04. Regular Expression

Contents

1. Introduction

Regular expression is a way to represent textual patterns. These patterns can be used to find and replace text. The syntax for patterns is mostly standardised (though variations exist). Pattern matching is performed by a text processing engine, which takes care of the details behind the scenes. As a user, familiarity with regular expression patterns gives you a powerful tool for text manipulation. Regular expression is a well-known and widely-used text matching technique. Every popular programming language and software development tool will support it.

Regular expression is often abbreviated to RegExp, RegEx, or RE. Patterns are often called "regexes".

See https://simple.wikipedia.org/wiki/Regular_expression and https://en.wikipedia.org/wiki/Regular_expression (particularly the subheadings: Patterns, Basic concepts and Uses).

1.1 Sample use cases

i. Find every occurrence of the word "analyse" in either NZ or US spelling

```
/analy[sz]e/
```

ii. Find all lines of text which begin with a lowercase English letter

```
/^[a-z]/
```

iii. Find all contiguous spaces and replace each with an underscore

Find:	`/\s+/`
Replace with:	`_`

iv. Find auckland.ac.nz URLs (terminated with a space) which may or may not be prefixed with "http://" and/or "www", and may or may not end with a slash

```
/(http:\/\/)?(www\.)?auckland\.ac\.nz\/?.*\/? /
```

Since the combinations and possibilities are endless, we will only cover regex basics in this course. Mastering the basics will enable you to meaningfully navigate online resources and tweak any patterns you find to suit your needs.

1.2 PCRE

MongoDB[1] and many other tools and platforms use the "PCRE" standard, so make sure whatever tutorial or references you use are for PCRE. Most of what we use in this course is universal across any regex implementation.

Visual Studio Code[2] and Atom[3] use Javascript[4] Regular Expressions.

[1] https://docs.mongodb.com/manual/reference/operator/query/regex/
[2] https://stackoverflow.com/questions/42179046/what-flavor-of-regex-does-visual-studio-code-use
[3] https://flight-manual.atom.io/using-atom/sections/find-and-replace/
[4] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

1.3 Recommended learning resources

https://regex101.com/ and http://regexr.com/ are useful tools to test and visualise regexes. They also provide cheat sheets.

Simple **tutorial** from Chapter 11 of "Introduction to Data Technologies" by Paul Murrell:
https://www.stat.auckland.ac.nz/~paul/ItDT/HTML/node82.html

Walkthrough of the basics with exercises by RegexOne: https://regexone.com/
Additional problem exercises by RegexOne: https://regexone.com/problem/matching_decimal_numbers

Solve programming problems involving regex:
https://www.codewars.com/kata/search/?tags=Regular+Expressions&beta=false

Random string generator defined using regex patterns: http://fent.github.io/randexp.js/

2. Usage in MongoDB find queries

We will practise using REs primarily with MongoDB data. The find command supports regular expressions using the $regex operator and regular expression objects (using the JavaScript-style "literal notation"). See https://docs.mongodb.com/manual/reference/operator/query/regex/.

Note that when using the `$in` operator, REs must use the "literal notation". There are other restrictions and differences between using `$regex` and "literal notation", but they will not affect queries used in this course.

2.1 Examples using the ig db

```
db.profiles.find({ full_name: /ab/ })   //fullname contains lowercase ab

db.profiles.find({ full_name: /^ab/ })  //fullname starts with lowercase ab

db.profiles.find({ full_name: /^ab/i }) //fullname starts with ab (case-insensitive)
db.posts.find(
   { 'comments.text': { $regex: 'NZ' } }, //post with a comment containing NZ
   { 'comments.$': 1 }                    //project first matching comment
)

db.posts.find(
    { 'comments.text': { $regex: 'NZ', $options: 'i' } }, //case-insensitive
    { 'comments.$': 1 }
)
```

3. Patterns

A pattern is a sequence of characters (atoms) and metacharacters (special atoms and instructions for matching).

3.1 Characters and anchors

Complete these lessons:

https://regexone.com/lesson/introduction_abcs
https://regexone.com/lesson/letters_and_digits
https://regexone.com/lesson/line_beginning_end

Pattern	Scenario
`/hello/`	Contains literal: hello
`/^hello/`	Starts with literal
`/hello$/`	Ends with literal
`/^hello$/`	Line matches literal
`/\d\d\d\d/`	Contains four digits
`/\d\d$/`	Ends with two digits

3.1.1 Exercises

Use the query below on the ig db to produce an _id-sorted list of profile biography strings. You can also paste the list into https://regex101.com/ to manually inspect your answers.

```
db.profiles
    .find({}, { biography: 1 })
    .sort({ _id: 1 })
    .forEach(d => {
        print(`${d._id + ' '.repeat(11 - d._id.length) +
         JSON.stringify(d.biography)}`)
    })
```

Write find queries to match the given conditions on ig.profiles.biography. Project the biography field only.

a) Contains: New
b) Contains: new
c) Contains: 🌍
d) Contains two digits that are surrounded by spaces on both sides `/ \d\d /`
e) Begins with a digit
f) Ends with a digit
g) Contains a usertag. Usertags are prefixed with the @ symbol
    - Avoid matching e-mail addresses
h) Contains a new line character—this is denoted by the character sequence `\n`
i) Contains a domain name ending in `.au`

3.2 Dot metacharacter

Complete this lesson:

https://regexone.com/lesson/wildcards_dot

3.2.1 Exercises

Use ig.profiles.biography. Assume words are delimited by spaces.

a) Contains a 4-letter word starting with `p` `/ p... /`
b) Contains a 4-letter word starting with `P`

c) Contains a 7-letter word starting with `t`
   - What are the problems here?

3.3 Sets

Complete these lessons:

https://regexone.com/lesson/matching_characters
https://regexone.com/lesson/excluding_characters
https://regexone.com/lesson/character_ranges

Note that `\w` in PCRE includes letters, numbers, and underscore. It is a shortcut for the set `[A-Za-z0-9_]`

Other predefined sets:

```
\W = [^A-Za-z0-9_]

\d = [0123456789] = [0-9]

\D = [^0123456789]
```

Pattern	Scenario
`/02[1257]\d\d\d\d\d\d/`	Choose one character
`/^@[^@]abc$/`	Avoid one character (or range)

`/rating: [12345]/`
`/0x[a-f0-9][a-f0-9]/`
	Choose one character from range

`/agent_[a-z]/`	Choose one lowercase letter only
`/Agent [A-Z]/`	Choose one uppercase letter only

`/\w\w\w/`
`/[A-Za-z0-9_][A-Za-z0-9_][A-Za-z0-9_]/`
	3 "word" characters

`/[A-Z][A-Z][A-Z]/`	3-letter alphabetic English word in uppercase

3.3.1 Exercises

Use ig.profiles.biography. Assume words are delimited by spaces or full stops.

a) Contains a 4-letter word starting with `p`

b) Contains a 4-letter word starting with `t`

c) Contains a 7-letter word starting with `t`

d) Ends in a space or full stop `/[ \.]$/`

e) Contains a usertag.
   - Usertags are prefixed with the @ symbol
   - The @ symbol cannot be prefixed with an alphanumeric character
   - The first character of a usertag must be a lowercase letter

f) Contains a number up to six digits that may contain comma separators. E.g. `10,000`

3.4 Quantifiers (metacharacters)

Complete these lessons:

https://regexone.com/lesson/repeating_characters
https://regexone.com/lesson/kleene_operators
https://regexone.com/lesson/optional_characters

Pattern	Scenario
`/\d{4}/`	4 digits

`/hello+/`
`/firstName +lastname/`
`/firstName[ _]+lastname/`
	One or more of an atom (character, set, etc.)

`/hello*/` `/hello!*/`	Zero or more of an atom

`/patterns?/`
`/hello[!\?]?/`
	Optional atom

3.4.1 Exercises

Use ig.profiles.biography.

a) Contains: `new zealand`
    - There may or may not be a space between the two words

b) At least 37 characters long

c) Exactly 37 characters long `/^.{37}$/`

d) Only contains alphanumeric characters `/^[a-zA-Z0-9]+$/`

e) Only contains alphanumeric characters, spaces, and full stops

f) Contains at least 3 numbers anywhere

g) Contains at least 3 non-English word characters anywhere

h) Contains a `.com` e-mail address. Assume e-mail addresses can only contain alphanumeric, underscore, and dot characters

3.5 Grouping

Complete these lessons:

https://regexone.com/lesson/capturing_groups
https://regexone.com/lesson/nested_groups
https://regexone.com/lesson/more_groups

Groups allow multiple characters to be treated as one atom. For example, we want to match lines consisting of `na` repeated any number of times. Run the code below to compare the patterns `/^n+a+$/` and `/^(na)+$/`

```
(() => {
    ungrouped = /^n+a+$/
    grouped = /^(na)+$/
    print(`using /^n+a+$/ on:
nanananana = ${ungrouped.test('nanananana')} (want to be true)
nanan = ${ungrouped.test('nanan')} (want to be false)

using /^(na)+$/ on:
nanananana = ${grouped.test('nanananana')}  (want to be true)
nanan = ${grouped.test('nanan')} (want to be false)`)
})()
```

Groups vs sets:

	Matches `/(\?!){2,}$/`	**Matches** `/[\?!]{2,}$/`
What??		
What!!??		
What?!?!		
Really!?!!???!?!		
What?!!?!		

Grouping is needed for OR logic and "capturing" (remembering) matches. These are shown in their respective sections below.

3.5.1 Exercises

Use ig.posts.caption.

a) Contains: `Wed` or `Wednesday` followed by a space
b) Contains even numbers of contiguous exclamation marks, e.g. matches `!!` but not `!` nor `!!!`
   - After the even number of exclamation marks, the next character cannot be an exclamation mark
c) Contains even numbers of contiguous exclamation marks, but only where there are 4 or more
d) Contains: `Thu` or `Thurs` or `Thursday` followed by a space
   - Use a nested group to avoid matching `Thuday`

3.6 OR logic (alternation)

Complete this lesson:

https://regexone.com/lesson/conditionals

Pattern	Scenario

`/(hello|goodbye|farewell)/`
`/th(is|at)/`
`/(\+64|00)0?\d{8,11}/`
	Choose one "group" from options

`/(C#|D♭)+/`
`/www\.\w+\.(org\.|net\.|co\.)?nz/`
`/((na|batman) ){2,}/`
	Choice "groups" are treated as atoms

3.6.1 Exercises

Use ig.posts.caption.

a) Contains: `Mon`, `Monday`, `Fri`, or `Friday` followed by a space

b) Contains: `weekend` or `weekday`
   - Avoid matching: `week`

c) Contains any of the following terms surrounded by spaces or full stops: `nz`, `NZ`, `newzealand`, `NewZealand`

d) As in a) but the leading `M` or `F` may be uppercase or lowercase (all other characters should remain lowercase)

e) Contains a hashtag or a usertag.

3.7 Options

Options are also known as "flags" or "switches".

There are 4 options when using MongoDB regular expressions: https://docs.mongodb.com/manual/reference/operator/query/regex. Only the "i" option is of interest to us. This option means to perform a case-insensitive match.

Options are listed after the last slash when using "literal notation", or in the `$options` field when using `$regex`.

Pattern	Scenario
`/hello/i`	Case-insensitive match option using "literal notation"
`{ $regex: /hello/, $options: 'i' }`	Case-insensitive match option using $regex

Consider `/hello/i` vs `/[Hh]ello/`

4. Find and Replace in text editors

A benefit of grouping patterns is that the matched pattern can be referenced. For example, to perform a conversion of UoA student e-mail addresses into quoted usernames would require preserving part of the matched e-mail address: the part before the @ symbol.

Find: `/([a-z]{2,4}\d{3})@(aucklanduni)\.ac\.nz/i`

Replace with: `"\1"`

`\1` here refers to the first matched grouping (the blue one). `\2` would refer to the second grouping (the green one).

- Notepad++ and TextWrangler/BBEdit use \1, \2, \n to refer to captured groups
- VSCode uses $1, $2, $n to refer to captured groups

4.1 Exercises

Copy and paste the text below into a text editor and perform the transformations given using regex-enabled find and replace.

```
username    phone           date (mm/dd/yyyy)
astu111     021-222-2222    01/01/2001
bgra099     022-555-5555    12/12/2002
hhe456      027-100-20000   11/13/2003
pxav334     022-334343434   04/18/2004
```

Transformations:

a) Change usernames into e-mail addresses, e.g. `abcd123` becomes `abcd123@aucklanduni.ac.nz`

b) Internationalise phone numbers by correctly prefixing `+64`, e.g. `021-123-1234` becomes `+6421-123-1234`

c) Change usernames into pdf filenames in a folder with the same name as the username, e.g. `abcd123` becomes `C:/abcd123/abcd123.pdf`

d) Swap the month and day digit positions, e.g. `05/23/2017` becomes `23/05/2017`

e) Format dates as yyyy-mm-dd, e.g. `05/23/2017` becomes `2017-05-23`

f) Transform the table into a list of comma-separated usernames.
   - Formula: replace EOL with a comma, then remove the last comma.
   - This can also be achieved using multi-line (a.k.a. column or block) editing which Notepad++, TextWrangler/BBEdit, and VSCode all support.

g) Transform the table into a list of comma-separated quoted usernames. This format could be used in a JSON array or SQL query, for example.

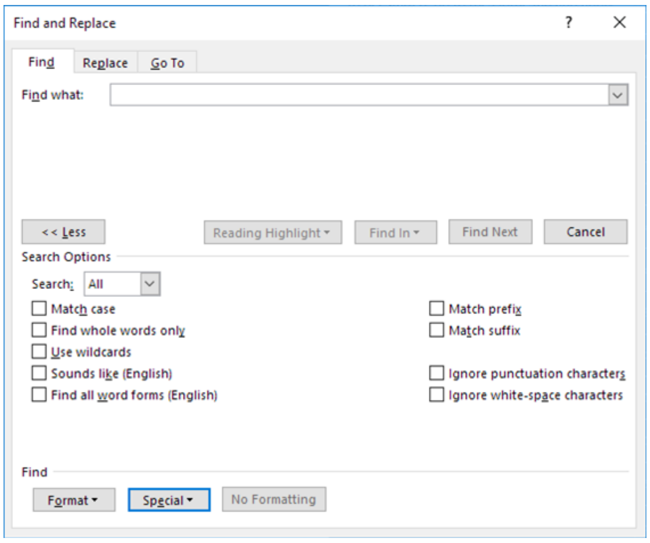
4.2 Google sheets

Google sheets offers regex features as functions: REGEXMATCH, REGEXEXTRACT, REGEXREPLACE

5. RegEx-like features in Microsoft Office

For Windows see https://support.office.com/en-us/article/replace-text-5b459a33-5bdf-4052-9508-f50127b90a75

For Mac see https://support.office.com/en-us/article/find-and-replace-text-c6728c16-469e-43cd-afe4-7708c6c779b7#ID0EAABAAA=macOS



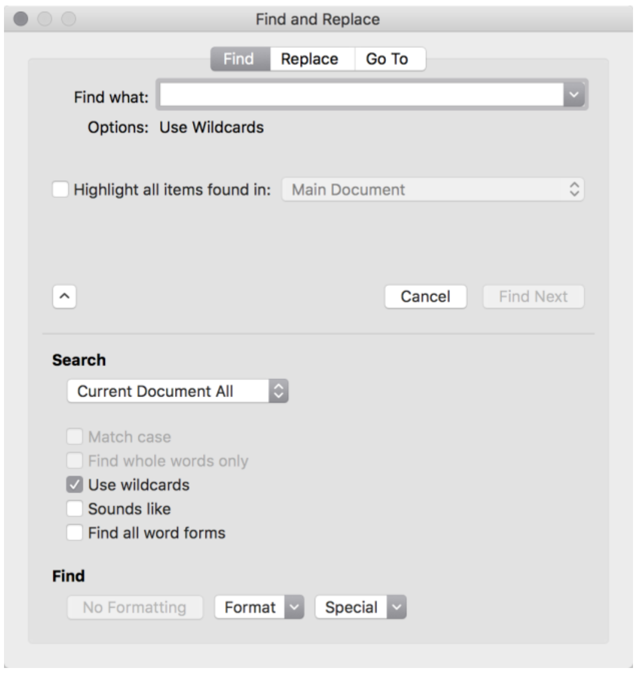
Find and Replace

Find what:

Search Options

Search: All

- ☐ Match case
- ☐ Find whole words only
- ☐ Use wildcards
- ☐ Sounds like (English)
- ☐ Find all word forms (English)
- ☐ Match prefix
- ☐ Match suffix
- ☐ Ignore punctuation characters
- ☐ Ignore white-space characters



Find and Replace

Find what:

Options: Use Wildcards

☐ Highlight all items found in: Main Document

Search

Current Document All

- ☐ Match case
- ☐ Find whole words only
- ☑ Use wildcards
- ☐ Sounds like
- ☐ Find all word forms

6. Document change history

v1.0 2022-04-05

- Initial release