12.9 PROJECTS

Project 12.1 Tracking GPS coordinates

A GPS (short for Global Positioning System) receiver (like those in most mobile phones) is a small computing device that uses signals from four or more GPS satellites to compute its three-dimensional position (latitude, longitude, altitude) on Earth. By recording this position data over time, a GPS device is able to track moving objects. The use of such tracking data is now ubiquitous. When we go jogging, our mobile phones can track our movements to record our route, distance, and speed. Companies and government agencies that maintain vehicle fleets use GPS to track their locations to streamline operations. Biologists attach small GPS devices to animals to track their migration behavior.

In this project, you will write a class that stores a sequence of two-dimensional geographical points (latitude, longitude) with their timestamps. We will call such a sequence a *track*. We will use tracking data that the San Francisco Municipal Transportation Agency (SF MTA) maintains for each of the vehicles (cable cars, streetcars, coaches, buses, and light rail) in its Municipal Railway ("Muni") fleet. For example, the following table shows part of a track for the Powell/Hyde cable car in metropolitan San Francisco.

Time stamp	Longitude	Latitude
2014-12-01 11:03:03	−122.41144	37.79438
2014-12-01 11:04:33	−122.41035	37.79466
2014-12-01 11:06:03	−122.41011	37.7956
2014-12-01 11:07:33	−122.4115	37.79538
2014-12-01 11:09:03	−122.4115	37.79538

Negative longitude values represent longitudes west of the prime meridian and negative latitude values represent latitudes south of the equator. After you implement your track class, write a program that uses it with this data to determine the Muni route that is closest to any particular location in San Francisco.

An abstract data type for a track will need the following pair of attributes.

Instance Variable	Description
points	a list of geographical points with associated times
name	an identifier for the track

In addition, the ADT needs operations that allow us to add new data to the track, draw a picture of the track, and compute various characteristics of the track.

Method	Arguments	Description
append	point, time	add a point and time to the end of the track
length	—	return the number of points on the track
averageSpeed	—	return the average speed over the track
totalDistance	—	return the total distance traversed on the track
diameter	—	return the distance between the two points that are farthest apart on the track
closestDistance	point, error	find the closest distance a point on the track comes to the given point; return this distance and the time(s) when the track comes within error of this distance
draw	conversion function	draw the track, using the given function to convert each geographical point to an equivalent pixel location in the graphics window

Part 1: Write a Time class

Before you implement a class for the `Track` ADT, implement a `Time` class to store the timestamp for each point. On the book website is a skeleton of a `time.py` module that you can use to guide you. The constructor of your `Time` class should accept two strings, one containing the date in `YYYY-MM-DD` format and one containing the time in `HH:MM:SS` format. Store each of these six components in the constructed object. Your class should also include a `duration` method that returns the number of seconds that have elapsed between a `Time` object and another `Time` object that is passed in as a parameter, a `__str__` method that returns the time in `YYYY-MM-DD HH:MM:SS` format, a `date` method that returns a string representing just the date in `YYYY-MM-DD` format, and a `time` method that returns a string representing just the time in `HH:MM:SS` format. Write a short program that thoroughly tests your new class before continuing.

Part 2: Add a timestamp to the `Pair` *class*

Next, modify the `Pair` class from earlier in the chapter so that it also includes an instance variable that can be assigned a `Time` object representing the timestamp of the point. Also, add a new method named `time` that returns the `Time` object

representing the timestamp of the point. Write another short program that thoroughly tests your modified `Pair` class before continuing.

Part 3: Write a Track class

Now implement a `Track` class, following the ADT description above. The points should be stored in a list of `Pair` objects. On the book website is a skeleton of a `track.py` module with some utility methods already written that you should use as a starting point.

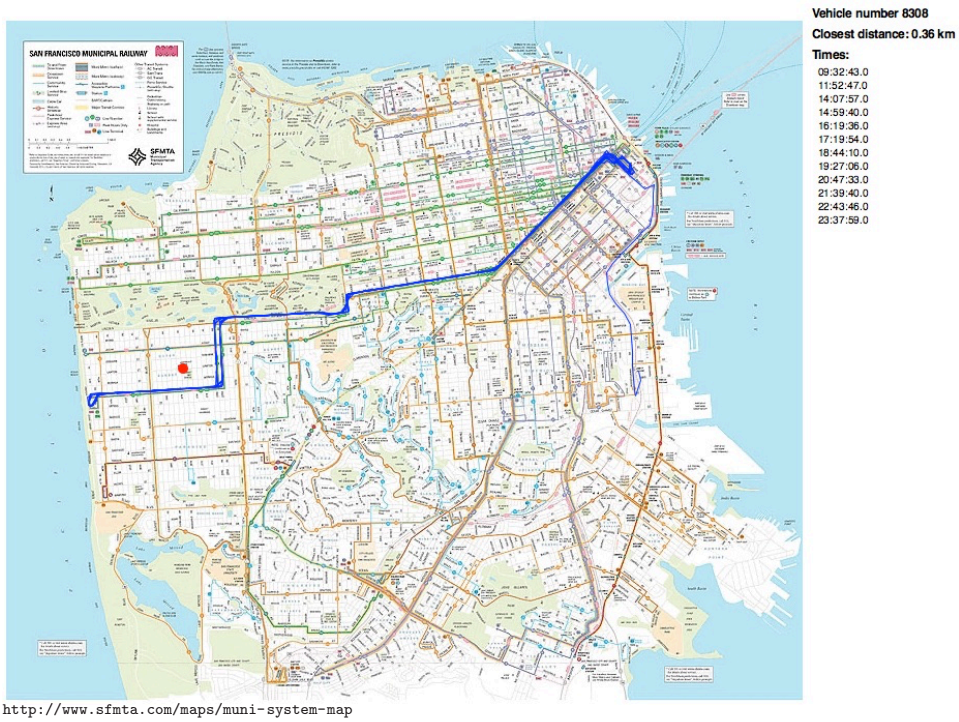
Question 12.1.1 *What does the* `_distance` *method do? Why is its name preceded with an underscore?*

Question 12.1.2 *What is the purpose of the* `degToPix` *function that is passed as a parameter to the* `draw` *method?*

After you write each method, be sure to thoroughly test it before moving on to the next one. For this purpose, design a short track consisting of four to six points and times, and write a program that tests each method on this track.

Part 4: Mysteries on the Muni

You are developing a forensic investigation tool that shows, for any geographical point in metropolitan San Francisco, the closest Muni route and the times at which the vehicle on that route passed by the given point. On the book website is an almost-complete program that implements this tool, named `muni.py`. The program should read one day's worth of tracking data from the San Francisco Municipal Railway into a list or dictionary of `Track` objects, set up a turtle graphics window with a map of the railway system, and then wait for a mouse click on the map. The mouse click triggers a call to a function named `clickMap`. The `clickMap` function draws a red dot where the click occurred, calls the `closestDistance` method of the `Track` class to find the Muni route that is closest to the click, and finally calls the `draw` method to draw the route on the map along with the times that the `Track` passed the click position. The output from the finished tool is visualized on the next page.



http://www.sfmta.com/maps/muni-system-map

In the `main` function of the program, the second-to-last function call

```
screen.onclick(clickMap)
```

registers the function named `clickMap` as the function to be called when a mouse click happens. Then the last function call

```
screen.mainloop()
```

initiates an *event loop* in the background that calls the registered `clickMap` function on each mouse click. The x and y coordinates of the mouse click are passed as parameters into `clickMap` when it is called.

The tracking data is contained in a comma-separated (CSV) file on the book website named `muni_tracking.csv`. The file contains over a million individual data points, tracking the movements of 965 vehicles over a 24-hour period. The only function in `muni.py` that is left to be written is `readTracks`, which should read this data and return a dictionary of `Track` objects, one per vehicle. Each vehicle is labeled in the file with a unique vehicle tag, which you should use for the `Track` objects' names and as the keys in the dictionary of `Track` objects.

Question 12.1.3 *Why is* `tracks` *a global variable? (There had better be a very good reason!)*

Question 12.1.4 *What are the purposes of the seven constant named values in all caps at the top of the program?*

Question 12.1.5 *What do the functions* `degToPix` *and* `pixToDeg` *do?*

Question 12.1.6 *Study the* `clickMap` *function carefully. What does it do?*

Once you have written the `readTracks` function, test the program. The program also uses the `closestDistance` method that you wrote for the `Track` class, so you may have to debug that method to get the program working correctly.