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 hardcode the names of the files:

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

Print a nice error message such as
"`Invalid command-line arguments provided. Usage: crimetime.py crimes.tsv times.tsv robberies.tsv`"
when the user typed an incorrect number of arguments and quit the program.

Your program must also print a nice error message and exit when your program fails to open the first and the second file (e.g.`crimes.tsv` *and* `times.tsv`). This means your program needs to handle IOError exceptions.

Your program will create a new file whose name will be specified by the user with the third argument to the program *crimetime.py*. Let's suppose that the file is called `robberies.tsv`. The file will be written 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.
Note:-> after a function header indicates the return value type of the function. A datatype following : indicates the datatype of the argument.

`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(int)` as read from `crimes.tsv`
- `category(str)` as read from `crimes.tsv`
- `day_of_week(str)` as read from `times.tsv`
- `month(str)` modified from `times.tsv` to be a full word
- `hour(str)` modified from `times.tsv` to be in AM/PM format

`__init__(self, crime_id:int, category:str)`

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

`__eq__(self, other)->bool`

Return `True` when both Crime objects have the values for all the attributes.

`__repr__(self)->str`

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)->list`

This function takes as input a list of strings, each of 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)->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**. **This means that you may not use Python builtin functions/methods to sort the list.** 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:Crime, day_of_week:str, month:int, hour:int)->None
```

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 not return any values. But the crime object will be changed in this function because objects of a custom class are mutable. This means that the function has a side-effect. Make sure that you describe the aspect of this function in the docstring.

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: list, lines: list)->None`

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. `Crime` objects are located using `find_crime`. Call `find_crime` to find the crime object whose id matches the crime id contained in each of the lines. Call `set_crimetime` to update the `Crime` object. This function does not return anything, but it mutates the list of crimes passed as one of the arguments. Make sure that you write about it in the docstring.

`find_crime(crimes:list, crime_id:int)->int`

This function takes as input a list of sorted `Crime` objects and a single crime ID integer and returns the position (index) of the `Crime` object with that ID in the list of crimes. **Returns -1 if the crime with the id is not found.** To receive full credit, this function must use **binary search** to find 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. For example, find_crime([Crime(1, 'ROBBERY'), Crime(2, 'ROBBERY')], 2) shall return 1.

Output

In addition to writing a new file (e.g. `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

Unit Test

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.

1. Create a new file crimetime_tests.py
2. At the top of the file, add this line: import unittest
3. Add this line below the previous line:
    a. from crimetime import *
        i. or
    b. import crimetime

i. # if you choose to do this, you have to append “crimetime.” to function names and the class name:
    1. c = crimetime.Crime(1, ‘ROBBERY’)
    2. crimetime.set_crimetime(c, ‘Tuesday’, 1, 14)
ii. But this is safer than the option above.

4. At the bottom of the file, add the following lines:

```
if __name__ == ‘__main__’:
    unittest.main()
```

6. Above the `if __name__ == '__main__':` line, add the following class with functions:

```
class MyTest(unittest.TestCase):
    def test_create_crimes(self):
```

- Write 1 test to test the function, create_crimes(lines)->list.
- To test, use this builtin function: self.assertEqual(test_val, expected_val), where test_val is the value returned from the function you are testing, and expected_val is the value you are expecting from the function.
- Create a list of strings (a string corresponds to one line in the input file) of crime information as the argument lines, and pass it to the create_crimes function. The list must contain at least 5 strings (information about 3 crimes), and it needs to include multiple crimes of ROBBERY with some duplicates and crimes other than ROBBERY.
- Create a list of Crime objects which are supposed to be produced as a result of the create_crimes function, and compare this list with the list returned by the function using self.assertEqual().
- For example,
    - lines=[‘150011660\tROBBERY\tROBBERY ON THE STREET, STRONGARM\n’,
      ‘150022065\tNON-CRIMINAL\tAIDED CASE, DOG BITE\n’,
      ‘150023994\tROBBERY\tROBBERY, BODILY FORCE\n’]
      expected=[Crime(150011660, ‘ROBBERY’), Crime(150023994, ‘ROBBERY’)]
      self.assertEqual(create_crimes(lines), expected)

```
    def test_sort_crimes(self):
```

- Write 2 tests to test the function, sort_crimes(crimes)->list.
- To test, use this builtin function: self.assertEqual(test_val, expected_val), where test_val is the value returned from the function you are testing, and expected_val is the value you are expecting from the function.
- Create lists of Crime objects, and pass it to the sort_crimes function. Each list must contain at least 5 Crime objects. Create two lists: a list of Crimes that are already in ascending order, another list of Crimes that are randomly ordered.

- Create two sorted lists of Crime objects which are supposed to be produced as a result of the sort_crimes function, and compare these lists with the lists returned by the function using self.assertEqual().
- For example,
    - crimes=[Crime(2, ‘ROBBERY’), Crime(3, ‘ROBBERY’), Crime(1, ‘ROBBERY’)]
    - expected=[Crime(1, ‘ROBBERY’), Crime(2, ‘ROBBERY’), Crime(3, ‘ROBBERY’)]
    - self.assertEqual(sort_crimes(crimes), expected)

def test_set_crimetime(self):

- Write 3 tests to test the function, set_crimetime(crime, day_of_week, month, hour).
- To test, use this builtin function: self.assertEqual(test_val, expected_val), where test_val is the value returned from the function you are testing, and expected_val is the value you are expecting from the function.
- Create 3 Crime objects: e.g. c1 = Crime(150001, ‘ROBBERY’). And pass each of the objects to set_crimetime() function, with day_of_week, month, and hour. For example, set_crimetime(c1, ‘Wednesday’, 7, 17).
- Create 3 Crime objects with the same three crime_ids as the previous three Crime objects with day_of_week, month, and hour populated with expected values. Compare these three Crime objects with the previous three objects using self.assertEqual().
- For example,
    - c1 = Crime(150001, ‘ROBBERY’)
    - set_crimetime(c1, ‘Wednesday’, 7, 17)
    - c4 = Crime(150001, ‘ROBBERY’)
    - c4.day_of_week = ‘Wednesday’
    - c4.month = ‘July’
    - c4.hour = ‘5PM’
    - self.assertEqual(c1, c4)

def test_update_crimes(self):

- Write 1 test to test the function, update_crimes(crimes, lines).
- To test, use this builtin function: self.assertEqual(test_val, expected_val), where test_val is the value returned from the function you are testing, and expected_val is the value you are expecting from the function.
- Create a sorted list of Crime objects: e.g.
    - crimes=[Crime(1, ‘ROBBERY’), Crime(2, ‘ROBBERY’), Crime(3, ‘ROBBERY’)]
- Create a list of strings as lines from the other input file: e.g.
    - lines=[‘1 Tuesday 01/06/2015 16:53’, ‘2 Saturday 01/03/2015 14:06’, ‘3 Thursday 01/08/2015 15:30’]

- Pass the list of crimes and the list of strings to update_crimes(crimes, lines).
- Create Crime objects having the same crime_ids as the objects in the list of Crime objects: the crimes, and populate the objects with expected information: crime_id, category, day_of_week, month, and hour.
- Compare Crime objects after calling the function, update_crimes(crimes, lines), using self.assertEqual(): e.g.
    - update_crimes(crimes, lines)
    - c1 = Crime(1, 'ROBBERY')
    - c1.day_of_week = 'Tuesday'
    - c1.month = 'January'
    - c1.hour = '4PM'
    - self.assetEqual(crimes[0], c1)

def test_find_crime(self):

- Write 3 tests to test the function, find_crime(crimes, crime_id)->int.
- To test, use this builtin function: self.assertEqual(test_val, expected_val), where test_val is the value returned from the function you are testing, and expected_val is the value you are expecting from the function.
- Create a sorted list of Crime objects: e.g.
    - crimes=[Crime(1, 'ROBBERY'), Crime(2, 'ROBBERY'), Crime(3, 'ROBBERY')]
- Call the function find_crime(crimes, crime_id), and compare the return value of the function with the expected value using self.assertEqual(): e.g.
    - self.assertEqual(find_crime(crimes, 3), 2)

7. To run the test, type the following command on a terminal or gitbash:
    a. On Mac/Unix/Linux: python3 crimetime_tests.py
    b. On Windows: py crimetime_tests.py

Integration Test

Do not forget to test run your program before you submit it!
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.

Peer Review

You are required to review two of your classmates' work. This part will be a part of Lab 9, but you will be accessing programs to review, and will be submitting your reviews in the Project 4 section on Canvas.

Submission

Submit `crimetime.py and crimetime_tests.py` to `Canvas.`