Python - Regular Expressions

A *regular expression* is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are widely used in UNIX world.

Group Comparison

```
# (...) group a regular expression
>>> m = re.search(r'(\d{4})-(\d{2})-(\d{2})', '2016-01-01')
>>> m
<_sre.SRE_Match object; span=(0, 10), match='2016-01-01'>
>>> m.groups()
('2016', '01', '01')
>>> m.group()
'2016-01-01'
>>> m.group(1)
'2016'
>>> m.group(2)
'01'
>>> m.group(3)
'01'

# Nesting groups
>>> m = re.search(r'(((\d{4})-\d{2})-\d{2})', '2016-01-01')
>>> m.groups()
('2016-01-01', '2016-01', '2016')
>>> m.group()
'2016-01-01'
>>> m.group(1)
'2016-01-01'
>>> m.group(2)
'2016-01'
>>> m.group(3)
'2016'
```

Non capturing group

```
# non capturing group
>>> url = 'http://stackoverflow.com/'
>>> m = re.search('(?:http|ftp)://([^/\r\n]+)(/[^\r\n]*)?', url)
>>> m.groups()
('stackoverflow.com', '/')

# capturing group
>>> m = re.search('(http|ftp)://([^/\r\n]+)(/[^\r\n]*)?', url)
>>> m.groups()
('http', 'stackoverflow.com', '/')
```

Back Reference

```
# compare 'aa', 'bb'
>>> re.search(r'([a-z])\1$','aa') != None
True
>>> re.search(r'([a-z])\1$','bb') != None
True
>>> re.search(r'([a-z])\1$','ab') != None
False

# compare open tag and close tag
>>> pattern = r'<([^>]+)>[\s\S]*?</\1>'
>>> re.search(pattern, '<bold> test </bold>') != None
True
>>> re.search(pattern, '<h1> test </h1>') != None
True
>>> re.search(pattern, '<bold> test </h1>') != None
False
```

Named Grouping `(?P<name>)`

```
# group reference ``(?P<name>...)``
>>> pattern = '(?P<year>\d{4})-(?P<month>\d{2})-(?P<day>\d{2})'
>>> m = re.search(pattern, '2016-01-01')
>>> m.group('year')
'2016'
>>> m.group('month')
'01'
>>> m.group('day')
'01'

# back reference ``(?P=name)``
>>> re.search('^(?P<char>[a-z])(?P=char)','aa')
<_sre.SRE_Match object at 0x10ae0f288>
```

Substitute String

```
# basic substitute
>>> res = "1a2b3c"
>>> re.sub(r'[a-z]',' ', res)
'1 2 3 '

# substitute with group reference
>>> date = r'2016-01-01'
>>> re.sub(r'(\d{4})-(\d{2})-(\d{2})',r'\2/\3/\1/',date)
'01/01/2016/'

# camelcase to underscore
>>> def convert(s):
...     res = re.sub(r'(.)([A-Z][a-z]+)',r'\1_\2', s)
...     return re.sub(r'([a-z])([A-Z])',r'\1_\2', res).lower()
...
>>> convert('CamelCase')
'camel_case'
>>> convert('CamelCamelCase')
'camel_camel_case'
>>> convert('SimpleHTTPServer')
'simple_http_server'
```

notation	compare direction
(?=...)	left to right
(?!...)	left to right
(?<=...)	right to left
(?!<...)	right to left

```
# basic
>>> re.sub('(?=\d{3})', ' ', '12345')
' 1 2 345'
>>> re.sub('(?!\d{3})', ' ', '12345')
'123 4 5 '
>>> re.sub('(?<=\d{3})', ' ', '12345')
'123 4 5 '
>>> re.sub('(?<!\d{3})', ' ', '12345')
' 1 2 345'
```

Match common username or password

```
>>> re.match('^[a-zA-Z0-9-_]{3,16}$', 'Foo') is not None
True
>>> re.match('^\w|[-_]{3,16}$', 'Foo') is not None
True
```

Match hex color value

```
>>> re.match('^#?([a-f0-9]{6}|[a-f0-9]{3})$', '#ffffff')
<_sre.SRE_Match object at 0x10886f6c0>
>>> re.match('^#?([a-f0-9]{6}|[a-f0-9]{3})$', '#ffffh')
<_sre.SRE_Match object at 0x10886f288>
```

Match email

```
>>> re.match('^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$',
...          'hello.world@example.com')
<_sre.SRE_Match object at 0x1087a4d40>

# or

>>> exp = re.compile(r'''^([a-zA-Z0-9._%-]+@
...                    [a-zA-Z0-9.-]+
                       \.[a-zA-Z]{2,4})*$''', re.X)
>>> exp.match('hello.world@example.hello.com')
<_sre.SRE_Match object at 0x1083efd50>
>>> exp.match('hello%world@example.hello.com')
<_sre.SRE_Match object at 0x1083efeb8>
```

Match URL

```
>>> exp = re.compile(r'''^(https?:\/\/)? # match http or https
...              ([\da-z\.-]+)            # match domain
...              \.([a-z\.]{2,6})         # match domain
...              ([\/\w \.-]*)\/?$        # match api or file
...              ''', re.X)
>>>
>>> exp.match('www.google.com')
<_sre.SRE_Match object at 0x10f01ddf8>
>>> exp.match('http://www.example')
<_sre.SRE_Match object at 0x10f01dd50>
>>> exp.match('http://www.example/file.html')
<_sre.SRE_Match object at 0x10f01ddf8>
>>> exp.match('http://www.example/file!.html')
```

Match IP address

notation	description
(?:...)	Don't capture group
25[0-5]	Match 251-255 pattern
2[0-4][0-9]	Match 200-249 pattern
[1]?[0-9][0-9]	Match 0-199 pattern

```
>>> exp = re.compile(r'''^(?:(?:25[0-5]
...                       |2[0-4][0-9]
...                       |[1]?[0-9][0-9]?)\.){3}
...                       (?:25[0-5]
...                       |2[0-4][0-9]
...                       |[1]?[0-9][0-9]?)$''', re.X)
>>> exp.match('192.168.1.1')
<_sre.SRE_Match object at 0x108f47ac0>
>>> exp.match('255.255.255.0')
<_sre.SRE_Match object at 0x108f47b28>
>>> exp.match('172.17.0.5')
<_sre.SRE_Match object at 0x108f47ac0>
>>> exp.match('256.0.0.0') is None
True
```

Match Mac address

```
>>> import random
>>> mac = [random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f),
...        random.randint(0x00, 0x7f)]
>>> mac = ':'.join(map(lambda x: "%02x" % x, mac))
>>> mac
'3c:38:51:05:03:1e'
>>> exp = re.compile(r'''[0-9a-f]{2}([:])
...                       [0-9a-f]{2}
...                       (\1[0-9a-f]{2}){4}$''', re.X)
>>> exp.match(mac) is not None
True
```

Lexer

```
>>> import re
>>> from collections import namedtuple
>>> tokens = [r'(?P<NUMBER>\d+)',
...           r'(?P<PLUS>\+)',
...           r'(?P<MINUS>-)',
...           r'(?P<TIMES>\*)',
...           r'(?P<DIVIDE>/)',
...           r'(?P<WS>\s+)']
>>> lex = re.compile('|'.join(tokens))
>>> Token = namedtuple('Token', ['type', 'value'])
>>> def tokenize(text):
...     scan = lex.scanner(text)
...     return (Token(m.lastgroup, m.group())
...         for m in iter(scan.match, None) if m.lastgroup != 'WS')
...
>>> for _t in tokenize('9 + 5 * 2 - 7'):
...     print(_t)
...
Token(type='NUMBER', value='9')
Token(type='PLUS', value='+')
Token(type='NUMBER', value='5')
Token(type='TIMES', value='*')
Token(type='NUMBER', value='2')
Token(type='MINUS', value='-')
Token(type='NUMBER', value='7')
```

Link to web resources:

1. https://bit.ly/3kgeX0R
2. Regular expressions in Python 3.10: https://bit.ly/2ZWN5HO