Assignment 10

Due at 4:00pm Monday, April 8

- There are no restrictions on solutions techniques for this assignment. You may use recursion, iteration, abstract list functions, string or list methods, built-in sorting techniques, or any other approaches you wish, as long as no additional modules are required beyond the **`check`** and **`math`** modules.
- **You should use a `try-except` block in your solution to question 1, but not in any other question.** When testing your solutions for questions 2 and 3, we will only use names of existing files.
- In all questions, the files used for testing will always follow the specified format. Sample data files for all questions consuming files are posted on the assignments page.
- To test your functions that read data files, you will need to create some of your own files (following the format described for each question). You will not submit these files when you submit your solutions. However, you should include with your test cases a comment giving a short description of the contents of your test file, and the purpose of each test (e.g. "file contains 1 temperature for the day", or "file contains multiple temperatures for the day", etc.). See the Style Guide for examples.
- **Be sure to close all files opened in your programs.**
- Download the testing module from the course Web page. Include **`import check`** in each file.
- Be sure to use the strings that exactly match those specified on the assignment and interface. Changing them in any way may result in the loss of correctness marks. **This applies specifically to constants used in Questions 2 and 4.**
- You are encouraged to use helper functions in your solutions as needed. Include them in the same file as your solution, but make helper functions separate functions from the main function, i.e. do NOT make them local functions. You do not need to provide examples and tests for these helper functions.
- Do not copy the purpose directly from the assignment description. The purpose should be written in your own words and include reference to the parameter names of your functions.
- Do not cut-and-past examples (or other text) from the assignment description, as this can result in your code failing all tests.
- The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.
- Assignments will not be accepted through email. Course staff will not debug code emailed to them.
- Test data for all questions will always meet the stated assumptions for consumed values.
- Read the course Web page for more information on assignment policies and how to organize and submit your work. Follow the instructions in the style guide. Specifically, your solutions should be placed in files **`a10qY.py`**, where Y is a value from 1 to 4.
- Download the interface file from the course Web page.
- For full marks, it is not sufficient to have a correct program. Be sure to follow all the steps of the design recipe given in the Python Style Guide for CS116, including the definition of constants and helper functions where appropriate.

Coverage: Module 10
Language: Python

1. The UW Weather Station maintains a great archive of weather information which can be downloaded from their website. For example, the ambient air temperature is recorded many times over the course of a day (usually, it is recorded every 15 minutes, but that can vary on a given day).

   Write a Python function **`get_low_high`**, that consumes a string (**`day_temps`**, the name of a file containing the temperature readings for a single day) and produces a list of two floats: **`[low,`**

Assignment 10

Due at 4:00pm Monday, April 8

high], where **low** is the lowest temperature recorded in the file and **high** is the highest. If there is no input file with the specified name, the function produces the empty list.

You may assume that the file, when it exists, contains at least two lines of data:

- A header line, which may be ignored –

```
Year    Day   Month      Time       Ambient_Air_Temperature(C)
```

- One line for each temperature reading, containing the year, day, month, time, and, finally the temperature in degrees Celsius (the data will always be in this order, and pieces of data are separated by whitespace):

```
2003    17    January    430   -13.23
```

- Note that the time is given using the 24 hour clock format (but you are not required to use that time information in any way).

For example, for the sample data file **"2003-Jan-17-temps.txt"**, the function should produce the list **[-19.32, -8.68]**. Because the produced list contains floating point values, your values may appear slightly different. You should use **check.within** to test the individual values in the produced list (using a tolerance of 10^{-5} is sufficient here).
Original source of data: ***http://weather.uwaterloo.ca/data.html***

As mentioned in the instructions at the beginning of the assignment, do not use a* `try-except` *block in questions 2, 3, and 4. You may assume the required data files exist in Questions 2 and 3.

2. BBM Canada© calculates weekly ratings for television programs in Canada, making a listing of the top shows for a week available on their website.

   Write a function **ratings_by_day** that consumes two nonempty strings: **ratings_file** (the name of an existing file containing the National ratings data for a week) and **day_of_week** (one of the strings **"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"**). The function must read in all the data in the file and will create a new file, containing the rank and program title for all shows in the file which air on the given day. The name of the new file will be the concatenation of **day_of_week**, the string **'-'**, and **ratings_file**. The function itself will produce **None**.

   The file **ratings_file** will have the following format (a sample file, **"ratings-feb25.txt"**, is provided):
   - Two information lines, which should be read in and ignored;
   - One line for each of the top rated shows, containing the ranking (starting in order from 1), the program title, the broadcast outlet, weekdays information, start time, end time, and a rating measure (separated from each other by a single comma, and no extra spaces). For example:

```
1,NCIS,Global Total,.T.....,20:00,21:00,2847
```

   - The weekdays information is a string of length 7, where the first character in the string corresponds to Monday (day 0), the second to Tuesday (day 1), etc. A show may air on multiple days. If a show does not air on a particular day, then **'.'** appears at its spot in the weekdays string. If it does air

Assignment 10
Due at 4:00pm Monday, April 8

on a day, the first letter in the day name appears at that spot. For example, `'…T…'` means the show airs only on Thursday, and `'MT.TF.."` means the show airs on Monday, Tuesday, Thursday and Friday.

- While the data provided by BBM Canada on its site lists 30 programs, do not assume that there are exactly 30 programs in the file. However, there will always be at least one program listed in the input file after the two header information lines.

The file created by the program will contain at least one line, a title line indicating the day of the week. After the first line, there will be one line for each program airing on that day. For example, for the sample data file provided, calling **`ratings_by_day("ratings-feb25.txt", "Saturday")`**, creates a new file called **`"Saturday-ratings-feb25.txt"`**, which contains the following information:

```
Top Shows airing on Saturday
10 HNIC PRIME EAST
21 CTV EVENING NEWS WKD
24 CTV NATIONAL NEWS
25 HNIC PRIME WEST
```

For each program on the specified day, include: Program rank, a single space, program title

Note that each line in the new file ends with a **`"\n"`** character. If there are no programs for the consumed day, then the file will contain only the first line (which will end with a **`"\n"`** character).

Original source of data: ***http://www.bbm.ca/en/weekly-top-30-tv-programs***

Questions 3 and 4 both use the following new type:

```
class concert:
''' A concert is an object concert(year, month, day, location), where
    * year is a positive integer (the year the concert was held),
    * month is an integer between 1 and 12 (the month it was held),
    * day is an integer between 1 and 31 (the day it was held), and
    * location is a nonempty string (where the concert was held)'''
```

The interface file provides several important functions for this class (including the constructor). Be sure to include the entire class in your solutions to Questions 3 and 4.

Devoted fans for musical artists like to keep track of the setlists (the list of songs performed) for concerts performed by their favourite artists. Several websites are available with this information, as listed below. In Question 3, you will create a dictionary of song information performed at all the concerts listed in a consumed file. In Question 4, you will use the information from such a dictionary to create a new file, with the information organized in a different manner.

Original source of data: ***http://brucebase.wikispaces.com/***
Related source of data: ***http://www.setlist.fm/***

Assignment 10

Due at 4:00pm Monday, April 8

3. Suppose we have setlists from several concerts in a data file, in the following format:
   - The data for each concert covers two consecutive lines:
     - The first line contains the date and location of the concert, for example:
       2009-05-07 - AIR CANADA CENTRE, TORONTO, ON, CANADA
       where the date is always in the **yyyy-mm-dd** format. It is followed by a single space, a dash (**'-'**), another single space, and then the location of the concert (which is usually the name of a building and a city, but should all be grouped together as the concert location).
     - The second line for the concert contains a list of all the songs performed at the concert, separated by a space, a slash ('/') and another space. For example,
       BADLANDS / NO SURRENDER / OUTLAW PETE / SHE'S THE ONE
   - All the songs will appear on one line in the file (even if it is a very long line!)
   - Some of the song titles have additional information stored in brackets after the title (for example, another part of the title, or additional performers). This additional information should not be included with the song title in our processing. For example, when processing
     57 CHANNELS (AND NOTHIN' ON) / GHOST OF TOM JOAD (with Tom Morello)
     the songs titles should be recorded as **"57 CHANNELS"** and **"GHOST OF TOM JOAD".**
   - There will be at least one blank line between the data for each concert. There may or may not be blank lines at the start and end of the file. Your program will need to be able to handle the formatting regardless of the number of blank lines throughout the file.
   - There will not be any blank lines between the concert date line and the setlist line.
   - You may assume that there is information about at least one concert in the file, and that at least one song was performed at each concert.
   - A sample data file is posted on the Assignments page (**"sample-concerts.txt"**).

   Complete the function **get_songs**, which consumes a nonempty string (**setlists**, the name of a file containing setlist information, in the format described above), and produces a dictionary which contains information about the songs and when and where they were performed. In the new dictionary:
   - The keys are the names of all the songs performed.
   - The values associated with the song titles are the lists of concerts at which they were performed.
   - The order of the concerts corresponds to the order in which they occur in **setlists**.

   For example,
   - **get_songs("sample-concerts.txt")** produces a dictionary with 86 keys.
   - In the dictionary, the key **"WE ARE ALIVE"** has the associated value
     [concert(2012, 4, 13, "FIRST NIAGARA CENTER, BUFFALO, NY"), concert(2012, 8, 24, "ROGERS CENTRE, TORONTO, ON")].

4. Complete a function **organize_songs**, that consumes a dictionary (**songs**, in the same format as the dictionary produced in Question 3), and a nonempty string (**output_file**, the name of a file that this function must create). The function produces None, and will write all the song titles to the new file, along with information about the concerts at which the song was performed. The titles will be in

Assignment 10

Due at 4:00pm Monday, April 8

decreasing order by the number of times they were played (use standard alphabetical ordering of titles in the case of ties). The concert information will be in the same order as it appears in the dictionary list.

For example, suppose

```
all_songs = {"TERRY'S SONG" : [concert(2012, 10, 21, "HAMILTON")],
             "BORN TO RUN": [concert(2009, 5, 7, "TORONTO"),
                             concert(2012, 10, 21, "HAMILTON")],
             "ROSALITA" : [concert(2012, 10, 21, "HAMILTON")]}
```

then calling

```
organize_songs(all_songs, "summary.txt")
```

writes the following information to the file **"summary.txt"**:

```
BORN TO RUN performed 2 time(s) at:
     7/ 5/2009: TORONTO
    21/10/2012: HAMILTON
ROSALITA performed 1 time(s) at:
    21/10/2012: HAMILTON
TERRY'S SONG performed 1 time(s) at:
    21/10/2012: HAMILTON
```

"BORN TO RUN" occurs first in the file because it was played the greatest number of times.

"ROSALITA" occurs before "TERRY'S SONG" because it comes first alphabetically and the songs were performed the same number of times.

Notes:

- There are no blank lines between song title information.
- There is a **"\n"** after each title and each concert.
- For each song title, the song title and the number of times played is written to the file in the format shown above.
- The information about each concert is preceded by 4 spaces **" "**. (Note that the first occurrence of **"BORN TO RUN"** appears to be indented 5 spaces, but the last of those comes from the format produced by the **__repr__** method in the **concert** class).
- If the dictionary is empty, the new file should still be created, but will be empty.
- *While the dictionary consumed by this question has the same format as the dictionary produced by Question 3, the questions are independent. You can complete this question and receive full marks even if you do not complete Question 3.*