I256:
Applied Natural Language Processing

Marti Hearst
Week 3

Upcoming Weeks

- *This week:
  - Tokenization, Regex
- Week 4:
  - Morphological Analysis, Parts of Speech
- Week 5:
  - Part of Speech Tagging
  - **Assignment** (due week 7)
- Week 6:
  - Partial Parsing, Chunking
- Week 7:
  - Word senses, Similarity
- Weeks 8-9
  - Text Classification and ML
  - **Assignment** (due week 10)
- Week 10:
  - Syntactic & Semantic Parsing
- Week 11:
  - Unsupervised Methods
  - **Assignment** (due week 13)
- *Week 12:
  - Discourse Processing

*short week

Upcoming Weeks

- Week 13:
  - Final Project Proposals
- *Week 14:
  - Dialogue and Coreference
- Week 15:
  - QA and Course wrap-up
- RRR Week
  - Final project posters & demos

Today

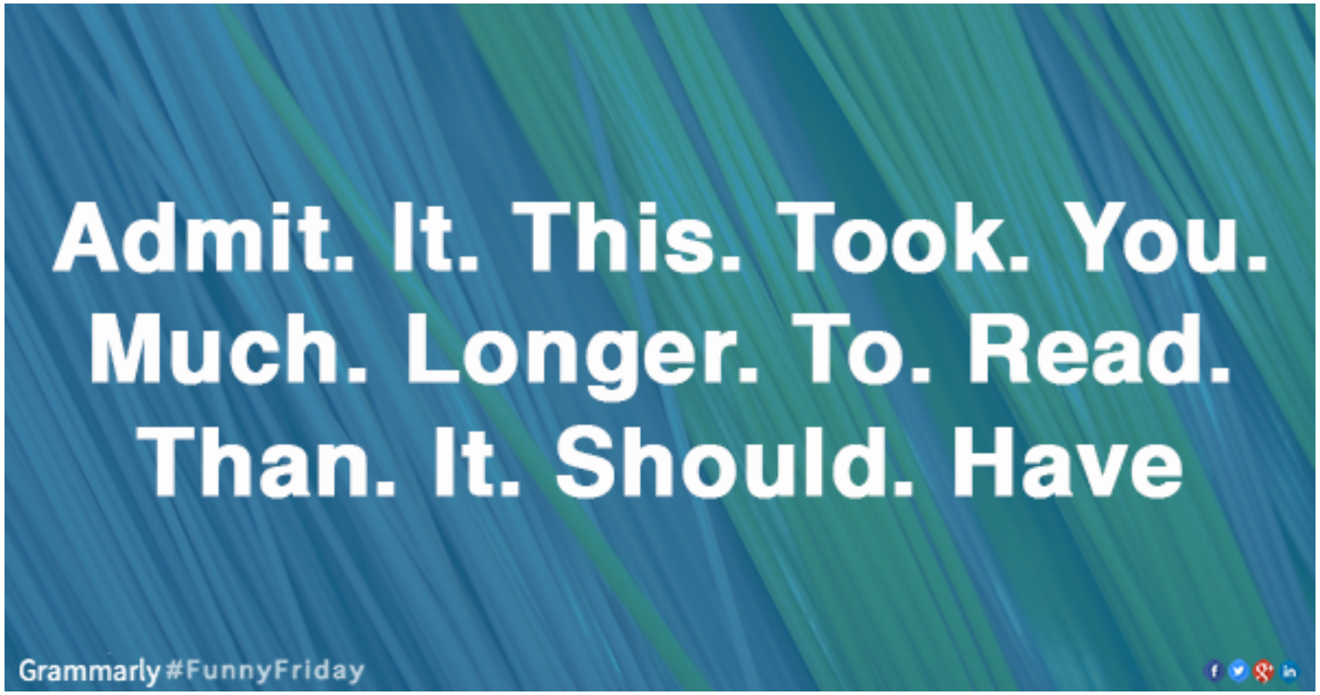
- Orthography
- Tokenization
  - With Regular Expressions
- Normalizing Text

COURSE CONCEPT MAP

	Word-Level	Phrase-Level	Sentence-Level	Discourse-Level
Segment-ation	Tokenization		Sentence Boundary Detection	
Syntax				
Semant-ics				

ORTHOGRAPHY

orthos (correct) + graphia (writing)

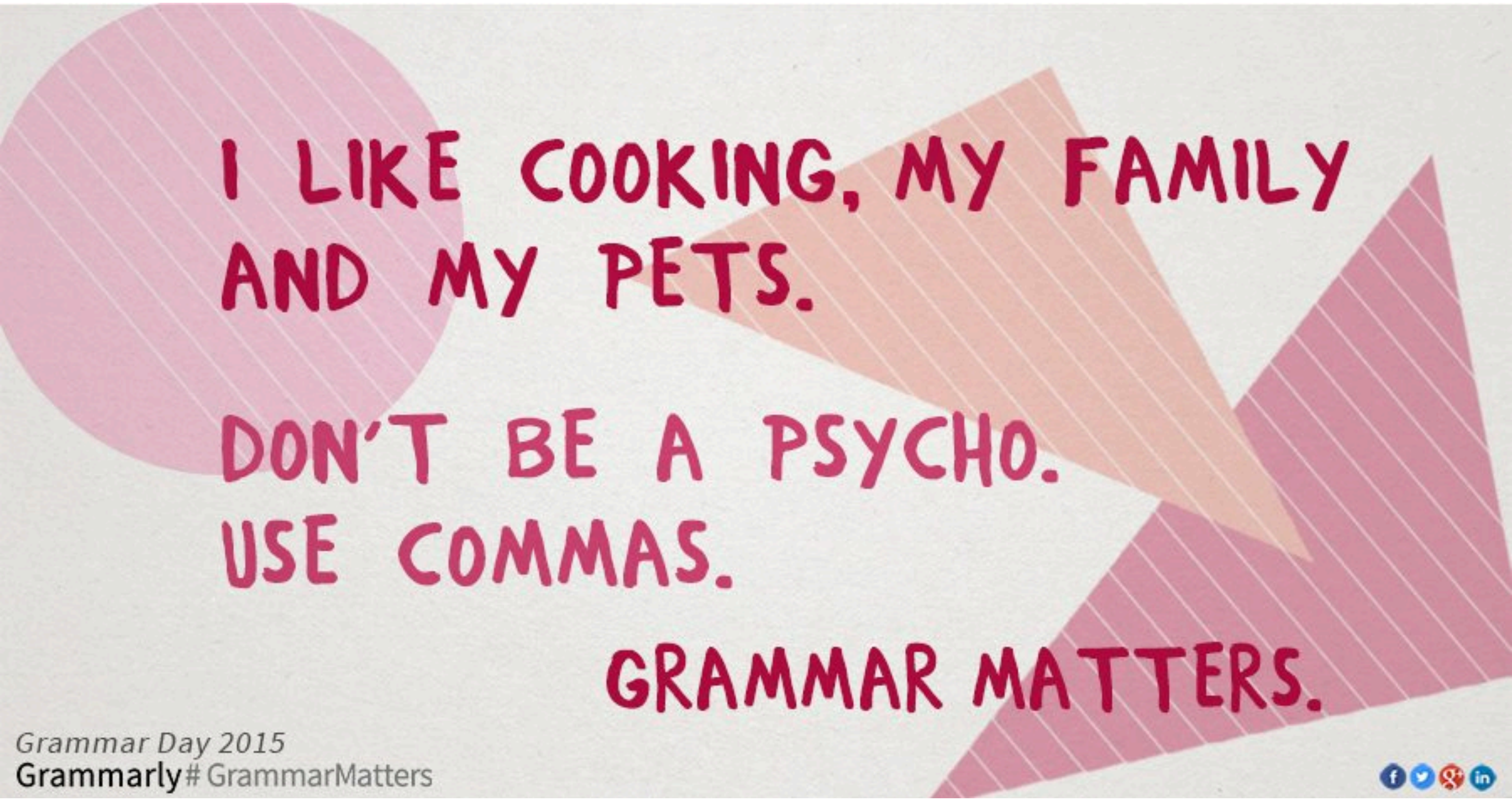

Admit. It. This. Took. You. Much. Longer. To. Read. Than. It. Should. Have
Grammarly #FunnyFriday

ORTHOGRAPHY

The method of writing a language, including rules of spelling, hyphenation, capitalization, word breaks, emphasis, and punctuation. (adapted from English Wikipedia)

STOP CLUBBING, BABY SEALS

Once again, punctuation makes all the difference ...


I LIKE COOKING, MY FAMILY AND MY PETS.

DON'T BE A PSYCHO. USE COMMAS.

GRAMMAR MATTERS.

Grammar Day 2015

Grammarly # GrammarMatters

Tokenization Issues

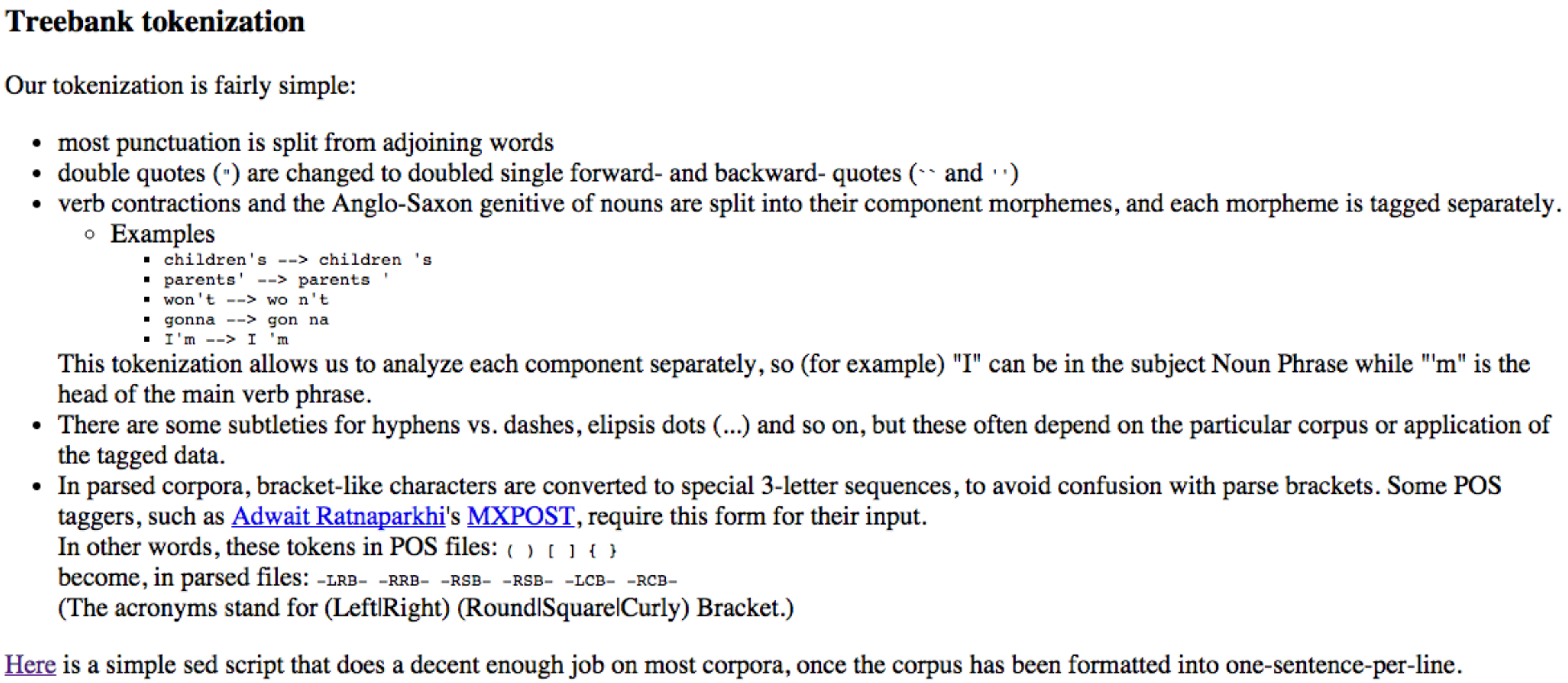
Whitespace often does not indicate a word break:

- San Francisco
- The New York-New Haven railroad
- Wake up, work out
  - I couldn't *work* the answer *out*

Tokenization

- What is a word?
  - String of continuous alphanumeric characters surrounded by white space
    - *$22.50 (oop … not just alphanumeric!)*
  - Main clue (in English) is the occurrence of whitespace
  - Problems
    - Periods: we often remove punctuation but sometimes it's useful to keep periods
      - (*Wash*. → *wash*)
    - Single apostrophes, contractions, possessive
      - (*isn't, didn't, dog's*: -> *is* + *n't* or *not*)
    - Hyphenation:
      - Sometimes best as a single word: *co-operate*
      - Sometimes best as separate words:
        - » *26-year-old, aluminum-export ban*

Penn Treebank Tokenization Rules



Treebank tokenization

Our tokenization is fairly simple:

- most punctuation is split from adjoining words
- double quotes (") are changed to doubled single forward- and backward- quotes (`` and '')
- verb contractions and the Anglo-Saxon genitive of nouns are split into their component morphemes, and each morpheme is tagged separately.
    - Examples
        - `children's --> children 's`
        - `parents' --> parents '`
        - `won't --> wo n't`
        - `gonna --> gon na`
        - `I'm --> I 'm`

  This tokenization allows us to analyze each component separately, so (for example) "I" can be in the subject Noun Phrase while "'m" is the head of the main verb phrase.
- There are some subtleties for hyphens vs. dashes, elipsis dots (...) and so on, but these often depend on the particular corpus or application of the tagged data.
- In parsed corpora, bracket-like characters are converted to special 3-letter sequences, to avoid confusion with parse brackets. Some POS taggers, such as Adwait Ratnaparkhi's MXPOST, require this form for their input.
  In other words, these tokens in POS files: `( ) [ ] { }`
  become, in parsed files: `-LRB- -RRB- -RSB- -RSB- -LCB- -RCB-`
  (The acronyms stand for (Left|Right) (Round|Square|Curly) Bracket.)

Here is a simple sed script that does a decent enough job on most corpora, once the corpus has been formatted into one-sentence-per-line.

Tokenization is Tricky

- Hard to get it right.
- Hard to define what is a token.
- No single solution works well across domains.
- The RegexTokenizer is probably the best way to go for your collection.
- Practice (if you like):
  - The NLTK corpus collection includes a sample of Penn Treebank data,
    - including the raw Wall Street Journal text (`nltk.corpus.treebank_raw.raw()`)
    - and the tokenized version, `nltk.corpus.treebank.words()`
  - So you can compare your algorithm to a standard.

Text Normalization

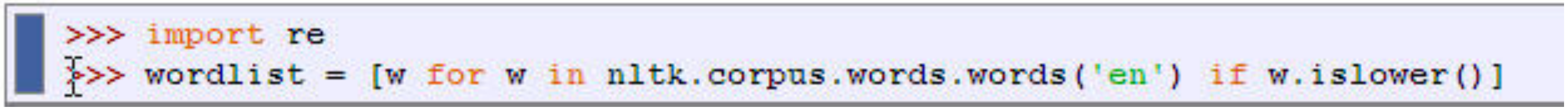
- Identify *non-standard words* including numbers, abbreviations, and dates,
- Map such tokens to a special vocabulary.
  - Examples:
    - Every decimal number could be mapped to a single token 0.0,
    - Every acronym could be mapped to AAA.
  - This keeps the vocabulary small and improves the accuracy of many language modeling tasks.

Regular Expressions for Detecting Word Patterns

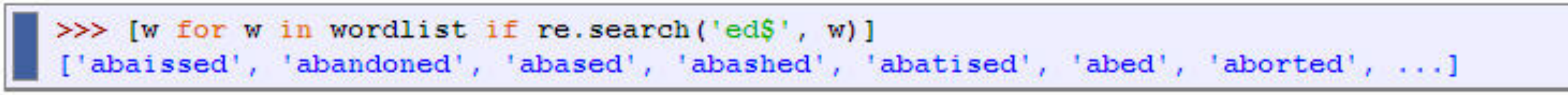
- Many linguistic processing tasks involve pattern matching.
- Regular expressions (RE) give us a powerful and flexible method for describing the character patterns we are interested in.
- To use regular expressions in Python we need to import the `re` library using: `import re`

Regular Expressions for Detecting Word Patterns

`re.search(p,s)` is a function to check whether the pattern p can be found somewhere inside the string s



```
>>> import re
>>> wordlist = [w for w in nltk.corpus.words.words('en') if w.islower()]
```



```
>>> [w for w in wordlist if re.search('ed$', w)]
['abaissed', 'abandoned', 'abased', 'abashed', 'abatised', 'abed', 'aborted', ...]
```

Crossword puzzles anyone?

LET’S HAVE SOME FUN WITH WORD LISTS!

```python
# The NLTK book assumes we import these each time
import nltk, re, pprint
from nltk import word_tokenize
from IPython.display import Image
```

Examples from the NLTK book: using regex's to find patterns from a wordlist

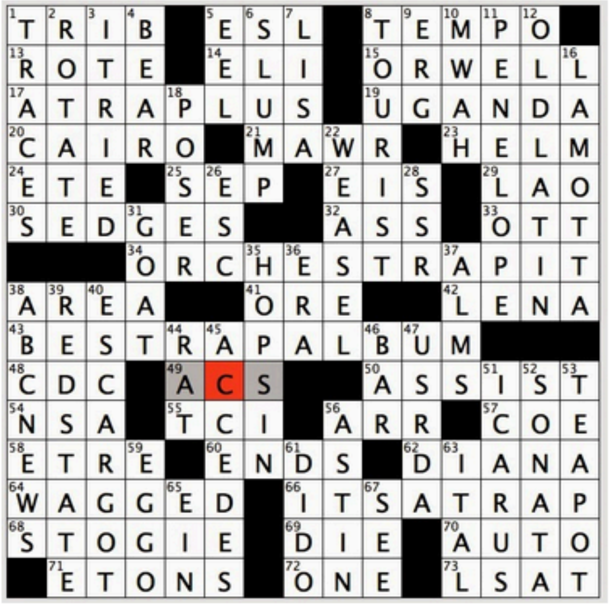
```python
wordlist = [w for w in nltk.corpus.words.words('en') if w.islower()]
```

```python
[w for w in wordlist if re.search('ed$', w)]
```

```python
[w for w in wordlist if re.search('^..j..t..$', w)]
```

Exercise: look at a crossword puzzle like the one below. Mentally remove some of the filled in letters and search for words that only match some of the remaining letters. See how well the regex does at generating a list containing the correct word.

```python
Image(url='http://2.bp.blogspot.com/-Z9EdELyjmOY/VSSQuSN0qRI/AAAAAAAAY4U/crxW-Bz9cnM/s1600/Apr8
```

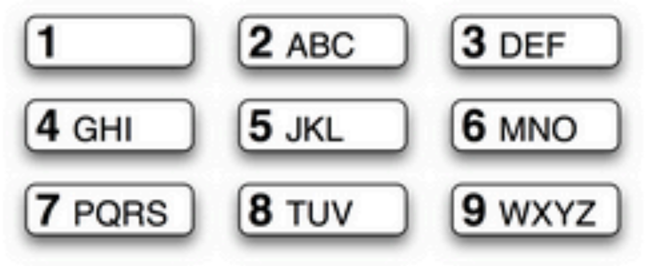


1 T	2 R	3 I	4 B	■	5 E	6 S	7 L	■	8 T	9 E	10 M	11 P	12 O	■
13 R	O	T	E	■	14 E	L	I	■	15 O	R	W	E	L	16 L
17 A	T	R	A	18 P	L	U	S	■	19 U	G	A	N	D	A
20 C	A	I	R	O	■	21 M	A	22 W	R	■	23 H	E	L	M
24 E	T	E	■	25 S	26 E	P	■	27 E	I	28 S	■	29 L	A	O
30 S	E	D	31 G	E	S	■	■	32 A	S	S	■	33 O	T	T
■	■	■	34 O	R	C	35 H	36 E	S	T	R	37 A	P	I	T
38 A	39 R	40 E	A	■	■	41 O	R	E	■	■	42 L	E	N	A
43 B	E	S	T	44 R	45 A	P	A	L	46 B	47 U	M	■	■	■
48 C	D	C	■	49 A	C	S	■	■	50 A	S	S	51 I	52 S	53 T
54 N	S	A	■	55 T	C	I	■	56 A	R	R	■	57 C	O	E
58 E	T	R	59 E	■	60 E	N	61 D	S	■	62 D	63 I	A	N	A
64 W	A	G	G	65 E	D	■	66 I	T	67 S	A	T	R	A	P
68 S	T	O	G	I	E	■	69 D	I	E	■	70 A	U	T	O
■	71 E	T	O	N	S	■	72 O	N	E	■	73 L	S	A	T

```
[w for w in wordlist if re.search('^...arg..$', w)]
```

Find the words that can be construed from keying in 4653 from a keypad

```
Image(url='http://www.nltk.org/images/T9.png', width=200)
```



```
[w for w in wordlist if re.search('^[ghi][mno][jlk][def]$', w)]
```

Exercise: finds words that fit the last four digits of your phone number

Abbreviation Recognition

- Schwartz & Hearst 2003
- Much simpler than other approaches.
- Extracts abbreviation-definition candidates adjacent to parentheses.
- Finds correct definitions by matching characters in the abbreviation to characters in the definition, **starting from the right of the parentheses**.
  - The first character in the abbreviation must match a character at the beginning of a word in the definition.
  - To increase precision a few simple heuristics are applied to eliminate incorrect pairs.
- Example: _**H**eat **s**hock transcription **f**actor (HSF)._
- _The algorithm finds the correct definition, but not the correct alignment: **H**eat shock tran**s**cription **f**actor_

Python Code Sample Output

- 9568 69 75 Apaf-1
- 9568 29 67 apoptosis protease activation factor-1

- 9824 45 49 CaMK
- 9824 8 43 calmodulin-dependent protein kinase
  - Error: should be Calcium/calmodulin-dependent protein kinase (our implementation would not miss this)

- 7439 72 76 Mts1
- 7439 34 70 metastasis-associated protein S100A4

Segmentation

- Word segmentation
  - For languages that do not put spaces between words
    - – Chinese, Japanese,
- Sentence segmentation
  - Divide text into sentences

Sentence Segmentation

- Sentence:
  - Something ending with a .. ?, ! (and sometimes :)

- Sentence boundary detection algorithms
  - Heuristic Rules, Regex
  - Statistical classification trees (Riley 1989)
    - Probability of a word to occur before or after a boundary, case and length of words
  - Machine learning (Neural network) (Palmer & Hearst 1997)
    - Part of speech distribution of preceding and following words
  - Maximum Entropy (Mikheev 1998)
  - Unsupervised (Kiss, Tibor, & Strunk, 2006)
    - Automatically identify abbreviations using collocations; then apply heuristics to determine if sentence boundary.
    - This is what NLTK’s punkt is based on.

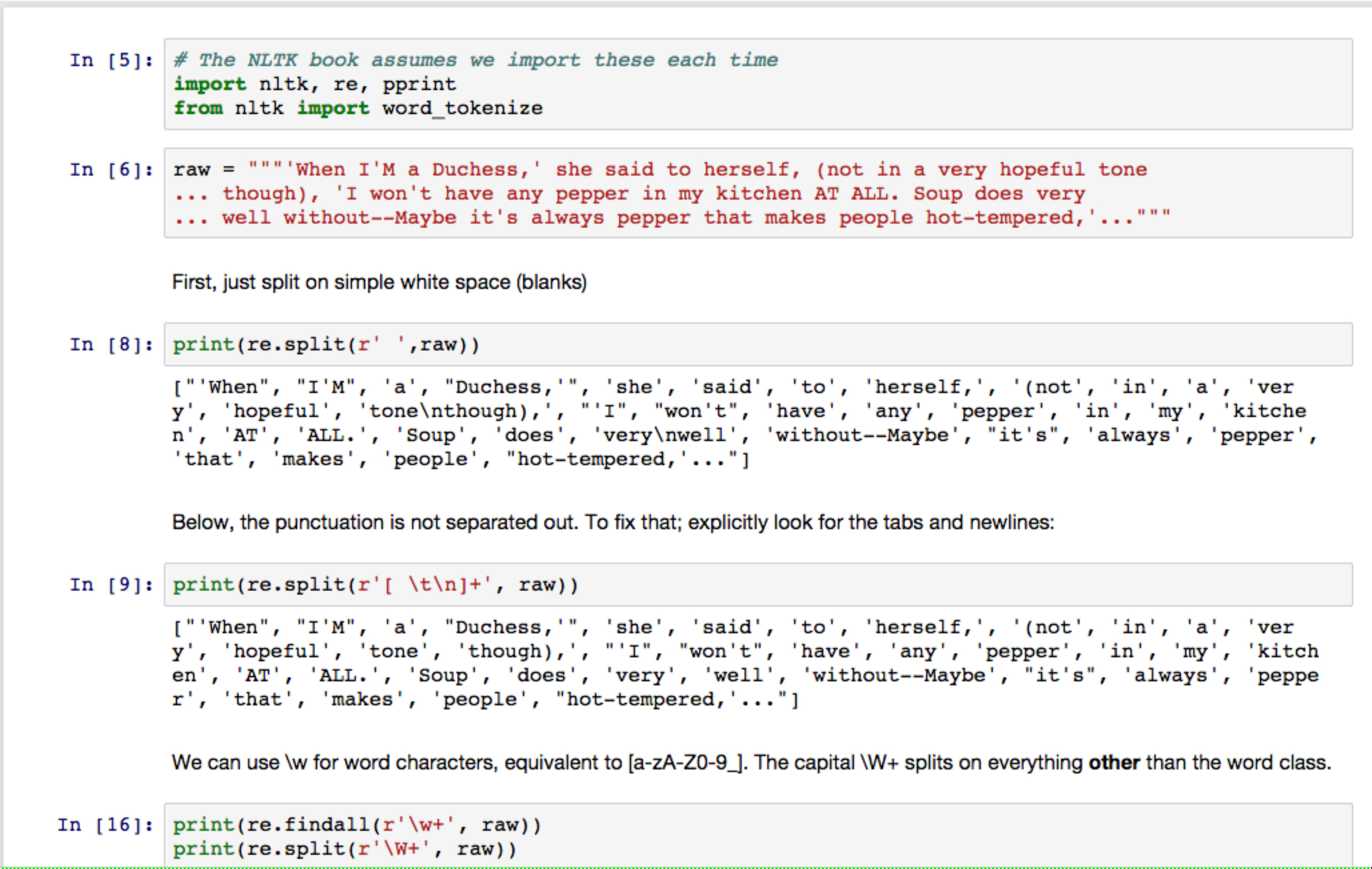
Sentence Segmentation in NLTK

- Some corpora are available in NLTK in sentence format.
  - In other cases, the text is in raw format
- NLTK provides the Punkt sentence segmenter
  - Advantage: doesn't rely on orthographic cues (except periods)

    sent_tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
  - You can train it to work more specifically for your corpus!
    - https://github.com/gooofy/voxforge/blob/master/lm-train-punkt.py http://stackoverflow.com/questions/21160310/training-data-format-for-nltk-punkt

LET'S PRACTICE TOKENIZING!

Tokenizing with Regex's



```
In [5]: # The NLTK book assumes we import these each time
        import nltk, re, pprint
        from nltk import word_tokenize
```

```
In [6]: 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,'..."""
```

First, just split on simple white space (blanks)

```
In [8]: print(re.split(r' ',raw))
```

```
["'When", "I'M", 'a', "Duchess,'", 'she', 'said', 'to', 'herself,', '(not', 'in', 'a', 'ver
y', 'hopeful', 'tone\nthough),', "'I", "won't", 'have', 'any', 'pepper', 'in', 'my', 'kitche
n', 'AT', 'ALL.', 'Soup', 'does', 'very\nwell', 'without--Maybe', "it's", 'always', 'pepper',
'that', 'makes', 'people', "hot-tempered,'..."]
```

Below, the punctuation is not separated out. To fix that; explicitly look for the tabs and newlines:

```
In [9]: print(re.split(r'[ \t\n]+', raw))
```

```
["'When", "I'M", 'a', "Duchess,'", 'she', 'said', 'to', 'herself,', '(not', 'in', 'a', 'ver
y', 'hopeful', 'tone', 'though),', "'I", "won't", 'have', 'any', 'pepper', 'in', 'my', 'kitch
en', 'AT', 'ALL.', 'Soup', 'does', 'very', 'well', 'without--Maybe', "it's", 'always', 'peppe
r', 'that', 'makes', 'people', "hot-tempered,'..."]
```

We can use \w for word characters, equivalent to [a-zA-Z0-9_]. The capital \W+ splits on everything **other** than the word class.

```
In [16]: print(re.findall(r'\w+', raw))
         print(re.split(r'\W+', raw))
```

Tokenizing with Regex's

```
In [16]: print(re.findall(r'\w+', raw))
         print(re.split(r'\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', 'peppe
r', 'that', 'makes', 'people', 'hot', 'tempered']
['', 'When', 'I', 'M', 'a', 'Duchess', 'she', 'said', 'to', 'herself', 'not', 'in', 'a', 'ver
y', 'hopeful', 'tone', 'though', 'I', 'won', 't', 'have', 'any', 'pepper', 'in', 'my', 'kitch
en', 'AT', 'ALL', 'Soup', 'does', 'very', 'well', 'without', 'Maybe', 'it', 's', 'always', 'p
epper', 'that', 'makes', 'people', 'hot', 'tempered', '']
```

But now the contraction *I'm* is split up, which is a problem! And we are losing the punctuation we do want, like those hyphens and apostrophes that do have meaning.

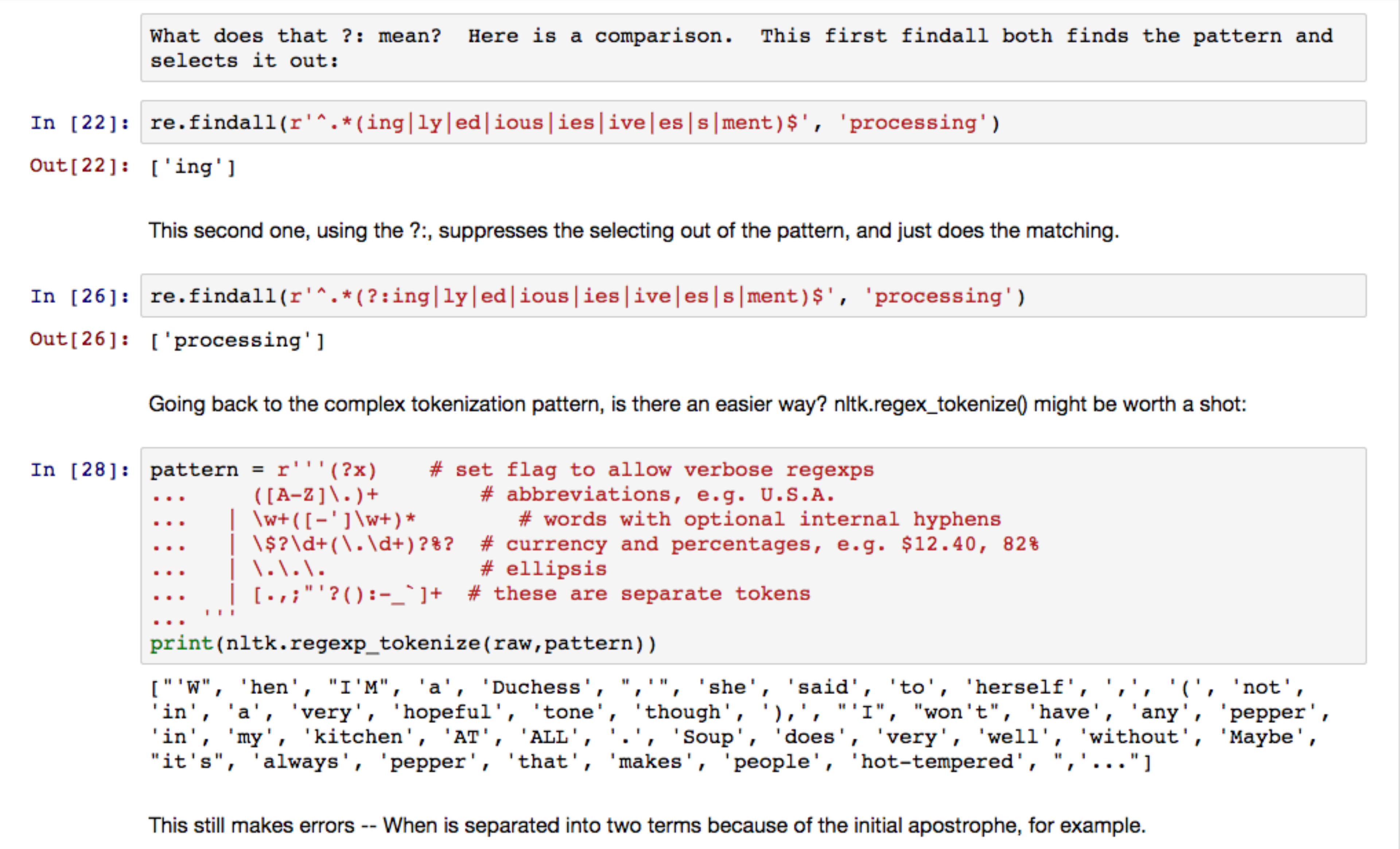
Now let's get more complicated:

- \w+ to match words
- \w+(?:[-']\w+)*|' to match word-internal hyphens and apostrophes
- \S+ to match non-whitespace characters (the complement of spaces)
- followed by \w* to optionally match more words
- precede these with [-.(]+ to match double hyphen, ellipses, and open parentheses, which are to be tokenized separately.

```
In [19]: print(re.findall(r"\w+(?:[-']\w+)*|'|[-.(]+|\S\w*", raw))
```

```
["'", 'When', "I'M", 'a', 'Duchess', ',', "'", 'she', 'said', 'to', 'herself', ',', '(', 'no
t', 'in', 'a', 'very', 'hopeful', 'tone', 'though', ')', ',', "'", 'I', "won't", 'have', 'an
y', 'pepper', 'in', 'my', 'kitchen', 'AT', 'ALL', '.', 'Soup', 'does', 'very', 'well', 'witho
ut', '--', 'Maybe', "it's", 'always', 'pepper', 'that', 'makes', 'people', 'hot-tempered',
',', "'", '...']
```

Tokenizing with Regex's



```
What does that ?: mean?  Here is a comparison.  This first findall both finds the pattern and
selects it out:
```

```
In [22]: re.findall(r'^.*(ing|ly|ed|ious|ies|ive|es|s|ment)$', 'processing')
Out[22]: ['ing']
```

This second one, using the ?:, suppresses the selecting out of the pattern, and just does the matching.

```
In [26]: re.findall(r'^.*(?:ing|ly|ed|ious|ies|ive|es|s|ment)$', 'processing')
Out[26]: ['processing']
```

Going back to the complex tokenization pattern, is there an easier way? nltk.regex_tokenize() might be worth a shot:

```
In [28]: pattern = r'''(?x)    # set flag to allow verbose regexps
         ...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
         ...   | \w+([-']\w+)*        # words with optional internal hyphens
         ...   | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
         ...   | \.\.\.            # ellipsis
         ...   | [.,;"'?():-_`]+  # these are separate tokens
         ... '''
         print(nltk.regexp_tokenize(raw,pattern))

["'W", 'hen', "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', ",'..."]
```

This still makes errors -- When is separated into two terms because of the initial apostrophe, for example.

Text Normalization

- You have to exercise judgement, make choices.
  - Convert to lower case?
    - What about proper nouns?
  - Remove punctuation?
    - What about abbreviations?
  - Remove hyphens?
    - What about compound words?
  - What about single quotes? Double quotes?
  - What about whitespace before punctuation?
  - Do you want to remove stopwords?
    - How do you define stopwords?

Next Week

- Stemming
- Lemmatization
- Morphology
- Parts of Speech