Tokenization

Tokenization is the first step in NLP. It is the process of breaking strings into tokens which in turn are small structures or units. Tokenization involves three steps which are breaking a complex sentence into words, understanding the importance of each word with respect to the sentence and finally produce structural description on an input sentence.

In [1]:
```
import pandas as pd
import numpy as np
import nltk
import os
import nltk.corpus
```

In [2]:
```
# sample text for performing tokenization
text = "In Brazil they drive on the right-hand side of the road, Brazil has a large coastline on the eastern side of South America"
```

In [3]:
```
# importing word_tokenize from nltk
from nltk.tokenize import word_tokenize
# Passing the string text into word tokenize for breaking the sentences
token = word_tokenize(text)
token
```

Out[3]:
```
['In',
 'Brazil',
 'they',
 'drive',
 'on',
 'the',
 'right-hand',
 'side',
 'of',
 'the',
 'road',
 ',',
 'Brazil',
 'has',
 'a',
 'large',
 'coastline',
 'on',
 'the',
 'eastern',
 'side',
 'of',
 'South',
 'America']
```

Finding frequency distinct in the text

In [4]:
```
# Finding the frequency distinct in the tokens
# Importing FreqDist library from nltk and passing token into FreqDist
from nltk.probability import FreqDist
fdist = FreqDist(token)
fdist
```

Out[4]:
```
FreqDist({'the': 3, 'Brazil': 2, 'on': 2, 'side': 2, 'of': 2, 'In': 1, 'they': 1, 'drive': 1, 'right-hand': 1, 'road': 1, ...})
```

'the' is found 3 times in the text, 'Brazil' is found 2 times in the text, etc.

In [5]:
```
# To find the frequency of top 10 words
fdist1 = fdist.most_common(10)
fdist1
```

Out[5]:
```
[('the', 3),
 ('Brazil', 2),
 ('on', 2),
 ('side', 2),
 ('of', 2),
 ('In', 1),
 ('they', 1),
 ('drive', 1),
 ('right-hand', 1),
 ('road', 1)]
```

Stemming

Stemming usually refers to normalizing words into its base form or root form.

Here, we have the words waited, waiting and waits. Here the root word is 'wait'. There are two methods in Stemming namely, Porter Stemming (removes common morphological and inflectional endings from words) and Lancaster Stemming (a more aggressive stemming algorithm).

In [6]:
```
# Importing Porterstemmer from nltk library
# Checking for the word 'giving'
from nltk.stem import PorterStemmer
pst = PorterStemmer()
pst.stem("waiting")
```

Out[6]:
```
'wait'
```

In [7]:
```
# Checking for the list of words
stm = ["waited", "waiting", "waits"]
for word in stm :
    print(word+ ":" +pst.stem(word))
```

```
waited:wait
waiting:wait
waits:wait
```

In [8]:
```
# Importing LancasterStemmer from nltk
from nltk.stem import LancasterStemmer
lst = LancasterStemmer()
stm = ["giving", "given", "given", "gave"]
for word in stm :
    print(word+ ":" +lst.stem(word))
```

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

Lancaster is more aggressive than Porter stemmer

Lemmatization

In simpler terms, it is the process of converting a word to its base form. The difference between stemming and lemmatization is, lemmatization considers the context and converts the word to its meaningful base form, whereas stemming just removes the last few characters, often leading to incorrect meanings and spelling errors.

For example, lemmatization would correctly identify the base form of 'caring' to 'care', whereas, stemming would cutoff the 'ing' part and convert it to car.

Lemmatization can be implemented in python by using Wordnet Lemmatizer, Spacy Lemmatizer, TextBlob, Stanford CoreNLP

In [9]:
```
#nltk.download('wordnet')
```

In [10]:
```
# Importing lemmatizer library from nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

print("rocks :", lemmatizer.lemmatize("rocks"))
print("corpora :", lemmatizer.lemmatize("corpora"))

rocks : rock
corpora : corpus
```

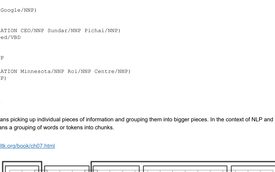
Stop Words

"Stop words" are the most common words in a language like "the", "a", "at", "for", "above", "on", "is", "all". These words do not provide any meaning and are usually removed from text. We can remove these stop words using nltk library

In [11]:
```
# importing stopwors from nltk library
from nltk import word_tokenize
from nltk.corpus import stopwords
a = set(stopwords.words("english"))
text = "Cristiano Ronaldo was born on February 5, 1985, in Funchal, Madeira, Portugal."
text1 = word_tokenize(text.lower())
print(text1)
stopwords = [x for x in text1 if x not in a]
print(stopwords)
```

```
['cristiano', 'ronaldo', 'was', 'born', 'on', 'february', '5', ',', '1985', ',', 'in', 'funchal', ',', 'madeira', ',', 'portugal', '.']
['cristiano', 'ronaldo', 'born', 'february', '5', ',', '1985', ',', 'funchal', ',', 'madeira', ',', 'portugal', '.']
```

Part of speech tagging (POS)



Part-of-speech tagging is used to assign parts of speech to each word of a given text (such as nouns, verbs, pronouns, adverbs, conjunction, adjectives, interjection) based on its definition and its context. There are many tools available for POS taggers and some of the widely used taggers are NLTK, Spacy, TextBlob, Stanford CoreNLP, etc.

In [12]:
```
#nltk.download('averaged_perceptron_tagger')
```

In [13]:
```
text = "vote to choose a particular man or a group (party) to represent them in parliament"
#Tokenize the text
tex = word_tokenize(text)
for token in tex:
    print(nltk.pos_tag([token]))
```

```
[('vote', 'NN')]
[('to', 'TO')]
[('choose', 'NN')]
[('a', 'DT')]
[('particular', 'JJ')]
[('man', 'NN')]
[('or', 'CC')]
[('a', 'DT')]
[('group', 'NN')]
[('(', '(')]
[('party', 'NN')]
[(')', ')')]
[('to', 'TO')]
[('represent', 'NN')]
[('them', 'PRP')]
[('in', 'IN')]
[('parliament', 'NN')]
```

Number Tag Description

1. CC Coordinating conjunction
2. CD Cardinal number
3. DT Determiner
4. EX Existential there
5. FW Foreign word
6. IN Preposition or subordinating conjunction
7. JJ Adjective
8. JJR Adjective, comparative
9. JJS Adjective, superlative
10. LS List item marker
11. MD Modal
12. NN Noun, singular or mass
13. NNS Noun, plural
14. NNP Proper noun, singular
15. NNPS Proper noun, plural
16. PDT Predeterminer
17. POS Possessive ending
18. PRP Personal pronoun
19. PRP$(dollar sign) Possessive pronoun
20. RB Adverb
21. RBR Adverb, comparative
22. RBS Adverb, superlative
23. RP Particle
24. SYM Symbol
25. TO to
26. UH Interjection
27. VB Verb, base form
28. VBD Verb, past tense
29. VBG Verb, gerund or present participle
30. VBN Verb, past participle
31. VBP Verb, non-3rd person singular present
32. VBZ Verb, 3rd person singular present
33. WDT Wh-determiner
34. WP Wh-pronoun
35. WP$(dollar sign) Possessive wh-pronoun
36. WRB Wh-adverb

https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Named entity recognition

It is the process of detecting the named entities such as the person name, the location name, the company name, the quantities and the monetary value.

In [14]:
```
#nltk.download('maxent_ne_chunker')
```

In [15]:
```
#nltk.download('words')
```

In [16]:
```
text = "Google's CEO Sundar Pichai introduced the new Pixel at Minnesota Roi Centre Event"
#importing chunk library from nltk
from nltk import ne_chunk
# tokenize and POS Tagging before doing chunk
token = word_tokenize(text)
tags = nltk.pos_tag(token)
chunk = ne_chunk(tags)
print(chunk)
```

```
(S
  (PERSON Google/NNP)
  's/NNP
  s/VBD
  (ORGANIZATION CEO/NNP Sundar/NNP Pichai/NNP)
  introduced/VBD
  the/DT
  new/JJ
  Pixel/NNP
  at/IN
  (ORGANIZATION Minnesota/NNP Roi/NNP Centre/NNP)
  Event/NNP)
```

Chunking

Chunking means picking up individual pieces of information and grouping them into bigger pieces. In the context of NLP and text mining, chunking means a grouping of words or tokens into chunks.

https://www.nltk.org/book/ch07.html

In [18]:
```
text = "We saw the yellow dog"
token = word_tokenize(text)
tags = nltk.pos_tag(token)
reg = "NP: {<DT>?<JJ>*<NN>}"
a = nltk.RegexpParser(reg)
result = a.parse(tags)
print(result)
```

```
(S We/PRP saw/VBD (NP the/DT yellow/JJ dog/NN))
```

In [ ]:
```

```