Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2017

Lab 10 - Working with Dictionaries

Demo/Grading

When you have finished all the exercises, call a TA, who will review your solutions, ask you to demonstrate them, and assign a grade. For those who don't finish early, a TA will will grade the work 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.**

Exercise 1

Step 1: Download build_word_list.py, sons_of_martha.txt and two_cities.txt from cuLearn. Open build_word_list.py in Wing 101.

Function `build_word_list` was presented in a recent lecture. Review the definition of this function.

Call `build_word_list` 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: In a recent lecture, you saw a script that used a dictionary to count the occurrences of each number in a sequence of random numbers. This scripts is available as Python Tutor example (links are in the *Lecture Materials* section of the cuLearn course page). Review the code for the script and use PyTutor to visualize its execution.

Step 3: A *histogram* is a dictionary in which the keys are words. The value associated with each key is is the number of occurrences of that word in a file.

Download word_histogram.py from cuLearn and open this file in Wing 101. Function `build_histogram` was presented in a recent lecture. Review the definition of this function. Notice that 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 as the "count the occurrences of random numbers" script.

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: Function `most_frequent_word` was presented in a recent lecture. Review the definition of this function.

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 of str, int pairs; 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]
```

Now 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 101, create a new file and save it with the name concordance.py.

In concordance.py, define a function named `build_concordance`. Here is the function header and docstring:

```
def build_concordance(filename):
    """ (str) -> dict of str, list pairs

    Return a dictionary in which the keys are the words in the
    specified file. The value associated with each key is a list
    containing the line numbers of all the lines in which each word
    occurs.

    >>> concordance = build_concordance('sons_of_martha.txt')
    """
```

For example, if a file contains the word `Python` on lines 1, 7 and 12, the concordance will contain this key/value pair: `'Python' : [1, 7, 12]`

The same word can appear in a line more than once, so your function 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's first line is:

```
Hello, hello, hello
```

the concordance should contain this key/value pair:

```
'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. We recommend using an iterative, incremental approach, in which you code and test your function in stages (this was demonstrated in a recent lecture), rather than attempting to write the entire function before you start to test and debug it.

Test your function using the file two_cities.txt:

```
>>> concordance = build_concordance('two_cities.txt')
```

When `concordance` is evaluated, Python should display this dictionary:

```
>>> concordance

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

Exercise 4

Write a Python script (program) that prompts the user to type the name of a text file. The script produces and prints a concordance of the words in that file. 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.

Your script must call your `build_concordance` function from Exercise 3.

Exercises 3 and 4 were 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 grading/sign-out sheet.

2. Remember to backup your project folder before you leave the lab; for example, copy it to a flash drive and/or a cloud-based file storage service.

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
```