Regular expressions

Meher Krishna Patel

Created on : Octorber, 2017

Last updated : May, 2020

Table of contents

Contents:

Chapter 1

Regular Expression

Regular expression are used for searching, validating and modifying the text.

1.1 Raw strings (r')

Since, regular expression are used for searching the words, therefore we encounter with all kinds of special characters and we need to take care of these characters. For example in below code, '' is used, which has the special meaning in python, therefore when 'print' statement is executed, then some of the letters are missing in the outputs. The best way, to remove this problem is to use "r'" with the expression, which consider the all the string as text i.e. no special meaning is considered for '' and we get the desired output.

```
>>> path = 'usr\etc\bin'
>>> print(path)
usr\etin
>>> path = r'usr\etc\bin'
>>> print(path)
usr\etc\bin
>>>
```

Note that, raw string is not a regular expression, but it is used frequently with regular expression as regular expression uses lots of ''.

1.2 Searching data

Suppose we want to check whether a 'string' exist in a variable, then we can perform following operation,

```
>>> msg = "Hello World"
>>> 'hello' in msg
False
>>> 'Hello' in msg
True
>>>
```

- Same can be achieved using regular expression as well. The 're' library contains various methods related to regular expressions. The 'search' method is used to find any string or character, as shown below,

```
>>> import re
>>> msg = "Hello World"
>>> # search for 'hello' in msg
>>> print(re.search(r'hello', msg))
None
```

(continues on next page)

(continued from previous page)

```
>>> # search for 'Hello' in msg
>>> print(re.search(r'Hello', msg))
<_sre.SRE_Match object; span=(0, 5), match='Hello'>
>>>

>>> # no result is displayed for no match
>>> re.search(r'hello', msg)  # no match
>>> re.search(r'Hello', msg)  # matched
<_sre.SRE_Match object; span=(0, 5), match='Hello'>
>>>
```

Note that re.search command does not show any result in python-shell when there is no match.

1.3 Character class [ ]

- Character class (i.e. [ ]) is used to find any character for a string. In below code, (a, b, c or d) are searched in msg. Since 'd' is present in msg, therefore matched result is returned. Also, note that only first match is return as shown in below code,

```
>>> re.search(r'[abcd]', msg)
<_sre.SRE_Match object; span=(10, 11), match='d'>

>>> # only first matched is returned
>>> re.search(r'[abcde]', msg)
<_sre.SRE_Match object; span=(1, 2), match='e'>
>>> re.search(r'[abced]', msg)
<_sre.SRE_Match object; span=(1, 2), match='e'>
>>>
```

1.3.1 Match a range

- We can define a range using '-' as shown below,

```
>>> text = 'please text on 8888'

>>> # [0-9] = 0, 1, 2, ..., 9
>>> re.search(r'[0-9]', text)
<_sre.SRE_Match object; span=(15, 16), match='8'>

>>> # [a-d] = a, b, c, d
>>> re.search(r'[a-d]', text)
<_sre.SRE_Match object; span=(3, 4), match='a'>
>>>
```

1.3.2 Match various range and non-range

- We can mix the ranges and non-ranges together as below,

```
>>> # match for a-z A-Z 0-0 _ $ and *
>>> re.search(r'[a-zA-Z0-9_$*]', text)
<_sre.SRE_Match object; span=(0, 1), match='p'>
>>>
```

1.3.3 Match except given string

- ^[] is used to specify the characters which we are not looking for, as shown below,

```
>>> # match anything but 0-9
>>> re.search(r'[^0-9]', text)
<_sre.SRE_Match object; span=(0, 1), match='p'>

>>> # match anything but 0-9 and a-z
>>> re.search(r'[^0-9a-z]', text) # space matched ' '
<_sre.SRE_Match object; span=(6, 7), match=' '>

>>> # match anything 0-9 a-z and space
>>> re.search(r'[^0-9 a-z]', text) # nothing matched
```

1.3.4 Starts with

Note:

- [^ap] : match anything but a and p
- ^ap : match for string which starts with 'ap'

```
>>> # matched only if starts with 'pl'
>>> re.search(r'^pl', text)
<_sre.SRE_Match object; span=(0, 2), match='pl'>
>>>
```

1.3.5 Ends with

$ sign is used at the end of the string for 'end with' search,

```
>>> re.search(r'88$', text)
<_sre.SRE_Match object; span=(17, 19), match='88'>
>>>
```

1.3.6 Exact match

Use ^ and $ together for exact match,

```
>>> re.search(r'88$', text)
<_sre.SRE_Match object; span=(17, 19), match='88'>
>>>
```

1.3.7 Match meta characters

Characters which have special meaning in regular expressions are known as meta characters e.g. ^, $ and [] etc. The backslash '' is used for matching these characters.

```
>>> # search for [Hello
>>> re.search(r'\[Hello', "[Hello World]")
<_sre.SRE_Match object; span=(0, 6), match='[Hello'>
>>>
```

1.3.8 Match any character

The dot (.) is used for matching any character.

```
>>> # match any character : h is matched
>>> re.search(r'.', 'hello')
<_sre.SRE_Match object; span=(0, 1), match='h'>

>>> # match any character : w is matched
>>> re.search(r'.', 'world')
<_sre.SRE_Match object; span=(0, 1), match='w'>

>>> # starts with wo and then any character
>>> re.search(r'^wo.', 'world')
<_sre.SRE_Match object; span=(0, 3), match='wor'>
>>>
```

1.3.9 Match starting and ending

- The (*) is used for defining any or no number of sequence. The dot (.) and (*) can be used together for searching something based on start and end. **Any character or no character can be between a and e in below example,**

```
>>> # starts with 'a', ends with 'e'
>>> re.search(r'^a.*e$', 'apple')
<_sre.SRE_Match object; span=(0, 5), match='apple'>
>>>

>>> # match for only numbers
>>> # ^[0-9]*$ : starts with number, * i.e. numbers with any length,
>>> # *$ ends with number of any length
>>> re.search(r'^[0-9]*$', '100')
<_sre.SRE_Match object; span=(0, 3), match='100'>
>>> re.search(r'^[0-9]*$', '$100')
>>>
```

- The (+) sign is used for at least on occurrence. In the below code, (*) matches the no number as well. This problem is removed by using (+).

```
>>> # * matches for no numbers as well
>>> re.search(r'^[0-9]*$', '')
<_sre.SRE_Match object; span=(0, 0), match=''>

>>> # not matched for no number
>>> re.search(r'^[0-9]+$', '')

>>> # matched for only numbers
>>> re.search(r'^[0-9]+$', '100')
<_sre.SRE_Match object; span=(0, 3), match='100'>
>>>
```

- The (?) is used for zero or one match only. Suppose, we want to match 'colour' and 'color' i.e. 'u' is optional, then we can use (?) as below,

```
>>> # match for color and colour
>>> re.match(r'colou?r', 'colours are ')
<_sre.SRE_Match object; span=(0, 6), match='colour'>
>>> re.match(r'colou?r', 'colors are ')
<_sre.SRE_Match object; span=(0, 5), match='color'>
>>> re.match(r'colou?r', 'These are ')
>>>
```

1.4 Special sequences

1.4.1 Match digits

Suppose, we want to match any digit i.e. from 0 to 9. In this case, [0-9] can be replaced by '\d' as shown below,

```
>>> # match for digits
>>> re.search(r'\d', 'apple')
>>> re.search(r'\d', 'apple 7')
<_sre.SRE_Match object; span=(6, 7), match='7'>
```

```
>>> # match for digits only
>>> re.search(r'^\d+$', '347a')
>>> re.search(r'^\d+$', '34')
<_sre.SRE_Match object; span=(0, 2), match='34'>
>>> re.search(r'^\d+$', '34 ')
>>> re.search(r'^\d+$', '')
>>>
```

1.4.2 Do not match digit

(\D) is used for not matching the digit,

```
>>> re.search(r'\D', 'apple')
<_sre.SRE_Match object; span=(0, 1), match='a'>
>>> re.search(r'\D', 'apple 7')
<_sre.SRE_Match object; span=(0, 1), match='a'>
>>>
```

1.4.3 Other special characters

- \w : match for 'characters', 'number' and 'underscore (_)' i.e. [a-zA-Z0-9_]
- \W : opposite of \w
- \s : match for spaces, tabs and end line(enter).
- \S : opposite of \s
- \b : looks for the non-character (i.e. \w) before and after the search string.

```
>>> # look for hello with no a-zA-Z0-9_ before h
>>> re.search(r'\bhello', 'hello there')
<_sre.SRE_Match object; span=(0, 5), match='hello'>

>>> # look for hello with no \w before and after hello
>>> re.search(r'\bhello\b', 'hello_there') # _ is character
>>> re.search(r'\bhello\b', 'hello-there') # - is not character
<_sre.SRE_Match object; span=(0, 5), match='hello'>
>>>
```

- {} are used to check for exact, more or less number of repetition, e.g. exactly 4 number of Pincode can be checked as follows,

```
>>> # match exactly 6 digits
>>> re.search(r'^\d{6}$', '12345')
>>> re.search(r'^\d{6}$', '123456')
<_sre.SRE_Match object; span=(0, 6), match='123456'>

>>> # match 3-6 digits
>>> re.search(r'^\d{3,6}$', '12345')
```

(continues on next page)

(continued from previous page)

```
<_sre.SRE_Match object; span=(0, 5), match='12345'>
>>>
>>> # match 3-6 characters i.e. \w
>>> re.search(r'^\w{3,6}$', 'Names3')
<_sre.SRE_Match object; span=(0, 6), match='Names3'>
>>> re.search(r'^\w{3,6}$', 'Na')
>>>
>>> # match for 4 or more characters
>>> re.search(r'^\w{4,}$', 'Tigers')
<_sre.SRE_Match object; span=(0, 6), match='Tigers'>
>>> re.search(r'^\w{4,}$', 'Cat')
>>>
>>> # match for 4 or less characters
>>> re.search(r'^\w{,4}$', 'Cat')
<_sre.SRE_Match object; span=(0, 3), match='Cat'>
>>> re.search(r'^\w{,4}$', 'Tigers')
>>>
```

1.5 Flags

Flags are used to modify the behaviour of the search,

1.5.1 Case insensitive match

```
>>> re.search(r'hello', 'Hello world')

>>> # method 1 : check for Hello and hello only (not HeLLo)
>>> re.search(r'[hH]ello', 'Hello world')
<_sre.SRE_Match object; span=(0, 5), match='Hello'>
>>> re.search(r'[hH]ello', 'hello world')
<_sre.SRE_Match object; span=(0, 5), match='hello'>

>>> # method 2 : flag, match for hello, Hello, HeLLo etc.
>>> re.search(r'hello', 'Hello world', re.IGNORECASE)
<_sre.SRE_Match object; span=(0, 5), match='Hello'>
>>>
```

1.5.2 Writing expression in different line

It is convinient to write long expression in multiple lines, as it becomes more readable. This can be done using re.VERBOSE flag,

```
>>> re.search(r'''
...     ^[a-z]+  # starts with a-z
...     -       # then contain -
...     [0-9]+$  # ends with number
... ''', 'cat-34', re.VERBOSE)
<_sre.SRE_Match object; span=(0, 6), match='cat-34'>
>>>

>>> # use two flags using |
>>> re.search(r'''
...     ^[a-z]+  # starts with a-z
...     -       # then contain -
...     [0-9]+$  # ends with number
```

(continues on next page)

(continued from previous page)

```
... ''', 'TIGer-34', re.IGNORECASE | re.VERBOSE)
<_sre.SRE_Match object; span=(0, 8), match='TIGer-34'>
>>>
```

Note: Verbose flag skips the spaces, therefore if we want match the spaces then \ must be used as shown below,

```
>>> # '' is matched instead of ' '
>>> re.search(r' ', 'Hello World', re.VERBOSE)
<_sre.SRE_Match object; span=(0, 0), match=''>

>>> use r'\ ' or r'[ ] for matching spaces in Verbose flag
>>> re.search(r'\ ', 'Hello World', re.VERBOSE)
<_sre.SRE_Match object; span=(5, 6), match=' '>
>>> re.search(r'[ ] ', 'Hello World', re.VERBOSE)
<_sre.SRE_Match object; span=(5, 6), match=' '>
>>>
```

1.6 Make group

Group can be made using () as shown below,

```
>>> # check for 4444-333 where -333 is optional
>>> # make group for -333 and then make it optional using ?

>>> # partial match is ok
>>> re.search(r'^\d{4}(-\d{3})?$', '1232')
<_sre.SRE_Match object; span=(0, 4), match='1232'>

>>> # complete match is ok
>>> re.search(r'^\d{4}(-\d{3})?$', '1232-323')
<_sre.SRE_Match object; span=(0, 8), match='1232-323'>

>>> # not matched
>>> re.search(r'^\d{4}(-\d{3})?$', '1232-3233')
>>>
```

Note: Group can be used for storing the part of the search as well for further processing as shown below,

```
>>> re.search(r'(^\w{4})(-\d{3})?$', 'Cats-323').group(0)
'Cats-323'
>>> re.search(r'(^\w{4})(-\d{3})?$', 'Cats-323').group(1)
'Cats'
>>> re.search(r'(^\w{4})(-\d{3})?$', 'Cats-323').group(2)
'-323'
```

1.7 Matching mutliple items

Suppose we want to select the name of all the countries from the sentence, "JAPAN is closer to INDIA than USA". This can be done as below,

```
>>> text = "JAPAN is closer to INDIA than USA"

>>> # we can use findall option
>>> ca = re.findall(r'\b[A-Z]+\b', text)
>>> ca
['JAPAN', 'INDIA', 'USA']
>>>

>>> # or we can use finditer, which return a iterator
>>> countries = re.finditer(r'\b[A-Z]+\b', text)
>>> countries
<callable_iterator object at 0xb700fa0c>

>>> for country in countries:
...     print(country.group())
...
JAPAN
INDIA
USA
>>>

>>> # same can be achieved with list-comprehension as below
>>> c = [m.group() for m in re.finditer(r'\b[A-Z]+\b', text)]
>>> c
['JAPAN', 'INDIA', 'USA']
>>>
```