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

Lab 9: Introduction to 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.

Problem 1 – Names and Ages

(You can copy the below code into a file, but you can also optionally download this code in a pre-written file, namesages.py.)

Define two lists at the top of your file:

```
NAMES = [“Alice”, “Bob”, “Cathy”, “Dan”, “Ed”, “Frank”,
         “Gary”, “Helen”, “Irene”, “Jack”, “Kelly”, “Larry”]
AGES = [20, 21, 18, 18, 19, 20, 20, 19, 19, 19, 22, 19]
```

These lists match up, so Alice's age is 20, Bob's age is 21, and so on.

Write a program that combines these lists into a dictionary. Then, write a function that, given an age, returns the names of all the people who are that age.

Test your program and function by running these lines: (replace `people` with your function)

```
print people(18) == [“Cathy”, “Dan”]
print people(19) == [“Ed”, “Helen”, “Irene”, “Jack”, “Larry”]
print people(20) == [“Alice”, “Frank”, “Gary”]
print people(21) == [“Bob”]
print people(22) == [“Kelly”]
print people(23) == []
```

All lines should print True. If the last line is giving you an error, look at the name of the error.

Problem 2 – Indexing the Web, Part 1

Every day, we search the web. When we type in a search term like "MIT" into Google, how does Google give us our results so fast? That's a loaded question to answer, but in this problem we'll study the bulk of the response.

In computer science, we model the web as a graph, where each node (vertex) is a webpage and each edge is a link we can click on. When we navigate the web, we're navigating this graph – each link we click on moves us from a node to a neighboring node on the web.

Google wants to search every page on the internet, so it has to somehow navigate this graph. That's a daunting task, and how exactly you go about navigating the web so that you can reach every page is a major challenge, so we're going to ignore that. Instead, we're going to assume that we've seen every page and we have a giant list of pages that we've read.

So now, when we search for something on Google, does Google scan every page and see which ones have the words we're searching for? Well, there are on the order of 10 billion webpages out there (maybe closer to 100 billion now*), so even if Google could check an average page for your words in 0.01 seconds, it would take on the order of 100 million seconds, or 3 years. So clearly, Google isn't doing the searching on the spot.

To fix this problem, we're going to use the analogy of a textbook. We want to find which pages mention a certain word. Similar to Google's problem above, if we read page by page and kept track of which pages mentioned the word, it would take us hours, maybe days.

So instead, most textbooks come with a pre-built **index**. The index allows us to easily look up which pages contain the word we're looking for. In the same way, Google has pre-built (and is continually updating) their index. Their index allows them to look up any word, and see what sites mention that word.

So now, when you search for a word, Google quickly looks up the word in its index (a massively faster procedure) and returns the sites that mention that word.** In this part of the problem, we're going to build a similar index for a small number of sites. We'll then be able to use the index as part of a simple search engine.

To begin, download the following files:

webindexer1.py – this is the file in which you'll write all of your code.
websearch1.py – this completed program is a search engine that will use your index.
htmltext.py – this module takes care of parsing HTML into text.
smallsites.txt, **mitsites20.txt**, **mitsites50.txt** – these files list the URLs of 10, 20 and 50 sites respectively that we will index and use for our searches.

* source: http://www.boutell.com/newfaq/misc/sizeofweb.html

** The use of an index is nothing new. Every search engine before Google used an index too – you have to. Google's fame and success came mostly due to their **rankings** (the order of the results). We'll explore rankings in part 2, but unfortunately, understanding Google's ranking algorithm requires an understanding of graph theory. So instead, we'll use an older approach.

Let's take a look at the main program, **websearch1.py**. This program lets the user repeatedly enter a search string and view the results. This isn't a difficult program to write, and it's nothing you haven't seen before, so I went ahead and wrote the full thing. Take a look and make sure you understand it.

It uses two functions which aren't in the program, **build_index** and **search_multiple_words**. Both are part of **webindexer1.py**, so let's open that up and take a look. This is a stub file where I've defined a bunch of functions for you, and some are implemented whereas others are not.

Near the top, we've defined a variable called **index**, which is initially set to an empty dictionary. Below that, I've implemented a few helper functions to take care of some of the logistics. At the bottom is the **build_index** function. It retrieves all the sites listed in **FILENAME** (initially set to the smallest file), reads them and finally indexes them. We'll have to implement the function **index_site**, which does the actual indexing.

Task 1 – Implement the **index_site** function. If you're stuck, this is analogous to **index_page** for a textbook. What does the index at the back of a textbook look like? In terms of a dictionary, what are the keys and what are the values? How will you modify the dictionary if you're given another page?

Hints: `"he came, he saw".split()` returns `["he", "came,", "he", "saw"]`. Don't worry about punctuation. Make sure to use only lowercase words, as the search program converts all search strings into lowercase first.

Once we've indexed our sites, we're able to search them! We see that the search program calls **search_multiple_words** in order to handle a multi-word search string. Before we think about multiple words, let's solve the problem of searching for one word.

Task 2 – Implement the **search_single_word** function. If you've designed your index well, this should be pretty trivial. But make sure you're accounting for all cases!

Now that we can handle one word, we'll use our solution to handle multiple words. The only question is, should we treat a multiple-word search as "sites that have ALL of these words", or "sites that have ANY of these words"? The latter is considerably easier to implement than the former, and it's what most search engines do as well, so we'll do "any".

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!

You should now have a working indexer, so run **websearch1.py** and try it out!

By default, **FILENAME** is set to the smallest file, which lists 10 sites (3 at google, 4 at MIT and 3 at facebook). These sites were chosen specifically because they are small, so they're fast to load. If your search engine seems to work fine, try **mitsites20.txt**, which lists the top 20 sites for a Google search of "MIT". Finally, if you are willing to wait a few minutes, try **mitsites50.txt**, which lists the top 50 sites for the same search.

On the next page, I've pasted my output for a few searches from the default smallest 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 1)

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? people google

6 site(s) found with the terms "people google":

http://www.google.com/intl/en/ads/
http://web.mit.edu/
http://www.eecs.mit.edu/
http://www.facebook.com/
http://www.google.com/
http://images.google.com/

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

What would you like to search for? ads

2 site(s) found with the terms "ads":

http://www.google.com/intl/en/ads/
http://www.facebook.com/ads/

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

What would you like to search for? ADS

2 site(s) found with the terms "ADS":

http://www.google.com/intl/en/ads/
http://www.facebook.com/ads/

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

What would you like to search for? advertisements

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

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

What would you like to search for? QUIT

Thanks for searching!
```