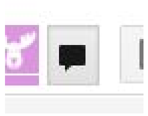
Special Topics: Big Data
Lecture 2: n-grams

Gregory S. DeLozier, Ph.D.

gdelozie@kent.edu

About this content...

- I am using Google Docs to create a presentation experience.
- I will be adding the slide content live over the course of a short while.
- You can ask questions using the chat window.
    - You can open the chat window with the chat icon. It looks like this: 
- I will answer questions as often as possible.
- You can also mail questions to my email at gdelozie@kent.edu.
- Let me know when you have gotten here and have read this page.

- This is all very experimental. I hope it works, but if not, just get the PDF at GH.

n-grams...

- n-grams are subsets of a sequence of a fixed length.
- If our string is "abcdefg", then n-grams of length 2 would be
  - "ab"
  - "bc"
  - "cd"
  - … and so on.
- n-grams of length 3 would be
  - "abc"
  - "bcd"
  - ...and so on.
- n-grams can be characters, words, whatever the sequence is made of.
- n-grams of various lengths are called unigrams, bigrams, trigrams, etc.

Text n-grams

- In a body of text, n-grams are words
  - Mary had
  - had a
  - a little
  - little lamb
- … these can be generated in 2-gram, 3-gram, etc.
- They turn out to be very useful in various ways.
  - Textual analysis
  - Speech recognition

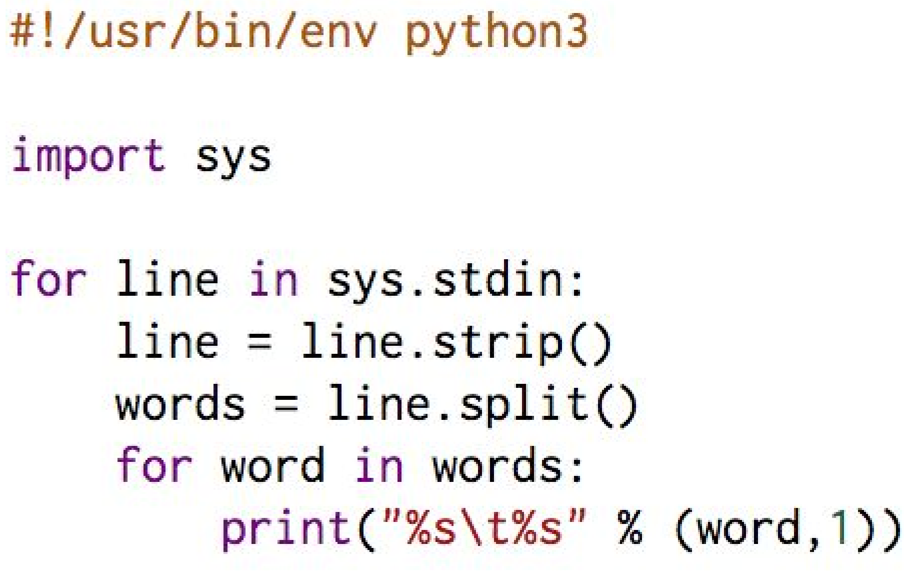
n-grams in Speech Recognition

- Suppose you make a list of, say, 4-grams from thousands of conversations
- Then you have things like ["to buy some dinner"], ["I want to go"], etc.
- Now suppose you're translating some speech.
- The machine hears
  - "want to buy some ***corn"
  - Not sure what that last word is.
    - "want to buy some popcorn", "want to buy some unicorn"?
    - Which one?
- Using the n-gram database, we find "to buy some popcorn" is much more popular than "to buy some unicorn".
- The n-gram database can tell us which words are likely to come next.

Creating n-grams with map-reduce

- So of course we're going to create our n-grams with map-reduce
    - This is a big data class, after all, right?
- The basic idea is that we have an n-gram buffer that starts out empty.
- Let's call an empty space, "BREAK" -- like a paragraph break, OK?
- For a 4-gram, it starts as
    - [EMPTY, EMPTY, EMPTY, EMPTY]
- Then for every word in the *corpus* -- latin word for the text we're using --
    - Drop the first word in the buffer
    - Add the new word
    - Generate a KV pair for that buffer by concatenating the words with a separator
    - Repeat until no more words...

Our previous map-reduce example



```
#!/usr/bin/env python3

import sys

for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print("%s\t%s" % (word,1))
```

Updating the mapper...

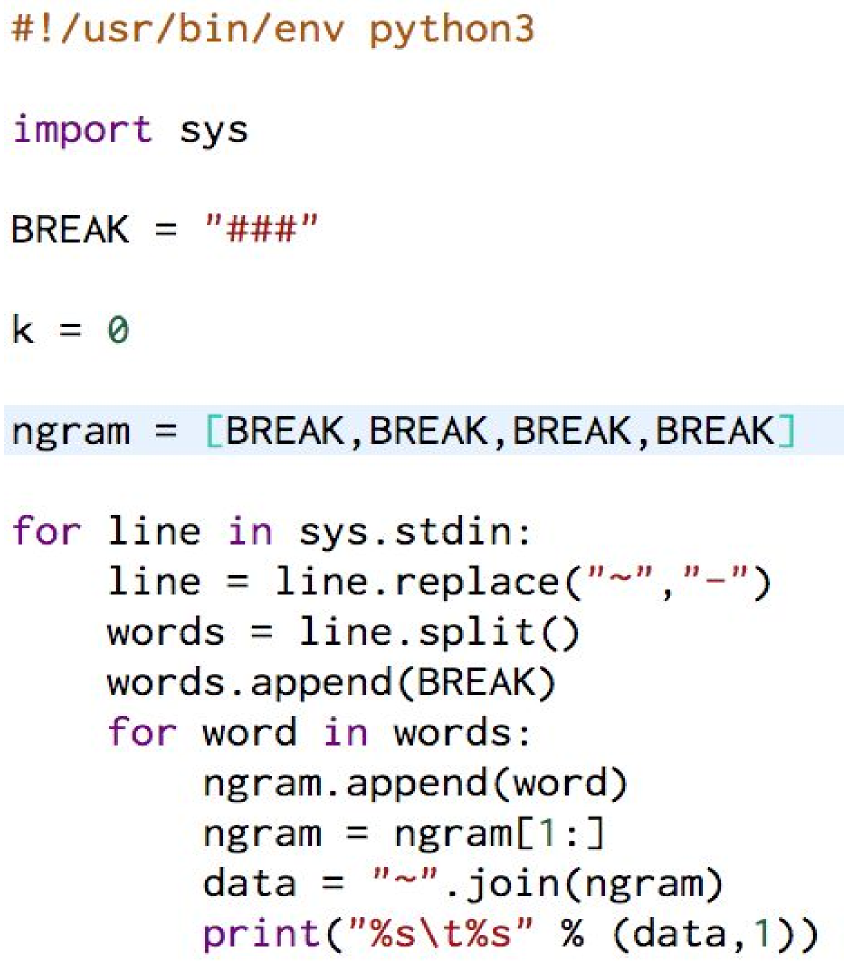
- In the first example, we put one word at a time into the KV stream.
- We are going to modify the code to put n-grams into the stream
- We will start with a symbol for BREAK
  - BREAK = "###"
- Then define the starting n-gram
  - ngram = [BREAK,BREAK,BREAK,BREAK]
  - (that is a 4-gram or a quadrigram)
- Then as we add words, we contatenate the current n-gram with "~" to make key values to stream out...

The new mapper.py

Here we get one line at a time
For each line, we split the line
For each word
- we add the word to the n-gram
- we drop the first word
- we concatenate the words with "~"
  - (note all the "~" were removed)
- we emit the n-gram with count 1

The stream then contains as many n-grams as there are words in the corpus. (In this case, our corpus is "alice.txt" but it could be anything.

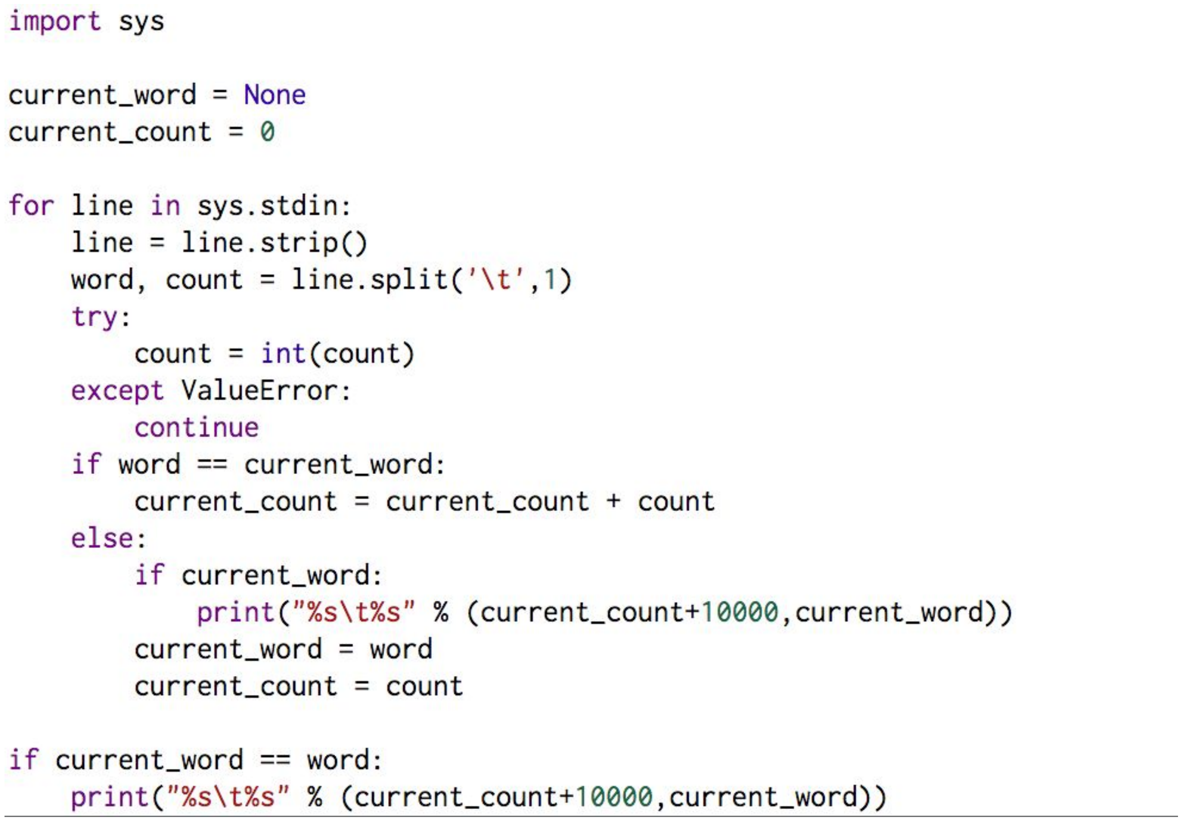
We add a BREAK at the end of each line to note the paragraph breaks in the n-grams.



```
#!/usr/bin/env python3

import sys

BREAK = "###"

k = 0

ngram = [BREAK,BREAK,BREAK,BREAK]

for line in sys.stdin:
    line = line.replace("~","-")
    words = line.split()
    words.append(BREAK)
    for word in words:
        ngram.append(word)
        ngram = ngram[1:]
        data = "~".join(ngram)
        print("%s\t%s" % (data,1))
```

What about the reducer?

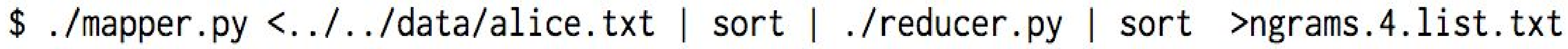
The reducer from last week works fine… It doesn't really care what it's sorting and counting. Words, n-grams, it's all the same.



```
import sys

current_word = None
current_count = 0

for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t',1)
    try:
        count = int(count)
    except ValueError:
        continue
    if word == current_word:
        current_count = current_count + count
    else:
        if current_word:
            print("%s\t%s” % (current_count+10000,current_word))
        current_word = word
        current_count = count

if current_word == word:
    print("%s\t%s” % (current_count+10000,current_word))
```

Running the map-reduce

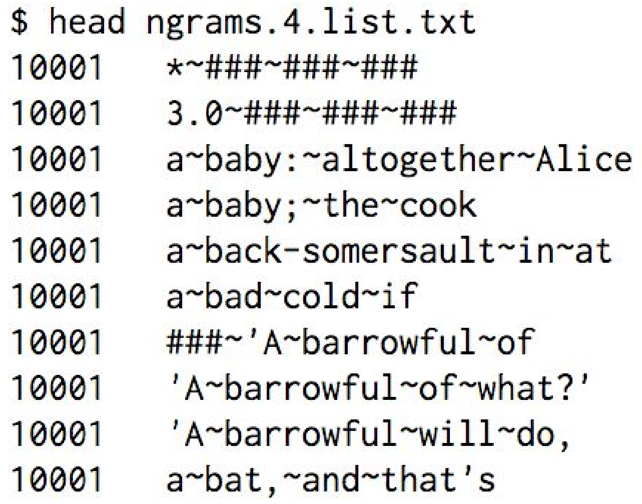
This is very easy, just like last week…



```
$ ./mapper.py <../../data/alice.txt | sort | ./reducer.py | sort  >ngrams.4.list.txt
```

This produces output like this:

There are about 20K in Alice.txt.



```
$ head ngrams.4.list.txt
10001   *~###~###~###
10001   3.0~###~###~###
10001   a~baby:~altogether~Alice
10001   a~baby;~the~cook
10001   a~back-somersault~in~at
10001   a~bad~cold~if
10001   ###~'A~barrowful~of
10001   'A~barrowful~of~what?'
10001   'A~barrowful~will~do,
10001   a~bat,~and~that's
```

Text prediction with n-grams

- We want to do text prediction.
- The basic predictor is:
  - Given n-1 words, predict the nth word, using a collection of n-grams
  - If there is more than one n-gram that matches the first n-1 words
    - Use probability to select the word based on how many times the nth word appears
- E.g. predicting the next word...
  - Here are some n-grams
    - "buy some socks", "buy some peaches", "buy some socks", "buy some socks"
  - You have so far, "buy some"
  - What's the most likely next word?
    - 75% - socks, 25% - peaches

n-gram Dictionary Structure

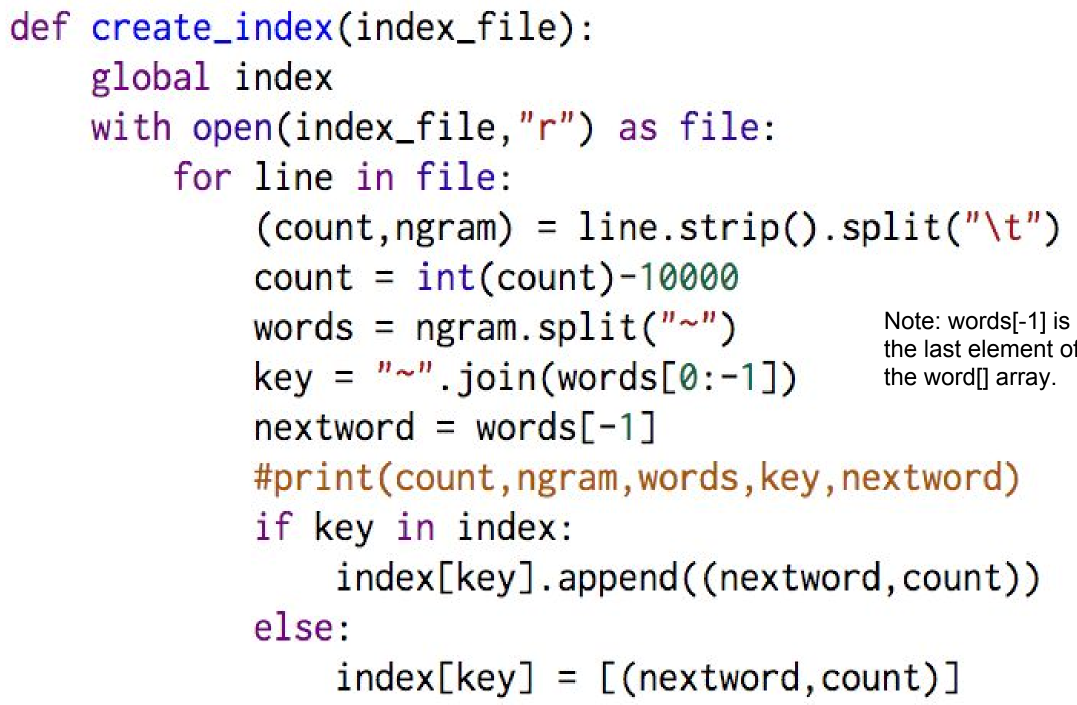
- We need a suitable data structure for this. We want to take all the n-grams with the same first n-1 words and group them together:
  - "buy-some-socks","buy-some-peaches" → "buy-some" ["socks","peaches"]
- We then create a dictionary (hash table) with the n-1 words as key
- The array of possible following words becomes the value (["socks","peaches"])
- We also have the problem of the last words having varying probabilities
- Let's add a count to the words like this:
  - "buy-some-socks" x 3,"buy-some-peaches" → "buy-some" [("socks",3),("peaches",1)]
- Doing that, we can generate a dictionary index for prediction.
- (Keep in mind that I am *really* oversimplifying this so that we can play with it)

A dictionary index generator

This is in "generator.py"

Here we read in each n-gram, and split it into count,n-gram. Then we join the first n-1 n-gram words to make a new key, and generate those tuples (nextword, count) to create the key's index value.

Note that we fix the count number by subtracting 10000.



```
def create_index(index_file):
    global index
    with open(index_file,"r") as file:
        for line in file:
            (count,ngram) = line.strip().split("\t")
            count = int(count)-10000
            words = ngram.split("~")
            key = "~".join(words[0:-1])
            nextword = words[-1]
            #print(count,ngram,words,key,nextword)
            if key in index:
                index[key].append((nextword,count))
            else:
                index[key] = [(nextword,count)]
```

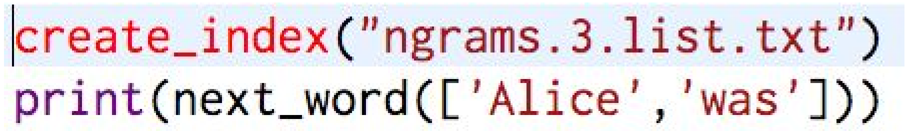
Note: words[-1] is the last element of the word[] array.

Using the n-gram dictionary...

- So we have generated this dictionary, and now we want to use it.
- We could easily store the dictionary for later, but it generates very quickly, so in today's example we will generate it on demand.
- Like this:
  - create_index("ngrams.3.list.txt")
- Notice that when I first made the n-gram mapper I started with a 4-gram buffer.
- That caused the output to be 4-grams.
- If I changed to a 3-buffer, I would get 3-grams.
- When I create the index, it splits the n-grams and determines the size.
  - (They could even be mixed -- 3-grams and 4-grams in the same file, for instance…)
  - So the dictionary works with whatever size you like.

Using the n-gram dictionary... (continued)

- So we have this dictionary.
- Given some words, we want to predict the next one.
- We want this:
    - w = next_word(["buy","some"])
- What should this return?
    - Given our dictionary, 75% socks, 25% peaches, right?
- So just for fun, we're going to try something like that.

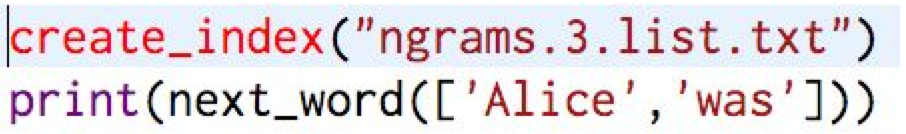


```
create_index("ngrams.3.list.txt")
print(next_word(['Alice','was']))
```

- There are a lot of possibilities, as we will see on the next slide.

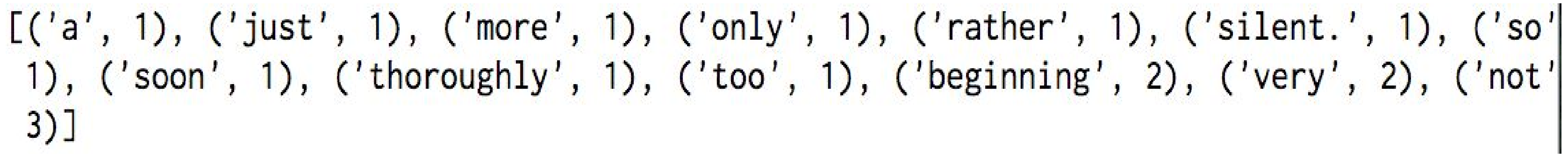
Predicting the next word...

- Our code:



```
create_index("ngrams.3.list.txt")
print(next_word(['Alice','was']))
```

- The relevant entry in the dictionary:



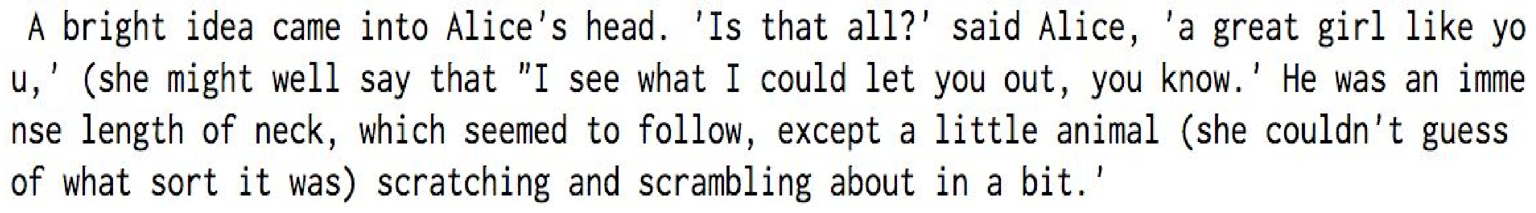
```
[('a', 1), ('just', 1), ('more', 1), ('only', 1), ('rather', 1), ('silent.', 1), ('so'
 1), ('soon', 1), ('thoroughly', 1), ('too', 1), ('beginning', 2), ('very', 2), ('not'
 3)]
```

- What to pick? First -> 'a'. Most popular -> 'not'
- If you heard in speech recognition "Alice was merry" how would you proceed?
- We will use these counts later on to generate next-word candidates.

Creating a text-generator toy...

- This next thing is a toy. These used to be called "travesty generators".
- We start with a phrase from the book, and predict the next word.
- We will add that word to the phrase.
- We know that the last n words came from the corpus, so we can look up the last n-1 words (which is a new n-gram) and predict a new next word.
- We can add that to the phrase, and repeat until we have some text.

Example text generated this way (starting with "a bright idea) in 4-gram space:



```
 A bright idea came into Alice's head. 'Is that all?' said Alice, 'a great girl like yo
u,' (she might well say that "I see what I could let you out, you know.' He was an imme
nse length of neck, which seemed to follow, except a little animal (she couldn't guess
of what sort it was) scratching and scrambling about in a bit.'
```

(Kind of reads like Alice.txt, without making a lot of overall sense…)

Probabilistic word selection...

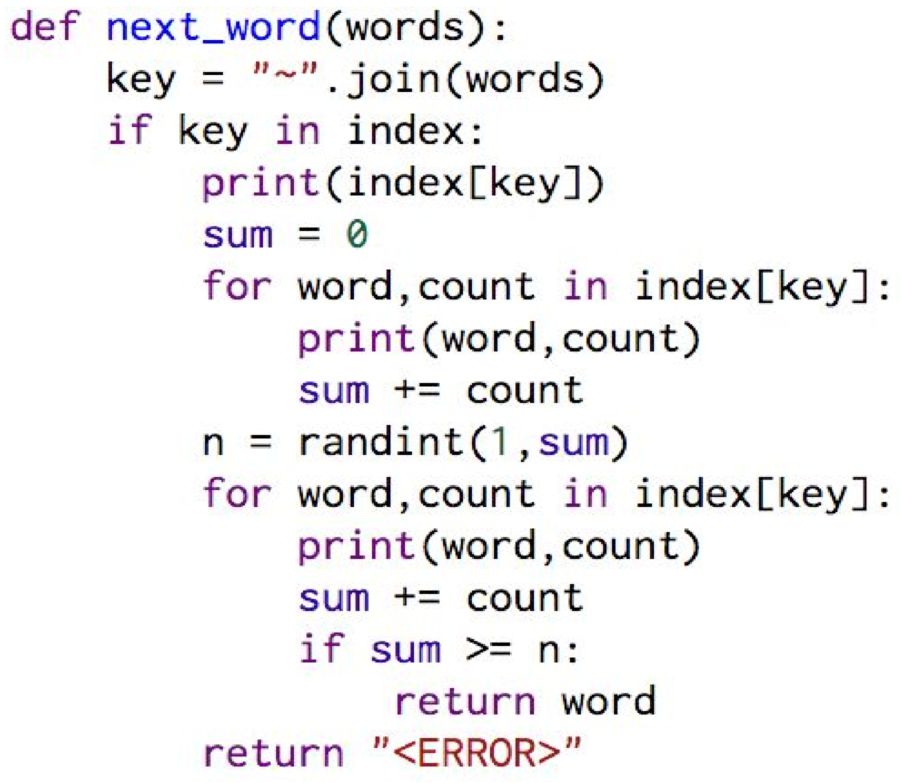
So we want to select a word from the choices according to its popularity.

First, sum the possibilities.

Then, generate a random number reflecting the range [1 .. (sum of all occurrence counts)]

Use that random number to select the chosen word, and return it.

I'll let you peruse this at a later time…



```
def next_word(words):
    key = "~".join(words)
    if key in index:
        print(index[key])
        sum = 0
        for word,count in index[key]:
            print(word,count)
            sum += count
        n = randint(1,sum)
        for word,count in index[key]:
            print(word,count)
            sum += count
            if sum >= n:
                return word
        return "<ERROR>"
```

Useful code:
  from random import randint

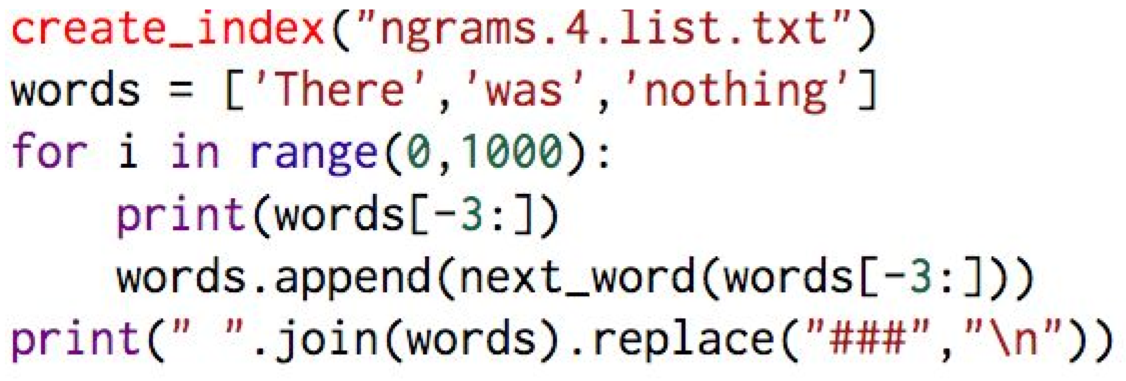
Running generator.py...

This is the main part of the program, currently.

It generates text pretty well, and you can play with it. It's checked into GH (or will be shortly) under /codes/ngram

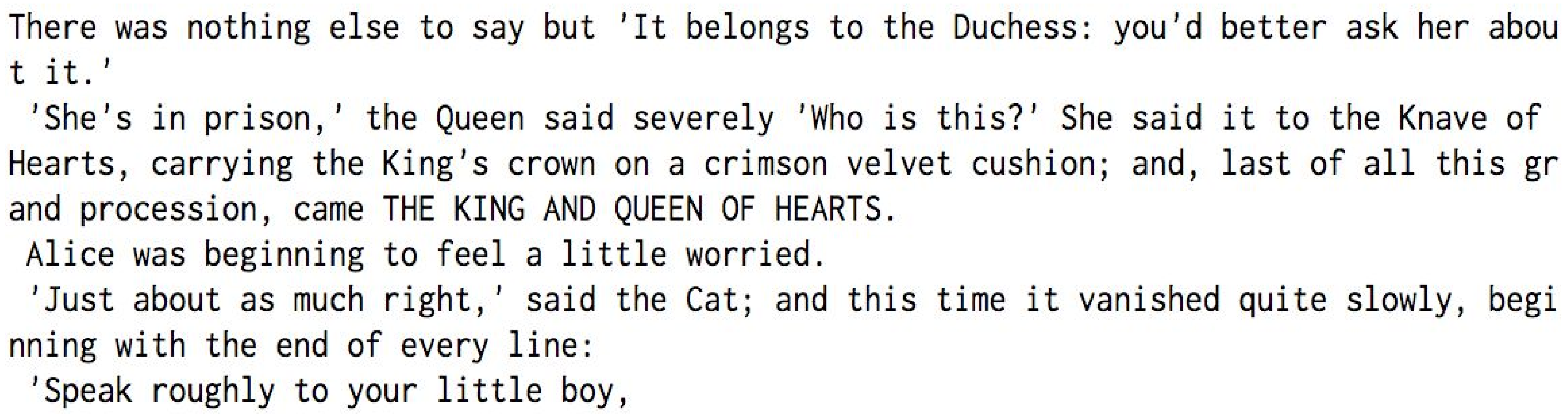
It tends, however, to run in circles sometimes. (Why?)

Try it!



```python
create_index("ngrams.4.list.txt")
words = ['There','was','nothing']
for i in range(0,1000):
    print(words[-3:])
    words.append(next_word(words[-3:]))
print(" ".join(words).replace("###","\n"))
```

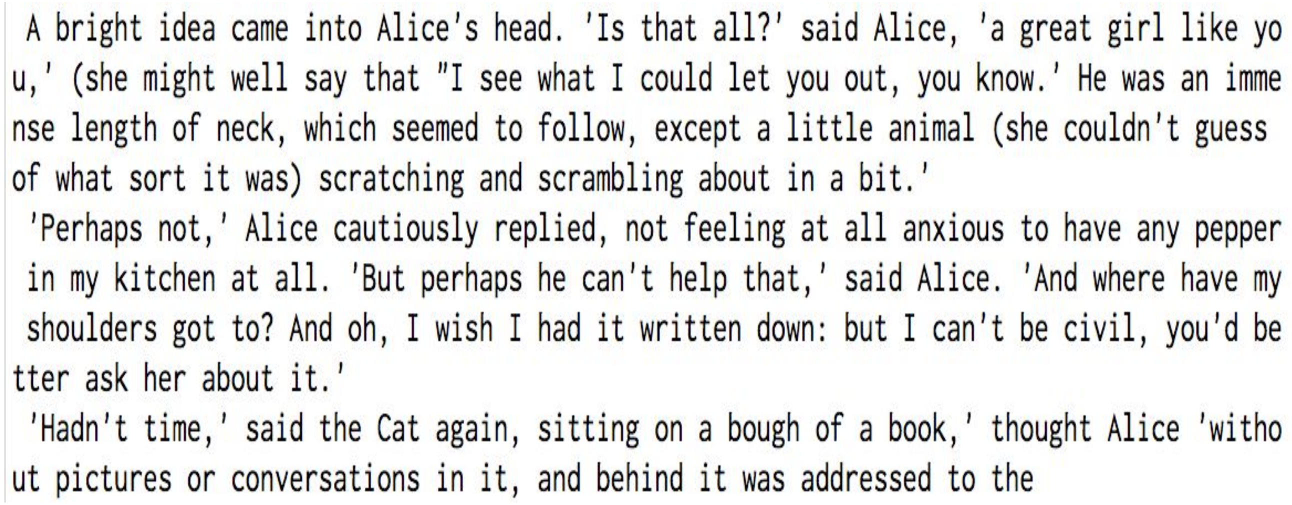
Some generated "alice text"



```
There was nothing else to say but 'It belongs to the Duchess: you'd better ask her abou
t it.'
 'She's in prison,' the Queen said severely 'Who is this?' She said it to the Knave of
Hearts, carrying the King's crown on a crimson velvet cushion; and, last of all this gr
and procession, came THE KING AND QUEEN OF HEARTS.
 Alice was beginning to feel a little worried.
 'Just about as much right,' said the Cat; and this time it vanished quite slowly, begi
nning with the end of every line:
 'Speak roughly to your little boy,
```

Comments…

- The longer the n-grams in the dictionary, the more "normal" the output is.
- Generating alice-text from 3-grams is more random.
    - Switch to a 3-gram dictionary
    - Use words[-2:] instead of words[-3:] in the main part of generator.py
- Example:



```
 A bright idea came into Alice's head. 'Is that all?' said Alice, 'a great girl like yo
u,' (she might well say that "I see what I could let you out, you know.' He was an imme
nse length of neck, which seemed to follow, except a little animal (she couldn't guess
of what sort it was) scratching and scrambling about in a bit.'
 'Perhaps not,' Alice cautiously replied, not feeling at all anxious to have any pepper
 in my kitchen at all. 'But perhaps he can't help that,' said Alice. 'And where have my
 shoulders got to? And oh, I wish I had it written down: but I can't be civil, you'd be
tter ask her about it.'
 'Hadn't time,' said the Cat again, sitting on a bough of a book,' thought Alice 'witho
ut pictures or conversations in it, and behind it was addressed to the
```

Other remarks

- Playing with n-grams is pretty entertaining
- Try using some other corpus. Departmental email makes a good source.
- So does Shakespeare.
- n-gram methods are used for lots of other applications
- You can read a lot about it here:
  - https://en.wikipedia.org/wiki/N-gram
- And more about word prediction here:
  - http://www.cs.columbia.edu/~kathy/NLP/ClassSlides/Class3-ngrams09/ngrams.pdf
  - The math in here is pretty involved, but we're not a machine learning class, so it's optional. :-)

Homework

- You knew it was coming. :-)
- Get the code running on your environment.
- Get some text from somewhere else and see what happens
- Try various lengths of n-grams
- Try to see if playing with punctuation, capitalization, or other cleanups helps
  - (Helps means generate more believable random text)
- Is there an improvement to the n-gram prediction you can make?
  - Modify probabilities after using a word?
  - Discount certain word
  - Ignore words in n-grams (like definite articles, etc) and if you do, how will you inject them?
- Any other interesting ideas would be nice
- Turn in work on Blackboard, as usual.

Conclusion…

- Back in the room next week.
- Probably I will be having online office hours Sunday late night.
  - (If so, I will announce via Blackboard email…)
- I hope you all have a great weekend…

- Almost forgot -- HW1 and HW2 will be due next Wednesday!
- (The network here is terrible -- the lag makes my typing look worse than it is)
- (Which is pretty bad already.)
- Bye! :-)