Handout 4: Wordlists and Frequencies

Texts and wordlists

1. A text behaves just like a list.

```
>>> text1[:6]
['[', 'Moby', 'Dick', 'by', 'Herman', 'Melville']
>>> text1[1]
'Moby'
>>> len(text1)
260819
```

2. But it has some additional **methods**.

```
>>> text1.name
'Moby Dick by Herman Melville 1851'
>>> text1.count('Ishmael')
19
>>> text1.concordance('monstrous')
```

3. You can create your own text from a list.

```
>>> mytext = Text(['Can', 'a', 'wood', 'chuck', 'chuck', 'wood', '?'])
>>> len(mytext)
7
>>> mytext.count('wood')
2
```

4. *Review.*

a. To eliminate duplicates, convert to a set: `vocab = set(text1)`

b. Elements of the vocabulary are **types**, elements of the text are **tokens**

c. Alphabetizing:

```
>>> v1 = set(mytext)
>>> sorted(v1)
['?', 'Can', 'a', 'chuck', 'wood']
```

d. Downcasing, eliminating punctuation: `set(w.lower() for w in text1 if w.isalpha())`

Sorting

5. Use the function `sorted`

a. Works with any iterable; produces a list

```
>>> words = {'hi', 'bobby', 'bye'}
>>> sorted(words)
['bobby', 'bye', 'hi']
>>> sorted([5, 0, 3])
[0, 3, 5]
```

   b. What if you want biggest to smallest?

```
>>> sorted([5, 0, 3], reverse=True)
[5, 3, 0]
```

   c. "`reverse=True`" is a **keyword argument**

6. What if you want to sort words by length?

   a. Keyword argument "`key`"

```
>>> sorted(words, key=len)
['hi', 'bye', 'bobby']
```

   b. Notice: the value of `len` is a *function.* The function is called on each word, and the words are sorted according to the values:

Word:	'hi'	'bobby'	'bye'
Len:	2	5	3

   c. Combining `key` and `reverse`:

```
>>> sorted(words, key=len, reverse=True)
['bobby', 'bye', 'hi']
```

7. Stability

   a. Sorting is guaranteed to be **stable**: words that tie are kept in their *original order*

   b. So—if we want tying words to be sorted alphabetically, do an alphabetic sort first:

```
>>> tmp = sorted({'hi', 'cat', 'bye'})
>>> tmp
['bye', 'cat', 'hi']
>>> sorted(tmp, key=len)
['hi', 'bye', 'cat']
```

8. *Exercises.*

   a. Sort `text1` … `text9` by their length (number of tokens).

   b. Sort them by their vocabulary size.

9. Note: one can create an "anonymous" function with `lambda`:

```
>>> sorted(x, key=lambda text: text.count('you')/len(text))
```

A little data exploration

10. Lexical diversity

  a. Contrary to the book, let us define the lexical diversity of a text to be the average number of types per 1000 tokens

  b. *Moby Dick* has 19,317 types per 260,819 tokens

```
>>> 19317 / 260819
0.07406285585022564
>>> _ * 1000
74.06285585022563
```

  c. General function

```
>>> def diversity (text):
...     k = len(set(text))
...     n = len(text)
...     return 1000 * k / n
...
>>> diversity(text1)
74.06285585022563
```

11. Note: `k` and `n` are **local variables**; created by assignment. Completely contained inside the function.

```
>>> k = ’hi’
>>> diversity(text1)
74.06285585022563
>>> k
’hi’
```

12. Diversity of personals

  a. Consider the following. Is this surprising?

```
>>> diversity(text8)
227.65564002465584
>>> text8
<Text: Personals Corpus>
```

  b. What does the personals corpus look like?

```
>>> ’ ’.join(text8[:8])
’25 SEXY MALE , seeks attrac older single’
```

  c. Seems like a pretty limited vocabulary. Why is the diversity so high?

  d. Is it any different somewhere in the middle?

```
>>> ’ ’.join(text8[2000:2010])
’rship . WLTM sincere , caring Lady to share life’
```

13. How else do the texts differ?

a. In length: `len(text1)` is 260,819; `len(text8)` is 4867.

b. What if we shorten *Moby Dick*? Aha!

```
>>> diversity(text1[:4867])
335.319498664475
```

14. *Homework.* Redefine `diversity()` to count the number of types in the first thousand words of the text. Which text is most diverse now? Sort the texts from most diverse to least diverse.

Frequency dists

15. A **dict** is a table mapping keys to values

```
>>> tab = dict()
>>> tab['moby'] = len(text1)
>>> tab['moby']
260819
```

16. A frequency distribution is a specialized dict that maps items to counts.

```
>>> tokens = list('abbdabdbbd')
>>> fd = FreqDist(tokens)
>>> fd.tabulate()
   b    d    a
   5    3    2
```

17. Access

a. Original items (tokens) are **keys**; access counts by key

```
>>> fd['a']
2
```

b. Relative frequency (probability):

```
>>> fd.freq('a')
0.2
>>> fd['a'] / fd.N()
0.2
```

18. Methods

a. How many types? How many tokens?

```
>>> len(fd)
3
>>> fd.N()
10
```

b. Dist behaves like list of keys (random order)

```
>>> for k in fd:
...     print(k, fd[k])
...
d 3
b 5
a 2
```

c. Sorting by frequency

```
>>> fd.most_common()
[('b', 5), ('d', 3), ('a', 2)]
>>> fd.most_common(1)
[('b', 5)]
```

19. `('b', 5)` is a **tuple**.

a. Tuples are just like lists, except they cannot be modified.

```
>>> x = ('b', 5)
>>> len(x)
2
>>> x[0]
'b'
```

b. A useful trick

```
>>> (w, ct) = x
>>> w
'b'
>>> ct
5
```

20. *Exercises.*

a. How do we find the five most-frequent tokens in *Moby Dick*?

b. What if we want just the words, without counts?

c. What if we want only the real words, not punctuation?

d. What are the commonest word lengths in *Moby Dick*?

21. Zipf's Law

a. Let w_1 be the most-frequent word, w_2 the second most-frequent, etc. Zipf's law states:

$$f(w_r) = K/r$$

b. It implies that the most-frequent words account for most of the text

c. It also implies that the commonest rate of occurrence is 1.

d. *Homework.* Determine whether these statements are true for our texts

22. Cumulative frequency: add up the frequencies of the first n elements

a. Aggregation function `sum`

```
>>> freqs = [.4, .3, .2, .1]
>>> sum(freqs)
0.9999999999999999
```

b. How do we get the sum of the first n freqs?

23. Joint distributions.

a. A distribution over pairs of items is a **joint** distribution.

b. Pairs of adjacent words are called **bigrams**. For our corpus:

	a	b	d	
a		2/9		2/9
b		2/9	3/9	5/9
d	1/9	1/9		2/9
	1/9	5/9	3/9	

24. Marginal probability.

a. The distribution over the individual items is called the **marginal** distribution.

b. In the case of bigrams, the marginal distribution is called the **unigram distribution**.

c. There are actually two unigram distributions. Explain. How could we fix that?

25. Bigrams

```
>>> bigrams(tokens)
<generator object bigrams at 0x1075e0990>
>>> list(bigrams(tokens))
[('a', 'b'), ('b', 'b'), ('b', 'd'), ('d', 'a'), ...]
>>> bd = FreqDist(bigrams(tokens))
>>> bd.freq(('a','b'))
0.2222222222222222
>>> bd.most_common(3)
[(('b', 'd'), 3), (('b', 'b'), 2), (('a', 'b'), 2)]
```

26. *Exercises.*

a. What are the most common pairs of words in *Moby Dick*?

```
[(',', 'and'), ('of', 'the'), ("'", 's'), ('in', 'the')]
```

(How did I get that result?)

b. What are the most common words preceding *whale* in *Moby Dick*?

```
[(',', 18713), ('the', 13721), ('.', 6861), ('of', 6536)]
```

c. How do we get the *most significant* pairs? → collocations