**REGULAR EXPRESSION NOTES**

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.

RegEx can be used to check if a string contains the specified search pattern.

The re module offers a set of functions that allows us to search a string for a match:

Metacharacters are characters with a special meaning.

Ex:

[]

\

. -> matches any character

^

$

\*

+

? -> 1 or 0 occurrences of preceding characters (? Occurs after a character) OR to make something non-greedy (? Occurs after + or \*)

{}

|

() is used in grouping/capturing

Raw string literals are **string literals** that are designed to make it easier to include nested characters like quotation marks and backslashes that normally have meanings as delimiters and escape sequence starts.

Special Sequences: \ followed by one of the characters in the list below, and has a special meaning:

Ex:

\A

\b: matches only at word boundary (doesn't actually match any character, just sets the rule). (Word is a sequence of alphanmeric characters plus underscore.)

\B

\d: [0-9]

\D: [^0-9]

\s: matches any whitespace character

\S: matches any non-whitespace character

\w: alphanumeric character [a-zA-Z0-9\_]

\W:  [^a-zA-Z0-9\_]

\Z

These sequences can be included inside a character class ([]). For example, [\s,.] is a character class that will match any whitespace character, or ',' or '.'. Inside character sets all special characters except \ and ^ when used at the starting lose there meaning.

\ -- inhibit the "specialness" of a character. So, for example, use \. to match a period or \\ to match a slash. If you are unsure if a character has special meaning, such as '@', you can put a slash in front of it, \@, to make sure it is treated just as a character.

Different nuances of sets:

[arn]

[a-n]

[^arn] -> Matching characters except ‘a’, ‘r’, and ‘n’

[1023]

[0-9]

[0-5][0-9]

[a-zA-Z]

[+]

In sets, +, \*, ., |, (), $,{} has no special meaning, so [+] means: return a match for any +character in the string. Metacharacter ‘.’ Does not match a \n.

Functions

The **findall()**function returns a list containing all matches.

Finditer()

The search() function searches the string for a match, and returns a [Match object](https://www.w3schools.com/python/python_regex.asp#matchobject) if there is a match. If there is more than one match, only the first occurrence of the match will be returned. (Match Object).start() -> Index of the first occurrence.

The split() function returns a list where the string has been split at each match. You can also specify the exact number of occurrences where you want to do the split.

The sub() function replaces the matches with the text of your choice. You can also specify the number of replacements. You can use back referencing here.

Match()-> matches from beginning of the string

A Match Object is an object containing information about the search and the result. If there is no match, the value None will be returned, instead of the Match Object. The Match object has properties and methods used to retrieve information about the search, and the result.

.span() returns a tuple containing the start-, and end positions of the match.  
.string returns the string passed into the function  
.group() returns the part of the string where there was a match

.groups()

.groupdict()

start()

end()

<https://docs.python.org/3/howto/regex.html?highlight=regular%20expressions>

Repetitions such as \* are *greedy*; when repeating a RE, the matching engine will try to repeat it as many times as possible. If later portions of the pattern don’t match, the matching engine will then back up and try again with fewer repetitions.

The question mark character, ?, matches either once or zero times; you can think of it as marking something as being optional. For example, home-?brew matches either 'homebrew' or 'home-brew'.

The most complicated repeated qualifier is {m,n}, where *m* and *n* are decimal integers. This qualifier means there must be at least *m* repetitions, and at most *n*. For example, a/{1,3}b will match 'a/b', 'a//b', and 'a///b'. It won’t match 'ab', which has no slashes, or 'a////b', which has four.

You can omit either *m* or *n*; in that case, a reasonable value is assumed for the missing value. Omitting *m* is interpreted as a lower limit of 0, while omitting *n* results in an upper bound of infinity.

Regular expressions are compiled into pattern objects, which have methods for various operations such as searching for pattern matches or performing string substitutions.

p = re.compile('ab\*')

Understanding the backslash plague:

print("\section")

\section

print("\\section")

\section

print("\\\section")

\\section

print("\\\\section")

\\section

Raw String Notation for regular expressions:

r"\n" is a two-character string containing '\' and 'n', while "\n" is a one-character string containing a newline.

Precompiling a regular expression

Regular expressions can be written in a way where different components could fit into different lines and comments could be added. Go through October 5th notes.

Naming captures

Suppressing captures

Back referencing captured strings

Processing datasets using RE’s

print(','.join(values))

values: list

Above returns a string