# Chunking and NER

## NLTK Ch. 7. Extracting Information from Text

# Outline

- Chunking
- NER
- Relation Extraction

# Information Extraction

- **entities** and **relationships** can be represented as a **tuple (entity, relation, entity)**
- "Which organizations operate in Atlanta?" could be translated as *[e1 for (e1, rel, e2) in locs if e2=='Atlanta']*
- We will look for specific kinds of information in text, e.g., **relation** between **organizations** and **locations**

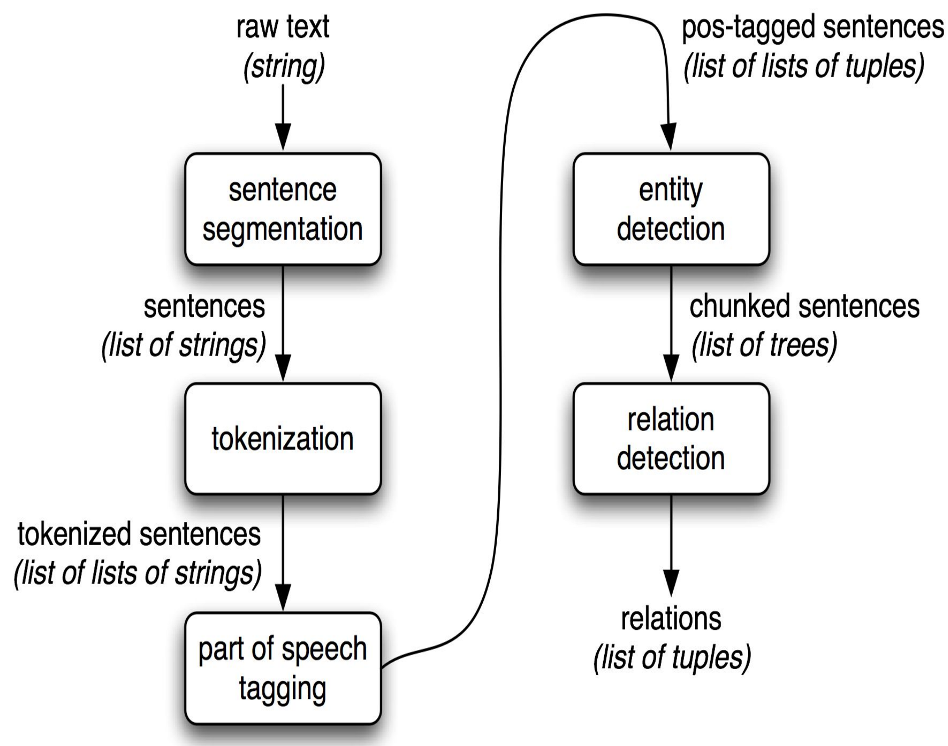
Snippet from nltk.corpus.ieer

The fourth Wells account moving to another agency is the packaged paper-products division of Georgia-Pacific Corp., which arrived at Wells only last fall. Like Hertz and the History Channel, it is also leaving for an Omnicom-owned agency, the BBDO South unit of BBDO Worldwide. BBDO South in Atlanta, which handles corporate advertising for Georgia-Pacific, will assume additional duties for brands like Angel Soft toilet tissue and Sparkle paper towels, said Ken Haldin, a spokesman for Georgia-Pacific in Atlanta.

Information Extraction

```
>>> locs = [('Omnicom', 'IN', 'New York'),
...        ('DDB Needham', 'IN', 'New York'),
...        ('Kaplan Thaler Group', 'IN', 'New York'),
...        ('BBDO South', 'IN', 'Atlanta'),
...        ('Georgia-Pacific', 'IN', 'Atlanta')]
>>> query = [e1 for (e1, rel, e2) in locs if e2=='Atlanta']
>>> print(query)
['BBDO South', 'Georgia-Pacific']
```

- **Information Extraction** means getting meaning from text
- Text is split into sentences using a **sentence** segmenter, and each sentence is subdivided into **words** using a tokenizer. Each sentence is tagged with **part-of-speech tags**, which are helpful in the next step, **named entity detection**. Finally, we use **relation detection** to search for likely relations between different entities in the text
- The system takes the text of a document as input, and generates a list of **(entity, relation, entity)** tuples
    - e.g., *the company Georgia-Pacific is located in Atlanta* ==> ***([ORG: 'Georgia-Pacific'] 'in' [LOC: 'Atlanta'])***.



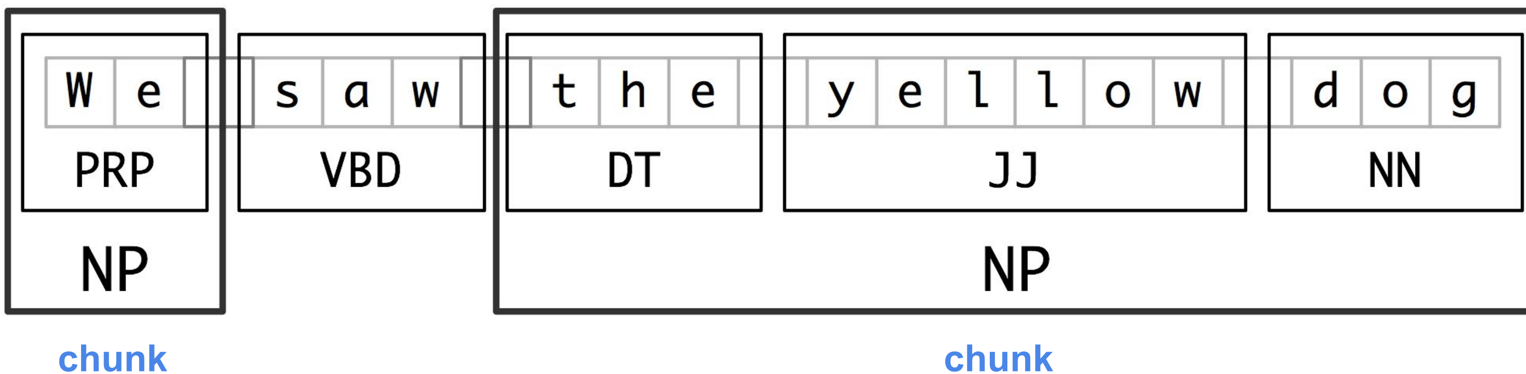

Simple Pipeline Architecture for Information Extraction

# Chunking

## (Shallow Parsing)

# Chunking

- For **entity detection**, we use **chunking**, which segments and labels token sequences
- Chunking usually selects a subset of the tokens, and the **pieces produced by a chunker do not overlap in the text**
- We will learn **regular expression** and **n-gram approaches to chunking**

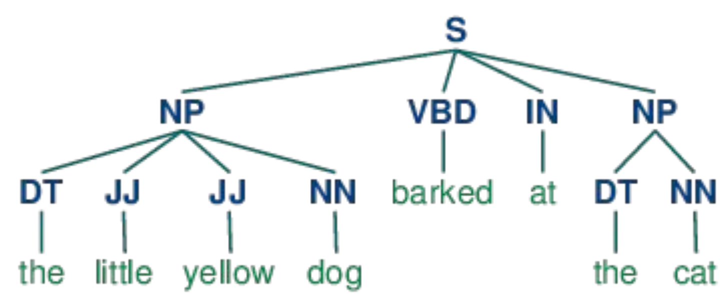


# Noun Phrase Chunking

- Chunks corresponding to **noun phrase**
- To create an NP-chunker, define a **chunk grammar**, consisting of rules that indicate how sentences should be chunked
- The rules use **tag patterns** to describe sequences of tagged words
- A tag pattern is a sequence of pos tags delimited using angle brackets, e.g. *<DT>?<JJ>*<NN>*

```
>>> sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),
... ("dog", "NN"), ("barked", "VBD"), ("at", "IN"),  ("the", "DT"),
("cat", "NN")]
>>> grammar = "NP: {<DT>?<JJ>*<NN>}"
>>> cp = nltk.RegexpParser(grammar)
>>> result = cp.parse(sentence)
>>> print(result)
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
>>> result.draw()
```

optional determiner (DT) followed by any number of adjectives (JJ) and a noun (NN)



# Tag Patterns

- Tag patterns are similar to regular expression patterns
- We can match these noun phrases using a refinement of the first tag pattern i.e. _<DT>?<JJ.*>*<NN.*>+_

Noun phrases from the Wall Street Journal

another/DT sharp/JJ dive/NN
trade/NN figures/NNS
any/DT new/JJ policy/NN measures/NNS
earlier/JJR stages/NNS
Panamanian/JJ dictator/NN Manuel/NNP Noriega/NNP

# Chunking with Regular Expressions

- Chunking rules are applied in turn, successively updating the chunk structure
- Once all of the rules have been invoked, the resulting chunk structure is returned
- If a tag pattern matches at overlapping locations, the leftmost match takes precedence

```
grammar = r"""
  NP: {<DT|PP\$>?<JJ>*<NN>}
      # chunk determiner/possessive, adjectives and noun
      {<NNP>+}    # chunk sequences of proper nouns
"""
>>> cp = nltk.RegexpParser(grammar)
>>> sentence = [("Rapunzel", "NNP"), ("let", "VBD"), ("down", "RP"),  ("her", "PP$"), ("long", "JJ"), ("golden", "JJ"), ("hair","NN")]
>>> print(cp.parse(sentence))
(S
  (NP Rapunzel/NNP)
  let/VBD
  down/RP
  (NP her/PP$ long/JJ golden/JJ hair/NN))
>>> nouns = [("money", "NN"), ("market", "NN"), ("fund", "NN")]
>>> grammar = "NP: {<NN><NN>}  # Chunk two consecutive nouns"
>>> cp = nltk.RegexpParser(grammar)
>>> print(cp.parse(nouns))
(S (NP money/NN market/NN) fund/NN)
```

# Exploring Text Corpora

- Interrogate a tagged corpus to extract phrases matching a particular sequence of pos tags with a chunker

```
>>> cp = nltk.RegexpParser('CHUNK: {<V.*> <TO> <V.*>}')
>>> brown = nltk.corpus.brown
>>> for sent in brown.tagged_sents():
...     tree = cp.parse(sent)
...     for subtree in tree.subtrees():
...         if subtree.label() == 'CHUNK': print(subtree)
...
(CHUNK combined/VBN to/TO achieve/VB)
(CHUNK continue/VB to/TO place/VB)
(CHUNK serve/VB to/TO protect/VB)
(CHUNK wanted/VBD to/TO wait/VB)
(CHUNK allowed/VBN to/TO place/VB)
(CHUNK expected/VBN to/TO become/VB)
...
(CHUNK seems/VBZ to/TO overtake/VB)
(CHUNK want/VB to/TO buy/VB)
```

# Chinking

- Chinking is the process of **removing a sequence of tokens from a chunk**
- Define a chink to be a sequence of tokens that is not included in a chunk. In the following example, barked/VBD at/IN is a chink

[ the/DT little/JJ yellow/JJ dog/NN ] barked/VBD at/IN [ the/DT cat/NN ]

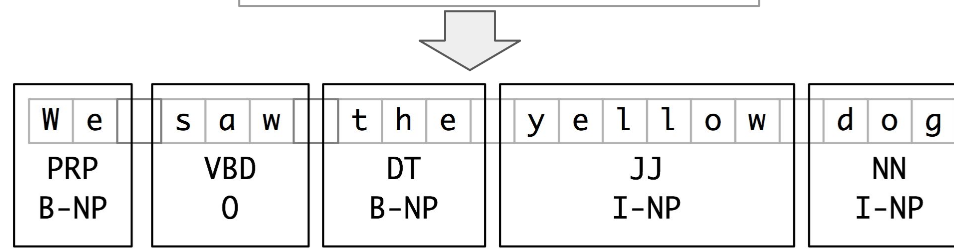
```
grammar = r"""
  NP:
    {<.*>+}          # Chunk everything
    }<VBD|IN>+{      # Chink sequences of VBD and IN
  """
sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),
       ("dog", "NN"), ("barked", "VBD"), ("at", "IN"),  ("the",
"DT"), ("cat", "NN")]
cp = nltk.RegexpParser(grammar)

>>> print(cp.parse(sentence))
 (S
   (NP the/DT little/JJ yellow/JJ dog/NN)
   barked/VBD
   at/IN
   (NP the/DT cat/NN))
```

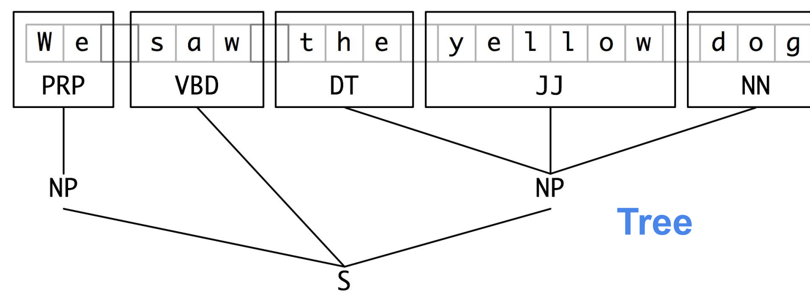
# Representing Chunks: Tags vs Trees

- **IOB tags** tags each token
  - **B (begin)** marks the **beginning of a chunk**
  - **I (inside)** tags **subsequent tokens** within the chunk
  - **O (outside)** tags all other tokens
- B and I tags are suffixed with the chunk type, e.g. B-NP, I-NP
- **Tree** representations have the benefit that each chunk is a constituent that can be manipulated directly

We PRP B-NP
saw VBD O
the DT B-NP
yellow JJ I-NP
dog NN I-NP

**IOB Tags**




**Tree**

# Evaluation of Chunking

# Developing and Evaluating Chunkers

- Convert IOB format into an NLTK tree using a chunked corpus
- How to score the accuracy of a chunker relative to a corpus

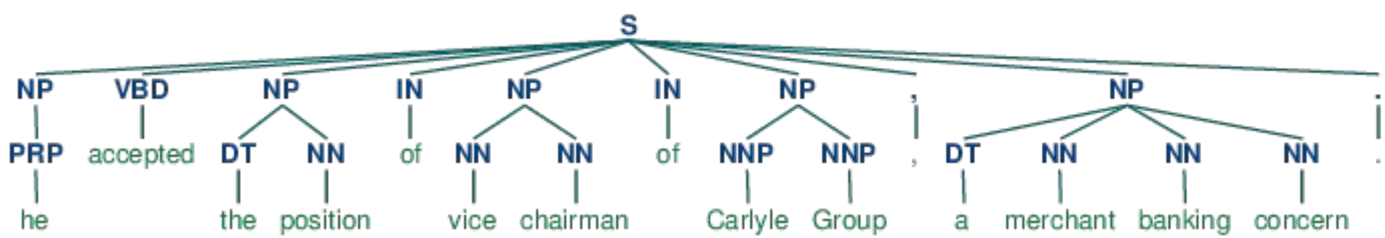
# Reading IOB Format and the CoNLL 2000 Corpus

- Wall Street Journal text that has been tagged then chunked using the IOB notation

  ```
  he PRP B-NP
  accepted VBD B-VP
  the DT B-NP
  position NN I-NP
  ```

- *chunk.**conllstr2tree()*** builds a tree representation

```
>>> text = '''
... he PRP B-NP
... accepted VBD B-VP
... the DT B-NP
... position NN I-NP
... of IN B-PP
... vice NN B-NP
... chairman NN I-NP
... of IN B-PP
... Carlyle NNP B-NP
... Group NNP I-NP
... , , O
... a DT B-NP
... merchant NN I-NP
... banking NN I-NP
... concern NN I-NP
... . . O
... '''
>>> nltk.chunk.conllstr2tree(text, chunk_types=['NP']).draw()
```



# Evaluation and Baselines

- A **baseline** parser creates no chunks
- A simple chunker that looks for tags beginning with letters that characterise **noun phrase tags** (e.g. CD, DT, and JJ).

```
>>> from nltk.corpus import conll2000
>>> cp = nltk.RegexpParser("") # 아무것도 추출 안함 O
>>> test_sents = conll2000.chunked_sents('test.txt',
chunk_types=['NP']) #NP chunk만 추출하여 테스트셋 구축
>>> print(cp.evaluate(test_sents))
ChunkParse score:
    IOB Accuracy:  43.4%
    Precision:      0.0%
    Recall:         0.0%
    F-Measure:      0.0%
```

*more than a third of the words are tagged with O*

```
>>> grammar = r"NP: {<[CDJNP].*>+}"
>>> cp = nltk.RegexpParser(grammar)
>>> print(cp.evaluate(test_sents))
ChunkParse score:
    IOB Accuracy:  87.7%
    Precision:     70.6%
    Recall:        67.8%
    F-Measure:     69.2%
```

# test_sents

```
[Tree('S', [Tree('NP', [('Rockwell', 'NNP'), ('International', 'NNP'), ('Corp.', 'NNP')]), Tree('NP', [("'s", 'POS'), ('Tulsa', 'NNP'), ('unit', 'NN')]), ('said', 'VBD'), Tree('NP', [('it', 'PRP')]), ('signed', 'VBD'), Tree('NP', [('a', 'DT'), ('tentative', 'JJ'), ('agreement', 'NN')]), ('extending', 'VBG'), Tree('NP', [('its', 'PRP$'), ('contract', 'NN')]), ('with', 'IN'), Tree('NP', [('Boeing', 'NNP'), ('Co.', 'NNP')]), ('to', 'TO'), ('provide', 'VB'), Tree('NP', [('structural', 'JJ'), ('parts', 'NNS')]), ('for', 'IN'), Tree('NP', [('Boeing', 'NNP')]), Tree('NP', [("'s", 'POS'), ('747', 'CD'), ('jetliners', 'NNS')]), ('.', '.')]), Tree('S', [Tree('NP', [('Rockwell', 'NNP')]), ('said', 'VBD'), Tree('NP', [('the', 'DT'), ('agreement', 'NN')]), ('calls', 'VBZ'), ('for', 'IN'), Tree('NP', [('it', 'PRP')]), ('to', 'TO'), ('supply', 'VB'), Tree('NP', [('200', 'CD'), ('additional', 'JJ'), ('so-called', 'JJ'), ('shipsets', 'NNS')]), ('for', 'IN'), Tree('NP', [('the', 'DT'), ('planes', 'NNS')]), ('.', '.')]), ...]

[Tree('S', [('``', '``'), ('Improving', 'VBG'), Tree('NP', [('profitability', 'NN')]), ('of', 'IN'), Tree('NP', [('U.S.', 'NNP'), ('operations', 'NNS')]), ('is', 'VBZ'), Tree('NP', [('an', 'DT'), ('extremely', 'RB'), ('high', 'JJ'), ('priority', 'NN')]), ('in', 'IN'), Tree('NP', [('the', 'DT'), ('company', 'NN')]), ('.', '.'), ("''", "''")]), Tree('S', [('To', 'TO'), ('focus', 'VB'), ('on', 'IN'), Tree('NP', [('its', 'PRP$'), ('global', 'JJ'), ('consumer-products', 'NNS'), ('business', 'NN')]), (',', ','), Tree('NP', [('Colgate', 'NNP')]), ('sold', 'VBD'), Tree('NP', [('its', 'PRP$'), ('Kendall', 'NNP'), ('health-care', 'NN'), ('business', 'NN')]), ('in', 'IN'), Tree('NP', [('1988', 'CD')]), ('.', '.')]), ...]
```

# Recursion in Linguistic Structure

## Building Nested Structure with Cascaded Chunkers

- To build chunk structures of arbitrary depth, create a multi-stage chunk grammar containing **recursive rules**
- The example shows the result **missing the VP** headed by *saw*

```
>>> grammar = r"""
  NP: {<DT|JJ|NN.*>+}     # Chunk sequences of DT, JJ, NN
  PP: {<IN><NP>}          # Chunk prepositions followed by NP
  VP: {<VB.*><NP|PP|CLAUSE>+$}
      # Chunk verbs and their arguments
  CLAUSE: {<NP><VP>}      # Chunk NP, VP
  """

>>> cp = nltk.RegexpParser(grammar)
>>> sentence = [("Mary", "NN"), ("saw", "VBD"), ("the", "DT"),
("cat", "NN"), ("sit", "VB"), ("on", "IN"), ("the", "DT"), ("mat", "NN")]
>>> print(cp.parse(sentence))
(S
  (NP Mary/NN)
  saw/VBD
  (CLAUSE
    (NP the/DT cat/NN)
    (VP sit/VB (PP on/IN (NP the/DT mat/NN)))))
```

# Loop

- Use the argument **loop** to specify the number of times the set of patterns should be run

```
>>> cp = nltk.RegexpParser(grammar, loop=3)
>>> print(cp.parse(sentence))
(S
  (CLAUSE
    (NP John/NNP)
    (VP
      thinks/VBZ
      (CLAUSE
        (NP Mary/NN)
        (VP
          saw/VBD
          (CLAUSE
            (NP the/DT cat/NN)
            (VP sit/VB (PP on/IN (NP the/DT mat/NN)))))))))
```

```
>>> sentence = [("John", "NNP"), ("thinks", "VBZ"), ("Mary", "NN"),
("saw", "VBD"), ("the", "DT"), ("cat", "NN"), ("sit", "VB"),
("on", "IN"), ("the", "DT"), ("mat", "NN")]
>>> print(cp.parse(sentence))
(S
  (NP John/NNP)
  thinks/VBZ
  (NP Mary/NN)
  saw/VBD
  (CLAUSE
    (NP the/DT cat/NN)
    (VP sit/VB (PP on/IN (NP the/DT mat/NN)))))

>>> cp = nltk.RegexpParser(grammar, loop=2)
>>> print(cp.parse(sentence))
(S
  (NP John/NNP)
  thinks/VBZ
  (CLAUSE
    (NP Mary/NN)
    (VP
      saw/VBD
      (CLAUSE
        (NP the/DT cat/NN)
        (VP sit/VB (PP on/IN (NP the/DT mat/NN)))))))
```

# 구문 분석

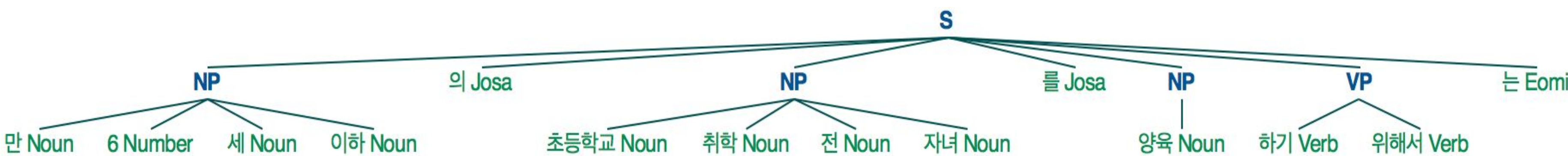
형태소 분석된 결과와
nltk.chunk.regexp.RegexpParser 를 이용

명사구(NP): 명사가 연속적으로 등장한 후 접미사(suffix)가 선택적으로 붙은 경우
유사한 방식으로 동사구(VP)와 형용사구(AP)를 정의

```
# Print whole tree
(S
  (NP 만/Noun 6/Number 세/Noun 이하/Noun)
  의/Josa
  (NP 초등학교/Noun 취학/Noun 전/Noun
자녀/Noun)
  를/Josa
  (NP 양육/Noun)
  (VP 하기/Verb 위해서/Verb)
  는/Eomi)
```

```
import konlpy
import nltk

# POS tag a sentence
sentence = '만 6세 이하의 초등학교 취학 전 자녀를 양육하기
위해서는'
words = konlpy.tag.Okt().pos(sentence)

# Define a chunk grammar, or chunking rules, then chunk
grammar = """
NP: {<N.*>*<Suffix>?}   # Noun phrase
VP: {<V.*>*}            # Verb phrase
AP: {<A.*>*}            # Adjective phrase
"""
parser = nltk.RegexpParser(grammar)
chunks = parser.parse(words)
print("# Print whole tree")
print(chunks.pprint())

# Display the chunk tree
chunks.draw()
```

```
# Print whole tree
(S
  (NP 만/Noun 6/Number 세/Noun 이하/Noun)
  의/Josa
  (NP 초등학교/Noun 취학/Noun 전/Noun
자녀/Noun)
  를/Josa
  (NP 양육/Noun)
  (VP 하기/Verb 위해서/Verb)
  는/Eomi)
```



# NER
# (Named Entity Recognition)

# Named Entity Recognition

- Named entities are **definite noun phrases** that refer to specific types of individuals, such as **organizations, persons, dates**
- **Named entity recognition (NER)** identifies all textual mentions of the named entities, breaking down into two sub-tasks: **identifying the boundaries of the NE**, and **identifying its type**
- While NER is frequently used for **identifying relations**, it can also contribute to other tasks, e.g., **Question Answering (QA)**

| **NE Type** | **Examples** |
|---|---|
| **ORGANIZATION** | *Georgia-Pacific Corp.*, *WHO* |
| **PERSON** | *Eddy Bonte*, *President Obama* |
| **LOCATION** | *Murray River*, *Mount Everest* |
| DATE | *June*, *2008-06-29* |
| TIME | *two fifty a m*, *1:30 p.m.* |
| MONEY | *175 million Canadian Dollars*, *GBP 10.40* |
| PERCENT | *twenty pct*, *18.75 %* |
| FACILITY | *Washington Monument*, *Stonehenge* |
| GPE((geo-political entity) | *South East Asia*, *Midlothian* |

# Example: Q&A

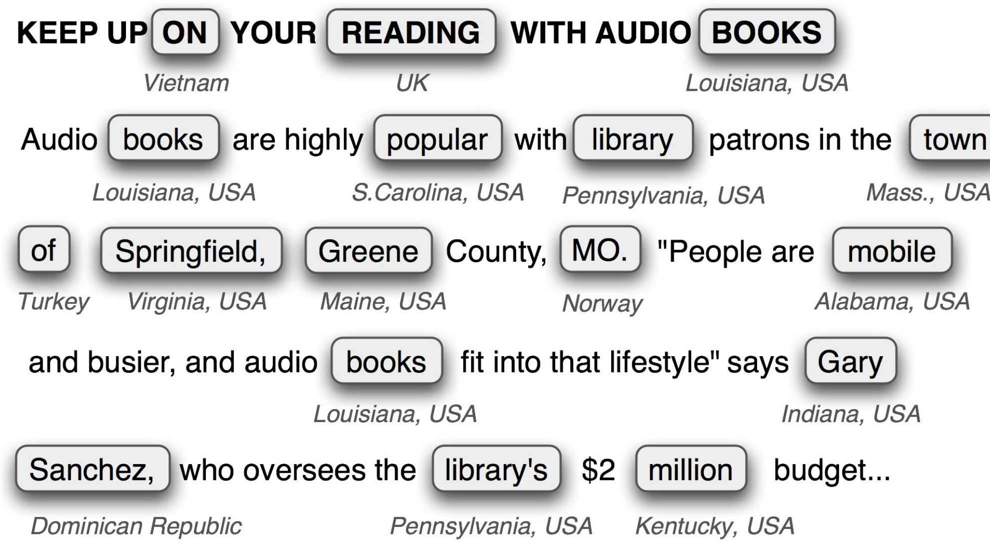
Question: *Who was the first President of the US?*

The **Washington** Monument is the most prominent structure in **Washington**, D.C. and one of the city's early attractions. It was built in honor of George **Washington**, who led the country to independence and then became its first President.

- Answer should be of the form *X was the first President of the US*, where X is a noun phrase referring to a named entity of type PERSON

# Identifying Named Entities

- Look up each word in a list of names
    - Locations use a gazetteer, or geographical dictionary
    - A *gazetteer* is a list of geographic names, together with their geographic locations and other descriptive information, e.g., Alexandria Gazetteer or Getty Gazetteer
- It gets harder in the case of names for people or organizations
- Named entity terms are ambiguous. e.g., *May* and *North* can be DATE and LOCATION, respectively, but could be part of a PERSON; Christian Dior looks like a PERSON but is more likely to be of ORGANIZATION




Location Detection by Simple Lookup for a News Story: Looking up every word in a gazetteer is error-prone

# nltk.ne_chunk()

- **nltk.ne_chunk()**: a classifier function that has **been trained** to recognize named entities
- **binary=True**, named entities are tagged as NE; otherwise, the classifier adds category labels such as **PERSON**, **ORGANIZATION**, and **GPE**

```
>>> sent = nltk.corpus.treebank.tagged_sents()[22]

>>> print(nltk.ne_chunk(sent))
(S
  The/DT
  (GPE U.S./NNP)
  is/VBZ
  one/CD
  ...
  according/VBG
  to/TO
  (PERSON Brooke/NNP T./NNP Mossman/NNP)
  ...)
```

# NER with SpaCy

- Trained on the [OntoNotes 5 corpus] recognizes 18 types
- Example code can be found at https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da
- Trained on Wikipedia recognizes 4 types (PER, LOC, ORG, MIS) https://spacy.io/api/annotation#named-entities

| TYPE | DESCRIPTION |
|---|---|
| PERSON | People, including fictional. |
| NORP | Nationalities or religious or political groups. |
| FAC | Buildings, airports, highways, bridges, etc. |
| ORG | Companies, agencies, institutions, etc. |
| GPE | Countries, cities, states. |
| LOC | Non-GPE locations, mountain ranges, bodies of water. |
| PRODUCT | Objects, vehicles, foods, etc. (Not services.) |
| EVENT | Named hurricanes, battles, wars, sports events, etc. |
| WORK_OF_ART | Titles of books, songs, etc. |
| LAW | Named documents made into laws. |
| LANGUAGE | Any named language. |
| DATE | Absolute or relative dates or periods. |
| TIME | Times smaller than a day. |
| PERCENT | Percentage, including "%". |
| MONEY | Monetary values, including unit. |
| QUANTITY | Measurements, as of weight or distance. |
| ORDINAL | "first", "second", etc. |
| CARDINAL | Numerals that do not fall under another type. |

# Stanford NLP Group

## A New Multi-Turn, Multi-Domain, Task-Oriented Dialogue Dataset

DRIVER I need to find the time and parties attending my optometrist appointment.
CAR I have 3 appointments scheduled, with Alex, your sister, and Jeff. Which are you referring to?
DRIVER I want to know about the one that Alex is joining me at.
CAR That optometrist appointment is at 4 pm.
DRIVER Thanks.
CAR No problem.

https://nlp.stanford.edu/blog/a-new-multi-turn-multi-domain-task-oriented-dialogue-dataset/

# Entities ('event', 'time', 'date', 'party', 'room', 'agenda', 'location', 'weekly_time', 'temperature', 'weather_attribute', 'traffic_info', 'poi_type', 'poi', 'distance')

**'event'**: ['taking medicine', 'conference', 'dinner', 'lab appointment', 'yoga activity', 'tennis activity', 'doctor appointment', 'meeting', 'swimming activity', 'optometrist appointment', 'football activity', 'dentist appointment'],

**'time'**: ['1pm', '2pm', '3pm', '4pm', '5pm', '6pm', '8pm', '9am', '10am', '11am', '7pm'],

**'date':** ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday', 'the 1st', 'the 2nd', 'the 3rd', 'the 4th', 'the 5th', '...],

**'party'**: ['sales team', 'HR', 'executive team', 'infrastructure team', 'vice president', 'boss', 'management', 'Tom', 'Jon', 'Martha', 'brother', 'mother', 'aunt', 'father', 'sister', 'Marie', 'Alex', 'Jeff', 'Ana']

**, 'room':** ['conference room 100', 'conference room 102', 'conference room 50']

**'agenda'**: ['go over budget', 'discuss the merger', 'discuss the company picnic', 'go over quarterly report', 'onboard new members', 'discuss dress code', 'sign the NDA'],

**'location'**: ['oakland', 'san mateo', 'san jose', 'grand rapids', 'boston', 'new york', 'cleveland', 'downtown chicago', 'seattle', 'los angeles', 'san francisco', 'redwood city', 'menlo park', 'exeter', 'mountain view', 'compton', 'inglewood', 'durham', 'manhattan', 'fresno', 'alameda', 'alhambra', 'atherton', 'brentwood', 'camarillo', 'carson', 'corona', 'danville'],

**'weekly_time'**: ['today', 'tomorrow', 'week', 'weekend', 'next few days', 'two day', 'next week'],

**'temperature'**: [20, 30, 40, 50, 60, 70, 80, 90, 100],

**'weather_attribute'**: ['lowest temperature', 'highest temperature', 'overcast', 'snow', 'stormy', 'hail', 'hot', 'rain', 'cold', 'clear skies', 'cloudy', 'warm', 'windy', 'foggy', 'humid', 'frost', 'blizzard', 'drizzle', 'dry', 'dew', 'misty'],

**'traffic_info'**: ['no traffic', 'moderate traffic', 'heavy traffic', 'road block nearby', 'car collision nearby'],

**'poi_type'**: ['friends house', 'home', 'coffee or tea place', 'chinese restaurant', 'pizza restaurant', 'grocery store', 'rest stop', 'shopping center', 'parking garage', 'gas station', 'hospital', 'certain address']

**'poi'**: [{'address': '593 Arrowhead Way', 'poi': "Chef Chu's", 'type': 'chinese restaurant'}, {'address': '394 Van Ness Ave', 'poi': 'Coupa', 'type': 'coffee or tea place'},

# Conll 2003 corpus

- Dataset link: https://raw.githubusercontent.com/Franck-Dernoncourt/NeuroNER/master/neuroner/data/conll2003/en/train.txt
- Tagset: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
  - LOC: location
  - ORG: organization
  - PER: person
  - MISC: Miscellaneous
- LSTM을 이용한 개체명 인식 : https://wikidocs.net/24682

```
EU NNP B-NP B-ORG
rejects VBZ B-VP O
German JJ B-NP B-MISC
call NN I-NP O
to TO B-VP O
boycott VB I-VP O
British JJ B-NP B-MISC
lamb NN I-NP O
. . O O

Peter NNP B-NP B-PER
Blackburn NNP I-NP I-PE
```

# ieer corpus (Information Extraction and Entity Recognition Corpus)

'ieer': ['LOCATION', 'ORGANIZATION', 'PERSON', 'DURATION', 'DATE', 'CARDINAL', 'PERCENT', 'MONEY', 'MEASURE'],

참고
Ieer: https://www.nltk.org/_modules/nltk/corpus/reader/ieer.html
Sem: https://www.nltk.org/_modules/nltk/sem/relextract.html

```
>>> from nltk.corpus import ieer
>>> ieer.fileids()
['APW_19980314', 'APW_19980424', 'APW_19980429',
 'NYT_19980315', 'NYT_19980403', 'NYT_19980407']
>>> docs = ieer.parsed_docs('APW_19980314')
>>> print(docs[0])
<IEERDocument APW19980314.0391: 'Kenyans protest tax hikes'>
>>> print(docs[0].docno)
APW19980314.0391
>>> print(docs[0].doctype)
NEWS STORY
>>> print(docs[0].date_time)
03/14/1998 10:36:00
>>> print(docs[0].headline)
(DOCUMENT Kenyans protest tax hikes)
>>> print(docs[0].text)
(DOCUMENT
  (LOCATION NAIROBI)
  ,
  (LOCATION Kenya)
  (
  (ORGANIZATION AP)
  )
  _
  (CARDINAL Thousands)
  of
  laborers,
  ...
  on
  (DATE Saturday)
  ...)
```

# Relation Extraction

- Triples of the form (X, α, Y), where X and Y are **named entities**, and α is the **relation between X and Y**
- Use regular expressions
- Example searches for strings that contain the word **in**. (?!\b.+ing\b) is a negative lookahead assertion that disregards strings such as where *in* is followed by a gerund
- More usages at [http://www.nltk.org/howto/relextract.html](http://www.nltk.org/howto/relextract.html)
- ieer corpus at [http://www.nltk.org/howto/corpus.html](http://www.nltk.org/howto/corpus.html)

```
IN = re.compile(r'.*\bin\b(?!\b.+ing)')
for doc in nltk.corpus.ieer.parsed_docs('NYT_19980315'):
    for rel in nltk.sem.extract_rels('ORG', 'LOC', doc,
                                     corpus='ieer', pattern = IN):
        print(nltk.sem.rtuple(rel))
```

```
[ORG: 'WHYY'] 'in' [LOC: 'Philadelphia']
[ORG: 'McGlashan &AMP; Sarrail'] 'firm in' [LOC: 'San Mateo']
[ORG: 'Freedom Forum'] 'in' [LOC: 'Arlington']
[ORG: 'Brookings Institution'] ', the research group in' [LOC: 'Washington']
[ORG: 'Idealab'] ', a self-described business incubator based in' [LOC: 'Los
Angeles']
[ORG: 'Open Text'] ', based in' [LOC: 'Waterloo']
[ORG: 'WGBH'] 'in' [LOC: 'Boston']
[ORG: 'Bastille Opera'] 'in' [LOC: 'Paris']
[ORG: 'Omnicom'] 'in' [LOC: 'New York']
```

# Role Extraction

Extract roles as **X, of (the) Y pattern**

IEER: has_role(PER, ORG) -- raw rtuples:
==============================================
[PER: 'Kivutha Kibwana'] ', of the' [ORG: 'National Convention Assembly']
[PER: 'Boban Boskovic'] ', chief executive of the' [ORG: 'Plastika']
[PER: 'Annan'] ', the first sub-Saharan African to head the' [ORG: 'United Nations']
[PER: 'Kiriyenko'] 'became a foreman at the' [ORG: 'Krasnoye Sormovo']
[PER: 'Annan'] ', the first sub-Saharan African to head the' [ORG: 'United Nations']
[PER: 'Mike Godwin'] ', chief counsel for the' [ORG: 'Electronic Frontier Foundation']
[PER: 'Robert Mergess'] ', the co-director of the' [ORG: 'Berkeley Center for Law and Technology']
[PER: 'Jack Balkin'] ", director of the school's program. ``What happened at" [ORG: 'Yale']
[PER: 'William Gale'] ', an economist at the' [ORG: 'Brookings Institution']

```
from nltk.corpus import ieer
trace = 0
roles = """ (.*(
analyst|chair(wo)?man|commissioner|counsel|director|economist|editor|executive|foreman|governor|head|lawyer|leader|librarian).*)|manager|partner|president|producer|professor|researcher|spokes(wo)?man|writer|
,\sof\sthe?\s*  # "X, of (the) Y" """
ROLES = re.compile(roles, re.VERBOSE)

for file in ieer.fileids():
    for doc in ieer.parsed_docs(file):
        lcon = rcon = False
        if trace:
            print(doc.docno)
            print("=" * 15)
            lcon = rcon = True
        for rel in nltk.sem.relextract.extract_rels('PER', 'ORG', doc, corpus='ieer', pattern=ROLES):
            print(nltk.sem.relextract.rtuple(rel, lcon=lcon, rcon=rcon))
```

# Role Extraction from Treebank corpus

```
print("1500 Sentences from Penn Treebank, as processed by NLTK NE Chunker")
print("=" * 45)
ROLE = re.compile(r'.*(chairman|president|trader|scientist|economist|analyst|partner).*')
rels = []
for i, sent in enumerate(nltk.corpus.treebank.tagged_sents()[:1500]):
    sent = nltk.ne_chunk(sent)
    rels = nltk.sem.relextract.extract_rels('PER', 'ORG', sent, corpus='ace', pattern=ROLE, window=7)
    for rel in rels:
        print('{0:<5}{1}'.format(i, nltk.sem.relextract.rtuple(rel)))
```

```
1    [PER: 'Vinken/NNP'] 'is/VBZ chairman/NN of/IN' [ORG: 'Elsevier/NNP']
254  [PER: 'Shugart/NNP'] ',/, currently/RB chairman/NN of/IN' [ORG: 'Seagate/NNP Technology/NNP']
325  [PER: 'George/NNP Foot/NNP'] ',/, a/DT managing/VBG partner/NN at/IN' [ORG: 'Newgate/NNP Management/NNP Associates/NNP']
331  [PER: 'Michael/NNP Porter/NNP'] ',/, an/DT analyst/NN at/IN' [ORG: 'Smith/NNP Barney/NNP']
391  [PER: 'Elliott/NNP Platt/NNP'] ',/, an/DT economist/NN at/IN' [ORG: 'Donaldson/NNP']
```

More about relation extraction at
[https://www.youtube.com/watch?v=pO3Jsr31s_Q&list=PLoROMvodv4rObpMCir6rNNUIFAn56Js20&index=7](https://www.youtube.com/watch?v=pO3Jsr31s_Q&list=PLoROMvodv4rObpMCir6rNNUIFAn56Js20&index=7)

# Summary

- Information extraction systems search **unrestricted** text for specific types of entities and relations, and use them to populate **databases**. These databases can be used to find answers for specific questions.
- Information extraction begins by segmenting, tokenizing, and part-of-speech tagging the text. The resulting data is then searched for specific types of **entity**. Finally, the system looks at **entities that are mentioned near one another** in the text, and tries to determine whether specific **relationships** hold between those entities.
- **Entity Recognition is often performed using chunkers**, which segment multi-token sequences, and label them with the appropriate entity type**, ORGANIZATION, PERSON, LOCATION**, DATE, TIME, MONEY, and GPE.
- **Chunkers** can be constructed using **rule-based systems**, such as the RegexpParser class provided by NLTK; or using **machine learning techniques**
- **Relation Extraction** can be performed using either rule-based systems which typically look for specific patterns in the text that connect entities and the intervening words; or using machine-learning systems which attempt to learn such patterns from a corpus.