**Regular Expression – Python**

A *regular expression* is a special sequence of characters that helps you match or find other strings or sets of strings [wild match]

Let’s see what regular expression is in general,

In unix, when want to do an operation to multiple file we generally use ‘\*’ like below,

*chmod 777 \*.\**

here asterisk means any character so all the file names will be matched. The same way we use different sequence of characters to match a string.

One of the general use case is, we wanted to grep the price of the product in a retails website says amazon. We have no idea what price is but we have an idea how it is formatted.

i.e, in generally price will be given like, $<dollervalue>.<centvalue>, we we know the format here, we can grep the value

In python regular expression is covered under the module called ***re***.

“re” module has good set of function which functions which helps to process regex on a string.

***match – Function***

This function returns match object (The object which contains all the matched string) and return none if regex not found in string

Syntax

re.match(pattern, string, flags=0)

here pattern is regex to be matched, string is where regex pattern would be searched to match and flags can be have different options which makes search specifications which we will see later.

In the below simple example,

import re

line = "I love Briyani"

matchObj = re.match('.\* love .\*', line)

if matchObj:

print "matchObj.group() : ", matchObj.group()

else:

print "No match!!"

in the above example, we are trying to match 3 words string and middle one is love and this one return the matched string. In this case its entire string (as we used \* it matches everything)

and output will be,

matchObj.group() : I love Briyani

If we need to get matched unknown portion of string, we should use brackets to mention which part we are interested and group function on match object to get them.

See the modified example code below,

import re

line = "I love Briyani"

matchObj = re.match('(.\*) love (.\*)', line)

if matchObj:

print "matchObj.group() : ", matchObj.group()

print "matchObj.group(1) : ", matchObj.group(1)

print "matchObj.group(2) : ", matchObj.group(2)

else:

print "No match!!"

In the above function, we added parenthesis to wild characters and we try to get them used group(num) function.

Output will be

matchObj.group() : I love Briyani

matchObj.group(1) : I

matchObj.group(2) : Briyani

We have one more function called “groups” (notice here it’s plural) with Match object. Which returns the value in tuple (it’s kind of list. All the function parameters are tuples, in my github doc, I used word list for it. Tuple and list are interchangeable and similar but not same). We read about tuples in another time.

As its tuples, we use all list operations including reverse index

Please find the example below,

import re

line = "I love Briyani"

matchObj = re.match('(.\*) love (.\*)', line)

if matchObj:

print "matchObj.groups() : ", matchObj.groups()

else:

print "No match!!"

and output will be,

matchObj.groups() : ('I', 'Briyani')

Flags are generally used to mention like ignore the case while matching, input string is a multiline string likewise. We will see this at the end of the document, for now we can move to next function called search

***search – Function***

match and search are similar and use same set of arguments and return match object.

Before going there, let’s take our previous example and use with search and it works, in the same way

The difference is, match checks for a match only at the beginning of the string, while search checks for a match anywhere in the string

import re

line = "I love Briyani"

matchObj = re.search('(.\*) love (.\*)', line)

if matchObj:

print "matchObj.groups() : ", matchObj.groups()

else:

print "No match!!"

and output is,

matchObj.groups() : ('I', 'Briyani')

Now let’s see what difference is,

match checks for a match only at the beginning of the string, while search as name suggest checks for a match anywhere in the string.

Let’s understand with the example,

import re

line = "I love Briyani";

matchObj = re.match('Briyani', line)

if matchObj:

print "match function output : ", matchObj.group()

else:

print "match function did not find it"

searchObj = re.search('Briyani', line)

if searchObj:

print "search function output : ", searchObj.group()

else:

print "search function did not find it"

in the above example, match is trying to find the word(Regex can be a string) “Briyani” in the given line and match coulnd’t find it because Briyani is not at the beginning. It can find if if we have given regex like,

".\* Briyani"

Whereas search will find it as name suggests its try to search throughout the string and not just beginning of it.

And output will be,

match function did not find it

search function output : Briyani

Let’s move on to next function called sub

***sub – function***

sub means substitute and here it is used to search and replace string. And syntax is similar but instead of flag, it has max parameter which will says how many iterations of replacements needed.

re.sub(pattern, repl, string, max=0)

import re

line = "I love Briyani"

nline = re.sub("B.\*", "Icecream", line)

print "output is : ", nline

In this example, anything starts with B is replaced with icecream so output would be like,

output is : I love Icecream

Let’s understand max argument as well,

import re

line = "ramkumar again ramkumar annoyingly once again ramkumar"

nline = re.sub("ramkumar", "ram", line, 2)

print "output is : ", nline

as we give max is 2, only first 2 ramkumar supposed to be replaced with ram and 3rd one should stay, so output is,

output is : ram again ram annoyingly once again ramkumar

***regex modifiers (Flags)***

let’s talk about skipped topic flags. Multiple flags can be given using “or” operation (“ | “ - pipe symbol)

Following modifiers are most important 3,

re.I - ignore case

re.M - multiline. [means generally $ means end of sting, when this flag used it means end of line if string contains newline character]

re.S - generally "." (dot) matchs any single character except newline and this flag makes "." to match any character including newline

***RegEx Patterns***

Let us see some of the important regex patterns

|  |  |
| --- | --- |
| **Pattern** | **Meaning** |
| ^ | Beginning of the line |
| $ | End of the line |
| . | Matches any character except new line when flag re.S used it matchs new line as well |
| \* | 0 or more occurances of preceding expression |
| + | 1 or more occurances of preceding expression |
| ? | 0 or 1 occurance of preceding expression |
| regex1| regex2 | try to match any of the given regexes |
| ( ) | Groups the match |
| \w | Matches word characters. |
| \W | Matches nonword characters. |
| \s | Matches whitespace. Equivalent to [\t\n\r\f]. |
| \S | Matches nonwhitespace. |
| \d | Matches digits. Equivalent to [0-9]. |
| \D | Matches nondigits. |
| Literal | Literal string can be used and it matched exactly as it is. No special meaning |
| [] | Generally used for character class. Next 8 rows explains the same |
| [Rr]am | Match "Ram" or "ram" |
| [aeiou] | Match any one lowercase vowel |
| [0-9] | Match any digit; same as [0123456789]… hypen can be used to mention sequence of character like numbers, alphabets |
| [a-z] | Match any lowercase ASCII letter |
| [A-Z] | Match any uppercase ASCII letter |
| [a-zA-Z0-9] | Match any of the above |
| [^aeiou] | Match anything other than a lowercase vowel. Carat sign can also be as "other than" in group |
| [^0-9] | Match anything other than a number. Carat sign can also be as "other than" in group |

Basics (All most everything) of the Reg ex had been covered here, once we started to practice the code, we can go get into more complex scenarios.