6.189 – Intro to Python
IAP 2008 – Class 8
Lead: Aseem Kishore

Lab 10: Compound Dictionaries

Quick Reference

`D = {}` – creates an empty dictionary
`D = {key1:value1, …}` – creates a non-empty dictionary

`D[key]` – returns the value that's mapped to by key. (What if there's no such key?)
`D[key] = newvalue` – maps newvalue to key. Overwrites any previous value.
`del D[key]` – deletes the mapping with that key from D.

`len(D)` – returns the number of entries (mappings) in D.
`x in D`, `x not in D` – checks whether the **key** x is in the dictionary D.

`D.items()` – returns the entries as a list of (key, value) tuples.
`for k in D` – iterates over all of the **keys** in D.

Problem 1 – Inventory Finder

Download the inventory.py file. The file shows eight different items, each having a name, a price and a count, like so:

HAMMER = "hammer"
HAMMER_PRICE = 10
HAMMER_COUNT = 100

We're going to consider that customers generally come in with an idea of how much money they want to spend. So we're going to think of items as either CHEAP (under $20), MODERATE (between $20 and $100) or EXPENSIVE (over $100).

First, fill in the variable inventory so that all of the data for the eight items is inside inventory. Make sure that you maintain the notion of CHEAP, MODERATE and EXPENSIVE. Then, implement the function get_info that takes a cheapness and returns a list of information about each item that falls under that category, as the function's information says.

Important: there should **NOT** be a loop inside this function. Our inventory is small, but for a giant store, the inventory will be big. The store can't afford to waste time looping over all of the inventory every time a customer has a request.

When you're finished, just run the program. All of the testing lines should print True.

Problem 2 – Indexing the Web, Part 2

So we have a working search engine. That's great! But how do we know which sites are better than others? Right now, they're just returning the sites in an arbitrary order (remember, dictionaries are unordered). In this problem, we'll implement a ranking system.

What will we rank based on? Google used an innovative ranking system that ranked a page higher if more *other* pages linked to it. We can't do that unfortunately, because that requires a considerable understanding of graph theory, so what else can we do? Well, before Google, most engines ranked based on either the **frequency** (i.e. number of hits) of search terms inside the page, or by the **percentage** of those search terms within the page's text. We'll go with the frequency arbitrarily – we found after Google that neither of these measures are particularly good, and there isn't a clear advantage between the two.

To begin, download the following files:

webindexer2.py – this is the file in which you'll write all of your code.
websearch2.py – this completed program is an updated search engine that will use your new index with a ranking system.

Again, take a look at the main program, **websearch2.py**. It's almost identical to the previous version, but you can see that it now expects to have tuples of (site, frequency) rather than just the sites themselves. This way, it is able to display how many hits each site has. It also expects that the sites are already sorted/ranked from highest to lowest frequency.

So let's take a look at **webindexer2.py**. Again, it's almost identical to the previous version, but the descriptions for the search functions now state that frequencyis returned along with each site, and the sites are sorted by rank.

In order to rank each site by the frequency of search terms in it, we'll have to store the information in our index.

To begin, you can copy your functions' code from **webindexer1.py** into **webindexer2.py**, but you don't have to.

Task 1 – Implement the **index_site** function. What information will each site in the index need to store with it? What's the best way to store this information? If we have more than one choice, which choice is mutable, and which one is immutable? While we're building the index, we'll be repeatedly making changes, so which choice is better?

Hints: If you're stuck, think very logically. When I'm searching, I have a word. I want to be able to look up this word and get what information? The information needs to be enough for me to sort it.

Now that we've taken care of indexing, we can again move on to searching. And again, we'll tackle one word first before multiple words. This should be very similar to your previous function, but we have to do one additional thing: sort the results based on frequency.

Task 2 – Implement the **search_single_word** function. We have to return a list of (site, frequency) tuples. If we have a list L of these tuples, to sort them, do this:

```
L.sort(key = lambda pair: pair[1], reverse = True)
```

Don't worry about what this means yet, but if you're interested, we can explain.

And again, now that we can handle one word, we'll handle multiple words. The same logic applies as before, but again, we have to sort the results before returning them.

Task 3 – Implement the **search_multiple_words** function. The argument `words` is a list, not a string. Make sure you don't return duplicate sites in your list! And as before, make sure you sort the list (using the same statement as above).

You should now have a working indexer with a ranking system, so run **websearch2.py** and try it out! And for some real fun, don't use the smallest set of files. Use the 20 set or the 50 set to see the ranking really come into play.

As before, on the next page, I've pasted my output for a few searches from the **mitsites20.txt** file. If your output is quite different, you may have done something wrong. If it's just slightly different, it may just be a change in the pages (e.g. web.mit.edu) from when I indexed the site to when you did.

Here is my output:

```
6.189 Web Search! (version 2)

Building the index... (this may take a while)
Done!

At any time, you may search for "QUIT" to quit.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

What would you like to search for? hockfield

1 site(s) found with the terms "hockfield":

(1 hits)    http://mitsloan.mit.edu/

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

What would you like to search for? susan hockfield

2 site(s) found with the terms "susan hockfield":

(2 hits)    http://mitsloan.mit.edu/
(1 hits)    http://mitpress.mit.edu/

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

What would you like to search for? computer science

11 site(s) found with the terms "computer science":

(102 hits)  http://ocw.mit.edu/OcwWeb/web/courses/courses/index.htm
(12 hits)   http://mitworld.mit.edu/
(7 hits)    http://dspace.mit.edu/
(6 hits)    http://ocw.mit.edu/
(4 hits)    http://www.eecs.mit.edu/
(4 hits)    http://mitpress.mit.edu/
(4 hits)    http://www.csail.mit.edu/
(3 hits)    http://dmse.mit.edu/
(1 hits)    http://www.media.mit.edu/
(1 hits)    http://architecture.mit.edu/
(1 hits)    http://laptop.media.mit.edu/

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

What would you like to search for? python

No sites found with the terms "python".
Try a broader search.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

What would you like to search for? QUIT

Thanks for searching!
```