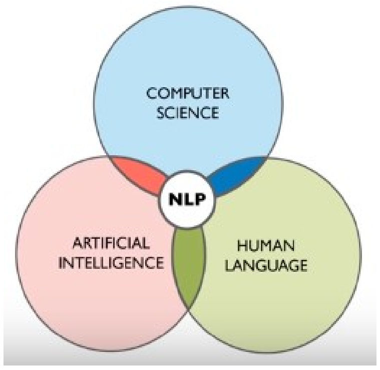
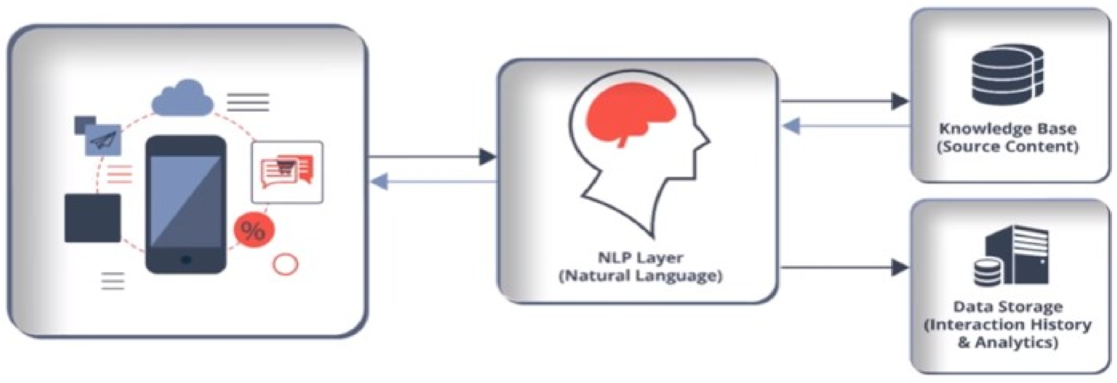
LAB -4

AIM: To implement sentimental analysis using NLTK.
DESCRIPTION:
Natural language processing is a computer science and artificial intelligence which deals with human languages.



The essence of Natural Language Processing lies in making computers understand the natural language. That's not an easy task though. Computers can understand the structured form of data like spreadsheets and the tables in the database, but human languages, texts, and voices form an unstructured category of data, and it gets difficult for the computer to understand it, and there arises the need for Natural Language Processing.

BASIC STRUCTURE OF NLP APPLICATION:



APPLICATIONS OF NLP:

- Sentiment Analysis
- Speech Recognition
- Chatbot
- Machine Translation
- Spell checking
- Keyword search
- Information Extraction

NLP Components:

1.Natural language understanding

- Mapping input into useful representations
- Analyzing different aspects of the language

2.Natural language generation :

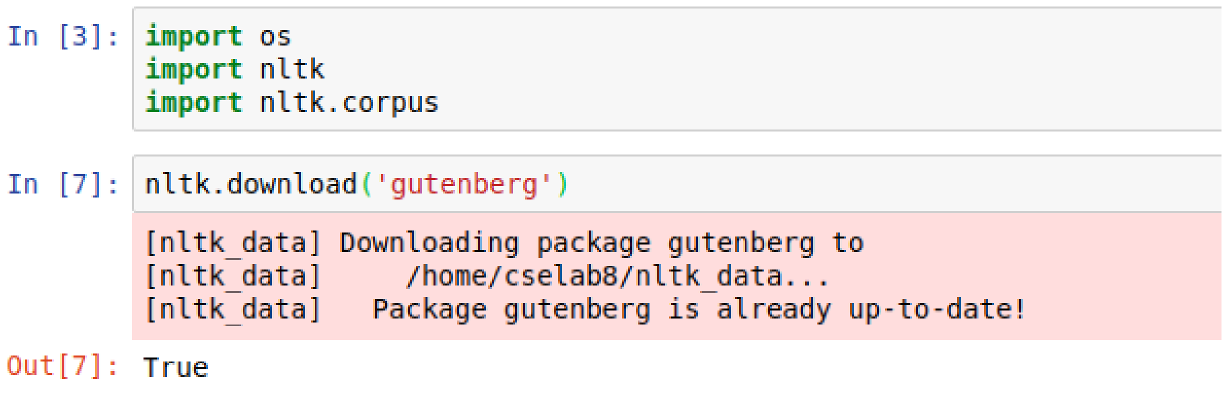
- Text planning
- Sentence planning
- Text realization

STEPS IN NLTK:

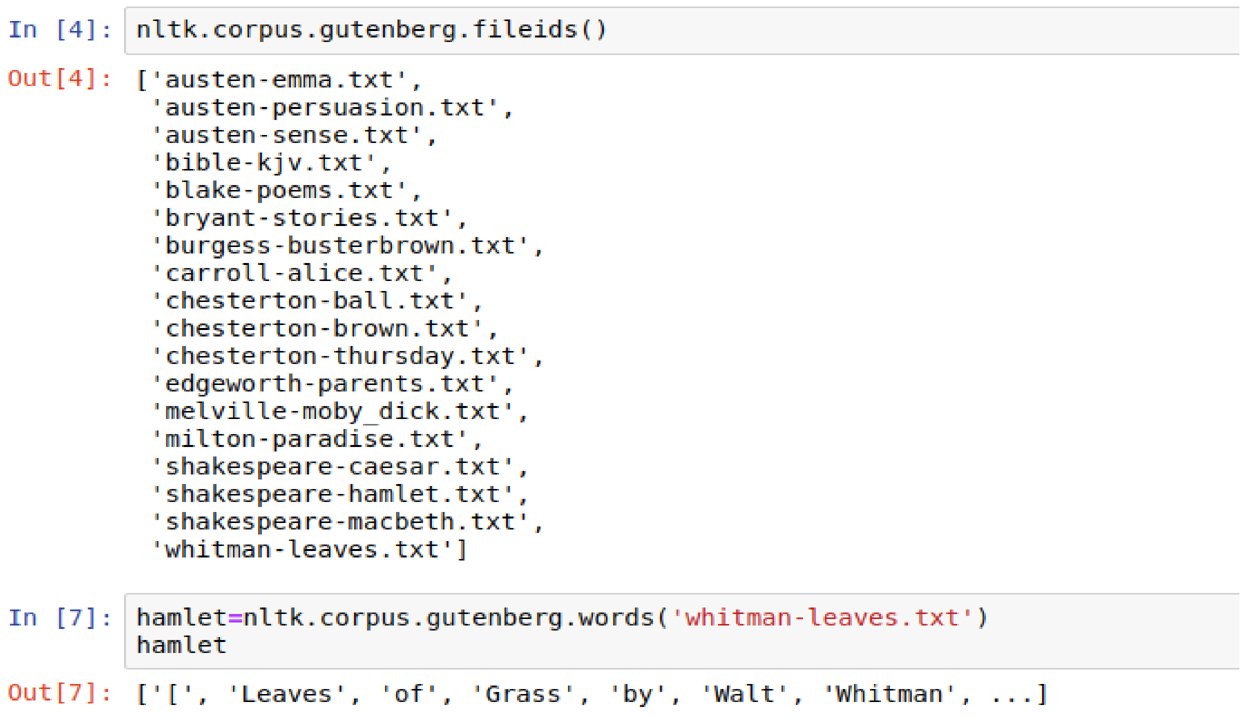
1.Tokenization:

- Break complex sentence into word.
- Understand the importance of each words with respect to the sentences.
- Produce a structural description on an input sentence.

Import all the libraries,



```
In [3]: import os
        import nltk
        import nltk.corpus

In [7]: nltk.download('gutenberg')

        [nltk_data] Downloading package gutenberg to
        [nltk_data]     /home/cselab8/nltk_data...
        [nltk_data]   Package gutenberg is already up-to-date!

Out[7]: True
```



```
In [4]: nltk.corpus.gutenberg.fileids()

Out[4]: ['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 [7]: hamlet=nltk.corpus.gutenberg.words('whitman-leaves.txt')
        hamlet

Out[7]: ['[', 'Leaves', 'of', 'Grass', 'by', 'Walt', 'Whitman', ...]
```

```
In [9]: for word in hamlet[:500]:
            print(word,sep=' ',end=' ')
```

```
[ Leaves of Grass by Walt Whitman 1855 ] Come , said my soul , Such verses for my Body let us write , ( for we are
one ,) That should I after return , Or , long , long hence , in other spheres , There to some group of mates the c
hants resuming , ( Tallying Earth ' s soil , trees , winds , tumultuous waves ,) Ever with pleas ' d smile I may k
eep on , Ever and ever yet the verses owning -- as , first , I here and now Signing for Soul and Body , set to the
m my name , Walt Whitman [ BOOK I . INSCRIPTIONS ] } One ' s - Self I Sing One ' s - self I sing , a simple separa
te person , Yet utter the word Democratic , the word En - Masse . Of physiology from top to toe I sing , Not physi
ognomy alone nor brain alone is worthy for the Muse , I say the Form complete is worthier far , The Female equally
with the Male I sing . Of Life immense in passion , pulse , and power , Cheerful , for freest action form ' d unde
r the laws divine , The Modern Man I sing . } As I Ponder ' d in Silence As I ponder ' d in silence , Returning up
on my poems , considering , lingering long , A Phantom arose before me with distrustful aspect , Terrible in beaut
y , age , and power , The genius of poets of old lands , As to me directing like flame its eyes , With finger poin
ting to many immortal songs , And menacing voice , What singest thou ? it said , Know ' st thou not there is hut o
ne theme for ever - enduring bards ? And that is the theme of War , the fortune of battles , The making of perfect
soldiers . Be it so , then I answer ' d , I too haughty Shade also sing war , and a longer and greater one than an
y , Waged in my book with varying fortune , with flight , advance and retreat , victory deferr ' d and wavering ,
( Yet methinks certain , or as good as certain , at the last ,) the field the world , For life and death , for the
Body and for the eternal Soul , Lo , I too am come , chanting the chant of battles , I above all promote brave sol
diers . } In Cabin ' d Ships at Sea In cabin ' d ships at sea , The boundless blue on every side expanding , With
whistling winds and music of the waves , the large imperious waves , Or some lone bark buoy ' d on the dense marin
e , Where joyous full of faith , spreading white sails , She cleaves
```

```
In [10]: a="""NLTK is intended to support research and teaching in NLP or closely related areas, including empirical linguis
```

```
In [11]: a
```

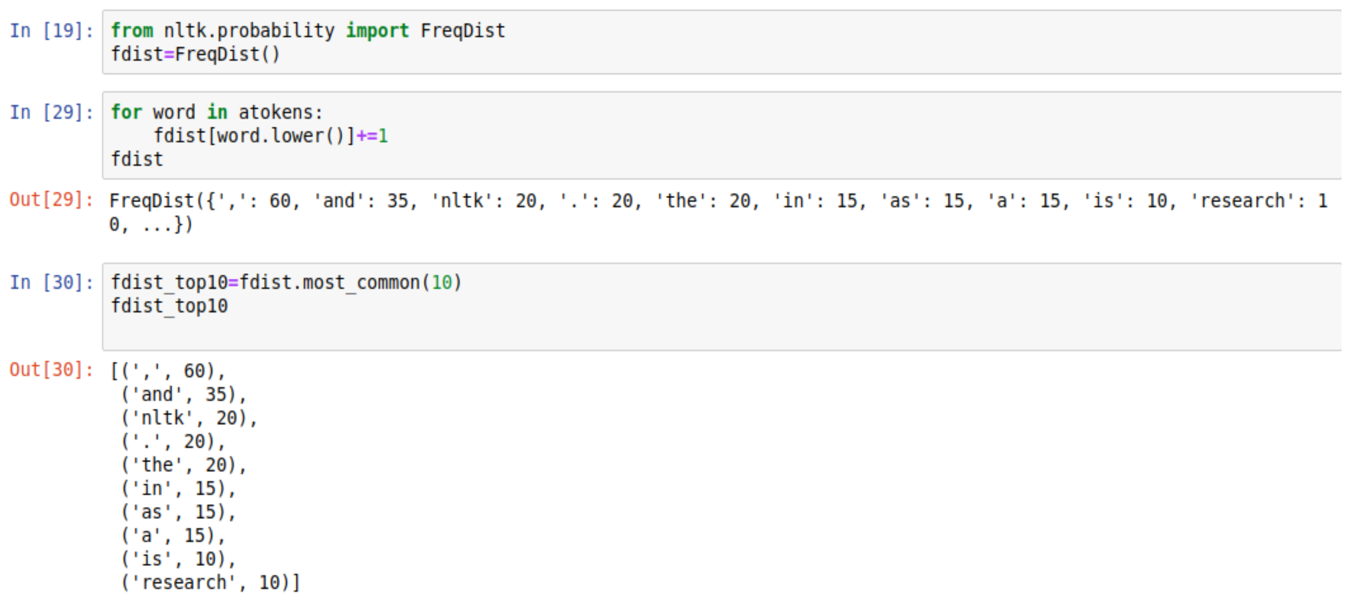
```
Out[11]: 'NLTK is intended to support research and teaching in NLP or closely related areas, including empirical linguistic
         s, cognitive science, artificial intelligence, information retrieval, and machine learning. NLTK has been used suc
         cessfully as a teaching tool, as an individual study tool, and as a platform for prototyping and building research
         systems. There are 32 universities in the US and 25 countries using NLTK in their courses. NLTK supports classific
         ation, tokenization, stemming, tagging, parsing, and semantic reasoning functionalities.NLTK includes graphical de
         monstrations and sample data. It is accompanied by a book that explains the underlying concepts behind the languag
         e processing tasks supported by the toolkit'
```

```
In [12]: type(a)

Out[12]: str
```

```
In [15]: from nltk.tokenize import word_tokenize
         import nltk
         nltk.download('punkt')

         [nltk_data] Downloading package punkt to /home/cselab8/nltk_data...
         [nltk_data]   Unzipping tokenizers/punkt.zip.

Out[15]: True
```

```
In [16]: atokens=word_tokenize(a)
         atokens

Out[16]: ['NLTK',
          'is',
          'intended',
          'to',
          'support',
          'research',
          'and',
          'teaching',
          'in',
          'NLP',
          'or',
          'closely',
          'related',
          'areas',
          ',',
          'including',
          'empirical',
          'linguistics',
          ',',
```

```
In [17]: len(atokens)

Out[17]: 118
```

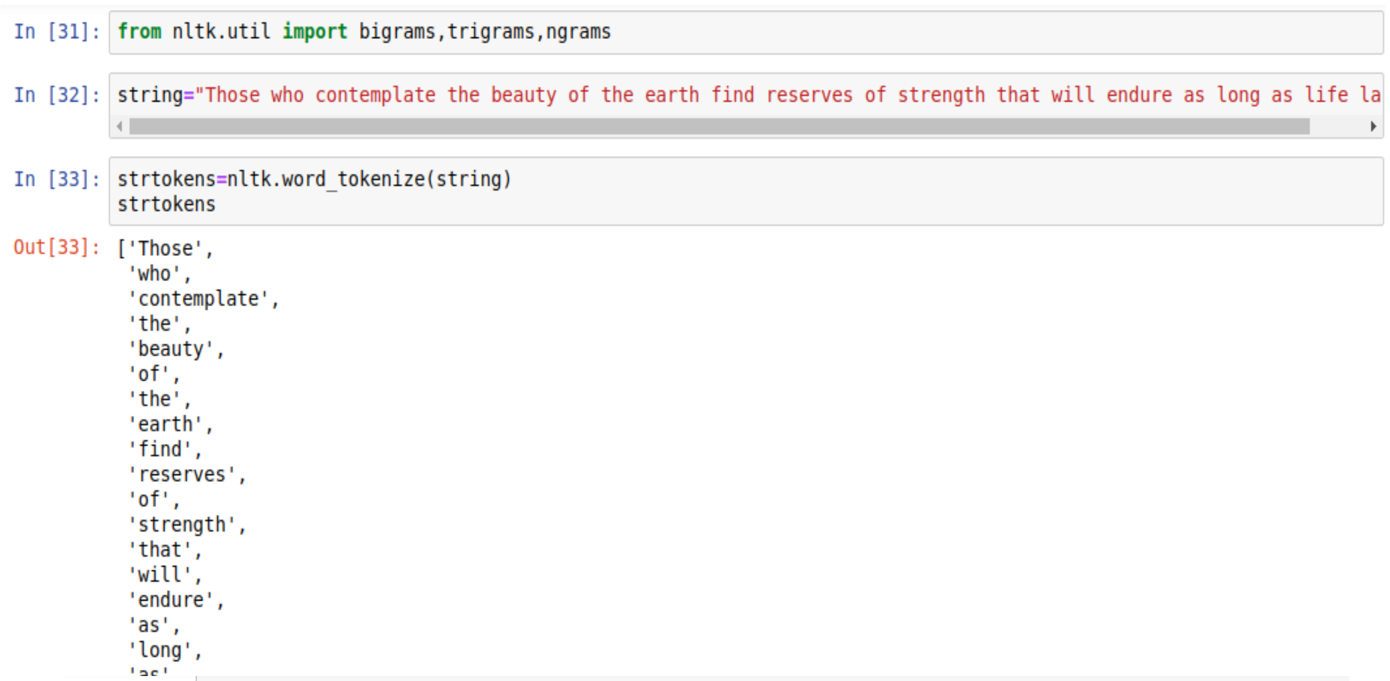


```
In [19]: from nltk.probability import FreqDist
         fdist=FreqDist()

In [29]: for word in atokens:
             fdist[word.lower()]+=1
         fdist

Out[29]: FreqDist({',': 60, 'and': 35, 'nltk': 20, '.': 20, 'the': 20, 'in': 15, 'as': 15, 'a': 15, 'is': 10, 'research': 1
         0, ...})

In [30]: fdist_top10=fdist.most_common(10)
         fdist_top10

Out[30]: [(',', 60),
          ('and', 35),
          ('nltk', 20),
          ('.', 20),
          ('the', 20),
          ('in', 15),
          ('as', 15),
          ('a', 15),
          ('is', 10),
          ('research', 10)]
```

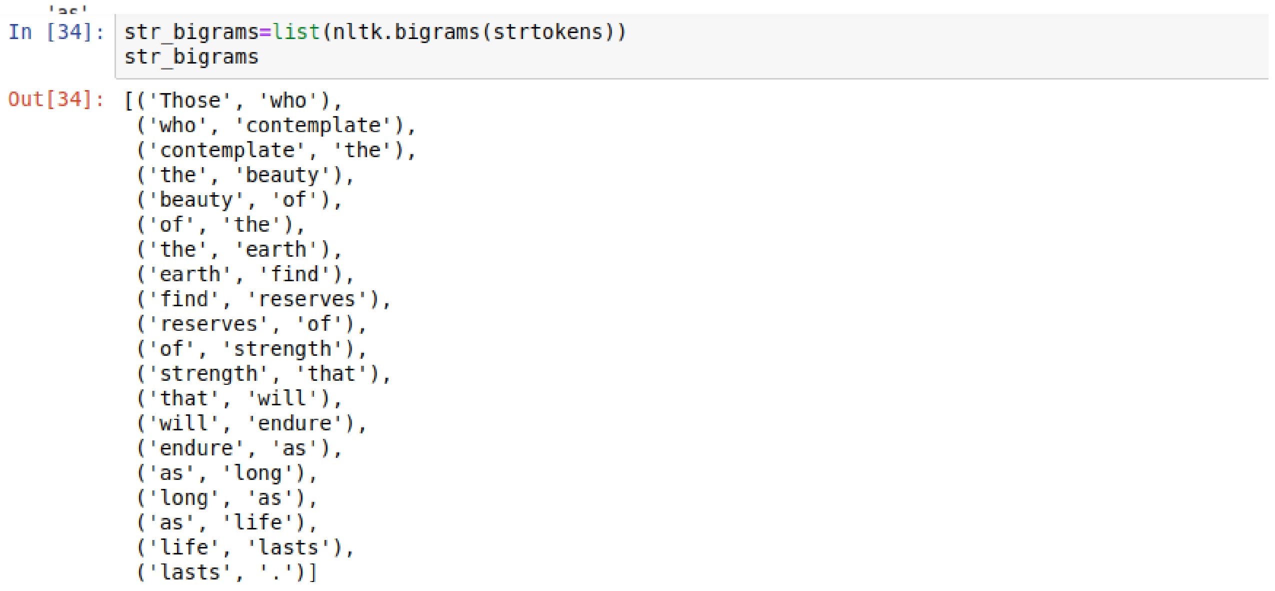
1.**Bigrams:** Tokens of two consecutive written words .

2.**Trigrams**: Tokens of three consecutive written words .

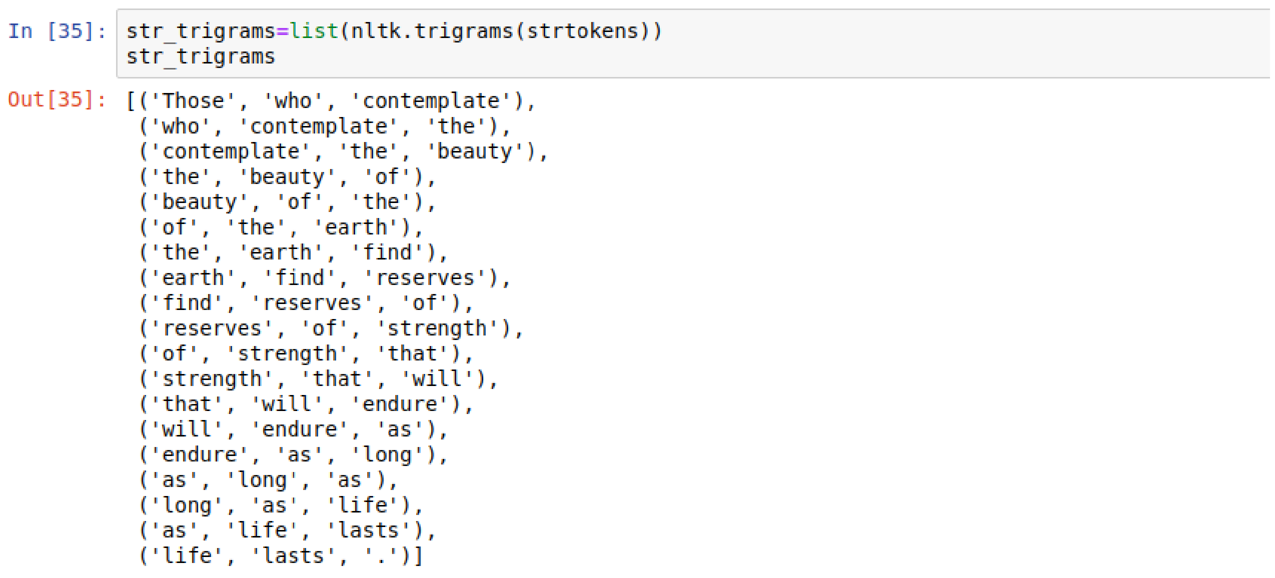
3.**Ngrams:** Tokens of any number of consecutive written words



```
In [31]: from nltk.util import bigrams,trigrams,ngrams

In [32]: string="Those who contemplate the beauty of the earth find reserves of strength that will endure as long as life la

In [33]: strtokens=nltk.word_tokenize(string)
         strtokens

Out[33]: ['Those',
          'who',
          'contemplate',
          'the',
          'beauty',
          'of',
          'the',
          'earth',
          'find',
          'reserves',
          'of',
          'strength',
          'that',
          'will',
          'endure',
          'as',
          'long',
```

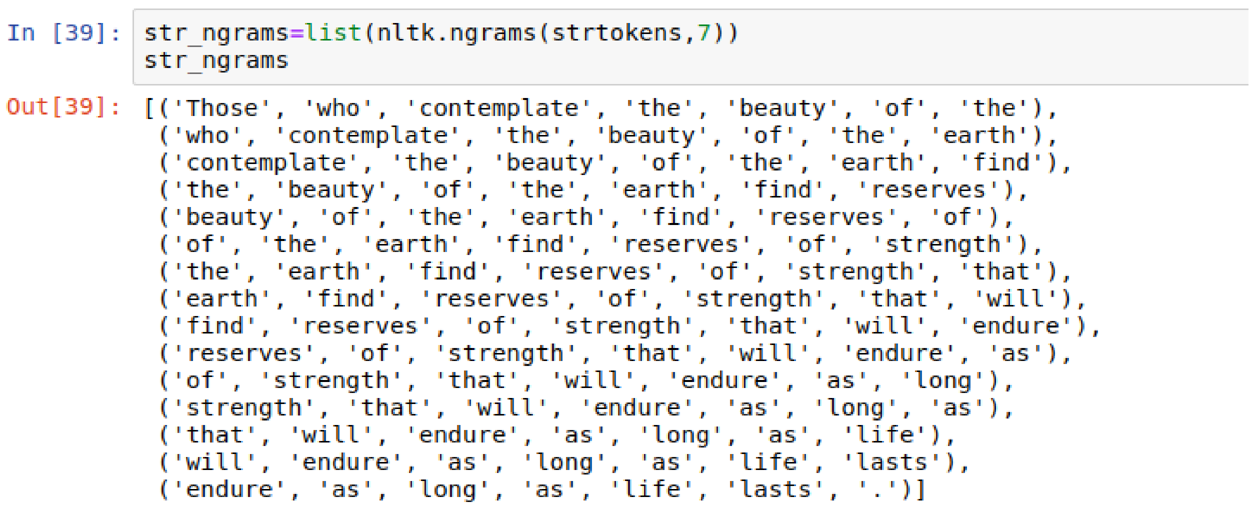


```
In [34]: str_bigrams=list(nltk.bigrams(strtokens))
         str_bigrams

Out[34]: [('Those', 'who'),
          ('who', 'contemplate'),
          ('contemplate', 'the'),
          ('the', 'beauty'),
          ('beauty', 'of'),
          ('of', 'the'),
          ('the', 'earth'),
          ('earth', 'find'),
          ('find', 'reserves'),
          ('reserves', 'of'),
          ('of', 'strength'),
          ('strength', 'that'),
          ('that', 'will'),
          ('will', 'endure'),
          ('endure', 'as'),
          ('as', 'long'),
          ('long', 'as'),
          ('as', 'life'),
          ('life', 'lasts'),
          ('lasts', '.')]
```



```
In [35]: str_trigrams=list(nltk.trigrams(strtokens))
         str_trigrams
```

```
Out[35]: [('Those', 'who', 'contemplate'),
          ('who', 'contemplate', 'the'),
          ('contemplate', 'the', 'beauty'),
          ('the', 'beauty', 'of'),
          ('beauty', 'of', 'the'),
          ('of', 'the', 'earth'),
          ('the', 'earth', 'find'),
          ('earth', 'find', 'reserves'),
          ('find', 'reserves', 'of'),
          ('reserves', 'of', 'strength'),
          ('of', 'strength', 'that'),
          ('strength', 'that', 'will'),
          ('that', 'will', 'endure'),
          ('will', 'endure', 'as'),
          ('endure', 'as', 'long'),
          ('as', 'long', 'as'),
          ('long', 'as', 'life'),
          ('as', 'life', 'lasts'),
          ('life', 'lasts', '.')]
```

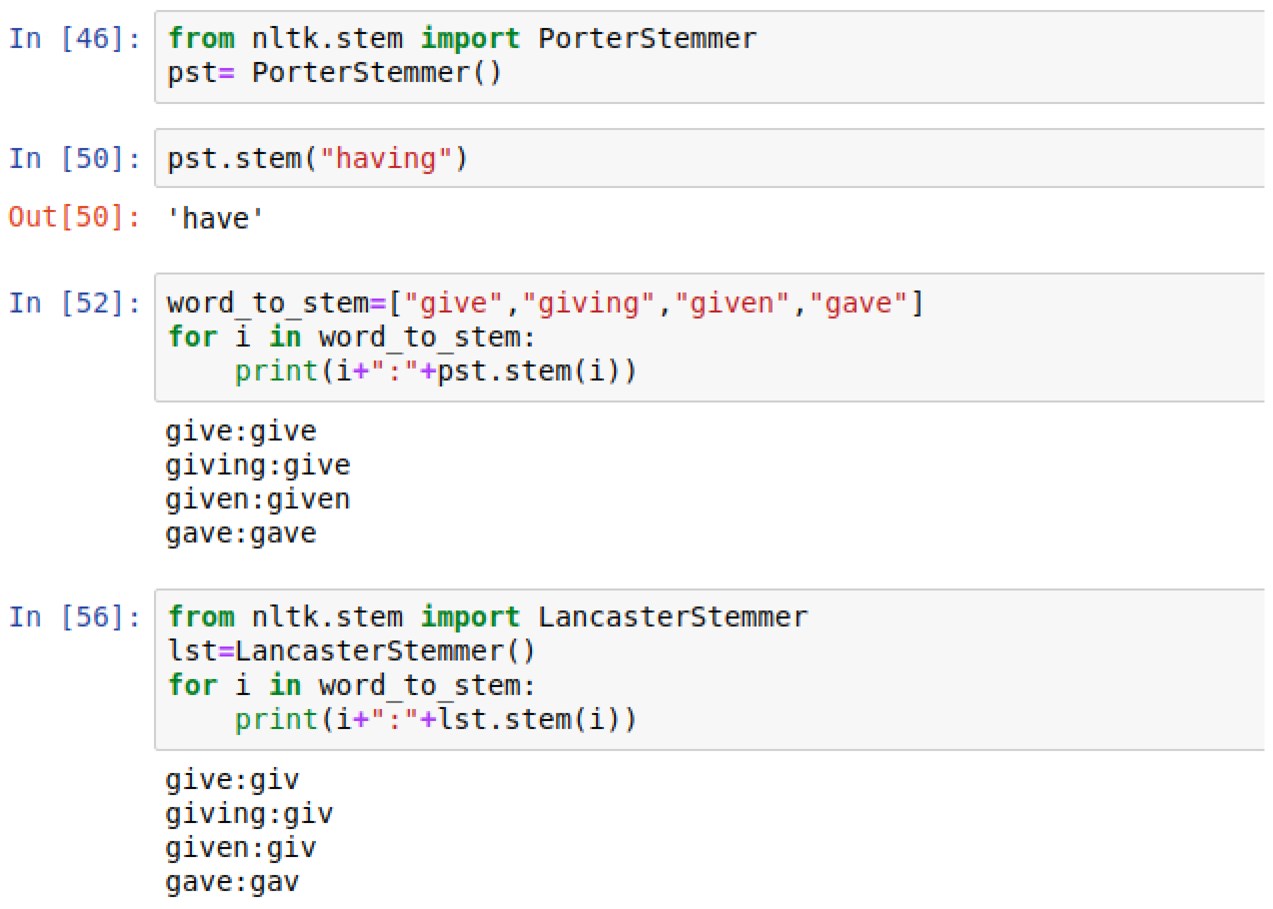


```
In [39]: str_ngrams=list(nltk.ngrams(strtokens,7))
         str_ngrams
```

```
Out[39]: [('Those', 'who', 'contemplate', 'the', 'beauty', 'of', 'the'),
          ('who', 'contemplate', 'the', 'beauty', 'of', 'the', 'earth'),
          ('contemplate', 'the', 'beauty', 'of', 'the', 'earth', 'find'),
          ('the', 'beauty', 'of', 'the', 'earth', 'find', 'reserves'),
          ('beauty', 'of', 'the', 'earth', 'find', 'reserves', 'of'),
          ('of', 'the', 'earth', 'find', 'reserves', 'of', 'strength'),
          ('the', 'earth', 'find', 'reserves', 'of', 'strength', 'that'),
          ('earth', 'find', 'reserves', 'of', 'strength', 'that', 'will'),
          ('find', 'reserves', 'of', 'strength', 'that', 'will', 'endure'),
          ('reserves', 'of', 'strength', 'that', 'will', 'endure', 'as'),
          ('of', 'strength', 'that', 'will', 'endure', 'as', 'long'),
          ('strength', 'that', 'will', 'endure', 'as', 'long', 'as'),
          ('that', 'will', 'endure', 'as', 'long', 'as', 'life'),
          ('will', 'endure', 'as', 'long', 'as', 'life', 'lasts'),
          ('endure', 'as', 'long', 'as', 'life', 'lasts', '.')]
```

2.Stemming:

- Once we have tokens we need to change something that is called stemming.
- Normalize words into its base form or root form.
- Example: Affects, Affected, Affection, Affecting..



```
In [46]: from nltk.stem import PorterStemmer
         pst= PorterStemmer()

In [50]: pst.stem("having")

Out[50]: 'have'

In [52]: word_to_stem=["give","giving","given","gave"]
         for i in word_to_stem:
             print(i+":"+pst.stem(i))

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

In [56]: from nltk.stem import LancasterStemmer
         lst=LancasterStemmer()
         for i in word_to_stem:
             print(i+":"+lst.stem(i))

         give:giv
         giving:giv
         given:giv
         gave:gav
```

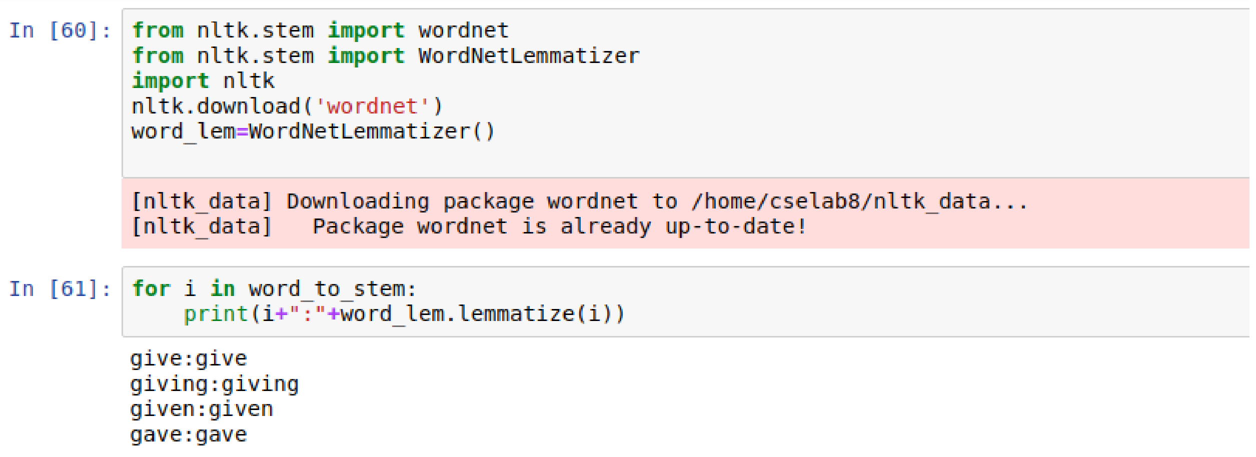
3.Lemmatization:

Lemmatization has taken for logical analysis of the word.

Uses of Lemmatization:

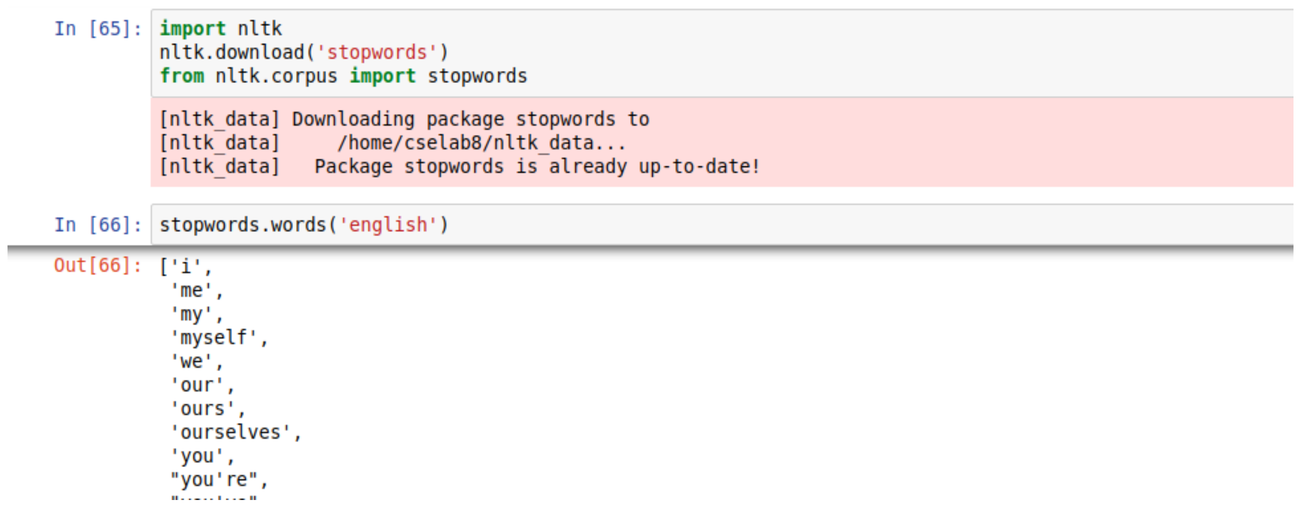
- Groups together different inflected forms of a word called lemma.
- Somehow similar to stemming, as it maps several words into one common root.
- Output of lemmatization is a proper word.

Example: Lemmatiser should map gone, going and went to go.

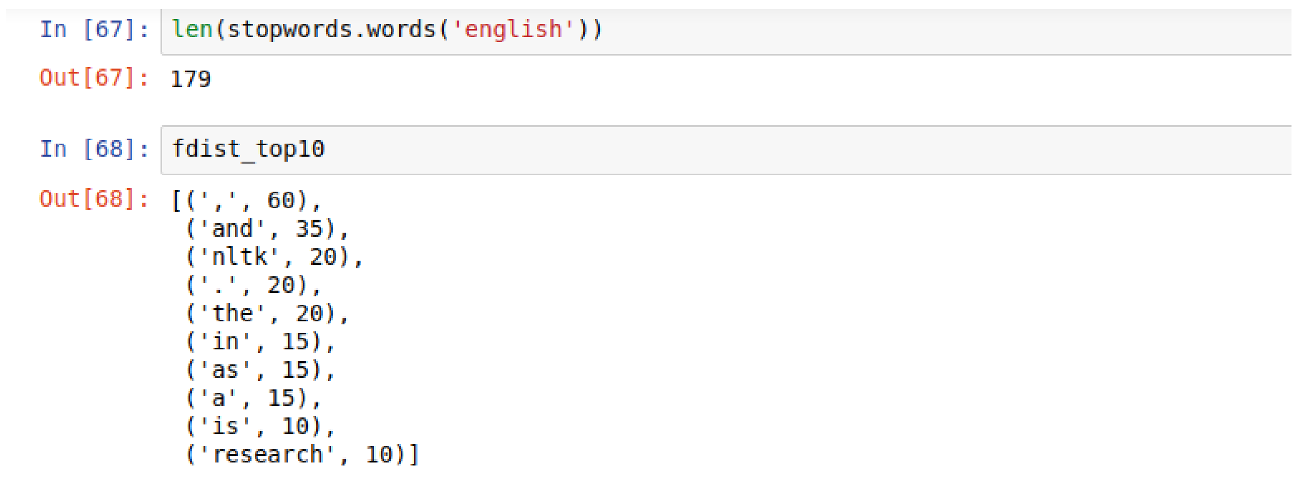


```
In [60]: from nltk.stem import wordnet
         from nltk.stem import WordNetLemmatizer
         import nltk
         nltk.download('wordnet')
         word_lem=WordNetLemmatizer()

         [nltk_data] Downloading package wordnet to /home/cselab8/nltk_data...
         [nltk_data]   Package wordnet is already up-to-date!

In [61]: for i in word_to_stem:
             print(i+":"+word_lem.lemmatize(i))

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

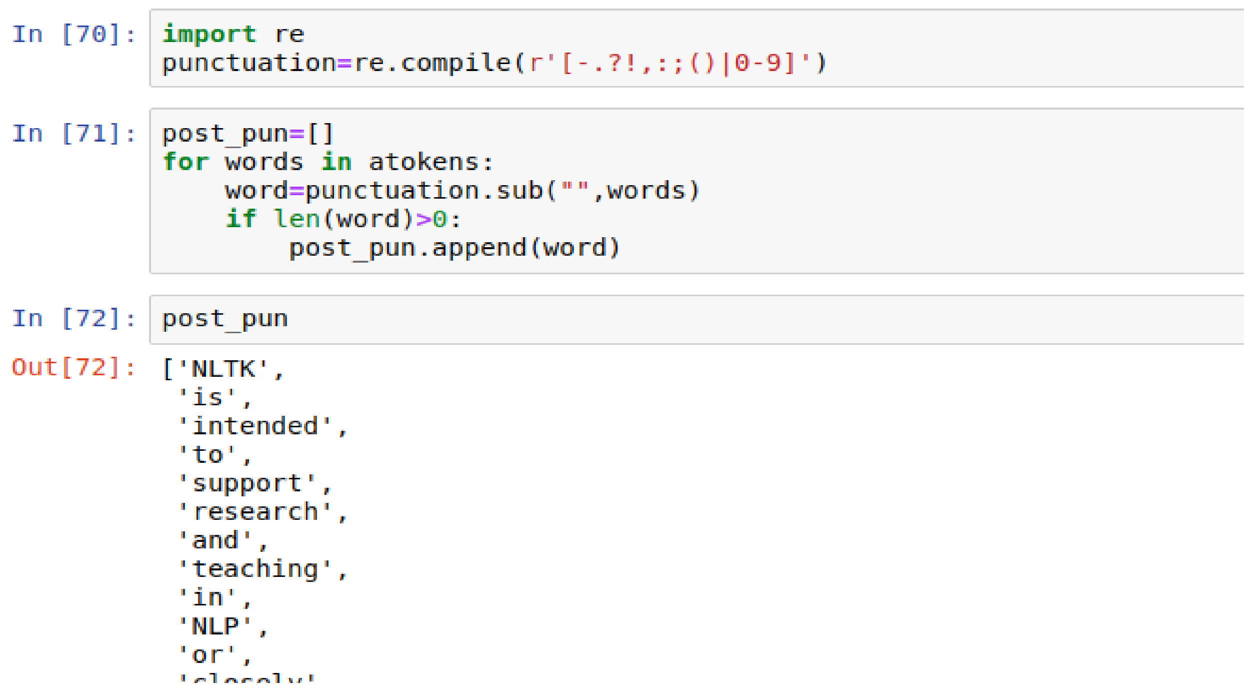
Stop Words



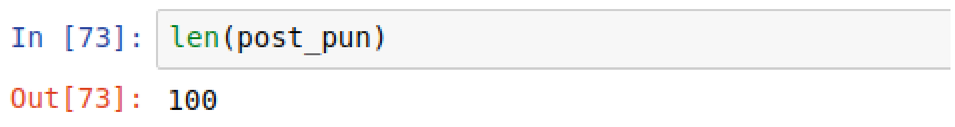
```
In [65]: import nltk
         nltk.download('stopwords')
         from nltk.corpus import stopwords

         [nltk_data] Downloading package stopwords to
         [nltk_data]     /home/cselab8/nltk_data...
         [nltk_data]   Package stopwords is already up-to-date!

In [66]: stopwords.words('english')

Out[66]: ['i',
          'me',
          'my',
          'myself',
          'we',
          'our',
          'ours',
          'ourselves',
          'you',
          "you're",
```



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

Out[67]: 179

In [68]: fdist_top10

Out[68]: [(',', 60),
          ('and', 35),
          ('nltk', 20),
          ('.', 20),
          ('the', 20),
          ('in', 15),
          ('as', 15),
          ('a', 15),
          ('is', 10),
          ('research', 10)]
```



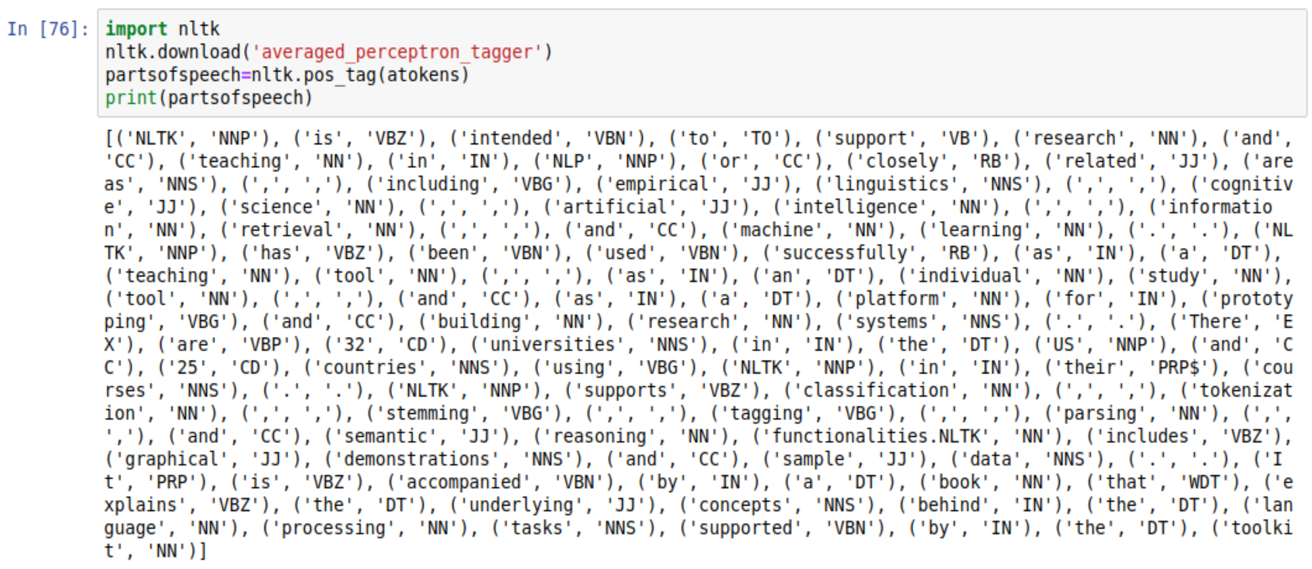
```
In [70]: import re
         punctuation=re.compile(r'[-.?!,:;()|0-9]')

In [71]: post_pun=[]
         for words in atokens:
             word=punctuation.sub("",words)
             if len(word)>0:
                 post_pun.append(word)

In [72]: post_pun

Out[72]: ['NLTK',
          'is',
          'intended',
          'to',
          'support',
          'research',
          'and',
          'teaching',
          'in',
          'NLP',
          'or',
```



```
In [73]: len(post_pun)

Out[73]: 100
```

Parts of Speech:

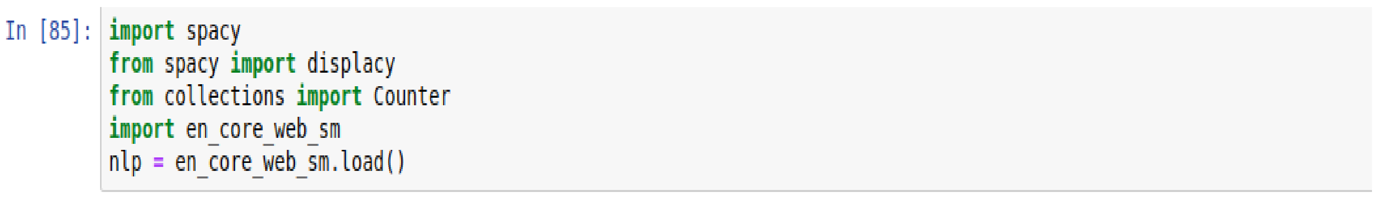
To print the parts of speech to all the words in the given sentence.



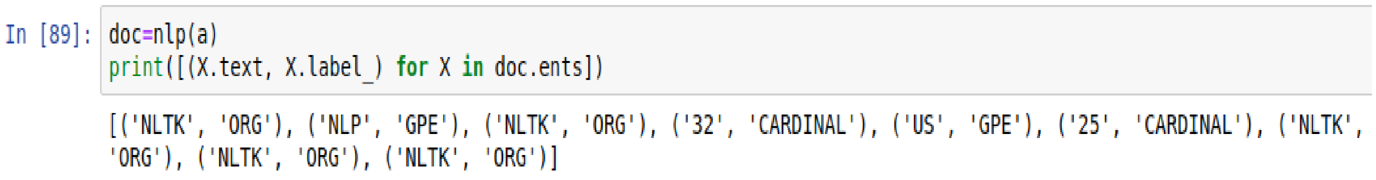
```
In [76]: import nltk
         nltk.download('averaged_perceptron_tagger')
         partsofspeech=nltk.pos_tag(atokens)
         print(partsofspeech)

         [('NLTK', 'NNP'), ('is', 'VBZ'), ('intended', 'VBN'), ('to', 'TO'), ('support', 'VB'), ('research', 'NN'), ('and',
         'CC'), ('teaching', 'NN'), ('in', 'IN'), ('NLP', 'NNP'), ('or', 'CC'), ('closely', 'RB'), ('related', 'JJ'), ('are
         as', 'NNS'), (',', ','), ('including', 'VBG'), ('empirical', 'JJ'), ('linguistics', 'NNS'), (',', ','), ('cognitiv
         e', 'JJ'), ('science', 'NN'), (',', ','), ('artificial', 'JJ'), ('intelligence', 'NN'), (',', ','), ('informatio
         n', 'NN'), ('retrieval', 'NN'), (',', ','), ('and', 'CC'), ('machine', 'NN'), ('learning', 'NN'), ('.', '.'), ('NL
         TK', 'NNP'), ('has', 'VBZ'), ('been', 'VBN'), ('used', 'VBN'), ('successfully', 'RB'), ('as', 'IN'), ('a', 'DT'),
         ('teaching', 'NN'), ('tool', 'NN'), (',', ','), ('as', 'IN'), ('an', 'DT'), ('individual', 'NN'), ('study', 'NN'),
         ('tool', 'NN'), (',', ','), ('and', 'CC'), ('as', 'IN'), ('a', 'DT'), ('platform', 'NN'), ('for', 'IN'), ('prototy
         ping', 'VBG'), ('and', 'CC'), ('building', 'NN'), ('research', 'NN'), ('systems', 'NNS'), ('.', '.'), ('There', 'E
         X'), ('are', 'VBP'), ('32', 'CD'), ('universities', 'NNS'), ('in', 'IN'), ('the', 'DT'), ('US', 'NNP'), ('and', 'C
         C'), ('25', 'CD'), ('countries', 'NNS'), ('using', 'VBG'), ('NLTK', 'NNP'), ('in', 'IN'), ('their', 'PRP$'), ('cou
         rses', 'NNS'), ('.', '.'), ('NLTK', 'NNP'), ('supports', 'VBZ'), ('classification', 'NN'), (',', ','), ('tokenizat
         ion', 'NN'), (',', ','), ('stemming', 'VBG'), (',', ','), ('tagging', 'VBG'), (',', ','), ('parsing', 'NN'), (',',
         ','), ('and', 'CC'), ('semantic', 'JJ'), ('reasoning', 'NN'), ('functionalities.NLTK', 'NN'), ('includes', 'VBZ'),
         ('graphical', 'JJ'), ('demonstrations', 'NNS'), ('and', 'CC'), ('sample', 'JJ'), ('data', 'NNS'), ('.', '.'), ('I
         t', 'PRP'), ('is', 'VBZ'), ('accompanied', 'VBN'), ('by', 'IN'), ('a', 'DT'), ('book', 'NN'), ('that', 'WDT'), ('e
         xplains', 'VBZ'), ('the', 'DT'), ('underlying', 'JJ'), ('concepts', 'NNS'), ('behind', 'IN'), ('the', 'DT'), ('lan
         guage', 'NN'), ('processing', 'NN'), ('tasks', 'NNS'), ('supported', 'VBN'), ('by', 'IN'), ('the', 'DT'), ('toolki
         t', 'NN')]
```

Named Enitity Recognition:

Named Entity Recognition and Classification(NERC) is a process of recognizing information units like names, including person, organization and location names, and numeric expressions including time, date, money and percent expressions from unstructured text. The goal is to develop practical and domain-independent techniques in order to detect named entities with high accuracy automatically.



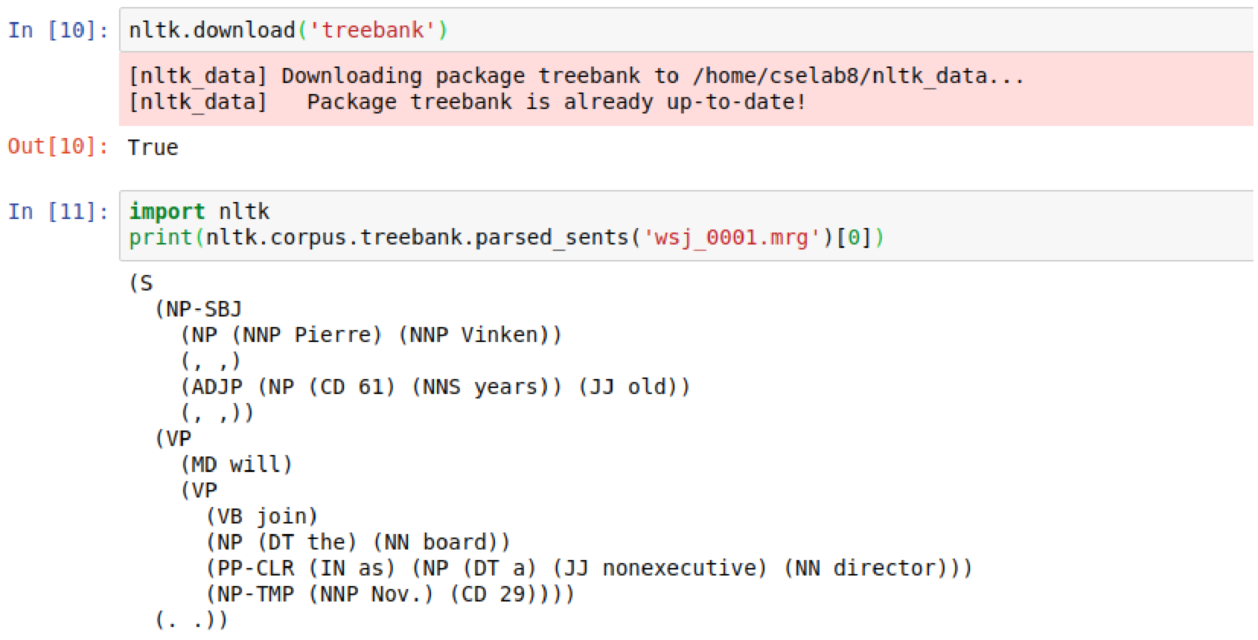
```
In [85]: import spacy
         from spacy import displacy
         from collections import Counter
         import en_core_web_sm
         nlp = en_core_web_sm.load()
```



```
In [89]: doc=nlp(a)
         print([(X.text, X.label_) for X in doc.ents])

         [('NLTK', 'ORG'), ('NLP', 'GPE'), ('NLTK', 'ORG'), ('32', 'CARDINAL'), ('US', 'GPE'), ('25', 'CARDINAL'), ('NLTK',
         'ORG'), ('NLTK', 'ORG'), ('NLTK', 'ORG')]
```

Syntax Parsing :

Syntactic parsing is a technique by which segmented, tokenized, and part-of-speech tagged text is assigned a structure that reveals the relationships between tokens governed by syntax rules, e.g. by grammars.



```
In [10]: nltk.download('treebank')

[nltk_data] Downloading package treebank to /home/cselab8/nltk_data...
[nltk_data]   Package treebank is already up-to-date!

Out[10]: True

In [11]: import nltk
         print(nltk.corpus.treebank.parsed_sents('wsj_0001.mrg')[0])

(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken))
    (, ,)
    (ADJP (NP (CD 61) (NNS years)) (JJ old))
    (, ,))
  (VP
    (MD will)
    (VP
      (VB join)
      (NP (DT the) (NN board))
      (PP-CLR (IN as) (NP (DT a) (JJ nonexecutive) (NN director)))
      (NP-TMP (NNP Nov.) (CD 29))))
  (. .))
```

Sentimental Analysis:

Sentiment Analysis is the process of ‘computationally’ determining whether a piece of writing is positive, negative or neutral. It’s also known as opinion mining, deriving the opinion or attitude of a speaker.

In [12]:
```
nltk.download('vader_lexicon')
```
```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /home/cselab8/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```
Out[12]:
```
True
```

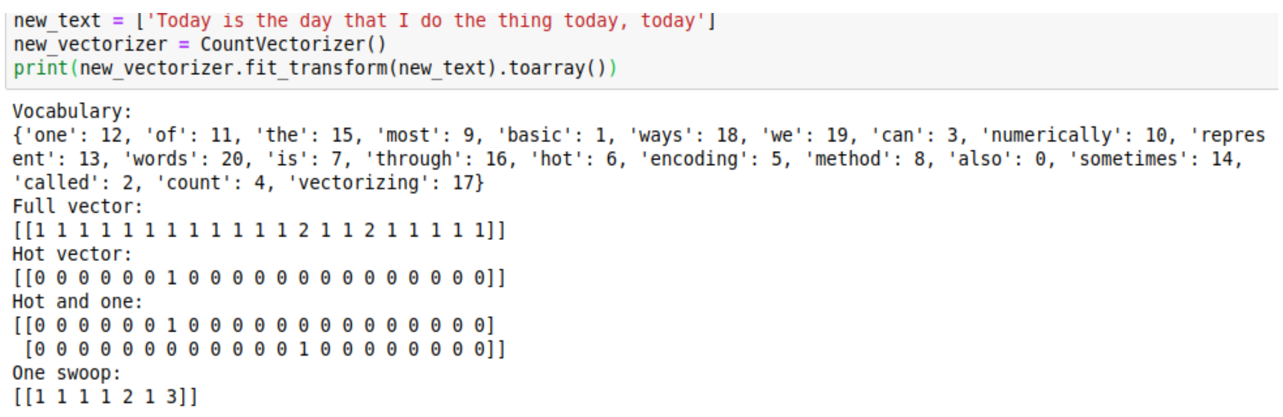
In [16]:
```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
hotel_rev = ['Great place to be when you are in Bangalore.',
"The place was being renovated when I visited so the seating was limited.",
'Loved the ambience, loved the food',
'The food is delicious but not over the top.',
'Service - Little slow, probably because too many people.',
'The place is not easy to locate'
'Mushroom fried rice was tasty']
sid = SentimentIntensityAnalyzer()
for sentence in hotel_rev:
    print(sentence)
    ss = sid.polarity_scores(sentence)
    for k in ss:
        print('{0}: {1}, '.format(k, ss[k]), end='')
        print()
```
```
Great place to be when you are in Bangalore.
neg: 0.0,
neu: 0.661,
pos: 0.339,
compound: 0.6249,
The place was being renovated when I visited so the seating was limited.
neg: 0.147,
neu: 0.853,
pos: 0.0,
compound: -0.2263,
Loved the ambience, loved the food
neg: 0.0,
neu: 0.339,
pos: 0.661,
compound: 0.8316,
```
```
The food is delicious but not over the top.
neg: 0.168,
neu: 0.623,
pos: 0.209,
compound: 0.1184,
Service - Little slow, probably because too many people.
neg: 0.0,
neu: 1.0,
pos: 0.0,
compound: 0.0,
The place is not easy to locateMushroom fried rice was tasty
neg: 0.194,
neu: 0.806,
pos: 0.0,
compound: -0.3412,
```

Join all the tokens into paragraph:

```
In [25]: from nltk.tokenize.treebank import TreebankWordDetokenizer
         TreebankWordDetokenizer().detokenize(atokens)

Out[25]: 'NLTK is intended to support research and teaching in NLP or closely related areas, including empirical linguistic
         s, cognitive science, artificial intelligence, information retrieval, and machine learning . NLTK has been used su
         ccessfully as a teaching tool, as an individual study tool, and as a platform for prototyping and building researc
         h systems . There are 32 universities in the US and 25 countries using NLTK in their courses . NLTK supports class
         ification, tokenization, stemming, tagging, parsing, and semantic reasoning functionalities.NLTK includes graphica
         l demonstrations and sample data . It is accompanied by a book that explains the underlying concepts behind the la
         nguage processing tasks supported by the toolkit'
```

Count vectorizer:

```
[26]: from sklearn.feature_extraction.text import CountVectorizer

      # To create a Count Vectorizer, we simply need to instantiate one.
      # There are special parameters we can set here when making the vectorizer, but
      # for the most basic example, it is not needed.
      vectorizer = CountVectorizer()
      # For our text, we are going to take some text from our previous blog post
      # about count vectorization
      sample_text = ["One of the most basic ways we can numerically represent words "
      "is through the one-hot encoding method (also sometimes called "
      "count vectorizing)."]

      # To actually create the vectorizer, we simply need to call fit on the text
      # data that we wish to fix
      vectorizer.fit(sample_text)
      # Now, we can inspect how our vectorizer vectorized the text
      # This will print out a list of words used, and their index in the vectors
      print('Vocabulary: ')
      print(vectorizer.vocabulary_)
      vector = vectorizer.transform(sample_text)
      # Our final vector:
      print('Full vector: ')
      print(vector.toarray())
      # Or if we wanted to get the vector for one word:
      print('Hot vector: ')
      print(vectorizer.transform(['hot']).toarray())
      # Or if we wanted to get multiple vectors at once to build matrices
      print('Hot and one: ')
      print(vectorizer.transform(['hot', 'one']).toarray())
      # We could also do the whole thing at once with the fit_transform method:
      print('One swoop:')
```



```
new_text = ['Today is the day that I do the thing today, today']
new_vectorizer = CountVectorizer()
print(new_vectorizer.fit_transform(new_text).toarray())

Vocabulary:
{'one': 12, 'of': 11, 'the': 15, 'most': 9, 'basic': 1, 'ways': 18, 'we': 19, 'can': 3, 'numerically': 10, 'repres
ent': 13, 'words': 20, 'is': 7, 'through': 16, 'hot': 6, 'encoding': 5, 'method': 8, 'also': 0, 'sometimes': 14,
'called': 2, 'count': 4, 'vectorizing': 17}
Full vector:
[[1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1]]
Hot vector:
[[0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
Hot and one:
[[0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]]
One swoop:
[[1 1 1 1 2 1 3]]
```

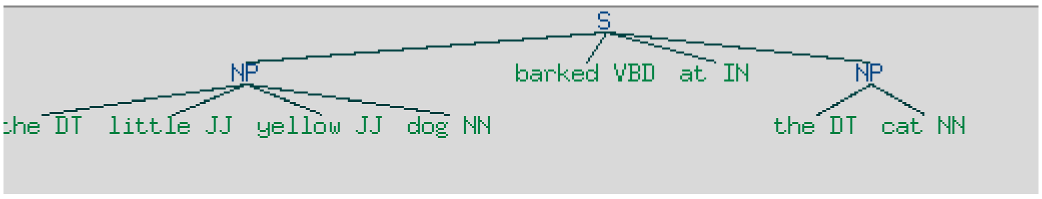
Chunking:

Chunking uses a special regexp syntax for rules that delimit the chunks. These rules must be converted to 'regular' regular expressions before a sentence can be chunked.

```
In [27]: sentence = "the little yellow dog barked at the cat"
         grammar = ('''
             NP: {<DT>?<JJ>*<NN>} # NP
             ''')
         chunkParser = nltk.RegexpParser(grammar)
         tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
         tagged

Out[27]: [('the', 'DT'),
          ('little', 'JJ'),
          ('yellow', 'JJ'),
          ('dog', 'NN'),
          ('barked', 'VBD'),
          ('at', 'IN'),
          ('the', 'DT'),
          ('cat', 'NN')]
```

```
In [28]: tree = chunkParser.parse(tagged)
         for subtree in tree.subtrees():
             print(subtree)

         (S
           (NP the/DT little/JJ yellow/JJ dog/NN)
           barked/VBD
           at/IN
           (NP the/DT cat/NN))
         (NP the/DT little/JJ yellow/JJ dog/NN)
         (NP the/DT cat/NN)
```

```
In [*]: tree.draw()
```


S
NP
barked VBD
at IN
NP
he DT
little JJ
yellow JJ
dog NN
the DT
cat NN