Homework 2-6

Due: Oct. 25, 2011, 2:25 pm

In this homework, you will finish the program from class and finally find out the vocabulary size of *Moby Dick*! **A word of wisdom**: your program will not work in the first try, a hundred percent sure (I never get my program work the first time). Advices to avoid sitting there clueless about what went wrong:

- Run your program as often as possible (even if it does not yet do what it's supposed to do) just to make sure there is no errors, so that when you get an error you know the source (between where I am and where I last run it).
- Whenever you write a function, test it in the interactive environment first as thoroughly as possible (by supplying contrived arguments) to make sure it does the right thing. Then if your program consists of function calls for the most part, you are confident about the most part, and that narrows down the places where things could go wrong.
- Write everything down and try to run it all at once is equivalent to this scenario: you built an aeroplane from parts according to some diagrams. After you are done, you have one screw left and you have to figure out where to put it back.

Task 1:

The first task makes sure that the starter code works for you and provides you examples of iterating through lists in a different way.

a. Download the starter code `HW2-6.py`. This is essentially the same as the one in class, except that I renamed the function `vocabSize` to `vocab` and instead of returning the size of the vocabulary, it returns the entire vocabulary list.

   Then, I put several lines demonstrating how to iterate through a list in two different ways. You are familiar with the second one, but you should notice that the first one gives you extra power. Note how I am able to print `'Apple comes after Steve Jobs'` using the first way, but I cannot do that with the second.

After the examples, I defined more functions to be used later in this homework (do not worry about them now).

b. Run the program. Then in the interactive environment, after the prompt, call our vocabulary-computing function by typing

```
v = vocab('MobyDick.txt')
```

(make sure that your path is correct). Then inspect the value of variable `v` by typing, unsurprisingly, `v`, afte prompt. You should see a list of strings (words), and if you then inspect the length of `v` (by typing `len(v)`) it should say `853`. I am only computing vocab list for the first `10000` characters in *Moby Dick* (Can you see why the program does not compute the vocab list for the whole text?) If the program does not work as expected, please email `cs0931tas@cs.brown.edu` with what you did and errors you get if any at all.

Task 2:

As we discussed in class, the way we get rid of duplicates in a list is slow. We conceived a faster way to do it assuming we can sort a list fast enough. Let us write a function called `fastNoReplicates` that takes a list of words and returns the unique word list in the new way. (You may want to consult the class slides for general algorithm) The first lines are provided for you. They sort the word list and intialize our result vocab list with the first word in the sorted list.

a. You want to iterate through the sorted word list in the iterating-by-index way (see how I iterated through the Steve Jobs list using the indices). One subtle difference is that you want to start with the *second* word, since (1) you already put the first one in the result list, and (2) later you will compare a word to the previous one, but the first word has no previous. Start a for-loop as such, by supplying `1` instead of `0` as the first argument to `range`.

b. Now we write the body of the loop (statements that will be executed multiple times). Suppose you called your looping variable (the variable right after `for`) `index`. Then each time the loop is run, `index` takes a different value (`0, 1, 2, 3, 4, ...`). What is the expression that evalues to the element of the list at position `index` (Put it in a variable

called `current`)? At the previous position (Put it in a variable called `previous`)?

c. (Still writing the body of the for-loop, be careful with indentation throughout the program!) If `current` is the same as `previous`, we want to say `pass` (`pass` is a special word in Python that means do nothing in this line. Do not try to put quotes or brackets around it); otherwise, we want to append `current` to our result(it's the first time we see it). Use the `+=` short-hand and **Be careful: `current` is a string and `result` is a list.**

d. After the loop is completed (again, be careful with the indentation), the result should hold the list without duplicates. Return it.

Task 3:

Now, let's deploy our new method of removing replicates.

a. Modify the `vocab` function so that it uses `fastNoReplicates` instead of `noReplicates`.

b. Run the program as in Task 1 and inspect the vocabulary list and its size. You should get the same answer as before (both methods are correct, just that the new one is faster).

c. Now, in the function `readFile`, instead of returning `myString[:10000]`, return `myString`. We are no longer afraid of processing large lists!

d. Run the program again. Are you impressed with how fast it computes the vocabulary list? This time, do not try to inspect the whole list because it takes too long to print. You can first look at how many words are in there. Then you can look at slices of this list, for example, `v[1000:2000]`.

Congratulations! You have just completed a **software upgrade**.

Task 4:

As you inspected different portions of the vocabulary list, you may wonder: this is a crapy vocab list! There are numbers, punctuations, mixed cases (`whale` and `Whale` should really be the same word). Now let us fix that.

Essentially, we want to do some clean-ups to the big string we get out of the file before we split it into words. Two possible things to do are turning all letters into lowercases, and replacing all numbers and punctuations with whitespaces (so that `eat,pray,love` can be split as if it is `eat pray love`).

a. I've written a function called `cleanup` for you. *It takes a string and returns a cleaned-up string* (always ask yourself what kind of arguments a function takes and what kind of value it returns). You can see I solve the problem by creating more problems (I have to define more functions for this to work!). First, I use a built-in function called `lower` to turn all letters in my string to lower cases (You can convince yourself it works by running a simple example in the interactive environment). Then `removeNumbers` and `removePunctuations` does what their names suggest (ask yourself, what type of values do they take as arguments? what type of values do they return?)

b. I also wrote `removeNumbers` for you as an example. Read it and make sure that you understand why it works. There are a couple of things to notice:

   - You can iterate through characters in a string in the same way you iterate through items in a list.
   - But you cannot modify a string. So I have to create an empty string first and as I iterate, append new characters to it.
   - When I append a new character, I use the `+=` short-hand because my result string will eventually grow really big.
   - A string that contains a whitespace is not the same as an empty string.

c. Now write the function `removePunctuations` that *takes a string and returns another string, replacing punctuations with whitespaces.*

Task 5:

a. Now, we need to insert this cleanup step into our assembly line. Our assembly line is in the function `vocab` (read→split→remove duplicates). Use the function `cleanup` to cleanup your big string before you split it.

b. Run your program and inspect your vocab list (partially!). You may discover that you may have missed a lot of possible punctuations. You can go back and improve your function if you are a perfectionist.

c. Hoooooraaaay! Now the vocab list looks pretty good. Yes it is not perfect: what about `create, creates, creating, created`?. We will deal with them next time.

Handin

Email your program to `cs0931tas@cs.brown.edu` and title the file `'YOURNAME'HW2-6.py` — for example, `DylanFieldHW2-6.py`.