The Little Regular Expressionist

Vilja Hulden

August 2016
v0.1b

This little pamphlet, which is inspired by *The Little Schemer* by Daniel Friedman and Matthias Felleisen, aims to serve as a gentle introduction to regular expressions.

You may want to cover the right half of the page, and only move the cover down answer by answer; that way you give yourself time to digest the question and maybe even come up with the answer (sometimes you will have enough information to at least take a guess, though not always).

This pamphlet is by no means exhaustive; there is much more to regular expressions. It's a good idea to test and play; `http://regexr.com/` has both a tester and reference resources, and a good comprehensive cheat sheet can be found at `https://www.cheatography.com/davechild/cheat-sheets/regular-expressions/`.

1 Basics

Is `a` a regular expression?	Yes, it matches string `a`.
Is `ba*c` a regular expression?	Yes.
Does `ba*c` match `bac`?	Yes.
Does `ba*c` match `baaaac`?	Yes.
Does `ba*c` match `bc`?	Yes.

Hmm. So do you mean that `ba*c` can be read as "`b` followed by zero or more `a`'s and then a `c`?"	I do indeed.
So `Mo*re` would match both `More` and `Moore`?	Yes, but it would of course also match `Mre`.
Oh. What if I don't want it to match `Mre`?	Use the plus (`+`) instead of the star (`*`).
Aha! So `ba+c` matches `bac` but not `bc`?	Exactly.
OK, so does `ba+c` match `baaaac`?	Yes, because `+` means "one or more."
And it also matches `baac` and `baaaaac` and `baaaaaaaac` and ...	Yes, you've got the idea.
So `ba+c` can be read as "`b` followed by one or more `a`'s and then a `c`?"	Quite so.
OK, let me check a few more. Does `b*a` match `ba`?	Yes.
Does `b*a` match `a`?	Yes, because the star means you don't have to have a `b`.
Does `b+a+` match `aaaa`?	No, because the plus means that you have to have at least one b.
Does `b+a` match `bbbba`?	Yes, because you can have as many `b`s as you like, as long as you have at least one.
Does `b*a` match `bbbb`?	No. The `a` is not followed by a star, so it has to be there.

Does `ba*c` match `BAC`?	No, because the matches are case-sensitive.
Does `b*a` match `bbBBa`?	No, because the matches are case-sensitive! A lowercase `b` only matches a lowercase `b`, not an uppercase `B`.
Oh, OK. What if I want to match both?	We'll get to that, don't worry.
Fine. What if I want to match any character?	Use `.` (the period).
Like this: `b.c` to match `bac`?	Yes.
Or the same thing, `b.c`, to match `bxc`?	Yes.
Does `b.c` match `baaaac` too?	No.
Does it match `bxxxxc`?	No. The period only matches a single character.
Oh. So does `b.*c` match `baaaac`?	Yes!
Does `b.*c` match `bbbbbaaaac`?	Yes, because `b` is also a character.
Does `b.*k` match `bark`?	Yes, because both `a` and `r` are characters.
Does `B.*K` match `bark`?	No, because the matches are case-sensitive. `B` only matches an uppercase `B`, not a lowercase `b`.
OK. But does `c.*p` match `co-op`?	Yes, because `-` (the hyphen) is a character too.

So `c.*p` would match `c&?$#e$p`?	Yes, those are all characters.
Does `a.*p` match `b635%#p`?	No, because there's no `a`.
Does `a.*p` match `a&4>p` ?	Yes.

What if I have a word that looks like this:
`ab`
`cd`
Will `a.*d` match it?
	No, the one character that `.` does not match is the line break.

Oh, so it will only match a string if the string is all on one line?	Yes.
So let's say I have a line like this: `Cats beat dogs` I want to match the `Cats` in that line. I write `C.*s` to do that, right?	Actually, no.
What??	`C.*s` will match the whole line.
Why??	Because **regular expressions are greedy.** They take everything they can.
Oh, so `C.*s` will actually match the whole string `Cats beat dogs` because that string ends with s too?	Exactly.
So, let me make sure I've got all this right.	Please do.
What's the minimum number of `a`'s a string has to have for the expression `ba*c` to match?	Zero.

And what's the minimum number of `a`'s a string has to have for the expression `ba+` to match?	One.
Is there an upper limit to how many consecutive `a`'s there can be in a string for the expressions `ba*` or `ba+` to match?	No.
What does `.` match?	Any character except line breaks.
And, **a regular expression is greedy.**	Yes!!
Can I make it "ungreedy?" Can I match the `Cats` in `Cats beat dogs` somehow?	Yes, you can add `?` to the quantifier (the star or plus).
Like this: `C.*?s`?	Exactly.

Cheat sheet	
`.`	any character
`*`	zero or more
`+`	one or more
`*?`	zero or more, ungreedy
`+?`	one or more, ungreedy

2 Alternatives

Does `a|b` match `a`?	Yes.
Does `a|b` match `b`?	Yes.
Does `a|b` match `c`?	No, there's nothing in the expression that could match `c`.
Does `a|b` match `ab`?	No.
So `a|b` means "a or b"?	Yes!
Does `a|b+` match `abbbb`?	No, it only matches one or the other side of the `|`, not both at the same time.
OK. So does `a|b+` match `bbbb`?	Yes.
And does `a|b+` match `a`?	Yes.
Does `a|b+` match `aaa`?	No, because there's only one `a` in the expression.
Does `a+|b+` match `aaabbb`?	No, it still only matches one or the other side of the pipe!
Oh, right. So does `a+|b+` match `aaa`?	Yes.
Does `a+|b+` match `bbb`?	Yes.
So `dog|cat` matches `dog`?	Yes.
But does `dog|cat` match `dogs`?	No, there's no `s` in the expression.

Oh yeah, that's true. But does `dog|cat` match the `dog` in `dogs`?	Yes!
Does `dog|cat` match `docat`?	No, because the whole expression on one side of the `|` (the pipe) has to match.
Ah, right. So does `dog|cat` match the `cat` in `docat`?	Yes!
What if I want to match both `dogat` and `docat`?	Group the `g|c` with parentheses.
Like this: `do(g|c)at`?	Yes. That matches `dogat` as well as `docat`.
Does `dog|cats` match `dogs`?	No, you can't combine the two sides of the pipe.
Does `dog|cats` match `cats`?	Yes, because `cats` is all on one side of the pipe.
Does `(dog|cat)s` match `dogs`?	Yes, because now you've grouped the alternatives.
Does `(dog|cat)s` match `cats`?	Yes, because you've grouped the alternatives.
Does `(dog|cat)s` match `cat`?	No, because the `s` has to be there.
Does `(dog|cat)s*` match `cat`?	Yes, because now you've made the `s` optional.

3 Character classes

Does `[abc]` match `a`?	Yes, because `a` is included in the class.
Does `[abc]` match `b`?	Yes, because `b` is included in the class.
Does `[abc]` match `d`?	No, because `d` is not included in the class.
Does `[abc]` match `ab`?	No, because the class only represents one of its members at a time.
Does `[abc][abc]` match `ab`?	Yes, because now there are two classes in a row.
Does `[abc][abc]` match `cb`?	Yes, because the class can represent any one of its members.
Does `[abc]+` match `ab`?	Yes! Very good.
Does `[abc]+` match `ba`?	Yes.
Does `b[abc]*` match `ba`?	Yes.
Does `[abc]+` match `bacab`?	Yes.
So if a class is followed by `*` or `+`, any number of the members of that class can be strung together in any order?	Yes (well, to be precise, zero or more for `*` and one or more for `+`).
If a class contains all letters between a and h, do I have to list them all, like this: `[abcdefgh]`?	No, you can define the range like this: `[a-h]`.
So, `[a-z]` matches `a`?	Yes.

And `[a-z]` matches `g`?	Yes.
And `[a-z]` match `k`?	Yes. We could go on.
Let's not. But does [`a-z]` match `A`?	No, the class is case sensitive.
Does `[a-z]` match `aa`?	No, because the class only represents one of its members at a time.
Does `[a-z]+` match `abc`?	Yes, of course.
Does `[a-z]+` match `bye`?	Yes, of course.
Does `[a-z]` match `a-z`?	No, the expression defines a class, not a string.
Does `[a-k]` match `k`?	Yes.
Does `[a-k]` match `x`?	No, `x` is not included in the range `a-k`.
Does `[a-z]` match `2`?	No, no digits are included in the range `a-z`.
Does `[a-z]` match `&`?	No, no punctuation marks are included in the range `a-z`.
Does [`0-5]` match `2`?	Yes.
Does `[0-5]` match `7`?	No, `7` is not included in the range `0-5`.
Does `[0-5]` match `b`?	No, no letters are included in the range `0-5`.

Does `[a-z0-9]` match `b`?	Yes.
Does `[a-z0-9]` match `5`?	Yes.
Does `[a-z0-9]` match `b5`?	No, because the class only represents one of its members at a time.
Does `[a-z0-9]+` match `b5`?	Yes, because both `b` and `5` are members of the class and `+` means one or more.
Does `[a-z0-9]+` match `good4you`?	Yes.
Does `[a-z0-9]+` match `good4you!` ?	No, because the exclamation mark is not a member of the class.
Does `[a-z0-9!]+` match `good4you!` ?	Yes, because now you've added the exclamation mark to the class.
So if I put any characters inside square brackets, those characters become members of a class?	Yes.
Hey, couldn't I use this to match both uppercase and lowercase letters, like I wanted to do earlier (on page 1)?	Yes!
Like this: `[Bb]ob` to match both "bob" and "Bob"?	Yes!
Or different spellings, like `gr[ae]y` to match both `grey` and `gray`?	Absolutely!
OK, so by saying `[0-9]` I can match anything that's a number.	That's right.

But what if I want to match anything *except* a number? Do I have to make a class that lists everything that's not a number?	No, silly, of course not.
So how do I match anything except a number?	You negate the number class with `^` (a caret).
Oh, like this: `[^0-9]`?	Exactly.

Cheat sheet	
`a|b`	a or b
`[abc]`	a or b or c
`[a-z]`	any lowercase letter in the range a-z
`[A-Z]`	any uppercase letter in the range a-z
`[Bb]`	uppercase or lowercase b
`[0-9]`	any number in the range 0-9
`[0-4]`	any number in the range 0-4
`[^246]`	not 2, 4, or 6
`[^0-9]`	not a number

4 Special characters and shorthands

So if a period is short for "any character," then how do I match a period and nothing else?	Good question! You have to "escape" it with a backslash, like this: `\.`
So `\.org` would only match `.org`, not, say, `borg`?	That's right.
Is it the same for `*` and `+`?	Yes. The period, star, and plus are all special characters with a special meaning. You have to escape them to make them represent the literal character.
So writing `\*borg\.org\+` would only match `*borg.org+` and nothing else?	That's right.
Hey, could I also match `.org` by saying `[.]org`?	Yes! Inside a character class, only `-` (the hyphen) and `^` the caret are special characters.
What if I want to include the hyphen in a character class?	You put it first, since then it can't define a range.
So `[-a-z]+` would match `bye-bye`?	Yes.
And the caret?	You put it anywhere but first, since it only negates if it's the first character.
So `[a-z^]+` would match `o^o`?	Yep.
What about if I want to match all whitespace? Do I make a character class?	You could, or you can just say `\s` to cover spaces, tabs, and the various flavors of newlines.
So `kitty\s*cat` would match both `kittycat` and `kitty cat`?	Exactly.

Are there more shorthands like that?	You bet. Too many to list here.
Is there a shorthand for "all digits"?	Yes! It's `\d`.
So `\d` matches the same thing as `[0-9]`	Exactly.
Is there another way to say `[^0-9]`, too?	Yes, `\D`.
Oh! So is `\S` then "any character except whitespace"?	It is!
Speaking of special characters, does the caret mean anything outside a character class?	Yes, it means "beginning of line."
So `^dog` would find all lines beginning with `dog`?	Exactly.
Is there a character for "end of line," too?	Yes, `$`.

5 Grouping and substitution

Say I want to match `kittykittykitty` as well as `kittykitty`. Can I do that with a plus, like I can match `aa` and `aaa` with `a+`?	Yes! You can make `kitty` a single group by putting it in parentheses.
Like this: `(kitty)+`?	Exactly.
So if I replace `(kitty)+` with `doggie`, do I get `doggiedoggiedoggie`?	No, you get `doggie`, because `(kitty)+` matches the whole string, no matter how many times `kitty` it has.

Oh. So actually to replace `kittykittykitty` with `doggiedoggiedoggie`, I should just replace `kitty` with `doggie`?	Yes.
Can I also say `(kitty|doggie)+` and match *either* `kittykittykitty` and `doggiedoggiedoggie`?	Yes, of course you can. And that will of course also match `kitty` and `doggiedoggie` and so on.
Right. So what if I wanted to replace `kittykitty` with `here-kittykitty` and `doggiedoggiedoggie` with `here-doggiedoggiedoggie`? Can I do that with one expression?	Yes; you can use a backreference. Any grouped expression is saved and numbered and can be accessed by `$1`, `$2`, and so on.
So replacing `That's a cute (kitty)` with `I like that $1` would produce `I like that kitty`?	Absolutely.
So then can I say replace `(kitty|doggie)+` with `here-$1` to get `here-doggiedoggie` and so on?	No, because now the grouped part only has `kitty` or `doggie` once.
Oh, so I'll always get `here-kitty` or `here-doggie` and never `here-doggiedoggie`.	Yes; you have to include the plus in a group.
Would this work: `((kitty|doggie|)+)`?	It would.

Cheat sheet	
`\`	escape character
`\.`	period (literal)
`\s`	any whitespace character (space, tab, newline)
`\d`	any digit
`\S`	any non-whitespace character
`\D`	any non-digit character
`(ab)*`	ab zero or more times
`$1`	`cat` in `kitty(cat)`
`$2`	`s` in `kitty(cat)(s)`
`$1`	`kittycat` in `(kitty(cat))`
`$2`	`cat` in `(kitty(cat))`