Project 4 Crime Time

CPE 101: Fundamentals of Computer Science
Winter 2019 - Cal Poly, San Luis Obispo

Purpose

To gain experience writing a class and instantiating objects in a full program, implementing sort and search algorithms, as well as using Python file I/O functions.

Description

For this assignment, you will write a program that reads and writes records to a file, which represents one of the most basic forms of persistent data storage.

You are provided two tab-separated value (TSV) files to be read:

- `crimes.tsv` contains a one-line header and 155,889 crime descriptions
  - Header: `ID Category Description`
  - e.g. `150011660 ROBBERY "ROBBERY ON THE STREET, STRONGARM"`
  - You may ignore the `Description` field for this assignment.
- `times.tsv` contains the time and date information for the crimes in `crimes.tsv`
  - Header: `ID DayOfWeek Date Time`
  - e.g. `150011660 Monday 01/05/2015 02:40`

Download the above files from Polylearn.

Your program will write a new file called `robberies.tsv` with data combined from the provided files, linked together by ID:

- Header: `ID Category DayOfWeek Month Hour`
- e.g. `150011660 ROBBERY Monday January 2AM`

To allow your program to produce more meaningful stats, all crimes processed will be of category `ROBBERY` only. All other categories should be filtered out when writing the file.

Implementation

In addition to `main`, your program must have, at a minimum, the following structure.

`class Crime`

Your program must store each line of data read from `crimes.tsv` in an object whose type is a class called `Crime`. This class must have the following attributes:

`crime_id`	as read from `crimes.tsv`
`category`	as read from `crimes.tsv`
`day_of_week`	as read from `times.tsv`
`month`	modified from `times.tsv` to be a full word
`hour`	modified from `times.tsv` to be in AM/PM format

`__init__(self, crime_id, category)`

The constructor need only take an ID and category as inputs. All other required attributes should be initialized to `None`.

`__eq__(self, other)`

Return `True` when both Crime objects have the same ID and `False` otherwise.

`__repr__(self)`

Return a string representation of the `Crime` object. This representation should match that of a line to be output to `robberies.tsv`. Use the `\t` character to place a tab between words in a string and `\n` for the newline character.

`set_time(self, day_of_week, month, hour)`

Given a day of the week (as a string) and integers for a month and hour, update the appropriate attributes of the `Crime` object by calling this method. The arguments to this method will derive from `times.tsv` and will be of the following format:

`day_of_week`	a string containing a day of the week
`month`	an integer between 1 and 12
`hour`	an integer between 0 and 23

This method will be called when a `Crime` object needs to be updated with time data and should transform the `month` and `hour` integer arguments to their appropriate string representations (see above) before updating the object's attributes. You will lose style points during the manual style check if you simply use a series of conditional statements to convert months and hours to strings. Instead, try to think of more inventive ways (e.g. using lists or `range`) to solve this problem.

`create_crimes(lines)`

This function takes as input a list of strings, each a line read from `crimes.tsv` (not including the header) and returns a list of `Crime` objects, one for each unique `ROBBERY` found. There may be duplicate crimes (with the same ID) in the data; your program should only create one `Crime` object for each unique ID.

`sort_crimes(crimes)`

This function takes as input a list of `Crime` objects and returns a list of `Crime` objects sorted by ID number using **selection sort**.

`update_crimes(crimes, lines)`

This function takes as input a list of sorted `Crime` objects and a list of strings, each a line read from `times.tsv` (not including a header) and returns a list of `Crime` objects with updated attributes. `Crime` objects are located using `find_crime`.

`find_crime(crimes, crime_id)`

This function takes as input a list of sorted `Crime` objects and a single crime ID integer and returns the `Crime` object with that ID. To receive full credit, this function must use **binary search** to find and return the `Crime` object; however, it is recommended that you first implement the simpler but slower linear search to get the program working and later return to replace it with binary search.

Output

In addition to writing a `robberies.tsv` file, your program must print the following crime stats (underscores indicate where data must be filled in by your program):

```
NUMBER OF PROCESSED ROBBERIES: __
      DAY WITH MOST ROBBERIES: __
    MONTH WITH MOST ROBBERIES: __
     HOUR WITH MOST ROBBERIES: __
```

Coding Style, Docstrings and comments

From this project, you are required to follow the standard coding style that the Python community uses, which were presented on the lecture and references to the style presented on the lecture slides.

- Use snake_case for function/method and variable names. Do not use one letter variable names except within loops (for and while).
- Use CamelCase for class names.
- Use ALL UPPER CASE SNAKE_CASE name for constants.

- Use 4 spaces for each indentation level.
- Each line of code can not be no longer than 79 characters.
- Each function body should not be no longer than 40 lines excluding comments and docstrings (This is the course specific coding style).

Add a module docstring at the top of every python file you write. It should look like:
In crimetime.py:

```
“”” The module file for Project 4 Crimetime
CPE 101
Winter 2019
Author:
   Your Name goes here.
“””
```

In crimetime_tests.py:

```
“”” The test file for Project 4 Crimetime
CPE 101
Winter 2019
Author:
   Your Name goes here.
“””
```

Add a function docstring just under the function header for every function. The docstring for function should look like this:
For example,

```
find_crime(crimes, crime_id):
   “””Finds a crime from a crime id.

   Args:
      crimes (list): a list of Crime objects
      crime_id (int): a crime id

   Returns:
      Crime object : A Crime object corresponding to the crime_id.
   “””
   # Write the function body under here.
```

Testing

You are required to write at least 3 tests for each function that returns a value (i.e. is not an I/O function). Since we are emphasizing test-driven development, you should write tests for each function first. In doing so, you will have a better understanding as to what the functions take as input and produce as output, which makes writing the function definitions easier.

In addition to module-level functions, you must also create at least 2 Crime objects in `tests.py` and update them with `set_time`. You must use `assert` statements to test these objects with the `==` and `!=` operators, as well as each object's string representation. Note that the automated style parser will not search for these test cases but points will be deducted during the manual style check if they are not present.

You should use the `diff` command to compare your `robberies.tsv` file against the provided `expected-robberies.tsv` file.

Make no assumptions about the order of the entries in `crimes.tsv` and `times.tsv`. Each project submission will be evaluated using shuffled versions of these files.

Submission

Zip `crimetime.py` and `crimetime_tests.py into one zip file and submit it to polylearn.`