Regular Expression

陳曾基

國立陽明交通大學醫學院醫務管理研究所
國立陽明交通大學醫學院急重症醫學研究所
國立陽明交通大學醫學院醫學系家庭醫學科
臺北榮民總醫院家庭醫學部
臺北榮民總醫院醫學研究部大數據中心

Topics

- RE basics
- RE in Notepad++
- RE in base R
- RE in tidyverse (stringr)
- Resources

SECTION I

RE BASICS

Regular Expression

Foundation of Text Mining/Data Science

A Concept/Technique/Sublanguage

Source:

RE Bible

https://en.wikipedia.org/wiki/Regular_expression

Regular expression

From Wikipedia, the free encyclopedia

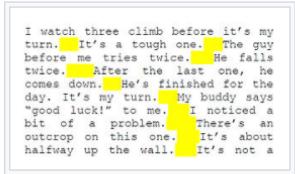
"Regex" redirects here. For the comic booox, see Re:Gex.

This article **may be too technical for most readers to understand**. Please help improve it to make it understandable to non-experts, without removing the technical details. *(September 2016) (Learn how and when to remove this template message)*

A **regular expression**, **regex** or **regexp**[1] (sometimes called a **rational expression**)[2][3] is, in theoretical computer science and formal language theory, a sequence of characters that define a *search pattern*. Usually this pattern is then used by string searching algorithms for "find" or "find and replace" operations on strings.

The concept arose in the 1950s when the American mathematician Stephen Cole Kleene formalized the description of a *regular language*. The concept came into common use with Unix text-processing utilities. Since the 1980s, different syntaxes for writing regular expressions exist, one being the POSIX standard and another, widely used, being the Perl syntax.

Regular expressions are used in search engines, search and replace dialogs of word processors and text editors, in text processing utilities such as sed and AWK and in lexical analysis. Many programming languages provide regex capabilities, built-in or via libraries.



The match results of the pattern

```
(?<=\.) {2,}(?=[A-Z])
```

At least two spaces are matched, but only if they occur directly after a period (.) and before an upper case letter.

https://zh.wikipedia.org/wiki/正则表达式

首頁
分類索引
特色內容
新聞動態
近期變更
隨機條目

說明

說明
維基社群
方針與指引
互助客棧
知識問答
字詞轉換
IRC即時聊天
聯絡我們
關於維基百科
資助維基百科

其他專案

正規表示式 [編輯]

維基百科，自由的百科全書

本條目存在以下問題，請協助改善本條目或在**討論頁**針對議題發表看法。 [合併]

- 本條目內容**疑欠準確，有待查證**。（*2013年11月22日*）
- 本條目**需要補充更多來源**。（*2013年11月22日*）

正規表示式（英語：Regular Expression，在代碼中常簡寫為regex、regexp或RE），又稱**正則表達式**、**正規表示法**、**正規運算式**、**規則運算式**、**常規表示法**，是電腦科學的一個概念。正規表示式使用單個字串來描述、符合一系列符合某個句法規則的字串。在很多文字編輯器裡，正則運算式通常被用來檢索、替換那些符合某個模式的文字。

「Regular Expression」的各地常用譯名	
中國大陸	正則表達式
臺灣	正規表示式、正規表示法、正規運算式、規則運算式
香港	正則表達式

許多程式設計語言都支援利用正則運算式進行字串操作。例如，在Perl中就內建了一個功能強大的正則運算式引擎。正則運算式這個概念最初是由Unix中的工具軟體（例如sed和grep）普及開的。正規表示式通常縮寫成**regex**，單數有regexp、regex，複數有regexps、regexes、regexen。

Regular Expression

- regular expressions、regex、RE、正規表現、正規表示式、正則表達式、正規表達式、常規表達式
- a kind of pattern matching
- 用來描述或者匹配一系列符合某個句法規則的字元串 (In computing, a regular expression is a string that is used to describe or match a set of strings, according to certain syntax rules.)
- 通常被用來檢索和/或替換那些符合某個模式的文本內容
- 普遍存在於文字編輯器(text editor)、文書處理器(word processor)、搜尋引擎與JavaScript、SAS等各種程式語言

Regular Expression

- Literals (normal text characters)
- Metacharacters

- 有一本童書提到：「…一匹大野狼吃掉(兩位數)隻小羊…」，請找出正確數字及本句在書上哪幾頁

Appetizer

有一個式子為 3+4=7 (或 33+44=77, 3x+4x=7x)：

- 取出加號與等號兩旁的數字（假設數字為一位數）
- 取出加號與等號兩旁的數字（不限數字位數）
- 取出加號與等號兩旁的數據（不限數字或文字）
- 將加號與等號兩旁的數字（假設數字為一位數）皆各乘以9
- 將加號兩旁的數字互換

Key Point

Abstraction

What can we do with RE?

- check if an email address is correctly formed
- parse dates, social security numbers and the like
- extract the 3rd word of a sentence
- find repeated words in a piece of text
- read;in;semicolon;separated;input
- find all dollar amounts in a string
- …

Commands in MS-DOS (not real RE)

- C:\>dir *.doc
- C:\>dir fm*.doc
- C:\>dir fm*cr.doc
- C:\>dir fm?.doc
- C:\>dir fm??.doc

RE Examples 1

- "Frodo"
- "Frodo|Pippin|Merry|Sam"
- "(Frodo|Drogo|Bilbo) Baggins"
- "(Frod|Drog|Bilb)o Baggins"
- "^(From|Subject|Date):"

RE Examples 2

- "pizza" => pizza apizza pizzas
- "\bpizza\b" => pizza pizza, pizza; pizza.
- "piz.a" => pizaa pizba pizca ……
- "piz*a" => pia piza pizza pizzza ……
- "piz+a" => piza pizza pizzza ……
- "piz?a" => pia piza

+ : 前面的字元出現至少一次 (1次或多次)
? : 前面的字元出現最多一次 (0次或1次)
* : 前面的字元可以不出現、出現一次或多次 (0次、1次或多次)

RE Examples 3

- "(pizza){3}" => pizzapizzapizza
- "(pizza){1,2}" => pizza pizzapizza
- "(pizza){2,}" => pizzapizza pizzapizzapizza ……

RE Examples 4

- "pizza\t" => pizza{tab}
- "pizza\n" => pizza{newline}
- "pizza$" => pizza{EndOfString}
- "^pizza$" => {BeginOfString}pizza{EndOfString}
- "\d pizza" => 0 pizza 1 pizza ……
- "(\d){2} pizza" => 00 pizza 01 pizza ……
- "[1-9][0-9] pizza" => 10 pizza 11 pizza ……

SECTION II

RE IN NOTEPAD++

https://npp-user-manual.org/docs/searching/#regular-expressions

Regular Expressions

Notepad++ regular expressions use the Boost regular expression library v1.70, which is based on PCRE (Perl Compatible Regular Expression) syntax, only departing from it in very minor ways. Complete documentation on the precise implementation is to be found on the Boost pages for search syntax and replacement syntax

The Notepad++ Community has a FAQ on other resources for regular expressions.

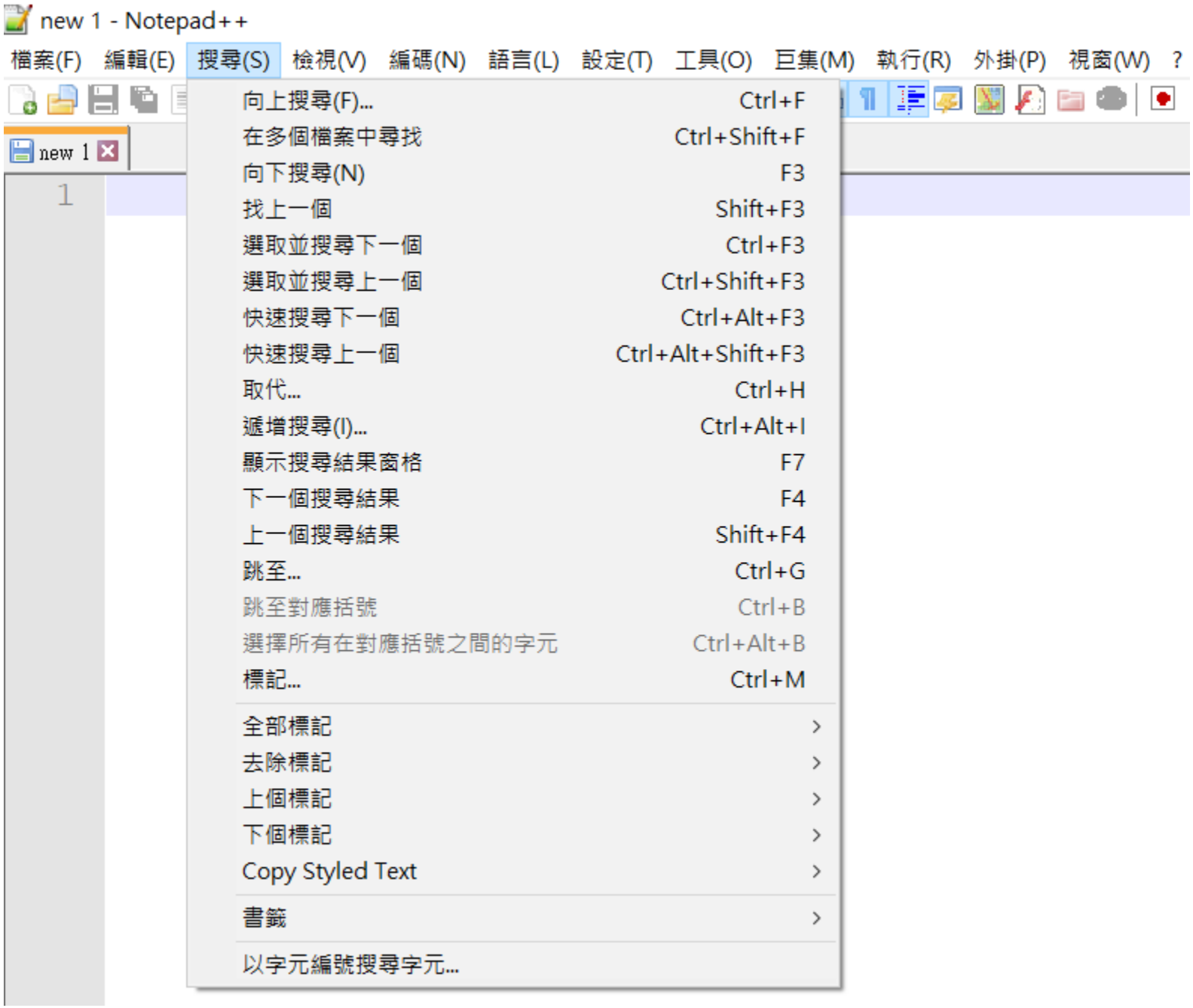
Note: Starting in v7.8.7, regex backward search is disallowed due to sometimes surprising results. If you really need this feature, please see **Allow regex backward search** to learn how to enable this option.

Regex Special Characters

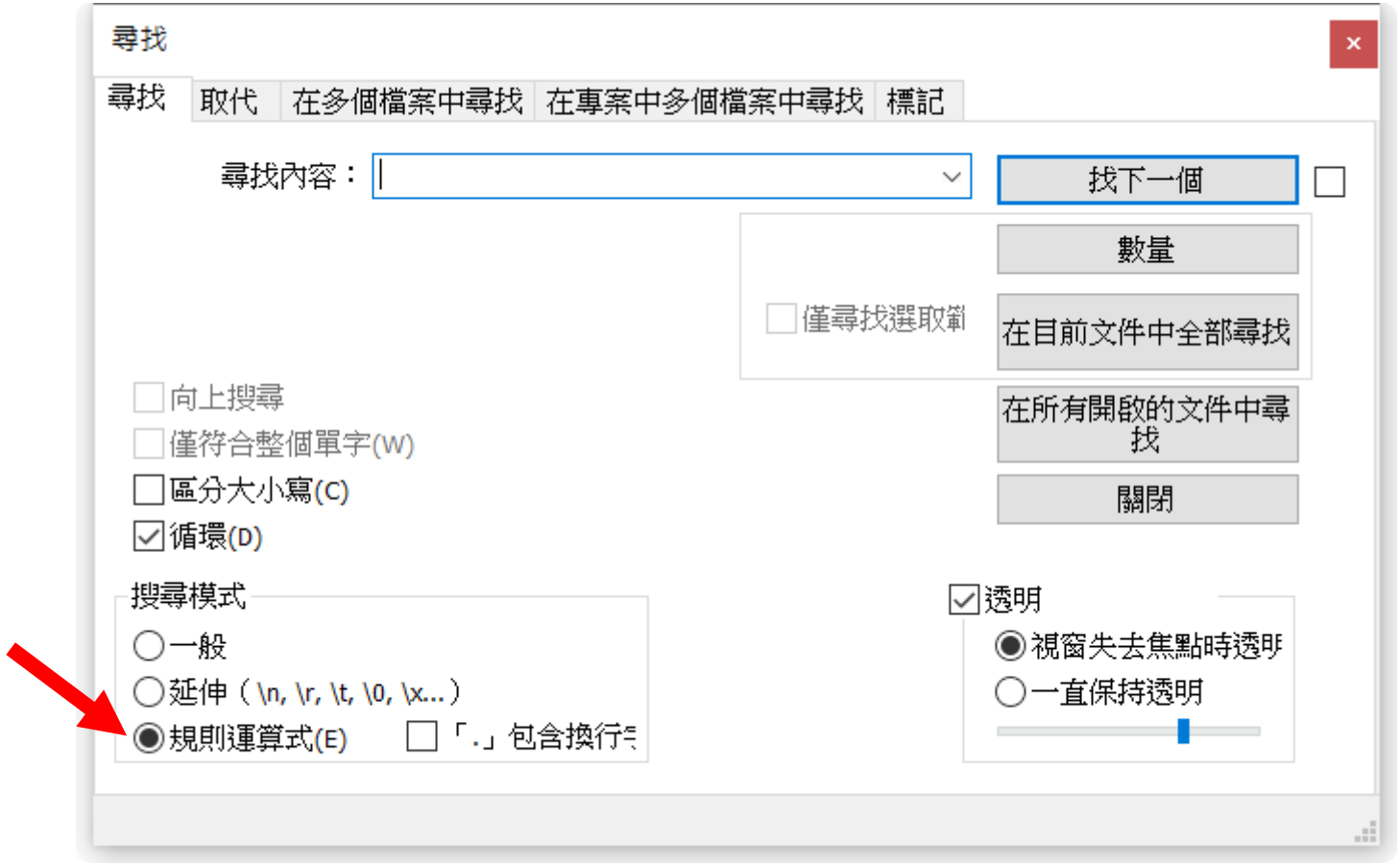
In a regular expression (shortened into regex throughout), special characters interpreted are:

Single-character matches

- `.` or `\C` ⇒ Matches any character. If you check the box which says **. matches newline**, the dot match any character, including newline sequences. With the option unchecked, then `.` will only match characters within a line, and not the newline sequences (`\r` or `\n`).
- `\X` ⇒ Matches a single non-combining character followed by any number of combining characters. This is useful if you have a Unicode encoded text with accents as separate, combining characters. For example, the letter o̲⃗, with four combining characters after the `o`, can be found either with the regex `(?-i)o\x{0304}\x{0328}\x{031a}\x{0333}` or with the shorter regex `\X`.

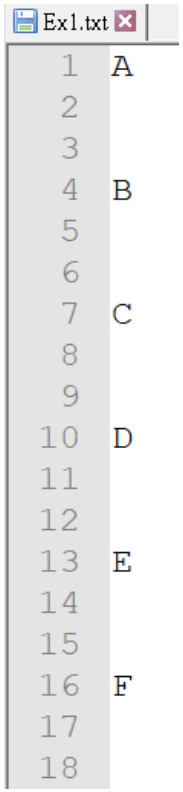

new 1 - Notepad++
檔案(F) 編輯(E) 搜尋(S) 檢視(V) 編碼(N) 語言(L) 設定(T) 工具(O) 巨集(M) 執行(R) 外掛(P) 視窗(W) ?
new 1
1
向上搜尋(F)... Ctrl+F
在多個檔案中尋找 Ctrl+Shift+F
向下搜尋(N) F3
找上一個 Shift+F3
選取並搜尋下一個 Ctrl+F3
選取並搜尋上一個 Ctrl+Shift+F3
快速搜尋下一個 Ctrl+Alt+F3
快速搜尋上一個 Ctrl+Alt+Shift+F3
取代... Ctrl+H
遞增搜尋(I)... Ctrl+Alt+I
顯示搜尋結果窗格 F7
下一個搜尋結果 F4
上一個搜尋結果 Shift+F4
跳至... Ctrl+G
跳至對應括號 Ctrl+B
選擇所有在對應括號之間的字元 Ctrl+Alt+B
標記... Ctrl+M
全部標記
去除標記
上個標記
下個標記
Copy Styled Text
書籤
以字元編號搜尋字元...

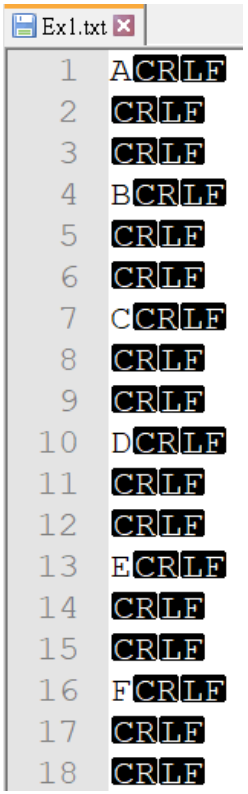
搜尋(S) / 向上搜尋(F)... / [尋找] [尋找] 搜尋模式 / 勾選 規則運算式



Exercise (Ex1.txt)

- Removing blank lines 請去除空白行





* 檔案在附件子目錄處

Solution (Ex1.txt)

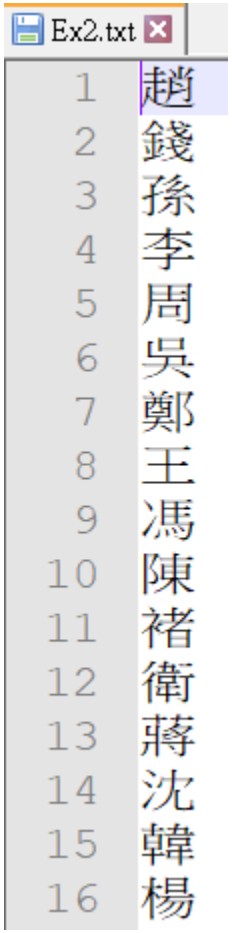
- Removing blank lines 請去除空白行

(\r\n)+
\r\n

* 在 TextPad 或其他 PCRE (Perl Compatible Regular Expressions) 軟體，\r\n 可寫成 \n

Exercise (Ex2.txt)

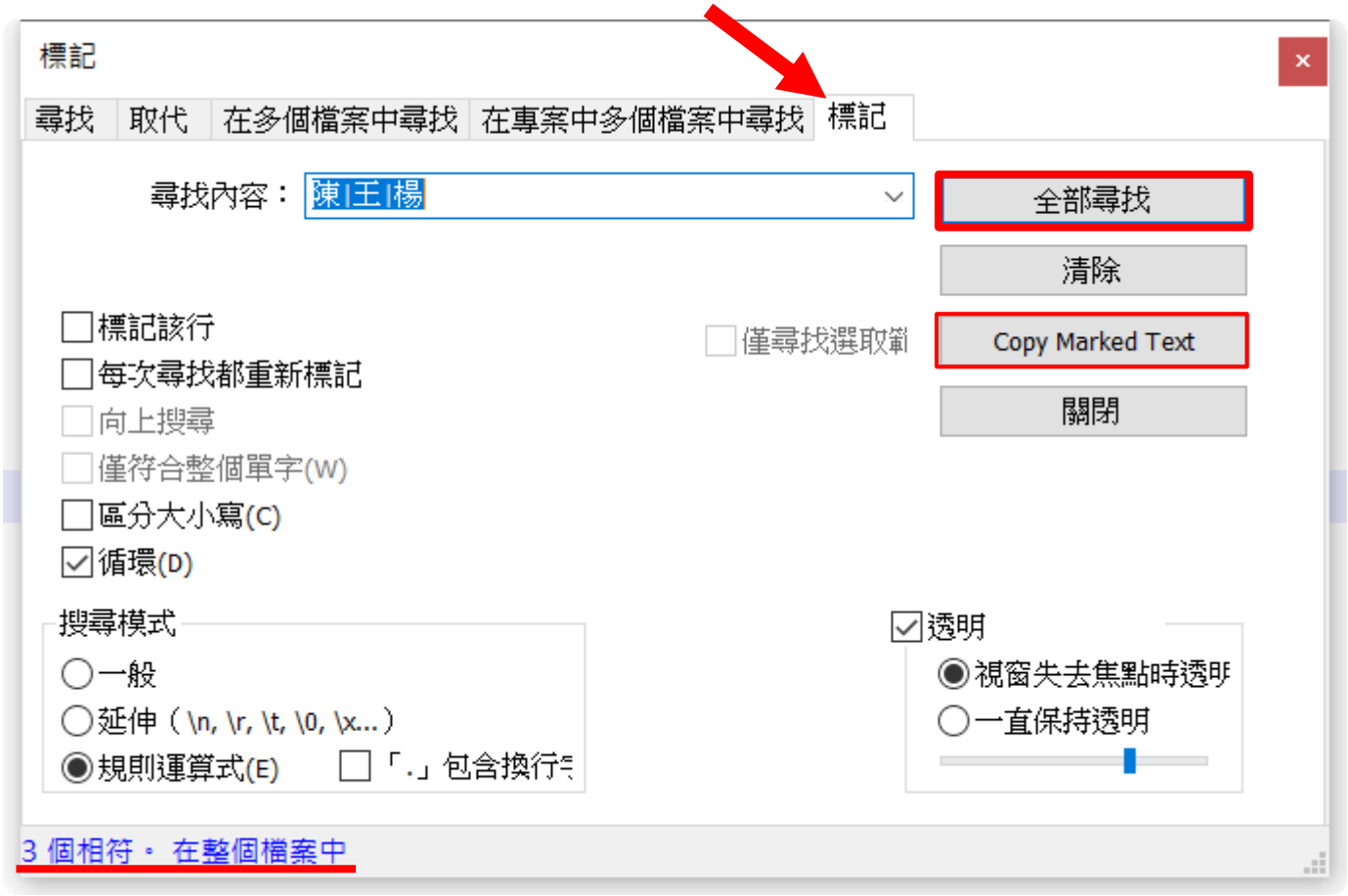
- Finding multiple words with one search



Solution (Ex2.txt)

- Finding multiple words with one search

陳|王|楊



Exercise (Ex3.txt)

- Searching for lines beginning with a word

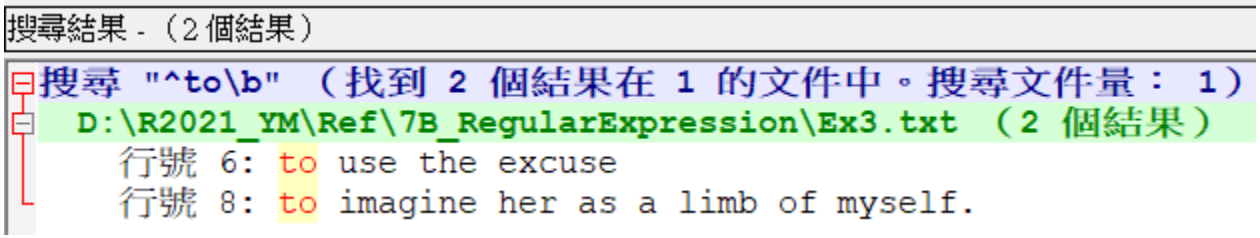
Ex3.txt

```
1  Promise
2  BY MARY O'DONNELL
3
4  I try not to cast too much shade.
5  Sin would be
6  to use the excuse
7  of her growth in my womb,
8  to imagine her as a limb of myself.
9  She is her own tree,
10 late-winter's indomitable shoot.
11 She takes cupfuls of sun.
12
13 I stand well clear
14 as the branches stretch
15 like flutes playing allegros.
16 Not for anything
17 would I poison her
18 with an act of possession,
19 conceal her from the woodsman
20 whose task is to make room for all.
```

Solution (Ex3.txt)

- Searching for lines beginning with a word

  ^to\b



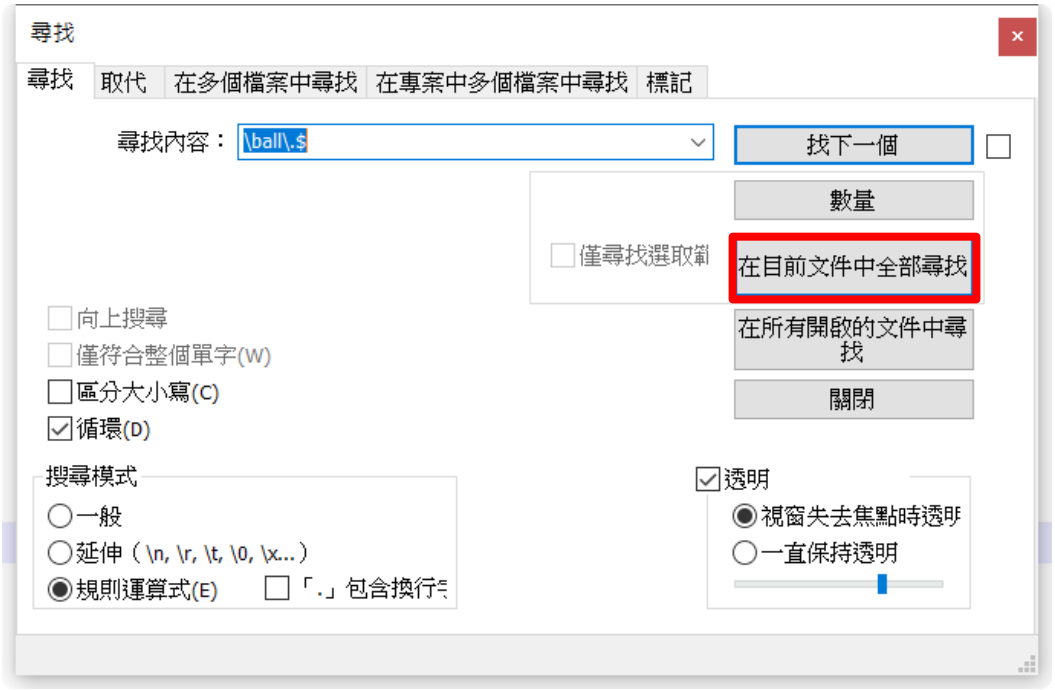
Exercise (Ex3.txt)

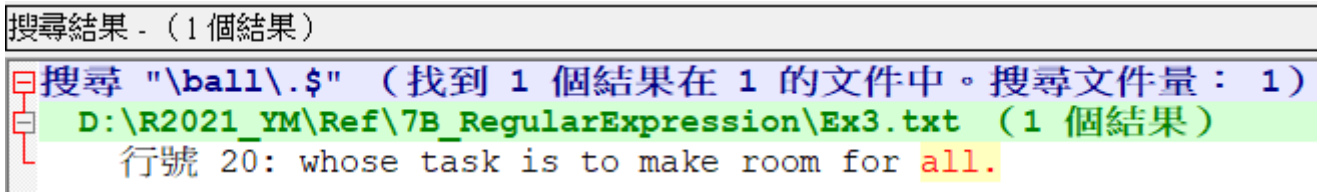
- Searching for lines ending with a word

Solution (Ex3.txt)

- Searching for lines ending with a word

\ball\.$





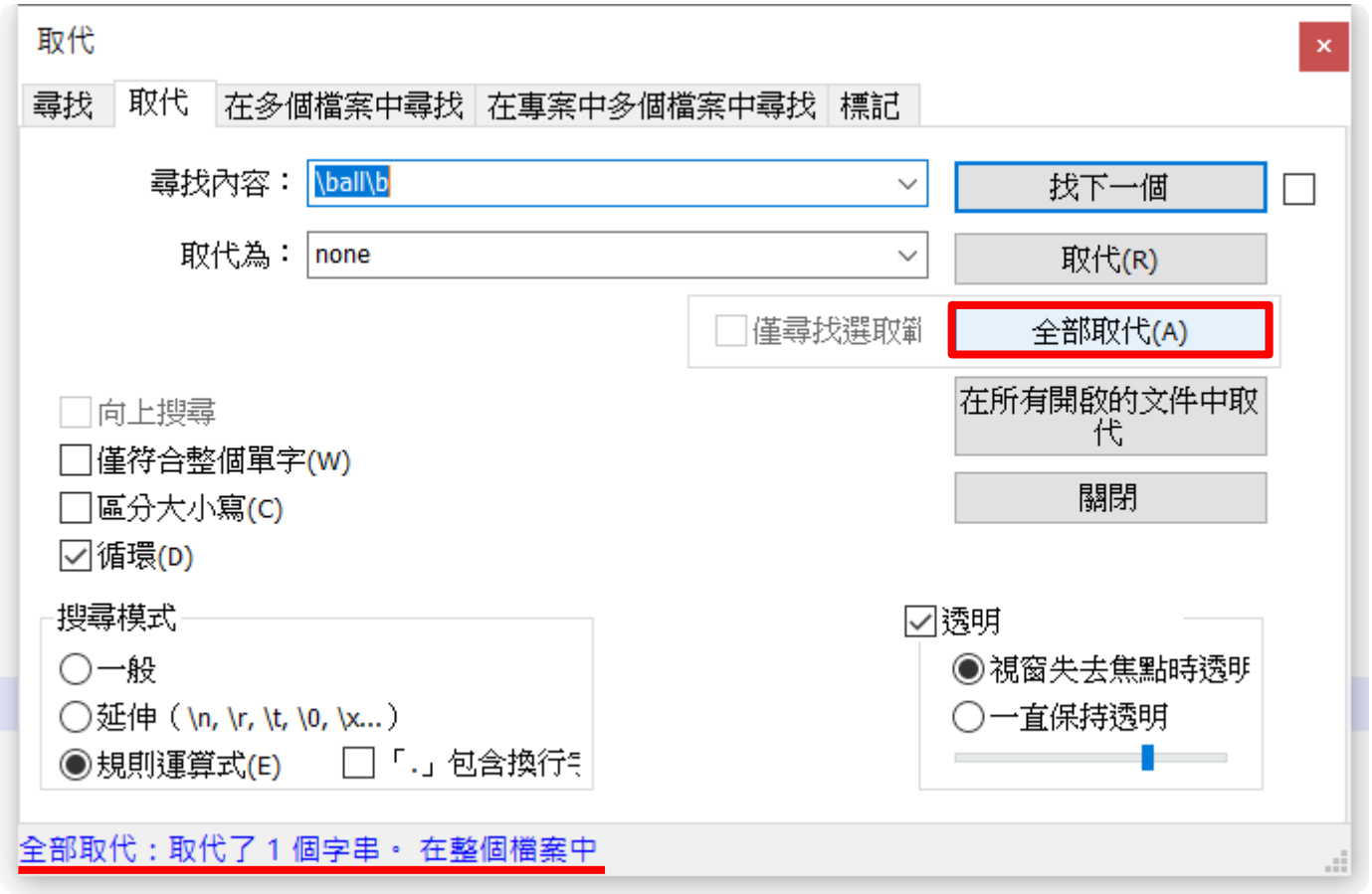
Exercise (Ex3.txt)

- Replacing words

Solution (Ex3.txt)

- Replacing words

\ball\b
none



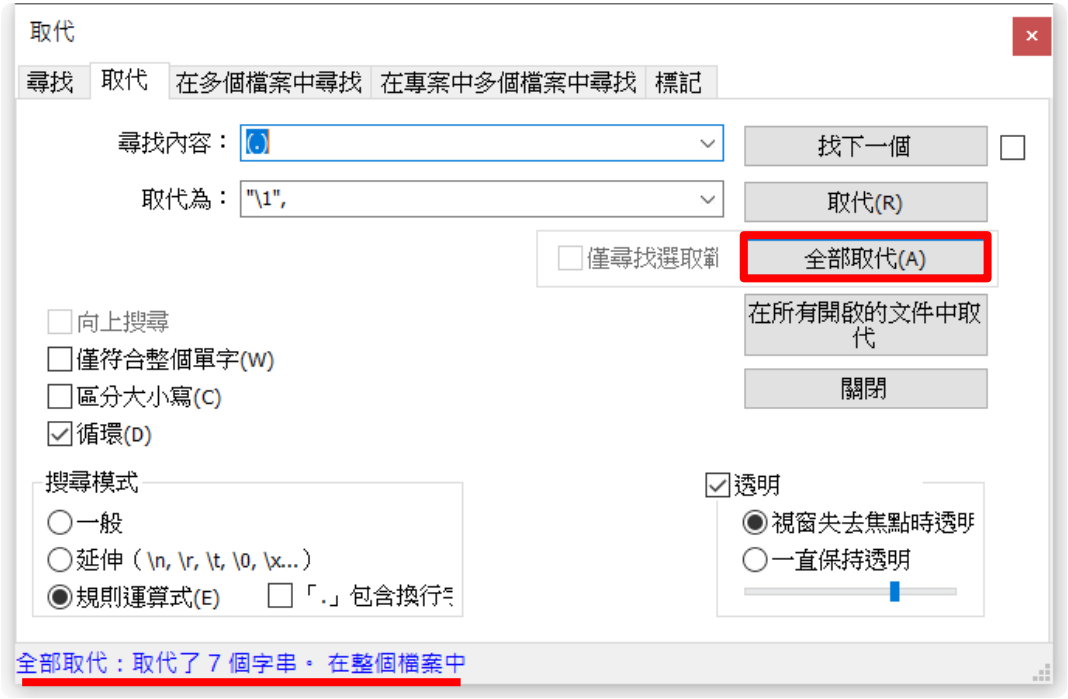
Exercise (Ex4.txt)

- Separating letters

Solution (Ex4.txt)

- Separating letters

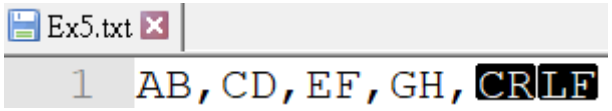
(.)
"\1",



"A", "B", "C", "D", "E", "F", "G", CRLF

Exercise (Ex5.txt)

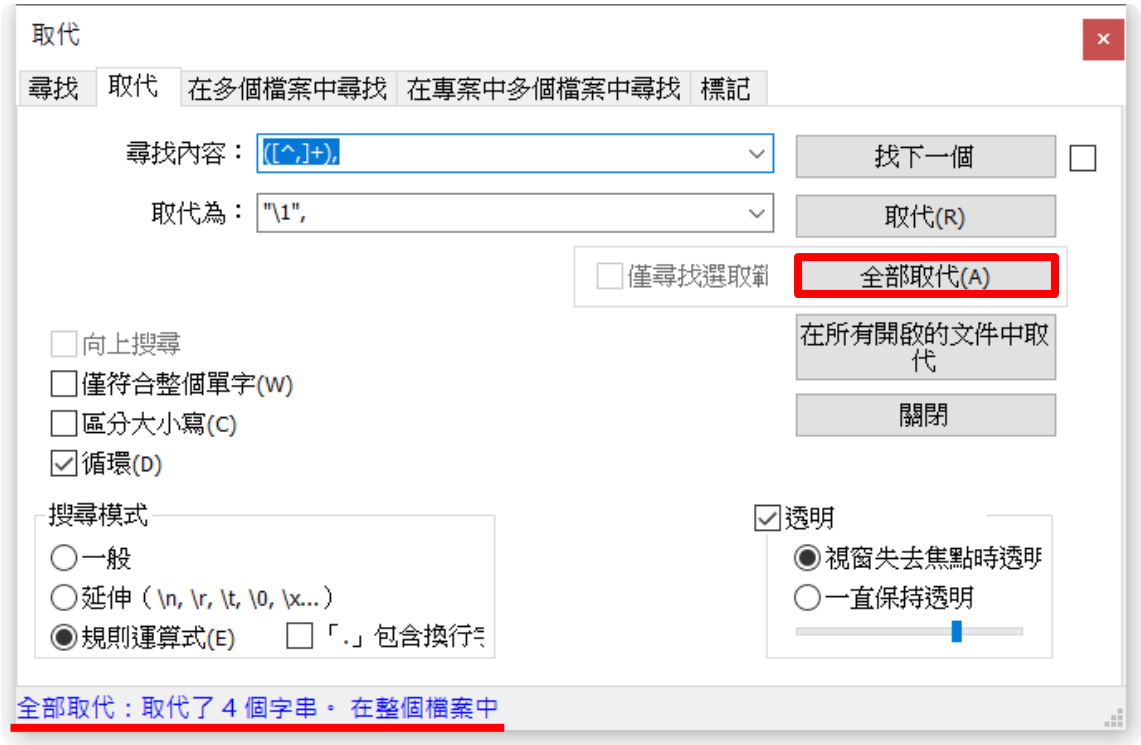
- Separating words



Solution (Ex5.txt)

- Separating words

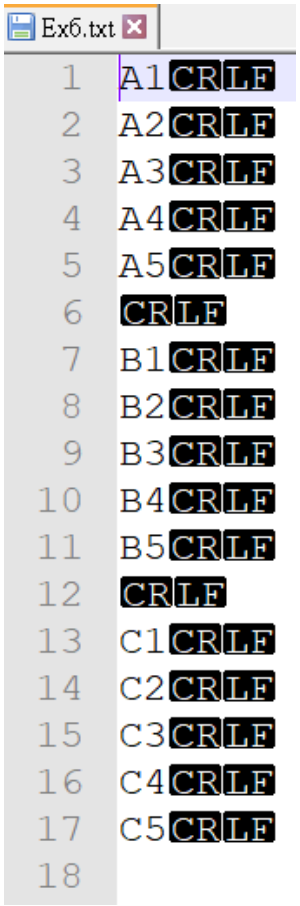
```
([^,]+),
"\1",
```



"AB", "CD", "EF", "GH", CRLF

Exercise (Ex6.txt)

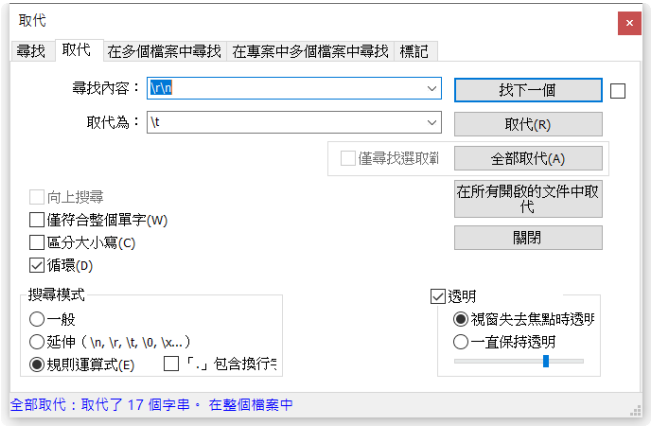
- Transposing data

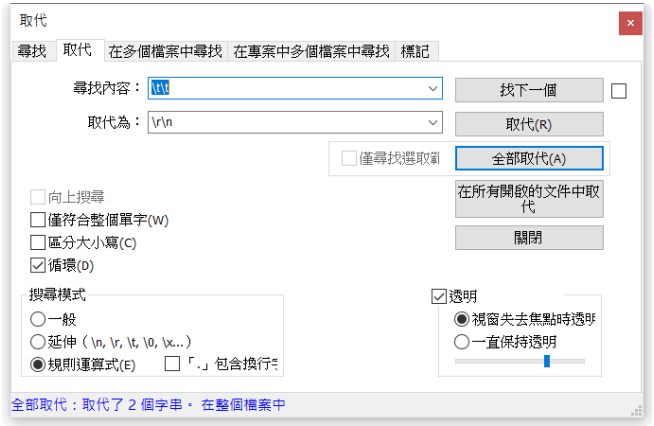


Solution (Ex6.txt)

- Transposing data

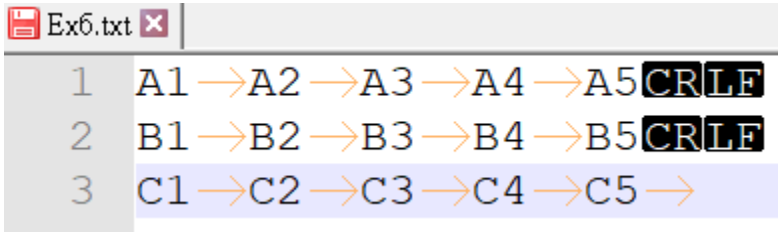
\r\n
\t





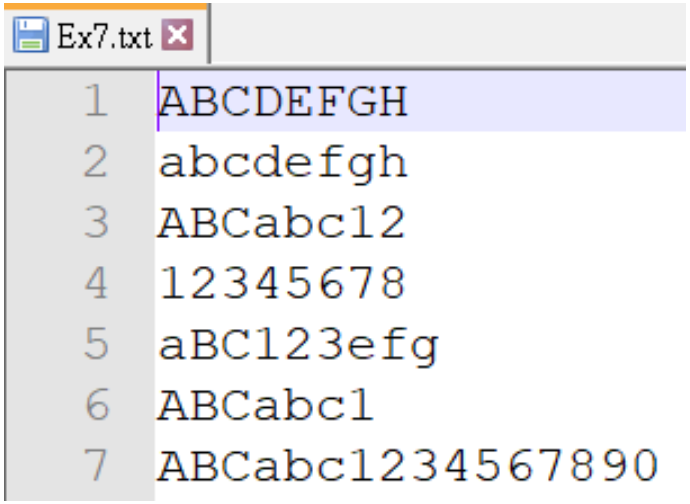
\t\t
\r\n

A1→A2→A3→A4→A5→→→B1→B2→B3→B4→B5→→→C1→C2→C3→C4→C5→



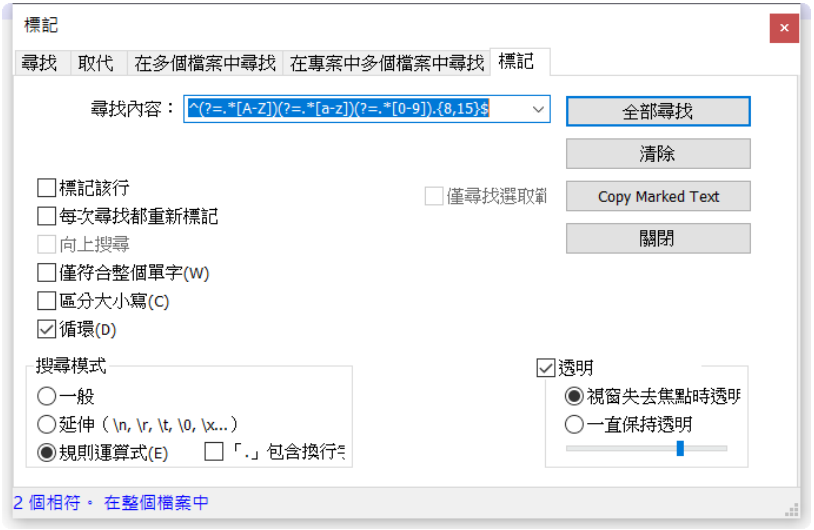
Exercise (Ex7.txt)

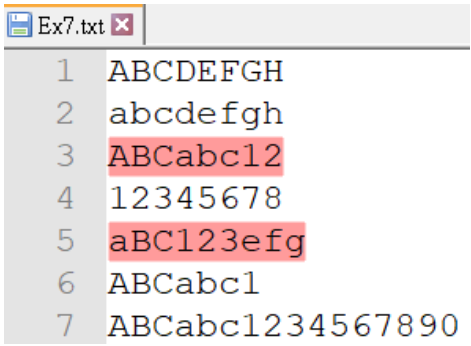
- Validating passwords with a combination of capital letters, small letters and numbers with a length between 8 and 15



Solution (Ex7.txt)

- Validating passwords with a combination of capital letters, small letters and numbers with a length between 8 and 15





^(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9]).{8,15}$

lookahead
lookbehind

Exercise (Ex8.txt)

- Removing leading numbers

Ex8.txt

```
1. 2D MATERIALS
2. 3 BIOTECH
3. 3D PRINTING AND ADDITIVE MANUFACTURING
4. 4OR-A QUARTERLY JOURNAL OF OPERATIONS RESEARCH
5. AAPG BULLETIN
6. AAPS JOURNAL
7. AAPS PHARMSCITECH
8. AATCC JOURNAL OF RESEARCH
9. AATCC REVIEW
10. ABDOMINAL RADIOLOGY
11. ABHANDLUNGEN AUS DEM MATHEMATISCHEN SEMINAR DER UNIVERSITA
12. ABSTRACTS OF PAPERS OF THE AMERICAN CHEMICAL SOCIETY
13. ACADEMIC EMERGENCY MEDICINE
14. ACADEMIC MEDICINE
15. ACADEMIC PEDIATRICS
16. ACADEMIC RADIOLOGY
17. ACAROLOGIA
18. ACCOUNTABILITY IN RESEARCH-POLICIES AND QUALITY ASSURANCE
19. ACCOUNTS OF CHEMICAL RESEARCH
20. ACCREDITATION AND QUALITY ASSURANCE
21. ACI MATERIALS JOURNAL
22. ACI STRUCTURAL JOURNAL
```

Solution (Ex8.txt)

- Removing leading numbers

^[0-9]+\.

也可寫為 ^\d+\.

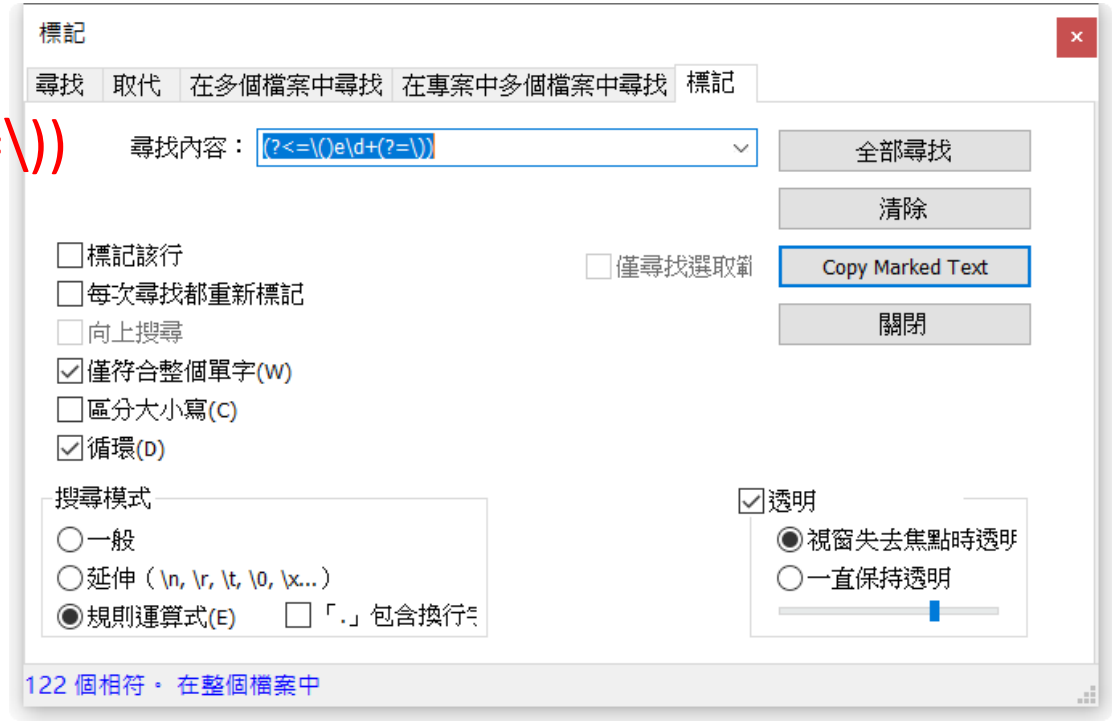
Exercise (Ex9.txt)

- Extracting e-document numbers

Solution (Ex9.txt)

- Extracting e-document numbers

(?<=\()e\d+(?=\))



此例也可寫為 \(\Ke\d+(?=\)) 或 \be\d+\b

lookahead
lookbehind

Motto

Accuracy in vagueness.

* Please study further details, e.g. non-greediness

SECTION III

RE IN BASE R

https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/grep

base (version 3.6.2)

grep: Pattern Matching and Replacement

Description

`grep`, `grepl`, `regexpr`, `gregexpr` and `regexec` search for matches to argument `pattern` within each element of a character vector: they differ in the format of and amount of detail in the results.

`sub` and `gsub` perform replacement of the first and all matches respectively.

Usage

```
grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE,
     fixed = FALSE, useBytes = FALSE, invert = FALSE)

grepl(pattern, x, ignore.case = FALSE, perl = FALSE,
      fixed = FALSE, useBytes = FALSE)

sub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
    fixed = FALSE, useBytes = FALSE)
```

grep grepl sub gsub regexpr gregexpr regexec

https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/regmatches

base (version 3.6.2)

regmatches: Extract or Replace Matched Substrings

Description

Extract or replace matched substrings from match data obtained by `regexpr`, `gregexpr` or `regexec`.

Usage

```
regmatches(x, m, invert = FALSE)
regmatches(x, m, invert = FALSE) <- value
```

Arguments

x	a character vector
m	an object with match data

https://bookdown.org/rdpeng/rprogdatascience/regular-expressions.html

17 Regular Expressions

Watch a video of this chapter

17.1 Before You Begin

If you want a very quick introduction to the general notion of regular expressions and how they can be used to process text (as opposed to how to implement them specifically in R), you should watch this lecture first.

17.2 Primary R Functions

The primary R functions for dealing with regular expressions are

- `grep()`, `grepl()`: These functions search for matches of a regular expression/pattern in a character vector. `grep()` returns the indices into the character vector that contain a match or the specific strings that happen to have the match. `grepl()` returns a `TRUE` / `FALSE` vector indicating

https://github.com/rstudio/cheatsheets/raw/master/regex.pdf

Basic Regular Expressions in R

Cheat Sheet

Character Classes

[[:digit:]] or \\d	Digits; `[0-9]`
\\D	Non-digits; `[^0-9]`
[[:lower:]]	Lower-case letters; `[a-z]`
[[:upper:]]	Upper-case letters; `[A-Z]`
[[:alpha:]]	Alphabetic characters; `[A-z]`
[[:alnum:]]	Alphanumeric characters `[A-z0-9]`
\\w	Word characters; `[A-z0-9_]`
\\W	Non-word characters
[[:xdigit:]] or \\x	Hexadec. digits; `[0-9A-Fa-f]`
[[:blank:]]	Space and tab
[[:space:]] or \\s	Space, tab, vertical tab, newline, form feed, carriage return
\\S	Not space; `[^[:space:]]`
[[:punct:]]	Punctuation characters; `!"#$%&'()*+,-./:;<=>?@[]^_`{|}~`
[[:graph:]]	Graphical characters; `[[:alnum:][:punct:]]`
[[:print:]]	Printable characters; `[[:alnum:][:punct:]\\s]`
[[:cntrl:]] or \\c	Control characters; `\n, \r` etc.

Special Metacharacters

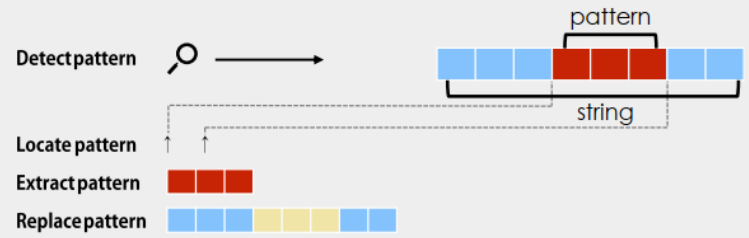
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\f	Form feed

Lookaraounds and Conditionals*

(?=)	Lookahead (requires PERL = TRUE), e.g. `(?=yx)`: position followed by 'xy'
(?!)	Negative lookahead (PERL = TRUE); position NOT followed by pattern
(?<=)	Lookbehind (PERL = TRUE), e.g. `(?<=yx)`: position following 'xy'
(?<!)	Negative lookbehind (PERL = TRUE); position NOT following pattern
?(if)then	If-then-condition (PERL = TRUE); use lookaheads, optional char. etc in if-clause
?(if)then|else	If-then-else-condition (PERL = TRUE)

*see, e.g. http://www.regular-expressions.info/lookaround.html http://www.regular-expressions.info/conditional.html

Functions for Pattern Matching




```
> string <- c("Hiphopopotamus", "Rhymenoceros", "time for bottomless lyrics")
> pattern <- "t.m"
```

Detect Patterns

grep(pattern, string)
`[1] 1 3`

grep(pattern, string, value = TRUE)
`[1] "Hiphopopotamus"`
`[2] "time for bottomless lyrics"`

grepl(pattern, string)
`[1] TRUE FALSE TRUE`

stringr::str_detect(string, pattern)
`[1] TRUE FALSE TRUE`

Locate Patterns

regexpr(pattern, string)
find starting position and length of first match

gregexpr(pattern, string)
find starting position and length of all matches

stringr::**str_locate(string, pattern)**
find starting and end position of first match

stringr::**str_locate_all(string, pattern)**
find starting and end position of all matches

Split a String using a Pattern

strsplit(string, pattern) or stringr::**str_split(string, pattern)**

Extract Patterns

regmatches(string, regexpr(pattern, string))
extract first match `[1] "tam" "tim"`

regmatches(string, gregexpr(pattern, string))
extract all matches, outputs a list
`[[1]] "tam" [[2]] character(0) [[3]] "tim" "tom"`

stringr::**str_extract(string, pattern)**
extract first match `[1] "tam" NA "tim"`

stringr::**str_extract_all(string, pattern)**
extract all matches, outputs a list

stringr::**str_extract_all(string, pattern, simplify = TRUE)**
extract all matches, outputs a matrix

stringr::**str_match(string, pattern)**
extract first match + individual character groups

stringr::**str_match_all(string, pattern)**
extract all matches + individual character groups

Replace Patterns

sub(pattern, replacement, string)
replace first match

gsub(pattern, replacement, string)
replace all matches

stringr::**str_replace(string, pattern, replacement)**
replace first match

stringr::**str_replace_all(string, pattern, replacement)**
replace all matches

Character Classes and Groups

.	Any character except \n
|	Or, e.g. `(a|b)`
[…]	List permitted characters, e.g. `[abc]`
[a-z]	Specify character ranges
[^…]	List excluded characters
(…)	Grouping, enables back referencing using \\N where N is an integer

Anchors

^	Start of the string
$	End of the string
\\b	Empty string at either edge of a word
\\B	NOT the edge of a word
\\<	Beginning of a word
\\>	End of a word

Quantifiers

*	Matches at least 0 times
+	Matches at least 1 time
?	Matches at most 1 time; optional string
{n}	Matches exactly n times
{n,}	Matches at least n times
{n,m}	Matches between n and m times

General Modes

By default R uses *extended* regular expressions. You can switch to *PCRE regular expressions* using `PERL = TRUE` for base or by wrapping patterns with `perl()` for stringr.

All functions can be used with literal searches using `fixed = TRUE` for base or by wrapping patterns with `fixed()` for stringr.

All base functions can be made case insensitive by specifying `ignore.case = TRUE`.

Escaping Characters

Metacharacters (`. * +` etc.) can be used as literal characters by escaping them. Characters can be escaped using `\\` or by enclosing them in `\\Q...\\E`.

Case Conversions

Regular expressions can be made case insensitive using `(?i)`. In backreferences, the strings can be converted to lower or upper case using `\\L` or `\\U` (e.g. `\\L\\1`). This requires `PERL = TRUE`.

Greedy Matching

By default the asterisk * is greedy, i.e. it always matches the longest possible string. It can be used in lazy mode by adding `?`, i.e. `*?`.

Greedy mode can be turned off using `(?U)`. This switches the syntax, so that `(?U)a*` is lazy and `(?U)a*?` is greedy.

Note

Regular expressions can conveniently be created using e.g. the packages rex or rebus.

匯入示範資料

- 利用 Import_tidyverse.r，匯入資料 CD2009.DAT
- 將 CD2009.DAT 等資料檔複製於 D 槽 SampleData 子目錄下，如果欲放置於其他位置，請同時更改 Import_tidyverse.r

```
# 資料與程式碼置於 SampleData 子目錄

# 如果資料檔都固定， Import_tidyverse.r 也改好了
source("D:/SampleData/Import_tidyverse.r")
# CHANGE HERE !!! 變更 .r 檔的置放位置
```

grep(), grepl()

只看cd檔的主診斷碼，有多少筆資料為糖尿病?

```
grep("^250", cd$ACODE_ICD9_1)
length(grep("^250", cd$ACODE_ICD9_1))

grepl("^250", cd$ACODE_ICD9_1)    # TRUE/FALSE
sum(grepl("^250", cd$ACODE_ICD9_1))

grep("^250", cd$ACODE_ICD9_1, value = TRUE)
subset(cd, grepl("^250", ACODE_ICD9_1))
```

Exercise (Ex8.txt)

- Removing leading numbers

Ex8.txt

```
1. 2D MATERIALS
2. 3 BIOTECH
3. 3D PRINTING AND ADDITIVE MANUFACTURING
4. 4OR-A QUARTERLY JOURNAL OF OPERATIONS RESEARCH
5. AAPG BULLETIN
6. AAPS JOURNAL
7. AAPS PHARMSCITECH
8. AATCC JOURNAL OF RESEARCH
9. AATCC REVIEW
10. ABDOMINAL RADIOLOGY
11. ABHANDLUNGEN AUS DEM MATHEMATISCHEN SEMINAR DER UNIVERSITA
12. ABSTRACTS OF PAPERS OF THE AMERICAN CHEMICAL SOCIETY
13. ACADEMIC EMERGENCY MEDICINE
14. ACADEMIC MEDICINE
15. ACADEMIC PEDIATRICS
16. ACADEMIC RADIOLOGY
17. ACAROLOGIA
18. ACCOUNTABILITY IN RESEARCH-POLICIES AND QUALITY ASSURANCE
19. ACCOUNTS OF CHEMICAL RESEARCH
20. ACCREDITATION AND QUALITY ASSURANCE
21. ACI MATERIALS JOURNAL
22. ACI STRUCTURAL JOURNAL
```

Solution (Ex8.txt) : sub()

```
# Removing leading numbers

Ex8 <-
    readLines("D:/R2021_YM/Ref/7B_RegularExpression/Ex8.txt")
Ex8A <- sub("^\\d+\\. ", "", Ex8)
writeLines(Ex8A,
    "D:/R2021_YM/Ref/7B_RegularExpression/Ex8A.txt")
# 請自行變更檔案的置放位置

# 在 R 裡，傳統 RE 的反斜槓(backslash)需寫成兩個 \\
```

Exercise (Ex9.txt)

- Extracting e-document numbers

Solution (Ex9.txt) : regmatches()

```
# Extracting e-document numbers

Ex9 <-
    readLines("D:/R2021_YM/Ref/7B_RegularExpression/Ex9.txt")
Ex9A <- regmatches(Ex9, regexpr("(?<=\\()e\\d+(?=\\))", Ex9,
    perl = TRUE))
writeLines(Ex9A,
    "D:/R2021_YM/Ref/7B_RegularExpression/Ex9A.txt")
# 請自行變更檔案的置放位置 / 假設要找的字串並沒跨行 / 假
    設每行最多只有一個要找的字串
# 在 base R 裡，欲使用 lookahead 與 lookbehind 功能，需加
    上參數 perl = TRUE  (使用 Perl-compatible regexps)
# 此例也可寫成 regmatches(Ex9, regexpr("\\be\\d+\\b", Ex9))
```

SECTION IV

RE IN TIDYVERSE (STRINGR)

https://r4ds.had.co.nz/strings.html#matching-patterns-with-regular-expressions

14.3 Matching patterns with regular expressions

Regexps are a very terse language that allow you to describe patterns in strings. They take a little while to get your head around, but once you understand them, you'll find them extremely useful.

To learn regular expressions, we'll use `str_view()` and `str_view_all()`. These functions take a character vector and a regular expression, and show you how they match. We'll start with very simple regular expressions and then gradually get more and more complicated. Once you've mastered pattern matching, you'll learn how to apply those ideas with various stringr functions.

14.3.1 Basic matches

The simplest patterns match exact strings:

```
x <- c("apple", "banana", "pear")
str_view(x, "an")
```

Copy

```
apple

banana

pear
```

The next step up in complexity is `.`, which matches any character (except a newline):

On this page

View source

Edit this page

https://github.com/rstudio/cheatsheets/raw/master/strings.pdf

String manipulation with stringr : : CHEAT SHEET

The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

Detect Matches

str_detect(string, **pattern**) Detect the presence of a pattern match in a string.
str_detect(fruit, "a")

str_which(string, **pattern**) Find the indexes of strings that contain a pattern match.
str_which(fruit, "a")

str_count(string, **pattern**) Count the number of matches in a string.
str_count(fruit, "a")

str_locate(string, **pattern**) Locate the positions of pattern matches in a string. Also **str_locate_all**. *str_locate(fruit, "a")*

Subset Strings

str_sub(string, start = 1L, end = -1L) Extract substrings from a character vector.
str_sub(fruit, 1, 3); str_sub(fruit, -2)

str_subset(string, **pattern**) Return only the strings that contain a pattern match.
str_subset(fruit, "b")

str_extract(string, **pattern**) Return the first pattern match found in each string, as a vector. Also **str_extract_all** to return every pattern match. *str_extract(fruit, "[aeiou]")*

str_match(string, **pattern**) Return the first pattern match found in each string, as a matrix with a column for each ( ) group in pattern. Also **str_match_all**.
str_match(sentences, "(a|the) ([^]+)")

Manage Lengths

str_length(string) The width of strings (i.e. number of code points, which generally equals the number of characters). *str_length(fruit)*

str_pad(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. *str_pad(fruit, 17)*

str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis.
str_trunc(fruit, 3)

str_trim(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. *str_trim(fruit)*

Mutate Strings

str_sub() **<-** value. Replace substrings by identifying the substrings with str_sub() and assigning into the results.
str_sub(fruit, 1, 3) <- "str"

str_replace(string, **pattern**, replacement) Replace the first matched pattern in each string. *str_replace(fruit, "a", "-")*

str_replace_all(string, **pattern**, replacement) Replace all matched patterns in each string. *str_replace_all(fruit, "a", "-")*

str_to_lower(string, locale = "en")[1] Convert strings to lower case.
str_to_lower(sentences)

str_to_upper(string, locale = "en")[1] Convert strings to upper case.
str_to_upper(sentences)

str_to_title(string, locale = "en")[1] Convert strings to title case. *str_to_title(sentences)*

Join and Split

str_c(..., sep = "", collapse = NULL) Join multiple strings into a single string.
str_c(letters, LETTERS)

str_c(..., sep = "", **collapse = ""**) Collapse a vector of strings into a single string.
str_c(letters, collapse = "")

str_dup(string, times) Repeat strings times times. *str_dup(fruit, times = 2)*

str_split_fixed(string, **pattern**, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split** to return a list of substrings.
str_split_fixed(fruit, " ", n=2)

str_glue(..., .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. *str_glue("Pi is {pi}")*

str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate.
str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")

Order Strings

str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Return the vector of indexes that sorts a character vector. *x[str_order(x)]*

str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Sort a character vector.
str_sort(x)

Helpers

str_conv(string, encoding) Override the encoding of a string. *str_conv(fruit,"ISO-8859-1")*

str_view(string, **pattern**, match = NA) View HTML rendering of first regex match in each string. *str_view(fruit, "[aeiou]")*

str_view_all(string, **pattern**, match = NA) View HTML rendering of all regex matches.
str_view_all(fruit, "[aeiou]")

str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. *str_wrap(sentences, 20)*

[1] See bit.ly/ISO639-1 for a complete list of locales.

RE in base R vs. in tidyverse

base R	tidyverse
grep(pattern, string)	str_which(string, pattern)
grepl(pattern, string)	str_detect(string, pattern)
grep(pattern, string, value = TRUE)	str_subset(string, pattern)
regexpr(pattern, string)	*str_locate(string, pattern)*
gregexpr(pattern, string)	*str_locate_all(string, pattern)*
regmatches(string, regexpr(pattern, string))	*str_extract(string, pattern)* *str_match(string, pattern)*
regmatches(string, gregexpr(pattern, string))	*str_extract_all(string, pattern)* *str_match_all(string, pattern)*
sub(pattern, replacement, string)	str_replace(string, pattern, repl.)
gsub(pattern, replacement, string)	str_replace_all(string, pattern, repl.)
strsplit(string, pattern)	str_split(string, pattern)

* 還有 str_count(), str_view(), str_view_all() 等函數

str_which(), str_detect()

只看cd檔的主診斷碼，有多少筆資料為糖尿病?

```
str_which(cd$ACODE_ICD9_1, "^250")
length(str_which(cd$ACODE_ICD9_1, "^250"))

str_detect(cd$ACODE_ICD9_1, "^250")    # TRUE/FALSE
sum(str_detect(cd$ACODE_ICD9_1, "^250"), na.rm = T)

str_subset(cd$ACODE_ICD9_1, "^250")
filter(cd, str_detect(ACODE_ICD9_1, "^250"))
```

Exercise (Ex8.txt)

- Removing leading numbers

Ex8.txt

```
1. 2D MATERIALS
2. 3 BIOTECH
3. 3D PRINTING AND ADDITIVE MANUFACTURING
4. 4OR-A QUARTERLY JOURNAL OF OPERATIONS RESEARCH
5. AAPG BULLETIN
6. AAPS JOURNAL
7. AAPS PHARMSCITECH
8. AATCC JOURNAL OF RESEARCH
9. AATCC REVIEW
10. ABDOMINAL RADIOLOGY
11. ABHANDLUNGEN AUS DEM MATHEMATISCHEN SEMINAR DER UNIVERSITA
12. ABSTRACTS OF PAPERS OF THE AMERICAN CHEMICAL SOCIETY
13. ACADEMIC EMERGENCY MEDICINE
14. ACADEMIC MEDICINE
15. ACADEMIC PEDIATRICS
16. ACADEMIC RADIOLOGY
17. ACAROLOGIA
18. ACCOUNTABILITY IN RESEARCH-POLICIES AND QUALITY ASSURANCE
19. ACCOUNTS OF CHEMICAL RESEARCH
20. ACCREDITATION AND QUALITY ASSURANCE
21. ACI MATERIALS JOURNAL
22. ACI STRUCTURAL JOURNAL
```

Solution (Ex8.txt) : str_replace()

```
# Removing leading numbers

Ex8 <-
   read_lines("D:/R2021_YM/Ref/7B_RegularExpression/Ex8.txt")
Ex8A <- str_replace(Ex8, "^\\d+\\. ", "")
write_lines(Ex8A,
   "D:/R2021_YM/Ref/7B_RegularExpression/Ex8A.txt")
# 請自行變更檔案的置放位置

# 在 R 裡，傳統 RE 的反斜槓(backslash)需寫成兩個 \\
```

https://stringr.tidyverse.org/reference/str_replace.html
https://readr.tidyverse.org/reference/read_lines.html

Exercise (Ex9.txt)

- Extracting e-document numbers

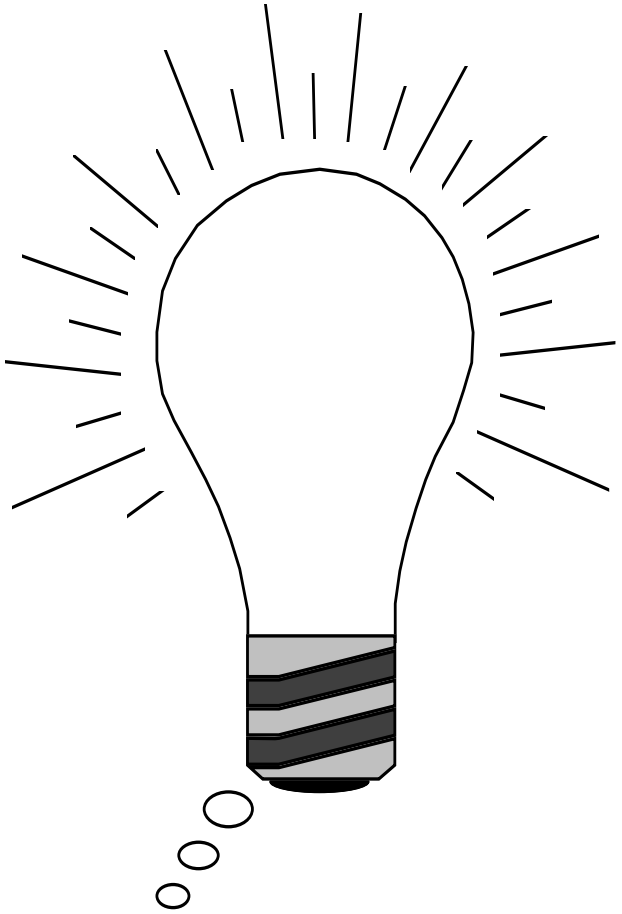
Solution (Ex9.txt) : str_extract()

```
# Extracting e-document numbers

Ex9 <-
   read_lines("D:/R2021_YM/Ref/7B_RegularExpression/Ex9.txt")
Ex9A <- na.omit( str_extract(Ex9, "(?<=\\()e\\d+(?=\\))") )
write_lines(Ex9A,
   "D:/R2021_YM/Ref/7B_RegularExpression/Ex9A.txt")
# 請自行變更檔案的置放位置 / 假設要找的字串並沒跨行 / 假
   設每行最多只有一個要找的字串
# 此例也可寫成 str_extract(Ex9, "\\be\\d+\\b")
```

https://stringr.tidyverse.org/reference/str_extract.html

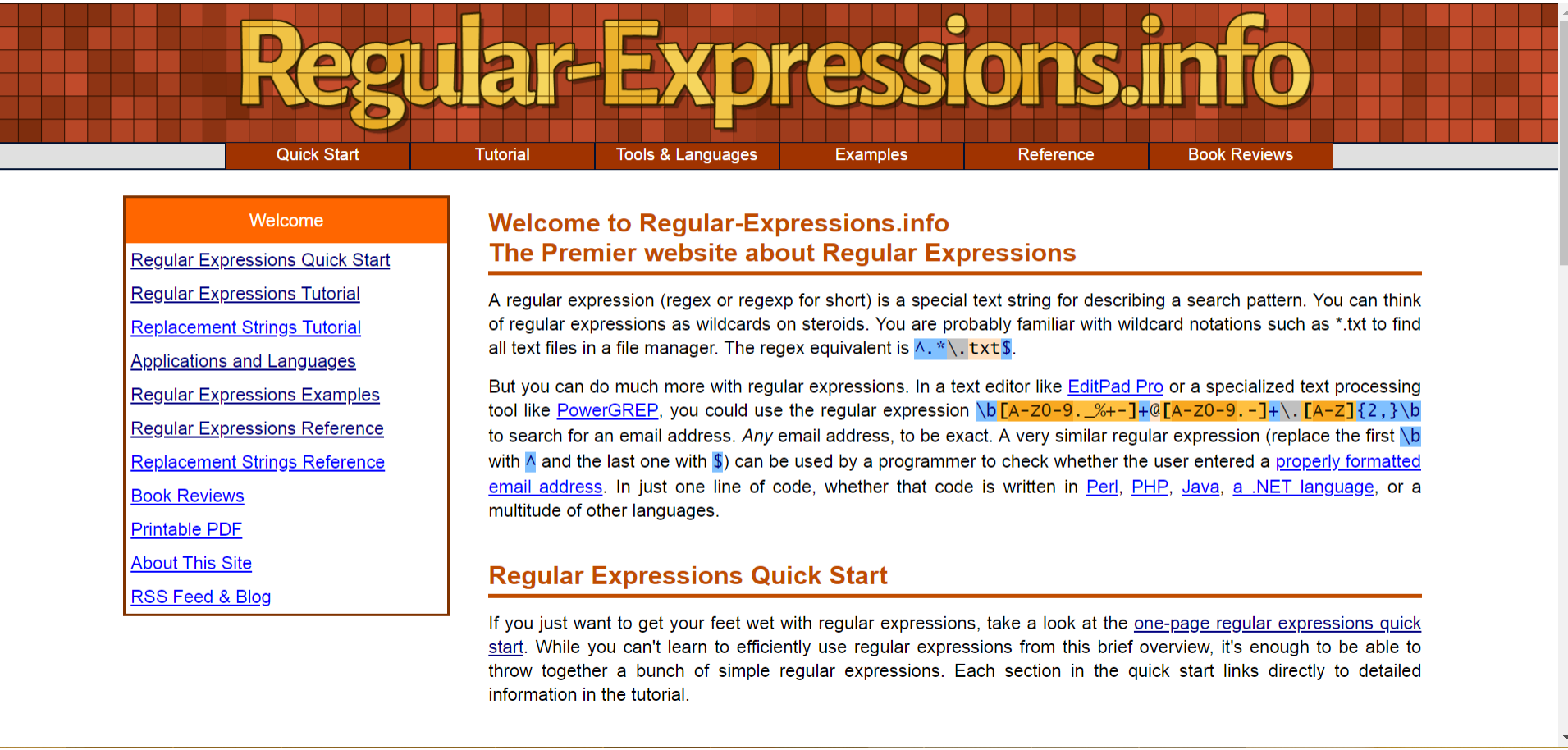
Thanks for Your Attention !



SECTION V

VALUABLE RESOURCES

https://www.regular-expressions.info/



https://regex101.com/