Due Sunday July 18th, 11:59pm.

Background

The purpose of this assignment is to explore dictionaries and file reading and writing. We will be reading in some data about hurricanes, creating a structure that groups storms by year, and then checking for most severe storms or years by various metrics.

Dictionaries give us an enriched way to store values by much more than just sequential indexes (as lists gave us); we identify key-value pairs, and treat keys like indexes of various other types. Though unordered, dictionaries help us simplify many tasks by keeping those key-value associations. Each key can only be paired with one value at a time in a dictionary.

When a file contains ASCII or Unicode text in it, we can readily write programs to open the file and compute with its contents. It turns out that reading and writing text files gives our programs far more longevity than open-to-quit; we can store data and results for later, save user preferences, and all sorts of things. We will be reading text files that happen to be in the CSV format.

Refer to our "Project Basics" in its entirety for information on naming your file, performing testing, grading details, and more. Consider it a part of this project's requirements, because it is.

- Project Basics document: https://mymasonportal.gmu.edu/bbcswebdav/xid-210009785_1
- sample files/definitions: https://mymasonportal.gmu.edu/bbcswebdav/xid-210099592_1
- Project Five tester file: https://mymasonportal.gmu.edu/bbcswebdav/xid-210099591_1

Grading Rubric

```
Code passes shared tests:     90
Well-documented/submitted:    10
TOTAL:                       100  +5 extra credit
```

What's allowed?

here is the exhaustive list of things you can use on the project.

- all basic expressions/operators, indexing/slicing
- all basic statements: assignment, selection, and loop statements, break/continue, return
- functions: `len(), range(), min(), max(), int(), str(), enumerate()`
- file reading: `open(), close(), read(), readline(), readlines(), with` syntax
- dictionaries: all methods listed in our slides on that one chart.
- methods: lists: `.remove(), .insert(), .append(), .extend(), .pop(), .popitem()`
  strings: `.strip(), .split(), .join(), .remove(), .insert(), .lower()`
- `sorted(), sort(), reversed(), reverse()`
- `list(), tuple(), set(), dict().`

This means that…

- you can't call anything not listed above. Focus on applying these functions to solve the task.
- you can't import any modules for this project. (so you can't import csv either – but it isn't that helpful)

Scenario

Hurricanes can wreak havoc when they make landfall; they can cause deaths and monetary damage, so we want to explore what has happened in the past. We've got some data stored in a comma-separated-values file; only `read_file` needs to interact with files, and all others will use our required structure to describe storms and overall databases of storms.

Definitions

`CSV file`: This is a file containing ASCII text where each line in the file represents one record of information, and each piece of info in the record is separated by a single comma. (Luckily the values won't ever contain commas themselves for this project). The very first line is the "header" row, which names the columns but is not part of the data. Here is a very small sample file that can be used in our project. It has extra columns in it. Note: the extension you use has no effect on the contents; you can edit them with your code editor, and you can give them any extension you want without changing the ability of your program.

```
YEAR,NAME,MinPressure,Gender,CATEGORY,deaths,DamageMillions
1992,Andrew,922,M,5,62,66730
2012,Isaac,966,M,1,5,24000
2012,Sandy,942,F,2,159,75000
```

`Storm`: we will always use the following representation for a storm inside our programs: a tuple containing these values in this order. Note that name is a string, but all other fields must be stored as integers. (Also, though a storm occurs in a specific year, notice we don't see it here – that's because it will be represented elsewhere in a database, and duplicated information is rarely a good idea in a database).

```
sample_storm = (name, category, deathtoll, cost_in_millions)
```

`Database`: a "database" of storms can store multiple storms from multiple years. Storm names may get reused in different years, but never in the same year. **Our database must be a dictionary** whose keys are years, and whose values are lists of storm values. Only years with stored storms may be present (no empty lists as values). Storms in the same year must be stored asciibetically by name.

```
sample_db = {
            1992: [('Andrew', 5, 62, 66730)],
            2012: [('Isaac', 1, 5, 24000), ('Sandy', 2, 159, 75000)]
           }
```

Functions

There are many pairs of functions; once you've solved one, the next may be just a cut-paste-edit away from solving both! I'd suggest completely solving one before bothering to duplicate your success.

- don't worry about ties (for costliest, deadliest); though an important detail, it adds a bit much to the project. You can safely add in a preference for earlier-year, asciibetically-earlier names, if you'd like.

`read_file(filename)`: This is the *only* function that needs to deal with reading a file. It will accept the file name, assume it is a `CSV file` as described above (with our hurricane date in the same format as the example, but with any number of rows after the header row). It will open the file, read all the described storms, and correctly create the `database`. It returns that database.

`add_storm (db,year,name,category,deaths,damage)`: This function accepts an existing database, details for a storm, and then it successfully updates the database to include that storm. None is returned.

`merge_databases (db1,db2)`: This function accepts two existing databases, merges them together into a new database with all storms in it, and returns this result.

`storms_by_name(db,name)`: accepts a database and name. It searches for all storms whose `name` matches the `name` argument, and builds/returns another database that only contains those matching storms.
`storms_by_years(db,years)`: accepts a database and a list of years. It builds/returns a new database with all storms from db whose `year` is in the `years` argument.
`storms_by_categories(db,categories)`: accepts a database and a list of categories. It builds/returns a new database with all storms from db whose `category` is in the `categories` argument.

`costliest_storm(db)`: accepts a database, finds the **costliest** storm in it. It returns a **tuple**: `(year,storm)`.
`deadliest_storm(db)`: accepts a database, finds the **deadliest** storm in it. It returns a **tuple**: `(year,storm)`.

`costliest_year(db)`: accepts a database and finds the **costliest** year of storms in it. It returns just the `year`.
`deadliest_year(db)`: accepts a database and finds the **deadliest** year of storms in it. It returns just the `year`.

`deadliest_in_category(db, category)`: accepts a database and category, and finds the deadliest storm of exactly that one category. It returns a tuple: `(year, storm)`. This might be a short definition.
`costliest_in_category(db, category)`: accepts a database and category, and finds the costliest storm of exactly that one category. It returns a tuple: `(year, storm)`. This might be a short definition.

Extra Credit

`years_by_deadliness(db)`: accepts a database, calculates how deadly each year is, and returns those years in descending deadliness (from largest amount to smallest amount).

Examples

Start by pasting the provided code into your project, and then add functions.

```
demo$ pyi yourcode.py
>>> d1 = read_file("file1.csv")
>>> d1
{1992: [('Andrew', 5, 62, 66730)], 2012:
[('Isaac', 1, 5, 24000), ('Sandy', 2, 159,
75000)]}

>>> add_storm(d1,2015,"CS112",4,0,1200)
>>> d1
{1992: [('Andrew', 5, 62, 66730)], 2012:
[('Isaac', 1, 5, 24000), ('Sandy', 2, 159,
75000)], 2015: [('CS112', 4, 0, 1200)]}

>>> add_storm(d1,2015,"CS211",3,0,900)
>>> d1
{1992: [('Andrew', 5, 62, 66730)], 2012:
[('Isaac', 1, 5, 24000), ('Sandy', 2, 159,
75000)], 2015: [('CS112', 4, 0, 1200), ('CS211',
3, 0, 900)]}


>>> merge_databases({1900:[("Early",3,3,3)]},
{2000:[("Recent",4,4,4),("Another",5,5,5)]})
{1900: [('Early', 3, 3, 3)], 2000: [('Another', 5,
5, 5), ('Recent', 4, 4, 4)]}
>>>
>>> merge_databases({1900:[("Early",3,3,3)]},
{2000:[("Recent",4,4,4)],1900:[("Another",5,5,5)]}
)
{1900: [('Another', 5, 5, 5), ('Early', 3, 3, 3)],
2000: [('Recent', 4, 4, 4)]}


>>> storms_by_name(database2,"Bob")
{1979: [('Bob', 1, 1, 70)], 1985: [('Bob', 1, 0,
130)], 1991: [('Bob', 2, 15, 3620)]}
>>> storms_by_name(database3,"C")
{2003: [('C', 2, 1, 1500)], 2004: [('C', 5, 1,
1000)], 2005: [('C', 4, 3, 500)]}

>>> storms_by_years(database3,[2003])
{2003: [('A', 3, 4, 200), ('B', 5, 5, 100), ('C',
2, 1, 1500)]}
>>> storms_by_years(database3,[2004,2005])
{2004: [('A', 4, 9, 900), ('C', 5, 1, 1000)],
2005: [('A', 3, 2, 350), ('C', 4, 3, 500), ('D',
2, 4, 150), ('E', 4, 2, 100), ('F', 5, 3, 200)]}
```

```
>>> storms_by_categories(database3,[3])
{2003: [('A', 3, 4, 200)], 2005: [('A', 3, 2,
350)]}
>>> storms_by_categories(database3,[2])
{2003: [('C', 2, 1, 1500)], 2005: [('D', 2, 4,
150)]}
>>> storms_by_categories(database3,[1])
{}


>>> costliest_storm(database2)
(2005, ('Katrina', 5, 1833, 125000))
>>> costliest_storm(database3)
(2003, ('C', 2, 1, 1500))


>>> deadliest_storm(database2)
(2005, ('Katrina', 5, 1833, 125000))
>>> deadliest_storm(database3)
(2004, ('A', 4, 9, 900))


>>> costliest_year(database2)
2005
>>> costliest_year(database3)
2004


>>> deadliest_year(database3)
2005
>>> deadliest_year(database2)
2005


>>> deadliest_in_category(database2,4)
(1960, ('Donna', 4, 50, 53270))
>>> deadliest_in_category(database3,5)
(2003, ('B', 5, 5, 100))


>>> costliest_in_category(database2,3)
(2005, ('Wilma', 3, 5, 25960))
>>> costliest_in_category(database3,4)
(2004, ('A', 4, 9, 900))


>>> years_by_deadliness(database2)
[2005, 1969, 1955, 2012, 2008, 1972, 1954, 1965,
1999, 1992, 2004, 2003, 1960, 1964, 1961, 2011,
1989, 1996, 1985, 1959, 1970, 1975, 1979, 1983,
1956, 1967, 1991, 1995, 1997, 1966, 1986, 1998,
1950, 1976, 1952, 1963, 1968, 1984, 1993, 1971,
1980, 2002, 1953, 1958, 1974, 1988, 2007, 1977,
1987]
>>> years_by_deadliness(database3)
[2005, 2003, 2004]
>>> years_by_deadliness(database1)
[2012, 1992]
>>>
```