Management of Multimedia Information in Internet

Module 5. Natural Language Processing (NLP)

Unit 2. Datasets for NLP

Boni García

http://bonigarcia.github.io/
boni.garcia@uc3m.es

Telematic Engineering Department
School of Engineering

2020/2021

uc3m | Universidad Carlos III de Madrid

Table of contents

1. Introduction

- In addition to some input text, practical NLP applications usually require large bodies of **text datasets** (collections of data), such as **text corpora** or **lexicons**
- As we will discover, there are different text corpus and lexicons available (e.g. accessed easily from NLTK, in Python)
- In addition, we might need to create our own corpus. For that, we typically use online information. This technique is known as **web scraping**
- In NLP applications (and also for web scraping) we need might need to implement some **text pre-processing** (tokenization, stop words removal, etc.) in our text dataset

Table of contents

Fork me on GitHub

2. Strings in Python

- Text can be seen as a list of words. Words are represented by programming languages as a fundamental data type known as a **string**
- In this section we review the strings in **Python**
- The **declaration** of strings in Python can be as follows:

```
# Strings can be specified using single quotes
monty = 'Monty Python'
print(monty)

# ... or double quotes
circus = "Monty Python's Flying Circus"
print(circus)

# If a string contains a single quote, we must backslash-escape the quote
circus = 'Monty Python\'s Flying Circus'
print(circus)
```

```
Monty Python
Monty Python's Flying Circus
Monty Python's Flying Circus
```

2. Strings in Python

- We have different alternatives to declare strings over **several lines**:

```
# Sometimes strings go over several lines.
# Python provides us with various ways of entering them:
# a) Using backslash
couplet = "Shall I compare thee to a Summer's day?"\
   "Thou are more lovely and more temperate:"
print(couplet)

# b) Using parentheses:
couplet = ("Rough winds do shake the darling buds of May,"
   "And Summer's lease hath all too short a date:")
print(couplet)

# c) Using a triple-quoted string (to keep the newlines):
couplet = """Shall I compare thee to a Summer's day?
Thou are more lovely and more temperate:"""
print(couplet)

couplet = '''Rough winds do shake the darling buds of May,
And Summer's lease hath all too short a date:'''
print(couplet)
```

```
Shall I compare thee to a Summer's day?Thou are more lovely and more temperate:
Rough winds do shake the darling buds of May,And Summer's lease hath all too
short a date:
Shall I compare thee to a Summer's day?
Thou are more lovely and more temperate:
Rough winds do shake the darling buds of May,
And Summer's lease hath all too short a date:
```

2. Strings in Python

- Some basic operations with strings are **concatenation**, or access to **individual characters**

```python
# Concatenation
display('very' + 'very' + 'very')
display('very' * 4)
```

```
'veryveryvery'
'veryveryveryvery'
```

```python
# Accessing individual characters
display(monty)
display(monty[0])
display(monty[3])
display(monty[5])

# -1 is the index of the last character
display(monty[-1])

# -N is the index of the last N character
display(monty[-2])
```

```
'Monty Python'
'M'
't'
' '
'n'
'o'
```

```python
# Iterate characters in strings
sent = 'colorless green ideas sleep furiously'
for char in sent:
   print(char, end=' ')
```

```
c o l o r l e s s   g r e e n   i d e a s   s l
e e p   f u r i o u s l y
```

Fork me on GitHub

2. Strings in Python

- We can make **substrings** using brackets, as follows:

```
# We use [ ] for slides (e.g. for substrings or sublists)
# It starts at the first index but finishes one before the end index
display(monty)
display(monty[6:10])
display(monty[-4:-1])

# If we omit the first value, the slide begins at the start of the string or list
display(monty[:5])

# If we omit the second value, the slide continues to the end of the string or list
display(monty[6:])
```

```
'Monty Python'
'Pyth'
'tho'
'Monty'
'Python'
```

- Basic **search** within strings can be done with the keyword `in` or the method `find()`:

```
# We can check if a string is contained in other using the in operator:
phrase = 'And now for something completely different'
if 'thing' in phrase:
   print('found "thing"')

# We can also find the position of a string within other using find():
monty.find('Python')
```

```
found "thing"
6
```

2. Strings in Python

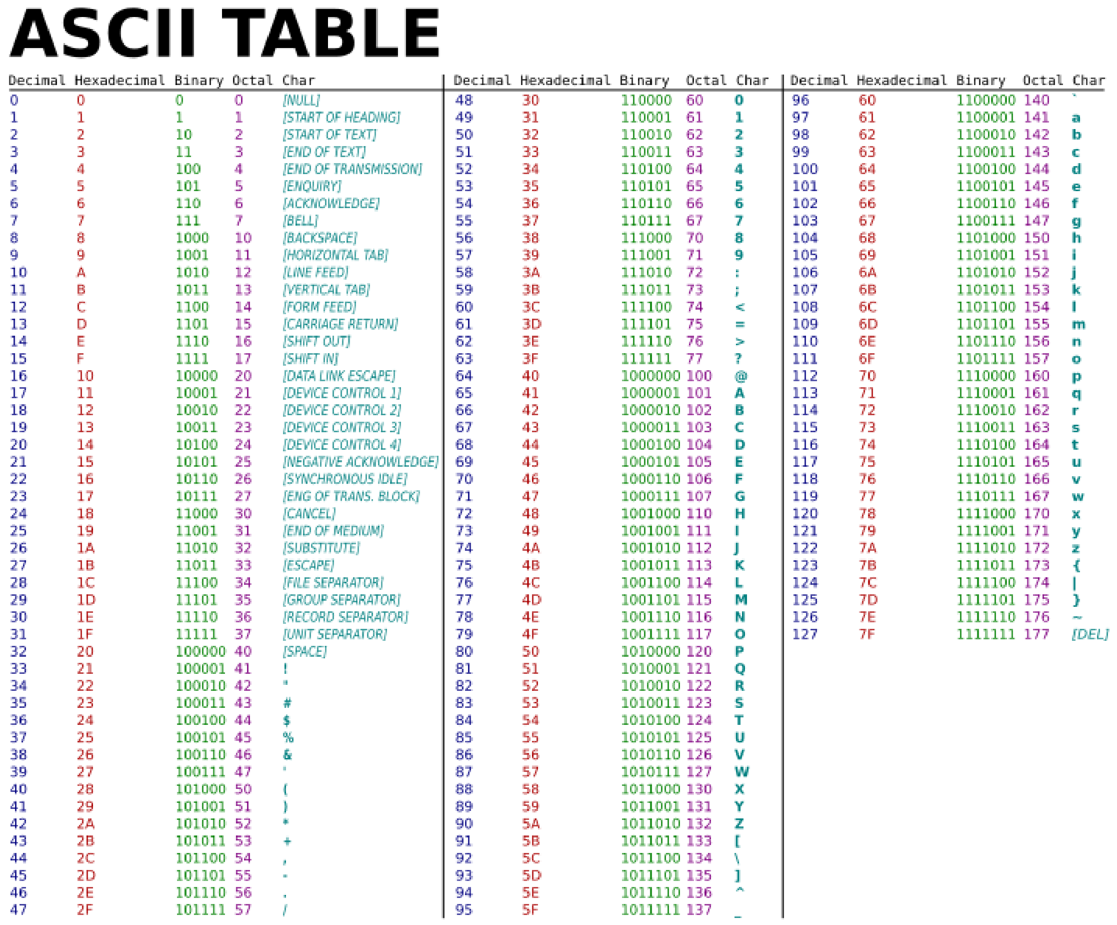
- The following table summarizes **other useful operations** with strings in Python:

Method	Functionality
`s.find(t)`	index of first instance of string t inside `s` (`-1` if not found)
`s.rfind(t)`	index of last instance of string t inside `s` (`-1` if not found)
`s.index(t)`	like `s.find(t)` except it raises `ValueError` if not found
`s.rindex(t)`	like `s.rfind(t)` except it raises `ValueError` if not found
`s.join(text)`	combine the words of the text into a string using `s` as the glue
`s.split(t)`	split `s` into a list wherever a `t` is found (whitespace by default)
`s.splitlines()`	split `s` into a list of strings, one per line
`s.lower()`	a lowercased version of the string `s`
`s.upper()`	an uppercased version of the string `s`
`s.title()`	a titlecased version of the string `s`
`s.strip()`	a copy of `s` without leading or trailing whitespace
`s.replace(t, u)`	replace instances of `t` with u inside `s`

2. Strings in Python - Unicode

- NLP applications deal with different languages. These languages use different **character sets** (e.g. Latin, Cyrillic, or Chinese, to name a few)
- In computing, the concept of "plain text" is a fiction. Instead, we use different **character encodings** to represent the character sets, e.g.:
  - ASCII (American Standard Code for Information Interchange)
    - The original ASCII table is encoded on **7 bits** therefore (128 characters)
    - Nowadays the extended ASCII table (ISO 8859-1) is encoded on **8 bits** (256 characters)
  - UTF-8 (Unicode Transformation Format - 8 bit)
    - UTF-8 is capable of encoding all 1,112,064 valid character code (using **1**, **2**, or **4 bytes**)
    - UTF-8 supports any Unicode character, which pragmatically means any natural language, as well as many non-spoken languages (e.g. music notation or mathematical symbols)
    - Nowadays, UTF-8 is by far the most common encoding for the Web, accounting for over 95% of all web pages, and up to 100% for some languages, as of 2020

2. Strings in Python - Unicode



ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[ENG OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	|
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(	88	58	1011000	130	X					
41	29	101001	51	)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[					
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135	]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

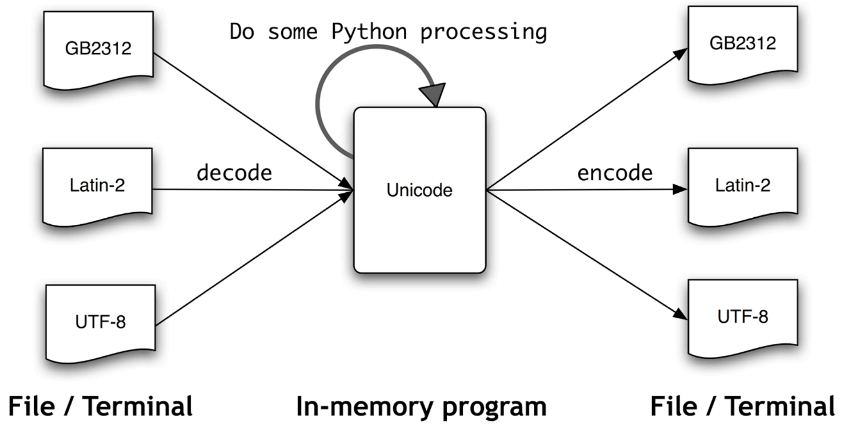
Source: Wikipedia
https://en.wikipedia.org/wiki/ASCII

2. Strings in Python - Unicode

UNICODE

https://home.unicode.org/

- **Unicode** is an standard for the encoding of text expressed
  - This standard is maintained by the Unicode Consortium
- Stings in **Python** use the Unicode for representing characters
  - When reading strings from a data source in Python (e.g. a text file), the character encoding is translated to Unicode
  - Each character is assigned a number, called a code point (encoded as \uXXXX for 16-bits or \UXXXXXXXX for 32-bits)



Source: Bird, S., Klein, E., Loper, E. (2009) ***Natural Language Processing with Python***. O'Reilly Media. http://www.nltk.org/book/

2. Strings in Python - Unicode

- Basic example using characters in code point format:

```python
print(ord('ñ'))

# 241 in decimal is the same than 0x00F1 in hexadecimal
n_tilde = '\u00F1'
print(n_tilde)

python_emoji = '\U0001F40D'
print('Learning', python_emoji)
```

```
241
ñ
Learning 🐍
```

2. Strings in Python - Regular Expressions

- Many linguistic processing tasks involve **pattern matching**
    - For example, finding words ending with *ed* (for regular past tense in English)
- **Regular expressions** (or simply RegEx) provides a powerful and flexible method for describing and searching character patterns
- Python has a built-in package called `re` to work with RegEx. This module offers the following functions (among others):

Method	**Functionality**
`re.findall(pattern, string)`	Returns a list containing all matches
`re.search(pattern, string)`	Returns a `Match` object if there is a coincidence anywhere in the string
`re.split(pattern, string)`	Returns a list where the string has been split at each match
`re.sub(pattern, replace, string)`	Replaces one or many matches with a string

2. Strings in Python - Regular Expressions

- RegEx patterns in Python can be built using metacharacters and special sequences:

Metacharacter	Functionality
[ ]	A set of characters
\	Signals a special sequence (can also be used to escape special characters)
.	Any character (except newline character)
^	Starts with
$	Ends with
*	Zero or more occurrences
+	One or more occurrences
{ }	Exactly the specified number of occurrences
|	Either or
( )	Capture and group
?	Take the shortest match (non-greedy)

Special seq.	Functionality
\A	Match if the specified characters are at the beginning of the string
\b	Match where the specified characters are at the beginning or at the end of a word
\B	Match where the specified characters are present, but NOT at the beginning
\d	Match where the string contains digits
\D	Match where the string DOES NOT contain digits
\s	Match where the string contains a white space character
\S	Match where the string DOES NOT contain a white space character
\w	Match where the string contains any word characters
\W	Match where the string DOES NOT contain any word characters
\Z	Match if the specified characters are at the end of the string

2. Strings in Python - Regular Expressions

- Basic example using RegEx (`findall` and `search`):

```
import re

txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)

# r prefix to a string indicates that the string is a raw string
# (i.e. backslashes \ should be treated literally and not as escape characters)
x = re.search(r"\bS\w+", txt) # Match object

# The Match object has properties and methods used to retrieve information about
 the search, and the result:
# .span() returns a tuple containing the start-, and end positions of the match
# .string returns the string passed into the function
# .group() returns the part of the string where there was a match

print(x.span())
print(x.string)
print(x.group())
```

```
['ai', 'ai']
(12, 17)
The rain in Spain
Spain
```

Table of contents

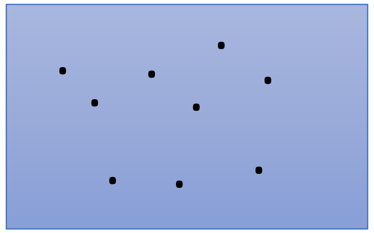
3. Text corpora

- A text **corpus** (corpora in plural) is a large body of text
- Text corpora is required in NLP applications for different reasons, such as:
  - To train and validate ML models (in statistical NLP approaches)
  - Hypothesis testing, for example checking occurrences or validating linguistic rules within a specific language
  - Benchmarking models (i.e. evaluation)

3. Text corpora

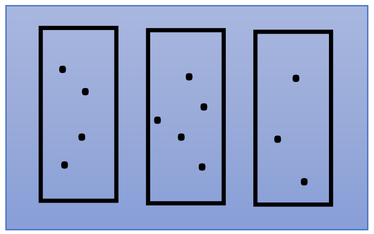
- There are different types of corpora, such as:
  - Isolated: texts with no particular organization
  - Categorized: structured into categories (e.g. genre)
  - Overlapping: documents can belong to different categories
  - Temporal: language use over time

Isolated



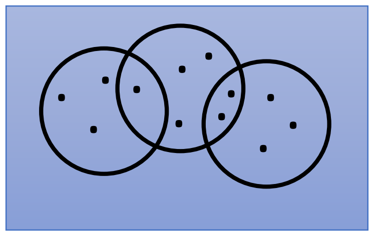
For example:
Gutenberg Corpus

Categorized



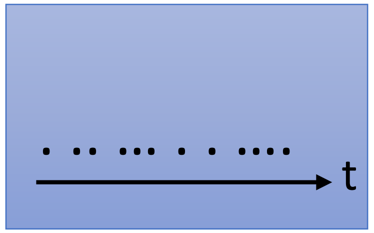
For example:
Brown Corpus

Overlapping



For example:
Reuters Corpus

Temporal



For example:
Inaugural Address Corpus

3. Text corpora - NLTK

- **NLTK** provides a convenient interface to access different text corpora in Python
- The `nltk.corpus` package defines a collection of **corpus reader** classes, which can be used to access the contents of a diverse set of corpora
- The list of available corpora in NLTK is given at:

http://www.nltk.org/nltk_data/

3. Text corpora - NLTK

- The basic Corpus functionality defined in NLTK is summarized in the following table:

Method	Functionality
`fileids()`	the files of the corpus
`fileids([categories])`	the files of the corpus corresponding to these categories
`categories()`	the categories of the corpus
`categories([fileids])`	the categories of the corpus corresponding to these files
`raw()`	the raw content of the corpus
`raw(fileids=[f1,f2,f3])`	the raw content of the specified files
`raw(categories=[c1,c2])`	the raw content of the specified categories
`words()`	the words of the whole corpus
`words(fileids=[f1,f2,f3])`	the words of the specified fileids
`words(categories=[c1,c2])`	the words of the specified categories
`sents()`	the sentences of the whole corpus
`sents(fileids=[f1,f2,f3])`	the sentences of the specified fileids
`sents(categories=[c1,c2])`	the sentences of the specified categories
`abspath(fileid)`	the location of the given file on disk
`encoding(fileid)`	the encoding of the file (if known)
`open(fileid)`	open a stream for reading the given corpus file
`root`	if the path to the root of locally installed corpus
`readme()`	the contents of the README file of the corpus

3. Text corpora - Gutenberg corpus

- The **Project Gutenberg** is an online library of over 60,000 free **eBooks**
  - It is a volunteer effort to digitize and archive cultural works
  - Most releases are in text format (but other formats, such as HTML, PDF, EPUB, MOBI are also available)
  - Most of them in English (but other languages are also available)
- NLTK includes a small selection of texts from the Project Gutenberg

https://www.gutenberg.org/

3. Text corpora - Gutenberg corpus

- Basic example using the Gutenberg corpus with NLTK:

```
import nltk
from nltk.corpus import gutenberg
nltk.download("Gutenberg")

# The method fileids() returns a list with the files of the corpus
print(gutenberg.fileids())

# The medhod words() returns a list with the words a given corpus (e.g., the ebook "Emma" by Jane Austen)
emma = gutenberg.words("austen-emma.txt")
print(len(emma))
```

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-kjv.txt', 'blake-poems.txt',
'bryant-stories.txt', 'burgess-busterbrown.txt', 'carroll-alice.txt', 'chesterton-ball.txt', 'chesterton-
brown.txt', 'chesterton-thursday.txt', 'edgeworth-parents.txt', 'melville-moby_dick.txt', 'milton-
paradise.txt', 'shakespeare-caesar.txt', 'shakespeare-hamlet.txt', 'shakespeare-macbeth.txt', 'whitman-
leaves.txt']
192427
```

Fork me on GitHub

Fork me on GitHub

3. Text corpora - Brown corpus

- The **Brown corpus** is an electronic collection of text samples of American English which contains text from 500 sources, and the sources have been **categorized** by genre, such as news, editorial, etc.

```
from nltk.corpus import brown
nltk.download("brown")

# The method categories() can be use in categorized corpus (such as Brown)
print(brown.categories())

news_text = brown.words(categories="news")
print(len(news_text))
```

```
[nltk_data] Downloading package brown to /root/nltk_data... [nltk_data]
Package brown is already up-to-date! ['adventure', 'belles_lettres',
'editorial', 'fiction', 'government', 'hobbies', 'humor', 'learned',
'lore', 'mystery', 'news', 'religion', 'reviews', 'romance',
'science_fiction'] 100554
```

The Brown Corpus is a convenient resource for studying systematic differences between genres, a kind of linguistic inquiry known as stylistics

Fork me on GitHub

3. Text corpora - Reuters corpus

- The **Reuters corpus** contains 10,788 **news** documents totaling 1.3 million words

The documents have been **categorize** into 90 topics, and grouped into two sets, called "training" and "test" (very convenient for training and validating ML model)

```
from nltk.corpus import reuters
nltk.download("reuters")

print(reuters.fileids())
print(reuters.categories())
print(reuters.words("training/9865")[:14])
```

```
[nltk_data] Downloading package reuters to /root/nltk_data...
[nltk_data]   Package reuters is already up-to-date!
['test/14826', 'test/14828', 'test/14829', 'test/14832', ..., 'training/9994', 'training/9995']
['acq', 'alum', 'barley', 'bop', 'carcass', 'castor-oil', 'cocoa', 'coconut', 'coconut-oil', 'coffee', 'copper', 'copra-
cake', 'corn', 'cotton', 'cotton-oil', 'cpi', 'cpu', 'crude', 'dfl', 'dlr', 'dmk', 'earn', 'fuel', 'gas', 'gnp', 'gold',
'grain', 'groundnut', 'groundnut-oil', 'heat', 'hog', 'housing', 'income', 'instal-debt', 'interest', 'ipi', 'iron-
steel', 'jet', 'jobs', 'l-cattle', 'lead', 'lei', 'lin-oil', 'livestock', 'lumber', 'meal-feed', 'money-fx', 'money-
supply', 'naphtha', 'nat-gas', 'nickel', 'nkr', 'nzdlr', 'oat', 'oilseed', 'orange', 'palladium', 'palm-oil',
'palmkernel', 'pet-chem', 'platinum', 'potato', 'propane', 'rand', 'rape-oil', 'rapeseed', 'reserves', 'retail', 'rice',
'rubber', 'rye', 'ship', 'silver', 'sorghum', 'soy-meal', 'soy-oil', 'soybean', 'strategic-metal', 'sugar', 'sun-meal',
'sun-oil', 'sunseed', 'tea', 'tin', 'trade', 'veg-oil', 'wheat', 'wpi', 'yen', 'zinc']
['FRENCH', 'FREE', 'MARKET', 'CEREAL', 'EXPORT', 'BIDS', 'DETAILED', 'French', 'operators', 'have', 'requested',
'licences', 'to', 'export']
```

3. Text corpora - Inaugural address corpus

- The **inaugural address corpus** is a collection for each US presidents' inaugural addresses since 1789

```
from nltk.corpus import inaugural
nltk.download("inaugural")

print(inaugural.fileids())
print(len(inaugural.fileids()))
```

```
[nltk_data] Downloading package inaugural to /root/nltk_data...
[nltk_data]   Package inaugural is already up-to-date!
['1789-Washington.txt', '1793-Washington.txt', '1797-Adams.txt', '1801-Jefferson.txt', '1805-Jefferson.txt',
'1809-Madison.txt', '1813-Madison.txt', '1817-Monroe.txt', '1821-Monroe.txt', '1825-Adams.txt', '1829-
Jackson.txt', '1833-Jackson.txt', '1837-VanBuren.txt', '1841-Harrison.txt', '1845-Polk.txt', '1849-Taylor.txt',
'1853-Pierce.txt', '1857-Buchanan.txt', '1861-Lincoln.txt', '1865-Lincoln.txt', '1869-Grant.txt', '1873-
Grant.txt', '1877-Hayes.txt', '1881-Garfield.txt', '1885-Cleveland.txt', '1889-Harrison.txt', '1893-
Cleveland.txt', '1897-McKinley.txt', '1901-McKinley.txt', '1905-Roosevelt.txt', '1909-Taft.txt', '1913-
Wilson.txt', '1917-Wilson.txt', '1921-Harding.txt', '1925-Coolidge.txt', '1929-Hoover.txt', '1933-Roosevelt.txt',
'1937-Roosevelt.txt', '1941-Roosevelt.txt', '1945-Roosevelt.txt', '1949-Truman.txt', '1953-Eisenhower.txt',
'1957-Eisenhower.txt', '1961-Kennedy.txt', '1965-Johnson.txt', '1969-Nixon.txt', '1973-Nixon.txt', '1977-
Carter.txt', '1981-Reagan.txt', '1985-Reagan.txt', '1989-Bush.txt', '1993-Clinton.txt', '1997-Clinton.txt',
'2001-Bush.txt', '2005-Bush.txt', '2009-Obama.txt', '2013-Obama.txt', '2017-Trump.txt']
58
```

Table of contents

4. Lexicons

- A **lexicon** (also known as or **lexical resource**) is a collection of words and/or phrases along with associated information, for example, part of speech (i.e., noun, pronoun, verb, adjective, adverb, preposition, conjunction, or interjection), definitions, semantic information, etc.
- A lexicon can be seen as a **dictionary** (typically stored as a lexical database)

4. Lexicons - Stop words

- A basic lexicon available in NLTK is the **stop words list**
- Stop words refers to the most common words in a language, (such as *the*, *to* and *also* in English)
  - We usually want to filter out of these words from a document as part of the pre-processing

```python
import nltk
from nltk.corpus import stopwords
nltk.download("stopwords")

sw_en = stopwords.words("english")
print(len(sw_en))
print(sw_en[:5], "...", sw_en[-5:])
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
179
['i', 'me', 'my', 'myself', 'we'] ... ["weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

4. Lexicons - WordNet

- **WordNet** is a lexical database of semantic relations between words in more than 200 languages
- WordNet links words into semantic relations including:
  - Synonyms, i.e. words or phrases that means exactly or nearly the same as another word
  - Hyponyms, i.e. type-of relationship among words (e.g. blue, yellow, or red are all hyponyms of *color*)
  - Meronyms, i.e. part-of relationship among words (e.g. finger is a meronym of hand)

https://wordnet.princeton.edu/

4. Lexicons - WordNet

- **NLTK** includes the English WordNet. A basic example is:

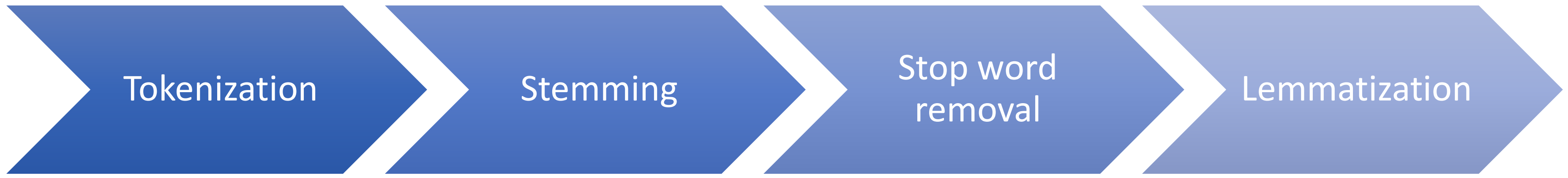
```
from nltk.corpus import wordnet
nltk.download("wordnet")

# The synsets() method return a list of Synset (synonym set) objects
ss_car_list = wordnet.synsets("motorcar")

# In this example, the word 'motorcar' has just one possible meaning and it is identified as 'car.n.01'
print(ss_car_list)

for ss_car in ss_car_list:
   print("Synonymous words (or 'lemmas'):", ss_car.lemma_names())
   print("Definition:", ss_car.definition())
   print("Examples:", ss_car.examples())
   print("Types:", ss_car.hyponyms())
   print("Parts:", ss_car.part_meronyms())
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[Synset('car.n.01')]
Synonymous words (or "lemmas"): ['car', 'auto', 'automobile', 'machine', 'motorcar']
Definition: a motor vehicle with four wheels; usually propelled by an internal combustion engine
Examples: ['he needs a car to get to work']
Types: [Synset('ambulance.n.01'), Synset('beach_wagon.n.01'), Synset('bus.n.04'), ..., Synset('used-
car.n.01')]
Parts: [Synset('accelerator.n.01'), Synset('air_bag.n.01'), Synset('auto_accessory.n.01'), ...,
Synset('window.n.02')]
```

Table of contents

5. Text preprocessing

- **Text preprocessing** is a common step in NLP pipelines
- It is aimed to transform text from human language to some more convenient format for further processing
- The typical steps in text preprocessing are: tokenization, removing stop words, stemming, and lemmatization



5. Text preprocessing - Tokenization

- Tokenization is usually the first step in NLP text preprocessing tasks
  - It is the technique where raw text is chopped into small pieces (called tokens)
  - **Tokens** are the basic units of text involved in any NLP task (typically, words)
- NLTK provides different types of **tokenizers** for doing this step, such as:

NLTK tokenizer	Description
`sent_tokenize`	Tokenize as sentences
`word_tokenize`	Tokenize as words (treat most punctuation characters as separate tokens)
`wordpunct_tokenize`	Tokenize as words (all special symbols and treat them as separate units)
`regexp_tokenize`	Custom tokenizers using NLTK's regular expression

Fork me on GitHub

5. Text preprocessing - Tokenization

- Tokenization examples:

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

my_message = "Hello there. Goodbye everybody."
tokens = sent_tokenize(my_message)
print(tokens)
```

```
['Hello there.', 'Goodbye everybody.']
```

```python
from nltk.tokenize import wordpunct_tokenize

tokens = wordpunct_tokenize(my_message)
print(tokens)
```

```
['@', 'Everybody', ':', 'Hello', 'NLP', '-', 'world', '!']
```

```python
from nltk.tokenize import word_tokenize
nltk.download("punkt")

my_message = "@Everybody: Hello NLP-world!"
tokens = word_tokenize(my_message)
print(tokens)
```

```
[nltk_data] Downloading package punkt to
/root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
['@', 'Everybody', ':', 'Hello', 'NLP-world', '!']
```

```python
from nltk.tokenize import regexp_tokenize

tokens = regexp_tokenize(my_message, r"\w+")
print(tokens)
```

```
['Everybody', 'Hello', 'NLP', 'world']
```

5. Text preprocessing - Stemming

- **Stemming** is a text preprocessing task for transforming related or similar variants of a word. For example:
  - Words sharing the sample meaning: *walking* to its base form (to walk)
  - Reduce a plural word to its singular form: *apples* is reduced to *apple*
- In NLTK there are different stemmers:
  - **Porter stemmer**. One of the most commonly used. It supports only the English language. It is based on the Porter Algorithm (written and maintained Martin Porter)
  - **Snowball stemmer**. It is an improvement on the Porter stemmer, and supports multiple languages

5. Text preprocessing - Stemming

- Some basic examples:

These examples use a list defined used the **list-comprehension** notation of Python

```python
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()
tokens = ["Enjoy", "enjoying", "enjoys", "enjoyable"]
stems = [stemmer.stem(token) for token in tokens]
print(stems)
```

```
['enjoy', 'enjoy', 'enjoy', 'enjoy']
```

```python
from nltk.stem.snowball import SnowballStemmer

stemmer = SnowballStemmer("english")
stems = [stemmer.stem(token) for token in tokens]
print(stems)
```

```
['enjoy', 'enjoy', 'enjoy', 'enjoy']
```

Fork me on GitHub

5. Text preprocessing - Removing stop words

- **Stop word removal** is a step in which remove words that do not signify any importance to the document, such as grammar articles and pronouns, such as (in English): *a*, *an*, *he*, and *her*
  - For this, we can use the stop words list lexicon available in NLTK

```
from nltk.corpus import stopwords
nltk.download("stopwords")

example_text = "This is an example sentence to test stopwords"
sw_en = stopwords.words("english")

text_no_stopwords = [word for word in example_text.split() if word not in sw_en]
print(example_text)
print(text_no_stopwords)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
This is an example sentence to test stopwords
['This', 'example', 'sentence', 'test', 'stopwords']
```

Fork me on GitHub

5. Text preprocessing - Lemmatization

- **Lemmatization** is closely related to stemming: both processes try to identify a canonical representative for a set of related word forms (i.e. to reduce morphological variation)
- The difference is that a stemmer operates on a single word without knowledge of the context, and a lemmatizer takes into consideration the morphological analysis of the words
- Lemmatization make sure that the resulting form is a known word in a dictionary

5. Text preprocessing - Lemmatization

- To implement lemmatization with NLTK, the part-of-speech (POS) tagging (e.g. label each word as a noun, verb, article, adjective, preposition, etc.) should be done first
- To avoid this, we can use the **spaCy** lemmatizer, which comes with pretrained models that can do this process automatically

```python
import spacy
nlp = spacy.load("en")

sentence="We are putting in efforts to enhance our understanding of Lemmatization"

lemmas = [token.lemma_ for token in nlp(sentence)]
print(lemmas)

lemmas = [w.lemma_ if w.lemma_ !='-PRON-' else w.text for w in nlp(sentence)]
print(lemmas)
```

```
['-PRON-', 'be', 'put', 'in', 'effort', 'to', 'enhance', '-PRON-', 'understanding',
'of', 'lemmatization']
['We', 'be', 'put', 'in', 'effort', 'to', 'enhance', 'our', 'understanding', 'of',
'lemmatization']
```

Unlike verbs and common nouns, there's no clear base form of a personal pronoun. spaCy's solution is to introduce the symbol −PRON−, which is used as the lemma for all personal pronouns

5. Text preprocessing - POS tagging

- **Part-of-speech (POS) tagging** refers to categorizing the words in a sentence into specific syntactic or grammatical functions
  - For instance, in English, label each word as: nouns, pronouns, adjectives, verbs, adverbs, prepositions, determiners, and conjunctions
  - POS tagging finds applications in Named Entity Recognition (NER), sentiment analysis, question answering, or word sense disambiguation
- NLTK provides the function `pos_tag()` to carry out POS tagging
  - This tagger uses a pre-trained model for the English language
  - By default, it uses the set of tags from the **Penn Treebank project,** which is a POS tagged corpus from journal articles, or telephone conversations

5. Text preprocessing - POS tagging

- Alphabetical list of POS tags used in the Penn Treebank Project:

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	to
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Fork me on GitHub

5. Text preprocessing - POS tagging

- Example of NLTK’s **`pos_tag()`**:

```python
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag
nltk.download("punkt")
nltk.download("averaged_perceptron_tagger")

sentence = "We are putting in efforts to enhance our understanding of Lemmatization"

tokens = word_tokenize(sentence)
pos_tagged = pos_tag(tokens)

print("tokens", tokens)
print("pos_tagged", pos_tagged)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
tokens ['We', 'are', 'putting', 'in', 'efforts', 'to', 'enhance', 'our', 'understanding', 'of',
'Lemmatization']
pos_tagged [('We', 'PRP'), ('are', 'VBP'), ('putting', 'VBG'), ('in', 'IN'), ('efforts', 'NNS'), ('to',
'TO'), ('enhance', 'VB'), ('our', 'PRP$'), ('understanding', 'NN'), ('of', 'IN'), ('Lemmatization', 'NN')]
```

5. Text preprocessing - POS tagging

- POS tagging is also required to carry out lemmatization with NLTK, using the `WordNetLemmatizer`
- This lemmatizer provides the method `lemmatize(word, pos)`,
  - The first argument is the word to be lemmatized
  - The second argument is the POS tag using the WordNet categories

Tag	Description
n	Nouns
v	Verbs
a	Adjective
r	Adverbs

To use **`WordNetLemmatizer`** together with **`pos_tag()`**, we need to make a conversion from the Treebank tag set to WordNet

5. Text preprocessing - POS tagging

- Example of NLTK's **`WordNetLemmatizer`**:

```python
from nltk.corpus import wordnet
from nltk.stem.wordnet import WordNetLemmatizer
nltk.download("wordnet")

# Return tag compliance to WordNet lemmatization (a, n, r, v)
def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN # Nouns by default

lemmatizer = WordNetLemmatizer()
lemmas = [lemmatizer.lemmatize(word, pos=get_wordnet_pos(tag)) for word, tag in pos_tagged]

print("lemmas", lemmas)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
lemmas ['We', 'be', 'put', 'in', 'effort', 'to', 'enhance', 'our', 'understanding', 'of', 'Lemmatization']
```

Table of contents

6. Exploratory analysis - Vocabulary size

- Once we have completed the text preprocessing, we can carry out further analysis
- One of the basic analyses that is commonly performed is counting the **vocabulary size**, i.e. its number of elements

```
import nltk
from nltk.corpus import gutenberg
nltk.download("gutenberg")

all_words = gutenberg.words("melville-moby_dick.txt")
print("Total words", len(all_words))

vocabulary = sorted(set(all_words))
print("Vocabulary size", len(vocabulary))
```

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
Total words 260819
Vocabulary size 19317
```

This example creates a **vocabulary** using a Python's `set` (i.e. a unorder collection of **unique** elements) in Python. The set can be ordered using the Python built-in function `sorted`

Fork me on GitHub

6. Exploratory analysis - Lexical diversity

- **Lexical diversity** is the ratio of different unique word to the total number of words
- It is considered as an important indicator of how complex and difficult to read a text is (lexical richness)
  - The more varied a vocabulary a text possesses, the higher lexical diversity

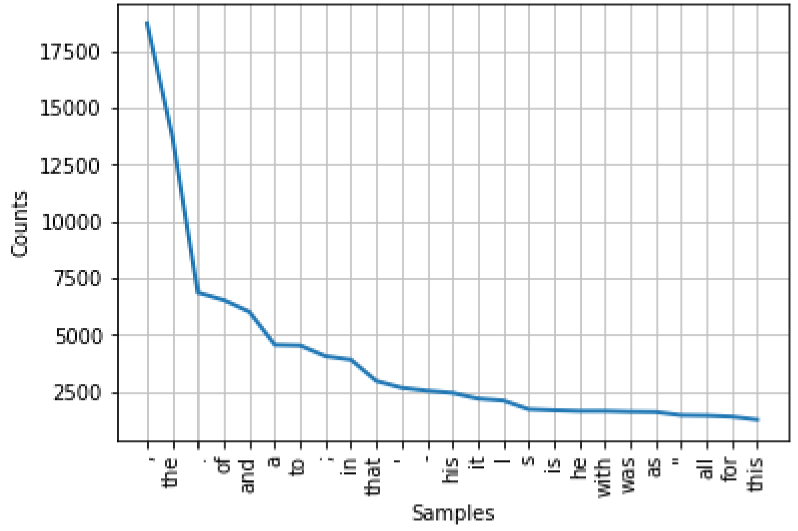
```
ld = len(vocabulary) / len(all_words)
print("Lexical diversity", ld)
```

```
Lexical diversity 0.07406285585022564
```

6. Exploratory analysis - Frequency distribution

- The NLTK's `FreqDist` class is used to encode the **frequency distribution**, which count the number of times that each outcome occurs

```python
from nltk import FreqDist

frequency_dist = FreqDist(all_words)
frequency_dist.plot(30)
```



Fork me on GitHub

6. Exploratory analysis - Word cloud

- We can generate a **word cloud** to get an intuitive visualization of our vocabulary. For this, we can use `wordcloud` Python library (http://amueller.github.io/word_cloud/)

```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt

wcloud = WordCloud().generate_from_frequencies(frequency_dist)
plt.imshow(wcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

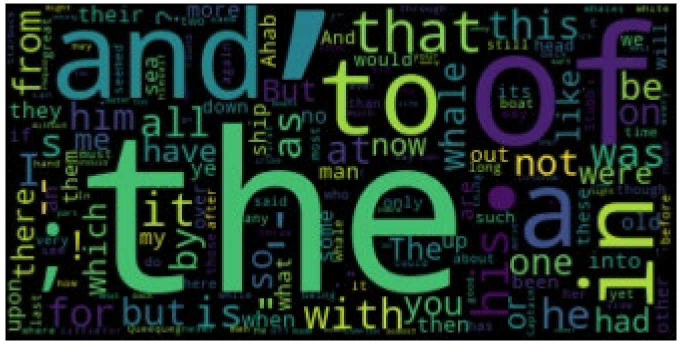


Table of contents

7. Web scraping

- **Web scraping** is a technique in which a computer program extracts information from semi-structured sources such as websites
- Web scraping is typically done following these steps:
  1. Send a **request** to a website (URL) and download the content (HTML)
  2. **Parse the HTML**. A web page is usually composed by different parts (header, navigation menu, footer, ads, etc.). For that reason, we can use a library like **Beautiful Soap**, which provides idiomatic ways of navigating and searching the HTML tree (called DOM, Document Object Model)
- In NLP, to build our own vocabulary from a webpage, we typically carry out some **text-preprocessing** (tokenization, stemming, stop word removal, and lemmatization) using the raw text obtained from web scraping

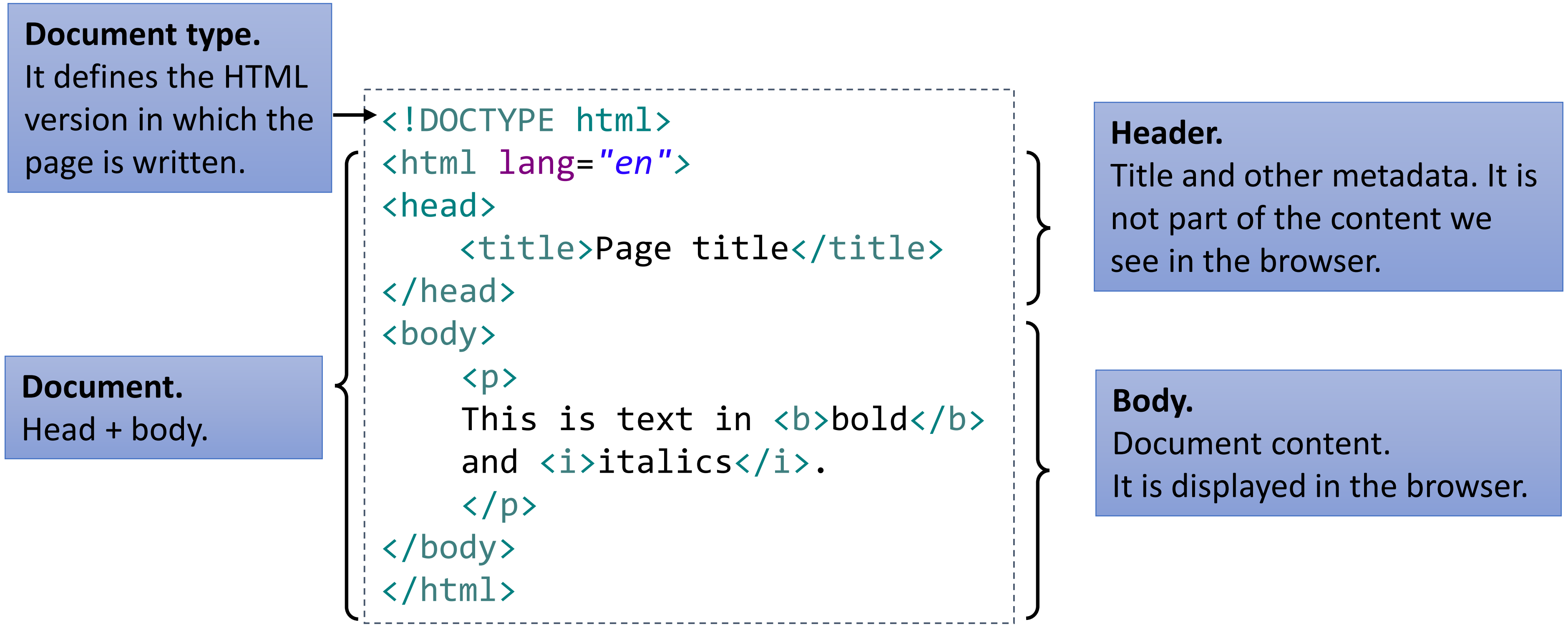
7. Web scraping - HTML

- Web pages are electronic documents written in **HTML** (HyperText Markup Language)
  - HTML is the standard markup language for documents designed to be displayed in a web browser (e.g. Chrome, Firefox, Edge, Safari, Opera)
  - HTML **tags** are like keywords which defines how web browser will format and display some content. Most tags come in pars, for example
    - `<html>` … and `</html>` : To declare the full content of a web page
    - `<h1>` … and `</h1>` : To declare a text header
  - Tags have attributes which define some feature of the tag, for example, `href` in tag `a` (used to create web links):

```
<a href="https://www.uc3m.es/">UC3M</a>
```

7. Web scraping - HTML

- The basic structure of a web page is as follows:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Page title</title>
</head>
<body>
    <p>
    This is text in <b>bold</b>
    and <i>italics</i>.
    </p>
</body>
</html>
```

Document type.
It defines the HTML version in which the page is written.

Document.
Head + body.

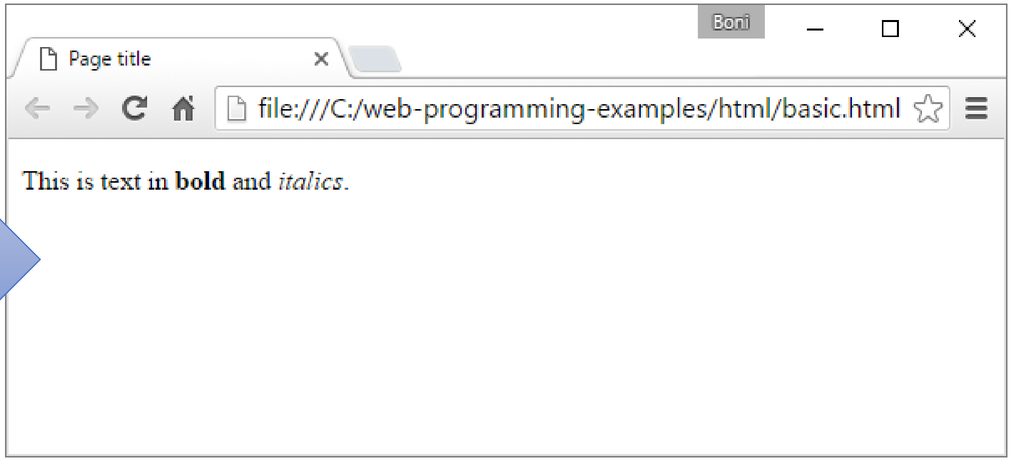
Header.
Title and other metadata. It is not part of the content we see in the browser.

Body.
Document content.
It is displayed in the browser.

7. Web scraping - HTML

- The basic structure of a web page is as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Page title</title>
</head>
<body>
    <p>
    This is text in <b>bold</b>
    and <i>italics</i>.
    </p>
</body>
</html>
```



This is text in **bold** and *italics*.

Fork me on GitHub

7. Web scraping - Beautiful Soap

- **Beautiful Soup** (https://www.crummy.com/software/BeautifulSoup/bs4/doc/) is a Python library for parsing HTML (and XML) documents
- The following example shows how to extract the text contained of a Wikipedia page

1. URL request

```
import requests

url = "https://en.wikipedia.org/wiki/Natural_language_processing"
page = requests.get(url)
print(page.content)
```

```
b'<!DOCTYPE html>\n<html class="client-nojs" lang="en"
dir="ltr">\n...\n</body></html>'
```

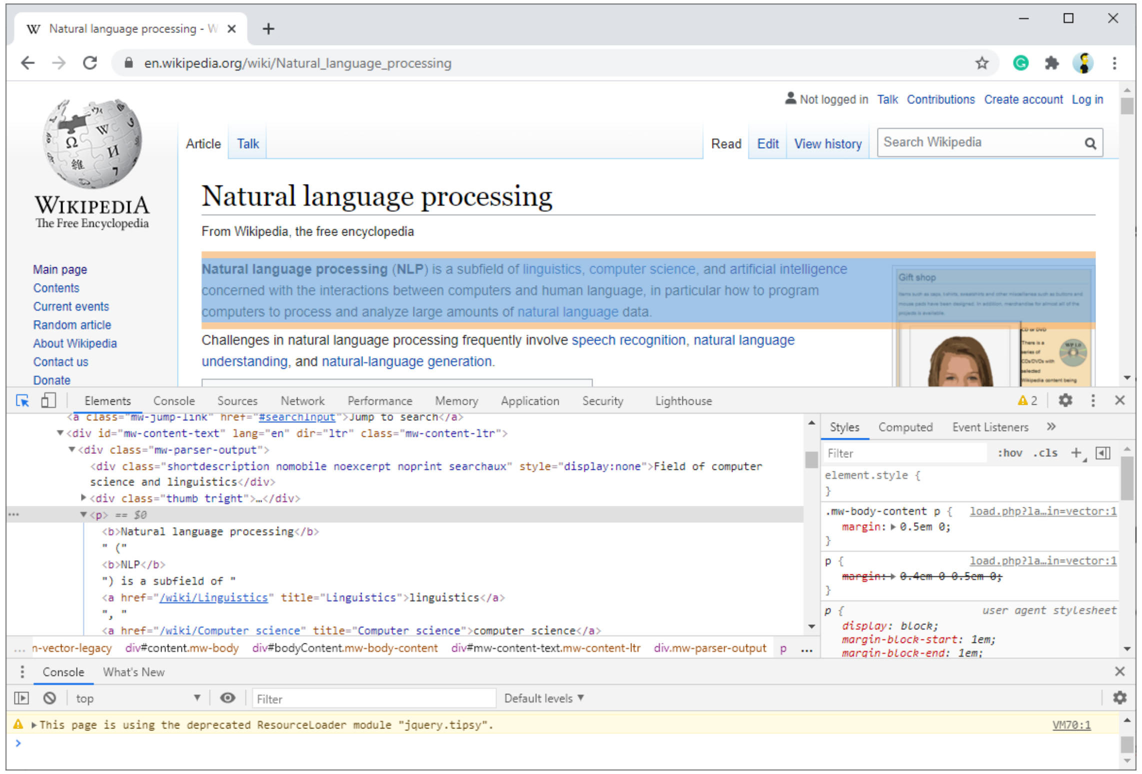
2. Parse with Beautiful Soap

```
from bs4 import BeautifulSoup

parsed_webpage = BeautifulSoup(page.content, "html.parser")
print(parsed_webpage.prettify())
```

```
b'<!DOCTYPE html>\n<html class="client-nojs" lang="en"
dir="ltr">\n...\n</body></html>'
```

7. Web scraping - Beautiful Soap

- We can inspect the HTML source code to identify the tags we can select with Beautiful Soap
  - For example, using **DevTools** in Chrome


Natural language processing - W
en.wikipedia.org/wiki/Natural_language_processing
Not logged in Talk Contributions Create account Log in
Article Talk Read Edit View history Search Wikipedia
WIKIPEDIA The Free Encyclopedia
Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate
Elements Console Sources Network Performance Memory Application Security Lighthouse
Styles Computed Event Listeners
Console What's New
top
Filter
Default levels

Natural language processing

From Wikipedia, the free encyclopedia

Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data.

Challenges in natural language processing frequently involve speech recognition, natural language understanding, and natural-language generation.

```
    <a class="mw-jump-link" href="#searchInput">Jump to search</a>
  ▼<div id="mw-content-text" lang="en" dir="ltr" class="mw-content-ltr">
    ▼<div class="mw-parser-output">
       <div class="shortdescription nomobile noexcerpt noprint searchaux" style="display:none">Field of computer
       science and linguistics</div>
      ▶<div class="thumb tright">…</div>
...   ▼<p> == $0
         <b>Natural language processing</b>
         " ("
         <b>NLP</b>
         ") is a subfield of "
         <a href="/wiki/Linguistics" title="Linguistics">linguistics</a>
         ", "
         <a href="/wiki/Computer_science" title="Computer science">computer science</a>
... n-vector-legacy  div#content.mw-body  div#bodyContent.mw-body-content  div#mw-content-text.mw-content-ltr  div.mw-parser-output  p ...
```

```
element.style {
}
.mw-body-content p {        load.php?la…in=vector:1
   margin: ▶ 0.5em 0;
}
p {                          load.php?la…in=vector:1
   ~~margin: ▶ 0.4em 0 0.5em 0;~~
}
p {                       user agent stylesheet
   display: block;
   margin-block-start: 1em;
   margin-block-end: 1em;
```

```
⚠ ▶This page is using the deprecated ResourceLoader module "jquery.tipsy".    VM70:1
>
```

Fork me on GitHub

7. Web scraping - Beautiful Soap

In this example, we select the HTML tags for headers (h1 to h3) and paragraphs (p)

We can use RegEx for filtering more the results (in this example, removing words like [edit] and \n from the results)

```python
import re

title = parsed_webpage.find("title")
print(title.text)

text_tags = parsed_webpage.findAll(["p", "h1", "h2", "h3"])
raw_text = [text_tag.text for text_tag in text_tags]
print(raw_text)

# If needed, we can use RegEx to filter more the results
raw_text = [re.sub(r"\[.*?\]|\n", "", text_tag.text) for text_tag in text_tags]
print(raw_text)
```

```
Natural language processing - Wikipedia
['Natural language processing', 'Natural language processing (NLP) is a subfield of
linguistics, computer science, and artificial intelligence concerned with the interactions
between computers and human language, in particular how to program computers to process and
analyze large amounts of natural language data.\n', 'Challenges in natural language processing
frequently involve speech recognition, natural language understanding, and natural-language
generation.\n', 'Contents', 'History[edit]', ...., '\nLanguages\n']
['Natural language processing', 'Natural language processing (NLP) is a subfield of
linguistics, computer science, and artificial intelligence concerned with the interactions
between computers and human language, in particular how to program computers to process and
analyze large amounts of natural language data.', 'Challenges in natural language processing
frequently involve speech recognition, natural language understanding, and natural-language
generation.', 'Contents', 'History', ..., 'Languages']
```

Table of contents

8. Takeaways

- A text **corpus** is a large body of text which can be used for several tasks (e.g. train and validate ML models) in NLP applications
- **Text preprocessing** is a common step in NLP pipelines, and it is made up by **tokenization** (chop raw text into small tokens), **stemming** (transform related word), **removing stop words** (such as a, an, he, or her, in English), and **lemmatization** (reduce morphological variation)
- Some exploratory indicators are: **vocabulary size**, **lexical diversity**, **frequency distribution**, or **word cloud**
- **Web scraping** is a technique to extract information programmatically from websites (e.g. using a Python parser like Beautiful Soap)