Summer Course: NLP Empowerment:

Delving into Prompt Engineering and LLMs for enhanced capabilities-Day 2

Prof. Ananthanagu
ananthanagu@pes.edu

NATURAL LANGUAGE PROCESSING

Some Important Terminologies

- **Language** is a method of communication with the help of which we can speak, read and write.
- **Natural Language Processing (NLP)** is the sub-field of Computer Science especially Artificial Intelligence (AI) that is concerned about enabling computers to understand and process human language.
- **Linguists**
  - How phrases and sentences can be formed with words?
- **corpus**
  - A corpus is a **collection of authentic text or audio organized into datasets.**
  - '**Authentic**' in this case means **text written** or **audio spoken** by a **native** of the **language** or dialect.
  - A corpus can be made up of everything from **newspapers, novels, recipes and radio broadcasts to television shows, movies and tweets.**

NATURAL LANGUAGE PROCESSING
Some Important Terminologies

Year	Name of the Corpus	Size (in words)
1960s - 70s	Brown and LOB	1 Million words
1980s	The Birmingham corpora	20 Million words
1990s	The British National corpus	100 Million words
Early 21st century	The Bank of English corpus	650 Million words

TreeBank Corpus- annotates syntactic or semantic sentence structure

PropBank more specifically called “**Proposition Bank**” is a corpus, which is annotated with verbal propositions and their arguments

VerbNet(VN) is the hierarchical domain-independent and largest lexical resource present in English that incorporates both semantic as well as syntactic information about its contents

WordNet, created by Princeton is a lexical database for English language. It is the part of the NLTK corpus. In WordNet, nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms called **Synsets**.

Some Important Terminologies

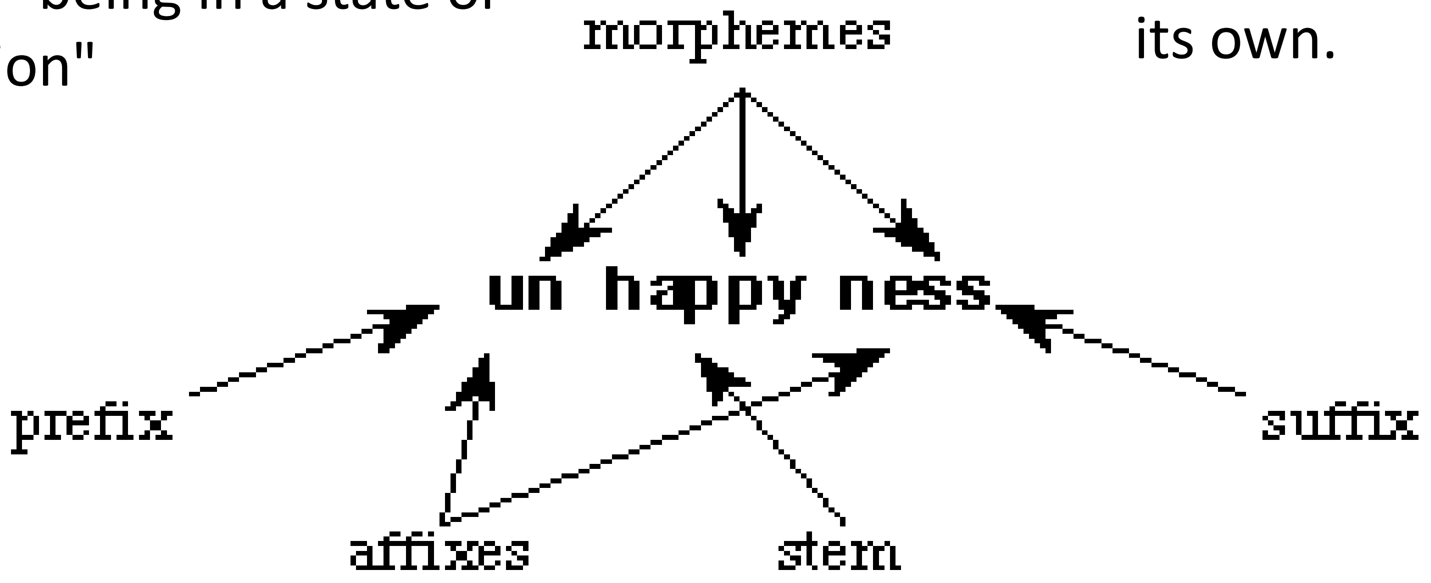
- **Morphology** is a field of **linguistics** that **studies the structure of words.**
- It identifies **how a word** is produced **through the use of morphemes**.
- A **morpheme** is a basic unit of the English language.
- The morpheme is the **smallest element of a word** that has grammatical function and meaning.

- **Lexicon**
  - It refers to the **dictionary of words** (**stem/root** word), their **categories** (**noun**, **verb**, **adjective**, etc.), **their sub-categories** (**singular noun, plural noun,** etc.) and the **affixes (prefix, suffix, infix)** that can be attached to these **stems**.
- **Orthographic rules**
  - It refers to the **spelling rules** used in a particular language to model the **spelling changes** that **occur in a word**.
  - For example, when a stem '***cook***' is attached with the morpheme '***ing***', it becomes a new valid word with the spelling '***cooking***'. What happened here was the **concatenation** of two strings.

NATURAL LANGUAGE PROCESSING
Morphological Analysis

un means "not", while ness means "being in a state or condition"

Happy is a free morpheme because it can appear on its own.

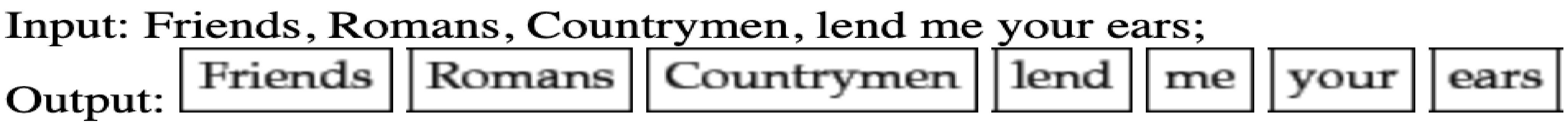


Some Important Terminologies

- **Finite State Transducer(FST):**
  - key algorithm for morphological parsing.
- **Stemming:**
  - In information retrieval and web search (IR), we may map from *foxes* to *fox*; (but might not need to also know that *foxes* is plural).
  - Just stripping off such word endings is called **stemming** in IR.
  - A simple stemming algorithm called the **Porter stemmer**.
- **Lemmatization:**
  - The words *sang*, *sung*, and *sings* are all forms of the verb *sing*.
  - The word *sing* is sometimes called the common ***lemma*** of these words, and mapping from all of these to *sing* is called **lemmatization**.
- The goal of **stemming and Lemmatization** is to **reduce** a word to a common base form

NATURAL LANGUAGE PROCESSING
Some Important Terminologies

- **Tokenization:**
  - **Tokenization** or **word segmentation** is the task of separating out (tokenizing) words from running text.
  - Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called ***tokens*** , perhaps at the same time throwing away certain characters, such as punctuation.
  - Here is an example of tokenization:



Input: Friends, Romans, Countrymen, lend me your ears;

Output: $\boxed{\text{Friends}}$ $\boxed{\text{Romans}}$ $\boxed{\text{Countrymen}}$ $\boxed{\text{lend}}$ $\boxed{\text{me}}$ $\boxed{\text{your}}$ $\boxed{\text{ears}}$

Tokenization

- **Tokenization in NLP: Types, Challenges, Examples, Tools**
- The first thing you need to do in any NLP project is text preprocessing. Preprocessing input text simply means putting the **data into a predictable and analyzable form**.
- It's a crucial step for building an amazing NLP application.
- There are different ways to preprocess text:
  - **stop word removal,**
  - **tokenization,**
  - **stemming.**
- Among these, the most important step is **tokenization**.
- It's the process of **breaking** a **stream of textual data into words, terms, sentences, symbols, or some other meaningful elements** called **tokens**.
- A lot of **open-source tools** are available to perform the tokenization process.

NATURAL LANGUAGE PROCESSING

Tokenization

- **Why do we need tokenization?**
- Tokenization is the first step in any **NLP pipeline**.
- It has an important effect on the rest of your pipeline.
- A **tokenizer** breaks **unstructured data** and **natural language text** into **chunks** of information that can be considered as discrete elements.
- The token occurrences in a document can be used **directly** as a **vector** representing that document.
- This immediately turns an **unstructured string (text document)** into a **numerical data structure** suitable for **machine learning**.
- They can also be used directly by a **computer** to **trigger** useful **actions** and **responses**. Or they might be used in a **machine learning pipeline** as features that **trigger** more **complex decisions or behavior**.
- Tokenization can separate **sentences**, **words**, **characters**, or **subwords**.
- When we split the **text** into **sentences**, we call it **sentence tokenization**. For **words**, we call it **word tokenization.**

Different tokenization techniques:

1. **Word Tokenization**:

- This technique breaks a **text** into **individual words** based on **spaces** or **punctuation marks**.
- For example, the sentence **"I love tokenization!"** would be tokenized into **['I', 'love', 'tokenization', '!'].**

2. **Sentence Tokenization**:

- Instead of tokenizing at the **word level**, this technique splits the **text into sentences**.
- The input text is divided into **individual sentences** based on ***punctuation marks like periods, question marks, or exclamation marks.***

3. **Subword Tokenization**:

- Subword tokenization splits words into **smaller meaningful units**, called s**ubwords or subword units**.
- This technique is commonly used in languages with complex morphology or for handling **out-of-vocabulary (OOV) words**.

Tokenization

4. Character Tokenization:

- Character-level tokenization breaks down a **text** into individual **characters**.
- Each **character becomes a separate token**.
- This technique is useful when working with languages that don't have clear word boundaries or for tasks such as sentiment analysis or text generation.

5. Byte-level Tokenization:

- In byte-level tokenization, **each byte in the input text** is treated as a **separate token**.
- This technique is often used in **machine translation** or **text generation tasks** where preserving the **exact byte sequence is important**.

6. Syntactic Tokenization:

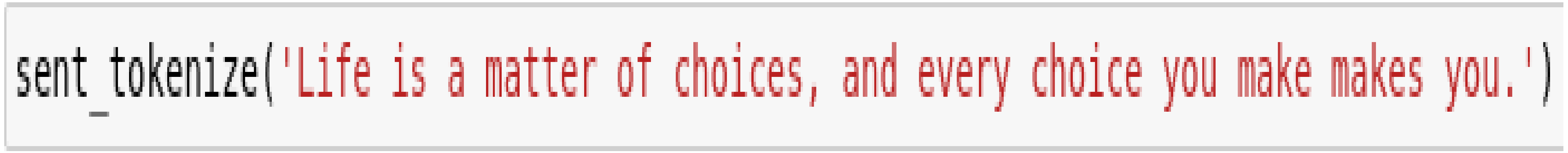
- Syntactic tokenization involves **splitting text** into **tokens** based on **syntactic structures**, such as **part-of-speech tags or named entities**.
- This technique is useful for tasks like **named entity recognition, part-of-speech tagging, or syntactic parsing.**

7. Domain-Specific Tokenization:

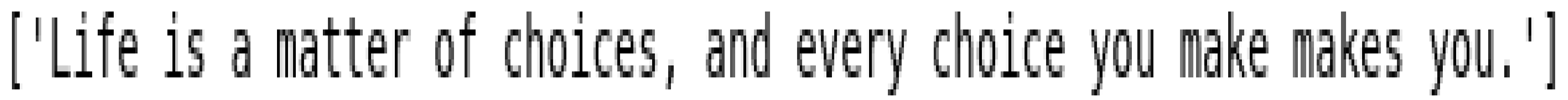
- In some cases, specialized tokenization techniques are used for specific **domains or tasks.**
- For example, in **bioinformatics, gene and protein sequences** may be tokenized into individual **nucleotides or amino acids**.

Tokenization

Example of sentence tokenization

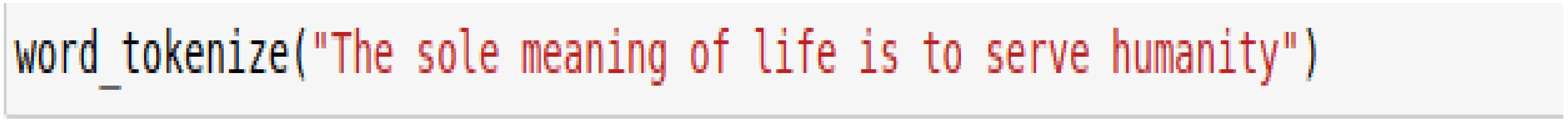


```
sent_tokenize('Life is a matter of choices, and every choice you make makes you.')
```

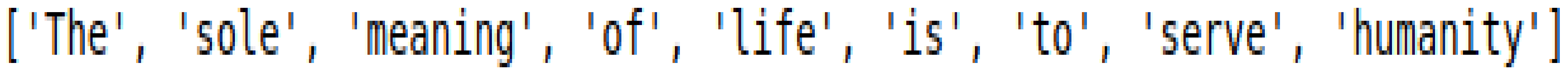


```
['Life is a matter of choices, and every choice you make makes you.']
```

Example of word tokenization



```
word_tokenize("The sole meaning of life is to serve humanity")
```



```
['The', 'sole', 'meaning', 'of', 'life', 'is', 'to', 'serve', 'humanity']
```

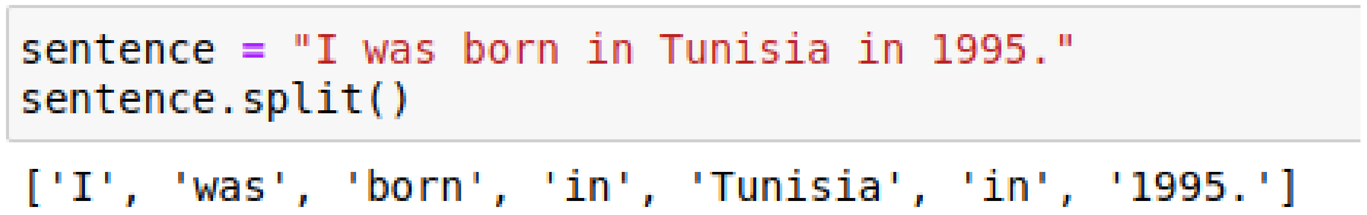
Tokenization

Different tools for tokenization

- White Space Tokenization
- NLTK Word Tokenize
  - Word and Sentence tokenizer
  - Punctuation-based tokenizer
  - Treebank Word tokenizer
  - Tweet tokenizer
  - MWET tokenizer
- TextBlob Word Tokenize
- spaCy Tokenizer
- Gensim word tokenizer
- Tokenization with Keras

Tokenization

- **White Space Tokenization**
  - The simplest way to tokenize text is to use whitespace within a string as the "**delimiter**" of words.
  - This can be accomplished with **Python's split function**, which is available on all string object instances as well as on the string built-in class itself.
  - You can change the **separator** any way you need.

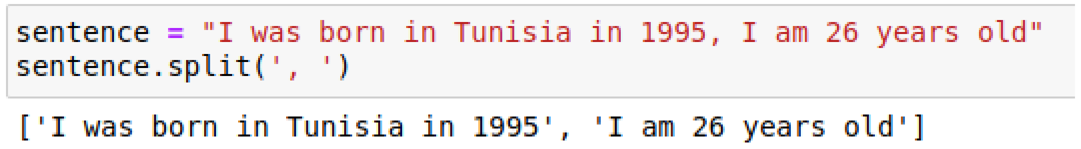


```
sentence = "I was born in Tunisia in 1995."
sentence.split()
```

```
['I', 'was', 'born', 'in', 'Tunisia', 'in', '1995.']
```

As you can notice, this **built-in Python** method already does a **good job tokenizing** a simple sentence.
It's "mistake" was on the **last word**, where it included the **sentence-ending punctuation** with the token "**1995.**".
We need the **tokens** to be **separated** from **neighboring punctuation and other significant tokens in a sentence.**

In the example below, we'll perform sentence tokenization using the **comma as a separator.**

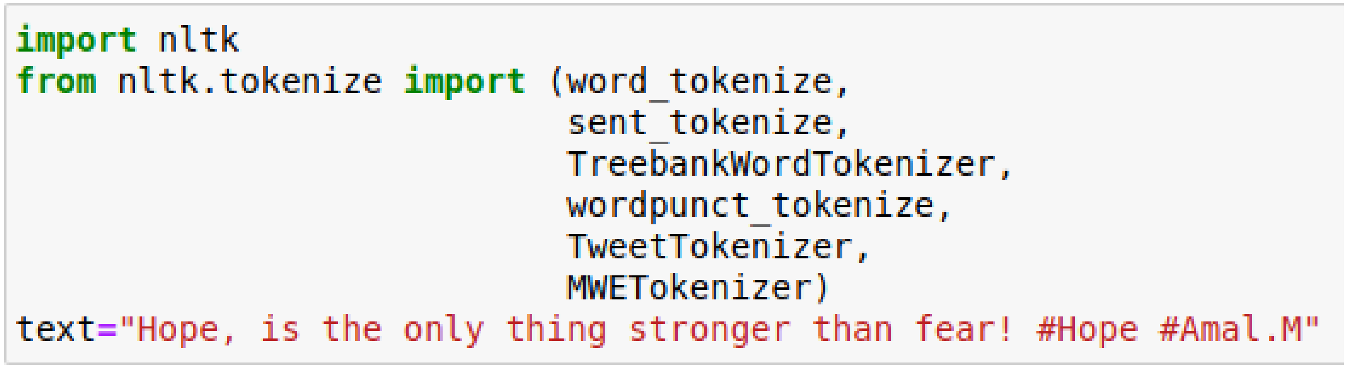


```
sentence = "I was born in Tunisia in 1995, I am 26 years old"
sentence.split(', ')
```

```
['I was born in Tunisia in 1995', 'I am 26 years old']
```

Tokenization

- **NLTK Word Tokenize**
  - **NLTK (Natural Language Toolkit)** is an **open-source Python library** for Natural Language Processing.
  - It has **easy-to-use interfaces** for over **50 corpora** and lexical resources such as **WordNet**, along with a set of text processing libraries for **classification**, **tokenization**, **stemming**, and **tagging**.
  - You can easily tokenize the sentences and words of the text with the tokenize module of NLTK.
  - First, we're going to import the relevant functions from the NLTK library:



```
import nltk
from nltk.tokenize import (word_tokenize,
                           sent_tokenize,
                           TreebankWordTokenizer,
                           wordpunct_tokenize,
                           TweetTokenizer,
                           MWETokenizer)
text="Hope, is the only thing stronger than fear! #Hope #Amal.M"
```

Tokenization

- **NLTK Word Tokenize**
  - **Word and Sentence tokenizer**

```
print(word_tokenize(text))
```

```
['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear', '!', '#', 'Hope', '#', 'Amal.M']
```

```
print(sent_tokenize(text))
```

```
['Hope, is the only thing stronger than fear!', '#Hope #Amal.M']
```

- **Punctuation-based tokenizer**

  This tokenizer splits the sentences into words based on whitespaces and punctuations.

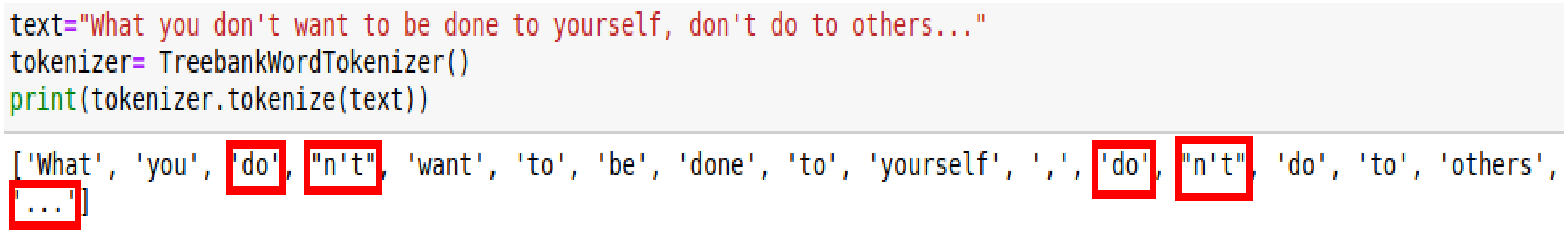
```
print(wordpunct_tokenize(text))
```

```
['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear', '!', '#', 'Hope', '#', 'Amal', '.', 'M']
```

We could notice the difference between considering “Amal.M” a word in word_tokenize and split it in the wordpunct_tokenize.

NATURAL LANGUAGE PROCESSING
Tokenization

- **NLTK Word Tokenize**
  - **Treebank Word tokenizer**
    - This tokenizer incorporates a variety of **common rules** for english word tokenization.
    - It separates **phrase-terminating punctuation** like (?!.;,) from **adjacent tokens** and retains **decimal numbers** as a **single token**. Besides, it contains rules for English contractions.
    - For example "**don't**" is tokenized as **["do", "n't"].**

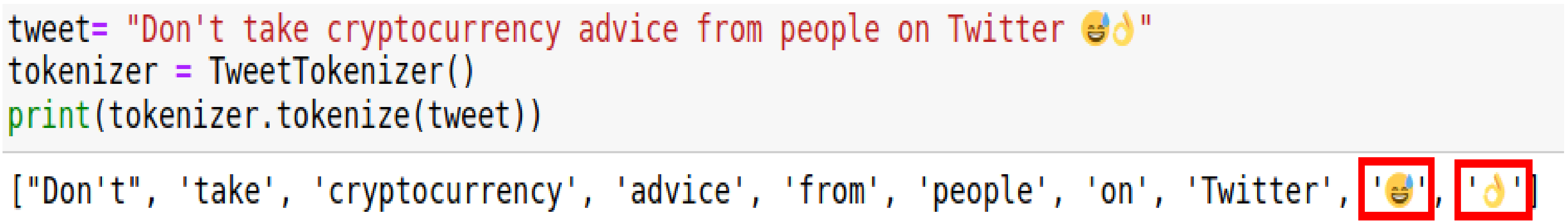


```
text="What you don't want to be done to yourself, don't do to others..."
tokenizer= TreebankWordTokenizer()
print(tokenizer.tokenize(text))
```

```
['What', 'you', 'do', "n't", 'want', 'to', 'be', 'done', 'to', 'yourself', ',', 'do', "n't", 'do', 'to', 'others',
'...']
```

Tokenization

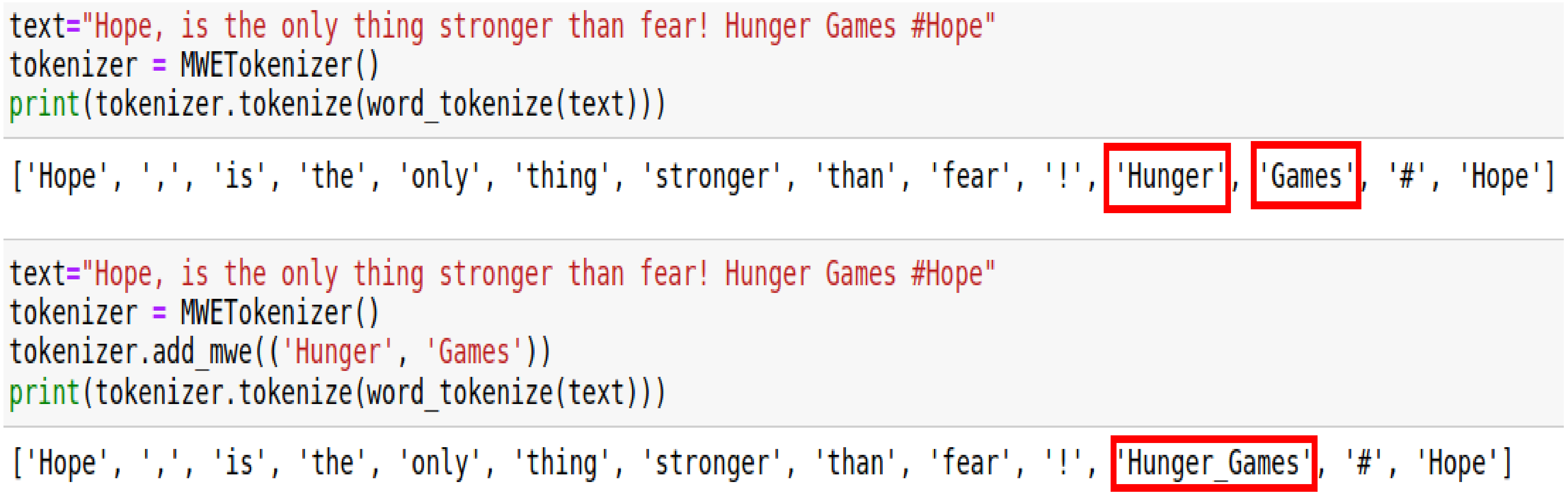
- **NLTK Word Tokenize**
  - **Tweet tokenizer**
    - When we want to apply tokenization in **text data like tweets**, the **tokenizers** mentioned above **can't produce practical tokens**.
    - Through this issue, NLTK has a **rule based tokenizer** special for **tweets**.
    - We can split **emojis** into **different words** if we need them for tasks like **sentiment analysis**.



```
tweet= "Don't take cryptocurrency advice from people on Twitter 😅👌"
tokenizer = TweetTokenizer()
print(tokenizer.tokenize(tweet))
```

```
["Don't", 'take', 'cryptocurrency', 'advice', 'from', 'people', 'on', 'Twitter', '😅', '👌']
```

- **MWET tokenizer**
  - NLTK's **multi-word expression tokenizer** (**MWETokenizer**) provides a function **add_mwe()** that allows the user to enter multiple word expressions before using the tokenizer on the text.
  - More simply, it can **merge multi-word expressions into single tokens.**



```
text="Hope, is the only thing stronger than fear! Hunger Games #Hope"
tokenizer = MWETokenizer()
print(tokenizer.tokenize(word_tokenize(text)))
```

```
['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear', '!', 'Hunger', 'Games', '#', 'Hope']
```

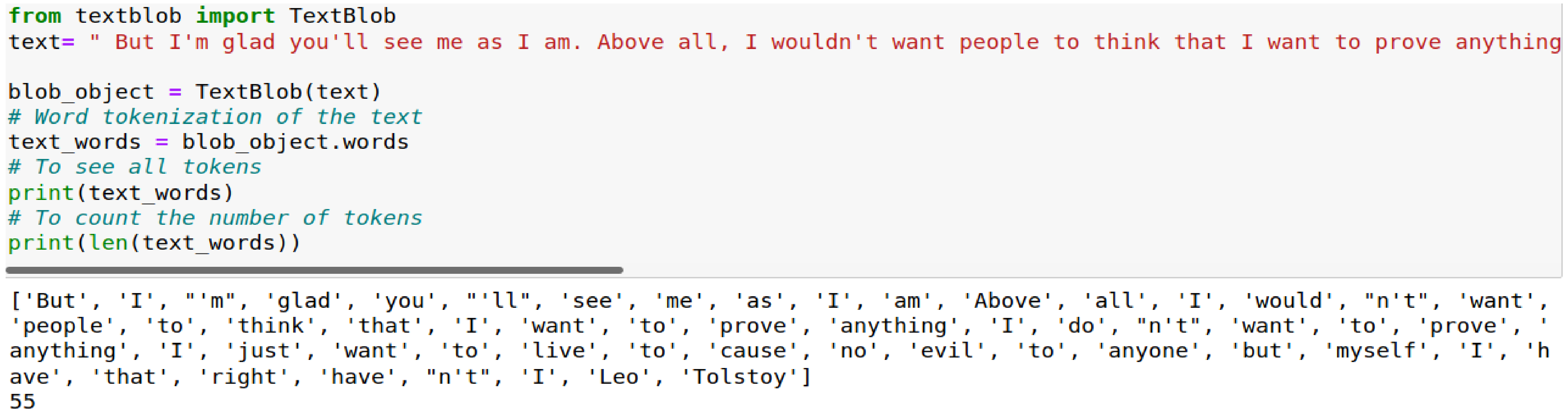
```
text="Hope, is the only thing stronger than fear! Hunger Games #Hope"
tokenizer = MWETokenizer()
tokenizer.add_mwe(('Hunger', 'Games'))
print(tokenizer.tokenize(word_tokenize(text)))
```

```
['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear', '!', 'Hunger_Games', '#', 'Hope']
```

NATURAL LANGUAGE PROCESSING

Tokenization

- **TextBlob Word Tokenize**
  - **TextBlob** is a **Python library** for processing **textual data**.
  - It provides a **consistent API** for diving into common natural language processing (NLP) tasks such as **part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.**
  - we perform **word tokenization** using TextBlob library:



```
from textblob import TextBlob
text= " But I'm glad you'll see me as I am. Above all, I wouldn't want people to think that I want to prove anything

blob_object = TextBlob(text)
# Word tokenization of the text
text_words = blob_object.words
# To see all tokens
print(text_words)
# To count the number of tokens
print(len(text_words))
```

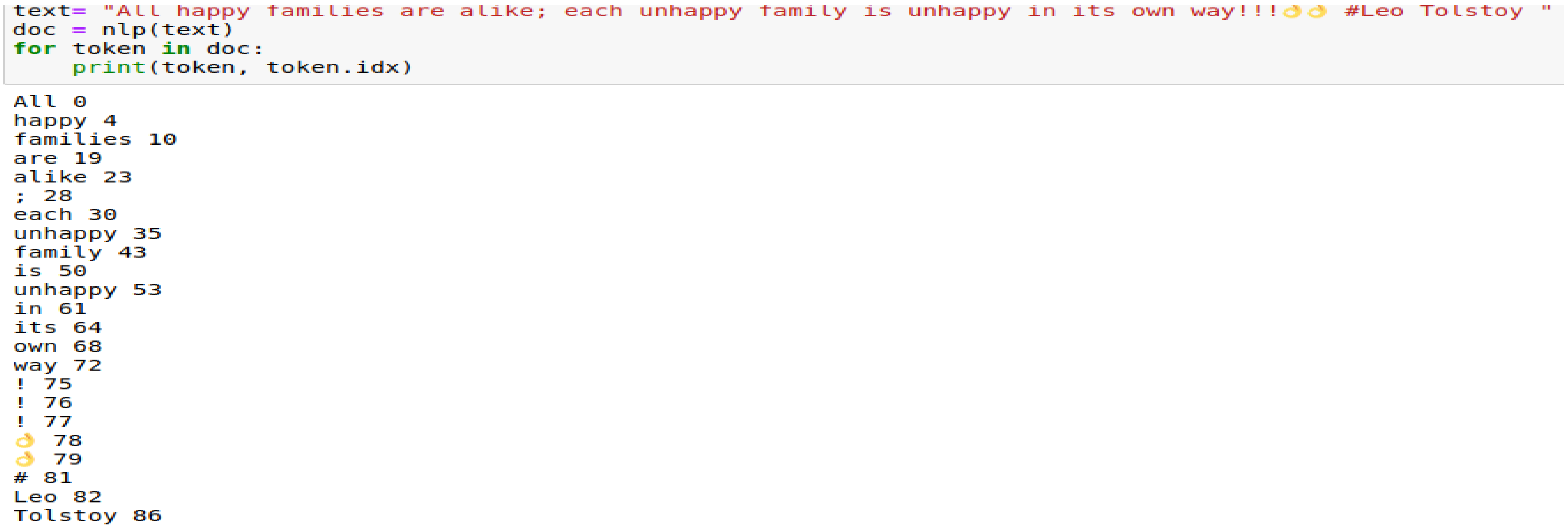
```
['But', 'I', "'m", 'glad', 'you', "'ll", 'see', 'me', 'as', 'I', 'am', 'Above', 'all', 'I', 'would', "n't", 'want',
'people', 'to', 'think', 'that', 'I', 'want', 'to', 'prove', 'anything', 'I', 'do', "n't", 'want', 'to', 'prove', '
anything', 'I', 'just', 'want', 'to', 'live', 'to', 'cause', 'no', 'evil', 'to', 'anyone', 'but', 'myself', 'I', 'h
ave', 'that', 'right', 'have', "n't", 'I', 'Leo', 'Tolstoy']
55
```

We could notice that the TextBlob tokenizer removes the punctuations. In addition, it has rules for English contractions.

NATURAL LANGUAGE PROCESSING

Tokenization

- **spaCy Tokenizer**
  - **SpaCy** is an **open-source Python** library that **parses** and **understands large volumes of text.**
  - **spaCy tokenizer** provides the flexibility to specify special tokens that don't need to be segmented, or need to be segmented using special rules for each language, for example punctuation at the end of a sentence should be split off – whereas **"U.K."** should remain **one token**.



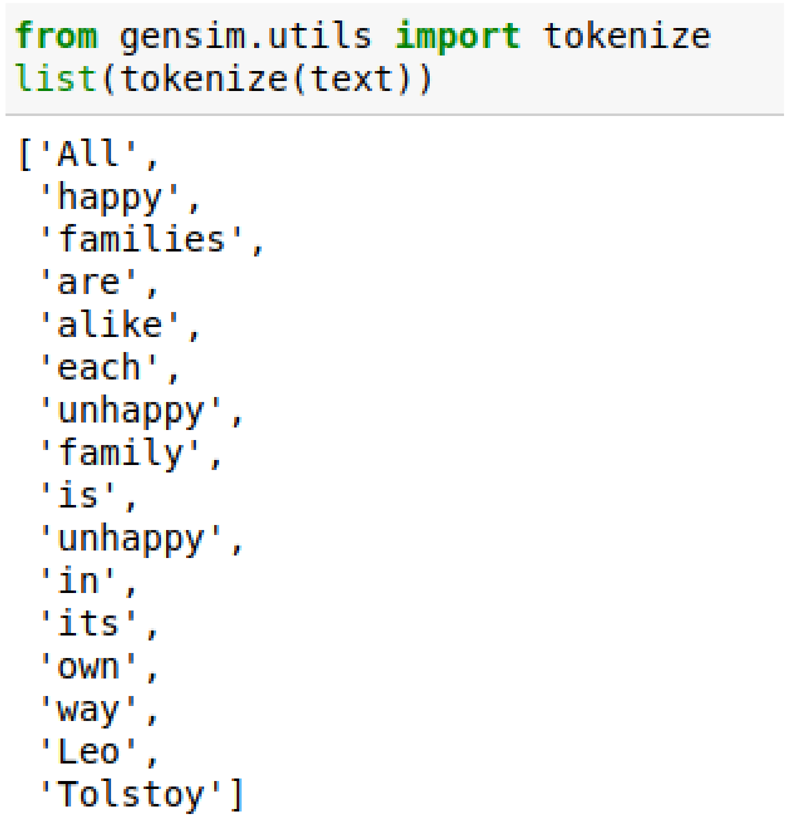
```
text= "All happy families are alike; each unhappy family is unhappy in its own way!!!👌👌 #Leo Tolstoy "
doc = nlp(text)
for token in doc:
    print(token, token.idx)
```

```
All 0
happy 4
families 10
are 19
alike 23
; 28
each 30
unhappy 35
family 43
is 50
unhappy 53
in 61
its 64
own 68
way 72
! 75
! 76
! 77
👌 78
👌 79
# 81
Leo 82
Tolstoy 86
```

NATURAL LANGUAGE PROCESSING

Tokenization

- **Gensim word tokenizer**
  - **Gensim** is a **Python library** for **topic modeling**, **document indexing**, and **similarity retrieval with large corpora**.



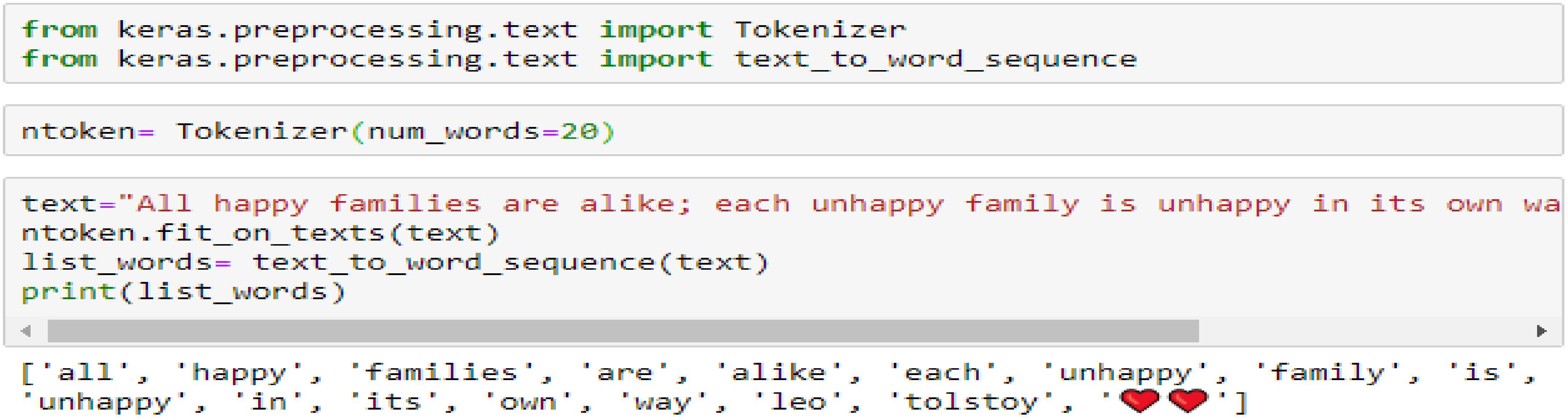
```python
from gensim.utils import tokenize
list(tokenize(text))
```

```
['All',
 'happy',
 'families',
 'are',
 'alike',
 'each',
 'unhappy',
 'family',
 'is',
 'unhappy',
 'in',
 'its',
 'own',
 'way',
 'Leo',
 'Tolstoy']
```

NATURAL LANGUAGE PROCESSING

Tokenization

- **Tokenization with Keras**
  - **Keras open-source library** is one of the most **reliable deep learning frameworks**.
  - To perform **tokenization** we use: **text_to_word_sequence** method from the Class **Keras.preprocessing.text class**.
  - The great thing about **Keras** is converting the **alphabet in a lower case before tokenizing it, which can be quite a time-saver.**



```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.text import text_to_word_sequence
```

```
ntoken= Tokenizer(num_words=20)
```

```
text="All happy families are alike; each unhappy family is unhappy in its own wa
ntoken.fit_on_texts(text)
list_words= text_to_word_sequence(text)
print(list_words)
```

```
['all', 'happy', 'families', 'are', 'alike', 'each', 'unhappy', 'family', 'is',
'unhappy', 'in', 'its', 'own', 'way', 'leo', 'tolstoy', '❤❤']
```

Problems of Tokenization

- Definition of Token
  - United States, AT&T, 3-year-old
- Ambiguity of punctuation as sentence boundary
  - Prof. Dr. J.M.
- Ambiguity in numbers
  - 123,456.78

Change Case

- Changing the **case** involves converting all text to **lowercase or uppercase** so that all word strings follow a consistent format.
- **Lowercasing** is the more **frequent choice in NLP software**.

```
# Lowercasing a sentence:
text = "This is an EXAMPLE sentence."
lowercased_text = text.lower()
print(lowercased_text)
```

```
this is an example sentence.
```

What are N-grams?

- **What are N-grams (unigram, bigram, trigrams)?**

  An N-gram is a sequence of N tokens (or words).

N-grams

- In the fields of Computational Linguistics an ***n*-gram** is a **contiguous sequence of *n* items** from a given sample of text or speech.
- The items can be letters, words or base pairs, phonemes or syllables according to the application.
- Eg Consider a sentence:
  - She was laughing at him.
  - “she was” or “was laughing” etc are **bigram(2-gram)**
  - “She was laughing” or “ laughing at him” etc are **trigrams(3-grams)**
- The *n*-grams typically are collected from a text or speech corpus.
- An ***n*-gram model** is a type of probabilistic language model for predicting the next item in such a sequence.

What are N-grams?

- Example: Consider the following sentence:

  "I love working with students on different research ideas."

  - A 1-gram (or unigram) is a one-word sequence. For the above sentence, the unigrams would simply be:
    "I", "love", "working", "with", "students", "on", "different", " project", "ideas".

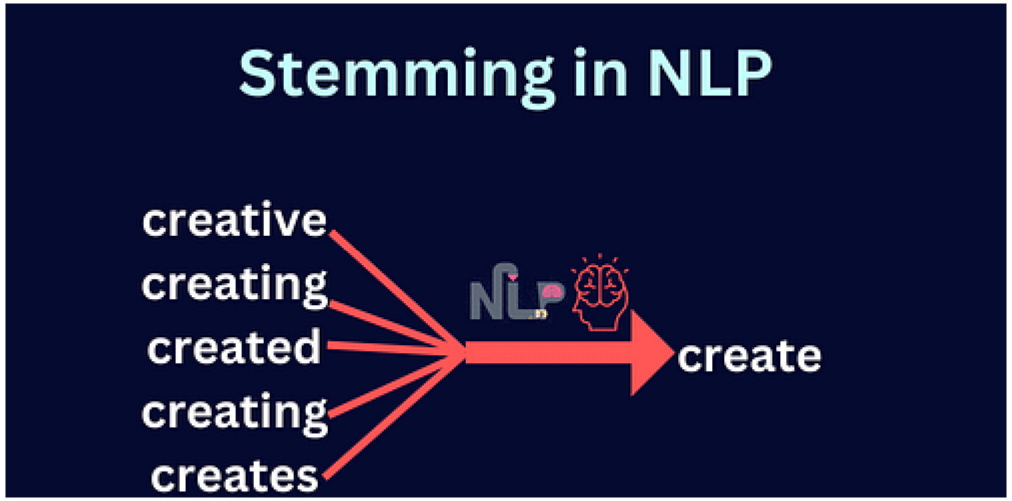
  - A 2-gram (or bigram) is a two-word sequence of words, like
    "I love", "love working ", "working with " ….

  - A 3-gram (or trigram) is a three-word sequence of words like
    "I love working ", "love working with ", ……..

Slide Courtesy :C O Prakash

Stemming

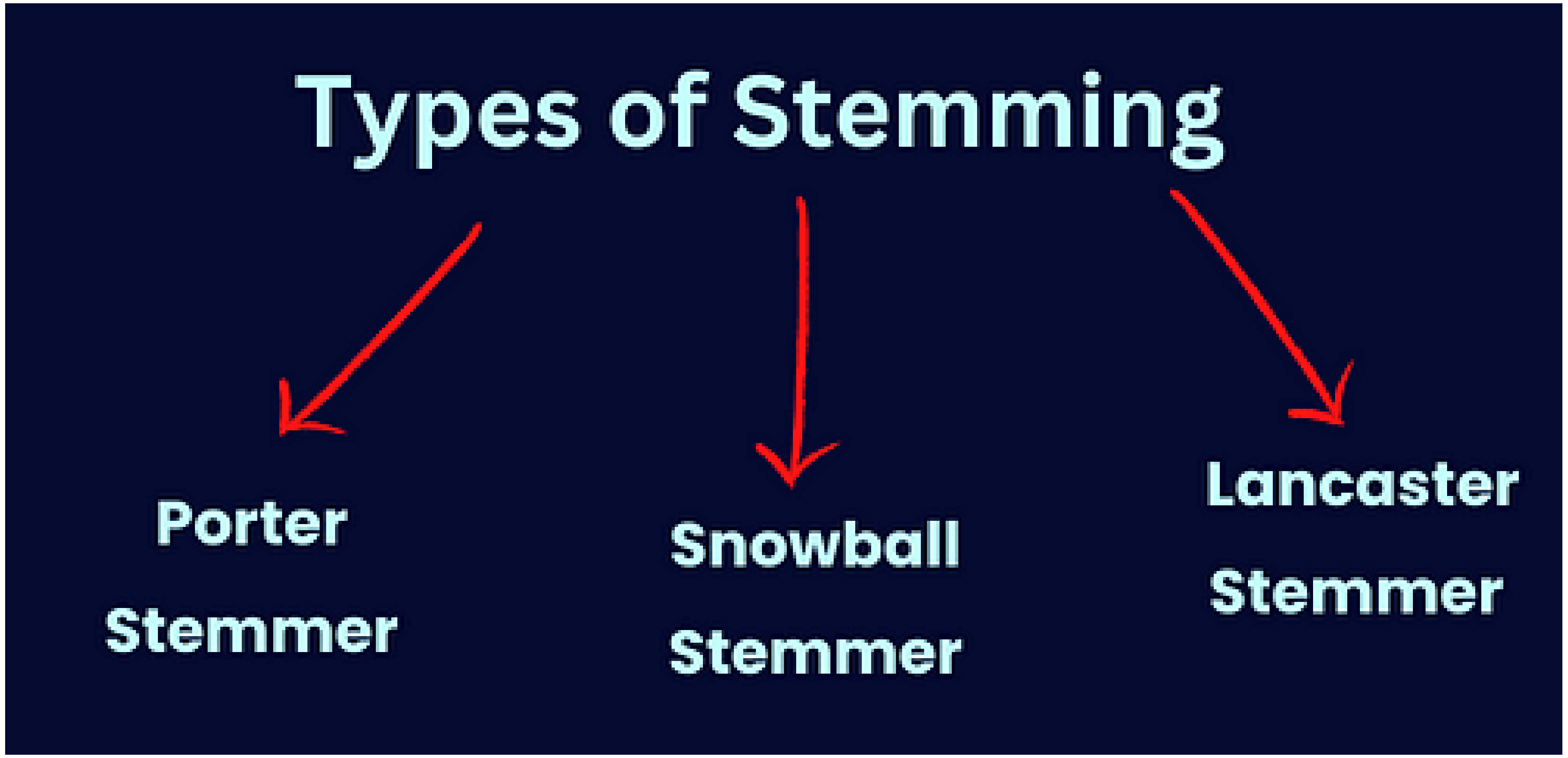
- The process of **removing affixes** from a **word** so that we are **left** with the **stem** of that word is called **stemming**.
- For example, consider the words ‘**run’, ‘running’, and ‘runs**’, all convert into the **root word** ‘**run**’ after stemming is implemented on them.
- One crucial point about stem words is that they need not be **meaningful**.
- For example, the word ‘**traditional**’ stem is ‘**tradi**’ and has **no meaning**.

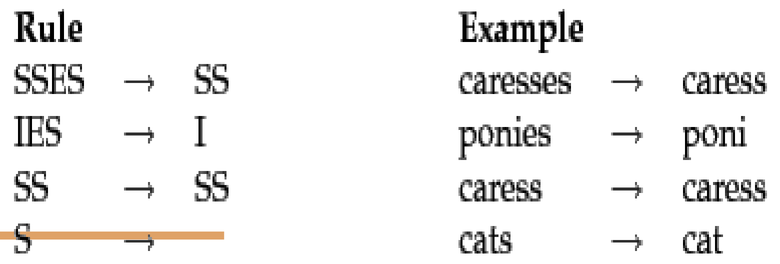


Stemming

- **Why use Stemming in NLP?**
  1. It **reduces the number of words** that serve as an **input** to the **Machine Learning/Deep Learning model.**
  2. It **minimizes** the **confusion around words** that have **similar meanings**.
  3. It **lowers** the **complexity** of the input space.
  4. When creating applications that search a **specific text** in a document, using **stemming** for **indexing** assists in retrieving relevant documents.
  5. It assists in eliminating the **out-of-vocabulary (OOV)** problem.
  6. For example, if the vocabulary **does not contain** the word '**oranges**', one can use the stem word '**orange**' as a **proxy**.
  7. It enhances the **accuracy** of the ML/DL model as the model does not have to deal with inflected word forms.

- **Types of Stemming in NLP**





Rule			Example		
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		cats	→	cat

Porter Stemming in NLP

- This stemming algorithm is named after the person who created it, **Martin Porter**.
- It is one of the **simplest** and **most commonly used** stemming algorithms.
- It is based on **simple rules** and **works only with** the **strings** data type.
- It only **supports the English language** and gives the best **output** as compared to other stemming algorithms, and it has **less error rate**.
- The first step in this algorithm comprises of either of the following:

1. **SSESS to SS**
   It suggests that if the word ends with the suffix '**sses**', the Porter algorithm will transform it into '**ss**' suffix. For example, **possess** will be transformed to **poss**.
2. **IES to I**
   It suggests that if the word ends with the suffix '**ies**', the Porter algorithm will transform it into **'i'** suffix. For example, **butterflies** will be transformed into '**butterfli**'.
3. **SS to SS**

- It suggests that if the word ends with the suffix '**ss**', the Porter algorithm will **not make any changes** to the word. For example, '**supress**' will remain as it is.

4. **S to _**

- It suggests that if the word ends with the suffix '**s**', the Porter algorithm will **not remove that suffix**. For example, '**chairs**' will be transformed to '**chair**'.

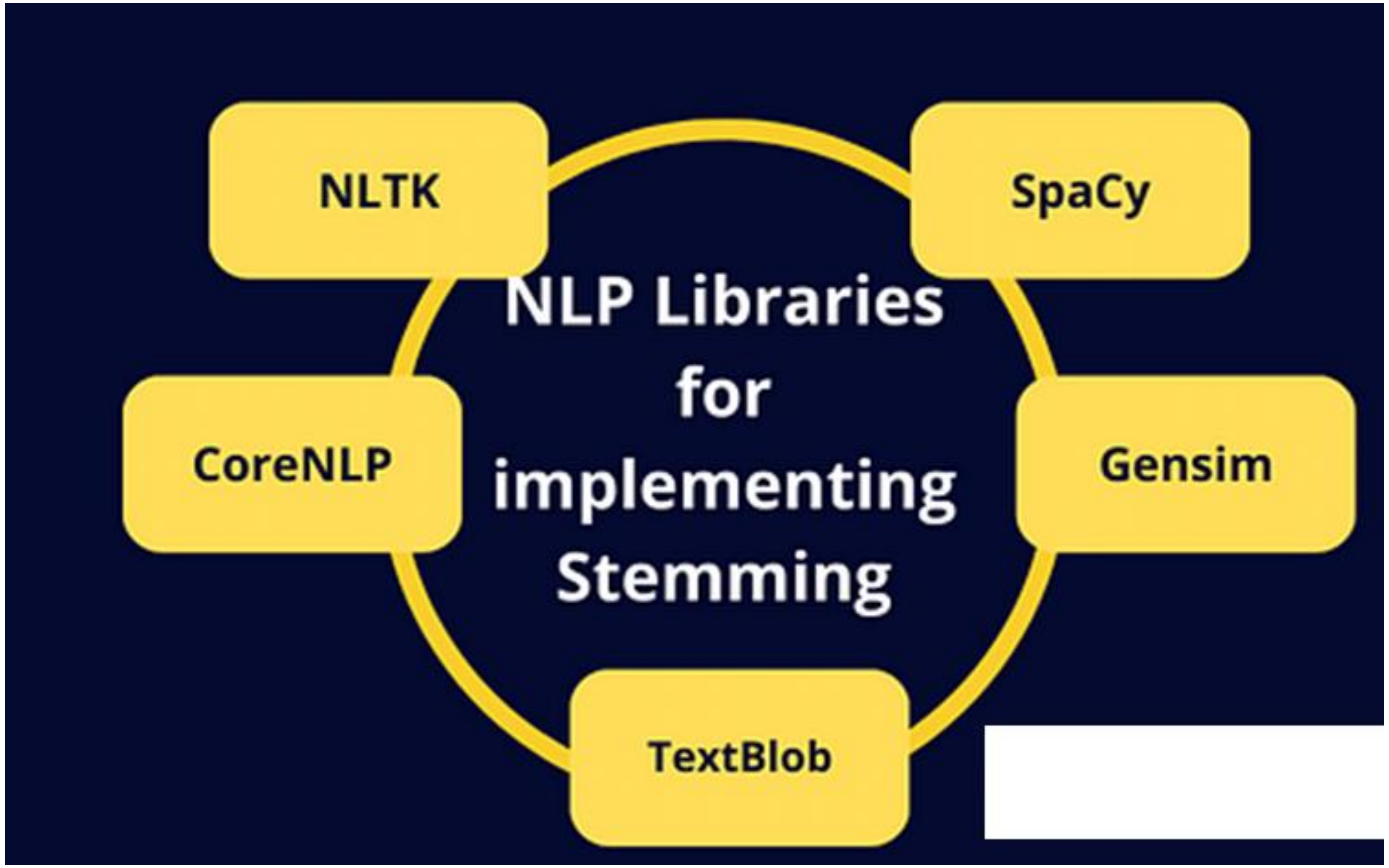
Snowball Stemming in NLP

- This stemming algorithm is also known as the **Porter2 algorithm** because it is an **improved version of the Porter algorithm** that **supports multiple languages**.
- It is more **accurate** than the Porter algorithm and works with **Unicode** and **string data.**
- In Snowball stemming algorithm, the **rule** is to **replace** the words with suffix- '**ied/ies**' by '**i**' if preceded by **more than one letter**, and by '**ie**' in other cases.
- It offers **higher computational speed than the Porter stemmer**.

Lancaster Stemming in NLP

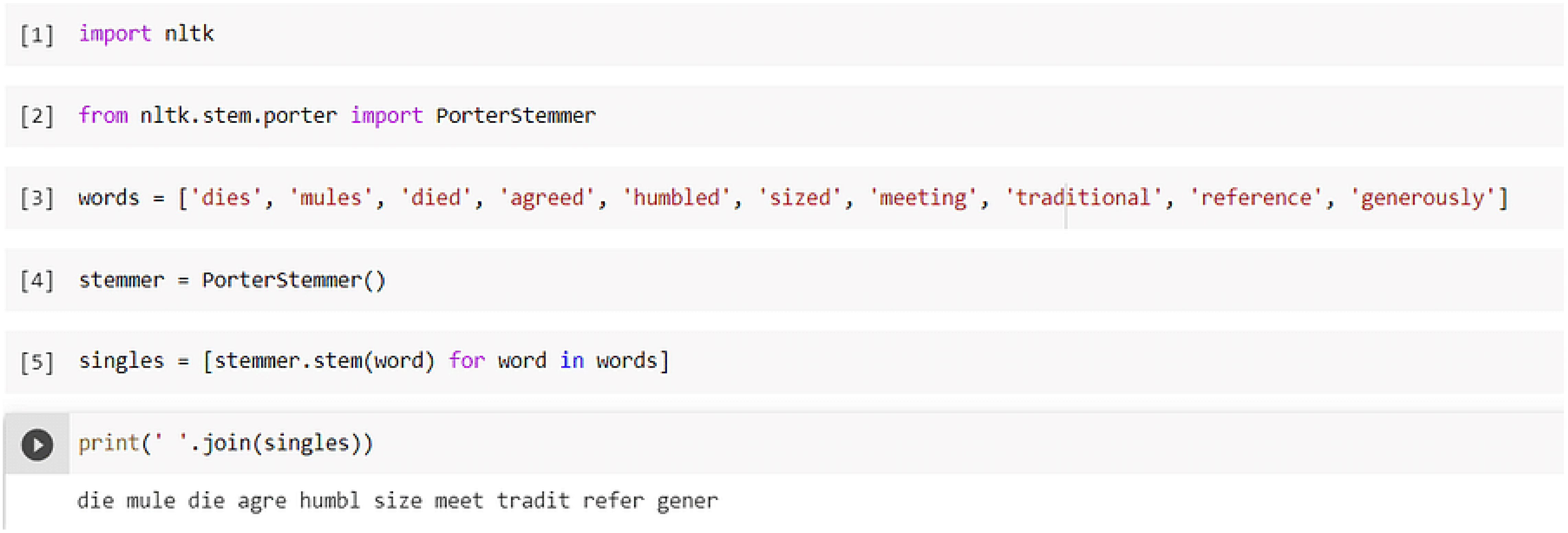
- This stemming algorithm is one of the **fastest algorithms** available out there.
- Unlike Snowball stemmer and Porter stemmer, the **stem words** in this algorithm are **not intuitive**.
- The Lancaster rule converts words with '**ies**' as a suffix into the **'y**' suffix.
- This stemming algorithm is not as efficient as the Snowball one.

Stemming Algorithms in NLP Libraries



Stemming

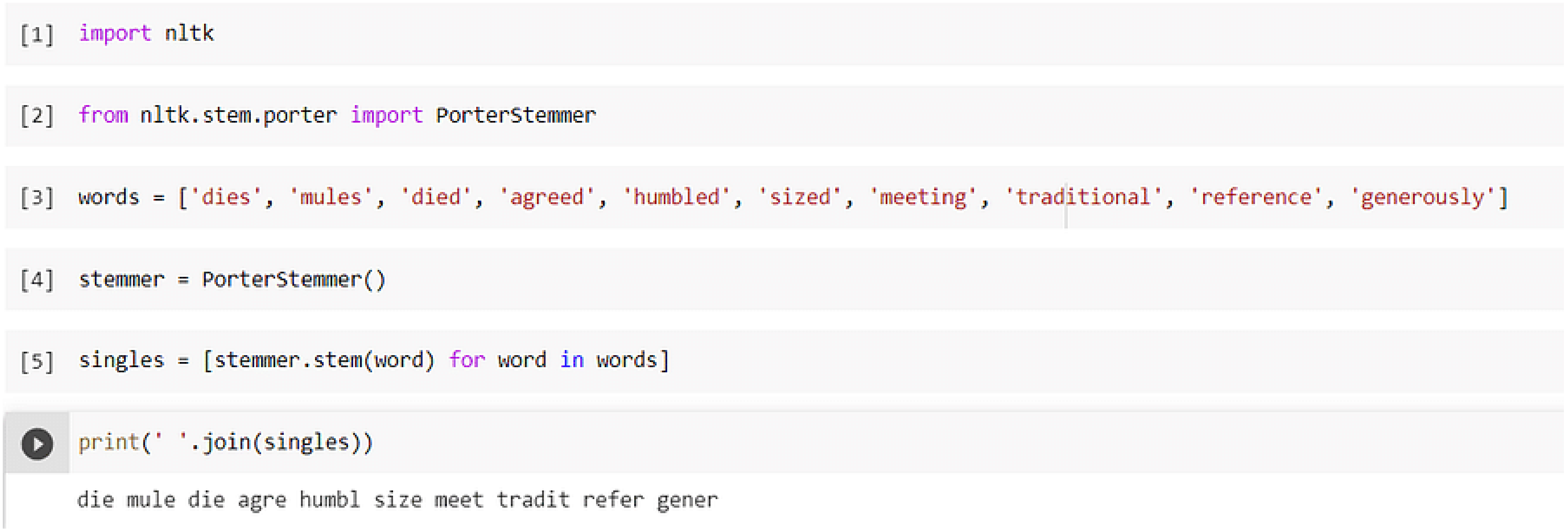
1.Porter Stemmer Algorithm with NLTK



```
[1]  import nltk

[2]  from nltk.stem.porter import PorterStemmer

[3]  words = ['dies', 'mules', 'died', 'agreed', 'humbled', 'sized', 'meeting', 'traditional', 'reference', 'generously']

[4]  stemmer = PorterStemmer()

[5]  singles = [stemmer.stem(word) for word in words]

     print(' '.join(singles))

     die mule die agre humbl size meet tradit refer gener
```

Notice how the words ‘dies’ and ‘died’ have been transformed to the same stem, ‘die’.

2.Snowball Stemmer Algorithm with NLTK



```
[1]  import nltk
```

```
[2]  from nltk.stem.porter import PorterStemmer
```

```
[3]  words = ['dies', 'mules', 'died', 'agreed', 'humbled', 'sized', 'meeting', 'traditional', 'reference', 'generously']
```

```
[4]  stemmer = PorterStemmer()
```

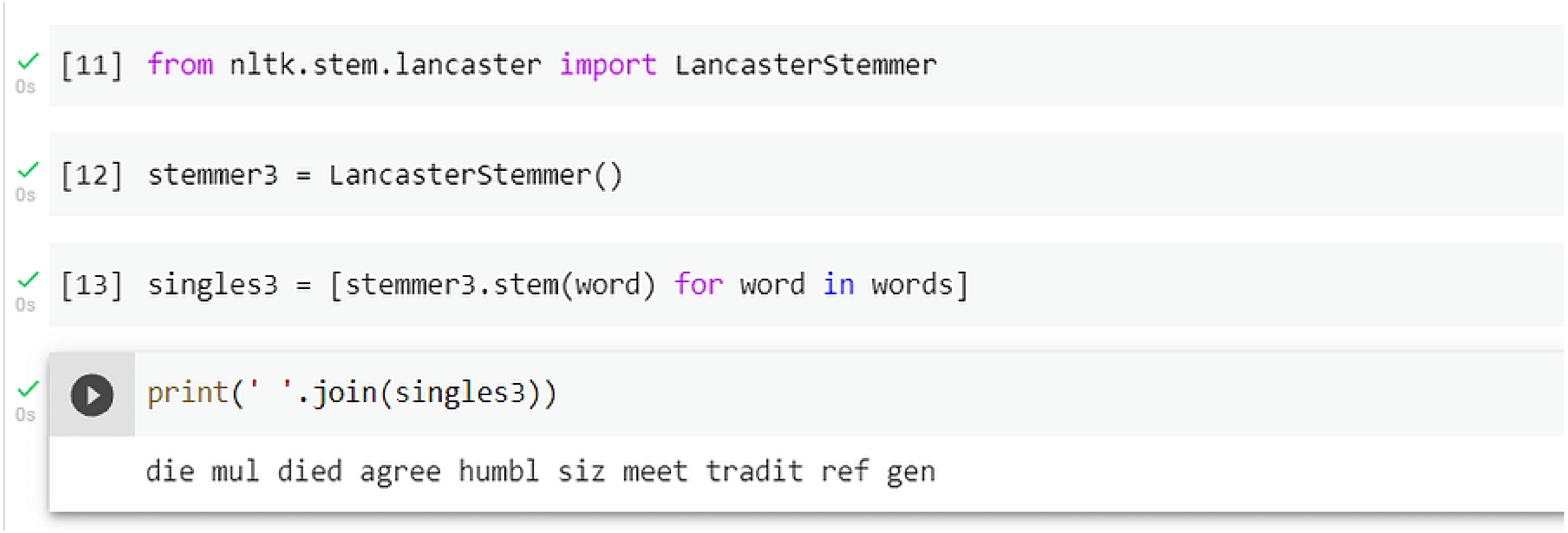
```
[5]  singles = [stemmer.stem(word) for word in words]
```

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

die mule die agre humbl size meet tradit refer gener
```

Notice how the **base form of most words** is the **same** as in the **case of Porter Stemmer** and **only the word** ‘**generously**’ has different **stems** for **Snowball and Porter Stemmer**.

Stemming

3.Lancaster Stemmer Algorithm with NLTK

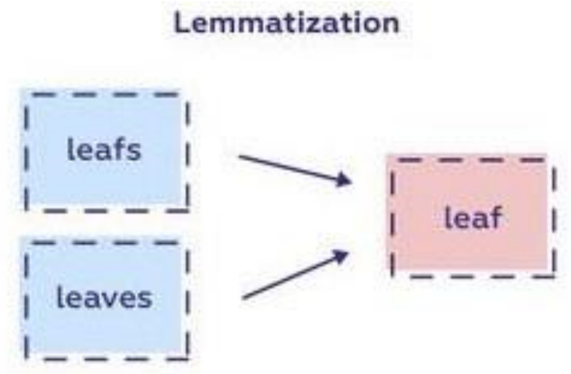


```
[11] from nltk.stem.lancaster import LancasterStemmer

[12] stemmer3 = LancasterStemmer()

[13] singles3 = [stemmer3.stem(word) for word in words]

print(' '.join(singles3))

die mul died agree humbl siz meet tradit ref gen
```

Notice how the root form for the word ‘**generous**’ is **differs** from the **Snowball and Porter stemmer**.

Lemmatization

- Lemmatization is a **text normalization technique** used in Natural Language Processing (NLP), that **switches** any **kind of a word** to its **base root mode**.
- Lemmatization is responsible for **grouping different inflected forms of words** into the **root form**, having the **same meaning**.
  Tagging systems, indexing, SEOs, information retrieval, and web search all use lemmatization to a vast extent.
- Lemmatization usually involves using a **vocabulary** and **morphological analysis of words**, **removing inflectional endings**, and **returning the dictionary form of a word** (the **lemma**).
- lemmatization is a **linguistic** term refers to the **act of grouping together words** that have the **same root or lemma** but **have different inflections** or **derivatives** of **meaning** so they can be analyzed as **one item**



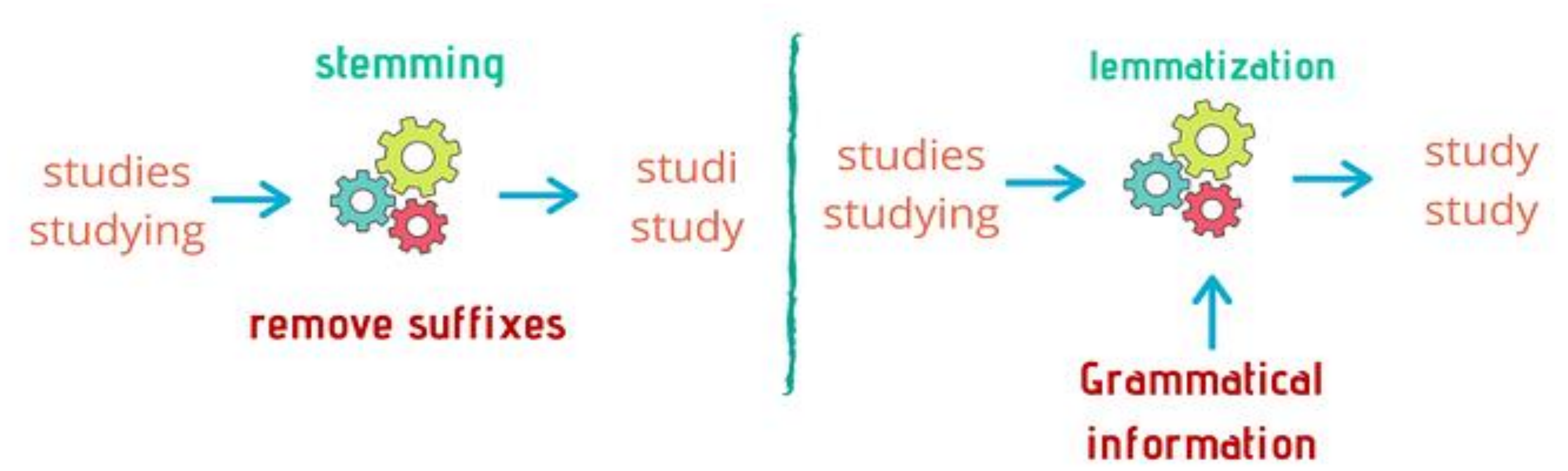
What is Lemmatization used for?

- To help **chatbots** understand your customers' queries to a better extent.
- To enable robots to speak and converse

What is the difference between Stemming and Lemmatization?

- In the process of **stemming**, the **end or beginning** of a **word** is cut off and **common prefixes, suffixes, and inflections** are taken into account.
- **Lemmatization** is a **morphological analysis** that **uses dictionaries** to find the **word's lemma (root form).**
- Lemmatization always returns the **dictionary meaning of the word** with a root-form conversion.

Lemmatization



Lemmatization

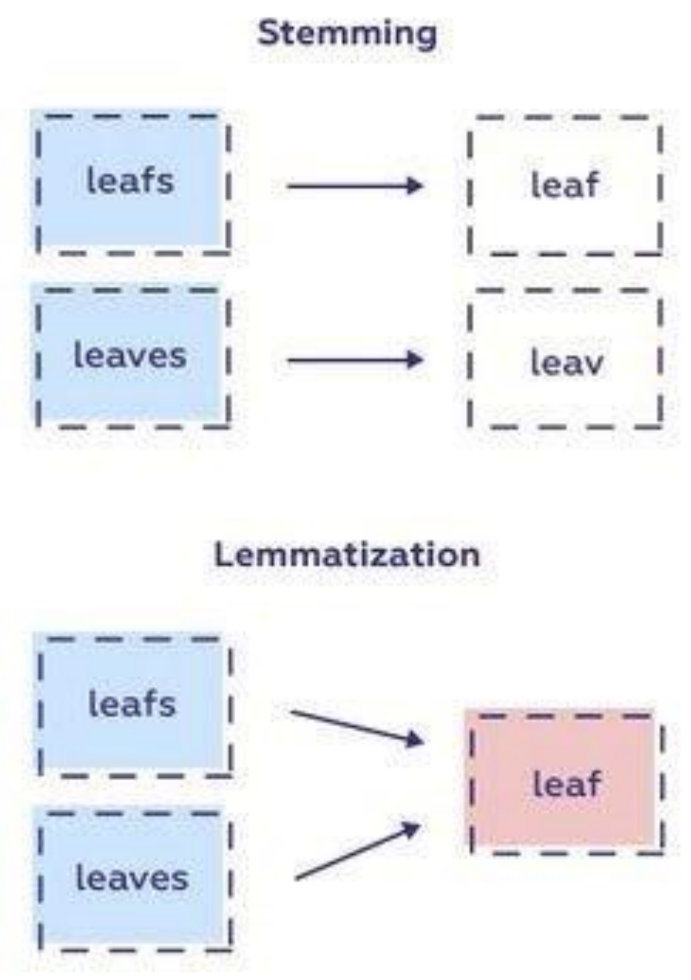
Why should you lemmatize your content?

- Lemmatization is **more accurate** than stemming, which means it will produce **better results** when you want to know the **meaning of a word**.
- This is especially **valuable** when you're working with a **chatbot** where it's important for the **bot** to **understand every message**.

The different ways in which lemmatization can be used?

- **Sentiment analysis**
- **Information Retrieval Environments**
- **Biomedical Sciences**
- **Document clustering**

NATURAL LANGUAGE PROCESSING
Lemmatization



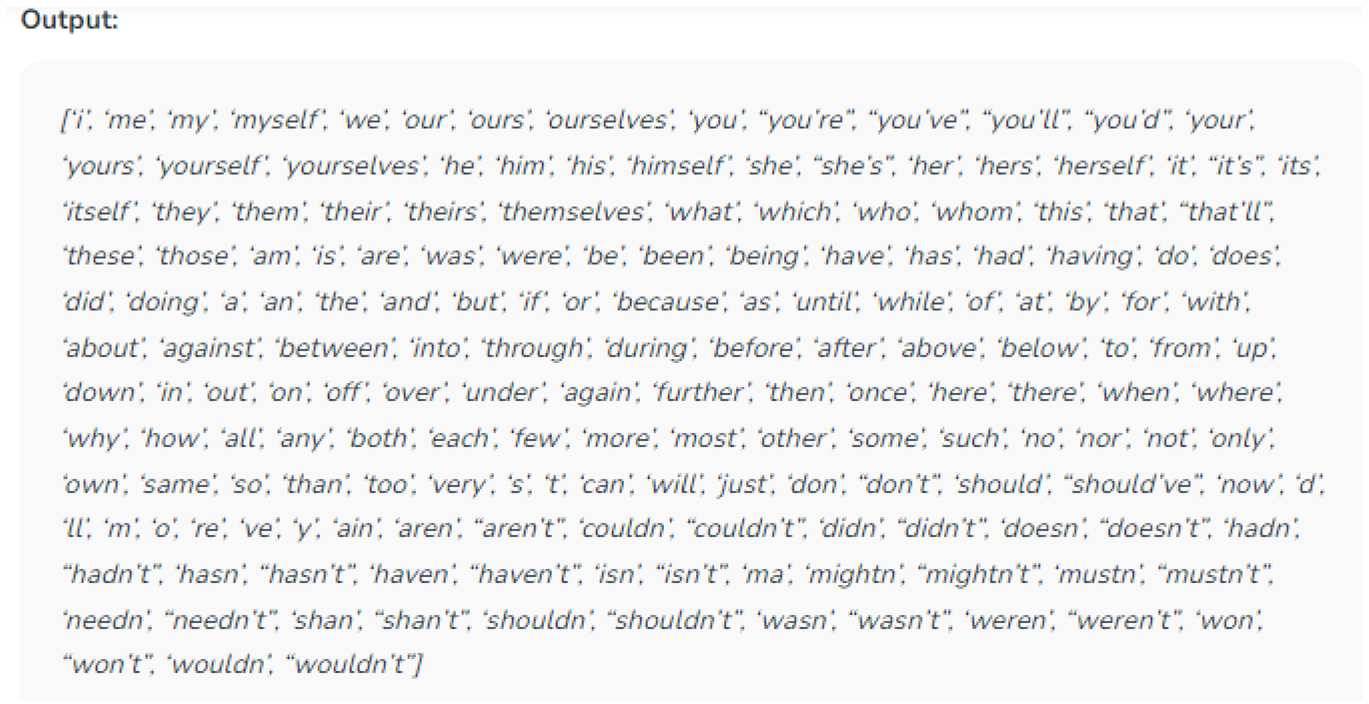
Stop Words

- **Stop words** are the **words** which are very common in **text documents** such as a, **an, the, you, your, etc**.
- The Stop Words highly appear in **text documents**.
- However, they are **not being helpful for text analysis** in many of the cases, So it is better to **remove from the text**.
- We would not want these words to **take up space** in our database, or taking up **valuable processing time**.
- For this, we can **remove them easily**, by **storing a list of words** that you consider to **stop words**.
- NLTK(Natural Language Toolkit) in python has a **list of stopwords** stored in **16 different languages.**

NATURAL LANGUAGE PROCESSING

Stop Words

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



Output:

```
[‘i’, ‘me’, ‘my’, ‘myself’, ‘we’, ‘our’, ‘ours’, ‘ourselves’, ‘you’, “you’re”, “you’ve”, “you’ll”, “you’d”, ‘your’,
‘yours’, ‘yourself’, ‘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’, ‘an’, ‘the’, ‘and’, ‘but’, ‘if’, ‘or’, ‘because’, ‘as’, ‘until’, ‘while’, ‘of’, ‘at’, ‘by’, ‘for’, ‘with’,
‘about’, ‘against’, ‘between’, ‘into’, ‘through’, ‘during’, ‘before’, ‘after’, ‘above’, ‘below’, ‘to’, ‘from’, ‘up’,
‘down’, ‘in’, ‘out’, ‘on’, ‘off’, ‘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’, ‘aren’, “aren’t”, ‘couldn’, “couldn’t”, ‘didn’, “didn’t”, ‘doesn’, “doesn’t”, ‘hadn’,
“hadn’t”, ‘hasn’, “hasn’t”, ‘haven’, “haven’t”, ‘isn’, “isn’t”, ‘ma’, ‘mightn’, “mightn’t”, ‘mustn’, “mustn’t”,
‘needn’, “needn’t”, ‘shan’, “shan’t”, ‘shouldn’, “shouldn’t”, ‘wasn’, “wasn’t”, ‘weren’, “weren’t”, ‘won’,
“won’t”, ‘wouldn’, “wouldn’t”]
```

Stop Words

Stop words can be used in a whole range of tasks and here are a few:

1. **Supervised machine learning** – removing stop words from the feature space
2. **Clustering** – removing stop words prior to generating clusters
3. **Information retrieval** – preventing stop words from being indexed
4. **Text summarization-** excluding stop words from contributing to summarization scores

Types of Stop Words

Examples of minimal stop word lists that you can use:

•**Determiners** – Determiners tend to **mark nouns** where a determiner usually will be followed by a **noun**
examples: **the, a, an, another**

•**Coordinating conjunctions** – Coordinating conjunctions **connect words**, **phrases**, and **clauses**
examples: **for, an, nor, but, or, yet, so**

•**Prepositions** – **Prepositions** express **temporal** or **spatial relations**
examples**: in, under, towards, before**

NATURAL LANGUAGE PROCESSING

Parts of Speech Tagging

POS (parts of speech)

- **Parts of speech (POS)** are **useful because they reveal a lot about a word and its neighbors.**
- **Part-of-speech (POS) tagging is a** process in natural language processing (NLP) where **each word** in a **text** is **labeled** with its **corresponding part of speech**.
- This can include **nouns, verbs, adjectives, and other grammatical categories.**
- **Knowing a word POS(noun or verb, …) tells us about**
  - likely neighboring words (nouns are preceded by determiners and adjectives, verbs by nouns) and
  - syntactic structure (nouns are generally part of noun phrases), making POS tagging a key aspect of parsing.

POS (parts of speech)

Parts of speech are useful features

1) POS tagging is **useful for a variety of NLP tasks**, such as **information extraction**, **named entity recognition**, and **machine translation**.
   - for labeling named entities(NER) like **people** or **organizations** in information extraction, or
   - for coreference resolution**(**the task of finding all expressions that refer to the **same entity in a text)**
   - **for sentiment analysis, question answering, and word sense disambiguation.**

2) It can also be used to identify the grammatical structure of a sentence and to disambiguate words that have multiple meanings.

3) To improve the accuracy of NLP tasks

4) To facilitate research in linguistics

POS (parts of speech)

Let’s take an example,

Text: “The cat sat on the mat.”

POS tags:

The: determiner
cat: noun
sat: verb
on: preposition
the: determiner
mat: noun

In this example, each word in the sentence has been labeled with its corresponding part of speech.

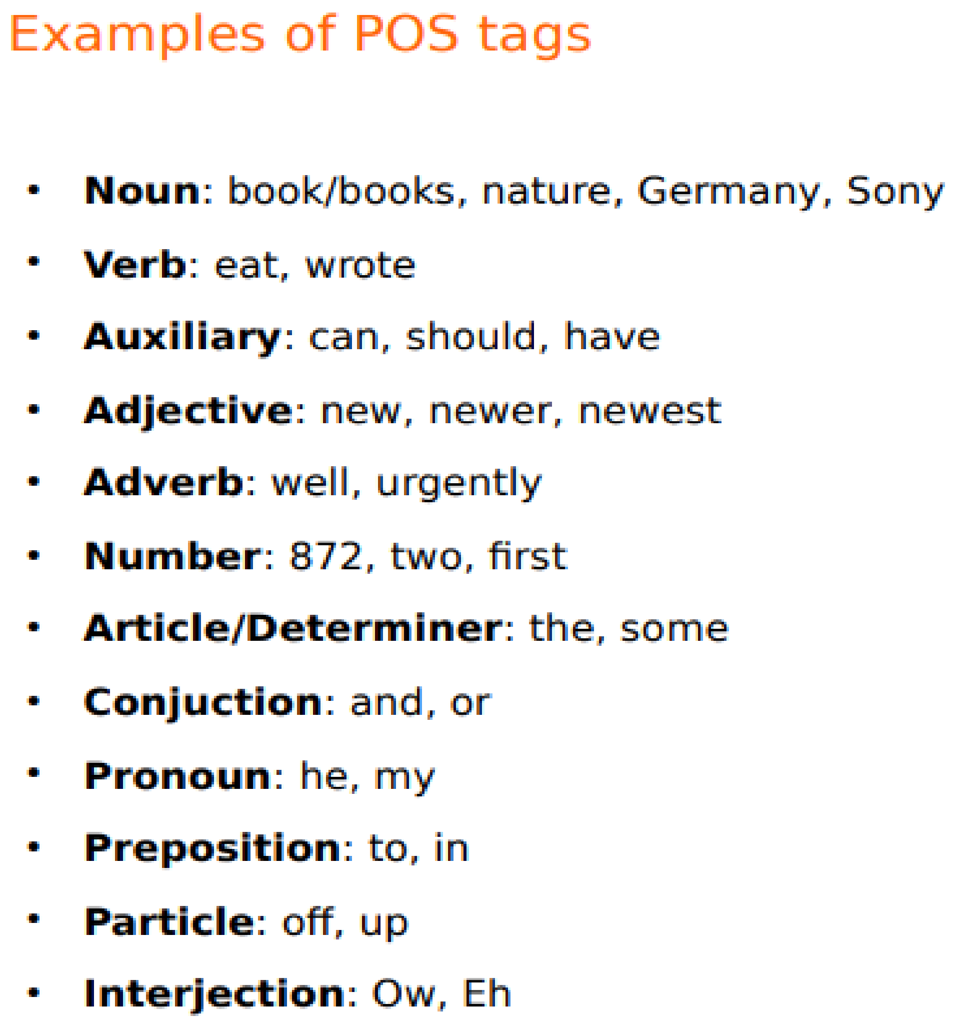
The determiner “**the**” is used to identify specific **nouns**, while the **noun** “**cat**” refers to a specific **animal**.

The **verb** “**sat**” describes an **action**, and the **preposition** “**on**” describes the **relationship between the cat and the mat.**

POS (parts of speech) examples

1)	N	**noun**	chair, bandwidth, pacing
2)	V	**verb**	study, debate, munch
3)	ADJ	**adjective**	purple, tall, ridiculous
4)	ADV	**adverb**	unfortunately, slowly,
5)	P	**preposition**	of, by, to
6)	PRO	**pronoun**	I, me, mine
7)	DET	**determiner**	the, a, that, those
8)	Conj	**conjunctions**	and, or

POS (parts of speech) examples



Examples of POS tags

- **Noun**: book/books, nature, Germany, Sony
- **Verb**: eat, wrote
- **Auxiliary**: can, should, have
- **Adjective**: new, newer, newest
- **Adverb**: well, urgently
- **Number**: 872, two, first
- **Article/Determiner**: the, some
- **Conjuction**: and, or
- **Pronoun**: he, my
- **Preposition**: to, in
- **Particle**: off, up
- **Interjection**: Ow, Eh

POS (parts of speech) examples

- **Application of POS Tagging**
  - Information extraction
  - Named entity recognition
  - Text classification
  - Machine translation
  - Natural language generation

POS (parts of speech)

- **Parts of speech can be divided into two broad categories:**
  - **Closed class type**
  - **Open class type**

- Closed
  - limited number of words, do not grow usually
  - e.g., Auxiliary, Article, Determiner, Conjuction, Pronoun, Preposition, Particle, Interjection

- Open
  - unlimited number of words
  - e.g., Noun, Verb, Adverb, Adjective

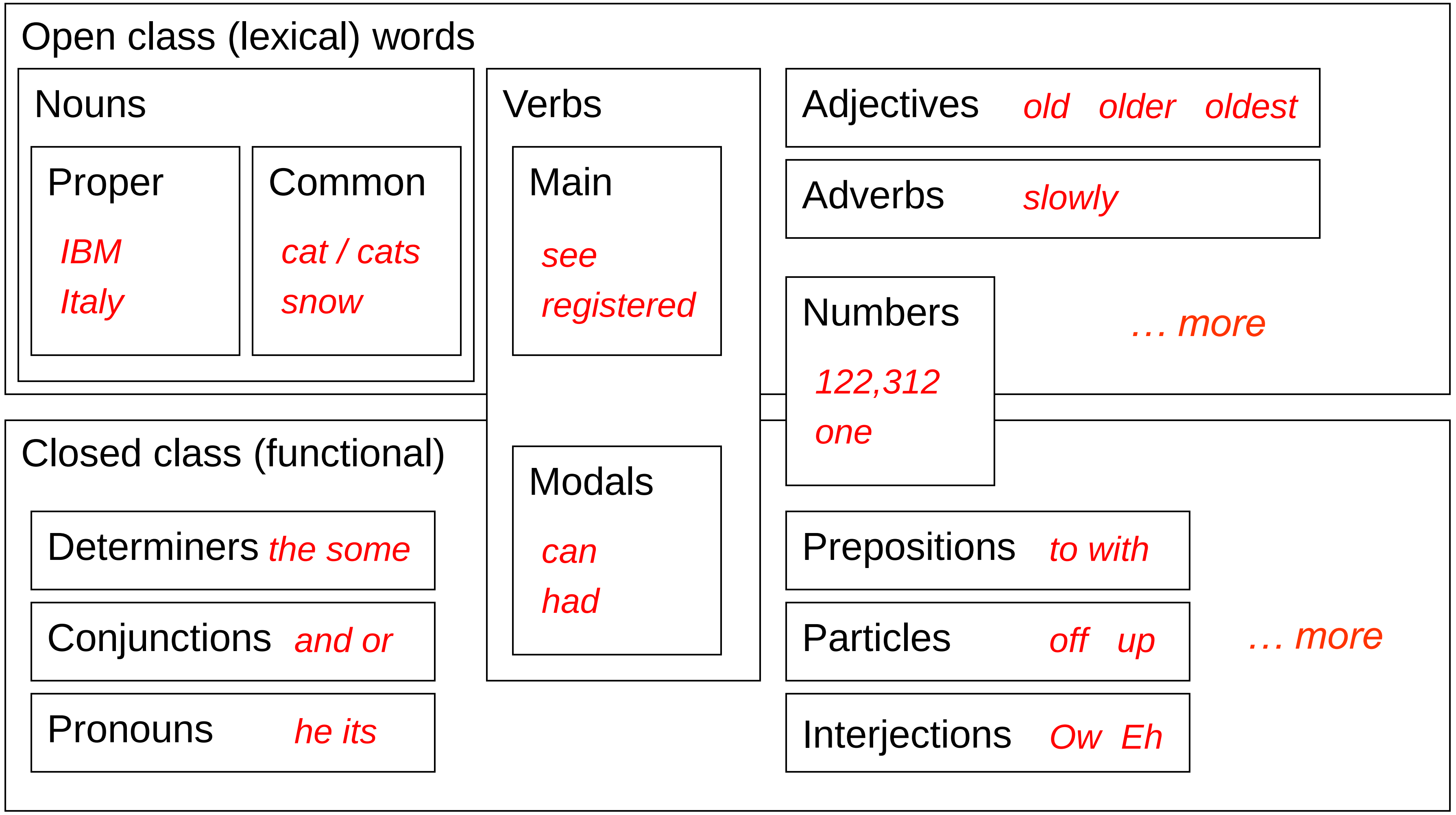
POS (parts of speech)

- **Closed classes** No inclusion of new words very often.
  - Closed classes are those with relatively **fixed membership**, such as **prepositions**—**new prepositions** are rarely coined. Like preposition, aux verbs etc
- **Closed class words are generally function words** like of, it, and, or you, **occur frequently.**
- **Nouns** and **verbs** are **open classes**—new nouns and verbs like **iPhone** or to fax are continually being created or borrowed.

POS (parts of speech)

- **The important closed classes in English include:**
  - **Prepositions**: on, under, over, near, by, at, from, to, with
  - **Particles**: up, down, on, off, in, out, at, by
  - **Determiners**: a, an, the, *this/that, these/those, its, our, their*
  - **Conjunctions**: and, but, or, as, if, when
  - **Pronouns**: she, who, I, others
  - **Auxiliary verbs**: can, may, should, is, are, do, have, ...
  - **Numerals**: one, two, three, first, second, third, …

POS (parts of speech)



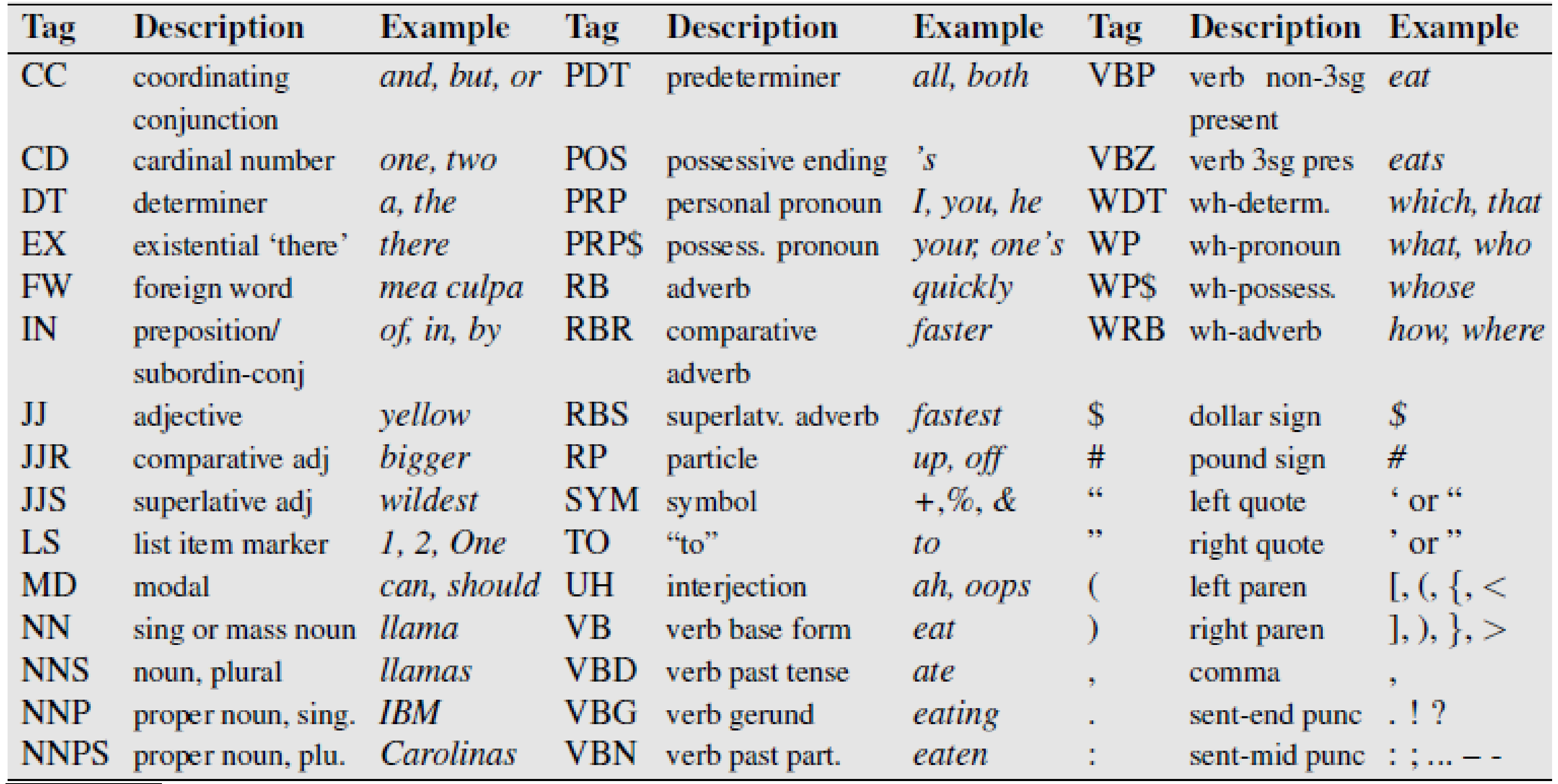
POS (parts of speech)

- **What are POS tags?**

  POS tags are also known as **word classes, morphological classes**, or **lexical tags**
- **Number of tags used by different systems/corpora/languages are different**
  - **Penn Treebank** (Wall Street Journal Newswire): **45 tags**
  - **Brown corpus** (Mixed genres like fiction, biographies, etc): **87 tags**
  - **Lancaster UCREL C5: 61 tags**
  - **Lancaster C7: 145 tags**

- POS tags can be of varying granularity.
- Morphologically complex languages have very different tag-sets from English or Dutch.

POS (parts of speech)



Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	*and, but, or*	PDT	predeterminer	*all, both*	VBP	verb non-3sg present	*eat*
CD	cardinal number	*one, two*	POS	possessive ending	*’s*	VBZ	verb 3sg pres	*eats*
DT	determiner	*a, the*	PRP	personal pronoun	*I, you, he*	WDT	wh-determ.	*which, that*
EX	existential ‘there’	*there*	PRP$	possess. pronoun	*your, one’s*	WP	wh-pronoun	*what, who*
FW	foreign word	*mea culpa*	RB	adverb	*quickly*	WP$	wh-possess.	*whose*
IN	preposition/ subordin-conj	*of, in, by*	RBR	comparative adverb	*faster*	WRB	wh-adverb	*how, where*
JJ	adjective	*yellow*	RBS	superlatv. adverb	*fastest*	$	dollar sign	$
JJR	comparative adj	*bigger*	RP	particle	*up, off*	#	pound sign	#
JJS	superlative adj	*wildest*	SYM	symbol	+,%, &	“	left quote	‘ or “
LS	list item marker	*1, 2, One*	TO	“to”	*to*	”	right quote	’ or ”
MD	modal	*can, should*	UH	interjection	*ah, oops*	(	left paren	[, (, {, <
NN	sing or mass noun	*llama*	VB	verb base form	*eat*	)	right paren	],), }, >
NNS	noun, plural	*llamas*	VBD	verb past tense	*ate*	,	comma	,
NNP	proper noun, sing.	*IBM*	VBG	verb gerund	*eating*	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	*Carolinas*	VBN	verb past part.	*eaten*	:	sent-mid punc	: ; ... – -

Figure 8.1 Penn Treebank part-of-speech tags (including punctuation).

POS (parts of speech)

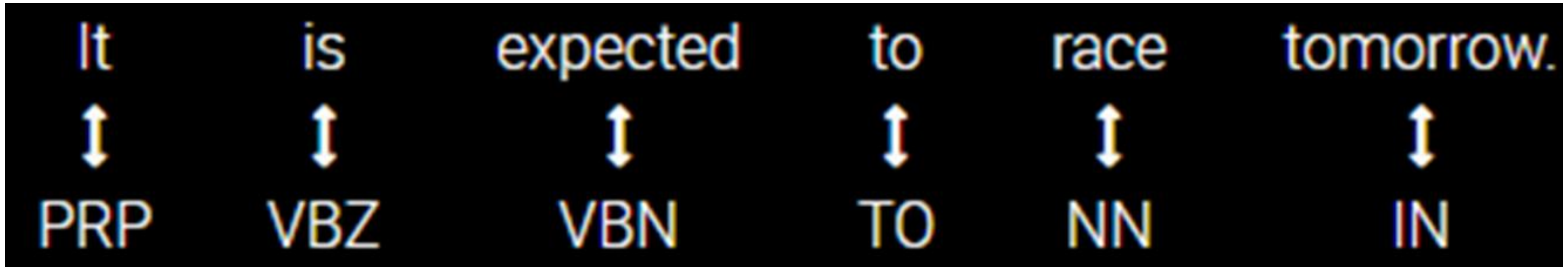
Abbreviation	Meaning
CC	coordinating conjunction
CD	cardinal digit
DT	determiner
EX	existential there
FW	foreign word
IN	preposition/subordinating conjunction
JJ	This NLTK POS Tag is an adjective (large)
JJR	adjective, comparative (larger)
JJS	adjective, superlative (largest)
LS	list market
MD	modal (could, will)
NN	noun, singular (cat, tree)
NNS	noun plural (desks)
NNP	proper noun, singular (sarah)
NNPS	proper noun, plural (indians or americans)
PDT	predeterminer (all, both, half)
POS	possessive ending (parent\ ‘s)
PRP	personal pronoun (hers, herself, him, himself)
PRP$	possessive pronoun (her, his, mine, my, our)
RB	adverb (occasionally, swiftly)
RBR	adverb, comparative (greater)
RBS	adverb, superlative (biggest)
RP	particle (about)
TO	infinite marker (to)
UH	interjection (goodbye)

POS (parts of speech)

VB	verb (ask)
VBG	verb gerund (judging)
VBD	verb past tense (pleaded)
VBN	verb past participle (reunified)
VBP	verb, present tense not 3rd person singular(wrap)
VBZ	verb, present tense with 3rd person singular (bases)
WDT	wh-determiner (that, what)
WP	wh- pronoun (who)
WRB	wh- adverb (how)

POS tagging

- Part-of-speech tagging is the **process of assigning a part-of-speech marker to each word in an input text**.



It	is	expected	to	race	tomorrow.
↕	↕	↕	↕	↕	↕
PRP	VBZ	VBN	TO	NN	IN

- The input to a tagging algorithm is a **sequence of (tokenized) words** and a **tagset**, and the **output** is a **sequence of tags, one per token.**

POS tagging

- **Tagging is a disambiguation task; words are ambiguous - have more than one possible POS - and the goal is to find the correct tag for the situation.**
- For example,
  - **book** can be a **verb** (*book* that flight) or a **noun** (hand me that *book*).
  - ***That*** can be a **determiner** (Does that flight serve dinner) or a **complementizer** (I thought *that* your flight was earlier).
- **The goal of POS-tagging is to resolve these ambiguities, choosing the proper tag for the context.**

Why is POS tagging hard?

- **Ambiguity**

1. **“Plants/N need light and water.”**

   “Each one plant/V one.”

2. **“Flies like a flower”**
   - Flies: noun or verb?
   - like: preposition, adverb, conjunction, noun, or verb?
   - a: article, noun, or preposition?
   - flower: noun or verb?

Why is POS tagging hard?

- **Words often have more than one POS**
- **For example, the word *back* in following sentences:**

1. The *back* door = JJ(Adjective)
2. On my *back* = NN(Noun singular)
3. Win the voters *back* = RB(Adverb)
4. Promised to *back* the bill = VB(Verb base form)

- **The POS tagging problem is to determine the POS tag for a particular instance of a word.**

3 approaches for POS tagging

1. **Rule-based tagging**
   - The ENGTWOL tagger (Voutilainen, 1995) is a rule based tagger based on two-stage architecture.
2. **Transformation-based tagging**
   - Brill tagger
3. **Stochastic (Probability/Frequency) tagging**
   - Most frequent tag algorithm
   - HMM (Hidden Markov Model) tagging

3 approaches for POS tagging

Rule-based part-of-speech (POS) tagging

- It is a method of **labeling words** with their **corresponding** parts of **speech** using a **set of pre-defined rules.**
- **words** are assigned **POS tags** based on their **characteristics** and the context in which they appear.
- For example, a **rule-based** POS tagger might assign the **tag** "**noun**" to any word that **ends** in **"-tion**" or **"-ment**," as these **suffixes** are often used to form **nouns**.

- Here is an example of how a rule-based POS tagger might work:
  - **Define a set of rules** for assigning POS tags to words.
  - For example:
    - If the word ends in "-tion," assign the tag "noun."
    - If the word ends in "-ment," assign the tag "noun."
    - If the word is all uppercase, assign the tag "proper noun."
    - If the word is a verb ending in "-ing," assign the tag "verb."

3 approaches for POS tagging

- Iterate through the words in the text and apply the rules to each word in turn.
- For example:
  - "Nation" would be tagged as "noun" based on the first rule.
  - "Investment" would be tagged as "noun" based on the second rule.
  - "UNITED" would be tagged as "proper noun" based on the third rule.
  - "Running" would be tagged as "verb" based on the fourth rule.
- Output the POS tags for each word in the text.

3 approaches for POS tagging

- **Statistical POS Tagging**
- Statistical part-of-speech (POS) tagging is a method of **labeling words** with their corresponding **parts of speech** using **statistical techniques**.
- This is in contrast to **rule-based** POS tagging, which relies on **pre-defined rules**, and to unsupervised learning-based POS tagging, which **does not use any annotated training data.**
- In **statistical POS tagging**, a model is **trained** on a **large annotated corpus of text** to learn the **patterns** and **characteristics** of **different parts of speech**.
- The model uses this training data to **predict the POS tag** of a given **word** based on the **context** in which it appears and the **probability** of different POS tags occurring in that context.
- Statistical POS taggers can be **more accurate** and **efficient** than rule-based taggers, especially for tasks with **large or complex datasets**.
- However, they **require a large amount of annotated training data** and can be **computationally intensive to train.**
- **statistical POS tagging uses machine learning algorithms, such as Hidden Markov Models (HMM) or Conditional Random Fields (CRF), to predict POS tags based on the context of the words in a sentence**

3 approaches for POS tagging

- Here is an example of how a statistical POS tagger might work:
  - Collect a large annotated corpus of text and divide it into training and testing sets.
  - Train a statistical model on the training data, using techniques such as maximum likelihood estimation or hidden Markov models.
  - Use the trained model to predict the POS tags of the words in the testing data.
  - Evaluate the performance of the model by comparing the predicted tags to the true tags in the testing data and calculating metrics such as precision and recall.
  - Fine-tune the model and repeat the process until the desired level of accuracy is achieved.
  - Use the trained model to perform POS tagging on new, unseen text

3 approaches for POS tagging

- Transformation-based tagging (TBT)
  - Transformation-based tagging (TBT) is a method of part-of-speech (POS) tagging that uses a **series of rules to transform the tags of words in a text**.
  - This is in contrast to rule-based POS tagging, which assigns tags to words based on pre-defined rules, and to statistical POS tagging, which relies on a trained model to predict tags based on probability.
  - In TBT, a **set of rules** is **defined** to **transform** the **tags of words** in a text based on the context in which they appear.
  - For example, a **rule might change the tag of a verb to a noun** if it appears after a **determiner** such as "**the**."
  - The rules are applied to the text in a **specific order**, and the tags are updated after **each transformation**.
  - TBT can be **more accurate** than rule-based tagging, especially for tasks with complex grammatical structures.
  - However, it can be more computationally intensive and requires a larger set of rules to achieve good performance.

3 approaches for POS tagging

- Here is an example of how a TBT system might work:
  - Define a **set of rules for transforming the tags of words** in the text.
  - For example:
  - If the **word** is a **verb** and appears after a **determiner**, change the tag to "**noun**."
  - If the **word** is a **noun** and appears after an **adjective**, change the tag to "**adjective**."
  - Iterate through the words in the text and apply the rules in a specific order. For example:
  - In the sentence "**The cat sat on the mat,"** the word "**sat**" would be **changed** from a **verb** to a **noun** based on the **first rule**.
  - In the sentence "**The red cat sat on the mat**," the **word** "**red**" would be **changed** from an **adjective** to a **noun** based on the second rule.
- Output the transformed tags for each word in the text.

3 approaches for POS tagging

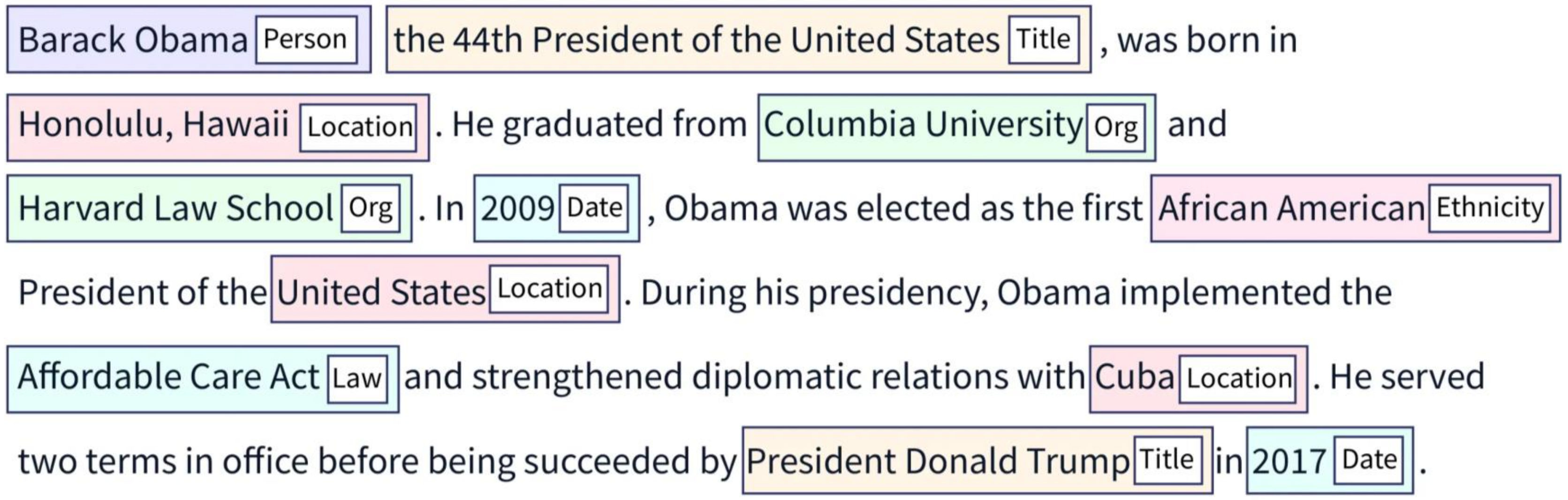
- The **benefits** of **rule-based POS taggers:**
  - Simple to implement and understand
  - It doesn’t require a lot of computational resources or training data
  - It can be easily customized to specific domains or languages
- **Disadvantages** of rule-based POS taggers:
  - Less accurate than statistical taggers
  - Limited by the quality and coverage of the rules
  - It can be difficult to maintain and update
- The **Benefits** of **statistical POS Tagger**:
  - More accurate than rule-based taggers
  - Don’t require a lot of human-written rules
  - Can learn from large amounts of training data
- **Disadvantages** of **statistical POS Tagger**:
  - Requires more computational resources and training data
  - It can be difficult to interpret and debug
  - Can be sensitive to the quality and diversity of the training data

Named Entity Recognization (NER)

- Named Entity Recognition (NER) is a **subfield** of Natural Language Processing (NLP) that focuses on **extracting** and **classifying** named entities in text.
- Named entities are specific terms that represent **real-world objects**, such as **people**, **organizations**, **locations**, and **dates**.
- The named entity recognition NLP model aims to **identify and classify** these named entities to **extract useful information** from **unstructured text data.**
- Named entities are specific terms that represent real-world objects, such as **people, organizations, locations, and dates**. NER aims to extract these named entities from text and classify them into predefined categories.

Named Entity Recognization (NER)

- For example, consider the following sentence: **"Barack Obama was born in Hawaii on August 4, 1961."**
- In this sentence, "**Barack Obama**" is a **person**, "**Hawaii**" is a **location**, and "**August 4, 1961**" is a **date**.
- We can extract valuable information about Barack Obama's birthplace and birthdate by identifying and classifying these named entities.



Barack Obama Person the 44th President of the United States Title , was born in Honolulu, Hawaii Location . He graduated from Columbia University Org and Harvard Law School Org . In 2009 Date , Obama was elected as the first African American Ethnicity President of the United States Location . During his presidency, Obama implemented the Affordable Care Act Law and strengthened diplomatic relations with Cuba Location . He served two terms in office before being succeeded by President Donald Trump Title in 2017 Date .

Named Entity Recognization (NER)

- A typically named entity recognition NLP model consists of several components, including:
  - **Tokenization:** Tokenization breaks **text** into **individual tokens** (usually words or punctuation marks).
  - **Part-of-speech tagging: Labelling** each **token** with its corresponding **part of speech (e.g., noun, verb, adjective, etc.).**
  - **Chunking:** Group **tokens** into "**chunks**" based on their **part-of-speech tags**.
  - **Name entity recognition:** Identifying **named entities** and **classifying** them into **predefined categories**.
  - **Entity disambiguation:** The process of determining the **correct meaning** of a named entity, especially when multiple entities with the same name are present in the text.

Named Entity Recognization (NER)

- let's walk through an example using the following sentence:

"Mark Zuckerberg founded Facebook in 2004 in Menlo Park, California."

1. **Tokenization:** The first step in our NER process is to tokenize the text, which means breaking it into individual tokens. In this case, our tokens would be: **['Mark', 'Zuckerberg', 'founded', 'Facebook', 'in', '2004', 'in', 'Menlo', 'Park', ',', 'California', '.']**
2. **Part-of-speech tagging:** We would label each token with its corresponding part of speech. This step might produce the following tags: **['NNP', 'NNP', 'VBD', 'NNP', 'IN', 'CD', 'IN', 'NNP', 'NNP', ',', 'NNP', '.']**
3. **Chunking:** Using the part-of-speech tags, we can now group the tokens into "chunks" based on their tags. In this case, we might have the following chunks: **[('Mark', 'NNP'), ('Zuckerberg', 'NNP'), ('founded', 'VBD'), ('Facebook', 'NNP'), ('in', 'IN'), ('2004', 'CD'), ('in', 'IN'), ('Menlo', 'NNP'), ('Park', 'NNP'), (',', ','), ('California', 'NNP'), ('.', '.')]**
4. **Named entity recognition:** Using the information from the chunking step, we can now identify and classify the named entities in the text. In this case, we have two named entities: "**Mark Zuckerberg**" and "**Facebook**", both of which are **people**, and "**Menlo Park, California", which is a location**.
5. **Entity disambiguation:** In this step, we would determine each named entity's meaning. For example, if multiple people have the name "**Mark Zuckerberg**", we must determine which one is being referred to in the text.

Named Entity Recognization (NER)

- **How does Named Entity Recognition Work?**
- Several approaches can be used to perform named entity recognition NLP models. The most common methods include the following:
- **Rule-based** methods use a set of predefined rules and patterns to identify named entities in text.
- **Statistical** methods use a probabilistic framework to identify named entities in a text by training a model on a large annotated text corpus.
- **Machine learning** methods also use probabilistic frameworks but rely on Machine Learning algorithms to learn the patterns in the data.
- Once the model is trained, we can identify named entities in a new text by applying the learned patterns and features.
- **Machine learning-based** methods tend to be more accurate and scalable than rule-based methods, but they require more labeled training data

Named Entity Recognization (NER)

- **How can I use NER Model?**
- Some common use cases include:
- **Extracting contact information from emails or online forms:** NER can extract contact information like names, phone numbers, and email addresses from emails or online forms.
- **Analyzing customer feedback:** NER can classify customer feedback by extracting product names and classifying the feedback as positive, negative, or neutral.
- **Summarizing news articles or social media posts:** NER can extract key information from news articles or social media posts, like main actors, locations, and events, to generate automatic summaries.
- **Identifying named entities in legal documents:** NER can extract important information from legal documents, such as contracts and agreements, by identifying the names of people, organizations, and locations.

Named Entity Recognization (NER)

- **Use-cases of Named Entity Recognition**
- **Healthcare:** Named Entity Recognition NLP models can extract patient information and diagnoses from medical records and identify drugs and treatments mentioned in the text.
- **Cyber Security:** NER can be used in cyber security to extract and classify entities such as IP addresses, URLs, and file names from security logs, network traffic, and other sources, which can help to identify and track cyber threats.
- **Finance:** NER can extract financial information from news articles, financial reports, and other text sources.
- **Marketing:** NER can analyze customer feedback and identify common themes or issues.
- **Human Resource:** NER can extract important information from resumes and job postings, such as job titles, names, and qualifications, which We can use to automate the recruiting and hiring process.
- **Legal:** NER can be used to identify named entities in legal documents for analysis and summarization.
- **Education:** NER can be used to extract important information from educational materials, such as named entities, concepts, and topics covered in the text, which can help to index, organize and search the educational materials and also help the students to search the information related to their queries.

Named Entity Recognization (NER)

- **Use-cases of Named Entity Recognition**
- **Healthcare:** Named Entity Recognition NLP models can extract patient information and diagnoses from medical records and identify drugs and treatments mentioned in the text.
- **Cyber Security:** NER can be used in cyber security to extract and classify entities such as IP addresses, URLs, and file names from security logs, network traffic, and other sources, which can help to identify and track cyber threats.
- **Finance:** NER can extract financial information from news articles, financial reports, and other text sources.
- **Marketing:** NER can analyze customer feedback and identify common themes or issues.
- **Human Resource:** NER can extract important information from resumes and job postings, such as job titles, names, and qualifications, which We can use to automate the recruiting and hiring process.
- **Legal:** NER can be used to identify named entities in legal documents for analysis and summarization.
- **Education:** NER can be used to extract important information from educational materials, such as named entities, concepts, and topics covered in the text, which can help to index, organize and search the educational materials and also help the students to search the information related to their queries.

Chunking

- Chunking is **extracting phrases** from an **unstructured text** by evaluating a **sentence** and **determining** its **elements (Noun Groups, Verbs, verb groups, etc.)**
- However, it does **not describe** their **internal structure** or their **function** in the introductory statement.
- To identify and **group noun phrases** or **nouns alone**, **adjectives** or **adjective** phrases, and so on, Chunking can be used.
- Consider the following sentence:
  - **"I had my breakfast, lunch and dinner."**
  - In this case, if we wish to **group** or **chunk noun phrases**, we will get "**breakfast**", "**lunch**", and "**dinner**", which are the **nouns** or **noun** groups of the sentence.

Chunking

- **Why do we need Chunking?**
  - It's critical to understand that the statement contains a **person**, a **date**, and a **location** (different entities). As a result, they're useless on their own.
  - **Chunking** can **break down** sentences into **phrases** that are more useful than **single words** and provide **meaningful outcomes**.
  - When extracting information from **text**, such as **places** and **person** names, **Chunking** is critical. (extraction of entities)

Chunking

- **Types of Chunking**
- **Chunking Up**
  - We don't go into great detail here; instead, we're content with a high-level overview.
  - It only serves to provide us with a quick overview of the facts.
- **Chunking Down**
  - Unlike the previous method of Chunking, chunking down allows us to obtain more detailed data.
  - Consider "chunking up" if you only need an insight; otherwise, "chunking down" is preferable.

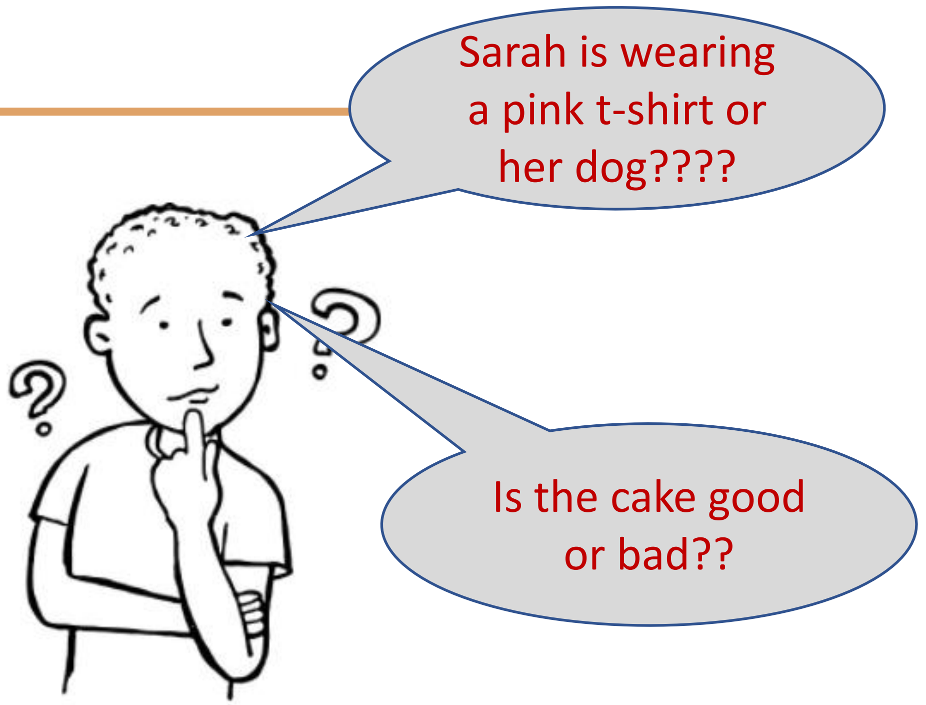
PES
UNIVERSITY
CELEBRATING 50 YEARS

THANK YOU

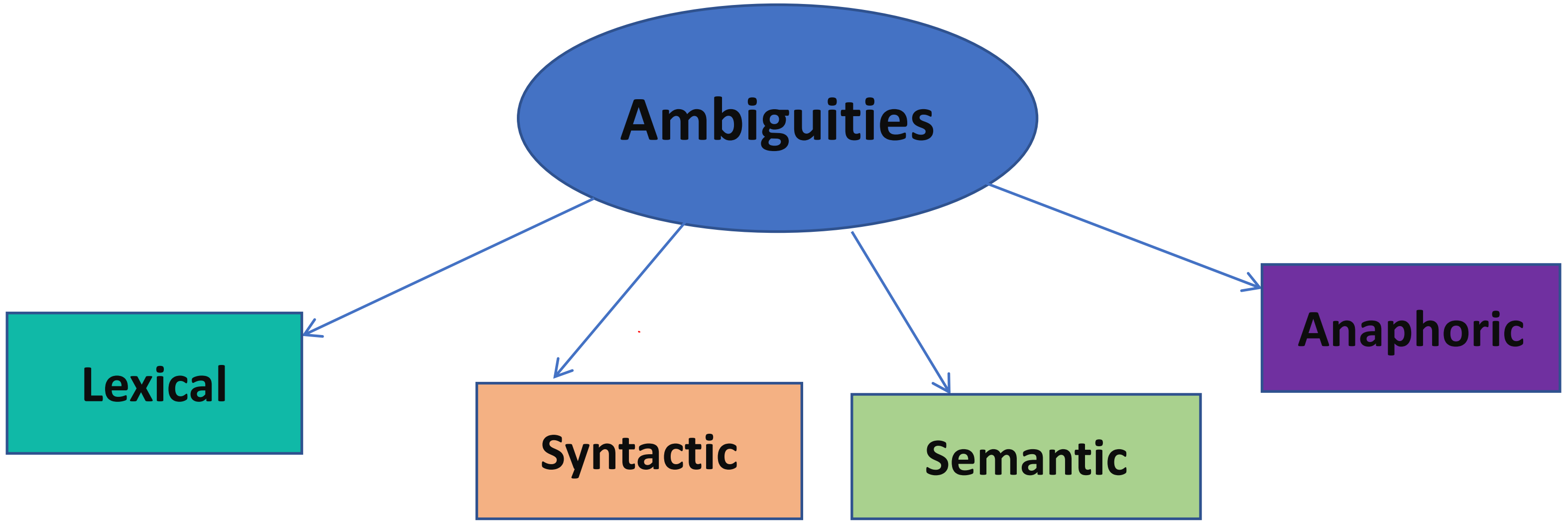
Prof. Ananthanagu
ananthanagu@pes.edu

Ambiguity

- Sarah gave a bath to her dog wearing a pink t-shirt.
- I have never tasted a cake quite like that one before!


Sarah is wearing a pink t-shirt or her dog????
Is the cake good or bad??

Types of Ambiguities



Lexical Ambiguity:

- Related to words.
- This type of ambiguity represents words that can have multiple assertions.
- For instance, in English, the word “back” can be a noun (back stage), an adjective (back door) or an adverb (back away).

Words have multiple meanings.

"I **saw** a **bat**."

bat = flying mammal / wooden club/ Sports equipment?

saw = past tense of "see" / present tense of "saw" (to cut with a saw.)

Syntactic Ambiguities

Syntactic Ambiguity:

This type of ambiguity represents sentences that how a particular sentence can be *parsed in multiple syntactical forms.*
In multiple Structural representation which one needs to be selected.

Eg.
“ I heard his cell phone ring in my office”.

The propositional phrase “in my office” can be parsed in a way

- that modifies the noun or
- another way that modifies the verb.

Syntactic Ambiguity

- Another Example:

- Mary ate a salad with spinach from California for lunch on Tuesday.“
  - **"with spinach**" can attach to "**salad**" or "**ate**“,
  - "**from California**" can attach to "**spinach**", "**salad**", or "**ate**".
  - "**for lunch**" can attach to "**California**", "**spinach**", "**salad**", or "**ate**"
  - and "on Tuesday" can attach to "lunch", "California", "spinach", "salad" or "ate".

- Nonetheless there are 42 possible different parse trees for this sentence.

Semantic Ambiguity

Semantic Ambiguity: Related to the interpretation of sentence. OR How you interpret the meaning of entire senetence.

- Eg.,
  - I heard his cell phone ring in my office can be interpreted as if "*I was physically present in the office*" or as if "*the cell phone was in the office*".
  - *Lucy owns a parrot* (existentially quantified) *that is larger than a cat* (either universally quantified or means "typical cats")

Another Example

- "The dog is chasing the cat." vs. "The dog has been domesticated for 10,000 years."
- In the first sentence, "The dog" means to a particular dog;
- In the second, it means the species "dog".

Anaphoric Ambiguity

- **Anaphoric ambiguity**
- A phrase or word refers to something previously mentioned, but there is more than one possibility for machine to understand that word.
- Eg.
- "Margaret invited Susan for a visit, and she gave her a good lunch." (she = Margaret; her = Susan)
- "Margaret invited Susan for a visit, but she told her she had to go to work" (she = Susan; her = Margaret.)
- "On the train to Boston, George chatted with another passenger. The man turned out to be a professional hockey player."
- (The man = another passenger).

Metonymy Ambiguity

Metonymy:
The most difficult type of ambiguity, *metonymy* deals with phrases in which the *literal meaning is different from the figurative assertion.*

Eg.,
Samsung is screaming for new management.

Here screaming doesn’t literally mean yelling.

NATURAL LANGUAGE PROCESSING
Examples

- Find the different types of ambiguities in the following sentence.

  "Elsa tried to reach her aunt on the phone, but she didn't answer".

Lexical ambiguity:

The word "tried" means "attempted" not "held a court proceedings", or "test" (as in "Elsa tried the lemonade"). The word "reach" means "establish communication" not "physically arrive at" (as in "the boat reached the shore").

- Syntactic ambiguity:

The phrase "on the phone" attaches to "reach" and thus means "using the phone" not to "aunt" which would mean "her aunt who was physically on top of the phone" (compare "her aunt in Seattle").

- Anaphoric ambiguity:

"she" means the aunt, not Elsa.

Ambiguities

A famous example is to determine the sense of pen in the following passage (Bar-Hillel 1960):

Little John was looking for his toy box. Finally, he found it. The box was in the pen. John was very happy.

WordNet lists five senses for the word pen:

pen — a writing instrument with a point from which ink flows.

pen — a small enclosure in which sheep, pigs, or other farm animals are kept.

playpen, pen — a small portable enclosure in which babies can play safely.

penitentiary, pen — a correctional institution for those convicted of major crimes.

pen — female swan.