

# Python 2.7 Regular Expressions

Non-special chars match themselves. Exceptions are special characters:

<code>\</code>	Escape special char or start a sequence.
<code>.</code>	Match any char except newline, see <code>re.DOTALL</code>
<code>^</code>	Match start of the string, see <code>re.MULTILINE</code>
<code>\$</code>	Match end of the string, see <code>re.MULTILINE</code>
<code>[]</code>	Enclose a set of matchable chars
<code>R S</code>	Match either regex R or regex S.
<code>()</code>	Create capture group, & indicate precedence

After '[', enclose a set, the only special chars are:

<code>]</code>	End the set, if not the 1st char
<code>-</code>	A range, eg. <code>a-c</code> matches <code>a</code> , <code>b</code> or <code>c</code>
<code>^</code>	Negate the set only if it is the 1st char

Quantifiers (append '?' for non-greedy):

<code>{m}</code>	Exactly m repetitions
<code>{m,n}</code>	From m (default 0) to n (default infinity)
<code>*</code>	0 or more. Same as <code>{,}</code>
<code>+</code>	1 or more. Same as <code>{1,}</code>
<code>?</code>	0 or 1. Same as <code>{,1}</code>

Special sequences:

<code>\A</code>	Start of string
<code>\b</code>	Match empty string at word ( <code>\w+</code> ) boundary
<code>\B</code>	Match empty string not at word boundary
<code>\d</code>	Digit
<code>\D</code>	Non-digit
<code>\s</code>	Whitespace [ <code>\t\n\r\f\v</code> ], see <code>LOCALE,UNICODE</code>
<code>\S</code>	Non-whitespace
<code>\w</code>	Alphanumeric: [ <code>0-9a-zA-Z_</code> ], see <code>LOCALE</code>
<code>\W</code>	Non-alphanumeric
<code>\Z</code>	End of string
<code>\g&lt;id&gt;</code>	Match prev named or numbered group, ' <code>&lt;</code> ' & ' <code>&gt;</code> ' are literal, e.g. <code>\g&lt;0&gt;</code> or <code>\g&lt;name&gt;</code> (not <code>\g0</code> or <code>\gname</code> )

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

<code>\a</code>	ASCII Bell (BEL)
<code>\f</code>	ASCII Formfeed
<code>\n</code>	ASCII Linefeed
<code>\r</code>	ASCII Carriage return
<code>\t</code>	ASCII Tab
<code>\v</code>	ASCII Vertical tab
<code>\\</code>	A single backslash
<code>\xHH</code>	Two digit hexadecimal character goes here
<code>\OOO</code>	Three digit octal char (or just use an initial zero, e.g. <code>\0</code> , <code>\09</code> )
<code>\DD</code>	Decimal number 1 to 99, match previous numbered group

Extensions. Do not cause grouping, except 'P<name>':

<code>(?i)lmsux</code>	Match empty string, sets <code>re.X</code> flags
<code>(?:...)</code>	Non-capturing version of regular parens
<code>(?P&lt;name&gt;...)</code>	Create a named capturing group.
<code>(?P=name)</code>	Match whatever matched prev named group
<code>(?#...)</code>	A comment; ignored.
<code>(?=...)</code>	Lookahead assertion, match without consuming
<code>(?!...)</code>	Negative lookahead assertion
<code>(?&lt;=...)</code>	Lookbehind assertion, match if preceded
<code>(?&lt;!...)</code>	Negative lookbehind assertion
<code>(?(id)y n)</code>	Match 'y' if group 'id' matched, else 'n'

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

<code>re.I</code>	<code>== re.IGNORECASE</code>	Ignore case
<code>re.L</code>	<code>== re.LOCALE</code>	Make <code>\w</code> , <code>\b</code> , and <code>\s</code> locale dependent
<code>re.M</code>	<code>== re.MULTILINE</code>	Multiline
<code>re.S</code>	<code>== re.DOTALL</code>	Dot matches all (including newline)
<code>re.U</code>	<code>== re.UNICODE</code>	Make <code>\w</code> , <code>\b</code> , <code>\d</code> , and <code>\s</code> unicode dependent
<code>re.X</code>	<code>== re.VERBOSE</code>	Verbose (unescaped whitespace in pattern is ignored, and '#' marks comment lines)

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)
escape(string) -> string
purge() # the re cache
```

RegexObjects (returned from `compile()`):

```
.match(string[, pos, endpos]) -> MatchObject
.search(string[, pos, endpos]) -> MatchObject
.findall(string[, pos, endpos]) -> list of strings
.finditer(string[, pos, endpos]) -> iter of MatchObjects
.split(string[, maxsplit]) -> 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 & group expansion
.group([group1...]) -> string or tuple of strings, 1 per arg
.groups([default]) -> tuple of all groups, non-matching=default
.groupdict([default]) -> {}, Named groups, non-matching=default
.start([group]) -> int, Start/end of substring match by group
.end([group]) -> int, Group defaults to 0, the whole match
.span([group]) -> tuple (match.start(group), match.end(group))
.pos int, Passed to search() or match()
.endpos int, "
.lastindex int, Index of last matched capturing group
.lastgroup string, Name of last matched capturing group
.re regex, As passed to search() or match()
.string string, "
```

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

<https://github.com/tartley/python-regex-cheatsheet>  
Version: v0.3.3