UDACITY

Return to Classroom

Investigating Near-Earth Objects

REVIEW

CODE REVIEW 2

HISTORY

Meets Specifications

Congratulations! 🎉 You've done a splendid job on this project! You understand all parts of this project as well. You properly make use of the csv and json libraries to read and write data. All the required classes, methods, functions, and attributes are correctly implemented. The code can work correctly and pass all test cases. Your code is also well structured and formatted. It mostly follows the guidelines of PEP 8 and docstring conventions of PEP 257.

Feel free to improve your code according to the comments on this page and **Code Review** especially the linking process in the `NEODatabase` class.

Enjoy learning and all the best!

Functionality

The `NearEarthObject` class represents a near-Earth object.

- The constructor assigns attributes for:
  - `designation`: The NEO's primary designation.
  - `name`: The NEO's IAU name (could be empty, or `None`)

    - `diameter`: The NEO's diameter, in kilometers, or NaN.
    - `hazardous`: Whether the NEO is potentially hazardous
    - `approaches`: A collection of this NEO's `CloseApproach`es (initially an empty collection). The `CloseApproach` class represents a close approach to Earth by an NEO.
- The constructor assigns attributes for:
- `time`: The date and time, in UTC, at which the NEO passes closest to Earth.
- `distance`: The nominal approach distance, in astronomical units, of the NEO to Earth at the closest point.
- `velocity`: The velocity, in kilometers per second, of the NEO relative to Earth at the closest point.
- `neo`: A reference to the `NearEarthObject` that is making the close approach (initially None).
- An additional attribute, to store the NEO's primary designation before the `CloseApproach` is linked to its `NearEarthObject`

Additionally, each of these classes should implement a `__str__` method that produces a human-readable description of the contents of the object.

The `NearEarthObject` class representing a near-Earth object and the `CloseApproach` class representing a close approach to Earth by an NEO are correctly completed.

You have error-handling code for the case in which an NEO has no name or no diameter. If there's no name, the `name` attribute is `None`. If there's no diameter, the `diameter` attribute is `float('nan')`. Well done!

The `load_neos` function loads NEO data from a CSV file.

- The function opens the given file for reading.
- The function uses the `csv` module to parse the file contents into a standard Python data structure (e.g. list, dict, etc).
- The function converts this raw data into a collection of `NearEarthObject`s
- The function returns a collection of `NearEarthObject`s

The `load_approaches` method loads close approach data from a JSON file.

- The function opens the given file for reading.
- The function uses the `json` module to parse the file contents into a dict.
- The function converts this raw data into a collection of `CloseApproach` objects.
- The function returns a collection of `CloseApproach` objects.

Data from the extraneous columns (CSV) and fields (JSON) shouldn't be bound to the constructed `NearEarthObject`s and `CloseApproach`es.

✅ The `load_neos` function loads NEO data from a CSV file.

✅ The `load_approaches` function loads close approach data from a JSON file.

Great job using csv.DictReader. It's a clean interface since the CSV file already has a header row.

The NEODatabase constructor captures and preprocesses a collection of NEOs and close approaches.

- The constructor captures and saves its arguments, a collection of `NearEarthObject` and a collection of `CloseApproach` es.
- The constructor precomputes auxiliary data structures to assist with the `get_neo_by_designation` and `get_neo_by_name` methods.
- At the end of the constructor, the `.neo` attribute is set on each close approach to the matching `NearEarthObject` .
- At the end of the constructor, the `.approaches` attribute is populated for each `NearEarthObject` with a collection of its close approaches.

The `get_neo_by_designation` method fetches an NEO by its primary designation, or returns None if no matches are found.

The `get_neo_by_name` method fetches an NEO by its name, or returns None if no matches are found.

✅ The NEODatabase constructor captures and preprocesses a collection of NEOs and close approaches.

- ✅ The constructor captures and saves its arguments, a collection of `NearEarthObject` and a collection of `CloseApproach` es.
- ❗ The constructor precomputes auxiliary data structures to assist with the `get_neo_by_designation` and `get_neo_by_name` methods.
- ✅ At the end of the constructor, the `.neo` attribute is set on each close approach to the matching `NearEarthObject` .
- ✅ At the end of the constructor, the `.approaches` attribute is populated for each `NearEarthObject` with a collection of its close approaches.

✅ The `get_neo_by_designation` method fetches an NEO by its primary designation, or returns None if no matches are found.

✅ The `get_neo_by_name` method fetches an NEO by its name, or returns None if no matches are found.

You've linked together `neos` and `approaches` correctly. however, your implementation is time consuming. You are encouraged to create a dictionary mapping `designatioin` to `neo` and then you just need only one for loop.

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
```

The `create_filters` function produces a collection that can be used by the `query` method to perform a search of close approaches.

- The function respects the `--date` filter mode.
- The function respects the `--start-date` filter mode.
- The function respects the `--end-date` filter mode.
- The function respects the `--min-distance` filter mode.
- The function respects the `--max-distance` filter mode.
- The function respects the `--min-velocity` filter mode.
- The function respects the `--max-velocity` filter mode.
- The function respects the `--min-diameter` filter mode.
- The function respects the `--max-diameter` filter mode.
- The function respects the `--hazardous` and `--not-hazardous` filter modes.

The filters accurately produce results based on the user-specified options.

You've created a dictionary to store the input parameter.
It's better to create subclasses of `AttributeFilter` and then create filters according to project instructions in the `README.md` or classroom. Here is an example:

```
class DateFilter(AttributeFilter):
    @classmethod
    def get(cls, approach):
        return approach.time.date()
```

```
filters = []
if date:
    filters.append(DateFilter(operator.eq, date))
```

The `NEODatabase`'s `query` method generates a stream of `CloseApproaches` that match the filters returned by `create_filters`.

- A `CloseApproach` is generated if and only if it passes all predicates.
- The method generates a stream of matching results, and doesn't precompute all matching results up front.

✅ The `query` method generates a stream of `CloseApproaches` that match the filters returned by `create_filters`.

- ✅ A `CloseApproach` is generated if and only if it passes all predicates.
- ✅ The method generates a stream of matching results, and doesn't precompute all matching results up front.

If you create filters according to the example in the project instructions, you can simply use the all function and list comprehensions like `if all(f(approach) for f in filters):`.

The `limit` function slices an iterator to its first `n` elements, at most.

- **The function is correct even if the first argument isn't an in-memory buffered aggregate data type (i.e. list, tuple, etc). That is, the function doesn't slice directly into the iterator.**
- **The function doesn't limit the results if the second argument is 0 or None.**

✅ The `limit` function slices an iterator to its first `n` elements, at most.

- ✅ The function is correct even if the first argument isn't an in-memory buffered aggregate data type (i.e. list, tuple, etc). The function doesn't slice directly into the iterator.
- ✅ The function doesn't limit the results if the second argument is 0 or None.

Great job using itertools.islice. Well done!

The `write_to_json` function writes a stream of `CloseApproach` objects to a file in JSON format.

- **The function opens the file for writing.**
- **The function prepares the stream of results according to the JSON output format specification in the instructions.**
- **The function uses the `json` module to write the data to the file.**

The two functions `write_to_json` and `write_to_csv` are correctly implemented.

Great job using csv.DictWriter. It is clean since the function's starter code includes a collection of field names.

Submitted code passes all test cases and runs without error.

✅ Code passes all test cases and runs without error. Well done!

```
----------------------------------------------------------------------
Ran 73 tests in 13.278s

OK
```

You can significantly reduce the run time by improving the implementation of the linking process in the `NEODatabase` class.

Style (Mechanics)

Submitted code follows the guidelines of PEP 8 - the Style Guide for Python.

✅ Code mostly follows the guidelines of PEP 8. Well done!
There are a plethora of command-line tools (`pycodestyle` , `pylint`) or websites (http://pep8online.com/) to check your code.
For example, You can install pycodestyle with `pip install pycodestyle` and check your code with `pycodestyle ./` .

I've checked your code with `pycodestyle` . Here are some tiny issues:

```
./database.py:144:45: E225 missing whitespace around operator
./database.py:144:45: E712 comparison to True should be 'if cond is True:' or 'if cond:'
./database.py:144:80: E501 line too long (90 > 79 characters)
./database.py:144:83: E712 comparison to True should be 'if cond is True:' or 'if cond:'
./database.py:145:25: W291 trailing whitespace
./database.py:146:45: E225 missing whitespace around operator
./database.py:146:45: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
./database.py:146:80: E501 line too long (92 > 79 characters)
./database.py:146:84: E712 comparison to False should be 'if cond is False:' or 'if not cond:'
./database.py:147:25: W291 trailing whitespace
./database.py:148:1: W293 blank line contains whitespace
./extract.py:1:80: E501 line too long (83 > 79 characters)
./extract.py:10:80: E501 line too long (80 > 79 characters)
./extract.py:24:80: E501 line too long (87 > 79 characters)
./extract.py:33:14: E111 indentation is not a multiple of 4
./extract.py:33:14: E117 over-indented
./extract.py:33:51: E231 missing whitespace after ','
./extract.py:33:80: E501 line too long (146 > 79 characters)
./extract.py:33:87: E231 missing whitespace after ','
./extract.py:33:117: E231 missing whitespace after ','
```

Submitted code follows the docstring conventions of PEP 257.

Each module contains a module-level comment describing the purpose of the module. Complex functions, classes, and methods include a docstring annotating the primary action of the callable in the imperative mood, any additional clarifications, followed by descriptions of parameters and return values.

✅ Code follows the docstring conventions of PEP 257. Each module contains a module-level comment describing the purpose of the module. Well done!

Feel free to install `pydocstyle` with `pip install pydocstyle` and check your code with `pydocstyle ./` .

There are no `# TODO` comments left in the submitted code. Portions of comments that say `# ELABORATE` have been filled in with a description of the corresponding code.

✅ There are no `# TODO` comments left. Well done!

Keep in mind you can search for text with `grep`: `grep -nrI TODO a_directory/`. You can find more about `grep` here.

Style (Design)

Attributes of `NearEarthObject`s and `CloseApproach`es are captured in the constructor from the supplied arguments.

Instances of `NearEarthObject` don't have attributes of individual close approaches.

Instances of `CloseApproach` don't have attributes of the associated NEO (except for the primary designation needed to initially link the close approach to its NEO.

Standalone functions are used when the functional operation doesn't depend on external state and does not conceptually belong on an object.

Represents concrete data (buffered file contents, static collections of NEOs or close approaches, auxiliary data structures) as concrete.

Represents streaming data (close approaches that match criteria, limited stream of results) as streaming.

The logic backing the filter system is consistent and doesn't contain excess duplicated code.

DOWNLOAD PROJECT

2 CODE REVIEW COMMENTS ›

RETURN TO PATH

Rate this review

START