**30/03/2019 Lecture22**

**OOPS revision.**

**Constructor : init method**

**Destructor:** method of class which get implecitly called when object gets out of scope

**Simple class in python**

Constructor in python : \_\_init\_\_

Destructor in python : \_\_del\_\_

**#Stack implementation**

class Stack:

def \_\_init\_\_(self,size):

print("Stack Constructed of size %d"%size)

self.stack=[]

self.size=size

def \_\_del\_\_(self):

print("Stack destructed")

del self.stack

def isempty(self):

return self.stack == []

def isfull(self):

return self.stack==self.size

def push(self,data):

status="FAILED"

if not self.isfull():

self.stack.append(data)

status="SUCCESS"

return status

def pop(self):

status="FAILED"

data=-1

if not self.isempty():

data=self.stack.pop()

status="SUCCESS"

return data,status

def main():

st=Stack(10)

data=input("Enter data to push")

print(st.push(data))

print(st.pop())

**Assignment**

**#Implement menu driven on stack**

**#WAP to implement queue(append and to remove take first position and remove)**

To make variables private add ‘\_\_’ to variable. e.g.\_\_stack,\_\_size e.g.

print st.\_Stack\_\_arg

in python class is not template, it is an object.

Python makes runtime classes which makes it powerful language.

Every object internally is a dictionary e.g.

print("object dictionary",st.\_\_dict\_\_)

print("Class dictionary",Stack.\_\_dict\_\_)

**Runtime object variable assignment**

st=Stack(10)

st1=Stack(10)

st.interesting=True

print("object dictionary",st.\_\_dict\_\_)

print("object dictionary",st1.\_\_dict\_\_)

**#WAP to implement a complex number on your own. FOllwoing operation should be supported**

**#1.Add complex number,**

**#2.substract complex number,**

**#3.multiply complex number by an integer**

class Complex:

def \_\_init\_\_(self,r=0,i=0):

self.real=r

self.imaginary=i

def \_\_del\_\_(self):

print("Stack destructed")

del self.real

del self.imaginary

def add(self,complex2):

return Complex(self.real+complex2.real,self.imaginary+complex2.imaginary)

def sub(self,complex2):

return Complex(self.real-complex2.real,self.imaginary-complex2.imaginary)

def \_\_repr\_\_(self):

return str(self.real)+"+"+str(self.imaginary)+"i"

def main():

cn=Complex(10,20)

print(cn.add(Complex(2,3)))

print(cn.sub(Complex(1,2)))

if \_\_name\_\_=='\_\_main\_\_':

main()