Regular Expression

A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern.

- The Module **re** provides full support for Regular Expression in Python.
- The re module raises the exception re.error if an error occurs while compiling or using a regular expression.

We would cover two important functions, which would be used to handle regular expressions.They are:

1. **match Function**
2. **search Function**

Basic patterns that match single chars

Expression And Matches	Meaning
a, X, 9, <	ordinary characters just match themselves exactly.
. (a period)	matches any single character except newline '\n'
\w	matches a "word" character: a letter or digit or underbar [a-zA-Z0-9_].
\W	matches any non-word character.
\b	boundary between word and non-word.
\s	matches a single whitespace character -- space, newline, return, tab
\S	matches any non-whitespace character.
\t, \n, \r	tab, newline, return
\d	decimal digit [0-9]
^	matches start of the string.
$	match the end of the string.
\	inhibit the "specialness" of a character.

The match Function

- This function attempts to match RE pattern to string with optional flags.
  Here is Syntax:

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

Here is the description of parameters:

1. pattern

```
- This is the regular expression to be matched.
```

2. string

```
- This is the string, which would be searched to match the pattern at the
beginning of string.
```

3. flags

```
 - You can specify different flags using bitwise OR (|). These are modifier
s, which are listed in the table below.
```

The re.match function returns a match object on success, None on failure. We use group(num) or groups() function of match object to get matched expression.

group(num = 0)

- This method returns entire match (or specific subgroup num)

groups()

- This method returns all matching subgroups in a tuple (empty if there weren't any)

Example:

In [1]:

```
import re
line = "A quick Brown Fox Jumps Over A lazy Dog"
matchObj = re.match( r'(.*) over (.*?)', line, re.M|re.I)
if matchObj:
    print(matchObj.groups())
    print("matchObj(0) is :",matchObj.group(0))
    print("matchObj(1) is :",matchObj.group(1))
    print("matchObj(2) is :",matchObj.group(2))
else:
    print ("Nothing found!!")
print(matchObj.span())
```

```
('A quick Brown Fox Jumps', '')
matchObj(0) is : A quick Brown Fox Jumps Over
matchObj(1) is : A quick Brown Fox Jumps
matchObj(2) is :
(0, 29)
```

The search Function

- This function attempts to match RE pattern to string with optional flags.

Syntax:

```
re.search(pattern, string, flags = 0)
```

In [2]:

```
import re

line = "A quick Brown Fox Jumps Over A lazy Dog"

searchObj = re.search( r'(.*) Jumps (.*?) .*', line, re.M|re.I)

if searchObj:
   print ("searchObj.groups() : ", searchObj.groups())
   print ("searchObj.group() : ", searchObj.group())
   print ("searchObj.group(1) : ", searchObj.group(1))
   print ("searchObj.group(2) : ", searchObj.group(2))
else:
   print ("Nothing found!!")
```

```
searchObj.groups() :  ('A quick Brown Fox', 'Over')
searchObj.group() :  A quick Brown Fox Jumps Over A lazy Dog
searchObj.group(1) :  A quick Brown Fox
searchObj.group(2) :  Over
```

Matching Versus Searching

Python offers two different primitive operations based on regular expressions: match checks for a match only at the beginning of the string, while search checks for a match anywhere in the string (this is what Perl does by default).

In [3]:

```
import re

line = "Welcome to Python course";

matchObj = re.match( r'Python', line, re.M|re.I)
if matchObj:
   print (matchObj.group())
else:
   print ("No match!!")

searchObj = re.search( r'Python', line, re.M|re.I)
if searchObj:
   print (searchObj.group())
else:
   print ("Nothing found!!")
```

```
No match!!
Python
```

Search and Replace

One of the most important re methods that use regular expressions is sub.

Syntax :

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

This method replaces all occurrences of the RE pattern in string with repl, substituting all occurrences unless max is provided. This method returns modified string.

In [4]:

```
#replace C with python
string= "Welcome to C course"
num = re.sub(r'C', "Python", string)
print ("String is : ", num)
```

```
String is :  Welcome to Python course
```

In [5]:

```
# Remove anything other than digits
phone= "for any enquiry please contact 95666-9556666"
num = re.sub(r'\D', "", phone)
print ("Phone Num : " , num)
```

```
Phone Num :  956669556666
```

Regular Expression Modifiers: Option Flags

Modifier	Description
re.I	Performs case-insensitive matching.
re.L	Interprets words according to the current locale. This interpretation affects the alphabetic group (\w and \W), as well as word boundary behavior (\b and \B).
re.M	Makes $ match the end of a line (not just the end of the string) and makes ^ match the start of any line (not just the start of the string).
re.S	Makes a period (dot) match any character, including a newline.
re.U	Interprets letters according to the Unicode character set. This flag affects the behavior of \w, \W, \b, \B.
re.X	Permits "cuter" regular expression syntax. It ignores whitespace (except inside a set [ ] or when escaped by a backslash) and treats unescaped # as a comment marker.

Regular Expression Patterns

Except for the control characters, (+ ? . * ^ $ ( ) [ ] { } |), all characters match themselves. You can escape a control character by preceding it with a backslash.

Character Classes

In [6]:

```
import re
string = "A quick brown fox jumps over a lazy Dog."
re.findall('Dog',string) #Match "Dog"
```

Out[6]:

```
['Dog']
```

In [7]:

```
# match "Dog" or "dog"
print(re.findall('[dD]og',string))
string = "A quick brown fox jumps over a lazy dog."
print(re.findall('[dD]og',string))
```

```
['Dog']
['dog']
```

In [8]:

```
#Match any one lowercase vowel
string = "A quick brown fox jumps over a lazy Dog."
print(re.findall(r'[aeiou]',string))

#Match anything other than a lowercase vowel
print(re.findall(r'[^aeiou]',string))
```

```
['u', 'i', 'o', 'o', 'u', 'o', 'e', 'a', 'a', 'o']
['A', ' ', 'q', 'c', 'k', ' ', 'b', 'r', 'w', 'n', ' ', 'f', 'x', ' ',
'j', 'm', 'p', 's', ' ', 'v', 'r', ' ', ' ', 'l', 'z', 'y', ' ', 'D',
'g', '.']
```

In [9]:

```
#Match any digit;
string = "cube of 4 is 64"
print(re.findall(r'[0-9]',string))  #same as [0123456789]
print(re.findall(r'[0-9][0-9]',string)) #consecutive occurence of two digit

#Match anything other than a digit
print(re.findall(r'[^0-9]',string))
```

```
['4', '6', '4']
['64']
['c', 'u', 'b', 'e', ' ', 'o', 'f', ' ', ' ', 'i', 's', ' ']
```

In [10]:

```
string = "A quick brown fox jumps over a lazy Dog."
print(re.findall(r'[a-z]',string))   #Match any lowercase ASCII letter
print(re.findall(r'[A-Z]',string ))  #Match any uppercase ASCII letter
```

```
['q', 'u', 'i', 'c', 'k', 'b', 'r', 'o', 'w', 'n', 'f', 'o', 'x', 'j',
'u', 'm', 'p', 's', 'o', 'v', 'e', 'r', 'a', 'l', 'a', 'z', 'y', 'o',
'g']
['A', 'D']
```

In [11]:

```
#Match any of the above
string = "phone is: ind/#92501444"
print(re.findall(r'[a-zA-Z0-9]',string))
```

```
['p', 'h', 'o', 'n', 'e', 'i', 's', 'i', 'n', 'd', '9', '2', '5', '0',
'1', '4', '4', '4']
```

Program to extract out Name and Age from String

In [12]:

```
import re

NameAge = ''' Ram is 10 and he is fond of listening music
              Mohan is 12 and he loves to play cricket
              Manish is 15 and he is studious
              Sooraj is 5 and youngest among all.'''

ages = re.findall(r'\d{1,3}',NameAge)
names = re.findall(r'[A-Z][a-z]*',NameAge)

agedict = dict(zip(names, ages))
print(agedict)
```

```
{'Mohan': '12', 'Sooraj': '5', 'Ram': '10', 'Manish': '15'}
```

Filter Email

In [13]:

```
import re
email = "abc@gmail.com ab.com hi.com py@.com pqr21@yahoo.com "
matches = re.findall(r'[\w._%+-]{1,20}@[\w]{2,20}.[A-Za-z]{2,3}',email)
print( "Number of Valid Email Address are:", len(matches))
print( "Valid Email Address are :" , matches)
```

```
Number of Valid Email Address are: 2
Valid Email Address are : ['abc@gmail.com', 'pqr21@yahoo.com']
```

Web Scraping

Example 1

In [14]:

```
import urllib.request
import re

url = 'http://pythonprogramming.net/parse-website-using-regular-expressions-urllib/

req = urllib.request.Request(url)
resp = urllib.request.urlopen(req)
respData = resp.read()
```

In [15]:

```
paragraphs = re.findall(r'<p>(.*?)</p>',str(respData))
```

In [16]:

```
for eachP in paragraphs:
    print(eachP)
```

```
In this video, we use two of Python 3\'s standard library modules, r
e and urllib, to parse paragraph data from a website. As we saw, ini
tially, when you use Python 3 and urllib to parse a website, you get
all of the HTML data, like using "view source" on a web page. This H
TML data is great if you are viewing via a browser, but is incredibl
y messy if you are viewing the raw source. For this reason, we need
to build something that can sift through the mess and just pull the
article data that we are interested in. There are some web scraping
libraries out there, namely BeautifulSoup, which are aimed at doing
this same sort of task.
On to the code:
Up to this point, everything should look pretty typical, as you\'ve
seen it all before. We specify our url, our values dict, encode the
values, build our request, make our request, and then store the requ
est to respData. We can print it out if we want to see what we\'re w
orking with. If you are using an IDE, sometimes printing out the sou
rce code is not the greatest idea. Many webpages, especially larger
ones, have very large amounts of code in their source. Printing all
of this out can take quite a while in the IDLE. Personally, I prefer
```

Example 2

In [17]:

```
import urllib.request
import re

url = 'https://www.goodreads.com/genres/art'

req = urllib.request.Request(url)
resp = urllib.request.urlopen(req)
respData = resp.read()
```

In [18]:

```
paragraphs = re.findall(r'<div class="description descriptionContainer">(.*?)</div>
paragraphs1 = re.findall(r'href=".*?"',str(paragraphs))
```

In [19]:

```
for eachP in paragraphs1:
    print(eachP)
```

```
href="https://www.goodreads.com/book/show/37683210-daemon-voices"
href="https://www.goodreads.com/author/show/3618.Philip_Pullman"
href="https://www.goodreads.com/book/show/36677251-fantasy-snowflakes-
coloring-book"
href="https://www.goodreads.com/author/show/8283255.J_S_Burke"
href="https://www.goodreads.com/book/show/39732774-bad-environmentalis
m"
href="https://www.goodreads.com/author/show/6561744.Nicole_Seymour"
```

In [ ]:

```

```