



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

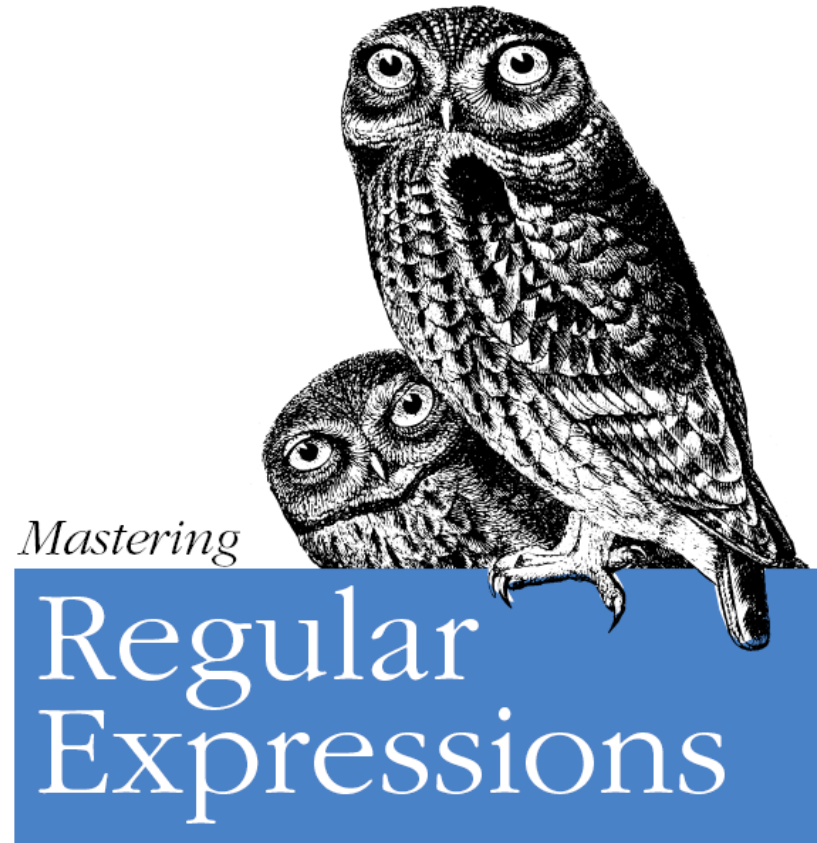
Foundation of Text Mining/Data Science

A Concept/Technique/Sublanguage

Source:

RE Bible

Powerful Techniques for Perl and Other Tools



O'REILLY®

Jeffrey E. F. Friedl



WIKIPEDIA
The Free Encyclopedia

[Main page](#)
[Contents](#)
[Featured content](#)
[Current events](#)
[Random article](#)
[Donate to Wikipedia](#)
[Wikipedia store](#)

[Interaction](#)
[Help](#)
[About Wikipedia](#)
[Community portal](#)
[Recent changes](#)
[Contact page](#)

[Tools](#)
[What links here](#)
[Related changes](#)
[Upload file](#)
[Special pages](#)
[Permanent link](#)
[Page information](#)

Not logged in [Talk](#) [Contributions](#) [Create account](#) [Log in](#)

Article [Talk](#)

[Read](#) [Edit](#) [View history](#)

Regular expression

From Wikipedia, the free encyclopedia

"Regex" redirects here. For the comic book, 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](#).

Contents [hide]

- [Patterns](#)
- [History](#)

I watch three climb before it's my turn. It's a tough one. The guy before me tries twice. He falls twice. After the last one, he comes down. He's finished for the day. It's my turn. My buddy says "good luck!" to me. I noticed a bit of a problem. There's an outcrop on this one. It's about halfway up the wall. It's not a

The match results of the pattern

`(?<=\.)(?=[A-Z])`

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



沒有登入 對話 貢獻 建立帳號 登入

條目 討論 台灣正體 ▾ 漢 漢

閱讀 編輯 檢視歷史

搜尋維基百科

維基台北寫作聚於每月第二個禮拜六舉行，歡迎報名參與

[\[關閉\]](#)

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。

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」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

目錄 [\[展開\]](#)

[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](#)

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}" $\Rightarrow$ pizzapizzapizza
- "(pizza){1,2}" $\Rightarrow$ pizza pizzapizza
- "(pizza){2,}" $\Rightarrow$ 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++

Notepad++ User Manual

Search
Getting started
Programming Languages
User Defined Languages
Editing
Searching
Sessions, Workspaces, and Projects
Task automation with macros
Auto-completion
Function List
Extend functionality with plugins
Plugin Communication
Preferences
Themes
Configuration Files Details
Usage via the command prompt
Shell Extension
Binary Translation

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 `ô`, 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`.

Dialog-based Searching

Find / Replace

"Search results" Window

RightClick commands in the client area of a **Search results** window's tab

Copying text from the **Search results** window

Other commands

Searching in previously-found results (secondary searching)

Find in Files

Highlighting and bookmarking

Dialog-free search/mark actions

Searching

Highlighting

Finding characters in a specific range

Incremental Search

Comparison between "Select and Find Next" and "Find Next (Volatile)"

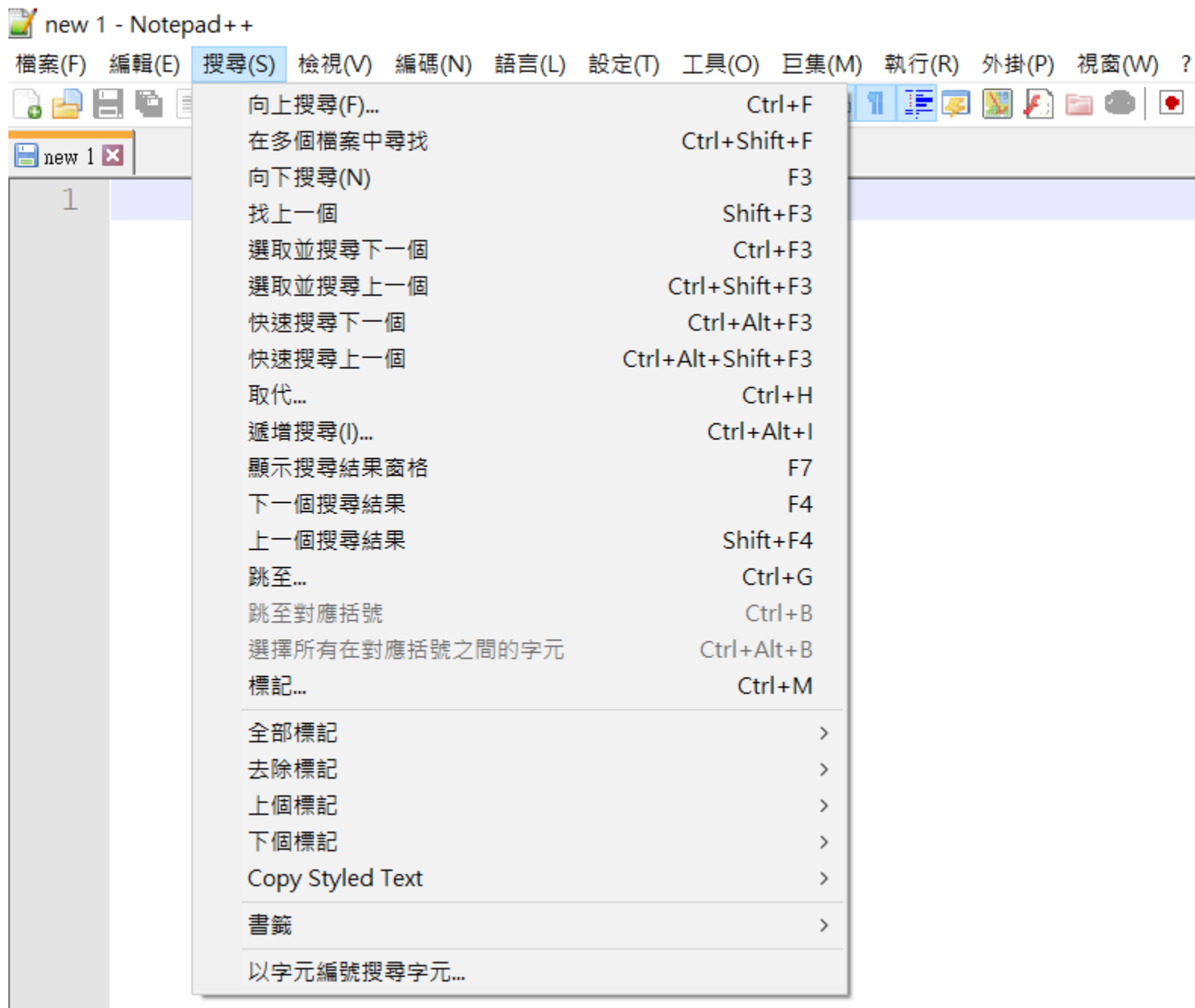
Extended Search Mode

Regular Expressions

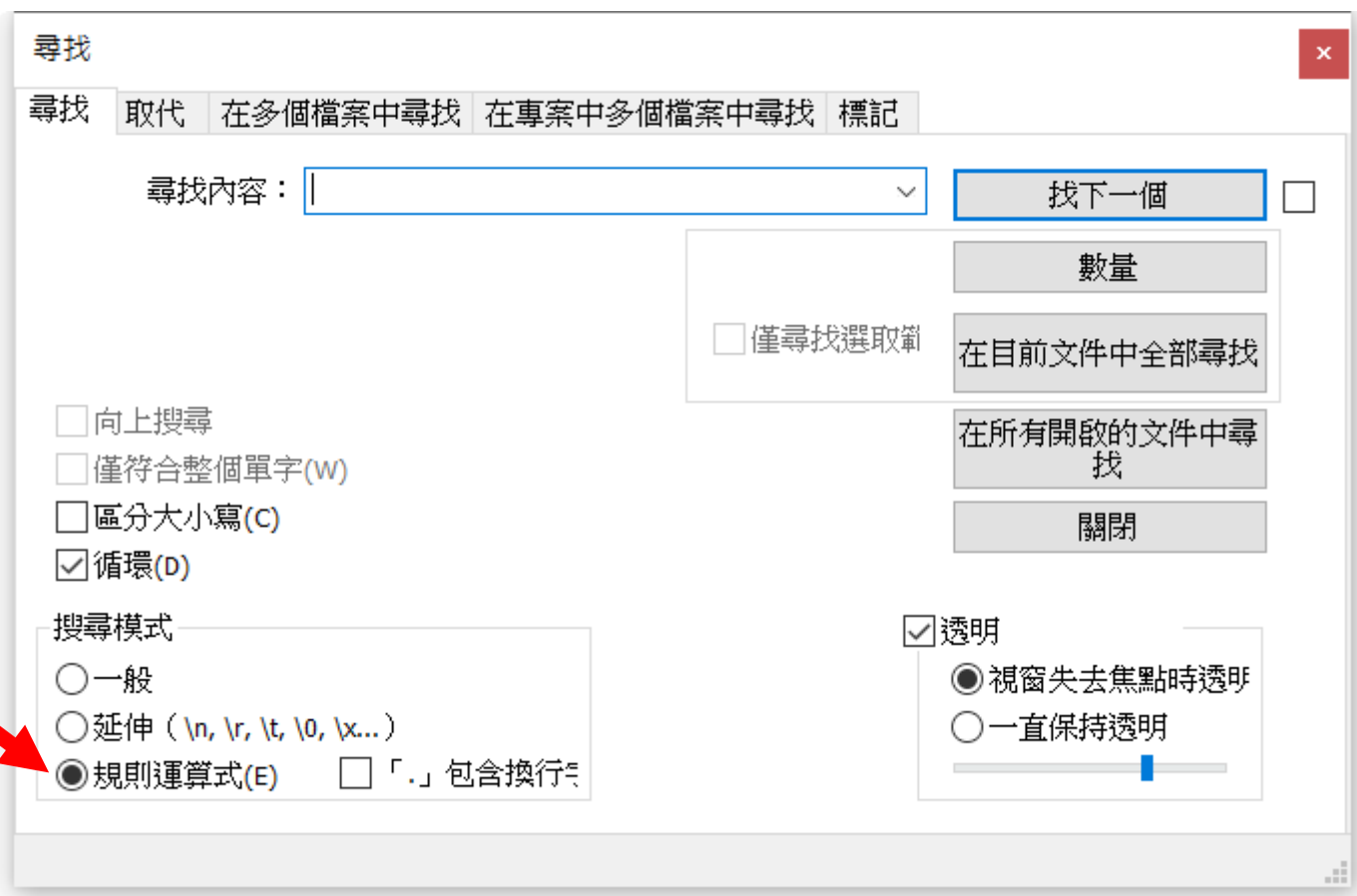
Regex Special Characters

Single-character matches

Non ASCII characters

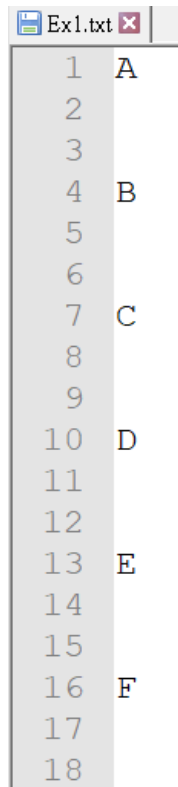


搜尋(S) / 向上搜尋(F)... / [尋找] [尋找] 搜尋模式 / 勾選 規則運算式

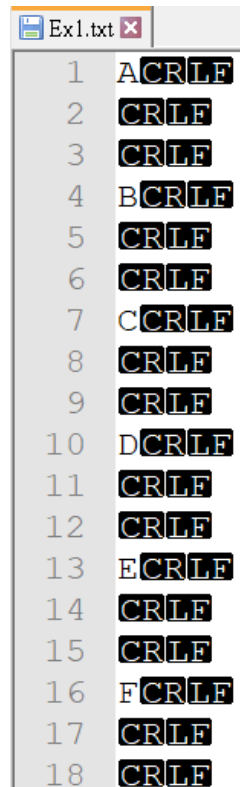


Exercise (Ex1.txt)

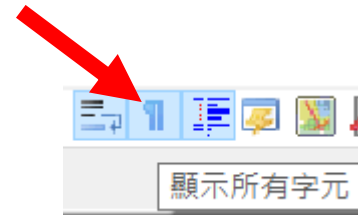
- Removing blank lines 請去除空白行



```
1  
2  
3  
4 B  
5  
6  
7 C  
8  
9  
10 D  
11  
12  
13 E  
14  
15  
16 F  
17  
18
```



```
1 A CRLF  
2 CRLF  
3 CRLF  
4 B CRLF  
5 CRLF  
6 CRLF  
7 C CRLF  
8 CRLF  
9 CRLF  
10 D CRLF  
11 CRLF  
12 CRLF  
13 E CRLF  
14 CRLF  
15 CRLF  
16 F CRLF  
17 CRLF  
18 CRLF
```



Solution (Ex1.txt)

- Removing blank lines 請去除空白行

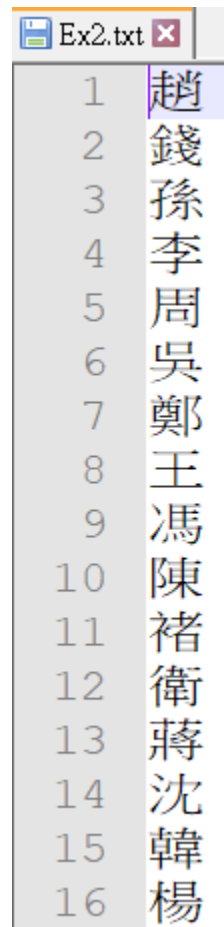
(\r\n)+
\r\n



* 在 TextPad 或其他 PCRE (Perl Compatible Regular Expressions) 軟體，`\r\n` 可寫成 `\n23`

Exercise (Ex2.txt)

- Finding multiple words with one search

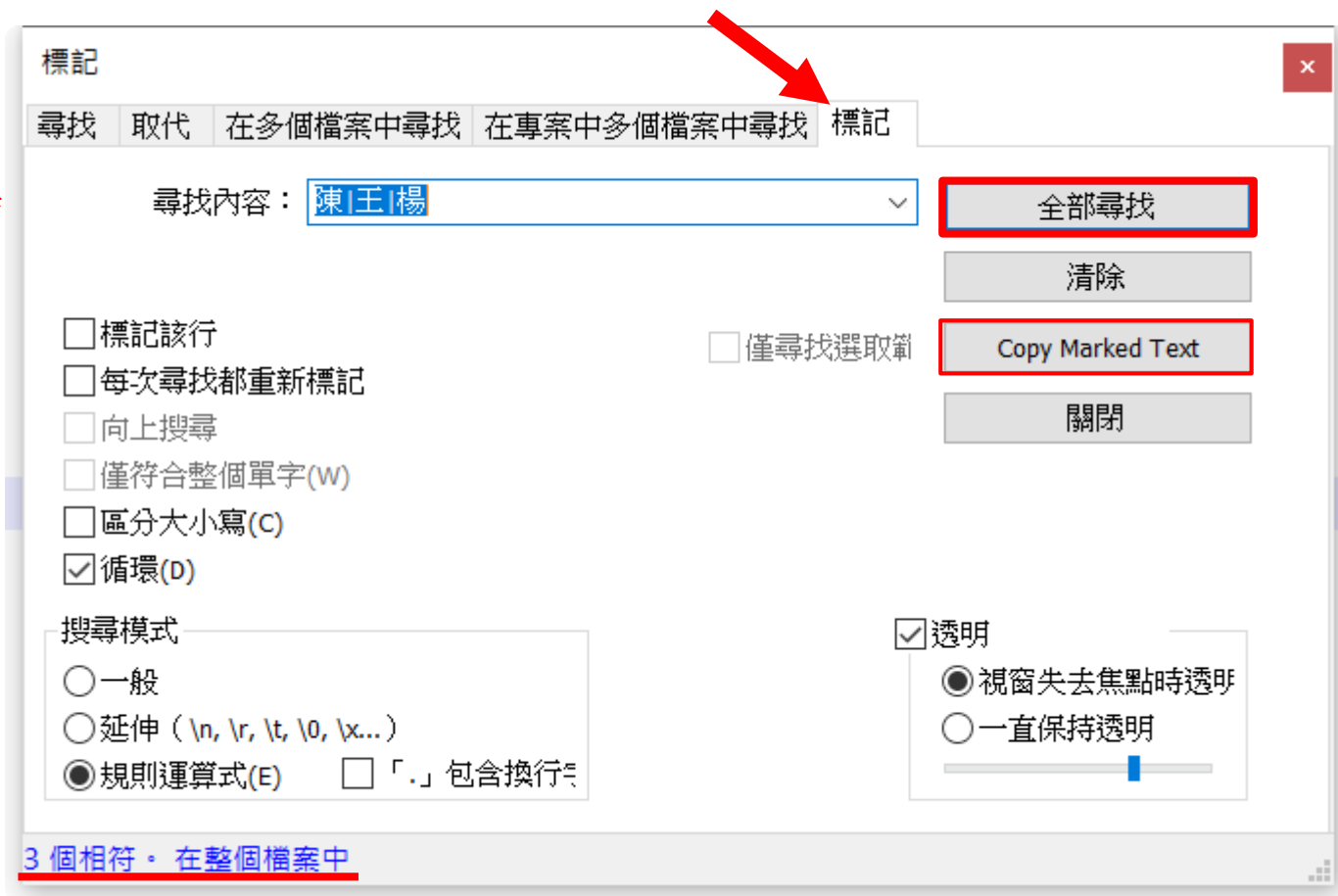


A screenshot of a text editor window titled "Ex2.txt". The window displays a list of 16 Chinese surnames, each preceded by a number from 1 to 16. The surnames are: 趙, 錢, 孫, 李, 周, 吳, 鄭, 王, 馮, 陳, 褚, 衛, 蔣, 沈, 韓, 楊. The first row, "1 趙", is highlighted with a light blue background.

1	趙
2	錢
3	孫
4	李
5	周
6	吳
7	鄭
8	王
9	馮
10	陳
11	褚
12	衛
13	蔣
14	沈
15	韓
16	楊

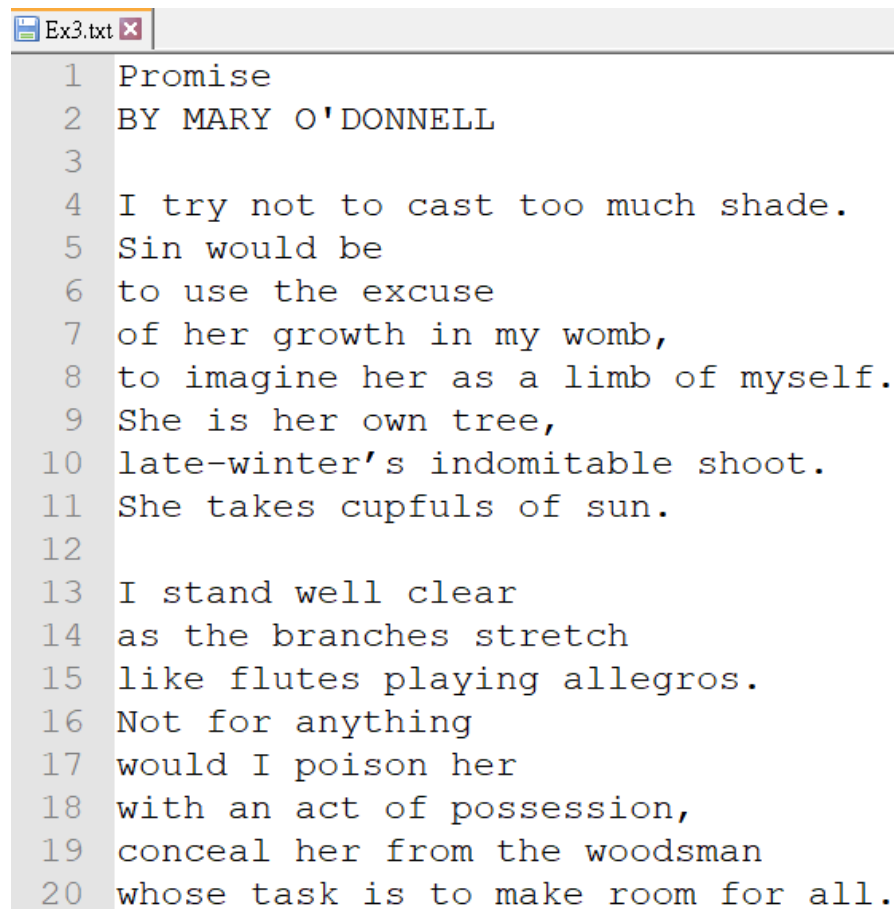
Solution (Ex2.txt)

- Finding multiple words with one search



Exercise (Ex3.txt)

- Searching for lines beginning with a word



```
Ex3.txt x
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\



搜尋結果 - (2 個結果)

搜尋 "^to\b" (找到 2 個結果在 1 的文件中。搜尋文件量： 1)
D:\R2021_YM\Ref\7B_RegularExpression\Ex3.txt (2 個結果)
行號 6: to use the excuse
行號 8: to imagine her as a limb of myself.

Exercise (Ex3.txt)

- Searching for lines ending with a word

Solution (Ex3.txt)

- Searching for lines ending with a word

`\ball\.$`



搜尋結果 - (1 個結果)

搜尋 "`\ball\.$`" (找到 1 個結果在 1 的文件中。搜尋文件量: 1)

D:\R2021_YM\Ref\7B_RegularExpression\Ex3.txt (1 個結果)

行號 20: whose task is to make room for `all`.

Exercise (Ex3.txt)

- Replacing words

Solution (Ex3.txt)

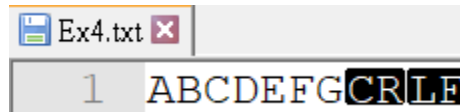
- Replacing words

\ball\b
none



Exercise (Ex4.txt)

- Separating letters

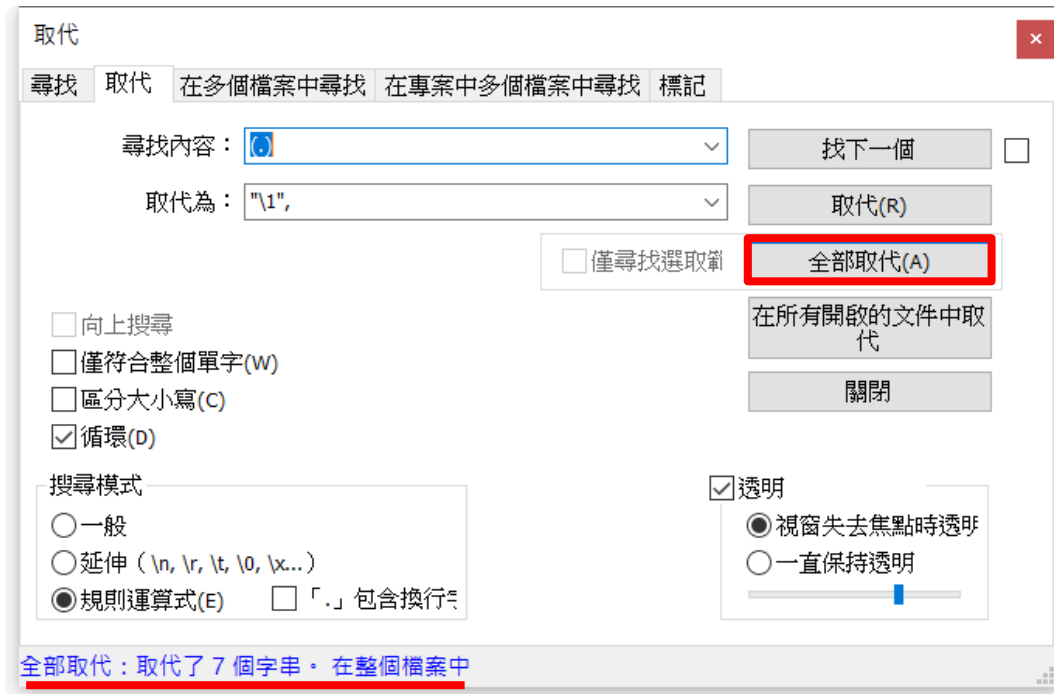


A screenshot of a text editor window titled 'Ex4.txt'. The window contains a single line of text: '1 ABCDEFGCRLE'. The characters 'C', 'R', and 'L' are highlighted in black, and the character 'E' is highlighted in red. The line number '1' is visible on the left side of the editor.

Solution (Ex4.txt)

- Separating letters

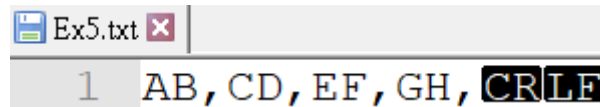
(.)
"\1",



"A", "B", "C", "D", "E", "F", "G", CR LF

Exercise (Ex5.txt)

- Separating words

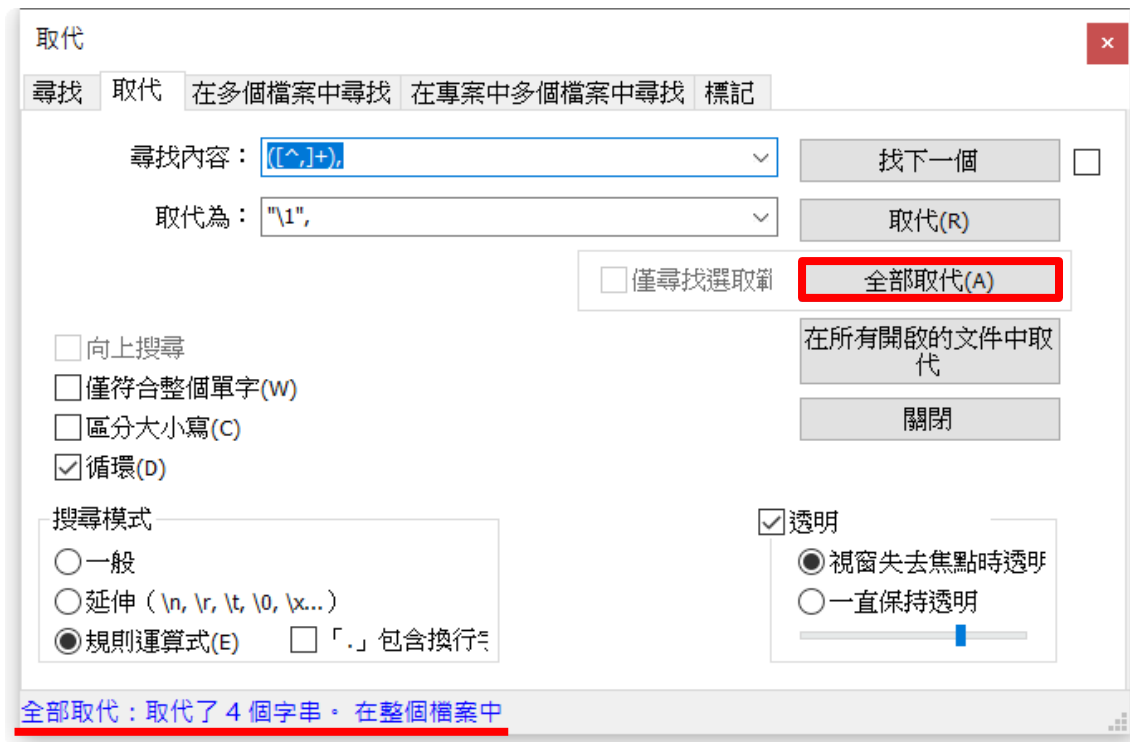


A screenshot of a text editor window titled "Ex5.txt". The window contains a single line of text: "1 AB, CD, EF, GH, CRLE". The text is displayed in a monospaced font. The characters "CRLE" are highlighted in black, indicating they are the focus of the exercise.

Solution (Ex5.txt)

- Separating words

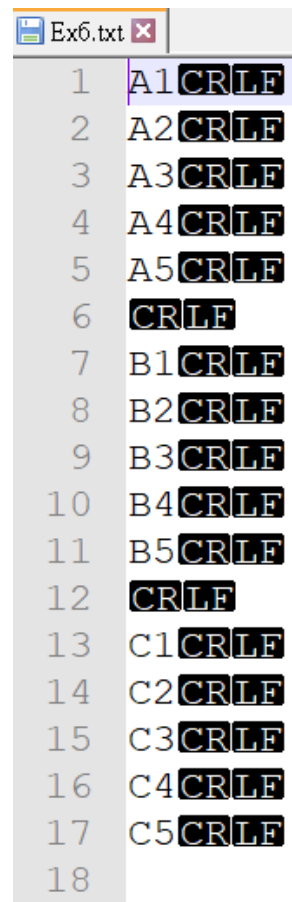
$([^\wedge,]+)$,
"\1",



"AB", "CD", "EF", "GH", CR LF

Exercise (Ex6.txt)

- Transposing data

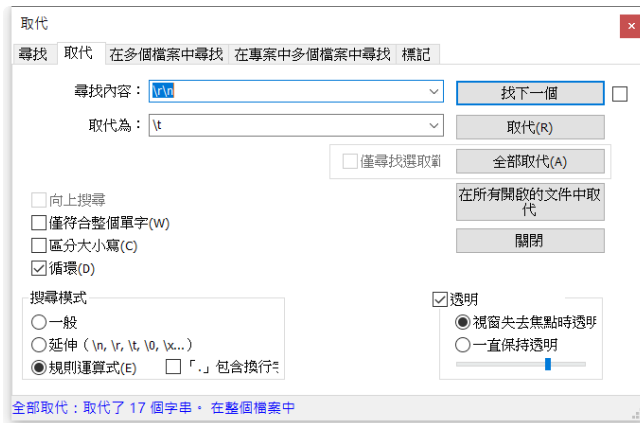


```
1 A1CR LF
2 A2CR LF
3 A3CR LF
4 A4CR LF
5 A5CR LF
6 CR LF
7 B1CR LF
8 B2CR LF
9 B3CR LF
10 B4CR LF
11 B5CR LF
12 CR LF
13 C1CR LF
14 C2CR LF
15 C3CR LF
16 C4CR LF
17 C5CR LF
18
```

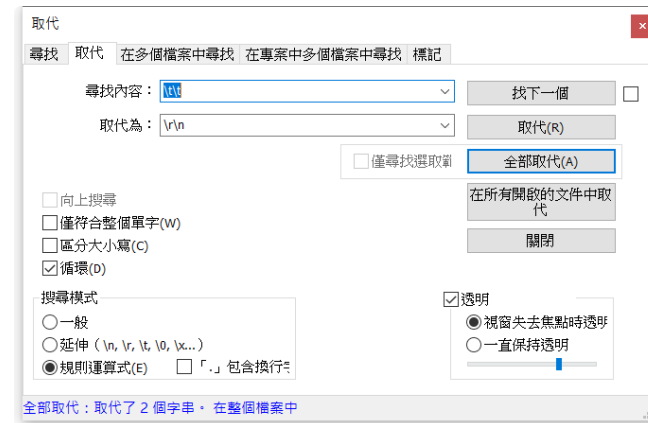
Solution (Ex6.txt)

- Transposing data

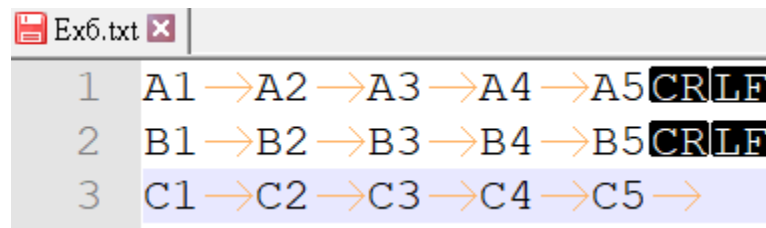
\r\n
\t



\t\t
\r\n

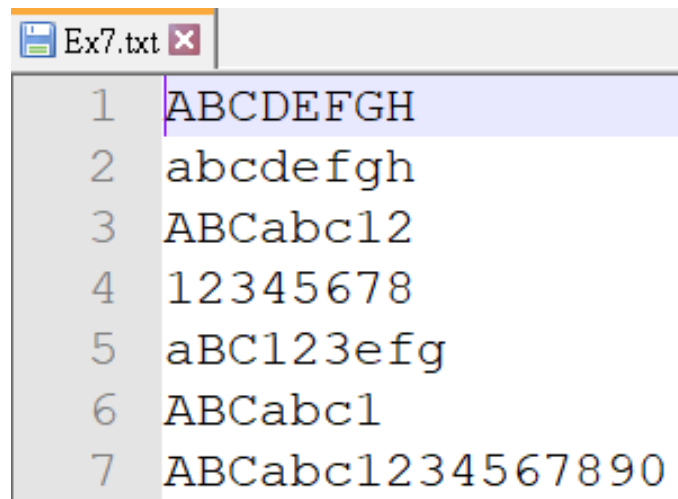


A1 → A2 → A3 → A4 → A5 → → B1 → B2 → B3 → B4 → B5 → → C1 → C2 → C3 → C4 → C5 →



Exercise (Ex7.txt)

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

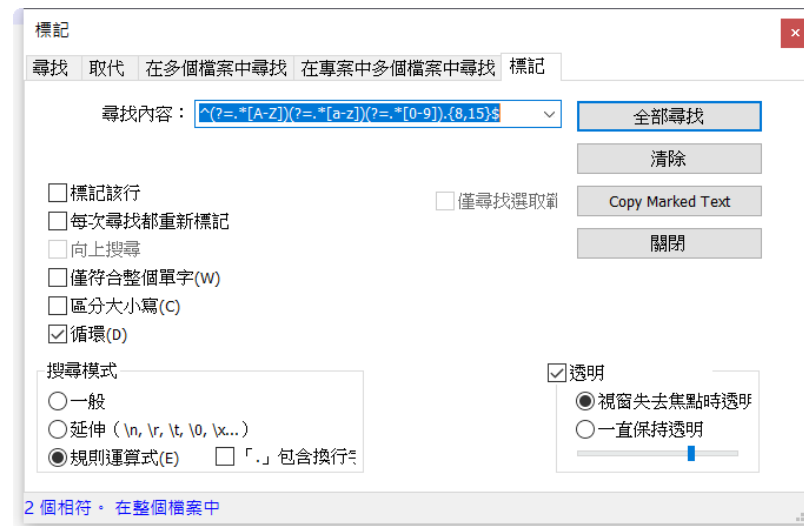


A screenshot of a text editor window titled "Ex7.txt". The window contains a list of 7 password examples, each on a new line and preceded by a line number from 1 to 7. The passwords are: 1. ABCDEFGH, 2. abcdefgh, 3. ABCabc12, 4. 12345678, 5. aBC123efg, 6. ABCabc1, and 7. ABCabc1234567890. The first line is highlighted with a light blue background.

Line	Password
1	ABCDEFGH
2	abcdefgh
3	ABCabc12
4	12345678
5	aBC123efg
6	ABCabc1
7	ABCabc1234567890

Solution (Ex7.txt)

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



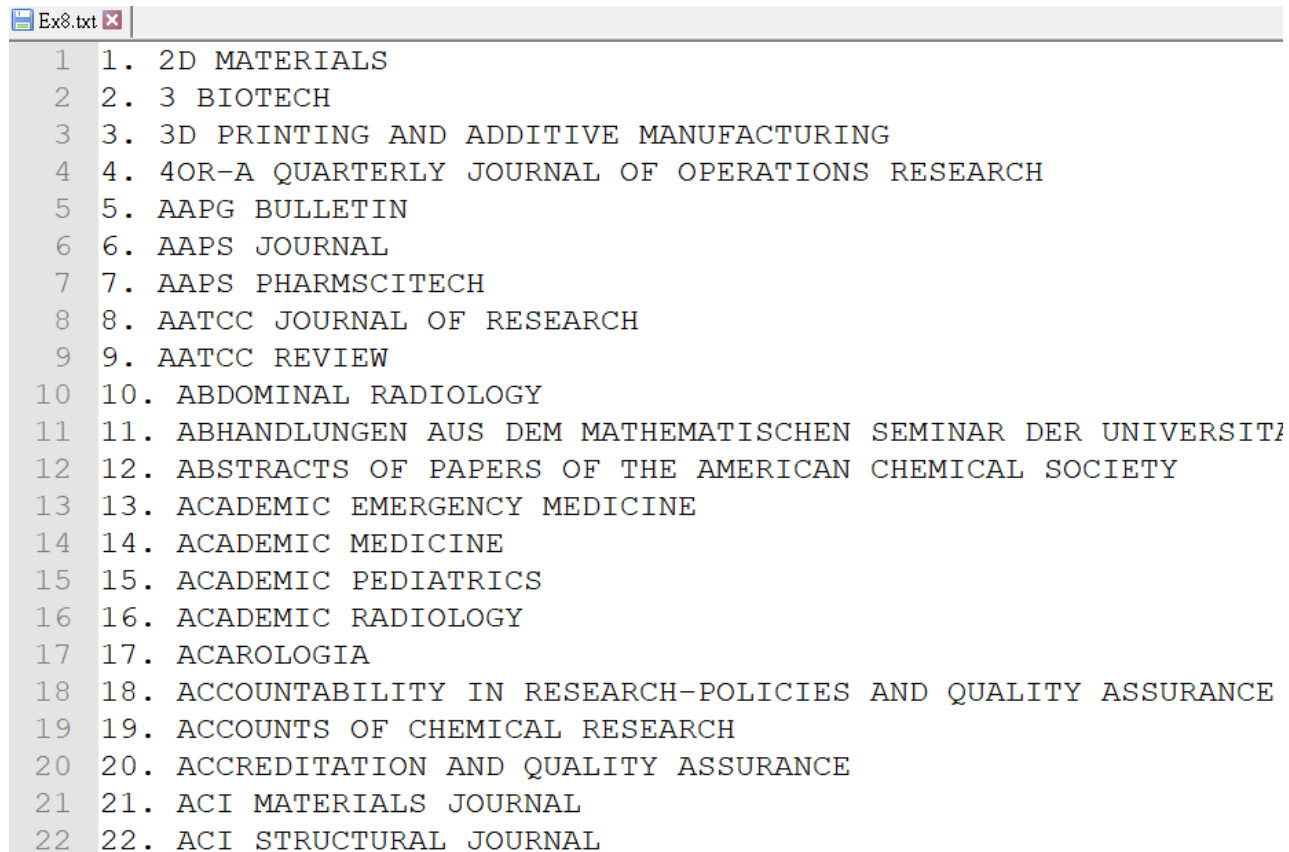
```
Ex7.txt
1 ABCDEFGH
2 abcdefgh
3 ABCabc12
4 12345678
5 aBC123efg
6 ABCabc1
7 ABCabc1234567890
```

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

lookahead
lookbehind

Exercise (Ex8.txt)

- Removing leading numbers

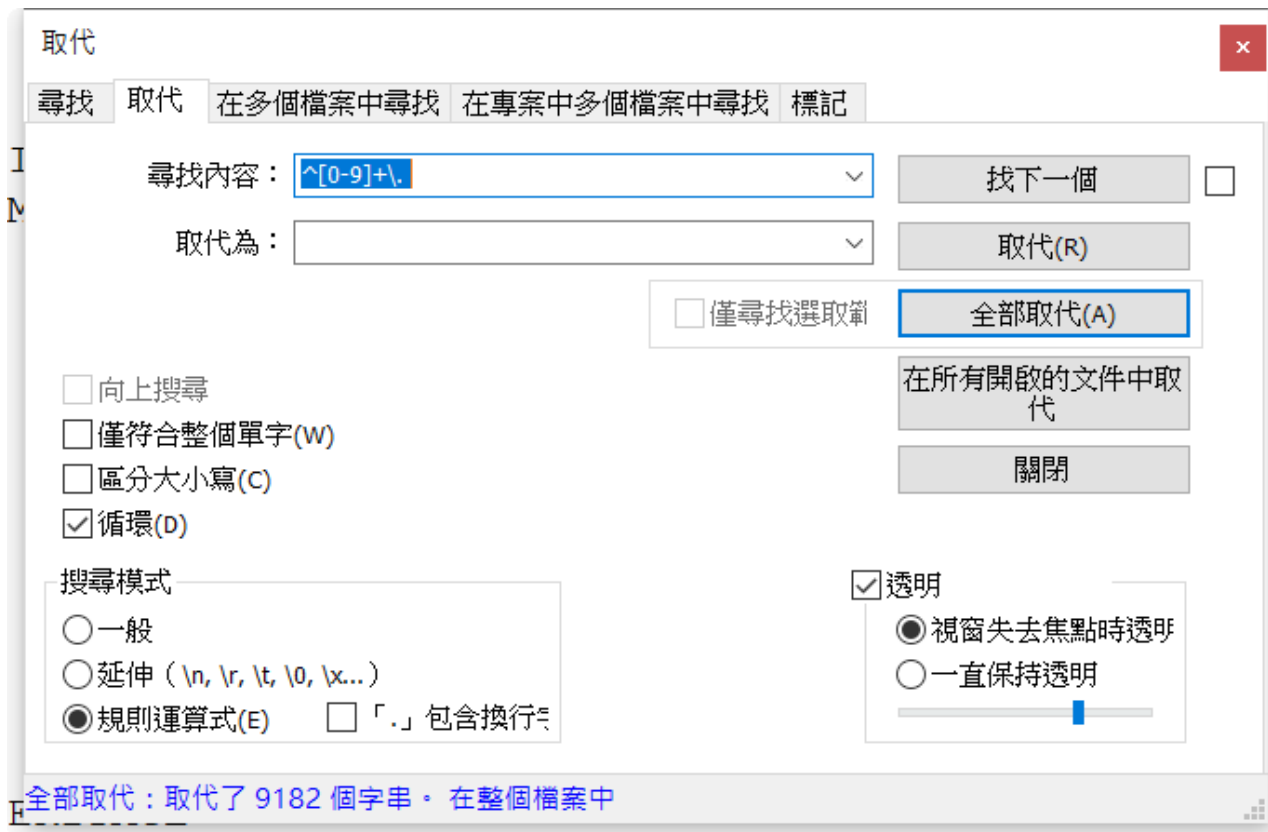


```
1 1. 2D MATERIALS
2 2. 3 BIOTECH
3 3. 3D PRINTING AND ADDITIVE MANUFACTURING
4 4. 4OR-A QUARTERLY JOURNAL OF OPERATIONS RESEARCH
5 5. AAPG BULLETIN
6 6. AAPS JOURNAL
7 7. AAPS PHARMSCITECH
8 8. AATCC JOURNAL OF RESEARCH
9 9. AATCC REVIEW
10 10. ABDOMINAL RADIOLOGY
11 11. ABHANDLUNGEN AUS DEM MATHEMATISCHEN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


Solution (Ex8.txt)

- Removing leading numbers

$^{[0-9]+\backslash.}$



也可寫為 $^{[0-9]+\backslash.}$

Exercise (Ex9.txt)

- Extracting e-document **numbers**

Implementation of the Operating Room Black Box Research Program at the Ottawa Hospital Through Patient, Clinical, and Organizational Engagement: Case Study ([e15443](#))

Sylvain Boet, Nicole Etherington, Sandy Lam, Maxime Lê, Laurie Proulx, Meghan Britton, Julie Kenna, Antoine Przybylak-Brouillard, Jeremy Grimshaw, Teodor Grantcharov, Sukhbir Singh. 11

Social Media Engagement and Influenza Vaccination During the COVID-19 Pandemic: Cross-sectional Survey Study ([e25977](#))

Arriel Benis, Anna Khodos, Sivan Ran, Eugene Levner, Shai Ashkenazi. 25

Development and Evaluation of a Mobile App Designed to Increase HIV Testing and Pre-exposure Prophylaxis Use Among Young Men Who Have Sex With Men in the United States: Open Pilot Trial ([e25107](#))

Katie Biello, Jonathan Hill-Rorie, Pablo Valente, Donna Futterman, Patrick Sullivan, Lisa Hightow-Weidman, Kathryn Muessig, Julian Dormitzer, Matthew Mimiaga, Kenneth Mayer. 40

A Novel Mobile App (Heali) for Disease Treatment in Participants With Irritable Bowel Syndrome: Randomized Controlled Pilot Trial ([e24134](#))

Aaron Rafferty, Rick Hall, Carol Johnston. 247

Development of a Web-Based Mindfulness Program for People With Multiple Sclerosis: Qualitative Co-Design Study ([e19309](#))

Amy-Lee Sesel, Louise Sharpe, Heidi Beadnall, Michael Barnett, Marianna Szabo, Sharon Naismith. 257

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

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)
```

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

<code>x</code>	a character vector
<code>m</code>	an object with match data

17 Regular Expressions

17.1 Before You Begin

17.2 Primary R Functions

17.3 `grep()`

17.4 `grep1()`

17.5 `regexpr()`

17.6 `sub()` and `gsub()`

17.7 `regexec()`

17.8 The `stringr` Package

17.9 Summary

18 Debugging

18.1 Something's Wrong!

18.2 Figuring Out What's Wrong

18.3 Debugging Tools in R

18.4 Using `traceback()`

18.5 Using `debug()`



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()` , `grep1()` : 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. `grep1()` returns a `TRUE / FALSE` vector indicating

Basic Regular Expressions in R

Cheat Sheet

Character Classes

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

Special Metacharacters

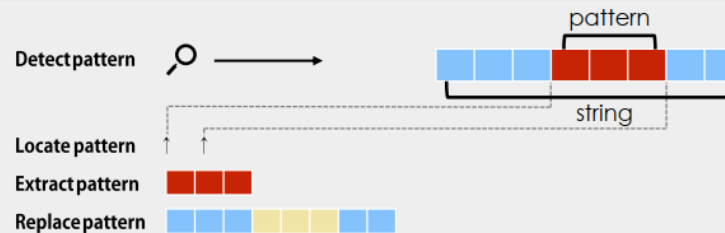
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Tab
<code>\v</code>	Vertical tab
<code>\f</code>	Form feed

Lookaheads and Conditionals*

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

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

Functions for Pattern Matching



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

Detect Patterns

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

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

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

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

Split a String using a Pattern

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

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
```

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

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

Anchors

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

Quantifiers

<code>*</code>	Matches at least 0 times
<code>+</code>	Matches at least 1 time
<code>?</code>	Matches at most 1 time; optional string
<code>{n}</code>	Matches exactly n times
<code>{n,}</code>	Matches at least n times
<code>{n,m}</code>	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
```

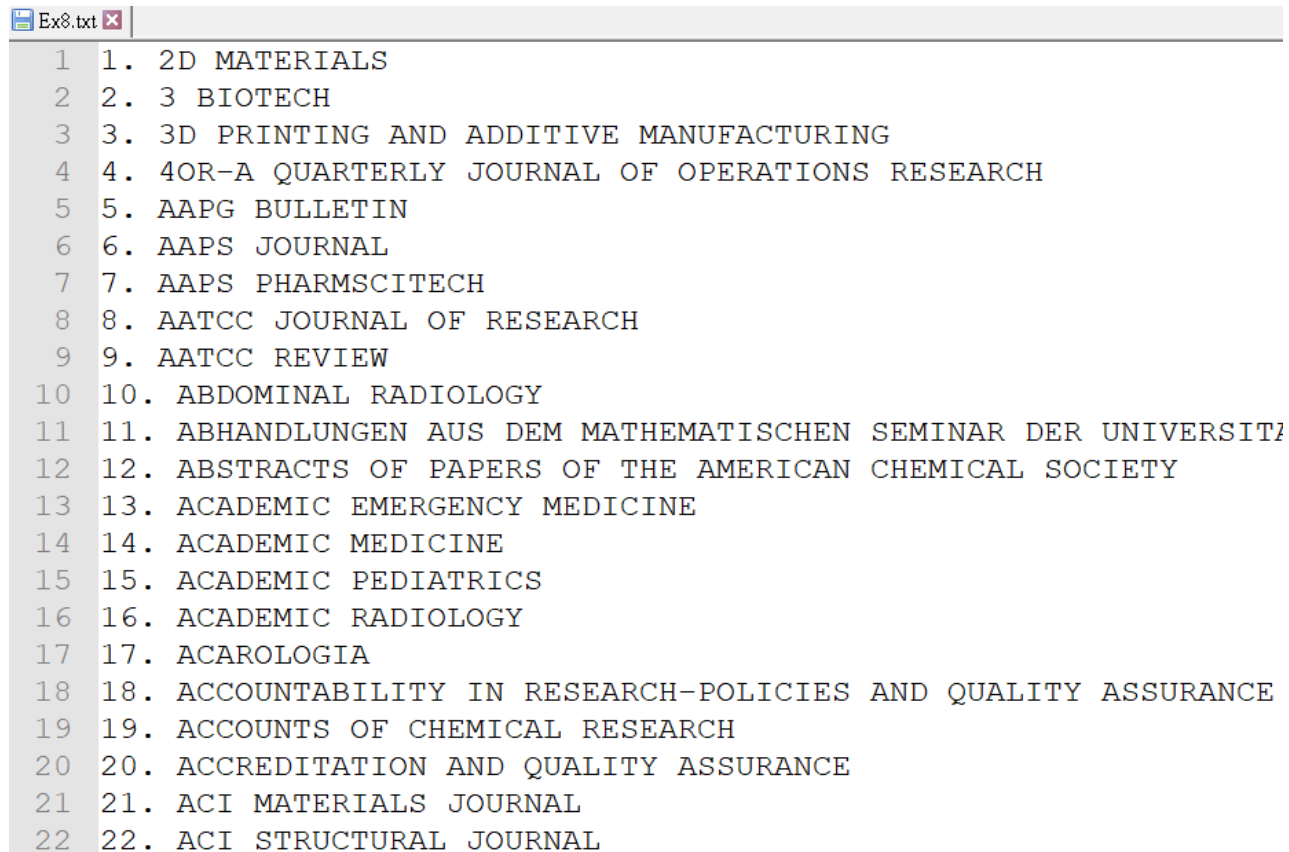
```
sum(grepl("^250", cd$ACODE_ICD9_1))
```

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

```
subset(cd, grepl("^250", ACODE_ICD9_1))
```

Exercise (Ex8.txt)

- Removing leading numbers



```
Ex8.txt x
1 1. 2D MATERIALS
2 2. 3 BIOTECH
3 3. 3D PRINTING AND ADDITIVE MANUFACTURING
4 4. 4OR-A QUARTERLY JOURNAL OF OPERATIONS RESEARCH
5 5. AAPG BULLETIN
6 6. AAPS JOURNAL
7 7. AAPS PHARMSCITECH
8 8. AATCC JOURNAL OF RESEARCH
9 9. AATCC REVIEW
10 10. ABDOMINAL RADIOLOGY
11 11. ABHANDLUNGEN AUS DEM MATHEMATISCHEN SEMINAR DER UNIVERSITÄT
12 12. ABSTRACTS OF PAPERS OF THE AMERICAN CHEMICAL SOCIETY
13 13. ACADEMIC EMERGENCY MEDICINE
14 14. ACADEMIC MEDICINE
15 15. ACADEMIC PEDIATRICS
16 16. ACADEMIC RADIOLOGY
17 17. ACAROLOGIA
18 18. ACCOUNTABILITY IN RESEARCH-POLICIES AND QUALITY ASSURANCE
19 19. ACCOUNTS OF CHEMICAL RESEARCH
20 20. ACCREDITATION AND QUALITY ASSURANCE
21 21. ACI MATERIALS JOURNAL
22 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**

Implementation of the Operating Room Black Box Research Program at the Ottawa Hospital Through Patient, Clinical, and Organizational Engagement: Case Study ([e15443](#))

Sylvain Boet, Nicole Etherington, Sandy Lam, Maxime Lê, Laurie Proulx, Meghan Britton, Julie Kenna, Antoine Przybylak-Brouillard, Jeremy Grimshaw, Teodor Grantcharov, Sukhbir Singh. 11

Social Media Engagement and Influenza Vaccination During the COVID-19 Pandemic: Cross-sectional Survey Study ([e25977](#))

Arriel Benis, Anna Khodos, Sivan Ran, Eugene Levner, Shai Ashkenazi. 25

Development and Evaluation of a Mobile App Designed to Increase HIV Testing and Pre-exposure Prophylaxis Use Among Young Men Who Have Sex With Men in the United States: Open Pilot Trial ([e25107](#))

Katie Biello, Jonathan Hill-Rorie, Pablo Valente, Donna Futterman, Patrick Sullivan, Lisa Hightow-Weidman, Kathryn Muessig, Julian Dormitzer, Matthew Mimiaga, Kenneth Mayer. 40

A Novel Mobile App (Heali) for Disease Treatment in Participants With Irritable Bowel Syndrome: Randomized Controlled Pilot Trial ([e24134](#))

Aaron Rafferty, Rick Hall, Carol Johnston. 247

Development of a Web-Based Mindfulness Program for People With Multiple Sclerosis: Qualitative Co-Design Study ([e19309](#))

Amy-Lee Sesel, Louise Sharpe, Heidi Beadnall, Michael Barnett, Marianna Szabo, Sharon Naismith. 257

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))` 55

SECTION IV

RE IN TIDYVERSE (STRINGR)

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

[14 Strings](#)

[14.1 Introduction](#)

[14.2 String basics](#)

[14.3 Matching patterns with regular expressions](#)

[14.3.1 Basic matches](#)

[14.3.2 Anchors](#)

[14.3.3 Character classes and alternatives](#)

[14.3.4 Repetition](#)

[14.3.5 Grouping and backreferences](#)

[14.4 Tools](#)

[14.5 Other types of pattern](#)

[14.6 Other uses of regular expressions](#)

[14.7 stringi](#)

[View source](#) 

[Edit this page](#) 

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)`

RE in base R vs. in tidyverse

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

* 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
```

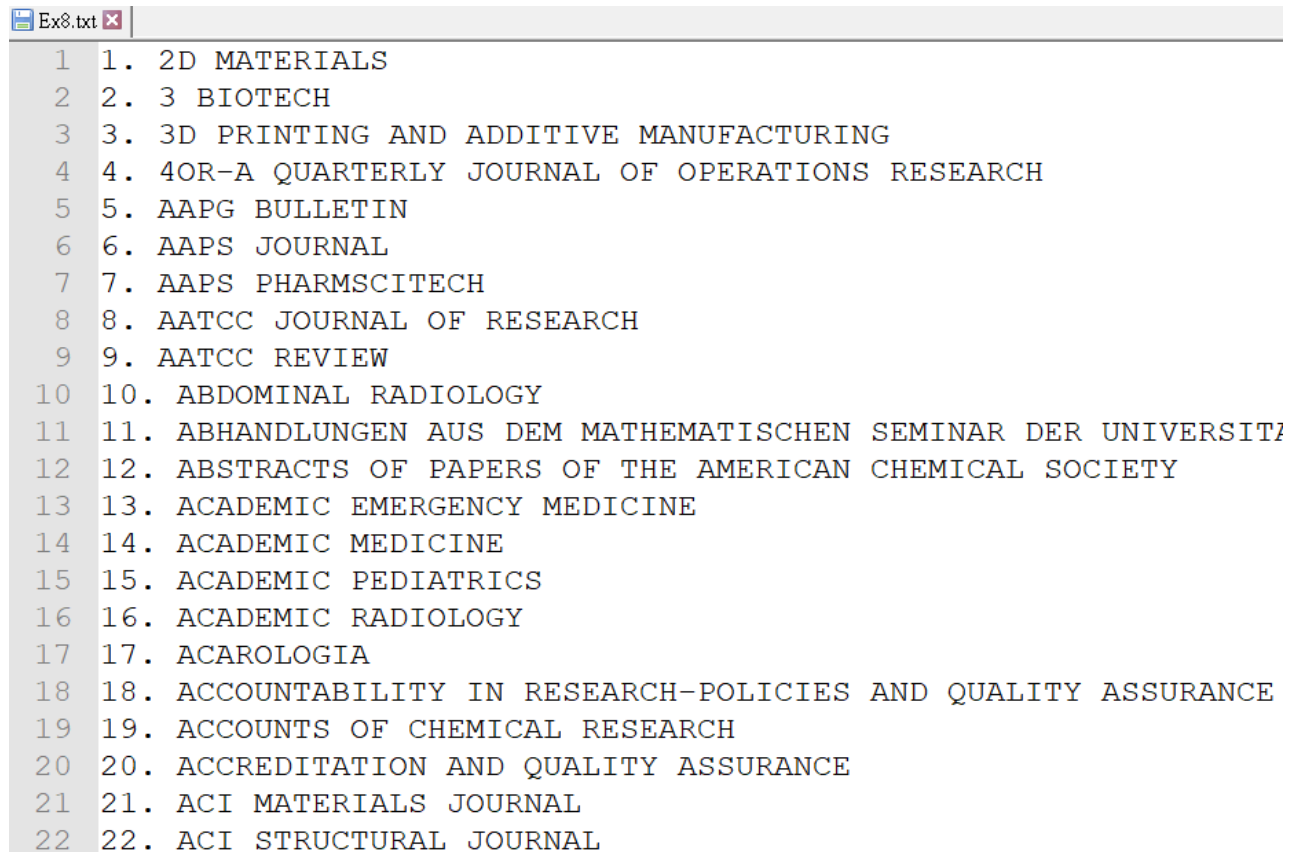
```
sum(str_detect(cd$ACODE_ICD9_1, "^250"), na.rm = T)
```

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

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

Exercise (Ex8.txt)

- Removing leading numbers



```
Ex8.txt x
1 1. 2D MATERIALS
2 2. 3 BIOTECH
3 3. 3D PRINTING AND ADDITIVE MANUFACTURING
4 4. 4OR-A QUARTERLY JOURNAL OF OPERATIONS RESEARCH
5 5. AAPG BULLETIN
6 6. AAPS JOURNAL
7 7. AAPS PHARMSCITECH
8 8. AATCC JOURNAL OF RESEARCH
9 9. AATCC REVIEW
10 10. ABDOMINAL RADIOLOGY
11 11. ABHANDLUNGEN AUS DEM MATHEMATISCHEN SEMINAR DER UNIVERSITÄT
12 12. ABSTRACTS OF PAPERS OF THE AMERICAN CHEMICAL SOCIETY
13 13. ACADEMIC EMERGENCY MEDICINE
14 14. ACADEMIC MEDICINE
15 15. ACADEMIC PEDIATRICS
16 16. ACADEMIC RADIOLOGY
17 17. ACAROLOGIA
18 18. ACCOUNTABILITY IN RESEARCH-POLICIES AND QUALITY ASSURANCE
19 19. ACCOUNTS OF CHEMICAL RESEARCH
20 20. ACCREDITATION AND QUALITY ASSURANCE
21 21. ACI MATERIALS JOURNAL
22 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**

Implementation of the Operating Room Black Box Research Program at the Ottawa Hospital Through Patient, Clinical, and Organizational Engagement: Case Study ([e15443](#))

Sylvain Boet, Nicole Etherington, Sandy Lam, Maxime Lê, Laurie Proulx, Meghan Britton, Julie Kenna, Antoine Przybylak-Brouillard, Jeremy Grimshaw, Teodor Grantcharov, Sukhbir Singh. 11

Social Media Engagement and Influenza Vaccination During the COVID-19 Pandemic: Cross-sectional Survey Study ([e25977](#))

Arriel Benis, Anna Khodos, Sivan Ran, Eugene Levner, Shai Ashkenazi. 25

Development and Evaluation of a Mobile App Designed to Increase HIV Testing and Pre-exposure Prophylaxis Use Among Young Men Who Have Sex With Men in the United States: Open Pilot Trial ([e25107](#))

Katie Biello, Jonathan Hill-Rorie, Pablo Valente, Donna Futterman, Patrick Sullivan, Lisa Hightow-Weidman, Kathryn Muessig, Julian Dormitzer, Matthew Mimiaga, Kenneth Mayer. 40

A Novel Mobile App (Heali) for Disease Treatment in Participants With Irritable Bowel Syndrome: Randomized Controlled Pilot Trial ([e24134](#))

Aaron Rafferty, Rick Hall, Carol Johnston. 247

Development of a Web-Based Mindfulness Program for People With Multiple Sclerosis: Qualitative Co-Design Study ([e19309](#))

Amy-Lee Sesel, Louise Sharpe, Heidi Beadnall, Michael Barnett, Marianna Szabo, Sharon Naismith. 257

Solution (Ex9.txt) : str_extract()

Extracting e-document numbers

Ex9 <-

```
read_lines("D:/R2021_YM/Ref/7B_RegularExpression/Ex9.txt")
```

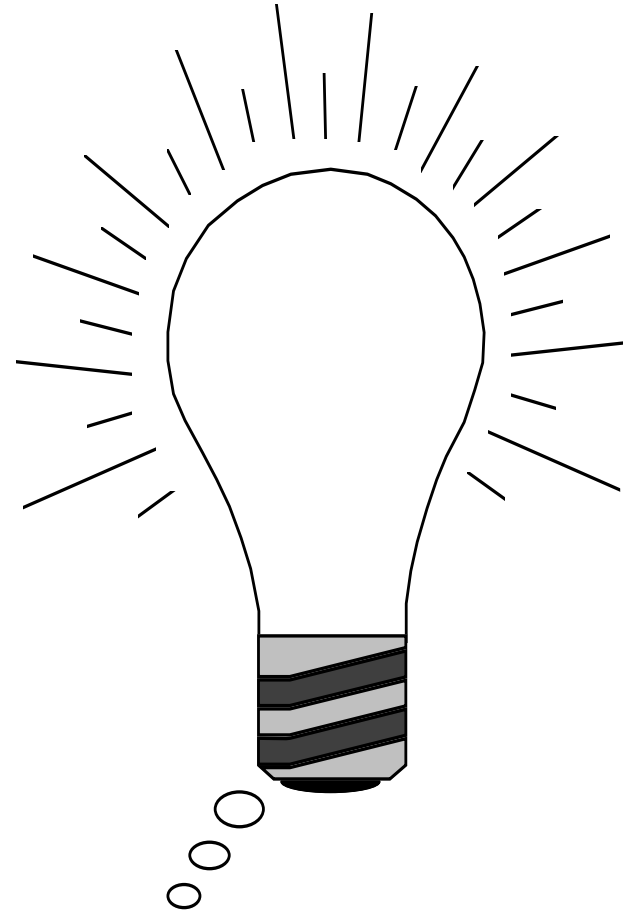
```
Ex9A <- na.omit( str_extract(Ex9, "(?<=\\()e\\d+(?=\\))" ) )
```

```
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")`

Thanks for Your
Attention !



SECTION V

VALUABLE RESOURCES

Regular-Expressions.info

[Quick Start](#)[Tutorial](#)[Tools & Languages](#)[Examples](#)[Reference](#)[Book Reviews](#)

Welcome

[Regular Expressions Quick Start](#)[Regular Expressions Tutorial](#)[Replacement Strings Tutorial](#)[Applications and Languages](#)[Regular Expressions Examples](#)[Regular Expressions Reference](#)[Replacement Strings Reference](#)[Book Reviews](#)[Printable PDF](#)[About This Site](#)[RSS Feed & Blog](#)

Welcome to Regular-Expressions.info The Premier website about Regular Expressions

A regular expression (regex or regexp for short) is a special text string for describing a search pattern. You can think of regular expressions as wildcards on steroids. You are probably familiar with wildcard notations such as *.txt to find all text files in a file manager. The regex equivalent is `^.*\.txt$`.

But you can do much more with regular expressions. In a text editor like [EditPad Pro](#) or a specialized text processing tool like [PowerGREP](#), you could use the regular expression `\b[A-Z0-9._%+-]+\@[A-Z0-9.-]+\.[A-Z]{2,}\b` to search for an email address. Any email address, to be exact. A very similar regular expression (replace the first `\b` with `^` and the last one with `$`) can be used by a programmer to check whether the user entered a [properly formatted email address](#). In just one line of code, whether that code is written in [Perl](#), [PHP](#), [Java](#), [a .NET language](#), or a multitude of other languages.

Regular Expressions Quick Start

If you just want to get your feet wet with regular expressions, take a look at the [one-page regular expressions quick start](#). While you can't learn to efficiently use regular expressions from this brief overview, it's enough to be able to throw together a bunch of simple regular expressions. Each section in the quick start links directly to detailed information in the tutorial.

regular expressions 101

@regex101 donate contact bug reports & feedback wiki

</>

SAVE & SHARE

Save Regex ctrl+s

FLAVOR

</> PCRE (PHP) ✓

</> ECMAScript (JavaScript)

</> Python

</> Golang

TOOLS

Code Generator

Regex Debugger

REGULAR EXPRESSION

no match

/ insert your regular expression here / gm

TEST STRING

SWITCH TO UNIT TESTS >

insert your test string here

SUBSTITUTION

EXPLANATION

An explanation of your regex will be automatically generated as you type.

MATCH INFORMATION

Detailed match information will be displayed here automatically.

QUICK REFERENCE

Search reference

All Tokens

★ Common Tokens ✓

General Tokens

Anchors

Meta Sequences

A single character of: a, b or c [abc]

A character except: a, b or c [^abc]

A character in the range: a-z [a-z]

A character not in the range... [^a-z]

A character in the range: ... [a-zA-Z]

Any single character .

68