# 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 with the following tab-separated value (TSV) files, which all contain a one-line header.

- `crimes.tsv` contains 155,889 crime descriptions
- `times.tsv` contains the time and date information for the crimes in `crimes.tsv`
- `expected-robberies.tsv` contains the expected output of the program for `diff` comparison

Download the files here:

http://users.csc.calpoly.edu/~dkauffma/101/crimetime.zip

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

All crimes processed will be of category `ROBBERY` only. All other categories should be filtered out when processing the data.

## Implementation

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

**class Crime**

Your program must store each robbery read from `crimes.tsv` in an object whose type is a class called `Crime`, which must have the following attributes.

| Attribute | Source | Type | Example |
|---|---|---|---|
| `crime_id` | `crimes.tsv` | `int` | `12345` |
| `category` | `crimes.tsv` | `str` | `ROBBERY` |

| | | | |
|---|---|---|---|
| day_of_week | times.tsv | str | Monday |
| month | times.tsv | str | January |
| hour | times.tsv | str | 12AM |

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

Initialize `crime_id` and `category` attributes; all other required attributes should be initialized to `None`.

```
__eq__(self, other) -> bool
```

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

```
__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.

```
set_time(self, day_of_week: str, month: int, hour: int) -> NoneType
```

Given a day of the week and integers for a month and hour, update the appropriate attributes of the calling `Crime` object. The arguments to this method will derive from `times.tsv`, with `month` an integer between 1 and 12 and `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 before updating the object's attributes. While some branching will be necessary, do not simply write a branch for every possible conversion; instead, think of more inventive ways, such as using lists and `range`, to solve this problem.

## Module Functions

```
main()
```

Using additional functions as needed, perform the following steps:

- Read lines from `crimes.tsv` and `times.tsv`
- Create, sort, and update `Crime` objects, one for each robbery found
- Write all sorted and updated robbery data to `robberies.tsv`
- Print crime stats about the robberies (see below)

If you receive a `UnicodeDecodeError` when **reading** a file, add the `encoding` argument to `open`:

```
open("crimes.tsv", "r", encoding = "utf-8")
open("times.tsv", "r", encoding = "utf-8")
```

```
create_crimes(lines: List[str]) -> List[Crime]
```

Given a list of strings, each a line read from `crimes.tsv` (not including the header), return a list of `Crime` objects, one for each robbery found. Although there may be crimes with the same ID in the data, this function should only create one `Crime` object for each unique ID.

```
sort_crimes(crimes: List[Crime]) -> List[Crime]
```

Sort and return the given list of `Crime` objects by ID using **Selection Sort**.

```
update_crimes(crimes: List[Crime], lines: List[str]) -> NoneType
```

Given a list of sorted `Crime` objects and a list of strings, each a line read from `times.tsv` (not including a header), update the time-related attributes in each object, using `find_crime` to efficiently find objects with a specific ID.

```
find_crime(crimes: List[Crime], crime_id: int) -> Union[Crime, NoneType]
```

Return the `Crime` object with the given ID by locating it using **Binary Search**. If no object in the list has the given ID, return None.

```
get_mode(List[Any]) -> Any
```

Return the most common value in the given list. If two values are equally common, return the value that occurs first in the list.

### Output

In addition to writing a `robberies.tsv` file, compute and 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

In addition to module-level functions, you must write test cases for methods of the `Crime` class, which will require calling the constructor to test `__init__`, using the == operator to test `__eq__`, and casting objects with `str` to test `__repr__`. Other functions that do not return a value, such as `update_crimes` and `set_time`, may be tested by using `assert` statements after they are called to ensure that relevant attributes were updated correctly.

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`. Submissions will be evaluated using shuffled versions of these files.

## Submission

On a CSL server with `crimetime.py` and `cttests.py` in your current directory:

```
/home/dkauffma/casey.exe 101 crimetime
```