Project 4 Crime Time

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 Canvas.

Example Usage

Your crometime.py shall expect two command-line arguments from the user, the name of a tsv file containing crimes, and the name of a tsv file containing the information about the time the crimes occurred.
This is how your program will be used ($ is the command-line prompt), but please do not hard code the names of the files:

```
$ python3 crimetime.py crimes.tsv times.tsv
```

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. `->` after a function header indicates the return value type of the function.

`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.

`create_crimes(lines)->list`

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)->list`

This function takes as input a list of `Crime` objects and returns a list of `Crime` objects sorted by ID number using either **selection sort** or **insertion sort**. Import `copy` module and use the `copy()` function provided in the module to shallow copy the list `crimes` into a new list, then sort the items in the new list. Return the new list.

`set_crimetime(crime, 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 function. The arguments to this method will derive from `times.tsv` and will be of the following format:

- `crime` `an object of Crime`
- `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 function 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 probably lose coding style points 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, `range`,or `enumerate`) to solve this problem.

`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 updates attributes of `Crime` objects in the list. Call `set_crimetime` to update a `Crime` object. `Crime` objects are located using `find_crime`.

`find_crime(crimes, crime_id)->Crime`

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: __
```

Testing

You should use the `diff` command to compare your `robberies.tsv` file against the provided `expected-robberies.tsv` file. The diff command outputs the differences between two files on the screen.

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.

You will do the following tests in Lab 8, not in this assignment, but make sure that your program outputs expected results (both robberies.tsv and the crime stats on the screen).

You are required to use the `unittest` module to test your code. Please read the documentation on the `unittest` module available on Canvas.
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.

You must use the `unittest` module to test your functions that return some values. Use `self.assertEqual(expr1, expr2)` to test equality between returned values and expected values. You can use `self.assertTrue(expr)` to test if a function returns True. You can use `self.assertFalse(expr)` to test if a function returns False. You can use `self.assertAlmostEqual(first, second)` to test if two float values are almost equal.

Peer Review

You are required to review two of your classmates' work. This part will be a part of Lab 8, but you will be accessing programs to review, and will be submitting your reviews in the Project 3 section on Canvas.

Submission

Submit `crimetime.py to Canvas.`