In [1]:

```
import textblob
```

Creating a TextBlob

In [2]:

```
from textblob import TextBlob
text = TextBlob( """Snowball is a small string processing language designed for creating st
This site describes Snowball, and presents several useful stemmers which have been implemen
)
```

Tokenizing

In [3]:

```
text.sentences #here which they use regular expressions
```

Out[3]:

```
[Sentence("Snowball is a small string processing language designed for creat
ing stemming algorithms for use in Information Retrieval."),
 Sentence("This site describes Snowball, and presents several useful stemmer
s which have been implemented using it puppies.")]
```

In [4]:

```
text.words #Here in Word tokenization , first text will tokenzie as sentence then to words
           #they are using TreeBlankTokenizer
```

Out[4]:

```
WordList(['Snowball', 'is', 'a', 'small', 'string', 'processing', 'languag
e', 'designed', 'for', 'creating', 'stemming', 'algorithms', 'for', 'use',
'in', 'Information', 'Retrieval', 'This', 'site', 'describes', 'Snowball',
'and', 'presents', 'several', 'useful', 'stemmers', 'which', 'have', 'been',
'implemented', 'using', 'it', 'puppies'])
```

In [5]:

```
#We can also use NLTK
from nltk.tokenize import TabTokenizer
tokenizer = TabTokenizer()
blob = TextBlob( """Snowball is a small string processing language designed for creating st
This site describes Snowball, and presents several useful stemmers which have been implemen
, tokenizer = tokenizer)
blob.tokens
```

Out[5]:

```
WordList(['Snowball is a small string processing language designed for creat
ing stemming algorithms for use in Information Retrieval. \nThis site descri
bes Snowball, and presents several useful stemmers which have been implement
ed using it puppies.'])
```

POS Tagging

In [6]:

```
text.tags #This is default POS Tagger in TextBlob called as PattrenTagger
```

Out[6]:

```
[('Snowball', 'NN'),
 ('is', 'VBZ'),
 ('a', 'DT'),
 ('small', 'JJ'),
 ('string', 'NN'),
 ('processing', 'NN'),
 ('language', 'NN'),
 ('designed', 'VBN'),
 ('for', 'IN'),
 ('creating', 'VBG'),
 ('stemming', 'VBG'),
 ('algorithms', 'NN'),
 ('for', 'IN'),
 ('use', 'NN'),
 ('in', 'IN'),
 ('Information', 'NNP'),
 ('Retrieval', 'NNP'),
 ('This', 'DT'),
 ('site', 'NN'),
 ('describes', 'VBZ'),
 ('Snowball', 'NNP'),
 ('and', 'CC'),
 ('presents', 'NNS'),
 ('several', 'JJ'),
 ('useful', 'JJ'),
 ('stemmers', 'NNS'),
 ('which', 'WDT'),
 ('have', 'VBP'),
 ('been', 'VBN'),
 ('implemented', 'VBN'),
 ('using', 'VBG'),
 ('it', 'PRP'),
 ('puppies', 'VBZ')]
```

In [7]:

```
#This is the second type with NLTKTagger
from textblob import TextBlob
from textblob.taggers import NLTKTagger
nltk_tagger = NLTKTagger()
blob = text
pos_tagger = nltk_tagger
blob.pos_tags
```

Out[7]:

```
[('Snowball', 'NN'),
 ('is', 'VBZ'),
 ('a', 'DT'),
 ('small', 'JJ'),
 ('string', 'NN'),
 ('processing', 'NN'),
 ('language', 'NN'),
 ('designed', 'VBN'),
 ('for', 'IN'),
 ('creating', 'VBG'),
 ('stemming', 'VBG'),
 ('algorithms', 'NN'),
 ('for', 'IN'),
 ('use', 'NN'),
 ('in', 'IN'),
 ('Information', 'NNP'),
 ('Retrieval', 'NNP'),
 ('This', 'DT'),
 ('site', 'NN'),
 ('describes', 'VBZ'),
 ('Snowball', 'NNP'),
 ('and', 'CC'),
 ('presents', 'NNS'),
 ('several', 'JJ'),
 ('useful', 'JJ'),
 ('stemmers', 'NNS'),
 ('which', 'WDT'),
 ('have', 'VBP'),
 ('been', 'VBN'),
 ('implemented', 'VBN'),
 ('using', 'VBG'),
 ('it', 'PRP'),
 ('puppies', 'VBZ')]
```

Noun Phrase Extraction

In [8]:

```
text.noun_phrases #This is the default NP extraction called FASTNPExtrator
```

Out[8]:

```
WordList(['snowball', 'small string processing language', 'information retri
eval', 'snowball', 'useful stemmers'])
```

In [9]:

```
#This is the second metho NP extrator by using Conlletractor based CoNLL 2000 Corpus
from textblob.np_extractors import ConllExtractor
extractor = ConllExtractor()
blob = text
blob.noun_phrases
```

Out[9]:

```
WordList(['snowball', 'small string processing language', 'information retri
eval', 'snowball', 'useful stemmers'])
```

Sentiment Analyzer

In [10]:

```
#This is a type of sentiment analyzer that which they used Pattren Analyzer
test = TextBlob("Snowball is a small ball which we made and designed by using snow for to p
test.sentiment
```

Out[10]:

```
Sentiment(polarity=-0.25, subjectivity=0.4)
```

In [11]:

```
#In this they used clasification menthod named NaiveBayes which gives (classification, p_po
from textblob.sentiments import NaiveBayesAnalyzer
blob = TextBlob("Snowball is a small ball which we made and designed by using snow for to p
blob.sentiment
```

Out[11]:

```
Sentiment(classification='pos', p_pos=0.950945918761427, p_neg=0.04905408123
857359)
```

Words Inflection and Lemmatization

In [12]:

```
#They perform word inflection like making singular into plural
sentence = text
sentence.words
```

Out[12]:

```
WordList(['Snowball', 'is', 'a', 'small', 'string', 'processing', 'languag
e', 'designed', 'for', 'creating', 'stemming', 'algorithms', 'for', 'use',
'in', 'Information', 'Retrieval', 'This', 'site', 'describes', 'Snowball',
'and', 'presents', 'several', 'useful', 'stemmers', 'which', 'have', 'been',
'implemented', 'using', 'it', 'puppies'])
```

In [13]:

```
sentence.words[11].singularize()
```

Out[13]:

```
'algorithm'
```

In [14]:

```
sentence.words[-1].singularize() #it hase ies in the ending it work good
```

Out[14]:

```
'puppy'
```

In [15]:

```
sentence.words[-9].pluralize() #in plularizing it only adds "S at the end whether it has pl
```

Out[15]:

```
'usefuls'
```

In [16]:

```
sentence.words[7].pluralize()
```

Out[16]:

```
'designeds'
```

Lemmatizing

In [17]:

```
#lemmatizing the words by using lemmatize funtion
from textblob import Word
w = Word("geese")
w.lemmatize()
```

Out[17]:

```
'goose'
```

In [18]:

```
w = Word("using")
w.lemmatize("VBG") #Pos_tagging of given word
```

Out[18]:

```
'use'
```

Wordnet

In [19]:

```
#in this we are trying get the different type of synonyms of words irrespetive of POS
word = Word("hack")
word.synsets
```

Out[19]:

```
[Synset('hack.n.01'),
 Synset('machine_politician.n.01'),
 Synset('hack.n.03'),
 Synset('hack.n.04'),
 Synset('cab.n.03'),
 Synset('hack.n.06'),
 Synset('hack.n.07'),
 Synset('hack.n.08'),
 Synset('chop.v.05'),
 Synset('hack.v.02'),
 Synset('hack.v.03'),
 Synset('hack.v.04'),
 Synset('hack.v.05'),
 Synset('hack.v.06'),
 Synset('hack.v.07'),
 Synset('hack.v.08')]
```

In [20]:

```
Word("hack").get_synsets(pos='v') #Here we getting only POS tag = VERB
```

Out[20]:

```
[Synset('chop.v.05'),
 Synset('hack.v.02'),
 Synset('hack.v.03'),
 Synset('hack.v.04'),
 Synset('hack.v.05'),
 Synset('hack.v.06'),
 Synset('hack.v.07'),
 Synset('hack.v.08')]
```

In [21]:

```
Word("hack").get_synsets(pos='n') # getting only nouns
```

Out[21]:

```
[Synset('hack.n.01'),
 Synset('machine_politician.n.01'),
 Synset('hack.n.03'),
 Synset('hack.n.04'),
 Synset('cab.n.03'),
 Synset('hack.n.06'),
 Synset('hack.n.07'),
 Synset('hack.n.08')]
```

In [22]:

```
from textblob.wordnet import VERB #anthor way of representing With POS
word = Word("went")
word.synsets
```

Out[22]:

```
[Synset('travel.v.01'),
 Synset('go.v.02'),
 Synset('go.v.03'),
 Synset('become.v.01'),
 Synset('go.v.05'),
 Synset('run.v.05'),
 Synset('run.v.03'),
 Synset('proceed.v.04'),
 Synset('go.v.09'),
 Synset('go.v.10'),
 Synset('sound.v.02'),
 Synset('function.v.01'),
 Synset('run_low.v.01'),
 Synset('move.v.13'),
 Synset('survive.v.01'),
 Synset('go.v.16'),
 Synset('die.v.01'),
 Synset('belong.v.03'),
 Synset('go.v.19'),
 Synset('start.v.09'),
 Synset('move.v.15'),
 Synset('go.v.22'),
 Synset('go.v.23'),
 Synset('blend.v.02'),
 Synset('go.v.25'),
 Synset('fit.v.02'),
 Synset('rifle.v.02'),
 Synset('go.v.28'),
 Synset('plump.v.04'),
 Synset('fail.v.04')]
```

In [23]:

```
Word("film").definitions #getting defination from wordnet
```

Out[23]:

```
['a form of entertainment that enacts a story by sound and a sequence of ima
ges giving the illusion of continuous movement',
 'a medium that disseminates moving pictures',
 'photographic material consisting of a base of celluloid covered with a pho
tographic emulsion; used to make negatives or transparencies',
 'a thin coating or layer',
 'a thin sheet of (usually plastic and usually transparent) material used to
wrap or cover things',
 'make a film or photograph of something',
 'record in film']
```

In [24]:

```
#checking the similarity between two different words
from textblob.wordnet import Synset
hack = Synset('hack.v.03')
went = Synset('run.v.03')
hack.path_similarity(went)
```

Out[24]:

```
0.14285714285714285
```

In [25]:

```
hack = Synset('hack.v.03')
hack = Synset('machine_politician.n.01')
hack.path_similarity(hack) #this works great
```

Out[25]:

```
1.0
```

Spelling Correction

In [26]:

```
#Doing spelling correction by using Correction() method
sent = TextBlob("Hwy Therw!") #i type HEY THERE
print(sent.correct())
```

```
Twy Her!
```

In [27]:

```
#spelling correction by using spellcorection() method
from textblob import Word
w = Word('faillling') #I typed failing
w.spellcheck()
```

Out[27]:

```
[('falling', 0.4778761061946903),
 ('filling', 0.26548672566371684),
 ('failing', 0.25663716814159293)]
```

In [28]:

```
w = Word('Hwy') # i typed Hey

w.spellcheck()
##They already mentioned spelling correction is only working with 70% accurate
```

Out[28]:

```
[('Twy', 0.75), ('Dwy', 0.25)]
```

Word and Noun Phrase Frequencies Count

In [29]:

```
text.word_counts['for'] #count of word in TextBlob text
```

Out[29]:

```
2
```

In [30]:

```
text.words.count('FOR', case_sensitive=True) #count with case sensitive on
```

Out[30]:

```
0
```

In [31]:

```
text.noun_phrases #list of NP
```

Out[31]:

```
WordList(['snowball', 'small string processing language', 'information retri
eval', 'snowball', 'useful stemmers'])
```

In [32]:

```
text.noun_phrases.count('snowball') #count of NP snowball
```

Out[32]:

```
2
```

n grams

In [33]:

```
text.ngrams(n=3) #getting the list of 3 successive words
```

Out[33]:

```
[WordList(['Snowball', 'is', 'a']),
 WordList(['is', 'a', 'small']),
 WordList(['a', 'small', 'string']),
 WordList(['small', 'string', 'processing']),
 WordList(['string', 'processing', 'language']),
 WordList(['processing', 'language', 'designed']),
 WordList(['language', 'designed', 'for']),
 WordList(['designed', 'for', 'creating']),
 WordList(['for', 'creating', 'stemming']),
 WordList(['creating', 'stemming', 'algorithms']),
 WordList(['stemming', 'algorithms', 'for']),
 WordList(['algorithms', 'for', 'use']),
 WordList(['for', 'use', 'in']),
 WordList(['use', 'in', 'Information']),
 WordList(['in', 'Information', 'Retrieval']),
 WordList(['Information', 'Retrieval', 'This']),
 WordList(['Retrieval', 'This', 'site']),
 WordList(['This', 'site', 'describes']),
 WordList(['site', 'describes', 'Snowball']),
 WordList(['describes', 'Snowball', 'and']),
 WordList(['Snowball', 'and', 'presents']),
 WordList(['and', 'presents', 'several']),
 WordList(['presents', 'several', 'useful']),
 WordList(['several', 'useful', 'stemmers']),
 WordList(['useful', 'stemmers', 'which']),
 WordList(['stemmers', 'which', 'have']),
 WordList(['which', 'have', 'been']),
 WordList(['have', 'been', 'implemented']),
 WordList(['been', 'implemented', 'using']),
 WordList(['implemented', 'using', 'it']),
 WordList(['using', 'it', 'puppies'])]
```