* Regular Expressions *

⊙ Regular Expressions (RegEx):-

↳ RegEx : Special sequence of char help to match/find other char / group of char using some Python syntax.

↳ RegEx widely used in UNIX world.

↳ Python provide re module, support us of regex in Python.

↳ Primary fun^n is to offer a search, where takes regular exp & a string.

↳ It either returns first math / else None.

↳ The re module raises the re.error excep^n when error occur while impl / using RegEx.

↳ Metacharacters: Many letters have particular meaning when utilized in a regular expression.

⊙ RegEx Functions :-

Functions	Description
re.compile(pattern, flags = 0)	Used to turn regular pattern into an obj of a Regular exp, that may be used in no. of ways for matching patterns in a string.
re.search (pattern, string, flags = 0)	Find first occurrence of a regex pattern in given string.
re.match (pattern, string, flags = 0)	Starts matching pattern at the beginning of the string.
re.fullmatch (pattern, string, flags = 0)	Used to match whole string with a regex pattern.
re.findall (pattern, strings, flags = 0)	Used to find all non-overlapping patterns in a string. It returns list of matched patterns.
re.finditer (pattern, string, flags = 0)	Return an iterator over all non-overlapping matches in the string. ↳ For each match, iterator returns Match object.

Function	Description
re.split(pattern, string, maxsplit = 0, flags = 0)	It is used to split the str, based on the RegEx pattern. ↳ Perform splitting operⁿ for maxsplit no. of times only.
re.sub(pattern, replace, string, count = 0, flags = 0)	Return the string obtained by replacing the leftmost matching occurrences of the 'pattern' in 'string' by the 'replace' string.
re.subn(pattern, replace, string, count = 0, flags = 0)	It works same as re.sub(), It returns a tuble as- (new_string, no_of_substituⁿ)
re.escape(pattern)	Escapes special char in string.
re.purge()	clears the regular exp cache.

search() vs match() in RegEx :-

- match() - Looks for match only where str starts
- search() - Looks for match everywhere in string

⊙ Regular Expression Modifiers: Flags:-

Flag	Description
re.I	Perform cas-insensitive matching.
re.L	Interprets words acc' to the current locale. It affects: ↳ the alphabetic group (\w & \W) ↳ world boundary behavior(\b & \B).
re.M	Makes $ match end of line. (not just end of string). ^ match start of any line (not just start of the string).
re.S	Makes a period(dot) match any char, including a newline.
re.U	Interprets letters acc' to Unicode char set. Affects behavior of \w, \W, \b, \B.
re.X	Permits 'cuter' RegEx syntax. It ignores whitespace (except inside a set [] / when escaped by backslash) & treats unescaped # as comment marker.

Metacharacters or Special Characters:-

Character	Description
[]	Defines set of characters.
{}	Matches exactly specified number of occurrences of a pattern.
|	Matches any of 2 defined patterns.
()	Enclose a group of RegEx.
\	Escape special character (shows next char is special seq).
^	Matches the beginning.
$	Matches the end.
.	Matches every char except newline.
+	One or more occurrences.
*	Zero or more occurrences.
?	Zero or one occur of pattern.

⊙ List of Special Sequences :-

↳ Special Sequence - '\' followed by single char alphabet.

Sequence	Description
\A	Match defined pattern at str start.
\b	Return match if char are at the beginning or at the end of a word.
\B	string should not start / end with the given pattern.
\d	Matches decimal digit, equivalent to [0-9].
\D	Matches non-digit char, equiv to [^0-9].
\s	Matches any whitespace characters. Equivalent to [\t \n \r \f \v].
\S	Matches non-space char. Equivalent to [^\t \n \r \f \v].
\w	Matches alphanumeric char. Equivalent to [a-zA-Z0-9_].
\W	Matches non-alphanumeric char.
\Z	Matches if string end with given RegEx.

Sets :-

↳ Set of chars inside pair of square brackets.

Set	Description
[arn]	Match one of specified char (a, r/n).
[a-n]	Single lower char bet^n a & n.
[^arn]	Char except a, r, n.
[0123]	Match 1 specified char of digits (0, 1, 2, 3).
[0-9]	Digit bet^n 0 & 9.
[0-5][0-9]	Two-digit number from 00 to 59.
[a-zA-Z]	Lower / Uppercase char alphabet between a & z.
[+]	In sets +, *, ., |, (), {}, $ has no special meaning. So, [+] matches '+' char in the string.

⊙ RegEx Functions from re Module :-

① re.compile() – used to create RegEx object, that can be used to match patterns in a string.

```
>>> s = 'I am snehal sanjay Mankar, from Saroud'
>>> obj = re.compile('[aeiouAEIOU]')
>>> res = obj.findall(s)
>>> print(res)
```

Can call any method using this object.

O/p → ['I', 'a', 'e', 'a', 'a', 'a', 'a', 'o', 'a', 'u']

② re.match() – Starts matching pattern from beginning of the string.

↳ Return match object if any match found with info like - start, end, span etc.

↳ Returns none, if no match is found.

- Example :

```
>>> s = 'I am Snehal Sanjay Mankar'
>>> r1 = re.match('snehal', s)
>>> print(r1)
```

O/p → None

```
>>> r2 = re.match('I', s)
>>> print(r2)
```

O/p → <re.Match object; span=(0,1), match='I'>

Returns Match object

③ re.search()–

↳ Look for 1st occur of RegEx seq & deliver it.

↳ If pattern matched, re.search() fun^n returns match object; otherwise, returns None.

```
>>> st = 'I am Snehal Mankar'
>>> print(re.search('Sanjay', st))
```

O/p → None.

```
>>> res = re.search('Snehal', s)
>>> print(res)
```

O/p → <re.Match object; span = (5, 11), match = 'Snehal'>

```
>>> res.span()   ⟶ (5, 11)
```

Return span of pattern match.

```
>>> res.start()  ⟶ 5
```

start index of match.

```
>>> res.end()    ⟶ 11
```

End index of match.

```
>>> res.endpos   ⟶ 18
```

End of the complete string.

```
>>> res.string   → I am Snehal Mankar
```

Return the complete string.

```
>>> res.re   ⟶ re.compile('Snehal')
```

Return pattern in object format.

```
>>> res.regs   ⟶ ((5, 11),)
```

④ re.fullmatch() – Match whole str with pattern.

↳ Returns corresponding match object.

↳ Return None, in case no match found.

```
>>> s = 'Snehal Mankar'
>>> res1 = re.fullmatch ('snehal', s)
>>> print (res1)
```

O/P → None

```
>>> res2 = re.fullmatch ('Snehal Mankar', s)
>>> print (res2)
```

O/P → <re.Match object; span = (0,13),
match = 'Snehal Mankar'>

⑤ re.findall() – Return a list of all non-overlapping matches in the string.

↳ If one / more capturing groups are present in the pattern, return list of grps; This will be list of tuples if pattern has more than 1 grp.

↳ Empty matches are included in the result.

```
>>> s = ' a aa aaa abaa aabaa'
>>> res = re.findall ('aa', s)
>>> print (res)
```

O/P → ['aa', 'aa', 'aa', 'aa', 'aa',]

⑥ re.finditer() – Returns an iterator that yields all non-overlapping matches of pattern in string.

↳ String scanned from Left to Right.

↳ Return matches in the order they discovered.

↳ For each match, iterator returns Match object.

```
>>> s = 'snehal snehal snehal'
>>> iter_ = re.finditer('snehal', s)
>>> print(iter_)
```

O/p → [<re.Match object; span=(0,6), match='snehal'>
<re.Match object; span=(7,13), match='snehal'>
<re.Match object; span=(14,20), match='snehal'>]

⑦ re.split() – Splits string by occur of pattern.

↳ If maxsplit=0, max splits occur.

↳ If maxsplit=1, split by first occur of patter & return remaining string as it is.

• Example:

```
>>> s = 'I am Snehal Mankar'
>>> print (re.split('\s', s))
```

Split by space.

O/p → ['I', 'am', 'Snehal', 'Mankar']

```
>>> print (re.split('\s', s, maxsplit=2))
```

O/p → ['I', 'am', 'Snehal Mankar']

(8) re.sub() – Substitute matching pattern with the 'replace' string.

```
re.sub(pattern, replace, string, count=0)
```

↳ count – No. of times to substitute.

- Example:

```
>>> s = 'I am Snehal Mankar'
>>> print(re.sub('\s', '##', s))
O/p → I##am##Snehal##Mankar
>>> print(re.sub('\s', '##', s, count=1))
O/p → I##am Snehal Mankar
```

↳ Substitute only 1 occur from left.

(9) re.subn() – works same as re.sub() fun^n.

↳ Returns a tuple – (new_str : str, substitu^n : int)

- Example:

```
>>> s = 'I am Snehal Mankar'
>>> print(re.subn('\s', '00', s))
O/p → ('I00am00Snehal00Mankar', 3)
```

Total 3 substitution

```
>>> print(re.subn('\s', '00', s, count=1))
O/p → ('I00am Snehal Mankar', 1)
```

Only 1 substitu^n

(10) re.escape()-

↳ Escapes special char in pattern.

↳ Becomes more IMP, when string contains RegEx metachars in it.

↳ Considers the metachar as regular chars.

```
>>> s1 = 'snehal11 snehal*'
>>> p1 = re.escape 'snehal*'
>>> print(re.search(p1, s1))
```

This is regex pattern, where * means & appears 0 or more times.

O/p → <re.Match object; span=(0,8), match='snehal11'>

```
>>> p = re.escape(p1)
>>> print(p1)
```

O/p → snehal* ← This is escape char, treats * as normal char, instead of its regular RegEx meaning.

```
>>> print(re.search(p, s1))
```

O/p → <re.Match object; span=(9,16), match='snehal*'>

(11) re.purge() - Simply clears RegEx cache.

```
>>> p = 'snehal*'
>>> print(re.search(p, 'snehal'))
```

The pattern first compiled & used once.

O/p → <re.Match object; span=(0,6), match='snehal'>

```
>>> re.purge() # Clear the regex cache.
>>> print(re.search(p, 'snehal'))
```

The pattern again compiled for using again.

O/p → <re.Match object; span=(0,6), match='snehal'>

↳ Compile with new regex object.