Finding strings

A humble introduction to regular expressions

[[User:Daimona Eaytoy]]

SWT Indic Workshop Series
6 December 2020

Base logo in PD-logo, modified with
font “Findet Nemo” by Jens R. Ziehn

Context

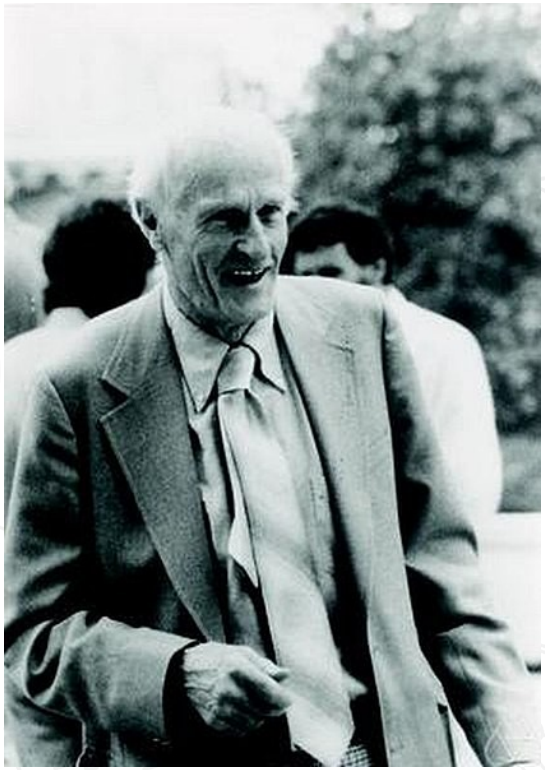
[[User:Camron lutz]], CC-BY-SA 3.0, via Commons

[[User:Tewy]], CC BY-SA 3.0, via Commons

Johan Fredriksson, CC BY-SA 3.0, via Commons

Context

[[User:Camron lutz]], CC-BY-SA 3.0, via Commons



Konrad Jacobs, Erlangen, Copyright is Mathematisches Forschungsinstitut Oberwolfach, CC BY-SA 2.0 DE

"clownfish"

Stephen Kleene
(1909-1994)

Just a quick introduction!

Warm up tasks

You're given a 1000 pages text.

Q1. Find all occurrences of the letter “a”

A1. Open a search tool and search for “a” :-)

WIKIMEDIA
FOUNDATION

Warm up tasks

You're given a 1000 pages text.

Q2. Find all occurrences of “My name is John Doe.”

A2. Do the same with “My name is John Doe.”

WIKIMEDIA
FOUNDATION

Warm up tasks

You're given a 1000 pages text.

Q3. Find all occurrences of the word “cat”

A3. Do the same with “cat”. Right?

Not quite: you'll also find “category”, “catch”, “duplicate”, and hundreds of other words. Not pretty :-[

WIKIMEDIA
FOUNDATION

Warm up tasks

You're given a 1000 pages text.

Q4. Find all occurrences of an email address

A4. WTF?

Enter regular expressions (“regexp”, or “regex”)

Motivation

Being able to search a text for a string that is:

- Incomplete
- Abstract
- Somewhat bound to its surrounding elements

Starting point

When searching “simple” strings, we intuitively think of a bijection:

$$\text{Search string} \iff \text{Strings it can match}$$

That is, if you search “tomato”, the only string you can match is “tomato”.

Starting point

Regular expressions destroy this correspondence, in that a single regular expression can match more than one string (in fact, one to infinity).

WIKIMEDIA
FOUNDATION

A formal aside

Formally, a regular expression is a string describing a *set* of strings. If the string Y is in the set L ("language") of strings matched by the regular expression X, we say that "X matches Y".

As already mentioned, the cardinality of L can be anything from 1 to infinity. You might be wondering what kind of elements can be inside L.

WIKIMEDIA
FOUNDATION

A formal aside

Historically, the elements in L could only be part of the family of so-called “regular languages”. These are quite simple languages, for instance:

1) The empty string (ε)
2) Strings consisting of any number of “x”s
3) Strings consisting of one or more “a”s followed by three or more “b”s

WIKIMEDIA
FOUNDATION

A formal aside

Since Kleene formalized regular expressions in the 1950s, a lot of different regexp engines have been created. Many of these engines have an enriched syntax, and regexps written for these engines can recognize much more than just regular languages (context-free and context-sensitive, IYKWIM, see “Chomsky hierarchy”).

We're going to focus on one of these “new” engines: PCRE.

Dissecting regular expressions

The main point of regular expressions is to allow one single string to match a set of strings. How to do that?

Metacharacters. That is, characters with a special meaning.

Learning regexps is all about learning how to use these metacharacters. Mastering regexps is all about practice.

WIKIMEDIA
FOUNDATION

Dissecting regular expressions

These are the components that we're going to see:

- Atoms
- Quantifiers
- Alternation
- Groups
- Escape character
- Special characters
- Boundaries

Atoms

An atom is a block of characters that are matched together.
The simplest atoms are (groups of) boring, ordinary characters:

Letters, numbers, and several symbols, simply match themselves.
That is, “foobar123” is a valid regexp, but the only string it can match is “foobar123”. Like in ordinary searches. No magic here!

WIKIMEDIA
FOUNDATION

Atoms

Everything can be made an atom by enclosing it in parentheses.
We'll see examples of this later on, as well as more types of atoms.

WIKIMEDIA
FOUNDATION

Dissecting regular expressions

Regular expressions are essentially concatenations of atoms.

Concatenating two or more atoms simply means: “match all of them in the given order”. So “f” matches just “f”, “fo” matches “fo”, etc.

WIKIMEDIA
FOUNDATION

Quantifiers

Quantifiers are used to decide how many times something (= the preceding atom) should be matched.

We're only going to see the most common quantifiers.

Quantifiers

- “*” means “zero or more copies of the preceding atom”
  “x*” will match “”, “x”, “xx”, “xxx”, …
- “+” means “one or more copies of the preceding atom”
  “x+” will match “x”, “xx”, …
- “?” means “zero or one copies of the preceding atom”
  “x?” will match “” and “x”

WIKIMEDIA
FOUNDATION

Quantifiers

Adding a quantifier to an atom creates a new atom: “x” is an atom, and “x+” is a single atom as well.

WIKIMEDIA
FOUNDATION

fo+barz?s*

= “as-is”
= “one or more”
= “zero or one”
= “zero or more”

Matches:

- “fobar”
- “fobarz”
- “fobarzs”
- “foobar”
- “fooooobarzs”
- “foobars”
- “foobarsssss”
- “fooobarzssss”
- …

WIKIMEDIA FOUNDATION

Alternation

Alternation allows matching a single atom from a list.
The pipe character (“|”) is used to alternate atoms.

- “x|y” matches “x” or “y”
- “foo|bar|baz” matches “foo”, “bar”, or “baz”
- “a+|b*|c?” matches one or more “a”s, OR zero or more “b”s, OR zero or one “c”s.

WIKIMEDIA
FOUNDATION

Alternation

When you only want to alternate single characters, there's a compact notation to use: character classes.

A character class is a list of single characters, enclosed in “[...]”. “[abcde]” is the same as “a|b|c|d|e”, and will match one of those letters. Character classes are another kind of atoms.

WIKIMEDIA
FOUNDATION

Alternation

Special syntax for character classes:

- [a-g] matches any letter from a to g; this also works for numbers.
- [^ab123] matches any character that is NOT in the set.

These techniques can be combined: [^a-g] means “anything but a letter in [a-g]”.

WIKIMEDIA
FOUNDATION

Grouping

You can group multiple atoms together, to make the result a single atom. As already mentioned, you do this with parentheses.

The advantage of grouping atoms together is that you can quantify the whole group at once.

WIKIMEDIA
FOUNDATION

Grouping

Example of a group: “(foo|bar)”. This will match “foo” or “bar”, but now you can quantify it! “(foo|bar)+” will match one or more occurrences of “foo” or “bar”: “foo”, “bar”, “foofoo”, “foobar”, …

At each repetition, either of the alternating tokens can be matched.

WIKIMEDIA
FOUNDATION

Escaping

Hermanus Backpackers, CC BY 2.0, via Commons

Escaping

Escaping means: “making a special character lose its meaning”, and is the answer to: “how can I match a literal +, or *, or another special character?”.

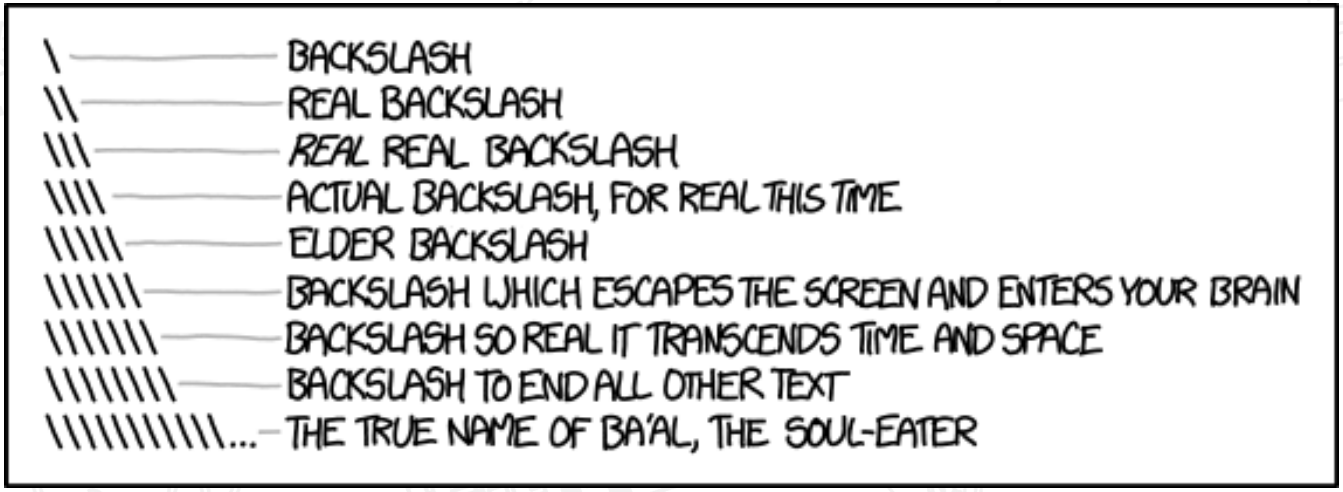
To escape a character, simply prefix it with a backslash (“\”):
“\+” will match “+”, and so on.

WIKIMEDIA
FOUNDATION

Escaping

And what if I want to escape a backslash? “\\” FTW

Which can often lead to ASCII art (“leaning toothpick syndrome”)



WIKIMEDIA FOUNDATION

Randall Munroe, CC-BY-NC 2.5, https://xkcd.com/1638/

“Fake escapes”

Some metacharacters are letters prefixed with a backslash, but they're not escaped letters. Don't let them fool you!

- “\n” means newline
- “\t” means a tab character

We're now going to see in detail a very important example.

WIKIMEDIA
FOUNDATION

Shorthand classes

Alternative notation to express commonly-used character classes.

- “\w” means “[a-zA-Z0-9_]” (“word”)
- “\d” means “[0-9]” (“digit”)
- “\s” means any (white)space (simple space, tab, newline, etc.)

Uppercase versions negate the class, so for instance:

- “\W” means “[^a-zA-Z0-9_]”

WIKIMEDIA
FOUNDATION

The mighty dot

The dot character “.” has a very special meaning: it stands for “match **any** character except newlines”.

There's an option to make it match newlines, but let's ignore that.

The dot might seem useful, but you usually want to be more specific to avoid matching unintended characters.

WIKIMEDIA
FOUNDATION

Boundaries

Boundaries are metacharacters that allow making assertions about the position of the match inside the string.

They don't match any character on their own, precisely because you only use them to make assertions.

WIKIMEDIA
FOUNDATION

Boundaries

- “\b” is a word boundary. It requires a letter (“\w”) to be on one side, and a non-letter (“\W”) on the other.

  “\bfoo\b” will match “foo”, but not “foobar” or “barfoo”

Often a no-brainer when you're looking for specific words.

WIKIMEDIA
FOUNDATION

Boundaries

- “^” matches the beginning of the string
  “^cat” matches “cat” in “cats are great”, but not in “my cat is great”

- “$” matches the end of the string
  “cats$” matches “cats” in “I love cats”, but not in “cats love me”

These are also called “anchors”.

WIKIMEDIA
FOUNDATION

One final word: notation

So far, we've been writing regular expressions just like ordinary strings, enclosed in quotes:

“I (like|love) ca+ts”

This is easy to read, and it's also how you write regexps for AbuseFilter.

One final word: notation

However, the standard notation for regular expressions is to put a slash on each side:

I (like|love) ca+ts

WIKIMEDIA FOUNDATION

Raph_PH, CC BY 2.0, via Commons

One final word: notation

However, the standard notation for regular expressions is to put a slash on each side:

/I (like|love) ca+ts/

WIKIMEDIA
FOUNDATION

One final word for real: modifiers

When using the “slash syntax”, you can set some engine options for a regexp, called “modifiers”. You do this by putting a special letter after the ending slash.

- /cat/i → case insensitive (matches “CAT”, “Cat”, etc.)
- /^cat$/m → multiline, ^ and $ match the end of a line, not of the whole string
- /.+/s → single-line, dot matches newline

Examples

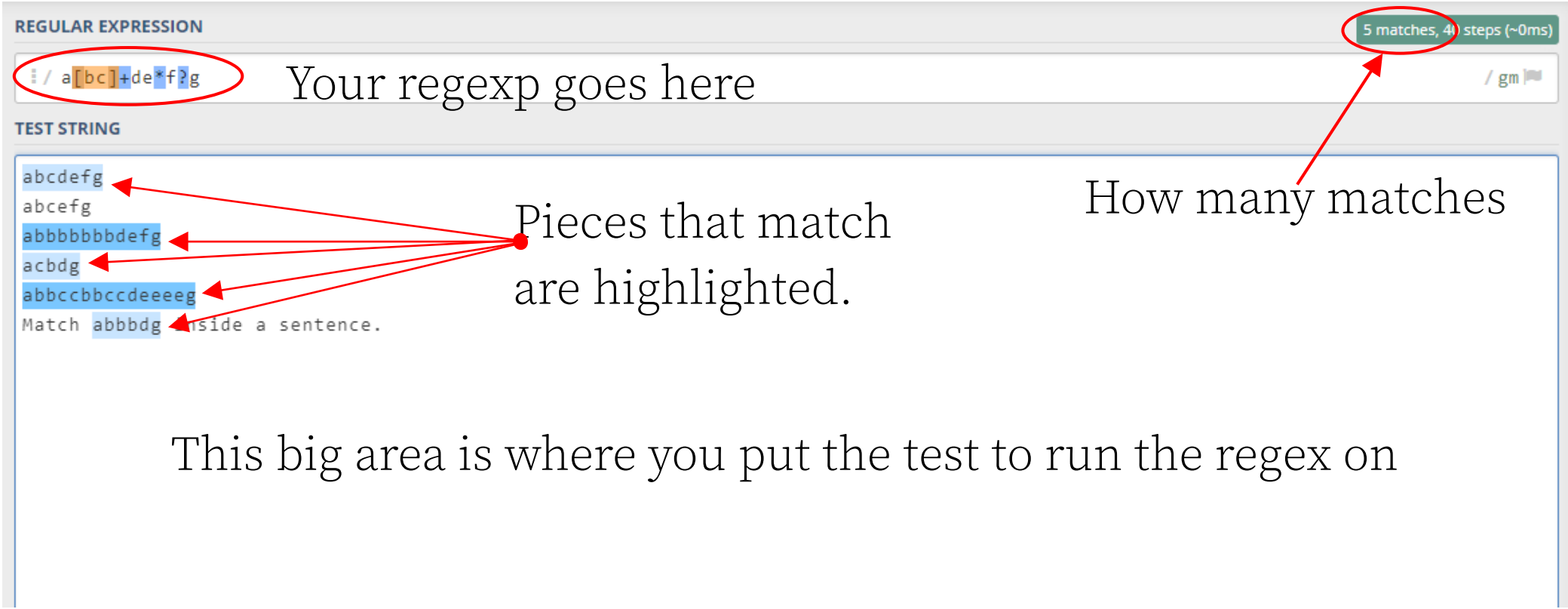
How to test a regexp

There are various sites that allow testing a regexp interactively.

My suggestion is regex101.com

The interface is very intuitive, and gives you detailed information about the regular expression.

How to test a regexp



Cheat sheet

Google “regex cheat sheet” for helpful tables with common regex syntax.

Many of those are great, I suggest choosing one of the small ones, as the others will contain a lot of things that we didn't talk about.

Task 1

“A window to the past”: write a regexp that matches the word “cat”.

To test your proposal, head to regex101, then try writing a regexp in the upper field, and some examples in the big text field. You'll want to ensure that the regexp will match all and every string that you want to match.

WIKIMEDIA
FOUNDATION

Task 1

“A window to the past”: write a regexp that matches the word “cat”.

For this example, you can use this text:

cat, category, scattered, duplicate, catch, copycat

Only the first “cat” should match.

Task 1

“A window to the past”: write a regexp that matches the word “cat”.

Answer: \bcat\b

Remember: word boundaries (“\b”) help you match single words.

WIKIMEDIA
FOUNDATION

Task 2

“Another window to the past”: write a regexp that matches emails. You can assume that “email” means: any letter, number, dot, dash, underscore, followed by “@”, followed by letters, followed by a dot, followed by letters.
Test it at https://bit.ly/3pQZ81F (points to regex101)

WIKIMEDIA
FOUNDATION

Task 2

“Another window to the past”: write a regexp that matches emails. You can assume that “email” means: any letter, number, dot, dash, underscore, followed by “@”, followed by letters, followed by a dot, followed by letters.
Test it at https://bit.ly/3pQZ81F (points to regex101)

Answer: [a-z0-9._-]+@[a-z]+\.[a-z]+

WIKIMEDIA
FOUNDATION

Task 3

“Helping Marlin”: write a regexp that matches an address. That is:

- an optional name followed by comma and space (in the form “P. Sherman, ”);
- a number optionally followed by a capital letter;
- one or more words starting with a capital letter;
- a comma, a space;
- one or two words starting with capital letters.

Test it at https://bit.ly/3fjbxa3

Task 3

“Helping Marlin”: write a regexp that matches an address.

Answer: ([A-Z]\. [A-Z][a-z]+,)?\d+[A-Z]?
([A-Z][a-z]+)+, [A-Z][a-z]+([A-Z][a-z]+)?

The task was a bit difficult, and the solution might seem complicated, but dissecting it should help.

WIKIMEDIA
FOUNDATION

Task 4

“WP:HERE”: write a regexp that matches wikilinks where the page title contains “(,” but the piped part doesn't.

Test it at https://bit.ly/3pPRRPV

WIKIMEDIA
FOUNDATION

Task 4

“WP:HERE”: write a regexp that matches wikilinks where the page title contains “(,” but the piped part doesn't.

Test it at https://bit.ly/3pPRRPV

Answer: `\[\[[^\]|]*\([^\]|]*\|[^(\]]*]]`

WIKIMEDIA
FOUNDATION

Task 5

“Redirect”: write a regexp that matches redirects.

Test it at https://bit.ly/3q1Yfnk

Task 5

“Redirect”: write a regexp that matches redirects.

Test it at https://bit.ly/3q1Yfnk

Answer: #\s*redirect\s*\[\[

WIKIMEDIA
FOUNDATION

Want more?

There are lots of things that we didn't cover. The internet is full of good tutorials. Some suggestions:

- https://regexone.com/ - Interactive tutorial with exercises;
- https://regexcrossword.com/ - Crosswords-like brain teasers where you need to write text that matches two or more regexp at the same time;
- https://alf.nu/RegexGolf - Hard exercises where you have to write the shortest regex possible to match the given text. Finding a suitable regexp is of intermediate difficulty, making it shorter is very hard.

WIKIMEDIA
FOUNDATION

THANK YOU

Appendix A - Things we didn't cover

You can google these; they're ordered from easier to harder {{cn}}

- “Range quantifier” (a{1,3})
- Capturing vs non-capturing groups
  - Backreferences
  - Search & replace
- “Lookarounds” (lookahead and lookbehind)
- Greedy vs lazy vs possessive quantifiers
- Backtracking (and why “catastrophic backtracking” can be a problem)
- Recursive and conditional patterns
- Control verbs, other inline engine directives

Appendix B - Non-regular languages

- Balanced parentheses: “(()((((()))(()))” - context-free language. Matched by \(([^()]+|(?R))*\)
- Squares: “wikiwiki”, “papa” - context-sensitive language (not context-free). Matched by ^(.+)\1$
- Language described by “$a^n b^n c^n$” - context-sensitive (regexps for this exist in the wild)
- Language described by {“a^p” : p is prime} - context-sensitive. Finding a regular expression for this is left as an exercise to the reader

WIKIMEDIA
FOUNDATION

Appendix C - Curious applications

Some crazy things to do with regular expressions:

- Find primes: using unary representation (2 = 11, 3 = 111, etc.), prime numbers will be matched by ^1?$|^(11+?)\1+$

  https://www.noulakaz.net/2007/03/18/a-regular-expression-to-check-for-prime-numbers/
- Divide a number by $\sqrt{2}$ https://codegolf.stackexchange.com/a/198428
- Match Fibonacci numbers

  https://doc.qt.io/archives/qq/qq01-seriously-weird-qregexp.html
- A surprising use case: https://www.youtube.com/watch?v=ub82Xb1C8os

WIKIMEDIA
FOUNDATION