Homework 4

MPCS 51042 – Python Programming

Due: November 3rd 2020, 11:59 pm

Initial Setup

Make sure to perform a pull upstream inside your repository. This will grab the distribution code for hw4. The command is the following:

```
$ git pull upstream master
```

Make sure to read the text in the starter files! There might be additional restrictions you must follow.

Style Guide

For this homework and all future homework assignments, we will follow the style guide used by the undergraduate Python course. It's located here: https://classes.cs.uchicago.edu/archive/2020/fall/12100-1/style-guide/index.html

Problem 1

Memoization is an optimization technique for speeding up function calls by caching the function result for a given set of inputs.

The standard library `functools` module contains a function wrapper (i.e., a function that encloses another function) called `lru_cache`, which performs memoization on any function that it wraps. Furthermore, it only stores function results for the N most recent calls. This is called a least-recently used (LRU) cache.

For this problem, you must write your own version of the `lru_cache` wrapper.

Specifications

`lru_cache(func, max_size=128)` should be a function wrapper. The returned function object should:

- Call the wrapped function (`func`) with its arbitrary arguments.
- Maintain an LRU cache called `cache`. It should store up to `max_size` arguments and corresponding results (default = 128). You can implement the cache with any data structure. Here a few additional requirements about maintaining this structure:
  - Distinct argument patterns **are** distinct calls with separate cache entries. For example, `f(a=1, b=2)` and `f(b=2, a=1)` differ in their keyword argument order and must be two separate cache entries.
  - Function arguments of different types **must** be cached separately. For example, `f(3)` and `f(3.0)` will be treated as distinct calls with distinct results.
  - Your cache must use a least recently used replacement policy. The Wikipedia article about cache replacement policies provides a good visual example on how LRU works:

https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_recently_used_(LRU)

- Inside **problem1.py**, you must define a new class called `CacheInfo`. It should have the following attributes:
  - `hits`: The number of calls to the function where the result was calculated previously and can be returns from the cache.
  - `misses`: The number of calls to the function where the result was not previously calculated.
  - `max_size`: The maximum number of entries that the cache can store.
  - `curr_size`: The number of entries currently stored in the cache.
  - These are not private attributes; therefore, they should be accessible outside the class.
  - The class must also provide its own string representation that looks like the following:

```
In [1]: from problem1 import CacheInfo

In [2]: cache_info = CacheInfo(0, 0, 4, 0)

In [3]: print(cache_info)
CacheInfo(hits=0, misses=0, max_size=4, curr_size=0)
```

- You should define a variable `cache_info` that holds the current state (i.e., `CacheInfo` object) of the `cache`.
- Have a function attribute, `get_info()` that returns `cache_info`.

We provide to you a **hw4/problem1/driver.py** file. It runs a few simple test-cases for your cache. Here is the expected output.

```
:$ ipython driver.py
CacheInfo(hits=0, misses=0, max_size=4, curr_size=0)
4
CacheInfo(hits=0, misses=1, max_size=4, curr_size=1)
16
CacheInfo(hits=0, misses=2, max_size=4, curr_size=2)
4
CacheInfo(hits=1, misses=2, max_size=4, curr_size=2)
25
CacheInfo(hits=1, misses=3, max_size=4, curr_size=3)
36
CacheInfo(hits=1, misses=4, max_size=4, curr_size=4)
49
CacheInfo(hits=1, misses=5, max_size=4, curr_size=4)
49
CacheInfo(hits=2, misses=5, max_size=4, curr_size=4)
***Starting Sum*****
9
CacheInfo(hits=0, misses=1, max_size=4, curr_size=1)
9
CacheInfo(hits=1, misses=1, max_size=4, curr_size=1)
9
CacheInfo(hits=1, misses=2, max_size=4, curr_size=2)
9
CacheInfo(hits=2, misses=2, max_size=4, curr_size=2)
9
CacheInfo(hits=2, misses=3, max_size=4, curr_size=3)
```

Problem 2

In this problem, we're going to continue exploring https://data.cityofchicago.org from the City of Chicago, except this time we're going to take advantage of object-oriented programming to build the foundation for a hypothetical "Chicago Public Schools application". We have provided you with a CSV file with data on each public school in Chicago including its name, location, address, what grades are taught, and what network it is part of. You are asked to write three classes that will allow a user to easily interact with this data.

Specifications

The specifications below indicate what classes/methods/functions must be implemented to receive full credit. However, you should feel free to implement helper methods/functions if you find doing so to be useful.

School Class

- Write a class named `School`, each instance of which represents a single public school in Chicago.
- The `__init__(self, data)` method should receive a dictionary corresponding to a row in the CSV file (see suggested CSV reading code in CPS Class). In its body, it should create the following attributes:
  - `self.id` – the unique ID of the school ("School_ID" column) stored as an int
  - `self.name` – short name of the school ("Short_Name" column)
  - `self.network` – the network the school is in ("Network" column)
  - `self.address` – 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 (must be an actual `list`, not just the string that appears in the CSV)
  - `self.location` – the location of the school as an instance of `Coordinate` (see specifications in Coordinate Class)
- The `open_website(self)` method should open a website showing information about the school using the `webbrowser.open_new_tab()`(https://docs.python.org/3/library/webbrowser.html#webbrowser.open_new_tab) function from the standard library. The URL you can use for this is "http://schoolinfo.cps.edu/schoolprofile/SchoolDetails.aspx?SchoolId=<id>" where "<id>" is the unique ID of the school as listed in the spreadsheet.
- The `distance(self, coord)` method should accept an instance of `Coordinate` and return the distance in miles "as the crow flies" from the specified location to the school. See the description below (Calculating Distance Between Points section) for how to calculate distances using latitudes/longitudes.
- The `full_address(self)` method should return a multi-line string (that is, a string with a newline character within it) with the street address, city, state, and ZIP code of the school. You can use the normal type addressing format. For example, `"4563 N. Broadway St. Chicago, IL 60640"`. The city and state will always be Chicago IL. This format does not have to be exact but rather ensure that the required information is presented in a readable way.

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 floats that represent the latitude and longitude in radians.

- The `distance(self, coord)` method should accept another instance of `Coordinate` and calculate the distance in miles to it from the current instance.
- 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 on a web browser with a point placed on the latitude/longitude. The URL you can use for this is "`http://maps.google.com/maps?q=<latitude>,<longitude>`" where 'latitude' and 'longitude' have been replaced by the corresponding decimal degrees.

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 called `schools` that is a list of `School` instances. To iterate over the rows in the CSV file, you can use the following code:

```
import csv

with open(filename, newline='') as f:
    reader = csv.DictReader(f)
    for row in reader:
        ...
```

- The `nearby_schools(self, coord, radius=1.0)` method accepts an instance of `Coordinate` 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 (e.g., 'Charter') and returns a list of `School` instances in that network.

Calculating distance between points

Since the school locations are given as latitude/longitude coordinates, we need a way to calculate distance given two such pairs. One recommended way to do this is using the Haversine formula(https://en.wikipedia.org/wiki/Haversine_formula), which is

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

where 'd' is the distance between the two points, 'r' is the radius of the Earth (use 3961 miles), φ_1, and φ_2 are the latitudes of the two points in radians, and λ_1 and λ_2 are the longitudes of the two points in radians. Note that the data you are given is in degrees, not radians, so make sure you convert it(https://en.wikipedia.org/wiki/Radian#Conversion_between_radians_and_degrees) first.

Degrees Conversion

Inside the **problem2.py** file, define a function `from_degrees(latitude, longitude)` that accepts two floats representing the latitude and longitude in degrees and returns an instance of `Coordinate` (i.e., they should now be converted to radians).

Example Interaction

The example below shows an example interaction with these classes at a Python console.

```
In [1]: from problem2 import CPS, from_degrees

In [2]: cps = CPS('schools.csv')

In [3]: cps.schools[:5]
Out[3]:
[School(GLOBAL CITIZENSHIP),
 School(ACE TECH HS),
 School(LOCKE A),
 School(ASPIRA - EARLY COLLEGE HS),
 School(ASPIRA - HAUGAN)]

In [4]: [s for s in cps.schools if s.name.startswith('OR')]
Out[4]:
[School(ORTIZ DE DOMINGUEZ),
 School(ORIOLE PARK),
 School(OROZCO),
 School(ORR HS)]

In [5]: ace_tech = cps.schools[1]

In [6]: ace_tech.location
Out[6]: Coordinate(41.7961215104, -87.6258490365)

In [7]: the_bean = from_degrees(41.8821512, -87.6246838)

In [8]: cps.nearby_schools(the_bean, radius=0.5)
Out[8]: [School(NOBLE - MUCHIN HS), School(YCCS - INNOVATIONS)]

In [9]: cps.get_schools_by_grade('PK', '12')
Out[9]: [School(FARRAGUT HS)]

In [10]: cps.get_schools_by_network('Contract')
Out[10]:
[School(CHIARTS HS),
 School(HOPE INSTITUTE),
 School(PLATO),
 School(CHICAGO TECH HS)]
```