ISTA 130: Spring 2020
Programming Assignment 9
(100 points)
Dictionaries with Lists as Values

Due : April 30th at 11:59 pm

(submit via D2L dropbox)

Please read the instructions below carefully and follow them closely. All problems (and sub-parts of the problems) are required except as noted in the instructions below. If you have any questions, please email the instructor or one of the section leaders.

Important: **You will lose 5 points if you use the tab character for indentation.**

Important: Your filenames must be identical to the filenames given below. For any functions you are asked to write, the function signature (header) must be exactly as described in the instructions. That is, you must use the exact function names given in the instructions, you must have the parameters the instructions ask for, and the parameters must be in the order the instructions give.

Important: Make sure you always save a backup of your work somewhere safe (such as your UA Box, Dropbox, Google Drive, etc).

Important: You are not just graded on whether your code produces the same result as the examples show. You should be using what you've learned to do your best to write good code. For example, things like poorly named variables/functions, duplicating code where you could instead use a loop, and missing documentation will cost you points.

1.0) FISH CATCH (40 POINTS)

1.1) DETAILS

Download the plain text files `"fishcatch.txt"` and `"fishcatch.dat"` from D2L and open them both with Sublime, Notepad++, or any plain text editor.

`"fishcatch.txt"` gives an explanation of the data in `"fishcatch.dat"`. Read the explanation to understand what is in `"fishcatch.dat"`. For this assignment

we'll only be interested in the name and weight of each fish in the `".dat"` file.

Create a new file called `"fishcatch.py"`. In the file:

1.) Write a function called **`fish_dict_from_file`** that takes a single string parameter giving the name of a file to read.

i.) In the function make a dictionary containing the mapping from the Numeric Species Code to the English fish name.

- For example, in your literal dictionary the key `1` will map to the value `"Bream"`. If your literal dictionary is called `fishmap` then the following expression would be `True: fishmap[1] == "Bream"`.
- You'll use this dictionary in the next step to get the name of each fish.

ii.) Next create an empty dictionary and read each fish Species and Weight from the `"fishcatch.dat"` file into the dictionary, mapping the English name of each fish species onto a list containing the weights of all of the fish of that type that were caught.

- i.e. for each different fish name the dictionary will contain a single list of floats.
  - e.g. the dictionary will have one ***key-value*** pair for `"Bream"`. The key will be `"Bream"` and the value will be a list of floats containing the weights of all of the Bream that were caught.
- Skip any fish that has a missing weight value (i.e. weight is "NA").

iii.) Return the dictionary that contains the fish names and weights.

2.) In **`main`**:

i.) Call your function to get a dictionary of fish names and weights.

ii.) Print a report showing for each fish the number of fish of that type, the name of the fish, and the mean weight of that fish type in grams.

- The report should show the fish in alphabetical order.

- Make it pretty as shown in the following example:

```
  # NAME            MEAN WT
 11 ?                154.8g
 34 Bream            626.0g
 56 Perch            382.2g
 17 Pike             718.7g
 20 Roach            152.1g
 14 Smelt             11.2g
  6 Whitefish        531.0g
```

NOTE: You will need to right justify the # by 4, left justify the **NAME** by 10 and finally right justify the **MEAN WT** by 10, if you pass 3 arguments to print. If you use string concatenation, you will have to add to spaces in the appropriate positions.

3.) Verify that your documentation makes sense and that you've added documentation to each of your functions.

4.) **Verify that your program works.**

5.) Upload your file to the Hw8 Assignments folder on D2L

2.0) EMOTICONS (60 POINTS)

2.1) DETAILS

Download the plain text file `"twitter_emoticons.dat"`.

The file contains actual Twitter data about tweets that contained emoticons. Each row in the file represents a single tweet. Each row contains the emoticon text characters, a tweet ID, a user ID, and a timestamp.

Here are the first few lines of the file:

```
"D:" 5646373703 "0049961833" 20091112110207
";-)" 5646377600 "0003829631" 20091112110222
";-)" 5646425002 "0035714667" 20091112110544
":(" 5646427235 "0031244602" 20091112110554
…
```

The first record is from a tweet containing D: with tweet ID `5646373703`, tweeted by user `0049961833` with a timestamp of `20091112110207`. The second record is from a tweet containing `;-)` with tweet ID `5646377600`, tweeted by user `0003829631` with a timestamp of `20091112110222`.

For this program we'll only be interested in the emoticon and the user ID. Create a new file called `"emoticons.py"`. In the file:

1.) Write a function called **`load_twitter_dicts_from_file`** that takes three parameters: `filename` (the name of a twitter data file to read), `emoticons_to_ids` (an empty dictionary), and `ids_to_emoticons` (an empty dictionary).

i.) The function should read the given file and load the two dictionaries with key-value pairs. **Quotes will hang the test program. You must get rid of them.**

- `emoticons_to_ids` will contain the emoticons as keys. Each emoticon will have as its value a list containing the user IDs (**strings – casting to `int` will hang the test**) from all tweets involving that emoticon.
  - there will be multiple entries in the list for users who tweeted the same emoticon on multiple occasions
- `ids_to_emoticons` will contain the user ID's as keys. Each user ID will have as its value a list containing the emoticons from all tweets authored by that user.
  - there will be multiple entries in the list for emoticons that were used on multiple occasions
- NOTE: in the ".dat" file the emoticons and the user IDs are wrapped in

double quotes (e.g. ";-)". You should remove the wrapping double quotes from around emoticons and ID's before using them as keys and values in your dictionaries.

ii.) The function does NOT return anything.

2.) Write a function called **`find_most_common`** that takes a single dictionary parameter. Assume the keys of the dictionary will be strings (e.g. emoticons) and each value will be a list of strings (e.g. user ID's).

i.) The function should find the key that has the longest list as a value and print out that key and the length of its list. Print in the following format:

```
:)   occurs   871 times
```

NOTE: In order to get this format you must left justify the emoticon by 21 and then right justify the number by 9 if using string concatenation, one less in each case if passing multiple arguments to `print`. `'occurs'` doesn't need to be justified.

ii.) The function returns the key that had the longest list.

3.) In **`main`**:

i.) Create two empty dictionaries (`emoticons_to_ids` and `ids_to_emoticons`)

ii.) Call your **`load_twitter_dicts_from_file`** function passing it the file name and the two dictionaries.

iii.) Print the number of different emoticons found in the data file:

```
Emoticons: 98
```

(one space after colon)

iv.) Print the number of different user ID's found in the data file:

```
UserIDs:   2878
```

(three spaces after colon)

v.) Next call your **`find_most_common`**, passing it the `emoticons_to_ids` dictionary. The result should look like this:

```
:)  occurs  871 times
```

- Now pop the most common emoticon from the dictionary and again call your **`find_most_common`** function on the `emoticons_to_ids` dictionary. Since the :) is gone a different emoticon will be the most common.
- Repeat this process five times. Your output from all five function calls should look like this:

```
:)  occurs  871 times
:D  occurs  377 times
:(  occurs  231 times
;)  occurs  219 times
:P  occurs  134 times
```

4.) Verify that your documentation makes sense and that you've added documentation to each of your functions.

5.) **Verify that your program works.**

6.) Upload your file to the Hw9 Assignments folder on D2L