Batch Name: Summer Internship 2018

Enrolment No: **R610217007**
SAPID: **500062812**
Name: **DIVYANSHU SINGH**
Sem: **II**
Branch: **CSE –MAINFRAME TECHNOLOGY**

FP5.0 Module-4 Assignments

Assignment 1

Identify Entities, list their properties (attributes) in following scenario:
a.in a college library
b.in a classroom

College Library
•List of Books
Librarian
•Name
•Employee id
Book
•Book Id
•Title
•Author
•Publisher
Issued Book
•Date of issue
•Book Id
•Borrower
Borrower
•Type(e.g.student/teacher)
•Borrowers id (e.g. enrollment number, employee id)

Classroom
•Whiteboard
•Desk
•Bench
Student
•Roll no.

•Name
Teacher
•Name
•Department
•Expertise (i.e. list of Subjects)

Assignment 2

•List properties (attributes) and behaviors(methods) for following entities:
•A Facebook account
•Bank Account
•Employee
#
Facebook Account
Name
Age
Gender
Address
Job
Email
Password
Education
Profile Picture
Cover Picture

Bank Account
Name
Age
Gender
Address
Email
Account no.
Balance
Loan
Savings
Current

Employee
Name
Age
Address
Gender
Email
Education

Designation
Salary
Employee ID

Relationships
A Bank Account can have an Employee who has a Facebook account

Bank Account -> Employee -> Facebook Account

Assignment 3

List entities, their properties in an online shopping website from following description:

♣The registration page takes customer name, mobile number, email id as input from new customer. A customer id is generated by the website on successful registration.

♣On login, customer can view the list of product. For every product- product name, id, prize, description and its picture is displayed.

♣The selected items by customer go to shopping cart. The shopping cart displays list of ordered items with their id, name, quantity and price.

♣Customer can then place the order by entering shipping detail.

♣The shop owner can see customer order information like order id, date of purchase, shipping address, customer information and every ordered item details like product id, product name, price and quantity

#

Customer: id, name, mobile number, email id
•Product: id, name, price, description, image path (since image will be separately stored)
•Shopping Cart:
    ♣list of Ordered_item
•Ordered_item:Product id, product name, price, quantity
•Order
    ♣Order id
    ♣Date of purchase
    ♣Shipping address
    ♣Customer id or Customer depending on design/logic
    ♣List of Ordered_item
#

Assignment 4

Create a class Employee with following properties
•First Name
•Last Name
•Pay
•Email : should be automatically generated as
•Firstname + ’.’ + Lastname + “@company.com”
•Test the code with following information of an Employee:
•First name is : Mohandas
•Last name is : Gandhi
•Pay is : 50000

```
#
class Employee:
    def __init__(self,first,last,pay):
        self.firstname = first
        self.lastname = last
        self.pay = pay
        self.email =first+'.'+last+'@company.com'
emp_1 = Employee('Mohandas','Gandhi',50000)
print(emp_1.firstname)
print(emp_1.lastname)
print(emp_1.pay)
print(emp_1.email)
#
```

```
class Employee:
    def __init__(self,first,last,pay):
        self.firstname = first
        self.lastname = last
        self.pay = pay
        self.email =first+'.'+last+'@company.co
emp_1 = Employee('Mohandas','Gandhi',50000)
print(emp_1.firstname)
print(emp_1.lastname)
print(emp_1.pay)
print(emp_1.email)
```

Console

```
<terminated> A:\Python1\Module 4\Assg4.py
Mohandas
Gandhi
50000
Mohandas.Gandhi@company.com
```

Assignment 5

In the previous example add following methods
•getEmail : should return the email id
•getFullName : should return full name (first name followed by last name)
•getPay : should return the pay
•Test the implementation with object of emp_1 as follows
emp_1 = Employee('Mohandas',‘Gandhi',50000)
print(emp_1.getFullName())
print(emp_1.getPay())
print(emp_1.getEmail())

```
#
class Employee:
    def __init__(self,first,last,pay):
        self.firstname = first
        self.lastname = last
        self.pay = pay
        self.email =first+'.'+last+'@company.com'
    def getmail(self):
        mail=self.email
        return mail
    def getfullname(self):
        full=self.firstname+' '+self.lastname
        return full
    def getpay(self):
        pay=self.pay
        return pay
emp_1 = Employee('Mohandas','Gandhi',50000)
print(emp_1.getfullname())
print(emp_1.getpay())
print(emp_1.getmail())
#
```

Assg4

```python
class Employee:
    def __init__(self,first,last,pay):
        self.firstname = first
        self.lastname = last
        self.pay = pay
        self.email =first+'.'+last+'@company.com'
    def getmail(self):
        mail=self.email
        return mail
    def getfullname(self):
        full=self.firstname+' '+self.lastname
        return full
    def getpay(self):
        pay=self.pay
        return pay
emp_1 = Employee('Mohandas','Gandhi',50000)
print(emp_1.getfullname())
print(emp_1.getpay())
print(emp_1.getmail())
```

Console

```
<terminated> A:\Python1\Module 4\Assg4.py
Mohandas Gandhi
50000
Mohandas.Gandhi@company.com
```

Assignment 6

Q: List the risk associated with the implementation of Account class. Suggest a solution.

```
class Account:
def __init__(self, initial_amount):
self.balance = initial_amount
def withdraw(self,amount):
self.balance = self.balance - amount
def deposit(self,amount):
self.balance = self.balance + amount
ac = Account(1000)
ac.balance = 2000 #stmt1
```

```
ac.balance = -1000 #stmt2
print(ac.balance) #stmt3

#
```

•The balance can be set to very high/low value accidentally. (#stmt1)
•The balance can be accessed or changed by user of the class.
•The balance can be set to non-permitted value (#stmt2)
Make balance a private variable (__balance)

```
def __init__(self, initial_amount):
self.__balance = initial_amount
#
```

Assignment 7

Q: A dog trainer had two dogs- Fido and Buddy.Fido was trained a trick of "roll over" and Buddy was learned "play dead". Is the code written correctly to represent this situation?

A. Yes

•prove by printing print(d.tricks)and print(e.tricks)

B. No

•if no, then rewrite the code

```
# NO
class Dog:
    def __init__(self, name):
        self.name = name
    def add_trick(self, trick):
        self.tricks=trick
d = Dog('Fido')
e = Dog('Buddy')
d.add_trick('roll over')
e.add_trick('play dead')
print(d.name,d.tricks)
print(e.name,e.tricks)
#
```

```python
class Dog:
    def __init__(self, name):
        self.name = name
    def add_trick(self, trick):
        self.tricks=trick
d = Dog('Fido')
e = Dog('Buddy')
d.add_trick('roll over')
e.add_trick('play dead')
print(d.name,d.tricks)
print(e.name,e.tricks)
```

Console

```
<terminated> A:\Python1\Module 4\Assg7.py
Fido roll over
Buddy play dead
```

Assignment 8

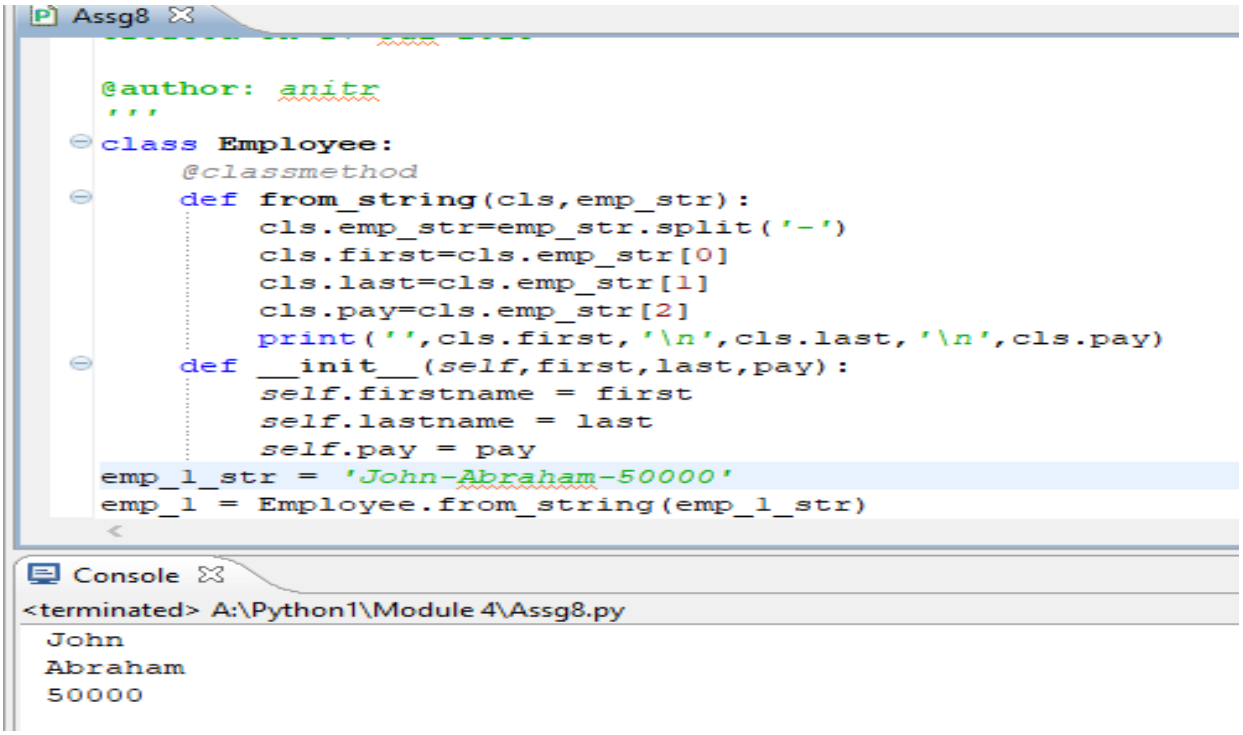
Q: Write logic for from_stringmethod such that it becomes alternative constructor; meaning it should create an object of Employee at#stmt1 with first name last name and pay values from emp_1_str string.

```
class Employee:
@classmethod
def from_string(cls,emp_str):
<Write logic here>
def __init__(self,first,last,pay):
self.firstname = first
self.lastname = last
self.pay = pay
emp_1_str = 'John-Abraham-50000'
emp_1 = Employee.from_string(emp_1_str) #stmt1
```

#

```python
class Employee:
    @classmethod
    def from_string(cls,emp_str):
        cls.emp_str=emp_str.split('-')
        cls.first=cls.emp_str[0]
        cls.last=cls.emp_str[1]
        cls.pay=cls.emp_str[2]
        print('',cls.first,'\n',cls.last,'\n',cls.pay)
    def __init__(self,first,last,pay):
```

```
        self.firstname = first
        self.lastname = last
        self.pay = pay
emp_1_str = 'John-Abraham-50000'
emp_1 = Employee.from_string(emp_1_str)
#
```



Assignment 9

Requirement:

Both the counters (counter1 & counter2) in following code, access the same __item_count from Store.

User can get the number of items in store by calling getItemCount method.

counter1 = Store()

counter2 = Store()

#add 2 items to store from counter1

#issue 1 item at counter1

#getItemCount in the Store

Q1: Provide body for the 3 methods. Logic is as follows:

1.addItem (count):

__item_count += count

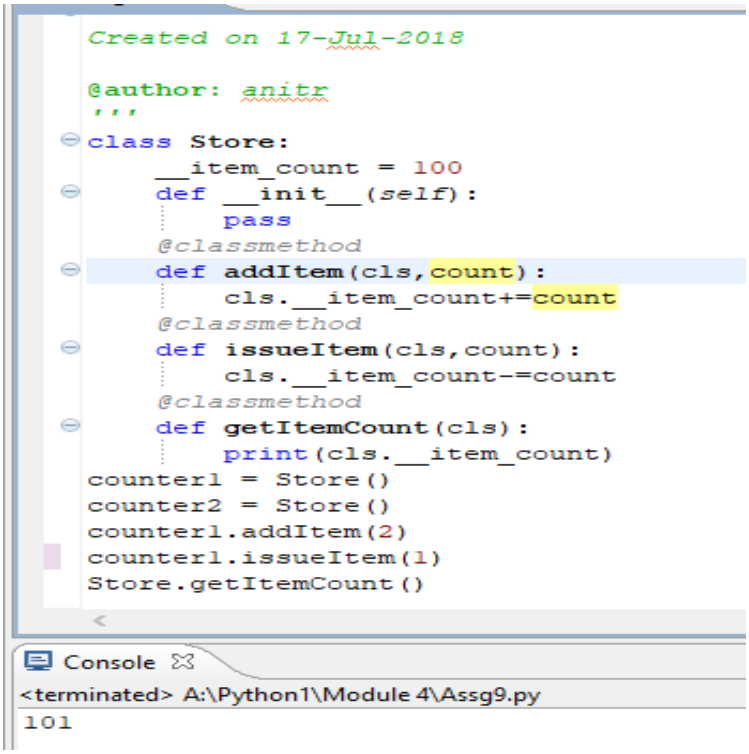
2.issueItem (count):

__item_count -= count

3.getItemCount(): returns

__item_count
Q2: Justify method type (instance or static or class) for each method. Test your logic for above requirement.

#
```
class Store:
    __item_count = 100
    def __init__(self):
        pass
    @classmethod
    def addItem(cls,count):
        cls.__item_count+=count
    @classmethod
    def issueItem(cls,count):
        cls.__item_count-=count
    @classmethod
    def getItemCount(cls):
        print(cls.__item_count)
counter1 = Store()
counter2 = Store()
counter1.addItem(2)
counter1.issueItem(1)
Store.getItemCount()
```
We have used Class Method Because we are using a class variable which remains unaffected under instance method.
#



Assignment 10

Q. Why objects can access class methods and class variables but class methods can not access instance methods or variables?

#

•The objects have information/knowledge about class from which it is instantiated but class does not contain information about objects created.

•The python runtime creates only one copy of static and class members (method and variable) and all instances share the same.

•Whereas, individual copies of instance members are created with respective objects.

•The instance handle, self is not available inside class and static methods.

#

Assignment 11

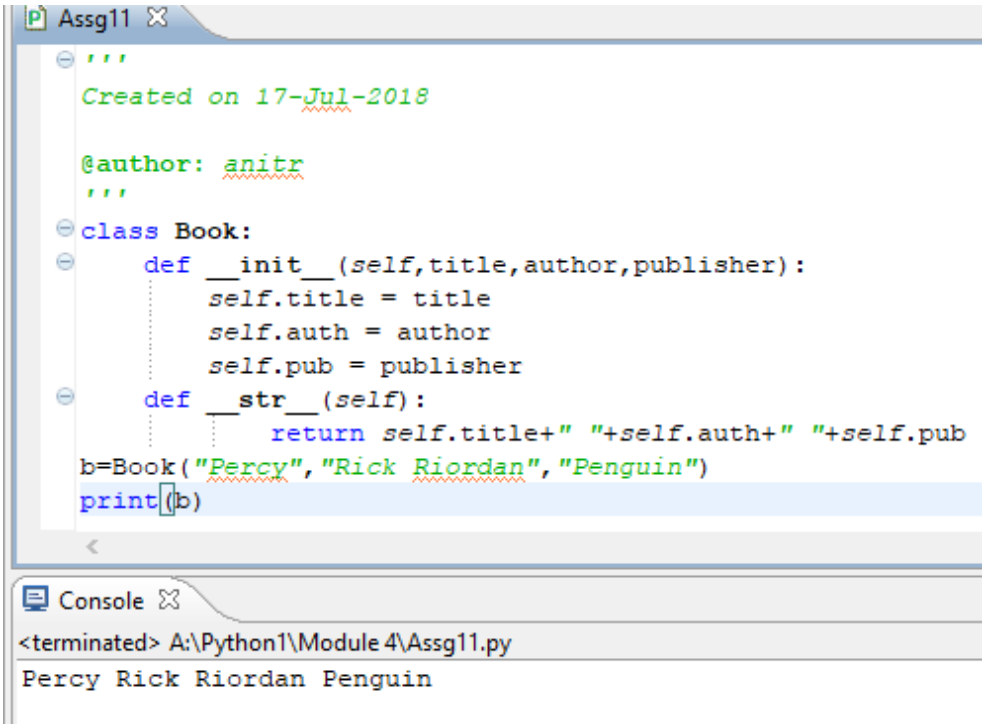
•The title, author and publisher information make key representations for a book.

•Q: print(b) should print these key information about "b" which is an object of a Book class. Create class Book.

#

```
class Book:
    def __init__(self,title,author,publisher):
        self.title = title
        self.auth = author
        self.pub = publisher
    def __str__(self):
            return self.title+" "+self.auth+" "+self.pub
b=Book("Percy","Rick Riordan","Penguin")
print(b)
```

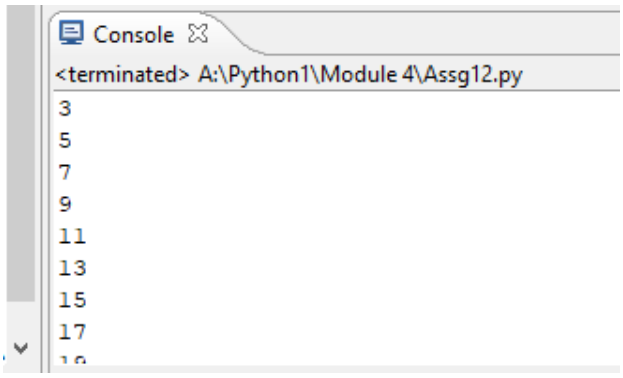
#



•A class Calculatorhas two methods
•getNextPrime() returns next primary number every time the function is called. For example, when first time invoked it returns 2 then 3,5,7… so on.
•isPrime(num) returns true if the argument “num” is prime.
•Create the class with above functions.
•Test your code with a loop that calls the getNextPrimeand prints first 50 prime numbers.
Hint :
1.Create a variable “self.lastprime” inside __init__ method to store last prime delivered.
2.First write isPrime function, test if it works fine then write logic for getNextPrime
3.Make call to isPrime from within getNextPrime function to avoid code duplication
#

```
class calculator:
    def __init__(self):
        pass
    @staticmethod
    def isprime(num):
        if num > 1:
            for i in range(2,num):
                if (num % i) == 0:
                    return False
                    break
                else:
                    return True
        else:
            return False
    def nextprime(self):
        for i in range(0,50):
            if(calculator.isprime(i)):
                print(i)
cal=calculator()
print(cal.nextprime())
```

#



Assignment 13

•Identify classes, and relationship among them in following retail scenario:

•All customers have Customer Id, Name, Telephone Number and Address
•The regular customer in addition is given discounts
•The privileged customer gets a membership card based on which gifts are given
#

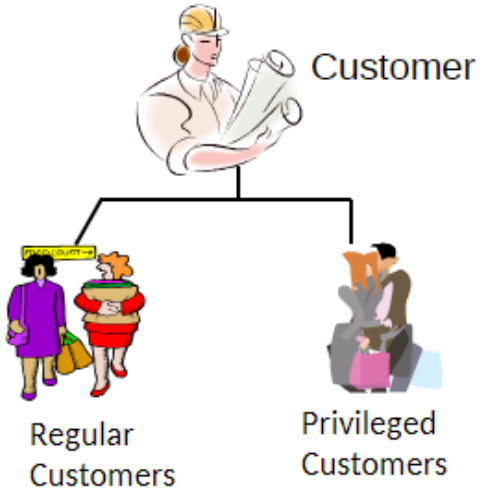
Classes-Customer, Regular Customer, Privileged customer

Customer-Customer ID(common), Name(common), Telephone no. (common) , Address(common)

Regular Customer- Discounts

Privileged Customer-Card, Gift

Hierarchical Inheritance



Assignment 14

•Requirement:
•A big box can contain small boxes.
Q: Write logic for function getCapacity(self,sBox) in BigBox that returns capacity(number of small boxes it can contain).
Formula for Capacity = BigBox volume / Small box volume.
•Test your code with following:

```
smallBox = Box(1,1,1)
bigBox = BigBox (4,4,4)
capacity = bigBox.getCapacity(smallBox)
print("capacity:",capacity)
```

```python
#
class Box:
    def getVolume(self):
        vol= self.length*self.breadth* self.height
        return vol
    def __init__(self, length, breadth,height):
        self.length = length
        self.breadth = breadth
        self.height = height
class BigBox(Box):
    def __init__(self, length, breadth,height):
        Box.__init__(self,length,breadth,height)
    def getCapacity(self, sBox):
        a=super().getVolume()
        b=sBox.getVolume()
        print("Volume of Big Box=",a)
        print("Volume of Small Box=",b)
        print("Capacity is",a/b)
smallbox=Box(1,1,1)
bigbox=BigBox(4,4,4)
capa=bigbox.getCapacity(smallbox)
#
```

```python
class Box:
    def getVolume(self):
        vol= self.length*self.breadth* self.height
        return vol
    def __init__(self, length, breadth,height):
        self.length = length
        self.breadth = breadth
        self.height = height
class BigBox(Box):
    def __init__(self, length, breadth,height):
        Box.__init__(self,length,breadth,height)
    def getCapacity(self, sBox):
        a=super().getVolume()
        b=sBox.getVolume()
        print("Volume of Big Box=",a)
        print("Volume of Small Box=",b)
```

Console

```
<terminated> A:\Python1\Module 4\fsffs.py
Volume of Big Box= 64
Volume of Small Box= 1
Capacity is 64.0
```

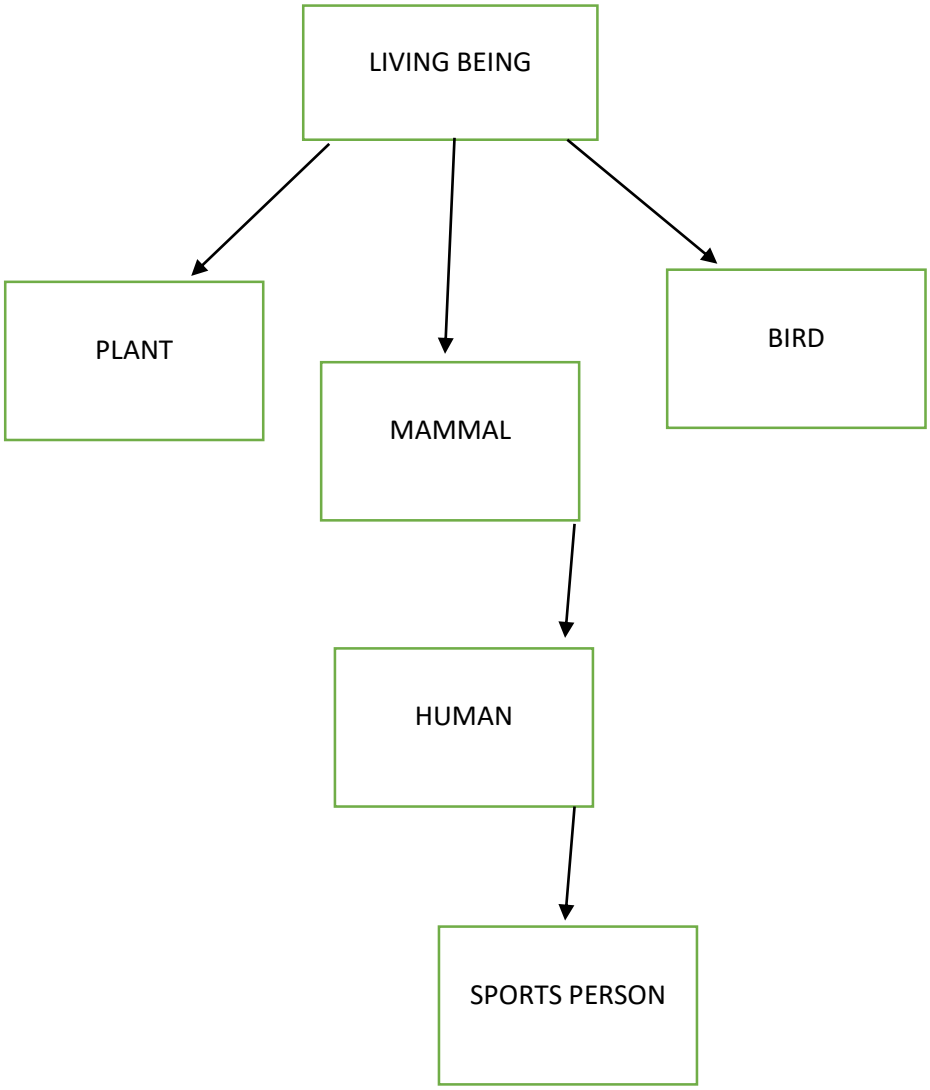
Assignment 15

•An animation film is in making. Following human actors are created in the animation film:(1) TennisPlayer (2) Professor (3) ShopKeeper (4)Carpenter
•This animation is coded as software. These actors are modeled as class entities.
•The film was so far silent, however a new method “talk” needs to be added to these entities.
Question : Which class(es) will be appropriate to add talk() method so that all actors can talk? Consider following before arriving at optimal solution:
•this functionality may undergo changes several times before final implementation.
•“sing” is next method to be added to these entities in near future.
•A Principal and a Pilot are two human actors to be added to the software model in next version.
#
All the Classes Will need to have the talk() method since none of these class have common attributes besides name to be inherited which while very inefficient is the only way they can talk as they also have different sentences too.
#

Assignment 16

Identify relationship between following entities. Depict the same with UML class diagram

•SportsPerson
•Human
•Mammal
•LivingBeing
•Plant
•Bird



Assignment 17

•Q1: Identify classes and their relationships; depict with UML class diagrams for following :

•External examiners and visiting faculties are visitors.

•All visitors are paid remuneration. The fees is calculated based on number of hours (end time – start time)

•Examiners examine.

•Clerks and regular faculties are the employees working on payroll of institution (salary).

•All employees get salary which is calculated based on pay days (calendar days - LOP(loss of pay))

•Clerks prepare balance sheet.

•A Faculty can be a Regular faculty from the same institute or a visiting faculty.

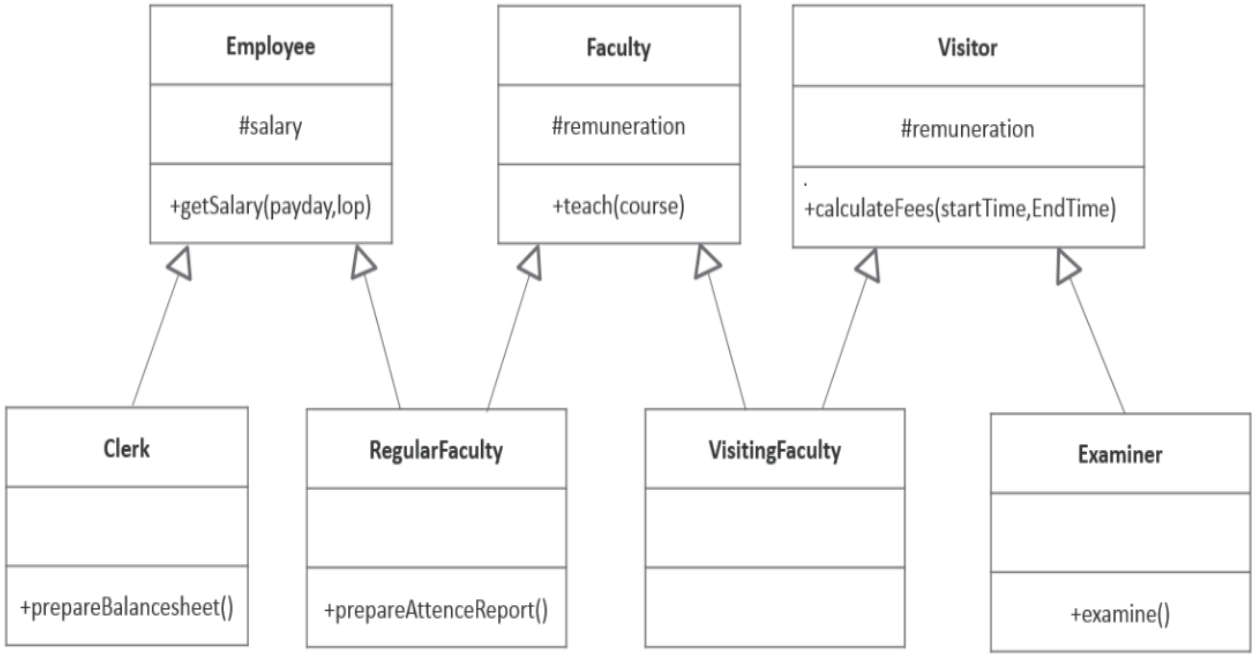
•Faculties teach one or more courses.

•Regular faculties prepare attendance report.

•Q2: Create classes in Python

•You may write "pass" for method bodies implementation, since no logic is provided

#



#

Assignment 18

In a retail outlet there are two modes of bill Payment
•Cash : Calculation includes VAT(15%)
Total Amount = Purchase amount + VAT
•Credit card: Calculation includes processing charge and VAT
Total Amount = Purchase amount + VAT(15%)+ Processing charge(2%)
The act of bill payment is same but the formula used for calculation of total amount differs as per the mode of payment.
Q: Can the Payment maker simply call a method and that method dynamically selects the formula for the total amount? Demonstrate this Polymorphic behavior with code.

#
Yes it is a valid mode of payment

```
from abc import ABC, abstractmethod
class Payment(ABC):
    VAT = 1.15
    @abstractmethod
    def totalAmount(self):
        pass
class CreditCardPayment(Payment):
    processingCharges = 1.02
    def getTotalAmount(self,purchaseAmt):
        amt = purchaseAmt * self.VAT #stmt1
        amt = amt * self.processingCharges
        return amt
class CashPayment(Payment):
    def getTotalAmount(self,purchaseAmt):
        return (purchaseAmt * self.VAT) #stmt2
class Bill:
    def __init__(self,purchaseAmount):
        self.__purchaseAmount = purchaseAmount
    def makePayment(self,mode):
        #Ensure that it is a valid mode of payment
        if (isinstance(mode, Payment)):
            #actual behavior is selected dynamically
            amount= mode.getTotalAmount(self.__purchaseAmount)
            print("Paid:",amount)
            #create a bill with
            #purchaseAmount=1000
bill = Bill(1000)
cc = CreditCardPayment()
bill.makePayment(cc)
cash = CashPayment()
```

```
bill.makePayment(cash)
#
```

Assignment 19

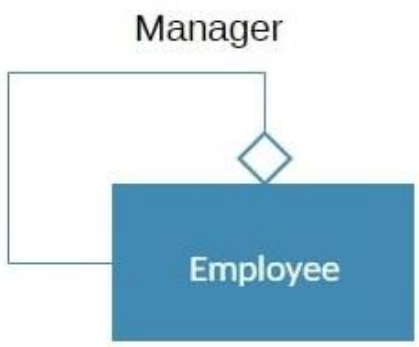
•What kind of relationship exists between car and its components given below:
•Engine
•Toolkit
•DVD Player
•DVD
#
Car is composed of Engine as integral part.
•Car <has-a: composition> Engine

DVD Player is a part of Car. However, it is not integral part (even if DVD player is removed, it can still be called as Car)
•Car <has-a: Aggregation> DVD Player

Car is NOT a Toolkit and DVD is not a part of Car, however car uses them
•Car <uses-a> Toolkit
•Car <uses-a> DVD

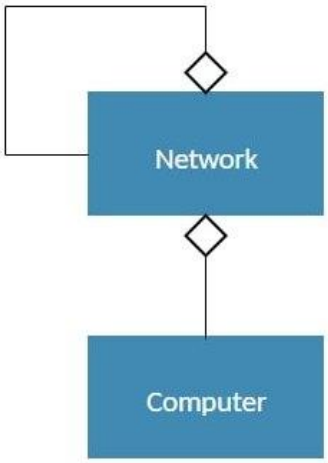
Assignment 20

In an organization for an employee to be a manager, he/she must be an employee of the same organization.
•Model this relationship in UML class diagram



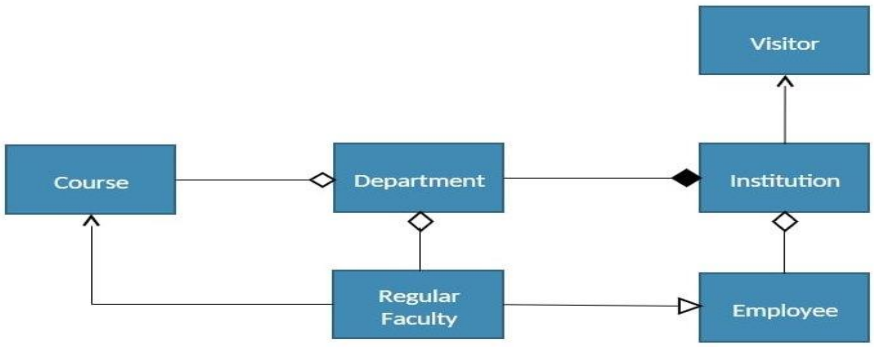
Assignment 21

Internet is network of networks and computers
Q - Model Internet using UML diagram



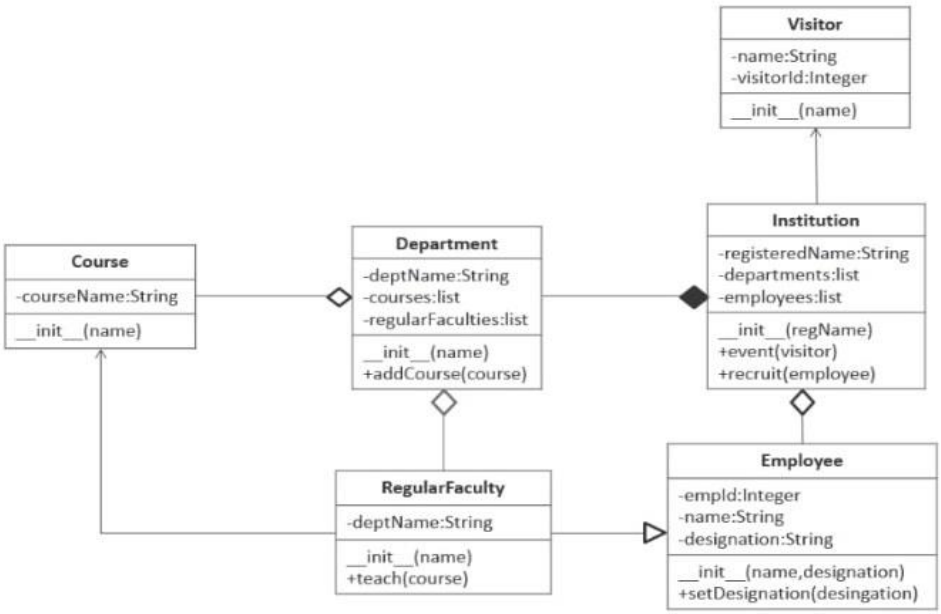
Assignment 22

Q1: Identify classes and their relationships; depict relationship with UML class diagrams for following :
•An institution has multiple departments and many employees.
•Every department offer multiple courses.
•All the regular faculties are the employees who belong to respective departments.
•Regular faculty teach one or more course.
•Visitors (Experts) are invited for a certain event.
Q2: Consider following details, improvise the class diagram and write Python code:
•The institution has a registered name. Every department has a unique name.
•Courses are identified by course name. A department can add new course anytime.
•Visitors are given an auto-generated visitor id at the time of their entrance.
•New employees can be recruited in the institution. Name & Designation are recorded and a unique id is generated for the employee
at the time of joining. The designation may change during service tenure.



Ans1-

Ans 2-



Assignment 23

Q:(1) Model following scenario using class diagram and (2) Create Python classes for the same.

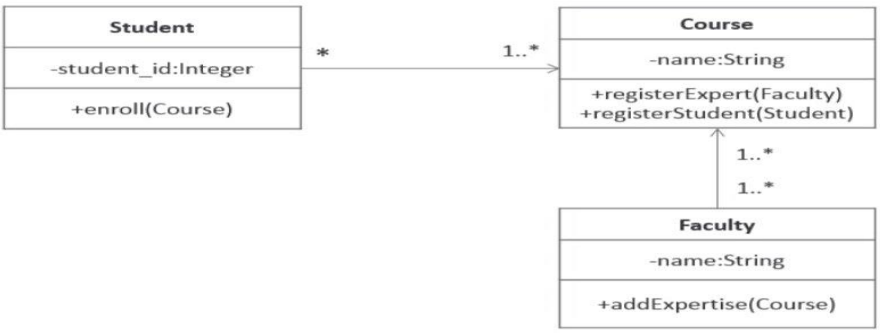
•The students can enroll for multiple courses.

•A course can be taught by one or more faculties.

•Every faculty has expertise to teach one or more courses. A faculty can acquire new expertise.

•Student is identified by student id, course is identified by course name and a faculty is identified by faculty
name.

step 1:

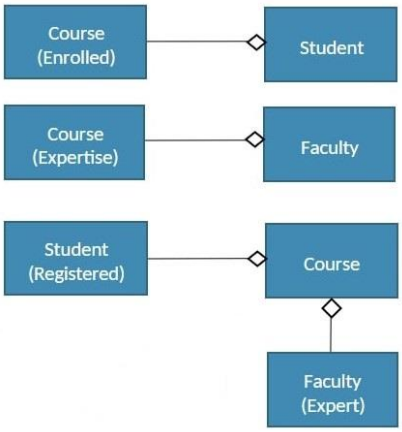


Step 2:

The entities(Student-Course and Course-Faculty) are associated with each other in a many-to-many relationship.

In such situations, we need to break the many-to-many relationship into one-to-many and many-to-one relationships as follows:

- Each student may enroll to multiple courses (one-to-many)
- Each faculty may have expertise to teach multiple courses (one-to-many)

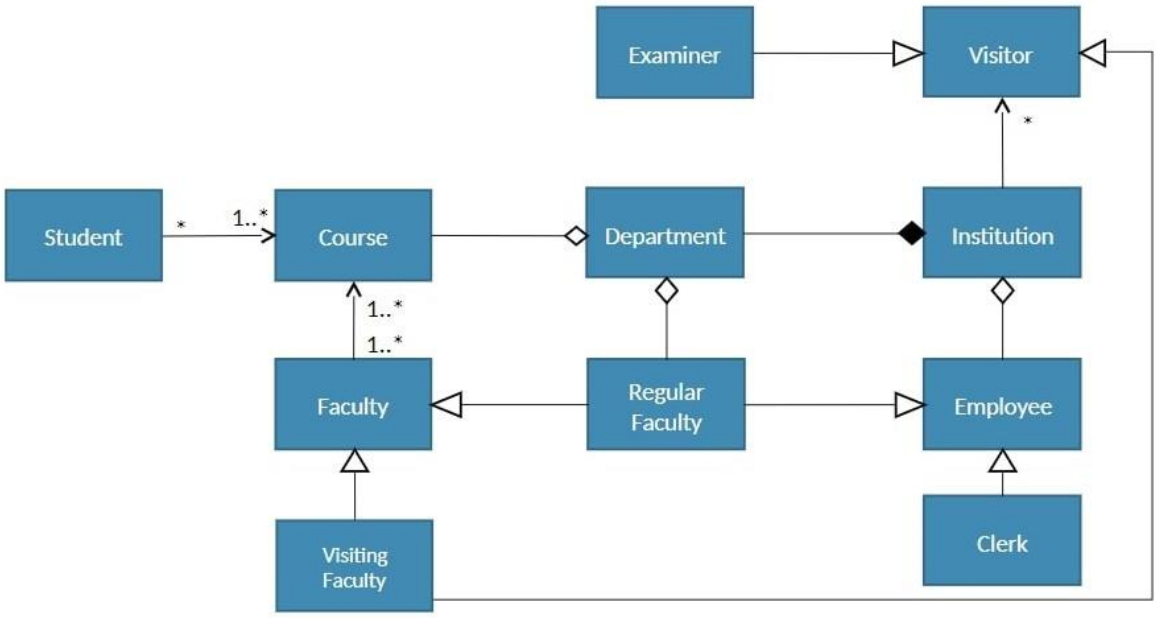



```
#
class Faculty:
    def addExpertise(self,course):
        self.coursesExpertise.append(course)
    def __init__(self,name):
        self.__name = name
        self.__coursesExpertise=[]
class Student:
    def enroll(self,course):
        self.enrolledCourses.append(course)
    def __init__(self,student_id):
        self.__id = student_id
        self.__enrolledCourses=[]
class Course:
    def registerStudent(self,student):
        self.__registeredStudents.append(student)
    def registerExpert(self,faculty):
        self.expertFaculties.append(faculty)
    def __init__(self,name):
        self.__name=name
        self.__registeredStudents=[]
        self.__expertFaculties=[]
#
```

Assignment 24

Identify entities/classes and their relationships; depict with UML class diagrams.
•An institution has multiple departments and many employees.
•Every department offer multiple courses.
•Students can enroll for multiple courses.
•Every faculty has expertise to teach one or more courses.
•All regular faculties of the institution belong to respective departments
•A course can be taught by one or more faculties
•External examiners and visiting faculties are the visitors who are invited for a certain event.
•Clerks, regular faculties are employees working on payroll of institution.
•A Faculty can be a Regular faculty from the same institute or a visiting faculty.

Ans-



Assignment 25

Write implementation logic for methods created in Assignment 22.

Test the same for following expected input/output:

1.d1 = Department("Computer") #create “computer” department
2.e1 = Employee("John", "Clerk") #create an employee “John” with designation “Clerk”
3.# Employee.__str__() should return:"<employee id>:<name>(<designation>)”
4.print(e1) #expected output=> “1:John(Clerk)”
5.#create regular faculty “Jack”, a professor in “computer” department.
6.e2 = RegularFaculty("Jack",d1,"Professor")

7.#RegularFaculty.__str__() should return:
“<employeeid>:<name>(<designation>)[<department>]”
8.print(e2) #expected output=> “2:Jack(Professor)[Computer]”
9.v1 = Visitor("Bill Gates") #create a visitor with name "Bill Gates“.
11.#create an Institute with registered name "Institute of Technology“
12.inst = Institution("Institute of Technology")
13.#Institution.addDepartment(department)method should print: “<department name> department is added”
14.inst.addDepartment(d1) # expected output=> “Computer department is added"
15.#Institution.event(visitor) method should print: "Visitor for the event: <visitor name>“
16.inst.event(v1) #expected output => “Visitor for the event: Bill Gates”
17.#Institution.recruit(employee)method should print=> "recruited: <employee designation>“
18.inst.recruit(e1) # expected output => "recruited: Clerk“
19.inst.recruit(e2) # expected output => "recruited: Professor“
20.c = Course(“Database") #create a course for course name “Database”
21.#RegularFaculty.teach(Course) method should print “<faculty name> teaches <course name>”
22.e2.teach(c) # expected output => ”Jack teaches Database”

Assignment 26

Create function asteriskChecker(myString) such that the method raises user defined InvalidStringexception if an asterisk (*) is found. The #statement1 should print “Found Asterisk” if the input string contains *.

```
Mymessage="abcde*fz"
try:
    asteriskChecker(mymessage)
except InvalidString as e :
    print(e) #statement1
else:
    print("String has no Asterisk ")

#
class InvalidString(Exception):
    def __init__(self):
        Exception.__init__(self,"Found Asterisk")
    def asteriskChecker(myString):
        for i in myString:
            if(i=="*"):
```

```
raise InvalidString() #statement1
```

Note: alternately we could have added error message at #statement1 as –

```
raise InvalidString("Found Asterisk")
```

in this case init method is not needed in InvalidString class

#

Assignment 27

Write a function that

•opens a file

•Reads and prints the existing content.

•Write a statement "hello" in the file.

•After the file is successfully opened, if the file read/ write operation results in an exception for any reason (like file is read only and user is not authorized to write, some OS error etc.), then ensure that the file is closed before the exit from the function

#

```
file = open("testfile.txt","w")
file.write('Hello World')
file.write('This is our new text file')
file.write("and this is another line.")
file.write("Why? Because we can.")
file.close()
file1 = open("testfile.text", "r")
print(file1.read())
file1 = open("testfile.txt", "r")
print(file1.read(5))
file1 = open("testfile.txt", "r")
print(file1.readline())
file1 = open("testfile.txt", "r")
print(file1.readline(3))
file1 = open("testfile.txt", "r")
print(file1.readlines())
print(file1.read())
file1.close()
```

#

Assignment 28

•Take email id, mobile number and age as inputs from user

•Validate each and raise user defined exceptions accordingly

Note:-

Email id :

•there must be only one @

•At least one "."

-Mobile number must be 10 digits and it can start with + symbol

-Age must be a positive number less than 101
#

```
// code contains regular expression for contact number and ema
il address in python
str='abc@example.com'
match=re.search(r'\w+@\w+',str) #return abc@example.com
num=555-555-555
match_num=re.search(r'^(\d{3}--\d{3}--\d{4}),num) #return 555-
555-5555
```

```
def inputNumber(message):
  while True:
    try:
       userInput = int(input(message))
    except ValueError:
       print("Not an integer! Try again.")
       continue
    else:
       return userInput
       break
     pyt = inputNumber("Enter your email and mobile number")
#
```

Assignment 29

Requirement from a function getDiscount(age) is given bellow. This function is supposed to return the percentage discount figure for the given input age value of customer in years.

•If the age of customer is less than 60, No discount..
•If age is 60 to 70 years then the discount should be 15%.
•Discount should be 30% for age greater than and equal to 70 years.
Complete the following table for boundary value test case.

Test case	*Description*	*Expected Discount*
#1	*Customer age <60*	*0*
#2	*Customer age =60*	*15*
#3	*Customer age>60 &<65*	*15*
#4	*Customer age>65&<70*	*15*
#5	*Customer age=70*	*30*
#6	*Customer age>70*	*30*

Assignment 30

•A programmer has written this code for getDiscount(age) function for the requirement given in previous assignment:
1.Write and Execute test cases. Use the below given table to note the details.
2.“Actual discount”: Write actual discount you get after execution of every test case.
3.Compare the actual discount with expected discount. If they match, write “pass” else “fail” for every test case in Result column
#

```
def getDiscount(age):
    discount = 0
    if age > 60 and age < 70:
        discount = 15
    elif age > 70:
        discount = 30
    return discount
myAge = int(input(“Enter Age:"))
myDiscount = getDiscount(myAge)
print("Discount percent=",myDiscount)
```

#

Test Case	*Description*	*Expected Discount*	*Actual Discount*	*Result*
#1	*Age=55*	*0*	*0*	*Pass*
#2	*Age=60*	*15*	*0*	*Fail*
#3	*Age=64*	*15*	*15*	*Pass*
#4	*Age=68*	*15*	*15*	*Pass*
#5	*Age=70*	*30*	*0*	*Fail*
#6	*Age=75*	*30*	*30*	*Pass*

Assignment 31

•A programmer has written this code for function getDiscount(age,gender), for the requirement given below :
•Non Senior (age<60) female = 15% discount;
•Senior citizen: Female=25%, Male=20%
1.Write and Execute test cases which covers all the paths of execution. Use the below given table to note the details.
2.Compare the actual discount with expected discount. If they match, write “pass” else “fail” for every test case in Result column
#

```
def getDiscount(age,gender):
```

```
    discount = 0
    if age >= 60:
        if gender == 'F':
            discount = 25
        discount = 20
    elif gender == 'F':
        discount = 15
    return discount
age = int(input("Enter age:"))
gender = input("Enter Gender as M or F:")
discount = getDiscount(age, gender)
print("discount",discount)
#
```

Test Case	*Description*	*Expected Discount*	*Actual Discount*	*Result*
#1	*Age=60 Gender="M"*	*20*	*20*	*PASS*
#2	*Age=60 Gender="F"*	*25*	*20*	*FAIL*
#3	*Age=40 Gender="M"*	*0*	*0*	*PASS*
#4	*Age=40 Gender="F"*	*15*	*15*	*PASS*

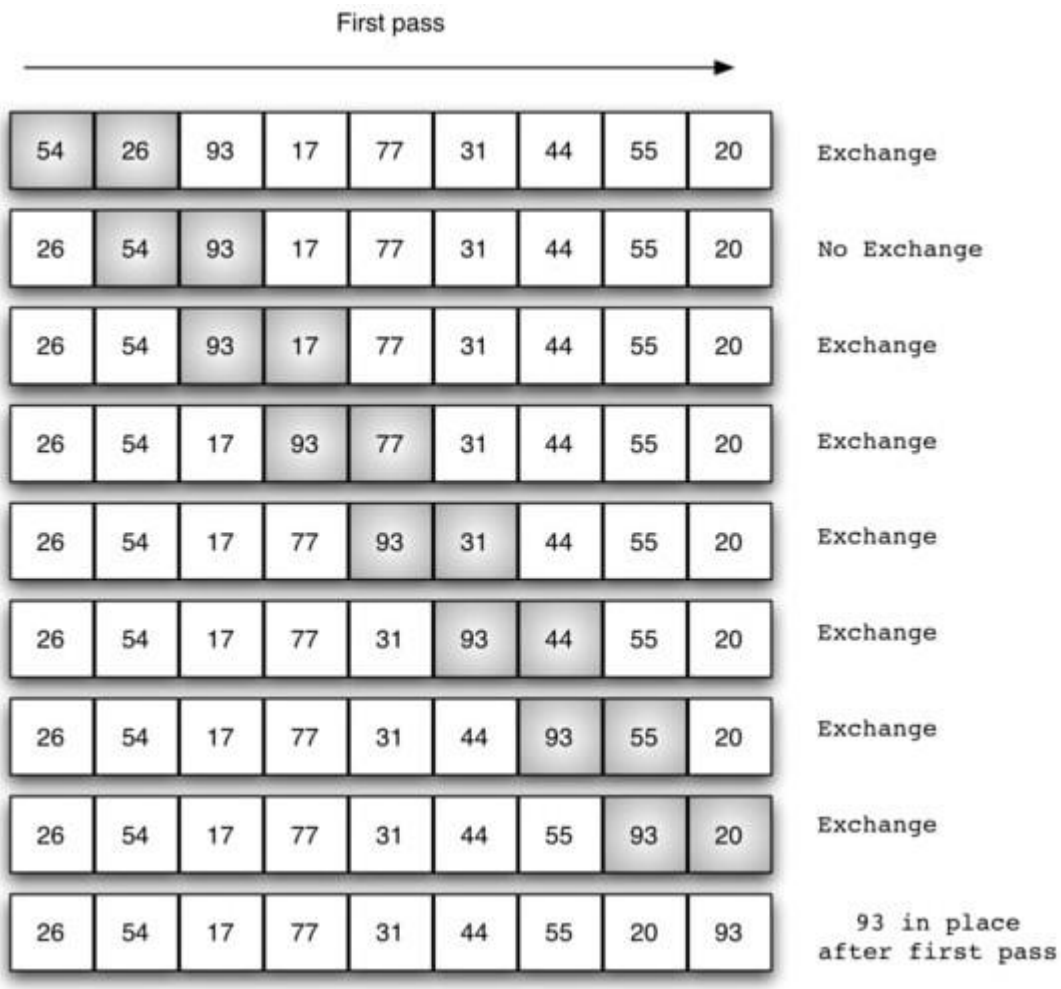
Assignment 32

Q: Write and Execute test cases for following algorithm

```
def bubbleSort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp
alist = [54,26,93,17,77,31,44,55,20]
bubbleSort(alist)
print(alist)

#
import unittest
from random import random
from Test3 import bubbleSort
class Test(unittest.TestCase):
    def test_bubble_sort(self):
        seq = [random() for _ in Test3.alist]
        sorted_seq = sorted(seq)
        self.assertEqual(bubbleSort(seq), sorted_seq)
```

#



Pass	Comparisons
1	n−1n−1
2	n−2n−2
3	n−3n−3
...	...
n−1n−1	1