DIGITAL TALENT SCHOLARSHIP

KOMINFO

DIDUKUNG OLEH : Microsoft

Penambangan Teks berbasis NLP dengan NLTK

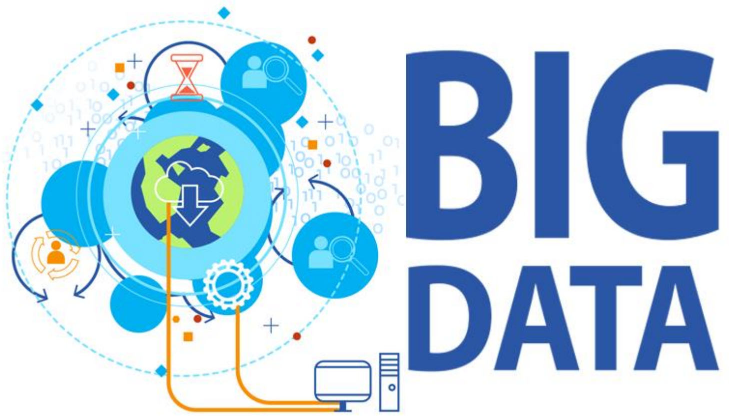
Supeno Mardi

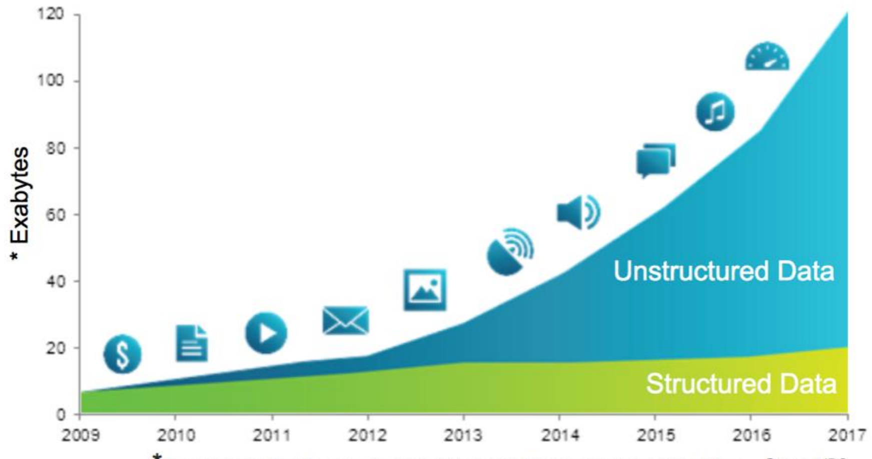
The Human Languages

6500 Languages

The Human Languages

The 21st Century

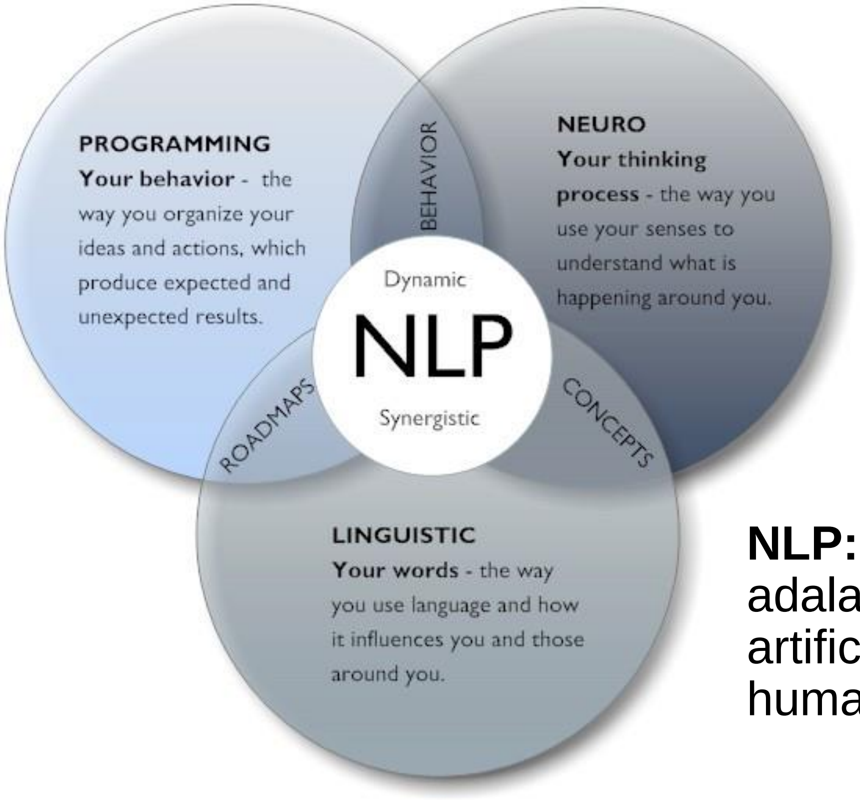




*1 exabyte = 1,000 petabytes =1 million terabytes = 1 billion gigabytes **Source:** IDC

Text Mining and NLP

Text Mining/Text Analytics adalah
proses mendapatkan informasi informasi yang bermakna dari teks bahasa alami


PROGRAMMING
Your behavior - the way you organize your ideas and actions, which produce expected and unexpected results.
NEURO
Your thinking process - the way you use your senses to understand what is happening around you.
LINGUISTIC
Your words - the way you use language and how it influences you and those around you.
BEHAVIOR
ROADMAPS
CONCEPTS
Dynamic
NLP
Synergistic

NLP: Natural Language Processing adalah bagian computer science dan artificial intelligence yang menangani human laguages.

Aplikasi NLP

Sentimental Analysis

Speech Recognition

Chatbot

Machine Translation

Aplikasi NLP

Speell Checker

Speech Recognition

Keyword Searching

Machine Translation

Bagian riset NLP

Natural Language Understanding

- Mapping the given input into natural languages into useful representation.
- Analyzing those aspect of the language.

Natural Language Generation

- The process of producing meaningful phrases and sentences.
- With the form of NLP from internal representation.

Komponen NLP

- Tokenization
- Stemming
- Lemmatization
- POS Tags
- Named Entity Recognition
- Chunking

Getting Started with NLTK

- Import NLTK module

```
>>> import os
>>> import nltk
>>> import nltk.corpus
```

- Import NLTK sample data

```
>>> print(os.listdir(nltk.data.find("corpo
ra")))
```

- Select gutenberg file:

```
>>> nltk.corpus.gutenberg.fileids()
```

Getting Started with NLTK

- Create *hamer* variable and show it

```
>>> hamlet=nltk.corpus.gutenberg.words('shake
speare-hamlet.txt')
>>> print(hamlet)
```

- Create *hamer* variable and show it

```
>>> for word in hamlet[:500]:
>>>     print(word, sep = ' ', end = ' ')
```

Tokenization

- Processing strings into tokens.
- Which in turn are small structure or units that can be used for tokenization.

- Example:
    - From:
      Tokenization is the first step in NLP

    - To:
      Tokenization **is** **the** **first** **step** **in** **NLP**

Tokenization

- Set sample paragraph of word:

```
>>> AI="""According to the father of Artificial
Intelligence, John McCarthy, it is “The science and
engineering of making intelligent machines, especially
intelligent computer programs”.

Artificial Intelligence is a way of making a computer, a
computer-controlled robot, or a software think
intelligently, in the similar manner the intelligent humans
think.

AI is accomplished by studying how human brain thinks, and
how humans learn, decide, and work while trying to solve a
problem, and then using the outcomes of this study as a
basis of developing intelligent software and systems."""
```

Tokenization

- Check the type of variable:

```
>>> type(AI)
```

- From NLTK import word_tokenizer

```
>>> from nltk.tokenize import word_tokenize
```

- Tokenize the words, print and check length

```
>>> AI_tokens = word_tokenize(AI)

>>> print(AI_tokens)

>>> len(AI_tokens)
```

Tokenization

- Import FreqDist from NLTK

```
>>> from nltk.probability import FreqDist
```

- Count all the words in paragraph:

```
>>> fdist = FreqDist()
>>> for word in AI_tokens:
>>>     fdist[word.lower()]+=1
>>> print(fdist)
```

- Show top 10 most common token

```
>>> fdist_top10 = fdist.most_common(10)
>>> fdist_top10
```

Tokenization

- Import blankline_tokenize

```
>>> from nltk.tokenize import blankline_tokenize
```

- Count blank paragraph

```
>>> AI_blank=blankline_tokenize(AI)
```

- Show the length of blank paragraph

```
>>> len(AI_blank)
```

Tokenization

- Import blankline_tokenize

```
>>> from nltk.tokenize import blankline_tokenize
```

- Count blank paragraph

```
>>> AI_blank=blankline_tokenize(AI)
```

- Show the length or number of blank paragraph

```
>>> len(AI_blank)
```

Tokenization

Important parts of tokenization:

- Bigram: Token of two consecutive written words
- Trigram: Token of three consecutive written words
- Ngram: Tokens of any number of consecutive written words

Tokenization

- Import bigrams, trigrams, ngrams

```
>>> from nltk.util import bigrams, trigrams, ngrams
```

- Implementation of bigrams, trigrams and etc.

```
>>> string = "The best and most beautiful things in
the world cannot be seen or even touched, they must
be felt with heart"

>>> quotes_tokens = nltk.word_tokenize(string)

>>> quotes_bigrams = list(nltk.bigrams(quotes_to
kens))

>>> quotes_trigrams = list(nltk.trigrams(quotes_to
kens))

>>> quotes_ngrams = list(nltk.ngrams(quotes_tokens,
5))
```

Stemming

- Normalize words into its base form or root form.

- From words:
  - ***Affectation***
  - ***Affects***
  - ***Affections***
  - ***Affected***
  - ***Affection***
  - ***Affecting***

- Originate from single root word:
  - ***Affect***

Stemming

- Import PorterStemmer

```
>>> from nltk.stem import PorterStemmer
```

- Stemming with PorterStemmer.

```
>>> pst=PorterStemmer()
>>> pst.stem("having")
>>> words_to_stem=["give","giving","given","gave"]
>>> for words in words_to_stem:
>>>     print(words+ ": " +pst.stem(words))
```

Stemming

- Other stemmer in NLTK.
  - **LancasterStemmer** is more agresive than **Porterstemmer**.
  - It usually for counting the used words.
  - **SnowBostonStemmer** which are using to specify the language are using.

Lemmatization

- The consideration morphological analysis of the words, it's necessary to have a detail dictionary which algorithm can look through to link the form back to original word itself or the root word which is also known as **lemma**.
- Step of Lemmatization:
  - Group together different inflected forms of a word, called Lemma.
  - Somehow similar to **Stemming**, as it maps several words into one common root.
  - Output of Lemmatization is a proper word.

- For example:
  - Lemmatiser should map **gone**, **going** and **went**.
  - Into **Go**.

Lemmatization

- Import wordnet and WordNetLemmatizer

```
>>> from nltk.stem import wordnet
>>> from nltk.stem import WordNetLemmatizer
```

- Stemming with PorterStemmer.

```
>>> words_to_lemm=["give","giving","given","gave"]
>>> word_lem=WordNetLemmatizer()
>>> for words in words_to_lemm:
>>>     print(words+ ": " +word_lem.lemmatize(words))
```

POS Tags

- **Stop Words**
  - Several words in the English or other languages such as “i, me, my, myself, above, below and etc”.
  - Which are very useful in the formation of sentence and without it the sentence wouldn’t make any sense.
  - But this words do not provide any help in the NLP.
  - NLTK has the list of stop words and we can just import it.

POS Tags

- Import stop words

```
>>> from nltk.corpus import stopwords
```

- List of english stop words

```
>>> stopwords.words('english')
```

- Check length of english stopwords

```
>>> len(stopwords.words('english'))
```

POS Tags

- Remove stop words

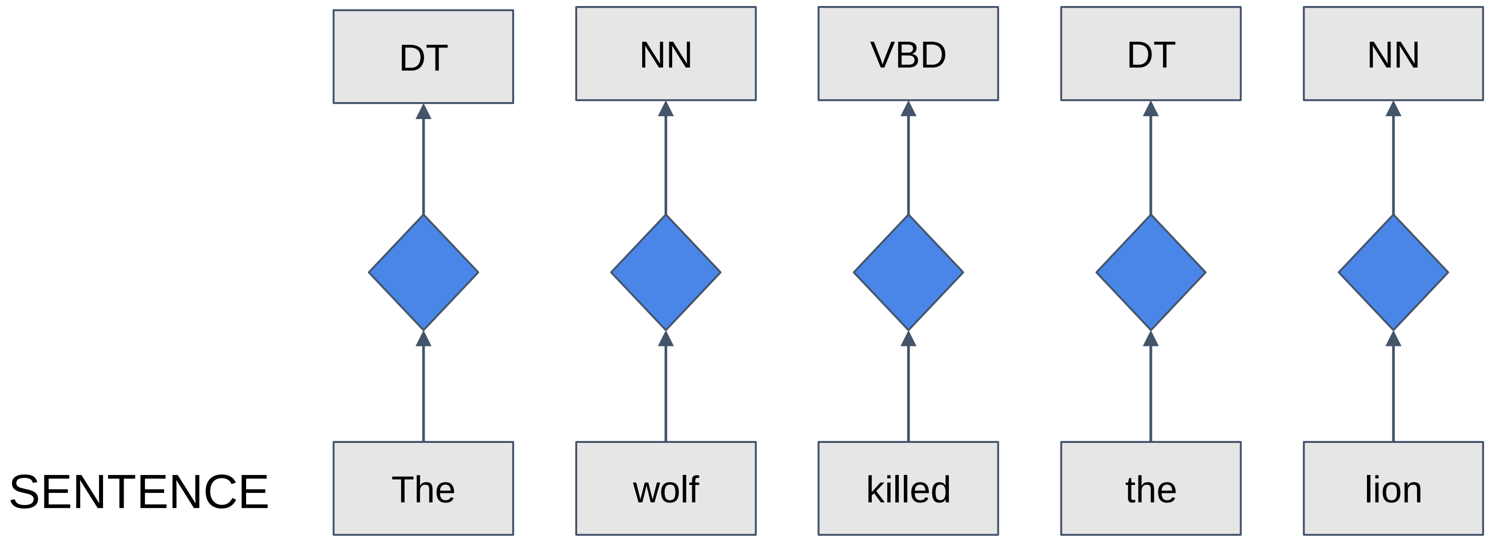
```
>>> import re
>>> punctuation=re.compile(r'[-.?!.:;()|0-9]')
>>> post_punctuation=[]
>>> for words in AI_tokens:
>>>     word=punctuation.sub("", words)
>>>     if len(word) > 0:
>>>         post_punctuation.append(word)
>>> post_punctuation
```

- Check length of words after remove the stop words

```
>>> len(post_punctuation)
```

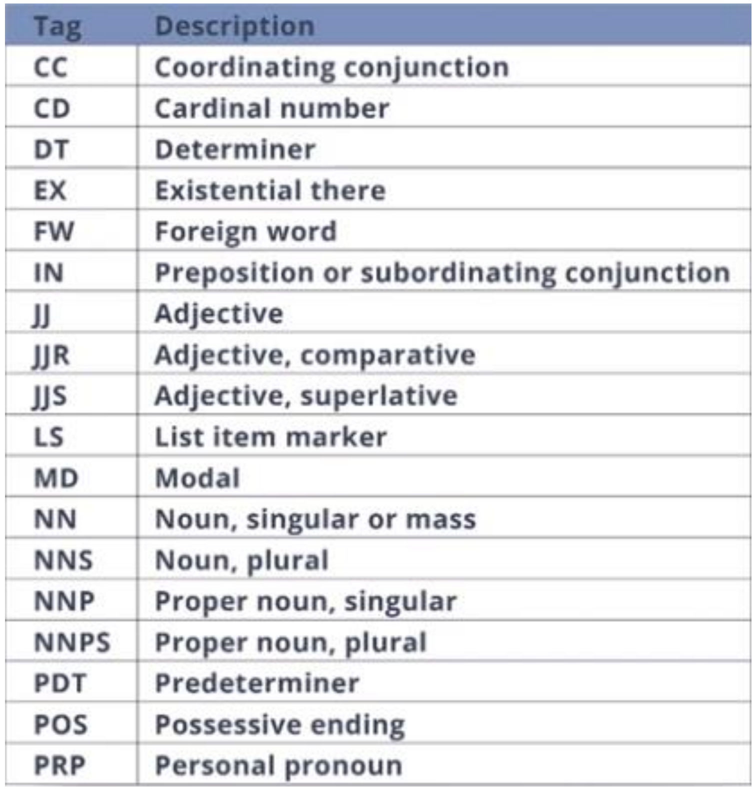
POS Tags

- The grammatically type of the word is referred to POS Tags or part of speech such as verb, noun, adjective, adverb and etc.
- It indicate how a word function in meaning as well as grammatically within the sentence.

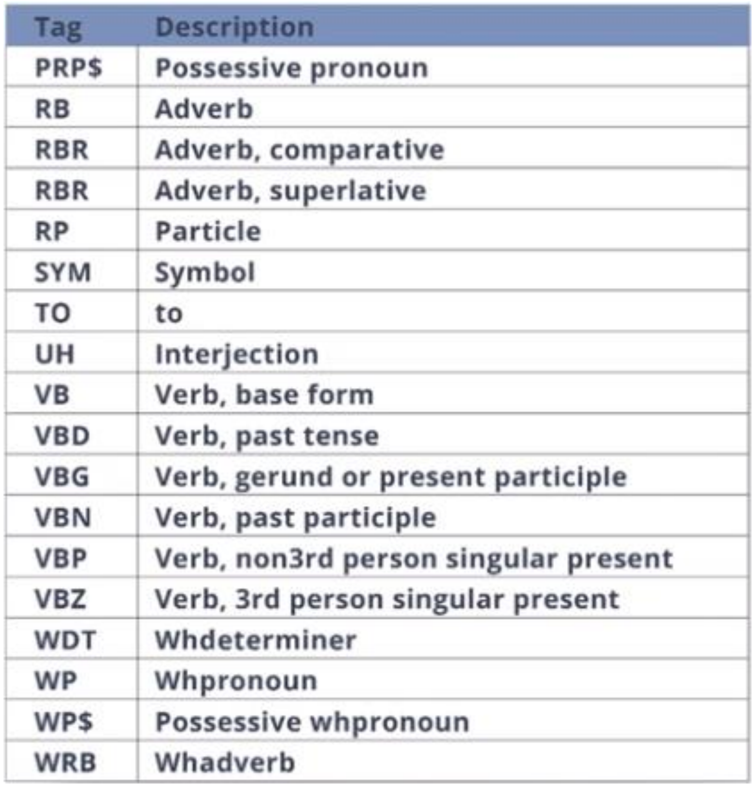


POS Tags

- NLTK POS Tags for part of speech



Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun



Tag	Description
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBR	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Whdeterminer
WP	Whpronoun
WP$	Possessive whpronoun
WRB	Whadverb

POS Tags

- Part of speech example 1

```
>>> sent = "Timothy is a natural when it comes to
drawing"
>>> sent_tokens = word_tokenize(sent)
>>> for token in sent_tokens:
>>>     print(nltk.pos_tag([token]))
>>> sent2 = "John is eating a delicious cake"
```

- Part of speech example 2

```
>>> sent_tokens2 = word_tokenize(sent2)
>>> for token in sent_tokens2:
>>>     print(nltk.pos_tag([token]))
```

POS Tags

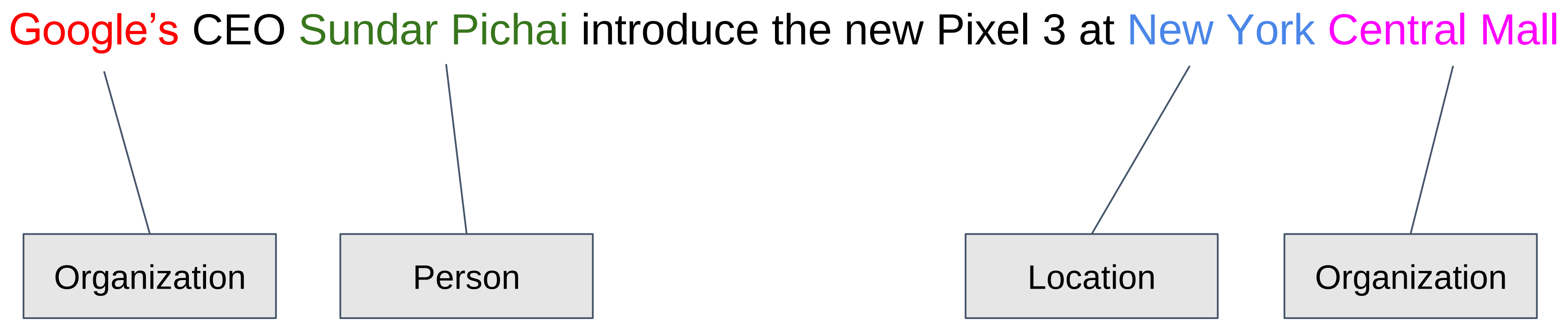
- Other example:

  “Google” something on the internet

- In this case **google** is used as a verb.
- The solution is **“Named Entity Recognition”** in the next capter.

Named Entity Recognition

- It’s a process of detecting the named entities such as the person name, the company names, quantity or the location name.
- The steps of NER:
    - The noun phrase identification.
    - The phrase classification.
    - Entity disambiguation.


Google’s CEO Sundar Pichai introduce the new Pixel 3 at New York Central Mall
Organization
Person
Location
Organization

Named Entity Recognition

- **Import Named Entity Library**

```
>>> from nltk import ne_chunk
```

- **Recognize Named Entity**

```
>>> NE_sent="The US President stays in the WHITE
HOUSE"
```

- **Tokenization**

```
>>> NE_tokens=word_tokenize(NE_sent)
```

- **Get POS Tags**

```
>>> NE_tags=nltk.pos_tag(NE_tokens)
```

- **Chunk and show the POS Tags**

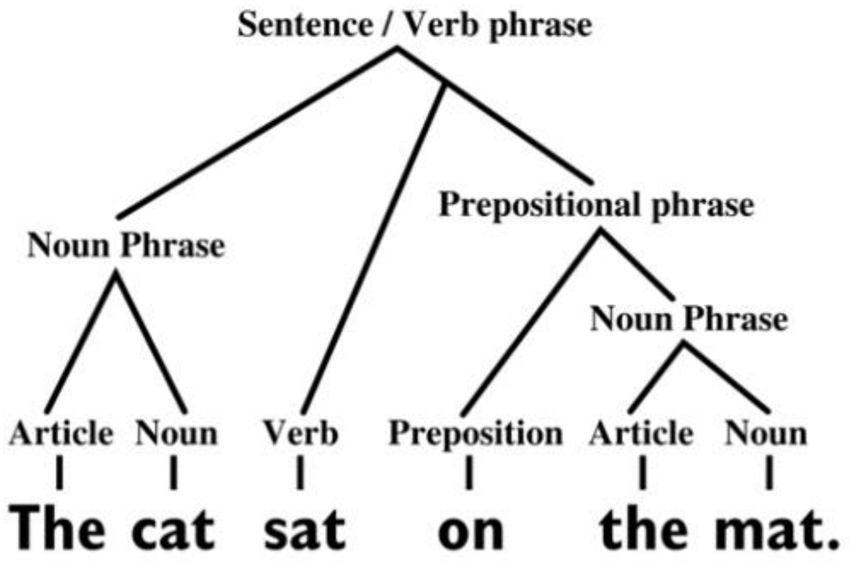
```
>>> NE_NER=ne_chunk(NE_tags)

>>> print(NE_NER)
```

Syntax Tree

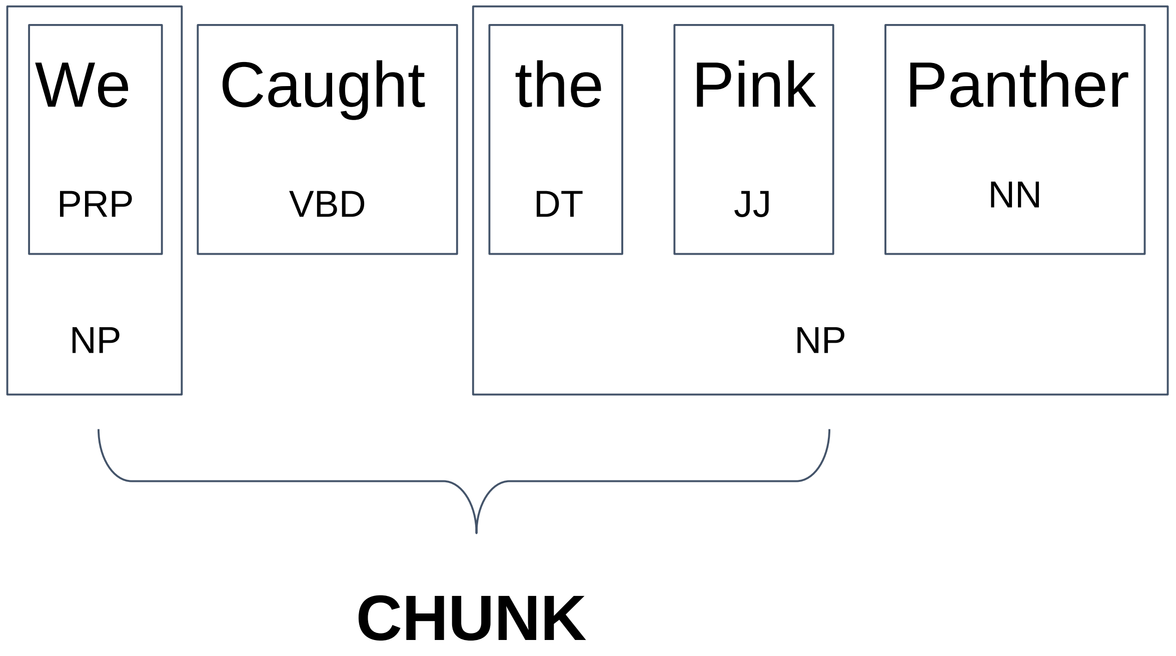
- Is a representation of **syntactic structure of sentences** or strings.

Basic constituent structure analysis of a sentence:



Chunking

- Picking up ***Individual*** pieces of information and ***Grouping*** them into bigger Pieces.



Chunking

- **Set word to chunk**

```
>>> new="The big cat ate the little mouse who was
after fresh cheese"
```

- **Tokenize and give POS Tags**

```
>>> new_tokens=nltk.pos_tag(word_tokenize(new))

>>> new_tokens
```

- **Create grammar from noun phrase**

```
>>> grammar_np=r"NP: {<DT>?<JJ>*<NN>}"

>>> chunk_parser=nltk.RegexpParser(grammar_np)
```

- **Get chunk result**

```
>>> chunk_result=chunk_parser.parse(new_tokens)

>>> chunk_result
```

Terima Kasih