REGULAR EXPRESSION Concept :

- Regular Expressions are used to extract the required information from the given data by following patterns.
- Regular expressions are also used to check whether the given input data is in proper foramt or not.
- example : email Id verification, mobile number verification,password verification.
- The regular expressions which are supported by the 'pearl' are also supported by the python.
- Python is providing the built-in- functions to work with the regular expression easily.
- All the predefined functions which are related to regular expression are present in "re" module.

Special characters which are used in regular expression are :

	Description	Example	Example matches
.	Any character	.	a, b, .
*	Zero or more of the preceding group	.*	a, ab, abab, '' (empty string)
^	Beginning of string	^b.*	b, baaaa
$	End of string	b.*b$	bb, baaaab, abab
[ ]	Match any one in a set of characters	[a-cz]	a, b, c, z
[^]	Set of characters	[^a]	b, c, 1, 2
( )	Captured subexpression	(a.*)	a, abb
{m, n}	Match at least m and at most n of preceding group	a{2,4}	aa, aaa, aaaa
|	Or, alternation, either one or the other	a|b	a, b
+	One or more of the proceeding group	a+	a, aa, aaa
?	Zero or one	a?	'' (empty string), a

\d	Digit	\d	1, 5, 0
\D	Non-digit	\D	a, b,)
\s	Whitespace		
\S	Non-whitespace		
\w	Word character		
\W	Non-word character		
\b	Word boundary		

A **Reg**ular **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 five letter string starting with `a` and ending with `s`**.

A pattern defined using RegEx can be used to match against a string.

Expression	String	Matched?
^a...s$	abs	No match
	alias	Match
	abyss	Match
	Alias	No match
	An abacus	No match

Python has a module named `re` to work with RegEx. Here's an example:

```
import re
pattern = '^a...s$'
test_string = input('enter any string :')
result = re.match(pattern, test_string)

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

Output 1:

```
enter any string :abcds
Search successful.
```

Output 2:

```
enter any string :asdsdsfss
Search unsuccessful.
```

Here, we used `re.match()` function to search `pattern` within the `test_string`. The method returns a match object if the search is successful. If not, it returns `None`.

Specify Pattern Using RegEx

To specify regular expressions, metacharacters are used. In the above example, `^` and `$` are metacharacters.

MetaCharacters

Metacharacters are characters that are interpreted in a special way by a RegEx engine. Here's a list of metacharacters:

[] . ^ $ * + ? {} () \ |

`[]` - Square brackets

Square brackets specifies 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

Here, `[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 characters 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

A 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.

`$` - **Dollar**

The dollar symbol `$` is used to check if a string **ends 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`)

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

`*` - **Star**

The star symbol `*` matches **zero or more occurrences** of the pattern left to it.

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

? - **Question Mark**

The question mark symbol ? matches **zero or one occurrence** of the pattern left to it.

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

{} - **Braces**

Consider this code: {n,m}. This means at least n, and at most m repetitions of the pattern left to it.

Expression	String	Matched?
a{2,3}	abc dat	No match
	abc daat	1 match (at daat)
	aabc daaat	2 matches (at aabc and daaat)
	aabc daaaat	2 matches (at aabc and daaaat)

Let's try one more example. This RegEx [0-9]{2, 4} matches at least 2 digits but not more than 4 digits

Expression	String	Matched?
[0-9]{2,4}	ab123csde	1 match (match at ab123csde)
	12 and 345673	3 matches (12, 3456, 73)
	1 and 2	No match

| - **Alternation**

Vertical bar | is used for alternation (or operator).

Expression	String	Matched?
a|b	cde	No match
	ade	1 match (match at ade)
	acdbea	3 matches (at acdbea)

Here, a|b match any string that contains either a or b

() - **Group**

Parentheses () is used to group sub-patterns. For example, (a|b|c)xz match any string that matches either a or b or c followed by xz

Expression	String	Matched?
(a|b|c)xz	ab xz	No match
	abxz	1 match (match at abxz)
	axz cabxz	2 matches (at axzbc cabxz)

\ - **Backslash**

Backlash \ is used to escape various characters including all metacharacters. For example,

\$a match if a string contains $ followed by a. Here, $ is not interpreted by a RegEx engine in a special way.

If you are unsure if a character has special meaning or not, you can put \ in front of it. This makes sure the character is not treated in a special way.

Special Sequences

Special sequences make commonly used patterns easier to write. Here's a list of special sequences:

\A - Matches if the specified characters are at the start of a string.

Expression	String	Matched?
\Athe	the sun	Match
	In the sun	No match

\b - Matches if the specified characters are at the beginning or end of a word.

Expression	String	Matched?
\bfoo	football	Match
	a football	Match
	afootball	No match
foo\b	the foo	Match
	the afoo test	Match
	the afootest	No match

`\B` - Opposite of `\b`. Matches if the specified characters are **not** at the beginning or end of a word.

Expression	String	Matched?
\Bfoo	football	No match
	a football	No match
	afootball	Match
foo\B	the foo	No match
	the afoo test	No match
	the afootest	Match

`\d` - Matches any decimal digit. Equivalent to `[0-9]`

Expression	String	Matched?
\d	12abc3	3 matches (at 12abc3)
	Python	No match

`\D` - Matches any non-decimal digit. Equivalent to `[^0-9]`

Expression	String	Matched?
\D	1ab34"50	3 matches (at 1ab34"50)
	1345	No match

`\s` - Matches where a string contains any whitespace character. Equivalent to `[ \t\n\r\f\v]`.

Expression	String	Matched?
\s	Python RegEx	1 match
	PythonRegEx	No match

`\w` - Matches any alphanumeric character (digits and alphabets). Equivalent to `[a-zA-Z0-9_]`. By the way, underscore `_` is also considered an alphanumeric character.

Expression	String	Matched?
`\w`	`12&": ;c`	3 matches (at `12&": ;c`)
	`%"> !`	No match

`\W` - Matches any non-alphanumeric character. Equivalent to `[^a-zA-Z0-9_]`

Expression	String	Matched?
`\W`	`1a2%c`	1 match (at `1a2%c`)
	`Python`	No match

`\Z` - Matches if the specified characters are at the end of a string.

Expression	String	Matched?
`Python\Z`	`I like Python`	1 match
	`I like Python Programming`	No match
	`Python is fun.`	No match

Some special charecters :

[0-9] --------> Any single digit
[a-z] --> Any one lower case alphabet
[A-Z] ---> Any one of upper case alphabet
[a-Az-Z] --> Any one alphabet
[a-zA-Z0-9] --> Any one alphanumeric
[^0-9]---> Any single non digit
[^A-Z]---> Any one non uppercase alphabet
[^a-z]---> Any one non lowercase alphabet
[^a-zA-Z]---> Any one non alphabet
[^a-zA-z0-9-]---> Any one non alphanumeric

(|)---> It matches any one string in the list
EX: (java | hadoop | python)
.net # error

python # True

{ m } -------->It matches exact occurance of preceding character
EX: ab{ 3 }c
abbbc # True
abc # error
abbc # error

{ m , n } ----->It matches minimum "m" occurances and maximum "n" occurances of preceding character.
EX: Ab{ 3 , 5 }c
Abbc # error
abbbc

{ m , } ------> It matches minimum "m" occurances and maximum no limit of preceding character.
EX: Ab{ 3 , }c
Abc # error
abbbc
abbbbc

{ , m } ---->> maximum 3 times and minimum no limit

\d or [0-9] --------> Matches digits. Equivalent to [0-9]. (Any single digit)

Example: [0-9][0-9] or [0-9]{4} or \d \d \d\d or \d{4}

\D or [*0-9] ------> Matches nondigits. (Any single non digit)
\w or [a-zA-z0-9]--> Matches word characters. (any alphanumeric)
\W or [^a-zA-z0-9-]---> Matches nonword characters.
\s ---->> matches any empty spaces
\S ---->> Matches nonwhitespace.
\A ----->> Matches beginning of string.
\z ------>> Matches end of string.

Raw strings

Methods in re module use raw strings as the pattern argument. A raw string is having prefix 'r' or 'R' to the normal string literal.

```
>>> normal="computer"
>>> print (normal)
computer
>>> raw=r"computer"
>>> print (raw)
computer
```

Both strings appear similar. The difference is evident when the string literal embeds escape characters ('\n', '\t' etc.)

```
>>> normal="Hello\nWorld"
>>> print (normal)
Hello
World
>>> raw=r"Hello\nWorld"
>>> print (raw)
Hello\nWorld
```

In case of normal string, the print() function interprets the escape character. In this case '\n' produces effect of newline character.

However because of the raw string operator 'r' the effect of escape character is not translated as per its meaning. The output shows actual construction of string not treating '\n' as newline character.

Regular expressions use two types of characters in the matching pattern string: Meta characters are characters having a special meaning, similar to * in wild card. Literals are alphanumeric characters.