Carleton University
Department of Systems and Computer Engineering
SYSC 1005 - Introduction to Software Development - Fall 2018

Lab 10 - Working with Dictionaries

Demo/Grading

After you finish all the exercises, a TA will review your solutions, ask you to demonstrate some of them, and assign a grade. For those who don't finish early, a TA 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.**

Part 1

This part reviews three functions that were presented in recent lectures:

- `build_word_list`, in module build_word_list_final.py;
- `build_histogram` and `most_frequent_word`, in module word_histogram.py.

Exercise 1

Step 1: Download build_word_list_final.py, sons_of_martha.txt and two_cities.txt from the Lab 10 folder on cuLearn. Open build_word_list_final.py in Wing 101.

Step 2: The test script at the end of the module calls `build_word_list`, using `'sons_of_martha.txt'` as the argument, and prints the list returned by the function.

Run the module, and observe the word list. Review the definition of `build_word_list` 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?

Exercise 2

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.

Step 1: Download word_histogram.py from the Lab 10 folder in cuLearn and open this file in Wing 101.

Step 2: Review the definition of `build_histogram`. Notice that this function reuses code that was developed for `build_word_list` to read lines of text from a file, split each line into words, and remove punctuation. The major difference is that, instead of building and returning a sorted list of the distinct words in a file, `build_histogram` uses a dictionary to count the

number occurrences of each word in a file.

Step 3: The test script at the end of the module calls `build_histogram`, using `'sons_of_martha.txt'` as the argument, and prints the histogram returned by the function. Run the module, and observe the histogram. Which word occurs most frequently in the file? How often does it occur?

Step 4: Review the definition of `most_frequent_word`.

Step 5: Edit the test script to call `most_frequent_word`, passing it the histogram returned words by `build_histogram`. In addition to displaying the histogram, the script should print the word that occurs most frequently and the number of occurrences, like this:

```
The most frequently occurring word is: word
# of occurrences: occurrences
```

Run the module. Which word occurs most frequently in the file? How often does it occur?

Part 2

Exercise 3

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?
    """
```

Hint: the function body requires fewer than 10 lines of code.

Test your function using the histogram for two_cities.txt:

```
>>> hist = build_histogram('two_cities.txt')
>>> words_with_frequency(hist, 1)
```

This should display the sorted list:

```
[best, worst]
```

```
>>> words_with_frequency(hist, 2)
```

This should return the sorted list:

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

Now test your function using the histogram for sons_of_martha.txt.

Part 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 all the lines in which the word appears.

Exercise 4

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]
```

Hint: you should be able to reuse much of the code in `build_word_list` (Exercise 1) and `build_histogram` (Exercise 2). Notice that `build_histogram` and `build_concordance` both return dictionaries. In the histogram, the keys are words and the value associated with each key is is the number of occurrences of that word in a file. In the concordance, the keys are words and the value associated with the key is a list of line numbers.

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 two_cities.txt:

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

When `concordance` is evaluated, Python should display this dictionary (note that the ordering of the key/value pairs may differ from what is shown here):

```
>>> concordance

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

Now test your function using sons_of_martha.txt.

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 files before you leave the lab; for example, copy them to a flash drive and/or a cloud-based file storage service.

There is an extra-practice exercise on the next page, which requires you to write a short script that uses your `build_concordance` function from Exercise 4.

Extra Practice - Exercise 5

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. The words must be printed in alphabetical order. Hint: Unlike lists, Python's `dict` type doesn't provide a function or method that sorts a dictionary according to its keys.

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]
```

Your script must call your `build_concordance` function from Exercise 4.

Exercises 4 and 5 were adapted from an example prepared by Tim Budd at Oregon State University.