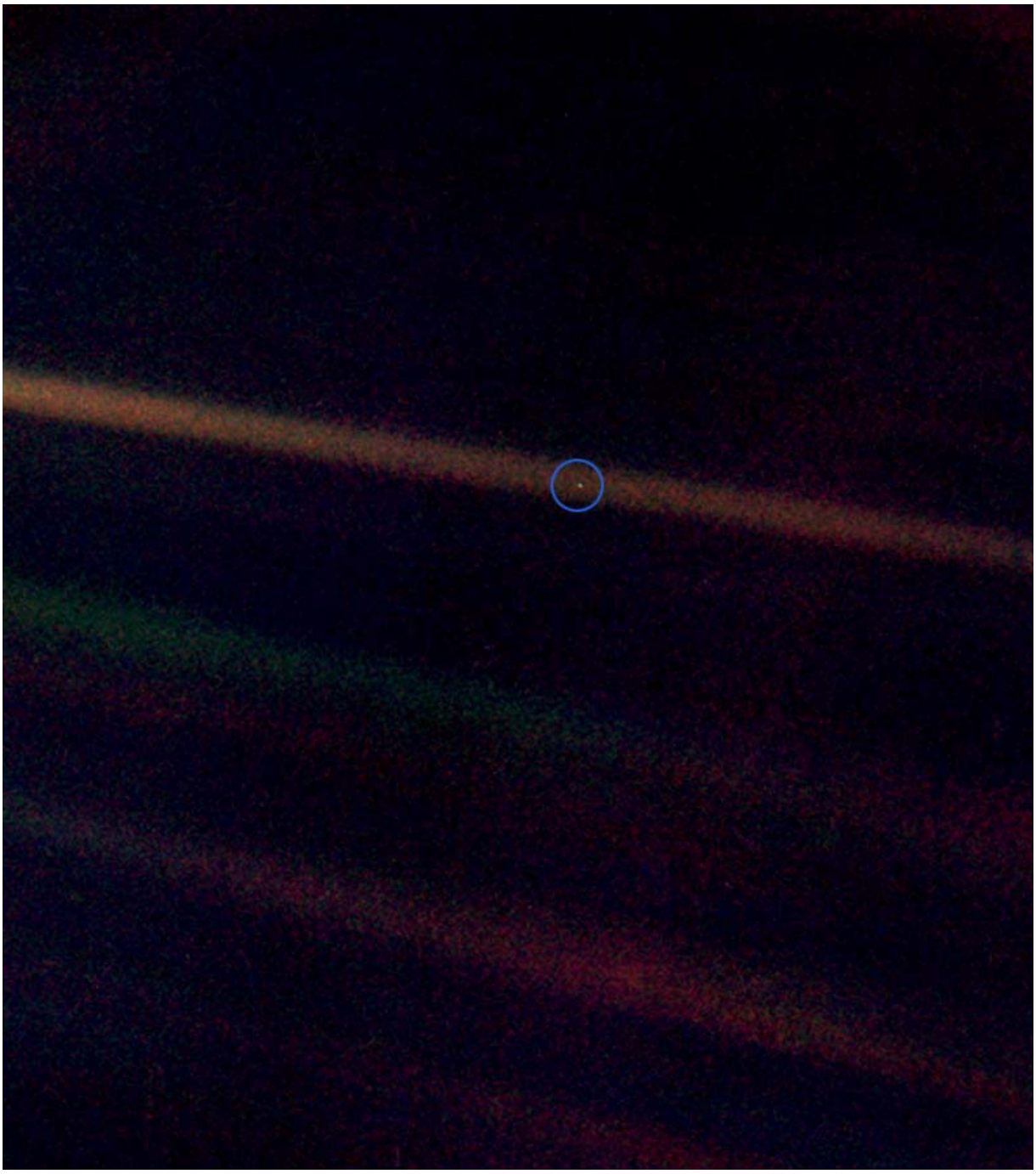
Homework 4: Pale Blue Dot



(https://en.wikipedia.org/wiki/Pale_Blue_Dot)

https://en.wikipedia.org/wiki/Pale_Blue_Dot

(https://en.wikipedia.org/wiki/Pale_Blue_Dot)

Due Date

Friday, April 8 @ 2 hours after 11:59pm (2am Saturday morning)

Topics

- Python
- Data Parsing
- Working with Geolocation
- Connecting to a Web API

Resources

Python Documentation: https://docs.python.org/3/ (https://docs.python.org/3/)
Lat/Long Conversions: http://www.movable-type.co.uk/scripts/latlong.html (http://www.movable-type.co.uk/scripts/latlong.html)
City Locations: https://www.maxmind.com/en/free-world-cities-database (https://www.maxmind.com/en/free-world-cities-database)
ISS API: http://api.open-notify.org (http://api.open-notify.org)
Observatory locations: http://www.minorplanetcenter.net/iau/lists/ObsCodes.html
(http://www.minorplanetcenter.net/iau/lists/ObsCodes.html)
Country Codes: https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2 (https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2)
Google is Your Friend: https://www.google.com/ (https://www.google.com/)

Pale Blue Dot

Implement the PaleBlueDot class with the following functionality pertaining to locations and distances on the planet Earth.

Data Parsing and Populating Data Structures (1 point)

```
def getCityLocation(self, city, region, country)
```

Given the name a city, its region, and country, returns its location as a Location object. Returns an empty list if the city is unknown, though bad inputs will not be tested for grading so an error can be thrown instead. This will require parsing the provided "worldCities.csv" file as part of the constructor call. This file contains the country, name, region, latitude, and longitude of more than 3 million cities on the planet. A separate "worldCities-abridged.csv" is also provided for more efficient development which contains a random sample of 1% of full dataset.

```
def getObservatoryName(self, observatoryCode)
```

Given a 3 digit observatory code as a string, return the name of the corresponding observatory. The most current list of observatories must be obtained from the Internet programmatically. The data is not provided, can't be submitted, and will not be present during testing. Don't worry, python simplifies the process of pulling data from Internet. Current data can be downloaded here: http://www.minorplanetcenter.net/iau/lists/ObsCodes.html Note that this data is not in the most friendly format and much care must be taken while parsing. Non-existing codes will not be tested.

Geographic Math (1 point)

```
def getObservatoryLocation(self, observatoryCode)
```

Given a 3 digit observatory code as a string, return the location of the corresponding observatory as a Location object with lat /long in degrees. Note that the data is given as longitude (in degrees), cos, and sin. Computing atan(sin/cos) will give the latitude in radians. Non-existing codes will not be tested.

```
def greatCircleDistance(self, location1, location2)
```

Return the great circle distance between two locations on Earth. This measure of distance is "as the crow flies" and takes into account the curvature of the Earth. For more information on great circle distance including sample code in JavaScript see: http://www.movable-type.co.uk/scripts/latlong.html (http://www.movable-type.co.uk/scripts/latlong.html) For this function, use the average Earth radius of 6,371km as given in the link above.

ISS API (1 point)

```
def getClosestObservatory(self, location)
```

Given a point on Earth by lat/long, return the code of the closest observatory in terms of great circle distance. This will be used as a helper function for the ISS calculation.

```
def getClosestObservatoryToISS(self)
```

Use the ISS web API (http://api.open-notify.org/ (http://api.open-notify.org/)) to get the current location of the ISS (projected to the Earth's surface) and return the closest observatory to that location in terms of great circle distance.

```
def nextPassTime(self, location)
```

Use the ISS web API's "pass time" functionality to return the next chance to observe the ISS from a given location. Parsing the JSON will result in a unix timestamp that must be converted and returned in a user friendly format via:
datetime.datetime.fromtimestamp(int("1457985637")).strftime('%Y-%m-%d %H:%M:%S')

Efficiency (1 point)

```
def getClosestCity(self, location)
```

Yes, I just said efficiency while referring to python. We don't expect to reach the performance levels of a compiled language while interpreting python, but we can still see significant improvements in runtime by using efficient data structures and algorithms.

Given a location on Earth by lat/long, return the name of the closest city in terms of great circle distance. Most solutions will technically be O(n), but an efficient solution should roughly be O(1) on average. To achieve this, you can take advantage of the fact that you have all the data in advance and it will not change during testing. The full "worldCities.csv" dataset will be used in testing. Return the city as a list in the form [city,region,countryCode]

No partial credit will be given if the function is correct, but doesn't meet the efficiency bound.

Bound: Less than 10 seconds on the benchmark in hw4Testing.py. This is the time that displays as "part 4 runtime:" when running the tests. Be sure to download the new testing file that is making 50 calls on 2.24 million cities.

```
def getClosestCityToISS(self)
```

Call getClosestCity with the current ISS location to find the closest city to the ISS. Return the city as a list in the form [city,region,countryCode]

Code

Download and complete PaleBlueDot.py with your solutions.

hw4.zip (homeworkCode/hw4.zip)

Submission

Submit: PaleBlueDot.py
Submission must be made on the submission website: https://www-student.cse.buffalo.edu/planetexpress/ (https://www-student.cse.buffalo.edu/planetexpress/)

Solution

PaleBlueDotSolution.py (homeworkSolutions/PaleBlueDotSolution.py)

Results