

**Assignment**

**Course code: CSE-221**

**Course Title: OOP II**

**Submitted to:**

**Name: Shahadat Hossain**

**Senior Lecturer, Department of CSE,**

**Daffodil international University**

**Submitted by:**

**Name: Anika Afrin Moumeta (221-15-5142)**

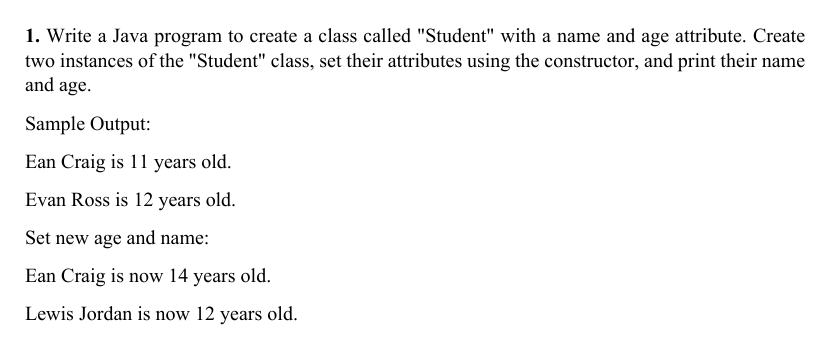
**Section :61-A2**

**Department of CSE,**

**Daffodil international University**

LAB 1-2

1.



#f in print means frozen

class student:

  def \_\_init\_\_(self,name,age):

    self.name=name

    self.age=age

  def getName(self):

    return self.name

  def getAge(self):

    return self.age

s1=student("Ean Craig",11)

s2=student("Evan Ross",12)

print(f"{s1.getName()} is {s1.getAge()} years old.")

print(f"{s2.getName()} is {s2.getAge()} years old.")

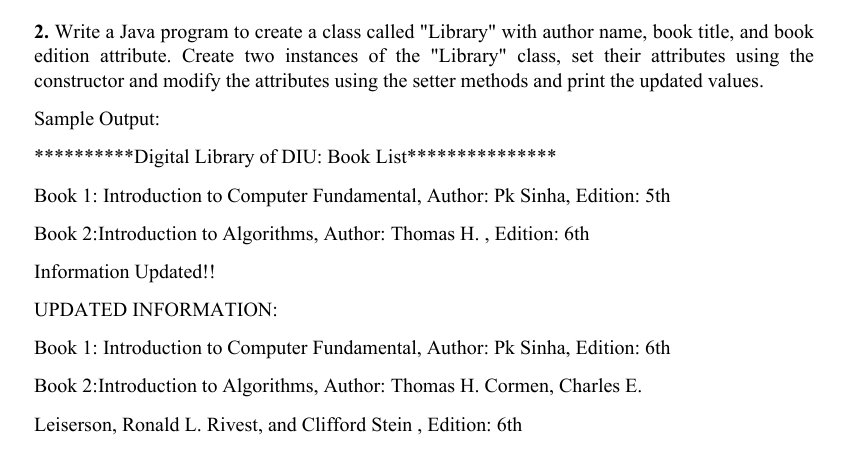
s1.age=14

s2.name="Lewis Jordan"

print(f"\nSet new age and name:\n{s1.getName()} is now {s1.getAge()} years old.")

print(f"{s2.getName()} is now {s2.getAge()} years old.")

2.



class Library:

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

    self.name=name

    self.title=title

    self.edition=edition

  def getName(self):

    return self.name

  def setName(self,name):

    self.name= name

  def getTitle(self):

    return self.title

  def setTitle(self,title):

    self.title=title

  def getEdition(self):

    return self.edition

  def setEdition(self,edition):

    self.edition=edition

b1=Library("Pk Sinha","Introduction to Computer Fundamental","5th")

b2=Library("Thomas H.","Introduction to Algorithms","6th")

print(f"\*\*\*\*\*\*\*\*\*\*Digital Library of DIU: Book List\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(f"Book 1: {b1.getTitle()}, Author: {b1.getName()}, Edition: {b1.getEdition()}")

print(f"Book 2:{b2.getTitle()}, Author: {b2.getName()} , Edition: {b2.getEdition()}")

b1.setEdition("6th")

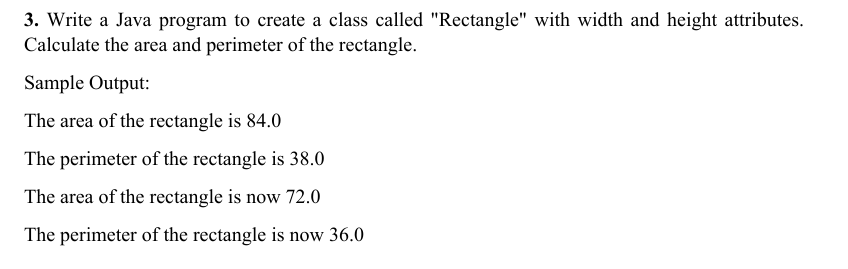
b2.setName("Thomas H. Cormen, Charles E.\nLeiserson, Ronald L. Rivest, and Clifford Stein")

print("\nInformation Updated!!\n\nUPDATED INFORMATION:\n")

print(f"Book 1: {b1.getTitle()}, Author: {b1.getName()}, Edition: {b1.getEdition()}")

print(f"Book 2:{b2.getTitle()}, Author: {b2.getName()} , Edition: {b2.getEdition()}")

3.



class Rectangle:

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

    self.width=width

    self.height=height

  def area(self):

    return self.width\*self.height

  def perimeter(self):

    return (2\*(self.height+self.width))

r1=Rectangle(12.0,7.0)

print(f"The area of the rectangle is {r1.area()}")

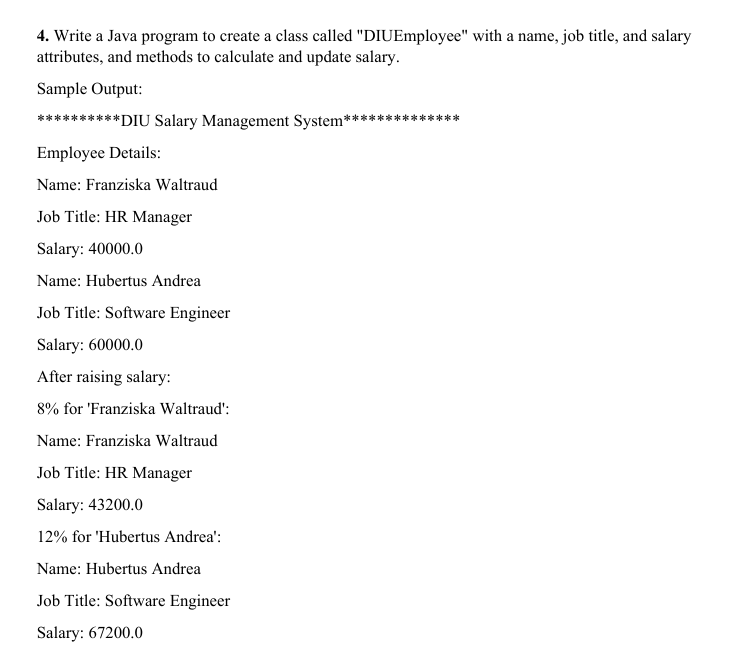
print(f"The perimeter of the rectangle is {r1.perimeter()}")

r1.height=6.0

r1.width=12.0

print(f"\nThe area of the rectangle is now {r1.area()}")

print(f"The perimeter of the rectangle is now {r1.perimeter()}")



class DIUEmployee:

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

    self.name=name

    self.job=job

    self.salary=salary

  def update(self,percent):

    self.percent=percent

    self.salary=self.salary+self.salary\*percent/100

    print(f"{percent}% for '{self.name}':")

  def display(self):

    print(f"Name: {self.name}\nJob Title: {self.job}\nSalary: {self.salary}")

e1=DIUEmployee("Franziska Waltraud","HR Manager",40000.0)

e2=DIUEmployee("Hubertus Andrea","HR Software Engineer",60000.0)

print("\*\*\*\*\*\*\*\*\*\*DIU Salary Management System\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\nEmployee Details:")

e1.display()

print("\n")

e2.display()

print("\nAfter raising salary:\n")

e1.update(8)

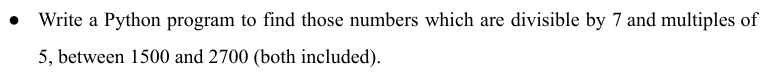
e1.display()

print("\n")

e2.update(12)

e2.display()

LAB-3



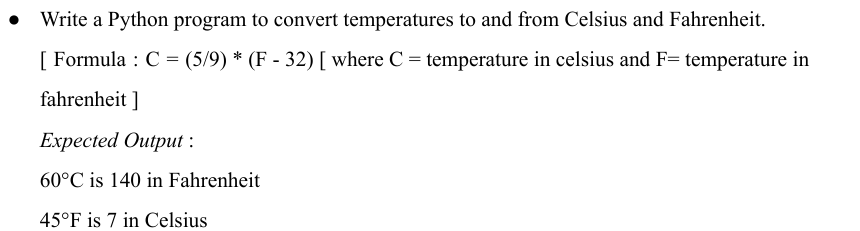
Numbers = []

for i in range(1500, 2701):

    if i%7 == 0 and i%5 == 0:

        Numbers.append(i)

print("Numbers that are Divisible by 7 and Multiples of 5: ",Numbers)



def convertCelsiustoFahrenheit(Celsius, Fahrenheit):

    Fahrenheit = ((Celsius \* 9)/5) + 32

    return Fahrenheit

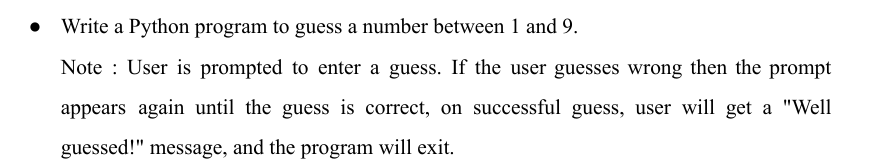
def convertFahrenheittoCelsius(Celsius, Fahrenheit):

    Celsius = ((5/9) \* (Fahrenheit - 32))

    return Celsius

print("60C is",int(convertCelsiustoFahrenheit(60, 45)),"in Fahrenheit")

print("45F is",int(convertFahrenheittoCelsius(60, 45)),"in Celsius")



import random

Number = random.randint(1, 9)

RightGuess = False

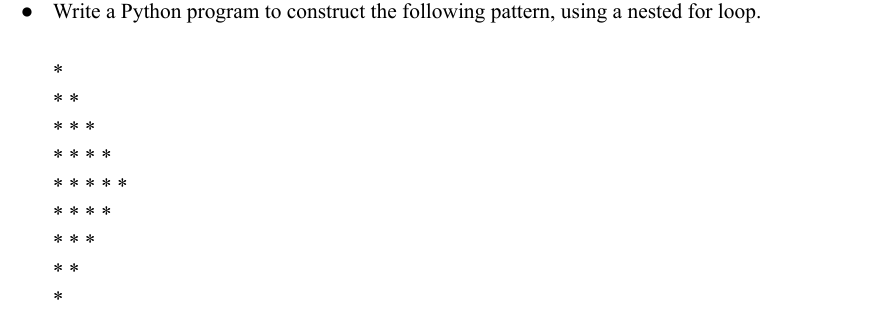
while not RightGuess:

    GuessNumber = int(input("Guess a Number Between 1 and 9:"))

    if GuessNumber == Number:

        print("Well Guessed!")

        exit()



for i in range(6):

    print("\* "\*i)

for i in range(4,0,-1):

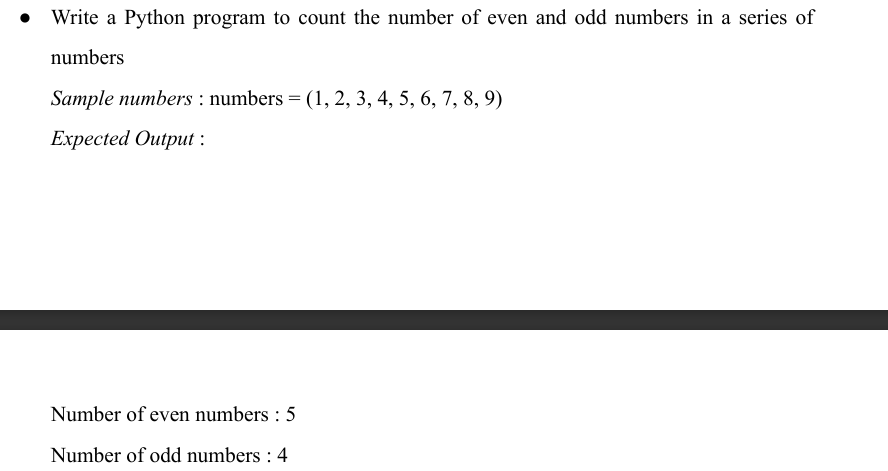
    print("\* "\*i)

print("\n")



Word = str(input("Enter a Word to reverse:"))

print(Word[::-1])



e,o=0,0

numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9)

for i in numbers:

  if i%2==0:

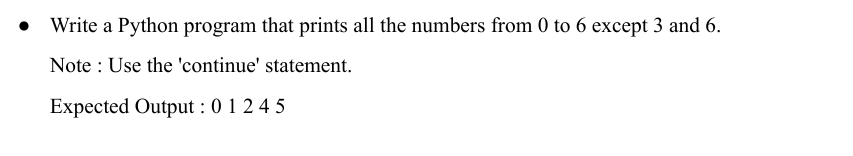
    e+=1

  else:

    o+=1

print(f"Number of even numbers : {e}")

print(f"Number of odd numbers : {o}")



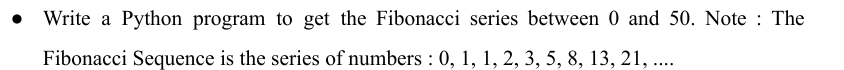
for i in range(0,7):

    if i == 3 or i == 6:

        continue

    else:

        print(i,end=" ")



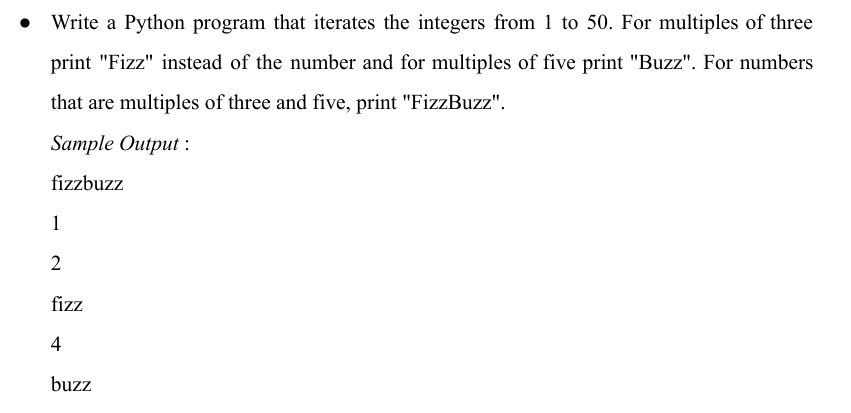
N1 = 0

N2 = 1

while N1<= 50:

    print(N1,end=" ")

    N1, N2 = N2, N1+N2



for i in range(51):

    if i%3 == 0 and i%5 ==0:

        print("fizzbuzz")

    elif i%3 == 0:

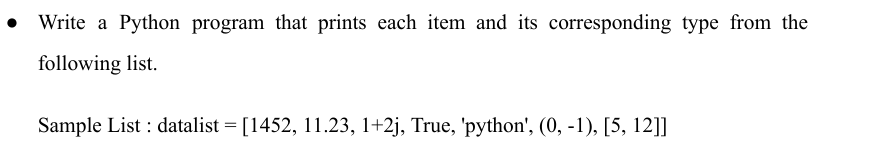
        print("fizz")

    elif i%5 == 0:

        print("buzz")

    else:

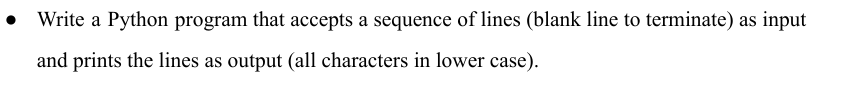
        print(i)



SampleDataList = [1452, 11.23, 1+2j, True, 'python', (0,-1),[5,12]]

for List in SampleDataList:

    print(List, type(List))



line=[]

while 1:

  x=input("Enter the string: ")

  line.append(x)

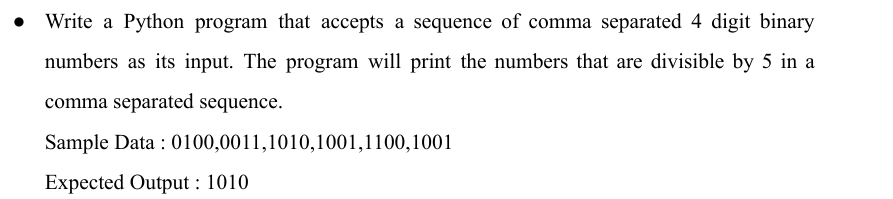
  if not x:

    break

print("Input is: ")

for i in line:

  print(i.lower(),end="")



binary\_numbers = input("Enter comma-separated 4-digit binary numbers: ")

binary\_list = binary\_numbers.split(',')

divisible\_by\_5 = []

for binary in binary\_list:

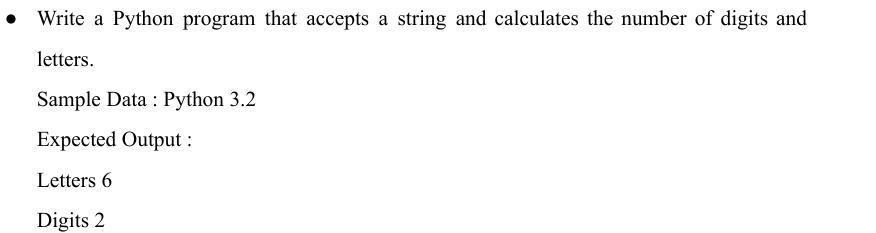
    decimal\_number = int(binary, 2)

    if decimal\_number % 5 == 0:

        divisible\_by\_5.append(binary)

result = ','.join(divisible\_by\_5)

print("Numbers divisible by 5:", result)



input\_string = input("Enter a String: ")

count\_letter = 0

count\_digits = 0

for character in input\_string:

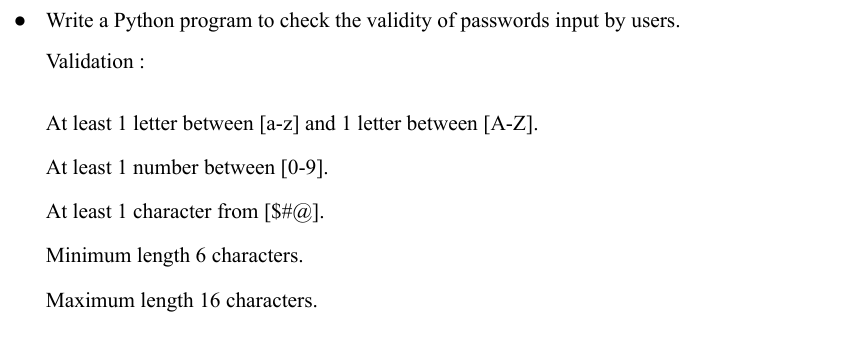
    if character.isalpha():

        count\_letter += 1

    elif character.isdigit():

        count\_digits += 1

print("Letters:", count\_letter, "\nDigits:", count\_digits)



password = input("Enter Your Password:")

count\_cap\_letter = 0

count\_small\_letter = 0

count\_digits = 0

count\_symbol = 0

length = len(password)

for character in password:

    if character.isupper():

        count\_cap\_letter += 1

    elif character.islower():

        count\_small\_letter += 1

    elif character.isdigit():

        count\_digits += 1

    else:

        count\_symbol += 1

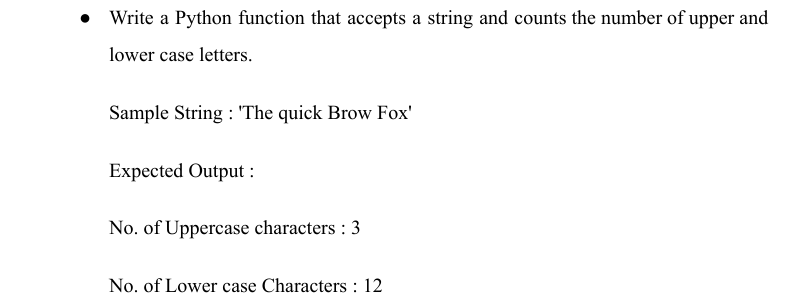
if count\_cap\_letter >= 1 and count\_small\_letter >= 1 and count\_digits >= 1 and count\_symbol >= 1 and 6 <= length <= 16:

    print("Valid Password.")

else:

    print("Invalid Password.")

LAB-4



def count\_upper\_lower(string):

    upper\_count = 0

    lower\_count = 0

    for char in string:

        if char.isupper():

            upper\_count += 1

        elif char.islower():

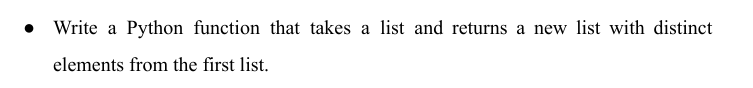
            lower\_count += 1

    print("No. of Uppercase characters:", upper\_count)

    print("No. of Lowercase characters:", lower\_count)

sample = str(input("Enter Any String:"))

count\_upper\_lower(sample)



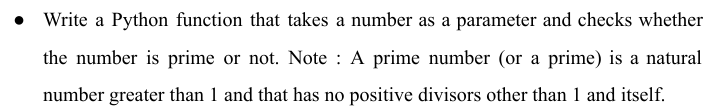
def distinct\_elements(input\_list):

    return list(set(input\_list))

original\_list = [1, 2, 2, 3, 4, 4, 5]

new\_list = distinct\_elements(original\_list)

print(new\_list)



def is\_prime(n):

  if n < 2:

    return False

  for i in range(2,n):

    if n % i == 0:

      return False

  return True

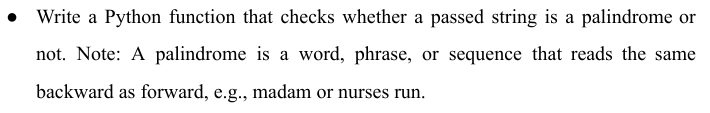
n=is\_prime(2)

if n== True:

 print(f"{n} is prime")

else:

  print(f"{n} is not prime")



def is\_palindrome(s):

    return s == s[::-1]

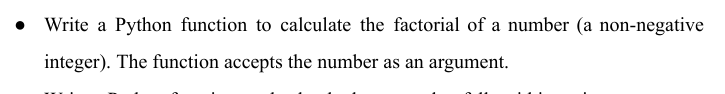
input\_string = input("Enter a string: ")

if is\_palindrome(input\_string):

    print("The string is a palindrome.")

else:

    print("The string is not a palindrome.")



def factorial(n):

    if n < 0:

        return "Factorial is not defined for negative numbers"

    elif n == 0:

        return 1

    else:

        result = 1

        for i in range(1, n + 1):

            result \*= i

        return result

num = int(input("Enter a non-negative integer: "))

print("Factorial of", num, "is", factorial(num))



def is\_within\_range(number, range\_start, range\_end):

    return range\_start <= number <= range\_end

num = float(input("Enter a number: "))

start = float(input("Enter the start of the range: "))

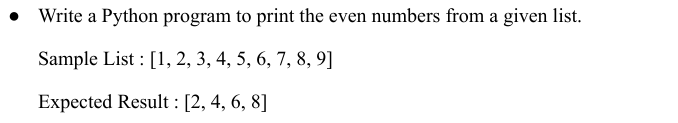
end = float(input("Enter the end of the range: "))

if is\_within\_range(num, start, end):

    print(num, "falls within the range [", start, ",", end, "]")

else:

    print(num, "does not fall within the range [", start, ",", end, "]")



def print\_even\_numbers(input\_list):

    even\_numbers = [num for num in input\_list if num % 2 == 0]

    return even\_numbers

sample\_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

result = print\_even\_numbers(sample\_list)

print("Even numbers from the sample list:", result)



def sum\_of\_digits(n):

  if n == 0:

    return 0

  else:

    return n % 10 + sum\_of\_digits(n // 10)

print(sum\_of\_digits(1234))

LAB-5



def main():

    total\_budget = float(input("Enter the total budget: $"))

    daily\_budget = float(input("Enter the total daily budget: $"))

    days\_of\_week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']

    daily\_expenses = []

    total\_expenses = 0

    exceeded\_budget = {}

    for day in days\_of\_week:

        expense = float(input(f"Enter expenses for {day}: $"))

        daily\_expenses.append(expense)

        total\_expenses += expense

        if expense > daily\_budget:

            exceeded\_budget[day] = expense

    print("\nExpense Summary:")

    print(f"Total Expenses for the week: ${total\_expenses:.2f}")

    print(f"Average Daily Expense: ${total\_expenses / 7:.2f}")

    if exceeded\_budget:

        print("Days where expenses exceeded the daily budget:")

        for day, expense in exceeded\_budget.items():

            print(f"{day}: ${expense:.2f}")

main()



my\_dict = {'apple': 10, 'banana': 5, 'orange': 8, 'grapes': 3}

sorted\_dict = sorted([(value, key) for (key, value) in my\_dict.items()])

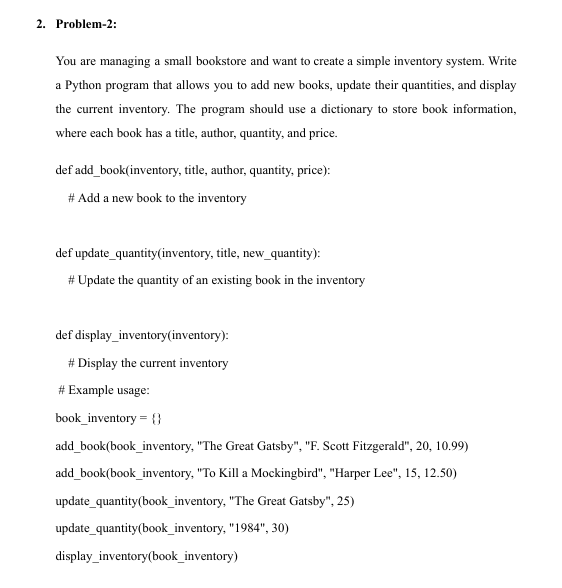
print("Ascending Order:")

for value, key in sorted\_dict:print(f"{key}: {value}")

sorted\_dict\_descending = sorted([(value, key) for (key, value) in my\_dict.items()], reverse=True)

print("Descending Order:")

for value, key in sorted\_dict\_descending:print(f"{key}: {value}")



def add\_book(inventory, title, author, quantity, price):

    inventory[title] = {'author': author, 'quantity': quantity, 'price': price}

def update\_quantity(inventory, title, new\_quantity):

    if title in inventory:

        inventory[title]['quantity'] = new\_quantity

    else:

        print(f"Book with title '{title}' not found in inventory.")

def display\_inventory(inventory):

    print("Current Inventory:")

    for title, details in inventory.items():

        print(f"Title: {title}, Author: {details['author']}, Quantity: {details['quantity']}, Price: ${details['price']:.2f}")

book\_inventory = {}

add\_book(book\_inventory, "The Great Gatsby", "F. Scott Fitzgerald", 20, 10.99)

add\_book(book\_inventory, "To Kill a Mockingbird", "Harper Lee", 15, 12.50)

update\_quantity(book\_inventory, "The Great Gatsby", 25)

update\_quantity(book\_inventory, "1984", 30)

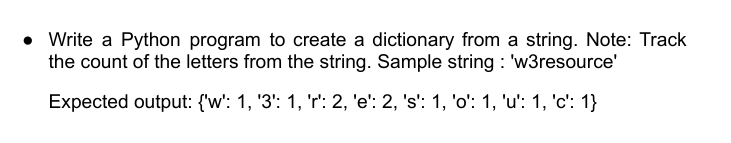
display\_inventory(book\_inventory)



python={}

for item in range(1,16):python[item]=item\*\*2

print(python)



w="w3resource"

fr={}

for i in w:

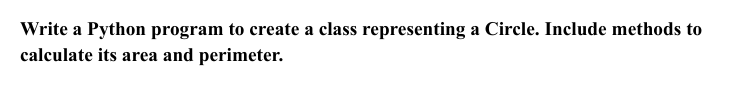
  if i in fr:

    fr[i]+=1

  else: fr[i]=1

print(fr)

LAB-16,17



import math

class Circle:

     def \_\_init\_\_(self, radius):

       self.radius = radius

     def calculate\_circle\_area(self):

       return math.pi \* self.radius\*\*2

     def calculate\_circle\_perimeter(self):

      return 2 \* math.pi \* self.radius

radius = float(input("Input the radius of the circle: "))

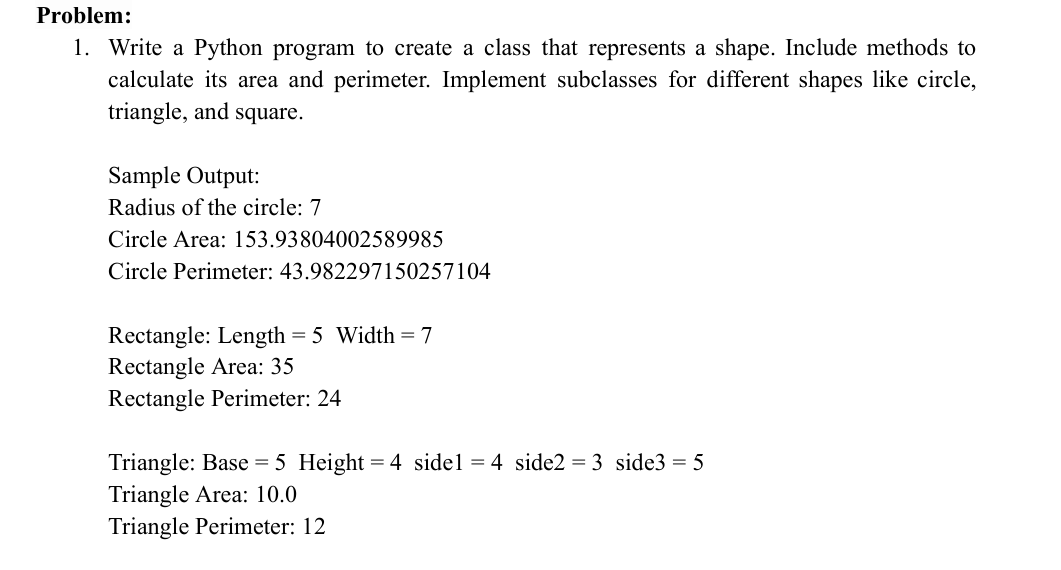
circle = Circle(radius)

area = circle.calculate\_circle\_area()

perimeter = circle.calculate\_circle\_perimeter()

print("Area of the circle:", area)

print("Perimeter of the circle:", perimeter)



import math

# Base class for all shapes

class Shape:

    def area(self):

        raise NotImplementedError("Subclasses must implement area method")

    def perimeter(self):

        raise NotImplementedError("Subclasses must implement perimeter method")

# Circle subclass

class Circle(Shape):

    def \_\_init\_\_(self, radius):

        self.radius = radius

    def area(self):

        return math.pi \* self.radius \*\* 2

    def perimeter(self):

        return 2 \* math.pi \* self.radius

# Rectangle subclass

class Rectangle(Shape):

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

        self.length = length

        self.width = width

    def area(self):

        return self.length \* self.width

    def perimeter(self):

        return 2 \* (self.length + self.width)

# Triangle subclass

class Triangle(Shape):

    def \_\_init\_\_(self, base, height, side1, side2, side3):

        self.base = base

        self.height = height

        self.side1 = side1

        self.side2 = side2

        self.side3 = side3

    def area(self):

        return 0.5 \* self.base \* self.height

    def perimeter(self):

        return self.side1 + self.side2 + self.side3

def main():

    # Circle

    circle\_radius = float(input("Enter the radius of the circle: "))

    circle = Circle(circle\_radius)

    print(f"\nCircle: Radius = {circle\_radius}")

    print(f"Circle Area: {circle.area()}")

    print(f"Circle Perimeter: {circle.perimeter()}")

    # Rectangle

    rectangle\_length = int(input("\nEnter the length of the rectangle: "))

    rectangle\_width = int(input("Enter the width of the rectangle: "))

    rectangle = Rectangle(rectangle\_length, rectangle\_width)

    print(f"\nRectangle: Length = {rectangle\_length}, Width = {rectangle\_width}")

    print(f"Rectangle Area: {rectangle.area()}")

    print(f"Rectangle Perimeter: {rectangle.perimeter()}")

    # Triangle

    triangle\_base = float(input("\nEnter the base of the triangle: "))

    triangle\_height = float(input("Enter the height of the triangle: "))

    triangle\_side1 = float(input("Enter the length of side 1 of the triangle: "))

    triangle\_side2 = float(input("Enter the length of side 2 of the triangle: "))

    triangle\_side3 = float(input("Enter the length of side 3 of the triangle: "))

    triangle = Triangle(triangle\_base, triangle\_height, triangle\_side1, triangle\_side2, triangle\_side3)

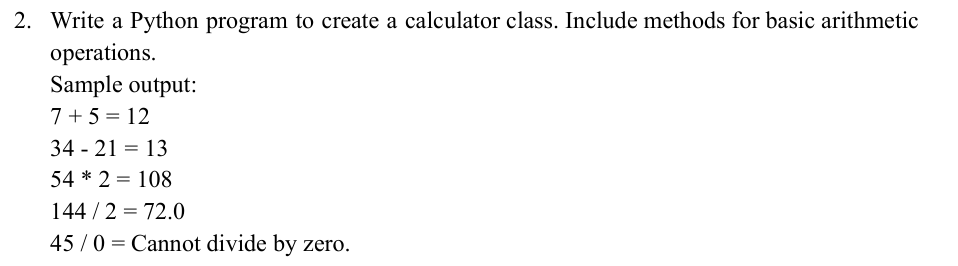
    print(f"\nTriangle: Base = {triangle\_base}, Height = {triangle\_height}, Side 1 = {triangle\_side1}, Side 2 = {triangle\_side2}, Side 3 = {triangle\_side3}")

    print(f"Triangle Area: {triangle.area()}")

    print(f"Triangle Perimeter: {triangle.perimeter()}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()



class Calculator:

    def add(self, Number1, Number2):

      return Number1 + Number2

    def sub(self, Number1, Number2):

      return Number1-Number2

    def mul(self, Number1, Number2):

      return Number1 \* Number2

    def div(self, Number1, Number2):

       if Number2 == 0:

        print("Cannot divide by zero.")

       else:

           return Number1 / Number2

calculate = Calculator()

print(f'7 + 5 ={calculate.add(7, 5)}')

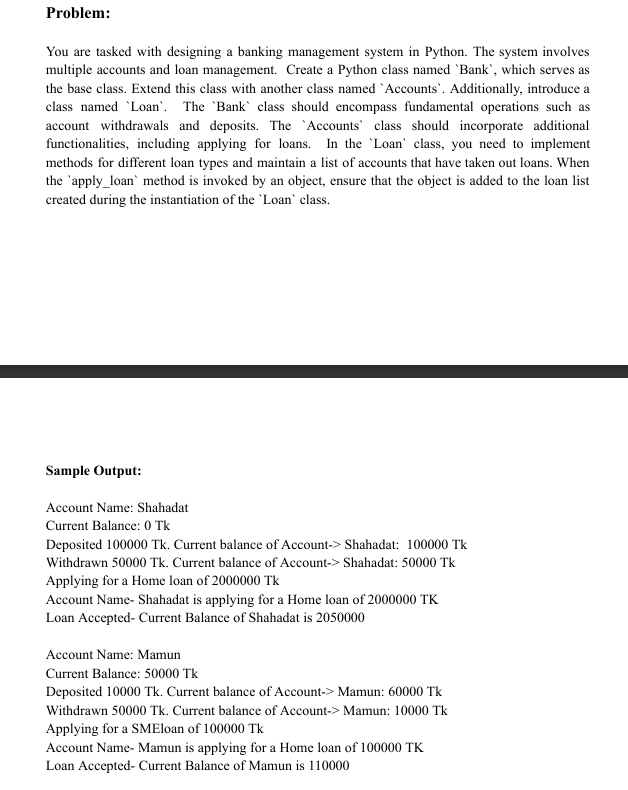
print(f'34- 21 ={calculate.sub(34, 21)}')

print(f'54 \* 2 ={calculate.mul(54, 2)}')

print(f'144 / 2 ={calculate.div(144, 2)}')

print(f'45/0 ={calculate.div(45, 0)}')

LAB-18



class Bank:

    def \_\_init\_\_(self, name, initial\_balance=0):

        self.name = name

        self.balance = initial\_balance

    def deposit(self, amount):

        self.balance += amount

        print(f"Deposited {amount} Tk. Current balance of Account-> {self.name}: {self.balance} Tk")

    def withdraw(self, amount):

        if self.balance >= amount:

            self.balance -= amount

            print(f"Withdrawn {amount} Tk. Current balance of Account-> {self.name}: {self.balance} Tk")

        else:

            print("Insufficient Balance.")

    def display(self):

        print(f"Account Name: {self.name}\nCurrent Balance: {self.balance} Tk")

class Accounts(Bank):

    def \_\_init\_\_(self, name, initial\_balance=0):

        super().\_\_init\_\_(name, initial\_balance)

    def apply\_loan(self, account, loan\_type, amount):

        print(f"Applying for a {loan\_type} loan of {amount} Tk")

        Loan().add\_loan(account, loan\_type, amount)

class Loan:

    loan\_accounts = []

    def add\_loan(self, account, loan\_type, amount):

        if account.balance >= 10000:

            self.loan\_accounts.append([account.name, loan\_type, amount])

            print(

                f"Account Name- {account.name} is applying for a {loan\_type} loan of {amount} TK")

            print(

                f"Loan Accepted- Current Balance of {account.name} is {account.balance + amount}")

        else:

            print(

                f"Account Name- {account.name} is applying for a {loan\_type} loan of {amount} TK")

            print(

                f"Loan Rejected- Current Balance of {account.name} is {account.balance}")

acc1 = Accounts("Shahadat")

acc1.display()

acc1.deposit(100000)

acc1.withdraw(50000)

acc1.apply\_loan(acc1, 'Home', 2000000)

print()

acc2 = Accounts("Mamun", 50000)

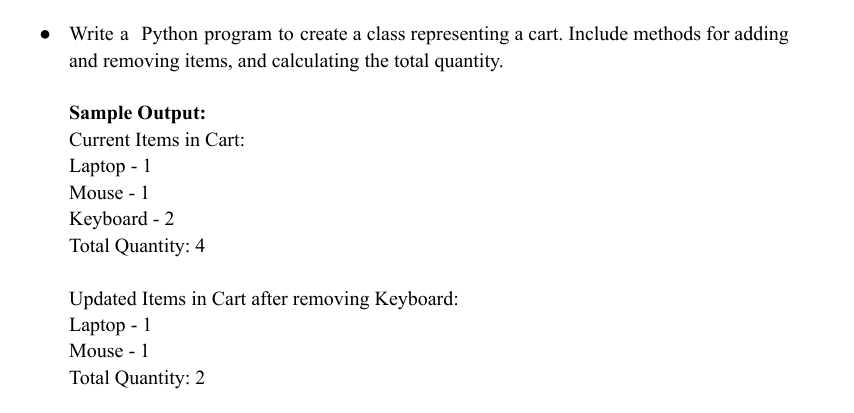
acc2.display()

acc2.deposit(10000)

acc2.withdraw(50000)

acc2.apply\_loan(acc2, 'SME', 100000)

LAB-19,20



class ShoppingCart:

    l={}

    def \_\_init\_\_(self, item\_list):

        self.l = item\_list

    def add(self, item\_name, item\_quantity):

        self.l[item\_name] = item\_quantity

    def remove(self, item\_name):

        if item\_name in self.l.keys():

            del self.l[item\_name]

    def calculate(self):

        self.total = 0

        for item, quantity in self.l.items():

            print(f"{item} - {quantity}")

            self.total += quantity

        print(f"Total Quantity: {self.total}")

obj = ShoppingCart({"Laptop": 1, "Mouse": 1})

obj.add("Keyboard", 2)

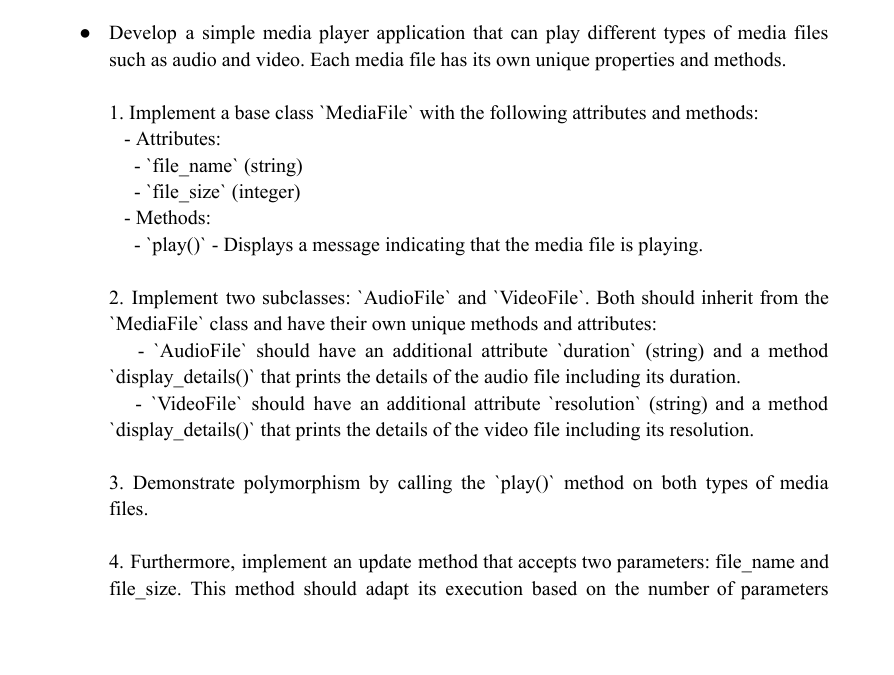
print("Current Items in Cart:")

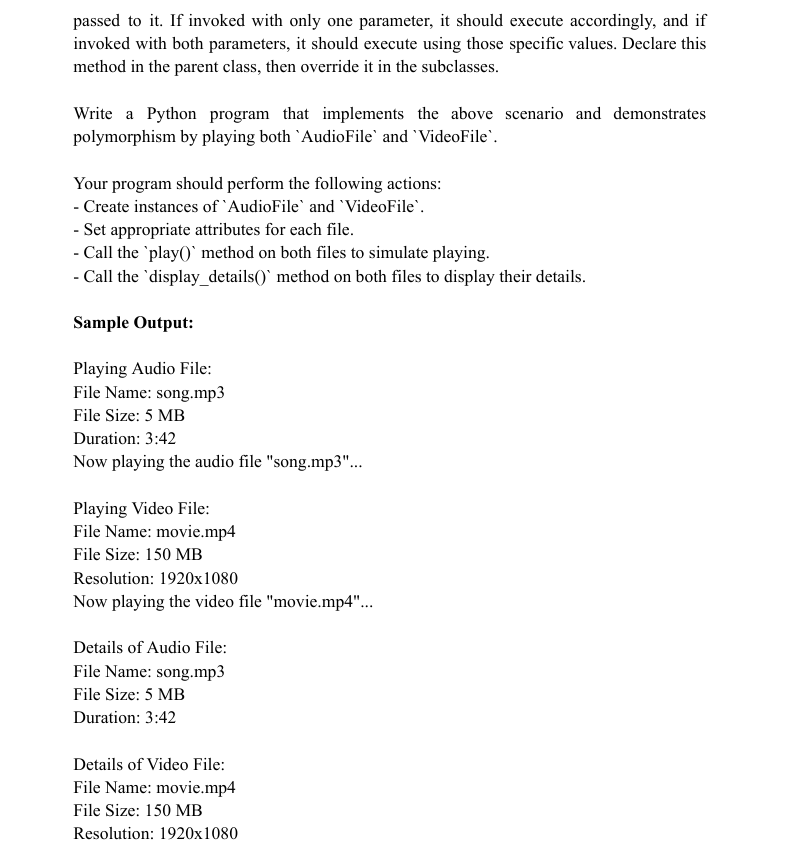
obj.calculate()

obj.remove("Keyboard")

print("Updated Items in Cart after removing Keyboard:")

obj.calculate()





class MediaFile:

    def \_\_init\_\_(self, file\_name, file\_size):

        self.file\_name = file\_name

        self.file\_size = file\_size

    def play(self):

        pass

    def display\_details(self):

        pass

    def update(self, file\_name=None, file\_size=None):

        if file\_name:

            self.file\_name = file\_name

        if file\_size:

            self.file\_size = file\_size

class AudioFile(MediaFile):

    def \_\_init\_\_(self, file\_name, file\_size, duration):

        super().\_\_init\_\_(file\_name, file\_size)

        self.duration = duration

    def play(self):

        print("Playing Audio File:")

        print(f"File Name: {self.file\_name}")

        print(f"File Size: {self.file\_size}")

        print(f"Duration: {self.duration}")

        print(f'Now playing the audio file "{self.file\_name}"...')

    def display\_details(self):

        print("Details of Audio File:")

        print(f"File Name: {self.file\_name}")

        print(f"File Size: {self.file\_size}")

        print(f"Duration: {self.duration}")

    def update(self, file\_name=None, file\_size=None):

        super().update(file\_name, file\_size)

        if file\_name or file\_size:

            print("Update occurred in audio file.")

class VideoFile(MediaFile):

    def \_\_init\_\_(self, file\_name, file\_size, resolution):

        super().\_\_init\_\_(file\_name, file\_size)

        self.resolution = resolution

    def play(self):

        print("Playing Video File:")

        print(f"File Name: {self.file\_name}")

        print(f"File Size: {self.file\_size}")

        print(f"Resolution: {self.resolution}")

        print(f'Now playing the video file "{self.file\_name}"...')

    def display\_details(self):

        print("Details of Video File:")

        print(f"File Name: {self.file\_name}")

        print(f"File Size: {self.file\_size}")

        print(f"Resolution: {self.resolution}")

    def update(self, file\_name=None, file\_size=None):

        super().update(file\_name, file\_size)

        if file\_name or file\_size:

            print("Update occurred in the video file.")

audio\_file = AudioFile("song.mp3", "5 MB", "3:42")

video\_file = VideoFile("movie.mp4", "150 MB", "1920x1080")

audio\_file.play()

print()

video\_file.play()

print()

audio\_file.display\_details()

print()

video\_file.display\_details()

print()

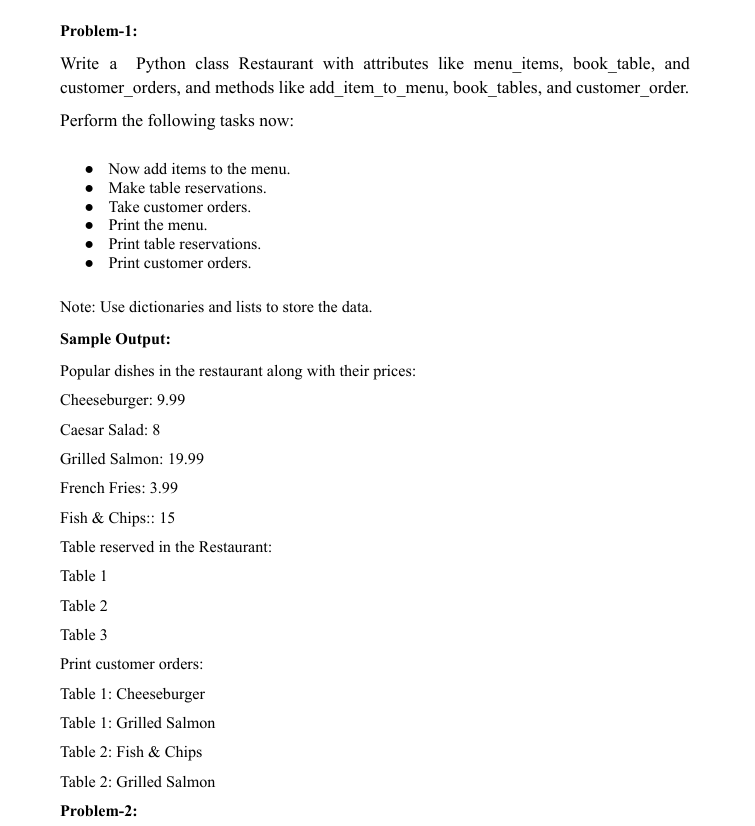
audio\_file.update("song23.mp3", 15)

audio\_file.display\_details()

video\_file.update("movie4.mp4", 180)

video\_file.display\_details()

LAB-21,22,23



class Restaurant:

    def \_\_init\_\_(self):

        self.menu\_items = {}

        self.book\_table = []

        self.customer\_orders = {}

    def add\_item\_to\_menu(self, item\_name, price):

        self.menu\_items[item\_name] = price

    def book\_tables(self, table\_number):

        if table\_number not in self.book\_table:

            self.book\_table.append(table\_number)

        else:

            print(f"Table {table\_number} is already reserved.")

    def customer\_order(self, table\_number, item\_name):

        if table\_number in self.book\_table:

            if table\_number not in self.customer\_orders:

                self.customer\_orders[table\_number] = []

            self.customer\_orders[table\_number].append(item\_name)

        else:

            print(f"Table {table\_number} is not reserved.")

    def print\_menu(self):

        print("Popular dishes in the restaurant along with their prices:")

        for item, price in self.menu\_items.items():

            print(f"{item}: {price}")

    def print\_table\_reservations(self):

        print("Table reserved in the Restaurant:")

        for table in self.book\_table:

            print(f"Table {table}")

    def print\_customer\_orders(self):

        print("Print customer orders:")

        for table, orders in self.customer\_orders.items():

            for order in orders:

                print(f"Table {table}: {order}")

restaurant = Restaurant()

restaurant.add\_item\_to\_menu("Cheeseburger", 9.99)

restaurant.add\_item\_to\_menu("Caesar Salad", 8)

restaurant.add\_item\_to\_menu("Grilled Salmon", 19.99)

restaurant.add\_item\_to\_menu("French Fries", 3.99)

restaurant.add\_item\_to\_menu("Fish & Chips", 15)

restaurant.book\_tables(1)

restaurant.book\_tables(2)

restaurant.book\_tables(3)

restaurant.customer\_order(1, "Cheeseburger")

restaurant.customer\_order(1, "Grilled Salmon")

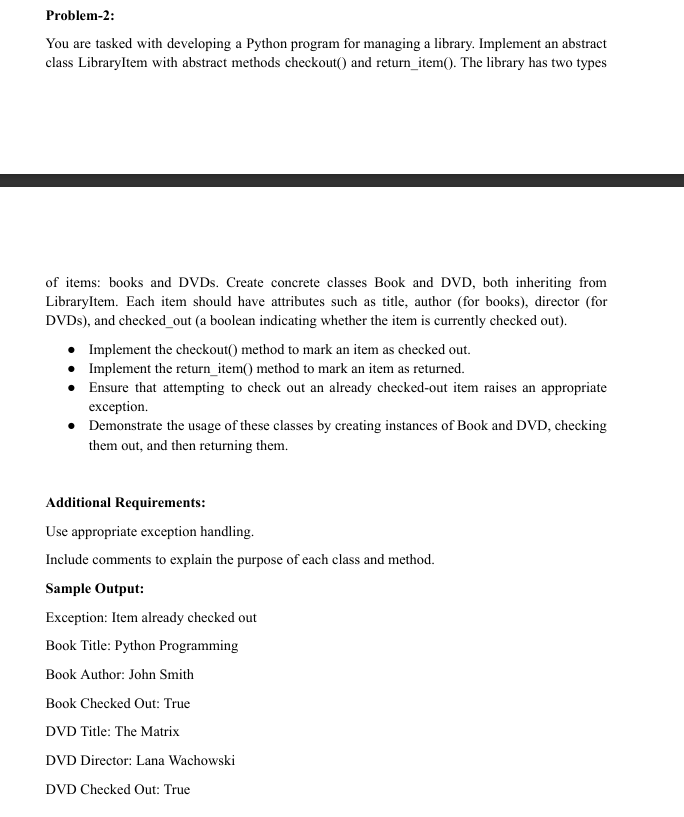
restaurant.customer\_order(2, "Fish & Chips")

restaurant.customer\_order(2, "Grilled Salmon")

restaurant.print\_menu()

restaurant.print\_table\_reservations()

restaurant.print\_customer\_orders()



from abc import ABC, abstractmethod

class LibraryItem:

  @abstractmethod

  def checkout(self):

    pass

  @abstractmethod

  def return\_item(self):

    pass

class Book(LibraryItem):

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

    self.title=title

    self.author=author

    self.checkout\_status=False

  def checkout(self):

      if not self.checkout\_status:

        self.checkout\_status=True

      else:

        raise ValueError("Items already checked out")

  def return\_item(self):

    if self.checkout\_status:

      print("Book returned successfully")

    else:

      raise ValueError ("Book's check out is not done")

  def details(self):

    print(f"Book Title: {self.title}\nBook Author: {self.author}\nBook Checked Out: {self.checkout\_status}")

class DVD(LibraryItem):

  def \_\_init\_\_(self,title,director):

    self.title=title

    self.director=director

    self.checkout\_status=False

  def checkout(self):     if not self.checkout\_status:

        self.checkout\_status=True

      else:

        raise ValueError("Items already checked out")

  def return\_item(self):

    if self.checkout\_status:

      print("DVD returned successfully")

    else:

      raise ValueError ("DVD check out is not done")

  def details(self):

    print(f"DVD Title: {self.title}\nDVD Director: {self.director}\nDVD Checked Out: {self.checkout\_status}")

try:

  book1=Book('Python Programming','John Smith')

  book1.checkout()

  book1.details()

  book1.return\_item()

  print()

  dvd1=DVD('The Matrix','Lana Wachowski')

  dvd1.checkout()

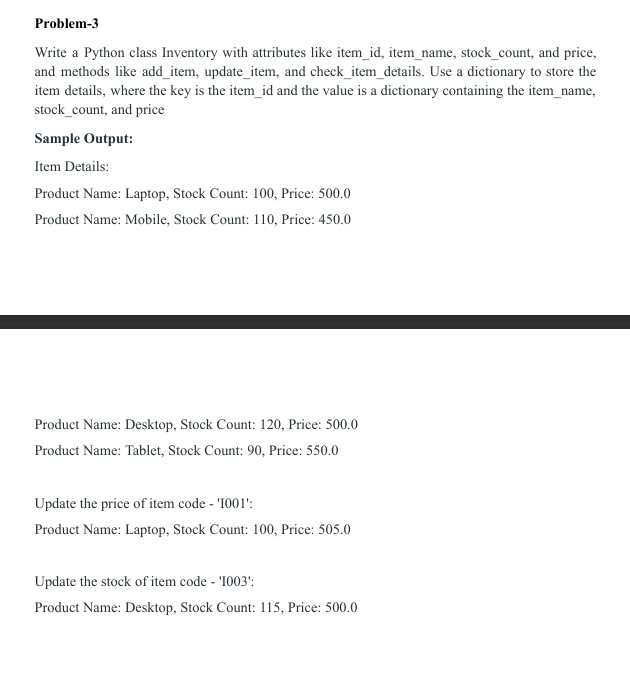
  dvd1.checkout()

  dvd1.details()

  dvd1.return\_item()

except Exception as e:

  print(f"Exception: {e}")



class Inventory:

  def \_\_init\_\_(self):

    self.items={}

  def add\_item(self, item\_id,item\_name,stock\_count,price):

    self.items[item\_id]=[item\_name,stock\_count,price]

  def update\_item(self,id,update\_attributes,x):

    if id in self.items.keys():

      if update\_attributes.lower()=='name':

        self.items[id][0]=x

      elif update\_attributes.lower()=='stock':

        self.items[id][1]=x

      elif update\_attributes.lower()=='price':

        self.items[id][2]=x

      else:

        print(f"Inventory has no attributes as {update\_attributes}")

  def details(self):

    print('Item Details:')

    for i in self.items.values():

      print(f"Product Name: {i[0]}, Stock Count: {i[1]}, Price: {i[2]}")

  def details\_by\_id(self,id):

    if id in self.items:

      print(f"Product Name: {self.items[id][0]}, Stock Count: {self.items[id][1]}, Price: {self.items[id][2]}")

obj1=Inventory()

obj1.add\_item('I001','Laptop',100,500.0)

obj1.add\_item('I002','Mobile',110,450.0)

obj1.add\_item('I003','Desktop',120,500.0)

obj1.add\_item('I004','Tablet',90,550.0)

obj1.details()

obj1.update\_item('I001','price',505.0)

print()

print("Update the price of item code - 'I001':")

obj1.details\_by\_id('I001')

obj1.update\_item('I003','stock',115)

print()

print("Update the stock of item code - 'I003':")

obj1.details\_by\_id('I003')