

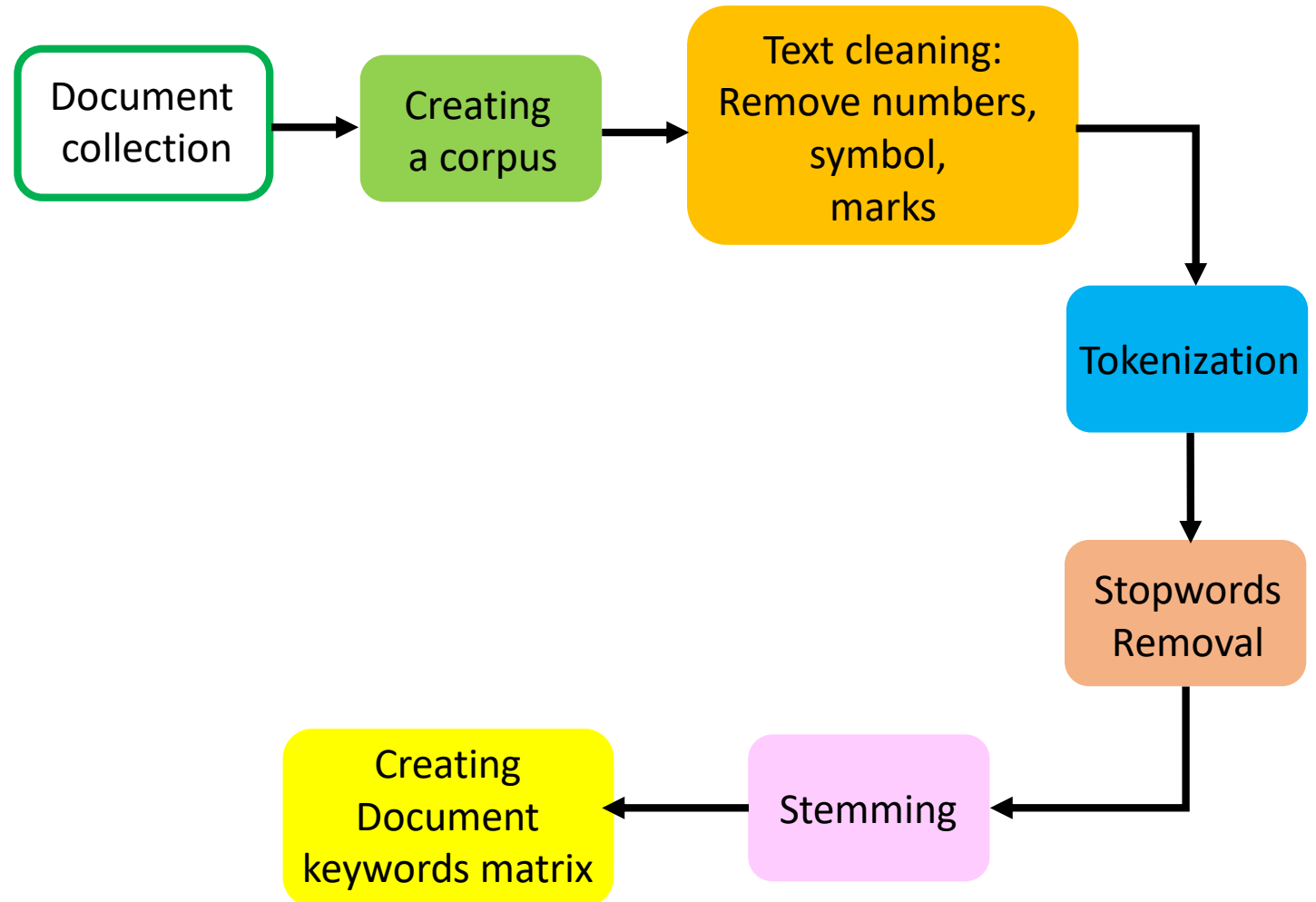
# Text Mining Tutorial 1:

---

## Pre-processing

# Pre-processing steps

- Tokenization
- Stop words removal
- Term frequency
- Stemming
- Normalization
- Lemmatization
- Part-of-speech tagging



# Tokenization

---

- **Tokenization:** Tokenization is the process of breaking the text up into separate units called tokens, which can be individual words, phrases, or whole sentences. In some cases, punctuation and special characters (symbols like %, &, \$) are discarded. The tokens usually become the input for the processes like parsing and text mining.

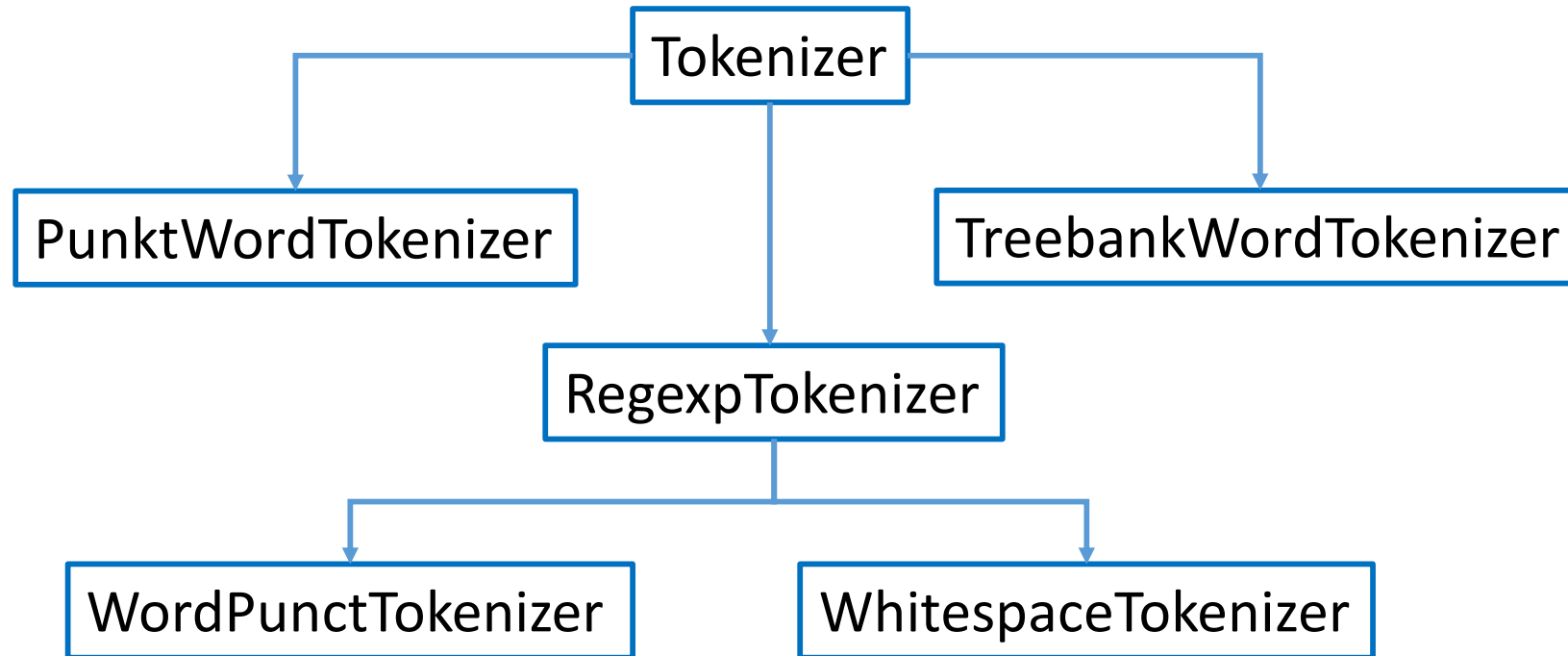
“This is a sample”

Tokenization

This is a sample

# Tokenization

---



# Word Tokenization

## Code

```
In [8]: from nltk.tokenize import word_tokenize
```

```
In [9]: text = "Hello everyone. Welcome to Text Mining class with Ghaluh. You are studying preprocessing now."  
word_tokenize(text)
```

## Output

```
Out[9]: ['Hello',  
         'everyone',  
         '.',  
         'Welcome',  
         'to',  
         'Text',  
         'Mining',  
         'class',  
         'with',  
         'Ghaluh',  
         '.',  
         'You',  
         'are',  
         'studying',  
         'preprocessing',  
         'now',  
         '.']
```

### How *word\_tokenize* works?

`word_tokenize()` function is a wrapper function that calls `tokenize()` on an instance of the `TreebankWordTokenizer` class.

# Sentence Tokenization

## Code

```
In [1]: from nltk.tokenize import sent_tokenize
```

```
In [3]: text = "Hello everyone. Welcome to Text Mining class with Ghaluh. You are studying preprocessing now."  
sent_tokenize(text)
```

## Output

```
Out[3]: ['Hello everyone.',  
        'Welcome to Text Mining class with Ghaluh.',  
        'You are studying preprocessing now.']
```

**How does `sent_tokenize` work?** The `sent_tokenize` function uses an instance of `PunktSentenceTokenizer` from the `nltk.tokenize.punkt` module, which is already been trained and thus very well knows to mark the end and beginning of sentence at what characters and punctuation.

# Tokenize sentence of different language

---

## Code

```
In [7]: spanish_tokenizer = nltk.data.load('tokenizers/punkt/PY3/spanish.pickle')  
  
text = 'Hola amigo. Estoy bien.'  
spanish_tokenizer.tokenize(text)
```

## Output

```
Out[7]: ['Hola amigo.', 'Estoy bien.']
```

# TreebankWordTokenizer

## Code

```
In [10]: from nltk.tokenize import TreebankWordTokenizer
```

```
In [11]: tokenizer = TreebankWordTokenizer()  
tokenizer.tokenize(text)
```

## Output

```
Out[11]: ['Hello',  
          'everyone.',  
          'Welcome',  
          'to',  
          'Text',  
          'Mining',  
          'class',  
          'with',  
          'Ghaluh.',  
          'You',  
          'are',  
          'studying',  
          'preprocessing',  
          'now',  
          '.']
```

### How TreebankWordTokenizer works?

These tokenizers work by separating the words using punctuation and spaces. And as mentioned in the code outputs above, it doesn't discard the punctuation, allowing a user to decide what to do with the punctuations at the time of pre-processing.



# PunktWordTokenizer

## Code

```
In [4]: import nltk.data
```

```
In [6]: tokenizer = nltk.data.load('tokenizers/punkt/PY3/english.pickle')  
tokenizer.tokenize(text)
```

## Output

```
Out[6]: ['Hello everyone.',  
         'Welcome to Text Mining class with Ghaluh.',  
         'You are studying preprocessing now.']
```

Type this if not installed yet → `import nltk`  
`nltk.download('punkt')`

# PunktWordTokenizer

---

## Code

```
from nltk.tokenize import PunktWordTokenizer

tokenizer = PunktWordTokenizer()
tokenizer.tokenize("Let's see how it's working.")
```

## Output

```
['Let', "'s", 'see', 'how', 'it', "'s", 'working', '.']
```

**Note:** It doesn't separate the punctuation from the words.

# WordPunctTokenizer

---

## Code

```
In [5]: from nltk.tokenize import WordPunctTokenizer
```

```
In [6]: tokenizer = WordPunctTokenizer()  
tokenizer.tokenize("Let's see how it's working.")
```

## Output

```
Out[6]: ['Let', "'", 's', 'see', 'how', 'it', "'", 's', 'working', '.']
```

**Note:** It separates the punctuation from the words.

# Stop words removal

- **Stop words** are actually the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc)
- Examples of a few stop words in English are “the”, “a”, “an”, “so”, “what”, etc.
- **Purpose:** By removing these words, we remove the low-level information from our text in order to give more focus to the important information

Exception



Ex: Perform Sentiment Analysis

**Movie Review:** “The movie was not good at all”

**Applying stop words removal:** ~~The movie was not good at all~~

**After stop words removal:** “movie good”

# Stop words removal: code example

## Code

```
import nltk
from nltk.corpus import stopwords
sw_nltk = stopwords.words('english')
print(sw_nltk)
```

## Output

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'y  
ourself', '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', 'a  
n', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'b  
etween', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of  
f', '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', 'ar  
en', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "have  
n't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should  
n't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Type this if not installed yet → `from nltk.corpus import stopwords`  
`nltk.download('stopwords')`


# Stop words removal: as dimensional reduction

## Code

```
text = "I saw alittle dog near the apple tree. But the dog ran away when he saw me."  
words = [word for word in text.split() if word.lower() not in sw_nltk]  
new_text = " ".join(words)  
print(new_text)  
print("Old length: ", len(text))  
print("New length: ", len(new_text))
```

## Output

```
saw alittle dog near apple tree. dog ran away saw me.  
Old length: 75  
New length: 53
```

 *Reduce the total number of text.*

**Additional** → Lowercase significantly helps to get a consistency of expected output.

# Stop words removal: in different library

## NLTK

```
import nltk  
  
print(len(sw_nltk))
```

executed in 2ms, finished 01:15:53 2023-03-06

179

## Gensim

```
import gensim  
from gensim.parsing.preprocessing import remove_stopwords, STOPWORDS  
print(len(STOPWORDS))
```

executed in 51ms, finished 01:27:41 2023-03-06

337

## Scikit-learn

```
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS  
print(len(ENGLISH_STOP_WORDS))
```

executed in 2ms, finished 01:30:53 2023-03-06

318

## Spacy

```
import spacy  
  
en = spacy.load('en_core_web_sm')  
sw_spacy = en.Defaults.stop_words  
print(len(sw_spacy))
```

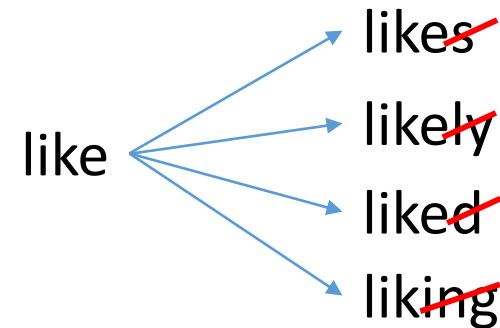
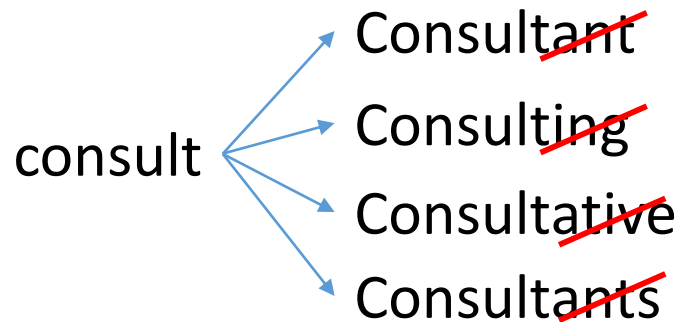
executed in 357ms, finished 01:20:21 2023-03-06

326

**Additional** → we also can remove stop words from the premade list

# Stemming

- **Stemming:** Stemming is the process of reducing words to their root form/ base word.
- **Purpose:** to normalize text and make it easier to process.



**Note:** Stemming process is not required that the stem be a valid word or identical to its morphological root. The goal is to reduce related words to the same stem.



# Stemming

Type suffixes	Purpose	Examples
Inflectional	<ul style="list-style-type: none"><li>Form is varied to express some grammatical feature such as singular/plural or present/past/future tense.</li><li>Inflections don't change part of speech or meaning.</li></ul>	Big → Biggest, Bigger Boy → Boys
Derivational	<ul style="list-style-type: none"><li>New forms are created from words.</li><li>New ones have a different part of speech or meaning.</li></ul>	Create → Creation Rationalizations → 'al', 'iz', 'ation' (derivational). 's' (inflectional)

**Note:** Stemming most likely put more focus on **suffixes**.

# Stemming: the error

---

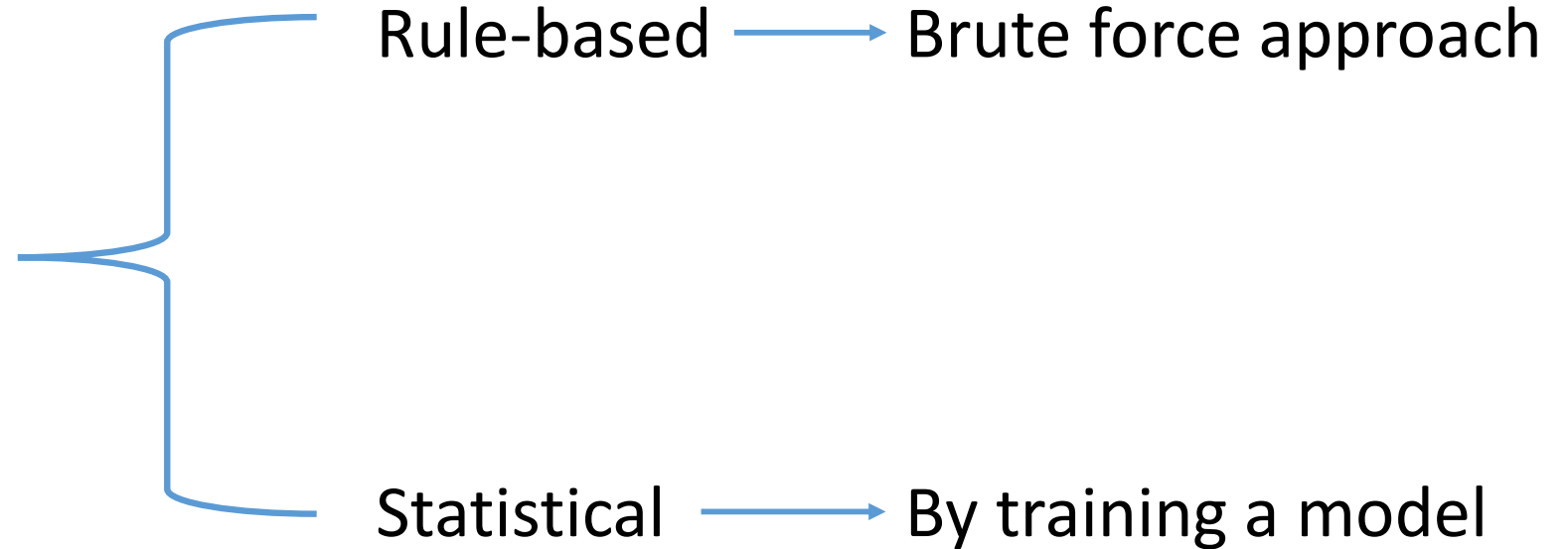
## Typical errors of stemming:

Over-stemming	Happens when too much is removed.	Wander → wand News → new Universal, universe, university, universities → univers
Under-stemming	Happens when words are from the same root but are not seen that way	Data → dat Datum → datu
Mis-stemming	Usually not a problem unless it leads for false conflation.	Relativity → relative

# Stemming: algorithm

---

- Porter's stemmer
- Lovins stemmer
- Dawson stemmer
- Krovetz stemmer
- Xerox stemmer
- N-gram stemmer
- Snowball stemmer
- Lancaster stemmer



# Stemming: code example

Code

```
from nltk.stem.porter import *
```

TextBlob

```
stemmer = PorterStemmer()
plurals = ['caresses', 'flies', 'dies', 'mules', 'denied',
           'died', 'agreed', 'owned', 'humbled', 'sized',
           'meeting', 'stating', 'siezing', 'itemization',
           'sensational', 'traditional', 'reference', 'colonizer',
           'plotted']

singles = [stemmer.stem(plural) for plural in plurals]

print(' '.join(singles))
```

Output

caress fli die mule deni die agre own humbl size meet state siez item sensat tradit refer colon plot

# Lemmatization

---

- **Lemmatization:** Lemmatization is a more complex approach to determining word stems, which addresses the potential problem on stemming process.
- Lemmatization involves word morphology, which is the study of word forms. Typically, we **identify the morphological tags of a word** before selecting the lemma, thus, this process often requires part of speech information.
- **Purpose:** By normalizing words to a common form, we get better results. Also, removing inflected wordforms can improve downstream NLP tasks .

**Note:** involve dictionary lookup or morphological analysis, and put more focus on the original word's meaning.

# Lemmatization: example

## Problems in stemming that lemmatization can solve:

- Two wordforms with different lemmas may stem to the same result.  
Example: 'universal' and 'university' result in same stem 'univers'.
- Two wordforms of same lemma may end as two different stems.  
Example: 'good' and 'better' have the same lemma 'good'.

### Lemmatization

Vs

### Stemming

was → (to) be

better → good

meeting → meeting

adjustable → adjust

formality → formaliti

formaliti → formal

airliner → airlin

### *Lemmatization example*

List of words: going, gone, went

Lemma: go

# Lemmatization: code example

Code

```
from nltk.stem import WordNetLemmatizer  
  
lemmatizer = WordNetLemmatizer()  
  
print("rocks :", lemmatizer.lemmatize("rocks"))  
print("corpora :", lemmatizer.lemmatize("corpora"))  
  
# a denotes adjective in "pos"  
print("better :", lemmatizer.lemmatize("better", pos = "a"))
```

spaCy, TextBlob,  
Pattern, Gensim

Output

```
rocks : rock  
corpora : corpus  
better : good
```

"n" for nouns  
"v" for verbs  
"r" for adverbs  
"s" for satellite adjectives

# Part-of-Speech (POS) Tagging

Lexical Term	Tag	Example
<i>Noun</i>	NN	England, Someone
<i>Verb, 3<sup>rd</sup> person singular</i>	VBZ	work, learn, run
<i>Determiner</i>	DT	the, a
Adjective	JJ	critical, calm
...	...	...

**Words** → This is a simple sentence.  
→ **Part-of-speech** → DT VBZ DT JJ NN



# Additional Linguistic Knowledge

## Syntax

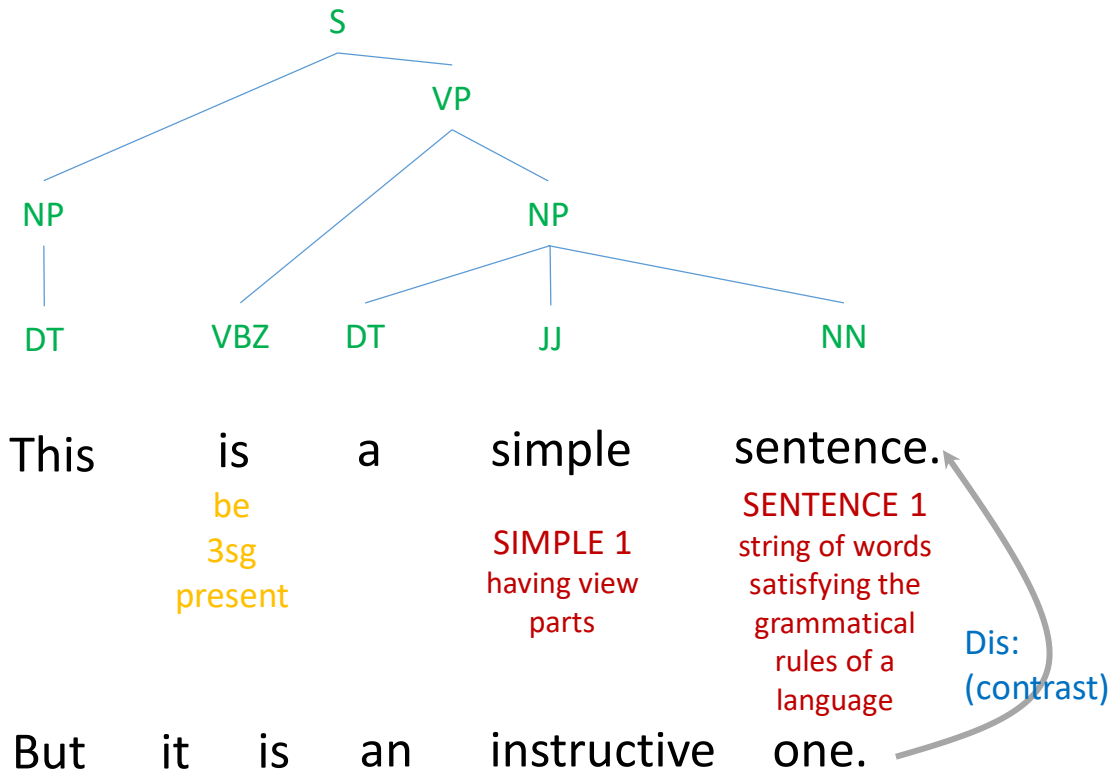
## Part-of-speech

## Words

## Morphology

## Semantics

## Discourse



# Additional steps of pre-processing

- Lower case
- Punctuation and its repetition removal
- Sentence segmentation: Chinese characters
- Part-of-speech (POS) tagging
- Spelling correction
- Numbers removal
- Emojis and emoticons removal
- Spaces and extra space removal

**Sentence:** The dog killed the bat.

**Parts of speech:** Definite article, noun, verb, definite article, noun.

- Name entity recognition

**Text:** Google CEO Sundar Pichai resides in New York.

**Named entity recognition:**

Google — Organization

Sundar Pichai — Person

New York — Location

**Thank you**  
**Q & A**