LING 388: Computers and Language

Lecture 14

Today's Topics

- Homework 6 Review
- re.IGNORECASE
- regex groups:
 - \n (n a number) for repeated groups
- re.finditer() and words in context
- Regex Exercises: extra credit only

- Question 1:
 - create a new corpus by lowercasing all the words in the Jane Austen novel austen-emma.txt in nltk.corpus.gutenberg
 - see previous lecture for how to access the words
 - recall a corpus is just a list of words
 - What is the vocabulary size before and after lowercasing?

```
python
>>> import nltk
>>> words = nltk.corpus.gutenberg.words('austen-emma.txt')
>>> len(words)
192427
>>> vocab = set(words)
>>> len(vocab)
7811
>>> words2 = [word.lower() for word in words]
>>> len(words2)
192427
>>> vocab2 = set(words2)
>>> len(vocab2)
7344
```

- Question 2: lowercase vocab vs. original vocab.
- >>> len(vocab vocab2)

766

>>> len(vocab2 - vocab)

299

- 766 words with at least one upper case character
- 299 lowercase words that don't exist *entirely* in lowercase in the original text.

- 766 words with at least one upper case character
 - many of these are proper nouns, chapter titles, start of sentences, etc.
 - {'_I_', 'Cobham', 'Kings', 'September', 'Park', '_Dixons_', 'Dixon', 'Letters', 'Ceremonies', 'Ought', 'Little', 'Proportions', 'JULY', 'Hymen', 'Worse', 'Patty', 'Under', 'They', 'CHARADE', 'Pembroke', 'Their', 'CHAPTER', ... }
- Code:

```
>>> for x in (vocab - vocab2):
```

if x.islower():

print(x)

should be nothing (*the empty set*)

- 299 lowercase words that don't exist in all lowercase in the original text.

• Set intersection:

• words that exist in lower case

>>> len(vocab & vocab2)

7045

```
>>> len(vocab)
```

7811

```
>>> len(vocab2)
```

7344

- Exclusive or:
 - either in vocab or vocab2 but not in both

>>> len([word for word in (vocab ^ vocab2) if word.islower()])
299

```
• same as:
```

```
>>> len(vocab2 - vocab)
```

299

Python regex recap

Unicode characters ok in

Python 3.x

- Summary:
 - \w a character [A-Za-z0-9_]
 - \d [0-9]
 - \b word boundary
 - \s space character [\t\n\r\f\v]
- Operators:
 - * zero or more repeats
 - + one or more repeats
 - ? zero or one repeats
 - () grouping
- Raw string (avoid escaping \):
 - r"\w+"

Negation:

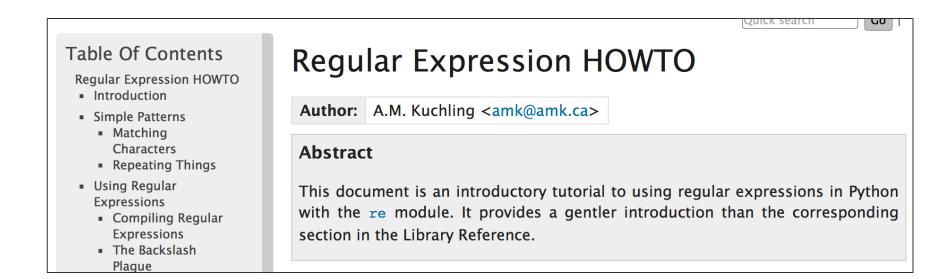
- \W anything not in \w
- \D anything not in \d
- Methods:
 - m = re.search(pattern, string)
 - return match object or None
 - m = re.match(pattern, string)
 - return match object or None
 - l = re.findal(pattern, string)
 - return list of strings/tuples
 - m = re.finditer(pattern, string)
 - return list of match objects or None

• Full Documentation:

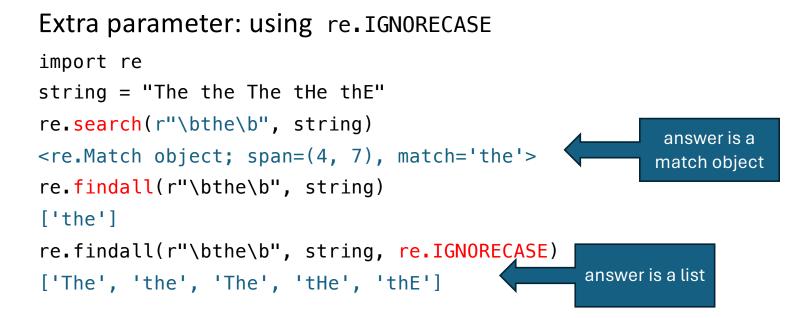
https://docs.python.org/3/library/re.html

Tutorial

More examples from https://docs.python.org/3/howto/regex.html



Case Insensitive regex search



The trouble with re.findall()

- Only capturing groups (...) are reported ...
- Repeating pattern ab: e.g. ababab..., we write the regex (ab)+
- Example:

```
>>> text = "ababcabababababacabd"
>>> import re
>>> re.findall(r'(ab)+', text)
['ab', 'ab', 'ab']
>>> re.findall(r'((ab)+)', text)
[('abab', 'ab'), ('abababab', 'ab'), ('ab', 'ab')]
```

re.findall(pattern, string, flags=0)

Return all non-overlapping matches of *pattern* in *string*, as a list of strings. The *string* is scanned left-to-right, and matches are returned in the order found. If one or more groups are present in the pattern, return a list of groups; this will be a list of tuples if the pattern has more than one group. Empty matches are included in the result.

The trouble with re.findall()

Example:

```
['abab', 'abababab', 'ab']
```

3. Can also specify a group as non-capturing (i.e. non-reporting) using (?: ...)
>>> re.findall(r'((?:ab)+)', text)
['abab', 'abababab', 'ab']

Group number: \n

- Other useful meta-characters:
 - ^ matches beginning of line
 - \$ matches end of line
 - \n n = group number, must match identically to group



Group number: \n

• Repeated Word Example:

• assuming you have the file looking-glass.txt in the same directory

>>> re.findall(r'\b(\w+)\W+\1\b',open('looking-glass.txt').read())

['PAWNS', 'Daisy', 'Oyster', 'Oyster', 'Daisy', 'had', 'it', 'had', 'thump', 'Faster', 'Faster', 'Now', 'Faster', 'I', 'it', 'I', 'Alice', 'long', 'oh', 'oh', 'oh', 'e', 'Feather', 'oh', 'to', 'had', 'with', 'unless', 'Ahoy', 'Ahoy', 'Alice', 'as', 'Aged', 'aged', 'no', 'Mutton', 'Alice', 'you']

• Suppose we limit ourselves to words beginning with a lowercase letter:

>> re.findall(r'\b([a-z]\w+)\W+\1\b',raw1)
['had', 'it', 'had', 'thump', 'it', 'long', 'oh', 'oh', 'oh', 'oh',
'to', 'had', 'with', 'unless', 'as', 'aged', 'no', 'you']

Suppose we want to see the repeated word in context?

Python's re module

	Method/Attribute	Purpose
	match()	Determine if the RE matches at the beginning of the string.
	<pre>search()</pre>	Scan through a string, looking for any location where this RE matches.
N	<pre>findall()</pre>	Find all substrings where the RE matches, and returns them as a list.
	<pre>finditer()</pre>	Find all substrings where the RE matches, and returns them as an
И		iterator.

Group number: \n

• Suppose we want to see the repeated word in context:

```
>>> raw = open('looking-glass.txt').read()
>>> for m in re.finditer(r'\b([a-z]\w+)\W+\1\b',raw):
... print('{}:'.format(m.group(1)),raw[m.start()-20:m.end()+20])
...
had: the _white_ kitten had had nothing to do
with
it: nothing to do
with it:-it was the black kitte
```

Group number: \n

had: Let's

pretend." She had had quite a long argume
thump: hear her footstep, thump, thump,
thump, along the g
it: n the other side of
it: it looked much darker
long: d been singing it a long long time!"

The other t

oh: ce unfinished. "Oh, oh, oh!" shouted the Queen oh: ger's bleeding! Oh, oh, oh, oh!"

Her screams
oh: , "but I soon shall-oh, oh,

oh!"

"When do you
oh: tle shrieks of "Oh, oh,
oh!" from poor Alice,
to: his very own
mouth_-to-to-"

"To send all his had: ly remarked.

(They had had quite enough of the with: Hideous. I fed him with-with-with Ham-sandwiches unless: e

said to herself, "unless—unless we're all part of $\ensuremath{\mathsf{t}}$

as:

Te

you: .

"We must support you, you know," the White Qu

Python's re module

• Numbers:

- \$ is a meta-character, it matches end of line. Suppose we want to match an actual dollar sign (\$), we need \\$.
- . is a meta-character, it's the wild-card character. If we want to match a decimal point (or period) (.), we need \.
- ? is a meta-character for optional. \\$? means the dollar sign is optional.

• Example:

- text = 'International benchmark Brent crude passed the long-anticipated threshold of \$80 per barrel on Tuesday, though it's since slipped back down to trade at \$78.47 as of Wednesday at 10:30 a.m. in London. West Texas Intermediate was trading at \$74.73 per barrel around the same time.'
- Let's write a regex for matching the decorated numbers in text.

regex for money:	>>> te:
\$ followed by	>>> te >>> re ['\$1,0 >>> te >>> re ['\$1 <u>,</u> 0
digits	L'\$1,0 >>> te
comma (for thousands, optional)	>>> re
decimal point (optional)	['\$1 <u>,</u> 0
,	>>>

```
>>> text3 = "$1,000,000.00 at No. 34"
>>> re.findall(r'\$[\d,]+', text3)
['$1,000,000']
>>> text3 = "$1,000,000.05 at No. 34"
>>> re.findall(r'\$[\d,\.]+', text3)
['$1,000,000.05']
>>>
```

Regex Exercises

- Text file on course website: Oliver Twist, Charles Dickens, 1838
 - imported from Project Gutenberg (<u>https://www.gutenberg.org</u>)
 - oliver_twist.txt
- How to import it:
 - first, be in the right working directory
 - raw = open('oliver_twist.txt', encoding='utf-8', errors='ignore').read()
- Check it has been imported correctly:

>>> len(raw)

893534

Regex Exercises

- 1. Look for all 3 letter words ending in *ly* in raw using a regex.
 - How many of them are there?
- 2. Look in raw for all words ending in ly that are 14 or more letters long.
 - How many of them are there?
- 3. Look in raw for bigrams (here: two words adjacent to each other but could be separated by non-word characters) that both end in ly.
 - How many of them are there?
- 4. Look in raw for two words both beginning with a capital letter but separated by a hyphen.
 - How many of them are there?
- Hints:
 - \w = word character, \W = non-word character, \b = word boundary

Regex Exercises

- Optional Homework for extra credit only
- Submit to sandiway@arizona.edu
- <u>SUBJECT</u>: 388 Regex Exercises **YOUR NAME**
- One PDF file only
 - include Python terminal and any screenshots in your answer
- Deadline:
 - midnight Monday 11th March