Grabs all the default texts in nltk.book

In [ ]:
```
from nltk.book import *
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

Code Comments

Stores text1 as tokens in the var text1Tokens then diplays the first 20 tokens in the list

Qustion Answers

The tokens method will return the tokens of a list.

The tokens method is highly aggressive in what it considers a token.

In [ ]:
```
text1Tokens = text1.tokens
text1Tokens[:20]
```

Code Comments

goes through text1 and displays the first 5 phrases that contain the word "sea" and ~40 characters around that word.

In [ ]:
```
text1.concordance("sea",80,5)
```

```
Displaying 5 of 455 matches:
 shall slay the dragon that is in the sea ." -- ISAIAH " And what thing soever
 S PLUTARCH ' S MORALS . " The Indian Sea breedeth the most and the biggest fis
cely had we proceeded two days on the sea , when about sunrise a great many Wha
many Whales and other monsters of the sea , appeared . Among the former , one w
 waves on all sides , and beating the sea before him into a foam ." -- TOOKE '
```

Code Commnets

create a list of a, b, and c
create a list of tokens from text1

print all occurences of 'sea' in text1Tokens
print the number of words in text1Tokens

print all occurence of 'a' in list
print the number of items in list

Question Answers

the python count method merely counts all the items in a list, while the nltk count method requires something to count and only returns the amount of what was counted

In [ ]:
```
list = ['a', 'b', 'c']
text1Tokens = text1.tokens

print(text1Tokens.count('sea'))
print(len(text1Tokens))

print(list.count('a'))
print(len(list))
```

```
433
260819
1
3
```

Code Comments

Raw_text taken from here
then convert that raw_text to word tokens
then print the first 10 tokens in that list

In [ ]:
```
import nltk
raw_text = "The quick brown fox jumps over the lazy dog. Sphinx of black quartz, judge
tokens = nltk.word_tokenize(raw_text)

tokens[:10]
```

Out[ ]:
```
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog', '.']
```

Code Comments

raw text same as above
then convert raw text to sentance tokens and then assign them to the variable tokens
then print the list of those tokens

In [ ]:
```
import nltk
raw_text = "The quick brown fox jumps over the lazy dog. Sphinx of black quartz, judge
tokens = nltk.sent_tokenize(raw_text)

print(tokens)
```

```
['The quick brown fox jumps over the lazy dog.', 'Sphinx of black quartz, judge my vo
w!', 'Waltz job vexed quick frog nymphs.', 'Go, lazy fat vizen; be shrewd, jump quic
k.', 'Five quacking zephyrs jolt my wax bed.']
```

Code Comments

raw text same as above

create a PorterStemmer object

split raw text into word tokens

for each word in rawTokens chop it down to its stem(well as close as an algo can get to stem)

In [ ]:
```
import nltk
raw_text = "The quick brown fox jumps over the lazy dog. Sphinx of black quartz, judge
stemmer = nltk.PorterStemmer()
rawTokens = nltk.word_tokenize(raw_text)
[stemmer.stem(t) for t in rawTokens]
```

Out[ ]:
```
['the',
 'quick',
 'brown',
 'fox',
 'jump',
 'over',
 'the',
 'lazi',
 'dog',
 '.',
 'sphinx',
 'of',
 'black',
 'quartz',
 ',',
 'judg',
 'my',
 'vow',
 '!',
 'waltz',
 'job',
 'vex',
 'quick',
 'frog',
 'nymph',
 '.',
 'go',
 ',',
 'lazi',
 'fat',
 'vizen',
 ';',
 'be',
 'shrewd',
 ',',
 'jump',
 'quick',
 '.',
 'five',
 'quack',
 'zephyr',
 'jolt',
 'my',
 'wax',
 'bed',
 '.']
```

Code Comments

raw text same as above
create WordNetLemmatizer object
split raw_text into word tokens
for each word token in raw_text stem it less aggressively

Question Answers

the-The lazi-lazy judg-judge vex-vexed quack-quacking

In [ ]:
```
import nltk
raw_text = "The quick brown fox jumps over the lazy dog. Sphinx of black quartz, judge
lemmatizer = nltk.WordNetLemmatizer()
rawTokens = nltk.word_tokenize(raw_text)
[lemmatizer.lemmatize(t) for t in rawTokens]
```

Out[ ]:
```
['The',
 'quick',
 'brown',
 'fox',
 'jump',
 'over',
 'the',
 'lazy',
 'dog',
 '.',
 'Sphinx',
 'of',
 'black',
 'quartz',
 ',',
 'judge',
 'my',
 'vow',
 '!',
 'Waltz',
 'job',
 'vexed',
 'quick',
 'frog',
 'nymph',
 '.',
 'Go',
 ',',
 'lazy',
 'fat',
 'vizen',
 ';',
 'be',
 'shrewd',
 ',',
 'jump',
 'quick',
 '.',
 'Five',
 'quacking',
 'zephyr',
 'jolt',
 'my',
 'wax',
 'bed',
 '.']
```

The NLTK library is super functional with many different options to parse texts, the ability to have just a prefab word and sentance tokenizer is incredibly powerfull and the added functionality for counting occurences in a list is very usefull. The code quality seems to be pretty on par, as the methods and such do make sense according to their names. However, I dislike the nltk documentation as if I want any more insight into what *exactly* a method does the documentation does not provide it. NTLK is going to be very useful in fixing how I process user input for my text adventure side project.