Object-Oriented Programming
50:198:113 (Spring 2016)

Homework:	**2**	**Professor:**	**Suneeta Ramaswami**
Due Date:	**2/26/16**	**E-mail:**	`rsuneeta@camden.rutgers.edu`
Office:	**321 BSB**	**URL:**	`http://crab.rutgers.edu/~rsuneeta`
		Phone:	**(856)-225-6439**

Homework Assignment 2

This assignment will review material covered in the introductory course on Python. It requires you to write several functions that use strings, string formatting, and dictionaries. You will also read from and write into files. In this and all assignments, you are graded not only on the correctness of the code, but also on clarity and readability. I will deduct points for poor indentation, poor choice of object names, and lack of documentation. For documentation, use a common sense approach. While I do not expect every line of code to be explained, all code blocks that carry out a significant task should be documented *briefly* in clear English. As emphasized in class, use triple-quote documentation to document modules as well as functions.

The assignment is due by 11:59PM of the due date (given above). The point value is indicated in square braces next to each problem. Each solution must be the student's own work. Assistance should be sought or accepted only from the course instructor or the TA. Any violation of this rule will be dealt with harshly.

Important note: When writing each of the following programs, it is important that you *name the functions exactly as described* because I will assume you are doing so when testing your programs. Points will be deducted if your program produces errors because the functions do not satisfy the stated prototype.
Please read the submission guidelines at the end of this document before you start your work.

Problem 1 [25 points] Calendars. In this problem, you are asked to write a program that will print out a calendar for a particular month of any given year in or after 1754 A.D (roughly when the Gregorian calendar, the most commonly used calendar today, came into place). This might seem like a tall order, but if you can determine the day of the week for the first of the month in any given year, then the task becomes much simpler.

We begin with certain relevant facts:

- Years that are not leap years have 365 days, with 28 in February.
- Leap years have 366 days, with 29 in February.
- A year is a leap year if it is divisible by 4 but not by 100, or if it is divisible by 400. For example, 1976, 2000, 2400, and 2552 are all leap years, whereas 1900, 2200, and 2999 are not.
- The months have 31 days, except for April, June, September and November, which have 30, and February, which has 28 or 29 as explained above.
- **January 1, 1754 was a Tuesday.**

In order to compute the day of the week for the first of any month of any given year, you need to find the number of days that have elapsed since January 1, 1754.

Write a program to *repeatedly* do the following: Prompt the user to input a year on or after 1754. It should then prompt the user for a month of the year (entered as an integer in the range 1 through 12). Make sure to check that input lies in the valid range. Your program must first read in the year and then the month, in that order. It should then print out the calendar in a neatly formatted manner with appropriate headings for months and days of the week.

Implement the following functions in your program. Your program should contain at least these functions. You may include others if you wish, but the following will be sufficient to implement a modular program.

1. (*3 points*) A function called `is_leap` with a single integer parameter `yr`. The function returns `True` if `yr` is a leap year, and `False` otherwise.
2. (*3 points*) A function called `monthdays` with two integer parameters, `yr` and `mon`. The function returns the number of days in the month `mon` in the year `yr`. You may assume that `mon` is a valid value (i.e., an integer between 1 and 12, inclusive). Note that the only month for which this value can change from one year to another is February. The function `is_leap` will come in handy here.
3. (*2 points*) A function called `yeardays` with a single integer parameter `yr`. The function returns the number of days in the year `yr`.
4. (*5 points*) A function called `wkday_on_first` with two integer parameters, `yr` and `mon`. The function returns (as a string) the day of week on the first of the month `mon` in the year `yr`. Therefore, this function will return one of seven possible strings: `"sunday"`, `"monday"`, `"tuesday"`, `"wednesday"`, `"thursday"`, `"friday"`, or `"saturday"`.

   The functions `yeardays` and `monthdays` will come in handy here. Also recall that 1/1/1754 was a Tuesday (see discussion above).
5. (*12 points*) A function called `print_calendar` with two integer parameters, `yr` and `mon`. This function should first check for the validity of the parameters; that is, `yr` should be greater than or equal to 1754, and `mon` must be an integer between 1 and 12 (inclusive). If the input is invalid, the function prints an error message and returns. If the input is valid, the function should print the calendar for month `mon` in year `yr` in a neatly formatted manner (see sample runs).

   Hints: Use the `wkday_on_first` function to figure out the weekday at which to start printing the dates. Use the `monthdays` function to figure out how many dates to print. Use string formatting to print in a neatly formatted manner.

Note: In a linux/unix shell, if you type `cal <monthnum> <year>`, it will give you the calendar for `<monthnum>` in the year `<year>`. For example, `cal 7 1776` gives the calendar for July 1776. You can use this to check if your program is running correctly.

You are expected to make the best use of all functions in your program. In other words, do not repeat code unnecessarily to carry out a task if there is already a function to do it. You will be penalized for repeating code unnecessarily. Some sample runs are shown below.

```
>>> print_calendar(1959, 12)
```

```
                 1959
               December

          Su Mo Tu We Th Fr Sa
                 1  2  3  4  5
           6  7  8  9 10 11 12
          13 14 15 16 17 18 19
          20 21 22 23 24 25 26
          27 28 29 30 31

>>> print_calendar(1756, 2)

                 1756
               February

          Su Mo Tu We Th Fr Sa
           1  2  3  4  5  6  7
           8  9 10 11 12 13 14
          15 16 17 18 19 20 21
          22 23 24 25 26 27 28
          29

>>> print_calendar(1776, 7)

                 1776
                 July

          Su Mo Tu We Th Fr Sa
              1  2  3  4  5  6
           7  8  9 10 11 12 13
          14 15 16 17 18 19 20
          21 22 23 24 25 26 27
          28 29 30 31
```

Problem 2 [25 points] Frequency distribution of words in a file. Several text processing applications require an analysis of the word frequency distribution in the text. The term *word frequency* refers to the number of occurrences of each word in the text.

In this problem, you are asked to find the frequency distribution of words in a given text file, i.e., the number of times each word occurs in the file. For our purposes, a word is simply a consecutive sequence of characters that are letters of the alphabet. We do not distinguish between upper and lower case (e.g., “The” and “the” are the same word). For example, the text “How old are you and how old is your half-brother?” consists of nine words: “how”, “old”, “are”, “you”, “and”, “is”, “your”, “half”, and “brother”. The words “how” and “old” have a frequency of 2 each, and the remaining words have a frequency of 1 each.

Note that we do not include punctuation characters in a word. This is why, in the example above, the word “half-brother” is treated as two words, “half” and “brother”. This also means that words such as “Sally’s” or “They’re” get treated as two words each (“Sally” and ’s’, “They” and “re”), but we’ll live with that. There are various ways in which you can split a word at its punctuation marks. For example, you can use the `isalpha()` method or the `replace()` method for strings. In the `string` module, you can get the string of all punctuation characters by using `string.punctuation`. You may use this if necessary.

You are asked to use a dictionary to find the frequency distribution of words in a text file. In particular, you are asked to implement the following functions:

1. (*7 points*) A function called `freq_distribution` with two parameters: `infile` and `distfile`. The first parameter, `infile`, is the name of the file for which we want to compute the frequency distribution. The second parameter, `distfile` is the name of the file into which the frequency distribution is to be written in *sorted order of the words.* That is, the words should be listed in alphabetically sorted order in the output file. Each line of the file should contain the word and its frequency. Use string formatting to make sure the output is *neatly formatted.*

In order to carry out the above task, you must first create a dictionary to store the frequency distribution. You will do this by *calling the helper function* `freq_dictionary` described in #(3) below. Remember to close all files used in the function.

2. (*8 points*) A function called `ordered_freq_distribution` with two parameters: `infile` and `ordered_distfile`. The first parameter, `infile`, is the name of the file for which we want to compute the frequency distribution. The second parameter, `ordered_distfile` is the name of the file into which the frequency distribution is to be written in *sorted order of the frequency.* That is, the words should be printed into the file in *decreasing* order of frequency. If several words have the same frequency, they should be listed in alphabetical order. Use string formatting to make sure that the output in `ordered_distfile` is neatly formatted.

Once again, in order to carry out this task, you must first create a dictionary to store the frequency distribution by calling the helper function `freq_dictionary` described in #(3) below. Remember to close all files used in the function.

3. (*10 points*) A function called `freq_dictionary` with a single parameter `infile`, the name of the text file for which we want to compute the frequency distribution. The function should first open `infile` for reading. It should then create a dictionary with key:value pairs, where the key is a word occurring in `infile` and its associated value is the frequency of that character in the file i.e., the number of times it occurs in the file. Refer to the earlier discussion for our definition of a word. Note that every word occurring in the file should be included in the dictionary (even if they are nonsense words). Words that do not occur in the file should not be included in the dictionary. The function must **return** the dictionary. Remember to close the file *prior to* the return statement.

For example, suppose `poem.txt` is the name of a file containing the following poem (courtesy of Dr. Seuss):

```
You have brains in your head.
You have feet in your shoes.
You can steer yourself in
any direction you choose.
```

Then, after the function call `freq_distribution("poem.txt", "freq.txt")`, a file called `freq.txt` should contain the following:

```
any             1
brains          1
can             1
```

```
choose        1
direction     1
feet          1
have          2
head          1
in            3
shoes         1
steer         1
you           4
your          2
yourself      1
```

After the function call `ordered_freq_distribution("essay.txt", "ordfreq.txt")`, a file called `ordfreq.txt` should contain the following:

```
you           4
in            3
have          2
your          2
any           1
brains        1
can           1
choose        1
direction     1
feet          1
head          1
shoes         1
steer         1
yourself      1
```

Submission Guidelines

Implement the first problem in a file called `problem1.py` and the second one in a file called `problem2.py`. *Your name and RUID should appear as a comment at the very top of each file.*

Test each of your programs thoroughly before submitting your homework. When you are ready to submit, upload your files on Sakai as follows:

1. Use your web browser to go to the website `https:sakai.rutgers.edu`.
2. Log in by using your Rutgers login id and password, and click on the `OBJECT-ORIENTED PROG S16` tab.
3. Click on the 'Assignments' link on the left and go to 'Homework Assignment #2' to find the homework file (`hw2.pdf`) and some sample text files (`poem.txt` and `testfile.txt`) that you can use to test Problem #2. .
4. Use this same link to upload your homework files (`problem1.py` and `problem2.py`) when you are ready to submit.

You must submit your assignment at or before 11:59PM on February 26, 2016.