Regular Expressions

What is Regular Expression

A **Re**gular **Ex**pression (RegEx) is a sequence of characters that defines a search pattern. For example,

^a.....s$

The above code defines a RegEx pattern. The pattern is: **any eight letter string starting** with a and ending with s.

Python has a module named **re** to work with regular expression. Here's an example:

```
import re

pattern = '^a...s$'
test_string = 'abyss'
Result = re.match(pattern, test_string)

if result:
    print("Search successful.")
else:
    print("Search unsuccessful")
```

Here, we used re.match function to grab pattern within the test_string. The method returns a match object if the search is successful. If not, it returns **No**

Specify Pattern Using RegEx:

To specify regular expressions, metacharacters are used. In the above example, ^ and meta characters.

Meta characters:

Metacharacters are characters that are interpreted specially by a RegEx engine.
Here's a list of metacharacters: []. ^ $ *

[] – Square brackets:

Square brackets specify a set of characters you wish to match.

Expression	String	Matched?
[abc]	a	1 match
	ac	2 matches
	Hey Jude	No match
	abc de ca	5 matches

Above, [abc] will match if the string you are trying to match contains any of the a, b or c.

You can also specify a range of chara ters using – inside square brackets.

- [a-e] is the same as [abcde].
- [1-4] is the same as [1234].
- [0-39] is the same as [01239].

You can complement (invert) the character set by using caret ^ symbol at the start of a square-bracket.

- [^abc] means any character except a or b or c.
- [^0-9] means any non-digit character.

– Period:

period matches any single character (except newline '\n').

Expression	String	Matched?
. .	a	No match
	ac	1 match
	acd	1 match
	acde	2 matches (contains 4 characters)

^ – Caret:

The caret symbol ^ is used to check if a string starts with a certain character.

Expression	String	Matched?
^a	a	1 match
	abc	1 match
	bac	No match
^ab	abc	1 match
	acb	No match (starts with `a` but not followed by `b`)

$ – Dollar:

The dollar symbol $ is used to check if a string ends with a certain character.

Expression	String	Matched?
a$	a	1 match
	formula	1 match
	cab	No match

+ – Plus:

The plus symbol + matches one or more occurrences of the pattern left to it.

Expression	String	Matched?
ma+n	mn	No match (no a character)
	man	1 match
	maaan	1 match
	main	No match (a is not followed by n)
	woman	1 match

Python RegEx

Python has a module named **re** to work with regular expressions. To use it, we need to import the module.

```
import re
```

The module defines several functions and constants to work with RegEx.

re.findall()

The re.findall() method returns a list of strings containing all matches

Example:

```
#program to extract numbers from a string

string = 'hello 12 hi 89. Howdy 34'
pattern = '\d+’ #[0-9]+
result = re.findall(pattern, string)
print(result)
```

If the pattern is no found, re.findall() returns an empty list.

re.split()

The re.split method splits the string where there is a match and returns a list of strings where the splits have occurred.

Example:

```
import re

string = 'Twelve:12 Eighty nine:89'
pattern = '\d+'
result = re.split(pattern, string)
print(result)
```

If the pattern is no found, re.split() returns a list containing an empty string.

re.sub()

The syntax of re.sub() is:

```
re.sub(pattern,  replace,  string)
```

Example:

```
#program to remove all white spaces

import re
string = 'abc 12\ de  23  \n  f4    6'

#matches all whitespace characters
pattern = '\s+' #'

#empty string
replace = ''

new_string = re.sub(pattern, replace, string)
```

If the pattern is no found, re.sub() retur n s the original string.

re.sub()

The re.subn() is similar to re.sub() expect it returns a tuple of 2 items containing the new string and the number of substitutions made.

Example:

```
#program to remove all whitespac  s
import re
#multiline string
string = 'abc  12\ de  23  \n  f45  6'

# matches all whitespace characters
pattern = '\s+'

#empty string
replace = ' '
new_string = re.subn(pattern, replace, string)
print(new_string)

#Output: ( 'abc12de23f456',  4 )
```

re.search()

The re.search() method takes two arguments: a pattern and a string. The method looks for the first location where the RegEx pattern produces a match with the string. If the search is successful, re.search() returns a match object; if not, it returns None.

```
match = re.search(pattern, str)
```

abababababa

Example:

```
import re
String = “Python is fun”

#check if ‘Python’ is at the beginning
match = re.search(‘\APython’, string)

if match:
    print(“pattern found inside the string”)
else:
     print(“pattern not found”)

#Output :  pattern found inside the string
```

Here, match contains a match object.

Math Object

You can get methods and attributes of a match object using dir() function.
Some of the commonly used methods and attributes of match objects are:

```
Import re
String = ‘ 39801  356, 2102  1111 ’

#Three digit number followed by   pace followed by two digit number
Pattern = ‘ (\d{3}) (\d{2}) (\d{5}) ’

#match variable contains a Match object.
Match = re.search(pattern, string)

If match:
    print(match.group())
Else:
    print(“ pattern not found ”)
# Output:  801  35
```

Here, match variable contains a match object.
our pattern (\d{3}) (\d{2}) has two subgroups (\d{3}) and (\d{2}). You can get th art of the string of these parenthesized subgroups. Here's how:

```
>>> match.group(1)
'801'

>>> match.group(2)
'35'

>>> match.group(3)
('801', '35')

>>> match.group()
('801', '35')
```

match.start(), match.end() and match.span()

The start() function returns the index of the matched substring. Similarly, end() returns the end index of the matched substring.

```
>>> match.start()
2
>>>match.end()
8
```

The span() function returns a tuple containing start and end index of the matched part.

```
>>> match.span()
(2, 8)
```

match.re and match.string

The re attribute of a matched object returns a regular expression object. Similarly, string attribute returns the passed string.

```
>>> match.re
Re.compile(‘(\\d{3}) (\\d{2})’

>>> match.string
‘ 39801 356, 2102 1111’
```