Program 3: Maps

Start Assignment

Due Apr 23 by 11:59pm **Points** 30 **Submitting** a file upload **File Types** py and txt

Working with data files created by other people remains central to the role of data scientists and engineers. In this assignment, students will work (either alone or with a partner) with an historic data file to understand its contents and store it in a method more consistent with modern programming. Specifically, students shall work with the Combat Operations Loss and Expenditure Data, Vietnam (COLEDV) from 1968.

The data contained in the archive represents the collected work of a multitude of warfighters operating under extreme circumstances. Consequently, they contain errors and omissions, but none of these should represent a major problem for our purposes.

This assignment gives student experience with:

- Developing a Python script working with real-world ASCII formatted data files.
- Interpreting a file specification.
- Using maps to simplify some computation.
- Creating formatted output files (i.e. CSV).

Requirements

Converting Fixed Format to CSV

Using maps and the AM68 file, available here on Canvas in the Files/Program 3 section, students shall write a python script to transform it into multiple, smaller, comma-separated-value files. These files will later serve as the basis for tables in a database, but that is an entirely different assignment. Students may name the CSV files they create for this assignment whatever they wish, but the file names should be descriptive.

If one opens an AM68 file in an ASCII text reader (e.g., Notepad, vi, more), it quickly becomes apparent that many of the engagements contained in the file contain a substantial number of multi-digit representations of 00000. This is because the fixed-format nature of the file allocates several columns to these data. We can simplify this in a CSV by simply storing it as a single 0 or even with nothing but a comma to indicate the next field. This will slightly reduce the file size right out of the gate. You should make this its own function and possibly even save the simplified results in an intermediate form just to verify it works.

Note that several Python modules provide functions that actually do something like this for you (e.g., pandas, csv). Feel free to use these tools to simplify this part of the program. The major requirement is successfully interpreting the specification document to extract the relevant data. There is no major technical insight in manually inserting the commas between values or using a third-party module, so feel free to use whichever approach you find most appealing. They require roughly the same work.

Next, if you open the CSV you created in the previous step in something like Excel or LibreOffice Calc, you will see that the unit name appears again, and again, and again, and again. We can simplify this by creating a mapping of unit names, which require many characters to represent, to numbers. We can then replace the long unit name with a much smaller number corresponding to its key in the other CSV table. Instead of recording "1ST40THARTY", the CSV may store something like "5," which takes much less space.

Then, in another CSV, it contains something like: 5,1ST40THARTY. In fact, you will want to save this csv (the number->unit mapping) in its own file. Consider calling it COLEDV-68-unit.csv, but if another format speaks to you, use that. So long as it is obvious what the table contains, it should work.

As was revealed in the lecture, it appears that the file may actually contain a unique mapping already, for the unit name also has a unit number and the subunit name has a corresponding subunit number. For example, A87AA seems mapped to "266TH CML PLT", and AG9AA is mapped to the "1ST INF DIV". These values themselves may serve as keys (assuming they are truly a mapping in the data set). Thus, the resulting output file may contain even fewer values. That is, rather than adding a CSV entry with 1 mapped to the 266th, create a CSV mapping of A87AA to the 266th and save that file. This allows you to strip out all the human-readable unit names from the output file.

After reducing the file size by mapping the unit and subunit names to numbers, we need to create CSV files that map the codes contained in the file to real world descriptions. There are a lot of small tables detailed in the COLED-V documentation. The terrain types, weapon codes, and weather codes represent just a few of these. We need a way of capturing these data in digital form so our application can print these data. You may either create these small CSV map files using your Python script, or you may simply create them using something like Excel or Notepad.

Using the Results

For the final part of the program, students shall create a script that uses the CSV files produced above. This script may either be included at the end of the CSV generation script or a separate script if it better suits the programming/group style. In this script, an interactive prompt shall give the user the option of either entering a range of event numbers or randomly selecting five (5) of them. Using these values, the script shall than unpack the data contained in the CSV and produce human-readable output about the event. It will also save this exact output in an ASCII readable text file (for subsequent submission).

Formatting remains key for the highest marks on this project. Columns should line up, and the output should appear professional. Do not exceed a line width of 80 characters. Make extensive use of f-strings or the format method to make sure things appear clean, and do not fear using several lines to represent the event. That is, consider something like:

```
EVENT 33 of 243,666:
Date: 1968-FEB-17
Subunit:  2ND BN 2ND INF (ANEAA)    Major Command: 1ST INF DIV
Mission: CORDON AND SEARCH                Terrain: CITY/VILLAGE
Weather: CLEAR                          DAY/NIGHT: NA

-- 105mm (ACQUIRED TGTS) --
    HE: 24
XM-629: 2
```

The above engagement is fictitious and does not appear in the data set. It is for output formatting illustration purposes only. I included the unit identification number but omitted the major command identification number.

Your solution should be consistent, for I am not certain the unit identification number adds anything, but if you choose to add it, then you should also add the major command identification number as well. Be consistent unlike the example.

As mentioned earlier, the script shall also save these results in an output file with a .txt extension. The file shall be formatted to match the screen output, and you will submit it here as demonstration of program correctness.

Delivery

Students shall not submit any of the CSV files produced by these scripts. Instead, teams shall submit a single copy of the Python scripts (with each team-member's name on it) as well as a copy of the output file.

Other Notes

Convert the Julian-date contained in the Fixed format AM68 file into either a standard Julian format or expand it to a more human-readable 1968-Jan-30 format. It is perfectly fine to use up more space to represent the date in our output CSV because it adds clarity. Our goal is to eliminate the need to reference back to the National Archive's documentation.

Make certain your script files are cleanly formatted. The grader may subtract points for messy submissions with large blocks of commented-out code, inconsistent variable and function names, or the lack of helper functions.

You will use the scripts and csv files you create here for the next assignment. Making things look clean here will make things better there.

Consider saving headings for each of the columns in the CSV. These should match the descriptions in the COLED-V documentation.