Handout 4 – Dictionaries and Web Indexer Part 1

[…]

- Dictionaries are a **table** or **map** of **key-value pairs**. We say that each key **maps** to its value.

- Whereas lists are indexed by their positions, e.g. L[0], L[1], L[2], ..., dictionaries are indexed by their keys, i.e. D[key] returns the value mapped to by key.

- If I have a sequence of 100 elements, let's say they're the top 100 baseball players today, a list is really useful because I can quickly and easily access an element **if I know its position**. So if I want the 7th best baseball player, it'll be something like players[i].

Dictionaries are useful when the **order doesn't matter** and there's no notion of position. In other words, we wouldn't use a dictionary to store the top 100 baseball players, but we'd use a dictionary to store which players play for which teams. So if I know a team, the dictionary will let me quickly and easily access the players on that team.

- The general way to use dictionaries when solving a problem is to ask:

1. **What do I know?** and
2. **What do I *want* to know?**

The keys in your dictionary should be based on your answer to number 1, and the values should be the answer to number 2.

In the names-ages example, we knew the age we cared about, and we wanted to know who all had that age. So the dictionary's keys were the various ages, and the values were a list of names.

- On that note, we often use **lists as values** because we have no other way of saying "this key maps to multiple values". So in the names-ages, example, we had multiple people with the age 20, but we can't keep multiple key-value entries with the same key of 20. So we make the value a list.

I think that for most people, syntax is not the worry. Syntax comes easy, and if you're having trouble, it's likely that you'll learn quickly. Conceptual stuff seems to be the hurdle here for most people -- they aren't understanding **when** to use dictionaries or **how** to apply them.

If that's the case, stick to the above guidelines.

So now, the web indexing problem. What was the purpose of building an index? Because we want to be able to to quickly and easily know which sites have the words we care about. In this case, what do we know? The words we care about. And what do we want to know? Which sites have those words.

So... our index (a dictionary) should have words as keys, and for each word, its value should be a list of sites which have that word in them. **Make sure you understand this -- if you don't, ask us in class.**

Really, there's a reason I used the textbook analogy -- it's almost identical to the web problem. Think of

each page in a textbook as a separate site. The index at the back of the textbook lets you quickly and easily see which pages (which sites) have the word in question.

So that's what we want to build. Now, I think some of the implementation stuff tripped people up. Some clarifications:

- you don't have to worry about **ANY** -- I'm going to repeat that, **ANY** -- of the code in the files EXCEPT the three functions you're asked to implement. You don't have to worry about how we connect to the sites and read them, how we parse their HTML, etc. **You do NOT have to call ANY functions inside your three functions. Everything is taken care of for you.**

- the function **index_site** is given two arguments, **both are strings**. The first is **site** -- this is just the site's URL. We need this for the index, to be able to say "these **sites** have the word you're looking for." The second is **text** -- this is one giant string of all the text that was read on that site. Now, what do we need for our index? We need words. We have to get those words from the site. That's where we use split.

- the function **index_site** is **ONLY going to index ONE site. You are not going to be looping over multiple sites in this function.** Think of it as saying "I'm going to add one more page into my textbook. I have an existing index, so I want to ``index this site``, meaning I want to read each word and update the index for each word I read."

- as we've been trying to hammer into you guys over and over -- **DO NOT PRINT INSIDE THESE FUNCTIONS. DO NOT ASK FOR INPUT INSIDE THESE FUNCTIONS.** I kept seeing people print "No sites found with the given word. Try a broader search" inside their **search_single_word** function. No! What should that function return if no sites were found? To answer that question, what does it return when 3 sites are found? A list of 3 sites. When 1 site is found? A list of 1 site. So when no sites are found? An empty list. Don't print anything!!!

- once you can get the search at least working, pat yourself on the back. There are details which make this a little tricky to polish, such as not having duplicate sites in your results, making sure that the site's text is lowercased before you index it, and so on.

[…]