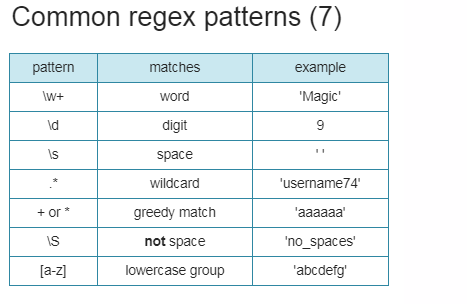
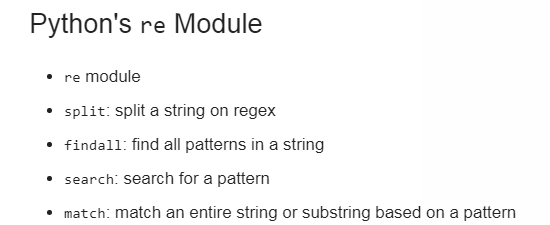
[**NLP in Python essentials**](https://livebook.manning.com/#!/book/machine-learning-with-tensorflow/chapter-2/v-9) **101**

Regex:



Lesson Notes

|  |  |  |
| --- | --- | --- |
|  | [abc…](https://regexone.com/lesson/introduction_abcs) | [*Letters*](https://regexone.com/lesson/introduction_abcs) |
|  | [123…](https://regexone.com/lesson/letters_and_digits) | [*Digits*](https://regexone.com/lesson/letters_and_digits) |
|  | [\d](https://regexone.com/lesson/letters_and_digits) | [*Any Digit*](https://regexone.com/lesson/letters_and_digits) |
|  | [\D](https://regexone.com/lesson/letters_and_digits) | [*Any Non-digit character*](https://regexone.com/lesson/letters_and_digits) |
|  | [.](https://regexone.com/lesson/wildcards_dot) | [*Any Character*](https://regexone.com/lesson/wildcards_dot) |
|  | [\.](https://regexone.com/lesson/wildcards_dot) | [*Period*](https://regexone.com/lesson/wildcards_dot) |
|  | [[abc]](https://regexone.com/lesson/matching_characters) | [*Only a, b, or c*](https://regexone.com/lesson/matching_characters) |
|  | [[^abc]](https://regexone.com/lesson/excluding_characters) | [*Not a, b, nor c*](https://regexone.com/lesson/excluding_characters) |
|  | [[a-z]](https://regexone.com/lesson/character_ranges) | [*Characters a to z*](https://regexone.com/lesson/character_ranges) |
|  | [[0-9]](https://regexone.com/lesson/character_ranges) | [*Numbers 0 to 9*](https://regexone.com/lesson/character_ranges) |
|  | [\w](https://regexone.com/lesson/character_ranges) | [*Any Alphanumeric character*](https://regexone.com/lesson/character_ranges) |
|  | [\W](https://regexone.com/lesson/character_ranges) | [*Any Non-alphanumeric character*](https://regexone.com/lesson/character_ranges) |
|  | [{m}](https://regexone.com/lesson/repeating_characters) | [*m Repetitions*](https://regexone.com/lesson/repeating_characters) |
|  | [{m,n}](https://regexone.com/lesson/repeating_characters) | [*m to n Repetitions*](https://regexone.com/lesson/repeating_characters) |
|  | [\*](https://regexone.com/lesson/kleene_operators) | [*Zero or more repetitions*](https://regexone.com/lesson/kleene_operators) |
|  | [+](https://regexone.com/lesson/kleene_operators) | [*One or more repetitions*](https://regexone.com/lesson/kleene_operators) |
|  | [?](https://regexone.com/lesson/optional_characters) | [*Optional character*](https://regexone.com/lesson/optional_characters) |
|  | [\s](https://regexone.com/lesson/whitespaces) | [*Any Whitespace*](https://regexone.com/lesson/whitespaces) |
|  | [\S](https://regexone.com/lesson/whitespaces) | [*Any Non-whitespace character*](https://regexone.com/lesson/whitespaces) |
|  | [^…$](https://regexone.com/lesson/line_beginning_end) | [*Starts and ends*](https://regexone.com/lesson/line_beginning_end) |
|  | [(…)](https://regexone.com/lesson/capturing_groups) | [*Capture Group*](https://regexone.com/lesson/capturing_groups) |
|  | [(a(bc))](https://regexone.com/lesson/nested_groups) | [*Capture Sub-group*](https://regexone.com/lesson/nested_groups) |
|  | [(.\*)](https://regexone.com/lesson/more_groups) | [*Capture all*](https://regexone.com/lesson/more_groups) |
|  | [(abc|def)](https://regexone.com/lesson/conditionals) | [*Matches abc or def*](https://regexone.com/lesson/conditionals) |

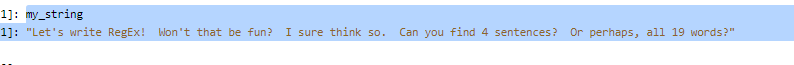


>>> my\_string = "Let's write RegEx!"

>>> re.findall(PATTERN, my\_string)

['Let', 's', 'write', 'RegEx']

PATTERN = r"\w+"

Example 2: 

# Import the regex module

import re

# Write a pattern to match sentence endings: sentence\_endings

sentence\_endings = r"[.?!]"

# Split my\_string on sentence endings and print the result

print(re.split(sentence\_endings, my\_string))

# Find all capitalized words in my\_string and print the result

capitalized\_words = r"[A-Z]\w+"

print(re.findall(capitalized\_words, my\_string))

# Split my\_string on spaces and print the result

spaces = r"\s+"

print(re.split(spaces, my\_string))

# Find all digits in my\_string and print the result

digits = r"\d+"

print(re.findall(digits, my\_string))

**Introduction to tokenization**

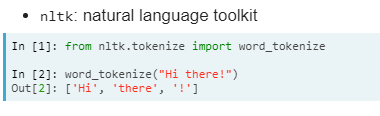
The process of turning a string or document into tokens is called tokenization

This is one step in preparing a text for NLP

There are several rules regarding tokenization and We can create your own tokenization using regularization.

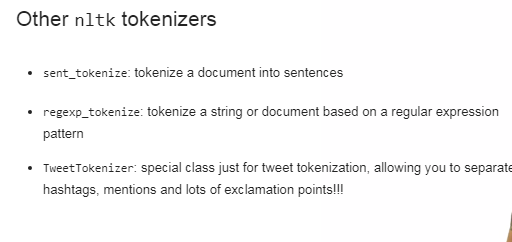
(separate punctinization, separating hash tag)

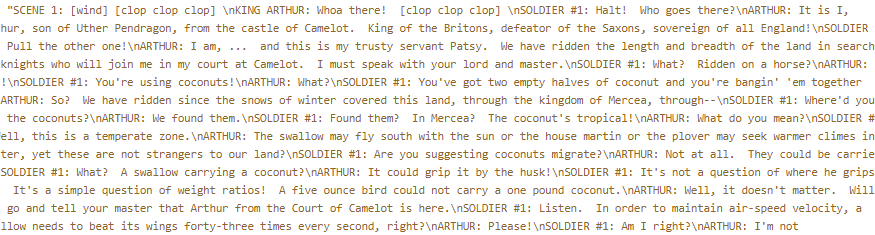
Nltk- library



Why tokenize

It can help us with simple processing tasks like mapping parts of speech, matching comman words and removing unwanted tokens



Example: 

*# Split scene\_one into sentences: sentences*

*sentences = sent\_tokenize(scene\_one)*

*# Use word\_tokenize to tokenize the fourth sentence: tokenized\_sent*

*tokenized\_sent = word\_tokenize(sentences[3]*

*# Make a set of unique tokens in the entire scene: unique\_tokens*

*unique\_tokens = set(word\_tokenize(scene\_one))*

*Exercise 2:*

*# Write a regular expression to search for anything in square brackets: pattern1*

*pattern1 = r"\[.\*\]"*

*r"[\w\s]+:"*  (ARTHUR:)

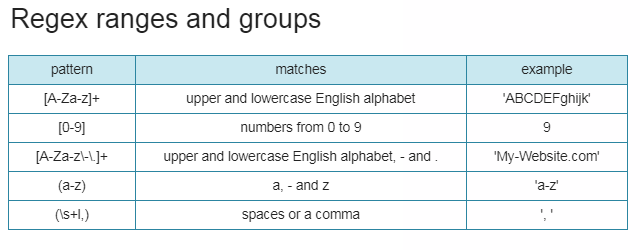
# Advanced tokenization with NLTK and regex

Advance: OR is represented by | character

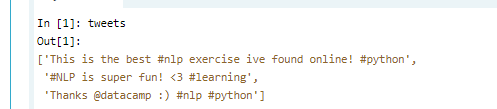
Using OR to match all digits and words in a group

=(‘(\d+|w+)’)

Then use, findall library to return all tokens



Example:



Find any hashtag- pattern1=r"#\w+"

regexp\_tokenize(tweets[0], pattern1)

# Write a pattern that matches both mentions and hashtags

pattern2 = r"([#|@]\w+)"

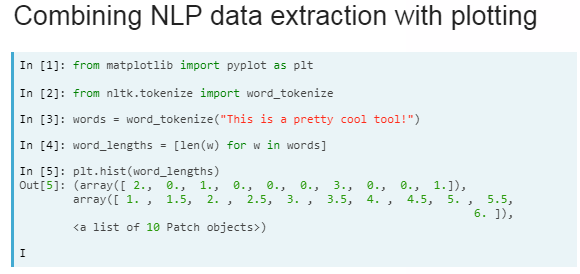
# Use the TweetTokenizer to tokenize all tweets into one list

tknzr = TweetTokenizer()

all\_tokens = [tknzr.tokenize(t) for t in tweets]

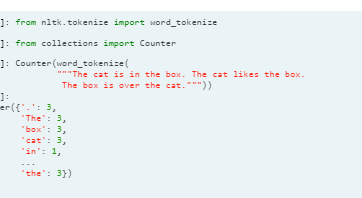
print(all\_tokens)

# Charting word length with NLTK



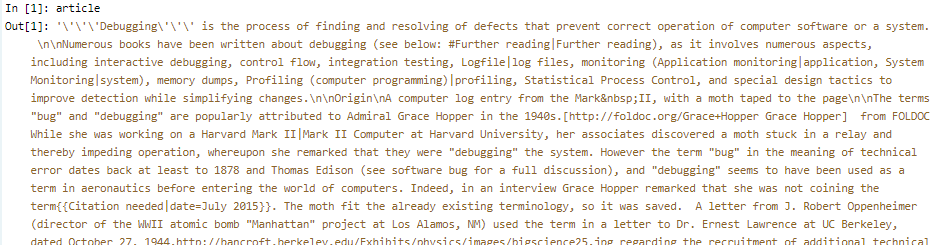
# Word counts with bag-of-words- basic method to find topics in a text

1. Create tokens
2. Count all the tokens





Building a Bag of words for article



**from collections import Counter**

**# Tokenize the article: tokens**

**tokens = word\_tokenize(article)**

**# Convert the tokens into lowercase: lower\_tokens**

**lower\_tokens = [t.lower() for t in tokens]**

**# Create a Counter with the lowercase tokens: bow\_simple**

**bow\_simple = Counter(lower\_tokens)**

**# Print the 10 most common tokens**

**print(bow\_simple.most\_common(10))**

It’s always good to preprocess your data

# Simple text preprocessing

Text preprocessing- tokenization, lowercasing are commonly used in NLP.

Lemmatization/Stemming, removing stop words, punctuation, other unwanted tokens.

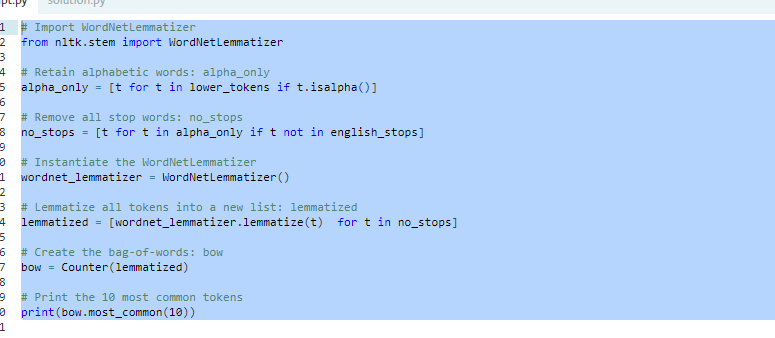
From stlk.corpus import stopwords

User lower case tokenization along with w.isalpha- return true if string only has alphabetical strings.

Tokens= **[w for w in word\_tokenize(text.lower()) if w.isalpha()]**

1. Remove stopwords- [t for t in tokens if t not in stopwords.words(‘english’)
2. Counter(no\_stops).most\_common(2)

Example

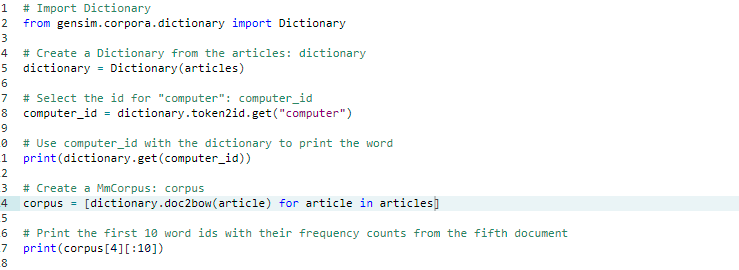


Gensism- Open source NLP library- new BOW , can be saved, updated and reused.

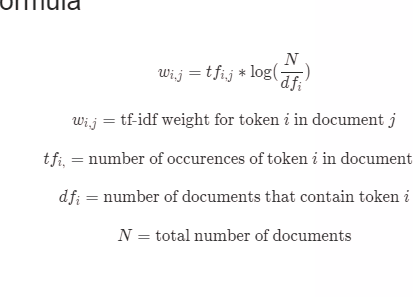
1. Import dictionary

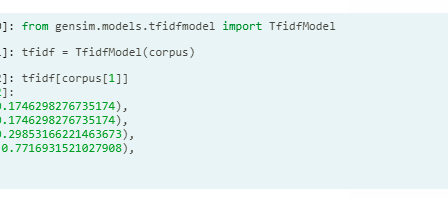


1. Tokenize the document
2. Pass the tokenized doc to Dictionary
3. Dictionary.token2id(use this)
4. Create genism corpus- [dictionary.doc2bow(doc) for doc in tokenized\_dic]

Example”” 

Td-Idf



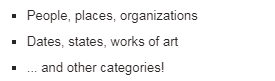


This helps us determine which topics have high weight

Name entity recognization

You'll learn how to identify the who, what and where of your texts using pre-trained models on English and non-English text. You'll also learn how to use some new libraries - polyglot and spaCy - to add to your NLP toolbox

1. Indentify following entities



1.import nltk

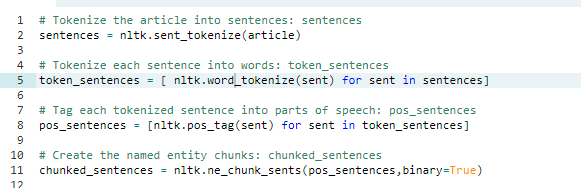
2. sentence=”sdasdasdas”

3. tokenized=nltk.word\_tokenize(sentence)

4. tagged=Nltk.pos\_tag(tokenized)

5.nltk.ne\_chunk(tagged)

Live Example:



# Introduction to SpaCy

# Focus on NLP pipelines and generate models.

# 

# Import spacy

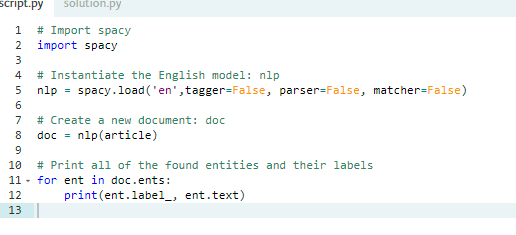
# nlp= spacy.load(‘en’)

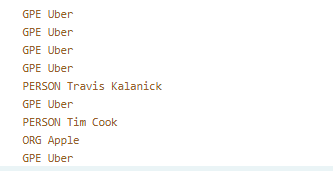
# load a document- doc= nlp(sdasdasd)

# doc.ents

# doc.ents[0], doc.ents[0].label\_

# 





# Multilingual NER with polyglot

NLP library uses word vectors

Main benefit- wide variety of languages it support

. Can be used to translate from English to over 130 language

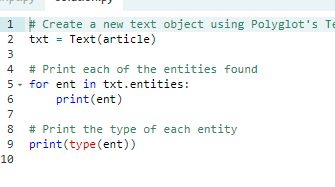
Can be used for NER for Spanish

from polyglot.text import Text

text=”””asdasdasdads”

ptext= Text(text)

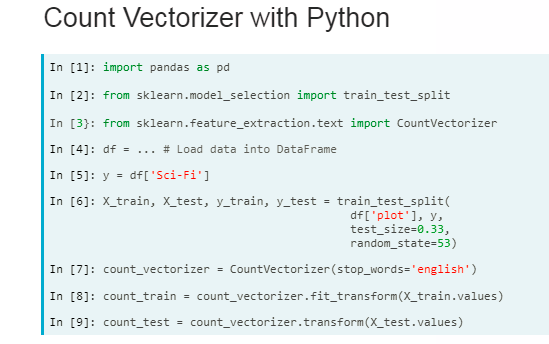
ptext.entities



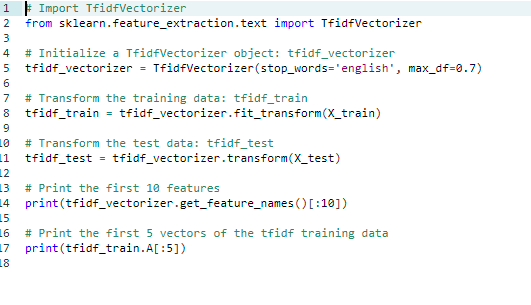
# Building word count vectors with scikit-learn

Transform words into BOW

**Count vectorization for text classification**



# TfidfVectorizer for text classification



# Training and testing a classification model with scikit-learn

Naiye bayes classifier is generally used for testing NLP classification problem

