Regular Expression

- What is Regex?
- Meta characters
- Pattern matching
- Functions in `re` module
- Usage of regex object
- String substitution

What is regular expression?

- String search
  - (e.g.) Find integer/float numbers in the given string given below.

    ‘123ABC D123 4 5.6 789 0.12.3 D.E024’
- Regular Expression (i.e., Regex)
  - Define search, extraction, substitution of string patterns
  - Python supports own regex module - re

Meta characters

- Meta characters?
  - Control characters that describe string patterns, rather than strings themselves
- Repeating mea characters

character	meaning	e.g.
*	0 or more	ca*t ➔ ct, cat, caat, caaaaaat...
+	1 or more	ca+t ➔ cat, caat, caaaat ...
?	0 or 1	ca?t ➔ ct, cat
{m}	m times	ca{2} ➔ caa
{m, n}	At least m, at most n	ca{2,4} ➔ caat, caaat, caaaat

Matching meta characters

Meta character	meaning
.	Match all characters except the new line character re.DOTALL can match all characters.
^	1. Matches the first character of the class 2. Matches beginning of each line in re.MULTILINE 3. Used for complementing the set if used in []
$	1. Matches the last character of the class 2. Used as $ character itself if used in []
[]	Indicates a selective character set
|	A|B means A or B.
()	Used for grouping regex

Match

- match(pattern, string [, flags])
  - Return: if matched, return matched object
    else , return None
- (e.g.)

```
>>> import re
>>> re.match('[0-9]', '1234')
<SRE_Match object at 00A03330>
>>> re.match('[0-9]', 'abc')
>>>
```

Match (more examples)

```
>>> m = re.match('[0-9]', '1234')
>>> m.group()
'1'
>>> m = re.match('[0-9]+', '1234')
>>> m.group()
'1234'
>>> m = re.match('[0-9]+', '1234  ')
>>> m.group()
'1234'
>>> re.match('[0-9]+', '    1234    ')
>>> re.match(r'[  \t]*[0-9]+', '   123')
<_sre.SRE_Match object at ....>
```

(pre-defined) special symbols

symbol	meaning	equivalence
\\	Backslash (i.e., '\') itself	
\d	Matches any decimal digit	[0-9]
\D	Matches any non-digit characters	[^0-9]
\s	Matches any whitespace character	[\t\n\r\f\v]
\S	Matches any non-whitespace character	[^ t\n\r\f\v]
\w	Matches any alphanumeric character	[a-zA-Z0-9_]
\W	Matches any non-alphanumeric character	[^a-zA-Z0-9_]
\b	Indicates boundary of a string	
\B	Indicates no boundary of a string	

(e.g) special symbols

```
>>> m = re.match('\s*\d+, '   123')
>>> m.group()
'   123'
>>> m = re.match('\s*(\d+)', '   1234   ')
>>> m.group(1)
'1234'
```

Search

- Match vs. Search
  - Match – from the beginning of a string
  - Search – partial string match in any position
- Search(pattern, string [, flags])
- (e.g)

```
>>> re.search('\d+', '   1034   ').group()
'1034'
>>> re.search('\d+', '  abc123  ' ).group()
'123'
```

Regex in raw mode

```
>>> '\d'
'\\d'
>>> '\s'
'\\s'
>>> '\b'
'\x08'
>>> '\\b'
'\\b'
>>> r'\b'
'\\b'
```

```
>>> '\\'
'\\'
>>> r'\\'
'\\\\'

>>> re.search('\\\\', '  \  ').group()
'\\'
>>> re.search(r'\\', '  \  ').group()
'\\'
```

Greedy/non-greedy matching

- Non-greedy matching

Symbol	Meaning
*?	* with non-greedy matching
+?	+ with non-greedy matching
??	? With non-greedy matching
{m, n}?	{m, n} with non-greedy matching

Greedy/non-greedy (e.g.)

```
>>> s = ‘<a href=“index.html”>HERE</a><font
    size=“10”>’

>>> re.search(r’href=“(.*)”’, s).group(1)
‘index.html”>HERE</a><font size=“10’

>>> re.search(r’href-”(.*?)”’, s).group(1)
‘index.html’
```

Extracting matched string

- Mathods in match() object

Method	Return
`group()`	Matched strings
`start()`	Starting position of matched string
`end()`	Ending position of matched string
`span()`	Tuple of (starting, ending) of matched string

Extracting matched string (cont.)

- Group()-related methods

Method	return
`group()`	whole strings matched
`group(n)`	N-th matched string. group(0) is equivalent to group()
`groups()`	All strings matched in a tuple format

E.g., Group()

```
>>> m = re.match(r'(\d+)(\w+)(\d+)', '123abc456')
>>> m.group()
'123abc456'
>>> m.group(0)
'123abc456'
>>> m.group(1)
'123'
>>> m.group(2)
'abc45'
>>> m.group(3)
'6'
```

```
>>> m.group()
('123', 'abc45', '6')
>>> m.start()
0
>>> m.end()
9
>>>m.span()
(0, 9)
```

E.g. group() (cont.)

```
>>> m = re.match('(a(b(c)) (d))', 'abcd')
>>> m.group(0)
'abcd'
>>> m.group(1)
'abcd'
>>> m.group(2)
'bc'
>>> m.group(3)
'c'
>>> m.group(4)
'd'
```

Group name

- Pattern: (?P<name>regex)
- Name: group name
- Regex: regular expression
- (e.g.)

```
>>> m = re.match(r'(?P<var>[_a-zA-Z]\w*)\s*
=\s*(?P<num>\d+)', 'a = 123')
>>> m.group('var')
'a'
>>> m.group('num')
'123'
>>> m.group()
'a = 1 2 3'
```

Methods in re module

Method	function
compile(pattern [, flags])	compile
search(pattern, string [, flags])	search
match(patterns, sting [, flags])	match from the begining
split(pattern, string [, maxsplit=0])	split
findall(pattern, string)	extracting all strings matched
sub(pattern, repl, string [, count=0])	substitute pattern with repl
subn(pattern,repl, string [, count=0])	substitute with count times
escape(string)	add backslash to non-alphanumeric strings

Re.split(pattern, string [, maxsplit=0])

```
>>> re.split('\W+', 'apple, orange and spam.')
['apple', 'orange', 'and', 'spam', '']
>>> re.split('(\W+)', 'apple, orange and spam.')
['apple', ', ',  'orange', ' ',  'and', ' ', 'spam', '.', '']
>>> re.split('\W}', 'apple, orange and spam.', 1)
['apple', 'orange and spam.']
>>> re.split('\W}', 'apple, orange and spam.', 2)
['apple', 'orange',  'and spam.']
>>> re.split('\W}', 'apple, orange and spam.', 3)
['apple', 'orange',  'and',  'spam.']
```

findall(pattern, string)

```
>>> re.findall(r'[_a-zA-Z]\w*', '123 abc 123
    def')
['abc', 'def']
>>> s = '''
<a href="link1.html">link1</a>
<a href="link2.html">link2</a>
<a href="link3.html">link3</a>'''
>>> re.findall(r'herf="(.*?)"', s)
['link1.html', 'link2.html', 'link3.html']
```

compile(patern [, flags])

- Returns a regex object that is compiled
- Useful for repeated many times
- (e.g)

```
>>> p = re.compile(r'([_a-zA-Z]\w*\s*=\s*(\d+)')
>>> m = p.match('a = 123')
>>> m.groups()
('a', '123')
>>> m.group(1)
'a'
>>> m.group(2)
'123'
```

Flags for regex object

flag	meaning
I, IGNORECASE	Ignore case of characters
L, LOCALE	\w, \W, \b, \B are affected by current locale
M, MULTILINE	^ and $ are applied to the beginning and end of each string
S, DOTALL	. Matches new line character (i.e., \n) as well
U, UNICODE	\w, \W, \b, \B are processed according to unicode
X, VERBOSE	Blanks inside regex are ignored. Backslash(\) needs to be used to consider blanks. # can be used for comments

E.g.

```
>>> import re
>>> p = re.compile('the', re.I)
>>> p.findall('The cat was hungry. They were scared because of the cat')
['The', 'The', 'the']
```

Methods of Regex object

Method	meaning
search(string [, pos [, endpos]])	search
match(string [, pos [, endpos]])	match from the begining
split(string [, maxsplit = 0])	re.split()
findall(string)	re.findall()
sub(repl, string [, count =0])	re.sub()
subn(repl, string [, count = 0])	re.subn()

Search(string [, pos [, endpos]])

```
>>> t = 'My ham, my egg, my spam'
>>> p = re.compiile('(my)', re.I)
>>> pos = 0
>>> while 1:
            m = p.search(t, pos)
            if m:
                print m.group(1)
                pos = m.end()
            else:   break

My
my
my
```

Sub(repl, string [, count = 0])

```
>>> re.sub(r'[.,:;]', '', 'a:b;c, d.')
'abc d'
>>> re.sub(r'\W', '', 'a:b:c, d.')
'abcd'
```

{\n}, {g\<n>}: substitute with string of group n

```
>>> p = re.compile('section{  (  [^}]*  )  }', re.VERBOSE)
>>> p.sub(r'subsection{\1}', 'section{First} section{second}')
'subsection{First} subsection{second}'
```

{\g<name>}: substitute with group name

```
>>> p = re.compile('section{   (  ?P<name>
    [^}]*    )   }', re.VERBOSE)
>>> p.sub(r'subsection{\g<name>}',
    'section{First}')
'subsection{First}'
```

Substitute with function

```
>>> def hexrepl( match):
        value = int (match.group())
        return hex(value)

>>> p = re.compile(r'\d+')
>>> p.sub(hexrepl, 'Call 65490 for printing,
    49152 for user code.')
'Call 0xffd2 for printing, 0xc000 for user code.'
```