Introduction to Regular Expressions

Exercise 1: Avian Internet?

Preparation

1. Copy the whole text of RFC2549 into the RegExr **Text** window.
2. Clear the existing regular expression

Finding literal words

1. `avian` - 2 matches
2. `Avian` - 12 matches
3. `avian` with the ignore case flag gives 14 matches
4. Clear the case insensitive flag

Most simple searches are just like traditional find and replace.

Find only capitalised words:

1. `[A-Z]\w*` - character classes [] and , wildcards *,+,?
2. `[A-Z]+` - match only "all caps" words. Not quite right
3. `\b[A-Z]+\b` - need to match on a word boundary using , an anchor class. This allows us to match a whole word.
4. `\b[A-Z]{2,}\b` - abbreviations are usually 2 or more upper case characters. {} allow for arbitrary repetition

Some characters (eg. "", "[") don't have a literal meaning. They are "meta-characters"

Match the last words of sentences:

1. `\w+.` This doesn't work because "." matches every character
2. `\w+\.` ("." is a meta-character. We need to escape it)
3. `\w+\.\s` (stops a match on the email address at end. "" matches whitespace)

We can restore a letter's literal meaning by "" escaping it.

Find all years

1. `\d\d\d\d` (lots of other things match)
2. `\d{4}` (more succinct, but doesn't improve things much)
3. `\b\d{4}\b` (match 4 digits surrounded by word boundaries. Still some false positives)
4. `\b(19|20)\d\d\b` (better if years span 1900-2099)

Note the "|" - alternation, alternatives. Note the "()" - grouping

Phone numbers

1. `\(\d{3}\) \d{3}-\d{4}` (very specific. OK if everyone writes phone numbers consistently)
2. `\(?\d+\)? ?[\d-]{5,}\d` (more permissive)

Note "\" to escape "(".

Note "?" to indicate optionality

Email addresses

1. `\w+@[\w\.]+`

This rule is quite permissive. It's likely to match some invalid email addresses. e.g. "fred@.invalid.net".

It's also likely miss valid email addresses. e.g. "luc.small@intersect.org.au"

Test your regular expressions with representative data!

Section headings

1. Flag multline - this enables "^" and "$" anchors
2. `^(\w+ ?)+$` (match repeating words + optional space)
3. Reset the multiline flag

Note how "+" can be applied to a group "(...)"

Exercise 2: To die upon a kiss

Preparation

1. Browse to Othello full-text.
2. Paste full text into regexr.

Exploring honesty

1. turn on case insensitive flag
2. `honour` - 14 matches
3. `honou?r` - optional "u"
4. `hon(our|ourable|esty?)` - honour, honourable, honest, honesty
5. turn off case insenstive flag

Acts and Scenes

1. turn on multiline matching
2. `^(ACT|SCENE) [IVXLCDM]+` (literal word, space, roman numerals)
3. turn off multline matching

Major Parts

1. turn on multiline matching
2. `^[A-Z]+$`
3. turn off multline matching

Questions

1. turn on multiline matching

2. `^.*\?` (from start of line to question mark
3. turn off multiline matching

Exercise 3: Random names

Preparation

1. List of random names
2. Select to list in text area
3. Copy to regexr.

Match given name and surname

1. `(\w+) (\w+)`
2. `"$&"` (quoting the match)
3. `- $2, $1` (swapping names)
4. `<b>$2</b>, $1` (swapping names, bolding surnames)

Shows how we can use regexes to make substitutions

Exercise 4: Tweets

Preparation

1. Twitter Data
2. Copy column "C". Cursor in C2, then Shift-Command-Down.
3. Paste into regexr

Match a #hashtag and a @handle:

1. `#\w\w+`
2. `@\w\w+`
3. `@[A-Za-z]\w+` (avoid matching a time)

Exercise 5: Reformatting Dates

Objective: convert the dates to ISO, and then the event name in quotes separated by a comma, for a spreadsheet.

Input: `The Massachusetts Bay Colony founded, on 3-4-1629`

Desired output: `1629-03-04,"The Massachusetts Bay Colony founded"`

Preparation

1. Dates in American History
2. Copy raw text and paste into regexr

Match event name with a catch-all regular expression

Notice that the event names are at the start of each line and go up until the sequence ", on". We can use this literal sequence to parse the event name from the line.

1. `.*` - This matches everything, or nothing. We need to limit this search to the event name.
2. `.*, on` - This now parses just the event name and stops before the date.

Matching dates

1. `\d{1,2}` - Captures one or two digits
2. `\d{1,2}[-\/\.]` - Captures the first two digits plus the separator. Double this:
3. `\d{1,2}[-\/\.]\d{1,2}[-\/\.]` - Doesn't match two dates with `, ` as separator.
4. `\d{1,2}[-\/\.]\d{1,2}(-|\.|\/|, )` - Matches correctly, note the change to parentheses set with pipes rather than bracketed set due to two-character separator.
5. `\d{1,2}[-\/\.]\d{1,2}(-|\.|\/|, )\d{4}` – add 4-digit year.

Setting capture groups

Place parentheses around the matched elements for replacement. Including `.*, on` at the start to capture the name of the event:

```
(.*), on (\d{1,2})[-\.\/](\d{1,2})(-|\.|\/|, )(\d{4})
```

The capture groups are:

1. The name of the event
2. The month
3. The day
4. The separator between the day and the year
5. The year.

Putting `?:` as the first sequence inside the parentheses will make it a non-capturing group. Add this to group 4 to not include the separator as a group: `(?:-|\.|\/|, )`.

Replacment

Open up the 'Replace' tab and output the captured elements in the format: `{day}-{month}-{year},"{event}".`

1. `$5-$2-$3,"$1"` – (if second separator is capturing group)
2. `$4-$2-$3,"$1"` – (if second separator is non-capturing group)

Zero-padding single-digits dates

ISO dates require zero-padding for single digits: `1629-03-04` rather than `1629-3-4`. Regex cannot easily do conditionality (if expression only matches one digit then output a zero at the start) – you would need a programming language to do this. So we simply pass it through another search and replace.

Copy the output and paste it into the input, and write another expression.

1. `-\d` – Needs to be delimited at the end of the string. Matches first of two digits.
2. `-\d(-|,)` – Does not correctly match the last date as second dash is already matched against `-3-`.
3. `\b\d\b` – class includes both `-` and `, ` and doesn't actually capture the character.
4. Simply replace with `0$1` to zero-pad the single digit dates.