Introduction

Air travel is one of the most popular and efficient methods of transportation, especially for long-distance travels. As we approach the holiday season, we can expect that millions will be travelling by airplane to visit family or friends. Air travel helps connect people around the globe. In this assignment, you are asked to implement a program that loads text files representing some of the airports and flights* around the world and analyzes the flights.

*The data in these files are not necessarily real/accurate.

In this assignment, you will get practice with:

- Creating classes and objects
- Constructors, getters, setters, and other class methods
- Loading data from text files
- Cleaning and parsing data from text files
- Working with dictionaries and sets
- Algorithm development and testing; designing test cases
- Following program specifications

Text Files

In this assignment, there are 2 types of text files that will need to be read in.

One is an airport file which lists a number of airports around the world. Each line in this type of file contains a single airport and it shows the 3-letter airport code, the country in which the airport is located, and then the city in which it is located. These three items are separated by commas. Some of them contain spaces and/or tabs around the individual portions of the line that must be cleaned up when being read in.

The other is a flight file that contains a list of flights including a 6-character flight code, the origin airport code, and the destination airport code. Each line in the file represents one flight. Again, there may be spaces and/or tabs around the individual portions of the line that must be cleaned up when being read in.

```
YYZ,Canada,Toronto
YVR,Canada,Vancouver
YHZ,Canada,Halifax
YOW,Canada,  Ottawa
YEG,Canada,Edmonton
YUL,    Canada,Montreal
YWG,Canada, Winnipeg
 DFW ,United States,Dallas
LAX,United States,Los Angeles
SFO,United States,  San Francisco
```

A small snippet showing the format of an airport file.

```
XJX595,LAX,CPT
 CSX772, MAA,YHZ
LJC201,FCO,YOW
EYS649,YVR,PVG
OXD016,ORD, JFK
    DAJ762,YOW,TIP
QUZ869,YUL,MIA
RTK498,YVR,LAX
VEB477,PVG,PEK
```

A small snippet showing the format of a flights file.

Files

For this assignment, you must create three (3) Python files: *Airport.py*, *Flight.py*, and *Assign4.py*

When submitting your assignment on Gradescope, please submit these 3 files only. Do not upload any other files. **Make sure the files are named EXACTLY as specified here**.

Airport.py

The Airport file must contain a class called Airport. Everything in this file must be in the Airport class. Do not have any code in the file that isn't part of this class. As suggested by its name, this class represents an Airport in the program. Each Airport object must have a unique 3-letter code which serves as an ID, a city, and a country which are strings representing its geographical location.

Within the Airport class, you must implement the following functions based on these descriptions and specifications:

- **__init__(self, code, city, country):**
  - Initialize the instance variables _code, _city, and _country based on the corresponding parameters in the constructor
- **__repr__(self):**
  - Return the representation of this Airport in the following format:
    code (city, country)
    i.e. YYZ (Toronto, Canada)
- **getCode(self):**
  - Getter that returns the Airport code
- **getCity(self):**
  - Getter that returns the Airport city
- **getCountry(self):**
  - Getter that returns the Airport country
- **setCity(self, city):**
  - Setter that sets (updates) the Airport city
- **setCountry(self, ~~city~~ country):**
  - Setter that sets (updates) the Airport country

Flight.py

The Flight file must contain a class called Flight. Note that this file must import from Airport as it makes use of Airport objects; add the line `from Airport import *` to the top of the Flight file. Other than this import line, everything in this file must be in the Flight class. Do not have any other code in the file that isn't part of this class. As suggested by its name, this class represents a Flight from one Airport to another Airport in the program. Each Flight object must have a flightNo (a unique 6-character code containing 3 letters followed by 3 digits), an origin, and a destination. Both the origin and destination must be Airport objects within the program.

Within the Flight class, you must implement the following functions based on these descriptions and specifications:

- **__init__(self, flightNo, origin, destination):**
  - First check that both origin and destination are Airport objects (hint: use the isinstance operator). If either or both are not Airport objects, raise a TypeError that states "The origin and destination must be Airport objects"
  - When the origin and destination are both Airport objects, proceed to initialize the instance variables _flightNo, _origin, and _destination based on the corresponding parameters in the constructor
- **__repr__(self):**
  - Return the representation of this Flight containing the flightNo, origin city, and destination city, and an indication of whether the Flight is domestic or international (see the isDomesticFlight method description below). The representation must be in the following format:
    Flight: flightNo from originCity to destinationCity {domestic/international}
    Examples:
    Flight: MCK533 from Toronto to Montreal {domestic}
    Flight: WCL282 from Toronto to Chicago {international}
- **__eq__(self, other):**
  - Method that returns True if self and other are considered the same Flight: if the origin and destination are the same for both Flights. Make sure that if "other" variable is not a Flight object, this means False should be returned.
- **getFlightNumber(self):**
  - Getter that returns the Flight number code
- **getOrigin(self):**
  - Getter that returns the Flight origin
- **getDestination(self):**
  - Getter that returns the Flight destination
- **isDomesticFlight(self):**
  - Method that returns True if the flight is domestic, i.e. within a country (the origin and destination are in the same country); returns False if the flight is international (the origin and destination are in different countries)
- **setOrigin(self, origin):**
  - Setter that sets (updates) the Flight origin
- **setDestination(self, destination):**
  - Setter that sets (updates) the Flight destination

Assign4.py

This file is meant to be the core of the program from which the Airport and Flight objects should be created as their corresponding text files are loaded in; and several functions must be implemented to analyze the data and retrieve results about specific queries.

Since this file will be used to create Airport and Flight objects, both of those Python files must be imported into this one. To do this, add the following lines of code to the top of this file:

```
from Flight import *
from Airport import *
```

You then need to create two containers, allAirports and allFlights, to store all the Airport and Flight objects respectively. The allAirports container can be any type you wish (i.e. list, set, or dictionary); but the allFlights container **MUST** be a dictionary.

The rest of this file must be a series of functions based on these descriptions and specifications:

- **loadData(airportFile, flightFile):**
  - Read in all the data from the airport file of the given name. Extract the information from each line and create an Airport object for each. Remove any whitespace from the outside of **each portion** of the line (not just the line itself). As you create each Airport object, add the object to the allAirports container. Review the format of the airports file as well as the order in which the parameters are expected in the Airport constructor to ensure you send in the correct values for the correct parameters.
  - Read in all the data from the flight file of the given name. Extract the information from each line and create a Flight object for each. Remove any whitespace from the outside of **each portion** of the line (not just the line itself). As you create each Flight object, add the object to the allFlights **dictionary** in this specific way: the key must be the origin's airport code and the corresponding value must be a list of the Flight object(s) that depart from this origin airport. For example, if "YYZ" is an entry in this dictionary, its value would be a list of all Flight objects in which the Flight's origin is "YYZ" (all flights that go out from Pearson airport). Review the format of the flights file as well as the order in which the parameters are expected in the Flight constructor to ensure you send in the correct values for the correct parameters.
  - If both files load properly without errors, return True. If there is any kind of exception that occurs while trying to open and read these files, return False. Use a try-except statement to handle these cases.
- **getAirportByCode(code):**
  - Return the Airport object that has the given code (i.e. YYZ should return the Airport object for the YYZ (Pearson) airport). If there is no Airport found for the given code, just return -1.
- **findAllCityFlights(city):**
  - Return a list that contains all Flight objects that involve the given city either as the origin or the destination
- **findAllCountryFlights(country):**

    - Return a list that contains all Flight objects that involve the given country either as the origin or the destination (or both)
- **findFlightBetween(origAirport, destAirport):**
    - Check if there is a direct flight from origAirport to destAirport. If so, return a string of the format:
      Direct Flight: origAirportCode to destAirportCode
      i.e. Direct Flight: YYZ to ORD
    - If there is not a direct flight, check if there is a single-hop connecting flight from origAirport to destAirport. This means a sequence of exactly 2 flights such that the first flight begins in origDestination and ends in some airport "X", and the second flight goes from airport "X" to destAirport. Create and return a set (not a list) of all possible "X" airport codes representing the airports that could serve as the connecting airport from origAirport to destAirport. Do not worry about multiple-hop connections as that becomes very complicated to track. You should **only** look at single-hop connecting flights. Also, this should only be done if there was no direct flight.
    - If there is no direct flight AND no single-hop connecting flights from origAirport to destAirport, then just return -1.
- **findReturnFlight(firstFlight):**
    - Take the given Flight object and look for the Flight object representing the return flight from that given flight. In other words, given a Flight from origin A to destination B, find the Flight object that departs from origin B and arrives in destination A. If there is no such Flight object that goes in the opposite direction as firstFlight, just return -1.

Tester File

You are provided with a file called Asst4Tests.py. This file will run tests to check that each of your classes and functions are working as expected. Note that Gradescope will run other tests that may be hidden from you. It is your responsibility to do thorough testing to ensure the code works not just for the given tests but for many different test cases. Gradescope may also use different files (same format but different names and data) so your program should not have anything hardcoded.

If you are failing one or more tests, you should look at what the test is checking for and each of the sub-tests within that test (most tests are checking 2-4 different cases to ensure it works for all those cases). It may help to print out each of test case values, i.e.
print(t1)

```
print(t2)
print(t3)
```

You should also add other print lines to help debug and narrow down the source of the errors in your code.

Functional Specifications

Your program must work for any airports file and flights file. The format will be the same, but they could have different names and a different list of airports or flights. Your program must work with the files of the given names, so **do not hardcode it** to only work for the files provided to you.

The Asst4Tests.py file is a great way to test that your functions are working according to specifications. **However**, you must do your own testing in addition to the tests given in this file. The Gradescope autograder may run additional tests that are not visible to you, so your program should be well-tested to ensure it works not just for the given tests but in a large multitude of test cases.

Non-Functional Specifications

1. The program should strictly adhere to the requirements and specifications of these classes and functions.
2. The program should include brief comments in your code identifying yourself, describing the program, and describing key portions of the code.
3. Assignments are to be done individually and must be your own work. Software may be used to detect academic dishonesty (cheating).
4. Use Python coding conventions and good programming techniques. For example:
    a. Meaningful variable names
    b. Conventions for naming variables and constants
    c. Use of constants where appropriate
    d. Readability, indentation, and consistency
5. The name of the files you submit must be **Airport.py**, **Flight.py**, and **Assign4.py**

Make sure you develop your code with **Python 3.9** as the interpreter. Failure to do so may result in the testing program failing.

Tips and Guidelines

- Variables should be named in lowercase for single words and camel case for multiple words, i.e. origAirport
- Do not hardcode filenames, numbers of lines from the files, or any other variables
- You **can** assume that the file formats for each of the 2 types of text files will be the same for any file being used for testing. You may **not** assume that the number of lines nor the values stored in them will be the same.
- Add comments throughout your code to explain what each section of the code is doing and/or how it works

Rules

- Read and follow the instructions carefully.
- Only submit the Python files described in the Files section of this document.
- Submit the assignment on time. Late submissions will receive a late penalty of 10% per day (except where late coupons are used).
- Forgetting to submit a finished assignment is **not** a valid excuse for submitting late.
- Submissions must be done on Gradescope. They will not be accepted by email.
- You may re-submit your code as many times as you would like. Gradescope uses your last submission as the one for grading by default. There are no penalties for re-submitting. However, re-submissions that come in after the due date **will** be considered late and subject to penalties (or to the use of late coupons).
- Assignments will be run through a similarity checking software to check for code that looks very similar to that of other students. Sharing or copying code in any way is considered plagiarism and may result in a mark of zero (0) on the assignment and/or reported to the Dean's Office. Plagiarism is a serious offence. Work is to be done **individually**.

Submission

Due: Wednesday, December 7, 2022 at 11:55pm

You must submit the 3 files (Airport.py, Flight.py, and Assign4.py) to the Assignment 4 submission page on Gradescope. There are several tests that will automatically run when you upload your files. Some of the tests are visible to you so it will give you an idea of how well your program is working. However, we will also run some hidden tests so you won't see those tests nor the grade you get from those tests. It is recommended that you create your own test cases to check that the code is working properly for a multitude of different scenarios.

Assignments will not be accepted by email or any other form. They **must** submitted on Gradescope.

Marking Guidelines

The assignment will be marked as a combination of your auto-graded tests (both visible and hidden tests) and manual grading of your code logic, comments, formatting, style, etc. Below is a breakdown of the marks for this assignment:

[50 marks] Auto-graded Tests

[20 marks] Code logic and completeness

[10 marks] Comments

[10 marks] Code formatting

[10 marks] Meaningful and properly formatted variables

Total: 100 marks

The weight of this assignment is 12% of the course mark.