Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions From test to large-scale
○○○○○○○○   ○○○   ○○   ○○   ○○○○
○○   ○○○   ○○○○○○   ○○○○○○
                         ○○○○○○   ○○○○○○○○○
                         ○○○○
                         ○○○

Teaching the Teacher
Day 4 - Morning » From text to features:
Natural Language Processing«

Anne Kroon

a.c.kroon@uva.nl
@annekroon

July 1, 2021

Afdeling Communicatiewetenschap
Universiteit van Amsterdam

1

Bottom-up vs. top-down   Natural Language Processing   Advanced NLP   Regular expressions   From test to large-scale
00000000   000   00   00   0000
00   000   000000   000000
                         000000   000000000
                         0000
                         000

Today

Bottom-up vs. top-down

  Approaches to working with text

Natural Language Processing

  Better tokenization

  Stopword and punctuation removal

  Stemming and lemmatization

  ngrams

Advanced NLP

  Parsing sentences

ACA using regular expressions

  What is a regexp?

Bottom-up vs. top-down

Automated content analysis can be either **bottom-up** (inductive, explorative, pattern recognition, . . .) or **top-down** (deductive, based on a-priori developed rules, . . .). Or in between.

The ACA toolbox

	Methodological approach		
	Counting and Dictionary	Supervised Machine Learning	Unsupervised Machine Learning
Typical research interests and content features	visibility analysis sentiment analysis subjectivity analysis	frames topics gender bias	frames topics
Common statistical procedures	string comparisons counting	support vector machines naive Bayes	principal component analysis cluster analysis latent dirichlet allocation semantic network analysis

deductive ⟶ inductive

Boumans and Trilling, 2016

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions From test to large-scale
○○○●○○○○   ○○○   ○○   ○○   ○○○○
○○   ○○○   ○○○○○○   ○○○○○○
                          ○○○○○○   ○○○○○○○○○
                          ○○○○
                          ○○○

Bottom-up vs. top-down

Bottom-up

- Count most frequently occurring words
- Maybe better: Count combinations of words $\Rightarrow$ Which words co-occur together?

We *don't* specify what to look for in advance

Top-down

- Count frequencies of pre-defined words
- Maybe better: patterns instead of words

We *do* specify what to look for in advance

Bottom-up vs. top-down   Natural Language Processing   Advanced NLP   Regular expressions   From test to large-scale
○○○○●○○○   ○○○   ○○   ○○   ○○○○
○○   ○○○   ○○○○○○   ○○○○○○
                         ○○○○○○   ○○○○○○○○○
                         ○○○○
                         ○○○

A simple bottom-up approach

```python
1  from collections import Counter
2
3  texts = ["I really really really love him, I do", "I hate him"]
4
5  for t in texts:
6      print(Counter(t.split()).most_common(3))
```

```
1  [('really', 3), ('I', 2), ('love', 1)]
2  [('I', 1), ('hate', 1), ('him', 1)]
```

5

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions From test to large-scale
0000000   000   00   00   0000
00   000   000000   000000
                         000000   000000000
                         0000
                         000

A simple top-down approach

```
1  texts = ["I really really really love him, I do", "I hate him"]
2  features = ['really', 'love', 'hate']
3
4  for t in texts:
5      print(f"\nAnalyzing '{t}':")
6      for f in features:
7          print(f"{f} occurs {t.count(f)} times")
```

```
1  Analyzing 'I really really really love him, I do':
2  really occurs 3 times
3  love occurs 1 times
4  hate occurs 0 times
5
6  Analyzing 'I hate him':
7  really occurs 0 times
8  love occurs 0 times
9  hate occurs 1 times
```

6

When would you use which approach?

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions From test to large-scale
ooooooo●
oo
ooo
ooo
oooooo
oooo
ooo
oo
oooooo
oo
oooooo
ooooooooo
oooo

Some considerations

- Both can have a place in your workflow (e.g., bottom-up as first exploratory step)

- You have a clear theoretical expectation? Bottom-up makes little sense.

- But in any case: you need to transform your text into something "countable".

Bottom-up vs. top-down

Approaches to working with text

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions From test to large-scale
○○○○○○○○   ○○○   ○○   ○○   ○○○○
○●   ○○○   ○○○○○○   ○○○○○○
                         ○○○○○○   ○○○○○○○○○
                         ○○○○
                         ○○○

The toolbox

Slicing

mystring[2:5] to get the characters with indices 2,3,4

String methods

- .lower() returns lowercased string
- .strip() returns string without whitespace at beginning and end
- .find("bla") returns index of position of substring "bla" or -1 if not found
- .replace("a","b") returns string where "a" is replaced by "b"
- .count("bla") counts how often substring "bla" occurs

9

Natural Language Processing

Bottom-up vs. top-down **Natural Language Processing** Advanced NLP Regular expressions From test to large-scale
00000000   ○●○   00   00   0000
00   000   000000   000000
                          000000   000000000
                          0000
                          000

Natural Language Processing

Bottom-up vs. top-down **Natural Language Processing** Advanced NLP Regular expressions From test to large-scale
00000000   000   00   00   0000
00   000   000000   000000
                         000000   000000000
                         0000
                         000

NLP: What and why?

Preprocessing steps

tokenization How do we (best) split a sentence into tokens
(terms, words)?

pruning How can we remove unneccessary words/
punctuation?

lemmatization How can we make sure that slight variations of the
same word are not counted differently?

parse sentences How can identify and encode grammatical
functions of tokens?

Natural Language Processing

Better tokenization

Bottom-up vs. top-down   Natural Language Processing   Advanced NLP   Regular expressions   From test to large-scale
00000000   000   00   00   0000
00   0●0   000000   000000
                         000000   000000000
                         0000
                         000

OK, good enough, perfect?

> ### .split()
>
> - space $\rightarrow$ new word
>
> - no further processing whatsoever
>
> - thus, only works well if we do a preprocessing ourselves (e.g., remove punctuation)

```
1   docs = ["This is a text", "I haven't seen John's derring-do. Second
        sentence!"]
2   tokens = [d.split() for d in docs]
```

```
1   [['This', 'is', 'a', 'text'], ['I', "haven't", 'seen', "John's", 'derring-do.', 'Second', '
        sentence!']]
```

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions From test to large-scale
00000000   000   00   00   0000
00   000●   000000   000000
                         000000   000000000
                         0000
                         000

OK, good enough, perfect?

Tokenizers from the NLTK pacakge

- multiple improved tokenizers that can be used instead of .split()
- e.g., Treebank tokenizer:
  - split standard contractions ("don't")
  - deals with punctuation

```
1   from nltk.tokenize import TreebankWordTokenizer
2   tokens = [TreebankWordTokenizer().tokenize(d) for d in docs]
```

```
1   [['This', 'is', 'a', 'text'], ['I', 'have', "n't", 'seen', 'John', "'s", 'derring-do.', 'Second
        ', 'sentence', '!']]
```

Notice the failure to split the . at the end of the first sentence in the second doc. That's because TreebankWordTokenizer expects *sentences* as input. See book for a solution.

Natural Language Processing

Stopword and punctuation removal

Bottom-up vs. top-down **Natural Language Processing** Advanced NLP Regular expressions From test to large-scale
00000000   000   00   00   0000
00   000   000000   000000
                          0●0000   000000000
                          0000
                          000

Stopword removal

The logic of the algorithm is very much related to the one of a simple sentiment analysis!

14

Bottom-up vs. top-down **Natural Language Processing** Advanced NLP Regular expressions From test to large-scale
00000000   000   00   00   0000
00   000   000000   000000
                 000●000   000000000
                 0000
                 000

Stopword removal

What are stopwords?

- Very frequent words with little inherent meaning

- the, a, he, she, ...

- context-dependent: if you are interested in gender, he and she are no stopwords.

- Many existing lists as basis

Bottom-up vs. top-down   **Natural Language Processing**   Advanced NLP   Regular expressions   From test to large-scale
00000000   000   00   00   0000
00   000   000000   000000
                         000●00   000000000
                         0000
                         000

Stopword removal: What and why?

Why remove stopwords?

- If we want to identify key terms (e.g., by means of a word count), we are not interested in them

- If we want to calculate document similarity, it might be inflated

- If we want to make a word co-occurance graph, irrelevant information will dominate the picture

Bottom-up vs. top-down   **Natural Language Processing**   Advanced NLP   Regular expressions   From test to large-scale
○○○○○○○○   ○○○   ○○   ○○   ○○○○
○○   ○○○   ○○○○○○   ○○○○○○○
                              ○○○○○●○   ○○○○○○○○○
                              ○○○○
                              ○○○

Stopword removal

```
1   from nltk.corpus import stopwords
2   mystopwords = stopwords.words("english")
3   mystopwords.extend(["test", "this"])
4
5   def tokenize_clean(s, stoplist):
6       cleantokens = []
7       for w in TreebankWordTokenizer().tokenize(s):
8           if w.lower() not in stoplist:
9               cleantokens.append(w)
10      return cleantokens
11
12  tokens = [tokenize_clean(d, mystopwords) for d in docs]
```

```
1   [['text'], ["n't", 'seen', 'John', 'derring-do.', 'Second', 'sentence', '!']]
```

You can do more!

For instance, in line 8, you could add an *or* statement to also exclude punctuation.

Bottom-up vs. top-down **Natural Language Processing** Advanced NLP Regular expressions From test to large-scale
00000000   000   00   00   0000
00   000   000000   000000
                          000000●   000000000
                          0000
                          000

Removing punctuation

```
1  from nltk.tokenize import RegexpTokenizer
2  tokenizer = RegexpTokenizer(r'\w+')
3  tokenizer.tokenize("Hi teachers, what's up!")
```

```
1  ['Hi', 'teachers', 'what', 's', 'up']
```

```
1  from string import punctuation
2  doc = "Today is @Toni's Birthday!!!"
3  "".join([w for w in doc if w not in punctuation])
```

```
1  'Today is Tonis Birthday'
```

Natural Language Processing

Stemming and lemmatization

Bottom-up vs. top-down **Natural Language Processing** Advanced NLP Regular expressions From test to large-scale
00000000   000   00   00   0000
00   000   000000   000000
                          000000   000000000
                          0●00
                          000

NLP: What and why?

Why do stemming?

- Because we do not want to distinguish between smoke, smoked, smoking, . . .
- Typical preprocessing step (like stopword removal)

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions From test to large-scale
00000000   000   00   00   0000
00   000   000000   000000
                         000000   000000000
                         00●0
                         000

Stemming and lemmatization

- Stemming: reduce words to its stem by removing last part
  (drinking $\rightarrow$ drink)
- Lemmatization: find word that you would need to look up in a
  dictionary (drinking $\rightarrow$ drink, but also went $\rightarrow$ go)
- stemming is simpler than lemmatization
- lemmatization often better

Example below: tokenization and lemmatization with spacy in one
go:

```
1  import spacy
2  nlp = spacy.load('en') # potentially you need to install the language
       model first
3  lemmatized_tokens = [[token.lemma_ for token in nlp(doc)] for doc in
       docs]
```

20

Bottom-up vs. top-down   Natural Language Processing   Advanced NLP   Regular expressions   From test to large-scale
00000000   000   00   00   0000
00   000   000000
                         000000   000000000
                         000●
                         000

Stemming and stopword removal - let's combine them!

```
1  from nltk.stem.snowball import SnowballStemmer
2  from nltk.corpus import stopwords
3  stemmer=SnowballStemmer("english")
4  mystopwords = stopwords.words("english")
5  frase="I am running while generously greeting my neighbors"
6  frasenuevo=""
7  for palabra in frase.lower().split():
8          if palabra not in mystopwords:
9                  frasenuevo=frasenuevo + stemmer.stem(palabra) + " "
```

Now, print(frasenuevo) returns:

```
1  run generous greet neighbor
```

Perfect! Or:

```
1  print(" ".join([stemmer.stem(p) for p in frase.lower().split() if p not
       in mystopwords]))
```

Natural Language Processing

ngrams

Bottom-up vs. top-down **Natural Language Processing** Advanced NLP Regular expressions From test to large-scale
00000000   000   00   00   0000
00   000   000000   000000
                          000000   000000000
                          0000
                          0●0

Instead of just looking at single words (unigrams), we can also use adjacent words (bigrams).

Bottom-up vs. top-down   **Natural Language Processing**   Advanced NLP   Regular expressions   From test to large-scale
○○○○○○○○   ○○○   ○○   ○○   ○○○○
○○   ○○○   ○○○○○○   ○○○○○○○
                                  ○○○○○○   ○○○○○○○○○
                                  ○○○○
                                  ○○●

ngrams

```
1  import nltk
2  texts = ['This is the first text text text first', 'And another text
      yeah yeah']
3  texts_bigrams = [["_".join(tup) for tup in nltk.ngrams(t.split(),2)] for
       t in texts]
4  print(texts_bigrams)
```

[['This_is', 'is_the', 'the_first', 'first_text',
'text_text', 'text_text', 'text_first'],
['And_another', 'another_text', 'text_yeah',
'yeah_yeah']]

Typically, we would combine both. **What do you think? Why is
this useful? (and what may be drawbacks?)**

Advanced NLP

Bottom-up vs. top-down   Natural Language Processing   **Advanced NLP**   Regular expressions   From test to large-scale
00000000   000   ○●   00   0000
00   000   000000
                         000000   000000   000000000
                         0000
                         000

Process and/or enrich

Advanced NLP

We did a lot of BOW (and some POS-tagging), but we can get more

- Named Entity Recognition (NER) to get names of people, organizations, ...
- Dependency Parsing to find out exact relationships ⇒ nltk, Stanford, FROG, Spacy

Advanced NLP

Parsing sentences

Bottom-up vs. top-down Natural Language Processing **Advanced NLP** Regular expressions From test to large-scale
00000000   000   00   00   0000
00   000   0●0000   000000
                         000000   000000000
                         0000
                         000

NLP: What and why?

Why parse sentences?

- To find out what grammatical function words have
- and to get closer to the meaning.

Bottom-up vs. top-down Natural Language Processing **Advanced NLP** Regular expressions From test to large-scale
00000000   000   00   00   0000
00   000   000●00   000000
                         000000   000000000
                         0000
                         000

Parsing a sentence using NLTK

Tokenize a sentence, and "tag" the tokenized sentence:

```
1  tokens = nltk.word_tokenize(sentence)
2  tagged = nltk.pos_tag(tokens)
3  print (tagged[0:6])
```

gives you the following:

```
1  [('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
2  ('Thursday', 'NNP'), ('morning', 'NN')]
```

And you could get the word type of "morning" with
tagged[5][1]!

Bottom-up vs. top-down Natural Language Processing **Advanced NLP** Regular expressions From test to large-scale
00000000   000   00   00   0000
00   000   000●00   000000
                         000000   000000000
                         0000
                         000

Named Entity Recognition with spacy

Terminal:

```
1  sudo pip3 install spacy
2  sudo python3 -m spacy download nl # or en, de, fr ....
```

Python:

```
1  import spacy
2  nlp = spacy.load('nl')
3  doc = nlp('Een 38-jarige vrouw uit Zeist en twee mannen moeten 24
        maanden de cel in voor de gecordineerde oplichting van Rabobank-
        klanten.')
4  for ent in doc.ents:
5      print(ent.text,ent.label_)
```

returns:

```
1  Zeist LOC
2  Rabobank ORG
```

27

More NLP

http://nlp.stanford.edu http://spacy.io http://nltk.org

https://www.clips.uantwerpen.be/pattern

Bottom-up vs. top-down Natural Language Processing **Advanced NLP** Regular expressions From test to large-scale
○○○○○○○○   ○○○   ○○   ○○   ○○○○
○○   ○○○   ○○○○○●   ○○○○○○
                        ○○○○○○   ○○○○○○○○○
                        ○○○○
                        ○○○

Main takeaway

- Preprocessing matters, be able to make informed choices.
- Keep this in mind when moving to Machine Learning.

Regular expressions

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions From test to large-scale
○○○○○○○○   ○○○   ○○   ○●   ○○○○
○○   ○○○   ○○○○○○   ○○○○○○
                        ○○○○○○   ○○○○○○○○○
                        ○○○○
                        ○○○

Automated content analysis using regular expressions

Regular expressions

What is a regexp?

Bottom-up vs. top-down Natural Language Processing Advanced NLP **Regular expressions** From test to large-scale
00000000   000   00   00   0000
00   000   000000   0●0000
                        000000   000000000
                        0000
                        000

Regular Expressions: What and why?

What is a regexp?

- a *very* widespread way to describe patterns in strings

- Think of wildcards like * or operators like OR, AND or NOT in search strings: a regexp does the same, but is *much* more powerful

- You can use them in many editors (!), in the Terminal, in STATA ...and in Python

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions From test to large-scale
○○○○○○○○   ○○○   ○○   ○○   ○○○○
○○   ○○○   ○○○○○○   ○○●○○○
                         ○○○○○○   ○○○○○○○○○
                         ○○○○
                         ○○○

An example

Regex example

- Let's say we wanted to remove everything but words from a tweet
- We could do so by calling the .replace() method
- We could do this with a regular expression as well:
  [^a-zA-Z] would match anything that is not a letter

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions From test to large-scale
00000000   000   00   00   0000
00   000   000000   000●00
                         000000   000000000
                         0000
                         000

Basic regexp elements

Alternatives

    [TtFf] matches either T or t or F or f

Twitter|Facebook matches either Twitter or Facebook

        . matches any character

Repetition

        * the expression before occurs 0 or more times

        + the expression before occurs 1 or more times

Bottom-up vs. top-down Natural Language Processing Advanced NLP **Regular expressions** From test to large-scale
○○○○○○○○   ○○○   ○○   ○○   ○○○○
○○   ○○○   ○○○○○○   **○○○○●○**
                ○○○○○○   ○○○○○○○○○
                ○○○○
                ○○○

regexp quizz

Which words would be matched?

1. [Pp]ython

2. [A-Z]+

3. RT ?:? @[a-zA-Z0-9]*

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions From test to large-scale
○○○○○○○○   ○○○   ○○   ○○   ○○○○
○○   ○○○   ○○○○○○   ○○○○○●
                         ○○○○○○   ○○○○○○○○○
                         ○○○○
                         ○○○

What else is possible?

See the table in the book!

Regular expressions

Using a regexp in Python

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions From test to large-scale
○○○○○○○○   ○○   ○○   ○○   ○○○○
○○   ○○○   ○○○○○○   ○●○○○○○○○
                         ○○○○○○
                         ○○○○
                         ○○○

How to use regular expressions in Python

The module `re`*****

`re.findall("[Tt]witter|[Ff]acebook",testo)` returns a list
with all occurances of Twitter or Facebook in the
string called `testo`

`re.findall("[0-9]+[a-zA-Z]+",testo)` returns a list with all
words that start with one or more numbers followed
by one or more letters in the string called `testo`

`re.sub("[Tt]witter|[Ff]acebook","a social medium",testo)`
returns a string in which all all occurances of Twitter
or Facebook are replaced by "a social medium"

Use the less-known but more powerful module `regex` instead to support all dialects used in the book

Bottom-up vs. top-down   Natural Language Processing   Advanced NLP   Regular expressions   From test to large-scale
00000000   000   00   00   0000
00   000   000000   000000
                         000000   00●000000
                         0000
                         000

How to use regular expressions in Python

The module re

re.match(" +([0-9]+) of ([0-9]+) points",line) returns
      None unless it *exactly* matches the string line. If it
      does, you can access the part between () with the
      .group() method.

Example:

```
1  line="          2 of 25 points"
2  result=re.match(" +([0-9]+) of ([0-9]+) points",line)
3  if result:
4      print (f"Your points: {}result.group(1)}, Maximum points: {result.
           group(2)})
```

Your points: 2 Maximum points: 25

Possible applications

Data preprocessing

- Remove unwanted characters, words, . . .
- Identify *meaningful* bits of text: usernames, headlines, where an article starts, . . .
- filter (distinguish relevant from irrelevant cases)

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions From test to large-scale
00000000   003   00   00   0000
00   000   000000   000000
                        000000   0000●0000
                        0000
                        000

Possible applications

Data analysis: Automated coding

- Actors
- Brands
- links or other markers that follow a regular pattern
- Numbers (!)

Example 1: Counting actors

```
1   import re, csv
2   from glob import glob
3   count1_list=[]
4   count2_list=[]
5   filename_list = glob("/home/damian/articles/*.txt")
6
7   for fn in filename_list:
8       with open(fn) as fi:
9           artikel = fi.read()
10          artikel = artikel.replace('\n',' ')
11
12          count1 = len(re.findall('Israel.*(minister|politician.*|[Aa]uthorit)',
                artikel))
13          count2 = len(re.findall('[Pp]alest',artikel))
14
15          count1_list.append(count1)
16          count2_list.append(count2)
17
18  output=zip(filename_list,count1_list, count2_list)
19  with open("results.csv", mode='w',encoding="utf-8") as fo:
20      writer = csv.writer(fo)
21      writer.writerows(output)
```

Bottom-up vs. top-down   Natural Language Processing   Advanced NLP   Regular expressions   From test to large-scale
00000000   000   00   00   0000
00   000   000000   000000
                         000000   000000●00
                         0000
                         000

Example 2: Which number has this Lexis Nexis article?

```
 1   All Rights Reserved
 2
 3   2 of 200 DOCUMENTS
 4
 5   De Telegraaf
 6
 7   21 maart 2014 vrijdag
 8
 9   Brussel bereikt akkoord aanpak probleembanken;
10   ECB krijgt meer in melk te brokkelen
11
12   SECTION: Finance; Blz. 24
13   LENGTH: 660 woorden
14
15   BRUSSEL  Europa heeft gisteren op de valreep een akkoord bereikt
16   over een saneringsfonds voor banken. Daarmee staat de laatste
```

41

Bottom-up vs. top-down   Natural Language Processing   Advanced NLP   Regular expressions   From test to large-scale
○○○○○○○○   ○○○   ○○   ○○   ○○○○
○○   ○○○   ○○○○○○   ○○○○○○
                 ○○○○○○   ○○○○○○○○●○
                 ○○○○
                 ○○○

Example 2: Check the number of a lexis nexis article

```
 1   All Rights Reserved
 2
 3   2 of 200 DOCUMENTS
 4
 5   De Telegraaf
 6
 7   21 maart 2014 vrijdag
 8
 9   Brussel bereikt akkoord aanpak probleembanken;
10   ECB krijgt meer in melk te brokkelen
11
12   SECTION: Finance; Blz. 24
13   LENGTH: 660 woorden
14
15   BRUSSEL  Europa heeft gisteren op de valreep een akkoord bereikt
16   over een saneringsfonds voor banken. Daarmee staat de laatste
```

```
 1   for line in tekst:
 2   matchObj=re.match(r" +([0-9]+) of ([0-9]+) DOCUMENTS",line)
```

42

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions From test to large-scale
00000000   000   00   00   0000
00   000   000000   000000
                        000000   00000000●
                        0000
                        000

Practice yourself!

Let's take some time to write some regular expressions. Write a script that

- extracts URLS form a list of strings
- removes everything that is not a letter or number from a list of strings

(first develop it for a single string, then scale up)

More tips: http://www.pyregex.com/

From test to large-scale

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions **From test to large-scale**
00000000   00   00   00   0●00
00   000   000000   000000
                        000000   000000000
                        0000
                        000

General approach

1. Take a single string and test your idea

```python
1  t = "This is a test test test."
2  print(t.count("test"))
```

2a. You'd assume it to return 3. If so, scale it up:

```python
1  results = []
2  for t in listwithallmytexts:
3      r = t.count("test")
4      print(f"{t} contains the substring {r} times")
5      results.append(r)
```

2b. If you *only* need to get the list of results, a list comprehension is more elegant:

```python
1  results = [t.count("test") for t in listwithallmytexts]
```

44

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions **From test to large-scale**
○○○○○○○○   ○○○   ○○   ○○   ○○○○
○○   ○○○   ○○○○○○   ○○○○○○○
              ○○○○○○○   ○○○○○○○○○○
              ○○○○
              ○○○

General approach

Test on a single string, then make a for loop or list comprehension!

Own functions

If it gets more complex, you can write your own function and then use it in the list comprehension:

```
1  def mycleanup(t):
2      # do sth with string t here, create new string t2
3      return t2
4
5  results = [mycleanup(t) for t in allmytexts]
```

45

Bottom-up vs. top-down Natural Language Processing Advanced NLP Regular expressions **From test to large-scale**
00000000   00   00   00   000●
00   000   000000   000000
                        000000   000000000
                        0000
                        000

Pandas string methods as alternative

If you select column with strings from a pandas dataframe, pandas offers a collection of string methods (via `.str.`) that largely mirror standard Python string methods:

```
1   df['newcoloumnwithresults'] = df['columnwithtext'].str.count("bla")
```

To pandas or not to pandas for text?

Partly a matter of taste.

Not-too-large dataset with a lot of extra columns? Advanced statistical analysis planned? Sounds like pandas.

It's mainly a lot of text? Wanna do some machine learning later on anyway? It's large and (potentially) messy? Doesn't sound like pandas is a good idea.