```
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\Jane Eyre\Documents\ling1330\HW2_bible_austen_bigrams.KEY.py

### Ran my HW2 script in order to create Bible and Austen data objects
### Script output clipped.

>>> dir()
['__annotations__', '__builtins__', '__doc__', '__file__', '__loader__', '__name__', '__package__',
'__spec__', 'a_bigramcfd', 'a_bigramfd', 'a_bigrams', 'a_etxt', 'a_ptxt', 'a_stxt', 'a_tokfd',
'a_toks', 'a_txt', 'b_bigramcfd', 'b_bigramfd', 'b_bigrams', 'b_sograms', 'b_tokfd', 'b_toks',
'b_txt', 'c', 'count', 'f', 'gram', 'nltk', 'observ_e1', 'observ_e2', 'observ_e3', 'observ_e4',
'observ_e5', 'observ_e6', 'pickle', 'w', 'w2']
>>> sent = 'she was not afraid .'.split()
>>> sent
['she', 'was', 'not', 'afraid', '.']
>>> a_tokfd['she']
5095
>>> b_tokfd['she']
982
>>> b_tokfd.freq('she')
0.0010369500574440764
>>> b_tokfd.freq('was')
0.004775038859228222
>>> b_tokfd.freq('not')
0.007159390416976414
>>> b_tokfd.freq('afraid')
0.0002037997567074407
>>> b_tokfd.freq('.')
0.027668192876934512
>>> sent
['she', 'was', 'not', 'afraid', '.']
>>> [w for w in sent]
['she', 'was', 'not', 'afraid', '.']
>>> [b_tokfd.freq(w) for w in sent]
[0.0010369500574440764, 0.004775038859228222, 0.007159390416976414, 0.0002037997567074407,
0.027668192876934512]

### Using numpy to easily multiply a list of probabilities (= "product")
### Unigram-based probability estimation for "she was not afraid ."

>>> import numpy
>>> numpy.prod([1,2,3,4,5])
120
>>> probs = [b_tokfd.freq(w) for w in sent]    # Bible
>>> probs
[0.0010369500574440764, 0.004775038859228222, 0.007159390416976414, 0.0002037997567074407,
0.027668192876934512]
>>> numpy.prod(probs)
1.9989192577000988e-13
>>> [a_tokfd.freq(w) for w in sent]            # Austen
[0.0118149037067956, 0.012972045404477836, 0.010653124166637681, 0.0002388488749753616,
0.031298479947128595]
>>> probs = [a_tokfd.freq(w) for w in sent]
>>> numpy.prod(probs)
1.2205683498160203e-11


### Bigram-based probability estimation for "she was not afraid ."

>>> sent
['she', 'was', 'not', 'afraid', '.']
>>> b_bigramcfd['she']
```

```
FreqDist({'said': 80, 'shall': 65, 'was': 63, 'is': 63, 'had': 58, 'hath': 51, 'bare': 30, 'came':
28, 'went': 26, 'be': 18, ...})
>>> b_bigramcfd['she']['was']
63
>>> b_bigramcfd['she'].freq('was')
0.06415478615071284
>>> b_tokfd.freq('she')      # first word: substitute with unigram probability
0.0010369500574440764
>>> [b_tokfd.freq('she'), b_bigramcfd['she'].freq('was')]
[0.0010369500574440764, 0.06415478615071284]
>>> [b_tokfd.freq('she'), b_bigramcfd['she'].freq('was'), b_bigramcfd['was'].freq('not'),
b_bigramcfd['not'].freq('afraid'), b_bigramcfd['afraid'].freq('.')]
[0.0010369500574440764, 0.06415478615071284, 0.033392304290137106, 0.005162241887905605,
0.16580310880829016]
>>> probs = [b_tokfd.freq('she'), b_bigramcfd['she'].freq('was'), b_bigramcfd['was'].freq('not'),
b_bigramcfd['not'].freq('afraid'), b_bigramcfd['afraid'].freq('.')]
>>> numpy.prod(probs)
1.9013598142591665e-09
>>> probs = [a_tokfd.freq('she'), a_bigramcfd['she'].freq('was'), a_bigramcfd['was'].freq('not'),
a_bigramcfd['not'].freq('afraid'), a_bigramcfd['afraid'].freq('.')]
>>> probs
[0.0118149037067956, 0.13758586849852797, 0.0650697175545227, 0.00108837614279495,
0.02912621359223301]
>>> numpy.prod(probs)
3.353095950422491e-09


### Processing count_1w.txt from Norvig
### Confirming the downloaded file location first, so I will know how to refer to it
### Read in the content as a list of lines

>>> import os
>>> os.getcwd()
'C:\\Users\\Jane Eyre\\Documents\\ling1330'
>>> os.listdir()
['austen_bigramcfd.pkl', 'bible_bigramcfd.pkl', 'count_1w.txt', 'enable1.txt',
'fox_in_sox.TEMPLATE.py', 'gettysburg_address.txt', 'gutenberg', 'hello.py',
'HW2_bible_austen_bigrams.KEY.py', 'idle_session.txt', 'pal_loop.py', 'pal_naive.py', 'words.pkl']

>>> f = open('count_1w.txt')
>>> lines = f.readlines()
>>> f.close()

>>> lines[0]
'the\t23135851162\n'
>>> lines[1]
'of\t13151942776\n'
>>> lines[-1]
'golgw\t12711\n'
>>> len(lines)
333333
>>> lines[1]
'of\t13151942776\n'
>>> lines[1].split()
['of', '13151942776']
>>> (lines[1].split())       # didn't work
['of', '13151942776']
>>> tuple(lines[1].split())  # this is the way
('of', '13151942776')


### Experiment first with a "mini" version

>>> mini = lines[:5]
>>> mini
```

```
['the\t23135851162\n', 'of\t13151942776\n', 'and\t12997637966\n', 'to\t12136980858\n',
'a\t9081174698\n']
>>> for line in mini:
...     print(line)
...
the     23135851162

of      13151942776

and     12997637966

to      12136980858

a       9081174698

>>> [line for line in mini]
['the\t23135851162\n', 'of\t13151942776\n', 'and\t12997637966\n', 'to\t12136980858\n',
'a\t9081174698\n']
>>> [line.split() for line in mini]
[['the', '23135851162'], ['of', '13151942776'], ['and', '12997637966'], ['to', '12136980858'], ['a',
'9081174698']]
>>> [tuple(line.split()) for line in mini]
[('the', '23135851162'), ('of', '13151942776'), ('and', '12997637966'), ('to', '12136980858'), ('a',
'9081174698')]

### List comprehension works so far, but then it's not clear what to do about the number,
###   which is in the string data format
### Let's fall back onto good-old for loop, so we can index the count and integer-fy it:

>>> for line in mini:
...    (word, count) = tuple(line.split())
...    print(word, int(count))       # turn count into integer
...
the 23135851162
of 13151942776
and 12997637966
to 12136980858
a 9081174698

>>> mini_rank = []                  # initialize an empty list
>>> for line in mini:
...     (word, count) = tuple(line.split())
...     mini_rank.append((word, int(count)))   # append while for-looping
...
>>> mini_rank
[('the', 23135851162), ('of', 13151942776), ('and', 12997637966), ('to', 12136980858), ('a',
9081174698)]
>>> mini_rank[0]
('the', 23135851162)
>>> mini_rank[1]
('of', 13151942776)

### Mini version looks right, now ready to build a full ranked list

>>> goog1w_rank = []
>>> for line in lines:
...     (word, count) = tuple(line.split())
...     goog1w_rank.append((word, int(count)))
...
>>> goog1w_rank[:10]
[('the', 23135851162), ('of', 13151942776), ('and', 12997637966), ('to', 12136980858), ('a',
9081174698), ('in', 8469404971), ('for', 5933321709), ('is', 4705743816), ('on', 3750423199),
('that', 3400031103)]
>>> goog1w_rank[-10:]
[('googllr', 12711), ('googlal', 12711), ('googgoo', 12711), ('googgol', 12711), ('goofel', 12711),
```

```
('gooek', 12711), ('gooddg', 12711), ('gooblle', 12711), ('gollgo', 12711), ('golgw', 12711)]
>>> goog1w_rank[999]    # huh?
('entry', 80717798)
>>> goog1w_rank[9999]   # huuuh?
('poison', 5056083)
>>> len(goog1w_rank)
333333


### An ordered list of (word, count) pairs complete.
### Now to build a FreqDist version.
### First experimenting with the mini version of goog1w_rank:

>>> mini_fd = nltk.FreqDist()       # initialize an empty FreqDist object
>>> mini_fd
FreqDist({})
>>> mini_fd['the'] = 13505      # populate FD
>>> mini_fd
FreqDist({'the': 13505})

>>> mini_fd = nltk.FreqDist()
>>> mini_rank = goog1w_rank[:5]
>>> mini_rank
[('the', 23135851162), ('of', 13151942776), ('and', 12997637966), ('to', 12136980858), ('a',
9081174698)]
>>> for (word, count) in mini_rank:
...     print(word, count)
...
the 23135851162
of 13151942776
and 12997637966
to 12136980858
a 9081174698
>>> for (word, count) in mini_rank:
...     mini_fd[word] = count
...
>>> mini_fd
FreqDist({'the': 23135851162, 'of': 13151942776, 'and': 12997637966, 'to': 12136980858, 'a':
9081174698})
>>> mini_fd['the']
23135851162
>>> mini_fd['a']
9081174698


### Now ready to build a full FreqDist

>>> goog1w_fd = nltk.FreqDist()
>>> for (word, count) in goog1w_rank:
...     goog1w_fd[word] = count
...
...
>>> goog1w_fd.most_common(10)
[('the', 23135851162), ('of', 13151942776), ('and', 12997637966), ('to', 12136980858), ('a',
9081174698), ('in', 8469404971), ('for', 5933321709), ('is', 4705743816), ('on', 3750423199),
('that', 3400031103)]
>>> goog1w_fd['the']
23135851162
>>> goog1w_fd['linguist']
557433
>>> goog1w_fd['teacher']
48002944
>>> goog1w_fd['platypus']      # sigh, more than "linguist"
565585
>>> goog1w_fd['pittsburgh']
```

```
19654781
>>> goog1w_fd['cleveland']
19041185
>>> goog1w_fd['philadelphia']
30179898


### goog1w_fd is good for looking up words,
### goog1w_rank is good for looking up based on rank

>>> goog1w_rank[0]
('the', 23135851162)
>>> goog1w_rank[9]
('that', 3400031103)
>>> goog1w_rank[99]
('find', 502043038)
>>> goog1w_rank[999]
('entry', 80717798)
>>> goog1w_rank[9999]
('poison', 5056083)


### Pickle the data, so we can re-use them later

>>> import pickle
>>> f = open('goog1w_rank.pkl', 'wb')
>>> pickle.dump(goog1w_rank, f, -1)
>>> f.close()
>>> f2 = open('goog1w_fd.pkl', 'wb')
>>> pickle.dump(goog1w_fd, f2, -1)
>>> f2.close()
```