A crash course regular expressions

Professor Hossein Saiedian

EECS 348: Software Engineering

Fall 2023

THE UNIVERSITY OF KANSAS

What is a regular expression

- A regular expression (regex) describes a pattern of text
  - For matching
  - To search and replace
  - An example: `^a...s$`
- Where to use
  - Text editors (vim)
  - Command line: Linux/Unix (with grep, sed, find, …)
  - Languages: JavaScript, Python, Perl, …

Some basic regexes and meta characters

/pattern

. to match any character

\ starts an escape sequence, for example, \. to match a dot

^ matches the beginning of a line; $ the end

| means or

( ) is used for grouping

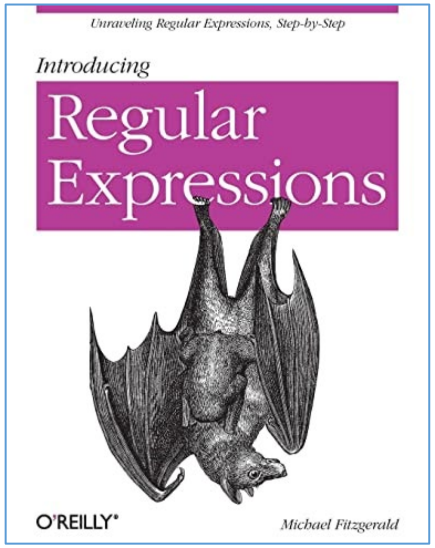
[ ] group characters into a character set

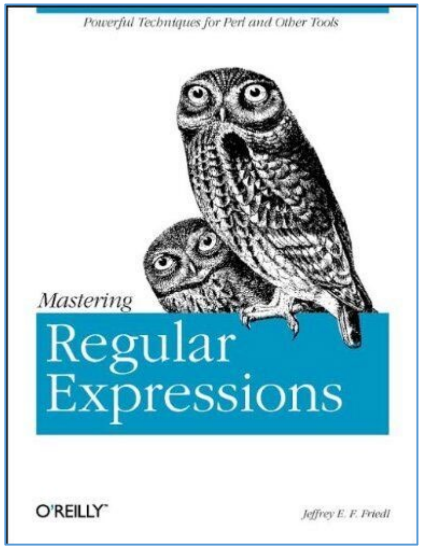
[a-e] a range of characters or numbers [1-4]

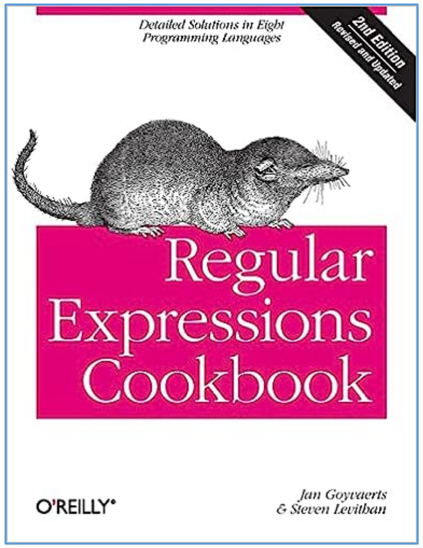
^ at the start of a bracket means any char except these; for example, [^0-9] means any non-digit character

Regular expressions in Unix

- Extremely powerful for pattern matching (vim, grep, sed, . . .)

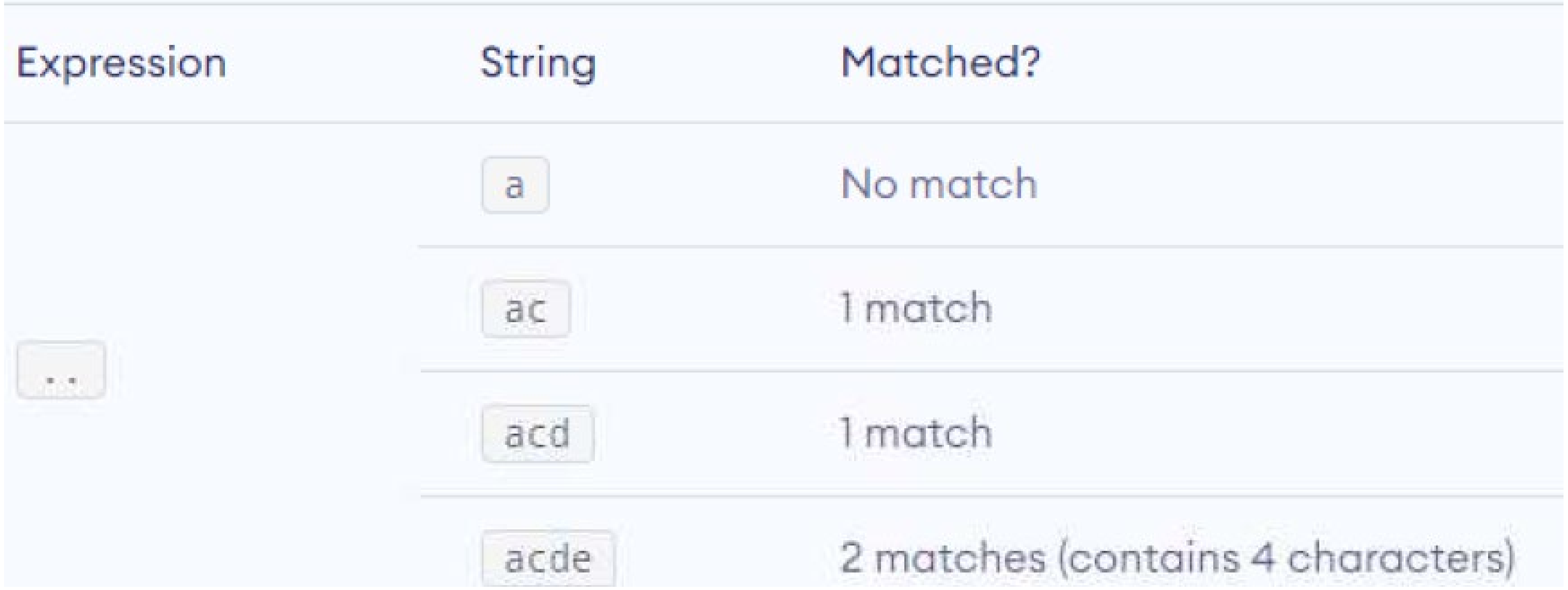





Detailed Solutions in Eight Programming Languages
2nd Edition
Revised and Updated
Regular Expressions Cookbook
O'REILLY
Jan Goyvaerts
& Steven Levithan

Some random examples

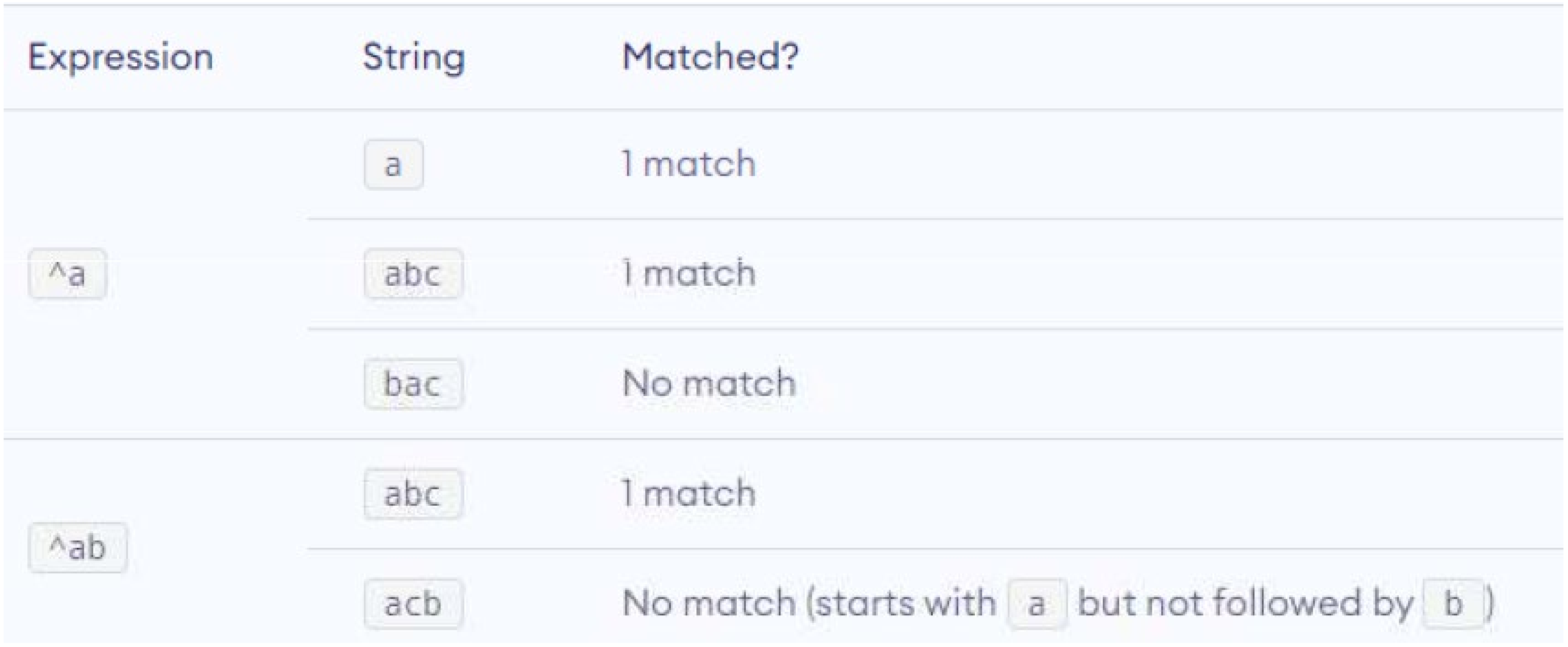
- A . matches any single character



Expression	String	Matched?
..	a	No match
	ac	1 match
	acd	1 match
	acde	2 matches (contains 4 characters)

Some random examples

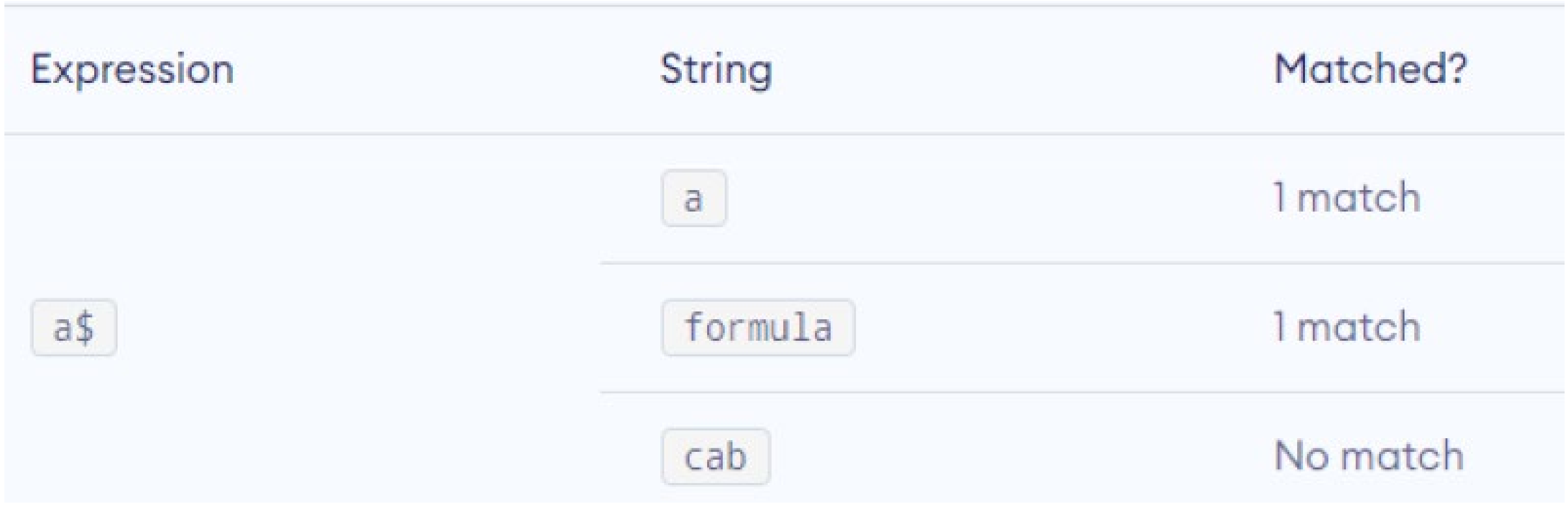
- A ^ is used to check if a string starts with a certain



Expression	String	Matched?
`^a`	`a`	1 match
	`abc`	1 match
	`bac`	No match
`^ab`	`abc`	1 match
	`acb`	No match (starts with `a` but not followed by `b`)

Some random examples

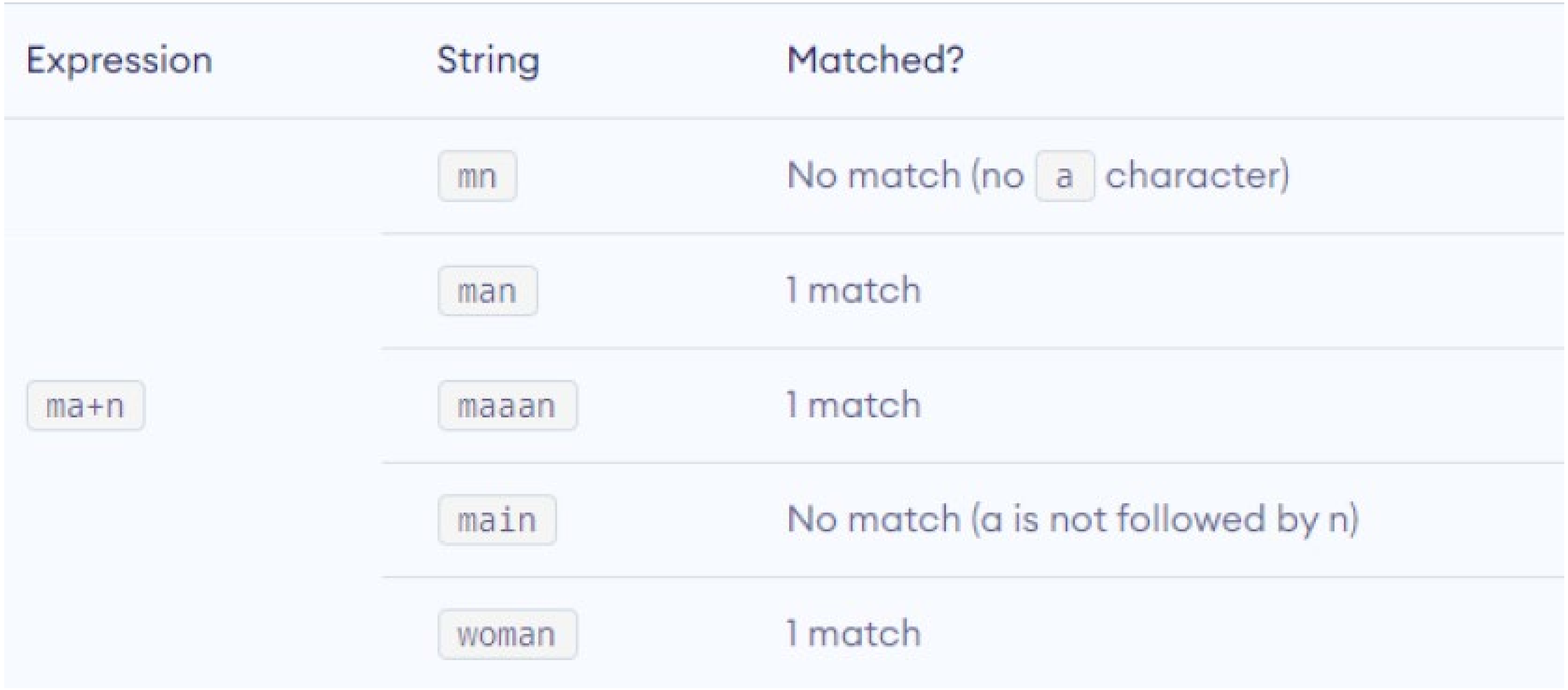
- A $ is used to check if a string ends with a certain character



Expression	String	Matched?
`a$`	`a`	1 match
	`formula`	1 match
	`cab`	No match

Some random examples

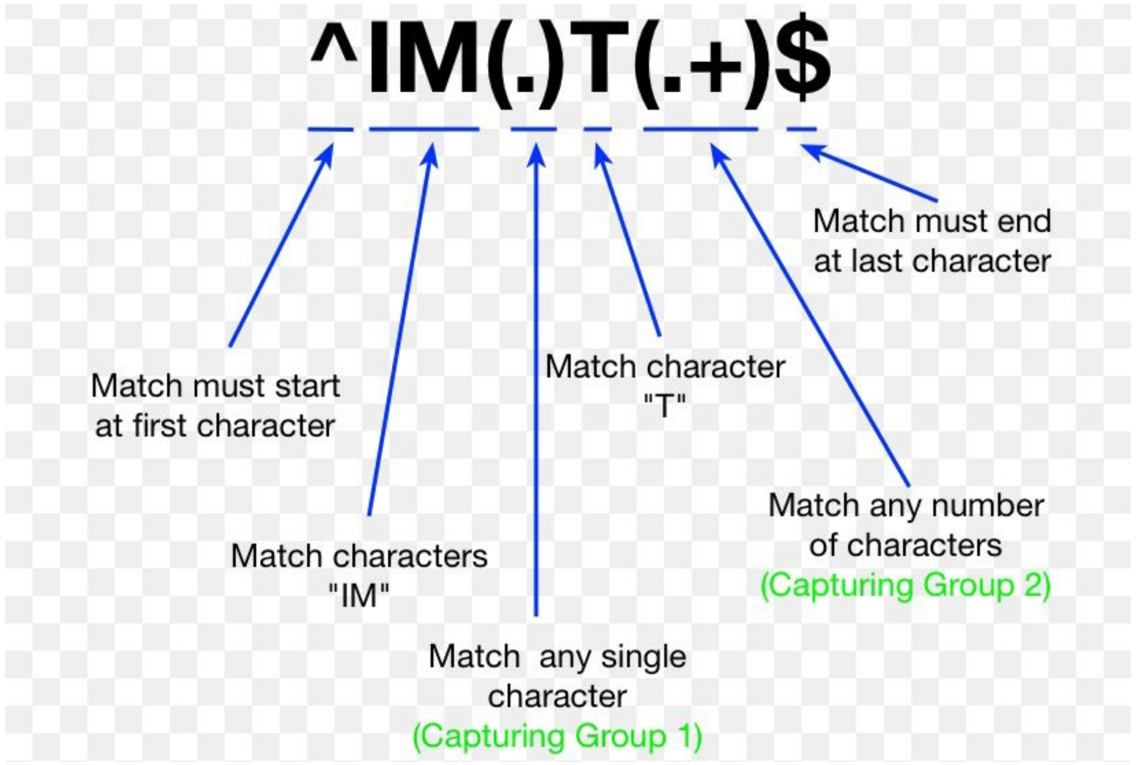
- A + matches one or more occurrences of the pattern left to it



Expression	String	Matched?
`ma+n`	`mn`	No match (no `a` character)
	`man`	1 match
	`maaan`	1 match
	`main`	No match (a is not followed by n)
	`woman`	1 match

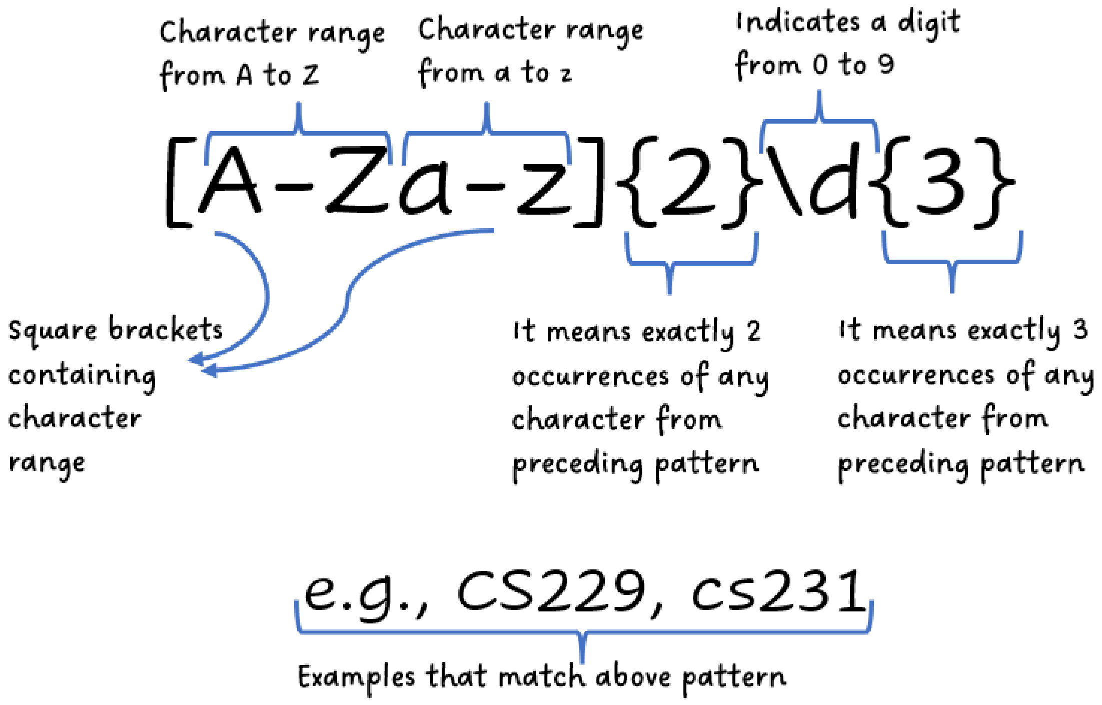
Regular expressions in Unix

- A very similar concept, slightly different notation
- Extremely powerful for pattern matching (vim, grep, sed, . . .)


^IM(.)T(.+)$
Match must end at last character
Match must start at first character
Match character "T"
Match any number of characters (Capturing Group 2)
Match characters "IM"
Match any single character (Capturing Group 1)

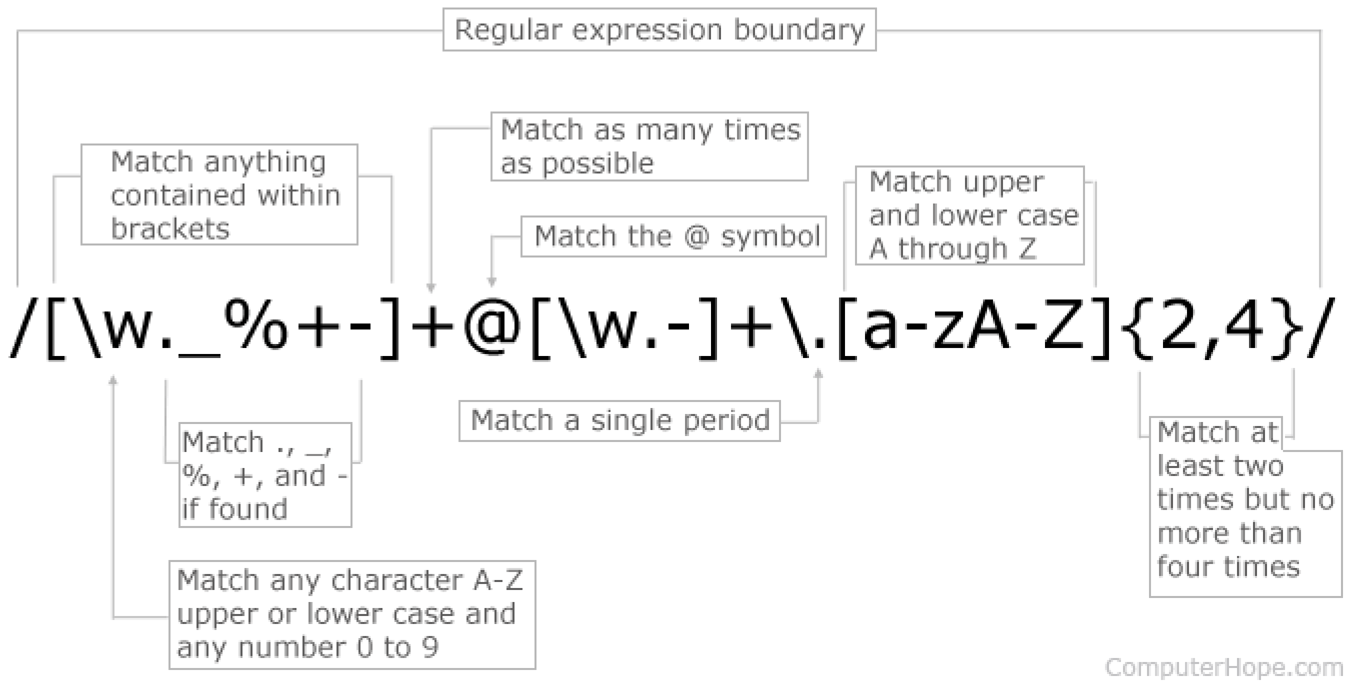
Regular expressions in Unix

- Extremely powerful for pattern matching (vim, grep, sed, …)


Character range from A to Z
Character range from a to z
Indicates a digit from 0 to 9
[A-Za-z]{2}\d{3}
Square brackets containing character range
It means exactly 2 occurrences of any character from preceding pattern
It means exactly 3 occurrences of any character from preceding pattern
e.g., CS229, cs231
Examples that match above pattern

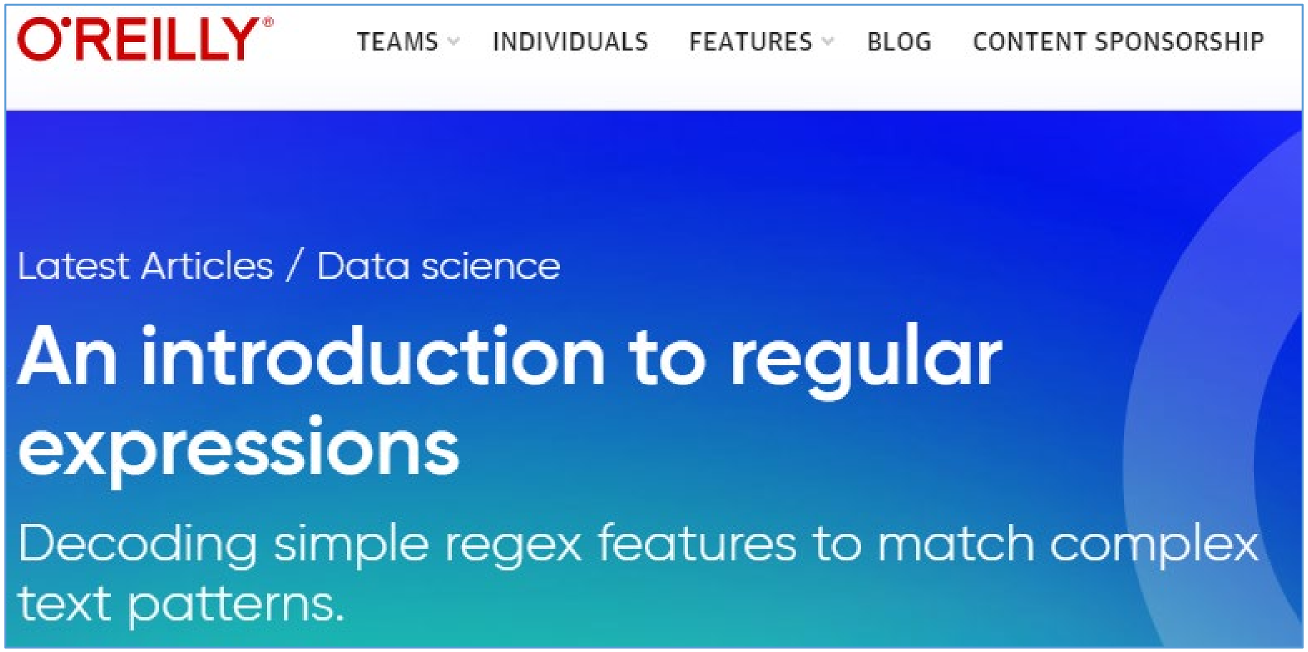
Regular expressions in Unix

- Extremely powerful for pattern matching (vim, grep, sed, …)


Regular expression boundary
Match anything contained within brackets
Match as many times as possible
Match the @ symbol
Match upper and lower case A through Z
/[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}/
Match ., _, %, +, and - if found
Match a single period
Match at least two times but no more than four times
Match any character A-Z upper or lower case and any number 0 to 9
ComputerHope.com

Regular expressions in Unix

- Extremely powerful for pattern matching (vim, grep, sed, ...)



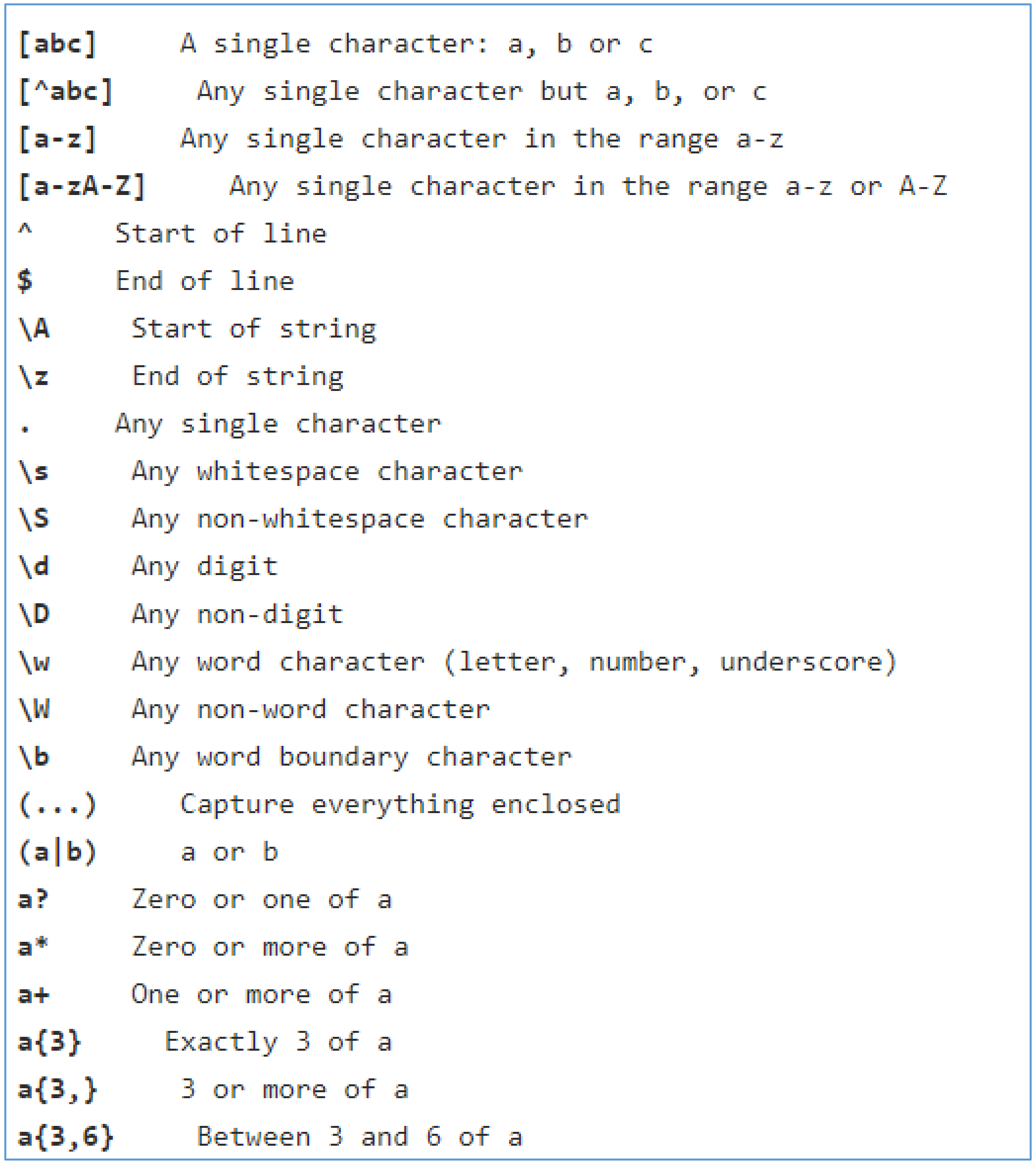
Latest Articles / Data science

An introduction to regular expressions

Decoding simple regex features to match complex text patterns.

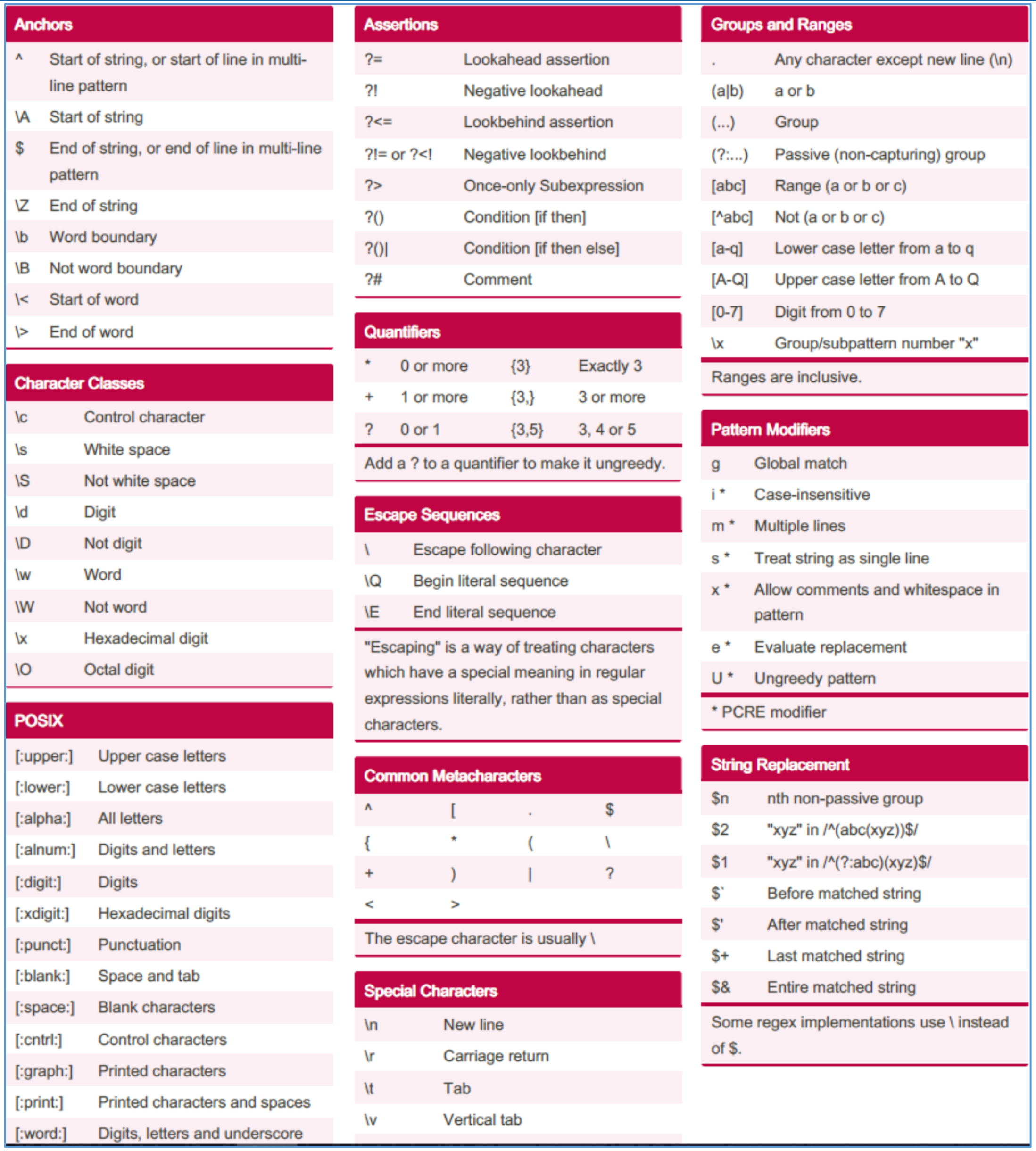
https://www.oreilly.com/content/an-introduction-to-regular-expressions

A short RE cheat sheet



[abc]	A single character: a, b or c
[^abc]	Any single character but a, b, or c
[a-z]	Any single character in the range a-z
[a-zA-Z]	Any single character in the range a-z or A-Z
^	Start of line
$	End of line
\A	Start of string
\z	End of string
.	Any single character
\s	Any whitespace character
\S	Any non-whitespace character
\d	Any digit
\D	Any non-digit
\w	Any word character (letter, number, underscore)
\W	Any non-word character
\b	Any word boundary character
(...)	Capture everything enclosed
(a|b)	a or b
a?	Zero or one of a
a*	Zero or more of a
a+	One or more of a
a{3}	Exactly 3 of a
a{3,}	3 or more of a
a{3,6}	Between 3 and 6 of a

A longer RE cheat sheet



Anchors

^	Start of string, or start of line in multi-line pattern
\A	Start of string
$	End of string, or end of line in multi-line pattern
\Z	End of string
\b	Word boundary
\B	Not word boundary
\<	Start of word
\>	End of word

Character Classes

\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\x	Hexadecimal digit
\O	Octal digit

POSIX

[:upper:]	Upper case letters
[:lower:]	Lower case letters
[:alpha:]	All letters
[:alnum:]	Digits and letters
[:digit:]	Digits
[:xdigit:]	Hexadecimal digits
[:punct:]	Punctuation
[:blank:]	Space and tab
[:space:]	Blank characters
[:cntrl:]	Control characters
[:graph:]	Printed characters
[:print:]	Printed characters and spaces
[:word:]	Digits, letters and underscore

Assertions

?=	Lookahead assertion
?!	Negative lookahead
?<=	Lookbehind assertion
?!= or ?<!	Negative lookbehind
?>	Once-only Subexpression
?()	Condition [if then]
?()|	Condition [if then else]
?#	Comment

Quantifiers

*	0 or more	{3}	Exactly 3
+	1 or more	{3,}	3 or more
?	0 or 1	{3,5}	3, 4 or 5

Add a ? to a quantifier to make it ungreedy.

Escape Sequences

\	Escape following character
\Q	Begin literal sequence
\E	End literal sequence

"Escaping" is a way of treating characters which have a special meaning in regular expressions literally, rather than as special characters.

Common Metacharacters

^	[	.	$
{	*	(	\
+	)	|	?
<	>		

The escape character is usually \

Special Characters

\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab

Groups and Ranges

.	Any character except new line (\n)
(a|b)	a or b
(...)	Group
(?:...)	Passive (non-capturing) group
[abc]	Range (a or b or c)
[^abc]	Not (a or b or c)
[a-q]	Lower case letter from a to q
[A-Q]	Upper case letter from A to Q
[0-7]	Digit from 0 to 7
\x	Group/subpattern number "x"

Ranges are inclusive.

Pattern Modifiers

g	Global match
i *	Case-insensitive
m *	Multiple lines
s *	Treat string as single line
x *	Allow comments and whitespace in pattern
e *	Evaluate replacement
U *	Ungreedy pattern

* PCRE modifier

String Replacement

$n	nth non-passive group
$2	"xyz" in /^(abc(xyz))$/
$1	"xyz" in /^(?:abc)(xyz)$/
$`	Before matched string
$'	After matched string
$+	Last matched string
$&	Entire matched string

Some regex implementations use \ instead of $.

Regular expressions in Python

- Python has a module named **re** to work
- To use it, need to import the module

```
import re
```

Summary

- It's all about patterns and pattern matching
  - Reserved characters: . * ? + ( ) [ ] { } / \ |
  - Repetition operators specify a recurring pattern
  - Some characters have special meanings based on their position in the expression
- A strong relationship with regular grammars used informal language theory and in compiler design
- Some dialects (minor differences)
- In vi or vim (the popular Unix/Linux editor)

  :g/re/p
  - Means to do a global match of all lines that match a regular expression and print those lines

Additional resources

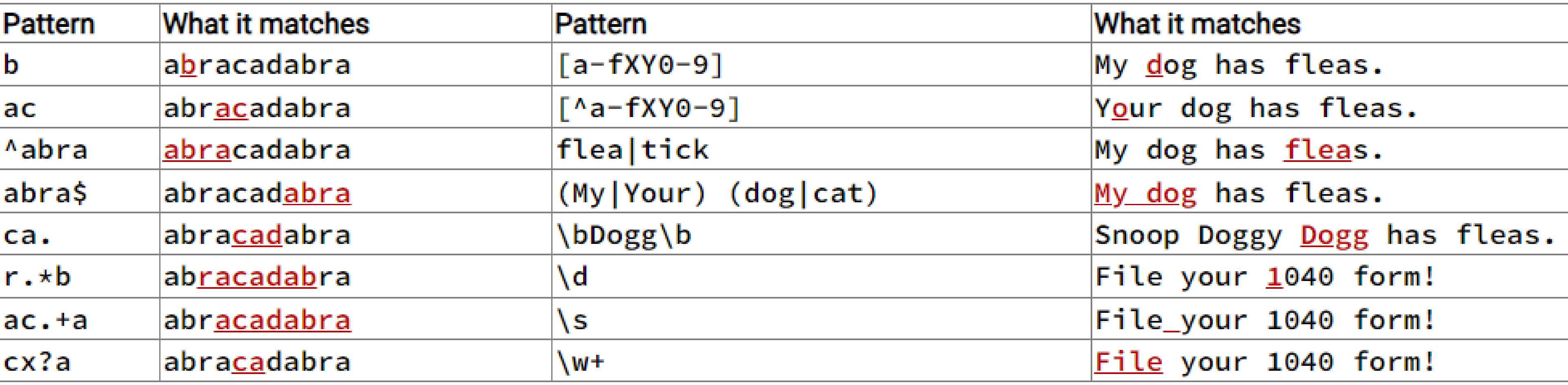
- Components of regular expressions

What	Description	What	Description
`.`	any one char but \n	`|`	alternation
`[a-fxy0-9]`	any *one* of these	`(…)`	grouping
`[^a-fxy0-9]`	any char *but* one of these	`\b`	word boundary
`*`	0–∞ of previous (any number)	`\d` or `\D`	`[0-9]` or not (just one char)
`+`	1–∞ of previous (many)	`\s` or `\S`	`[  \n\r…]` or not (just one char)
`?`	0–1 of previous (optional)	`\w` or `\W`	`[0-9a-zA-Z_]` or not (just one char)
`{17}`	17 of previous	`^`	beginning of line
`{3,8}`	3–8 of previous	`$`	end of line

https://www.cs.colostate.edu/~cs253/Spring20/Lecture/RegularExpressions

Additional resources

- Examples



Pattern	What it matches	Pattern	What it matches
`b`	`abracadabra`	`[a-fXY0-9]`	`My dog has fleas.`
`ac`	`abracadabra`	`[^a-fXY0-9]`	`Your dog has fleas.`
`^abra`	`abracadabra`	`flea|tick`	`My dog has fleas.`
`abra$`	`abracadabra`	`(My|Your) (dog|cat)`	`My dog has fleas.`
`ca.`	`abracadabra`	`\bDogg\b`	`Snoop Doggy Dogg has fleas.`
`r.*b`	`abracadabra`	`\d`	`File your 1040 form!`
`ac.+a`	`abracadabra`	`\s`	`File your 1040 form!`
`cx?a`	`abracadabra`	`\w+`	`File your 1040 form!`

https://www.cs.colostate.edu/~cs253/Spring20/Lecture/RegularExpressions