Regular Expressions

Regular Expressions in Python

The term Regular Expression is popularly shortened as regex. A regex is a sequence of characters that defines a search pattern, used mainly for performing find and replace operations in search engines and text processors.

Python offers regex capabilities through the re module bundled as a part of the standard library.

Raw strings

Different functions in Python's re module use raw string as an argument. A normal <u>string</u>, when prefixed with 'r' or 'R' becomes a raw string.

```
Example: Raw String

>>> rawstr = r'Hello! How are you?'

>>> print(rawstr)

Hello! How are you?
```

The difference between a normal string and a raw string is that the normal string in print() function translates escape characters (such as \n, \t etc.) if any, while those in a raw string are not.

meta characters

Some characters carry a special meaning when they appear as a part pattern matching string. In Windows or Linux DOS commands, we use * and ? - they are similar to meta characters. Python's re module uses the following characters as meta characters:

When a set of alpha-numeric characters are placed inside square brackets [], the target string is matched with these characters. A range of characters or individual characters can be listed in the square bracket. For example:

Pattern	Description
[abc]	match any of the characters a, b, or c
[a-c]	which uses a range to express the same set of characters.
[a-z]	match only lowercase letters.
[0-9]	match only digits.

Pattern	Description
\d	Matches any decimal digit; this is equivalent to the 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
\W	Matches any non-alphanumeric character.
	Matches with any single character except newline '\n'.
?	match 0 or 1 occurrence of the pattern to its left
+	1 or more occurrences of the pattern to its left
*	0 or more occurrences of the pattern to its left
[]	Matches any single character in a square bracket
\	It is used for special meaning characters like . to match a period or $\boldsymbol{+}$ for plus sign.
{n,m}	Matches at least n and at most m occurrences of preceding
a b	Matches either a or b

[] - Square brackets

Square brackets specifies a set of characters you wish to match.

Expression	String	Matched?	
[abc]	а	1 match	
	ac	2 matches	
	Hey Jude	No match	
	abc de ca	5 matches	

Here, [abc] will match if the string you are trying to match contains any of the a, b or c.

You can also specify a range of characters using - inside square brackets.

- [a-e] is the same as [abcde].
- [1-4] is the same as [1234].
- [0-39] is the same as [01239].

You can complement (invert) the character set by using caret A symbol at the start of a square-bracket.

- [^abc] means any character except a or b or c.
- [^0-9] means any non-digit character.



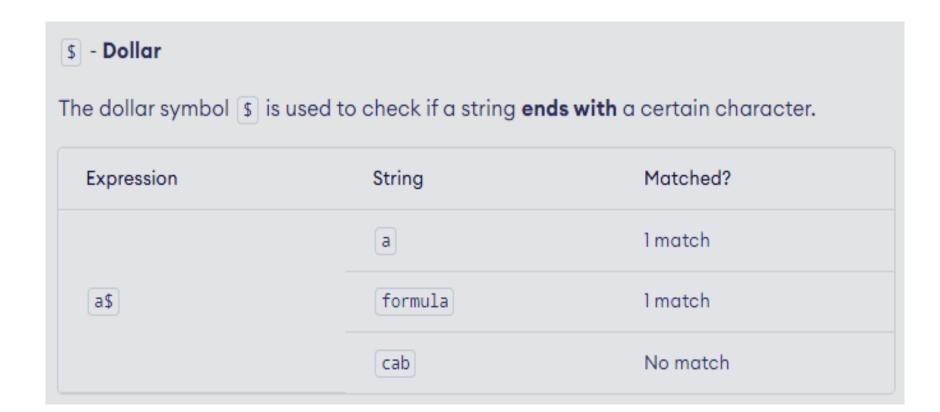
A period matches any single character (except newline '\n').

Expression	String	Matched?
	а	No match
	ac	1 match
	acd	1 match
	acde	2 matches (contains 4 characters)

^ - Caret

The caret symbol \land is used to check if a string **starts with** a certain character.

Expression	String	Matched?
	а	1 match
^a	abc	1 match
	bac	No match
^ab —	abc	1 match
	acb	No match (starts with a but not followed by b)





The star symbol * matches zero or more occurrences of the pattern left to it.

Expression	String	Matched?
	mn	1 match
	man	1 match
ma*n	maaan	1 match
	main	No match (a is not followed by n)
	woman	1 match

+ - Plus

The plus symbol + matches **one or more occurrences** of the pattern left to it.

Expression	String	Matched?
	mn	No match (no a character)
	man	1 match
ma+n	maaan	1 match
	main	No match (a is not followed by n)
	woman	1 match

? - Question Mark

The question mark symbol ? matches **zero or one occurrence** of the pattern left to it.

Expression	String	Matched?
	mn	1 match
	man	1 match
ma?n	maaan	No match (more than one a character)
	main	No match (a is not followed by n)
	woman	1 match

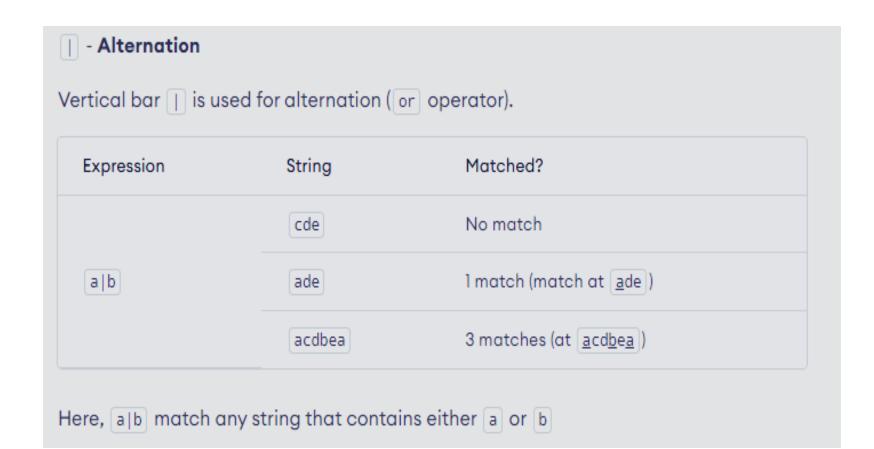
{} - Braces

Consider this code: $\{n,m\}$. This means at least [n], and at most [m] repetitions of the pattern left to it.

Expression	String	Matched?
	abc dat	No match
	abc daat	1 match (at d <u>aa</u> t)
a{2,3}	aabc daaat	2 matches (at <u>aa</u> bc and d <u>aaa</u> t)
	aabc daaaat	2 matches (at <u>aa</u> bc and <u>daaa</u> at)

Let's try one more example. This RegEx [0-9]{2, 4} matches at least 2 digits but not more than 4 digits

Expression	String	Matched?
	ab123csde	1 match (match at ab <u>123</u> csde)
[0-9]{2,4}	12 and 345673	3 matches (<u>12</u> , <u>3456</u> , <u>73</u>)
	1 and 2	No match



() - Group

Parentheses () is used to group sub-patterns. For example, (a|b|c)xz match any string that matches either a or b or c followed by xz

Expression	String	Matched?
	ab xz	No match
(a b c)xz	abxz	1 match (match at a <u>bxz</u>)
	axz cabxz	2 matches (at <u>axz</u> bc ca <u>bxz</u>)

√ - Backslash

\\$a match if a string contains \\$ followed by \[a\]. Here, \\$ is not interpreted by a RegEx engine in a special way.

If you are unsure if a character has special meaning or not, you can put $\sqrt{\ }$ in front of it. This makes sure the character is not treated in a special way.

\A - Matches if the specified characters are at the start of a string.

Expression	String	Matched?
\Athe	the sun	Match
Actie	In the sun	No match

\b - Matches if the specified characters are at the beginning or end of a word.

Expression	String	Matched?
\bfoo	football	Match
	a football	Match
	afootball	No match
foo\b	the foo	Match
	the afoo test	Match
	the afootest	No match

\B - Opposite of \b. Matches if the specified characters are **not** at the beginning or end of a word.

Expression	String	Matched?
\Bfoo	football	No match
	a football	No match
	afootball	Match
foo\B	the foo	No match
	the afoo test	No match
	the afootest	Match

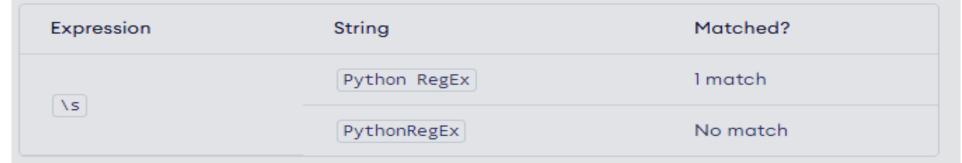
\d - Matches any decimal digit. Equivalent to [0-9]

Expression	String	Matched?
\\d	12abc3	3 matches (at 12abc3)
	Python	No match

\D - Matches any non-decimal digit. Equivalent to [^0-9]

Expression	String	Matched?
\D -	1ab34"50	3 matches (at 1 <u>ab</u> 34 <u>"</u> 50)
	1345	No match

\s - Matches where a string contains any whitespace character. Equivalent to [\t\n\r\f\v].



\S - Matches where a string contains any non-whitespace character. Equivalent to [^ \t\n\r\f\v].

Expression	String	Matched?
١٥	a b	2 matches (at <u>a</u> <u>b</u>)
(15)		No match

w - Matches any alphanumeric character (digits and alphabets). Equivalent to [a-zA-Z0-9_]. By the way, underscore _ is also considered an alphanumeric character.

Expression	String	Matched?
\w -	12&": ;c	3 matches (at <u>12</u> &": ; <u>c</u>)
	%"> !	No match

W - Matches any non-alphanumeric character. Equivalent to [^a-zA-Z0-9_]

Expression	String	Matched?
\W	1a2%c	1 match (at 1 <u>a</u> 2 <u>%</u> c)
	Python	No match

\Z - Matches if the specified characters are at the end of a string.

Expression	String	Matched?
	I like Python	1 match
Python\Z	I like Python Programming	No match
	Python is fun.	No match

Tip: To build and test regular expressions, you can use RegEx tester tools such as regex101. This tool not only helps you in creating regular expressions, but it also helps you learn it.

re.match() function

This function in re module tries to find if the specified pattern is present at the beginning of the given string.

```
re.match(pattern, string)
```

The function returns None, if the given pattern is not in the beginning, and a match objects if found.

```
from re import match

mystr = "Welcome to TutorialsTeacher"
obj1 = match("We", mystr)
print(obj1)
obj2 = match("teacher", mystr)
print(obj2)
```

The following example demonstrates the use of the range of characters to find out if a string starts with 'W' and is followed by an alphabet.

re.search() function

The re.search() function searches for a specified pattern anywhere in the given string and stops the search on the first occurrence.

```
Example: re.search()

from re import search

string = "Try to earn while you learn"

obj = search("earn", string)
print(obj)
print(obj.start(), obj.end(), obj.group())
7 11 earn
```

```
Output

Cre.Match object; span=(7, 11), match='earn'>
```

This function also returns the Match object with start and end attributes. It also gives a group of characters of which the pattern is a part of.

```
import re
string = "Python is fun"
# check if 'Python' is at the beginning
match = re.search('\APython', string)
if match:
  print("pattern found inside the string")
else:
  print("pattern not found")
# Output: pattern found inside the string
```

re.findall() Function

As against the search() function, the findall() continues to search for the pattern till the target string is exhausted. The object returns a list of all occurrences.

```
Example: re.findall()

from re import findall

string = "Try to earn while you learn"

obj = findall("earn", string)
print(obj)
```

```
Output
['earn', 'earn']
```

This function can be used to get the list of words in a sentence. We shall use \W* pattern for the purpose. We also check which of the words do not have any vowels in them.

```
Example: re.findall()

obj = findall(r"\w*", "Fly in the sky.")
print(obj)

for word in obj:
   obj= search(r"[aeiou]",word)
   if word!='' and obj==None:
        print(word)
```

```
Output
['Fly', '', 'in', '', 'the', '', 'sky', '', '']
Fly
sky
```

```
# Program to extract numbers from a string
import re
string = 'hello 12 hi 89. Howdy 34'
pattern = '\d+'
result = re.findall(pattern, string)
print(result)
# Output: ['12', '89', '34']
```

If the pattern is not found, re.findall() returns an empty list.

re.finditer() function

The re.finditer() function returns an iterator object of all matches in the target string. For each matched group, start and end positions can be obtained by span() attribute.

```
Example: re.finditer()

from re import finditer

string = "Try to earn while you learn"
 it = finditer("earn", string)
 for match in it:
    print(match.span())
```

```
Output
(7, 11)
(23, 27)
```

re.split() function

The re.split() function works similar to the split() method of str object in Python. It splits the given string every time a white space is found. In the above example of the findall() to get all words, the list also contains each occurrence of white space as a word. That is eliminated by the split() function in re module.

```
from re import split

string = "Flat is better than nested. Sparse is better than dense."
words = split(r' ', string)
print(words)
```

```
import re
string = 'Twelve:12 Eighty nine:89 Nine:9.'
pattern = '\d+'
# maxsplit = 1
# split only at the first occurrence
result = re.split(pattern, string, 1)
print(result)
# Output: ['Twelve:', ' Eighty nine:89 Nine:9.']
```

re.compile() Function

The re.compile() function returns a pattern object which can be repeatedly used in different regex functions. In the following example, a string 'is' is compiled to get a pattern object and is subjected to the search() method.

```
from re import *

pattern = compile(r'[aeiou]')
string = "Flat is better than nested. Sparse is better than dense."
words = split(r' ', string)
for word in words:
    print(word, pattern.match(word))
```

The same pattern object can be reused in searching for words having vowels, as shown below.

```
Example: search()

for word in words:
    print(word, pattern.search(word))
```

Output

```
Flat <re.Match object; span=(2, 3), match='a'>
is <re.Match object; span=(0, 1), match='i'>
better <re.Match object; span=(1, 2), match='e'>
than <re.Match object; span=(2, 3), match='a'>
nested. <re.Match object; span=(1, 2), match='e'>
Sparse <re.Match object; span=(2, 3), match='a'>
is <re.Match object; span=(0, 1), match='i'>
better <re.Match object; span=(1, 2), match='e'>
than <re.Match object; span=(2, 3), match='a'>
dense. <re.Match object; span=(1, 2), match='e'>
```

re.sub()

The syntax of re.sub() is:

```
re.sub(pattern, replace, string)
```

The method returns a string where matched occurrences are replaced with the content of replace variable.

re.subn()

The re.subn() is similar to re.sub() except it returns a tuple of 2 items containing the new string and the number of substitutions made.

```
import re
# multiline string
string = 'abc 12\
de 23 \n f45 6'
# matches all whitespace characters
pattern = '\s+'
replace = ''
new_string = re.sub(r'\s+', replace, string, 1)
print(new_string)
# Output:
# abc12de 23
# f45 6
```

match.start(), match.end() and match.span()

The start() function returns the index of the start of the matched substring. Similarly, end() returns the end index of the matched substring.

```
>>> match.start()
2
>>> match.end()
8
```

The span() function returns a tuple containing start and end index of the matched part.

```
>>> match.span()
(2, 8)
```

match.re and match.string

The re attribute of a matched object returns a regular expression object. Similarly, string attribute returns the passed string.