datacamp

Regular Expressions Cheat Sheet

Learn regular expressions online at www.DataCamp.com

What is a regular expression?

Regular expression (regex or regexp) is a pattern of characters that describes an amount of text. To process regexes, you will use a "regex engine." Each of these engines use slightly different syntax called regex flavor. A list of popular engines can be found here. Two common programming languages we discuss on DataCamp are Python and R which each have their own engines.

Since regex describes patterns of text, it can be used to check for the existence of patterns in a text, extract substrings from longer strings, and help make adjustments to text. Regex can be very simple to describe specific words, or it can be more advanced to find vague patterns of characters like the top-level domain in a url.

> Definitions

Literal character: A literal character is the most basic regular expression you can use. It simply matches the actual character you write. So if you are trying to represent an "r," you would write `r`.

Metacharacter: Metacharacters signify to the regex engine that the following character has a special meaning. You typically include a `\` in front of the metacharacter and they can do things like signify the beginning of a line, end of a line, or to match any single character.

Character class: A character class (or character set) tells the engine to look for one of a list of characters. It is signified by `[` and `]` with the characters you are looking for in the middle of the brackets.

Capture group: A capture group is signified by opening and closing, round parenthesis. They allow you to group regexes together to apply other regex features like quantifiers (see below) to the group.

> Anchors

Anchors match a position before or after other characters.

Syntax	Description	Example pattern	Example matches	Example non-matches
`^`	match start of line	`^r`	rabbit raccoon	parrot ferret
`$`	match end of line	`t$`	rabbit foot	trap star
`\A`	match start of line	`\Ar`	rabbit raccoon	parrot ferret
`\Z`	match end of line	`t\Z`	rabbit foot	trap star
`\b`	match characters at the start or end of a word	`\bfox\b`	the red fox ran the fox ate	foxtrot foxskin scarf
`\B`	match characters in the middle of other non-space characters	`\Bee\B`	trees beef	bee tree

> Matching types of character

Rather than matching specific characters, you can match specific types of characters such as letters, numbers, and more.

Syntax	Description	Example pattern	Example matches	Example non-matches
`.`	anything except for a linebreak	`c.e`	clean cheap	acert cent
`\d`	match a digit	`\d`	6060-842 2b|^2b	two **___
`\D`	match a non-digit	`\D`	The 5 cats ate 12 Angry men	52 10032
`\w`	match word characters	`\wee\w`	trees bee4	The bee eels eat meat
`\W`	match non-word characters	`\Wbat\W`	At bat Swing the bat fast	wombat bat53
`\s`	match whitespace	`\sfox\s`	the fox ate his fox ran	it's the fox. foxfur
`\S`	match non-whitespace	`\See\S`	trees beef	the bee stung The tall tree
`\metacharacter`	escape a metacharacter to match on the metacharacter	`\.` `\^`	The cat ate. 2^3	the cat ate 23

> Character classes

Character classes are sets or ranges of characters.

Syntax	Description	Example pattern	Example matches	Example non-matches
`[xy]`	match several characters	`gr[ea]y`	gray grey	green greek
`[x-y]`	match a range of characters	`[a-e]`	amber brand	fox join
`[^xy]`	does not match several characters	`gr[^ea]y`	green greek	gray grey
`[\^-]`	match metacharacters inside the character class	`4[\^\.-+*/]\d`	4^3 4.2	44 23

> Repetition

Rather than matching single instances of characters, you can match repeated characters.

Syntax	Description	Example pattern	Example matches	Example non-matches
`x*`	match zero or more times	`ar*o`	cacao carrot	arugula artichoke
`x+`	match one or more times	`re+`	green tree	trap ruined
`x?`	match zero or one times	`ro?a`	roast rant	root rear
`x{m}`	match m times	`\we{2}\w`	deer seer	red enter
`x{m,}`	match m or more times	`2{3,}4`	671-2224 2222224	224 123
`x{m,n}`	match between m and n times	`12{1,3}3`	1234 1222384	15335 1222223
`x*?, x+?, etc.`	match the minimum number of times - known as a lazy quantifier	`re+?`	tree freeeee	trout roasted

> Capturing, alternation & backreferences

In order to extract specific parts of a string, you can capture those parts, and even name the parts that you captured.

Syntax	Description	Example pattern	Example matches	Example non-matches
`(x)`	capturing a pattern	`(iss)+`	Mississippi missed	mist persist
`(?:x)`	create a group without capturing	`(?:ab)(cd)`	Match: abcd Group 1: cd	acbd
`(?<name>x)`	create a named capture group	`(?<first>\d)(?<scrond>\d)\d*`	Match: 1325 first: 1 second: 3	2 hello
`(x|y)`	match several alternative patterns	`(re|ba)`	red banter	rant bear
`\n`	reference previous captures where n is the group index starting at 1	`(b)(\w*)\1`	blob bribe	bear bring
`\k<name>`	reference named captures	`(?<first>5)(\d*)\k<first>`	51245 55	523 51

> Lookahead

You can specify that specific characters must appear before or after you match, without including those characters in the match.

Syntax	Description	Example pattern	Example matches	Example non-matches
`(?=x)`	looks ahead at the next characters without using them in the match	`an(?=an)` `iss(?=ipp)`	banana Mississippi	band missed
`(?!x)`	looks ahead at next characters to not match on	`ai(?!n)`	fail brail	faint train
`(?<=x)`	looks at previous characters for a match without using those in the match	`(?<=tr)a`	trail translate	bear streak
`(?<!x)`	looks at previous characters to not match on	`(?!tr)a`	bear translate	trail strained

> Literal matches and modifiers

Modifiers are settings that change the way the matching rules work.

Syntax	Description	Example pattern	Example matches	Example non-matches
`\Qx\E`	match start to finish	`\Qtell\E` `\Q\d\E`	tell \d	I'll tell you this I have 5 coins
`(?i)x(?-i).`	set the regex string to case-insensitive	`(?i)te(?-i)`	sTep tEach	Trench bear
`(?x)x(?-x)`	regex ignores whitespace	`(?x)t a p(?-x)`	tap tapdance	c a t rot a potato
`(?s)x(?-s)`	turns on single-line/ DOTALL mode which makes the "." include new-line symbols (\n) in addition to everything else	`(?s)first and second(?-s) and third`	first and Second and third	first and second and third
`(?m)x(?-m)`	changes ^ and $ to be end of line rather than end of string	`^eat and sleep$`	eat and sleep eat and sleep	treat and sleep eat and sleep.

> Unicode

Regular expressions can work beyond the Roman alphabet, with things like Chinese characters or emoji.

- **Code Points:** The hexadecimal number used to represent an abstract character in a system like unicode.
- **Graphemes:** Is either a codepoint or a character. All characters are made up of one or more graphemes in a sequence.

Syntax	Description	Example pattern	Example matches	Example non-matches
`\X`	match graphemes	`\u0000gmail`	@gmail www.email@gmail	gmail @aol
`\X\X`	match special characters like ones with an accent	`\u00e8 or \u0065\u0300`	è	e