CS5590 APS - Python Programming
Lab Work Group

University of Missouri – Kansas City

LAB Report 1

Andrew Knowles # 17

Shah Rukh Khan # 16

Lee Turnock # 39

Tutor: Goudarzvand, Saria

Deadline: 2/13/2019

Current Date: 2/11/2019

I. Introduction

Laboratory problem 1 of the Course CS5590 is about Classes and Web Scrapping. The Laboratory Problem 1 consists of 6 problems, each of different nature, which cover the topics that have been taught till now in the class.

This report consists of all the codes and outputs of those codes that produce the results as desired by the instructor.

II. Objectives

The main objective of this Laboratory 2 is to test the familiarity of the students with all that is taught till now in CS5590 class. Including but not limited to Classes, Inheritance, Web Scrapping, Numpy and all of which is important towards the successful completion of the Deep Learning Tasks that are yet to come in the next lectures as we proceed towards the end of semester.

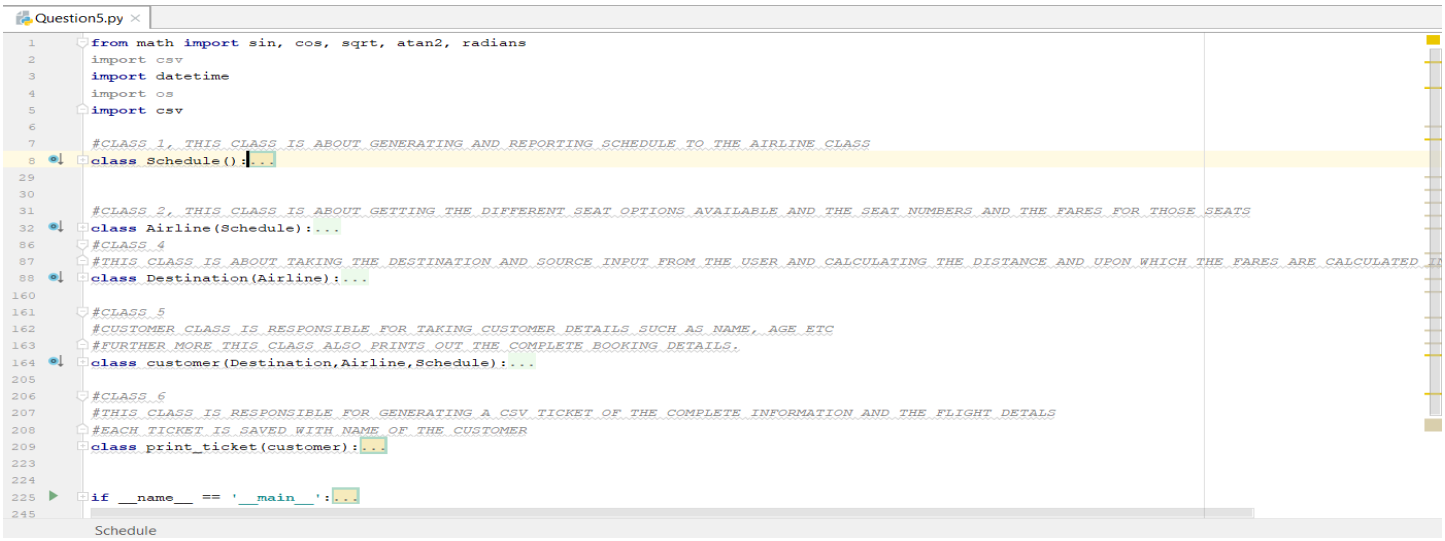
This reports objective is to Provide the code and detailed explanation of the codes written in response to each question of Lab Problem 1.

The following assignment focus on to make one familiar with python basic topics

III. Approaches/Methods

The Approaches and Methods are all those that have been directed by the supervisor. She gave us the complete guidelines as to which method to follow, these were different for each question.

Example she asked us to make 5 classes in Question 5. Please see below in the screen shot for the 5 Classes:



IV. Work/Code/Answers

There are 6 Questions and all have been done in order which they were mention in the main paper. Please see below the Questions and their Complete responses.

1. Write a program that computes the net amount of a bank account based a transaction log from console input. The transaction log format is shown as following:

```python
#Question 1: Bank Account class
class BankAccount:
    #Class variables
    name = ""
    totalAmount = 0

    #Constructor setting name and initial amount
    def __init__(self, name, initial_deposit):
        self.name = name
        self.totalAmount = initial_deposit

    #Deposit method
    def deposit(self, amount):
        self.totalAmount += amount

    #Withdrawal method
    def withdrawal(self, amount):
        self.totalAmount -= amount

    #For User Input to Account
    def User_deposit(self):
        self.totalAmount+=int(input("Enter Amount that you want to deposit: "))

    #For User withdrawal from Account
    def User_withdrawal(self):
        self.totalAmount-=int(input("Enter Amount that you want to withdraw: "))

#Bank Account uses for question 1
myAccount = BankAccount("Andrew", 0)
myAccount.User_deposit()
myAccount.User_withdrawal()
myAccount.deposit(300)
myAccount.deposit(250)
myAccount.withdrawal(100)
myAccount.deposit(50)
```

Figure 1 Question 1 Code

The Result of this code is Below:

```
C:\Python\python.exe "C:/Users/srkro/PycharmProjects/Lab Problem 1/Lab 1.py"
Enter Amount that you want to deposit: 1000
Enter Amount that you want to withdraw: 20
1480
```

Figure 2 Result Output of Question 1 Code

There are two option for each, Deposit and Withdrawal. You can either enter the amount in the Function in the code or you can ask the user to add the amount on run time.

2. Suppose you have a list of tuples as follows:

[(‘John’, (‘Physics’, 80)) , (‘ Daniel’, (‘Science’, 90)), (‘John’, (‘Science’, 95)), (‘Mark’,(‘Maths’, 100)), (‘Daniel’, (’History’, 75)), (‘Mark’, (‘Social’, 95))]

Create a dictionary with keys as names and values as list of (subjects, marks) in sorted order.

{ John : [(‘Physics’, 80), (‘Science’, 95)]
Daniel : [(’History’, 75), (‘Science’, 90)]
Mark : [(‘Maths’, 100), (‘Social’, 95)] }

The Complete code for this Question 2 is below:

```
#Question 2: Initialize our list of tuples and an empty dictionary
listofTuples = [( 'John', ('Physics', 80)), ( 'Daniel', ('Science', 90)), ('John', ('Science', 95)), ('Mark',('Maths', 100)), ('Daniel', ('History', 75)), ('Mark', ('Social', 95))]
dictofTuples = {}
#for every tuple in out list...
for tup in listofTuples:
    #If the name is not in our dictionary
    if tup[0] not in dictofTuples.keys():
        #add the tuple to the dictionary
        dictofTuples.setdefault(tup[0], [tup[1]])
    #Otherwise, add the class and score to the corresponding key value name
    else:
        dictofTuples[tup[0]] += [tup[1]]

print(dictofTuples)
```

Figure 3 Question 2 Code

The Result output for the Question 2 is below:

```
{'John': [('Physics', 80), ('Science', 95)], 'Daniel': [('Science', 90), ('History', 75)], 'Mark': [('Maths', 100), ('Social', 95)]}
```

Figure 4 Result output for Question 2 Code

One important thing that must be noted here is that Dictionary never maintains order, the same code might give different order for the same inputs in another environment.

3. Consider the following scenario. You have a list of students who are attending class "Python" and another list of students who are attending class "Web Application".

- Find the list of students who are attending both the classes.
- Also find the list of students who are not common in both the classes.

Print the both lists. Consider accepting the input from the console for list of students that belong to class “Python” and class “Web Application”.

Complete code for this Question is below:

```
#Question 3: initialize our list of students by class and 2 empty lists for common names, and not common names
pythonStudents = ['John', 'Mark', 'Sally']
webAppStudents = ['Jane', 'Mark', 'Jones', 'Gordon']
both = []
notCommon = []
temp=""
print("Current Python Students: ", pythonStudents)
while temp!="x":
    temp=str(input("Please Enter Name of Student in Python else enter x: "))
    if temp!="x":
        pythonStudents.append(temp)
print("Current Python Students: ", pythonStudents)
print("Current WebApp Students: ", webAppStudents)
temp=""
while temp!="x":
    temp=str(input("Please Enter Name of Student in webApp else enter x: "))
    if temp!="x":
        webAppStudents.append(temp)
print("Current WebApp Students: ", webAppStudents)
#for eeach person in our first class
for person in webAppStudents:
    #if that person is in the second class
    if person in pythonStudents:
        #add that person to our both list
        both.append(person)
#for each person in our first class
for person in webAppStudents:
    #if that person isn't in both class list
    if person not in both:
        #add that person to not common list
        notCommon.append(person)
#do the same for the second class
for person in pythonStudents:
    if person not in both:
        notCommon.append(person)
print("Students in Both Classes: ",both,"\n","Students Not Common in both Classes: ", notCommon)
```

Figure 5 Question 3 Complete Code

The Result of this Question is below:

```
Current Python Students:  ['John', 'Mark', 'Sally']
Please Enter Name of Student in Python else enter x: abc
Please Enter Name of Student in Python else enter x: cd
Please Enter Name of Student in Python else enter x: x
Current Python Students:  ['John', 'Mark', 'Sally', 'abc', 'cd']
Current WebApp Students:  ['Jane', 'Mark', 'Jones', 'Gordon']
Please Enter Name of Student in webApp else enter x: abc
Please Enter Name of Student in webApp else enter x: axd
Please Enter Name of Student in webApp else enter x: x
Current WebApp Students:  ['Jane', 'Mark', 'Jones', 'Gordon', 'abc', 'axd']
Students in Both Classes:  ['Mark', 'abc']
 Students Not Common in both Classes:  ['Jane', 'Jones', 'Gordon', 'axd', 'John', 'Sally', 'cd']
```

Figure 6 Result of Question 3 Code

This Question also required a condition where the user can add names of students in each class. So we have added the option, there are a few student names already in the list, user can add other names by himself.

4. Given a string, find the longest substring without repeating characters along with the length.

Input: "pwwkew"Output: wke,3

Complete code for Question 4 is below:

```
#Question 4:
#enter any string from the console
inputString = input("enter a string\n")

#grab first character in string
string = inputString[0]
#length will be at least 1
length = 1
#add the first character to our dictionary of string and length
stringDict = {1: inputString[0]}

#for each character in our input string starting with the second character
for x in range(1, len(inputString)):
    #if character is not the same as the last
    if inputString[x] != inputString[x-1]:
        #add that character to our string
        string += inputString[x]
        #increase the length by 1
        length+=1
        #otherwise
    else:
        #reset length to 1
        length = 1
        #reset search string to our current character
        string = inputString[x]
    #add our current string and length to our dictionary
    stringDict[length] = string

#print out the entry with the highest length in our dictionary
print(max(stringDict), stringDict[max(stringDict)])
```

Figure 7 Code for Question 4

The result for Question 4 code is below:

```
enter a string
HellowBowHowAreyou
15 lowBowHowAreyou
```

Figure 8 Result fo Question 4 Code

5. Write a python program to create any one of the following management systems.

1. Airline Booking Reservation System (e.g. classes Flight, Person, Employee, Passenger etc.)
2. Library Management System (eg: Student, Book, Faculty, Department etc.)

This Question was done in 2 Different ways by different members of the same group. Please see below the Complete codes:

```
#Question 5:
#Person class
#Class 1
class Person:...

#Employee class derived from Person
#Class 2
class Employee(Person):...

#Pilot class derived from Employee
#Class 3
class Pilot(Employee):...

#Passenger class derived from Person
#Class 4
class Passenger(Person):...

#Flight class
#Class 5
class Flight:...
```

Figure 9 Question 5 Code 1

```
#Question 5:
#Person class
#Class 1
class Person:
    #Person class variable
    name = ""

    #constructor setting name
    def __init__(self, name):
        self.name = name
#Employee class derived from Person
#Class 2
class Employee(Person):
    #Employee class variables
    id = 0
    dept = ""

    #constructor setting id and dept
    def __init__(self, id, dept):
        self.id = id
        self.dept = dept
        #super call to Person class
        super(Person, self).__init__()
```

Figure 10 1st two Classes of Question 5

```
#Pilot class derived from Employee
#Class 3
class Pilot(Employee):
    #Pilot class variables
    license_number = 0

    #constructor setting license
    def __init__(self, license):
        self.license_number = license
        #super call to Employee class
        super(Employee, self).__init__()
#Passenger class derived from Person
#Class 4
class Passenger(Person):
    #Person class variables
    flight_number = 0
    __credit_card_num = 0

    #constructor setting flight number
    def __init__(self, flight_number):
        self.flight_number = flight_number
        #super call to Person class
        super(Person, self).__init__()

    #add card number method
    def add_card(self, card):
        self.__credit_card_num = card
```

Figure 11 Class 3 and Class 4 of Question 5

```python
#Flight class
#Class 5
class Flight:
    #Flight class variables
    number = 0
    pilot = ""
    passengers =[]

    #constructor setting number and pilot
    def __init__(self, number, pilot):
        self.number = number
        self.pilot = pilot

    #method adding passenger to list of passengers
    def add_passenger(self, passenger):
        self.passengers.append(passenger)
```

Figure 12 5th Class of Question 5

```python
from math import sin, cos, sqrt, atan2, radians
import csv
import datetime
import os
import csv

#CLASS 1, THIS CLASS IS ABOUT GENERATING AND REPORTING SCHEDULE TO THE AIRLINE CLASS
class Schedule():
    FlightID=""
    Date=datetime.datetime.now()+datetime.timedelta(hours=7)
    def __int__(self,depart=datetime.datetime.now()):
        super(Schedule, self).__int__()
        super(customer, self).__int__()
        self.date=depart
        self.day=0
        self.month=0
        self.year=0
        self.flightID=""
    def get_date(self):
        self.day=int(input("Enter Date of Flight: "))
        self.month=int(input("Enter Month of Flight: "))
        self.year=int(input("Enter Year of Flight: "))
        Schedule.Date=\
        Schedule.Date.replace(year=self.year,month=self.month,day=self.day)
        self.FlightID=str(Schedule.Date.time().hour)
    def print_det(self):
        print(self.FlightID)
        print(self.Date)


#CLASS 2, THIS CLASS IS ABOUT GETTING THE DIFFERENT SEAT OPTIONS AVAILABLE AND THE SEAT NUMBERS AND THE FARES FOR THOSE SEATS
class Airline(Schedule):
    seat = [["Economy Class", 100], ["Economy Plus", 100], ["Business Class", 100]]
    economy_class = "Economy Class"
    economy_plus = "Economy Plus"
    business_class = "Business Class"
    def __init__(self):
        super(Schedule, self).__init__()
        self.fares = [["Economy Class",500],["Economy Plus",1200],["Business Class",2000]]
        self.price=0
        self.seat_type=""
        self.AD=Schedule.FlightID
#CHECKS IF A SEAT IS AVAILABLE
    def is_available(self, booking_class):
        for i in self.seat:
            if booking_class in i:
                return self.seat[self.seat.index(i)][1] > 0
#IF SEAT IS AVAILABLE IT OKAY'S THE TRANSACTION AND REDUCES 1 SEAT FROM THE LOT
    def make_reservation(self, seat_class):
        if self.is_available(seat_class):
            for i in Airline.seat:
                if seat_class in i:
                    Airline.seat[Airline.seat.index(i)][1] = \
                    Airline.seat[Airline.seat.index(i)][1] - 1
                    self.price+=int(self.fares[self.seat.index(i)][1])
                    break
#IF RESERVATION IS CANCELLED, THIS CREDITS BACK THE SEAT WHICH WAS DEDUCTED
    def cancel_reservation(self, seat_class):
        for i in self.seat:
            if seat_class in i:
                if Airline.seat[Airline.seat.index(i)][1]<100:
                    Airline.seat[Airline.seat.index(i)][1] = \
                    Airline.seat[Airline.seat.index(i)][1] + 1
                break
#HERE OPTIONS ARE GIVEN TO THE CUSTOMER WHICH SEAT HE WANTS TO TAKE DEPENDING UPON THE FARES
    def Fares(self,distance):
        print("Please Select from below available Seats and their Fares in USD: ")
        for i in range(0,3):
            if self.is_available(Airline.seat[i][0]):
```

```
                print(i ,Airline.seat[i][0]," ",Airline.seat[i][1]," Seats available each priced at: $",
(self.fares[i][1]+(distance/(4-i))))
        i=int(input("Please Enter the Seat Class number you want to reserve: "))
        if(i==0):
            self.price+=distance/4
            self.seat_type=self.economy_class
            return self.economy_class

        elif(i==1):
            self.price+=distance/3
            self.seat_type=self.economy_plus
            return self.economy_plus

        elif(i==2):
            self.price+=distance/2
            self.seat_type=self.business_class
            return self.business_class
#CLASS 4
#THIS CLASS IS ABOUT TAKING THE DESTINATION AND SOURCE INPUT FROM THE USER AND CALCULATING THE DISTANCE AND UPON WHICH THE FARES
ARE CALCULATED IN AIRPLANE CLASS
class Destination(Airline):
    #__distancex is a private variable and is not accessible from any other class
    __distancex=0
    def __init__(self):
        super(Airline, self).__init__()
        self.source_city="Kansas City"
        self.destination ="New York"
        self.source_cood="0"
        self.destination_cood="0"
        self.distance=0
    def Departure_date(self):
        self.get_date()
#WE READ FROM A CSV FILE THE LIST OF CITIES THAT ARE AVAILABLE FOR TRAVELLING TO AND FROM
    def Departure(self):
        print(" Select from below list, your Departure City: ")
        with open('Destination.csv', 'r') as csvfile:
            source = []
            self.source_city = ""
            self.source_cood = ""
            spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
            id = 0
            for row in spamreader:
                source.append([id, ','.join(row)])
                id += 1
            source.pop(0)
            for row in source:
                print(row)
            id = int(input("Source ID : "))
            for row in source:
                if (int(row[0]) == id):
                    self.source_city = row[1].split(",")[0]
                    self.source_cood = row[1].split(",")[-1]
            print(self.source_city)
            #print(source_cood)
    def Destination(self):
        print(" Select from below list, your Departure City: ")
        with open('Destination.csv', 'r') as csvfile:
            source = []
            self.destination = ""
            self.destination_cood = ""
            spamreader = csv.reader(csvfile, delimiter=',', quotechar='|')
            id = 0
            for row in spamreader:
                source.append([id, ','.join(row)])
                id += 1
            source.pop(0)
            for row in source:
                print(row)
            id = int(input("Source ID : "))
            for row in source:
                if (int(row[0]) == id):
                    self.destination = row[1].split(",")[0]
                    self.destination_cood = row[1].split(",")[-1]
            print(self.destination)
            #print(destination_cood)

#THIS FUNCTION CALCULATES THE DISTANCE BETWEEN THE TWO SELECTED CITIES. THIS DISTANCE IS THEN LATER ADDED TO THE FARE RATE
    def Distance(self):
        R = 6373.0
        lat1 = radians(float(self.source_cood.split(" ")[1][:-1]))
        lon1 = radians(float(self.source_cood.split(" ")[0][:-1]))
        lat2 = radians(float(self.destination_cood.split(" ")[1][:-1]))
        lon2 = radians(float(self.destination_cood.split(" ")[0][:-1]))
        dlon = lon2 - lon1
        dlat = lat2 - lat1

        a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) ** 2
        c = 2 * atan2(sqrt(a), sqrt(1 - a))

        self.distance = R * c
```

```
        Destination.__distancex=self.distance
        print("Distance: ", Destination.__distancex)

#CLASS 5
#CUSTOMER CLASS IS RESPONSIBLE FOR TAKING CUSTOMER DETAILS SUCH AS NAME, AGE ETC
#FURTHER MORE THIS CLASS ALSO PRINTS OUT THE COMPLETE BOOKING DETAILS.
class customer(Destination,Airline,Schedule):
    def __init__(self,name="default",age="999",passport="NA"):
        super(Airline, self).__init__()
        super(Schedule,self).__init__()
        super(Destination, self).__init__()
        self.name=name
        self.age=age
        self.passport=passport

    def get_detail(self):
        self.name=input("Enter Name: ")
        self.age=int(input("Enter Age: "))
        self.passport=input("Enter Passport Number: ")
        self.Departure()
        self.Destination()
        self.Departure_date()
        self.Distance()
        Seat=self.Fares(self.distance)
        self.make_reservation(Seat)
    def print_details(self):
        print("Seat Type: ",self.seat_type)
        print("Customer Name: ",self.name)
        print("Customer Age: ",self.age)
        print("Customer Passport: ",self.passport)
        print("Departure City: ",self.source_city)
        print("Destination City: ",self.destination)
        print("Total Cost: $",self.price)
        print("AirPlane ID: ", (self.source_city[0:2]+self.FlightID+self.destination[0:2]) )
        print("Departure Time: ", self.Date)
    def save_ticket(self):
        self.ticket=[]
        self.ticket.append(["Seat Type: ",self.seat_type])
        self.ticket.append(["Customer Name: ",self.name])
        self.ticket.append(["Customer Age: ",self.age])
        self.ticket.append(["Customer Passport: ",self.passport])
        self.ticket.append(["Departure City: ",self.source_city])
        self.ticket.append(["Destination City: ",self.destination])
        self.ticket.append(["Total Cost: $",self.price])
        self.ticket.append(["AirPlane ID: ", (self.source_city[0:2]+self.FlightID+self.destination[0:2]) ])
        self.ticket.append(["Departure Time: ", self.Date])
        return self.ticket

#CLASS 6
#THIS CLASS IS RESPONSIBLE FOR GENERATING A CSV TICKET OF THE COMPLETE INFORMATION AND THE FLIGHT DETALS
#EACH TICKET IS SAVED WITH NAME OF THE CUSTOMER
class print_ticket(customer):
    def __int__(self):
        super(customer,self).__int__()
    def print_it(self):
        self.get_detail()
        self.print_details()
        self.ticketx=self.save_ticket()
        csv.register_dialect('myDialect',quoting=csv.QUOTE_ALL,skipinitialspace=True)
        with open(str(self.name)+".csv", 'w') as f:
            writer = csv.writer(f, dialect='myDialect')
            for row in self.ticketx:
                writer.writerow(row)

        f.close()
```

Complete Code for Question 5 2nd way

This Code basically is done in 2 ways, The 2nd way Asks for Ticket details as following

Customer name:
Customer Age:
Customer Passport:
Customer Depart and Destination: (I have placed list of cities in a Csv file and the program reads from there)
Customer departure date: (Asks for day, month and year)
Asks Customer for a Seat type: (Economy, Economy Plus or Business)

Fares of each are calculated based on the distance of source and destination:

Results are below:

```
Enter Name: Abraham
Enter Age: 24
Enter Passport Number: VJ41092342
 Select from below list, your Departure City:
[1, 'New York,New York,40.6643N 73.9385W']
[2, 'Los Angeles,California,34.0194N 118.4108W']
[3, 'Chicago,Illinois,41.8376N 87.6818W']
[4, 'Houston,Texas,29.7805N 95.3863W']
[5, 'Philadelphia,Pennsylvania,40.0094N 75.1333W']
[6, 'Phoenix,Arizona,33.5722N 112.0880W']
[7, 'San Antonio,Texas,29.4724N 98.5251W']
[8, 'San Diego,California,32.8153N 117.1350W']
[9, 'Dallas,Texas,32.7757N 96.7967W']
[10, 'San Jose,California,37.2969N 121.8193W']
[11, 'Austin,Texas,30.3072N 97.7560W']
[12, 'Jacksonville,Florida,30.3370N 81.6613W']
[13, 'Indianapolis,Indiana,39.7767N 86.1459W']
[14, 'San Francisco,California,37.7751N 122.4193W']
```

Figure 13 Entering Customer Details

```
[135, 'Everett,Washington,48.0033N 122.1742W']
[136, 'Wichita Falls,Texas,33.9067N 98.5259W']
Source ID : 65
Visalia
Enter Date of Flight: 12
Enter Month of Flight: 04
Enter Year of Flight: 2019
Distance:  1612.232303903403
Please Select from below available Seats and their Fares in USD:
0 Economy Class    100  Seats available each priced at: $ 903.0580759758507
1 Economy Plus    100  Seats available each priced at: $ 1737.410767967801
2 Business Class    100  Seats available each priced at: $ 2806.1161519517013
Please Enter the Seat Class number you want to reserve: 2
Seat Type:  Business Class
Customer Name:  Abraham
Customer Age:  24
Customer Passport:  VJ41092342
Departure City:  Denver
Destination City:  Visalia
Total Cost: $ 2806.1161519517013
AirPlane ID:  De21Vi
Departure Time:  2019-04-12 21:06:59.150733
Seat Type:  Business Class
Customer Name:  Abraham
Customer Age:  24
Customer Passport:  VJ41092342
Departure City:  Denver
Destination City:  Visalia
Total Cost: $ 2806.1161519517013
AirPlane ID:  De21Vi
Departure Time:  2019-04-12 21:06:59.150733
```

Figure 14 Complete Ticket printed at the end

The code also saves a copy of the Ticket and saves it in the code directory by name if the customer:

Abraham	2/11/2019 2:07 PM	Microsoft Excel Co...	1 KB
Destination	2/9/2019 4:54 PM	Microsoft Excel Co...	6 KB
Lab 1	2/11/2019 12:51 PM	Python File	8 KB
Question5	2/11/2019 2:06 PM	Python File	10 KB
Table_Info	2/11/2019 12:43 PM	File	1 KB

Figure 15 Ticket of Abraham

The code contains following

Prerequisites:

a. Your code should have at least five classes **(Shown in pictures above)**
b. Your code should have _init_ constructor in all the classes **(Shown in pictures above)**
c. Your code should show inheritance at least once **(Shown in pictures above)**
d. Your code should have one super call **(Shown in pictures above)**
e. Use of self is required **(Shown in pictures above)**
f. Use at least one private data member in your code **(Shown in pictures above)**
g. Use multiple Inheritance at least once **(Shown in pictures above)**
h. Create instances of all classes and show the relationship between them **(Shown in pictures above)**

6. Program a code which download a webpage contains a table using Request library, then parse the page using Beautiful soup library. You should save the information about the states and their capitals in a file.

Complete Code for this Question is below:

```
#Question 6:
#input search term
search = input("Enter Wikipedia search term\n")
#replace spaces with underscores
search.replace(' ', '_')
#open that Wikipedia link
with urllib.request.urlopen('https://en.wikipedia.org/wiki/' + search) as response:
    #open link for parsing and initialize list of data
    soup = BeautifulSoup(response, 'html.parser')
    data = []
    #find all occurrences of tables
    tables = soup.findAll('table')
    #open a file for writing
    file = open("Table_Info", 'w')
    #for every table in out collection of tables...
    for table in tables:
        #find the body of the table
        table_body = table.find('tbody')
        #find all rows in the table
        rows = table_body.findAll('tr')
        #for every row in our current table
        for row in rows:
            #find all the column values in our row
            cols = row.find_all('td')
            #clean the column entry
            cols = [ele.text.strip() for ele in cols]
            #add that column entry to it's row and that row to our list of data
            data.append([ele for ele in cols if ele])

    #for all rows in our list of data...
    for x in range(0, len(data)):
        #for all column entries in our row...
        for y in range(0, len(data[x])):
            #write that column entry followed by a newline for formatting
            file.write(data[x][y] + "\n")
        print(data[x])
```

Figure 16 Code for Question 6

The result output for the Code is below:

```
Enter Wikipedia search term
Python
['Look up Python\xa0or python in Wiktionary, the free dictionary.']
['Disambiguation page providing links to topics that could be referred to by the same search termThis disambiguation page lists  articles associated with the title Python.  If an internal link led

Process finished with exit code 0
```

Figure 17 output on Console for Question 6

.idea	2/11/2019 12:51 PM	File folder	
Destination	2/9/2019 4:54 PM	Microsoft Excel Co...	6 KB
Lab 1	2/11/2019 12:51 PM	Python File	8 KB
Question5	2/9/2019 10:17 PM	Python File	10 KB
Table_Info	2/11/2019 12:43 PM	File	1 KB

Figure 18 Output File for saving table in Question 6

All Questions have been duly answered and their codes are attached in screen shots and also uploaded the the Git Repository

V. Datasets (if applicable)

There is no extra data except for the names of cities that have been stored in a CSV file that will be uploaded to the repository for smooth functioning of code if someone wishes to download and run or make further improvements. Please site this repository incase the code is used in development or commercial projects.

Destination.csv file in GIT Repository

VI. Parameters

No extra parameters except for those required in each question have been used. They do not require special mention as they have been explained in comments section of each code and in the workflow section

VII. Evaluation & Discussion

The codes for all the questions are complete to the best of our knowledge and fulfil the criteria and guidelines mentioned by the tutor to fulfil while coding. Naming and coding conventions have been followed to the best of out knowledge.

VIII. Conclusion

This Lab report and all the codes are our own work and has not been copied or taken from anywhere. The resources are all our own and are not liable to any type of copy right violation.
The assignment and the lab problem is complete with Detailed codes and results of all 6 questions. All guidelines have been followed and we are now able to deduce and conclude that the main purpose of the complete assignment has been met.

LAB Submission Guidelines (for both In Class and Online students):

1. LAB submission is in a group of three students.
2. Submit your source code and documentation to GitHub and represent the work through wiki page properly (submit your screenshots as well. The screenshot should have both the code and the output)
3. Comment your code appropriately
4. Video Submission (2 – 3 min video showing the demo of the LAB, with brief voice over on the code explanation)
5. Submit **only** report at Turnitin in UMKC blackboard
6. Remember that similarity score should be less than **15%**
7. Use this link to submit your LAB#: https://goo.gl/forms/l6q2rzkKCEGTigfp2
8. Report should include below details

I. Introduction
II. Objectives
III. Approaches/Methods
IV. Workflow
V. Datasets (if applicable)
VI. Parameters
VII. Evaluation & Discussion
VIII. Conclusion