Exercise :- 1

step :- 1

```
import io
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt.
with io.open('count_2w.txt','r',encoding='utf8') as f:
    text = f.readlines()
```

step 2 :-

```
mini = text[:10]
mini = text[:7]
mini[0].split()
mini_list = []
for m in mini:
    (w1, w2, count) = m.split()
    count = int(count)
    mini_list.append(((w1, w2), count))
mini_list
    mini_list[0]
geog2w_list = []
for m in mini:
    (w1, w2, count) = m.split()
    count = int(count)
    geog2w_list.append(((w1,w2), count))
geog2w_list
```

Natural Language Processing Lab
Lab2. Computing Bigram Frequencies

EXERCISE-1: Process simple bigram data file

STEP 1: OPEN the file, count_2w.txt

When you open it, make sure to specify UTF-8 encoding, otherwise you will see the very last line break. Then, it's business as usual, i.e., reading the file in as a list of lines.

STEP 2: build goog2w_list

First up, build goog2w_list as a list of ((w1, w2), count) tuples. The file this time around is not ordered by frequency, it's ordered alphabetically. That's why we are calling it _list instead of _rank. Here's the process with a mini version:

```
>>> mini = lines[:10]

>>> mini[0]
'0Uplink verified\t523545\n'

>>> mini[0].split()
['0Uplink', 'verified', '523545']

>>> mini_list = []
>>> for m in mini:
        (w1, w2, count) = m.split()
        count = int(count)
        mini_list.append(((w1, w2), count))

>>> mini_list

>>> mini_list[0]
(('0Uplink', 'verified'), 523545)
```

STEP 3: build goog2w_fd

Next, build goog2w_fd as a frequency distribution, implemented as nltk.FreqDist. When finished, it should work like:

```
>>> goog2w_fd[('of', 'the')]
2766332391
>>> goog2w_fd[('so', 'beautiful')]
612472
```

STEP 4: explore

Now explore the two data objects to familiarize yourself with the bigram data. Answer the following questions:

1. What are the top-10 bigrams?
2. What are the top *so*-initial bigrams?
3. Back to those bigrams necessary for computing the probability of the sentence 'She was not afraid.'. Are they all found in this data?
4. Find a bigram that you think should be represented and it is.
5. Find a bigram that you think should be represented but is not.

goog2w_list[0]

Steps: build goog2w_fd.

```
! pip install nltk

import nltk.
goog2w_fd = nltk.FreqDist()
goog2w_fd.
for m in text:
    w1, w2, count = m.split()
    goog2w_fd[(w1,w2)] = count.
goog2w_fd[('ab', 'the')]
goog2w_fd[('so', 'beautiful')]
```

Step: 4 explore

1) goog2w_fd.most_common[:10]

Step: 5

```
import pickle as pkl
with open('goog2w_list.pkl', 'ab') as handle:
    pkl.dump(goog2w_list, handle)
with open('goog2w_fd.pkl', 'ab') as handle:
    pkl.dump(goog2w_fd, handle)
```

Exer: 2

```
with open('austen-emma.txt', 'r') as f1:
    cona = f1.read()
```

STEP 5: pickle the data

Pickle `goog2w_list` as 'goog2w_list.pkl', and `goog2w_fd` as 'goog2w_fd.pkl'.

EXERCISE-2: Frequency distribution from Jane Austen Novels

Goto www.nltk.org/nltk_data. Download the zipped archive, **gutenberg.zip**.

This zip file contains 3 novels of Jane Austen (**austen-emma.txt, austen-persuasion.txt, austen-sense.txt**)

Your job is to write a python script that processes the corpora for some basic stats:

A. opens (and later closes) the text file, reads in the string content,
B. builds a list of individual sentences,
C. prints out how many sentences there are,
D. builds a flat tokenized word list and the type list,
E. prints the token and the type counts of this corpus,
F. builds a frequency count dictionary of words,
G. prints the top 50 word types and their counts.

Finally, make one observation about the corpora. It could involve some new code of your own not included above, or it could be based off of A.--F. above.

EXERCISE-3: Bigram Frequencies of Jane Austen Novels

Here, we will take a close look at the bigram frequencies of ***Jane Austen*** novels. We are interested in what types of word bigrams are frequently found in the corpus, and also what types of words are found following the word 'so', and in what probability. Additionally, we will pickle the bigram frequency dictionaries so we can re-use them later.

A. imports necessary modules,
B. opens the text files and reads in the content as text strings,
C. builds the following objects, `a_` for Austen:
    1. `a_toks`: word tokens, all in lowercase
    2. `a_tokfd`: word frequency distribution
    3. `a_bigrams`: word bigrams, cast as a list
    4. `a_bigramfd`: bigram frequency distribution
    5. `a_bigramcfd`: bigram `(w1, w2)` conditional frequency distribution ("CFD"), where `w1` is construed as the condition and `w2` the outcome
D. pickles the bigram CFDs (conditional frequency distributions) using the highest binary protocol: name the file as `austen_bigramcfd.pkl`.
E. answers the following questions by exploring the objects:
    1. How many word tokens and types are there? what is its size?
    2. What are the top 20 most frequent words and their counts?. Draw chart using Matplotlib's plot() method.
    3. What are the top 20 most frequent word bigrams and their counts?, omitting bigrams that contain stopwords
    4. What are the top 20 most frequent word bigrams and their counts, omitting bigrams that contain stopwords?
    5. What are the top 20 most frequent word bigrams and their counts, omitting bigrams that contain stopwords?. Draw chart using Matplotlib's plot() method.

```
with open ('austen_persuasion.txt', 'r') as flp:
    comp = flp.read()
with open ('austen_sense.txt', 'r') as fls:
    cons = fls.read()
```

B. builds a list of individual sentences,

```
from nltk.tokenize import sent_tokenize as st
st(cora)
st(comp)
st(cons)
```

C. prints out how many sentences

```
print(len(st(cora)))
print(len(st(comp)))
print(len(st(cons)))
```

E. prints the token and the type counts of this corpus

```
from nltk.tokenize import word_tokenize
t1 = word_tokenize(cora)
print(t1)
t2 = word_tokenize(comp)
print(t2)
t3 = word_tokenize(cons)
print(t3)
```

F. builds a frequency count dictionary of words,

```
from nltk import
```

6. How many times does the word 'so' occur? What are their relative frequency against the corpus size (= total # of tokens)?
7. What are the top 20 'so-initial' bigrams (bigrams that have the word "so" as the first word) and their counts?
8. Given the word 'so' as the current word, what is the probability of getting 'much' as the next word?
9. Given the word 'so' as the current word, what is the probability of getting 'will' as the next word?

```
dal = freqDist(t1)
dal

da2 = freq.Dist(t2)
da2

da3 = FreqDist(t3)
da3
dal.most_common(50)
da2.most_common(50)
da3.most_common(50)
```

Exercise 3:

A. Imports necessary modules,

B. Opens the text files:

```
with open("jane_austen.txt").as fn:
    nov = fn.read()
print(nov)

tokenizer = nltk.tokenize.WhitespaceTokenizer()
tok = tokenizer.tokenize(nov)
tok,
b2 = list(nltk.bigrams(tok))
b2fd = nltk.FreqDist(b2)
b2fd
```

```
import re
from collections import Counter.
words = re.findall(r'\s+\w+', open('jane_austen.txt').read())
ab = Counter(zip(words))
print(ab)
```

c. builds the following objects.

1. a_toks: word tokens,

```
tokenizer = nltk.tokenize.WhitespaceTokenizer()
a_toks = tokenizer.tokenize(nov.lower())
a_toks
```

2. a_tokfd : word frequency distribution

```
a_tokfd = freqDist(a_toks)
a_tokfd
```

3. a_bigrams :

```
a_bigrams = list(nltk.bigrams(a_toks))
a_bigrams.
```

4. a_bigramfd : bigram freq dist.

```
a_bigramfd = nltk.FreqDist(a_bigrams)
a_bigramfd.
```

5. a_bigramcfd : bigram (w1, w2) conditional frequency distribution ("CFD")

```
from nltk.probability import conditionalFreqDist
from nltk.tokenize import word_tokenize
a_bigramcfd = conditionalFreqDist()
a_bigramfd = conditionalFreqDist()
for word in a_toks:
    condition = len(word)
    a_bigramcfd[condition][word] += 1
```

NOTES

a_bigramfd

D) with open ('austen_bigramfd.pkl', 'ab') as handle:
    pkl.dump(a_bigramfd, handle)

E. Answers the following

1) len(a_toks)

2) ws = a_tokfd.most_common(20)
   m = dict(ws)
   m
   df = pd.dataframe(list(m.items()))
   df.columns = ['single', 'count']
   df

   df.plot(kind = 'line', x = 'single', y = 'count', color = 'blue')
   plt.show()

4) v = a_bigramfd.most_common(20)
   m = dict(v)
   m
   df2 = pd.Dataframe(list(m.items()))
   df2.columns = ['bigram', 'count']
   df2.

5) what: df2.plot(kind = 'line', x = 'bigram', y = 'count', color = 'red')
   plt.show()

6) so_count = a_tokfd['so']
   print(so_count)
   tot = len(a_tokfd)
   print(tot)
   rel_freq = so count / tot
   rel_freq

```
ab.most_common(20)

ab_dict = dict(ab)

ab_dict

tot_occ = len(ab_dict)

tot_occ

for i,j in ab_dict.items():
    if i == ('so much'):
        print(i,j)
        print(j/tot_occ)

for i,j in ab_dict.items():
    if i == ('so will'):
        print(i,j)
        print(j/tot_occ)
```