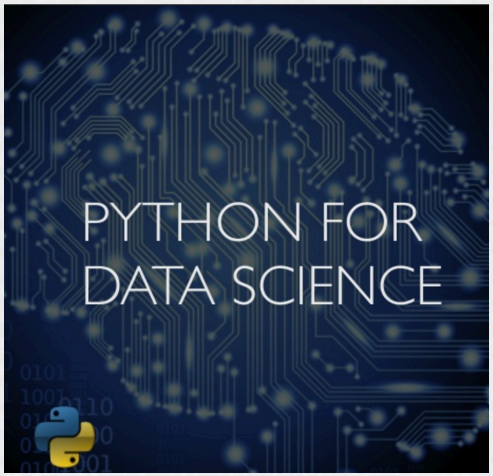
UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS



Python for Data Science

Lecture 12 (04/11, 04/13): Text Mining (1)

Decision, Operations & Information Technologies
Robert H. Smith School of Business
Spring, 2016

Regular expression

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- In computing, a regular expression, also referred to as “regex” or “regexp”, provides a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters. A regular expression is written in a formal language that can be interpreted by a regular expression processor.

Regular expression

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Use regular expressions to:
  - ❑Search a string (search and match)
  - ❑Replace parts of a string (sub)
  - ❑Break strings into smaller pieces (split)

Regular expression syntax

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Most characters match themselves
  - The regular expression "test" matches the string 'test', and only that string
- [x] matches any *one* of a list of characters
  - "[abc]" matches 'a', 'b', or 'c'
- [^x] matches any *one* character that is not included in *X*
  - "[^abc]" matches any single character *except* 'a', 'b', or 'c'

Regular expression syntax

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- "." matches any single character
- Parentheses can be used for grouping
  - "(abc)" matches 'abc'
- xly matches *x* or *y*
  - "thislthat" matches 'this' or 'that', but not 'thisthat'.

Regular expression syntax

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- *x** matches zero or more *x*'s
  - "a*" matches '', 'a', 'aa', etc.
- *x*+ matches one or more *x*'s
  - "a+" matches 'a', 'aa', 'aaa', etc.
- *x*? matches zero or one *x*'s
  - "a?" matches '' or 'a'
- *x{m, n}* matches *i x*'s, where $m \leq i \leq n$
  - "a{2,3}" matches 'aa' or 'aaa'

Regular expression syntax

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- “\d” matches any digit; “\D” any non-digit
- “\s” matches any whitespace character; “\S” any non-whitespace character
- “\w” matches any alphanumeric character; “\W” any non-alphanumeric character
- “^” matches the beginning of the string; “$” the end of the string
- “\b” matches a word boundary; “\B” matches a character that is not a word boundary

Regular expression module

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Before you use regular expressions in your program, you must import the library using "import re"

Search and match

- The two basic functions are **re.search** and **re.match**
  - Search looks for a pattern anywhere in a string
  - Match looks for a match starting at the beginning
- Both return *None* (logical false) if the pattern isn't found and a "match object" instance if it is

```
>>> import re
>>> pat = "a*b"
>>> re.search(pat,"fooaaabcde")
<_sre.SRE_Match object at 0x809c0>
>>> print re.match(pat,"fooaaabcde")
None
```

Using re.search()

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
import re

fh = open('test.txt','r')
for line in fh:
    line = line.rstrip()
    if re.search('^Hello',line):
        print line
```

test.txt:

1:Hello world

2:World, hello

3:Hello, hello

4:hello world

5:World

6:world, Hello

Output:

1:Hello world

3:Hello, hello

Group() function

- Using group to get the matched object

```
>>> r1 = re.search("a*b","fooaaabcde")
>>> r1.group()  # group returns string matched
'aaab'
>>> r1.start()  # index of the match start
3
>>> r1.end()    # index of the match end
7
>>> r1.span()   # tuple of (start, end)
(3, 7)
```

What got matched?

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Here's a pattern to match simple email addresses

  \w+@(\w+\.)+(com|org|net|edu)

```
>>> pat1 = "\w+@(\w+\.)+(com|org|net|edu)"
>>> r1 = re.match(pat, "kpzhang@umd.edu")
>>> r1.group()
'kpzhang@umd.edu'
```

- We might want to extract the pattern parts, like the email name and host

What got matched?

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- We can put parentheses around groups we want to be able to reference

```
>>> pat2 = "(\w+)@((\w+\.)+(com|org|net|edu))"
>>> r2 = re.match(pat2, "kpzhang@umd.edu")
>>> r2.group(1)
'kpzhang'
>>> r2.group(2)
'umd.edu'
>>> r2.groups()
r2.groups()
('kpzhang', 'umd.edu', 'umd.', 'edu' )
```

- Note that the ‘groups’ are numbered in a preorder traversal of the forest

re.findall()

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- When we use re.findall() it returns a list of zero or more sub-strings that match the regular expression

```
>>> import re
>>> x = 'My 2 favorite numbers are 19 and 42’
>>> y = re.findall('[0-9]+',x)
>>> print y
['2', '19', '42']
>>> y = re.findall('[AEIOU]+',x)
>>> print y
[]
```

More re functions

- re.split() is like split but can use patterns

```
>>> re.split("\W+", “This... is a test,  short and sweet, of split().”)
['This', 'is', 'a', 'test', 'short',  'and', 'sweet', 'of', 'split', '']
```

- re.sub() substitutes one string for a pattern

```
>>> re.sub('(blue|white|red)', 'black', 'blue socks and red shoes')
'black socks and black shoes’
```

- re.findall() finds all matches

```
>>> re.findall("\d+","12 dogs,11 cats, 1 egg")
['12', '11', '1']
```

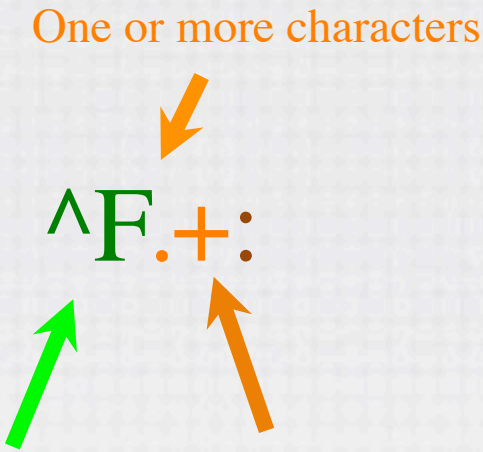
Warning: greedy matching

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- The repeat characters (* and +) push outward in both directions (greedy) to match the largest possible string

```
>>> import re
>>> x = 'From: Using the : character'
>>> y = re.findall('^F.+:', x)
>>> print y
['From: Using the :']
```

Why not 'From:'?

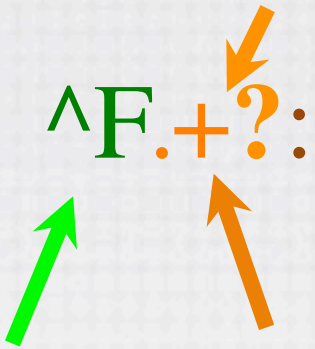


Non-greedy matching

- Not all regular expression repeat codes are greedy. If you add a ? Character after + and *, then it is non-greedy matching

```
>>> import re
>>> x = 'From: Using the : character'
>>> y = re.findall('^F.+?:', x)
>>> print y
['From:']
```

One or more characters but non-greedy



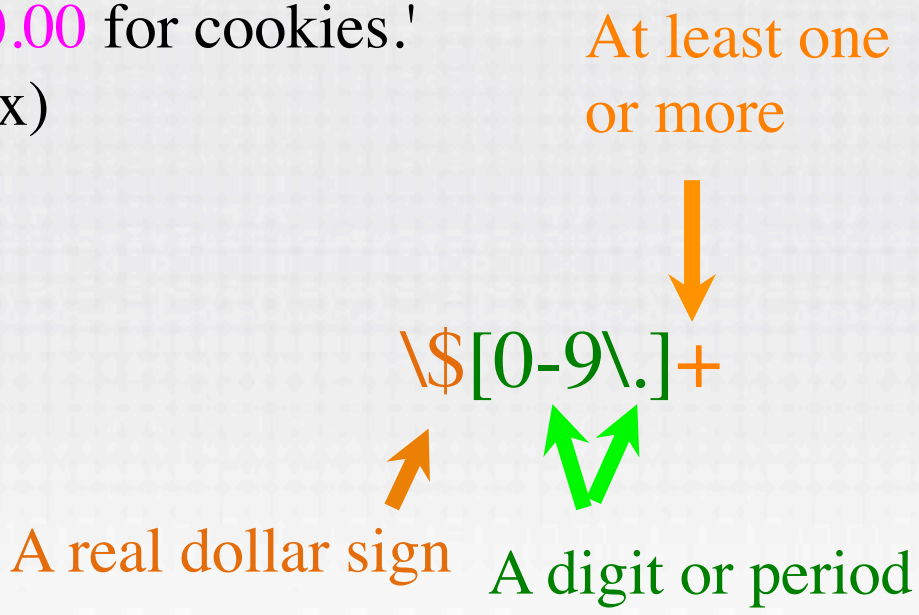
First character in the match is an F

Last character in the match is a :

Escape Character

- If you want a special regular expression character to just behave normally (most of the time) you prefix it with '\'

```
>>> import re
>>> x = 'We just received $10.00 for cookies.'
>>> y = re.findall('\$[0-9\.]+',x)
>>> print y
['$10.00']
```



Regular expression quick guide

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
^          Matches the beginning of a line
$          Matches the end of the line
.          Matches any character
\s         Matches whitespace
\S         Matches any non-whitespace character
*          Repeats a character zero or more times
*?         Repeats a character zero or more times (non-greedy)
+          Repeats a chracter one or more times
+?         Repeats a character one or more times (non-greedy)
[aeiou]    Matches a single character in the listed set
[^XYZ]     Matches a single character not in the listed set
[a-z0-9]   The set of characters can include a range
(          Indicates where string extraction is to start
)          Indicates where string extraction is to end
```

Natural language toolkit

- Natural language toolkit (NLTK) package ("import nltk") contains data and a number of functions.
  - nltk.corpus: standard natural language processing corpora
  - nltk.tokenize, nltk.stem: sentence and words segmentation and stemming or lemmatization
  - nltk.tag: part-of-speech tagging
  - nltk.classify, nltk.cluster: supervised and unsupervised classification
  - And many others…

Splitting words: word tokenization

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
>>> s = """To suppose that the eye with all its inimitable contrivances
for adjusting the focus to different distances, for admitting
different amounts of light, and for the correction of spherical and
chromatic aberration, could have been formed by natural selection,
seems, I freely confess, absurd in the highest degree."""

>>> s.split()
['To', 'suppose', 'that', 'the', 'eye', 'with', 'all', 'its',
'inimitable', 'contrivances', 'for', 'adjusting', 'the', 'focus',
'to', 'different', 'distances,', ...

>>> re.split('\W+', s)                    # Split on non-alphanumeric
['To', 'suppose', 'that', 'the', 'eye', 'with', 'all', 'its',
'inimitable', 'contrivances', 'for', 'adjusting', 'the', 'focus',
'to', 'different', 'distances', 'for',
```

Word normalization

- Converting “talking”, ”talk”, talked”, “Talk”, etc. to the lexeme “talk”

```
>>> porter = nltk.PorterStemmer()
>>> [porter.stem(t.lower()) for t in tokens]
['to', 'suppos', 'that', 'the', 'eye', 'with', 'all', 'it', 'inimit',
'contriv', 'for', 'adjust', 'the', 'focu', 'to', 'differ', 'distanc',
',', 'for', 'admit', 'differ', 'amount', 'of', 'light', ',', 'and',
```

- Another stemmer is lancaster.stem()
- The Snowball stemmer works for non-English, e.g.,

```
>>> from nltk.stem.snowball import SnowballStemmer
>>> stemmer = SnowballStemmer("danish")
>>> stemmer.stem('universiteterne')
'universitet'
```

Word normalization

- Normalize with a word list (WordNet)

```
>>> wnl = nltk.WordNetLemmatizer()
>>> [wnl.lemmatize(token) for token in tokens]
['To', 'suppose', 'that', 'the', 'eye', 'with', 'all', 'it',
'inimitable', 'contrivance', 'for', 'adjusting', 'the', 'focus', 'to',
'different', 'distance', ',', 'for', 'admitting', 'different',
'amount', 'of', 'light', ',', 'and', 'for', 'the', 'correction',
```

- Here words “contrivances” and “distances” have lost the plural “s” and “its” the genitive “s”

Word categories

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Part-of-speech tagging with NLTK

```
>>> words = nltk.word_tokenize(s)
>>> nltk.pos_tag(words)
[('To', 'TO'), ('suppose', 'VB'), ('that', 'IN'), ('the', 'DT'),
('eye', 'NN'), ('with', 'IN'), ('all', 'DT'), ('its', 'PRP$'),
('inimitable', 'JJ'), ('contrivances', 'NNS'), ('for', 'IN'),
```

NN noun, VB verb, JJ adjective, RB adverb, etc., see, common tags.

```
>>> tagged = nltk.pos_tag(words)
>>> [word for (word, tag) in tagged if tag=='JJ']
['inimitable', 'different', 'different', 'light', 'spherical',
'chromatic', 'natural', 'confess', 'absurd']
```

"confess" is wrongly tagged.