Python - Regular Expressions

1. Regular expression deals with

a. If developer wants to represents a group of strings according to a particular pattern

b. To represent all mobile no’s, mail id’s etc…

c. To represent all java expressions or python expressions

2. Applications of Regular Expressions

a. Validations

b. find 🡪 command in Windows

c. grep, egrep, fgrep🡪 commands in Unix evntetc use regular expressions.

d. Lexical Analysis 🡪 Scanning or Tokenization

e. Syntax Analysis🡪 parsing

f. ICG🡪 Intermediate Code Generation

g. CO🡪 Code Optimization

h. TCG🡪 Target Code Generation implementations

i. Translators like compilers, interpreters and assemblers we use RE.

j. To develop digital circuits🡪 Finite Automata for Moore and Melay machines, Binary Incremental

k. Binary Adder

l. Communication protocols TCP/IP etc…

m. Compare passwords, Generate OTP.

NOTE: To use RE’s in python we go for module named ‘re’.

‘re’ module imp functions:

1. compile() 🡪converts the given input to a RE format.
2. finditer() 🡪 returns an iterator object after finding in the given pattern object.
3. start() 🡪 start index of the match
4. end() 🡪 end+1 index of the match
5. group() 🡪 returns matched string.

Example 1:

import re

pattern = re.compile('Python')

print(type(pattern))

Example 2:

import re

count = 0

pattern = re.compile('sa')

matcher = pattern.finditer('sankar is having sambar-idly in samarlakota')

for m in matcher:

print('Match available at index: ', m.start())

count = count + 1

print("Total matches are: ", count)

Example 3:

import re

count = 0

pattern = re.compile('sa')

matcher = pattern.finditer('sankar is having sambar-idly in samarlakota')

for m in matcher:

count += 1

print("Start is: {}, End is: {}, Group is: {}".format(m.start(), m.end(), m.group()))

print("Total no of occurances: ", count)

#Here, end() function gives last index of the pattern match with +1 index

Example 4:

import re

#easy way for the above prog

count = 0

#pattern = re.compile('sa')     #not required

matcher = re.finditer('sa','sankar is having sambar-idly in samarlakota')

for m in matcher:

count += 1

print("Start is: {}, End is: {}, Group is: {}".format(m.start(), m.end(), m.group()))

print("Total no of occurances: ", count)

#Here, end() function gives last index of the pattern match with +1 index

**Character Classes in Python:**

1. To search for a character or a number or a lower or upper case characters in the given pattern we can use ‘character classes’.

2. [abc] 🡪 either a or b or c

[^abc] 🡪 Except a and b and c

[a-z] 🡪 any lower case alphabet

[A-Z] 🡪 Any upper case alphabet

[a-zA-Z] 🡪 Any alphabet symbol

[0-9] 🡪 0 to 9 numbers

[a-zA-Z0-9] 🡪 any alphanumeric characters

[^a-zA-Z0-9] 🡪 except any alphanumeric characters(for special symbols)

Example 1:

import re

# To get the a or b or c

matcher = re.finditer('[abc]', 'a7b@k9z')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

# To get characters except abc

matcher = re.finditer('[^abc]', 'a7b@k9z')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

# To get any alphabet symbol from a-z

matcher = re.finditer('[a-z]', 'a7b@k9z')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

# To get the digits from 0-9

matcher = re.finditer('[0-9]', 'a7b@k9z')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

# To get the digits from 0-9 and alphabets from a-z or A-Z

matcher = re.finditer('[a-zA-Z0-9]', 'a7b@k9z')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

# To get only special symbols

matcher = re.finditer('[^a-zA-Z0-9]', 'a7b@k9z')

for m in matcher:

    print(m.start(), '----', m.group())

**Pre-defined Character classes:**

1. \s 🡪 search for space character(white scape)
2. \S 🡪 except space character any other character.
3. \d 🡪 any digit from 0-9
4. \D 🡪 except digits any character
5. \w 🡪 any word character(any a-z or A-Z or 0-9)
6. \W 🡪 except alpha numeric(i:e, special characters only)
7. . 🡪 any character

Example 1:

import re

#To get the white space

matcher = re.finditer('\s', 'a7b @k 9z')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

#To get characters except white space

matcher = re.finditer('\S', 'a 7 b @k 9z')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

#To get any alphabet symbol from a-z

matcher = re.finditer('\D', 'a7b@k9z')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

#To get only digits

matcher = re.finditer('\d', 'a7b@k9z')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

#To get the digits from 0-9 and alphabets from a-z or A-Z

matcher = re.finditer('\w', 'a7 b@k9z')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

#To get only special symbols

matcher = re.finditer('\W', 'a 7b@k 9Z')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

#To get only special symbols

matcher = re.finditer('.', 'a 7b@k 9Z')

for m in matcher:

    print(m.start(), '----', m.group())

**Quantifiers:**

1. These are used to specify no of occurrences of a match in the given target string.
2. If string ‘a’ is taken as a Pattern/RE to be matched, then
   1. a🡪 Exactly one a is considered as a match
   2. a+ 🡪Atleast one a
   3. a\* 🡪 any no of a’s including number of not ‘a’ also
   4. a? 🡪atmost one ‘a’
   5. a{n} 🡪Exactaly n-numbers of a’s are only considered
   6. a{m,n} 🡪 minimum m-no of a’s and maximum n-no of a’s are considered

are considered as ‘Quantifiers’.

1. Here, the symbols used are called as Quantifiers.

Example:

import re

#Quantifiers in Python

# Exactly one a is considered as a match

matcher = re.finditer('a','abaabaaab')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

# Atleast one a

matcher = re.finditer('a+','abaabaaab')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

# any no of a’s including number of not ‘a’ also

# In python lastindex + 1 index is considered

matcher = re.finditer('a\*','abaabaaab')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

# atmost one ‘a’

# In python lastindex + 1 index is considered

matcher = re.finditer('a?','abaabaaab')

for m in matcher:

    print(m.start(), '----', m.group())

import re

#Quantifiers in Python

# Exactly n-number of 'a's should be considered as a match

matcher = re.finditer('a{3}','abaabaaab')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

# minimum 2-no of a’s and maximum 3-no of a’s are considered

matcher = re.finditer('a{2,3}','abaabaaabaaaab')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

# two a's are mandatory and then after any no of a's are considered

matcher = re.finditer('a{2}a\*','abaabaaab')

for m in matcher:

    print(m.start(), '----', m.group())

import re

#Quantifiers in Python

# Except 'a' all the remaining

matcher = re.finditer('[^a]','abaabaaab')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

# Weather the given target string starts with 'a' or not

matcher = re.finditer('^a','abaabaaabaaaab')

for m in matcher:

    print(m.start(), '----', m.group())

print('=============================')

# Weather the given target string ends with 'a' or not

matcher = re.finditer('a$','abaabaaaba')

for m in matcher:

    print(m.start(), '----', m.group())

**Important functions of ‘re’ module:**

* **match()**🡪 To check the given pattern is available at the beginning of the target string or not.
  + If available then return match object, else return None

Example 1:

import re

s = input('Enter pattern to check: ')

m = re.match(s, 'abcdefghijklmnopqr')

if m != None:

    print('Match is available at beginning of the string...')

    print("Start index is: {} and end index is: {}".format(m.start(), m.end()))

else:

    print("Match is not available at the beginning of the string...")

Example 2:

**fullmatch()🡪**returns match object when complete string matches, otherwise return None.

import re

s = input('Enter pattern to check: ')

m = re.fullmatch(s, 'abcdefghijklmnopqr')

if m != None:

    print('Full string matched')

    print("Start index is: {} and end index is: {}".format(m.start(), m.end()))

else:

    print("Full string not matched...")

Example 3:

* **search()🡪returns match object if search string found, else returns None**

import re

s = input('Enter pattern to check: ')

m = re.search(s, 'abaacdbbefaaagaahi')

if m != None:

    print('Match is found...')

    print("Start index is: {} and end index is: {}".format(m.start(), m.end()))

else:

    print("Match not found...")

Example 4:

* **findall()🡪 finds all available matches and store them into a list, else returns empty list.**

import re

l = re.findall('[0-9]', 'abc de@#$KLz')

print(l)

print("===========================")

l = re.findall('\W', 'abc de@#$KLz')

print(l)

print("============================")

l = re.findall('\d', 'abc de34@#$KLz')

print(l)

print("============================")

l = re.findall('[a-z]', 'abc de@#$KLz')

print(l)

print("============================")

Example 6:

* **finditer()**
* **sub()🡪 substitution or replacement. Search for the string and replace if found, else no change will be made to the target-string**

import re

'''

re.sub('regex','replacementvalue','target-string')

'''

#replace any number with '#' in the target string

s=re.sub('\d','#','a7b9K5@# 431Rtrw&\*t')

print(s)

Example 7:

* **subn()🡪 just like ‘sub()’, but also tells the no of replacements made. Return type is a ‘tuple’.**
  + **First value in the tuple is the replaced string and**
  + **Second value in the tuple is an int which tells the no of replacements.**

import re

t = re.subn(r'\d','xxxx','a7b9K5t9K')

print(type(t))

print('The result String: ', t[0])

print('The no of replacements',t[1])

Example 8:

* **split()🡪 splits the target-string to given delimeter**

import re

l = re.split('-','1-2-3-4-5-6-7-8')

print(l)

import re

l = re.split('.', 'w@ww.first.man.computers')

for x in l:

    print(x)

# result for the above prog is empty white spaces

# because, all the characters are considered when we use '.'

print("====================")

# solution for the above problem is '\.'

l = re.split('\.', 'w@ww.first.man.computers')

for x in l:

    print(x)

# or we can also use '[.]' also

print("====================")

l = re.split('[.]', 'w@ww.first.man.computers')

for x in l:

    print(x)

Example 9:

NOTE:

^ 🡪means starts with

$ 🡪means ends with

import re

# To check weather the target string starts with the given string or not

# Just like match() function

# If match found at starting returns object, else returns None

m = re.search('^Telugu', 'Telugu style: My state is Telugu desam and my mother tongue is Telugu.')

print(m)

print("==================================")

# one more example

s = 'Learning python is very easy...'

result = re.search('^Learn',s)

if result != None:

    print('Target string starts with Learn')

else:

    print('Target string does not starts with Learn')

Example 10:

import re

# To check weather the target string ends with the given string or not

# If match found at starting returns object, else returns None

m = re.search('Telugu$', 'Telugu style: My state is Telugu desam and my mother tongue is Telugu')

print(m)

print("==================================")

Example 11:

# one more example

s = 'Learning python is very easy'

result = re.search('easy$',s)

if result != None:

    print('Target string ends with easy')

else:

    print('Target string doesnot ends with easy')

Example 12:

import re

# one more example to ignore the case

s = 'Learning python is very EASy'

result = re.search('easy$',s, re.IGNORECASE)

if result != None:

    print('Target string ends with easy')

else:

    print('Target string doesnot ends with easy')

**Practice Programs:**

1. Write a RE which satisfies the following rules
   1. Allowed characters are alphabets, digits and #.
   2. First character should be lower case alphabet symbol from a to k.
   3. Second character should be any digit divisible by 3.
   4. Length of identifier should be at least 2.
   5. [a-k][0369][a-zA-Z0-9#]\*

'''

a.  Allowed characters are alphabets, digits and #.

b.  First character should be lower case alphabet symbol from a to k.

c.  Second character should be any digit divisible by 3.

d.  Length of identifier should be at least 2.

'''

import re

s = input("Enter a value to validate: ")

m = re.fullmatch('[a-k][0369][a-zA-Z0-9#]\*', s)

if m != None:

print(s, 'is valid Value...')

else:

print(s, 'is not valid...')

import re

# RE to represent all 10 digit mobile no's

i = input("Enter your mobile no: ")

m = re.fullmatch('[6789][0-9]{9}', i)

if m != None:

print(i, 'is valid Mobile Number')

else:

print(i, 'is not valid Mobile Number')

# other possible combinations are

# [6789][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]

# [6789][0-9]{9}

# [6-9]\d{9}

'''

write a RE for 10 or 11 or 12 or 13 digit mobile no:

1. For 10 digit: 6-9, 9 digits 🡪 [0][6-9]\d{9}

2. For 11 digit: First digit should be 0

3. For 12 digit: First 2 digits should be 91

4. For 13 digit: First 3 digits should be +91

'''

'''

Write a RE to get the mobile no's from a text file and copy to another text file

'''

import re

# Write a RE to extract all mobile no's from a text file:

f1 = open(r'C:\\Users\sanpe\AppData\Local\Programs\Python\Python36-32\PythonTestProgms\data.txt', 'r')

f2 = open(r'C:\\Users\sanpe\AppData\Local\Programs\Python\Python36-32\PythonTestProgms\output.txt', 'w')

for line in f1:

result = re.findall('[6-9]\d{9}', line)

fornumin result:

f2.write(num + '\n')

print("All mobile no's are extracted to output.txt file...")

f1.close()

f2.close()