```
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', '.']
>>> b_tokfd['she']
982
>>> b_tokfd.freq('she')
0.001037164716965987
>>> b_tokfd.freq('was')
0.004776027342281256
>>> b_tokfd.freq('not')
0.007160872485773311
>>> b_tokfd.freq('afraid')
0.00020384194539148216
>>> b_tokfd.freq('.')
0.02767392048263013
>>> sent
['she', 'was', 'not', 'afraid', '.']
>>> [w for w in sent]
['she', 'was', 'not', 'afraid', '.']
>>> [b_tokfd.freq(w) for w in sent]
[0.001037164716965987, 0.004776027342281256, 0.007160872485773311, 0.00020384194539148216,
0.02767392048263013]


### 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
>>> [b_tokfd.freq(w) for w in sent]
[0.001037164716965987, 0.004776027342281256, 0.007160872485773311, 0.00020384194539148216,
0.02767392048263013]
>>> probs = [b_tokfd.freq(w) for w in sent]
>>> numpy.prod(probs)
2.0009891005865551e-13
>>> probs = [a_tokfd.freq(w) for w in sent]
>>> probs
[0.011819179315160331, 0.012976739762317347, 0.010656979347173023, 0.00023893532275986537,
0.03130980632320294]
>>> numpy.prod(probs)
1.2227784612297631e-11


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

>>> sent
['she', 'was', 'not', 'afraid', '.']
>>> a_bigramcfd['she']['was']
701
```

```
>>> a_bigramcfd['she'].freq('was')
0.13758586849852797
>>> [a_tokfd.freq('she'), a_bigramcfd['she'].freq('was'), a_bigramcfd['was'].freq('not'),
a_bigramcfd['not'].freq('afraid'), a_bigramcfd['afraid'].freq('.') ]
[0.011819124480086108, 0.13758586849852797, 0.0650697175545227, 0.00108837614279495,
0.02912621359223301]
>>> probs = [a_tokfd.freq('she'), a_bigramcfd['she'].freq('was'), a_bigramcfd['was'].freq('not'),
a_bigramcfd['not'].freq('afraid'), a_bigramcfd['afraid'].freq('.') ]
>>> numpy.prod(probs)
3.3542938152700833e-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', 'Ex4_emma_enable.py',
'gettysburg_address.txt', 'gift-of-magi.txt', 'gift_shell_explore.pdf', 'gutenberg', 'hello.py',
'HW2_bible_austen_bigrams.KEY.py', 'nltk_practice.txt', 'process_gift.py', 'process_gift_out.txt',
'words.pkl']
>>> f = open('count_1w.txt')
>>> lines = f.readlines()
>>> f.close()
>>> lines[0]
'the\t23135851162\n'
>>> lines[1]
'of\t13151942776\n'
>>> lines[2]
'and\t12997637966\n'
>>> len(lines)
333333
>>> lines[-1]
'golgw\t12711\n'


### 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

>>> for line in mini:
...     print(line.split())
...
['the', '23135851162']
['of', '13151942776']
['and', '12997637966']
['to', '12136980858']
```

```
['a', '9081174698']
>>> for line in mini:
...     print(tuple(line.split()))
...
('the', '23135851162')
('of', '13151942776')
('and', '12997637966')
('to', '12136980858')
('a', '9081174698')
>>> mini_rank = []
>>> for line in mini:
...     tu = tuple(line.split())
...     mini_rank.append(tu)
...
>>> mini_rank
[('the', '23135851162'), ('of', '13151942776'), ('and', '12997637966'), ('to', '12136980858'), ('a',
'9081174698')]
>>> mini_rank = []
>>> for line in mini:
...     (word, count) = line.split()
...     mini_rank.append((word, int(count)))
...
>>> mini_rank
[('the', 23135851162), ('of', 13151942776), ('and', 12997637966), ('to', 12136980858), ('a',
9081174698)]


### Now ready to build a full ranked list

>>> goog1w_rank = []
>>> for line in lines:
...     (word, count) = line.split()
...     goog1w_rank.append((word, int(count)))
...
>>> goog1w_rank[:20]
[('the', 23135851162), ('of', 13151942776), ('and', 12997637966), ('to', 12136980858), ('a',
9081174698), ('in', 8469404971), ('for', 5933321709), ('is', 4705743816), ('on', 3750423199),
('that', 3400031103), ('by', 3350048871), ('this', 3228469771), ('with', 3183110675), ('i',
3086225277), ('you', 2996181025), ('it', 2813163874), ('not', 2633487141), ('or', 2590739907), ('be',
2398724162), ('are', 2393614870)]
>>> goog1w_rank[-20:]
[('goooglo', 12711), ('gooogla', 12711), ('gooogd', 12711), ('gooofa', 12711), ('goooao', 12711),
('goollo', 12711), ('goolld', 12711), ('goolh', 12711), ('goolgee', 12711), ('googook', 12711),
('googllr', 12711), ('googlal', 12711), ('googgoo', 12711), ('googgol', 12711), ('goofel', 12711),
('gooek', 12711), ('gooddg', 12711), ('gooblle', 12711), ('gollgo', 12711), ('golgw', 12711)]
>>> 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_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['the']
23135851162
>>> goog1w_fd['platypus']
565585
>>> goog1w_fd['penguin']
5835109
>>> goog1w_fd['pittsburgh']
19654781
>>> goog1w_fd['cleveland']
19041185
>>> goog1w_fd['philadelphia']
30179898
>>> goog1w_fd['philly']
2102437


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