UDACITY

Return to Classroom

# Investigating Near-Earth Objects

REVIEW

CODE REVIEW 2

HISTORY

▼ database.py 2

```
"""A database encapsulating collections of near-Earth objects and their clo

A `NEODatabase` holds an interconnected data set of NEOs and close approach
It provides methods to fetch an NEO by primary designation or by name, as w
as a method to query the set of close approaches that match a collection of
user-specified criteria.

Under normal circumstances, the main module creates one NEODatabase from th
data on NEOs and close approaches extracted by `extract.load_neos` and
`extract.load_approaches`.

You'll edit this file in Tasks 2 and 3.
"""
from datetime import datetime

class NEODatabase:
    """A database of near-Earth objects and their close approaches.

    A `NEODatabase` contains a collection of NEOs and a collection of close
    approaches. It additionally maintains a few auxiliary data structures t
    help fetch NEOs by primary designation or by name and to help speed up
    querying for close approaches that match criteria.
    """
    def __init__(self, neos, approaches):
        """Create a new `NEODatabase`.
```

```
26
27          As a precondition, this constructor assumes that the collections of
28          and close approaches haven't yet been linked - that is, the
29          `.approaches` attribute of each `NearEarthObject` resolves to an em
30          collection, and the `.neo` attribute of each `CloseApproach` is Non
31
32          However, each `CloseApproach` has an attribute (`._designation`) th
33          matches the `.designation` attribute of the corresponding NEO. This
34          constructor modifies the supplied NEOs and close approaches to link
35          together - after it's done, the `.approaches` attribute of each NEO
36          a collection of that NEO's close approaches, and the `.neo` attribu
37          each close approach references the appropriate NEO.
38
39          :param neos: A collection of `NearEarthObject`s.
40          :param approaches: A collection of `CloseApproach`es.
41          """
42          self._neos = neos
43          self._approaches = approaches
44
45          # Link together the NEOs and their close approaches.
46          for x in range(len(self._approaches)):
47              for i in range(len(self._neos)):
48                  if self._neos[i].designation == self._approaches[x]._design
49                      self._neos[i].approaches.append(self._approaches[x])
50                      self._approaches[x].neo = self._neos[i]
```

SUGGESTION

This nested for loop is time cosuming.
It's better to create a dictionary mapping `designation` to `neo` and then link together `neos` and `ap`
For example,

```
for approach in self._approaches:
    neo = ... # retrieve neo through designation in the dictionary
    # set approach.neo to neo
    # append approach to neo.approaches
```

```
51
52
53      def get_neo_by_designation(self, designation):
54          """Find and return an NEO by its primary designation.
55
56          If no match is found, return `None` instead.
57
58          Each NEO in the data set has a unique primary designation, as a str
59
60          The matching is exact - check for spelling and capitalization if no
61          match is found.
62
63          :param designation: The primary designation of the NEO to search fo
64          :return: The `NearEarthObject` with the desired primary designation
65          """
66          # Fetch an NEO by its primary designation.
67          neo_by_designation = ''
68
69          for i in range(len(self._neos)):
```

```
70              if self._neos[i].designation == designation:
71                  neo_by_designation = self._neos[i]
72
73          if neo_by_designation=='' or neo_by_designation == None:
74             return None
75
76          return neo_by_designation
```

SUGGESTION

If you create a dictionary(self.neo_by_designation) mapping `designation` to `neo` in initialization, you
For example, `self.neo_by_designation.get(designation)`.

```
77
78
79      def get_neo_by_name(self, name):
80          """Find and return an NEO by its name.
81
82          If no match is found, return `None` instead.
83
84          Not every NEO in the data set has a name. No NEOs are associated wi
85          the empty string nor with the `None` singleton.
86
87          The matching is exact - check for spelling and capitalization if no
88          match is found.
89
90          :param name: The name, as a string, of the NEO to search for.
91          :return: The `NearEarthObject` with the desired name, or `None`.
92          """
93
94
95          # Fetch an NEO by its name.
96          neo_by_name = ''
97          for i in range(len(self._neos)):
98              if self._neos[i].name == name:
99                  neo_by_name = self._neos[i]
100
101         if neo_by_name=='' or neo_by_name == None:
102            return None
103
104         return neo_by_name
105
106     def query(self, filters=()):
107         """Query close approaches to generate those that match a collection
108
109         This generates a stream of `CloseApproach` objects that match all o
110         provided filters.
111
112         If no arguments are provided, generate all known close approaches.
113
114         The `CloseApproach` objects are generated in internal order, which
115         guaranteed to be sorted meaningfully, although is often sorted by t
116
117         :param filters: A collection of filters capturing user-specified cr
118         :return: A stream of matching `CloseApproach` objects.
119         """
120         # ELABORATE: Generate `CloseApproach` objects that match all of the
```

```
121         self.filters = filters
122
123         for approach in self._approaches:
124             if self.filters['date'] and self.filters['date'] != datetime.da
125                 continue
126             if self.filters['start_date'] and not (self.filters['start_date
127                 continue
128             if self.filters['end_date'] and not (self.filters['end_date'] >
129                 continue
130             if self.filters['distance_min'] and not self.filters['distance_
131                 continue
132             if self.filters['distance_max'] and not self.filters['distance_
133                 continue
134             if self.filters['velocity_min'] and not self.filters['velocity_
135                 continue
136             if self.filters['velocity_max'] and not self.filters['velocity_
137                 continue
138             if self.filters['diameter_min'] and approach.neo.diameter != ''
139                 if not (self.filters['diameter_min'] <= float(approach.neo.
140                     continue
141             if self.filters['diameter_max'] and approach.neo.diameter != ''
142                 if not (self.filters['diameter_max'] >= float(approach.neo.
143                     continue
144             if self.filters.get('hazardous')==True and not approach.neo.haz
145                 continue
146             if self.filters.get('hazardous')==False and not approach.neo.ha
147                 continue
148
149             yield approach
150
```

▸ .github/workflows/manual.yml

▸ README.md

▸ extract.py

▸ filters.py

▸ helpers.py

▸ main.py

▸ models.py

▸ tests/__init__.py

▸ tests/test_data_files.py

- tests/test_database.py
- tests/test_extract.py
- tests/test_limit.py
- tests/test_python_version.py
- tests/test_query.py
- tests/test_write.py
- write.py

RETURN TO PATH

Rate this review

START