Chapter One Exercises

Set up the environment

In [2]:

```
import nltk
from nltk.book import *

import matplotlib.pyplot as plt                          # This piece of code is used to change the pl
ot size
plt.rcParams['figure.figsize'] = [20, 10]                # so it produces a 20 x 10 figure
```

1. Try using the Python interpreter as a calculator, and typing expressions like:

12 / (4 + 1) .

In [1]:

```
12 / (4 + 1)
```

Out[1]:

```
2.4
```

2. Given an alphabet of 26 letters, there are 26 to the power 10, or 26 ** 10 , 10-letter strings we can form. That works out to 141167095653376L (The L at the end just indicates that this is Python's lone-number format). How many hundred-letter strings are possible?

In [2]:

```
26 ** 100
```

Out[2]:

```
3142930641582938830174357788501626427282669988762475256374173175398995908420104023465432599069702288
964075081611719197835869803511992549376
```

3. The Python multiplication operation can be applied to lists. What happens when you type ['Monty','Python'] * 20 , or 3 * sent1 ?

In [4]:

```
['Monty', 'Python'] * 20
```

Out[4]:

```
['Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python',
 'Monty',
 'Python']
```

In [7]:

```
3 * sent1
```

Out[7]:

```
['Call',
 'me',
 'Ishmael',
 '.',
 'Call',
 'me',
 'Ishmael',
 '.',
 'Call',
 'me',
 'Ishmael',
 '.']
```

4. Review Section 1.1 on computing with language. How many words are there in text2? How many distinct words are there?

In [12]:

```
len(text2) # Number of words (tokens)
```

Out[12]:

```
141576
```

In [14]:

```
len(set(text2)) # Number of word types
```

Out[14]:

```
6833
```

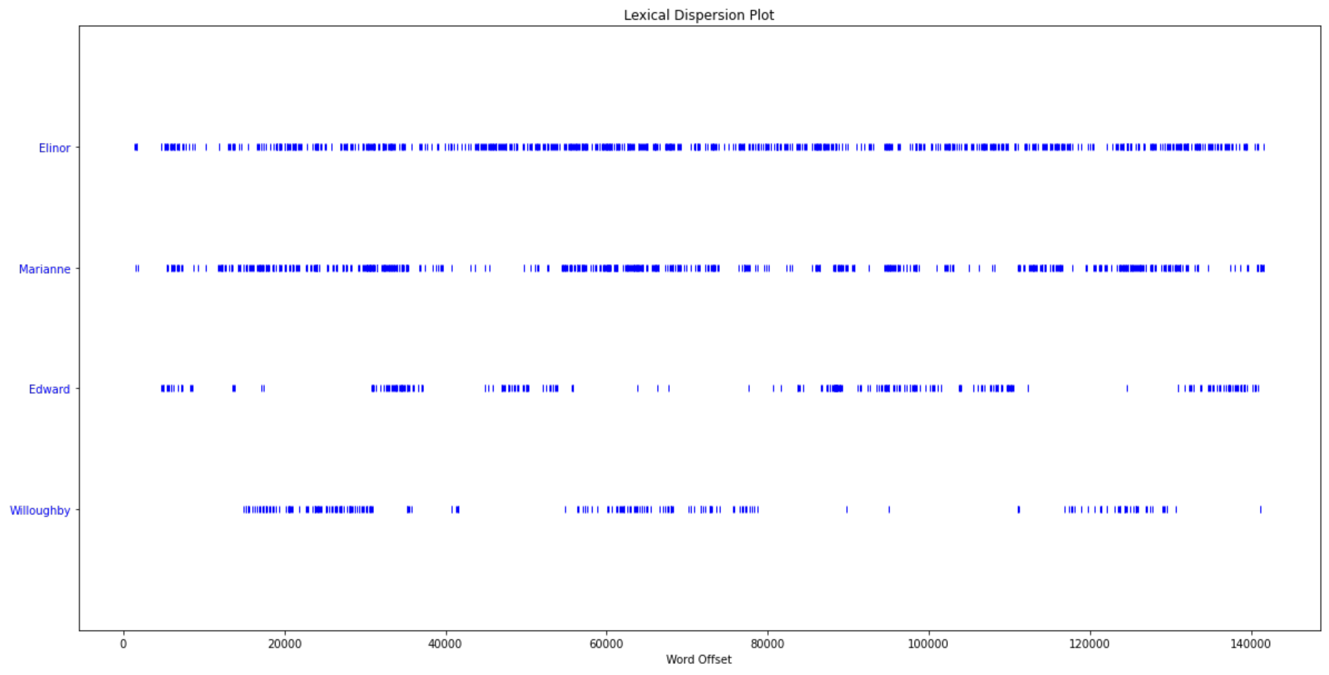
5. Compare the lexical diversity scores for humor and romance fiction in Table 1-1. Which genre is more lexically diverse?

Romance fiction is more lexically diverse that humor since it has a higher score.

6. Produce a dispersion plot of the four main protagonist in Sense and Sensibility: Elinor, Marianne, Edward, and Willoughby. What can you observe about the different roles played by the males and females in this novel? Can you identify the couples?

In [3]:

```
text2.dispersion_plot(['Elinor', 'Marianne', 'Edward', 'Willoughby'])
```



Clearly, the two main roles are those of Elinor and Marianne, with Elinor being the most prominent.

The word Edward appears often with both Elinor and Marianne, but on closer look the word Elinor appears more often with Edward than Marianne does. So, I could guess that Elinor and Edward are a couple. Similarly, Marianne and Willoghby appear together more often, so I could guess that Marianne and Willoghby are a couple.

7. Find the collocations in text5.

In [16]:

```
text5.collocations()
```

```
wanna chat; PART JOIN; MODE #14-19teens; JOIN PART; PART PART;
cute.-ass MP3; MP3 player; JOIN JOIN; times .. .; ACTION watches; guys
wanna; song lasts; last night; ACTION sits; -...)...- S.M.R.; Lime
Player; Player 12%; dont know; lez gurls; long time
```

8. Consider the following Python expression: len(set(text4)) . State the purpose of this expression. Describe the two steps involved in preforming this computation.

In [18]:

```
# The purpose of this expression is to count the number of tokens
# in text4, without counting repetitions.

# The first setp, which is the set(text4) command, is to obtain a set of
# elements consisting of tokens from text4, but without any token appearing
# in the set more than once.
# The second step, len(), is to count the number of tokens in the set.

len(set(text4))
```

Out[18]:

```
9913
```

9. Review Section 1.2 on lists and strings.

a. Define a string and assign it to a variable, e.g. my_string = 'My String' (but put something more interesting in the string). Print the contents of this variable in two ways, first by simply typing the variable name and pressing Enter , then by using the print statement.

In [24]:

```
my_string = 'This is a string with something more interesting in it than what was originally sugge
sted.'
```

In [25]:

```
my_string
```

Out[25]:

```
'This is a string with something more interesting in it than what was originally suggested.'
```

In [23]:

```
print(my_string)
```

```
This is a string with something more interesting in it than what was originally suggested.
```

b. Try adding the string to itself using my_string + my_string , or by multiplying it by a number, e.g., my_string * 3 . Notice that the strings are joined together without any spaces. How could you fix this?

In [26]:

```
my_string + my_string       # Adding string to itself
```

Out[26]:

```
'This is a string with something more interesting in it than what was originally suggested.This is
a string with something more interesting in it than what was originally suggested.'
```

In [27]:

```
my_string * 3        # Multiplying string by 3
```

Out[27]:

```
'This is a string with something more interesting in it than what was originally suggested.This is
a string with something more interesting in it than what was originally suggested.This is a string
with something more interesting in it than what was originally suggested.'
```

In [28]:

```
# One solution could be to add a "buffer", like a space character ' '
# in between the two string, like so:
my_string + ' ' + my_string
```

Out[28]:

```
'This is a string with something more interesting in it than what was originally suggested. This i
s a string with something more interesting in it than what was originally suggested.'
```

In [32]:

```
# Same solution would work with multiplication, with the caveat that
# the space character ' ' would need to be added to the string first.
(my_string  + ' ') * 3
```

Out[32]:

```
'This is a string with something more interesting in it than what was originally suggested. This i
s a string with something more interesting in it than what was originally suggested. This is a str
ing with something more interesting in it than what was originally suggested. '
```

10. Define a variable my_sent to be a list of words, using the syntax my_sent = ["My", "sent"] (but with your own words, or a favorite saying).

In [48]:

```
my_sent = ["The", "limits", "of", "my", "language", "mean", "the", "limits", "of", "my", "world"]
```

a. Use ' '.join(my_sent) (note the space between the single quotes) to convert this into a string.

In [49]:

```
my_sent = ' '.join(my_sent)
my_sent
```

Out[49]:

```
'The limits of my language mean the limits of my world'
```

b. Use split() to split the string back into the list form you had to start with.

In [50]:

```
my_sent = my_sent.split()
my_sent
```

Out[50]:

```
['The',
 'limits',
 'of',
 'my',
 'language',
 'mean',
 'the',
 'limits',
 'of',
 'my',
 'world']
```

11. Define several variables containing lists of words, e.g., phrase1 , phrase2 , and so on. Join them together in various combinations (using the plus operator) to form whole sentences. What is the relationship between len(phrase1 + phrase2) and len(phrase1) + len(phrase2) ?

In [51]:

```
phrase1 = ['This', 'is', 'the', 'first', 'sentence']
phrase2 = ['blue', 'red', 'green', 'yellow', 'brown']
phrase3 = ['California', 'Texas', 'Oregon', 'Colorado', 'Montana']
phrase4 = ['left', 'right', 'up', 'down']
```

In [52]:

```
phrase1 + phrase2
```

Out[52]:

```
['This',
 'is',
 'the',
 'first',
 'sentence',
 'blue',
 'red',
 'green',
 'yellow',
 'brown']
```

In [53]:

```
phrase2 + phrase3
```

Out[53]:

```
['blue',
 'red',
 'green',
 'yellow',
 'brown',
 'California',
 'Texas',
 'Oregon',
 'Colorado',
 'Montana']
```

In [54]:

```
phrase3 + phrase4
```

Out[54]:

```
['California',
 'Texas',
 'Oregon',
 'Colorado',
 'Montana',
 'left',
 'right',
 'up',
 'down']
```

In [55]:

```
phrase1 + phrase4
```

Out[55]:

```
['This', 'is', 'the', 'first', 'sentence', 'left', 'right', 'up', 'down']
```

In [56]:

```
len(phrase1 + phrase2)
```

Out[56]:

```
10
```

In [57]:

```
len(phrase1) + len(phrase2)
```

Out[57]:

```
10
```

The relationship between the two commands above shows the distributive property of the len( ) function.

12. Consider the following two expressions, which have the same value. Which one will typically be more relevant in NLP? Why?

a. "Monty Python"[6:12]

b. ["Monty", "Python"][1]

I believe that option b will be more helpful since the expression is a list, and list manipulation is significantly easier to do than individual string manipulation, as it's the case with option a.

13. We have seen how to represent a sentence as a list of words, where each word is a sequence of characters. What does sent1[2][2] do? Why? Experiment with other index values.

In [59]:

```
sent1
```

Out[59]:

```
['Call', 'me', 'Ishmael', '.']
```

In [58]:

```
sent1[2][2]
```

Out[58]:

```
'h'
```

What the expression sent1[2][2] does is return the third character from the third element in the sent1 list (the 'h' in Ishmael).

14. The first sentence of text3 is provided to you in the variable sent3 . The index of *the* in sent3 is 1, because sent3[1] gives us 'the' . What are the indexes of the two other occurences of this word in sent3?

In [61]:

```
sent3
```

Out[61]:

```
['In',
 'the',
 'beginning',
 'God',
 'created',
 'the',
 'heaven',
 'and',
 'the',
 'earth',
 '.']
```

In [68]:

```
for w,i in zip(sent3, range(len(sent3))):
    if w == "the":
        print(i)
```

```
1
5
8
```

The indexes of the other two occurences of the word 'the' are 5 and 8.

15. Review the discussion of conditionals in Section 1.4. Find all words in the Chat Corpus (text5) starting with the letter b. Show them in alphabetical order.

In [6]:

```
# Assuming search is case sensitive (i.e., only lowercase b words)
# Also, output has been suppressed since it takes up a lot of space.

sorted([w for w in text5 if w.startswith('b')]);
```

In [7]:

```
# If search need to be case sensitive (i.e., words with b or B), then
sorted([w for w in text5 if w.lower().startswith('b')]);                # output suppressed
```

16. Type the expression range(10) at the interpreter prompt. Now try range(10, 20), range(10, 20, 2), and range(20, 10, -2). We will see a variety of uses for this built-in function in later chapters.

In [1]:

```
range(10)
```

Out[1]:

```
range(0, 10)
```

In [2]:

```
range(10, 20)
```

Out[2]:

```
range(10, 20)
```

In [3]:

```
range(10, 20, 2)
```

Out[3]:

```
range(10, 20, 2)
```

In [4]:

```
range(10, 20, -2)
```

Out[4]:

```
range(10, 20, -2)
```

17. Use text9.index( ) to find the index of the word *sunset*. You'll need to insert this word as an argument between the parentheses. By a process of trial and error, find the slice for the complete sentence that contains this word.

In [6]:

```
text9.index('sunset')
```

Out[6]:

```
629
```

In [8]:

```
' '.join(text9[600:650])
```

Out[8]:

```
', and you may safely read . G . K . C . CHAPTER I THE TWO POETS OF SAFFRON PARK THE suburb of Saf
fron Park lay on the sunset side of London , as red and ragged as a cloud of sunset . It was built
of a bright'
```

In [9]:

```
' '.join(text9[600:630])
```

Out[9]:

```
', and you may safely read . G . K . C . CHAPTER I THE TWO POETS OF SAFFRON PARK THE suburb of Saf
```

```
fron Park lay on the sunset'
```

In [11]:

```
' '.join(text9[580:630])
```

Out[11]:

```
'We have found common things at last and marriage and a creed , And I may safely write it now , an
d you may safely read . G . K . C . CHAPTER I THE TWO POETS OF SAFFRON PARK THE suburb of Saffron
Park lay on the sunset'
```

In [12]:

```
' '.join(text9[570:630])
```

Out[12]:

```
'strength in striking root and good in growing old . We have found common things at last and marri
age and a creed , And I may safely write it now , and you may safely read . G . K . C . CHAPTER I
THE TWO POETS OF SAFFRON PARK THE suburb of Saffron Park lay on the sunset'
```

In [15]:

```
' '.join(text9[580:630])
```

Out[15]:

```
'We have found common things at last and marriage and a creed , And I may safely write it now , an
d you may safely read . G . K . C . CHAPTER I THE TWO POETS OF SAFFRON PARK THE suburb of Saffron
Park lay on the sunset'
```

18. Using list addition, and the set and sorted operations, compute the vocabulary of the sentences sent1 ... sent8.

In [18]:

```
sorted(set(sent1 + sent2 + sent3 + sent4 + sent5 + sent6 + sent7 + sent8))
```

Out[18]:

```
['!',
 ',',
 '-',
 '.',
 '1',
 '25',
 '29',
 '61',
 ':',
 'ARTHUR',
 'Call',
 'Citizens',
 'Dashwood',
 'Fellow',
 'God',
 'House',
 'I',
 'In',
 'Ishmael',
 'JOIN',
 'KING',
 'MALE',
 'Nov.',
 'PMing',
 'Pierre',
 'Representatives',
 'SCENE',
 'SEXY',
 'Senate',
 'Sussex',
 'The',
 'Vinken',
 'Whoa',
 '[',
 ']',
 'a',
 'and',
 'as',
 'attrac',
 'been',
 'beginning',
 'board',
 'clop',
 'created',
 'director',
 'discreet',
 'earth',
 'encounters',
 'family',
 'for',
 'had',
 'have',
 'heaven',
 'in',
 'join',
 'lady',
 'lol',
 'long',
 'me',
 'nonexecutive',
 'of',
 'old',
 'older',
 'people',
 'problem',
 'seeks',
 'settled',
 'single',
 'the',
 'there',
 'to',
 'will',
 'wind',
 'with',
 'years']
```

19. What is the difference between the following two lines? Which one will give a larger value? Will this be the case for other texts?

>>> sorted(set([w.lower( ) for w in text1]))

>>> sorted([w.lower( ) for w in set(text1)])

In [19]:

```
len(set([w.lower() for w in text1]))
```

Out[19]:

```
17231
```

In [20]:

```
len([w.lower() for w in set(text1)])
```

Out[20]:

```
19317
```

One difference between these two expressions is that the first performes the pruning of the text1 after all words have been set to lowercase. The second expression does the pruning first, then what ever words are left are set to lowercase.

The second expression will contain more tokens because, to start with, the set funciton is case sensitive when it comes to

The second expression will contain more tokens because, to start with, the set funciton is case sensitive when it comes to processing input. So, when the first expression sets all the words to lowercase before doing the pruning some words were set equal to their lowercase version (i.e. 'The' and 'the' are now the same).

Yes, this will always be the case.

20. What is the difference between the following two tests: w.isupper( ) and not w.islower( )?

The main difference between the two test is the not. In the second test, the not negates the result of the w.islower( ) test.

In [26]:

```
my_string = "Test"
```

In [27]:

```
my_string.isupper()
```

Out[27]:

```
False
```

In [29]:

```
my_string.islower()
```

Out[29]:

```
False
```

21. Write the slice expression that extracts the last two words of text2.

In [30]:

```
text2[-2:]
```

Out[30]:

```
['THE', 'END']
```

22. Find all the four-letter words in the *Chat Corpus* (text5). With the help of a frequency distribution (FreqDist), shows these words in decreasing order of frequency.

In [10]:

```
freq_dist = FreqDist([w for w in text5 if len(w) == 4])
```

In [11]:

```
freq_dist.keys();
```

23. Review the discussion of looping with conditions in Section 1.4. Use a combination of for and if statements to loop over the words of the movie script for *Monty Python and the Holy Grail* (text6) and print all the uppercase words, one per line.

In [13]:

```
# Output has been suppressed since it takes up a lot of space.

for w in text5:
    if w.isupper():
        print(w)
```

24. Write expressions for finding all words in text6 that meet the following conditions. The result should be in the form of a list of words: ['word1', 'word2', ...]

list of words: ['word1', 'word2', ...].

a. Ending in *ize*

b. Containing the letter *z*

c. Containing the sequence of letters *pt*

d. All lowercase letters except for an initial capital(i.e., titlecase)

In [43]:

```
[w for w in text6 if w.endswith('ize')]         # a
```

Out[43]:

```
[]
```

In [44]:

```
[w for w in text6 if 'z' in w]                  # b
```

Out[44]:

```
['zone',
 'amazes',
 'Fetchez',
 'Fetchez',
 'zoop',
 'zoo',
 'zhiv',
 'frozen',
 'zoosh']
```

In [45]:

```
[w for w in text6 if 'pt' in w]                  # c
```

Out[45]:

```
['empty',
 'aptly',
 'Thpppppt',
 'Thppt',
 'Thppt',
 'empty',
 'Thppppt',
 'temptress',
 'temptation',
 'ptoo',
 'Chapter',
 'excepting',
 'Thpppt']
```

In [14]:

```
[w for w in text6 if w.istitle()];                 # d (Output suppressed)
```

25. Define sent to be the list of words ['she', 'sells', 'sea', 'shells', 'by', 'the', 'sea', 'shore']. Now write code to perform the following tasks:

a. Print all words beginning with sh.

b. Print all words longer than four characters.

In [47]:

```
sent = ['she', 'sells', 'sea', 'shells', 'by', 'the', 'sea', 'shore']
```

In [49]:

```
# a
for w in sent:
    if w.startswith('sh'):
        print(w)
```

```
she
shells
shore
```

In [50]:

```
# b
for w in sent:
    if len(w) > 4:
        print(w)
```

```
sells
shells
shore
```

26. What does the following Python code do? sum([len(w) for w in text1]). Can you use it to work out the average word length of a text?

The above expression sums up all the numbers in the list.

In [54]:

```
avg_word_length = sum([len(w) for w in text1]) / len(text1)
avg_word_length
```

Out[54]:

```
3.830411128023649
```

27. Define a function called vocab_size(text), that has a single parameter for the text, and which returns the vocabulary size of the text.

In [55]:

```
def vocab_size(text):
    return len(set(text))
```

28. Define a function percent(word, text), that calculates how often a given word occurs in a text and express the result as a percentage.

In [4]:

```
def percent(word,text):
    count = text.count(word)
    percent =  100 * (count / len(text))
    print(str(round(percent, 2)) + '%')
```

In [5]:

```
percent('the', text1)
```

```
5.26%
```

29. We have been using sets to store vocabularies. Try the following Python expression: set(sent3) < set(text1). Experiment with this using different arguments to set( ). What does it do? Can you think of a practical application for this?

In [75]:

```
set(sent3) < set(text1)
```

Out[75]:

```
True
```

In [76]:

```
set(sent1) < set(text1)
```

Out[76]:

```
True
```

In [77]:

```
A = ['a', 'b', 'c']
B = ['d', 'e', 'f']
S = ['a', 'b', 'c', 'e', 'f']
```

In [78]:

```
A < S
```

Out[78]:

```
True
```

In [79]:

```
B < S
```

Out[79]:

```
False
```

From this smaller example it can be seen that the way the relational operator works with sets is the same way as they work in mathematics.

A practical application could be to categorize customers/users.