Carleton University
SYSC 1005 – Introduction to Software Development – Fall 2013
Lab 12 - Dictionaries

Attendance/Demo

To receive credit for this lab, you must make an effort to complete the exercises, and demonstrate your work.

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. Finish any exercises that you don't complete by the end of the lab on your own time.

Reference

Think Python, Chapter 11, *Dictionaries*. (Ignore Section 11.5, *Memos*.) This book's URL is:

http://greenteapress.com/thinkpython/thinkpython.pdf

Getting Started

Step 1: Create a new folder named Lab 12.

Step 2: Download Cimpl.py and the three image (JPEG) files that you used throughout the term to your Lab 12 folder.

Step 3: Download convolution_filter.py and Lab_12_modified_images.zip from cuLearn to your Lab 12 folder.

Lab_12_modified_images.zip is a compressed (zipped) folder that contains the images produced by my solutions to Part 1. You can use these images to help you determine if the modified images produced by your filter is correct.

Convolution and Image Processing

Convolution kernels and their application to image processing were presented in recent lectures. A brief discussion of the technique can be found here:

http://en.wikipedia.org/wiki/Kernel_(image_processing)

Launch Wing IDE 101 and open convolution_filter.py.

I've made one change to the `convolution_filter` function I presented in class. Instead of *normalizing* the new component values by dividing them by 9 (the number of elements in the convolution kernel), the function now divides the components by the sum of the values in the kernel. This change improves the effects that are achieved with some kernels.

The test function in convolution_filter.py demonstrates `convolution_filter`, using a a kernel

for blurring images. Different effects can be obtained by passing different kernels to `convolution_filter`. Here are the matrices for five different kernels:

blur

```
1  1  1
1  1  1
1  1  1
```

sharpen

```
 0   -3   0
-3   21  -3
 0   -3   0
```

emboss

```
-18  -9   0
 -9   9   9
  0   9  18
```

edge detect 1

```
0    9   0
9  -36   9
0    9   0
```

edge detect 2

```
-9  -9  -9
-9  72  -9
-9  -9  -9
```

Exercise 1

In convolution_filter.py, define a function named `build_kernel_table` that returns a dictionary that has been initialized with the five 3-by-3 kernels shown earlier. The dictionary keys are strings containing the names of the filters; for example, `"blur"`, `"sharpen"`, etc. The value associated with each key is a kernel, which should be implemented as a tuple containing three tuples (for an example, look at the initialization of `blur_kernel` in convolution_filter.py).

Use the shell to interactively test `build_kernel_table`. For example, the following statements will retrieve and display the blur kernel

```
>>> kernels = build_kernel_table()
>>> kernel = kernels["blur"]
>>> kernel
((1, 1, 1), (1, 1, 1), (1, 1, 1))
```

Exercise 2

Modify function `convolution_filter` so that it is passed an image, the table (dictionary) of kernels returned by `build_kernel_table`, and the name of a kernel. The function will look up the specified kernel in the table, then apply it to the image. For example, to blur an image, we call `convolution_filter` this way:

```
>>> kernels = build_kernel_table()
        ...
>>> new_image = convolution_filter(img, kernels, "blur")
>>> show(new_image)
```

Interactively test your `convolution_filter` function. Try all of the kernels in the table.

Text Processing

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 will be a list of the line numbers of the lines in which the word appears.

Write a Python 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 program'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.

Remember that the same word can appear in a line more than once. Ensure that the line number is appended to the list only if it is not already in the list.

Feel free to use the code in build_words_list.py, which is posted on cuLearn (see Week 10, Lecture 2) , as a starting point. Feel free to define one or more functions that are called by your program. I recommend an iterative, incremental approach, in which you code and test your program in stages, rather than writing the entire program 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 convolution_filter.py file and your concordance program give you a grade (Satisfactory, Marginal or Unsatisfactory) before you leave the lab.
2. Remember to save a copy of your work (copy it to a flash drive, or email a copy to yourself, or store it on your M: drive - remember, files left on your desktop or in your Documents folder are not guaranteed to be there the next time you log in).
3. Log in to cuLearn, click the Submit Lab 12 link and submit convolution_filter.py. You do not have to submit your concordance program. After you click the Add submission button, drag the file to the File submissions box. After the icon for the file appears in the box, click the Save changes button. At this point, the submission status for your files is "Draft (not submitted)". You can resubmit the file by clicking the Edit my submission button. After you've finished uploading the file, remember to click the Submit assignment button to change the submission status to "Submitted for grading".