As this is jupyter notebook, the nltk (including the nltk book), libraries, etc are already downloaded beforehand. The code below is extracting the first 20 tokens from text1 of the nltk text object and the output should be the first twenty tokens of text1.

Two things that I have learned about the mentioned tokens() method or text objects includes how if you want to tokenize text objects you would need to specify what text you're working with instead of doing something like calling the method. The other thing I learned was that when it comes to the tokens() method you should specify the quantity of tokens you need otherwise the ending result is that the entirety of the text is displayed.

In [1]:
```
from nltk.text import Text
from nltk.book import text1
print(text1.tokens[0:20])
```

```
*** 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
['[', 'Moby', 'Dick', 'by', 'Herman', 'Melville', '1851', ']', 'ETYMOLOGY', '.', '(',
'Supplied', 'by', 'a', 'Late', 'Consumptive', 'Usher', 'to', 'a', 'Grammar']
```

The code is printing a concordance for text1 word "sea" where we select only 5 lines and the code's output should be the 5 lines of concordance for text1 word "sea".

In [8]:
```
con_list = text1.concordance_list("sea")
print(con_list[0].line)
print(con_list[1].line)
print(con_list[2].line)
print(con_list[3].line)
print(con_list[4].line)
```

```
 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 '
```

The code experiments with both count() methods and the code's output should be the result of using the api's count() method and the result of using python's count method.

The API's count() method works through the usage of using the given word to figure out the quantity of times the work appeared in the text, where it goes through the specified text and tracks the quantity of times the word appears. The API's count() method and python's count method both keep track by counting what it is they're suppose to keep track of. However, the way they differ is through how the api's count() method counts the frequency of a work in a given text instead of doing what the python's count method does which is counting unique occurrences.

In [23]:
```
string_example = 'takodachi takodachi aloupeeps dragoons distractible'
count_example = string_example.count('takodachi')
print(count_example)

other_example = text1.count('Moby')
print(other_example)
```

```
2
84
```

The code is saving the raw text into the raw_text variable and usign the nltk's word tokenizer the thext is tokenized into tokens where the first 10 tokens are printed. The output should be the first 10 tokens of the tokenized text.

Work Cited: Hewlett-Packard Development Company. “User Guide - Hp.” HP, Hewlett-Packard Company, 29 July 2005, http://h10032.www1.hp.com/ctg/Manual/c06160570.

In [26]:
```
from nltk import word_tokenize
raw_text = "Since the display cannot show more than 10 digits of a number, numbers gre
tokens = word_tokenize(raw_text)
print(tokens[0:10])
```

```
['Since', 'the', 'display', 'can', 'not', 'show', 'more', 'than', '10', 'digits']
```

The code is using the given raw text and nltk's sentence tokenizer sent_tokenize() to do sentence segmentation. The code's output should be the segmented sentences.

In [27]:
```
from nltk import sent_tokenize
sentences = sent_tokenize(raw_text)
print(sentences)
```

```
['Since the display cannot show more than 10 digits of a number, numbers greater than
9,999,999,999 cannot be entered into the display by keying in all the digits in the n
umber.', 'However, such numbers can be easily entered into the display if the number
is expressed in a mathematical shorthand called scientific notation.', 'To convert a
number into scientific notation, move the decimal point until there is only one digi
t, a nonzero digit, to its left.', 'The resulting number is called the mantissa of th
e original number, and the number of decimal places you moved the decimal point is ca
lled the exponent of the original number.', 'If you moved the decimal point to the le
ft, the exponent is positive; if you moved the decimal point to the right, this would
occur for numbers less than one, the exponent is negative.']
```

The code uses nltk's PorterStemmer() to write a list comprehension to stem the text. The code's output should be the list of stemmed text.

In [28]:
```
from nltk.stem.porter import *
stemmer = PorterStemmer()
stemmed = [stemmer.stem(t) for t in tokens]
print(stemmed)
```

```
['sinc', 'the', 'display', 'can', 'not', 'show', 'more', 'than', '10', 'digit', 'of',
'a', 'number', ',', 'number', 'greater', 'than', '9,999,999,999', 'can', 'not', 'be',
'enter', 'into', 'the', 'display', 'by', 'key', 'in', 'all', 'the', 'digit', 'in', 't
he', 'number', '.', 'howev', ',', 'such', 'number', 'can', 'be', 'easili', 'enter',
'into', 'the', 'display', 'if', 'the', 'number', 'is', 'express', 'in', 'a', 'mathema
t', 'shorthand', 'call', 'scientif', 'notat', '.', 'to', 'convert', 'a', 'number', 'i
nto', 'scientif', 'notat', ',', 'move', 'the', 'decim', 'point', 'until', 'there', 'i
s', 'onli', 'one', 'digit', ',', 'a', 'nonzero', 'digit', ',', 'to', 'it', 'left',
'.', 'the', 'result', 'number', 'is', 'call', 'the', 'mantissa', 'of', 'the', 'origi
n', 'number', ',', 'and', 'the', 'number', 'of', 'decim', 'place', 'you', 'move', 'th
e', 'decim', 'point', 'is', 'call', 'the', 'expon', 'of', 'the', 'origin', 'number',
'.', 'if', 'you', 'move', 'the', 'decim', 'point', 'to', 'the', 'left', ',', 'the',
'expon', 'is', 'posit', ';', 'if', 'you', 'move', 'the', 'decim', 'point', 'to', 'th
e', 'right', ',', 'thi', 'would', 'occur', 'for', 'number', 'less', 'than', 'one',
',', 'the', 'expon', 'is', 'neg', '.']
```

The code uses nltk's WordNetLemmatizer and a list comprehension to lemmatize the text. The output should be the lemmatized text of the raw text.

stem results in some amount or portion of each word removed at the end of it-lemma doesn't result in some amount or portion of each word removed as high of a frequency that stem does

stem has removed the affixes from the raw text words-lemma instead found the raw text variable's root words

stem results in the given text becoming lowercase-lemma results in not of the text becoming lowercase

stem results in some words having different spellings-lemma results in the words having more or less being the same spelling, even if it is slighty cut off

stem removes some of the words end portion that have a non-character symbol attached to them-lemma results in words that have a smaller portion of each words content removed when compared to stem

In [33]:
```python
from nltk.stem import WordNetLemmatizer
wnl = WordNetLemmatizer()
lemmas = [wnl.lemmatize(t) for t in tokens]
print(lemmas)
```

```
['Since', 'the', 'display', 'can', 'not', 'show', 'more', 'than', '10', 'digit', 'o
f', 'a', 'number', ',', 'number', 'greater', 'than', '9,999,999,999', 'can', 'not',
'be', 'entered', 'into', 'the', 'display', 'by', 'keying', 'in', 'all', 'the', 'digi
t', 'in', 'the', 'number', '.', 'However', ',', 'such', 'number', 'can', 'be', 'easil
y', 'entered', 'into', 'the', 'display', 'if', 'the', 'number', 'is', 'expressed', 'i
n', 'a', 'mathematical', 'shorthand', 'called', 'scientific', 'notation', '.', 'To',
'convert', 'a', 'number', 'into', 'scientific', 'notation', ',', 'move', 'the', 'deci
mal', 'point', 'until', 'there', 'is', 'only', 'one', 'digit', ',', 'a', 'nonzero',
'digit', ',', 'to', 'it', 'left', '.', 'The', 'resulting', 'number', 'is', 'called',
'the', 'mantissa', 'of', 'the', 'original', 'number', ',', 'and', 'the', 'number', 'o
f', 'decimal', 'place', 'you', 'moved', 'the', 'decimal', 'point', 'is', 'called', 't
he', 'exponent', 'of', 'the', 'original', 'number', '.', 'If', 'you', 'moved', 'the',
'decimal', 'point', 'to', 'the', 'left', ',', 'the', 'exponent', 'is', 'positive',
';', 'if', 'you', 'moved', 'the', 'decimal', 'point', 'to', 'the', 'right', ',', 'thi
s', 'would', 'occur', 'for', 'number', 'le', 'than', 'one', ',', 'the', 'exponent',
'is', 'negative', '.']
```

My opinion of the functionality of the nltk library is that it is a useful library that makes processing text easier that works decently well. As for my opinon of the code quality of the nltk library, I think that it's pretty good quality in terms of ease of use and speed. There could be some changes to make it easier for first time users of this library to be able to better understand it though. One of the ways that I may use nltk in future projects would be to use them to complete school projects that involve processing text. Other ways I may use nltk in future projects involve using it to deal with text faster, process text in a easier way to handle, use it to make a chatbot, utilize it to improve a text to speech program, or use it for developing a apps that process text.