "join()" method

GIL in python:
===========

GIL stands for "global Interpreter Lock"

it is a program which uses by the "python" and make only

one thread has execute at a time even we have multi-core

processors and it is basically  a locking mechanism

python provides, this is in -built ,this will automatically called

when we working with "multi- threading"

async and await  keywords in Python:
=============================

async is a keyword and which make the function can execute

like asynchronously

await is a keyword and using this  "we can pause execution"

of the function

these both keywords we will apply on "functions"

async is "make a function can execute parallel" with other

tasks in the program

await is used to "pause the execution of the function"

example:
========

working with browser, downloading the file from internet while working with notepad, we can also listen songs from

the music player

in order to work with "async and await", we will use a module

called  "asyncio" in Python

when we give the "keyword" called "async" to the function,

function can work like a "asynchronous" function

when  a function is "asynchronous" , this function can run with

other functions

in order to pause the "Asynchronous" function execution, we

will use a keyword called "await"

syntax to use asyncio module:
========================

import asyncio

how to use "async" to a function:
=========================

syntax:
======

async def function_name(arg1,arg2,arg3,........argn)

        #logic

how to use "await" to a function:
==========================

await function_name()

or

await asyncio.sleep(time_slice)

example:
=======

```
"""working with async and await"""
import asyncio
"""create the a method with async"""
async def method1():
```

```
    print("method 1 execution is started")
    await asyncio.sleep(3)
    print("method 1 execution is ended")
async def method2():
    print("method 2 execution is started")
    await asyncio.sleep(3)
    print("method 2 execution is ended")
async def main():
  await asyncio.gather(method1(),method2())
asyncio.run(main())
```

example-2:
=========

```
"""working with async and await"""
import asyncio
"""create the a method with async"""
async def method1():
    print("method 1 execution is started")
    await asyncio.sleep(3)
    print("method 1 execution is ended")
async def method2():
    print("method 2 execution is started")
    await asyncio.sleep(3)
    print("method 2 execution is ended")
async def main():
  await method1()
  await method2()
asyncio.run(main())
```

example-1:
=========

```
"""working with multi-threading"""
import threading,time
start_time=time.perf_counter()
def mythread_1():
   for i in range(1,5):
        time.sleep(1.4)
def mythread_2():
   for i in range(1,5):
        time.sleep(1.4)
def mythread_3():
    for i in range(1,5):
        time.sleep(1.4)
t1=threading.Thread(target=mythread_1)
"""get the thread name in python"""
print(t1.name)
t2=threading.Thread(target=mythread_2)
"""get the thread name in python"""
print(t2.name)
t3=threading.Thread(target=mythread_3)
"""get the thread name in python"""
print(t3.name)
```

```
end_time=time.perf_counter()
print(f"total_time_taken={end_time-start_time}")
```

example-2:
=========

```
"""working with multi-threading"""
import threading
def mythread_1():
   print(threading.current_thread().name)
   for i in range(1,5):
       pass
def mythread_2():
   print(threading.current_thread().name)
   for i in range(1,5):
        pass
def mythread_3():
    print(threading.current_thread().name)
    for i in range(1,5):
         pass
t1=threading.Thread(target=mythread_1)
t2=threading.Thread(target=mythread_2)
t3=threading.Thread(target=mythread_3)
t1.start()
t2.start()
t3.start()
```

example-3:
=========

```
"""working with multi-threading"""
import threading
def mythread_1():
   print(threading.current_thread().name)
   for i in range(1,5):
       pass
   threading.current_thread().name="thread-1"
   print("after the change:",threading.current_thread().name)
def mythread_2():
   print(threading.current_thread().name)
   for i in range(1,5):
        pass
   threading.current_thread().name="thread-2"
   print("after the change:",threading.current_thread().name)
def mythread_3():
    print(threading.current_thread().name)
    for i in range(1,5):
         pass
    threading.current_thread().name="thread-3"
    print("after the change:",threading.current_thread().name)
t1=threading.Thread(target=mythread_1)
t2=threading.Thread(target=mythread_2)
t3=threading.Thread(target=mythread_3)
t1.start()
t2.start()
```

```
t3.start()
```

example-1:
=========

```
#get the name of the current running thread
import threading
print("thread name:",threading.current_thread().name)
```

example-2:
=========

```
import threading
#tread creation
#create the thread as function
def display():#here display as target for thread
    for i in range(1,10):
        print("i am thread-1")
#create the object for thread using threading module
t1=threading.Thread(target=display)
t1.start()
for i in range(1,10):
    print("Main Thread")
```

example-3:
=========

```
import threading
#create the class
class myclass(threading.Thread):
    #create the thread with name run
    def run(self):
        for i in range(1,10):
            print("this thread from my class",threading.current_thread().name)
#create the thread object
t1=myclass()
t1.start()
#create the code main thread
for i in range(1,10):
    print("Main Thread",threading.current_thread().name)
```

example-4:
=========

```
import threading
#create the class
class myclass:
    def display(self):
        for i in range(1,10):
            print("this is thread from the my class")
#create th object for the my class
my1=myclass()
#create thread object using threading Thread class
t1=threading.Thread(target=my1.display)
t1.start()
```

```
for i in range(1,10):
    print("this is main thread")
```

example-5:
=========

```
from threading import *
#get the thread name
print("get the current thread_name",current_thread().getName())
print("get the current thread_name",current_thread().name)
#set the thread name
current_thread().setName("my_main_thread")
print("get the current thread_name",current_thread().getName())
print("get the current thread_name",current_thread().name)
current_thread().name="new_my_thread"
print("get the current thread_name",current_thread().getName())
print("get the current thread_name",current_thread().name)
```

example-6:(Recommended to use)
==========================

```
from threading import *
#get the thread name
print("get the current thread_name",current_thread().name)
#set the thread name
current_thread().name="new_my_thread"
print("get the current thread_name",current_thread().name)
```

example-7:
=========

```
from threading import *
import time as t
def thread1():
    for i in range(1,5):
        print("child thread-1")
        print("thread_name",current_thread().name)
        t.sleep(1)

def thread2():
   for i in range(1,5):
       print("child thread-2")
       print("thread_name",current_thread().name)
       t.sleep(1)

#create the object for thread-1
t1=Thread(target=thread1)
#create the object for thread-2
t2=Thread(target=thread2)
t1.start()
t2.start()
t1.join()
t2.join()
#main thread
for i in range(1,10):
    print("Main thread")
```

```
    print("thread_name",current_thread().name)
    t.sleep(1)
```

example-7:
========

```
from threading import *
import time as t
def thread1():
    for i in range(1,5):
        print("child thread-1")
        print("thread_name",current_thread().name)


def thread2():
   for i in range(1,5):
       print("child thread-2")
       print("thread_name",current_thread().name)

print("the number of active threads",active_count())
#create the object for thread-1
t1=Thread(target=thread1)
#create the object for thread-2
t2=Thread(target=thread2)
t1.start()
t2.start()
print("the number of active threads",active_count())
print("the main thread id",current_thread().ident)
print("the t1 thread  id",t1.ident)
print("the t2 thread id",t2.ident)
```

example-8:
=========

```
from threading import *
import time as t
def thread1():
    for i in range(1,5):
        print("child thread-1")
        print("thread_name",current_thread().name)
def thread2():
   for i in range(1,5):
       print("child thread-2")
       print("thread_name",current_thread().name)
#create the object for thread-1
t1=Thread(target=thread1)
#create the object for thread-2
t2=Thread(target=thread2)
t1.start()
t2.start()
print("t1 is running or not",t1.is_alive())
print("t2 is running or not",t2.is_alive())
print("main thread",current_thread().is_alive())
```

example-9:

=========

```
import threading as th
import time as t
"""Create the thread-1 as a function"""
def even_numbers(start,end):
    for i in range(start,end+1):
        if i%2==0:print(f"even:{i}")
        t.sleep(3)
"""Create the thread-2 as a function"""
def pallendrome(start,end):
    for i in range(start,end+1):
        if str(i)==str(i)[::-1]:print(f"pallendrome:{i}")
        t.sleep(2)
"""Create the thread-3 as a function"""
def prime_digit(start,end):
    for i in range(start,end+1):
        if str(i)[-1] in "2357":print(f"prime:{i}")
        t.sleep(1)
t1=th.Thread(target=even_numbers,args=(1,10))
t2=th.Thread(target=pallendrome,args=(100,110))
t3=th.Thread(target=prime_digit,args=(100,110))
t1.start()
t2.start()
t3.start()
for i in range(1,10):
    print(f"the main thread:{i}")
```

example-10:
==========

```
import threading as th
import time as t
"""Create the thread-1 as a class"""
class Even(th.Thread):
    def run(self,start,end):
      for i in range(start,end+1):
        if i%2==0:print(f"even:{i}")
        t.sleep(3)
"""Create the thread-2 as a class"""
class Pallendrome(th.Thread):
   def run(self,start,end):
     for i in range(start,end+1):
        if str(i)==str(i)[::-1]:print(f"pallendrome:{i}")
        t.sleep(2)
"""Create the thread-3 as a class"""
class Prime_digit(th.Thread):
   def run(self,start,end):
     for i in range(start,end+1):
        if str(i)[-1] in "2357":print(f"prime:{i}")
        t.sleep(1)
t1=Even()
t2=Pallendrome()
t3=Prime_digit()
t1.start()
```

```
t2.start()
t3.start()
for i in range(1,10):
    print(f"the main thread:{i}")
```

example-11:
==========

```
import threading as th
import time
"""create the thread with name mythread1"""
def display():
    for i in range(1,10):
        print("this is thread-1")
        time.sleep(1)
        print(f"thread name:{th.current_thread()}")
#create the thread object using Thread class
t1=th.Thread(target=display)
t1.start()
#create the main thread code
for i in range(1,11):
    print(i)
    time.sleep(0.2)
    print(f"thread name:{th.current_thread()}")
```

example-12:
==========

```
import threading as th
import time
"""create the thread with name mythread1"""
def display():
    for i in range(1,10):
        print("this is thread-1")
        time.sleep(4)
        print(f"thread name:{th.current_thread().getName()}")
#create the thread object using Thread class
t1=th.Thread(target=display)
t1.start()
#create the main thread code
for i in range(1,11):
    print(i)
    time.sleep(0.2)
    print(f"thread name:{th.current_thread().getName()}")
```

example-13:
==========

```
import threading as th
import time
"""create the thread with name mythread1"""
def display():
    for i in range(1,10):
        print("this is thread-1")
        time.sleep(4)
        th.current_thread().setName("mythread")
```

```
        print(f"thread name:{th.current_thread().getName()}")
#create the thread object using Thread class
t1=th.Thread(target=display)
t1.start()
#create the main thread code
for i in range(1,11):
    print(i)
    time.sleep(0.2)
    th.current_thread().setName("my main thread")
    print(f"thread name:{th.current_thread().getName()}")
```

example-14:
==========

```
import threading as th
import time
"""create the thread with name mythread1"""
def display():
    for i in range(1,10):
        print("this is thread-1")
        time.sleep(4)
        #to give the thread name
        th.current_thread().name="my thread-1"
        print(f"thread name:{th.current_thread().name}")
#create the thread object using Thread class
t1=th.Thread(target=display)
t1.start()
#create the main thread code
for i in range(1,11):
    print(i)
    time.sleep(0.2)
    #to give the thread name
    th.current_thread().name="my main thread"
    print(f"thread name:{th.current_thread().name}")
```

example-15:
==========

```
import threading as th
import time as t
def thread1():
    for i in range(1,5):
        t.sleep(5)
        print("child thread-1")

        print("thread_name",th.current_thread().name)

def thread2():
   for i in range(1,5):
       print("child thread-2")
       t.sleep(5)
       print("thread_name",th.current_thread().name)

print("the number of active threads",th.active_count())
```

```
t1=th.Thread(target=thread1)
t2=th.Thread(target=thread2)
t1.start()
t2.start()
print("the number of active threads",th.active_count())
```

example-16:
==========

```
import threading as th
import time as t
def thread1():
    for i in range(1,5):
        t.sleep(5)
        print("thread_name",th.current_thread().name)

def thread2():
   for i in range(1,5):
       t.sleep(5)
       print("thread_name",th.current_thread().name)

t1=th.Thread(target=thread1)
t2=th.Thread(target=thread2)
t1.start()
t2.start()
t1.join()
t2.join()
for i in range(1,10):
    t.sleep(1)
    print(f"Thread_name:{th.current_thread().name}")
```

example-18:
==========

```
"""without multi-threading"""
import threading as th
import time as t
start_time=t.perf_counter()
def display():
    for i in range(1,6):
        t.sleep(1)
def display2():
    for i in range(1,6):
        t.sleep(1)
display()
display2()
for i in range(1,10):
    t.sleep(1)
end_time=t.perf_counter()
print("total-time",end_time-start_time)
```

example-19:
==========

```
"""with multi-threading"""
import threading as th
import time as t
start_time=t.perf_counter()
def display():
    for i in range(1,6):
        t.sleep(1)
def display2():
    for i in range(1,6):
        t.sleep(1)
t1=th.Thread(target=display)
t2=th.Thread(target=display2)
t1.start()
t2.start()
for i in range(1,10):
    t.sleep(1)
end_time=t.perf_counter()
print("total-time",end_time-start_time)
```

Data Structures and algorithms Using Python:
======================================

what is data:
==========

data means "what we can able to store in computer memory"

example:
=======

image, audio file, video file, text documents, ........

in general, when we are working with python programming,

we used to store the data like number(integer or real number),

character data(strings), Boolean values, .................

to store the data like numbers / text data / Boolean data,

we are going to use variables in python.

the variable can able to "hold only one value" at a time

for example,

```
a=10   <== a=10

a=1.234 <=== a=1.234
```

the limitation of the variable in python is "it can have only one

value"

when we want to store the multiple values, then we need to

use multiple variables

for example,

```
a=10

b=20

c=30

d=40

.
.
.

x=100
```

Data Structure:
============

data structure provides a way to organize the data,

using data structures, we can able do the following :

1. easy to perform the operations on the data

   (in general, we can able to perform operations like

   insertion,  deletion, updating, searching ....

2. maintaining the data at memory also so easy and very

   efficient

3. retrieval of the data is very easy

4.solving the real-time problems with data structures is also

   very easy

   examples:
   ========

   1. text editor like notepad uses the "Stack" data structure

      to peform the "undo(ctrl+Z)"

   2. in computers, the task scheduler do the all tasks in the

      computers with help of "Queue" data structure

   3. web browser also uses the "Stack" data structure, to

      open the what webpage, when the user click the "back"

      button

   4.Banking system also uses "Queue" data structure, to

      process  the all bank transactions

   5.social media platform like "Facebook" uses a data

      structure called "Graph" , here the all users of the platform

      can be represented as "nodes" and their friendship can be

      represented as "edges"

   6. apps like "Ola, Uber, Rapido...." uses a data structure

called "Graph", when user book a ride, to set up the ride ,with help of graph data structure, app can calculate the distance ,fare and also best route using BFS algorithm

7. application like amazon, flipkart,.... uses the data structures Dictionary and trees for the operations like:

   amazon uses hashmap (implementation of Dictionary), with help of hashmap, we can easily map the product ids to product details, due to that we can able to perform the searching very efficiently

   in amazon, we also uses the Trees to implementing the auto-complete suggestions

8. in the case of the booking request for any tickets, in the applications like irctc, BMS, make my trip,......., we are using "Queue" as a data structure to handling the booking requests

in general data structures are two types:
================================

1.linear data structure:
==================

in linear data structures, all elements(data) are arranged sequentially and each element is connected with to the previous element and next element

these data structure are can be implemented so easy

these data structures can be maintained at memory also very easy

types of the linear data structures:
===========================

we have the following:
==================

1.Array:
=======

array is "collection of elements of similar type or same type"

and all elements are stored at contiguous memory locations

2.Stack:
=======

Stack is "Last-In-First-Out"(LIFO) data structure

the element which is inserted first, will be deleted at last

or

the element which is inserted last, will be deleted first

example:
=======

a stack of plates, where the last plate added is the first

removed

3.Queue:
=======

Queue is  a "First-In-First-Out"(FIFO) data structure

the element which is inserted first, will be deleted at first

in Queue

example:
=======

a line of customers at booking counter for tickets, where the

first person in the line, will be the tickets first

4.Linked List:
===========

linked list is "collections of elements and in this elements can

be termed as "nodes", each node will have data and also have

a reference to the next node

in general, we will have two types of linked lists:
=====================================

1.single linked list:
===============

in this linked list,

each node have data and only reference of the next node

2.double linked list:
===============

in this  double linked list,

each node have the data along with reference of the next node

and also reference of the previous node

3.circular linked list:
================

in this linked list,

the last node in the linked list will connected with starting node

of the linked list

the circular linked list may be "single linked list or double

linked list"

5.Deque(Double Ended Queue)
=========================

Deque is a queue data structure only

in this Deque, we can perform the insertion and deletion at

both ends of the queue

2.non linear data structure
======================

in non-linear data structure, elements can be arranged in a hierarchical  manner , where each element can represent one or

more elements

in this data structures, we will have the following:
==========================================

1.Trees:
=======

it is a "hierarchical structure consisting of nodes", where each

node has a "parent-child" relation ship

the top most node in a tree called as "root" and the nodes

below to root node are called as "children" nodes

the following are different types of the trees:
======================================

1.binary tree

2.Binary Search Tree(BST)

3.AVL Tree

4.Heap Tree

2.Graph:
=======

a collection of nodes(vertices) and edges connecting them

these are generally used to represent the "complex relationship

between the objects"

the following are various types of graphs:
==================================

1.Directed Graph

2.Un-directed Graph

3.Weighted Graph

3.Trie(Prefix Tree)
==============

Trie is "a specialized version of the Tree"

using Tries " we can able perform an efficient way searching the strings and especially for pre-fix based operations

in this every node represents a "character"

3.Hash-based Data Structure:
======================

hashing involves "a data or a value is mapped with some

specific value", with help of that we can able to find or search

the element very fast

in this, we are going to arrange the data in the form of

"Key and value pair" mechanism

there are two types of Hash-Based Data structures:
========================================

1.Hash Table

2.Hash Map(Python Dictionary)

algorithm:
========

algorithm "it is a step by step procedure to solve a problem"

using computer

how we can write the algorithm in real-time:
==================================

we can write the algorithm in real-time, using "any natural

language" (English, hindhi, german, French, ......)

algorithm is informal solution and which understand only

by the programmers or developers

when we have a problem to be solved,

generally problem means "which requires solution"

every problem may have "n" number of solutions

every solution is converted into "an algorithm"

once we write the algorithm, we need to check the performance of the algorithm, by using  "analysis of algorithm" we can

evaluate the performance of the algorithm in 2 ways:
==========================================

1. Time complexity:
===============

time complexity measures performance

"based on how much time taken for execution"

when we are evaluating the Time complexity of an algorithm

, we will use following  Asymptotic notations:
=================================================

1.Big Oh(O)===>it give the performance in "Worst Case"

2.Omega ===>it will give the performance in "Best Case"

3.Theta====>it will give the performance in "Average Case"

example-1:
=========

write a an algorithm to "Swap the given two numbers"
===========================================

```
algorithm swap_two_numbers(a,b)

       read a,b  <----    2

       temp<--a <---     1

       a<--b      <---     1

       b<-temp   <---   1

      write a,b   <---    1
  ======================
      total  ======>   O(6)<== it we have any number, it

                                             can be considered as "constant"
```

O(any number) is always equal to O(1)

O(1) is always "constant"

write a algorithm to check given number is even or odd:
=============================================

algorithm even_or_odd(number)

read number <----- 1

if number is divisible by 2

write "even" <---1

else write "odd"
=======================================

total ===============> O(2) ===>O(1)

3. write a algorithm to print the numbers from 1 to 10:
=============================================

algorithm print_1_to_10

i<---1

repeat

write i

i<--i+1

until i become 11

time complexity of the above is "O(1)"

write a an algorithm for print the numbers from 1 to n:
=============================================

for i in range(1,n):

print(i)

time complexity is O(n)

the single loop time complexity is always "O(n)"

write an algorithm print the even number for given start

and end:
==================================================

algorithm print_even_numbers(start,end)

read start <---- 1

read end <-----1

repeat

```
        if start is dividible by 2 <---- 1

                write start  <---- 1
       start<--start+1  <---- 1

      until start become end
```

if we get any time complexity like O(n+any number), then

it can assumed as O(n)

in general, we will always consider highest polynomial degree

suppose,

O(n+10) ====>O(n)

O(n power 2+n +1)===>O(n power 2)

O(n power 3+ n power 2+n power 1 +10) ===>O(n power 3)

write a python program to calculate the time complexity

to check the given number is prime or not:
==================================================

code:
====

```
n=int(input("enter the number:")) #O(1)
fact=1#O(1)
count=0#O(1)
while fact<=n:   #O(n+1)
    if n%fact==0:
        count+=1
    fact+=1
if count==2:
    print("prime") #==>O(2)
else:
    print("not a prime")#===>O(1)
#total:O(n+7) ==>O(n)
```

write a python program print the even numbers for given

range:
================================================

code:
====

```
start=int(input("enter the start value:"))
end=int(input("enter the end value:"))
if start>end:start,end=end,start
```

```
if start%2!=0:start+=1
for i in range(start,end+1):
        print(i)
#time complexity: O(len(rang)/2)===>O(n/2)===>O(n)
```

write a python program to calculate the time complexity

on "check the given number is Armstrong number or not"
===================================================

code:
====

```
number=int(input("enter the number:"))
length=0
sum=0
#length of the given number
for i in str(number):length+=1 #O(n)
#find sum of the digits of the given number
for i in str(number):#O(n)
    sum+=int(i)**length
if sum==number:print("armstrong")# O(2)
else:print("not a armstrong")#O(1)
#time complexity:O(2n+6)===>O(n)
```

code-2:
======

```
number=int(input("enter the number:"))
if number>0 and number<1000000000:
    if number in [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407,
                  1634, 8208, 9474, 54748, 92727, 93084, 548834,
                  1741725, 4210818, 9800817, 9926315, 24678050,
                  24678051, 88593477, 146511208, 472335975,
                  534494836, 912985153,4679307774]:
        print("armstrong")
    else:
        print("not a armstrong")
#time complexity: O(1)
```

calculate the time complexity of the below code:
=======================================

```
start=int(input("enter the start value:"))
end=int(input("enter the end value:"))
for i in range(start,end+1):
    for j in range(1,start):
         print(j)
```

timecomplexity: O(n*n)

calculate the time complexity for the given code:
========================================

```
start=int(input("enter the start value:"))#o(1)
```

```
end=int(input("enter the end value:"))#o(1)
for i in range(start,end+1):#o(n)
    for j in range(start**2):#O(logn)
        print(j)
```

time Complexity: O(nlogn)

calculate the time complexity for the following:
=====================================

```
def print_number(num):
    if num>=0:
        print(num)
        num//=2
        print_number(num)
print_number(int(input("enter the number:")))
#time Complexity: O(logn)
```

calculate the time complexity of the below code:
======================================

```
times=0
def fibnocii(num):
    global times
    if num<=0:
        times+=1
        return 1
    else:
        times+=1
        return fibnocii(num-1)*fibnocii(num-2)
fibnocii(int(input("enter the number:")))
print(times)
#time complexity: O(2 power n)
```

calculate the time complexity for the below code:
======================================

```
rows=int(input("enter the rows:"))#5
for row in range(1,rows+1):#5
    for data in range(1,(rows-row)+2):#
        print(data,end="")
    print()
```

time complexity: O(n power 2)

calculate the time complexity:
=======================

```
#take a number from user
a=int(input("enter the number:"))
#convert the given decimal number into hexadecimal
```

```
pow=0
res=’’
while a!=0:
    if a%16==10:res+=’a’
    elif a%16==11:res+=’b’
    elif a%16==12:res+=’c’
    elif a%16==13:res+=’d’
    elif a%16==14:res+=’e’
    elif a%16==15:res+=’f’
    else:
        res+=str(a%16)
    a=a//16
print(res[::-1])
```

2.Space Complexity:
================

space complexity measures performance

"based on how much space taken for store"

Python Data Structures:
==================

in python, we will have the following data structures:
==========================================

1.list:
=====

in python,

list is used to store "collection of or group of elements

under single name"

list can have any type of data

list can have any number of elements

list can have duplicate elements

list can be empty

list can also called as "sequence type" (on this we can apply

both indexing and slicing")

list can be also called as "ordered collection"(the way we

give the elements, in the same way it will store the

elements, i.e insertion order preserved) list is also called as "mutable type"( on list we can perform

operations like insert/ update/ delete)

list can be created using "[]"

list can be also created using "list()" constructor

in python , we can create the list using a concept called

"list comprehension"

using list comprehension, we can create the list using

another list or iterable(using range(), tuple, set,...........

syntax:

list_name=[variable_name for variable_name in itereable if

cond.]

note:
====

in python, list can termed as "list object" and it’s class name

is "list"

to know the list object type, we will use a function called as

"type()"

on list, we will perform the following operations:
======================================

1.insertion

2.update

3.deletion

4.search

5.sorting

6.traverse

list functions in python:
===================

1.len()<=== using this we can able to find the "number of

elements in the list"

2.append()<=== using this we can able to insert element at end

of the list

3.insert()<=== using this we can able to insert the element at

any function

4.pop()<=== using this we can delete the element from list

5.sort()<=== using this we can able to sort the elements in

the list either in ascending or descending order

6.count()<== using this we can able to count the given element

how many repeated or frequency of the given

element in the list

7.index()<== using this we can able to find the index of the

given element in the list (based on it's first

occurrence in the list)

8.copy()<=== using this we can copy all elements of list into

another list

9.extend()<=== using this we can add the elements of the

another list into the list(at ending)

on list, we can use the following built-in functions:
========================================

1.sorted()

2.reversed()

3.zip()

4.enumerate()

5.map()

6.filter()

7.reduce() (it is from python built-in module)

8.range()

9.tuple()

10.set()

11.dict()

example-1:
=========

```
"""create the list"""
l1=[] #empty list
print(l1)
l2=[1,2,3,4,5,1.2,3.4,5.6]
print(l2)
l3=[1,2,3,True,False,"hello","hai",1.234]
print(l3)
l4=[1,2,1,2,1,2,1,2]#list with duplicate elements
print(l4)
"""finding the number of elements in the list
using len() function"""
print(len(l1))#0
print(len(l2))#8
print(len(l3))#8
print(len(l4))#8
```

example-2:
=========

```
"""create the list using list() constructor"""
l1=list(range(1,11))#here we given range() as data
print(l1)
l2=list((1,2,3,4))#here we given tuple as data
print(l2)
l3=list({1,2,3,4,5})##here we given set as data
print(l3)
l4=list({1:2,3:4,5:6,7:8})##here we given dictionary as data
print(l4)
```

example-3:
=========

```
"""create the list using list comprehension"""
l1=[1,2,3,4,5,6,7,8,9,10]
l2=[i for i in l1]
print(l2)
l3=[i for i in l1 if i%2==0]
print(l3)#[2,4,6,8,10]
l4=[i for i in range(1,11) if i%5==0]
print(l4)
l5=[i for i in range(1,11) if i>5 and i<=10]
print(l5)
```

example-4:

=========

"""insert the data into list l1 at end using append()"""
l1=[1,2,3,4,5]
l1.append(20)#[1,2,3,4,5,20]
print(l1)
l1.append(30)#[1,2,3,4,5,20,30]
print(l1)
l1.append(40)#[1,2,3,4,5,20,30,40]
print(l1)
l1.append(50)#[1,2,3,4,5,20,30,40,50]
print(l1)
"""insert the data into list l1 at any position using insert()"""
l1=[1,2,3,4,5]
l1.insert(0,10)#[10,1,2,3,4,5]
print(l1)
l1.insert(2,20)#[10,1,20,2,3,4,5]
print(l1)
l1.insert(4,60)#[10,1,20,2,60,3,4,5]
print(l1)

example-5:
=========

l1=[1,2,3,4,5]
l1.insert(3,500)
print(l1)#[1,2,3,500,4,5]
"""
if we given positive position/index and it is more than the
list size, then insert will simply insert the element at end
"""
l1.insert(10,100)
print(l1)
"""
if we given positive position/index and it is not there,
then insert will simply insert the element at begin
"""
l1.insert(-20,500)
print(l1)

exmaple-6:
=========

insert the elements into the list, without  using any built-in

functions

(i) insert at the begin:
====================================

example:
=======

```
l1=[]
"""insert the element into the list,without using built-in
functions"""
"""insert at begin of the list"""
count=int(input("enter the number of elements to add at begin:"))
while count!=0:
    l1=[int(input("enter element"))]+l1
    count-=1
    print(l1)
```

(ii) insert at the end:
==============================================

example:
=======

```
l1=[]
"""insert the element into the list,without using built-in
functions"""
"""insert element at end of the list"""
count=int(input("enter the number of elements to add at begin:"))
while count!=0:
    l1=l1+[int(input("enter element:"))]
    count-=1
    print(l1)
```

(iii) insert at the any position
=================================================

example:
=======

```
l1=[1,2,3,4,5,6,7,8]
"""insert the element into the list,without using built-in
functions"""
"""insert element at any position of the list"""
position=int(input("enter the position:"))
if position<len(l1):
    l1=l1[:position-1]+[int(input("element:"))]+l1[position-1:]
    print(l1)
```

example-7:
=========

```
"""working with count() function on list"""
l1=[1,2,3,10,20,30,40,1,2,3,10,20,30]
print(l1.count(1))#2
print(l1.count(10))#2
print(l1.count(40))#1
print(l1.count(1000))#0
"""working with index() function on list"""
print(l1.index(10))#3
print(l1.index(40))#6
print(l1.index(3))#2
```

example-8:

=========

"""working with sort() function on list"""
l1=[1,2,3,10,20,30,40,1,2,3,10,20,30]
"""
here when we call the sort() function using sort() method,
sort() will sort the data in ascending order by default
"""
l1.sort()
print(l1)
l1.sort(reverse=True)#for descending order
print(l1)

example-9:
=========

"""working with sorted() function on list"""
l1=[1,2,3,10,20,30,40,1,2,3,10,20,30]
"""
here when we call the sorted() function using sort() method,
sort() will sort the data in ascending order by default
"""
l1=sorted(l1)
print(l1)
l1=sorted(l1,reverse=True)
print(l1)

example-10:
==========

"""working with copy() function on list"""
l1=[1,2,3,4,5]
print(l1)
l2=l1.copy()
print(l2)
"""working with reverse() function on list"""
l1=[1,2,3,4,5,6,7,8,9,10]
l1.reverse()
print(l1)
print(l1[::-1])

example-12:
==========

"""working with extend() function on list"""
l1=[1,2,3,4,5]
l2=[1,2,3,4,5,6,7,8,9,10]
l3=[10,20,30,40,50]
l1.extend(l2)
print(l1)
l2.extend(l3+l1)
print(l2)

example-13:

==========

"""updating the list elements in given list
using indexing"""
l1=[10,20,30,40,50,60,70,80,90,100]
l1[2]=300
print(l1)
l1[8]=900
print(l1)
l1[9]=1000
print(l1)

example-14:
==========
"""delete the elements from the list using
del keyword"""
l1=[10,20,30,40,50,60,70,80,90,100]
"""
del keyword is used to remove the element based on the index
"""
del l1[0]
print(l1)
del l1[5]
print(l1)
del l1[7]
print(l1)

example-15:
==========

"""delete the elements from the list using
pop() function"""
l1=[10,20,30,40,50,60,70,80,90,100]
l1.pop()
print(l1)
l1.pop()
print(l1)
l1.pop(2)#here it is remove the element at index 2 in the list
print(l1)

delete the elements from the list, without using any built-in

function:

(i) delete the element from begin:
==================================================

example:
=======

"""delete the elements from the list, without using any built-in
function"""
l1=[10,20,30,40,50,60,70,80,90,100]
"""delete the element from the begin"""

```
l1=l1[1:]
print(l1)
```

(ii) delete the element at end
==================================================

example:
=======

```
"""delete the elements from the list, without using any built-in
function"""
l1=[10,20,30,40,50,60,70,80,90,100]
"""delete the element from the end"""
l1=l1[:-1]
print(l1)
```

(iii) delete the element from any position:
==================================================

example:
=======

```
"""delete the elements from the list, without using any built-in
function"""
l1=[10,20,30,40,50,60,70,80,90,100]
"""delete the element from the list at any position"""
position=int(input("enter the position:"))
if position<len(l1) and position>0:
   l1=l1[:position-1]+l1[position:]
   print(l1)
```

write a python program , to square the each element of the

given list

example:

l1=[1,2,3,4,5,6,7,8,9,10]

result=[1,4,9,16,25,36,49,64,81,100]

code:
====

```
l1=[1,2,3,4,5,6,7,8,9,10]
print([i*i for i in l1])
```

write a python program , find the each element frequency and

by excluding the duplicates in the given list

x=[1,2,3,4,1,2,3,4,5,6,7,8,3,4,5,6,7]

code:

====

```
x=[1,2,3,4,1,2,3,4,5,6,7,8,3,4,5,6,7]
#remove the duplicate elements from the list
new_list=[]
for i in x:
    if i not in new_list:
        new_list+=[i]
print(new_list)
for i in new_list:
    print(f"{i} frequency:{x.count(i)}")
```

write a python program ,find highest frequency element in the

given list, excluding the duplicate elements:

x=[1,2,3,4,1,2,3,4,5,6,7,8,3,4,5,6,7]
==================================================

code:
====

```
x=[1,2,3,4,1,2,3,4,5,6,7,8,3,4,5,6,7]
#remove the duplicate elements from the list
new_list=[]
#find the frequency of the each element
frequency_list=[]
for i in x:
    if i not in new_list:
        new_list+=[i]
        frequency_list+=[x.count(i)]
#sort the all frequecy list elements in descending order
frequency_list.sort(reverse=True)
for i in new_list:
    if x.count(i)==frequency_list[0]:
        print(i)
```

write a python program find the unique elements from the

given two lists:

x=[1,2,3,4,5,6,7,8,9,10]
y=[3,4,5,6,10,20,30,40,6,7,8]

================================================

code:
====

```
x=[1,2,3,4,5,6,7,8,9,10]
y=[3,4,5,6,10,20,30,40,6,7,8]
#output:[1,2,9,20,30,40]
result=[]
for i in x+y:
    if i not in y and i in x:
```

```
        result+=[i]
    if i in y and i not in x:
        result+=[i]
print(result)
```

calculating the time complexities on lists operations:
=============================================

(i) insertion operation:
====================

(i) insertion at begin:
==================

```
l1=[1,2,3,4,5,6,7,8,9,10]
element=int(input("enter the element:"))
l1=[element]+l1
print(l1)
#time complexity:O(n)
```

(ii)insertion at end:
=================

```
l1=[1,2,3,4,5,6,7,8,9,10]
element=int(input("enter the element:"))
l1=l1+[element]
print(l1)
#time complexity:O(1)
```

(iii) insertion at any position:
=========================

```
l1=[1,2,3,4,5,6,7,8,9,10]
element=int(input("enter the element:"))
position=int(input("enter the position:"))
if position<len(l1):
    l1=l1[0:position-1]+[element]+l1[position-1:]
print(l1)

#time complexity:O(1)
```

(ii) deletion operation on list:
=========================

(i) deletion at begin:
===============================

```
l1=[1,2,3,4,5,6,7,8,9,10]
"""delete the element from the starting"""
l1=l1[1:]
print(l1)
#time complexity:O(n)
```

(ii)deletion at end :
==============================

```
l1=[1,2,3,4,5,6,7,8,9,10]
"""delete the element from the ending"""
l1=l1[:-1]
print(l1)
#time complexity:O(1)
```

(iii)deletion at any position :
======================

```
l1=[1,2,3,4,5,6,7,8,9,10]
"""delete the element from any position"""
position=int(input("enter the position:"))
if position<len(l1):
    l1=l1[:position]+l1[position+1:]
print(l1)
#time complexity:O(n)
```

(iii)update the element:
==================

(i) update the begin:
========================

```
l1=[1,2,3,4,5,6,7,8,9,10]
"""update the element at begin"""
l1[0]=int(input("enter the element:"))
print(l1)
#time complexity:O(1)
```

(ii) update the end:
==========================

```
l1=[1,2,3,4,5,6,7,8,9,10]
"""update the element at end"""
l1[-1]=int(input("enter the element:"))
print(l1)
#time complexity:O(1)
```

(iii) update the any position element:
================================

```
l1=[1,2,3,4,5,6,7,8,9,10]
"""update the element at end"""
position=int(input("position:"))
if position<len(l1) and position>=0:
    l1[position-1]=int(input("enter the element:"))
print(l1)
#time complexity:O(1)
```

traverse the complete list using for loop:
==================================

code:
=====

```
l1=[1,2,3,4,5,6,7,8,9,10]
"""traverse the list using for loop"""
for i in l1:
    pass
#time complexity:O(n)
```

working with map() function on list and calculate the

time complexity of the map() function:
================================================

map() function is a built-in function and takes a function as

argument and apply on function on each element of the given

iterable

map() function always takes two arguements ,those are

one is function and another is iterable(list,tuple,set,range()....)

in python, map() function will be used to "change the data of

given iterable based on the function in python"

map(), will return the result as a object

syntax for map() function:
=====================

map(function or function_name, iterable_name)

example-1:
=========

```
l1=[1,2,3,4,5,6,7,8,9,10]
"""apply the map() function on l1"""
l1=list(map(lambda x:x*x,l1))
print(l1)
#time complexity:O(n)
```

working with filter() function in python:
=================================

this is also takes "a function as argument and takes data as iterable, return the result which are satisfies the condition

which is given in the function", it will not change the data like

map() function

it will also return the result  as object

syntax:
======

filter(function/function_name,iterable_name)

example-1:
=========

```
l1=[1,2,3,4,5,6,7,8,9,10]
"""apply the map() function on l1"""
l1=list(filter(lambda x:x>5,l1))
print(l1)#[6,7,8,9,10]
#time complexity:O(n)
```

example-2:
=========

```
l1=[1,2,3,4,5,6,7,8,9,10]
"""apply the filter() function on l1"""
l1=list(filter(lambda x: x>5 and x%2==0,l1))
print(l1)#[6,8,10]
#time complexity:O(n)
```

example-3:
=========

```
l1=[1,2,3,4,5,6,7,8,9,10]
"""apply the filter() function on l1"""
l1=list(filter(lambda x:len([i for i in range(1,x+1) if x%i==0])==2,l1))
print(l1)#[6,8,10]
#time complexity:O(n)
```

example-4:
=========

```
s1=["aaa","abc","cba","dac","efg","klm"]
"""apply the filter() function on l1"""
l1=list(filter(lambda x:x[1]=='b',s1))
print(l1)
#time complexity:O(n)
```

example-5:
=========

```
s1=["aaa",'aba',"abab","aaaaba"]
"""apply the map() function on l1"""
```

```
l1=list(map(lambda x:x[0:3],s1))
print(l1)
#time complexity:O(n)
```

tuple:
=====

tuple used in python, to store the "collection of elements or

group of elements"

tuple can store the "duplicate elements"

tuple can store the "any type of values"

tuple can store any number of values as data

tuple allow the indexing same as "list or string"

tuple is a immutable type(it means it does not allow the

insertion ,updation ,deletion)

to create the tuple, we will use "tuple() constructor or ()"

example:
=======

```
"""create the tuple"""
"""here tuple is empty"""
t1=()
print(t1)#()
"""here tuple is having different type elements"""
t2=(1,2,3,4,5,1.2,3.4,5.6,7.8,9.0)
print(t2)
"""here tuple is having duplicate elements"""
t3=(1,2,3,1,2,3)
print(t3)
"""indexing on tuple"""
print(t2[0])#1
print(t2[-8])#2
"""slicing on tuple"""
print(t2[4:7])
print(t2[-5:-1])
print(t3[::-1])
```

functions, we can apply on the tuple:
============================

1.len() <=== using this we can find the "number of elements

present in the tuple

2.count()<=== using this we can find the "frequency of the each element in the tuple"

3.index()<== using this we can find the "given element index

in the tuple"

4.list()<=== using this, we can convert the given tuple into

list , using tuple() we can convert the given list

into tuple

example:
=======

```
"""working with tuple using functions"""
t1=(10,20,30,40,50,60,70,80,90,100)
"""working with index() function"""
print(t1.index(80))#7
print(t1.index(50))#4
"""working with count() function"""
print(t1.count(20))#1
print(t1.count(100))#1
print(t1.count(1000))#0
"""workingw with list() function"""
l1=list(t1)
print(l1)
"""working with len() function"""
print(len(t1))#10
```

to serach the given element in the tuple, we will use

membership operators:
==================================================

```
"""working with tuple using functions"""
t1=(10,20,30,40,50,60,70,80,90,100)
"""working with membership operators"""
element=int(input("eneter the element to search:"))
if element in t1:
    print("Element is Found!")
else:
    print("Element is not Found!")
```

time complexities on tuple operations:
==============================

calculate the time complexity of the operation "getting element

from the tuple using index"
==================================================

code:

```
====
"""calculate the time complexity of the operation
"getting element
from the tuple using index"""
t1=(10,20,30,40,50,60,70,80,90,100)
index=int(input("enter the index:"))
if index>0 and index<len(t1):
    print(t1[index])
else:
    print("please give the right index,or index may not be there")
#time complexity: O(1)
```

calculate the time complexity of the operation "getting element

from the tuple using index()"
==================================================

code:

```
====

"""calculate the time complexity of the operation
"getting element
from the tuple using index"""
t1=(10,20,30,40,50,60,70,80,90,100)
element=int(input("enter the index:"))
t1.index(element)
#time complexity: O(n)
```

calculate the time complexity for "searching an element

in the tuple"
==================================================

code:

```
====

t1=(10,20,30,40,50,60,70,80,90,100)
element=int(input("enter the index:"))
if element in t1:
    print("Found")
else:
    print("Not Found")
#time complexity:O(n)
```

calcaute the time complexity of the concatenating the two

tuples :
=================================================

code:

```
====

t1=(10,20,30,40,50,60,70,80,90,100)
```

t2=(1,2,3,4,4,5,6,7,8,9,9,0,10,100)
"""concatenate the two tuples"""
print(t1+t2)
#time complexity:O(n1+n2)

calculate the time complexity of the "count the given element

is how many times is repeated in the tuple"

================================================

code:
====

t2=(1,2,3,4,4,5,6,7,8,9,9,0,10,100)
"""count the given element repeated how many times"""
print(t2.count(3))
#time complexity:O(n)

calculate the time complexity of the given code:
=====================================

code:
====

t2=(1,2,3,4,4,5)
"""repetititon of the tuple for n times"""
print(t2*int(input("enter the times")))
#time complexity:O(n*m) where m=number of times

calculate the time complexity for the given code:
=======================================
code:
====
t2=(10,20,30,40,50,60,70,80,90,100)
print(t2[2:8])
#time complexity:
O(len(tuple_name[m:n]))
===>O(n) ==>where n means size of the slice

Strings in python:
==============

in python strings, we can store using "single or double or

triple quotes"

string can be "empty" in python

string can have "any number of chracters"

string can have "duplicate chracters"

string can have "any type of characters" on strings, we can apply the "indexing and slicing"

in python, string is a immutable type(it means we can not apply

insertion/ update and delete)

example:
=======

```
"""create the string"""
s1="abc"
print(s1)
s2=’abc’
print(s2)
s3="""abc"""
print(s3)
"""indexing on the strings"""
print(s1[0])#a
print(s1[-1])#c
"""slicing on strings"""
print(s1[:])
print(s1[::-1])
```

on strings, we can apply the following functions:
======================================

1.len()===>it used to find the length of the string

code:
====

```
"""length of the string"""
print(len(’’))#0
print(len(’ ’))#1
print(len("hello world"))
```

2.count():
========

```
"""frequency of the character in the string"""
s1="hello world"
print(s1.count("l"))#3
print(s1.count("o"))#2
print(s1.count("H"))#0
```

3.index(): <== it will return given character index position in
=======
string

code:

```
s1="hello world"
char=input("enter the character:")
```

print(s1.index(char))
#time complexity: O(n)

4.upper():
========

it is used to "convert the  given string lower case letters into upper case letters"

example:
=======
s1="hello world"
print(s1.upper())
print("HellWOrld".upper())
#time compexity:O(n)

5.lower():
=======

it is used to convert the given string characters into "lower"

case

example:
=======

s1="HELLO WORLD"
print(s1.lower())
print("HellWOrld".lower())
#time compexity:O(n)

6.swapcase():
===========

it is used to convert the "lower case letters into upper case"

and "upper case letters into lower case" in the given string

example:
=======

s1="HELLO WORLD"
print(s1.swapcase())
print("HellWOrld".swapcase())
#time compexity:O(n)

7.casefold():
=========

it is used to convert the given  string characters into lower case

example:
=======

```
s1="HELLO WORLD"
print(s1.casefold())
print("HellWOrld".casefold())
#time compexity:O(n)
```

8.isspace():
==========

it is used to check "given string having all characters are spaces

or not"

it will give the result either "True or False"

example:
=======

```
s1=" "
print(s1.isspace())
print("Hell WOrld".isspace())
#time compexity:O(n)
```

9.isdigit():
========

it is used to check all "characters in  the  are digits or not"

if all are digits in the given string, it will return "True" as result

if all are not digits in the given string, it will retrun "False"

as result

example:
=======

```
s1="1234"
print(s1.isdigit())
print("HellWOrld1234".isdigit())
#time compexity:O(n)
```

10.islower():
==========

it will check the all characters of the string are "lower case"

or not

if all are lower case in the given string, then it will return "True"

as result

if all are not lower case in the given string, then it will return

"False" as result

example:
=======

```
s1="abcd"
print(s1.islower())
print("HellWOrld1234".islower())
#time compexity:O(n)
```

11.isupper():
==========

it will check the all characters of the string are "uppercase"

or not

if all are upper case in the given string, then it will return "True"

as result

if all are not upper case in the given string, then it will return

"False" as result

example:
=======

```
s1="ABCD"
print(s1.isupper())
print("HellWOrld1234".isupper())
#time compexity:O(n)
```

12.join():
========

this is used to join the "given character" as character between

character of the string

example:
=======

```
s1="ABCD"
print("=".join(s1))
s1="A B C D"
print("**".join(s1))
#time compexity:O(n)
```

13.split():
=========

this is used to divide or split the given into "n" sub strings

example:
=======

```
s1="hello world"
print(s1.split(" "))
print(s1.split("w"))
print("abc bca cab".split("a"))
print("abc cab dfg".split(" "))
#time compexity:O(n)
```

write a the python programs on the following:

without using any built-in function
====================================

1.remove the spaces in the string:
===========================

example:
=======

input: ab bc c   ab cd

output:abbcabcd

code:
====

```
"""remove the spaces in the string"""
str1=input("Enter the string:")
res=''
for i in str1:#ab bc c   ab cd
  if i!=" ":res+=i
  print(res)
#time complexity:O(n)
```

2.print the two strings common characters as a string:
============================================

example:
=======

input1 :abcdef

input2:bcadfgh

ouput:abcdf

code:
====

```
"""remove the spaces in the string"""
```

```
str1=input("Enter the string1:")
str2=input("Enter the string1:")
res=''
for i in str1:#abcdef
  if i in str1 and i in str2:
      res+=i
print(res)
#time complexity:O(n)
```

3.print the two strings not common characters as a string:
==============================================

input1 :abcdef

input2:bcadfgh

ouput:egh

code:
====

```
"""remove the spaces in the string"""
str1=input("Enter the string1:")
str2=input("Enter the string1:")
res=''
for i in str1+str2:#abcdef
  if i in str1 and i not in str2:
      res+=i
  if i in str2 and i not in str1:
      res+=i
print(res)
#time complexity:O(n)
```

4.write a python program print the characters which common

and also same position in the two strings ,display the result as

string:

=================================================

input1: abcd

input2: abcdfg

result:  abcd

code:
====

```
"""remove the spaces in the string"""
str1=input("Enter the string1:")
str2=input("Enter the string1:")
index=0
res=''
```

```
for i in str1:
    if str1[index]==str2[index]:
        res+=i
    index+=1
print(res)
#time complexity:O(n)
```

write a python program for the following:
================================

input: aaaabbccddgffgghhjj

ouput:a4b2c2d2g3f2h2j2

code:
=====

```
"""remove the spaces in the string"""
str1="aaaabbccddgffgghhjj"
res=''
"""remove the duplicate characters"""
for i in str1:
    if i not in res: res+=i
for i in res:
    count=0
    for j in str1:
        if i==j:
            count+=1
    print(f"{i}{count}",end="")
```

14.replace():
==========

this is function is used to replace the given string in the

wherever, the string matches

example:
=======

```
"""working with strings"""
s1="hello world"
s1=s1.replace("l","L")
print(s1)
s1="hello world"
s1=s1.replace("l","L",1)
print(s1)
s1="hello world"
s1=s1.replace("l","L",2)
print(s1)
```

15.copy():

========

copy() function is used to "copy the string" as list, it is not

a string function

example:
=======
```
"""working with strings"""
s1="hello world"
s1="".join(list(s1).copy())
print(s1)
```

16.reverse():
==========

reverse() function is function of list and using this we can make

a string into reverse string

example:
=======

```
"""working with strings"""
s1="hello world"
print(s1[::-1])
s1=list(s1)
s1.reverse()
s1="".join(s1)
print(s1)
```

17.center():
==========

center() is used to "Add the space at starting and ending"

of the given string

example:
=======

```
"""working with strings"""
"""when we use the center() function, the center() function
value must be more than length of the string, to see the
spaces"""
s1="hello world"
s1=s1.center(5)
print(s1)
s1=s1.center(15)
print(s1)
s1=s1.center(30)
print(s1)
```

18.strip():
========

it used to remove or trail the spaces in the string at begin or

end

example:
=======

```
"""working with strings"""
s1="hello world"
s1=s1.strip()
print(s1)
s1="    hello world"
print(s1)
s1=s1.strip()
print(s1)
s1="hello     "
print(len(s1))
s1=s1.strip()
print(s1)
print(len(s1))
s1="hello world"
s1=s1.strip("hello ")
print(s1)
```

19.lstrip():
========

it is used to "Remove the spaces from starting of the string"

only

example:
=======

```
"""working with strings"""
s1="hello world"
s1=s1.lstrip()
print(s1)
s1="    hello world"
print(s1)
s1=s1.lstrip()
print(s1)
```

20.rstrip():
=========

it is used to "Remove the spaces from ending of the string"

only

example:
=======

```
"""working with strings"""
s1="hello world"
s1=s1.rstrip()
print(s1)
s1="hello world       "
print(s1)
print(len(s1))
s1=s1.rstrip()
print(s1)
print(len(s1))
```

21.title():
=======

it is used to "make the every word and very new word starting

character as uppercase"

example:
=======
```
"""working with strings"""
s1="hello world"
s1=s1.title()
print(s1)
```

22.capitalize():
===========

it is used to "make the starting character of string has to be

upper case"

example:
=======

```
"""working with strings"""
s1="hello world"
s1=s1.capitalize()
print(s1)
```

23.find():
=======

this function is used to "check the given string is present in

the string or not"

example:
=======

```
"""working with strings"""
s1="hello world"
position=s1.find("hello")
print(position)#index
s1="hello world"
position=s1.find("L")
print(position)#index
s1="hello world"
position=s1.find("l")
print(position)#index
```

note:
====

find() function will return "index", if the given string found in

the string

find() function will return "-1", if the given string is not found

in the string

24.startswith():
============

this function is going to check the given string is starting string

or not in the string

this function will return the result "either True or False"

example:
=======

```
"""working with strings"""
s1="hello world"
print(s1.startswith("hai"))#False
print(s1.startswith("Hai"))#False
print(s1.startswith("hello"))#True
```

24.endswith():
============

this function is going to check the given string is ending string

or not in the string

this function will return the result "either True or False"

example:
=======

```
"""working with strings"""
```

```
s1="hello world"
print(s1.endswith("hai"))#False
print(s1.endswith("Hai"))#False
print(s1.endswith("hello"))#False
print(s1.endswith("world"))#True
```

python iterators:
=============

iterator are used to retrieve the data from the iterables

iterators are used on the "list, tuple, set, range(), dictionary,

................."

iterators can used to retrieve the data "one after another or

one by one" from the iterables(list, string, tuple, range,....)

using iterators, we can create iterator objects

in python, in order to work with iterators, we will use the

following functions:
==================================================

1.iter() ===>it is used to create the iterator for the given

iterable

2.next()==> it is used to retrieve the data from the iterator

one by one

example-1:
=========

```
"""working with iterators"""
l1=[1,2,3,4,5,6,7,8,9]
"""creating the iterator object using
iter()"""
i1=iter(l1)
"""retrive the data from iterator"""
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
#print(next(i1))#StopIteration
```

example-2:
=========

"""working with iterators"""
s1="hello world"
"""creating the iterator object using
iter()"""
i1=iter(s1)
print(i1)
"""retrive the data from iterator"""
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))

example-3:
=========

"""working with iterators"""
t1=(1,2,3,4,5,6,7,8,9)
"""creating the iterator object using
iter()"""
i1=iter(t1)
print(i1)
"""retrive the data from iterator"""
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))
print(next(i1))

working of the iterator:
==================

l1=[1,2,3,4,5,6,7]

create the iterator using "iter()" ,

i1=iter(l1)

1 2 3 4 5 6 7

^
||

print(nex(i1)) ===>1

2 3 4 5 6 7

^
||

print(next(i1)) ====>2

3 4 5 6 7

^
||

print(next(i1)) ====>3

.
.
it complete the process, until all elements are deleted

example:
=======

```
"""working with iterators"""
t1=(1,2,3,4,5,6,7,8,9)
"""creating the iterator object using
iter()"""
i1=iter(t1)
print(i1)
"""retrive the data from iterator"""
print(next(i1))#1
t1=tuple(i1)
print(t1)
i1=iter(t1)
print(next(i1))#2
t1=tuple(i1)
print(t1)
i1=iter(t1)
print(next(i1))#2
t1=tuple(i1)
print(t1)
```

note:
=====

when we call the iterator with next(), the element which is

pointing by next() will be deleted automatically from the

iterator

when we are using next() function on the iterator, if there is no element in the iterator, iterator will raise an error called

"StopIteration"

example:
=======

```
"""working with iterators"""
t1=(1,2,3,4,5,6,7,8,9)
"""creating the iterator object using
iter()"""
i1=iter(t1)
print(i1)
"""retrive the data from iterator"""
for i in i1:
    print(i)
t1=tuple(i1)
print(t1)#()
```

python generators:
===============

generator  is a function in python, when we say a function

is a generator, the function should have at least one "yield"

statement

to create the generator in python, we will use the following

syntax:
================================================

```
def generator_name(arg1,arg2,arg3,....argn):

      yield  value1

      yield  value2

      .
      .
     yiled   value3
```

example-1:
=========

```
"""working with generator"""
def mygenerator():
    yield  1
    yield  2
    yield  3
    yield  4
    yield  5
```

```
"""call the generator"""
result=mygenerator()
print(type(result))
print(next(result))
print(next(result))
print(next(result))
print(next(result))
print(next(result))
print(next(result))
```

generator create using function and call the generator like

a "iterator", generator works like iterator, that is reason

generator can be called as "custom or user defined iterator"

example-2:
=========

```
"""working with generator"""
def mygenerator():
    yield  1
    yield  2
    yield  3
    yield  4
    yield  5
"""call the generator"""
for i in mygenerator():
    print(i)
```

example:
=======

```
"""working with generator"""
def mygenerator():
    yield  1
    yield  2
    yield  3
    yield  4
    yield  5
"""call the generator"""
result=list(mygenerator())
print(result)
result=tuple(mygenerator())
print(result)
result=set(mygenerator())
print(result)
```

generator as expression:
===================

```
result=(i for i in iterable_name if condition)
```

example:
=======

```
"""working with generator as expression"""
result=(i for i in range(1,11) if i>5)
print(result)
print(list(result))
result=(i for i in range(1,11) if i<=5)
print(result)
print(list(result))
```

python nested lists, tuples:
====================

in python we can create

"a list inside another list"

" a tuple inside another tuple"

example:
=======

```
"""working with nested list"""
l1=[1,2,3,[10,20,30,40],5,6]
print(l1)
print(len(l1))#6
print(l1[0])#1
print(l1[3])
print(l1[3][0])#10
print(l1[3][2])#30
l1=[[[1,2,3],[4,5,6]]]
print(len(l1))#1
print(l1[0])
print(len(l1[0]))#2
print(l1[0][0])
print(len(l1[0][0]))#3
print(l1[0][1])
print(len(l1[0][1]))#3
```

example-2:
========

```
"""working with nested list"""
l1=[1,2,[4,5,6,7,8]]
print(l1)
print(len(l1))#3
print(l1[2])#[4,5,6,7,8]
print(len(l1[2]))#5
l2=[[1,2,3,4]]
print(len(l2))#1
print(l2[0][3])#4
l3=[[[1,2,3,4]]]
```

```
print(l3[0][0][3])#4
print(len(l3))#1
print(len(l3[0]))#1
print(len(l3[0][0]))#4
```

example-3:
=========

```
"""working with nested tuple"""
t1=(10,20,(30,40,50))
print(t1)
print(len(t1))#3
print(t1[2])#(30,40,50)
t2=(1,2,3,(50,60,70))
print(t2[3][2])
t2=(1,2,(50,60,70,(100,200)))
print(t2[2][3][1])#200
t3=(1,2,(50,60,(100,200,(400,500))))
print(t3[2][2][2][1])#500
t4=(1,2,(100,(500,600),(200,(900,1000,(3,6789)))))
print(t4[2][2][1][2][1])#6789
```

python shallow and deep copy:
========================

in order to work with shallow copy and deep copy, we will use

a module called "copy" in Python

to perform the shallow copy, in python copy module, we will

use a function called "copy()"

to perform the deep copy, in python copy module, we will use

a function called  "deepcopy()"

example-1:
=========

working with shallow copy:
=====================

```
"""working with shallow copy"""
#import the copy module
import copy
l1=[1,2,3,4,5,6,7,8,9,10]
print(l1)
#apply the shallow copy on l1, copy the data into l2
l2=copy.copy(l1)
print(l2)
#appned the 100 into l1
l1.append(100)
print(l1)
```

```
print(l2)
l1=[[1,2,3,4,5]]
print(l1)
l2=copy.copy(l1)
print(l2)
l1[0].append(1000)
print(l1)
print(l2)
l2[0].append(2000)
print(l1)
```

note:
====

when we apply the shallow copy on the normal lists, then both

original list and shallow list are independent with each other

when we apply the shallow copy on list with nested lists, all

nested lists of the original list and shallow copy list will

dependent with each other(the change what we made on

original nested list, we will also see the same on shallow copy

nested list)

example-2:
=========

```
"""working with deep copy"""
#import the copy module
import copy
l1=[1,2,3,4,5,6,7,8,9,10]
print(l1)
#apply the deep copy on l1, copy the data into l2
l2=copy.deepcopy(l1)
print(l2)
#appned the 100 into l1
l1.append(100)
print(l1)
print(l2)
l1=[[1,2,3,4,5]]
print(l1)
l2=copy.deepcopy(l1)
print(l2)
l1[0].append(1000)
print(l1)
print(l2)
l2[0].append(2000)
print(l1)
print(l2)
```

note:
====

when we apply the deep copy on the normal lists, then both

original list and deep list are independent with each other

when we apply the deep copy on list with nested lists, all

nested lists of the original list and deep copy list will

independent with each other(the change what we made on

original nested list, we will never effect on deep copy

nested list)

python sets:
==========

set is a "collection of unique/distinct elements"

set will never allow the "duplicate elements"

set is may or may not store the elements, the way we given

to the set , that is reason set is also called as "un-ordered

collection" in Python

in python sets are not allow the following:
=================================

1.indexing

2.slicing

3.nested sets

python sets are mutable ( on sets we can apply insertion,

updation, deletion)

in python, to create the set, we will use "{ }" or "set()"

constructor

in python sets, we can store any number of elements and

any type of elements

to find the number of elements in the set, we will use a

function called "len()"

example-1:
=========

"""working with Sets"""
s1={}
print(s1)#{}
print(type(s1))#
s2=set()
print(s2)
print(type(s2))
s3={1,2,3,4,1.2,3.4,"hello","hai"}
print(s3)
s4={1,2,2,1,1,1,2,2,2}
print(s4)
"""finding the length of the set,we will use
len() function"""
print(len(s3))
print(len(s4))

operations on the sets:
==================

1.union:
==============================================

union is used to "combine the two sets into single without

any duplicate elements"

example:
=======

s1={1,2,3,4,5,6}

s2={2,3,4,5,7,8,9}

s1 union s2 ===>{1,2,3,4,5,6,7,8,9}

in python to apply the union, we will use the following ways:
=================================================

1.using "|" operator

2.using "union()" function

example:
========

"""working with Sets"""
s1={1,2,3,4,5,6}
s2={2,3,4,5,7,8,9}
print(s1|s2)
print(s1.union(s2))

2.intersection:
=============

intersection means "it will give the common elements between

the given two sets"

s1={1,2,3,4,5}

s2={4,5,6,7}

s1 intersection s2 ===>{ 4,5}

in python, to apply the intersection we will use the following

ways:

1. using & operator

2. using intersection() function

3. using intersection_update() function

example:
=======

```
"""working with sets"""
s1={1,2,3,4,5}
s2={4,5,6,7}
"""in python to apply the intersection using & operator"""
print(s1&s2)
"""in python to apply the intersection using intersection() fucntion"""
print(s1.intersection(s2))
"""in python to apply the intersection using intersection_update()
"""
s1.intersection_update(s2)
print(s1)
```

3.difference:
==========

difference of the two sets means "it will give the elements

which are not there in the another set"

s1={1,2,3,4,5}

s2={4,5,6,7}

s1-s2===> {1,2,3}

s12-s1===>{6,7}

in Python, to implement the  difference of the two sets,

we will use the following ways:
==================================================

1.using "-" operator

2.using "difference()" function

3.using "difference_update()" function

example:
========

```
"""working with sets"""
s1={1,2,3,4,5}
s2={4,5,6,7}
"""difference on the sets using "-" operator"""
print(s1-s2)
print(s2-s1)
"""difference on the sets using difference() function"""
print(s1.difference(s2))
print(s2.difference(s1))
"""difference on the sets using difference_update() function"""
s1.difference_update(s2)
print(s1)#s1-s2({1,2,3})
```

4.symmetric difference:
===================

symmetric difference means "finding the unique elements from

the two sets"

s1={1,2,3,4,5}

s2={4,5,6,7}

s1 symm.diff s2={1,2,3,6,7}

in python, to find the symmetric difference between the two

sets, we will use the following ways:
====================================================

1.using "^" operator:
==================

2.using symmetric_difference() function

3.using symmetric_difference_update() function

example:
=======

```
"""working with sets"""
s1={1,2,3,4,5}
s2={4,5,6,7}
"""symmetric difference on the sets using "^" operator"""
print(s1^s2)
"""symmetric difference on the sets using
"symmetric_difference()" operator"""
print(s1.symmetric_difference(s2))
"""symmetric difference on the sets using
symmetric_difference_update()"""
s1.symmetric_difference_update(s2)
print(s1)
```

example:
=======

```
"""usecase-1"""
s1="""abc wxyz jkl abc dfg hju ikl nop pqr tyu klo ukj"""
s2="""abc wxyz jkl abc dfg hju ikl nop pqr tyu klo hlo plm png tyu"""
s1=set(s1.split(" "))
s2=set(s2.split(" "))
"""similar words in the given two paragrapghs"""
print(s1&s2)
"""distinct words from the given two paragraphs"""
print(s1^s2)
"""give me the words which are not there in the s1 , but in s2"""
print(s2-s1)
"""give me the words which are not there in s2, but in s1"""
print(s1-s2)
```

example-2:
=========

```
"""usecase-2:
in the given paragraph eliminate the vowels from the each word
display the paragraph with vowels in the each word as a result
"""
s2="""abc wxyz jkl abc dfg hju ikl nop pqr tyu klo hlo plm png tyu"""
```

5.insertion:

=========

to insert the data into set, in python we will use  a function

called "add()"

example:
=======

```
"""working with sets"""
s1={1,2,3,4,5,6,7,8,9,10}
s1.add(20)
s1.add(30)
s1.add(40)
print(s1)
```

6.update:
========

to update the data in set, in python we will use  a function

called "update()"

update() function will take data as "iterable"

in python, updating the set means "Adding the data, but not

changing a particular value

example:
=======

```
"""working with sets"""
s1={1,2,3,4,5,6,7,8,9,10}
s1.update([10,20,30])
print(s1)
s1.update((100,200,300))
print(s1)
s1.update("50")
print(s1)
```

7.delete:
=======

to delete the elements from the set, we will use the following

functions:
=================================================

1.remove():
=============================================

when we are using this function, we will take "an element"

,the element in the remove() function defines the "element

which is deleting from the set"

if the given element in the remove() function and the element

is not there in the set, remove() function will raises an error

2.discard():
=========

when we are using this function, we will take "an element"

,the element in the discard() function defines the "element

which is deleting from the set"

if the given element in the discard() function and the element

is not there in the set, discard() function will not raise any error

example:
=======

```
"""working with sets"""
s1={1,2,3,4,5,6,7,8,9,10}
s1.remove(10)
print(s1)
s1.remove(3)
print(s1)
s1.discard(100)
print(s1)
s1.discard(6)
print(s1)
#s1.remove(100)
```

set comprehension:
===============

using this concept, we can create any set using existing set

or using any iterable in python

syntax:
======

{variable_name for i in variable_name if condition}

example:
=======

```
"""working with sets"""
```

```
s1={1,2,3,4,5,6,7,8,9,10}
s2={i for i in s1 if i>5}
print(s2)
s3={i for i in range(10,110,10)}
print(s3)
s4={i for i in [1,2,3,4,5,6] if i%2==0}
print(s4)
```

in sets, we will have the other functions:
===============================

clear()<== using this we can remove all elements from the set,

but not set

copy()<=== using this we can copy the all elements of the set

issubset()<=== it is used to check the given set is subset or not

issuperset() <=== it is used to check the given set is super set

or not

isdisjoint()<=== it used to check the the two sets are disjoint or

not

note:
====

issubset(), issuperset(), iddisjoint() will return result either True

or False

disjoint sets means "the two sets will never have same

elements"

subset is a set and the all  elements in the set are part of

another set

superset is a set and which is having all elements of subset

or

the subset which is drawn from set called "super set"

example:
=======

```
"""working with sets"""
s1={1,2,3,4,5,6,7,8,9,10}
s2={4,5,6,7}
s3={10,11,12,13}
```

```
print(s1.issuperset(s2))
print(s1.issuperset(s3))
print(s2.issubset(s1))
print(s3.issubset(s1))
print(s1.isdisjoint(s2))
print(s1.isdisjoint({11,12,13}))
"""in python, we will use the following:

    >(super set)
    <(subset)
    ==(sets equal or not)
"""
print(s1>s2)#s1 is super set of s2
print(s2<s1)#s2 is subset of s1
print(s2==s3)#s2 and s3 having same elements or not
s1.clear()
print(s1)
s4=s2.copy()
print(s4)
```

to check the given value is present in the set or not, we will

use the following operators:
================================================

1.in

2.not in

example:
=======

```
"""working with sets"""
s1={1,2,3,4,5,6,7,8,9,10}
print(1 in s1)
print(2 in s1)
print(1 not in s1)
print(2 not in s1)
print(100 not in s1)
```

python array:
===========

array data structure is "similar to python list"

it means array data structure can store any number elements

like python list

array can allow both indexing and slicing

array can allows insertion/ updation/ deletion (in python

array is mutable type)

in python, array is not a "built-in" data structure like python

list

in python, in order to work with arrays, we will use "array"

module

syntax:

```
import array as arr
```

note:

the main difference between array and python list is ,

in python array, we will have "Type safety"( it means in array

we can store only similar kind of elements"

in python,

when we are creating the array, we will specify the "type"

of the array, using a symbol like a character

where the "symbol" denotes "type of the array"

in python, to create the array, we will use a function called

array() function

syntax:

```
array(type, [val1,val2,val3,..........valn])
```

this array() function is part of arrays module

| symbol | type |
|---|---|
| ’b’ | signed character |
| ’B’ | un- signed character |
| ’h’ | signed short |
| ’H’ | un-signed short |
| ’i’ | signed integer |
| ’I’ | un-signed integer |

’u’ Unicode character

’f’ float

’d’ double

’q’ signed long long

’Q’ un-signed long long

example-1:
=========

```
import array
"""we can create the array"""
a1=array.array(’i’,[1,2,3,4,5,6,7])
print(a1)
print(type(a1))
a1=array.array(’l’,[11,2,3,4,5,6,7])
print(a1)
print(type(a1))
a1=array.array(’f’,[1,2,3,4,5,6,7])
print(a1)
print(type(a1))
a1=array.array(’d’,[11,2,3,4,5,6,7])
print(a1)
print(type(a1))
```

in array module, we will have the following functions:
==========================================

1.insert() <=== used to add element in the array at any position

2.append()<=== used to add element in the array at end

3.index()<==== used to find the given element index in the array

4.len()<====== used to find the number of elements in the array

5.remove()<=== used to remove the given element

6.pop()<===== used to remove the element from array end or position

7.buffer_info()<== this will give the array address and size of the array

8.typecode<=== it will return type code of the array

9.fromlist()<=== create the array using given list

10.tolist()<=== convert the given array into list

example-2:
=========

```
import array
"""get the type code of the array using
typecode from array module
get the array size and address using
buffer_info()
"""
a1=array.array(’i’,[1,2,3,4,5,6,7])
print(a1.typecode)
print(a1.buffer_info())
print(len(a1))
a1=array.array(’l’,[11,2,3,4,5,6,7])
print(a1.typecode)
print(a1.buffer_info())
a1=array.array(’f’,[1,2,3,4,5,6,7])
print(a1.typecode)
print(a1.buffer_info())
a1=array.array(’d’,[11,2,3,4,5,6,7])
print(a1.typecode)
print(a1.buffer_info())
```

exmaple-3:
=========

```
import array
"""
create the array using list, with help
of fromlist() fucntion
convert the array into list using tolist()
fucntion
"""
a1=array.array(’i’,[1,2,3,4,5,6,7])
print(dir(a1))
"""convert the a1 into list"""
print(a1)
a1=a1.tolist()
print(a1)
print(type(a1))
"""create the array using list a1"""
a2=array.array(’i’,[])
a2.fromlist(a1)
print(a2)
a3=array.array(’l’,a1)
print(a3)
```

Leet code questions SET-1(Basic Level):
==============================

---

1. Reverse an Array

Description: Reverse the order of elements in an array.

Example Input: [1, 2, 3, 4, 5]

Example Output: [5, 4, 3, 2, 1]

code:
=====

```
"""
question-1:
Reverse the order of elements in an array.
"""
a1=[1,2,3,4,5,6,7,8,9,10]
"""way-1"""
print(a1[::-1])
"""way-2"""
a1.reverse()
print(a1)
```

---

2. Two Sum Problem

Description: Find two numbers that sum up to a target value.

Example Input:

nums = [2, 7, 11, 15], target = 9

Example Output: [0, 1] (Since nums[0] + nums[1] = 9)

code:
=====

```
"""
question-2:
Find two numbers that sum up to a target value.
"""
nums = [2, 7, 11, 15]
target = int(input("Target:"))
nums=[i for i in nums if i<=target]
print(nums)
sum=0
position=0
for i in range(0,len(nums)):
    if target==nums[i]:
        print(f"{[i]}")
    else:
```

```
    for j in range(i+1,len(nums)):
        if target==nums[i]+nums[j]:
            print(f"[{i},{j}]")
```

---

3. Maximum Subarray Sum (Kadane’s Algorithm)

Description: Find the contiguous subarray with the maximum sum.

Example Input: [-2, 1, -3, 4, -1, 2, 1, -5, 4]

Example Output: 6 (From subarray [4, -1, 2, 1])

---

code:
====

```
"""Find the contiguous subarray
with the maximum sum.
"""
nums=[-2, 1, -3, 4, -1, 2, 1, -5, 4]
max_sum=0
result=[]
start,end=0,0
for i in range(0,len(nums)):
    for j in range(i+1,len(nums)):
        if max_sum<sum(nums[i:j+1]):
            max_sum=sum(nums[i:j+1])
            start,end=i,j+1
print(nums[start:end])
```

4. Rotate Array by K Steps

Description: Rotate the array to the right by k positions.

Example Input: nums = [1, 2, 3, 4, 5, 6, 7], k = 3

Example Output: [5, 6, 7, 1, 2, 3, 4]

---

code:
=====

```
"""Rotate the array to the right by k positions.
.
"""
nums=[1, 2, 3, 4, 5, 6, 7]
```

```
k=int(input("K:"))
while k!=0:
    nums=[nums[-1]]+nums[0:len(nums)-1]
    k-=1
print(nums)
```

5. Check if Palindrome String

Description: Check if a given string reads the same backward.

Example Input: "racecar"

Example Output: True

---

6. Missing Number in an Array

Description: Find the missing number from a sequence of integers.

Example Input: [3, 0, 1]

Example Output: 2

---

7. Merge Two Sorted Arrays

Description: Merge two sorted arrays into one sorted array.

Example Input: nums1 = [1, 3, 5], nums2 = [2, 4, 6]

Example Output: [1, 2, 3, 4, 5, 6]

---

8. Longest Substring Without Repeating Characters

Description: Find the length of the longest substring with unique characters.

Example Input: "abcabcbb"

Example Output: 3 (The substring is "abc")

---

9. Move Zeros to End

Description: Move all zeros in the array to the end while maintaining the order of non-zero elements.

Example Input: [0, 1, 0, 3, 12]

Example Output: [1, 3, 12, 0, 0]

---

10. Find Duplicates in Array

Description: Identify any duplicate elements in the array.

Example Input: [1, 3, 4, 2, 2]

Example Output: 2

---

11. Check if Array is Sorted

Description: Verify if the array is in non-decreasing order.

Example Input: [1, 2, 3, 4]

Example Output: True

---

12. Maximum contiguous Product Subarray

Description: Find the subarray with the maximum product.

Example Input: [2, 3, -2, 4]

Example Output: 6 (From subarray [2, 3])

---

13. Count Occurrences of an Element

Description: Count the occurrences of a specified element in the array.

Example Input: arr = [1, 2, 1, 3, 1], element = 1

Example Output: 3

---

14. First Non-Repeating Character in a String

Description: Find the first character that does not repeat in the string.

Example Input: "aabbccd"

Example Output: "d"

---

15. Find All Pairs with Sum Equal to K

Description: Find all pairs of elements that add up to a given sum.

Example Input: arr = [1, 5, 7, -1], k = 6

Example Output: [(1, 5), (7, -1)]

---

16. Remove Duplicates from a Sorted Array

Description: Remove duplicate values from a sorted array.

Example Input: [1, 1, 2, 2, 3]

Example Output: [1, 2, 3]

---

17. Longest Palindromic Substring

Description: Find the longest substring that is a palindrome.

Example Input: "babad"

Example Output: "bab" or "aba"

---

18. Check if Two Strings are Anagrams

Description: Check if two strings are anagrams of each other.

Example Input: "listen", "silent"

Example Output: True

---

19. Intersection of Two Arrays

Description: Find the intersection (common elements) of two arrays.

Example Input: arr1 = [1, 2, 2, 1], arr2 = [2, 2]

Example Output: [2, 2]

---

20. Union of Two Arrays

Description: Combine two arrays into a single array without duplicates.

Example Input: arr1 = [1, 2, 3], arr2 = [2, 3, 4]

Example Output: [1, 2, 3, 4]

---

21. Rotate a Matrix by 90 Degrees

Description: Rotate a 2D matrix by 90 degrees clockwise.

Example Input:

```
[[1, 2, 3],
 [4, 5, 6],
 [7, 8, 9]]
```

Example Output:

```
[[7, 4, 1],
 [8, 5, 2],
 [9, 6, 3]]
```

---

22. Kth Largest Element in an Array

Description: Find the kth largest element in an unsorted array.

Example Input: arr = [3, 2, 1, 5, 6, 4], k = 2

Example Output: 5 (The 2nd largest element)

---

23. Subarray with Given Sum

Description: Find a contiguous subarray that adds up to a specific sum.

Example Input: arr = [1, 2, 3, 7, 5], sum = 12

Example Output: [2, 3, 7]

---

24. Majority Element

Description: Find the element that appears more than n/2 times in the array.

Example Input: [3, 3, 4, 2, 4, 4, 2, 4, 4]

Example Output: 4

---

25. Implement strStr() (Find Substring)

Description: Find the index of the first occurrence of a substring in another string.

Example Input: haystack = "hello", needle = "ll"

Example Output: 2

---

26. Count Inversions in Array

Description: Count the number of inversions in the array (an inversion occurs when a larger element appears before a smaller one).

Example Input: [8, 4, 2, 1]

Example Output: 6

---

27. Longest Common Prefix

Description: Find the longest common prefix shared among all strings in an array.

Example Input: ["flower", "flow", "flight"]

Example Output: "fl"

---

28. Minimum in Rotated Sorted Array

Description: Find the smallest element in a rotated sorted array.

Example Input: [4, 5, 6, 7, 0, 1, 2]

Example Output: 0

---

29. Find All Subarrays with Sum 0

Description: Identify all subarrays in the array whose elements sum to 0.

Example Input: [6, -1, -3, 4, -2, 2, 4, 6, -12, -7]

Example Output: [(2, 4), (0, 10)]

---

30. Rearrange Positive and Negative Numbers

Description: Rearrange the array so that positive and negative numbers alternate.

Example Input: [12, -7, -5, 6, -3, 5]

Example Output: [12, -7, 6, -5, 5, -3]

---

31. Find Duplicates with Limited Range

Description: Identify duplicate elements when values are within a known range.

Example Input: [4, 3, 2, 7, 8, 2, 3, 1]

Example Output: [2, 3]

---

32. Valid Parentheses

Description: Check if a string containing parentheses, braces, and brackets is valid.

Example Input: "()[]{}"

Example Output: True

---

33. Longest Consecutive Sequence

Description: Find the length of the longest sequence of consecutive numbers.

Example Input: [100, 4, 200, 1, 3, 2]

Example Output: 4 (Sequence: [1, 2, 3, 4])

---

34. Find Pivot Index

Description: Find the index where the sum of numbers on the left is equal to the sum on the right.

Example Input: [1, 7, 3, 6, 5, 6]

Example Output: 3

---

35. Group Anagrams

Description: Group strings that are anagrams of each other.

Example Input: ["eat", "tea", "tan", "ate", "nat", "bat"]

Example Output: [["eat", "tea", "ate"], ["tan", "nat"], ["bat"]]

---

36. Add Two Numbers Represented by Lists

Description: Add two numbers represented as linked lists.

Example Input: l1 = [2, 4, 3], l2 = [5, 6, 4]

Example Output: [7, 0, 8] (Sum is 807)

---

37. Find Missing Ranges

Description: Find the missing ranges in an array given a lower and upper bound.

Example Input: nums = [0, 1, 3, 50, 75], lower = 0, upper = 99

Example Output: ["2", "4->49", "51->74", "76->99"]

---

38. Find Median of Two Sorted Arrays

Description: Find the median of two sorted arrays.

Example Input: nums1 = [1, 3], nums2 = [2]

Example Output: 2.0

---

39. Product of Array Except Self

Description: Calculate the product of all elements except the current one for each element.

Example Input: [1, 2, 3, 4]

Example Output: [24, 12, 8, 6]

---

40. Next Permutation

Description: Find the next lexicographically greater permutation of the array.

Example Input: [1, 2, 3]

Example Output: [1, 3, 2]

---

41. Trapping Rain Water

Description: Calculate the amount of rainwater that can be trapped between bars of different heights.

Example Input: [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]

Example Output: 6

---

42. Valid Palindrome II

Description: Check if a string can become a palindrome by removing at most one character.

Example Input: "abca"

Example Output: True

---

43. Merge Intervals

Description: Merge overlapping intervals.

Example Input: [[1, 3], [2, 6], [8, 10], [15, 18]]

Example Output: [[1, 6], [8, 10], [15, 18]]

---

44. Monotonic Array

Description: Check if the array is either entirely non-increasing or non-decreasing.

Example Input: [1, 2, 2, 3]

Example Output: True

---

45. Find Peak Element

Description: Find a peak element in the array (an element greater than its neighbors).

Example Input: [1, 2, 3, 1]

Example Output: 2

---

46. Distribute Candies

Description: Distribute candies to children such that each child with a higher rating gets more candy.

Example Input: [1, 0, 2]

Example Output: 5

---

47. Find All Subsets

Description: Find all subsets of an array.

Example Input: [1, 2, 3]

Example Output: [[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]]

---

48. Remove Element

Description: Remove all occurrences of a given value from the array.

Example Input: nums = [3, 2, 2, 3], val = 3

Example Output: [2, 2]

---

49. Minimum Window Substring

Description: Find the minimum window in a string containing all characters of another string.

Example Input: s = "ADOBECODEBANC", t = "ABC"

Example Output: "BANC"

---

50. Find K Closest Elements

Description: Find the k closest elements to a given value.

Example Input: arr = [1, 2, 3, 4, 5], k = 4, x = 3

Example Output: [1, 2, 3, 4]

python dictionary:
==============

python dictionary is used to store the in the form of

key and value pairs

in python dictionary,

1) keys are always unique or distinct

2) values are may be duplicate also

3) we can have any number key and value pairs

4)if key is string type, we need to keep in double quotes

5)if value is string type, we need to keep in double quotes

6)we can have key as any type or value as any type

7)we can not have any indexing like list or tuple, but keys

are act as indexes in dictionary

to create the dictionary, we will use "{ }"

syntax for dictionary in python is:
==========================

dictionary_name={key1:value,key2:value,......keyn:value}

example:
=======

```
"""working with dictionary"""
d1={1:2,3:4,5:6,7:8}
print(d1)
d2={"name":"ram","location":"hyderabad","email":"ram@gmail.com"}
print(d2)
d3={1:2,1:3,1:4}
print(d3)
d4={1.2:3.4,4.5:6.5,7.8:9.0}
print(d4)
```

in python , dictionary is mutable , because on dictionary we

can apply the following operations:
==================================================

1.insertion:
=========

in python when we want to add any element, we will use

following syntax:
==================================================

dictionary_name[key_name]=value

example:
=======

```
"""working with dictionary"""
d1={1:2,3:4}
print(d1)
d1[9]=400
print(d1)
d1[5]=600
print(d1)
d1[7]=800
print(d1)
d1[3]=1000
print(d1)
```

2.deletion:
========

if we want to delete the any key and value pair from the

dictionary, in python we will use the following fucntions:
==================================================

1.pop():
======

example:
=======

```
"""working with dictionary"""
d1={1:2,3:4,5:6,7:8,9:10,100:110}
print(d1)
d1.pop(5)#here 5 is key
print(d1)
d1.pop(9)
print(d1)
d1.pop(1)
print(d1)
```

note:
====

when we are working with pop() function, pop() will always

take key as argument

when we are working with pop() function, if we give key which

is not there in the dictionary, pop() will raises an error called

"Key Error"

2.popitem():
=========

```
"""working with dictionary"""
d1={1:2,3:4,5:6,7:8,9:10,100:110}
print(d1)
d1.popitem()
print(d1)
d1.popitem()
print(d1)
d1.popitem()
print(d1)
d1.popitem()
print(d1)
```

note:
====

it will always removes the "last key and value pair from the

dictionary"

3.clear():
=======

it will remove the "All key and value pairs from the dictionary"

syntax:
======

dictionary_name.clear()

4.using del keyword:
================

using del keyword also we can delete the key and value pair

from the dictionary

del dictionary_name[key_name]

example:
=======

```
"""working with dictionary"""
d1={1:2,3:4,5:6,7:8,9:10,100:110}
print(d1)
d1.clear()
print(d1)
d1={1:2,3:4,5:6,7:8,9:10,100:110}
print(d1)
del d1[1]
print(d1)
del d1[5]
print(d1)
```

3.updation:
==========

when we want to update the dictionary, in python we will use

the following ways:
================================================

1. using key value:
================

example:
=======

```
"""working with dictionary"""
d1={1:2,3:4,5:6,7:8,9:10,100:110}
print(d1)
d1[1]=100
print(d1)
d1[5]=200
print(d1)
d1[9]=1000
print(d1)
```

2.using update() function:
====================

example:
=======

```
"""using update() function"""
d1={1:2,3:4,5:6,7:8,9:10,100:110}
d1.update({1:100})
print(d1)
d1.update({3:300,5:500})
```

print(d1)

on dictionary, we can apply the following functions:
========================================

1.len() ===>to find the number of key and value pairs in the

dictionary

2.values()===>to get the only values from the dictionary

3.keys()====>used to get the only keys from the dictionary

4.items()==>used to get the both keys and values from

the dictionary

5.copy()====>it used to copy the all keys and values from

one dictionary to another dictionary

6.get()====>it used to the values from the dictionary based

on the given key

7.fromkeys() ==>it used to create the dictionary based on the

given keys and  values

8.setdefault()===>it used to set the value for a  key which is not

found, it simply create key and

value pair in the dictionary , if the key is

there it will not do any changes

example:
=======

```
"""working with dictionary"""
d1={1:2,3:4,5:6,7:8,9:10,100:110}
print(d1)
"""apply the len() fucntion"""
print(len(d1))
"""apply the keys() function"""
print(d1.keys())
"""apply the values() function"""
print(d1.values())
"""apply the items() function"""
print(d1.items())
```

```
"""apply the copy() function"""
d2=d1.copy()
print(d2)
"""apply the get() function"""
print(d2.get(100))#110
```

example-2:
=========

```
"""working with dictionary"""
d1={1:2,3:4,5:6,7:8,9:10,100:110}
print(d1)
d2={}
d2=d2.fromkeys(("a","b","c"),100)
print(d2)
d1.setdefault(30,1000)
print(d1)
d1.setdefault(50,5000)
print(d1)
d1.setdefault(5,590)
print(d1)
```

in python , we can create nested dictionaries, it means

in dictionary, we can create another dictionary

example:
=======

```
"""working with dictionary"""
d1={1:{2:3,4:5,6:7},3:{4:5,6:7,8:9}}
print(d1)
print(d1[1])
print(d1[3])
print(d1[1][4])
print(d1[3][6])
```

matrix data structure:
==================

matrix means ===>collection of "rows and columns"

in python, in order to represent the matrix we will use

list comprehension

create the 2*2 matrix
=================

```
     0      1 <=== column number starts from 0 to n-1
0    1      2

1    3      4

^
```

ll

row number starts from 0 to n-1

l1=[[1,2],[3,4]] <=== 2*2 matrix

create the two 2*2 matrix using list comprehension:
=========================================

```
"""working with matrix"""
l1=[[j for j in range(i,i+2)] for i in range(1,3)]
print(l1)
```

create the 3*3 matrix using list comprehension:
======================================

example:
=======

l1=[[1,2,3],[4,5,6],[7,8,9]]

code:
====

```
"""working with matrix"""
l1=[[col for col in range(rownum,rownum+3)] for rownum in range(1,4)]
print(l1)
```

create the matrix which are having same rows and columns:
===============================================

```
"""working with matrix"""
rows=int(input("rows:"))
cols=int(input("cols:"))
"""create the matrix using given rows and columns"""
if rows==cols:
    matrix=[[col+1 for col in range(rownum,rownum+3)] for rownum in range(0,rows)]
print(matrix)
```

matrix operations:
==============

1.matrix addition of 2*2 matrix:
==========================

```
"""working with matrix addition"""
m1=[[1,2],[3,4]]
m2=[[5,6],[7,8]]
sum=[]
result=[]
for i in range(0,len(m1)):
    for j in range(0,len(m1)):
```

```
        sum+=[m1[i][j]+m2[i][j]]
    result.append(sum)
    sum=[]
print(result)
```

2.matrix addition of 3*3 matrix:
=========================

```
"""working with matrix addition"""
m1=[[1,2,3],[3,4,5],[6,7,8]]
m2=[[5,6,7],[7,8,9],[10,11,12]]
sum=[]
result=[]
for i in range(0,len(m1)):
    for j in range(0,len(m1)):
        sum+=[m1[i][j]+m2[i][j]]
    result.append(sum)
    sum=[]
print(result)
```

3. 2*2 matrix subtraction:
=====================

```
"""working with matrix subtraction"""
m1=[[1,2],[3,4]]
m2=[[5,6],[7,8]]
sub=[]
result=[]
for i in range(0,len(m1)):
    for j in range(0,len(m1)):
        sub+=[m2[i][j]-m1[i][j]]
    result.append(sub)
    sub=[]
print(result)
```

4.3*3 matrix subtraction:
======================

```
"""working with matrix subtraction"""
m1=[[1,2,3],[3,4,5],[6,7,8]]
m2=[[5,6,7],[7,8,9],[10,11,12]]
sub=[]
result=[]
for i in range(0,len(m1)):
    for j in range(0,len(m1)):
        sub+=[m2[i][j]-m1[i][j]]
    result.append(sub)
    sub=[]
print(result)
```

5.display the 2*2 matrix diagonal elements:
==================================

```
"""working with matrix subtraction"""
m1=[[1,2],[3,4]]
result=[]
"""display the 2*2 diagnaol elements"""
for i in range(0,len(m1)):
    result+=[m1[i][i]]
print(result)
```

6.display the 3*3 matrix diagonal elements:
==================================

```
"""working with matrix """
m1=[[1,2,3],[3,4,5],[7,8,9]]
result=[]
"""display the 3*3 diagnaol elements"""
for i in range(0,len(m1)):
    result+=[m1[i][i]]
print(result)
```

7.display the diagonal elements sum, product of given 2*2 matrix:
===============================================

```
"""working with matrix """
m1=[[1,2],[3,4]]
sum=0
product=1
"""display the 2*2 diagnaol elements"""
for i in range(0,len(m1)):
    sum+=m1[i][i]
    product*=m1[i][i]
print(sum)
print(product)
```

7.display the diagonal elements sum, product of given 3*3 matrix:
===============================================

```
"""working with matrix """
m1=[[1,2,3],[3,4,5],[6,7,8]]
sum=0
product=1
"""display the 3*3 diagnaol elements"""
for i in range(0,len(m1)):
    sum+=m1[i][i]
    product*=m1[i][i]
print(sum)
print(product)
```

8.transpose of the 2*2 matrix:
========================

code:
====

```
m1=[[1,2],[3,4]]
sum=[]
result=[]
"""display the 2*2 transpose"""
for i in range(0,len(m1)):
  for j in range(0,len(m1)):
      sum+=[m1[j][i]]
  result+=[sum]
  sum=[]
print(result)
```

9.transpose of the 3*3 matrix:
========================

```
m1=[[1,2,3],[3,4,5],[5,6,7]]
sum=[]
result=[]
"""display the 3*3 transpose"""
for i in range(0,len(m1)):
  for j in range(0,len(m1)):
      sum+=[m1[j][i]]
  result+=[sum]
  sum=[]
print(result)
```

multiplication:
===========

multiplication of 2*2 matrix:
======================

```
A=[[1,2],[3,4]]
B=[[2,5],[6,8]]
result=[[0,0],[0,0]]
sum=0
for i in range(len(A)):
    for j in range(len(B)):
        for k in range(len(result)):
            result[i][j]+=A[i][k]*B[k][j]
print(result)
```

multiplication of 3*3 matrix:
======================

code:
====

```
A=[[1,2,3],[3,4,5],[1,2,3]]
B=[[2,5,6],[6,8,10],[2,3,4]]
result=[[0,0,0],[0,0,0],[0,0,0]]
sum=0
for i in range(len(A)):
    for j in range(len(B)):
        for k in range(len(result)):
            result[i][j]+=A[i][k]*B[k][j]
```

print(result)

determinant:
==========

2*2 matrix determinant:
===================

code:
====

```
A=[[1,2],[3,4]]
result=A[0][0]*A[1][1]-A[0][1]*A[1][0]
print(result)
A=[[2,5],[6,8]]
result=A[0][0]*A[1][1]-A[0][1]*A[1][0]
print(result)
```

3*3 determinant:
=============

code:
====

```
import numpy as np
a=[[10,2,30],[4,5,6],[17,8,9]]
minor1=a[0][0]*(a[1][1]*a[2][2]-a[1][2]*a[2][1])
minor2=a[0][1]*(a[1][0]*a[2][2]-a[1][2]*a[2][0])
minor3=a[0][2]*(a[1][0]*a[2][1]-a[1][1]*a[2][0])
print(minor1-minor2+minor3)
```

inverse:
======

2*2 inverse:
=========

code:
=====

```
a=[[1,2],[3,4]]
det_a=a[0][0]*a[1][1]-a[1][0]*a[0][1]
a=[[4/det_a,-2/det_a],[-3/det_a,1/det_a]]
print(a)
```

how to create the 2*2 matrix dynamically:
=================================

code:
====

```
rows=int(input("rows:"))
col=int(input("col:"))
```

```
matrix=[[0,0],[0,0]]
for i in range(rows):
    for j in range(col):
        matrix[i][j]=int(input(f"enter the value for matrix[{i}][{j}]:"))
print(matrix)
```

how to create the 3*3 matrix dynamically:
==================================

code:
====

```
rows=int(input("rows:"))
col=int(input("col:"))
matrix=[[0,0,0],[0,0,0],[0,0,0]]
for i in range(rows):
    for j in range(col):
        matrix[i][j]=int(input(f"enter the value for matrix[{i}][{j}]:"))
print(matrix)
```

or

```
rows=int(input("rows:"))
col=int(input("col:"))
matrix=[]
for i in range(rows):
    for j in range(col):
        matrix[i][j]=0
print(matrix)
for i in range(rows):
    for j in range(col):
        matrix[i][j]=int(input(f"enter the value for matrix[{i}][{j}]:"))
print(matrix)
```

python numpy and pandas :
=====================

working with numpy:
=================

in order to work with NumPy, we need to install  NumPy and

import the NumPy in every program.

how to import the numpy, in python program:
=====================================

to import the numpy, in python program, we will use the

following syntax:
==================================================

```
import numpy as np
```

in python, we are using NumPy to perform data manipulation or

in NumPy we will work any dimensional array

in order to create the array in NumPy, we will use following

syntax:
=====================================================

array_name=np.array()

note:

where the array what we create in python NumPy, is called as

"nd-array"

example-1:
=========

```
import numpy as np
"""create the nd-array in numpy"""
a1=np.array([1,2,3,4,5,6,7,8,9,10])
print(a1)
print(type(a1))
"""to know the dimention of the nd-array, we will
use "ndim" """
print(f"the dimention of the a1 is:{a1.ndim}")
a1=np.array([[1,2,3,4,5,6,7,8,9,10]])
print(a1)
print(type(a1))
"""to know the dimention of the nd-array, we will
use "ndim" """
print(f"the dimention of the a1 is:{a1.ndim}")
a1=np.array([[[1,2,3,4,5,6,7,8,9,10]]])
print(a1)
print(type(a1))
"""to know the dimention of the nd-array, we will
use "ndim" """
print(f"the dimention of the a1 is:{a1.ndim}")
```

example-2:
=========

```
import numpy as np
"""create the nd-array in numpy"""
a1=np.array([[[[[[[]]]]]]])
"""to know the dimention of the nd-array, we will
use "ndim" """
print(f"the dimention of the a1 is:{a1.ndim}")
a1=np.array([[],[],[],[],[],[]])
"""to know the dimention of the nd-array, we will
```

use "ndim" """
print(f"the dimention of the a1 is:{a1.ndim}")
a1=np.array([[[[],[],[]]]])
"""to know the dimention of the nd-array, we will
use "ndim" """
print(f"the dimention of the a1 is:{a1.ndim}")

note:
====

in order to find the dimension of the nd-array, we will use

"ndim" in python NumPy

to know the data type of the nd-array in NumPy, we will use

"dtype"

example:
=======

```
import numpy as np
"""create the nd-array in numpy"""
a1=np.array([1,2,3,4,5,6,7,8,9,10])
"""data type of the a1 nd array"""
print(a1.dtype)
a1=np.array([1.3,4.5,6.7,8.9,10.0])
"""data type of the a1 nd array"""
print(a1.dtype)
a1=np.array([’a’,’b’,’c’,’d’,’e’])
"""data type of the a1 nd array"""
print(a1.dtype)
a1=np.array([True,False,True,False])
print(a1.dtype)
a1=np.array([1,2,3,1.34,5.67,8.90])
print(a1.dtype)
a1=np.array([1,2,3,1.34,5.67,8.90,"hello"])
print(a1.dtype)
```

to know the number of elements in the nd-array, we will use

"size"

example:
=======

```
import numpy as np
"""create the nd-array in numpy"""
a1=np.array([1,2,3,4,5,6,7,8,9,10])
"""to know the number of elements in the nd-array
,we will use size
"""
print(a1.size)#10
```

```
print(len(a1))#10
a1=np.array([[1,2,3,4,5],[6,7,8,9,10]])
print(a1.size)#10
print(len(a1))#2
a1=np.array([[[1,2,3],[4,5,6],[7,8,9]]])
print(a1.size)#9
print(len(a1))#1
print(len(a1[0]))#3
```

to know the number of rows and columns of the nd-array, we

will use "shape"

example:
=======

```
import numpy as np
"""create the nd-array in numpy"""
a1=np.array([1,2,3,4,5,6,7,8,9,10])
"""to know the number of rows and columns, we will
use shape
"""
print(a1.shape)
a1=np.array([[1,2,3],[4,5,6]])
print(a1.shape)
a1=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a1.shape)
a1=np.array([[[1,2,3]]])
print(a1.shape)#(1,1,3)
a1=np.array([[[[1,2,3,4]]]])
print(a1.shape)#(1,1,1,4)
```

to convert the any nd-array data type, in numpy we will use

astype() function

example-1:
=========

```
import numpy as np
"""create the nd-array in numpy"""
a1=np.array([1,2,3,4,5,6,7,8,9,10])
"""to change the data type of the nd-array, we will
use astype()
"""
#a1 into float type
a1=a1.astype(float)
print(a1)
a1=np.array([1,2,3,0,5,6,7,8,9,0])
#a1 into boolean type
a1=a1.astype(bool)
print(a1)
#a1 into string type
a1=np.array([1,2,3,0,5,6,7,8,9,0])
a1=a1.astype(str)
```

print(a1)

to perform the element wise operations on the nd-arrays, we

will do the following way:
==================================================

element wise sum

element wise subtraction

element wis multiplication

above all we can perform , when the two ND-arrays are in same

shape

element wise operations on the 1-d array:
==================================

code:
====

```
import numpy as np
#create the 1-D ND-array
a1=np.array([1,2,3,4,5,6,7,8,9,10])
a2=np.array([10,20,30,40,50,60,70,80,90,100])
print(a1+a2)#element  wise sum
print(a2-a1)#element wise subtraction
print(a1*a2)#elment wise product
```

element wise operations on the 2D-array:
=================================

code:
=====

```
import numpy as np
#create the 1-D ND-array
a1=np.array([[1,2],[4,5]])
a2=np.array([[10,20],[30,40]])
print(a1.shape)
print(a2.shape)
print(a1+a2)#element  wise sum
print(a2-a1)#element wise subtraction
print(a1*a2)#element wise product
```

element wise operations on the 3D-array:
=================================

code:
====

```
import numpy as np
#create the 1-D ND-array
a1=np.array([[[1,2],[4,5]]])
```

a2=np.array([[[10,20],[30,40]]])
print(a1.shape)#(1,2,2)
print(a2.shape)#(1,2,2)
print(a1+a2)#element  wise sum
print(a2-a1)#element wise subtraction
print(a1*a2)#element wise product

indexing and slicing on the nd-arrays:
==============================

in python, even the numpy nd-arrays will follow the same

indexing and slicing like python lists only

indexing and slicing on 1D-arrays:
============================

code:
====

```
import numpy as np
#create the 1-D ND-array
a1=np.array([1,2,3,4,5,6,7,8,9,10])
print(a1[0])
print(a1[9])
print(a1[-5])
print(a1[-2])
print(a1[1:])
print(a1[1:7:2])
```

indexing and slicing on 2D-arrays:
============================

code:
====

```
import numpy as np
#create the 2-D ND-array
a1=np.array([[1,2,3],[4,5,6]])
print(a1[0])#
print(a1[1])#
print(a1[1][2])#6
print(a1[0][2])#3
print(a1[0][1:])#[2,3]
print(a1[1][::-1])#[6,5,4]
```

indexing and slicing on 3d array:
===========================

code:
====

```
import numpy as np
#create the 3-D ND-array
```

```
a1=np.array([[[1,2,3],[4,5,6]]])
print(a1[0])#[[1,2,3],[4,5,6]]
print(a1[0][0])
print(a1[0][1])
print(a1[0][1][2])#6
print(a1[0][1][1])#5
print(a1[0][::-1])
print(a1[0][0][::-1])
```

how to flatten the nd-array :
======================

flatten the nd-array means "make the any dimention nd-array

into single dimention nd-array"

[[]] ===>flattening ===>[]

[[[[[]]]]] ===>flattening ===>[]

to flatten the nd-array in python, we will use "flatten()" fucntion

example:
=======

```
import numpy as np
#create the 3-D ND-array
a1=np.array([[[1,2,3],[4,5,6]]])
#flattening the a1 into single 1-d array
a1=a1.flatten()
print(a1)
#create the 6-D ND-array
a1=np.array([[[[[[1,2,3],[4,5,6]]]]]])
#flattening the a1 into single 1-d array
a1=a1.flatten()
print(a1)
```

in numpy, we can also use a function called "ravel()", to flatten

the nd-array

example:
=======

```
import numpy as np
#create the 3-D ND-array
a1=np.array([[[1,2,3],[4,5,6]]])
#flattening the a1 into single 1-d array
a1=a1.ravel()
print(a1)
#create the 6-D ND-array
a1=np.array([[[[[[1,2,3],[4,5,6]]]]]])
#flattening the a1 into single 1-d array
```

a1=a1.ravel()
print(a1)

how to split the nd-arrays into sub nd-arrays in  numpy:
===========================================

to split the nd-arrays into sub nd-arrays in  numpy, we will use

split() function

syntax:
======

np.split(array_name, size)

example:
=======

```
import numpy as np
#create the 1-D ND-array
a1=np.array([1,2,3,4,5,6,7,8,9,10])
#split the a1 into 2 parts
a2=np.split(a1,2)
print(a1)
#split the a1 into 3 parts
a1=np.array([1,2,3,4,5,6,7,8,9,10,11,12])
a1=np.split(a1,3)
print(a1)
#split the a1 into 2 parts
a1=np.array([1,2,3,4,5,6,7,8,9,10,11,12])
a1=np.split(a1,2)
print(a1)
#split the a1 into 4 parts
a1=np.array([1,2,3,4,5,6,7,8,9,10,11,12])
a1=np.split(a1,4)
print(a1)
```

how to re-shape the nd-array in numpy:
===============================

to re-shape the nd-array in numpy, we will use a function called

reshape()

syntax:
======

reshape(rows, columns)

example:
=======

```
import numpy as np
#create the 1-D ND-array
```

```
a1=np.array([1,2,3,4])
print(a1)
print(a1.shape)
#re-shape the a1 into (2,2)
a1=a1.reshape(2, 2)
print(a1)
print(a1.shape)
#create the 1-D ND-array
a1=np.array([1,2,3,4,5,6,7,8,9])
print(a1)
print(a1.shape)
#re-shape the 1-d array into (3,3)
a1=a1.reshape(3,3)
print(a1)
print(a1.shape)
```

example-2:
=========

```
import numpy as np
#create the 2-D ND-array
a1=np.array([[1,2,3,4]])
print(a1)
print(a1.shape)
#re-shape the a1 into (2,2)
a1=a1.reshape(2,2)
print(a1)
print(a1.shape)
#create the 3-D ND-array
a1=np.array([[[1,2,3,4,5,6]]])
print(a1)
print(a1.shape)
#re-shape the a1 into (2,3)
a1=a1.reshape(2,3)
print(a1)
print(a1.shape)
```

example-3:
=========

```
import numpy as np
#create the 2-D ND-array
a1=np.array([[1,2,3,4]])
print(a1)
print(a1.shape)
#re-shape the a1 into (4,)
a1=a1.reshape(4,)
print(a1)
print(a1.shape)
#create the 5-D ND-array
a1=np.array([[[[[1,2,3,4]]]]])
print(a1)
print(a1.shape)
#re-shape the a1 into (4,)
a1=a1.reshape(a1.size,) #flattening
```

```
print(a1)
```

numpy arrays filtering:
==================

to apply the filtering on the numpy nd-array, we will use

the following syntax:
==================================================

array_name[condition]

working  with 1D-nd arrays:
=======================

code:
====

```
"""working with numpy nd-arrays """
import numpy as np
a1=np.array([1,2,3,4,5,6,7,8,9,10])
print(a1[a1>5])
print(a1[a1<5])
print(a1[a1%2==0])
print(a1[a1%2!=0])
```

working  with 2D-nd arrays:
=======================

code:
====

```
"""working with numpy nd-arrays """
import numpy as np
a1=np.array([[1,2,3,4,5],[6,7,8,9,10]])
print(a1[a1>5])
print(a1[a1<5])
print(a1[a1%2==0])
print(a1[a1%2!=0])
```

working  with 3D-nd arrays:
=======================

code:
====

```
import numpy as np
a1=np.array([[[1,2,3,4,5]],[[6,7,8,9,10]]])
print(a1[a1>5])
print(a1[a1<5])
print(a1[a1%2==0])
print(a1[a1%2!=0])
```

matrix operations using numpy arrays:
==============================

create the matrix using ones() and zeros() function

in numpy:
=================================================

create the matrix using ones() function:
===============================

code:
====

```
"""create the matrix using ones()
function """
import numpy as np
a1=np.ones((2,2))
print(a1)
a1=np.ones((3,3))
print(a1)
a1=np.ones((1,3,3))
print(a1)
```

create the matrix using zeros() function:
===============================

code:
====

```
"""create the matrix using zeros()
function """
import numpy as np
a1=np.zeros((2,2))
print(a1)
a1=np.zeros((3,3))
print(a1)
a1=np.zeros((1,3,3))
print(a1)
```

how to create the identity matrix in python numpy:
========================================

to create the identity matrix in python numpy, we will use

a function called "eye()"

example-1:
=========

```
"""create the  identity matrix using
```

eye()  function """
import numpy as np
a1=np.eye(2)
print(a1)
a1=np.eye(3)
print(a1)
a1=np.eye(2,dtype=int)
print(a1)
a1=np.eye(3,dtype=int)
print(a1)

matrix addition in python numpy:
==========================

example:
=======

"""matrix addition in python numpy"""
import numpy as np
a1=np.array([[1,2],[3,4]])
a2=np.array([[5,6],[7,8]])
print(a1+a2)#matrix addtiion
a1=np.array([[1,2,3],[4,5,6],[7,8,9]])
a2=np.array([[5,6,10],[7,8,9],[11,12,13]])
print(a1+a2)#matrix addtiion

matrix subtraction in python numpy:
============================

example:
=======

"""matrix subtraction in python numpy"""
import numpy as np
a1=np.array([[1,2],[3,4]])
a2=np.array([[5,6],[7,8]])
print(a1-a2)#matrix subtraction
a1=np.array([[1,2,3],[4,5,6],[7,8,9]])
a2=np.array([[5,6,10],[7,8,9],[11,12,13]])
print(a1-a2)#matrix subtraction

matrix multiplication in numpy:
=========================

in numpy to apply the matrix multiplication, we will use a

function called "dot()"

example:
=======
"""matrix multiplication in python numpy"""
import numpy as np

a1=np.array([[1,2],[3,4]])
a2=np.array([[5,6],[7,8]])
print(np.dot(a1,a2))#matrix multiplication
a1=np.array([[1,2,3],[4,5,6],[7,8,9]])
a2=np.array([[5,6,10],[7,8,9],[11,12,13]])
print(np.dot(a1,a2))#matrix multiplication

in numpy, we can also use "@" symbol to perform the

matrix multiplication

example:
=======

```
"""matrix multiplication in python numpy"""
import numpy as np
a1=np.array([[1,2],[3,4]])
a2=np.array([[5,6],[7,8]])
print(a1@a2)#matrix multiplication
a1=np.array([[1,2,3],[4,5,6],[7,8,9]])
a2=np.array([[5,6,10],[7,8,9],[11,12,13]])
print(a1@a2)#matrix multiplication
```

matrix transpose:
==============

to perform matrix transpose in python, we will use "T" in

numpy

syntax:
======

array_name.T

example:
=======

```
"""matrix Transpose in python numpy"""
import numpy as np
a1=np.array([[1,2],[3,4]])
print(a1.T)
a2=np.array([[5,6],[7,8]])
print(a2.T)
a1=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a1.T)
a2=np.array([[5,6,10],[7,8,9],[11,12,13]])
print(a2.T)
```

matrix determinant:
================

to find the matrix determinant in numpy, we will use following

syntax:
======

np.linalg.det(matrix_name)

example:
=======
```
"""matrix determinant in python numpy"""
import numpy as np
a1=np.array([[1,2],[3,4]])
print(np.linalg.det(a1) )
a2=np.array([[5,6],[7,8]])
print(np.linalg.det(a2) )
a1=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(np.linalg.det(a1) )
a2=np.array([[5,6,10],[7,8,9],[11,12,13]])
print(np.linalg.det(a2) )
```

matrix inverse:
============

to find the inverse of the given matrix in python, we will use

the following syntax:
================================================

np.linalg.inv(matrixname)

example:
=======

```
"""matrix inverse in python numpy"""
import numpy as np
a1=np.array([[1,2],[3,4]])
print(np.linalg.inv(a1) )
a2=np.array([[5,6],[7,8]])
print(np.linalg.inv(a2) )
a1=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(np.linalg.inv(a1) )
a2=np.array([[5,6,10],[7,8,9],[11,12,13]])
print(np.linalg.inv(a2) )
```

numpy statistical functions:
======================

in numpy we will have the following statistical fucntions:
============================================

sum():
=====

example:
=======
```
"""matrix  in python numpy"""
```

```
import numpy as np
a1=np.array([[1,2],[3,4]])
print(np.sum(a1))
print(np.sum(a1,axis=0))#column wise
print(np.sum(a1,axis=1))#row wise
print(np.sum(a1,axis=-2))#column wise
print(np.sum(a1,axis=-1))#row wise
```

min():
====

```
import numpy as np
a1=np.array([[1,2],[3,4]])
print(np.min(a1))#1
print(np.min(a1,axis=0))#column wise
print(np.min(a1,axis=1))#row wise
print(np.min(a1,axis=-2))#column wise
print(np.min(a1,axis=-1))#row wise
```

max():
=====

```
import numpy as np
a1=np.array([[1,2],[3,4]])
print(np.max(a1))#1
print(np.max(a1,axis=0))#column wise
print(np.max(a1,axis=1))#row wise
print(np.max(a1,axis=-2))#column wise
print(np.max(a1,axis=-1))#row wise
```

median():
=======

```
"""working with numpy"""
import numpy as np
"""create the nd-array"""
a1=np.array([1,2,3,4,5,6,7,8,9,10])
print(np.median(a1))
a1=np.array([[1,2,3,4,5],[6,7,8,9,10]])
print(np.median(a1))
print(np.median(a1,axis=0))
print(np.median(a1,axis=1))
a1=np.array([[[1,2,3],[4,5,6],[7,8,9]]])
print(np.median(a1))
print(np.median(a1,axis=0))
print(np.median(a1,axis=1))
```

average():
========

```
"""working with numpy"""
```

```
import numpy as np
"""create the nd-array"""
a1=np.array([1,2,3,4,5,6,7,8,9,10])
print(np.average(a1))
a1=np.array([[1,2,3,4,5],[6,7,8,9,10]])
print(np.average(a1))
print(np.average(a1,axis=0))#column wise
print(np.average(a1,axis=1))#row wise
```

std():
====

```
"""working with numpy standard deviation"""
import numpy as np
"""create the nd-array"""
a1=np.array([1,2,3,4,5,6,7,8,9,10])
print(np.std(a1))
a1=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(np.std(a1))
print(np.std(a1,axis=0))#column wise
print(np.std(a1,axis=1))#row wise
```

var():
====

```
"""working with numpy standard deviation"""
import numpy as np
"""create the nd-array"""
a1=np.array([1,2,3,4,5,6,7,8,9,10])
print(np.var(a1))
a1=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(np.var(a1))
print(np.var(a1,axis=0))#column wise
print(np.var(a1,axis=1))#row wise
```

other functions:
=============

1.where():
========

this function will return the result , based on the values which

are satisfies the given condition

example:
=======

```
"""working with numpy where()"""
import numpy as np
"""create the nd-array"""
a1=np.array([1,2,3,4,5,6,7,8,9,10])
```

```
print(np.where(a1>5))# we got indexes
print(np.where(a1>5,a1,0))
print(np.where(a1>5,a1,100))
```

2.all():
=====

```
"""working with numpy all()"""
import numpy as np
"""create the nd-array"""
a1=np.array([1,2,3,4,5,6,7,8,9,10])
print(np.all(a1))
a1=np.array([1,2,3,4,5,6,7,8,9,0])
print(np.all(a1))
a1=np.array([1,2,3,4,5,6,7,8,1.2,3.4,5.6])
print(np.all(a1))
```

3.any():
======

```
"""working with numpy any()"""
import numpy as np
"""create the nd-array"""
a1=np.array([1,2,3,4,5,6,7,8,9,10])
print(np.any(a1))
a1=np.array([1,2,3,4,5,6,7,8,9,0])
print(np.any(a1))
a1=np.array([0,0,0,0,0])
print(np.any(a1))
```

3.take():
=======

this function will take the elements only which are satisfies the

given condition

example:
=======

```
"""working with numpy any()"""
import numpy as np
"""create the nd-array"""
a1=np.array([1,2,3,4,5,6,7,8,9,10])
indexes=[0,1,2,6]
print(np.take(a1,indexes))
a1=np.array([1,2,3,4,5,6,7,8,9,10])
indexes=[6,7,8]
print(np.take(a1,indexes))
a1=np.array([1,2,3,4,5,6,7,8,9,10])
indexes=[9,4,5,6]
```

print(np.take(a1,indexes))

4.put():
=======

using this function, in the numpy array we can change only

particular index elements only

example:
=======

```
"""working with numpy any()"""
import numpy as np
"""create the nd-array"""
a1=np.array([1,2,3,4,5,6,7,8,9,10])
indexes=[0,1,2,6] #[1,2,3,7]
values=[10,20,30,40]
np.put(a1,indexes,values)
print(a1)
np.put(a1,[3,4],[400,500])
print(a1)
```

5.sort():
======

this used to sort the data

example:
=======

```
"""working with numpy sort()"""
import numpy as np
"""create the nd-array"""
a1=np.array([100,2,10,7,-67,7,8,9,10])
a1.sort()
print(a1)
a1=a1[::-1]
print(a1)
```

6.argsort():
=========

this will give the "indexes which are going to tell the elements

actual sorted order of the given numpy array"

example:
=======

```
"""working with numpy sort()"""
import numpy as np
"""create the nd-array"""
```

```
a1=np.array([100,2,10,7,-67,7,8,9,10])
result=np.argsort(a1)
print(result)
print(a1[result])
result=result[::-1]
print(a1[result])
```

7.argmax():
=========

it is going to return the maximum element index in the

given numpy array

example:
=======

```
"""working with numpy argmax()"""
import numpy as np
"""create the nd-array"""
a1=np.array([100,2,10,7,-67,7,8,9,10])
result=np.argmax(a1)
print(result)
print(a1[result])
```

8.argmin():
=========

it is going to return the minimum element index in the

given numpy array

example:
=======

```
"""working with numpy argmax()"""
import numpy as np
"""create the nd-array"""
a1=np.array([100,2,10,7,-67,7,8,9,10])
result=np.argmin(a1)
print(result)
print(a1[result])
```

9.arange():
=========

```
"""working with numpy arange()"""
import numpy as np
"""create the nd-array"""
a1=np.arange(1,10)
print(a1)
a1=np.arange(1,10,4)
```

print(a1)

10.hstack():
=========

```
"""working with numpy hstack()"""
import numpy as np
"""create the nd-array"""
a1=np.array([[5,6],[7,8]])
a2=np.array([[1,2],[3,4]])
print(np.hstack((a1,a2)))
a1=np.array([1,2,3,4])
a2=np.array([6,7,8,9])
print(np.hstack((a1,a2)))
```

11.vstack():
=========

combine the elements vertically of the give numpy arrays

example:
=======

```
"""working with numpy vstack()"""
import numpy as np
"""create the nd-array"""
a1=np.array([[5,6],[7,8]])
a2=np.array([[1,2],[3,4]])
print(np.vstack((a1,a2)))
a1=np.array([1,2,3,4])
a2=np.array([6,7,8,9])
print(np.vstack((a1,a2)))
```

12.add():
=======

example:
=======

```
"""working with numpy airthmetic operations"""
import numpy as np
"""create the nd-array"""
a1=np.array([[5,6],[7,8]])
a2=np.array([[1,2],[3,4]])
print(np.add(a1,a2))
print(np.subtract(a1,a2))
print(np.divide(a1,a2))
print(np.multiply(a1,a2))
print(np.sqrt(a1))
print(np.exp(a1))
```

12.subtract():
===========

```
"""working with numpy airthmetic operations"""
import numpy as np
"""create the nd-array"""
a1=np.array([[5,6],[7,8]])
a2=np.array([[1,2],[3,4]])
print(np.subtract(a1,a2))
```

13.multiply()
===========

example:
=======

```
"""working with numpy airthmetic operations"""
import numpy as np
"""create the nd-array"""
a1=np.array([[5,6],[7,8]])
a2=np.array([[1,2],[3,4]])
print(np.add(a1,a2))
print(np.multiply(a1,a2))
```

14.divide():
===========
example:
=======

```
"""working with numpy airthmetic operations"""
import numpy as np
"""create the nd-array"""
a1=np.array([[5,6],[7,8]])
a2=np.array([[1,2],[3,4]])
print(np.subtract(a1,a2))
```

15.sqrt()
=======

example:
=======

```
"""working with numpy airthmetic operations"""
import numpy as np
"""create the nd-array"""
a1=np.array([[5,6],[7,8]])
a2=np.array([[1,2],[3,4]])
print(np.sqrt(a1))
```

16.exp():

=======

example:
========
```
"""working with numpy airthmetic operations"""
import numpy as np
"""create the nd-array"""
a1=np.array([[5,6],[7,8]])
a2=np.array([[1,2],[3,4]])
print(np.exp(a1))
```

broadcasting on numpy arrays:
========================

1.when we arrays are same dimention and same share, we

can apply the broadcast

2.when one array dimension is "1", other is another dimension

lower dimension can be stretched

3.if the two NumPy arrays are in different shape, then

broadcast is not possible

example:
=======

```
"""working with numpy airthmetic operations"""
import numpy as np
"""create the nd-array"""
a1=np.array([[5,6],[7,8]])
a2=np.array([[1,2],[3,4]])
print(a1+10)
print(a2+100)
a1=np.array([1,2,3])
a2=np.array([[1],[2],[3]])
print(a1+a2)
a1=np.array([[1,2,3,4]])#(1,4) #2
a2=np.array([[[1,2,3,4],[5,6,7,8],[7,8,9,10]]])#(1,3,4) #3
print(a1+a2)
```

recursion:
========

1. write a python program to get the maximum element

from the given list using recursion:
===============================================

code:
=====
```
"""maximum element from the list using recursion"""
def maximum_element(list1,max):
```

```
    if len(list1)==1:
        return list1,list1[0] if max<list1[0] else max
    else:
        element=list1[0]
        if max<element:
            max=element
        return maximum_element(list1[1:],max)

list1=[7]
max=0
result1,result2=maximum_element(list1,max)
print(result2)
```

2. find the sum of the squares of the upto given number using recursion:
==============================================

input: 5

1+4+9+16+25=55

code:
====

```
"""maximum element from the list using recursion"""
def sum_of_the_squares(num):
    if num<=0:
        return 0
    else:
        return (num*num)+sum_of_the_squares(num-1)
num=int(input("number:"))
print(sum_of_the_squares(num))
```

3.print the maximum prime digit in the given number using

 recursion:
====================================================

code:
====

```
"""maximum prime digit from the list using recursion"""
def maximum_prime_digit(num,max):
    if len(num)==1:
        if num[0] in '2357' and max!=0:
            return num[0] if max<int(num[0]) else max
        elif num[0] not in '2357' and max!=0:
             return max
        else:
            return "no result"

    else:
         if num[0] in '2357':
              if max<int(num[0]):
                  max=int(num[0])
         return maximum_prime_digit(num[1:],max)
num=int(input("number:"))
```

print(maximum_prime_digit(str(num),0))

working with Pandas:
=================

when want to work with pandas, we need to install as follows:
================================================

pip install pandas

to check with pandas version, we will use following syntax:
=================================================

import pandas as pd
print(pd.__version__)

when we want to work with data in pandas, we will use the

following data structures:
================================================

1)Series:
=======

in pandas, series will maintain the data in a single -dimension

it means it like a "a single column in the table"

or

it means it like a "list in python"

in order to create the series in pandas, we will use a function

called "Series()"

syntax:
======
import pandas as pd
pd.Series(data)

example:
=======

```
"""create the series using list using pandas"""
s1=pd.Series([10,20,30,40,50,60,70,80,90,100])
print(s1)
s1=pd.Series([1.2,3.0,45.6,7.8,9.0])
print(s1)
s1=pd.Series(["hello","abcd","ghij"])
print(s1)
s1=pd.Series([1,2,3,1.2,3.4,5.6,7.8,9.0,"hello","hai"])
```

print(s1)

when we create the Series in the pandas using list, it will take

index from 0 to n-1

where "n" represents "number of values in the series"

example:
========

```
"""create the pandas Series using dictionary"""
s1=pd.Series({1:2,3:4,5:6,7:8,9:10})
print(s1)
s2=pd.Series({"a":1,"b":2,"c":3,"d":4,"e":5})
print(s2)
s3=pd.Series({1:"abc",2:"cba",3:"xyz"})
print(s3)
s4=pd.Series({1:1.234,2:2.345,3:456})
print(s4)
```

when we create the series using dictionary, in the series:

all keys of the dictionary act as "indexes" in pandas Series

all values of the dictionary act as "values" in the pandas

Series

the data type of the pandas series is always depends values

what we taken in the dictionary

example:
=======

```
"""create the pandas series using index attribute"""
s1=pd.Series([10,20,30,40,50,60,70,80,90,100],index=[1,2,3,4,5,6,7,8,9,10])
print(s1)
s1=pd.Series([10,20,30,40,50],index=[1.2,3.4,5.6,7.8,8.9])
print(s1)
s1=pd.Series([10,20,30,40,50],index=["a","b","c","d","e"])
print(s1)
s1=pd.Series([10,20,30,40,50],index=[1,2,3,"a","b"])
print(s1)
s1=pd.Series(100,[1,2,3,4,5,6,7,8,9,10])
print(s1)
```

example:
=======

```
"""create the pandas series using numpy array"""
import numpy as np
```

```
a1=np.array([10,20,30,40,50,60,70,80,90,100])
s1=pd.Series(a1)
print(s1)
```

in pandas, we can also create the series using numpy array

indexing and slicing with pandas Series:
================================

working indexing on pandas Series:
=============================

example:
=======

```
"""working with pandas series"""
s1=pd.Series([10,20,30,40,50,60,70,80,90,100])
print(s1[0])#10
print(s1[9])#100
print(s1[5])#60
print(s1[7])#40
"""
we can display all values of the pandas series using
for loop
"""
for i in range(len(s1)):
    print(f"s1[{i}]:{s1[i]}")
```

working with slicing on pandas Series:
===============================

example:
=======

```
"""working with pandas series slicing"""
s1=pd.Series([10,20,30,40,50,60,70,80,90,100])
print(s1[1:])
print(s1[3:])
print(s1[3:9])
s1=pd.Series([10,20,30,40,50],index=["a","b","c","d","e"])
print(s1)
print(s1["a":"c"])
```

working with pandas series attributes:
===============================

code:
====

```
"""create the pandas series"""
s1=pd.Series([10,20,30,40,50],index=[1,2,3,4,5],name="my_series")
"""know the name of the pandas seires"""
print(s1.name)
"""udpate the series name as follows in pandas"""
```

s1.name="myseries_new"
print(s1.name)
"""know the shape of the pandas series"""
print(s1.shape)#(5,)
"""know the indexes of the pandas series"""
print(s1.index)
"""know the values of the pandas series"""
print(s1.values)
"""check the pandas series having any duplicate or not"""
print(s1.is_unique)
"""to know the number of values in the pandas series"""
print(s1.size)

operations on pandas Series:
=======================

1.performing the arithmetic operations:
================================

example:
=======

"""perfoming the airthmetic operation on the pandas series"""
s1=pd.Series([10,20,30,40,50],index=[1,2,3,4,5],name="my_series")
s2=pd.Series([100,200,300,400,500],index=[1,2,3,4,5],name="my_series2")
"""addition of s1 and s2"""
print(s1+s2)
"""subtraction of s1 and s2"""
print(s1-s2)
"""multiplication of s1 and s2"""
print(s1*s2)
"""floor division of s1 and s2"""
print(s1//s2)
"""true division of s1 and s2"""
print(s1/s2)
"""modulo division of s1 and s2"""
print(s2%s1)

performing the aggregate operations on the pandas Series:
==============================================

example:
=======

"""aggregate operations on pandas series"""
s1=pd.Series([10,20,30,40,50])
"""sum of the all values of the pandas series s1 """
print(s1.sum())
"""maximum value of the pandas series"""
print(s1.max())
"""minimum value of the pandas series"""
print(s1.min())
"""mean value of the pandas series"""
print(s1.mean())

"""median value of the pandas series"""
print(s1.median())

performing the element-wise operations on pandas series:
==============================================

example:
=======

"""create the pandas series"""
s1=pd.Series([10,20,30,40,50])
print(s1**2)
print(s1**(0.5))
print(s1**(1/3))
print(s1.unique())
s1=pd.Series([10,10,10,10,101])
print(s1.unique())

filtering operations on the pandas series:
=================================

example:
=======

s1=pd.Series([100,200,300,400,500,600,700,800,900,1000])
print(s1[s1>100])
print(s1[s1<500])
print(s1[s1==500])
print(s1[s1<=300])
print(s1[s1!=500])

working with missing values in the pandas series:
=======================================

example:
=======

"""create the pandas series with None values
 here None is condiered as a no value or missing in pandas series
  NaN means "there is no value or no number or it is missing value"
 """
s1=pd.Series([10,20,30,40,None,60,70,None,80,90,None])
print(s1)
"""to check the missing values,we will use a fucntion isna()"""
print(s1.isna())
"""to remove the missing values, we will use a fucntion called dropna()"""
print(s1.dropna())
"""to fill the some value in the place missing,we will use a fucntion called
fillna(value) on pandas series"""
print(s1.fillna(10000))

working with apply() function:
=======================

in pandas, we will use "apply() function, to apply any function

on the pandas series"

example:
=======

```
s1=pd.Series([1,2,3,4,5,6,7,8,9,10])
print(s1.apply(lambda x:x**2))
print(s1.apply(lambda x:x if x>5 else 0))
def get_only_even(x):
    if x%2==0:
        return x
    else:
        return 0
print(s1.apply(get_only_even))
```

working with string operations on the pandas series:
==========================================

example:
=======

```
s1=pd.Series(["hello","abcd","pqrs","jklm"])
print(s1.str.upper())
print(s1.str.lower())
print(s1.str.len())
print(s1.str.swapcase())
```

2)DataFrame:
============

in pandas, dataframe is used to "create the data in the form

of tabular format or table"

data frame is data structure in pandas, to represent the data

in the form table

data frame can be indexed like series

data frame can be sliced like series

data frame can missing values like series

to create the data frame in pandas we will use a function

called "DatFrame()"

example-1:
=========

```
"""create the data frame using DataFrame() function of pandas"""
df1=pd.DataFrame([[1,2,3],[5,6,7],[9,10,11]],
                 columns=['col1','col2','col3'])
print(df1)
df2=pd.DataFrame([[1,2,3,4],[5,6,7,8],[9,10,11,12]],
                 columns=['col1','col2','col3','col4'])
print(df2)
```

example-2:
=========

create the data frame using dictionary of lists

code:
=====

```
data={"name":["abc","bca","cab","def"],
      "email":["abc@gmail.com","bca@gmail.com","cab@gmail.com","def@gmail.com"]
,"salary":[10000,20000,30000,40000]}
df=pd.DataFrame(data)
print(df)
```

example-3:
=========

create the data frame using list of dictionaries

code:
=====

```
data=[{"name":"abc","email":"abc@gmail.com","salary":20000},
     {"name":"abc","email":"abc@gmail.com","salary":20000},
     {"name":"abc","email":"abc@gmail.com","salary":20000},
     {"name":"abc","email":"abc@gmail.com","salary":20000},
     {"name":"abc","email":"abc@gmail.com","salary":20000}]
df=pd.DataFrame(data)
print(df)
```

example:
=======

create the data frame using dictionary of series

code:
=====

```
data={"col1":pd.Series([1,2,3,4,5]),
     "col2":pd.Series([1,2,3,4,5]),
     "col3":pd.Series([1,2,3,4,5]),
     "col4":pd.Series([1,2,3,4,5]),
     "col5":pd.Series([1,2,3,4,5])}
```

```
df=pd.DataFrame(data)
print(df)
```

example:
=======

create the data frame using zip() function

code:
=====

```
l1=[10,20,30,40,50]
l2=[10,20,30,40,50]
l3=[10,20,30,40,50]
l4=[10,20,30,40,50]
list1=list(zip(l1,l2,l3,l4))
print(list1)
df=pd.DataFrame(list1,columns=['col1','col2','col3','col4'])
print(df)
```

example:
=======

create the data frame using numpy array

code:
====

```
data=np.array([[1,2,3,4],[5,6,7,8],[10,11,12,13],[14,15,16,17]])
df=pd.DataFrame(data,columns=['col1','col2','col3','col4'])
print(df)
```

example:
=======

create the data frame using csv data

code:
====

```
"""read the csv data using read_csv() fucntion"""
df=pd.read_csv("mydata.csv")
print(df)
```

example:
=======

create data frame using json data

code:
====

```
"""read the json data using read_json() fucntion"""
df=pd.read_json("myjson.json")
```

print(df)

example:
========

create the data frame using excel

example:
=======

data=pd.read_excel("data.xlsx")
print(data)
data1=pd.read_excel("data2.xlsx")
print(data1)

working with dataframe:
===================

to know the basic info about the data frame, we will use the

following functions:
==================================================

info() <=== it is used to give the information about number

of columns, columns data types in the given

data frame

syntax: data_frame_name.info()

head() <=== it will give the rows from the starting the

data frame

by default it will give the first 5 rows as result

we can also mention the "number of rows

required" in the head() function

syntax: data_frame_name.head()

tail() <=== it will give the rows from the ending of the

data frame

by default it will give the last 5 rows as result

we can also mention the "number of rows

required" in the tail() function

syntax: data_frame_name.tail()

describe()===> it will give the statistical information about

number type columns present the given data

frame, basically it will give the information

about min value, max value, count value,

mean value, standard deviation, quartiles

syntax:  data_frame_name.describe()

working with data frame columns:
===========================

accessing the columns:
==================

to retrieve the columns from the data frame we will use the

following syntax:

data_frame_name[[col1name,col2_name,col3_name,.......]]

example:
=======

print(data[['id']])
print(data[['id','salary']])

accessing the data frame rows:
===========================

in order to access the data frame rows, we will use the

following functions:
===================================================

iloc()  ===>index-based selection

loc()   ===> label-based selection

working with iloc():
===============

example-1:
=========

print(data.iloc[0])#only first row
print(data.iloc[0:3])#only first 3 rows
print(data.iloc[:-1]) #except last, retrive the all rows

```
#first 3 rows of first 3 columns
print(data.iloc[0:3,0:3])
#all rows, except last 3 columns
print(data.iloc[:,:-3])
```

example-2:
=========

```
print(data.loc[0])#only first row
print(data.loc[0:3])#only first 3 rows
print(data.loc[0:5])
print(data.loc[0:4,['id','salary']])
print(data.loc[0:4,['id','salary','email']])
```

example-3:
=========

```
data={"name":["abc","bca","cab","def"],
      "email":["abc@gmail.com","bca@gmail.com","cab@gmail.com","def@gmail.com"]
,"salary":[10000,20000,30000,40000]}
df=pd.DataFrame(data,index=["row1","row2","row3","row4"])
print(df)
print(df.loc[["row1","row2","row3"]])
print(df.iloc[0:3])
```

adding the new columns into data frame:
================================

example-1:
========

```
print(df)
df['name2']=["abc","bca","cab","def"]
df['salary2']=[10000,20000,30000,40000]
print(df)
```

example-2:
=========

```
data={"name3":["abc","bca","cab","def"],
      "email3":["abc@gmail.com","bca@gmail.com","cab@gmail.com","def@gmail.com"]
,"salary3":[10000,20000,30000,40000]}
df=df.assign(**data)
print(df)
```

removing the columns from the data frame:
==================================

example:
=======

```
print(df)
df=df.drop(columns=['name3','email3','salary3'])
```

```
print(df)
df=df.drop(columns=["name2","salary2"])
print(df)
```

filtering the data from the data frame:
==============================

example:
=======

```
print(df)
print(df[df[’salary’]>10000])
print(df[df[’salary’]<40000])
print(df[df[’salary’]==10000])
```

how to write the data frame data into csv file in pandas:
============================================

here we converting pandas data frame again into csv file

in order to convert the data frame into csv file in pandas we

will use a function called "to_csv()"

example:
=======

```
data=pd.read_excel("data.xlsx")
print(data)
#here data is data frame, we convert into csv using "to_csv()"
data.to_csv("mydata.csv")
```

how to write the data frame data into excel file in pandas:
============================================

here we converting pandas data frame again into excel file

in order to convert the data frame into excel file in pandas we

will use a function called "to_excel()"

example:
=======

```
data=pd.read_csv("mydata.csv")
print(data)
data.to_excel("mydata.xlsx")
```

how to write the data frame data into json file in pandas:
============================================

here we converting pandas data frame again into json file in order to convert the data frame into json file in pandas we

will use a function called "to_json()"

example:
=======

```
data=pd.read_csv("mydata.csv")
print(data)
data.to_json("mydata.json")
```

working with NaN values or empty values in data frame:
==========================================

when we want to work with pandas data frame, to handle

the "NaN" values, in pandas module we will have the following

functions:

1.isna():
=======

using this function we can check the "NaN" values in the

Data frame

example:
=======

```
data=pd.read_excel("data2.xlsx")
print(data)
print(data.isna())
print(data.isna().sum())
```

2.dropna():
=========

using this function we can remove the NaN values either from

"row" or "column"

remove the NaN values from data frame row:
==================================

```
data=pd.read_excel("data2.xlsx")
print(data)
print(data.dropna())#row wise deletion based on NaN values
```

remove the NaN values from the Data Frame Column:
=========================================

```
data=pd.read_excel("data2.xlsx")
```

print(data)
print(data.dropna(axis=1))#column wise deletion based on NaN values

3.fillna()
=======

using this function we can fill the "NaN" values with some value

where columns which are having the "NaN" values

example:
=======

```
data=pd.read_excel("data2.xlsx")
print(data)
data['location']=data['location'].fillna("hyderabad")
print(data)
data['password']=data['password'].fillna("abc@123")
print(data)
data['salary']=data['salary'].fillna(10000)
print(data)
print(data.isna().sum())
```

other functions on the Pandas Data Frame:
=================================

1.group by:
==========

```
print(data.groupby('location').agg({'salary':['mean','max','min','sum']}))
print(data.groupby('location')['salary'].sum())
print(data.groupby('location')['salary'].max())
print(data.groupby('location')['salary'].min())
print(data.groupby('location')['salary'].mean())
print(data.groupby('location')['salary'].median())
```

2.sort_values():
===========

```
print(data.sort_values('salary'))
print(data.sort_values('salary',ascending=False))
print(data.sort_values('salary',ascending=True))
```

3.concat():
========

```
data2=pd.read_excel("data.xlsx")
print(pd.concat([data,data2]))
print(pd.concat([data,data2],axis=0))
print(pd.concat([data,data2],axis=1))
```

stack:
=====

stack is a linear data structure  and it is also called as

"LIFO( Last In First Out)" data structure

when we are adding the elements into stack data structure,

the elements will always added "top" of the stack for every new

insertion

when we are performing the "insertion" operation on the stack,

it is termed as "Push" operation

top means "first element" in the stack

top is also called "peek" element in the stack

when we are deleting the any element, it always removed the

top element  from the stack and which element is added

recently will deleted first always from the stack

when we are deleting the element from the stack, this

operation is termed as "Pop" operation

push===>when we done this operation on stack, a new element

is added at top

pop===>when we done this operation on stack, a  element

is deleted at top, basically it is top element element

only

peek===>when we done this operation, it returns "always top

element"

when the stack is empty, we can not perform "pop or deletion"

even we perform the deletion on the empty stack, this scenario

is called as "stack underflow"

when the stack is empty , top=-1

when the stack is full, we can not perform "insertion or push"

even we perform the insertion on the filled stack, this scenario

is called as "stack overflow"

when the stack is full , stack size is max size

implementing the stack using List:
==========================

```
"""working with stack"""
stack=[]
max_size=10
def is_empty():
    return len(stack)
def push():
    if is_empty()<=max_size:
      element=int(input("Element:"))
      stack.insert(0,element)
    else:
        return "Stack is Overflow"
def pop():
    if is_empty()==0:
        return "stack is underflow"
    else:
        stack.pop(0)
def peek():
    return stack[0]
def display_stack():
    print(stack)
flag=True
while flag:
    print("1.display stack\n2.push\n3.pop\n4.peek")
    print("choose the option(1-4):")
    option=int(input("Option:"))
    match option:
        case 1: display_stack()
        case 2: push()
        case 3: pop()
        case 4: print(peek())
        case _:print("given wrong option")
    print("do you want to continue or not: yes or no:")
    flag=input("yes/no:").lower()
    if flag=="no":
       break
```

implementing the stack using deque class from the collection

module:
==================================================

example:
=======

```
"""working with stack using deque"""
from collections import deque
stack=deque()
stack.append(10)
stack.append(20)
```

```
stack.append(30)
stack.append(40)
stack.append(50)
print(stack)
stack.pop()
print(stack)
stack.pop()
print(stack)
print(stack[-1] if stack else "stack is empty")
stack.pop()
stack.pop()
stack.pop()
print(stack[-1] if stack else "stack is empty")
```

implement stack using custom class:
==============================

```
"""working with stack using custom class"""
class Stack:
    def __init__(self):
        self.stack=[]
        self.max_size=10
    def push(self,element):
        if len(self.stack)<self.max_size:
            self.stack.insert(0,element)
        else:
            print("stack is overflow")
    def pop(self):
        if len(self.stack)!=0:
         return self.stack.pop(0)
        else:
            print("stack is under flow")
    def peef(self):
        print(self.stack[0])
s1=Stack()
s1.push(1)
s1.push(2)
s1.push(3)
print(s1.stack)
s1.pop()
s1.pop()
print(s1.stack)
```

stack real-time examples:
=====================

Queue:
======

queue is a linear data structure  and it is also called as

"FIFO( First In First Out)" data structure

when we are adding the elements into queue data structure, the elements will always added  at "rear" of the queue for every

new insertion

when we are performing the "insertion" operation on the queue,

it is termed as "enqueue" operation

when we are deleting the any element, it always removed the

front of the queue and which element is added

first into the queue will deleted first always from the queue

when we are deleting the element from the queue, this

operation is termed as "dequeue" operation

enqueue===>when we done this operation on queue, a new

element is added at rear

dequeue===>when we done this operation on queue, a

element  is deleted at front, basically it is first

element  only from queue

when the queue is empty, we can not perform "deque"

even we perform the deletion on the empty stack, this scenario

is called as "queue underflow"

when the queue is empty , front and rear at same position

when the queue is full, we can not perform "enque"

even we perform the insertion on the queue is full, this

scenario is called as "queue overflow"

queue can implemented in following ways:
==================================

1.using list(but is not recommended):
==================================================
code:
====

```
"""
implementing the Queue using list
"""
```

```
queue=[]
Flag=True
while Flag:
    print()
    print("1.Insert\n2.delete")
    value=int(input("enter the value:"))
    match value:
        case 1:
              element=input("Element:")
              queue.append(element)
              for i in queue:
                print(f"{i}<--",end=" ")
        case 2:
              queue=queue[1:]
              for i in queue:
                  print(f"{i}<--",end=" ")
    if value!=1 or value!=2:
        Flag=False
```

2.using collections module, deque class:
==================================================

enque <==== insert the element into the queue

deque <=== delete the element from  the queue

example:
=======

```
"""
implementing the Queue using deque
"""
from collections import deque
import time as t
"""create the object for deque"""
q1=deque()
print("inserting the 4 elements.... ")
q1.append(100)
q1.append(200)
q1.append(300)
q1.append(400)
"""displaying the queue"""
t.sleep(3)
for i in q1:
    print(f"{i}<==",end="")
"""deleting the queue"""
q1.popleft()
q1.popleft()
print()
"""displaying the queue"""
t.sleep(3)
for i in q1:
    print(f"{i}<==",end="")
```

3.using collection module, priority queue class
==================================================

which will have the more priority will be removed first from

queue

to implement the priority queue  will use a module called

queue, in this we will have a class called "PriorityQueue"

example:
=======

```
from queue import PriorityQueue
pq=PriorityQueue()
"""adding the elements"""
pq.put((2,'Job1'))
pq.put((3,'Job3'))
pq.put((4,'Job4'))
pq.put((1,'Job1'))
pq.put((0,'Job0'))
"""remove the elements from priority queue based on
priority"""
while not pq.empty():
    priority,job=pq.get()
    print(f"{job} with {priority} is deleted now")
```

4. using thread safe approach

5. using custom class approch
===========================

```
"""
implementing the Queue using class
"""
class Queue:
    def __init__(self):
        self.queue=[]
    """insert"""
    def insert(self):
        element=input("element:")
        self.queue.append(element)
    """delete"""
    def delete(self):
        self.queue=self.queue[1:]
```

```
    """display"""
    def display(self):
        for i in self.queue:
            print(f"{i}<==",end=" ")
"""create the object to class"""
q1=Queue()
print("1.Insert\n2.Delete\n")
value=input("Enter the Value:")
match value:
    case 1:
          q1.insert()
          q1.display()
    case 2:
          q1.delete()
          q1.display()
```

linked list:
========

linked list is a "collection of nodes are connected together"

in a linked list , entire the  data in the form node

node is a basic element or basic building block of linked

list

the node can contain both "data and pointer to reference

the next node in the linked list"

the first node in the linked list is called as "head" node

the last node in the linked list is called as "tail" node

in real-time, we will have three types of linked list:
========================================

1)Single linked list :
================

in this linked list, we can have only one data field and only one

reference field in the node of the linked list

example:
=======

```
"""create node class"""
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
```

```
"""create the linked list class"""
class LinkedList:
    def __init__(self):
        self.head=None
    """insert the node in the linked list"""
    def append(self,data):
        new_node=Node(data)
        if not self.head:
            self.head=new_node
            return
        current=self.head
        while current.next:
            current=current.next
        current.next=new_node
    def display(self):
        current=self.head
        while current:
            print(current.data,end="-->")
            current=current.next
    def delete(self,value):
        if not self.head:
            print("linked List is Empty")
        """to remove the head only"""
        if self.head.data==value:
            self.head=self.head.next
            return
        """check the value, delete the node"""
        current=self.head
        while current.next and current.next.data!=value:
                current=current.next
        if current.next:
                current.next=current.next.next
        else:
            print("value is not found in the list")
l1=LinkedList()
l1.append(10)
l1.append(20)
l1.append(30)
l1.append(40)
l1.display()
l1.delete(10)
print()
l1.display()
```

2)double linked list:
================

```
"""create node class"""
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
        self.prev=None
```

```python
"""create the linked list class"""
class LinkedList:
    def __init__(self):
        self.head=None
    """insert the node in the linked list"""
    def append(self,data):
        new_node=Node(data)
        if not self.head:
            self.head=new_node
            return
        current=self.head
        while current.next:
            current=current.next
        current.next=new_node
        new_node.prev=current
    def display(self):
        current=self.head
        while current:
            print(current.data,end="-->")
            current=current.next

l1=LinkedList()
l1.append(10)
l1.append(20)
l1.append(30)
l1.append(40)
l1.display()
```

searching and sorting algorithms:
==========================

1.searching algorithms:
===================

1.linear search:
============

linear search is a "blind search algorithm"

when we want to perform linear search, we need a key( search

element) and also data

example-1:
=========

```
"""implementing linearch search using python list"""
l1=[1,2,3,4,5,6,7,8,9,10]
search_element=int(input("enter the search element:"))
if search_element in l1:
    print(f"the element is found at:{l1.index(search_element)} th index")
else:
    print("search is unsuccessful")
#time complexity:O(n)
```

example-2:
=========

```
"""implementing linearch search using python list"""
l1=[1,2,3,4,5,6,7,8,9,10]
search_element=int(input("enter the search element:"))
count=0
for i in l1:
    if search_element==i:
      print(f"the element is found at:{l1.index(search_element)} th index")
      break
    else:
        count+=1
if count==len(l1):print("search is unsuccessful")
#time complexity:O(n)
```

note:
====

when we want to perform the linear search, the data need not

be in any sorting order

Binary search:
===========

when we want to perform the binary search , the data need

to be sorted

when we are doing the binary search, we will find the mid

element

mid=(low+high)/2

check the given search element with mid element,

if mid element is same as search element, stop searching

display "search is successful and element found at position"

if search element is less than mid element, please perform

searching at left side of the elements to mid element

if search element is greter than mid element, please perform

searching at right side of the elements to mid element

```
search<mid                   search>mid
[left side]    mid element  [right side]
```

code:
====

```
"""implementing binary search using python list"""
l1=[1,2,3,4,5,6,7,8,9,10]
target=int(input("enter the target:"))
mid=(0+len(l1)-1)//2
if l1[mid]==target:
    print(f"element is found at {l1.index(target)} index")
elif target<l1[mid]:
    if target in l1[:mid]:
        print(f"element is found at {l1.index(target)} index")
    else:
        print("target is not found")
else:
    if target in l1[mid+1:]:
        print(f"element is found at {l1.index(target)} index")
    else:
        print("target is not found")


#time complexity:O(logn)
```

sorting algorithms:
===============

bubble sort:
==========

bubble sort algorithm is used to sort the data  and  one of

simple sorting technique

in sorting algorithm, we will use two techniques:
========================================

(i)  Comparison

(ii)  Swap

this is best for "when the data set is small", it not

recommended sorting technique for "large dataset"

in bubble sort, the maximum number of passes are "n-1"

code:
====

```
"""code for bubble sort"""
l1=[7,6,4,3]
for i in range(len(l1)):
    for j in range(0,len(l1)-i-1):
        if l1[j]>l1[j+1]:
            l1[j],l1[j+1]=l1[j+1],l1[j]
print("final result:",l1)
#time complexity:O(n power 2)
```

2.insertion sort:
=============

insertion sort algorithm is used to sort the data  and  one of

simple sorting technique

in sorting algorithm, we will use two techniques:
========================================

(i)  Comparison

(ii)  Swap

this is best for "when the data set is small", it not

recommended sorting technique for "large dataset"

code:
====

```
"""implementing the insertion sort"""
l1=[12,11,5,6,13,9,2]
for i in range(1,len(l1)):
```

```
    key=l1[i]
    j=i-1
    while j>=0 and key<l1[j]:
        l1[j+1]=l1[j]
        j-=1
        l1[j+1]=key
print(l1)
#time complexity: O(n power 2)
```

selection sort:
===========

code:
====

```
def selection_sort(list1):
    #length of the given list
   length=len(list1)
   for i in range(length):
       min_index=i #min_index=0
       for j in range(i+1,length):
           if list1[j]<list1[min_index]:
              min_index=j
       list1[i],list1[min_index]=list1[min_index],list1[i]
   return list1
list1=[64,25,12,22,11]
result=selection_sort(list1)
print(result)
```

quick sort:
========

code:
====

```
def quick_sort(list1):
    #check the numbers element in the given 1 or 0
    if len(list1)<=1:
        return list1
    else:
        pivot=list1[-1]
        left=[x for x in list1[:-1] if x<=pivot]
        right=[x for x in list1[:-1] if x>pivot]
        return quick_sort(left)+[pivot]+quick_sort(right)
list1=[10,80,30,90,40,50,70]
result=quick_sort(list1)
print(result)
```

merge sort:
=========

code:

====

```
def merge_sort(list1):
    #check the numbers element in the given 1 or 0
    if len(list1)<=1:
        return list1
    mid=len(list1)//2
    left=merge_sort(list1[:mid])
    right=merge_sort(list1[mid:])
    return merge(left,right)
def merge(left,right):
    result_list=[]
    i=j=0
    while i<len(left) and j<len(right):
        if left[i]<right[j]:
            result_list.append(left[i])
            i+=1
        else:
            result_list.append(right[j])
            j+=1
    result_list.extend(left[i:])
    result_list.extend(right[j:])
    return result_list
list1=[10,80,30,90,40,50,70]
result=merge_sort(list1)
print(result)
```

MySQL introduction:
================
MySQL is a "DBMS"  or Database Tool and it is a software and

using this we can able to perform the following operations, like:

create database | table | view | index | ............

insert the data into table

update the data in the table

delete the data from the table ..........................

DBMS is a software and using DBMS we can able to perform the

various operations on the database

database is a "way to store the data in the real time application"

or

in Real-Time, all "application users" data will reside at Database

Database is like  a "Storage Unit of the application"

where DBMS stands for "Database Management System"

in real-time, based the way we store the data, all databases

are classified into various types, in our course we are learning

the following databases:
=================================================

1.relational databases:
==================

in this databases, all the data will be stored in the form of

"Table"

in these databases, all tables can have the relation using

various constraints

these databases are also called as "SQL-databases", because

when we want to perform any operation on the database, we

will use a query language called "SQL"

here the data is in the form of "table"

where table is collection of "rows and columns"

in sql,

row is also called as "record or tuple"

column is also called as "field or attribute"

example SQL databases tools are:
==========================

MySQL, oracle, SQL Server,  SQL Lite, IBM DB2,...........

2.document oriented databases :
==========================

in this databases, all the data will be stored in the form of

"documents"

in these databases, all documents are stored as a "Collection"

in this database, the entire data is grouped as collection, where

the collection will have the "documents"

these database is also called as "NO-SQL database", it means

it will never uses any language like "SQL", to store the any data

or to perform any operation

in this databases, the data will be stored in the form of

collections

where the collection will have data in the form of  "documents"

where the document will have the data in the form "key and

value pair"

the document-oriented database will use "BSON(it is a JSON)",

where JSON will have the data in the form "key and value" pairs

where JSON stands for "JavaScript object notation"

example:

mongoDB, cassendra,.............

when we give the data in "JSON", to the MongoDB it will convert

the entire data into "BSON" format

where BSON stands for "binary object notation"

what is SQL and sub-languages of MySQL:
=================================

SQL is a query language and using this language we can able

to perform various operations on relational databases

SQL act a interface between the developer and relational

database, relational database will store the data in the form

of  "table", that is the reason any database uses table to store the data, then the database is called as relational database and

it's software or tool is called as "RDBMS"

RDBMS stands for "Relational Database Management System"

SQL is a language

DBMS is a software

database is a collection of related data and where data is store

in some structure in the database

table is a database data structure and where the data will be

stored in the form of "Rows and Columns"

example: MySQL

document is a database data structure and where the data will

be stored in the form of "keys and values"

example: MongoDB

in the SQL, again we will have the following SQL sub-languages:
=================================================

1.DDL(Data Definition Language)

2.DML(Data Manipulation Language)

3.DQ  L(Data Query Language)

4.DCL(Data Control Language)

5.TCL(Transaction Control Language)3

MySQL with DDL commands:
======================

in SQL, we will have the following  DDL commands:
========================================

1.create :
=======

this command we will use to perform:

to create the Database | table |  view | index .......

2.drop:
======

this command we will use to perform:

to drop/delete  the Database | table |  view | index .......

3.rename:
========

this command we will used to perform:

to rename Database| table

4.alter:
======

this command we will use to perform:

to add the column into existing table

to remove the column from existing table

to change the data type of the column in existing table

to change or rename the column in existing table

to change or rename the existing table

5.truncate:
=========

this command we will use to perform:

to data only from the table and but not table

MySQL with DML:
==============

in DML, we will have the following commands:
====================================

1.insert <== using this we can insert the data into table

2.update<== using this we can update the data in the table

3.delete<== using this we can delete the data from the table

MySQL DQL(select):
================

in DQL, we will use only one command called "select"

using this command we can retrieve the data from the table

select also called as "DRL" command

where DRL stands for "Data Retrieval Language"

MySQL with DCL commands:
======================

these commands are used in the MySQL, to provide the

access permission on the database based on user type

it means , which user what to perform on the database, can be

given using "DCL" commands

in this we will have the following:

==========================

1.Grant:  it used to give the permission to user

2.Revoke: it is used to take back the permissions from  the user

MySQL with TCL commands:
======================

in MySQL, we can also work with transaction processing

when want to perform the transactions in MySQL, we will use

the following TCL commands:
================================================

1. save point:
=============

using save point, we can do any operation on the database,

because when we are doing any operation on the database via

save point, even when we done any wrong operator it never

change the database data, it means always safe

2. commit:
=========

when we want to do the changes permanently whatever we

done in the save point , in the database we will do "commit"

operation

commit is referred as "permanent save"

3. rollback:
==========

in MySQL, when want to undo the operation, we will use

"Rollback"

how to create the database in the MySQL:
================================

to create the database in the MySQL, we will use the following

syntax:
================================================

create database database_name;

example:
=======

create database employee_fp6_2024;

create database hr_fp6_2024;

how to show the list of databases in the MySQL:
=====================================

to show the list of databases in the MySQL, we will use the

following syntax:
=================================================

show databases; <=== it will give the all databases in the

MySQL

how to remove the database from the MySQL:
====================================

to remove the database from the MySQL, we will use the

following syntax:
================================================

drop database database_name;

create the database with name "Employee_fp6_2024"

list the all databases in the MySQL using "show databases"

when we are creating the any table, the table has to be created

inside the a particular database, to use a particular database,

we will use the following syntax:

use employee_fp6_2024;

to check the current database of the MySQL, we will use the

following syntax:

select database()

create the table with name "employee"

employee table:
============

| column_name | datatype | constraint |
| --- | --- | --- |
| emp_id | int | primary key |
| emp_firstname | varchar | not null |
| emp_lastname | varchar | not null |
| emp_email | varchar | not null+unique |

emp_mobileno bigint not null+ unique

emp_blood_group varchar not null

emp_gender varchar not null

emp_deptid int not null

emp_dept_name varchar not null

emp_location varchar not null

emp_zipcode int not null

emp_dob date not null

emp_jod date not null

emp_role varchar not null

emp_salary float not null

```
create table employee(emp_id int primary key,
emp_firstname varchar(100) not null,
emp_lastname varchar(100) not null,
emp_email varchar(100) not null,
emp_mobileno bigint not
null, emp_blood_group varchar(10) not null,
emp_gender varchar(10) not null,
emp_dob date not null,
empjod date not null,
emp_location varchar(100) not null,
emp_zipcode int not null,
emp_deptid int not null,
emp_deptname varchar(100) not null,
emp_role varchar(100) not null,
emp_salary float not null);
```

once we create the table, we will use the following syntax to

see the table structure,

desc employee;

to insert the data into MySQL table, we will use a command

called "insert"

```
insert into employee values(1000,'ram','A',
'ram@gmail.com',9874561231,'B+ve',
'male','1988-9-8','2010-5-5',
'hyderabad',587896,1234,'Development',
'developer',15600);

insert into employee values(1001,'kumar','p',
'kumar@gmail.com',7874555231,'A+ve',
'male','1988-9-18','2012-12-5',
'Banglore',555596,1234,'Development',
'developer',65600);
```

insert into employee values(1003,'rajitha','k',
'rajitha@gmail.com',9844555231,'o+ve',
'female','1998-9-18','2020-9-5',
'chennai',578996,4321,'Testing',
'Test Engineer',95600);

insert into employee values(1004,'kalyan','j',
'kalyan@gmail.com',7234555231,'o+ve',
'male','1998-9-18','2020-8-15',
'chennai',456996,4321,'Testing',
'Test Engineer',95600);

how to retrieve the data from the MySQL table:
====================================

to retrieve the data from the MySQL table, we will use a

command called "select"

to get the all columns from the MySQL table, we will use the

following syntax:
=================================================

select * from table_name;

or

to get the all columns from the MySQL table, but based on

some condition, we will use the following syntax:
=================================================

select * from table_name where condition;

when we want to retrive the only particular columns from the

MySQL table, we will use the following syntax:
=================================================

select col1, col2, col3, .....coln from table_name where

condition;

example:
=======

select * from employee;

get the data related salary less than 10000:
==================================

select * from employee where emp_salary<10000;

get the data related salary greater than 10000:
=====================================
select * from employee where emp_salary>10000;

get the employee names from the employee table, where salary

is not equal to 10000:
==================================================

select emp_firstname,emp_lastname from employee where
salary!=10000;

get the employee details , employees who born before

'1999-01-01' and after '1980-01-01':
==================================================

select * from employee where emp_dob>'1980-01-01' and

emp_dob<'1999-01-01';

or

select * from employee where emp_dob between '1980-01-01'

and '1999-01-01';

get the employee id, firstname, email from employee table ,

employees id need to be display in descending order:
=================================================

select emp_id, emp_firstname, emp_email from employee

order by emp_id  desc;

select emp_id, emp_firstname, emp_email from employee

order by 1  desc;

select emp_firstname, emp_id,  emp_email from employee

order by 2  desc;

get the employee details, show only where the employee name

starts with "a":
==================================================

select * from employee where emp_firstname like 'a%';

get the employee details, show only where the employee name

ends with "a":
==================================================

select * from employee where emp_firstname like '%a';

get the employee details , show only gmail related emails

only from employee table:
==================================================

select * from employee where emp_email like "%gmail%";

get the employee details from employee, employee name

length must be 6 characters only:
====================================================

select * from employee where emp_firstname like '______';

get the employee details, where salaries must be descending

order and display only top 5 rows from the employee table:
====================================================

select * from employee order by emp_salary desc limit 5;

get the employee details,  whose salary is highest in the

company:
=============================================

select * from employee order by emp_salary desc limit 1;

get the company's second highest salary of the employee:
=============================================

select emp_salary from employee order by emp_salary desc

limit 1,1;

get the company's second lowest salary of the employee:
=============================================

select emp_salary from employee order by emp_salary asc

limit 1,1;

get the total salary paid by the company to the employees:
====================================================

select sum(emp_salary) from employee;

get the total average salary paid by the company to the

employees:
====================================================

select avg(emp_salary) from employee;

get the how many rows in the employee table:
====================================================

select count(*) from employee;

get the how many people working in the each location count

from the employee table:
================================================

select emp_location, count(*) from employee group by

emp_location;

get the how many people working in the each location count

from the employee table, (minimum 2 employee has to work

in the every location related data only):
================================================

select emp_location, count(*) from employee group by

emp_location having count(*)>=2;

MySQL Data types
==============

data types in MySQL are very important, to define the column of

the table, will have the what type of data

when we are creating the table, each column we need to define

with data type

the data type of the column defines, the what type of data the

column can have

in MySQL, we will have the following data types:
=====================================

numeric type data or number type data:
================================

for integer data:
=============

int, integer

tinyint, smallint, bigint, long

for floating-point number type data:
============================

float, double, decimal

character type or string type data:
===========================

char, varchar

smalltext, mediumtext, text

binary type data:
=============

blob

time and date type data:
===================

time<==== HH:MM:SS

date<====  YYYY-MM-DD

datetime<====YYYY-MM-DD HH:MM:SS

timestamp<=== current time stamp(which have both date and

time)

year<==== using this we will give just year number

Enum type===>this  is used to take a value from multiple

values

set type===>this is used to store multiple values

how to create the table in MySQL:
==========================

to create the table in the MySQL, we will use the following syntax:
=================================================

create table table_name (col1 type constraint, col2 type

constraint,.....coln type constraint);

how to see the table structure in the MySQL:
===================================

to see the table structure in the MySQL, we will use the

following syntax:
=================================================

desc table_name

or

describe table_name;

how to remove the table or drop the table in the MySQL:
============================================

to drop the table in the MySQL, we will use following syntax:
===============================================

drop table table_name;

MySQL constraints:
=================================================

constraints are used to make the columns of the table, will have right data or make the columns will have valid data

in order to make the columns of the MySQL table will have valid

data, we will use the following constraints:
=================================================

1.NOT NULL:
=================================================

"when we make the column of the table as not null, the

column will not allow the null values, but allow duplicate

values"

in MySQL, any number columns can have not null as a

constraint

2.UNIQUE:
========

"when we make the column of the table as unique, the

column will  allow the null values, but not allow duplicate

values"

in MySQL, any number of columns can have "unique" as a

constraint

3.PRIMARY KEY(NOT NULL+UNIQUE):
============================

"when we make the column of the table as primary key, the

column will not allow the null values and not allow duplicate

values"

in MySQL, only one column can be act as primary key column

if we want to make another column can act like primary

key column, then define the column with both unique

and not null

4.DEFAULT:
=========

"when we want to define the column with some value as

default value, while creating the table using "DEFAULT"

constraint

this value will be taken by the table, only we are not giving

any value while inserting the data into the table

5.CHECK:
=======

"when we define the any column with CHECK constraint, the column will always check the data what we given is valid or not using the condition what we given for CHECK constraint, if the value is not valid, then the MySQL will raises an error, otherwise it will insert the data into the column"

example:
=======

salary>=10000

age>=20 and age<=70

6.AUTO_INCREMENT:
=================

"when we make the column as auto_increment, the column will get the next value, by incrementing the previous value by 1"

the default value of the auto_increment column is "1" , but reset this value whatever we want using "alter"

when we give the column as auto_increment the column must be "primary key or unique" column

7.FOREIGN KEY:
============

it will provide the relationship between two tables

foreign key is a key which represent the primary key column of the another table

it means "the column in one table which represents same as another table column"

the two tables will have a same related column which is makes the relation in between two tables

the column which represents as "primary key" in the table, the table called as "Master or Parent table"

the column which represents as "Foreign key" in the table, the table called as "Child table" the master table column can be also "unique", but unique may

allow "Null" values, that is reason "we are making the column

always primary key"

MySQL operators:
==============

in MySQL, we will have the following operators:
=====================================

1.airthmetic operators:
==================

+ ===>addition ===> select 10+20; ===> 30

- ===>subtraction ===> select 10-20; ===> -10

*===>product ======>select 10*20===>200

/===>division ======>select 10/20 ===>0.5

%===> modulo division ===>select 10%20 ===>10

2.assignment operator:
===================

it used to assign the data to the variable in MySQL, using

the following symbol:
================================================

:=

syntax:
======

set  variable_name:=value

3.comparision operator:
===================

in MySQL, we will have the following comparison operators:
==============================================

1.> ====>  select 10>20; ===>0

2.< ====>  select 10<20; ===>1

3.>=====>  select 10>=20; ===>0

4.<= ====>  select 10<=20; ===>1

5.= ====>  select 10=20; ===>0

6.!= or <> ====>  select 10!=20; ===>1

4.Logical Operator:
===============

in MySQL, we will have the following logical operators:
==========================================

logical or:

or =======>select (10>20) or (10<20) ===> 1

logical and:

and=====> select (10<20) and (10>20) ====> 0

logical not:

not ====>select not (10>20);  ===> 1

note:
====

in MySQL,

true means "1"

false means "0"

5.bitwise operator:
===============

in MySQL, we will have the following bitwise operators:
===========================================

1.bitwise or     ====> select 10 | 20 ===>30

2.bitwise and ====> select 10 & 20 ===>0

3.bitwise exclusive or ====> select 10 ^ 20 ===> 30

4.bitwise left shift ====> select 10<<2 ===>40

5.bitwise right shift====> select 10>>2 ===>2

in operator:
==========

this operator we are using "if we want to get the data related to

multiple values, we are using in operator in MySQL".

in MySQL, we can also use "not in", not in give opposite to

in operator

is operator:
=========

this operator we are using "to get the null values of the given

column"

in MySQL, we can also use "is not", using this we can get "non

null value of the given column"

between operator:
==============

this operator we are using "to get the data which a particular

range of the given column"

example:
=======

salary between 10000 and 50000;

like operator:
===========

this operator is used to get the data based on the

given pattern

when we are create the pattern, we will use the following

symbols with like operator:
=================================================

_  ===>underscore represents only single character

% ===> percentage represents zero or more characters

example:
========

firstname like a% ===> starting with a, after a any number of

characters

example:  arun, anand,  Akhilesh,

firstname like %a ===>ends with a, starting with any number of

characters

example:  india, malasiya, ..................

firstname like %a% ====>starts or ends with any number of

characters, but "a" should be there in

the name in any place

firstname like a___n ====>starts with a, ends with n and

between  a and n, there should be

three characters only

example:   aaaan,  a000n,

firstname like ______ ===>it check exactly any name with any

characters with length six characters

salary like 1% ===>salary start with 1

salary like 1%0 ===>salary start with 1and ends with 0

note:
====

the both "_" and "%" are called as "wild card characters"

special operators in MySQL:
======================

case

if

coelase

MySQL clauses(where, from, order by, group by, having):
============================================

where clause:
===========
where clause is used to " specify the condition in query"

from clause:
==========

from clause is used to "specify from which table or view"

order by clause:
=============

order by clause is used to "to sort the data either in the

ascending order or descending order"

asc ====> ascending order

desc ===>descending order

note:
====

by default, in MySQL we will have the asceding order

group by:
========

this clause is sued to "group the data based on the certain

category"

having:
======

this clause is used to "to specify the condition to group by

query"

limit clause:
=========

this clause will be used in the MySQL, "to get the a specific

number of rows or records from the table"

note:
====

limit will also take syntax as follows:
=============================

limit rownumber, number_of_rows;

for example,

limit 1,2 ===>from row number 2, we need 2 rows or records

limit 5,12 ===>from row number 5, we need 12 rows or records

limit 10,5 ===> from row number 10, we need 5 rows or

records

note:
====

in MySQL, the row number always starts from "0"

MySQL aggregate functions:
======================

in MySQL, we will have the following aggregate functions:
=============================================

1. sum() ===>it will give the sum of the all values of the given

column

2. min()===>it will give the minimum value of the given

column

3. max()===>it will give the maximum value of the given

column

4.count()===>it will give the number of values present in the

given column

5. avg()===>it will give the average value of the values which

are present in the column

how to create the duplicate tables or how to copy the data of

the one table into another table:
==================================================

to create the duplicate table in the MySQL,  we will use the

following ways:
================================================

1.using like clause:
===============

to create the duplicate table using like clause, we will the

following syntax:
===============================================

```
create table table_name like original_table_name;
```

example:
=======

```
create table employee_new like employee;
desc employee;
desc employee_new;
select * from employee_new;
insert into employee_new select * from employee;
```

note:
====

when we create the duplicate table using "like", data is not

copied, only structure is copied

2.using select :
============

using select, in MySQL we can create the duplicate table

using select, when we create the duplicate table the both table

structure and data will copied

syntax:

```
create table table_name select * from original_table_name;
```

example:
=======

```
create table employee_new2 select * from
employee;
desc employee_new2;
select * from employee_new2;
```

MySQL views:
==========

views are "virtual tables" and these tables will always represent

only "parent table data only"

views are used to make the table  into "n" number of virtual

tables, where we can maintain  the data in views based on the

certain condition  or a particular value or category the main advantage of the creating the view,  to increase the

performance while retrieving the data and other operations

both view and table will refer same data, that is reason when

we do the changes in view will also see the same changes in the

table also

on view, same like normal table, we can do  "all insert, update,

delete operations"

to create the view in MySQL, we will use the following syntax:
===============================================

```
create view view_name as query;
```

example:
=======

```
create view v1 as select * from employee
where emp_location='hyderabad';
create view v2 as select emp_id,emp_firstname,
emp_lastname from employee where
emp_location='chennai';
select * from v1;
select * from v2;
```

to change the data or update the view, we will use the following

syntax in MySQL:
=================================================

```
create or  replace  view view_name as query;
```

example:
=======

```
create view v1 as select * from employee
where emp_location='hyderabad';
create or replace view v1 as select * from
employee where emp_deptname='development';
select * from v1;
```

to remove the view from the MySQL database, in MySQL we will

use the following syntax:
=================================================

```
drop view view_name;
```

example:
=======

```
drop view v1;
drop view v2;
```

note:
====

the operations what we do on the table, we can perform on

the "view" also, the changes what we done on the "view" , we

can see in the table and vice versa

when we remove the view from the table, it will not effect the

table. only the changes what we done on the data, will be

reflect in both view and table

MySQL Temporary tables:
====================

Temporary table is a table in MySQL, which is created like

normal table only, but it will be deleted automatically when

close the MySQL session

these tables created to "Save the memory and these tables used

to test the operations on tables without working on the original

tables directly"

this table will not be "shown" in the original table list of the

current database

to create temporary table in MySQL, we will use the following

syntax:
=================================================

```
create temporary table table_name(col1 type, col2 type,....... );
```

example:
========

```
create temporary table sample like employee;
create temporary table sample2 select *
from employee;
select * from sample;
select * from sample2;
show tables;
```

working with alter command in MySQL:
==============================

using alter, we can add the column in the  existing MySQL table

using following syntax:
=================================================

```
alter table table_name add col_name data type first | after , add

col_name  data type,.............................................
```

in the alter query,

    if we use first, first represents the add the column at

    starting of the MySQL table

    if we use after, after represents add the column after a

    particular column in the MySQL table

example:
=======

```
desc employee;
alter table employee add emp_sno int unique
auto_increment first;
alter table employee add emp_midname
varchar(100) not null after emp_firstname;
alter table employee add emp_project_id int
not null;
```

to remove the any column from the MySQL table, we will use

the following syntax:
================================================

```
alter table table_name drop column_name
```

example:
=======

```
desc employee;

alter table employee drop emp_sno;

alter table employee drop emp_midname;

alter table employee drop emp_project_id;
```

using alter, we can modify the any column data type in the

table using following syntax:
================================================

```
alter table table_name modify column_name data type

constraint_name,..................
```

example:
=======

```
desc employee;
alter table employee modify emp_firstname
varchar(500);
alter table employee modify emp_lastname
varchar(500), modify emp_email varchar(500);
```

using alter we can change the column name in MySQL table

using following syntax:
=================================================

code:

====

```
use employee_fp6_2024;
select * from employee;
alter table employee change column
emp_id id int;
alter table employee change column
emp_firstname
firstname varchar(255),
change column
emp_lastname lastname varchar(255);
```

how to rename the table in MySQL using alter:
====================================

to rename the table in MySQL using alter, we will use the

following syntax:
================================================

```
alter table table_name rename to new_table_name;
```

example:
=======

```
use employee_fp6_2024;
show tables;
select * from employee_fp6;
alter table employee rename to  employee_fp6;
```

note:
====

using alter at a time we can able to all actions like

add the column, rename the column, modify the data type ,

drop the column

example:
=======

```
desc employee_fp6;
alter table employee_fp6
add sno int auto_increment
not null unique, change column
emp_mobileno mobileno bigint,
modify emp_email varchar(200);
```

MySQL unions:
============

unions are used to combine the two results

or

unions are used to combine the two queries or two select

statements results

in MySQL, we can use the union in two ways:
===================================

union <=== it excludes the duplicates also union all <=== it includes duplicates also

syntax for union :
==============

```
select col1, col2,col3,col4....coln from table1;

union

select col1,col2,col3,col4,....coln from table2;
```

syntax for union all:
================

```
select col1, col2,col3,col4....coln from table1;

union all

select col1,col2,col3,col4,....coln from table2;
```

when we want to apply the union, we need to remember the

following:
=================================================

1. the number of columns in the both queries must be same

2. the result of the union always will appear under the names

    first query column names

3. when we want to apply the union on two quires, the

    data type of the columns in the queries must be same and

    same as position wise also (there will be a convention also)

```
create table mysample(id int, location varchar(100));
create table mysample2(id int, location varchar(100), email
varchar(100));

create table mysample(id int,
location varchar(100));
create table mysample2(id int,
location varchar(100), email
varchar(100));
insert into mysample values(1,'abc');
insert into mysample values(1,'xyz');
insert into mysample values(1,'pqr');
insert into mysample2 values(1,'abc',"no value");
insert into mysample2 values(1,'xyz',"no value");
insert into mysample2 values(1,'pqrl',"no value");
insert into mysample2 values(1,'abcd',"no value");
insert into mysample2 values(1,'xyza',"no value");
insert into mysample2 values(1,'pqrs',"no value");
```

working with union and union all:
=========================

```
select * from mysample;
select * from mysample2;
```

```
select id from mysample
union select id from mysample2;
select id from mysample
union all select id from mysample2;

select location from mysample
union select location from mysample2;
select location from mysample
union all select location from mysample2;
```

MySQL joins:
==========

joins are used in MySQL, two retrieve the data from two or more

tables at a time

when we are using joins, we no need to  bother about how

many columns we need to retrieve  from the each table, it

means we can take any number column from each table while

using joins

joins are combine the result of multiple data as a single result

based on the certain condition

in MySQL, we can have the following joins:
=================================

1.inner join or equi join

2.left join  or left outer join

3.right join  or right outer join

4.cross join or Cartesian product of the tables

5.full join

6.self join

create the following table in the database

called "employee_fp6_2024"

employee_project ( id, firstname, lastname, project_id,

project_name, project_lead_name, project_start_date,

project_end_date)

working with joins:
===============

working with inner join:
===================

```
select employee_fp6.id,employee_fp6.firstname,
employee_fp6.lastname,employee_fp6.email,
employee_project.project_id,
```

```
employee_project.project_name from
employee_fp6
inner join
employee_project
on
employee_fp6.id=employee_project.id;
```

working with left join:
=================

```
select employee_fp6.id,employee_fp6.firstname,
employee_fp6.lastname,employee_fp6.email,
employee_project.project_id,
employee_project.project_name from
employee_fp6
left join
employee_project
on
employee_fp6.id=employee_project.id;
```

working with right join:
==================

```
select employee_fp6.id,employee_fp6.firstname,
employee_fp6.lastname,employee_fp6.email,
employee_project.project_id,
employee_project.project_name from
employee_fp6
right join
employee_project
on
employee_fp6.id=employee_project.id;
```

working with full join:
=================

in MySQL, we do not implement full join directly, to implement

full join, we will use union on both  left join and right join

example:
========

```
select employee_fp6.id,employee_fp6.firstname,
employee_fp6.lastname,employee_fp6.email,
employee_project.project_id,
employee_project.project_name from
employee_fp6
left join
employee_project
on
employee_fp6.id=employee_project.id

union

select employee_fp6.id,employee_fp6.firstname,
employee_fp6.lastname,employee_fp6.email,
employee_project.project_id,
employee_project.project_name from
employee_fp6
right join
employee_project
on
employee_fp6.id=employee_project.id;
```

working with cross join:
=================

syntax:
======

```
select col1,col2,col3, ...coln from table_name1

cross join table_name2;
```

example:
=======

```
select employee_fp6.id, employee_fp6.firstname,

employee_fp6.lastname from employee_fp6

cross join employee_project;
```

working with self join:
=================

self join means "apply the join operation (inner or left or right)

on same table(both left and right table is same table)"

example:
=======

```
select e1.id,e1.firstname,e1.lastname,e2.location,e2.email
from employee_fp6 e1
inner join
employee_fp6 e2
on
e1.location!=e2.location;


select e1.id,e1.firstname,e1.lastname,e2.location,e2.email
from employee_fp6 e1
left join
employee_fp6 e2
on
e1.location!=e2.location;


select e1.id,e1.firstname,e1.lastname,e2.location,e2.email
from employee_fp6 e1
right join
employee_fp6 e2
on
e1.location!=e2.location;
```

working with nested queries / sub-queries:
================================

sub-query or nested query:
====================

sub-query means "Writing the query inside the another query"

generally we will write the sub-query for sake of the a query result is depend on the another query

in general, we will have the following types of sub-queries:
=============================================

1.single row sub-query ===>it will exactly one row as result

2.multi-row sub-query====> it will exactly return more than

one row as result

3.multi-column subquery===> when we create the sub-query

more than one column

4.correlated subquery ====>this sub-query will have effect the

multiple subquery results in the

entire nested or sub-query

example:
=======

1.write a query to retrieve the result of employees where their

salary is more then Hyderabad location average salary

select * from employee where salary > (select avg(salary)

from employee where location='hyderabad' )

2. write a query to retrieve the result of the employee where

employee should born after the employee who's id is 1003

select * from employee where dob > (Select dob from employee

where empid=1003)

3.get then employee name and where the employee who is

getting the salary as second highest in the employee

organization

select firstname from employee where salary=(select

distinct salary from employee order by salary desc limit 1,1)

4. get the employee details where the employee work in

more than 1 department in the organization

select * from employee where empid in (

```
    select empid  from employee group by empid
    having count(departement_name)>1)
```

5.get the employee details and employee must work in
Hyderabad location and employee salary is more than
salary of the employee works in Mumbai and the salary
is average salary of the Bangloore

```
  select * from employee where location='hyderabad' and
  salary>( select salary from employee where location='mumbai'
  and salary=(select avg(salary) from employee where
  location='bangloore'))
```

get top 5 employee details where employee works in more than
one department and where display their salaries in the
descending order

```
   select * from employee where empid in (
    select empid  from employee group by empid
    having count(departement_name)>1)
   order by salary desc limit 5
```

get the employee details where salary is more than average
salary of their respective department

```
select e1.* from employee e1  where salary >
( select avg(salary) from employee e2 where
  e2.dept_id= e1.dept_id)
```

Python Database Communication(PDBC):
===============================

in order to communicate with MySQL database, in python we
will use the following module:
=================================================

mysql-connector-python

to install above module, we will use the following syntax:
=================================================

```
pip install mysql-connector-python
```

to work with MySQL  and python, we will use the following code template:

```
#import the MySQL module
import mysql.connector as mysql

"""create the connection between mysql and python"""
con=mysql.connect(host="localhost",user="root",password="mysql_password")

"""create the cursor"""
cur=con.cursor()

"""execute the sql query using cursor"""
cur.execute("write the sql query here")

"""close the mysql connection with python"""
con.close()
```

to create the database using python in MySQL, we will use the

following code:
==================================================

```
#import the MySQL module
import mysql.connector as mysql
"""create the connection between mysql and python"""
con=mysql.connect(host="localhost",user="root",password="123456789")
"""after creating the connection, we will create the cursor
to execute the any sql query"""
cur=con.cursor()
"""execute the sql query using cursor"""
cur.execute("create database Sample_fp6_new_2024")
"""close the mysql connection with python"""
con.close()
```

to list the all databases of the mysql using python, we will use

the following code:
==================================================

code:
====

```
#import the MySQL module
import mysql.connector as mysql
"""create the connection between mysql and python"""
con=mysql.connect(host="localhost",user="root",password="123456789")
"""after creating the connection, we will create the cursor
to execute the any sql query"""
cur=con.cursor()
"""execute the sql query using cursor"""
cur.execute("show databases")
for i in cur:
    for j in i:
        print(i,end=",")
"""close the mysql connection with python"""
con.close()
```

to create the table with name "sample" in the mysql database

called "sample_fp6_new_2024"  using python:
==================================================

code:
====

```
#import the MySQL module
import mysql.connector as mysql
"""create the connection between mysql and python"""
con=mysql.connect(host="localhost",user="root",password="123456789",
                  database="sample_fp6_new_2024")
"""after creating the connection, we will create the cursor
to execute the any sql query"""
cur=con.cursor()
"""execute the sql query using cursor"""
cur.execute("create table sample(id int,firstname varchar(100),email
varchar(100),location varchar(100))")
for i in cur:
    for j in i:
        print(i,end=",")
"""close the mysql connection with python"""
con.close()
```

to insert the data into table called "sample", in the mysql

database called "sample_fp6_new_2024" using python:
===============================================

code:
====

```
#import the MySQL module
import mysql.connector as mysql
"""create the connection between mysql and python"""
con=mysql.connect(host="localhost",user="root",password="123456789",
                  database="sample_fp6_new_2024")
"""after creating the connection, we will create the cursor
to execute the any sql query"""
cur=con.cursor()
"""execute the sql query using cursor"""
cur.execute("insert into sample values(101,'ram','ram@gmail.com','hyderabad')")
"""commit the operation only for all dml operations(insert/update/delete)"""
con.commit()
"""close the mysql connection with python"""
con.close()
```

to insert the multiple rows into mysql table, we will use the

following code using python
=================================================

code:
====

```
#import the MySQL module
import mysql.connector as mysql
"""create the connection between mysql and python"""
con=mysql.connect(host="localhost",user="root",password="123456789",
                  database="sample_fp6_new_2024")
"""after creating the connection, we will create the cursor
to execute the any sql query"""
cur=con.cursor()
"""execute the sql query using cursor"""
cur.execute("insert into sample values(102,'raj','raj@gmail.com','chennai'),
```

```
            (103,'kalyan','kalyan@gmail.com','bangloore')")
"""commit the operation only for all dml operations(insert/update/delete)"""
con.commit()
"""close the mysql connection with python"""
con.close()
```

to get the data from "Sample" table from the "sample_fp6_new_

2024" database using python in mysql:
=================================================

code:
====

```
#import the MySQL module
import mysql.connector as mysql
"""create the connection between mysql and python"""
con=mysql.connect(host="localhost",user="root",password="123456789",
                  database="sample_fp6_new_2024")
"""after creating the connection, we will create the cursor
to execute the any sql query"""
cur=con.cursor()
"""execute the sql query using cursor"""
cur.execute("select * from sample")
for i in cur:
    print(i)
"""close the mysql connection with python"""
con.close()
```

to get the data from "Sample" related to Hyderabad location

data from the "sample_fp6_new_2024" mysql database using

python:
================================================

code:

```
#import the MySQL module
import mysql.connector as mysql
"""create the connection between mysql and python"""
con=mysql.connect(host="localhost",user="root",password="123456789",
                  database="sample_fp6_new_2024")
"""after creating the connection, we will create the cursor
to execute the any sql query"""
cur=con.cursor()
"""execute the sql query using cursor"""
location="hyderabad"
cur.execute("select * from sample where location='hyderabad'")
for i in cur:
    print(i)
"""close the mysql connection with python"""
con.close()
```

to get the data from the employee table using python from

mysql database called "employee_fp6_2024":
================================================

code:

====

```
#import the MySQL module
import mysql.connector as mysql
"""create the connection between mysql and python"""
con=mysql.connect(host="localhost",user="root",password="123456789",
                  database="employee_fp6_2024")
"""after creating the connection, we will create the cursor
to execute the any sql query"""
cur=con.cursor()
"""execute the sql query using cursor"""
location="hyderabad"
cur.execute("select * from employee_new")
for i in cur:
    print(i)
"""close the mysql connection with python"""
con.close()
```

to update the any data in mysql table using python, we will use

the following code:
=================================================

code:
====

```
#import the MySQL module
import mysql.connector as mysql
"""create the connection between mysql and python"""
con=mysql.connect(host="localhost",user="root",password="123456789",
                  database="sample_fp6_new_2024")
"""after creating the connection, we will create the cursor
to execute the any sql query"""
cur=con.cursor()
"""execute the sql query using cursor"""
location="hyderabad"
cur.execute("update sample set location='hyderabad_new'
            where location='hyderabad'")
con.commit()
"""close the mysql connection with python"""
con.close()
```

to delete the any data from the table using python, we will use

the following code:
=================================================

code:
====

```
import mysql.connector as mysql
"""create the connection between mysql and python"""
con=mysql.connect(host="localhost",user="root",password="123456789",
                  database="sample_fp6_new_2024")
"""after creating the connection, we will create the cursor
to execute the any sql query"""
cur=con.cursor()
"""execute the sql query using cursor"""
location="hyderabad"
cur.execute("delete from sample where location='hyderabad_new'")
con.commit()
"""close the mysql connection with python"""
con.close()
```

MySQL number or numeric functions:
=============================

in mysql, we will have the following number functions:
==================================================

1. floor() ===> it will give the floor value for given float value

2. ceil() ===>it will give the ceil value for given float value

3. round()==> it will give the rounded value for given float value

4. truncate()=>it will give the remove the digits after the

decimal point in the given float value

5.pow()===>it used to calculate the power for given base and

exponent

6. sqrt()===> it used to calculate the square root of given value

7. greatest()==>it used to find the maximum number from list

of given values

8. least() ==>it used to find the minimum number from list of

given values

9. mod()===>it used to find the remainder of the given two

numbers

10. div ===> it used to perform the division on given two

numbers

11.exp()===>it used to get the exponential value of the given

value

12. log()===> it used to get the logerthmetic value of the given

number

13.bin()===> it used to get the binary number for given number

14.power() ===>it is also same as pow() function

15.abs() ===>it will always return positive number

example:
=======

```
select floor(1.234);
select floor(6.9999);
select ceil(5.0);
select ceil(5.01);
select round(1.2345,2);
select round(1.2345,3);
select sqrt(16);
```

```
select sqrt(64.0);
select truncate(5.0,0);
select truncate(5.01,1);
select truncate(1.2345,3);
select greatest(1,2,3,4,5);
select least(10,20,30,40,50,-2);
select log(10);
select log(1);
select log(0);
select power(2,3);
select exp(1);
select mod(10,20);
select mod(20,10);
select 13 div 2;
select abs(-10);
```

MySQL strings functions:
====================

1.concat()===>it is used to concat the given two strings

2.length()===>it is used to give the length of the string

3.concat_ws()=>it is used to concat strings with given separator

4.ucase()=>it is used to convert given string characters into

upper case

5.upper()=>it is used to convert given string characters into

upper case

6.lcase()=>it is used to convert given string characters into

lower case

7.lower()=>it is used to convert given string characters into

lower case

8.ltrim()=>remove the spaces from the starting of the string

9.rtrim()=>remove the spaces from the end of the string

10.trim()=>remove the spaces from the starting of the string

and end of the string

11.insert()=>it is used to insert the string in the given position

12.mid()==> it is used to get the substring for given position

in the string

13.left() ===>it used to take a sub string from the left side of

the string or  starting of the string

14.right()===>it is used to take a sub string from the right side

of the string or end of the string

15.position()==> it used to know the "position of the given character"

16.repeat()==>it will repeat the given string for given number

of times

17.replace()==> it will replace the given string with specified

string

18.reverse()==> it will reverse the given string

19.substr()===> it will give the sub string for given position

and length(number character need to extract)

20.substring() ==> it will give the sub string for given position

and length(number character need to extract)

21.strcmp()==> it will used to compare the two strings

if both strings are same, it will return "zero" as

result

if string 1 length is more than string 2, it will

return "1"

if string 1 length is less than string 2, it will

return "0"

example:
=======
```
select concat("hello","world");
select concat("hello"," ","world");
select length("hello");
select concat_ws(" ","hello","world");
select concat_ws("==","hello","world");
select ucase("hello");
select upper("hello");
select lcase("HELLO");
select lower("HELLO");
select length("   hello");
select length(ltrim("   hello"));
select length("hello   ");
select length(rtrim("hello    "));

select length("   hello      ");
select length(trim("   hello    "));

select insert("hello world",1,0,"hai");
select insert("hello world",1,3,"hai");

select mid("hello world",3,5);
select mid("hello world",3,7);
select mid("hello world",2);

select left("hello world",4);
select left("hello world",5);
select right("hello world",3);
select right("hello world",8);
```

```
select repeat("hello",5);
select replace("hello world","l","L");
select replace("hello world","h","H");
select reverse("hello world");
select substr("hello world",3,3);
select substr("hello world",3,8);

select substring("hello world",4,4);
select strcmp("hello","hello");
select strcmp("hello","hello world");
select strcmp("hello world","hello");
select strcmp("hello","world");
```

MySQL date and time functions
========================

1.current_date()===>it will give today date

2.curdate()===>it will give today date

3.current_time() ===>it will give the current time

4.curtime()====>it will give the current time

5.now() ====> it will give the current date and time

6.current_timestamp()===>it will give the both date and time

7.day() ===> it will give the day number

8.month() ===>it will give the month

9.year()===> it will give the year

10.dayname()===> it will give the day name for given date

11.monthname===> it will give the month name for given date

12.dayofweek()===> it will give the  day number in the week

13.dayofmonth()===> it will give the day number in the month

14.dayofyear()===>it will give the day number in the year

15.hour() ===>it will give the hour value from  the given time

16.minute()===>it will give the minute value from  the given
time

17.second()==>it will give the second value from  the given
time

18.date_add()==>it will used to add the a value to date

19.date_sub()==>it will used to subtract the given two dates
and give the result in days

example:
=======

```
select current_date();
select current_time();
select curdate();
select curtime();
select now();
select current_timestamp();
select day("2024-11-13");
select month("2024-11-13");
select year("2024-11-13");
select dayname("1988-05-05");
select monthname("1988-05-05");
select dayofmonth('2024-11-13');
select dayofweek('2024-11-13');
select dayofyear('2024-11-13');
select hour(curtime());
select minute(curtime());
select second(curtime());
select date_add('2024-11-14',interval 10 day);
select date_add('2024-11-14',interval 10 month);
select date_add('2024-11-14',interval 10 year);
select floor(datediff(curdate(),emp_dob)/365)
from employee_fp6;


MySQL window functions
====================

create a table with name sales:
========================

id,name,year_of_sale,sales

create database sales;

use sales;

create table sales(id int primary key, name varchar(1000)

not null, year_of_sale int not null,sales float not null);

desc sales;

insert the data into sales table:
========================

insert into sales values(102,'suresh',2018,6500);
insert into sales values(103,'saliesh',2018,2500);
insert into sales values(104,'kiran',2017,7500);
insert into sales values(105,'swathi',2017,9500);
insert into sales values(106,'supraja',2016,10500);
insert into sales values(107,'kalyan',2016,3500);
insert into sales values(108,'mahesh',2015,15500);
insert into sales values(109,'srindhi',2015,16500);

generally, we will have two types of window functions:
==========================================

1.aggregate window functions:
========================
sum():
=====

example:
=======
```

```
select year_of_sale,sum(sales) from sales
group by year_of_sale order by year_of_sale asc;
```

avg():
=====

example:
=======

```
select year_of_sale,avg(sales) from sales
group by year_of_sale order by year_of_sale asc;
```

min():
=====

example:
=======

```
select year_of_sale,min(sales) from sales
group by year_of_sale order by year_of_sale asc;
```

max():
=====

example:
=======

```
select year_of_sale,max(sales) from sales
group by year_of_sale order by year_of_sale asc;
```

2.analytical window functions:
=======================

1.row_number():
=============

it will give the row number in a incremental way for the given

data

example:
=======

```
select id,name,year_of_sale,
row_number() over(order by id desc) as rownum
from sales;
```

row_number() with partition by:
=========================

```
select id,name,year_of_sale,
row_number() over(partition by year_of_sale) as rownum
from sales;
```

example-2 on partititon by:
=====================

```
select id,name,year_of_sale,
row_number() over(partition by year_of_sale
order by id desc)
```

as rownum
from sales;

rank() and dense_rank():
===================

select id,name,sales,
rank() over(order by sales) as 'rank',
dense_rank() over(order by sales) as 'denserank'
from sales;

ntile():
=====

this function will be used to "divide the table rows into  "n"

number of groups"

example:
=======
select *,ntile(2) over(order by sales desc)
as "group" from
sales;
select *,ntile(3) over(order by sales desc)
as "group" from
sales;
select *,ntile(4) over(order by sales desc)
as "group" from
sales;

get the employee details based on the department wise using

partition by
=================================================

select *,row_number()
over(partition by dept_name
order by dept_id)
as rownum
from employee;

get the all employee salaries in rank order :
==================================

select *,rank() over(order by salary desc) as rank from

employee;

select *,dense_rank() over(order by salary desc) as rank from

employee;

MySQL Comments:
===============

to write the comments in MySQL we will use a symbol called

"--"

in MySQL, we can write the comments as follows:
======================================

-- write the comments here in the mysql

MySQL Variables:
==============

variables are used to store the data

in MySQL,  we can also create the variables

in MySQL, we can have two types of variables:

1)session variables:
================

these variables will available in the MySQL till we close the

MySQL session and these variables can be created during the

MySQL session

to create the session variable, we will use the following syntax:
=================================================

```
  set @variable_name:=value
```

example:
=======

```
-- define the varibale a
set @a:=10;
-- define the variable b
set @b:=20;
-- find the result of the a and b
select @a+@b as result;
```

example:
=======

```
use employee_fp1_2024;
-- calcuate the maximum salary of employee
select @max_salary:=max(salary)
from employee;
select * from employee where
salary<@max_salary;
```

2)local variable:
=============

local variable is a variable and which is created in the procedure

in the MySQL

these variables can be created only inside the procedure only

to create the local variable in the stored procedure, we will

use  a keyword called  " DECLARE "

MySQL Stored procedures
====================

stored procedure is a callable and it is like a fucntion based

behaviour in mysql and which represents a block of code

this is used in the mysql, to perform repetitive tasks

to create the stored procedure in the mysql, we will use

following  syntax:
================================================

```
DELIMETER $$

create procedure

procedure_name(param1,param2,param3...paramn)

begin
        here we will write the code for stored procedure
end$$

DELIMETER;
```

when we are working with stored procedures, it will take two

three types of parameters

1. IN parameter  (which is used read the data and it is

                           act like input to the stored procedure)

2. OUT parameter (which is used write the data and it is

                           act like output of the stored procedure

                           , after the execution)

3. INOUT parameter ( this parameter we can use for both read

                                  write for the stored procedure)

to execute the stored procedure we will use the following

syntax in the MySQL:

```
    call name_of_the_procedure(param1,param2,.....)
```

similar like python, in MySQL stored procedures we will also

use the following:

         1)conditional statements

         2)looping statements

in MySQL, we will have the following conditional statements:
===============================================

1) if-else statement

              syntax:
                              if condition then

```
                                        statement

                              else

                                  statement

                              end if ;
```

2) else-if ladder:
==============

```
                 syntax:

                            if condition then

                                         statement

                           elseif condition then

                                 statement

                           .
                           .
                          else

                                statement

                          end if;
```

3) case statement:
================

```
       case

            when condition then

                     statement

           when  condition then

                    statement

          .

          .

          Else statement

       end case;
```

2)Looping statements:
==================

1)while loop:
==========

```
    syntax:
    ======
             while condition do
```

```
                -- logic for loop

             end while

2)repeat loop:
===========

             repeat

                 --  write the logic here
            until condition

3)for loop:
=========

       syntax:

           loop_label: loop

                if condition then

                    leave loop
               end if

                -- write the logic for loop


           end loop



example:
=======

DELIMITER $$
create procedure sum_of_two_numbers(
in a int, in b int, out result int)
BEGIN
   set result=a+b;
END$$
DELIMITER ;
call sum_of_two_numbers(10,20,@result);
select @result;


example-2:
=========
DELIMITER $$
create procedure greteast_of_two_numbers(
in a int, in b int, out result int)
BEGIN
    if a>b then
         set result=a;
      else
            set result=b;
      end if;
END$$
DELIMITER ;
call greteast_of_two_numbers(10,20,@result);
select @result;

example-3:
=========
```

```
DELIMITER $$
create procedure greteast_of_three_numbers(
in a int, in b int, in c int, out result int)
BEGIN
    if a>b and a>c then
         set result=a;
      elseif b>c then
            set result=b;
      else
        set result=c;
      end if;
END$$
DELIMITER ;
call greteast_of_three_numbers(10,20,30,@result);
select @result;
```

example-4:
=========

```
DELIMITER $$
create procedure product_numbers(
in a int, in b int,out result int)
BEGIN
 set result=a*b;
END $$
DELIMITER ;
call product_numbers(10,20,@result);
select @result;
```

example-5:
==========

```
DELIMITER $$
create procedure even_or_odd(
in a int,out result char(10))
BEGIN
    if mod(a,2)=0 then
      set result="even";
      else
      set result="odd";
      end if;
END $$
DELIMITER ;
call even_or_odd(10,@result);
select @result;
```

example-6:
=========

```
DELIMITER $$
CREATE PROCEDURE display_choice(
in choice int,out result char(100))
BEGIN
    case
      when choice=1 then
      set result="given choice is 1";
      when choice=2 then
      set result="given choice is 2";
      when choice=3 then
      set result="given choice is 3";
      else set result="no choice is matched";
    end case;
END$$
DELIMITER ;
```

exmaple-7:
=========
```
DELIMITER $$
CREATE PROCEDURE print_numbers(in max int)
BEGIN
   -- define the local variable
   declare i int default 1; -- i=1
   while i<=max do
      select i;
      set i=i+1;
   end while;
END$$
DELIMITER ;
call print_numbers(10);
```

example-8:
=========
```
DELIMITER $$
CREATE PROCEDURE print_numbers(in max int)
BEGIN
   -- define the local variable
   declare i int default 1; -- i=1
   while i<=max do
      select i;
      set i=i+1;
   end while;
END$$
DELIMITER ;
call print_numbers(10);
```

example-9:
=========

```
DELIMITER $$
CREATE PROCEDURE print_numbers2(in max int)
BEGIN
   -- define the local variable
   declare i int default 1; -- i=1
   repeat
     select i;
     set i=i+1;
   until i>max
  end repeat;
END$$
DELIMITER ;
call print_numbers2(10);
```

example-10:
==========

```
use employee_fp6_2024;
DELIMITER $$
CREATE PROCEDURE print_numbers3(in max int)
BEGIN
   -- define the local variable
   declare i int default 1; -- i=1
   loop_label:loop
    if i>max then
       leave loop_label;
      end if;
```

```
    select i;
    set i=i+1;
   end loop;
END$$
DELIMITER ;
call print_numbers3(10);
```

example-11:
==========

get the employee salary, employee name which is maximum in

their respective department from the employee table using

stored procedure:
=================================================

code:
====

```
use employee_fp1_2024;
select firstname,max(salary) as max_salary
from employee group by deptname;
DELIMITER $$
CREATE PROCEDURE max_salary_department_wise()
BEGIN
   select firstname,max(salary) as max_salary,deptname
   from employee group by deptname;
END$$
DELIMITER ;
call max_salary_department_wise();
```

MySQL CTE
=========

in MySQL, CTE stands for "common table expression"

when we have a complex query, we can make the entire

complex query into a simple  CTE expression, using this can

able to achieve the "better readability of the query"

this is introduced in MySQL from version 8 onwards

syntax for CTE:
============

```
with expression_name(

      write the query here
)
select col1,col2,col3,....coln from CTE_name where condition;
```

example-1:
=========

```
with basic_employee_info as (
select firstname,lastname,email from
employee
)
```

```
select * from basic_employee_info;
```

example-2:
=========

```
with top_10_salaries as (
  select * from employee order by salary desc limit 10
)
select * from top_10_salaries;
```

example-3:
========

get the employee details, using CTE, while retrieving the data,

make sure employee details must follow the following:

1) employee name must start with a, b, p, k

2) employee salary more than 70000

3) employee should born after 1990-01-01

code:
=====

```
with mycte_1 as (
  select * from employee where firstname like
  'a%' or 'b%' or "p%" or "k%" and salary>=70000 and
  dob>='1990-01-01'
)
select * from mycte_1;
```

get the employee details where employee salary should greater

than employee average salary of Hyderabad location:
=================================================

```
with mycte_2 as (
  select * from employee where salary >
  (select avg(salary) from employee where
  location='hyderabad')
)
select * from mycte_2;
```

MySQL indexes:
============

index is a data structure used by the MySQL , to make the

data retrieval very faster than normal retrieval from the table

when we apply the index on the MySQL table columns, it make

the table can give the data very fast due to "it apply the query

based on the index"

index data structure in the MySQL is "B+ trees(Balenced Tree)"

index is also called as "a lookup table"

when we apply the index on a column

when we retrieve the data from table using indexed column,

it will never scan entire table, with help of indexing it will scan

only particular  data based on the given condition and using

index (with help of B+ trees)

example:

salary is having index

```
select firstname, lastname, email, salary from employee
where salary>=50000;
```

when we  do not apply the index on the column , for given

query with any condition, it will scan entire table

when we are creating the table, we will always use one column

should act as primary key column , when a column is acting

as primary key column is nothing but  "primary key index"

types of indexes in the MySQL:
========================

1.primary key index:
================

define the one column in the table as "primary key", then it

like as primary index in the table

example:
=======

```
create table sample(id int primary key);
```

2.unique index:
============

unique index means "when we are creating the unique index on

the column, the index will allow the only unique values, but

when we apply the unique index on the column, it may allow

the null values"

to create the unique index on the any column in the table,

in MySQL, we will use the following syntax:
=================================================

create unique index index_name on employee(column_name);

example:
=======
create unique index email_index on employee(email);
show index from employee;

3.composite index:
===============

when we apply the index two or more columns at a time, then

index is called as "composite index"

to create the composite index on the table, in MySQL we will

use the following syntax:
================================================

create index index_name on table_name(col1,col2,.....)

example:
=======

create index group_index on employee(mobileno,location);

show index from employee;

4.full text index:
=============

when  we apply the index on the "text column", then the index

is called as "full text index"

to create the full text index on the table, we will use the

following syntax, when we want to apply full text the column

type can be char, varchar:
=================================================

create full text index index_name on table_name(col1,...)

5.regular index:
============

when we apply the index on a single column, then the index

can be termed as "regular index"

to create the regular index on the any table we will use

following syntax in MySQL:

create index index_name on table_name(column_name)

6.spatial index:
============

when we apply the index on spatial data(geo graphical data)

column,  then the index is called as "spatial index"

to see the indexes on the table, in MySQL we will use the

following syntax:
=================================================

show index from table_name;

to remove the indexes from the any table in the MySQL, we will

use the following syntax:
=================================================

drop index index_name from table_name;

example:
=======

```
drop index email_index on employee;
drop index group_index on employee;
```

triggers in MySQL:
==============

trigger is a "database object" in the MySQL

trigger is a "an event" , which is happens automatically , when

are doing the following operations:
=================================================

1.insert:
===============================================

in MySQL, we can create the triggers with respect to insert

operation:

while doing the insertion operation, we can fire the trigger,

before insert ===>we can fire the trigger the before the

insertion of the record or row

after insert===>we can fire the trigger the after the

insertion of the record or row

2.update:
===============================================

in MySQL, we can create the triggers with respect to update operation:

while doing the updation operation, we can fire the trigger,

before update ===>we can fire the trigger the before the

updation of the record or row

after update===>we can fire the trigger the after the

updation of the record or row

3.delete:
===============================================

in MySQL, we can create the triggers with respect to delete

operation:

while doing the deletion operation, we can fire the trigger,

before delete ===>we can fire the trigger the before the

deletion of the record or row

after delete===>we can fire the trigger the after the

deletion of the record or row

syntax to create the trigger in MySQL:
=============================

```
create trigger trigger_name

before | after  insert | update | delete

on

table_name

for each row

begin

        -- here we write the code for trigger
end;
```

example:
=======

create the table with name "employees" in the employee_fp6_

2024:
=================================================

```
create table employess(id int primary key
auto_increment,
name varchar(100) not null,
salary float not null,
created_at datetime, updated_at datetime);
```

create the trigger for before insert operation:
===================================

example:
=======

```
DELIMITER $$
create trigger
before_trigger
before insert on employess
for each row
begin
  set NEW.created_at=now();
end;
insert into employess(id,name,salary)
values(1001,'raj',20000);
select * from employess;
```

create the trigger after the insertion:
=============================

create the employee_logs:
====================

```
create table employee_log(logid
int primary key
auto_increment,id int,
name varchar(100),
action_name varchar(100),
inserted_date datetime);
```

example:
=======

```
delimiter $$
create trigger
after_insert_trigger
after insert on employess
for each row
begin
   insert into
   employee_log(id,name,action_name,
   inserted_date)
   values(new.id,new.name,'insert',now());
end

insert into employess(id,name,salary)
values(1003,'kumar',25000);
select * from employee_log;
```

create the trigger before updation:
===========================

```
DELIMITER $$
create trigger salary_update_before_trigger
before update on employess
for each row
begin
   insert into employee_log(id,name,
   action_name,
   inserted_date)
   values(old.id,old.name,
```

```
   concat("salary changed from",
   old.salary,"to",new.salary),
   now());
end;

update employess set salary=50000
where salary=10000;
select * from employee_log;
```

create the trigger after the update:
===========================

```
DELIMITER $$
create trigger salary_update_after_trigger
after update on employess
for each row
begin
   insert into employee_log(id,name,
   action_name,
   inserted_date)
   values(old.id,old.name,
   concat("salary updated from ",
   old.salary," to ",new.salary),
   now());
end;


select * from employess;
update employess set salary=100000
where salary=20000;
select * from employee_log;
```

create the trigger before the delete:
============================

```
DELIMITER $$
create trigger user_delete_before_trigger
before delete on employess
for each row
begin
   insert into employee_log(id,name,
   action_name,
   inserted_date)
   values(old.id,old.name,
   concat(old.name," user is deleted"),
   now());
end;

select * from employess;
delete from employess where name="raj";
select * from employess;
select * from employee_log;
```

create the trigger after the delete:
===========================

```
DELIMITER $$
create trigger user_delete_after_trigger
after delete on employess
for each row
begin
   insert into employee_log(name,inserted_date)
```

```
   values(old.name,now());
end;


select * from employess;
delete from employess where name="ram";
select * from employess;
select * from employee_log;
```

normalaization:
============

this is a process and which is used to eliminate redundancy and

make data be stored more efficiently in the table

by applying the normalization:

we can avoid the Data redundancy

we can make data should be more accurate and consistent

using this we can also improve the overall query performance

also

types of normalization:
==================

1. First Normal Form (1NF):
=================

when we say the table is in 1 NF:

1) each attribute or column will only atomic values

2)each column has unique data

un-normalized table

| id | name | mobile_numbers |
|---|---|---|
| 1 | ram | 9875634567,8765432890 |

normalized table:
==============

| id | name | mobile_numbers |
|---|---|---|
| 1 | ram | 987563567 |
| 1 | ram | 8765432890 |

2.Second Normal Form(2NF):
=====================

when we say the table is in 2NF:
========================

1)first the table need to be in 1NF

2) remove the partial dependencies

example:
=======

| student_id | course_id | student_name | course_name | mentor |
|---|---|---|---|---|
| 1 | 101 | Dileep | java | xyz |
| 2 | 102 | rishi | java | xyz |
| 1 | 101 | Dileep | python | abc |

| sid | sname | |
|---|---|---|
| 1 | dileep | |
| 2 | rishi | |

<=== strudent

course table

| course_id | course_name |
|---|---|
| 101 | Java |
| 102 | python |

enrollment table:
==============

| sid | cid |
|---|---|
| 1 | 101 |
| 1 | 102 |
| 2 | 102 |

3)third normal form:
================

when we say the table is in 3NF:
=========================

1)first the table need to be in 2 NF

2) remove the transitive dependencies

| student_id | course_id | student_name | course_name | mentor |
|---|---|---|---|---|
| 1 | 101 | Dileep | java | xyz |
| 2 | 102 | rishi | java | xyz |

1 101 Dileep python abc

student table (student id, course id)

course table (coursid,coursename,instructorid)

Instructor table(instructor_id,instructor_name)

BCNF:(boyce-codded nofrmal form):
===========================

when we say the table in BCNF, the table must have the following:
==============================================

1)first table need to be in 3NF

2)every determinant is a candidate key

Fourth Normal Form:
=================

when we say the table in BCNF, the table must have the following:
==============================================

1)first table need to be in BCNF

2)eliminate the multi-value dependency into atomic

MongoDB:
========

mongo DB is used to store the data in the form of documents

where the document will have the data in the form of

key and value pairs

in mongoDB, the data actual form is "Json" form and when

store the data in the mongoDB, it will again convert into

BSON format

in mongoDB, we will have the data in the form of documents,

where the documents are part of "collection", where the

collection is part of the "database"

when we are working with mongoDB, we will use the following

commands:
=================================================

step-1:
======
mongod ===> to start the mongoDB server

step-2:

======
mongosh ===> to start the mongoDB shell

when we want to create the database in the mongoDB, we will

use following syntax:
==================================================

use database_name

when we create the database in the mongoDB, first it will look

into given database is there or not, if the database is there,

then it will make the database as current database,  if the

data base is not there, then it will create the database, make

the database as current database

to know the current database in the mongoDB, we will use

following command:
==================================================

db

to list the all available databases in the mongoDB, we will use

the following syntax:
==================================================

show dbs;

when we run the above command, it will only show  databases

which are having at least one collection in it

to create the collection, in the mongoDB database, we will use

the following syntax:
==================================================

db.createCollection("name of the collection")

here db refers "current database"

to list all collections in the current database, we will use

the following syntax:
=================================================

show collections

insert the data into mongoDB:
========================

insert into any data into mongoDB, we will use the following

functions:
==================================================

```
 1)insertOne()
 ===========

   using this function, we can insert only one document

 syntax:
 ======


db.collection_name.insertOne({key_name:value,key_name:value.....................
..})

example:
=======


db.employee.insertOne({name:"ram",location:"hyderabad",email:"ram@gmail.com",dep
tname:"developmenet"});

or

db.employee.insert({_id:10000,name:"ram",location:"hyderabad",email:"ram@gmail.c
om",deptname:"developmenet"});



to see the data in the mongoDB collection, we will use the

following syntax:
=================================================

                       db.collection_name.find()

example:
=======

 db.employee.find();

 2)insertMany():
=============

   using this function, we can insert only one or more

   documents in the mongoDB collection

syntax:
======

           db.collection_name.insertMany({},{},{},{}......)

example:
=======

> db.employee.insertMany([{_id:10003,name:"raj",location:"delhi"},
{_id:10004,name:"kalyan",location:"bangloore"}]);

 3)insert():
=========

   using this function, we can insert only one or more

   documents in the mongoDB collection
```

this is very old function in the mongoDB, better to use

insertOne() or insertMany()

example:
=======

```
db.employee.insert({_id:10000,name:"ram",location:"hyderabad",email:"ram@gmail.com",deptname:"developmenet"});

 db.employee.insert([{_id:10005,name:"kiran",location:"delhi"},
{_id:10006,name:"kumar",location:"bangloore"}]);
```

to update the data in the mongoDB, we will use the following

fucntions:
==================================================

1)updateOne():
============

this function can update only one document at a time for

given condition

syntax:

```
db.collection_name.updateOne({},{})
```

example:
=======

```
db.employee.updateOne({name:"ram"},{$set:{location:"hyderabad_new"}})
```

2)updateMany():
=============

this function can update only one or more documents at a time

for given condition

syntax:

```
db.collection_name.updateMany({},{})
```

example:
=======

```
 db.employee.updateMany({name:"ram"},{$set:{location:"hyderabad_new"}})
```

to delete the documents from the mongoDB, we will use the

following functions:
==================================================

1. deleteOne():
=================================================

this function can delete only one document at a time for

given condition

syntax:
======

```
db.collection_name.deleteOne(condition)
```

example:
=======

```
db.employee.deleteOne({name:"ram"});
```

2. deleteMany():
=============

this function can delete one or more documents at a time for given condition

syntax:
======

```
db.collection_name.deleteMany(condition)
```

example:
=======

```
db.employee.deleteMany({name:"ram"});
```

3.remove():
=========

this function can delete one or more documents at a time for given condition

syntax:
======

```
db.collection_name.remove(condition)
```

example:
=======

```
db.employee.remove({name:"raj"});
```

to remove the all documents from the mongoDB collection,

we will use following syntax:
================================================

```
db.collection_name.deleteMany({})
```

or

```
db.collection_name.remove({})
```

example:
=======

```
db.employee.deleteMany({});
```

or

```
 db.employee.remove({});
```

working with mongoDB collections data:
================================

get the employee details whose location is Hyderabad:
===========================================

```
db.employee.find({location:"hyderabad"})
```

get the employee details whose email is ram@gmail.com:
=============================================

```
 db.employee.find({email:"ram@gmail.com"})
```

get the employee details where get only name, email:
===========================================

```
db.employee.find({},{name:1,email:1,_id:0})
[ { name: 'ram', email: 'ram@gmail.com' } ]
 db.employee.find({},{name:1,email:1})
[ { _id: 10000, name: 'ram', email: 'ram@gmail.com' } ]
db.employee.find({},{name:1,email:1,_id:0})
[ { name: 'ram', email: 'ram@gmail.com' } ]
```

working with operators:
===================

comparision operators in the mongoDB:
================================

$eq, $ne, $gt, $gte, $lt, $lte

$eq (equal to)
===========

```
db.employee.find({email:{$eq:"ram@gmail.com"}});

db.employee.find({email:{$eq:"ram@gmail.com"}},{name:1,email:1,_id:0});
```

$nq(not equal to):
==============

```
db.employee.find({email:{$ne:"ram@gmail.com"}});

db.employee.find({email:{$ne:"ram@gmail.com"}},{name:1,email:1,_id:0});
```

$gt(greater than):
=====================

```
db.employee.find({salary:{$gt:40000}})
```

$gte(greter than or equal to):
========================

```
db.employee.find({salary:{$gte:40000}})
```

$lt(less than):
=====================

```
db.employee.find({salary:{$lt:40000}})
```

```
$lte(less than or equal to):
========================

db.employee.find({salary:{$lte:40000}})


logical operators:
==============

$or, $nor, $and, $not

$or:
====

syntax:

       db.collection_name.find({$or:[{},{}]})

get the employee salary is greter than 10000 or less than

50000:
================================================


 db.employee.find({salary:{$gt:10000}})

 db.employee.find({salary:{$lt:50000}})

  db.employee.find({$or:[{salary:{$gt:10000}},{salary:{$gt:50000}}]});})



$and:
====

syntax:

       db.collection_name.find({$and:[{},{}]})


example:
=======

db.employee.find({$and:[{salary:{$gt:10000}},{salary:{$gt:50000}}]});

$nor(opposite to or):
================

syntax:

       db.collection_name.find({$nor:[{},{}]})

example:
=======

 db.employee.find({$nor:[{salary:{$gt:10000}},{salary:{$gt:50000}}]});


$not:
=====

example:
=======
```

```
               db.employee.find({salary:{$not:{$lte:50000}}});
```

Array operators:
=============

$in, $nin, $all, $size

$in:
===

```
    db.employee.find({location:{$in:['hyderabad','delhi']}})
```

$nin:
====

```
        db.employee.find({location:{$nin:['hyderabad','delhi']}})
```

$all:
===

```
          db.employee.find({location:{$all:['hyderabad','delhi']}})
```

$size: (it looks for size of the array must be same as given
number)
=====

```
      syntax:  db.employee.find(filed_name:{$size:n})
```

example:
=======

```
        db.employee.find({techstack:{$size:3}})
```

sorting the mongoDB collection documents:
===================================

in order to sort the mongoDB documents, we will use a function

called "sort()"

syntax:

```
               db.collection_name.find().sort()
```

when we are using sort() function in mongoDB, we will use

1 ===>ascending order

-1 ===>descending order

example:
=======

sort the document based on the name:
==============================

```
db.employee.find().sort({name:1});

db.employee.find().sort({name:-1});
```

sort the salary data, display only salary data:
==================================

```
 db.employee.find({},{salary:1,_id:0}).sort({salary:-1});

 db.employee.find({},{salary:1,_id:0}).sort({salary:1});
```

limit():
=====

using this function we can limit number of documents in the

result

syntax:
======

```
                db.collection_name.find().sort().limit(n)
```

where n represents "number of documents"

example:
=======

get the only one document:
======================

```
 db.employee.find({},{salary:1,_id:0}).sort({salary:-1}).limit(1);
[ { salary: 90000 } ]
```

get the only 2 documents:
======================

```
 db.employee.find({},{salary:1,_id:0}).sort({salary:-1}).limit(2);
[ { salary: 90000 }, { salary: 50000 } ]
```

skip():
=====

this is used to "skip the number of documents in the mongodb

result"

syntax:
======

```
      db.collection_name.find().sort().limit().skip()
```

 example:
========

skip the first document:
===================

```
 db.employee.find({},{salary:1,_id:0}).sort({salary:-1}).limit(1).skip(1);
```

skip first two documents:
====================

```
 db.employee.find({},{salary:1,_id:0}).sort({salary:-1}).limit(1).skip(2);
```

python with mongoDB:
==================

in order to work with mongoDB using python, we need to install

a module called "pymongo"

to install pymongo, we will use the following syntax:
=========================================

pip install pymongo

after the installing the pymongo, we need to follow the

following steps:
=================================================

step-1:

create the mongoDB client using url of the mongoDB

step-2:

create the database using mongoDB client

step-3:

create the collection using mongoDB database

step-4:

perform the operations on mongoDB collections

(insert , data retrieval, update , delete)

example-1:
=========

```
#import the pymongo module
import pymongo as mongo
#create the connection between python and mongoDB using "MongoClient"
con=mongo.MongoClient("mongodb://localhost:27017/")
#create the database
db=con['sample_2024_fp6']#mongoDB database is "sample_2024_fp6"
#create the mongoDB collection with name "sample_new"
col=db['sample_new']
#list the all collections in the database called "sample_2024_fp6"
print(db.list_collection_names())
```

example-2:
==========

```
"""
how to see the list of avaliable database
in the mongoDB using python
"""
databaselist=con.list_database_names()
print(databaselist)
```

example-3:
=========

```
#how to see the list of collections are avaliable in the mongoDB database using
python
collectionlist=db.list_collection_names()
```

```
print(collectionlist)


example-4:
=========
#how to see the list of collections are avaliable in the mongoDB database using
python
collectionlist=db.list_collection_names()
print(collectionlist)

example-5:
=========

#insert the multiple documents at a time using insert_multiple()
d1=[{"name":"ram","location":"hyderabad","pincode":50123},
   {"name":"kiran","location":"chennai","pincode":58947},
   {"name":"hemanth","location":"delhi","pincode":565789}]
col.insert_many(d1)

example-6:
=========
#how to retrive the data from mongoDB collection using python
data=col.find_one()#here it fetch only one document
print(data)

example-7:
=========
#how to retrive the multiple documents from the mongoDB collection using python
data=col.find()#here it fetch multiple documents
for i in data:
    print(i)

example-8:
=========
#get the data from the mongoDB collection using, projection
data=col.find({},{'_id':0,'pincode':1,'name':1})#here it fetch multiple
documents
for i in data:
    print(i)

example-9:
=========
#apply the sorting and get the limited documents from mongoDB collection

#get only particular documents
data=col.find({},{'_id':0,'pincode':1}).sort({'pincode':1}).limit(3)
for i in data:
    print(i)

example-10:
==========

#update the document in the mongoDB collection using python
source={'pincode':565789}
target={'$set':{'pincode':500047}}
mycol.update_one(source,target)
data=mycol.find({},{'_id':0,'pincode':1})
for i in data:
    print(i)

example-12:
==========

#update the document in the mongoDB collection using python
```

```
source={'pincode':50123}
target={'$set':{'pincode':500047}}
mycol.update_many(source,target)
data=col.find()
for i in data:
    print(i)
```

example-13:
==========

```
#how to delete the data from mongoDB collection
mycol.delete_one({'name':'raj'})
data=mycol.find()
for i in data:
    print(i)
```

example-14:
==========

```
#how to delete the data from mongoDB collection
mycol.delete_many({})
data=mycol.find()
print(list(data))
for i in data:
    print(i)
```

in pymongo, we will use the following functions to perform the

operations on the mongoDB using python
======================================================

list_database_names()<=== to get all databases in the mongoDB

list_collection_names()<=== to get the all collection in the mongoDB

database

insert_one() <=== it used to insert only one document in the mongoDB

collection

insert_many()<=== it used to insert one or more documents in the

mongoDB collection

find_one()<===== it is used to get the only one document from the

mongoDB collection

find()<===== it is used to get the one or more documents from the

mongoDB collection

update_one()<=== it is used to update only one document in the

mongodb collection

update_many()=== it is used to update  one or more documents in the

mongodb collection

delete_one()<=== it is used to delete only one document in the

mongodb collection

delete_many()<= it is used to delete one or more documents from the

mongodb collection

examples:
========

db.sample2.aggregate([{$group:{_id:null,maxPrice:{$max:"$price"}}}])

db.sample2.aggregate([{$group:{_id:null,maxPrice:{$min:"$price"}}}])

db.sample2.aggregate([{$group:{_id:null,minPrice:{$min:"$price"}}}])

db.sample2.aggregate([{$group:{_id:null,avgPrice:{$avg:"$price"}}}])

db.sample2.aggregate([{$group:{_id:null,sumPrice:{$sum:"$price"}}}])

db.sample2.aggregate([{$group:{_id:null,uniquePrice:{$first:"$price"}}}])

how to remove the collection from the mongoDB:
======================================

to remove the collection from the mongoDB, we will use a

function called "dropCollection()"

syntax:
======

db.collection_name.drop()

example:
=======

db.employee.drop()

db.hr.drop()

how to remove the database in the mongoDB:
====================================

to remove the database in the mongoDB,we will use a function

called "dropDatabase()"

syntax:
=====

db.dropDatabase();

example:
=======
use sample2023;
db.dropDatabase();

when we perform the "dropDatabase()",it will remove the

all collections which are avaliable in the "mongoDB database".

mongoDB data types:
=================

string:
======
any text information in the mongoDB assume as "string"

string can be stored in the form of "single/double quotes"

integer:
======

numbers with no decimal or fractional part

example:
=======

1234,5678,90123,.........

double:
======

any number with decimal point/fractional part

examples:
========

1.234,5.678,9.01234,.............

boolean:
=======

any data either "true or false"

null data:
========

no data/no value can be stored in the  mongoDB as "null"

array:
=====

we can store data with multiple values in mongoDB field

example:

"names":["ram",'raj","kiran"]

object format:
===========

we can also  store a object  as data in the mongoDB field

syntax:
======

filed_name:{key1:value1,key2:value2,key3:value3,.....keyn:valuen}

example:
=======

"product_1":{id:123,name:"per123",category:"electronics"}

Capped Collection in MongoDB:
=========================

when we need to create the collection with spesifc memory

size and spesific number of documents,the mongoDB provide

a collection as "capped collection"

syntax:
======

```
db.createCollection("name of the collection",{capped:true,size:number of bytes,max:number of documents})
```

example:
=======

```
db.createCollection("sample2",{capped:true,size:10000,max:10000});
```

note:
====

fixed-size collections are also called as "capped collections"

in MongoDB

DBMS concepts(Normalization, Transaction control)
========================================

Query Questions:
==============

Retrieve all records from the employee table.

Select the emp_firstname and emp_lastname of all employees.

Find the emp_location of the employee with emp_id 1.
Display unique values of emp_blood_group.

Count the total number of employees in the table.

Retrieve employees where emp_gender is "Male".

List employees with emp_deptname as "Finance".

Select all employees who joined in 2020.

Find employees with a salary greater than 50,000.

Retrieve employees with emp_location as "Delhi".

List employees with emp_blood_group "O+".

Retrieve employees who joined after 2021-01-01.

Select employees born before 1990-01-01.

Retrieve employees with a salary between 60,000 and 80,000.

Find employees with emp_zipcode as 110001.

List employees whose emp_role is "Manager".

Select employees whose emp_deptid is 102.

Find employees with emp_lastname ending with "a".

Select employees with a salary less than or equal to 70,000.

Retrieve employees whose first name starts with "A".

Find employees with emp_blood_group "B+" or "A+".

Select employees who do not have "Lead" as their role.

Display employees born in 1991.

Retrieve employees with a emp_mobileno starting with 98.

List employees with an ID between 10 and 100.

Retrieve employees with a emp_location not equal to "Chennai".

Display employees who joined before 2021-12-31.

Select employees born between 1980-01-01 and 2000-12-31.

Retrieve employees whose emp_role is "Analyst".

Find employees with emp_salary exactly equal to 60,000.

List employees in the "IT" department.

Display employees who joined on a Monday.

Retrieve employees whose emp_gender is "Female" and whose salary is over 75,000.

Find employees whose emp_dob falls in December.

List employees whose first name contains "sha".

Retrieve employees with a salary greater than 85,000.

Find employees with the emp_deptname "Operations".

Retrieve employees who joined in the month of January.
Display employees whose emp_mobileno ends in "9".
Retrieve employees whose emp_lastname starts with "K".
Find employees with a salary greater than or equal to 100,000.
Select employees with emp_gender as "Male" and emp_blood_group as "O+".
List employees in the "Marketing" department.
Retrieve employees who joined in 2021 or later.
Display employees with emp_deptid 103.
Find employees whose emp_role contains "Developer".
Retrieve employees with emp_zipcode 400001 or 560001.
Find employees with an "AB+" blood group.
List employees whose emp_dob is after 1995-01-01.
Retrieve employees whose emp_role is "Executive".
Display employees with a emp_deptname "HR".
Select employees who joined on or after 2020-06-15.
Retrieve employees with a emp_mobileno starting with 91.
Find employees with a emp_location as "Pune".

List employees who have an emp_id greater than 50.
Display employees with an emp_zipcode not equal to 600001.
Retrieve employees who joined on 2020-01-10.
List employees whose emp_firstname is exactly 4 characters long.
Find employees born on a weekend.
Display employees with emp_role "Lead".
Retrieve employees with emp_blood_group "A+" or "B-".
Find employees with emp_salary between 80,000 and 90,000.
Display employees whose emp_dob falls in the 1990s.
Retrieve employees with emp_gender as "Female" and emp_blood_group as "A-".
List employees with a emp_location not in "Mumbai", "Delhi", or "Pune".
Retrieve employees with an emp_salary that is a multiple of 5,000.
Select employees whose first name ends with "n".
Retrieve employees who joined after 2021-01-01 but before 2022-01-01.
Find employees with emp_lastname containing "Singh".
List employees in "Finance" or "HR" departments.
Display employees with emp_role containing "Manager" or "Lead".
Retrieve employees whose emp_firstname starts with a vowel.
Find employees with a salary under 50,000.
Retrieve employees who joined before 2019-01-01.
Display employees with a emp_dob on 1990-05-15.
List employees with a emp_zipcode of 400001.
Retrieve employees with an even emp_id.
Select employees whose emp_mobileno contains "1234".
Display employees who joined on a holiday.
Find employees in the department with emp_deptid 105.
Retrieve employees with the emp_deptname "IT" or "Operations".
Display employees who joined in the last quarter of the year.
List employees who have "Manager" in their role.
Retrieve employees with a salary ending in 000.
Find employees with emp_location in either "Mumbai" or "Delhi".
Display employees who joined in July.
List employees with a salary exactly 75,000.
Retrieve employees born on or after 1991-01-01.
Find employees who do not have "Executive" in their role.
Display employees in the "Sales" department.
Retrieve employees whose first name is more than 5 characters.
Select employees with emp_role beginning with "D".
Display employees born in the 1980s.
Find employees with emp_zipcode greater than 500000.
Retrieve employees with emp_salary under 60,000 and emp_deptname as "Marketing".
List employees with emp_blood_group as "O+" and emp_role as "Analyst".
Retrieve employees with emp_firstname containing the letter "v".
Find employees who joined before 2021-06-30.
Display employees with emp_role ending in "ive".
List employees born in October.
Retrieve employees whose emp_salary is a multiple of 10,000.
Display employees with an emp_deptid of 104 or 105.
Retrieve employees who joined on the last day of a month.
Find employees who joined in an odd-numbered year.
Select employees whose emp_lastname contains three or more vowels.
Display employees with a salary of 65,000 or more.
Retrieve employees born on a public holiday.
List employees with emp_blood_group "AB+" and in "Operations".
Find employees with a emp_location in any capital city.
Retrieve employees whose emp_mobileno has no repeated digits.
Display employees with a emp_role containing "Data".
Select employees who joined on a specific day of the week.
Retrieve employees with emp_salary ending in "500".
Display employees who joined in the first week of January.
List employees with emp_deptname as "Legal" or "Audit".
Retrieve employees whose emp_salary does not end in zero.
Display employees with emp_dob falling on the 15th of any month.
Find employees with a emp_location outside the primary office location.

Retrieve employees whose emp_role is similar to "Admin".
List employees with emp_blood_group starting with "O". ...

Retrieve employees with emp_salary greater than or equal to 100,000.
List employees whose emp_location is either "Hyderabad" or "Kolkata".
Find employees with a emp_dob in the month of April.
Display employees with emp_role containing "Assistant".
Select employees with emp_zipcode in the range 100000 to 400000.
Retrieve employees who joined in the last quarter of any year.
List employees whose emp_firstname begins and ends with the same letter.
Find employees with emp_deptname starting with "M".
Display employees with a emp_salary that is a multiple of 15,000.
Select employees who have a "J" in their emp_lastname.
Retrieve employees with an odd emp_id.
List employees who have "Engineer" in their emp_role.
Find employees whose emp_firstname is more than 6 characters long.
Display employees who joined before 2018-12-31.
Select employees with emp_blood_group "B-" or "A+".
Retrieve employees with emp_location ending in "pur".
List employees whose emp_mobileno contains "456".
Find employees who joined on a leap day.
Display employees with a emp_zipcode ending in "5".
Select employees with emp_deptid in the range 100 to 105.
Retrieve employees with emp_dob on the first of any month.
List employees with a salary greater than 95,000.
Display employees whose emp_firstname includes the substring "deep".
Retrieve employees who joined after 2019-05-01 and before 2021-05-01.
Find employees in the "Research" department.
Select employees with a emp_role beginning with "A".
Retrieve employees with emp_blood_group "O-" and emp_gender "Male".
List employees with emp_zipcode between 500000 and 600000.
Display employees with emp_firstname starting with "P".
Find employees with emp_mobileno ending in "123".
Retrieve employees who joined in the year 2022.
List employees whose emp_lastname contains the letter "d".
Display employees who have a "Lead" position in "IT".
Select employees with emp_blood_group "A+" or "B+".
Retrieve employees with a salary less than 70,000.
List employees born on a Friday.
Find employees with an even-numbered emp_id.
Display employees with emp_dob on the last day of any month.
Retrieve employees with a emp_role ending in "st".
Find employees who joined in an odd-numbered month.
List employees with a emp_salary below 45,000.
Retrieve employees whose emp_location contains "nagar".
Display employees with emp_firstname containing the substring "Raj".
Select employees who joined after 2015-01-01.
Find employees with emp_deptname as "Compliance".
Retrieve employees with a emp_zipcode below 200000.
Display employees whose emp_blood_group is not "AB+".
List employees with emp_role starting with "T".
Find employees with emp_lastname ending with "i".
Retrieve employees with emp_salary divisible by 25,000.
Display employees whose emp_gender is "Female" and emp_salary above 60,000.
Select employees with emp_dob on the 10th of any month.
Find employees in the "Support" department.
Retrieve employees with emp_location starting with "Ch".
List employees whose emp_mobileno starts with "99".
Find employees with a emp_role containing "Specialist".
Display employees who joined before 2010-01-01.
Retrieve employees whose emp_firstname includes an uppercase "R".
List employees with emp_blood_group "O+" and a salary over 80,000.
Find employees with emp_deptid less than 120.

Display employees with emp_role in "Human Resources".
Retrieve employees with an emp_id that is a multiple of 5.
List employees with a emp_zipcode in the 400000 range.
Display employees born on a public holiday.
Find employees with emp_location as "Noida".
Retrieve employees with a salary under 65,000.
List employees with emp_role containing the substring "Coord".
Display employees with an emp_lastname beginning with "Th".
Retrieve employees with emp_blood_group "A-" or "AB-".
Select employees whose emp_mobileno ends with "4321".
Find employees with emp_firstname shorter than 4 characters.
Display employees whose emp_deptname is "Engineering".
Retrieve employees with emp_zipcode starting with 5.
List employees with emp_gender as "Male" and emp_blood_group "B+".
Find employees with a emp_salary above 95,000.
Select employees whose emp_role is "Technician".
Retrieve employees with emp_lastname containing "s".
Display employees who joined between 2017-01-01 and 2018-12-31.
List employees in the department with emp_deptid 106.
Find employees with an even-numbered emp_id.
Display employees whose emp_salary is exactly 75,000.
Retrieve employees with emp_dob in the month of May.
List employees with emp_firstname beginning with "R".
Select employees with a salary of 90,000 or above.
Retrieve employees with a emp_role of "Administrator".
Display employees whose emp_deptname is "Audit".
List employees with a emp_mobileno starting with "987".
Find employees with emp_zipcode not equal to 600004.
Retrieve employees who joined in 2015.
Select employees with emp_blood_group "B+" or "O+".
Find employees who do not have "Admin" in their role.
Display employees whose emp_role contains the letter "i".
Retrieve employees with emp_location in the northern region.
List employees with emp_deptid over 100.
Display employees with emp_salary exactly 80,000.
Retrieve employees with emp_firstname ending in "ya".
List employees with emp_dob on the 31st of any month.
Display employees with emp_gender "Female" and a emp_role containing "Tech".
Retrieve employees whose emp_lastname begins with "V".
Find employees with a salary less than 50,000.
Display employees with emp_deptname in "Operations" or "Support".
Retrieve employees with an emp_mobileno containing the sequence "789".
List employees born in a leap year.
Select employees with a emp_role of "Consultant".
Find employees with emp_zipcode as 400001 or 600001.
Display employees with a emp_blood_group starting with "AB".
Retrieve employees whose emp_firstname has an odd number of letters.
List employees who joined in April.
Display employees with an even-numbered emp_id.
Find employees with emp_gender as "Male" and emp_location as "Chennai".
Retrieve employees with a emp_deptid greater than 110.
List employees with emp_salary between 75,000 and 85,000.
Display employees with emp_role starting with "S".
Retrieve employees whose emp_firstname has more than one word.
List employees in "Sales" or "Marketing".
Find employees with emp_dob on the last day of a month.
Retrieve employees with emp_mobileno starting with "91".
Display employees who joined in the first quarter.
Find employees with emp_salary ending in "50".
List employees with emp_role beginning with "C".
Retrieve employees in "Human Resources".
Display employees with emp_location as "Ahmedabad".
List employees with emp_blood_group "O+" and a salary above 75,000.
Retrieve employees with an emp_id divisible by 10.

Select employees who have "Lead" in their emp_role.

Find employees with emp_dob on a Saturday.

Display employees whose emp_firstname has fewer than 3 vowels.

List employees with emp_salary not in multiples of 10,000.

Retrieve employees with emp_zipcode ending in "10".

React with JS:
===========

react is a "JavaScript Library"

pre-requisites for React:
===================

1. we need to install the node js (once we install node js, we will

get NPM(node package manager))

2.to create the react project, we need to use the following

syntax:

```
npx create-react-app app_name or project_name;
```

once we create the project, we need to move into the project

```
cd    project-name
```

launch the project in the vs code editor

```
code .
```

when we create the any react app or project, we will get the

following project structure:
=================================================

```
app_name (root directory)

    node_modules

    public folder

    src

   .git

    package.json

    package-lock.json

    readme.md
```

using react, we can able to create the "any complex UI of

application"

using react, we can able to create the "SPA's" , where SPA

means "Single Page Application"

using react, we can able to see the any changes very fast

without refreshing the webpage

react uses a "component-based " structure, to design the entire

any web application UI.

using components In the any react app, component can be

maintained very easily (it means any changes we can do

component alone itself, because of this, any other components

will never effected)

in react, we will have concept called "Virtual DOM"

every react project always runs in the "Server"

in order to run the react project, we will use the following

syntax:

npm start

JavaScript ES6 concepts and other concepts:
==================================

JavaScript variables(using let, var, const)

example:
=======

```
//create the variables
var a=10;
let b=1.234;
const s1="hello"; //constant
document.write(a+"<br>");
document.write(b+"<br>");
document.write(s1+"<br>");
//in react, we will use console.log(),but not document.write()
console.log(a)
console.log(b)
console.log(s1)
```

JavaScript operators:
================

example-1:
=========

```
/* working with operators:
  airthmetic,relational,logical,
  assignment,bitwise,conditional,
  increment and decrement*/
  let a=10,b=20;
  //airthmetic operators
  console.log(a+b);//30
  console.log(a-b);//-10
  console.log(a*b);//200
  console.log(a/b);//0.5
  console.log(a**b);//100000000000000000000
  console.log(a%b);//10
  //relational operators
  console.log(a>b);//false
  console.log(a<b);//true
  console.log(a>=b);//false
  console.log(a<=b);//true
  console.log(a==b);//false
```

```
  console.log(a!=b);//true
  //logical operators
  console.log((a>b)||(a<b));//logical or true
  console.log((a>b)&&(a<b));//logical and false
  console.log(!(a>b));//logical not true
```

example-2:
=========

```
/* working with operators:
  airthmetic,relational,logical,
  assignment,bitwise,conditional,
  increment and decrement*/
  let a=10,b=20;
  //bitwise operators
  console.log(a|b)
  console.log(a&b)
  console.log(a^b)
  console.log(a<<2)
  console.log(a>>2)
  console.log(~a)
  //conditional operators
  result=(a>b)?a:b
  console.log("the value of the result is:"+result)
  result=(a<b)?a:b
  console.log("the value of the result is:"+result)
  //increment and decrement operator
  console.log(++a)//11
  console.log(b++) //21
  console.log(--a)//10
  console.log(b--) //20
```

JavaScript conditional:
=================

example:
=======

```
// working with conditional statements
a=10
b=20
if(a>b)
{
    console.log("a is greater than b")
}
else
{
    console.log("a is less than b")
}
```

example-2:
=========

```
// working with conditional statements
a=10
b=20
c=30
if((a>b)&&(a>c))
{
    console.log("a is greater than b and c")
}
```

```
else if((b>c))
{
    console.log("b is greter than a and c")
}
else
{
    console.log("c is greter than a and b")
}
```

JavaScript unconditional:
===================

javascript loops:
=============

example-1:
=========

```
// working with looping statements
a=1
while(a<=10)
{
    console.log(a);
    a+=1
}
```

example-2:
=========

```
// working with looping statements
a=1
do
{
    console.log(a);
    a+=1
}while(a<=10)
a=100
do
{
    console.log(a);
    a+=1
}while(a<=10)
```

example-3:
=========

```
// working with looping statements
for(var a=1;a<=10;a++)
{
    console.log(a);
}
```

JavaScript arrays, strings and objects:
=============================

example-1:
=========

```
// working with arrays
var a1=[1,2,3,4,5,6,"hello","hai",1.234,5.678]
var a2=new Array(10,20,30,40,50)
console.log(a1)
```

```
console.log(a2)
//working with string
var s1="hello world"
var s2=new String("Hello World")
console.log(s1)
console.log(s2)
```

exmaple-2:
=========

```
// working with objects
var a1={name:"ram",location:"hyderabad",salary:10000,email:"ram@gmail.com"}
console.log(a1)
console.log(a1.name)
console.log(a1.location)
console.log(a1.salary)
console.log(a1.email)
```

JavaScript functions and arrow functions:
================================

example-1:
=========

```
// working with fucntion
function display()
{
    console.log("this is my first function in JS")
}
function display2(a,b)
{
    console.log("the value of the a is:"+a);
    console.log("the value of the b is:"+b);
}
function display3(a,b,c)
{
    return a+b+c;
}
display()
display2(10,20)
result=display3(10,20,30)
console.log(result)
```

javaScript Events

JavaScript DOM

working with react project:
=====================

step-1: once we create the project, we will remove the all

files in the react folder

step-2:  create an empty file with name "index.js" in the src

folder

step-3: import the following modules in the index.js

```
//import the two important modules
 import React from "react";
 import ReactDOM from "react-dom/client"
```

step-4:

create the html element using js

```
const myheading=<h1>this is heading-1 from index.js</h1>
```

display on  html content on the webpage

```
const root=ReactDOM.createRoot(document.getElementById('root'));
root.render(myheading)
```

example-1:
=========

display the paragraph tag in react app
==============================

```
import React from 'react'
import ReactDOM from 'react-dom/client'
//create the html element using js
const paragraph=<p>this is a paragraph from index.js</p>
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the html content
root.render(paragraph)
```

exmaple-2:
=========

display the division tag data on html page in react app
=============================================

```
import React from 'react'
import ReactDOM from 'react-dom/client'
//create the html element using js
const mydiv=(<div>
    <h1>this is h1 is from the division</h1>
    <h1>this is h1 is from the division</h1>
    <h1>this is h1 is from the division</h1>
    <h1>this is h1 is from the division</h1>
    <h1>this is h1 is from the division</h1>
    <h1>this is h1 is from the division</h1>
    </div>)
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the html content
root.render(mydiv)
```

in react, to write the html , we will use a format called "JSX"

JSX stands for "javascript xml"

using jsx format, we can write the any html code in the react

in the react, we will never write the html code directly, for that

we will write the html code, using JSX format

example:
=======

```
const myhtmlelemnt=htmlcode in tags
```

when we are writing the multiple html lines, we will use ()

const myhtmlelement=(htmlcode)

example-3:
=========

display the html table in the react app:
==============================

```
import React from 'react'
import ReactDOM from 'react-dom/client'
//create the html element using js
const mytable=(
    <table>
        <tr>
            <th>col1</th><th>col2</th><th>col3</th><th>col4</th>
        </tr>
        <tr>
            <td>data1</td><td>data2</td><td>data3</td><td>data3</td>
        </tr>
        <tr>
            <td>data1</td><td>data2</td><td>data3</td><td>data3</td>
        </tr>
        <tr>
            <td>data1</td><td>data2</td><td>data3</td><td>data3</td>
        </tr>
    </table>
)
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the html content
root.render(mytable)
```

example-4:
=========

display the swapping of the two numbers in the html page

using react

=================================================

```
import React from 'react'
import ReactDOM from 'react-dom/client'
let a=10,b=20;
let temp=a;
a=b;//20
b=temp;//10
//create the html element
const myh1=<h1>the value of the a is: {a} and b is :{b} </h1>
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the html content
root.render(myh1)
```

in react,  the entire application development can be done in

the form of several components

each component can created in the react either  using

function or class

in react, basically we will have two types of components:
============================================

1. functional component (functional component means creating

the component using function in the react app)

2.class component(class component means creating the

component using class in the react)

note:
====

while creating the functional or class component, we will

always use "uppercase" letter in the component name starting

working with functional components:
=============================

step-1:

create a function with some name and name must starts with uppercase letter

step-2:

return the html content

step-3:

while using in the render function give as follows

<function_name/>

example:
=======

```
import React from 'react'
import ReactDOM from 'react-dom/client'
//create the functioncal component
function Myh1()
{
    return <h1>this is heading-1</h1>
}
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the component
root.render(<Myh1/>)
```

example-2:
=========

```
import React from 'react'
import ReactDOM from 'react-dom/client'
//create the functioncal component
function Myh1()
{



    return (
    <>
    <h1>this is heading-1</h1>
    <h1>this is heading-1</h1>
    <h1>this is heading-1</h1>
    <h1>this is heading-1</h1>
    </>)
}
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the component
root.render(<Myh1/>)
```

example-3:
=========

calling one functional component in another functional

component
=================================================

```
import React from 'react'
import ReactDOM from 'react-dom/client'
//create the functioncal component
function Myh1()
{

    return (
    <>
    <h1>this is heading-1</h1>
    <h1>this is heading-1</h1>
    <h1>this is heading-1</h1>
    <h1>this is heading-1</h1>
    </>)
}
function Myh2()
{

    return (
    <>
    <Myh1/>
    <h1>this is heading-1</h1>
    <h1>this is heading-1</h1>
    <h1>this is heading-1</h1>
    <h1>this is heading-1</h1>
    </>)
}
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the component
root.render(<Myh2/>)
```

example-4:
=========

creating the functional component in the separate file
===========================================

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import Myh1 from './Myh1.js'
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the component
root.render(<Myh1/>)
```

example-5:
=========

inline css:
========

```
function Myh1()
{
 return <h1 style={{color:'red'}}>this is from h1</h1>
}
export default Myh1;
```

inline css:
========

```
import Myh1 from './Myh1.js'
function Myh2()
{
    return (
    <>
    <Myh1/>
    <h1 style={{color:`red`}}>this is function component from the file called
Myh2.js</h1>
    </>)
}
export default Myh2;
```

example-6:
=========

```
import React from 'react'
import ReactDOM from 'react-dom/client'
//create the class component
class Myclass extends React.Component
{
    render()
    {
        return <h1> this my first class component in react</h1>
    }
}
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(<Myclass/>)
```

example-7:
=========

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import Myclass from './Myclass'
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
```

```
root.render(<Myclass/>)
```

working with react events:
=====================

example-1:
========

```
import React from 'react'
import ReactDOM from 'react-dom/client'
//create a function,which will called when we done any event
function myclick()
{
    alert("i am called!")
}
//create the html element
const myelement=<input type="button" value="click" onClick={myclick}/>
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(myelement)
```

example-2:
=========

```
import React from 'react'
import ReactDOM from 'react-dom/client'
//create a  functional component
function Mybutton()
{
    //create a arrow function , to write the code for event
    const myevent=()=>{
        alert("you click the button!")
    }
    return <button onClick={myevent}>click</button>
}
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(<Mybutton/>)
```

react props:
=========

props are nothing but "arguments(data) to the react components"

these are look like " function arguments in the JavaScript or attributes

in html"

create a functional component as follows

```
function  name_of_the_function(props)
{
        //code
}
```

calling the function component with props values as follows:
===============================================

```
<function_name attr1="value attr2="value2" />

example-1:
=========
import React from 'react'
import ReactDOM from 'react-dom/client'
//create a  functional component
function Mybutton(props)
{
   return (
    <>
          <h1> the name of the employee is:{props.name}</h1>
          <h1> the email of the employee is:{props.email}</h1>
    </>
   )
}
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(<Mybutton name="ram" email="ram@gmail.com"/>)


example-2:
=========
import React from 'react'
import ReactDOM from 'react-dom/client'
//create a  functional component with  name Myprops
function MyProps(props)
{
   return (
    <>
          <h1> the name of the employee is:{props.name}</h1>
          <h1> the email of the employee is:{props.email}</h1>
    </>
   )
}
//create a  functional component with  name Myprops2
function MyProps2()
{
    return <MyProps name="ram" email='ram@gmail.com'/>
}

//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(<MyProps2/>)


exmaple-3:
=========

import React from 'react'
import ReactDOM from 'react-dom/client'
//create a  functional component with  name Myprops
function MyProps(props)
{
   return (
    <>
          <h1> the name of the employee is:{props.name}</h1>
          <h1> the email of the employee is:{props.email}</h1>
    </>
   )
}
//create a  functional component with  name Myprops2
```

```
function MyProps2()
{
    const name="ram"
    const email="ram@gmail.com"
    return <MyProps name={name} email={email}/>
}

//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(<MyProps2/>)
```

React conditioning or React conditionals:
================================

example-1:
=========

```
import React from 'react'
import ReactDOM from 'react-dom/client'
//create a  functional component with  name MyComp1
function MyComp1()
{
   return(
    <>
    <h1 style={{backgroundColor:"red"}}> this is component-1</h1>
    </>
   )
}
//create a  functional component with  name MyComp2
function MyComp2()
{
    return(
        <>
        <h1 style={{backgroundColor:"green"}}> this is component-2</h1>
        </>
       )

}
//create the another component as follows
function MyComp3(props)
{
    if(props.value==1)
    return <MyComp1/>
    else
    return <MyComp2/>

}

//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(<MyComp3 value="2"/>)
```

working with lists using React:
========================

```
import React from 'react'
import ReactDOM from 'react-dom/client'
//create a  functional component with  name MyComp1
function MyListItemComp(props)
{
   return <li>{props.myname}</li>  //returns always list item
```

```
}
//create a  functional component with  name MyComp2
function MyListComp()
{
     //create the arrays of names
     const names=["ram","kumar","karan","rajat","sunil"]
    return(
        <>
         <ul>

            {names.map((name) => <MyListItemComp myname={name}/>)}

         </ul>
        </>
       )

}

//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(<MyListComp/>)
```

## example-2:

```
import React from 'react'
import ReactDOM from 'react-dom/client'
//create a  functional component with  name MyComp1
function MyListItemComp(props)
{
   return <li>{props.myname}</li>  //returns always list item
}
//create a  functional component with  name MyComp2
function MyListComp()
{
     //create the arrays of names
     const names=["ram","kumar","karan","rajat","sunil"]
    return(
        <>
         <ol type="A">

            {names.map((name) => <MyListItemComp myname={name}/>)}

         </ol>
        </>
       )

}

//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(<MyListComp/>)
```

## working with Tables using React:

```
import React from 'react'
import ReactDOM from 'react-dom/client'
//create a  functional component with  name MyComp2
function MyTableData()
```

```
{
     //create the data for tables
     const data=[
      {id:101,name:"Ram",location:"hyderabad",email:"ram@gmail.com"},
      {id:101,name:"Ram",location:"hyderabad",email:"ram@gmail.com"},
      {id:101,name:"Ram",location:"hyderabad",email:"ram@gmail.com"},
      {id:101,name:"Ram",location:"hyderabad",email:"ram@gmail.com"},
      {id:101,name:"Ram",location:"hyderabad",email:"ram@gmail.com"},
      {id:101,name:"Ram",location:"hyderabad",email:"ram@gmail.com"}
     ]

    return(
        <>
        <table style={{border:"2px",borderColor:"green"}}>
         <thead>
            <tr style={{border:"2px",borderColor:"green"}}>
               <th>id</th><th>name</th><th>location</th><th>email</th>
            </tr>
         </thead>
         <tbody>
           {data.map((row)=>(
            <tr>
               <td>{row.id}</td>
               <td>{row.name}</td>
               <td>{row.location}</td>
               <td>{row.email}</td>

            </tr>
           ))}
         </tbody>
        </table>

        </>
       )

}

//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(<MyTableData/>)
```

working with State:
================

state is used with in the component and state is represent a

value or data

the value or data of the state will changes any point of time

using useState in react

example:
=======

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import { useState } from 'react'
//create a  functional component with  name MyComp2
function MyStateData()
{
    const [state, setState]=useState(0)
```

```
    //create an arrow function with name "increment()"
    const increment=()=>{
       setState(state+1)

    }
    //create an arrow function with name "decrement()"
    const decrement=(data)=>{
      setState(state-1)

    }
    return(
      <div>
         <h1>State Value:{state}</h1>
         <button onClick={increment}>increment</button>
         <button onClick={decrement}>decrement</button>
      </div>
    )

}
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(<MyStateData/>)
```

working with Hooks and Forms using React:
==================================

hooks are "functions in react" and using "hooks we can manage

the state of the component"

because the hooks," we need to create the class components"

in React application

in react, we will have the following components:
=====================================

1.useState:
================================================

using this,

we can add the "state to the functional component"

in order to work with react "useState" hook, we will use the

following syntax:
================================================

```
import { useState} from 'react';
```

example:
=======

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import { useState } from 'react'
//create a  functional component with  name MyComp2
function MyStateData()
{
   const [data,setData]=useState(10)
```

```
   return(
    <div>
      <h1>mydata:{data}</h1>
      <button onClick={()=>setData(data+10)}>addby10</button>
    </div>
   )

}
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(<MyStateData/>)
```

2.useEffect
==============================================

it is used for side effects like "data fetching, any changes in the

DOM"

in order to work with react "useEffect" hook, we will use the

following syntax:
================================================

```
import { useEffect} from 'react';
```

example:
=======

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import { useState,useEffect } from 'react'
//create a  functional component with  name MyComp2
function MyStateData()
{
   //set the seconds value as zero
   const[seconds,setSeconds]=useState(10)
   useEffect(
    ()=>{
      const interval=setInterval(()=>
      {
        setSeconds((prev)=>prev+1);
      },3000) //for every 1 sec
      return ()=>clearInterval(interval); //for any refresh it will state as
initial value
    },[])
    return <h1> timer: {seconds}</h1>
}
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(<MyStateData/>)
```

3.useContext:
===========

allows the components to consume the context directly without needing to pass  the props at calling or entry level

in order to work with react "useContext" hook, we will use the

following syntax:
================================================

```
import { useContext} from 'react';
```

example:
=======

```
import React, { createContext } from 'react'
import ReactDOM from 'react-dom/client'
import { useState,useEffect,useContext } from 'react'
//create the user context  using createContext()
const userContext=createContext()
//create a  functional component with  name MyComp2
function ShowUser()
{
   const user= useContext(userContext)
   return <h1> my username:{user}</h1>
}
function DisplayContextData()
{
  return(
    <userContext.Provider value="ram">
      <ShowUser/>
    </userContext.Provider>
  )
}
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(<DisplayContextData/>)
```

4.useRef:
=================================================

```
import React, { createContext } from 'react'
import ReactDOM from 'react-dom/client'
import { useEffect,useRef } from 'react'
//create the functional component
function InputFocus()
{
  const inputRef=useRef(null);
  useEffect(()=>{
    alert("you keep cursor in text box")
    inputRef.current.focus();
  },[])
  return <input ref={inputRef} type="text" placeholder='focus changes'/>
}
//create the root element
const root=ReactDOM.createRoot(document.getElementById('root'));
//render the class component
root.render(<InputFocus/>)
```

5.useReducer :
=================================================

it used to manage the any "complex state logic also using  use reducer"

6.useMemo:
==================================================

it is used to optimize the performance by memorizing

expensive calculations

7.useCallback
===================================================

it is going to return "the memorized version of a function to

prevent the un-necessary re-renders"

Flask and Django in Python:
======================

flask is a light-weight frame work for developing the small to

medium scale web applications

in order to work with flask in python, we need to install flask

in our system using following syntax:
=================================================

pip install flask

Django is a web framework for developing the small to large

scale web applications and enterprise applications

in order to work with django in python, we need to install

Django in our system using following syntax:
=================================================

pip install Django

in real time , when we are developing any application, we will

use the following technologies:

UI tech:  html, css, javascript

server-side:  python

databases: MySQL

when we are developing the application using any framework,

the entire application can be developed using architectures

/ design patterns are :

MVC (Model - View - Controller)

MVC:
====

model:  it is related to "database"

example:  MySQL, oracle, postgres sql,............

view:    it is related to "UI of the application"

example:  html, css, javascript

controller : it is related to "Server -side"

example:  java |  python | php ........

in this design pattern,

entire view and  model can be controlled by the "controller" this is "tighly-coupled" architecture

example frameworks, which are uses MVC archicture means:

Spring framework,  Angular framework, code igniter ,

Ruby on Rails

MVT(Model - View - Template):
========================

model:  it is related to "database"

example:  MySQL, oracle, postgres sql,............

view:    it is related to "server"

example: python

templates : it is related to "UI of the application"

example: html, CSS, JavaScript

example frameworks, which are uses MVT archicture means:

Django

in MVT model/ architecture, both view and template can control

overall application for any changes or any updates

this is "loosely-coupled" architecture

note:
====

flask never uses any pre-defined model like spring or Django,

in flask , if we are developing any application, we can able to

use either "MVC" or "MVT" .

when we we want to create the Django projects with Django,

we need to have the following:
================================================

1) python need to installed

python --version

or

py  --version

2) Django need to installed

django-admin --version

3) virtualenv need to installed

4) we need to install sql related libraries

how we install python libraries:
========================

when we want to install any python library, we will use python

package installer program  called "PIP"

pip stands for "python installs packages" and using this

1) install the any python package or library

2) upgrade the any python package or library

3) uninstall the any python package or library

how to install python package in python using pip:
========================================

pip install package_name

example:  pip install numpy

how to install python package in python with specific version

using pip:
================================================

pip install pakcage_name==version_number

example: pip install numpy==1.0

how to install multiple python packages using pip in python:
===============================================

pip install package_name1 package_name2 package_name3...

when we want to the all available python packages in

system or computer using pip, we will use the following

syntax:
================================================

pip list

or

pip freeze

when we want to upgrade the any python package, we will use

the following syntax using pip:
=================================================

pip install --upgrade package_name

example:

pip install --upgrade Django

pip install --upgrade pip

when we want to remove any python package or uninstall

any python package, we will use following syntax using pip:
===============================================

pip uninstall pakcage_name

example:
=======

    pip uninstall numpy

to see the any package information using pip, we will use the

following syntax:
================================================

pip show package_name

example:
=======

pip show pandas

when we are working with any Django projects, we will always

create the "virtual environment"

to create the virtual environment for Django project, we will

use  a package called "virtualenv"

to install virtualenv, we will use the following syntax
===========================================

    pip install virtualenv

to create the virtual enviornment, we will use the following

syntax:
==============================================

    python -m venv enviorment_name

example:

    python -m venv myenv

after we are creating the virtual environment, we will install the

following:
=================================================

    Django

    mysqlclient

to check Django is installed or not, we will use the following

syntax:
==================================================

```
django-admin --version
```

before we are creating the any Django project, we need to

activate the virtual environment

```
virtual_envionemnt_name\Scripts\activate.bat
```

example:
=======

```
myenv\Scripts\activate.bat
```

when we want to create the Django project, we will use the

following syntax:
==================================================

```
django-admin startproject project_name
```

create a django project with name "myproject" :
======================================

```
django-admin startproject myproject
```

when we create the Django project, in the Django project we

will get the following:
==============================================

a  subfolder with the project_name

manage.py

in the project sub folder ,we will have the following files:
=============================================

1)__init__.py ===> it is a empty python file and it represents

the sub folder as package

2) asgi.py  (asgi ==> asynchronous gateway interface)

this is file is used to handle asynchronous requests

and it used to work with web sockets

3) settings.py:
==================================================

this file generally we will used to give the settings related to:

==================================================

1. about apps

in the settings.py, we will have a section with name

"installed_apps"

example:
=======

```
   INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'home',
    'login',
    'signup',
    'contact',
]
```

2. about templates folder(html folder)
==================================================

templates folder is a folder of "all html files of  the Django

project"

in Django project , we want to create any html file, we need to

create  the html file inside the "templates" folder

to give the settings in the "settings.py" of the django project,

in settings.py, we will have a section called "TEMPLATES"

in that we give the settings of the project templates folder as

following:

```
               'DIRS': [os.path.join(BASE_DIR,'templates')]
```

example:
=======

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR,'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
```

```
    },
]
```

3. about static folder:
=================

in Django project, we will use the static folder, to store the files

related to css files, images, js files

in the static folder, again we will create the separate folders,

    1) css folder

    2) images folder

    3) js folder

using the following way, we can settings related "static" folder

inside the settings.py

```
STATICFILES_DIRS=[os.path.join(BASE_DIR,'static')]
```

note:
====

template file refers to "html file"

static file refers to "css, image, js file"

4. about  media folder
==================

media folder refers to " stores any user uploaded any

document, audio, video, image" in the Django application

when we want to work with media folder, this folder related

settings  also we need to create inside the "settings.py" of the

Django project

```
MEDIA_URL='media/'
MEDIA_ROOT=os.path.join(BASE_DIR,'media')
```

5. about database
=================================================

in Django, we will have a default database called "sqllite"

when we create the project, Django already give the settings for

the "Sqllite"

when we are working with database other than SQLite, then

we give the database settings too

Django can able to work with databases like  MySQL, oracle,

postgres sql,...............

Django can not able to work with no-sql databases directly

like mongoDB

for MySQL, we will give the settings in the settings.py as follows:
=================================================

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'database_name',
        'USER':'root'  #mysql user name
        'PASSWORD':'123456789', #mysql password
    }
}
```

     6. about any middleware
=================================================

middleware is a framework-level hook

using this we are going to process the all http requests and

http responses, before view or after view

generally using middle ware we will do the following jobs in the

Django:
=================================================

1.request processing:
=================

using the middleware  " we can able to modify or validate the

http request before it reaches to the view"

2.response processing:
===================

using the middleware  " we can able to modify or validate the

http response after it  leaves view"

3.cross-cutting concerns:
====================

using middleware, foer every request, we need to apply the

following :

1. securtity using csrf_token or csrf protection

2. Authentication

3.Logging

4. Session Management

5.Caching

in the settings.py, we will have the a section called MiddleWare,

there we will define any middle ware to add to the Django

project

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

7. about security:
=================

using CSRF tokens we can add the  security to each requests

and also while Authentication of the users, while transferring

data over the application
=================================================

4) urls.py :
==============================================

in Django, urls.py is called as "Router or URL Dispatcher"

using urls.py, we can able to give the "url" for each view

in the application, using this url, we can able to call the any

view in the application and also over the server

5) wsgi .py (wsgi stands for web server gateway interface)

when we are working with Django projects, we need to create

the apps in the Django project:
================================================

before we creating the app, we need to move into

project folder

```
cd myproject
```

create the Django app in the project as follows:
========================================

```
python manage.py startapp app_name
```

example:
=======

python manage.py startapp home

python manage.py startapp signup

python manage.py startapp login

python manage.py startapp contact

when we create the app in the Django project, we will the

following app structure:
================================================

1) __init__.py ===> it is a empty python file and it represents

the sub folder as package

2) admin.py ===> in this we will register the all models(tables)

related information inside the admin.py

3) apps.py ===> in this we give any configurations related

to app inside the Django project

4) models.py ====> in this we create the models or database

schemas or tables using Django ORM

Django ORM stands for "object relational Mapper"

when we want to create the any table in django, we no need to

create table directly inside the MySQL, using python code we

can create  the table inside the MySQL database, this is

happens in the Django using a concept called "ORM"

we will write the python code for table creation in the Django

app models.py as a class and this class automatically converted

into a table inside the MySQL database

python code for table ===>Django ORM ===>table in MySQL

when we create the any python code for table creation, after

we need to doe the following steps:
================================================

python manage.py makemigrations

python manage.py migrate

using make migrations any changes related to model or any

model we created inside the Django project, we need to run

always run make migrations (it is going to detect any changes

related to model)

after doing the make migrations, we will run the command

called migrate  to apply the changes on models in the project

 5) tests.py :
==========

this file are used to  write any test cases related information

6) views.py:
==========

this file is very important file in the server , when we run the

project in the server , any application will be called or any

response  rendered via view only with help of URL

which is created either using function or class

when we create the view using function , then the view is called

as "functional view "

when we create the view using class, then the view is called

as "Class based view"

when we want to run the Django project in the server, we will

use the following syntax:
==================================================

```
python manage.py runserver
```

  example:
=========

```
python manage.py runserver
```

when we want to run the Django project in the server  with

desired port number, we will use the following syntax:
==================================================

```
python manage.py runserver port_number
```

example:
=======

```
python manage.py runserver  5000
```

note:  port number always in between 0 to 65535

to stop the server for any Django project, we will use

ctrl + c

after we are creating the Django project along with apps, we need to launch the project in the vs code editor

to launch the project in the vs code editor from the command prompt, we will use:

code .

how to work with admin inside the Django project:
=======================================

admin also called as "super user"

to work with  admin of the Django project, first we need to create the credentials for the admin

to create the credentials for the admin, first we need to run the following:

```
python manage.py makemigrations

python manage.py migrate
```

after doing the above, we need to do the following creating the admin credentials:
=================================================

```
python manage.py createsuperuser
```

to access the admin dashboard of the any Django application, we will use the following steps:
=================================================

1. run the project in server ===> python manage.py runserver

2. access the admin using url==> 127.0.0.1:portnumber/admin

using this admin dashboard, we will can able to create / update / delete the any object related to any model, using admin we can also create/ update/ delete the users of the application

what is manage.py in the Django project?
================================

using manage.py we can do the following :
==================================

we can able to create the app in the Django project:
========================================

python manage. startapp app_name

we can run the project on the server
=========================================

python manage.py runserver

we can do the migrations in the project :
=================================================

python manage.py makemigrations

python manage.py migrate

we can create the super user of the project:
=================================================

python manage.py createsuperuser

what is WSGI in Django:
=================================================

WSGI stands for "web application gateway interface"

it used to define the how the web server can communicates

with  web application

WSGI act as a bridge in between web servers and  python web

application

generally web servers related to wsgi are Apache, Nginx,

Gunicorn

using wsgi.py we are creating the entry point to the all

wsgi compatible web servers to load and intereact with Django

application

application ======>  WSGI  ===> web servers

flask:
====

flask is a light-weight web framework

using flask we can develop the small to medium applications

flask is also called as "micro framework"

because like Django, it does not have following:

=====================================

it does not have any pre-defined project structure

it does not any default database

it does not have any built-in ORM

it does not have authentication, form validations

in Falsk, if we need any database support, authentication, form

validations we need to install all third party libraries

in Falsk, every file we need create the by own for the project.

in real time, flask can be used to develop the API's or to

develop the micro services

in real-time, flask we will use to develop the applications like

chat application

real-time dashboards

drawing board

notification Apps

Iot applications

to work with flask, we need to install the flask in our computer
=================================================

pip install flask

after doing the installation, we will create the app using flask

create a folder with name and this name is app name

when we want to work with any flask app, in side the

app  folder, we will create the app.py and which is responsible

to  launch application on the server

in app.py, we will write the following code:
=================================

```
from flask import Flask,render_template
#create the flask app
app=Flask(__name__)
#display the home page
@app.route('/')
def home():
    return render_template('home.html')
if __name__ == '__main__':
    app.run(debug=True)
```

when we are displaying the multiple html pages, in the flask

app, like Django, we will use templates folder, to store the

all html files

when we are  working with flask app, when we want to work

with css, images, js files, we will use "static folder"

using templates folder, we can store the all html files in the

flask app

using static folder, we can store the all css, js and images

in the flask app

to run the flask app inside the server, we will use a command

called    "python app.py"

in the template (html file),

when want to add any css file link, we will use the following

syntax:

```
<link href="{{url_for('static',filename='css/file_name.css')}}"
    rel="stylesheet" type="text/css">
```

when we want to add any js file in the template file(html) file we

will use the following syntax:

=================================================

```
  <script src="{{url_for('static',filename='js/home.js')}}">
  </script>
```

when we want to add any images in the template file, we will

use the following syntax:

=================================================

```
<img
src="{{url_for('static',filename='images/image_name.extension')}}">
```

when we are working Flask or Django, we will have a concept

called "templating"

using templating, we can able to write the python code inside

the "html page"

in flask, to understand the templating in the html, it uses a

template engine called "jinja2" template engine

in Django, to understand the templating in html, it uses a

template engine called "Django template engine" in order to work with any database, flask does not any default database like Django

when we want to work with any database in flask, we need to do manually

Flask does not have any ORM like Django, because Django having a default ORM

in general, when we want to implement the ORM in Flask app, we will use a python library called " SQLAlchemy"

SQLAlchemy is a python library and it act like "Object Relational Mapper and as a SQL toolkit"

Django provides the default forms and user authentication features, flask will not have these features.

to work with forms in the flask, we will use a library called "flask-wtf"

to work with user authentication in the flask will use the "flask-login" library

even the session management can de automatically in the Django and while flask session management is not default feature

when we want to work with restful web services, using Django, Django provides a framework or library called "DRF" , but in flask we will implement the all rest web services manually.

rest ful web serivces is used to create the "API's"

using web service we can able to do a job or task in the real time

in general, when we working with rest ful webservices, using this we can able to share the data from one server to another server

every rest ful web services, it follow the http standard methods for doing the actions

http methods are GET, POST, PUT, DELETE

using GET REST ful web service, we can get the data using POST, REST ful  web service, we can add the data

using PUT, REST ful web service we can update the data

using DELETE, REST ful web service we can delete the data

in this rest full services the response  data will be in json format,

using Django REST Framework, we can also implement serialization and de-serialization
=================================================

steps to work with Django project:
===========================

step-1: activate the virtual environment

enviornment_name\Scripts\activate.bat

step-2: create the Django project with name using following syntax:

django-admin startproject project_name

step-3: move to the project folder

cd project_name

step-4: create the app for the project as follows

python manage.py startapp app_name

step-5:  create the admin credentials using following syntax:

python manage.py makemigrations

python manage.py migrate

python manage.py createsuperuser

step-6: launch the project in the vs code editor as follows

code  .

step-7:  create a folder with name "templates" inside the project folder

step-8: create a folder with name " static" inside the  project folder

in the static folder, create the sub folders with name css, js, images step-9: create a folder with name "media" folder in the

project folder

step-10: give the settings for the following:
============================

1) give the settings for all apps(all created)

2) give the settings for templates folder

3) give the settings for static folder

4) give the settings for database

5) give the settings for media folder

step-10:

create the template(html) file

create the css file in the static css folder

step-11: create the view the for any template to display

on the server

step-12: give the url for the view in the urls.py of the project

folder

step-13 : run the project in the server

how to display the images in the template using Django:
============================================

step-1: keep the all images in the "images" folder of the

static folder in the Django project

step-2: access the any image from the images folder as follows

{% static 'images/image_name.extension' %}

how to work with any MySQL table, which is already created

inside the MySQL database:
=================================================

step-1: check the MySQL and use which database you need to

access the data from the table,give the settings inside

the setttings.py of the project

step-2: to access the table in the model in Django project

we will use the following syntax:

when we want to create the table in the Django using ORM
===============================================

step-1: create a class with all fields of the table , where the

class name is act as "table name" and this we will

create inside the models.py of any app

example:
=======

```
from django.db import models
# Create your models here.
class Employee_sample_fp6(models.Model):
   #create the columns
   id=models.IntegerField(primary_key=True)
   firstname=models.CharField(max_length=100,null=False)
   lastname=models.CharField(max_length=100,null=False)
   email=models.CharField(max_length=100,null=False)
   mobileno=models.BigIntegerField(null=False)
   location=models.CharField(max_length=100,null=False)
```

step-2:
======

once we create the class in the models.py, we need to run the

following:

```
python manage.py makemigrations

python manage.py migrate
```

note:
====

when we create the model, any update or modification or

change to the model , we will always run the above commands

step-3: check the table in the MySQL database

when we want to insert the data into the table using html

form to MySQL database table in MySQL, we will use the

following steps:
=================================================

step-1: create the html file with form and inside the form

tag, add the following token, for secure data

transfers:

```
{% csrf_token %}
```

step-2: define the every form field with name, to access the data in the server, we will use same name

step-3: define the form tag with method as POST and define

the which url need to be call, when we submit the

form and this will be given in the action

how to work with ORM queries:
========================

1) how to retrive the all rows from the table related to all

columns
===============================================

```
data= model_name.objects.all()

in sql,  select * from table_name
```

2. retrive the data related to specific columns from the table

or model:
=================================================

```
data=model_name.objects.values('col1','col2',...'coln')
```

in sql,

select col1,col2,col3,.....coln from table_name;

```
def data(request):
    #we need to get the data from the employee table
    #writing the same query using ORM
    # get only id,firstname,lastname
 mydata=Employee_sample_fp6.objects.values('id','firstname','lastname')
    """
    when we are sending the data to the template
    we need to pass it as a dictionary
    """
    result={'data':mydata}
    return render(request,'data.html',result)
```

3. get the data based on the any condition:
==================================

syntax:
======

```
data= model_name.objects.filter()
```

in sql,

select * from table_name where condition

example:
=======

```
def data(request):
    # get only id,firstname,lastname where id is 10000
```

```
mydata=Employee_sample_fp6.objects.values('id','firstname','lastname').filter(id
=1000)
    result={'data':mydata}
    return render(request,'data.html',result)

example-2:
=========
def data(request):
    # get all columns data where id is 10000
    mydata=Employee_sample_fp6.objects.filter(id=1000)
    result={'data':mydata}
    return render(request,'data.html',result)



4.>,<,==, (any condition related to > , < , >=,<=, ==)
=================================================

  data=model_name.objects.filter(column_name__gt=value);

   in sql,

  select * from table_name where col>=value

example:
=======

def data(request):
    # get all columns data where id is greter than or equal to 20
    mydata=Employee_sample_fp6.objects.filter(id__gte=20)
    result={'data':mydata}
    return render(request,'data.html',result)


example-2:
=========

def data(request):
    # get all columns data where id is greter than 20
    mydata=Employee_sample_fp6.objects.filter(id__gt=20)
    result={'data':mydata}
    return render(request,'data.html',result)

example:
=======
def data(request):
    # get all columns data where id is less than or equal to 20
    mydata=Employee_sample_fp6.objects.filter(id__lte=20)
    result={'data':mydata}
    return render(request,'data.html',result)

example:
=======

def data(request):
    # get all columns data where id is less than 20
    mydata=Employee_sample_fp6.objects.filter(id__gt=20)
    result={'data':mydata}
    return render(request,'data.html',result)

example:
=======

def data(request):
```

```
    # get all columns data where id is greter than or equal to 20
    mydata=Employee_sample_fp6.objects.filter(id=20)
    result={'data':mydata}
    return render(request,'data.html',result)
```

5.how to display the data which is in sorting order using ORM:
=================================================

```
   data= model_name.objects.order_by('columnname)
```

by default it will give the result in the ascending order

if we want the data in descending order, we will use - before

the column name

example:
=======

```
def data(request):
    # get all columns data where is order  by  firstname ascedning order
    mydata=Employee_sample_fp6.objects.order_by('firstname')
    result={'data':mydata}
    return render(request,'data.html',result)
```

example:
========

```
def data(request):
    # get all columns data where is order  by  firstname descedning order
    mydata=Employee_sample_fp6.objects.order_by('-firstname')
    result={'data':mydata}
    return render(request,'data.html',result)
```

6. display the first 5 rows only from the table:
====================================

```
  data=model_name.objects.all[:row_number]
```

example:
=======

```
def data(request):
    # get all columns data where we need to display data only top 5 rows
    mydata=Employee_sample_fp6.objects.all()[:5]
    result={'data':mydata}
    return render(request,'data.html',result)
```

6. display the data from the last 5 rows from the model:
============================================

```
data=model_name.objects.all()[totalrows-numberrows:totalrows]
```
example:
=======
```
def data(request):
   #get the number fo rows from the mysql table
    count=Employee_sample_fp6.objects.all().count()
    # get all columns data where we need to display data only last 5 rows
    mydata=Employee_sample_fp6.objects.all()[count-5:count]
    result={'data':mydata}
```

```
return render(request,'data.html',result)
```