In [1]:

```
#TASK 1 DEMOSTRATING COCA CORPUS
'''The Corpus of Contemporary American English (COCA) is the only large, genre-balanced cor
COCA is probably the most widely-used corpus of English,
and it is related to many other corpora of English that we have created,
which offer unparalleled insight into variation in English.'''
```

In [2]:

```
#TASK 2 STEMMING (PorterStemmer):

'''Stemming is basically removing the suffix from a word and reduce it to its root word.
For example: “Flying” is a word and its suffix is “ing”,
if we remove “ing” from “Flying” then we will get base word or root word which is “Fly”.
We uses these suffix to create a new word from original stem word.'''

import nltk
from nltk.stem import PorterStemmer
Stemmerporter=PorterStemmer()
Stemmerporter.stem('lemmatization')
```

Out[2]:

```
'lemmat'
```

In [3]:

```
#TASK 2 STEMMING:
import nltk
from nltk.stem import PorterStemmer
Stemmerporter=PorterStemmer()
Stemmerporter.stem('cheerfulness')
```

Out[3]:

```
'cheer'
```

In [15]:

```
#LancasterStemmer

'''Lancaster: Very aggressive stemming algorithm, sometimes to a fault.
With porter and snowball, the stemmed representations are usually fairly intuitive to a rea
not so with Lancaster, as many shorter words will become totally obfuscated.'''

import nltk
from nltk.stem import LancasterStemmer
stemmerLan =LancasterStemmer()
stemmerLan.stem('technology')
```

Out[15]:

```
'technolog'
```

In [5]:

```
#RegexpStemmer

'''A stemmer that uses regular expressions to identify morphological affixes.
Any substrings that match the regular expressions will be removed.'''

import nltk
from nltk.stem import RegexpStemmer
stemmerregexp=RegexpStemmer('learn')
stemmerregexp.stem('learning')
```

Out[5]:

```
'ing'
```

In [6]:

```
#SnowballStemmer : Snowball is a small string processing language designed for creating ste

# Include multiple languages.

import nltk
from nltk.stem import SnowballStemmer
SnowballStemmer.languages
frenchstemmer=SnowballStemmer('french')
frenchstemmer.stem('manges')
```

Out[6]:

```
'mang'
```

In [7]:

```
#TASK 3: STEMMING PARAGRAPHS

# Here all the individual tokens of the sentence are reduced to their stems.

from nltk.stem import PorterStemmer

stemmer = PorterStemmer()
example = "Am quick brown fox jumps over a lazy dog"
example = [stemmer.stem(token) for token in example.split(" ")]
print (" ".join(example))
```

```
Am quick brown fox jump over a lazi dog
```

In [8]:

```
# TASK 4: LEMMATIZER
'''Lemmatization is the process of grouping together the different inflected forms of a wor
Lemmatization is similar to stemming but it brings context to the words. So it links words


'''Text preprocessing includes both Stemming as well as Lemmatization.
Many times people find these two terms confusing. Some treat these two as same.
Actually, lemmatization is preferred over Stemming because lemmatization does morphological

import nltk
from nltk.corpus import wordnet as wn
from nltk.stem.wordnet import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("cacti"))
print(lemmatizer.lemmatize("mice"))
print(lemmatizer.lemmatize("rocks"))

print(lemmatizer.lemmatize("better", pos = 'a')) # given the part-of-speech, better Lemmati

print(lemmatizer.lemmatize("Am"))   # This error is fixed when the PArt of Speech is given

print(lemmatizer.lemmatize("am", pos = 'v'))
```

```
cactus
mouse
rock
good
Am
be
```

In [9]:

```
# TASK 5: CHINESE SEGMENTATION USING JIEBA

import jieba
seg = jieba.cut("把句子中所有的可以成词的词语都扫描出来", cut_all = True)
print(" ".join(seg))
```

```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\91814\AppData\Local\Temp\jieba.cache
Loading model cost 1.431 seconds.
Prefix dict has been built successfully.

把 句子 中所 所有 的 可以 成 词 的 词语 都 扫描 描出 描出来 出来
```

In [10]:

```
# TASK 6: BASIC TEXT PROCESSING PIPELINE
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
import nltk
sent = "Become an expert in NLP"
words = nltk.word_tokenize(sent)  #Tokenizer just seprate apart the words or Tokens of a se
print(words)
```

```
['Become', 'an', 'expert', 'in', 'NLP']

[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\91814\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\91814\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

In [11]:

```
texts = ["""The only true wisdom is in knowin' you know nothing.
Beware the barrenness of a busy life.
I decided that it was not wisdom that enabled poets to write their poetry,
but a kind of ins. or inspiration, such as you find in seers and prophets who deliver all t
count=0;
for text in texts:
    sentences = nltk.sent_tokenize(text)
    for sentence in sentences:
        words = nltk.word_tokenize(sentence)
        print(words)
        tagged = nltk.pos_tag(words) #This taggs the part of speech as we can see that in c
        print(tagged)
num_words = [len(sentence.split()) for sentence in texts]
print('total word',num_words)
```

```
['The', 'only', 'true', 'wisdom', 'is', 'in', 'knowin', "'", 'you', 'kno
w', 'nothing', '.']
[('The', 'DT'), ('only', 'JJ'), ('true', 'JJ'), ('wisdom', 'NN'), ('is',
'VBZ'), ('in', 'IN'), ('knowin', 'NN'), ("'", "''"), ('you', 'PRP'), ('kno
w', 'VBP'), ('nothing', 'NN'), ('.', '.')]
['Beware', 'the', 'barrenness', 'of', 'a', 'busy', 'life', '.']
[('Beware', 'NNP'), ('the', 'DT'), ('barrenness', 'NN'), ('of', 'IN'),
('a', 'DT'), ('busy', 'JJ'), ('life', 'NN'), ('.', '.')]
['I', 'decided', 'that', 'it', 'was', 'not', 'wisdom', 'that', 'enabled',
'poets', 'to', 'write', 'their', 'poetry', ',', 'but', 'a', 'kind', 'of',
'ins', '.']
[('I', 'PRP'), ('decided', 'VBD'), ('that', 'IN'), ('it', 'PRP'), ('was',
'VBD'), ('not', 'RB'), ('wisdom', 'JJ'), ('that', 'IN'), ('enabled', 'VB
D'), ('poets', 'NNS'), ('to', 'TO'), ('write', 'VB'), ('their', 'PRP$'),
('poetry', 'NN'), (',', ','), ('but', 'CC'), ('a', 'DT'), ('kind', 'NN'),
('of', 'IN'), ('ins', 'NNS'), ('.', '.')]
['or', 'inspiration', ',', 'such', 'as', 'you', 'find', 'in', 'seers', 'an
d', 'prophets', 'who', 'deliver', 'all', 'their', 'sublime', 'messages',
'without', 'knowing', 'in', 'the', 'least', 'what', 'they', 'mean', '.']
[('or', 'CC'), ('inspiration', 'NN'), (',', ','), ('such', 'JJ'), ('as',
'IN'), ('you', 'PRP'), ('find', 'VBP'), ('in', 'IN'), ('seers', 'NNS'),
('and', 'CC'), ('prophets', 'NNS'), ('who', 'WP'), ('deliver', 'VBP'), ('a
ll', 'DT'), ('their', 'PRP$'), ('sublime', 'NN'), ('messages', 'NNS'), ('w
ithout', 'IN'), ('knowing', 'VBG'), ('in', 'IN'), ('the', 'DT'), ('least',
'JJS'), ('what', 'WP'), ('they', 'PRP'), ('mean', 'VBP'), ('.', '.')]
['Be', 'as', 'you', 'wish', 'to', 'seem', '.']
[('Be', 'VB'), ('as', 'IN'), ('you', 'PRP'), ('wish', 'VBP'), ('to', 'T
O'), ('seem', 'VB'), ('.', '.')]
['Wonder', 'is', 'the', 'beginning', 'of', 'wisdom', '.']
[('Wonder', 'NNP'), ('is', 'VBZ'), ('the', 'DT'), ('beginning', 'NN'), ('o
f', 'IN'), ('wisdom', 'NN'), ('.', '.')]
['Be', 'kind', ',', 'for', 'everyone', 'you', 'meet', 'is', 'fighting',
'a', 'hard', 'battle', '.']
[('Be', 'NNP'), ('kind', 'NN'), (',', ','), ('for', 'IN'), ('everyone', 'N
N'), ('you', 'PRP'), ('meet', 'VBP'), ('is', 'VBZ'), ('fighting', 'VBG'),
('a', 'DT'), ('hard', 'JJ'), ('battle', 'NN'), ('.', '.')]
['Our', 'prayers', 'should', 'be', 'for', 'blessings', 'in', 'general',
',', 'for', 'God', 'knows', 'best', 'what', 'is', 'good', 'for', 'us',
'.']
[('Our', 'PRP$'), ('prayers', 'NNS'), ('should', 'MD'), ('be', 'VB'), ('fo
r', 'IN'), ('blessings', 'NNS'), ('in', 'IN'), ('general', 'JJ'), (',',
','), ('for', 'IN'), ('God', 'NNP'), ('knows', 'VBZ'), ('best', 'JJS'),
('what', 'WP'), ('is', 'VBZ'), ('good', 'JJ'), ('for', 'IN'), ('us', 'PR
P'), ('.', '.')]
total word [100]
```

In [14]:

```
from collections import Counter
my_counter = Counter()
for word in words:
    my_counter.update(word)
print(my_counter)
```

```
Counter({'s': 9, 'o': 8, 'r': 7, 'e': 6, 'n': 4, 'u': 3, 'a': 3, 'l': 3,
'd': 3, 'b': 3, 'f': 3, 'i': 3, 'g': 3, 'h': 2, 'w': 2, 't': 2, 'O': 1, 'p':
1, 'y': 1, ',': 1, 'G': 1, 'k': 1, '.': 1})
```

In [ ]: