Regular Expression in Python with Examples | Set 1

Difficulty Level : Medium • Last Updated : 19 Oct, 2020

Module Regular Expressions(RE) specifies a set of strings(pattern) that matches it.
To understand the RE analogy, MetaCharacters are useful, important and will be used in functions of module re.
There are a total of 14 metacharacters and will be discussed as they follow into functions:

```
\   Used to drop the special meaning of character
    following it (discussed below)
[]  Represent a character class
^   Matches the beginning
$   Matches the end
.   Matches any character except newline
?   Matches zero or one occurrence.
|   Means OR (Matches with any of the characters
    separated by it.
*   Any number of occurrences (including 0 occurrences)
+   One or more occurrences
{}  Indicate number of occurrences of a preceding RE
    to match.
()  Enclose a group of REs
```

- **Function compile()**
  Regular expressions are compiled into pattern objects, which have methods for various operations such as searching for pattern matches or performing string substitutions.

Python

```python
# Module Regular Expression is imported using __import__().
import re

# compile() creates regular expression character class [a-e],
# which is equivalent to [abcde].
# class [abcde] will match with string with 'a', 'b', 'c', 'd', 'e'.
p = re.compile('[a-e]')

# findall() searches for the Regular Expression and return a list upon finding
print(p.findall("Aye, said Mr. Gibenson Stark"))
```

Output:

```
['e', 'a', 'd', 'b', 'e', 'a']
```

Understanding the Output:
First occurrence is 'e' in "Aye" and not 'A', as it being Case Sensitive.
Next Occurrence is 'a' in "said", then 'd' in "said", followed by 'b' and 'e' in "Gibenson", the Last 'a' matches with "Stark".
Metacharacter backslash '\' has a very important role as it signals various sequences. If the backslash is to be used without its special meaning as metacharacter, use'\\'

```
\d   Matches any decimal digit, this is equivalent
     to the set class [0-9].
\D   Matches any non-digit character.
\s   Matches any whitespace character.
\S   Matches any non-whitespace character
\w   Matches any alphanumeric character, this is
     equivalent to the class [a-zA-Z0-9_].
\W   Matches any non-alphanumeric character.
```

Set class [\s,.] will match any whitespace character, ',', or,'.' .

Python

```
# \d is equivalent to [0-9].
p = re.compile('\d')
print(p.findall("I went to him at 11 A.M. on 4th July 1886"))

# \d+ will match a group on [0-9], group of one or greater size
p = re.compile('\d+')
print(p.findall("I went to him at 11 A.M. on 4th July 1886"))
```

Output:

```
['1', '1', '4', '1', '8', '8', '6']
['11', '4', '1886']
```

Python

```
import re

# \w is equivalent to [a-zA-Z0-9_].
p = re.compile('\w')
print(p.findall("He said * in some_lang."))

# \w+ matches to group of alphanumeric character.
p = re.compile('\w+')
print(p.findall("I went to him at 11 A.M., he said *** in some_language."))

# \W matches to non alphanumeric characters.
p = re.compile('\W')
print(p.findall("he said *** in some_language."))
```

Output:

```
['H', 'e', 's', 'a', 'i', 'd', 'i', 'n', 's', 'o', 'm', 'e', '_', 'l', 'a', 'n', 'g']
['I', 'went', 'to', 'him', 'at', '11', 'A', 'M', 'he', 'said', 'in', 'some_language']
[' ', ' ', '*', '*', '*', ' ', ' ', '.']
```

Python

```
import re

# '*' replaces the no. of occurrence of a character.
p = re.compile('ab*')
print(p.findall("ababbaabbb"))
```

Output:

```
['ab', 'abb', 'a', 'abbb']
```

Understanding the Output:
Our RE is ab*, which 'a' accompanied by any no. of 'b's, starting from 0.
Output 'ab', is valid because of single 'a' accompanied by single 'b'.
Output 'abb', is valid because of single 'a' accompanied by 2 'b'.
Output 'a', is valid because of single 'a' accompanied by 0 'b'.
Output 'abbb', is valid because of single 'a' accompanied by 3 'b'.

- **Function split()**
  Split string by the occurrences of a character or a pattern, upon finding that pattern, the remaining characters from the string are returned as part of the resulting list.
  Syntax :

```
re.split(pattern, string, maxsplit=0, flags=0)
```

The First parameter, pattern denotes the regular expression, string is the given string in which pattern will be searched for and in which splitting occurs, maxsplit if not provided is considered to be zero '0', and if any nonzero value is provided, then at most that many splits occurs. If maxsplit = 1, then the string will split once only, resulting in a list of length 2. The flags are very useful and can help to shorten code, they are not necessary parameters, eg: flags = re.IGNORECASE, In this split, case will be ignored.

Python

```python
from re import split

# '\W+' denotes Non-Alphanumeric Characters or group of characters
# Upon finding ',' or whitespace ' ', the split(), splits the string from that point
print(split('\W+', 'Words, words , Words'))
print(split('\W+', "Word's words Words"))

# Here ':', ' ' ,',' are not AlphaNumeric thus, the point where splitting occurs
print(split('\W+', 'On 12th Jan 2016, at 11:02 AM'))

# '\d+' denotes Numeric Characters or group of characters
# Splitting occurs at '12', '2016', '11', '02' only
print(split('\d+', 'On 12th Jan 2016, at 11:02 AM'))
```

Output:

```
['Words', 'words', 'Words']
['Word', 's', 'words', 'Words']
['On', '12th', 'Jan', '2016', 'at', '11', '02', 'AM']
['On ', 'th Jan ', ', at ', ':', ' AM']
```

Python

```python
import re

# Splitting will occurs only once, at '12', returned list will have length 2
print(re.split('\d+', 'On 12th Jan 2016, at 11:02 AM', 1))

# 'Boy' and 'boy' will be treated same when flags = re.IGNORECASE
print(re.split('[a-f]+', 'Aey, Boy oh boy, come here', flags = re.IGNORECASE))
print(re.split('[a-f]+', 'Aey, Boy oh boy, come here'))
```

Output:

```
['On ', 'th Jan 2016, at 11:02 AM']
['', 'y, ', 'oy oh ', 'oy, ', 'om', ' h', 'r', '']
['A', 'y, Boy oh ', 'oy, ', 'om', ' h', 'r', '']
```

- **Function sub()**
  Syntax:

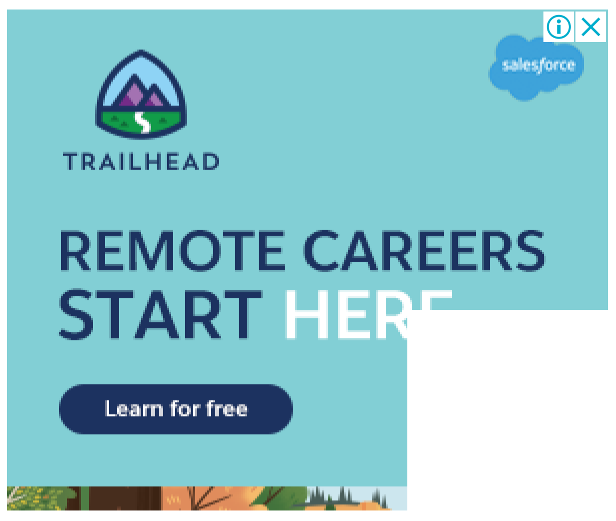
```
re.sub(pattern, repl, string, count=0, flags=0)
```

The 'sub' in the function stands for SubString, a certain regular expression pattern is searched in the given string(3rd parameter), and upon finding the substring pattern is replaced by repl(2nd parameter), count checks and maintains the number of times this occurs.

Python

```python
import re

# Regular Expression pattern 'ub' matches the string at "Subject" and "Uber".
# As the CASE has been ignored, using Flag, 'ub' should match twice with the string
# Upon matching, 'ub' is replaced by '~*' in "Subject", and in "Uber", 'Ub' is replaced.
print(re.sub('ub', '~*' , 'Subject has Uber booked already', flags = re.IGNORECASE))

# Consider the Case Sensitivity, 'Ub' in "Uber", will not be reaplced.
print(re.sub('ub', '~*' , 'Subject has Uber booked already'))

# As count has been given value 1, the maximum times replacement occurs is 1
print(re.sub('ub', '~*' , 'Subject has Uber booked already', count=1, flags = re.IGNORECASE))

# 'r' before the patter denotes RE, \s is for start and end of a String.
print(re.sub(r'\sAND\s', ' & ', 'Baked Beans And Spam', flags=re.IGNORECASE))
```

Output



```
S~*ject has ~*er booked already
S~*ject has Uber booked already
S~*ject has Uber booked already
Baked Beans & Spam
```

- **Function subn()**
  Syntax:

```
 re.subn(pattern, repl, string, count=0, flags=0)
```

subn() is similar to sub() in all ways, except in its way to providing output. It returns a tuple with count of total of replacement and the new string rather than just the string.

Python

```python
import re
print(re.subn('ub', '~*' , 'Subject has Uber booked already'))
t = re.subn('ub', '~*' , 'Subject has Uber booked already', flags = re.IGNORECASE)
print(t)
print(len(t))

# This will give same output as sub() would have
print(t[0])
```

Output

```
('S~*ject has Uber booked already', 1)
('S~*ject has ~*er booked already', 2)
Length of Tuple is:  2
S~*ject has ~*er booked already
```

- **Function escape()**
  Syntax:

```
re.escape(string)
```

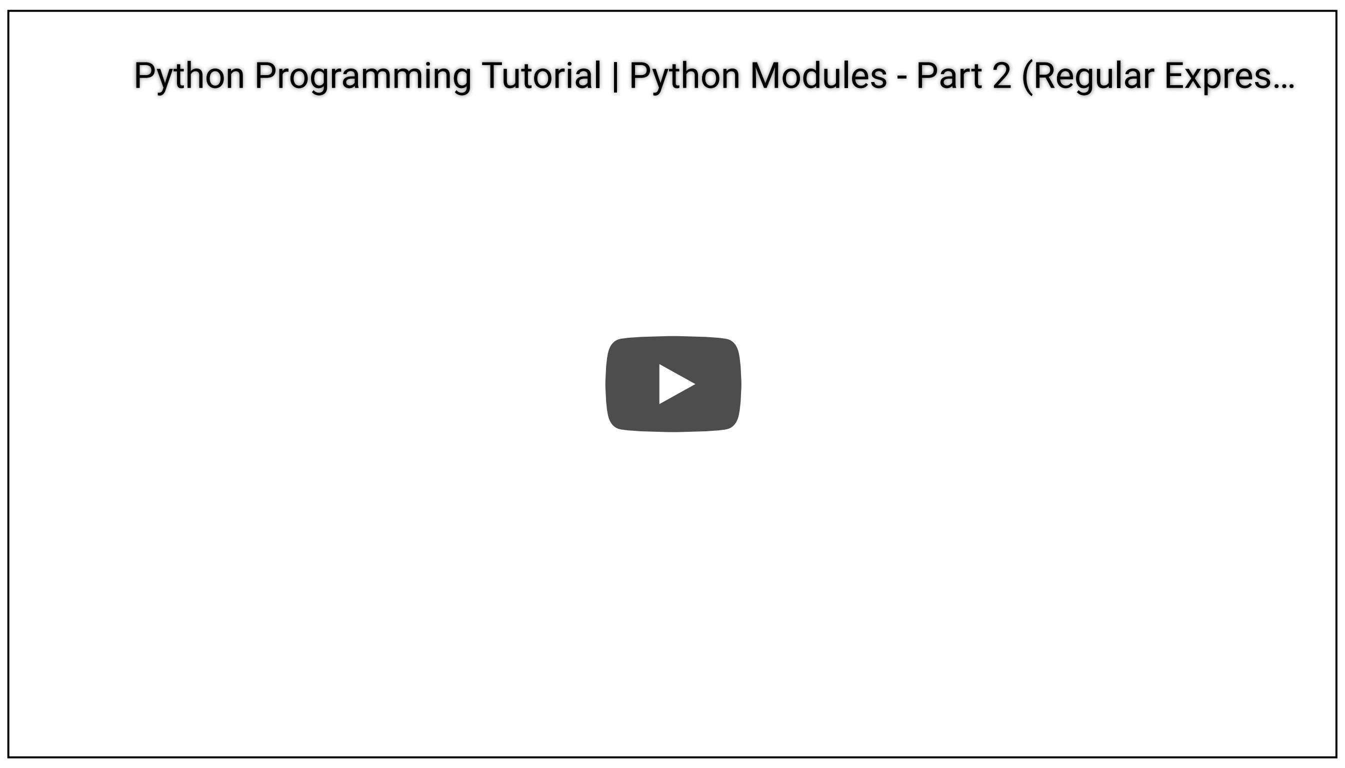
Return string with all non-alphanumerics backslashed, this is useful if you want to match an arbitrary literal string that may have regular expression metacharacters in it.

Python

```python
import re

# escape() returns a string with BackSlash '\', before every Non-Alphanumeric Character
# In 1st case only ' ', is not alphanumeric
# In 2nd case, ' ', caret '^', '-', '[]', '\' are not alphanumeric
print(re.escape("This is Awseome even 1 AM"))
print(re.escape("I Asked what is this [a-9], he said \t ^WoW"))
```

Output

```
This\ is\ Awseome\ even\ 1\ AM
I\ Asked\ what\ is\ this\ \[a\-9\]\,\ he\ said\ \    \ \^WoW
```


Python Programming Tutorial | Python Modules - Part 2 (Regular Expres...

Related Article :
https://www.geeksforgeeks.org/regular-expressions-python-set-1-search-match-find/
Reference:
https://docs.python.org/2/library/re.html

This article is contributed by **Piyush Doorwar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Attention geek! Strengthen your foundations with the **Python Programming Foundation** Course and learn the basics.

To begin with, your interview preparations Enhance your Data Structures concepts with the **Python DS** Course. And to begin with your Machine Learning Journey, join the Machine Learning – Basic Level Course

Like 0

Previous
Python append to a file

Next
Regular Expressions in Python – Set 2 (Search, Match and Find All)

RECOMMENDED ARTICLES

Page : 1 2 3

01 Find all the numbers in a string using regular expression in Python
12, Dec 18

02 Python - Check whether a string starts and ends with the same character or not (using Regular Expression)
20, Apr 20

03 Regular Expressions in Python – Set 2 (Search, Match and Find All)
20, Jun 16

04 Regular Dictionary vs Ordered Dictionary in Python
06, Jan 20

05 Check the equality of integer division and math.floor() of Regular division in Python
20, Apr 20

06 Python Flags to Tune the Behavior of Regular Expressions
25, Sep 20

07 Extracting email addresses using regular expressions in Python
28, Jan 18

08 Map function and Lambda expression in Python to replace characters
13, Dec 17

Article Contributed By :
GeeksforGeeks

Vote for difficulty
Current difficulty : Medium
Easy Normal Medium Hard Expert

Improved By : nidhi_biet, ManasChhabra2, shubham_singh, chaudhary_19, manasdutta

Article Tags : Python

Improve Article Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments