**FP5.0 Module-1 Assignment**

**Batch Name:**

Infosys FP5.0 Summer 2018

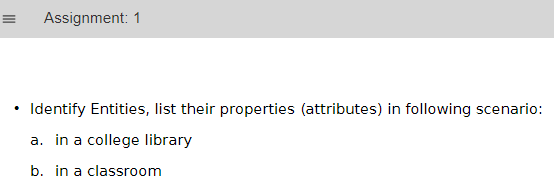
**Enrollment Number:** R171217041

**SAPID:** 500060720

**Name:** Nishkarsh Raj Khare

**Sem:** Semester-III

**Branch:** CSE-DevOps-Xebia

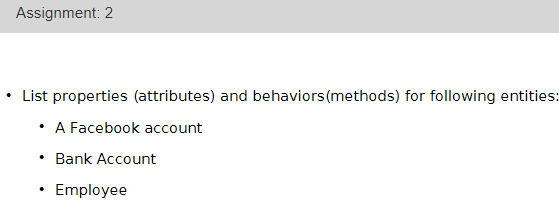
**Assignment-1**

**Entities in college library:**

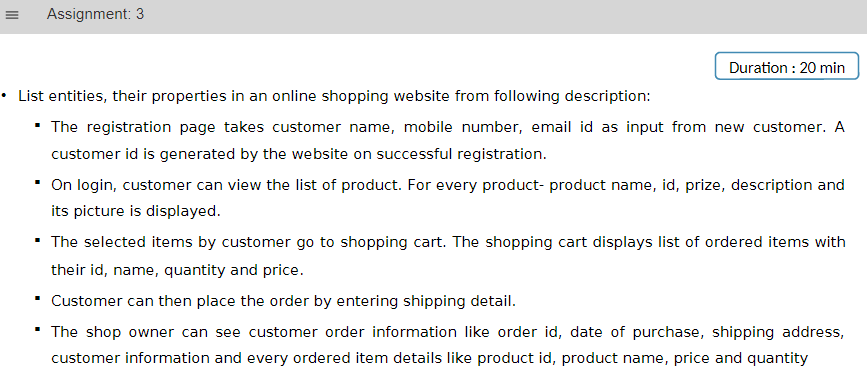
1. College Library:  
   1) List of Books
2. Librarian:  
   1) Name  
   2) EmployeeID
3. Book:  
   1) BookID  
   2) Title   
   3) Author  
   4) Publisher
4. Issued Book:  
   1) Date of Issue  
   2) BookID  
   3) Publisher

**Entities in a Classroom:**

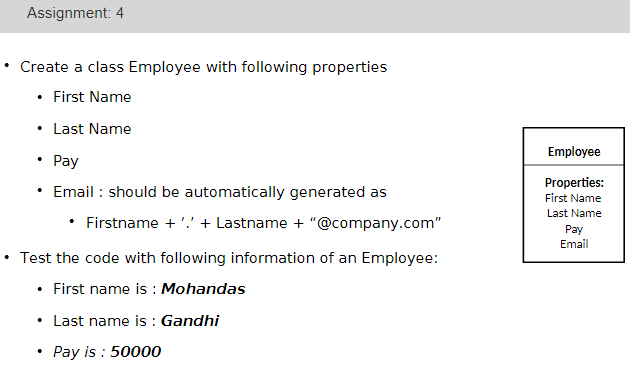
1. Student:  
   1) Roll No  
   2) Name
2. Classroom:  
   1) Whiteboard  
   2) Desk  
   3) Bench
3. Teacher:  
   1) Name  
   2) Department  
   3) Expertise

**Assignment-2**

1. **Facebook Account:**  
   Attributes:  
   1) Name  
   2) Age  
   3) Nickname  
   4) Friendlist  
   5) Notifications  
   6) Wall  
   7) Messages etc.  
   Methods:  
   1) Makefriend()  
   2) VisitWall()  
   3) Message()
2. Bank Account:  
   Attributes:  
   1) Name  
   2) CVV  
   3) Account Number  
   4) Pin  
   5) Valid Through  
   6) Valid from  
   Methods:  
   1) Deposit()  
   2) Withdraw()  
   3) Transfercurrency()
3. Employee:  
   Attributes:  
   1) Employee Name  
   2) Employee ID  
   3) Gender  
   4) Age  
   5) Nationality  
   6) Department  
   7) Project  
   8) Salary etc.  
   Methods:  
   1) GetSalary()  
   2) CheckWork() etc.

**Assignment-3**

* Customer: id, name, mobile number, email id
* Product: Id, Name, description, price, image path
* Shopping cart: list of ordered\_item
* Ordered\_item: product id, product name, price, quantity
* Order:  
  1) order id  
  2) Date of purchase  
  3) Shipping Address  
  4) Customer ID or   
  5) List of Ordered\_item

**Assignment-4**

**Code:**

#Assignment 4

class Employee:

def \_\_init\_\_(self,fn,ln,p): #doubleunderscoresused

self.first\_name = fn

self.Last\_name = ln

self.Pay = p

self.Email = fn + '.' + ln + '@company.com' #Inside constructors never use instance variables on RHS

def show\_details(self):

print("First name of the employee is:",self.first\_name)

print("Last name of the employee is:",self.Last\_name)

print("Pay of the employee is:",self.Pay)

print("Email of the employee is:",self.Email)

print("Enter details of Employee!!!")

fname = input("Enter first name of the employee: ")

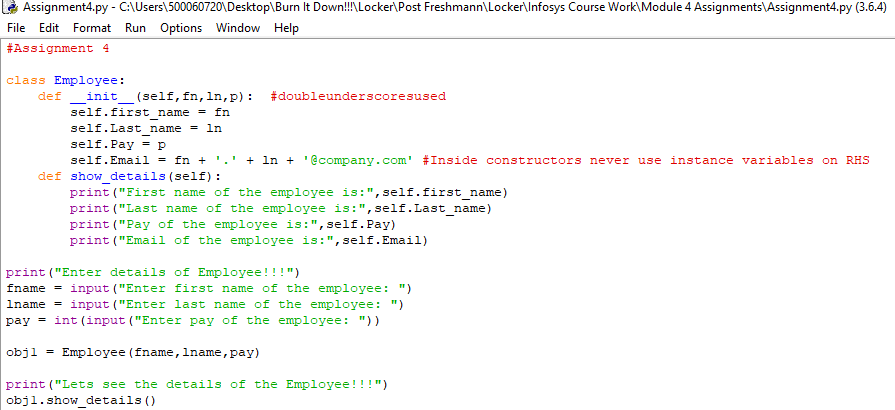
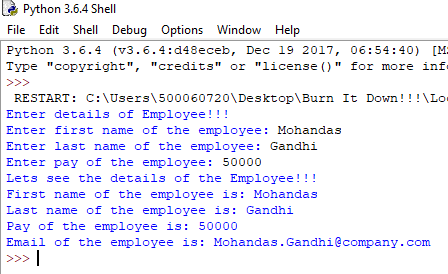
lname = input("Enter last name of the employee: ")

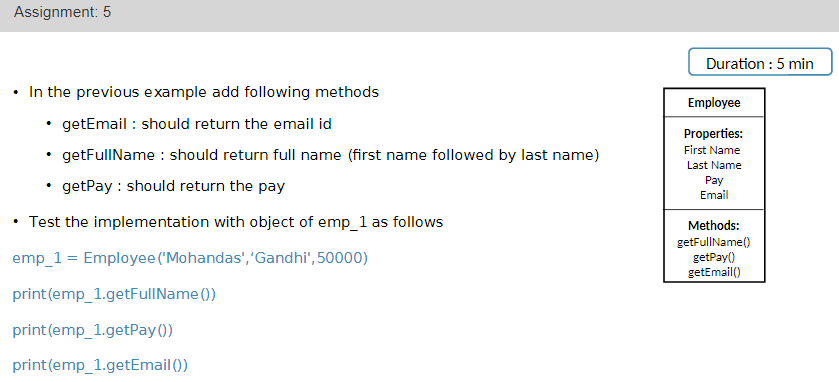
pay = int(input("Enter pay of the employee: "))

obj1 = Employee(fname,lname,pay)

print("Lets see the details of the Employee!!!")

obj1.show\_details()

**Code Screenshot:  
Output Screenshot:**

**Assignment-5**

**Code:**

#Assignment 5

class Employee:

def \_\_init\_\_(self,fn,ln,p): #doubleunderscoresused

self.first\_name = fn

self.Last\_name = ln

self.Pay = p

self.Email = fn + '.' + ln + '@company.com' #Inside constructors never use instance variables on RHS

def getFullname(self):

print("Full name of the employee is:",self.first\_name+' '+self.Last\_name)

def getEmail(self):

print("Email of the employee is:",self.Email)

def getPay(self):

print("Pay of the employee is:",self.Pay)

print("Enter details of Employee!!!")

fname = input("Enter first name of the employee: ")

lname = input("Enter last name of the employee: ")

pay = int(input("Enter pay of the employee: "))

obj1 = Employee(fname,lname,pay)

#Full name of the Employee

print("Lets see the full name!!!")

obj1.getFullname()

#Email of the employee

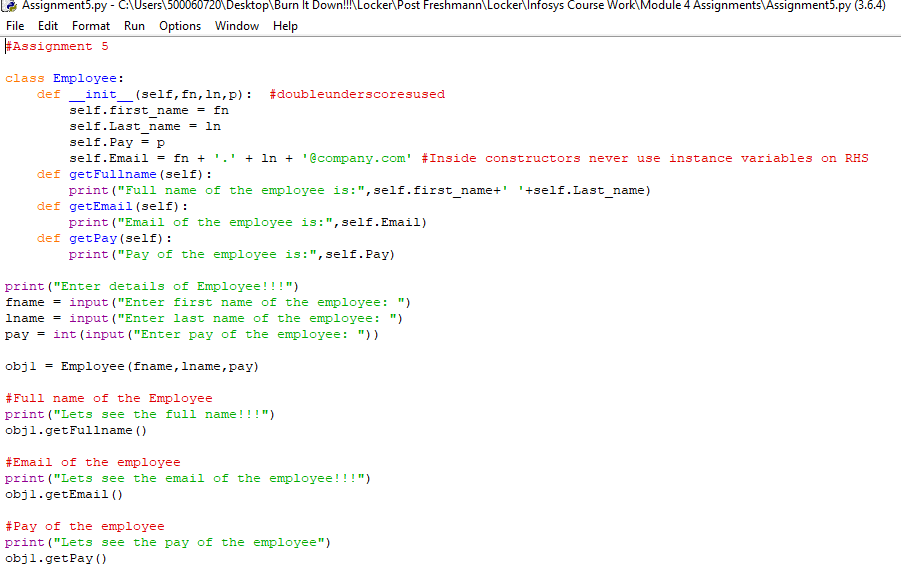
print("Lets see the email of the employee!!!")

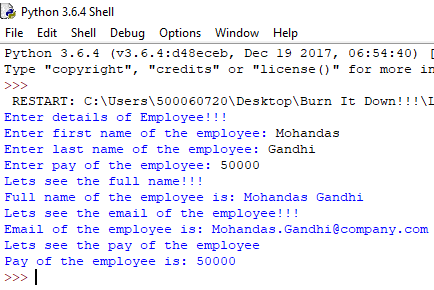
obj1.getEmail()

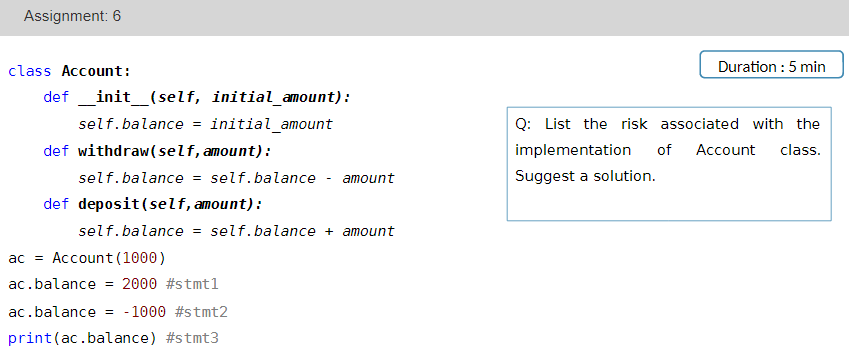
#Pay of the employee

print("Lets see the pay of the employee")

obj1.getPay()

**Code Screenshot:**

**Output Screenshot:**

**Assignment-6**

**Code:**

class Account:

def \_\_init\_\_(self, initial\_amount):

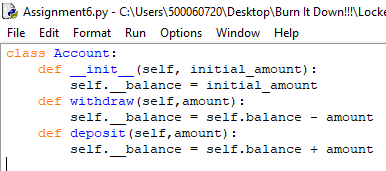
self.\_\_balance = initial\_amount

def withdraw(self,amount):

self.\_\_balance = self.balance - amount

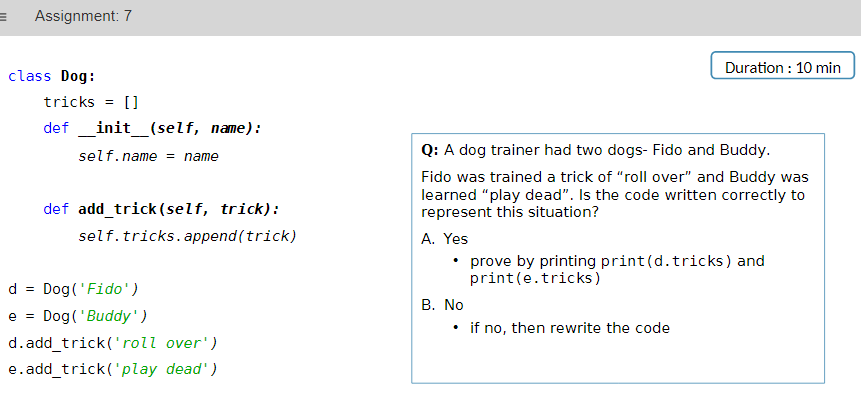
def deposit(self,amount):

self.\_\_balance = self.balance + amount

**Code Screenshot:**

**Solution:**

* The balance can be set to very high/low value accidently.
* The balance can be accessed or changed by any user of the class
* The balance can be set to non-permitted value
* Best solution is to create a private variable (\_\_balance):  
    
  def \_\_init\_\_(self, initial\_amount):  
   self.\_\_balance = initial\_amount

**Assignment-7**

**Code:**

#Assignment 7

'''

class Dog:

tricks = []

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

self.name = name

def add\_trick(self, trick):

self.tricks.append(trick)

d = Dog('Fido')

e = Dog('Buddy')

d.add\_trick('roll over')

e.add\_trick('play dead')

'''

''' The Above code is wrongly written!! Because a class variables store common values and thus it will be difficult to distinguish the UNIQUE features of both dogs '''

#Corrected Code

class Dog:

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

self.name = name

self.tricks = tricks

def print\_details(self):

print("Name of the dog is:",self.name)

print("Tricks dog does are:",self.tricks)

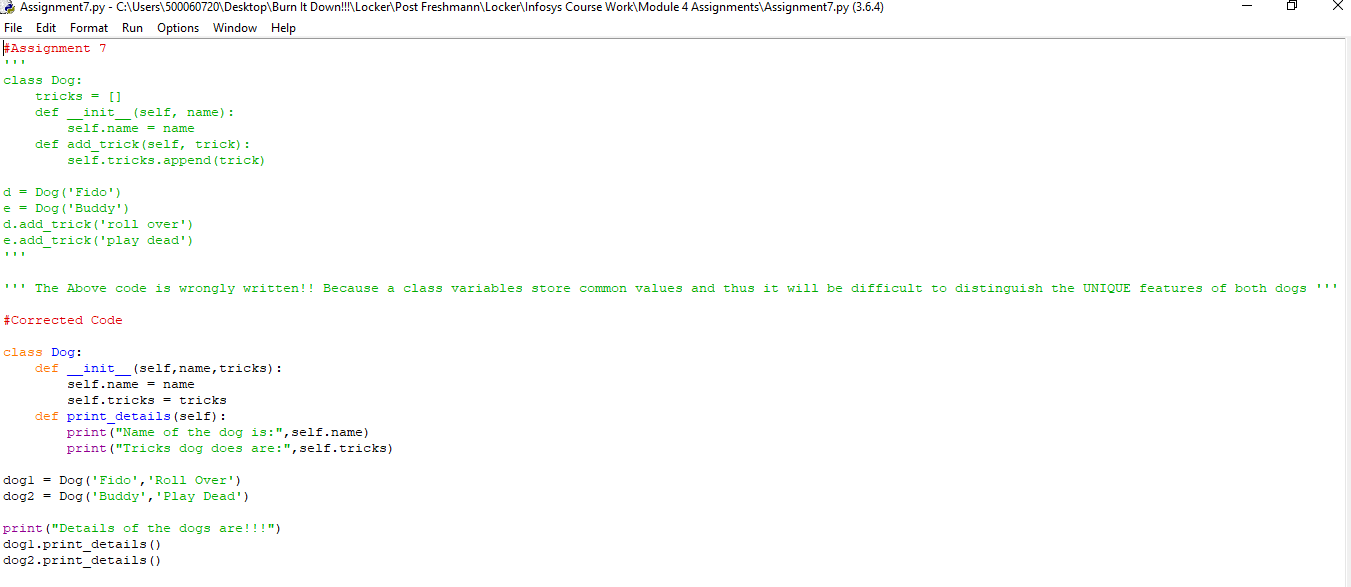
dog1 = Dog('Fido','Roll Over')

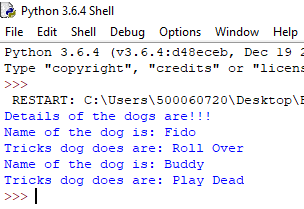
dog2 = Dog('Buddy','Play Dead')

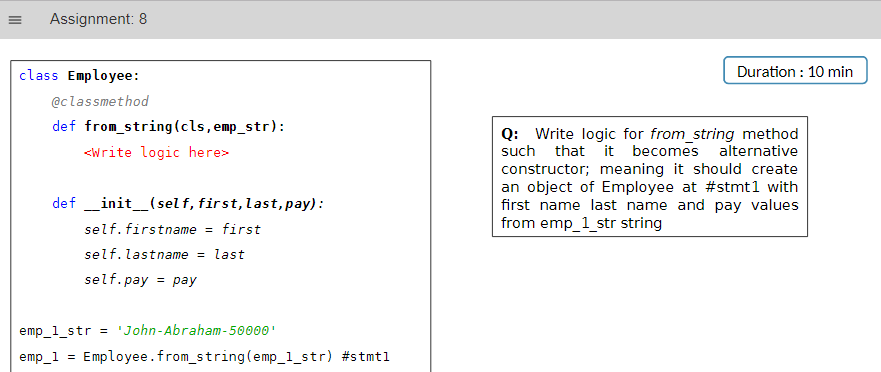
print("Details of the dogs are!!!")

dog1.print\_details()

dog2.print\_details()

**Code Screenshot:**

****  
**Output Screenshot:**

**Assignment-8**

**Code:**

#Assignment 8

class Employee:

@classmethod

def from\_string(cls,emp\_str):

#Coded from here

fn , ln , p = emp\_str.split('-')

return cls(fn,ln,p)

def \_\_init\_\_(self, first,last,pay):

self.firstname = first

self.lastname = last

self.pay = pay

def showdetails(self):

print("Firstname of the employee is:",self.firstname)

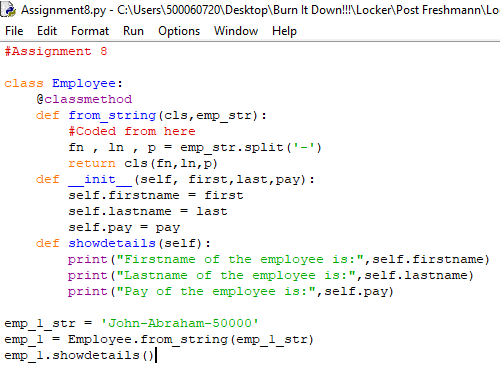
print("Lastname of the employee is:",self.lastname)

print("Pay of the employee is:",self.pay)

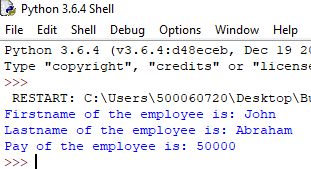
emp\_1\_str = 'John-Abraham-50000'

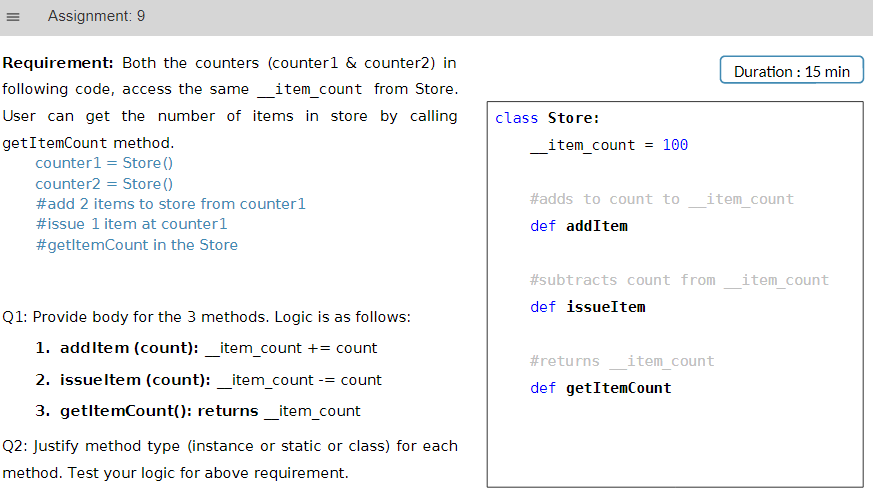
emp\_1 = Employee.from\_string(emp\_1\_str)

emp\_1.showdetails()

**Code Screenshot:**

**Output Screenshot:**

****

**Assignment-9**

**Code:**

#Assignment 9

class Store:

\_\_item\_count = 100 #private class variable

#Adds to count to \_\_item\_count

'''Instance method because transactions are done from individual counters'''

def addItem(self,count):

self.\_\_item\_count += count

#Subtracts count from \_\_item\_count

'''Instance method because transactions are done from individual counters '''

def issueItem(self,count):

self.\_\_item\_count -= count

#Returns \_\_item\_count

'''Class method because total amount is of the entire store '''

def getItemCount(self):

return self.\_\_item\_count #cannot directly return private member

counter1 = Store()

counter2 = Store()

#add 2 items from counter 1

counter1.addItem(2)

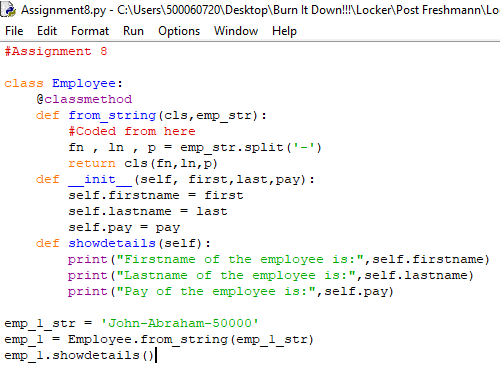
#issue 1 items from counter 1

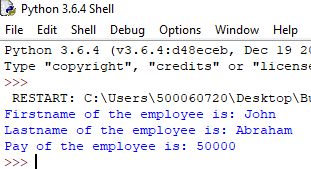
counter1.issueItem(1)

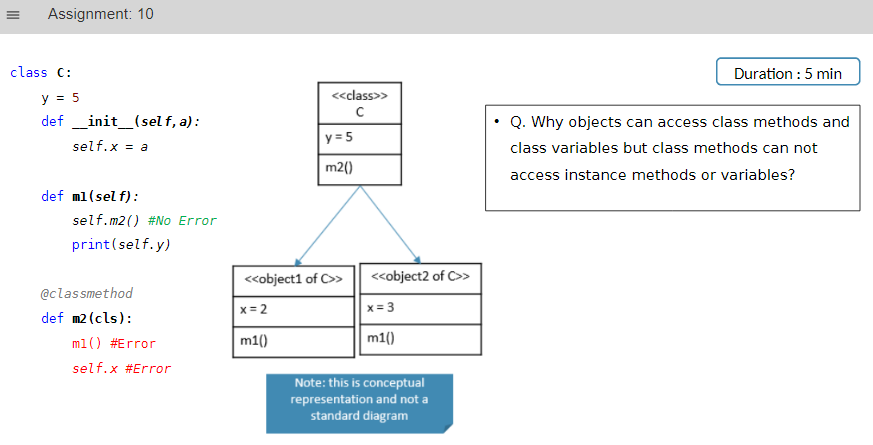
#get item count in the store

count = counter1.getItemCount()

print(count)

**Code Screenshot:**

**Output Screenshot:**

**Assignment-10**

**Answer:**

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

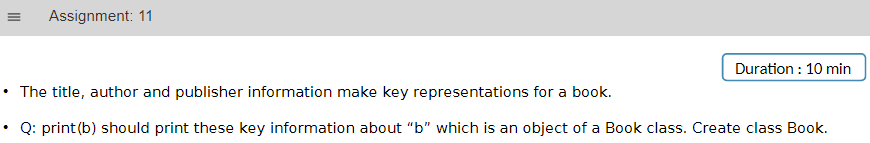
information about objects created.

•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**

**Code:**

#Assignment 11

class Book:

def \_\_init\_\_(self,title,author,publisher):

self.title = title

self.author = author

self.publisher = publisher

def print\_details(b):

print("Book name:",b.title)

print("Author:",b.author)

print("Publisher:",b.publisher)

book1 = Book('Artificial Intelligence','Nishkarsh Raj','UPES')

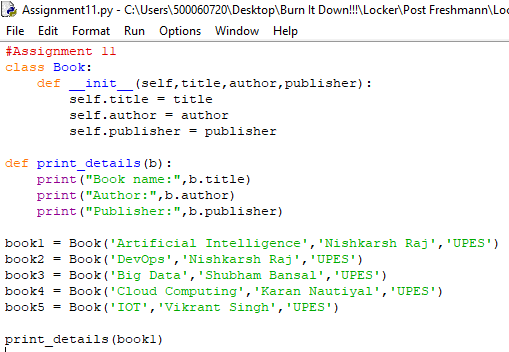
book2 = Book('DevOps','Nishkarsh Raj','UPES')

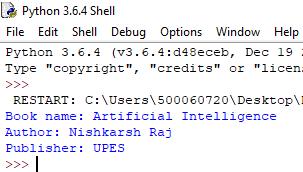
book3 = Book('Big Data','Shubham Bansal','UPES')

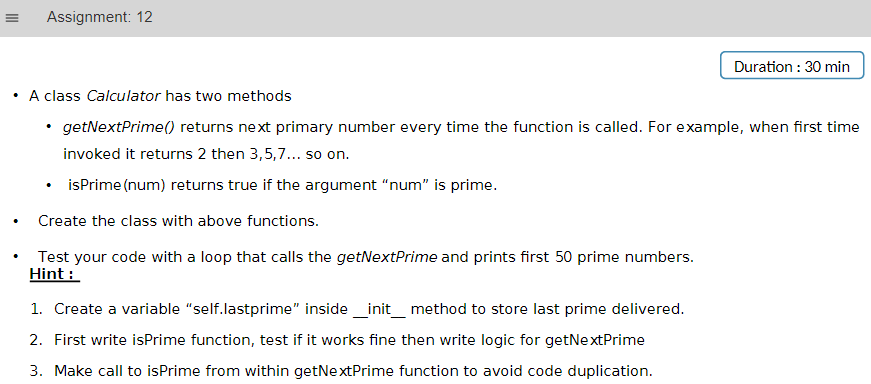
book4 = Book('Cloud Computing','Karan Nautiyal','UPES')

book5 = Book('IOT','Vikrant Singh','UPES')

print\_details(book1)

**Code Screenshot:**

**Output Screenshot:**

**Assignment-12**

**Code:**

#Assignment 12

class Calculator:

currentnum = 1 #for start of program

#isPrime Function

@staticmethod

def isPrime(num):

count = 0

for k in range (1,num+1):

if num % k == 0:

count = count + 1

else:

pass

k = k+1

if count == 2:

return True

else:

return False

#getNextPrime()

@staticmethod

def getNextPrime():

Calculator.currentnum += 1

while(Calculator.isPrime(Calculator.currentnum)!=True):

Calculator.currentnum +=1

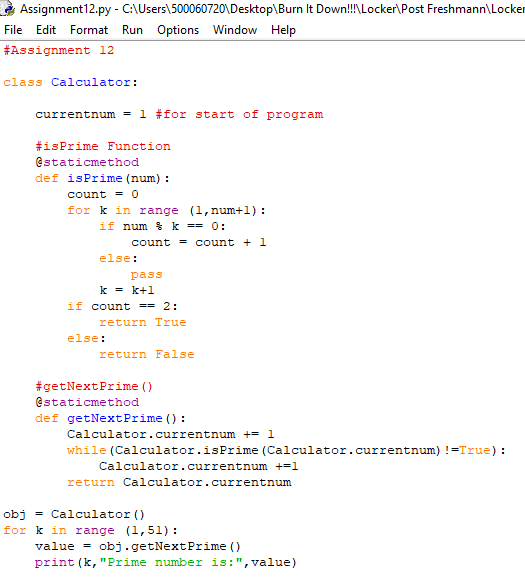
return Calculator.currentnum

obj = Calculator()

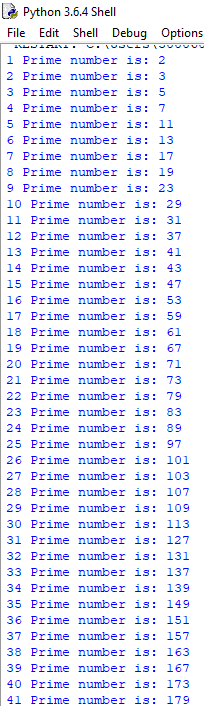
for k in range (1,51):

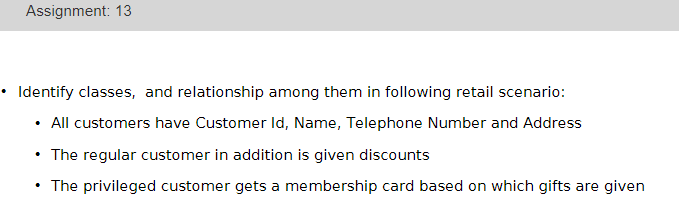
value = obj.getNextPrime()

print(k,"Prime number is:",value)

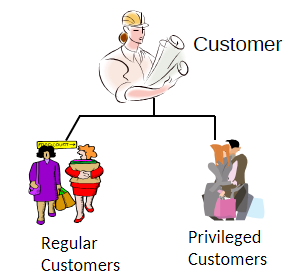
**Code Screenshot:**

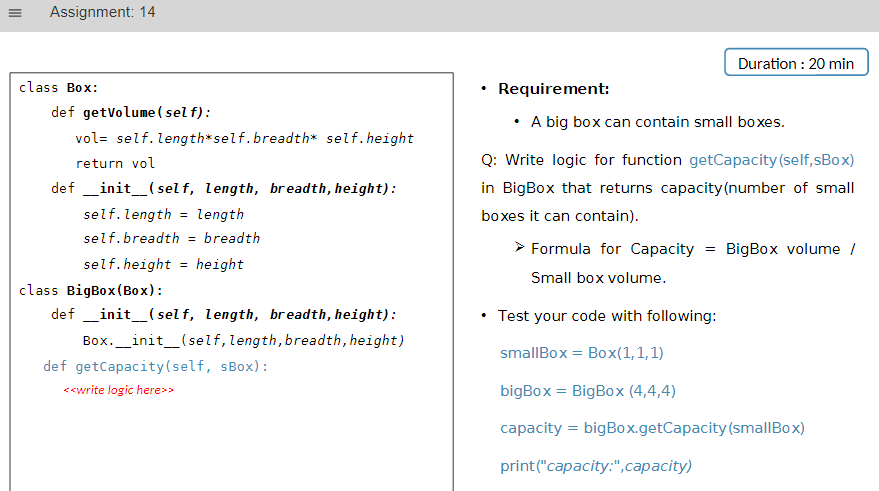
**Output Screenshot:**

****

**Assignment-13**

**Solution:**

****

**Assignment-14**

**Code:**

#Assignment 14

''' Parent Class Box '''

class Box:

def getVolume(self):

vol = self.length \* self.breadth \* self.height

return vol

def \_\_init\_\_(self,length,breadth,height):

self.length = length

self.height = height

self.breadth = breadth

''' Child Class BigBox '''

class BigBox(Box):

def \_\_init\_\_(self,length,breadth,height):

Box.\_\_init\_\_(self,length,breadth,height)

def getCapacity(self,sBox):

bigvol = self.getVolume()

smallvol = sBox.getVolume()

return int(bigvol/smallvol)

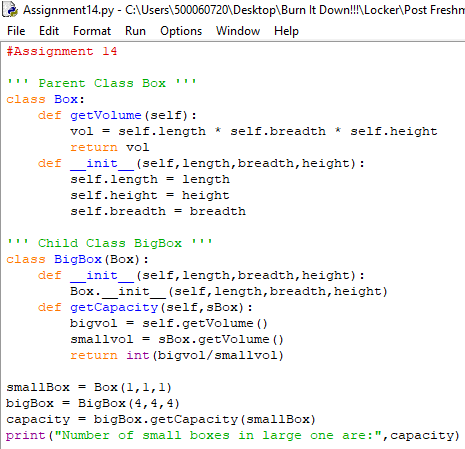
smallBox = Box(1,1,1)

bigBox = BigBox(4,4,4)

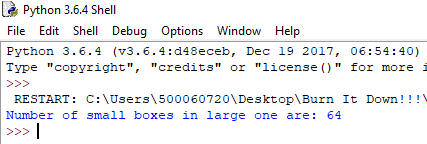
capacity = bigBox.getCapacity(smallBox)

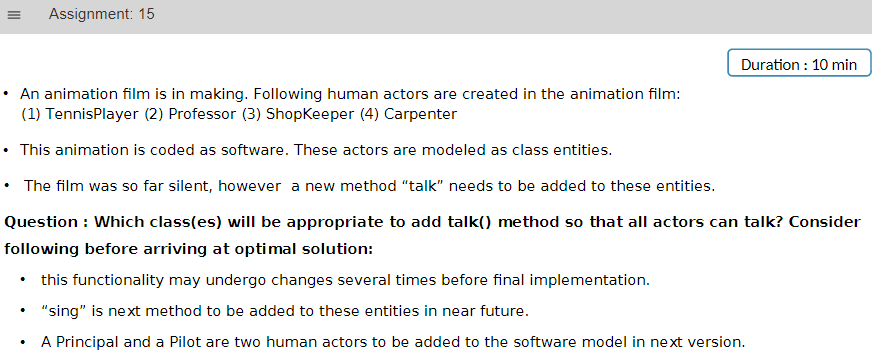
print("Number of small boxes in large one are:",capacity)

**Code Screenshot:**

****

**Output Screenshot:**

****

**Assignment-15**

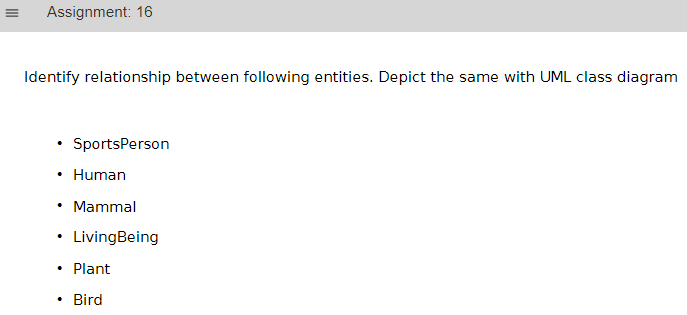
**Solution:**

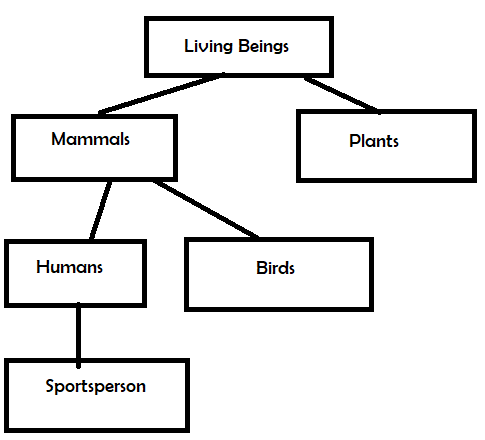
Since, all the actors follow general action of talking and all humans can sing.  
We must create a general class “Actors” with methods:

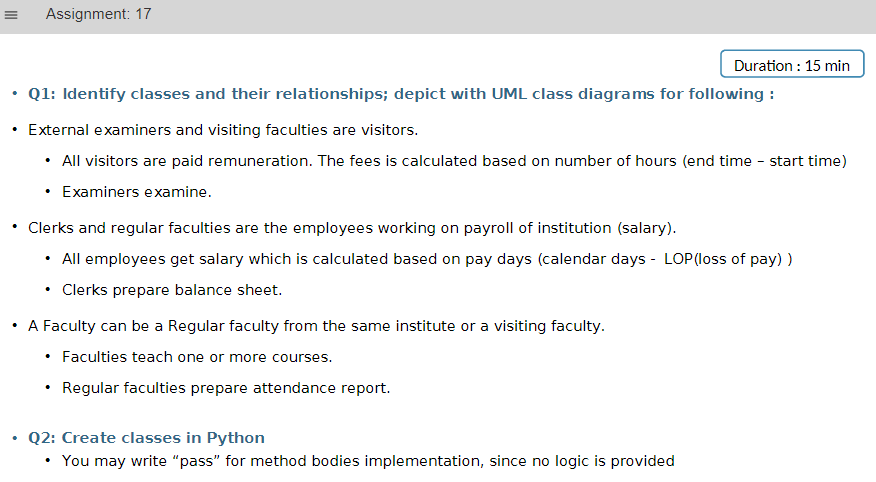
1. Sing()
2. Talk()

Thus, all the actors, i.e. Tennis Player, Professor etc. must derive inheritance relationship with actor.

Finally, two more child class of Principal and Pilot can be added anytime that derive from actors.

**Assignment-16**

**UML Diagram:**

**Assignment-17**

**Code:**

#Assignment 17

''' Parent Class 1: Employee '''

class Employee:

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

self.\_salary = salary #Protected Data Member

def getSalary(payday,lop):

pass

''' Parent Class 2: Faculty '''

class Faculty:

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

self.\_remuneration = rem #Protected Data Member

def teach(course):

pass

''' Parent Class 3: Visitor '''

class Visitor:

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

self.remuneration = rem #Protected Data Member

def calculateFees(startTime,EndTime):

pass

''' Child class 1: Clerk -> Inherits Employee '''

class Clerk(Employee):

def prepareBalancesheet():

pass

''' Child class 2: RegularFaculty -> Inherits Employee and Faculty '''

class RegularFaculty(Employee,Faculty):

def prepareBalancesheet():

pass

''' Child class 3: VisitingFaculty '''

class VisitingFaculty(Visitor,Faculty):

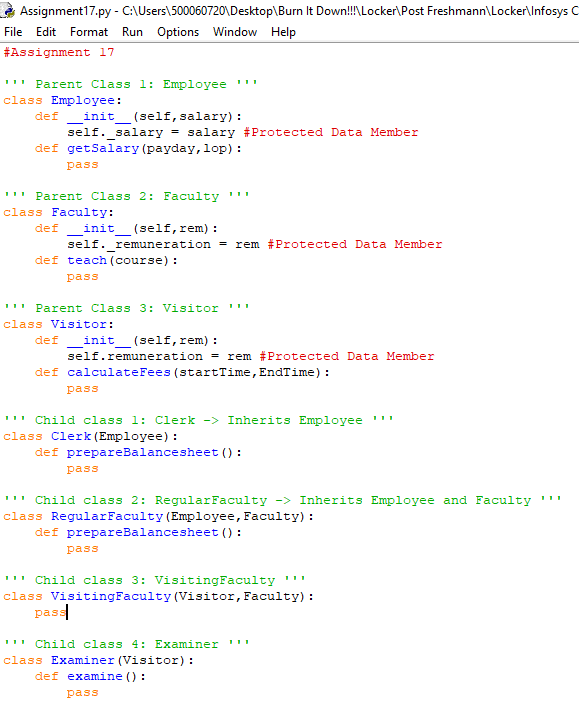
pass

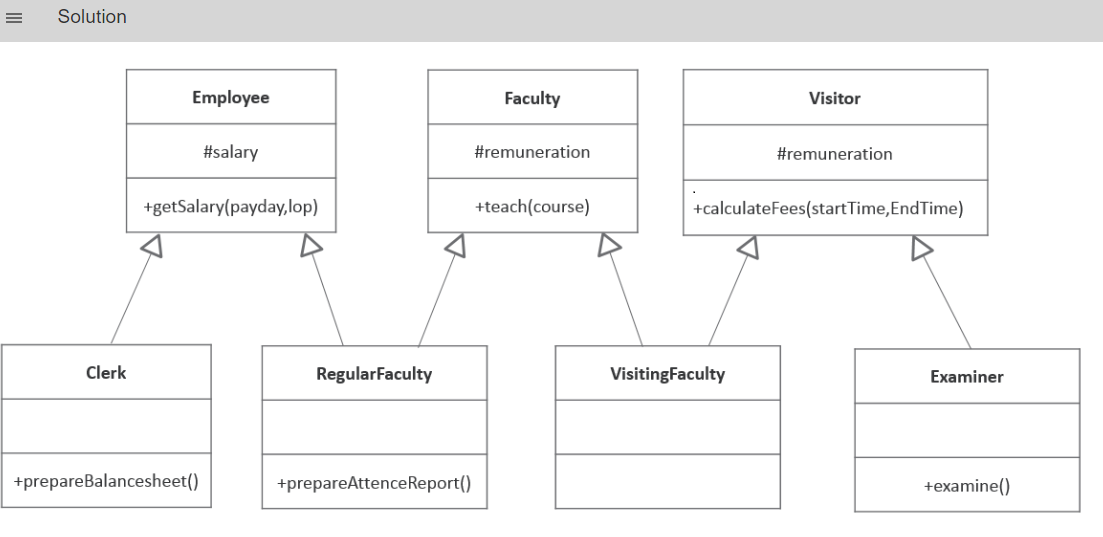
''' Child class 4: Examiner '''

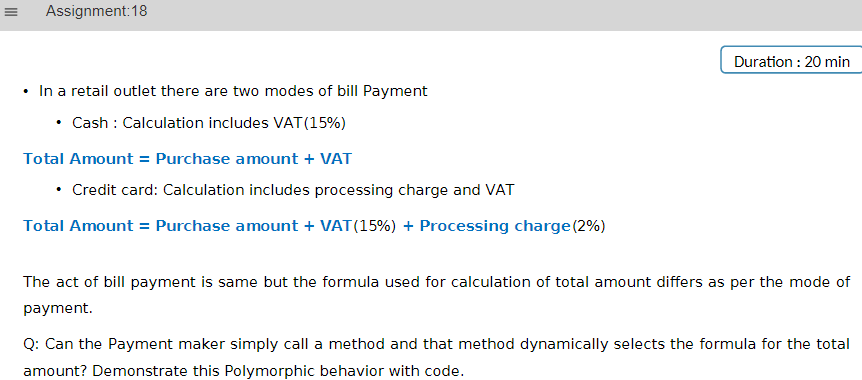
class Examiner(Visitor):

def examine():

pass

**Code Screenshot:**

**Solution:**

**Assignment-18**

**Code:**

**#Assignment 18**

**from abc import ABC, abstractmethod**

**''' Parent Class Payment '''**

**class Payment(ABC):**

**VAT = 1.15 #15% VAT**

**def \_\_init\_\_(self,purchasecost):**

**self.purchasecost = purchasecost**

**@abstractmethod**

**def final\_payment(self):**

**pass**

**''' Subclass Cash Payment '''**

**class Cash(Payment):**

**def \_\_init\_\_(self,purchasecost):**

**Payment.\_\_init\_\_(self,purchasecost)**

**def final\_payment(self):**

**finalcost = self.purchasecost \* Payment.VAT**

**return finalcost**

**''' Subclass Credit Card Payment '''**

**class Creditcard(Payment):**

**processingfee = 1.02**

**def \_\_init\_\_(self,purchasecost):**

**Payment.\_\_init\_\_(self,purchasecost)**

**def final\_payment(self):**

**finalcost = self.purchasecost \* Payment.VAT \* Creditcard.processingfee**

**return finalcost**

**''' Payment Function outside class '''**

**def payshop():**

**costprice = int(input("Enter Purchase Cost: "))**

**print("Payment Method:")**

**print("Press 1: Cash")**

**print("Press 2: Credit Card")**

**inputmenu = int(input("Enter your Payment Choice: "))**

**if inputmenu == 1:**

**print("Payment by Cash")**

**obj = Cash(costprice)**

**sellingprice = obj.final\_payment()**

**print("The total amount to be paid is:",sellingprice)**

**elif inputmenu == 2:**

**print("Payment by Creditcard")**

**obj = Creditcard(costprice)**

**sellingprice = obj.final\_payment()**

**print("The total amount to be paid is:",sellingprice)**

**else:**

**print("Wrong Choice! Please try Again")**

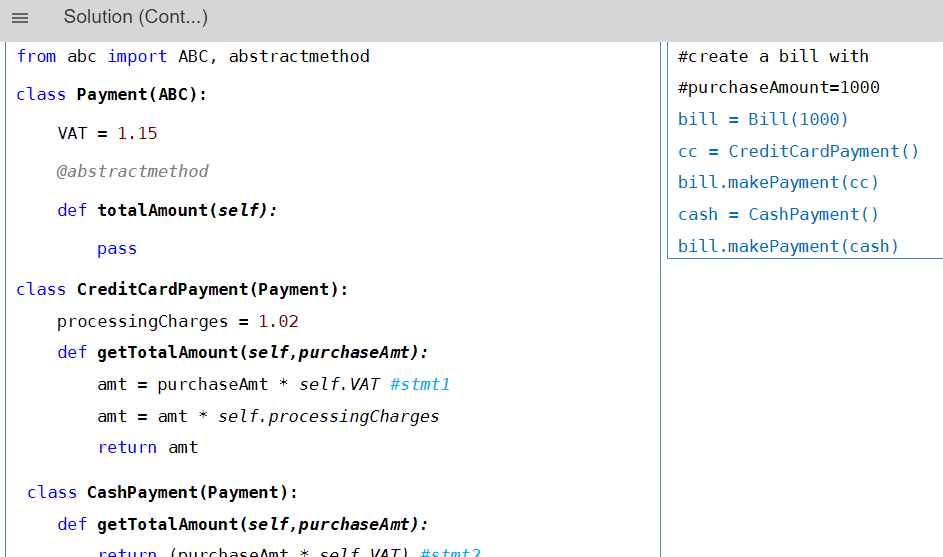
**payshop()**

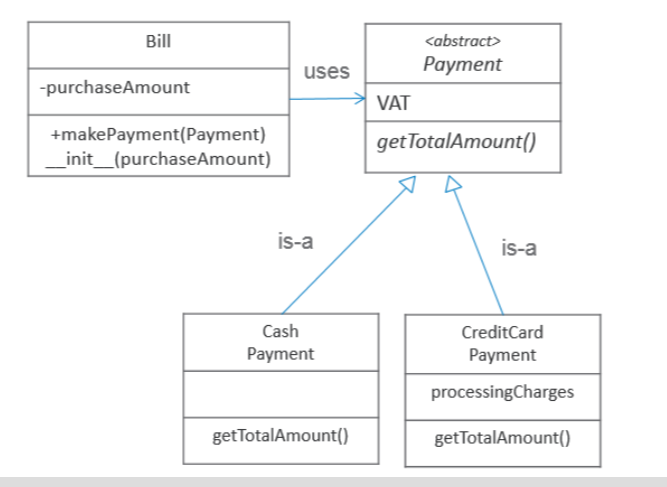
**''' Main Program '''**

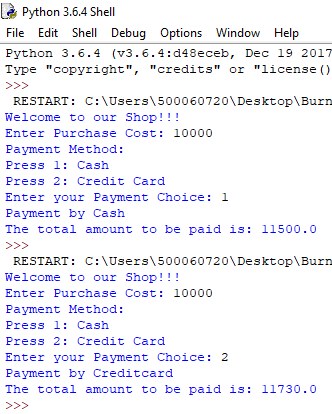
**print("Welcome to our Shop!!!")**

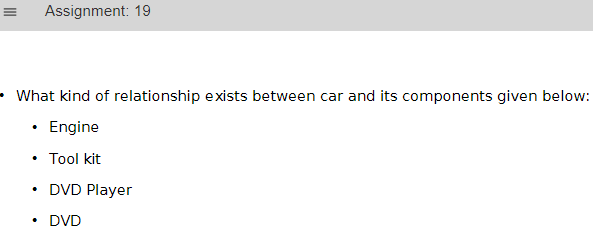
**payshop()**

**Code Screenshot:**

**Output ScreenshoT:**

****

**Python Shell Screenshot:**

**Assignment-19**

**Solution:**

Problem description and justifying logic will determine the relationship type.

• 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

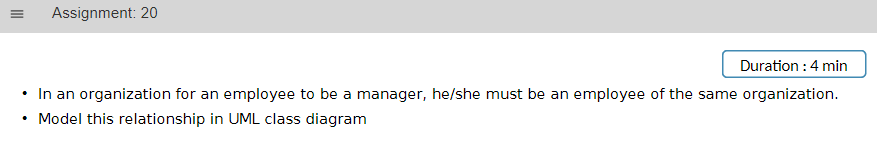
called as Car)

• Car <has-a: Aggregation> DVD Player

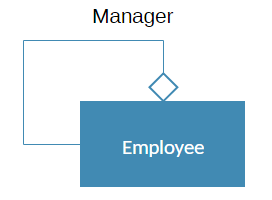
• Car is NOT a Toolkit and DVD is not a part of Car, however car uses them

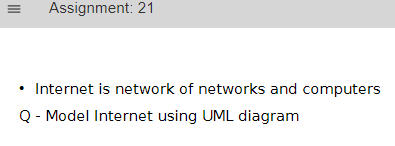
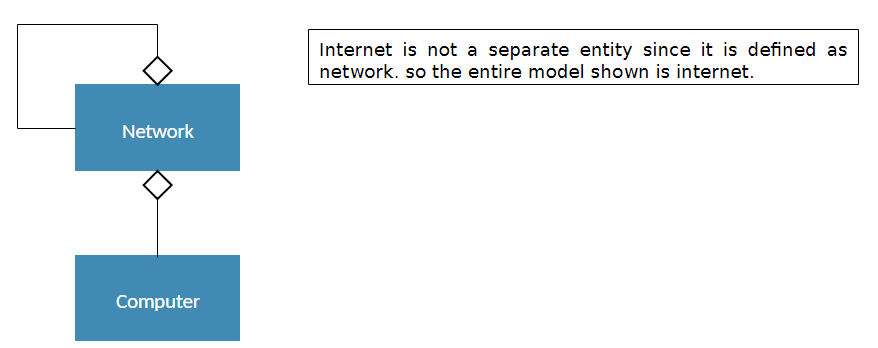
• Car <uses-a> Toolkit

• Car <uses-a> DVD

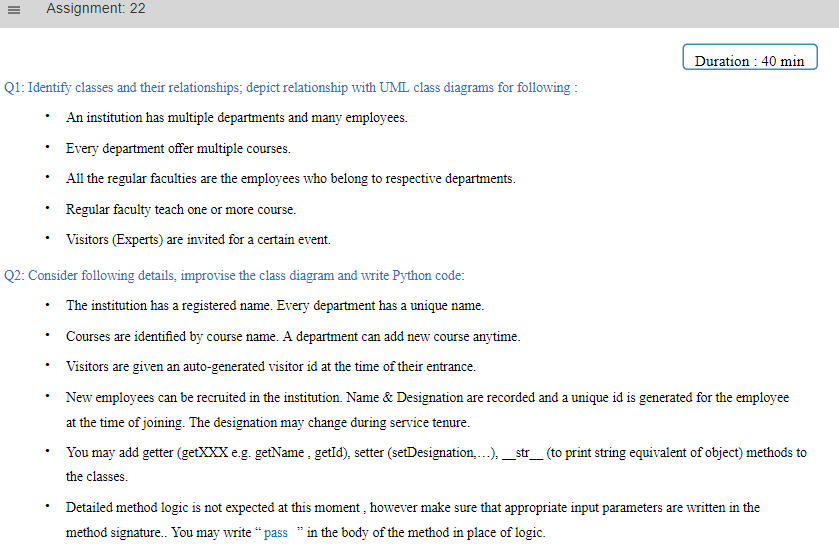
**Assignment-20**

**UML DIAgram:**

****

**Assignment-21**

**Assignment-22**

****

**Code:**

''' Class 1: Visitor '''

class Visitor:

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

self.\_\_name = name

self.\_\_visitorId = int()

''' Class 2: Employee '''

class Employee:

def \_\_init\_\_(name,designation):

self.\_\_empId = int()

self.\_\_name = name

self.\_\_designation = designation

def setDesignation(self,designation):

self.\_\_designation = designation

''' Class 3: Course '''

class Course:

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

self.\_\_courseName = name

''' Class 4: RegularFaculty '''

class RegularFaculty(Employee):

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

self.\_\_deptName = name

def teach(self,course):

#here sent course is object of class course

pass

''' Class 5: Department '''

class Department:

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

self.\_\_deptName = name

self.\_\_courses = []

self.\_\_regularFaculties = []

def addCourse(self,course):

self.\_\_courses.append(course)

def addFaculty(self,faculty):

self.\_\_regularFaculties.append(faculty)

''' Class 6: Institution '''

class Institution:

def \_\_init\_\_(self,regName,department\_count):

self.\_\_registeredName = regName

self.\_\_departments = []

self.\_\_employees = []

for i in range(0,department\_count):

d = Department(name)

self.\_\_departments.append(d)

def event(visitor):

#Visitor is a object of visitor Uses-a

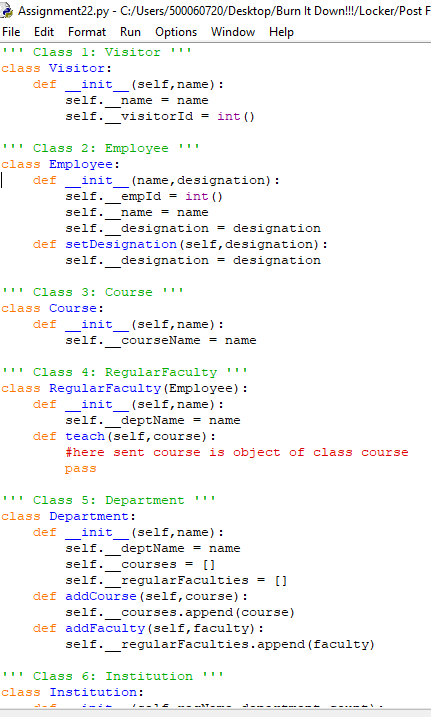
pass

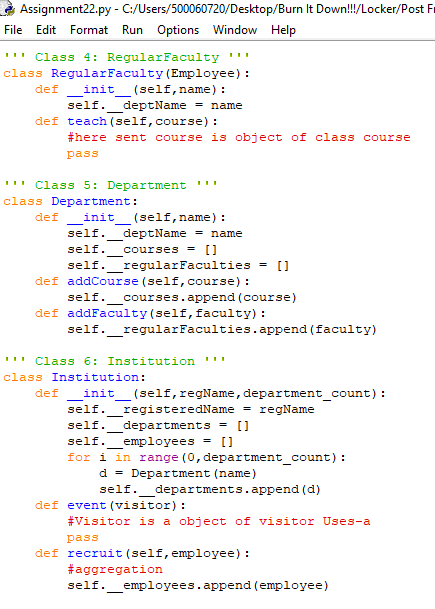
def recruit(self,employee):

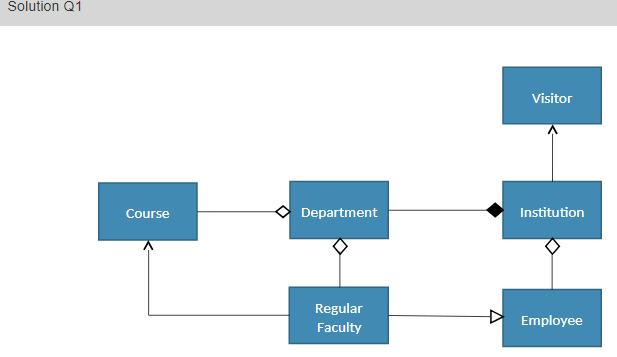
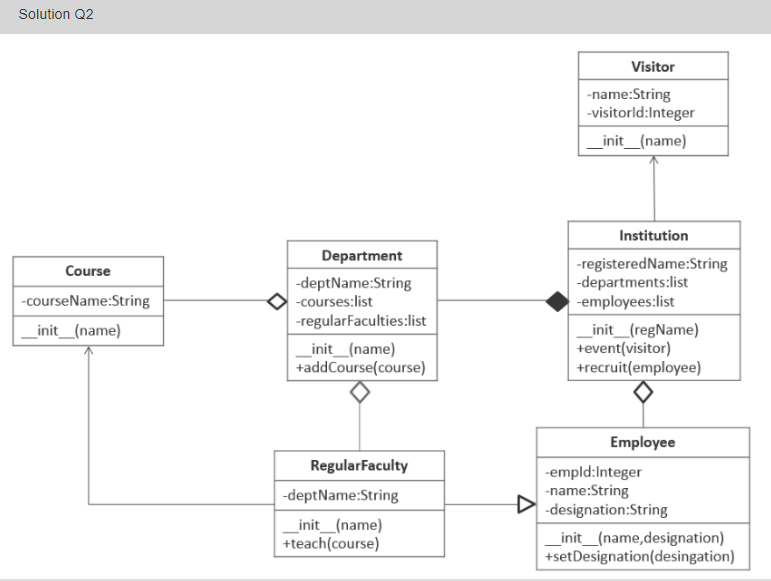
#aggregation

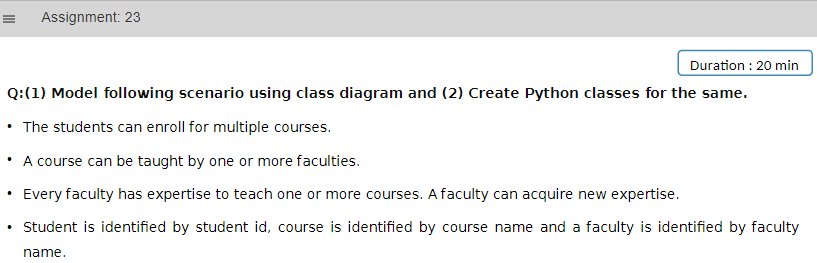
self.\_\_employees.append(employee)

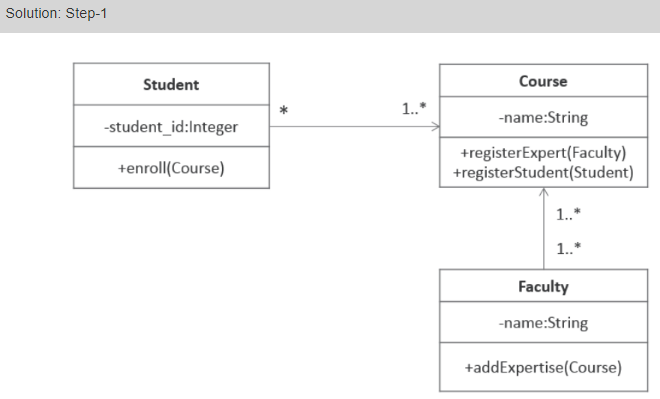
**Code Screenshot**

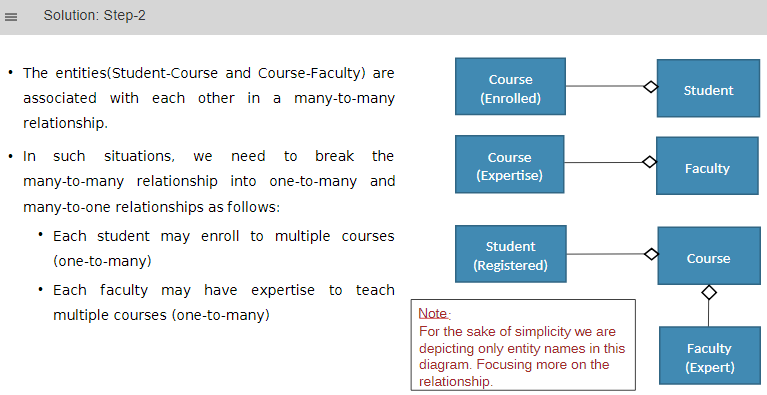
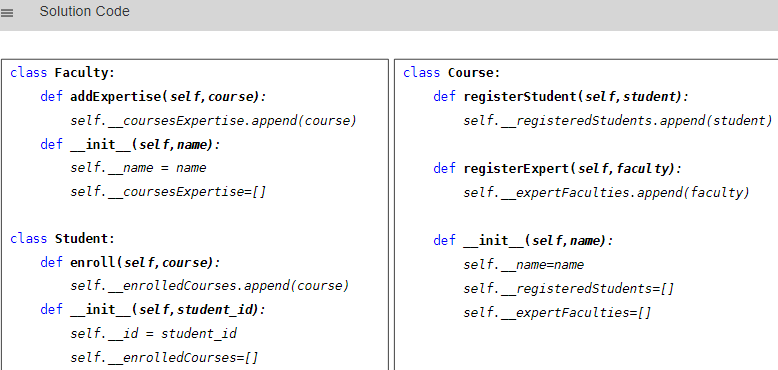
****

****

****

**Assignment-23**

**Solution:**

****

**Code:**

class Faculty:

def addExpertise(self,course):

self.\_\_courseExpertise.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):

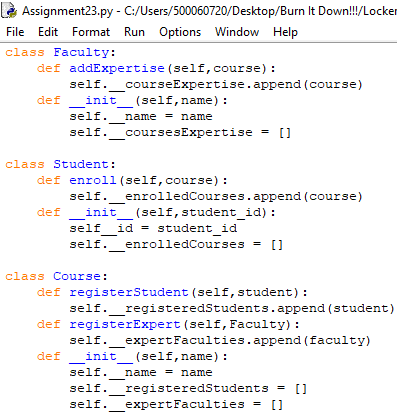
self.\_\_expertFaculties.append(faculty)

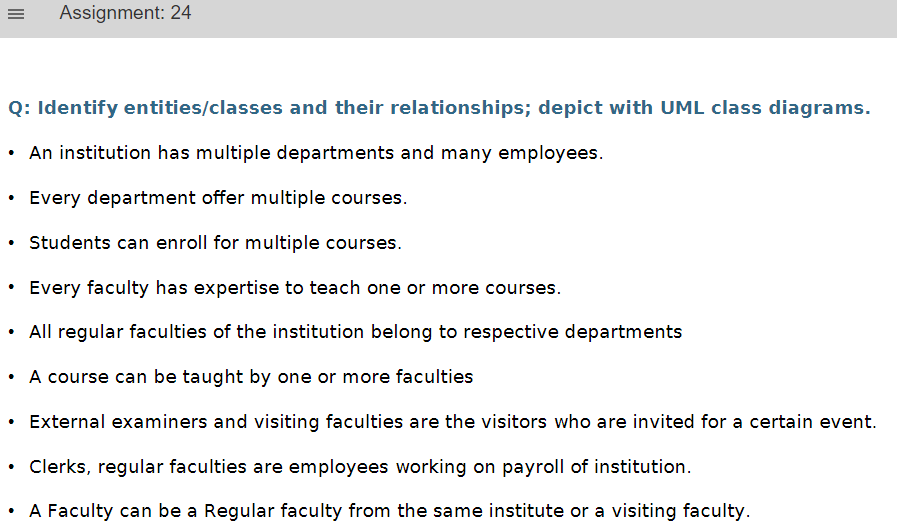
def \_\_init\_\_(self,name):

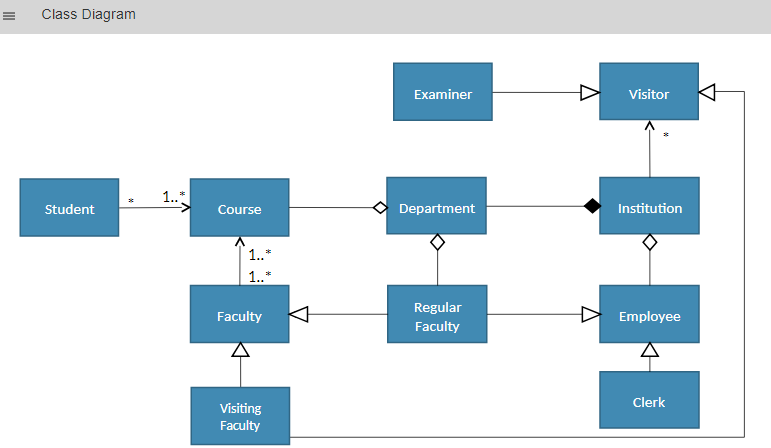
self.\_\_name = name

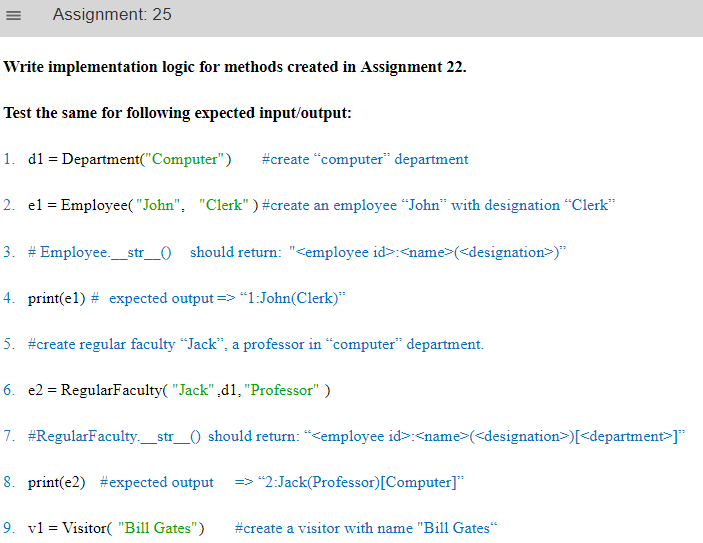
self.\_\_registeredStudents = []

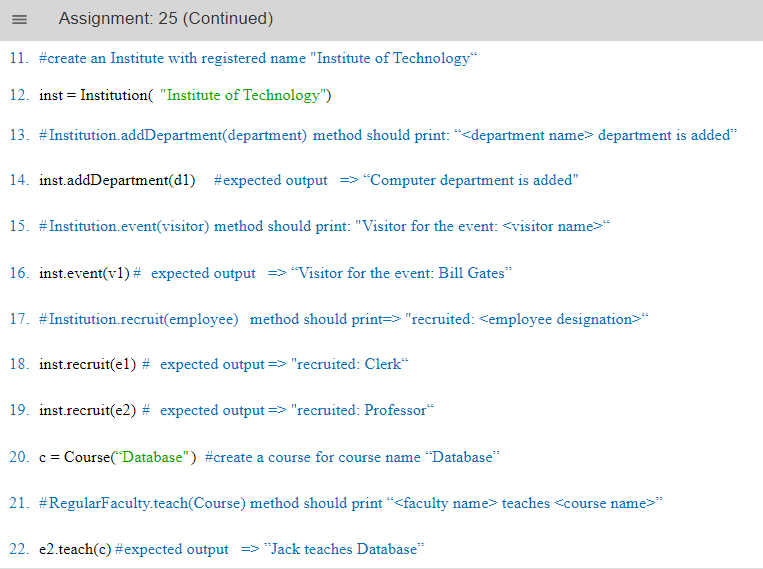
self.\_\_expertFaculties = []

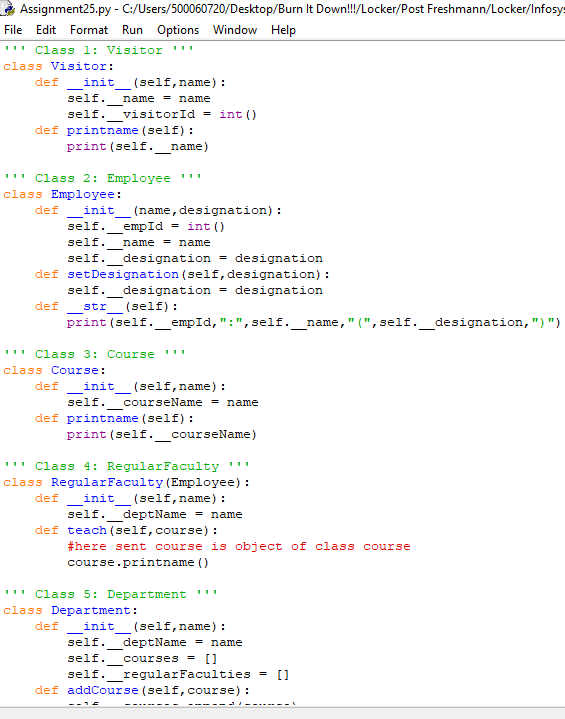
**Code Screenshot:**

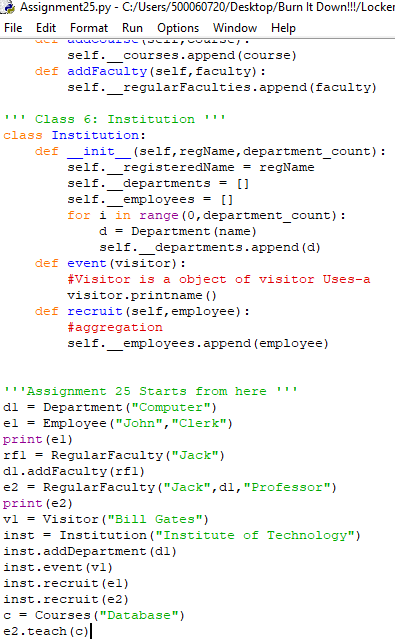
**Assignment-24**

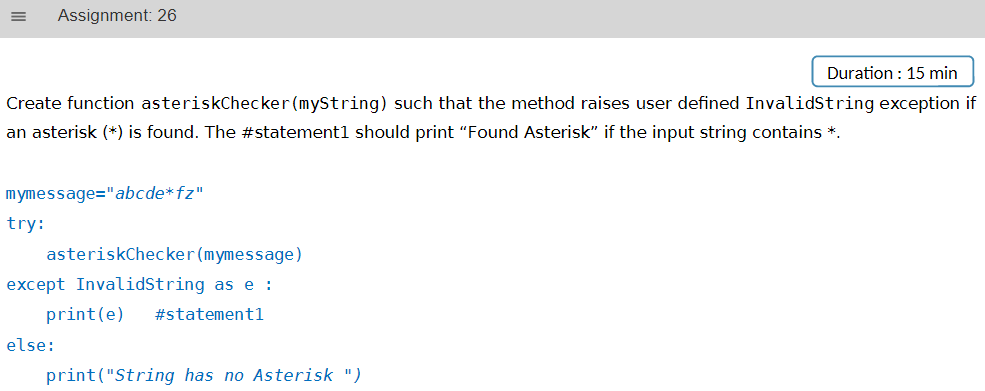
****

**Assignment-25**

****

**Code Screenshot:**

****

**Assignment-26**

**Code:**

class InvalidString(Exception):

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

self.msg = msg

def print\_exception(self):

print("Invalid String Exception:",self.msg)

def asteriskChecker(mymessage):

try:

flag = 0

for k in mymessage :

if k == '\*':

flag = 1

else:

pass

if flag == 1:

raise InvalidString('Asterisk found!!')

else:

pass

except InvalidString as e:

e.print\_exception()

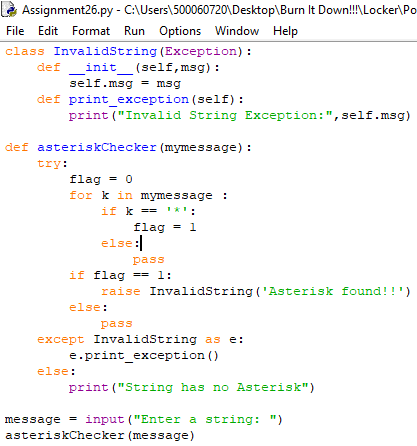
else:

print("String has no Asterisk")

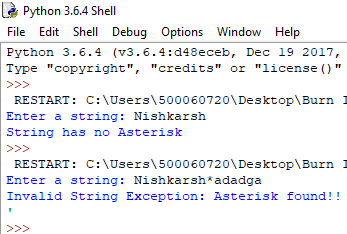
message = input("Enter a string: ")

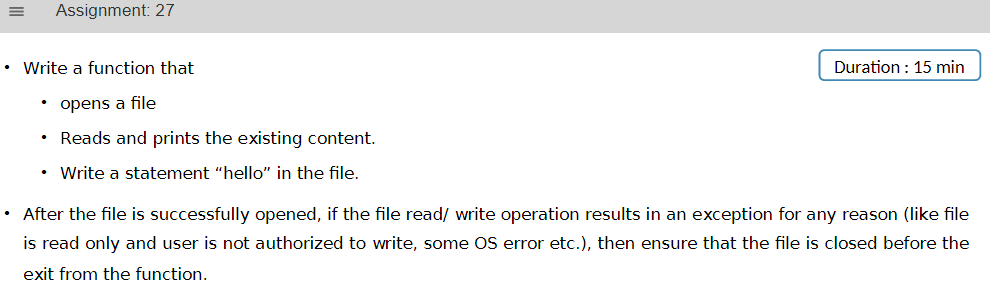
asteriskChecker(message)

**Code Screenshot:**

****

**Output Screenshot:**

****

**Assignment-27**

**Code :**

try:

file = open("Assignment27.txt",'r',1)

content = file.read()

print(content)

file.close()

file = open("Assignment27.txt",'a',1)

file.write("Hello!")

except TypeError:

print("Type Error")

except ValueError:

print("Value Error")

except IOError:

print("Input Output Error")

except:

print("Unknown Runtime error")

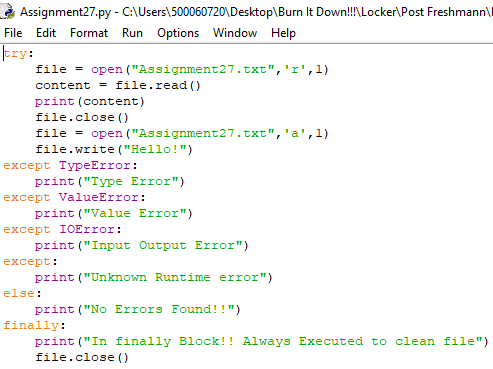
else:

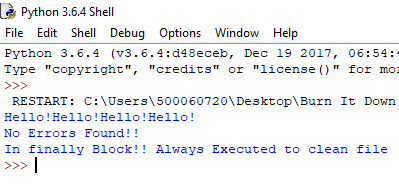
print("No Errors Found!!")

finally:

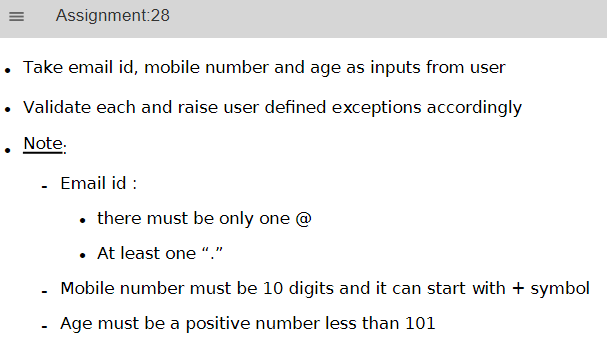
print("In finally Block!! Always Executed to clean file")

file.close()

**Code Screenshot:**



**Output Screenshot**

**Assignment-28**

**Code:**

class validation(Exception):

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

self.msg = msg

def print\_exception(self):

print('User Defined Exception found:',self.msg)

try:

''' Start Try Block '''

#1) Age

age = int(input("Enter your age: "))

if age >= 0 and age <101:

pass

else:

raise validation("Age is not in given limit 0-100") #1

#2) Mobile Number

mobnum = input("Enter your mobile number: ")

if len(mobnum) == 10:

for k in mobnum:

if int(k) >=0 and int(k) <= 9:

pass

else:

raise validation('Invalid character in phone number') #3

elif len(mobnum) == 11:

if mobnum[0] == '+':

for k in range(1,len(mobnum)):

if int(mobnum[k]) >=0 and int(mobnum[k]) <=9:

pass

else:

raise validation('Invalid character in phone number')

else:

raise validation('11 Character Phone number must start with +') #4

else:

raise validation('Incorrect number length') #4

#3) Email

email = input("Enter Email Id: ")

count\_at = 0

count\_period = 0

for k in email:

if k == '@':

count\_at += 1

elif k == '.':

count\_period +=1

else:

pass

if count\_at == 1 and count\_period >=1:

pass

elif count\_at != 1:

raise validation('Incorrect number of @ Symbols used') #5

else:

raise validation('Atleast one period must be used!!!') #6

''' End Try Block '''

except validation as v:

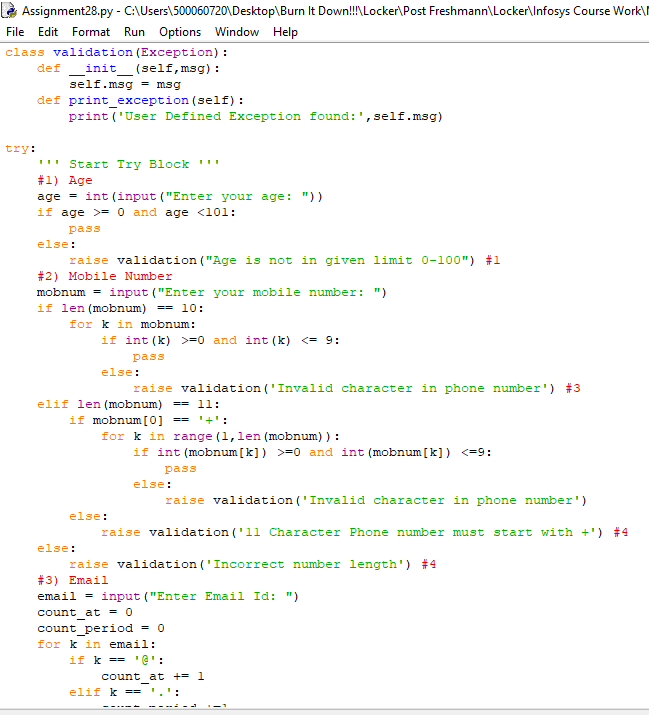
v.print\_exception()

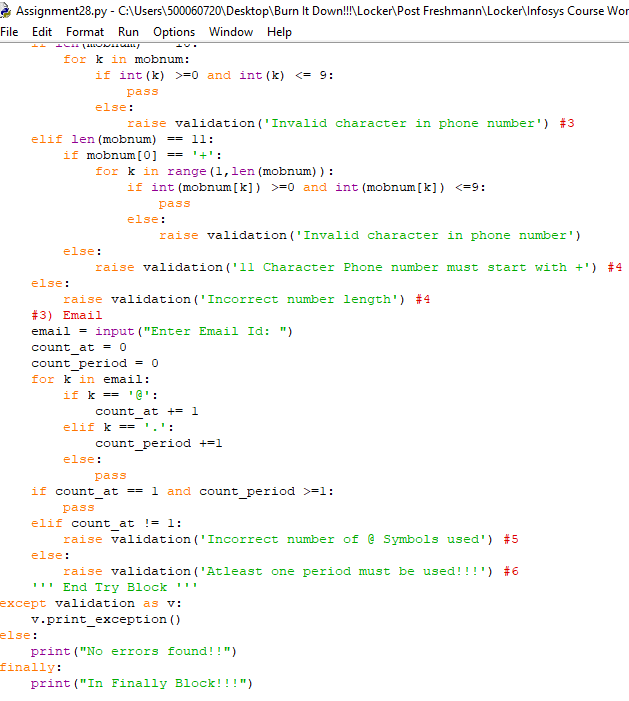
else:

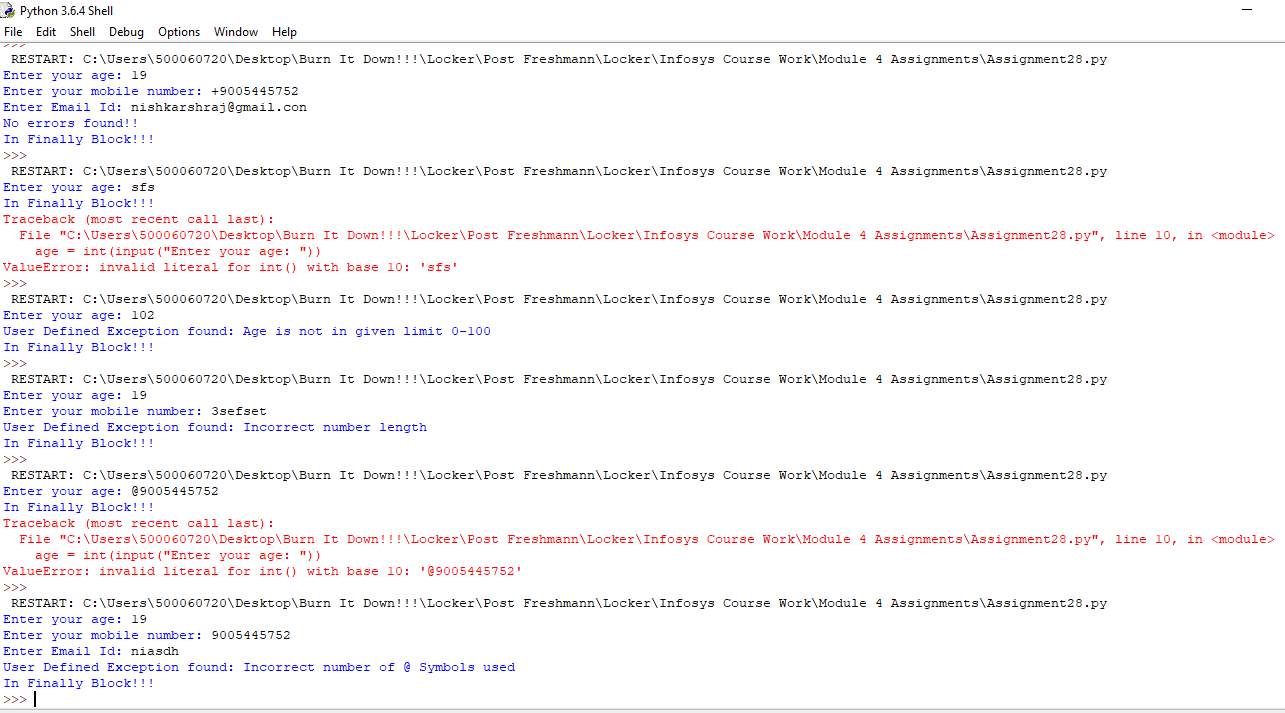
print("No errors found!!")

finally:

print("In Finally Block!!!")

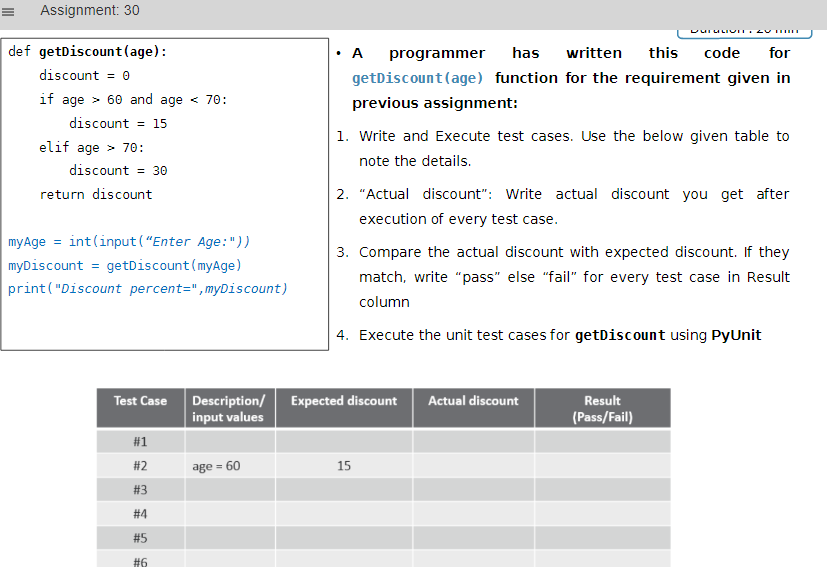
**Code Screenshot:**

****

**Output Screenshot**

**Assignment29**

1. <60 : 0
2. 60 : 15
3. >60 : 15
4. <70 : 15
5. 70 : 30
6. >70 : 30

**Assignment-30**

**Code:**

#Assignment 30

def getDiscount(age):

discount = 0

if age > 60 and age < 70:

discount = 15

elif age > 70: #must be >=70

discount = 30

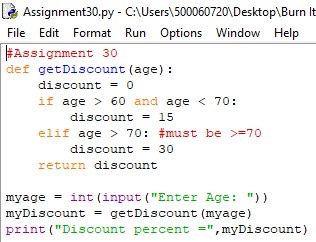
return discount

myage = int(input("Enter Age: "))

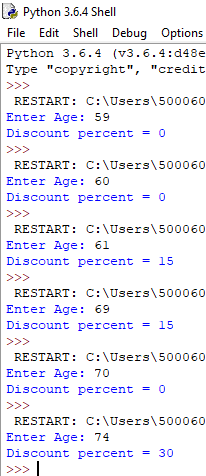
myDiscount = getDiscount(myage)

print("Discount percent =",myDiscount)

**Code Screenshot:**

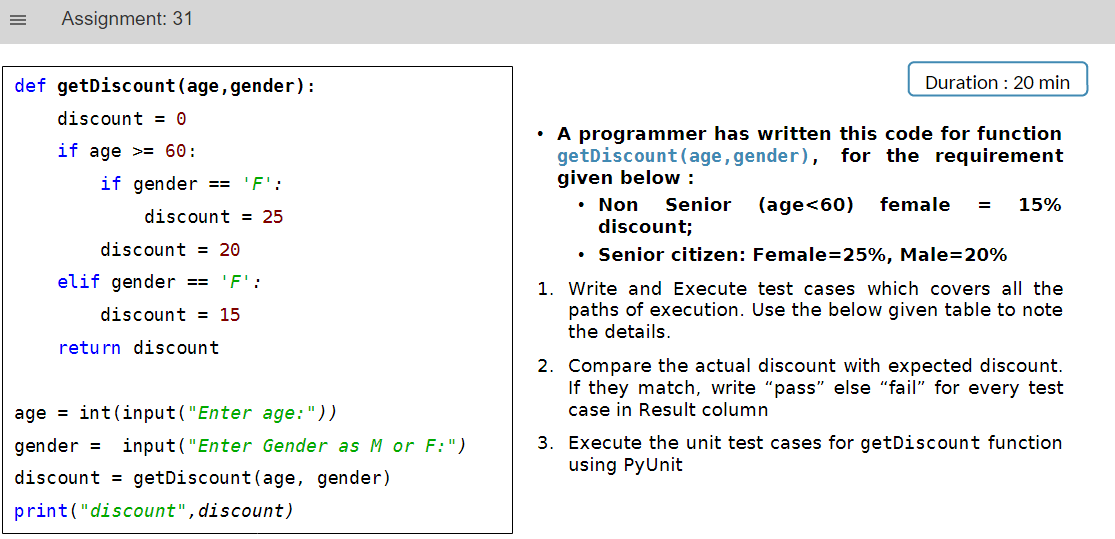
****

**Output Screenshot:**

****

Wrong results:

1. At 60 : must be 15
2. At 70 : must be 30

**Assignment-31**

**Incorrect code:**

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)

**Corrected Code:**

def getDiscount(age,gender):

discount = 0

if age >= 60:

if gender == 'F' or gender == 'f':

discount = 25

elif gender == 'M' or gender == 'm':

discount = 20

elif gender == 'F' or gender == 'f':

discount = 15

return discount

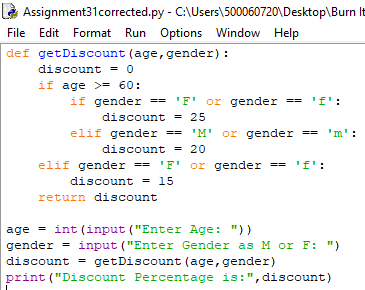
age = int(input("Enter Age: "))

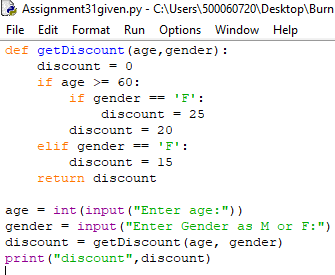
gender = input("Enter Gender as M or F: ")

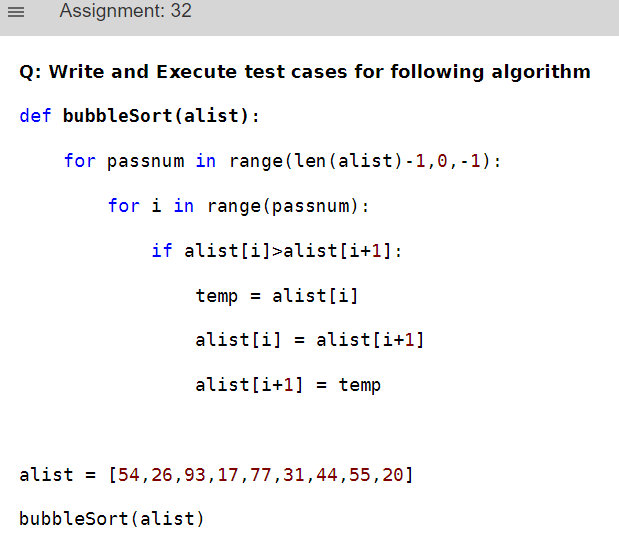
discount = getDiscount(age,gender)

print("Discount Percentage is:",discount)

**Code Screenshot:**

****



**Assignment-32**

**Code:**

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]

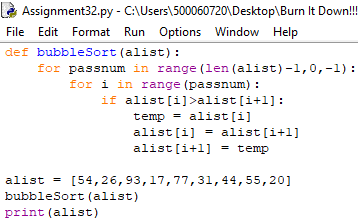
alist[i] = alist[i+1]

alist[i+1] = temp

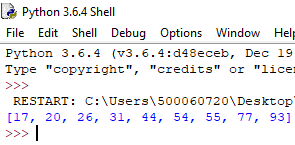
alist = [54,26,93,17,77,31,44,55,20]

bubbleSort(alist)

print(alist)

**Code Screenshot**

**Output Screenshot:**

****