# Cheatography

# python regular expression (regex) Cheat Sheet

by mutanclan (mutanclan) via cheatography.com/79625/cs/19404/

## Special characters

| | |
|---|---|
| . | Default: Match any character except newline |
| . | DOTALL: Match any character including newline |
| ^ | Default: Match the start of a string |
| ^ | MULTILINE: Match immediatly 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 *few* characters as possible |
| {m} | Match exactly *m* copies of the previous RE |
| {m,n} | Match from *m* to *n* repetitions of RE |
| {m,n}? | Match non-greedy |
| \ | Escape special characters |
| [] | Match a *set* of characters |
| \| | *RE1\|RE2*: Match either RE1 *or* RE2 non-greedy |
| (...) | Match RE inside parantheses 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**(*pattern*, *flags=0*) | Compile a regular expression pattern into a regular expression object. Can be used with *match()*, *search()* and others |
| re.**search**(*pattern*, *string*, *flags=0* | Search through *string* matching the first location of the RE. Returns a **match object** or **None** |
| re.**match**(*pattern*, *string*, *flags=0*) | If zero or more characters at the beginning of a string match *pattern* return a **match object** or **None** |
| re.**fullmatch**(*pattern*, *string*, *flags=0*) | If the whole *string* matches the *pattern* return a **match object** or **None** |
| re.**split**(*pattern*, *string*, *maxsplit=0*, *flags=0*) | Split *string* by the occurrences of *pattern maxsplit* times if non-zero. Returns a **list** of all groups. |
| re.**findall**(*pattern, string, flags=0*) | Return all non-overlapping matches of *pattern* in *string* as **list** of strings. |
| re.**finditer**(*pattern, string, flags=0*) | Return an **iterator** yielding **match objects** over all non-overlapping matches for the *pattern* in *string* |

## Methods of 're' module (cont)

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

## Raw String Notation

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

```
>>> re.mat ch( r"\W (.) \1 \W", "
<re.Match object; span=(0, 4), mat
>>> re.mat ch( " \\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 |
| (?aiLmsux) | The letters set the correspondig flags *See flags* |
| (?:...) | A non-capturing version of regular parantheses |

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

Published 19th April, 2019.
Last updated 25th February, 2020.

## Extensions (cont)

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

## Match objects

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

## 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

| | |
|---|---|
| \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 *not* 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 |

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

Published 19th April, 2019.
Last updated 25th February, 2020.

Cheatography

# python regular expression (regex) Cheat Sheet

by mutanclan (mutanclan) via cheatography.com/79625/cs/19404/

## Regular Expression Objects

| | |
|---|---|
| Pattern.**search**(*string*[, *pos*[, *endpos*]]) | See `re.search()`. *pos* gives an index where to start the search. *endpos* limits how far the string will be searched. |
| Pattern.**match**(*string*[, *pos*[, *endpos*]]) | Likewise but see `re.match()` |
| Pattern.**fullmatch**(*string*[, *pos*[, *endpos*]]) | Likewise but see `re.fullmatch()` |
| Pattern.**split**(*string, maxsplit=0*) | Identical to `re.split()` |
| Pattern.**findall**(*string*[, *pos*[, *endpos*]]) | Similar to `re.findall()` but with additional parameters *pos* and *endpos* |
| Pattern.**finditer**(*string*[, *pos*[, *endpos*]]) | Similar to `re.finditer()` but with additional parameters *pos* and *endpos* |
| Pattern.**sub**(*repl, string, count=0*) | Identical to `re.sub()` |
| Pattern.**subn**(*repl, string, count=0*) | Identical to `re.subn()` |
| 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 25th February, 2020.