Python Regular Expression

Gaurav Kr. suman

MIT08

Python Regular Expression (Regex)

Essentially, a Python regular expression is a sequence of characters, that defines a search pattern. We can then use this pattern in a string-searching algorithm to “find” or “find and replace” on strings. You would’ve seen this feature in Microsoft Word as well.

In this Python Regex tutorial, we will learn the basics of regular expressions in Python. For this, we will use the ‘re’ module. Let’s import it before we begin.

```
>>> import re
```

Python Regex – Metacharacters

Each character in a Python Regex is either a metacharacter or a regular character. A metacharacter has a special meaning, while a regular character matches itself. Python has the following metacharacters:

Metacharacter	Description
^	Matches the start of the string
.	Matches a single character, except a newline But when used inside square brackets, a dot is matched
[ ]	A bracket expression matches a single character from the ones inside it [abc] matches ‘a’, ‘b’, and ‘c’ [a-z] matches characters from ‘a’ to ‘z’ [a-cx-z] matches ‘a’, ’b’, ’c’, ’x’, ’y’, and ‘z’
[^]	Matches a single character from those except the ones mentioned in the brackets[^abc] matches all characters except ‘a’, ‘b’ and ‘c’
( )	Parentheses define a marked subexpression, also called a block, or a capturing group
\t, \n, \r, \f	Tab, newline, return, form feed
*****	Matches the preceding character zero or more times ab*c matches ‘ac’, ‘abc’, ‘abbc’, and so on [ab]* matches ‘’, ‘a’, ‘b’, ‘ab’, ‘ba’, ‘aba’, and so on (ab)* matches ‘’, ‘ab’, ‘abab’, ‘ababab’, and so on
{m,n}	Matches the preceding character minimum m times, and maximum n times a{2,4} matches ‘aa’, ‘aaa’, and ‘aaaa’
{m}	Matches the preceding character exactly m times
?	Matches the preceding character zero or one times ab?c matches ‘ac’ or ‘abc’

+	Matches the preceding character one or one times ab+c matches 'abc', 'abbc', 'abbbc', and so on, but not 'ac'
|	The choice operator matches either the expression before it, or the one after abc|def matches 'abc' or 'def'
\w	Matches a word character (a-zA-Z0-9) \W matches single non-word characters
\b	Matches the boundary between word and non-word characters
\s	Matches a single whitespace character \S matches a single non-whitespace character
\d	Matches a single decimal digit character (0-9)
\	A single backslash inhibits a character's specialness Examples- \. \\ * When unsure if a character has a special meaning, put a \ before it: \@
$	A dollar matches the end of the string

A raw string literal does not handle backslashes in any special way. For this, prepend an 'r' before the pattern. Without this, you may have to use '\\\\' for a single backslash character. But with this, you only need r'\'.

Regular characters match themselves.

Rules for a Match

So, how does this work? The following rules must be met:

1. The search scans the string start to end.
2. The whole pattern must match, but not necessarily the whole string.
3. The search stops at the first match.

If a match is found, the group() method returns the matching phrase. If not, it returns None.

```
>>> print(re.search('na','no'))
```

None

Python Regular Expression Functions

We have a few functions to help us use Python regex.

a. match()

match() takes two arguments- a pattern and a string. If they match, it returns the string. Else, it returns None. Let's take a few Python regular expression match examples.

```
>>> print(re.match('center','centre'))
```

None

```
>>> print(re.match('...\w\we','centre'))
```

<_sre.SRE_Match object; span=(0, 6), match='centre'>

b. search()

search(), like match(), takes two arguments- the pattern and the string to be searched. Let's take a few examples.

```
>>> match=re.search('aa?yushi','ayushi')
>>> match.group()
```

'ayushi'

```
>>> match=re.search('aa?yushi?','ayush ayushi')
>>> match.group()
```

'ayush'

```
>>> match=re.search('\w*end','Hey! What are your plans for the weekend?')
>>> match.group()
```

'weekend'

```
>>> match=re.search('^\w*end','Hey! What are your plans for the weekend?')
>>> match.group()
```

Traceback (most recent call last):

File "<pyshell#337>", line 1, in <module>

match.group()

AttributeError: 'NoneType' object has no attribute 'group'

Here, an AttributeError raised because it found no match. This is because we specified that this pattern should be at the beginning of the string. Let's try searching for space.

```
>>> match=re.search('i\sS','Ayushi Sharma')
>>> match.group()
```

'i S'

```
>>> match=re.search('\w+c{2}\w*','Occam\'s Razor')
>>> match.group()
```

'Occam'

It really will take some practice to get it into habit what the metacharacters mean. But since we don't have so many, this will hardly take an hour.

Python Regex Options

The functions we discussed may take an optional argument as well. These options are:

a. Python Regular Expression IGNORECASE

This Python Regex ignore case ignores the case while matching.

Take this example of Python Regex IGNORECASE:

```
>>> match=re.findall(r'hi','Hi, did you ship it, Hillary?',re.IGNORECASE)
>>> for i in match:
print(i)
```

Hi

hi

Hi

b. Python MULTILINE

Working with a string of multiple lines, this allows ^ and $ to match the start and end of each line, not just the whole string.

```
>>> match=re.findall(r'^Hi','Hi, did you ship it, Hillary?\nNo, I didn\'t, but Hi',re.MULTILINE)
>>> for i in match:
print(i)
```

Hi

c. Python DOTALL

.* does not scan everything in a multiline string; it only matches the first line. This is because . does not match a newline. To allow this, we use DOTALL.

```
>>> match=re.findall(r'.*','Hi, did you ship it, Hillary?\nNo, I didn\'t, but Hi',re.DOTALL)
>>> for i in match:
print(i)
```

Hi, did you ship it, Hillary?

No, I didn't, but Hi

12. Greedy vs Non-Greedy

The metacharacters *, +, and ? are greedy. This means that they keep searching. Let's take an example.

```
>>> match=re.findall(r'(<.*>)','<em>Strong</em> <i>Italic</i>')
>>> for i in match:
print(i)
```

```
<em>

</em>

<i>

</i>
```

This gave us the whole string, because it greedily keeps searching. What if we just want the opening and closing tags? Look:

print(i)

```
>>> match=re.findall(r'(<.*?>)','<em>Strong</em> <i>Italic</i>')
>>> for i in match:
print(i)
```

```
<em>

</em>

<i>

</i>
```

The .* is greedy, and the ? makes it non-greedy.