A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern.

The python module **re** provides regular expressions in python. The re module raises the exception re.error if an error occurs while compiling or using a regular expression.

Regular expression patterns

- ^ => Matches beginning of line
- $ => Matches end of line
- . => Matches any single character except newline. Using m option allows it to match newline as well.
- [...] => Matches any single character in brackets.
- [^...] => Matches any single character not in brackets
- re* => Matches 0 or more occurances of preceding expression.
- re+ => Matches 1 or more occurances of preceding expression.
- re? => Matches 0 or 1 occurances of preceding expression.
- re{n} => Matches exactly n number of occurances of preceding expression.
- re{n,} => Matches n or more occurances of preceding expression.
- re{n,m} => Matches at least n at most m occurances of preceding expression.
- a|b => Matcches either a or b
- (re) => Groups regular expressions and remembers matched text
- (?imx) => Temporarily toggles on i,m, or x options within a regular expression. If in parentheses, only that area is affected.
- (?-imx) => Temporarily toggles off i,m, or x options within a regular expression. If in parentheses, only that area is affected.
- (?:re) => Groups regular expression without remembering matched text.
- (?imx:re) => Temporarily toggles on i,m or x options within parentheses.
- (?-imx:re) => Temporarily toggles off i,m or x options within parentheses.
- (?#...) => Comment.
- (?=re) => Specifies position using a pattern. Doesn't have a range.
- (?!re) => Specifies position using pattern negation. Doesn't have a range.
- (?>re) => Matches independent pattern without backtracking.
- \w => Mathces word character
- \W => Matches nonword character
- \s => matches whitespace.Equivalent to [\t\n\r\f]
- \S => matches nonwhitespace.

- \d => Matches digits. Equivalent to [0-9]
- \D => matches non digits.
- \A => Matches beginning of string.
- \Z => Matches end of string. If a newline exits, it matches just before newline.
- \z => Matches end of string.
- \G => Matches point where last match finished.

Repetition Cases

```
ruby? => Match 'rub' or "ruby": the y is optional.
ruby* => Match 'rub' plus 0 or more ys
ruby+ => Match 'rub' plus 1 or more ys
Python(?=!) => Match 'Python', if followed by an exclamation point.
Python(?!!) => Match 'Python', if not followed by an exclamation point.
```

Non greedy repeatition This matches the smallest number of repeatition.

```
<.*> Greedy repeatition: matches "<python>perl>"

<.*?> Nongreedy repeatition: matches "<python>" in "<python>perl>"
```

Backreferences

This matches a previous matched group again

```
- ([Pp])ython&\1ails => Match python&pails or Python&Pails
- (['"])[^\1]*\1 => Single or double quoted string.
```

The match function

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

-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.
```

re.match function returns a match object on success, None on failure. We use group(num) or groups() function of match object to get matched expression.

- group(num=0) - This method returns entire match (or specific subgroup num)
- groups() - This method returns all matching subgroups in a tuple (empty if there weren't any)

In [2]:
```
import re

line = "Cats are smarter than dogs"
matchObj = re.match(r'(.*) are (.*?).*',line,re.M|re.I)
```

In [ ]:
```

```