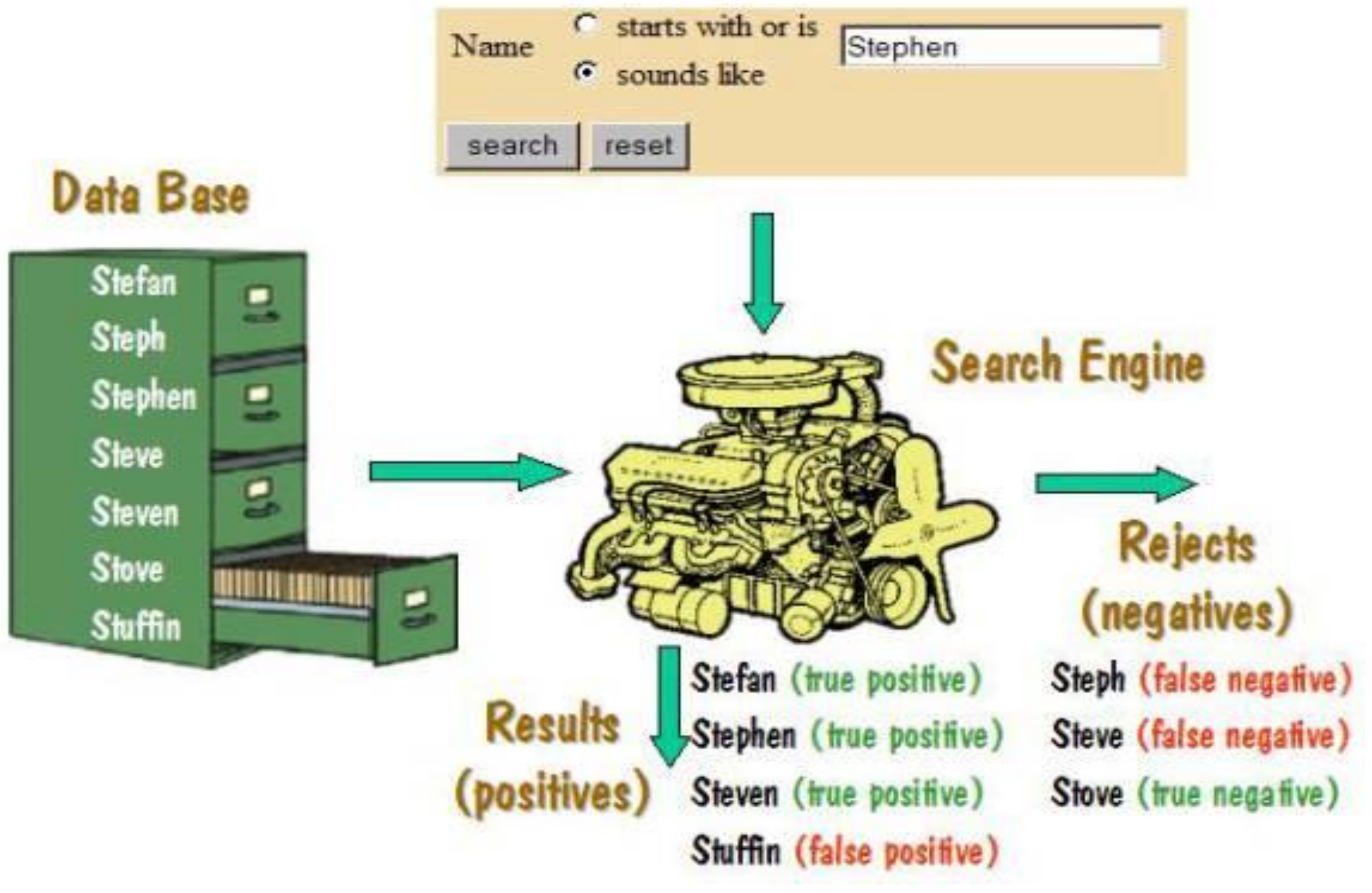
Introduction au Traitement Automatique des Langues

4 - Les niveaux de traitement – le niveau lexical

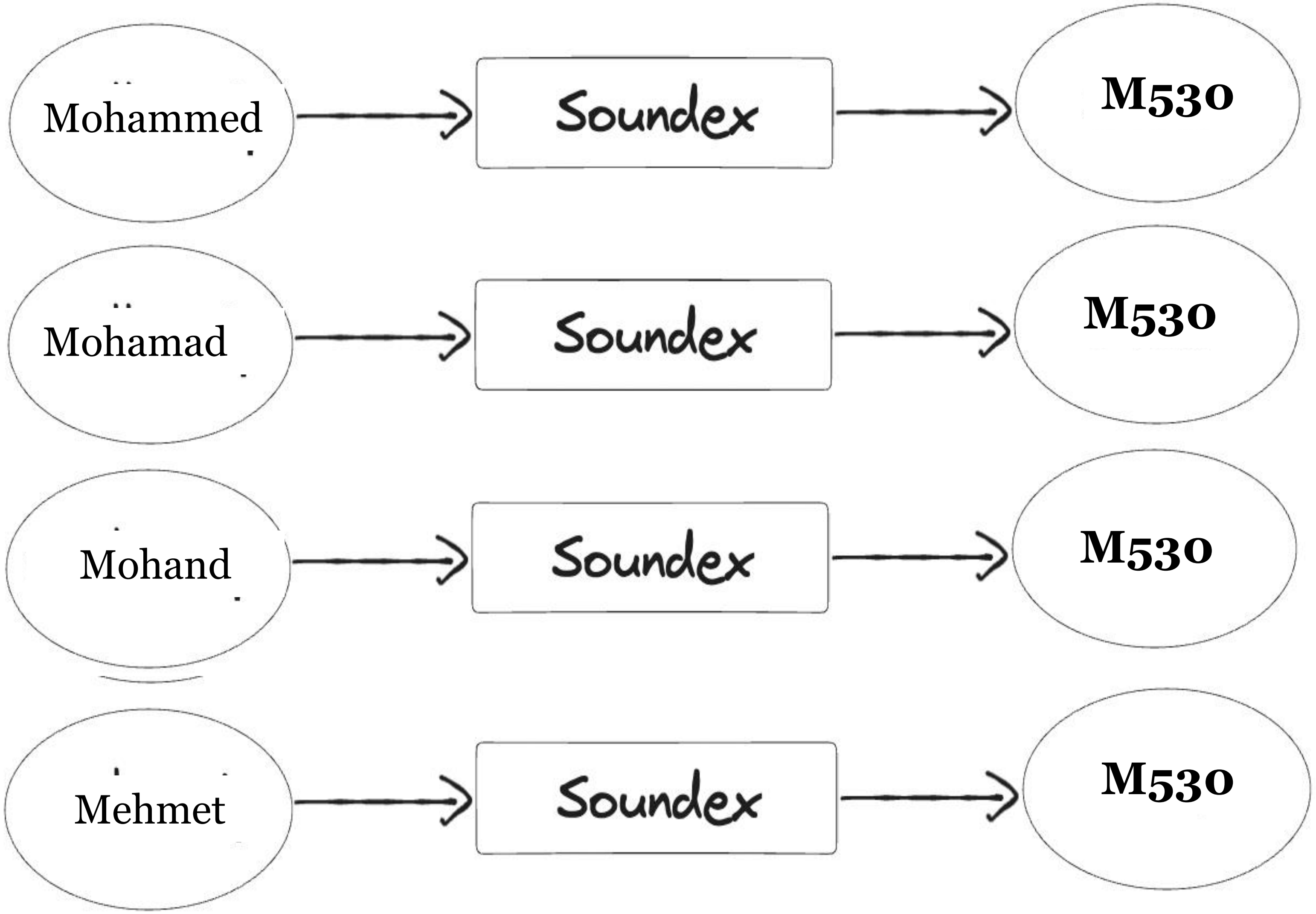
Introduction au traitement automatique des langues

Contenu de la matière :

Phonetic Matching - Soundex Algorithm



Phonetic Matching - Soundex Algorithm

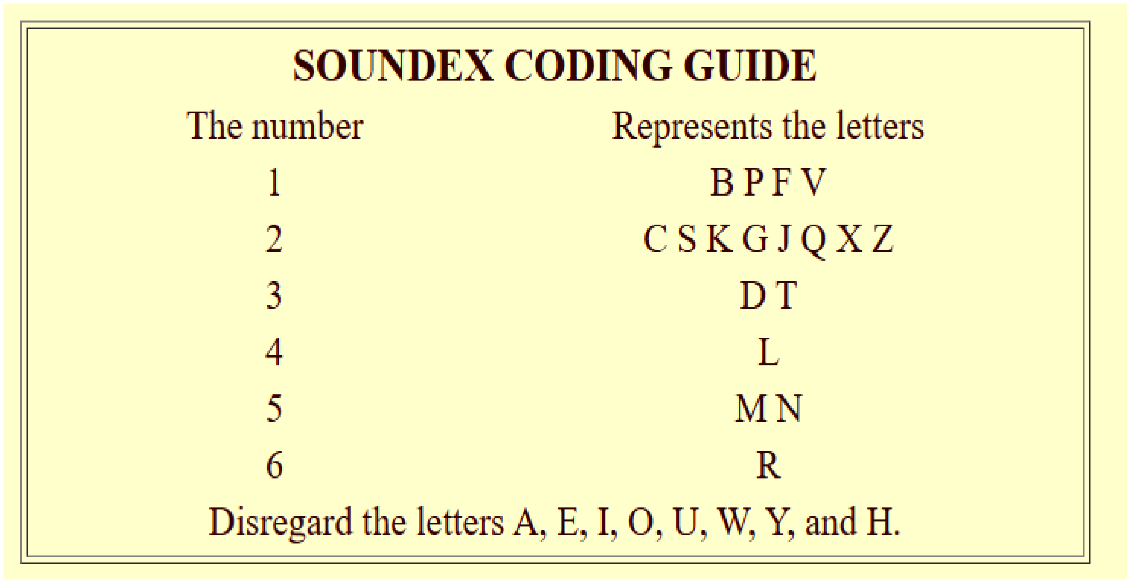


Soundex Algorithm

- **Soundex** is a phonetic algorithm for indexing names by sound, as pronounced in English. Dev. by Robert C. Russell & Margaret King Odell.

- American Soundex - The Soundex code consists of a combination of a **letter** followed by **three numbers** for each name: the letter corresponds to the first of the name, and the numbers encode the remaining consonants. Soundex code - example : M530

- Consonants with similar pronunciation have the same code:



SOUNDEX CODING GUIDE

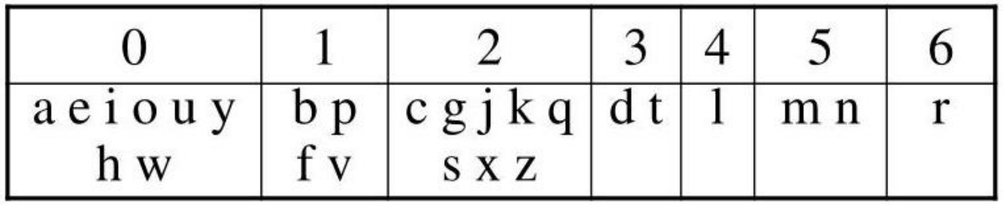
The number	Represents the letters
1	B P F V
2	C S K G J Q X Z
3	D T
4	L
5	M N
6	R

Disregard the letters A, E, I, O, U, W, Y, and H.

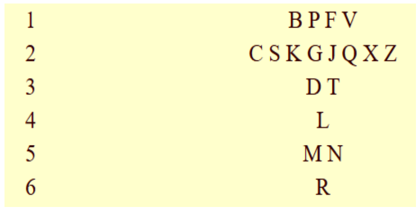
Soundex Algorithm

1. Retain the first letter of the name and drop all other occurrences of a, e, i, o, u, y, h, w.

2. Replace consonants with digits as follows (after the first letter):



0	1	2	3	4	5	6
a e i o u y h w	b p f v	c g j k q s x z	d t	l	m n	r



1	B P F V
2	C S K G J Q X Z
3	D T
4	L
5	M N
6	R

3. If two or more letters with the same number are adjacent in the original name (before step 1), only retain the first letter; also two letters with the same number separated by 'h', 'w' or 'y' are coded as a single number, whereas such letters separated by a vowel are coded twice. This rule also applies to the first letter.

4. If there are too few letters in the word to assign three numbers, append zeros until there are three numbers. If there are four or more numbers, retain only the first three.

Soundex Algorithm

http://www.searchforancestors.com/utility/soundex.html

- Example :

1. **Retain** the **first letter** of the name and **drop** all other occurrences of **a, e, i, o, u, y, h, w**.

	Mohamed
Step 1 :	
Step 2 :	
Step 4 :	
Sndx Code	

Soundex Algorithm

- Example :

1. **Retain** the **first letter** of the name and **drop** all other occurrences of **a, e, i, o, u, y, h, w**.

	Mohamed
Step 1 :	M | _ _ _ M _ D
Step 2 :	
Step 4 :	
Sndx Code	

Soundex Algorithm

- Example :

1	B P F V
2	C S K G J Q X Z
3	D T
4	L
5	M N
6	R

	Mohamed
Step 1 :	M | _ _ _ M _ D
Step 2 :	**M | 5 3**
Step 4 :	
Sndx Code	

Soundex Algorithm

- Example :
- If there are too few letters in the word to assign three numbers, append zeros until there are three numbers. If there are four or more numbers, retain only the first three.

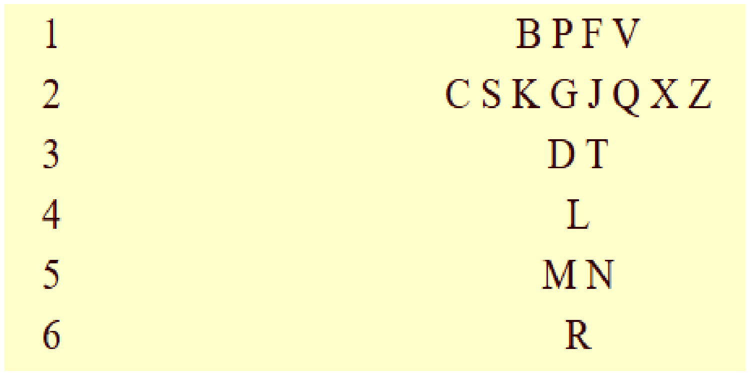
	Mohamed
Step 1 :	M | _ _ _ M _ D
Step 2 :	**M | 5 3**
Step 4 :	**M | 5 3 0**
Sndx Code	

Soundex Algorithm

- Example :
- Final Soundex Code :

	Mohamed
Step 1 :	M | _ _ _ M _ D
Step 2 :	**M | 5 3**
Step 4 :	**M | 5 3 0**
Sndx Code	M530

Soundex Algorithm



1	B P F V
2	C S K G J Q X Z
3	D T
4	L
5	M N
6	R

- Example :
- Final Soundex Code :

	Phonetic
Step 1 :	P | _ _ N _ T _ C
Step 2 :	**P | 5 3 2**
Step 4 :	**P | 5 3 2**
Sndx Code	P532

Soundex Algorithm

- Example :
- If two or more letters with the **same number** are adjacent in the original name (before step 1), only retain the first letter; also two letters with the **same number separated by 'h', 'w' or 'y'** are coded as a single number.

	Mohamed	**Mohammed**
Step 1 :	M | _ _ _ M _ D	M | _ _ _ M M _ D
Step 2 :	**M | 5 3**	**M | 5 ~~5~~ 3**
Step 4 :	**M | 5 3 0**	**M | 5 3 0**
Sndx Code	M530	M530

Soundex Algorithm

1	B P F V
2	C S K G J Q X Z
3	D T
4	L
5	M N
6	R

- Example :

- whereas such letters (two or more letters with the same number) separated by a vowel are coded twice. This rule also applies to the **first letter**.

	Pfizer
Step 1 :	P | ~~F~~ _ Z _ R
Step 2 :	**P | ~~1~~ 2 6**
Step 4 :	**P | 260**
Sndx Code	P260

Soundex Algorithm

1	B P F V
2	C S K G J Q X Z
3	D T
4	L
5	M N
6	R

- Example :

- whereas such letters (two or more letters with the same number) separated by a **vowel** are coded twice. This rule also applies to the **first letter**.

	Pfizer	**Pifizer**
Step 1 :	P | ~~F~~ _ Z _ R	P | _ F _ Z _ R
Step 2 :	**P | ~~1~~ 2 6**	**P | 1 2 6**
Step 4 :	**P | 260**	**P | 126**
Sndx Code	P260	P126

Niveaux de Traitement - Analyse Lexicale

- **Série TP 3 – Analyse Lexicale avec NLTK**

Partie 1 - Découverte

Partie 2 - Exercices

Références

Livre - Speech and Language Processing, de Dan Jurafsky.

Cours - *François Yvon* – Une petite introduction au Traitement Automatique des Langues Naturelles, https://perso.limsi.fr/anne/coursM2R/intro.pdf

Codage des caractères : https://www.fil.univ-lille1.fr/~wegrzyno/portail/Info/Doc/HTML/seq7_codage_caracteres.html

Text Processing with Unicode - http://nltk.sourceforge.net/doc/en/app-unicode.html

Data Cleaning Challenge: Character Encodings - https://www.kaggle.com/rtatman/data-cleaning-challenge-character-encodings

Tokenization for Natural Language Processing - https://towardsdatascience.com/tokenization-for-natural-language-processing-a179a891bad4?gi=6b15f97fe07d

Cours - ARIES Abdelkrime - Le traitement automatique du langage naturel. https://github.com/projeduc/ESI_2CS_TALN

Niveaux de Traitement - Analyse Lexicale

- **Série TP 3 – Analyse Lexicale avec NLTK**

Partie 1 - Découverte

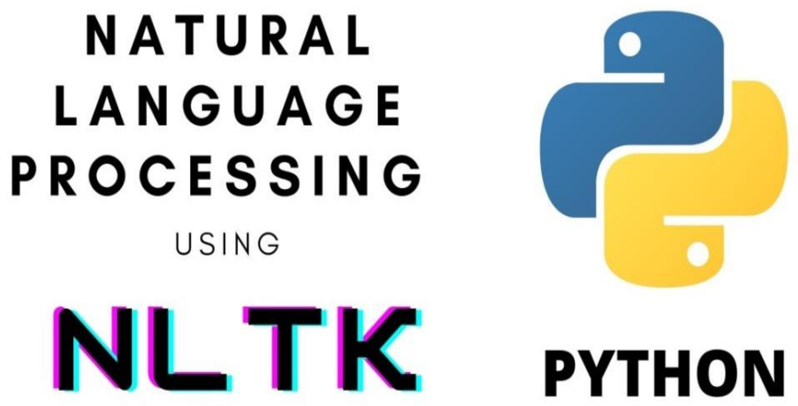
Partie 2 - Exercices

Série TP 3 – Analyse Lexicale

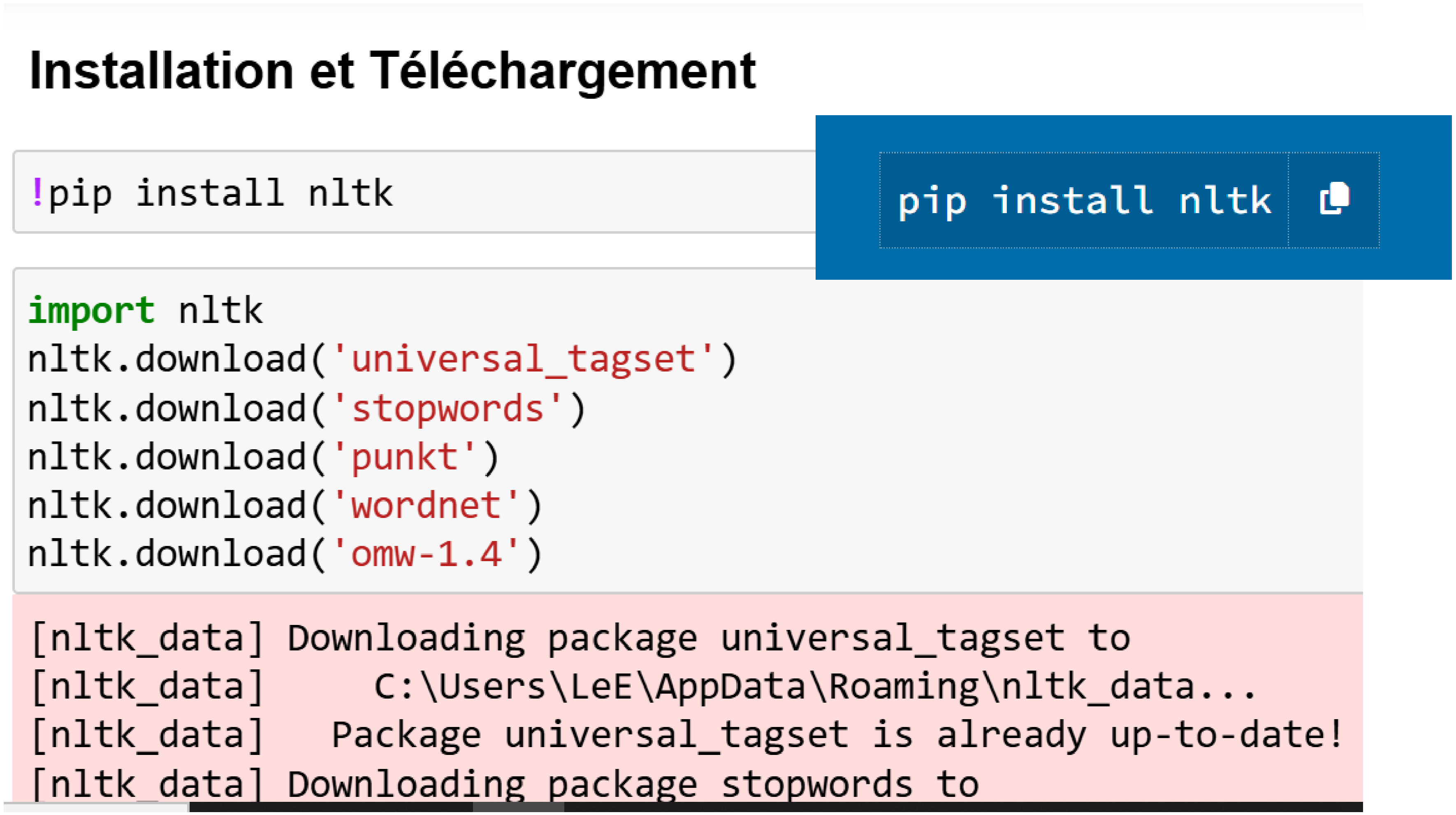
- Natural Language ToolKit (**NLTK**) est une bibliothèque Python permettant de créer des programmes fonctionnant avec le langage naturel.

- Il fournit une interface conviviale aux ensembles de données contenant plus de 50 corpus et ressources lexicales telles que WordNet. La bibliothèque peut effectuer différentes opérations : la tokenization, le stemming, la lemmatisation, la classification, le parsing, le pos tagging.

- NLTK peut être utilisé par les étudiants, les chercheurs et les industriels. C'est une bibliothèque Open Source et gratuite.



Série TP 3 – Analyse Lexicale



Installation et Téléchargement

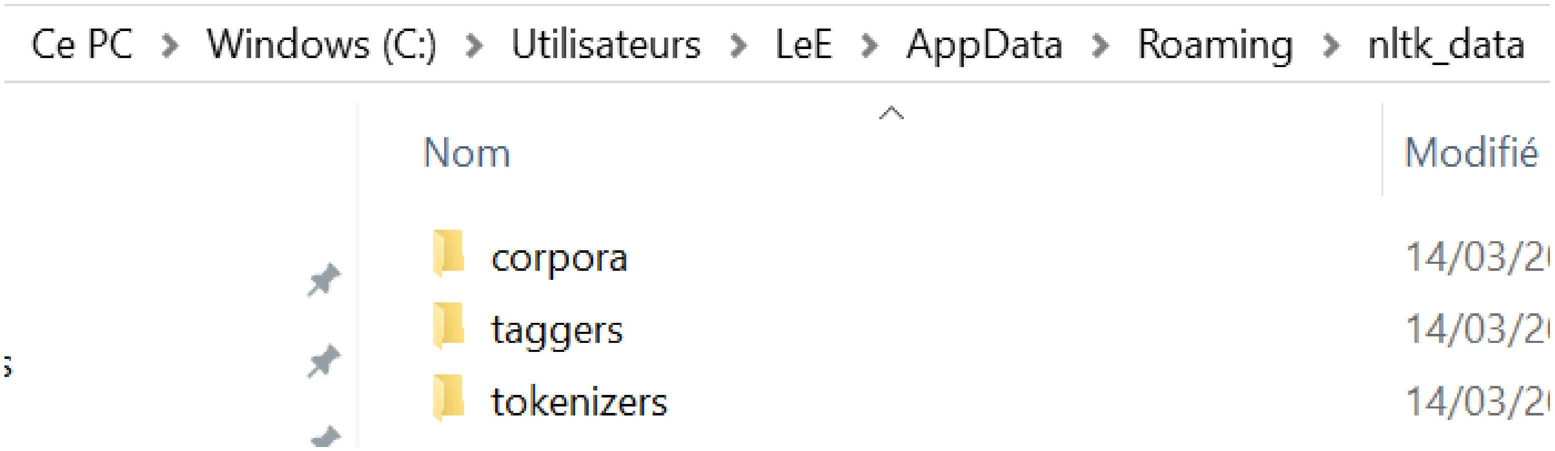
```
!pip install nltk
```

```
pip install nltk
```

```
import nltk
nltk.download('universal_tagset')
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

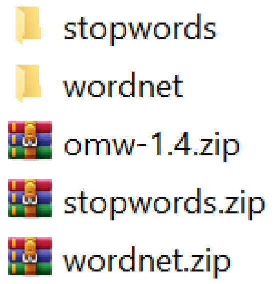
```
[nltk_data] Downloading package universal_tagset to
[nltk_data]     C:\Users\LeE\AppData\Roaming\nltk_data...
[nltk_data]   Package universal_tagset is already up-to-date!
[nltk_data] Downloading package stopwords to
```

C:\Users\LeE\AppData\Roaming\nltk_data

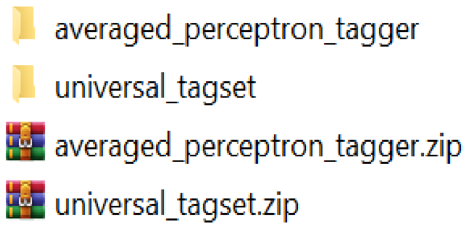


Ce PC › Windows (C:) › Utilisateurs › LeE › AppData › Roaming › nltk_data

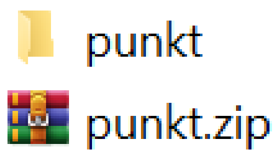
Nom	Modifié
corpora	14/03/2
taggers	14/03/2
tokenizers	14/03/2



stopwords
wordnet
omw-1.4.zip
stopwords.zip
wordnet.zip



averaged_perceptron_tagger
universal_tagset
averaged_perceptron_tagger.zip
universal_tagset.zip



punkt
punkt.zip

Série TP 3 – Analyse Lexicale

1. Tokenization with NLTK
2. Stopwords with NLTK
3. POS Tagging with NLTK
4. Stemming with NLTK
5. Lemmatization with NTLK
6. N-grams with NLTK
7. WordNet with NLTK

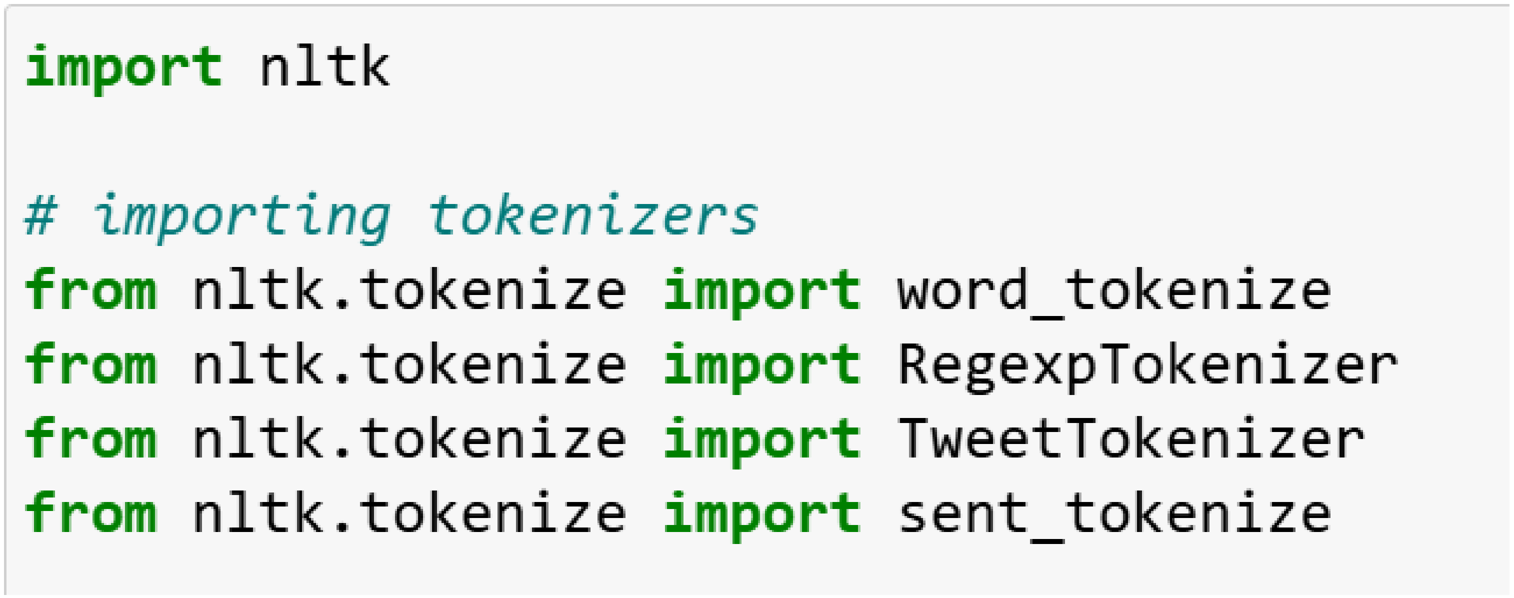
NATURAL LANGUAGE PROCESSING USING NLTK

PYTHON

Série TP 3 – Analyse Lexicale

- **Tokenization with NLTK**

https://www.nltk.org/api/nltk.tokenize.html

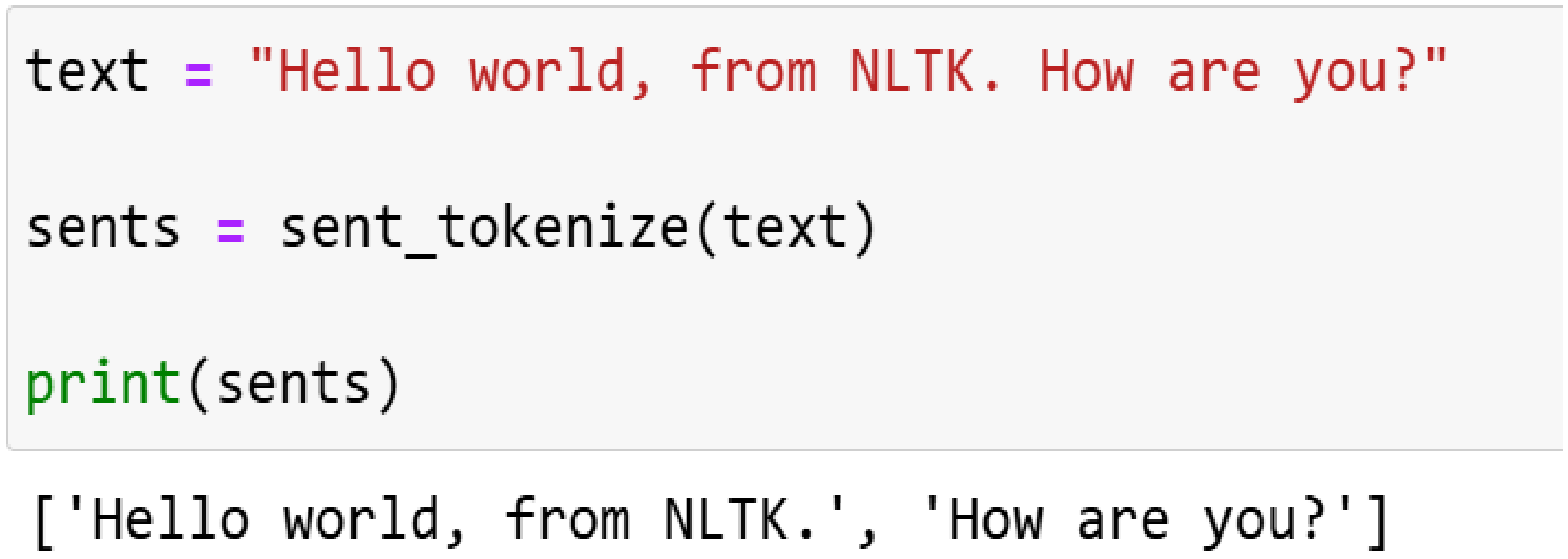


```python
import nltk

# importing tokenizers
from nltk.tokenize import word_tokenize
from nltk.tokenize import RegexpTokenizer
from nltk.tokenize import TweetTokenizer
from nltk.tokenize import sent_tokenize
```

Série TP 3 – Analyse Lexicale

- **Tokenization with NLTK**

```
from nltk.tokenize import sent_tokenize
```



```
text = "Hello world, from NLTK. How are you?"

sents = sent_tokenize(text)

print(sents)
```

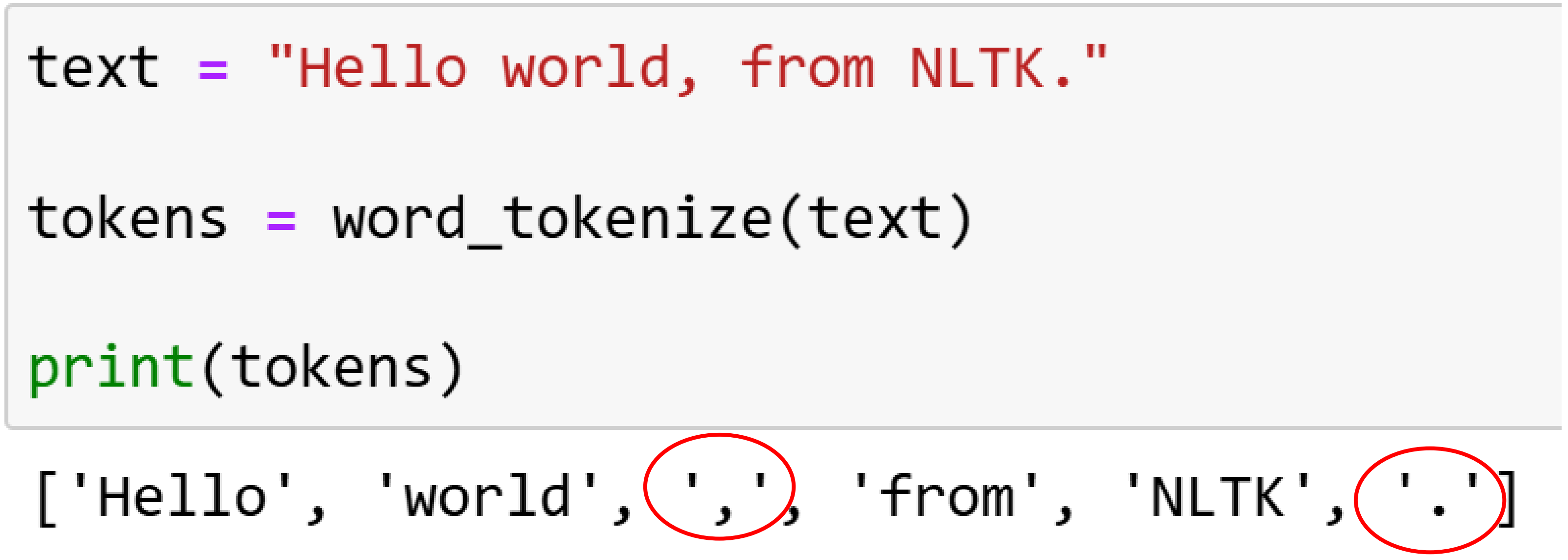
```
['Hello world, from NLTK.', 'How are you?']
```

Série TP 3 – Analyse Lexicale

- **Tokenization with NLTK**

With punkts in tokens list

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



```
text = "Hello world, from NLTK."

tokens = word_tokenize(text)

print(tokens)
```

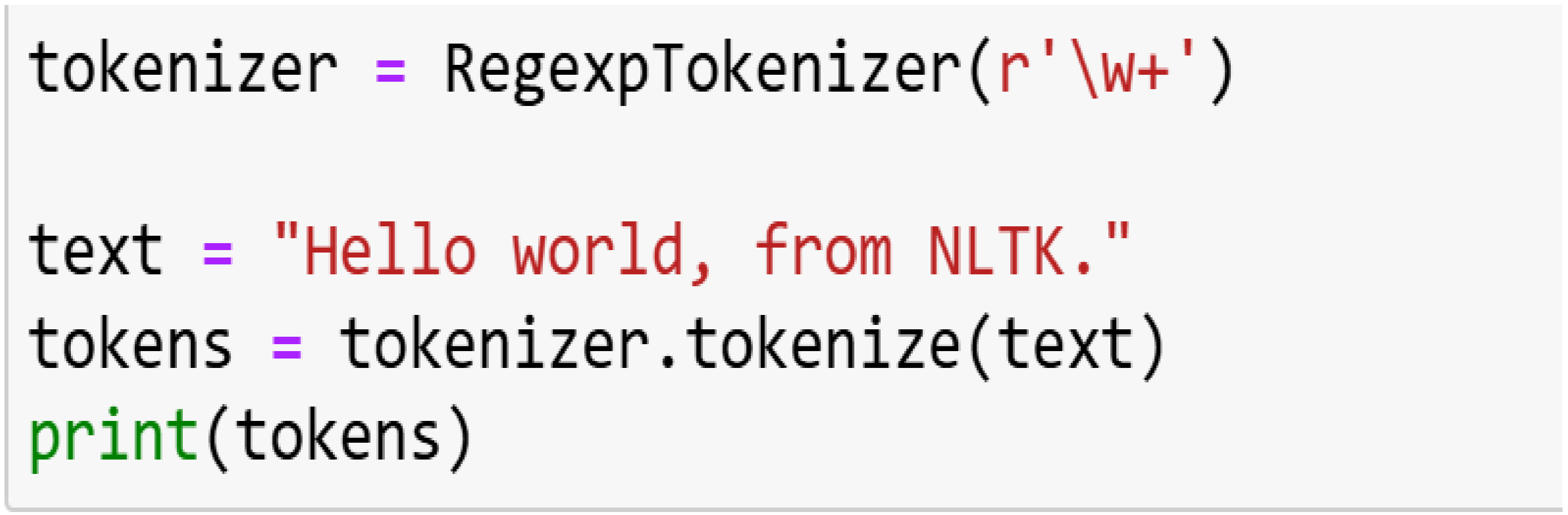
```
['Hello', 'world', ',', 'from', 'NLTK', '.']
```

Série TP 3 – Analyse Lexicale

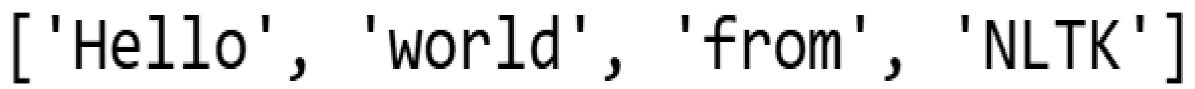
- **Tokenization with NLTK**

Without punkts in tokens list

```
from nltk.tokenize import RegexpTokenizer
```



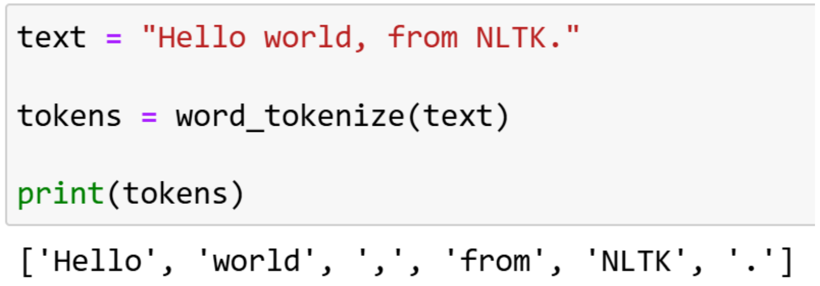
```
tokenizer = RegexpTokenizer(r'\w+')

text = "Hello world, from NLTK."
tokens = tokenizer.tokenize(text)
print(tokens)
```



```
['Hello', 'world', 'from', 'NLTK']
```

Série TP 3 – Analyse Lexicale

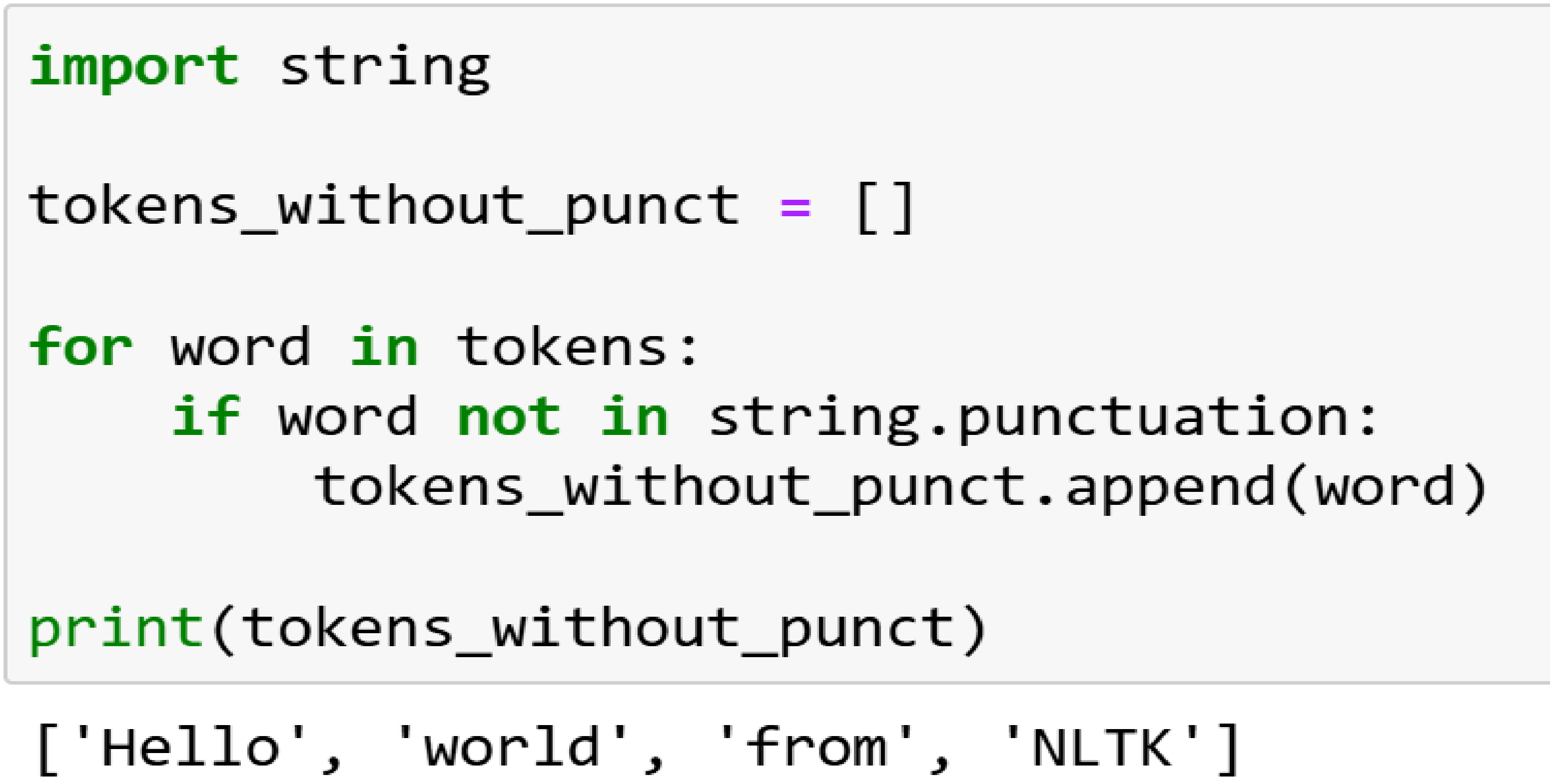
- **Tokenization with NLTK**



```
text = "Hello world, from NLTK."

tokens = word_tokenize(text)

print(tokens)
```

```
['Hello', 'world', ',', 'from', 'NLTK', '.']
```

string.punctuation



```
import string

tokens_without_punct = []

for word in tokens:
    if word not in string.punctuation:
        tokens_without_punct.append(word)

print(tokens_without_punct)
```

```
['Hello', 'world', 'from', 'NLTK']
```

Série TP 3 – Analyse Lexicale

- **Tokenization with NLTK**

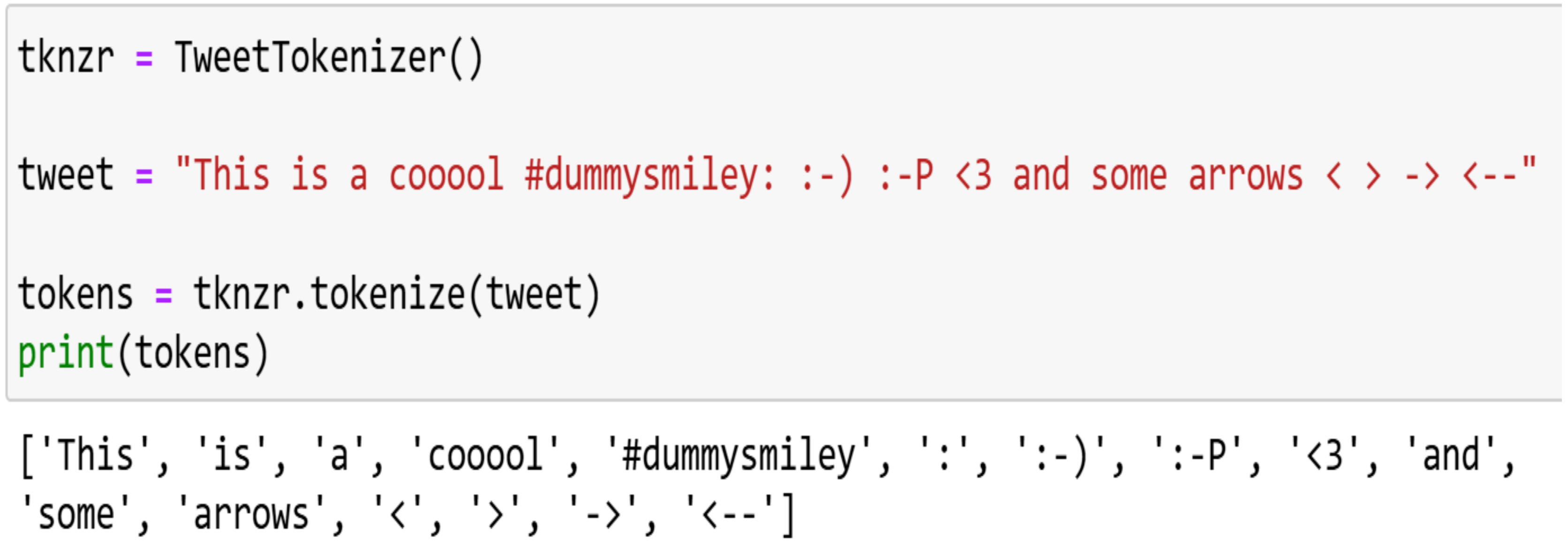
- **string.punctuation**

```
['!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/',
':', ';', '<', '=', '>', '?', '@', '[', '\\', ']', '^', '_', '`', '{', '|',
'}', '~']
```

Série TP 3 – Analyse Lexicale

- **Tokenization with NLTK**

```
from  nltk.tokenize  import  TweetTokenizer
```



```
tknzr = TweetTokenizer()

tweet = "This is a cooool #dummysmiley: :-) :-P <3 and some arrows < > -> <--"

tokens = tknzr.tokenize(tweet)
print(tokens)
```

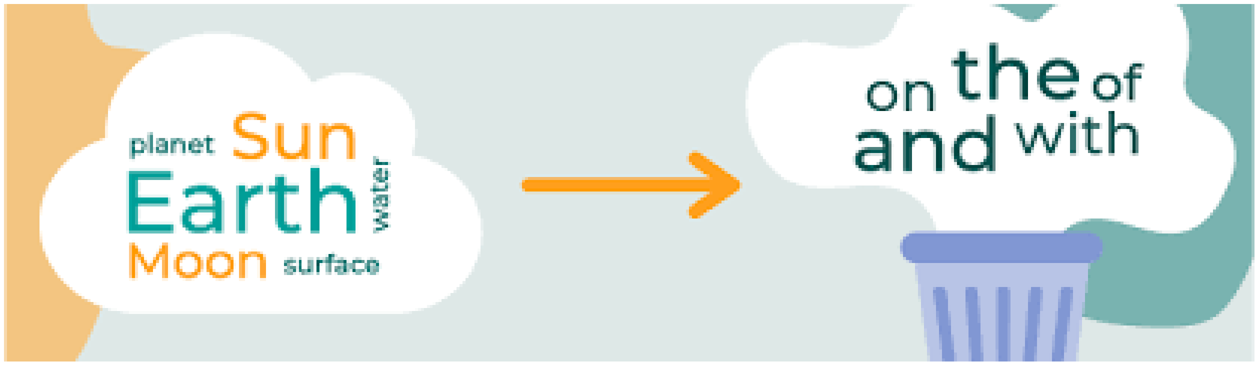
```
['This', 'is', 'a', 'cooool', '#dummysmiley', ':', ':-)', ':-P', '<3', 'and',
'some', 'arrows', '<', '>', '->', '<--']
```

Série TP 3 – Analyse Lexicale

- **Stopwords with NLTK**

```
print(stopwords.fileids())
```

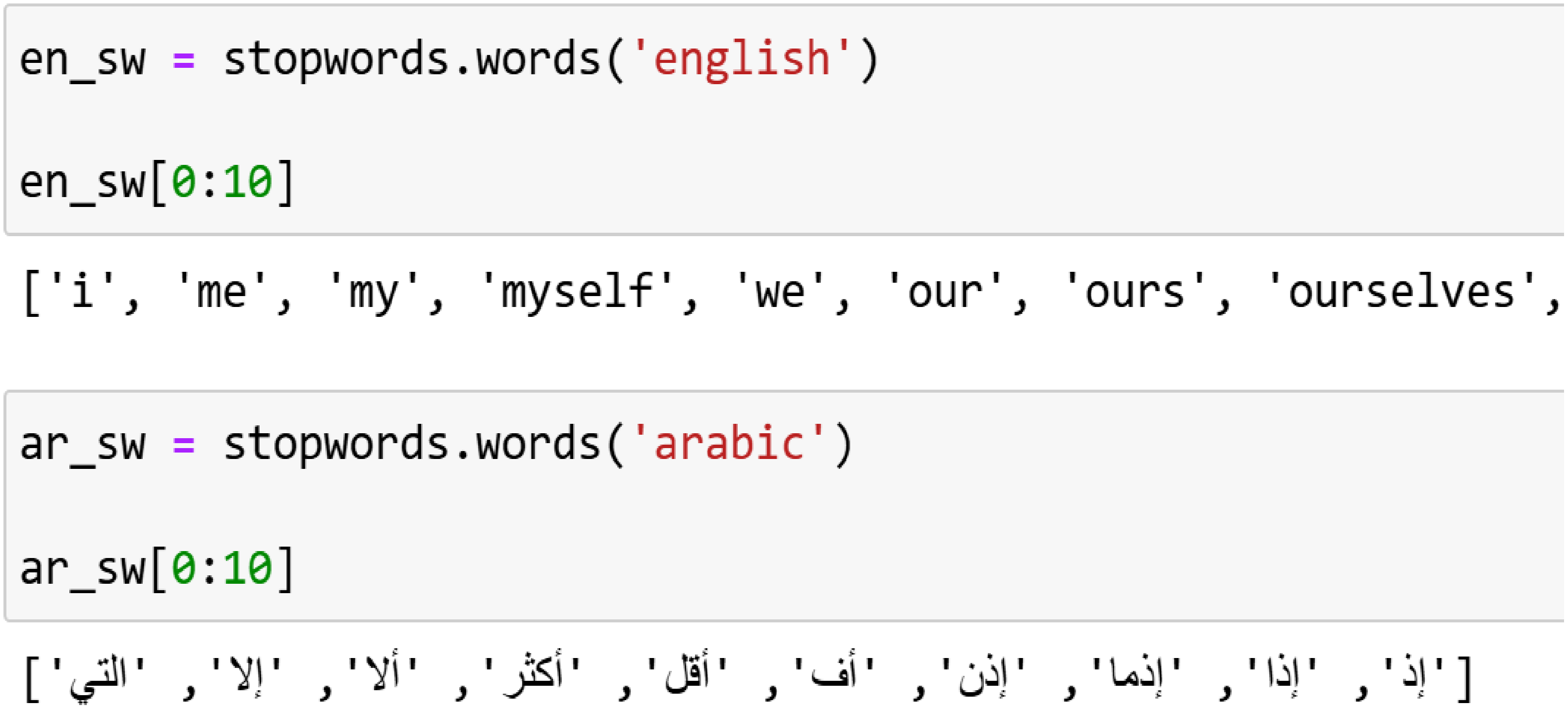
```
['arabic', 'azerbaijani', 'basque', 'bengali', 'catalan', 'chinese', 'danish',
'dutch', 'english', 'finnish', 'french', 'german', 'greek', 'hebrew', 'hinglis
h', 'hungarian', 'indonesian', 'italian', 'kazakh', 'nepali', 'norwegian', 'por
tuguese', 'romanian', 'russian', 'slovene', 'spanish', 'swedish', 'tajik', 'tur
kish']
```



Série TP 3 – Analyse Lexicale

- **Stopwords with NLTK**

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



```
en_sw = stopwords.words('english')

en_sw[0:10]
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves',
```

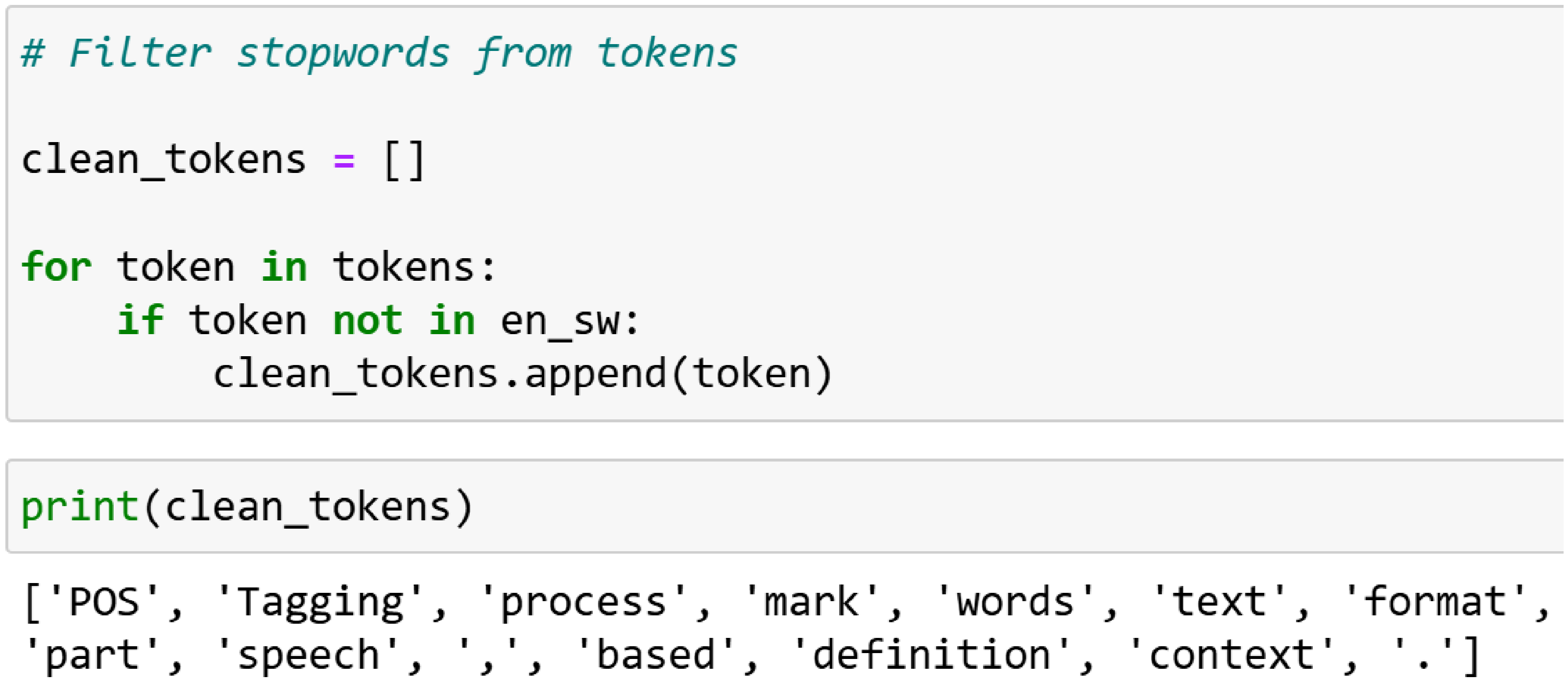
```
ar_sw = stopwords.words('arabic')

ar_sw[0:10]
```

```
['إذ', 'إذا', 'إذما', 'إذن', 'أف', 'أقل', 'أكثر', 'ألا', 'إلا', 'التي'
```

Série TP 3 – Analyse Lexicale

- **Stopwords with NLTK**

```python
from nltk.corpus import stopwords
```



```python
# Filter stopwords from tokens

clean_tokens = []

for token in tokens:
    if token not in en_sw:
        clean_tokens.append(token)
```

```python
print(clean_tokens)
```

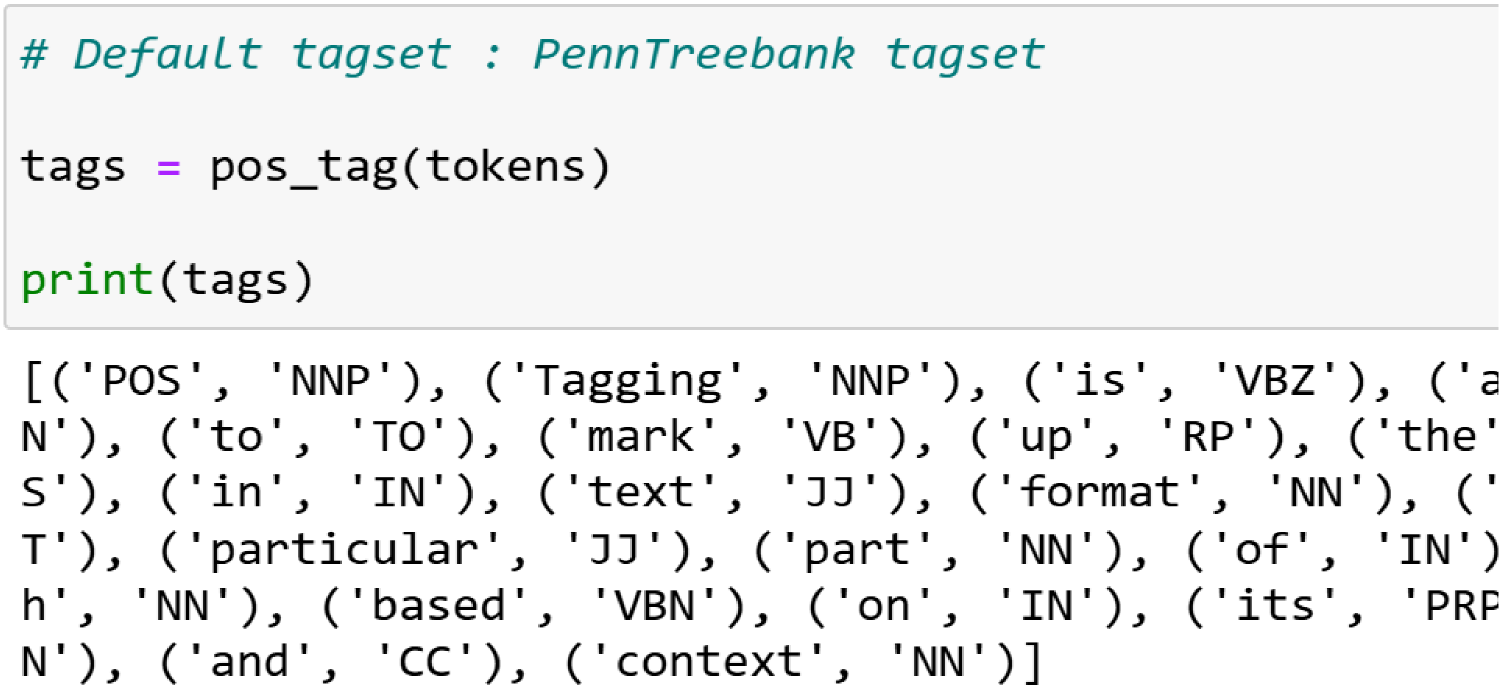
```
['POS', 'Tagging', 'process', 'mark', 'words', 'text', 'format',
'part', 'speech', ',', 'based', 'definition', 'context', '.']
```

Série TP 3 – Analyse Lexicale

- **POS Tagging with NLTK**

```
nltk.tag.pos_tag(tokens_list,
        tagset = None,
        lang = 'eng') :

list(tuple(str, str))
```

```
from nltk.tag import pos_tag
```



```
# Default tagset : PennTreebank tagset

tags = pos_tag(tokens)

print(tags)
```

```
[('POS', 'NNP'), ('Tagging', 'NNP'), ('is', 'VBZ'), ('a
N'), ('to', 'TO'), ('mark', 'VB'), ('up', 'RP'), ('the'
S'), ('in', 'IN'), ('text', 'JJ'), ('format', 'NN'), ('
T'), ('particular', 'JJ'), ('part', 'NN'), ('of', 'IN')
h', 'NN'), ('based', 'VBN'), ('on', 'IN'), ('its', 'PRP
N'), ('and', 'CC'), ('context', 'NN')]
```

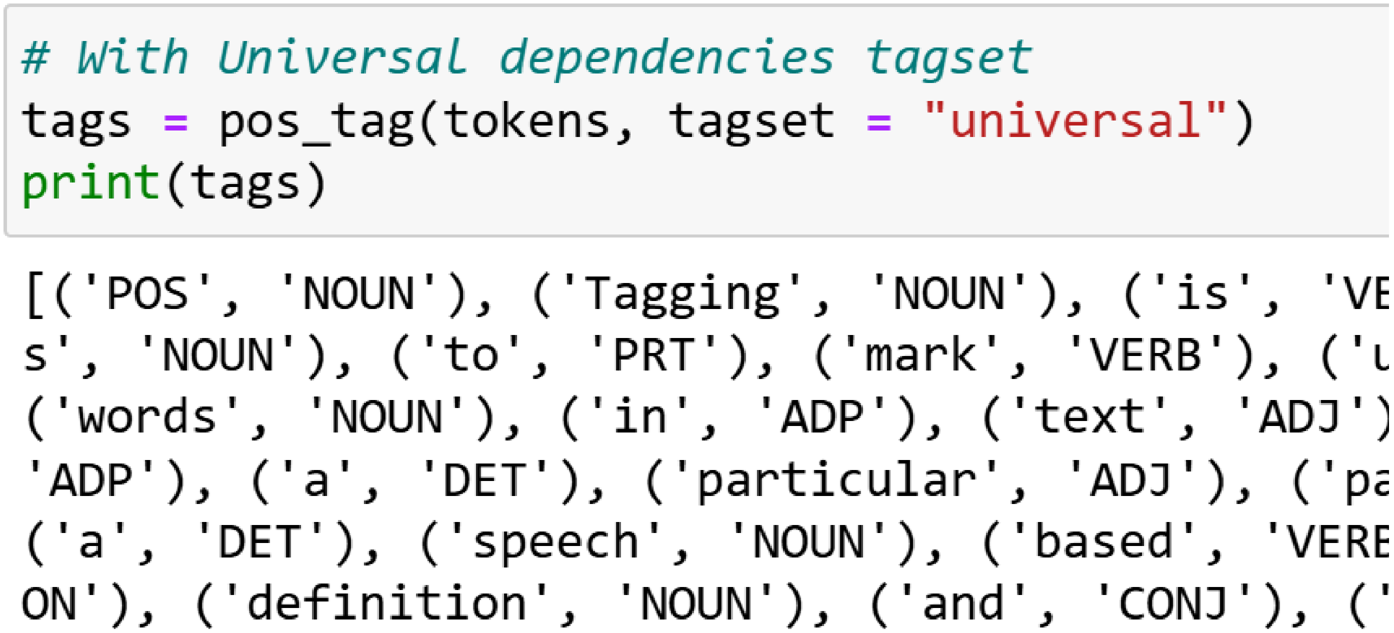
Série TP 3 – Analyse Lexicale

- **POS Tagging with NLTK**

nltk.tag.**pos_tag**(tokens_list,
        tagset = None,
        lang = 'eng') :

list(tuple(str, str))

```
from nltk.tag import pos_tag
```



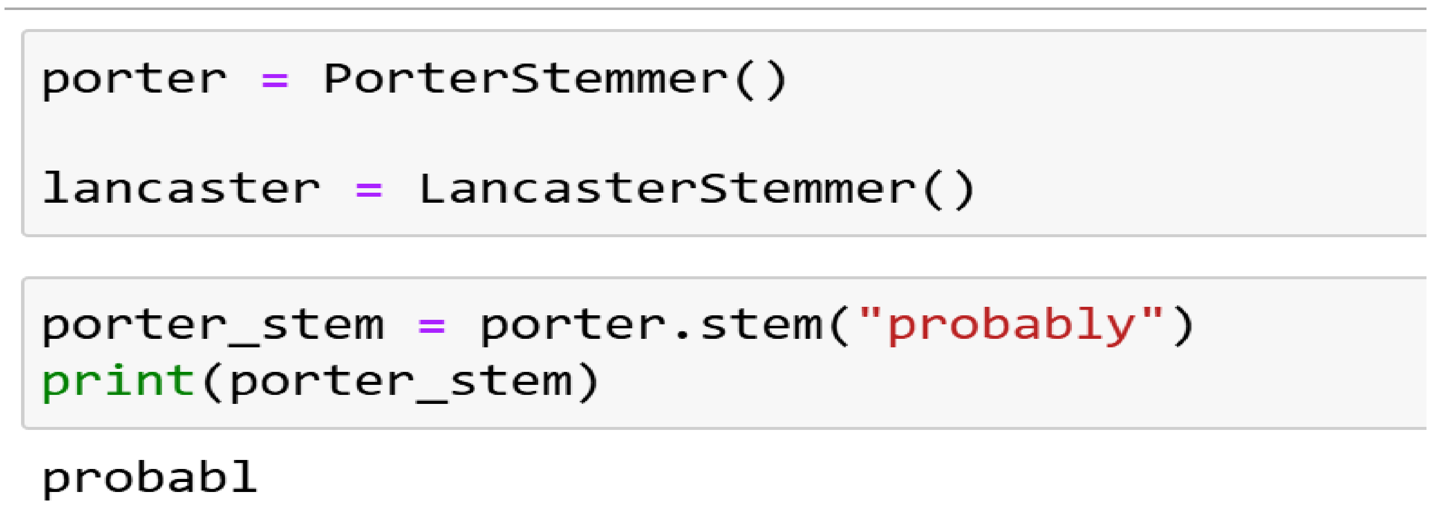
```
# With Universal dependencies tagset
tags = pos_tag(tokens, tagset = "universal")
print(tags)
```

```
[('POS', 'NOUN'), ('Tagging', 'NOUN'), ('is', 'VE
s', 'NOUN'), ('to', 'PRT'), ('mark', 'VERB'), ('u
('words', 'NOUN'), ('in', 'ADP'), ('text', 'ADJ')
'ADP'), ('a', 'DET'), ('particular', 'ADJ'), ('pa
('a', 'DET'), ('speech', 'NOUN'), ('based', 'VERE
ON'), ('definition', 'NOUN'), ('and', 'CONJ'), ('
```

Série TP 3 – Analyse Lexicale

- **Stemming with NLTK**

```
from nltk.stem import PorterStemmer, LancasterStemmer
```



```
porter = PorterStemmer()

lancaster = LancasterStemmer()
```

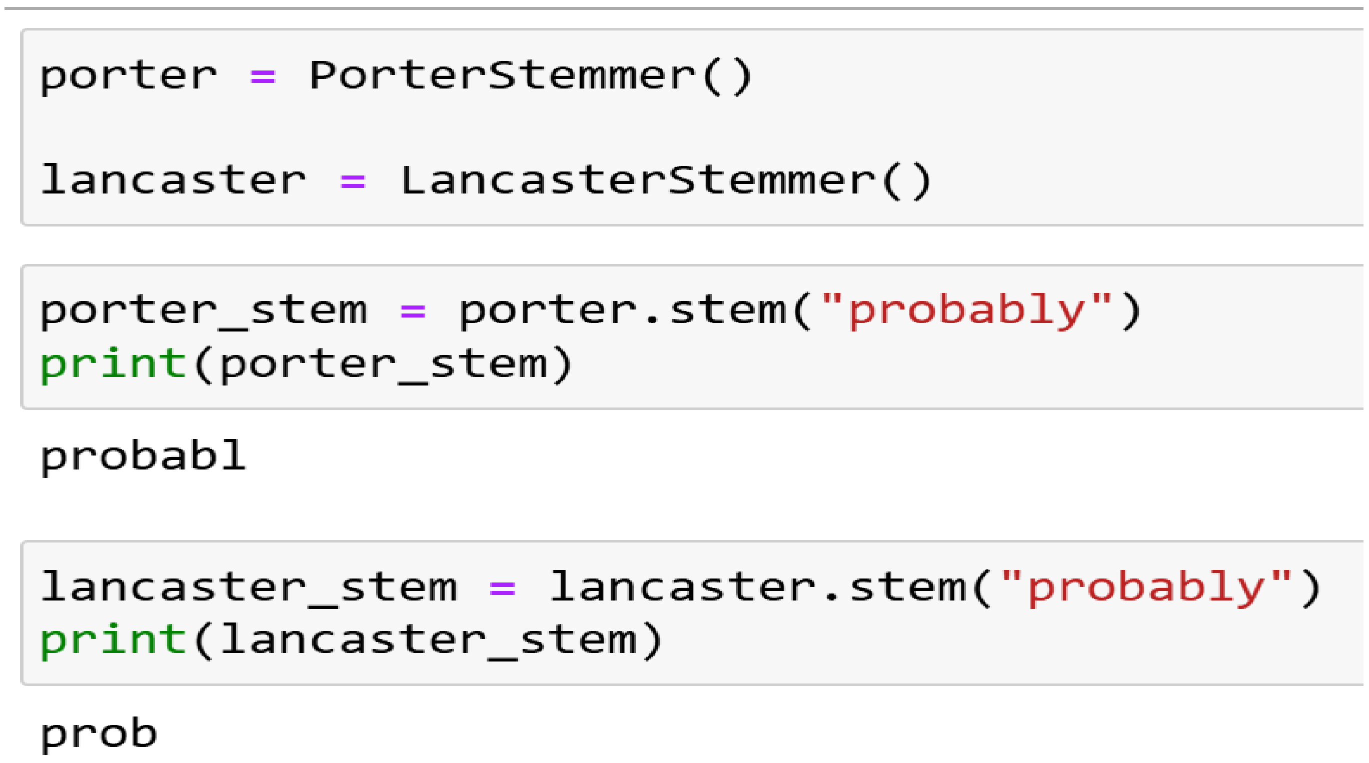
```
porter_stem = porter.stem("probably")
print(porter_stem)
```

```
probabl
```

Série TP 3 – Analyse Lexicale

- **Stemming with NLTK**

```
from nltk.stem import PorterStemmer, LancasterStemmer
```



```
porter = PorterStemmer()

lancaster = LancasterStemmer()
```

```
porter_stem = porter.stem("probably")
print(porter_stem)
```

```
probabl
```

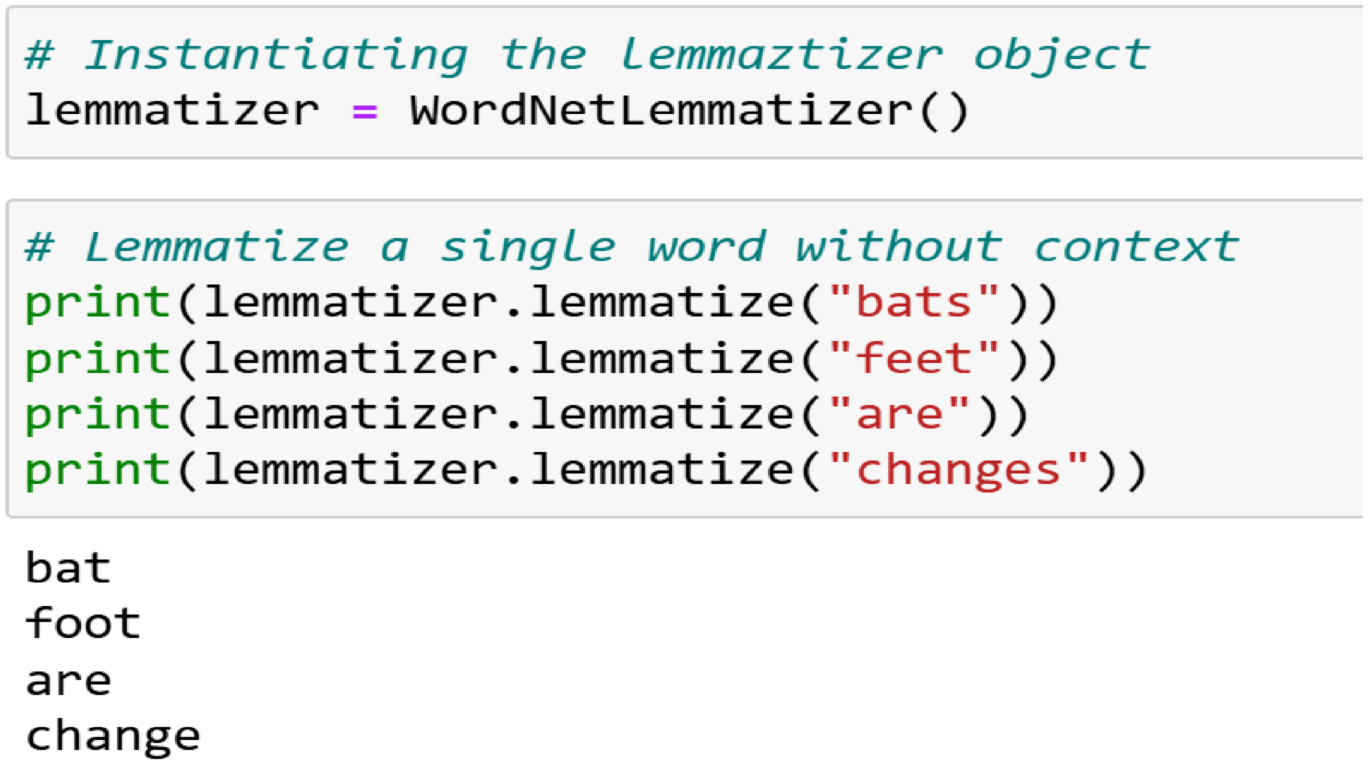
```
lancaster_stem = lancaster.stem("probably")
print(lancaster_stem)
```

```
prob
```

Série TP 3 – Analyse Lexicale

- **Lemmatization with NLTK**

```
from nltk.stem import WordNetLemmatizer
```



```
# Instantiating the lemmaztizer object
lemmatizer = WordNetLemmatizer()
```

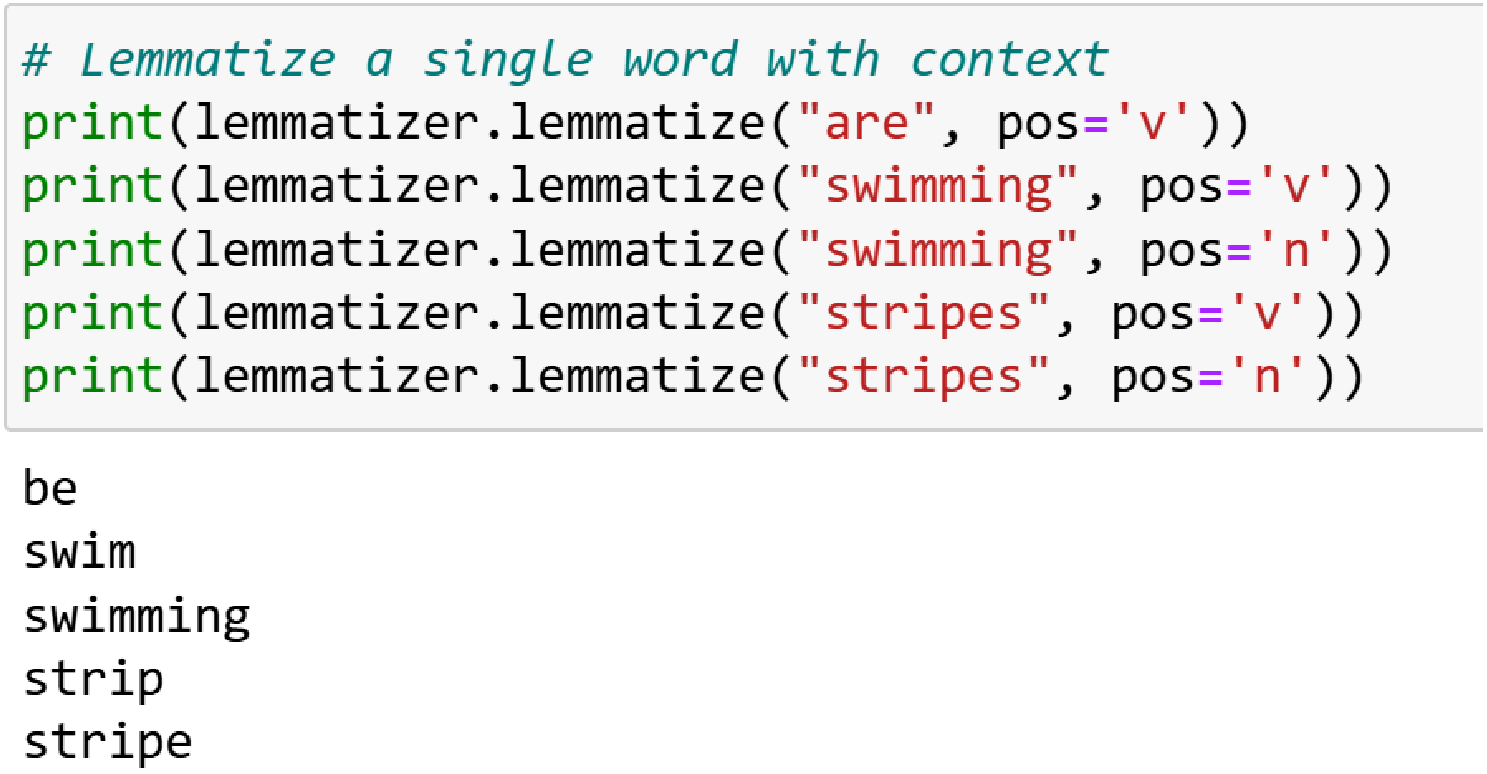
```
# Lemmatize a single word without context
print(lemmatizer.lemmatize("bats"))
print(lemmatizer.lemmatize("feet"))
print(lemmatizer.lemmatize("are"))
print(lemmatizer.lemmatize("changes"))
```

```
bat
foot
are
change
```

Série TP 3 – Analyse Lexicale

- **Lemmatization with NLTK**

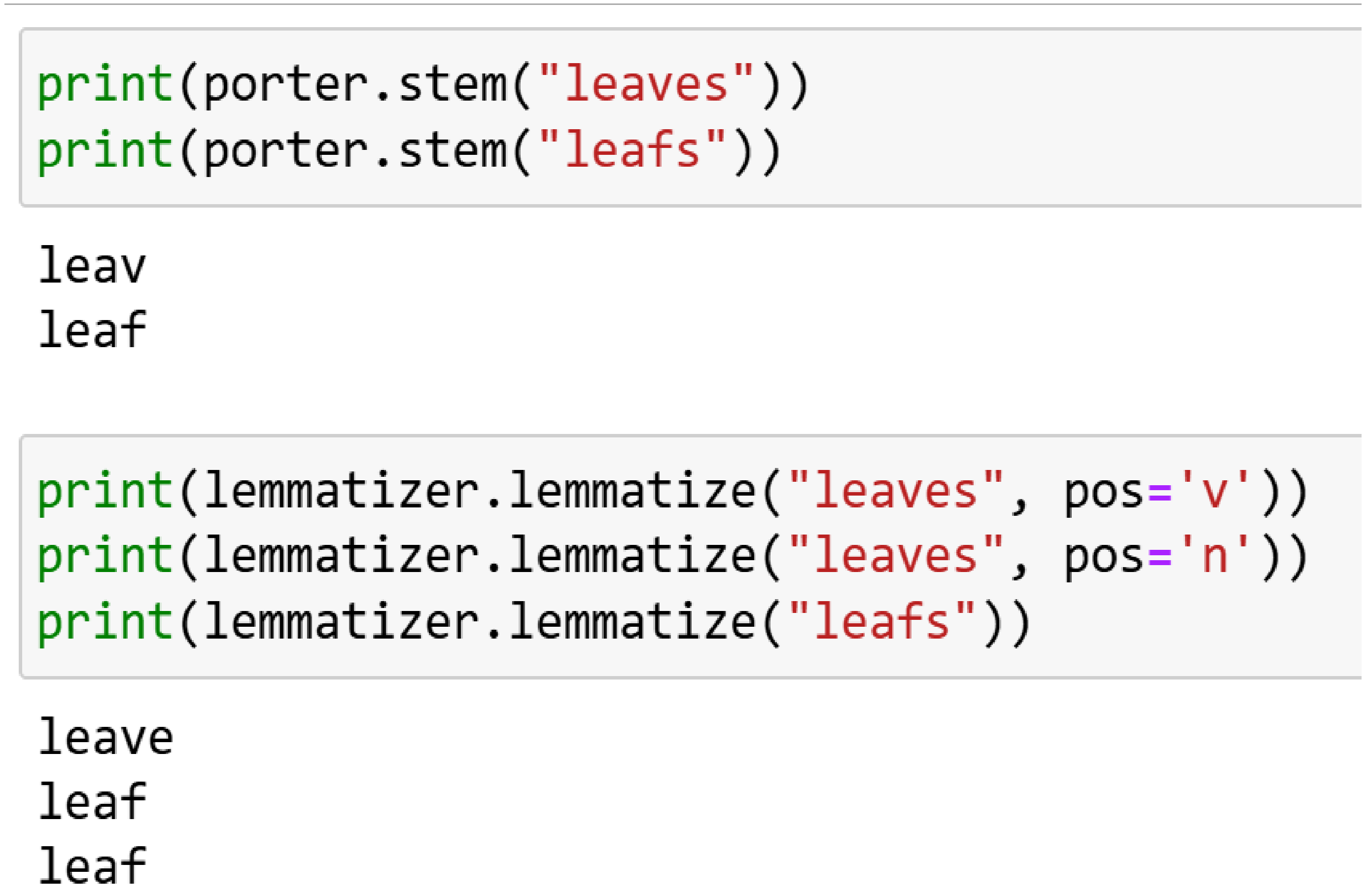
```
from  nltk.stem  import  WordNetLemmatizer
```



```
# Lemmatize a single word with context
print(lemmatizer.lemmatize("are", pos='v'))
print(lemmatizer.lemmatize("swimming", pos='v'))
print(lemmatizer.lemmatize("swimming", pos='n'))
print(lemmatizer.lemmatize("stripes", pos='v'))
print(lemmatizer.lemmatize("stripes", pos='n'))
```

```
be
swim
swimming
strip
stripe
```

- **Stemming Vs Lemmatization with NLTK**



```python
print(porter.stem("leaves"))
print(porter.stem("leafs"))
```

```
leav
leaf
```

```python
print(lemmatizer.lemmatize("leaves", pos='v'))
print(lemmatizer.lemmatize("leaves", pos='n'))
print(lemmatizer.lemmatize("leafs"))
```

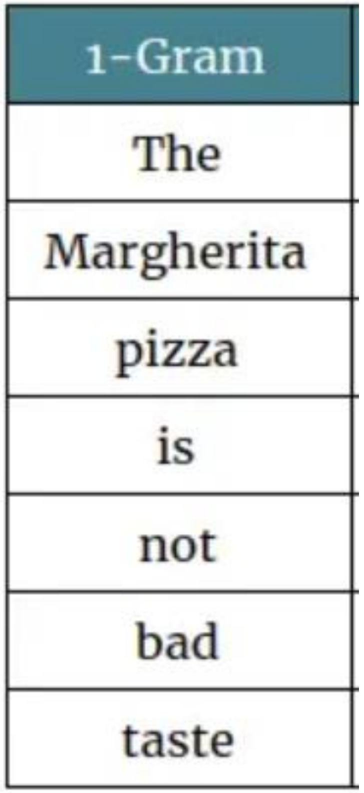
```
leave
leaf
leaf
```

Série TP 3 – Analyse Lexicale

Uni-Gram

- **N-grams with NLTK**

text = The Margherita pizza is not bad taste



1-Gram
The
Margherita
pizza
is
not
bad
taste

Série TP 3 – Analyse Lexicale

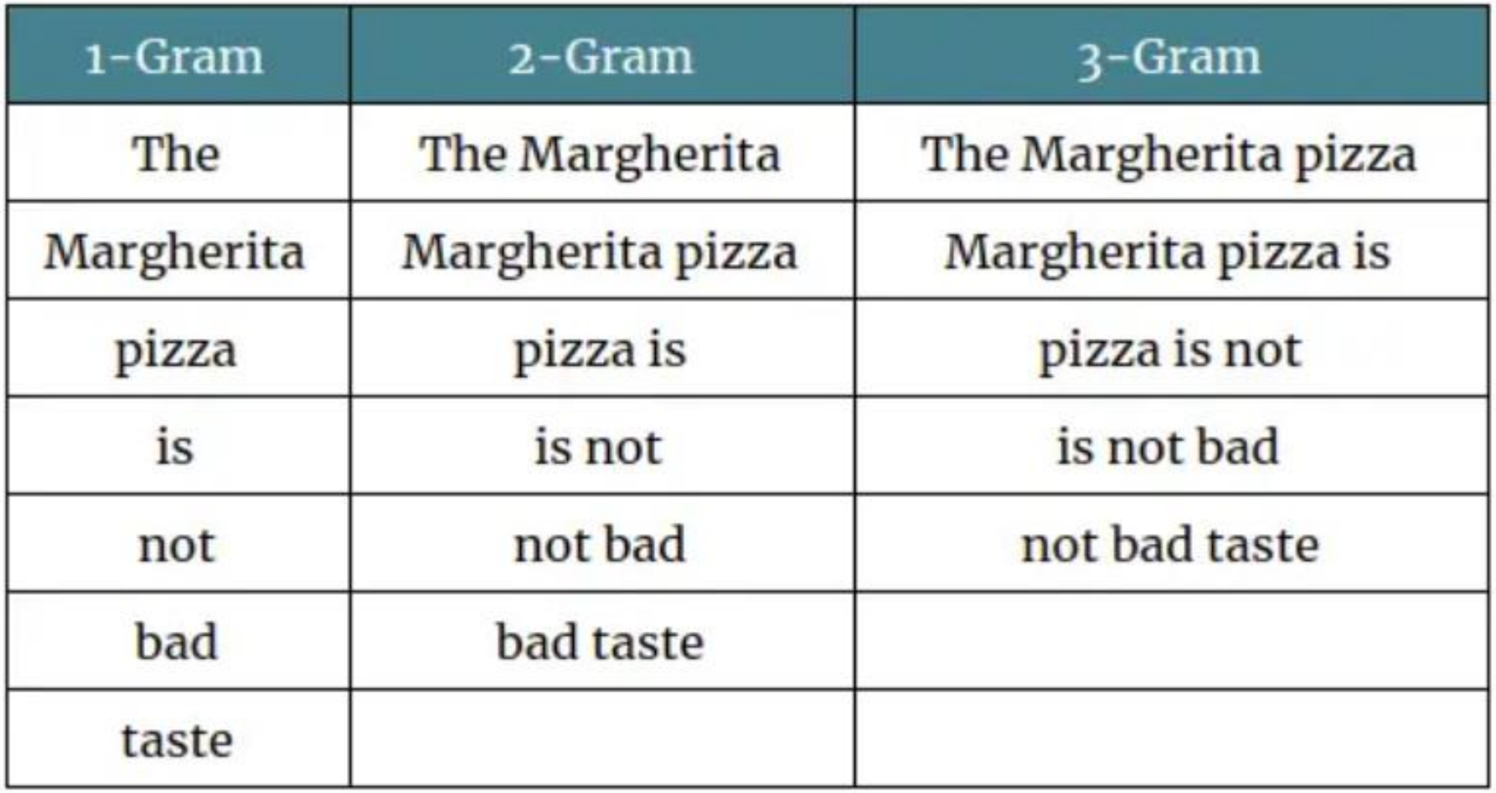
- **N-grams with NLTK**

Uni-Gram

Bi-Gram

Tri-Gram

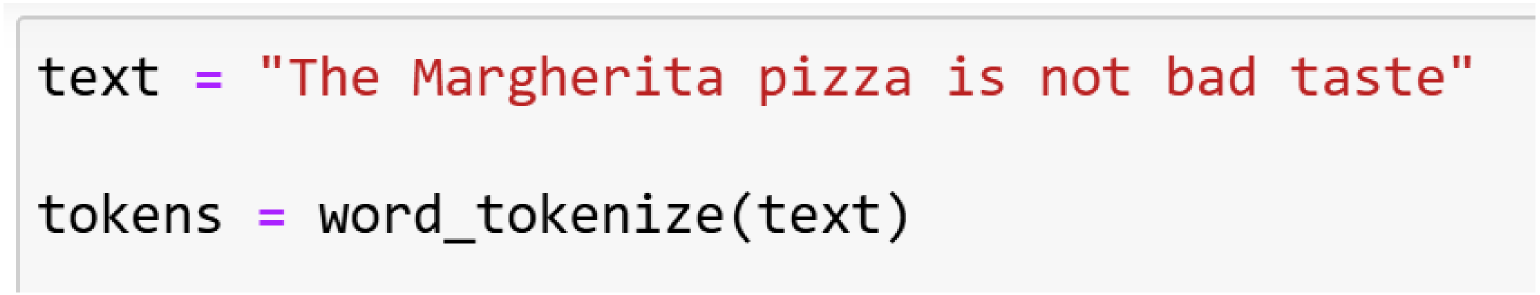
text = The Margherita pizza is not bad taste



1-Gram	2-Gram	3-Gram
The	The Margherita	The Margherita pizza
Margherita	Margherita pizza	Margherita pizza is
pizza	pizza is	pizza is not
is	is not	is not bad
not	not bad	not bad taste
bad	bad taste	
taste		

Série TP 3 – Analyse Lexicale

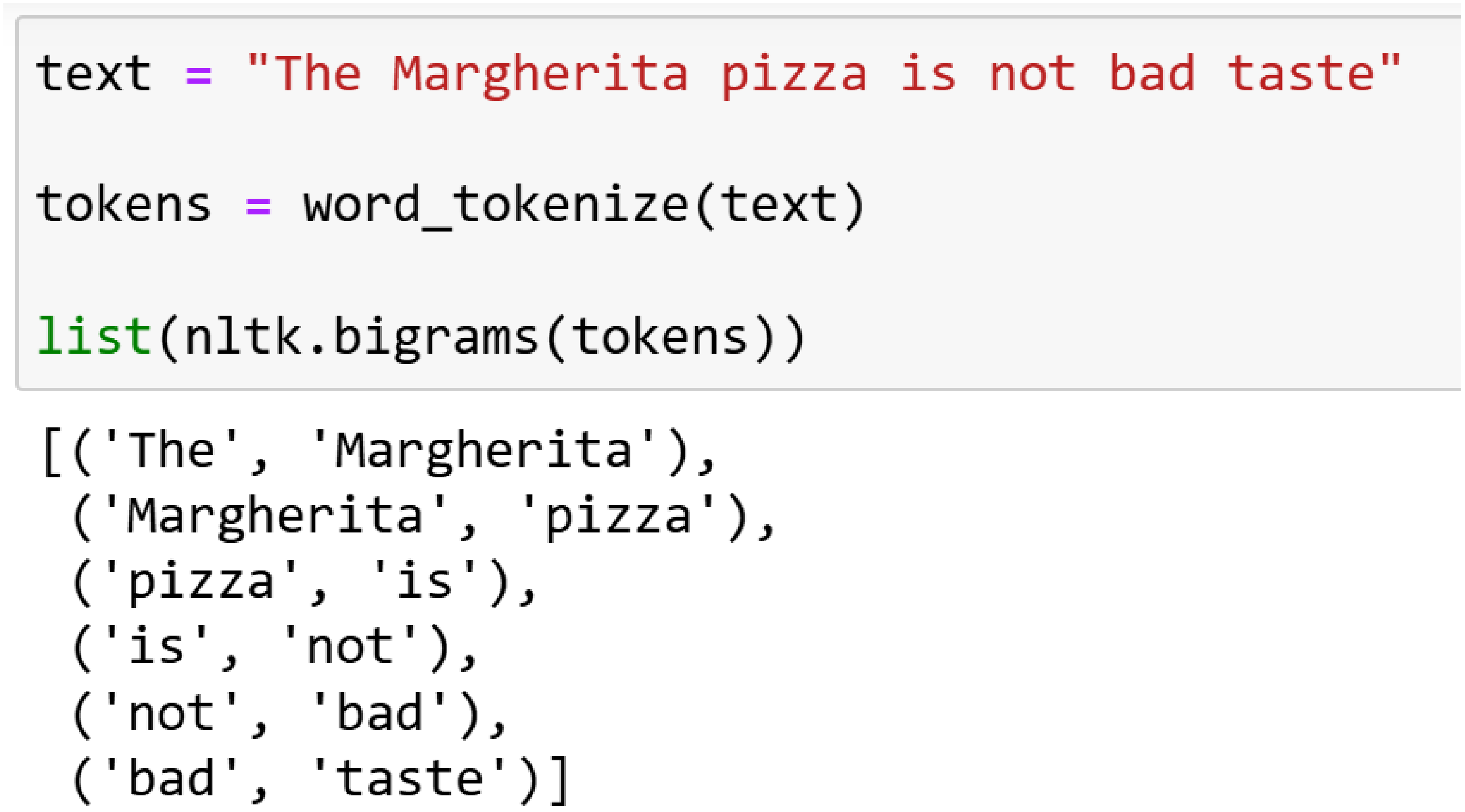
- **N-grams with NLTK**



```
text = "The Margherita pizza is not bad taste"

tokens = word_tokenize(text)
```

Série TP 3 – Analyse Lexicale

- **N-grams with NLTK**

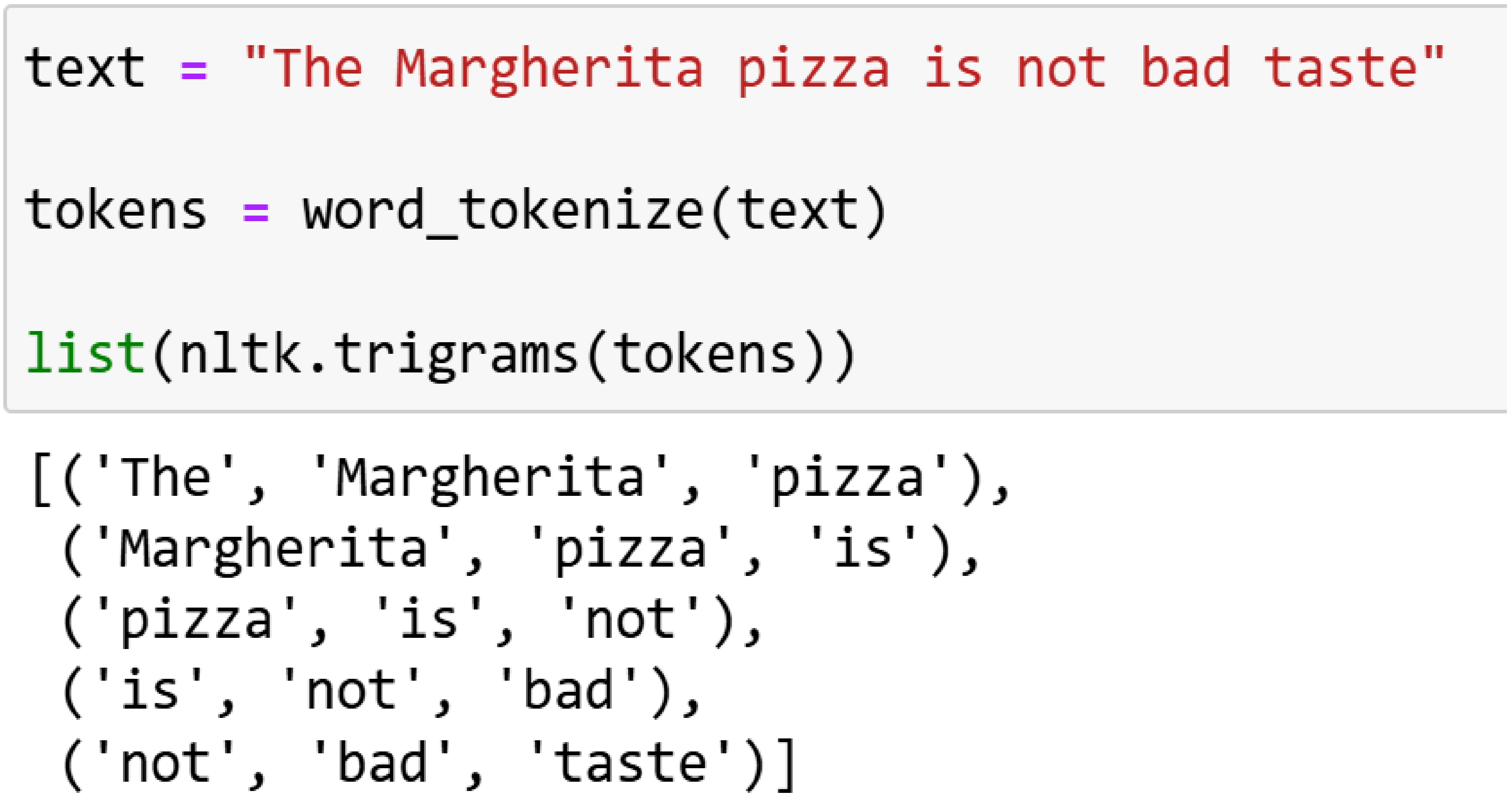


```
text = "The Margherita pizza is not bad taste"

tokens = word_tokenize(text)

list(nltk.bigrams(tokens))
```

```
[('The', 'Margherita'),
 ('Margherita', 'pizza'),
 ('pizza', 'is'),
 ('is', 'not'),
 ('not', 'bad'),
 ('bad', 'taste')]
```

Série TP 3 – Analyse Lexicale

- **N-grams with NLTK**



```python
text = "The Margherita pizza is not bad taste"

tokens = word_tokenize(text)

list(nltk.trigrams(tokens))
```

```
[('The', 'Margherita', 'pizza'),
 ('Margherita', 'pizza', 'is'),
 ('pizza', 'is', 'not'),
 ('is', 'not', 'bad'),
 ('not', 'bad', 'taste')]
```

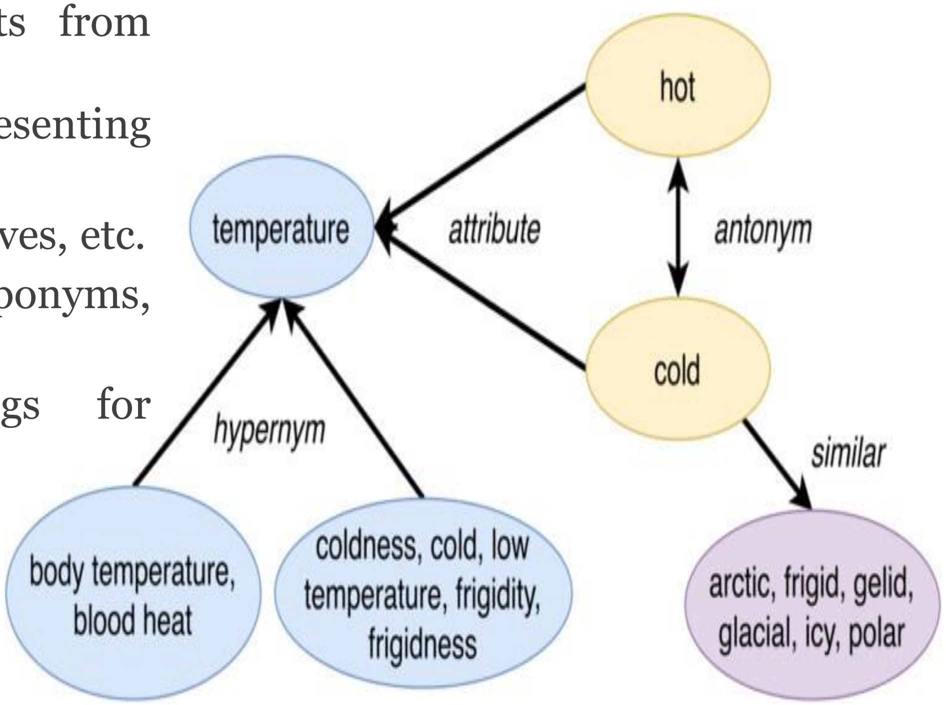
Série TP 3 – Analyse Lexicale

- **WordNet with NLTK**

- WordNet is a lexical database that organizes words into sets of synonyms called **synsets**, each representing a distinct concept.

- **WordNet Structure :**

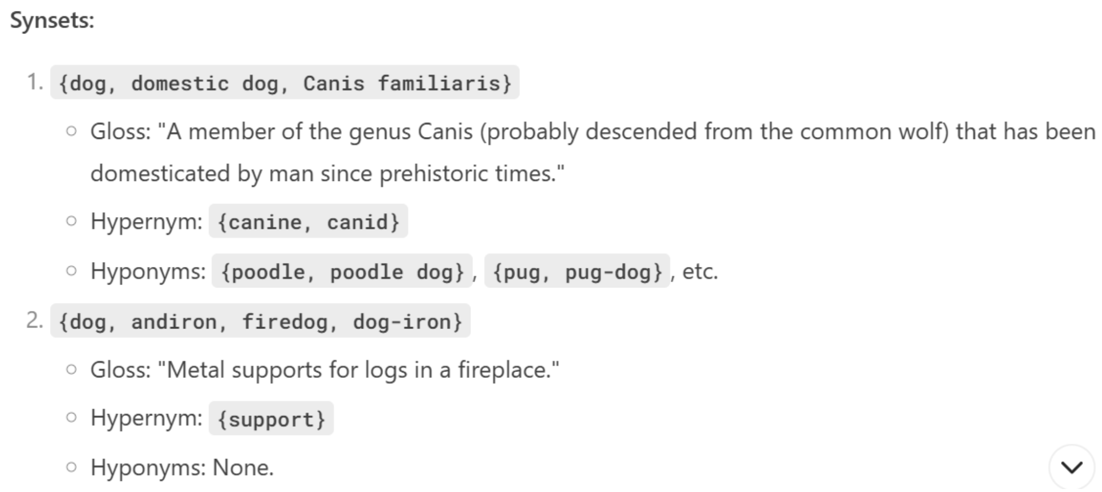
- **Hierarchy**: A taxonomy of concepts from general to specific.
- **Synsets**: Groups of synonyms representing concepts.
- **Parts of Speech**: Nouns, verbs, adjectives, etc.
- **Relationships**: Hypernyms, hyponyms, meronyms, holonyms, antonyms, etc.
- **Word Senses**: Multiple meanings for polysemous words.


hot
temperature
attribute
antonym
cold
hypernym
similar
body temperature, blood heat
coldness, cold, low temperature, frigidity, frigidness
arctic, frigid, gelid, glacial, icy, polar

Série TP 3 – Analyse Lexicale

- **WordNet with NLTK**

○ Example, the word: **dog**



Synsets:

1. `{dog, domestic dog, Canis familiaris}`
   - Gloss: "A member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times."
   - Hypernym: `{canine, canid}`
   - Hyponyms: `{poodle, poodle dog}`, `{pug, pug-dog}`, etc.
2. `{dog, andiron, firedog, dog-iron}`
   - Gloss: "Metal supports for logs in a fireplace."
   - Hypernym: `{support}`
   - Hyponyms: None.

```python
from nltk.corpus import wordnet
```

```python
dog = wordnet.synsets('dog')
```

```python
dog
```

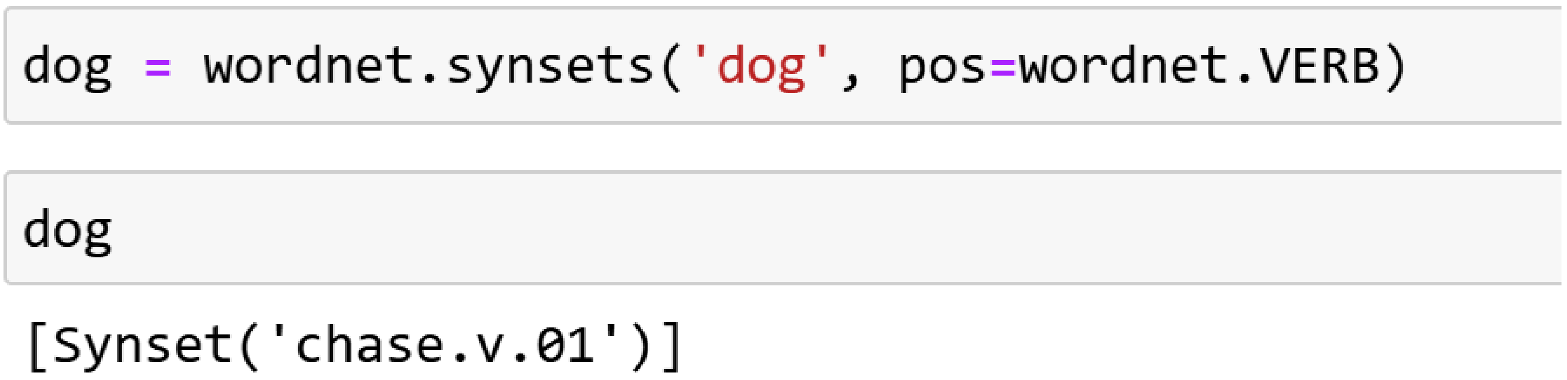
```
[Synset('dog.n.01'),
 Synset('frump.n.01'),
 Synset('dog.n.03'),
 Synset('cad.n.01'),
 Synset('frank.n.02'),
 Synset('pawl.n.01'),
 Synset('andiron.n.01'),
 Synset('chase.v.01')]
```

Série TP 3 – Analyse Lexicale

- **WordNet with NLTK**

- Example, the word: **dog**

```
from nltk.corpus import wordnet
```



```
dog = wordnet.synsets('dog', pos=wordnet.VERB)
```

```
dog
```

```
[Synset('chase.v.01')]
```

Série TP 3 – Analyse Lexicale

- **WordNet with NLTK**

- Example, the word: **dog**

```
dog
```

```
[Synset('dog.n.01'),
 Synset('frump.n.01'),
 Synset('dog.n.03'),
 Synset('cad.n.01'),
 Synset('frank.n.02'),
 Synset('pawl.n.01'),
 Synset('andiron.n.01'),
 Synset('chase.v.01')]
```

```
dog = wordnet.synsets('dog')
```

```
dog[0].definition()
```

```
'a member of the genus Canis (probably descended from the common wolf) that has
been domesticated by man since prehistoric times; occurs in many breeds'
```

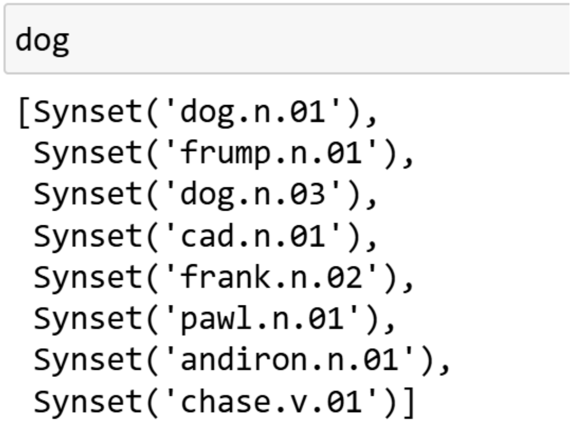
```
dog[0].examples()
```

```
['the dog barked all night']
```

Série TP 3 – Analyse Lexicale

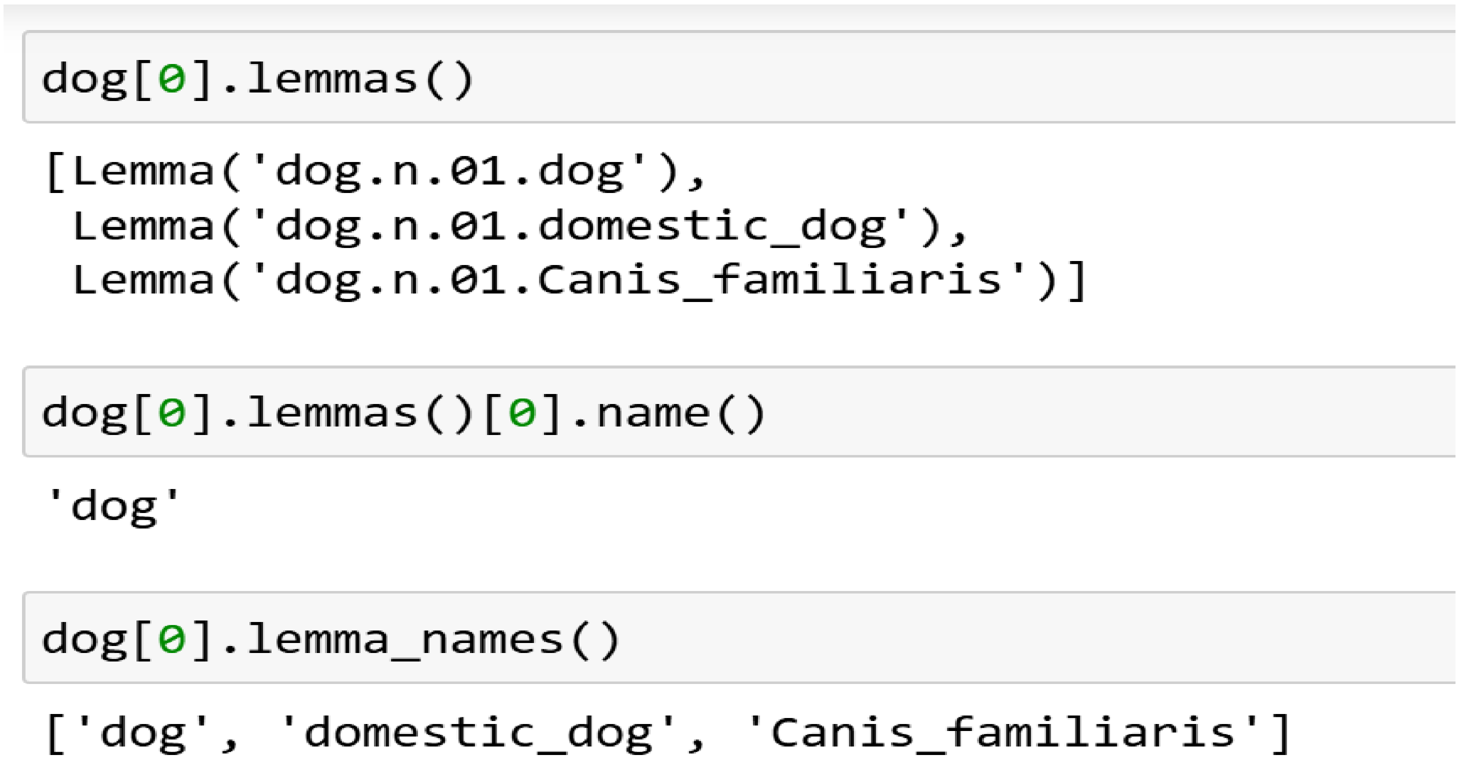
- **WordNet with NLTK**

○ Example, the word: **dog**



```
dog
```

```
[Synset('dog.n.01'),
 Synset('frump.n.01'),
 Synset('dog.n.03'),
 Synset('cad.n.01'),
 Synset('frank.n.02'),
 Synset('pawl.n.01'),
 Synset('andiron.n.01'),
 Synset('chase.v.01')]
```



```
dog[0].lemmas()
```

```
[Lemma('dog.n.01.dog'),
 Lemma('dog.n.01.domestic_dog'),
 Lemma('dog.n.01.Canis_familiaris')]
```

```
dog[0].lemmas()[0].name()
```

```
'dog'
```

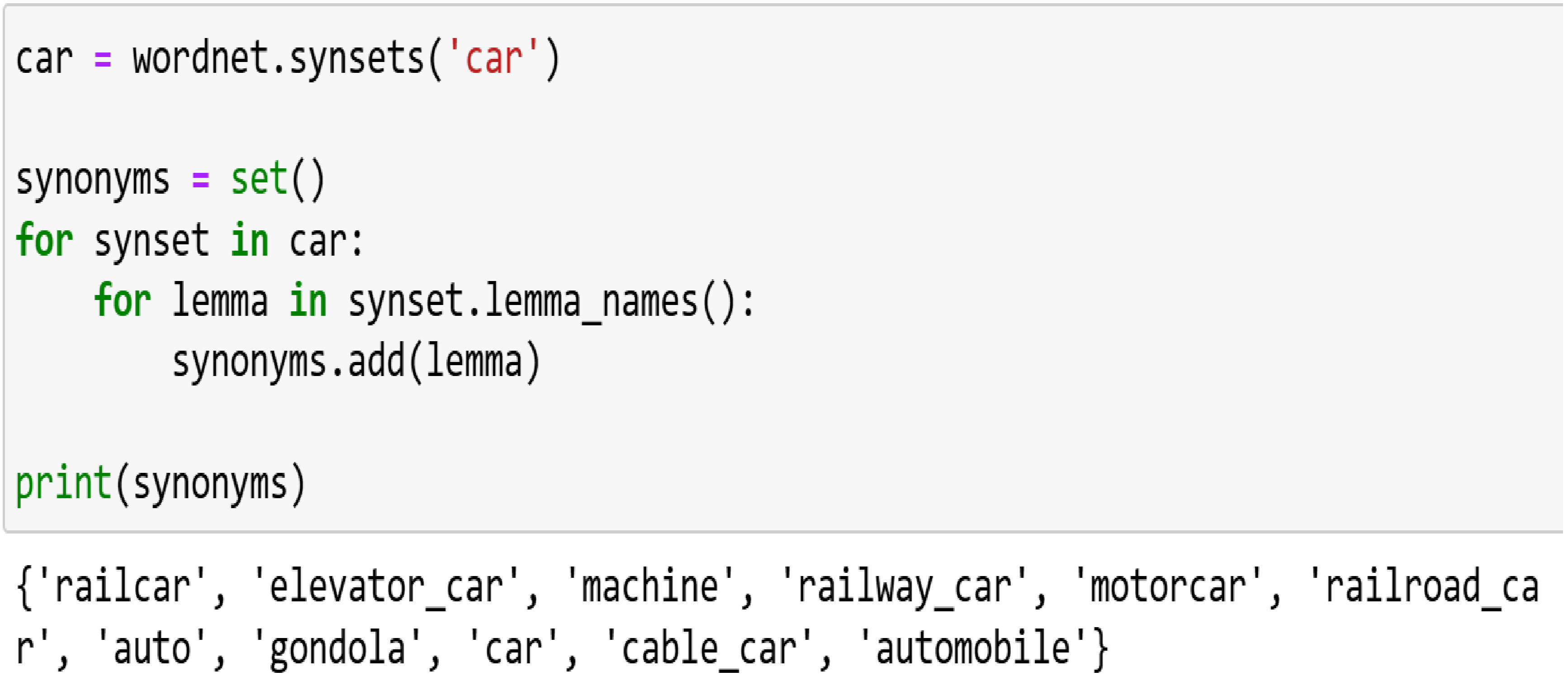
```
dog[0].lemma_names()
```

```
['dog', 'domestic_dog', 'Canis_familiaris']
```

Série TP 3 – Analyse Lexicale

- **WordNet with NLTK**

- Example, the word: **car**

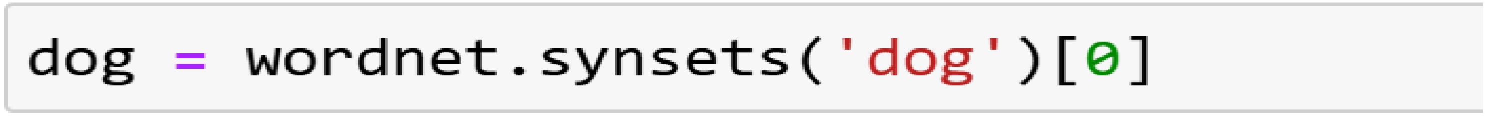


```
car = wordnet.synsets('car')

synonyms = set()
for synset in car:
    for lemma in synset.lemma_names():
        synonyms.add(lemma)

print(synonyms)
```

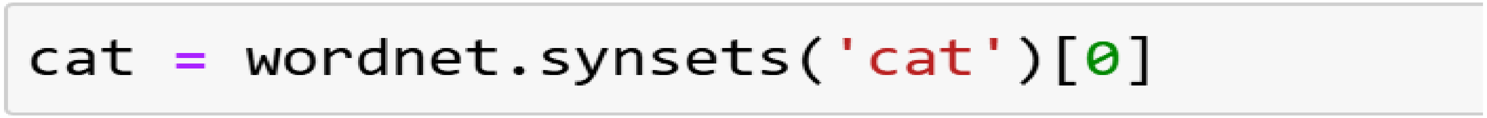
```
{'railcar', 'elevator_car', 'machine', 'railway_car', 'motorcar', 'railroad_ca
r', 'auto', 'gondola', 'car', 'cable_car', 'automobile'}
```

Série TP 3 – Analyse Lexicale

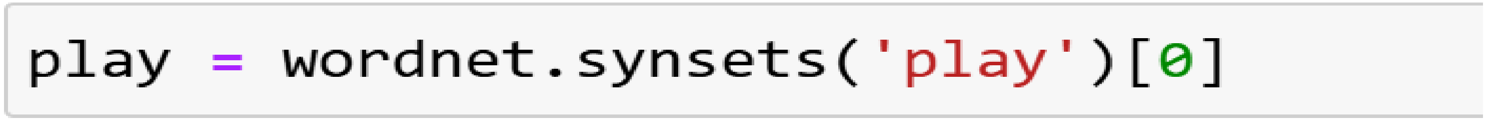
- **WordNet with NLTK**



```
dog = wordnet.synsets('dog')[0]
```



```
cat = wordnet.synsets('cat')[0]
```

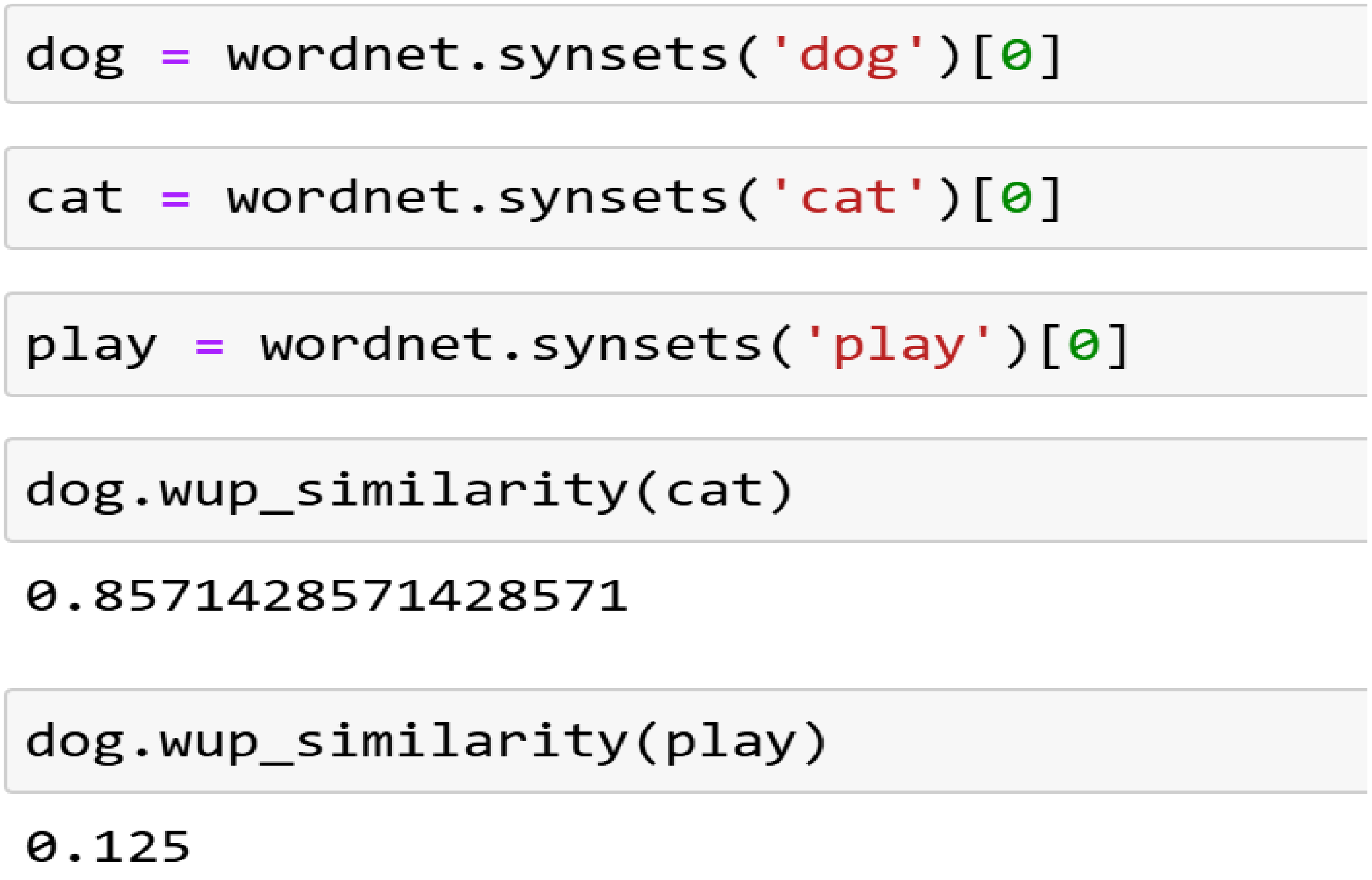


```
play = wordnet.synsets('play')[0]
```

Série TP 3 – Analyse Lexicale

$$\text{Wu} - \text{Palmer} = 2 * \frac{\text{depth}\,(\text{lcs}(s1, s2))}{(\text{depth}\,(s1)\ +\ \text{depth}\,(s2))}$$

- **WordNet with NLTK**



```
dog = wordnet.synsets('dog')[0]
```

```
cat = wordnet.synsets('cat')[0]
```

```
play = wordnet.synsets('play')[0]
```

```
dog.wup_similarity(cat)
```

```
0.8571428571428571
```

```
dog.wup_similarity(play)
```

```
0.125
```

Série TP 3 – CheatSeet

Tokenization	sent_tokenize(text)
	
	word_tokenize(text)
	
	tokenizer = RegexpTokenizer(r'\w+')
	tokens = tokenizer.tokenize(text)
	
	string.punctuation
	
	tokenizer = TweetTokenizer()
	tokens = tokenizer.tokenize(tweet)

Stopwords	stopwords.fileids()
	
	sw = stopwords.words('english')

POS Tagging	tags = pos_tag(tokens)
	
	tags = pos_tag(tokens, tagset = "universal")

Série TP 3 – CheatSeet

Stemming	porter = PorterStemmer()
	mystem = porter.stem("word")

Lemmatization	lemmatizer = WordNetLemmatizer()
	mylemma = lemmatizer.lemmatize("swimming")
	mylemma = lemmatizer.lemmatize("swimming", pos='v')

N-grams	bigrams = list(nltk.bigrams(tokens))
	
	trigrams = list(nltk.trigrams(tokens))

WordNet	dog = wordnet.synsets('dog') dog[0].definition() dog[0].examples() dog[0].lemmas() dog[0].lemmas()[0].name() dog[0].lemma_names() dog.wup_similarity(cat)

Références

Livre - Speech and Language Processing, de Dan Jurafsky.

Cours - *François Yvon* – Une petite introduction au Traitement Automatique des Langues Naturelles,
https://perso.limsi.fr/anne/coursM2R/intro.pdf

Codage des caractères : https://www.fil.univ-lille1.fr/~wegrzyno/portail/Info/Doc/HTML/seq7_codage_caracteres.html

Text Processing with Unicode - http://nltk.sourceforge.net/doc/en/app-unicode.html

Data Cleaning Challenge: Character Encodings - https://www.kaggle.com/rtatman/data-cleaning-challenge-character-encodings

Tokenization for Natural Language Processing - https://towardsdatascience.com/tokenization-for-natural-language-processing-a179a891bad4?gi=6b15f97fe07d

Cours - ARIES Abdelkrime - Le traitement automatique du langage naturel. https://github.com/projeduc/ESI_2CS_TALN