Python RegEx

Petr Svarny, 2020

Regular expressions

- Language for effective search in text
- Very compact
- Various dialects (Perl, Python, Java, etc.)

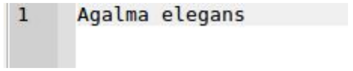
- Great sites for testing and trying:
  regex101
  regexr

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.
PERL!
TAP TAP

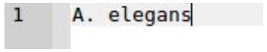
http://xkcd.com/208/

Shorten name of organisms

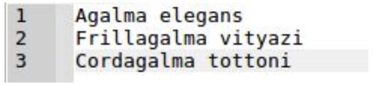
Agalma elegans -> A. elegans



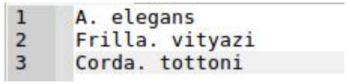
```
Agalma elegans
```



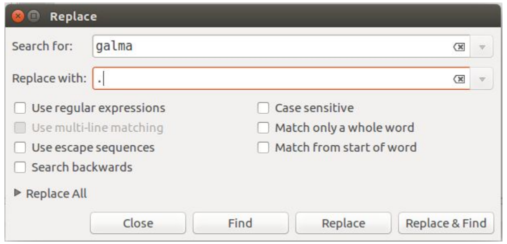
```
A. elegans
```



```
Agalma elegans
Frillagalma vityazi
Cordagalma tottoni
```



```
A. elegans
Frilla. vityazi
Corda. tottoni
```



Replace

Search for: galma

Replace with: .

- [ ] Use regular expressions
- [ ] Use multi-line matching
- [ ] Use escape sequences
- [ ] Search backwards
- [ ] Case sensitive
- [ ] Match only a whole word
- [ ] Match from start of word

From [Practical computing for biologists](#)

Wildcards

- Regex uses wildcards to broaden search
- Special meaning symbols in regular expression: .?[ ]*
- They need to be ‘escaped’ by backslash if you search them
  e.g.: * -> * ? -> \?
- Some letters have special meaning, used with backslash,
  e.g.: \w \d \t \n \s

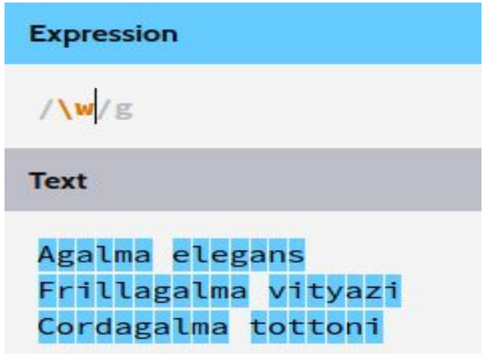
Wildcards

- . - any symbol
- a*b - a can be 0 or multiple times (b, ab, aab, aaaaaab)
- a+b - a can be 1 or multiple times (ab, aab, aaaaaab)
- a?b - a can be 0 or 1 time (b, ab)

Wildcards

- \w - word character is a character from a-z, A-Z, 0-9, including the _ (underscore) [A-Za-z0-9_]
- \d - all digits
- \s - whitespace character (space, a tab, a line break, or a form feed) [\t\r\n\f]
- \b - allows you to perform a "whole words only" search using a regular expression in the form of \bword\b

\w - letters (A-z), numbers (0-9) and _

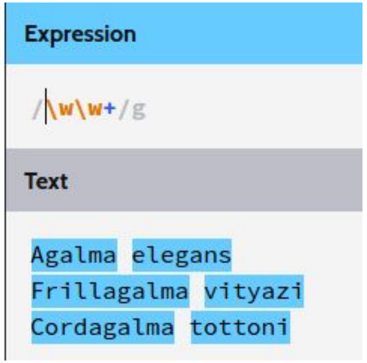


Expression

```
/\w/g
```

Text

```
Agalma elegans
Frillagalma vityazi
Cordagalma tottoni
```



Expression

```
/\w\w+/g
```

Text

```
Agalma elegans
Frillagalma vityazi
Cordagalma tottoni
```

http://regexr.com/

Capturing text using parentheses ()

Original text		Modified text
5th	→	Position: 5
3rd		Position: 3
2nd		Position: 2
4th		Position: 4
1st		Position: 1

Capturing text using parentheses ()

Original text	Modified text
5th	Position: 5
3rd	Position: 3
2nd	Position: 2
4th	Position: 4
1st	Position: 1

```
(\w)\w\w
```

First group

```
Position: $1
Position: \1
```

Capturing text using parentheses ()

Original text

```
Agalma elegans
Frillagalma vityazi
Cordagalma tottoni
```

Modified text

```
Agalma elegans A_elegans
Frillagalma vityazi F_vityazi
Cordagalma tottoni C_tottoni
```

Capturing text using parentheses ()

Original text

```
Agalma elegans
Frillagalma vityazi
Cordagalma tottoni
(\w)(\w+) (\w+)
```

Modified text

```
Agalma elegans A_elegans
Frillagalma vityazi F_vityazi
Cordagalma tottoni C_tottoni
\1\2 \3 \1_\3
```

Exercise

- In your text editor, replace original text
- Write down the regex that you used

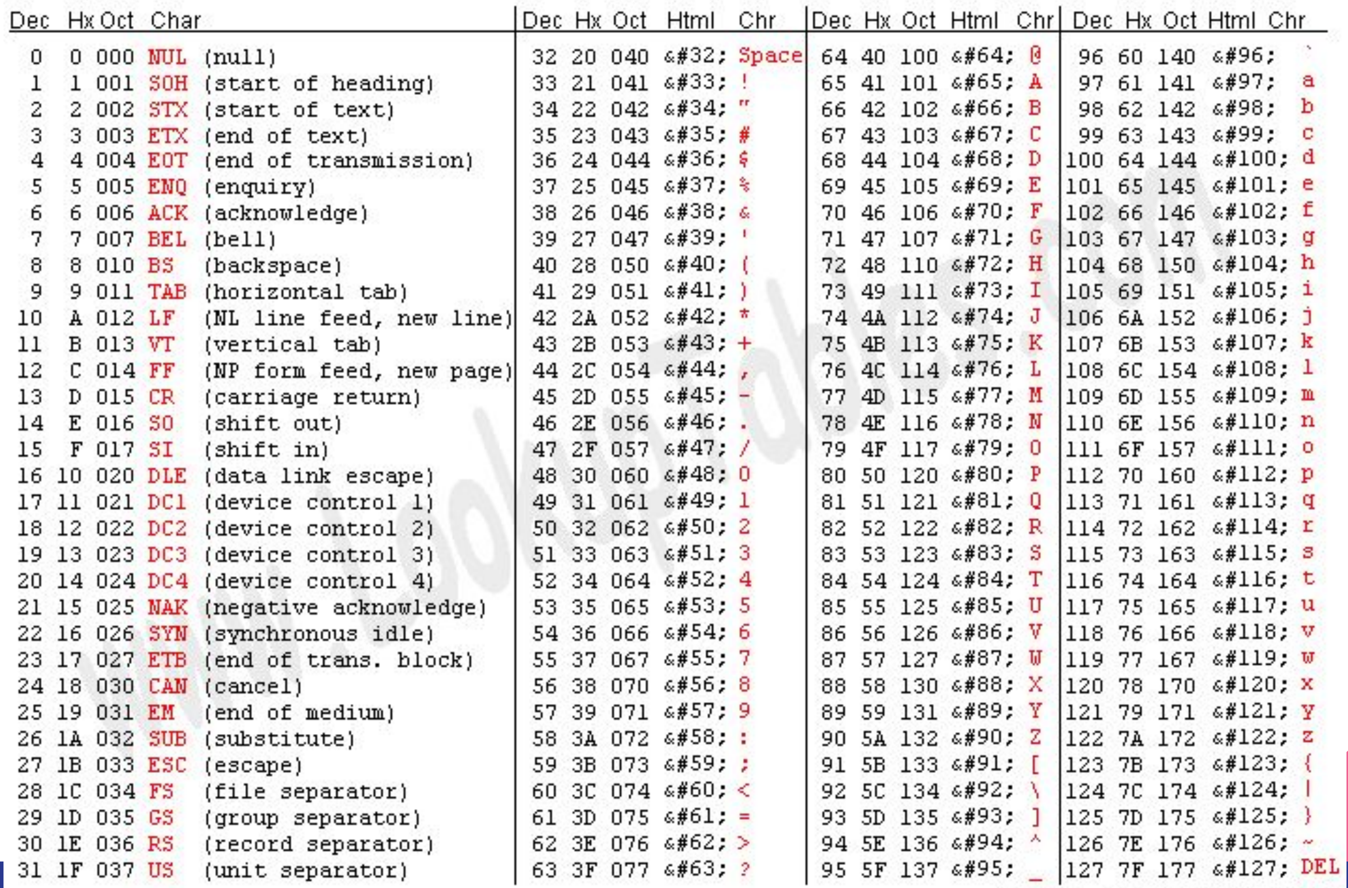
Original text

```
Agalma elegans
Frillagalma vityazi
Cordagalma tottoni
```

Modified text

```
A. elegans
F. vityazi
C. tottoni
```

List of symbols - [xyz]



Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX	(end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT	(end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ	(enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK	(acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL	(bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS	(backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB	(horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT	(vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR	(carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO	(shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI	(shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE	(data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1	(device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2	(device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3	(device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4	(device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN	(synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB	(end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN	(cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM	(end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB	(substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC	(escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS	(file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS	(group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS	(record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US	(unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

List of symbols

```
\w = [A-Za-z_0-9]
```

```
\w does work only in some dialects
```

You can create your own groups of symbols

```
[YN]
```

List of symbols

```
([A-Z]+)([0-9]+)(o?)
```

```
A1
A2
A3
A1o
A2o
A3o
S1
S2
S3
PP1o
PP2o
PP3o
```

$\rightarrow$

\1	\2	\3
material	replicate	variant
A	1	
A	2	
A	3	
A	1	o
A	2	o
A	3	o
S	1	
S	2	
S	3	
PP	1	o
PP	2	o
PP	3	o

Repetition- x{3,9}

A{X} A is repeated exactly X times

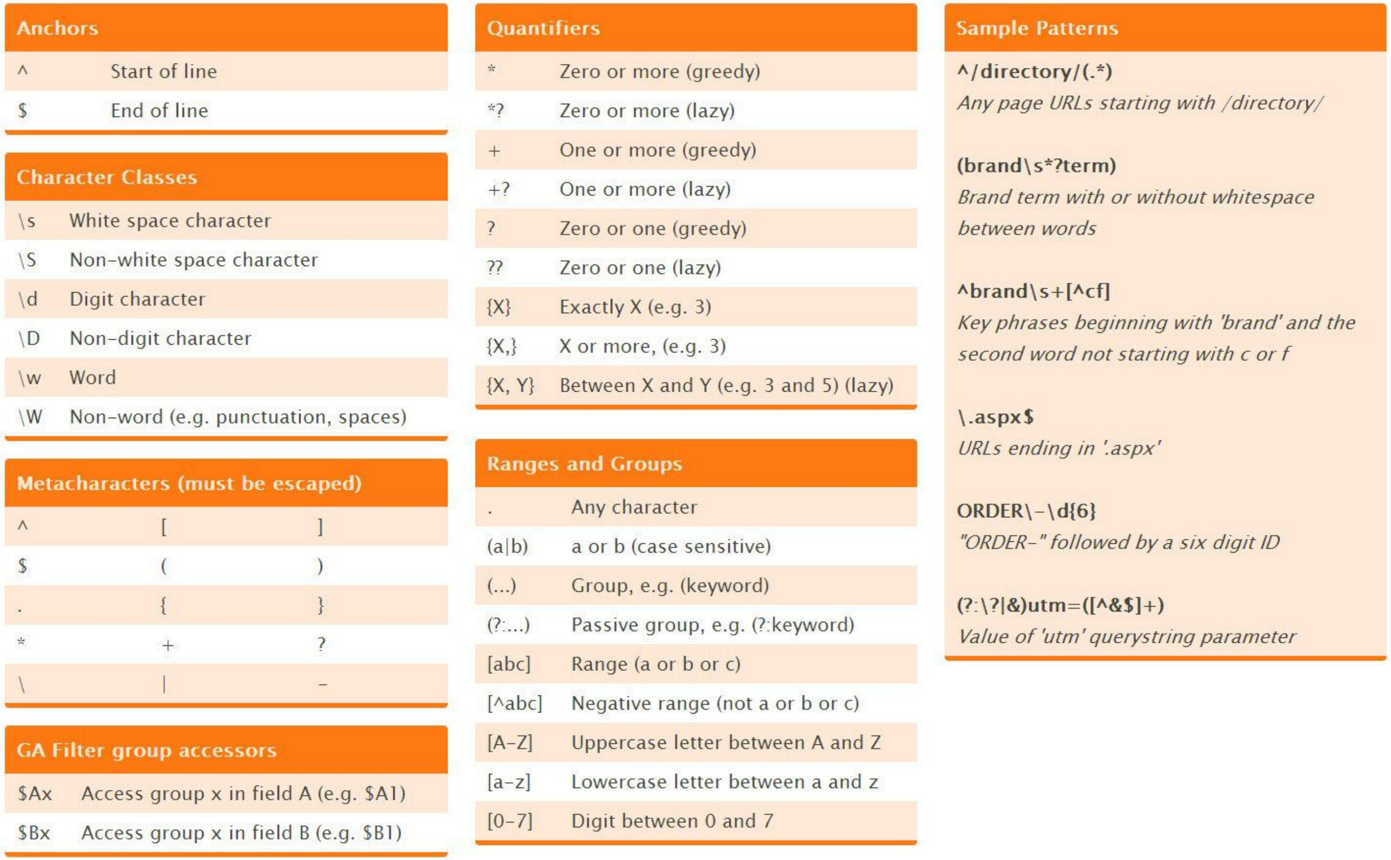
A{X,Y} A is repeated between X and Y times

A{X,} A is repeated X times or more

```
TACAACAGCAGCAGCAGCAGCAGCAGCAGCAGCAGGGGAC
(CAG){3}                CAGCAGCAG
(CAG){1,3}          CAGCAGCAG
CAG{3}                  CAGGG
```

Exercise

- Using your text editor, find using regex and count in the Zen of Python (e.g., import this)
  - Lines where there is letter a or o before . (dot)
  - Alphanumeric symbols
  - T letter (uppercase and lowercase) is repeated two times
  - T (uppercase and lowercase) letter is repeated 1 to 2 times



Anchors

^	Start of line
$	End of line

Character Classes

\s	White space character
\S	Non-white space character
\d	Digit character
\D	Non-digit character
\w	Word
\W	Non-word (e.g. punctuation, spaces)

Metacharacters (must be escaped)

^	[	]
$	(	)
.	{	}
*	+	?
\	|	-

GA Filter group accessors

$Ax	Access group x in field A (e.g. $A1)
$Bx	Access group x in field B (e.g. $B1)

Quantifiers

*	Zero or more (greedy)
*?	Zero or more (lazy)
+	One or more (greedy)
+?	One or more (lazy)
?	Zero or one (greedy)
??	Zero or one (lazy)
{X}	Exactly X (e.g. 3)
{X,}	X or more, (e.g. 3)
{X, Y}	Between X and Y (e.g. 3 and 5) (lazy)

Ranges and Groups

.	Any character
(a|b)	a or b (case sensitive)
(...)	Group, e.g. (keyword)
(?:...)	Passive group, e.g. (?:keyword)
[abc]	Range (a or b or c)
[^abc]	Negative range (not a or b or c)
[A-Z]	Uppercase letter between A and Z
[a-z]	Lowercase letter between a and z
[0-7]	Digit between 0 and 7

Sample Patterns

^/directory/(.*)
Any page URLs starting with /directory/

(brand\s*?term)
Brand term with or without whitespace between words

^brand\s+[^cf]
Key phrases beginning with 'brand' and the second word not starting with c or f

\.aspx$
URLs ending in '.aspx'

ORDER\-\d{6}
"ORDER-" followed by a six digit ID

(?:\?|&)utm=([^&$]+)
Value of 'utm' querystring parameter

Cheatography

Learning tip when facing a Cheatsheet

Don’t try to memorize it as a whole.

Try to keep it at hand for use, you will learn what you need with time.

Memorize maybe one or two that seem the most useful for you at the moment.

Regex in python

- Using built-in re library
- re.compile
  - Compile a regular expression pattern into a regular expression object
- re.search
  - Takes a regular expression pattern and a string and searches for that pattern within the string, finds first occurence
- group method is used to extract groups that matched
- re.findall
  - Finds ALL the matches and returns them as a list of strings

Regex in python - basic patterns

- a, X, 9 -- ordinary characters just match themselves exactly.
- . -- matches any single character except newline '\n'
- \w -- (lowercase w) matches a "word" character: a letter or digit or underbar [a-zA-Z0-9_]
- \W -- matches any non-word character.
- \b -- boundary between word and non-word
- \s -- matches a single whitespace character
- \S -- matches any non-whitespace character
- \t, \n, \r -- tab, newline, return
- \d -- decimal digit [0-9]
- ^ = start, $ = end -- match the start or end of the string
- \ -- inhibit the "specialness" of a character

Regex in python

```
>>> import re
>>> re.search(r'oo', 'i loove python').group()
'oo'

>>> re.search(r'\w+@\w+\.com', 'me@gmail.com').group()
'me@gmail.com'

>>> re.search(r'\w+@\w+\.com', 'me@seznam.cz').group()
AttributeError: 'NoneType' object has no attribute 'group'

>>> match = re.search(r'\w+@\w+\.com', 'me@seznam.cz')
>>> if match:
...     print(match.group())
```

Regex in python

```
>>> re.search(r'(\w+)@(\w+\.com'), 'me@gmail.com').groups()
('me', 'gmail.com')

>>> re.search(r'(\w+)@(\w+\.com'), 'me@gmail.com').group(1)
'me'

>>> re.search(r'(\w+)@(\w+\.com'), 'me@gmail.com').group(2)
'gmail.com'
```

Regex in python

```
>>> import re
>>> pattern = re.compile(r'Colou?r')
>>> match = re.search(pattern, 'Color')
>>> if match:
...     print(match.group())
'Color'
```

Exercise

- Write function that
  - Will ask user to type a name
  - Using regular expression check if
    - The name contains only letters
    - The name starts with an uppercase letter
  - If there is any problem with the name, write a message and ask the user to type the name again.

Search and Replace

- **`re.sub(pattern, repl, string, max=0)`**
- Replaces all occurrences of the regex pattern in string with replace function (repl), substituting all occurrences unless max provided

```
>>> phone = '2004-959-559 #This is Phone number'
>>> num = re.sub(r'#.*$', '', phone)
>>> print ('Phone number: {}'.format(num))
Phone number: 2004-959-559
```

Difference between re.match() and re.search()

re.match() checks for a match only at the beginning of the string, while re.search() checks for a match anywhere in the string

```
>>> re.match("c", "abcdef")     # No match
>>> re.search("c", "abcdef")    # Match
<_sre.SRE_Match object at ...>
```

Exercise

- Create function `only_digits` that will have string as an input parameter and remove everything except for digits

```
>>> numbers = only_digits('2004-959-559 # This is Phone Number')
>>> print(numbers)
2004959559
>>> numbers = only_digits("I don't have any numbers.")
>>> len(numbers)
0
```