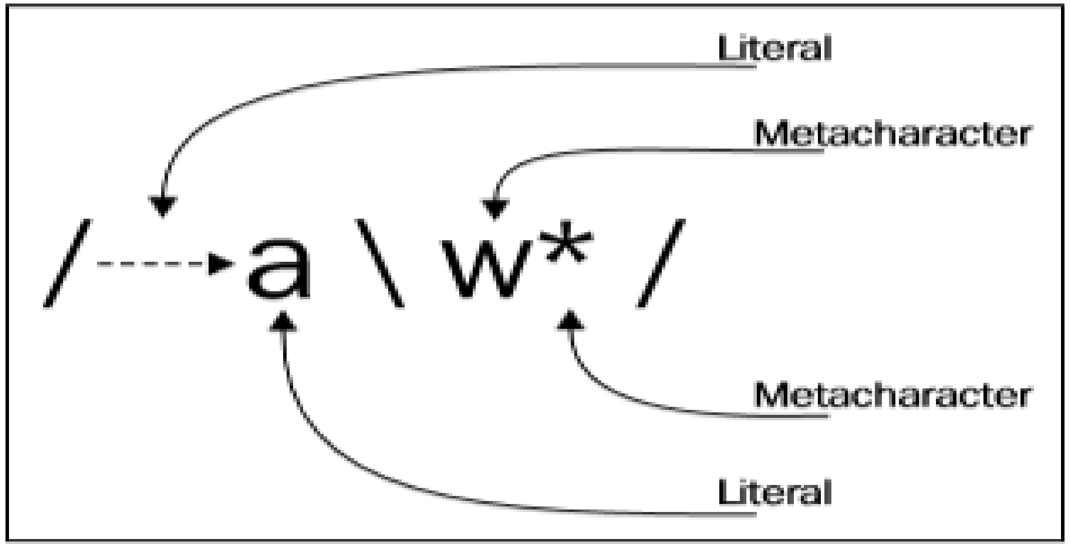
Regular Expressions in Python

Intro

- Python provides support for regular expressions via *re module*.
- Regular expressions are a powerful and standardized way of searching, replacing, and parsing text with complex patterns of characters. There are several good places to look:
- http://docs.python.org/3.7/library/re.html
- http://www.regular-expressions.info/
- https://developers.google.com/edu/python/regular-expressions
- Stack Overflow

RE Module

- Python Regular Expression Module is RE
- It supports Literals and Special (Meta Characters)



- ➢Regular expressions use two types of characters:

  a) Meta characters: As the name suggests, these characters have a special meaning,

  b) Literals

In Python, we have module “re” that helps with regular expressions. So you need to import library re before you can use regular expressions in Python.

- ➢Use this code --> Import re
- The most common uses of regular expressions are:

  - ❑ Search a string (search and match)
  - ❑ Finding a string (findall)
  - ❑ Break string into a sub strings (split)
  - ❑ Replace part of a string (sub)
  - ❑ Let’s look at the methods that library “re” provides to perform these tasks.

Meta Characters

- In regular expressions, there are twelve met characters that should be escaped if they are to be used with their literal meaning:
  - Backslash \
  - Caret ^
  - Dollar sign $
  - Dot .
  - Pipe symbol |
  - Question mark ?
  - Asterisk *
  - Plus sign +
  - Opening parenthesis (
  - Closing parenthesis)
  - Opening square bracket [
  - The opening curly brace {

Operators	Description
.	Matches with any single character except newline ‘\n’.
?	match 0 or 1 occurrence of the pattern to its left
+	1 or more occurrences of the pattern to its left
*	0 or more occurrences of the pattern to its left
\w	Matches with a alphanumeric character whereas \W (upper case W) matches non alphanumeric character.
\d	Matches with digits [0-9] and /D (upper case D) matches with non-digits.
\s	Matches with a single white space character (space, newline, return, tab, form) and \S (upper case S) matches any non-white space character.
\b	boundary between word and non-word and /B is opposite of /b
[..]	Matches any single character in a square bracket and [^..] matches any single character not in square bracket
\	It is used for special meaning characters like \. to match a period or \+ for plus sign.
^ and $	^ and $ match the start or end of the string respectively
{n,m}	Matches at least n and at most m occurrences of preceding expression if we write it as {,m} then it will return at least any minimum occurrence to max m preceding expression.
a| b	Matches either a or b
( )	Groups regular expressions and returns matched text
\t, \n, \r	Matches tab, newline, return

Problem 1: Return the first word of a given string

Solution 1

By using '.'

```
>>> import re
>>> result=re.findall(r'.','Python is largest Programming Language used in world')
>>> print(result)

['P', 'y', 't', 'h', 'o', 'n', ' ', 'i', 's', ' ', 'l', 'a', 'r', 'g', 'e', 's', 't', ' ', 'P', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g', ' ', 'L', 'a', 'n', 'g', 'u', 'a', 'g', 'e', ' ', 'u', 's', 'e', 'd', ' ', 'i', 'n', ' ', 'w', 'o', 'r', 'l', 'd']
>>>
```

Above Space also Extracted

```
>>> result=re.findall(r'\w','Python is largest Programming Language used in world')

>>> print(result)

['P', 'y', 't', 'h', 'o', 'n', 'i', 's', 'l', 'a', 'r', 'g', 'e', 's', 't', 'P', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g', 'L', 'a', 'n', 'g', 'u', 'a', 'g', 'e', 'u', 's', 'e', 'd', 'i', 'n', 'w', 'o', 'r', 'l', 'd']
>>>
```

Solution-2 Extract each word **(**using “*****” or “+“**)**

```
>>> result=re.findall(r'\w*','Python is largest Programming Language used in world')

>>> print(result)

['Python', '', 'is', '', 'largest', '', 'Programming', '', 'Language', '', 'used', '', 'in', '', 'world', '']
```

- Again, it is returning space as a word because “*****” returns zero or more matches of pattern to its left. Now to remove spaces we will go with “**+**“.

```
>>> result=re.findall(r'\w+','AV is largest Analytics community of India')
>>> result=re.findall(r'\w+','Python is largest Programming Language used in world')
>>> print(result)
['Python', 'is', 'largest', 'Programming', 'Language', 'used', 'in', 'world']
>>>
```

Solution-3 Extract each word **(**using “^“**)**

```
>>> result=re.findall(r'^\w+','Python is largest Programming Language used in world')
 >>> print(result)

['Python']
>>>
```

- If we will use “$” instead of “^”, it will return the word from the end of the string. Let’s look at it.

```
>>> result=re.findall(r'\w+$','Python is largest Programming Language used in world')

>>> result

['world']
```

-

Problem 2: Return the first two character of each word

Solution-1 Extract consecutive two characters of each word, excluding spaces (using “\w“)

```
>>> result=re.findall(r'\w\w','Python is Best
Programming Language')
>>> print(result)
['Py', 'th', 'on', 'is', 'Be', 'st', 'Pr', 'og', 'ra', 'mm', 'in', 'La',
'ng', 'ua', 'ge']
>>>
```

Solution-2 Extract consecutive two characters those available at start of word boundary (using “\b“)

```
>>> result=re.findall(r’\b\w','Python is Best Programming Language')
>>> print(result)
['P', 'i', 'B', 'P', 'L']
>>>
```

Problem 3: Return the domain type of given email-ids

Solution-1 Extract all characters after “@”

```
>>> result=re.findall(r'@\w+','abc.test@yahoo.com, xyz@indiaqoinfotech.in, test.first@python.org, first.test@rest.biz')
>>> print(result)
['@yahoo', '@indiaqoinfotech', '@python', '@rest']
>>>
```

- Above, you can see that “.com”, “.in” part is not extracted. To add it, we will go with below code.

```
>>> result=re.findall(r'@\w+.\w+','abc.test@yahoo.com, xyz@indiaqoinfotech.in, test.first@python.org, first.test@rest.biz')
>>> print(result)
['@yahoo.com', '@indiaqoinfotech.in', '@python.org', '@rest.biz']
>>>
```

- Solution – 2 Extract only domain name using “( )”

```
>>> result=re.findall(r'@(\w+.(\w+)','abc.test@yahoo.com,
xyz@indiaqoinfotech.in, test.first@python.org, first.test@rest.biz')
>>> print(result)
['com', 'in', 'org', 'biz']
>>>
```

Problem 4: Return date from given string

- Here we will use “**\d**” to extract digit.

```
>>> result=re.findall(r'\d{2}-\d{2}-\d{4}','Amit 34-3456 12-05-2007,
XYZ 56-4532 11-11-2011, ABC 67-8945 12-01-2009')
>>> print(result)
['12-05-2007', '11-11-2011', '12-01-2009']
>>>
```

Full Script Example

```
import re
list = ["camp get", "camp give", "camp Selenium"]
for element in list:
    z = re.match("(g\w+)\W(g\w+)", element)
    if z:
        print(z.groups())
patterns = [ 'software testing', 'camp99' ]
text = 'software testing is fun?'
for pattern in patterns:
    print ('Looking for "%s" in "%s" ->' % (pattern, text))
    if re.search(pattern,  text):
        print ('found a match!')
    else:
            print ('no match')
abc='camp@google.com, camp@hotmail.com, users@yahoomail.com'
emails = re.findall(r'[\w\.-]+@[\w\.-]+', abc)
for email in emails:
    print(email)
```

ExaMple 2:

- For files, you may be in the habit of writing a loop to iterate over the lines of the file, and you could then call findall() on each line. Instead, let findall() do the iteration for you -- much better! Just feed the whole file text into findall() and let it return a list of all the matches in a single step (recall that f.read() returns the whole text of a file in a single string):

- # Open file
- f = open('test.txt', 'r')
- # Feed the file text into findall(); it returns a list of all the found strings
- strings = re.findall(r'some pattern', f.read())

EX 3: Findall and groups

- The parenthesis ( ) group mechanism can be combined with findall(). If the pattern includes 2 or more parenthesis groups, then instead of returning a list of strings, findall() returns a list of *tuples*. Each tuple represents one match of the pattern, and inside the tuple is the group(1), group(2) .. data. So if 2 parenthesis groups are added to the email pattern, then findall() returns a list of tuples, each length 2 containing the username and host, e.g. ('alice', 'google.com').

```
•  str = 'purple alice@google.com, blah monkey bob@abc.com blah dishwasher'
•  tuples = re.findall(r'([\w\.-]+)@([\w\.-]+)', str)
•  print tuples  ## [('alice', 'google.com'), ('bob', 'abc.com')]
•  for tuple in tuples:
•    print tuple[0]  ## username
•    print tuple[1]  ## host
```

# .   Any character except for a new line

```
>>> text='indiaqoinfotech.com'
>>> print(re.findall('.', text))
['i', 'n', 'd', 'i', 'a', 'q', 'o', 'i', 'n', 'f', 'o', 't', 'e', 'c', 'h', '.', 'c', 'o', 'm']
>>> print(re.findall('...', text))
['ind', 'iaq', 'oin', 'fot', 'ech', '.co']
>>>
```

Period

- text = ‘indiaqoinfotech.com'
- print(re.findall('\.', text)) # matches a period
- print(re.findall('[^\.]', text)) # matches anything but a period
- #> ['.']
- #> ['i', 'n', 'd', 'i', 'a', 'q', 'o', 'i', 'n', 'f', 'o', 't', 'e', 'c', 'h', 'c', 'o', 'm']
- text='www.indiaqoinfotech.com'
- >>> print(re.findall('\.', text))
- ['.', '.']
- >>>

Any digit

- text = '01, Jan 2015'
- print(re.findall('\d+', text)) # \d Any digit. The + mandates at least 1 digit.
- #> ['01', '2015']

Anything but a digit

- text = '01, Jan 2015'
- print(re.findall('\D+', text)) # \D Anything but a digit
- #> [', Jan ']

Any character including digit

- text = '01, Jan 2015'
- print(re.findall('\w+', text)) # \w Any character
- #> ['01', 'Jan', '2015']

Anything but a character

- text = '01, Jan 2015'
- print(re.findall('\W+', text)) # \W Anything but a character
- #> [', ', ' ']

Collection of characters

- text = '01, Jan 2015'
- print(re.findall('[a-zA-Z]+', text)) # [] Matches any character inside
- #> ['Jan']

Match something upto ‘n’ times

- text = '01, Jan 2015'
- print(re.findall('\d{4}', text)) # {n} Matches repeat n times.
- print(re.findall('\d{2,4}', text))
- #> ['2015']
- #> ['01', '2015']

Match 1 or more occurrences

- print(re.findall(r'Co+l', 'So Cooool')) # Match for 1 or more occurrences
- #> ['Cooool']

Match any number of occurrences (0 or more times)

- print(re.findall(r'Pi*lani', 'Pilani'))
- #> ['Pilani']

Match exactly zero or one occurrence

- print(re.findall(r'colou?r', 'color'))
- ['color']

Exercise Question

- 1. Extract the user id, domain name and suffix from the following email addresses.
- emails = """zuck26@facebook.com
- page33@google.com
- jeff42@amazon.com"""
- desired_output = [('zuck26', 'facebook', 'com'),
- ('page33', 'google', 'com'),
- ('jeff42', 'amazon', 'com')]

Solution

- # Solution
- pattern = r'(\w+)@([A-Z0-9]+)\.([A-Z]{2,4})'
- re.findall(pattern, emails, flags=re.IGNORECASE)
- #> [('zuck26', 'facebook', 'com'),
- #> ('page33', 'google', 'com'),
- #> ('jeff42', 'amazon', 'com')]
- # There are more sophisticated patterns for matching the email domain and suffix. This is just one version of the answer.

Exercise 2

- Retrieve all the words starting with ‘b’ or ‘B’ from the following text.
- text = """Betty bought a bit of butter, But the butter was so bitter, So she bought some better butter, To make the bitter butter better."""

Solution

- import re
- re.findall(r'\bB\w+', text, flags=re.IGNORECASE)
- #> ['Betty', 'bought', 'bit', 'butter', 'But', 'butter', 'bitter', 'bought', 'better', 'butter', 'bitter', 'butter', 'better']

- # '\b' mandates the left of 'B' is a word boundary, effectively requiring the word to start with 'B'.
- # Setting 'flags' arg to 're.IGNORECASE' makes the pattern case insensitive.

Exercise 3:

- Split the following irregular sentence into words
- sentence = """A, very   very; irregular_sentence"""
- desired_output = "A very very irregular sentence"

Solution

- import re
- " ".join(re.split('[;,\s_]+', sentence))
- #> 'A very very irregular sentence'

- # Add more delimiters into the pattern as needed.

Exercise 4

- Clean up the following tweet so that it contains only the user’s message. That is, remove all URLs, hashtags, mentions, punctuations, RTs and CCs.
- tweet = '''Good advice! RT @TheNextWeb: What I would do differently if I was learning to code today http://t.co/lbwej0pxOd cc: @garybernhardt #rstats'''
- desired_output = 'Good advice What I would do differently if I was learning to code today'

solution

```
• import re
• def clean_tweet(tweet):
•     tweet = re.sub('http\S+\s*', '', tweet)  # remove URLs
•     tweet = re.sub('RT|cc', '', tweet)  # remove RT and cc
•     tweet = re.sub('#\S+', '', tweet)  # remove hashtags
•     tweet = re.sub('@\S+', '', tweet)  # remove mentions
•     tweet = re.sub('[%s]' % re.escape(""""!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~""""), '', tweet)  # remove
  punctuations
•     tweet = re.sub('\s+', ' ', tweet)  # remove extra whitespace
•     return tweet

• print(clean_tweet(tweet))
• #> Good advice What I would do differently if I was learning to code today
```

Exercise 5

- Extract all the text portions between the tags from the following HTML page: https://raw.githubusercontent.com/selva86/datasets/master/sample.html
- Code to retrieve the HTML page:
- import requests
- r = requests.get("https://raw.githubusercontent.com/selva86/datasets/master/sample.html")
- r.text # html text is contained here
- desired_output = ['Your Title Here', 'Link Name', 'This is a Header', 'This is a Medium Header', 'This is a new paragraph! ', 'This is a another paragraph!', 'This is a new sentence without a paragraph break, in bold italics.']

solution

- # Note: remove the space after < and /.*> for the pattern to work
- re.findall('<.*?>(.*)< /.*?>', r.text)
- #> ['Your Title Here', 'Link Name', 'This is a Header', 'This is a Medium Header', 'This is a new paragraph! ', 'This is a another paragraph!', 'This is a new sentence without a paragraph break, in bold italics.']

Greedy Matching

- >>> import re
- >>> text = "< body>Regex Greedy Matching Example < /body>"

- >>> re.findall('<.*>', text)
- ['< body>Regex Greedy Matching Example < /body>']
- >>> re.findall('<.*?>', text)
- ['< body>', '< /body>']
- >>>

Modifiers

- $ ->   End of string
- ^ -> Start of string
- ab|cd   -> Matches ab or de.
- [ab-d] -> One character of: a, b, c, d
- [^ab-d] -> One character except: a, b, c, d
- ()   -> Items within parenthesis are retrieved
- (a(bc)) -> Items within the sub-parenthesis are retrieved

Repetition

- [ab]{2} Exactly 2 continuous occurrences of a or b
- [ab]{2,5} 2 to 5 continuous occurrences of a or b
- [ab]{2,} 2 or more continuous occurrences of a or b
- + One or more *
- Zero or more
- ? 0 or 1