Chapter: RE

Regular Expression

Text matching patterns
Analogous to wild cards like *
Has a formal syntax with almost global acceptance
Almost all programming languages have regular expression support
Short hand – regex or regexp

re module of Python

regexp support provided through the “re” module

```
>>> import re
>>> pattern = "text"
>>> sometext = "Is this found here"
>>> if re.search(pattern, sometext):
        print "Found a match"
else:
        print "Not Found“
```

Return value of re.search

In the previous example what object did re.search return ? – match object
match object is a storehouse of information

```
>>> print match.re.pattern
this
>>> print match.string
Is this found here
>>> print match.start()
3
>>> print match.end()
7
```

Pre-compiling patterns (regexp object)

Compiled patterns lead to faster searches
Repeated patterns can be cached. In the previous case caching is limited

```
>>> reo = re.compile(pattern)
>>> m = reo.search(sometext)
>>> m.start(), m.end()
(3, 7)
```

Multiple matches

findall instead of search

```
>>> pattern = 'xy'
>>> string = 'xxyyxyxyxyxy'
>>> re.findall(pattern, string)
['xy', 'xy', 'xy', 'xy', 'xy']
>>> for m in re.finditer(pattern, string):
        print "found at", m.start()


found at 1
found at 4
found at 6
found at 8
found at 10
```

More complex patterns

Pattern syntax = metacharacters + literal text

Repetition

Complete list of metacharacters

. ^ $ * + ? { } [ ] \ | ( )

Some special sequences

\d \D \s \S \w \W

Literal repetition

a followed by zero or more b
'ab*'

a followed by one or more b
'ab+‘

a followed by zero or one b
'ab?‘

a followed by three b
'ab{3}‘

a followed by two to three b
'ab{2,3}'

Literal repetition

```
>>> re.findall('xy*', string)
['x', 'xyy', 'xy', 'xy', 'xy', 'xy']

>>> re.findall('xy+', string)
['xyy', 'xy', 'xy', 'xy', 'xy']

>>> re.findall('xy?', string)
['x', 'xy', 'xy', 'xy', 'xy', 'xy']

>>> re.findall('xy{3}', string)
[]

>>> re.findall('xy{1,2}', string)
['xyy', 'xy', 'xy', 'xy', 'xy']
```

Character set

either a or b
'[ab]'

followed by one or more a or b
'a[ab]+‘

a followed by one or more a or b, not greedy
a 'a[ab]+?’

Character Set

```
>>> re.findall('[xy]', string)
['x', 'x', 'y', 'y', 'x', 'y', 'x', 'y', 'x', 'y', 'x', 'y']

>>> re.findall('x[xy]+', string)
['xxyyxyxyxyxy']

>>> re.findall('x[xy]+?', string)
['xx', 'xy', 'xy', 'xy', 'xy']
```

Positioning

start of string, or line
^

end of string, or line
$

start of string
\A

end of string
\Z

Examples 1

```
>>> re.findall('^[0-9]', '0abs')
['0']
>>> re.findall('^[0-9]+', '0abs')
['0']
>>> re.findall('^[0-9]+', 'what')
[]
>>> re.findall('[0-9]$', 'abs0')
['0']
>>> re.findall('[0-9]$', 'abs')
[]
```

Grouping

‘a' followed by literal 'ab'
'a(ab)‘

'a' followed by 0-n 'a' and 0-n 'b'
'a(a*b*)‘

‘a' followed by 0-n 'ab'
'a(ab)*‘

‘a' followed by 1-n 'ab’
'a(ab)+'

Common regex

```
# username
^[a-z0-9_-]{3,16}$

#color hex code
^#?([a-f0-9]{6}|[a-f0-9]{3})$

#email
^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$
```

Examples 1

```
>>> re.findall('^[0-9]', '0abs')
['0']
>>> re.findall('^[0-9]+', '0abs')
['0']
>>> re.findall('^[0-9]+', 'what')
[]
>>> re.findall('[0-9]$', 'abs0')
['0']
>>> re.findall('[0-9]$', 'abs')
[]
```

Working of a Regex engine

- Character by character and left to right -> always
- Remembers previous success state
- Can backtrack
- The onus is on the developer to write efficient reg ex
- Patterns are basically directed graphs

Examples and gotchas

- Search for the words man and mango
- Text : man eats mango
- Motivation :
  - 1. patterns with subpatterns
  - 2. wasting of successful matches

Examples and gotchas

- Sentences ending with number and extract the number
- Text : class strength is 18. Only 12 available
- Lets start with : **.+(\d+)[.]**
- Motivation :
  - 1. Greedy patterns consuming the whole string
  - 2. Greedy and lazy patterns
  - 3. Backtracking on text and pattern

Examples and gotchas

- Indexed Groups
- Named Groups
- Ex : 10-05-2017
- Only indexed access : (\d{2})-(\d{2})-(\d{4})
- Naming a group : (**?P<day>**\d{2})-(**?P<mon>**\d{2})-(**?P<year>**\d{4})

Substitution

```
>>> re.sub(r"monday", r"Monday", "monday is the beginning of the week")
'Monday is the beginning of the week‘
>>> re.sub(r"(\d+)\s(\d+)", r"\1:\2", "1234 4567 and something else ")
'1234:4567 and something else with something else'
>>> re.sub(r"(\d{4})-(\d{2})-(\d{2})", r"\3-\2-\1", "2010-23-03")
'03-23-2010'
>>> re.sub(r"(?P<year>\d{4})-(?P<day>\d{2})-(?P<month>\d{2})",
r"\g<day>-\g<month>-\g<year>", "2010-23-03")
'23-03-2010'
```

Regex performance

- Motivation : Greedy groups can be very bad. Need to check performance for negative matches also
- Pattern : ^(\w*)*$
- Text : 1234567