Homework 2-4

Mar. 20, 2012, 2:25 pm

Reminders

For the following problems you may discuss the concepts that will help solve these problems with classmates and course staff. You may *not* simply copy down the answers of your classmates as that is a violation of the collaboration policy. The one exception to this rule are those problems marked as Independent. You may discuss independent problems *with course staff only.*

Task 1:

Even though Python was meant to be a light-weight language, it has grown significantly over the years. It is practically impossible to learn every bit of it in lectures. On the other hand, Python developers have made comprehensive documentations of the language and put them online. It's time for us to learn to use this resource. The more you program, the more you will need it.

a. Go to this website http://docs.python.org/release/2.7.2/library/stdtypes.html#dict. It describes the usage of the dictionary object, operators and functions that are defined on it. Skim the documentation and see if you can relate it to what we've learned in class (it's ok that you do not understand most of it).

b. Using the search function of your web browser (usually it's Ctrl+F) or just your bare eyes, find the descriptions of these three functions:

```
clear()
keys()
pop(key)
```

Study them and complete HW2-4.py according to the instructions given as comments (you do not necessarily have to use these functions).

In this homework, you will write a program that produces a concordance for any given texts in one second. You have leaned all the necessary pieces and all you have to do is to put them together. This does not mean that this homework is easy; on the contrary, it may be the hardest one you have encountered so far. So before you start, bear the following advice in mind:

- When your program is wrong and does not work, you will almost certainly make it more wrong by trying to change random part of it, hoping to stump on the correct answer (putting brackets around stuff, change the indentations, etc). Only make changes when you have good reasons ("Ahuh, I see what I did wrong here: I incremented a list with a number. I should put brackets around that number so that I increment that list with another list." or "This statement should be executed every time the iteration runs, otherwise it makes no sense algorithmically. Let me indent it.")
- Try to read the error message. The error message usually comes with a line number to indicate where the error occurs, together with what error it is ("`global name 'inventory'undefined`" means that you've used a variable that has not been assigned a value; "`key error: Star Wars`" means that you are trying to look up `'Star Wars'` in a dictionary that does not contain it as a key. Sometimes it is obvious how to fix them. If it is not clear, try to run the program on paper (using the model we gave you in class 2-3) and you may find you've called an object-specific function on the wrong type of object or asking for the value of a key which is not in the dictionary.
- All other advices in HW2-3, in which you wrote a program that counts the vocabulary size of some texts.

Task 2:

Consider the need of organizing phonebooks. My phonebook is in the following format:

```
[['Peter', '345-8766'],
['Lois', '459-2346'],
['Stewie', '345-2354'],
['Peter', '854-1198']]
```

That is, a list of lists. Each enclosed list contains two elements, the first being a name and the second a telephone number. Note that I may have

several phone numbers for the same name. I want to organize my phonebooks, so that it is a dictionary that maps names to lists of phonenumbers. For this example, the organized phonebook should look like this:

```
{'Peter':  ['345-8766', '854-1198'],
'Lois':  ['459-2346'],
'Stewie':  ['345-2354']}
```

Write a function that does this job (takes a list of that format, returns a dictionary). You do not have to hand it in, but keep it as a reference as you build your concordance. Getting this function correct is very important for the rest of this homework. Test it on contrived examples (make up some phonebooks).

Important: Review the first part of the homework for how to query a dictionary with a key and how to add/change a key and its value in a dictionary. You will also need to check if a dictionary contains a certain key by using this expression:

```
k in dict
```

which evaluates to `True` if `k` is a key in dictionary `dict`, or `False` otherwise. Try it with your own examples first.

Task 3:

Let us first develop the algorithm for creating a concordance. Please pay attention to how I break up a big task into small ones and attack them separately (you will have to do that for your project!).

First we define the goal. A concordance maps words to all their occurrences in the book, with surrounding texts. That sounds like what a dictionary does. The keys of such a dictionary are strings (words), and the values are list of strings (surrounding texts). To save space and to make it simpler, the values can be a list of integers, which represent the *positions* of the words in the texts. So the function that *builds* the concordance (`build_concordance`)

- Takes a string as its argument, for which we build a concordance.
- Returns a dictionary that maps words to a list of integers. (The integers tell where that word occurs in the text)

We then need another function to query the concordance to make it useful. The function that queries a concordance (`print_concordance`)

- Takes a word, a concordance (dictionary), a string (text) as arguments
- Prints all occurrences of the word in the text, with surrounding contexts, using the dictionary.

Task 4:

Let us tackle the function `build_concordance` first. In the starter code for this homework, the function `build_concordance` just iterates through some text, and prints out matches with positions. Make sure you understand what it is doing (and run it!).

In each iteration, you get a word and its postion. You want to organize them into a dictionary that maps a word to a list of positions. Hmmm... does that remind you of the warmup question? Modify the function `build_concordance` so that it returns a dictionary that maps a word to a lists of position.

Task 5:

`print_concordance` is relatively simple. Given a word, a concordance returned by `build_concordance`, and the corresponding text, you want to print all occurrences of that word in the text with contexts. Here are the steps:

- Query the concordance (dictionary) for all the positions at which the word occurs.
- For each position
  - Form a string that spans the position. (e.g., if your word starts at position `584`, then `text[564:604]` ought to give you reasonable context surrounding that occurrence of the word. How many characters you want to print before and after the word is your choice)
  - Print that string
- Return nothing

Task 6:

Run your program without changing the string `s` at the end of the program to something big (we are still in the "debug" mode). In the interactive environment, see what this function call gives you:

```
>>> print_concordance('ishmael', conc, s)
```

What is going on? Can you fix this problem? (Hint: you have to do some extra work when calculating the start and end indices of your strings. Make sure that no index is smaller than 0 or larger than the size of your text)

Task 7:

Extra Credit. Run your program again. This time, change the string `s` at the end of the program to be read from the Moby Dick file. In the interactive environment, try to invoke the function `print_concordance` with proper arguments. Remember, that all the variables you defined in your program outside of any function definitions are still "alive" in the interactive environment that pops up after you run the program.

You may notice that the text printed by your concordance does not look quite good, mainly because of the line breaks. Let us do some cosmetic work. In the last task, you formed some string and printed them out. Before you print them, replace all the line breaks (`\n`) with white spaces. You can choose to write a for-loop, or call a function that you yourself defined, or call this one http://docs.python.org/release/3.1.3/library/re.html#re.sub

Task 8:

Play with your program! You should by now feel really, really proud of yourself.

Handin

Email your program to `cs0931handin@cs.brown.edu` and title the file `'YOURNAME'HW2-5.py` — for example, `GiliKligerHW2-5.txt`.