Regular Expressions

If we want to represent a group of Strings according to a particular format/pattern then we should go for Regular Expressions.

i.e Regualr Expressions is a declarative mechanism to represent a group of Strings accroding to particular format/pattern.

Eg 1: We can write a regular expression to represent all mobile numbers
Eg 2: We can write a regular expression to represent all mail ids.

The main important application areas of Regular Expressions are

1. **To develop validation frameworks/validation logic**
2. **To develop Pattern matching applications (ctrl-f in windows, grep in UNIX etc)**
3. **To develop Translators like compilers, interpreters etc**
4. **To develop digital circuits**
5. **To develop communication protocols like TCP/IP, UDP etc.**

We can develop Regular Expression Based applications by using python module: re
This module contains several inbuilt functions to use Regular Expressions very easily in our applications.

1. compile()

re module contains compile() function to compile a pattern into RegexObject.

```
pattern = re.compile("ab")
```

2. finditer():

Returns an Iterator object which yields Match object for every Match

```
matcher = pattern.finditer("abaababa")
```

On Match object we can call the following methods.

1. **start() ➔ Returns start index of the match**
2. **end() ➔ Returns end+1 index of the match**
3. **group() ➔ Returns the matched string**

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Eg:

```
1)  import re  count=0
2)  pattern=re.compile("ab")
3)  matcher=pattern.finditer("abaababa")
4)  for match in matcher:
5)      count+=1
6)      print(match.start(),"...",match.end(),"...",match.group())
7)  print("The number of occurrences: ",count)
```

Output:

```
0 ... 2 ... ab
3 ... 5 ... ab
5 ... 7 ... ab
The number of occurrences:  3
```

Note: We can pass pattern directly as argument to finditer() function.

Eg:

```
1)  import re
2)  count=0
3)  matcher=re.finditer("ab","abaababa")
4)  for match in matcher:
5)      count+=1
6)      print(match.start(),"...",match.end(),"...",match.group())
7)  print("The number of occurrences: ",count)
```

Output:

```
0 ... 2 ... ab
3 ... 5 ... ab
5 ... 7 ... ab
The number of occurrences:  3
```

Character classes:

We can use character classes to search a group of characters

1. [abc]===>Either a or b or c
2. [^abc] ===>Except a and b and c
3. [a-z]==>Any Lower case alphabet symbol
4. [A-Z]===>Any upper case alphabet symbol
5. [a-zA-Z]==>Any alphabet symbol
6. [0-9] Any digit from 0 to 9
7. [a-zA-Z0-9]==>Any alphanumeric character
8. [^a-zA-Z0-9]==>Except alphanumeric characters(Special Characters)

Eg:

```
1)  import re
2)  matcher=re.finditer("x","a7b@k9z")
3)  for match in matcher:
4)      print(match.start(),"......",match.group())
```

x = [abc]
0 ..... a
2 ..... b

x = [^abc]
1 7
3 ..... @
4 ..... k
5 9
6 ..... z

x = [a-z]
0 ..... a
2 ..... b
4 ..... k
6 ..... z

x = [0-9]
1 7
5 9

x = [a-zA-Z0-9]
0 ..... a
1 7
2 ..... b
4 ..... k
5 9
6 ..... z

x = [^a-zA-Z0-9]
3 ..... @

Pre defined Character classes:

\s ➔ Space character
\S ➔ Any character except space character
\d ➔ Any digit from 0 to 9
\D ➔ Any character except digit
\w ➔ Any word character [a-zA-Z0-9]
\W ➔ Any character except word character (Special Characters)
. ➔ Any character including special characters

Eg:

```
1)  import re
2)  matcher=re.finditer("x","a7b k@9z")
3)  for match in matcher:
4)      print(match.start(),"......",match.group())
```

x = \s:
3 ......

x = \S:
0 ..... a
1 7
2 ..... b
4 ..... k
5 ..... @
6 9
7 ..... z

x = \d:
1 7
6 9

x = \D:
0 ..... a
2 ..... b
3 .....
4 ..... k
5 ..... @
7 ..... z

x = \w:
0 ..... a
1 7
2 ..... b
4 ..... k
6 9
7 ..... z

x = \W:
3 .....
5 ..... @

x = .
0 ..... a
1 7
2 ..... b
3 .....

4 ..... k
5 ..... @
6 9
7 ..... z

Qunatifiers:

We can use quantifiers to specify the number of occurrences to match.

a ➔ Exactly one 'a'
a+ ➔ Atleast one 'a'
a* ➔ Any number of a's including zero number
a? ➔ Atmost one 'a' ie either zero number or one number
a{m} ➔ Exactly m number of a's
a{m,n} ➔ Minimum m number of a's and Maximum n number of a's

Eg:

```
1) import re
2) matcher=re.finditer("x","abaabaaab")
3) for match in matcher:
4)     print(match.start(),".....",match.group())
```

x = a:
0 ..... a
2 ..... a
3 ..... a
5 ..... a
6 ..... a
7 ..... a

x = a+:
0 ..... a
2 ..... aa
5 ..... aaa

x = a*:
0 ..... a
1 .....
2 ..... aa
4 .....
5 ..... aaa
8 .....
9 .....

x = a?:
0 ..... a
1 .....
2 ..... a
3 ..... a
4 .....
5 ..... a
6 ..... a
7 ..... a
8 .....
9 .....

x = a{3}:
5 ..... aaa

x = a{2,4}:
2 ..... aa
5 ..... aaa

Note:
^x ➔ It will check whether target string starts with x or not
x$ ➔ It will check whether target string ends with x or not

Important functions of re module:

1. match()
2. fullmatch()
3. search()
4.findall()
5.finditer()
6. sub()
7.subn()
8. split()
9. compile()

1. match():

We can use match function to check the given pattern at beginning of target string.
If the match is available then we will get Match object, otherwise we will get None.

Eg:

```
1)  import re
2)  s=input("Enter pattern to check: ")
3)  m=re.match(s,"abcabdefg")
4)  if m!= None:
5)      print("Match is available at the beginning of the String")
6)      print("Start Index:",m.start(), "and End Index:",m.end())
```

```
7)  else:
8)      print("Match is not available at the beginning of the String")
```

Output:

```
D:\python_classes>py test.py
Enter pattern to check: abc
Match is available at the beginning of the String
Start Index: 0 and End Index: 3

D:\python_classes>py test.py
Enter pattern to check: bde
Match is not available at the beginning of the String
```

2. fullmatch():

We can use fullmatch() function to match a pattern to all of target string. i.e complete string should be matched according to given pattern.
If complete string matched then this function returns Match object otherwise it returns None.

Eg:

```
1)  import re
2)  s=input("Enter pattern to check: ")
3)  m=re.fullmatch(s,"ababab")
4)  if m!= None:
5)      print("Full String Matched")
6)  else:
7)      print("Full String not Matched")
```

Output:

```
D:\python_classes>py test.py
Enter pattern to check: ab
Full String not Matched

D:\python_classes>py test.py
Enter pattern to check: ababab
Full String Matched
```

3. search():

We can use search() function to search the given pattern in the target string.
If the match is available then it returns the Match object which represents first occurrence of the match.
If the match is not available then it returns None

Eg:

```
1)  import re
2)  s=input("Enter pattern to check: ")
3)  m=re.search(s,"abaaaba")
4)  if m!= None:
5)      print("Match is available")
6)      print("First Occurrence of match with start index:",m.start(),"and end index:",m.end())
7)  else:
8)      print("Match is not available")
```

Output:

```
D:\python_classes>py test.py
Enter pattern to check: aaa
Match is available
First Occurrence of match with start index: 2 and end index: 5

D:\python_classes>py test.py
Enter pattern to check: bbb
Match is not available
```

4. findall():

To find all occurrences of the match.
This function returns a list object which contains all occurrences.

Eg:

```
1)  import re
2)  l=re.findall("[0-9]","a7b9c5kz")
3)  print(l)
```

Output: ['7', '9', '5']

5. finditer():

Returns the iterator yielding a match object for each match.
On each match object we can call start(), end() and group() functions.

Eg:

```
1)  import re
2)  itr=re.finditer("[a-z]","a7b9c5k8z")
3)  for m in itr:
4)      print(m.start(),"...",m.end(),"...",m.group())
```

Output:

```
D:\python_classes>py test.py
0 ... 1 ... a
2 ... 3 ... b
4 ... 5 ... c
6 ... 7 ... k
8 ... 9 ... z
```

6. sub():

sub means substitution or replacement

re.sub(regex,replacement,targetstring)
In the target string every matched pattern will be replaced with provided replacement.

Eg:

```
1) import re
2) s=re.sub("[a-z]","#","a7b9c5k8z")
3) print(s)
```

Output: #7#9#5#8#

Every alphabet symbol is replaced with # symbol

7. subn():

It is exactly same as sub except it can also returns the number of replacements.
This function returns a tuple where first element is result string and second element is number of replacements.
(resultstring, number of replacements)

Eg:

```
1) import re
2) t=re.subn("[a-z]","#","a7b9c5k8z")
3) print(t)
4) print("The Result String:",t[0])
5) print("The number of replacements:",t[1])
```

Output:

```
D:\python_classes>py test.py
('#7#9#5#8#', 5)
The Result String: #7#9#5#8#
The number of replacements: 5
```

8. split():

If we want to split the given target string according to a particular pattern then we should go for split() function.
This function returns list of all tokens.

Eg:

```
1) import re
2) l=re.split(",","sunny,bunny,chinny,vinny,pinny")
3) print(l)
4) for t in l:
5)     print(t)
```

Output:

```
D:\python_classes>py test.py
['sunny', 'bunny', 'chinny', 'vinny', 'pinny']
sunny
bunny
chinny
vinny
pinny
```

Eg:

```
1) import re
2) l=re.split("\.","www.durgasoft.com")
3) for t in l:
4)     print(t)
```

Output:

```
D:\python_classes>py test.py
www
durgasoft
com
```

^ symbol:

We can use ^ symbol to check whether the given target string starts with our provided pattern or not.

Eg:

```
res=re.search("^Learn",s)
```

if the target string starts with Learn then it will return Match object,otherwise returns None.

test.py:

```
1) import re
2) s="Learning Python is Very Easy"
3) res=re.search("^Learn",s)
4) if res != None:
5)     print("Target String starts with Learn")
6) else:
7)     print("Target String Not starts with Learn")
```

Output: Target String starts with Learn

$ symbol:

We can use $ symbol to check whether the given target string ends with our provided pattern or not

Eg: res=re.search("Easy$",s)

If the target string ends with Easy then it will return Match object,otherwise returns None.

test.py:

```
1) import re
2) s="Learning Python is Very Easy"
3) res=re.search("Easy$",s)
4) if res != None:
5)     print("Target String ends with Easy")
6) else:
7)     print("Target String Not ends with Easy")
```

Output: Target String ends with Easy

Note: If we want to ignore case then we have to pass 3rd argument re.IGNORECASE for search() function.

Eg: res = re.search("easy$",s,re.IGNORECASE)

test.py:

```
1) import re
2) s="Learning Python is Very Easy"
3) res=re.search("easy$",s,re.IGNORECASE)
4) if res != None:
5)     print("Target String ends with Easy by ignoring case")
6) else:
7)     print("Target String Not ends with Easy by ignoring case")
```

Output: Target String ends with Easy by ignoring case

App1: Write a Regular Expression to represent all Yava language identifiers

Rules:

1. The allowed characters are a-z,A-Z,0-9,#
2. The first character should be a lower case alphabet symbol from a to k
3. The second character should be a digit divisible by 3
4. The length of identifier should be atleast 2.

[a-k][0369][a-zA-Z0-9#]*

App2: Write a python program to check whether the given string is Yava language identifier or not?

```
1)  import re
2)  s=input("Enter Identifier:")
3)  m=re.fullmatch("[a-k][0369][a-zA-Z0-9#]*",s)
4)  if m!= None:
5)      print(s,"is valid Yava Identifier")
6)  else:
7)      print(s,"is invalid Yava Identifier")
```

Output:

```
D:\python_classes>py test.py
Enter Identifier:a6kk9z##
a6kk9z## is valid Yava Identifier

D:\python_classes>py test.py
Enter Identifier:k9b876
k9b876 is valid Yava Identifier

D:\python_classes>py test.py
Enter Identifier:k7b9
k7b9 is invalid Yava Identifier
```

App3: Write a Regular Expression to represent all 10 digit mobile numbers.

Rules:

1. Every number should contains exactly 10 digits
2. The first digit should be 7 or 8 or 9

[7-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]
or

[7-9][0-9]{9}
or
[7-9]\d{9}

App4: Write a Python Program to check whether the given number is valid mobile number or not?

```
1) import re
2) n=input("Enter number:")
3) m=re.fullmatch("[7-9]\d{9}",n)
4) if m!= None:
5)     print("Valid Mobile Number")
6) else:
7)     print("Invalid Mobile Number")
```

Output:

```
D:\python_classes>py test.py
Enter number:9898989898
Valid Mobile Number

D:\python_classes>py test.py
Enter number:6786786787
Invalid Mobile Number

D:\python_classes>py test.py
Enter number:898989
Invalid Mobile Number
```

App5: Write a python program to extract all mobile numbers present in input.txt where numbers are mixed with normal text data

```
1) import re
2) f1=open("input.txt","r")
3) f2=open("output.txt","w")
4) for line in f1:
5)     list=re.findall("[7-9]\d{9}",line)
6)     for n in list:
7)         f2.write(n+"\n")
8) print("Extracted all Mobile Numbers into output.txt")
9) f1.close()
10) f2.close()
```

Web Scraping by using Regular Expressions:

The process of collecting information from web pages is called web scraping. In web scraping to match our required patterns like mail ids, mobile numbers we can use regular expressions.

Eg:

```
1)  import re,urllib
2)  import urllib.request
3)  sites="google rediff".split()
4)  print(sites)
5)  for s in sites:
6)      print("Searching...",s)
7)      u=urllib.request.urlopen("http://"+s+".com")
8)      text=u.read()
9)      title=re.findall("<title>.*</title>",str(text),re.I)
10)     print(title[0])
```

Eg: Program to get all phone numbers of redbus.in by using web scraping and regular expressions

```
1)  import re,urllib
2)  import urllib.request
3)  u=urllib.request.urlopen("https://www.redbus.in/info/contactus")
4)  text=u.read()
5)  numbers=re.findall("[0-9-]{7}[0-9-]+",str(text),re.I)
6)  for n in numbers:
7)      print(n)
```

Q. Write a Python Program to check whether the given mail id is valid gmail id or not?

```
1)  import re
2)  s=input("Enter Mail id:")
3)  m=re.fullmatch("\w[a-zA-Z0-9_.]*@gmail[.]com",s)
4)  if m!=None:
5)      print("Valid Mail Id");
6)  else:
7)      print("Invalid Mail id")
```

Output:

```
D:\python_classes>py test.py
Enter Mail id:durgatoc@gmail.com
Valid Mail Id
```

```
D:\python_classes>py test.py
Enter Mail id:durgatoc
Invalid Mail id

D:\python_classes>py test.py
Enter Mail id:durgatoc@yahoo.co.in
Invalid Mail id
```

Q. Write a python program to check whether given car registration number is valid Telangana State Registration number or not?

```
1) import re
2) s=input("Enter Vehicle Registration Number:")
3) m=re.fullmatch("TS[012][0-9][A-Z]{2}\d{4}",s)
4) if m!=None:
5)     print("Valid Vehicle Registration Number");
6) else:
7)     print("Invalid Vehicle Registration Number")
```

Output:

```
D:\python_classes>py test.py
Enter Vehicle Registration Number:TS07EA7777
Valid Vehicle Registration Number

D:\python_classes>py test.py
Enter Vehicle Registration Number:TS07KF0786
Valid Vehicle Registration Number

D:\python_classes>py test.py
Enter Vehicle Registration Number:AP07EA7898
Invalid Vehicle Registration Number
```

Q. Python Program to check whether the given mobile number is valid OR not (10 digit OR 11 digit OR 12 digit)

```
1) import re
2) s=input("Enter Mobile Number:")
3) m=re.fullmatch("(0|91)?[7-9][0-9]{9}",s)
4) if m!=None:
5)     print("Valid Mobile Number");
6) else:
7)     print("Invalid Mobile Number")
```

Summary table and some more examples.