Update 11/21: included the dictionary used for testing top_10_ranks_across_years()

CSE 231 **Fall 2020**

Project #9

This assignment focuses on the design, implementation and testing of a Python program which uses control structures to solve the problem described below.
It is worth 55 points (5.5% of course grade) and must be completed no later than **11:59 PM on Monday, November 23, 2020**.

Assignment Overview
(learning objectives)
This assignment will give you more experience on the use of:

- lists
- dictionaries
- data structures
- functions
- iteration
- data analysis

The goal of this project is to analyze data relevant to the happiness index in countries across the world.

Assignment Background

Happiness is an abstract concept that can be very difficult to systematically define, study, and quantify. Even the most complex comparisons of happiness can be fundamentally flawed since there is a wide array of factors that could influence happiness – and these factors are different for different people. The society around us can create expectations blinding us from our otherwise happy lives. Happiness is also highly ephemeral, the happiest person, living in the best part of the world, can be happy one day and sad the next day. The pursuit of happiness is therefore always just that – a pursuit.

Project Description

This project focuses on analyzing a publicly available dataset containing quantifications of several factors that could influence happiness. Due to the reasons described above, we need to take this data with a grain of salt. However, for our purposes, we will structure this data into a dictionary and then extract, analyze, and display information as needed for an attempted comparison of happiness in different countries. The function descriptions follow.

`open_file(str) -> fp`

This function takes a string parameter that is the year string for the file name, opens the file, and returns a file pointer to the opened data file. File names have the format `year+.csv`, e.g. `2015.csv`. You likely have a similar copy from a previous project. It *should* have a try-except statement. If the specified file is not successfully opened, `return None`. (Note: this time the prompting loop for the year is in main.)

`build_dictionary(fp) -> dictionary`

This function accepts the previously generated file pointer as input and returns the required dictionary. You will find `csv.reader` useful to read the data file. This function iterates over the CSV reader and with each iteration, extracts the needed data and then creates a dictionary that holds all of the data. Remember to skip the header line. Also, the order of countries and states in the dictionary will be the same as the order observed in the CSV file. This dictionary is then returned by the function. The file is guaranteed to have only one entry for each country which allows you to write simpler code. However, the data file headings vary slightly from year to year, column 5 differs (but we don't use it), and some files have a final column named 'dystopia' (but we don't use it). If there is no data available as indicated by the string `'N/A'` ignore that line of input. The data to be extracted is as follows (note that the "column" information starts counting at 1 whereas the indices start at 0):

Region – column 2
Country - column 1
Happiness Rank - column 3
Happiness Score - column 4
Economy (GDP per Capita – column 6
Trust (Government Corruption) - column 10
Family – column 7
Health (Life Expectancy) – column 8
Freedom – column 9

The structure of the dictionary is as follows:

```
'Western Europe':
  {'Switzerland': ((1, 7.587),(1.39651, 0.41978),(1.34951, 0.94143, 0.66557)),
   'Iceland': ((2, 7.561), (1.30232, 0.14145), (1.40223, 0.94784, 0.62877)),
   'Denmark': ((3, 7.527), (1.32548, 0.48357), (1.36058, 0.87464, 0.64938)),
  …}
'Latin America and Caribbean':
  {'Costa Rica': ((12, 7.226),(0.95578, 0.10583),(1.23788, 0.86027, 0.63376)),
   'Mexico': ((14, 7.187), (1.02054, 0.21312), (0.91451, 0.81444, 0.48181)),
   'Brazil': ((16, 6.983), (0.98124, 0.17521), (1.23287, 0.69702, 0.49049)),
  …}
```

The dictionary has region as key and value which is a dictionary of countries. Each country dictionary has a tuple of tuples. These tuples contain the data needed for analysis.

The data within the dictionary represents the following factors:

```
{'Region':
  {'Country1': (('happiness rank', 'happiness score'), ('economy', 'trust'),
('family', 'health', 'freedom')),
   'Country2': (('happiness rank', 'happiness score'), ('economy', 'trust'),
('family', 'health', 'freedom')),
   …}
```

Important: close the file.

combine_dictionaries(int,dictionary,dictionary) -> dictionary

We provide this function. Using the `build_dictionary` function we created a *region* dictionary for a particular *year*. This function puts that dictionary as a value within a dictionary that has year as the key. We refer to the year-key dictionary as the super-dictionary. The first call to this function will have the last parameter, the super-dictionary, as an empty dictionary. This function accepts three parameters in this order: an integer value which is a year, a data dictionary (such as created by the `build_dictionary` function above), and the year-keyed dictionary. Our super-dictionary now has a format like this---it is the same as the one above but now within a dictionary that has the year as key. We could have done this all within the `build_dictionary` function, but did it as a separate function so the `build_dictionary` function would be easier for you to write.

```
{YEAR:
{‘Region’:
  {'Country1’: (('happiness rank', 'happiness score'), ('economy', 'trust'),
('family', 'health', 'freedom')),
   'Country2': (('happiness rank', 'happiness score'), ('economy', 'trust'),
('family', 'health', 'freedom')),
   …}
```

search_by_country(string, dictionary, boolean) -> list

This function takes in a string (the country name), a dictionary (the super year-key dictionary created by `combine_dictionary` above) and a Boolean variable that indicates whether to print (`True`) the country details on the screen or not (`False`). This Boolean variable indicates whether the function is called for the purpose of printing or for the purpose of returning a data list containing details relevant to the country. If the function is called for printing, then this function searches the dictionary for the country (for all years) and prints the details of year, country, rank and happiness index with proper formatting. The function does not return anything when called for printing. Example, for Ireland, for the year 2015:

```
Year:     2015
Country:  Ireland
Rank:     18
Score:    6.94
Family:   1.37
Health:   0.90
Freedom:  0.62
--------------------
```

Use the following format strings:

```
'{:<10s}{:<5d}'   # for ints
'{:<10s}{:<s}'    # for strings
'{:<10s}{:<5.2f}' # for floats
```

If the function is being called for returning the list containing details relevant to the country (mainly used by other functions), then it will not print anything but will instead return this list. Returned data structure is a list of tuples where each list has the last four values printed, e.g.
(Happiness_Score, Family, Health, Freedom)

For example, for Switzerland for 2015 and 2016:

```
[(7.587, 1.34951, 0.94143, 0.66557), (7.509, 1.14524, 0.86303, 0.58557)]
```

`top_10_ranks_across_years(dictionary, int, int) -> list, list`

This function accepts the super dictionary and produces 2 lists of tuples. List 1 contains the top 10 countries and their ranks for year 1. List 1 (for year 1) is sorted by the ranks for that year. For instance, for example, if we had the following dictionary:

```
{2015: {'Western Europe': {'Switzerland': ((1, 7.587), (1.39651, 0.41978),
               (1.34951, 0.94143, 0.66557)), 'Iceland': ((2, 7.561), (1.30232,
               0.14145), (1.40223, 0.94784, 0.62877)), 'Denmark': ((3, 7.527),
               (1.32548, 0.48357), (1.36058, 0.87464, 0.64938)), 'Norway': ((4,
               7.522), (1.459, 0.36503), (1.33095, 0.88521, 0.66973)), 'Finland':
               ((6, 7.406), (1.29025, 0.41372), (1.31826, 0.88911, 0.64169)),
               'Netherlands': ((7, 7.378), (1.32944, 0.31814), (1.28017, 0.89284,
               0.61576)), 'Sweden': ((8, 7.364), (1.33171, 0.43844), (1.28907,
               0.91087, 0.6598)), 'France': ((29, 6.575), (1.27778, 0.20646),
               (1.26038, 0.94579, 0.55011))},
         'North America': {'Canada': ((5, 7.427), (1.32629, 0.32957), (1.32261,
               0.90563, 0.63297)), 'United States': ((15, 7.119), (1.39451,
               0.1589), (1.24711, 0.86179, 0.54604))},
         'Latin America and Caribbean': {'Costa Rica': ((12, 7.226), (0.95578,
               0.10583), (1.23788, 0.86027, 0.63376)), 'Mexico': ((14, 7.187),
               (1.02054, 0.21312), (0.91451, 0.81444, 0.48181)), 'Brazil': ((16,
               6.983), (0.98124, 0.17521), (1.23287, 0.69702, 0.49049)),
               'Venezuela': ((23, 6.81), (1.04424, 0.11069), (1.25596, 0.72052,
               0.42908)), 'Panama': ((25, 6.786), (1.06353, 0.0927), (1.1985,
               0.79661, 0.5421)), 'Chile': ((27, 6.67), (1.10715, 0.12869),
               (1.12447, 0.85857, 0.44132)), 'Argentina': ((30, 6.574), (1.05351,
               0.08484), (1.24823, 0.78723, 0.44974)), 'Uruguay': ((32, 6.485),
               (1.06166, 0.24558), (1.2089, 0.8116, 0.60362)), 'Colombia': ((33,
               6.477), (0.91861, 0.0512), (1.24018, 0.69077, 0.53466))}},
 2016: {'Western Europe': {'Denmark': ((1, 7.526), (7.592, 0.57941), (1.44178,
               1.16374, 0.79504)), 'Switzerland': ((2, 7.509), (7.59,
               0.58557), (1.52733, 1.14524, 0.86303)), 'Iceland': ((3, 7.501),
               (7.669, 0.56624), (1.42666, 1.18326, 0.86733)), 'Norway': ((4,
               7.498), (7.575, 0.59609), (1.57744, 1.1269, 0.79579)),
               'Finland': ((5, 7.413), (7.475, 0.57104), (1.40598, 1.13464,
               0.81091)), 'Netherlands': ((7, 7.339), (7.394, 0.55211),
               (1.46468, 1.02912, 0.81231)), 'Sweden': ((10, 7.291), (7.355,
               0.58218), (1.45181, 1.08764, 0.83121)), 'France': ((32, 6.478),
               (6.559, 0.46562), (1.39488, 1.00508, 0.83795))},
 'North America': {'Canada': ((6, 7.404), (7.473, 0.5737), (1.44015,
       1.0961, 0.8276)), 'United States': ((13, 7.104), (7.188,
       0.48163), (1.50796, 1.04782, 0.779))},
 'Latin America and Caribbean': {'Costa Rica': ((14, 7.087), (7.175,
       0.55225), (1.06879, 1.02152, 0.76146)), 'Brazil': ((17, 6.952),
       (7.029, 0.40425), (1.08754, 1.03938, 0.61415)), 'Mexico': ((21,
       6.778), (6.876, 0.37709), (1.11508, 0.7146, 0.71143)), 'Chile':
       ((24, 6.705), (6.795, 0.37789), (1.2167, 0.90587, 0.81883)),
       'Panama': ((25, 6.701), (6.801, 0.48927), (1.18306, 0.98912,
       0.70835)), 'Argentina': ((26, 6.65), (6.74, 0.42284), (1.15137,
       1.06612, 0.69711)), 'Uruguay': ((29, 6.545), (6.634, 0.54388),
       (1.18157, 1.03143, 0.72183)), 'Colombia': ((31, 6.481), (6.578,
       0.44735), (1.03032, 1.02169, 0.59659)), 'Venezuela': ((44,
       6.084), (6.195, 0.19847), (1.13367, 1.03302, 0.61904))}}}
```

For 2015, the list produces (note the sorted ranks):

```
 [('Switzerland', 1), ('Iceland', 2), ('Denmark', 3), ('Norway',
4), ('Canada', 5), ('Finland', 6), ('Netherlands', 7),
('Sweden', 8), ('Costa Rica', 12), ('Mexico', 14)]
```

List 2 will be created such that it contains countries in the same order as List 1 but with appropriate ranks for year 2. For instance, List 2 will be returned as:

```
[('Switzerland', 2), ('Iceland', 3), ('Denmark', 1), ('Norway',
4), ('Canada', 6), ('Finland', 5), ('Netherlands', 7),
('Costa Rica', 14), ('New Zealand', 8), ('Mexico', 21)]
```

`print_ranks(dictionary, list, list, int, int)`

This function simply displays the data returned by the function above with the correct formatting. The two lists are the two lists returned by `top_10_ranks_across_years`; the `int` parameters are the years of the lists respectively. The following format string was used to display data for country, year1, year2, and average Happiness Score (average score for the two years displayed):

```
"{:<15s} {:>7s} {:>7s} {:>12s}"  # header
"{:<15s} {:>7d} {:>7d} {:>12.2f}"# data
```

The dictionary is the superdictionary which can be passed to the `search_by_country` function to get the happiness scores.

`prepare_plot(string, string, dictionary) -> tuple, tuple`

This function takes the names of the two countries and the super dictionary and returns 2 tuples. Each of the returned tuple contain 4 items pertaining to the respective country during the first year encountered in the super dictionary:

(Happiness Score, Family, Health, Freedom)

For example, when comparing Norway and Sweden, the tuples returned are:

```
(7.522, 1.33095, 0.88521, 0.66973), (7.364, 1.28907, 0.91087, 0.6598)
```

`bar_plot(string, string, list, list) -> plot`

This function is provided. You do not need to modify the source code for this function. However, you *do* need to invoke this function at the appropriate place. This function accepts 2 strings (country names) and 2 list. These two lists are the ones that were returned by the `prepare_plot()` function above.

`main()`

This function prints the provided BANNER and then asks the user for the year(s) to be analyzed (separated by comma; no space). There are data files for years 2015-2019. These files will be appropriately read according to the year(s) specified by the user. Use the open_file() function. Note: for simplicity assume that two valid years are correctly entered. The appropriate dictionary is then created using `build_dictionary()` and `combine_dictionaries().` Now you have a combined dictionary we refer to as the super-dictionary, `superD`. Next, the user is shown the MENU and asked for input.

If the choice is 1, the program asks the name of the country and then looks up this country's details in the super-dictionary by calling `search_by_country` which prints these details on the screen.

If the choice is 2, the program prints out a formatted comparison of the top 10 ranks across the years. Hint: use the `top_10_ranks_across_years` and `print_ranks` functions.

If the choice is 3, the program asks the user to specify the two countries separated by commas. If both of these countries are found in the super-dictionary then the comparison is shown to the user and the user is asked if plotting is needed. If user wants plotting, the appropriate bar plot is displayed using the function we provide. If any or both countries are missing from the dictionary (user specified incorrect name) then an error message is shown and the user is indefinitely re-prompted to input the two countries until both countries are correctly specified.

If the choice is x, the program exits.

Note: the program needs to catch exceptions in all the input sequences. For example, if the choice is not 1,2,3, or x, then an error message is displayed, and user is asked for input again. These errors should be handled without causing a program crash.

Assignment Deliverables

The deliverable for this assignment is the following file:

`proj09.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** before the project deadline.

Assignment Notes

1. Use `itemgetter()` from the `operator` module to specify the key for sorting.
2. Items 1-9 of the Coding Standard will be enforced for this project.

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 is best 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.

- Write a simple version of the program. Run the program and track down any errors.

- Use the debugger available in Spyder to locate and resolve errors. Set breakpoints right before the instructions where you perceive the program begins and then step through the code one instruction at a time. While doing this, keep an eye on the 'variable explorer' window in Spyder to observe the change in variables.

- Use the **Mimir** system to turn in the first version of your program.

- Cycle through the steps to incrementally develop your program:
    - Edit your program to add new capabilities.
    - Run the program and fix any errors.

- Use the **Mimir** system to submit your final version.

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

Test 1

```
                    __ooooooooo__
                 oOOOOOOOOOOOOOOOOOOOOOo
             oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
          oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
        oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
      oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
     oOOOOOOOOOOO*  *OOOOOOOOOOOOOO*  *OOOOOOOOOOOOo
    oOOOOOOOOOOO      OOOOOOOOOOOO      OOOOOOOOOOOOo
    oOOOOOOOOOOOOo  oOOOOOOOOOOOOOOo  oOOOOOOOOOOOOo
    oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
    oOOOO     OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO     OOOOo
    oOOOOOO OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO OOOOOOo
    *OOOOO  OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO  OOOOO*
    *OOOOOO  *OOOOOOOOOOOOOOOOOOOOOOOOOOOOO*  OOOOOO*
     *OOOOOO  *OOOOOOOOOOOOOOOOOOOOOOOOOOO*  OOOOOO*
      *OOOOOOo  *OOOOOOOOOOOOOOOOOOOOOOO*  oOOOOOO*
        *OOOOOOOo  *OOOOOOOOOOOOOOOOO*  oOOOOOOO*
          *OOOOOOOOo  *OOOOOOOOOOO*  oOOOOOOOO*
             *OOOOOOOOo           oOOOOOOOO*
                 *OOOOOOOOOOOOOOOOOOOOO*
                      ""ooooooooo""

Input Years comma-separated as A,B: 2015,2016
Opening Data file for year 2015:
Opening Data file for year 2016:

    1. Search by country
    2. Top 10 countries
    3. Compare countries
    x. Exit
    :1
[ ? ] Please specify the country: Ireland
Year:     2015
Country:  Ireland
Rank:     18
Score:    6.94
Family:   1.37
Health:   0.90
Freedom:  0.62
--------------------
Year:     2016
Country:  Ireland
Rank:     19
Score:    6.91
Family:   1.48
Health:   1.16
Freedom:  0.81
--------------------

    1. Search by country
    2. Top 10 countries
    3. Compare countries
    x. Exit
    :x
```

Test 2

```
                    __ooooooooo__
                 oOOOOOOOOOOOOOOOOOOOOOo
             oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
          oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
        oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
      oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
     oOOOOOOOOOOO*  *OOOOOOOOOOOOOO*  *OOOOOOOOOOOOo
    oOOOOOOOOOOO      OOOOOOOOOOOO      OOOOOOOOOOOOo
    oOOOOOOOOOOOOo  oOOOOOOOOOOOOOOo  oOOOOOOOOOOOOOo
    oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
    oOOOO     OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO     OOOOo
    oOOOOOO OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO OOOOOOo
    *OOOOO  OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO  OOOOO*
    *OOOOOO  *OOOOOOOOOOOOOOOOOOOOOOOOOOOOO*  OOOOOO*
     *OOOOOO  *OOOOOOOOOOOOOOOOOOOOOOOOOOO*  OOOOOO*
      *OOOOOOo  *OOOOOOOOOOOOOOOOOOOOOOO*  oOOOOOO*
        *OOOOOOOo  *OOOOOOOOOOOOOOOOO*  oOOOOOOO*
          *OOOOOOOOo  *OOOOOOOOOOO*  oOOOOOOOO*
             *OOOOOOOOo           oOOOOOOOO*
                 *OOOOOOOOOOOOOOOOOOOOO*
                      ""ooooooooo""

Input Years comma-separated as A,B: 2015,2016
Opening Data file for year 2015:
Opening Data file for year 2016:

    1. Search by country
    2. Top 10 countries
    3. Compare countries
    x. Exit
    :2

Country            2015    2016  Avg.H.Score
Switzerland           1       2         7.55
Iceland               2       3         7.53
Denmark               3       1         7.53
Norway                4       4         7.51
Canada                5       6         7.42
Finland               6       5         7.41
Netherlands           7       7         7.36
Sweden                8      10         7.33
New Zealand           9       8         7.31
Australia            10       9         7.30

    1. Search by country
    2. Top 10 countries
    3. Compare countries
    x. Exit
    :x
```

Test 3

```
                    __ooooooooo__
                 oOOOOOOOOOOOOOOOOOOOOOo
             oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
          oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
        oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
      oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
     oOOOOOOOOOOO*  *OOOOOOOOOOOOOO*  *OOOOOOOOOOOOo
    oOOOOOOOOOOO      OOOOOOOOOOOO      OOOOOOOOOOOOo
    oOOOOOOOOOOOOo  oOOOOOOOOOOOOOOo  oOOOOOOOOOOOOOo
    oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
    oOOOO     OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO     OOOOo
    oOOOOOO OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO OOOOOOo
    *OOOOO  OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO  OOOOO*
    *OOOOOO  *OOOOOOOOOOOOOOOOOOOOOOOOOOOOO*  OOOOOO*
     *OOOOOO  *OOOOOOOOOOOOOOOOOOOOOOOOOOO*  OOOOOO*
      *OOOOOOo  *OOOOOOOOOOOOOOOOOOOOOOO*  oOOOOOO*
        *OOOOOOOo  *OOOOOOOOOOOOOOOOO*  oOOOOOOO*
          *OOOOOOOOo  *OOOOOOOOOOO*  oOOOOOOOO*
             *OOOOOOOOo           oOOOOOOOO*
                 *OOOOOOOOOOOOOOOOOOOOO*
                      ""ooooooooo""

Input Years comma-separated as A,B: 2015,2016
Opening Data file for year 2015:
Opening Data file for year 2016:

    1. Search by country
    2. Top 10 countries
    3. Compare countries
    x. Exit
    :3
[ ? ] Please specify the two countries (A,B): Costa Rica,Switzerland

Country             Hap.Score Family   Life Ex. Freedom
Costa Rica           7.23      1.24     0.86     0.63
Switzerland          7.59      1.35     0.94     0.67
[ ? ] Plot (y/n)? n

    1. Search by country
    2. Top 10 countries
    3. Compare countries
    x. Exit
    :x
```

Test 4

```
                    __ooooooooo__
                 oOOOOOOOOOOOOOOOOOOOOOo
             oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
          oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
        oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
      oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
     oOOOOOOOOOOO*  *OOOOOOOOOOOOOO*  *OOOOOOOOOOOOo
    oOOOOOOOOOOO      OOOOOOOOOOOO      OOOOOOOOOOOOo
    oOOOOOOOOOOOOo  oOOOOOOOOOOOOOOo  oOOOOOOOOOOOOOo
    oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
    oOOOO     OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO     OOOOo
    oOOOOOO OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO OOOOOOo
    *OOOOO  OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO  OOOOO*
    *OOOOOO  *OOOOOOOOOOOOOOOOOOOOOOOOOOOOO*  OOOOOO*
     *OOOOOO  *OOOOOOOOOOOOOOOOOOOOOOOOOOO*  OOOOOO*
      *OOOOOOo  *OOOOOOOOOOOOOOOOOOOOOOO*  oOOOOOO*
        *OOOOOOOo  *OOOOOOOOOOOOOOOOO*  oOOOOOOO*
          *OOOOOOOOo  *OOOOOOOOOOO*  oOOOOOOOO*
             *OOOOOOOOo           oOOOOOOOO*
                 *OOOOOOOOOOOOOOOOOOOOO*
                      ""ooooooooo""

Input Years comma-separated as A,B: 2018,2019
Opening Data file for year 2018:
Opening Data file for year 2019:

    1. Search by country
    2. Top 10 countries
    3. Compare countries
    x. Exit
    :3
[ ? ] Please specify the two countries (A,B): Switzerland,United States

Country             Hap.Score Family   Life Ex. Freedom
Switzerland          7.49      1.55     0.93     0.66
United States        6.89      1.47     0.82     0.55
[ ? ] Plot (y/n)? n

    1. Search by country
    2. Top 10 countries
    3. Compare countries
    x. Exit
    :x
```

Test 5 (plotting; no Mimir test)

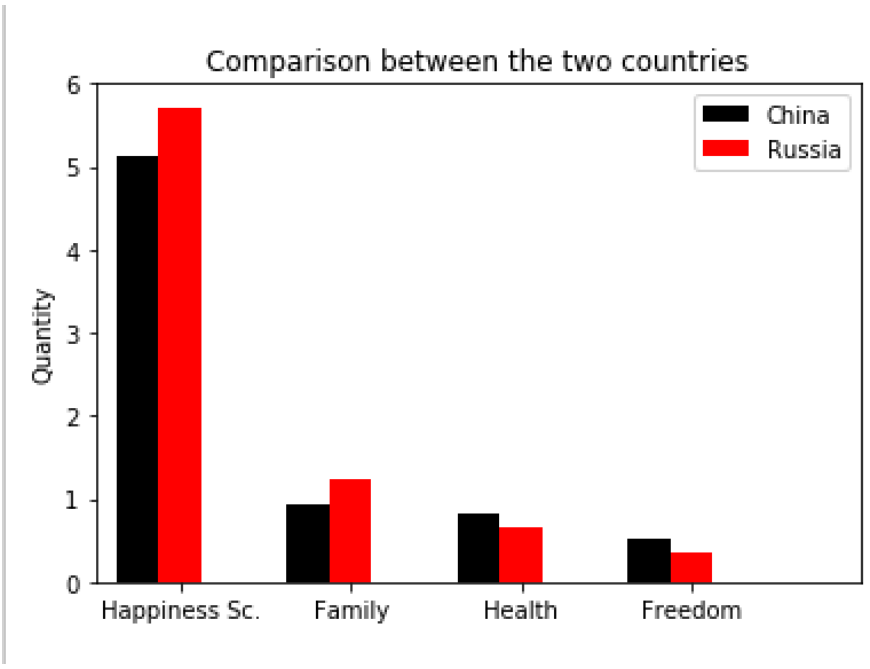
```
                    __ooooooooo__
                 oOOOOOOOOOOOOOOOOOOOOOo
             oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
          oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
        oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
      oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
     oOOOOOOOOOOO*  *OOOOOOOOOOOOOO*  *OOOOOOOOOOOOo
    oOOOOOOOOOOO      OOOOOOOOOOOO      OOOOOOOOOOOOo
    oOOOOOOOOOOOOo  oOOOOOOOOOOOOOOo  oOOOOOOOOOOOOOo
    oOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOo
    oOOOO     OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO     OOOOo
    oOOOOOO OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO OOOOOOo
    *OOOOO  OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO  OOOOO*
    *OOOOOO  *OOOOOOOOOOOOOOOOOOOOOOOOOOOOO*  OOOOOO*
     *OOOOOO  *OOOOOOOOOOOOOOOOOOOOOOOOOOO*  OOOOOO*
      *OOOOOOo  *OOOOOOOOOOOOOOOOOOOOOOO*  oOOOOOO*
        *OOOOOOOo  *OOOOOOOOOOOOOOOOO*  oOOOOOOO*
```

```
          *000000000o  *00000000000*  o00000000*
             *000000000o           o00000000*
                 *0000000000000000000000*
                      ""ooooooooo""

Input Years comma-separated as A,B: 2015,2016
Opening Data file for year 2015:
Opening Data file for year 2016:

    1. Search by country
    2. Top 10 countries
    3. Compare countries
    x. Exit
    :3
[ ? ] Please specify the two countries (A,B): China,Russia

Country             Hap.Score Family   Life Ex. Freedom
China                5.14      0.95     0.82     0.52
Russia               5.72      1.24     0.67     0.37
[ ? ] Plot (y/n)? y
```



```
    1. Search by country
    2. Top 10 countries
    3. Compare countries
    x. Exit
    :x
```

Grading Rubric

```
Computer Project #09                                Scoring Summary
General Requirements:
```

4 pts
Coding Standard 1-9
(descriptive comments, function headers, etc...)

Function Tests:
3 pts open_file (no Mimir test)
6 pts build_dictionary()
6 pts top_10_ranks_across_years()
5 pts search_by_country()
3 pts prepare_plot()
4 pts print_ranks() (no Mimir test – tested in tests TestX)

Program Tests
5 pts Test1
5 pts Test2
5 pts Test3
5 pts Test4
4 pts Test5 (plotting; manual, no Mimir test)