

WHENEVER I LEARN A
NEW SKILL I CONCOCT
ELABORATE FANTASY
SCENARIOS WHERE IT
LETS ME SAVE THE DAY.

OH NO! THE KILLER
MUST HAVE FOLLOWED
HER ON VACATION!



BUT TO FIND THEM WE'D HAVE TO SEARCH
THROUGH 200 MB OF EMAILS LOOKING FOR
SOMETHING FORMATTED LIKE AN ADDRESS!

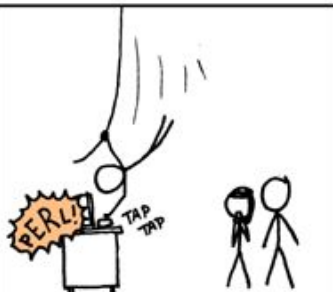


IT'S HOPELESS!

EVERYBODY STAND BACK.



I KNOW REGULAR
EXPRESSIONS.



`import re`

Write only code

1. Slow performance for bad regexps
2. Case sensitive: “Lol” vs “lol”
3. Sensitiveness to problem changes
4. For simple queries, built-in function are much more efficient
5. Difficult to read

```
(?:[a-z0-9!#$%&'*/+=?^_`{|}~-]+(?:\. [a-z0-9!#$%&'*/+=?^_`{|}~-]+) *|\"(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f]) *\") @ (?: (?: [a-z0-9] (?: [a-z0-9-]* [a-z0-9]) ?\\. ) + [a-z0-9] (?: [a-z0-9-]* [a-z0-9]) ? | \\[ (?: (? : 25 [0-5] | 2 [0-4] [0-9] | [01] ? [0-9] [0-9] ?) \\.) {3} (? : 25 [0-5] | 2 [0-4] [0-9] | [01] ? [0-9] [0-9] ?) ? | [a-z0-9-]* [a-z0-9] : (? : [\\x01-\\x08\\x0b\\x0c\\x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f]) +) \\])
```

Simple rules

Rule	Description	Example	Result
.	Any symbol except new line \n	e.e	eye, elephant, exe, prelevement
\d	Any digit	\ds	In the beginning of 60s
\D	Any symbol, except digits	\D	In the beginning of 60s
\s	Any whitespace symbol	M \s S	M Soyer, ISM series, GYM Section B
\S	Any non-whitespace symbol	\S123	K1234, 1 + 123, #123
\w	Any letter, digit or underscore	\w\w\w\w	Year, K1_0, cubics
\W	Any symbol except letters, digits and underscore	quite\W	quite!, 'quite, please', quietest

Simple rules

Rule	Description	Example	Result
[. . .]	Any symbol in brackets, or within the range specified	[0-9][0-9A-Fa-f]	12, 1F, 4S, E4
[. . .]	Any symbol in brackets, or within the range specified	gr[ae]y	gray, grey
[^ . . .]	Any symbol except specified in brackets	[^_ ^o]	2^4, k_pop
	Minus must be either in the end or in the beginning of matching letters	[abc-], [-1]	
	Only \ and] symbols should be escaped	[*[(+\\ \\)\t]	
\b	Word boundary	\bcorn\b	Pop corn, unicorn, corner
\b	Word boundary	\bcorn	Pop corn, unicorn, corner
\B	Not word boundary	corn\B	Pop corn, unicorns, corner

Quantifiers

Rule	Description	Example	Result
$\{n\}$	Exact n times	<code>\d{4}</code>	12,123, 1234 , 12345
$\{m,n\}$	From m to n times	<code>\d{3,4}</code>	12, 123 , 1234 , 12345
$\{m,\}$	At least m times	<code>\d{3,}</code>	12, 123 , 1234 , 12345
$\{,n\}$	At most n times	<code>\d{,4}</code>	12 , 123 , 1234 , 12345
<code>?</code>	0 or 1 entry = $\{0,1\}$	eggs?	eggs , eggy , egg
<code>*</code>	0 or more times = $\{0,\}$	Ex\d*	Ex , Ex1 , Ex12 , Ex666
<code>+</code>	1 or more times = $\{1,\}$	Ba+m	Baaaaaam , Bam , Bm
<code>*?</code> <code>+?</code> <code>??</code> $\{m,n\}?$	Quantifiers are greedy - accept the most possible number of symbols. Question mark make them lazy and take the least possible number of letters.	<code>. * ? -</code>	Quantifiers are greedy - accept the most possible number of symbols.

Regular Expressions in Python

Function	Description	Example
<code>re.search(pattern, string)</code>	Find first match of <code>pattern</code> in <code>string</code>	<code>re.search(r'\W+', 'Words, words, words.')</code> - <code><_sre.SRE_Match object; span=(5, 7), match=', '></code>
<code>re.fullmatch(pattern, string)</code>	Check if the whole <code>string</code> matches the regular expression <code>pattern</code>	<code>re.fullmatch(r'bar', 'barcode')</code> - None
<code>re.split(pattern, string, maxsplit=0)</code>	Similar to <code>str.split()</code> . Splits <code>string</code> using <code>pattern</code> as separator	<code>re.split(r'\W+', 'Words, words, words.')</code> - <code>['Words', 'words', 'words', '']</code>
<code>re.findall(pattern, string)</code>	Find all non-overlapping <code>patterns</code> in <code>string</code>	<code>re.findall(r'\w+', 'Words, words, words.')</code> - <code>['Words', 'words', 'words']</code>
<code>re.finditer(pattern, string)</code>	Return an iterator with all non-overlapping matches of <code>pattern</code> in <code>string</code> . Empty matches are included in the result.	<code>m=re.finditer(r'\w+', 'Words, words, words.')</code> <code>print([m1[0] for m1 in m])</code>
<code>re.sub(pattern, repl, string, count=0)</code>	Substitute all non-overlapping matches of <code>pattern</code> for <code>repl</code> in <code>string</code>	<code>re.sub(r'\w+', 'dog', 'Words, words, words.')</code> - <code>dog, dog, dog.</code>

Escape

- For regex `.` `^` `$` `*` `+` `?` `{` `}` `[` `]` `\` `|` `(` `)` should be escaped with `\`.
- For Python string variable `\` should be escaped with `\` as well.

```
re.search('\\\\\\par', '\\\\\\par'):
```

1. `\\\\\\par` in Python is read as `\\par`, since first `\` escapes second `\`.
2. `\\par` received by regex is read as `\par`, since first `\` escapes second `\`.
3. Therefore `\\\\\\par` corresponds to `\par` in the end.

Instead, use `re.search(r'\\par', '\\\\\\par')` where `r` tells Python not to treat `\` as escape symbol (except the case of opening quotes).

Grouping and Enumerating (or operation)

Rule	Description	Example	Result
	Logical Or	cat dog l[io]on	I have pets - 1 cat , 2 dogs , small loon ; and I want to get a lion .
(. .)	Grouping the pattern	((\d) (\d)) ((\d) (\d))	123456789 -> 1234 -> 12 -> 1,2 34 -> 3,4
(?:. .){x}	Grouping the pattern to be able to repeat it	(?:[0-9A-Fa-f]{2}[:-]) {5} [0-9A-Fa-f]{2}	01:23:45:67:89:ab

Examples

Example	Result
<code>(?:\w\w\d\d)+</code>	I saw ISE14a, MUI86, but one told me about ISE13MUI76.
<code>(?:\w+\d+)+</code>	I saw ISE14a, MUI86, but one told me about ISE13MUI76.
<code>(?:\+33 07 06) (?:\d{2,3}){4}</code>	+33 761 61 24 22, 07 61 61 24 22, 06 61 61 24 22
<code>(?:[Hh][aeiy]+)+</code>	Hahahahaa, hahahihihhi, hyperactive
<code>\b(?:[Hh][aeiy]+)+\b</code>	Hahahahaa, hahahihihhi, hyperactive
<code>re.sub(r'(\d\d)/(\d\d)/(\d{4})', r'\2.\1.\3', text)</code>	text: "We arrive on 03/25/2018. So you are welcome after 04/01/2018." Out: "We arrive on 25.03.2018. So you are welcome after 01.04.2018."

LABS TIME

