Regular Expressions In Python

Many times, we are needed to extract required information from given data. For example, we want to know the number of people who contacted us in the last month through Gmail or we want to know the phone numbers of employees in a company whose names start with 'A' or we want to retrieve the date of births of the patients in a hospital who joined for treatment for hypertension, etc. To get such information, we have to conduct a searching operation on the data. Once we get the required information, we have to extract that data for further use. Regular expressions are useful to perform such operations on data. Let's learn more about regular expressions.

Regular Expressions

A regular expression is a string that contains special symbols and characters to find and extract the information needed by us from the given data. A regular expression helps us to search information, match, find and split information as per our requirements. A regular expression is also called simply regex. Regular expressions are available not only in Python but also in many languages like Java, Perl, AWK, etc.

Python provides re module that stands for regular expressions. This module contains methods like compile(), search(), match(), findall(), split(), etc. which are used in finding the information in the available data. So, when we write a regular expression, we should import re module as:

import re

Regular expressions are nothing but strings containing characters and special symbols. A simple regular expression may look like this:

reg - r'm\w\w’

In the preceding line, the string is prefixed with ‘r' to represent that it is a raw string. Generally, we write regular expressions as raw strings. Let's understand why this is so. When we write a normal string as:

str=’This is normal\nstring’

Now, print(str) will display the preceding string in two lines as:

This is normal
string

Thus the '\n' character is interpreted as a new line in the normal string by the Python interpreter and hence the string is broken there and shown in the new line. In regular expressions when '\n' is used, it does not mean to throw the string into a new line. There ‘\n' has a different meaning and it should not be interpreted as a new line. For this purpose, we should take this as a 'raw' string. This is done by prefixing before the string.

str = r'This is raw\nstring’

When we display this string using print(str), the output will be:

This is raw\nstring

So, the normal meaning of '\n' is escaped and it is no longer an escape character in the preceding example. Since '\n' is not an escape character, it is interpreted as a character with different meaning in the regular expression by the Python interpreter. Similarly, the characters like it. ‘\t’,’\w’,’\c’, etc. should be interpreted as special characters in the regular expressions and hence the expressions should be written as raw strings. If we do not want to write the regular expressions as raw strings, then the alternative is to use another backslash before such characters. For example, we can write:

```
reg=r'm\w\w’   #as raw string
reg='m\\w\\w'  #as normal string
```

But using backslashes like this may be confusing for the programmer. Now, let's go back to our first regular expression:

reg r'm\w\w’

This expression is written in single quotes to represent that it is a string.The first character ‘m' represents that the words starting with 'm' should be matched. The next character "\w' represents any one character in A to 2, a to z and 0 to 9. Since we used two w characters,they represent any two characters after 'm'. So, this regular expression represents words or strings having three characters and with 'm' as the first character. The next two characters can be any alphanumeric.

Yes, we developed our first regular expression! The next step is to compile this expression using compile() method of 're' module as:

```
prog=re.compile(r'm\w\w')
```

Now, prog represents an object that contains the regular expression. The next step is to run this expression on a string 'str' using the search() method or match() method as:

```
str='cat mat bat rat'
result=prog.search(str)
```

The result is stored in 'result object and we can display it by calling the group() method on the object as:

```
print (result.group())
mat
```

This is how a regular expression is created and used. We write all the steps at one place to have better idea:

```
import re
prog=re.compile(r'm\w\w')
str= 'cat mat bat rat'
result=prog. search(str)
print (result.group())
mat
```

In the preceding code, the regular expression after compilation is available in prog object. So, we need not compile the expression again and again when we want to use the same expression on other strings. This will improve the speed of execution. For example, let's use the same regular expression on a different string as:

```
str1= 'Operating system format'
result = prog.search(str1)
print (result.group())
mat
```

Instead of compiling the first regular expression and then running the next one, we can use a single step to compile and run all the regular expression as:

```
result = re.search(r'm\w\w, str)
```

The preceding code is equivalent to:

```
prog=re.compile(r'm\w\w')
result=prog.search(str)
```

So, the general form of writing regular expressions is as follows:

result re.search('expression', 'string')

In the following program, the same regular expression is used on a different string. Consider the following program.

A Python program to create a regular expression to search for strings starting with m and having a total of 3 characters using the search() method.

```
import re
str='man sun mop run'
result=re.search(r'm\w\w', str)
if result:
    print(result.group())
```

Output:

```
===== RESTART: C:/Users/user/AppData/Local/Programs/Python/Python38/reg.py =====
man
>>>
```

Observe the output of the program. The search() method searches for the strings according to the regular expression and returns only the first string. This is the reason that even though there are two strings 'man' and 'mop', only the first one is returned. This string can be extracted using the group() method. In case, the search() method could not find any strings matching the regular expression, then it returns None. So, to display the result, we can use either of the statements given here:

```
if result is not None:
   print (result.group())
```

```
if result:
   print (result.group())
```

Suppose, we want to get all the strings that match the pattern mentioned in the regular expression, we should use findall() method instead of search() method. The final method returns all resultant strings into a list. This is shown in the program below.

A Python program to create a regular expression to search for strings starting with m and having a total of 3 characters using the findall() method.

```
import re
str= 'man sun mop run'
result = re.findall(r'm\w\w', str)
print (result)
```

Output:

```
===== RESTART: (
['man', 'mop']
```

In the program above, the findall() method returned the result as a list. The elements of the list can be displayed using a for loop as:

for s in result:
print(s)
There is another method by the name match() that returns the resultant string only if it is found in the beginning of the string. The match() method will give None if the string is not in the beginning. Let's consider the next two programs.

A Python program to create a regular expression using the match() method to search for strings starting with m and having a total of 3 characters.

```
import re
str = 'man sun mop run'
result=re.match(r'm\w\w', str)
print (result.group())
```

Output:

```
===== RESTART: C:/U
man
>>>
```

A Python program to create a regular expression using the match() method to search for strings starting with m and having a total of 3 characters.

```
import re
str = 'sun man mop run'
result = re.match(r'm\w\w', str)
print (result)
```

Output:

```
===== RESTA
None
>>>
```

There is a method split() that splits the given string into pieces according to the regular expression and returns the pieces as elements of a list. Suppose we write a regular. expression as:

re.split('\W+', str)

Observe the regular expression '\W'. This is capital 'W' which is reverse to the small 'w' so far used. 'w' represents any one alphanumeric character, ie. A-Z, a-z, 0-9. But W represents any character that is not alphanumeric. So, the work of this regular expression is to split the string 'str' at the places where there is no alpha numeric character. The after W represents to match 1 or more occurrences indicated by W. The result is that the string will be split into pieces where 1 or more non alphanumeric characters are found. Consider the following program.

A Python program to create a regular expression to split a string into pieces where one or more non alphanumeric characters are found.

```
import re
str='This; is: "in" Python '
result = re.split(r'\W+', str)
print(result)
```

Output:

```
===== RESTART: C:/Users/user/AppData
['This', 'is', 'in', 'Python', '']
```

Sometimes, regular expressions can also be used to find a string and then replace it with a new string. For this purpose, we should use the sub() method of the 're' module. The format of this method is:

sub(regular expression, new string, string)

For example, sub('Ahmedabad', Allahabad, str) will replace 'Ahmedabad' with 'Allahabad' in the string 'str'. Consider the following program.

A Python program to create a regular expression to replace a string with a new string.

```
import re
str= 'Kumbhmela will be conducted at Ahmedabad in India.'
res= re.sub (r'Ahmedabad', 'Allahabad', str)
print (res)
```

Output:

```
===== RESTART: C:/Users/user/AppData/Local/Programs
Kumbhmela will be conducted at Allahabad in India.
>>>
```

So, regular expressions are used to perform the following important operations: Matching strings

- Searching for strings
- Finding all strings
- Splitting a string into pieces
- Replacing strings

The following methods belong to the 're' module that are used in the regular expressions:

- The match() method searches in the beginning of the string and if the matching string is found, it returns an object that contains the resultant string, otherwise it returns None. We can access the string from the returned object using the group() method.
- The search() method searches the string from beginning till the end and returns the first occurrence of the matching string, otherwise it returns None. We can use the group() method to retrieve the string from the object returned by this method.
- The findall() method searches the string from beginning till the end and returns all occurrences of the matching string in the form of a list object. If the matching strings are not found, then it returns an empty list. We can retrieve the resultant strings from the list using a for loop.
- The split() method splits the string according to the regular expression and the resultant pieces are returned as a list. If there are no string pieces, then it returns an empty list. We can retrieve the resultant string pieces from the list using a for loop.
- The sub() method substitutes (or replaces) new strings in the place of existing strings.After substitution, the main string is returned by this method.

Sequence Characters in Regular Expressions

Sequence characters match only one character in the string. Let's list out the sequence characters which are used in regular expressions along with their meanings.

Character	Description
\d	Represents any digit(0 to 9)
\D	Represents any non-digit
\s	Represents white space.Ex: \t\n\r\f\v
\S	Represents non-white space character

\w	Represents any alphanumeric(A to Z, a to z, 0 to 9)
\W	Represents non-alphanumeric
\b	Represents a space around words
\A	Matches only at start of the string
\Z	Matches only at end of the string

Each of these sequence characters represents a single character matched in the string For example, '\w' indicates any one alphanumeric character. Suppose we write it as [\w]*. Here represents 0 or more repetitions. Hence [\w]* represents 0 or more alphanumeric characters.

Let's write a regular expression to retrieve all words starting with 'a'. This can be written as:

r'a[\w]*

Here, 'a' represents the word that should start with 'a'. Then [\w]* represents repetition of alphanumeric characters. Consider the following program.

A Python program to create a regular expression to retrieve all words starting with a in a given string.

```
import re
str= 'an apple a day keeps the doctor away'
result = re.findall(r'a[\w]*', str)
for word in result:
    print (word)
```

Output:

```
===== RESTART: C:/Users
an
apple
a
ay
away
```

Please observe the output. It contains 'ay' which is not a word. This 'ay' is part of the word 'away'. So, it is displaying both 'ay' and 'away' as they are starting with : We do not want like this. We want only the words starting with 'a'. Since a word will have space in the beginning or ending, we can use 'b' before and after the words in the regular expression. So, the regular expression will become:

result = re.findall (r\ba[\]\b, ste)

This will retrieve only the word and not the part of the words. So, the output will be
an
apple
away

The program below is an attempt to retrieve all the words starting with a numeric digit like 0, 1, 2 or 9. The numeric digit is represented by '\d' and hence, the expression will be:

r\d[\w]*

A Python program to create a regular expression to retrieve all words starting with a numeric digit.

```
import re
str='The meeting will be conducted on 1st and 21st of every month'
result = re.findall(r'\d[\w]*', str)
for word in result:
    print (word)
```

Output:

```
===== RES
1st
21st
```

In the program below, we are trying to retrieve all words having 5 characters length. The expression can be written something like this: r'\b\w{5}\b. The character '\b' represents a space. We used this character in the beginning and ending of the expression so that we get the words surrounded by spaces. \w{5} represents a word containing any alphanumeric characters repeated 5 times. A character in curly braces, e.g. {m} represents repetition for m times.

A Python program to create a regular expression to retrieve all words having 5 characters length.

```
import re
str= 'one two three four five six seven 8 9 10'
result = re.findall(r'\b\w{5}\b', str)
print (result)
```

Output:

```
===== RESTART: C:/Us
['three', 'seven']
>>>
```

In the program above, instead of using the findall() method, if we use the search() method, it will return the first occurrence of the result only.

A Python program to create a regular expression to retrieve all words having 5 characters length using search().

```
import re
str= 'one two three four five six seven 8 9 10'
result = re.search(r'\b\w{5}\b', str)
print(result.group())
```

Output:

```
===== RI
three
```

We will improve our search a little bit further. Now, we want to find all words which are at least 4 characters long. That means words with 4, 5 or any number of characters will be retrieved. For this purpose, we can write the regular expression as: r'\b\w{4}\b'. Observe the number 4 and a comma in curly braces. This represents 4 or above the number of characters.

A Python program to create a regular expression to retrieve all the words that are having the length of at least 4 characters.

```
import re
str='one two three four five six seven 8 9 10'
result = re.findall(r'\b\w{4,}\b', str)
print (result)
```

Output:

```
===== RESTART: C:/Users/user/AppData
['three', 'four', 'five', 'seven']
>>>
```

Similarly, the program below helps us to retrieve all words with 3 to 5 characters length. Observe the curly braces with 3, 5 as: {3,5} that indicate 3 to 5 number of characters

A Python program to create a regular expression to retrieve all words with 3 or 4 or 5 characters length.

```
import re
str='one two three four five six seven 8 9 10'
result = re.findall(r'\b\w{3,5}\b', str)
print (result)
```

Output:

```
===== RESTART: C:/Users/user/AppData/Local/Programs/Pythc
['one', 'two', 'three', 'four', 'five', 'six', 'seven']
```

We know 'd' represents a numeric digit (0 to 9). So, if we use it as: r'\b\d\b', it represents single digits in the string amidst spaces.

A Python program to create a regular expression to retrieve only single digits from a string.

```
import re
str = 'one two three four five six seven 8 9 10'
result = re.findall(r'\b\d\b', str)
print (result)
```

Output:

```
===== RESTART
['8', '9']
```

Suppose in the preceding program, we are the regular expression as: r'\b\d\d\b', it retrieves double digits (like 10, 02, 89 etc.) from the string.

'\A' is useful to match the words at the beginning of a string. Similarly, '\Z' is useful to match the words at the end of a string. For example, we want to find whether a string contains at its end a word starting with 't' or not. We can write an expression as: r't[\w]*\Z'. Here, 't' represents that the word should start with 't'. [\w]* represents any characters after 't'. The last '\Z' represents searching that should be done at the end of the string. Consider the following program.

A Python program to create a regular expression to retrieve the last word of a string, if it starts with t.

```
import re
str= 'one two three one two three'
result=re.findall (r't[\w]*\Z', str)
print (result)
```

Output:

```
===== RESTAR
['three']
```

Quantifiers in Regular Expressions

In regular expressions, some characters represent more than one character to be matched in the string. Such characters are called 'quantifiers'. For example, if we write "+" it represents 1 or more repetitions of the preceding character. Hence, if we write an expression as: r'\d+', this indicates that all numeric digits which occur 1 or more times should be extracted.

Character	Description
+	1 or more repetitions of the preceding regexp
*	0 or more repetitions of the preceding regexp
?	0 or 1 or more repetitions of the preceding regexp
{m}	Exactly m occurrences
{m,n}	From m to n. m defaults to 0. n to infinity

A Python program to create a regular expression to retrieve the phone number of a person.

```
import re
str= 'Chloe: 1234567809'
res=re.search(r'\d+', str)
print (res.group())
```

Output:

```
===== RESTART
1234567809
>>>
```

In the program above, suppose we are asked to retrieve the person's name and not his phone number. Now, how to do that? Very simple. Instead of writing 'd', we use ‘D' in the regular expression as ‘D' represents all characters except numeric characters.

A Python program to create a regular expression to extract only name but not number from a string

```
import re
str='Chloe: 1234567809'
res= re.search(r'\D+', str)
print (res.group())
```

Output:

```
===== RE
Chloe:
>>>
```

The special character ‘+’ represents 1 or more repetitions. Similarly, ‘*’ represents 0 or more repetitions. Suppose, we want to write a regular expression that finds all words starting with either 'an' or 'ak', then we can use: r'a[nk]|\w*’. Here, observe a[nk]. This represents either 'n' or 'k' or both after 'a'.

A Python program to create a regular expression to find all words starting with 'an' or 'ak'.

```
import re
str='anil akhil anant arun arati arundhati abhijit ankur'
res= re.findall(r'a[nk][\W]*', str)
print (res)
```

Output:

```
===== RESTART: C:/Users/user/Ap
['an', 'ak', 'an', 'an', 'an']
>>>
```

We know that {m,n} indicates m to n occurrences. Suppose we take '\d{1,3}', it represents 1 to 3 occurrences of '\d'. Let's see how to use this. We have a string that contains names, id numbers and date of births as:

str = 'Ram 20 1-5-2001, Sita 21 22-10-1990, Lakshman 22 15-09-2000'

Now, we want to retrieve only the date of births of the candidates. We can write a regular expression as: r'\d{2}-\d{2}-\d{4}'. This retrieves only numeric digits in the format of 2digits-2 digits-4 digits. Hence this can be used to retrieve the date of births as shown in the program below.

A Python program to create a regular expression to retrieve date of births from a string.

```
import re
str = 'Ram 20 1-5-2001, Sita 21 22-10-1990, Lakshman 22 15-09-2000'
res= re.findall(r'\d{2}-\d{2}-\d{4}', str)
print (res)
```

Output:

```
===== RESTART: C:/Users/user/
['22-10-1990', '15-09-2000']
>>>
```

Please observe that the date of birth of Ram, i.e. 1-5-2001 is not retrieved. The reason is the date and month here have only one digit. But our regular expression retrieves only if there are 2 digits. So, how to solve this problem? We can modify the regular expression such that it retrieves either 1 or 2 digits in the date or month as:

```
import re
str = 'Ram 20 1-5-2001, Sita 21 22-10-1990, Lakshman 22 15-09-2000'
res=re.findall (r'\d{1,2}-\d{1,2}-\d{4}',str)
print (res)
```

Output:

```
===== RESTART: C:/Users/user/AppData/Local/
['1-5-2001', '22-10-1990', '15-09-2000']
>>>
```

Special Characters in Regular Expressions

Characters with special significance shown in the table below can be used in regular expressions. These characters will make our searching easy.

Character	Description
\	Escape special character nature
.	Matches any character except new line
^	Matches beginning of a string
$	Matches ending of a string
[...]	Denotes a set of possible characters. Ex:[6b-d] matches any characters ‘6’,‘b’,‘c’ or ‘d’
[^...]	Matches every character except the one inside brackets. Ex:[^a-c6] matches any character except ‘a’,’b’,’c’ or ‘6’.
(...)	Matches the regular expression inside the parentheses and the results can be captured.
R|S	Matches either regex R or regex S

The carat (^) symbol is useful to check if a string is starting with a substring or not. For example, to know if a string is starting with 'He' or not, we can write the expression: r”^He”. This is shown in the program below.

A Python program to create a regular expression to search whether a given string is starting with 'He' or not.

```
import re
str = "Hello world"
res = re.search(r"^He", str)
if res:
    print ("String starts with 'He'")
else:
    print("String does not start with 'He'")
```

Output:

```
===== RESTART: C:/Users/u
String starts with 'He'
```

Similarly, to know whether a string is ending with a word, we can use the dollar ($) symbol. Suppose we write an expression as r"World$". This indicates a search for World in the ending of the main string, as shown in the program below.

A Python program to create a regular expression to search for a word at the ending of a string.

```
import re
str = "Hello world"
res = re.search(r"world$", str)
if res:
    print ("String ends with 'world'")
else:
    print("String does not end with 'world'")
```

Output:

```
===== RESTART: C:/Users/us
String ends with 'world'
>>>
```

Regular expressions conduct case sensitive searching for the strings. Hence, in the program above, if we use the expression with a capital ‘W' as: r”World$", then we will end up with wrong output as:

string does not end with ‘world'

We can specify a case insensitive matching of strings with the help of IGNORECASE constant of 're' module. This is shown in the program below.

A Python program to create a regular expression to search at the ending of a string by ignoring the case.

```
import re
str="Hello world"
res= re.search(r"World$", str, re.IGNORECASE)
if res:
    print ("string ends with 'world'")
else:
    print("String does not end with 'world'")
```

Output:

```
string ends with 'world'
>>>
```

The square brackets [] in regular expressions represent a set of characters. For example, if we write [ABC], it represents any one character A or B or C. A hyphen (-) represents a range of characters. For example, [A-Z] represents any single character from the range of capital letters A to Z. Similarly, [a-z] represents any single lowercase letter. This is useful to retrieve names

from a string. For example, a name like 'Rahul' starts with a capital letter and remaining are lowercase letters. To search such strings, we can use a regular expression as: '[A-Z][a-z]*'. This means the first letter should be any capital letter (from A to Z). Then the next letter should be a small letter. Observe the *, it represents 0 or more repetitions of small letters should be considered.

A Python program to create a regular expression to retrieve marks and names from a given string.

```
import re
str = 'Rahul got 75 marks, Vijay got 55 marks, whereas Subbu got 98 marks.'
marks = re.findall('\d{2}', str)
print(marks)
names= re.findall('[A-Z][a-z]*', str)
print(names)
```

Output:

```
['75', '55', '98']
['Rahul', 'Vijay', 'Subbu']
>>>
```

The pipe symbol (|) represents 'or'. For example, if we write 'am | pm', it finds the strings which are either 'am' or 'pm'

A Python program to create a regular expression to retrieve the timings either 'am' or 'pm'.

```
import re
str='The meeting may be at 8am or 9am or 4pm or 5pm.'
res=re.findall(r'\dam|\dpm', str)
print(res)
```

Output:

```
===== RESTART: C:/Users/user/A
['8am', '9am', '4pm', '5pm']
>>>
```