Finishing Regular Expressions
&
XML / Web Scraping

Nov 10 2015

Today

- Iterators
- Do ACT 3-2
- Finish Regular Expressions
- XML Parsing in Python

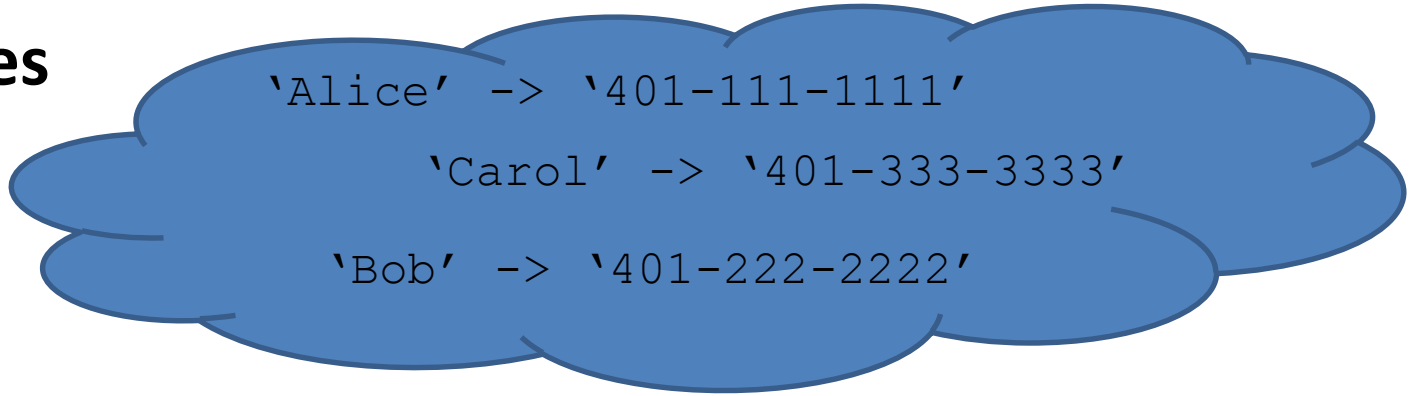
Data Structures

Lists

content	‘a’	‘b’	‘c’	‘d’	‘e’	‘f’	‘g’	‘h’	‘i’	‘j’
indices	0	1	2	3	4	5	6	7	8	9

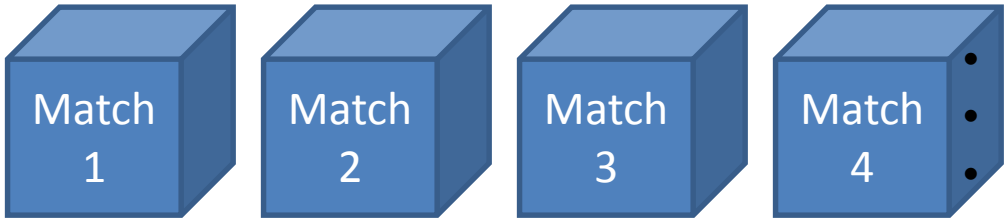
Dictionaries

keys & values



Iterators

Match Objects



- Matched String
- Matched String Start
- Matched String End

Iterators

The cat in the hat sat on a mat

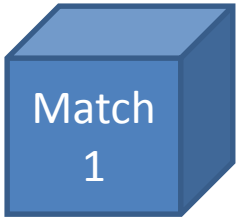
Regular Expression:'\wat'

Iterators

The cat in the hat sat on a mat

Regular Expression:'\wat'

Iterator

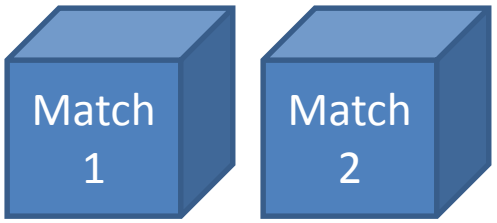


Iterators

The cat in the hat sat on a mat

Regular Expression:'\wat'

Iterator

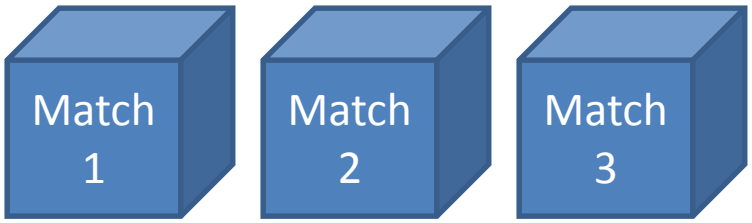


Iterators

The cat in the hat sat on a mat

Regular Expression: '\wat'

Iterator

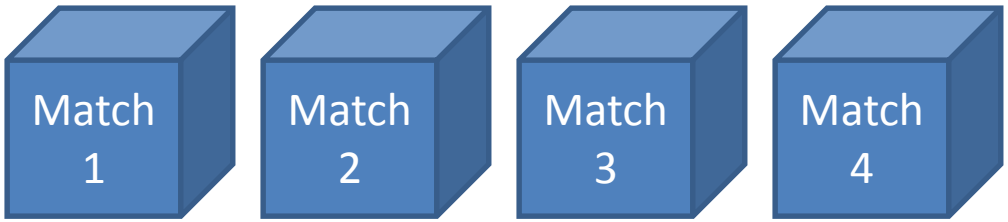


Iterators

The cat in the hat sat on a mat

Regular Expression:'\wat'

Iterator



Using Regular Expressions in Python

```
def printRegEx(regex,myStr):
    '''Prints all occurrences of the regular expression.'''

    myIter = re.finditer(regex,myStr) # Iterator!

    # This iterator contains MATCHES of the regex.
    # The following functions work on regex matches:
    #  group(0): returns the string that matches the regex
    #  start(0): the starting position of the string in myStr
    #  end(0): the ending position of the string in myStr

    # For loops work on iterators in addition to lists
    # Each match of the regex in myStr is stored in
    # a variable called 'occ'
    for occ in myIter:
        print('matches',occ.group(0), \
            'at positions',occ.start(0),'-',occ.end(0))
    return
```

Search using RegEx in P

- To use, type `import re`
  Python code, then functio
  - `re.search('a\w+a',`
    - Evaluates to `True`
      - You can find 'anica' in the string
  - `re.search('x\w+i', 'mechanical')`
    - Evaluates to `False`
      - You cannot find any substring starting with 'x' and finishing with 'i' in the string

In groups, Do ACT 3-2 Task 1

Match using RegEx in P

- To use, type `import re`
  Python code, then functio
  - `re.match('a\w+a', 'a`
    - Evaluates to `False`
      - The word is not in the specified form
  - `re.match('a\w+a', 'abracadabra')`
    - Evaluates to `True`
      - The word is in the specified form

In groups, Do ACT 3-2 Task 2

Task 2

```
for word in myList:
    # print word only if it rhymes with 'ping pong'
```

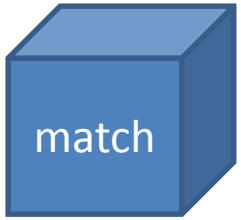
Task 2

```
for word in myList:
    match = re.match('\w*ing\s*\w*ong', word)
    if match:
        print(word)
```

Search and Match return MatchObjects

```
match = re.search('\s\w+\s', 'the cat jumped')
```

```
if match :
    print match.group(0)
```



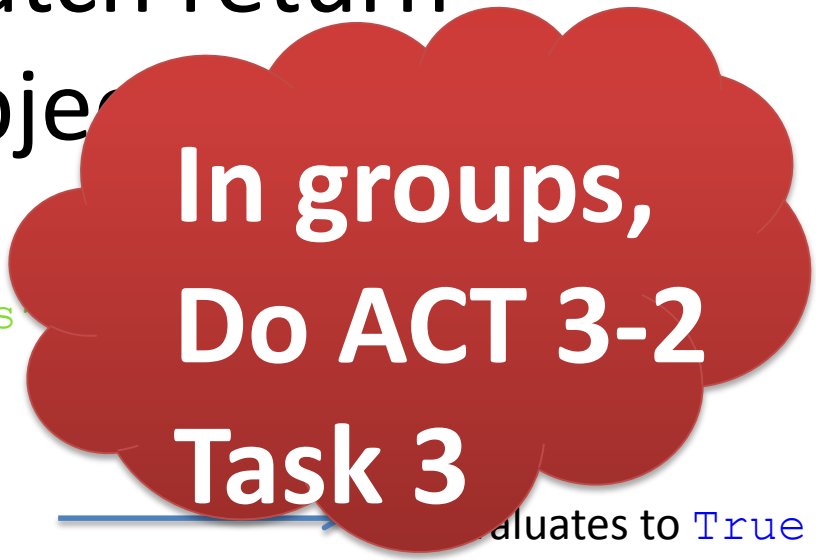
Evaluates to `True`

Gives you back what matched

```
' cat '
```

Not really, you don't want the spaces

Search and Match return MatchObje




```
match = re.search('\s(\w+)\s
```

aluates to `True`

```
if match :
    print match.group(0)
    print match.group(1)
```

Gives you back

0: the whole match

1…n: the corresp. parenthesis match

```
' cat '
'cat'
```

Task 3

```
for word in myList:
    match = re.match('(\w*ing)\s*(\w*ong)', word)
    if match:
        print('Whole match: ', match.group(0))
        print('First sub-match: ', match.group(1))
        print('Second sub-match: ', match.group(2))
```

Match Iterators

- Before, we just checked to see whether a pattern existed in the text...
- Or if the text started with a pattern

- Let’s now do some computing with those parts of the text that match

Data Structures

Lists

content	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	'i'	'j'
indices	0	1	2	3	4	5	6	7	8	9

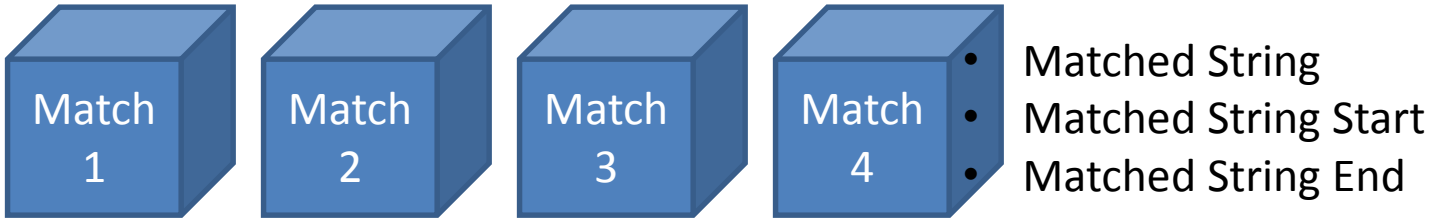
Dictionaries

keys & values

Key	Value
`'alice'`	`'401-111-1111'`
`'carol'`	`'401-222-2222'`
`'bob'`	`'401-333-3333'`

Iterators

Match Objects



Matching iterative


```
def printRegEx(regex,myStr):
    '''Prints all occurrences of the regular exp...'''

    match_iterator = re.finditer(regex,myStr) # Iterator!

    # This iterator contains MATCHES of the regex.
    # The following functions work for each match:
    #  group(0): returns the string that matches the regex
    #  start(0): the starting position of the string in myStr
    #  end(0): the ending position of the string in myStr

    # For loops work on iterators in addition to lists
    # Each match of the regex in is stored
    # in the variable 'match', in sequence
    for match in match_iterator:
        print('matches',match.group(0), \
            'at positions',match.start(0), '-', match.end(0))
    return
```

Pattern	Description
\s	Match a whitespace character (except newline, if you use the "re.MULTILINE" option).
\w	Match a letter, number, or underscore character.
\d	Match a digit.
^	Matches beginning of string (or beginning of line with the "re.MULTILINE" option).
$	Matches end of string (or end of line with the "re.MULTILINE" option).
.	Matches any single character except newline (also including newline with the "re.MULTILINE" option).
[aeiou]	Matches any single character in brackets.
[A-Z]	Matches any single character in the range indicated in the brackets.
[^aeiou]	Matches any single character not in brackets.
[aeiou]*	Matches 0 or more occurrences of preceding expression.
[aeiou]+	Matches 1 or more occurrence of preceding expression.
[aeiou]?	Matches 0 or 1 occurrence of preceding expression.
[aeiou]{n}	Matches exactly n number of occurrences of preceding expression.
[aeiou]{n,}	Matches n or more occurrences of preceding expression.
[aeiou]{n, m}	Matches at least n and at most m occurrences of preceding expression.
a|b	Matches either a or b.
([aeiou]+)	Groups regular expressions and remembers matched text.
…	…

Check how we got the words from the *Online Plain Text English Dictionary*

```
# opens output file for writing
outputFile = open(outputFilename, 'w’)

# for each letter
for letter in 'abcdefghijklmnopqrstuvwxyz':
  # download dictionary contents for the letter
  # urlLetter is the URL ‘constructed” based on it
  remoteFile = urllib.request.urlopen(urlLetter)
  contents = remoteFile.read()
  remoteFile.close()

  # collect all the words (made only by characters)
  # that are inside ‘<B> </B>’ and write to the output file
  regex = re.compile(b('<[Bb]>(\w+)</[Bb]>')
  for match in regex.finditer(contents):
    outputFile.write(match.group(1).decode('utf-8'))
    outputFile.write('\n')

outputFile.close()
```

group(1)…group(n) also work for each match in the iterator

re_tester.py

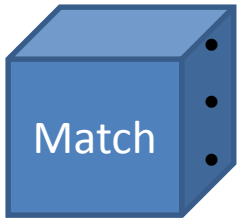
```
def showMatches(pattern, text):
    print('Text: ' + text)
    print('Patt: ' + pattern)
    match_iterator = re.finditer(pattern, text)
    for match in match_iterator:
        print('------- New Match -------')
        print('  match.start() = ' + str(match.start()))
        print('  match.end()   = ' + str(match.end()))
        print('  Whole match: ' + match.group(0))
        if match.lastindex is not None:
            print('  Match groups:')
            for i in range(1, match.lastindex+1):
                print('    match.group(' + str(i) + '): '
                      + match.group(i))
```

Match Iterators

- Compare the output of these expressions:
  - `showMatches('a\w+b.z', 'foobarbaz')`
  - `showMatches('(a)\w+(b.z)', 'foobarbaz')`

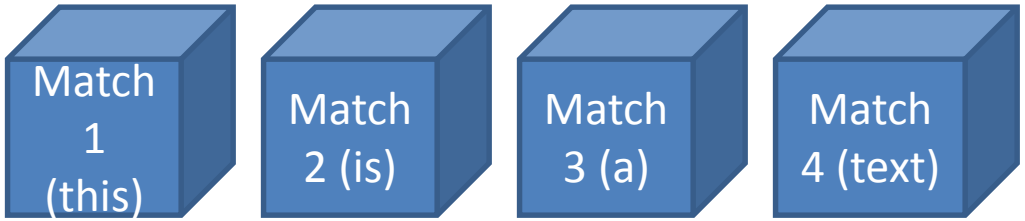
Search VS Match VS Iterator

- `re.search('a(\w+)a', 'mechanical')`
- `re.match('m(..)', 'mechanical')`



group(0) Matched string (in red)
start(0) or end(0) Start or Finishing position
group(1) Matched 1st parenthesis group (in **bold**)

- `re.finditer('\w+', 'This is a text.')`



Substitution using RegEx in Python

- `re.sub(pattern, replacement, string)`
  - Returns the *modified string*…
  - … with all occurrences of *pattern* substituted by *replacement*
- *Ex.:*
- `re.sub('\d', '', '123foo')`
  - Returns 'foo'
- `re.sub('\w+', '<word>', 'This is a text.')`
  - Returns '<word> <word> <word> <word>.'
- `re.sub('\n', ' – ', 'Line 1\nLine 2')`
  - Returns 'Line 1 – Line 2'

Splitting using RegEx in Python

- `re.split(pattern, string)`
  - Returns a *list of words*…
  - … separated by occurrences of *pattern*
- *Ex.:*
- `re.split('\s', 'Text with whitespaces.')`
  - Returns ['Text', 'with', 'whitespaces.']
- `re.split('\n', 'Line 1\nLine 2')`
  - Returns ['Line 1', 'Line 2']

In groups, do Task 5

Regex exercises

- Let's try to create the foll
  expressions:

- Capitalized words
- Hyphenated names
- Dates like 10/13/2014 or 4/4/03
- Words in quotes

Web Scraping Introduction

- Why isn't there a nice "importXML" in Python?

- *lxml* module
  - Not in the default installation
  - Install with "`easy_install lxml`" in your terminal (Mac OS X terminal or Windows console)
    - That's how you install any other non-default Python module

Web Scraping Introduction

Open the *Terminal* program and type this (example for Mac OS X):

Using lxml

```
import lxml.etree as et

filename = 'example.xml'

file = open(filename, 'r')
contents = file.read()
file.close()

tree = et.fromstring(contents)

for node in tree.xpath('//car'):
  for subnode in node.xpath('./year'):
    print(subnode.tag, ": ", subnode.text)
  for subnode in node.xpath('./color'):
    print(subnode.tag, ": ", subnode.text)
```

contents:

```
<inventory>
  <car>
    <year> 2010 </year>
    <color> black </color>
  </car>
  <car>
    <year> 2012 </year>
    <color> red </color>
  </car>
  <car>
    <year> 2014 </year>
    <color> yellow </color>
  </car>
</inventory>
```

Using lxml

```
import lxml.etree as et

filename = 'example.xml'

file = open(filename, 'r')
contents = file.read()
file.close()

tree = et.fromstring(contents)

for node in tree.xpath('//car'):
  for subnode in node.xpath('./year'):
    print(subnode.tag, ": ", subnode.text)
  for subnode in node.xpath('./color'):
    print(subnode.tag, ": ", subnode.text)
```

Internal rep. of the XML

contents:

```
<inventory>
  <car>
    <year> 2010 </year>
    <color> black </color>
  </car>
  <car>
    <year> 2012 </year>
    <color> red </color>
  </car>
  <car>
    <year> 2014 </year>
    <color> yellow </color>
  </car>
</inventory>
```

Using lxml

```
import lxml.etree as et

filename = 'example.xml'

file = open(filename, 'r')
contents = file.read()
file.close()

tree = et.fromstring(contents)

for node in tree.xpath('//car'):
    for subnode in node.xpath('./year'):
        print(subnode.tag, ": ", subnode.text)
    for subnode in node.xpath('./color'):
        print(subnode.tag, ": ", subnode.text)
```

Gives back **list** of "car" nodes

contents:

```
<inventory>
  <car>
    <year> 2010 </year>
    <color> black </color>
  </car>
  <car>
    <year> 2012 </year>
    <color> red </color>
  </car>
  <car>
    <year> 2014 </year>
    <color> yellow </color>
  </car>
</inventory>
```

Using lxml

```
import lxml.etree as et

filename = 'example.xml'

file = open(filename, 'r')
contents = file.read()
file.close()

tree = et.fromstring(contents)

for node in tree.xpath('//car'):
    for subnode in node.xpath('./year'):
        print(subnode.tag, ": ", subnode.text)
    for subnode in node.xpath('./color'):
        print(subnode.tag, ": ", subnode.text)
```

Searches for “year” inside “ car

contents:

```
<inventory>
  <car>
    <year> 2010 </year>
    <color> black </color>
  </car>
  <car>
    <year> 2012 </year>
    <color> red </color>
  </car>
  <car>
    <year> 2014 </year>
    <color> yellow </color>
  </car>
</inventory>
```

Using lxml

```
import lxml.etree as et

filename = 'example.xml'

file = open(filename, 'r')
contents = file.read()
file.close()

tree = et.fromstring(contents)

for node in tree.xpath('//car'):
    for subnode in node.xpath('./year'):
        print(subnode.tag, ": ", subnode.text)
    for subnode in node.xpath('./color'):
        print(subnode.tag, ": ", subnode.text)
```

Gives back **list** of " year" nodes

contents:

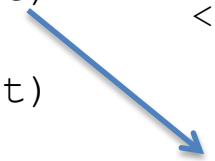
```
<inventory>
  <car>
    <year> 2010 </year>
    <color> black </color>
  </car>
  <car>
    <year> 2012 </year>
    <color> red </color>
  </car>
  <car>
    <year> 2014 </year>
    <color> yellow </color>
  </car>
</inventory>
```

Using lxml

```
import lxml.etree as et

filename = 'example.xml'

file = open(filename, 'r')
contents = file.read()
file.close()

tree = et.fromstring(contents)

for node in tree.xpath('//car'):
  for subnode in node.xpath('./year'):
    print(subnode.tag, ": ", subnode.text)
  for subnode in node.xpath('./color'):
    print(subnode.tag, ": ", subnode.text)
```

contents:

```
<inventory>
  <car>
    <year> 2010 </year>
    <color> black </color>
  </car>
  <car>
    <year> 2012 </year>
    <color> red </color>
  </car>
  <car>
    <year> 2014 </year>
    <color> yellow </color>
  </car>
</inventory>
```



subnode.text gives back text
subnode.tag gives back tag

Using lxml

```
import lxml.etree as et

filename = 'example.xml'

file = open(filename, 'r')
contents = file.read()
file.close()

tree = et.fromstring(contents)

for node in tree.xpath('//car'):
  for subnode in node.xpath('./year'):
    print(subnode.tag, ": ", subnode.text)
  for subnode in node.xpath('./color'):
    print(subnode.tag, ": ", subnode.text)
```

contents:

```
<inventory>
  <car c='good' h='10'>
    <year> 2010 </year>
    <color> black </color>
  </car>
  <car>
    <year> 2012 </year>
    <color> red </color>
  </car>
  <car>
    <year> 2014 </year>
    <color> yellow </color>
  </car>
</inventory>
```

node.attrib gives back attribute dictionary:

```
{'c': 'good', 'h':'10}
```

Getting data from the web

1) Use urllib.request

```
url = 'http://www.example.com/cars.xml'

remoteFile = urllib.request.urlopen(url)
contents = remoteFile.read()
remoteFile.close()
```

2) Use lxml

```
for node in tree.xpath('//car'):
    print node.text
    print node.tag
    attributes = node.attrib
    for key in attributes.keys:
        print(key, attributes[key])
```

Show me how that's useful!

- Go to:
  - https://maps.googleapis.com/maps/api/geocode/xml?address=115+Waterman+Street,+Providence,+RI

- This is Google letting you:
  - Give an address
  - Get back an XML containing latitude/longitude

```
<location>
  <lat>41.8271609</lat>
  <lng>-71.3995390</lng>
</location>
```

Show me how that's useful!

- Go to:
  - https://maps.googleapis.com/maps/api/geocode/xml?latlng=41.826273,-71.401628

- This is Google letting you:
  - Give latitude/longitude
  - Get back an XML with address information

```
<address_component>
  <long_name>Brown University</long_name>
  <short_name>Brown University</short_name>
  <type>establishment</type>
</address_component>
<address_component>
  <long_name>George Street</long_name>
  <short_name>George St</short_name>
  <type>route</type>
</address_component>
<address_component>
  <long_name>College Hill</long_name>
  <short_name>College Hill</short_name>
  <type>neighborhood</type>
  <type>political</type>
```

This happens a lot!

- Web services *expose functionality* via *XML*
- Or via something called *JSON*
  - Even *more convenient*!

- Stay tuned!
- Next class we *interact with Google Maps!*