CSC 157

Name \_\_James Aniciete\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date \_\_4/21/2020\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lab No. \_\_\_12\_\_\_\_

Purpose of the Lab Activity

**The purpose of this activity is to create a program that lets a user search for flights based on inputted variables.**

Source Code

**Class file:**

import pickle

# class file name: flights.py

class flights(object) :

""" description of class """

flightCount = 0

def \_\_init\_\_(self, date, time, destination, cost) :

self.date = date

self.time = time

self.destination = destination

self.cost = cost

flights.flightCount += 1

""" end class description """

# class file includes these flights objects

myFlight = [

flights("12/13", 12.00, "SFO", 170.00),

flights("12/13", 15.00, "SAT", 220.00),

flights("12/14", 13.00, "JFK", 150.00),

flights("12/15", 11.00, "LAX", 190.00),

flights("12/15", 12.00, "JNU", 550.00),

flights("12/16", 08.00, "MSY", 320.00),

flights("12/16", 10.00, "CLE", 70.00)

]

# save the data to a file

pickle.dump(myFlight, open("target.p", "wb"))

**Application file:**

# Programmer: James Aniciete

# Course No.: CSC 157

# Lab No.: 12

# Date: 4/18/2020

import pickle

from flights import \* # imports the flights class

# store the list of flights in a variable named fl

fl = pickle.load(open("target.p", "rb"))

# initialize variable to terminate program

response = "Y"

while (response == "Y" or response == "y") :

# ask user to choose a sorting option

answer = input("How would you like to sort the available flights?"

"\n1) Date"

"\n2) Destination"

"\n3) Cost\n")

# sort by date

if (answer == "1") :

d = input("Please enter a date (MM/YY)\n")

for i in range (6) :

if (d == fl[i].date) :

print (

fl[i].date, "%.2f" % fl[i].time,

fl[i].destination, "$", "%.2f" % fl[i].cost)

# sort by US airport code

elif (answer == "2") :

dest = input("Please enter a US airport code\n")

for i in range (6) :

if (dest == fl[i].destination) :

print (

fl[i].date, "%.2f" % fl[i].time,

fl[i].destination, "$", "%.2f" % fl[i].cost)

# sort by cost

elif (answer == "3") :

c = float(input("Please enter a cost in US dollars\n"))

for i in range (6) :

if (c >= fl[i].cost) :

print (

fl[i].date, "%.2f" % fl[i].time,

fl[i].destination, "$", "%.2f" % fl[i].cost)

# otherwise, display that an invalid response was entered

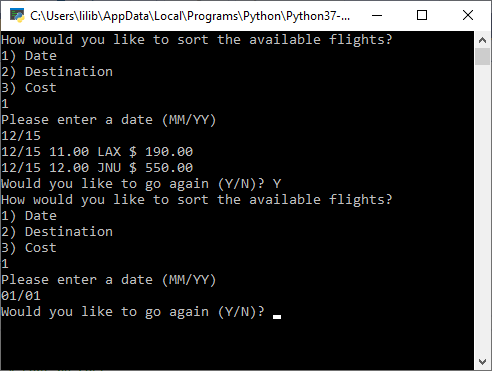
else :

print ("invalid response")

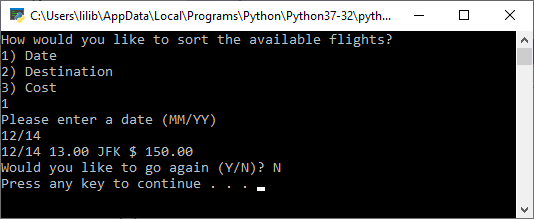
response = input("Would you like to go again (Y/N)? ")

Snippet(s) of Output(s) – input in flights.py followed by input not in flights.py

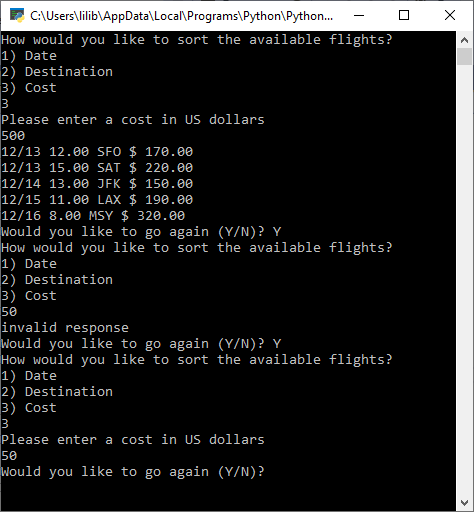
**Sorted by date:**



**Sorted by U.S. airport code:**



**Sorted by cost:**



Modified Source Code

**Class file:**

import pickle

# class file name: flights.py

class flights(object) :

""" description of class """

flightCount = 0 # will be the end range of for loops

def \_\_init\_\_(self, date, dow, time, destination, airline, cost) :

self.date = date

self.dow = dow

self.time = time

self.destination = destination

self.airline = airline

self.cost = cost

flights.flightCount += 1

""" end class description """

# class file includes these flights objects

# list of 12 flights from 12/9/2020 to 12/16/2020

myFlight = [

flights("12/12", "Saturday", "07.00", "JFK", "Spirit", 35.00), # NYC

flights("12/12", "Saturday", "02.00", "BWI", "Delta", 120.00), # Baltimore

flights("12/13", "Sunday", "20.00", "BOS", "JetBlue", 89.00), # Boston

flights("12/13", "Sunday", "12.00", "SFO", "American", 170.00), # San Fran

flights("12/13", "Sunday", "15.00", "SAT", "United", 220.00), # San Antonio

flights("12/14", "Monday", "13.00", "JFK", "Southwest", 150.00), # NYC

flights("12/14", "Monday", "23.00", "JFK", "American", 75.00), # NYC

flights("12/15", "Tuesday", "11.00", "LAX", "United", 190.00), # Los Angeles

flights("12/15", "Tuesday", "12.00", "JNU", "Delta", 550.00), # Juneau

flights("12/16", "Wednesday", "08.00", "MSY", "United", 320.00), # New Orleans

flights("12/16", "Wednesday", "10.00", "CLE", "Spirit", 10.00), # Cleveland

flights("12/16", "Wednesday", "01.00", "ATL", "Southwest", 110.00) # Atlanta

]

# save the data to a file

pickle.dump(myFlight, open("target.p", "wb"))

**Application File:**

# Programmer: James Aniciete

# Course No.: CSC 157

# Lab No.: 12

# Date: 4/21/2020

import pickle

from flights import \* # imports the flights class

# store the list of flights in a variable named fl

fl = pickle.load(open("target.p", "rb"))

# initialize variables

response = "Y" # to continue/end search program

matches = 0 # counts matches per search

alts = 0 # counts alternative flights per search

altIndices = [] # stores indices of alternative flights

# function to convert military time to regular time

def regTime(militaryTime):

time = ""

newTime = 0

if int(militaryTime[0:2]) - 12 > 0:

newTime = int(militaryTime[0:2])-12

if newTime < 10:

time = "0" + str(newTime) + ":" + militaryTime[3:5] + " PM"

else:

time = militaryTime[0:2] + ":" + militaryTime[3:5] + " PM"

else:

time = militaryTime[0:2] + ":" + militaryTime[3:5] + " AM"

return time

# based on user inputs, search for available flights + possible alternative bookings

while response == "Y" or response == "y":

# ask user to choose a sorting option

answer = input("How would you like to sort the available flights?"

"\n\t1) Date"

"\n\t2) Airport Code (Destination)"

"\n\t3) Cost"

"\n\t4) Airline"

"\n\t5) All of the Above\n"

)

# sort by date

if answer == "1":

date = input("\nPlease enter a date (MM/YY): ")

print("")

for i in range(flights.flightCount):

if (date == fl[i].date):

# increment match counter

matches += 1

# display initial result header and flight info

if matches == 1:

print("Result(s): ")

print ("\t",

fl[i].date, fl[i].dow, regTime(fl[i].time),

fl[i].destination, fl[i].airline, "$", "%.2f" % fl[i].cost)

elif (date < fl[i].date):

alts += 1

altIndices.append(i)

# display alternative flights on later dates

if alts > 0:

# message for no matches

if matches == 0:

print("No flights meet that condition.")

# if there were matches, reset the counter

else:

matches = 0

print("\nAlternative booking(s) for later dates: ")

for i in altIndices:

print ("\t",

fl[i].date, fl[i].dow, regTime(fl[i].time),

fl[i].destination, fl[i].airline, "$", "%.2f" % fl[i].cost)

# reset alternative variables

alts = 0

altIndices = []

# message for no matches or alternatives

elif matches == 0:

print("No flights scheduled for dates past and including {}.".format(date))

# reset match counter if necessary

else:

matches = 0

# sorted by US airport code

elif answer == "2":

dest = input("Please enter a US airport code: ")

print("")

for i in range(flights.flightCount):

if (dest == fl[i].destination):

matches += 1

if matches == 1:

print("Result(s): ")

print ("\t",

fl[i].date, fl[i].dow, regTime(fl[i].time),

fl[i].destination, fl[i].airline, "$", "%.2f" % fl[i].cost)

if matches == 0:

print("No flights found for {}.".format(dest))

else:

matches = 0

# sort by max cost

elif answer == "3":

cost = float(input("Please enter a maximum cost: $"))

print("")

for i in range(flights.flightCount):

if (cost >= fl[i].cost):

matches += 1

if matches == 1:

print("Result(s): ")

print ("\t",

fl[i].date, fl[i].dow, regTime(fl[i].time),

fl[i].destination, fl[i].airline, "$", "%.2f" % fl[i].cost)

# alternative bookings that are up to $50 more

elif (cost < fl[i].cost) and (cost < fl[i].cost + 50):

alts += 1

altIndices.append(i)

if alts > 0:

if matches == 0:

print("No flights meet that condition.")

else:

matches = 0

print("Alternative booking(s) within $50: ")

for i in altIndices:

print ("\t",

fl[i].date, fl[i].dow, regTime(fl[i].time),

fl[i].destination, fl[i].airline, "$", "%.2f" % fl[i].cost)

alts = 0

altIndices = []

elif matches == 0:

print("All flights cost more than $%.2f." % (cost+50))

else:

matches = 0

# sort by airline

elif answer == "4":

airline = input("Please enter an airline: ")

print("")

for i in range(flights.flightCount):

if (airline == fl[i].airline):

matches += 1

if matches == 1:

print("Result(s): ")

print ("\t",

fl[i].date, fl[i].dow, regTime(fl[i].time),

fl[i].destination, fl[i].airline, "$", "%.2f" % fl[i].cost)

if matches == 0:

print("No flights found for {} Airlines.".format(airline))

else:

matches = 0

# sort by date, destination, and max cost

elif answer == "5":

date = input("Please enter a date (MM/YY): ")

dest = input("Please enter a US airport code: ")

cost = float(input("Please enter a maximum cost: $"))

print("")

# for loop to find exact match searches

for i in range(flights.flightCount):

if (date == fl[i].date) and (dest == fl[i].destination) and (cost >= fl[i].cost):

matches += 1

if matches == 1:

print("Result(s): ")

print ("\t",

fl[i].date, fl[i].dow, regTime(fl[i].time),

fl[i].destination, fl[i].airline, "$", "%.2f" % fl[i].cost)

# alternatives = flights that match two criteria

elif (date == fl[i].date and dest == fl[i].destination) or (date == fl[i].date and cost >= fl[i].cost) or \

(dest == fl[i].destination and cost >= fl[i].cost):

alts += 1

altIndices.append(i)

if alts > 0:

if matches == 0:

print("No flights meet all of those conditions.")

else:

matches = 0

print("Alternative booking(s) that satisfy two conditions: ")

for i in altIndices:

print ("\t",

fl[i].date, fl[i].dow, regTime(fl[i].time),

fl[i].destination, fl[i].airline, "$", "%.2f" % fl[i].cost)

alts = 0

altIndices = []

elif matches == 0:

print("No flights found that match at least two of those conditions.")

else:

matches = 0

# invalid search option entered

else:

print("Invalid response.")

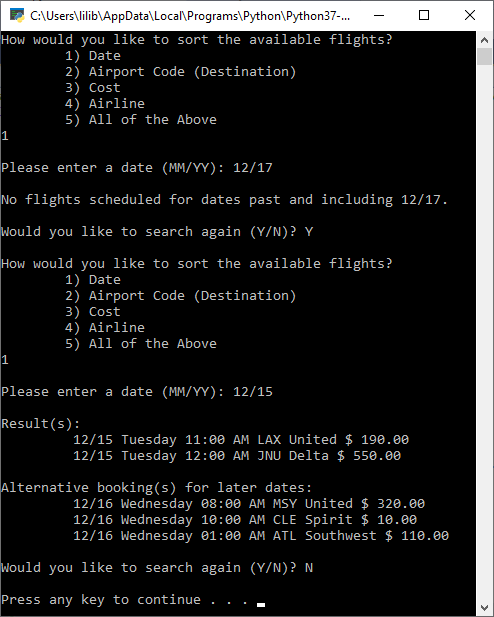
# search again or quit program

response = input("\nWould you like to search again (Y/N)? ")

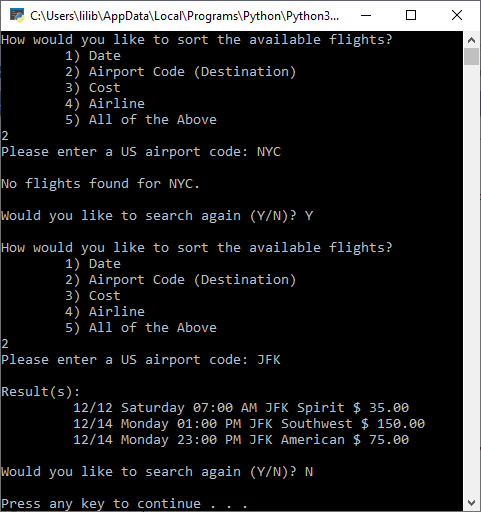
print("")

Snippet(s) of Output(s) from execution of modified Code

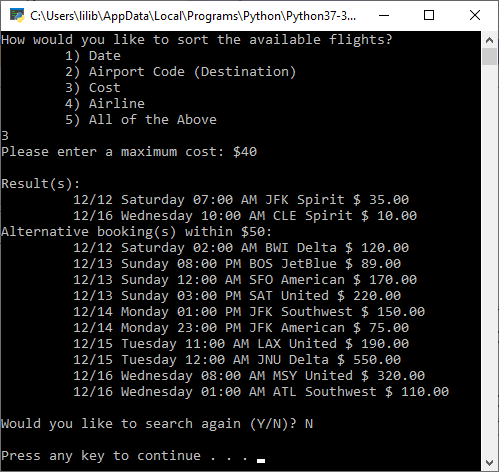
**Sorted by Date:**



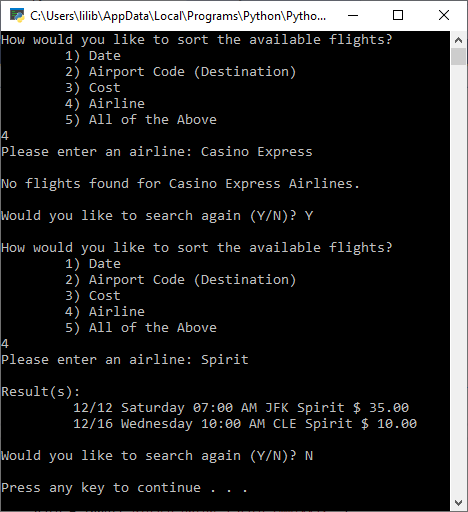
**Sorted by Destination:**



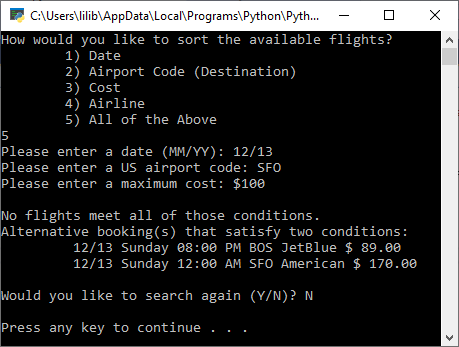
**Sorted by Cost:**



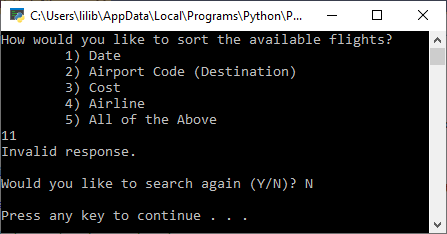
**Sorted by Airline:**



**Sorted by Date, Destination, and Cost:**



**Invalid search option:**



Excel Spreadsheet (when Calculations are involved)

**N/A**

Answers to Questions (Be sure to copy the questios themselves!)

**(1)** Explain how pickling is used in this application.

**Pickling is used to store a class for flight information, instances of that class, and a count of those instances in a class file called flights.py which is written in binary. Then, the application file loads that flight information and allows the user to query it.**

**(2)** The following block of statements appear in the starter code for this application.

|  |
| --- |
| **class flights(object) :**  **""" description of class """**  **flightCount = 0**  **def \_\_init\_\_(self, date, time, destination, cost) :** |

If you had to add an additional parameter to the **\_\_init\_\_()** method, aside from the date and time, etc., what would you choose as the parameter that would make a class object more " robust " with data.

**If I had to add an additional parameter, I would add a parameter for the name of the airlines.**

**(3)** The following code statement appears in the starter code for this application.

**pickle.dump(myFlight, open("target.p", "wb"))**

Explain, in detail, the function / purpose of the above line of code.

**The above line writes the pickled representation of myFlight, a list of flight information, to the file target.p in binary.**

**See:** [**https://docs.python.org/3/library/pickle.html#pickle.dumps**](https://docs.python.org/3/library/pickle.html#pickle.dumps)

**(4)** The following code statement appears in the starter code for this application.

**fl = pickle.load(open("target.p", "rb"))**

Explain, in detail, the function / purpose of the above line of code.

**The above line reads the pickled representation of the file target.p and returns the reconstituted object hierarchy specified therein to the variable fl.**

**See:** [**https://docs.python.org/3/library/pickle.html#pickle.dumps**](https://docs.python.org/3/library/pickle.html#pickle.dumps)

**(5)** What have you learned from performing and coding this lab assignment?

**I have learned more about pickling, if-elif-else statements, substrings, and about making programs that require user input.**