Unit 2 – Standard Libraries

P.N. Anantharaman

18 Feb 2013

References

- Core Python Application Programming Third Edition by Wesley J Chun - Pearson

Text Processing with string functions

- Example: Removing leading and trailing spaces and characters

```
txt = ‘    hello world    ‘
print ‘***’, txt.lstrip(), ‘***’, txt.rstrip(), ‘***’, txt.strip(), ‘***’

x = 'xyxxyy hejyx  yyx‘
print '|'+x.strip('xy')+'|‘
```

| hejyx |

```
>>> x = 'www.example.com'
>>> print x.strip('w.')
```

example.com

- Example 1 - Controlling case

```
>>> abc = 'hello world'
>>> print abc.upper(), '|', abc.lower(), '|', abc.capitalize(), '|', abc.title()
HELLO WORLD | hello world | Hello world | Hello World
Example 2 – Indenting
>>> txt = """ line 1
                line 2
        line3
        """
>>> print txt
 line 1
                line 2
        line3
>>> def reindent(s, numspaces):
        leading_space = numspaces * ' '
        lines = [ leading_space + line.strip()
          for line in s.splitlines()]
        return '\n'.join(lines)
>>> print reindent(txt, 7)
       line 1
       line 2
       line3
```

```
>>> txt.find('line1')
-1
>>> txt.find('line 1')
1
>>> txt.find('line 2')
26
>>> 'line' in txt
True
>>> print txt.replace('line', 'fine')
 fine 1
                fine 2
        fine3
 >>> print txt.replace('line', 'fine', 2)
 fine 1
                fine 2
        line3
```

String examples contd

```
>>> txt.startswith(" line")
True
>>> txt.startswith("line")
False
>>> txt.startswith("line", 1)
True
>>> txt.endswith("line", 0, 5)
True
>>> txt.endswith("line", 0, 6)
False
```

String methods contd

str.isalnum()Return true if all characters in the string are alphanumeric and there is at least one character, false otherwise.

str.isalpha()Return true if all characters in the string are alphabetic and there is at least one character, false otherwise.

str.isdigit()Return true if all characters in the string are digits and there is at least one character, false otherwise.

str.islower()Return true if all cased characters in the string are lowercase and there is at least one cased character, false otherwise.

str.isspace()Return true if there are only whitespace characters in the string and there is at least one character, false otherwise.

str.istitle()Return true if the string is a titlecased string and there is at least one character, for example uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return false otherwise.

str.isupper()Return true if all cased characters in the string are uppercase and there is at least one cased character, false otherwise.

Why Regular Expressions?

- Python String processing functions are quite handy to address many text processing requirements.
  - Example: Removing leading and trailing white spaces
  - splitting a text in to constituent words etc ‘/usr/ananth/doc’.split(‘/’)
- In many situations, we may not know the exact input string that we need to process but we know the pattern that we are looking for. Regular expressions may be used to solve such problems.
  - Example: Extract the text out of a HTML file by filtering out the tags
  - Split a text using a pattern: re.split(‘,| ’, ‘split the text,space and comma’)
- Some Applications
  - Tokenization
  - Validation
  - Find/Replace
  - NLP

Search and Match in Python

- In Python there are 2 ways to perform pattern matching:
  - Matching
  - Searching
- Searching refers to looking for a pattern in any part of the base text
- Matching refers to attempting to match a pattern to an entire string starting from begining

Example

```
>>> txt = 'hello world'
>>> m = re.match('hello', txt)
>>> m.group()
'hello'
>>> m = re.match('world', txt)
>>> m.group()
Traceback (most recent call last):
  File "<pyshell#118>", line 1, in <module>
    m.group()
AttributeError: 'NoneType' object has no attribute 'group‘
>>> m = re.search('world', txt)
>>> m.group()
'world'
```

Example

```
>>> m = re.match('hello | world', txt)
>>> m.group()
'hello ‘
>>> m = re.match('.ello', txt)
>>> m.group()
'hello‘
>>> m = re.match('^hello', txt)
>>> m.group()
'hello‘
>>> m = re.match('^.ello', txt)
>>> m.group()
'hello'
```

Example

```
>>> m = re.search('d$', txt)
>>> m.group()
'd'
>>> m = re.search(' $', txt)
>>> m.group()
Traceback (most recent call last):
  File "<pyshell#142>", line 1, in <module>
    m.group()
AttributeError: 'NoneType' object has no attribute 'group‘
>>> m = re.match('hello .*d$', txt)
>>> m.group()
'hello world‘
>>> m = re.match('.*d$', txt)
>>> m.group()
'hello world'
```

Example

```
>>> m = re.match('.+d$', txt)
>>> m.group()
'hello world‘
>>> m = re.match('hello?', txt)
>>> m.group()
'hello'
>>> m = re.match('(hello)?', txt)
>>> m.group()
'hello'
>>> m.group(1)
'hello'
>>> m = re.match('helloa?', txt)
>>> m.group()
'hello'
```

Examples

```
>>> m = re.match('(hello){1}', txt)
>>> m.group()
'hello'
>>> m = re.match('(hello){2}', txt)
>>> m.group()

Traceback (most recent call last):
  File "<pyshell#159>", line 1, in <module>
    m.group()
AttributeError: 'NoneType' object has no attribute 'group'
>>> m = re.match('(hello){1,2}', txt)
>>> m.group()
'hello'
>>> m = re.match('hel{2}o', txt)
>>> m.group()
'hello'
```

Examples

```
>>> m = re.match('[hel]*lo', txt)
>>> m.group()
'hello'
>>> m = re.match('[a-o]*', txt)
>>> m.group()
'hello'
>>> m = re.match('[a-l]*lo', txt)
>>> m.group()
'hello'
>>> m = re.match('[a-l]+lo', txt)
>>> m.group()
'hello'
>>> m = re.match('[e-h]+lo', txt)
>>> m.group()
Traceback (most recent call last):
  File "<pyshell#175>", line 1, in <module>
    m.group()
AttributeError: 'NoneType' object has no attribute 'group‘
>>> m = re.match('h[e-l]+lo', txt)
>>> m.group()
'hello'
```

```
>>> m = re.match('[^p-z]', txt)
>>> m.group()
'h'
>>> m = re.match('h[^p-z]', txt)
>>> m.group()
'he‘
>>> m = re.match('h[^p-z]+', txt)
>>> m.group()
'hello '
>>> m = re.match('h[^p-z]+ world', txt)
>>> m.group()
'hello world'
```

```
>>> m = re.match('.*', txt)
>>> m.group()
'hello world'
>>> m = re.match('.*?', txt)
>>> m.group()
''
>>> m = re.match('.*?[a-z]', txt)
>>> m.group()
'h'
>>> m = re.match('.+?[a-z]', txt)
>>> m.group()
'he'
>>> m = re.match('.+?', txt)
>>> m.group()
'h'
```

Examples

```
>>> m = re.match('(.*)\s(.*)', txt)
>>> m.group()
'hello world'
>>> m.groups()
('hello', 'world')
>>> m.group(1)
'hello'
>>> m.group(2)
'world‘
>>> m = re.match('[^\d]', txt)
>>> m.group()
'h'
>>> m.groups()
()
>>> m = re.match('([^\d])', txt)
>>> m.groups()
('h',)
>>> m = re.match('(\d)', '1234')
>>> m.groups()
('1',)
```

Examples

```
>>> m = re.match('(\w)*', 'abcd    1234')
>>> m.groups()
('d',)
>>> m = re.match('(\w*)', 'abcd    1234')
>>> m.groups()
('abcd',)
>>> m = re.match('(\w*)', 'abcd1234')
>>> m.groups()
('abcd1234',)
>>> m = re.match('(\Aa)', 'abcd1234')
>>> m.groups()
('a',)
>>> m = re.match('\..*', '.txt')
>>> m.groups()
()
>>> m.group()
'.txt'
```

Extension Notation

(?iLmsux) – Embed one or more special flags within regex itself

>>> m = re.match('(HeLLo)(?i)', txt)

>>> m.group()

'hello‘

(?:…) - Signifies a group whose value is not saved

Example: (?:\w+\.)*

(?P<name>…) – Like a regular group match only identified with name than ID

Example: (?P<firstName>)

(?P=name…) – matches the name identified with name than ID

Example: (?P=firstName)

(?#...) – specifies a comment

>>> m = re.match('HeLLo(?i)(?#this is a comment!)', txt)

>>> m.group()

'hello'

Extension Notation Contd

(?=…) – Matches if … comes next without consuming the input string, called positive lookahead assertion. Example: (?=.com)

(?!…) – Matches if … doesn’t come next without consuming the input string, called negative lookahead assertion. Example: (?!.org)

(?<=…) – Matches if … comes prior without consuming the input string, called positive lookbehind assertion. Example: (?<=800-)

(?<!…) – Matches if … doesn’t come prior without consuming the input string, called negative lookbehind assertion. Example: (?<!192\.168\.)

(?(id/name)Y|N) – Conditional Match of regex Y if group with given id or name exists else N; |N is optional. Example: (?(1)y|x)

Exercises

- What are the strings matched by: b[aeiu]t
  - bat, bit, bet, but
- What are the strings to be matched by:

[cr][23][dp][02]

- What is the regular expression to match pin codes in India?
- What is the regular expression to check a 6 letter password that has at least one capital letter and one numeral? Example: h1B1Ac
- Write the regular expression to recognize dates that are written of the form: 1 Jan 12, 01 Jan 12, 01 Jan 2012 and so on
- What does the reg ex </?[^>]+> match?
- What does the reg ex \d+(\.\d*)? match?
- Write a reg ex to extract the form: title firstname and optional lastname. Eg: Dr Abdul Kalam, Smt Roopa, Sri Sachin Tendulkar, Ms Geetha

Python Functions for regular expression processing

- compile(pattern, flags = 0)
- match(pattern, string, flags = 0)
- search(pattern, string, flags = 0)
- findall(pattern, string, [,flags])
- finditer(pattern, string, [,flags])
- split(pattern, string, max=0)
- sub(pattern, repl, string, count = 0)
- purge()