RegEx - Regular Expressions in Python.

A regular expression, regex or regexp (sometimes called a rational expression) is a sequence of characters that define a search pattern. Usually this pattern is used by string searching algorithms for "find" or "find and replace" operations on strings, or for input validation. It is a technique that developed in theoretical computer science and formal language theory.

To learn more about RegEx please follow this link : https://docs.python.org/3/howto/regex.html

Now lets see how to use regex functions in Python code.

In [1]:

```
#Python has a built-in package called re, which can be used to work with Regular Expressions.
#Import the re module:

import re
```

In [10]:

```
#Let us start with an example
txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
print(x.string == txt)
```

```
True
```

RegEx Functions

The Search Function

The Substitue Function

The Sprilt Function

The Find All Fucntion

The Search Function

This function searches for first occurrence of RE pattern within string with optional flags.

Here is the syntax for this function −

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

Here is the description of the parameters −
pattern- This is the regular expression to be matched.
string - This is the string, which would be searched to match the pattern anywhere in the string.
flags - You can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.

In [12]:

```
#Example


string = "The rain in Spain"
x = re.search("\s", string)

print("The first white-space character is located in position:", x.start())
```

```
The first white-space character is located in position: 3
```

Great now we know how to search in a string. The question is what is this "\s" ??? Is you answer pattern? if yes then give a pat on your back.

What else can be a Pattern ?

Meta Characters

Metacharacters are characters with a special meaning:

```
Character   Description
    []  A set of characters "[a-m]"
    \   Signals a special sequence (can also be used to escape special characters)  "\d"
    .   Any character (except newline character)    "he..o"
    ^   Starts with "^hello"
    *   Zero or more occurrences    "aix*"
    +   One or more occurrences "aix+"
    {}  Excactly the specified number of occurrences    "al{2}"
    |   Either or   "falls|stays"
    ()  Capture and group
    $   Ends with   "world$"
```

Special Sequences

A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

```
Character   Description

    \A  Returns a match if the specified characters are at the beginning of the string
"\AThe"
    \b  Returns a match where the specified characters are at the beginning or at the end
of a word                    r"\bain"
        r"ain\b"
    \B  Returns a match where the specified characters are present, but NOT at the
beginning (or at the                end) of a word  r"\Bain"
        r"ain\B"
    \d  Returns a match where the string contains digits (numbers from 0-9) "\d"
    \D  Returns a match where the string DOES NOT contain digits    "\D"
    \s  Returns a match where the string contains a white space character   "\s"
    \S  Returns a match where the string DOES NOT contain a white space character   "\S"
    \w  Returns a match where the string contains any word characters (characters from a to
Z, digits from             0-9, and the underscore _ character) "\w"
    \W  Returns a match where the string DOES NOT contain any word characters   "\W"
    \Z  Returns a match if the specified characters are at the end of the string
"Machine\Z"
    .   Find a single character, except newline or line terminator
    \0  Find a NUL character
    \n  Find a new line character
    \f  Find a form feed character
    \r  Find a carriage return character
    \t  Find a tab character
    \v  Find a vertical tab character
    \xxx    Find the character specified by an octal number xxx
    \xdd    Find the character specified by a hexadecimal number dd
    \uxxxx  Find the Unicode character specified by a hexadecimal number xxxx
```

Sets

A set is a set of characters inside a pair of square brackets [] with a special meaning:

```
Set     Description
[arn]   Returns a match where one of the specified characters (a, r, or n) are present
[a-n]   Returns a match for any lower case character, alphabetically between a and n
[^arn]  Returns a match for any character EXCEPT a, r, and n
```

```
[0123]  Returns a match where any of the specified digits (0, 1, 2, or 3) are present
[0-9]   Returns a match for any digit between 0 and 9
[0-5][0-9]  Returns a match for any two-digit numbers from 00 and 59
[a-zA-Z]    Returns a match for any character alphabetically between a and z, lower case OR
upper case
 [+]    In sets, +, *, ., |, (), $,{} has no special meaning, so [+] means: return a match
for any +                   character in the string
```

For More details on Regular expressions please go though the following link : https://docs.python.org/3/howto/regex.html

Compiling Regular Expressions

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

In [19]:

```
compiler = re.compile('[a-z]+')
compiler
```

Out[19]:

```
re.compile(r'[a-z]+', re.UNICODE)
```

compile function also accepts an optional flags argument,used to enable various special features and syntax variations.

In [20]:

```
compiler = re.compile('[a-z]+',re.IGNORECASE)
compiler
```

Out[20]:

```
re.compile(r'[a-z]+', re.IGNORECASE|re.UNICODE)
```

In [24]:

```
print(compiler.match('Schoolofai'))
```

```
<_sre.SRE_Match object; span=(0, 10), match='Schoolofai'>
```

The Match Function

Once you have an object representing a compiled regular expression, what do you do with it? Pattern objects have several methods and attributes. Only the most significant ones will be covered here; consult the re docs for a complete listing.

Here is the syntax for this function −

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

Here is the description of the parameters −

pattern This is the regular expression to be matched.

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

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

In [32]:

```
line = "Lets teach the machines to learn"

matchObject = re.match( r'(.*) the (.*?) .*', line, re.M|re.I)
```

```
if matchObject:
   print ("matchObj.group() : ", matchObject.group())
   print ("matchObj.group(1) : ", matchObject.group(1))
   print ("matchObj.group(2) : ", matchObject.group(2))
else:
   print ("No match!!")
```

```
matchObj.group() :  Lets teach the machines to learn
matchObj.group(1) :  Lets teach
matchObj.group(2) :  machines
```

The Search Function

The search() function searches the string for a match, and returns a Match object if there is a match.

If there is more than one match, only the first occurrence of the match will be returned.

In [35]:

```
string = "The School of AI"
x = re.search("\s", string)

print("The first white-space character is located in position:", x.start())
```

```
The first white-space character is located in position: 3
```

Match Vs Search

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 [38]:

```
line = "Cats are smarter than dogs";

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

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

```
match --> No match!!
search --> searchObj.group() :  dogs
```

The split() Function

The split() function returns a list where the string has been split at each match

In [39]:

```
string = "Let's make the machine learn"
split = re.split("\s", string)
print(split)
```

```
["Let's", 'make', 'the', 'machine', 'learn']
```

The substitute sub() Function

The sub() function replaces the matches with the text of your choice

In [40]:

```
string = "Let's make the machine learn"
substitute = re.sub("\s", "_", string)
print(substitute)
```

```
Let's_make_the_machine_learn
```

The findall() Function

The findall() function returns a list containing all matches

In [43]:

```
string = "Let's make the machine learn, the school of ai"
alloccurances = re.findall("the", string)
print(alloccurances)
```

```
['the', 'the']
```

Hope you guys got a hold of regular expressions. For more deeper fucntions follow this link https://docs.python.org/3/library/re.html