ESCI 386 – Scientific Programming, Analysis and Visualization with Python

Lesson 8 – Regular Expressions
Regular Expressions

• Regular expressions allow for much more flexible and complex searches of patterns within strings.

• To use regular expressions in Python you must import the re module.

• The search patterns in regular expressions are a combination of text and special characters.
The `re.findall(p, s)` method accepts a pattern `p` and a string `s`, and returns a list with all the matches of the pattern.

```python
>>> import re
>>> re.findall(r'H', 'Honolulu, Hawaii')
['H', 'H']
>>> re.findall(r'lu', 'Honolulu, Hawaii')
['lu', 'lu']
>>> re.findall(r'e', 'Honolulu, Hawaii')
[]
```
Patterns

• The pattern for a regular expression should always be expressed as a raw string!
  – r’pattern’, not ‘pattern’
The Matches are Non-overlapping

```python
>>> re.findall(r'xax', 'xaxaxaxax')
['xax', 'xax']
```
**Syntax for Regular Expression Patterns**

<table>
<thead>
<tr>
<th>Char</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Matches any character other than the newline character. &lt;br&gt;s = 'mouse'&lt;br&gt;re.findall(r'.', s) =&gt; ['m', 'o', 'u', 's', 'e']</td>
</tr>
<tr>
<td>\A</td>
<td>Matches only at the start of the string &lt;br&gt;s = 'omnivore omnibus'&lt;br&gt;re.findall(r'om', s) =&gt; ['om', 'om']&lt;br&gt;re.findall(r'^om', s) =&gt; ['om']&lt;br&gt;re.findall(r'^vore', s) =&gt; []</td>
</tr>
<tr>
<td>\Z</td>
<td>Matches at the end of the string &lt;br&gt;s = 'omnivore omnibus'&lt;br&gt;re.findall(r'bus$', s) =&gt; ['bus']&lt;br&gt;re.findall(r'omni$', s) =&gt; []</td>
</tr>
<tr>
<td>or ^</td>
<td>Matches only at the start of the string &lt;br&gt;s = 'omnivore omnibus'&lt;br&gt;re.findall(r'om', s) =&gt; ['om', 'om']&lt;br&gt;re.findall(r'^om', s) =&gt; ['om']&lt;br&gt;re.findall(r'^vore', s) =&gt; []</td>
</tr>
<tr>
<td>or $</td>
<td>Matches at the end of the string &lt;br&gt;s = 'omnivore omnibus'&lt;br&gt;re.findall(r'bus$', s) =&gt; ['bus']&lt;br&gt;re.findall(r'omni$', s) =&gt; []</td>
</tr>
</tbody>
</table>
# Syntax for Regular Expression Patterns

<table>
<thead>
<tr>
<th>Char</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Matches zero or more repetitions of the preceding character or expression, matching as many repetitions as possible.</td>
</tr>
<tr>
<td></td>
<td>\textit{s} = 'xxoxooxooxoooo'</td>
</tr>
<tr>
<td></td>
<td>\textit{re.findall(r'xo*', s)} =&gt; ['x', 'xo', 'xoo', 'xooo', 'xoooo']</td>
</tr>
<tr>
<td>+</td>
<td>Matches one or more repetitions of the preceding character or expression, matching as many repetitions as possible.</td>
</tr>
<tr>
<td></td>
<td>\textit{s} = 'xxoxooxooxoooo'</td>
</tr>
<tr>
<td></td>
<td>\textit{re.findall(r'xo+', s)} =&gt; ['xo', 'xoo', 'xooo', 'xoooo']</td>
</tr>
<tr>
<td></td>
<td>\textit{re.findall(r'(xo)+', s)} =&gt; ['xo', 'xo', 'xo']</td>
</tr>
<tr>
<td></td>
<td>\textit{Note the difference between xo+ and (xo)+ here. xo+ matches an x followed by at least one o, and returns the x and all the trailing o's. (xo)+ only returns the x and a single trailing o.}</td>
</tr>
<tr>
<td>?</td>
<td>Matches zero or one repetitions of the preceding string.</td>
</tr>
<tr>
<td></td>
<td>\textit{s} = 'xxoxooxooxoooo'</td>
</tr>
<tr>
<td></td>
<td>\textit{re.findall(r'xo?', s)} =&gt; ['x', 'xo', 'xo', 'xo', 'xo']</td>
</tr>
<tr>
<td>*?</td>
<td>Matches zero or more repetitions of the preceding string, matching as few repetitions as possible.</td>
</tr>
<tr>
<td></td>
<td>\textit{s} = 'xxoxooxooxoooo'</td>
</tr>
<tr>
<td></td>
<td>\textit{re.findall(r'xo*?', s)} =&gt; ['x', 'x', 'x', 'x', 'x']</td>
</tr>
</tbody>
</table>
# Syntax for Regular Expression Patterns

<table>
<thead>
<tr>
<th>Char</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>+?</td>
<td>Matches one or more repetitions of the preceding string, matching as few repetitions as possible.</td>
</tr>
<tr>
<td></td>
<td>(s = 'xxoxooxooxooxoooo')</td>
</tr>
<tr>
<td></td>
<td><code>re.findall(r'xo+?', s) =&gt; ['xo', 'xo', 'xo', 'xo']</code></td>
</tr>
<tr>
<td>??</td>
<td>Matches zero or one repetitions of the preceding string, matching as few repetitions as possible.</td>
</tr>
<tr>
<td></td>
<td>(s = 'xxoxooxooxooxoooo')</td>
</tr>
<tr>
<td></td>
<td><code>re.findall(r'xo??', s) =&gt; ['x', 'x', 'x', 'x', 'x']</code></td>
</tr>
<tr>
<td>{}</td>
<td>Matches (m) repetitions of the preceding character or expression</td>
</tr>
<tr>
<td></td>
<td>(s = 'xxoxooxooxooxoooo')</td>
</tr>
<tr>
<td></td>
<td><code>re.findall(r'xo{2}', s) =&gt; ['xoo', 'xoo', 'xoo']</code></td>
</tr>
<tr>
<td>{(m,n)}</td>
<td>Matches (m) to (n) repetitions of the preceding character or expression.</td>
</tr>
<tr>
<td></td>
<td>(s = 'xxoxooxooxooxoooo')</td>
</tr>
<tr>
<td></td>
<td><code>re.findall(r'xo{1,3}', s) =&gt; ['xo', 'xoo', 'xooo', 'xoooo']</code></td>
</tr>
<tr>
<td>{(n)}</td>
<td>Matches 0 to (n) repetitions of the preceding character or expression.</td>
</tr>
<tr>
<td></td>
<td>(s = 'xxoxooxooxooxoooo')</td>
</tr>
<tr>
<td></td>
<td><code>re.findall(r'xo{3}', s) =&gt; ['x', 'xo', 'xoo', 'xooo', 'xooo']</code></td>
</tr>
<tr>
<td>{(m,}}</td>
<td>Matches (m) to (\infty) repetitions of the preceding character or expression.</td>
</tr>
<tr>
<td></td>
<td>(s = 'xxoxooxooxooxoooo')</td>
</tr>
<tr>
<td></td>
<td><code>re.findall(r'xo{2,}', s) =&gt; ['xoo', 'xooo', 'xoooo']</code></td>
</tr>
</tbody>
</table>
## Syntax for Regular Expression Patterns

<table>
<thead>
<tr>
<th>Char</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>Matches the beginning or end of a word</td>
</tr>
<tr>
<td></td>
<td>s = 'Honolulu luau'</td>
</tr>
<tr>
<td></td>
<td>re.findall(r'lu', s) =&gt; ['lu', 'lu', 'lu']</td>
</tr>
<tr>
<td></td>
<td>re.findall(r'\blu', s) =&gt; ['lu']</td>
</tr>
<tr>
<td></td>
<td>re.findall(r'lu\b', s) =&gt; ['lu']</td>
</tr>
<tr>
<td></td>
<td>re.findall(r'lu\b', s) =&gt; []</td>
</tr>
<tr>
<td>\d</td>
<td>Matches any decimal digit</td>
</tr>
<tr>
<td></td>
<td>s = 'abc123'</td>
</tr>
<tr>
<td></td>
<td>re.findall(r'\d', s) =&gt; ['1', '2', '3']</td>
</tr>
<tr>
<td>\D</td>
<td>Matches any non-digit</td>
</tr>
<tr>
<td></td>
<td>s = 'abc 1-2:3'</td>
</tr>
<tr>
<td></td>
<td>re.findall(r'\D', s) =&gt; ['a', 'b', 'c', ' ', '-', ':']</td>
</tr>
<tr>
<td>\s</td>
<td>Matches any whitespace character</td>
</tr>
<tr>
<td></td>
<td>s = 'abc 1-2:3'</td>
</tr>
<tr>
<td></td>
<td>re.findall(r'\s', s) =&gt; [' ']</td>
</tr>
<tr>
<td>\S</td>
<td>Matches any non-whitespace character</td>
</tr>
<tr>
<td></td>
<td>re.findall(r'\S', s) =&gt; ['a', 'b', 'c', '1', '-', '2', ':', '3']</td>
</tr>
<tr>
<td>\w</td>
<td>Matches any alphanumeric character (letters and numbers)</td>
</tr>
<tr>
<td></td>
<td>re.findall(r'\w', s) =&gt; ['a', 'b', 'c', '1', '2', '3']</td>
</tr>
<tr>
<td>\W</td>
<td>Matches any non-alphanumeric character</td>
</tr>
<tr>
<td></td>
<td>s = 'abc 1-2:3'</td>
</tr>
<tr>
<td></td>
<td>re.findall(r'\W', s) =&gt; [' ', '-', ':']</td>
</tr>
</tbody>
</table>
Matching Either-Or

• The pipe character | character has the meaning of or.

• Suppose we wanted to test whether a certain string contained either of the substrings ‘boo’ and ‘foo’. We can do this using the pattern r'(boo|foo)'. In this syntax the construct (x|y) means match either x or y.

```python
>>> s, t, u = 'foobar', 'boobar', 'moobar'
>>> re.findall(r'(boo|foo)', s)
['foo']
>>> re.findall(r'(boo|foo)', t)
['boo']
>>> re.findall(r'(boo|foo)', u)
[]
```
Using Groups

• Groups are contained within parentheses

• When groups are used, findall() returns a tuple containing the matches within the group only.

```python
>>> re.findall(r'is (blue|red)', 'My ball is blue.')
['blue']
>>> re.findall(r'is (blue|red)', 'My ball was blue.')
[]
```
Using Groups (cont.)

• If you wanted to return the entire match of the pattern, you must then enclose the whole pattern in parentheses.

• The tuple contains an item for each group in the pattern.

```python
>>> re.findall(r'(is (blue|red))', 'My ball is blue.')[
['is blue', 'blue']
]```
Another Example of Groups

```python
>>> s = 'June 26, 2013, 6:57 p.m., Aurora, Colorado'

>>> p = r'\b[12]\d{3}\b.*(\b\d{1,2}:\d{2}\s[ap]\.m\.)'  # Group for year and time

>>> result = re.findall(p, s)
>>> result
[('2013', '6:57 p.m.')]

>>> year, time = result[0]
>>> year
'2013'
>>> time
'6:57 p.m.'
```
Patterns are Just Strings

• Patterns can be saved to variables.

```python
>>> p = r'(boo|foo)'
>>> re.findall(p, s)
['foo']
```
Matching from a Set of Characters

• The use of square brackets [] allows us to define a set of characters for matching.

• So, to find all the vowels...

```python
>>> s = 'The quick brown fox.'
>>> re.findall(r'[a,e,i,o,u]', s)
['e', 'u', 'i', 'o', 'o']
```
Conjugate Set

• The ^ symbol used in a set means the conjugate set, or not.

• So, to find all the non-vowels...

```python
>>> s = 'The quick brown fox.'
>>> re.findall(r'[^a,e,i,o,u]',s)
['T', 'h', ' ', 'q', 'c', 'k', ' ', 'b', 'r', 'w', 'n', ' ', 'f', 'x', '.']
```
The Hyphen Indicates a Range

• To find all the digits from 2 to 6...

```python
>>> s = 'abc3 56 78 t45 m179'
>>> re.findall(r'[2-6]', s)
['3', '5', '6', '4', '5']
```
Example

• Suppose we are reading data and need to identify any fields that represent longitude or latitude.

• All such fields will consist of one to three digits followed by a decimal and then one or more digits followed by N, S, E, or W.

• We can use the regular expression pattern
  \[ p = r'\d{1,3}\.\d+[N,S,E,W]' \]
Example (cont.)

>>> p = r'\d{1,3}\d+[N,S,E,W]'
>>> s = '45.6   34   23.45S   34.67 934.56J   045.363E'
>>> re.findall(p,s)
['23.45S', '045.363E']
re.sub()

• This function is used to find the pattern in a string and replace it with an alternate substring.

```python
>>> s = '135abc821xyz'
>>> re.sub(r'[1-4]', 'x', s)  
'xx5abc8xxxyz'
```

Replace digits 1 thru 4 with x.

```python
>>> s = '   45   38   hello   ' 
>>> re.sub(r'\s{2,}', ' ', s) 
' 45 38 hello '  
```

Replace multiple white space with single space.
re.subn()

• Like re.sub(), but also returns number of substitutions made.

```python
>>> s = '135abc821xyz'
>>> re.subn(r'[1-4]', 'x', s)
('xx5abc8xxxyz', 4)
```
re.split()

• Splits a string based on a pattern.
Regular Expression Objects

• The re module also allows us to create a regular expression object from a pattern, using the `re.compile()` function
  
  \[1-5]+\s'
  
  `r = re.compile(p)`

• `r` is now a regular expression object that matches any digits between 1 and 5 that are also followed by a whitespace.
Regular Expression Methods

• Regular expression objects have methods such as `r.findall()`, `r.split()`, `r.sub()`, `r.subn()`, and other methods.

• These are similar to the `re.findall()`, `re.split()`, `re.sub()` and `re.subn()` functions.
Examples

```python
>>> p = '[1-5]+\s'
>>> r = re.compile(p)
>>> s = 'abc 34x 256 342 sx'
>>> r.findall(s)
['342 ']
>>> r.sub('x', s)
'abc 34x 256 xsx'
>>> r.subn('x', s)
('abc 34x 256 xsx', 1)
>>> r.split(s)
['abc 34x 256 ', 'sx']
```
re.findall() vs. r. findall()

• Almost anything that can be done with regular expression objects can also be done using the re module functions.

• The only place where regular expression objects have an advantage is for searches that are not case sensitive.
  – These can only be done by compiling a regular expression object.
Example

```python
>>> p = r'[a-c]'
>>> s = 'aBcD'
>>> re.findall(p, s)
['a', 'c']

>>> r = re.compile(p, re.I)
>>> r.findall(s)
['a', 'B', 'c']
```

Only finds lower-case matches.

Flag for case insensitivity.

Finds lower and upper-case matches.