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

Index Object Oriented Principles or Features or Concepts

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

Index:

---------

=>What are the advantages of OOPs

=>List of Object-Oriented Principles

1. Classes

2. Objects

3. Data Encapsulation

4. Data Abstraction

5. Inheritance

6. Polymorphism

7. Message Passing(already)

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

1. Classes

=>Importance and purpose of Classes concept

=>Syntax for Defining Class

=>Types of Data Members

a) Instance Data Members

b) Class Level Data Members

=>Types of Methods

a) Instance Method

b) Class Level Method

c) Static Method

=>What is "self" and "cls"

=>Programming Examples

2. Object

=>Importance and purpose of Object Concept

=>Syntax for creating Object

=>Programming Examples

=>PRAMMING Examples related to pickling and Data base communication with Classes and objects.

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

=>Constructors in OOPs

=>Importance and purpose of Constructors

=>Types of Constructors

a) Default Constructors

b) Parameterized Constructors

=>Rules for Constructors

=>Programming Examples

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

=>Destructors in OOPs with Garbage Collector

=>Importance and purpose of Detstructrors

=>Syntax for defining Detstructrors

=>Internal flow of Detstructrors

=>relation between Detstructrors and Garbage Collector

=>gc module

=>Pgramming Examples

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

3. Data Encapsulation and Data Abstraction

=>Importance and purpose of Data Encapsulation

=>Importance and purpose of Data Abstraction

=>Implementation of data encapsulation and Data Abstraction

=>Programming Examples

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

5. Inheritance

=>Importance and purpose of Inheritance

=>Types of Inheritances

a) single

b) multi-level

c) hierarchical

d) multiple

e) Hybrid

=>Syntax for Inheritance

=>Programming Examples

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

Method Overriding in OOPs

=>Importance and purpose of Method Overriding

=>memory management in Method Overriding

=>Programming Examples

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

6.Polymorphism

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

=>Importance and purpose of Polymorphism

=>Difference between Polymorphism and Inheritance

=>Method Overring with Polymorphism

=>super () and class name approaches in Polymorphism

=>Programming Examples

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

Object Oriented Principles or Features or Concepts

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

=>In real time, to develop any project or application, we must choose a language and it can satisfy two types of principles. They are

1. Functional(Procedure) Oriented Principles-----C,Pascal, cobol,8086,oracle7.3,PYTHON

2. Object Oriented Priciples. --------PYTHON C++ JAVA, .NET, from Oracle8 onwadrs.......

=>Even though, PYTHON programming Belongs to both Functional and Object Oriented Programming language and internally everything is treated as object.

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

"Every Thing is an object " --Benifits

(OR)

Adavtanges of Object-Oriented Principles

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

1. Objects allows us to store Large Volume of Data (Platform Independent)

2. The Data is visiting between two machines in the form of Ciphet Text (encrypted Format)So that we can achieve the Security

3. The Large Volume of Data can be transfered between multiple machines all at once in the form of objects and obtains effective communication.

4. With Objects we can build high effective Re-Usable Applications (Redundency of the Code is Minimized).

5. The Data is always available arround Objects (Effective Memory Usage) and functions can operate on the objects.

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

List of Object Oriented Principles

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

=>To say a languge is Object Oriented, It has to sastisfy the following Principles.

1. Classes

2. Objects

3. Data Encapsulation

4. Data Abstraction

5. Inheritance

6. Polymorphism

7. Message Passing(already discussed )

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

Classes

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

=>The Purpose of Classes Concept is that "To Develop Programmer-Defined Data Type To develop any real Time Application in OOPs"

=>The Purpose of Developing Programmer-Defined Data Type is that "To Store Customized Data and To Perform Customized Operations".

=>To Develop Programmer-Defined Data Type by using Classes concept, we use "class" keyword

=>In Python Programming, All Class Names are treated as Programmer-Defined Data Type.

=>Every Program in OOPs, must starts with Classes concept.

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

Definition:

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

=>A Class is a collection Variables (Data Members) and Methods.

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

=>When we define a class, Memory space is not created for Data Members and Methods but Whose Memory Space is created when we create an object w.r.t Class Name.

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

Syntax for Defining a class in Python

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

class <class-name>:

Class Level Data Members

def InstanceMethodName(self,List of formal params if any):

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

Specify Instance Data Members---Perform Specific Operations

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

@classmethod

def classlevelmethodname(cls,list of Formal params if any)

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

Specify Class Level Data Members--Perform Common Operations

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

@staticmethod

def staticmethodname(list of formal params if any):

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

Performs Utility / Universal Operations

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

Types of Data Members in a Class

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

=>In class of Python, we can define Two Types of Data Members. They are

1. Instance Data Members

2. Class Level Data Members

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

1. Instance Data Members

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

=>Instance Data Members are those which are used for Storing Specific Values

=>Instance Data Members are those whose memory space created Every Time when an object is crated.

=>Instance Data Members Must be Accessed w.r.t Object Name or self

ObjName . InstanceData Member Name

(OR)

self.InstanceData Member Name

=>To Add the Instance Data Members to the corresponding object, we use 3 approaches. They are

a) By Using Object Name

b) By Using Instance Methods

c) By using Constructors.

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

2. Class Level Data Members

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

=>Class Level Data Members are those which are used for Common Values.

=>Class Level Data Members are those Whose Memory space created One time irrespective of Number of Objects are Created.

=>Class Level Data Members can be accssed w.r.t Class Name.

Class Name.Class Level Data Member Name

(OR)

cls.Class Level Data Member Name

(OR)

ObjectName.Class Level Data Member Name

(OR)

self.Class Level Data Member Name

=>To Add the Class Level Data Members we use 2 approaches. They are

a) Inside of Class Definition

b) Inside of Class Level Method.

#Program for storing sno,sname ,marks in an object of Programmer-defined class

#StudEx1.py----Instance Data members

class Student:pass

#main program

s1=Student()

s2=Student()

print("Content of s1 before adding the data={} and Number of values={}".format(s1.\_\_dict\_\_, len(s1.\_\_dict\_\_)))

print("Content of s2 before adding the data={} and Number of values={}".format(s2.\_\_dict\_\_, len(s2.\_\_dict\_\_)))

print("-"\*50)

#add Instance Data Members to s1

s1.sno=10

s1.name="RS"

s1.marks=22.22

print("Content of s1 after adding the data={} and Number of values={}".format(s1.\_\_dict\_\_, len(s1.\_\_dict\_\_)))

#add Instance Data Members to s2

s2.stno=20

s2.sname="TR"

s2.smarks=42.22

print("Content of s2 after adding the data={} and Number of values={}".format(s2.\_\_dict\_\_, len(s2.\_\_dict\_\_)))

#Program for storing sno,sname ,marks in an object of Programmer-defined class

#StudEx2.py----Instance Data members and Class Level Data Members

class Student:

crs="PYTHON" # here crs is called Class Level Data member

#main program

s1=Student()

s2=Student()

print("Content of s1 before adding=",s1.\_\_dict\_\_)

print("Content of s2 before adding=",s2.\_\_dict\_\_)

#Read Instance Data members such as sno,sname and marks to s1

print("-"\*50)

print("First Student Information")

print("-"\*50)

s1.sno=int(input("Enter Student Number:"))

s1.sname=input("Enter Student Name:")

s1.marks=float(input("Enter Student Marks:"))

print("-"\*50)

print("Second Student Information")

print("-"\*50)

s2.sno=int(input("Enter Student Number:"))

s2.sname=input("Enter Student Name:")

s2.marks=float(input("Enter Student Marks:"))

print("-"\*50)

print("\nContent of s1 after adding=",s1.\_\_dict\_\_)

print("Content of s2 after adding=",s2.\_\_dict\_\_)

#Program for storing sno,sname ,marks in an object of Programmer-defined class

#StudEx3.py----Instance Data members and Class Level Data Members

class Student:

crs="PYTHON" # here crs is called Class Level Data member

#main program

s1=Student ()

s2=Student ()

print("Content of s1 before adding=",s1.\_\_dict\_\_)

print("Content of s2 before adding=",s2.\_\_dict\_\_)

#Read Instance Data members such as sno,sname and marks to s1

print("-"\*50)

print("First Student Information")

print ("-"\*50)

s1.sno=int(input("Enter Student Number:"))

s1.sname=input("Enter Student Name:")

s1.marks=float(input("Enter Student Marks:"))

print("-"\*50)

print("Second Student Information")

print("-"\*50)

s2.sno=int(input("Enter Student Number:"))

s2.sname=input("Enter Student Name:")

s2.marks=float(input("Enter Student Marks:"))

print("-"\*50)

print("First Student Information:")

print("-"\*50)

print("\tStudent Number:{}".format(s1.sno))

print("\tStudent Name:{}".format(s1.sname))

print("\tStudent Marks:{}".format(s1.marks))

print("\tStudent Course:{}".format(Student.crs))# accessing Class Level Data member w.r.t Class Name

print("-"\*50)

print("Second Student Information:")

print("-"\*50)

print("\tStudent Number:{}".format(s2.sno))

print("\tStudent Name:{}".format(s2.sname))

print("\tStudent Marks:{}".format(s2.marks))

print("\tStudent Course:{}".format(Student.crs))## accessing Class Level Data member w.r.t Class Name

print("-"\*50)

#Program for storing sno,sname ,marks in an object of Programmer-defined class

#StudEx4.py----Instance Data members and Class Level Data Members

class Student:

crs="PYTHON" # here crs is called Class Level Data member

#main program

s1=Student()

s2=Student ()

print("Content of s1 before adding=",s1.\_\_dict\_\_)

print("Content of s2 before adding=",s2.\_\_dict\_\_)

#Read Instance Data members such as sno,sname and marks to s1

print("-"\*50)

print("First Student Information")

print("-"\*50)

s1.sno=int(input("Enter Student Number:"))

s1.sname=input("Enter Student Name:")

s1.marks=float(input("Enter Student Marks:"))

print("-"\*50)

print("Second Student Information")

print("-"\*50)

s2.sno=int(input("Enter Student Number:"))

s2.sname=input("Enter Student Name:")

s2.marks=float(input("Enter Student Marks:"))

print("-"\*50)

print("First Student Information:")

print("-"\*50)

print("\tStudent Number:{}".format(s1.sno))

print("\tStudent Name:{}".format(s1.sname))

print("\tStudent Marks:{}".format(s1.marks))

print("\tStudent Course:{}".format(s1.crs)) # accessing Class Level Data member w.r.t object Name

print("-"\*50)

print("Second Student Information:")

print("-"\*50)

print("\tStudent Number:{}".format(s2.sno))

print("\tStudent Name:{}".format(s2.sname))

print("\tStudent Marks:{}".format(s2.marks))

print("\tStudent Course:{}".format(s2.crs))# accessing Class Level Data member w.r.t object Name

print("-"\*50)

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

Types of Methods in class of Python

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

=>In a class of Python, we can define 3 types of Methods. They are

1. Instance Methods

2. Class Level Methods

3. Static Methods.

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

1. Instance Methods

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

=>Instance Methods are used for performing Specific Operation or Operation on Instance Data Members of Object and these Methods are also called Object Level Methods.

=>Instance Methods always Takes "self" as a First Formal Parameter.

=>The Syntax for Instance Method

def InstanceMethodName (self,list of formal params if any)

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

Specific Operations

Specify Instance Data Members

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

=>All Instance Methods Must be accessed w.r.t Object Name or self

ObjectName.InstanceMethod Name()

(OR)

self.InstanceMethod Name()

=>What is "self"?

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

=>"self" is one of the First Formal Parameter in Instance Methods

=>"self" contains Reference / Address of Current Class Object

=>"self" can be used Inside of Corresponding Instance Method Body Only But not Possible to access other part of Program

-----------------------------------------------------------------------2. Class Level Methods

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

=>Class Level Methods are used for performing Common Operation on Class Level Data Members.

=>In Order to define Class Level Methods, whose defeintion must be preceded with a pre-defined decorator called @classmethod and takes "cls" as a First Formal Parameter.

=>The Syntax for Class Level Method is

@Classmethod

def classlevelmethod(cls,list of formal params if any):

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

Specify Class Level Data Memebrs

Common Operations

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

=>All Class Level Methods can be accessed w.r.t Class Name or cls or object name or self

classname.classlevelmethod()

(OR)

cls.classlevelmethod()

(OR)

objname.classlevelmethod()

(OR)

self.classlevelmethod()

=>What is "cls"?

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

=>"cls" is one of the First Formal Parameter in Class Level Methods

=>"cls" contains Current Class name

=>"cls" can be used Inside of Corresponding Class Level Method Body Only But not Possible to access other part of Program

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

3. Static Methods

-----------------------------------------------------------------------=>Static Methods are used for performing Universal Operations or Utility Operations

=>Static Methods defeinition must be preceded with a predefined decorator called

@Staticmethod and it never takes "cls" or "self" but always takes object of other classes.

=>The Syntax for Static Method is

@staticmethod

def staticmethodname(list of Formal Params):

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

Utility Operation / Universal Operations

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

=>Static Methods can be accessed w.r.t Class Name or object name.

ClassName.static method name()

(OR)

ObjectName.static method name()

#Program for Reading sno,sname ,marks in an object of Programmer-defined class w.r.t Instance Methods

#InstanceMethodEx1.py

class Student:

def getstuddata(self):

print("-"\*40)

self.sno=int(input("Enter Student Number:"))

self.sname=input("Enter Student Name:")

self.marks=float(input("Enter Student Marks:"))

print("-"\*40);..

def dispstuddata(hyd):

print("-"\*40)

print("\tStudent Number:{}".format(hyd.sno))

print("\tStudent Name:{}".format(hyd.sname))

print("\tStudent Marks:{}".format(hyd.marks))

print("-"\*40)

#main program

s1=Student()

s2=Student()

print("Content of s1 before reading=",s1.\_\_dict\_\_)

print("Content of s2 before reading=",s2.\_\_dict\_\_)

print("-"\*50)

print("Enter First Student Information:")

s1.getstuddata()

print("Enter Second Student Information:")

s2.getstuddata()

print("Details of First Student:")

s1.dispstuddata()

print("Details of Second Student:")

s2.dispstuddata()

#Program for Reading sno,sname ,marks in an object of Programmer-defined class w.r.t Instance Methods

#InstanceMethodEx2.py

class Student:

crs="PYTHON" # Class Level Data Member

def getstuddata(self):

print("-"\*40)

self.sno=int(input("Enter Student Number:"))

self.sname=input("Enter Student Name:")

self.marks=float(input("Enter Student Marks:"))

print("-"\*40)

self.dispstuddata() # calling Instance method from another instance method

def dispstuddata(self):

print("-"\*40)

print("\tStudent Number:{}".format(self.sno))

print("\tStudent Name:{}".format(self.sname))

print("\tStudent Marks:{}".format(self.marks))

print("\tStudent Course name:{}".format(self.crs))

print("-"\*40)

#main program

s1=Student()

s2=Student()

print("Content of s1 before reading=",s1.\_\_dict\_\_)

print("Content of s2 before reading=",s2.\_\_dict\_\_)

print("-"\*50)

print("Enter First Student Information:")

s1.getstuddata()

print("Enter Second Student Information:")

s2.getstuddata()

#Program for demonstrating Class Level Methods and Class Level Data members

#ClassLevelMethodEx1.py

class Student:

@classmethod

def getcourse(cls):

cls.crs="PYTHON"

@classmethod

def getDeveloper(cls):

Student.dev="Rossum"

#main program

Student.getcourse() # calling Class Level method w.r.t class name

Student.getDeveloper() # calling Class Level method .w.r.t classname

s1=Student()

s2=Student()

print(s1.crs,s1.dev)

print(s2.crs,s2.dev)

#Program for demonstrating Class Level Methods and Class Level Data members

#ClassLevelMethodEx2.py

class Student:

@classmethod

def getcourse(cls):

cls.crs="PYTHON"

@classmethod

def getDeveloper(cls):

Student.dev="Rossum"

#main program

s1=Student()

s2=Student()

s1.getcourse() # calling Class Level method w.r.t object name

s2.getDeveloper() # calling Class Level method w.r.t object name

print(s1.crs,s1.dev)

print(s2.crs,s2.dev)

#Program for demonstrating Class Level Methods and Class Level Data members

#ClassLevelMethodEx3.py

class Student:

@classmethod

def getcourse(cls):

cls.crs="PYTHON"

cls.getDeveloper() # Calling Class Level Method w.r.t cls

@classmethod

def getDeveloper(cls):

Student.dev="Rossum"

#main program

Student.getcourse() # Calling Class Level Method w.r.t Class Name

s1=Student()

s2=Student()

print(s1.crs,s1.dev)

print(s2.crs,s2.dev)

#Program for demonstrating Class Level Methods and Class Level Data members

#ClassLevelMethodEx4.py

class Student:

@classmethod

def getcourse(cls):

cls.crs="PYTHON"

cls.getDeveloper("ROSSUM") # Calling Class Level Method w.r.t cls

@classmethod

def getDeveloper(cls,dname):

Student.dev=dname

def getstudentdet(self,sno,sname,marks):

self.sno=sno

self.sname=sname

self.marks=marks

def dispstuddata(self):

self.getcourse() # calling Class Level Method Name w.r.t self

print("-"\*40)

print("\tStudent Number:{}".format(self.sno))

print("\tStudent Name:{}".format(self.sname))

print("\tStudent Marks:{}".format(self.marks))

print("\tStudent Courrse Name:{}".format(self.crs))

print("\tCourrse Dev By:{}".format(self.dev))

print("-"\*40)

#main program

s1=Student()

s2=Student()

s1.getstudentdet(10,"RS",11.11)

s2.getstudentdet(20,"TR",21.11)

s1.dispstuddata()

s2.dispstuddata()

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

3. Static Methods

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

=>Static Methods are used for performing Universal Operations or Utility Operations

=>Static Methods defeinition must be preceded with a predefined decorator called

@staticmethod and it never takes "cls" or "self" but always takes object of other classes.

=>The Syntax for Static Method is

@staticmethod

def staticmethodname(list of Formal Params):

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

Utility Operation / Universal Operations

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

=>Static Methods can be accessed w.r.t Class Name or object name.

ClassName.static method name()

(OR)

ObjectName.static method name()

(OR)

self.static method name()

#StaticMethodEx1.py

class Student:

def getstuddet(self):

self.sno=int(input("\nEnter Student Number:"))

self.sname=input("Enter Student Name:")

class Employee:

def getempdet(self):

self. Eno=int(input("\nEnter Employee Number:"))

self.ename=input("Enter Employee Name:")

self.sal=input("Enter Employee Salary:")

class Teacher:

def getteacherdet(self):

self.tno=int(input("\nEnter Teacher Number:"))

self.ename=input("Enter Teacher Name:")

self.subject=input("Enter Teacher Subject:")

class Hyd:

@Staticmethod

def dispobjectdata(kvr,pinfo):

print ("-"\*50)

print("Information about:{}".format(pinfo))

for k,v in kvr.\_\_dict\_\_.items():

print("\t{} {}".format(k,v))

print ("-"\*50)

#main program

s=Student()

e=Employee()

t=Teacher()

s.getstuddet()

e.getempdet()

t.getteacherdet()

#calling Static Method w..r.t Class Name

Hyd.dispobjectdata(s,"Student")

Hyd.dispobjectdata(e,"Employee")

Hyd.dispobjectdata(t,"Teacher")

#StaticMethodEx2.py

class Student:

def getstuddet(self):

self.sno=int(input("\nEnter Student Number:"))

self.sname=input("Enter Student Name:")

class Employee:

def getempdet(self):

self.eno=int(input("\nEnter Employee Number:"))

self.ename=input("Enter Employee Name:")

self.sal=input("Enter Employee Salary:")

class Teacher:

def getteacherdet(self):

self.tno=int(input("\nEnter Teacher Number:"))

self.ename=input("Enter Teacher Name:")

self.subject=input("Enter Teacher Subject:")

class Hyd:

@staticmethod

def dispobjectdata(kvr,pinfo):

print("-"\*50)

print("Information about:{}".format(pinfo))

for k,v in kvr.\_\_dict\_\_.items():

print("\t{} {}".format(k,v))

print("-"\*50)

#main program

s=Student()

e=Employee()

t=Teacher()

s.getstuddet()

e.getempdet()

t.getteacherdet()

#calling Static Method w..r.t Object Name

H=Hyd()

H.dispobjectdata(s,"Student")

H.dispobjectdata(e,"Employee")

H.dispobjectdata(t,"Teacher")

#StaticMethodEx3.py

class Student:

def getstuddet(self):

self.sno=int(input("\nEnter Student Number:"))

self.sname=input("Enter Student Name:")

self.dispobjectdata(self,"Student") # Calling Static Method w.r.t self

@staticmethod

def dispobjectdata(kvr,pinfo):

print ("-"\*50)

print ("Information about:{}".format(pinfo))

for k,v in kvr.\_\_dict\_\_.items():

print("\t{} {}".format(k,v))

print("-"\*50)

class Employee:

def getempdet(self):

self.eno=int(input("\nEnter Employee Number:"))

self.ename=input("Enter Employee Name:")

self.sal=input("Enter Employee Salary:")

Student.dispobjectdata(self,"Employee") #Calling Static Method w.r.t class name

class Teacher:

def getteacherdet(self):

self.tno=int(input("\nEnter Teacher Number:"))

self.ename=input("Enter Teacher Name:")

self.subject=input("Enter Teacher Subject:")

Student.dispobjectdata(self,"Teacher") #Calling Static Method w.r.t class name

#main program

s=Student()

e=Employee()

t=Teacher()

s.getstuddet()

e.getempdet()

t.getteacherdet()

"""write a python program which will accept student details such as student number student name,marks in three subjects.

calculate the total marks

calculate the Percentage and give the grades.(fail/pass)

give the grade = #fail provided student secured less than 40 in any of the three subjects.

give the grade = #distinction provided student totall marks lies within 250 - 300

give the grade = #First class provided totall marks lies within 200- 249

give the grade= second class provided totall marks lies within 150- 199

give the grade= third class provided totall marks lies within 120- 149

save the students result in the database."""

#StudentOOPsDataBase.py

import cx\_Oracle

class Student:

def getstuddet(self):

self.sno=int(input("Enter Student Number:"))

self.sname=input("Enter Student Name:")

#validation of C Marks

while(True):

self.cm=int(input("Enter Marks in C(100):"))

if(self.cm>=0) and (self.cm<=100):

break

#Validation of C++ Marks

while(True):

self.cppm=int(input("Enter Marks in C++(100):"))

if(self.cppm>=0) and (self.cppm<=100):

break

#validation of Python Marks

while(True):

self.pym=int(input("Enter Marks in PYTHON(100):"))

if(self.pym>=0) and (self.pym<=100):

break

def compute(self):

self.totmarks=self.cm+self.cppm+self.pym

self.percent=(self.totmarks/300)\*100

#Decide Grade

if(self.cm<40) or (self.cppm<40) or (self.pym<40):

self.grade="FAIL"

else:

if(self.totmarks>=250) and (self.totmarks<=300):

self.grade="DISTINCTION"

elif(self.totmarks>=200) and (self.totmarks<=249):

self.grade="FIRST"

elif(self.totmarks>=150) and (self.totmarks<=199):

self.grade="SECOND"

elif(self.totmarks>=120) and (self.totmarks<=149):

self.grade="THIRD"

def savestuddata(self):

#We must write PDBC Code

con=cx\_Oracle.connect("scott/tiger@localhost/orcl")

cur=con.cursor()

cur.execute("insert into result values(%d,'%s',%d,%d,%d,%d,%f,'%s')" %(self.sno,self.sname,self.cm,self.cppm,self.pym,self.totmarks,self.percent,self.grade) )

con.commit()

print("Student Record Saved Successfully in Result Table:")

#main program

s=Student()

s.getstuddet()

s.compute()

s.savestuddata()

"""write a python program which will accept student details such as student number student name,marks in three subjects.

calculate the total marks

calculate the Percentage and give the grades.(fail/pass)

give the grade = #fail provided student secured less than 40 in any of the three subjects.

give the grade = #distinction provided student totall marks lies within 250 - 300

give the grade = #First class provided totall marks lies within 200- 249

give the grade= second class provided totall marks lies within 150- 199

give the grade= third class provided totall marks lies within 120- 149

save the students result in the database."""

#StudentOOPsDataBaseMySQL.py

import mysql.connector

class Student:

def getstuddet(self):

self.sno=int(input("Enter Student Number:"))

self.sname=input("Enter Student Name:")

#Validation of C Marks

while(True):

self.cm=int(input("Enter Marks in C(100):"))

if(self.cm>=0) and (self.cm<=100):

break

#Validation of C++ Marks

while(True):

self.cppm=int(input("Enter Marks in C++(100):"))

if(self.cppm>=0) and (self.cppm<=100):

break

#validation of Python Marks

while (True):

self.pym=int(input("Enter Marks in PYTHON(100):"))

if(self.pym>=0) and (self.pym<=100):

break

def compute(self):

self.totmarks=self.cm+self.cppm+self.pym

self. Percent=(self.totmarks/300)\*100

#decide Grade

if(self.cm<40) or (self.cppm<40) or (self.pym<40):

self.grade="FAIL"

else:

if(self.totmarks>=250) and (self.totmarks<=300):

self.grade="DISTINCTION"

elif(self.totmarks>=200) and (self.totmarks<=249):

self.grade="FIRST"

elif(self.totmarks>=150) and (self.totmarks<=199):

self.grade="SECOND"

elif(self.totmarks>=120) and (self.totmarks<=149):

self.grade="THIRD"

def savestuddata(self):

#We must write PDBC Code

try:

con=mysql.connector.connect(host="localhost",

user="root",

passwd="root",

database="batch11am")

cur=con.cursor()

cur.execute("insert into result values(%d,'%s',%d,%d,%d,%d,%f,'%s')" %(self.sno,self.sname,self.cm,self.cppm,self.pym,self.totmarks,self.percent,self.grade) )

con.commit()

print("Student Record Saved Successfully in Result Table:")

except mysql.connector.DatabaseError as db:

print("Prob in DB",db)

#main program

s=Student()

s.getstuddet()

s.compute()

s.savestuddata()

#Student.py-------File Name and acts as Module Name

import mysql.connector

class Student:

def getstuddet(self):

self.sno=int(input("Enter Student Number:"))

self.sname=input("Enter Student Name:")

#validation of C Marks

while(True):

self.cm=int(input("Enter Marks in C(100):"))

if(self.cm>=0) and (self.cm<=100):

break

#validation of C++ Marks

while(True):

self.cppm=int(input("Enter Marks in C++(100):"))

if(self.cppm>=0) and (self.cppm<=100):

break

#validation of Python Marks

while(True):

self.pym=int(input("Enter Marks in PYTHON(100):"))

if(self.pym>=0) and (self.pym<=100):

break

def compute(self):

self.totmarks=self.cm+self.cppm+self.pym

self.percent=(self.totmarks/300)\*100

#decide Grade

if(self.cm<40) or (self.cppm<40) or (self.pym<40):

self.grade="FAIL"

else:

if(self.totmarks>=250) and (self.totmarks<=300):

self.grade="DISTINCTION"

elif(self.totmarks>=200) and (self.totmarks<=249):

self.grade="FIRST"

elif(self.totmarks>=150) and (self.totmarks<=199):

self.grade="SECOND"

elif(self.totmarks>=120) and (self.totmarks<=149):

self.grade="THIRD"

def savestuddata(self):

#We must write PDBC Code

try:

con=mysql.connector.connect(host="localhost",

user="root",

passwd="root",

database="batch11am")

cur=con.cursor()

cur.execute("insert into result values(%d,'%s',%d,%d,%d,%d,%f,'%s')" %(self.sno,self.sname,self.cm,self.cppm,self.pym,self.totmarks,self.percent,self.grade) )

con.commit()

print("Student Record Saved Successfully in Result Table:")

except mysql.connector.DatabaseError as db:

print("Prob in DB",db)

Studen

#StudDemo.py

from Student import Student

s=Student()

s.getstuddet()

s.compute()

s.savestuddata()

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

Index of Constructors in Class of Python

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

Index

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

=>Purpose of Constructors

=>Definition of Constructors

=>Rules / Properties of Constructors

=>Types of Constructors

a) Default Constructors

b) Parameterized Constructors

=>Programming Examples

---------------------------------------------------------------------------d

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

Constructors in Python

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

=>The purpose of Constructors in Python is that "To Initlize the object ".

=>Initlizing the object is nothing but placing our own values in the object without leaving an object empty.

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

=>Definition of Constructor:

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

=>A Constructor is a special Method which is automatically or implicitly called by PVM during object creation and whose purpose is to initlize the object without leaving an object empty.

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

Syntax for Defining Constructor:

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

def \_\_init\_\_(self, list of formal params if any):

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

Block of Statements--Initlization

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

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

Rules or Properties of Constructor:

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

1.The Name of the constructor is def \_\_init\_\_(self)

2. The Constructors automatically or implicitly called by PVM during object creation

3. Constructors should not return any value ( It can return only None value)

4. Constructors paricipates in Inheritance Process.

5. Constructors can be Overridden (can re-defined)

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

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

Types Constructors in Python

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

=>In Python Programming, we have two types of Constructors. They are

1. Default or Paremeter-Less Constructor

2. Parameterized Constructor

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

1. Default or Paremeter-Less Constructor

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

=>A Constructor is said to be Default iff it never takes any argument (s ) or Formal Param(s)

=>The purpose of Default or Paremeter-Less Constructor is that "To Initlize multiple objects of same class with same values".

=>Syntax:

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

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

Block of Stmts--Initlization

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

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

=>Examples

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

#DefConstEx1.py

class Test:

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

print("i am from Default Constructor:")

self.a=10

self.b=20

print("Value of a:{}".format(self.a))

print("Value of b:{}".format(self.b))

#main program

t1=Test()

t2=Test()

t3=Test()

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

2.Paremeterized Constructor

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

=>A Constructor is said to be Paremeterized iff it always takes any argument(s ) or Formal Param(s)

=>The purpose of Paremeterized Constructor is that "To Initlize multiple objects of same class with Different values".

=>Syntax:

def \_\_init\_\_(self, list of formal params):

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

Block of Stmts--Initlization

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

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

Examples:

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

#ParamConstEx1.py

class Test:

def \_\_init\_\_(self,a,b):

print("i am from Parameterized Constructor:")

self.a=a

self.b=b

print("Value of a:{}".format(self.a))

print("Value of b:{}".format(self.b))

#main progra m

t1=Test(10,20)

t2=Test(100,200)

t3=Test("RS","PYTHON")

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

Note:

----------

Note: In Class of Python, we can't define both default and Parameterized constructors bcoz PVM can remember only latest constructor (due to its interpretation Process). To full fill the need of both default and parameterized constructors, we define single constructor with default parameter mechanism.

Examples:

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

#DefultParamConstEx1.py

class Test:

def \_\_init\_\_(self,a=1,b=2):

print("i am from Default /Parameterized Constructor:")

self.a=a

self.b=b

print("Value of a:{}".format(self.a))

print("Value of b:{}".format(self.b))

#main progra m

t1=Test() # Object Creation calls Default Constructor

t2=Test(100,200) # Object Creation calls Parameterized Constructor

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

#Program for demostrating the need of Constructor

#Non-ConstEx1.py

class Student:

def getstudvalues(self):

self.sno=10

self.sname="Rossum"

#main program

s=Student() # Object Creation

print("Initial Content of s=",s.\_\_dict\_\_) # { }

#To place the data inside of object s, we must call getstudvalues() explicitly

s.getstudvalues()

print("Content of s after calling method=",s.\_\_dict\_\_) # { }

#Program for demostrating the need of Constructor

#ConstEx1.py

class Student:

def \_\_init\_\_(self): # default or Parameterless constructor

self.sno=10

self.name="Rossum"

#main program

s=Student() # Object Creation---PVM Calls implicitly constructor

print("Initial Content of s=",s.\_\_dict\_\_) # { }

s1=Student() # Object Creation---PVM Calls implicitly constructor

print("Initial Content of s1=",s1.\_\_dict\_\_) # { }

s2=Student() # Object Creation---PVM Calls implicitly constructor

print("Initial Content of s2=",s2.\_\_dict\_\_) # { }

#Program for demostrating the need of Constructor

#ConstEx2.py

class Student:

def \_\_init\_\_(self,sno,sname): # Parametrized Constructor

self.sno=sno

self.sname=sname

#main program

s1=Student(10,"Rossum") # Object Creation---PVM Calls implicitly constructor

print("Initial Content of s1=",s1.\_\_dict\_\_) # { }

s2=Student(20,"Travis") # Object Creation---PVM Calls implicitly constructor

print("Initial Content of s2=",s2.\_\_dict\_\_) # { }

s3=Student(30,"Kinney") # Object Creation---PVM Calls implicitly constructor

print("Initial Content of s3=",s3.\_\_dict\_\_) # { }

#DefaultConstEx1.py

class Test:

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

print("\nI am from default constructor")

self.a=10

self.b=20

print("Val of a=",self.a)

print("Val of b=",self.b)

#main program

t1=Test() # Object Creating-----calls default constructor

t2=Test() # Object Creating------calls default constructor

t3=Test() # Object Creating------calls default constructor

#ParamConstEx1.py

class Test:

def \_\_init\_\_(self,a,b):

print ("\nI am from Parameterized constructor")

self.a=a

self.b=b

print("Val of a=",self.a)

print("Val of b=",self.b)

#main program

t1=Test(10,20) # Object Creating-----calls Parameterized constructor

t2=Test(100,200) # Object Creating------callsParameterized constructor

t3=Test(1000,2000) # Object Creating------calls Parameterized constructor

#DefaultParamConstEx1.py

class Test:

def \_\_init\_\_(self,a=1,b=2):

print ("\nI am from default / Parameterized constructor")

self.a=a

self.b=b

print ("Val of a=",self.a)

print ("Val of b=",self.b)

#main program

t1=Test() # Object Creating-----calls Default constructor

t2=Test(10,20) # Object Creating------calls Parameterized constructor

t3=Test(100)# Object Creating------calls Parameterized constructor

t3=Test(b=100)# Object Creating------calls Parameterized constructor

t4=Test(b=100,a=200)# Object Creating------calls Parameterized constructor

t5=Test(b=100,a="KVR")# Object Creating------calls Parameterized constructor

t6=Test("Python",b="Java")# Object Creating------calls Parameterized constructor

#write a python program which will read student values such as sno,sname and marks. save the details of the student in a file by using pickling and read the student record values from the file by using unpickling.impliment this example by using classes and objects.

#Student.py--File Name and Module Name

class Student:

def \_\_init\_\_(self,sno,sname,marks):

self.sno=sno

self.sname=sname

self.marks=marks

def dispstuddata(self):

print("\t{}\t{}\t{}".format(self.sno,self.sname,self.marks))

#StudOopsPickEx.py

import sys,pickle

from Student import Student

class StudPick:

def savestuddata(self):

with open("oopsstud.data","ab") as fp:

while(True):

print("-"\*50)

sno=int(input("Enter Student Number:"))

sname=input("Enter Student Name:")

marks=float(input("Enter Student Marks:"))

s=Student(sno,sname,marks) # Calling PC of Student Class

pickle.dump(s,fp)

print("Student Data Saved in a File:")

print("-"\*50)

ch=input("Do u want to Insert another record(yes/no):")

if(ch.lower()=="no"):

print("Thx for using this program")

print("-"\*50)

sys.exit()

#main Program

sp=StudPick()

sp.savestuddata()

#StudOopsUnPickEx.py

import pickle

class StudUnPick:

def readrecords(self):

with open("oopsstud.data","rb") as fp:

print("-"\*50)

print("\tSNO\tNAME\tMARKS")

print("-"\*50)

while(True):

try:

obj=pickle.load(fp)

obj.dispstuddata()

except EOFError:

print("-"\*50)

break

#main program

so=StudUnPick()

so.readrecords()

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

Destructors in Python

and

Garbage Collector

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

=>We know that Garbage Collector is one of the in-built program in python, which is running behind of every python program and whose is role is to collect un-used memory space and it improves the performnace of python based applications.

=>Every Garbage Collector Program is internally calling its Own Destructor Functions.

=>The destructor function name in python is def \_\_del\_\_(self).

=>By default, The destructor always called by Garbage Collector when the program execution completed for de-allocating the memory space of objects which are used in that program. Where as constructor called By PVM implicitly when object is created for initlizing the object.

=>When the program execution is completed, GC calls its own destructor to de-allocate the memory space of objects present in program and it is called automatic Garbage Collection.

=>Hence, We have THREE programming conditions for calling GC and to make the garbage collector to call destructor Function.

a) By default (or) automatically GC calls destructor, when the program execution

completed.

b) Make the object reference as None for calling Forcefull Garbage Collection

Syntax : objname=None

c) delete the object by using del operator for calling Forcefull Garbage Collection

Syntax:- del objname

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

=>Syntax:

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

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

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

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

=>No Need to write destructor in class of Python bcoz GC contains its own Destructor

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

Garbage Collector

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

=>Garbage Collector contains a pre-defined module called "gc"

=>Here gc contains the following Functions.

1) isenabled()

2) enable()

3) disable()

=>GC is not under control of Programmer but it always maintained and managed by OS and PVM.

NOTE: Python Programmers need not to write destructor method / function and need not to deal with Garbage Collection Process by using gc module bcoz PVM and OS takes care about Automatic Garbage Collection Process.

==============================x============================================

#Non-DestEx.py

class Student:

def \_\_init\_\_(self,sno,sname):

self.sno=sno

self.sname=sname

print("\t{}\t{}".format(self.sno,self.sname))

#main program

print("\nProgram Execution Started")

s1=Student(10,"RS")# Object Creation

s2=Student(20,"TR")# Object Creation

print("Program Execution Ended")

#DestEx1.py

import sys

class Student:

def \_\_init\_\_(self,sno,sname):

print("I am from PC")

self.sno=sno

self.sname=sname

print("\t{}\t{}".format(self.sno,self.sname))

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

global totmem

print("GC calls \_\_del\_\_")

print("At Present Memory Space:{}".format(totmem))

print("Now Memory space:",sys.getsizeof(self))

totmem=totmem-sys.getsizeof(self)

print("Remaing Memory Space:{}".format(totmem))

4

#main program

print("\nProgram Execution Started")

s1=Student(10,"RS")# Object Creation

s2=Student(20,"TR")# Object Creation

totmem=sys.getsizeof(s1)+sys.getsizeof(s2)

print("Now Memory space in main program:{}",totmem)

print("Program Execution Ended")

#DestEx2.py

import time

class Student:

def \_\_init\_\_(self,sno,sname):

print("I am from PC")

self.sno=sno

self.sname=sname

print("\t{}\t{}".format(self.sno,self.sname))

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

print("GC calls \_\_del\_\_")

#main program

print("\nProgram Execution Started")

s1=Student(10,"RS")# Object Creation

print("Now we are No Longer interested in maintaing S1 object memory space:")

time.sleep(5)

s1=None # Calling GC Forcefully and it inturns calls Destructor

time.sleep(5)

s2=Student(20,"TR")# Object Creation

print("Program Execution Ended")

time.sleep(5)

#DestEx3.py

import time

class Student:

def \_\_init\_\_(self,sno,sname):

print("I am from PC")

self.sno=sno

self.sname=sname

print("\t{}\t{}".format(self.sno,self.sname))

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

print("GC calls \_\_del\_\_")

#main program

print("\nProgram Execution Started")

s1=Student(10,"RS")# Object Creation

print("Now we are No Longer interested in maintaing S1 object memory space:")

time.sleep(5)

s1=None # Calling GC Forcefully and it inturns calls Destructor

time.sleep(5)

s2=Student(20,"TR")# Object Creation

print("Now we are No Longer interested in maintaing S2 object memory space:")

time.sleep(5)

s2=None # Calling GC Forcefully and it inturns calls Destructor

print("Program Execution Ended")

#DestEx4.py

import time

class Student:

def \_\_init\_\_(self,sno,sname):

print ("I am from PC")

self.sno=sno

self.sname=sname

print ("\t {} \t{}”. format(self.sno,self.sname))

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

print ("GC calls \_\_del\_\_")

#main program

print("\nProgram Execution Started")

s1=Student(10,"RS")# Object Creation

s3=Student(30,"KVR") # Object Creation

print("Now we are No Longer interested in maintaing S1 object memory space:")

time.sleep(5)

del s1 # Calling GC Forcefully and its inturns calls Destructor

time.sleep(5)

s2=Student(20,"TR")# Object Creation

print("Now we are No Longer interested in maintaing S2 object memory space:")

time.sleep(5)

del s2 # Calling GC Forcefully and its inturns calls Destructor

print("Program Execution Ended")

time.sleep(5)

#DestEx5.py

import time

class Student:

def \_\_init\_\_(self,sno,sname):

print ("I am from PC")

self.sno=sno

self.sname=sname

print("\t{} \t{}".format(self.sno,self.sname))

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

print ("GC calls \_\_del\_\_")

#main program

print("\nProgram Execution Started")

s1=Student(10,"RS”) # Object Creation

s2=s1 #Deep Copy

s3=s1 #Deep Copy

print(id(s1),id(s2),id(s3))

time.sleep(5)

print("\nProgram Execution Ended")

time.sleep(5)

#Here GC calls \_\_del\_\_(self) only once even we have 3 object and they have same address..

#DestEx6.py

import time

class Student:

def \_\_init\_\_(self,sno,sname):

print("I am from PC")

self.sno=sno

self.sname=sname

#print("\t{}\t{}".format(self.sno,self.sname))

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

print("GC calls \_\_del\_\_")

#main program

print("\nProgram Execution Started")

s1=Student(10,"RS")# Object Creation

s2=s1 #Deep Copy

s3=s1 #Deep Copy

print("Now we are No Longer interested in maintaing S1 object memory space:")

time.sleep(5)

s1=None # GC will not call \_\_del\_\_(self) bcoz still s2 and s3 to same memory space

print("Now we are No Longer interested in maintaing S2 object memory space:")

time.sleep(5)

del s2 # GC will not call \_\_del\_\_(self) bcoz still s3 to same memory space

print("\nProgram Execution Ended")

time.sleep(5)

#Here GC calls \_\_del\_\_(self)

#DestEx7.py

import time

class Student:

def \_\_init\_\_(self,sno,sname):

print("I am from PC")

self.sno=sno

self.sname=sname

#print("\t{}\t{}".format(self.sno,self.sname))

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

print("GC calls \_\_del\_\_")

#main program

print("\nProgram Execution Started")

s1=Student(10,"RS")# Object Creation

s2=s1 #Deep Copy

s3=s1 #Deep Copy

print("Now we are No Longer interested in maintaing S1 object memory space:")

time.sleep(5)

s1=None # GC will not call \_\_del\_\_(self) bcoz still s2 and s3 to same memory space

print("Now we are No Longer interested in maintaing S2 object memory space:")

time.sleep(5)

del s2 # GC will not call \_\_del\_\_(self) bcoz still s3 to same memory space

print("Now we are No Longer interested in maintaing S3 object memory space:")

time.sleep(5)

del s3 # GC will not call \_\_del\_\_(self) bcoz still s3 to same memory space

print("\nProgram Execution Ended")

time.sleep(5)

#Here GC will not call \_\_del\_\_(self)

#gcex1.py

import gc

print("Line-3-->Is GC Running:",gc.isenabled())

print("\nThis Pthon class")

print("Destructor Topic is Going on")

gc.disable()

print("Line-5-->Is GC Running:",gc.isenabled())

print("Pythin is an OOp alng")

gc.enable()

print("Line-10-->Is GC Running:",gc.isenabled())

print("Python is also fun Prog")

#gcex2.py

import time,gc

class Student:

def \_\_init\_\_(self,sno,sname):

print("I am from PC")

self.sno=sno

self.sname=sname

print("\t{}\t{}".format(self.sno,self.sname))

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

print("GC calls \_\_del\_\_")

#main program

print("\nProgram Execution Started")

print("Line-15-->Is GC is RUNNING:",gc.isenabled())

s1=Student(10,"RS")# Object Creation

print("\nNow we are No Longer interested in maintaing S1 object memory space:")

time.sleep(5)

s1=None # Calling GC Forcefully and it inturns calls Destructor

time.sleep(5)

gc.disable()

print("\nLine-22-->Is GC is RUNNING:",gc.isenabled())

s2=Student(20,"TR")# Object Creation

print("\nNow we are No Longer interested in maintaing S2 object memory space:")

time.sleep(5)

s2=None

print("Program Execution Ended")

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

objects in Python

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

=>When we define a class, memory space is not created for Data Members and Methods but whose memory is created when we create an object w.r.t class name.

=>To Store the data and to do any Data Processing, it is mandatory to create an object.

=>To create an object, there must exists a class Definition otherwise we get NameError.

Definition of object:

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

=>Instance of a class is called object (Instance is nothing but allocating sufficient memory space for the Data Members and Methods of a class).

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

Syntax for creating an object

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

varname=classname()

(or)

varname=classname(Val1,Val2...val-n)

Examples: create an object of Student

so=Student()

Example:- create an object Employee

eo=Employee(10,"Rossum")

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

Differences Betwwen Classes and Objects

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

Class:

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

1) A class is a collection of Data Members and Methods

2) When we define a class, memory space is not created for Data Members and Methods and it can be treated as specification / model for real time application.

3) Definition of a perticular exists only once

4) When we develop any Program with OOPs principles, Class Definition Loaded First in main memory only once.

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

Objects:

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

1) Instance of a class is called Object

2) When we create an object, we get the memory space for Data members and Methods of Class.

3)w.r.t One class Definition, we can create multiple objects.

4)we can create an object after loading the class definition otherwise we get NameError

===============================X=================================

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

Inheritance

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

=>Inheritance is one of distinct features of OOPs

=>The purpose of Inheritance is that " To build Re-usable Applications in Python Object Oriented Programming".

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

=>Definition of Inheritance:

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

=>The Process obtaining Data members, get Methods and Constructors (Features ) of one class into another class is called Inheritance.

=>The class which is giving Data members , Methods and Constructors (Features ) is called Super or Base or Parent Class.

=>The Class which is taking Data members , Methods and Constructors (Features ) is called Sub or Derived or Child Class.

=>The Inheritance concept always follows Logical (Virtual ) Memory Management. This Memory Management says that " Neither we write Source Code nor Takes Physical Memory Space ".

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

Advatnages of Inheritance:

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

=>When we develop any inheritance-based application, we get the following advantages.

1. Application Development Time is Less

2. Application Memory Space is Less

3. Application Execution time is Fast / Less

4. Application Performance is enhanced (Improved)

5. Redundancy (Duplication) of the code is minimized.

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

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

Types of Inheritances

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

=>A Type of Inheritance is a model / Paradigm , which makes us to understand how the

features are In herited from Base Class into Derived Class.

=>In Python Programming, we have 5 types of Inheritances. They are

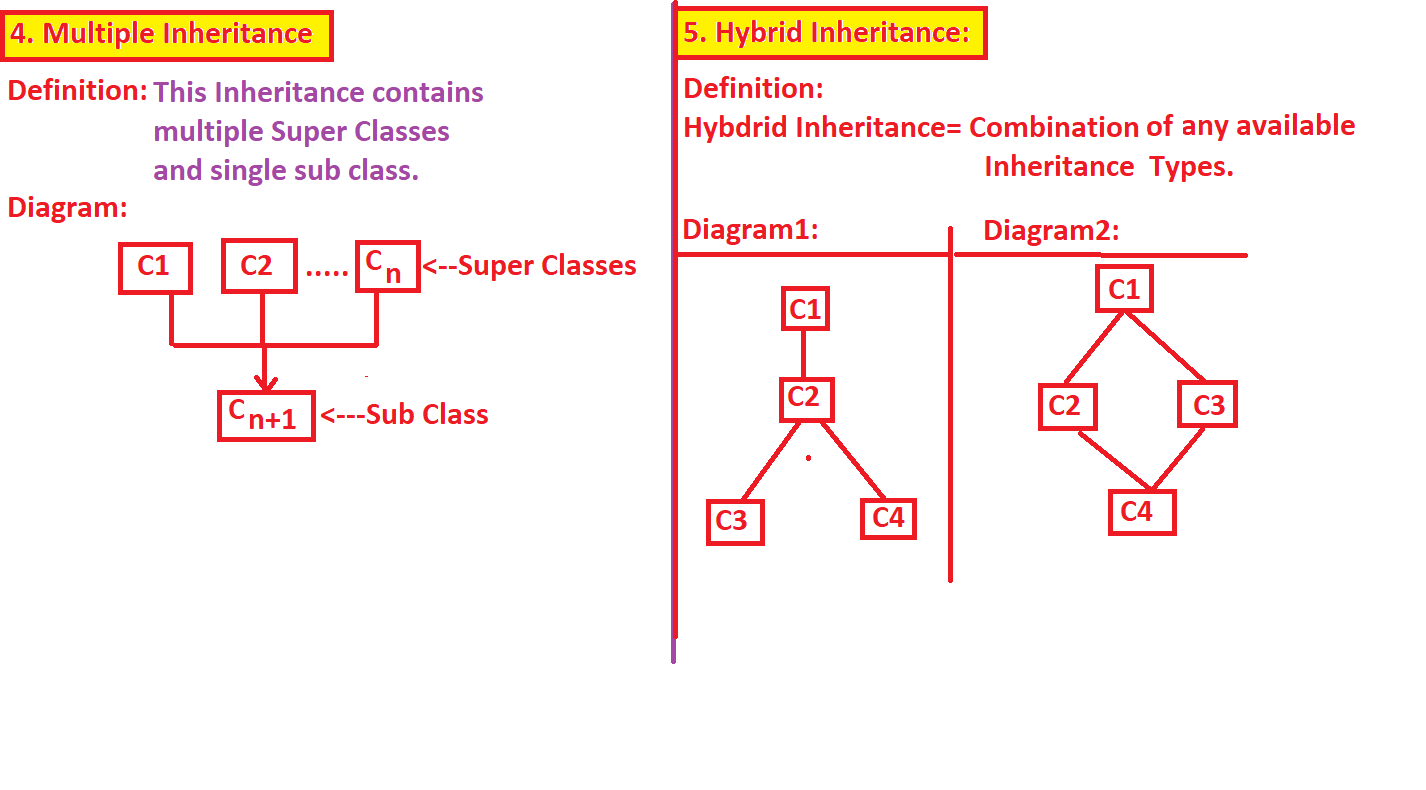
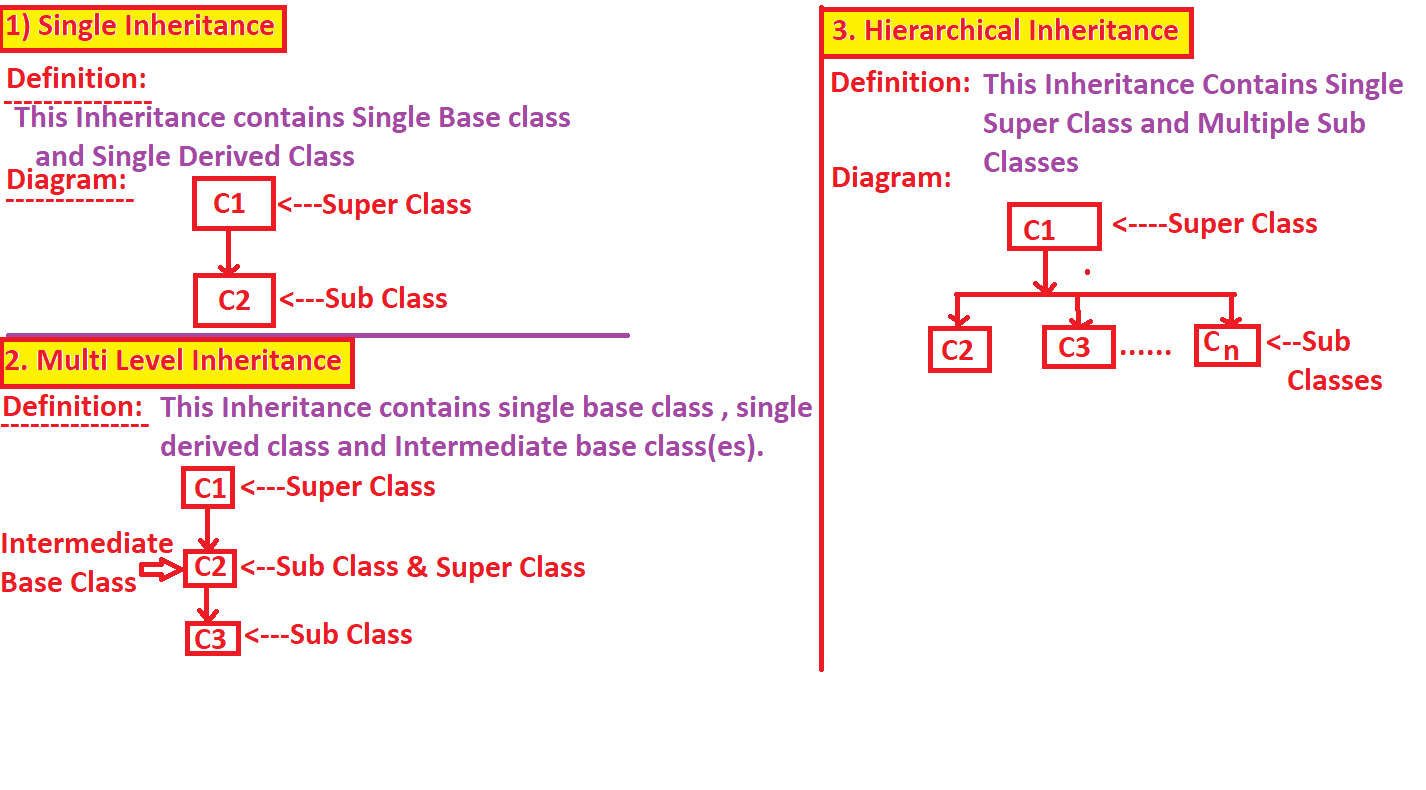
1. Single Inheritance

2. Multi Level Inheritance

3. Hierarchical Inheritance

4. Multiple Inheritance

5. Hybrid Inheritance

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

Data Encapsulation and Data Abstraction

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

Data Encapsulation:

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

=>The Process of Hiding the confidential Information / Data / Methods from external Programmers / end users is called Data Encapsulation.

=>The Purpose of Encapsulation concept is that "To Hide Confidental Information / Features of Class (Data Members and Methods ) ".

=>Data Encapsulation can be applied in two levels. They are

a) At Data Members Level

b) At Methods Level

=>To implement Data Encapsulation in python programming, The Data Members , Methods must be preceded with double under score ( \_ \_ )

Syntax1:- (Data member Lavel )

class <ClassName>:

def methodname(self):

self.\_\_Data MemberName1=Value1

self.\_\_Data MemberName2=Value2

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

self.\_\_Data MemberName-n=Value-n

(OR)

Syntax1:- ( Data member Lavel )

class <ClassName>:

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

self.\_\_Data MemberName1=Value1

self.\_\_Data MemberName2=Value2

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

self.\_\_Data MemberName-n=Value-n

Syntax2:- (Method Level)

class <ClassName>:

def \_\_methodname(self):

self.Data MemberName1=Value1

self.Data MemberName2=Value2

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

self.Data MemberName-n=Value-n

Examples: Refer Acc1.py, Acc2.py, Acc3.py and Acc4.py Programs

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

Data Abstraction:

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

=>The Process of retrieving / extracting Essential Details without considering Hidden Deta

ils is called Data Abstraction.

Examples: Others1.py Others2.py

Others3.py Others4.py Programs

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

Note:- We can't apply Data Encapsulation on Constructors in Python but whose Initlized Data Memebrs can be encapsulated.

#Acc1.py-----File Name and Module Name

class Account:

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

self. \_\_acno=10

self.cname="Rossum"

self.\_\_bal=34

self.\_\_pin=1234

self.bname="SBI"

#Acc2.py-----File Name and Module Name

class Account:

def getaccdet(self):

self.\_\_acno=10

self.cname="Rossum"

self.\_\_bal=34

self.\_\_pin=1234

self.bname="SBI"

#Acc3.py-----File Name and Module Name

class Account:

def \_\_getaccdet(self):

self.acno=10

self.cname="Rossum"

self.bal=34

self.pin=1234

self.bname="SBI"

#Acc4.py-----File Name and Module Name

class Account:

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

self.acno=10

self.cname="Rossum"

self.bal=34

self.pin=1234

self.bname="SBI"

#Others1.py

from Acc1 import Account

ac=Account() # Object Creation

#print("\nAccount Number:{}".format(ac.acno))

print("Account Name:{}".format(ac.cname))

#print("Account Balance:{}".format(ac.bal))

#print("Account Pin:{}".format(ac.pin))

print("Account Branch Name:{}".format(ac.bname))

#Others2.py

from Acc2 import Account

ac=Account() # Object Creation

ac.getaccdet()

#print("\nAccount Number:{}".format(ac.acno))

print("Account Name:{}".format(ac.cname))

#print("Account Balance:{}".format(ac.bal))

#print("Account Pin:{}".format(ac.pin))

print("Account Branch Name:{}".format(ac.bname))

#Others3.py

from Acc3 import Account

ac=Account() # Object Creation

print(ac.\_\_dict\_\_)

ac.getaccdet()

#Others4.py

from Acc4 import Account

ac=Account() # Object Creation

ac.\_\_\_\_\_\_init\_\_()

print("\nAccount Number:{}".format(ac.acno))

print("Account Name:{}".format(ac.cname))

print("Account Balance:{}".format(ac.bal))

print("Account Pin:{}".format(ac.pin))

print("Account Branch Name:{}".format(ac.bname))

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

Inheritaing the Features of Base Class into Derived Class

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

Syntax:

---------

class <clsname-1>:

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

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

class <clsname-2>:

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

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

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

class <clsname-n>:

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

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

class <clsname-n+1>(<clsname-1>,<clsname-2>,.....<clsname-n>):

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

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

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

Explanation:

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

=>Here <clsname-1>,<clsname-2>.....<clsname-n> are called Base Classes / Super Classes

=>here <clsname-n+1> is called Derived Class.

=>Here The features of <clsname-1>,<clsname-2>.....<clsname-n> are Inherited into <clsname-n+1> and these Features are available logically In <clsname-n+1> and we can access them w.r.t Object Name (OR) Self (Instance ) /Class Name (Class level and Static ).

=>When we develop any Inheritance Application, we are always recommeding to create an object of Bottom Most Derived Class bcoz It inherits the features of Intermediate Base Class(es) and Base class.

=>For Every Class in Python, there exist a pre-defined implicit super class called "object" bcoz It provides Garbage Collection Facility to its sub classes.

==============================x=======================================

#Program for demonstarting Inheritance

#InhProg1.py

class C1:

def setA(self):

self.a=10

class C2(C1): # Single Inheritance C-- is called Base Class and C2 is called Derived Class

def setB(self):

self.b=20

def disp(self):

print("Val of a(C1-BC):{}".format(self.a))

print("Val of b(C2-DC):{}".format(self.b))

#main program

o2=C2()

print("content of o2:",o2.\_\_dict\_\_)

o2.setB()

print("content of o2:",o2.\_\_dict\_\_)

o2.setA()

print("content of o2:",o2.\_\_dict\_\_)

o2.disp()

#Program for demonstarting Inheritance

#InhProg2.py

class C1:

def setA(self):

self.a=10

class C2(C1): # Single Inheritance C-- is called Base Class and C2 is called Derived Class

def setB(self):

self.b=20

self.setA() # calling Base Class Method from derived class method

self.disp() # calling Current Class Method from other method of current class

def disp(self):

print("Val of a(C1-BC):{}".format(self.a))

print("Val of b(C2-DC):{}".format(self.b))

#main program

o2=C2()

print("content of o2:",o2.\_\_dict\_\_)

o2.setB()

#Program for demonstarting Inheritance

#InhProg3.py

class GrandParent:

def getgpprop(self):

self.gpp=3.4

class Parent(GrandParent):

def getpprop(self):

self.pp=13.4

class Child (Parent):

def childprop(self):

self.cprop=4.5

def totalproperty(self):

prop=self.gpp+self.pp+self.cprop

print("-"\*50)

print("\tProperties of Child:")

print("-"\*50)

print("\tGrand Parent Properties:{}".format(self.gpp))

print("\tParent Properties:{}".format(self.pp))

print("\tChild Properties:{}".format(self.cprop))

print("-"\*50)

print("\tTotal Propery:{}".format(prop))

print("-"\*50)

#main program

co=Child()

co.childprop()

co.getpprop()

co.getgpprop()

co.totalproperty()

#Program for demonstarting Inheritance

#InhProg4.py

class GrandParent:

def getgpprop(self):

self.gpp=3.4

class Parent(GrandParent):

def getpprop(self):

self.pp=13.4

class Child(Parent):

def childprop(self):

self.cprop=4.5

def totalproperty(self):

self.getgpprop()

self.getpprop()

self.childprop()

prop=self.gpp+self.pp+self.cprop

print("-"\*50)

print("\tProperties of Child:")

print("-"\*50)

print("\tGrand Parent Properties:{}".format(self.gpp))

print("\tParent Properties:{}".format(self.pp))

print("\tChild Properties:{}".format(self.cprop))

print("-"\*50)

print("\tTotal Propery:{}".format(prop))

print("-"\*50)

#main program

co=Child ()

co.totalproperty()

#Program for demonstarting Inheritance

#InhProg5.py

class GrandParent:

def getgpprop(self):

self.gpp=float(input("Enter Grand Parent Properties:"))

class Parent(GrandParent):

def getpprop(self):

self.pp=float(input("Enter Parent Properties:"))

class Child(Parent):

def childprop(self):

self.cprop=float(input("Enter Child Properties:"))

def totalproperty(self):

self.getgpprop()

self.getpprop()

self.childprop()

prop=self.gpp+self.pp+self.cprop

print("-"\*50)

print("\tProperties of Child:")

print("-"\*50)

print("\tGrand Parent Properties:{}".format(self.gpp))

print("\tParent Properties:{}".format(self.pp))

print("\tChild Properties:{}".format(self.cprop))

print("-"\*50)

print("\tTotal Propery:{}".format(prop))

print("-"\*50)

#main program

co=Child()

co.totalproperty()

#Program for demonstarting Inheritance

#InhProg6.py

class Father:

def getfatherprop(self):

self.fp=float(input("Enter Father Properties:"))

return self.fp

class Mother:

def getmotherprop(self):

self.mp=float(input("Enter Mother Properties:"))

return self.mp

class Child(Mother,Father):

def totalprop(self):

fp=self.getfatherprop()

mp=self.getmotherprop()

totprop=fp+mp

print("\tProperties of Child:")

print("-"\*50)

print("\tFather Properties:{}".format(fp))

print("\tMother Properties:{}".format(mp))

print("\tChild Total Properties:{}".format(totprop))

print("-"\*50)

#main program

c=Child()

c.totalprop()

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

Polymorphism in Python

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

=>Polymorphism is one of the distinct features of OOPs

=>The purpose of Polymorphism is that "Efficient Utilization Memory Space (OR) Less Memory space is achieved".

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

=>Def. of Polymorphism:

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

=>The Process of Representing "One Form in multiple Forms " is called Polymorphism.

=>The Polymorphism Principle is implemented (Bring into action) by Using "Method Overriding" feature of all OO Programming Languages.

=>In The definition of polymorphism, "One Form" represents "Original Method" and multiple forms represents Overridden Methods.

=>A "Form" is nothing but existence of a Method. if the method is existing in base class then it is called "Original Method (one form)" and if the method existing in derived class(es) then it is called "Overridden Method (multiple Forms)".

---------------------------------------------------X-----------------------

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

Method Overriding in Python

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

=>Method Overriding=Method Heading is same + Method Body is Different

(OR)

=>The process of re-defining the original method of base class into various derived classes for performing different operations is called Method Overriding.

=>To use Method Overriding in python program we must apply Inheritance Principle.

=>Method Overriding used for implementing Polymorphism Principle.

(PLOYMORPHISM<----METHOD OVERRIDING<-----INHERITANCE<----CLASS AND OBJECTS )

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

Examples:

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

#methodoverex1.py

class Circle:

def draw(self): # original Method

print("Drawing Circle")

class Rect(Circle):

def draw(self): # overridden Method

print("Drawing Rect:")

super().draw()

class Square (Rect):

def draw(self): # overridden Method

print("Drawing Square:")

super().draw()

#main program

so=Square ()

so.draw()

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

#teacher.py

class Teacher:

def readsub(self):

print("Teacher advises to read 2 hours")

class LazyStudent(Teacher):

def readsub(self):

print("LazyStudent never read at all")

class PerfectStudent(Teacher):

def readsub(self):

print(" Perfect Student 2hrs reading and practicing")

ls=LazyStudent()

ls.readsub()

ps=PerfectStudent()

ps.readsub()

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

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

Number of approaches to call original methods / constructors from

Overridden methods / Constructors

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

=>We have two approches to call original method / constructors of base class from overridden method / constructors of derived class. They are

1) By using super()

2) By using Class Name

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

1) By using super():

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

=>super () is one of the pre-defined function, which is used for calling super class original method / constructor from overridden method / constructors of derived class.

Syntax1:- super().methodname(list of values if any)

Syntax2:- super().\_\_init\_\_(list of values if any)

=>with super() we are able to call only immediate base class method but unable to call Specified method of base Class . To do this we must use class name approach.

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

2) By using Class Name:

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

=>By using ClassName approach, we can call any base class method / constructor name from the context of derived class method / constructor names.

Syntax1:- ClassName.methodname(self, list of values if any)

Syntax2:- ClassName.\_\_init\_\_(self, list of values if any)

-----------------------------------------------X---------------------------

#PolyEx1.py

class Circle:

def draw(self): # Original Method

print ("Drawing Circle")

class Rect (Circle):

def draw(self): # Overridden Method

print ("ssDrawing Rect")

super().draw()

class Square(Rect):

def draw(self): # Overridden method

print("Dawing Square")

super().draw()

#main program

s=Square()

s.draw()

#PolyEx2.py

class Circle:

def draw(self): # Original Method

print("Drawing Circle")

class Rect(Circle):

def draw(self): # Overridden Method

print("Drawing Rect")

class Square(Rect):

def draw(self): # Overridden method

print("Dawing Square")

Circle.draw(self)

Rect.draw(self)

#main program

s=Square()

s.draw()

#PolyEx3.py

class Circle:

def \_\_init\_\_(self): # Original default Constructor

print("Drawing Circle--DC")

class Rect(Circle):

def \_\_init\_\_(self): # Overriddent default constructor

print("Drawing Rect--DC")

super().\_\_init\_\_()

#main program

ro=Rect() # Object Creation and calls Default Constructor

#PolyEx4.py

class Circle:

def \_\_init\_\_(self): # Original default Constructor

print("Drawing Circle--DC")

class Rect(Circle):

def \_\_init\_\_(self): # Overriddent default constructor

print("Drawing Rect--DC")

Circle.\_\_init\_\_(self)

#main program

ro=Rect() # Object Creation and calls Default Constructor

#PolyEx5.py

class Circle:

def area(self): # Orioginal Method (One Form)

self.r=float(input("Enter Radious:"))

self.ac=3.14\*self.r\*\*2

print("Area of Circle:{}".format(self.ac))

class Square(Circle):

def area(self): # Overridden Methods (Multiple Forms)

self.s=float(input("Enter Side:"))

self.sa=self.s\*\*2

print("Area of Square:{}".format(self.sa))

print("-"\*50)

super().area()

class Rect(Square):

def area(self): # Overridden Methods (Multiple Forms)

self.l=float(input("Enter Length:"))

self.b=float(input("Enter Breadth:"))

self.ar=self.l\*self.b

print("Area of Rect:{}".format(self.ar))

print("-"\*50)

super().area()

#main program

r=Rect()

r.area()

#PolyEx6.py

class Circle:

def area(self): # Orioginal Method (One Form)

self.r=float(input("Enter Radious:"))

self.ac=3.14\*self.r\*\*2

print("Area of Circle:{}".format(self.ac))

class Square(Circle):

def area(self): # Overridden Methods (Multiple Forms)

self.s=float(input("Enter Side:"))

self.sa=self.s\*\*2

print("Area of Square:{}".format(self.sa))

class Rect(Square):

def area(self): # Overridden Methods (Multiple Forms)

self.l=float(input("Enter Length:"))

self.b=float(input("Enter Breadth:"))

self.ar=self.l\*self.b

print("Area of Rect:{}".format(self.ar))

print("-"\*50)

Circle.area(self)

print("-"\*50)

super().area()

#main program

r=Rect()

r.area()

#PolyEx7.py

class Circle:

def area(self): # Orioginal Method (One Form)

self.r=float(input("Enter Radious:"))

self.ac=3.14\*self.r\*\*2

print("Area of Circle:{}".format(self.ac))

class Square:

def area(self): # Orioginal Method (One Form)

self.s=float(input("Enter Side:"))

self.sa=self.s\*\*2

print("Area of Square:{}".format(self.sa))

class Rect(Square,Circle):

def area(self): # Overriddent Method

self.l=float(input("Enter Length:"))

self.b=float(input("Enter Breadth:"))

self.ar=self.l\*self.b

print("Area of Rect:{}".format(self.ar))

print("------------------------------------------------------")

Square.area(self)# OR super().area()

Circle.area(self)

#main program

r=Rect()

r.area()

#PolyEx8.py

class Circle:

def \_\_init\_\_(self): # Original Constructor (One Form)

self.r=float(input("Enter Radious:"))

self.ac=3.14\*self.r\*\*2

print("Area of Circle:{}".format(self.ac))

class Square:

def \_\_init\_\_(self): # Original Constructor (One Form)

self.s=float(input("Enter Side:"))

self.sa=self.s\*\*2

print("Area of Square:{}".format(self.sa))

class Rect(Square,Circle):

def \_\_init\_\_(self): # Overriddent Constructor

self.l=float(input("Enter Length:"))

self.b=float(input("Enter Breadth:"))

self.ar=self.l\*self.b

print("Area of Rect:{}".format(self.ar))

print("------------------------------------------------------")

Square.\_\_init\_\_(self)

Circle.\_\_init\_\_(self)

#main program

r=Rect()

#PolyEx8.py

class Circle:

def \_\_init\_\_(self,r): # Original Constructor (One Form)

self.ac=3.14\*r\*\*2

print("Area of Circle:{}".format(self.ac))

class Square:

def \_\_init\_\_(self,s): # Original Constructor (One Form)

self.sa=s\*\*2

print("Area of Square:{}".format(self.sa))

class Rect(Square,Circle):

def \_\_init\_\_(self,l,b): # Overriddent Constructor

self.ar=l\*b

print("Area of Rect:{}".format(self.ar))

print("------------------------------------------------------")

super().\_\_init\_\_(float(input("Enter Side:")))

print("------------------------------------------------------")

Circle.\_\_init\_\_(self,float(input("Enter Radious:")))

#main program

l=float(input("Enter Length:"))

b=float(input("Enter Breadth:"))

r=Rect(l,b)

#PolyEx10.py

class Circle:

def area(self,r): # Original Constructor (One Form)

self.ac=3.14\*r\*\*2

print("Area of Circle:{}".format(self.ac))

class Square:

def area(self,s): # Original Constructor (One Form)

self.sa=s\*\*2

print("Area of Square:{}".format(self.sa))

class Rect(Square,Circle):

def area(self,l,b): # Overriddent Constructor

self.ar=l\*b

print("Area of Rect:{}".format(self.ar))

print("------------------------------------------------------")

super().area(float(input("Enter Side:")))

print("------------------------------------------------------")

Circle.area(self,float(input("Enter Radious:")))

#main program

l=float(input("Enter Length:"))

b=float(input("Enter Breadth:"))

r=Rect()

r.area(l,b)

#PolyEx11.py

class Circle:

def area(self,r): # Original Constructor (One Form)

self.ac=3.14\*r\*\*2

print("Area of Circle:{}".format(self.ac))

class Square:

def \_\_init\_\_(self,s): # Original Constructor (One Form)

self.sa=s\*\*2

print("Area of Square:{}".format(self.sa))

class Rect(Square,Circle):

def \_\_init\_\_(self,l,b): # Overriddent Constructor

self.ar=l\*b

print("Area of Rect:{}".format(self.ar))

print("------------------------------------------------------")

super().\_\_init\_\_(float(input("Enter Side:")))

print("------------------------------------------------------")

Circle.area(self,float(input("Enter Radious:")))

#main program

l=float(input("Enter Length:"))

b=float(input("Enter Breadth:"))

r=Rect(l,b)

#UnivCollStud.py

class Univ:

def getunivdet(self):

self.uname=input("Enter University Name:")

self.uloc=input("Enter University Location:")

def dispunivdet(self):

print("-"\*40)

print("University details:")

print("-"\*40)

print("University Name:",self.uname)

print("University Location:",self.uloc)

print("-"\*40)

class College(Univ):

def getcolldet(self):

self.cname=input("Enter College Name:")

self.cloc=input("Enter College Location:")

def dispcolldet(self):

print("-"\*40)

print("College details:")

print("-"\*40)

print("College Name:",self.cname)

print("College Location:",self.cloc)

print("-"\*40)

class Student(College):

def getstuddet(self):

self.sno=int(input("Enter Student Number:"))

self.sname=input("Enter Student Name:")

self.crs=input("Enter Student Course:")

def dispstuddet(self):

print("-"\*40)

print("Student details:")

print("-"\*40)

print("Student Number:",self.sno)

print("Student Name:",self.sname)

print("Student Course:",self.crs)

print("-"\*40)

"""def savestuddata(self):

#PDBC CODE

con=cx\_Oracle.connect("scott/tiger@localhost/orcl")

cur=con.cursor()

cur.execute("insert into StudentDet values(%d,'%s','%s','%s','%s','%s','%s')" %(self.sno,self.sname,self.crs,self.cname,self.cloc,self.uname,self.uloc))

con.commit()

print("Student Data Saved in Table successfully")"""

#main program

s=Student()

s.getstuddet()

s.getcolldet()

s.getunivdet()

s.dispunivdet()

s.dispcolldet()

s.dispstuddet()

""""

create table StudentDet(sno number(2),sname varchar2(10),crs varchar2(10), cname varchar2(10),cloc varchar2(10),uname varchar2(10),uloc varchar2(10))

"""

#UnivCollStudDb.py

import cx\_Oracle

class Univ:

def getunivdet(self):

self.uname=input("Enter University Name:")

self.uloc=input("Enter University Location:")

def dispunivdet(self):

print("-"\*40)

print("University details:")

print("-"\*40)

print("University Name:",self.uname)

print("University Location:",self.uloc)

print("-"\*40)

class College(Univ):

def getcolldet(self):

self.cname=input("Enter College Name:")

self.cloc=input("Enter College Location:")

def dispcolldet(self):

print("-"\*40)

print("College details:")

print("-"\*40)

print("College Name:",self.cname)

print("College Location:",self.cloc)

print("-"\*40)

class Student(College):

def getstuddet(self):

self.sno=int(input("Enter Student Number:"))

self.sname=input("Enter Student Name:")

self.crs=input("Enter Student Course:")

def dispstuddet(self):

print("-"\*40)

print("Student details:")

print("-"\*40)

print("Student Number:",self.sno)

print("Student Name:",self.sname)

print("Student Course:",self.crs)

print("-"\*40)

def savestuddata(self):

#PDBC CODE

con=cx\_Oracle.connect("scott/tiger@localhost/orcl")

cur=con.cursor()

cur.execute("insert into StudentDet values(%d,'%s','%s','%s','%s','%s','%s')" %(self.sno,self.sname,self.crs,self.cname,self.cloc,self.uname,self.uloc))

con.commit()

print("Student Data Saved in Table successfully")

#main program

s=Student()

s.getstuddet()

s.getcolldet()

s.getunivdet()

s.dispunivdet()

s.dispcolldet()

s.dispstuddet()

s.savestuddata()

""""

create table StudentDet(sno number(2),sname varchar2(10),crs varchar2(10), cname varchar2(10),cloc varchar2(10),uname varchar2(10),uloc varchar2(10))

"""

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

Regular Expressions in Python--3 Days

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

Index

----------

=>Purpose of Regular Expressions

=>Applications of Regular Expressions

=>Definition of Regular Expressions

=>Module Name (re ) for dealing with Regular Expressions

=>Functions in re module

a) search()

b) findall()

c) finditer()

d) start()

e) end()

f) group()

=>Programming Examples

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

=>Programmer-Defined Classes in Regular Expressions

=>Programming Examples

=>Pre-Defined-Defined Classes in Regular Expressions

=>Programming Examples

=>Quantifiers in Regular Expressions

=>Programming Examples:

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

=>Combined Programming Examples on Programmer, Pre-Defined-Defined Classes and Quantifiers

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

Introduction to Regular Expressions in Python

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

=>Regular Expressions is one of the Indepedent Concept from Proramming Languages.

=>The purpose of Regular Expressions in Python is that " To Build Robust Application by Validation of Data".

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

Applications of Regular Expressions

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

=>Regular Expressions are used in development of Pattern Matching / text matching

=>Regular Expressions are used in development of Language Compilers and Interpreters.

=>Regular Expressions are used in development of Universal Protocols.

=>Regular Expressions are used in development of Electonic Cuircutary deisgning

=>Regular Expressions are used in development of Customized data Validation / Extractions from

given data.....many more.

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

Definition of Regular Expressions:

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

=>A Regular Expression is one of the Search Pattern which is a comibation of Alpabets,Digits and special symbols and it used to Search or Match or Find in given data and obtains Desired Result.

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

=>To deal with Regular Expressions programming / applications, we have a pre-defined module called "re".

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

Pre-defined Functions in re module

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

=>The 're' module contains the follwing essential Functions.

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

1) finditer():

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

Syntax:- varname=re.finditer("search-pattern","Given data")

=>here varname is an object of type <class,'Callable\_Itetaror'>

=>This function is used for searching the "search pattern" in given data iteratively and it returns table of entries which contains start index , end index and matched value based on the search pattern.

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

2) findall():

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

Syntax:- varname=re.findall("search-pattern","Given data")

=>here varname is an object of <class,'list'>

=>This function is used for searching the search pattern in entire given data and find all occurences / matches and it returns all the matched values in the form an object <class,'list'> but not returning Start and End Index Values.

-----------------------------------------------------------------------3) search():

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

Syntax:- varname=re.search("search-pattern","Given data")

=>here varname is an object of <class,'re.match'> or <class,'NoneType'>

=>This function is used for searching the search pattern in given data for first occuence / match only but not for other occurences / matches.

=>if the search pattern found in given data then it returns an object of match class which contains matched value and start and end index values and it indicates search is successful.

=>if the search pattern not found in given data, then it returns None which is type <class, "NoneType"> and it indicates search is un-successful

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

4) group():

-----------------------------------------------------------------------=>This function is used obtaining matched value by the object of match class

=>This function present in match class of re module

Syntax:- varname=matchtabobj.group()

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

5) start():

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

=>This function is used obtaining starting index of matched value

=>This function present in match class of re module

Syntax: varname=matchobj.start()

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

6) end():

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

=>This function is used obtaining end index+1 of matched value

=>This function present in match class of re module

Syntax: varname=matchobj.end()

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

7) sub() Function

=> This function replaces the matches with the text of your choice:

import re

txt = "The rain in Spain"

x = re.sub("\s", "9", txt)

print(x)---------------------------------- The9rain9in9Spain

#RegExpr1.py

import re

gd="Python is an oop lang. Python is also Func Prog lang"

sp="Python"

words=re.findall(sp,gd) # words= ["Python","Python"]

print(words, type(words)) # ['Python', 'Python'] <class 'list'>

print("\n{} time(s) '{}' available".format(len(words),sp))

#RegExpr2.py

import re

gd="Python is an oop lang. Python is also Func Prog lang"

sp="Python"

res=re.search(sp,gd) #

print("Type of res=",type(res)) # Success: <re.Match> Un-Success: NoneType

if(res!=None):

print("Search is Sucessful")

print("Satrting Index={}".format(res.start()))

print("End Index={}".format(res.end()))

print("Matched Value={}".format(res.group()))

else:

print("Search is Un-Sucessful")

print("{} is not Found:".format(sp))

#RegExpr3.py

import re

gd="Python is an oop lang. Python is also Func Prog lang"

sp="python"

mattable=re.finditer(sp,gd)

print(type(mattable)) # <class 'callable\_iterator'>

print("-"\*60)

for mat in mattable: # here mat is in an object of <class, re.match>

print ("Starting Index:{} End Index:{} Value:{}".format(mat.start(),mat.end(),mat.group()))

#RegExpr4.py

import re

gd="Python is an oop lang. Python is also Func Prog lang"

sp="Python"

mattable=re.finditer(sp,gd)

print(type(mattable)) # <class 'callable\_iterator'>

print("-"\*60)

cnt=0

for mat in mattable: # here mat is in an object of <class, re.match>

print("Starting Index:{} End Index:{} Value:{}".format(mat.start(),mat.end(),mat.group()))

cnt=cnt+1

print("-"\*60)

print("'{}' found {} time(s)".format(sp,cnt))

print("-"\*60)

#program for search either 'a' or 'b' or 'c' only

#RegExpr5.py

import re

mat=re.finditer("[abc]","cAh#6K\*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

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

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr5.py

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

Start Index:0 End Index:1 Value:c

Start Index:8 End Index:9 Value:a

Start Index:13 End Index:14 Value:b

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

"""

#program for search for all except ‘a' or 'b' or 'c'

#RegExpr6.py

import re

mat=re.finditer("[^abc]","cAh#6K\*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

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

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr6.py

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

Start Index:1 End Index:2 Value:A

Start Index:2 End Index:3 Value:h

Start Index:3 End Index:4 Value:#

Start Index:4 End Index:5 Value:6

Start Index:5 End Index:6 Value:K

Start Index:6 End Index:7 Value:\*

Start Index:7 End Index:8 Value:3

Start Index:9 End Index:10 Value:M

Start Index:10 End Index:11 Value:R

Start Index:11 End Index:12 Value:9

Start Index:12 End Index:13 Value:!

Start Index:14 End Index:15 Value:Q

Start Index:15 End Index:16 Value:T

Start Index:16 End Index:17 Value:6

Start Index:17 End Index:18 Value:%

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

"""

#program for search for all lower case alphabets

#RegExpr7.py

import re

mat=re.finditer("[a-z]","cAh#6K\*3aMR9!bQT6%")

noc=0

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

noc=noc+1

print("-"\*50)

print("No. of small alphabets=",noc)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr7.py

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

Start Index:0 End Index:1 Value:c

Start Index:2 End Index:3 Value:h

Start Index:8 End Index:9 Value:a

Start Index:13 End Index:14 Value:b

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

No. of small alphabets= 4

"""

#program for search for all except lower case alphabets

#RegExpr8.py

import re

mat=re.finditer("[^a-z]","cAh#6K\*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

noc=noc+1

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr8.py

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

Start Index:1 End Index:2 Value:A

Start Index:3 End Index:4 Value:#

Start Index:4 End Index:5 Value:6

Start Index:5 End Index:6 Value:K

Start Index:6 End Index:7 Value:\*

Start Index:7 End Index:8 Value:3

Start Index:9 End Index:10 Value:M

Start Index:10 End Index:11 Value:R

Start Index:11 End Index:12 Value:9

Start Index:12 End Index:13 Value:!

Start Index:14 End Index:15 Value:Q

Start Index:15 End Index:16 Value:T

Start Index:16 End Index:17 Value:6

Start Index:17 End Index:18 Value:%

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

"""

#program for search for all Upper case alphabets only

#RegExpr9.py

import re

mat=re.finditer("[A-Z]","cAh#6K\*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr9.py

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

Start Index:1 End Index:2 Value:A

Start Index:5 End Index:6 Value:K

Start Index:9 End Index:10 Value:M

Start Index:10 End Index:11 Value:R

Start Index:14 End Index:15 Value:Q

Start Index:15 End Index:16 Value:T

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

"""

#program for search for all except Upper case alphabets

#RegExpr10.py

import re

mat=re.finditer("[^A-Z]","cAh#6K\*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr10.py

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

Start Index:0 End Index:1 Value:c

Start Index:2 End Index:3 Value:h

Start Index:3 End Index:4 Value:#

Start Index:4 End Index:5 Value:6

Start Index:6 End Index:7 Value:\*

Start Index:7 End Index:8 Value:3

Start Index:8 End Index:9 Value:a

Start Index:11 End Index:12 Value:9

Start Index:12 End Index:13 Value:!

Start Index:13 End Index:14 Value:b

Start Index:16 End Index:17 Value:6

Start Index:17 End Index:18 Value:%

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

"""

#program for search for all Digits

#RegExpr11.py

import re

mat=re.finditer("[0-9]","cAh#6K\*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr11.py

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

Start Index:4 End Index:5 Value:6

Start Index:7 End Index:8 Value:3

Start Index:11 End Index:12 Value:9

Start Index:16 End Index:17 Value:6

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

""

#program for search for all except Digits

#RegExpr12.py

import re

mat=re.finditer("[^0-9]","cAh#6K\*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr12.py

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

Start Index:0 End Index:1 Value:c

Start Index:1 End Index:2 Value:A

Start Index:2 End Index:3 Value:h

Start Index:3 End Index:4 Value:#

Start Index:5 End Index:6 Value:K

Start Index:6 End Index:7 Value:\*

Start Index:8 End Index:9 Value:a

Start Index:9 End Index:10 Value:M

Start Index:10 End Index:11 Value:R

Start Index:12 End Index:13 Value:!

Start Index:13 End Index:14 Value:b

Start Index:14 End Index:15 Value:Q

Start Index:15 End Index:16 Value:T

Start Index:17 End Index:18 Value:%

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

"""

#program for searches for all alphabets

#RegExpr13.py

import re

mat=re.finditer("[A-Za-z]","cAh#6K\*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr13.py

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

Start Index:0 End Index:1 Value:c

Start Index:1 End Index:2 Value:A

Start Index:2 End Index:3 Value:h

Start Index:5 End Index:6 Value:K

Start Index:8 End Index:9 Value:a

Start Index:9 End Index:10 Value:M

Start Index:10 End Index:11 Value:R

Start Index:13 End Index:14 Value:b

Start Index:14 End Index:15 Value:Q

Start Index:15 End Index:16 Value:T

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

"""

program for searches for all except alphabets

#RegExpr14.py

import re

mat=re.finditer("[^A-Za-z]","cAh#6K\*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr14.py

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

Start Index:3 End Index:4 Value:#

Start Index:4 End Index:5 Value:6

Start Index:6 End Index:7 Value:\*

Start Index:7 End Index:8 Value:3

Start Index:11 End Index:12 Value:9

Start Index:12 End Index:13 Value:!

Start Index:16 End Index:17 Value:6

Start Index:17 End Index:18 Value:%

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

"""

#program for searches for all Alphabets and Digits (Alpha nums)

#RegExpr15.py

import re

mat=re.finditer("[A-Za-z0-9]","cAh#6K\*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr15.py

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

Start Index:0 End Index:1 Value:c

Start Index:1 End Index:2 Value:A

Start Index:2 End Index:3 Value:h

Start Index:4 End Index:5 Value:6

Start Index:5 End Index:6 Value:K

Start Index:7 End Index:8 Value:3

Start Index:8 End Index:9 Value:a

Start Index:9 End Index:10 Value:M

Start Index:10 End Index:11 Value:R

Start Index:11 End Index:12 Value:9

Start Index:13 End Index:14 Value:b

Start Index:14 End Index:15 Value:Q

Start Index:15 End Index:16 Value:T

Start Index:16 End Index:17 Value:6

--------------------------------------------------"""

#program for searching Special Symbols

#RegExpr16.py

import re

mat=re.finditer("[^A-Za-z0-9]","cAh#6K\*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

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

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr16.py

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

Start Index:3 End Index:4 Value:#

Start Index:6 End Index:7 Value:\*

Start Index:12 End Index:13 Value:!

Start Index:17 End Index:18 Value:%

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

"""

#program for searching Space Character only

#RegExpr17.py

import re

mat=re.finditer("\s"," cAh#6K \*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr17.py

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

Start Index:0 End Index:1 Value:

Start Index:7 End Index:8 Value:

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

"""

#program for searching all except Space Character

#RegExpr18.py

import re

mat=re.finditer("\S"," cAh#6K \*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr18.py

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

Start Index:1 End Index:2 Value:c

Start Index:2 End Index:3 Value:A

Start Index:3 End Index:4 Value:h

Start Index:4 End Index:5 Value:#

Start Index:5 End Index:6 Value:6

Start Index:6 End Index:7 Value:K

Start Index:8 End Index:9 Value:\*

Start Index:9 End Index:10 Value:3

Start Index:10 End Index:11 Value:a

Start Index:11 End Index:12 Value:M

Start Index:12 End Index:13 Value:R

Start Index:13 End Index:14 Value:9

Start Index:14 End Index:15 Value:!

Start Index:15 End Index:16 Value:b

Start Index:16 End Index:17 Value:Q

Start Index:17 End Index:18 Value:T

Start Index:18 End Index:19 Value:6

Start Index:19 End Index:20 Value:%

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

"""

#program for searching all digits

#RegExpr19.py

import re

mat=re.finditer("\d"," cAh#6K \*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr19.py

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

Start Index:5 End Index:6 Value:6

Start Index:9 End Index:10 Value:3

Start Index:13 End Index:14 Value:9

Start Index:18 End Index:19 Value:6

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

"""

#program for searching all except Digits

#RegExpr20.py

import re

mat=re.finditer("\D"," cAh#6K \*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr20.py

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

Start Index:0 End Index:1 Value:

Start Index:1 End Index:2 Value:c

Start Index:2 End Index:3 Value:A

Start Index:3 End Index:4 Value:h

Start Index:4 End Index:5 Value:#

Start Index:6 End Index:7 Value:K

Start Index:7 End Index:8 Value:

Start Index:8 End Index:9 Value:\*

Start Index:10 End Index:11 Value:a

Start Index:11 End Index:12 Value:M

Start Index:12 End Index:13 Value:R

Start Index:14 End Index:15 Value:!

Start Index:15 End Index:16 Value:b

Start Index:16 End Index:17 Value:Q

Start Index:17 End Index:18 Value:T

Start Index:19 End Index:20 Value:%

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

"""

#program for searching all word Characters

#RegExpr21.py

import re

mat=re.finditer("\w"," cAh#6K \*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr21.py

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

Start Index:1 End Index:2 Value:c

Start Index:2 End Index:3 Value:A

Start Index:3 End Index:4 Value:h

Start Index:5 End Index:6 Value:6

Start Index:6 End Index:7 Value:K

Start Index:9 End Index:10 Value:3

Start Index:10 End Index:11 Value:a

Start Index:11 End Index:12 Value:M

Start Index:12 End Index:13 Value:R

Start Index:13 End Index:14 Value:9

Start Index:15 End Index:16 Value:b

Start Index:16 End Index:17 Value:Q

Start Index:17 End Index:18 Value:T

Start Index:18 End Index:19 Value:6

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

"""

#program for searching all Special Symbols Except word Characters

#RegExpr22.py

import re

mat=re.finditer("\W"," cAh#6K \*3aMR9!bQT6%")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr22.py

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

Start Index:0 End Index:1 Value:

Start Index:4 End Index:5 Value:#

Start Index:7 End Index:8 Value:

Start Index:8 End Index:9 Value:\*

Start Index:14 End Index:15 Value:!

Start Index:19 End Index:20 Value:%

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

"""

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

Pre-Defined Character Classes in Reg Expr

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

=>Pre-Defined Character Classes are those which are already available in Python software and they are used for Preparing or Designing Search Pattern for searching in given data for obtaing Desired Result.

=>Syntax for Using Pre-Defined Character Classes

" \Search Pattern "

=>The Following are Pre-Defined Character Classes.

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

1) \s----------->Searches for Space Character only

2) \S----------->Searches for all except Space Character

3) \d----------->Searches for Digits only OR [0-9]

4) \D----------->Searches for all except Digits OR [^0-9]

5) \w---------->Searches for Word Character OR [A-Za-z0-9]

6) \W---------->Searches for Special Symbols OR [^A-Za-z0-9]

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

Quantifiers in Regular Expressions

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

=>The purpose of Quantifiers in Regular Expressions is that "To Search for zero or one more Value in given data".

=>Quantifiers are also used in Search Patterns to search in given data for obtaining desired result.

=>The essential Quantifiers in Regular Expressions are given bellow

1. k------------->Searches for Exactly one 'k'

2. k+------------>Searches for One k or More k's

3. k\*------------>Searches for either zero k or 1 k or more k's

4. k?------------>Searches for either zero k or one k

5. . --------------->Searches for all

Imp Special Points

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

1) \d+----->Searches one or more digits (OR) [0-9]+

2) \ddd----->Searches for 3 Digit Number OR \d{3} \d{10} (OR) [0-9]{3}

3) \d{n}----->Searches for N-Digit Number OR [0-9]{n}

4) \d{n}.\d{m}---->Searches N-Digit Integer and M-Digit floating point value(Ex 34.56 56.78..)

5) \w{n}----->Seraches n-length word OR [A-Za-z0-9]{n}

6) \w+------>Searches for either one word char or more word Characters.(OR) [A-Za-z0-9]+

7) \d{n,m }--->Searches for Min n-Digit Number and Maximum m-digit number

OR [0-9]{n,m}

#program for searching Exatcly One k

#RegExpr23.py

import re

mat=re.finditer("k","kkvkkvkkkvkv")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr23.py

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

Start Index:0 End Index:1 Value:k

Start Index:1 End Index:2 Value:k

Start Index:3 End Index:4 Value:k

Start Index:4 End Index:5 Value:k

Start Index:6 End Index:7 Value:k

Start Index:7 End Index:8 Value:k

Start Index:8 End Index:9 Value:k

Start Index:10 End Index:11 Value:k

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

"""

#program for searching one or more k's

#RegExpr24.py

import re

mat=re.finditer("k+","kvkkvkkkvkv")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr24.py

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

Start Index:0 End Index:1 Value:k

Start Index:2 End Index:4 Value:kk

Start Index:5 End Index:8 Value:kkk

Start Index:9 End Index:10 Value:k

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

"""

#program for searching zero k or one k or more k's

#RegExpr25.py

import re

mat=re.finditer("k\*","kvkkvkkkvkv")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr25.py

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

Start Index:0 End Index:1 Value:k

Start Index:1 End Index:1 Value:

Start Index:2 End Index:4 Value:kk

Start Index:4 End Index:4 Value:

Start Index:5 End Index:8 Value:kkk

Start Index:8 End Index:8 Value:

Start Index:9 End Index:10 Value:k

Start Index:10 End Index:10 Value:

Start Index:11 End Index:11 Value:

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

"""

#program for searching zero k or one k

#RegExpr26.py

import re

mat=re.finditer("k?","kvkkvkkkvkv")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr26.py

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

Start Index:0 End Index:1 Value:k

Start Index:1 End Index:1 Value:

Start Index:2 End Index:3 Value:k

Start Index:3 End Index:4 Value:k

Start Index:4 End Index:4 Value:

Start Index:5 End Index:6 Value:k

Start Index:6 End Index:7 Value:k

Start Index:7 End Index:8 Value:k

Start Index:8 End Index:8 Value:

Start Index:9 End Index:10 Value:k

Start Index:10 End Index:10 Value:

Start Index:11 End Index:11 Value:

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

"""

#program for searching for all occurences

#RegExpr27.py

import re

mat=re.finditer(".","kvkkvkkkvkv")

print("-"\*50)

for m in mat:

print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))

print("-"\*50)

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr27.py

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

Start Index:0 End Index:1 Value:k

Start Index:1 End Index:2 Value:v

Start Index:2 End Index:3 Value:k

Start Index:3 End Index:4 Value:k

Start Index:4 End Index:5 Value:v

Start Index:5 End Index:6 Value:k

Start Index:6 End Index:7 Value:k

Start Index:7 End Index:8 Value:k

Start Index:8 End Index:9 Value:v

Start Index:9 End Index:10 Value:k

Start Index:10 End Index:11 Value:v

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

"

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

Programmer-Defined Character Classes in Reg Expr

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

=>Programmer-Defined Character Classes are those which defined by Programmers and they are for Preparing or Designing Search Pattern for searching in given data for obtaing Desired Result.

=>Syntax for Using Programmer-Defined Character Classes

" [ Search Pattern ] "

=>The Following are Programmer-Defined Character Classes.

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

1. [abc]------->Searches for either a or b or c

2. [^abc]----->Searches for all except a or b or c

3. [a-z]-------->Searches for all Lower case alphabets only

4. [^a-z]------>Searches for all except Lower Case Alphabets

5. [A-Z]------->Searches for all Upper Case Alphabets

6.[^A-Z]------->Searches for all except Upper Case Alphabets

7. [0-9]------->Searches for all Digits only

8. [^0-9]----->Searches for all except Digits

9. [A-Za-z]---->Searches for all Alphabets (Lower and Upper) only

10. [^A-Za-z]---->Searches for all except Alphabets (Lower and Upper)

11. [A-Za-z0-9]--->Searches for all Alphabers and Digits(Alpha numberics--Word Character)

12. [^A-Za-z0-9]--->Searches for all Special Symbols ( ie Not Alphabets and Digits)

13. [a-z0-9]----->Searches for lower alphabets and digits

14. [^a-z0-9]----->Searches for all except lower alphabets and digits

15. [A-Z0-9]---->Sreaches for all Upper case Alphabets and Digits

16. [^A-Z0-9]---->Sreaches for all except Upper case Alphabets and Digits

#Program for validating and valid mailds from KBD

#MailldsValidation.py

import re

lst=[]

while(True):

mail=input("Enter Mail Id:")

res=re.search("\S+@\S+",mail)

if(res!=None):

lst.append(mail)

ch=input("Do u want to insert another e-mail id(yes/no):")

if(ch.lower()=="no"):

break

else:

print("Invalid mail--try again")

print("Valid mails:{}".format(lst))

#Program for extrating mail-id from given string data

#MailsExamples1.py

import re

gd="Rossum maild is rossum123@psf.com , Dennis maild is dennis\_c@bellland.co.in , Travis maild is travis\_numpy@numpy.org , Kinney maild is kinney\_pandas@pandas.net.in and James maild is james.java@sun.com"

sp="\S+@\S+"

mails=re.findall(sp,gd)

print("List of Mails:")

for mail in mails:

print("\t{}".format(mail))

#Program for Extracting the Marks from given Data

#MarksListEx.py

import re

gd="Rossum got 22 marks , Dennis got 11 marks , Travis got 44 marks , Kinney got 55 marks and James got 999 marks"

sp="\d{2,3}"

markslist=re.findall(sp,gd)

for marks in markslist:

print("\t{}".format(marks))

#Program for vcalidating Mobile Number

#MobileNumberValidationEx1.py

import re

while(True):

mno=input("Enter Mobile Number:")

if(len(mno)==10):

res=re.search("\d{10}",mno)

if(res!=None):

print("Ur Mobile Number is Valid:")

break

else:

print("Invalid Mobile Number bcoz It contains non-numeric")

else:

print("Invalid Mobile Number bcoz length is >10--try again")

#Program for validating Mobile Number where it starts 98xxxxxxxx

#MobileNumberValidationEx2.py

import re

while(True):

mno=input("Enter Mobile Number:")

if(len(mno)==10):

res=re.search("98\d{8}",mno)

if(res!=None):

print("Ur Mobile Number is Valid:")

break

else:

print("Invalid Mobile Number bcoz It contains non-numeric")

else:

print("Invalid Mobile Number bcoz length is >10--try again")

#Program for Extracting the Names from given Data

#NamesListEx.py

import re

gd="Rossum got 22 marks , Dennis got 11 marks , Travis got 44 marks , Kinney got 55 marks and James got 99 marks"

sp="[A-Z][a-z]+"

print("--------By using findall()----------")

nameslist=re.findall(sp,gd)

print("Names of Student:")

for name in nameslist:

print("\t{}".format(name))

print("-------------------------------------------")

print("--------By using finditer()----------")

matchnames=re.finditer(sp,gd)

print("Names of Student:")

for name in matchnames:

print("Start Index:{} End Index:{} Name:{}".format(name.start(),name.end(),name.group()))

print("-------------------------------------------")

#Program for extrating names and mail-ids from maisdata.data file

#NamesMailsExamples2.py

import re

with open("mailsdata.data","r") as fp:

filedata=fp.read()

mails=re.findall("\S+@\S+",filedata)

nameslist=re.findall("[A-Z][a-z]+",filedata)

print("\tnames\tMails:")

for names,mail in zip(nameslist,mails):

print("\t{}\t{}".format(names,mail))

#Program for Extracting the Names and Marks from given Data

#NamesMarksListEx.py

import re

gd="Rossum got 22 marks , Dennis got 11 marks , Travis got 44 marks , Kinney got 55 marks and James got 999 marks"

sp1="\d{2,3}"

sp2="[A-Z][a-z]+"

markslist=re.findall(sp1,gd)

nameslist=re.findall(sp2,gd)

print("-"\*50)

print("\tNames\tMarks")

print("-"\*50)

for names,marks in zip(nameslist,markslist):

print("\t{}\t{}".format(names,marks))

print("-"\*50)

#Program extracting the names and marks from stud.info file

#studnamesmarksfiles.py

import re

with open("stud.info","r") as fp:

filedata=fp.read()

nameslist=re.findall("[A-Z][a-z]+",filedata)

markslist=re.findall("\d{2,3}",filedata)

print("-"\*50)

print("\tNames\tMarks")

print("-"\*50)

for names,marks in zip(nameslist,markslist):

print("\t{}\t{}".format(names,marks))

print("-"\*50)

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

generator in python

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

=>generator is one of the function

=>The generator function always contains yield keyword

=>If the function contains return statement then it is called Normal Function

=>If the function contains yield keyword then it is called generator

=>Syntax:

def function\_name(start,stop,step):

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

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

yield value

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

=>The 'yield' key word is used for giving the value back to function call from function defintion and continue the function execution until condition becomes false.

=>The advantage of generators over functions concept is that it save lot of memory space in the case large sampling of data. In otherwords Functions gives all the result at once and it take more memory space where as generators gives one value at a time when programmer requested and takes minimized memory space.

========================X===================================

#non-genex.py

r=range(1,2000)

print(r)

for v in r:

print(v)

print("-------------------------------")

#GenEx1.py

def kvrrange(b,e):

while(b<=e):

yield b

b=b+1

#main program

k=kvrrange(10,15) # here k is an object <class, generator>

while(True):

try:

print(next(k))

except StopIteration:

break

#GenEx2.py

def kvrrange(b,e,s):

while(b<=e):

yield b

b=b+s

#main program

k=kvrrange(10,50,5) # here k is an object <class, generator>

while(True):

try:

print(next(k))

except StopIteration:

break