Python 2.7 Regular Expressions

Special characters:

\ escapes special characters.
* matches any character
+ matches start of the string (or line if MULTILINE)
$ matches end of the string or line if MULTILINE)
[5b-d] matches any chars '5', 'b', 'c' or 'd'
[^a-c6] matches any char except 'a', 'b', 'c' or '6'
R|S matches either regex R or regex S.
() creates a capture group, and indicates precedence.

Within [], no special chars do anything special, hence they don't need escaping, except for ')', which only need escaping if they are not the 1st char.

\g<name> or \g<id> Match previous named or numbered group,
\A Start of string
\b Matches empty string not at word boundary (between \w and \W)
\B Matches empty string at word boundary (between \w and \W)
\B Matches empty string not at word boundary
\d Digit
\D Non-digit
\s Whitespace: \s, more if LOCAL or UNICODE
\S Non-whitespace
\w Alphanumeric: \w, is LOCAL or UNICODE
\W Non-alphanumeric
\[ End of string
\g<id> Match previous named or numbered group, e.g. \g<0> or \g<name>

Special character escapes are much like those already escaped in Python string literals. Hence regex '\n' is same as regex '\n':

\xHH Two digit hex character
\OOO Three digit octal char
\DD Decimal number 1 to 99, matches previous numbered group

Extensions. These do not cause grouping, except for (?P<name>...):

(?P<name>...) Creates a named capturing group.
(?P=...) Matches whatever matched previously named group.
(?...) A comment; ignored.
(?<...) Lookbehind assertion: Matches without consuming
(?[..]) Negative lookahead assertion
(?<[[..]]) Lookahead assertion: Matches if preceded

Flags for re.compile(), etc. Combine with |:

re.I == re.IGNORECASE Ignore case
re.L == re.LOCALE Make \w, \b, and \s locale dependent
re.M == re.MULTILINE Multiline
re.O == re.UNICODE Make \w, \b, \d, and \s unicode dependent
re.X == re.VERBOSE Verbose (unescaped whitespace in pattern
is ignored, and '#' marks comments)

Module level functions:

compile(pattern[, flags]) -> RegexObject
match(pattern, string[, flags]) -> MatchObject
search(pattern, string[, flags]) -> MatchObject
findall(pattern, string[, flags]) -> list of strings
finditer(pattern, string[, flags]) -> iter of MatchObjects
split(pattern, string[, maxsplit, flags]) -> list of strings
sub(pattern, repl, string[, count, flags]) -> string
subn(pattern, repl, string[, count, flags]) -> (string, int)

RegexObjects (returned from compile()):

.match(string, pos, endpos) -> MatchObject
.search(string, pos, endpos) -> MatchObject
.findall(pattern, string[, flags]) -> list of strings
.finditer(pattern, string[, flags]) -> iter of MatchObjects
.split(pattern, string[, maxsplit, flags]) -> list of strings
.sub(repl, string[, count]) -> string
.subn(repl, string[, count]) -> (string, int)
.flags # int passed to compile()
.groups # int number of capturing groups
.groupindex # {} maps group names to ints
.pattern # string passed to compile()

MatchObjects (returned from match() and search()):

.expand(template) -> string, backslash and group expansion
.group([group1...]) -> string or tuple of strings, i per arg
.groups([default]) -> (,) of all groups, non-matching-default
.start([group]) -> int, start/end of substring matched by group
.end([group]) -> int, start/end of substring matched by group
.pos # value passed to search() or match()
.endspos #
.lastindex # int index of last matched capturing group
.lastgroup # string name of last matched capturing group
.re # regex passed to search() or match()
.string # string passed to search() or match()

Gleaned from the python 2.7 're' docs.
http://docs.python.org/library/re.html

Version: v0.3.1
Contact: tartley@tartley.com