

Regex cheat sheet

For more awesome cheat sheets
visit rebellabs.org!



Character classes

- `[abc]` matches **a** or **b**, or **c**.
- `[^abc]` negation, matches everything except **a**, **b**, or **c**.
- `[a-c]` range, matches **a** or **b**, or **c**.
- `[a-c[f-h]]` union, matches **a**, **b**, **c**, **f**, **g**, **h**.
- `[a-c&&[b-c]]` intersection, matches **b** or **c**.
- `[a-c&&[^b-c]]` subtraction, matches **a**.

Predefined character classes

- `.` Any character.
- `\d` A digit: `[0-9]`
- `\D` A non-digit: `[^0-9]`
- `\s` A whitespace character: `[\t\n\x0B\f\r]`
- `\S` A non-whitespace character: `[^\s]`
- `\w` A word character: `[a-zA-Z_0-9]`
- `\W` A non-word character: `[^\w]`

Boundary matches

- `^` The beginning of a line.
- `$` The end of a line.
- `\b` A word boundary.
- `\B` A non-word boundary.
- `\A` The beginning of the input.
- `\G` The end of the previous match.
- `\Z` The end of the input but for the final terminator, if any.
- `\z` The end of the input.

Pattern flags

- `Pattern.CASE_INSENSITIVE` - enables case-insensitive matching.
- `Pattern.COMMENTS` - whitespace and comments starting with `#` are ignored until the end of a line.
- `Pattern.MULTILINE` - one expression can match multiple lines.
- `Pattern.UNIX_LINES` - only the `\n` line terminator is recognized in the behavior of `.`, `^`, and `$`.

Useful Java classes & methods

PATTERN

A pattern is a compiler representation of a regular expression.

`Pattern.compile(String regex)`

Compiles the given regular expression into a pattern.

`Pattern.compile(String regex, int flags)`

Compiles the given regular expression into a pattern with the given flags.

`boolean matches(String regex)`

Tells whether or not this string matches the given regular expression.

`String[] split(CharSequence input)`

Splits the given input sequence around matches of this pattern.

`String quote(String s)`

Returns a literal pattern String for the specified String.

`Predicate<String> asPredicate()`

Creates a predicate which can be used to match a string.

MATCHER

An engine that performs match operations on a character sequence by interpreting a Pattern.

`boolean matches()`

Attempts to match the entire region against the pattern.

`boolean find()`

Attempts to find the next subsequence of the input sequence that matches the pattern.

`int start()`

Returns the start index of the previous match.

`int end()`

Returns the offset after the last character matched.

Quantifiers

Greedy	Reluctant	Possessive	Description
<code>X?</code>	<code>X??</code>	<code>X?+</code>	<i>X, once or not at all.</i>
<code>X*</code>	<code>X*?</code>	<code>X*+</code>	<i>X, zero or more times.</i>
<code>X+</code>	<code>X+?</code>	<code>X++</code>	<i>X, one or more times.</i>
<code>X{n}</code>	<code>X{n}?</code>	<code>X{n}+</code>	<i>X, exactly n times.</i>
<code>X{n,}</code>	<code>X{n,}?</code>	<code>X{n,}+</code>	<i>X, at least n times.</i>
<code>X{n,m}</code>	<code>X{n,m}?</code>	<code>X{n,m}+</code>	<i>X, at least n but not more than m times.</i>

Greedy - matches the longest matching group.

Reluctant - matches the shortest group.

Possessive - longest match or bust (no backoff).

Groups & backreferences

A group is a captured subsequence of characters which may be used later in the expression with a backreference.

`(...)` - defines a group.

`\n` - refers to a matched group.

`(\d\d)` - a group of two digits.

`(\d\d)/\1` - two digits repeated twice.

`\1` - refers to the matched group.

Logical operations

`XY` `X` then `Y`.

`X|Y` `X` or `Y`.

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

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

re.sub() <i>pattern</i> , <i>repl</i> , <i>string</i> , <i>count=0</i> , <i>flags=0</i>)	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.
re.subn() <i>pattern</i> , <i>repl</i> , <i>string</i> , <i>count=0</i> , <i>flags=0</i>)	Like sub but return a tuple (<i>new_string</i> , <i>number_of_subs_made</i>)
re.escape() <i>pattern</i>)	Escape special characters in <i>pattern</i>
re.purge()	Clear the regular expression cache

Raw String Notation

In raw string notation `r"text"` there is no need to escape the backslash character again.

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

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
(?	The letters set the correspondig aiLmsux) flags See <i>flags</i>
(?:...)	A non-capturing version of regular parantheses

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 doing 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 <i>m</i> [0], <i>m</i> [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 and end of the substring matched by <i>group</i>
Match.end([<i>group</i>])	
Match.span([<i>group</i>])	For a match <i>m</i> , return the 2-tuple (<i>m.start(group)</i> <i>m.end(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.last-index	The integer index of the last matched capturing group, or <i>None</i> .
Match.last-group	The name of the last matched capturing group or <i>None</i>
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

\A	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\f\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

Pattern.search() See `re.search()`. `string`, `pos`, `endpos` gives an index where to start the search. `endpos` limits how far the string will be searched.

Pattern.match() Likewise but see `re.match()` `string`, `pos`, `endpos`

Pattern.fullmatch() Likewise but see `re.fullmatch()` `string`, `pos`, `endpos`

Pattern.split() Identical to `re.split()` `string`, `maxsplit=0`

Pattern.findall() Similar to `re.findall()` but with additional parameters `pos` and `endpos` `string`, `pos`, `endpos`

Pattern.finditer() Similar to `re.finditer()` but with additional parameters `pos` and `endpos` `string`, `pos`, `endpos`

Pattern.sub() Identical to `re.sub()` `repl`, `string`, `count=0`

Pattern.subn() Identical to `re.subn()` `repl`, `string`, `count=0`

Pattern.flags The regex matching flags.

Regular Expression Objects (cont)

Pattern.groups The number of capturing groups in the pattern

Pattern.groupindex A dictionary mapping any symbolic group names to group members

Pattern.pattern 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 `\w`, `\b`, `\s` and `\d`

IGNORECASE, I ignore case

LOCALE, L do a local-aware match

MULTILINE, M multiline matching, affecting `^` and `$`

DOTALL, S dot matches all

u unicode matching (just in `(?aiLmsux)`)

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



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

Published 19th April, 2019.
Last updated 29th August, 2019.
Page 3 of 3.

Sponsored by **CrosswordCheats.com**
Learn to solve cryptic crosswords!
<http://crosswordcheats.com>