Appendix: Regex in OpenRefine

Some tips about OpenRefine

- The undo/redo list allows you to go back to any step of your project
- You can export it, share it and re-import it in another project
- Super useful!
- It also saves the regular expressions you have been using
- What are regular expressions?

Regular expressions

- With regular expressions you can define search patterns to find matches in the text
- You use sequence of characters that define patterns
- Might seem complex at the beginning but it is very powerful and, ultimately, not that hard
- https://regex101.com/ is a great tool for exercising
- Different languages follow slightly different conventions, but the basic principles are common

- Regular expressions start and end with /
- /a/: simplest regular expression, will just match all the “a” in the text
- In order to build efficient regular expressions, some characters are “special characters” (meta-characters) used to identify the patterns

Special characters

Chara cter	Meaning	Example	Match
\d	any digit 0-9	/call\d/	call3
\w	word characters (i.e. not whitespaces etc.)	/\w-\w/	9-a
\s	whitespace character	/a\sb/	a b
.	Any character except line-break	/.../	a-1
\	escapes special characters		
\.	a period	/\.\\s/	.\s

Escaping “metadata characters”

- Characters used to indicate fields in the MARCXML derived files (Kadoc and Ancient books) are special characters with respect to regex (namely, $ indicates the end of file)
- So pay attention!
- For example, for matching the sequence “^$1$1$c” you must write /\^\$.\$.\$./ and not simply /^$.$.$./

^	Start of string or start of line depending on multiline mode. (But when [^inside brackets], it means "not")	^abc .*	abc (line start)
$	End of string or end of line depending on multiline mode. Many engine-dependent subtleties.	.*? the end$	this is the end

Quantifiers

+	One or more	Version \w-\w+	Version A-b1_1
{3}	Exactly three times	\D{3}	ABC
{2,4}	Two to four times	\d{2,4}	156
{3,}	Three or more times	\w{3,}	regex_tutorial
*	Zero or more times	A*B*C*	AACC
?	Once or none	plurals?	plural

http://www.rexegg.com/regex-quickstart.html

Quantifiers

- Difference between greedy and lazy quantifier:
  - a greedy quantifier is going to search for a match “as long as it can”
  - So, for instance, /.+/ (1 or more, greedy), will match the whole string “aaaaaaaa”

  - A lazy quantifier will stop after it has met the “minimal requirement”
  - To make a quantifier lazy, you can use “?”
  - So /.+?/ (1 or more lazy) will stop after the first “a” in “**a**aaaaaa”
  - /.*/ (0 or more greedy) will match the whole sequence “aaaaaaaaaa”
  - /.*?/ (0 or more lazy) will not match any char in “aaaaaaa”

Why is it important to know the difference?

- Example from the “Ancient Books3 dataset. Imagine in field 245 you want to capture only the initial title (subfield $0$0$a), and you have the following text:

  ^$0$0$a Dido, tragoedia nova ex qvatuor prioribvs (potissimvm primo et quarto) libris Æneidos Virgilii desumpta & Louanii olim publicè exhibita**,$ $ $c**authore Petro Ligneo ... Adiectis postea in eosdem quatuor Virgilii libros ab eodem authore ... annotatiunculis ...

- You want to make sure that you stop when the following field starts, i.e. when a “^” or a “$” is found. But you are not sure whether in every row there is a second field after the first.

^$0$0$a Dido, tragoedia nova ex qvatuor prioribvs (potissimvm primo et quarto) libris Æneidos Virgilii desumpta & Louanii olim publicè exhibita,**$** **$** **$c**authore Petro Ligneo ... Adiectis postea in eosdem quatuor Virgilii libros ab eodem authore ... annotatiunculis ...

- The regex /\^\$0\$0\$a.*?(\$|\^|$)/ is the right solution, but why?

1. \^\$0\$0\$a matches the initial **^$0$0$a**
2. **.*? will match as many characters as possible before**
3. reaching a $ OR a ^ OR the end of line (vertical bars in (\$|\^|$) indicate the OR)

- /\^\$0\$0\$a.*(\$|\^|$)/ will not work because the first part (**\^\$0\$0\$a.**) will simply match the whole text, because .* is greedy, and after that it won't find any additional element to match (neither "$", nor ^, not the end of file)

Logic

|	Alternation / OR operand	22|33	33
(…)	Capturing group	A(nt|pple)	Apple (captures "pple")
\1	Contents of Group 1	r(\w)g\1x	regex
\2	Contents of Group 2	(\d\d)\+(\d\d)=\2\+\1	12+65=65+12
(?: …)	Non-capturing group	A(?:nt|pple)	Apple

http://www.rexegg.com/regex-quickstart.html

Capturing groups are very important for text-cleaning purposes: if in one column you want to match dates and only copy the year in a separate column, you can for instance use /\d\d-\d\d-(\d\d)/ and then indicate “\1” for replacing purposes

/\^\$0\$0\$a(.*?)(\$|\^|$)/

Useful resources for regex

- https://docs.oracle.com/javase/tutorial/essential/regex/
- https://docs.openrefine.org/manual/expressions#regular-expressions
- https://docs.openrefine.org/manual/grelfunctions
- https://gist.github.com/pmgreen/6e133c5dcde65762d29c
- http://www.rexegg.com/regex-quickstart.html
- https://regex101.com/