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.

re.findall()

- 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

```
>>> 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

```
>>> re.findall(r'xax', 'xaxaxax')
['xax', 'xax']
```

Syntax for Regular Expression Patterns

Char	***Purpose***
.	Matches any character other than the newline character. s = 'mouse' re.findall(r'.', s) => ['m', 'o', 'u', 's', 'e']
\A or ^	Matches only at the start of the string s = 'omnivore omnibus' re.findall(r'om', s) => ['om', 'om'] re.findall(r'^om', s) => ['om'] re.finall(r'^vore',s) = > []
\Z or $	Matches at the end of the string s = 'omnivore omnibus' re.findall(r'bus$', s) => ['bus'] re.findall(r'omni$', s) => []

Syntax for Regular Expression Patterns

Char	***Purpose***
*	Matches zero or more repetitions of the preceding character or expression, matching as many repetitions as possible. s = 'xxoxooxoooxoooo' re.findall(r'xo*',s) => ['x', 'xo', 'xoo', 'xooo', 'xoooo']
+	Matches one or more repetitions of the preceding character or expression, matching as many repetitions as possible. s = 'xxoxooxoooxoooo' re.findall(r'xo+',s) => ['xo', 'xoo', 'xooo', 'xoooo'] re.findall(r'(xo)+', s) => ['xo', 'xo', 'xo'] 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.
?	Matches zero or one repetitions of the preceding string. s = 'xxoxooxoooxoooo' re.findall(r'xo?',s) => ['x', 'xo', 'xo', 'xo', 'xo']
*?	Matches zero or more repetitions of the preceding string, matching as few repetitions as possible. s = 'xxoxooxoooxoooo' re.findall(r'xo*?',s) = > ['x', 'x', 'x', 'x', 'x']

Syntax for Regular Expression Patterns

Char	*Purpose*
+?	Matches one or more repetitions of the preceding string, matching as few repetitions as possible. s = 'xxoxooxoooxoooo' re.findall(r'xo+?',s) => ['xo', 'xo', 'xo', 'xo']
??	Matches zero or one repetitions of the preceding string, matching as few repetitions as possible. s = 'xxoxooxoooxoooo' re.findall(r'xo??',s) => ['x', 'x', 'x', 'x', 'x']
{*m*}	Matches *m* repetitions of the preceding character or expression s = 'xxoxooxoooxoooo' re.findall(r'xo{2}', s) => ['xoo', 'xoo', 'xoo']
{*m,n*}	Matches *m* to *n* repetitions of the preceding character or expression. s = 'xxoxooxoooxoooo' re.findall(r'xo{1,3}', s) => ['xo', 'xoo', 'xooo', 'xooo']
{,*n*}	Matches 0 to *n* repetitions of the preceding character or expression. s = 'xxoxooxoooxoooo' re.findall(r'xo{,3}', s) => ['x', 'xo', 'xoo', 'xooo', 'xooo']
{*m*,}	Matches *m* to ∞ repetitions of the preceding character or expression. s = 'xxoxooxoooxoooo' re.findall(r'xo{2,}', s) => ['xoo', 'xooo', 'xoooo']

Syntax for Regular Expression Patterns

Char	***Purpose***
\b	Matches the beginning or end of a word
	s = 'Honolulu luau'
	re.findall(r'lu', s) => ['lu', 'lu', 'lu']
	re.findall(r'\blu', s) => ['lu']
	re.findall(r'lu\b', s) => ['lu']
	re.findall(r'b\no', s) => []
\d	Matches any decimal digit
	s = 'abc123'
	re.findall(r'\d', s) => ['1', '2', '3']
\D	Matches any non-digit
	s = 'abc 1-2:3'
	re.findall(r'\D', s) => ['a', 'b', 'c', ' ', '-', ':']
\s	Matches any whitespace character
	s = 'abc 1-2:3'
	re.findall(r'\s', s) => [' ']
\S	Matches any non-whitespace character
	re.findall(r'\S', s) => ['a', 'b', 'c', '1', '-', '2', ':', '3']
\w	Matches any alphanumeric character (letters and numbers)
	re.findall(r'\w', s) => ['a', 'b', 'c', '1', '2', '3']
\W	Matches any non-alphanumeric character
	s = 'abc 1-2:3'
	re.findall(r'\W', s) => [' ', '-', ':']

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.

```
>>> 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.

```
>>> 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.

```
>>> re.findall(r'(is (blue|red))', 'My ball is blue.')
[('is blue', 'blue')]
```

Another Example of Groups

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

>>> p = r'(\b[12]\d{3}),\s*(\b\d{1,2}:\d{2}\s[ap]\.m\.)'
```

Group for time

Group for year

```
>>> 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.

```
>>> 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…

```
>>> 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…

```
>>> 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…

```
>>> 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.

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

Replace digits 1 thru 4 with x.

```
>>> 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.

```
>>> 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

  ```
  p = '[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

```
>>> 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

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

Only finds lower-case matches.

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

Flag for case insensitivity.

Finds lower and upper-case matches.