* OOPS (object Oriented Programs)

→ to deal with real time projects oops is used.
→ here we make use of class and objects.
→

class :-

* it is a collection of variables and methods or it is a blueprint which consist of properties and functionalities. or

class is a container where the datas are stored and later it can be access

object :- * it is an instance of class or it is a variable created for a class (it can be inbuilt or userdefine)

→ class are of two types.
① inbuilt class ② userdefine class.

① inbuilt class :- These are the classes which are predefine with some properties or functionalities
ex :- int, float, complex, list, set, dict etc.

② userdefine class :- it is a class created based on user requirement
→ to create a class in python we use keyword (class)

Syntax :- class cname:
properties / attributes
functionalities / behaviour

obj = uname(args)

example :-

```
class lion:
    pass
ob = lion()
print(type(ob))
```

o/p = z = 12

```
type(z)
<class 'int'>
<class '__main__.lion'>
```

Memory allocation :-

once a class is created internally memory will be creating a key value layer.

→ all the properties of the class will be stored in that dictionary.

→ the values will be having reference address

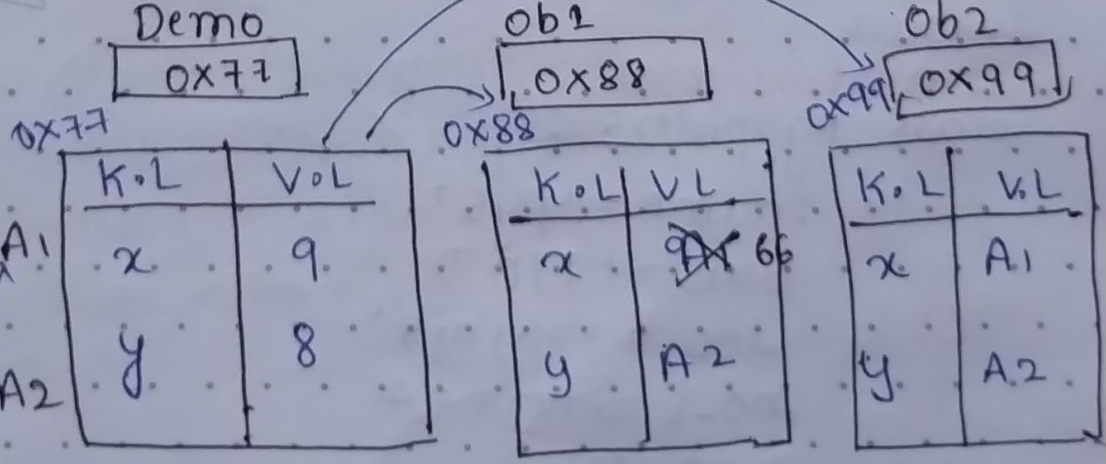
→ key layer will have a address which will be stored with respect to class name.

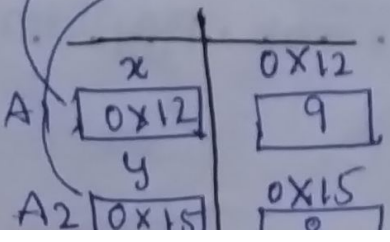
object creation :- when object is created then this will create a dictionary and the properties from the class will be derived into this dictionary (object

→ key layer will have address and it will be stored with respect to object name

Note :- a class can have n number of objects

```
class Demo:
    x = 9
    y = 8
ob1 = Demo()
ob2 = Demo()
```





Access property	Modify
classname.pname	cname.pname = value /
objname.pname	objname.pname = value

Types of states / members / attributes / properties :-

It is of 2 types.

① class members ② object Members

* class members :- these are the members of class which are common for all the object

* object Members :- these are the members which are different for all the object.

ex :- if a bank is considered as class, customers can be considered as objects.

→ class Members are Security, bankname, Manager

→ object Members are :- account no., Passbook, amount

ex :- class Metro

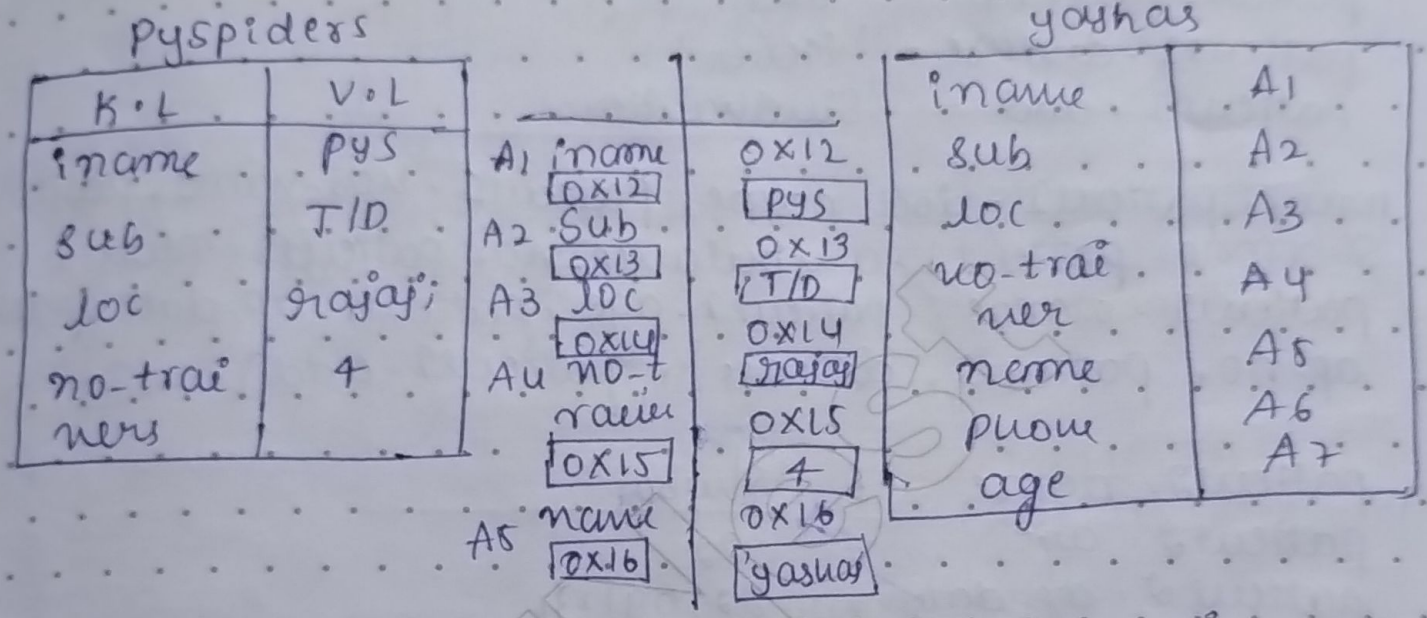
object :- passengers

C.M :- Station, cardtype, AC, voice

O.M = destination, Source mode pay

```
class pyspiders :
    iname = 'pys'
    sub = 'TID'
    loc = 'rajaji'
    no-trainers = 4
yashas = pyspiders()
print (pyspiders.iname, pyspiders.sub, pyspiders.loc
       pyspiders.no-trainers)
```

yashas.name = 'yashas'
yashas.phone = 9876543456
yashas.age = 20
print(yashas.iname, yashas.sub, yashas.loc, yashas.no-trainers, yashas.name, yashas.phone, yashas.age)



→ in the above example iname, sub, loc, no-trainers are classMembers.
→ name, phone age are objectmembers.

Class = Pyspiders Object = yashas

1) create a class called ——— with minimum 5 classproperties, 3 objects, 6 objects properties

```
class hospital:
        doct-name = 'SangMesh'
        Hos-name = 'Modi Hospital'
        SC-rmno = '201'
        ambulance-no = 108
        nurs = 'Ganga'

patient1 = hospital()
patient2 = hospital()
patient3 = hospital()
```

print(hospital.doct-name, hospital.HOS-name, hospital.
SC-rmno, hospital.ambulance-no, hospital.nurs)
patient.name = 'pooja'
patient1.age = 22
patient1.ap-date = 21/09/23
patient1.ap-no = 809
patient1.disease = 'fever'
patient1.add = 'Sindhankera'

print(patient1.doct-name, patient1.hos-name, patient1.
SC-rmno, patient1.ambulance-no, patient1.nurs,
patient1.name, patient1.age, patient1.ap-date, patient1.
ap-no, patient1.disease, patient1.add)

patient2.name = 'shubbi'
patient2.age = 22
patient2.ap-date = 22/09/23
patient2.ap-no = 810
patient2.disease = 'Stomach pain'
patient2.add = 'Wanjarkheda'

print(Patient2.doct-name, patient2.hos-name, patient2.
SC-rmno, patient2.ambulance-no, patient2.nurs,
patient2.name, patient2.age, patient2.ap-date, patient2
.ap-no, patient2.disease, patient2.add)
patient3.name = 'Gouri'
patient3.age = 22
patient3.ap-date = 23/09/24
patient3.ap-no = 815
patient3.disease = 'headache'
patient3.add = 'Kalwra'

* Initialization / __init__ / constructor

→ it is a method which is used to initialize the Members of objects
→ in this self have to be pass as the first argument
→ any other variables can be used other than self
→ the first argument ~~is used to~~ pass will take the address of object
→ it is not necessary to call the init Method outside the class

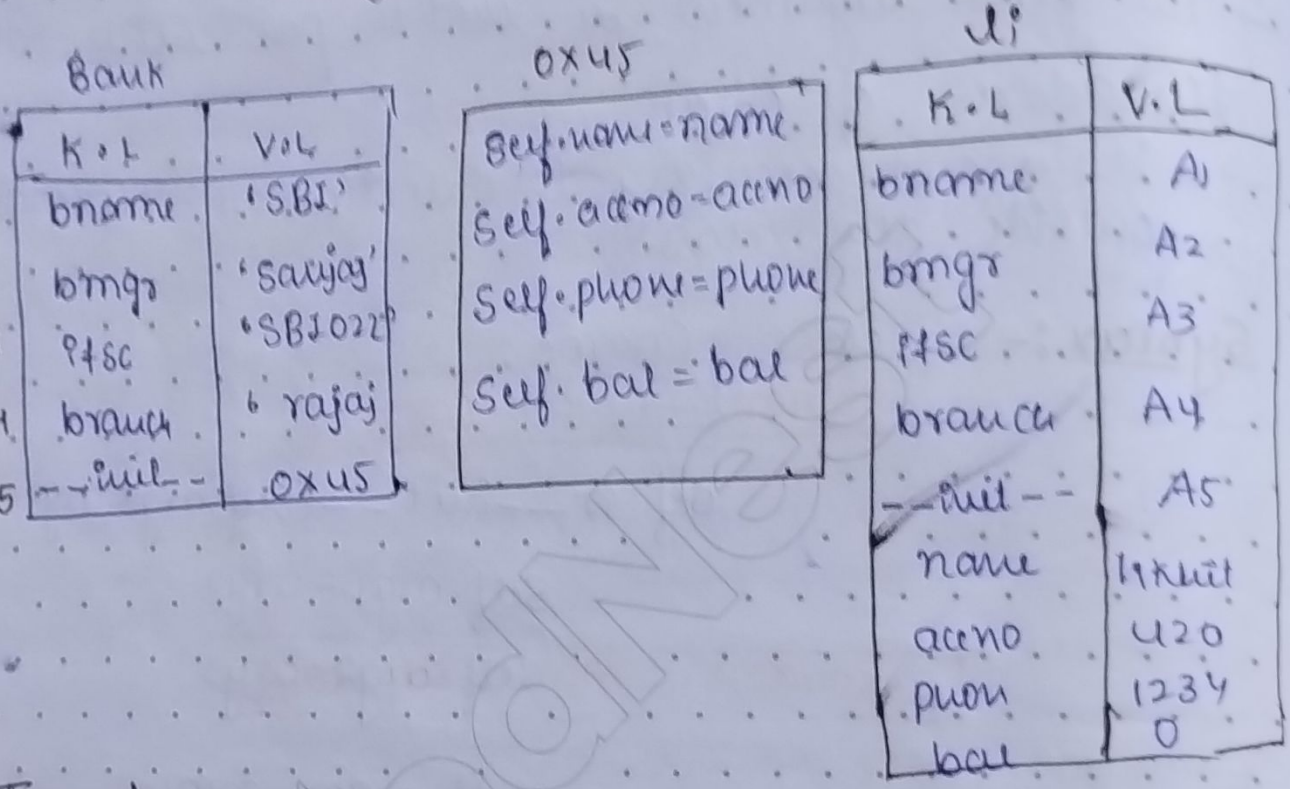
Syntax :-

```
class cname :
    def __init__(self, arg1, arg2 ... argn):
        self.arg1 = arg1
        self.arg2 = arg2
        :
        self.argn = argn
ob = cname(val1, val2 ..... valn)
```

```
class Bank:
    bname = 'SBI'
    bmgr = 'sanjay'
    ifsc = 'SBI00225'
    branch = 'rajaji'
    def __init__(self, name, acno, phone, bal):
        self.name = name
        self.acno = acno
        self.phone = phone
        self.bal = bal
```

```
li = Bank('Lingu', 43042, 9112345678, 10000)
man = Bank('Manja', 11111, 12345,6789, 500)
print(li.name, li.accno, li.phone, li.bal)
print(man.name, man.accno, man.phone, man.bal)
```

Memory allocation



Types of Methods

→ a function return inside a class is called as Method.

→ There are 3 types of Methods

1) object Method
2) class Method
3) Static Method

Object Method :- it is a Method which is used to access and Modifie members of object

Syntax : class cname :

    def mname(self,args1,...argn):
        S.B

obj = cname(args)

→ for object Method passing self is mandatory other than self different variables can be use but according to standards self must be used

→ to access a object method

objname.mname(args)

→ when a object method is called through a classname name of the object (self) have to be passed.

classname.mname(self, args)

example:-

```
class Bank:
    bname = 'yes bank'
    bmgr = 'sanjay'
    ifsc = 'yes0225'
    brainch = 'rajaji'
    def __init__(self,name,acno,phone,bal):
        self.name = name
        self.acno = acno
        self.phone = phone
        self.bal = bal
    def display(self):
        print(self.name,self.acno,self.phone
              self.bal)
    def change_phone(self,new):
        self.phone = new
```

li = Bank('likith', 4207220000, 911007662, 100000000)
Mau = Bank('Manjaa', 111111111, 88888888, 500)
li.display()
Bank.change-phone(li, 8765423214)
li.change-phone(4859632145)

class Method

→ it is a method which is used to access and modify
- its the members of class
→ in this a decorator @classMethod have to be used
→ cls have to be passed as the first argument (other variables can be used)
→ to call a class Method `classname.mname(args)`

example :-
```
@classmethod
def ch-mgr(cls):
    new = input("enter the new mgr:")
    cls.bmgr = new
    print("congrajulation", new)
```

1) create a class with minimum 4 class members & 5 object members 3 objects with 2 object Methods & 1 class Method.

```
class Hostel:
    warden-name = 'Savitri'
    location = 'Vidya nagar'
    incharge = 'Raju Anile'
    last-entry = '6 pm'
```

~~Student1 = Hostel()~~
~~Stud2 = Hostel()~~
~~Stud3 = Hostel()~~

```
def __init__(self, name, roomno, floor_no, section, branch):
    self.name = name
    self.roomno = roomno
    self.floor_no = floor_no
    self.section = section
    self.branch = branch

def display(self):
    print(self.name, self.roomno, self.floor_no, self.section, self.branch)

def change_room(self, new_room):
    self.roomno = new_room

@classmethod
def ch_wardname(cls, newname):
    cls.warden_name = newname)
@classmethod
def ch_timing(cls, newtime):
    cls.last_entry = newtime
@classmethod
def display(cls):
    print(cls.warden_name, cls.incharge, cls.last_entry)
```

```
neelu = Hostel('nilambika', '57', '1st', '01', 'EC')
shubbi = Hostel('shubhangi', '59', '2nd', 'c2', 'CSE')
chiku = Hostel('chiketa', '511', 'Ground', 'M35', 'civil')
```

```
neelu.display()
Shubbi.display()
chiku.display()
chiku.avail_room('M6S')
Hostel.display()
Hostel.ch_timelig('7pm')
```

Static method :- it is a method which is neither belong to class members or object members but it place a supportive role.

→ it is required to pass a decorater before creating method.

→ it is not required to pass self or cls.

→ it is not necessary to call static method ourside the class

Syntax :-

```
class Cname:
    @staticmethod
    def mname(args):
        S.B
obj = cname(args)
```

how we call a Static Method `cls/self.mname(args)`

```
class Bank:
    Bname = 'KBL'
    Bmgr = 'Kishan-kinthau'
    loc = 'GD bar'
    ifsc = 'KBLHK00490'
```

```
    def __init__(self, name, phone, accno, bal):
        self.name = name
        self.phone = phone
        self.accno = accno
        self.bal = bal
        self.transaction=[]
    def display(self):
        print(self.name, self.phone, self.acno,
              self.bal)
    def withdraw (self, amt):
        if amt<self.bal:
            self.bal=self.sub(self.bal, amt)
            self.transaction.append(f'{amt}
is debited from account| balance left --{self.bal}')
        else:
            print('Thupukk dudd illaa')
    def deposite(self, amt):
        self.bal = self.add(self.bal, amt)
        self.transaction.append(f'{amt} is
        credited to account| balance left -{self.bal}')
    def transfer(self, to_acc, amt):
        if amt<self.bal:
            self.bal=self.sub(self.bal, amt)
            to_acc.bal = self.add( to_acc.bal, amt)
            self.transaction.append(f'{amt}
transfered to {to_acc.name}| balance left ---{self.bal}')
            print(f'amount transferred successfully
                     to {to_acc.name}')
        else:
            print('print Magaa dudd nale kala
                     time')
```

```
    def print_pass(self):
        for i in self.transaction:
            print(i)
    @staticmethod
    def add(a,b):
        return a+b
    @staticmethod
    def sub(a,b):
        return a-b
man=Bank('Manjaa',987654321,11111000,20000)
pro=Bank('prodeep',93142567,222002222,150)
man.display()
man.withdraw(5000)
man.bal
man.deposit(4000)
man.bal
man.print_pass()
```

2) Create a class called as library with basic properties and methods along with issue-book and return book methods

A student can purchase maximum of 5 books.

3) Create a class called as flipcart with basic property. Create methods called buy and return add functionalities like if a person purchases a product the amount should get detected from his account, create payment options like COD, UPI, Debitcard

when Debitcard payment is done OTP must be received to the Compared for otp generation (random).

1) Inheritance :-
it is a phenomenon of deriving properties from one class to another class

→ the class from where properties are derived are called parent class or superclass or base class
→ the class to which properties are derived is called child class subclass or derived class

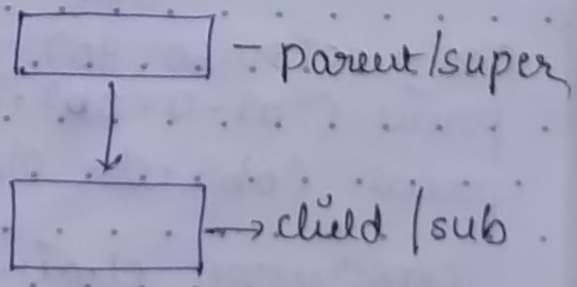
There are 5 types of Inheritance

① Single level Inheritance
② Multi Level Inheritance
③ Multiple Inheritance
④ Hierarchical Inheritance
⑤ Hybrid Inheritance

① Single level Inheritance :- it is the phenomenon of deriving properties from one parent class to one child class

Syntax

```
class PC :
    S.B
class CC(PC):
    S.B
```



Example

```
class PC:
    a=33
    b=66
class CC(PC):
    x=6
ob1=CC()
ob2=PC()
```

```
>>> ob1.x=6
ob1.a=33
ob1.b=66
```

```
>>> ob2.a=33
>>> ob2.b=66
>>> ob2.x=error
```

Note:- When there are methods or variables which are repeated it always takes a latest value (overriding)

```
class PC:
    a=33
    b=66
    def __init__(self,m,k):
        self.m=m
        self.k=k
class CC(PC):
    x=6
    a=55
    def __init__(self,d,s,n):
        self.d=d
        self.s=s
        self.n=n
ob1=CC(30,60,90)
print(ob1.a,ob1.b)
print(ob1.d,ob1.s,ob1.n,ob1.x)
```

Constructor chaining :- It is a phenomenon of invoking constructor from parent class to child class

Syntax :- `super().--init--(args)`

`super(child,self).--init--(args)`

`pcname.--init--(self,args)`

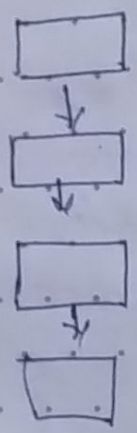
method chaining :- it is a phenomenon of invoking method from of parent class to child class.

syntax :- `super().mname(args)`

```
class Bank:
    bname='syndicate'
    loc='rajajinagar'
    ifsc='syndoo1'
    bmgr='gouri'
    def __init__(self,name,phone,bal):
        self.name=name
        self.phone=phone
        self.bal=bal
    def disp(self):
        print(self.name,self.phone,self.bal
              end=' ')

    def ch-phone(self,new):
        self.phone=new

class Bank1(Bank):
    bname='Canara'
    ifsc='CANRB0029'
    bmgr='yuva'
    def __init__(self,name,phone,bal,addr,pan):
        super().__init__(name,phone,bal)
```

```
        self.aadr = aadr
        self.pan = pan
    def disp(self):
        super().disp()
        print(self.aadr, self.pan)

pr = Bank2('prem', 9876543, 100000, 99884321, 'BIR1P0332')
```

Multilevel Inheritance :- It is a phenomenon of deriving property from one class to another class by considering more than one class




```
class C1:
    <--> S.B
class C2(C1):
    <--> S.B
class C3(C2):
    <--> S.B
```

Basic

```
class A:
    a=10
    b=20
class B(A):
    d=22
    e=40
class C(B):
    f=41
```

```
class resume-10:
    class = '10th'
    S-name = 'Vishwabharati'
    def __init__(self, name, t-per, t-passout):
        self.name = name
        self.t-per = t-per
        self.t-passout = t-passout
    def display(self):
        print(self.name, self.t-per, self.t-passou)
```

end

```
class resume-12(resume-10):
    clas = '12th'
    c-name = 'shaheen'
    def __init__(self, name, t_per, t_passout, pv_per,
                 pv_passout):
        super().__init__(name, t_per, t_passout)
        self.pv_per = pv_per
        self.pv_passout = pv_passout

    def disp1(self):
        super().disp()
        print(self.pv_per, self.pv_passout, end=' ')

class degree_resume(resume_12):
    class = 'B.Tech'
    dc_name = 'S.B.R'
    def __init__(self, name, t_per, t_passout,
    pv_per, pv_passout, d_per, d_passout):
        super().__init__(name, t_per, t_passout,
                         pv_per, pv_passout)
        self.d_per = d_per
        self.d_passout = d_passout

    def disp2(self):
        super().disp1()
        print(self.d_per, self.d_passout)

pooja = degree_resume('poojas', '83.14', '2018', '83.64',
                      '2020', '89', '2024')
```

Multiple Inheritance - it is a phenomenon of deriving property from multiple parent class to single child class

class PC1:
    S.B
class PC2:
    S.B
class PC3
    S.B
class CC(PC1, PC2, PC3):
    S.B

class A:
    x=5
    y=10
class B:
    m=3
    n=1
    x=11
class C:
    pass
class D(A,B,C)
    p=7
    q=11
ob=D()

Note:- Properties will be derived from RHS to LHS

Hierchaical Inheritance:- it is the phenomenon of deriving properties from single parent class to multiple child class

Syntax:-

class PC:
    S.B
class C1(PC):
    S.B
class C2(PC):
    S.B
class C3(PC):
    S.B

example

class A:
    x=9
    y=8
class B(A):
    m=6
class C(A):
    p=4
    q=11
ob=C()

>>> ob.p
4
>>> ob.x
9
>>> ob.y
8
>>> ob.q
11
>>> ob.m
error

A

	K.L	V.L
A1	x	9
A2	y	8

B

	K.L	V.L
	x	A1
	y	A2
B1	m	6

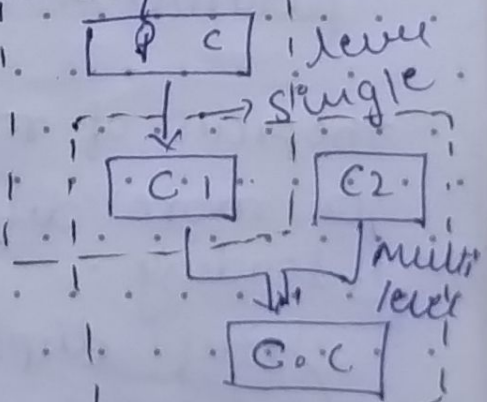
C

K.L	V.L
x	A1
y	A2
p	4
q	11

```
class chat:
    contacts={'ranga':[],'manja':[],'ranghu':[]}
    def send-message(self):
        name=input("enter the name to chat:")
        if name in chat: contacts:
            msg=input("enter:")
            chat:contacts[name].append(msg)
        else:
            print('invlivl yaroo goth illaa')
class WA(chat):
    pass
class Insta(chat):
    pass
yas=WA()
yas.send-message()
```

Hybrid inheritance:- it is the phenomenon of deriving properties by considering more than one type of inheritance

note:- since it is done based on user requirement there is no proper syntax




```
class Addition:
    @staticmethod
    def add(a,b)
        return a+b
class Substraction(Addition):
    @staticmethod
    def sub(a,b):
        return a-b
```

```
class division:
    @staticmethod
    def div(a,b):
        return a/b
class calculator(Substraction, Division):
    @staticmethod
    def mul(a,b):
        return a*b
```

Polymorphism:- it is a process or phenomenon of making a method or an operator to work on two or more operations or tasks.

→ there are 2 types of polymorphism

① method overloading

② operator overloading

① method overloading :- defining or creating different methods with the same method name with different parameters is called as method overloading

or

the process of making the method to work on two or more different tasks is called as method over-loading.

```
def display(a):
    print(a+20)
def display(a,b):
    print(a+b)
def display(x,y,z):
    print(x+y+z)
```

display
~~0x11~~
~~0x22~~
0x33

method area

0x11 | print(a+20)

0x22 | print(a+b)

0x33 | print(x+y+z)

Note:- in python everything we try to perform method overloading internally it performs method overriding

method overriding:- when two or more methods are created with the same name than the last method will override all the previous method; this process is called as method overriding.

→ in the above example the third method has overridd all previous methods by overriding the address. hence the display function must be called by exactly passing 3rd arguments.

```
class first:
    a=30
    b=40
    def __init__(self,p,q):
        self.p=p
        self.q=q
    def show(self):
        print(self.p)
    def show(self):
        print(self.q)
    def show(self):
        print(self.p,self.q)
f1=first(400,800)
f1.show()
```

K.L	V.L
a	30
b	40
__init__	0x22
show	~~0x33~~ ~~0x44~~ 0x55

```
class first:
    a=30
    b=40
    def __init__(self,p,q):
        self.p=p
        self.q=q
    def show(self):
        print(self.p,self.q)
class second(first):
    x=60
    def __init__(self,p,q,r,s):
        super().__init__(p,q)
        self.r=r
        self.s=s
    def show():
        super().show()
        print(self.p,self.q,self.r,self.s)
f1=first(20,30)
f1.show()
f2=second(20,30,40,50)
f2.show()
```

Operator overloading :- it is a phenomenon of making objects of userdefine class work on operators.

→ when an operators are used internally it invoks there respective magic methods

Magic methods :- any methods which are starting and ending with (double underscore) __ which are called magic methods

→ All the magic methods will have some internal operation.
→ It is not necessary to call a magic method explicitly
→ for operators like arith, bitwise, relational etc will have magic methods

① __add__ :— it is a method which will be internally called or invoked when + operator is used.
→ Arithmetic operator

operator name	Symbol	internal operation
addition	+	ob1.__add__(ob2)
Substraction	−	ob1.__sub__(ob2)
multiplication	*	ob1.__mul__(ob2)
truedivision	/	ob1.__truediv__(ob2)
floordivision	//	ob1.__floordiv__(ob2)
modulus	%	ob1.__mod__(ob2)
power	**	ob1.__pow__(ob2)

Relational operator

greater than	>	ob1.__gt__(ob2)
less than	<	ob1.__lt__(ob2)
greater than equal to	>=	ob1.__ge__(ob2)
lessthan equal to	<=	ob1.__le__(ob2)
equal to	==	ob1.__equal__(ob2)
not equal to	!=	ob1.__ne__(ob2)

and	&&	ob1.__and__(ob2)
or	||	ob1.__or__(ob2)
xor	^	ob1.__xor__(ob2)
leftshift	<<	ob1.__lshift__(ob2)
rightshift	>>	ob1.__rshift__(ob2)
not	~	ob1.__invert__

ex:- x=10
x.--add--(20)
30
>>> x+20
30

>>> x.--sub--(90)
-80
>>> x-90
-80

>>> 60+90
150

>>> 25+18
43

>>> 'lover boy' + 'thippu'
'lover boy thippu'

>>> 'Thippu thippi' - 'thippi'
error

>>> {45, 90} + {2, 3}
error

Note:-(+) operator will not work on set datatype because set class doesn't contain magic method --add--
Similarly in String class --sub-- is not present because of this minus operator will not work on String data type.

```
class gandhi:
    def __init__(self,a):
        self.a=a
    def __add__(self,other):
        return self.a+other.a
    def __sub__(self,other):
        return self.a-other.a
ob1=gandhi(10)
ob2=gandhi(20)
print(ob1+ob2)
print(ob1-ob2)
```

o/p = 30
-10

```
class manipav(set):
    def __init__(self, a):
        self.a = a
    def __add__(self, other):
        res = self.a
        for i in other.a:
            res.add(i)
        return res
ob1 = manipav({'mani', 'pav', 'pavii'})
ob2 = manipav({90, 180, 'rangaa'})
print(ob1 + ob2)
print(ob1 - ob2)
```

Encapsulation :- It is the phenomenon of providing security to the data or properties.

access specifiers :- these are members of class which specifies wether it can be access outside the class or not
it is of three type
① public Access Specifier
② protected " "
③ private " "

public access Specifiers :- these are the members of class which can be access the outside the class or sub class
Note :- the programs written till know in oops are public access specifier

Protected access Specifier :- these are the members of class which ~~are~~ will not provide security to the data in python

to declare a protected access specifier Single underscore (_) must be written before a variable or method

Syntax:- class cname:

_var = value

def _mname(self, args):

S.B

Obj = cname()

note:- it is a convention that a protected access specifier should be used only in that particular class. → a programmer should avoid using protected access specifier outside the class or in any subclasses.

private access specifier:- these are the member of class which provides security to the data (it cannot be access outside the class or in any subclasses.

→ to declare a private access specifiers (__) double underscore must be used before declaring a variable or method.

Syntax:- class cname:

__var = value

def __mname(self, args):

S.B

obj = cname()

```
class Company:
    cname = 'TY'
    ceo = 'girish sir'
    __revenue = '***'
    def __init__(self, name, phone, sal):
        self.name = name
        self.phone = phone
        self.__sal = sal
```

```
        def __disp(self):
            print(self.name, self.phone, self.__sal)
vj = Company('vijay', 9110823456, 100000)
print(vj.name)
print(vj.phone)
```

note:- private access specifier cannot be access directly syntax to access for accessing

`cname/objname._cname__pname` = value

`cname/objname._cname__mname()` = value

* property decorator:-

It is a inbuilt decorator which is used to access and modify private access specifier (or) private members.

→ decorator property have to be passed on a particular method

→ using getter & setters it is possible to access the private members.

→ to delete the private member deleter will be used.

→ the method name with a property decorator will be converted into a property (variable).

→ to modify the property :- @pname.setter decorator must be used & similarly :- @pname.deleter must be used to delete the attribute

```
class Cname:
    @property
    def pname(self):
        return self.__pname

    @pname.setter
    def pname(self,value)
        self.__pname = value
    @pname.deleter
    def pname(self):
        del self.__pname
```

example :-

```
class Company:
    cname = 'TY'
    ceo = 'girish sir'
    __revenue = '***'
    def __init__(self, name, phone, sal):
        self.name = name
        self.phone = phone
        self.__sal = sal

    def __disp(self):
        print(self.name, self.phone, self.__sal)
VJ = company ('vijay', 8024263914, 100000)
print(VJ.name)
print(VJ.phone)
```

Abstraction :- it is a phenomenon of hiding the implementation from the user so that the functionality will work

→ to learn abstraction concepts like ① abstract method ② abstract class ③ concrete class must be know.

① abstract method:- any method which has only function declaration but not function definition is called as abstract method

```
def fname(args):      → declaration
    pass              → definition
```

② abstract class:- if a class contain atleast one abstract method it is called as abstract class.
to declare abstract class in python ABC class must be imported from abc package

Syntax

```
from abc import ABC, abstractmethod
class cname(ABC):
    @abstractmethod
    def mname(args):
        pass
```

③ Concrete class:- any class which doesn't contain any abstract method in it is called as concrete class.

ex:- abstract class

```
from abc import ABC, abstractmethod
class Demo_insta(ABC):
    @abstractmethod
    def signup():
        pass
    @abstractmethod
    def login():
        pass
```

concrete class

```
class insta(Demo_insta):
    def signup(self):
        print('Good morning')
    def login(self):
        print('Good evening')
ob = Instagram()
```

Note :- it is not possible to create an object for abstract class when it is required to convert abstract class into concrete class, abstract class must be inheritated to new class and give implementation to all the abstract method.

Advance TOPIC

① WAP which return True if a integer number is even if not return false.

```
def check(a):
    if a%.2==0:
        return True
    else:
        return False
print(check.(int(input("enter:"))))
```

(or)

```
if a%.2==0
    return True
```

→ def even-odd(n): (or) : def even-odd(n):

```
    if n%.2==0:                    return n%.2==0
        return True
    return False
```

Lambda : it is a keyword which acts like anonymous function to perform simple operations in python.

→ Since it doesn't contain any name, it is called as anonymous when a variable is passed for lamba function that variable will store the lamba function which can be called as function name (variable)

→ Syntax :

```
var = lambda args : expressions
print(var(values))
```

→ Example : even-odd = lambda n : n%2 == 0
print(even-odd(5))

→ output : even-odd(66)
True

program :- 1) WAP to find cube of a integer number using lambda

→ cube = lambda a : a**3
print(cube(int(input())))

2) WAP to print yes if the string is palindrom if not print No

→ palindrom = lambda a : 'yes' if a[::-1] == a
else 'No'
print(palindrom(input('Enter:')))

3) WAP lamda function which reverse only the value are collection if not print the value as it is

reverse = lambda a : a=[::-1] if type(a) in [str, list, tuple] else a
print(reverse(input()))

Syntax :-

```
Var = lamda args : TSB if condition
                   else FSB
```

4) WAP to print the square of number if it is odd print cube of of the number if it is even

```
sq_no = lambda a: n**2 if a%2!=0 else n**3
print(sq_no(input()))
```

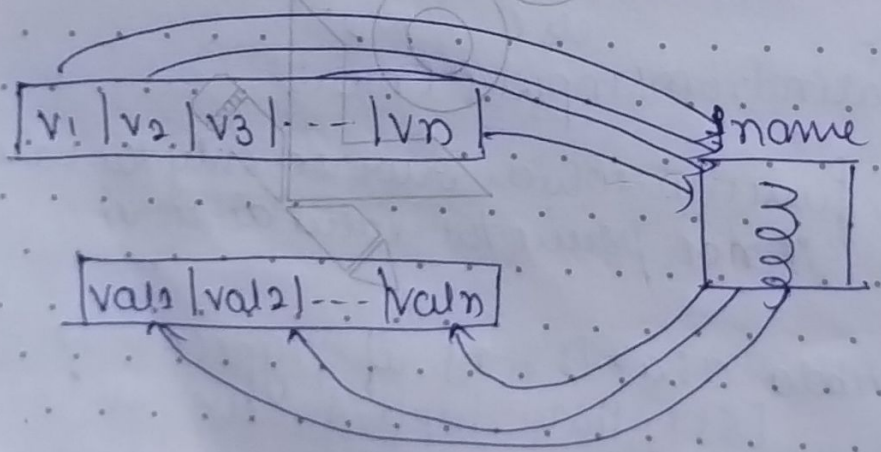
5 WAP to print the Ascii value of a character if is vowel if not print not vowel

→
```
Ascii_value = lambda a: ord(a) if a in 'AEIOUaeiou' else 'not vowel'
print(Ascii_value(input('Enter: ')))
```

map(): it is a function which performs same action or operation for every value present in the collection

Syntax:

```
var = map(fname, collection)
print(type_cast(var))
```



Example

1) WAP to find square of number present in homogeneous list Collection.

```
li = eval(input('Enter: '))
sqr = []
for i in li
    sqr += [i**2]
print(sqr)
```

(or)

```
sqr = lamda a:a**2
ll = eval(input('enter:'))
sqr_series = map(sqr, li)
print(list(sqr_series))
```

In advance

```
print(list(map(lambda a:a**2, eval(input()))))
```

Note :- The function name used in map function can be inbuilt function userdefine function or lambda functio

② Consider a String collection & store the length of individual words in the form of list collection.

```
x = input('enter:').split()
length_word = lamda a: len(a)
coll = map(length_word, x)
print(list(coll))
```

or

```
print(list(map(lamda a: len(a) input('enter').
                split())))
```

or

```
n = input('enter:')
x = n.split()
coll = map(len, x)
print(list(coll))
```

3) WAP to store word & its length as key value pairs present in a string collection

```
→ St = input ('enter'). Split()
  word_len = lambda a:[a, len(a)]
  string = map (word_len, St)
  print (dict (string))
```

or

```
print (dict (map (lambda a:[a, len(a)], input ('ent n')
                 .Split())))
```

WAP to find a factorial of a number between range 1 to 5

```
def fact(n):
    f = 1
    for i in range(1, n+1):
        f *= i
    return f

series_fact = map(fact, range(1, 6))
print (list(series_fact))
```

or

```
import math
series_fact = map(math.factorial, range(1, 6))
print (list(series_fact))
```

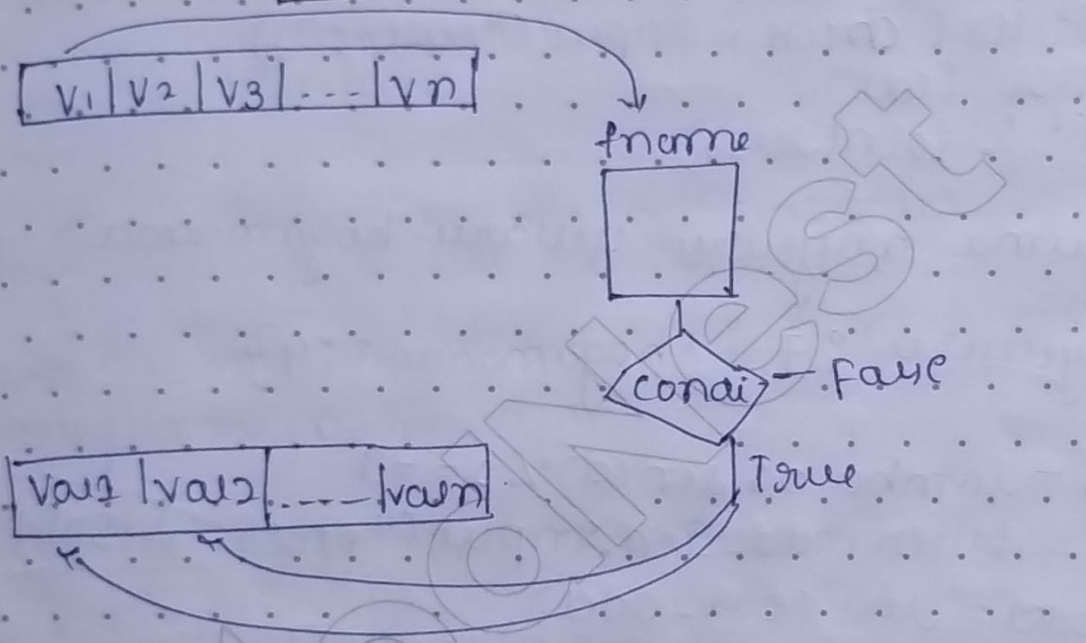
filter() :- it is a function which is used to get the required value from the collection.

① in case of map function the length of input and output same.

② the function used in filter function it should return only true or False

Syntax :-

```
var = filter (fname, coll)
print (datatype (var))
```



① WAP to extract all the integer present in a list collection

```
n = eval(input("enter"))
out = [ ]
for i in n:
    if i%2 == 0
        out += [i]
```

```
n = eval(input("enter"))
out = [ ]
for i in n:
    if type(i) == int:
        out += [i]
```

```
check_int = lambda a: type(a) == int
ext_int = filter(check_int, eval(input("enter:")))
print(list(ext_int))
```

OR

```
print(list(filter(lambda a: type(a) == int, eval(input("enter:")))))
```

WAP to extract all the vowels having ASCII values are odd present in String collection.

```
out=[]
St=input("enter a String"))
for i in St:
    if i in 'aciouAEIOU' and ord(i)%2!=0:
        out+=[i]
```

```
check=lambda a:a in 'aciouAEIOU' and ord(a)%2!=0
ext = filter(check, input("enter:"))
res = list(ext)
print(''.join(res))
```

St = 'hoguma myswru yevaru hogumaaaa'

O/P = {'yevaru':7, 'hogumaaaa':9}

```
odd_len = lamba a: len(a)%2==1
ext_odd = filter(odd_len, input("enter:").split())
words_odd = list(ext_odd)
len_words = lamba x:[x, len(x)]
ext-length = map(len_words, words_odd)
print(dict(ext_length))
```

WAP to extract all the even digits in a String collection

~~a:type(a)==int and~~

```
extract-even = lambda ~~a~~ a%2==0 a: int(a)%2==0
extract = filter(extract_even, input("enter:"))~~.split()~~
print(''.join(list(coll)))
```

wap to store number and its qube in a key value pair between rang (1,10) only if it is odd numbe

package architecture

package :- package is a folder where we stored the different files or modules.

Module :- module is a file which consist of different properties or attributes

There are 2 types of modules

① inbuilt modules ② user define modules

inbuit modules :- The files which are define by the developer are called as inbuilt modules.
ex:- math, random, json, pickle, re etc.

user define modules :- the files which are define by the user are called as userdefine modules.

import :- import is a keyword which is used to import one file into another file.

Syntax :- import module_name

dir :- it is an inbuilt which is used to know diffrent properties and function present in given module

→ import math
dir(math)

['factorial', 'floor', 'ceil', 'pi', 'pow']

① Math.factorial(5) = 120

② Math.floor(3.5) = 3

③ Math.ceil(3.67) = 4

④ Math.sqrt(9) = 3.0

⑤ Math.pi = 3.141592

from :- it is a keyword which is used to import the specific function from the given module

Syntax :- from module-name import fname

ex : >>> from math import factorial
>>> factorial(3)
6
>>> sqrt(4)
error
>>> Math.Sqrt(4)
error

>>> from math import *
>>> factorial(8)
40320
>>> sqrt(9)
3.0
>>> pi
3.141

from math import factorial
sqrt
>>> factorial(8)
40320
>>> sqrt(9)
3.0
>>> pi
error

aliasing :- it's a process of giving an alternative name for a specific module.

syntax :- import module-name as alter-name

ex:-

```
>>> import math as m
>>> m.factorial(3)
    6
>>> m.sqrt(16)
    4.0
>>> math.pi
    error
```

random module

→ it is an inbuilt module which can be used to perform some random or unpredictable operations.

```
>>> from random import*
>>> random()  (it is used to genirate a number
    0.4547      between range 0 & 1)

>>> randint(1,10)  (it is used to generate a integer
    7                number between a given range)
>>> randint(1000,9999)
    4523

>>> choice()  (it is used to selct a random value
                from collection)
>>> S=[2,3,9,8,9]
>>> choice(S)          >>> choice(s)
    8                       9

>>> shuffle()  (it is applicable only for list it is
                used to shuffle a value
                randomly)
```

```
>>> d=[2,9,'hello',64,83]
>>> shuffle(d)
>>> d
[9,'hello',83,2,64]
>>> shuffle(d)
>>> d
['hello',9,83,64,2]
```

userdefine function

① files present in same folder

Syntax:- ① import filename
filename.pname/mname

② from filename import pname/mname
pname/mname() (* to import every thing)

② accessing files from multiple folders

① from foldername import file1, file2,...

Access

filename.pname/mname()

② from packagname.filename import pname/mname

Generators :- it is a function which is used to give a sequence of value using yield keyword

Return

→ it is a keyword which is used to stop the function execution and return values from it

Yield

→ it is a keyword which is used to pause the function execution & continues the next instruction

→ it is used in normal userdefine function

→ it is used in generator function

→ typecasting is not required

→ typecasting is required

Note:- if a userdefine function consist of atleast one yield keyword it becomes Generator function

ex:-
```
def Jinke():
    print(47)
    return(30)
    print(16)
    return(27)
    print(20)
    return25
print(jinke())
```

O/P = 47
30

```
def Jinke():
    print(47)
    yield 30
    print(16)
    yield 27
    print(20)
    yield 25
print(list(jinke()))
```

O/P 47
16
20
[30, 27, 25]

① WAP extract all the string data item present in the list

```
def st-extract(li):
    out=[]
    for i in li:
        if type(i)==str:
            out+=[i]
    return out
print(st-extract(eval(input("enter a list"))))
```

by using yield

```
def ext(li):
    for i in li:
        if type(i)==str:
            yield i
print(list(ext(eval(input('enter:')))))
```

② WAP to get multipels of integer number between 1 to 10.

```
def multiple(n):
    for i in range(1,11):
        yield n*i
print(list(multiple(eval(input("enter:")))))
```

③ WAP gurator fibanoci Series of nth number.

```
def fibanoci(n)
    a=0
    b=1
    while n>a:
        yield a
        c=a+b
        b=c
        a=b
print('list(fibanoci(int(input('enter')))))
```

→

```
def fibo(n):
    a=0
    b=1
    for i in range(n)
        yiud a
        a, b = b, a+b
Print(list(fibo(8)))
```

generate prime number between range 1,100

```
def prime_check(n):
if n>=2:
for
```

```
def prime_check(n):
    if n==2:
        return True
    elif n>1:
        for i in range(2,n):
            if n%i==0:
                retur: False
    else:
        return True

def prime_no():
    for i in range(1,101):
        if prime_check(i):
            yield i

print(list(prime_no()))


def is_prime1(n):
    if n>1:
        out=[]
        for i in range(1,n+1):
            if n%i==0:
                out+=[i]
        return len(out)
def series_prime():
    for i in range(1,101):
```

```
        if is_prime1(i)==2:
            yield i
print(list(series_prime()))
```

Iterator

Iteration:- it is the phenomenon of traversing through the collection by fetching individual value

→ the function used for the iteration is called iterator

→ for loop consider as inbuilt iterator
for loop will traverse (start) from initial value, traverse through the collection one by one & stops when it reaches end of the collection

→ when it is required to traverse manually we use **iter()** and **next()** function

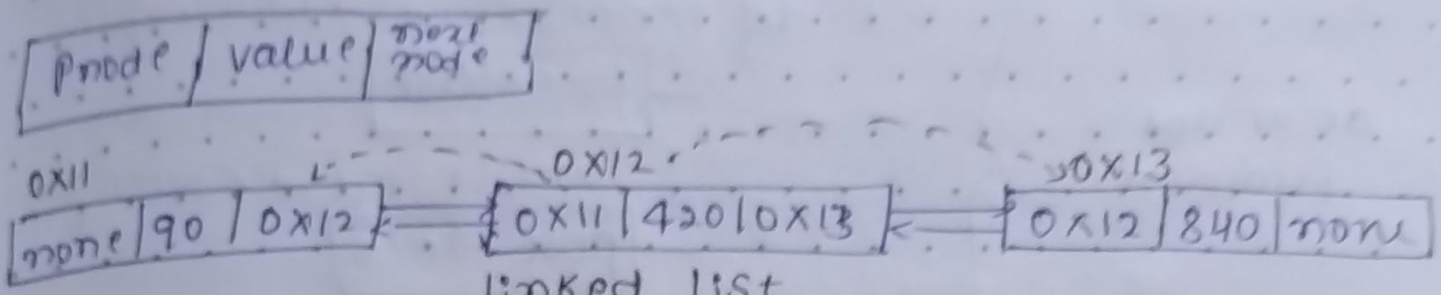
① **iter()** :- it is a function which is used to point toward a initial node address

Syntax :- `iter_var = iter(var/value)`

② **next()** it is a function which is used to fetch the value one-by-one

syntax - `next(iter_var)`

x = [90, 420, 840]



linked list

```
>>> x = [90, 420, 840]
>>> i = iter(x)
>>> i
<some address>
>>> next(i)
90
>>> next(i)
420
>>> next(i)
840
>>> next(i)
error
```

```
x = [90, 420, 840]
i = iter(x)
print(next(i))
print(next(i))
print(next(i))
```

note :- while using next() function through a collection and when it reaches end of the collection, if we try to fetch the next value it throws stop iteration error message

Decorator :- it is a function which is used to add additional features to the existing function it is of 2 types.

① inbuilt decorator ② userdefine decorator

① Inbuilt decorator :- ex:- classmethod, staticmethod, property, abstractmethod

user-defined-decorator :- it is created based on user requirement

Syntax:-

decorator function

```
def deconame(func):
    def inner(*args, **kwargs):
        → pre-task
        func(*args, **kwargs)
        → post task
    return inner
```

decorated function

```
@deconame
def fname():
    S.B.
```

Note:- it is not mandatory to have both pre and post task.

```
def instagram(func):
    def inner(*args, **kwargs):
        print('--login--')
        func(*args, **kwargs)
        print('--logout--')
    return inner

@instagram
def insta_kishan():
    print('reels')
    print('chat')

@instagram
def insta_dhyan():
    print('VC')
    print('stalking')

insta_kishan()
insta_dhyan()
```

O/P:-

```
--login--
reels
chat
--logout--
--login--
VC
stalking
--logout--
```

① WAP which calculate the total execution time of a function or program.

```
def exca-time(func):
    def inner(*args, **kwargs):
        import time
        start = time.time()
        fun((*args, **kwargs)
        end = time.time()
        return f'total time -{end-start}'
    return inner
@exca-time
def series():
    for q in range(1,101):
        print(q)
@exca-time
def rev-str(a):
    out = ' '
    for i in a:
        out = i+out
    print(out)
```

O/P
1
2
|
100
'total time - 0.284'
rev-str('deepa')
apeed
'total time - 0.062'

② WAP which give 5 second delay before function execution.

```
def exca-time(func):
    def inner(*args, **kwargs):
        import time
        time.sleep(5)
        fun(*args, **kwargs)
    return inner
@ exca-time
def series():
    for i in range(1,101):
        print(i)
```

O/P Series()
1
2
3
4
|
100.

3) WAP which gives positive result always when an arithmatic operation is perform

```
def arith-op(fname):
    def inner(*args, **kwargs):
        val = func(*args, **kwargs)
        return abs(val)
    return inner
@arith-op
def sub(a,b)
    print(10-20)
    return(a+b)
```

Comprehensions :- It is the phenomenon of creating a new collection by reducing a instruction it is done through list, set & dictionary

① list comprehension :- it is the phenomenon of creating a new list collection by reducing a instruction

Syntax :-
```
res = [var for var in coll]
print(res)
```

WAP to extract all the number between the range 1:10

```
out = []
for i in range(1,11):
    out += [i]
print(out)
```

ex :-
```
res = [i for i in range(1,11)]
print(res)
```

2) WAP to extract all the even no between range 1 to 20 by passing updation as 1

~~out=[]~~

```
res = [ i for i in range(1,21) if i%2==0]
print(res)
```

3) WAP to store square of the no if it is even, store cubes of the no if it is odd.

```
res = [i**2 if i%2==0 else i**3 for i in range(0,11)]
print(res)
res = [n**2 if n%2==0 else n**3 for i in
                    range(1,21)]
print(res)
```

Syntax :-

(2)
```
res = [TSB for var in coll if cond]
print(res)
```

(3)
```
res = [TSB if cond else FSB for var
        in coll]
print(res)
```

4) WAP to get following output

```
res = [len(i) if type(i) in [list,str,set,dict] else
        1 for i in ['male bartaithe',['yen
        madodu','Tea'],999,'old Monk', 2+5j]
print(res)
```

or

```
for i in eval(input("Enter"))
```

```
n = input("Enter:").split()
out = ''
for i in range(0, len(n)):
    if i%2!=0
        out += ''.join(n[i].upper())
    else:
        out += ''.join(n[i].lower())
print(out)
```

St = 'mALE enDA madhyana raja'

comprehension :- O/P = 'MALE enda MADHYANA raja'

```
print(''.join(i.upper() if i%2!=0 else ''.join(i.lower() for i in input("Enter:").split()))
```

```
St = input("Enter:").split()
out = [St[i].upper() if i%2==0 else St[i].lower()
       for i in range(len(St))]
print(''.join(out))
```

---

(4) I/P = NO O/P = [(1,10),(1,20),(1,30),(2,10),(2,20),(2,30)]

Normal :-

```
out = []
for i in range(1,3):
    for j in range(10,31,10):
        out += [(i,j)]
print(out)
```

```
res = [(i,j) for i in range(1,3) for j in
       range(10,31,10)]
print(res)
```

Syntax :- nested for loop

res = [(var1, var2) for var1 in coll for var2 in coll]

Set Comprehension :-

① res = {var for var in coll} ② res = {TSB for var in coll if cond}
print(res) print(res)

③ res = {TSB if cond else FSB for var in coll}
print(res)

④ res = {(var1, var2) for var1 in coll1 for var2 in coll2}
print(res)

① WAP to store qube of no if it is even, square of no if it is odd blw range 1,10

res = set()
{res.add(i**3) if i%2==0 else ~~out~~ res.add(i**2) for i in range(1,11)}
print(res) or

~~print({i for i in eval~~

② o/p [(1,10), (1,20), (1,30)]
o/p [1,10, 1,20, 1,30]

③ 'mungaru male yash'
o/p = {('mungaru', 3), ('male', 4), ('yash', 4)}

Ans :- ① print({i**3 if i%2==0 else i**2 for i in range(1,11)})

② print([i for i in eval(input('enter a list:')) for i in i])

3). print({i: len(i) if len(i)%2==0 else (i, len(i)//2) for i in input("enter:").split()})

Dictionary comprehension -> it is the phenomenon of creating new dictionary collection by reducing the instructions.

Syntax:- res = {K:V for var in coll}
print(res)

① WAP to store no & its square in the form of keyvalue between range 1 to 10
res = {i : i**2 for i in range(1, 11)}
print(res)

***

② WAP to store word and its length as key value pair present in string collection.

res = {i: len(i) for i in input("enter string").split()}
print(res)

③ WAP to store all the string data items along with its count of 'a' as key value pairs present in list collection

res = {i: i.count('a') for i in eval(input("enter list")) if type(i) == str}
print(res)

Syntax :-> res = {K:V for var in coll if cond}
print(res)

4). WAP to store no & its cubes if its even else square if it is odd.

Syntax :-
```
res = {k:v if cond else v for var in coll}
print(res)
```

③ res = {i:i**3 if i%2==0 else i**2 for i in range(1,11)}
print(res)

5) WAP to 'map' two list collection

```
x = eval(input("enter first:"))
y = eval(input("enter second:"))
out = {}
if len(x) > len(y):      or   low = min((len(x), len(y)))
    low = y
else:
    low = x
for i in range(len(low)):
    out[x[i]] = y[i]
print(out)
```

Zip function :- it is a function which used when multiple collection have to be passed in for loop

Syntax :-
```
for var1, var2 --- varn in zip(coll1, coll2 --- colln):
    S.B
```

it will traverse upto least length of collection.

ex:-
```
for i, j, k in zip('errorrrr', 'dhyan', 'msg'):
    print(i, j, k)
```

o/p =
```
e d m
r h s
r y g
```

```
for i, j, k in zip('errorrrr', [22, 33, 44], 'msg'):
    print(i, j, k)
```

o/p =
```
e 22 m
r 33 s
r 44 g
```

same question

5)
```
x=eval(input("enter:"))
y=eval(input("enter:"))
out={}
for i,j in zip(x,y):
    out[i]=j
print(out)
```

O/P
enter:['a','b','c']
enter:[2,3,4,5]
O/P
{'a':2,'b':3,'c':4}

using comprehension

```
res={i:j for i,j in zip(['a','b','c'],[2,3,4])}
print(res)
```

File Handling :-

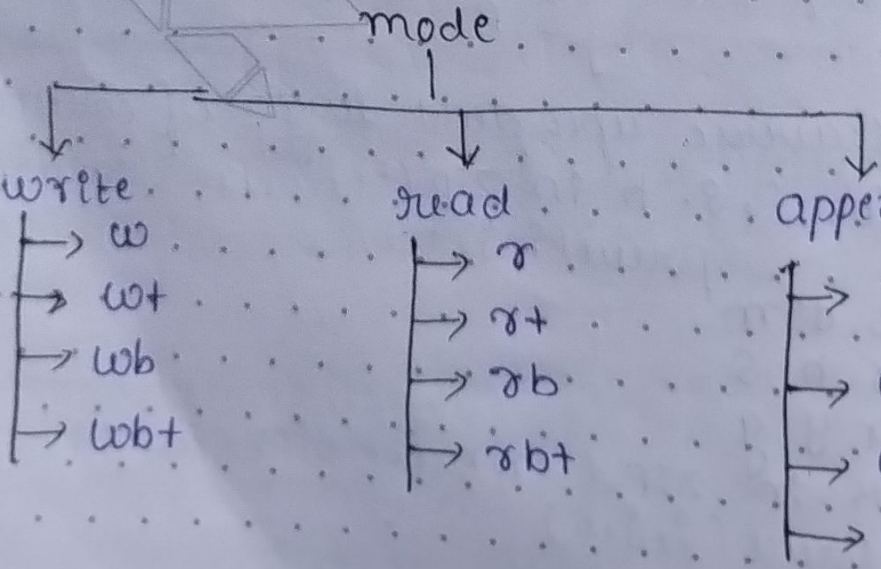
file :- it is a collection of data or placed where data's are stored.

→ based on the extension it is possible to identifie type of file

→ in python to access or open the file we have open function

Syntax :- var=open('filename/loc:ext','mode')

mode :- it is a type of operation performed on files



handling text file :-

* write mode :- when a file is opened using write mode, if the file is not existing it creates a new file
→ if the file is already existing it overrides the existing file

* write function() :- it is a function which is used to write the data into text file.

Syntax :- `var.write('data')`

using write function it is possible to write data at once

* writelines() :- it is a function which is used to write multiple lines of data into text file.

* close() :- it is a function which is used to close the file note :- if the file is not closed data will not be stored into the file.

Syntax :-
```
var.writelines(['data1', 'data2', ...'datan'])
var.close()
```

read mode :- it is a mode which is used to read the file
if file is not existing it throws a error message

read function :- it is a function which is used to read the complete data from the file at once

```
var1 = var.read()
print(var1)
```
syntax

readline() :- it is a function which is used to read a single line of data at once

```
var1 = var.readline()
print(var1)
```

readlines() :- it is a function which is used to read the complete data and gives output in the form of list collection

```
var1 = var.readlines()
print(var1)
```

ex :-

```
li = open('loveletter.txt', 'w')
li.write('Dear Hello\n')
li.write('How are you\n')
li.write('Good\n')
li.writelines(['ok\n', 'bye\n', 'Good night\n'])
li.write('Good evening')
li.close()


ki = open('loveletter.txt', 'r')
res = ki.read()
ki.close()
```

```
ki=open('loveletter.txt','r')
print(ki.readline())
print(ki.readline())
print(ki.readline())
ki.close()
```

```
ki=open('loveletter.txt','r')
res=ki.readlines()
print(res[2])
ki.close   # to read specific line
```

```
ki=open('loveletter.txt','r')
import time
res=ki.readlines()
for lines in res:
    print(lines)
    time.sleep(4)
```

Context Manager :-

→ using with keyword a file can be opened by using a variable

→ it is a Standad way to perform any operation on a file.

Syntax :-

```
with open('filename/loc.ext','mode')
                as var:
<------>    S.B
```

Note :- it is not necessary to close the file when it is opened using with keyword

wt

→ it is used to write and read the file.

→ if the file is not existing it creates a new file

rt

→ it is used to write and read the file.

→ if the file is not existing it throws a error message

* seek() :- it is a function which is used to make the cursor point towards specified index

note :- when wt or rt mode is used on file and written data, if user tries to read the file it gives empty line.

→ to avoid this seek() function is used.

append mode :- it is a mode which is used to add new data to the existing file

→ if the file is existing it adds the data, if the file is not existing it creates a new file.

example :-

```
with open('breakup.txt', 'wt') as a:
    a.write('taataa byee..hogbutt baa')
    a.seek(0)
    res = a.read()
    print(res)
```

```
with open('breakup.txt', 'rt') as a:
    a.write('tataa byee..hogbutt baa')
    a.seek(0)
    res = a.read()
    print(res)
```

```
with open('breakups.txt', 'a') as b:
    b.write('bithodalle hudgiii')
```

create mode :- It is a mode which makes the user to create a new file.

→ if the file is existing it throws a error message

```
with open(r"C:\Users\OSPR\Desktop\m35.txt") as a:
    x = a.read()
    print(x)

print('manjaa\nisha\trishaaa')
print(r'manjaa\nisha\trishaaa')
print('manjaa\nisha\trishaaa')
```

manjaa
isha rishaaa

```
with open(r"C:\Users\OSPR\Desktop\m35.txt", 'x') as a
    a.write('bitbudayyaa time aaithu')
```

Note :- back slash have special behaviors like \n \t , \b to avoid that \\ have to be used or raw string can be used.

Handling CSV file (comma Seperated value)

It is a file formate where the datas are Stored in real time to perform operation On CSV file we have to import CSV

Writing into CSV file

```
import csv
With open ('filename/loc.csv', 'w') as var:
    var1 = csv.write (var)
    var1.writerow (['data1', 'data2', ---, 'datan'])
            (or)
    var1.writerows ([['d1','d2','d3',---,'dn'],
                     ['d1','d2','d3',---,'dn'],
                     ---
                     ])
```

```
Writerow()
var.writerow (['data1','data2',---,'datan'])
```

```
Writerows()
var.writerows ([['data1','data2',---,'datan'],
                ['data1','data2',---,'datan'],
                ---
                ])
```

Reading from CSV

```
import csv
With open ('filename/loc.csv','r') as var:
    var1 = csv.reader (var)
    res = list (var1)
    print (res)
```

```
import csv
With Open('marks.csv','w',newline='')as x:
    y = csv.writer(x)
    y.writerow(['name','sub','marks','result'])
    y.writerow(['Sumanth','maths','70','p'])
    y.writerows([['Subangi','Science',85,'p'],
                 ['dayaak','bio',90,'p']])

With Open('marks.csv','r')as a:
    y = csv.reader(a)
    res = list(y)
    Print(res)
```

WAP to fech only unique words from a text file

```
R = Open('file22.txt','r')
res = R.read()
print(res)
res2 = res.Split()
out = Set(res2.Set())
for i in res2:
    out
```

① With open(r"c:\users\qspr\Desktop\m35.txt") as a:
    res = a.read().lower()
    New = Sorted(Set(res.split()))

To Store all the unique values in new text file.
With open(r'c:\users\qspr\Desktop\duplicate.txt','w')as x:
    for i in new:
        x.write(i)
        x.write('\n')

Create a CSV file having first row as name, age, Gender, percentage, Store minimum 4 Students data into it. Extract all the females who as scores more than 80% & Store that into new CSV file

```
→import csv
with open('file3.csv','w') as var:
    var1 = csv.writer(var)  → newline=''
    var1.writerow(['Name','age','Gen','per'])
    var1.writerows([['R','21','M','80'],
                    ['B','20','F','83'],
                    ['C','22','F','85'],
                    ['D','23','M','75']])
with open('file3.csv','r') as var2:
    var2 = csv.reader(var1)
    res = list(var2)
for i in range(1,len(res)):
    if res[i][2]=='M' and res[i][3]>='80':
        out+=[[res[i][0],res[i][2],res[i][3]]]
with open('Shin.csv','w') as p:
    q = csv.writer(p)
    q.writerows(out)
```

① Store the 10 most frequent words present in a text file

```
with open("C:\Users\asPR\Desktop\m35.txt") as Q:
    res = Q.read().split()
    c=0
    out = {}
    for i in res:
        out[i] = res.count(i)
```

```
def my_sort(x):
    return x[-1]
res1 = sorted(out.items(), key=my_sort, reverse=True)
top_ten = res1[0:10]
print(dict(top_ten))
```

Parsing Technique

→ it is the phenomenon of providing security to the data while sending from source to destination

→ it can be done in 2 ways.

① json parsing ② pickle parsing

① json parsing :→ it is the type of parsing technique which is used to provide security to the data by converting data into string format

* dumps() :- it is a function which is used to convert a original data into string format (encryption)

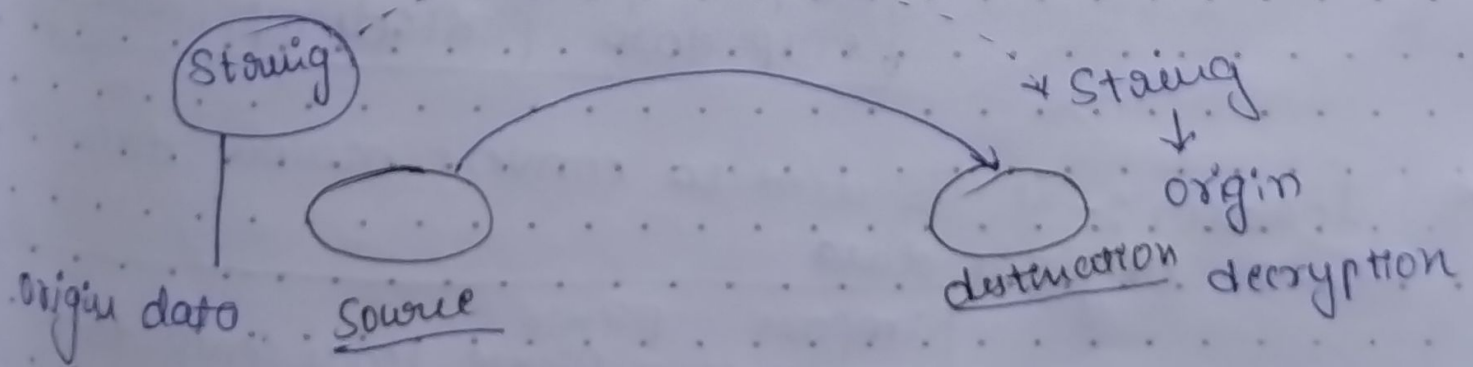
Syntax :-

```
import json
encdata = json.dumps(data)
```

* loads() :- it is a function which is used to convert a string data into original data (decryption)

Syntax :-

```
import json
org_data = json.loads(enc_data)
```



examples

```
a=[8,6,9]
import json
with open('json.txt','w') as f:
    enc_data = json.dumps(a)
    f.write(enc_data)
```

```
import json
with open('json.txt','r') as x:
    enc_data = x.read()
    org_data = json.loads(enc_data)
    print(type(enc_data),'enc')
    print(type(org_data),'orig')
```

o/p

<class 'str'> enc

<class 'list'> orig

Pickle parsing: it is the phenomenon of providing security

→ it is a type of parsing technique which converts data into binary format or bytes by providing security

* dumps(): - it is used to convert original data into by binary format (encryption)

```
import pickle
encp_data = pickle.dumps(data)
```

* loads(): - it is used to convert binary data into original data

```
import pickle
org_data = pickle.loads(encp_data)
```

example :- ① a=[8,6,9]

```
import pickle
with open('uppinkai.txt','wb') as l:
encp-data = pickle.dumps(a)
l.write(encp-data)
```

②
```
import pickle
with open('uppinkai.txt','rb') as x:
    encp-data = x.read()
    org-data = pickle.loads(encp-data)
    print(encp-data,'enc')
    print(org-data,'orig')
```

Backend Connectivity ISQL Connection

→ it is a phenomenon of connecting to a database file through python file

→ it can be doen using sqlite3 module.

* Connect() :- it is a function which is used to establish connection to a database file.

* cursor() :- it is a function which makes a user to write the queries by pointing to required position

* execute() :- it is used to execute the query

* Commit() :- it is used to save the queries

* close() :- it is used to close the database file.

Syntax :-

```
import Sqlite3
var = Sqlite3.connect('dbname.db')
var1 = var.cursor()
var1.execute('queries')
var.commit()
var.close()
```

example :-

```
import Sqlite3
x = Sqlite3.connect('criminals1.db')
y = x.cursor()
y.execute('create table jail (name varchar,
        age number, khaidi_no number)')
y.execute("insert into jail values('hintham', '21',
        6110)")
y.execute("insert into jail values('raghu', '22',
        '4200')")
x.commit()
x.close()
import Sqlite3
x = Sqlite3.connect('criminals1.db')
y = x.cursor()
y.execute("select * from jail where age>'21'")
resl = y.fetchall()
print(resl)
```

o/p = [('raghu', 22, 4200), ('pradeep', 22, 611)]

Exception Handling

* Exception :- it is an unauthorize event occur during execution of program

→ when exception is occured it stops the flow of execution, to avoid this it has to be handled

→ exception handling is a phenomenon of handling the exception. Such that it doesnot interrupt the flow of execution of program

→ in python to handle exception we use try and except

→ syntax error cannot be handle (compile time error)

try :- it is a keyword where in it is used to write the program ~~except block~~

except block :- the solution for the exception occured in try block will be given here

exception handling of 3 types

① Specific exception handling
② Generic " "
③ Default " "

① Specific exception Handling :- It is a type of E.H which is used when an exception is known

Syntax :-

```
try :
    program
except ExceptionName :
    Solution
```

→ The exception name used in it must be a standared exception name.

ex :- ZeroDivisionError , AttributeError
ValueError , NameError

Note :- for a single try block multiple except block can be written

ex :-
```
def handle():
    try:
        a=int(input("enter:"))
        b=int(input("enter:"))
        print(a/b)
    except ZeroDivisionError:
        print('thupakk denominator o kodbedvoo')
    except ValueError as a:
        print('given different data type')
        print(a)
handle()
print('namskara guru')
```

op enter : 6
enter : 3
2.0

>>> handle()
enter : 3
enter : 0
namskara gareee thupakk denominator 0 kodbedvo

>>> handle()
enter : 'py'
given different data type

Syntax :- `except ExceptionName as Var:`

→ using the above syntax the message of the exception can be stored to a variable

Generic Exception handling :- it is a type of exception handling which handles all the errors except keyboard interupt error.

Syntax :-
```
try:
    program
except Exception:
    solution
```

ex:-
```
try:
    while True:
        print('Vinayashree')
except Exception:
    print('some errorr')
```

<u>default Exception Handling</u> :- it is a type of exception handling which handle all the error include keyboard interupt error

Syntax:-

```
try:
    program
except:
    solution
```

ex:-

```
try:
    while True:
        print("Vinayashree")
except:
    print('some error')
```

Note:- it is possible to pass else block for try and except

① except block will execute when an error is occured in try block

② else block will be executed when there is no error in try block

③ finally:- it is a block which executes wether an error is occured or not

ex:-

```
try:
    a=int(input("enter:"))
    b=int(input("enter:"))
    print(a/b)
except:
    print('guruu geruu tapp aithee')
else:
    print('no errors')
finally:
    print('nand yeu iduii')
```

o/p = enter : 4
enter : 0
guruu yenu kapp aithu
nond yelu idiu

custom Exception

in python it is possible to throw a error message using raise keyword

syntax :-

```
raise ErrorName('message')
```

(message → optional)

the errorname used must be standard name

user define Exception :- in python it is possible to create a exception name based on the user.

syntax :-

```
class cname(Exception):
    pass

raise cname('message')
```

ex:

```
a = int(input('enter: '))
if a%2==0:
    print('even')
else:
    raise ValueError
```

o/p Enter : 4
even

o/p : enter : 9
ValueError

```
class subbi_dovy(Exception):
    pass

nod = int(input('enter no. of dovy:'))
if nod<=9:
    print('you are shubhamsi')
else:
    raise subbi_dovy("yeht beke ninge")
```

op: ① enter no. of dovy: 4
you are shubhamsi

② enter no. of dovy: 12
Error
shubbi_dovy: yehte beke ninge

Regular Expression / Regex

→ when a required data have to be extracted from huge data regular expression will be used

→ required pattern have to be written to fetch the data

pattern

[A-Z] → one uppercase

[a-z] → one lower case

\d [0-9] → one digit

\w [A-Za-z0-9] → one alphanumeric

[A-Z]{n} → n number of uppercase

[A-Z]{m,n} → m to n no of uppercase

\s → one space

\S → one non-space

+ → 1 to n of character

* → 0 to n of character

? → 0 or 1 no of character

\ → removes special behaviour of special char

1) write a pattern to math phone no

first digit	Nine digit
6	0
7	
8	
9	9

[6-9][0-9]{9}

1st dig — remaining 9-digit

2) WApattern to math PAN number

AXEPY8032F

[A-Z]{5}[0-9]{4}[A-Z]

3) WApattern to match Vehicle number

① KA-13-Z-6106 ② KA-16-KD-2080

[A-Z]{2}-[0-9]{2}-[A-Z]{1,2}-[0-9]{4}

Note :- in regular expression except - some special character will have some behaviour.

→ when other special character have to be match \ have to be written

for USN

① 1DB24CV051

[0-9]{1,4}[A-Z][0-9]{2}[A-Z]{2}[0-9]{3}

[1-4]

② SW20CSE090

[A-Z]{2}[0-9]{2}[A-Z]{3}[0-9]{3}

③ Date 23-Oct-2024

[0-9]{2}-[a-z]{3}-[0-9]{4} or {2,4}

④ 12:32:42 (time)

[0-9]{2}\:[0-9]{2}\:[0-9]{2}

\d \d \d

⑤ KARB000032 (IFSC)

[A-Z]{4}[0-9]{6}

① r.deena6364@gmail.com

vijay.S@pyspiders.in

\w+\.\w+\@[a-z]{5}\.[a-z]{2,3}

② darshanseena6106@gmail.com
gokul_linga@gmail.com

\w+.\-?\@[a-z]{5}\.[a-z]{3}

for all the 4 email the below pattern will match (.co.in)

\w+\-*\.?\w*\@[a-z]+\.[a-z]{2,3}\.?[a-z]*

how to extract a pattern.

s = 'pooja welcome 9878765577575755 and poojashri @gmail.com 8088802427'

```
import re
phone = re.findall(r'\b[6-9][0-9]{9}\b', s)
print(phone)
```