In [1]:
```
#TASK 1 DEMOSTRATING COCA CORPUS
''''A concordance is an alphabetical list of the principal words used in a book o
listing every instance of each word with its immediate context.'''
```

Out[1]:
```
"'A concordance is an alphabetical list of the principal words used in a book o
r body of work, \nlisting every instance of each word with its immediate contex
t."
```

In [3]:
```
'''
Morphological analysis is the process of providing grammatical information about
Morphological analyzer and generator are the two essential and basic tools for bu

Stemming is the process of reducing a word to its word stem that affixes to suffi
and prefixes or to the roots of words known as a lemma. Stemming is important in
and natural language processing (NLP).'''
#TASK 2 STEMMING:
import nltk
from nltk.stem import PorterStemmer
Stemmerporter=PorterStemmer()
Stemmerporter.stem('lemmatization')
```

Out[3]:
```
'lemmat'
```

In [4]:
```
''''The Porter stemming algorithm (or 'Porter stemmer')
is a process for removing the commoner morphological and inflexional endings fro
Its main use is as part of a term normalisation process that is usually done whe
#TASK 2 STEMMING:
import nltk
from nltk.stem import PorterStemmer
Stemmerporter=PorterStemmer()
Stemmerporter.stem('cheerfulness')
```

Out[4]:
```
'cheer'
```

In [5]:
```
'''Lancaster: Very aggressive stemming algorithm, sometimes to a fault.
With porter and snowball, the stemmed representations are usually fairly intuitiv
not so with Lancaster, as many shorter words will become totally obfuscated'''
import nltk
from nltk.stem import LancasterStemmer
stemmerLan =LancasterStemmer()
stemmerLan.stem('happiness')
```

Out[5]:
```
'happy'
```

In [16]:
```
''''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[16]: 'ing'

In [17]:
```
'''Snowball. Snowball is a small string processing language designed for creating
for use in Information Retrieval.
presents several useful stemmers which have been implemented using it.'''
import nltk
from nltk.stem import SnowballStemmer
SnowballStemmer.languages
frenchstemmer=SnowballStemmer('french')
frenchstemmer.stem('manges')
```

Out[17]: 'mang'

In [18]:
```
#TASK 3: STEMMING PARAGRAPHS

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 [19]:
```
'''WordNet is the lexical database i.e.
dictionary for the English language, specifically designed for natural language p


Lemmatization is the process of grouping together the different inflected forms o
so they can be analysed as a single item. Lemmatization is similar to stemming bu
So it links words with similar meaning to one word.

Text preprocessing includes both Stemming as well as Lemmatization. Many times pe
Some treat these two as same.
Actually, lemmatization is preferred over Stemming because lemmatization does mor

'''
# TASK 4: LEMMATIZER
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"))

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

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

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

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

In [20]:
```
# TASK 5: CHINESE SEGMENTATION USING JIEBA
#Chinese word segmentation module.

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))
```

```
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 [13]:
```
#Task : Importing WEBTEXT CORPUS and Access Data
import nltk
nltk.download('webtext')
from nltk.corpus import webtext
webtext.fileids()

for fileid in webtext.fileids():
    print(fileid, webtext.raw(fileid)[:])
```

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

In [14]:
```
# Task 5: Frequency Distribution of words in a text
text1 = '''1962 Tour de France was the 49th edition of the Tour de France, one of
fd = nltk.FreqDist(text1.split())
```

In [15]:
```
fd
```

Out[15]:
```
FreqDist({'the': 6, 'of': 5, 'Tour': 4, 'de': 3, 'was': 3, 'in': 3, 'and': 3,
'stages,': 2, 'on': 2, 'his': 2, ...})
```

In [16]:
```
# Task 6. Conditional Frequency Distribution of words in a text
# tells us how many 2 letter words or 3 letter words
from nltk.probability import ConditionalFreqDist
cfd = ConditionalFreqDist((len(word), word) for word in text1.split())
cfd[3]
```

Out[16]:
```
FreqDist({'the': 6, 'was': 3, 'and': 3, 'his': 2, 'one': 1, 'The': 1, 'mi)': 1,
'two': 1, 'des': 1, 'won': 1, ...})
```

In [17]:
```
cfd[6]
```

Out[17]:
```
FreqDist({'France': 1, 'Tours.': 1, '(2,656': 1, 'stages': 1, 'years,': 1, 'tea
ms.': 1, 'placed': 1, 'third,': 1, 'behind': 1})
```

In [18]:
```
import nltk
from nltk.corpus import webtext
from nltk.probability import FreqDist
from wordcloud import WordCloud
import matplotlib.pyplot as plt

nltk.download('webtext')
wt_words = webtext.words('C:/Users/Roshan/SNLP/testing.txt')  # Sample data
data_analysis = nltk.FreqDist(wt_words)

filter_words = dict([(m, n) for m, n in data_analysis.items() if len(m) > 3])

wcloud = WordCloud().generate_from_frequencies(filter_words)

# Plotting the wordcloud
plt.imshow(wcloud, interpolation="bilinear")

plt.axis("off")
(-0.5, 399.5, 199.5, -0.5)
plt.show()
```

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

In [19]:
```
# TASK 6: BASIC TEXT PROCESSING PIPELINE
import nltk
'''Punkt Sentence Tokenizer. This tokenizer divides a text into a list of sentenc
by using an unsupervised algorithm to build a model for abbreviation words, collo
It must be trained on a large collection of plaintext in the target language befo

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
import nltk
sent = "Become an expert in NLP"
words = nltk.word_tokenize(sent)
print(words)

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 del
count=0;
for text in texts:
    '''Tokenization is the process of tokenizing or splitting a string, text into
    One can think of token as parts like a word is a token in a sentence, and a s
    How sent_tokenize works ? The sent_tokenize function uses an instance of Punk
    sentences = nltk.sent_tokenize(text)
    for sentence in sentences:
        '''NLTK provides a function called word_tokenize() for splitting strings
        It splits tokens based on white space and punctuation.
        For example, commas and periods are taken as separate tokens'''
        words = nltk.word_tokenize(sentence)
        print(words)
        tagged = nltk.pos_tag(words)
        print(tagged)
num_words = [len(sentence.split()) for sentence in texts]
print('total word',num_words)
```

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

['Become', 'an', 'expert', 'in', 'NLP']
['The', 'only', 'true', 'wisdom', 'is', 'in', 'knowin', "'", 'you', 'know',
'nothing', '.']
[('The', 'DT'), ('only', 'JJ'), ('true', 'JJ'), ('wisdom', 'NN'), ('is', 'VB
Z'), ('in', 'IN'), ('knowin', 'NN'), ("'", "''"), ('you', 'PRP'), ('know', 'V
BP'), ('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', 'po
ets', 'to', 'write', 'their', 'poetry', ',', 'but', 'a', 'kind', 'of', 'ins',
'.']
[('I', 'PRP'), ('decided', 'VBD'), ('that', 'IN'), ('it', 'PRP'), ('was', 'VB
```

```
D'), ('not', 'RB'), ('wisdom', 'JJ'), ('that', 'IN'), ('enabled', 'VBD'), ('p
oets', 'NNS'), ('to', 'TO'), ('write', 'VB'), ('their', 'PRP$'), ('poetry',
'NN'), (',', ','), ('but', 'CC'), ('a', 'DT'), ('kind', 'NN'), ('of', 'IN'),
('ins', 'NNS'), ('.', '.')]
['or', 'inspiration', ',', 'such', 'as', 'you', 'find', 'in', 'seers', 'and',
'prophets', 'who', 'deliver', 'all', 'their', 'sublime', 'messages', 'withou
t', 'knowing', 'in', 'the', 'least', 'what', 'they', 'mean', '.']
[('or', 'CC'), ('inspiration', 'NN'), (',', ','), ('such', 'JJ'), ('as', 'I
N'), ('you', 'PRP'), ('find', 'VBP'), ('in', 'IN'), ('seers', 'NNS'), ('and',
'CC'), ('prophets', 'NNS'), ('who', 'WP'), ('deliver', 'VBP'), ('all', 'DT'),
('their', 'PRP$'), ('sublime', 'NN'), ('messages', 'NNS'), ('without', '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', 'TO'),
('seem', 'VB'), ('.', '.')]
['Wonder', 'is', 'the', 'beginning', 'of', 'wisdom', '.']
[('Wonder', 'NNP'), ('is', 'VBZ'), ('the', 'DT'), ('beginning', 'NN'), ('of',
'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'), ('for',
'IN'), ('blessings', 'NNS'), ('in', 'IN'), ('general', 'JJ'), (',', ','), ('f
or', 'IN'), ('God', 'NNP'), ('knows', 'VBZ'), ('best', 'JJS'), ('what', 'W
P'), ('is', 'VBZ'), ('good', 'JJ'), ('for', 'IN'), ('us', 'PRP'), ('.', '.')]
total word [100]
```