'''

(1)=> Write a program to create a Student class with name,age and marks as data members. Also

create a method named display() to view the student details. Create an object to Student

class and call the method using the object.

'''

class student:

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

self.name = "xyz"

self.age = 18

self.marks = 90

def display(self):

print("Name=>",self.name)

print("Age=>",self.age)

print("Marks=>",self.marks)

s=student()

s.display()

'''

(2)=> Write a program to create student class with a constructor having more than one parameters.

'''

class student:

def \_\_init\_\_(self,nm = "none",ag=15,m=0):

self.name = nm

self.age = ag

self.marks = m

def display(self):

print("Name=>",self.name)

print("Age=>",self.age)

print("Marks=>",self.marks)

s=student("ABC",20,91)

s1=student()

s.display()

s1.display()

'''

(3)=> write a program to demostrate the use of instance and class/static variables.

'''

print("Use of class variable")

class sample:

var = 10 *#class variable*

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

self.x = 10 *#instance variable*

def modify(self):

self.x += 1

@classmethod

def new\_modify(cls):

cls.var += 1

s=sample()

print("Class Varible=>",sample.var)

sample.new\_modify() *#increment of class variable*

print("Class Varible=>",sample.var)

s1 = sample()

print("Instance Varible =>",s1.x)

s1.modify()

print("Instance Varible =>",s1.x)

'''

(4)=> Write a program to store data into instances usng mutator methods and to retrieve

data from the instances using accessor methods.

'''

class student:

*#mutator method*

def setname(self,name):

self.name = name

*#accesor method*

def getname(self):

return self.name

def setmarks(self,marks):

self.marks = marks

def getmarks(self):

return self.marks

n=int(input("How many student=>"))

i=0

while(i<n):

s=student()

name = input("Enter name->")

s.setname(name)

marks = int(input("Enter marks->"))

s.setmarks(marks)

print("Name=>",s.getname())

print("Marks=>",s.getmarks())

i += 1

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

'''

(5)=> Write a program to use class method to handle the common feature of all the

instance of studente class.

'''

class student:

dept = "BCA"

total\_student = 0

def \_\_init\_\_(self,name,age):

self.name = name

self.age = age

student.increment()

@classmethod

def increment(cls):

cls.total\_student += 1

@classmethod

def return\_i(cls):

return cls.total\_student

def display(self):

print("Name=>",self.name)

print("Age=>",self.age)

print("Department=>",student.dept)

Obj1 = student("abc",43)

Obj1.display()

Obj2 = student("xyz",65)

Obj2.display()

print("Total student =>",student.return\_i())

'''

(6) => write a program to create a static method that counts the number

of instances created for a class.

'''

class test:

n = 0

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

test.n += 1

@staticmethod

def num():

print("Number of object created=>",test.n)

t1=test()

t2=test()

t3=test()

test.num()

'''

(7)=> Create a Bank class with two variables name and balance. Implement a constructor

to initalize the variables. Also implement deposit and withdrawls using instance

methods.

'''

import sys

class Bank:

def \_\_init\_\_(self,name,balance = 0.0):

self.name = name

self.balance = balance

def deposit(self,amount):

self.balance += amount

return self.balance

def withdrawls(self,amount):

if amount > self.balance:

print("Low Balance!!, cannot withdraw")

sys.exit()

else:

self.balance -= amount

name = input("Enter name->")

b=Bank(name)

while True:

print("\n1.Deposit\n2.Withdraw\n3.Exit")

choice = int(input("Enter Your choice->"))

if choice == 1:

amount = float(input("Enter Amount->"))

print("Balance After Deposit=>",b.deposit(amount))

elif choice == 2:

amount = float(input("Enter Amount->"))

print("Balance After withdraw=>",b.withdrawls(amount))

elif choice == 3:

sys.exit()

'''

(8)=> Write a program to create a Emp class and make all the memebers of the

emp class available to another class(myclass).[by passing memembers of one class to another]

'''

class emp:

def \_\_init\_\_(self,id,name,sal):

self.id = id

self.name = name

self.salary = sal

def display(self):

print("ID=>",self.id)

print("Name=>",self.name)

print("Salary=>",self.salary)

class myclass:

@staticmethod

def mymethod(e):

e.salary += 1000

e.display()

e=emp(1001,"idk",5000)

myclass.mymethod(e)

'''

(9)=> Create a student class to with the methods of set\_id, get\_id, set\_name,get\_name , set\_marks

and get\_marks where the method name starting with set are sued to assign the values and method

name starting with get are running the values. Save the program by student.py

create another program to use the student class which is already available in

student.py

'''

#student.py (five9.py)

class student:

def setid(self,id):

self.id = id

def getid(self):

return self.id

def setname(self,name):

self.name = name

def getname(self):

return self.name

def setmarks(self,marks):

self.marks = marks

def getmarks(self):

return self.marks

#main.py

from five9 import student

s=student()

s.setid(1000)

s.setname("abc")

s.setmarks(50)

print("Student ID=.",s.getid())

print("Student Name=>",s.getname())

print("Student Marks=>",s.getmarks())

'''

(10)=> write a program to access the base class constructor from a sub class

by using super() method and also without using super() method.

'''

class base:

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

print("welcome to base class")

class superClass(base):

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

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

print("Welcome to parent class")

class superClasswithoutsuper(base):

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

base.\_\_init\_\_(self)

print("you called base class constructor without super()")

s=superClass()

s1=superClasswithoutsuper()

'''

(11)=> Write a program to override super class constructor and method in sub class.

'''

class Teacher:

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

self.id = 1001

print("Teacher constrctor called")

def display(self):

print("Teacher ID=>",self.id)

class Student(Teacher):

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

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

self.id = 1

print("Student conctructor called")

def display(self):

print("Student ID=>",self.id)

s=Student()

s.display()

'''

(12)=> Write a program to implement single inheritence in which two sub classes

are derived from a single base class.

'''

class parent:

id = 1

def display(self):

print("parent class")

class child(parent):

def display(self):

print("id=>",self.id)

class child2(parent):

def display(self):

print("id=>",self.id)

one= child()

one.display()

two = child2()

two.display()

'''

(13)=> Write a program to implement multiple inheritence using two base classes.

'''

class a:

def ab(self):

print("class a")

class b:

def ba(self):

print("class b")

class c(a,b):

def show(self):

self.ab()

self.ba()

c1=c()

c1.show()

'''(14)=> Write a program to understand the order of execution of methonds in

several base class according to method resultion order(MRO).

'''

class a(object):

def method(self):

print("class a called")

super().method()

class b(object):

def method(self):

print("class b called")

super().method()

class c(object):

def method(self):

print("class c called")

super().method()

class x(a,b):

def method(self):

print("class x called")

super().method()

class y(b,c):

def method(self):

print("class y called")

super().method()

class p(x,y,c):

def method(self):

print("class p called")

obj=p()

print(p.mro())

obj.method()

'''

(15)=> Write a program to check the object type to know whether

the method exists in the object or not.

'''

class test:

def abc(self):

print("method exist")

obj1=test()

def checking(obj,methodName):

if hasattr(obj,methodName):

print(f"method '{methodName}' exist in the object")

else:

print(f"method '{methodName}' not exist in this object")

checking(obj1,"abc")

checking(obj1,"noting")

'''

(16)=> Write a program to overload the addition operator(+) to

make it act on the class objects.

'''

class bookx:

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

self.pages = pages

def \_\_add\_\_(self,other):

return self.pages + other.pages

class booky:

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

self.pages = pages

b1=bookx(500)

b2=booky(1000)

print("Total Pages=>",(b1+b2))

'''

(17)=> write a program to show method overloading to find sum o f

two or there numbers.

'''

class myclass:

def sum(self,a=None,b=None,c=None):

if a != None and b != None and c != None:

print("Additon=>",(a+b+c))

elif a != None and b != None:

print("Addition=>",(a+b))

else:

print("pass atleast 2 or 3 argument")

x=myclass()

x.sum(10,20)

x.sum(101,1,2)

x.sum()

'''

(18)=> write a program to override the super class method in subclass.

'''

class base:

def display(self):

print("super class")

class child(base):

def display(self):

print("child class")

c=child()

c.display()