In [1]:
```
import os
import nltk
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml (https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml)
```

```
True
```

In [2]:
```
import nltk.corpus
```

In [3]:
```
# we will see what is mean by corpora and what all are availabel in nltk python library
#print(os.listdir(nltk.data.find('corpora')))

#you get a lot of file , some of have some textual document, different function associated with that functi
#for our example i will lets take consideration as brown & we will understand what exactly nlp can do
```

In [4]:
```
from nltk.corpus import brown
brown.words()
```

```
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

In [5]:
```
nltk.corpus.brown.fileids()
```

```
['ca01',
 'ca02',
 'ca03',
 'ca04',
 'ca05',
 'ca06',
 'ca07',
 'ca08',
 'ca09',
 'ca10',
 'ca11',
 'ca12',
 'ca13',
 'ca14',
 'ca15',
 'ca16',
 'ca17',
 'ca18',
 'ca19',
 'ca20',
 'ca21',
 'ca22',
 'ca23',
 'ca24'
```

In [6]:
```
nltk.corpus.gutenberg
```

```
<PlaintextCorpusReader in 'C:\\Users\\srava\\AppData\\Roaming\\nltk_data\\corpora\\gutenberg'>
```

In [7]:
```
nltk.corpus.gutenberg.fileids()
```

```
['austen-emma.txt',
 'austen-persuasion.txt',
 'austen-sense.txt',
 'bible-kjv.txt',
 'blake-poems.txt',
 'bryant-stories.txt',
 'burgess-busterbrown.txt',
 'carroll-alice.txt',
 'chesterton-ball.txt',
 'chesterton-brown.txt',
 'chesterton-thursday.txt',
 'edgeworth-parents.txt',
 'melville-moby_dick.txt',
 'milton-paradise.txt',
 'shakespeare-caesar.txt',
 'shakespeare-hamlet.txt',
 'shakespeare-macbeth.txt',
 'whitman-leaves.txt']
```

In [8]:
```
# we can also create our own words
AI = '''Artificial Intelligence refers to the intelligence of machines. This is in contrast to the natural i
humans and animals. With Artificial Intelligence, machines perform functions such as learning, planning, rea
problem-solving. Most noteworthy, Artificial Intelligence is the simulation of human intelligence by machine
It is probably the fastest-growing development in the World of technology and innovation. Furthermore, many
AI could solve major challenges and crisis situations.'''
```

In [9]:
```
AI
```

```
'Artificial Intelligence refers to the intelligence of machines. This is in contrast to the natural intelligence of \nhumans and animals.
With Artificial Intelligence, machines perform functions such as learning, planning, reasoning and \nproblem-solving. Most noteworthy, Art
ificial Intelligence is the simulation of human intelligence by machines. \nIt is probably the fastest-growing development in the World of
technology and innovation. Furthermore, many experts believe\nAI could solve major challenges and crisis situations.'
```

In [10]:
```
type(AI)
```

```
str
```

Word Tokenize

In [11]:
```
from nltk.tokenize import word_tokenize
```

In [12]:

```
AI_Tokens = word_tokenize(AI) #tokenize refers to every single word in the paragraph Word  == Tokenize
AI_Tokens
```

```
['Artificial',
 'Intelligence',
 'refers',
 'to',
 'the',
 'intelligence',
 'of',
 'machines',
 '.',
 'This',
 'is',
 'in',
 'contrast',
 'to',
 'the',
 'natural',
 'intelligence',
 'of',
 'humans',
 'and',
 'animals',
 '.',
 'With',
 'Artificial',
 'Intelligence',
 ',',
 'machines',
 'perform',
 'functions',
 'such',
 'as',
 'learning',
 ',',
 'planning',
 ',',
 'reasoning',
 'and',
 'problem-solving',
 '.',
 'Most',
 'noteworthy',
```

```
 ',',
 'Artificial',
 'Intelligence',
 'is',
 'the',
 'simulation',
 'of',
 'human',
 'intelligence',
 'by',
 'machines',
 '.',
 'It',
 'is',
 'probably',
 'the',
 'fastest-growing',
 'development',
 'in',
 'the',
 'World',
 'of',
 'technology',
 'and',
 'innovation',
 '.',
 'Furthermore',
 ',',
 'many',
 'experts',
 'believe',
 'AI',
 'could',
 'solve',
 'major',
 'challenges',
 'and',
 'crisis',
 'situations',
 '.']
```

In [13]:
```
len(AI_Tokens)
```

```
81
```

SENT Tokenize

In [15]:
```
from nltk.tokenize import sent_tokenize
```

In [17]:
```
AI_sent = sent_tokenize(AI) #sent_tokenize gives the each sentence of the paragaraph(totol number of senten
AI_sent
```

```
['Artificial Intelligence refers to the intelligence of machines.',
 'This is in contrast to the natural intelligence of \nhumans and animals.',
 'With Artificial Intelligence, machines perform functions such as learning, planning, reasoning and \nproblem-solving.',
 'Most noteworthy, Artificial Intelligence is the simulation of human intelligence by machines.',
 'It is probably the fastest-growing development in the World of technology and innovation.',
 'Furthermore, many experts believe\nAI could solve major challenges and crisis situations.']
```

In [18]:
```
len(AI_sent)
```

```
6
```

In [19]:
```
AI
```

```
'Artificial Intelligence refers to the intelligence of machines. This is in contrast to the natural intelligence of \nhumans and animals. With Artificial Intelligence, machines perform functions such as learning, planning, reasoning and \nproblem-solving. Most noteworthy, Artificial Intelligence is the simulation of human intelligence by machines. \nIt is probably the fastest-growing development in the World of technology and innovation. Furthermore, many experts believe\nAI could solve major challenges and crisis situations.'
```

blankline_tokenize

In [20]:
```
from nltk.tokenize import blankline_tokenize # GiVE YOU HOW MANY PARAGRAPH
AI_blank = blankline_tokenize(AI)
AI_blank
```

```
['Artificial Intelligence refers to the intelligence of machines. This is in contrast to the natural intelligence of \nhumans and animals.
With Artificial Intelligence, machines perform functions such as learning, planning, reasoning and \nproblem-solving. Most noteworthy, Art
ificial Intelligence is the simulation of human intelligence by machines. \nIt is probably the fastest-growing development in the World of
technology and innovation. Furthermore, many experts believe\nAI could solve major challenges and crisis situations.']
```

In [21]:
```
len(AI_blank)
```

```
1
```

Types of tokenization

In [22]:
```
# NEXT WE WILL SEE HOW WE WILL USE UNI-GRAM,BI-GRAM,TRI-GRAM USING NLTK
from nltk.util import bigrams,trigrams,ngrams
```

In [23]:
```
string ='the best and most beautifull thing in the world cannot be seen or even touched,they must be felt wi
quotes_tokens = nltk.word_tokenize(string)
```

In [24]:
```
quotes_tokens # we saw that how many words are there in the quotes_tokens
```

```
['the',
 'best',
 'and',
 'most',
 'beautifull',
 'thing',
 'in',
 'the',
 'world',
 'can',
 'not',
 'be',
 'seen',
 'or',
 'even',
 'touched',
 ',',
 'they',
 'must',
 'be',
 'felt',
 'with',
 'heart']
```

In [25]:
```
len(quotes_tokens)
```

```
23
```

In [28]:
```
quotes_bigrams = list(nltk.bigrams(quotes_tokens)) #bigram whic indicates 2 words == we get two two words c
#following the first 2nd  ending word will start in the secound line..
quotes_bigrams
```

```
[('the', 'best'),
 ('best', 'and'),
 ('and', 'most'),
 ('most', 'beautifull'),
 ('beautifull', 'thing'),
 ('thing', 'in'),
 ('in', 'the'),
 ('the', 'world'),
 ('world', 'can'),
 ('can', 'not'),
 ('not', 'be'),
 ('be', 'seen'),
 ('seen', 'or'),
 ('or', 'even'),
 ('even', 'touched'),
 ('touched', ','),
 (',', 'they'),
 ('they', 'must'),
 ('must', 'be'),
 ('be', 'felt'),
 ('felt', 'with'),
 ('with', 'heart')]
```

In [29]:
```
len(quotes_bigrams)
```

```
22
```

In [31]:
```
print(quotes_tokens)
print(len(quotes_tokens))
```

```
['the', 'best', 'and', 'most', 'beautifull', 'thing', 'in', 'the', 'world', 'can', 'not', 'be', 'seen', 'or', 'even', 'touched', ',', 'the
y', 'must', 'be', 'felt', 'with', 'heart']
23
```

In [33]:
```
quotes_trigrams = list(nltk.trigrams(quotes_tokens))
quotes_trigrams
```

```
[('the', 'best', 'and'),
 ('best', 'and', 'most'),
 ('and', 'most', 'beautifull'),
 ('most', 'beautifull', 'thing'),
 ('beautifull', 'thing', 'in'),
 ('thing', 'in', 'the'),
 ('in', 'the', 'world'),
 ('the', 'world', 'can'),
 ('world', 'can', 'not'),
 ('can', 'not', 'be'),
 ('not', 'be', 'seen'),
 ('be', 'seen', 'or'),
 ('seen', 'or', 'even'),
 ('or', 'even', 'touched'),
 ('even', 'touched', ','),
 ('touched', ',', 'they'),
 (',', 'they', 'must'),
 ('they', 'must', 'be'),
 ('must', 'be', 'felt'),
 ('be', 'felt', 'with'),
 ('felt', 'with', 'heart')]
```

In [34]:
```
len(quotes_trigrams)
```

```
21
```

In [35]:
```
quotes_ngrams = list(nltk.ngrams(quotes_tokens , 4))
quotes_ngrams
#it has given n-gram of length 4 , we need to give the lenght of ngram
```

```
[('the', 'best', 'and', 'most'),
 ('best', 'and', 'most', 'beautifull'),
 ('and', 'most', 'beautifull', 'thing'),
 ('most', 'beautifull', 'thing', 'in'),
 ('beautifull', 'thing', 'in', 'the'),
 ('thing', 'in', 'the', 'world'),
 ('in', 'the', 'world', 'can'),
 ('the', 'world', 'can', 'not'),
 ('world', 'can', 'not', 'be'),
 ('can', 'not', 'be', 'seen'),
 ('not', 'be', 'seen', 'or'),
 ('be', 'seen', 'or', 'even'),
 ('seen', 'or', 'even', 'touched'),
 ('or', 'even', 'touched', ','),
 ('even', 'touched', ',', 'they'),
 ('touched', ',', 'they', 'must'),
 (',', 'they', 'must', 'be'),
 ('they', 'must', 'be', 'felt'),
 ('must', 'be', 'felt', 'with'),
 ('be', 'felt', 'with', 'heart')]
```

In [37]:
```
quotes_ngrams_1 = list(nltk.ngrams(quotes_tokens , 5))
quotes_ngrams_1
```

```
[('the', 'best', 'and', 'most', 'beautifull'),
 ('best', 'and', 'most', 'beautifull', 'thing'),
 ('and', 'most', 'beautifull', 'thing', 'in'),
 ('most', 'beautifull', 'thing', 'in', 'the'),
 ('beautifull', 'thing', 'in', 'the', 'world'),
 ('thing', 'in', 'the', 'world', 'can'),
 ('in', 'the', 'world', 'can', 'not'),
 ('the', 'world', 'can', 'not', 'be'),
 ('world', 'can', 'not', 'be', 'seen'),
 ('can', 'not', 'be', 'seen', 'or'),
 ('not', 'be', 'seen', 'or', 'even'),
 ('be', 'seen', 'or', 'even', 'touched'),
 ('seen', 'or', 'even', 'touched', ','),
 ('or', 'even', 'touched', ',', 'they'),
 ('even', 'touched', ',', 'they', 'must'),
 ('touched', ',', 'they', 'must', 'be'),
 (',', 'they', 'must', 'be', 'felt'),
 ('they', 'must', 'be', 'felt', 'with'),
 ('must', 'be', 'felt', 'with', 'heart')]
```

In [38]:
```
quotes_ngrams = list(nltk.ngrams(quotes_tokens, 9))
quotes_ngrams
```

```
[('the', 'best', 'and', 'most', 'beautifull', 'thing', 'in', 'the', 'world'),
 ('best', 'and', 'most', 'beautifull', 'thing', 'in', 'the', 'world', 'can'),
 ('and', 'most', 'beautifull', 'thing', 'in', 'the', 'world', 'can', 'not'),
 ('most', 'beautifull', 'thing', 'in', 'the', 'world', 'can', 'not', 'be'),
 ('beautifull', 'thing', 'in', 'the', 'world', 'can', 'not', 'be', 'seen'),
 ('thing', 'in', 'the', 'world', 'can', 'not', 'be', 'seen', 'or'),
 ('in', 'the', 'world', 'can', 'not', 'be', 'seen', 'or', 'even'),
 ('the', 'world', 'can', 'not', 'be', 'seen', 'or', 'even', 'touched'),
 ('world', 'can', 'not', 'be', 'seen', 'or', 'even', 'touched', ','),
 ('can', 'not', 'be', 'seen', 'or', 'even', 'touched', ',', 'they'),
 ('not', 'be', 'seen', 'or', 'even', 'touched', ',', 'they', 'must'),
 ('be', 'seen', 'or', 'even', 'touched', ',', 'they', 'must', 'be'),
 ('seen', 'or', 'even', 'touched', ',', 'they', 'must', 'be', 'felt'),
 ('or', 'even', 'touched', ',', 'they', 'must', 'be', 'felt', 'with'),
 ('even', 'touched', ',', 'they', 'must', 'be', 'felt', 'with', 'heart')]
```

In [40]:
```
quotes_ngrams = list(nltk.ngrams(quotes_tokens,15))
quotes_ngrams
```

```
[('the',
  'best',
  'and',
  'most',
  'beautifull',
  'thing',
  'in',
  'the',
  'world',
  'can',
  'not',
  'be',
  'seen',
  'or',
  'even'),
 ('best',
  'and',
  'most',
  'beautifull',
  'thing',
  'in',
  'the',
  'world',
  'can',
```

Stemming -- Root form of word

In [41]:
```
# Next we need to make some changes in tokens and that is called as stemming, stemming will gives you root j
# also we will see some root form of the word & limitation of the word

#porter-stemmer
from nltk.stem import PorterStemmer
pst = PorterStemmer()
```

In [42]:
```
pst.stem('having') #stem will gives you the root form of the word
```

```
'have'
```

In [43]:
```
pst.stem('affection')
```

```
'affect'
```

In [44]:
```
pst.stem('playing')
```

```
'play'
```

In [45]:
```
pst.stem('give')
```

```
'give'
```

In [48]:
```
words_to_stem=['give','giving','given','gave']
for words in words_to_stem:
    print(words+  ':' + pst.stem(words))
```

```
give:give
giving:give
given:given
gave:gave
```

In [49]:
```
words_to_stem=['give','giving','given','gave','thinking', 'loving', 'final', 'finalized', 'finally']
for words in words_to_stem:
    print(words+ ':' +pst.stem(words))

#in porterstemmer removes ing and replaces with e
```

```
give:give
giving:give
given:given
gave:gave
thinking:think
loving:love
final:final
finalized:final
finally:final
```

lancasterstemmer

In [50]:

```
#another stemmer known as lencastemmer stemmer and lets see what the different we will get hear
#stem the same thing using lencastemmer
from nltk.stem import LancasterStemmer
lst = LancasterStemmer()
for words in words_to_stem:
    print(words + ':' + lst.stem(words))

# lancasterstemmer is more aggresive then the porterstemmer
```

```
give:giv
giving:giv
given:giv
gave:gav
thinking:think
loving:lov
final:fin
finalized:fin
finally:fin
```

In [51]:
```
words_to_stem=['give','giving','given','gave','thinking', 'loving', 'final', 'finalized', 'finally']
for words in words_to_stem:
    print(words+ ':' +pst.stem(words))
```

```
give:give
giving:give
given:given
gave:gave
thinking:think
loving:love
final:final
finalized:final
finally:final
```

In [52]:
```
#we have another stemmer called as snowball stemmer lets see about this snowball stemmer

from nltk.stem import SnowballStemmer
sbst = SnowballStemmer('english')
for words in words_to_stem:
    print(words+ ':' +sbst.stem(words))

#snowball stemmer is same as portstemmer
#different type of stemmer used based on different type of task
#if you want to see how many type of giv has occured then we will see the lancaster stemmer
```

```
give:give
giving:give
given:given
gave:gave
thinking:think
loving:love
final:final
finalized:final
finally:final
```

In [53]:
```
#sometime stemming does not work & lets say e.g - fish,fishes & fishing all of them belongs to root word fi
#one hand stemming will cut the end & lemmatization will take into the morphological analysis of the word

from nltk.stem import wordnet
from nltk.stem import WordNetLemmatizer
word_lem = WordNetLemmatizer()

#Hear we are going to wordnet dictionary & we are going to import the wordnet lematizer
```

In [54]:
```
words_to_stem
```

```
['give',
 'giving',
 'given',
 'gave',
 'thinking',
 'loving',
 'final',
 'finalized',
 'finally']
```

In [55]:
```
#word_lem.lemmatize('corpora') #we get output as corpus

#refers to a collection of texts. Such collections may be formed of a single language of texts, or can span

for words in words_to_stem:
    print(words+ ':' +word_lem.lemmatize(words))
```

```
give:give
giving:giving
given:given
gave:gave
thinking:thinking
loving:loving
final:final
finalized:finalized
finally:finally
```

In [56]:
```
pst.stem('final')
```

```
'final'
```

In [57]:
```
lst.stem('finally')
```

```
'fin'
```

In [58]:
```
sbst.stem('finalized')
```

```
'final'
```

In [59]:
```
lst.stem('final')
```

```
'fin'
```

In [60]:
```
lst.stem('finalized')
```

```
'fin'
```

In [61]:
```
# there is other concept called POS (part of speech) which deals with subject, noun, pronoun but before of t
# STOPWORDS = i, is, as,at, on, about & nltk has their own list of stopewords

from nltk.corpus import stopwords
```

In [62]:
```
stopwords.words('english')
```

```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
 'himself',
 'she',
 "she's",
 'her'
```

In [64]:
```
len(stopwords.words('spanish'))
```

```
313
```

In [65]:
```
stopwords.words('french')
```

```
 'aurions',
 'auriez',
 'auraient',
 'avais',
 'avait',
 'avions',
 'aviez',
 'avaient',
 'eut',
 'eûmes',
 'eûtes',
 'eurent',
 'aie',
 'aies',
 'ait',
 'ayons',
 'ayez',
 'aient',
 'eusse',
 'eusses',
 'eût',
 'eussions',
 'eussiez',
 'eussent']
```

In [66]:
```
len(stopwords.words('french'))
```

```
157
```

In [67]:
```
print(stopwords.words('german'))
len(stopwords.words('german') )
```

```
['aber', 'alle', 'allem', 'allen', 'aller', 'alles', 'als', 'also', 'am', 'an', 'ander', 'andere', 'anderem', 'anderen', 'anderer', 'ander
es', 'anderm', 'andern', 'anderr', 'anders', 'auch', 'auf', 'aus', 'bei', 'bin', 'bis', 'bist', 'da', 'damit', 'dann', 'der', 'den', 'de
s', 'dem', 'die', 'das', 'dass', 'daß', 'derselbe', 'derselben', 'denselben', 'desselben', 'demselben', 'dieselbe', 'dieselben', 'dasselb
e', 'dazu', 'dein', 'deine', 'deinem', 'deinen', 'deiner', 'deines', 'denn', 'derer', 'dessen', 'dich', 'dir', 'du', 'dies', 'diese', 'die
sem', 'diesen', 'dieser', 'dieses', 'doch', 'dort', 'durch', 'ein', 'eine', 'einem', 'einen', 'einer', 'eines', 'einig', 'einige', 'einige
m', 'einigen', 'einiger', 'einiges', 'einmal', 'er', 'ihn', 'ihm', 'es', 'etwas', 'euer', 'eure', 'eurem', 'euren', 'eurer', 'eures', 'fü
r', 'gegen', 'gewesen', 'hab', 'habe', 'haben', 'hat', 'hatte', 'hatten', 'hier', 'hin', 'hinter', 'ich', 'mich', 'mir', 'ihr', 'ihre', 'i
hrem', 'ihren', 'ihrer', 'ihres', 'euch', 'im', 'in', 'indem', 'ins', 'ist', 'jede', 'jedem', 'jeden', 'jeder', 'jedes', 'jene', 'jenem',
'jenen', 'jener', 'jenes', 'jetzt', 'kann', 'kein', 'keine', 'keinem', 'keinen', 'keiner', 'keines', 'können', 'könnte', 'machen', 'man',
'manche', 'manchem', 'manchen', 'mancher', 'manches', 'mein', 'meine', 'meinem', 'meinen', 'meiner', 'meines', 'mit', 'muss', 'musste', 'n
ach', 'nicht', 'nichts', 'noch', 'nun', 'nur', 'ob', 'oder', 'ohne', 'sehr', 'sein', 'seine', 'seinem', 'seinen', 'seiner', 'seines', 'sel
bst', 'sich', 'sie', 'ihnen', 'sind', 'so', 'solche', 'solchem', 'solchen', 'solcher', 'solches', 'soll', 'sollte', 'sondern', 'sonst', 'ü
ber', 'um', 'und', 'uns', 'unsere', 'unserem', 'unseren', 'unser', 'unseres', 'unter', 'viel', 'vom', 'von', 'vor', 'während', 'war', 'war
en', 'warst', 'was', 'weg', 'weil', 'weiter', 'welche', 'welchem', 'welchen', 'welcher', 'welches', 'wenn', 'werde', 'werden', 'wie', 'wie
der', 'will', 'wir', 'wird', 'wirst', 'wo', 'wollen', 'wollte', 'würde', 'würden', 'zu', 'zum', 'zur', 'zwar', 'zwischen']
```

```
232
```

In [68]:
```
stopwords.words('hindi') # research phase
```

. . .

In [69]:
```
stopwords.words('marathi')
```

. . .

In [70]:
```
stopwords.words('telugu')
```

. . .

In [71]:
```
# first we need to compile from re module to create string that matched any digits or special character
import re
punctuation = re.compile(r'[-.?!,:;()|0-9]')

#now i am going to create to empty list and append the word without any punctuation & naming this as a post
```

In [72]:
```
punctuation
```

```
re.compile(r'[-.?!,:;()|0-9]', re.UNICODE)
```

In [74]:
```
AI
```

```
'Artificial Intelligence refers to the intelligence of machines. This is in contrast to the natural intelligence of \nhumans and animals. With Artificial Intelligence, machines perform functions such as learning, planning, reasoning and \nproblem-solving. Most noteworthy, Artificial Intelligence is the simulation of human intelligence by machines. \nIt is probably the fastest-growing development in the World of technology and innovation. Furthermore, many experts believe\nAI could solve major challenges and crisis situations.'
```

In [76]:
```
print(AI_Tokens)
```

```
['Artificial', 'Intelligence', 'refers', 'to', 'the', 'intelligence', 'of', 'machines', '.', 'This', 'is', 'in', 'contrast', 'to', 'the', 'natural', 'intelligence', 'of', 'humans', 'and', 'animals', '.', 'With', 'Artificial', 'Intelligence', ',', 'machines', 'perform', 'functions', 'such', 'as', 'learning', ',', 'planning', ',', 'reasoning', 'and', 'problem-solving', '.', 'Most', 'noteworthy', ',', 'Artificial', 'Intelligence', 'is', 'the', 'simulation', 'of', 'human', 'intelligence', 'by', 'machines', '.', 'It', 'is', 'probably', 'the', 'fastest-growing', 'development', 'in', 'the', 'World', 'of', 'technology', 'and', 'innovation', '.', 'Furthermore', ',', 'many', 'experts', 'believe', 'AI', 'could', 'solve', 'major', 'challenges', 'and', 'crisis', 'situations', '.']
```

In [77]:
```
len(AI_Tokens)
```

```
81
```

#POS [part of sppech] is always talking about grammaticaly type of the word called verbs, noun, adjective, proverb,

#how the word will function in grammatically within the sentence, a word can have more then one pos based on context in which it will use

#so lets see some pos tags & description, so pos tags are usualy used to descrie weather te word is used for noun,adjective,pronoun, propernoun, singular, plural, is it symbol or is it adverb

#in this slide we have so many tags along with their description with different tags

#this tags are beginning from coordinating conjunction to whadverb & lets understand about one of the example

#next we will see how we will implement this POS in our text

In [78]:
```
sent = 'kathy is a natural when it comes to drawing'
sent_tokens = word_tokenize(sent)
sent_tokens
```

```
['kathy', 'is', 'a', 'natural', 'when', 'it', 'comes', 'to', 'drawing']
```

In [79]:
```
for token in sent_tokens:
    print(nltk.pos_tag([token]))
```

```
[('kathy', 'NN')]
[('is', 'VBZ')]
[('a', 'DT')]
[('natural', 'JJ')]
[('when', 'WRB')]
[('it', 'PRP')]
[('comes', 'VBZ')]
[('to', 'TO')]
[('drawing', 'VBG')]
```

In [80]:
```
sent2 = 'john is eating a delicious cake'
sent2_tokens = word_tokenize(sent2)
for token in sent2_tokens:
    print(nltk.pos_tag([token]))
```

```
[('john', 'NN')]
[('is', 'VBZ')]
[('eating', 'VBG')]
[('a', 'DT')]
[('delicious', 'JJ')]
[('cake', 'NN')]
```

In [81]:
```
# Another concept of POS is called NER ( NAMED ENTITIY RECOGNITION ), NER is the process of detecting name s
# there are 3 phases of NER - ( 1ST PHASE IS - NOUN PHRASE EXTRACTION OR NOUN PHASE IDENTIFICATION - This st
# 2nd step we have phrase classification - this is the classification where all the extracted nouns & phrase
# some times entity are misclassification
# so if you are use NER in python then you need to import NER_CHUNK from nltk library
```

In [82]:
```
from nltk import ne_chunk
```

In [83]:
```
NE_sent = 'The US president stays in the WHITEHOUSE '
```

```
# IN NLTK also we have syntax- set of rules,principals & process
# lets understand set of rules & that will indicates the syntax tree & in the real time also you have
build this type of tree from the sentenses

# now lets understand the important concept called CHUNKING using the sentence structure
# chunking means grouping of words into chunks & lets understand the example of chunking
```

```
# chunking will help to easy process the data
```

In [85]:
```
NE_tokens = word_tokenize(NE_sent)
#after tokenize need to add the pos tags

NE_tokens
```

```
['The', 'US', 'president', 'stays', 'in', 'the', 'WHITEHOUSE']
```

In [86]:
```
NE_tags = nltk.pos_tag(NE_tokens)
NE_tags
```

```
[('The', 'DT'),
 ('US', 'NNP'),
 ('president', 'NN'),
 ('stays', 'NNS'),
 ('in', 'IN'),
 ('the', 'DT'),
 ('WHITEHOUSE', 'NNP')]
```

In [87]:
```
#we are passin the NE_NER into ne_chunks function and lets see the outputs


NE_NER = ne_chunk(NE_tags)
print(NE_NER)
```

```
(S
  The/DT
  (GSP US/NNP)
  president/NN
  stays/NNS
  in/IN
  the/DT
  (ORGANIZATION WHITEHOUSE/NNP))
```

In [88]:
```
new = 'the big cat ate the little mouse who was after fresh cheese'
new_tokens = nltk.pos_tag(word_tokenize(new))
new_tokens
```

```
[('the', 'DT'),
 ('big', 'JJ'),
 ('cat', 'NN'),
 ('ate', 'VBD'),
 ('the', 'DT'),
 ('little', 'JJ'),
 ('mouse', 'NN'),
 ('who', 'WP'),
 ('was', 'VBD'),
 ('after', 'IN'),
 ('fresh', 'JJ'),
 ('cheese', 'NN')]
```

In [90]:
```
pip install wordcloud
```

```
Collecting wordcloudNote: you may need to restart the kernel to use updated packages.

  Obtaining dependency information for wordcloud from https://files.pythonhosted.org/packages/34/ac/72a4e42e76bf549dfd91791a6b10a9832f046c1d48b5e778be9ec012aa47/wordcloud-1.9.2-cp311-cp311-win_amd64.whl.metadata (https://files.pythonhosted.org/packages/34/ac/72a4e42e76bf549dfd91791a6b10a9832f046c1d48b5e778be9ec012aa47/wordcloud-1.9.2-cp311-cp311-win_amd64.whl.metadata)
  Downloading wordcloud-1.9.2-cp311-cp311-win_amd64.whl.metadata (3.4 kB)
Requirement already satisfied: numpy>=1.6.1 in c:\users\srava\anaconda3\lib\site-packages (from wordcloud) (1.24.3)
Requirement already satisfied: pillow in c:\users\srava\anaconda3\lib\site-packages (from wordcloud) (9.4.0)
Requirement already satisfied: matplotlib in c:\users\srava\anaconda3\lib\site-packages (from wordcloud) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\srava\anaconda3\lib\site-packages (from matplotlib->wordcloud) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\srava\anaconda3\lib\site-packages (from matplotlib->wordcloud) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\srava\anaconda3\lib\site-packages (from matplotlib->wordcloud) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\srava\anaconda3\lib\site-packages (from matplotlib->wordcloud) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\srava\anaconda3\lib\site-packages (from matplotlib->wordcloud) (23.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\srava\anaconda3\lib\site-packages (from matplotlib->wordcloud) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\srava\anaconda3\lib\site-packages (from matplotlib->wordcloud) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\srava\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib->wordcloud) (1.16.0)
Downloading wordcloud-1.9.2-cp311-cp311-win_amd64.whl (151 kB)
   ---------------------------------------- 0.0/151.4 kB ? eta -:--:--
   -------- ------------------------------- 30.7/151.4 kB ? eta -:--:--
   ------------------------------- ------- 122.9/151.4 kB 1.4 MB/s eta 0:00:01
   ---------------------------------------- 151.4/151.4 kB 1.3 MB/s eta 0:00:00
Installing collected packages: wordcloud
Successfully installed wordcloud-1.9.2
```

In [91]:
```
# Libraries
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

In [92]:
```
# Create a list of word
text=("Python Python Python Matplotlib Matplotlib Seaborn Network Plot Violin Chart Pandas Datascience Wordc
```

In [93]:
```
text
```

```
'Python Python Python Matplotlib Matplotlib Seaborn Network Plot Violin Chart Pandas Datascience Wordcloud Spider Radar Parrallel Alpha Co
lor Brewer Density Scatter Barplot Barplot Boxplot Violinplot Treemap Stacked Area Chart Chart Visualization Dataviz Donut Pie Time-Series
Wordcloud Wordcloud Sankey Bubble'
```

In [94]:
```
# Create the wordcloud object
wordcloud = WordCloud(width=480, height=480, margin=0).generate(text)
```

In [95]:
```
# Display the generated image:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.margins(x=0, y=0)
plt.show()
```

In [ ]: