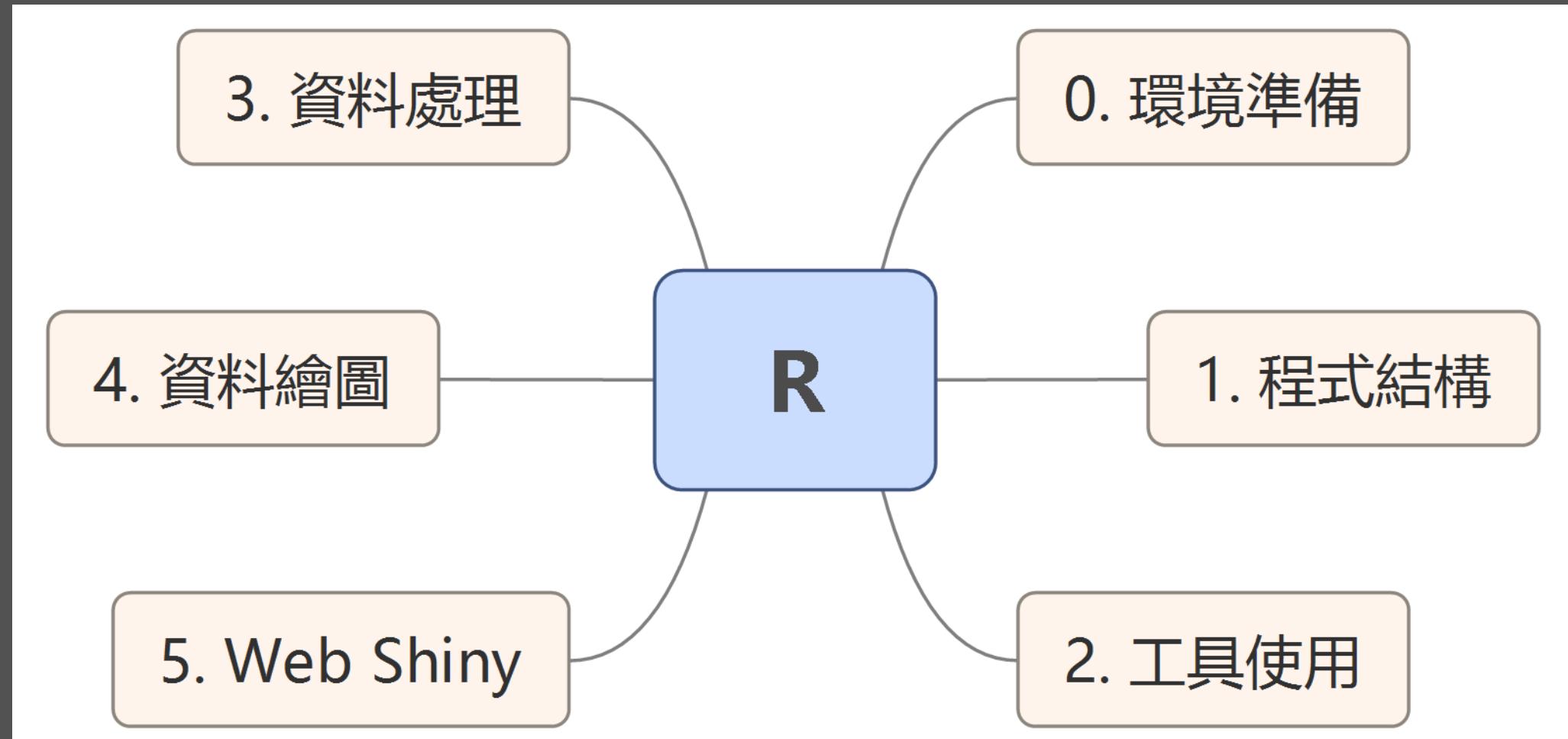


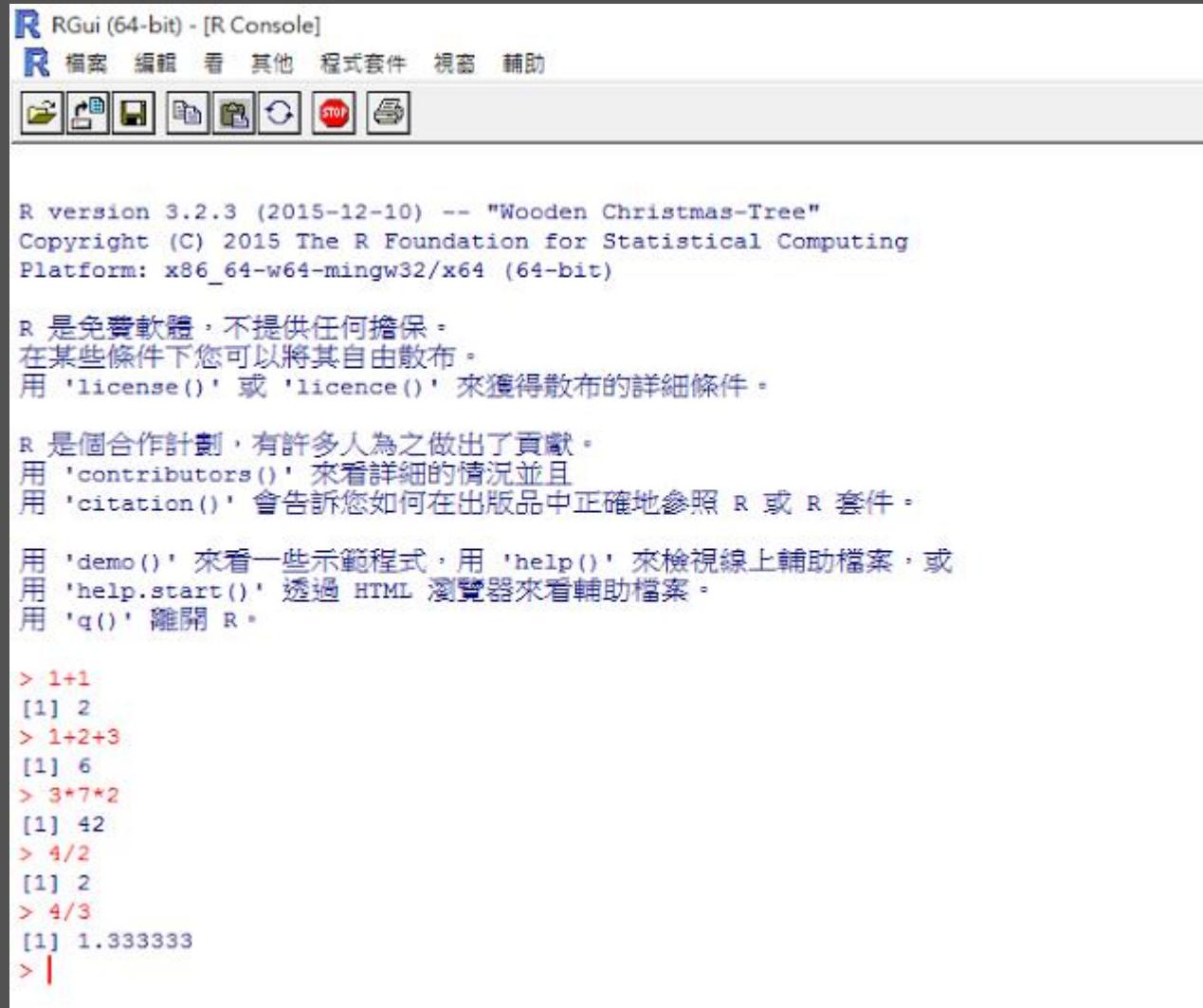
# R 程式結構

# 目的

在此說明 R 的基本資料型態與程式處理。



# 加、減、乘、除



R Gui (64-bit) - [R Console]

R 檔案 編輯 看 其他 程式套件 視窗 輔助

STOP

```
R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R 是免費軟體，不提供任何擔保。
在某些條件下您可以將其自由散布。
用 'license()' 或 'licence()' 來獲得散布的詳細條件。

R 是個合作計劃，有許多人為之做出了貢獻。
用 'contributors()' 來看詳細的情況並且
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。

用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。
用 'q()' 離開 R。


> 1+1
[1] 2
> 1+2+3
[1] 6
> 3*7*2
[1] 42
> 4/2
[1] 2
> 4/3
[1] 1.333333
> |
```

```
> 1 + 2 + 3
> 3 * 7 * 2
> 4/2
> 4/3
```

# 基本運算順序

R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"  
Copyright (C) 2015 The R Foundation for Statistical Computing  
Platform: x86\_64-w64-mingw32/x64 (64-bit)

R 是免費軟體，不提供任何擔保。  
在某些條件下您可以將其自由散布。  
用 'license()' 或 'licence()' 來獲得散布的詳細條件。

R 是個合作計劃，有許多人為之做出了貢獻。  
用 'contributors()' 來看詳細的情況並且  
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。

用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或  
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。  
用 'q()' 離開 R。

```
> 4*6+5
[1] 29
> (4*6)+5
[1] 29
> 4*(6+5)
[1] 44
> |
```

括號 -> 指數 -> 乘法 -> 除法 -> 加減

```
> 4 * 6 + 5
> (4 * 6) + 5
> 4 * (6 + 5)
```

# 指派 ( $<-$ ) & ( $=$ )

The screenshot shows the R Console window with the following text:

```
R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R 是免費軟體，不提供任何擔保。
在某些條件下您可以將其自由散布。
用 'license()' 或 'licence()' 來獲得散布的詳細條件。

R 是個合作計劃，有許多人為之做出了貢獻。
用 'contributors()' 來看詳細的情況並且
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。

用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。
用 'q()' 離開 R 。

> x <- 2
> x
[1] 2
> y = 5
> y
[1] 5
> |
```

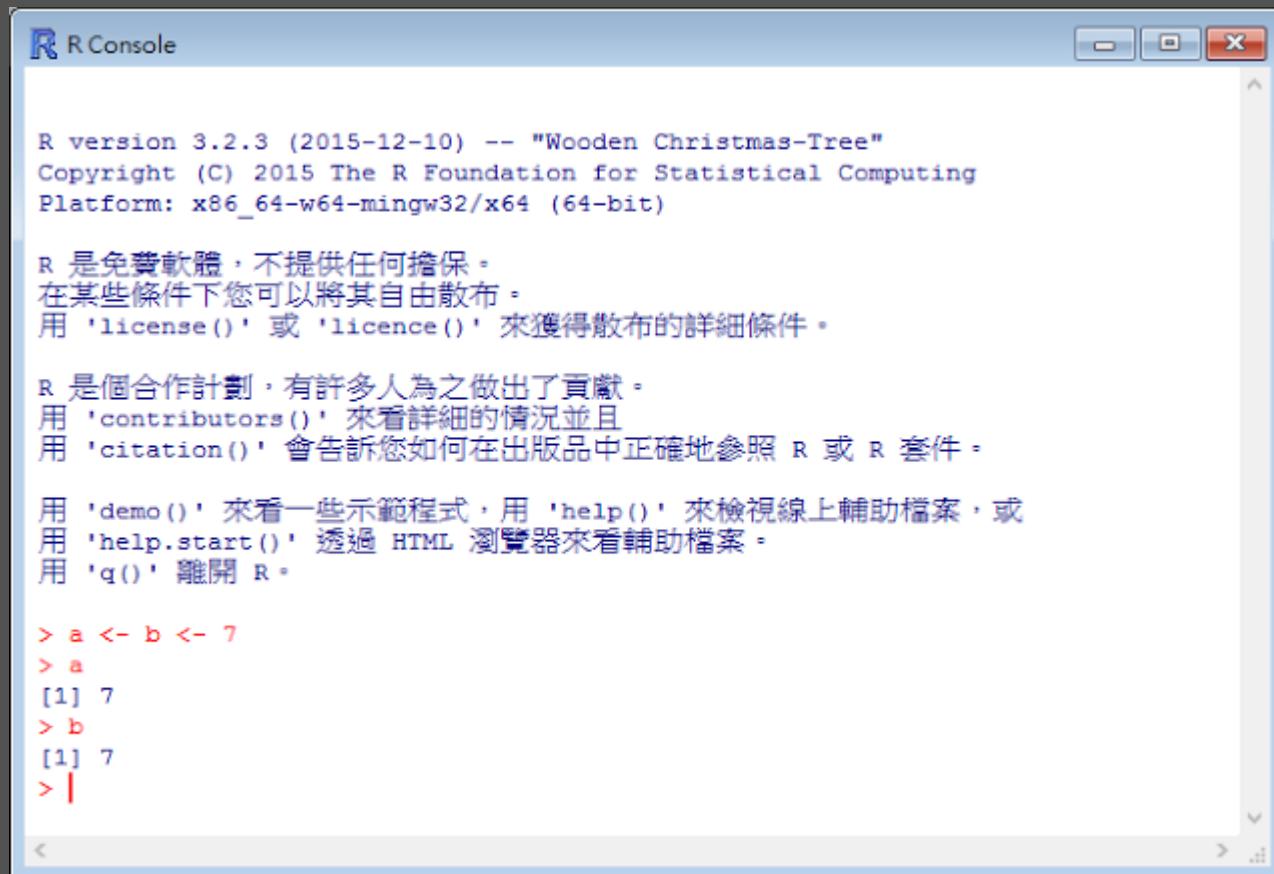
```
> x <- 2
```

```
> x
```

```
> y = 5
```

```
> y
```

# 連續指派



R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"  
Copyright (C) 2015 The R Foundation for Statistical Computing  
Platform: x86\_64-w64-mingw32/x64 (64-bit)

R 是免費軟體，不提供任何擔保。  
在某些條件下您可以將其自由散布。  
用 'license()' 或 'licence()' 來獲得散布的詳細條件。

R 是個合作計劃，有許多人為之做出了貢獻。  
用 'contributors()' 來看詳細的情況並且  
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。

用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或  
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。  
用 'q()' 離開 R。

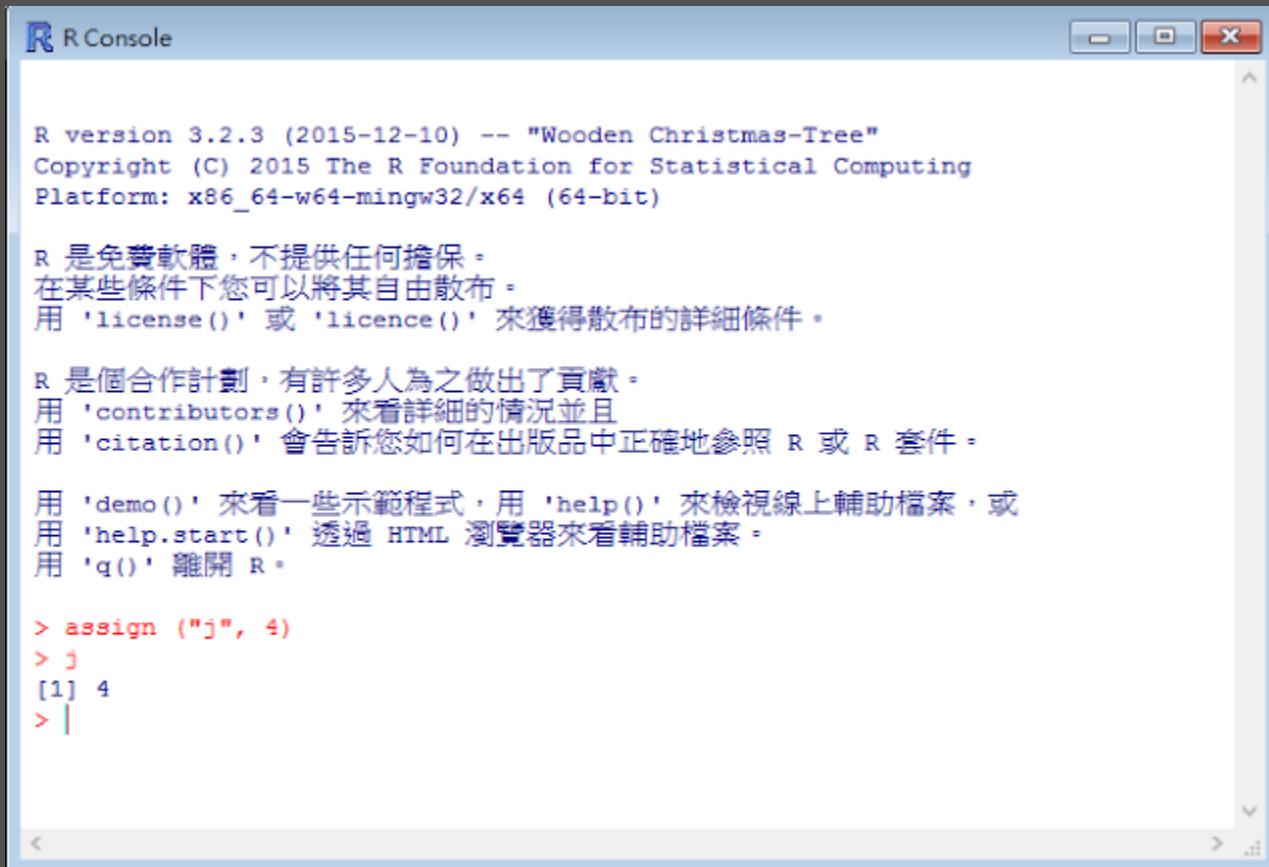
```
> a <- b <- 7
> a
[1] 7
> b
[1] 7
> |
```

> a <- b <- 7

> a

> b

# Assign 紿值



R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"  
Copyright (C) 2015 The R Foundation for Statistical Computing  
Platform: x86\_64-w64-mingw32/x64 (64-bit)

R 是免費軟體，不提供任何擔保。  
在某些條件下您可以將其自由散布。  
用 'license()' 或 'licence()' 來獲得散布的詳細條件。

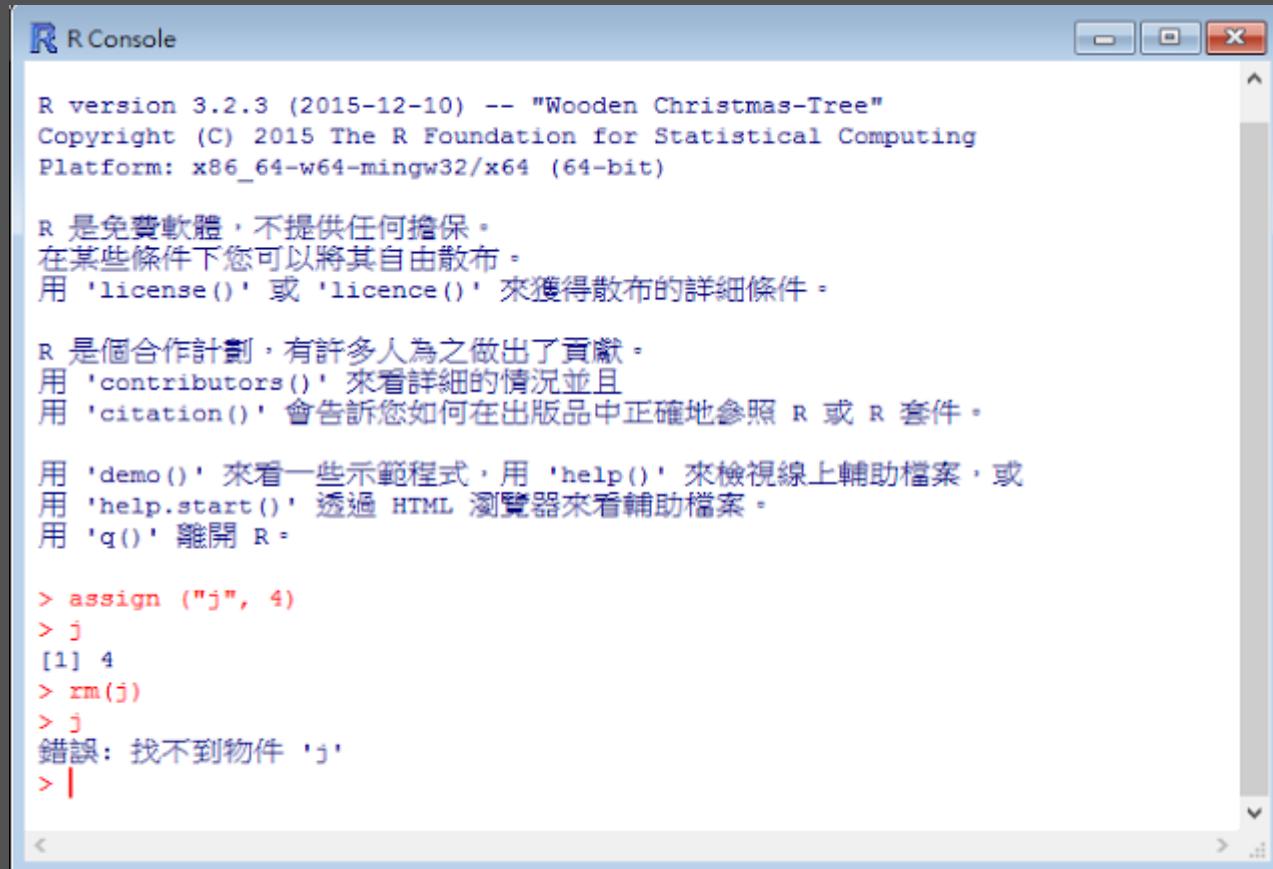
R 是個合作計劃，有許多人為之做出了貢獻。  
用 'contributors()' 來看詳細的情況並且  
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。

用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或  
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。  
用 'q()' 離開 R。

```
> assign ("j", 4)
> j
[1] 4
> |
```

```
> assign ("j", 4)
> j
```

# 移除物件



R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"  
Copyright (C) 2015 The R Foundation for Statistical Computing  
Platform: x86\_64-w64-mingw32/x64 (64-bit)

R 是免費軟體，不提供任何擔保。  
在某些條件下您可以將其自由散布。  
用 'license()' 或 'licence()' 來獲得散布的詳細條件。

R 是個合作計劃，有許多人為之做出了貢獻。  
用 'contributors()' 來看詳細的情況並且  
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。

用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或  
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。  
用 'q()' 離開 R：

```
> assign ("j", 4)
> j
[1] 4
> rm(j)
> j
錯誤: 找不到物件 'j'
> |
```

```
> rm(j)
> j
```

# 命名規則

The screenshot shows the R Console window with the title 'R Console'. The window displays the R startup message in Chinese, followed by a series of R commands demonstrating variable naming:

```
R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R 是免費軟體，不提供任何擔保。
在某些條件下您可以將其自由散布。
用 'license()' 或 'licence()' 來獲得散布的詳細條件。

R 是個合作計劃，有許多人為之做出了貢獻。
用 'contributors()' 來看詳細的情況並且
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。

用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。
用 'q()' 離開 R。

> theVariable <- 17
> theVariable
[1] 17
> THEVARIABLE
錯誤: 找不到物件 'THEVARIABLE'
> |
```

> theVariable <- 17  
> theVariable

> THEVARIABLE

# 資料型別

R Console

Copyright (C) 2015 The R Foundation for Statistical Computing  
Platform: x86\_64-w64-mingw32/x64 (64-bit)

R 是免費軟體，不提供任何擔保。  
在某些條件下您可以將其自由散布。  
用 'license()' 或 'licence()' 來獲得散布的詳細條件。

R 是個合作計劃，有許多人為之做出了貢獻。  
用 'contributors()' 來看詳細的情況並且  
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。

用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或  
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。  
用 'q()' 離開 R。

```
> x <- 0
>
> class(x)
[1] "numeric"
>
> is.numeric(x)
[1] TRUE
>
> i <- 5L
> i
[1] 5
>
> is.integer(i)
[1] TRUE
>
> is.numeric(i)
[1] TRUE
> |
```

在此列出常見的幾種型別

數值 - numeric

字元、字串 - character

時間 - Date/POSIXct

邏輯 - logical (TRUE/FALSE)

```
> x <- 0
```

```
> class(x)
```

```
> is.numeric(x)
```

```
> j <- 5L
```

```
> j
```

```
> is.integer(i)
```

```
> is.numeric(i)
```

# 數值 & 自動轉換

The screenshot shows the R Console window with the following text:

用 'contributors()' 來看詳細的情況並且  
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。

用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或  
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。  
用 'q()' 離開 R。

```
> class(4L)
[1] "integer"
> class(2.8)
[1] "numeric"
> 4L * 2.8
[1] 11.2
> class(4L * 2.8)
[1] "numeric"
>
> class(5L)
[1] "integer"
> class(2L)
[1] "integer"
> 5L/2L
[1] 2.5
> class(5L/2L)
[1] "numeric"
> |
```

注意!! R 處理時會將 integer 自動轉換成 numeric，尤其在相乘相除的時候就會發生。

```
> class(4L)
> class(2.8)
> 4L * 2.8
> class(4L * 2.8)
>
> class(5L)
> class(2L)
> 5L/2L
> class(5L/2L)
```

1. `class(4L * 2.8)`, 整數乘小數，資料型別變成 numeric

2. `class(5L/2L)`, 整數相除，資料型別變成 numeric

# 文字 & 自動轉換

R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"  
Copyright (C) 2015 The R Foundation for Statistical Computing  
Platform: x86\_64-w64-mingw32/x64 (64-bit)

R 是免費軟體，不提供任何擔保。  
在某些條件下您可以將其自由散布。  
用 'license()' 或 'licence()' 來獲得散布的詳細條件。

R 是個合作計劃，有許多人為之做出了貢獻。  
用 'contributors()' 來看詳細的情況並且  
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。

用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或  
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。  
用 'q()' 離開 R。

```
> x <- "data"
> x
[1] "data"
>
> y <- factor("data")
> y
[1] data
Levels: data
> nchar(x)
[1] 4
> nchar("hello")
[1] 5
> nchar(3)
[1] 1
> nchar(52)
[1] 2
> nchar(y)
Error in nchar(y) : 'nchar()' 需要字元向量
> |
```

1. character

2. factor

\* 要注意的是文字轉為數值時，若非阿拉伯數字的字串，R 會給 NA。

\* Factor 這個型別在也可以用於其他類型的資料，而在之後在處理繪圖的 X 軸與 Y 軸的標示非常好用。

```
> x <- "data"
```

```
> x
```

```
>
```

```
> y <- factor("data")
```

```
> y
```

其大小寫字母在character有所區別，在此可用 nchar() 函數，並利用前面指定的 x 和 y。

注意!! y 為 factor 並不適用 nchar() 的函數。

```
> nchar(x)
```

```
> nchar("hello")
```

```
> nchar(3)
```

```
> nchar(452)
```

```
> nchar(y)
```

# 時間 - Date/POSIXct

分為判斷天數和秒數

1. Date -> 存日期
2. POSIXct -> 存日期和時間

The screenshot shows an R console window titled "R Console". The code entered is:

```
> date1 <- as.Date("2012-06-28")
> date1
[1] "2012-06-28"
> class(date1)
[1] "Date"
> as.numeric(date1)
[1] 15519
> date2 <- as.POSIXct("2012-06-28 17:42")
> date2
[1] "2012-06-28 17:42:00 CST"
> class(date2)
[1] "POSIXct" "POSIXt"
> as.numeric(date2)
[1] 1340876520
>
> class(date1)
[1] "Date"
> class(as.numeric(date1))
[1] "numeric"
> |
```

\*\* 兩個物件都代表自 1970/1/1 算起的天數和秒數

```
> date1 <- as.Date("2012-06-28")
> date1
> class(date1)
> as.numeric(date1)
> date2 <- as.POSIXct("2012-06-28 17:42")
> date2
> class(date2)
> as.numeric(date2)
```

當然了，有兩個更方便的套件可以使用(lubridate、chron)。在這裡要注意一件事，使用 as.numeric 跟 as.Date，不只轉換物件格式，也會轉換其資料型別。

```
> as.numeric(date1)
> class(as.numeric(date1))
```

時間會是以自己系統為標準的中原標準時間。

# 邏輯資料

The screenshot shows an R console window titled "R Console". It contains the following R session:

```
> TRUE * 5
[1] 5
> FALSE * 5
[1] 0
> k <- TRUE
> class(k)
[1] "logical"
> is.logical(k)
[1] TRUE
> TRUE
[1] TRUE
> T
[1] TRUE
> class(T)
[1] "logical"
> T <- 7
> T
[1] 7
> class(T)
[1] "numeric"
> |
```

TRUE -> 1, FALSE -> 0

```
> TRUE * 5
> FALSE * 5
```

TRUE \* 5 結果會是5, FALSE \* 5 結果為0, 面對其他型別也可以檢測, 可以用 is.logical 函數

```
> k <- TRUE
> class(k)
> is.logical(k)
```

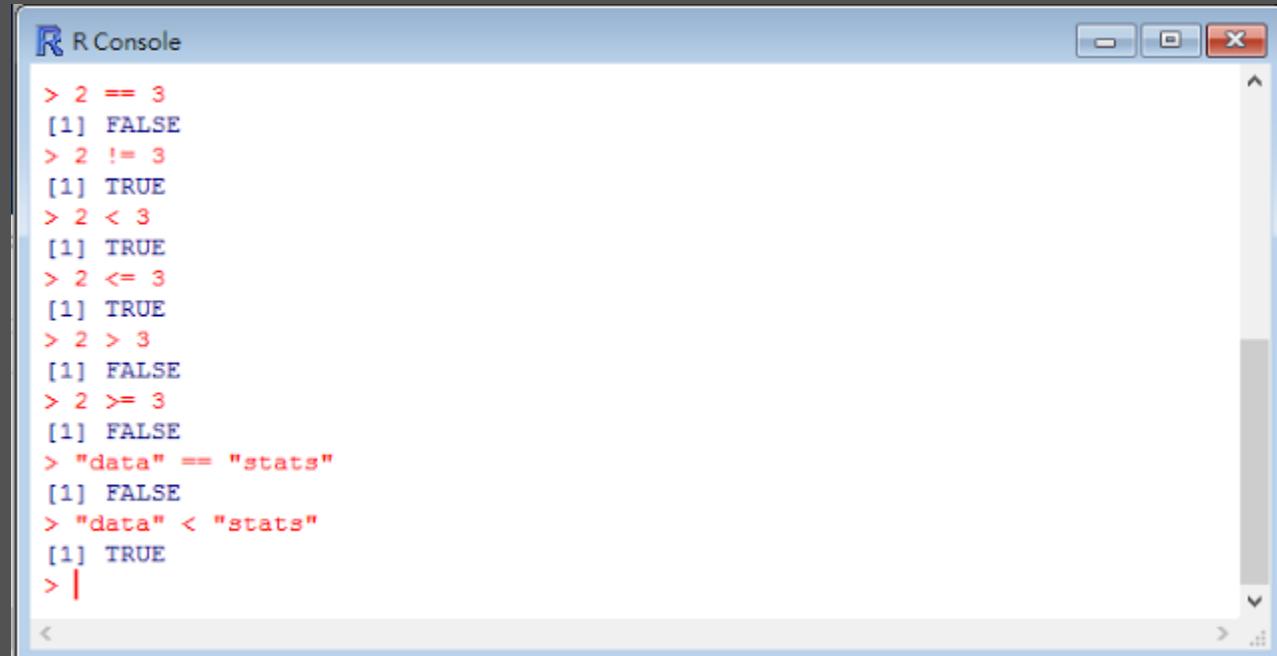
TRUE、FALSE 在 R 當中皆有縮寫(T & F)。

```
> TRUE
> T
> class(T)
> T <- 7
> T
> class(T)
```

注意!! 當指定一個值, 令一個變數 T , 很難分出來。

# 邏輯判斷

當然 `logicals` 可以比較兩個 “字元” 或是 “數字” 間的差異，可以使用 `==`(等於)、`!=`(不等於)、`<`(小於)、`<=`(小於等於)、`>`(大於)

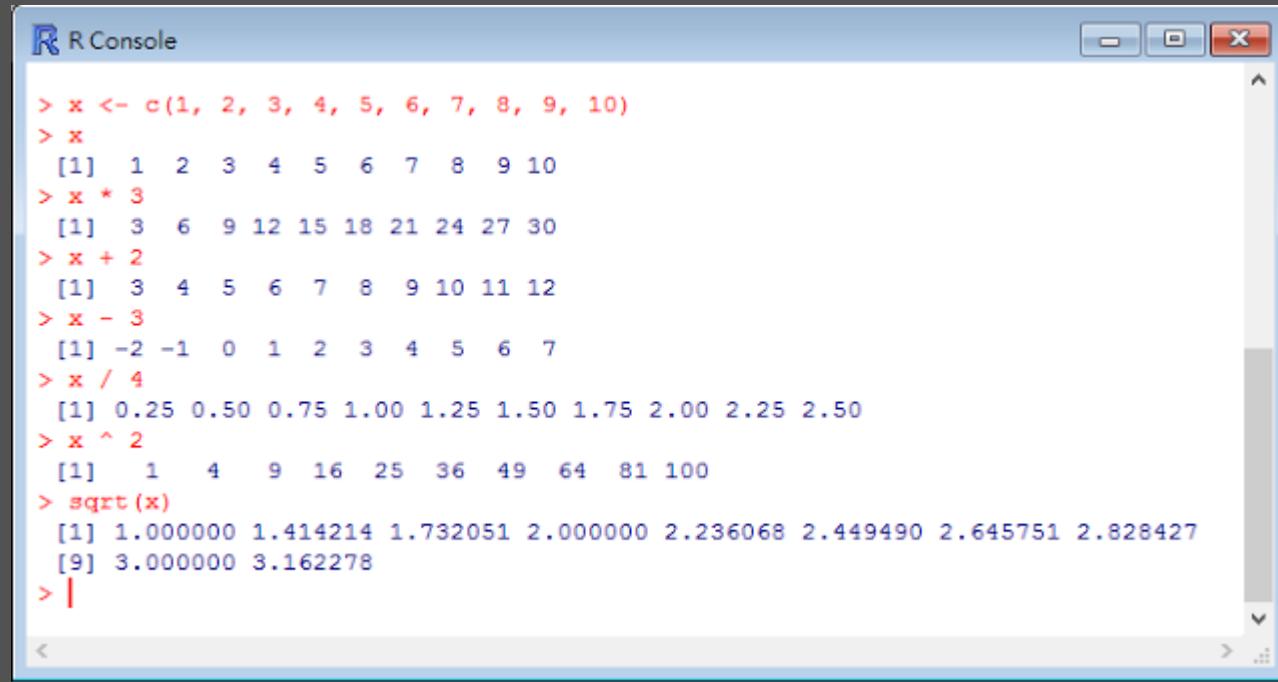


The screenshot shows an R console window titled "R Console". It displays a series of R commands and their results. The commands include comparisons between the numbers 2 and 3, and between the strings "data" and "stats". The results are all FALSE except for the comparisons involving the string "data".

```
R Console
> 2 == 3
[1] FALSE
> 2 != 3
[1] TRUE
> 2 < 3
[1] TRUE
> 2 <= 3
[1] TRUE
> 2 > 3
[1] FALSE
> 2 >= 3
[1] FALSE
> "data" == "stats"
[1] FALSE
> "data" < "stats"
[1] TRUE
> |
```

```
> 2 == 3
> 2 != 3
> 2 < 3
> 2 <= 3
> 2 > 3
> 2 >= 3
> "data" == "stats"
> "data" < "stats"
```

# vector



The screenshot shows the R Console window with the following session history:

```
> x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x * 3
[1] 3 6 9 12 15 18 21 24 27 30
> x + 2
[1] 3 4 5 6 7 8 9 10 11 12
> x - 3
[1] -2 -1 0 1 2 3 4 5 6 7
> x / 4
[1] 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00 2.25 2.50
> x ^ 2
[1] 1 4 9 16 25 36 49 64 81 100
> sqrt(x)
[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
[9] 3.000000 3.162278
> |
```

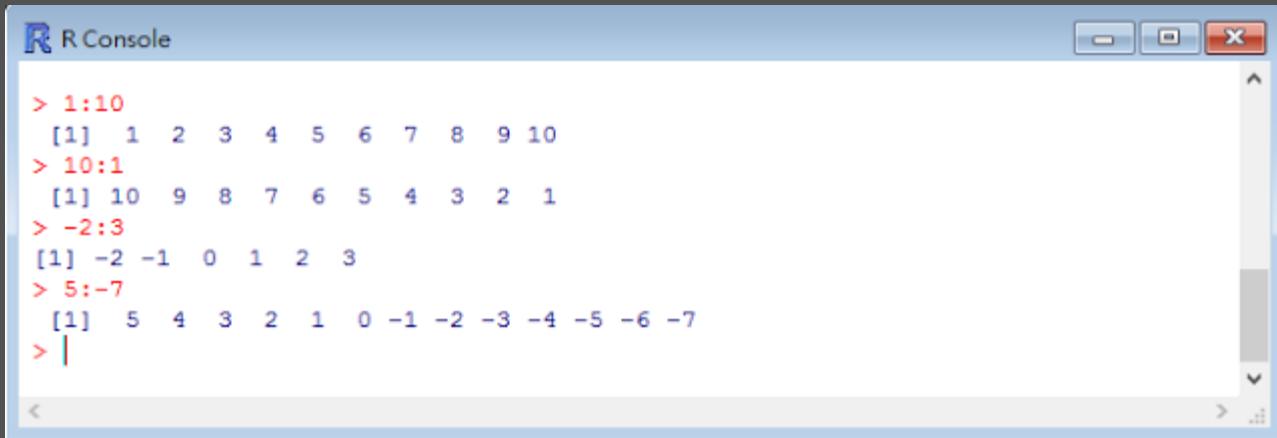
當然 vector 並不像 數學所謂的向量，但可以視為向量化的語言，所以當中元素的值可以被大量套用進行運算，R 當中的 vector 並無維度，也沒所謂的行向量、列向量，而且必須是同型別元素的集合，字串只能跟字串、數字只能跟數字。

注意!!! vector 是之後所有的資料型態的基礎，  
data.frame (資料框架)、List (列表)、Matrix (矩陣)、  
Array (陣列)。

```
> x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
> x
> x * 3
> x + 2
> x - 3
> x / 4
> x ^ 2
> sqrt(x)
```

如果同樣的動作用 java 來進行運算，用 for 迴圈。

# ( :) 運算子



R Console window showing R code and output:

```
> 1:10
[1] 1 2 3 4 5 6 7 8 9 10
> 10:1
[1] 10 9 8 7 6 5 4 3 2 1
> -2:3
[1] -2 -1 0 1 2 3
> 5:-7
[1] 5 4 3 2 1 0 -1 -2 -3 -4 -5 -6 -7
> |
```

1:10，在此就是指自動產生1~10之間的數值

> 1:10

> 10:1

> -2:3

> 5:-7

# length() & 運算

注意!!! 兩個向量一起運算時要注意，兩個向量彼此間的元素要對應，當然在這裡運算上(加、減、乘、除)與前面類似，然後我們可以利用 length() 來判斷其長度。

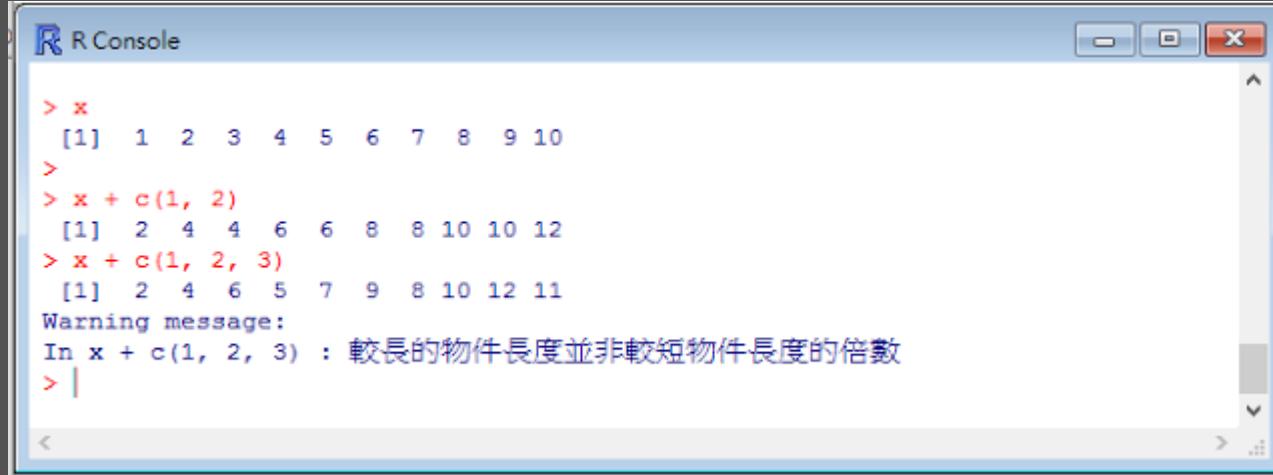
The screenshot shows an R console window with the following session history:

```
> x <- 1:10
> y <- -5:4
>
> x + y
[1] -4 -2  0  2  4  6  8 10 12 14
> x - y
[1]  6  6  6  6  6  6  6  6  6  6
> x * y
[1] -5 -8 -9 -8 -5  0  7 16 27 40
> x/y
[1] -0.2 -0.5 -1.0 -2.0 -5.0 Inf  7.0  4.0  3.0  2.5
> x^y
[1] 1.000000e+00 6.250000e-02 3.703704e-02 6.250000e-02 2.000000e-01
[6] 1.000000e+00 7.000000e+00 6.400000e+01 7.290000e+02 1.000000e+04
> length(x)
[1] 10
> length(y)
[1] 10
> length(x + y)
[1] 10
> |
```

```
> x <- 1:10
> y <- -5:4
> x + y
> x - y
> x * y
> x/y
> x^y
> length(x)
> length(y)
> length(x+y)
```

注意 !!!  $\text{length}(x+y)$  其長度應為 10，因為兩者長度一樣。

# length() 長度不對等



R Console window showing R code and its output:

```
> x
[1] 1 2 3 4 5 6 7 8 9 10
>
> x + c(1, 2)
[1] 2 4 4 6 6 8 8 10 10 12
> x + c(1, 2, 3)
[1] 2 4 6 5 7 9 8 10 12 11
Warning message:
In x + c(1, 2, 3) : 較長的物件長度並非較短物件長度的倍數
> |
```

短的會被循環的重複使用，在這個情況下長的向量若非短的向量倍數，會產生"錯誤訊息"。

```
> x + c(1, 2)
> x + c(1, 2, 3)
```

x 向量為 1 ~ 10 的值，則面對一行的  $c(1, 2)$  為其倍數，所以可以進行處理，但是 x 面對  $c(1, 2, 3)$  時因為 x 不為第二行的 c 向量的倍數，所以產生"錯誤訊息"。

# any( ) 、 all( )

The screenshot shows an R console window with the title 'R R Console'. The console displays the following R code:

```
> x <= 5
[1] TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE
> x > y
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
> x < y
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
>
> x <- 10:1
> y <- -4:5
> any(x < y)
[1] TRUE
> all(x < y)
[1] FALSE
>
> q <- c("Hockey", "Football", "Baseball", "Curling", "Rugby",
+        "Lacrosse", "Basketball", "Tennis", "Cricket", "Soccer")
> nchar(q)
[1] 6 8 8 7 5 8 10 6 7 6
> nchar(y)
[1] 2 2 2 2 1 1 1 1 1 1
>
> x[1]
[1] 10
> x[1:2]
[1] 10 9
> x[c(1, 4)]
[1] 10 7
> |
```

如同前面有判斷邏輯運算一樣，在向量這裡也能進行。

```
> x <= 5
```

```
> x > y
```

```
> x < y
```

1. any( ) -> 查看當中至少有一個比較結果為 TRUE

2. all( ) -> 查看所有比較結果是否都為 TRUE

使用 nchar() 觀察其character字元長度，中括號  
“[]”，在此可以查看向量(vector)中的某單一元素。

```
> x <- 10:1
```

```
> y <- -4:5
```

```
> any(x < y)
```

```
> all(x < y)
```

```
> q <- c("Hockey", "Football", "Baseball", "Curling",
"Rugby", "Lacrosse", "Basketball", "Tennis", "Cricket",
"Soccer")
```

```
> nchar(q)
```

```
> nchar(y)
```

```
> x[1]
```

```
> x[1:2]
```

```
> x[c(1, 4)]
```

# 向量中的元素命名

R 是免費軟體，不提供任何擔保。  
在某些條件下您可以將其自由散布。  
用 'license()' 或 'licence()' 來獲得散布的詳細條件。

R 是個合作計劃，有許多人為之做出了貢獻。  
用 'contributors()' 來看詳細的情況並且  
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。

用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或  
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。  
用 'q()' 離開 R。

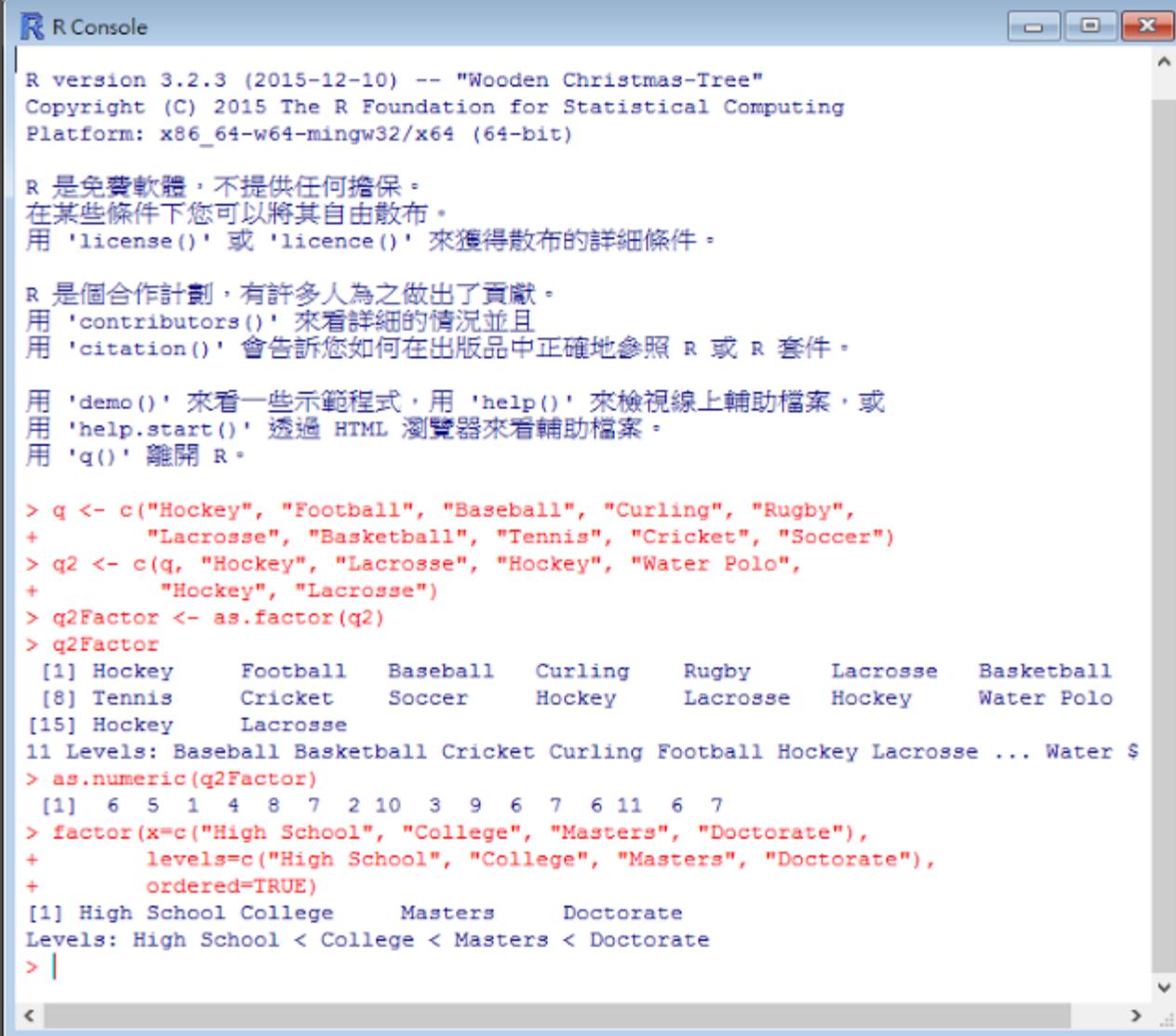
```
> c(One = "a", Two = "y", Last = "r")
One  Two Last
"a"  "y"  "r"
>
> w <- 1:3
>
> names(w) <- c("a", "b", "c")
> w
a b c
1 2 3
> |
```

對向量中的元素命名，此部分是不管建立向量前或者建立向量後皆可 !!

```
> c(One = "a", Two = "y", Last = "r")
>
> w <- 1:3
> names(w) <- c("a", "b", "c")
> w
```

# Factor Vector

前面實做的 q character 向量進行實作，在其多塞幾個重複的元素，並指定為 q2。然後我們使用 as.factor 將原本是字元向量的 q2 進行轉換成 factor 向量。



R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"  
Copyright (C) 2015 The R Foundation for Statistical Computing  
Platform: x86\_64-w64-mingw32/x64 (64-bit)

R 是免費軟體，不提供任何擔保。  
在某些條件下您可以將其自由散布。  
用 'license()' 或 'licence()' 來獲得散布的詳細條件。

R 是個合作計劃，有許多人為之做出了貢獻。  
用 'contributors()' 來看詳細的情況並且  
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。

用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或  
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。  
用 'q()' 離開 R。

```
> q <- c("Hockey", "Football", "Baseball", "Curling", "Rugby",
+       "Lacrosse", "Basketball", "Tennis", "Cricket", "Soccer")
> q2 <- c(q, "Hockey", "Lacrosse", "Hockey", "Water Polo",
+          "Hockey", "Lacrosse")
> q2Factor <- as.factor(q2)
> q2Factor
[1] Hockey   Football  Baseball Curling   Rugby    Lacrosse Basketball
[8] Tennis   Cricket   Soccer    Hockey   Lacrosse  Hockey   Water Polo
[15] Hockey  Lacrosse
11 Levels: Baseball Basketball Cricket Curling Football Hockey Lacrosse ... Water $
```

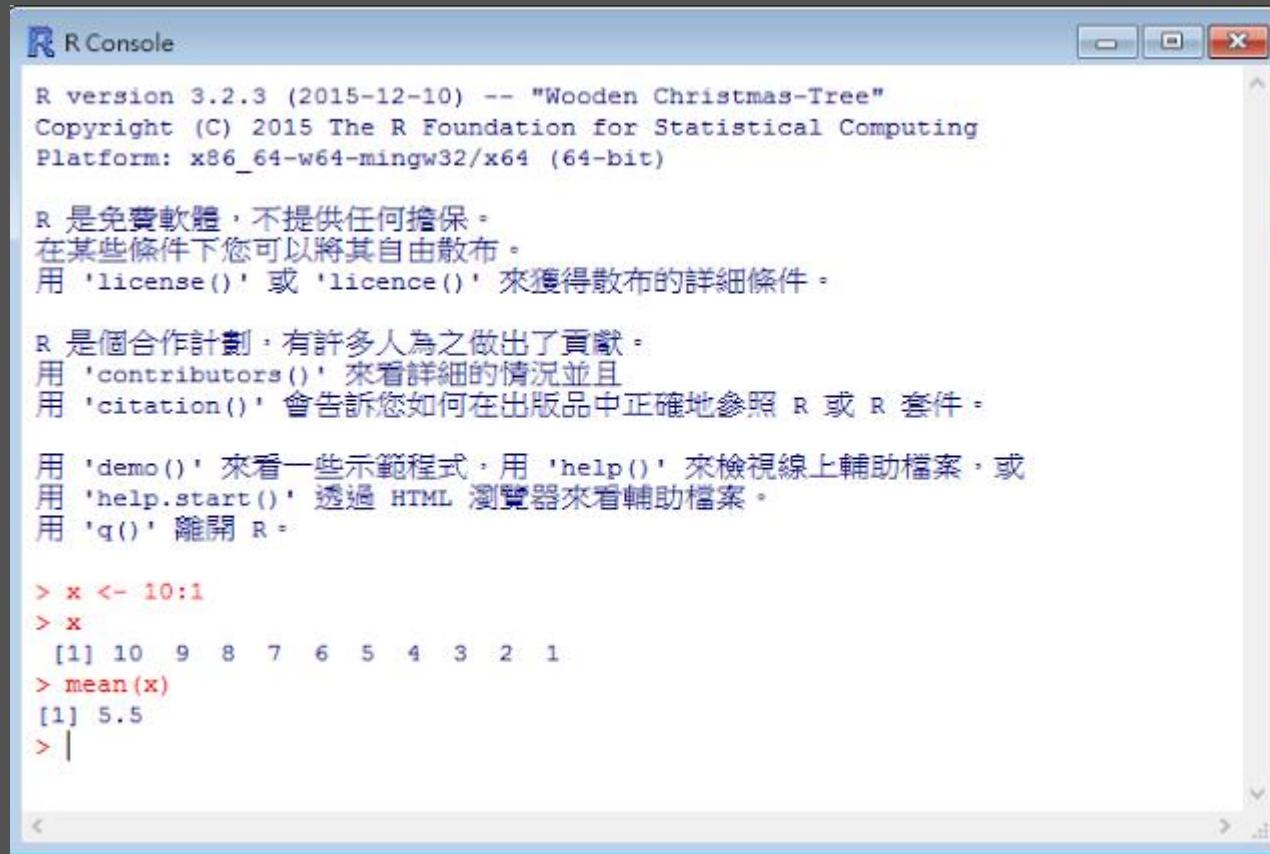
```
> as.numeric(q2Factor)
[1] 6 5 1 4 8 7 2 10 3 9 6 7 6 11 6 7
> factor(x=c("High School", "College", "Masters", "Doctorate"),
+         levels=c("High School", "College", "Masters", "Doctorate"),
+         ordered=TRUE)
[1] High School College Masters Doctorate
Levels: High School < College < Masters < Doctorate
> |
```

其 Factor 特性 會將不重複的元素，給予一個專屬的 integer (整數) 用字元來表示每個 integer，並且分等級 (levels)，可以使用 as.numeric 來觀察。

這個功能對其建立模型有很大的幫助，但是他只會存不重複的元素，好處在於可以大量減少變數容量。

```
> q <- c("Hockey", "Football", "Baseball", "Curling",
"Rugby", "Lacrosse", "Basketball", "Tennis", "Cricket",
"Soccer")
> q2 <- c(q, "Hockey", "Lacrosse", "Hockey", "Water Polo",
"Water Polo", "Hockey", "Lacrosse")
> q2Factor <- as.factor(q2)
> q2Factor
> as.numeric(q2Factor)
> factor(x=c("High School", "College", "Masters", "Doctorate"),
+         levels=c("High School", "College", "Masters", "Doctorate"),
+         ordered=TRUE)
[1] High School College Masters Doctorate
Levels: High School < College < Masters < Doctorate
> |
```

# 呼叫函數



R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"  
Copyright (C) 2015 The R Foundation for Statistical Computing  
Platform: x86\_64-w64-mingw32/x64 (64-bit)

R 是免費軟體，不提供任何擔保。  
在某些條件下您可以將其自由散布。  
用 'license()' 或 'licence()' 來獲得散布的詳細條件。

R 是個合作計劃，有許多人為之做出了貢獻。  
用 'contributors()' 來看詳細的情況並且  
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。

用 'demo()' 來看一些示範程式，用 'help()' 來檢視線上輔助檔案，或  
用 'help.start()' 透過 HTML 瀏覽器來看輔助檔案。  
用 'q()' 離開 R。

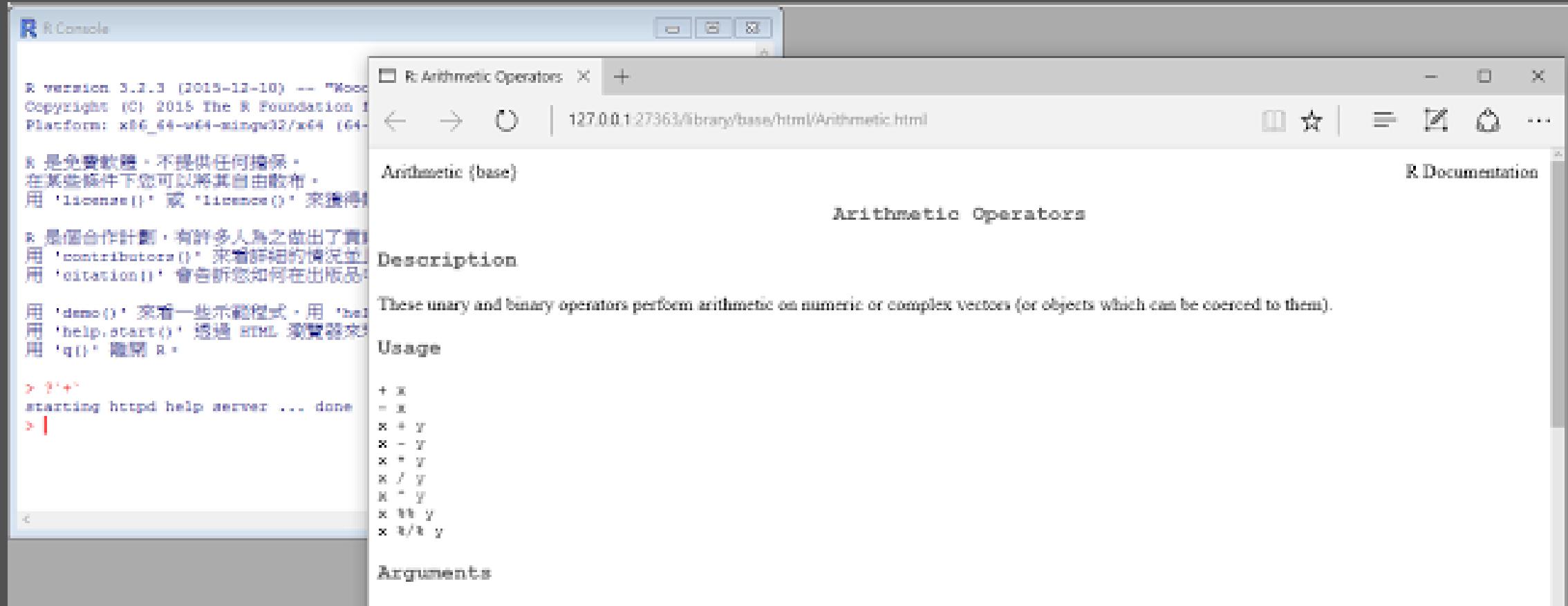
```
> x <- 10:1
> x
[1] 10  9  8  7  6  5  4  3  2  1
> mean(x)
[1] 5.5
> |
```

前面已經用過不少函數，當然了我們也使以自己定義變數去執行。

在這裡介紹計算平均數的函數，可以計算數字向量的平均值，其最基本的呼叫方式是將向量作為其引數!!

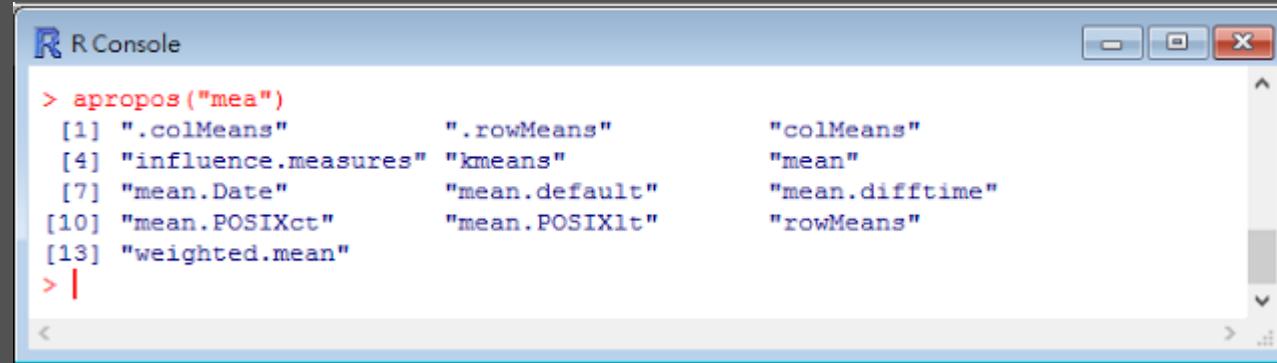
```
> x = 10:1
> mean(x)
```

# R 文件(?) & help()



```
> ?'+  
> ?'*  
> ?'=='
```

# tab & apropos()

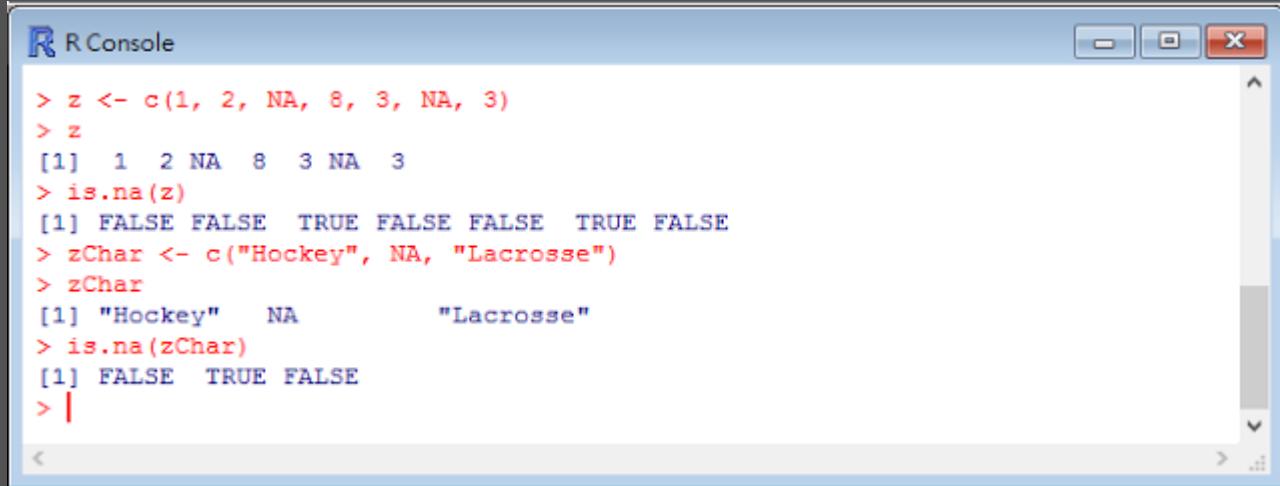


The screenshot shows an R console window titled "R Console". The command `> apropos("mea")` is entered, and the output is a vector of 13 elements:

```
> apropos("mea")
[1] ".colMeans"           ".rowMeans"          "colMeans"
[4] "influence.measures" "kmeans"            "mean"
[7] "mean.Date"           "mean.default"      "mean.difftime"
[10] "mean.POSIXct"        "mean.POSIXlt"      "rowMeans"
[13] "weighted.mean"
```

```
> apropos("mea")
```

# 遺失值與不存在



The screenshot shows the R Console window with the following session history:

```
> z <- c(1, 2, NA, 8, 3, NA, 3)
> z
[1] 1 2 NA 8 3 NA 3
> is.na(z)
[1] FALSE FALSE TRUE FALSE FALSE TRUE FALSE
> zChar <- c("Hockey", NA, "Lacrosse")
> zChar
[1] "Hockey"    NA          "Lacrosse"
> is.na(zChar)
[1] FALSE  TRUE FALSE
> |
```

R 分為兩種，雖然差不多但實際上並不相同。

1. NA
2. NULL

NA 為遺漏的資料，統計軟體大多數會用此值來記錄遺失值，但是對於向量(vector)來說，他被視為一種元素。

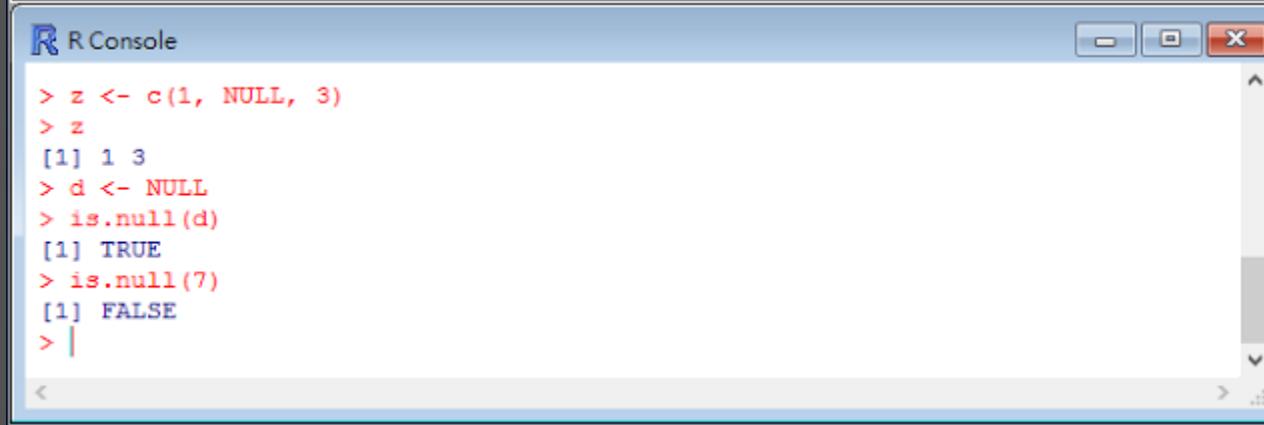
```
> z <- c(1, 2, NA, 8, 3, NA, 3)
> z
> is.na(z)
```

至於 `is.na()`，其函數可以讓你用邏輯的方式讓你觀察向量中的 NA 值。而在面對字元向量時，僅單純輸入‘N’ 和 ‘A’ 的字元即可放入字元向量中進行處理。

```
> zChar <- c("Hockey", NA, "Lacrosse")
> zChar
> is.na(zChar)
```

# 遺失值與不存在

NULL在此我們則是視為不存在(這在 JAVA與其他語言也是如此), 在其向量中也不會被視為元素而不能存在, 既不能存入向量中, 而且還會自消失在向量裡。



The screenshot shows an R console window titled "R Console". The code entered is:

```
> z <- c(1, NULL, 3)
> z
[1] 1 3
> d <- NULL
> is.null(d)
[1] TRUE
> is.null(7)
[1] FALSE
> |
```

要檢測其 NULL的函數為 `is.null()`

```
> z <- c(1, NULL, 3)
> z
> d <- NULL
> is.null(d)
> is.null(7)
```

# class() & mode() & str()

## 1. 模式 - mode

mode 跟 class 有些類似，他會顯示該物件的型態。

## 2. 屬性 - attributes & attr

在此建立矩陣，可以從 attributes 看到可以觀察的參數，當中可以用 “\$” 將參數下的資料叫出來，而且可以跟該參數的名稱，用相同的名稱的函數進行檢視，如果有 factor 型態的資料殼能看到 Levels 的顯示。data.frame 同樣也可以使用，當中會有 names、row.names、class。

## 3. 結構 - str

str 則會告知我們該物件的型態是什麼，數字的向量回傳 int 與長度，文字的資料回傳 chr 與向量的長度，而內建山貓資料(lynx)，則回傳是時間序列資料的訊息，iris 則回傳 data.frame，最後就是傳入一般的函數名稱則回傳該函數的部分函數資訊。

# class() & mode() & str()

```
tnm = 9:29  
mode(tnm)  
tch = as.character(tnm)  
mode(tch)  
tint = as.integer(tch)  
mode(tint)  
tlog = tint <= 20  
mode(tlog)
```

```
tat = matrix( 9:29, ncol = 3)  
tat  
attributes(tat)  
attr( tat, "dim")  
dim(tat)  
attributes(tat)$dim
```

```
tatvt = c( attributes(tat)$dim,  
         sum(attributes(tat)$dim),  
         mean(attributes(tat)$dim),  
         sd(attributes(tat)$dim))  
tatvt
```

```
> tnm = 9:29  
> mode(tnm)  
[1] "numeric"  
> tch = as.character(tnm)  
> mode(tch)  
[1] "character"  
> tint = as.integer(tch)  
> mode(tint)  
[1] "numeric"  
> tlog = tint <= 20  
> mode(tlog)  
[1] "logical"  
> |
```

```
> tat = matrix( 9: 29, ncol = 3)  
> tat  
     [,1] [,2] [,3]  
[1,]    9   16   23  
[2,]   10   17   24  
[3,]   11   18   25  
[4,]   12   19   26  
[5,]   13   20   27  
[6,]   14   21   28  
[7,]   15   22   29  
> attributes(tat)  
$dim  
[1] 7 3  
  
> attr( tat, "dim")  
[1] 7 3  
> dim(tat)  
[1] 7 3  
> attributes(tat)$dim  
[1] 7 3  
>  
> tatvt = c( attributes(tat)$dim,  
+           sum(attributes(tat)$dim),  
+           mean(attributes(tat)$dim),  
+           sd(attributes(tat)$dim)  
+         )  
> tatvt  
[1] 7.000000 3.000000 10.000000 5.000000 2.828427  
> |
```

# class() & mode() & str()

```
tatdf = data.frame(matrix( 9:29, ncol = 3))
```

```
tatdf
```

```
attributes(tatdf)
```

```
attr( tatdf, "names" )
```

```
names(tatdf)
```

```
str(tnm)
```

```
tletch = letters[6:16]
```

```
str(tletch)
```

```
str(lynx)
```

```
str(iris)
```

```
str(attr)
```

```
str(mode)
```

```
> tatdf = data.frame(matrix( 9:29, ncol = 3))
> tatdf
  X1 X2 X3
1  9 16 23
2 10 17 24
3 11 18 25
4 12 19 26
5 13 20 27
6 14 21 28
7 15 22 29

> attributes(tatdf)
$names
[1] "X1" "X2" "X3"

$row.names
[1] 1 2 3 4 5 6 7

$class
[1] "data.frame"

> attr( tatdf, "names" )
[1] "X1" "X2" "X3"
> names(tatdf)
[1] "X1" "X2" "X3"
> |
```

```
> str(tnm)
int [1:21] 9 10 11 12 13 14 15 16 17 18 ...
> tletch = letters[6:16]
> str(tletch)
chr [1:11] "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p"
> str(lynx)
Time-Series [1:114] from 1821 to 1934: 269 321 585 871 1475 ...
> str(iris)
'data.frame': 150 obs. of 5 variables:
$ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
$ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
$ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
$ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
$ Species     : Factor w/ 3 levels "setosa","versicolor",...: 1 1 ...
> str(attr)
function (x, which, exact = FALSE)
> str(mode)
function (x)
> |
```

# ls() & dir() & history()

```
> ls()
[1] "Satisfactory" "Sex"          "survey"        "survey.ex"
[5] "Times"
>
```

> dir()

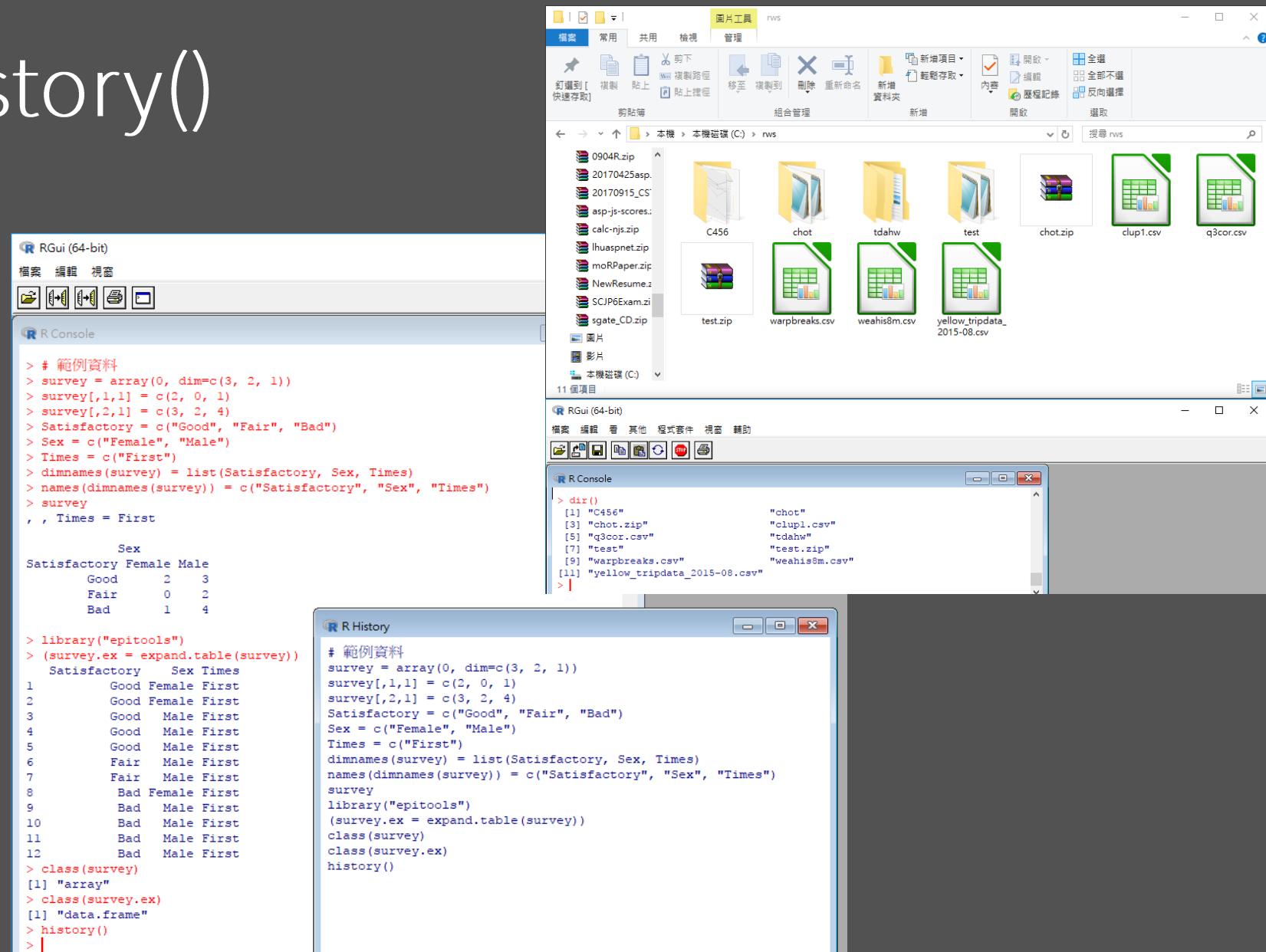
> ls()

> history()

dir() 查詢工作目錄下的檔案跟目錄。

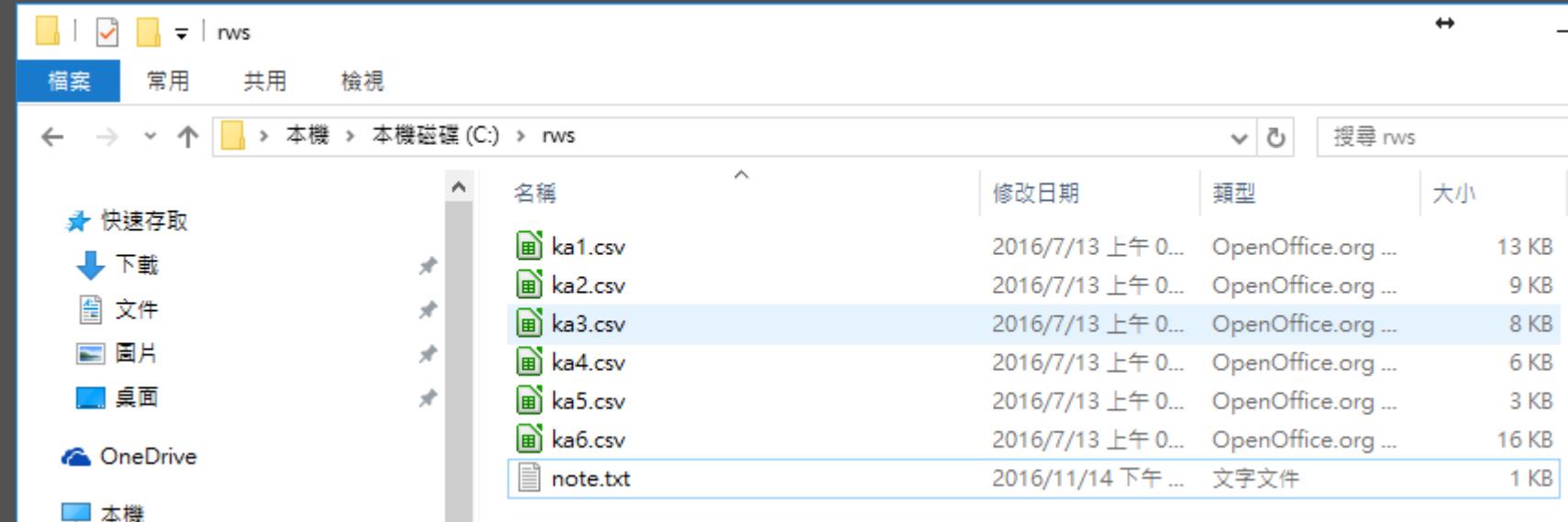
ls() 查詢 R 環境下的變數。

history() 查詢 R 執行過的指令。



而在 R GUI 會在 Windows R GUI 上跳出一個指令紀錄的視窗。

# file.exists()



```
> file.exists()
```

```
# R 查看目錄檔案
```

```
> getwd()
[1] "C:/rws"
> dir()
[1] "ka1.csv"   "ka2.csv"   "ka3.csv"   "ka4.csv"   "ka5.csv"   "ka6.csv"   "note.txt"
> a = "ka1.csv"
> b = "kan.csv"
> c = "note.txt"
> file.exists(a)
[1] TRUE
> file.exists(b)
[1] FALSE
> file.exists(c)
[1] TRUE
> |
```

# R 資料結構

常見"資料結構"

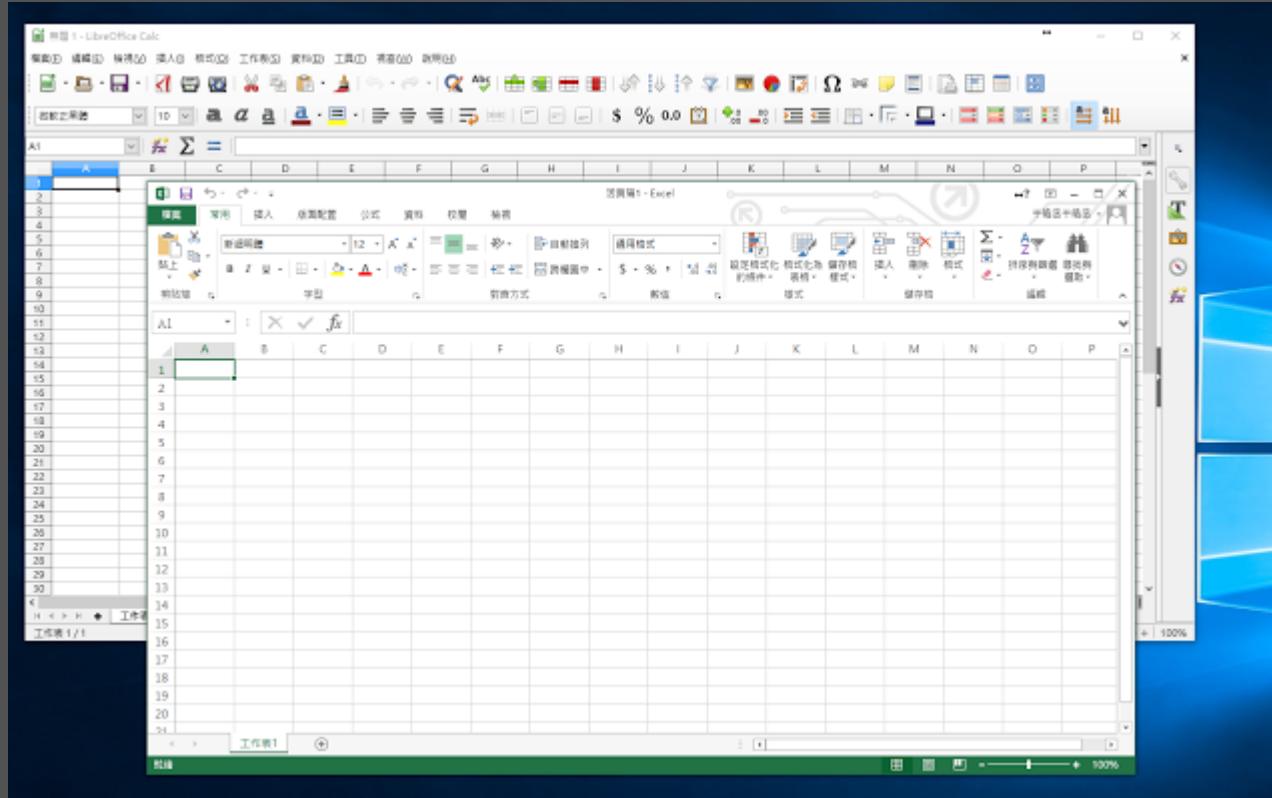
1. Data.frame - 資料框 (資料框架)

2. List - 列表

3. Matrix - 矩陣

4. Array - 陣列

# Data.frame



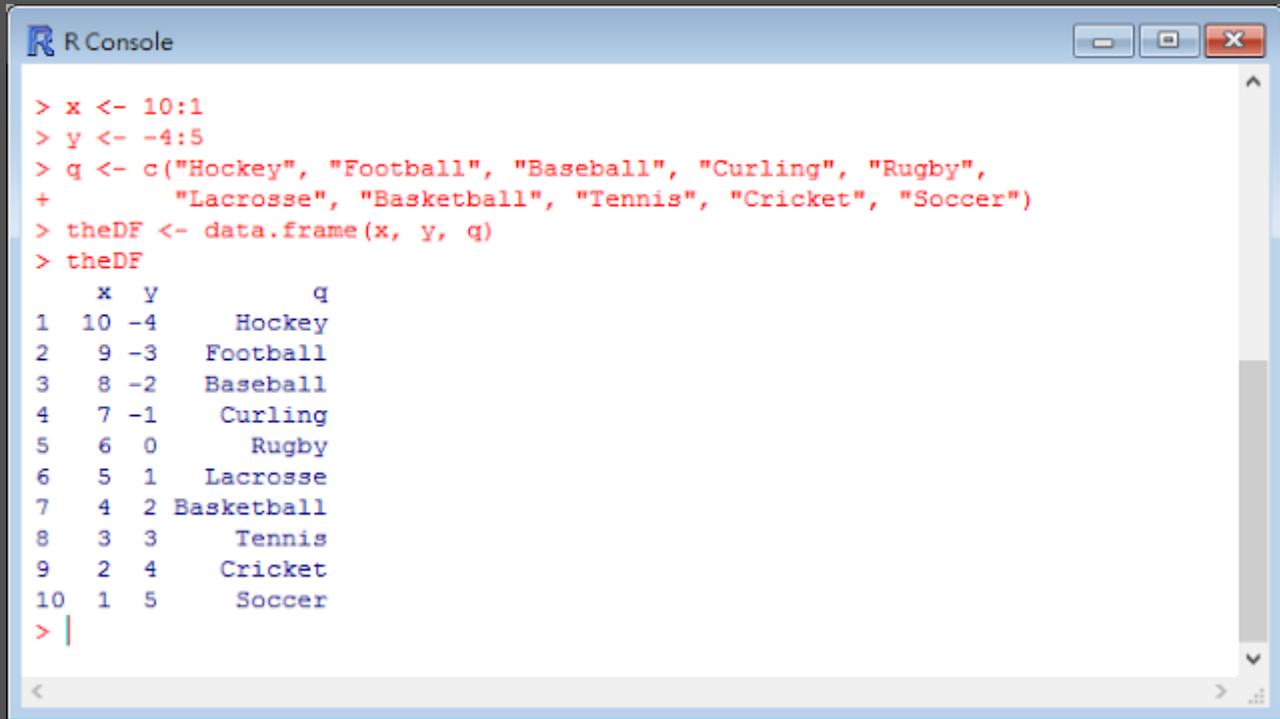
data.frame 屬於 R 特有的資料結構，因為有此結構使R相對比其他程式語言。

每一直行 -> 代表一變數

每一橫列 -> 代表一個值(該變數的蒐集資料)

每一直行代表一向量，那該行中的列就是該向量的元素，所以一張表(table)可以當作一群向量的合體。

# Data.frame



The screenshot shows the R Console window. The code entered is:

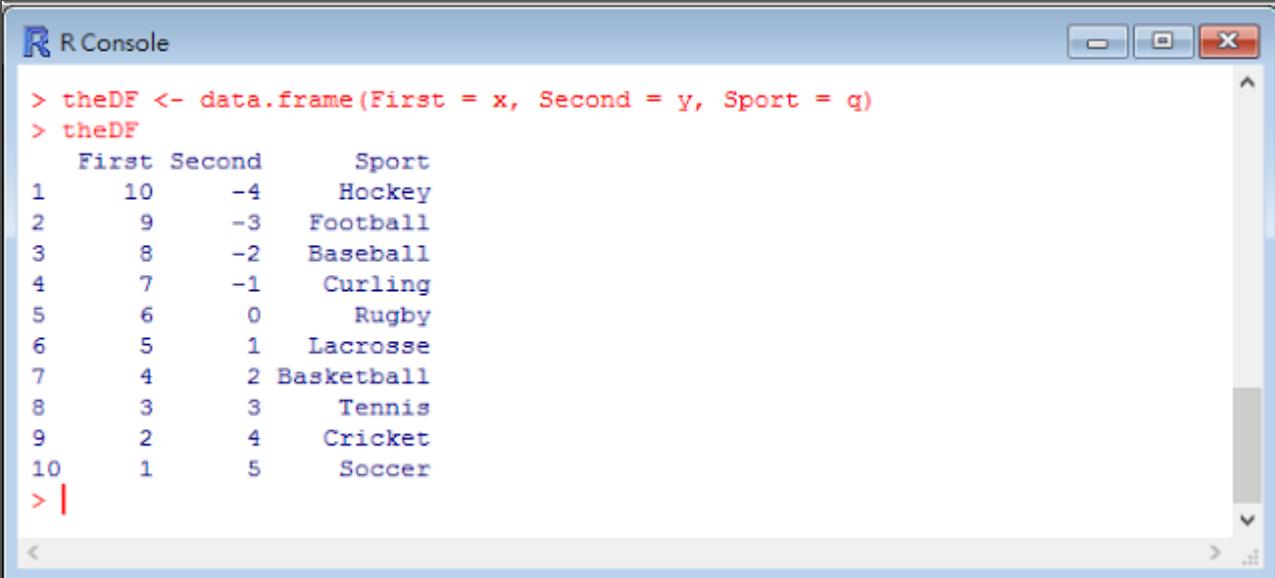
```
> x <- 10:1
> y <- -4:5
> q <- c("Hockey", "Football", "Baseball", "Curling",
+       "Rugby", "Lacrosse", "Basketball", "Tennis", "Cricket",
+       "Soccer")
> theDF <- data.frame(x, y, q)
> theDF
```

The resulting output is:

	x	y	q
1	10	-4	Hockey
2	9	-3	Football
3	8	-2	Baseball
4	7	-1	Curling
5	6	0	Rugby
6	5	1	Lacrosse
7	4	2	Basketball
8	3	3	Tennis
9	2	4	Cricket
10	1	5	Soccer

```
> x <- 10:1
> y <- -4:5
> q <- c("Hockey", "Football", "Baseball", "Curling",
+       "Rugby", "Lacrosse", "Basketball", "Tennis", "Cricket",
+       "Soccer")
> theDF <- data.frame(x, y, q)
> theDF
```

# Data.frame



The screenshot shows the R Console window with the following text:

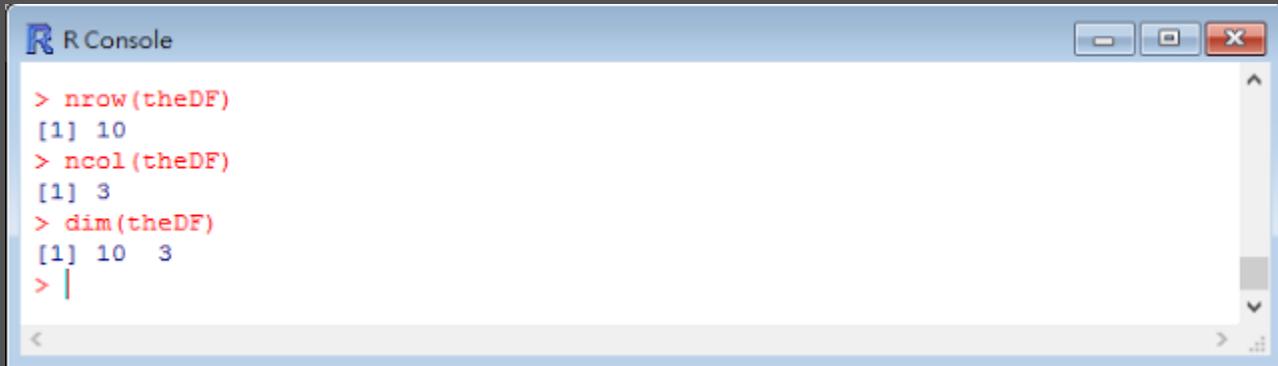
```
R Console
> theDF <- data.frame(First = x, Second = y, Sport = q)
> theDF
  First Second     Sport
1     10      -4    Hockey
2      9      -3   Football
3      8      -2  Baseball
4      7      -1   Curling
5      6       0    Rugby
6      5       1 Lacrosse
7      4       2 Basketball
8      3       3    Tennis
9      2       4   Cricket
10     1       5   Soccer
> |
```

theDF 是一個包含三個向量的  $10 \times 3$  的 data.frame(資料框架)，再來 theDF 即為變數名稱，最好在宣告資料框架時就必須同時命名好變數名稱!!接著我們再給 theDF 資料框架中的每個向量命名。

```
> theDF <- data.frame(First = x, Second = y, Sport = q)
> theDF
```

# Data.frame

data.frame(資料框架)是一個擁有很多屬性(attributes)的複雜物件(既然他能塞不同的向量)



The screenshot shows the R Console window with the following text:

```
R R Console
> nrow(theDF)
[1] 10
> ncol(theDF)
[1] 3
> dim(theDF)
[1] 10  3
> |
```

最常需要觀察其行列數量，一共有三種函數可以使用

1. nrow() -> 觀察列的數量
2. ncol() -> 觀察行的數量
3. dim() -> 同時看行列的數量

```
> nrow(theDF)
> ncol(theDF)
> dim(theDF)
```

# Data.frame

```
> names(theDF)
[1] "First"  "Second" "Sport"
> names(theDF)[3]
[1] "Sport"
> rownames(theDF)
[1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"
> rownames(theDF) <- c("One", "Two", "Three", "Four", "Five", "Six",
+                      "Seven", "Eight", "Nine", "Ten")
> rownames(theDF)
[1] "One"   "Two"   "Three" "Four"   "Five"   "Six"   "Seven" "Eight"   "Nine"
[10] "Ten"
> rownames(theDF) <- NULL
> rownames(theDF)
[1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"
```

再來則是觀察 資料框架的直行名稱(每一直行代表一向量)，可以使用 names() 函數，他會以字元、字串向量的方式輸出，再來後面用 []，可以觀察特定直行的名稱!!!

```
> names(theDF)
> names(theDF)[3]
```

反之我們也可以找出資料框的橫列名稱或者指派其橫列名稱，在此使用 rownames() 函數

```
> rownames(theDF)
> rownames(theDF) <- c("One", "Two", "Three", "Four",
"Five", "Six", "Seven", "Eight", "Nine", "Ten")
> rownames(theDF)
> rownames(theDF) <- NULL
> rownames(theDF)
```

rownames(theDF) <- NULL，可以讓其設回預設值!!!

# Data.frame

```
> head(theDF)
  First Second     Sport
1    10      -4   Hockey
2     9      -3 Football
3     8      -2 Baseball
4     7      -1  Curling
5     6       0   Rugby
6     5       1 Lacrosse
> head(theDF, n = 7)
  First Second     Sport
1    10      -4   Hockey
2     9      -3 Football
3     8      -2 Baseball
4     7      -1  Curling
5     6       0   Rugby
6     5       1 Lacrosse
7     4       2 Basketball
> tail(theDF)
  First Second     Sport
5     6       0   Rugby
6     5       1 Lacrosse
7     4       2 Basketball
8     3       3   Tennis
9     2       4 Cricket
10    1       5 Soccer
> |
```

head()、tail() 函數，我們可以聯想 Linux 指令中的 head、tail 類似，前者顯示前幾列，後者顯示後幾列。

當然了可以指定前幾列要顯示多少行、後幾列要顯示多少行!!!

```
> head(theDF)
> head(theDF, n = 7)
> tail(theDF)
```

# Data.frame

The screenshot shows an R console window titled "R Console". The code entered and its output are as follows:

```
> class(theDF)
[1] "data.frame"
> theDF$Sport
 [1] Hockey     Football   Baseball   Curling    Rugby      Lacrosse
[7] Basketball Tennis    Cricket    Soccer
10 Levels: Baseball Basketball Cricket Curling Football Hockey ... Tennis
> theDF[3, 2]
[1] -2
> theDF[3, 2:3]
  Second Sport
3     -2 Baseball
> theDF[c(3, 5), 2]
[1] -2 0
> theDF[c(3, 5), 2:3]
  Second Sport
3     -2 Baseball
5     0   Rugby
> theDF[, 3]
 [1] Hockey     Football   Baseball   Curling    Rugby      Lacrosse
[7] Basketball Tennis    Cricket    Soccer
10 Levels: Baseball Basketball Cricket Curling Football Hockey ... Tennis
> theDF[, 2:3]
  Second Sport
1     -4   Hockey
2     -3   Football
3     -2  Baseball
4     -1  Curling
5     0   Rugby
6     1  Lacrosse
7     2 Basketball
8     3   Tennis
9     4  Cricket
10    5  Soccer
> theDF[2, ]
  First Second Sport
2      9     -3 Football
> theDF[2:4, 1]
  First Second Sport
2      9     -3 Football
3      8     -2 Baseball
4      7     -1 Curling
>
>
```

head()、再來就是利用 class() 函數 觀察資料框架的型別

> class(theDF)

利用資料框架的某直行已經被指定的名稱來查詢該直行的所有內容

> theDF\$Sport

"[]" 在此應用於 放入兩個值，各自代表行與列，這樣便能叫出特定位置的值

> theDF[3, 2]

可以用先前的 "[]" 跟 ":" 組合在一起使用

## 資料框變數名稱[橫列, 直行]

":" 使用連續的組合與 "c( , )" 等方式，也可以全部混用，必要時能將某行空者查詢整列，反之可將某列位置空著查詢整行。

# Data.frame

The screenshot shows an R console window titled "R Console". The code examples demonstrate various ways to subset a data frame named "theDF".

```
> theDF[, c("First", "Sport")]
   First     Sport
1    10    Hockey
2     9   Football
3     8   Baseball
4     7    Curling
5     6     Rugby
6     5 Lacrosse
7     4 Basketball
8     3    Tennis
9     2   Cricket
10    1   Soccer
> theDF[, "Sport"]
[1] Hockey   Football  Baseball  Curling   Rugby      Lacrosse
[7] Basketball Tennis   Cricket   Soccer
10 Levels: Baseball Basketball Cricket Curling Football Hockey ... Tennis
> class(theDF[, "Sport"])
[1] "factor"
> theDF["Sport"]
      Sport
1    Hockey
2   Football
3   Baseball
4    Curling
5     Rugby
6   Lacrosse
7 Basketball
8    Tennis
9   Cricket
10   Soccer
> class(theDF["Sport"])
[1] "data.frame"
> theDF[["Sport"]]
[1] Hockey   Football  Baseball  Curling   Rugby      Lacrosse
[7] Basketball Tennis   Cricket   Soccer
10 Levels: Baseball Basketball Cricket Curling Football Hockey ... Tennis
> class(theDF[["Sport"]])
[1] "factor"
>
> |
```

使用直行的名稱來查看整行，並將其名稱定為字元向量作為行的引數，也可使用指定的直行名稱去查特定行的內容

利用字元向量的方式

```
> theDF[, c("First", "Sport")]
```

利用 [] 直接抓那行名稱，但此法回傳的會是 "factor"

```
> theDF[, "Sport"]
```

```
> class(theDF[, "Sport"])
```

class(theDF[, "Sport"]) 是看這單一行的組合，所以 class 紿的是 factor，但接下來用 class(theDF[["Sport"]]) 問的是data.frame 整行所以 class 紿的是 data.frame。

只顯示直排 !!!

```
> theDF["Sport"]
```

```
> class(theDF["Sport"])
```

```
> theDF[["Sport"]]
```

```
> class(theDF[["Sport"]])
```

"drop = FALSE"，是讓由原本的橫行排列轉為直行排列

# Data.frame

```
> theDF[, "Sport"]
 [1] Hockey    Football   Baseball  Curling    Rugby     Lacrosse
[7] Basketball Tennis    Cricket   Soccer
10 Levels: Baseball Basketball Cricket Curling Football ... Tennis
> theDF[, "Sport", drop = FALSE]
      Sport
One    Hockey
Two    Football
Three  Baseball
Four   Curling
Five   Rugby
Six    Lacrosse
Seven Basketball
Eight  Tennis
Nine   Cricket
Ten    Soccer
> theDF[, "Sport", drop = TRUE]
 [1] Hockey    Football   Baseball  Curling    Rugby     Lacrosse
[7] Basketball Tennis    Cricket   Soccer
10 Levels: Baseball Basketball Cricket Curling Football ... Tennis
> |
```

```
> theDF[, "Sport", drop = FALSE]
> theDF[, "Sport", drop = TRUE]
> class(theDF[, "Sport", drop = FALSE])
> class(theDF[, "Sport", drop = TRUE])
> theDF[, 3, drop = FALSE]
> theDF[, 3, drop = TRUE]
> class(theDF[, 3, drop = FALSE])
> class(theDF[, 3, drop = TRUE])
```

# Data.frame

```
R R Console
> newFactor <- factor(c("Pennsylvania", "New York", "New Jersey", "New York",
+                         "Tennessee", "Massachusetts", "Pennsylvania", "New York"))
> model.matrix(~newFactor - 1)
   newFactorMassachusetts newFactorNew Jersey newFactorNew York
1                      0                      0                      0
2                      0                      0                      1
3                      0                      1                      0
4                      0                      0                      1
5                      0                      0                      0
6                      1                      0                      0
7                      0                      0                      0
8                      0                      0                      1
   newFactorPennsylvania newFactorTennessee
1                     1                      0
2                     0                      0
3                     0                      0
4                     0                      0
5                     0                      1
6                     0                      0
7                     1                      0
8                     0                      0
attr("assign")
[1] 1 1 1 1 1
attr("contrasts")
attr("contrasts")$newFactor
[1] "contr.treatment"
> |
```

factor 可被特別的方式儲存，若查看 data.frame 表示方式，可利用 model.matrix 來建立指標 (虛擬變數 - dummy variable)。

```
newFactor <- factor(c("Pennsylvania",
"New York", "New Jersey",
"New York", "Tennessee",
"Massachusetts",
"Pennsylvania", "New York"))

model.matrix(~newFactor - 9)
```

產生數個直行，每行代表 factor 一個 level，某列有存在 level，存在 -> 1、不存在 -> 0  
New York 這個元素存在的位置上就會顯示 1，意思他在這三個位置皆有重複，可以觀察結果中的 newFactorNew York!!!

# List

在有些的中譯本會翻譯成"清單"，他跟向量不一樣的地方是可以塞不同型別的資料，包含了前面的 data.frame、向量、numeric、character 混合 XD

```
R Console
> list(1, 2, 3)
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

> list(c(1, 2, 3))
[[1]]
[1] 1 2 3

> (list3 <- list(c(1, 2, 3), 3:7))
[[1]]
[1] 1 2 3

[[2]]
[1] 3 4 5 6 7

> list(theDF, 1:10)
[[1]]

  First Second     Sport
1     10      -4   Hockey
2      9      -3 Football
3      8      -2  Baseball
4      7      -1  Curling
5      6       0    Rugby
6      5       1 Lacrosse
7      4       2 Basketball
8      3       3    Tennis
9      2       4   Cricket
10     1       5   Soccer

[[2]]
[1] 1 2 3 4 5 6 7 8 9 10

> |
```

注意!!! List 自己也可以塞自己

我們利用前面所建的 theDF 資料框架與兩個向量來玩!!!

```
> list(1, 2, 3)
> list(c(1, 2, 3))
> (list3 <- list(c(1, 2, 3), 3:7))
> list(theDF, 1:10)
> list5 <- list(theDF, 1:10, list3)
> list5
```

# List

在有些的中譯本會翻譯成"清單"，他跟向量不一樣的地方是可以塞不同型別的資料，包含了前面的 data.frame、向量、numeric、character 混合 XD

```
R R Console

> list5 <- list(theDF, 1:10, list3)
> list5
[[1]]
   First Second     Sport
1      10     -4    Hockey
2       9     -3   Football
3       8     -2  Baseball
4       7     -1   Curling
5       6      0    Rugby
6       5      1 Lacrosse
7       4      2 Basketball
8       3      3    Tennis
9       2      4   Cricket
10      1      5   Soccer

[[2]]
[1] 1 2 3 4 5 6 7 8 9 10

[[3]]
[[3]][[1]]
[1] 1 2 3

[[3]][[2]]
[1] 3 4 5 6 7
```

注意!!! List 自己也可以塞自己

我們利用前面所建的 theDF 資料框架與兩個向量來玩!!!

```
> list(1, 2, 3)
> list(c(1, 2, 3))
> (list3 <- list(c(1, 2, 3), 3:7))
> list(theDF, 1:10)
> list5 <- list(theDF, 1:10, list3)
> list5
```

# List

既然 data.frame 資料框可以命名，類推 List 也是可行的，在此用 names( ) 進行對 list 的查詢與指派，如未曾命名會給你 NULL 值。

```
R R Console
> names(list5)
NULL
> names(list5) <- c("data.frame", "vector", "list")
> names(list5)
[1] "data.frame" "vector"      "list"
> list5
$data.frame
  First Second     Sport
1     10     -4    Hockey
2      9     -3   Football
3      8     -2  Baseball
4      7     -1   Curling
5      6      0    Rugby
6      5      1 Lacrosse
7      4      2 Basketball
8      3      3    Tennis
9      2      4   Cricket
10     1      5   Soccer

$vector
[1] 1 2 3 4 5 6 7 8 9 10

$list
$list[[1]]
[1] 1 2 3

$list[[2]]
[1] 3 4 5 6 7

> |
```

```
> names(list5)
> names(list5) <- c("data.frame", "vector", "list")
> names(list5)
> list5
```

# List

```
R R Console

> list6 <- list(TheDataFrame = theDF, TheVector = 1:10, TheList = list3)
> names(list6)
[1] "TheDataFrame" "TheVector"      "TheList"
> list6
$TheDataFrame
   First Second     Sport
1     10     -4    Hockey
2      9     -3  Football
3      8     -2  Baseball
4      7     -1   Curling
5      6      0    Rugby
6      5      1 Lacrosse
7      4      2 Basketball
8      3      3    Tennis
9      2      4   Cricket
10     1      5   Soccer

$TheVector
[1] 1 2 3 4 5 6 7 8 9 10

$TheList
$TheList[[1]]
[1] 1 2 3

$TheList[[2]]
[1] 3 4 5 6 7

> |
```

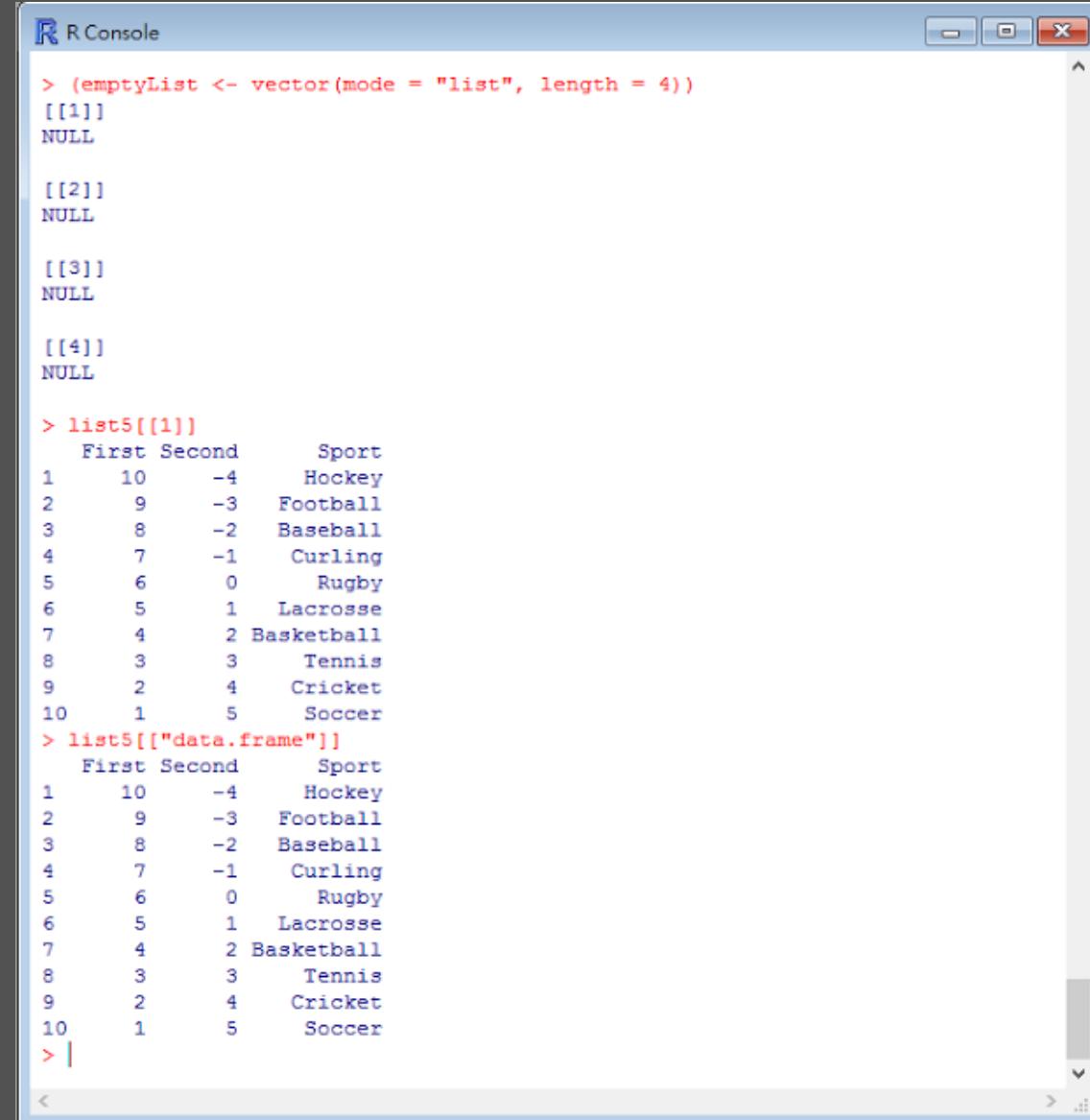
一開始直接建立 list 時直接指派!!!

```
> list6 <- list(TheDataFrame = theDF, TheVector = 1:10,
  TheList = list3)
> names(list6)
> list6
```

# List

用 vector 建立 某長度為空的 list

> (emptyList <- vector(mode = "list", length = 4))



R Console window showing R code execution and output. The code creates an empty list of length 4 and then prints its elements at indices 1 through 4, all of which are NULL. It then prints the contents of list5[[1]] and list5[["data.frame"]], both of which show a data frame with columns First, Second, and Sport, containing 10 rows of data.

```
> (emptyList <- vector(mode = "list", length = 4))
[[1]]
NULL

[[2]]
NULL

[[3]]
NULL

[[4]]
NULL

> list5[[1]]
   First Second     Sport
1      10     -4    Hockey
2       9     -3   Football
3       8     -2  Baseball
4       7     -1   Curling
5       6      0    Rugby
6       5      1 Lacrosse
7       4      2 Basketball
8       3      3    Tennis
9       2      4   Cricket
10      1      5   Soccer

> list5[["data.frame"]]
   First Second     Sport
1      10     -4    Hockey
2       9     -3   Football
3       8     -2  Baseball
4       7     -1   Curling
5       6      0    Rugby
6       5      1 Lacrosse
7       4      2 Basketball
8       3      3    Tennis
9       2      4   Cricket
10      1      5   Soccer
```

然後也可以查詢 list 單一元素，用索引位置或者索引名稱皆可!!!

> list5[[1]]

> list5[["data.frame"]]

# List

```
R R Console
> list5[[1]]$Sport
[1] Hockey   Football  Baseball  Curling   Rugby      Lacrosse
[7] Basketball Tennis   Cricket    Soccer
10 Levels: Baseball Basketball Cricket Curling Football Hockey ... Tennis
> list5[[1]][, "Second"]
[1] -4 -3 -2 -1  0  1  2  3  4  5
> list5[[1]][, "Second", drop = FALSE]
  Second
1     -4
2     -3
3     -2
4     -1
5      0
6      1
7      2
8      3
9      4
10     5
> length(list5)
[1] 3
> list5[[4]] <- 2
> length(list5)
[1] 4
> list5[["NewElement"]] <- 3:6
> length(list5)
[1] 5
```

視單一元素的問題，當然我們會碰到 factor 問題，所以前面的 drop = FALSE 又可以繼續拿來玩。

我們可以觀看下面"結果"當中的 list5[[1]]\$Sport，"\$Sport" 是 theDF 資料框當中的某直行。

```
> list5[[1]]$Sport
> list5[[1]][, "Second"]
> list5[[1]][, "Second", drop = FALSE]
```

再來就是我們能在 list 中，進行進行查詢或者做附加元素並命名的動作。

```
> length(list5)
> list5[[4]] <- 2
> length(list5)
> list5[["NewElement"]] <- 3:6
> length(list5)
> names(list5)
> list5
```

# List

```
R R Console
> names(list5)
[1] "data.frame" "vector"      "list"        ""
> list5
$data.frame
  First Second     Sport
1    10     -4   Hockey
2     9     -3 Football
3     8     -2 Baseball
4     7     -1  Curling
5     6      0   Rugby
6     5      1 Lacrosse
7     4      2 Basketball
8     3      3   Tennis
9     2      4 Cricket
10    1      5 Soccer

$vector
[1] 1 2 3 4 5 6 7 8 9 10

$list
$list[[1]]
[1] 1 2 3

$list[[2]]
[1] 3 4 5 6 7

[[4]]
[1] 2

$NewElement
[1] 3 4 5 6

> |
```

視單一元素的問題，當然我們會碰到 factor 問題，所以前面的 drop = FALSE 又可以繼續拿來玩。

我們可以觀看下面"結果"當中的 list5[[1]]\$Sport，"\$Sport" 是 theDF 資料框當中的某直行。

```
> list5[[1]]$Sport
> list5[[1]][, "Second"]
> list5[[1]][, "Second", drop = FALSE]
```

再來就是我們能在 list 中，進行進行查詢或者做附加元素並命名的動作。

```
> length(list5)
> list5[[4]] <- 2
> length(list5)
> list5[["NewElement"]] <- 3:6
> length(list5)
> names(list5)
> list5
```

# Matrix & Array

## 1-1 基本觀念

### 一、定義：

由  $m \times n$  個實數  $a_{ij}$ ,  $1 \leq i \leq m$ ,  $1 \leq j \leq n$  排成的  $m \times n$  長方形陣列(array)稱為  $m \times n$  矩陣，通常以大寫英文字母  $A, B, C, \dots$  代表，

$$\text{如下 : } A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

所有  $m \times n$  的矩陣所成的集合記作  $M_{m \times n}(\mathbb{R})$ (若  $a_{ij}$  是複數，則記作  $M_{m \times n}(\mathbb{C})$ )。

### 二、專有名詞

1.  $m \times n$  稱為矩陣的階 (order)。
2.  $1 \times n$  的矩陣稱為列矩陣(row matrix)或列向量(row vector)，形如  $[a_1 \ a_2 \ \cdots \ a_n]$ ， $m \times 1$  的矩陣稱為行矩陣(column matrix)或行向量(column vector)  $\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$ 。
3.  $a_{ij}$  稱為  $A$  的第  $i$  列，第  $j$  行的元素(entry)  
 $(a_{i1} \ a_{i2} \ \cdots \ a_{in})$  稱為  $A$  的第  $i$  列  
 $\begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{bmatrix}$  稱為  $A$  的第  $j$  行。
4. 若  $A$  的行、列個數相等，即  $m = n$ ，則稱  $A$  為  $n$  階方陣(square matrix of order  $n$ )。
5. 若  $A$  是  $n$  階方陣， $a_{11}, a_{22}, \dots, a_{nn}$  稱為  $A$  的主對角線(main diagonal)。
6. 若  $a_{ij} = 0, 1 \leq i \leq m, 1 \leq j \leq n$ ，則稱  $A$  為零矩陣，記作  $O$ 。

# Matrix & Array

## 1-2 矩陣的運算

### 一、矩陣相等

$A, B \in M_{m \times n}(\mathbb{R})$ , 若  $a_{ij} = b_{ij}, \forall 1 \leq i \leq m, 1 \leq j \leq n$ , 則稱  $A$  與  $B$  相等。

例 1:  $A = \begin{bmatrix} x+2y & z^2 \\ 3 & 4 \end{bmatrix}$ ,  $B = \begin{bmatrix} 7 & 3z-2 \\ 3 & 2x-y \end{bmatrix}$ , 若  $A = B$ , 求  $x, y, z$ 。

解:  $x+2y=7, 4=2x-y, z^2=3z-2 \Rightarrow x=3, y=2, z=1\text{或}2$

### 二、加法(sum)

若  $A, B \in M_{m \times n}(\mathbb{R})$ , 則  $A+B = [a_{ij} + b_{ij}]$

例 2:  $A = \begin{bmatrix} 1 & 2 & 5 \\ -3 & 0 & 2 \end{bmatrix}$ ,  $B = \begin{bmatrix} 0 & -1 & 3 \\ 4 & 2 & 1 \end{bmatrix} \Rightarrow A+B = \begin{bmatrix} 1 & 1 & 8 \\ 1 & 2 & 3 \end{bmatrix}$

### 三、純量積(scalar multiplication)

1. 若  $A \in M_{m \times n}(\mathbb{R})$ , 則  $kA = [ka_{ij}], k \in \mathbb{R}$

例 3:  $A = \begin{bmatrix} 1 & 2 & 5 \\ -3 & 0 & 2 \end{bmatrix} \Rightarrow 3A = \begin{bmatrix} 3 & 6 & 15 \\ -9 & 0 & 6 \end{bmatrix}$

2. 加法與純量積滿足下列運算規則( $A, B, C \in M_{m \times n}(\mathbb{R}), k, l \in \mathbb{R}$ )

(1) 交換律:  $A+B=B+A$ 。

(2) 結合律:  $(A+B)+C=A+(B+C), (kl)A=k(lA)$ 。

(3) 分配律:  $k(A+B)=kA+kB, (k+l)A=kA+lA$

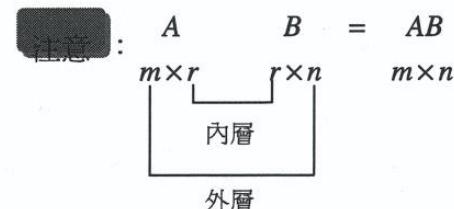
(4) 加法單位元元素:  $A+O=O+A=A$ , 即  $O$  是加法單位元元素。

注: 減法可以看成加”負的”, 即  $A-B=A+(-B)=\begin{bmatrix} a_{ij}-b_{ij} \end{bmatrix}$

### 四、乘法(product)

1. 若  $A \in M_{m \times r}(\mathbb{R})$ , 則  $B \in M_{r \times n}(\mathbb{R})$ , 則  $C = AB$  為  $m \times n$  矩陣, 其中

$$C_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{ir}b_{rj} = \sum_{k=1}^r a_{ik}b_{kj}.$$



# Matrix & Array

The screenshot shows the R Console window with the following session history:

```
> A <- matrix(1:10, nrow = 5)
> B <- matrix(21:30, nrow = 5)
> C <- matrix(21:40, nrow = 2)
> A
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
> B
     [,1] [,2]
[1,]   21   26
[2,]   22   27
[3,]   23   28
[4,]   24   29
[5,]   25   30
> C
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]   21   23   25   27   29   31   33   35   37   39
[2,]   22   24   26   28   30   32   34   36   38   40
> nrow(A)
[1] 5
> ncol(A)
[1] 2
> dim(A)
[1] 5 2
> |
```

視單一在 R 的世界中 Matrix 只為兩個維度矩陣，而陣列 Array 則為多維度矩陣!!!

基本上從上面的數學意涵，我們就可以知道矩陣當中處理的，必須是同一個資料型別，因為是以元素對元素的方式進行運算，而我們之前在資料框架 (data.frame) 中的幾項查詢函數都能再拿來玩!!!

1. nrow()
2. ncol()
3. dim()

在此我們建立 A、B、C 三個  $5 \times 2$  matrix 矩陣，並嘗試查詢。

```
> A <- matrix(1:10, nrow = 5)
> B <- matrix(21:30, nrow = 5)
> C <- matrix(21:40, nrow = 2)
> A
> B
> C
> nrow(A)
> ncol(A)
> dim(A)
```

矩陣的運算必須符合上述矩陣的規則!!!

# Matrix & Array

```
R R Console

> A + B
 [,1] [,2]
[1,] 22 32
[2,] 24 34
[3,] 26 36
[4,] 28 38
[5,] 30 40

> A * B
 [,1] [,2]
[1,] 21 156
[2,] 44 189
[3,] 69 224
[4,] 96 261
[5,] 125 300

> A == B
 [,1] [,2]
[1,] FALSE FALSE
[2,] FALSE FALSE
[3,] FALSE FALSE
[4,] FALSE FALSE
[5,] FALSE FALSE

> A %*% t(B)
 [,1] [,2] [,3] [,4] [,5]
[1,] 177 184 191 198 205
[2,] 224 233 242 251 260
[3,] 271 282 293 304 315
[4,] 318 331 344 357 370
[5,] 365 380 395 410 425

> |
```

> A + B

> A \* B

> A == B

> A %\*% t(B)

"A == B" -> 可勘查A、B二矩陣元素是否一樣

"A %\*% t(B)" -> A、B二矩陣進行運算，t()的意義為轉置，必須符合矩陣運算規則的要求> dim(A)

# Matrix & Array

The screenshot shows an R console window titled "R Console". The code entered is:

```
> colnames(A)
NULL
> rownames(A)
NULL
> colnames(A) <- c("Left", "Right")
> rownames(A) <- c("1st", "2nd", "3rd", "4th", "5th")
>
> colnames(B)
NULL
> rownames(B)
NULL
> colnames(B) <- c("First", "Second")
> rownames(B) <- c("One", "Two", "Three", "Four", "Five")
>
> colnames(C)
NULL
> rownames(C)
NULL
> colnames(C) <- LETTERS[1:10]
> rownames(C) <- c("Top", "Bottom")
> t(A)
   1st 2nd 3rd 4th 5th
Left    1    2    3    4    5
Right   6    7    8    9   10
> A %*% C
      A     B     C     D     E     F     G     H     I     J
1st 153 167 181 195 209 223 237 251 265 279
2nd 196 214 232 250 268 286 304 322 340 358
3rd 239 261 283 305 327 349 371 393 415 437
4th 282 308 334 360 386 412 438 464 490 516
5th 325 355 385 415 445 475 505 535 565 595
> |
```

The output shows the creation of matrices A, B, and C, and their transpose t(A). Matrix A has columns Left and Right, and rows 1st through 5th. Matrix B has columns First and Second, and rows One through Five. Matrix C has columns A through J, and rows Top and Bottom. The final output is the product of t(A) and C.

資料框架(data.frame)跟矩陣(Matrix)兩者的共同點，在於其二者行與列皆可查詢指派名稱!!!

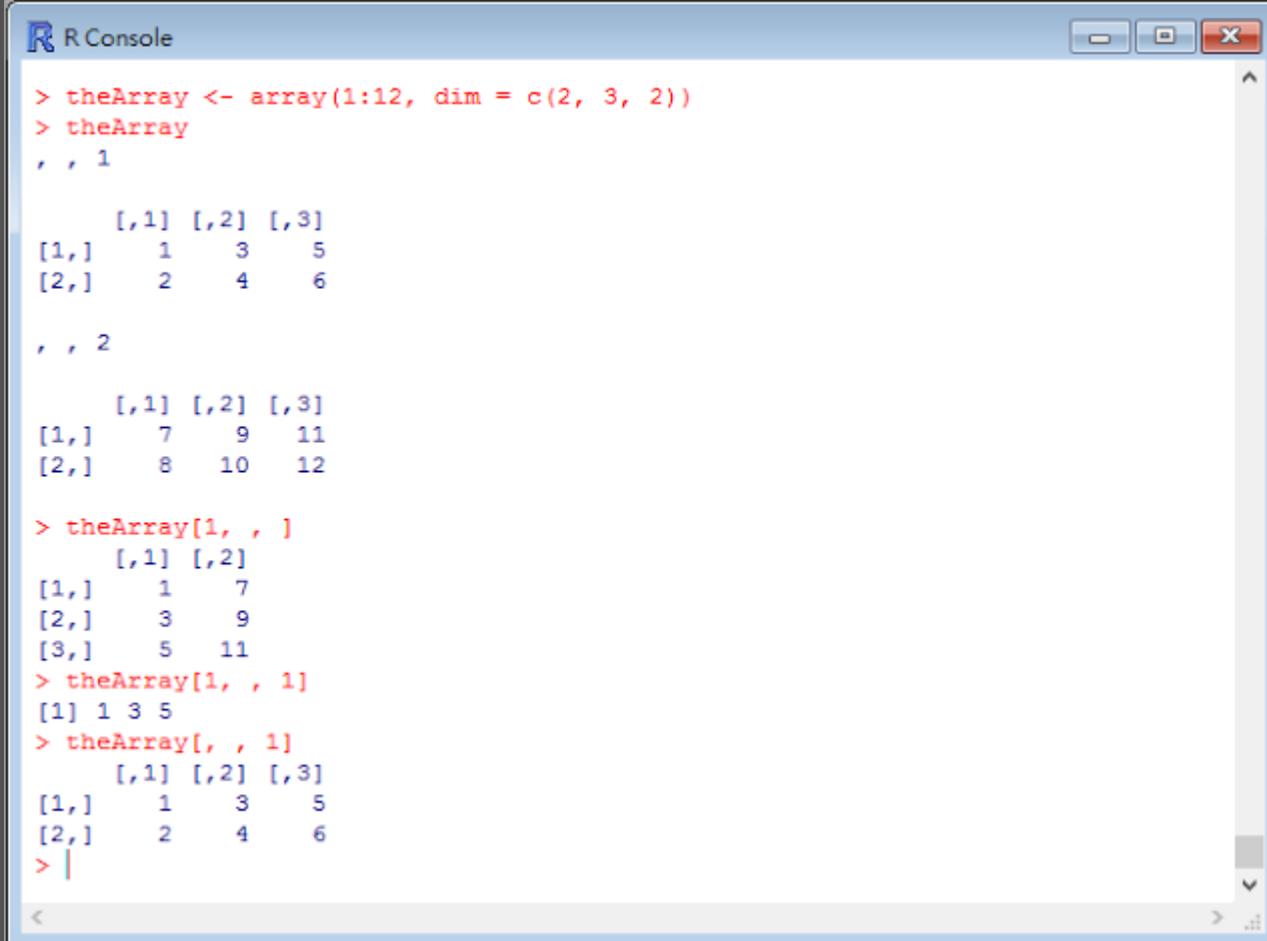
```
> colnames(A)
> rownames(A)
> colnames(A) <- c("Left", "Right")
> rownames(A) <- c("1st", "2nd", "3rd", "4th", "5th")
> colnames(B)
> rownames(B)
> colnames(B) <- c("First", "Second")
> rownames(B) <- c("One", "Two", "Three", "Four",
"Five")
> colnames(C)
> rownames(C)
> colnames(C) <- LETTERS[1:10]
> rownames(C) <- c("Top", "Bottom")
```

“LETTERS[1:10]” , LETTERS指的是大寫，1:10是指連續產生1~10，這樣會變成1 - A、2 - B、3 - C ...，再來這個部分就是說明矩陣轉置在R當中的樣子。

```
> t(A)
> A %*% C
```

R 當中的陣列就是多維度的矩陣!!

# Matrix & Array



The screenshot shows the R Console window with the following session history:

```
> theArray <- array(1:12, dim = c(2, 3, 2))
> theArray
, , 1

 [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

 [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

> theArray[1, , ]
, , 1

 [,1] [,2]
[1,]    1    7
[2,]    3    9
[3,]    5   11
> theArray[1, , 1]
[1] 1 3 5
> theArray[, , 1]
, , 1

 [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
```

```
> theArray <- array(1:12, dim = c(2, 3, 2))
> theArray
> theArray[1, , ]
> theArray[1, , 1]
> theArray[, , 1]
```

# 重點

R 資料結構基本上都是以向量(vector)為基礎組成

在資料結構中只有 list 能將不同的資料形式塞在一起(包含 list)

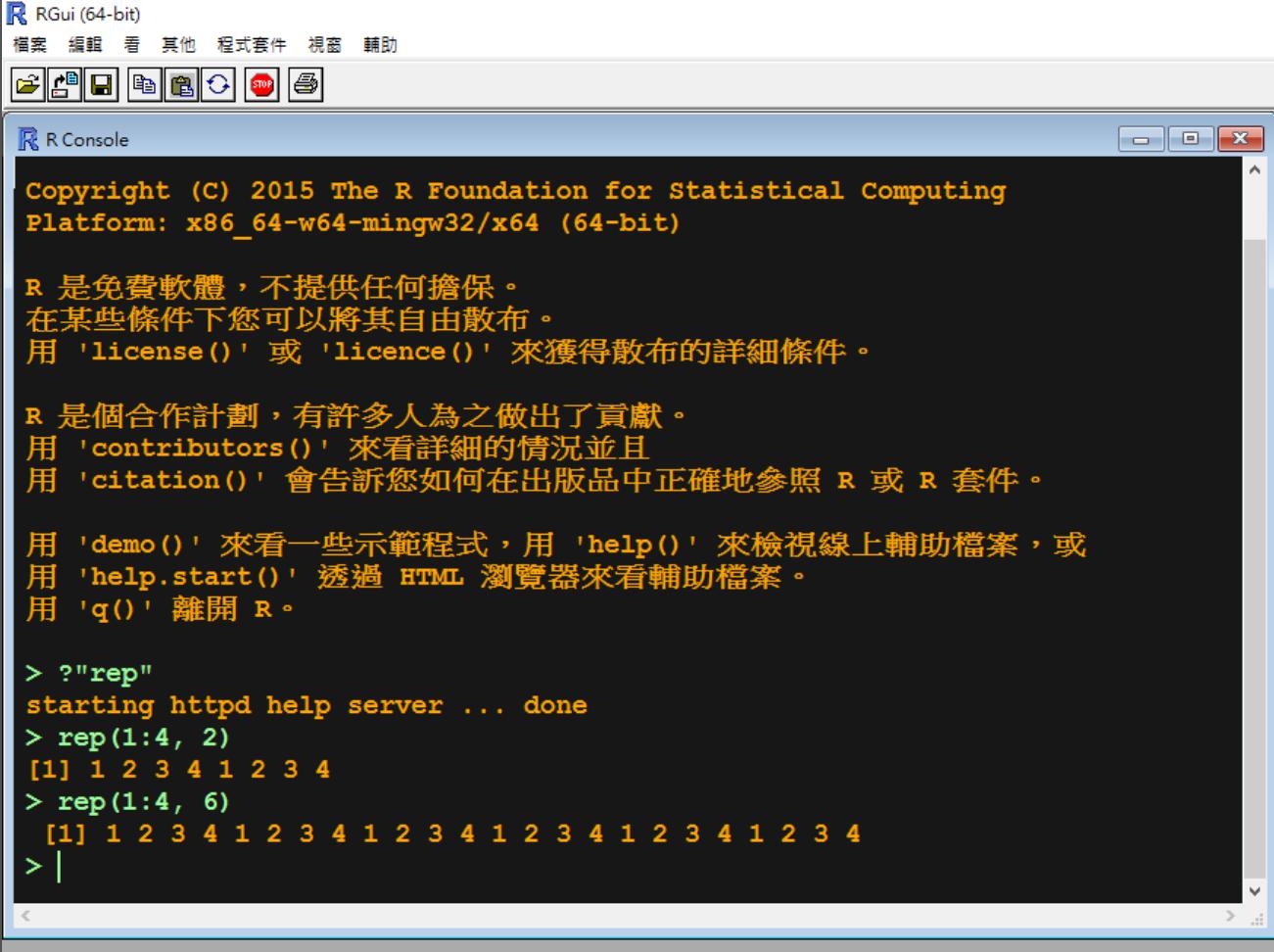
data.frame則是能同時處理不同資料形式 (不同資料形式向量排成表)

Matrix只為兩個維度矩陣，而陣列 Array 則為多維度矩陣

# R rep()

rep( [ 指定的向量 ], [使之重複的次數] )

```
rep(1:4, 2)  
rep(1:4, 6)
```



The screenshot shows the RGui (64-bit) application window. The title bar says "RGui (64-bit)". Below it is a menu bar with Chinese labels: 檔案 (File), 編輯 (Edit), 看 (View), 其他 (Other), 程式套件 (Programs), 視窗 (Windows), and 輔助 (Help). Under the "視窗" (Windows) menu, there are several icons: a document, a folder, a magnifying glass, a search icon, a circular arrow, a stop sign, and a file. The main window is titled "R Console". It displays the R startup message in yellow text:

```
Copyright (C) 2015 The R Foundation for Statistical Computing  
Platform: x86_64-w64-mingw32/x64 (64-bit)
```

Then, it shows the R license information in yellow text:

```
R 是免費軟體，不提供任何擔保。  
在某些條件下您可以將其自由散布。  
用 'license()' 或 'licence()' 來獲得散布的詳細條件。
```

It also provides information about the R contributors and citation in yellow text:

```
R 是個合作計劃，有許多人為之做出了貢獻。  
用 'contributors()' 來看詳細的情況並且  
用 'citation()' 會告訴您如何在出版品中正確地參照 R 或 R 套件。
```

Finally, it shows examples of the 'rep' function in green text:

```
> ?"rep"  
starting httpd help server ... done  
> rep(1:4, 2)  
[1] 1 2 3 4 1 2 3 4  
> rep(1:4, 6)  
[1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4  
> |
```

# R 對於 data.frame 資料進行序列編號

```
> a = c(11:20)
> b = c(91:100)
> kan = data.frame( data1 = a, data2 = b )
> kan
   data1 data2
1      11    91
2      12    92
3      13    93
4      14    94
5      15    95
6      16    96
7      17    97
8      18    98
9      19    99
10     20   100
> nrow(kan)
[1] 10
> |
```

kan 是 data.frame 是由自行建立的資料，當中包含了 a 和 b 兩個向量，並命名為 data1、data2，在此利用 nrow() 這個函數，他會顯示這個欄位有多少 “列”。而 rep() 函數則是會產生重複的資料。

所以我就利用 `rep(1:nrow(kan))` 產生一個與 `kan` 列數一樣多的直行向量資料，並利用 `cbind()` 合併起來。

```
> a = c(11:20)
> b = c(91:100)
> kan = data.frame( data1 = a, data2 = b )
> kan
> nrow(kan)
> id = rep(1:nrow(kan))
> kan = cbind( id, kan)
> kan
> ikk = rep(10)
> ikk
> kan = cbind( ikk, kan)
> kan
```

# R 處理單一 data.frame 直欄合併

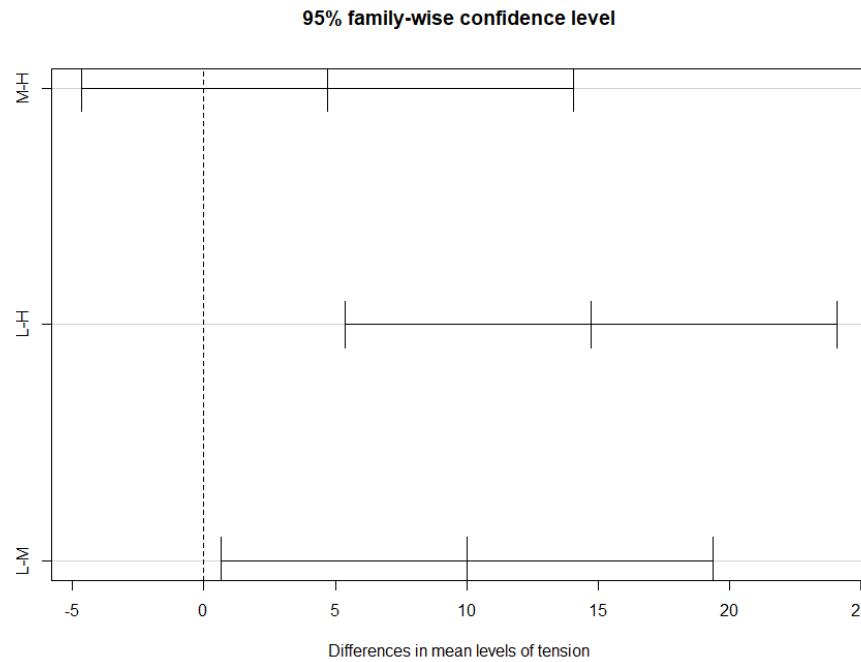
```
> require(reshape2)
Loading required package: reshape2
> head(tips)
  total_bill  tip    sex smoker day   time size
1     16.99 1.01 Female   No Sun Dinner     2
2     10.34 1.66   Male   No Sun Dinner     3
3     21.01 3.50   Male   No Sun Dinner     3
4     23.68 3.31   Male   No Sun Dinner     2
5     24.59 3.61 Female   No Sun Dinner     4
6     25.29 4.71   Male   No Sun Dinner     4
> names(tips)
[1] "total_bill" "tip"      "sex"       "smoker"    "day"      "time"      "size"
> tipc1 = tips[,c( 1, 3, 5)] > tipc2 = tips[,-c( 1, 3, 5)]
> head(tipc1)                > head(tipc2)
  total_bill    sex day          tip smoker   time size
1     16.99 Female Sun 1 1.01   No Dinner     2
2     10.34   Male Sun 2 1.66   No Dinner     3
3     21.01   Male Sun 3 3.50   No Dinner     3
4     23.68   Male Sun 4 3.31   No Dinner     2
5     24.59 Female Sun 5 3.61   No Dinner     4
6     25.29   Male Sun 6 4.71   No Dinner     4
>
> tmtipc3 = which(!names(tips) %in% c( "tip", "size"))
> tmtipc3
[1] 1 3 4 5 6
> tipc3 = tips[,tmtipc3]
> head(tipc3)
  total_bill    sex smoker day   time
1     16.99 Female   No Sun Dinner
2     10.34   Male   No Sun Dinner
3     21.01   Male   No Sun Dinner
4     23.68   Male   No Sun Dinner
5     24.59 Female   No Sun Dinner
6     25.29   Male   No Sun Dinner
> |
```

在這裡我先用 names 可以看到 reshape2 套件的內建資料小費資料 (tips) 每個欄位的開頭字串。

```
> require(reshape2)
> head(tips)
> names(tips)
> tipc1 = tips[,c( 1, 3, 5)]
> head(tipc1)
  total_bill    sex day          tip smoker   time size
1     16.99 Female Sun 1 1.01   No Dinner     2
2     10.34   Male Sun 2 1.66   No Dinner     3
3     21.01   Male Sun 3 3.50   No Dinner     3
4     23.68   Male Sun 4 3.31   No Dinner     2
5     24.59 Female Sun 5 3.61   No Dinner     4
6     25.29   Male Sun 6 4.71   No Dinner     4
>
> tmtipc3 = which(!names(tips) %in% c( "tip", "size"))
> tmtipc3
[1] 1 3 4 5 6
> tipc3 = tips[,tmtipc3]
> head(tipc3)
  total_bill    sex smoker day   time
1     16.99 Female   No Sun Dinner
2     10.34   Male   No Sun Dinner
3     21.01   Male   No Sun Dinner
4     23.68   Male   No Sun Dinner
5     24.59 Female   No Sun Dinner
6     25.29   Male   No Sun Dinner
```

# R broom 套件 統計模型轉換成 data.frame 格式

```
> # install.packages("broom")
> library("broom")
> require(graphics)
> fm = aov(breaks ~ wool + tension, data = warpbreaks)
> fm.th = TukeyHSD(fm, "tension", ordered = TRUE)
> plot(fm.th)
> (fm.thdf =tidy(fm.th))
  term comparison estimate conf.low conf.high adj.p.value
1 tension      M-H   4.722222 -4.6311985 14.07564 0.447421021
2 tension      L-H  14.722222  5.3688015 24.07564 0.001121788
3 tension      L-M  10.000000  0.6465793 19.35342 0.033626219
>
> class(fm.th)
[1] "TukeyHSD" "multicomp"
> class(fm.thdf)
[1] "data.frame"
> |
```



```
# install.packages("broom")
library("broom")
require(graphics)
fm = aov(breaks ~ wool + tension, data = warpbreaks)
fm.th = TukeyHSD(fm, "tension", ordered = TRUE)
plot(fm.th)
(fm.thdf =tidy(fm.th))
```

```
class(fm.th)
class(fm.thdf)
```

# R 時間序列物件

ts {stats}

## Time-Series Objects

### Description

The function `ts` is used to create time-series objects.

`as.ts` and `is.ts` coerce an object to a time-series and test whether an object is a time series.

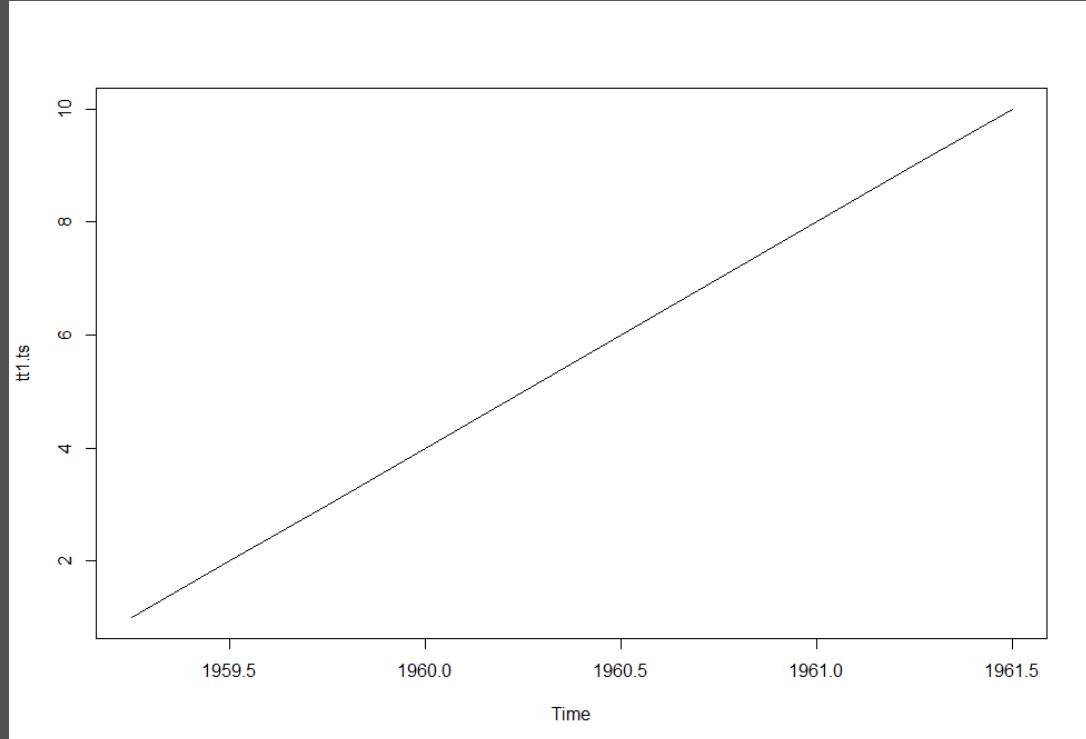
### Usage

```
ts(data = NA, start = 1, end = numeric(), frequency = 1,  
    deltat = 1, ts.eps = getOption("ts.eps"), class = , names = )  
as.ts(x, ...)  
is.ts(x)
```

1. `data` 參數為所要放入的資料來源，大多為擁有時間序列矩陣或向量。
2. `start` 為要設定的初始觀察時間。
3. `end` 為結束的時間。
4. `frequency` 為每單位時間的觀察次數。

# R 時間序列物件

```
> ttl.ts = ts(1:10, frequency = 4, start = c(1959, 2))
> class(ttl.ts)
[1] "ts"
> print(ttl.ts, calendar = TRUE)
   Qtr1 Qtr2 Qtr3 Qtr4
1959      1     2     3
1960      4     5     6     7
1961      8     9    10
> plot(ttl.ts)
> |
```

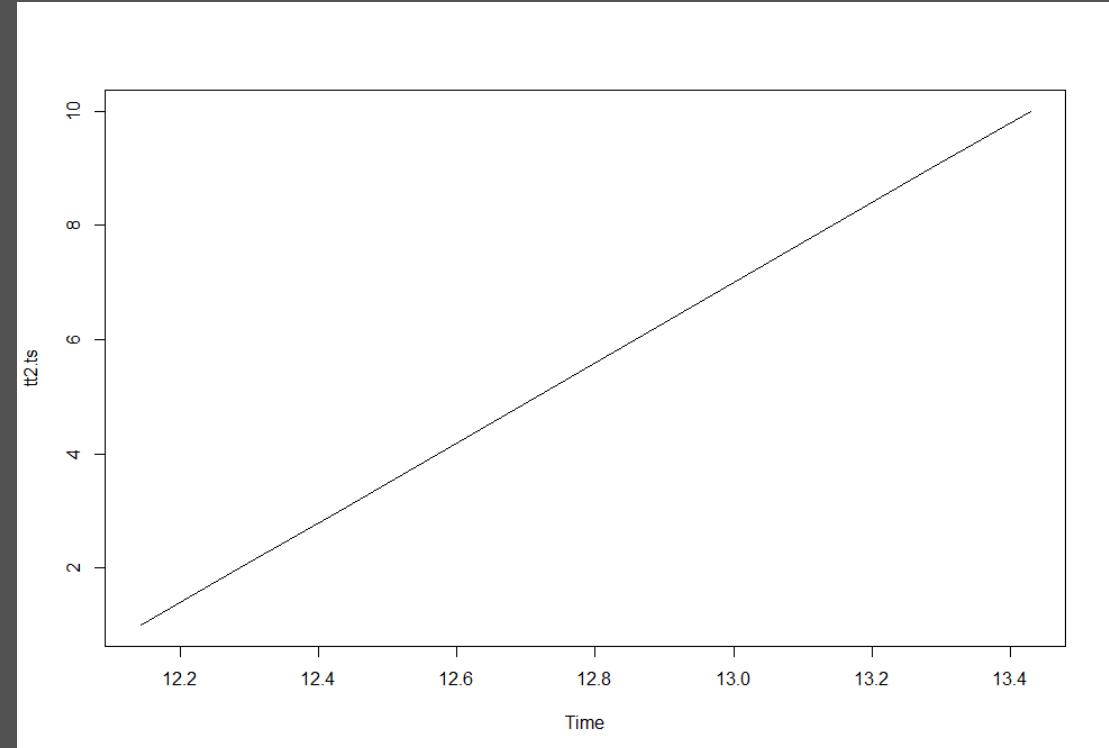


在此建立一個 1 ~ 10 的資料，範圍為 4 季，  
當中起始年分為 1959 年，而 start 的 2 則  
產生的 0.5 的間隔。

```
tt1.ts = ts(1:10, frequency = 4, start =  
c(1959, 2))  
class(tt1.ts)  
print(tt1.ts, calendar = TRUE)  
plot(tt1.ts)
```

# R 時間序列物件

```
> tt2.ts = ts(1:10, frequency = 7, start = c(12, 2))
> class(tt2.ts)
[1] "ts"
> print(tt2.ts, calendar = TRUE)
  p1 p2 p3 p4 p5 p6 p7
12   1   2   3   4   5   6
13   7   8   9  10
> plot(tt2.ts)
> |
```

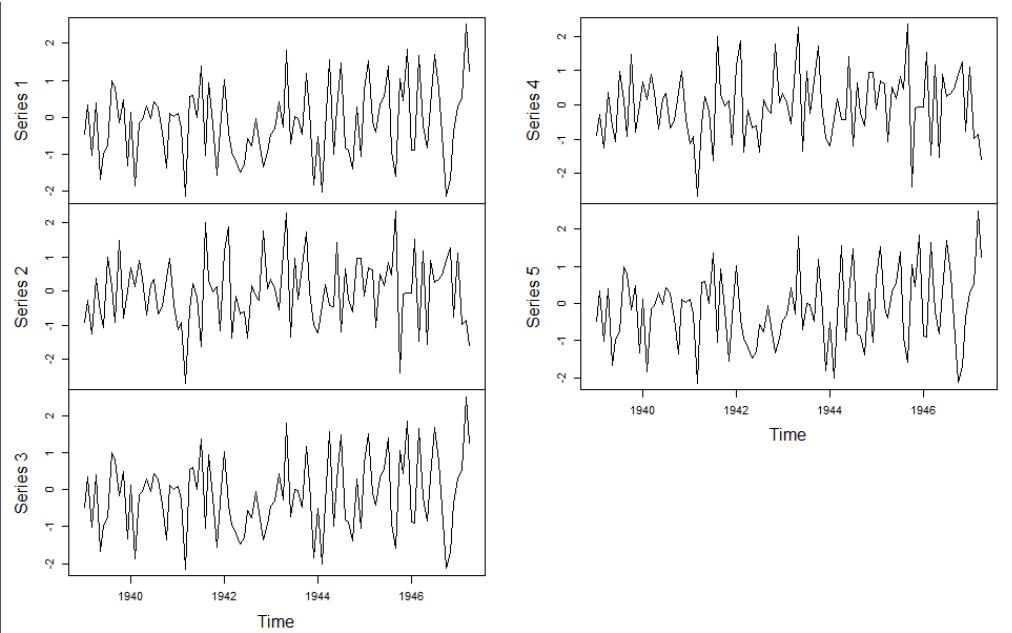


建立一個 1 ~ 10 的資料，範圍為 7 期，當中起始時間為 12，而 start 的 2 則產生的 0.2 的間隔。

```
tt2.ts = ts(1:10, frequency = 7, start = c(12, 2))
class(tt2.ts)
print(tt2.ts, calendar = TRUE)
plot(tt2.ts)
```

# R 時間序列物件

```
> demo.ts = ts(matrix(rnorm(200), 100, 5),
+ start = c(1939, 1), frequency = 12)
> head(demo.ts)
  Series 1   Series 2   Series 3   Series 4   Series 5
[1,] -0.4731973 -0.9036188 -0.4731973 -0.9036188 -0.4731973
[2,]  0.3340248 -0.2826973  0.3340248 -0.2826973  0.3340248
[3,] -1.0219492 -1.2644636 -1.0219492 -1.2644636 -1.0219492
[4,]  0.4048839  0.3789126  0.4048839  0.3789126  0.4048839
[5,] -1.6806031 -0.6103802 -1.6806031 -0.6103802 -1.6806031
[6,] -0.9871078 -1.0854620 -0.9871078 -1.0854620 -0.9871078
> NROW(demo.ts)
[1] 100
> class(demo.ts)
[1] "mts"      "ts"       "matrix"
> plot.ts(demo.ts)
> |
```

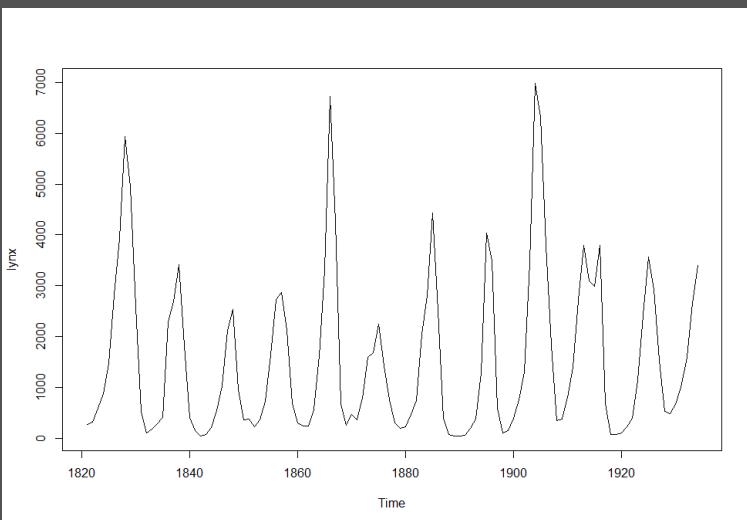


由 R 範例文件進行修改，產生一個 100 筆 5季，而 rnorm 則會使產生的隨機資料服从常態分佈，起始年份為 1939 年間隔為 1，期數為 12。

```
demo.ts = ts(matrix(rnorm(200), 100, 5),
  start = c(1939, 1), frequency = 12)
head(demo.ts)
NROW(demo.ts)
class(demo.ts)
plot.ts(demo.ts)
```

# R 時間序列物件

```
> lynx
Time Series:
Start = 1821
End = 1934
Frequency = 1
[1] 269 321 585 871 1475 2821 3928 5943 4950 2577 523 98 184 279 409 2285 2685
[18] 3409 1824 409 151 45 68 213 546 1033 2129 2536 957 361 377 225 360 731
[35] 1638 2725 2871 2119 684 299 236 245 552 1623 3311 6721 4254 687 255 473 358
[52] 784 1594 1676 2251 1426 756 299 201 229 469 736 2042 2811 4431 2511 389 73
[69] 39 49 59 188 377 1292 4031 3495 587 105 153 387 758 1307 3465 6991 6313
[86] 3794 1836 345 382 808 1388 2713 3800 3091 2985 3790 674 81 80 108 229 399
[103] 1132 2432 3574 2935 1537 529 485 662 1000 1590 2657 3396
> class(lynx)
[1] "ts"
> summary(lynx)
   Min. 1st Qu. Median      Mean 3rd Qu.      Max.
 39.0   348.2  771.0  1538.0  2566.8  6991.0
> plot(lynx)
> |
```



R 內建的 lynx 資料集，又稱為加拿大山貓數量資料，該筆資料記錄著加拿大西北部馬更些(Mackenzie)流域附近每年的山貓數量，資料所收集的時間是西元1821至1934年。

```
lynx
class(lynx)
summary(lynx)
plot(lynx)
```

# R 建立函數

```
> say.hello = function()
+ {
+   print("Hello, World!")
+ }
>
> say.hello()
[1] "Hello, World!"
> |
```

這裡開始我會先建議將要寫的程式碼，先打在記事本上或是任意自己習慣的編輯器，不然到後面越來越複雜，有可能會迷失自己這樣。

## 1. Hello World!

在此我們建立自己的 R 函數，召喚跟其他語言介紹中最常登場的 Hello World!

```
say.hello = function()
{
  print("Hello, World!")
}
```

```
say.hello()
```

# R 建立函數

```
> sprintf("Hello %s", "Jared")
[1] "Hello Jared"
>
> sprintf("Hello %s, today is %s", "Jared", "Sunday")
[1] "Hello Jared, today is Sunday"
> |
```

```
> sprintf("Hello %s, today is %s", "A", "B")
[1] "Hello A, today is B"
> |
```

## 2. 函數的引數

接下來使用 sprintf 函數，這個東西類似於JAVA中的格式化輸出!!!

前面的 %s 可以將後面所要數入的字輸入至指定的位置，在此我們可以觀察到單一項與類推多項輸入

```
sprintf("Hello %s", "Jared")
sprintf("Hello %s, today is %s", "Jared", "Sunday")
sprintf("Hello %s, today is %s", "A", "B")
```

# R 建立函數

```
> hello.person = function(name)
+ {
+   print(sprintf("Hello %s", name))
+ }
>
> hello.person("Jared")
[1] "Hello Jared"
> hello.person("Bob")
[1] "Hello Bob"
> hello.person("Sarah")
[1] "Hello Sarah"
```

利用前面講的 sprintf() 函數建字串，其自建的函數我們命為 hello.person

```
hello.person=function(name)
{
  print(sprintf("Hello %s", name))
}
```

```
hello.person("Jared")
hello.person("Bob")
hello.person("Sarah")
```

# R 建立函數

```
> hello.person = function(first, last)
+ {
+   print(sprintf("Hello %s %s", first, last))
+ }
>
> hello.person("Jared", "Lander")
[1] "Hello Jared Lander"
>
>
> hello.person(first = "Jared", last = "Lander")
[1] "Hello Jared Lander"
>
> hello.person(last = "Lander", first = "Jared")
[1] "Hello Jared Lander"
>
> hello.person("Jared", last = "Lander")
[1] "Hello Jared Lander"
>
>
> hello.person(first = "Jared", "Lander")
[1] "Hello Jared Lander"
>
> hello.person(last = "Lander", "Jared")
[1] "Hello Jared Lander"
>
> hello.person(fir = "Jared", l = "Lander")
[1] "Hello Jared Lander"
> |
```

注意!!! 我們剛剛建 hello.person() 函數 當中有一個 name 引數，在函數裡面視為作為變數利用，離開這個函數後不能存在，當然它可以同時被其他函數利用。當呼叫有多一個引數的函數，他會輸入到引數的相對位置上，或指定其引數名稱。

```
hello.person = function(first, last)
{
  print(sprintf("Hello %s %s", first, last))
}
```

如果是我會選擇用相對位置的方式，或是更保險的透過引數名稱的用法，因為透過引數名稱的方式，就算順序不一，還是能達到使用上的結果。

```
hello.person("Jared", "Lander")
hello.person(first = "Jared", last = "Lander")
hello.person(last = "Lander", first = "Jared")
hello.person("Jared", last = "Lander")
hello.person(first = "Jared", "Lander")
hello.person(last = "Lander", "Jared")
hello.person(fir = "Jared", l = "Lander")
```

# R 建立函數

```
> hello.person <- function(first, last = "Doe")
+ {
+   print(sprintf("Hello %s %s", first, last))
+ }
>
>
> hello.person("Jared")
[1] "Hello Jared Doe"
>
>
> hello.person("Jared", "Lander")
[1] "Hello Jared Lander"
> |
```

## 1. 設引數預設值

接下來我們利用前面建的 hello.person() 函數，為他的引數設其預設值，在這裡是將 Doe作為預設的姓氏 - last name

```
hello.person = function(first, last = "Doe")
{
  print(sprintf("Hello %s %s", first, last))
}
```

觀察下面的 Code 有無指定的差異!!

```
hello.person("Jared")
hello.person("Jared", "Lander")
```

# R 建立函數

```
> hello.person("Jared", extra = "Goodbye")
Error in hello.person("Jared", extra = "Goodbye") :
  unused argument (extra = "Goodbye")
>
>
> hello.person("Jared", "Lander", "Goodbye")
Error in hello.person("Jared", "Lander", "Goodbye") :
  unused argument ("Goodbye")
>
>
> hello.person <- function(first, last = "Doe", ...)
+ {
+   print(sprintf("Hello %s %s", first, last))
+ }
>
>
> hello.person("Jared", extra = "Goodbye")
[1] "Hello Jared Doe"
>
> hello.person("Jared", "Lander", "Goodbye")
[1] "Hello Jared Lander"
> |
```

## 2. 設附加引數

根據前面的設定引數預設值，我們可以在 hello.person( ) 函數中，在多插入額外引數。如果直接執行會有錯誤!!!可以觀察下面的執行結果

```
hello.person("Jared", extra = "Goodbye")
hello.person("Jared", "Lander", "Goodbye")
```

如果要成功必須在其 hello.person( ) 函數做一個小小改動，將第一行的 hello.person = function(first, last = “Doe” )，改為 hello.person = function(first, last = “Doe” , ...)。

```
hello.person = function(first, last = "Doe", ...)
{
  print(sprintf("Hello %s %s", first, last))
}
```

我們在執行剛剛的插入額外引數的動作

```
hello.person("Jared", extra = "Goodbye")
hello.person("Jared", "Lander", "Goodbye")
```

# R 建立函數

## 3.值回傳方式

正常是執行完就結束，還有我們可以利用 return() 在執行中給予明確條件的方式回傳，在此我們建立一個名為 double.num() 的函數，輸入任意 x 值 \* 2。

```
double.num = function(x)
{
  x * 2
}

# 指定任意值 x 為 5
double.num(5)
```

接下來我們在 double.num() 的函數中加入 return() 的條件

```
double.num = function(x)
{
  return(x * 2)
}
```

# 執行 任意值 x 為 5  
double.num(5)

```
double.num = function(x)
{
  return(x * 2)

  print("Hello!")
  return(17)
}
```

# 執行 任意值 x 為 5  
double.num(5)

我們可以觀察到當我們執行任意值 x 為 5，也就是 double.num(5)，我們自建的函數符合了函數第一行的條件 return(x\*2) 就跳出函數了，所以 "print("Hello!")" 與 "return(17)" 這兩行不會被執行!!!

```
> double.num = function(x)
+ {
+   x * 2
+ }
>
> double.num(5)
[1] 10
>
>
> double.num = function(x)
+ {
+   return(x * 2)
+ }
> double.num(5)
[1] 10
>
>
> double.num = function(x)
+ {
+   return(x * 2)
+ }

+   print("Hello!")
+   return(17)
+ }
> double.num(5)
[1] 10
> |
```

# R 建立函數

## 4. do.call() 函數

do.call() 函數允許我們用字串 - character 亦或是用一個物件名詞 指定其函數名稱，輸入的引數則以列表(list)形式呈現，在此我們利用前面自建的函數 hello.person() 玩玩看!!!

args = [放入引數的 list]

觀察下面的程式碼，第一行用的是字串的方式、第二行用的是物件名稱的方法 !!!

```
do.call("hello.person", args = list(first = "Jared", last = "Lander"))
do.call(hello.person, args = list(first = "Jared", last = "Lander"))
```

# 建一個名為 run.this 的函數，然後我們可以在當中求算 平均值、總和、標準差 !!!

```
run.this = function(x, func = mean)
{
  do.call(func, args = list(x))
}
```

# mean(平均值)、sum(總和)、sd(標準差)，在此我們設 1~10 之間的資料來玩

```
run.this(1:10)
run.this(1:10, mean)
run.this(1:10, sum)
run.this(1:10, sd)
```

```
> do.call("hello.person", args = list(first = "Jared", last = "Lander"))
[1] "Hello Jared Lander"
>
> do.call(hello.person, args = list(first = "Jared", last = "Lander"))
[1] "Hello Jared Lander"
>
>
> run.this <- function(x, func = mean)
+ {
+   do.call(func, args = list(x))
+ }
>
> run.this(1:10)
[1] 5.5
>
>
> run.this(1:10, mean)
[1] 5.5
>
>
> run.this(1:10, sum)
[1] 55
>
>
> run.this(1:10, sd)
[1] 3.02765
> |
```

# R 流程控制

1. if ... else...
2. switch
3. ifelse
4. 複合條件檢測

# R 流程控制

## 1. if ... else...

先說明最基本的程式條件檢測

( == ) -> 等於  
( < ) -> 小於  
( <= ) -> 小於等於  
( > ) -> 大於  
( >= ) -> 大於等於  
( != ) -> 不等於

再來就是回傳值的 TRUE 和  
FALSE(TRUE -> 1、 FALSE -> 0),  
在此我們利用 as.numeric() 函數。

as.numeric(TRUE)  
as.numeric(FALSE)

```
> as.numeric(TRUE)
[1] 1
>
> as.numeric(FALSE)
[1] 0
> |
```

# R 流程控制

條件判斷：

```
> 1 == 1  
[1] TRUE  
>  
> 1 < 1  
[1] FALSE  
>  
> 1 <= 1  
[1] TRUE  
>  
> 1 > 1  
[1] FALSE  
>  
> 1 >= 1  
[1] TRUE  
>  
> 1 != 1  
[1] FALSE  
> |
```

接下來將 檢測條件用入 if 敘述中，在此先建立一個名為 toCheck 的變數，其值為1。然後我們寫兩個 if 的判斷，一個條件為1、一個條件為0，成立為TRUE 則輸出 字串 " hello"。

toCheck = 1

# 兩個 if-else的判斷

```
if (toCheck == 1)  
{  
  print("hello")  
}
```

```
if (toCheck == 0)  
{  
  print("hello")  
}
```

```
> toCheck = 1  
>  
>  
> if (toCheck == 1)  
+ {  
+   print("hello")  
+ }  
[1] "hello"  
>  
>  
> if (toCheck == 0)  
+ {  
+   print("hello")  
+ }
```

# R 流程控制

if-else 判斷：

在此我們建立 check.bool 函數，將其條件設為成立輸出 hello、不成立輸出 goodbye，如要用多個條件則跟JAVA一樣使用{}。

P.S: 記得程式排版整齊 !!!

```
check.bool = function(x)
{
  if (x == 1)
  {
    print("hello")
  }
  else
  {
    print("goodbye")
  }
}
```

我們可以觀察到只要結果為 1、TRUE 就會輸出 hello，不為 1、0、FALSE 即失敗不成立則輸出 goodbye

```
check.bool(1)
check.bool(0)
check.bool("k")
check.bool(TRUE)
```

```
> check.bool = function(x)
+ {
+   if (x == 1)
+   {
+     print("hello")
+   } else
+   {
+   }
+   print("goodbye")
+ }
>
> check.bool(1)
[1] "hello"
> check.bool(0)
[1] "goodbye"
> check.bool("k")
[1] "goodbye"
> check.bool(TRUE)
[1] "hello"
> |
```

# R 流程控制

多個 if-else 的判斷，將兩 if-else 合起來。

```
check.bool = function(x)
{
  if (x == 1){
    print("hello")
  }
  else if (x == 0){
    print("goodbye")
  }
  else
  {
    print("confused")
  }
}
```

其條件為  $x = 1$  輸出 hello ,  $x = 0$  輸出 goodbye ,  
皆不是 輸出 confused 。

```
check.bool(1)
check.bool(0)
check.bool(2)
check.bool("k")
```

```
> check.bool <- function(x)
+ {
+   if (x == 1)
+   {
+     print("hello")
+   } else if (x == 0)
+   {
+     print("goodbye")
+   } else
+   {
+     print("confused")
+   }
+ }
>
> check.bool(1)
[1] "hello"
> check.bool(0)
[1] "goodbye"
> check.bool(2)
[1] "confused"
> check.bool("k")
[1] "confused"
> |
```

# R 流程控制

## 2. switch

在重複用 if-else 等多個條件時，我們可以改用 switch 來設定多個條件!!!  
在此我們自建一個名為 use.switch() 函數。

```
use.switch = function(x)
{
  switch(x,
    "a"="first",
    "b"="second",
    "z"="last",
    "c"="third",
    "other")
}
```

觀察下面的 Code 可以看到輸入指定的條件能得到特定的值 !!!

```
use.switch("a")
use.switch("b")
use.switch("c")
use.switch("d")
use.switch("e")
use.switch("z")
```

相對的我們可以給引數位置的數值，而得到特定的值。但在 use.switch(6) 因為沒有任何值，所以會是 NULL。

```
use.switch(1)
use.switch(2)
use.switch(3)
use.switch(4)
use.switch(5)
use.switch(6)
is.null(use.switch(6))
```

```
> use.switch(1)
[1] "first"
> use.switch(2)
[1] "second"
> use.switch(3)
[1] "last"
> use.switch(4)
[1] "third"
> use.switch(5)
[1] "other"
> use.switch(6)
> is.null(use.switch(6))
[1] TRUE
> |
```

# R 流程控制

## 3. ifelse

ifelse( 判斷條件 , 成立 , 不成立 )

跟前面的 if-else 效果一樣，其實就是簡略的語法，像 JAVA 也有類似的東西，而書中則說明 ifelse 跟 Excel 的 if 函數的用法類似。

在此我們用 `1==1` 與 `1==0`，做兩個範例 成立 輸出 Yes、不成立 輸出 No

```
ifelse(1 == 1, "Yes", "No")
ifelse(1 == 0, "Yes", "No")
> toTest[2] = NA
> ifelse(toTest == 1, "Yes", "No")
[1] "Yes" NA      "No"    "Yes"  "No"    "Yes"
>
> ifelse(toTest == 1, toTest * 3, toTest)
[1]  3 NA   0  3  0  3
> ifelse(toTest == 1, toTest * 3, "Zero")
[1] "3"     NA      "Zero" "3"      "Zero" "3"
> |
```

接下來在做向量化的範例，先令其名為 `toTest` 的變數，並指令一個向量，並利用 ifelse 紿入條件做判斷 !!!

`ifelse(toTest == 1, "Yes", "No")` -> 成立 "Yes"、不成立 "No"  
`ifelse(toTest == 1, toTest * 3, toTest)` -> 成立就 "原值" 乘三、不成立顯示"原值"  
`ifelse(toTest == 1, toTest * 3, "Zero")` -> 成立就 "原值" 乘三、不成立顯示 "Zero"

```
toTest = c(1, 1, 0, 1, 0, 1)
ifelse(toTest == 1, "Yes", "No")
ifelse(toTest == 1, toTest * 3, toTest)
ifelse(toTest == 1, toTest * 3, "Zero")
```

可指定向量的特定位置為 NA (遺失值)

```
toTest[2] = NA
ifelse(toTest == 1, "Yes", "No")
ifelse(toTest == 1, toTest * 3, toTest)
ifelse(toTest == 1, toTest * 3, "Zero")
```

當然 NA 就算經 ifelse 判斷，不管狀況如何成立還是不成立依舊會是 NA。

# R 流程控制

## 4. 複合條件檢測

在這裡說明的是判斷的 and、 or

and -> &&、 &

or -> ||、 |

這在其中有細微的差異

建議雙符號 &&、 || 用在單純的 if、 if-else，  
而單符號 &、 | 用在ifelse。

雙符號只是判斷兩邊有無成立，而單符號則是  
判斷兩邊的每一個有無成立

```
a = c(1, 1, 0, 1)
b = c(2, 1, 0, 1)
ifelse(a == 1 & b == 1, "Yes", "No")
ifelse(a == 0 & b == 1, "Yes", "No")
ifelse(a == 1 && b == 1, "Yes", "No")
```

```
> a = c(1, 1, 0, 1)
> b = c(2, 1, 0, 1)
>
>
> ifelse(a == 1 & b == 1, "Yes", "No")
[1] "No"  "Yes" "No"  "Yes"
>
>
> ifelse(a == 1 && b == 1, "Yes", "No")
[1] "No"
> |
```

# R 迴圈、迭代元素

1. for 迴圈

2. while 迴圈

3. 迴圈強制處理

# R 迴圈、迭代元素

## 1. for 迴圈

大家常用的即為 for，在這裡會根據索引 index 進行迭代並做運算，R 是根據使用者的 vector 向量 索引來決定迭代如何進行。

for ( 變數名稱 in 被指定的變數向量 )

for ( [A] in [B] )

A -> 變數名稱

B -> A 變數名稱中的向量

A 是 B 這個向量的指定變數名稱

範例

```
for (i in 1:10)
{
  print(i)
```

我們也可以解釋 for 建立 變數 i 由 1 加到 10 而後輸出，當然用 R 內建向量化的方式也可以做，因為 R 的基礎是由向量組成

print(1:10)

前後兩者是一樣的 !!!

```
> for (i in 1:10)
+ {
+   print(i)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
>
> print(1:10)
[1] 1 2 3 4 5 6 7 8 9 10
> |
```

# R 迴圈、迭代元素

當然了 for 迴圈中的向量也可以是非連續性的任意 vector。

```
> fruit = c("apple", "banana", "pomegranate")
> fruitLength = rep(NA, length(fruit))
> fruitLength
> names(fruitLength) = fruit
> fruitLength
```

由上我們建立了 fruit 的變數名稱並指定有三個字串的向量，再建一個名為 fruitLength 變數塞入向量，其向量中包括第一個值為 NA(遺失值)做起始後面塞前面建立的fruit 向量名稱的長度。

利用 names() 函數，替 fruitLength 指向 fruit，再次顯示只會看到原先的 fruit 與而後的 NA 值。接下來用之前用過的 nchar()函數進 for 迴圈中把名稱長度存入名為 a 的 vector 中

```
for (a in fruit)
{
  fruitLength[a] = nchar(a)
}

# 顯示長度
fruitLength
```

	apple	banana	pomegranate
	NA	NA	NA

```
>
> for (a in fruit)
+ {
+   fruitLength[a] = nchar(a)
+ }
>
> fruitLength
      apple      banana pomegranate
                 5             6            11
> |
```

# R 迴圈、迭代元素

用 nchar( ) 函數算字串長度與 names( ) 函數命名， fruitLength 與 fruitLength2 我們可以從 identical( , ) 函數進行比較，他們的結果會是一樣。

```
fruitLength2 = nchar(fruit)
names(fruitLength2) = fruit
fruitLength2
identical(fruitLength, fruitLength2)
```

```
> fruitLength2 = nchar(fruit)
>
> names(fruitLength2) = fruit
>
> fruitLength2
      apple      banana pomegranate
              5             6            11
>
> identical(fruitLength, fruitLength2)
[1] TRUE
> |
```

# R 迴圈、迭代元素

## 2. while 迴圈

我們在此設  $x = 1$  為初始值， 條件為  $x \leq 5$ ，  $x = x + 1$  符合條件時停止。

```
x = 1
```

```
while (x <= 5)
{
  print(x)
  x = x + 1
}
```

```
> x = 1
>
> while (x <= 5)
+ {
+   print(x)
+   x = x + 1
+
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
> |
```

# R 迴圈、迭代元素

## 3. 迴圈強制處理

在這裡分為 next 和 break，在迴圈中可以加入的條件，他們二者的差異在於 next 執行的話，只會停止那部分的條件，而 break 執行的話會整個停止!!!

```
# next  
  
for (i in 1:10)  
{  
  if (i == 3)  
  {  
    next  
  }  
  print(i)  
}
```

```
> for (i in 1:10)  
+ {  
+   if (i == 3)  
+   {  
+     next  
+   }  
+   print(i)  
+ }  
[1] 1  
[1] 2  
[1] 4  
[1] 5  
[1] 6  
[1] 7  
[1] 8  
[1] 9  
[1] 10  
> |
```

```
# break  
  
for (i in 1:10)  
{  
  if (i == 4)  
  {  
    break  
  }  
  print(i)  
}
```

```
> for (i in 1:10)  
+ {  
+   if (i == 4)  
+   {  
+     break  
+   }  
+   print(i)  
+ }  
[1] 1  
[1] 2  
[1] 3  
> |
```

# R 使用函數的注意事項

```
test = function(x){  
  a = x^5  
  print(a)  
}
```

```
test(2)
```

```
test
```

```
> test = function(x) {  
+   a = x^5  
+   print(a)  
+ }  
>  
> test(2)  
[1] 10  
>  
> ls()  
[1] "test"  
>  
> test  
function(x) {  
  a = x^5  
  print(a)  
}  
>
```

# R Function 全域變數

Function 內的變數執行完後會收回  
去(區域變數)，而直接在 R 建立的  
變數是全域變數，在function 下也  
可以使用。

```
a <- 10
```

```
assign("a", 10, env = .GlobalEnv)
```

在這裡的 a 是變數名稱 10 則是值。

```
TCH = "Hello Function!"
```

```
ktestgv = function(k){
```

```
  a <- 10:1
```

```
  assign("b", 1:10, env = .GlobalEnv)
```

```
  print(k)
```

```
}
```

```
ls()
```

```
ktestgv(TCH)
```

```
ls()
```

```
a
```

```
b
```

```
> TCH = "Hello Function!"  
> ktestgv = function(k) {  
+   a <- 10:1  
+   assign("b", 1:10, env = .GlobalEnv)  
+   print(k)  
+ }  
> ls()  
[1] "ktestgv" "TCH"  
> ktestgv(TCH)  
[1] "Hello Function!"  
> ls()  
[1] "a"          "b"          "ktestgv" "TCH"  
> a  
[1] 10 9 8 7 6 5 4 3 2 1  
> b  
[1] 1 2 3 4 5 6 7 8 9 10  
>
```