REGEX CHEATSHEET

https://regexlearn.com/learn/regex101

Expression		Examples		
		text	*expression*	*returns*
Intro				
		“I have no special talents. I am only passionately **curious**.” — Albert Einstein	`/curious/gm`	curious
		“Every man takes the limits **of** his own field **of** vision for the limits **of** the world.” — Arthur Schopenhauer	`/of/g`	of, of, of
`.`	**Period** The period `.` allows selecting any character, including special characters and spaces.	abcABC123 .:!?	`/./gm`	a, b, c, A, B, C, 1, 2, 3, , ., :, !, ?
		az AZ 09 _- = !? ., :;	`/./g`	*All characters*
`[]`	**Character Sets** If one of the characters in a word can be various characters, we write it in square brackets `[]` with all alternative characters.	bar ber bir bor bur	`/b[aeiou]r/g`	bar, ber, bir, bor, bur
		beer deer feer	`/[bdf]eer/g`	beer, deer, feer
`[^]`	**Negated Character Sets** If one of the characters in a word cannot be within various characters, we write it in square brackets with all unwanted characters preceeded by a caret `[^]`.	bar ber bir bor bur	`b[^eo]r`	bar, bir, bur
		bear beor beer beur	`/be[^ou]r/g`	bear, beer
`[-]`	**Letter Range** To find the letters in the specified range, the starting letter and the ending letter are written in square brackets `[]` with a dash between them `-`. It is case-sensitive.	abcdefghijklmnopqrstuvwxyz	`/[e-o]/g`	e, f, g, h, i, j, k, l, m, n, o
		abcdefghijklmnopqrstuvwxyz	`/[g-k]/g`	g, h, i, j, k
`[-]`	**Number Range** To find the numbers in the specified range, the starting number and the ending number are written in square brackets `[]` with a dash `-` between them.	0123456789	`/[3-6]/g`	3, 4, 5, 6
		0123456789	`/[2-7]/g`	2, 3, 4, 5, 6, 7
Repetition				

*	**Asterisk** We put an asterisk `*` after a character to indicate that the character may either not match at all or can match many times.	br ber beer	/be*r/g	br, ber, beer
		dp dep deep	/de*p/g	dp, dep, deep
+	**Plus Sign** To indicate that a character can occur one or more times, we put a plus sign `+` after a character.	br ber beer	/be+r/g	ber, beer
		dp dep deep	/de+p/g	dep, deep
?	**Question Mark** To indicate that a character is optional, we put a `?` question mark after a character	color colour	/colou?r/g	color, colour
		a an	/an?/g	a, an
{n}	**Curly Braces** To express a certain number of occurrences of a character, at the end we write curly braces `{n}` along with how many times we want it to occur.	ber beer beeer beeeer	/be{2}r/g	beer
		Release 10/9/2021	/[0-9]{4}/g	2021
{n,}	**Curly Braces** To express at least a certain number of occurrences of a character, we write the end of the character at least how many times we want it to occur, with a comma `,` at the end, and inside curly braces `{n,}.	ber beer beeer beeeer	/be{3,}r/g	beeer, beeeer
		Release 10/9/2021	/[0-9]{2,}/g	10, 2021
{x,y}	**Curly Braces** To express the occurrence of a character in a certain number range, we write curly braces `{x,y}` with tihe interval we want to go to the end.	ber beer beeer beeeer	/be{1,3}r/g	ber, beer, beeer
		Release 10/9/2021	/[0-9] {1,4}/g	10, 9, 2021

Groupping

()	**Parentheses** We can group an expression and use these groups to reference or enforce some rules. To group an expression, we enclose in parentheses `()`.	ha-ha,haa-haa	/(haa)/g	haa haa
\1	**Referencing a Group** The first group is used by writing `\1` to avoid rewriting. Here `1` denotes the order of the grouping	ha-ha,haa-haa	/(ha)-\1, (haa)-\2/g	ha-ha,haa-haa
(?:)	**Non-capturing Group** You can group an expression and ensure that it is not captured by references.	ha-ha,haa-haa	/(?:ha)-ha, (haa)-\1/g	ha-ha,haa-haa
|	**Pipe Character**	cat Cat rat	/(C|c)at| rat/g	cat, Cat, rat

	It allows to specify that an expression can be in different expressions. Thus, all possible statements are written separated by the pipe sign `|`. This differs from charset `[abc]`. Charsets operate at the character level. Alternatives are at the expression level.			

Characters

`\`	**Escape Character** There are special characters that we use when writing regex. `{}[]/\+*.$^|?`. Before we can select these characters themselves, we need to use an escape character `\`.	(*) Asterisk.	`/(\*|\.)/g`	*, .
`^`	**Caret Sign. Selecting by Line Start** To find a pattern at the beginning of a line, prefix this expression with the `^` sign.	Basic Omelette Recipe **1**. 3 eggs, beaten **2**. 1 tsp sunflower oil **3**. 1 tsp butter	`/^[0-9]/gm`	1, 2, 3
`$`	**Dollar sign. Selecting by End of Line** To find a pattern at the end of the line, postfix this expression with the `$` sign.	https://domain.com/ what-is-html.**html** https://otherdomain.com/ html-elements https://website.com/html 5-features.**html**	`/html$/gm`	html, html
`\w`	**Word Character** The expression `\w` is used to find letters, numbers and underscore characters.	abcABC123 _.:!?	`/\w/g`	a, b, c, A, B, C, 1, 2, 3, _
`\W`	**Except Word Character** The expression `\W` is used to find characters other than letters, numbers and underscores.	abcABC123 _.:!?	`/\W/g`	·, ., :, !, ?
`\d`	**Number Character** `\d` is used to find only number characters.	abcABC123 _.:!?	`/\d/g`	1, 2, 3
`\D`	**Except Number Character** `\D` is used to find non-numeric characters.	abcABC123 _.:!?	`/\D/g`	a, b, c, A, B, C, ·, ., :, !, ?
`\s`	**Space Character** `\s` is used to find only space characters.	abcABC123 _.	`/\s/g`	·
`\S`	**Except Space Character** `\s` is used to find non-space characters.	abcABC123 _.	`/\S/g`	a, b, c, A, B, C, ., :, !, ?

Lookarounds

`(?=)`	**Positive Lookahead** To select a pattern that has a specific	Date: 4 Aug **3**PM	`/\d+(?=PM)/g`	3

	pattern after them, we write the positive look-ahead expression `(?=)` after our target expression.			
`(?!)`	**Negative Lookahead** To select a pattern that doesn't have a specific pattern after them, we write the negative look-ahead expression `(?!)` after our target expression.	Date: **4** Aug 3PM	`/\d+(?!PM)/g`	4
`(?<=)`	**Positive Lookbehind** To select a pattern that has a specific pattern before them, we write the positive look-ahead expression `(?<=)` before our target expression.	Product Code: 1064 Price: $**5**	`/(?<=\$)\d+/g`	5
`(?<!)`	**Negative Lookbehind** o select a pattern that doesn't have a specific pattern before them, we write the positive look-ahead expression `(?<!)` before our target expression.	Product Code: **1064** Price: $5	`/(?<!\$)\d+/g`	1064

Flags

`//g`	**Global Flag** The `global` flag causes the expression to select all matches. If not used it will only select the first match	domain.com test.com site.com	`/\w+\.com$/g`	domain.com, test.com, site.com
		domain.com test.com site.com	`/\w+\.com$/`	domain.com
`//m`	**Multiline Flag** Regex sees all text as one line. But we use the `multiline` flag to handle each line separately. In this way, the expressions wwe write to identify patterns at the end of lines work separately for each line.	domain.com test.com site.com	`/\w+\.com$/gm`	domain.com, test.com, site.com
		domain.com test.com site.com	`/\w+\.com$/g`	site.com
		domain.com test.com site.com	`/\w+\.com$/m`	domain.com
`//i`	**Case-insensitive Flag** In order to remove the case-sensitivity of the expression we have written, we must activate the `case-insensitive` flag.	DOMAIN.COM TEST.COM SITE.COM	`/\w+\.com$/gmi`	DOMAIN.COM, TEST.COM, SITE.COM
		DOMAIN.COM TEST.COM SITE.COM	`/\w+\.com$/gm`	*Nothing*

Matching

	Greedy Matching Regex does a greedy match by deafult. This means that the matchmaking will be as long as possible.	ber beer beeer beeeer	`/.*r/`	ber·beer·beeer·beeeer *(All text)*
?	**Lazy Matching** Lazy matchmaking, unlike greedy matching, stops at the first matching.	ber beer beeer beeeer	`/.*?r/`	ber, ·beer, ·beeer, ·beeeer