

CS310 Natural Language Processing

自然语言处理

Lecture 00 - Python and Basic Text Processing

Instructor: Yang Xu

主讲人：徐炀

xuyang@sustech.edu.cn

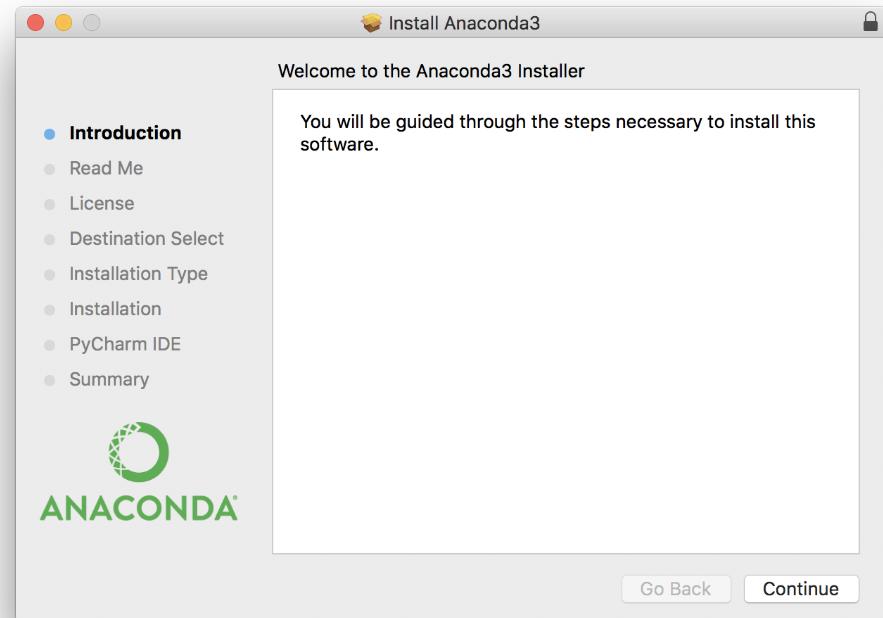
Some slides credit to Dan Jurafsky: <https://web.stanford.edu/~jurafsky/slp3/>

Content

- Python Basics
 - Install Python and Jupyter
 - String processing
 - Regular expression
- Basic Text Processing
 - Words and corpora
 - Tokenization

Python Installation

- Python installation via anaconda is recommended



Link:

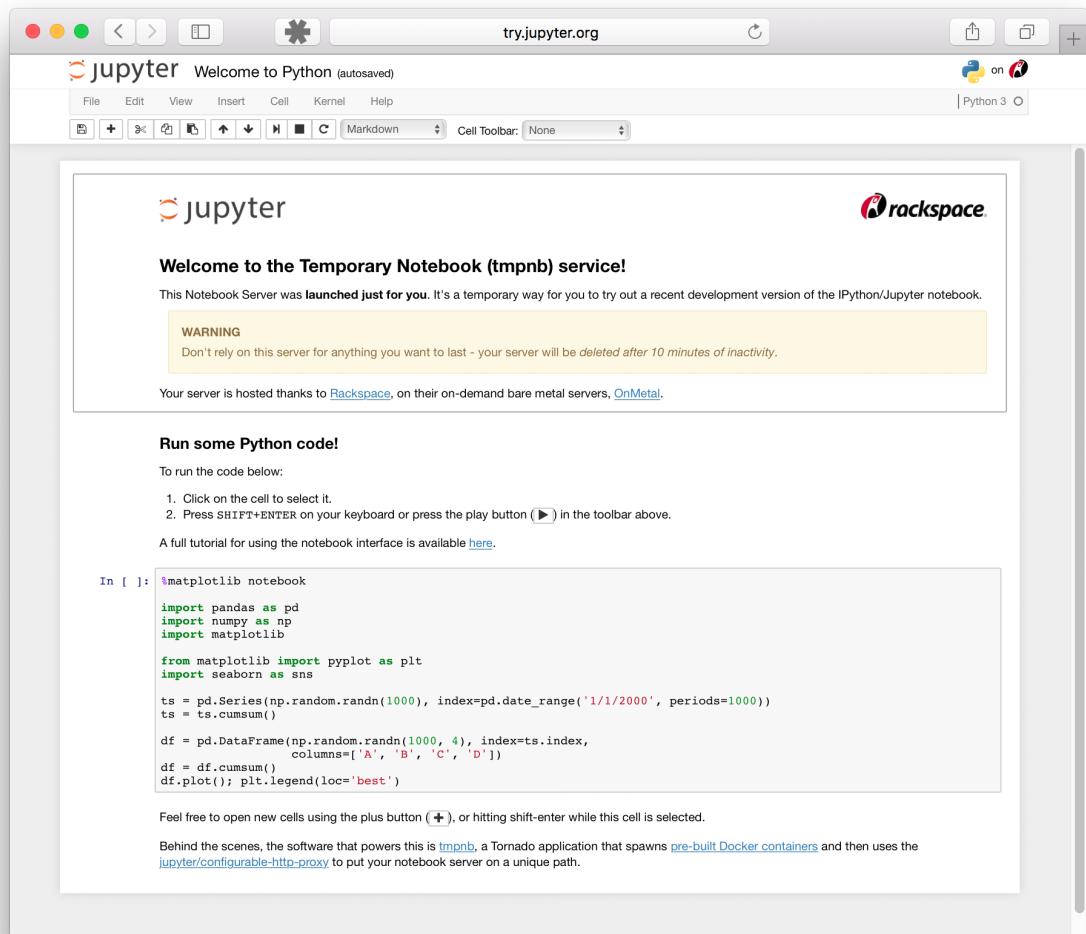
<https://www.anaconda.com/distribution/>

==> choose the distribution that suits you

Python 3.8 - 3.12 is recommended

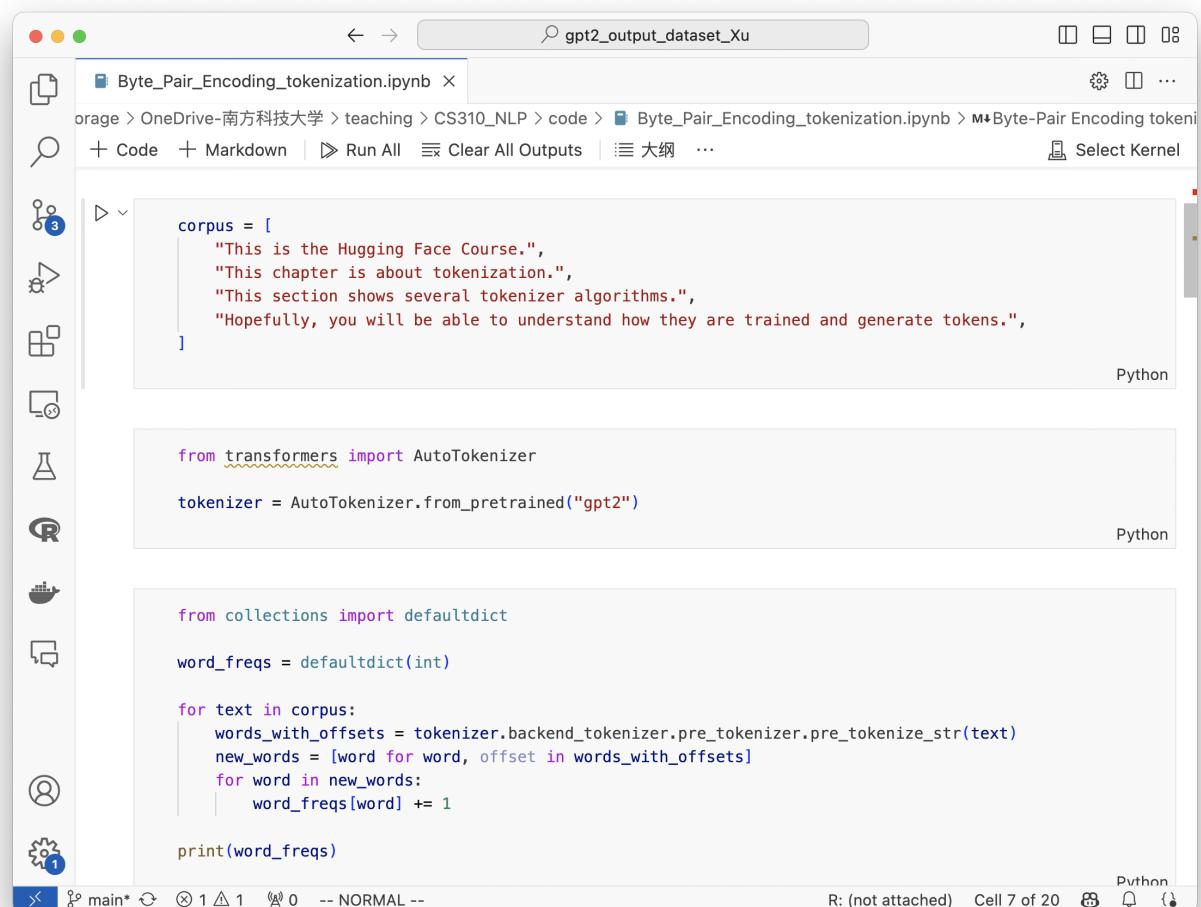
Jupyter Installation

- Official site:
<https://jupyter.org/install>
- or via anaconda:
<https://anaconda.org/anaconda/jupyter>
- Read the document:
<https://docs.jupyter.org/en/latest/start/index.html>



IDEs recommendation

- Visual studio code
- Or JetBrains PyCharm



```
corpus = [
    "This is the Hugging Face Course.",
    "This chapter is about tokenization.",
    "This section shows several tokenizer algorithms.",
    "Hopefully, you will be able to understand how they are trained and generate tokens.",

]

from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")

from collections import defaultdict

word_freqs = defaultdict(int)

for text in corpus:
    words_with_offsets = tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str(text)
    new_words = [word for word, offset in words_with_offsets]
    for word in new_words:
        word_freqs[word] += 1

print(word_freqs)
```

Python basics

- **str** is a built-in type for handling strings in Python
- Common sequence operations, on **str**, **list**, and **tuple**
- **x in s** True if an item of *s* is equal to *x*, else False
- **x not in s** False if an item of *s* is equal to *x*, else True
- **s + t** the concatenation of *s* and *t*
- **len(s)** the length of *s*
- **s[i]** the *i*th element of *s*, **start from 0**

```
s1 = [1, 2, 3]
s2 = [4, 5, 6]
print(s1 + s2)
```

```
[1, 2, 3, 4, 5, 6]
```

```
s = [1, 2, 3, 4]
print(s[0])
print(s[len(s)-1])
print(s[-1])
print(s[-2])
```

```
1
```

```
4
```

```
4
```

```
3
```

Common operations on string (str)

- **str.split(sep)** Return a **list** of substrings (e.g., words) in the string, resulted from using *sep* as the delimiter. The default separator is space ‘ ’.
E.g., ‘I love Python’.split() = ['I', 'love', 'Python']
‘I love Python’.split(' ') = ['I', 'love', 'Python']
‘144.182.67.1’.split('.') = ['144', '182', '67', '1']
- **str.upper()** Return a copy of the string with all the cased characters converted to uppercase
- **str.lower()** Return a copy of the string with all the cased characters converted to lowercase

String is Immutable

- Operations on string do NOT change the value of the string.

S1= “hello world!”

S1.split(' ')

What does Python return?

一定得要变量接收

What is the value of S1 now?

S2=S1.split(' ')

What is the value of S1 now?

What is the value of S2 now?

S1=S1.split(' ')

What is the value of S1 now?

Common operations on string (str)

- **str.strip()** Returns a **copy** of the string with the leading and trailing characters removed
 - E.g., ‘ hello\n ’.strip() = ‘hello’
- **str.startswith(prefix)** Returns True if string starts with the *prefix*
 - E.g., ‘Python’.startswith(‘P’) = True
- **str.endswith(suffix)**

Common operations on string

- **str.replace(*old*, *new*)** Return a copy of the string with all occurrences of substring *old* replaced by *new*

```
In [1]: 'This-is-a-long-string'.replace('-', ' ')
```

```
Out[1]: 'This is a long string'
```

```
In [2]: 'I do not want spaces'.replace(' ', '')
```

```
Out[2]: 'Idonotwantspaces'
```

Regular expressions

- A **formal language** for specifying text strings
- How can we search for any of these?
 - woodchuck
 - woodchucks
 - Woodchuck
 - Woodchucks



Stephen C Kleene

The concept of regular expressions began in the 1950s, when the American mathematician Stephen Cole Kleene formalized the concept of a regular language.

Slides credit to Dan Jurafsky: <https://web.stanford.edu/~jurafsky/sl3/>

Regular Expressions: Disjunctions

- Letters inside square brackets []

Pattern	Matches
[wW]oodchuck	Woodchuck , woodchuck
[1234567890]	Any digit

- Ranges [A-Z]

Pattern	Matches	
[A-Z]	An upper case letter	<u>D</u> r enched Blossoms
[a-z]	A lower case letter	<u>m</u> y beans were impatient
[0-9]	A single digit	Chapter <u>1</u> : Down the Rabbit Hole

Regular Expressions: Negation in Disjunction

- Negations **[^Ss]**

- Carat (^) means negation only when first in []

Pattern	Matches	
[^A-Z]	Not an upper case letter	Oyfn pripetchik
[^Ss]	Neither 'S' nor 's'	I have no exquisite reason"
[^e^]	Neither e nor ^	Look <u>here</u>
a^b	The pattern a carat b	Look up <u>a^b</u> now

Slides credit to Dan Jurafsky: <https://web.stanford.edu/~jurafsky/slp3/>

Regular Expressions: More Disjunction

- Woodchuck is another name for groundhog!
- The **pipe |** for disjunction

Pattern	Matches
groundhog woodchuck	woodchuck
yours mine	yours
a b c	= [abc]
[gG]roundhog [Ww]oodchuck	Woodchuck



Slides credit to Dan Jurafsky: <https://web.stanford.edu/~jurafsky/slp3/>

Regular Expressions: ? * + .

? question mark
 * asterisk
 + plus
 . dot

Pattern	Matches	
colou?r	Optional <u>previous char</u>	<u>color</u> <u>colour</u>
oo*h!	0 or more of <u>previous char</u>	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
o+h!	1 or more of <u>previous char</u>	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
baa+		<u>baa</u> <u>baaa</u> <u>baaaa</u> <u>baaaaa</u>
beg.n	任何字符	<u>begin</u> <u>begun</u> <u>begun</u> <u>beg3n</u>

Regular Expressions: Anchors ^ \$

Note that ^
is outside []
匹配行的开始位置

Pattern	Matches
<code>^[A-Z]</code>	<u>P</u> alo Alto
<code>^[^A-Za-z]</code>	<u>1</u> <u>“H</u> ello”
<code>\.\$</code>	The end. <u>_</u>
<code>.\$</code> 匹配行的结尾位置	The end? <u>_</u> The end! <u>_</u>

Example

- Find me all instances of the word “the” in a text.

the

Misses capitalized examples

[tT]he

Incorrectly returns other or theology

[^a-zA-Z][tT]he[^a-zA-Z]

Errors

- The previous examples show two kinds of errors:
 1. Matching strings that we should not have matched
(**there**, **then**, **other**)
False positives (Type I errors)
 2. Not matching things that we should have matched
(**The**)
False negatives (Type II errors)

Errors (cont.)

- We are always dealing with these kinds of errors.
- Reducing the error rate for an application often involves two antagonistic efforts:
 - Increasing ***precision*** (minimizing false positives)
 - Increasing ***recall*** (minimizing false negatives).

trade-off problem

Summary

- Regular expressions play a surprisingly large role
 - Sophisticated sequences of regular expressions are often the “first model” for any text processing text
- For hard tasks, we use machine learning classifiers
 - But regular expressions are still used for pre-processing, or as features in the classifiers
 - Can be very useful in capturing generalizations

Regular expression in Python

- Use `re` module
- Doc: <https://docs.python.org/3/library/re.html>
- Example: `search()` vs. `match()`
- `re.match()` checks for a match only at the beginning of the string
- `re.search()` checks for a match anywhere in the string (this is what Perl does by default)
- `re.fullmatch()` checks for entire string to be a match

```
>>> re.match("c", "abcdef") # No match
>>> re.search("^c", "abcdef") # No match
>>> re.search("^a", "abcdef") # Match
<re.Match object; span=(0, 1), match='a'>
```

>>>

- `re.findall()` Return all non-overlapping matches of pattern in string, as a list of strings or tuples

```
>>> re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
['foot', 'fell', 'fastest']
>>> re.findall(r'(\w+)=(\d+)', 'set width=20 and height=10')
[('width', '20'), ('height', '10')]
```

>>>

RegEx for Chinese characters

```
raw = """"  
一些中文夹杂着English words以及数字1234  
"""  
✓ 0.0s
```

```
re.findall(r'[\u4e00-\u9fa5]', raw)  
✓ 0.0s  
['一', '些', '中', '文', '夹', '杂', '着', '以', '及', '数', '字']
```

```
re.findall(r'【--龟】', raw)  
✓ 0.0s  
['一', '些', '中', '文', '夹', '杂', '着', '以', '及', '数', '字']
```

```
re.findall(r'[\u4e00-\u9fa5]+[0-9]+', raw)  
✓ 0.0s  
['以及数字1234']
```

Table of Content

- Python Basics
 - Install Python and Jupyter
 - string processing
 - regular expression
- **Basic Text Processing**
 - **Words and corpora**
 - Tokenization

How many words in a sentence?

- "I do uh main- mainly business data processing"
 - Fragments, filled pauses
- "Seuss's **cat** in the hat is different from other **cats!**"
 - **Lemma:** same stem, part of speech, rough word sense
 - **cat** and **cats** = same lemma
 - **Wordform:** the full inflected surface form
 - **cat** and **cats** = different wordforms

Lemma: 词目

Word form: 词形

How many words in a sentence?

they lay back on the San Francisco grass and looked at
the stars and their

- **Type:** an element of the vocabulary. 不同的单词类型 (忽略变形)
- **Token:** an instance of that type in running text. 文本的最小单位
- How many?
 - 15 tokens (or 14)
 - 13 types (or 12) (or 11?)

How many words in a corpus?

N = number of tokens

词汇量（即不同的单词数量）随着文档数量的增加而增长，但这种增长是有限的。

V = vocabulary = set of types, $|V|$ is size of vocabulary

Heaps Law = Herdan's Law = $|V| = kN^\beta$ where often $.67 < \beta < .75$

i.e., vocabulary size grows with $>$ square root of the number of word tokens

	Tokens = N	Types = V
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13+ million

Corpora

Words don't appear out of nowhere!

A text is produced by

- a specific writer(s),
- at a specific time,
- in a specific variety,
- of a specific language,
- for a specific function.

Corpora vary along dimension like

- **Language:** 7097 languages in the world
- **Variety**, like African American Language varieties.
 - Double Negatives: "I don't know nothing" to emphasize negation.
 - "I ain't got no time for that" (I don't have any time for that).
 - "He be running every morning" (He runs every morning).
- **Code switching**, e.g., Spanish/English, Hindi/English:
 - “Selina, double check 一下我的 case”
- **Genre:** news article, fiction, scientific articles, Wikipedia document
- **Author Demographics:** writer's age, gender, ethnicity, SES

Corpus datasheets

Gebru et al (2020), Bender and Friedman (2018)

Motivation:

- Why was the corpus collected? By whom? Who funded it?

Situation: In what situation was the text written?

Collection process: If it is a subsample how was it sampled? Was there consent? Pre-processing?

- +Annotation process, language variety, demographics, etc.

Table of Content

- Python Basics
 - Install Python and Jupyter
 - string processing
 - regular expression
- **Basic Text Processing**
 - Words and corpora
 - Tokenization

Text Normalization

- Every NLP task requires text normalization:
 1. **Tokenizing (segmenting) words**
 2. Normalizing word formats
 3. Segmenting sentences

Space-based tokenization

- A very simple way to tokenize
 - For languages that use space characters between words
 - Arabic, Cyrillic, Greek, Latin, etc., based writing systems
 - Segment off a token between instances of spaces
- Unix tools for space-based tokenization
 - The "tr" command
 - Inspired by Ken Church's UNIX for Poets
 - Given a text file, output the word tokens and their frequencies

Simple Tokenization in UNIX

- (Inspired by Ken Church's UNIX for Poets.)
- Given a text file, output the word tokens and their frequencies

```
tr -sc 'A-Za-z' '\n' < shakes.txt      Change all non-alpha to newlines
```

```
| sort      Sort in alphabetical order
```

```
| uniq -c  Merge and count each type
```

1945 A	25 Aaron
72 AARON	6 Abate
19 ABBESS	1 Abates
5 ABBOT	5 Abbess
...	6 Abbey
...	3 Abbot

Slides credit to Dan Jurafsky: <https://web.stanford.edu/~jurafsky/slp3/>

The first step: tokenizing

```
tr -sc 'A-Za-z' '\n' < shakes.txt | head
```

THE

SONNETS

by

William

Shakespeare

From

fairest

creatures

We

...

Slides credit to Dan Jurafsky: <https://web.stanford.edu/~jurafsky/slp3/>

The second step: sorting

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | head
```

A
A
A
A
A
A
A
A
A
...

Slides credit to Dan Jurafsky: <https://web.stanford.edu/~jurafsky/slp3/>

More counting

- Merging upper and lower case

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c
```

- Sorting the counts

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c | sort -n -r
```

23243 the
22225 i
18618 and
16339 to
15687 of
12780 a
12163 you
10839 my
10005 in
8954 d

What happened here?

Slides credit to Dan Jurafsky: <https://web.stanford.edu/~jurafsky/slp3/>

Issues in Tokenization

- Can't just blindly remove punctuation:
 - m.p.h., Ph.D., AT&T, cap'n
 - prices (\$45.55)
 - dates (01/02/06)
 - URLs (<http://www.stanford.edu>)
 - hashtags (#nlproc)
 - email addresses (someone@cs.colorado.edu)
- Clitic: a word that doesn't stand on its own
 - "are" in we're, French "je" in j'ai, "le" in l'honneur
- When should multiword expressions be words?
 - New York, rock 'n' roll

Slides credit to Dan Jurafsky: <https://web.stanford.edu/~jurafsky/slp3/>

Solution: Regular Expressions for Tokenizing Text

```
>>> raw = """'When I'M a Duchess,' she said to herself, (not in a very hopeful tone
... though), 'I won't have any pepper in my kitchen AT ALL. Soup does very
... well without--Maybe it's always pepper that makes people hot-tempered,'..."""
```

```
>>> re.split(r' ', raw) ①
["'When", "I'M", 'a', "Duchess", "", 'she', 'said', 'to', 'herself', '(not', 'in',
'a', 'very', 'hopeful', 'tone\nthough)', "'I", "won't", 'have', 'any', 'pepper',
'in', 'my', 'kitchen', 'AT', 'ALL.', 'Soup', 'does', 'very\nwell', 'without--Maybe',
"it's", 'always', 'pepper', 'that', 'makes', 'people', "hot-tempered", "..."]
>>> re.split(r'[ \t\n]+', raw) ②
["'When", "I'M", 'a', "Duchess", "", 'she', 'said', 'to', 'herself', '(not', 'in',
'a', 'very', 'hopeful', 'tone', 'though)', "'I", "won't", 'have', 'any', 'pepper',
'in', 'my', 'kitchen', 'AT', 'ALL.', 'Soup', 'does', 'very', 'well', 'without--Maybe',
"it's", 'always', 'pepper', 'that', 'makes', 'people', "hot-tempered", "..."]
```

```
>>> print(re.findall(r"\w+(:[-]\w+)*|'|-.(.)+|\S\w*", raw))
[ "", 'When', "I'M", 'a', 'Duchess', "", "", 'she', 'said', 'to', 'herself', '',
('not', 'in', 'a', 'very', 'hopeful', 'tone', 'though', ''), "", "", 'I',
"won't", 'have', 'any', 'pepper', 'in', 'my', 'kitchen', 'AT', 'ALL', '.', 'Soup',
'does', 'very', 'well', 'without', '--', 'Maybe', "it's", 'always', 'pepper',
'that', 'makes', 'people', "hot-tempered", '', "", "..."]
```

Source: <https://www.nltk.org/book/ch03.html>

Symbol	Function
\b	Word boundary (zero width)
\d	Any decimal digit (equivalent to [0-9])
\D	Any non-digit character (equivalent to [^0-9])
\s	Any whitespace character (equivalent to [\t\n\r\f\v])
\S	Any non-whitespace character (equivalent to [^\t\n\r\f\v])
\w	Any alphanumeric character (equivalent to [a-zA-Z0-9_])
\W	Any non-alphanumeric character (equivalent to [^a-zA-Z0-9_])
\t	The tab character
\n	The newline character

Tokenization in languages without spaces

How about languages that don't use spaces to separate words (like Chinese, Japanese, Thai)?

How do we decide where the token boundaries should be?

Word tokenization in Chinese

- Chinese words are composed of characters called "hanzi" (or sometimes just "zi")
- Each one represents a *meaning* unit called a **morpheme**. 词素
- Each word has on average 2.4 of them.
- But deciding what counts as a word is complex and not agreed upon.

How to do word tokenization in Chinese?

- 姚明进入总决赛 “Yao Ming reaches the finals”

How to do word tokenization in Chinese?

- 姚明进入总决赛 “Yao Ming reaches the finals”

- 3 words?

- 姚明 进入 总决赛

- YaoMing reaches finals

How to do word tokenization in Chinese?

- 姚明进入总决赛 “Yao Ming reaches the finals”

- 3 words?

- 姚明 进入 总决赛

- YaoMing reaches finals

- 5 words?

- 姚 明 进 入 总 决 赛

- Yao Ming reaches overall finals

How to do word tokenization in Chinese?

- 姚明进入总决赛 “Yao Ming reaches the finals”

- 3 words?

- 姚明 进入 总决赛

- YaoMing reaches finals

- 5 words?

- 姚 明 进 入 总 决 赛

- Yao Ming reaches overall finals

- 7 characters? (don't use words at all):

- 姚 明 进 入 总 决 赛

- Yao Ming enter enter overall decision game

Word tokenization / segmentation

In Chinese it's common to just treat each character (*zi*) as a token.

- So the **segmentation** step is very simple

In other languages (like Thai and Japanese), more complex word segmentation is required.

- The standard algorithms are neural sequence models trained by supervised machine learning.

From a modern Chinese perspective

- 词 word \neq 字 character
- Roughly, 词 = [字]+
- 中文分词 Chinese word segmentation: Segment a sequence of characters into a list of words

```
seg_list = jieba.cut("我来到北京清华大学", cut_all=False)
print("Default Mode: " + "/ ".join(seg_list)) # 精确模式

seg_list = jieba.cut("他来到了网易杭研大厦") # 默认是精确模式
print(", ".join(seg_list))

seg_list = jieba.cut_for_search("小明硕士毕业于中国科学院计算所, 后在日本京都大学深造") # 搜索引擎模式
print(", ".join(seg_list))
```

Source: <https://github.com/fxsjy/jieba>

【精确模式】：我/ 来到/ 北京/ 清华大学

【新词识别】：他, 来到, 了, 网易, 杭研, 大厦 (此处, “杭研”并没有在词典中, 但是也被Viterbi算法识别出来了)

【搜索引擎模式】： 小明, 硕士, 毕业, 于, 中国, 科学, 学院, 科学院, 中国科学院, 计算, 计算所, 后, 在, 日本, 京都, 大学, 日本京都大学, 深造