Homework 3

MPCS 51042 - Python Programming - Fall 2021

Deadline: October 19 at 5:00 pm CDT

Write each problem's solutions in a separate source file: `problem1.py` and `problem2.py`. Upload the 2 source files to Gradescope before the deadline.

- The source files, functions, and classes **must** be named as specified.
- You may define additional functions, if your solution requires them.
- You **must** comment each function definition with the following info. Place each function's comments immediately below its `def` statement.
  - The purpose of the function
  - The meaning of the function's parameters, if any
  - The meaning of the function's return value, if any

Problem 1

Write a function called `vectorize(func)` that takes a function as its only positional argument and returns a new function that calls `func` multiple times. `func` is a function that takes only one argument and the new function should accept any number of positional arguments. If N arguments are passed to the new function, then `func` is called N times with each of the values as a single argument. The new function should return a list of values from each invocation of `func`.

Sample use:

```
>>> from math import sqrt
>>> multi_sqrt = vectorize(sqrt)
>>> multi_sqrt(4.0, 25.0, 1.0)
[2.0, 5.0, 1.0]

>>> multi_upper = vectorize(str.upper)
>>> multi_upper("hi", "HELLO", "Bye")
['HI', 'HELLO', 'BYE']
```

Problem 2

In this problem, you will explore data from the City of Chicago `https://data.cityofchicago.org` and take advantage of object-oriented programming to build the foundation for a hypothetical "Chicago Public Schools application." Use the "schools.csv", a CSV file with data on each public school in Chicago including its name, location, address, along with other information. For this problem you will write three classes that will allow a user to easily interact with this data.

Specifications

The details below indicate what classes, methods, and functions **must** be implemented to receive full credit. However, you should feel free to write whatever helper functions or methods that you find useful.

Coordinate class

The `Coordinate` class stores a latitude, longitude pair indicating a physical location on Earth.

- The `__init__(self, latitude, longitude)` method should accept two floating point numbers that represent latitude and longitude in radians.
- `Coordinate.fromdegrees(cls, latitude, longitude)` should be a `@classmethod` that accepts two floats representing latitude and longitude in degrees and returns an instance of `Coordinate`.
- The `distance(self, coord)` method should accept another instance of `Coordinate` and calculate the distance in miles to it from the current instance. Below there is a description for how to calculate distances using latitude and longitude.
- The `as_degrees(self)` method should return a tuple of the latitude and longitude in degrees.
- The `show_map(self)` method should open up Google Maps in a web browser with a point placed on the latitude/longitude of the coordinate. You can use the URL `http://maps.google.com/maps?q=<latitude>,<longitude>` where `<latitude>` and `<longitude>` have been replaced by the corresponding decimal degrees.

Calculating the distance between points

One way to calculate the distance between a pair of points given as latitude, longitude coordinates is using the Haversine formula (`https://en.wikipedia.org/wiki/Haversine_formula`), which is given by:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos\varphi_1 \cos\varphi_2 \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right).$$

Above, d is the distance between two points, r is the radius of Earth (you can use 3961 miles), φ_1 and φ_2 are the latitudes of two points in radians, and λ_1 and λ_2 are the longitude of two points in radians. (`https://en.wikipedia.org/wiki/Radian#Conversion_between_radians_and_degrees`).

School class

- Write a class named `School` where each instance represents a single public school in Chicago.
- The `__init__(self, data)` method should receive a dictionary corresponding to a row of the "schools.csv" CSV file. In its body, it should create the following attributes:
  - `self.id` – the unique ID of the school ("School_ID" column) stored as an integer
  - `self.name` – short name of the school ("Short_Name" column)
  - `self.network` – the network the school is in ("Network" column)
  - `self.address` – the street address of the school
  - `self.zip` – the ZIP code of the school
  - `self.phone` – the phone number of the school
  - `self.grades` – a list of grades taught at the school (stored as an actual list, not just the string from the CSV file)
  - `self.location` – the location of the school as an instance of the `Coordinate` class (which you wrote above). Note that our data is given in degrees, not radians.

- The distance(self, coord) method should accept an instance of the Coordinate class and return the distance in miles from the specified location to the school.
- The full_address(self) method should return a multi-line string (a string that includes a newline character with the street address, city, state, and ZIP code of the school.

CPS class

The CPS class stores a list of public schools in the city.

- The __init__(self, filename) method accepts a filename for the CSV file in which our school data is stored. It should create an attribute calls schools that is a list of School instances.
- The nearby_schools(self, coord, radius=1.0) method accepts an instance of the Coordinate class and returns a list of School instances that are within radius miles of the given coordinate.
- The get_schools_by_grade(self, *grades) method accepts one or more grades as strings ("K", "3", etc.) and returns a list of School instances that teach all of the given grades.
- The get_schools_by_network(self, network) method accepts the network name as a string (like "Charter") and returns a list of School instances in that network.

Sample session:

The example below shows an example interaction with the classes above. Note that we've implemented a __repr__() method in the School and Coordinate classes for nice printing (https://docs.python.org/3/reference/datamodel.html#object.__repr__). Feel free to implement one as well, if you wish.

```
>>> cps = CPS("schools.csv")
>>> cps.schools[:5]
[GLOBAL CITIZENSHIP, ACE TECH HS, LOCKE A, ASPIRA - EARLY COLLEGE HS, ASPIRA - HAUGAN]
>>> len(cps.schools)
661
>>> [sc for sc in cps.schools if sc.name.startswith("OR")]
[ORTIZ DE DOMINGUEZ, ORIOLE PARK, OROZCO, ORR HS]
>>> ace_tech = cps.schools[1]
>>> print(ace_tech.full_address())
5410 S STATE ST
Chicago, IL
60609
>>>
>>> the_bean = Coordinate.fromdegrees(41.8827, -87.6233)
>>> cps.nearby_schools(the_bean, radius=0.5)
[NOBLE - MUCHIN HS, YCCS - INNOVATIONS]
>>> cps.get_schools_by_grade("PK", "12")
[FARRAGUT HS]
>>> cps.get_schools_by_network("Contract")
[CHIARTS HS, HOPE INSTITUTE, PLATO, CHICAGO TECH HS]
>>>
>>> mount_rushmore = Coordinate.fromdegrees(43.8791, -103.4591)
>>> mount_rushmore.show_map()
```