Regular Expression（正則表達式）

入門修練 Ver3.50

A Tutor for Regular Expression

正則表達式的教學檔案

參考之原作者：A.M. Kuchling amk@amk.ca http://www.amk.ca/python/howto.

參考之原作者：笑容 http://oo8h.51.net/docs/regular_expression.htm.

中文版編著：陳昌江 AJ CHEN 2006.06～2007.02於剎那搜尋工坊

註1：作者在Emeditor上驗證發展本教材。

註2：由於Regular Expression 也有各種版本差別，EmEditor 所用的標準是 Boost library Regular Expression++.

MEMO: 以GFDL釋放著作聲明

待辦：1.製作優質 WINK 檔，準備放到YOUTUBE等網站。。

2. PPT加演講的短片，上YOUTUB。。。等網站。

3. 開一個 regular expression googlegroups 論壇

正則表達式概述（Regular Expression Introduction）：

Regular Expression 稱為正則表達式，又稱為正規表達式或正規表示式， Regular Expression也可簡寫為REGEXP。

這是一份正則表達式的入門教學檔案（ A Regular Expression tutorial document），提供了正則表達式的基本描述，其目標在於提供一個精致的正則表達式教學文件，作者阿江（陳昌江，AJ）並製作電腦教學的錄影檔案 http://tutor.ksana.tw/，提供更有效率的影音學習。

由於Regular Expression(正則表達式，往後在本文中也簡稱REGEXP)是一種利用特用字元及一些資料的標定、搜尋及替換的規則。REGULAR EXPRESSION並不是一個高深複雜的學問，只是一套符號規則以及這些規則背後可以解決的搜尋與替換問題，這些規則

並不難，然而，一堆符號規則加起來，對於還沒有經驗的人來說，就不是那麼的自然直覺，好像有一點難。

這個現象很像是告訴你a = 蘋果、b = 香蕉、c = 椰子，而A = 蘋果以外全部的水果（這樣叫做a的補集合），而B = 香蕉以外全部的水果，而C = 椰子以外全部的水果。。。這樣符號一多，你就會開始受不了。不是難，而是符號關係不熟而已，所以就只要多練習就會成為自己的東西。本文特別注重案例的分解和練習，只要你忠實地跟個這些案例走一遍，一定會覺得很扎扎實實實地了解Regular Expression(正則表達式）內容。

不過，要真熟練地使用REGULAR EXPRESSION，只憑閱讀是不夠的，「讀者」閱讀本篇文字，只是用來掃瞄REGULAR EXPRESSION大概的範圍，評估要如何來學習。要真正熟習REGULAR EXPRESSION的用法，必須經歷「學習者」的「修練」過程，透過的實際運用的體驗，來真正瞭解熟習並善用REGULAR EXPRESSION這些命令式來發揮其強大的資料處理功能。

所以，我建議的學習程序如下：

（1）「讀者」先瀏覽一遍，第一遍不在於深入熟練，而在瞭解正則表達式(REGULAR EXPRESSION)的語意與功能大要，細節可以暫時忽略。

（2）然後，「學習者」再從練習逐一體驗其運用之道。此乃佛家所謂「聞慧、思慧以及修慧」的三個進程也。最後的「修慧」就是說「唯有親身實踐的體驗」才是「修成正果」之道。

本簡介教材中的所有指令都可以在Emeditor逐一當場互動式的練習測試，每一個搜尋比對吻合的所有資料，全部都會變色讓你一目了然，這是Emeditor的很不錯的人機介面，有如以X光機隨時全面透視每一個指令及參數對所有資料的符合情況。這種即時互動式的體驗對於學習效果很有幫助，當然你也可以在其他有Regular Expression支援的編輯器中演練。

最後，在本文在最後面附有兩個精心設計的練習檔（附錄一、附錄二），讀者請自行拷貝到Emeditor中實際循序逐一練習，Emeditor http://www.emeditor.com 是一個充分支持UNICODE的軟體，因為是日本人做的，所以處理漢字方面很周全，不會像西方人的編輯器總是有很多疏漏。可以免費試用，請自行下載安裝。以便能從這個精心設計每一個練習和觀察中，得到最直接的親身體驗。

最後附錄二是一個以搜尋目標為題的進階研究，其中每一個式子都附有相關的探討和說明，有助於讓你快速的體驗REGULAR EXPRESSION的更進一步運用情形。

0. REGULAR EXPRESSION簡介

Regular Expression (正則表達式 or REGEXP) 在本質上是一種小型的特用語言。這種語言主要是用來做字串的比對、刪除、替換或拷貝重製等工作。

當**資料量龐大**且**形式規律**時，Regular Expressions特別顯得有效益，因為，幾十幾百甚至上千上萬的替換工作，要用人工來作業或巨集(marcro)指令，都幾乎是一個不可能的任務。

學習 REGULAR EXPRESSION也可說是對「文字資料」這種材料，學習一種新的**透視能力**，我們在稍後面的內容中，我們將要對資料碼、資料的「**族群**」以及資料中不可見的碼，建立一種新的識別能力，根據這種識別能力，我們才能展開這一場 REGULAR EXPRESSION的修練之旅。

Emeditor的實驗準備：
在首先安裝Emeditor試用版，這個版本的好處是可以互動式地檢視 REGULAR EXPRESSION搜尋的結果。

1. 打開Emeditor, 在EDIT--FIND功能表(快速鍵Ctrl-F)中勾選Regular Expression 選項。

2. 對於大小寫是否視為相同等選擇性項目也要加以勾選確認。

3. 在這功能表中，還可以選擇對多檔同時搜尋，必要時可以打開。

REGULAR EXPRESSION < 搜尋—替換 > 作業中的經驗提醒：
在 REGULAR EXPRESSION最主要的資料整理的應用中，由於 < 搜尋—替換 > 大多是屬於極大量的資料，我們在這種作業過程疏漏、錯誤的可能，然而其結果人並無法直觀察覺或驗證。因此，在資料搜尋替換的過程中，要注意在作業前以流水號等方式，進行版本備份，以便可以在發現錯誤時，退回正確的版本。這個版本存留的過程要到整個作業完成無誤時，才將所有歷史版本一起刪除，現在硬碟空間龐大，不需要為了疏忽或省空間而後悔。

1. 認識 REGULAR EXPRESSION（正則表達式）

1.1 認識幾個基本詞彙

為了更準確地傳達，讀者對於以下幾個籠統的概念術語必須先加以釐清，以便可以更精準地溝通本文中的文意。

字集

字集就是各國所制訂的字元數碼的集合。 例如 英文字母的**A** 字碼是**65**， **B** 是**66**。十進位的**65,66**也可用十六進位的**$41, $66**表示，這是因為十六進位制在電腦上比較適合而自然，十進位是人習慣用法。本文後面為了解釋各種字元，將會交互使用十和十六進位制來表達 。

註：十六進數字有 0 1 2 3 4 5 6 7 8 9 A B C D E F.。超過9以後就用A來代表十，B代表十一，依此類推，十五就是F了。$符號是在會有所混淆時特別標示這個數字是十六進，十進位則大部分不另行標示。

有些地方也會用0x41來代表先前的$41，依各種軟體系統會有不同的規則。 例如16=$10 , 十進位的15 是 $F ，$F再加1就是 $10了。

註：若要將$41轉換回十進位：十六進的 4 是由滿15後近位而來，所以 $41= 4x16 + 1 = 64。

由於對數位資料來說，十六進位在許多時候都比要自然而直覺，因此長期接觸十六進位制以後反而會直接使用 $41 $2A 也不再需要先轉成十進位制。

這裡列舉幾個我們現隨時都在用的字集，這些字集中， 美國國家標準碼**ASCII** 碼是最有名最早的。台灣的正體 **BIG5**碼是為了把英文電腦中文化所推出來的電腦碼，在**BIG5**之前是一個各大單位自己定碼的「萬碼本騰」時代，到了**BIG5**才統一起來，至於大陸的 **GBK** 碼發展較慢一些，因為大陸早期尚未改革開放，所以中文電腦的進展比較落後 。

不過，現在電腦所用的字集在現在和可見的將來，應該已經漸漸地被**UNICODE** **(http://zh.wikipedia.org/wiki/Unicode)** 所漸漸取代了，**UNICODE**是**1990**年代才發展出來的萬國碼，主要是把全球的文字的字碼放在一起，各分配一段或幾段字碼空間（就是一段數字碼）來用，這樣在這國際化的的文件流通上，就可以省去因各國字集的字碼重疊而需要進行轉碼的麻煩了。

作者註：作者乃是從事中文資料及搜尋引擎的事業，作者要特別要提醒，**UNICODE** 對中文漢字的本質表達來說，是一個史性的錯誤，請參考http://docs.google.com/Doc?id=dgnqz2jv_2dg76zr 「等待新漢碼」一篇有詳細解析。

ASCII CODE – 美國國家標準碼指定了**0~127**的字碼 http://www.asciitable.com/ ，其中包含了不可見的控制碼，這是目前最基本最普遍的碼。

BIG5 – BIG5 是當年台灣所發展的五大電腦廠商共推的標準交換碼。參考：http://zh.wikipedia.org/wiki/%E5%A4%A7%E4%BA%94%E7%A2%BC.
目前（2006年）台灣多數文件仍然沿用，然而新的文件或大型資料庫，已漸漸轉換成UNICODE，因為BIG5碼只有13,053 個字碼，在大量中文資料的時代，已經不敷使用了。

GB – GBK碼是早期中國大陸所用標準碼，經過了多次的演化，目前以GB18030-2000（公元2007年）為標準。

UNICODE – 萬國碼，為了要解決多國語言碼能在一個統一的編碼架構下，於是產生了UNICODE，這也是目前Windows/Linux/MacOS所採用的內部核心碼。自從2000年以來，UNICODE不斷地快速擴充中文字碼的數量，暫時抒解了大部分中文有字無碼的問題。

從字族的角度看字元

為了進行搜尋比對或標示的工作，我們要再把所有字元加以分類，（例如0-9十個字元都是數字，並且命名為數字族或簡稱數族），這樣我們就可以用Regular Expression的命令來

搜尋所有這個類別的字元了。讀者請注意：大量資料的整理工作中，字元族群的分類能力可是很重要的能力，你先會標定這些族群了，你就能夠運用RegExp程式來替你把它們找出來。

字元 – 指ASCII 0～255的所有字元和中文字元（也包中文括符號字元）

數字-- 指0-9的字元（數字是ASCII字元的一部份）

字母 --指英文字母，指 abc….z AB…Z（英文字母是ASCII字元的一部份）

字 -- 字是指被符號、空白隔開的文數字串，例如「 張三 來 台北 」" Hello World!" 。在Regular expression 中 word 的字串的字元有 0-9 a-z A-Z 英文底線以及 中文字元（不包括中文字符）。

英文字 -- 特別指英文的word，如 “Hello world!”，是兩個「英文字」 Hello 和 world.

文數字 -- 指英文字母,中文字元和0-9的數字字元的總稱

字元族 -- Character Class，一些特性相同的字元集合，如「數字(族)」是0-9的字元集合，「大寫字(族)」則是A-Z的所有大寫字母的總稱。能夠準確精細地標稱這些資料族群就是正則表達式（REGEXP）重要的能力。本文後面將會由常識性的字元族群擴展到更精細的字元族群的稱呼。

白字 ── **whitespace**

> 這是在操作資料時，一個隱藏在資料中的重要字族(Character Class)，因為空白、換行、跳格(tab)這類的控制字元都是看不見的，所以稱為「**白字(whitespapce)**」也就是下列字元：

\t \n \r \f \v

因為這些字元都沒有可見圖像字形，所以必須要用 "****" 加上一個字母來標示這個白字元。白字元的集合就是在 REGULAR EXPRESSION中的白字族可以以 **[\t\n\r\f\v]** 來表示 (注意[....]中第一個字元為空白字元)，分別說明如下：

\t Tab ascii 碼 08，水平跳格

\n new line feed ascii 碼 12 ($0C)，跳下一行

\r carrige return ascii碼13 ($0D), 回車，回到一行的開始

\f form feed ascii 碼 14 ($0E), 換到下一頁

（空白字元 ascii碼32）

由於整個 REGULAR EXPRESSION 的所有探討重心，都是針對「要尋找哪些」或「要標定哪些」而來，所以，對於每一個討論，每一個規格，都將以「比到了」「搜到了」「標示了」等來描述（在英文中，也"matched"為主要概念）。

註：由於Windows/Linux/Mac系統的差異，各Editor在實做時對\n 這個白字元通常有特殊處理，對白字元族(White Character Class)也會另外說明，這裡列的是 REGULAR EXPRESSION的標準定義，至於Emeditor實際作法，則會在文中相關處加以說明。

1.2 認識字集：ASCII, BIG5, GBK, UNICODE

亂碼的問題是當今電腦的使用者共同經歷的問題，而這些問題乃導因於各國字集的歷史殘留。而當今全球化資訊流通的結果，使得UNICODE（萬國碼）成為最佳的選擇，然而以當今（2007年）的使用情況來看，各國字碼的使用仍然處在交叉混用的情況，這是因為已經存在的舊系統，更新不易，也使得應用軟體所面對的字碼變化多端，解譯不正確就成了亂碼，我們也只有讓時間來慢慢克服這個問題了。

然而，對一個文字處理的專業人員來說，進一步瞭解這些字集和字碼的實際本質，對於能正確理解 Regular Expression 的操作非常有用，這樣字碼就不再抽象，而是可以透過工具加以檢核的實際資料。

舉例來說，我們把下列最傳統的ASCII + BIG5碼的兩行文字檔資料，用十六進位的字碼印出來（紅色為ASCII，綠色為BIG5）：

AAB CC

漢華華A B

十六進位的列印如下，紅色為可見的字碼，灰色為控制資料展現方式的不可見字元，此處$0D $0A是換行（已知是十六進，所以$符號在表中不標示）：

41 41 42 20 43 43 0D 0A

BA 7E B5 D8 B5 D8 41 20 42

其中 41 就是 A 字母，42 就是 B，20則是沒有字的「空白」字元，而0D 0A則是換行的字碼。 這個檔案是在Windows底下的 ASCII 字碼檔， 若在Linux 或MacOS 中，則換行字碼可能有所異動。

畫了底線的綠色的字碼是16位元（2 BYTE）的BIG5中文字的字碼，$BA7E 就是「漢」這個字的BIG5碼， $A672則是「華」這個字的字碼。

下面這張表是針對一個兩行、只有幾個字元的小檔案的關察分析。
這個檔案內容為：
AAB CC
漢華華A B
下表乃借用這個簡單的檔案，來觀察存檔的字碼和編輯器工作時的字碼。（UNICODE以及各國本有字碼的情況還有很多種，不過和這裡的例子大約相同）

以不同的字碼格式儲存	觀察儲存在檔案中的資校（本表中數字皆為十六進位制）	觀察這個檔案在Ultreditor的工作期間在記憶體中所擺放的資料，基本上以UTF-16LE為碼（本表中數字皆為十六進位制）
AAB CC 漢華華A B 以 ASCII BIG5 格式存檔	41 41 42 20 43 43 0D 0A BA 7E B5 D8 B5 D8 41 20 42 說明： 以ASCII 和 BIG5為編碼來存檔。 其中： 41為字母A，42為B，43 為C， 20為空白(space)， 0D為「回車」(carriage return) 0A 為「下一行」(next line) 0D 0A 在Windows 中構成一個「換行」 BA 7E 意兩個連續BYTE讀出後，組成 $7EBA 的16位元的BIG5碼（第一個讀到的BA這個BYTE放在16位元中的小數部分，這樣叫做 little-endian 方式）。 $7EBA 就是「漢」字的BIG5碼，同理， $D8B5則是「華」字的BIG5碼。 注意：在Linux/MacOS中的「換行」組合就不相同，但是其 換行(newline)的邏輯不變，REGEXP中是以 \n 為邏輯代號，不管在哪一種作業系統都適用。	41 00 41 00 42 00 20 00 43 00 43 00 0D 00 0A 00 22 6F EF 83 EF 83 41 00 20 00 42 00 （編輯器把檔案資料部署到記憶體上時都一律轉成UTF-16LE）

AAB CC 漢華華A B 以 UTF-16LE 格式存檔	41 00 41 00 42 00 20 00 43 00 43 00 0D 00 0A 00 22 6F EF 83 EF 83 41 00 20 00 42 00 說明：以UTF-16LE為編碼來存檔，這是UNICODE儲存格式中的一種，16代表16位元，LE是Little Endian的縮寫，就是先讀到的BYTE當作16位元中的右邊兩個位數，例如上面的 41 00 連讀兩個BYTE組成 $0041這個數。參考：http://zh.wikipedia.org/wiki/UTF-16 其中： 話底線的 00 41 會構成一個16位元的UNICODE字碼 $0041。依此類推， $0020是空白，$000D 是回車，$000A是下一行。而 $6F22 是「漢」的UNICODE 碼。 $83EF 是「華」字的UNICODE 碼。	41 00 41 00 42 00 20 00 43 00 43 00 0D 00 0A 00 22 6F EF 83 EF 83 41 00 20 00 42 00 （編輯器把檔案資料部署到記憶體上時都一律轉成UTF-16LE）
AAB CC 漢華華A B 以 UTF8.TXT 格式存檔	41 41 42 20 43 43 0D-0A E6 BC A2 E8 8F AF E8 8F AF 41 20 42 說明： UTF-8 是一個特別考慮資料精簡的UNICODE表示法，主要適用在傳遞含有大量ASCII CODE的資料，因為她在一般英文字母仍然以一個ASCII BYTE維基處，而遇到各種變化的UNICODE時，則以變動長度的方式來編UNICODE碼。 缺點就是，比較難解讀。 看上面的資料，ASCII 英文字母和換行、空白都沒變，但是遇到中文字反而要用3 BYTE來編碼。 注意：這三個BYTE要透過一個重組規則重組，才能還原回原來的UNICODE！ E6 BC A2 經重組後回到UNICODE $6F22。 E8 8F AF 經重組後回到UNICODE $83EF 。	41 00 41 00 42 00 20 00 43 00 43 00 0D 00 0A 00 22 6F EF 83 EF 83 41 00 20 00 42 00 （編輯器把檔案資料部署到記憶體上時都一律轉成UTF-16LE）

上面這張表就是要經由一個只有兩行幾個字的資料檔，來觀察它在存檔時的資料，並觀察其載入編輯時的字碼部署情形。 有了這些資料本質上的認識，日後在碰到無法理解的現象時，最終的手段，就是清查檔案中的數碼資料加以比對。 所以我要請讀者仔細核對這

個小資料檔中的字碼的情形，盡可能地瞭解到，各字集在數碼資料方面的本質。

1.3 認識各種字元族(Character class)

由於我們將要進入更全面而精準地操作文件中的所有文字資料，所以我們對於文件中可見的字元及不可見字元，都必須再進一步深入認識，這是應用 REGULAR EXPRESSION必要知識。

首先我們先從學著認識各種「字元族(Character class)」開始。

「字元族」(character class，也簡稱字族)的描述法

(「字集」這個詞已經用來指BIG5/UNITCODE等字集，故改叫「字元族(Character class)」以資區別。)

「字元族」(Character Class)就是描述某類**字元的集合。**

例如A–Z的「大寫字」族，0-9的「數字」族。

這裡，我們先從 "[" 和 "]" 這兩個**特用字元(metacharacter)**開始，而"[" 和 "]" 就是在 REGEXP搜尋指令中使用的特用字元(metacharacter)。 把一群字元放入 [ ] 之中，就可以定義一個臨時的集合，我們也可以說是一個臨時的沒有名字的「**字元族**」。

字元族(Character class)描述可以用列舉內容成員：

如 [abc] 就是描述了"a" "b" "c" 三個字母為一個臨時的列舉式的字元族(Character Class)集合，用來做比對時，只要比到了其中任何一個字元都算比到了。

注意：[abc] 是指 a 或 b 或 c 中的任何一個，而不是"abc"字串。

例如：正規表達式搜尋(Regular Expression FIND)：[abc]e

上面這個 [abc]e 式是在搜尋比對一個雙字元的搜尋字串，字串的第一個字元必須是 a b c 中的任何一個，字串中的第二個字元則必須是 e。

搜尋樣本資料（紅色為搜尋比對成功的字串符合）： abdZZceQQbe

再如 [akmg] 就是要比對"a" "k" "m" "g" 四個中的任何一個字母。

「補字元族(Negated Character Classes)」這是我們第二個要認識的特用字元是："**[^]**"

補集合就是「**除了後面的字元族之外的全部都算**」，"^" 這個字元用來標示「字元集的補集合」，這種邏輯式的描述雖然嚴謹但並不直覺（這是RegExp的優點，也是缺點），所以讓我們透過例子來得容易理解：

[^abc] ---就是除了 "a,"b","c" 三個字元之外所有的字元都是（ ，所以"P","Q",「好」。。。等等所有不是 "a,"b","c"的字元--所以都是。 注意： [^abc] 和[abc]都是定義了一個列舉式沒有名字的字族，[^] 定一了整個內涵的字元以外的所有字元。

Regular Expression中所有的特用字元(metacharacter)

下列就是 REGULAR EXPRESSION中所有的特用字元

. ^ $ * + ? { } [ ] \ | ( )

這些字是 REGULAR EXPRESSION 所選用的特用字，其中 "****" （**跳脫或還原字符**）又衍生出

\l \L \u \U \d \D \w \W \s \S 等**字元族代號**(後面中有完整描述)

以及 \a \e \f \r \t \v 等控制字元代號

以及 \Odd （八進；注意\O是字母的'O'）和 \0xXX \x{XXXX} 十六進(兩種格式皆可)的表示法。

跳脫或還原字符——我們第三個要認識是無所不在的特用字元**(metacharacter)**：**"\"**

而上面這些 **^ $ * + ? { } [ ] \ | ( )**

在 REGULAR EXPRESSION 搜尋樣版中若是要用來表達本來的字元，就要在這個字元前面加上"\"來跳脫 REGULAR EXPRESSION而回歸到本來的普通字元。也就是讓"\"後面的特用字元回歸成普通字元，而不在具有 REGEXP 所特別規定的特用功能。

例如：

\[這時，在反斜線後面的 "[" 字元就不再有標示字元族的的特用功能，而回歸的「只是一個普通的 "[" 字元。 "\\" 則表示 要把 "\" 「還原」，就只是一個普通的 "\" 反斜線字元了。相同的，"\^"則讓"^"回歸到只是一個普通的字元符號而不具有 REGEXP中的功能。

例如： **\^ * \\ \. \[**

至於這些特用字元的各個功能則在下面各節中詳述，也是 REGULAR EXPRESSION的重要的部分。

讀者請注意：這種表示法是為了讓電腦處理方便而正確，人需要適應一下，習慣了就好。

字元族(Character Class)也可以用 - 這範圍符號來標示「從。。到。。」的範圍

如 [a-d] 這樣標示和前面的[abcd]效果一樣, 就是"a" "b" "c" "d"。

這標示範圍指令可用來標示[a-z]，代表只要是一個小寫的英文字母，都算比中了（比對成功）；例如"Hello World!"中，標示紅色的小寫字，都是符合[a-z]的字元。

[0-9]則表示只要是一個0,1,2,3,4,5,6,7,8,9 字元都是。

[A-Z]則表示，所有的大寫英文字母都是。

[0-9A-Za-z]則表示所有的數字加上大小寫英文字母都是，也就是所謂的英文的文數字。

字元族（也簡稱字族,Character Class）也可以直接引用 REGEXP內部預設的字族名稱：

因為每次寫[a-z]，[0-9A-za-z]可讀性直覺性比較不夠，因此， REGEXP又內定了一些主要的字族名，並以[:classname:]的形式來表現，這樣我們就可以選擇使用字族名而不必列舉字族的內容。例如，[:digit:]就是[0-9]，[:upper:]則是[A-Z]。另外，這些常用的字族， REGEXP也有訂出了相對的字元族代號，並以"\d"這種最精簡的特用字元形式來代表，例如，[:digit:]就是[0-9]，也是\d，而[:upper:]則是[A-Z]，也可用\u來代表。

下列這張表列出所有 REGEXP預設的字族的代號、列舉式、字族名以及舉例或說明。舉例來說，第一欄\d就是[0-9]也是[:digit:]也就是0,1,2....8,9.

延伸的例子：如果數字之外還要加上幾個符號（小計算機用），那麼就要這樣寫：

[[:digit:]+-*/=.%] 或

[[\d]+-*/=.%] 或 [\d+-*/=.%] （也就是把[:digit:]整個字族看成是一個字元）

1.4 REGULAR EXPRESSION內部預設的字元族代號以及字元族(Character Class)名稱

REGEXP預設字元族代號	REGEXP字元族列舉式描述	預設字元族(Character Class)名[[:classname:]]	說明
\d	[0-9]	[:digit:]	數字族(數族)，也就0123456789
\D	[^0-9] 注意：^對後面的所有列舉都有效	[^[:digit:]]	[0-9]的字元族的補集(Negated Character Class)，也就是非數字以外的所有字元，也就是所有\d以外的字。（也可以寫成[^\d]，也可稱為非數字族)
\l	[a-z]	[:lower:]	小寫字元族(小寫族)，也就是abcdefghijklmnopqrstuvwxyz ，不分大小寫時含中文字
\L	[^a-z]	[^[:lower:]]	非小寫字元族，即所有\l以外的字
\u	[A-Z]	[:upper:]	大寫字元族(大寫族)ABCDEFABCDEFGHIJKLMNOPQRSTUVWXYZ ，不分大小寫時含中文字。(大寫族)
\U	[^A-Z]	[^[:upper:]]	非大寫字元族，所有\u以外的字
	[0-9a-zA-Z]	[:alnum:]	文數字族(文數族)，就是數字和英文字和中文字（不含英文符號、中文符號）
\w	[_A-Za-z_0-9]	[:word:]	文字族，文數字加上”_”底線，含中文字，也等於文數字加上底線 [_[:alnum:]]

\W	[^A-Za-z_0-9]	[^[:word:]]	非文字元族，也就是\w字元族以外的所有字元
\s	[\t] **Emeditor只能確認有空白和跳格，對於換行則統一由\n來確認** **注意：\s的定義隨系統常有變動**	[:space:]	白字元族(白族)， REGEXP一般定義為[\t\n\r\f] 但各編輯器為了配合各系統不同的換行碼Windows (0D0A), Linux(0D), Mac(0A)所以都會有所不同的作法。**本例在Windows系統Emeditor中 \s只確認 \t space兩個。**
\S	[^ \t]	[^[:space:]]	非白字元族，也就是\s以外的字
\C	[^\n]	\n	和' “.” 字相同，除了\n以外的所有字元
		[:unicode:]	和' “.” 字相同，除了\n以外的所有字元
\Q	\Q		在\Q .....\E 之間的所有字都是普通字元。這是為了方便大量的特用字元(metacharacter)的輸入。注意：**本例Emeditor中實驗\Q+?*\E無效**
\E	\E	\E	與 \Q成配對，參見\Q

關於中文字：在 \w 字族中包含了英文大小寫以及中文字。 而 \u \l 在 case-senstive（分大小寫）時保留給準確的英文大小寫，因此不含中文，

所以只有在case-insenstive時才包含中文。（反之可推，\U \L則在case-senstive時包含中文）

[[:alnum:]] 則包含中文字。

1.5 特用字元(metacharacter)的表示

許多控制字元族、白字元族是看不到的，所以必須透過 "\" 這個跳脫字元才能表達。還有 REGEXP所選用的特用字元(metacharacter)族 **. ^ $ * + ? { [ ] \ | ( )** 這些則各有其作用，如果要回歸到普通字元（跳脫特用字元功能）就必須透過 "\" 這個跳脫字元才能回歸普通字元。瞭解和操作這些不可見的字元，正是進階搜尋必要的一種新的「透視力」。

以下就是這些特用字元(metacharacter)在 REGEXP中的 \ 表達方式：

0x07	\a	Bell 鈴聲控制字元.
0x0C	\f	Form feed. 進下一頁
0x0A	\n	Newline character新一行（參考1.2表中資料就很請處） **注意：Emeditor 的\n是以"0D 0A"序列為準。** **在沒啟用 REGEXP用， \n 和 \r\n 都相同** **在啟用 REGEXP時，到底用\r\n, \r, 或 \n則要看檔案原來的情況。**
0x0D	\r	Carriage return. 回車，螢幕光點或印表頭回到行頭
0x09	\t	Tab character. 跳一格
0x0B	\v	Vertical tab. 垂直跳格
0x1B	\e	ASCII Escape character. ASCII所定義的跳脫控制碼 注意：ASCII所定義的ESC碼和REGEXP中的 \並不相同
0xXX	\xXX	XX 是一或多個十六進至的ASCII碼/萬國碼

0xXXXX	\x{XXXX}	{XXXX}內含一個或多個十六進制的ASCII碼/萬國碼

1.6 搜尋替換式中幾個重要特用字元（metacharacter)

搜尋替換的操作中，最常見莫過於把某一個字串替換成另一個字串，那麼應該如何正確指出到底要替換哪一個字串？

其方法就是，我們要先把後面還想要參照的字串樣本括弧起來，

例如：

"AB(CDE)FGH(123)456"

字串中標定了(CDE),(123)兩個樣本， REGEXP會自動地將(...)內容依序放入字串變數\1~\9中供樣本叫用（"CDE"放入了\1，而"123"放入了\2），\1~\9在搜尋或替換時都可以叫用。

注意：本表的準確內容用法，常隨系統會有些微差別，在使用時，通常該系統HELP等相關文件，可查詢相關的說明。

表達符號	說明
\0	先前的整個 REGEXP子式的全部內容不必被( )所包住
\1 - \9	先前的 REGEXP子式子(...) 所搜到的內容將依序被放入了 \1 \2 ... \9 (Emeditor 提供9個字串組變數) ***這個功能太重要了，所以先在這裡舉一個例子： Regular Expression FIND:"a(bc)" Replace:"ZZ\1" 則所有"abc"都會換成"ZZbc"，其中"bc"就是由叫用 \1 中所含有字串樣本而來。
\n	換新行 讀者請注意：Emeditor 的\n是以"0D 0A"序列為准。這個\n的準確用法常隨著系統而不同。 在沒啟用 REGEXP時，使用 \n 和 \r\n 都相同 但在啟用 REGEXP時，到底用\r\n, \r, 或 \n則要看檔案原來的情況。

\r	回車到行頭並換到下一行。 注意：\r所填入的碼因系統有所差異，Windows 放的是0x0D 0x0A連續兩個字元，而Mac系統則放一個0x0D Linux系統則是一個0x0A字元
\t	跳格字元

2． 樣本的比對

OK，對於我將要操作的對象──各種字元族(Character Class)，有了認識之後，我們就可以試著來運用 REGEXP的功能，牛刀小試了。就讓我們從最簡單的比對字元開始。

比對字元

一般定長的字串比對是最簡單的：

"hello" "Hello" 兩字串相同

至於 "Hello" "hello" 大小寫方面的差異算不算相同，則要看編輯器REGEXP功能現在設定的狀態是對大小寫是「精準」(case-sensitive)或「視為相同」(case-insensitive)來比對。通常這都是可以當場設定的。

例： 要找出 "hello "

Regular Expression **FIND:** hello

例： 要找出 "hello" 並替換成"hello world"

Regular Expression **FIND:** hello

Regular Expression **REPLACE:** hello world

更精簡的寫法：

前面提到，已經出現過的字串，如果用(...)括起來，就會自動依序存入\1 \2 ...的字串樣本變數中，因此，也可以利用字串樣本來更精簡表答：

Regular Expression **FIND:** (hello)　　說明：( ) 將 hello 這個字存入 \1 中

Regular Expression **REPLACE:** \1 world （\1 就是叫用先前被存入的樣本 hello，結果還是hello world）

2.1 搜尋比對時的邊界對齊

除了上面的字元族表之中的描述之外，在這裡我們特別提出幾個例子來加強印象，例如：

搜尋獨立的完整字串

\b —— 要在字詞(word)的邊界上，”\bthe\b” 意思就是搜尋單獨的”the”。

例：”... the theater is other ...”　（標紅色的獨立字 the 會被搜到，不是獨立字的綠色the則不會被搜出。）

搜尋靠邊的字串

而 “\bthe” 就是只在左邊有限制：

例：”...the theater is other ...,then dog，there... "

共四個 the 會被搜到在，other裡的the就不會。注意：她們所臨接的 **.，,** (句號、空白、英文逗號，中文逗號都是合法的邊）。

搜尋不限制位置的字串

而搜尋 “the” 沒有加上限制出現的位置時就對任何字串都有效：

例：”… the theater is other …” 三個 the 字串都會被搜到。

搜尋不再邊邊上的字串

\B —— 與 \b 剛好想反互補，就是不可在字詞(word)的邊界上，\Bt\B 意思就是尋t不在兩端的t。

例：”… the theater is other …” 只有不在兩邊的紅t會被搜到，靠邊的綠色t則不會被搜出。

搜尋必須是在一行開頭的字串

^ —— 從行頭開始比對

^From --- 搜尋在一行開頭的“From”字串（不是在行頭的都不算）

例： "From London and from Taipei " 只有行頭的From 會被搜到

搜尋必須是在一行結尾的字串

$ —— 從行尾開始比對

bye**$** --- 搜尋在行尾的 “bye” 字串（不是在行尾的”bye”都不算）

例： "Good Bye! , just say good bye" 只有行尾的bye 會被搜到

例： "Good Bye! , just say good bye." 不是行尾最後字串，所以不會被搜到。

2.2 重複方式的描述

<搜尋──替換>目標字串可以說是我們運用 REGEXP 的重要目的， REGEXP提供了幾個特用字元(metacharacter)來描述字元或樣版的重複的方式，以便可以精準有效地描繪搜尋樣本，這也是 REGEXP功力的重要的一環，因為你若講不出或講錯了「你要找的是什麼」，就沒戲可唱了！

下面是幾個用來描述重複規格的特用字元(metacharacter)。

. —— (dot) 「**除了換行（newline）字元外的所有字元**」 因為是「所有的字」當然是一個最常用的特用字啦！」

例：Regular Expression FIND: r**.**t --- "rat" "rot" "r t" 都可以，但root 就不行，只能一個字元。

? —— 比對一個可能存在的字元 (?之前的字元)，有無都算

Regular Expression FIND: colou?r 就是比對color 和 colo**u**r 兩個字都會被搜出。

注意： "?"若是用在 (*, +, ?, {n}, 之後時，則用來指定保守的「非重複擴張」模式 (不貪婪,non-greedy)，也就是若是可多可少的重複次數時，「以搜尋比對最少次數為準」。

例： A.*B 會搜出： xxxA xxxB xxxxB zzzzB 當有好幾種可能時，REGEXP基本原則上是「盡可能搜出」，在這裡就是到最右邊的B。 (xxxB xxxxB zzzz這一段就是滿足 * 的內容)

A.*?B 會搜出：xxxA xxxB xxxxB zzzzB **?** 就是指定採取保守模式(不貪婪，non-greedy)，也就是只搜到A之後第一個遇到的B就停下來。

***** —— 0或多次的重複。

Regular Expression FIND: cu*****t --- 其中"cter" "cuter" "cuuter" "cuuuuter" 都會搜到，也就是不管幾個 "u" 都可以。

記憶口訣：「閃爍的繁星，時有時無」

+ —— 1或多次的重複。

Regular Expression FIND: ca+t --- “ct” 不會被搜到，而“cuter” “cuuter” “cuuuuter” 這三個會被搜到。“u” 要至少一個。（ 與 * 號類似，只是 + 號要求至少一次 ）

{ } —— 用來定義重複的次數。{ m,n } 在m次到n 次之間都都是。{n} 則是準確的 **n** 次。

Regular Expression FIND: a/**{1,3}**b

代表 “a/b” “a//b” “a///b” 都是，但 “ab” “a////b” 因為不是1 ~ 3所以都不是。

{,3} 表示0 ~ 3次； {3 ，} 表示3 ~ 無限次。

註：m若沒標則表示從0算, n沒有標表示可以到系統的最大數。

註：用範圍符號來標示，則 *= {0,} ，而 + = {1,} 而 ?={0,1}

| ——「樣本二選一」(OR) 「或」，也就是以 | 的兩邊的字串做樣版比對搜尋，只要一個符合，就是成功。

Regular Expression FIND: A|B

"A" or "B" 都對。注意， | 的兩邊的字串當作兩個樣本，然後再 「或」起來，也就是 | 以「最低優序」運作，

請看例子：

Regular Expression FIND: Crow|Servo

「最低優序」就是 "Crow" 和 "Servo"兩個樣版會被「或」，而不是 | 兩邊的字元 w | S。也就是 Crow|Servo 而不是 Crow|Servo 。

2.3 字串樣本群組

這裡要進一步說明在前面1.6節的表中有提到「字串樣本」的使用。

一般的Regular Expression（正則表達式）系統會提供數個的字串樣本變數來容納由"(...)"內的字串。在字串樣本中，也可以放上各種特用字元(metacharacter)就像數學式(sub-expression)一般，而成為一種字串樣本的一部份，而可以在稍後以 \1 \3 的方式加以引用，"\1" 就是叫用先前建立的第一個 (...) 內的字串「樣本1」；"\3" 則表示引用（先前建立的）「樣本3」。

另外，(?...) 形式的有特定作用，不會被編入樣本群組（稍後就會介紹），就會被依序指定到\0 \1 ~ \9 （Emeditor有十個，也有系統提供100組）的字串群組中去。其中 \0 是固有的「所有先前式子中的字元」，不必經由(...)來建立就會自動進入 \0 中。

注意：各個系統的樣本群組容量不同，EMEDITOR提供\0,\1,\2....\9共九個樣本群組的字串變數

你將要發現，在大量的資料搜尋替換中，這些群組樣本的運用威力非常強大。

Regular Expression FIND: Q(ab)*Z --- "ab" 重複 0或多次，

可以搜到 QZ123 QabZ123 QabcdZ123 QababZ123

Regular Expression FIND: (AX)bc\1 這個樣本字串也相當於AXbcAX, 也就是 \1 叫用了先前的AX

Regular Expression FIND: (Regular expression) (正則表達式)

Regular Expression REPLACE： \2 XXX\1 --- 把 「Regular expression正則表達式」替換成「正則表達式 XXXRegular expression」

非常好用的(?....)

下列三個(?....)以?號開頭的就都不會進入字串樣本變數中，看起來形式很類似，但用途的出發點大不相同，而且很好用。不會用這三個字就太可惜了！

我們以Regular expression, REGEXP，正則表達式，正規表達式 四個等同的字串來作為練習的例子：

(?:pattern) --- 多個樣本的比對。看下面例子比較好瞭解：

Regular Expression FIND: (**?:**Regular expression|REGEXP|正則表達式|正規表達式)

其中"**?:**"會讓( )內的字串不進入樣本中，這個案例若直接用 (| | |) 而不加 **?:** 也是可以的。

(?:….)的用途就是「用來列舉多個版本」，配合 "|" 超好用。

(?=pattern) --- pattern 吻合前面才搜尋比對

Regular Expression FIND: file.(**?**=dat) --- 當 "dat" 比到了，前面的"file."才要比。

(?= …)就是用pattern來「**選擇前面的樣本要不要**」（不含pattern）。

(?!pattern) --- pattern吻合，前面的就不比對（跟?=相反的邏輯），(?!....)就是含有pattern時「排除比對」。

Regular Expression FIND: file[.](**?!**exe|bat) --- 若有 "exe" 和 bat 就整個都不要比了。

2.4 實例解說與練習

下面所舉的例子附有分解動作的解說，可以引導你熟悉 **REGULAR EXPRESSION**特用字元(metacharacter)、字族的觀念和運用。

這些練習如果能透過Emeditor，在即時互動的情境下驗證，學習效果最好，所以學者應該將這些案例COPY到Emeditor進行實際的體驗，畢竟，唯有透過「修練」所到達的「修慧」，才是真正我們能力的一部份，而不只是「知慧」而已。

觀念的進化：搜尋也是一種標定

Find 是發現也是「搜尋」，也是一種「標定」的工作，在標定了目標字串之後，就可以依照需要，進行刪除、替換操作。

當我們標示要搜尋 **\u** (也可寫成 [A-Z] 或 [:upper:]) 時，我們就是要從所可搜尋的資料範圍中，**標定**所有的英文大寫字母，也就是把資料分成大寫的和不是大寫的兩個世界。

觀念的進化：以 REGULAR EXPRESSION看見資料的紋理

現在，如果我們把所有可掃瞄的資料，看成是一個 REGEXP的世界，那麼，我們很像是戴上了 Regular Expression的透視鏡，看出存在這個世界的脈絡，不再僅僅是簡單的 像 "hello" 這樣字串的搜尋而已。在這 Regular Expression的世界裡，存在著下列這些字元族群(Character Class)) ：大寫族、小寫族、數字族、白字族、文數族還有「字族」（也就是 **\u \l \d \s [:alnum:] \w**，以 **\u** 為例 **\u** 是字族代號，字族名為[:upper:]，詳見前面各節關於字元族的表列），

必要時也可以用 [ ] 組成臨時的無名字族，以便進行搜尋標定的工作。

善用 ( ) 這對補助符號

(和) 除了用在 (?:) (?!) (?=) 的場合外，( ) 用來標示字串樣板，並依照順序，標上號碼，之後便可以 \1, \2 … \9 等方式來加以叫用，這個功能在高級的搜尋替換中，效益宏大。

( ) 括弧除了可讓字串編入樣板外，因為（）不會影響搜尋樣本的內容，所以為了增加 Regular Expression 符號組的可讀性，也被廣用來將一堆有意義的符號組刮起來，這樣可以讓符號組更具有清晰的意義。

欣賞 REGULAR EXPRESSION

如果用欣賞的角度來看，第一個要認識的特用字當然是 . ，因為他除了換行以外可以標定所有的字元。因此， . 可以把 Regular Expression世界的所有字元，分成「以換行為界的區段」，也就是「可以**標示**出以換行為界的所有段落」。這就是Regular Expression世界的新觀點！

註（ . 也可以表示換行字元的補集合 [^\n] ）

第二個要認識的特用字是 * ，搭配上 . 字元之後， .* 就可以標示一行之中（一行就是以 \n 換行為界的資料）的所有字元了。

.* 是任意字元（換行除外）的「任意」次數的搜尋比對，所以我們也可以把 .* 看成在一行之內的往前「掃瞄」，或者「掃過這一行」，我覺得用這個觀點來思考REGEXP的操作，常常有更傳神更富有意義。

字族的部分，要特別介紹的文字族是 **[:word:]** 族（字族代號 **\w**），這個文字族就是所有文數字（包括中文字，但不包括中文符）再加上底線 _ 這個字元。例如，我們要進行檔名、帳號名稱的標定時，通常都會以 **\w** 為基礎再進行修整。

範例解說

2.4.1. 在一行中任意長度的字串標定

由於"**.**"是一個「換行」之外的所有字元，所以 ."**.*** 就是在一行中任意長字串的重要元素。

例1：標定從行頭到";"的字串

Regular Expression FIND: **^.*;** （紅色加底線的就是搜尋樣本指令）

正則表達式搜尋資料樣本： Regular expression 正則表達式; ABCD （紅色標定了搜尋所得的結果）

說明： **^** 表示從一行的開頭起，

.* 其中"**.**"是「除了換行之外的任意字元」

而 "*"則是前一個字"."可以重複任意次數。

";"則是以 ";"為結束，如果這一行中有好幾個 ";"則以最後一個為準（因為REGEXP的基本假設是「貪婪」的 greedy模式）。

例2：重複拷貝每一個字(word) （注意：這裡的「字」是指被符號、空白分開的中英文字串）

Regular Expression FIND: **\w+** （為了看得更清楚，也可以寫成[\w]+）（也可以寫[\w]*）

Regular Expression REPLACE: \0 \0 （\0 就是整個 REGEXP的樣本字串）

說明： **\w** 就是搜尋屬於「字」(word，英文字母，中文字再加上英文底線，參見1.4字元族) 的字元，直到不是字的字元（例如空白、符號等）。 **\w+**就是符合的字元**一直往前掃瞄**直到不符合 **\w** 的自為止。

\ 0 是換上整個樣本（就是樣本自己）， \0 \0 則是複製自己加上一個空格再複製一次。

例3：標定單行的C語言敘述

Regular Expression FIND: **^.*[\w]+\(.*\).***

正則表達式搜尋資料樣本：**public func0(x,y) { z=x+y; }** （C 語言的典型樣本）

分解說明：

^ 是從行頭開始掃瞄搜尋。

^.* 是從行頭掃瞄任意長度（0或多個字元）

[\w]+ 要求掃瞄過所有文字族的字元，只要是屬文字族的字元都算符合。**+**是要確定至少有一個合法的文字。

[\w]+\(則是要求掃瞄過所有文字族的字元，直到遇上 **(** 為止。這 **(** 是 REGEXP特用字元(metacharacter)，所以要**\(**。

.*\) 以 **.*** 往前搜尋掃瞄，直到 **)**。

.*\).* 以 **.*** 往前搜尋掃瞄，直到 **)**。並「直到這行結束」，因為 **.*** 會搜尋到碰到換行字元才結束。

2.4.2. 典型的檔名搜尋標定

DOS檔名字串是由文數字及減號所組成，REGULAR EXPRESSION的表示為 **[-_a-zA-Z0-9]**或者**[-\w]**

WinXP的名字串則可以多包含 **!@#$%^&**，並且檔型可以超過3個字元。

所以一個DOS檔名的搜尋標定可以：

Regular Expression FIND: **[-\w]+[.][-\w]{1,3}\b**

分解說明：

[-\w]+ 用來標示至少一個字的任意長度的檔名。只要是屬文字族的字元和 **-** 字元都算符合。

[ ] 裡面列舉了 **-** 號和 文字族 **\w** 。**[-\w]**是文字族再加上 **-** 這個字元， **+** 則是「至少要重複一次」。

[-\w]+[.] 其中 **.** 因為是 REGEXP的特用字元(metacharacter)，所以必須用 **\.** 或 **[.]**來回歸普通字元

[-\w]{1,3} DOS的檔型長度不能超過3個字元，由{1,3}標示之，也就是符合 **[-\w]** 的字元可以1～3個字元長度。

[-\w]{1,3}\b **\b**就是指定必須以右邊為界，也就是右邊只能是符號，例如空白，. 等符號等，不可以是文字字元。

如果這是WinXP，則更精細的搜尋命令樣本為：

Regular Expression FIND: **[-!@#$%^&\w]+[.][-!@#$%^&\w]*\b**

正則表達式搜尋資料樣本：（各種檔名樣本）

DOS, WinXP合法檔名例: file.txt file-0_1.dat

WinXP合法的檔名: file.1235 file-@#$%^&.txt01 (-_@#$%^&可以當檔名，檔型可超過3個字)

不合法檔名例：file*.txt file1.t<>

2.4.3. 標定一個範圍

2.4.3.1 標定 " " 所包含的字串 (strings surrounded by double-quotation marks)

Regular Expression FIND: **".*"**
正則表達式搜尋資料樣本 (藍色標定了搜尋所得) : "Regular expression" "正則表達式"
(藍色是標定了搜尋所得的邊界)
說明：
從頭開始掃瞄標定，直到換行前的最後一個 "。因為REGEXP基本上是貪婪模式(greedy)，以掃瞄最多為準。

Regular Expression FIND: **".*?"**
正則表達式搜尋資料樣本： "Aabc" zzzzzzB" (藍色是標定了搜尋所得)
說明： 其中 **?** 是用來指示 REGEXP進行「保守，不貪婪」(non-greedy)模式。所以只掃瞄到最近的 " 就終止了。
" 不是特用字元(metacharacter)，所以都不必用 \ 來回歸普通字元。

2.4.3.2 更多的範圍標定的例子

Regular Expression FIND: **R.***式
正則表達式搜尋資料樣本 (藍色是標定了搜尋所得) : Regular expression正則表達式
Regular expression
說明： 從開始掃瞄標定，直到換行前的最後一個 B。因為REGEXP基本上是貪婪模式(greedy)，以掃瞄最多為准。

Regular Expression FIND: AA.*?BB
正則表達式搜尋資料樣本 (藍色標定了搜尋所得) : AAzzzzbbBBzzzzzzz zzzzzBB
注意：這是分大小寫, case-sensitive的情況，在不分大小寫時可以手到更多
說明： 其中 **?** 是用來指示 REGEXP進行「保守，不貪婪」(non-greedy)模式。所以只掃瞄到最近的 BB 就終止了。
如果是大小寫不分(case-insensitive)則在bb就因成功比對而終止了：
AAaazzbbBBzzzzzzz zzzzzBB (藍色是標定了搜尋所得)

2.4.4 成對的字串範圍標定 [....] 「....」

Regular Expression FIND: 「.*」
正則表達式搜尋資料樣本（藍色標定了搜尋所得）： 「正則表達式」AA 「正歸表達式」BB 正規標示式」
說明： 從開始的「掃瞄標定，直到換行前的最後一個 」。因為REGEXP基本上是貪婪模式(greedy)，以掃瞄最多為準。

Regular Expression FIND: 「.*?」
正則表達式搜尋資料樣本（藍色標定了搜尋所得）： 「正則表達式」 AA「正歸表達式」BB 正規標示式」
說明： 其中 ? 是用來指示 REGEXP進行「保守，不貪婪」(non-greedy)模式。所以只掃瞄到最近的 」 就終止了，所以一共掃瞄標定了兩組。

Regular Expression FIND: \[.*\]
正則表達式搜尋資料樣本（藍色標定了搜尋所得）： [regular expression] Regular Expression [z]zzzzzz] [X]
說明： 從開始掃瞄標定，直到換行前的最後一個]。因為REGEXP基本上是貪婪模式(greedy)，以掃瞄最多為准。[是特用字元(metacharacter)，所以需要 \[和 \]。

Regular Expression FIND: \[.*?\]
正則表達式搜尋資料樣本（藍色標定了搜尋所得）： [regular expression] Regular Expression [z]zzzzzz] [X]
說明： 其中 ? 是用來指示 REGEXP進行「保守，不貪婪」(non-greedy)模式。所以只掃瞄到最近的] 就終止了，所以一共掃瞄標定了三組。

2.4.5 標定跨兩行的 AA....CC 字串

什麼是一行，現在我們用Regular Expression 的觀點來看，就是資料中有「換行」$0D $0A字元系列 ，換行在REGEXP 代號就是 **\n** ，但實際的資料碼，則因系統而異。Windows Emeditor 的 **\n** 是一個 $0D $0A 兩個 ascii code的序列，但在 Linux，MacOS上則會有些微的差距。然而，不管在哪一個系統，REGEXP的 **\n 代碼**標示換行的定義則是不變的。
因此，所謂「兩行」，就是被 **\n** 所隔開的的兩段文字，而在編輯器展現的時候，則會在空間上以換一個新行來展現。

Regular Expression FIND: **AA.*\n.*CC**
正則表達式搜尋資料樣本（藍色標定了搜尋所得）：

AA AA B　　(在這一行B的後面，從資料的角度來看，就是有0x0D 0x0A兩個BYTE，來指示EDITER進行換行的動作)
B CC DD; EE
說明：**AA** 從AA開始標定，**.*** 就是只要是非換行字元就往前掃瞄。**\n** 經過一個換行，**.*CC** 再繼續搜尋直到CC為止。

2.4.6 標定所有出現「 」的資料行

Regular Expression FIND: **^.*(?:「|」).***
正則表達式搜尋資料樣本(藍色標定了搜尋所得)：
AA AA 「資料搜尋」BB CC BB; Regular expression 正則表達式搜尋成功範例
AA AA 「資料　　Regular expression 正則表達式搜尋成功範例
AA AA 搜尋」BB CC BB; BB　Regular expression 正則表達式搜尋成功範例
AA AA 搜尋 BB CC BB; BB　Regular expression 正則表達式搜尋沒有成功。
說明：
^　表示從一行的開頭起，
.*　其中"**.**"是「除了換行之外的任意字元」，而 "***"則是前一個字"**.**"可以重複任意次數。
(?:「|」) 則是在 **(?:|)** 中套入「和」兩個字元，其中 | 是 OR 的功能，
　　也就是 | 兩邊的樣本都會被採用來搜尋(請參考2.4節字串樣本群組)。
.* 最後的 **.*** 就是掃瞄標定「直到這行結束」的意思。因為行尾才會碰到換行字元。

2.4.7 標定IP字串 (IP Addresses)

Regular Expression FIND: **([0-9]{1,3})\.([0-9]{1,3})\.([0-9]{1,3})\.([0-9]{1,3})**
正則表達式搜尋資料樣本(藍色標定了搜尋所得)：192.168.001.192 1.1.1.1 1.1.12345.1 1.1.1.12345
說明：
就是列舉出IP地址標準格式。
[0-9]　就是直接列舉 0-9之間的字元為搜尋標定的字元(也可以直接用 \d 或 [:digit:] 或 [\d] 來標示。
[0-9]{1,3}　就是搜尋標定0-9之間的這個字元，1～3位數都是搜尋比對成功。
([0-9]{1,3})　加了 ( ) 就是要指定 REGEXP 把其中找到的字串，放入樣本變數群組中，在這裡因為是第一個出現的，
　　所以會放入「樣本1」，要呼叫的話，就以 \1 來呼叫。
\.　就是 **.** 直因為 **.** 是特用字元(metacharacter)所以必須寫成 **\.** 才能回歸特用字元。

([0-9]{1,3})\.([0-9]{1,3})\.([0-9]{1,3}) 後面的三組跟第一組是重複的內容，就不再說明，然而其中進入樣本變數會依序是\2 \3 \4 。
如果要更嚴謹地排除最後一組異常 1.1.1.12345的情形 則要再加上 \b ....\b 也就是
\b([0-9]{1,3})\.([0-9]{1,3})\.([0-9]{1,3})\.([0-9]{1,3})\b
註：這裡的 ( ) 並沒有要引用，只是用來做觀念上集合。

2.4.8 標定email字串

Regular Expression FIND: **[;,]*([\.\w]+@[\.\w]+)**
正則表達式搜尋資料樣本（藍色標定了搜尋所得）： aa@gg.com
abk@gmail.com;ajkimo$@gmail.com,cc.kao@ebay.com
說明：
[;,]* 在email字串之前可以是 **, ;** 空白或者是一行的剛開始，要接受這種情況，所以先掃瞄過去
([\.\w]+@[\.\w]**)** **(.....)** 括弧是把真正的email字串刮起來。這裡只是補助閱讀，在替換的時候則會有需要。
[\.\w]+@ 在@之前，可能是一個串字（一般為字族 \w）或是有 **.** 的兩個字串
@[\.\w] 在@之後，接受[\.\w] 這些字元，也就是 **.** 和 **[a-zA-Z_0-9]**（\w還包括了中文字，嚴格來說，中文是不合法的）。
. 是REGEXP的特用字元，所以要以 **\ .** 回歸普通字元。
[\.\w]+ **+** 標示了 [\.\w] 這些字元的重複至少必須有一次。（嚴格來說，email地址的字元數都一定有好幾個字元，這裡都是不完美的指令標示）

更準確的email標定樣板:

Regular Expression FIND: **\b[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}\b**
正則表達式搜尋資料樣本（藍色標定了搜尋所得）： aa@gg.com
abk@gmail.com;ajkimo$@gmail.com,cc.kao@ebay.com
\b ....\b 用來確認整個EMAIL地址字串兩邊都不是文字，保證構成一個整體。
[A-Za-z0-9._%-]+ **[A-Za-z0-9._%-]** 是email 帳號的合法字元的集合，
[A-Za-z0-9._%-]+ 就是至少要掃過一個這類的字元，直到第一個不是所列的字元為止。
因為中文字不合，所以不用 \w 。如果是「至少兩個字元」則要寫成 **[A-Za-z0-9._%-]{2,}**
註：更嚴謹的規格就要看相關文件了，這裡我只能確定，已經更接近完美而已！
[A-Za-z0-9.-]+ 次網域名稱，因為 \w 含有中文字，所以不用 \w
\.[A-Za-z]{2,4} 最右邊的字數有限制（.com .tw .cn .org .... ）

2.4.9 刪除出所有行尾的白字(white space)

Regular Expression FIND: \s*$
Regular Expression Replace with:
正則表達式搜尋資料樣本（紅色標定了搜尋所得的空白）：AA_____CC_DD_____
正則表達式替換結果（紅色標定了替換之後的空白）：AA_____CC_DD
說明：替換成「沒有字串」就是刪除

2.4.10 在行頭加上//

Regular Expression FIND: ^
Regular Expression Replace with: //
說明： 行頭放 // 是許多程式語言的註解

2.4.11 把重複的空白或跳格都壓縮成單一個空白字元

Regular Expression FIND: \s+
Regular Expression Replace with: _（此處以紅色底線代表一個空白字元以協助顯示）
正則表達式搜尋資料樣本（紅色標定了搜尋所得的空白）：AA_____CC_DD____EE
正則表達式替換結果（紅色標定了替換所得的空白）：AA_CC_DD_EE

2.4.12 把 (abc) 替換成 [abc]

Regular Expression FIND: \((.*?)\)
R eplace: \[\1\]
正則表達式搜尋資料樣本（藍色標定了搜尋所得）：(abc) ZZ (12345) ZZ (Regular Expression) ZZZ
正則表達式替換結果（紅色標定了替換異動之處）：**[abc]** ZZ **[12345]** ZZ **[Regular Expression] ZZZ**
說明：
\(\) 是搜尋的邊界。
(.*?) **( )** 括弧起來的字串就會被編入樣本群組 **\1** 這是第一組括弧資料，所以隨後可以用 **\1** 來叫用。
.*? 這 **?** 就是用來指示 REGEXP進行「保守，不貪婪」(non-greedy)模式，沒有 **?** 的結果會如下：
正則表達式搜尋資料樣本，藍色標定了 \((.*)\) 搜尋所得：(abc) ZZ (12345) ZZ (Regular Expression) ZZZ
\[\] 分別替換了 **\(\)**，中間則用 **\1** 來放回原來樣本的字串。

2.4.13 把 <H3 ...> 替換成 <H4 ...>

Regular Expression FIND: **<H3(.*?)>**
Regular Expression Replace: **<H4\1>**
正則表達式搜尋資料樣本（藍色標定了搜尋所得）： <H3 abc> <H312345> (hello) ZZZ
正則表達式替換結果（紅色標定了替換異動之處）： <H**4** abc> <H**4**12345> (hello) ZZZ
說明： 與(abc)情況類似。

2.4.14 把 Regular Expression 替換成 REGEXP

Regular Expression FIND: [Rr]egular [Ee]xpression
Regular Expression Replace: REGEXP
正則表達式搜尋資料樣本（藍色標定了case-sensitive時的搜尋所得）： Regular Expression regular expression REGULAR EXPRESSION
正則表達式替換結果（紅色標定了替換異動之處）： REGEXP REGEXP REGULAR EXPRESSION
說明：
[Rr] 和 **[Ee]** 都是定義了一個臨時字元族來做多容許多重選擇。
單字元時可以使用 **[ ]** 來完成，但如果多種樣本字串不是字元時，就要用 **(?: |)** 了。

2.4.15 把 9/13/2003 替換成 2003.9.13

搜尋替換的例子中，這是一個經典的的例子。
Regular Expression FIND: ([0-9]{1,2})/([0-9]{1,2})/([0-9]{2,4})
Regular Expression REPLACE: **\3\.\1\.\2**
正則表達式搜尋資料樣本（藍色標定了搜尋所得）： 9/13/2003 2003.9.13 02/27/2007
正則表達式替換結果（紅色標定了替換異動之處）： 2003.9.13 2003.9.13 2007.02.27
說明： 原始樣本中有月/日/年三組資料要被移動到新格式，所以要用(月)/(日)/(年)格式，分別建立
\1 內含有月的資料，也就是**([0-9]{1,2})**。日和年的情況相同。
\2 內含有日的資料， **\3** 內含有年的資料。
替換時， **\3\.\1\.\2** 就分別替代成 年.月.日 其中 **.** 是REGEXP的特用字原，所以要寫成 **\.** 回歸普通字元。

========== 以上為Regular Expression 正則表達式的教學文件內容 ==========

==========以下為Regular Expression 正則表達式的教學文件中的附錄 ==========

請將下面的附錄一「 阿江的 REGULAR EXPRESSION修練密笈.TXT 」 和附錄二「EmEditor REGULAR EXPRESSION練功檔.txt」 COPY下來，貼到Emeditor去實際搜尋練習。

這個兩個練習檔都是經過精心安排，並可以在Emeditor上互動式地觀察驗證的，從EmEditor平台的即時互動的反應中，讀者的每一個想法和疑問，都可以當場的實驗及觀察，這是修得真必經的過程。

附錄一

********** 阿江的 REGULAR EXPRESSION修練密笈.TXT **********

你已經先快速閱讀了「阿江的 REGULAR EXPRESSION入門修練手冊」，再透過Emeditor來體驗，交互參照，互動式地觀察螢幕上的搜尋標定結果，你就可以修得REGULAR EXPRESSION的真功夫了！── 阿江AJ 2006.6～2006.8, 2007.2.22

（請拷貝下列練習檔內容到EmEditor去逐一搜尋，並觀察其結果，驗證你所學到的 REGULAR EXPRESSION 知識）

練習一：搜尋各種字元族(Character Class):

正規表達式（正則表達式）搜尋資料樣本：
0123456789
ABCDEFABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz_
REGULAR EXPRESSION FIND: BCD

注意1:FIND的功能中有一個Match Case選項，選擇是否要分大小寫。
注意2:FIND的功能中有一個「從頭搜尋」選項，讓你可以每次從頭掃瞄。
注意3:FIND的功能中有一個「以word為範圍」選項，就是比對時，不會跨越英文的word。

REGULAR EXPRESSION FIND: \u，[[:upper:]] (所有大寫英文字母）
注意：**[[:upper:]]**才能正確叫用大寫字族，**[:upper:]**只 是定義在定義 **:、p、p、e、r、:** 字元的成一個暫時的無名族，

正規表達式搜尋： \U，[^[upper:]]（非大寫）

正規表達式搜尋： \l，[[:lower:]]，（小寫字母， 非小寫 = \L，[^[:lower:]] ）

正規表達式搜尋： \d，[[:digit:]]。（數字 ， 非數字 = \D，[^[:digit:]] ）

正規表達式搜尋： [[:xdigit:]]。（十六進族， 非十六進族 = [^[:xdigit:]] ；沒有特用字族代號）

正規表達式搜尋： [[:alnum:]]，（ 文數字， 非文數字 = [^[:alnum:]]；沒有特用字族代號）

正則表達式搜尋： [[:word:]]。（詞字族也簡稱「字族」，非詞字族 = [^[:word:]] ；沒有特用字族代號）

正規表達式搜尋： \s，[[:space:]]。（白字，非白字 = \S，[^[:space:]]。 ）
注意：要表示字元族(Negated Character Class)，一定要在[ ] 之內，例如[^\s] ，不可以^\s

正規表達式搜尋： \n，（換行，非換行 = [^\n] ，換行請參考1.4 特用字元(metacharacter)的表示這一節，各系統會有一些差異）

練習二：搜尋樣本的重複與擴展

正規表達式搜尋，藍色標定了搜尋所得： .　　---樣本： 123 YES Hello (換行之外的所有字碼全都算是)
正規表達式搜尋，藍色標定了搜尋所得：A+B　---樣本： B ABB AABB ZZ AAAAABB (1或多個A, 以greedy模式盡量擴張)
正規表達式搜尋，藍色標定了搜尋所得：A*B　---樣本： B AB ABB AABB ZZ AAAAABB (0個A合法,所以單獨B字元也符合！)
正規表達式搜尋，藍色標定了搜尋所得：A.*B　---樣本： B ABB AABB ZZ AAAAABB (A之後以greedy模式盡量擴張到最後的B)
正規表達式搜尋，藍色標定了搜尋所得：A.*?B　---樣本： B ABB AABB ZZ AAAAABB (0或多個A, 以non-greedy保守模式盡量不擴張)
正規表達式搜尋，藍色標定了搜尋所得：A.{1,3}B ---樣本： B A123BB AAaBB ZZ AAAAABB (A,B之間可以1～3個任意字元)
正規表達式搜尋，藍色標定了搜尋所得：A.{1,2}B ---樣本： B A123BB AAaBB ZZ AAAAABB (A,B之間可以1～2個任意字元)
正規表達式搜尋，藍色標定了搜尋所得：a/{2,3}b ---樣本：a////b a///b a//b a/b ab （中間必須剛好有2～3個/）
正規表達式搜尋，藍色標定了搜尋所得：a/{,3}b　---樣本： a////b a///b a//b a/b ab （中間必須剛好有0～3個/）
正規表達式搜尋，藍色標定了搜尋所得：a/{3}b　---樣本： a////b a///b a//b a/b ab （中間必須剛好有3個/）
正規表達式搜尋，藍色標定了搜尋所得：colo**u?**r ---樣本： color colo**u**r （? 用來表示u可有可無)
說明：透過這些練習和觀察，可以「看見」每一個搜尋標定的結果。

練習三：搜尋樣本有效位置的指定

正規表達式搜尋，藍色標定了搜尋所得：\b133　---樣本： **133**191 332081133 **133** (搜字在頭的”13”)
正規表達式搜尋，藍色標定了搜尋所得：\b133\b　--樣本： 133191 332081133 **133** (搜一個獨立的”133”字)
正規表達式搜尋，藍色標定了搜尋所得：133\b　--樣本： 133191 332081**133** **133** (搜一個獨立的”133”字)
正規表達式搜尋，藍色標定了搜尋所得：\B3　---樣本： 133191 **3**32081133 133 （不在字頭的3）
正規表達式搜尋，藍色標定了搜尋所得：\B3\B　---樣本： 1**33**191 3**3**2081**3**3 1**3**3 (不在字頭和字尾的3)

練習四：搜尋樣本的列舉與排除

註：下面 (?:pattern) (?=pattern) (?!pattern) 都是不存入樣本群組變數的(?...)

正規表達式搜尋，藍色標定了搜尋所得：industr(**?:**y|ies)　---樣本：industry industries f.txt (兩種樣本都要搜出來)
正規表達式搜尋，藍色標定了搜尋所得：Win(**?=**98|95|NT) --樣本：Win98 Win95 WinNT WinXP
說明：?= 指定要搜出後面含有98,96,NT三種的Win，但不包含98,95,NT

正規表達式搜尋，藍色標定了搜尋所得：[-\w]+[.](**?=**exe|bat)　　---樣本：file.txt file.exe file.bat
正規表達式搜尋，藍色標定了搜尋所得：[-\w]+[.](**?=**exe|bat)[-\w]{1,3} ---樣本：file.txt file.exe file.bat
注意：(?=)只是供判斷的資料，並不在搜尋樣本之內，因此要[-\w]{1,3} 再標示一次（綠色指令用綠色標示結果）。

正規表達式搜尋，藍色標定了搜尋所得：[-\w]+[.](exe|bat)　---樣本：file.txt file.exe file.bat
正規表達式搜尋，藍色標定了搜尋所得：file[.](**?!**exe|bat)　---樣本：file.exe file.dat file.bat file.txt （後面是exe和bat的就排除）
正規表達式搜尋，藍色標定了搜尋所得：[-\w]+[.](**?!**exe|bat)[-\w]{1,3} ---例子：file.txt file.exe file.bat
注意：(?!)只是供判斷的資料，並不在搜尋樣本之內，因此要[-\w]{1,3} 再標示一次。

練習五：搜尋標定──在兩個邊界之間的樣本

正則表達式搜尋：**\bonly .{1,5} but\b**
正規表達式搜尋，藍色標定了搜尋所得：
not only 中1234567 but also ABCDE （距離超過5 個字，所以不符）
not **only 中1234 but** also ABCDE （距離在5個字以內，符合）
說明：**.** 是任意字元都接受，我們也可以加上各種限定條件，例如
正則表達式搜尋：\bonly \w{1,5} but\b （限文字，英文、中文及底線）
正則表達式搜尋：\bonly \d{1,5} but\b （限數字）
正則表達式搜尋：\bonly \u{1,5} but\b （限大寫英文字）

練習六：搜尋─替換練習

把樣本所列的水果名的字串都轉換成一行一句話，形式如下：
(a水果) is delicious!
(b水果) is delicious!
.....

正規表達式搜尋，藍色標定了搜尋所得： **apple banana orange cheery**
REGULAR EXPRESSION FIND: \w+
Regular Expression REPLACE: \n\0 is delicious!
說明：
\w+ 搜尋所有的字， 這樣就可以將 apple banana orange cheery 逐字取出。
\n 是替換上一個「換行」（Windows 版就是放入 $0D $0A 這樣的碼）
\0 就是把搜得的水果名（如apple)放上去）然後再加上 is delicious!
正則表達式替換結果（紅色標定了替換結果）：如下：
apple is delicious!
banana is delicious!
orange is delicious!
cheery is delicious!

********** 阿江的 REGULAR EXPRESSION修練密笈.TXT 結束 *********

附錄二

Regular Expression 正則表達式案例再研究

本研究案例是以「目標導向」為題目進行 REGULAR EXPRESSION搜尋式字的探討，並說明每一個指令的意義及相關問題。透過這些進階案例，你將會發現善用 REGULAR EXPRESSION可以做許多想都沒想到的事。

1. 刪除行內的空白

1.1 刪除每行內的白字（整個搜尋空間）

Regular Expression FIND: [\t]+
Regular Expression Replace:（替換的內容為「沒有內容」,也可以表示成 \s+ ）

說明：**[\t]** 在[...] 中標示了對於 空白 和 Tab 兩種字元，”+”則說明前面的”[\t]”樣本包括了「一次或多次的重複」。這樣的樣本搜到了之後，全部都刪除（也就是替換成沒有）。 對於搜尋字元族的標示，可以用[...]來列舉(如[ako])或標示範圍[A-Z]，也可以選用系統內建的 字族帶碼 \s,\u 等方式。

讀者注意1：

> WhiteSpace的定義是： Space, Tab, FF, LF, CR,（請看1.1 說明），然而Editor REGULAR EXPRESSION 對 \s 實做時確有所取捨，本文所用的EmEditor的 \s 就不包括 CR LF。（在EmEditor中，”[\s]+”也是可以的。若要除去CR LF 則要 \n 來指定）

讀者注意2：各系統對於文數字資料的換行碼各不相同：
WINDOWS使用：CR LF（0x0D+0x0A）
LINUX 使用：CR（0x0D）
MAC OS 使用：LF（0x0A）

1.2 刪除每行內行頭的白字

Regular Expression FIND: ^[\t]+
Regular Expression Replace:（替換的內容為「沒有內容，就是刪除的意思）
說明：**^** 是用來指定「樣本從一行的開始搜尋」也就是在行頭才有效。

2. 搜索 HTML 標記

2.0 定位 HTML 標記的例子

對於 <TR> a sample line </TR>
Regular Expression FIND: <.+>
正則表達式搜尋的樣本，藍色標定了搜尋所得： <TR> a samle line </TR>
註：以「最貪婪的搜尋」所看到的是一行中的最大範圍，如果要保守搜尋(non-greedy，就是在可大可小時，以最小範圍為準)要加上 ? 號 <.+?> 後面幾行有例子。
因為，<+.> 搜尋引擎會因為 ".+" 而不斷往前搜尋，於是跳過了第一個”>”
正則表達式搜尋標定：<.+?> -- 就可以搜到比較準確的HTML tag <TR> a samle line </TR>
“?” 號可以告訴 REGEXP搜尋引擎對”.+”做最保守的非重複擴展搜尋，也就是，先找一次，就去看是否後面的”>”已經出現。（這樣的用法在<.*?也一樣）

<[^>]+> 則是先找到 **<** 然後掃瞄過所有不是 **>** 的字元直到遇到 **>** 就停下來。

2.1 搜尋標定成對的HTML 標記段落（<TR>..... ... ..</TR>）

Regular Expression FIND: <TR[^>]*>(.*?)</TR>
說明：
[^>]*> --- **^** 是取 **> 的「補集」**，也就是要「掃過」「所有非 **>** 字元」（也是指遇到時就 **>** 停止）。
[^>]*> --- ***** 標示前面的[^>] 可以0個或多個字元(涵蓋Open tag 所帶的參數)，其中
[^>]*> --- 最後的 **>** 就是這個open tag的左 > 號為止。
(.*?)</TR> --- 其中**(.*?)**用來掃過任意字元，直到遇到</TR>為止。其意義就是「要涵蓋到</TR>為止的所有字元」。

2.2 所有標記段落

Regular Expression FIND: <([A-Z][-A-Z0-9]*)[^>]*>(.*?)</\1>
<([A-Z][A-Z0-9]*)[^>]*>(.*?)</\1> --- [A-Z]tag的第一字母一定是英文字母開頭，第二個字母以後就可以是文數字。
<([A-Z][-A-Z0-9]*)[^>]*>(.*?)</\1> ---[-A-Z0-9]* 第二個字起就可以是文數字。(更嚴謹的規範請參考相關規格）
<([A-Z][-A-Z0-9]*)[^>]*>(.*?)</\1> ---後面\1就是沿用前面([A-Z][-A-Z0-9]*)樣本。

3. 鎖定含或不含特定字串的一整行

Regular Expression FIND: ^.*John.* -- 鎖定含有John的一整行。

Regular Expression FIND: ^.*\b(one|two|three)\b.*
正則表達式搜尋的樣本，藍色標定了搜尋所得：in **one** day there are **two** men came ...
正則表達式搜尋的樣本，藍色標定了搜尋所得：No one can answer this question

說明：從行頭鎖定含有 one 或 two 或 three 的一整行，只要一個字符合就行。

找出含有同時含有 台北 和 展 兩個字串的所有行
說明：

(?=.*?台北)(?=.*?展).*　兩組 (?=　) 的意思是說「必須同時含有 台北 和 展 」這個搜尋式才會運作。如果只有一個符合也不算。

4. 深入探索IP address

IP 地址大部分的人都有一點概念，經由探索IP地址的各種表示法，及其相關差別，對於 REGULAR EXPRESSION的進一步深入很有用。

Regular Expression **表示法一**： **\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b**
正規表達式搜尋，藍色標定了搜尋所得：**1.1.0.0 255.255.255.255**
2.2.4444.2 192.168.1.10 256.999.999.999
這個式子的缺點是範圍只大略正確，因為把 999.999.999.999 都涵蓋了。
其中：
\b … \b 說明這個式子是在一個完整的字內（前後沒有空白）
\d{1,3} 是指1~3位數的數字都可以。
\. 因為 "."是特用字，所以所有的 "."都要加上"****" 使之回歸"**.**"字母。

Regular Expression **表示法二**：
\b(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\b
正規表達式搜尋，藍色標定了搜尋所得：**1.1.0.0 255.255.255.255**
2.2.4444.2 **192.168.1.10** 256.999.999.999
這是準確而完整的 IP 地址描述。為了表達 0～255，這裡分成(0～199 | 200~249 | 250~255)來表示。
這四段相同的0~255的嚴謹描述，說明如下：

(**25[0-5]**|2[0-4][0-9]|[01]?[0-9][0-9]?)\.　是250-255都對。在二百五十幾時，個位只能只有[0-5]。
(25[0-5]|**2[0-4][0-9]**|[01]?[0-9][0-9]?)\.　200～249 都對；在兩百多時，十位數只能0-4.
(25[0-5]|2[0-4][0-9]|**[01]?**[0-9][0-9]?)\.　**[01]?** 就是說前面這個0或1字元或沒有任何字元都對。
(25[0-5]|2[0-4][0-9]|**[01]?[0-9][0-9]?**)\.　0 ~ 199 都對。
(25[0-5]**|**2[0-4][0-9]**|**[01]?[0-9][0-9]?**)**\.　**(| | |)** 就是指IP的第一段 250~255 或 200~249 或 0~199
(25[0-5]**|**2[0-4][0-9]**|**[01]?[0-9][0-9]?)**\.**　因為 **.** 是的特用字元，所以要 **\.** 回普通字元。
Regular Expression **表示法三**：**\b(?:\d{1,3}\.){3}\d{1,3}\b**

正規表達式搜尋，藍色標定了搜尋所得： **1.1.0.0 255.255.255.255**
2.2.4444.2 **192.168.1.10 256.999.999.999**
-- **(?:\d{1,3}\.)** 這裡的提供了一個**(?:)**形成一組樣本(pattern)以便於後面要標示重複三次{3}，也就是**(?:\d{1,3}\.){3}**。也可以表示成**(\d{1,3}\.){3}**，差別就是這個(...)會依序錄入樣本群組中，供後面叫用以 \1 \2 ... \9 方式叫用，而**(?:\d{1,3}\.){3}**則不會收錄到樣本群組中。
-- {3}標示了前三組都要三位數，最後一組{1,3}則標示1~3位數字皆可。

Regular Expression **表示法四：**
\b**(?:(?:**25[0-5]**|**2[0-4][0-9]**|**[01]?[0-9][0-9]?**)**\.**){3}(?:**25[0-5]**|**2[0-4][0-9]**|**[01]?[0-9][0-9]?**)**\b
正規表達式搜尋，藍色標定了搜尋所得： **1.1.0.0 255.255.255.255**
2.2.4444.2 **192.168.1.10** 256.999.999.999
-- （老實說，我都覺得有一點難解，不過我已經加以標示了，請讀者慢慢解讀吧！）

5. 刪除整行內容重複的連續行

Regular Expression FIND: **^(.*)(\n\1)+$** -- 當連續多行重複時。執行一次可以刪除一行。
Regular Expression FIND: **\1**
^ 從行的起頭，**(.*)** 中的 **.*** 就是重複（掃瞄）到行尾.
\n... -- 用來找新行。
\1 -- 查換新行之後是否與先前**(.*)**所錄入的內容相同。
(\n\1)**+** **+**要讓前面的**(...)** 搜尋動作1次以上。
(\n\1)+**$** **$**用來確認到了行尾。

6. 刪除一行內由，分開的連續重複字串

REGULAR EXPRESSION FIND: **([^,]*)(,\1)+(?=,|$)** – 刪除連續字，刪到只剩一個
REGULAR EXPRESSION REPLACE：**\1**
正規表達式搜尋，藍色標定了搜尋所得：
AA,BB,BB,CC,CC,CC,4444,4444,4444,4444
正規表達式替換所得： AA,BB,CC,4444
([^,]*) -- [^,] 是 **,** 的字元族(Negated Character Class)，也就是以 **,** 之為區隔的字。
(,\1)+ -- **\1** 指的是 **([^,]*)** 樣本字串，因為是最搜尋(greddy)所以有多個重複時會擴張到最後。
(?=,|$) -- **?=** 要確認每一次的樣本後面是 **,** 或者是 **$** 也就是逗點或行尾。

7. 關於中、日、韓文的搜尋標示

（大量的文字處理，尤其是表意文字，其空間很雜亂，所以必須以碼表的方式彙整來進行標示）
這些字碼的空間如下： 待補