Entity Extraction

In this example, we will use NLTK for entity extraction.

- Firstly, install python environment
- Install NLTK: pip install nltk
- Download data distribution for NLTK. Install using NLTK downloader: `nltk.download()` . If cannot download using `nltk.download()` , try download manually from https://github.com/nltk/nltk_data/tree/gh-pages![image.png](attachment:image.png (https://github.com/nltk/nltk_data/tree/gh-pages![image.png](attachment:image.png)) or https://pan.baidu.com/s/1wONWpaa86_wnsIksKda8eQ (https://pan.baidu.com/s/1wONWpaa86_wnsIksKda8eQ) (code:tfon)
- Unzip the downloaded file to the following folder: `nltk.data.find(".")`
- Unzip each zip file in the ten folders: *chunkers, corpora, grammers, help, misc, models, sentiment, stemmers, taggers, tokenizers*

In [1]:
```
# import all packages
import nltk
from nltk import word_tokenize,pos_tag, ne_chunk
from nltk import Tree
```

In [2]:
```
# Tokenize sentence:
raw = """John was born in Liverpool, to Julia and Alfred Lennon"""
tokens = word_tokenize(raw)
tokens
```

Out[2]:
```
['John',
 'was',
 'born',
 'in',
 'Liverpool',
 ',',
 'to',
 'Julia',
 'and',
 'Alfred',
 'Lennon']
```

```
In [3]: # pos-tag of inputs
        tagged = nltk.pos_tag(tokens)
        print(tagged)
```

```
[('John', 'NNP'), ('was', 'VBD'), ('born', 'VBN'), ('in', 'IN'), ('Liverpool', 'NNP'), (',', ','), ('to', 'TO'), ('Ju
lia', 'NNP'), ('and', 'CC'), ('Alfred', 'NNP'), ('Lennon', 'NNP')]
```

If you want to know the detail information of each tag, use the following statements:

```
In [4]: nltk.help.upenn_tagset('NNP')
```

```
NNP: noun, proper, singular
    Motown Venneboerger Czestochwa Ranzer Conchita Trumplane Christos
    Oceanside Escobar Kreisler Sawyer Cougar Yvette Ervin ODI Darryl CTCA
    Shannon A.K.C. Meltex Liverpool ...
```

Chunking:

- Use `ne_chunk` provided by NLTK. `ne_chunk` needs part-of-speech annotations to add `NE` labels to the sentence. The output of the `ne_chunk` is a `nltk.Tree` object
- `ne_chunk` produces 2-level trees:
  - Nodes on Level-1: outsides any chunk
  - Nodes on Level-2: inside a chunk (the label of the chunk is denoted by the label of the subtree)

```
In [5]: chunks = ne_chunk(pos_tag(word_tokenize(raw)))
        print(chunks)
        chunks.draw()
```

```
(S
  (PERSON John/NNP)
  was/VBD
  born/VBN
  in/IN
  (GPE Liverpool/NNP)
  ,/,
  to/TO
  (GPE Julia/NNP)
  and/CC
  (PERSON Alfred/NNP Lennon/NNP))
```

Traverse the chunked tree structure to get each chunk and words inside each chunk:

```
In [6]: for i in chunks:
            print(i, type(i))
            if type(i) == Tree:
                print('Chunk detect!')
                chunk_phrase = []
                for token,pos in i.leaves():
                    print(token,pos)
```

```
(PERSON John/NNP) <class 'nltk.tree.Tree'>
Chunk detect!
John NNP
('was', 'VBD') <class 'tuple'>
('born', 'VBN') <class 'tuple'>
('in', 'IN') <class 'tuple'>
(GPE Liverpool/NNP) <class 'nltk.tree.Tree'>
Chunk detect!
Liverpool NNP
(',', ',') <class 'tuple'>
('to', 'TO') <class 'tuple'>
(GPE Julia/NNP) <class 'nltk.tree.Tree'>
Chunk detect!
Julia NNP
('and', 'CC') <class 'tuple'>
(PERSON Alfred/NNP Lennon/NNP) <class 'nltk.tree.Tree'>
Chunk detect!
Alfred NNP
Lennon NNP
```

Exercise1

Extract all named entities as well as its type/label

In [7]:
```
# Exercise1, define a function to extract all named enties together with labels
def get_labeled_chunks(text):
    # your implementation
    return label_entities
get_labeled_chunks(raw)
```

Out[7]:
```
{'John': 'PERSON',
 'Liverpool': 'GPE',
 'Julia': 'GPE',
 'Alfred Lennon': 'PERSON'}
```

Exercise2

Extract only *PERSON* entities

In [8]:
```
# Exercise2, extract all the entities of specific type
def get_type_chunks(text,label):
    # your implementation
    return entity
get_type_chunks(raw,'PERSON')
```

Out[8]:
```
['John', 'Alfred Lennon']
```

Exercise3: Noun phrase chunking

Define your own grammer for noun phrase chunking using `nltk.RegexpParser`

```
In [10]: def np_chunking(sentence):
             grammer = "NP: {<NN.*>+}"  # chunker rule(s), try think of more rules
             # your implementation
             return entity

         print(np_chunking("""the little dog barked at the cat"""))
         print(np_chunking("""Jonh was born in Liverpool, to Julia and Alfred Lennon"""))
```

```
['little dog', 'cat']
['Jonh', 'Liverpool', 'Julia', 'Alfred Lennon']
```