CSE 231 **Fall 2020**

Computer Project #5

This assignment focuses on the design, implementation and testing of a Python program that uses lists and tuples.

It is worth 40 points (4.0% of course grade) and must be completed no later than **11:59 PM on Monday, October 19).**

Assignment Overview

In this assignment, you will practice with files and functions.

Assignment Background

Some have raised questions about Covid-19 data, especially questioning if Covid-19 is causing significant problems. One advantage of learning to program is that you can examine raw data yourself. In this assignment, we compare 2020 deaths versus the number of expected deaths (based on a five-year history). If there are many more deaths than expected, something extraordinary is going on, with Covid-19 being the strongest candidate. Also, this excessive-death data has been used to support the hypothesis that Covid-19 deaths are being undercounted by comparing the Covid-19 count versus the excessive deaths (something that we are not doing in this assignment, but something you could examine on your own).

Assignment Specifications

You will develop a Python program that extracts and displays information from a data file. The basic concept is that you will read the file, extract data of interest, and display it. We provide a file which is in the comma-separated-value format (file extension is `.csv`). You can open the file to look at it using a spreadsheet program such as Excel. The file has death data from around the world as well as data from a few U.S. cities. The file has nearly 7000 entries, but we are only interested in the few dozen related to overall U.S. deaths so far in 2020. We will plot the data crudely using strings (later in the semester we will learn how to make proper plots).

String `split()` method and lists.

The string split() method returns a list, the topic of Chapter 7. A list is an ordered collection of items and has similarities to a string which is an ordered collection of characters. Most useful to us here is that indexing is the same and, of course, starts at zero. For example, the last line of data (after the header) of the deaths.csv file is

```
     United States,,weekly,,,2015-2019 average,12,51,50286,,,
```

It is read in as a string so let's assign the string to a variable named `line` as follows

```
line = 'United States,,weekly,,,2015-2019 average,12,51,50286,,, '
```

If we apply the string `split()` method with comma as an argument, we get a list; let's name the list `line_list` as follows

```
    line_list = line.split(',')
```
if we look at the `line_list` it looks like this list of strings
```
['United States','','weekly','','','2015-2019 average','12','51','50286','','','']
```
The deaths are at index 8 so we can extract that value like this
```
    deaths = line_list[8]
```
That value is a string so we will need to convert it to an int for this program.
Similarly you can extract other values needed for this program by examining the header line to determine the appropriate index—noting, for example, that "deaths" appears at index 8 (remember the first index is 0):
```
country,placename,frequency,start_date,end_date,year,month,week,deaths,expected_deaths,
excess_deaths,baseline
```

open_file () → file pointer

a. This function takes no arguments, prompts for a file name and returns the file pointer to that file. The function will keep asking until a file is found. Use `try-except` and `FileNotFoundError`.
b. Parameters: none
c. Returns : file pointer
d. The function displays a prompt and possibly an error message.
   prompt: `'Please input a file to use: '`
   error message: `'Invalid filename, please try again'`

convert (n)→ int

a. Convert a 5-digit number into a 2-digit number by creating a decimal with one digit to the right of the decimal point and then rounding. Hint: use // and / to create the decimal. For example, if the parameter is 48,771 first convert it to 48.7 and then use the `round()` function to yield 49. Note, this function can be done in one line of code.
b. Parameters: `int`
c. Returns : `int`
d. The function displays nothing.

month_name (n)→ string

a. This function takes as input an int representing a month and returns the three-character, all-caps abbreviation for the month. For example, if the parameter is 4, `'APR'` will be returned. You can use a cascade if `if-elif` statements, or you can use a `list` of month abbreviations using the parameter as an index into the list (with a little arithmetic):
```
MONTHS=["JAN","FEB","MAR","APR","MAY","JUN","JUL","AUG","SEP","OCT","NOV
","DEC"]
```
   If the parameter is **not** between 1 and 12 inclusive, print an error message `"Error in month_name, n="` with the argument n and return `"XXX"`.
b. Parameters: `int`
c. Returns : string
d. The function displays nothing, unless there is an error.

`str_plot(month_int,week_int,n_int,type_str)→ string`

a. This function uses the `type_str` argument to create the two different types of strings for displaying either actual `"deaths"` or `"expected"` deaths. For `"deaths"` the output character will be `"D"` whereas for `"expected"` the output character will be `"E"`. The `month_int` will be converted to a string by calling the `month_name` function. The `n_int` will be converted to a two-digit value by calling the `convert` function. If we name that two-digit value `N`, the output string will have N `"D"`s or N `"E"`s depending on the value of `type_str`. The output string should have the month name, week number `(week_int),` number of deaths (`n_int`) and the string that has N “`D`”s or N “`E`”s in that order. Use the following format for the string returned:
   `"{:3s} ({:2d}) {:6,d}: {}"`

b. Parameters: `int, int, int, string`
c. Returns : `string`
   formatted as specified above
d. The function displays nothing.

`main():`

This function calls `open_file` to get a file pointer and then using that file pointer reads through the file one line at a time, e.g.

```
    for line in fp:
```

The variable `line` is a string. Convert that string to a list of strings using

```
    line_list = line.split(',')
```

as described above. Then index into `line_list` to get values for month, week, deaths, and expected_deaths (remember to convert them to int).
The challenge here is that we only want values for the *United States* and we do **not** want the *average* values (the word *`“average”`* is not in the line) and we do **not** want the values for individual cities (we want only the values with an empty `placename`). Therefore, you must create a Boolean expression to select only those lines. For each appropriate line selected call the `str_plot` function to get a string to print. Note that you will call the str_plot function twice: once with argument `"deaths"` and once with argument `"expected"`.

After opening the file and before reading the file and printing the data, print a blank line and two header lines:

```
Actual deaths (D) vs. Expected Deaths (E)
MTH (WK) Deaths
```

The last header line should be formatted using the following string format:
`"{:3s} ({:2s}) {:6s}"`

Assignment Notes and Hints

1. The coding standard for CSE 231 is posted on the course website:

   http://www.cse.msu.edu/~cse231/General/coding.standard.html

Items 1-9 of the Coding Standard will be enforced for this project.

2. The program will produce reasonable and readable output, with appropriate labels for all values displayed.

3. We provide a `proj05.py` program for you to start with.

4. If you "hard code" answers, you will receive a grade of zero for the whole project. An example of hard coding is to simply print the approximate value of e rather than calculating it and then printing the calculated average.

Suggested Procedure

The last version of your solution is the program which will be graded by your TA.

*You should use the **Mimir** system to back up your partial solutions, especially if you are working close to the project deadline. That is the easiest way to ensure that you won't lose significant portions of your work if your machine fails or there are other last-minute problems.*

Assignment Deliverable

The deliverable for this assignment is the following file:

`proj05.py` – the source code for your Python program

Be sure to use the specified file name and to submit it for grading via the **Mimir system** before the project deadline.

Test 1

```
Please input a file to use: deaths.csv

Actual deaths (D) vs. Expected Deaths (E)
MTH (WK) Deaths
JAN ( 2) 50,693: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
JAN ( 2) 50,668: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
JAN ( 3) 49,475: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
JAN ( 3) 50,118: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
JAN ( 4) 49,016: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
JAN ( 4) 49,916: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
FEB ( 5) 48,765: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
FEB ( 5) 50,651: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
FEB ( 6) 49,371: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
FEB ( 6) 50,511: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
FEB ( 7) 48,909: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
FEB ( 7) 50,175: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
FEB ( 8) 49,063: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
FEB ( 8) 49,827: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
MAR ( 9) 49,405: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
MAR ( 9) 49,491: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
MAR (10) 49,625: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
MAR (10) 49,151: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
MAR (11) 48,521: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
MAR (11) 48,771: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
MAR (12) 49,202: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
MAR (12) 48,316: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
MAR (13) 52,553: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
MAR (13) 47,883: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
APR (14) 60,860: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
APR (14) 47,436: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
APR (15) 66,466: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
APR (15) 47,035: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
APR (16) 64,086: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
APR (16) 46,548: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
APR (17) 60,034: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
APR (17) 46,109: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
MAY (18) 56,254: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
MAY (18) 45,760: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
MAY (19) 54,240: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
MAY (19) 45,424: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
MAY (20) 50,783: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
MAY (20) 45,185: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
```

Test 2

```
Please input a file to use: xxxx
Invalid filename, please try again

Please input a file to use: abcd
Invalid filename, please try again

Please input a file to use: short.csv

Actual deaths (D) vs. Expected Deaths (E)
MTH (WK) Deaths
JAN ( 2) 50,693: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
JAN ( 2) 50,668: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
JAN ( 3) 49,475: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
JAN ( 3) 50,118: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
JAN ( 4) 49,016: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
JAN ( 4) 49,916: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
FEB ( 5) 48,765: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
FEB ( 5) 50,651: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
FEB ( 6) 49,371: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
FEB ( 6) 50,511: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
FEB ( 7) 48,909: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
FEB ( 7) 50,175: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
FEB ( 8) 49,063: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
FEB ( 8) 49,827: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
MAR ( 9) 49,405: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
MAR ( 9) 49,491: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
MAR (10) 49,625: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
MAR (10) 49,151: EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
```

Grading Rubric

```
Computer Project #05                             Scoring Summary
General Requirements:
  ( 6 pts) Coding Standard 1-9
     (descriptive comments, function headers, mnemonic identifiers,
     format, etc...)

Implementation:
 ( 5 pts)  open_file function (no Mimir test)

 ( 5 pts)  str_plot function

 ( 5 pts)  convert function

 ( 5 pts)  month_name function

 ( 9 pts)  Test 1

 ( 5 pts)  Test 2


Note: hard coding an answer earns zero points for the whole project
-10 points for not using main()
```