

**Lab Report**

**Course code:** CSE222

**Course Title:** Object Oriented Programming II Lab

**Experiment No:** All Lab

**Experiment name:** All Python Code

**Submitted By**

Refat Pasha

Student id: **0242220005101568**

Section: **63\_k**

Department of Computer Science and Engineering

**Submitted to**

Mr. Muhammed Masum Bakaul

Senior Lecturer

Department of Computer Science and Engineering

**Date of Submission: 07/12/2024**



**Lab Report**

**Course code:** CSE222

**Course Title:** Object Oriented Programming II Lab

**Experiment name:** First Python

**Submitted By**

Refat Pasha

Student id: **0242220005101568**

Section: **63\_k**

Department of Computer Science and Engineering

**Submitted to**

Mr. Muhammed Masum Bakaul

Senior Lecturer

Department of Computer Science and Engineering

**Date of Submission: 07/12/2024**

**Lab 1**

1. variable

a = 5  
b = 6  
c = 'a'  
d = 4.5  
def add(a,d):  
 Sum = a+b  
 return Sum  
  
  
print(add(a, b))  
  
print(type(a))  
print(type(b))  
print(type(c))

1. input

x = input("enter a number: ")  
y = input("enter 2nd number: ")  
sum = int(x)+ int(y)  
print(sum)

1. Function

a = int(input("enter a number: "))  
b = int(input("enter 2nd number: "))  
  
def great\_less(a,b):  
 if a> b:  
 print("a in greater than b!")  
 else:  
 print("b is greater than a!")  
  
  
great\_less(a, b)

3.

b = "Hello, World!"  
print(b[2:5])

4.

for x in range(10):

print('attempt', x+1,(x+1) \* '.')

5.

i=0

while i<10:

print(i)

i+=1

**Output:**

0

1

2

3

4

5

6

7

8

9



**Lab Report**

**Course code:** CSE222

**Course Title:** Object Oriented Programming II Lab

**Experiment name:** For loops, If-else, Methods, Default Parameters

**Submitted By**

Refat Pasha

Student id: **0242220005101568**

Section: **63\_k**

Department of Computer Science and Engineering

**Submitted to**

Mr. Muhammed Masum Bakaul

Senior Lecturer

Department of Computer Science and Engineering

**Date of Submission: 07/12/2024**

6.

for x in (Refat, Pasha):  
 if x == Refat:  
 for y in range(10):  
 print(Pasha)  
print('not present')

**Output:**

PASHA

PASHA

PASHA

not present

7.

age = int(input('Enter your age: '))   
  
if age > 18:  
 print("Congratulations! You are a voter")  
 if age > 30:  
 print("You are middle-aged")  
 else:  
 print("You are older")  
else:  
 print("Sorry! You are not a voter")

**output:**

Enter your age: 21

Congratulations! You are a voter

You are older

8.  
def place\_order(

customer\_name,

price,

quantity,

discount = 0,

shipping\_fee = 10):

total\_price = (price \* quantity) - discount + shipping\_fee

if total\_price > 1000:

discount = .1

cal\_discount = total\_price \* discount

return f"{customer\_name} have %discount: {cal\_discount} \nand total price is: {total\_price-cal\_discount}"

else:

return f"{customer\_name} have total price: {total\_price}"

print(place\_order("refat", price=10, quantity=10))

print(place\_order("pasha", price=1000, quantity=55))

**Output:**

refat have total price: 110

pasha have %discount: 5501.0

and total price is: 49509.0



**Lab Report**

**Course code :** CSE222

**Course Title :** Object Oriented Programming II Lab

**Experiment name:** List, Tuples, Dictionaries

**Submitted By**

Refat Pasha

Student id: **0242220005101568**

Section: **63\_k**

Department of Computer Science and Engineering

**Submitted to**

Mr. Muhammed Masum Bakaul

Senior Lecturer

Department of Computer Science and Engineering

**Date of Submission: 07/12/2024**

9.

thislist = [2,3,2,3,4,5,4]  
newlist = []  
for x in thislist:  
 if x in newlist:  
 continue  
 else:  
 newlist.append(x)  
print(newlist)

**Output:**

**[2, 3, 4, 5]**

10.

#using dictionary

new = {

'name': 'refat',

'city': 'Darshona',

'country': 'bangladesh',

'age': 23,

'color': ['red','black','white']

}

print(new)

#using constructor

new2 = dict(name = 'refat',salary = 600000)

print(new2)

n = new['name']

c = new['color']

print(n)

print(type(n))

#usign get() method

print(new.get('color'))

print(type(c))

#to get keys

x = new.keys()

print(x)

#to get values using values() method

y = new.values()

print(y)

#to get dictionary keye and values!

z = new.items()

print(z)

if 'name' in new:

print("yes its present!")

else:

print("not present!")

#to update

new['name'] = 'pasha'

print(new)

#using update() method

new.update({'name':'agun', 'salary': 3000000000})

print(new)

#to delete usnig pop() method

new.pop('name')

print(new)

#to delete last item using popitem() method

new.popitem()

print(new)

#to delete item using del

del new['age']

print(new)

#to delete entire dict using clear() method

"""new.clear()

print(new)"""

#loop in dictionary

new3 = {

'name': 'refat',

'city': 'Darshona',

'country': 'bangladesh',

'age': 23,

'color': ['red','black','white']

}

for i in new3:

print(i)

##using keys() method to get the values

for i in new3.keys():

print(i)

#using values() method to get the values

for i in new3.values():

print(i)

#using items() method to get all the values

for i in new3.items():

print(i)

#using items() method to get all the values in i, j format

for i , j in new3.items():

print(i, j)

#dictionary copy

#copy using copy() method

new4 = new3.copy()

print(new4)

#copy using dict() method

new5 = dict(new3)



**Lab Report**

**Course code :** CSE222

**Course Title :** Object Oriented Programming II Lab

**Experiment name:** Class and Object

**Submitted By**

Refat Pasha

Student id: **0242220005101568**

Section: **63\_k**

Department of Computer Science and Engineering

**Submitted to**

Mr. Muhammed Masum Bakaul

Senior Lecturer

Department of Computer Science and Engineering

**Date of Submission: 07/12/2024**

class BankAccount:

def \_\_init\_\_(self, account\_holder\_name, initial\_balance=0):

self.account\_holder\_name = account\_holder\_name

self.balance = initial\_balance

def deposit(self, amount):

if amount > 0:

self.balance += amount

print(f"Deposited {amount}. New Balance: {self.balance}")

else:

print("Deposit amount must be positive.")

def withdraw(self, amount):

if amount > 0:

if amount <= self.balance:

self.balance -= amount

print(f"Withdrew {amount}. Remaining balance: {self.balance}")

else:

print("Insufficient funds.")

else:

print("Withdrawal amount must be positive.")

def check\_balance(self):

print(f"Account holder: {self.account\_holder\_name}, Current balance: {self.balance}")

# Create an object of BankAccount with an initial balance of 500

account = BankAccount("Refat", 500)

# Perform a deposit

account.deposit(200)

# Perform a withdrawal

account.withdraw(100)

# Display the balance

account.check\_balance()

**Excercise 1: Class and Object**

class Triangle:

def \_\_init\_\_(self,base,height):

self.base = base

self.height = height

def calculate\_area(self):

area = .5\*(self.base\*self.height)

print(f" Area is: {area}")

t1 = Triangle(10,20)

t1.calculate\_area()

t2 = Triangle(20,30)

t2.calculate\_area()

**Excercise 2: Class and Object**

class Shape: #parent class

def \_\_init\_\_(self,dimention1, dimention2): #constructor

self.dimention1 = dimention1

self.dimention2 = dimention2

def area(self):

print("I am area method of shape class")

class triangle(Shape):

def area(self):

area = .5\*(self.dimention1\*self.dimention2)

print(f"Area of triangle: {area}")

class rectangle(Shape):

def area(self):

area = (self.dimention1\*self.dimention2)

print(f"Area of rectangle: {area}")

t1 = triangle(20,30)

t1.area()

r1 = rectangle(20,30)

r1.area()



**Lab Report**

**Course code :** CSE222

**Course Title :** Object Oriented Programming II Lab

**Experiment name:** Inheritance, Constructor

**Submitted By**

Refat Pasha

Student id: **0242220005101568**

Section: **63\_k**

Department of Computer Science and Engineering

**Submitted to**

Mr. Muhammed Masum Bakaul

Senior Lecturer

Department of Computer Science and Engineering

**Date of Submission: 07/12/2024**

12.

class Vehicle:  
 def \_\_init\_\_(self,model,year):  
 self.model = model  
 self.year = year  
  
  
class Car(Vehicle):  
 def \_\_init\_\_(self,model,year,mileage):  
 super().\_\_init\_\_(model,year)  
 self.mileage = mileage  
  
 def display(self):  
 print(f"name:{self.model} age:{self.year} Stu\_id:{self.mileage}")  
  
  
c1 = Car('Toyota',2020,170)  
print(c1.display())

name:Toyota age:2020 Stu\_id:170

13.

class Persion:  
 def \_\_init\_\_(self,name,age):  
 self.name = name  
 self.age = age  
  
class Student(Persion):  
 def \_\_init\_\_(self,name,age,Stu\_id):  
 super().\_\_init\_\_(name, age)  
 self.Stu\_id = Stu\_id  
 def display(self):  
 print(f"name:{self.name} age:{self.age} Stu\_id:{self.Stu\_id}")  
  
  
s1 = Student(Refat,21,1568)  
print(s1.display())

name:Refat age:21 Stu\_id:1568

14.

class Persion:

name = "Refat " city = "pabna" def pdisplay(self):

print(f"name:{self.name} city:{self.city}")

class Student(Persion):

name = "Pasha" batch =63 def sdisplay(self):

print(f"name:{self.name} batch:{self.batch}")

def address(self):

print("Iam living in Dhaka")

class Teacher(Student):

designation = "Professor" def tdisplay(self):

print(f"job role{self.designation} ")

t = Teacher()

print("from Teacher class")

t.tdisplay()

print("from Student class")

t.sdisplay()

print("from Persion class")

t.pdisplay()

15.

class Employee:

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

self.name = name

self.emp\_id = emp\_id

class Developer(Employee):

def \_\_init\_\_(self, name, emp\_id,prog\_lan,exp\_year):

super().\_\_init\_\_(name, emp\_id)

self.prog\_lan = prog\_lan

self.exp\_year = exp\_year

class SeniorDeveloper(Employee,Developer):

def \_\_init\_\_(self,name, emp\_id,prog\_lan,exp\_year,proj\_man,mon\_team\_size):

super().\_\_init\_\_(name, emp\_id,prog\_lan,exp\_year)

self.proj\_man = proj\_man

self.mon\_team\_size = mon\_team\_size

def display(self):

print(f"name:{self.name},emp\_id:{self.emp\_id},prog\_lan:{self.prog\_lan},exp\_year:{self.exp\_year},proj\_man:{self.proj\_man},mon\_team\_size:{self.mon\_team\_size}")

s1 = SeniorDeveloper("Refat",1568,"python",2024,"yes",5)

print(s1.display())

16.

class Phone: #Phone is parent class

def call(self):

print("You can Call")

def message(self):

print("You can message")

def photo(self):

print("You can capture photo")

class Brand(Phone): #Brand is child class/sub-class

def functions(self):

print("name of functions are :")

a = Brand()

a.functions()

a.photo()

a.call()

a.message()

**Constructor**

class Students: #class

roll = "" #

gpa = "" #

def \_\_init\_\_(self, roll,gpa): #constructor/spacial type of method

self.roll= roll

self.gpa = gpa

def display(self): #method

print(f"Roll: {self.roll}, GPA: {self.gpa}")

rahim = Students(101,3.77) #object

rahim.display()

kahim = Students(102,3.70) #object

kahim.display()



**Lab Report**

**Course code :** CSE222

**Course Title :** Object Oriented Programming II Lab

**Experiment name:** Polymorphism, Method Overriding

**Submitted By**

Refat Pasha

Student id: **0242220005101568**

Section: **63\_k**

Department of Computer Science and Engineering

**Submitted to**

Mr. Muhammed Masum Bakaul

Senior Lecturer

Department of Computer Science and Engineering

**Date of Submission: 07/12/2024**

16.

class Payment:  
 def process\_payment(self):  
 print("Processing payment...")  
  
class CreditCard(Payment):  
 def \_\_init\_\_(self, card\_number, cvv):  
 self.card\_number = card\_number  
 self.cvv = cvv  
  
 def process\_payment(self):  
 print(f"Processing Credit Card payment:\n Card Number: {self.card\_number}, CVV: {self.cvv}")  
  
class PayPal(Payment):  
 def \_\_init\_\_(self, email):  
 self.email = email  
  
 def process\_payment(self):  
 print(f"Processing PayPal payment:\n Email: {self.email}")  
  
class Bitcoin(Payment):  
 def \_\_init\_\_(self, wallet\_address):  
 self.wallet\_address = wallet\_address  
  
 def process\_payment(self):  
 print(f"Processing Bitcoin payment:\n Wallet Address: {self.wallet\_address}")  
  
payments = [  
 CreditCard("1234-5678-9012-3456", "123"),  
 PayPal("user@example.com"),  
 Bitcoin("1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa"),  
]  
  
for payment in payments:  
 payment.process\_payment()

**17.**

class Message:

def send(self):

print("Sending message...")

class SMS(Message):

def \_\_init\_\_(self, phone\_number, message):

self.phone\_number = phone\_number

self.message = message

def send(self):

print(f"Sending SMS to {self.phone\_number}:\n Message: {self.message}")

class Email(Message):

def \_\_init\_\_(self, email\_address, subject, body):

self.email\_address = email\_address

self.subject = subject

self.body = body

def send(self):

print(f"Sending Email to {self.email\_address}:\n Subject: {self.subject}\n Body: {self.body}")

class PushNotification(Message):

def \_\_init\_\_(self, device\_id, priority, message):

self.device\_id = device\_id

self.priority = priority

self.message = message

def send(self):

print(f"Sending Push Notification to {self.device\_id}:\n Priority: {self.priority}\n Message: {self.message}")

messages = [

SMS("123-456-7890", "Hello via SMS!"),

Email("user@example.com", "Greetings", "This is the body of the email."),

PushNotification("device\_1234", "high", "New notification!"),

]

for message in messages:

message.send()

**Method Overriding**

class Phone: #Phone is parent class  
  
 def \_\_init\_\_(self):  
 print("I am in Phone Class")#first call  
  
  
class Brand(Phone): #Brand is child class/sub-class  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_() #return to super class/parent class  
 print("I am in Brand class")#overring call of super class  
  
  
a = Brand()



**Lab Report**

**Course code :** CSE222

**Course Title :** Object Oriented Programming II Lab

**Experiment name:** Abstraction

**Submitted By**

Refat Pasha

Student id: **0242220005101568**

Section: **63\_k**

Department of Computer Science and Engineering

**Submitted to**

Mr. Muhammed Masum Bakaul

Senior Lecturer

Department of Computer Science and Engineering

**Date of Submission: 07/12/2024**

18.

**Problem 1:**

from abc import ABC,abstractmethod

class BankAccount(ABC):

@abstractmethod

def deposit(self):

pass

@abstractmethod

def withdraw(self):

pass

@abstractmethod

def get\_balance(self):

pass

class SavingAccount(BankAccount):

def \_\_init\_\_(self, dep\_bal, with\_bal ):

self.dep\_bal = dep\_bal

self.with\_bal = with\_bal

self.bal = 0

self.deposit()

self.withdraw()

def deposit(self):

if self.dep\_bal > 0:

self.bal += self.dep\_bal

def withdraw(self):

if self.with\_bal > self.bal:

print("Insufficient Balance")

else:

self.bal -= self.with\_bal

def get\_balance(self):

print(f"Balance after Deposit: {self.dep\_bal}\n"

f"Balance after Withdraw: {self.with\_bal}\n"

f"Current Balance: {self.bal}")

obj = SavingAccount(2000,200)

obj.get\_balance()

**Problem 2:**

from abc import ABC,abstractmethod

class Shape(ABC):

@abstractmethod

def area(self):

pass

class Rectangle(Shape):

def \_\_init\_\_(self, width, height):

self.width = width

self.height = height

self.area()

def area(self):

self.Rect\_area = self.width\*self.height

def display(self):

print(f"Area of Rectangle: {self.Rect\_area}")

class Circle(Shape):

def \_\_init\_\_(self,radius,pi = 3.1416):

self.radius = radius

self.pi = pi

self.area()

def area(self):

self.Circle\_area = self.pi\*(self.radius\*self.radius)

def display(self):

print(f"Area of Circle: {self.Circle\_area}")

obj = Rectangle(5,10)

obj.display()

obj2 = Circle(4)

obj2.display()



**Lab Report**

**Course code :** CSE222

**Course Title :** Object Oriented Programming II Lab

**Experiment name:** Exception Handling

**Submitted By**

Refat Pasha

Student id: **0242220005101568**

Section: **63\_k**

Department of Computer Science and Engineering

**Submitted to**

Mr. Muhammed Masum Bakaul

Senior Lecturer

Department of Computer Science and Engineering

**Date of Submission: 07/12/2024**

20.

try:

num1 = 10

num2 = 0

result = num1/num2

except ZeroDivisionError as e:

print(f"You can't divide a number by zero {e}")

else:

print(result)

finally:

print('It will always run')

21.

#value\_error\_try:

num = 10

result = num + num2

except NameError as e:

print(f"exception caught", e)

finally:

print('It will always run')

22.

class CustomError(Exception):

pass

try:

raise CustomError('This is a custom error')

except CustomError as e:

print(f"Custom Error: {e}")

def check\_age(age):

try:

if age < 18:

raise CustomError('You are not eligible')

else:

print('You are eligible')

except CustomError as e:

print(f"Custom Error: {e}")

check\_age(20)

check\_age(15)



**Lab Report**

**Course code :** CSE222

**Course Title :** Object Oriented Programming II Lab

**Experiment name:** AllLab Evaluations

**Submitted By**

Refat Pasha

Student id: **0242220005101568**

Section: **63\_k**

Department of Computer Science and Engineering

**Submitted to**

Mr. Muhammed Masum Bakaul

Senior Lecturer

Department of Computer Science and Engineering

**Date of Submission: 07/12/2024**

**Lab Evaluation 1**

Question 1 solve:

class Employee:  
 def \_\_init\_\_(self,name,employee\_id):  
 self.name = name  
 self.employee\_id = employee\_id  
 def display(self):  
 print(f"Employee Name :{self.name} \nSalary : {self.employee\_id}")  
class Developer(Employee):  
 def \_\_init\_\_(self,programming\_lang,exp\_year):  
 self.programming\_lang = programming\_lang  
 self.exp\_year = exp\_year  
  
 def display(self):  
 print(f"Developer Name : {self.name} \nID : {self.employee\_id} "  
 f"Programming Language : {self.programming\_lang} \nExprience Year : {self.exp\_year}")  
class SeniorDeveloper(Developer):  
 def \_\_init\_\_(self,name,employee\_id,programming\_lang,  
 exp\_year,project\_managed,mentoring\_team\_size):  
 self.name = name  
 self.employee\_id = employee\_id  
 self.programming\_lang = programming\_lang  
 self.exp\_year = exp\_year  
 self.project\_managed = project\_managed  
 self.mentoring\_team\_size = mentoring\_team\_size  
 def display(self):  
 print(f"Developer Name : {self.name} \nID : {self.employee\_id} "  
 f"\nProgramming Language : {self.programming\_lang} \nExprience Year : {self.exp\_year}"  
 f"\nProject Managed : {self.project\_managed} \nTeam Size : {self.mentoring\_team\_size}")  
  
x = SeniorDeveloper("Refat",101,"Python",3,5,10)  
x.display()

**OutPut:**

Developer Name : Refat

ID : 101

Programming Language : Python

Exprience Year : 3

Project Managed : 5

Team Size : 10

Question 2 Solve:

class Vehicle:  
 def \_\_init\_\_(self,seating\_capacity):  
 self.seating\_capacity = seating\_capacity  
 def fare(self):  
 print(f"Fare is : {self.seating\_capacity\*100}")  
class Bus(Vehicle):  
 def \_\_init\_\_(self,seating\_capacity):  
 self.seating\_capacity = seating\_capacity  
 def fare(self):  
 total = self.seating\_capacity\*100  
 final\_amount = total + total\*.1  
 print(f"Final amount : {final\_amount}")  
x = Bus(50)  
x.fare()

**OutPut:**

Final amount : 5500.0

**Lab Evaluation 2**

Question 1 solve:

class staff:  
  
 def evaluatee\_performance(self):  
 print("this is staff class")  
  
  
class teacher(staff):  
  
 def \_\_init\_\_(self,avg\_score,num\_of\_class):  
 self.avg\_score = avg\_score  
 self.num\_of\_class = num\_of\_class  
  
 def evaluate\_performance(self):  
 print(f"Teacher evaluate using avg num: {self.avg\_score} and num of class taught : {self.num\_of\_class}")  
  
class administrator(staff):  
  
 def \_\_init\_\_(self, num\_of\_event\_org, office\_efficiency\_score):  
 self.num\_of\_event\_org = num\_of\_event\_org  
 self.office\_efficiency\_score = office\_efficiency\_score  
  
 def evaluate\_performance(self):  
 print(f"Administrator number of event organized: {self.num\_of\_event\_org} and Office Efficiency Score : {self.office\_efficiency\_score}")  
  
  
class counselor(staff):  
  
 def \_\_init\_\_(self, counseling\_session, stu\_feedback\_score):  
 self.counseling\_session = counseling\_session  
 self.stu\_feedback\_score = stu\_feedback\_score  
  
 def evaluate\_performance(self):  
 print(f"Counselor number of counseling session: {self.counseling\_session} and Student Feedback Score : {self.stu\_feedback\_score}")  
  
evaluates = {  
 teacher(97,16),  
 administrator(9,7.8),  
 counselor(8,8.9)  
}  
for evaluate in evaluates:  
 evaluate.evaluate\_performance()

**OutPut:**

Administrator number of event organized: 9 and Office Efficiency Score : 7.8

Teacher evaluate using avg num: 97 and num of class taught : 16

Counselor number of counseling session: 8 and Student Feedback Score : 8.9

Question 2 solve:

class Participant:  
  
 def perform\_action(self):  
 print("this is participant class")  
  
  
class Student(Participant):  
 def \_\_init\_\_(self,view\_course\_material, submit\_assignment):  
 self.view\_course\_material = view\_course\_material  
 self.submit\_assignment = submit\_assignment  
 def perform\_action(self):  
 print(f"Student course: {self.view\_course\_material} and submit assignment: {self.submit\_assignment}")  
  
  
  
class Instructor(Participant):  
 def \_\_init\_\_(self, upload\_material, grade\_assignment, Conduct\_class):  
 self.upload\_material = upload\_material  
 self.grade\_assignment = grade\_assignment  
 self.Conduct\_class = Conduct\_class  
  
 def perform\_action(self):  
 print(f"Instructor Material Upoloaded: {self.upload\_material} , grade assignment: {self.grade\_assignment} and conduct Class: {self.Conduct\_class}")  
  
  
  
class teachingAssustabt(Participant):  
  
 def \_\_init\_\_(self, assist\_class, grade\_assignment):  
 self.assist\_class = assist\_class  
 self.grade\_assignment = grade\_assignment  
  
 def perform\_action(self):  
 print(f"Teacher assist class : {self.assist\_class} and grade assignment: {self.grade\_assignment}")  
  
  
  
class Course(Student,Instructor,teachingAssustabt):  
 def \_\_init\_\_(self,view\_course\_material, submit\_assignment, upload\_material, grade\_assignment, Conduct\_class, assist\_class):  
 self.view\_course\_material = view\_course\_material  
 self.submit\_assignment = submit\_assignment  
 self.upload\_material = upload\_material  
 self.grade\_assignment = grade\_assignment  
 self.Conduct\_class = Conduct\_class  
 self.assist\_class = assist\_class  
   
 def perform\_action(self):  
 print(f"Student course: {self.view\_course\_material} and submit assignment: {self.submit\_assignment}")  
 print(f"Instructor Material Upoloaded: {self.upload\_material} , grade assignment: {self.grade\_assignment} and conduct Class: {self.Conduct\_class}")  
 print(f"Teacher assist class : {self.assist\_class} and grade assignment: {self.grade\_assignment}")  
  
  
  
#with class class  
x = Course(8,10,11,9.9,16,3)  
  
x.perform\_action()  
  
  
#without Course class  
# evaluates = {  
# Student(97,16),  
# Instructor(9,7.8,5),  
# teachingAssustabt(8,8.9)  
# }  
# for evaluate in evaluates:  
# evaluate.perform\_action()

**OutPut:**

Student course: 8 and submit assignment: 10

Instructor Material Upoloaded: 11 , grade assignment: 9.9 and conduct Class: 16

Teacher assist class : 3 and grade assignment: 9.9

Question 3 Solve:

class ProductSearch:  
  
 def search(self,\*a):  
 print(f"{a}")  
  
product\_search = ProductSearch()  
product\_search.search("Smartphone")  
product\_search.search("Smartphone", "300-600")  
product\_search.search("samsung", "300-600", 4.0)

**OutPut:**

('Smartphone',)

('Smartphone', '300-600')

('samsung', '300-600', 4.0)

**Lab Evaluation 2**

Question Solve:

from abc import ABC, abstractmethod  
  
class Order(ABC):  
 @abstractmethod  
 def calculate\_total(self):  
 pass  
  
 @abstractmethod  
 def get\_order\_details(self):  
 pass  
  
  
class OnlineOrder(Order):  
 def \_\_init\_\_(self, order\_id, delivery\_charge):  
 self.order\_id = order\_id  
 self.items = [] # To store multiple items  
 self.delivery\_charge = delivery\_charge  
   
 def add\_item(self, name, price, quantity):  
 self.items.append({"name": name, "price": price, "quantity": quantity})  
  
 def calculate\_total(self):  
 total = sum(item['price'] \* item['quantity'] for item in self.items) + self.delivery\_charge  
 return total  
  
 def get\_order\_details(self):  
 print(f"Online Order ID: {self.order\_id}")  
 print("Items:")  
 for item in self.items:  
 print(f" {item['name']}: ${item['price']} x {item['quantity']}")  
 print(f"Delivery Charge: ${self.delivery\_charge}")  
 print(f"Total: ${self.calculate\_total()}")  
  
  
class InStoreOrder(Order):  
 def \_\_init\_\_(self, order\_id, discount=0):  
 self.order\_id = order\_id  
 self.items = [] # To store multiple items  
 self.discount = discount # Discount as a percentage  
  
 def add\_item(self, name, price, quantity):  
 self.items.append({"name": name, "price": price, "quantity": quantity})  
  
 def calculate\_total(self):  
 subtotal = sum(item['price'] \* item['quantity'] for item in self.items)  
 total = subtotal - (subtotal \* self.discount / 100)  
 return total  
  
 def get\_order\_details(self):  
 print(f"In-Store Order ID: {self.order\_id}")  
 print("Items:")  
 for item in self.items:  
 print(f" {item['name']}: ${item['price']} x {item['quantity']}")  
 print(f"Discount: {self.discount}%")  
 print(f"Total: ${self.calculate\_total()}")  
  
  
  
  
# Online Order  
online\_order = OnlineOrder("xyz", delivery\_charge=25)  
online\_order.add\_item("Laptop", 1000, 1)  
online\_order.add\_item("Mouse", 50, 2)  
online\_order.add\_item("cleaning kit", 50, 5)  
online\_order.get\_order\_details()  
  
print("\n")  
  
# In-Store Order  
in\_store\_order = InStoreOrder("abc", discount=10)  
in\_store\_order.add\_item("Monitor", 300, 1)  
in\_store\_order.add\_item("Keyboard", 100, 1)  
in\_store\_order.get\_order\_details()

**OutPut:**

Online Order ID: xyz

Items:

Laptop: $1000 x 1

Mouse: $50 x 2

cleaning kit: $50 x 5

Delivery Charge: $25

Total: $1375

In-Store Order ID: abc

Items:

Monitor: $300 x 1

Keyboard: $100 x 1

Discount: 10%

Total: $360.0