Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2015

Lab 10 - Working with Dictionaries

Attendance/Demo

To receive credit for this lab, you must demonstrate the code you complete. **Also, you must submit your lab work to cuLearn by the end of the lab period**. (Instructions are provided in the *Wrap Up* section at the end of this handout.)

When you have finished all the exercises, call a TA, who will review the code you wrote. For those who don't finish early, a TA will ask you to demonstrate whatever code you've completed, starting about 30 minutes before the end of the lab period. **Any unfinished exercises should be treated as "homework"; complete these on your own time, before your next lab.**

Getting Started

1. Launch Wing IDE 101. Check the message Python displays in the shell window and verify that Wing is running Python version 3.4. If another version is running, ask your instructor or a TA for help to reconfigure Wing to run Python 3.4.

Exercise 1

Step 1: Download build_word_list.py, sons_of_martha.txt and two_cities.txt from cuLearn and open this file in Wing IDE.

Function `build_word_list` was presented in a recent lecture. Call this function from the shell, once with `'two_cities.txt'` as the argument and once with `'sons_of_martha.txt'`. Observe the lists returned by the function. Read the function's code and make sure you can answer these questions:

- How are the individual words extracted from the lines of text read from a file?
- How does the function ensure that duplicate words aren't stored in the word list?
- How is the list sorted into ascending order?

Step 2: Review the OPT examples from the Nov. 23 and Nov. 25 lectures (count the occurrences of each number in a sequence of random numbers, using a list of counters and a dictionary of counters). Links to these examples are in the Lectures section of the cuLearn course page.

Step 3: Download word_histogram.py from cuLearn and open this file in Wing IDE. Read function `build_histogram`. This function uses code "borrowed" from `build_word_list` to read lines of text from a file, split each line into words, and remove punctuation. It uses a dictionary to count the occurrences of each word, using the same approach shown in the OPT

examples.

Call `build_histogram` from the shell, once with `'two_cities.txt'` as the argument and once with `'sons_of_martha.txt'`. Observe the histograms (dictionaries) returned by the function.

Step 4: Read function `most_frequent_word`. Call this function from the shell, passing it the histogram of the words in sons_of_martha.txt. Which word occurs most frequently in that file? How often does it occur?

Exercise 2

In word_histogram.py, define a function named `words_with_frequency`. This function is passed a histogram returned by `build_histogram` and a positive integer, `n`. Here is the function header and docstring:

```
def words_with_frequency(hist, n):
    """ (dict, int) -> list of str

    Returns a list of all words in dictionary hist that occur
    with frequency n. The list is sorted in ascending order.

    >>> hist = build_histogram('sons_of_martha.txt')
    >>> words_with_frequency(hist, 1)  # Which words occur once
                                       # in the file?
    >>> words_with_frequency(hist, 5)  # Which words occur five
                                       # times?
    """
```

Test your function using the histogram for two_cities.txt.

`>>> hist = build_histogram('two_cities.txt')`
`>>> words_with_frequency(hist, 1)` should return the sorted list:

```
[best, worst]
```

`>>> words_with_frequency(hist, 2)` should return the sorted list:

```
[it, of, the, times, was]
```

Test your function using the histogram for sons_of_martha.txt.

Exercise 3

A *concordance* is an alphabetical listing of the words in a file, along with the line numbers in which the each word occurs. A dictionary is a natural data structure for representing a concordance. Each word in the file will be used as a key, while the value associated with the key will be a list of the line numbers of the lines in which the word appears.

In Wing IDE 101, create a new file and save it with the name concordance.py.

In concordance.py, write a Python script (program) that produces and prints a concordance. For example, if a file contains:

```
It was the best of times.
It was the worst of times.
```

the script's output will be:

```
best : [1]
it : [1, 2]
of : [1, 2]
the : [1, 2]
times : [1, 2]
was : [1, 2]
worst : [2]
```

Notice that the words are printed in alphabetical order.

The same word can appear in a line more than once, so your script must ensure that there are no duplicate line numbers in each list; that is, it must ensure a line number is appended to a list only if it is not already in the list. For example, if a file contains:

```
Hello, hello, hello
```

the script's output will be:

```
hello : [1]
```

not:

```
hello : [1, 1, 1]
```

Feel free to use the code in build_words_list.py and word_histogram.py as a starting point. Feel free to define one or more functions that are called by your script. I recommend using an iterative, incremental approach, in which you code and test your script in stages (this was demonstrated in a recent lecture), rather than attempting to write the entire script before you start to test and debug it.

This exercise was adapted from an example prepared by Tim Budd at Oregon State University.

Wrap-up

1. Remember to have a TA review your solutions to the exercises, assign a grade (Satisfactory, Marginal or Unsatisfactory) and have you initial the demo/sign-out sheet.

2. Before you leave the lab, log in to cuLearn and submit concordance.py. (Do not submit

your edited word_histogram.py file.) To do this:

2.1. Click the Submit Lab 10 link. A page containing instructions and your submission status will be displayed. After you've read the instructions, click the Add submission button. A page containing a File submissions box will appear. Drag concordance.py to the File submissions box. **Do not submit a file with a different name** (if your module has a different name, rename it before submitting it). **Do not submit another type of file (e.g., a zip file, a RAR file, a .txt file, etc.)**

2.2. After the icon for concordance.py appears in the box, click the Save changes button. At this point, the submission status of your file is "Draft (not submitted)". If you're ready to finish submitting the file, jump to Step 2.4. If you aren't ready to do this; for example, you want to do some more work on the code and resubmit it later, you can leave the file with "draft" submission status. When you're ready to submit the final version, you can replace or delete your "draft" file submission, by following the instructions in Step 2.3, then do Step 2.4.

2.3. You can replace or delete the file by clicking the Edit my submission button. The page containing the File submissions box will appear.

2.3.1. To overwrite a file you previously submitted with a file having the same name, drag another copy of the file to the File submissions box, then click the Overwrite button when you are told the file exists ("There is already a file called..."). After the icon for the file reappears in the box, click the Save changes button.

2.3.2. To delete a file you previously submitted, click its icon. A dialogue box will appear. Click the Delete button., then click the OK button when you are asked, "Are you sure you want to delete this file?" After the icon for the file disappears, click the Save changes button.

2.4. Once you're sure that you don't want to make any changes, click the Submit assignment button. A Submit assignment page will be displayed containing the message, "Are you sure you want to submit your work for grading? You will not be able to make any more changes." Click the Continue button to confirm that you are ready to submit your lab work. This will change the submission status to "Submitted for grading".

There is a challenge exercise on the next page.

Challenge Exercise

My nephew's Grade 4 math textbook contained this exercise, which stumped the parents of the students in his math class (as well as the teacher!):

Letters `e`, `n`, `o`, `s`, `u`, `y` represent integers between 0 and 9, inclusive. Each letter represents a different integer; for example, if `e` represents 2, then `n` cannot be 2. Determine all values of `e`, `n`, `o`, `s`, `u`, and `y` that satisfy the sum:

```
  see
+ you
 ----
 soon
```

A bit of analysis revealed that this problem has more than a few solutions, so I decided to write a Python script to crank out the numbers.

Challenge: write a Python script that calculates all integers `e`, `n`, `o`, `s`, `u`, `y` such that `see + you` equals `soon`. Remember, each letter represents a different integer between 0 and 9. Print these numbers using the format `see` + `you` = `soon`, sorted in ascending order. The output should look like this:

```
99 + 124 = 223
99 + 125 = 224
99 + 126 = 225
   ...
199 + 803 = 1002
   ...
199 + 807 = 1006
```

This problem can be solved with less than 30 lines of code, if you use lists, sets and tuples. You do not need to use dictionaries. Hint: use nested loops to generate all the integers; for example:

```
for e in ...:
    for n in ...:
        for o in ...:
           ...
           # do something with e, n, o, s, u, y
```

I'll post my solution after all sections have finished Lab 10.