

Special characters

.	Default: Match any character except newline
.	DOTALL: Match any character including newline
^	Default: Match the start of a string
^	MULTILINE: Match immediately after each newline
\$	Match the end of a string
\$	MULTILINE: Also match before a newline
*	Match 0 or more repetitions of RE
+	Match 1 or more repetitions of RE
?	Match 0 or 1 repetitions of RE
*?, *, +, ??	Match non-greedy as <i>few</i> characters as possible
{m}	Match exactly <i>m</i> copies of the previous RE
{m,n}	Match from <i>m</i> to <i>n</i> repetitions of RE
{m,n}?	Match non-greedy
\	Escape special characters
[]	Match a <i>set</i> of characters
	<i>RE1 RE2</i> : Match either RE1 or RE2 non-greedy
(...)	Match RE inside parentheses and indicate start and end of a group

With RE is the resulting regular expression.

Special characters must be escaped with \ if it should match the character literally

Methods of 're' module

<code>re.compile(pattern, flags=0)</code>	Compile a regular expression pattern into a regular expression object. Can be used with <i>match()</i> , <i>search()</i> and others
<code>re.search(pattern, string, flags=0)</code>	Search through <i>string</i> matching the first location of the RE. Returns a match object or None
<code>re.match(pattern, string, flags=0)</code>	If zero or more characters at the beginning of a string match <i>pattern</i> return a match object or None
<code>re.fullmatch(pattern, string, flags=0)</code>	If the whole <i>string</i> matches the <i>pattern</i> return a match object or None
<code>re.split(pattern, string, maxsplit=0, flags=0)</code>	Split <i>string</i> by the occurrences of <i>pattern</i> <i>maxsplit</i> times if non-zero. Returns a list of all groups.
<code>re.findall(pattern, string, flags=0)</code>	Return all non-overlapping matches of <i>pattern</i> in <i>string</i> as list of strings.
<code>re.finditer(pattern, string, flags=0)</code>	Return an iterator yielding match objects over all non-overlapping matches for the <i>pattern</i> in <i>string</i>

Methods of 're' module (cont)

<code>re.sub(pattern, repl, string, count=0, flags=0)</code>	Return the string obtained by replacing the leftmost non-overlapping occurrences of <i>pattern</i> in <i>string</i> by the <i>replacement repl</i> . <i>repl</i> can be a function.
<code>re.subn(pattern, repl, string, count=0, flags=0)</code>	Like sub but return a tuple (<i>new_string</i> , <i>number_of_subs_made</i>)
<code>re.escape(pattern)</code>	Escape special characters in <i>pattern</i>
<code>re.purge()</code>	Clear the regular expression cache

Raw String Notation

In raw string notation `r"te xt"` there is no need for the backslash character again.

```
>>> re.match(r"\W(.)\1\W", "
<re.Match object; span=(0, 4), mat
>>> re.match(" \\W (.) \\1 \\W "
<re.Match object; span=(0, 4), mat
```

Reference

<https://docs.python.org/3/howto/regex.html>
<https://docs.python.org/3/library/re.html>

Extensions

(?...)	This is the start of an extension
(? <i>aiLmsux</i>)	The letters set the corresponding flags <i>See flags</i>
(?:...)	A non-capturing version of regular parentheses



By **mutanclan** (mutanclan)
cheatography.com/mutanclan/

Published 19th April, 2019.
 Last updated 25th February, 2020.
 Page 1 of 3.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Extensions (cont)

(?P<name>...)	Like regular paranthes but with a <i>named</i> group
(?P=name)	A backreference to a <i>named</i> group
(?#...)	A comment
(?=...)	<i>lookahead assertion</i> : Matches if ... matches next without consuming the string
(?!...)	<i>negative lookahead assertion</i> : Matches if ... doesn't match next
(?<=....)	<i>positive lookbehind assertion</i> : Match if the current position in the string is preceded by a match for ... that ends the current position
(?<!...)	<i>negative lookbehind assertion</i> : Match if the current position in the string is not preceded by a match for ...
(?(id/name)yes-pattern no-pattern)	Match with <i>yes-pattern</i> if the group with gived <i>id</i> or <i>name</i> exists and with <i>no-pattern</i> if not

Match objects

Match.expand(<i>template</i>)	Return the string obtained by backslash substitution on <i>template</i> , as done by the sub() method
Match.group([<i>group1</i> ,...])	Returns one or more subgroups of the match. 1 Argument returns string and more arguments return a tuple .
Match.__getitem__(<i>g</i>)	Access groups with m[0], m[1]...
Match.groups(<i>default=None</i>)	Return a tuple containing all the subgroups of the match
Match.groupdict(<i>default=None</i>)	Return a dictionary containing all the <i>named</i> subgroups of the match, keyed by the subgroup name.
Match.start([<i>group</i>])	Return the indices of the start of the substring matched by <i>group</i>
Match.end([<i>group</i>])	Return the indices of the end of the substring matched by <i>group</i>
Match.span([<i>group</i>])	For a match <i>m</i> , return the 2-tuple (m.start([<i>group</i>]), m.end([<i>group</i>]))
Match.pos	The value of <i>pos</i> which was passed to the search() or match() method of the regex object
Match.endpos	Likewise but the value of <i>endpos</i>

Match objects (cont)

Match.lastindex	The integer index of the last matched capturing group, or None .
Match.lastgroup	The name of the last matched capturing group or None
Match.re	The regular expression object whose match() or search() method produced this match instance
Match.string	The string passed to match() or search()

Special escape characters

^	Match only at the start of the string
\b	Match the empty string at the beginning or end of a word
\B	Match the empty string when <i>not</i> at the beginning or end of a word
\d	Match any Unicode decimal digit this includes [0-9]
\D	Match any character which is not a decimal digit
\s	Match Unicode white space characters which includes [\t\n\r\v]
\S	Matches any character which is not a whitespace character. The opposite of \s
\w	Match Unicode word characters including [a-zA-Z0-9_]
\W	Match the opposite of \w
\Z	Match only at the end of a string

Regular Expression Objects

<code>Pattern.search(string[, pos[, endpos]])</code>	See <code>re.search()</code> . <code>pos</code> gives an index where to start the search. <code>endpos</code> limits how far the string will be searched.
<code>Pattern.match(string[, pos[, endpos]])</code>	Likewise but see <code>re.match()</code>
<code>Pattern.fullmatch(string[, pos[, endpos]])</code>	Likewise but see <code>re.fullmatch()</code>
<code>Pattern.split(string, maxsplit=0)</code>	Identical to <code>re.split()</code>
<code>Pattern.findall(string[, pos[, endpos]])</code>	Similar to <code>re.findall()</code> but with additional parameters <code>pos</code> and <code>endpos</code>
<code>Pattern.finditer(string[, pos[, endpos]])</code>	Similar to <code>re.finditer()</code> but with additional parameters <code>pos</code> and <code>endpos</code>
<code>Pattern.sub(repl, string, count=0)</code>	Identical to <code>re.sub()</code>
<code>Pattern.subn(repl, string, count=0)</code>	Identical to <code>re.subn()</code>
<code>Pattern.flags</code>	The regex matching flags.

Regular Expression Objects (cont)

<code>Pattern.groups</code>	The number of capturing groups in the pattern
<code>Pattern.groupindex</code>	A dictionary mapping any symbolic group names to group members
<code>Pattern.pattern</code>	The pattern string from which the pattern object was compiled

These objects are returned by the `re.compile()` method

Flags

ASCII, A	ASCII-only matching in <code>\w</code> , <code>\b</code> , <code>\s</code> and <code>\d</code>
IGNORECASE, I	ignore case
LOCALE, L	do a local-aware match
MULTILINE, M	multiline matching, affecting <code>^</code> and <code>\$</code>
DOTALL, S	dot matches all
u	unicode matching (just in <code>(?aiLmsux)</code>)
VERBOSE, X	verbose

Flags are used in `(?aiLmsux-imsx:...)` or `(?aiLmsux)` or can be accessed with `re.FLAG`. In the first form flags are set or removed.

This is useful if you wish to include the flags as part of the regular expression, instead of passing a flag argument to the `re.compile()` function

