Individual Programming Project: Concordance Program

Regular Edition 60XP

A concordance is a display of words and their immediate context. It is also called a KWIC index (Key Words In Context). You will be writing a program that will allow a user to search for a word in a text. Your program will output each occurrence of that word followed by the next n words (for your initial tests n will equal 4, but when I evaluate your program I will use different values).

You will need to write 2 functions:

def generate_index(filename):

This function will take a filename as input and create an index.
You will need to design the best data structure to use for this index.

def search(term):

This function will take a search term as input and output all the occurrences of that word in context. Here is an example:

```
>>> generate_index('walden.txt')
>>> search('Walden')
Walden Pond, in Concord, Massachusetts,
Walden Pond was not to
Walden Pond would be a
Walden Pond, nearest to where
Walden nymphs will pardon the
Walden Pond of their own
Walden road, driving a pair
Walden Pond itself. What company
Walden Pond, and stoned it,
Walden Pond was thought to
Walden is on a humble
Walden is blue at one
```

Tuples

Consider the following text:

Traveling by car is like watching a movie, but riding a motorcycle is like being in one.

You will need to go through the text word by word collecting tuples of length *n*. To do this you might have a list, buf, of length n + 1, where each entry is the empty string. So for example, consider the case where n = 3. In this case we will have a word and its three neighbors to the right.

buf[0]	buf[1]	buf[2]	buf[3]
“”	“”	“”	“”

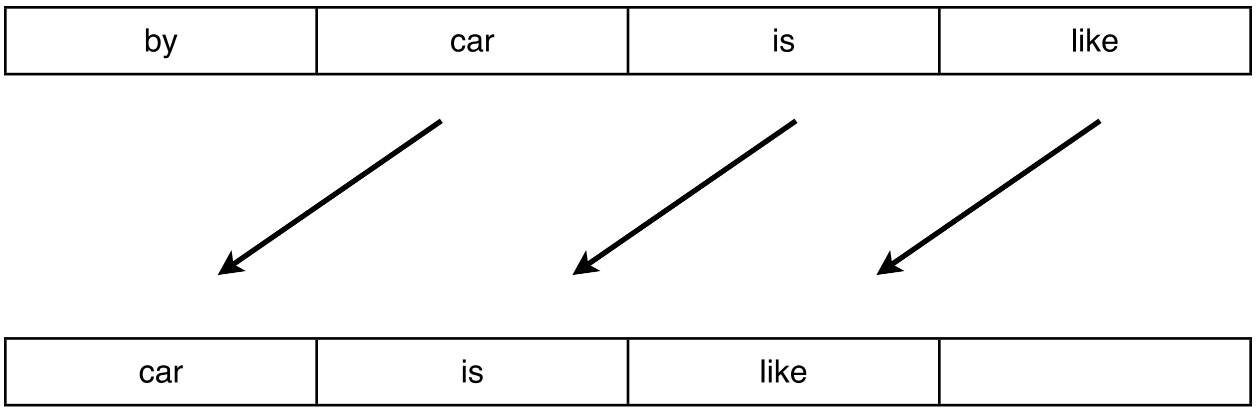
Now we are going to march through the text filling in this buffer:

buf[0]	buf[1]	buf[2]	buf[3]
“”	“”	“”	“”
””	””	””	Traveling
””	””	Traveling	by
””	Traveling	by	car
Traveling	by	car	is
by	car	is	like

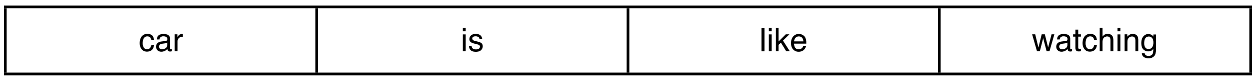
To get a clearer idea of this, let’s start where we left off, with

buf = [‘by’, ‘car’, ‘is’, ‘like’]

and see how we get to the next line. The next word we encounter is *watching*. First, we move whatever is in buf[1] to buf[0]; what is in buf[2] to buf[1] and so on:



Then we add our new word, *watching,* to the last spot in buf:



and we repeat this process for all the words in the text. A function demonstrating this is in the file sampler.py available on our website.

Data structure

Of course, the structure of buf may not be the best one to store the information in our index. The Udacity videos have a good discussion about this topic. You will need to design a data structure for index, that facilitates searching and displaying results. If you are considering working on hacker modification 2: displaying the left context, you should design a data structure that allows for easily adding the left context.

Hacker Modification 1 (+25XP):

This hacker modification should display the search results in alphabetical order.

```
>>> generate_index('walden.txt')
>>> search('Walden')
...
Walden all over and all
Walden and White Ponds, adds,
Walden appeared like an artificial
Walden are great crystals on
Walden at long intervals serves
Walden by the turning of
Walden even. I am its
```

Hacker Modification 2 (+50XP):

This hacker modification also prints the left context. The entries should be sorted alphabetically on the right context:

```
              head or hands. I love a broad margin
                as the moon. I love better to see
          done forever. But we love better to talk
      even the wildest animals love comfort and warmth
                of the day. We love eloquence for its
  little satisfaction with and love for it, is
     much. Philanthropy is not love for one's fellow-man
```

Junior Hacker Modification 3 (+5XP):

At the end of your concordance output please print the number of entries found.

XP deductions (possible 25% of XP awarded):

XP may be deducted for poor style or poor logic. Minimum style requirements include an initial comment stating your name, the class and date. Variables should be well named. That is, code like

```
# compute percentages
```

```
annPercent = annCrayon / total
benPercent = benCrayon / total
```

is preferred over:

```
x2 = x / xy
y2 = y / xy
```

By poor logic I mean writing:

```
stars = “”
for i in range(n):
   stars += ‘*’
```

instead of

```
stars = “*” * n
```

How to submit

Please send email to the gmail account submit.o.bot with your code attached.