

J 語言初步

東華大學經濟學系副教授郭平欣

認識 J	1
學習 J	2
安裝與執行	3
第一次接觸	3
算數運算	3
邏輯比較	4
數字格式	5
文法	6
句子的構成	6
詞彙	7
名詞的維數	8
框架與格子	8
名詞的成員	9
動詞的雙價型	10
副詞與雙價動詞	11
動詞串列	11
動詞選項	12
動詞的秩	13
詞彙管理	14
文稿檔	16
系統	17
外務連接詞	17
時間	18
空間	18
檔案	19
資料檔的存取	19
文稿檔動詞	20
檔案管理	20
程式	21
隱式與顯式	21
邏輯結構控制	22
圖形使用者介面	23
公用程式庫	28
繪圖	31
應用 J	33
資料表格	33

資料檔案.....	35
敘述統計.....	35
排序.....	37
次數分配.....	37
枝葉圖.....	38
迴歸分析.....	40
三角函數.....	43
計算 e 值.....	43
多項式.....	44
平均數.....	46
黃金率.....	46
結語.....	47

認識 J

本文的目的在介紹電腦程式語言 J 的一些基本功能。J 是由 Jsoftware 公司生產的一種陣列語言，為 APL 的繼承者，與 Java、JavaScript 或 JScript 沒有關係。它的命名早過 Sun 的 Java，只是由於普遍性不如，名字被鳩佔鵲巢了。

若要處理一個以上的數字，不論是時間數列或者田野調查的資料，集合（陣列）是比較有效率的處理單位，而 J 就是被設計來處理陣列資料工作的。因此特別適合計量經濟、財務金融與保險業或任何其他需要統計與分析大量資料的使用者；在教授前述課程，或者線性代數及數值分析等類似課程，也是給學生練習的很好工具。

J 尤其擅長處理陣列資料的數學、統計與邏輯分析，讓許多複雜的工作，只要一個或數個字元的指令就可完成。

例如，定義一個 2x2 的矩陣 A，指令為

```
A=.2 2$1 2 3 4
```

其中=.代表指定，\$的前面為矩陣的形狀，後面接著為要輸入矩陣的資料，將會依序置入 A 的各列。本例的矩陣維度只是個例子，用類似的方法可以輕易地輸入任何維度與形狀的陣列資料。

計算反矩陣是線性代數一個重要的工作，用 J 更是易如反掌，指令只有%。兩個字元，故在前例 A 為方陣的情況下，

```
B=. %A
```

得到 A 的反矩陣，並將之置於 B。

至於矩陣的乘法，其指令為+/.*，故

```
A+/. *B
```

會得到單位矩陣。另一個長得很類似的指令-/.*被用來計算行列式，故

```
-/. *A
```

會得到 A 的行列式。

如果要計算 A 各列的平均數，用

```
%#A
```

即可。

以上幾個簡單的例子，可以看出 J 語言的強大功能。J 提供了許多種類似的指令與函數來處理文字或者數字陣列的相關工作，對於為作研究需要寫程式者，將會很高興地發現用 J 寫程式是如此的精簡而迅速，複雜的矩陣操作工作就好像用計算器做算數運算一般。話說回來，由於 J 支援 Palm OS 及 WinCE，若將之灌入掌上電腦，那還真的是一台超級計算器呢！

J 並不是憑空生出來的新的程式語言，它是由 APL 衍生出來的一般目的高階程式語言。對程式語言發展熟悉者，可能會聽過由 Ken Iverson 發展出來的 APL 語言，由一堆稱為 Iverson 符號的奇怪符號組成，例如行政院主計處過去就曾在 IBM 主架電腦上用 APL 來做台灣的經濟預測。APL 最強的功能是那一堆處理陣列的符號。許多工作若以其他語言來寫，需要許多行的指令，APL 只要少數幾個符號就解決了。

世間事往往利弊相隨，APL 的特異功能多，但是相對就不易學習。不過對中國人來說，中國字就是一堆符號，幾千幾萬個字都在學了，不差這幾百個。但是 APL 所用的符號，大半在一般鍵盤上找不到，這就有點麻煩了。在過去，IBM 主架電腦必須用很貴的終端機，這就大大地不利於 APL 的普及。到了 PC

年代，總算有一些公司將 APL 轉過來，但是用特殊符號不利於跨平台作業。且對老外來說，怪怪的符號總是學習的障礙，不利於推廣，所以一直有人想要以標準的 ASCII 來取代。

解鈴還需繫鈴人，Ken Iverson 又與 Roger Hui 共同研發設計 J 語言來取代 APL，但是這回只用標準的 ASCII 字元，將 APL 功能強大的特殊字元，完全以標準文數字拼字的方式來取代。

在 APL 數十年經驗的基礎上，J 自 1990 起又經過了十數年的演化，雖然目前仍在發展中，整套系統基本上已經相當穩定。它的 ASCII 符號表示方法設計得非常巧妙，使得即使是使用簡單的字母，也能夠建構出一套完整而有一致性的語言體系。作為一個完整的程式語言，J 也提供整合的程式發展環境，具有編寫應用程式的圖形介面與設計程式視窗的工具，以及與其他程式語言或應用程式溝通的介面，而且已經建有許多標準的程式庫、公用程式與套裝程式組，對學習 J 語言與開發研發應用程式提供很大的助益。

由於使用標準 ASCII，J 支援許多種工作平台。例如微軟視窗家族的 Windows 9x/2k、WinCE 2.1、Windows 3.1 等，Apple Macintosh family、Palm PC、Handheld PC、Linux、RS6000/AIX、SunOS、NetBSD 以及 FreeBSD 等。

J 語言由 jsoftware 公司在銷售，過去一套七、八百美元跑不掉。對一個還不確定能用來做啥的新學習者來說，這個價錢實在不便宜。不過在千禧年的暑假，大概爲了要推廣，該公司居然大方送，雖然少了一些編寫商業程式用的除錯與保密功能，但仍然是一套功能強大的跨平台語言。作為需要處理大量研究數據，或者作數值分析的使用者，絕對值得花一點時間學習，必然是終生受用。有興趣想進一步瞭解或者使用者，可以到該公司之網頁看看，其網址爲 www.jsoftware.com。

學習 J

想要完全精通 J 並不是那麼容易，它不像一般程式語言，只要懂語法就可以了。學習 J，若沒有陣列的概念，你永遠不會懂反矩陣指令到底在作啥；不懂統計，就不可能欣賞 J 求迴歸係數的精簡迅速。因此當你在學 J 的同時，你也必須學習其相關的知識。如果你已經有相關知識的背景，學起來才會是事半功倍。

雖然完全精通不易，但是 J 語言的功能繁多，一些不需要特殊知識背景的功能，例如一般數學指令、字串處理功能以及編程指令等，要上手並不難。只是有許多複雜的符號以及語法，卻也可能讓想要登堂者望之卻步。本文的目的是讓新手能夠很快的上手，略覓堂奧。一旦入門，學習過程的喜悅自然會讓想要深入研究者上癮，欲罷不能。

學習一個新語言，最佳的途徑就是看範例。看看別人怎麼作，依樣畫葫蘆即可，故本文用一些範例用來展示 J 語言的基本指令。所選的範例只是 J 所有功能的一小部分，主要是數學或統計方面的問題，因為比較容易瞭解。各範例顯示互動式 J 的輸入與輸出，並附上數行的說明。

除了看範例，依樣畫葫蘆也是很重，所以讀者不妨進入 J 系統，一邊看，一邊作，作了以後，看到的知識才會變成自己的知識。

安裝與執行

先到 www.jsoftware.com 取得適合作業平台的版本，依照其說明將 J 系統安裝完成。以下的學習將以 PC 作業平台視窗(Windows 9x/NT)版本為準。安裝好以後，程式集裡就多了一個 J 的檔案夾，按 J 的圖案即可進入 J 預設的第一個工作區 l.jx。

工作區為 J 的執行視窗，是互動作業的區域，任何指令輸入後立即執行，完成後將結果立即送回該視窗。工作區的副檔名為 ijk，使用者可以由 J 系統的選單 File->New IJK 開啓新的工作區，且同時可以開啓多個工作區。

使用者的輸入列與 J 回應的輸出列都在相同的工作區顯示，差異在於輸入列內縮四個空白，而 J 的回應只有一個空白縮頭。安裝完成以後，測試是否安裝成功的最簡單測試為，在工作區鍵入大寫之 CR，按 enter 鍵輸入後若回應空行，如下所示，表示安裝都沒問題，可以開始使用 J 系統了。

```
CR
```

陰影的段落為 J 系統的指令與回應。

第一次接觸

算數運算

在進一步介紹之前，先來看一些算數運算的範例。在工作區直接敲進下列句子，按下 enter 後立即得到結果的回應。

```
1^2+3-4*5%6  
1
```

算數運算元加減乘除分別以+*%代表，而^為乘冪。讀者一定會覺得奇怪，前式如此複雜的混合運算，結果怎麼會是 1 呢？

傳統數學規定，算數運算的優先順序(precedence)為乘冪最優先，再過來先乘除後加減。而 J 語言的規則更簡單，一切指令都是由右向左執行。這是因為 J 的運算元非常之多，非但不易訂出優先順序，即使勉強為之，也沒幾個使用者能夠搞得清楚，因此乾脆以簡單法則規定，一律由右向左。

知道 J 的指令由右向左執行以後，上例的結果就很明白了，乘冪最後做，不論 2+右方的結果為何，1 的任何乘冪永遠為 1。

雖然這樣的規定可以將指令的優先性化繁為簡，可是算數運算不合一般的習慣，是必須付出的代價。新的習慣不難養成，真受不了，也可以用小刮號的方式來解決優先順序的問題。下式即會先算刮號內的乘冪，再加上右方運算之結果。

```
(1^2)+3-4*5%6
0.666667
```

本例的結果為一個無限循環小數，但顯示的結果只有六位數的精確度，這只是預設的列印精確度，J 在電腦內儲存的精確度遠高於此。要更改列印精確度的方式如下：

```
pps =. 9!:11
pps 9
(1^2)+3-4*5%6
0.666666667
```

其中=. 為指定指令，用來指定 9!:11 給 pps，這相當於將 pps 指定為列印精確度的指令，而用 pps 9 來指定列印精確度為 9。列印精確度的上限為 20 位數。

若運算的引數不只有一個數字，合法的運算必須兩邊維數相同，或一邊為純量。若兩邊維數相同，運算元作用於每一個對應的元素；若一邊為純量，則作用於另一邊的每一個元素。例如

```
1 2 3+4 5 6      NB. 向量相加
5 7 9
1+4 5 6          NB. 純量加上向量
5 6 7
```

邏輯比較

如同許多其他程式語言，J 將真值訂為 1，而偽值訂為 0。邏輯運算元的「且」表作*，「或」表作+，而「非」表作-。或與且的真值表可作如下

```
d (+./ ; *./) d=: 0 1
+---+---+
|0 1|0 0|
|1 1|0 1|
+---+---+
```

由右方開始，d=:0 1 定義 d 為二個元素的向量，而 d (...) d 表示將左方的

d 與右方的 d 做運算。運算元在括號內，以分號;區分不同的運算元，本範例有兩個運算元，+./與*./，其中/代表插入，故將或(+.)及且(*.)插入左方 d 與右方 d 的所有元素間，得到邏輯運算表。相同的元素分別做兩種運算，結果分別置於不同的格子內，如前面的結果所顯示。

用來比較大小的二元關係有小於<、小於或等於<=:、等於=、大於或等於>:、大於>等。

函數=| 用來測試數字是否等於其絕對值，故為非負檢定；函數=<.用來測試數字是否等於其底整數，故為整數測試。而(=|)+.(=<.)測試數字為非負或整數：

```
(test=: (=|) +. (= < .)) _2 _2.4 3 3.5
1 0 1 1
```

由於電腦內數字儲藏方式為二位元，因此精確度有其限制，數字關係的比較無法做到完全正確，因此存在容許誤差(tolerance)，為比較數字中最大數字的 2^{-44} 。

數字格式

除了一般的實數外，J 語言也定義了許多其他數字表達形式。複數以 j 連結實數與虛數而成，如 3j4 代表 $3+4i$ ；分數由 r 連結兩個整數，如 3r4 代表 $3/4$ ；負號以_來與動詞減去符號-作區別，動作-3 的結果為名詞_3。

```
 %:_9          NB. 開根號
0j3
 2r3*3r4      NB. 分數相乘
1r2
```

式中 NB. 的後面為註釋。前例%:為開根號指令，根號負九得到虛數三；而由分數相乘的例子可以看出，計算的結果會自動化簡。

其他類似的數字表達式還有科學計數，

```
 2e3          NB. 2*10^3
2000
```

圓周率，

```
 2p1          NB. 2*Pi^1
6.28319
```

以及非十進位轉換成十進位，

```
 16b1a
26
```

其中 16b 指定數字 1a 為十六進位，輸入後立即轉換成十進位的結果 $1*16^1+10*16^0$ 。

文法

J 語言的術語採用英文法術語，如字母(alphabet)、單字(word)、句子(sentence)、動詞(verb)、名詞(noun)、副詞(adverb)與連接詞(conjunction)等。主要理由是因為數學用語與程式用語有許多類似而不盡然相等的意涵，不如自然語有較大的解釋空間。雖然 J 語言的開發者建議使用英文術語，但並不強求，使用者仍然可以採用自己熟悉的術語，並不會影響對 J 語言的瞭解與學習。

句子的構成

J 的句子由單字組成，而單字又由一個或以上的字母組成。句子由一群單字組成的完整指令，一個句子一列，結尾不需要特別符號。以下列句子為例，

```
3.4+4.5
7.9
```

式中有三個單字，構成一個簡單加法的完整指令。在比較複雜的句子，到底有幾個單字就不是那麼清楚，可以用指令 `::` 來剖析句子

```
:: '3.4+4.5'
+---+---+
|3.4|+|4.5|
+---+---+
```

其中 `+|` 等字元稱為框架字元，其所圍成的方格稱為框格，每一框格內為一單字。

單字由數個字母構成，能表達某種意義。前例算式中有三個單字，其中兩個表達數字，一個表達加法動作。

表達數字以外的單字，依系統指定或者使用者指定，分成基本單字(primitive)與指定單字(name)：

基本單字是 J 語言定義的單字，意義固定，使用者不能更改。基本單字大致分兩種，由一個或以上之圖形字母(英文字母以外之字母)，或任何字母(包括非圖形字母)加上尾飾點(英文句點或冒號)拼成。例如 `+號`，`+`、`+`、`+`、`+` 分別代表不同的基本單字；而 `i.` 也為基本單字。

指定單字由使用者定義的單字，一般是以指定動詞 `=.` 或 `=:` 來定義。例如

```
x=. 3.4
y=.4.5
x + y
```

7.9

指定單字 x 、 y 的內容為數字，故可以用來運算。一般而言，指定單字的內容可說是五花八門，可以為資料陣列，包括文字或數字，也可以為其他單字，包括基本單字。

動詞 $=$ 與 $:=$ 的差異在於， $=$ 為局部指定動詞，而 $:=$ 為全域指定動詞。以全域動詞指定的單字為全域單字，以局部動詞定義的單字稱之為局部單字。在工作區使用時兩者功能完全相同，但是在寫程式的時候，兩者就有差別了。

J 語言所使用的 ASCII 字母，包括英文字母 26 個小寫字母(a to z)與 26 個大寫字母(A to Z)，大小寫有分別。其他字母包括數字 0 1 2 3 4 5 6 7 8 9 與 $= < > _ + * - \% \wedge \$ \sim | . : , ; \# ! \backslash [] \{ \} " ` @ \& ? () '$ 等。

雖然這些都是常用的符號，但是為了唸得溜口，除了英文字母與數字外，我們對其他符號取一個一到二個音節的簡單中文名：

```
+ 加 - 減 * 乘 % 除 ^ 冪 $ 錢 ~ 浪
# 井 ! 階 ? 問 | 豎 _ 負 & 安 @ 在
, 逗 . 點 : 冒 ; 分 < 小 = 等 > 大
/ 上 \ 下 " 闊 ` 捺 ' 撇
左右( )刮 []框 {}弓
```

以後要唸出程式內容，會比較容易。

詞彙

單字以其在句子中所發揮的功能，區分為名詞、動詞、副詞與連接詞等。名詞單字為文數字資料。文字的表達方式將字串以單引號刮起來，如 'text'。

動詞指具有動作意義的單字，一般作用於名詞。前例中的兩個數字為名詞， $+$ 為動詞，將其他兩個單字相加。

使用者可以用指定動詞 $:=$ 來指定一筆資料給一個變數，稱之為替該資料命名。例如， $AS:= 'ieas'$ ，將字串 'ieas' 命名為 AS，由於 AS 代理一字串名詞，故又稱之為代名詞。同樣地，若給動詞一個名字，稱之為代動詞(proverb)。例如指定

```
plus =. +
```

名詞 plus 為代動詞，用來代替動詞 $+$ 。一旦定義後，我們就可以在算式中使用

3 plus 4

7

在英文句子的結構，名詞往往有主詞、受詞之分，如主詞+動詞+受詞。但在前面加法的例子，將 3.4 與 4.5 分成主詞或受詞沒啥麼意義，加法同時作用於該二名詞，故可通通稱作是此動詞的受詞，或稱該二名詞為此動詞的引數(arguments)。

連接詞與副詞用來修飾與結合兩個動詞或代動詞，來定義新的動詞或代動詞。例如副詞 /，將配合之動詞插入所有的成員間，故

+/3 4 5

12

*/3 4 5

60

例中 +/ 為加總，而 */ 為連乘。

名詞的維數

內容只有單一物件的名詞稱之為元素(atom)，元素的維數(dimension)為 0。如果內容為數字，則元素同於數學的純量，但 J 的元素也可以是一段文字串。

相對於數學的向量，內容含一系列元素的名詞稱之為條列(list)，其維數為 1，而長度指其元素個數。

二維之元素陣列稱之為表單(table)。表單的內容由雙邊動詞\$(塑形)來指定，而表單的形狀由\$的單邊動詞顯示。指數 i 也可定義表單，例如 i. 2 3 將數字 0 到 5 依序排入 2x3 的表單中。

J 也可以定義更高維數的陣列，例如 2x3x4 的陣列。陣列的維數稱之為軸(axis)，元素沒有軸，條列有一條軸，表單有兩條軸，而軸的個數稱之為秩(rank)。至於陣列的形狀(shape)指的是陣列在每一軸上的元素個數。

框架與格子

陣列可以框架(frame)與格子(cell)的方式來表達。一個 2x3x4 的陣列可以視作一個 2x3x4 的框架，其中每一個格子內有一個元素；也可視為 2x3 的框架，每一個格子含一個長度為 4 的條列；或者視為 2 格的框架，每一個格子內為一個 3x4 的表單；或者為單一格子，含 2x3x4 的陣列。因此框架的形狀與其格子內物件的形狀互相替代，框架的維數與格子內物件的維數合起來為原來陣列的維數。

k 維格子(k-cell)是維數為 k 的格子。因此 k 維陣列可以是含 0 維格子之 k 維框架，或者是含 1 維格子之 k-1 維框架，以此類推。

名詞的成員

當以+/來加總一個名詞，如果此名詞為一個條列名詞，則將其所有元素加總，但如果此一名詞為一個表單，則加總的動作並不是針對其所有的元素，而是將所有的條列加總成一個新的加總條列。

```
]x=.3 4$i.12
0 1 2 3
4 5 6 7
8 9 10 11
+/x
12 15 18 21
```

各列條列稱為此資料名詞的成員(item)，而加總指令作用於名詞的各個成員。右中刮號] 為複製指令，將名詞 x 的內容複製到結果區域顯示出來。

任一資料名詞，其成員為沿著其第一軸上的各個子陣列，每個子陣列為內容少一維數之另一個資料名詞。條列的成員為其內之元素，表單的成員為其內之條列，而 $2 \times 3 \times 4$ 的陣列，成員有二，各為 3×4 的表單。故一個名詞的成員，其秩較原名詞少一。

```
2 3 4$i.24
0 1 2 3
4 5 6 7
8 9 10 11

12 13 14 15
16 17 18 19
20 21 22 23
```

例中 i. 為指標函數，由零起算。故 i.24 得到前 24 個非負整數，亦即 0 到 23。故上式的意思是將 24 個指標指定給 2 個 3×4 的表單。在這個陣列裡，成員為表單。

表單的成員為條列，亦即表單中的各列。

```
3 4$i.12
0 1 2 3
4 5 6 7
8 9 10 11
```

在此表單中，有三個成員，每一個成員都是有四個元素的條列。

單邊動詞計數(#)可以用來計算資料名詞內成員的個數。例如

```
i.2 3
0 1 2
3 4 5
```

得到一個表單，而此表單的成員有二，為兩個條列。

```
#i.2 3
2
```

動詞的雙價型

動詞的定義，可以因為只有右邊有引數，或者雙邊都有引數，而有不同。若只有右邊有引數，稱之為單邊(monad)動詞，若必須雙邊都有引數，稱之為雙邊(dyad)動詞。一個動詞若單邊或雙邊都有定義，稱之為雙價(ambivalence)。

以+為例。單邊+得到其引數的共軛數，

```
+3j4
3j_4
```

而雙邊+則為其左右引數之和。由於+的單邊與雙邊都有定義，故為一雙價動詞。

一個代表動詞的字元符號，依其尾飾的類型以及引數的單、雙邊型，可以有六種不同的定義，這大大的增加了侷限於標準碼的J語言的豐富性及複雜度。雖然如此，在設計上此六種定義往往是有關連的。仍然以+為例，單邊+將複數化為向量，雙邊+得到引數的最大公約數，同時也是邏輯運算元的「或」；單邊+:為雙倍指令，雙邊+:為邏輯運算元非或。

```
+. 3j4 5j6 7j8
3 4
5 6
7 8
```

在結果的表格中，每一列為對應複數的向量。

```
9 +. 15          NB. 最大公約數
3
+.: 2 3 4        NB. 一倍
4 6 8
1 +: 0           NB. 邏輯非或
0
```

副詞與雙價動詞

前面提到副詞修飾動詞，得到另一個意義類似的動詞，並以動詞+爲例，單邊+爲共軛數，但單邊+ / 爲加總。修飾單邊動詞的副詞稱之爲單邊副詞，探究單邊副詞 / 的功能，是將所修飾的動詞插入其右引數的成員間，即+ / 1 2 3 同於 1+2+3，故稱之爲插入副詞。修飾雙邊動詞的副詞 / 稱爲作表副詞，其將左右引數各取一個成員兩兩成對做雙邊運算，構成一結果表單。以乘法*爲例，雙邊* / 爲作表乘，很容易就可以得到乘法表，

```
(1+i.9)*/(1+i.9)
1 2 3 4 5 6 7 8 9
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

其中 1+i.9 爲由 1 到 9 的整數。

動詞串列

兩個或以上的動詞串成一列稱爲動詞串列(train)，一般而言，當 J 見到動詞串列，它會由右到左依序執行。但若將動詞串以小刮號刮起來，與名詞分開，則形成一個新的動詞，其執行順序就不再是單純的由右到左，而會先將若干動詞串作合併，另有不同的執行順序。

兩個動詞併在一塊的串列稱之爲勾串(hook)，三個動詞併在一塊的串列稱之爲叉串(fork)。勾與叉的執行順序有不同的定義，以下以 f、g 代表動詞，x、y 代表名詞，分別說明之。勾串的執行方式爲

(f g) y 相當於 y f g y

x (f g) y 相當於 x f g y

因此(% +)y 等於 y% +/y，而 x(% +)y 同於 x % +/y。

```

% +/ 2 3          NB. 由右到左
0.2
(% +/) 2 3       NB. 單邊勾串
0.4 0.6
1 (% +/) 2 3     NB. 雙邊勾串
0.2

```

叉串的執行方式為

(f g h) y 的分解動作是 (f y) g (h y)
x (f g h) y 的分解動作是 (x f y) g (x h y)。

勾串及叉串在某些應用上非常方便，例如單邊叉串(+ / % #)可用來求平均數，其意義為加總資料+ / 除以%計算資料個數#。

```

(+ / % #) 1 2 3 4 5
3

```

我們將上式再複雜化一點，前面加一個減號

```

(- + / % #) 1 2 3 4 5
_2 _1 0 1 2

```

將後三個動詞視為一個求平均數的叉串動詞，則此四個動詞串等於減號與求平均兩動詞形成的單邊勾串動詞。其意義為所有資料減去平均數，亦即為均差。

若我們要求均差方，必須使用平方動詞 *:，先直接加進去看看

```

(*: - + / % #) 1 2 3 4 5
_2 1 6 13 22

```

結果不是我們所預期的。這是因為*:和 - 與+ / % #三個動詞構成另一個單邊叉串，而不是我們想要的取平方*:於均差- + / % #。也就是說，動詞串的切割，叉串優先於勾串，在優先於單字。若要將叉串拆成單字配勾串，可以在兩者間加入一個作用連接詞@。故

```

(*: @ - + / % #) 1 2 3 4 5
4 1 0 1 4

```

就可以得到我們想要的均差方。

動詞選項

有的時候我們並不希望所有的動詞都依序執行，而要看看前方的結果來決定適當的後續工作，這就需要動詞選項指令@.。

動詞選項指令的作用，是將其右引數執行結果的數字用來選擇左邊的動詞選項。故其左引數為以「\`」分隔的數個動詞串，作為供選擇的工作項目，而這些項目的編號依序為0、1、...，以作為右引數結果的參照。

例如下例，動詞串0為輸出恆為0的常數函數，動詞串1則為雙倍函數，動詞串2為減半函數。選項指令右方的單邊*，用來判定符號，如果正數得到1，呼叫指令串1，即雙倍函數；負數得到-1，呼叫倒數第1個指令，即減半函數。0的符號為0，呼叫動詞串0，送出0的結果。

```
0:`+:`-:@.*_4_2 0 2 4
_2_1 0 4 8
```

動詞選項可以用來作遞迴運算，以下用一個計算階層的指令作為寫遞迴程式的範例：

```
1:`([*$:@<:.)@.* 5
120
```

上例中常數函數 1: 的編號為 0，動詞串]*\$:@<: 的編號為 1。要執行哪一件工作，由 @. 的右引數執行結果的數字決定。例中右引數 *5 的結果為 1，故先執行編號為 1 的動詞。在動詞串 1 當中，] 複製右引數的資料 5，而 \$: 取得右引數的資料 5 在記憶體的地址，帶入減一函數，成為 4，兩項相乘。再將參照地址內容判斷是否大於 0，反覆執行直到等於 0，則取 1，並結束程式執行。

動詞的秩

動詞加總+ / 作用於一矩陣，到底得到列和、行和或總和？我們先定義一個資料表

```
data=.2 3$i.6
data
0 1 2
3 4 5
```

對之加總，得到

```
+ / data
3 5 7
```

由結果很容易看出，預設的加總對象是資料表的成員，0 1 2 與 3 4 5 兩條列，故得到行加總。那我們要怎麼做列加總？指令為

```
+ /"1 data
3 12
```

上式中我們用了一個秩連接詞 "。秩連接詞連接動詞與秩參數，產生一個限定受詞的新動詞。本例秩連接詞連接動詞加總+ / 與秩數 1，得到新的動詞秩 1 加總，指定加總的受詞為右引數 data 中所有秩為 1 的名詞，即 0 1 2 與 3 4 5 兩條列。條列的成員為元素，故對條列加總相當於將條列內的元素加總，故兩條列分別加總得到 3 與 12。

那秩 0 加總又為何？

```
+ /"0 data
0 1 2
3 4 5
```

得到的是原來的陣列。這是因為秩 0 受詞為各個元素，各自都只有單一元素，對單一元素加總得到的是原來的單一元素。

此例中秩 2 加總的結果同於不限制秩的加總，因為 data 本身為秩 2。

```
+/ "2 data
3 5 7
```

若資料為三維，如

```
]data=.2 3 4 $i.24
0 1 2 3
4 5 6 7
8 9 10 11

12 13 14 15
16 17 18 19
20 21 22 23
```

則秩 3 加總同於不限制秩的加總，為 2 個 3x4 表單之和，仍為 3x4 的表單。

```
+/ "3 data
12 14 16 18
20 22 24 26
28 30 32 34
```

秩 2 加總為分別對各表單的成員做加總，故得到 2x4 的表單。

```
+/ "2 data
12 15 18 21
48 51 54 57
```

秩 1 加總為分別對各條列的成員做加總，故得到 2x3 的表單。

```
+/ "1 data
6 22 38
54 70 86
```

秩 0 加總對各單一元素分別做加總動作，故得到原來的 2x3x4 陣列。

詞彙管理

在工作時程中所定義的詞彙，很可能過一會兒就忘了，要如何查看目前有哪些定義過的詞彙呢？可用 4!:1 定義查看指令，例如 names：

```
names=: 4!:1
```

有了察看指令，右引數為要查詢的詞類，0 為名詞，1 為副詞，2 為連接詞，3 為動詞。例如

```
names 0
+---+---+
|a|b|t|ri|
+---+---+
names 3
+-----+-----+
|names|sum|
+-----+-----+
```

或者想要查詢某個詞彙的詞性，用

```
(4!:0) <'names'
3
```

記得名字是加框的字串；如果要同時查兩個或以上的名字屬性，用分號分隔字串也可以得到加框的效果。

```
(4!:0) 'names';'a'
3 0
```

結果除了以上的詞類外，還有_2 代表不正確，_1 代表未使用，6 代表區位。

至於定義的字彙如何刪除呢？可以 4!:55 定義刪除指令如下：

```
erase=: 4!:55
```

刪除字彙'tri'的方法為

```
erase <'tri'
1
```

結果 1 代表成功地刪除了。我們再來看看還剩哪些名詞與動詞，

```
names 0 3
+---+-----+-----+---+
|a|b|erase|names|sum|
+---+-----+-----+---+
```

如果我們要刪除所有的名詞，可將 names 0 置於 erase 的右方，

```
erase names 0
1 1
```

現在再來看看還剩下什麼名詞與動詞，

```
names 0 3
+-----+-----+---+
|erase|names|sum|
+-----+-----+---+
```

只剩下動詞了。

如果想要刪除所有的名字，用 4!:56 加上空白字串”即可。我們先看看有哪些個名字，

```
names 0 1 2 3 6
+---+---+---+---+---+---+---+---+---+---+---+---+
|a|base|j|j|browser|names|perm|x|z|
+---+---+---+---+---+---+---+---+---+---+---+---+
```

下刪除指令後再看看

```
4!:56 ''
names 0 1 2 3 6
|value error: names
| names 0 1 2 3 6
```

先前所定義的動詞 **names** 不能再使用了。直接用外務詞

```
(4!:1) 0 1 2 3 6
+---+---+
|base|z|
+---+---+
```

記得要將外務詞以刮號來與其右引數區隔，否則會有錯誤。刪除全部名字之後，只剩下 J 系統的基本場合名字 **base** 與 **z** 了。

文稿檔

在工作區所鍵入的單字定義，一旦關掉視窗後就沒有了。若同樣的單字要重複使用，每次都要重新鍵入將不勝其繁。J 以文稿檔(script file)的方式將單字定義儲存於檔案中。在工作區時，執行選單 File->Save As...，並指定一檔名，即可將工作區的所有單字定義儲存於指定的檔內。

文稿檔的副檔名為 ijs，要載入其中的單字定義，在工作區執行指令

```
load'c:\j405\temp\test.ijs'
```

其中 test.ijs 為範例檔名。檔名包括完整目錄，並以單引號刮起來，表示檔名為字串名詞。

文稿檔為標準文字檔，任何編輯程式，如微軟視窗所附送的記事本，或者共享軟體如 UltraEdit，都可以用來修改。J 系統本身也有提供編輯文稿檔的功能，只要執行選單 File->Open，或按鍵 Ctrl+O，開啓要修改的文稿檔，得到一新視窗稱為文稿區，在文稿區內即可修改各種單字定義，並以選單儲存。

文稿區的定義並沒有載入執行區，因此開啓檔案後，裡面定義的指令還不能使用。必須執行選單 Run->Window，或按鍵 Ctrl+W，才會將文稿區的定義載入執行區，定義的指令才能夠使用。在文稿區修改定義後，也是不能夠馬上生效，必須重新 Run 過之後，才能夠變更執行區的定義。

當我們將工作區儲存的時候，原來的工作區就會被改名成文稿檔，而不再有工作區可以作及時交談的工作。如果此時想要繼續工作，必須另開一個新的工作區，並在文稿區執行 Run->Window 以載入先前的定義。

在文稿檔作定義，應以全域指定動詞=:取代區域指定動詞=。否則文稿區指定的單字定義將無法載入工作區。例如，定義 test.ijs 的內容為

```
x=.1 2 3
y=:4 5 6
```

在工作區 1.ijs，執行以下指令

```
load'c:\j405\temp\test.ijs'
x
| Value error: x
y
4 5 6
```

以區域動詞定義的 x，在載入過程中執行完畢就消失了，只有全域動詞所定義的單字，載入後依然存在。

要知道硬碟上有啥劇本檔可供載入，可以指令「scripts」察看。例如列出以 v 開始之劇本檔指令為

```
scripts 'v'
bmp          c:\j406\system\packages\graphics\bmp.ijs      z
colib        c:\j406\system\main\colib.ijs              z
color16      c:\j406\system\packages\color\color16.ijs
```

每次進入新的工作區段時，都會自動將 profile.ijs 的內容載入工作區。使用者可以將自己常用的自訂定義放在該檔案裡，如此就不必每次都需要手動來載入了。

系統

外務連接詞

除了前述名詞、動詞、副詞等以外，還有一些指令與作業系統關係密切，稱之為外務詞「!:」。例如，存取檔案、查詢檔名、時間、計算使用時間等等都是以外務詞來完成。

外務詞的引數為整數，左邊引數指定工作類型，右邊引數則選擇工作，不同的引數定義不同的動詞，除了 5!:0 為副詞。所定義的動詞，就如同其他動詞一樣，可以讀入引數處理後輸出，也可以被副詞或連接詞修飾。

以不同的左引數來區分工作類型，有以下幾類：

0!:	文稿	7!:	空間
1!:	檔案	9!:	全域參數
2!:	主機	11!:	視窗驅動
3!:	轉換	13!:	除錯

4!:	名稱	15!:	動態連結館 DLL
5!:	表現	18!:	場合
6!:	時間	128!:	數值函數

以下將選擇幾項比較重要的項目來敘述。

時間

與時間有關的外務詞，重要的有查詢系統時間的指令：

```
6!:0 ''
2001 6 26 16 50 12.107
```

結果為西元年、月、日、時、分、秒。即使不需要右引數，也必須輸入空字串「''」。

若要衡量指令的執行時間，可用計時外務詞(6!:2)，

```
x=:?50 50$100
10 (6!:2) '%.x'
0.0307812
```

上例先產生一個 50x50 的矩陣 x，每個元素都隨機取自前 100 個指標數(非負整數)，再詢問計算反矩陣的時間。計時指令的左引數為重複執行的次數，右引數以單引號將執行指令刮起來，結果得到平均執行時間。

如果想要知道從進入 J 系統，到現在已經工作多久，指令為

```
6!:1''
3831.71
```

夠久了，該休息啦。

空間

空間指得是系統記憶體空間。要知道目前使用多少記憶體空間，用

```
7!:0 ''
369728
```

這個 J 時段到目前已經用了多少空間，指令為

```
7!:1 ''
7392192
```

查詢執行一個工作所用的記憶空間

```
7!:2 ',/x' [ x=: 3000 2 5$'kerygmatic'  
33792
```

被查詢的工作指令，以文字格式作為右引數輸入。

J 記憶管理程式以區塊大小以及數量來管理記憶體，要顯示其剩餘情形，指令為

```
7!:3 ''  
64 3514  
128 1376  
256 239  
512 125  
1024 58
```

顯示的結果有兩欄，結果第一欄為區塊大小，第二欄為剩餘的數量。

檔案

資料檔的存取

除了以文字編輯器產生文字檔外，J 的工作區也可以指令產生文字檔。定義讀檔指令

```
readfile =: 1!:1
```

及寫檔指令

```
writefile =: 1!:2
```

檔名是以加框字串來定義，例如

```
]fn =. < 'user\test.txt'  
+-----+  
|user\test.txt|  
+-----+
```

式子中的] fn，其中 fn 定義檔名變數，而] 的作用是將其內容顯示出來。

有了檔名，就可以直接將資料以字串的形式寫進檔案裡，指令為

```
'testing 1 2 3' writefile fn
```

左引數為要寫進檔案的來源字串，右引數為目標檔名。

儲存的資料要如何取出呢？要將資料由檔案中讀回工作區，方法為

```
data =. readfile fn  
data  
testing 1 2 3
```

文稿檔動詞

前面敘述執行文稿檔的方式，是以檔案輸入的方式，而我們也可以用外務連接詞「0!」直接載入文稿檔來執行。0!:0 表示由檔案輸入，右引數以加框的文字串作為檔名，例如如<'filename.ijs'。因此若要執行此檔，指令為

```
0!:0<'filename.ijs'
```

相對於由檔案入，指令 0!:1 為名詞輸入，右引數名詞的內容會被當作指令來執行。例如

```
0!:1 '+/ 1 3 5'  
+/ 1 3 5  
9
```

一般而言，0!:0 與 0!:1 的後面可以再加上兩個二位元，連同第一個位元，其意義如下表：

	第一位元	第二位元	第三位元
0	執行檔案	遇錯則終止	緘默
1	執行名詞	執行到底	顯示

故 0!:111 abc 會完整地執行名詞 abc，並顯示過程與結果。

每次進入 J 的新的工作區段，預設的載入檔案指令為 0!:0<'profile.ijs'，亦即，緘默、遇錯則中止地載入 profile.ijs。

檔案管理

除了之前所介紹的讀入與寫出，其他與檔案有關比較重要的指令有顯示目錄動詞 1!:0，功能類似 DOS 指令的 DIR，例如

```
1!:0 'j.*'  
+-----+-----+-----+-----+-----+  
|j.dl1|2001 7 18 21 40 22|765952|rw-|-----a|  
+-----+-----+-----+-----+-----+  
|j.exe|2001 7 18 21 33 18|684032|rwx|-----a|  
+-----+-----+-----+-----+-----+
```

當資料寫入檔案時，如果以 1!:2 作，會將舊檔蓋掉，如果要附加在原來檔案之後，指令為 1!:3。

如果想要產生新的目錄，用 1!:5，檔名必須為加框字串，例如

```
1:5 <'test'  
1
```

產生目錄 test。相對地，刪除檔案或目錄的指令為 1!:55，故

```
1:5 <'test'  
1
```

會將剛剛產生的目錄 test 刪除。以上兩個指令執行的結果顯示 1，表示執行成功。

程式

隱式與顯式

程式語言最重要的功能，就是使用者能夠依據所遇到的問題，自行設計解決問題的程式。那 J 語言如何編寫程式呢？J 解決問題用的是各種單字，當我們在定義單字的時候，我們就是在寫程式了。例如當我們定義

```
average=:+ / % #
```

我們就得到計算平均數的程式。此一種動詞定義，並沒有明顯的宣告，而引數也沒有進入定義中，故稱隱式(tacit form)定義。

相對於隱式定義，J 另外提供顯式(explicit form)定義動詞的方法，其方式是利用定義連接詞 `:`。定義連接詞的左引數以一個數字來指定所定義單字的詞類，如單字定義 `s=:0:0` 定義 `s` 為文稿，用 `4:0` 定義雙邊動詞，用 `3:0` 定義動詞，用 `2:0` 定義連接詞，用 `1:0` 定義副詞，而用 `0:0` 定義名詞。左引數 13 到 10 的意義類似 3 到 0，只是將結果儘可能轉換成隱式定義。

顯式定義與隱式定義最大的差別在於，顯式定義的程式中左引數以「x.」代表，右引數以「y.」代表，而隱式定義完全不用引數。

右引數 0 指定定義由鍵盤輸入，以「0」接在「:」後面，程式另起新行，最後以 `)` 獨自為一行來結束。以下以定義面積平方公尺轉坪數的單邊指令為例，

```
centigrade =. 3 : 0  
t1 =. y. - 32  
t2 =. t1 * 5  
t3 =. t2 % 9  
)
```

若程式只有一列，則可省略 0，以單引號將程式兩邊括起來成字串，直接接在 `:` 後面。

在定義中的指定指令用區域指定，故前例中的 t1、t2、t3 為區域變數，只暫存於程式執行過程，如果工作區有同名變數，兩者並不衝突。

```
mcentigrade =. 3 : '(5%9) *y.-32'
```

在動詞雙價定義中，以：分隔單邊與雙邊定義。以下為減號的文字版

```
minus =. 3 : 0
- y.
:
x. - y.
)
```

在工作區新定義的單字，只暫存在電腦中，必須儲存於文稿檔，才能長久儲存，並供以後重複使用。

工作區的指定單字，要嘛由文稿檔載入，不然就重新定義，一旦工作區有了定義，就可以呼叫使用。要將指定單字的定義由工作區刪除，指令為 erase。下例顯示指定單字 centigrade 刪除前後執行的差異。

```
centigrade 50
10
erase 'centigrade'
1
centigrade 50
|value error: centigrade
| centigrade 50
```

邏輯結構控制

邏輯結構控制為程式最重要的功能之一，讓程式可以依照不同的條件作不同的事情，當然在 J 語言的顯式定義中，也有許多好用的控制結構。

J 語言的控制結構以指令 if.、try.、while.、whilst、for.、for_i. 或者 select. 開始，而以相對的 end. 結束。J 的結構控制指令共有九種：

```
if. T do. B end.
if. T do. B else. B1 end.
if. T do. B elseif. T1 do. B1 elseif. T2 do. B2 end.
try. B catch. B1 end.
while. T do. B end.
whilst. T do. B end.
for. T do. B end.
for_i. T do. B end.

select. T
case. T0 do. B0
case. T1 do. B1
fcase. T2 do. B2
case. T3 do. B3
```

end.

以上以 T 或 B 開頭的字，代表任何長度的程式區塊，包括其他的控制結構。T 區塊最後一條程式的結果決定了下一步的動作，包括程式是否終了。

控制結構 **try**. 會先執行 B 區塊程式，如果沒有錯誤，就會跳到程式結束，否則就會執行 B1 程式區塊。

控制結構 **while**. 先執行 T 區塊，如果結果不是 0，則執行 B 區塊，再回頭執行 T 區塊，不斷重複，只要 T 區塊得到 0，程式就立刻終止。

控制結構 **whilst**. 類似於 **while**.，差別在於跳過第一次的 T 區塊，故 B 區塊至少會被執行一次。

這些邏輯結構控制指令與其他程式語言雷同，限於篇幅就留待以後有機會再行詳細介紹。

圖形使用者介面

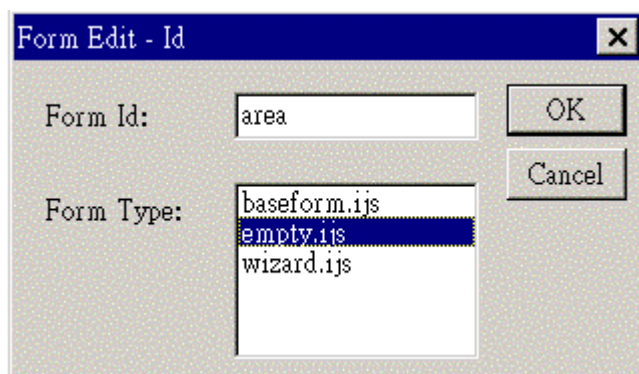
視窗當道的現在，程式必須要有圖形使用者介面(GUI)才算完整。J 語言有提供 GUI 的功能，以下將以一個轉換面積的坪數與平方公尺的例子來說明其用法。

編寫 GUI 的第一步是編寫表格定義(form definition)。表格定義同其他單字定義一樣，儲存在文稿檔。

新建一文稿檔，將之存到使用者目錄，並呼叫表格編輯器。

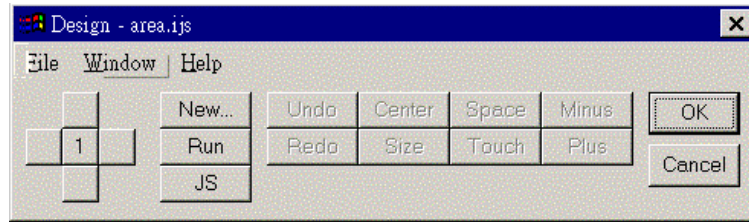
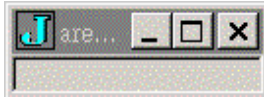
- 以 **File->New IJS** 建立新的文稿檔
- 以 **File->Save As...**將之存於使用者目錄 user，命名為 area.ijs
- 以 **Edit->Form Edit** 啓動表格編輯器

銀幕上應該出現表格編輯器的對話框



- 在 Form Id 右方空格鍵入 area 做為表格名稱
- 表格類型 Form Type 由右方的列表格中選空白表格 empty.js。
- 按 OK 結束此對話表格

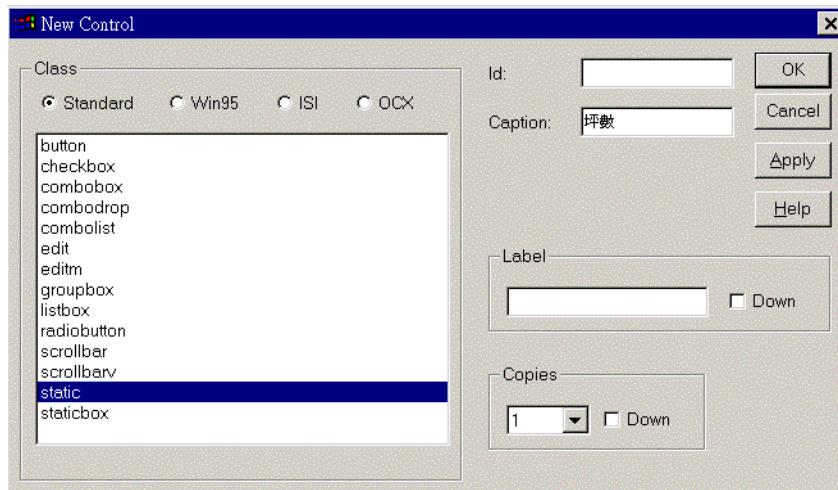
此時銀幕會新開兩個視窗，一個位於左上角為 GUI 的雛形表格，另一個位於銀幕正中央，為設計指令中心。：



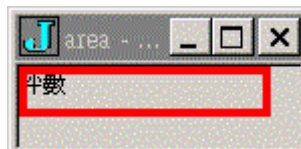
設計指令中心用來修飾 GUI 表格，修改後的表格定義將會加入文稿檔 area.ijs 中。

接著要在 GUI 表格中新增一個靜態控制物件(static control)——坪數標籤。靜態控制物件用來給其他控制物件作標籤。

- 在設計對話表格按 New...
- 於 New Control 的 Class 框中，點 Standard，並選 static
- 將坪數鍵入 caption 編輯格
- 按 OK



坪數標籤控制物件出現在 GUI 表格的左上角。



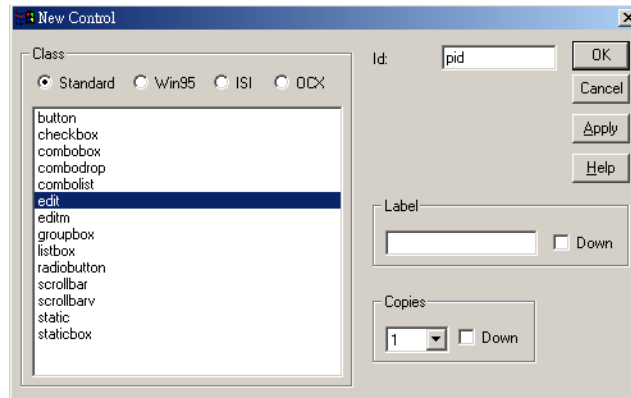
標籤的位置似乎不太理想。控制物件的位置可以滑鼠來拖拉(drag)，即先用滑鼠點，再按左鍵不放並移動滑鼠。

- 往左下方拖拉坪數標籤移動一些。

再來新建一個編輯控制物件，以作為坪數輸出入之用。命物件識別名(id)為 pid。此一識別名很重要，用來作控制物件的名字，以及程式指定內容用。

- 按 New... 鍵

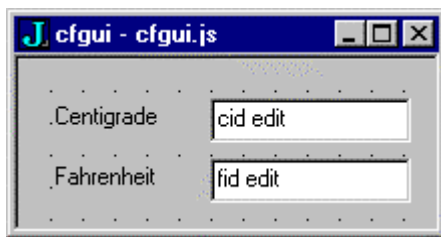
- 在 Class 框 Standard 按鈕下選擇 edit
- 鍵入 pid 為 control id
- 按 OK
- 拖拉 pid 編輯控制物件到標籤控制物件之右方。



類似前面步驟，建立平方公尺的標籤控制物件以及編輯控制物件，標題為平方公尺，識別名為 mid。

- 步驟類似建立坪數控制物件

GUI 表格現在應該如下



表格設計到此結束，現在可以離開表格編輯器並試執行表格。

- 在設計對話框按 OK

表格定義現在應該已進入文稿檔 area.ijs，其內容應類似以下

```
NB. base form
AREA=: 0 : 0
pc area;
xywh 19 8 29 9;cc ccstatic static;cn "坪數:";
xywh 7 19 50 10;cc ccstatic static;cn "平方公尺:";
xywh 39 5 50 11;cc pid edit ws_border es_autohscroll;
xywh 39 17 50 11;cc mid edit ws_border es_autohscroll;
pas 6 6;pcenter;
rem form end;
)
```

```
area_run=: 3 : 0
wd AREA
NB. initialize form here
wd 'pshow;'
)
```

```
area_close=: 3 : 0
wd'pclose'
)
```

其中定義了一個名詞 AREA 與兩個動詞 area_run 、area_close 。

指令行中以 wd 起首者，其後字串為與 GUI 表格定義有關的指令。AREA 為所定義的表格，故 wd AREA 將 GUI 表格在記憶體中準備好，而 wd 'pshow;' 將之送到銀幕。

修改過的文稿檔 area.ijs 尚未執行過，故文稿檔內的定義尚未載入工作區內。要執行新建 GUI 表格，需先載入定義，再執行新定義之指令：

- 在文稿區 area.ijs 執行 **Run->Window**
- 到工作區鍵入 area_run 0

執行 area_run 0 將使前面定義的表格出現於銀幕中央。右引數 0 一定要記得打，否則會出錯。此時鍵入數字不會得到任何結果，因為還沒有定義表格上的控制事件(event)。

要結束程式執行，在工作區鍵入

```
wd 'reset'
```

當鍵入數值到坪數編輯控制物件，按 Enter 後立刻擊發一個事件。事件由表格、控制物件以及事件類型三者共同來識別。在編輯控制物件按 Enter 為一按鈕(button)事件(在編輯欄位按 enter 相當於壓按鈕控制物件)。本例的事件為表格 area 中控制物件 pid 的按鈕事件。

當事件被觸發時，會呼叫事件處理動詞(event handler)，該動詞的名稱由三部分構成：formid_controlid_event。故本例的事件處理動詞為 area_pid_button。

事件處理動詞 area_pid_button 會將 pid 編輯物件內容的數值（坪數面積）轉換成平方公尺面積數值，並將結果顯示在 mid 編輯物件。

定義事件處理動詞，可經由表格編輯器所提供的快捷鍵。當表格編輯器執行時，按 Ctrl 鍵同時以滑鼠點控制物件，立刻會進入文稿區中定義該控制物件事件處理動詞的位置。

咱們回頭瞧瞧整個程序。由於表格編輯器已經關閉，故要重新開啓。點選文稿區 area.ijs，再選擇 **Edit->Form Edit** 以開啓表格編輯器。

- 點選 area.ijs
- 以 **Edit->Form Edit** 啓動表格編輯器

GUI 表格再度在銀幕的角落開啓

- 按 Ctrl 鍵並點選 pid 控制物件

此時會進入文稿區 area.ijs 定義事件處理動詞 area_pid_button 的位置。此動詞執行時，名詞 pid 自動取得編輯欄的內容。內容預設為數字字串，故不能直接用來計算，必須先用文字轉數字的系統單字 ". 將之轉換成數字。單字". 為雙邊，其右引數為要轉換成數字的字串，其左引數為右方引數不能轉換時的替代值。

```
t =. 0 ". pid
```

接著將坪數轉換成平方公尺，

```
t =. t % 0.3025
```

名詞 t 為數字，送到 mid 編輯控制物件前，必須先用系統單字 ": 轉換成文字串，

```
t =. ": t
```

送到編輯欄的指令為

```
wd 'set mid *' , t
```

引數包括命令 set、要設定的控制物件識別名，及要設定的資料 t，其中 * 用來指定其後設定的資料為文字。

整個來說，加入文稿檔 area.ijs 中的事件處理動詞定義為，

```
area_pid_button=: 3 : 0
t =. 0 ". pid
t =. t % 0.3025
t =. ": t
wd 'set mid *', t
)
```

定義完事件處理動詞，按 Ctrl 鍵點選文稿區視窗，即可回到表格編輯器視窗。

- 鍵入 area_pid_button 的定義
- 按 Ctrl 鍵點選文稿區視窗，回到表格編輯器視窗
- 在設計指令中心按 OK

新的定義尚未載入工作區，故文稿區必須執行載入動作。

- 以 **Run->Window** 執行文稿載入

現在可以執行 GUI，在工作區 ijs 視窗執行新的動詞，

```
area_run 0
```

表格出現後，在坪數欄位中輸入度數並按 enter 後，平方公尺數立刻在其欄位中顯現出來。

但是在平方公尺欄位中輸入數字，並不會得到坪數，這是因為我們沒有定義坪數欄位的事件處理動詞 area_pid_button。類似定義前面平方公尺欄位事件處理動詞，我們可以定義如下

```

area_mid_button=: 3 : 0
t =. 0 ". mid
t =. t * 0.3025
wd'set mid *', ": t
)

```

以 **Run->Window** 重新載入文稿區的修正定義到工作區，並在工作區執行

```
area_run 0
```

現在在平方公尺欄位鍵入數值，坪數面積立刻顯示在其對應欄位。

我們可以在 GUI 表格上設計一個結束按鈕。結束事件處理動詞的內容為：

```

area_close_button=: 3 : 0
wd 'pclose'
)

```

其中 wd 指令 pclose (parent close) 關閉表格。

- 在工作區執行 wd 'reset'
- 進入表格編輯視窗 **Edit->Form Edit**
- 按 New... 按鈕
- 在 Class 框 Standard 按鈕下選擇 button
- 鍵入 close 作為控制物件 id
- 在 Caption 欄鍵入 結束
- 按 OK
- 拖拉 結束 按鈕到表格內靠右處
- 按著 Ctrl 鍵同時以滑鼠點選結束按鈕
- 鍵入定義 wd 'pclose'
- 按著 Ctrl 鍵同時以滑鼠點選文稿區以回到表格編輯視窗
- 按設計指令中心之 OK 鈕
- 以 **Run->Window** 執行文稿指令
- 到工作區執行 area_run 0

現在有了結束按鈕，不想做面積轉換的時候，隨時可以壓按鈕結束程式。下回想做面積轉換的時候，以選單 File->Open 到 user 目錄開啓 area.ijs，再以選單 Run->Window 將之載入工作區，即可在工作區鍵入 area_run 0 來執行程式。

公用程式庫

J 系統提供許多有用的公用程式，以文稿檔的方式儲存。我們可以 scripts 指令來看看有哪些可用。

```

scripts ''
bmp          colib          color16      colortab     compare     convert
csv          dates          dd           debug        dir         dll

```

files	format	gl2	gl3	graph	isigraph
jadelib	jdirs	jfgrid	jfiles	jfreport	jinput
jmf	jndict	jpm	jpwd	jselect	jsgrid
jsview	jtable	jtgrid	jvgrid	jview	jwatch
jwdict	jwgrid	jzdict	jzgraph	jzgrid	jzopengl
jzplot	keyfiles	kfiles	loadlib	menu	misc
myutil	nfiles	numeric	odbc	opengl	pack
parts	plot	plotdefs	primitives	print	printf
publish	random	regex	registry	rgb	scriptdoc
socket	statdist	statfns	stats	stdlib	strings
text	trig	validate	winapi	winlib	write

這眾多程式庫每一個都可以好好研究，但如果只能顧名思義，大約猜不出幾個，那我們又怎麼能知道這些程式庫能作些什麼呢？

還好我們有 scriptdoc 程式庫，看名字就知道應該是用來看文稿檔說明文件的程式。故我們先載入該程式

```
load 'scriptdoc'
```

就可以用來看程式庫的說明。例如執行

```
scriptdoc `trig`
```

會開一新視窗顯示以下內容：

file: system\main\trig
trigonometric functions

sin (v) sin

cos (v) cos

tan (v) tan

sinh (v) sinh

cosh (v) cosh

tanh (v) tanh

arcsin (v) arcsin

arccos (v) arccos

arctan (v) arctan

arcsinh (v) arcsinh

arccosh (v) arccosh

arctanh (v) arctanh

pi (n) pi

dfr (v) degrees from radians

rfd (v) radians from degrees

indegrees (a) convert function to use degrees:

sind (v) sin in degrees

cosd (v) cos in degrees

```
tand (v) tan in degrees
```

這是三角函數程式庫，內容第一列為程式庫的儲存位置，第二列為其內容說明，第三列以後為其內容的名字。

名字一般以三欄形式說明，

```
sin (v) sin
```

第一欄為名字，第二欄為類型，第三欄為說明。故前例說內容有動詞 sin，為正弦函數 sin。

一旦知道三角函數的定義置於 trig.ijs 內，因此可以指令 load 載入，

```
load 'trig'
```

載入各種三角函數的定義後，我們就可以直接使用常用的三角函數的名稱，而不必使用不易記憶的三角函數動詞。例如

```
sin 0 0.2p1 0.4p1 0.6p1 0.8p1 1p1  
0 0.587785 0.951057 0.951057 0.587785 0
```

直接用 sin 來計算正弦函數的函數值。

繪圖

一張圖勝過千言萬語，如果資料可以圖形的方式顯示，更容易表達資料內容。J 語言有相當的繪圖功能，能夠將資料繪成商業以及科學用圖形。

圖形指令置於 plot 程式庫，故要先載入，

```
load 'plot'
```

會新增兩個指令：「pd」(Plot Driver)與「plot」。

大部分 2D 或 3D 的簡易圖形可以指令 plot 繪出，如果要繪複雜圖形，則需要指令 pd，該指令用來處理所有低階的繪圖指令。

由於範例需要，我們先載入以下兩程式庫：

```
load 'numeric trig'
```

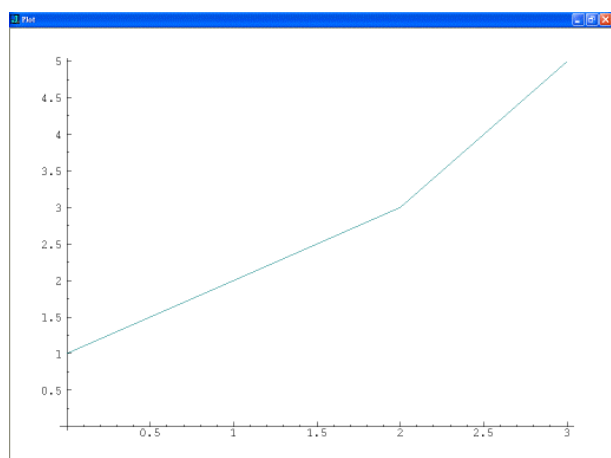
指令 plot 的用法如下：

```
opt plot data
```

右引數為要繪圖的資料，左引數為繪圖選項。例如

```
plot 1 2 3 5
```

圖形會以新視窗的方式展現。當資料為向量時，橫軸為資料個數的指標數，如上例為 0 到 3 的整數。



下個例子要繪出正弦函數，

```
plot sin steps 0 10 100
```

右引數為 `sin steps 0 10 100`，除了正弦函數 `sin`，還用到 `steps` 指令。

指令 `steps` 的右引數為三個數字 `x y z`，結果得到一個間距為 $(y-x)/z$ 的 `x` 到 `y` 的數列，元素個數為 `z+1`。例如

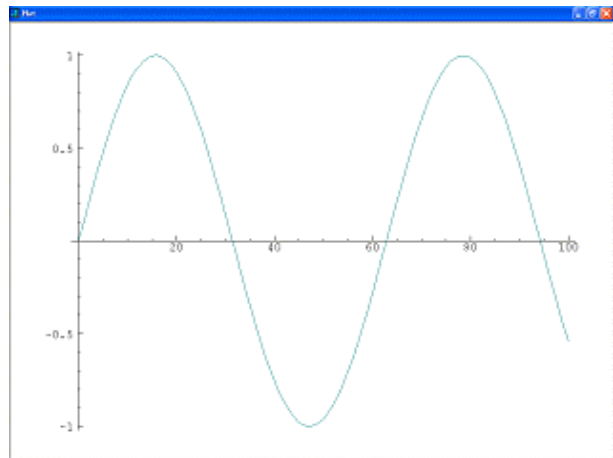
```
steps 1 2 5
```

```
1 1.2 1.4 1.6 1.8 2
```

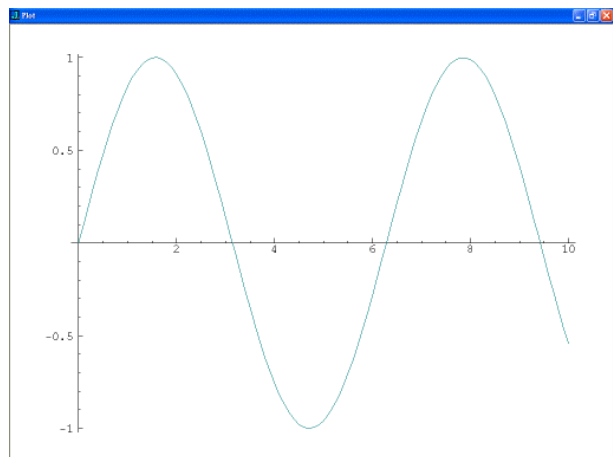
得到由 1 到 2，間距為 0.2 的 6 個元素的數列。所以

```
sin steps 0 10 100
```

只是一個由 0 到 10 間距 0.1 的 101 個元素正弦值的數列。將之以 `plot` 指令描繪出來，就得到正弦函數的圖形。



不過這個圖形的 X 軸座標並不正確，因為橫軸為指標數，故橫軸為 2 時的縱軸並非 $\sin(2)$ ，而是 $\sin(0.02)$ 。我們希望的圖形應該是 $(x, \sin(x))$ ，要如何修正呢？



如果資料為兩個以「;」區隔的向量，構成的兩個向量封包，則第一個向量將被視作 X 軸的資料，而第二個向量則是 Y 軸的資料。

```
x=: steps 0 10 100
```

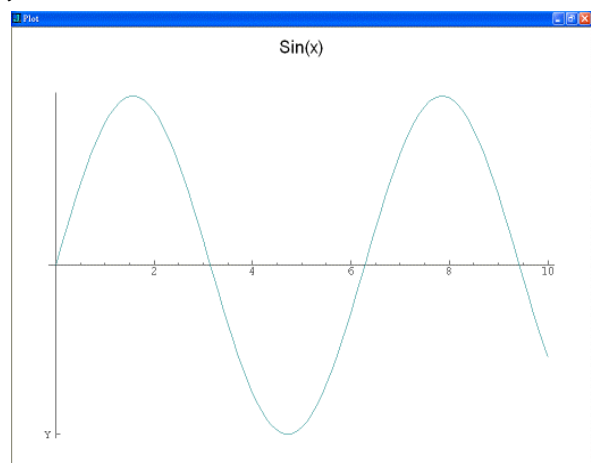
```
plot x;sin x
```

經過這樣的修正，圖形的座標就正確了。

上述指令可以寫得更為精簡：

```
plot (;sin) steps 0 10 100
```

拷貝指令 `]` 與 `sin` 以 `;` 合成兩個指令封包，將右方資料分別代入封包來計算，就得到 `plot` 指令所需要的兩個向量封包。



如果要加上標題與軸名稱，以左引數的字串來定義選項。標題是以 `title` 起頭的任何字串；兩軸分別以 `xlabel` 與 `ylabel` 起頭的字串：

```
'title Sin(x);ylabel Y' plot (;sin) steps 0 10 100
```

應用 J

資料表格

不論是總體經濟或者金融市場的財務資料，或者是實驗數據，都有大量資料以表格的方式儲存，一般田野調查的結果常常也是以 coding sheet 的方式儲存。J 語言如何將資料儲存為表格，以及取出想要用來作分析的部分？

通常資料表的格式，列為觀察值，行為變數，若我們知道變數與觀察值的個數，我們可以前述形狀指令的方式，直接指定資料表的形狀，並將資料依列連續輸入，如

```
a=. 3 2$1 2 3 4 5 6
a
1 2
3 4
5 6
```

但是如果資料太多，隨時可能增加新觀察值或變數，比較方便的方法是以列輸入或者以行(欄)輸入。以列輸入的指令為

```
a=.1 2 3,4 5 6,:7 8 9
a
1 2 3
4 5 6
7 8 9
```

其中 4 5 6 與 7 8 9 兩個條列間的 ,: 為堆疊指令，先將此兩項分別化成表單的一個成員，再將兩表單堆疊成一個新表單。而 , 為附加指令，將條列 1 2 3 加在前面得到的表單上方。

若將附加指令改成堆疊指令將會出錯，因為堆疊指令會將其引數先成員化，即化成較高維數的成員。因此 1 2 3 原為一維，先化成二維，而 4 5 6,:7 8 9 原為二維，先化成三維，二維堆疊到三維陣列的結果，為一個三維陣列。

```

a=.1 2 3 ,:4 5 6,:7 8 9
a
1 2 3
0 0 0

4 5 6
7 8 9

```

以行輸入方式需用編結(stitch)指令,。如

```

a=.1 2 3, .4 5 6,. 7 8 9
a
1 4 7
2 5 8
3 6 9

```

若 a 與 b 為兩資料表。上下相疊的指令為 a,b，相當於增加新觀察值；左右相鄰合併的指令為 a,.b，相當於增加新變數。若兩表原來形狀不同，在差異處補 0，化成同樣形狀後再合併。例如

```

a=.1 3,3 4,:5 6
b=. 7 8 9,:10 11 12
a,b
1 3 0
3 4 0
5 6 0
7 8 9
10 11 12

```

如何從資料表單取出部分資料？取出指令為{，此為雙邊指令，左引數為位置指標，指出要取出的行或列，右邊參數則為資料表名稱。

取出元素：

取出 t 中第 2 列與第 0 列(2 0{t)。

取出 t 中第 2 列與第 0 欄的元素((<2 0){t)。

取出 t 中位於 (2, 0) 與 (1,3) 的兩元素((2 0;1 3){t)。

取出 t 中位於 (2, 1)、(2,3) 與 (0,1)、(0,3) 四元素((<2 0;1 3){t)。

位置指標為列標，亦即觀察值指標。要取變數，必須指定指令作用於表的維數「1」。

```
x1=. 0 1{"1 a
```

NB. 將 0 與 1 兩欄取出，並指定給變數 x1

若取的變數只有 1 欄，取出的結果將無法維持欄的格式，而會成爲一列。故需在前加上指令(n)，其中 n 爲欄爲指標。

```
x2=. (,2){"1 a
```

 NB. 將第2欄取出

```
x2=. -999".3 4 5{"1 a
```

 NB. 由3、4、5欄取出三位數的數字欄，
NB. missing value為 -999

資料檔案

調查資料往往是以一連串的文數字存於檔案中，每一列代表一觀察值，而若干行構成一觀察值的某一變數。要如何才能由中取出所要的資料？令我們有一個置於usr目錄下的資料檔test.dat，其內容為

```
xxx1234  
yyy5678  
zzz 901
```

假設前三欄為人名，後四欄為所得。

J系統提供一專門處理檔案輸出入的公用程式文稿檔files，我們第一步就要先載入檔案處理指令。

```
load 'files'
```

接著將資料由檔案user/test.dat載入，成為一個文字矩陣

```
x=. 'm' fread 'user/test.dat'
```

其中fread為files中所定義的指令，左引數為參數'm'，指定載入資料成一矩陣。我們現在得到一個資料變數x。

取出的資料為文字串，如果要運算，必須先化為數字，指令為"。

敘述統計

面對大量觀察資料，如何以最簡單的數據來描述資料，統計上有一些標準的統計量，如平均平均數、中位數、變異數、相關係數等，利用J指令可以很容易地得到這些所謂的敘述統計量。

算數平均數可以用前面所提的又串動詞來定義：

```
mean=: +/ % #
```

```
mean 1 2 3
```

```
2
```

幾何平均數 $\sqrt[n]{\prod x_i}$ ，J以*/計算元素乘積，以#計算元素個數，再以雙邊指令%:計算元素個數的開方根，以又串動詞定義為#%:*/，樣子與算術平均數很類似，若將#與*/對調更像，而%:~中~的功能就是將%:的左與右引數對調。。

```
geomean=: */ %:~ #
geomean 1 2 3
1.81712
```

調和平均數為倒數平均數的倒數。先求倒數再求平均數，可視為平均函數與倒數函數的合成函數，J 提供連接詞&來合成兩個函數，例如，mean & %。而&. 則將合成函數的結果代入第二個函數的反函數，在本例子為倒數函數的反函數，仍然為倒數函數，故 mean &. %為倒數平均數的倒數。

```
harmean=: mean &. (%"_ )
harmean 1 2 3
1.63636
```

為什麼(%"_)要在倒數函數的後面加上"_? 此動詞的秩副詞，若不加此，合成函數對右引數的每一個成員作用，單一元素倒數平均的倒數為原來的元素，看起來好像啥也沒做。

```
(mean &. %) 1 2 3
1 2 3
```

均差

```
dev=: -"_1 _ mean
```

均差方和

```
ssdev=: +/ @: *: @ dev
```

變異數

```
var=: ssdev % <:@#
```

標準差

```
stddev=: %: @ var
```

最小值

```
min=: <./
```

最大值

```
max=: >./
```

中間

```
midpt=: -:@<:@#
```

中位數

```
median=: -:@(+/@)((<. , >.)@midpt { /:~)
```

```
NB. descriptive statistics
dstat=: 3 : 0
t=. '/sample size/minimum/maximum/median/mean'
t=. t, '/std dev'
t=. ,&': ' ;_1 t
v=. $,min,max,median,mean,stddev
t,. ": ,. v y.
)
```

```

dstat 1 2 3 4 5 6 5 4 5 3
sample size:      10
minimum:         1
maximum:         6
median:          4
mean:            3.8
std dev:         1.54919

```

排序

一堆觀察值，除了最大、最小、平均數、中位數等統計量外，常常用到的還有將數字排序。單邊動詞「/:」將其右引數中的數字，由小到大循序列出其位置指標列。例如

```

/: 13 41 25 1
3 0 2 1

```

最小的數字 1 的位置指標為 3，次小數字 13 的位置指標為 0，第 3 小數字 25 的位置指標為 2，而最大數字 41 的位置指標為 1。

雙邊動詞「/:」將其右引數得到的循序位置指標，拿來選取左引數的元素，也就是依右引數的大小順序來排列左引數。例如

```

1 2 3 4 /: 13 41 25 1
4 1 3 2

```

右引數的循序位置指標由前例知道為 3 0 2 1，而左引數位置指標為 3 者是 4，為 0 者是 1，為 2 者是 3，而為 3 者是 4。

如果左右引數相同，即是以自身的循序位置指標來作排列，也就是排序。

```

13 41 25 1 /: 13 41 25 1
1 13 25 41

```

由於雙邊引數相同，J 提供一反身副詞「~」來修飾動詞，得到排序單邊動詞「/:~」，其功能同於左右引數相同的排序雙邊動詞「/:」。所以前例可以進一步簡化成

```

/:~ 13 41 25 1
1 13 25 41

```

次數分配

NB. frequency count (value, frequency) sorted by decreasing frequency

```

freqcount=: (\: {"1})@(~. ,. #/.~)

```

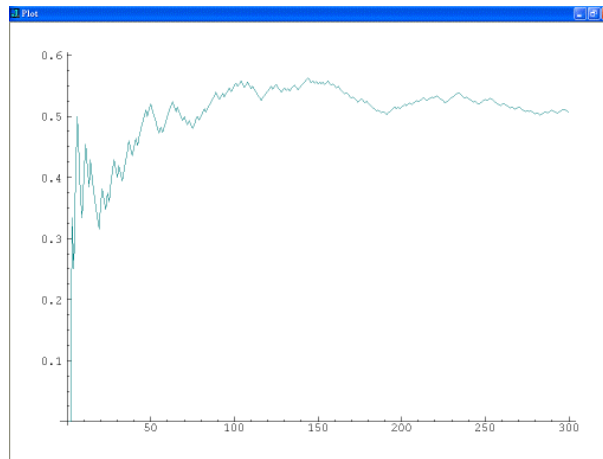
```

fr=:+"1 @ (=)
x=. 1 1 2 2 2 3 4 4 5 5 5 6
1 2 3 4 5 fr x

```

23124 投硬幣實驗

```
N=:300          NB. 試行次數
]tossnum=:>:i.N  NB. 由1開始的試行次數
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 2...
]heads=:+/\?N$2  NB. 人頭的次數
0 1 1 2 2 2 3 4 5 5 6 7 7 7 8 9 9 9 9 10 11 12 13 14 14 15 15 1...
]ratio=:heads%tossnum  NB. 出現人頭的比率
0 0.5 0.333333 0.5 0.4 0.333333 0.428571 0.5 0.555556 0.5 0.5...
load 'plot'     NB. 載入plot程式庫
plot tossnum;ratio  NB. 描圖顯示出現人頭的比率對照試行次數
```



這個圖是統計有名的大數法則的展示，樣本平均數會隨者樣本數增加而趨向母體平均數。

枝葉圖

將一群數字分組，每十分為一組，因此二位數的數字，十位數為枝，而個位數為葉，以枝為縱軸葉為橫軸所作的圖表，稱之為枝葉圖。

```

]marks=.60+?30$40          NB.隨機產生30個60到99的數字
93 98 73 81 81 92 97 86 69 97 83 65 69 95 67 66 93 84 84 93 86 75 94
86 64 60 87 85 78 82
stem=:10&* @ <. @ %&10      NB.取枝，除以10後取下整數在乘以10
leaf=:10&l                  NB.取葉，以10除取餘數
sldiagram =:~.@stem;"0 stem </. leaf      NB. 定義枝葉圖指令
sldiagram /:~ marks        NB.先將資料marks排序，再畫枝葉圖
+---+-----+
|60|0 4 5 6 7 9 9          |
+---+-----+
|70|3 5 8                  |
+---+-----+
|80|1 1 2 3 4 4 5 6 6 6 7|
+---+-----+
|90|2 3 3 3 4 5 7 7 8     |
+---+-----+

```

定義枝葉圖指令比較複雜，需要略微說明。指令 sldiagram 的定義分成三部分，「~.@stem」「;"0」與「stem </. Leaf」。其中 stem </. leaf 為三叉動詞，取枝動詞取出每一個數字的十位數，

```

stem /:~marks
60 60 60 60 60 60 60 70 70 70 80 80 80 80 80 80 80 80 80 80 90 90
90 90 90 90 90 90 90

```

取葉動詞取出每一個數字的個位數

```

leaf /:~marks
0 4 5 6 7 9 9 3 5 8 1 1 2 3 4 4 5 6 6 6 7 2 3 3 3 4 5 7 7 8

```

中間的雙邊動詞「</.」為分類封包，依其左引數的分類來將右引數分裝，

```

(stem </. leaf) /:~marks
+-----+-----+-----+-----+
|10 4 5 6 7 9 9|3 5 8|1 1 2 3 4 4 5 6 6 6 7|2 3 3 3 4 5 7 7 8|
+-----+-----+-----+-----+

```

例如枝幹 60 的位置指標為 0 到 6，而相同位置的葉片數字為 0 4 5 6 7 9 9，合成一個封包。而四種不同的枝幹合成四個封包。

指令~.@stem 中「~.」為取出相異元素，「@」為作用於，故整個指令為於 stem 中取出所有相異元素，也就是取出所有相異的枝幹：

```

(~.@stem)/:~marks
60 70 80 90

```

第二部分的指令「;"0」為被維度副詞「"0」修飾的雙邊連結指令「;」，其左引數為四個相異的枝幹，其右引數為四個葉片封包，維度 0 連結指左右引

數各取一個元素作連結，得到的就是枝葉圖。

迴歸分析

多變量線性迴歸廣泛用在許多學門的研究上，用來尋找應變數與自變數間的關係。對模型 $y = x\beta$ ，迴歸係數可以 $b = (x'x)^{-1}x'y$ 來估計。

上述估計式需要三種運算指令，矩陣轉置 `l'`、矩陣相乘 `+`、`*` 與求逆矩陣 `%`。下例先定義方陣 `x`，求逆，將結果指定給變數 `y`，再送回工作區。

```
[y=%x=.2 2$1 2 3 4
_2 1
1.5 _0.5
```

內積動詞為 `+`、`*`，以下證明 `x` 與其反矩陣 `y` 的內積為單位矩陣。

```
mp= . +/ . *
x mp y
1 0
0 1
```

轉置矩陣指令如下

```
l:x
1 3
2 4
```

以研究體重與身高和年齡的關係為例，體重為應變數，而身高與年齡為自變數。如果有 5 組觀察值，輸入體重資料的指令為

```
y=. 90 55 65 80 62
```

而輸入身高與年齡資料的指令為

```
[x0=.167 45,165 22,170 28,177 36,:158 40
167 45
165 22
170 28
177 36
158 40
```

加入常數項相當於加一欄全為 1 的變數，

```
[x=.1,.x0
1 167 45
1 165 22
1 170 28
1 177 36
1 158 40
```

因此，依照公式迴歸係數為

```
(%.(l:x) mp x) mp (l:x) mp y
_165.875 1.15109 1.27433
```

實際上，J 已經內建 OLS 求迴歸係數的指令，「%.」的雙邊型為「矩陣除」，而 $y\%.x$ 得到 y 對 x 的迴歸係數。

```
y%.x
_165.875 1.15109 1.27433
```

通常迴歸分析還提供其他相關統計量資訊，以下範例為一個完整的 OLS 程式，由於指令繁多，說明將會插在程式碼之間。

```
NB. =====
NB. ols
NB. 格式： 應變數向量 ols 自變數矩陣
NB. 應變數向量= n 個觀察值的向量 (Y值)，以下以5為例
NB. 自變數矩陣= nxp 矩陣，p個自變數的 n 組觀察值(X值)，以下p以3為例
ols=: 4 : 0 NB.定義ols為雙邊動詞
x=. y. NB.定義右引數y.為自變數矩陣x
y=. x. NB.定義左引數x.為應變數向量y
b=. y %.x NB.計算迴歸係數  $b = (x'x)^{-1} x'y$ 

k=. <:{:$x NB.變數個數k為x的形狀向量尾數加1。$x得到5 3，尾數為3
NB.，減去常數項就得到變數的個數。其中{:為取尾數指令，
NB.而<:為減1指令。
n=. $y NB.觀察值個數，為應變數向量的元素個數
sst=. +/*:y-(+/y) % #y NB.總平方和  $\sum (y - \bar{y})^2$ 。指令讀作
NB.y的個數被y的總和除(即平均數)，再被y減(均差)，
NB.取平方後再加總
```

```

sse=. +/*:y-x +/ .* b          NB.估計誤差平方和 $\sum(y-xb)^2$ 
mse=. sse%n->k                NB. 平均估計誤差平方和，為其
                              NB. 平方和除以自由度n-k-1
                              NB.其中>:為加1指令，故>:k得到k+1
seb=. %:({.mse)*(<0 1)|:%.(l:x) +/ .* x
                              NB.估計係數標準誤。迴歸係數變異矩陣為 $\sigma_b^2 = \sigma_e^2(x'x)^{-1}$ ，
                              NB.其對角線即為各係數的估計變異數。上式l:出現兩次，
                              NB.單邊l:x得到x的轉置矩陣，而雙邊(<0 1)l:得到右引數的
                              NB.對角元素。變異矩陣的對角元素即為各估計係數的變異數
                              NB.，開跟號後即得到其標準誤。
ssr=. sst-sse                NB.迴歸平方和
msr=. ssr%k                  NB.平均迴歸平方和
rsq=. ssr%sst                NB.R2值
F=. msr%mse                  NB.F值

```

以下為輸出報表的程式，說明見下段。

```

r=. ,: '          Var.      Coeff.      S.E.      t'
r=. r, 15.0 15.5 15.5 12.2 ": (i.>:k),.b,.seb,.b%seb
r=. r, ' '
r=. r, ' Source      D.F.      S.S.      M.S.      F'
r=. r, 'Regression', 5.0 15.5 15.5 12.2 ": k, ssr,msr,F
r=. r, 'Error      ', 5.0 15.5 15.5": (n-k+1), sse,mse
r=. r, 'Total      ', 5.0 15.5 ": (n-1), sst
r=. r, ' '
r=. r, 'S.E. of estimate      ', 12.5":%:mse
r=. r, 'Corr. coeff. squared', 12.5": rsq
)

```

第一列定義表頭，而「,:」表示其後方的資料將構成只有一列的一個表單，再以附加成員的方式，將其他結果一列列的加入表單中。

計算的的結果可以更清楚的方式顯示出來，主要的關鍵在於使用格式化輸出指令「":」，此為雙邊指令，右引數為要輸出的變數，左引數則為各個變數輸出的格式，下例中 15.5 表示全寬 15，小數點 5 位。

```

r=. r, 15.0 15.5 15.5 12.2 ": (i.>:k),.b,.seb,.b%seb

```

以前面體重分析的資料來執行迴歸，結果如以下：

```

y ols x
      Var.      Coeff.      S.E.      t
      0      _165.87477      82.29568      _2.02
      1       1.15109       0.47770       2.41
      2       1.27433       0.35968       3.54

Source  D.F.      S.S.      M.S.      F
Regression  2      726.10343      363.05171      8.34
Error      2      87.09657      43.54829
Total      4      813.20000

S.E. of estimate      6.59911
Corr. coeff. squared  0.89290

```

三角函數

三角函數的計算，並不需要載入三角函數公用程式庫，J 語言的基本單字已經有支援。三角函數使用雙邊動詞「o.」，由字母 o 尾綴一點，左邊引數指定函數類型，右邊引數則為徑度量。例如 1 o. 代表正弦函數，下例則計算徑度量由 0 到 5 之正弦函數值。

```

1 o. i.6
0 0.841471 0.909297 0.14112 _0.756802 _0.958924

```

左引數 1 2 3 分別代表 sin、cos、tan，而 _1 _2 _3 分別代表 arcsin、arccos 與 arctan。下面式子同時計算此六種函數在徑度量為 0 時的函數值：

```

_1 _2 _3 1 2 3 o. 0
0 1.5708 0 0 1 0

```

計算 e 值

雖然 J 可以用指令 ^1 來計算尤拉常數 e，以下將以其泰勒展式的計算，來展示 J 的級數計算功能。

由泰勒展式我們知道

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

一般而言，取前十項已經足夠接近，我們可以 0 到 9 的倒數數列相加來估算自然對數。以 J 來做，指令如下所示：

```
+ / % ! i.10
2.71828
```

其中「i.10」產生 0 到 9 的數列，稱為取前 10 個指標。接著以階層動詞「!」計算這個指標數列每一元素的階層，並以倒數動詞%分別計算倒數，最後以加總動詞「+ /」得到結果 2.71828。

以數學的眼光看，由右而左的運算是有些奇怪，但是若以英語語法來看，倒是很自然，"the sum of the reciprocals of the factorials of the first 10 indices"，當然如果要以中文來念也是可以，只是必須從右方念起，「前 10 個指標取階層後倒數的加總」。

多項式

泰勒展式可以將任何函數化成多項式的形式，J 又如何定義多項式呢？只要將多項式的係數由低項寫起，以連結動詞「&」連結多項式動詞「p.」即可。例如要定義 $f(x)=1+2x+3x^2$ ，作法為

```
f=: 1 2 3 & p.
```

定義的函數為新的動詞，會將其右引數的數字用來計算此多項式的函數值。

```
f 1 2 3 4
6 17 34 57
```

以下定義另外一個函數 $g(x)=3+2x$

```
g=: 3 2 & p.
g 1 2 3 4
5 7 9 11
```

兩個多項式的運算可以用其動詞直接作。例如多項式的乘法 $f(x)*g(x)=3+8x+13x^2+6x^3$ ，直接定義為 $f*g$ ，故

```
(f*g) 1 2 3 4
30 119 306 627
```

合成函數的定義為 $f(g(x))=1+2(3+2x)+3(3+2x)^2=34+40x+12x^2$ ，而 J 以「@」來合成兩個函數，故

```
(f@g) 1 2 3 4
86 162 262 386
```

除了以數字驗算，有沒有辦法確知合成動詞得到的的確是合成函數，也就是說，能不能確知函數形式？「u t. x」可以得到 u 函數泰勒展式第 x 項的係數，故要得到合成函數前五項係數的指令為：

```
(f@g) t. i.5
34 40 12 0 0
```

這的確是合成函數 $34+40x+12x^2$ 的係數。

多項式指令 p.還有許多其他的用法，例如

```
p. 6 _9 3
```

```
+-+---+
```

```
|3|2 |1|
```

```
+-+---+
```

得到的為以根為中心的乘積，亦即， $6-9x+3x^2=3(x-2)(x-1)$ 。前例的結果分兩部分，第一部份為係數，第二部分為多項式的根。任何複雜的多項式，都可以很快的解出根來。另外一個例子：

```
p. 1 _1 3 _5
```

```
+-+-----+
```

```
|_5|0.712947 _0.0564737j|0.526627 _0.0564737j|_0.526627|
```

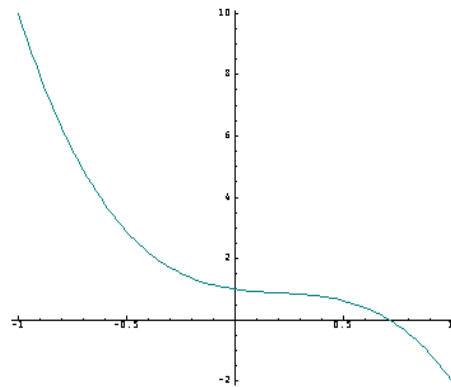
```
+-+-----+
```

此三次多項式有三個根，一個實根，兩個複數根。以圖形來看

```
load 'plot'
```

```
h=: 1 _1 3 _5 & p.
```

```
plot (;h)steps _1 1 100
```



平均數

平均數是 J 語法三叉(fork)應用的最佳範例。

```
Sum =. +/  
Average =. Sum % #  
c =. 1 2 3 4  
Average c  
2.5
```

前兩行定義代動詞，第三行定義代名詞，第四行以代動詞「Average」處理代名詞「c」。Average c 可以直接寫作

```
(+ / % #) 1 2 3 4
```

其中「(+ / % #)」為加總「+ /」、除以「%」、計數「#」三個動詞的串連，這三個動詞串在一塊稱之為三叉，三叉動詞若為雙邊，其意義如下：

$x (f g h) y$ 代表 $(x f y) g (x h y)$ ，

若為單邊，其意義如下：

$(f g h) y$ 代表 $(f y) g (h y)$ 。

因此(+ / % #) 1 2 3 4 相當於(+ / 1 2 3 4) % (# 1 2 3 4)，亦即對於給定的數列 1 2 3 4，加總後再除以其項數。

三叉(LT OR EQ)的意義可以由下述敘述間的相關性來理解：

$x (LT OR EQ) y$
 $(x LT y) OR (x EQ y)$ 。

黃金率

黃金率是西方傳統認為最適宜人眼的圖畫寬與高的比例，一般以 τ 表之。黃金率減 1 的倒數恰好為其本身，略微計算得知 $\tau = (1 + \sqrt{5})/2$ 。以 J 計算，

```
-:1+%:5  
1.6180339887498949
```

其中「-:」為對半，而「%:」為根號，因此上式可以想成「對半 1+根號 5」。黃金率也可以表成

$$\tau = 1 + \frac{1}{1 + \frac{1}{1 + \dots}}$$

式中 τ 減 1 的倒數仍為 τ 。上述定義為無限多項倒數相加，若以有限項來求近似值，例如 4 項，則可寫成

$$\tau = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1}}}$$

為((((1 的倒數)加 1)的倒數)加 1)的倒數)加 1。以 J 計算的式子為

```
(+%) / 1 1 1 1
1.6666666666666665
```

動詞串「(+%)」為「加倒」，兩動詞串在一塊稱為雙鉤(hook)動詞，「(+%) /」將加倒雙鉤動詞由右到左依序加諸於陣列之各個元素，因此對 1 構成的數列連續加倒，就成為黃金率的近似值，數列愈長近似值愈接近真值。長度為 m 元素為 n 的數列以 J 表示為 m\$n，因此前式可表為(+%) / 39\$1。當數列長度為 39 時，

```
(+%) / 39$1
1.6180339887498949
```

計算出的近似值與在精確位數 20 之下的真值相同。

雙鉤動詞必須以小刮號圍繞，因為若不用刮號，動詞將由右方先執行，意義完全不同。例如

```
+% / 1 1 1 1
1
```

先執行連續除「%/1 1 1 1」得到 1，再由「+1」取 1 的共軛數，結果仍然為 1。

除了以刮號界定雙鉤動詞外，也可以代動詞定義雙鉤動詞。例如

```
ar = .+%
ar / 1 1 1 1
1.6666666666666665
```

結語

讀到這裡，相信讀者已經可以領略 J 的強大功能，也會有動機進一步學習。J 系統已經有相當不錯的學習規劃，進入 J 系統後，由選單 Help->Help 呼叫 J 的輔助系統(J Help System)，為超文體(HTML)寫成，有鍊結到一些相當有用的參考文章。初學者可以先閱讀 Eric Iverson 的 J Primer，或者 Roger Stokes 寫的 Learning J，很快就可以應用自如；進階者可以由 J Phrases 或者 J 系統提供的文稿檔學得許多技巧。J Dictionary 有所有系統單字的定義，若有人願意發揮國中時期背英文字典的精神，是一個迅速入門的途徑。若要查詢某單字的用法，也有依字母順序排列的索引可以檢索。加入 [J Forum](#) 也是個不錯的考慮，一些高手常在上面替人解惑，也可由其中瞭解 J 的最新發展。

有這麼多資源，學習 J 絕對容易，而其強大的功能絕對值得花時間去學習，當然，邊用邊學用遠是成功學習的不二法門。

參考文獻

Dickey, Leroy J. (1992) "What is J? An introduction," University of Waterloo.
Hui, Roger K.W. and Kenneth E. Iverson, (2001) J Dictionary, Jsoftware Inc, 5/3.
Iverson Eric, J Primer, (2001) Jsoftware Inc, 3/29.
Burke, Chris, Roger K. W. Hui, Kenneth E. Iverson, Eugene E. McDonnell and
Donald B. McIntyre, (2001) J Phrases, Jsoftware Inc, 3/29.