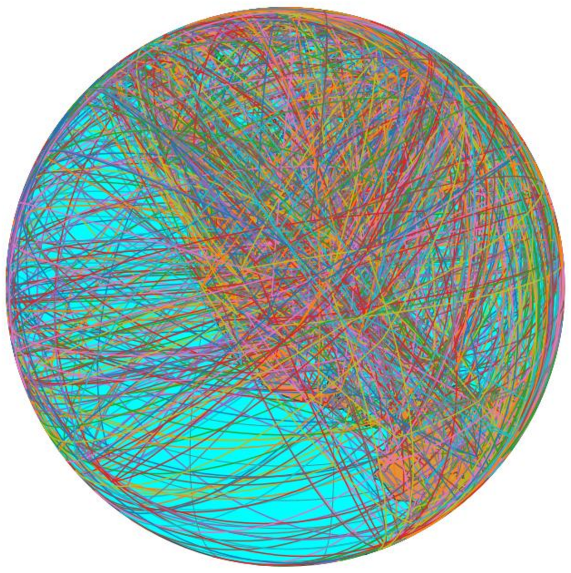
Please find the files to this project located at the following GitHub repository:

https://github.com/Slaporter/RubberBandBall.git

A ReadME has been included which holds detailed instructions on how to run the project. Please note that the unit tests were compiled using PyCharm and will not run from the command line unless "Term_Project." Is deleted from the top page.



Function Specification

My program has been designed to deliver a fuel management system to a large airline company. As fuel cost is the main constraint, my program has been designed to return a route with the lowest fuel cost possible while landing in all required airports and return home again. It checks if the aircraft can fly the distances required between the airports with its fuel capacity. If it can not the program will return not feasible.

All distances have been calculated using the metric system. This mean any ranges for the aircraft stored as Imperial measurements have been converted to Km. Distances between airports have been calculated using the Great Circle Distance as requested.

When an airplane lands in an airport it will be refuelled for the amount it lost travelling. The cost of this is converted from local currency to Euro. The final cost figure displayed by the system is an aggregated cost after landing in each airport and home again.

A plot has been designed to show the feasible routs on a map. Currently it is showing all routes in the "testroutes.csv" file that are feasible. This could be easily modified to show only one or a selection of routes as required.

Design

The program has been designed using a mixture of Object Orientating Programming and Dictionaries.

A class has been created for each element such as: aircraft, airport, Country currencies and currency rates. A class for all airports called airportAtlas has also been created. A dictionary is used within this class to store all the airport information. Please see appendix for class diagram.

A dictionary has been created to store the aircraft, currency, country currency. The aircraft dictionary updates the imperial measure system as it populates with data.

For each line in the testroute.csv file two functions will be called.

The first is a function to create a dictionary which holds all possible routes between airports, their distance, the rate to refuel in Euro and the total cost to refuel after that distance has been flown. A dictionary has been used to store this information as they are able to hold a key and a value efficiently. Look up in dictionaries have an efficiency O(1), they take up quite a bit of memory but I felt efficiency was important for this project.

The distance is stored in the dictionary calling the getDistanceBetweenAirports class. This class takes in two airport codes and calls the static method which works out the difference between the longitude and latitude. A distance in Km is returned.

The rate is stored in the dictionary by linking several classes together. To get the currency rate off the airport code, we must reference the country from the airport code, look into country currency with the country name and retrieve the currency code. This currency code can then be looked up in the currency rate and a rate to Euro returned. If there is an error anywhere in this process an error log will be created highlighting the area of issue and the currency rate set at 1. This means for some routes the quote for total cost may be not accurate. It is advised the error log is checked and updated accordingly.

Cost is calculated by multiplying distance and cost.

The second function takes to parameters, a line of flight details which should be in the format of: [‘DUB’,’LHR’,’SYD’,’JFK’,’AAL’,’777’] and the dictionary described above. The function uses permutations for itertools to return a list of all possible route. I found this function to be efficient and reliable for this project work. I call the function on the list which excludes the home airport and the type of aircraft. For each permutation possible, I add the home airport to the start and end of the list. I loop through the list checking if the aircraft specified is able to fly the distance between the stops. If it is not it will return not feasible on that route selection. If a route can be flown by the aircraft, I calculate the total cost of that route. I then loop through all possible results for the initial route and return the cheapest route if it is feasible at all. This result is stored in an excel sheet, accessible by the user.

For illustrative purposes I have written code that looks at all the cheapest, feasible routes, exports their Longitudes and Latitudes to a csv file. This file is then read in and plotted on a globe.

For all test cases run in the excel sheet provided, the plotted globe looks quite like a rubber band ball, as can be seen at the top of the document. If run on smaller test cases this visualization could be very useful in helping the company keep track of their airlines and the suggested routes.

Testing

Unit tests have been complied on the two functions required to calculate the fuel-efficient routes and the airportAtlas dictionary.

These tests have sample answers provided and then check if the code run matches the supplied answer. These tests check the accuracy of creating dictionaries, storing information in a dictionary, the distance calculation and the cost calculation.

All tests ran returned with status: Ok.

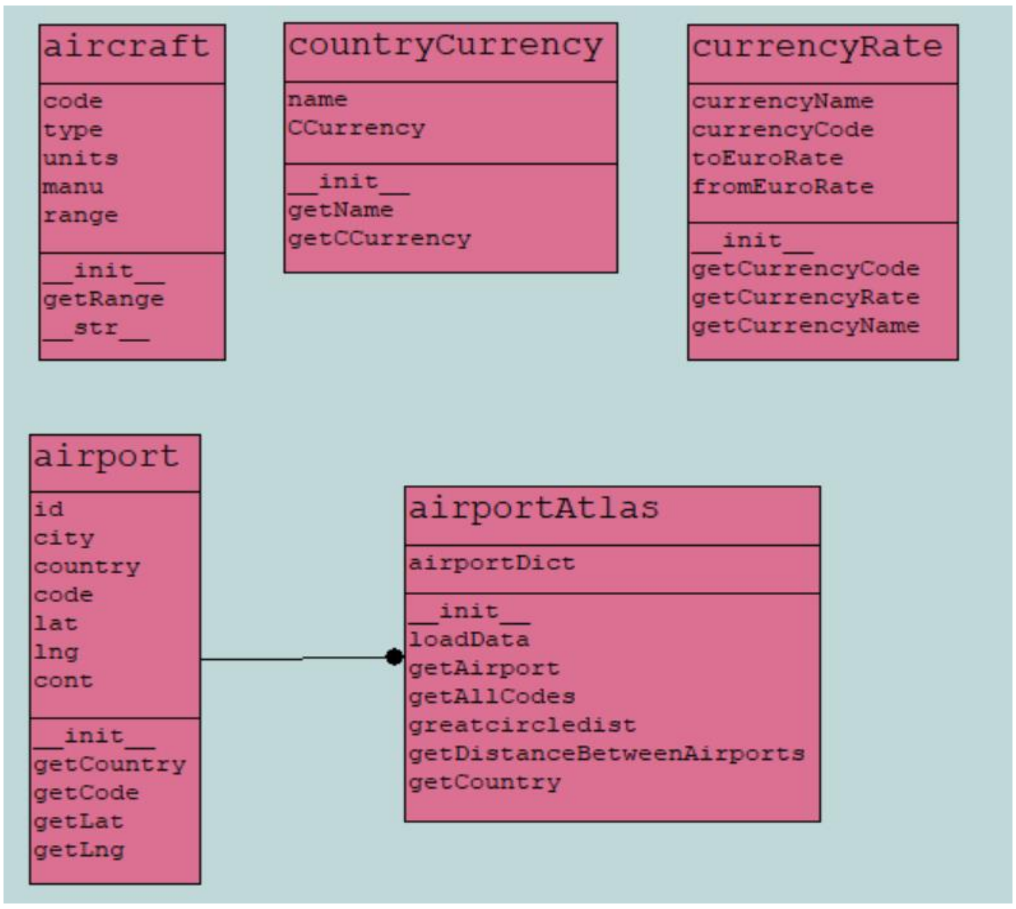
To further test the code I edited the sample output data supplied and created a testroute.csv file in the style specified. I ran my code on this file and my code produced the same results as the file.

Future Development

If I was working on this project again in the future I would look into implementing caching optimisations, increasing the number and quality of the visualisations produced and creating a user interface.

Appendix

Class Diagram


aircraft
code
type
units
manu
range
__init__
getRange
__str__
countryCurrency
name
CCurrency
__init__
getName
getCCurrency
currencyRate
currencyName
currencyCode
toEuroRate
fromEuroRate
__init__
getCurrencyCode
getCurrencyRate
getCurrencyName
airport
id
city
country
code
lat
lng
cont
__init__
getCountry
getCode
getLat
getLng
airportAtlas
airportDict
__init__
loadData
getAirport
getAllCodes
greatcircledist
getDistanceBetweenAirports
getCountry