CSE 231 **Fall 2015**

Computer Project #7

Assignment Overview

This assignment focuses on the design, implementation and testing of Python programs to process data files using lists and functions, as described below.

It is worth 50 points (5% of course grade) and must be completed no later than 11:59 PM on Monday, November 2.

Assignment Deliverable

The deliverable for this assignment is the following file:

`proj07.py` – your source code program

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

Assignment Background

The Benghazi Congressional Hearing is currently in the news. One complaint is that the hearings are expensive. How can we use data and Python to investigate that complaint? The costs of some previous congressional hearings are in a file named `hearings.txt`. However, the dollar amounts in that file need to be adjusted to account for inflation. To help with those adjustments, the inflation figures for the past 100 years are in a file named `inflation.txt`.

In this assignment, you will develop a program to calculate and display the inflation-adjusted cost of the congressional hearings.

Assignment Specifications

1. You will develop the Python program described below, paying particular attention to the required functions.

2. Your program may not use sets or dictionaries.

3. Your programs may use any built-in or standard library functions; it may not import any modules which are not in the Anaconda distribution.

4. You are provided with two input files: `inflation.txt` and `hearings.txt` . Your program must work with files with the same name that have a different number of lines and values, but that are formatted the same. Note that you are not to prompt for these file names; your program will just use those names.

5. One function reads a file, skips past a specified number of header lines, reads values in specified columns, and returns a list of lists where each nested list corresponds to the values read from one line of the file.

For example, consider the file `inflation.txt` and suppose that we need the year and the year's average inflation (as a percent) for all years, in that order. The year is in the first column (index 0) and the average inflation for that year is in the last column (index -1). In addition there is one header line to be skipped. The call to return a list of lists containing the data for each year will be as follows:

```
get_cols_from_file( "inflation.txt", [0,-1], 1 )
```

The general form of the function is:

```
get_cols_from_file( str, list, int ) -> list (of lists)
```

where

- the first parameter (of type `str`) is the file name to read from;
- the second parameter (of type `list`) is a list of integers defining the columns to be read;
- and the third parameter (of type `int`) is the number of header lines to ignore.

The function will return a list of lists, where the nested lists contain the data from the specified columns, in the order specified by the second parameter.

For example, given the file `inflation.txt` whose first lines are as follows:

```
YEAR   JAN    FEB    MAR    APR    MAY    JUN    JUL    AUG    SEP    OCT    NOV    DEC    AVE
1914   2.0    1.0    1.0    0.0    2.1    1.0    1.0    3.0    2.0    1.0    1.0    1.0    1.0
1915   1.0    1.0    0.0    2.0    2.0    2.0    1.0    -1.0   -1.0   1.0    1.0    2.0    1.0
1916   3.0    4.0    6.1    6.0    5.9    6.9    6.9    7.9    9.9    10.8   11.7   12.6   7.9
1917   12.5   15.4   14.3   18.9   19.6   20.4   18.5   19.3   19.8   19.5   17.4   18.1   17.4
```

The following function call

```
get_cols_from_file( "inflation.txt", [0,-1], 1 )
```

will return the following list

```
[ ['1914', '1.0'], ['1915', '1.0'], ['1916', '7.9'], ['1917', '17.4'], … ]
```

Note that outside of the function you will want to change the values in the lists to ints., e.g.

```
[ [1914, 1.0], [1915, 1.0], [1916, 7.9], [1917, 17.4], … ]
```

6. One function must take a dollar amount, year, and list of inflation percents, and adjust the amount for inflation from the specified year to the present (2015).

The general form of the function is:

```
adjust_for_inflation( float, int, list ) -> float
```

where

- the first parameter (a `float`) is the original dollar amount;
- the second parameter (an `int`) is the year corresponding to the dollar amount;
- and the third parameter (a `list`) is a list of lists of ints (as described above) of years and inflation values for the last 100 years in the form [[year, inflation], [year, inflation], …]

The function will apply inflation for each year up to and including 2014 (because we haven't reached the end of 2015 yet). Since the original dollar amount occurred some time in the middle of the initial year we will not count inflation for the first year. For example, the call

```
adjust_for_inflation( 14.4, 2002, inflation_list )
```

will apply inflation for the years 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, and 2014. It will return the float 18.9 (representing 18.9 million dollars, the inflation-adjusted value of 14.4 million dollars in 2002).

7. One challenge for calculating the inflation-adjusted amount for a given amount and year is to find the index in the `inflation_list` of the (nested) list containing the inflation information for that year. You must create and use the following function

```
find_index( int, list ) -> int
```

where

- the first parameter (an `int`) is a year;
- the second parameter (a `list`) is the inflation list (a list of lists of integers);
- and the return value is the index, in the input list, of the nested list that indicates the inflation value for that year.

For example,

```
find_index( 1917, inflation_list )
```

will return 3 which is the index of the inflation information for the year 1917 in the inflation list.

8. You will use the provided function `draw_bar_graph( list, list )` to display the inflation adjusted values as a bar graph where

- the first parameter is a list of strings
- the second parameter is a corresponding list of numbers

The function is provided in the file `draw_bar_graph.py` and is complete. Copy the function into your source code. You will call it with the list of names of congressional hearings and the list of their corresponding costs as arguments.

9. You will create a function to create a new file named `hearings_adjusted.txt` that has the same format as, but need not be copied from, the `hearings.txt file`. They can simply be string literals in your function. The function will have the form

```
write_file( list )
```

where the only parameter is a list of inflation-adjusted hearing data.

The function does not return anything.

10. Your program will

i. Display the inflation-adjusted values in a bar graph (using the provided plotting function).
ii. Create a file named `hearings_adjusted.txt` that is formatted like the `hearings.txt` input file, but with the original values replaced with inflation-adjusted values.

Assignment Notes

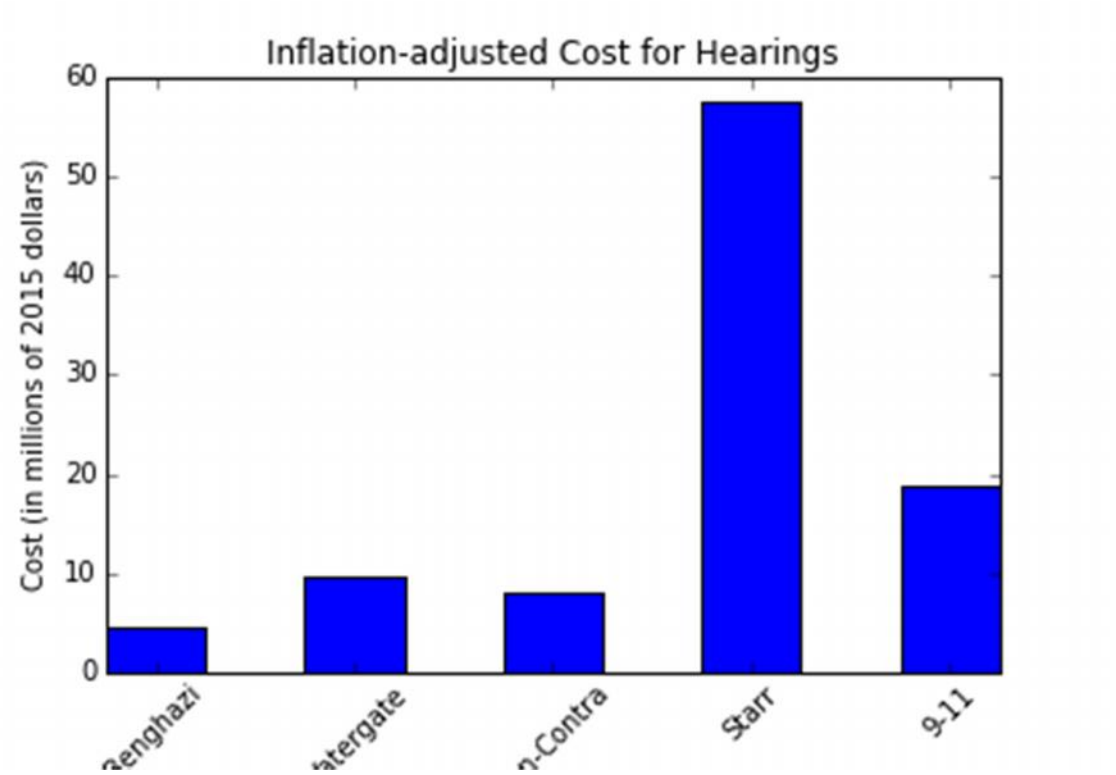
1. Items 1-8 of the Coding Standard will be enforced for this project.

2. Function `draw_bar_graph` in the file named `draw_bar_graph.py` is complete as-is. You should copy the source code for that function into your file `proj07.py` file. Remember to include the `import pylab` line.

3. So you can check your calculations the inflation-adjusted values rounded to one decimal place for the values in the file `hearings.txt` are:

```
Benghazi     4.5
Watergate    9.6
Iran-Contra  8.1
Starr       57.5
9-11        18.9
```

4. Here is what the bar graph will look like for those values:



Suggested Procedure

- *Solve the problem using pencil and paper first.* You cannot write a program until you have figured out how to solve the problem. This first step may be done collaboratively with another student. However, once the discussion turns to Python specifics and the subsequent writing of Python statements, you must work on your own.

- Use the **handin system** to turn in the first version of your solution.

- Cycle through the steps to incrementally develop your program:

    - Edit your library program to add new capabilities.
    - Run the program and fix any errors.
    - Use the **handin system** to submit the current version of your solution.

- Use the **handin system** to submit the final version of your solution.

- You would be wise to back up your files on your H: drive, also.

- Be sure to log out when you leave the room, if you're working in a public lab.