# Data Cleaning & Analysis

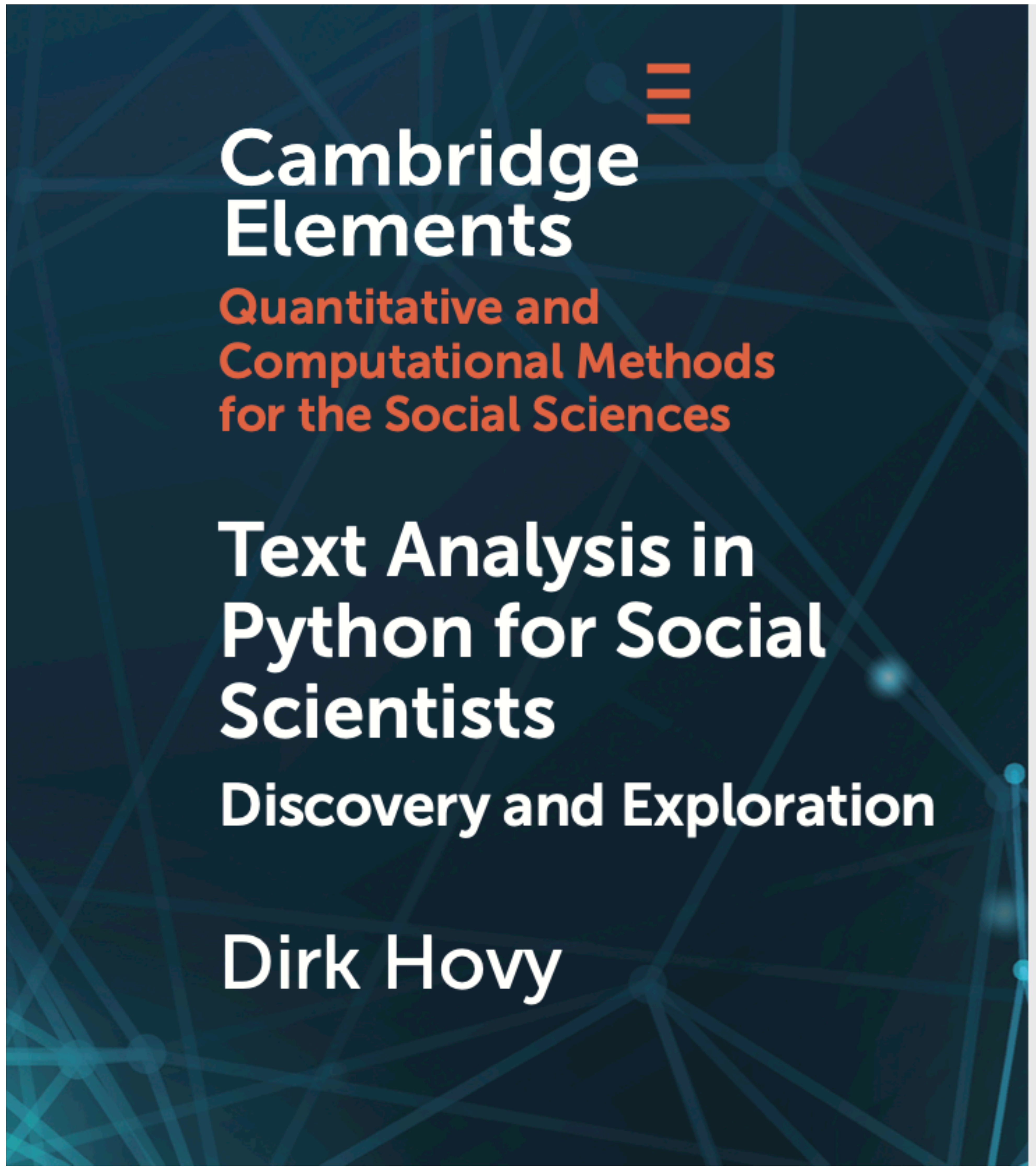
*text analysis*

IMT 547 - Social Media Data Mining and Analysis

4-Feb-2021 (Week 5, Day 10)

# Today's Topics

- Text Analytics
  - NLTK
  - Tokenization
  - Stemming, lemmatization
  - Regular expression
- Lab: text analytics + data cleaning
- Project Groups


Cambridge Elements
Quantitative and Computational Methods for the Social Sciences
Text Analysis in Python for Social Scientists
Discovery and Exploration
Dirk Hovy

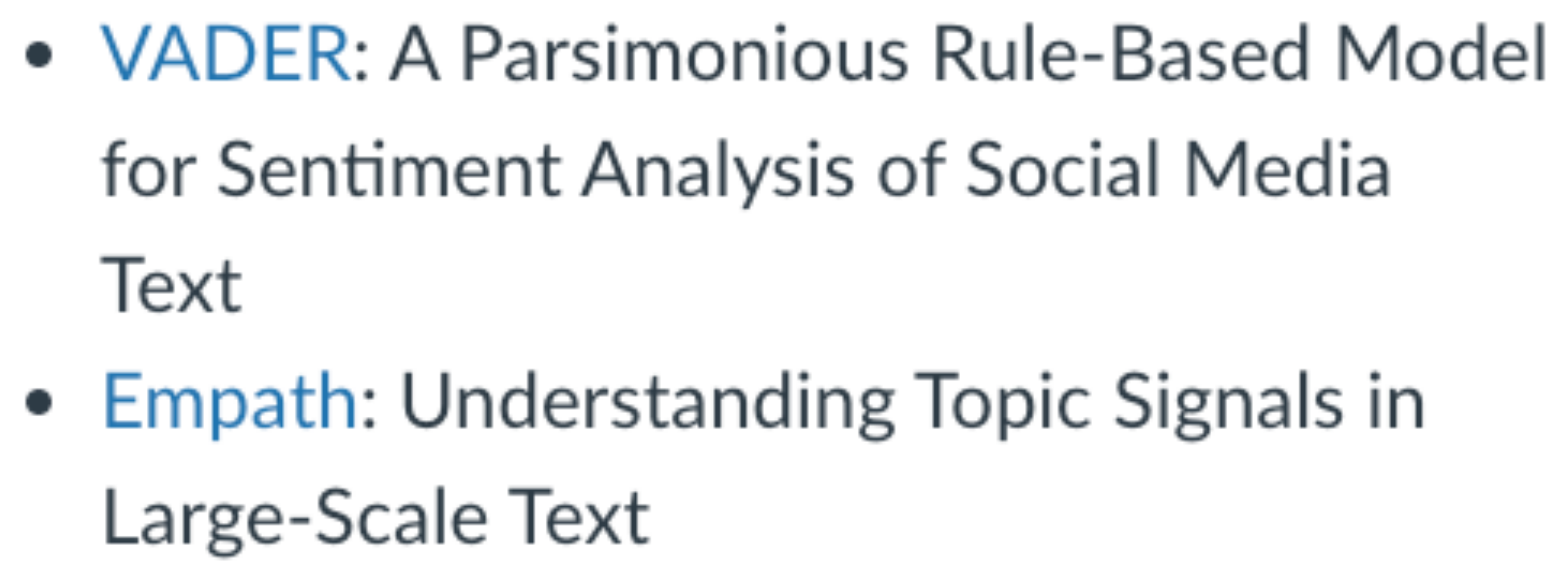
## Project pitch deadline extended

**Feb 8th Monday 9AM**

**NO LATE DAYS!**

| Office hour time | Project Groups |
|---|---|
| 3:00 to 3:10 | 1, 2 |
| 3:10 - 3:20 | 3, 4 |
| 3:20 - 3:30 | 5, 6 |
| 3:30 - 3:40 | 7, 8 |
| 3:40 - 3:50 | 9, 10 |
| 3:50 - 4:00 | 11 |

# Next Class

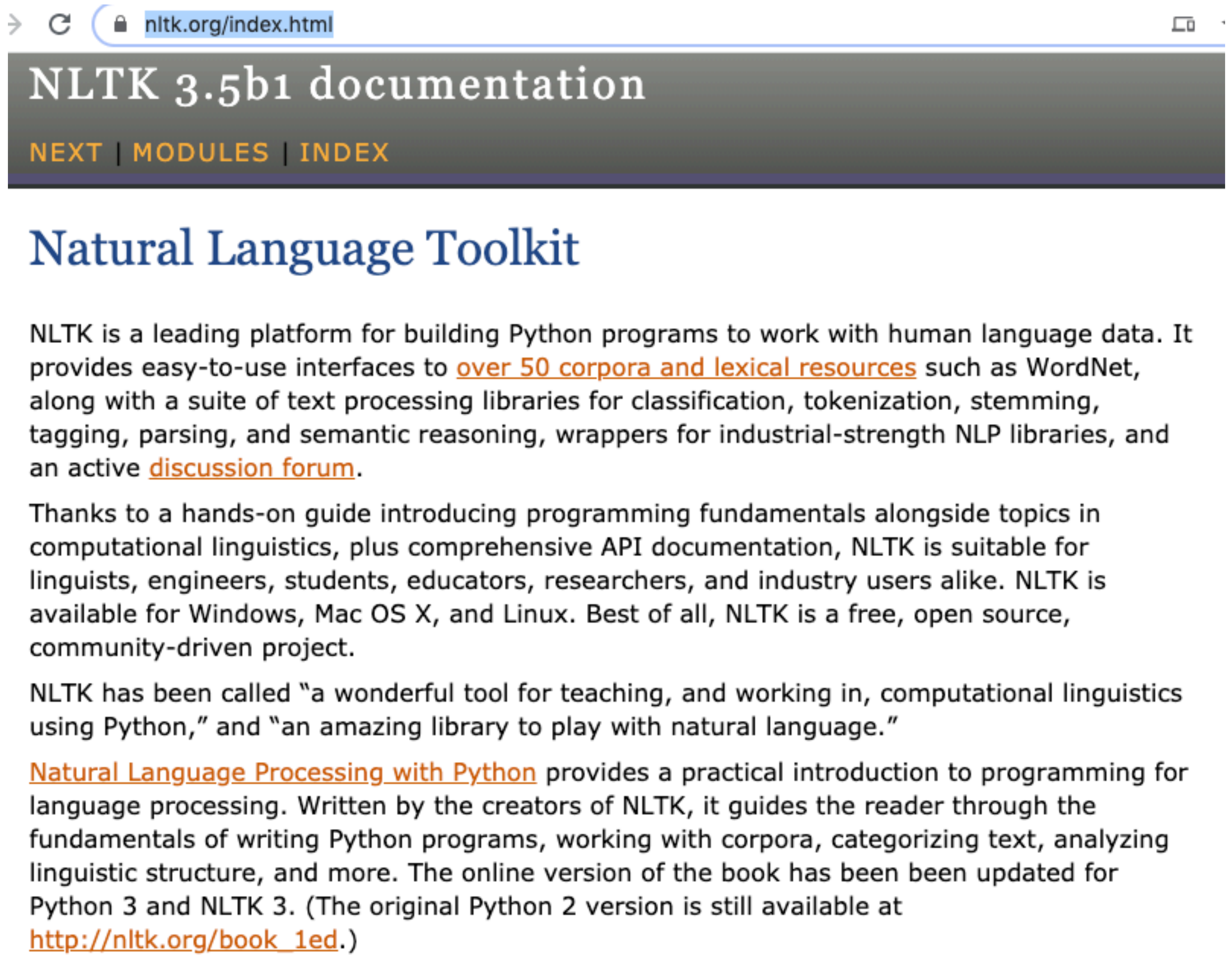


- VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text
- Empath: Understanding Topic Signals in Large-Scale Text

No formal reading reflection due, but please read before class and come prepared to participate

## Natural Language Toolkit NLTK

https://www.nltk.org/index.html



nltk.org/index.html

NLTK 3.5b1 documentation

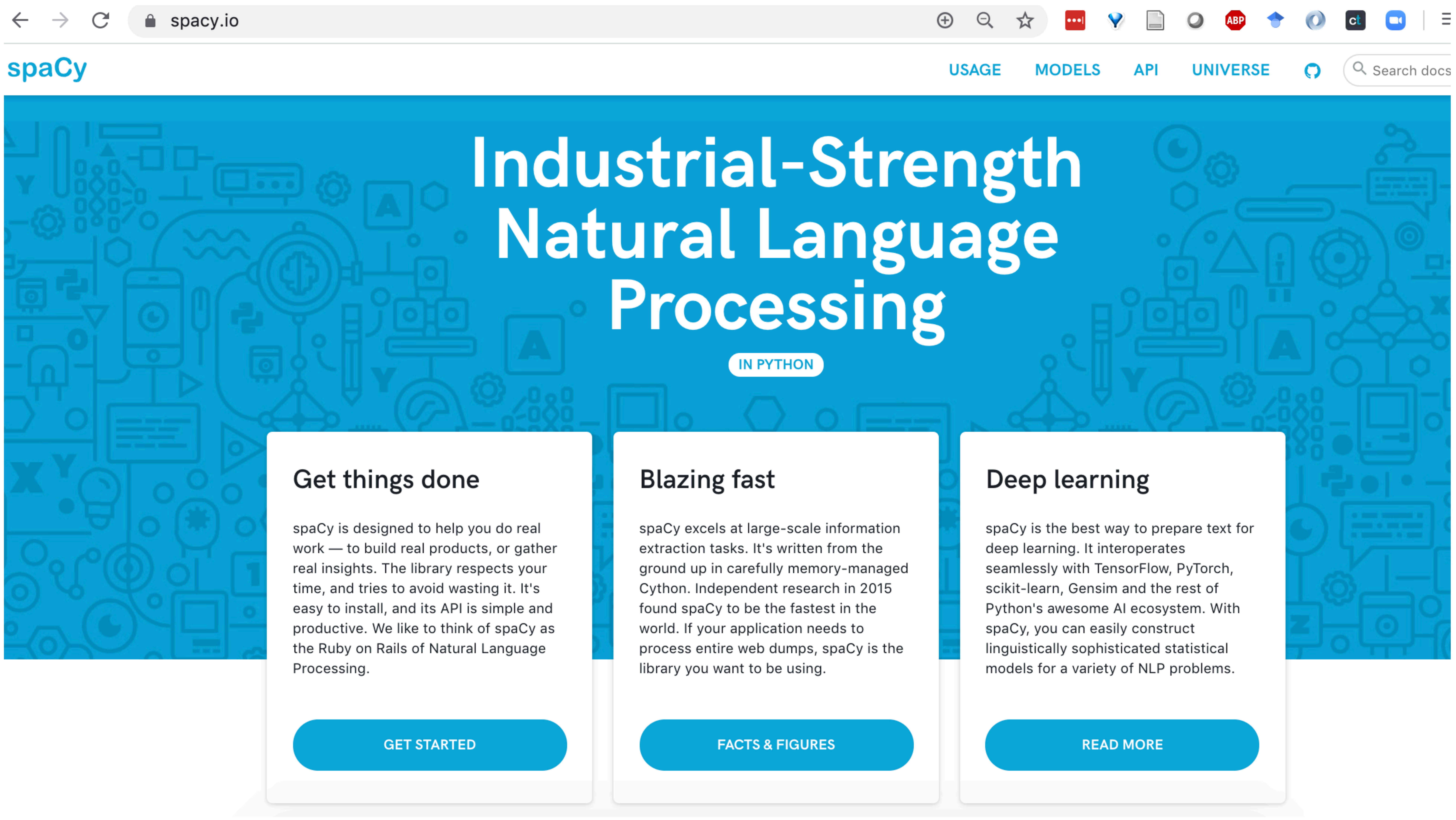
Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called "a wonderful tool for teaching, and working in, computational linguistics using Python," and "an amazing library to play with natural language."

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more. The online version of the book has been been updated for Python 3 and NLTK 3. (The original Python 2 version is still available at http://nltk.org/book_1ed.)



spacy.io

spaCy

USAGE MODELS API UNIVERSE

Search docs

# Industrial-Strength Natural Language Processing

IN PYTHON

## Get things done

spaCy is designed to help you do real work — to build real products, or gather real insights. The library respects your time, and tries to avoid wasting it. It's easy to install, and its API is simple and productive. We like to think of spaCy as the Ruby on Rails of Natural Language Processing.

GET STARTED

## Blazing fast

spaCy excels at large-scale information extraction tasks. It's written from the ground up in carefully memory-managed Cython. Independent research in 2015 found spaCy to be the fastest in the world. If your application needs to process entire web dumps, spaCy is the library you want to be using.

FACTS & FIGURES

## Deep learning

spaCy is the best way to prepare text for deep learning. It interoperates seamlessly with TensorFlow, PyTorch, scikit-learn, Gensim and the rest of Python's awesome AI ecosystem. With spaCy, you can easily construct linguistically sophisticated statistical models for a variety of NLP problems.

READ MORE

## Unit of Analysis: Document & Corpus (**Text Analytics Terminologies**)

When we work with text, the unit we are interested in depends strongly on the problem we are investigating.

**Document**: **unit of analysis**. When working with social media data or other online data, it can mean many things. Examples:

- Single message
- Entire history of user's posts
- Individual sentence
- A news article

**One document always represents one observation in our data**

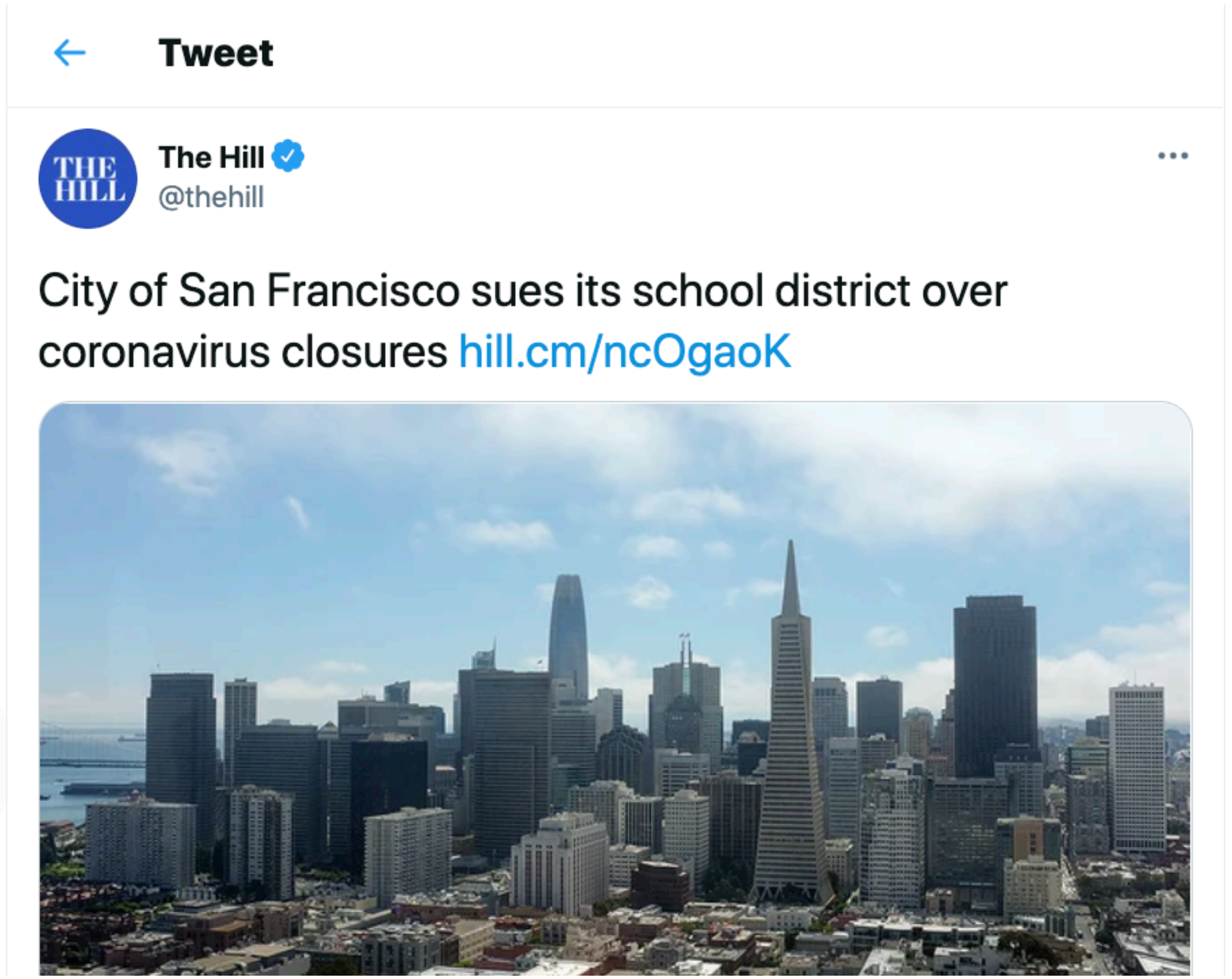
**Corpus**: entire collection of documents

**Vocabulary (V)**: set of all the unique terms in our data

**Token (V)**: Each element in the vocabulary. e.g.: "hello world" two tokens: *hello, world*

## How many words in this tweet?

Definition of word


Tweet

The Hill
@thehill

City of San Francisco sues its school district over coronavirus closures hill.cm/ncOgaoK

**Tokenization**: separating each symbol by white space

**Sentence splitting**: separating a document into sentences

# How many words in this tweet?

Definition of word

They had many cats and one their cat died.

**Lemma**: words that have same stem, same rough word sense.

- cat and cats
- they and their

Wordform: the full surface form

- Where you consider cat and cats as different word forms

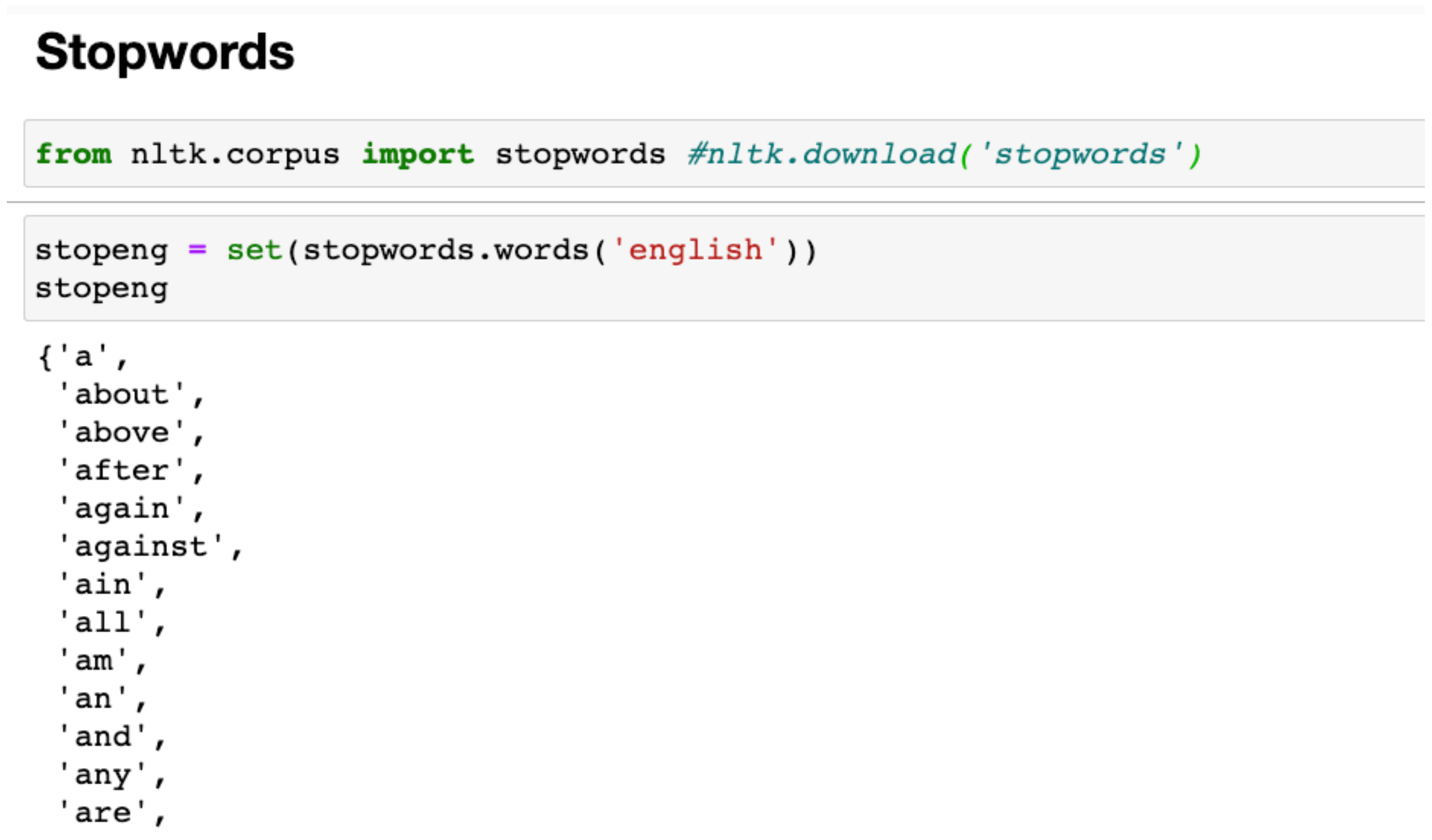
## Stemming

Stemming in NLP is the process of removing prefixes and suffixes from words so that they are reduced to simpler forms which are called stems.

```
from nltk import PorterStemmer
sentence = ["This","sentence","was","transformed", "using",
"Porter", "Stemmer"]
porterStemmer = PorterStemmer()
print (" ".join([porterStemmer.stem(word) for word in
sentence]))
# Prints "thi sentenc wa transform use porter stemmer"
```

# Stopwords

Set of ignorable function words. *the*, *a*,



### Stopwords

```
from nltk.corpus import stopwords #nltk.download('stopwords')
```

```
stopeng = set(stopwords.words('english'))
stopeng
```

```
{'a',
 'about',
 'above',
 'after',
 'again',
 'against',
 'ain',
 'all',
 'am',
 'an',
 'and',
 'any',
 'are',
```

## N-grams

Unigrams: looking at one word

Bigrams: looking at two words together. *e.g:* “New York”

Trigrams: 3 word combinations. e.g.: “New York bagels”

```python
from nltk import ngrams
bigrams = [gram for gram in ngrams(tokens[0], 2)]
```

**Code 7** Extracting bigrams from a sentence.

This gives us

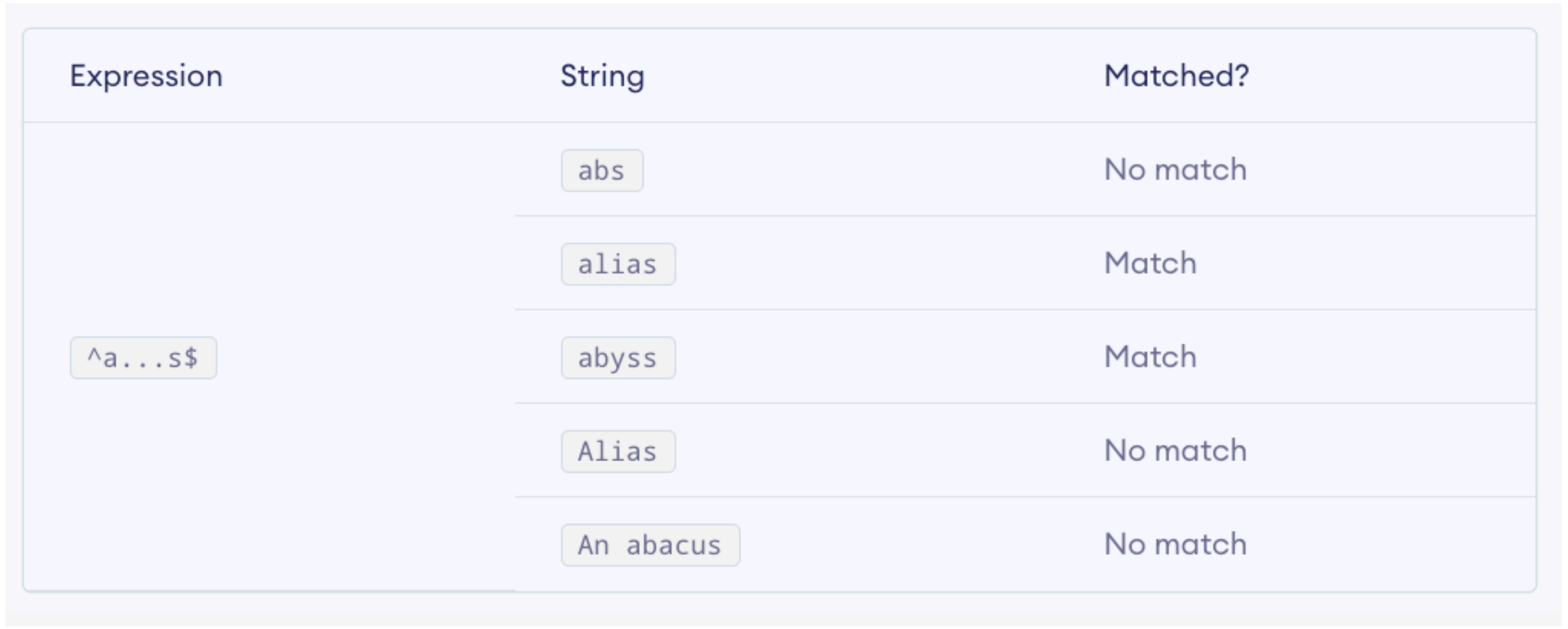
```
[('I', "'ve"),
 ("'ve", 'been'),
 ('been', '2'),
 ('2', 'times'),
 ('times', 'to'),
 ('to', 'New'),
 ('New', 'York'),
 ('York', 'in'),
 ('in', '2011'),
```

## Regular expression

**A formal language for specifying text strings**

A Regular Expression (RegEx) is a sequence of characters that defines a search pattern.

`For example: ^a…s$ :` any five letter string starting with **a** and ending with **s**.

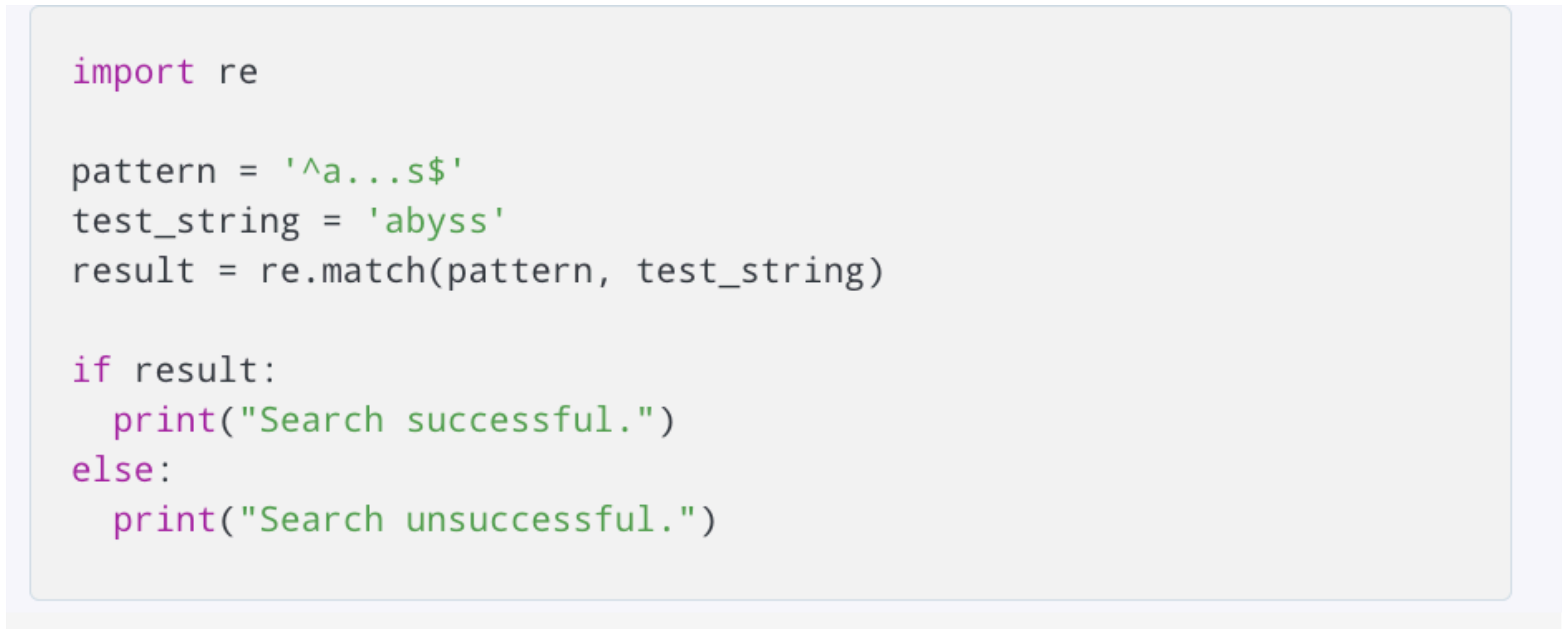


| Expression | String | Matched? |
|---|---|---|
| `^a...s$` | `abs` | No match |
| | `alias` | Match |
| | `abyss` | Match |
| | `Alias` | No match |
| | `An abacus` | No match |

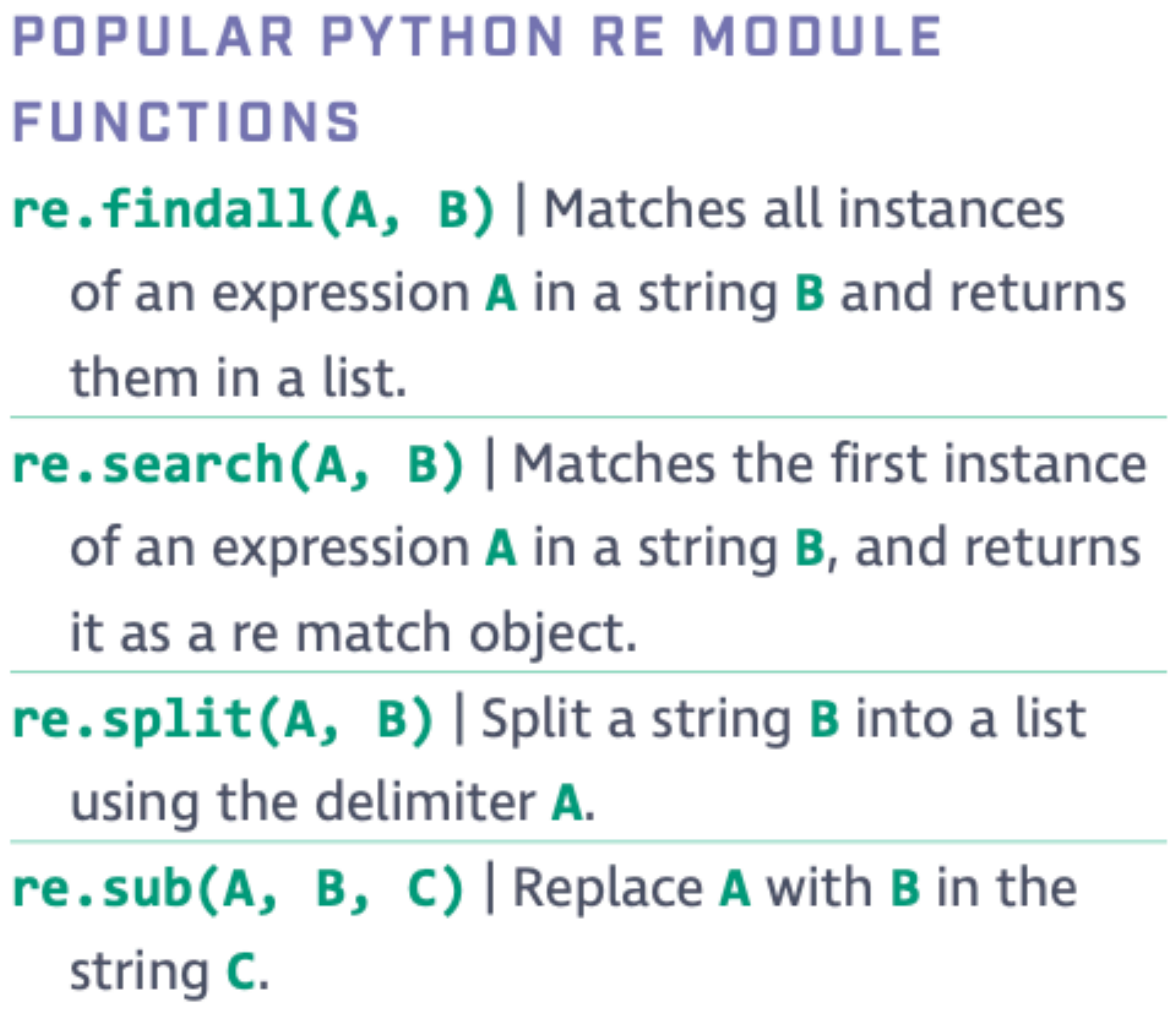
# Regular expression

A Regular Expression (RegEx) is a sequence of characters that defines a search pattern.

Python has module named `re` to work with regular expressions



```python
import re

pattern = '^a...s$'
test_string = 'abyss'
result = re.match(pattern, test_string)

if result:
  print("Search successful.")
else:
  print("Search unsuccessful.")
```



**POPULAR PYTHON RE MODULE FUNCTIONS**

**`re.findall(A, B)`** | Matches all instances of an expression **A** in a string **B** and returns them in a list.

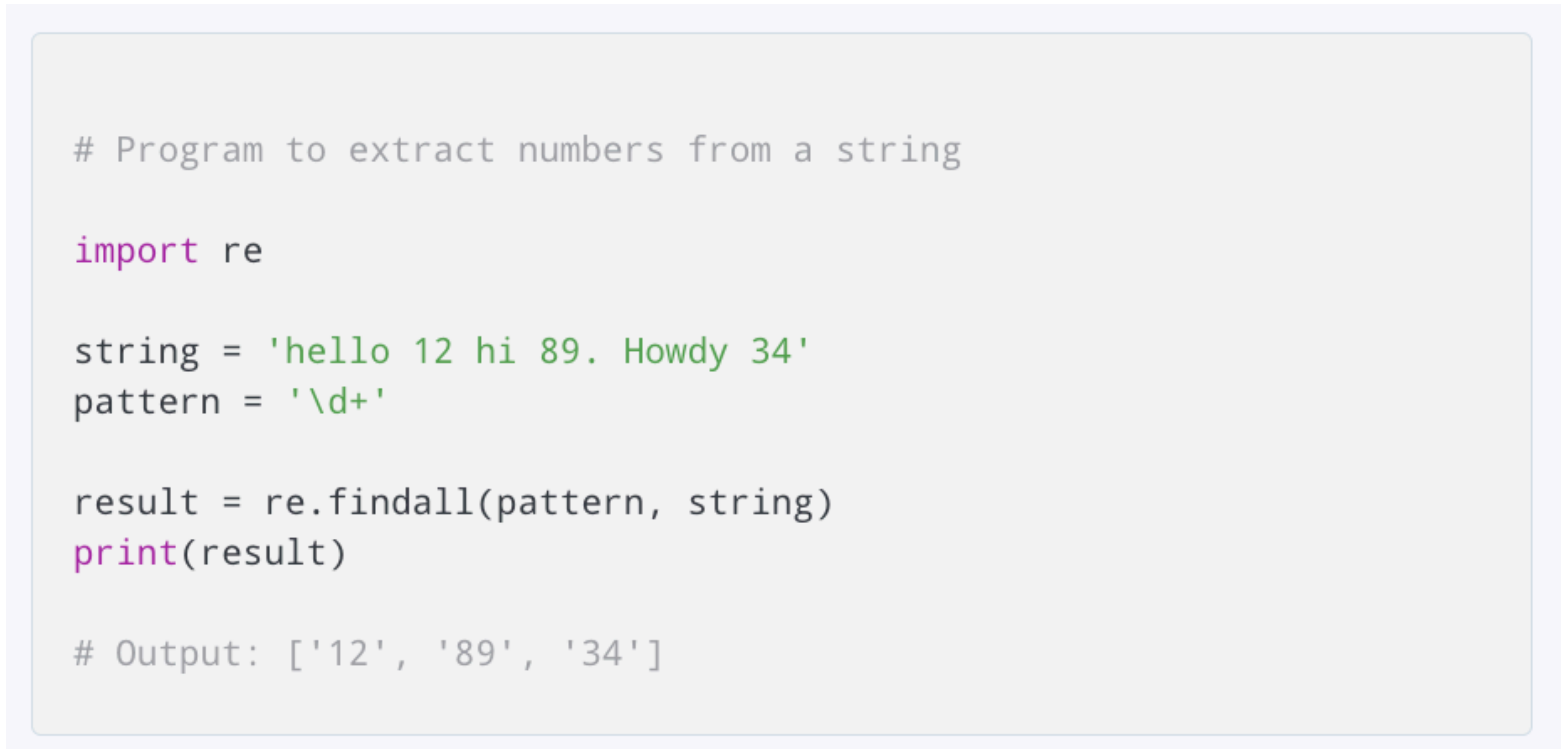
**`re.search(A, B)`** | Matches the first instance of an expression **A** in a string **B**, and returns it as a re match object.

**`re.split(A, B)`** | Split a string **B** into a list using the delimiter **A**.

**`re.sub(A, B, C)`** | Replace **A** with **B** in the string **C**.

# Regular expression

Commonly used re functions



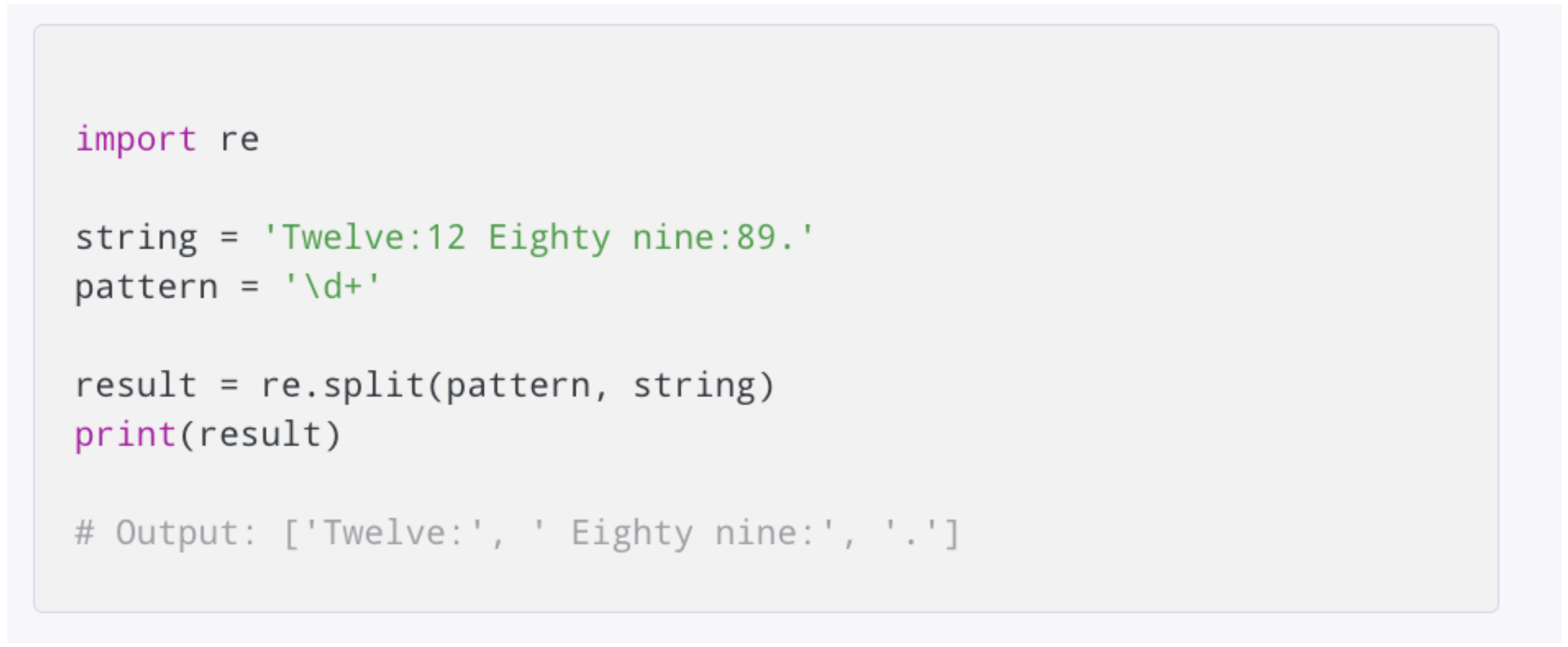
```
# Program to extract numbers from a string

import re

string = 'hello 12 hi 89. Howdy 34'
pattern = '\d+'

result = re.findall(pattern, string)
print(result)

# Output: ['12', '89', '34']
```

The `re.findall()` method returns a list of strings containing all matches.

If the pattern is not found, re.findall() returns an empty list.

# Regular expression

Commonly used re functions



```python
import re

string = 'Twelve:12 Eighty nine:89.'
pattern = '\d+'

result = re.split(pattern, string)
print(result)

# Output: ['Twelve:', ' Eighty nine:', '.']
```

The `re.split` method splits the string where there is a match and returns a list of strings where the splits have occurred.

If the pattern is not found, `re.split()` returns a list containing the original string.

# Regular expression

Commonly used re functions

```
# Program to remove all whitespaces
import re

# multiline string
string = 'abc 12\
de 23 \n f45 6'

# matches all whitespace characters
pattern = '\s+'

# empty string
replace = ''

new_string = re.sub(pattern, replace, string)
print(new_string)

# Output: abc12de23f456
```

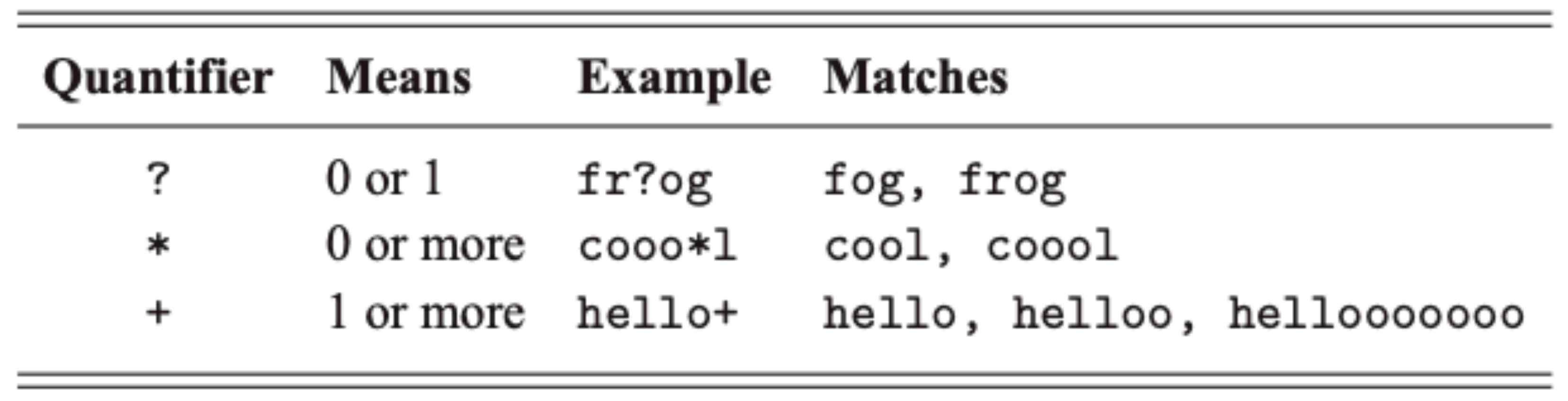
The `re.sub` returns a string where matched occurrences are replaced with the content of replace variable.

```
re.sub(pattern, replace, string)
```

# Regular expression: Pattern matching

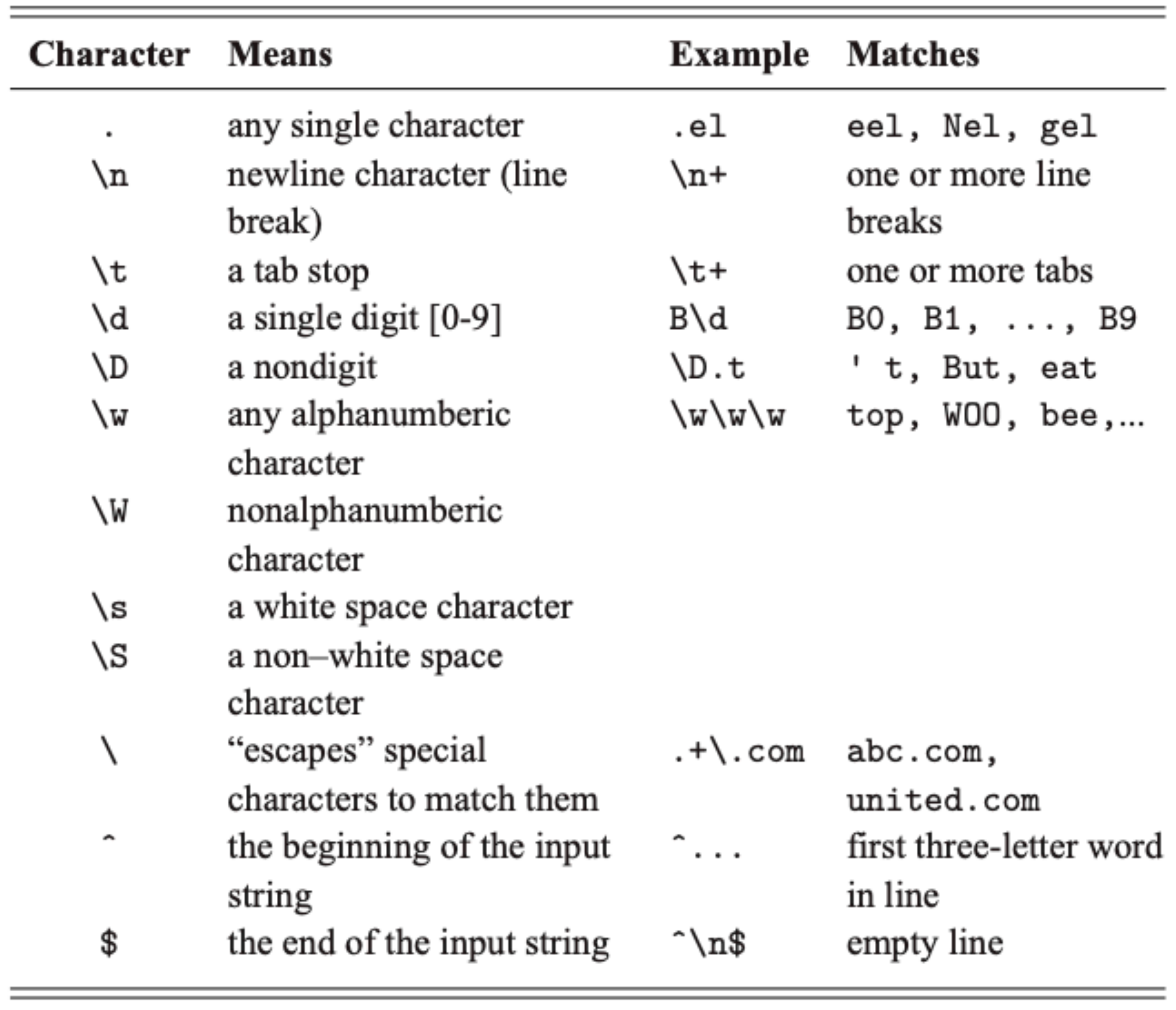
**Forming the pattern for re to match**

**Table 2** Examples of quantifiers



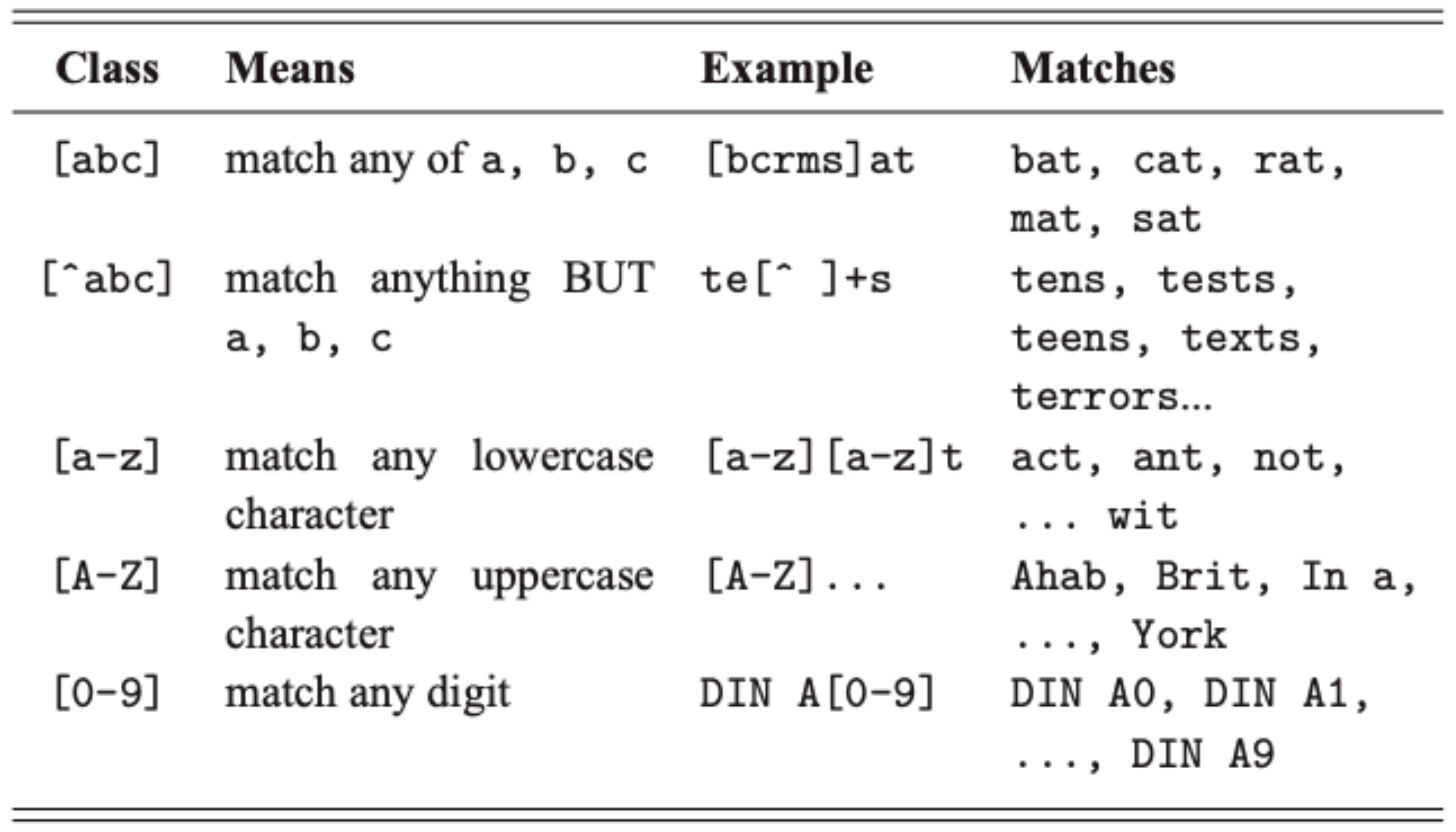
| Quantifier | Means | Example | Matches |
|---|---|---|---|
| ? | 0 or 1 | fr?og | fog, frog |
| * | 0 or more | cooo*l | cool, coool |
| + | 1 or more | hello+ | hello, helloo, hellooooooo |

**Table 3** Examples of special characters



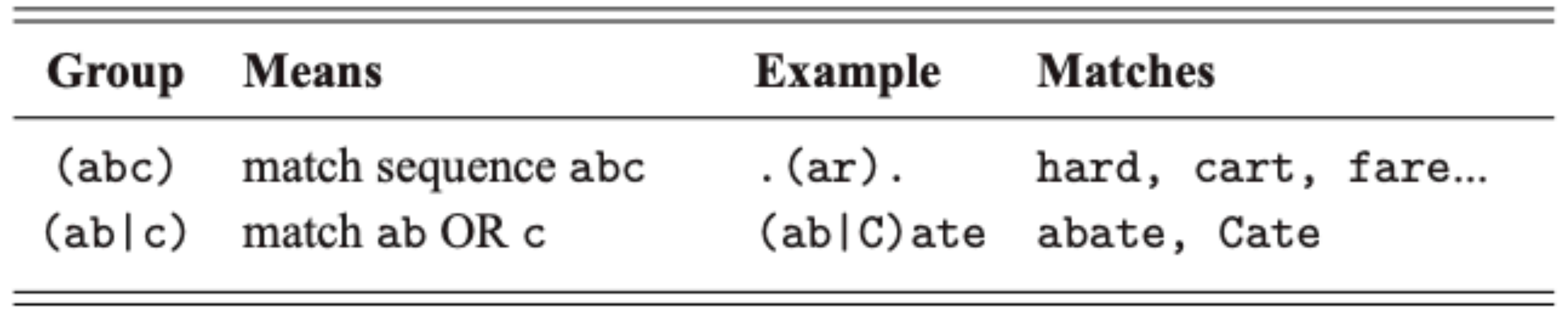
| Character | Means | Example | Matches |
|---|---|---|---|
| . | any single character | .el | eel, Nel, gel |
| \n | newline character (line break) | \n+ | one or more line breaks |
| \t | a tab stop | \t+ | one or more tabs |
| \d | a single digit [0-9] | B\d | B0, B1, ..., B9 |
| \D | a nondigit | \D.t | ' t, But, eat |
| \w | any alphanumberic character | \w\w\w | top, WOO, bee,... |
| \W | nonalphanumberic character | | |
| \s | a white space character | | |
| \S | a non–white space character | | |
| \ | "escapes" special characters to match them | .+\.com | abc.com, united.com |
| ^ | the beginning of the input string | ^... | first three-letter word in line |
| $ | the end of the input string | ^\n$ | empty line |

**Table 4** Examples of character classes



| Class | Means | Example | Matches |
|---|---|---|---|
| [abc] | match any of a, b, c | [bcrms]at | bat, cat, rat, mat, sat |
| [^abc] | match anything BUT a, b, c | te[^ ]+s | tens, tests, teens, texts, terrors... |
| [a-z] | match any lowercase character | [a-z][a-z]t | act, ant, not, ... wit |
| [A-Z] | match any uppercase character | [A-Z]... | Ahab, Brit, In a, ..., York |
| [0-9] | match any digit | DIN A[0-9] | DIN A0, DIN A1, ..., DIN A9 |

**Table 5** Examples of groups



| Group | Means | Example | Matches |
|---|---|---|---|
| (abc) | match sequence abc | .(ar). | hard, cart, fare... |
| (ab\|c) | match ab OR c | (ab\|C)ate | abate, Cate |

# Regular expression

Many online resources on regex

RegEx Testing: https://www.regextester.com

https://www.regexpal.com/

**REGEX cheatsheet**: https://www.dataquest.io/wp-content/uploads/2019/03/python-regular-expressions-cheat-sheet.pdf

## Break

Be back at 9 :20

## Lab

## Project Group meetup in class