Encapsulation:

class student:

def \_\_init\_\_(self,marks):#default constructor

self.marks=marks

self.\_\_marks=marks #private

def getter(self):

return self.\_\_marks

def setter (self,marks):

self.\_\_marks=marks

s1=student(67)

# set the data

s1.setter(67)

# get the data

ans=s1.getter()

print(ans)

o/p: 67

abstraction :

from abc import ABC,abstractmethod

class four\_wheeler(ABC):

@abstractmethod

def engine():

pass

class swift(four\_wheeler):

def car\_start():

return"car is moving"

car\_1=swift

ans=car\_1.car\_start()

print(ans)

single inheritance:

#single inheritance

class father:#parent class

def father\_method():

return"this is father method"

#inheriting father class

class child (father):

def child\_method():

return "this is child method"

parent\_object=father

child\_object=child

print(parent\_object.father\_method())

print(child\_object.child\_method())

print(child\_object.father\_method())

multi level ihetitance:

#multiple ihetitance

class father:

def father\_method():

return "this is father method"

class mother:

def mother\_method():

return"this is mother method"

class child(father,mother):

def child\_method():

return "i have propeties of mother and father"

father\_object=father

mother\_object=mother

child\_object=child

print(father\_object.father\_method())

print(mother\_object.mother\_method())

print(child\_object.child\_method())

print(child\_object.father\_method())

print(child\_object.mother\_method())

multipul inheritance

#multipul inheritance

class gfather:

def gfather\_method():

return"this is gfather mehod"

class father(gfather):

def father\_method():

return "this is father method"

class child (father):

def child\_method():

return "i have properties of father"

gfather\_object=gfather

father\_object=father

child\_object=child

print(gfather\_object.gfather\_method())

print(father\_object.father\_method())

print(child\_object.child\_method())

polymer phrism:

class animal:

def speak():

return "ANIMAL IS SPEAKING "

class bird(animal):

def speak():

return "bird is speaking"

animal\_object=animal

bird\_object=bird

print(animal\_object.speak())

print(bird\_object.speak())

stack:

# Enter your code here

def push(value):

top=1

if(top==4):

return "stack is full"

else:

top=top+1

return stack.append(value)

stack=[]

push(10)

push(20)

push(30)

push(40)

print(stack)