Regex cheat sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

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
X?	X??	X?+	*X, once or not at all.*
X*	X*?	X*+	*X, zero or more times.*
X+	X+?	X++	*X, one or more times.*
X{n}	X{n}?	X{n}+	*X, exactly n times.*
X{n,}	X{n,}?	X{n,}+	*X, at least n times.*
X{n,m}	X{n,m}?	X{n,m}+	*X, at least n but not more than m times.*

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**.

BROUGHT TO YOU BY JRebel

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"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
(?aiLmsux)	The letters set the correspondig flags *See flags*
(?:...)	A non-capturing version of regular parantheses

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***]**) 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

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