|  |  |
| --- | --- |
|  |  |
| STEMCourage Task 25 |  |
|  |  |
| Flight Planner Coding | |

# Instructions

Your client has tasked you to develop a text-based flight planning application using Python. The program should ask the user for a route (in the form of a list of waypoints), calculate the total distance and then determine the fuel and flight time for several different types of aircraft.

**Requirements:**

The client has provided a list of requirements that have been organised using the MoSCoW method. MoSCoW stands for **must**, **should**, **could** and **would** – each signifying the importance of the requirement. For this task, deliverables will increase in difficulty as you move down the list. Ideally, you should aim to complete all **must** and **should**objectives, however the further you get the better.

**Must Have:**

1. The system shall ask the user to input any number of coordinates (lat,lon).
2. The system shall allow the user to exit the input process by typing ‘done’ into the console window.
3. The system shall be programmed defensively and validate all user input to a reasonable degree. (Eg. What if someone types ‘ done’, ‘done ‘ or ‘Done’)
4. The system shall calculate distance between two coordinates using Pythagoras' theorem.
5. The system shall calculate the total route distance using the Pythagoras function created in requirement #4. (Hint: see **Helpful Pointers**)
6. The system shall output the total length of the user’s route in kilometers.

**Should Have:**

1. The system shall validate a user’s coordinates to ensure they are a valid latitude and longitude pair.
2. The system shall limit a user’s route to **20** waypoints and display an appropriate message.
3. The system shall present the user with a list of aircraft (as defined in the *Aircraft Specficiations* table).
4. The user shall be able to select an aircraft by typing its name into the console.
5. The system shall display the following flight information (calculated using the selected aircraft)
   1. ‘Flight Possible’ - A **boolean** value which shows if the selected aircraft can complete the supplied route.
   2. ‘Fuel Load’ – A value which contains the aircraft’s fuel load in the format **“*fuel required/*fuel capacity”**
   3. **‘**Flight Time’ – A value which displays the flight time in hours (assuming the aircraft completes the journey at its cruise speed

**Could Have: (Optional)**

1. The system shall use the provided **DataService** class to display a list of pre-created waypoints to the user.
2. The system shall allow the user to select a waypoint via it’s name and add it to the route.
3. The system shall allow the user to input custom aircraft parameters (fuel capacity, fuel burn, fuel reserve and cruise speed)
4. The system shall save the user’s custom aircraft so it can be used again.

**Would Have: (Optional)**

Only attempt this part of the task if you have completed **all** of the previous requirements.

Your client has decided that they require greater accuracy when calculating the total distance of a route. Rather than using Pythagoras' theorem (which handles coordinates in a 2D space), allow the user to select the Haversine formula as an option to calculate distance. Haversine is used to calculate the distance between two points on a sphere (the sphere in this case being the Earth). This method is far more accurate and will output significantly higher distances due to the curvature of the planet.

Haversine Formula:



Where:

* *d* is distance between the two points
* *r* is the radius of the sphere
* φ1, φ2 are the latitude of point 1 and latitude of point 2 (in radians)
* λ1, λ2 are the longitude of point 1 and longitude of point 2 (in radians)

*λ1 - λ2* and *φ1 - φ2* may also be written as *Δφ or Δλ*.(Where delta ‘Δ’ represents the change of a value)

**Helpful Pointers:**

To calculate the length of a route, you will need to calculate the distance between each pair of coordinates and the next. This can be achieved using the following code:

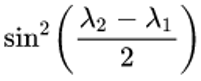


*Cosine*, *atan2* and *square root* operations can be performed in Python using the following code:



Python does not natively support the *sin2*, operation, however the same result can be achieved with the following code:

When implementing an algorithm in code, try to identify its core parts and split them up. For instance:



*becomes*



The following resource provides a more detailed explanation and JavaScript implementation of the Haversine formula. Make sure to check here if you get stuck: <https://www.movable-type.co.uk/scripts/latlong.html>

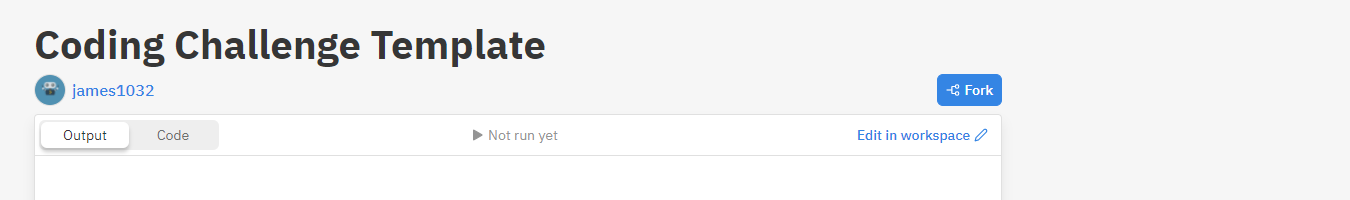
**Aircraft Specifications:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Aircraft Type** | **Fuel Capacity (kg)** | **Fuel Burn (kg/hr)** | **Fuel Reserve (%)** | **Cruise Speed (KTS)** |
| Extra 300 | 159 | 72 | 5 | 178 |
| Cessna 182 | 333 | 59 | 10 | 140 |
| BAES Hawk | 1360 | 374 | 10 | 430 |
| Learjet | 1870 | 645 | 10 | 503 |
| Boeing 737-8 | 20726 | 2657 | 10 | 453 |

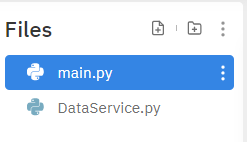
**Task Setup:**

You will be creating your program using Replit, an online IDE. Sign up for a free account and make a copy of the following project: [Coding Challenge Template - Replit](https://replit.com/@james1032/Coding-Challenge-Template)

You can do this by clicking the ‘Fork’ button, which will create a version that you can edit.

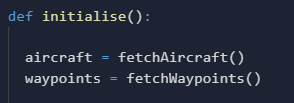


Inside you will find two files, **main.py** and **DataService.py** - write your code inside main.py.



You will see several empty functions. These can be used as a template for your program, but you are welcome to edit or add new ones as you wish.

**DataService.py** contains two pre-written functions for you to use. These will return arrays of waypoint and aircraft objects respectively. You can access these functions from **main.py** as follows:



**Example Output:**

The client has provided you with an example of what the program should look like.

# 

# Deliverables

Place your source code from **main.py** into a document. Please include a short (~200 words) explanation of your program and talk about any decisions you made.

Return to your mentor by email on or before the end of the month, including your name and your task ID