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

<table>
<thead>
<tr>
<th>content</th>
<th>'a'</th>
<th>'b'</th>
<th>'c'</th>
<th>'d'</th>
<th>'e'</th>
<th>'f'</th>
<th>'g'</th>
<th>'h'</th>
<th>'i'</th>
<th>'j'</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Dictionaries

- 'Alice' -> '401-111-1111'
- 'Carol' -> '401-333-3333'
- 'Bob' -> '401-222-2222'

Iterators

Match Objects

- Match 1
- Match 2
- Match 3
- Match 4
  - 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`
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`
Iterators

The cat in the hat sat on a mat

Regular Expression: \`\wat\`

Iterator

Match 1  Match 2  Match 3  Match 4
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 Python

• To use, type `import re` at the top of your Python code, then functions like

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

- To use, type `import re` in your Python code, then functions like
  - `re.match('a\w+a', 'foobarbaz')`
    - 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

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

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

Evaluates to True

Gives you back
0: the whole match
1...n: the corresp. parenthesis match

\'cat\'
\'cat\'
Task 3

```python
for word in myList:
    match = re.match(r'\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

<table>
<thead>
<tr>
<th>content indices</th>
<th>‘a’</th>
<th>‘b’</th>
<th>‘c’</th>
<th>‘d’</th>
<th>‘e’</th>
<th>‘f’</th>
<th>‘g’</th>
<th>‘h’</th>
<th>‘i’</th>
<th>‘j’</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

### Dictionaries

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘alice’</td>
<td>‘401-111-1111’</td>
</tr>
<tr>
<td>‘carol’</td>
<td>‘401-222-2222’</td>
</tr>
<tr>
<td>‘bob’</td>
<td>‘401-333-3333’</td>
</tr>
</tbody>
</table>

### Iterators

**Match Objects**
- Match 1
- Match 2
- Match 3
- Match 4
  - Matched String
  - Matched String Start
  - Matched String End
def printRegEx(regex, myStr):
    '''Prints all occurrences of the regular expression.'''

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

outputFile.close()
group(1)...group(n) also work for each match in the iterator

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', 'foobazb')`
  – `showMatches('a\w+(b.z)', 'foobazb')`
Search VS Match VS Iterator

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

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

**Match 1** (this)  
**Match 2** (is)  
**Match 3** (a)  
**Match 4** (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(r'\d', '', '123foo')`
  - Returns ‘foo’
- `re.sub(r'\w+', '<word>', 'This is a text.')`
  - Returns ‘<word> <word> <word> <word>.’
- `re.sub(r'\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...

**Ex.:**
- `re.split(\s', 'Text with whitespaces.')`  
  - Returns ['Text', 'with', 'whitespaces.]
- `re.split(\n', 'Line 1\nLine 2')`  
  - Returns ['Line 1', 'Line 2']
Regex exercises

• Let’s try to create the following regular expressions:
  
• Capitalized words
• Hyphenated names
• Dates like 10/13/2014 or 4/4/03
• Words in quotes

In groups, do Task 6
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

```python
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)
```

```
<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

```python
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

```python
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

```xml
<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>
```
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)
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>

Gives back list of "year" nodes
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

```python
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)

def node.attrib:
    print('gives back attribute dictionary: ', node.attrib)
```

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>
```
Getting data from the web

1) Use urllib.request

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

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

2) Use lxml

```python
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:

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