In [1]:

```
#Lexicons
#import the stopwords
from nltk.corpus import stopwords
stopwords.words('english')
```

Out[1]:

```
['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',
 'hers',
 'herself',
 'it',
 "it's",
 'its',
 'itself',
 'they',
 'them',
 'their',
 'theirs',
 'themselves',
 'what',
 'which',
 'who',
 'whom',
 'this',
 'that',
 "that'll",
 'these',
 'those',
 'am',
 'is',
 'are',
 'was',
 'were',
 'be',
 'been',
 'being',
 'have',
 'has',
 'had',
 'having',
 'do',
 'does',
 'did',
 'doing',
 'a',
 'an',
 'the',
 'and',
 'but',
 'if',
 'or',
 'because',
 'as',
 'until',
 'while',
 'of',
 'at',
 'by',
 'for',
 'with',
 'about',
 'against',
 'between',
 'into',
 'through',
 'during',
 'before',
 'after',
 'above',
 'below',
 'to',
 'from',
 'up',
 'down',
 'in',
 'out',
 'on',
 'off',
 'over',
 'under',
 'again',
 'further',
 'then',
 'once',
 'here',
 'there',
 'when',
 'where',
 'why',
 'how',
 'all',
 'any',
 'both',
 'each',
 'few',
 'more',
 'most',
 'other',
 'some',
 'such',
 'no',
 'nor',
 'not',
 'only',
 'own',
 'same',
 'so',
 'than',
 'too',
 'very',
 's',
 't',
 'can',
 'will',
 'just',
 'don',
 "don't",
 'should',
 "should've",
 'now',
 'd',
 'll',
 'm',
 'o',
 're',
 've',
 'y',
 'ain',
 'aren',
 "aren't",
 'couldn',
 "couldn't",
 'didn',
 "didn't",
 'doesn',
 "doesn't",
 'hadn',
 "hadn't",
 'hasn',
 "hasn't",
 'haven',
 "haven't",
 'isn',
 "isn't",
 'ma',
 'mightn',
 "mightn't",
 'mustn',
 "mustn't",
 'needn',
 "needn't",
 'shan',
 "shan't",
 'shouldn',
 "shouldn't",
 'wasn',
 "wasn't",
 'weren',
 "weren't",
 'won',
 "won't",
 'wouldn',
 "wouldn't"]
```

In [2]:

```
import nltk
entries = nltk.corpus.cmudict.entries()
len(entries)
```

Out[2]:

```
133737
```

In [3]:

```
entries[:100]
```

Out[3]:

```
[('a', ['AH0']),
 ('a.', ['EY1']),
 ('a', ['EY1']),
 ('a42128',
  ['EY1',
   'F',
   'AO1',
   'R',
   'T',
   'UW1',
   'W',
   'AH1',
   'N',
   'T',
   'UW1',
   'EY1',
   'T']),
 ('aaa', ['T', 'R', 'IH2', 'P', 'AH0', 'L', 'EY1']),
 ('aaberg', ['AA1', 'B', 'ER0', 'G']),
 ('aachen', ['AA1', 'K', 'AH0', 'N']),
 ('aachener', ['AA1', 'K', 'AH0', 'N', 'ER0']),
 ('aaker', ['AA1', 'K', 'ER0']),
 ('aalseth', ['AA1', 'L', 'S', 'EH0', 'TH']),
 ('aamodt', ['AA1', 'M', 'AH0', 'T']),
 ('aancor', ['AA1', 'N', 'K', 'AO2', 'R']),
 ('aardema', ['AA0', 'R', 'D', 'EH1', 'M', 'AH0']),
 ('aardvark', ['AA1', 'R', 'D', 'V', 'AA2', 'R', 'K']),
 ('aaron', ['EH1', 'R', 'AH0', 'N']),
 ("aaron's", ['EH1', 'R', 'AH0', 'N', 'Z']),
 ('aarons', ['EH1', 'R', 'AH0', 'N', 'Z']),
 ('aaronson', ['EH1', 'R', 'AH0', 'N', 'S', 'AH0', 'N']),
 ('aaronson', ['AA1', 'R', 'AH0', 'N', 'S', 'AH0', 'N']),
 ("aaronson's", ['EH1', 'R', 'AH0', 'N', 'S', 'AH0', 'N', 'Z']),
 ("aaronson's", ['AA1', 'R', 'AH0', 'N', 'S', 'AH0', 'N', 'Z']),
 ('aarti', ['AA1', 'R', 'T', 'IY2']),
 ('aase', ['AA1', 'S']),
 ('aasen', ['AA1', 'S', 'AH0', 'N']),
 ('ab', ['AE1', 'B']),
 ('ab', ['EY1', 'B', 'IY1']),
 ('ababa', ['AH0', 'B', 'AA1', 'B', 'AH0']),
 ('ababa', ['AA1', 'B', 'AH0', 'B', 'AH0']),
 ('abacha', ['AE1', 'B', 'AH0', 'K', 'AH0']),
 ('aback', ['AH0', 'B', 'AE1', 'K']),
 ('abaco', ['AE1', 'B', 'AH0', 'K', 'OW2']),
 ('abacus', ['AE1', 'B', 'AH0', 'K', 'AH0', 'S']),
 ('abad', ['AH0', 'B', 'AA1', 'D']),
 ('abadaka', ['AH0', 'B', 'AE1', 'D', 'AH0', 'K', 'AH0']),
 ('abadi', ['AH0', 'B', 'AE1', 'D', 'IY0']),
 ('abadie', ['AH0', 'B', 'AE1', 'D', 'IY0']),
 ('abair', ['AH0', 'B', 'EH1', 'R']),
 ('abalkin', ['AH0', 'B', 'AA1', 'L', 'K', 'IH0', 'N']),
 ('abalone', ['AE2', 'B', 'AH0', 'L', 'OW1', 'N', 'IY0']),
 ('abalos', ['AA0', 'B', 'AA1', 'L', 'OW0', 'Z']),
 ('abandon', ['AH0', 'B', 'AE1', 'N', 'D', 'AH0', 'N']),
 ('abandoned', ['AH0', 'B', 'AE1', 'N', 'D', 'AH0', 'N', 'D']),
 ('abandoning', ['AH0', 'B', 'AE1', 'N', 'D', 'AH0', 'N', 'IH0', 'NG']),
 ('abandonment',
  ['AH0', 'B', 'AE1', 'N', 'D', 'AH0', 'N', 'M', 'AH0', 'N', 'T']),
 ('abandonments',
  ['AH0', 'B', 'AE1', 'N', 'D', 'AH0', 'N', 'M', 'AH0', 'N', 'T', 'S']),
 ('abandons', ['AH0', 'B', 'AE1', 'N', 'D', 'AH0', 'N', 'Z']),
 ('abanto', ['AH0', 'B', 'AE1', 'N', 'T', 'OW0']),
 ('abarca', ['AH0', 'B', 'AA1', 'R', 'K', 'AH0']),
 ('abare', ['AA0', 'B', 'AA1', 'R', 'IY0']),
 ('abascal', ['AE1', 'B', 'AH0', 'S', 'K', 'AH0', 'L']),
 ('abash', ['AH0', 'B', 'AE1', 'SH']),
 ('abashed', ['AH0', 'B', 'AE1', 'SH', 'T']),
 ('abate', ['AH0', 'B', 'EY1', 'T']),
 ('abated', ['AH0', 'B', 'EY1', 'T', 'IH0', 'D']),
 ('abatement', ['AH0', 'B', 'EY1', 'T', 'M', 'AH0', 'N', 'T']),
 ('abatements', ['AH0', 'B', 'EY1', 'T', 'M', 'AH0', 'N', 'T', 'S']),
 ('abates', ['AH0', 'B', 'EY1', 'T', 'S']),
 ('abating', ['AH0', 'B', 'EY1', 'T', 'IH0', 'NG']),
 ('abba', ['AE1', 'B', 'AH0']),
 ('abbado', ['AH0', 'B', 'AA1', 'D', 'OW0']),
 ('abbas', ['AH0', 'B', 'AA1', 'S']),
 ('abbasi', ['AA0', 'B', 'AA1', 'S', 'IY0']),
 ('abbate', ['AA1', 'B', 'EY0', 'T']),
 ('abbatiello', ['AA0', 'B', 'AA0', 'T', 'IY0', 'EH1', 'L', 'OW0']),
 ('abbe', ['AE1', 'B', 'IY0']),
 ('abbe', ['AE0', 'B', 'EY1']),
 ('abbenhaus', ['AE1', 'B', 'AH0', 'N', 'HH', 'AW2', 'S']),
 ('abbett', ['AH0', 'B', 'EH1', 'T']),
 ('abbeville', ['AE1', 'B', 'V', 'IH0', 'L']),
 ('abbey', ['AE1', 'B', 'IY0']),
 ("abbey's", ['AE1', 'B', 'IY0', 'Z']),
 ('abbie', ['AE1', 'B', 'IY0']),
 ('abbitt', ['AE1', 'B', 'IH0', 'T']),
 ('abbot', ['AE1', 'B', 'AH0', 'T']),
 ('abbotstown', ['AE1', 'B', 'AH0', 'T', 'S', 'T', 'AW1', 'N']),
 ('abbott', ['AE1', 'B', 'AH0', 'T']),
 ("abbott's", ['AE1', 'B', 'AH0', 'T', 'S']),
 ('abbottstown', ['AE1', 'B', 'AH0', 'T', 'S', 'T', 'AW1', 'N']),
 ('abboud', ['AH0', 'B', 'UW1', 'D']),
 ('abboud', ['AH0', 'B', 'AW1', 'D']),
 ('abbreviate', ['AH0', 'B', 'R', 'IY1', 'V', 'IY0', 'EY2', 'T']),
 ('abbreviated', ['AH0', 'B', 'R', 'IY1', 'V', 'IY0', 'EY2', 'T', 'AH0', 'D']),
 ('abbreviated', ['AH0', 'B', 'R', 'IY1', 'V', 'IY0', 'EY2', 'T', 'IH0', 'D']),
 ('abbreviates', ['AH0', 'B', 'R', 'IY1', 'V', 'IY0', 'EY2', 'T', 'S']),
 ('abbreviating',
  ['AH0', 'B', 'R', 'IY1', 'V', 'IY0', 'EY2', 'T', 'IH0', 'NG']),
 ('abbreviation',
  ['AH0', 'B', 'R', 'IY2', 'V', 'IY0', 'EY1', 'SH', 'AH0', 'N']),
 ('abbreviations',
  ['AH0', 'B', 'R', 'IY2', 'V', 'IY0', 'EY1', 'SH', 'AH0', 'N', 'Z']),
 ('abbruzzese', ['AA0', 'B', 'R', 'UW0', 'T', 'S', 'EY1', 'Z', 'IY0']),
 ('abbs', ['AE1', 'B', 'Z']),
 ('abby', ['AE1', 'B', 'IY0']),
 ('abco', ['AE1', 'B', 'K', 'OW0']),
 ('abcotek', ['AE1', 'B', 'K', 'OW0', 'T', 'EH2', 'K']),
 ('abdalla', ['AE2', 'B', 'D', 'AE1', 'L', 'AH0']),
 ('abdallah', ['AE2', 'B', 'D', 'AE1', 'L', 'AH0']),
 ('abdel', ['AE1', 'B', 'D', 'EH2', 'L']),
 ('abdella', ['AE2', 'B', 'D', 'EH1', 'L', 'AH0']),
 ('abdicate', ['AE1', 'B', 'D', 'AH0', 'K', 'EY2', 'T']),
 ('abdicated', ['AE1', 'B', 'D', 'AH0', 'K', 'EY2', 'T', 'AH0', 'D']),
 ('abdicates', ['AE1', 'B', 'D', 'AH0', 'K', 'EY2', 'T', 'S']),
 ('abdicating', ['AE1', 'B', 'D', 'IH0', 'K', 'EY2', 'T', 'IH0', 'NG'])]
```

In [4]:

```
from nltk.corpus import wordnet as wn
wn.synsets('automobile')
```

Out[4]:

```
[Synset('car.n.01'), Synset('automobile.v.01')]
```

In [5]:

```
wn.synset('car.n.01').lemma_names()
```

Out[5]:

```
['car', 'auto', 'automobile', 'machine', 'motorcar']
```

In [6]:

```
import nltk
from nltk.stem import PorterStemmer
stemmerporter = PorterStemmer()
stemmerporter.stem('happiness')
```

Out[6]:

```
'happi'
```

In [7]:

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

Out[7]:

```
'happy'
```

In [8]:

```
import nltk
from nltk.stem import SnowballStemmer
SnowballStemmer.languages
frenchStemmer = SnowballStemmer('french')
frenchStemmer.stem('parle')
```

Out[8]:

```
'parl'
```

In [9]:

```
from nltk.stem import PorterStemmer

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

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

In [10]:

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("dice"))
```

```
dice
```

In [11]:

```
print(lemmatizer.lemmatize("better", pos = 'a'))
```

```
good
```

In [12]:

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

```
be
```

In [18]:

```
#Chineese segmentation
import sys
!{sys.executable} -m pip install 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))
```

```
Requirement already satisfied: jieba in /anaconda3/lib/python3.7/site-packages
(0.42.1)

Building prefix dict from the default dictionary ...
Dumping model to file cache /var/folders/6p/mglm5phx3zv79b11m2f7wby80000gn/T/ji
eba.cache
Loading model cost 1.095 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 [ ]:

```

```

In [ ]:

```

```